超强抗干扰，超级加密

车规 AEC-Q100 Grade1 MCU 设计公司，高稳定，低功耗

技术网站论坛：www.STCAI.com 分销商电话：0513-5501 2928，89896509

深圳国芯人工智能有限公司

车规 AEC-Q100 Grade1 MCU 设计公司

Ai8051U

《单片机原理及应用》

《实验指导书》

Ai8051U 系列是 32 位 8051
也是优秀的 16 位机
更是兼容 8 位机的最强悍的 1 位机

- 10 个 32 位累加器
- 16 个 16 位累加器
- 16 个 8 位累加器
- 32 位加减指令
- 16 位乘除指令
- 32 位乘除运算（MDU32）
- 单精度浮点运算+三角/反三角函数（TFPU）
- 32 位算术比较指令
- 所有的 SFR（80H~FFH）均支持位寻址
- ebdata（20H~7FH）全部支持位寻址
- 单时钟 32/16/8 位数据读写（edata）
- 单时钟端口读写
- 堆栈理论深度可达 64K（实际取决于 edata）
- 编译器：Keil C51 / 8 位，Keil C251 / 32 位
- 已移植的 RTOS：uC/OS-II，FreeRTOS
- 已移植的文件系统：FATFS，Petit FATFS
- 已移植的 GUI：uGFX，U8g2
- 工作温度：－40℃ ～ ＋125℃

扫码去微信小商城

技术支持网站：www.STCAI.com
官方技术论坛：www.STCAIMCU.com
资料更新日期：**2025/7/14**
(本文档可直接添加备注和标记)

目录

1 单片机基础概述 2
1.1 什么是单片机/微控制器 2
1.1.1 经典单片机 89C52RC/89C58RD+系列组成框图 2
1.1.2 经典单片机 12C5A60S2 组成框图 4
1.1.3 Ai8051U 内部结构图 5
1.1.3.1 Ai8051U 8-Bit 内部结构图，内部总线工作在 8 位宽度 5
1.1.3.2 Ai8051U 32-Bit 内部结构图，内部总线工作在 32 位宽度 6
1.1.4 Intel 80251 内部结构图 8
1.2 数制与编码 9
1.2.1 数制转换 10
1.2.2 原码、反码及补码 13
1.2.3 常用编码 13
1.3 几种常用的逻辑运算及其图形符号 14
2 集成开发环境的使用与 ISP 下载软件的介绍 17
2.1 下载 Keil 集成开发环境 17
2.2 安装 Keil 集成开发环境 19
2.2.1 安装 Keil C251 集成开发环境 19
2.2.2 安装 Keil C51 集成开发环境 22
2.2.3 Keil 的 C51、C251 和 MDK 可安装在同一台电脑同一个目录 25
2.2.4 从哪购买 KEIL，获取无限制版 KEIL 26
2.3 安装 AIapp-ISP 下载/编程/烧录 工具,含强大的辅助开发工具 27
2.3.1 安装 AIapp-ISP 下载/编程/烧录 工具软件，取代 STC-ISP 27
2.3.2 STCAI 单片机上电工作过程 29
2.3.3 Ai8051U 的 ISP 下载流程图（硬件 USB+串口模式） 30
2.3.4 Ai8051U 的 USB 直接下载电路及 ISP 下载操作步骤 32
2.4 添加型号和头文件到 Keil 34
2.5 Ai8051U 的 8 位/32 位【头文件，编译器，ISP 烧录时的设置】说明 36
2.6 新建一个 32 位 8051 项目，在 Keil 中 38
2.6.1 准备工作，见前面几小节 38
2.6.2 Keil 新建一个 32 位 8051 项目 39
2.6.2.1 新建工程 39
1）在上方的菜单栏 Project 标签中，找到 New uVision Project 选项，开始新建工程。 39
2）将文件存入刚刚建好的文件夹 E:\project1 中 39
3）选择单片机 39
4）选择完单片机后，要给工程项目增加一个主程序文件 40
5）给工程项目增加一个主程序文件。 41
2.6.2.2 32 位 8051 工程项目的初步简单设置，必须按此流程过一遍 43
1）Target 选项卡--Memory Model 选择 XSmall 模式、勾选 4 字节对齐 43
2）Output 选项卡--勾选 Create HEX File，选择 HEX-80 43
3）L251 Misc 选项卡--在 Misc controls 输入框输入‘REMOVEUNUSED’ 44
4）Debug 选项卡--使用 STCAI 工具进行硬件仿真设置 44

2.7 堆栈不用用户设置@Ai8051U-32Bit 46
2.8 新建一个 8 位 8051 项目，在 Keil 中 47
2.8.1 准备工作，见前面几小节 47
2.8.2 Keil 新建一个 8 位 8051 项目 48
2.8.2.1 新建工程 48
1）在上方的菜单栏 Project 标签中，找到 New uVision Project 选项，开始新建工程。 .. 48
2）将文件存入刚刚建好的文件夹 E:\project1 中 48
3）选择单片机 48
4）选择完单片机后，要给工程项目增加一个主程序文件 49
5）给工程项目增加一个主程序文件。 50
2.8.2.2 8 位 8051 工程项目的各种基础选项设置。 52
1）Device 选项卡--勾选 Use Extended Linker（LX51）instead of BL51 可支持 REMOVEUNUSED 参数 52
2）Output 选项卡--勾选 Create HEX File，选择 HEX-80 52
3）LX51 Misc 选项卡--在 Misc controls 输入框输入‘REMOVEUNUSED’ 53
4）Debug 选项卡-- Ai8051U-8bit 工作在 8 位模式，暂不支持任何硬件仿真，无需设置 53
2.9 在 Keil uVision5 的编辑器中，输入中文出现乱码的解决办法 54
2.10 Keil 软件中因含 0xFD 编码的汉字而出现乱码的问题 55
2.11 C 语言中 printf()函数打印输出数据时，常用的各种输出格式 56
2.12 实验一：printf_usb("Hello World !\r\n")，第一个完整的 C 语言程序 57
2.12.1 printf( )，printf_usb( )实验程序代码 57
2.12.2 准备工作，如已准备，跳过此节 58
2.12.3 认识 Keil 软件的 Build Toolbar（Build 工具栏） 61
2.12.4 下载用户程序到实验箱 62
2.12.4.1 连接实验箱和电脑 62
2.12.4.2 硬件 USB 下载用户程序步骤 63
2.12.4.3 下载用户程序文件并查看实验现象 65
2.12.5 使用【AiCube】工具创建 -- printf_usb("Hello World !\r\n")，第一个完整的 C 语言程序 68
a) 安装 AIapp-ISP 下载/编程/烧录/软件 68
b) 打开【AiCube】 69
c) 使用【AiCube】工具创建项目 70
d) 增加实验项目代码 73
e) 使用 AIapp-ISP 软件系统和 Ai8051U 实验箱查验实验效果 73
2.12.6 USB 不停电下载，ISP 软件如何设置 75
2.12.6.1 手动点击【下载/编程】按钮，ISP 软件自动重新打开文件，不停电下载 75
2.12.6.2 编译器重新编译后，ISP 软件会自动重新调入文件，进行不停电下载 76
2.13 实验二：查询方式，查询到电脑命令后，printf_usb("Hello World !\r\n")或其他 77
2.13.1 printf( )，printf_usb( )实验程序代码 77
2.13.2 准备工作，如已准备，跳过此节 78
2.13.3 下载用户程序到实验箱 81
2.13.3.1 连接实验箱和电脑 81
2.13.3.2 硬件 USB 下载用户程序步骤 82
2.13.3.3 下载用户程序文件 84

2.13.4 查看实验效果 86
2.13.5 使用【AiCube】工具创建“查询方式，查询到电脑命令后，printf_usb("Hello World !\r\n")或其他” 87
a) 安装 AIapp-ISP 下载/编程/烧录/软件 87
b) 打开【AiCube】 88
c) 使用【AiCube】创建项目 89
d) 增加实验项目代码 92
e) 使用 AIapp-ISP 软件系统和 Ai8051U 实验箱查验实验效果 93
2.13.6 USB 不停电下载，ISP 软件如何设置 95
2.13.6.1 手动点击【下载/编程】按钮，ISP 软件自动重新打开文件，不停电下载 95
2.13.6.2 编译器重新编译后，ISP 软件会自动重新调入文件，进行不停电下载 96
2.14 实验三：中断方式，单片机 USB 中断接收服务函数收到命令后，printf_usb("Hello World !\r\n")或其他 97
2.14.1 printf( )，printf_usb( )实验程序代码 97
2.14.2 查看实验效果 98
2.14.3 使用【AiCube】工具创建“中断方式，单片机 USB 中断接收服务函数收到命令后，printf_usb("Hello World !\r\n")或其他” 99
a) 安装 AIapp-ISP 下载/编程/烧录/软件 99
b) 打开【AiCube】助手 100
c) 使用【AiCube】创建项目 101
d) 增加实验项目代码 104
e) 使用 AIapp-ISP 软件系统和 Ai8051U 实验箱查验实验效果 105
2.14.4 USB 不停电下载，ISP 软件如何设置 107
2.14.4.1 手动点击【下载/编程】按钮，ISP 软件自动重新打开文件，不停电下载 107
2.14.4.2 编译器重新编译后，ISP 软件会自动重新调入文件，进行不停电下载 108
2.15 点灯实验一：点亮 LED 灯，继续完成前面的工程项目，有视频讲解 109
2.15.1 原理介绍，在 Ai8051U 实验箱中有如下的一排 LED 指示灯 109
2.15.2 认识 Keil 软件的 Build Toolbar（Build 工具栏） 110
2.15.3 代码部分 111
2.15.4 下载用户程序到实验并查看实验现象，实物图 113
2.15.4.1 连接实验箱和电脑 113
2.15.4.2 硬件 USB 下载用户程序步骤 114
2.15.4.3 下载用户程序文件并观察实验现象 116
2.15.5 使用【AiCube】项目创建助手工具创建并实现点亮 LED 灯 119
a) 安装【AIapp-ISP】下载/编程/烧录/软件 119
b) 打开【AiCube】助手 120
c) 使用【AiCube】助手创建项目 120
d) 增加实验项目代码 123
e) 使用 Ai8051U 实验箱查验实验效果 124
2.15.6 USB 不停电下载，ISP 软件如何设置 125
2.15.6.1 手动点击【下载/编程】按钮，ISP 软件自动重新打开文件，不停电下载 125
2.15.6.2 编译器重新编译后，ISP 软件会自动重新调入文件，进行不停电下载 126
2.16 点灯实验二：让 LED 灯闪烁起来，有视频讲解 127
2.16.1 原理介绍，在 Ai8051U 实验箱中有如下的一排 LED 指示灯 127

2.16.2 代码部分 128
2.16.3 下载用户程序到实验箱并查看实验现象，实物图 132
2.16.3.1 连接实验箱和电脑 132
2.16.3.2 硬件 USB 下载用户程序步骤 133
2.16.3.3 下载用户程序文件并观察实验效果 135
2.16.4 使用【AiCube】项目创建助手工具创建并实现 LED 灯闪烁 137
a) 安装【AIapp-ISP】下载/编程/烧录/软件 137
b) 打开【AiCube】助手 138
c) 使用【AiCube】助手创建项目 138
d) 增加实验项目代码 142
e) 使用 Ai8051U 实验箱查验实验效果 143
2.16.5 USB 不停电下载，ISP 软件如何设置 144
2.16.5.1 手动点击【下载/编程】按钮，ISP 软件自动重新打开文件，不停电下载 144
2.16.5.2 编译器重新编译后，ISP 软件会自动重新调入文件，进行不停电下载 145
2.17 点灯实验三：通过 USB-CDC 方式控制 Ai8051U 进行 LED 点灯 146
2.17.1 原理介绍，在 Ai8051U 实验箱中有如下的一排 LED 指示灯 146
2.17.2 准备工作 147
2.17.3 代码部分 149
2.17.4 下载用户程序到实验箱并查看实验现象，实物图 153
2.17.4.1 连接到实验箱 153
2.17.4.2 硬件 USB 下载用户程序步骤 154
2.17.4.3 下载用户程序文件并观察实验现象 156
a) 点击 CDC/HID-串口助手选项卡中的“点亮 P00-P03 灯”按钮 159
b) 点击 CDC/HID-串口助手选项卡中的“点亮 P04-P07 灯”按钮： 160
2.17.5 使用【AiCube】项目创建助手工具创建通过 USB-CDC 方式控制 Ai8051U 进行 LED 点灯 161
a) 安装 AIapp-ISP 下载/编程/烧录/软件 161
b) 打开【AiCube】助手 162
c) 使用【AiCube】助手创建项目 162
d) 增加实验项目代码 165
e) 使用 AIapp-ISP 软件系统和 Ai8051U 实验箱查验实验效果 166
2.17.6 USB 不停电下载，ISP 软件如何设置 169
2.17.6.1 手动点击【下载/编程】按钮，ISP 软件自动重新打开文件，不停电下载 169
2.17.6.2 编译器重新编译后，ISP 软件会自动重新调入文件，进行不停电下载 170
2.18 点灯实验四：使用 Ai8051U 内置 USB 直接仿真流水灯 171
2.18.1 原理介绍，在 Ai8051U 实验箱中有如下的一排 LED 指示灯 171
2.18.2 实验程序 C 语言代码 172
2.18.3 下载用户程序到实验箱，实物图 178
2.18.3.1 连接到实验箱 178
2.18.3.2 硬件 USB 下载用户程序步骤 179
2.18.3.3 制作仿真芯片 181
2.18.4 Keil 仿真并观察实验现象 182
2.18.4.1 认识 Keil 软件的 Debug Toolbar（Debug 工具栏） 183
2.18.4.2 使用 Run 按钮，观察全速运行效果的实验现象 185

2.18.4.3 观察外设寄存器数据与实验现象 185
2.18.5 实验程序汇编语言代码 191
2.18.6 使用【AiCube】项目创建助手工具创建使用 Ai8051U 内置 USB 仿真流水灯 193
a) 安装 AIapp-ISP 下载/编程/烧录/软件 193
b) 打开【AiCube】助手 194
c) 使用【AiCube】助手创建项目 194
d) 增加实验项目代码 197
e) 使用 AIapp-ISP 软件设置仿真芯片并连接 Ai8051U 实验箱验证实验效果 198
2.19 扩展 Keil 对中断号数量的支持，中断号大于 31 编译出错的处理 199
2.19.1 如何下载 Keil 中断号拓展工具，及安装 199
2.19.2 BL51/LX51 链接器和扩展中断号工具，REMOVEUNUSED 说明 202
2.19.3 如不使用扩展中断号工具，则需借用保留中断号进行中转 203
2.20 单片机程序中头文件的使用方法 212
2.21 超过 64K 代码的项目（Source 模式）设置为 Huge，不超过选择 Large 214
2.21.1 设置项目路径和项目名称 214
2.21.2 选择目标单片机型号 215
2.21.3 添加源代码文件到项目 216
2.21.4 设置 1（“CPU Mode”选择 Source 模式） 217
2.21.5 设置 2（“Memory Model”选择 XSmall 模式） 218
2.21.6 设置 3，Code Rom Size：超 64K 选 Huge，小于 64K 选 Large 221
2.21.7 设置 4（超 64K 代码的相关设置） 222
2.21.8 设置 5（HEX 文件格式设置）超 64K 选 Hex-386，小于 64K 可选 Hex-80 223
2.22 嵌入汇编代码在 C 语言中，Keil C251 编译器 224
2.23 嵌入汇编代码在 C 语言中，Keil C51 编译器 226
2.24 汇编代码演示，在 Keil 中编写 Ai8051U 汇编代码 228
2.24.1 代码大小在 64K 以内的汇编程序编写方法 228
2.24.2 代码大小超过 64K 的汇编程序编写方法 229
2.25 对变量、常量、表格数据、函数指定绝对地址，在 C251 中 231
2.25.1 变量如何指定绝对地址，Keil C251 中 231
2.25.2 常量如何指定绝对地址，Keil C251 中 232
2.25.3 表格数据如何指定绝对地址，Keil C251 中 233
2.25.4 函数如何指定绝对地址，Keil C251 中 234
2.26 获取帮助的简单方法，Keil 软件中 236
2.27 建立多文件项目的方法，在 Keil 中 239
2.28 外部变量和外部函数的声明和使用，在多文件项目中 241
2.29 变量的声明要求在可执行语句之前，C89 标准，KEIL C 语言 243
2.30 程序超 64K 时如何设置保留 EEPROM 空间，暂时用不到 244
2.31 用户程序复位到系统区进行 USB 模式 ISP 下载的方法（不停电） 246
2.32 ISP 下载流程及典型应用线路图，USB-TypeC 接口 249
2.32.1 ISP 下载流程图（硬件/软件模拟 USB+串口模式） 249
2.32.2 ISP 下载流程图（串口下载模式） 251
2.32.3 USB-TypeC / TypeA / 普通插座都支持 USB 直接下载，5V 253
2.32.4 USB-TypeC / TypeA / 普通插座都支持 USB 直接下载，3.3V 255
2.32.5 使用 USB 转双串口/TTL 下载（有外部晶振） 256

2.32.6 使用 USB 转双串口/TTL 下载（无外部晶振）.....257
2.32.7 使用 USB 转双串口/TTL 下载（自动停电/上电）.....259
2.32.8 使用 USB 转双串口/RS485 下载（5.0V）.....260
2.32.9 使用 USB 转双串口/RS485 下载（3.3V）.....260
2.32.10 使用 USB 转双串口/RS232 下载（5.0V）.....262
2.32.11 使用 USB 转双串口/RS232 下载（3.3V）.....262
2.32.12 使用 PL2303-GL 下载.....263
2.33 ISP 下载软件高级应用,下载需口令,控制下载次数,程序加密后传输,发布项目程序/远程现场升级 App 发布等.....264
2.33.1 发布项目程序/远程现场升级 App 发布.....264
2.33.2 程序加密后传输（防烧录时串口分析出程序）.....268
2.33.3 发布项目程序/远程现场升级 App 发布+程序加密后传输结合使用.....272
2.33.4 用户自定义下载（实现不停电下载）.....273
2.33.5 如何简单的控制下载次数，通过 ID 号来限制实际可以下载的 MCU 数量.....277
2.33.6 下载需口令高级功能.....283
2.34 用【USB 转串口的下载器】下载，不要勾选【下次使用 HID 接口进行 ISP 下载】.....286
3 USB 库函数使用说明，简单的 USB-CDC 虚拟串口通信，USB 不停电下载.....287
3.1 强大而简单的 USB-CDC 虚拟串口概述.....287
3.2 使用 C#/C++/VB 开发 USB-CDC 虚拟串口的应用程序与普通串口一样吗？.....288
3.3 USB 库函数简介.....289
3.4 下载 USB 库函数.....289
3.5 USB 库函数的详细说明.....291
3.6 建立基于 USB 库函数的 Keil 项目.....295
3.7 USB-CDC 虚拟串口实现不停电自动 ISP 下载，及 ISP 软件如何设置.....304
3.8 关于“下次使用 HID 接口进行 ISP 下载”的选项说明.....305
3.9 USB 范例程序，查询方式和中断方式各举一例.....306
3.9.1 范例程序 1（查询方式调用 CDC/HID 库函数）.....306
3.9.2 范例程序 2（中断方式调用 CDC/HID 库函数）.....308
4 Ai8051U 系列选型简介、特性/价格/管脚图/ISP 下载/编程线路图.....309
4.1 Ai8051U-LQFP48 总体介绍，USB 下载，烧录/仿真 线路图.....309
4.1.1 特性及价格.....309
4.1.2 Ai8051U 系列内部结构图.....312
4.1.2.1 Ai8051U-32Bit 内部结构图.....312
4.1.2.2 Ai8051U-8Bit 内部结构图.....313
4.1.3 LQFP48/QFN48 管脚图，5V 线路，USB-ISP 下载/烧录/仿真.....314
4.1.4 LQFP48/QFN48 管脚图，3.3V 线路一，USB-ISP 下载/仿真.....321
4.1.5 LQFP48/QFN48 管脚图，3.3V 线路二，USB-ISP 下载/仿真.....323
4.1.6 如果使用 3.3V-LDO 供电，建议用有电源关断功能的 6211/6231.....325
4.1.7 管脚说明.....326
4.1.8 USB-Link1D 对 Ai8051U 自动停电/上电烧录，串口仿真+串口通讯.....333
4.1.9 【一箭双雕之 USB 转双串口】工具进行烧录，串口仿真+串口通讯.....337
4.1.10 USB 转双串口芯片全自动停电/上电烧录，串口仿真+串口通讯，5V.....338
4.1.11 USB 转双串口芯片全自动烧录，串口仿真+串口通讯，3.3V 原理图.....339

4.1.12 USB 转双串口芯片进行自动烧录/仿真+串口通讯，5V/3.3V 跳线选择 340
4.1.13 通用 USB 转串口芯片全自动停电/上电烧录，串口仿真，5V 原理图.....341
4.1.14 通用 USB 转串口芯片全自动停电/上电烧录，串口仿真，3.3V 原理图.....342
4.1.15 USB 转串口芯片全自动停电/上电烧录/仿真/通信，5V/3.3V 跳线选择 343
4.1.16 USB 转串口芯片进行烧录/串口仿真，手动停电/上电，5V/3.3V 原理图.....345
4.1.17 USB 转串口芯片进行烧录，串口仿真，手动停电/上电，3.3V 原理图.....347
4.1.18 USB-Link1D 支持 脱机下载 说明.....348
4.1.19 USB-Link1D 支持 脱机下载，如何免烧录环节.....349
4.1.20 USB-Writer1A 编程器/烧录器 支持 插在 锁紧座上 烧录.....351
4.1.21 USB-Writer1A 支持 自动烧录机，通信协议和接口 352
4.2 LQFP44 管脚图，USB-ISP 下载，各种 烧录/仿真 线路图.....354
4.2.1 LQFP44 管脚图，5V 线路，USB-ISP 下载/烧录/仿真.....354
4.2.2 LQFP44 管脚图，3.3V 线路一，USB-ISP 下载/仿真.....361
4.2.3 LQFP44 管脚图，3.3V 线路二，USB-ISP 下载/仿真.....363
4.2.4 如果使用 3.3V-LDO 供电，建议用有电源关断功能的 6211/6231 365
4.2.5 管脚说明.....366
4.2.6 USB-Link1D 对 Ai8051U 自动停电/上电烧录，串口仿真+串口通讯.....373
4.3 PDIP40 管脚图，USB-ISP 下载，各种 烧录/仿真 线路图.....379
4.3.1 PDIP40 管脚图，5V 线路，USB-ISP 下载/烧录/仿真.....379
4.3.2 PDIP40 管脚图，3.3V 线路一，USB-ISP 下载/仿真.....386
4.3.3 PDIP40 管脚图，3.3V 线路二，USB-ISP 下载/仿真.....388
4.3.4 如果使用 3.3V-LDO 供电，建议用有电源关断功能的 6211/6231 390
4.3.5 管脚说明.....391
4.3.6 USB-Link1D 对 Ai8051U 自动停电/上电烧录，串口仿真+串口通讯.....398
4.3.7 USB 转双串口芯片进行自动烧录/仿真+串口通讯，5V/3.3V 跳线选择 403
4.3.8 USB 转串口芯片进行全自动烧录/仿真/通信，5V/3.3V 跳线选择.....404
4.3.9 USB 转串口芯片进行烧录/串口仿真，手动停电/上电，5V/3.3V 原理图.....407
4.4 USB-Link1D 工具强大的配套多种接口豪华线，使用注意事项.....410
4.4.1 工具接口说明.....410
4.4.2 附送的各种强大的人性化配线图片及使用说明 411
4.4.3 USB-Link1D 实际应用.....416
4.4.4 USB-Link1D 插上电脑并正常识别到后的显示.....423
4.5 如电脑端新版本 ISP 软件提示 USB-Link1D 工具固件需升级 424
4.6 重新制作 USB-Link1D 主控芯片，主动升级或遇到异常时 425
4.7 ISP 下载相关硬件选项的说明 428
4.8 通用 USB 转双串口芯片：USB-2UART, TSSOP20/SOP16.....429
4.8.1 USB 转双串口芯片 USB-2UART-45I-TSSOP20，自动停电上电（MOS 管）.....429
4.8.2 USB 转双串口芯片 USB-2UART-45I-TSSOP20，自动停电上电（三极管）.....430
4.8.3 USB 转双串口芯片 USB-2UART-45I-TSSOP20，手动停电上电 431
4.8.4 USB 转双串口芯片 USB-2UART-45I- SOP16，自动停电上电（MOS 管）.....432
4.8.5 USB 转双串口芯片 USB-2UART-45I- SOP16，自动停电上电（三极管）.....433
4.8.6 USB 转双串口芯片 USB-2UART-45I-SOP16，手动停电上电.....434
5 Ai8052U 系列选型简介、特性、价格、管脚图.....435
5.1 Ai8052U-LQFP100 总体介绍，USB 下载，烧录/仿真 线路图.....435

5.1.1 特性及价格 435
5.1.2 管脚图，最小系统（LQFP100/QFN100） 439
5.1.3 多功能管脚切换 440
5.1.4 管脚图，最小系统（LQFP64/QFN64） 441
5.1.5 管脚图，最小系统（LQFP48） 442
5.1.6 管脚图，最小系统（LQFP44） 443
5.2 两组独立 DAC、4 组运放、4 组比较器结构及应用，整理中 445
5.2.1 反相放大 450
5.2.2 同相/正相放大 454
5.2.3 Input BUFFER Mode（输入缓冲模式） 456
5.2.4 DAC BUFFER Mode（DAC 缓冲模式） 457
5.2.5 常见的运放应用电路，不是本器件的内部特定电路 Other mode:(建议使用 GP MODE 来外接电路) 458
6 功能脚切换 461
6.1 功能脚切换相关寄存器 461
6.1.1 串口 1/SPI，功能脚切换控制（P_SW1） 462
6.1.2 串口 2/3/4/I^2C/比较器输出，功能脚切换控制（P_SW2） 463
6.1.3 I2S/串口 1 的 SPI/串口 2 的 SPI，功能脚切换控制（P_SW3） 464
6.1.4 QSPI，功能脚切换控制（P_SW4） 464
6.1.5 SPI 的 MOSI 和 MISO 脚交换控制（HSSPI_CFG2） 465
6.1.6 主时钟输出脚，输出选择寄存器（MCLKOCR） 466
6.1.7 PCA/CCP 输出/捕获，功能脚切换控制（CMOD） 466
6.1.8 高级 PWM 输出脚/外部捕获脚，切换控制（PWMn_PS） 466
6.1.9 高级 PWM 外部触发脚/刹车脚，切换控制（PWMx_ETRPS） 469
6.1.10 ADC 外部触发脚，切换控制（ADCEXCFG） 469
6.2 范例程序 470
6.2.1 串口 1 切换 470
6.2.2 串口 2 切换 470
6.2.3 串口 3 切换 471
6.2.4 串口 4 切换 472
6.2.5 SPI 切换 472
6.2.6 I2C 切换 473
6.2.7 比较器输出切换 474
6.2.8 主时钟输出切换 474
7 封装尺寸图 476
7.1 SOP8 封装尺寸图 476
7.2 DFN8 封装尺寸图（3mm*3mm） 477
7.3 SOP16 封装尺寸图 478
7.4 SOP20 封装尺寸图 479
7.5 TSSOP20 封装尺寸图 480
7.6 QFN20 封装尺寸图（3mm*3mm） 481
7.7 SOP28 封装尺寸图 482
7.8 TSSOP28 封装尺寸图 483
7.9 LQFP32 封装尺寸图（9mm*9mm） 484

7.10 QFN32 封装尺寸图（4mm*4mm）.....485
7.11 PDIP40 封装尺寸图.....486
7.12 LQFP44/QFP44 封装尺寸图（12mm*12mm）.....487
7.13 LQFP48/QFP48 封装尺寸图（9mm*9mm）.....488
7.14 QFN48 封装尺寸图（6mm*6mm）.....489
7.15 LQFP64 封装尺寸图（12mm*12mm）.....490
7.16 QFN64 封装尺寸图（8mm*8mm）.....491
7.17 USB-TypeC 封装尺寸图，原理图.....492
7.18 单片机命名规则.....493
8 时钟管理，芯片上电工作过程，外部晶振抗干扰.....494
8.1 系统时钟控制.....494
8.2 芯片上电工作过程：.....496
8.3 时钟配置寄存器，外设，高频 PLL 时钟及分频，外部晶振抗干扰.....497
8.3.1 USB 时钟控制寄存器（USBCLK）.....498
8.3.2 系统时钟选择寄存器（CLKSEL）.....499
8.3.3 时钟分频寄存器（CLKDIV）.....499
8.3.4 内部高速高精度 IRC 控制寄存器（HIRCCR）.....500
8.3.5 外部振荡器控制寄存器（XOSCCR），外部晶振抗干扰.....500
8.3.6 内部低速 IRC 控制寄存器（IRC32KCR）.....501
8.3.7 主时钟输出控制寄存器（MCLKOCR）.....502
8.3.8 内部高速 IRC 时钟恢复振荡到稳定需要等待的时钟数控制寄存器（IRCDB）.....503
8.3.9 内部 48MHz 高速 IRC 控制寄存器（IRC48MCR）.....504
8.3.10 外部 32K 振荡器控制寄存器（X32KCR）.....504
8.3.11 高速时钟分频寄存器（HSCLKDIV）.....504
8.3.12 SPI 时钟分频寄存器（SPI_CLKDIV）.....505
8.3.13 PWMA 时钟分频寄存器（PWMA_CLKDIV）.....505
8.3.14 PWMB 时钟分频寄存器（PWMB_CLKDIV）.....505
8.3.15 TFPU 时钟分频寄存器（TFPU_CLKDIV）.....506
8.3.16 I2S 时钟分频寄存器（I2S_CLKDIV）.....506
8.3.17 I2C 总线速度控制.....506
8.4 Ai8051U 系列内部高速 IRC 时钟频率调整.....507
8.4.1 IRC 频段选择寄存器（IRCBAND）.....507
8.4.2 内部 IRC 频率调整寄存器（IRTRIM）.....508
8.4.3 时钟分频寄存器（CLKDIV）.....508
8.5 外部晶振及外部时钟电路.....509
8.5.1 外部晶振输入电路.....509
8.5.2 外部时钟输入电路（P5.6 为高阻输入模式，可当输入口使用）.....509
8.6 范例程序.....510
8.6.1 选择内部高速 IRC（HIRC）作为系统时钟源.....510
8.6.2 在用户程序区用户如何动态选择不同的系统时钟频率.....510
8.6.3 选择内部 IRC（IRC32K）作为系统时钟源.....513
8.6.4 选择内部 48M 的 IRC（IRC48M）作为系统时钟源.....513
8.6.5 选择外部高速晶振（XOSC）作为系统时钟源.....514
8.6.6 选择外部低速晶振（X32K）作为系统时钟源.....515

8.6.7 选择内部 PLL 作为系统时钟源 515
8.6.8 选择主时钟（MCLK）作为高速外设时钟源 516
8.6.9 选择内部 PLL 时钟作为高速外设时钟源 517
8.6.10 选择系统时钟（SYSCLK）作为 USB 时钟源 518
8.6.11 选择内部 PLL 时钟作为 USB 时钟源 519
8.6.12 选择内部 USB 专用 48M 的 IRC 作为 USB 时钟源 520
8.6.13 主时钟分频输出 520
9 校准内部 IRC 时钟，利用外部 32768Hz 晶振，降低电磁辐射 522
9.1 相关寄存器 522
9.1.1 CRE 控制寄存器（CRECR） 522
9.1.2 CRE 校准计数值寄存器（CRECNT） 523
9.1.3 CRE 校准误差值寄存器（CRERES） 523
9.2 内部高速 IRC 的 4 个频段的中心频率，供参考 524
9.3 范例程序 525
9.3.1 自动校准内部高速 IRC（HIRC） 525
10 复位、看门狗、掉电唤醒专用定时器与电源管理 527
10.1 系统复位 527
10.1.1 看门狗控制寄存器（WDT_CONTR） 528
10.1.2 IAP 控制寄存器（IAP_CONTR） 529
10.1.3 复位配置寄存器（RSTCFG） 529
10.1.4 复位标志寄存器（RSTFLAG） 530
10.1.5 复位控制寄存器（RSTCRx） 532
10.1.6 内置专业级复位电路，不需传统的阻容式上电延时电路 533
10.1.7 外部低电平按键复位电路 533
10.1.8 传统 8051 高电平上电复位参考电路 533
10.2 主时钟停振/省电模式，系统电源管理 534
10.2.1 电源控制寄存器（PCON） 534
10.2.2 内部高速 IRC 时钟恢复振荡到稳定需要等待的时钟数控制寄存器（IRCDB） 535
10.3 可以唤醒省电模式/主时钟停振模式的中断资源 536
10.4 主时钟停振/省电模式，I/O 口如何设置才省电 537
10.5 掉电唤醒定时器 538
10.5.1 掉电唤醒定时器计数寄存器（WKTCL，WKTCH） 538
10.6 省电模式，I/O 口如何设置才省电 540
10.7 范例程序 541
10.7.1 看门狗定时器应用 541
10.7.2 软复位实现自定义下载 541
10.7.3 低压检测 542
10.7.4 省电模式 543
10.7.5 使用 INT0/INT1/INT2/INT3/INT4 管脚中断唤醒省电模式 544
10.7.6 使用 T0/T1/T2/T3/T4 管脚中断唤醒省电模式 546
10.7.7 使用 RxD/RxD2/RxD3/RxD4 管脚中断唤醒省电模式 548
10.7.8 使用 I2C 的 SDA 脚唤醒 MCU 省电模式 549
10.7.9 使用掉电唤醒定时器唤醒省电模式 550
10.7.10 LVD 中断唤醒省电模式，建议配合使用掉电唤醒定时器 551

10.7.11 比较器中断唤醒省电模式，建议配合使用掉电唤醒定时器 553
10.7.12 使用 LVD 功能检测工作电压（电池电压） 554
10.7.13 使用 RTC 实时时钟中断唤醒省电模式 556
11 第三方对 Ai8051U 下载程序范例及用户程序如何选择内部不同时钟频率 558
11.1 电源控制参考电路 558
11.2 通信协议流程图 559
11.3 参考代码（C 语言） 562
11.4 如何在用户程序中设置程序运行时的工作频率 569
11.4.1 在用户程序区用户如何动态选择不同的系统时钟频率 569
11.5 如何在用户程序中设置复位脚、低压检测门槛电压等参数 571
12 存储器（32/16/8 位访问），在哪存储重要参数，及全球唯一 ID 号 CHIPID 572
12.1 程序存储器 574
12.1.1 程序读取等待控制寄存器（WTST） 575
12.2 数据存储器（32 位访问，16 位访问，8 位访问） 576
12.2.1 Keil 选项 Memory Model 设置 577
12.2.2 内部 edata-RAM（C 语言声明关键字为 edata） 579
12.2.3 程序状态寄存器（PSW） 580
12.2.4 程序状态寄存器 1（PSW1） 580
12.2.5 内部 xdata-RAM（C 语言声明关键字为 xdata） 581
12.2.6 辅助寄存器（AUXR） 581
12.2.7 片外用户自己扩展的外部 XRAM/xdata 581
12.2.8 片外用户扩展 RAM 的总线速度控制寄存器（BUS_SPEED） 582
12.2.9 片内 MCU 扩展 RAM 数据总线时钟控制寄存器（CKCON） 583
12.2.10 片外扩展 RAM 地址总线扩展寄存器（MXAX） 583
12.2.11 Ai8051U 系列单片机中可位寻址的数据存储器 584
12.2.12 扩展 SFR 使能寄存器 EAXFR 的使用说明 586
12.3 寄存器堆 587
12.3.1 寄存器堆结构图 587
12.3.2 字节、字以及双字寄存器 588
12.3.3 专用寄存器 588
12.3.4 扩展数据指针，DPX 590
12.3.5 扩展堆栈指针，SPX 590
12.4 只读特殊功能寄存器中存储的唯一 ID 号和重要参数（CHIPID 和 CHIPIDX） 591
12.4.1 CHIPID 相关寄存器 591
12.4.2 CHIPIDX 相关寄存器 592
12.4.3 CHIPID 之全球唯一 ID 号解读 594
12.4.4 CHIPID 之内部参考信号源解读 594
12.4.5 CHIPID 之内部 32K 的 IRC 振荡频率解读 594
12.4.6 CHIPID 之高精度 IRC 参数解读 596
12.4.7 CHIPIDX 之高精度 IRC 参数解读 597
12.4.8 CHIPID 之测试时间参数解读 598
12.4.9 CHIPID 之芯片封装形式编号解读 598
12.5 范例程序 599
12.5.1 读取内部 1.19V 参考信号源值（BGV） 599

12.5.2 读取全球唯一 ID 号 600
12.5.3 读取 32K 掉电唤醒定时器的频率 602
12.5.4 用户自定义内部 IRC 频率 603
12.5.5 读写片外扩展 RAM 605
13 特殊功能寄存器（SFR、XFR），EAXFR 允许访问 XFR 607
13.1 EAXFR 允许访问扩展的 XFR，取指令不减速，访问片内部 XRAM 不减速 607
13.2 Ai8051U 系列 608
13.3 特殊功能寄存器列表（SFR：0x80-0xFF） 613
13.4 扩展特殊功能寄存器列表（XFR：0x7EFE00-0x7EFEFF） 616
13.5 扩展特殊功能寄存器列表（XFR：0x7EFD00-0x7EFDFF） 621
13.6 扩展特殊功能寄存器列表（XFR：0x7EFB00-0x7EFBFF） 626
13.7 扩展特殊功能寄存器列表（XFR：0x7EFA00-0x7EFAFF） 627
13.8 扩展特殊功能寄存器列表（XFR：0x7EF900-0x7EF9FF） 632
14 I/O 口 634
14.1 I/O 口相关寄存器 635
14.1.1 端口数据寄存器（Px） 637
14.1.2 端口模式配置寄存器（PxM0，PxM1） 637
14.1.3 端口上拉电阻控制寄存器（PxPU） 638
14.1.4 端口施密特触发控制寄存器（PxNCS） 638
14.1.5 端口电平转换速度控制寄存器（PxSR） 638
14.1.6 端口驱动电流控制寄存器（PxDR） 639
14.1.7 端口数字信号输入使能控制寄存器（PxIE） 639
14.1.8 端口下拉电阻控制寄存器（PxPD） 640
14.1.9 端口模式用户控制寄存器（PxBP） 640
14.2 配置 I/O 口 641
14.3 I/O 的结构图 642
14.3.1 准双向口（弱上拉） 642
14.3.2 强推挽输出 643
14.3.3 高阻输入 643
14.3.4 开漏模式 644
14.3.5 新增 4K@5V/6.3K@3.3V 上拉电阻和 47K@5V/32K@3.3V 下拉电阻 645
14.3.6 如何设置 I/O 口对外输出速度 646
14.3.7 如何设置 I/O 口电流驱动能力 646
14.3.8 如何降低 I/O 口对外辐射 646
14.4 AIapp-ISP | I/O 口配置工具 647
14.4.1 普通配置模式 647
14.4.2 高级配置模式 648
14.5 典型发光二极管控制电路 649
14.6 一种典型三极管控制电路 650
14.7 混合电压供电系统 3V/5V 器件 I/O 口互连 651
14.8 如何让 I/O 口上电复位时为低电平 653
14.9 I/O 口直接驱动 LED 数码管应用线路图 654
14.10 利用 74HC595 驱动 8 个数码管(串行扩展,3 根线)，节省 I/O 655
14.11 使用 I/O 和 R-2R 电阻分压实现 DAC 的经典线路图 656

14.12 利用 ADC 做按键扫描节省 I/O 口 657
14.13 范例程序 658
14.13.1 端口模式设置（适用于所有的 I/O） 658
14.13.2 双向口读写操作（适用于所有的 I/O） 658
14.13.3 打开 I/O 口内部上拉电阻（适用于所有的 I/O） 659
14.13.4 将端口设置成传统 8051 I/O 模式（适用于所有的 I/O） 660
14.14 深圳大学--I/O 口实验：I/O 口读取按键状态并点灯 662
14.14.1 实验要求 662
14.14.2 简单介绍 662
14.14.3 实验建议 662
14.14.4 原理图 663
14.14.5 I/O 口工作模式和内部上拉/下拉电阻介绍 665
a) 内部上拉电阻和外部上拉/下拉电阻的部分区别： 665
b) 准双向口介绍 665
c) 强推挽输出 666
d) 高阻输入 666
e) 开漏模式 667
14.14.6 实验 1：位操作，用 USB 直接仿真来调试 668
a) I/O 口设置 668
b) 实验程序 C 语言代码 669
c) 实验程序汇编语言代码 670
14.14.7 实验 1A--使用【AiCube】项目创建助手实现“位操作，用 USB 仿真来点灯” 674
a) 安装 AIapp-ISP 下载/编程/烧录/软件 674
b) 打开【AiCube】助手 675
c) 使用【AiCube】助手创建项目 675
d) 增加实验项目代码 678
e) 使用 AIapp-ISP 软件设置仿真芯片并连接 Ai8051U 实验箱验证实验效果 679
14.14.8 实验 2：整组 8 位端口操作，用 USB 直接仿真来调试 681
a) 实验程序代码 681
14.14.9 实验 2A--使用【AiCube】项目创建助手实现“整组 8 位端口操作，用 USB 仿真来点灯” 683
a) 安装 AIapp-ISP 下载/编程/烧录/软件 683
b) 打开【AiCube】助手 684
c) 使用【AiCube】助手创建项目 684
d) 增加实验项目代码 687
e) 使用 AIapp-ISP 软件设置仿真芯片并连接 Ai8051U 实验箱验证实验效果 688
14.14.10 实验 3：用 USB-CDC 读取按键状态在串口助手显示并点亮 LED 灯 690
a) 实验介绍： 690
b) 准备工作--添加头文件和库函数 690
c) 实验程序代码 693
d) 程序下载 695
e) 程序实现效果 696
14.14.11 实验 3A--使用【AiCube】项目创建助手实现“用 USB-CDC 读取按键状态在串口助手显示并点亮 LED 灯” 697

a) 安装 AIapp-ISP 下载/编程/烧录/软件.....697
b) 打开【AiCube】助手.....698
c) 使用【AiCube】助手创建项目.....698
d) 增加实验项目代码.....702
e) 使用 AIapp-ISP 软件系统并连接 Ai8051U 实验箱验证实验效果.....703
14.14.12 实验 4：用 USB-CDC 读取按键状态，在 ISP 软件的 【调试仿真接口 | 擎天柱-LED-DIP40】 虚拟显示.....706
a) 通信协议.....706
b) 实验介绍.....706
c) 准备工作--添加头文件和库函数.....706
d) 实验程序代码.....709
e) 程序下载.....711
f) 程序实现效果.....711
14.14.13 实验 4A--使用【AiCube】项目助手实现“用 USB-CDC 读取按键状态并在【调试接口 | 擎天柱-LED-DIP40】虚拟显示”.....713
a) 安装 AIapp-ISP 下载/编程/烧录/软件.....713
b) 打开【AiCube】助手.....714
c) 使用【AiCube】助手创建项目.....714
d) 增加实验项目代码.....718
e) 使用 AIapp-ISP 软件系统并连接 Ai8051U 实验箱验证实验效果.....719
14.14.14 实验 5：I/O 口行列按键扫描，蜂鸣器发声大型实战.....722
a) 原理图.....722
b) 实验介绍.....722
c) 准备工作--添加头文件和库函数.....723
d) 实验程序代码.....725
e) 程序下载.....730
f) 程序实现效果.....731
15 中断系统及外部中断 INT0/1/2/3/4 实验.....732
15.1 Ai8051U 系列中断源.....732
15.2 Ai8051U 中断及中断优先级结构图.....734
15.3 Ai8051U 系列中断向量地址及同级中断优先级中断查询次序表.....735
15.4 中断相关寄存器.....738
15.4.1 中断使能寄存器（中断允许位）.....741
15.4.2 中断请求寄存器（中断标志位）.....752
15.4.3 中断优先级寄存器.....759
15.5 范例程序.....766
15.5.1 INT0 中断（上升沿和下降沿），可同时支持上升沿和下降沿.....766
15.5.2 INT0 中断（下降沿）.....767
15.5.3 INT1 中断（上升沿和下降沿），可同时支持上升沿和下降沿.....767
15.5.4 INT1 中断（下降沿）.....768
15.5.5 INT2 中断（下降沿），只支持下降沿中断.....769
15.5.6 INT3 中断（下降沿），只支持下降沿中断.....770
15.5.7 INT4 中断（下降沿），只支持下降沿中断.....771
15.5.8 定时器 0 中断.....772

15.5.9 定时器 1 中断 772
15.5.10 定时器 2 中断 773
15.5.11 定时器 3 中断 774
15.5.12 定时器 4 中断 775
15.5.13 UART1 中断 776
15.5.14 UART2 中断 777
15.5.15 UART3 中断 778
15.5.16 UART4 中断 779
15.5.17 ADC 中断 780
15.5.18 LVD 中断 781
15.5.19 比较器中断 782
15.5.20 SPI 中断 783
15.5.21 I2C 中断 783
15.6 深圳大学--传统外部中断实验：INT0，INT1，INT2，INT3 785
15.6.1 深大--中断实验 1(外部中断 INT0/P3.2，INT1/P3.3 控制点灯并 USB 仿真) 785
a) 原理图 785
b) 实验介绍 786
c) 实验程序 C 语言代码 787
d) 制作仿真芯片 789
e) Keil 仿真 790
f) 认识 Keil 软件的 File Toolbar（File 工具栏） 791
g) 程序实现效果 794
h) 实验程序汇编语言代码 797
15.6.2 深大--中断实验 2(单片机主动触发外部中断 INT2/P3.6，INT3/P3.7 控制点灯并 USB 仿真) 800
a) 原理图 800
b) 实验介绍 802
c) 实验程序代码 803
d) 制作仿真芯片 804
e) Keil 仿真 806
f) 程序实现效果 807
16 普通 I/O 口均可中断，不是传统外部中断 812
16.1 I/O 口中断相关寄存器 812
16.1.1 端口中断使能寄存器（PxINTE） 813
16.1.2 端口中断标志寄存器（PxINTF） 814
16.1.3 端口中断模式配置寄存器（PxIM0，PxIM1） 814
16.1.4 端口中断优先级控制寄存器（PINIPL，PINIPH） 815
16.1.5 端口中断掉电唤醒使能寄存器（PxWKUE） 815
16.2 范例程序 816
16.2.1 P0 口下降沿中断 816
16.2.2 P1 口上升沿中断 817
16.2.3 P2 口低电平中断 819
16.2.4 P3 口高电平中断 821
17 定时器/计数器，16 位自动重装载定时器 + 8 位预分频 = 24 位 823

17.1 定时器的相关寄存器 824
17.2 定时器 0/1，16 位自动重装载，+ 8 位预分频就是 24 位定时器 825
17.2.1 定时器 0/1 控制寄存器（TCON） 825
17.2.2 定时器 0/1 模式寄存器（TMOD） 826
17.2.3 定时器 0 模式 0，16 位自动重装载，启动 8 位预分频就是 24 位定时器 827
17.2.4 定时器 0 模式 1（16 位不可重装载模式） 828
17.2.5 定时器 0 模式 2（8 位自动重装载模式） 829
17.2.6 定时器 0 模式 3（不可屏蔽中断 16 位自动重装载，实时操作系统节拍器） 830
17.2.7 定时器 1 模式 0，16 位自动重装载，启动 8 位预分频就是 24 位定时器 831
17.2.8 定时器 1 模式 1（16 位不可重装载模式） 832
17.2.9 定时器 1 模式 2（8 位自动重装载模式） 833
17.2.10 定时器 0 计数寄存器（TL0，TH0） 834
17.2.11 定时器 1 计数寄存器（TL1，TH1） 834
17.2.12 辅助寄存器 1（AUXR） 834
17.2.13 中断与时钟输出控制寄存器（INTCLKO） 835
17.2.14 定时器 0 的 8 位预分频寄存器（TM0PS） 835
17.2.15 定时器 1 的 8 位预分频寄存器（TM1PS） 835
17.2.16 定时器 0 计算公式 836
17.2.17 定时器 1 计算公式 837
17.3 定时器 2，16 位自动重装载，+ 8 位预分频就是 24 位定时器 838
17.3.1 辅助寄存器 1（AUXR） 838
17.3.2 中断与时钟输出控制寄存器（INTCLKO） 838
17.3.3 定时器 2 计数寄存器（T2L，T2H） 838
17.3.4 定时器 2 的 8 位预分频寄存器（TM2PS） 838
17.3.5 定时器 2 工作模式 839
17.3.6 定时器 2 计算公式 839
17.4 定时器 3/4，24 位定时器，8 位预分频+16 位自动重装载定时器 840
17.4.1 定时器 3/4 功能脚切换 840
17.4.2 定时器 4/3 控制寄存器（T4T3M） 840
17.4.3 定时器 3 计数寄存器（T3L，T3H） 841
17.4.4 定时器 4 计数寄存器（T4L，T4H） 841
17.4.5 定时器 3 的 8 位预分频寄存器（TM3PS） 841
17.4.6 定时器 4 的 8 位预分频寄存器（TM4PS） 841
17.4.7 定时器 3 工作模式 842
17.4.8 定时器 4 工作模式 843
17.4.9 定时器 3 计算公式 844
17.4.10 定时器 4 计算公式 844
17.5 定时器 T11（24 位定时器，8 位预分频+16 位自动重装载定时器） 845
17.5.1 定时器 T11 控制寄存器（T11CR） 845
17.5.2 定时器 T11 的 8 位预分频寄存器（T11PS） 846
17.5.3 定时器 T11 计数寄存器（T11L，T11H） 846
17.5.4 定时器 T11 工作模式 847
17.5.5 定时器 T11 计算公式 848
17.6 AIapp-ISP | 定时器计算器工具 849

17.7 范例程序 850
17.7.1 定时器 0（模式 0－16 位自动重载），用作定时 850
17.7.2 定时器 0（模式 1－16 位不自动重载），用作定时 850
17.7.3 定时器 0（模式 2－8 位自动重载），用作定时 851
17.7.4 定时器 0（模式 3－16 位自动重载不可屏蔽中断），用作定时 852
17.7.5 定时器 0（外部计数－扩展 T0 为外部下降沿中断） 853
17.7.6 定时器 0（测量脉宽－INT0 高电平宽度） 854
17.7.7 定时器 0（模式 0），时钟分频输出 855
17.7.8 定时器 1（模式 0－16 位自动重载），用作定时 856
17.7.9 定时器 1（模式 1－16 位不自动重载），用作定时 857
17.7.10 定时器 1（模式 2－8 位自动重载），用作定时 857
17.7.11 定时器 1（外部计数－扩展 T1 为外部下降沿中断） 858
17.7.12 定时器 1（测量脉宽－INT1 高电平宽度） 859
17.7.13 定时器 1（模式 0），时钟分频输出 860
17.7.14 定时器 1（模式 0）做串口 1 波特率发生器 861
17.7.15 定时器 1（模式 2）做串口 1 波特率发生器 863
17.7.16 定时器 2（16 位自动重载），用作定时 864
17.7.17 定时器 2（外部计数－扩展 T2 为外部下降沿中断） 865
17.7.18 定时器 2，时钟分频输出 866
17.7.19 定时器 2 做串口 1 波特率发生器 867
17.7.20 定时器 2 做串口 2 波特率发生器 869
17.7.21 定时器 2 做串口 3 波特率发生器 870
17.7.22 定时器 2 做串口 4 波特率发生器 872
17.7.23 定时器 3（16 位自动重载），用作定时 874
17.7.24 定时器 3（外部计数－扩展 T3 为外部下降沿中断） 875
17.7.25 定时器 3，时钟分频输出 876
17.7.26 定时器 3 做串口 3 波特率发生器 877
17.7.27 定时器 4（16 位自动重载），用作定时 878
17.7.28 定时器 4（外部计数－扩展 T4 为外部下降沿中断） 879
17.7.29 定时器 4，时钟分频输出 880
17.7.30 定时器 4 做串口 4 波特率发生器 881
17.7.31 定时器 T11 应用范例 883
17.8 深圳大学--定时器实验：定时器 0/1，24 位定时器 885
17.8.1 深大--定时器 0/1，24 位定时器实验（定时器中断点灯并通过 USB-CDC 端口输出信息） 885
a) 实验介绍 885
b) 定时器 0 模式 0 结构图，定时器 1 模式 0 结构图 885
c) 配合本实验的 LED 指示灯原理图 886
d) 准备工作--添加头文件和库函数 887
e) 实验程序代码 889
f) 程序下载 893
g) 程序实现效果 894
17.8.2 深大--使用【AiCube】工具实现：定时器 0/1，24 位定时器实验（定时器中断点灯并通过 USB-CDC 端口输出信息） 895

a) 实验介绍 895
b) 定时器 0 模式 0 结构图，定时器 1 模式 0 结构图 895
c) 配合本实验的 LED 指示灯原理图 896
d) 安装 AIapp-ISP 下载/编程/烧录/软件 897
e) 打开【AiCube】 898
f) 使用【AiCube】创建项目 898
g) 增加实验项目代码 902
h) 使用 Ai8051U 实验箱查验实验效果 903
17.8.3 深大--定时器 0/1，INT0/INT1，中断组合应用实验，演示程序 904
17.9 大型综合实战--深大课程设计：时钟+数码管+矩阵按键+蜂鸣器发声 908
17.9.1 实验介绍 908
a) 按键扫描说明： 908
b) 实验现象和操作说明： 908
c) Keil 环境下多文件项目管理说明 909
17.9.2 原理图 909
17.9.3 准备工作--添加头文件和库函数 910
a) 添加常用头文件 ai_usb.h 910
b) 添加 usb_cdc 库函数文件 910
17.9.4 流程图 912
a) main -- 程序主函数 流程图 912
b) key -- 按键扫描以及按键事件处理 流程图 913
c) led -- 扫描 LED 数码管 流程图 914
d) timer -- 定时器初始化以及定时器中断服务程序 流程图 915
e) rtc -- 软件模拟 RTC 程序 流程图 916
f) beep -- 蜂鸣器发声控制程序 流程图 918
17.9.5 实验程序代码 919
a) main.c -- 程序主函数 919
b) key.c -- 按键扫描以及按键事件处理 920
c) led.c -- 扫描 LED 数码管 922
d) timer.c -- 定时器初始化以及定时器中断服务程序 924
e) rtc.c -- 软件模拟 RTC 程序 926
f) beep.c -- 蜂鸣器发声控制程序 929
g) config.h -- 项目配置的头文件 931
h) key.h -- 项目配置文件的头文件 932
i) led.h -- 项目配置文件的头文件 932
j) timer.h -- 项目配置文件的头文件 932
k) rtc.h -- 项目配置文件的头文件 932
l) beep.h -- 蜂鸣器发声控制程序的头文件 933
17.9.6 程序下载 933
17.9.7 程序实现效果 934
18 RTC 实时时钟，年/月/日/时/分/秒 935
18.1 RTC 相关的寄存器 936
18.1.1 RTC 控制寄存器（RTCCR） 937
18.1.2 RTC 配置寄存器（RTCCFG） 937

18.1.3 RTC 中断使能寄存器（RTCIEN） 938
18.1.4 RTC 中断请求寄存器（RTCIF） 939
18.1.5 RTC 闹钟设置寄存器 939
18.1.6 RTC 实时时钟初始值设置寄存器 940
18.1.7 RTC 实时时钟计数寄存器 941
18.2 RTC 实战线路图 942
18.3 范例程序 943
18.3.1 串口打印 RTC 时钟范例 943
18.3.2 利用 ISP 软件的用户接口实现不停电下载保持 RTC 参数 946
18.3.3 内部 RTC 时钟低功耗休眠唤醒-比较器检测电压程序 952
18.4 深圳大学--RTC 实验 957
18.4.1 深大-RTC 实验(拓展学习之使用 RTC 功能设定闹钟) 957
a) 原理图 957
b) 实验介绍 957
c) 准备工作--添加头文件和库函数 957
d) 实验程序代码 961
e) 程序下载 965
f) 程序实现效果 965
18.4.2 深大-使用【AiCube】助手实现 RTC 实验(测试 RTC，使用 USB-CDC 打印年/月/日/时/分/秒) 967
a) 原理图 967
b) 实验介绍 967
c) 准备工作--添加 Keil 中断拓展插件 967
d) 安装 AIapp-ISP 下载/编程/烧录/软件 969
e) 打开【AiCube】助手 969
f) 使用【AiCube】助手创建项目 970
g) 增加实验项目代码 974
h) 使用 AIapp-ISP 软件系统并连接 Ai8051U 实验箱验证实验效果 975
19 同步/异步串口通信，硬件奇偶校验（USART1、USART2） 977
19.1 串口功能脚切换 978
19.2 串口相关寄存器 979
19.3 串口 1（同步/异步串口 USART） 981
19.3.1 串口 1 控制寄存器（SCON） 981
19.3.2 串口 1 数据寄存器（SBUF） 982
19.3.3 电源管理寄存器（PCON） 982
19.3.4 辅助寄存器 1（AUXR） 982
19.3.5 串口 1 模式 0，模式 0 波特率计算公式 983
19.3.6 串口 1 模式 1，模式 1 波特率计算公式 985
19.3.7 串口 1 模式 2，模式 2 波特率计算公式 987
19.3.8 串口 1 模式 3，模式 3 波特率计算公式 988
19.3.9 自动地址识别，从机地址控制寄存器（SADDR，SADEN） 989
19.3.10 串口 1 同步模式控制寄存器 1（USARTCR1） 990
19.3.11 串口 1 同步模式控制寄存器 2，硬件奇偶校验位设置（USARTCR2） 991
19.3.12 串口 1 同步模式控制寄存器 3（USARTCR3） 991

19.3.13 串口 1 同步模式控制寄存器 4（USARTCR4）.....992
19.3.14 串口 1 同步模式控制寄存器 5（USARTCR5）.....992
19.3.15 串口 1 同步模式保护时间寄存器（USARTGTR）.....993
19.3.16 串口 1 同步模式波特率寄存器（USARTBR）.....993
19.3.17 串口 1 接收超时控制寄存器（UR1TOCR）.....994
19.3.18 串口 1 超时状态寄存器（UR1TOSR）.....994
19.3.19 串口 1 超时长度控制寄存器（UR1TOTE/H/L）.....994
19.4 串口 2（同步/异步串口 USART2）.....995
19.4.1 串口 2 控制寄存器（S2CON）.....995
19.4.2 串口 2 数据寄存器（S2BUF）.....996
19.4.3 串口 2 配置寄存器（S2CFG）.....996
19.4.4 串口 2 模式 0，模式 0 波特率计算公式.....997
19.4.5 串口 2 模式 1，模式 1 波特率计算公式.....999
19.4.6 串口 2 模式 2，模式 2 波特率计算公式.....1001
19.4.7 串口 2 模式 3，模式 3 波特率计算公式.....1002
19.4.8 串口 2 自动地址识别.....1003
19.4.9 串口 2 从机地址控制寄存器（S2ADDR，S2ADEN）.....1003
19.4.10 串口 2 同步模式控制寄存器 1（USART2CR1）.....1004
19.4.11 串口 2 同步模式控制寄存器 2（USART2CR2）.....1005
19.4.12 串口 2 同步模式控制寄存器 3（USART2CR3）.....1006
19.4.13 串口 2 同步模式控制寄存器 4（USART2CR4）.....1006
19.4.14 串口 2 同步模式控制寄存器 5（USART2CR5）.....1007
19.4.15 串口 2 同步模式保护时间寄存器（USART2GTR）.....1007
19.4.16 串口 2 同步模式波特率寄存器（USART2BR）.....1007
19.4.17 串口 2 接收超时控制寄存器（UR2TOCR）.....1008
19.4.18 串口 2 超时状态寄存器（UR2TOSR）.....1008
19.4.19 串口 2 超时长度控制寄存器（UR2TOTE/H/L）.....1008
19.5 AIapp-ISP | 串口波特率计算器工具.....1009
19.6 AIapp-ISP | 串口助手/USB-CDC.....1010
19.7 范例程序.....1014
19.7.1 串口 1 使用定时器 2 做波特率发生器.....1014
19.7.2 串口 1 使用定时器 1（模式 0）做波特率发生器.....1015
19.7.3 串口 1 使用定时器 1（模式 2）做波特率发生器.....1017
19.7.4 串口 2 使用定时器 2 做波特率发生器.....1019
19.7.5 使用 USART1 的 SPI 接口访问串行 FLASH（DMA 方式）.....1021
19.7.6 USART1 和 USART2 的 SPI 接口相互传输数据（中断方式）.....1026
19.7.7 串口多机通讯（主机模式）.....1030
19.7.8 串口多机通讯（从机模式）.....1035
19.8 深大实验--串口通讯(硬件 UART1 与电脑收发通讯).....1042
19.8.1 实验介绍.....1042
19.9 深大实验--串口通讯(硬件 UART1 与 UART2 收发通讯).....1042
19.9.1 实验介绍.....1042
19.10 深大实验--串口通讯(硬件 UART1 与 UART2 数据透传).....1042
19.10.1 实验介绍.....1042

19.11 大型综合实战--深大课程设计：硬件 UART1 发送按键码，接收端蜂鸣器根据收到的键码发声 1042
19.11.1 实验介绍 1042
19.12 深大实验--串口通讯(串口通讯，串口绘图，ADC 检测按键，ADC 热敏电阻测温) 1043
19.12.1 实验介绍 1043
19.13 深大实验--串口通讯(硬件 UART1 与 UART2 收发通讯-奇偶校验) 1043
19.13.1 实验介绍 1043
19.14 深大实验--串口通讯(USB-CDC 发命令控制硬件 UART1 与 UART2 多机通讯) 1044
19.14.1 实验介绍 1044
19.15 大学实验--串口 1_2_3_4(中断设置标志位，主循环查询)，演示实验 1044
19.16 大学实验--串口 1_2_3_4 + 定时器 0_1_3_4 + 外部中断 INT0_1_2_3 (中断设置标志位，主循环查询)，演示实验 1050
20 异步串口通信（UART3、UART4） 1062
20.1 串口功能脚切换 1062
20.2 串口相关寄存器 1063
20.3 串口 3（异步串口 UART3） 1064
20.3.1 串口 3 控制寄存器（S3CON） 1064
20.3.2 串口 3 数据寄存器（S3BUF） 1064
20.3.3 串口 3 接收超时控制寄存器（UR3TOCR） 1065
20.3.4 串口 3 超时状态寄存器（UR3TOSR） 1065
20.3.5 串口 3 超时长度控制寄存器（UR3TOTE/H/L） 1065
20.3.6 串口 3 模式 0，模式 0 波特率计算公式 1066
20.3.7 串口 3 模式 1，模式 1 波特率计算公式 1067
20.4 串口 4（异步串口 UART4） 1068
20.4.1 串口 4 控制寄存器（S4CON） 1068
20.4.2 串口 4 数据寄存器（S4BUF） 1068
20.4.3 串口 4 接收超时控制寄存器（UR4TOCR） 1069
20.4.4 串口 4 超时状态寄存器（UR4TOSR） 1069
20.4.5 串口 4 超时长度控制寄存器（UR4TOTE/H/L） 1069
20.4.6 串口 4 模式 0，模式 0 波特率计算公式 1070
20.4.7 串口 4 模式 1，模式 1 波特率计算公式 1071
20.5 串口注意事项 1072
20.6 范例程序 1073
20.6.1 串口 3 使用定时器 2 做波特率发生器 1073
20.6.2 串口 3 使用定时器 3 做波特率发生器 1074
20.6.3 串口 4 使用定时器 2 做波特率发生器 1076
20.6.4 串口 4 使用定时器 4 做波特率发生器 1078
21 比较器，掉电检测，内部固定比较电压 1080
21.1 比较器内部结构图 1080
21.2 比较器功能脚切换 1080
21.3 比较器相关的寄存器 1081
21.3.1 比较器控制寄存器 1（CMPCR1） 1081
21.3.2 比较器控制寄存器 2（CMPCR2） 1082

21.3.3 比较器扩展配置寄存器（CMPEXCFG）.....1083
21.4 范例程序.....1084
21.4.1 比较器的使用（中断方式）.....1084
21.4.2 比较器的使用（查询方式）.....1085
21.4.3 比较器的多路复用应用（比较器+ADC 输入通道）.....1086
21.4.4 比较器作外部掉电检测（掉电过程中应及时保存用户数据到 EEPROM 中）.....1088
21.4.5 比较器检测工作电压（电池电压）.....1089
22 IAP/EEPROM.....1092
22.1 EEPROM 操作时间.....1092
22.2 关于 EEPROM 操作时是否需要关闭中断的问题.....1092
22.3 EEPROM 相关的寄存器.....1093
22.3.1 EEPROM 数据寄存器（IAP_DATA）.....1093
22.3.2 EEPROM 地址寄存器（IAP_ADDR）.....1093
22.3.3 EEPROM 命令寄存器（IAP_CMD）.....1093
22.3.4 EEPROM 触发寄存器（IAP_TRIG）.....1094
22.3.5 EEPROM 控制寄存器（IAP_CONTR）.....1095
22.3.6 EEPROM 擦除等待时间控制寄存器（IAP_TPS）.....1095
22.4 EEPROM 大小及地址.....1096
22.4.1 Ai8051U 系列 EEPROM 操作.....1096
22.5 范例程序.....1098
22.5.1 EEPROM 基本操作.....1098
22.5.2 使用 MOV 读取 EEPROM（Ai8051U 系列）.....1100
22.5.3 使用串口送出 EEPROM 数据.....1102
22.5.4 串口 1 读写 EEPROM-带 MOV 读.....1104
22.5.5 口令擦除写入-多扇区备份-串口 1 操作.....1111
23 ADC 模数转换、传统 DAC 实现.....1120
23.1 ADC 内部结构图（ADC 框图）.....1121
23.2 ADC 相关的寄存器.....1122
23.2.1 ADC 控制寄存器（ADC_CONTR），PWM 触发 ADC 控制.....1123
23.2.2 ADC 配置寄存器（ADCCFG）.....1124
23.2.3 ADC 转换结果寄存器（ADC_RES，ADC_RESL）.....1124
23.2.4 ADC 时序控制寄存器（ADCTIM）.....1125
23.2.5 ADC 扩展配置寄存器（ADCEXCFG）.....1126
23.3 ADC 静态特性.....1127
23.4 ADC 相关计算公式.....1128
23.4.1 ADC 速度计算公式.....1128
23.4.2 ADC 转换结果计算公式.....1128
23.4.3 反推 ADC 输入电压计算公式.....1128
23.4.4 反推工作电压计算公式.....1128
23.5 ADC 应用参考线路图.....1129
23.5.1 ADC 参考线路图，输入信号与内部 1.19V 或外接 2.5V 辅助信号源进行比较，只需计算一次 1129
23.6 AIapp-ISP | ADC 转换速度计算器工具.....1132
23.7 范例程序.....1133

23.7.1 ADC 基本操作（查询方式）.....1133
23.7.2 ADC 基本操作（中断方式）.....1134
23.7.3 格式化 ADC 转换结果.....1135
23.7.4 利用 ADC15 通道在内部固定接的 1.19V 辅助固定信号源，反推其他通道的输入电压或 VCC，只需计算一次.....1136
23.7.5 ADC 作按键扫描应用线路图.....1141
23.7.6 检测负电压参考线路图.....1141
23.7.7 常用加法电路在 ADC 中的应用.....1142
23.8 大型综合实战 -- 深大课程设计：模拟 RTC 时钟+数码管+ADC 查询方式检测按键 1143
23.8.1 实验介绍.....1143
a) 按键扫描说明：.....1143
b) ADC 按键实验现象和操作说明：.....1143
c) Keil 环境下多文件项目管理说明.....1144
23.8.2 原理图.....1145
a) ADC 按键原理图.....1145
b) 74HC595 驱动数码管原理图.....1145
23.8.3 准备工作--添加头文件和库函数.....1146
a) 添加常用头文件 ai_usb.h.....1146
b) 添加 usb_cdc 库函数文件.....1146
23.8.4 实验程序代码.....1148
a) main.c -- 程序主函数.....1148
b) key.c -- 按键扫描以及按键事件处理.....1149
c) led.c -- 扫描 LED 数码管.....1151
d) timer.c -- 定时器初始化以及定时器中断服务程序.....1154
e) rtc.c -- 软件模拟 RTC 程序.....1155
f) adc.c -- ADC 初始化和 ADC 查询方式检测按键.....1158
g) config.h -- 项目配置的头文件.....1159
h) key.h -- 项目配置文件的头文件.....1160
i) led.h -- 项目配置文件的头文件.....1160
j) timer.h -- 项目配置文件的头文件.....1160
k) rtc.h -- 项目配置文件的头文件.....1160
l) adc.h -- 项目配置文件的头文件.....1161
23.8.5 程序下载.....1161
23.8.6 程序实现效果.....1162
23.9 深大实验 -- ADC 中断方式检测 NTC 热敏电阻测温+数码管显示.....1163
23.9.1 实验介绍.....1163
23.9.2 ADC 通过采集 NTC 热敏电阻的电压值进行测温.....1163
a) 操作说明和实验现象：.....1165
b) Keil 环境下多文件项目管理说明.....1165
23.9.3 原理图.....1166
a) NTC 温度检测原理图.....1166
b) 74HC595 驱动数码管原理图.....1166
23.9.4 准备工作--添加头文件和库函数.....1167

a) 添加常用头文件 ai_usb.h....1167
b) 添加 usb_cdc 库函数文件....1167
23.9.5 实验程序代码....1169
a) main.c -- 程序主函数....1169
b) led.c -- 扫描 LED 数码管....1170
c) timer.c -- 定时器初始化以及定时器中断服务程序....1172
d) adc.c -- ADC 初始化和 ADC 中断方式数据采集程序....1173
e) ntc.c -- ADC 中断方式检测 NTC 热敏电阻测温程序....1174
f) config.h -- 项目配置的头文件....1180
g) led.h -- 项目配置文件的头文件....1180
h) timer.h -- 项目配置文件的头文件....1181
i) adc.h -- 项目配置文件的头文件....1181
j) ntc.h -- 项目配置文件的头文件....1181
23.9.6 程序下载....1181
23.9.7 实验现象....1182
23.10 使用 I/O 和 R-2R 电阻分压实现 DAC 的经典线路图....1183
23.11 利用 PWM 实现 16 位 DAC 的参考线路图....1184
24 PCA/CCP/PWM 应用....1185
24.1 PCA 功能脚切换....1185
24.2 PCA 相关的寄存器....1186
24.2.1 PCA 控制寄存器（CCON）....1187
24.2.2 PCA 模式寄存器（CMOD）....1187
24.2.3 PCA 计数器寄存器（CL，CH）....1188
24.2.4 PCA 模块模式控制寄存器（CCAPMn）....1188
24.2.5 PCA 模块模式捕获值/比较值寄存器（CCAPnL，CCAPnH）....1188
24.2.6 PCA 模块 PWM 模式控制寄存器（PCA_PWMn）....1189
24.3 PCA 工作模式....1190
24.3.1 捕获模式....1190
24.3.2 软件定时器模式....1191
24.3.3 高速脉冲输出模式....1191
24.3.4 PWM 脉宽调制模式及频率计算公式....1192
24.3.4.1 8 位 PWM 模式....1192
24.3.4.2 7 位 PWM 模式....1193
24.3.4.3 6 位 PWM 模式....1194
24.3.4.4 10 位 PWM 模式....1195
24.3.4.5 如何控制 PWM 固定输出高电平/低电平....1195
24.4 范例程序....1196
24.4.1 PCA 输出 PWM（6/7/8/10 位）....1196
24.4.2 PCA 捕获测量脉冲宽度....1197
24.4.3 PCA 实现 16 位软件定时....1198
24.4.4 PCA 实现 16 位软件定时（ECI 外部时钟模式）....1199
24.4.5 PCA 输出高速脉冲....1200
24.4.6 PCA 扩展外部中断....1201
25 同步串行外设接口（SPI）....1203

25.1 SPI 的 MOSI 和 MISO 脚交换控制 1203
25.2 SPI 功能脚切换 1203
25.3 SPI 相关的寄存器 1204
25.3.1 SPI 状态寄存器（SPSTAT） 1204
25.3.2 SPI 控制寄存器（SPCTL），SPI 速度控制 1205
25.3.3 SPI 数据寄存器（SPDAT） 1205
25.3.4 SPI 从机超时控制寄存器（SPITOCR） 1206
25.3.5 SPI 从机超时状态寄存器（SPITOSR） 1206
25.3.6 SPI 从机超时长度控制寄存器（SPITOTE/H/L） 1206
25.4 SPI 通信方式 1207
25.4.1 单主单从 1207
25.4.2 互为主从 1208
25.4.3 单主多从 1209
25.5 配置 SPI 1210
25.6 数据模式 1212
25.7 范例程序 1213
25.7.1 SPI 单主单从系统主机程序（中断方式） 1213
25.7.2 SPI 单主单从系统从机程序（中断方式） 1214
25.7.3 SPI 单主单从系统主机程序（查询方式） 1215
25.7.4 SPI 单主单从系统从机程序（查询方式） 1216
25.7.5 SPI 互为主从系统程序（中断方式） 1216
25.7.6 SPI 互为主从系统程序（查询方式） 1218
25.7.7 实战硬件 SPI 控制 74HC595 驱动数码管显示模拟 RTC 时钟 1219
25.8 深大实验 -- 硬件 SPI 控制 74HC595 驱动数码管显示模拟 RTC 时钟 1222
a) 实验介绍 1222
b) 原理图 1222
c) 实验程序代码 1223
d) 程序下载 1230
e) 实验现象 1230
25.9 大型综合实战--大学课程设计：硬件 SPI 读写串行 Flash 1231
25.9.1 实验介绍 1231
25.10 深大实验--硬件 SPI 驱动 OLED 显示图片、图形、字符 1232
25.10.1 实验介绍 1232
a) Aiapp-ISP 系统软件图片取模工具的使用 1232
b) Aiapp-ISP 系统软件字库生成工具的使用 1233
c) 使用 Aiapp-ISP 系统软件图片取模工具生成 ASCII 的点阵字符 C 代码 1234
d) 认识 Ai8051U 实验箱和 OLED 屏 1236
e) Keil 环境下多文件项目管理说明 1236
25.10.2 原理图 1237
26 高速 SPI（HSSPI） 1238
26.1 相关寄存器 1238
26.1.1 高速 SPI 配置寄存器（HSSPI_CFG） 1239
26.1.2 高速 SPI 配置寄存器 2（HSSPI_CFG2）（交换 MISO 和 MOSI） 1239
26.1.3 高速 SPI 状态寄存器（HSSPI_STA） 1240

26.1.4 高速 SPI 时钟分频器（HSSPI_PSCR）.....1240
26.2 范例程序.....1241
26.2.1 使能 SPI 的高速模式.....1241
27 四线 SPI（QSPI）.....1243
27.1 QSPI 功能脚切换.....1243
27.2 QSPI 主要特性.....1243
27.3 QSPI 功能说明.....1244
27.3.1 QSPI 框图.....1244
27.3.2 QSPI 应用注意事项.....1244
27.3.3 QSPI 命令序列.....1245
27.3.4 QSPI 信号接口协议模式.....1246
27.3.5 QSPI 间接模式.....1247
27.3.6 QSPI 状态标志轮询模式.....1249
27.3.7 QSPI Flash 配置.....1250
27.3.8 QSPI 延迟数据采样.....1250
27.3.9 QSPI 配置.....1250
27.3.10 QSPI 的用法.....1251
27.3.11 指令仅发送一次.....1252
27.3.12 QSPI 差错管理.....1252
27.3.13 QSPI 的繁忙位和中止功能.....1252
27.3.14 nCS 行为.....1253
27.4 QSPI 中断.....1254
27.5 QSPI 相关的寄存器.....1255
27.5.1 QSPI 控制寄存器 1（QSPI_CR1）.....1255
27.5.2 QSPI 控制寄存器 2（QSPI_CR2）.....1256
27.5.3 QSPI 控制寄存器 3（QSPI_CR3）.....1256
27.5.4 QSPI 控制寄存器 4（QSPI_CR4）.....1257
27.5.5 QSPI 器件配置寄存器 1（QSPI_DCR1）.....1257
27.5.6 QSPI 器件配置寄存器 2（QSPI_DCR2）.....1257
27.5.7 QSPI 状态寄存器 1（QSPI_SR1）.....1258
27.5.8 QSPI 状态寄存器 2（QSPI_SR2）.....1258
27.5.9 QSPI 标志清零寄存器（QSPI_FCR）.....1258
27.5.10 QSPI 数据长度寄存器（QSPI_DLR）.....1259
27.5.11 QSPI 通信配置寄存器 1（QSPI_CCR1）.....1259
27.5.12 QSPI 通信配置寄存器 2（QSPI_CCR2）.....1260
27.5.13 QSPI 通信配置寄存器 3（QSPI_CCR3）.....1260
27.5.14 QSPI 通信配置寄存器 4（QSPI_CCR4）.....1261
27.5.15 QSPI 地址寄存器（QSPI_AR）.....1261
27.5.16 QSPI 交替字节寄存器（QSPI_ABR）.....1261
27.5.17 QSPI 数据寄存器（QSPI_DR）.....1262
27.5.18 QSPI 状态屏蔽寄存器（QSPI_PSMKR）.....1262
27.5.19 QSPI 状态匹配寄存器（QSPI_PSMAR）.....1262
27.5.20 QSPI 轮训间隔寄存器（QSPI_PIR）.....1262
27.6 范例程序.....1263

27.6.1 使用 QSPI 直接读写串行 Flash....1263
27.6.2 使用 QSPI DMA 读写串行 Flash....1263
28 I2C 总线....1264
28.1 I2C 功能脚切换....1264
28.2 I2C 相关的寄存器....1265
28.3 I2C 主机模式....1266
28.3.1 I2C 配置寄存器（I2CCFG），总线速度控制 1....1266
28.3.2 I2C 主机时钟分频寄存器（I2CPSCR），总线速度控制 2....1266
28.3.3 I2C 主机控制寄存器（I2CMSCR）....1268
28.3.4 I2C 主机辅助控制寄存器（I2CMSAUX）....1270
28.3.5 I2C 主机状态寄存器（I2CMSST）....1270
28.4 I2C 从机模式....1271
28.4.1 I2C 从机控制寄存器（I2CSLCR）....1271
28.4.2 I2C 从机状态寄存器（I2CSLST）....1271
28.4.3 I2C 从机地址寄存器（I2CSLADR）....1273
28.4.4 I2C 数据寄存器（I2CTXD，I2CRXD）....1273
28.4.5 I2C 从机超时控制寄存器（I2CTOCR）....1274
28.4.6 I2C 从机超时状态寄存器（I2CTOSR）....1274
28.4.7 I2C 从机超时长度控制寄存器（I2CTOTE/H/L）....1274
28.5 范例程序....1275
28.5.1 I2C 主机模式访问 AT24C256（中断方式）....1275
28.5.2 I2C 主机模式访问 AT24C256（查询方式）....1278
28.5.3 I2C 主机模式访问 PCF8563....1281
28.5.4 I2C 从机模式（中断方式）....1284
28.5.5 I2C 从机模式（查询方式）....1286
28.5.6 测试 I2C 从机模式代码的主机代码....1288
29 16 位高级 PWM 定时器，支持正交编码....1290
29.1 高级 PWM 定时器（PWMA）内部结构框图....1292
29.2 高级 PWM 定时器（PWMB）内部结构框图....1294
29.3 简介....1296
29.4 主要特性....1296
29.5 时基单元....1297
29.5.1 读写 16 位计数器....1298
29.5.2 16 位 PWMA_ARR 寄存器的写操作....1298
29.5.3 预分频器....1298
29.5.4 向上计数模式....1299
29.5.5 向下计数模式....1301
29.5.6 中间对齐模式（向上/向下计数）....1303
29.5.7 重复计数器....1305
29.6 时钟/触发控制器....1306
29.6.1 预分频时钟（CK_PSC）....1306
29.6.2 内部时钟源（fMASTER）....1306
29.6.3 外部时钟源模式 1....1307
29.6.4 外部时钟源模式 2....1308

29.6.5 触发同步 1309
29.6.6 与 PWMB 同步 1311
29.7 捕获/比较通道 1314
29.7.1 16 位 PWMA_CCRi 寄存器的写流程 1316
29.7.2 输入模块 1317
29.7.3 输入捕获模式 1318
29.7.4 输出模块 1320
29.7.5 强制输出模式 1321
29.7.6 输出比较模式 1321
29.7.7 PWM 模式 1322
29.7.8 使用刹车功能（PWMFLT） 1328
29.7.9 在外部事件发生时清除 OCiREF 信号 1330
29.7.10 编码器接口模式 1330
29.8 中断 1332
29.9 PWMA/PWMB 寄存器描述 1334
29.9.1 功能脚切换（PWMx_PS） 1334
29.9.2 高级 PWM 功能脚选择寄存器（PWMx_ETRPS） 1336
29.9.3 输出使能寄存器（PWMx_ENO） 1337
29.9.4 输出附加使能寄存器（PWMx_IOAUX） 1338
29.9.5 控制寄存器 1（PWMx_CR1） 1339
29.9.6 控制寄存器 2（PWMx_CR2），及实时触发 ADC 1340
29.9.7 从模式控制寄存器(PWMx_SMCR) 1343
29.9.8 外部触发寄存器(PWMx_ETR) 1346
29.9.9 中断使能寄存器(PWMx_IER) 1347
29.9.10 状态寄存器 1(PWMx_SR1) 1347
29.9.11 状态寄存器 2(PWMx_SR2) 1348
29.9.12 事件产生寄存器（PWMx_EGR） 1349
29.9.13 捕获/比较模式寄存器 1（PWMx_CCMR1） 1350
29.9.14 捕获/比较模式寄存器 2（PWMx_CCMR2） 1353
29.9.15 捕获/比较模式寄存器 3（PWMx_CCMR3） 1355
29.9.16 捕获/比较模式寄存器 4（PWMx_CCMR4） 1357
29.9.17 捕获/比较使能寄存器 1（PWMx_CCER1） 1359
29.9.18 捕获/比较使能寄存器 2（PWMx_CCER2） 1360
29.9.19 计数器高 8 位（PWMx_CNTRH） 1361
29.9.20 计数器低 8 位（PWMx_CNTRL） 1361
29.9.21 预分频器高 8 位（PWMx_PSCRH），输出频率计算公式 1361
29.9.22 预分频器低 8 位（PWMx_PSCRL） 1361
29.9.23 自动重装载寄存器高 8 位（PWMx_ARRH） 1362
29.9.24 自动重装载寄存器低 8 位（PWMx_ARRL） 1362
29.9.25 重复计数器寄存器（PWMx_RCR） 1362
29.9.26 捕获/比较寄存器 1/5 高 8 位（PWMx_CCR1H） 1362
29.9.27 捕获/比较寄存器 1/5 低 8 位（PWMx_CCR1L） 1363
29.9.28 捕获/比较寄存器 2/6 高 8 位（PWMx_CCR2H） 1363
29.9.29 捕获/比较寄存器 2/6 低 8 位（PWMx_CCR2L） 1363

29.9.30 捕获/比较寄存器 3/7 高 8 位（PWMx_CCR3H）.....1363
29.9.31 捕获/比较寄存器 3/7 低 8 位（PWMx_CCR3L）.....1363
29.9.32 捕获/比较寄存器 4/8 高 8 位（PWMx_CCR4H）.....1363
29.9.33 捕获/比较寄存器 4/8 低 8 位（PWMx_CCR4L）.....1364
29.9.34 刹车寄存器（PWMx_BKR）.....1364
29.9.35 死区寄存器（PWMx_DTR）.....1365
29.9.36 输出空闲状态寄存器（PWMx_OISR）.....1366
29.10 范例程序及应用线路图.....1367
29.10.1 PWMA+PWMB 实现 8 组定时器.....1367
29.10.2 高级 PWM 时钟输出应用（系统时钟分频输出）.....1371
29.10.3 输出任意周期和任意占空比的波形.....1372
29.10.4 输出占空比为 100%和 0%的 PWM 波形的方法（以 PWM1P 为例）.....1374
29.10.4.1 方法 1：设置 PWMx_ENO 禁止输出 PWM.....1374
29.10.4.2 方法 2：设置 PWMx_CCMRn 寄存器强制输出有效/无效电平.....1374
29.10.5 高级 PWM 输出-频率可调-脉冲计数（软件方式）.....1375
29.10.6 高级 PWM 输出-频率可调-脉冲计数（硬件方式）.....1378
29.10.7 高级 PWM 端口做外部中断（仅下降沿中断或者仅上升沿中断）.....1381
29.10.8 高级 PWM 一个端口做外部中断（同时捕获下降沿和上升沿）.....1382
29.10.9 输入捕获模式测量脉冲周期（捕获上升沿到上升沿或者下降沿到下降沿）.....1383
29.10.10 输入捕获模式测量脉冲高电平宽度（捕获上升沿到下降沿）.....1386
29.10.11 输入捕获模式测量脉冲低电平宽度（捕获下降沿到上升沿）.....1388
29.10.12 输入捕获模式同时测量脉冲周期和高电平宽度（占空比）.....1390
29.10.13 同时捕获 4 路输入信号的周期和高电平宽度（占空比）.....1392
29.10.14 带死区控制的 PWM 互补输出.....1397
29.10.15 利用 PWM 实现互补 SPWM.....1398
29.10.16 产生 3 路相位差 120 度的互补 PWM 波形（网友提供）.....1402
29.10.17 使用 PWM 的 CEN 启动 PWMA 定时器，实时触发 ADC.....1404
29.10.18 PWM 周期重复触发 ADC.....1405
29.10.19 利用 PWM 实现 16 位 DAC 的参考线路图.....1406
29.10.20 正交编码器模式演示程序.....1406
29.10.21 使用高级 PWM 实现编码器演示程序.....1407
29.10.22 无 HALL 三相无刷电机驱动演示程序，电位器调速用.....1411
29.10.23 带 HALL 三相无刷电机驱动演示程序，电位器调速，捕捉中断换相.....1420
30 高级 PWM-硬件移相.....1428
30.1 高级 PWM 硬件移相功能脚切换.....1428
30.2 相关寄存器.....1429
30.2.1 输出使能寄存器 2（PWMA_ENO2）.....1430
30.2.2 输出附加使能寄存器 2（PWMA_IOAUX2）.....1430
30.2.3 控制寄存器 3（PWMA_CR3）.....1431
30.2.4 状态寄存器 3(PWMA_SR3).....1432
30.2.5 捕获/比较使能寄存器 3（PWMA_CCER3）.....1432
30.2.6 捕获/比较模式扩展寄存器 1（PWMA_CCMR1X）.....1433
30.2.7 捕获/比较模式扩展寄存器 2（PWMA_CCMR2X）.....1434
30.2.8 捕获/比较模式扩展寄存器 3（PWMA_CCMR3X）.....1434

30.2.9 捕获/比较模式扩展寄存器 4（PWMA_CCMR4X）.....1434
30.2.10 捕获/比较模式寄存器 5（PWMx_CCMR5）.....1435
30.2.11 捕获/比较模式扩展寄存器 5（PWMA_CCMR5X）.....1435
30.2.12 捕获/比较模式寄存器 6（PWMx_CCMR6）.....1435
30.2.13 捕获/比较模式扩展寄存器 6（PWMA_CCMR6X）.....1435
30.2.14 捕获/比较寄存器 5 高 8 位（PWMA_CCR5H）.....1436
30.2.15 捕获/比较寄存器 5 低 8 位（PWMA_CCR5L）.....1436
30.2.16 捕获/比较扩展寄存器 5（PWMA_CCR5X）.....1436
30.2.17 捕获/比较寄存器 6 高 8 位（PWMA_CCR6H）.....1437
30.2.18 捕获/比较寄存器 6 低 8 位（PWMA_CCR6L）.....1437
30.3 移相 PWM 输出模式.....1438
30.3.1 不对称 PWM 模式.....1438
30.3.2 组合 PWM 模式.....1439
30.3.3 组合三相 PWM 模式.....1440
30.4 PWM 硬件移相范例程序.....1441
30.5 利用不对称 PWM 实现高速正交编码信号输出 （热心网友冲哥提供）.....1441
31 高速高级 PWM（HSPWM），可以使用 PLL 高速时钟作为时钟源.....1450
31.1 相关寄存器.....1450
31.1.1 HSPWM 配置寄存器（HSPWMn_CFG）.....1451
31.1.2 HSPWM 地址寄存器（HSPWMn_AD）.....1451
31.1.3 HSPWM 数据寄存器（HSPWMn_DAT）.....1452
31.2 高速高级 PWM + DMA（组合使用）特别注意事项.....1453
31.3 范例程序.....1454
31.3.1 使能高级 PWM 的高速模式（异步模式）.....1454
32 USB 2.0-FS 通用串行总线.....1457
32.1 USB 相关的寄存器.....1458
32.1.1 USB 控制寄存器（USBCON）.....1458
32.1.2 USB 时钟控制寄存器（USBCLK）.....1459
32.1.3 USB 间址地址寄存器（USBADR）.....1460
32.1.4 USB 间址数据寄存器（USBDAT）.....1460
32.2 USB 控制器寄存器（SIE）.....1461
32.2.1 USB 功能地址寄存器（FADDR）.....1462
32.2.2 USB 电源控制寄存器（POWER）.....1462
32.2.3 USB 端点 IN 中断标志位（INTRIN1）.....1463
32.2.4 USB 端点 OUT 中断标志位（INTROUT1）.....1463
32.2.5 USB 电源中断标志（INTRUSB）.....1464
32.2.6 USB 端点 IN 中断允许寄存器（INTRIN1E）.....1464
32.2.7 USB 端点 OUT 中断允许寄存器（INTROUT1E）.....1465
32.2.8 USB 电源中断允许寄存器（INTRUSBE）.....1465
32.2.9 USB 数据帧号寄存器（FRAMEn）.....1466
32.2.10 USB 端点索引寄存器（INDEX）.....1466
32.2.11 IN 端点的最大数据包大小（INMAXP）.....1466
32.2.12 USB 端点 0 控制状态寄存器（CSR0）.....1467
32.2.13 IN 端点控制状态寄存器 1（INCSR1）.....1468

32.2.14 IN 端点控制状态寄存器 2（INCSR2）.....1469
32.2.15 OUT 端点的最大数据包大小（OUTMAXP）.....1469
32.2.16 OUT 端点控制状态寄存器 1（OUTCSR1）.....1470
32.2.17 OUT 端点控制状态寄存器 2（OUTCSR2）.....1471
32.2.18 USB 端点 0 的 OUT 长度（COUNT0）.....1471
32.2.19 USB 端点的 OUT 长度（OUTCOUNTn）.....1471
32.2.20 USB 端点的 FIFO 数据访问寄存器（FIFOn）.....1472
32.2.21 USB 跟踪控制寄存器（UTRKCTL）.....1472
32.2.22 USB 跟踪状态寄存器（UTRKSTS）.....1473
32.3 USB 产品开发注意事项.....1474
32.4 如何软复位到系统区启动【USB 直接下载，不管 P3.2】.....1475
32.5 范例程序.....1476
32.5.1 HID 人机接口设备范例.....1476
32.5.2 HID(Human Interface Device)协议范例.....1487
32.5.3 CDC(Communication Device Class)协议范例.....1487
32.5.4 基于 HID 协议的 USB 键盘范例.....1487
32.5.5 基于 HID 协议的 USB 鼠标范例.....1487
32.5.6 基于 HID 协议的 USB 鼠标+键盘二合一范例.....1488
32.5.7 基于 WINUSB 协议的范例.....1488
32.5.8 MSC(Mass Storage Class)协议范例.....1488
33 TFT 彩屏接口接口（8/16 位 i8080/M6800 接口）.....1489
33.1 TFT 彩屏接口功能脚切换.....1489
33.2 TFT 彩屏相关的寄存器.....1490
33.2.1 TFT 彩屏接口配置寄存器（LCMIFCFG）.....1490
33.2.2 TFT 彩屏接口配置寄存器 2（LCMIFCFG2）.....1491
33.2.3 TFT 彩屏接口控制寄存器（LCMIFCR）.....1491
33.2.4 TFT 彩屏接口状态寄存器（LCMIFSTA）.....1491
33.2.5 TFT 彩屏接口数据寄存器（LCMIFDATL，LCMIFDATH）.....1491
33.2.6 TFT 彩屏接口时钟预分频（LCMIFPSCR）.....1492
33.3 i8080/M6800 模式 TFT 彩屏接口时序图.....1493
33.3.1 i8080 模式.....1493
33.3.2 M6800 模式.....1494
34 DMA，支持解放 CPU 的外设到外设传输（P2P）.....1495
34.1 外设到外设（P2P）DMA 矩阵.....1496
34.2 存储器与存储器之间的数据读写（M2M_DMA）.....1497
34.2.1 相关的寄存器.....1497
34.2.2 M2M_DMA 配置寄存器（DMA_M2M_CFG）.....1498
34.2.3 M2M_DMA 控制寄存器（DMA_M2M_CR）.....1498
34.2.4 M2M_DMA 状态寄存器（DMA_M2M_STA）.....1499
34.2.5 M2M_DMA 传输总字节寄存器（DMA_M2M_AMT）.....1499
34.2.6 M2M_DMA 传输完成字节寄存器（DMA_M2M_DONE）.....1499
34.2.7 M2M_DMA 发送地址寄存器（DMA_M2M_TXAx）.....1499
34.2.8 M2M_DMA 接收地址寄存器（DMA_M2M_RXAx）.....1499
34.3 ADC 数据自动存储（ADC_DMA）.....1500

34.3.1 相关的寄存器 1500
34.3.2 ADC_DMA 配置寄存器（DMA_ADC_CFG） 1501
34.3.3 ADC_DMA 控制寄存器（DMA_ADC_CR） 1501
34.3.4 ADC_DMA 状态寄存器（DMA_ADC_STA） 1501
34.3.5 ADC_DMA 循环扫描总次数寄存器（DMA_ADC_AMT） 1502
34.3.6 ADC_DMA 完成扫描次数寄存器（DMA_ADC_DONE） 1502
34.3.7 ADC_DMA 接收地址寄存器（DMA_ADC_RXAx） 1502
34.3.8 ADC_DMA 配置寄存器 2（DMA_ADC_CFG2） 1502
34.3.9 ADC_DMA 通道使能寄存器（DMA_ADC_CHSWx） 1502
34.3.10 ADC_DMA 转换周期（DMA_ADC_ITVx） 1503
34.3.11 ADC_DMA 的数据存储格式 1504
34.4 SPI 与存储器之间的数据交换（SPI_DMA） 1506
34.4.1 相关的寄存器 1506
34.4.2 SPI_DMA 配置寄存器（DMA_SPI_CFG） 1507
34.4.3 SPI_DMA 控制寄存器（DMA_SPI_CR） 1508
34.4.4 SPI_DMA 状态寄存器（DMA_SPI_STA） 1508
34.4.5 SPI_DMA 传输总字节寄存器（DMA_SPI_AMT） 1508
34.4.6 SPI_DMA 传输完成字节寄存器（DMA_SPI_DONE） 1509
34.4.7 SPI_DMA 发送地址寄存器（DMA_SPI_TXAx） 1509
34.4.8 SPI_DMA 接收地址寄存器（DMA_SPI_RXAx） 1509
34.4.9 SPI_DMA 配置寄存器 2（DMA_SPI_CFG2） 1509
34.4.10 SPI_DMA 传输周期（DMA_SPI_ITVx） 1509
34.5 串口 1 与存储器之间的数据交换（UR1T_DMA，UR1R_DMA） 1510
34.5.1 相关的寄存器 1510
34.5.2 UR1T_DMA 配置寄存器（DMA_UR1T_CFG） 1511
34.5.3 UR1T_DMA 控制寄存器（DMA_UR1T_CR） 1511
34.5.4 UR1T_DMA 状态寄存器（DMA_UR1T_STA） 1512
34.5.5 UR1T_DMA 传输总字节寄存器（DMA_UR1T_AMT） 1512
34.5.6 UR1T_DMA 传输完成字节寄存器（DMA_UR1T_DONE） 1512
34.5.7 UR1T_DMA 发送地址寄存器（DMA_UR1T_TXAx） 1512
34.5.8 UR1_DMA 传输周期（DMA_UR1_ITVx） 1512
34.5.9 UR1R_DMA 配置寄存器（DMA_UR1R_CFG） 1513
34.5.10 UR1R_DMA 控制寄存器（DMA_UR1R_CR） 1513
34.5.11 UR1R_DMA 状态寄存器（DMA_UR1R_STA） 1514
34.5.12 UR1R_DMA 传输总字节寄存器（DMA_UR1R_AMT） 1514
34.5.13 UR1R_DMA 传输完成字节寄存器（DMA_UR1R_DONE） 1514
34.5.14 UR1R_DMA 接收地址寄存器（DMA_UR1R_RXAx） 1514
34.6 串口 2 与存储器之间的数据交换（UR2T_DMA，UR2R_DMA） 1515
34.6.1 相关的寄存器 1515
34.6.2 UR2T_DMA 配置寄存器（DMA_UR2T_CFG） 1516
34.6.3 UR2T_DMA 控制寄存器（DMA_UR2T_CR） 1516
34.6.4 UR2T_DMA 状态寄存器（DMA_UR2T_STA） 1517
34.6.5 UR2T_DMA 传输总字节寄存器（DMA_UR2T_AMT） 1517
34.6.6 UR2T_DMA 传输完成字节寄存器（DMA_UR2T_DONE） 1517

34.6.7 UR2T_DMA 发送地址寄存器（DMA_UR2T_TXAx）.....1517
34.6.8 UR2_DMA 传输周期（DMA_UR2_ITVx）.....1517
34.6.9 UR2R_DMA 配置寄存器（DMA_UR2R_CFG）.....1518
34.6.10 UR2R_DMA 控制寄存器（DMA_UR2R_CR）.....1518
34.6.11 UR2R_DMA 状态寄存器（DMA_UR2R_STA）.....1519
34.6.12 UR2R_DMA 传输总字节寄存器（DMA_UR2R_AMT）.....1519
34.6.13 UR2R_DMA 传输完成字节寄存器（DMA_UR2R_DONE）.....1519
34.6.14 UR2R_DMA 接收地址寄存器（DMA_UR2R_RXAx）.....1519
34.7 串口 3 与存储器之间的数据交换（UR3T_DMA，UR3R_DMA）.....1520
34.7.1 相关的寄存器.....1520
34.7.2 UR3T_DMA 配置寄存器（DMA_UR3T_CFG）.....1521
34.7.3 UR3T_DMA 控制寄存器（DMA_UR3T_CR）.....1521
34.7.4 UR3T_DMA 状态寄存器（DMA_UR3T_STA）.....1522
34.7.5 UR3T_DMA 传输总字节寄存器（DMA_UR3T_AMT）.....1522
34.7.6 UR3T_DMA 传输完成字节寄存器（DMA_UR3T_DONE）.....1522
34.7.7 UR3T_DMA 发送地址寄存器（DMA_UR3T_TXAx）.....1522
34.7.8 UR3_DMA 传输周期（DMA_UR3_ITVx）.....1522
34.7.9 UR3R_DMA 配置寄存器（DMA_UR3R_CFG）.....1523
34.7.10 UR3R_DMA 控制寄存器（DMA_UR3R_CR）.....1523
34.7.11 UR3R_DMA 状态寄存器（DMA_UR3R_STA）.....1524
34.7.12 UR3R_DMA 传输总字节寄存器（DMA_UR3R_AMT）.....1524
34.7.13 UR3R_DMA 传输完成字节寄存器（DMA_UR3R_DONE）.....1524
34.7.14 UR3R_DMA 接收地址寄存器（DMA_UR3R_RXAx）.....1524
34.8 串口 4 与存储器之间的数据交换（UR4T_DMA，UR4R_DMA）.....1525
34.8.1 相关的寄存器.....1525
34.8.2 UR4T_DMA 配置寄存器（DMA_UR4T_CFG）.....1526
34.8.3 UR4T_DMA 控制寄存器（DMA_UR4T_CR）.....1526
34.8.4 UR4T_DMA 状态寄存器（DMA_UR4T_STA）.....1527
34.8.5 UR4T_DMA 传输总字节寄存器（DMA_UR4T_AMT）.....1527
34.8.6 UR4T_DMA 传输完成字节寄存器（DMA_UR4T_DONE）.....1527
34.8.7 UR4T_DMA 发送地址寄存器（DMA_UR4T_TXAx）.....1527
34.8.8 UR4_DMA 传输周期（DMA_UR4_ITVx）.....1527
34.8.9 UR4R_DMA 配置寄存器（DMA_UR4R_CFG）.....1528
34.8.10 UR4R_DMA 控制寄存器（DMA_UR4R_CR）.....1528
34.8.11 UR4R_DMA 状态寄存器（DMA_UR4R_STA）.....1529
34.8.12 UR4R_DMA 传输总字节寄存器（DMA_UR4R_AMT）.....1529
34.8.13 UR4R_DMA 传输完成字节寄存器（DMA_UR4R_DONE）.....1529
34.8.14 UR4R_DMA 接收地址寄存器（DMA_UR4R_RXAx）.....1529
34.9 TFT 彩屏与存储器之间的数据读写（LCM_DMA）.....1530
34.9.1 相关的寄存器.....1530
34.9.2 TFT 彩屏 DMA 配置寄存器（DMA_LCM_CFG）.....1531
34.9.3 TFT 彩屏 DMA 控制寄存器（DMA_LCM_CR）.....1531
34.9.4 TFT 彩屏 DMA 状态寄存器（DMA_LCM_STA）.....1532
34.9.5 TFT 彩屏 DMA 传输总字节寄存器（DMA_LCM_AMT）.....1532

34.9.6 TFT 彩屏 DMA 传输完成字节寄存器（DMA_LCM_DONE）.....1532
34.9.7 TFT 彩屏 DMA 发送地址寄存器（DMA_LCM_TXAx）.....1532
34.9.8 TFT 彩屏 DMA 接收地址寄存器（DMA_LCM_RXAx）.....1532
34.9.9 TFT 彩屏 DMA 传输周期（DMA_LCM_ITVx）.....1533
34.10 I2C 与存储器之间的数据交换（I2CT_DMA，I2CR_DMA）.....1534
34.10.1 相关的寄存器.....1534
34.10.2 I2CT_DMA 配置寄存器（DMA_I2CT_CFG）.....1535
34.10.3 I2CT_DMA 控制寄存器（DMA_I2CT_CR）.....1535
34.10.4 I2CT_DMA 状态寄存器（DMA_I2CT_STA）.....1535
34.10.5 I2CT_DMA 传输总字节寄存器（DMA_I2CT_AMT）.....1536
34.10.6 I2CT_DMA 传输完成字节寄存器（DMA_I2CT_DONE）.....1536
34.10.7 I2CT_DMA 发送地址寄存器（DMA_I2CT_TXAx）.....1536
34.10.8 I2C_DMA 传输周期（DMA_I2C_ITVx）.....1536
34.10.9 I2CR_DMA 配置寄存器（DMA_I2CR_CFG）.....1537
34.10.10 I2CR_DMA 控制寄存器（DMA_I2CR_CR）.....1538
34.10.11 I2CR_DMA 状态寄存器（DMA_I2CR_STA）.....1538
34.10.12 I2CR_DMA 传输总字节寄存器（DMA_I2CR_AMT）.....1538
34.10.13 I2CR_DMA 传输完成字节寄存器（DMA_I2CR_DONE）.....1538
34.10.14 I2CR_DMA 接收地址寄存器（DMA_I2CR_RXAx）.....1539
34.10.15 I2C_DMA 控制寄存器（DMA_I2C_CR）.....1539
34.10.16 I2C_DMA 状态寄存器（DMA_I2C_ST）.....1539
34.11 I2S 与存储器之间的数据交换（I2ST_DMA，I2SR_DMA）.....1540
34.11.1 相关的寄存器.....1540
34.11.2 I2ST_DMA 配置寄存器（DMA_I2ST_CFG）.....1541
34.11.3 I2ST_DMA 控制寄存器（DMA_I2ST_CR）.....1541
34.11.4 I2ST_DMA 状态寄存器（DMA_I2ST_STA）.....1541
34.11.5 I2ST_DMA 传输总字节寄存器（DMA_I2ST_AMT）.....1542
34.11.6 I2ST_DMA 传输完成字节寄存器（DMA_I2ST_DONE）.....1542
34.11.7 I2ST_DMA 发送地址寄存器（DMA_I2ST_TXAx）.....1542
34.11.8 I2SR_DMA 配置寄存器（DMA_I2SR_CFG）.....1542
34.11.9 I2S_DMA 传输周期（DMA_I2S_ITVx）.....1543
34.11.10 I2SR_DMA 控制寄存器（DMA_I2SR_CR）.....1543
34.11.11 I2SR_DMA 状态寄存器（DMA_I2SR_STA）.....1543
34.11.12 I2SR_DMA 传输总字节寄存器（DMA_I2SR_AMT）.....1543
34.11.13 I2SR_DMA 传输完成字节寄存器（DMA_I2SR_DONE）.....1544
34.11.14 I2SR_DMA 接收地址寄存器（DMA_I2SR_RXAx）.....1544
34.12 QSPI 与存储器之间的数据交换（QSPI_DMA）.....1545
34.12.1 相关的寄存器.....1545
34.12.2 QSPI_DMA 配置寄存器（DMA_QSPI_CFG）.....1546
34.12.3 QSPI_DMA 控制寄存器（DMA_QSPI_CR）.....1547
34.12.4 QSPI_DMA 状态寄存器（DMA_QSPI_STA）.....1547
34.12.5 QSPI_DMA 传输总字节寄存器（DMA_QSPI_AMT）.....1547
34.12.6 QSPI_DMA 传输完成字节寄存器（DMA_QSPI_DONE）.....1548
34.12.7 QSPI_DMA 发送地址寄存器（DMA_QSPI_TXAx）.....1548

34.12.8 QSPI_DMA 接收地址寄存器（DMA_QSPI_RXAx）.....1548
34.12.9 QSPI_DMA 传输周期（DMA_QSPI_ITVx）.....1548
34.13 PWMA 与存储器之间的数据交换（PWMAT_DMA，PWMAR_DMA）.....1549
34.13.1 相关的寄存器.....1549
34.13.2 PWMAT_DMA 配置寄存器（DMA_PWMAT_CFG）.....1550
34.13.3 PWMAT_DMA 控制寄存器（DMA_PWMAT_CR）.....1550
34.13.4 PWMAT_DMA 状态寄存器（DMA_PWMAT_STA）.....1551
34.13.5 PWMAT_DMA 传输总字节寄存器（DMA_PWMAT_AMT）.....1551
34.13.6 PWMAT_DMA 传输完成字节寄存器（DMA_PWMAT_DONE）.....1551
34.13.7 PWMAT_DMA 发送地址寄存器（DMA_PWMAT_TXAx）.....1551
34.13.8 PWMA_DMA 传输周期（DMA_PWMA_ITVx）.....1552
34.13.9 PWMAR_DMA 配置寄存器（DMA_PWMAR_CFG）.....1552
34.13.10 PWMAR_DMA 控制寄存器（DMA_PWMAR_CR）.....1552
34.13.11 PWMAR_DMA 状态寄存器（DMA_PWMAR_STA）.....1553
34.13.12 PWMAR_DMA 传输总字节寄存器（DMA_PWMAR_AMT）.....1553
34.13.13 PWMAR_DMA 传输完成字节寄存器（DMA_PWMAR_DONE）.....1553
34.13.14 PWMAR_DMA 接收地址寄存器（DMA_PWMAR_RXAx）.....1553
34.13.15 PWMA_DMA 事件使能寄存器（PWMA_DER）.....1554
34.13.16 PWMA_DMA 基址寄存器（PWMA_DBA）.....1554
34.13.17 PWMA_DMA 突发传输次数寄存器（PWMA_DBL）.....1555
34.13.18 PWMA_DMA 控制寄存器（PWMA_DMACR）.....1555
34.14 外设与外设之间的数据交换（P2P_DMA）.....1557
34.14.1 相关的寄存器.....1557
34.14.2 P2P_DMA 配置寄存器 1（DMA_P2P_CR1）.....1557
34.14.3 P2P_DMA 配置寄存器 2（DMA_P2P_CR2）.....1559
34.15 范例程序.....1560
34.15.1 串口 1 中断模式与电脑收发测试 - DMA 接收超时中断.....1560
34.15.2 串口 1 中断模式与电脑收发测试 - DMA 数据校验.....1565
34.15.3 使用 SPI_DMA+TFT 彩屏 DMA 双缓冲对 TFT 刷屏.....1573
34.15.4 实战硬件 SPI+DMA 驱动 WS2812 彩灯.....1581
35 I2S 音频总线，可以使用 PLL 高速时钟作为时钟源.....1585
35.1 I2S 功能脚切换.....1585
35.2 I2S 相关的寄存器.....1585
35.2.1 I2S 控制寄存器（I2SCR）.....1586
35.2.2 I2S 状态寄存器（I2SSR）.....1586
35.2.3 I2S 数据寄存器（I2SDRH、I2SDRL）.....1587
35.2.4 I2S 分频寄存器高字节（I2SPRH）.....1587
35.2.5 I2S 分频寄存器低字节（I2SPRL）.....1587
35.2.6 I2S 配置寄存器高字节（I2SCFGH）.....1588
35.2.7 I2S 分频寄存器低字节（I2SCFGL）.....1588
35.2.8 I2S 从模式控制寄存器（I2SMD）.....1589
35.2.9 I2S 主时钟分频寄存器（I2SMCKDIV）.....1589
35.3 范例程序.....1590
35.3.1 输出 2 路三角波.....1590

35.3.2 输出 2 路正弦波 1593
36 32 位硬件乘除单元（MDU32） 1596
36.1 相关的特殊功能寄存器 1597
36.2 运算执行时间表 1597
36.3 MDU32 算术运算 1598
36.3.1 32 位乘法 1598
36.3.2 32 位无符号除法 1598
36.3.3 32 位有符号除法 1598
36.4 范例程序 1599
37 TFPU（三角函数+单精度浮点运算器），可以使用 PLL 高速时钟作为时钟源 1602
37.1 TFPU 浮点运算器简介 1602
37.2 相关的特殊功能寄存器 1602
37.3 运算执行时间表 1603
37.4 TFPU 基本算术运算 1604
37.4.1 浮点数加法（+） 1604
37.4.2 浮点数减法（－） 1604
37.4.3 浮点数乘法（×） 1604
37.4.4 浮点数除法（÷） 1605
37.4.5 浮点数开方/平方根（sqrt） 1605
37.4.6 浮点数比较（comp） 1605
37.4.7 浮点数检测（check） 1606
37.5 TFPU 三角函数 1607
37.5.1 正弦函数（sin） 1607
37.5.2 余弦函数（cos） 1607
37.5.3 正切函数（tan） 1607
37.5.4 反正切函数（arctan） 1607
37.6 TFPU 数据转换操作 1608
37.6.1 浮点数转 8 位整数（float → char） 1608
37.6.2 浮点数转 16 位整数（float → short） 1608
37.6.3 浮点数转 32 位整数（float → long） 1608
37.6.4 8 位整数转浮点数（char → float） 1608
37.6.5 16 位整数转浮点数（short → float） 1609
37.6.6 32 位整数转浮点数（long → float） 1609
37.7 TFPU 协处理器控制操作 1610
37.7.1 初始化协处理器 1610
37.7.2 清除异常 1610
37.7.3 读状态寄存器 1610
37.7.4 写状态寄存器 1610
37.7.5 读控制寄存器 1610
37.7.6 写控制寄存器 1610
37.8 如何选择 TFPU 的时钟源 1611
37.8.1 选择系统时钟（和 CPU 时钟同步）作为 TFPU 时钟源 1611
37.8.2 选择 PLL 时钟（和 CPU 时钟异步）作为 TFPU 时钟源 1611
37.9 范例程序 1612

38 DPU32（DSP 指令，32 位及 64 位整数运算器）.....1614
38.1 相关的特殊功能寄存器.....1614
38.2 DPU32 操作寄存器说明.....1615
38.3 运算执行时间表.....1616
38.4 DPU32 指令说明.....1619
附录 A 指令集.....1640
A.1 指令集简介.....1640
A.1.1 BINARY 模式和 SOURCE 模式.....1640
A.1.2 指令集标记.....1640
A.1.3 指令表（功能排序）.....1642
A.1.4 指令表（机器码排序）.....1648
A.2 指令详解.....1652
A.3 多级流水线内核的中断响应.....1734
附录 B Ai8051U 系列和 Intel-80251 内部结构图对比.....1736
附录 C 应用注意事项.....1738
C.1 关于 Ai8051U 系列的注意事项.....1738
附录 D Ai8051U 系列单片机取代 STC89 系列的注意事项.....1739
附录 E Ai8051U 系列单片机取代 STC12C5A60S2 系列的注意事项.....1741
附录 F 7.4.12 固件版本说明.....1743
附录 G 电气特性.....1744
附录 H STCAI Reliability Report V1.2.....1748
附录 I 单片机电源系统最简易自我保护电路.....1750
附录 J 关于回流焊前是否要烘烤.....1751
附录 K 如何使用万用表检测芯片 I/O 口好坏.....1752
附录 L 大批量生产，如何省去专门的烧录人员，如何无烧录环节.....1753
附录 M 如何使用 ISP 下载软件制作和编辑 EEPROM 文件.....1754
附录 N 开源汇编语言调用 USB-CDC 库文件实现 USB-CDC 虚拟串口通信.....1755
附录 O 可以自己去生产的【USB 转双串口】资料.....1756
附录 P 串口中断收发－MODBUS 协议.....1757
附录 Q 如何通过读取寄存器获取芯片版本.....1767
附录 R 单总线（1-Wire）.....1768
R.1 单总线工作模式.....1768
R.1.1 WS2812 模式.....1768
R.1.2 DS18B20 模式.....1768
R.1.3 DHT11 模式.....1769
R.1.4 红外模式.....1770
R.1.5 UARTX 模式.....1770
R.1.6 UART 模式.....1771
R.2 单总线相关的寄存器.....1772
R.3 寄存器说明.....1773
R.3.1 单总线控制寄存器（OWCR）.....1773
R.3.2 单总线中断寄存器（OWIR）.....1774
R.3.3 单总线状态寄存器（OWSR）.....1774
R.3.4 单总线数据寄存器（OWDR）.....1775

R.3.5 单总线时钟预分频寄存器（OWPSCR）.....1775
R.3.6 单总线模式寄存器（OWMR）.....1775
R.3.7 单总线 FIFO 控制/状态寄存器（OWFCSR）.....1776
R.3.8 单总线配置寄存器（OWCFGn）.....1776
R.4 单总线 DMA 相关的寄存器.....1780
R.5 单总线 DMA 寄存器说明.....1781
R.5.1 OWT_DMA 配置寄存器（DMA_OWT_CFG）.....1781
R.5.2 OWT_DMA 控制寄存器（DMA_OWT_CR）.....1781
R.5.3 OWT_DMA 状态寄存器（DMA_OWT_STA）.....1781
R.5.4 OWT_DMA 传输总字节寄存器（DMA_OWT_AMT）.....1782
R.5.5 OWT_DMA 传输完成字节寄存器（DMA_OWT_DONE）.....1782
R.5.6 OWT_DMA 发送地址寄存器（DMA_OWT_TXAx）.....1782
R.5.7 OWR_DMA 配置寄存器（DMA_OWR_CFG）.....1782
R.5.8 OWR_DMA 控制寄存器（DMA_OWR_CR）.....1783
R.5.9 OWR_DMA 状态寄存器（DMA_OWR_STA）.....1783
R.5.10 OWR_DMA 传输总字节寄存器（DMA_OWR_AMT）.....1783
R.5.11 OWR_DMA 传输完成字节寄存器（DMA_OWR_DONE）.....1783
R.5.12 OWR_DMA 接收地址寄存器（DMA_OWR_RXAx）.....1783
附录 S USB 原理及应用线上培训教程（何老师）.....1784
S.1 USB 协议概述.....1784
S.1.1 USB 通信基础.....1784
S.1.2 通信的时域构成.....1789
S.1.3 USB 通信模型.....1793
S.1.4 USB 标准请求.....1799
S.1.5 USB 通信过程.....1804
S.1.6 USB 描述符.....1805
S.2 USB2.0 程序设计实现.....1811
S.2.1 USB 的寄存器操作和数据传输.....1811
S.2.2 USB SETUP 阶段实现.....1815
S.3 人机交互设备原理.....1827
S.3.1 HID 类设备简介.....1827
S.3.2 主机请求.....1827
S.3.3 描述符.....1832
S.3.4 报告.....1849
S.3.5 HID 协议小结.....1852
S.4 人机交互设备程序设计.....1857
S.4.1 设计分层.....1857
S.4.2 描述符数据结构.....1861
S.4.3 例程使用说明.....1862
S.5 通信设备类原理.....1863
S.5.1 USB 通信类设备简介.....1863
S.5.2 USB 通信类设备抽象模型介绍.....1867
S.5.3 USB 通信类设备类特定码.....1869
S.5.4 USB 通信类设备描述符.....1871

S.5.5 USB 通信类设备主机请求以及设备通知 1876
S.6 通信设备类程序设计 1884
S.6.1 CDC 程序设计原理(缺少 LINECODING 数据结构的说明) 1884
附录 T 关于“深大--定时器 0/1，24 位定时器实验”的定时器解释 1891
附录 U 设置屏幕显示为对比鲜明的黑底白字 1894
U.1 Win7 操作系统： 1894
U.2 Win11 操作系统： 1895
附录 V Windows 系统页面字体放大功能 1897
附录 W 更新记录 1898
附录 X 自主产权，生产可控 3

1　单片机基础概述

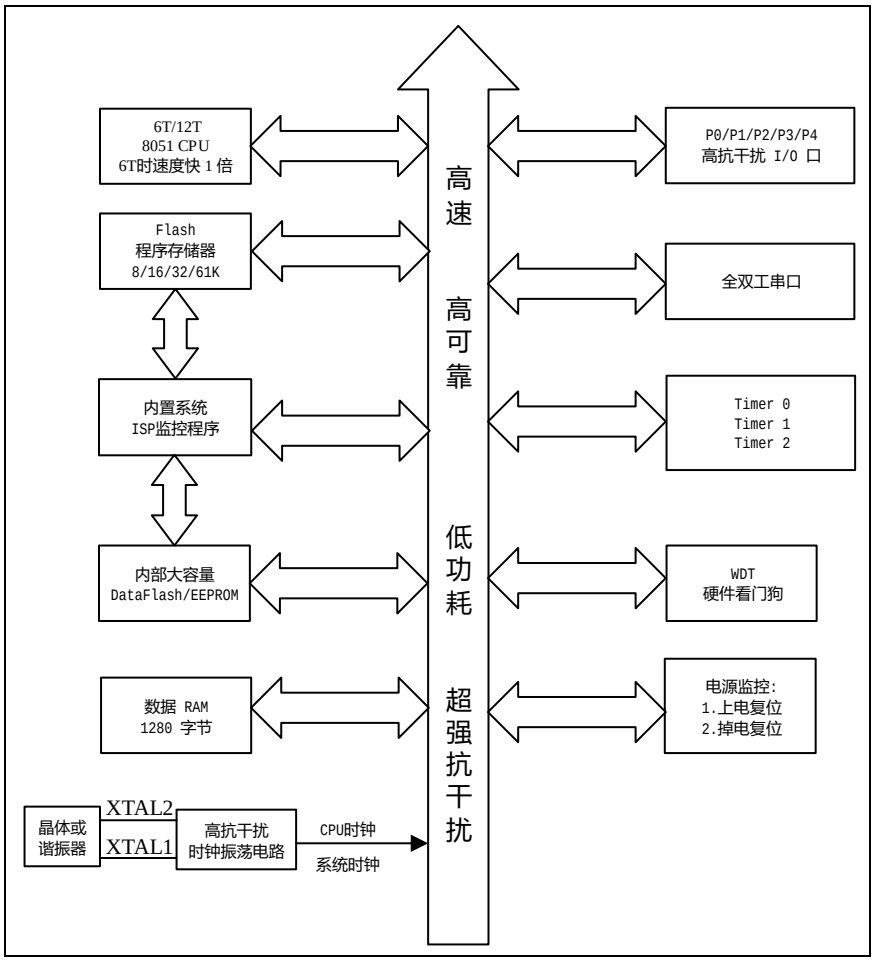
——无微机原理的用户请从本章开始学习

这一章主要讲述的内容有：①在数字设备中进行算术运算的基本知识——数制和编码；②数字电路中一些常用逻辑运算及其图形符号。它们是学习单片机这门课程的基础。对于没有微机原理基础的用户和同学，请从这章开始学习。

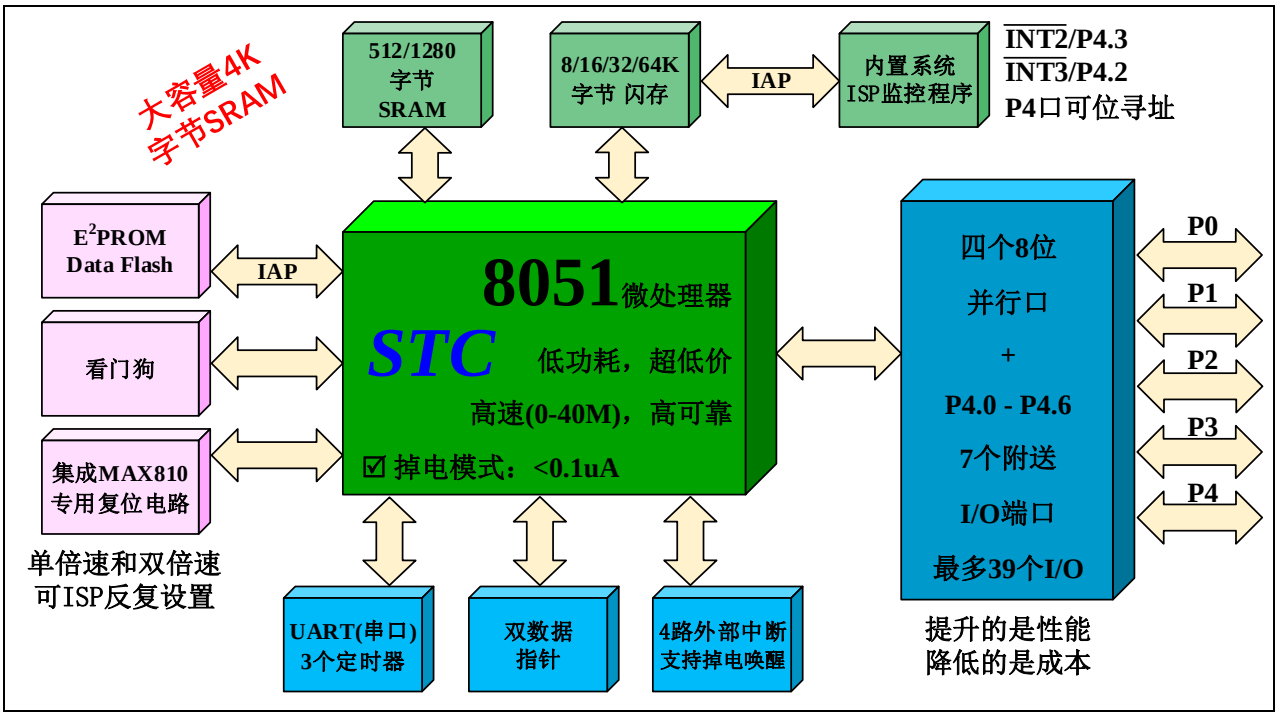
1.1　什么是单片机/微控制器

在一个芯片上实现一个完整的计算机系统就是单片微型计算机，简称单片机，俗称 MCU/微控制器。当然，有些有机械件的外围，由于材料的问题无法集成，如键盘，如显示器。

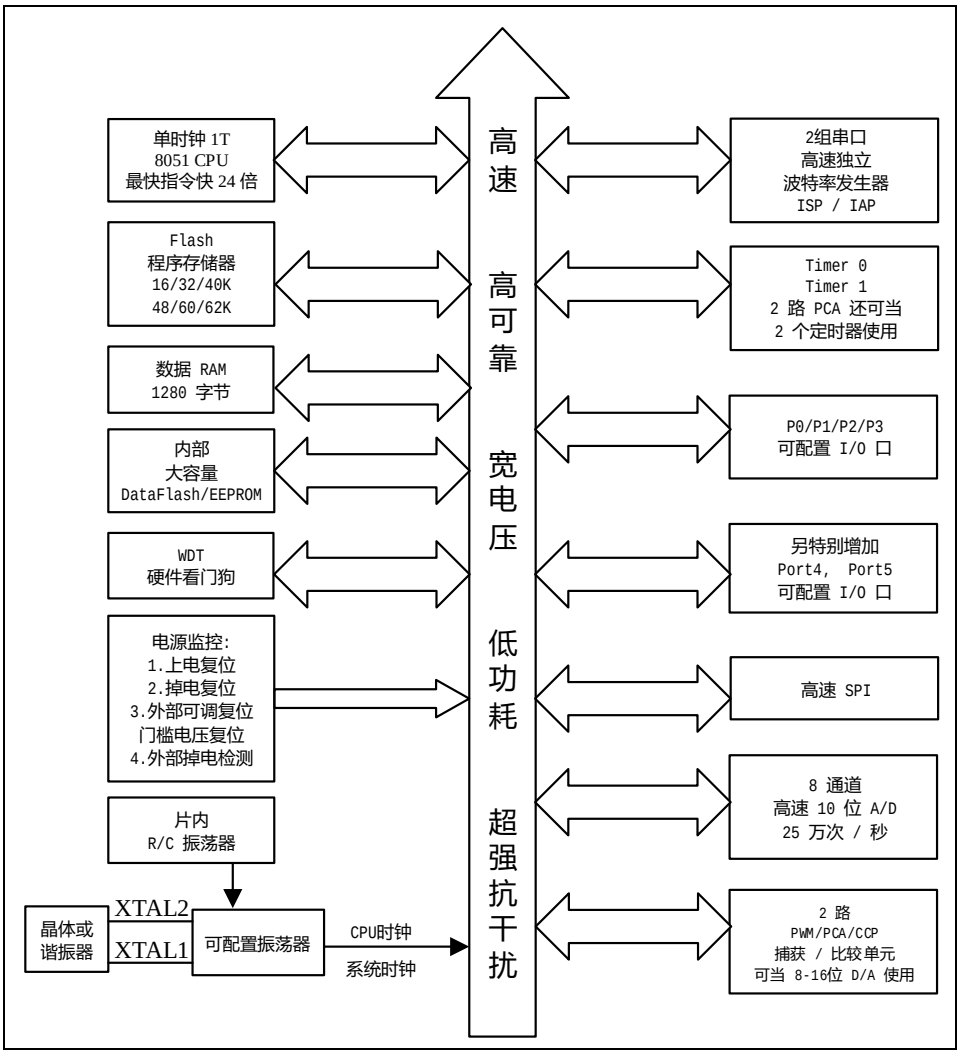
1.1.1　经典单片机 89C52RC/89C58RD+系列组成框图



更通俗的框图：


大容量4K
字节SRAM
512/1280
字节
SRAM
8/16/32/64K
字节 闪存
IAP
内置系统
ISP监控程序
INT2/P4.3
INT3/P4.2
P4口可位寻址
E2PROM
Data Flash
IAP
看门狗
集成MAX810
专用复位电路
8051微处理器
STC
低功耗，超低价
高速(0-40M)，高可靠
☑ 掉电模式：<0.1uA
四个8位
并行口
+
P4.0 - P4.6
7个附送
I/O端口
最多39个I/O
P0
P1
P2
P3
P4
单倍速和双倍速
可ISP反复设置
UART(串口)
3个定时器
双数据
指针
4路外部中断
支持掉电唤醒
提升的是性能
降低的是成本

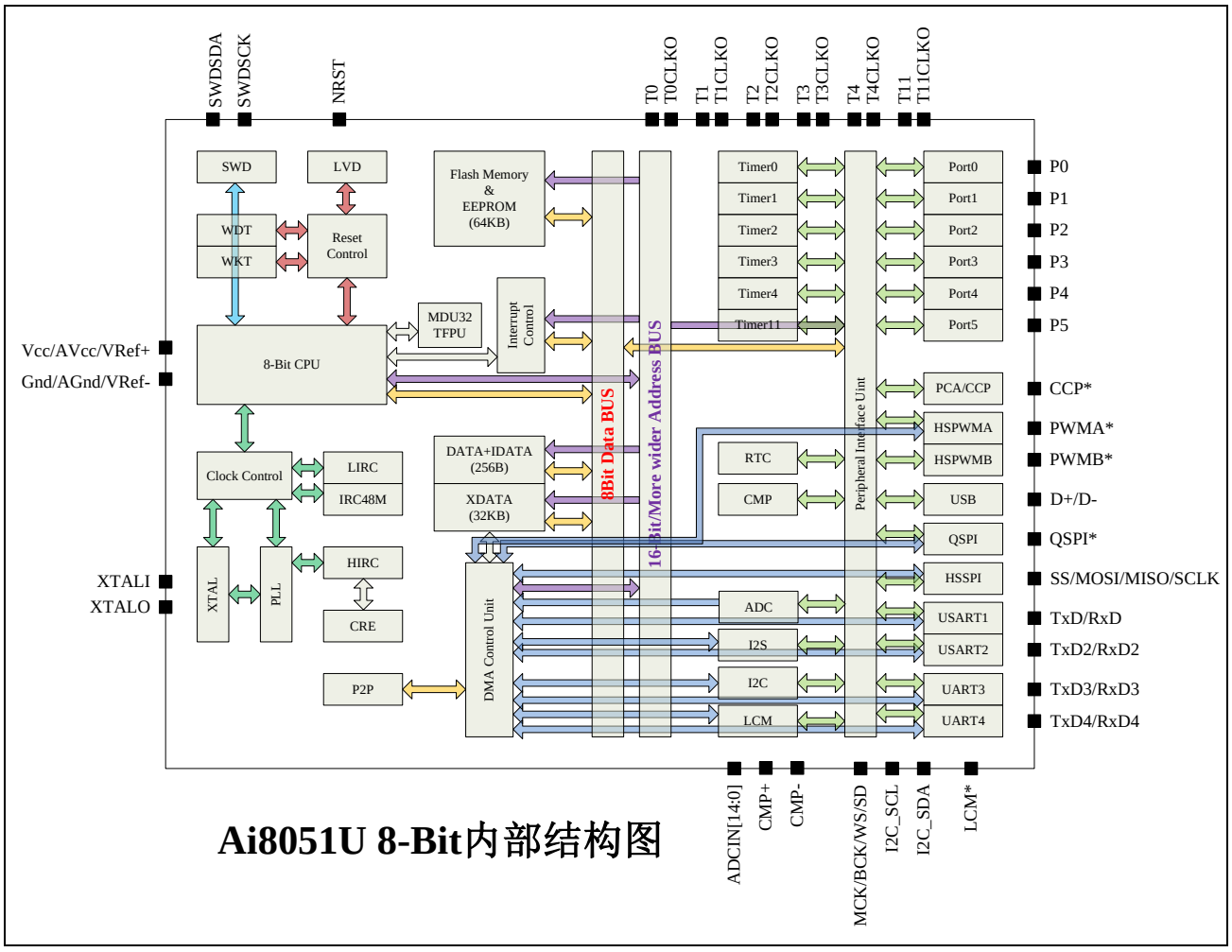
1.1.2　经典单片机 12C5A60S2 组成框图



1.1.3　Ai8051U 内部结构图

Ai8051U 系列内部的数据总线宽度是 32 位（兼容 16/8/1 位）。

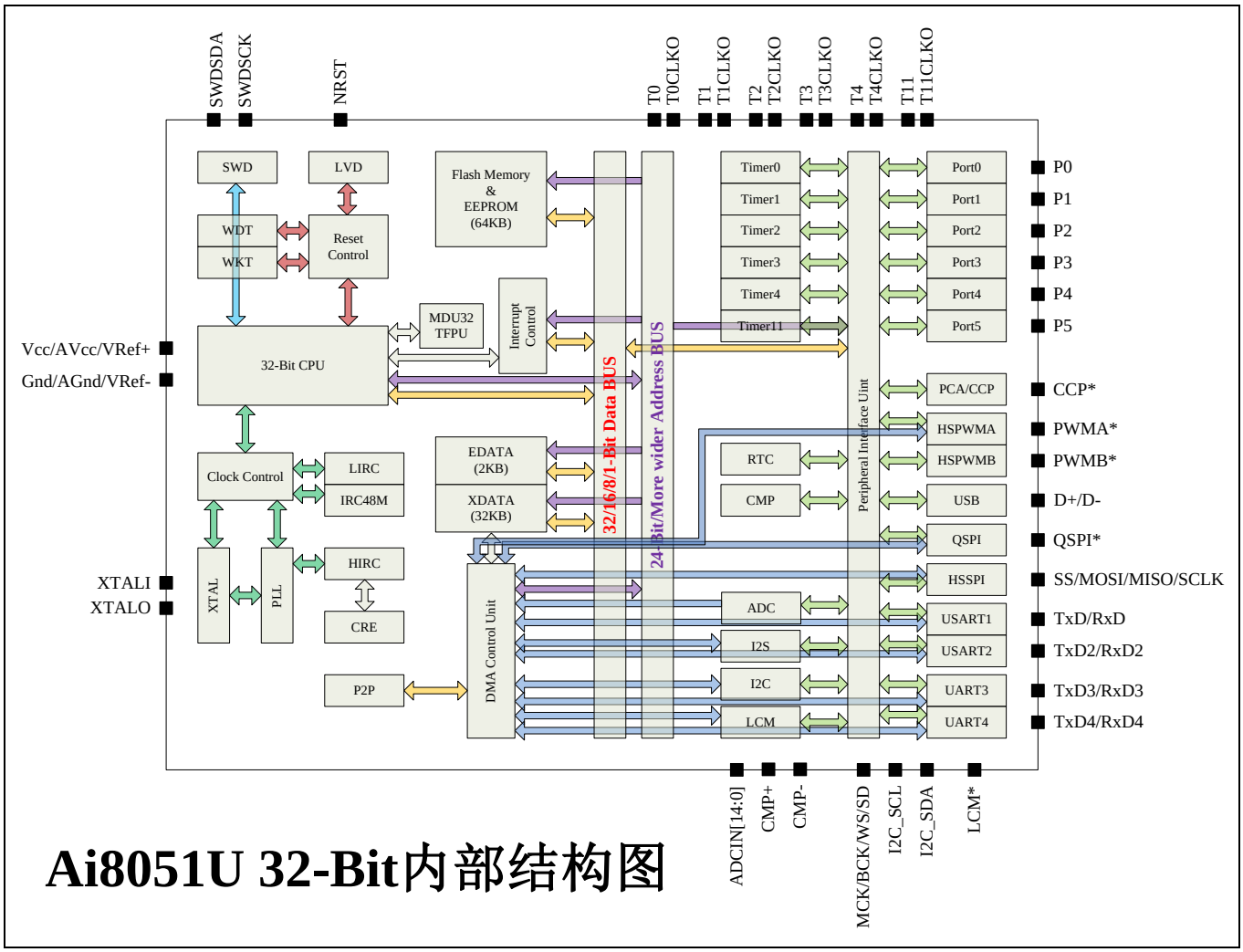
1.1.3.1　Ai8051U 8-Bit 内部结构图，内部总线工作在 8 位宽度



Ai8051U 8-Bit内部结构图

单片机型号	工作电压（V）	Flash 程序存储器 10 万次 字节	edata 内部扩展 DATA RAM 可做堆栈或变量 字节	xdata 内部大容量扩展 SRAM 可做变量 字节	EEPROM 10 万次 字节	I/O 口最多数量	传统 I/O 中断（INT0/INT1/INT2/INT3/INT4）并可掉电唤醒	所有 I/O 口均支持高电平 低电平 上升沿 下降沿中断，可掉电唤醒	DMA 8080/6800 接口 TFT 彩屏模块驱动(8 位和 16 位)	RTC 实时时钟 年/月/日/时/分/秒 对内部时钟追频 掉电唤醒	DMA UART 异步串口并可掉电唤醒	DMA USART 同步/异步串口并可掉电唤醒 额外支持两组 SPI/LIN	全速 USB，支持 USB-CDC, USB-HID	DMA I2S 音频总线 掉电唤醒	DMA SPI 并可掉电唤醒	DMA QSPI	DMA I^2C 并可掉电唤醒	TFPU (108MHz)硬件浮点 / 硬件三角 / 反三角函数运算器	MDU32，硬件 32 位乘除法器	定时器/计数器（T0/T1/T2/T3/T4/T11 外部管脚也可掉电唤醒）	144MHz，16 位高级 PWM 定时器 互补对称死区控制	传统的 PCA/CCP/PWM（可当外部中断并可掉电唤醒） 支持上升沿中断、下降沿中断以及边沿中断	掉电唤醒专用定时器	DMA 15 路高速 ADC（8 路 PWM 可当 8 路 D/A 使用）	比较器（可当 1 路 A/D，可作外部掉电检测）	内部低压检测中断并可掉电唤醒	看门狗 复位定时器	内部高可靠复位（可选复位门槛电压）	内部高精准时钟（42MHz 以下可调），满足串口通信 追频	可对外输出时钟及复位	程序加密后传输（防拦截）	可设置下次更新程序需口令	支持 RS485 下载	支持硬件 USB 直接下载和硬件 USB 仿真	本身就可在线仿真	价格及封装			供货信息
																																					LQFP48	LQFP44	PDIP40	
Ai8051U-34K64	1.9-5.5	64K	2K	32K	IAP	45	有	有	有	有	2	2	有	有	3	有	有	有	有	有	6	8	3	有	12 位	有	有	有	4 级	有	是	有	是	是	是	是	¥2.3	¥3.9	¥4.0	

1.1.3.2　Ai8051U 32-Bit 内部结构图，内部总线工作在 32 位宽度



Ai8051U 32-Bit内部结构图

Ai8051U-32Bit 内部数据总线是 32 位宽度，从而确保 32 位的 edata 数据可以一个 CPU 时钟就可完成数据读写。而 Intel 的 80251，由于制造工艺和成本的原因，芯片内部的数据总线是 8 位宽度设计，同样 32 位的 edata 数据的读写，Intel 的 80251 则需要读写 4 次，再进行数据合并。
即使读 1 个 8 位字节，也是读 32 位，只是舍弃不需要的 24 位
即使读 1 个 16 位的 WORD，2 个字节，也是读 32 位，只是舍弃不需要的 16 位
这样设计成本低，速度快！

Intel 80251 读写 32 位 4 个字节的数据，要分 4 次读写 4 个 8 位的字节，效率低。
Intel 80251 读写 16 位 2 个字节的数据，要分 2 次读写 2 个 8 位的字节，效率低。

Ai8051U 单片机性能概述

Ai8051U 系列单片机是不需要外部晶振和外部复位的单片机，是以超强抗干扰/超低价/高速/低功耗为目标的 32 位 8051 单片机，在相同的工作频率下，Ai8051U 系列单片机比传统的 8051 约快 70 倍。

Ai8051U 系列单片机是单时钟(1T)的单片机，是宽电压/高速/高可靠/低功耗/强抗静电/较强抗干扰的新一代 32 位 8051 单片机，超级加密。

MCU 内部集成高精度 R/C 时钟(±0.3%，常温下+25℃)，−0.88%~+1.05%温漂(-20℃~+65℃)，−1.38%~+1.42%温漂(-40℃~+85℃)，−3%~+3%温漂(-40℃~+125℃)。ISP 编程时工作频率设置，可彻底省掉外部昂贵的晶振和外部复位电路(内部已集成高可靠复位电路，ISP 编程时 4 级复位门槛电压可选)。

MCU 内部有 4 个可选时钟源：内部高精度 IRC 时钟（可 ISP 编程时调整频率）、内部 32KHz 的低速 IRC、外部 4M ~ 40M 晶振或外部时钟信号以及内部 PLL 输出时钟。用户代码中可自由选择时钟源，时钟源选定后可再经过 8-bit 的分频器分频后再将时钟信号提供给 CPU 和各个外设（如定时器、串口、SPI 等）。

MCU 提供两种低功耗模式：IDLE 模式和 STOP 模式。IDLE 模式下，MCU 停止给 CPU 提供时钟，CPU 无时钟，CPU 停止执行指令，但所有的外设仍处于工作状态，此时功耗约为 1.3mA（6MHz 工作频率）。STOP 模式即为主时钟停振/省电模式，即传统的掉电模式/停电模式/停机模式，此时 CPU 和全部外 设都停止工作，功耗可降低到 1uA 以下。

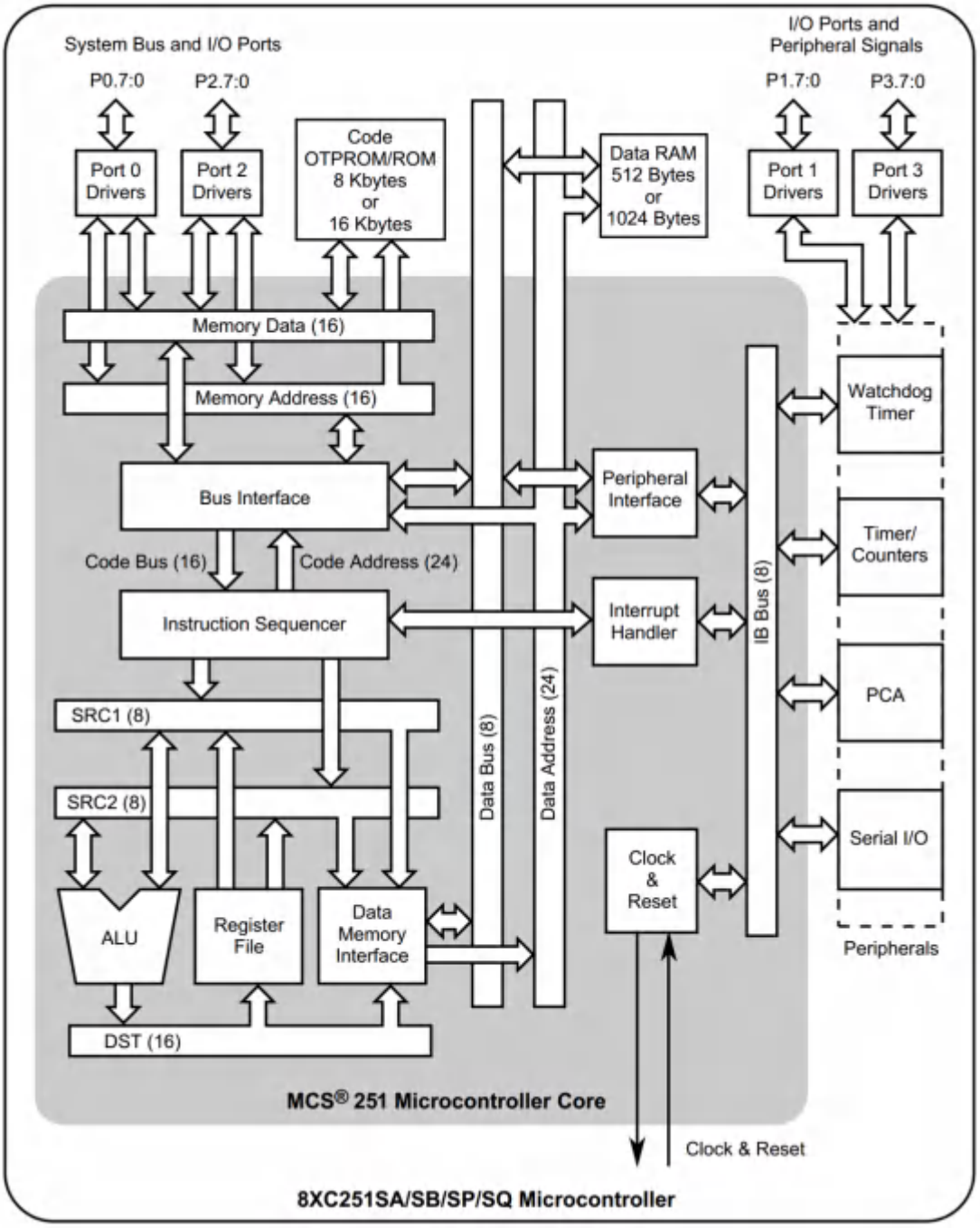
MCU 提供了丰富的数字外设（4 个串口、6 个定时器、2 组针对三相电机控制能够输出互补/对称/带死区控制信号的 16 位高级 PWM 定时器以及 I2C、SPI、QSPI、USB）接口与模拟外设（超高速 12 位 ADC、比较器），可满足广大用户的设计需求。

Ai8051U 系列单片机有 268 条强大的指令，包含 32 位加减法指令和 16 位乘除法指令。硬件扩充了 32 位硬件乘除单元 MDU32（包含 32 位除以 32 位和 32 位乘以 32 位）以及硬件三角函数/浮点运算单元 TFPU。

Ai8051U-LQFP48，RMB2.3
USB 型 32 位 8051，**1T 8051**，支持 32 位 8051 指令集和 8 位 8051 指令集
比普通 32 位机强太多的地方：
1， Ai8051U 有 TFPU@120MHz，算力强，uS 级硬件三角函数/浮点运算器；
2， Ai8051U 的抗干扰强；12 位 ADC 全新设计更精准，轨到轨比较器
3， Ai8051U 内置专业级复位电路，彻底省外部复位；
4， Ai8051U 的内部高可靠时钟完全满足串口通信要求，4 组高速串口；
5， Ai8051U 有 QSPI 读 Flash，直送 i8080/M6800-TFT 接口，视频级刷彩屏，C8T6 没有；
6， Ai8051U 的 PWM 支持硬件移相@120MHz，DMA 可以外设直接到外设，P2P
7， Ai8051U 是 34K SRAM，64K Flash
8， Ai8051U 是自带硬件 USB，1 个芯片就能直接 USB 连接电脑仿真/下载，全球唯一
9， 有 PDIP40，LQFP44，LQFP48 封装，管脚兼容经典的 89C52RC，12C5A60S2
10，当 32 位 8051 使用，可以用 Keil C251 编译器，双核兼容设计
11，当 8 位 8051 使用，可以用 Keil C51/IAR/SDCC 编译器
Ai8051U-LQFP48，USB 型 32 位 8051，RMB2.3 含税

1.1.4 Intel 80251 内部结构图

Intel 的 80251 芯片内部数据总线宽度是 8 位，但是 32 位指令集，号称 16 位单片机。内部结构图如下：



Intel 80251 读写 32 位 4 个字节的数据，要分 4 次读写 4 个 8 位的字节，效率低。
Intel 80251 读写 16 位 2 个字节的数据，要分 2 次读写 2 个 8 位的字节，效率低。

1.2 数制与编码

数制是人们利用符号进行计数的科学方法。

数制有很多种，常用的数制有：二进制，十进制和十六进制。

进位计数制是把数划分为不同的位数，逐位累加，加到一定数量之后，再从零开始，同时向高位进位。进位计数制有三个要素：数码符号、进位规律和计数基数。下表是各常用数制的总体介绍。

常用的数制	表示符号	数码符号	进制规律	计数基数
二进制	B	0、1	逢二进一	2
十进制	D	0、1、2、3、4、5、6、7、8、9	逢十进一	10
十六进制	H	0、1、2、3、4、5、6、7、8、9、 A、B、C、D、E、F	逢十六进一	16

我们日常生活中计数一般采用十进制。计算机中采用的是二进制，因为二进制具有运算简单，易实现且可靠，为逻辑设计提供了有利的途径、节省设备等优点。为区别于其它进制数，二进制数的书写通常在数的右下方注上基数 2，或加后面加 B 表示。二进制数中每一位仅有 0 和 1 两个可能的数码，所以计数基数为 2。二进制数的加法和乘法运算如下：

0 + 0 = 0　　0 + 1 = 1 + 0 = 1　　1 + 1 = 10

0 × 0 = 0　　0 × 1 = 1 × 0 =0　　1 × 1 = 1

由于二进制数在使用中位数太长,不容易记忆，为了便于描述，又常用十六进制作为二进制的缩写。十六进制通常在表示时用尾部标志 H 或下标 16 以示区别。

1.2.1 数制转换

现在我们来介绍这些常用数制之间的转换。

一：二进制 — 十进制转换

方法：将二进制数按权(如下式)展开，然后将各项的数值按十进制数相加，就得到相应的等值十进制数。

例如：N=(1101.101)B，那么 N 所对应的十进制数时多少呢?

按权展开 $N=1\times2^3+1\times2^2+0\times2^1+1\times2^0+1\times2^{-1}+0\times2^{-2}+1\times2^{-3}=8+4+0+1+0.5+0+0.125=(13.625)D$

二：十进制 — 二进制转换

方法：分两部分进行即整数部分和小数部分。

①整数部分转换(基数除法)：

★把我们要转换的数除以二进制的基数(二进制的基数为 2)，把余数作为二进制的最低位；

★把上一次得的商在除以二进制基数(即 2)，把余数作为二进制的次低位；

★继续上一步,直到最后的商为零,这时的余数就是二进制的最高位.

②小数部分转换(基数乘法)：

★把要转换数的小数部分乘以二进制的基数(二进制的基数为 2)，把得到的整数部分作为二进制小数部分的最高位；

★把上一步得的小数部分再乘以二进制的基数(即 2)，把整数部分作为二进制小数部分的次高位；

★继续上一步，直到小数部分变成零为止。或者达到预定的要求也可以。

例如：将$(213.8125)_{10}$化为二进制数可按如下进行：
先化整数部分：

2	213	余数=1=k_0
2	106	余数=0=k_1
2	53	余数=1=k_2
2	26	余数=0=k_3
2	13	余数=1=k_4
2	6	余数=0=k_5
2	3	余数=1=k_6
2	1	余数=1=k_7
	0	

于是整数部分$(213)_{10}=(11010101)_2$
再化小数部分：

0.8125 × 2 = 1.6250 ------ 整数部分=1=k_{-1}

0.6250 × 2 = 1.2500 ------ 整数部分=1=k_{-2}

0.2500 × 2 = 0.5000 ------ 整数部分=0=k_{-3}

0.5000 × 2 = 1.0000 ------ 整数部分=1=k_{-4}

于是小数部分$(0.8125)_{10}=(0.1101)_2$

综上所述，十进制数 $213.8125=(11010101.1101)_2=(11010101.1101)B$

三：二进制 — 十六进制转换

方法：二进制和十六进制之间满足 24 的关系，因此把要转换的二进制从低位到高位每 4 位一组，高位不足时在有效位前面添“0”，然后把每组二进制数转换成十六进制即可。

例如，将(010111011110.10110010)B 转换为十六进制数：

(0101 1101 1110 . 1011 0010)B
　　↓　　↓　　↓　　↓　　↓
= (5　　D　　E　　B　　2)H

于是，(010111011110.10110010)B=(5DE.B2)H

四：十六进制 — 二进制转换

方法：十六进制转换为二进制时，把上面二进制转换十六进制的过程逆过来，即转换时只需将十六进制的每一位用等值的 4 位二进制代替就行了。

例如：将(C1B.C6)H 转换为二进制数：

(　C　　1　　B .　C　　6　)H
　　↓　　↓　　↓　　↓　　↓
= (1100　0001　1011　1100　0110)B

于是，(C1B.C6)H=(110000011011.11000110)B

五：十六进制 — 十进制转换

方法：将十六进制数按权(如下式)展开，然后将各项的数值按十进制数相加，就得到相应的等值十进制数。

例如：N=(2A.7F)H，那么 N 所对应的十进制数时多少呢？
　　按权展开 $N=2\times16^{1}+10\times16^{0}+7\times16^{-1}+15\times16^{-2}=32+10+0.4375+0.05859375=(42.49609375)D$
于是，(2A.7F)H =(42.49609375)D

六：十进制 — 十六进制转换

方法：将十进制数转换为十六进制数时，可以先将十进制数转换为二进制数，然后再将得到的二进制数转换为等值的十六进制数。

1.2.2 原码、反码及补码

在生活中,数有正负之分,在计算机中是怎样表示数的正负符号呢？

在生活中表示数的时候一般都是把正数前面加一个“+”，负数前面加一个“-”，但是计算机是不认识这些的，通常在二进制数前面增加一位符号位。符号位为“0”表示“+”，符号位为“1”表示“-”。这种形式的二进制数称为原码。如果原码为正数，则原码的反码和补码都与原码相同。如果原码为负数，则将原码(除符号位外)按位取反，所得的新二进制数称为原码的反码，反码加 1 为其补码。

原码、反码、补码这三种形式的总结如下表所示：

	真值	原码	反码	补码
正数	+N	0N	0N	0N
负数	-N	1N	$(2^n-1)+N$	2^n+N

例 1：求+18 和-18 八位原码、反码、补码形式。

真值	原码	反码	补码
+18	00010010	00010010	00010010
-18	10010010	11101101	11101110

1.2.3 常用编码

指定某一组二进制数去代表某一指定的信息，就称为编码。

一：十进制编码

用二进制码表示的十进制数，称为十进制编码。它具有二进制的形式，还具有十进制的特点它可作为人们与数字系统的联系的一种间表示。十进制编码有很多种，最常用的一种是 BCD 码，又称 8421 码。

下面我们用表列出几种常见的十进制编码:

编码种类 / 十进制数	8421 码（BCD 码）	余 3 码	2421 码	5211 码	7321 码
0	0000	0011	0000	0000	0000
1	0001	0100	0001	0001	0001
2	0010	0101	0010	0100	0010
3	0011	0110	0011	0101	0011
4	0100	0111	0100	0111	0101
5	0101	1000	1011	1000	0110
6	0110	1001	1100	1001	0111
7	0111	1010	1101	1100	1000
8	1000	1011	1110	1101	1001
9	1001	1100	1111	1111	1010
权	8421		2421	5211	7321

十进制编码分为有权和无权编码。有权编码码是指每一位十进制数符均用一组四位二进制码来表示，而且二进制码的每一位都有固定权值。无权编码码是指二进制码中每一位都没有固 定的权值。上表中 8421 码(即 BCD 码)、2421 码、5211 码、7321 码都是有权编码，而余 3 码是无权编码。

二：奇偶校验码

在数据的存取、运算和传送过程中，难免会发生错误，把“1”错成“0”或把“0”错成“1”。奇偶校验码是一种能检验这种错误的代码。它分为两部分；信息位和奇偶校验位。有奇数个“1”称为奇校验，有偶数个“1”则称为偶校验。

1.3 几种常用的逻辑运算及其图形符号

逻辑代数中常用的运算有：与(AND)、或(OR)、非(NOT)、与非(NAND)、或非(NOR)、与或非(AND-NOR)、异或(EXCLUSIVE OR)、同或(EXCLUSIVE NOR)等。其中与(AND)、或(OR)、非(NOT)运算时三种最基本的运算。

一：与运算及与门

与运算：决定事件结果的全部条件同时具备时，事件才发生。
逻辑变量 A 和 B 进行与运算时可写成：Y=A・B

真值表		
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

与门：实行与逻辑运算的单元电路。
与门图形符号：

二：或运算及或门

或运算：决定事件结果的各条件中只要有任何一个满足，事件就会发生。
逻辑变量 A 和 B 进行或运算时可写成：Y=A+B

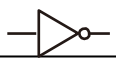
真值表		
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

或门：实行或逻辑运算的单元电路。
或门图形符号：

三：非运算及非门

非运算：条件具备时，事件不会发生；条件不具备时，事件才会发生。
逻辑变量 A 进行非运算时可写成：Y=A′

真值表	
A	Y
0	1
1	0

非门：实行非逻辑运算的单元电路。
非门图形符号： 

四：与非运算及与非图形符号

与非运算：先进行与运算，然后将结果求反，最后得到的即为与非运算结果。
逻辑变量 A 和 B 进行与非运算时可写成：Y=(A・B)′

真值表		
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

与非图形符号：

五：或非运算及或非图形符号

或非运算：先进行或运算，然后将结果求反，最后得到的即为或非运算结果。
逻辑变量 A 和 B 进行或非运算时可写成：Y=(A+B)′

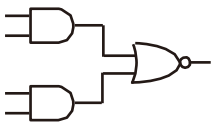
真值表		
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

或非图形符号：

六：与或非运算及与或非图形符号

与或非运算：在与或非逻辑运算中有 4 个逻辑变量 A、B、C、D。假设 A 和 B 为一组，C 和 D 为一 组，A、B 之间以及 C、D 之间都是与的关系，只要 A、B 或 C、D 任何一组同时为 1， 输出 Y 就是 0。只有当每一组输入都不全是 1 时，输出 Y 才是 1。
逻辑变量 A 和 B 进行或非运算时可写成：Y=(A・B+C・D)′

真值表				
A	B	C	D	Y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

与或非图形符号： 

七：异或运算及异或图形符号

异或运算：当 A、B 不同时，输出 Y 为 1；而当 A、B 相同时，输出 Y 为 0。 逻辑变量 A 和 B 进行异或运算时可写成：$Y = A \oplus B = (A \cdot B') + (A' \cdot B)$

真值表		
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

异或图形符号：

八：同或运算及同或图形符号

同或运算：当 A、B 不同时，输出 Y 为 0；而当 A、B 相同时，输出 Y 为 1。 逻辑变量 A 和 B 进行同或运算时可写成：$Y = A \odot B = (A \cdot B)+(A' \cdot B')$

真值表		
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

同或图形符号：

2　集成开发环境的使用与 ISP 下载软件的介绍

2.1　　下载 Keil 集成开发环境

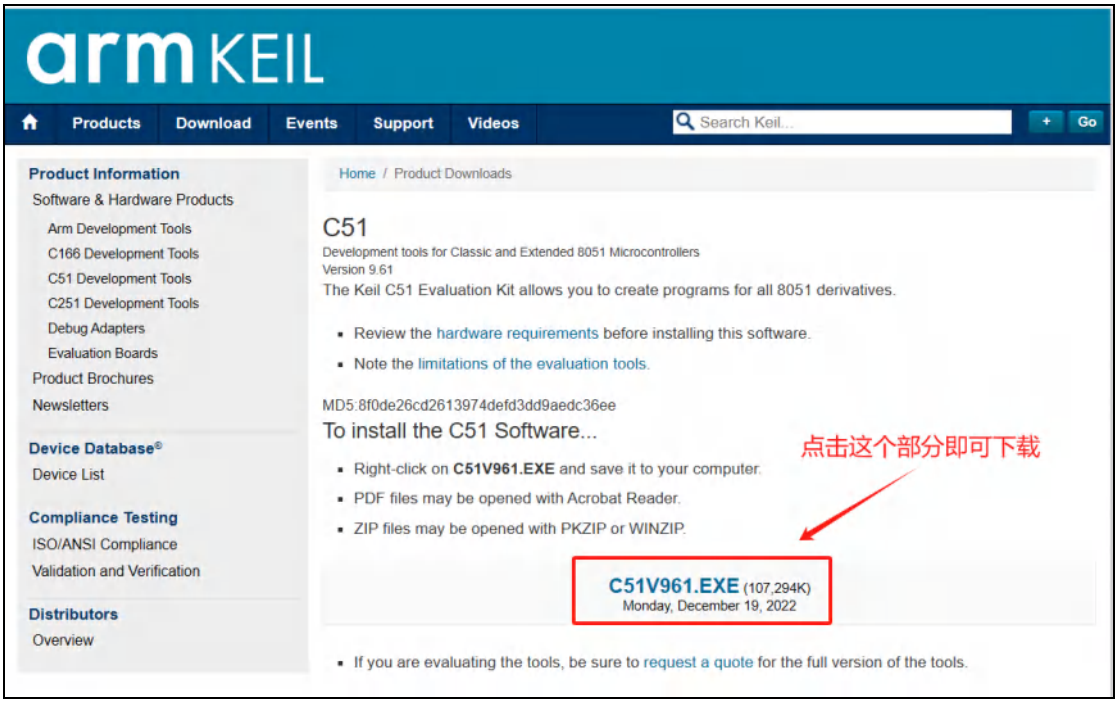
登录 Keil 官网，下载最新版的 C251 和 C51 编译器，下载链接如下：

https://www.keil.com/download/product/

进入下载页面，我们可以看到 C51 和 C251 的下载链接，这里将这两个安装包都下载到桌面上

如下载 C51 编译器，则点击 C51 下载图标，进入如下界面，输入联系信息，注意只有 E-mail 拼写检查较严谨，必须用类似 c@qq.com 这样的格式，其他任意字母填写。然后点击 Submit。

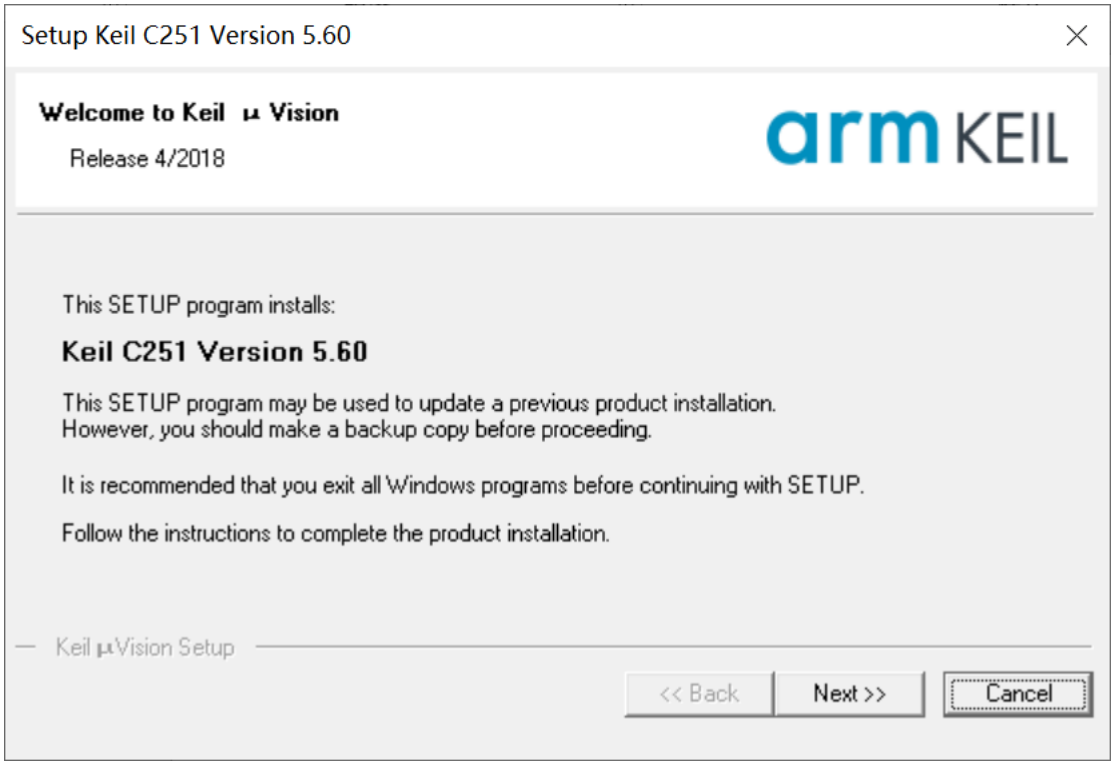
出现下载 C51V961.EXE 的界面/或更高版本的界面，点击粗体蓝色字即可下载 C51 安装包。同样步骤下载 C251 安装包。



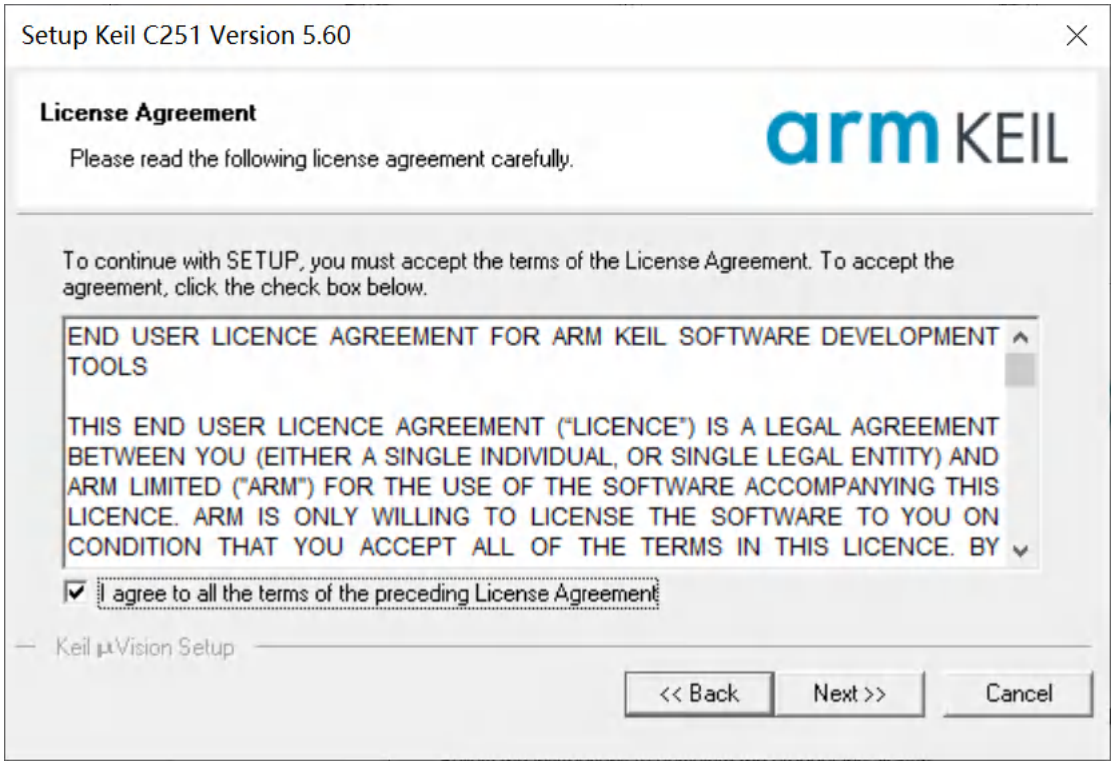
2.2 安装 Keil 集成开发环境

2.2.1 安装 Keil C251 集成开发环境

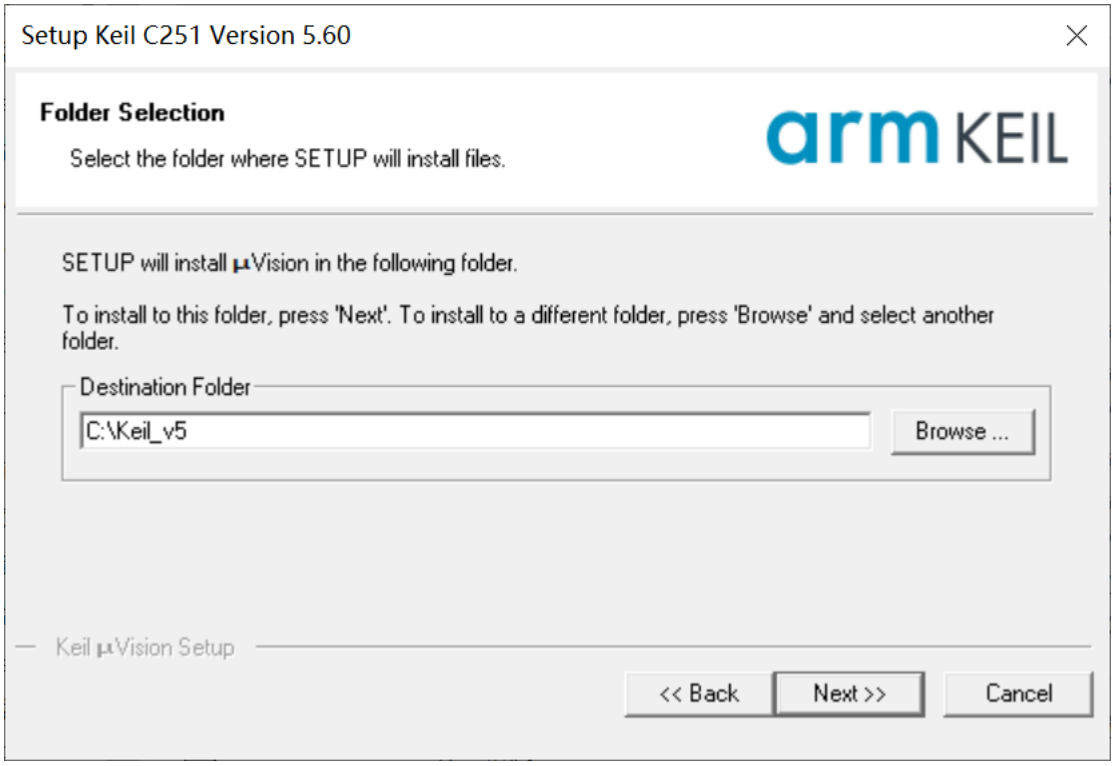
双击下载的安装包开始安装，点击“Next”：



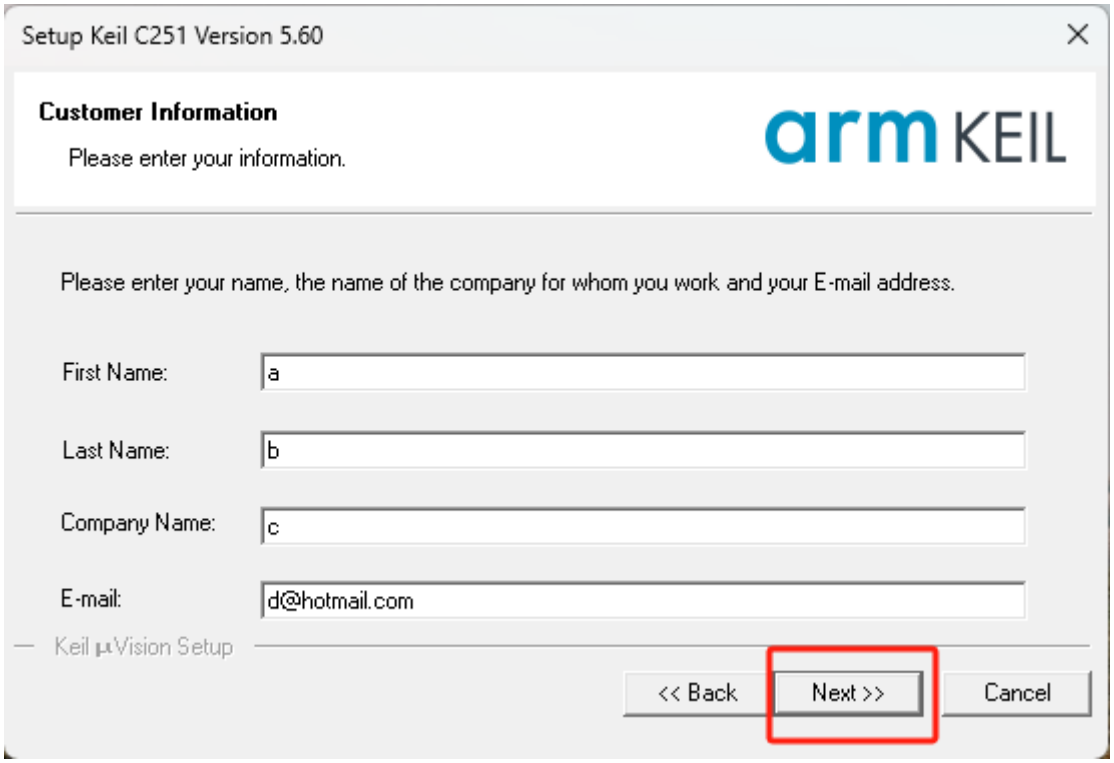
勾选“I agree to all the terms of the preceding License Agreement”,然后点击“Next”：



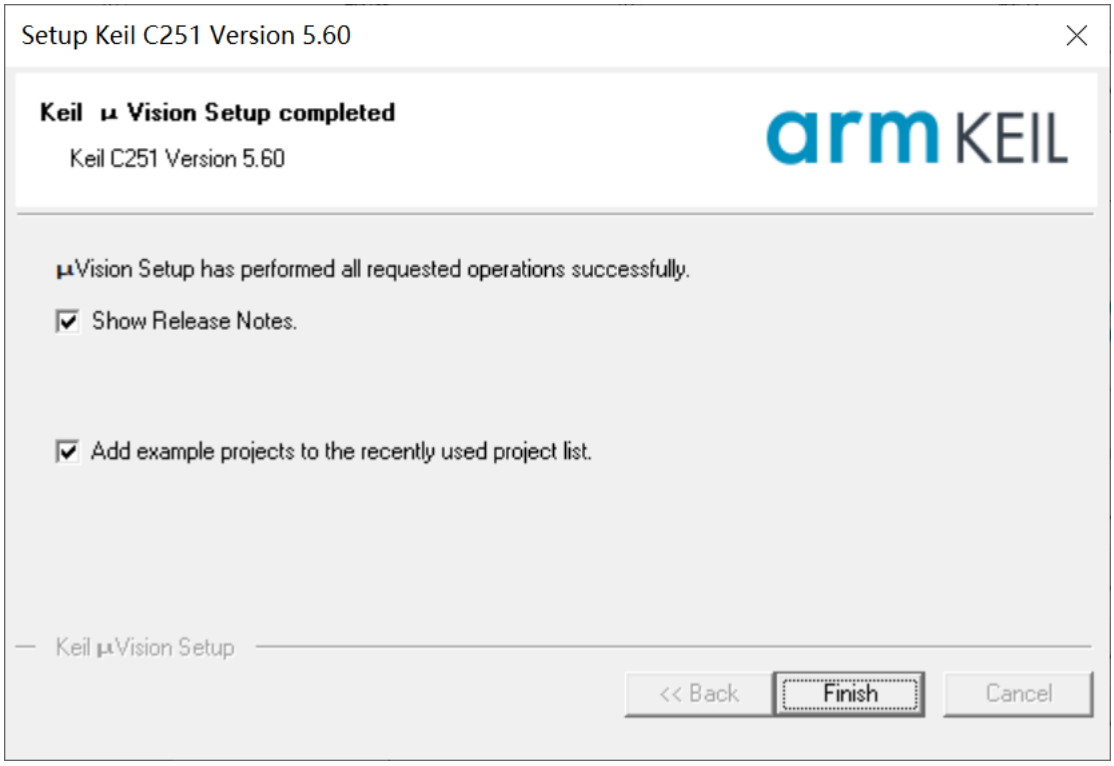
选择安装目录，然后点击“Next”：



填写个人信息，然后点击“Next”：

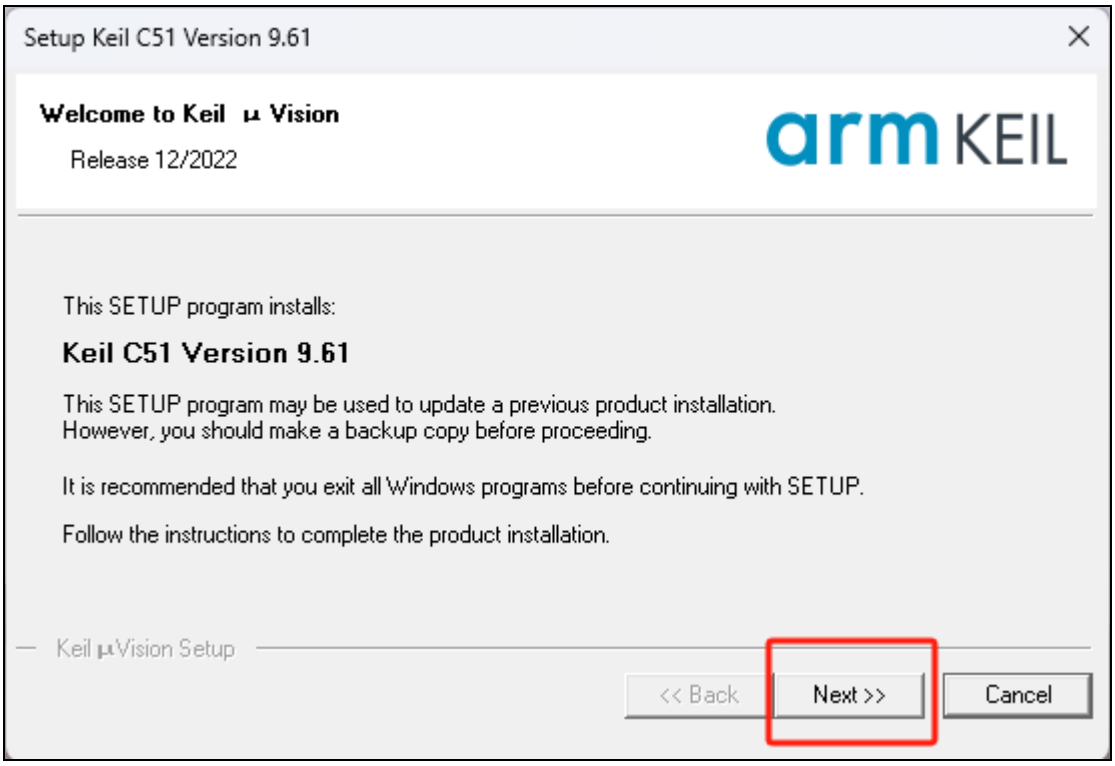


安装完成，点击“Finish”结束。

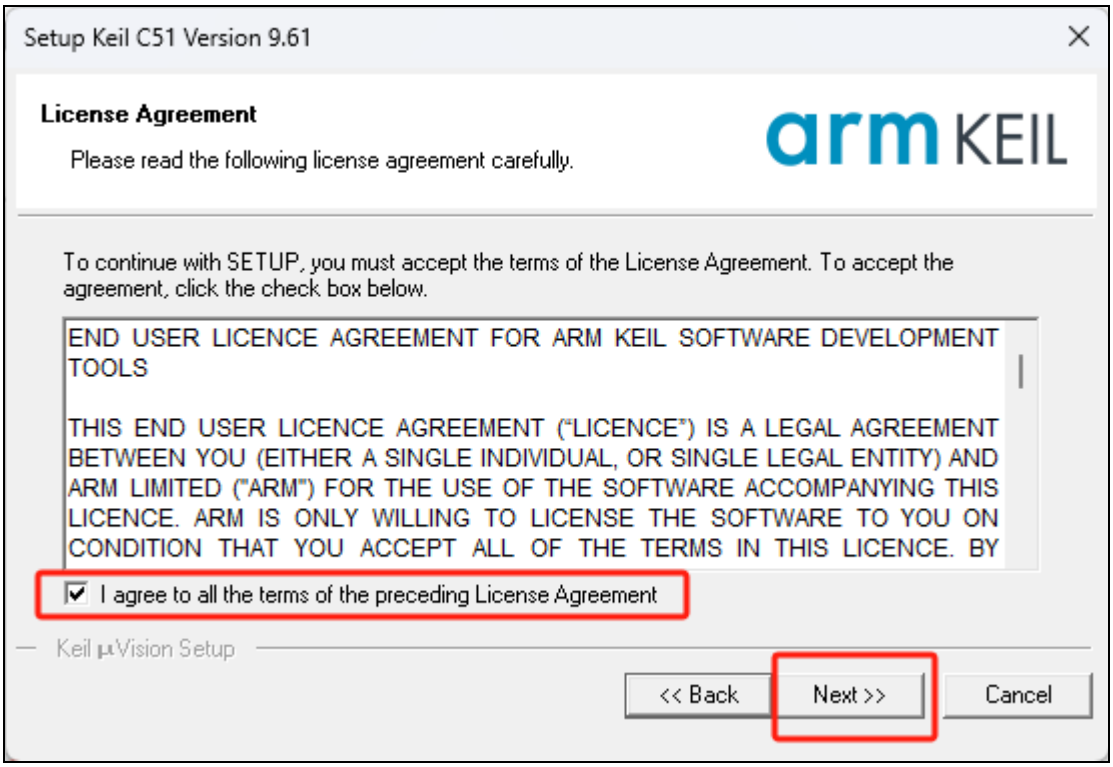


2.2.2　安装 Keil C51 集成开发环境

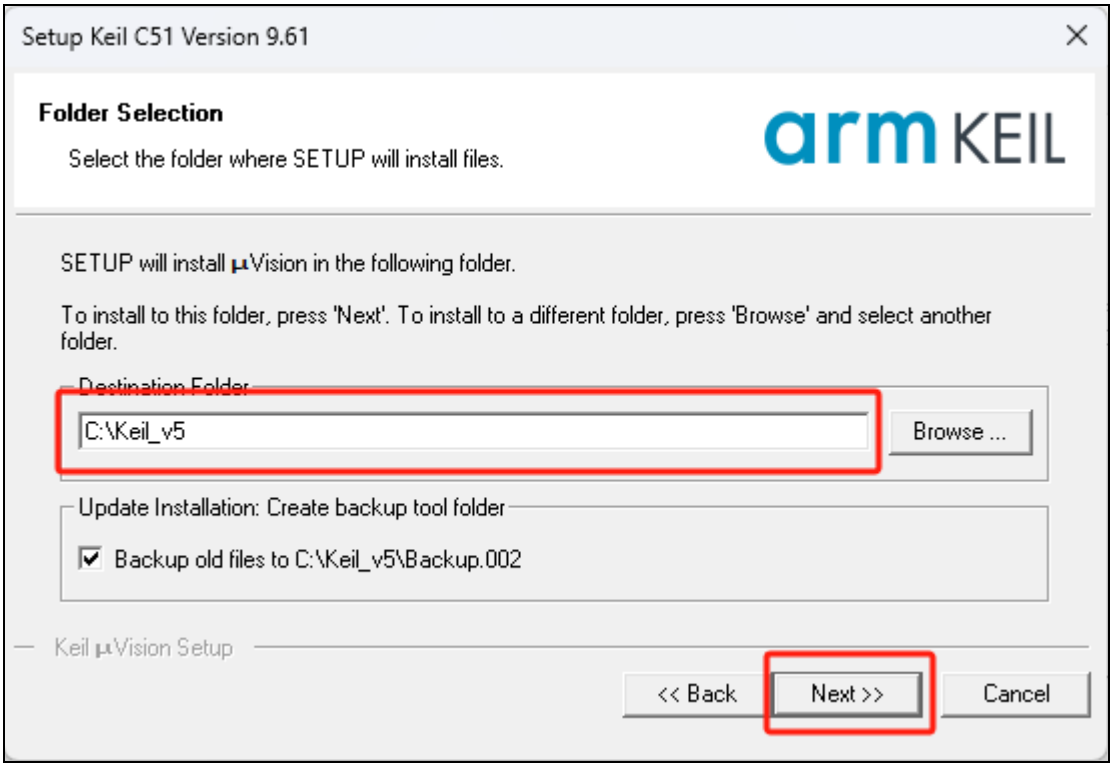
双击下载的安装包开始安装，点击“Next”：



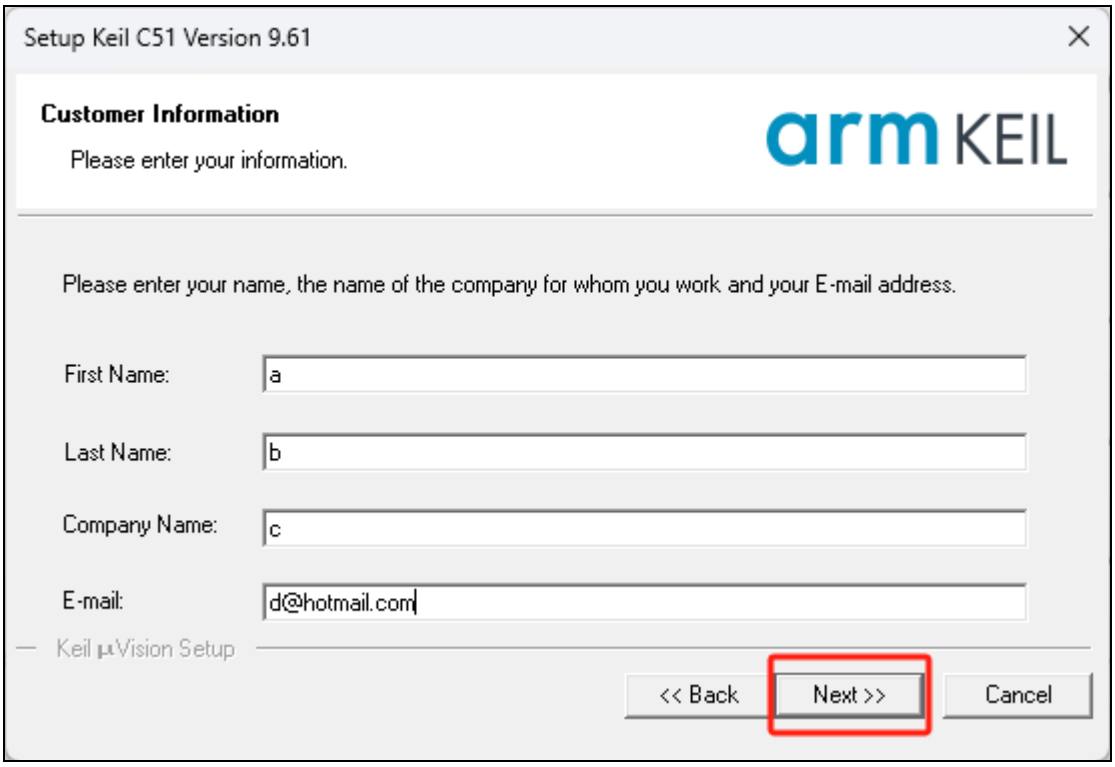
勾选“I agree to all the terms of the preceding License Agreement”,然后点击“Next”：



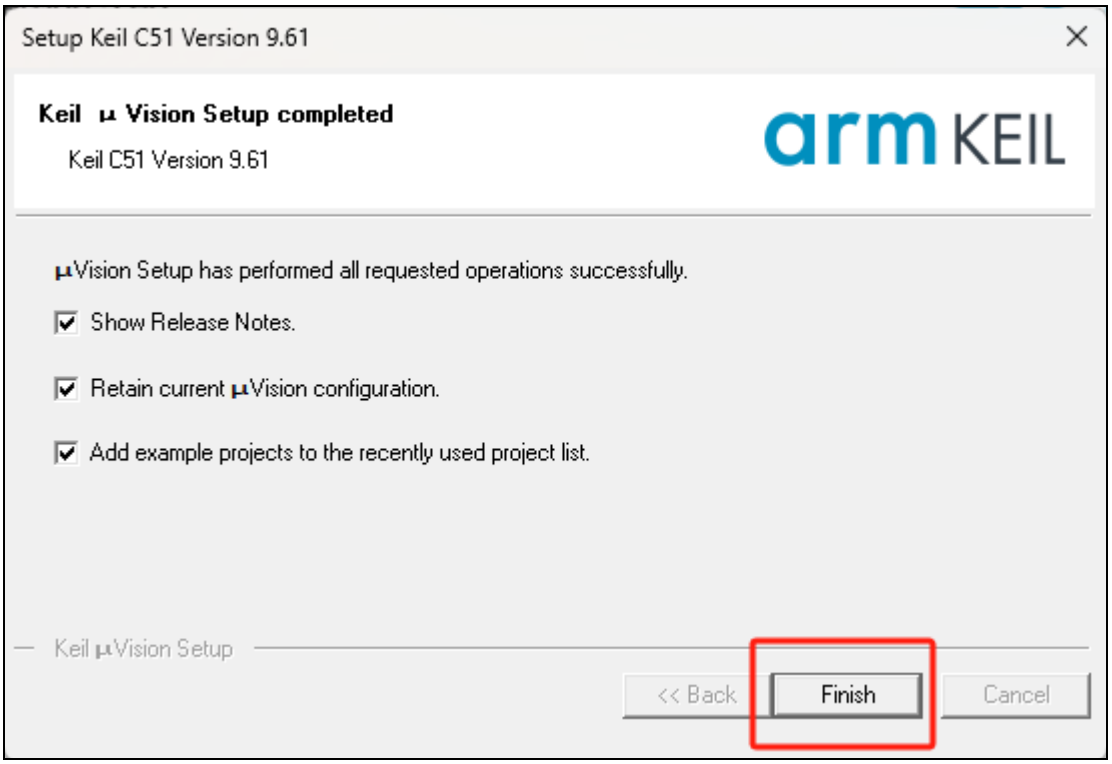
选择安装目录，然后点击“Next”：



填写个人信息，然后点击“Next”：

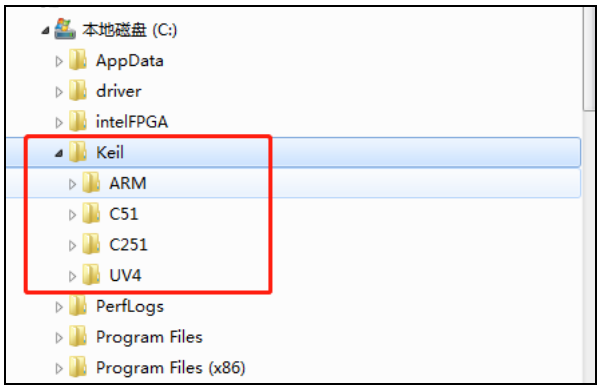


安装完成，点击“Finish”结束。

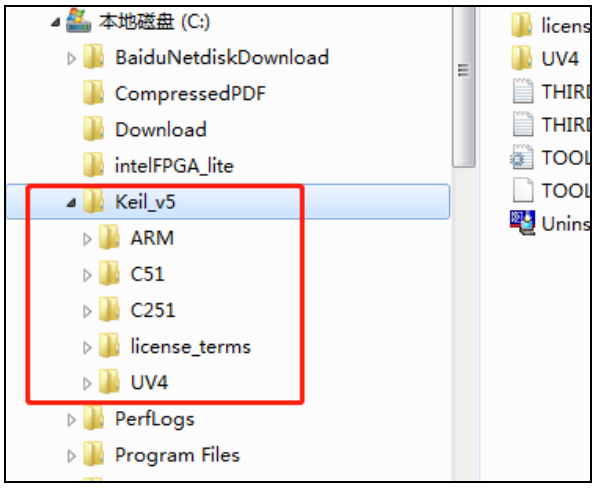


2.2.3 Keil 的 C51、C251 和 MDK 可安装在同一台电脑同一个目录

旧版本的 Keil 软件的安装目录默认是 C:\Keil，C51、C251 和 MDK 分别会被安装在 C:\Keil 目录下的 C51、C251 和 ARM 目录中，如下图所示。



新版本的 Keil 软件的安装目录默认是 C:\Keil_v5，C51、C251 和 MDK 分别会被安装在 C:\Keil_v5 目录下的 C51、C251 和 ARM 目录中，如下图所示。



无论是新版本还是旧版本，C51、C251 和 MDK 是安装在不同的目录，并不会有冲突。软件的和谐也是 3 个软件分别进行的，之前已经安装完成并设置好的软件，并不会因为后续有安装新的软件而改变。所以安装时只需要按照默认方式安装即可，Keil 软件会自动处理好。

2.2.4　从哪购买 KEIL，获取无限制版 KEIL

可以 BAIDU:

方法 1, BAIDU：从哪购买 KEIL，获取无限制版 KEIL

方法 2, BAIDU：如何购买 KEIL C251

方法 3, BAIDU：如何购买 KEIL C51

方法 4, BAIDU：如何获取无限制版 KEIL

2.3 安装 AIapp-ISP 下载/编程/烧录 工具,含强大的辅助开发工具

2.3.1 安装 AIapp-ISP 下载/编程/烧录 工具软件，取代 STC-ISP

安装 STCAI 公司研发的 AIapp-ISP 下载 / 编程 / 烧录 / 软件，含强大的辅助开发工具包。

登录 STCAI 官网，下载最新版的 AIapp-ISP 安装包，下载链接如下：https://www.stcai.com/gjrj

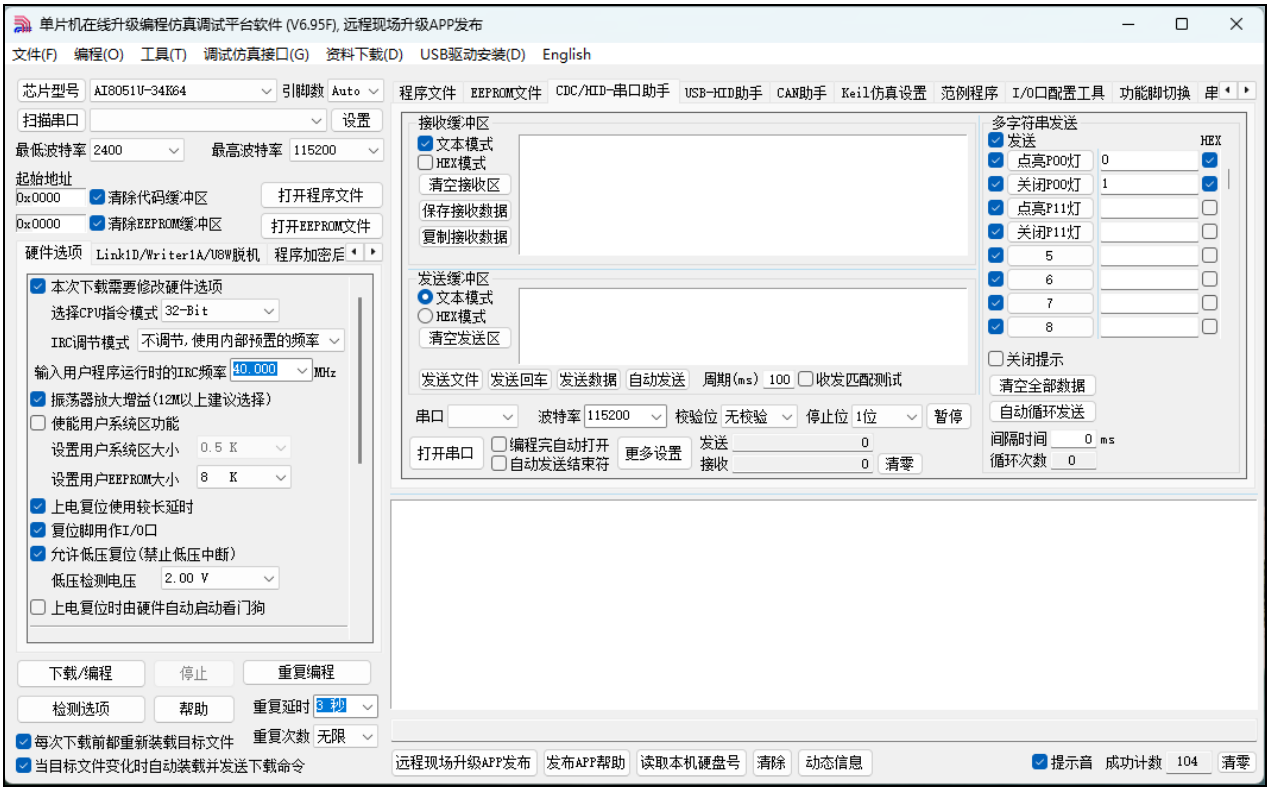
下载 AIapp-ISP 压缩包到桌面：

下载下来的是压缩包文件如下图：

双击后解压缩如下图：

如果是解压缩到非桌面的其他目录，如 D:\STCAI-ISP，建议创建快捷方式到桌面方便打开。这实际是一个免安装的绿色软件。

双击图标，即可打开 AIapp-ISP 软件界面，即可直接使用，如下图：

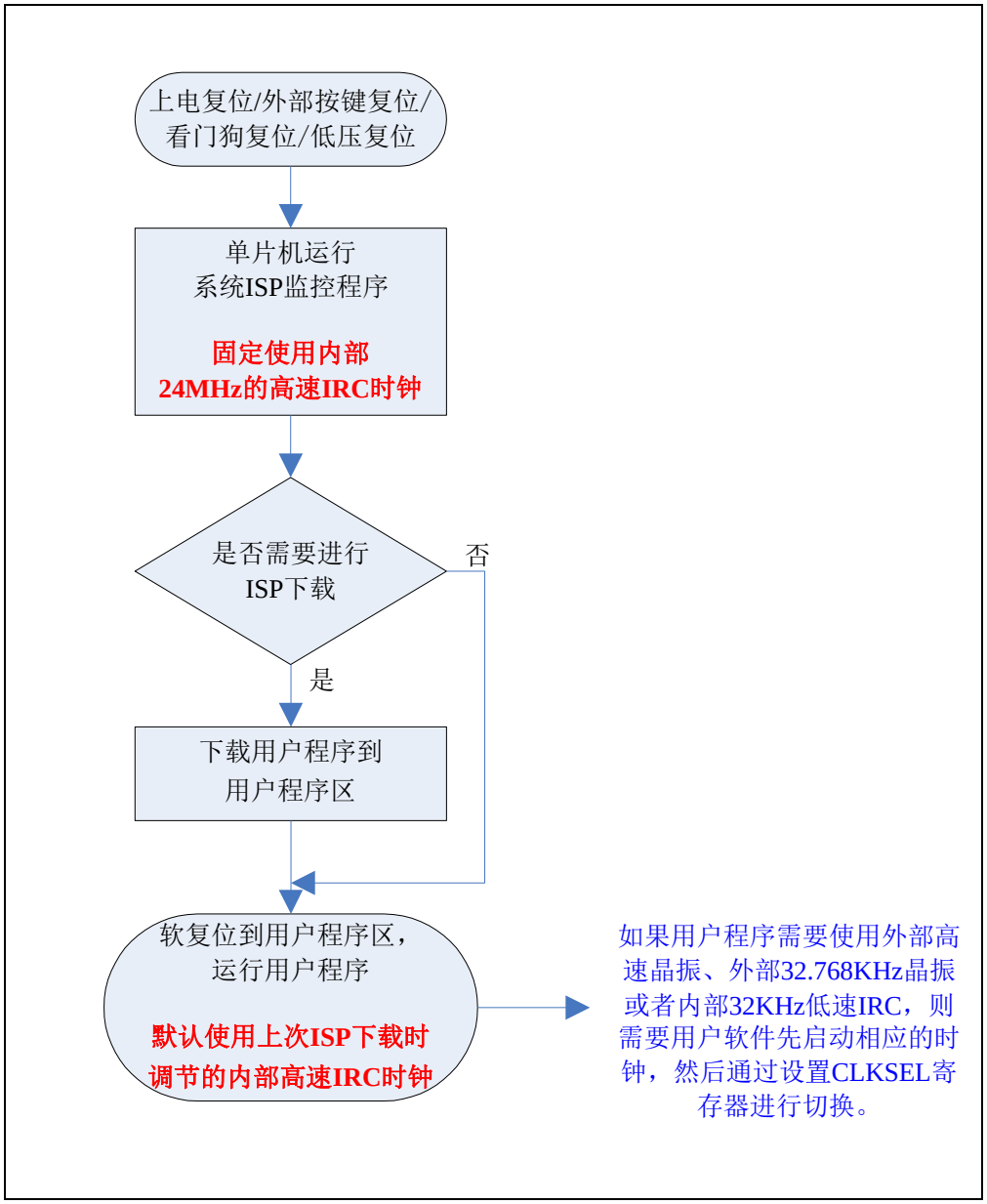

单片机在线升级编程仿真调试平台软件 (V6.95F), 远程现场升级APP发布
文件(F) 编程(O) 工具(T) 调试仿真接口(G) 资料下载(D) USB驱动安装(D) English
芯片型号 AI8051U-34K64 引脚数 Auto
扫描串口 设置
最低波特率 2400 最高波特率 115200
起始地址
0x0000 清除代码缓冲区 打开程序文件
0x0000 清除EEPROM缓冲区 打开EEPROM文件
硬件选项 Link1D/Writer1A/U8W脱机 程序加密后
本次下载需要修改硬件选项
选择CPU指令模式 32-Bit
IRC调节模式 不调节，使用内部预置的频率
输入用户程序运行时的IRC频率 40.000 MHz
振荡器放大增益(12M以上建议选择)
使能用户系统区功能
设置用户系统区大小 0.5 K
设置用户EEPROM大小 8 K
上电复位使用较长延时
复位脚用作I/O口
允许低压复位(禁止低压中断)
低压检测电压 2.00 V
上电复位时由硬件自动启动看门狗
下载/编程 停止 重复编程
检测选项 帮助 重复延时 3 秒
每次下载前都重新装载目标文件 重复次数 无限
当目标文件变化时自动装载并发送下载命令
程序文件 EEPROM文件 CDC/HID-串口助手 USB-HID助手 CAN助手 Keil仿真设置 范例程序 I/O口配置工具 功能脚切换
接收缓冲区
文本模式
HEX模式
清空接收区
保存接收数据
复制接收数据
发送缓冲区
文本模式
HEX模式
清空发送区
发送文件 发送回车 发送数据 自动发送 周期(ms) 100 收发匹配测试
串口 波特率 115200 校验位 无校验 停止位 1位 暂停
打开串口 编程完自动打开 自动发送结束符 更多设置
发送 0
接收 0 清零
多字符串发送
发送 HEX
点亮P00灯 0
关闭P00灯 1
点亮P11灯
关闭P11灯
5
6
7
8
关闭提示
清空全部数据
自动循环发送
间隔时间 0 ms
循环次数 0
远程现场升级APP发布 发布APP帮助 读取本机硬盘号 清除 动态信息
提示音 成功计数 104 清零

2.3.2　　STCAI 单片机上电工作过程

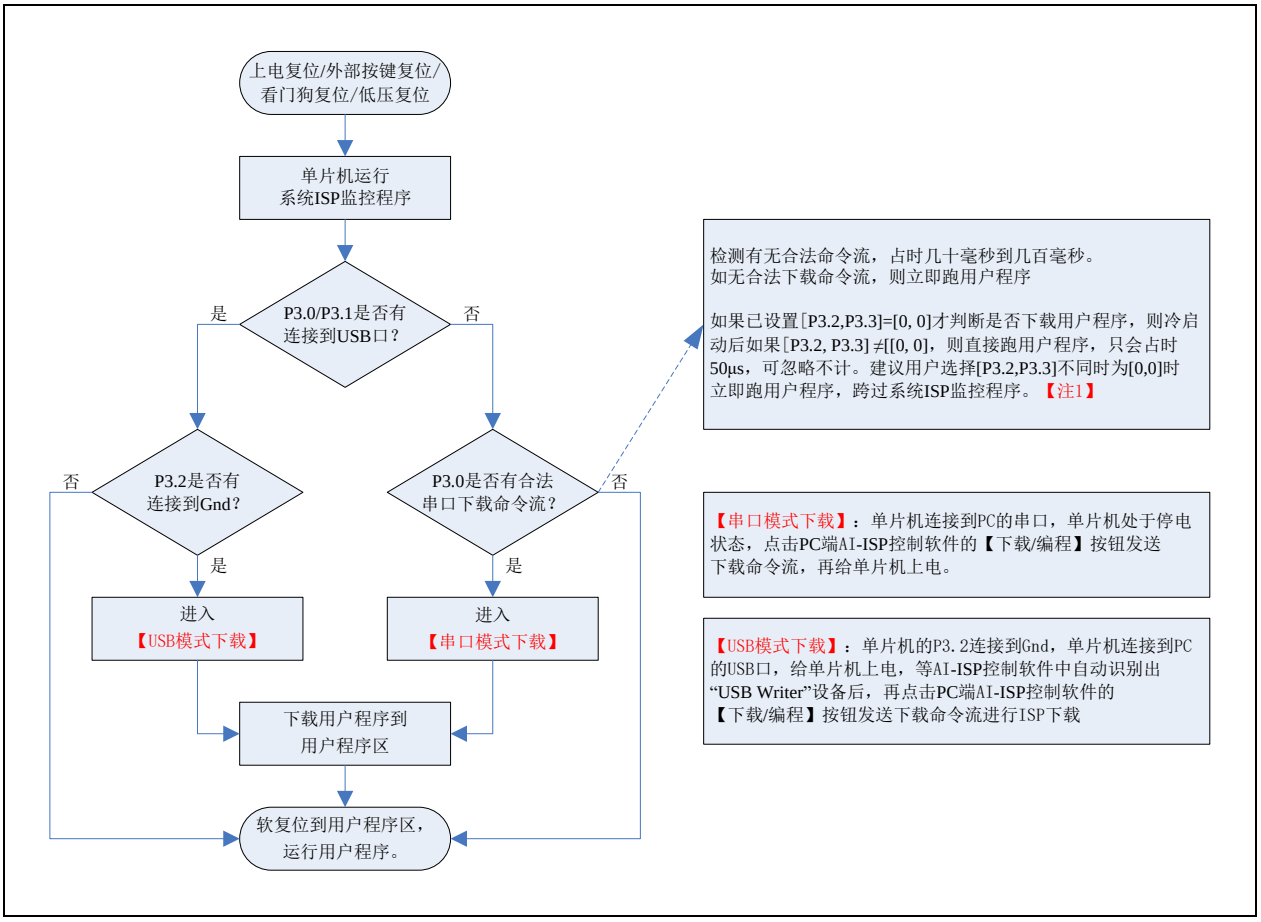
上电复位/复位脚复位/看门狗复位/低压检测复位时，芯片默认从 ISP 系统程序区开始执行系统程序代码，此时固定使用内部 24MHz 的高速 IRC 时钟。

如发现需要下载用户程序且下载完成后复位到用户程序区或者不需要下载直接复位到用户程序区时，默认会使用上次用户下载时所调节的高速 IRC 时钟，如果用户程序需要使用外部高速晶振、外部 32.768KHz 晶振或者内部 32KHz 低速 IRC，则需要用户软件先启动相应的时钟，然后通过设置 CLKSEL 寄存器进行切换。

启动流程如下：



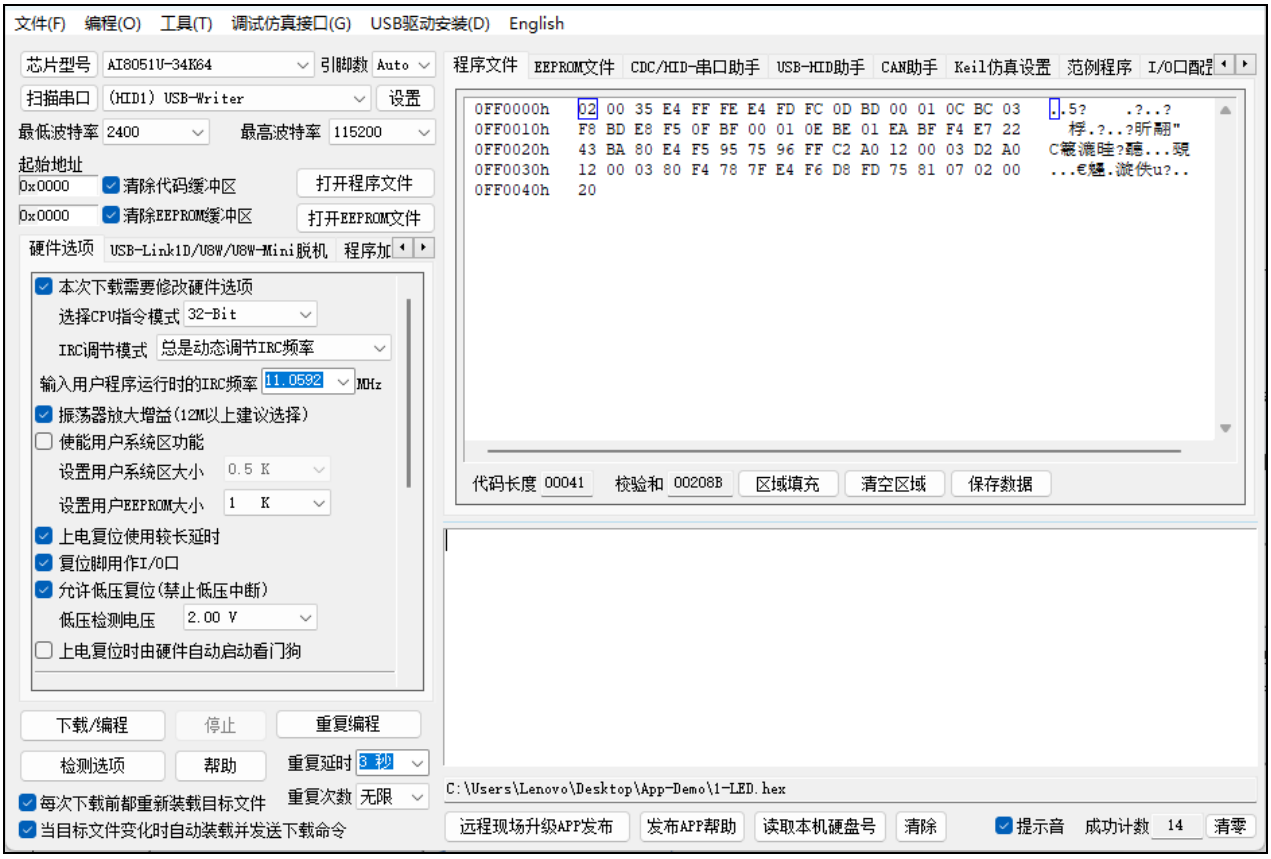
2.3.3 Ai8051U 的 ISP 下载流程图（硬件 USB+串口模式）



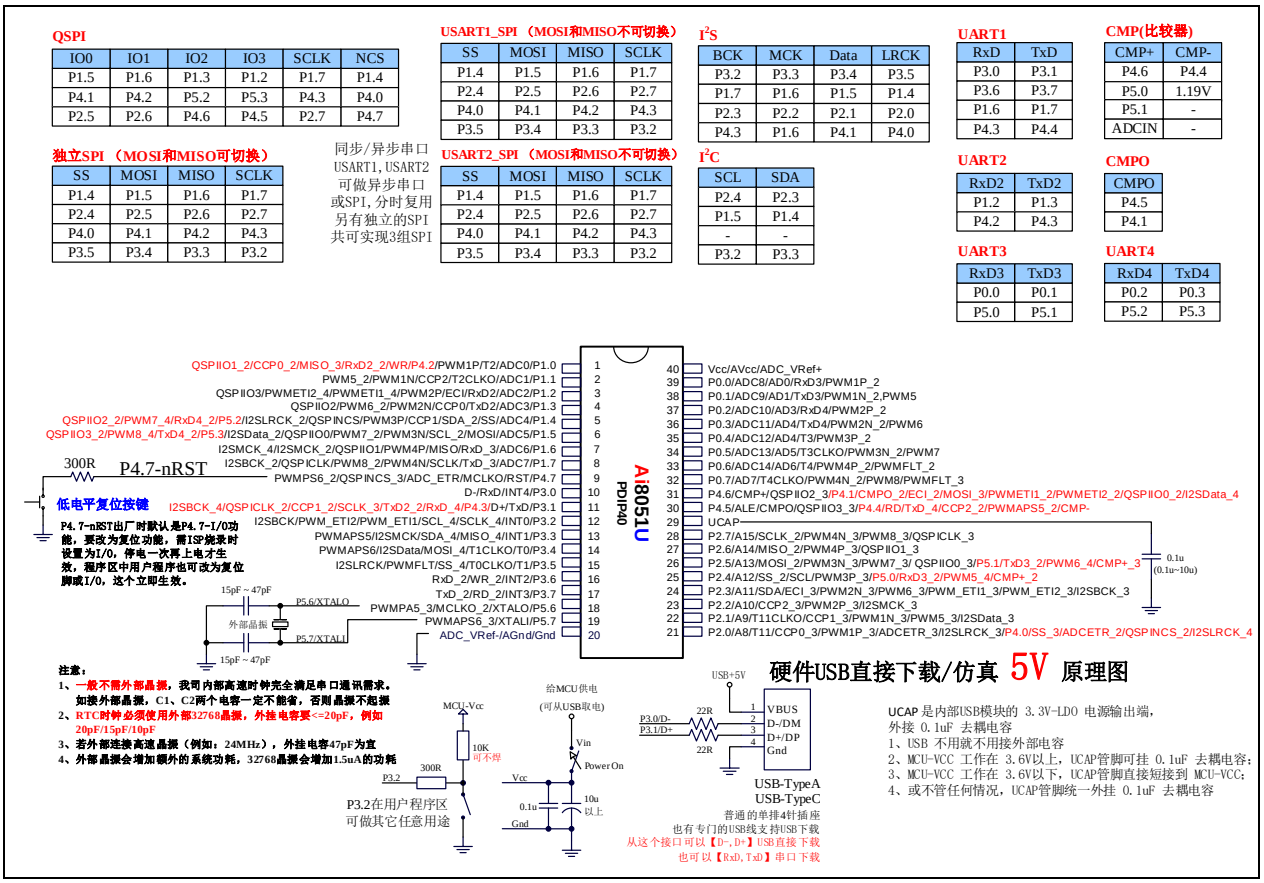
注意：因 [P3.0, P3.1] 作下载/仿真用(下载/仿真接口仅可用 [P3.0, P3.1])，故建议用户将串口 1 放在 P3.6/P3.7 或 P1.6/P1.7 ，若用户不想切换，坚持使用 P3.0/P3.1 工作或作为串口 1 进行通信，则务必在下载程序时，在软件上勾选“下次冷启动时，P3.2/P3.3 为 0/0 时才可以下载程序”。【注 1】

【注 1】：AI15，AI8 系列及以后新出的芯片的烧录保护引脚为 P3.2/P3.3，之前早期芯片的烧录保护引脚为 P1.0/P1.1。

AIapp-ISP 界面如下图所示：



2.3.4 Ai8051U 的 USB 直接下载电路及 ISP 下载操作步骤



备注：上图中红色管脚部分是 2024 年 10 月 15 日之后生产的芯片才有。

现在带硬件 USB 的 MCU 支持用硬件 USB 下载，因为用的是 USB-HID 通信协议，不需要安装任何驱动。只要 USB 鼠标、USB 键盘能工作，USB-HID 驱动就是好的，不要安装 USB-HID 驱动，免驱。

在 D-/P3.0，D+/P3.1 与 PC-USB 端口连接好的状况下，USB-ISP 下载程序有如下三种模式：

【USB 下载方法一，P3.2 按键，再结合停电上电下载】

1、按下板子上的 P3.2/INT0 按键，就是 P3.2 接地

2、给目标芯片重新上电，不管之前是否已通电。

===电子开关是按下停电后再松开就是上电

等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后，就与 P3.2 状态无关了，这时可以松开 P3.2 按键（P3.2 在用户程序区可做其它任意用途）

===传统的机械自锁紧开关是按上来停电，按下去是上电

3、点击电脑端下载软件中的【下载/编程】按钮（注意：USB 下载与串口下载的操作顺序不同）

下载进行中，几秒钟后，提示下载成功！

【USB 下载方法二，复位管脚低电平复位下载】

USB 连接好并已上电的情况下，外部按键复位也可进入 USB 下载模式，注意：

P4.7-nRST 出厂时默认是 P4.7-I/O 功能，要改为复位功能，需 ISP 烧录时取消设置复位脚用作 I/O 口，停电一次再上电才生效，程序区中用户程序也可改为复位脚或 I/O，这个立即生效。

1、按下 P4.7-nRST 外接的低电平复位按键复位 MCU，

松开复位键，MCU 从系统程序区启动，判断是否要下载用户程序，

等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后

2、点击电脑端下载软件中的【下载/编程】按钮

下载进行中，几秒钟后，提示下载成功！

备注：如果目标 MCU 的用户程序区已有 USB 通讯程序（含 CDC、HID 等），则通过复位脚复位进入 USB 下载模式不被支持，只能通过停电上电冷启动下载或者从用户程序区软复位到系统区下载

【USB 下载方法三，从用户程序区软复位到系统区下载】

USB 连接好并已上电的情况下，从用户程序区软复位到系统区也可进入 USB 下载模式

1、在用户程序区运行软复位到系统区的程序，就是 IAP_CONTR 寄存器送 60H

等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后

2、点击电脑端下载软件中的【下载/编程】按钮

下载进行中，几秒钟后，提示下载成功！

USB 下载 注意事项：

拔插 USB 插头不能代替上面线路图中的电源开关。正常操作步骤如下：

USB 的【Gnd，D+，D-】接好的情况下，按下 P3.2 按键接地，再通过正常的电源开关给 MCU 供电或重新供电，让 MCU 冷启动进入系统程序区，判断是否需要等待电脑端 USB 下载程序。

拔插 USB 插头代替电源开关不能每次都能成功进行 USB 下载的原因：

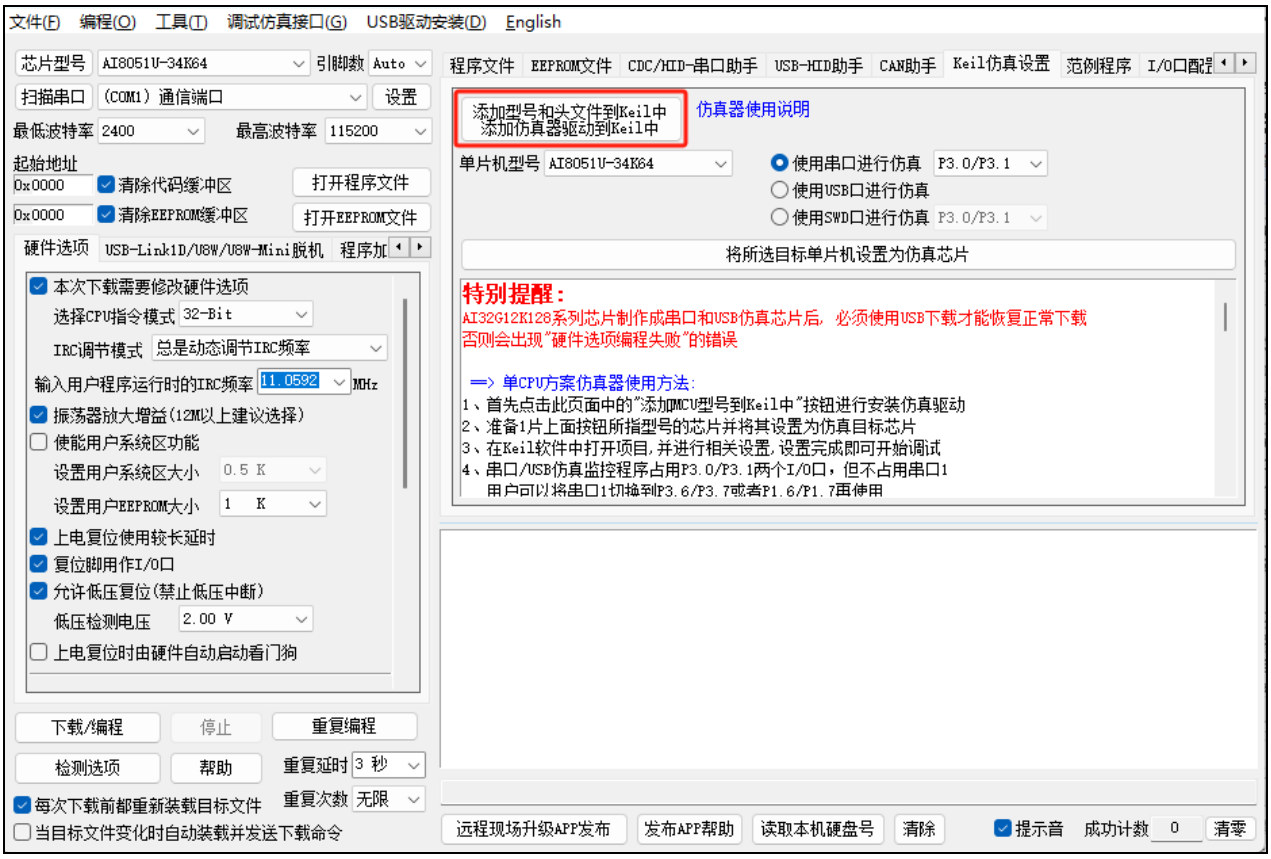
拔插 USB 插头，如【Gnd，USB+5V】已接触好，已供电，而【D+，D-】有一个甚至两个信号线还没有接触好，MCU 已上电，开始跑系统区程序时，发现 USB 还没接触好，则会从系统区软复位到用户程序区跑用户程序，不再进入等待 USB 下载模式，本次就无法顺利进行 USB 下载。

很多人经过多次插拔 USB，才能碰到 1 次【D+，D-】接触好的情况下，【Gnd，USB+5V】才开始接触好，才开始供电，才能成功进入 USB 下载。插 USB 插头代替供电，不能保证【Gnd，D+，D-，USB+5V】的接触顺序，所以，必须使用正常的电源开关，才能确保每次下载都能成功。

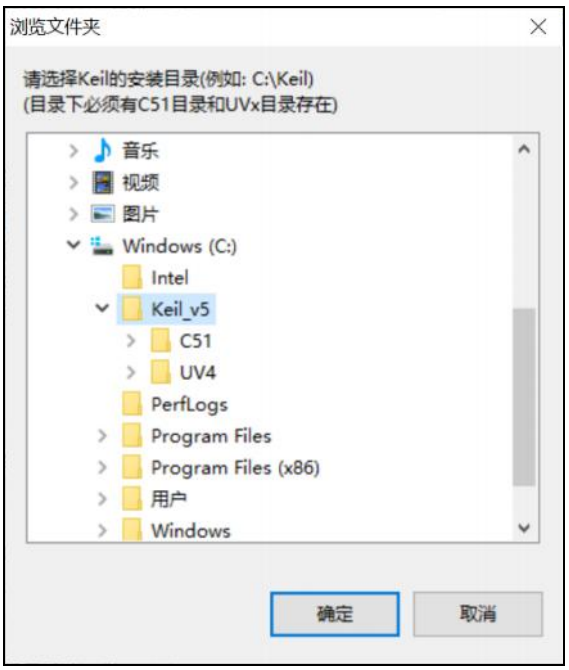
2.4 添加型号和头文件到 Keil

使用 Keil 之前需要先安装 仿真驱动。仿真驱动的安装步骤如下：

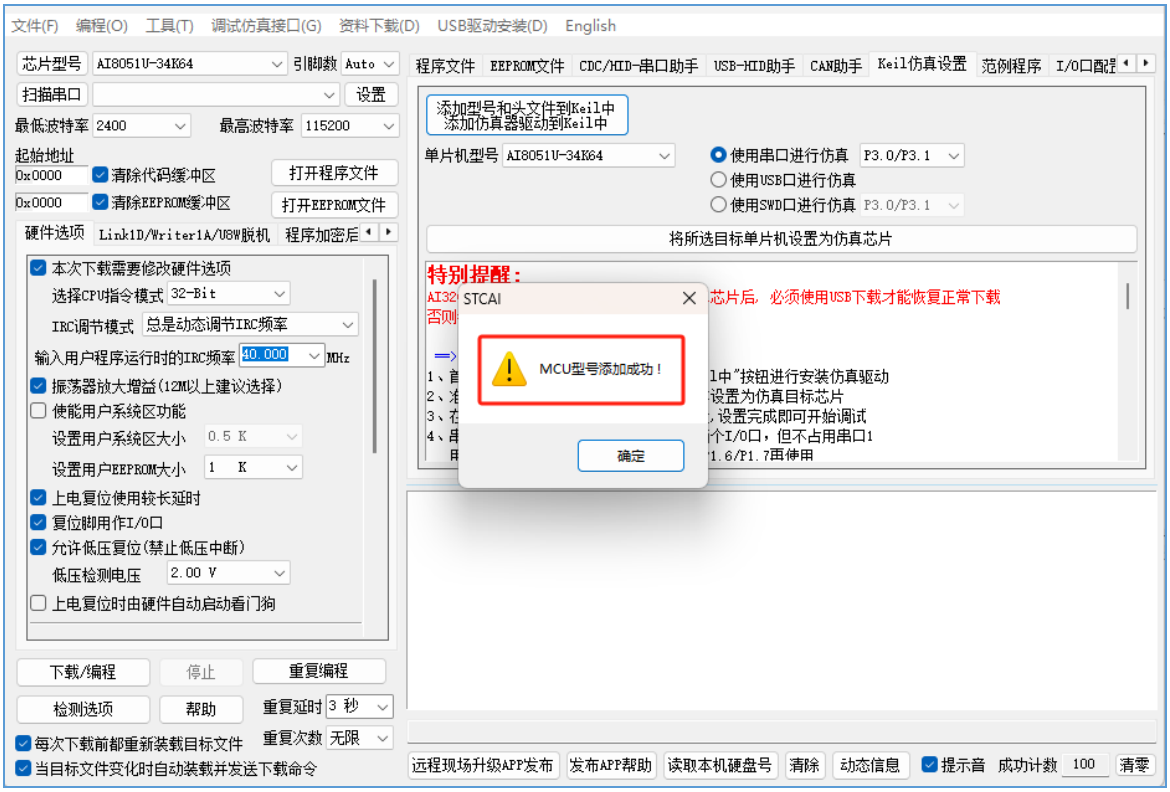
首先打开 ISP 下载软件，然后在软件右边功能区的“Keil 仿真设置”页面中点击“添加型号和头文件到 Keil 中 添加仿真器驱动到 Keil 中”按钮：



按下后会出现如下画面：



将目录定位到 Keil 软件的安装目录，然后确定。安装成功后会弹出如下的提示框：



即表示驱动正确安装了

头文件默认复制到 Keil 安装目录下的“Keil\C51\INC\STC”目录和“Keil\C251\INC\STC”目录中

在 C 代码中使用“#include <Ai8051U.H>”或者“#include "Ai8051U.H"”均可正确使用

2.5 Ai8051U 的 8 位/32 位【头文件，编译器，ISP 烧录时的设置】说明

Ai8051U 型号可支持 32 位模式和 8 位模式。如果用户需要使用 32 位模式，则需要安装 Keil 的 C251 编译器；如果用户需要使用 8 位模式，则需要安装 Keil 的 C51 编译器；如果需要同时使用 32 位模式和 8 位模式，则 C251 和 C51 都需要安装。

“AIapp-ISP”下载软件的如下界面可自动安装 Keil 的仿真驱动程序和所有系列的头文件

对于 Ai8051U 系列：

如果有安装 Keil C51 编译器，则会在 Keil 安装目录中的“C51\INC\STC”目录，安装“Ai8051U.h”头文件，这个是 8 位 8051 的文件；

如果有安装 Keil C251 编译器，则会在 Keil 安装目录中的“C251\INC\STC”目录，安装“Ai8051U.h”头文件。这个是 32 位 8051 的同名但放在不同目录的实际不同的 32 位 8051 的头文件。

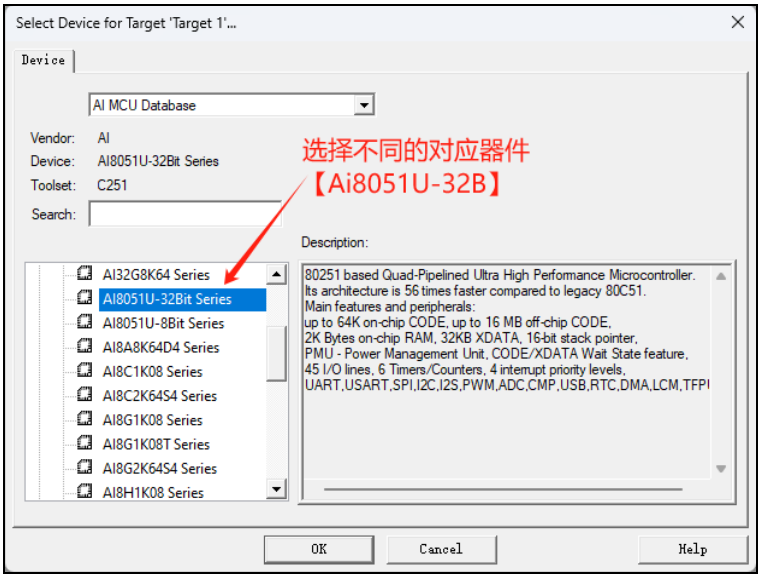
特别注意:

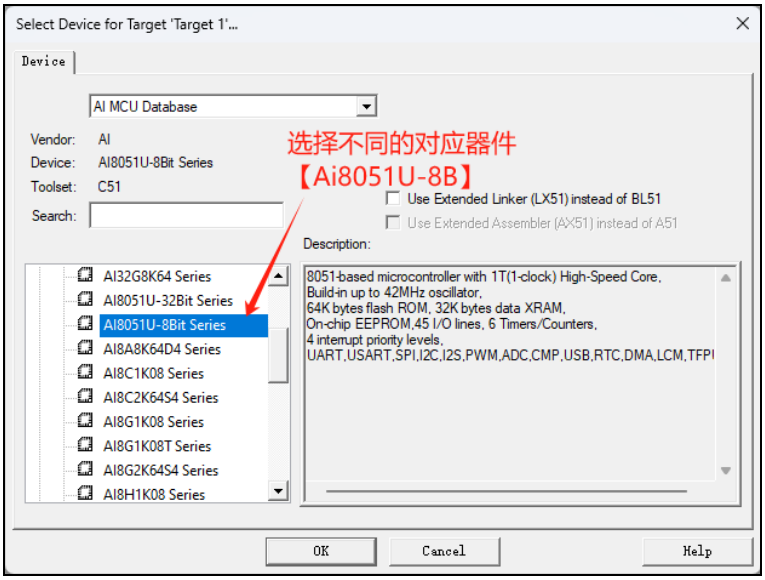
我们 ISP 软件在 Keil 中添加的 Ai8051U 系列，32 位 8051 的头文件，和 8 位 8051 的头文件的文件名相同，但放在不同的目录，是不同的文件，内容实际不相同，不可混用，不建议用户自行修改和复制。

如果在 Keil 创建项目时选择的器件是“Ai8051U-8B”,即 8 位模式,且在代码中包含“Ai8051U.h”，Keil C51 编译器会自动在“C51\INC\STC”目录中查找 8 位模式的“Ai8051U.h”头文件。

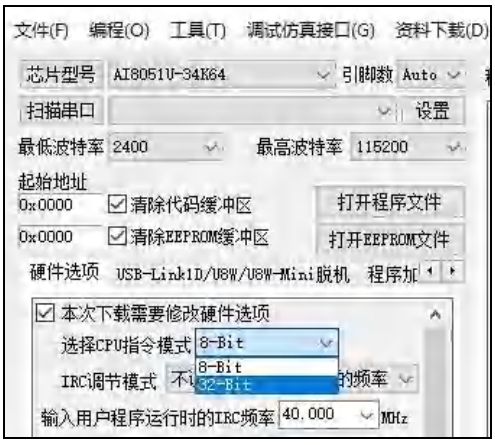
如果在 Keil 创建项目时选择的器件是“Ai8051U-32B”，即 32 位模式，在代码中包含“Ai8051U.h”， Keil C251 编译器会自动在“C251\INC\STC”目录中查找 32 位模式的“Ai8051U.h 头文件。

所以对于用户来说,只需要在创建项目时选择好正确的 Ai8051U 模式，就是在 Keil 创建项目时选择不同的对应器件：【Ai8051U-8B】，【Ai8051U-32B】





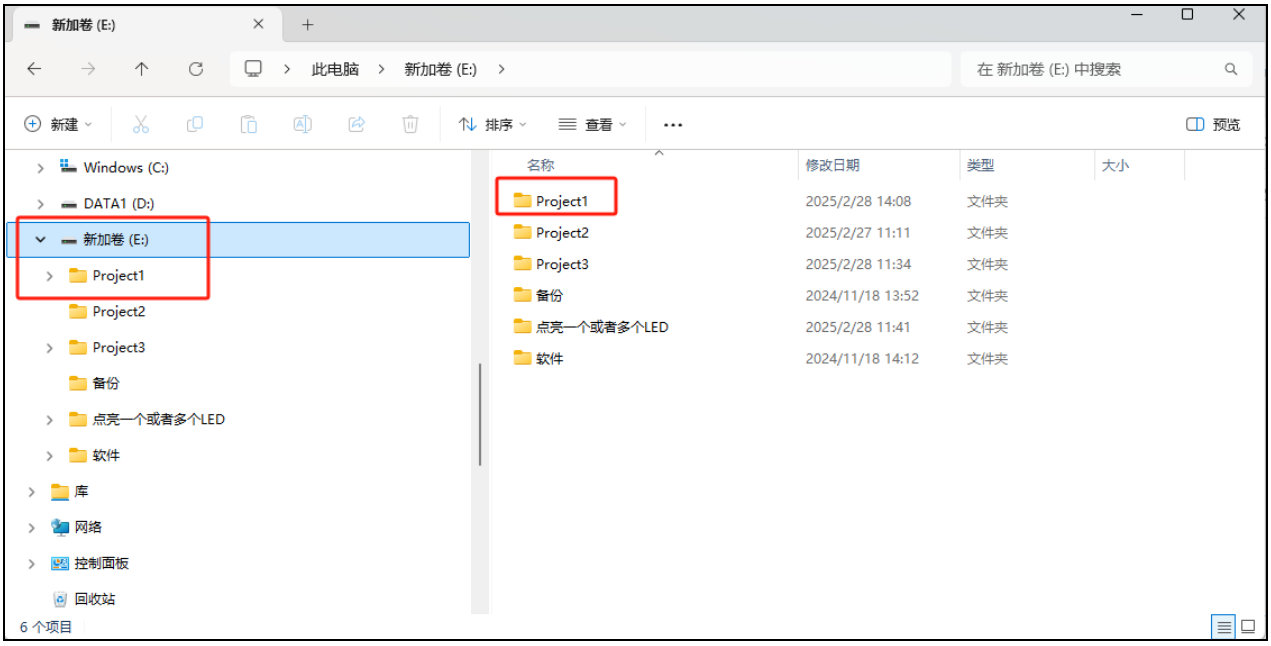
对应的 Keil C51 编译器或 Keil C251 编译器就会自动搜索正确的头文件，用户不用担心 32 位模式的头文件和 8 位模式的头文件名称相同会弄错，实际是地址不同的同名文件而已。编译器生成的对应目标文件，用 AIapp-ISP 烧录时，指定对应的模式 8-Bit / 32-Bit，如下图：



2.6　新建一个 32 位 8051 项目，在 Keil 中

2.6.1　准备工作，见前面几小节

如何能建立自己的 Project，需要做的准备工作：
一、下载、安装 Keil C251/C51，见前面几小节；
二、下载、安装 AIapp-ISP，见前面几小节；
三、添加型号和头文件，见前面几小节；
四、建立一个文件夹存放你的 Projects。
在 E 盘创建一个文件夹 Project1，用于存放自己的项目文件

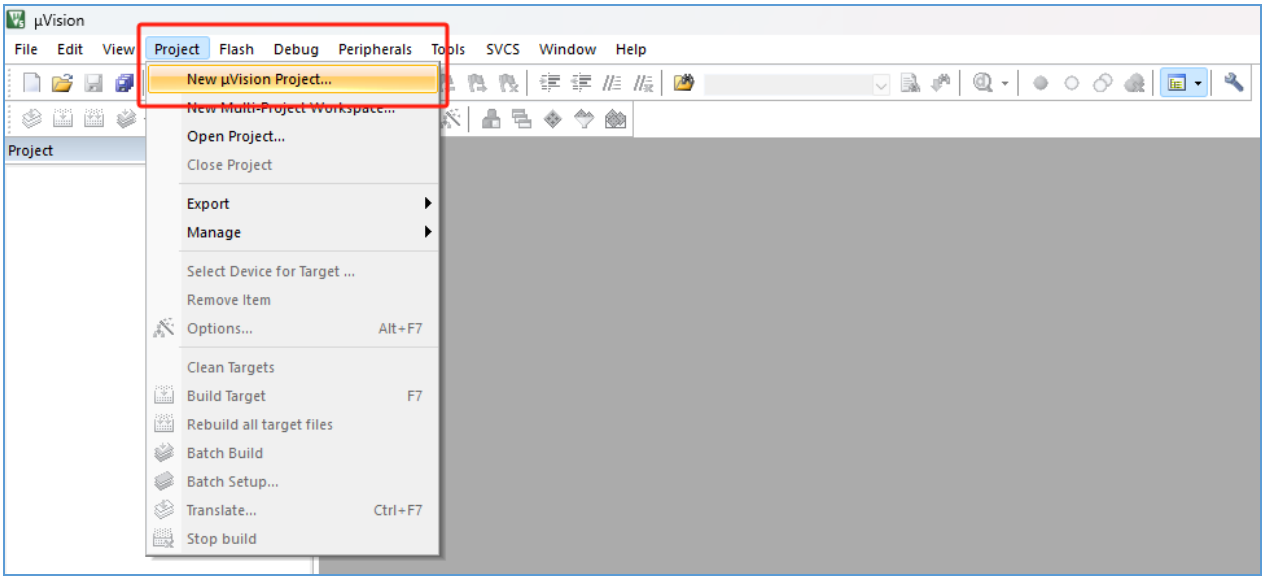


2.6.2 Keil 新建一个 32 位 8051 项目

2.6.2.1 新建工程

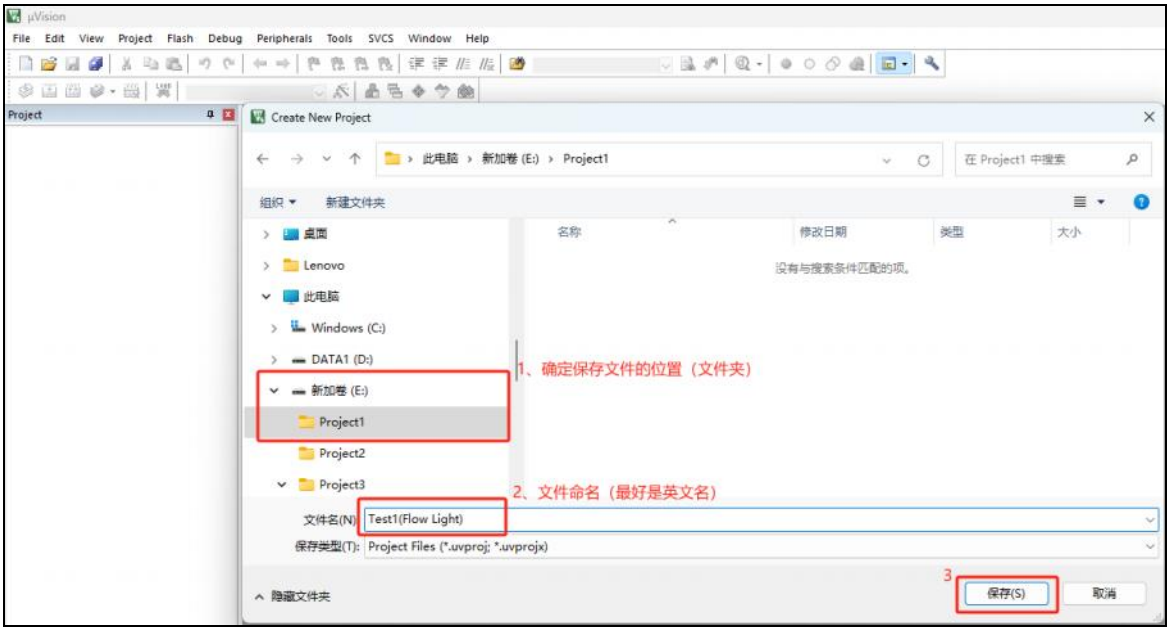
打开 Keil 软件：

1）在上方的菜单栏 Project 标签中，找到 New uVision Project 选项，开始新建工程。



2）将文件存入刚刚建好的文件夹 E:\project1 中

再给项目起个名称（方便查找）（尽量是英文的，如 Test…），点击“保存”。如下图：



3）选择单片机

在“Select Device for Target’Target1’”选项卡中：

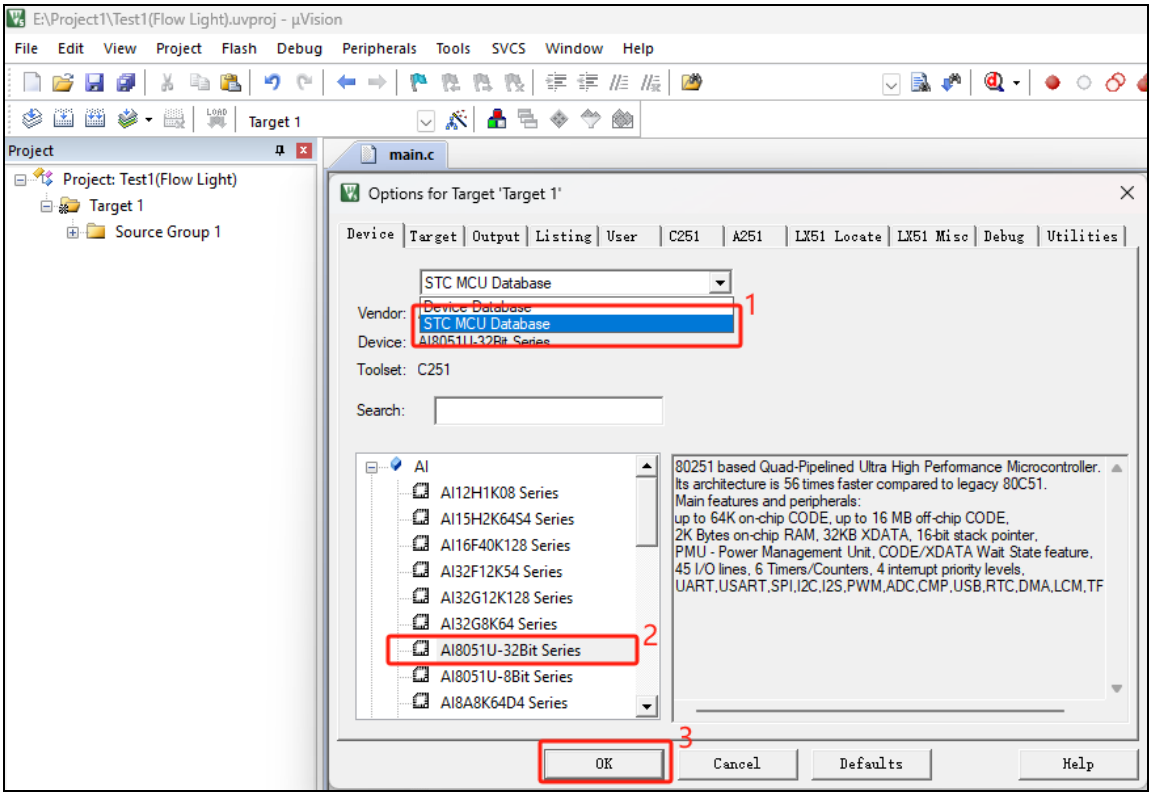
- 选择 STC MCU Database
- 然后展开 AI 选项，选择“Ai8051U-32Bit Series”选项（详细请查看第 6.5）

  注意：这是建立 Ai8051U 的 32 位 8051 工程项目，会调用 Keil C251 编译器来编译本项目。

  注意：如果在此处选择了“Ai8051U-8Bit Series”，这是建立 Ai8051U 的 8 位 8051 工程项目，

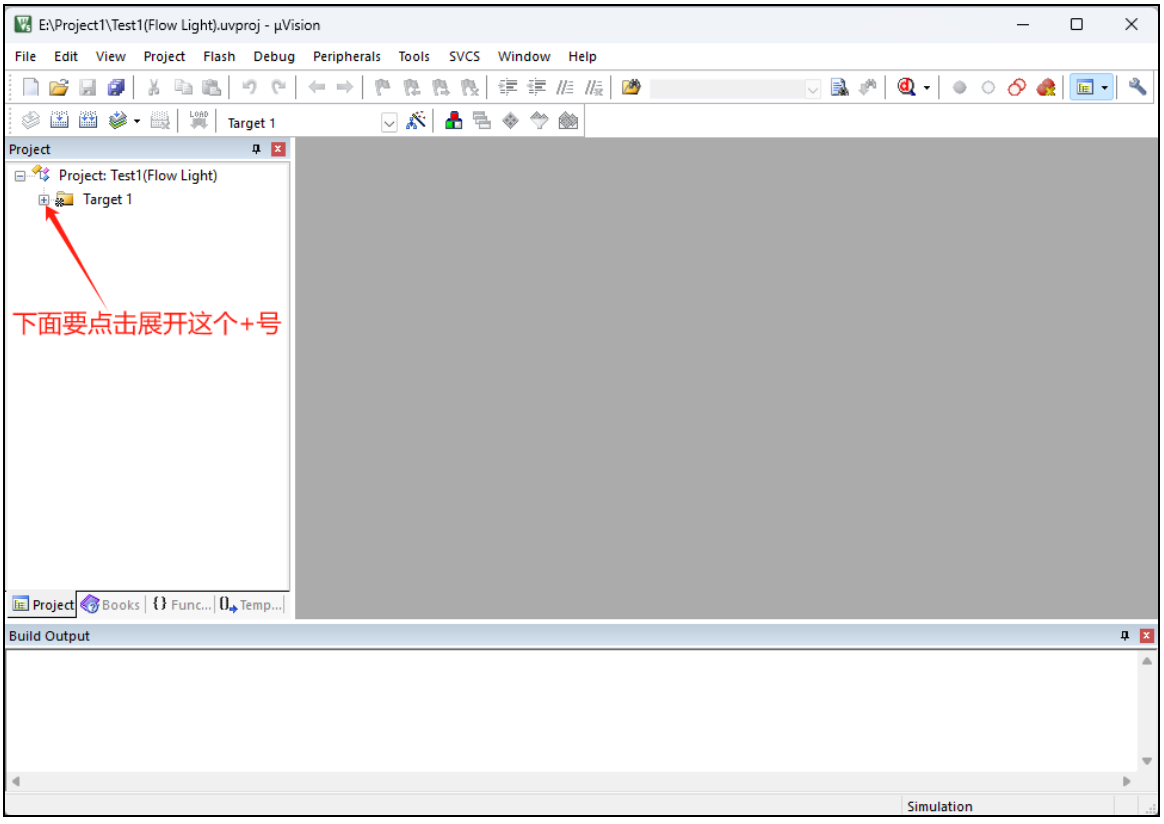
会调用 Keil C51 编译器来编译 8 位的项目。

➲ 最后点击‘OK’以完成单片机选择。



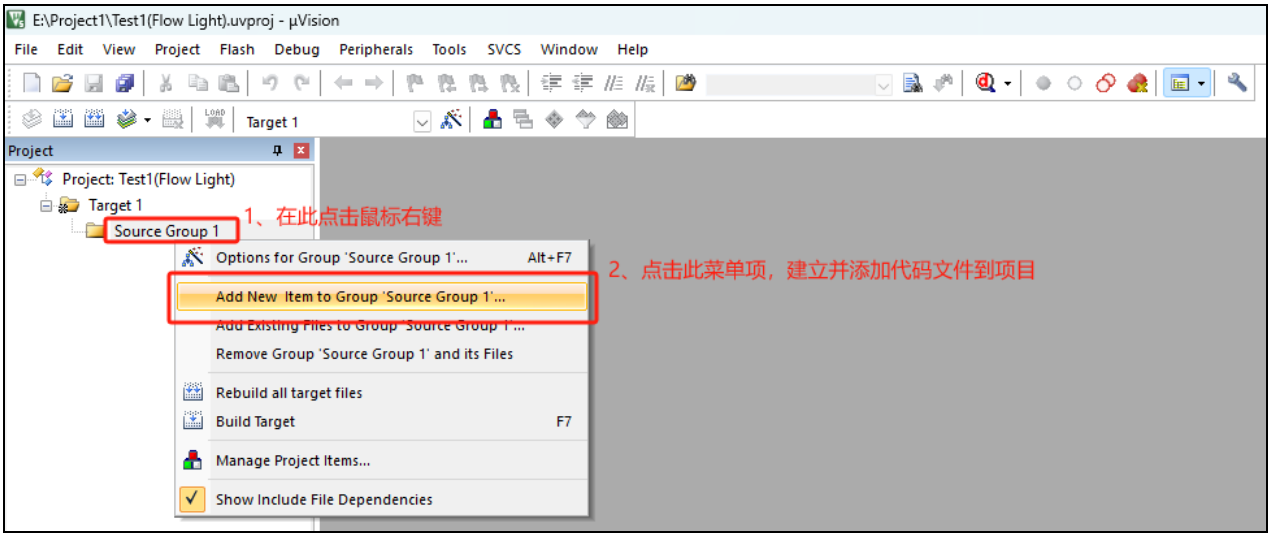
4）选择完单片机后，要给工程项目增加一个主程序文件

选择完单片机后，工程界面是一片灰白。这时要给工程项目增加一个主程序文件，可以命名为 main.c 文件



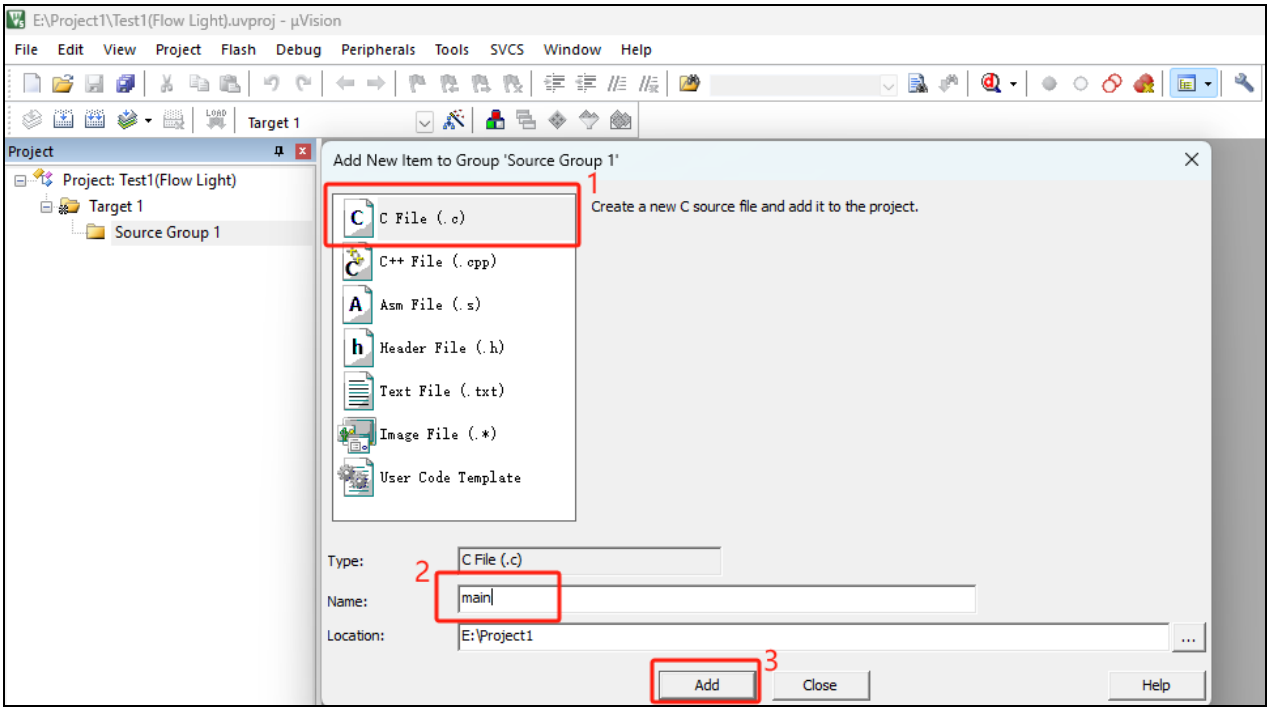
5）给工程项目增加一个主程序文件。

点击 Project 窗口上 Target1 的+，出现“Source Group1”，右键点击“Source Group1”，出现下图。



再选择快捷菜单中的“Add New Item to Group ‘Source Group 1’”

出现如下界面提示：



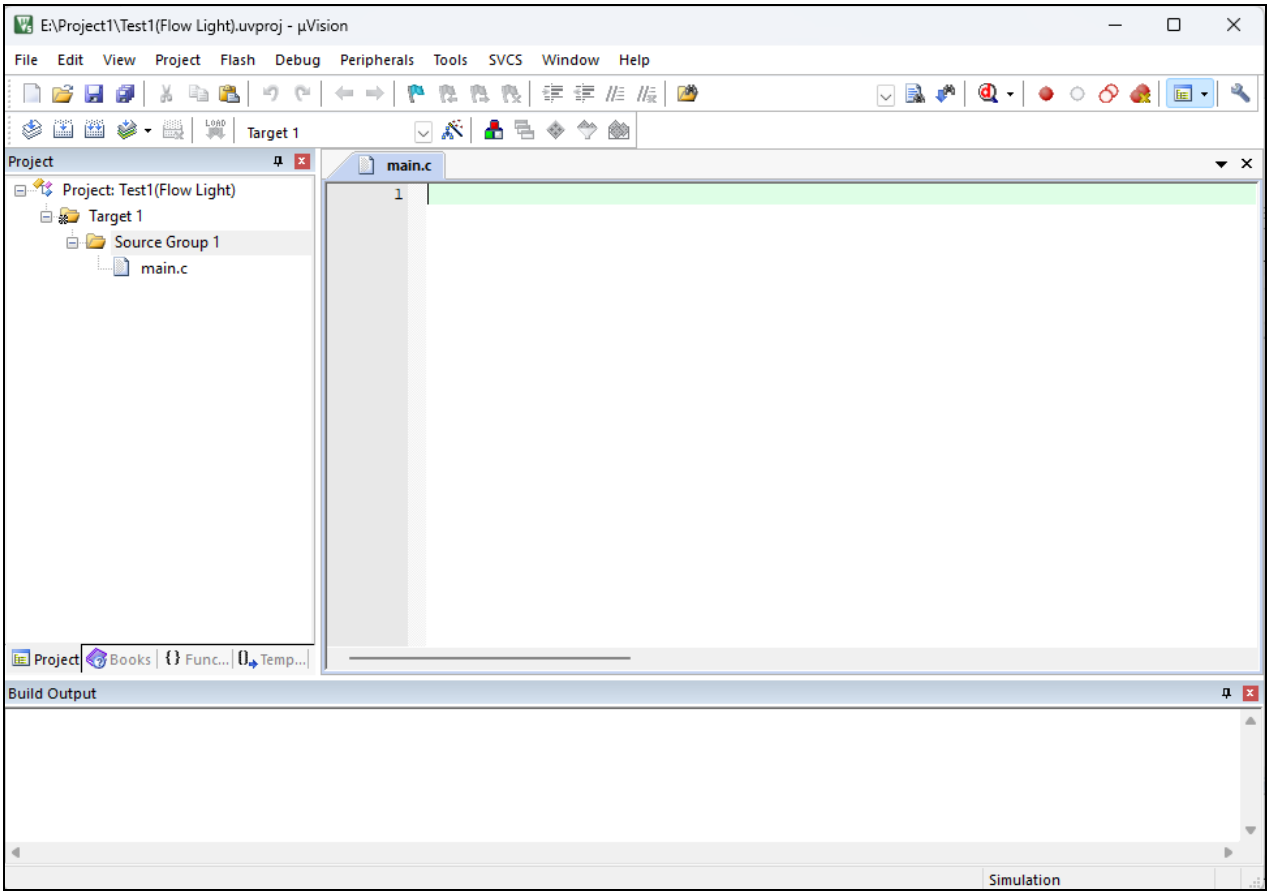
在上图中选择文件类型是：*.c

在 Name 输入框中输入文件名是：main 或其他任意文件名，最好是英文名

在 Location 输入框中选择当前项目的目录

然后点击 Add 按钮，将该新文件加入到当前项目。

至此，生成了一个空的 main.c 主程序文件并且已经加入到当前项目。可以在该文件中编写代码了。后续的实验例程会演示如何将程序代码输入到其中。

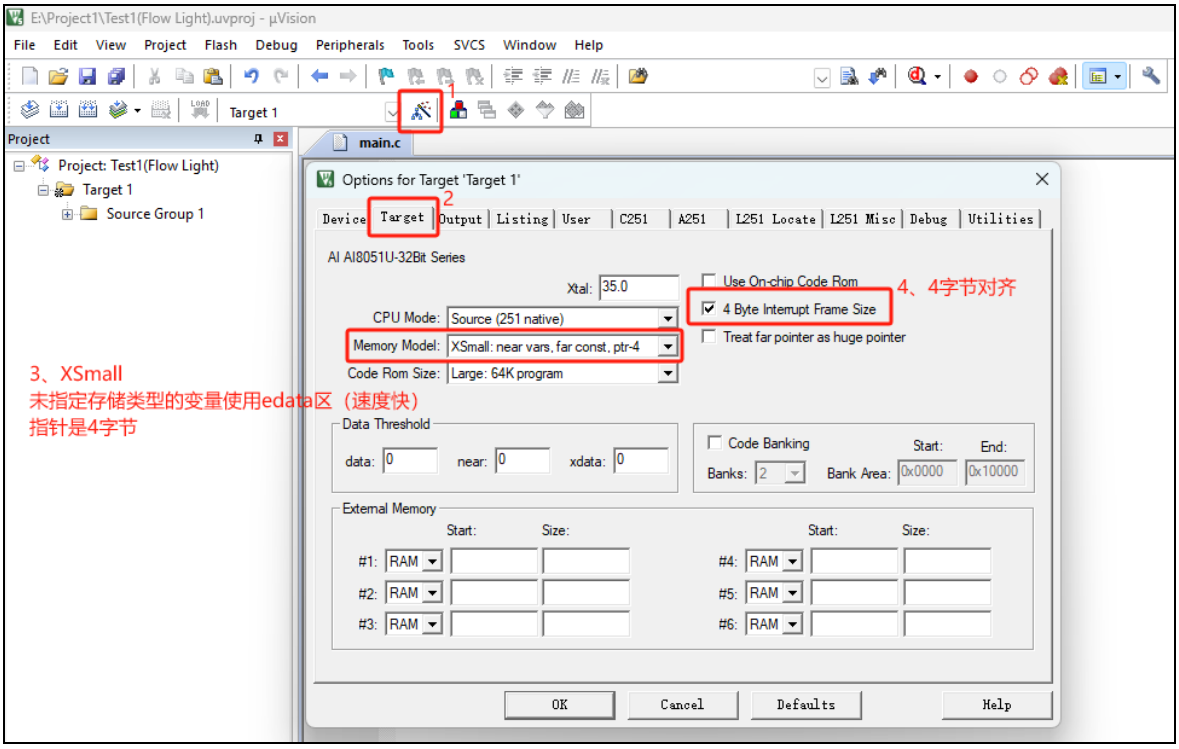


当然，在编写代码前，最好先进行 Project 的各项基础设置。

2.6.2.2　32 位 8051 工程项目的初步简单设置，必须按此流程过一遍

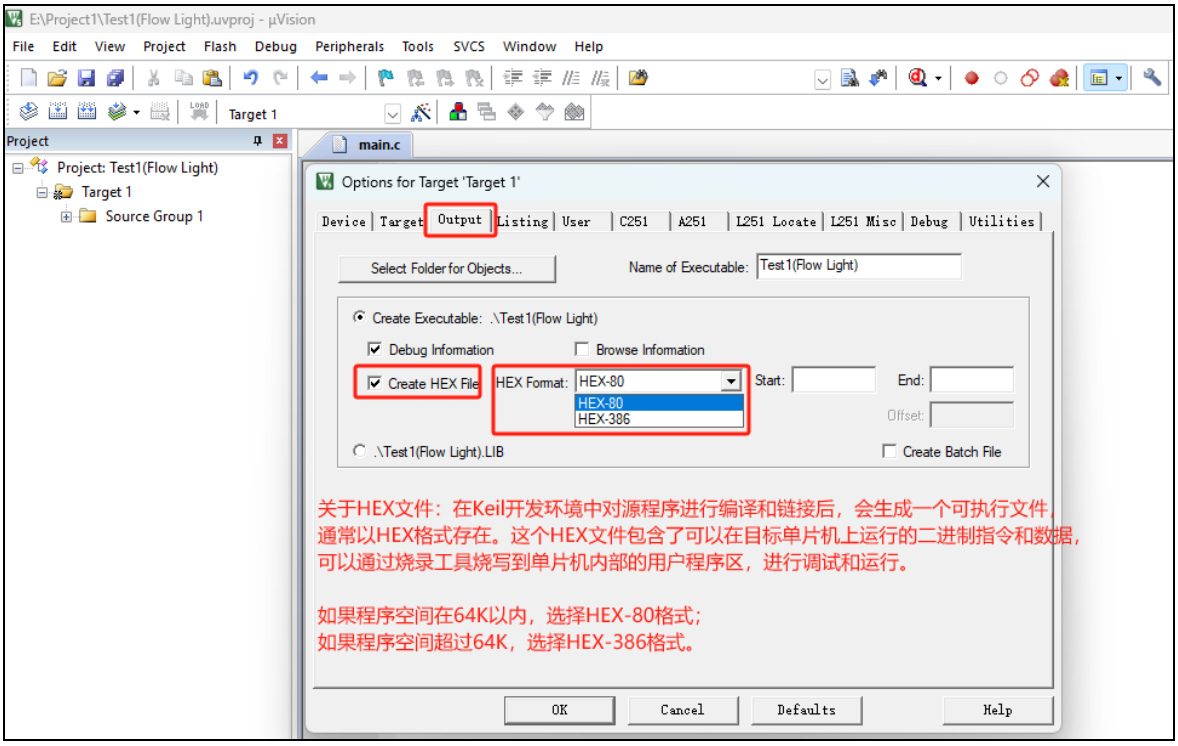
在菜单栏下方找到一个类似魔术棒的工具，就是任务选项的设置。

1）Target 选项卡--Memory Model 选择 XSmall 模式、勾选 4 字节对齐



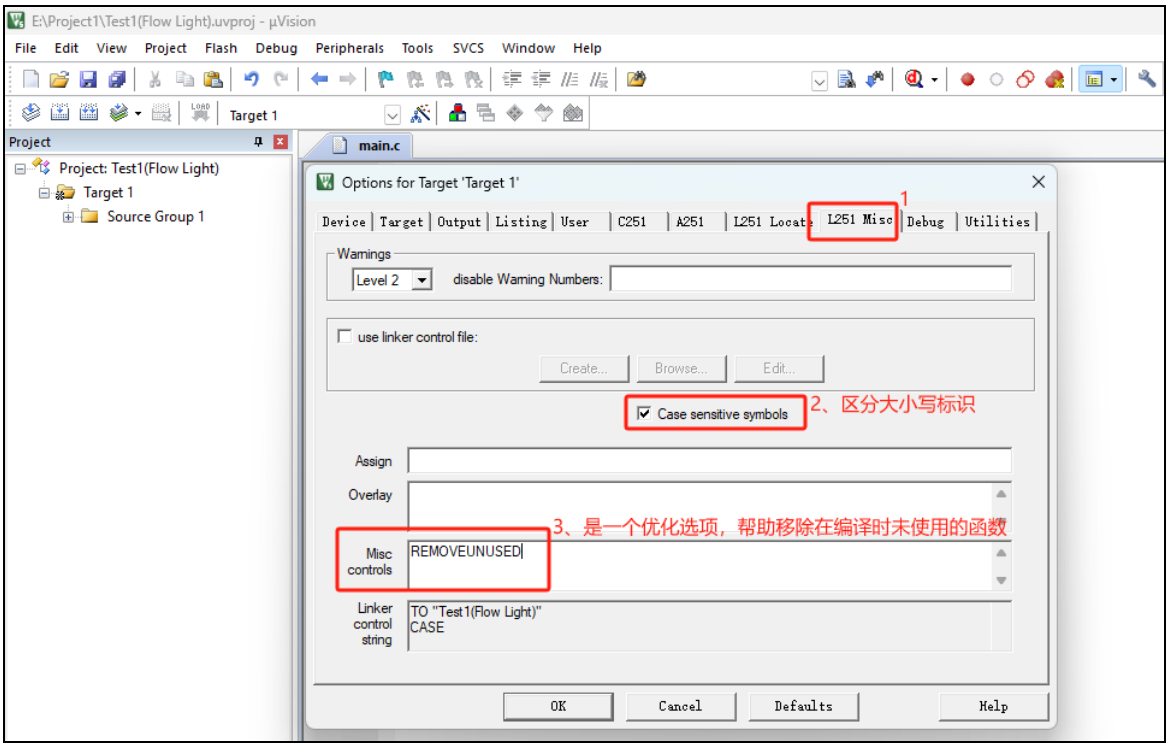
2）Output 选项卡--勾选 Create HEX File，选择 HEX-80

切换到 Output 选项卡，勾选‘Creat HEX File’选项，否则编译后是不会创建 HEX 文件的。HEX 文件是可以通过烧录工具烧录到单片机用户程序区。



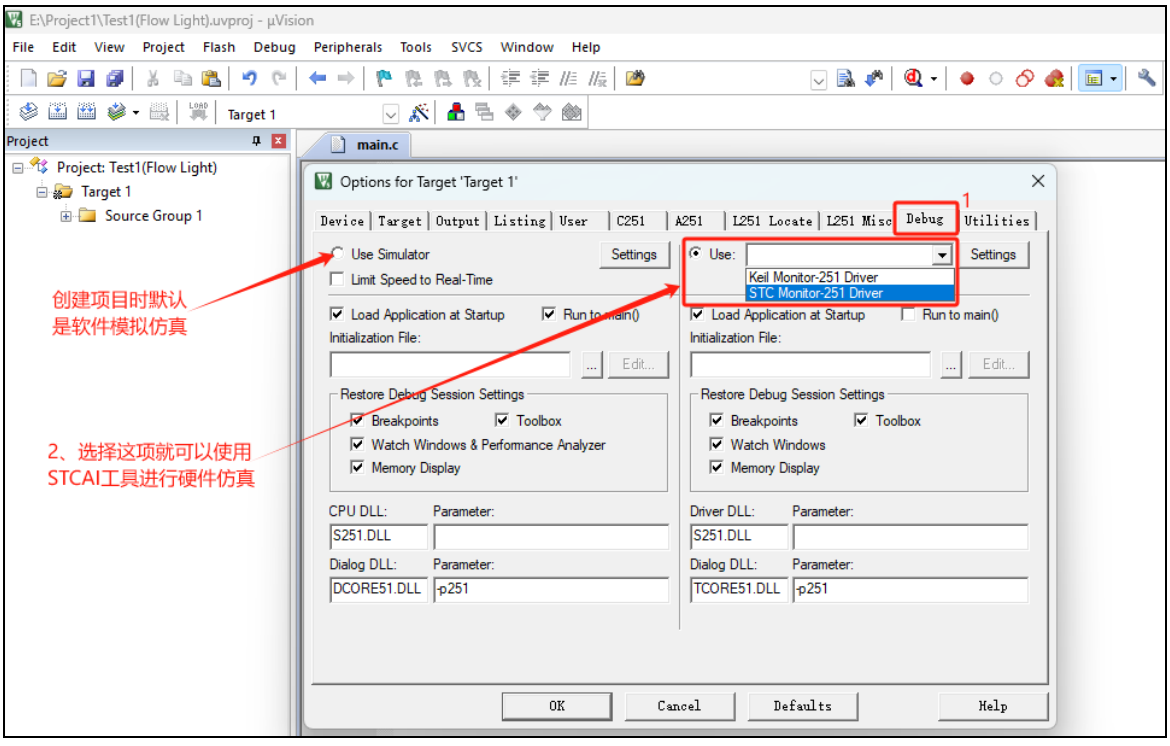
3）L251 Misc 选项卡--在 Misc controls 输入框输入‘REMOVEUNUSED’

切换到 L251 Misc 选项卡，在 Misc controls 输入框中，输入大写的‘REMOVEUNUSED’，移除未使用的函数（如：库函数），以实现不使用的函数不进行链接。



4）Debug 选项卡--使用 STCAI 工具进行硬件仿真设置

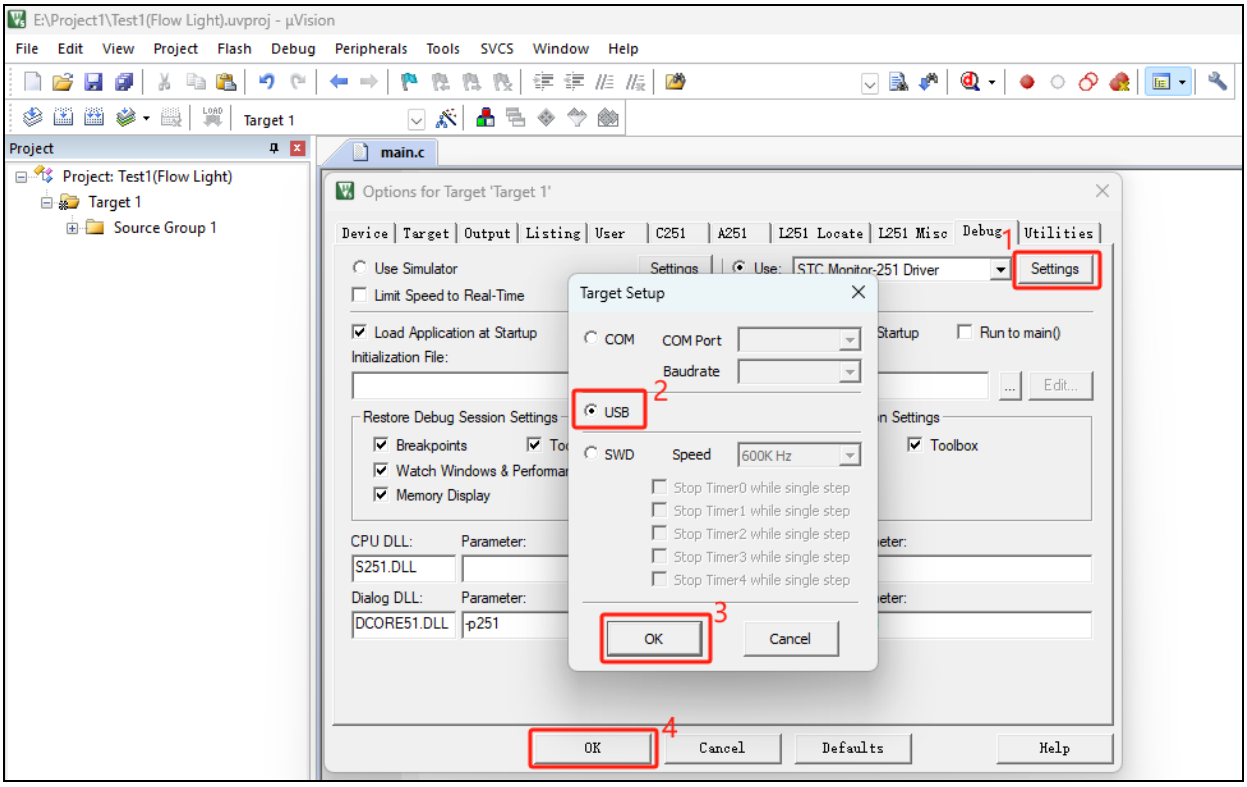
切换到 Debug 选项卡，创建项目时，Keil 默认是软件模拟仿真。我们可以借助 STCAI 的工具进行硬件仿真，如下图：



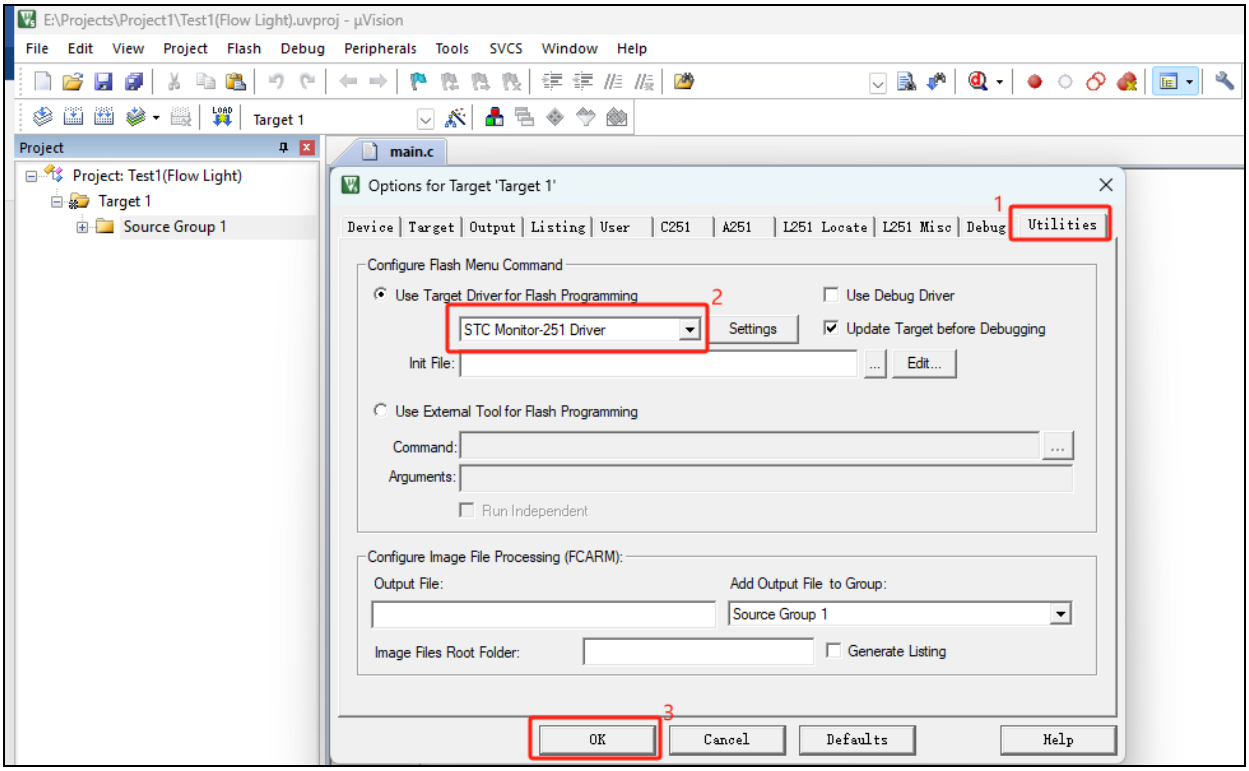
Ai8051U 如工作在 Ai8051U-32Bit 模式，可以 USB 仿真和 COM 串口进行仿真，SWD 仿真暂不支持；
Ai8051U 如工作在 Ai8051U-8Bit 模式，暂不支持任何仿真。

如果是选择通过芯片自带的硬件 USB 通讯仿真，则如下图设置：

Ai8051U 如工作在 Ai8051U-32Bit 模式，可以 USB 仿真和 COM 串口进行仿真，SWD 仿真暂不支持；



切换到“Utilities”选项卡，选择“STC Monitor-251 Driver”，点击“OK”，完成设置。



至此，工程项目的各种基础选项设置结束。可以进入下一节，在程序文件中添加实际的代码。

如果不使用仿真，USB 仿真设置可以跳过。

2.7 堆栈不用用户设置@Ai8051U-32Bit

对于 Ai8051U-32 位的 C 语言程序，如无特殊需求，**堆栈是不用用户设置的**。

首先，编译器会将堆栈的起始地址设置在用户代码中实际使用的 edata 结束的位置。

比如：用户代码中实际使用了 1K 字节的 edata，则编译器会将堆栈指针 SPX 设置为 0x3FF

Ai8051U-32 的内部有 2K 的 edata，从 0x400 ~ 0x7FF 的 edata 全部都是堆栈，虽然编译器默认只保留了 256 字节，但实际程序运行是超过了 256，也不会有任何问题。

除非程序中有使用绝对地址访问 edata，否则用户代码使用的 edata 和堆栈不会有任何冲突

总结：

程序中如果需要定义比较大的数组，建议单独使用 xdata 关键字将数组分配到 xdata 区域，

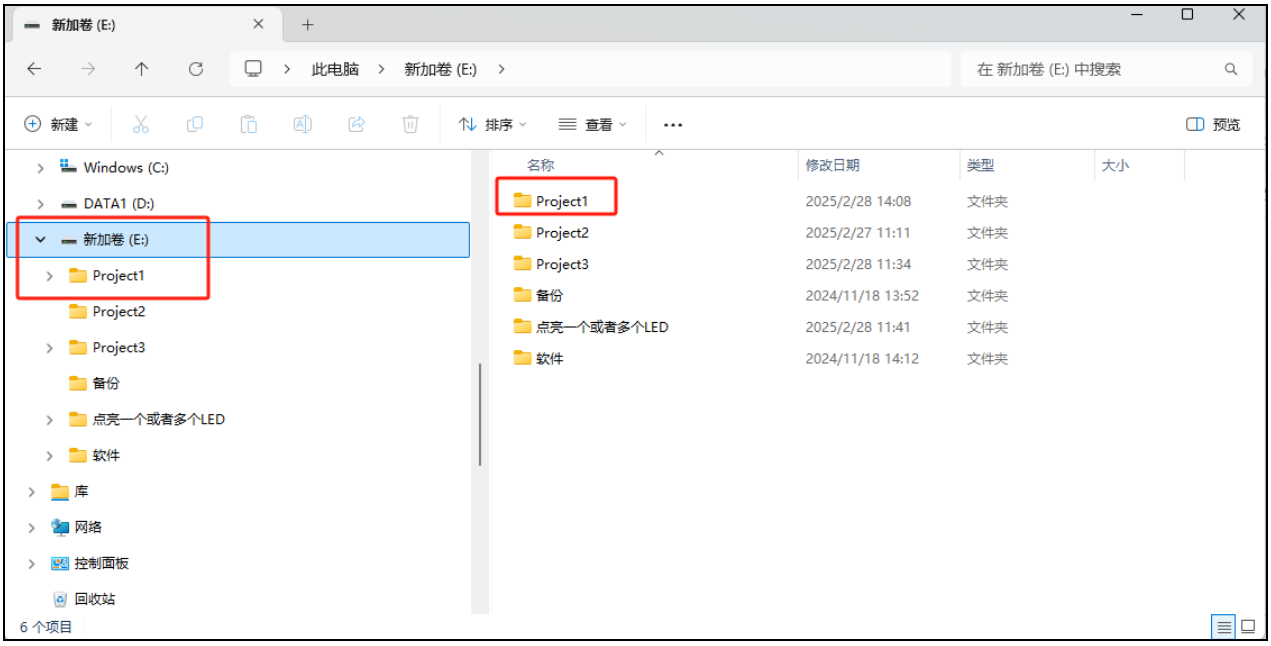
比如：int xdata buffer[256];

memory model 还是尽量使用官方建议的 XSmall 模式

2.8　新建一个 8 位 8051 项目，在 Keil 中

2.8.1　准备工作，见前面几小节

如何能建立自己的 Project，需要做的准备工作：
一、下载、安装 Keil C251/C51，见前面几小节；
二、下载、安装 AIapp-ISP，见前面几小节；
三、添加型号和头文件，见前面几小节；
四、建立一个文件夹存放你的 Projects。
在 E 盘创建一个文件夹 Project1，用于存放自己的项目文件

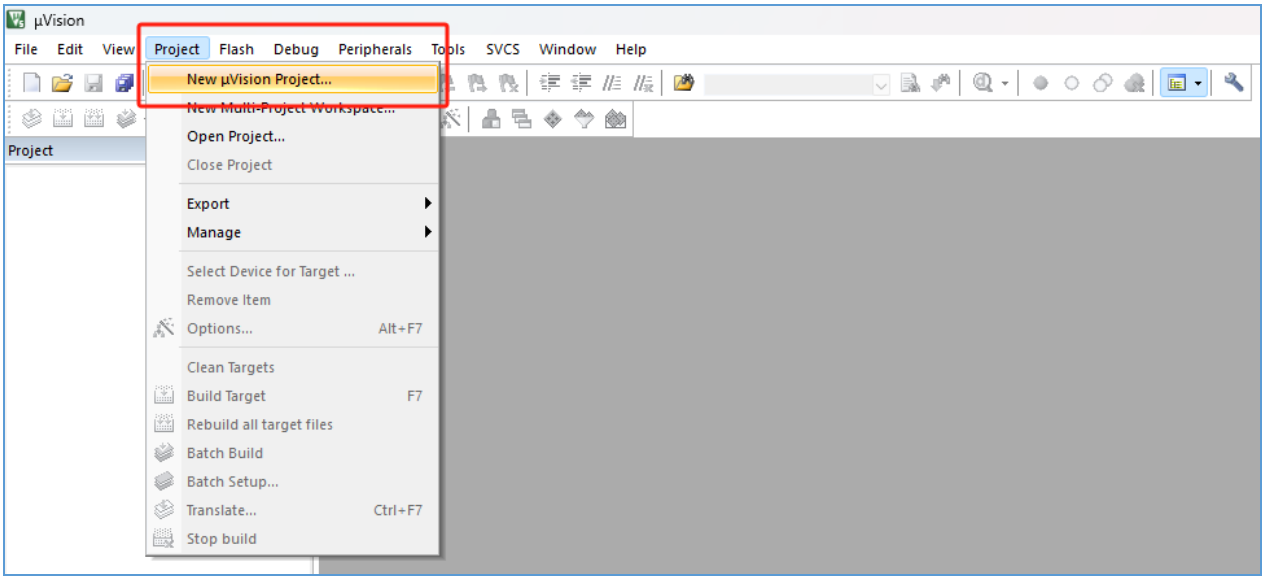


2.8.2 Keil 新建一个 8 位 8051 项目

2.8.2.1 新建工程

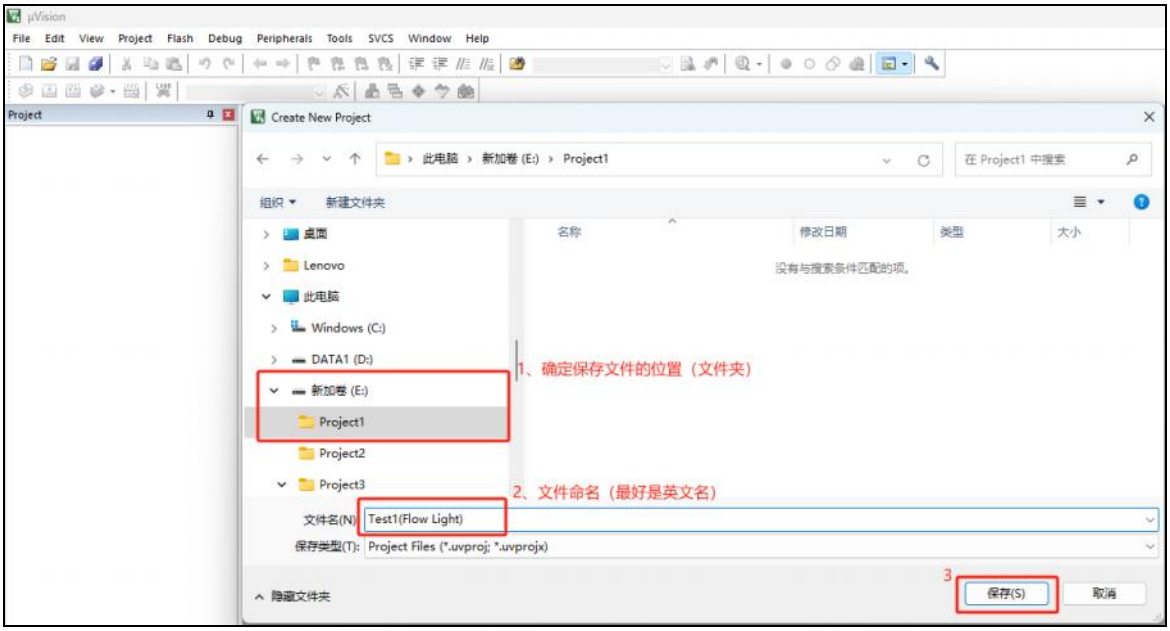
打开 Keil 软件

1）在上方的菜单栏 Project 标签中，找到 New uVision Project 选项，开始新建工程。



2）将文件存入刚刚建好的文件夹 E:\project1 中

再给项目起个名称（方便查找）（尽量是英文的，如 Test…），点击“保存”。如下图：



3）选择单片机

在“Select Device for Target’Target1’”选项卡中：

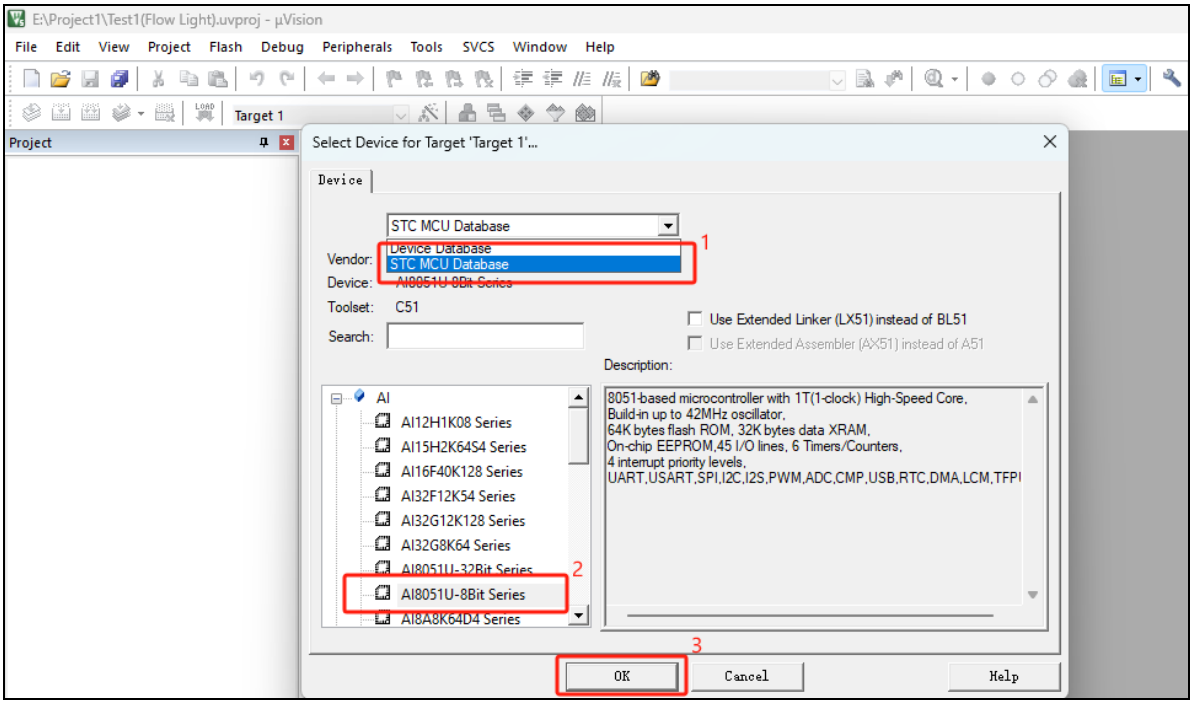
- 选择 STC MCU Database
- 然后展开 AI 选项，选择“Ai8051U-8Bit Series”选项（详细请查看第 6.5）

  注意：这是建立 Ai8051U 的 8 位 8051 工程项目，会调用 Keil C51 编译器来编译本项目。

  注意：如果在此处选择了“Ai8051U-32Bit Series”，这是建立 Ai8051U 的 32 位 8051 工程项目，

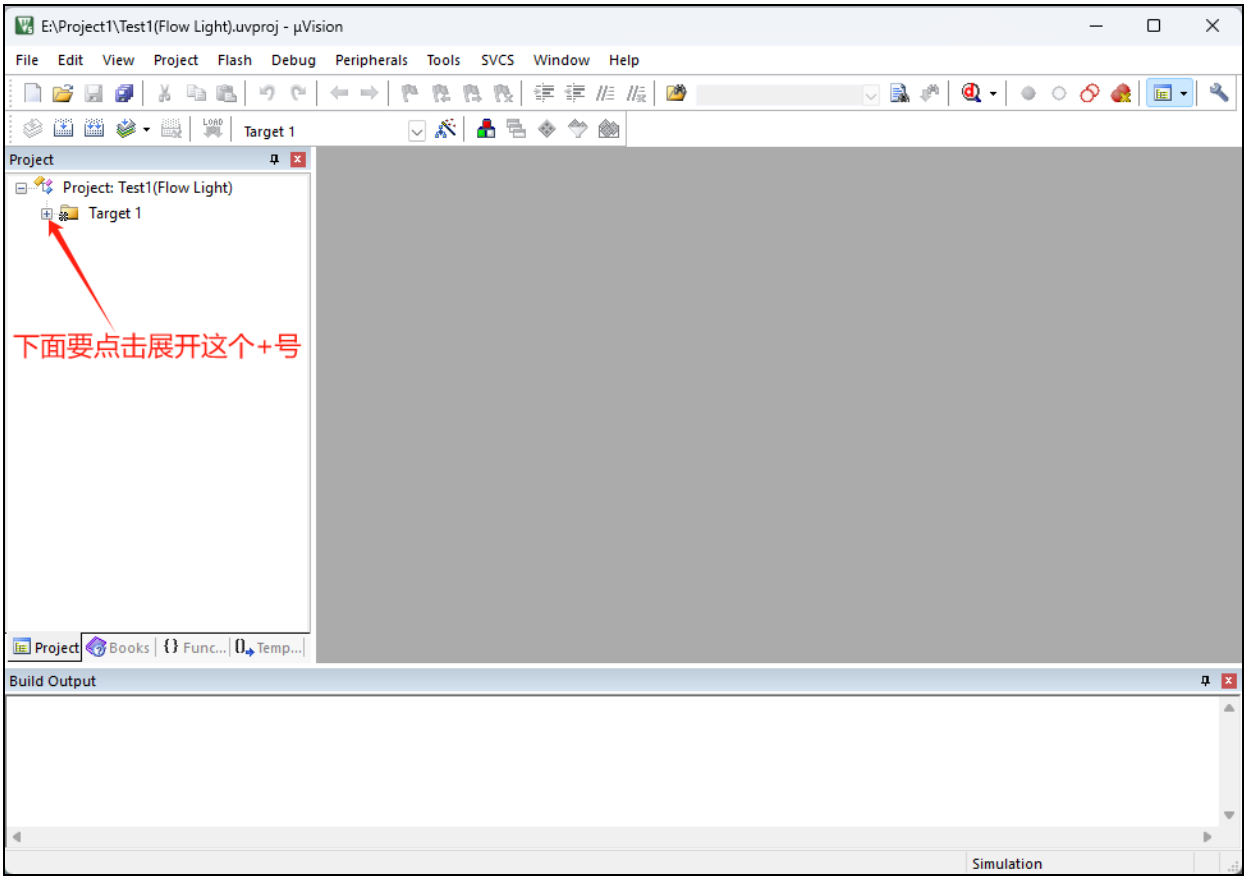
会调用 Keil C251 编译器来编译 32 位的项目。

➲ 最后点击‘OK’以完成单片机选择。



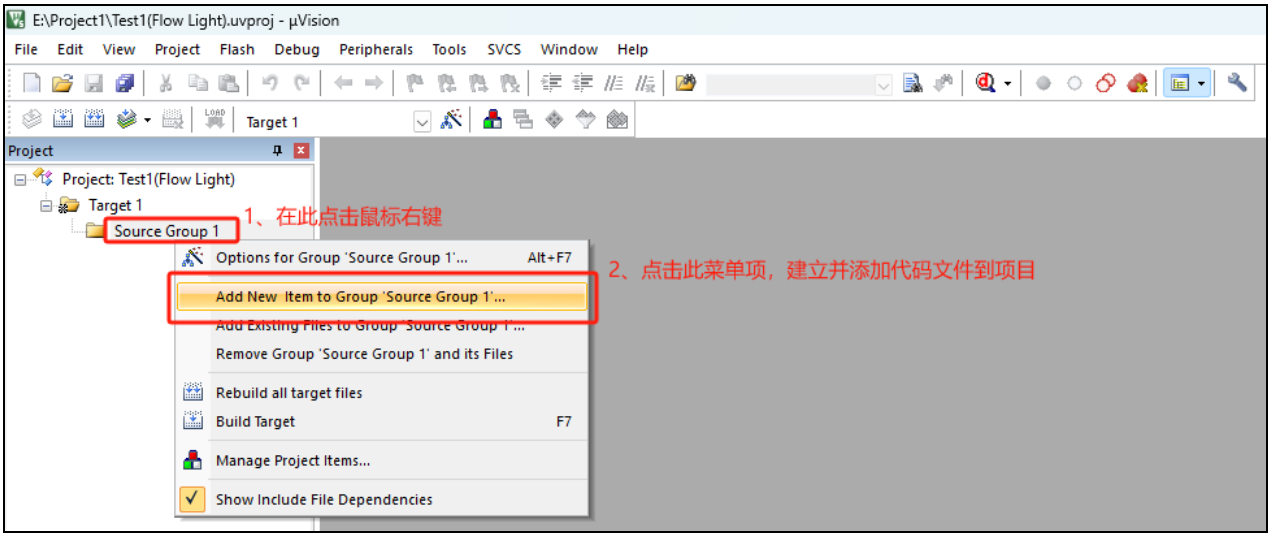
4）选择完单片机后，要给工程项目增加一个主程序文件

选择完单片机后，工程界面还是一片灰白的。这时要给工程项目增加一个主程序文件，可以命名为 main.c 文件



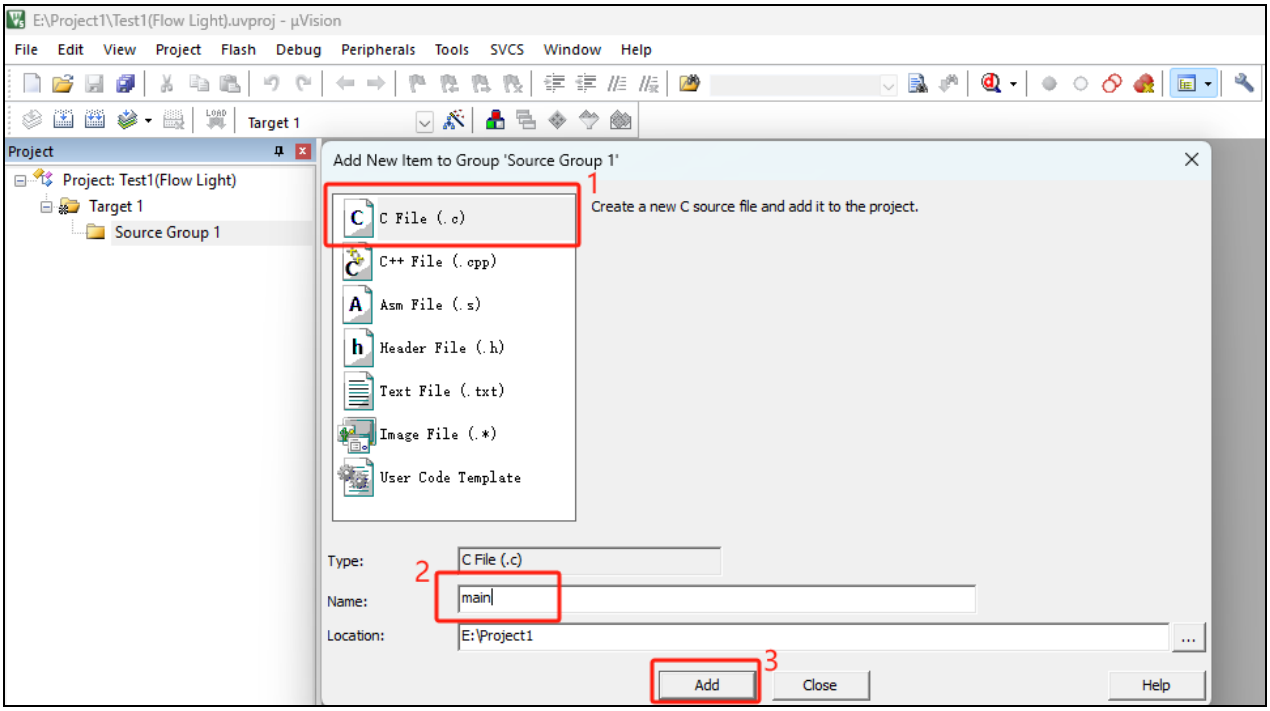
5）给工程项目增加一个主程序文件。

点击 Project 窗口上 Target1 的+，出现“Source Group1”，右键点击“Source Group1”，出现下图。



再选择快捷菜单中的“Add New Item to Group ‘Source Group 1’”

出现如下界面提示：



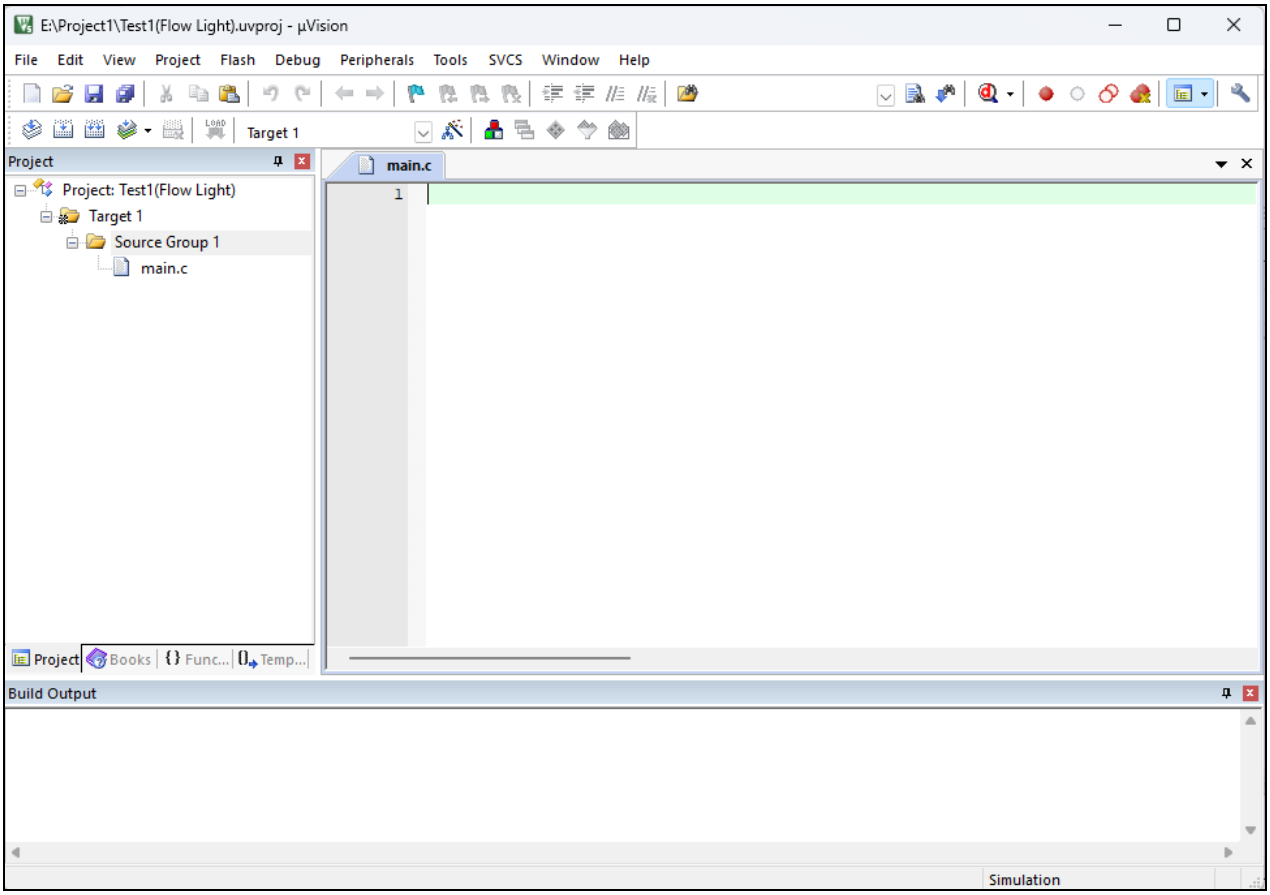
在上图中选择文件类型是：*.c

在 Name 输入框中输入文件名是：main 或其他任意文件名，最好是英文名

在 Location 输入框中选择当前项目的目录

然后点击 Add 按钮，将该新文件加入到当前项目。

至此，生成了一个空的 main.c 主程序文件并且已经加入到当前项目。可以在该文件中编写代码了。后续的实验例程会演示如何将程序代码输入到其中。

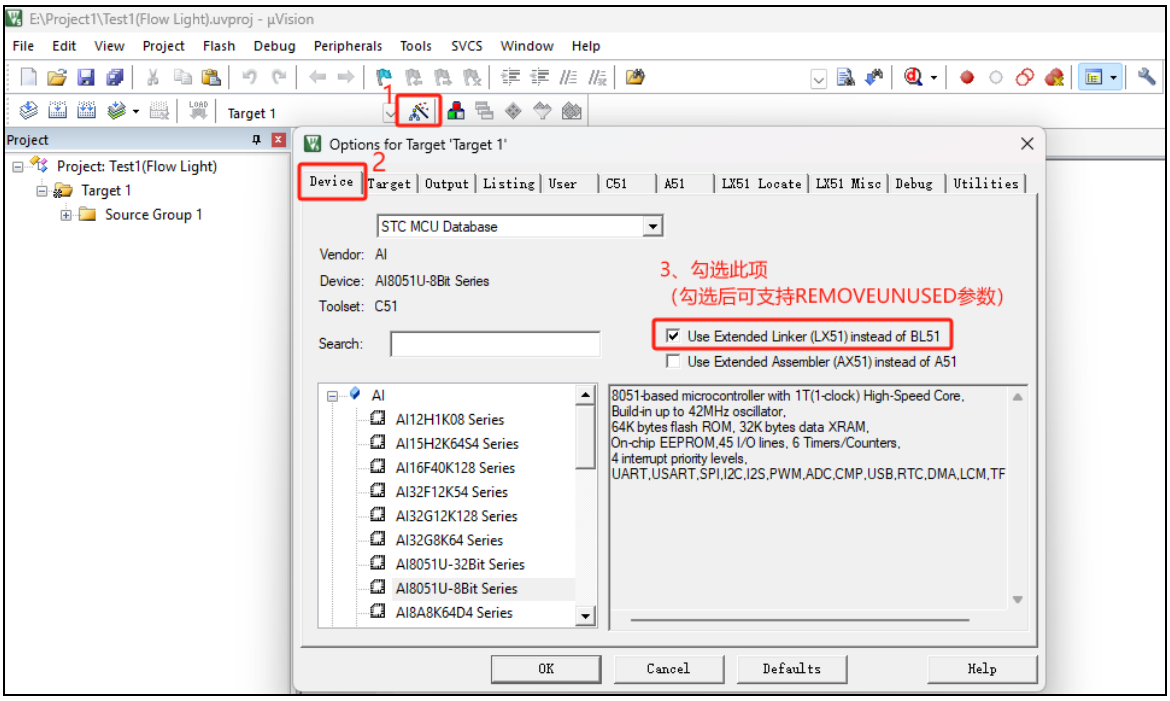


当然，在编写代码前，最好先进行 Project 的各项基础设置。

2.8.2.2　8 位 8051 工程项目的各种基础选项设置。

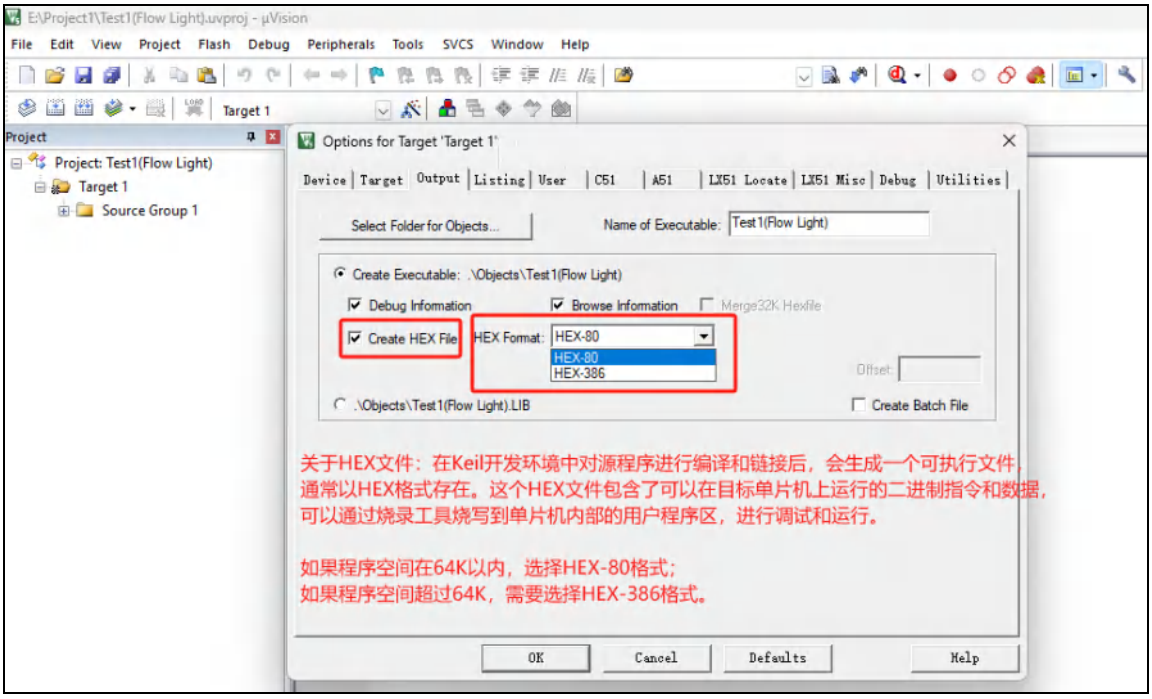
在菜单栏下方找到一个类似魔术棒的工具，就是任务选项的设置。

1）Device 选项卡--勾选 Use Extended Linker（LX51）instead of BL51 可支持 REMOVEUNUSED 参数



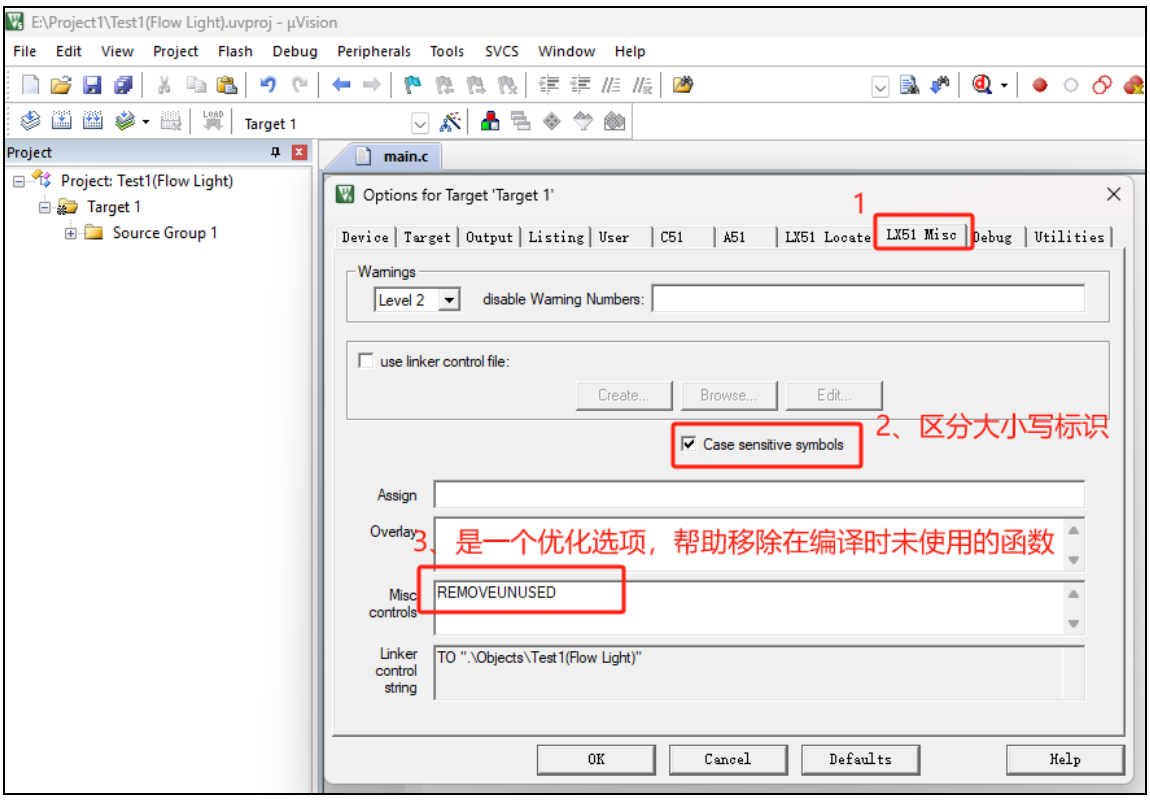
2）Output 选项卡--勾选 Create HEX File，选择 HEX-80

切换到 Output 选项卡，勾选‘Creat HEX File’选项，否则编译后是不会创建 HEX 文件的。HEX 文件是可以通过烧录工具烧录到单片机用户程序区。



3）LX51 Misc 选项卡--在 Misc controls 输入框输入‘REMOVEUNUSED’

切换到 LX51 Misc 选项卡，在 Misc controls 输入框中，输入大写的‘REMOVEUNUSED’，移除未使用的函数（如：库函数），以实现不使用的函数不进行链接。

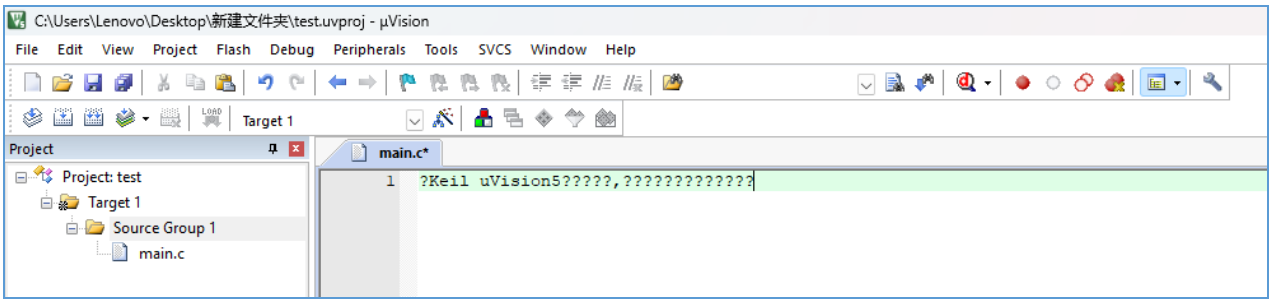


4）Debug 选项卡-- Ai8051U-8bit 工作在 8 位模式，暂不支持任何硬件仿真，无需设置

至此，工程项目的各种基础选项设置结束。可以进入下一节，在程序文件中添加实际的代码。

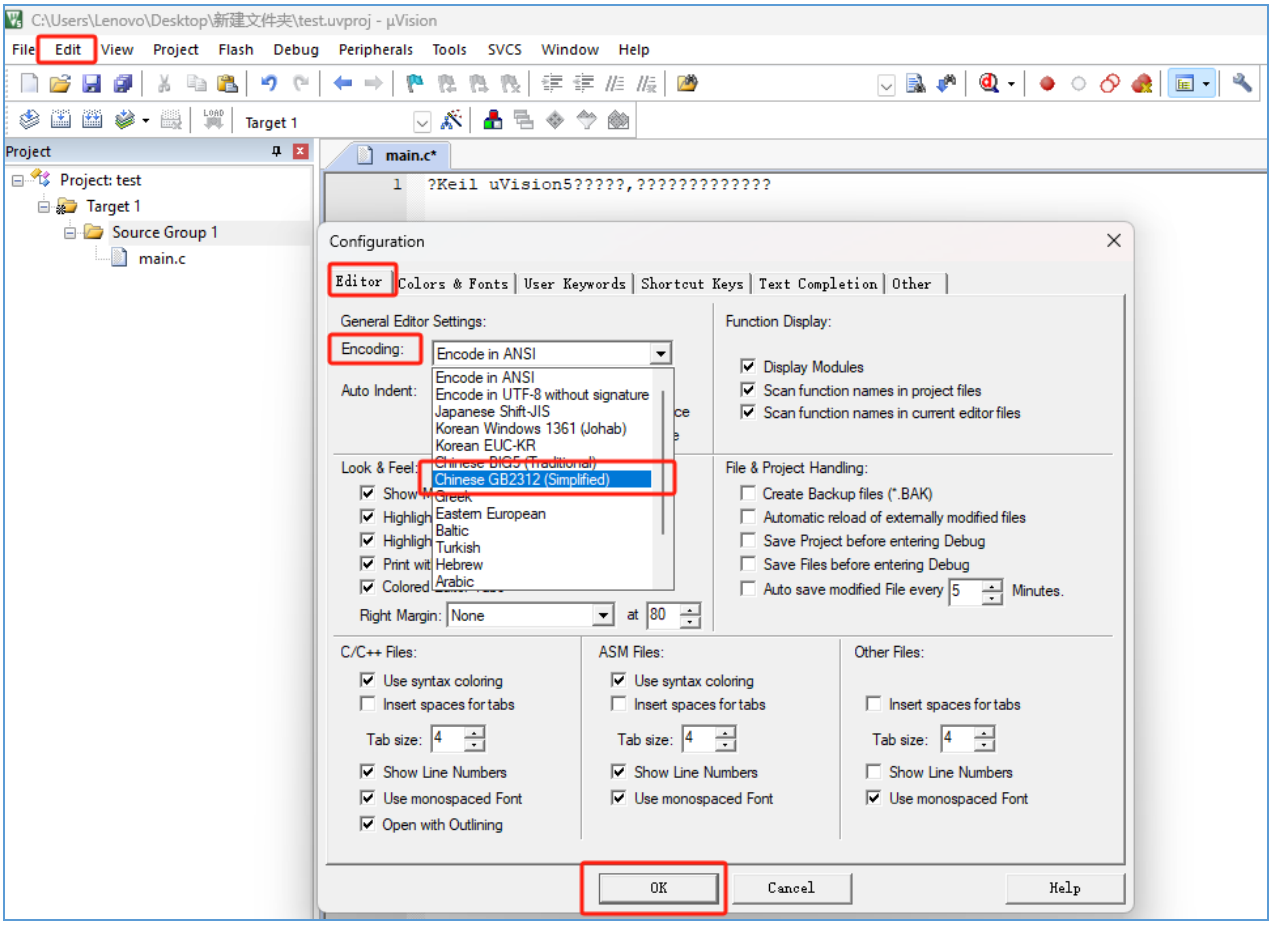
2.9　在 Keil uVision5 的编辑器中，输入中文出现乱码的解决办法

在使用 Keil uVision5 的编辑器中，有输入中文出现乱码的情况，如下图：



这是由于 Keil uVision5 编译器的字符编码设置问题导致的。Keil uVision5 编译器默认情况下使用的是 ANSI 编码，对中文 GB2312 编码不支持，不兼容。按如下步骤更改设置即可恢复正常：

在菜单栏找到并点击 Edit 菜单→选择 Configuration 菜单项→选择 Editor 选项卡→找到 Encoding 的下拉按钮→选择“Chinese GB2312(Simplified)”→点击“OK”，即可。



如果设置后编译器的代码中仍是乱码，则需重新复制粘贴即可正确显示。

2.10 Keil 软件中因含 0xFD 编码的汉字而出现乱码的问题

众所周知，Keil 软件的 8051 和 80251 编译器的所有版本都有一个叫做 0xFD 的问题，主要表现在字符串中不能含有带 0xFD 编码的汉字，否则 Keil 软件在编译时会跳过 0xFD 而出现乱码。

关于这个问题，Keil 官方的回应是：0xfd、0xfe、0xff 这 3 个字符编码被 Keil 编译器内部使用，所以代码中若包含有 0xfd 的字符串时，0xfd 会被编译器自动跳过。

Keil 官方提供的解决方法：在带有 0xfd 编码的汉字后增加一个 0xfd 即可。例如：

```
printf("数学");                    //Keil 编译后打印会显示乱码
printf("数\xfd 学");               //显示正常
```

这里的“\xfd”是标准 C 代码中的转义字符，“\x”表示其后的 1~2 个字符为 16 进制数。“\xfd”表示将 16 进制数 0xfd 插入到字符串中。

由于“数”的汉字编码是 0xCAFD，Keil 在编译时会将 FD 跳过，而只将 CA 编译到目标文件中，后面通过转义字符手动再补一个 0xfd 到目标文件中，就形成完整的 0xCAFD，从而可正常显示。

关于 0xFD 的补丁网上有很多，基本只对旧版本的 Keil 软件有效。打补丁的方法均是在可执行文件中查找关键代码[80 FB FD]，并修改为[80 FB FF]，这种修改方法查找的关键代码过于简单，很容易修改到其它无关的地方，导致编译出来的目标文件运行时出现莫名其妙的问题。所以，代码中的字符串有包含如下的汉字时，建议使用 Keil 官方提供的解决方法进行解决

GB2312 中，包含 0xfd 编码的汉字如下：

褛饼昌除待谍洱俘庚过糊积箭烬君魁
例笼慢谬凝琵讫驱三升数她听妄锡淆
旋妖引育札正铸　佚冽邶埤荤蔟摭啐
帻猃恺泯潺妪纨琮絮辇掌臊忑睚铨稞
瘕颀螨籪酏觚鳊鼾

另外，Keil 项目路径名的字符中也不能含有带 0xFD 编码的汉字，否则 Keil 软件会无法正确编译此项目。

2.11　C 语言中 printf()函数打印输出数据时，常用的各种输出格式

单片机 C 语言中的 printf_usb()函数和标准 C 语言中 printf()函数格式一致
在%后面根据数据格式需要加关键字：8 位"b", (默认)16 位"h"或不加, 32 位"l"

例如打印有符号十进制数据：
打印 8 位数据：%bd
打印 16 位数据：%hd 或者 %d
打印 32 位数据：%ld

```
printf("cnt8=%bu, ",cnt8++);       //C51 编译器 printf 输出 8 位数据需要使用"%b"
printf("cnt16=%hu, ",cnt16++);     //C51 编译器 printf 输出 16 位数据需要使用"%h"或不加参数
printf("cnt32=%lu\r\n",cnt32++); //C51 编译器 printf 输出 32 位数据需要使用"%l"
```

keil C251 编译器使用 printf 打印数据时，8 位/16 位不用加参数, 32 位加"l"

```
printf("cnt8=%u, ",cnt8++);              //C251 编译器 printf 可直接输出 8 位/16 位数据
printf("cnt16=%u, ",cnt16++);            //C251 编译器 printf 可直接输出 8 位/16 位数据
printf("cnt32=%lu\r\n",cnt32++);       //C251 编译器 printf 输出 32 位数据需要使用"%l"
```

支持数据类型：

Type	Argument Type	Input Format
d	int	Signed decimal number.
u	unsigned int	Unsigned decimal number.
o	unsigned int	Unsigned octal number.
x	unsigned int	Unsigned hexadecimal number using "0123456789abcedf".
X	unsigned int	Unsigned hexadecimal number using "0123456789ABCDEF".
f	float	Floating-point number formatted as <[>-<]>*dddd.dddd*.
e	float	Floating-point number formatted as <[>-<]>*d.ddddd*e<[>-<]>*dd*.
E	float	Floating-point number formatted as <[>-<]>*d.ddddd*E<[>-<]>*dd*.
g	float	Floating-point number using either the e or f format, whichever is more compact for the specified value and precision.
G	float	Floating-point number using either the E or f format, whichever is more compact for the specified value and precision.
c	char	A single character.
s	*	A string of characters terminated by a null character ('\0').
p	*	A generic pointer formatted as *t:aaaa* where *t* is the memory type and *aaaa* is the hexadecimal address.

2.12　实验一：printf_usb("Hello World !\r\n")，第一个完整的 C 语言程序

看下面的程序，其实还是那个熟悉的标准 C 语言的 printf()，只是到了单片机 C 语言中，变成了 printf_usb()

2.12.1　printf( )，printf_usb( )实验程序代码

```
#include   "ai8051u.h"   //头文件不分大小写，但关键字分大小写
#include   "ai_usb.h"    //USB 库和 USB 虚拟设备的头文件，需要在项目文件中添加对应的 USB 库
//由于在 USB 库函数的代码中已经定义了，不停电 ISP 下载命令字"@STCISP#"，
//只需要在下载软件的“收到用户命令后复位到 ISP 监控程序区”选项卡中进行相应的设置，
//即可实现 USB 不停电 ISP 下载功能，或串口不停电 ISP 下载功能。

void   main(void)
{
    EAXFR = 1;   //允许访问扩展的特殊寄存器，XFR
    WTST = 0;    //设置取程序代码等待时间，赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;  //设置访问片内的 xdata 速度，赋值为 0 表示用最快速度访问，不增加额外的等待时间

    usb_init();
    //初始化 USB 模块，并且会将【P3.0, P3.1】初始化为高阻输入，不影响 USB【D-, D+】通讯
    EA = 1;         //总中断允许位打开

    while (1)
    {
        printf_usb("Hello World !\r\n");
        //直接打印输出 “Hello World !” 到主机相应软件的 USB【CDC/HID-串口助手】
        //   \r 是回车，跳到行首，\n 是换行，就是换到下一行；\r\n 不要交换次序
    }
}
```

以上是完整的第一个程序代码

printf_usb()函数：

printf_usb()函数和标准 C 语言的格式完全一致，详见 printf()函数章节
进入 printf_usb()函数后，首先会判断 USB 主机是否已经对 USB 设备配置完成，
如果没有配置完成，printf_usb()函数会直接返回退出。
如果已经配置完成，还会继续判断如有发送前一笔数据，则判断是否发送完成，
如果没有发送完成，会继续等待，直到前一笔数据发送完成，
才会发送新的 printf_usb()需发送的数据后再返回退出。
但如等待超过 50ms@24MHz，会直接退出。
如果主时钟不是 24MHz，20MHz 等待时间会变长，40MHz 等待时间会变短。
如果已经配置完成且前一笔数据已经发送完成，
则会直接发送新的 printf_usb()需发送的数据并返回退出。

如何获得上面"ai8051u.h"和"ai_usb.h"这 2 个头文件

如已按照前面介绍的方法：
从 www.STCAI.com 网站，【软件工具 | 工具软件】处，下载安装最新的【AIapp-ISP 软件】，
选择其中的选项卡 【Keil 仿真设置 】，

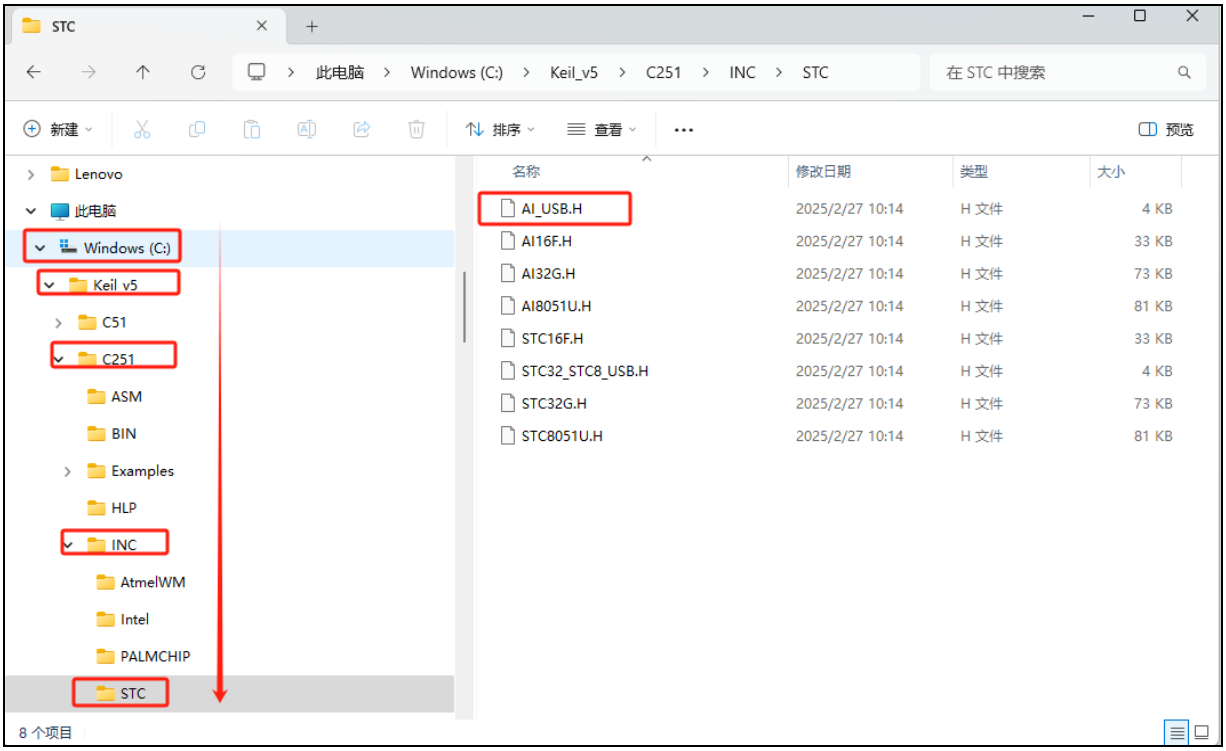
点击【添加型号和头文件到 Keil 中，并添加仿真器驱动到 Keil 中】按钮，
则 Ai8051U.h 和 AI_usb.h 都会自动复制到 Keil 中，包含进来就可以直接使用，
大家可以打开看下其中的内容，AI_usb.h 和 stc32_stc8_usb.h 是相似的，只需用其中的 1 个头文件，
具体见 Keil_v5\C251\INC\STC 目录，或 Keil_v5\C51\INC\STC 目录。
如将对应的 USB 库文件添加到 项目中，后续用户程序中，USB 如收到命令字符串 "@STCISP#"
USB 库函数就会自动软复位到系统 ISP 程序区，实现 USB 不停电下载，具体见相应部分的说明。

2.12.2　准备工作，如已准备，跳过此节

1、添加常用头文件 ai_usb.h

```
#include "ai_usb.h"                    //使用 usb 所需要包含的头文件。
```

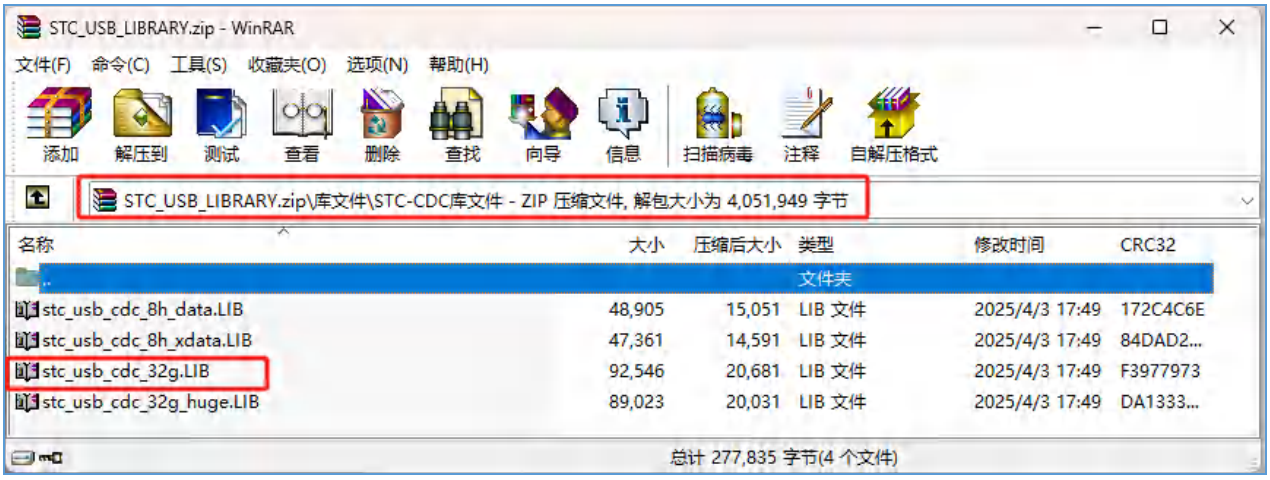
这个头文件在使用 AIapp-ISP 软件添加型号和头文件到 Keil 中时就已经一并被添加到 Keil 中了。



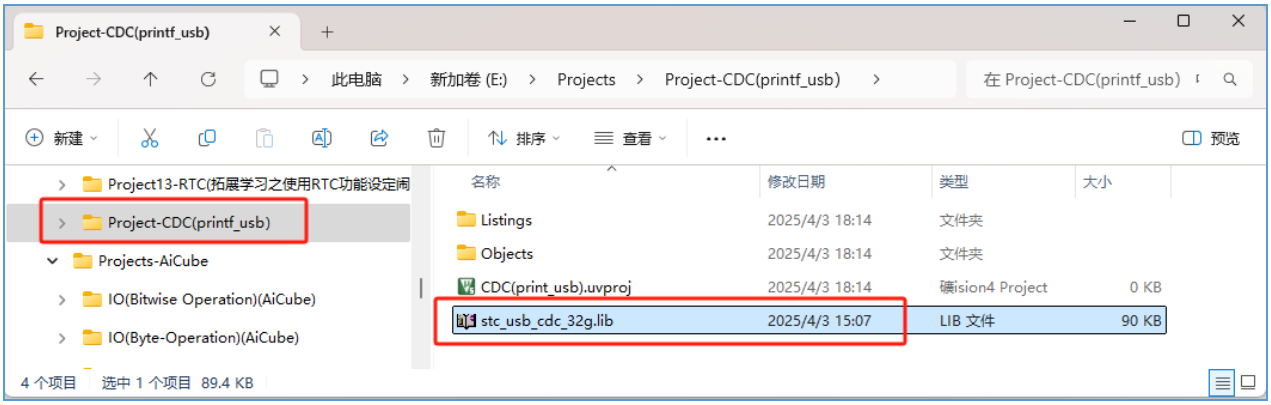
2、添加 usb_cdc 库函数文件

因为要使用到 USB 功能，必须在项目中添加 USB 库函数。这部分可以到 STCAI 的官网：
https://www.stcai.com/khs 的 USB 库文件中进行下载

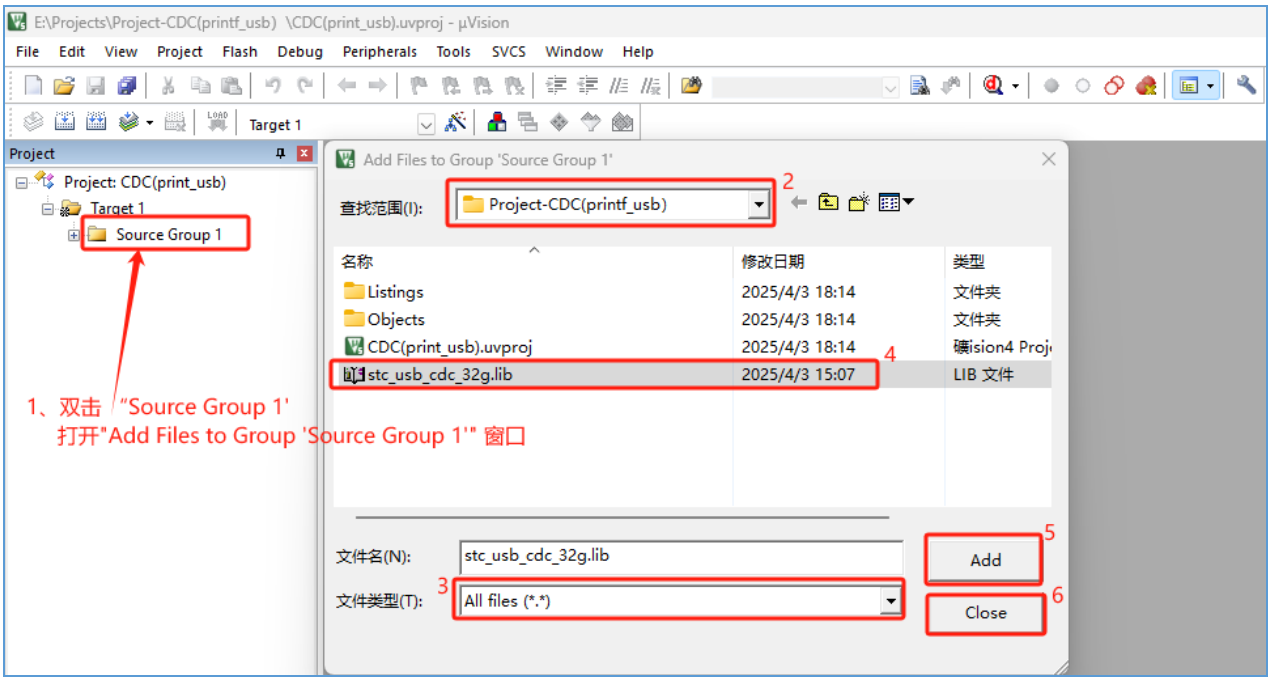
下载完成后，打开压缩包“STC_USB_LIBRARY.zip”，
在’库文件\查询模式 STC-CDC 库文件’中，复制出来 stc_usb_cdc_32g.LIB



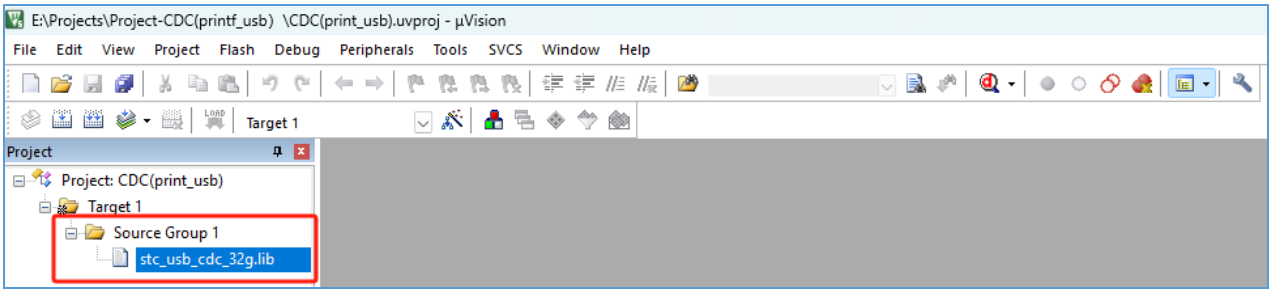
将库函数“stc_usb_cdc_32g.LIB”复制到创建工程所在的文件夹 Project-CDC(print_usb)中：



将库函数 stc_usb_cdc_32g.LIB 添加进入此工程项目中
在 Keil 中，双击 Source Group，在弹出的 Add Files to Group 'Source Group 1 窗口中，将库函数 stc_usb_cdc_32g.LIB 添加进入此工程，就可以使用 USB-CDC 功能。



这样，Keil 界面变化如下图：



【注意】：

USB-HID 通讯不需要安装驱动，手机及 Windows 各种系统都自带 HID 驱动。

USB-CDC 通讯，手机不需要安装驱动，手机的各种系统已自带 USB-CDC 驱动。

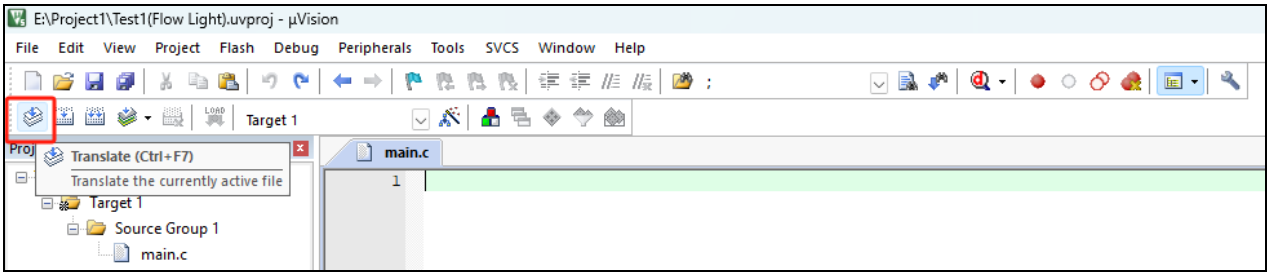
USB-CDC 通讯，Win10（1903）及以后版本不需要安装 USB-CDC 驱动，较早的 Windows 版本才需要手动点击 ISP 软件中的【USB 驱动安装】，只有较早的 Windows 系统才需要安装 USB-CDC 驱动程序。

USB-CDC 虚拟串口类似传统串口的形式，但不需要进行波特率的设置，通讯时是以 12Mbps 的速度进行通讯，USB-CDC 虚拟串口转串口才需要设置转传统硬件串口需要的波特率、奇偶校验位等参数。

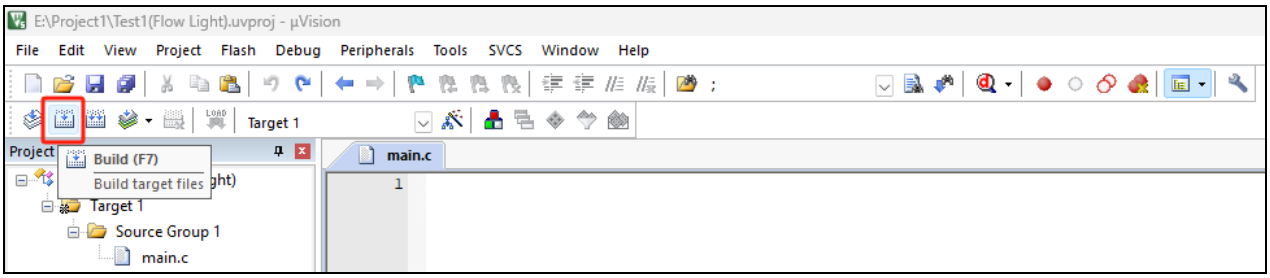
2.12.3　认识 Keil 软件的 Build Toolbar（Build 工具栏）

Keil 是一款软件开发系统，输入工程项目的程序源代码后，要进行编译（Build）。编译过程是将 C 语言源代码转换为机器可以执行的机器代码（通常是 HEX 文件），这样程序才能在目标硬件上运行。编译过程中，编译器对源代码进行预处理，处理宏定义、条件编译指令等。然后将预处理后的源代码转换成汇编语言。这一步会检查语法错误，并将高级语言转换为低级语言。下面我们认识一下 Build 工具栏的几个重要的功能按钮。

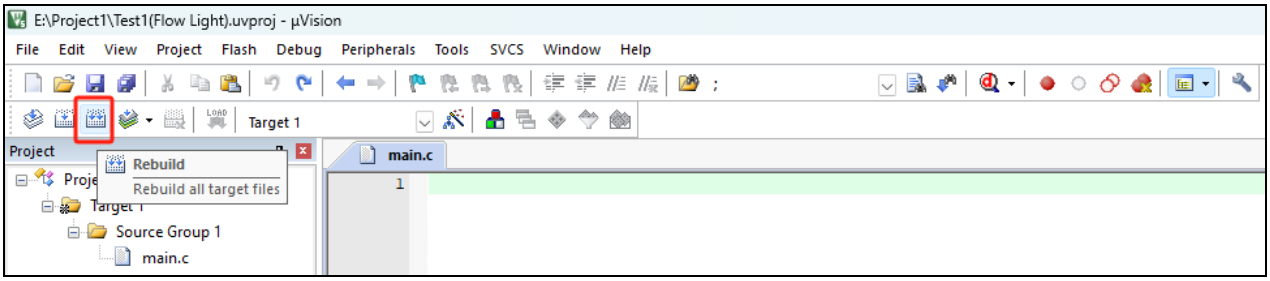
Translate（Translate the currently active file）按钮：编译当前正在编辑的源文件(main.c 仅生成 main.o)，在 Translate 过程中会检查源文件中的语法错误，但并不生成最终可执行文件。



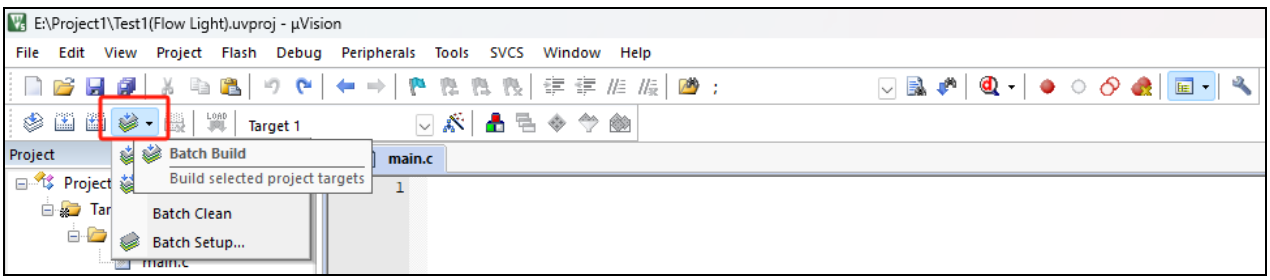
Build（Build target files）按钮：Build 在技术文档中是 increase build，所以理解 Build 按钮是编译工程中当前修改的文件及与此文件相关联的所有文件（模块），检查语法错误，并重新链接生成可执行文件。这样特别在调试大型项目时，因为不需要每次都从头开始编译整个项目，可以节省时间。平时项目开发调试验证过程中，也更多选择用 Build 编译生成可执行文件验证。如果工程是首次编译，会直接调用 Rebuild All。



Rebuild（Rebuild all target files）按钮：编译全部文件，检查语法错误，并重新链接生成可执行文件。



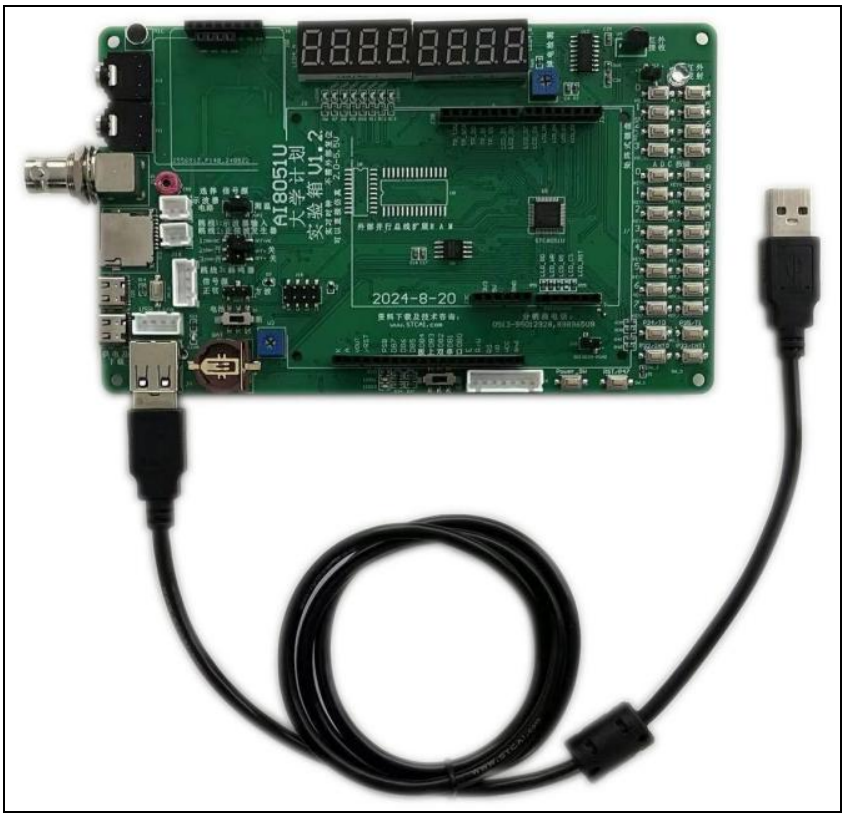
Batch Build（Build selected project targets）按钮：一次编译多个项目。在 Keil 中，可创建多个 projects，并通过 Batch Build 同时对这些 projects 进行编译。



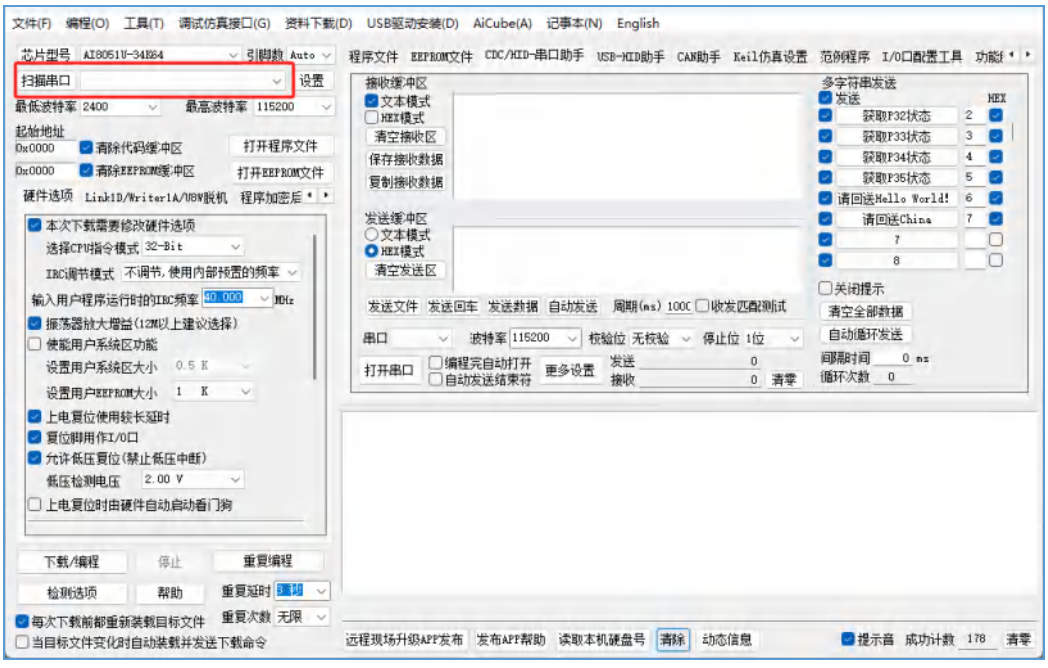
2.12.4　下载用户程序到实验箱

2.12.4.1　连接实验箱和电脑

拿出实验箱中配置的 USB-TypeA 到 USB-TypeA 线将实验箱与电脑正确连接。

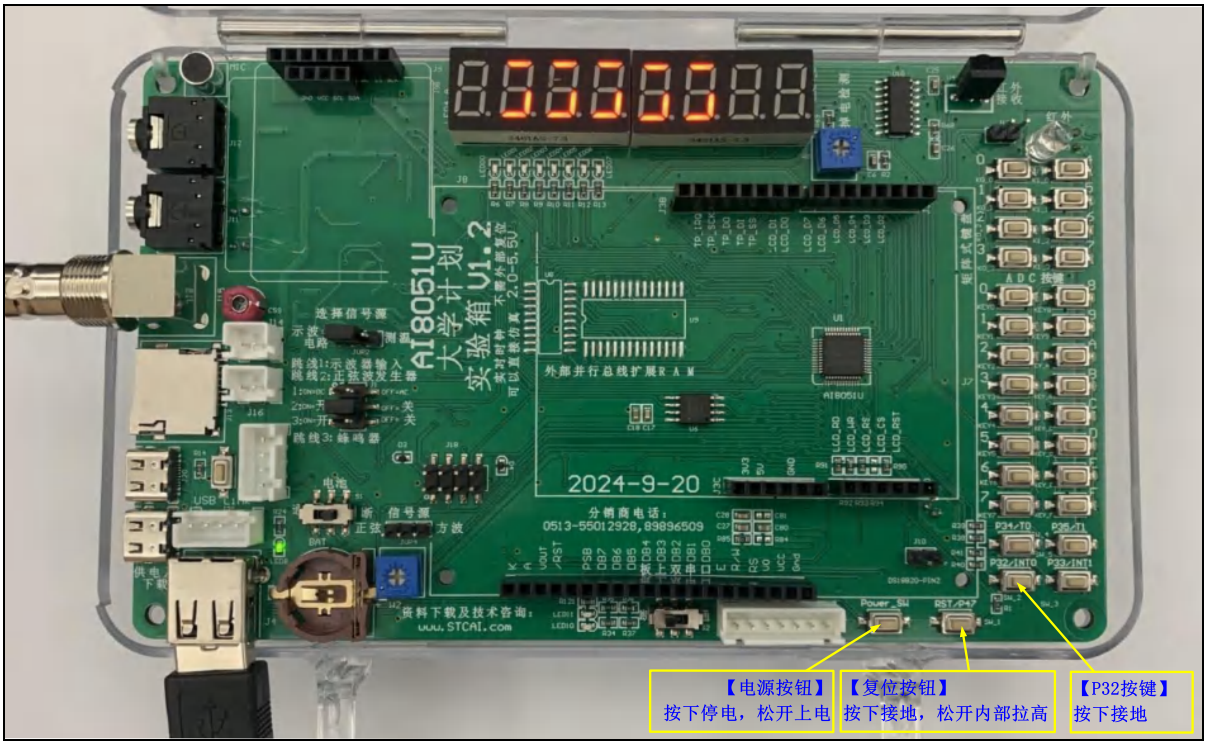


然后打开 AIapp-ISP-v6.95K 或以上版本。选择单片机型号：Ai8051U-34K64，如下图，我们看到“扫描串口”后面的端口列表框不是“（HID1）USB-Writer”，说明还不能下载程序到实验箱。

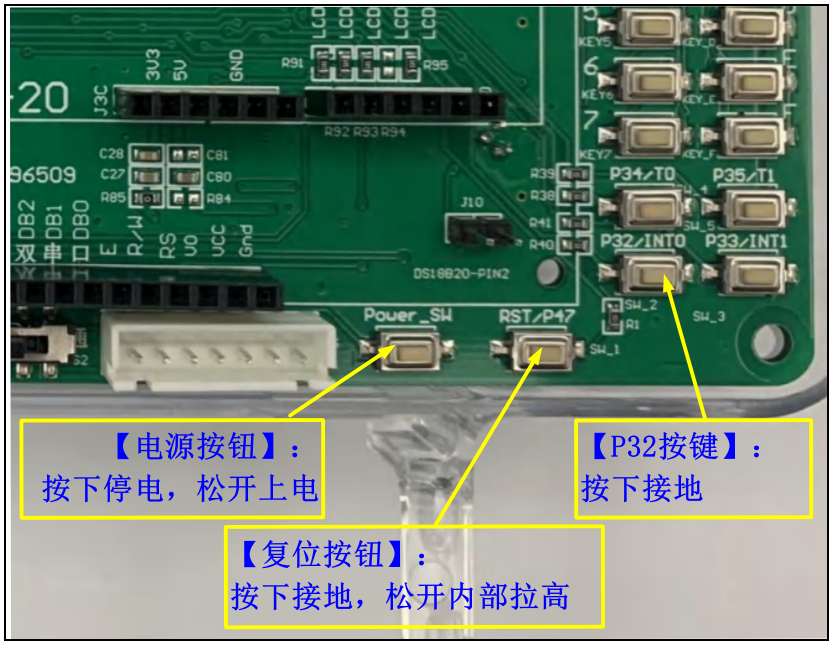


2.12.4.2 硬件 USB 下载用户程序步骤

下图是 Ai8051U 实验箱已经和电脑相连的正面图片，数码管是随机显示的，**此实验的目的把 Ai8051U 实验箱作为 USB-CDC 设备和电脑进行通讯，收发数据。**



局部放大实验箱图片，其正下方有两排按键，找到 P3.2 按键和 Power_SW 按键



【方法一，P3.2 按键，再结合停电上电下载】

1、按下板子上的 P3.2 按键，就是 P3.2 接地

2、给目标芯片重新上电，不管之前是否已通电。

===本 Ai8051U 实验箱是：按下电源按钮停电，再松开电源按钮上电，

这时电脑端 ISP 下载软件会自动识别出“(HID1) USB Writer”，此时就与 P3.2 状态无关了，可以松开 P3.2 按键（P3.2 在用户程序区可做其它任意用途）

3、点击电脑端下载软件中的【下载/编程】按钮（注意：USB 下载与串口下载的操作顺序不同）

电脑端 ISP 下载软件提示：下载进行中…，几秒钟后，提示下载成功！

【方法二：复位管脚低电平复位下载】

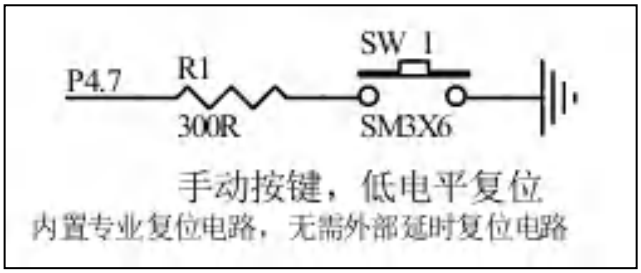
USB 连接好并已上电的情况下，外部按键复位也可进入 USB 下载模式，注意：

P4.7-nRST 出厂时默认是 P4.7-I/O 功能，要改为复位功能，需 ISP 烧录时取消设置复位脚用作 I/O 口，停电一次再上电才生效，程序区中用户程序也可改为复位脚或 I/O，这个立即生效。

1、按下复位管脚/P4.7-nRST 外接的低电平复位按键，复位 MCU，

再松开复位键，MCU 首先从系统程序区启动运行系统程序，判断是否要下载用户程序，

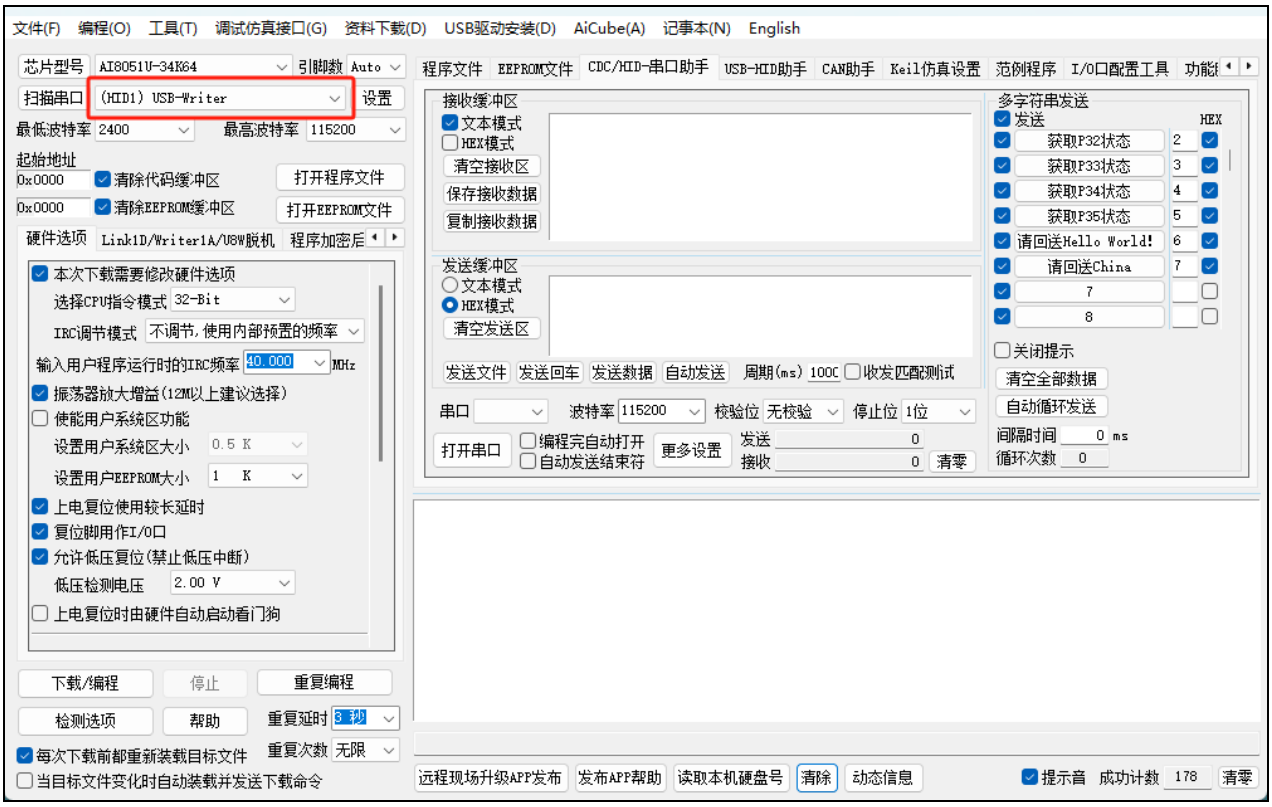
等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后



2、点击电脑端下载软件中的【下载/编程】按钮

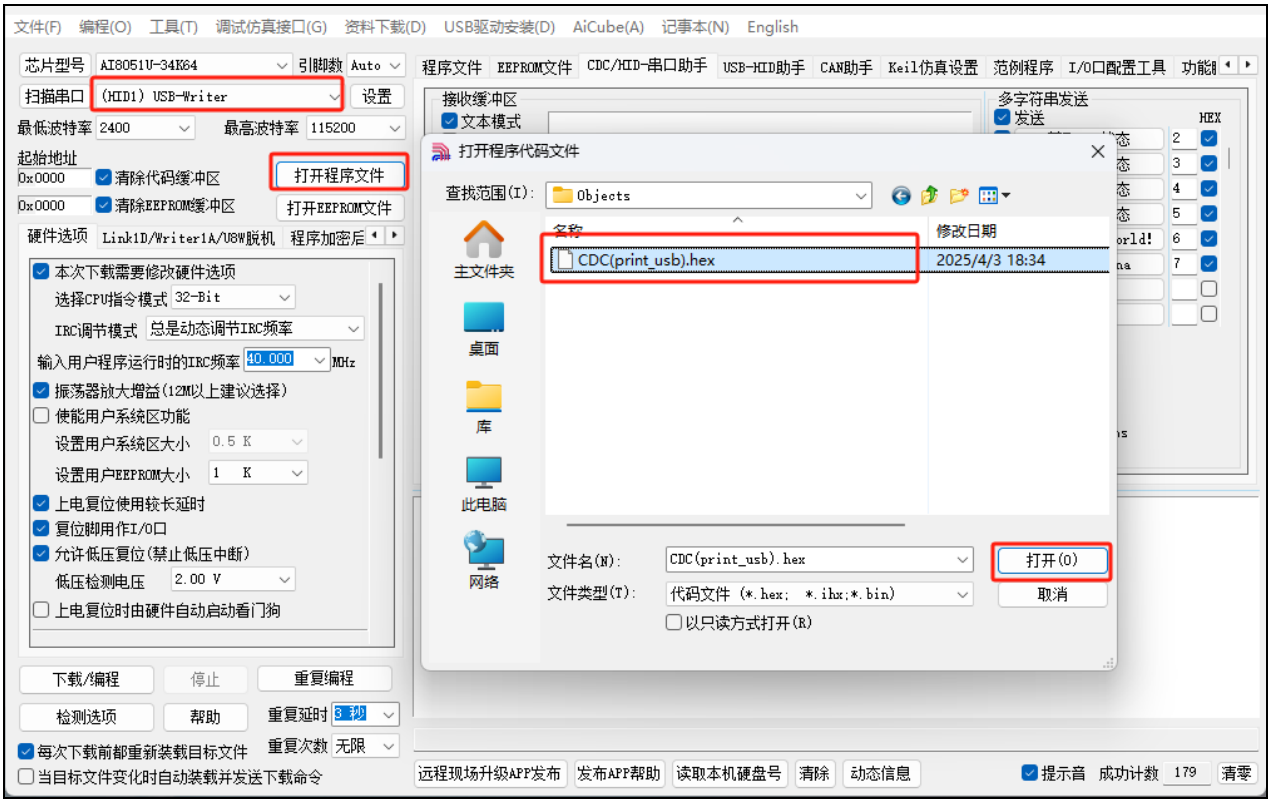
电脑端 ISP 下载软件提示：下载进行中…，下载成功！

备注：如果目标 MCU 的用户程序区已有 USB 通讯程序（含 CDC、HID 等），则通过复位脚复位进入 USB 下载模式不被支持，只能通过停电上电冷启动下载或者从用户程序区软复位到系统区下载

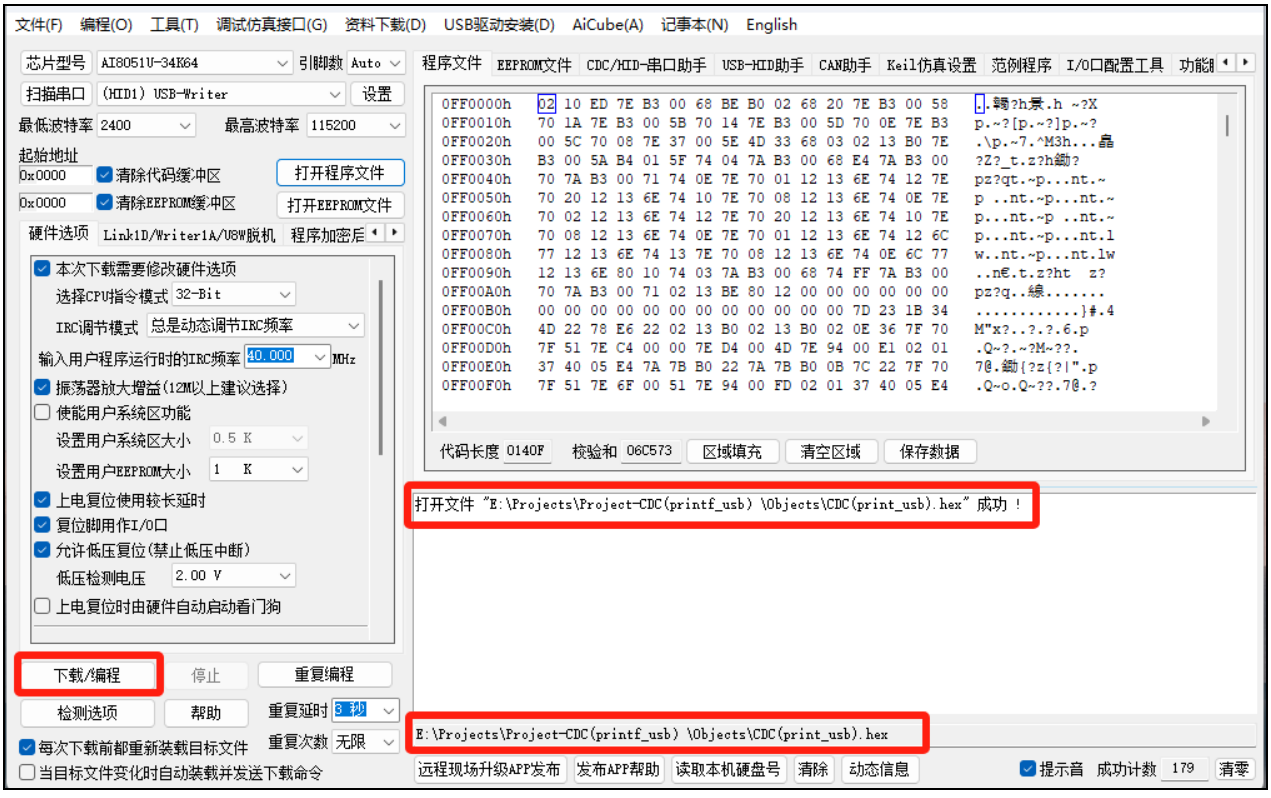


2.12.4.3 下载用户程序文件并查看实验现象

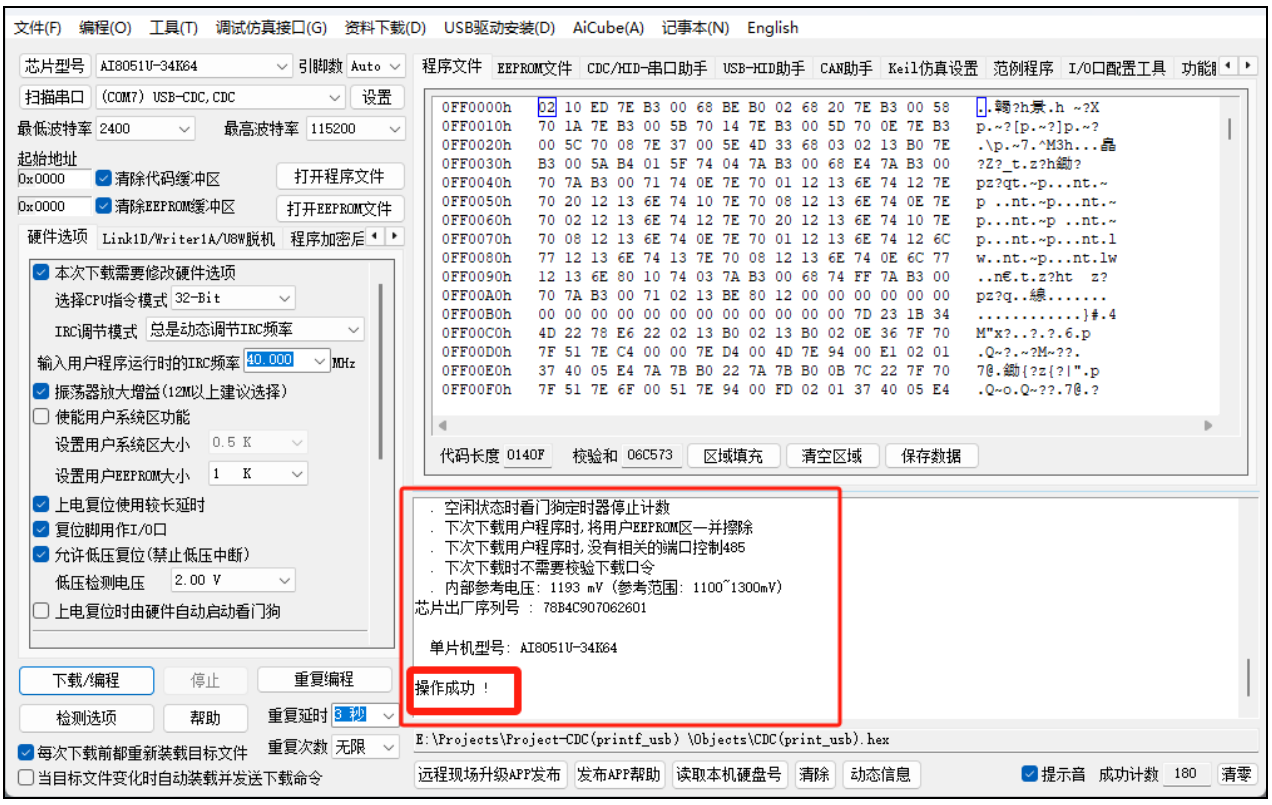
现在就打开用户程序文件，点击 AIapp-ISP 软件界面上的“打开程序文件”（打开的是项目程序文件编译无误后生成的 HEX 文件，一般默认存放在该项目的 Objects 目录中），调入在前面小节编译好的 HEX 文件：



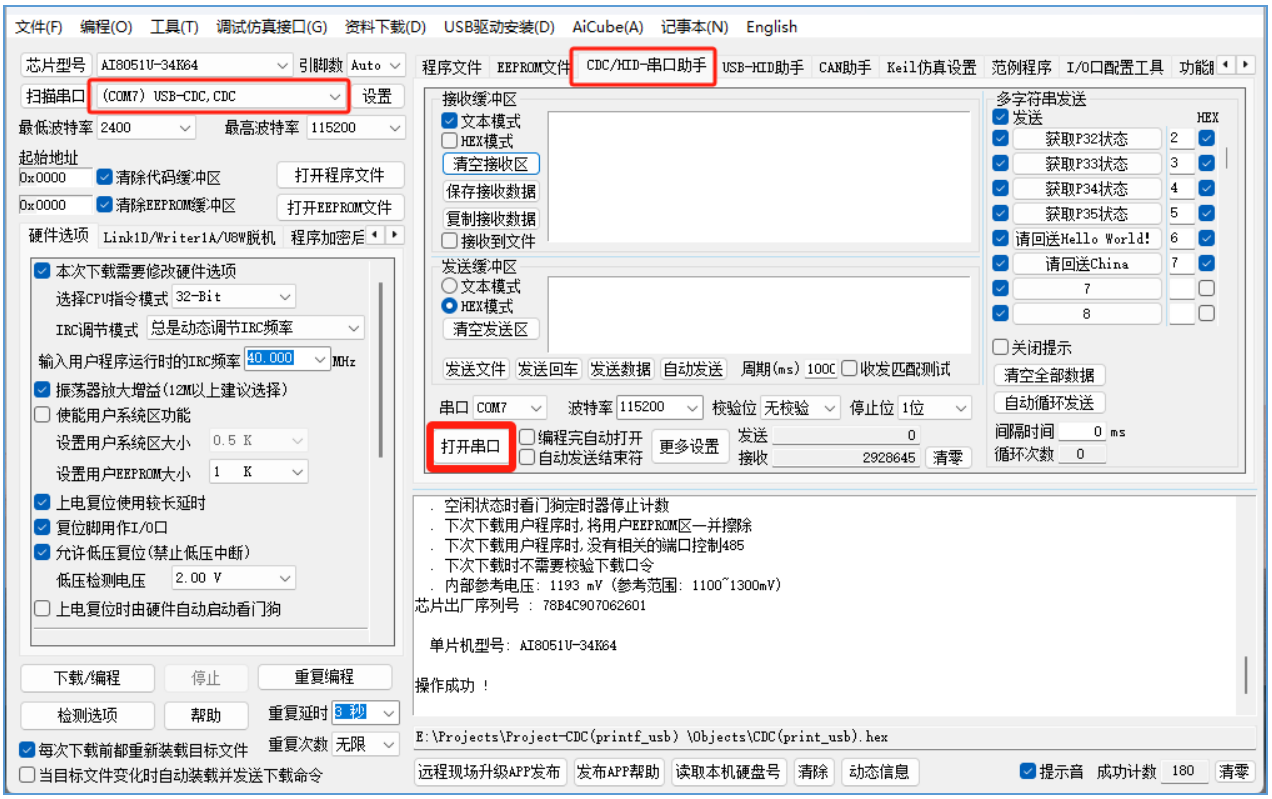
然后，AIapp-ISP 软件界面如下图所示，出现“‘打开文件***.hex’成功！”



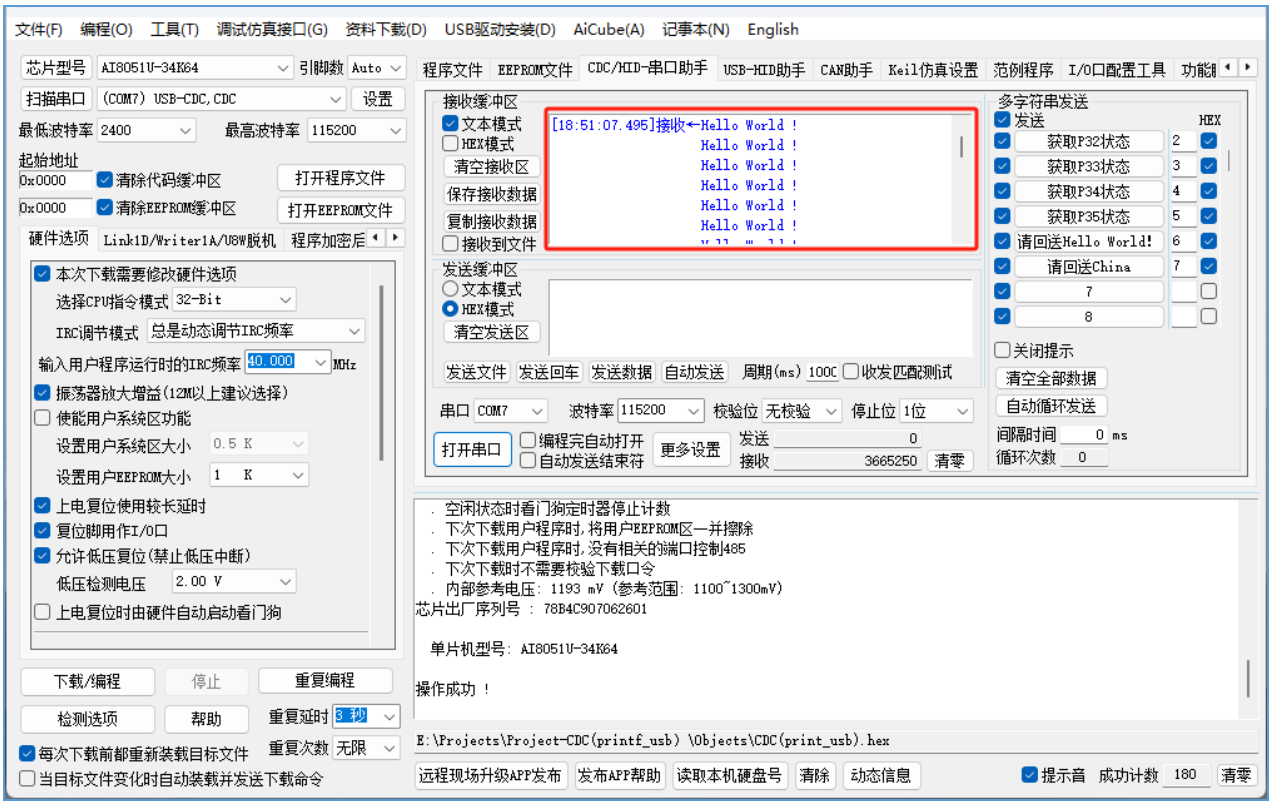
点击“下载/编程”，把 CDC(print_usb).hex 烧录到实验箱，AIapp-ISP 软件界面出现：操作成功！恭喜你！



我们看到此时串口显示：（COM7）USB-CDC，CDC，选择 AIapp-ISP 系统软件的“CDC/HID-串口助手”选项卡，找到“打开串口”按钮，如下图：



选择好对应的串口号，点击“打开串口”按钮，CDC/HID-串口助手的“接收缓冲区”，就会不停的显示“Hello World！”，如下图：



至此，printf_usb()显示“Hello World！”实验成功！

2.12.5　使用【AiCube】工具创建 -- printf_usb("Hello World !\r\n")，第一个完整的 C 语言程序

STCAI 在最新版的 AIapp-ISP 软件中，推出【AiCube】项目创建工具，目前支持 Ai8051U-32Bit 各功能模块的智能化配置，并生成程序框架。这项功能在程序员建立项目时，简化了项目的初始化难度，提高效率，避免因不太熟悉或不完全了解该单片机的性能而忽略个别配置，引起编译错误等问题。尤其对刚涉足单片机的初学者，此工具不仅引导其了解单片机编程流程，还极大的降低了入门门槛，缩短了摸索的时间，提高入门的兴趣和信心。现在我们使用这个工具来生成实现此实验功能的配置和代码。

a) 安装 AIapp-ISP 下载/编程/烧录/软件

安装 STCAI 公司研发的 AIapp-ISP 下载 / 编程 / 烧录 / 软件，含强大的辅助开发工具包。登录 STCAI 官网，下载最新版的 AIapp-ISP 安装包，下载链接如下：https://www.stcai.com/gjrj

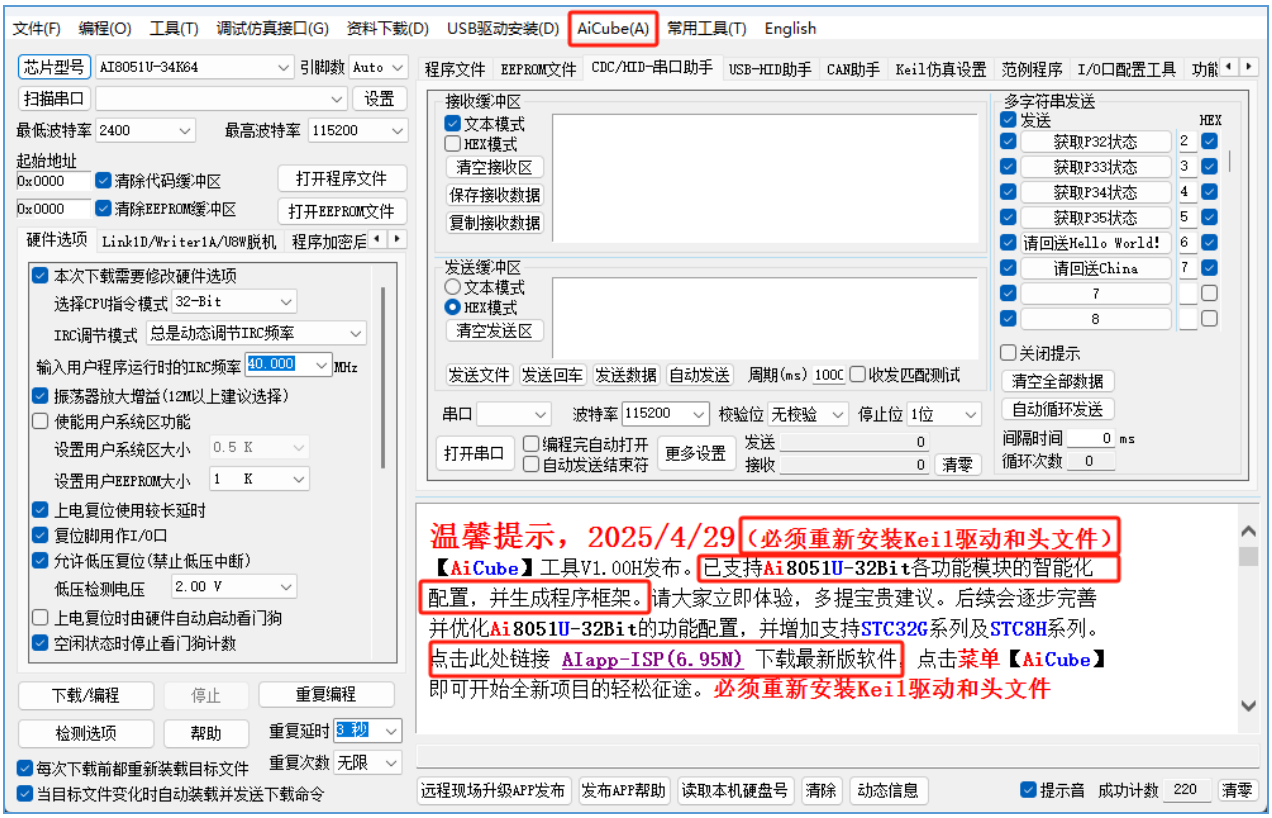
下载下来的是压缩包文件如下图：

双击后解压缩如下图：

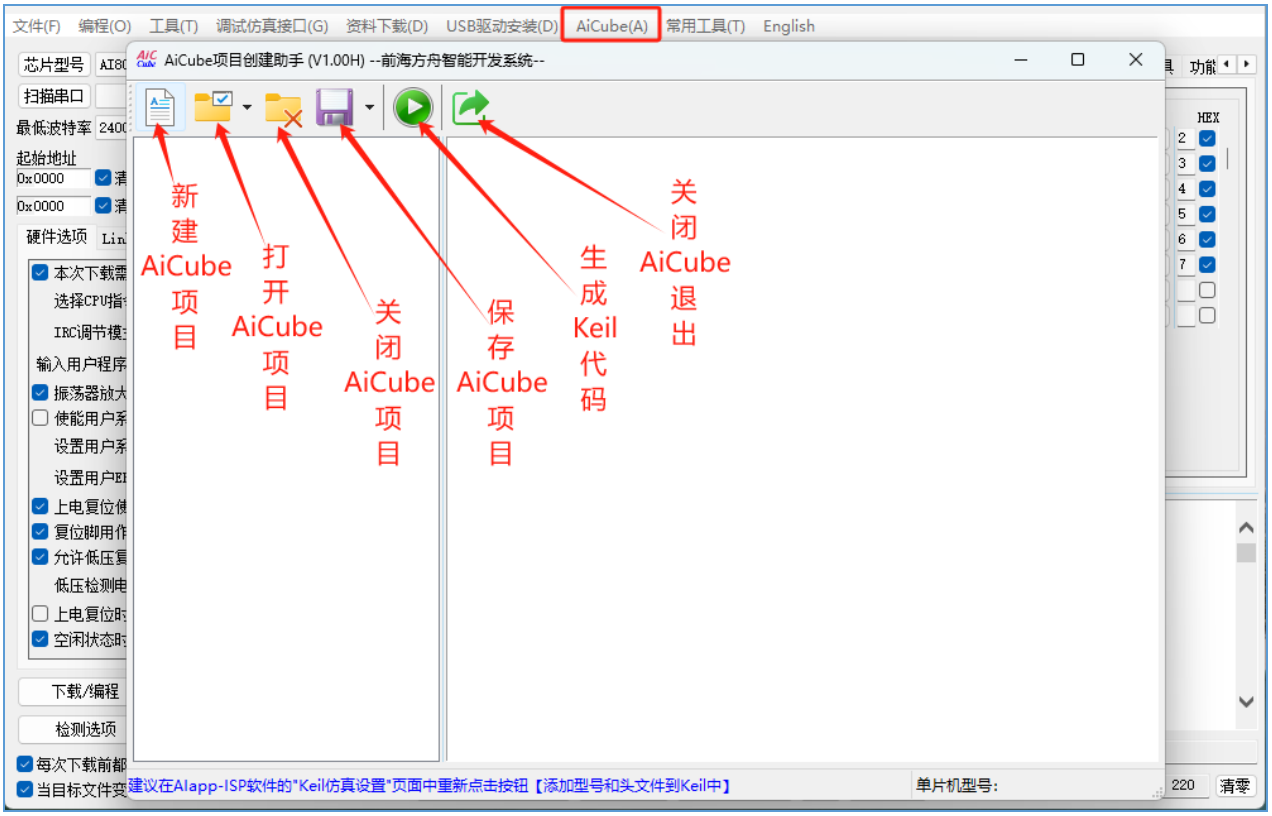
如果是解压缩到非桌面的其他目录，如 D:\STCAI-ISP，建议创建快捷方式到桌面方便打开。这实际是一个免安装的绿色软件。

b) 打开【AiCube】

双击图标，打开 AIapp-ISP 软件界面，即可直接使用，如下图，在菜单栏上找到【AiCube(A)】菜单：

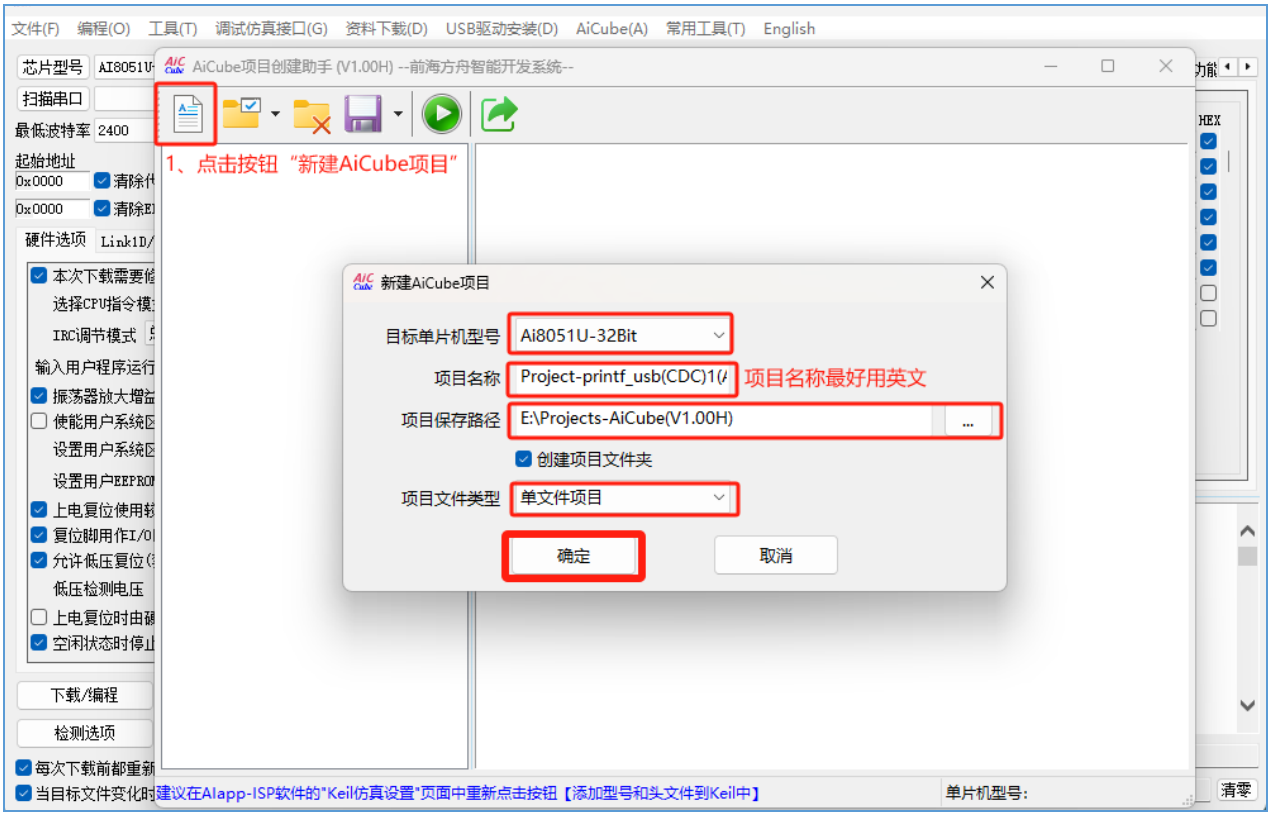


点击【AiCube(A)】，弹出如下窗口：

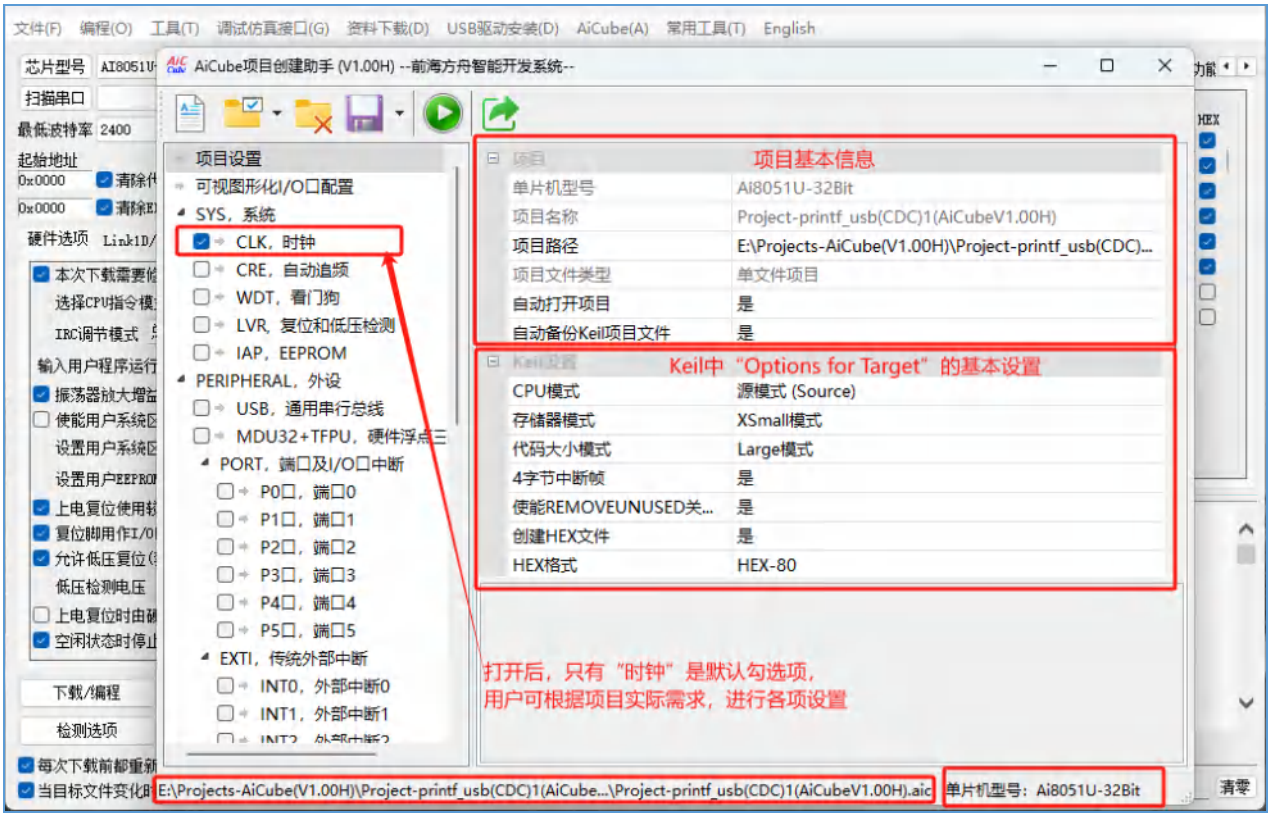


c) 使用【AiCube】工具创建项目

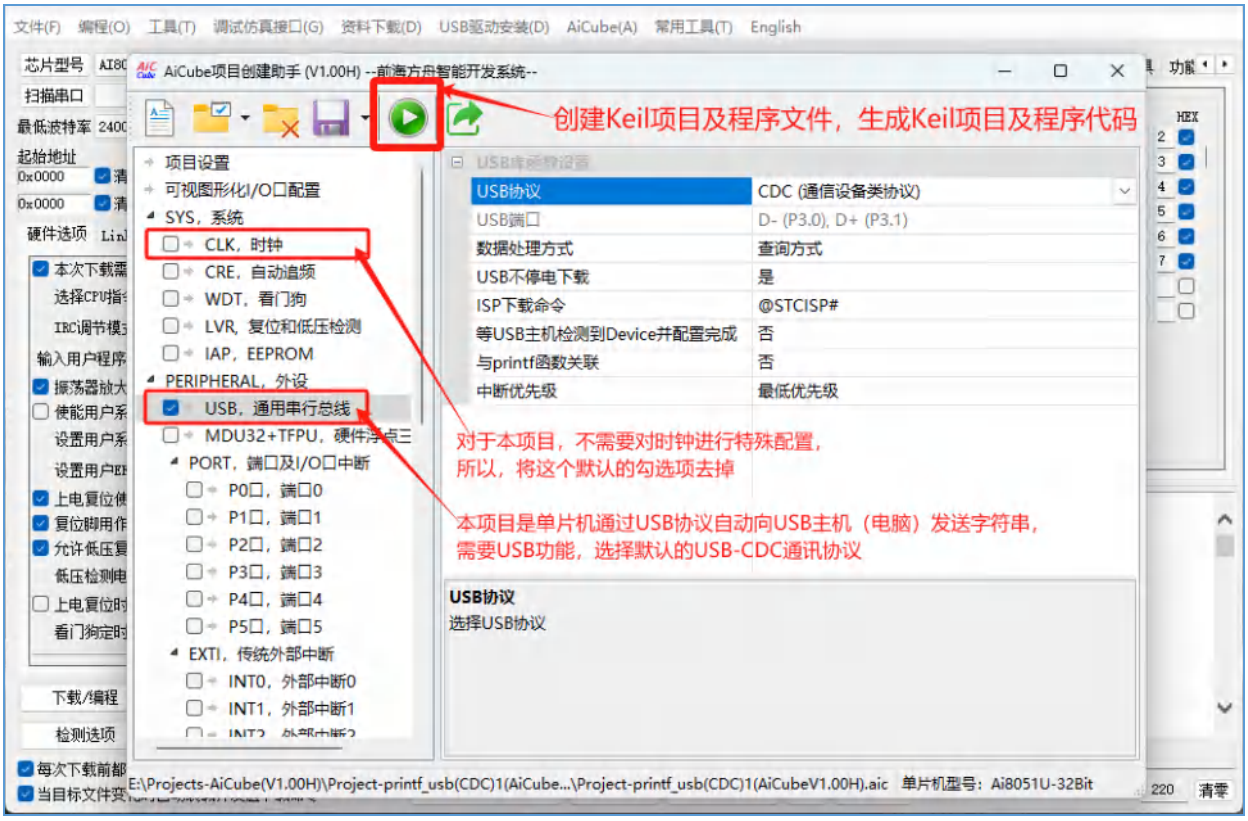
点击按钮“新建 AiCube 项目”，出现如下浮窗，选择并填写项目基本信息，建议项目名称用英文！



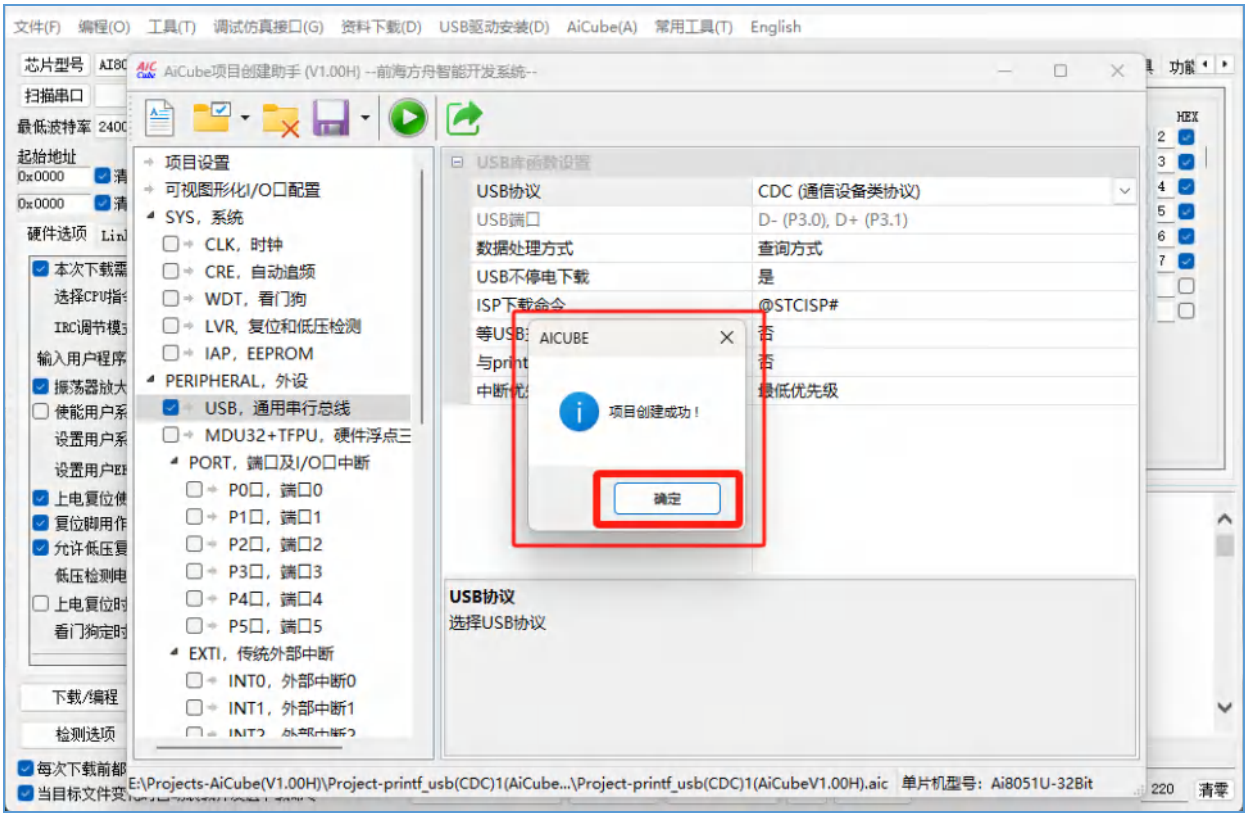
点击“确定”，出现如下界面：



因本项目用到 USB-CDC 功能，要加载 USB-CDC 库函数，USB 初始化等，使用【AiCube】工具，根据本项目的实际需求，对配置进行如下修改：



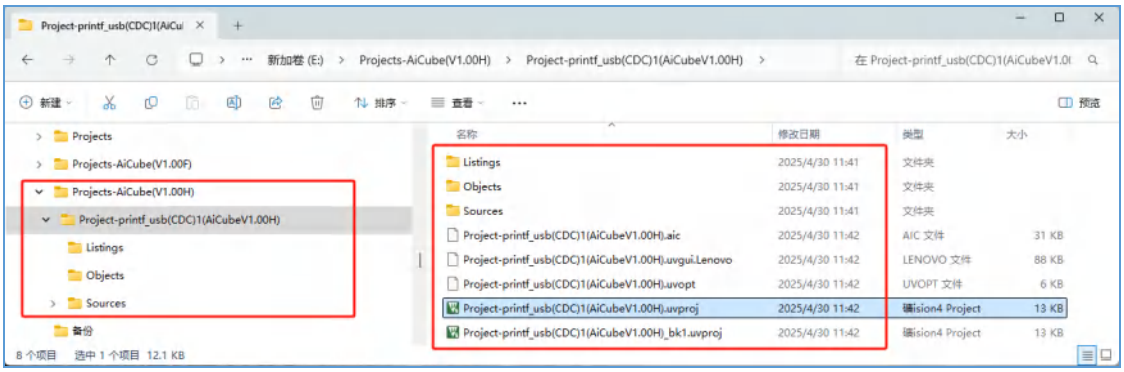
然后点击工具栏上的“创建 Keil 项目及程序文件”按钮，【AiCube】提示：“项目创建成功！”，如下图：



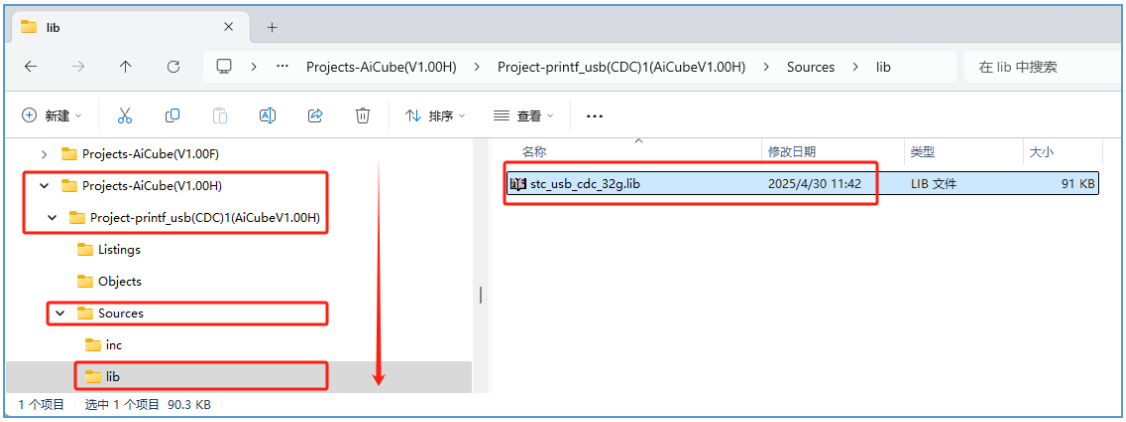
点击“确定”按钮后，系统会自动打开 Keil 软件，调入新创建的 Keil 工程项目，本实验创建的项目是“Project -printf_usb(CDC)1(AiCubeV1.00H)”。现在，可以关闭【AiCube】工具，当然，也可以不关闭。

➲ 找到并打开刚才创建的项目目录

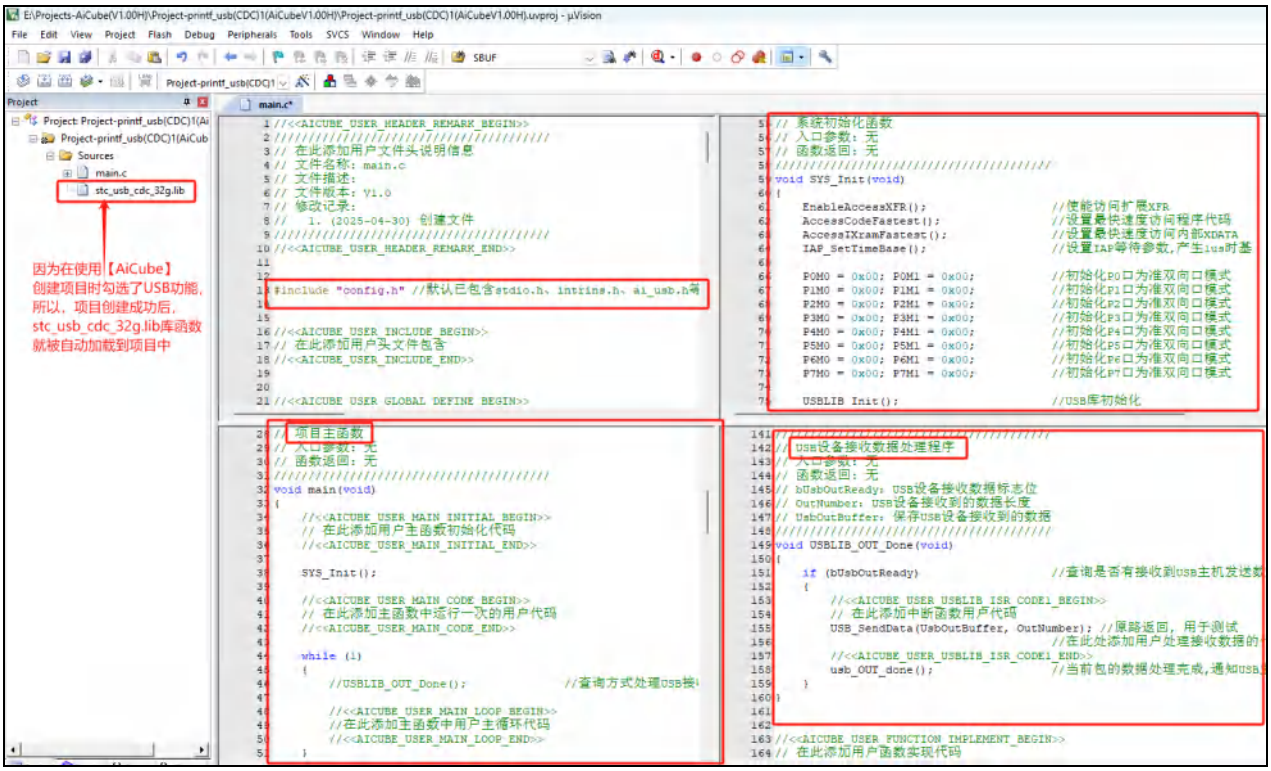
上述操作过程，【AiCube】工具就已经帮我们建立了项目目录及 Project-printf_usb(CDC)1(AiCube V1.00H).uvproj 文件，如下图：



再打开 Sources>lib 子目录，发现，【AiCube】助手将相应的 USB-CDC 库函数加载进项目目录：



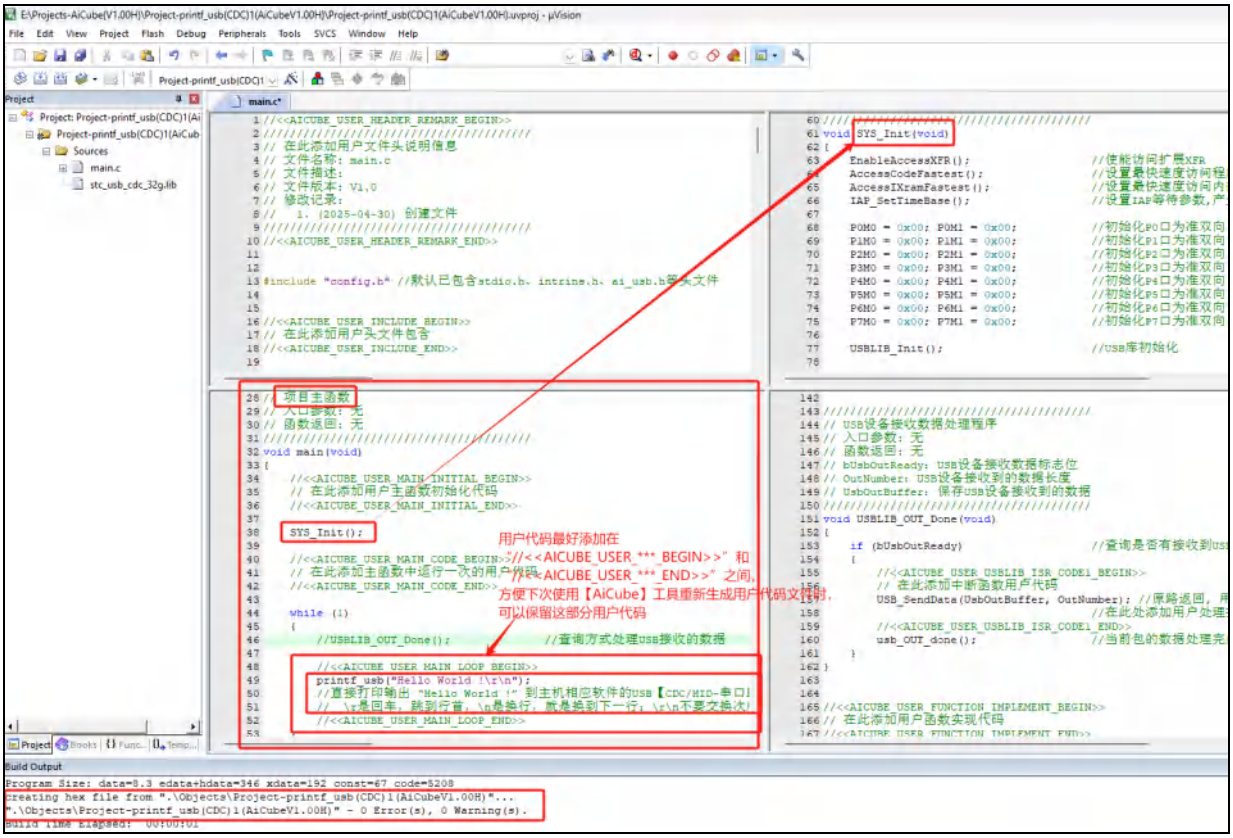
➲ 观察已打开的 Keil 项目文件：Project-printf_usb(CDC)1(AiCubeV1.00H).uvproj，如下图：



根据在【AiCube】工具中的配置，从上图中我们看到【AiCube】创建的本实验项目中不仅有了 main.c，还已经自动加载了 USB-CDC 库函数（stc_usb_cdc_32g.lib）。main.c 中按模块化设计，已经包含了：必要的头文件、主函数、系统初始化函数、微秒延时函数、毫秒延时函数、USB 相关的配置及处理函数。

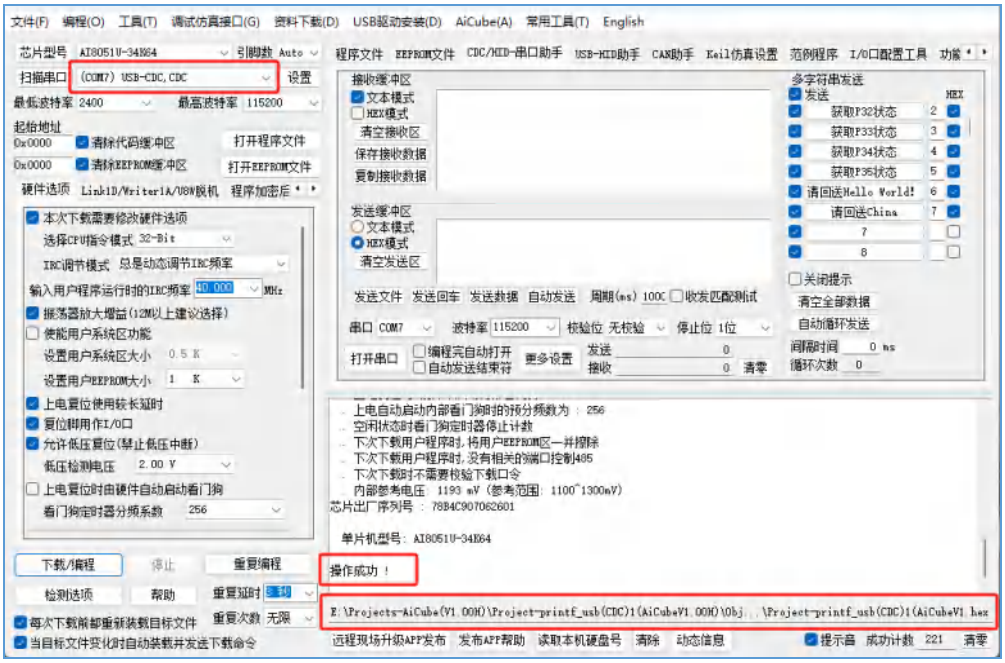
d) 增加实验项目代码

为了实现单片机通过 USB 协议向 USB 主机（电脑）发送字符串"Hello World ! "，我们现在只需要在主函数的 while 循环中加入“printf_usb("Hello World !\r\n");”代码即可，保存，编译，如下图（为方便查看，使用菜单“Window”→“Split”将原窗口分裂出 4 个窗口，对比观察）：

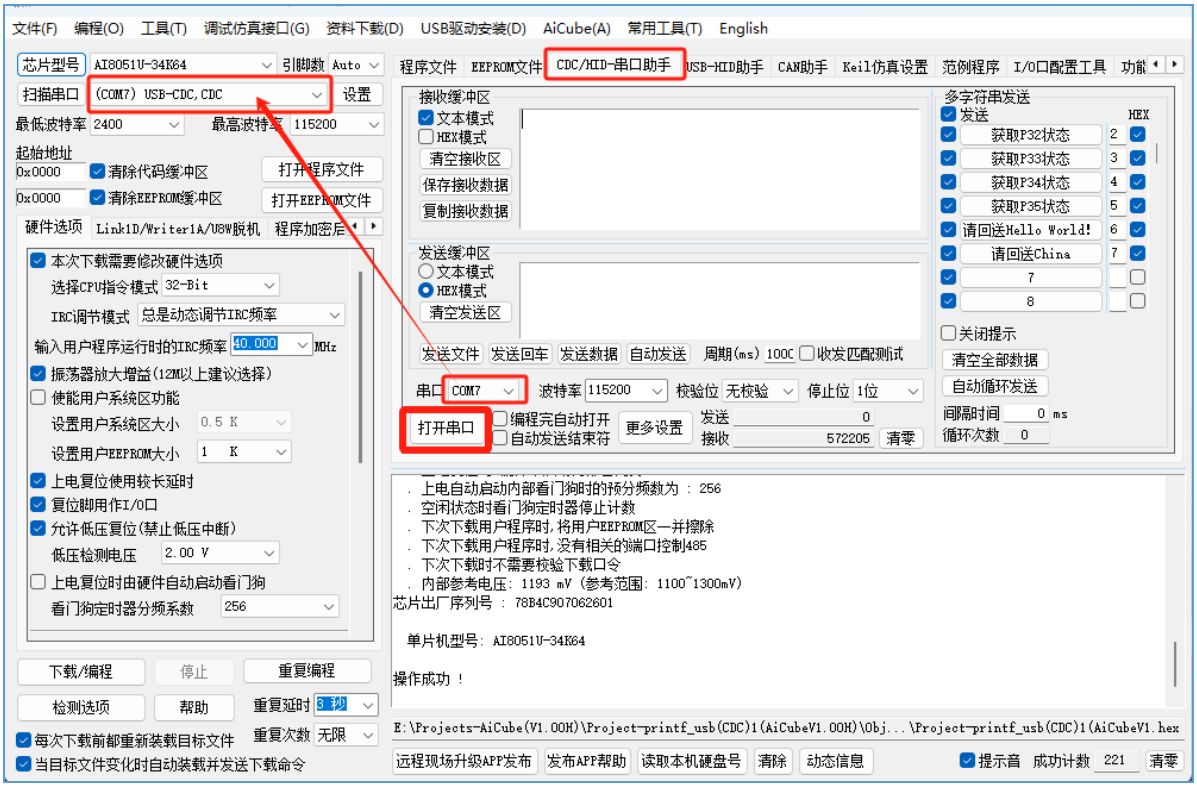


e) 使用 AIapp-ISP 软件系统和 Ai8051U 实验箱查验实验效果

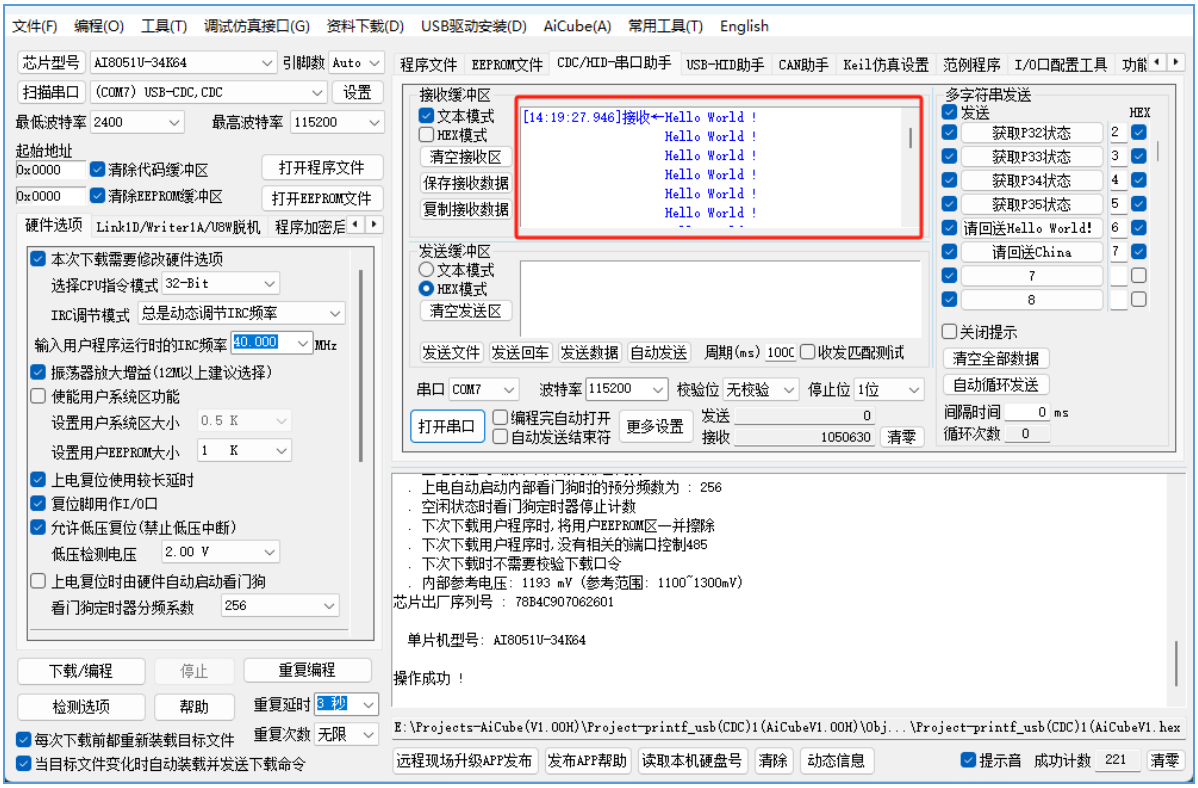
连接好 Ai8051U 实验箱，运行 AIapp-ISP 软件系统，按步骤打开并下载程序“Project-printf_usb(CDC)1(AiCubeV1.00H).hex”，如下图：



我们看到此时串口显示：“（COM7）USB-CDC，CDC”，在 AIapp-ISP 系统软件的“CDC/HID-串口助手”选项卡，找到“打开串口”按钮，如下图：



选择好对应的串口号，点击“打开串口”按钮，CDC/HID-串口助手的“接收缓冲区”，就会不停的显示“Hello World！”，如下图：



至此，printf_usb()显示“Hello World！”实验成功！

有了【AiCube】工具，单片机编程就是这么简单！快用起来吧！

2.12.6　USB 不停电下载，ISP 软件如何设置

在开发过程中，程序经常需要反复修改，重新编译后生成新的 HEX 文件，就需要停电/上电进入下载模式，很不方便。其实可以利用我们提供的 USB 不停电 ISP 下载功能，进行自动下载。

在有 USB 功能的实验中，都要加载 USB 库函数，并对 USB 进行初始化。由于在 USB 库函数的代码中已经定义了不停电 ISP 下载命令"@STCISP#"，只需要在下载软件的"收到用户命令后复位到 ISP 监控程序区"的选项卡中进行如下设置，即可实现【USB 不停电下载】功能。

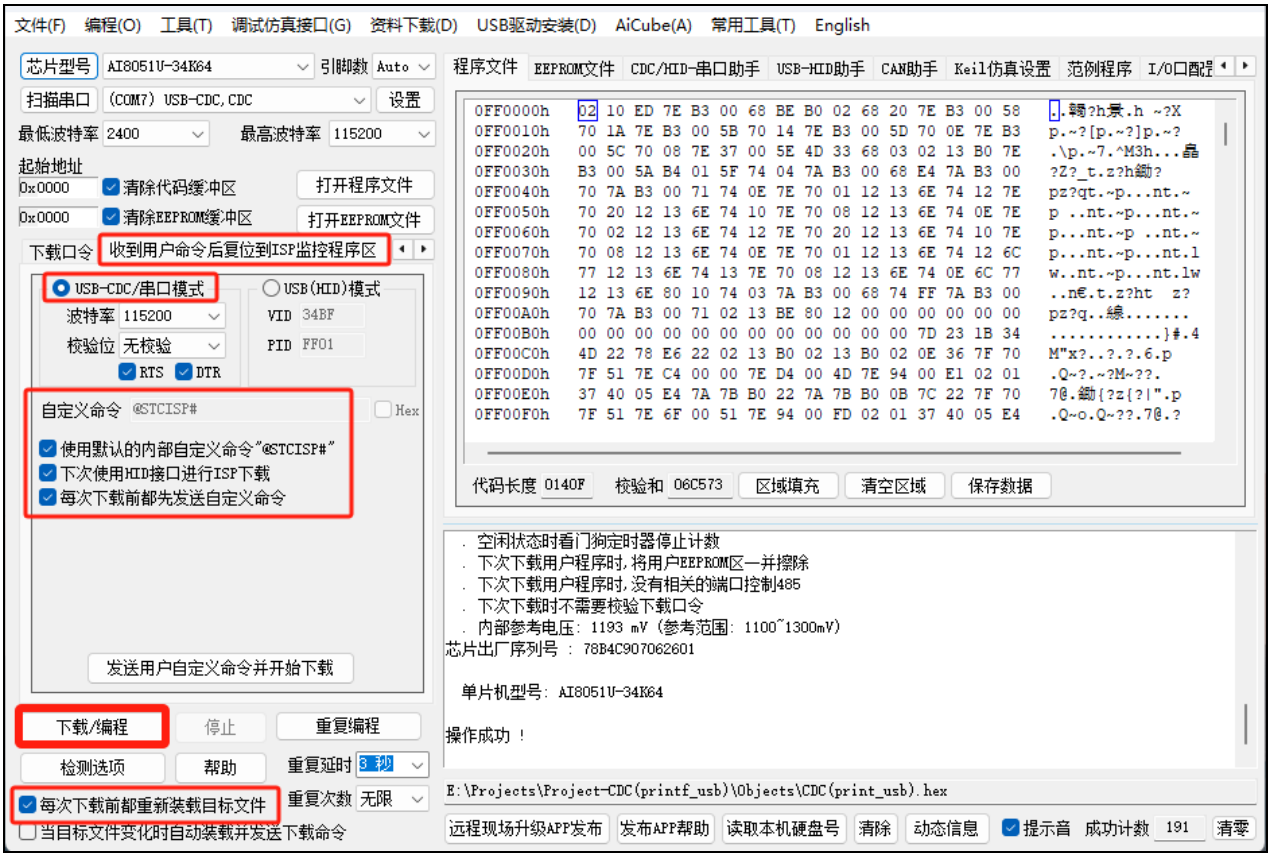
- 如使用 USB-CDC 库函数产生 USB-CDC 虚拟串口，可实现 USB-CDC 不停电自动 ISP 下载功能，并可忽略传统串口的波特率、校验位等参数的设置，下载时系统程序区还是使用的 USB-HID 通讯协议；
- 如果是传统的串口，需正确设置波特率、校验位等参数，才能实现串口的不停电自动 ISP 下载功能；
- 如果使用 USB-HID 库函数产生 USB-HID 设备，则选择"USB（HID）模式"就能实现不停电自动 ISP 下载功能，下载时系统程序区还是使用的 USB-HID 通讯协议。

ISP 软件如何设置：

2.12.6.1　手动点击【下载/编程】按钮，ISP 软件自动重新打开文件，不停电下载

如下图设置，就可以实现：

在点击"下载/编程"按钮后，ISP 软件会重新调入已打开的目标文件一次（不管目标文件有无变化），接着通过 USB-CDC 通讯协议自动发送自定义下载命令如"@STCISP#"给单片机，单片机中的用户软件收到该命令后，就会软复位到单片机的系统监控程序区，等待 ISP 下载，本设置还指定了电脑端软件后续是通过 USB-HID 通讯协议传输用户程序到目标单片机进行 ISP 下载。

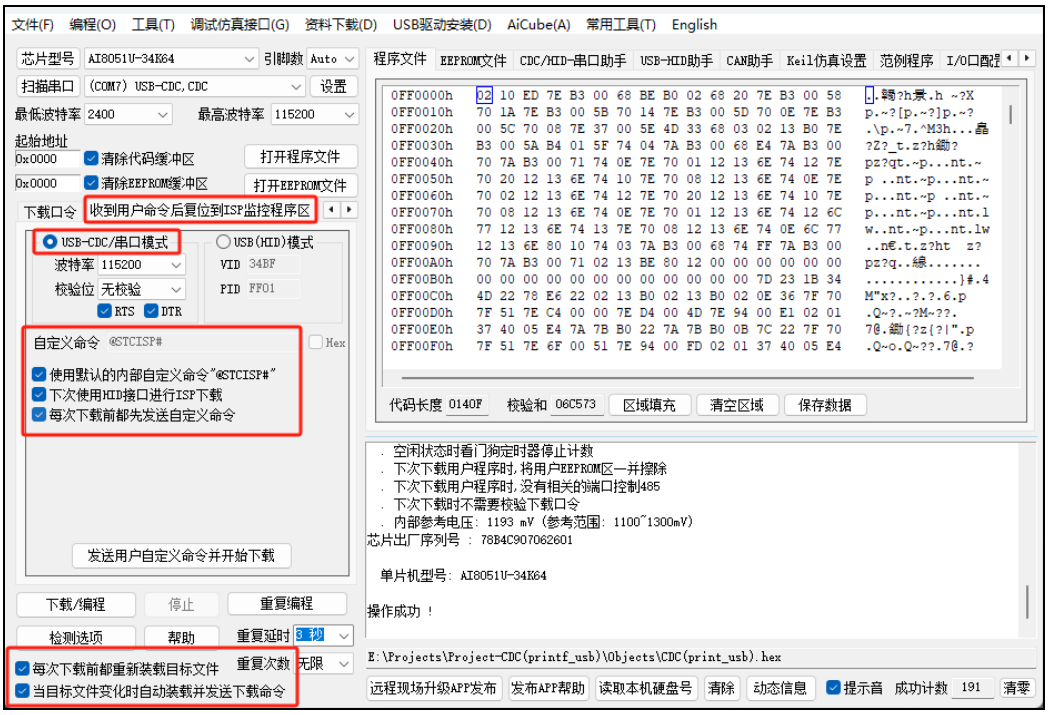


如上所示，不停电下载实验是成功的，但本设置还需要手工点击一次"下载/编程"按钮。

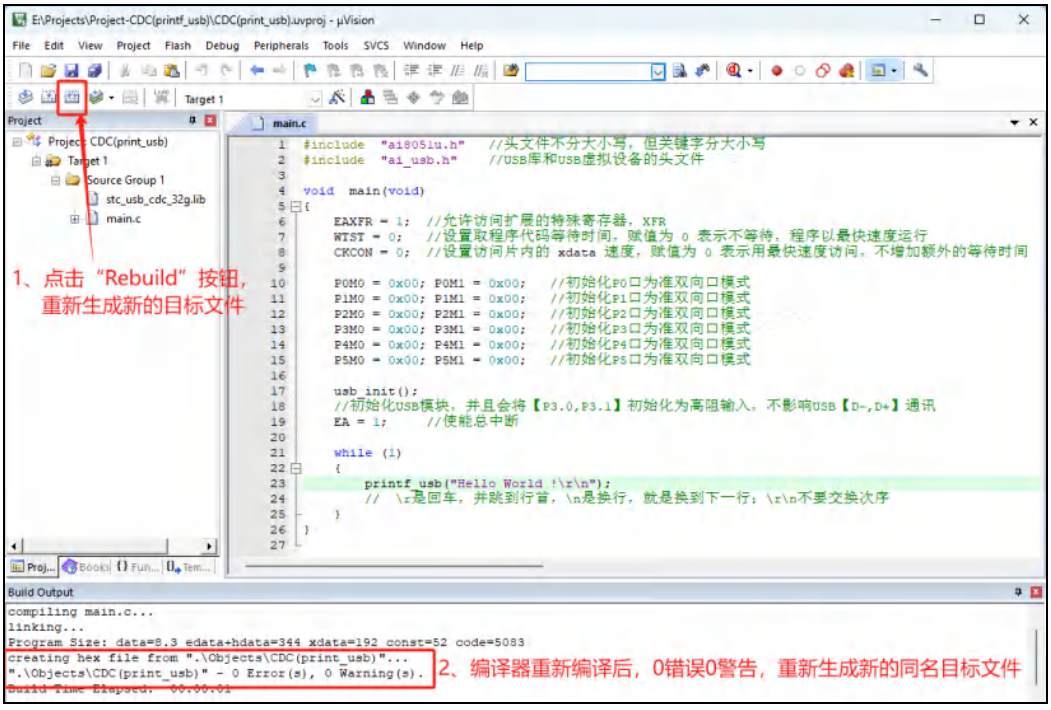
2.12.6.2　编译器重新编译后，ISP 软件会自动重新调入文件，进行不停电下载

如下图设置：

下载软件会每秒检测多次目标文件被编译的时间是否已更新，如时间已更新，则 ISP 软件会自动重新调入已打开的目标文件，接着通过 USB-CDC 通讯协议自动发送自定义下载命令如"@STCISP#"给单片机，单片机中的用户软件收到该命令后，就会软复位到单片机的系统监控程序区，等待 ISP 下载，本设置还指定了电脑端软件后续是通过 USB-HID 通讯协议传输用户程序到目标单片机进行 ISP 下载。



如上所示，不停电下载实验是成功的，但本设置的优点是不需要手工点击"下载/编程"按钮。



在编译集成环境中，点击"Rebuild"按钮，目标文件会被重新编译生成一次，ISP 软件发现目标文件被编译的时间发生变化后，ISP 软件就会执行前面描述的不停电下载动作。

2.13 实验二：查询方式，查询到电脑命令后，printf_usb("Hello World !\r\n")或其他

看下面的程序，其实还是那个熟悉的标准 C 语言的 printf()，只是到了单片机 C 语言中，变成了 printf_usb()

2.13.1 printf( )，printf_usb( )实验程序代码

```
#include   "ai8051u.h"   //头文件不分大小写，但关键字分大小写
#include   "ai_usb.h"    //USB 库和 USB 虚拟设备的头文件，需要在项目文件中添加对应的 USB 库
//由于在 USB 库函数的代码中已经定义了，不停电 ISP 下载命令字"@STCISP#"，
//只需要在下载软件的“收到用户命令后复位到 ISP 监控程序区”选项卡中进行相应的设置，
//即可实现 USB 不停电 ISP 下载功能，或串口不停电 ISP 下载功能。
void main(void)
{
    EAXFR = 1;   //允许访问扩展的特殊寄存器，XFR
    WTST = 0;    //设置取程序代码等待时间，赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;  //设置访问片内的 xdata 速度，赋值为 0 表示用最快速度访问，不增加额外的等待时间

    usb_init();                         //初始化 USB 模块
    //初始化 USB 模块，并且会将【P3.0, P3.1】初始化为高阻输入，不影响 USB【D-, D+】通讯
    EA = 1;                             //总中断允许位打开

    while(1)
    {
        if(bUsbOutReady)
        {    //查询方式：USB-Device 单片机查询是否有接收完成的 USB 主机送来的数据包
            //“bUsbOutReady”:主机发送给 HID/CDC 设备的数据状态标志。
            //USB 库中有定义 bUsbOutReady 位变量，查询方式必须调用一次“usb_OUT_done();”
            //状态位为 1 表示，主机发送给 HID/CDC 设备数据已完成，
            //设备已成功接收一包数据，等待对数据继续处理。

            if(UsbOutBuffer[0]==6)
                printf_usb("HelloWorld!\r\n");
                    //接收到的数据包的第 0 个字节是 6，打印“HelloWorld！”
            else if(UsbOutBuffer[0]==7)
                printf_usb("China!\r\n");
                    //接收到的数据包的第 0 个字节是 7 后，打印“China！”

            usb_OUT_done();  //USB 查询方式，Device 单片机通知【主机可以发下一个数据包】
                //当使用查询方式 if(bUsbOutReady)，查询调用 USB 库函数时，在数据处理完成后，
                //必须调用一次“usb_OUT_done();”，这样才能接收下一包数据。
                //如果是使用中断方式调用 USB 库函数，用户程序不需要调用“usb_OUT_done();”
                //中断方式，USB 库函数内部会自动处理。
        }
    }
}
```

以上是完整的第一个程序代码

printf_usb()函数：

printf_usb()函数和标准 C 语言的格式完全一致，详见 printf()函数章节

进入 printf_usb()函数后，首先会判断 USB 主机是否已经对 USB 设备配置完成，

如果没有配置完成，printf_usb()函数会直接返回退出。

如果已经配置完成，还会继续判断如有发送前一笔数据，则判断是否发送完成，

如果没有发送完成，会继续等待，直到前一笔数据发送完成，

才会发送新的 printf_usb()需发送的数据后再返回退出。

但如等待超过 50ms@24MHz，会直接退出。

如果主时钟不是 24MHz，20MHz 等待时间会变长，40MHz 等待时间会变短。

如果已经配置完成且前一笔数据已经发送完成，

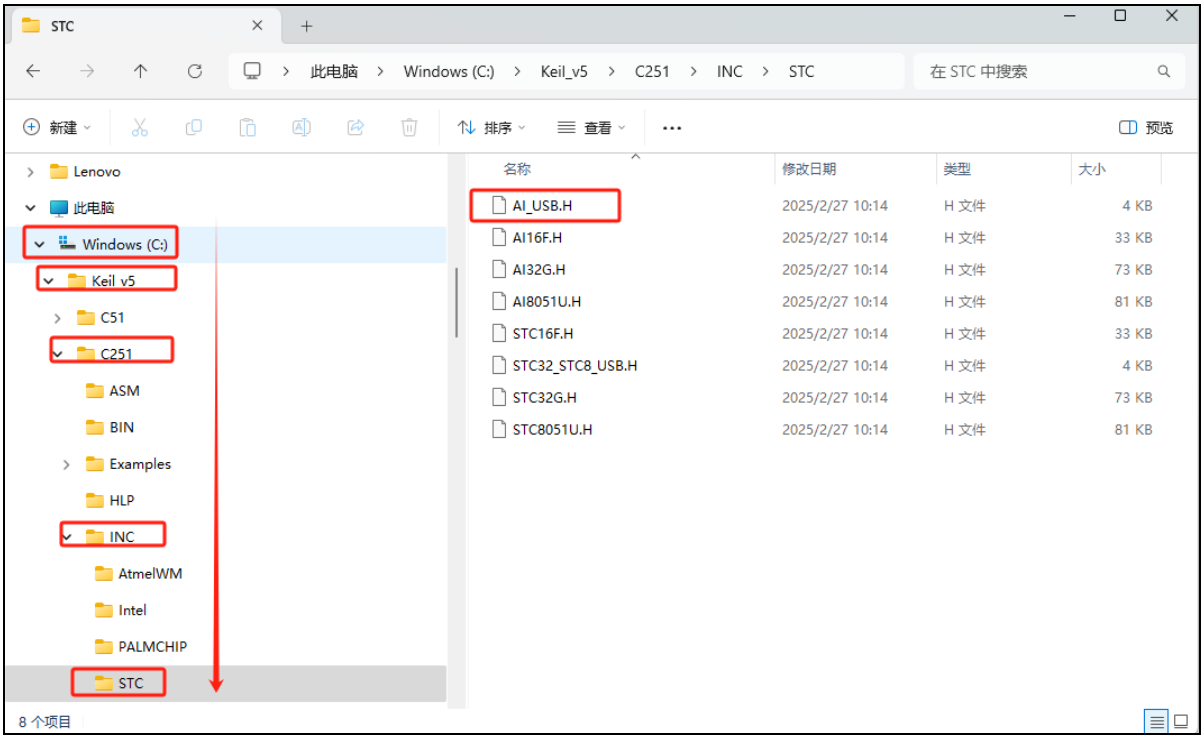
则会直接发送新的 printf_usb()需发送的数据并返回退出。

2.13.2　准备工作，如已准备，跳过此节

1、添加常用头文件 ai_usb.h

```
#include "ai_usb.h"                    //使用 usb 所需要包含的头文件。
```

这个头文件在使用 AIapp-ISP 软件添加型号和头文件到 Keil 中时就已经一并被添加到 Keil 中了。

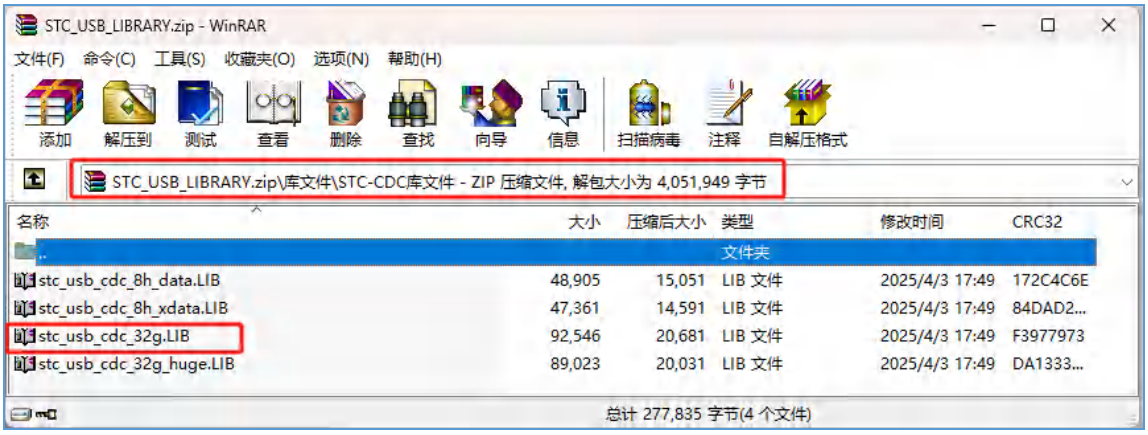


2、添加 usb_cdc 库函数文件

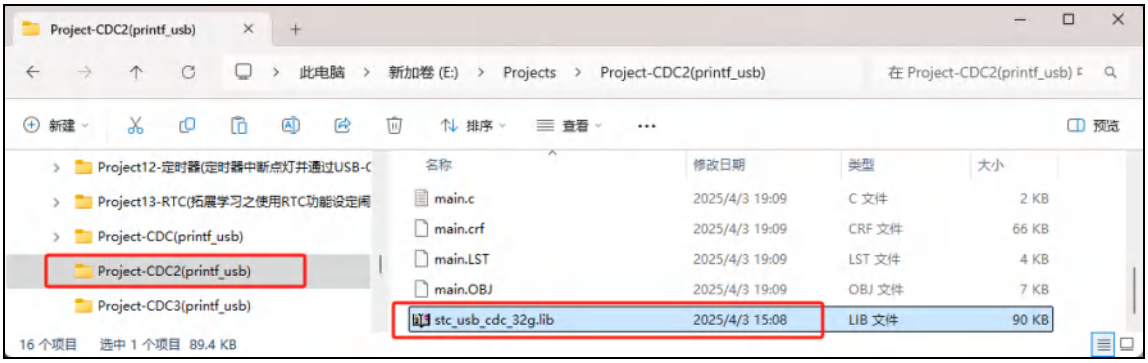
因为要使用到 USB 功能，必须在项目中添加 USB 库函数。这部分可以到 STCAI 的官网：

https://www.stcai.com/khs 的 USB 库文件中进行下载

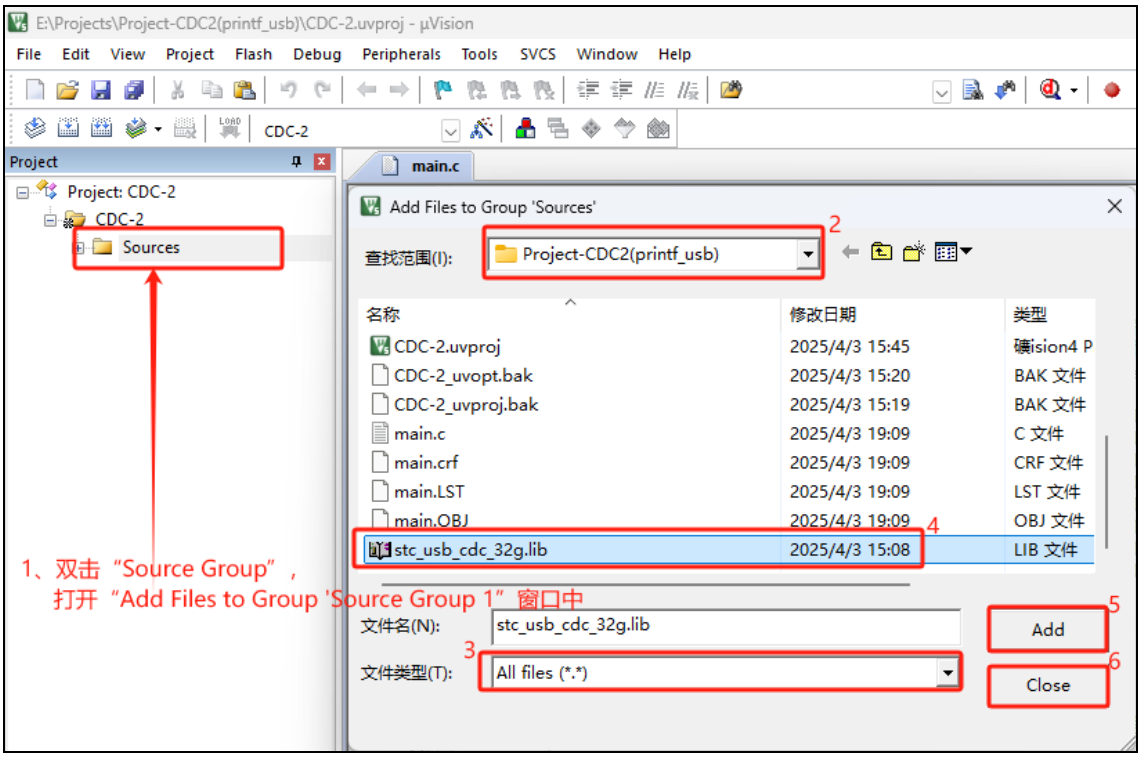
下载完成后，打开压缩包“STC_USB_LIBRARY.zip”，
在’库文件\查询模式 STC-CDC 库文件’中，复制出来 stc_usb_cdc_32g.LIB



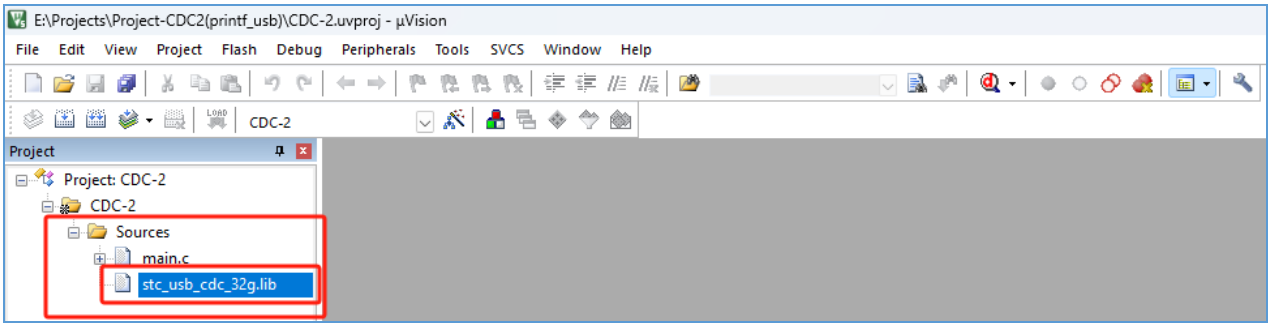
将库函数“stc_usb_cdc_32g.LIB”复制到创建工程所在的文件夹 Project-CDC2(print_usb)中：



将库函数 stc_usb_cdc_32g.LIB 添加进入此工程项目中
在 Keil 中，双击 Source Group，在弹出的 Add Files to Group 'Source Group 1 窗口中，将库函数 stc_usb_cdc_32g.LIB 添加进入此工程，就可以使用 USB-CDC 功能。



这样，Keil 界面变化如下图：

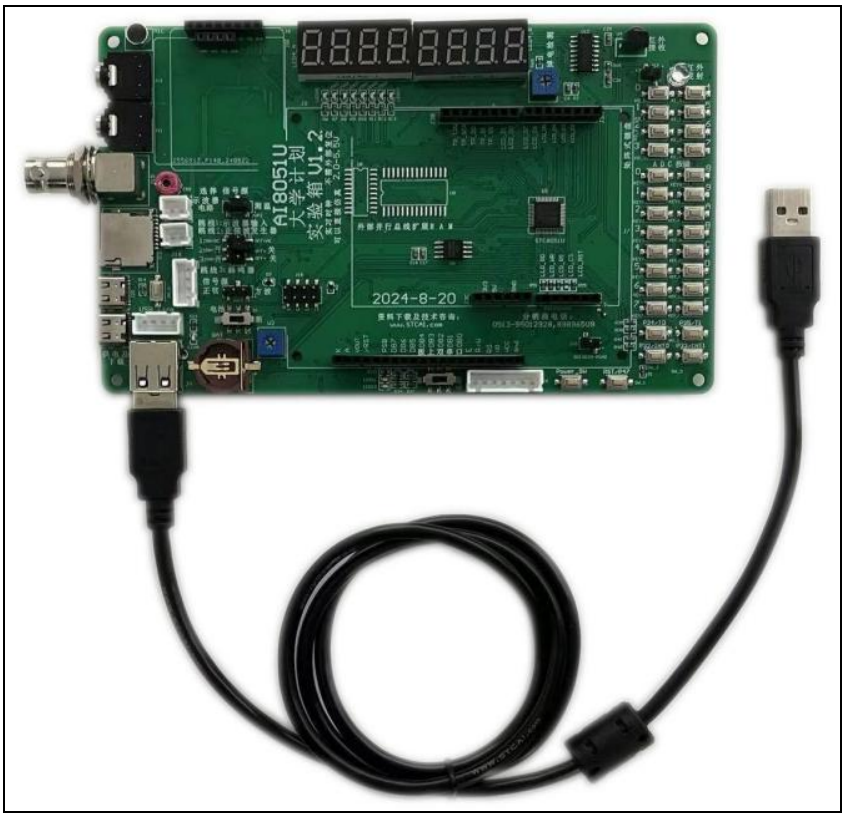


【注意】：USB-CDC 串口则更类似传统串口的形式，使用 COM 口进行通讯，但是只有 Win10（1903）及以后版本不需要安装 USB-CDC 驱动，较早的 Windows 版本才需要运行 ISP 软件中的 USB 驱动安装，手动安装，手机自带 USB-CDC 驱动。USB-HID 通讯不需要安装驱动，各种系统都自带 HID 驱动。

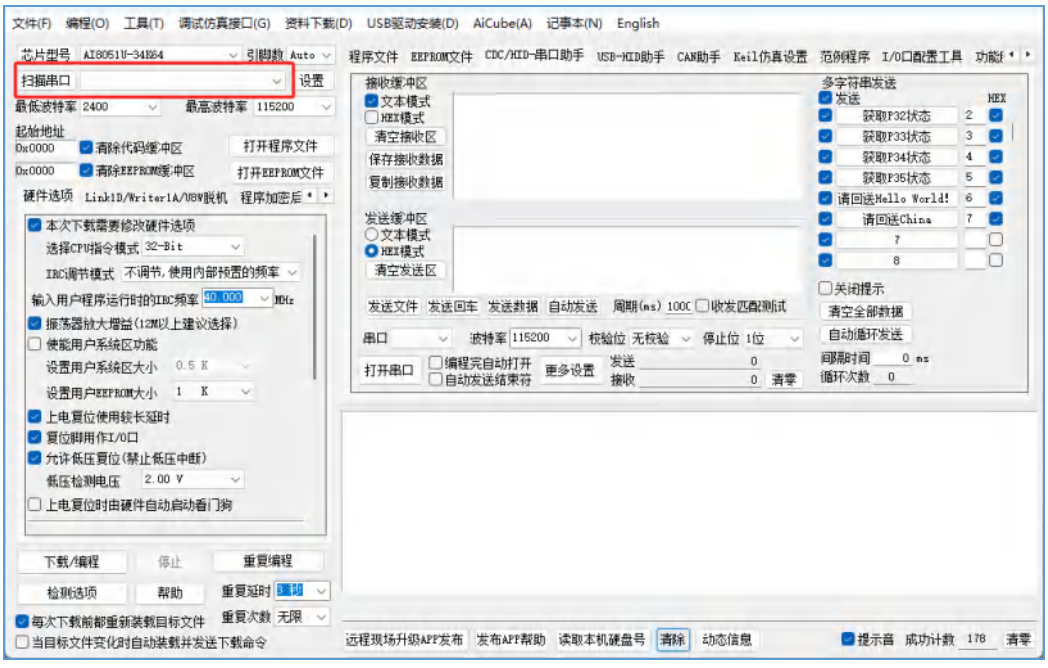
2.13.3　下载用户程序到实验箱

2.13.3.1　连接实验箱和电脑

拿出实验箱中配置的 USB-TypeA 到 USB-TypeA 线将实验箱与电脑正确连接。

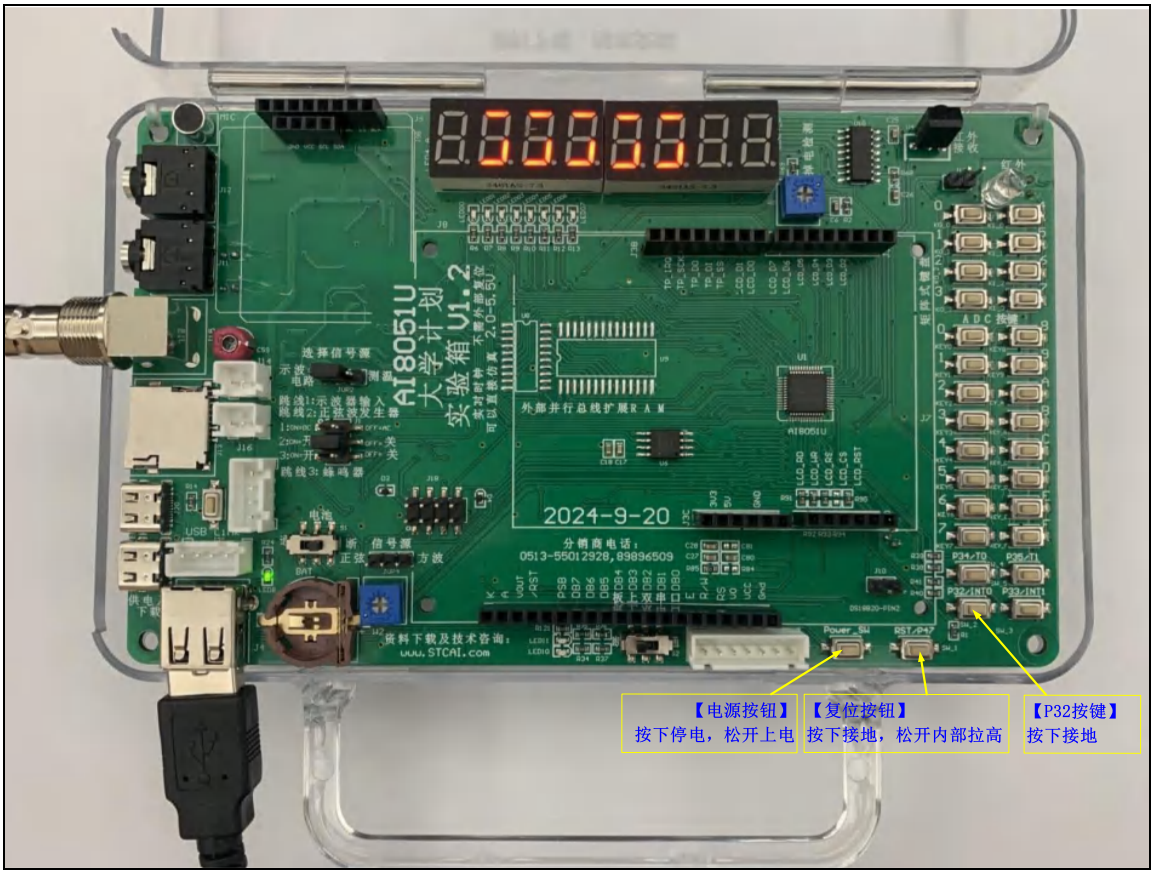


然后打开 AIapp-ISP-v6.95K 或以上版本。选择单片机型号：Ai8051U-34K64，如下图，我们看到“扫描串口”后面的端口列表框不是“（HID1）USB-Writer”，说明还不能下载程序到实验箱。

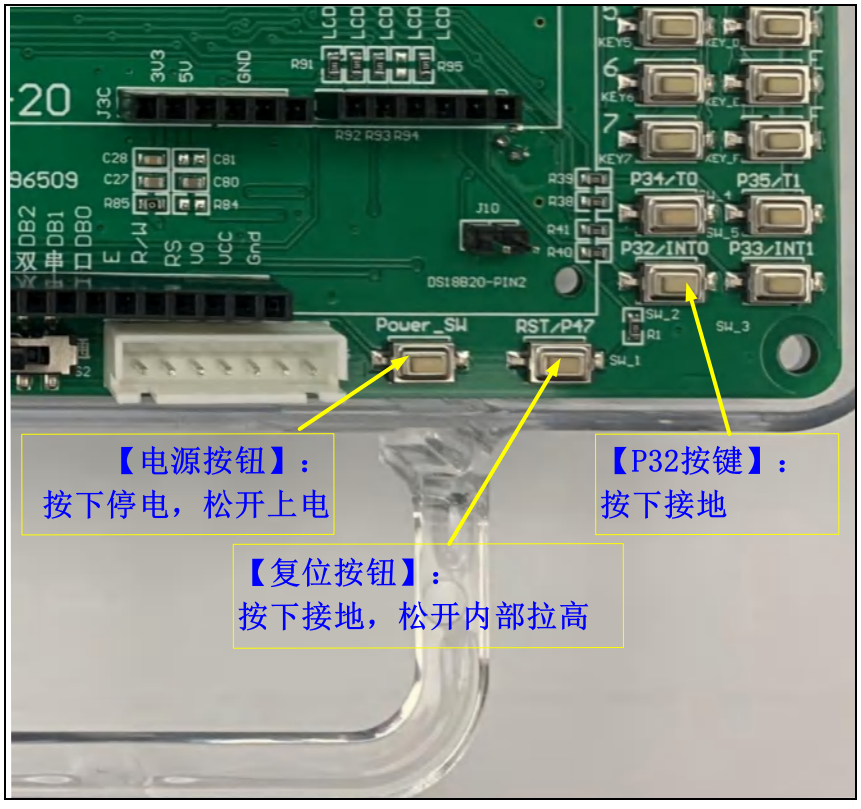


2.13.3.2 硬件 USB 下载用户程序步骤

下图是 Ai8051U 实验箱已经和电脑相连的正面图片，数码管是随机显示的，**此实验的目的把 Ai8051U 实验箱作为 USB-CDC 设备和电脑进行通讯，收发数据。**



局部放大实验箱图片，其正下方有两排按键，找到 P3.2 按键和 Power_SW 按键



【方法一，P3.2 按键，再结合停电上电下载】

1、按下板子上的 P3.2 按键，就是 P3.2 接地

2、给目标芯片重新上电，不管之前是否已通电。

===本 Ai8051U 实验箱是：按下电源按钮停电，再松开电源按钮上电，

这时电脑端 ISP 下载软件会自动识别出“(HID1) USB Writer”，此时就与 P3.2 状态无关了，可以松开 P3.2 按键（P3.2 在用户程序区可做其它任意用途）

3、点击电脑端下载软件中的【下载/编程】按钮（注意：USB 下载与串口下载的操作顺序不同）

电脑端 ISP 下载软件提示：下载进行中…，几秒钟后，提示下载成功！

【方法二：复位管脚低电平复位下载】

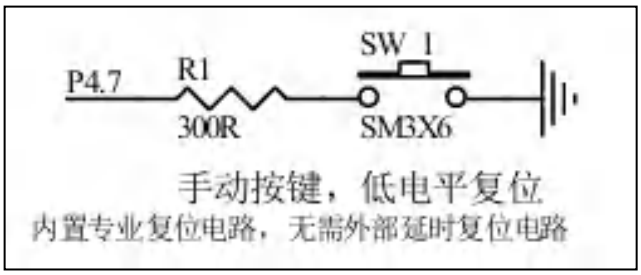
USB 连接好并已上电的情况下，外部按键复位也可进入 USB 下载模式，注意：

P4.7-nRST 出厂时默认是 P4.7-I/O 功能，要改为复位功能，需 ISP 烧录时取消设置复位脚用作 I/O 口，停电一次再上电才生效，程序区中用户程序也可改为复位脚或 I/O，这个立即生效。

1、按下复位管脚/P4.7-nRST 外接的低电平复位按键，复位 MCU，

再松开复位键，MCU 首先从系统程序区启动运行系统程序，判断是否要下载用户程序，

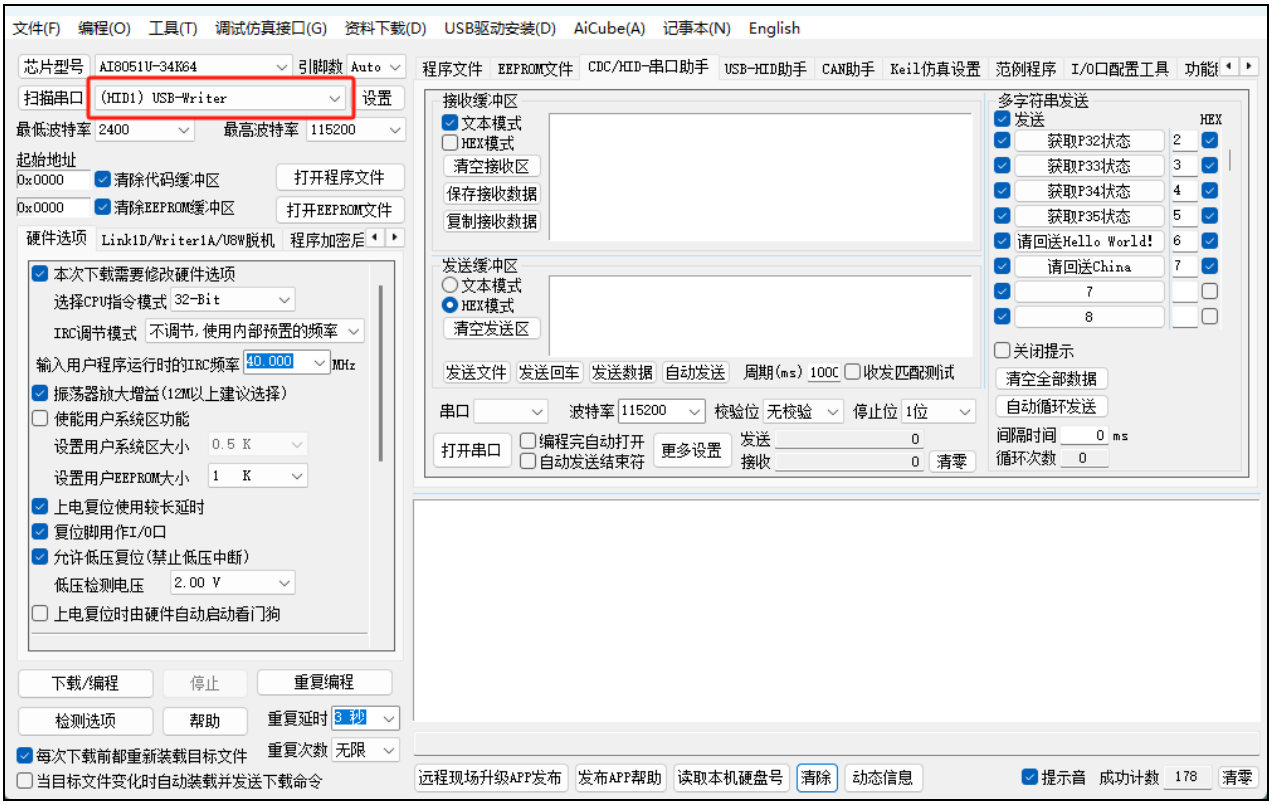
等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后



2、点击电脑端下载软件中的【下载/编程】按钮

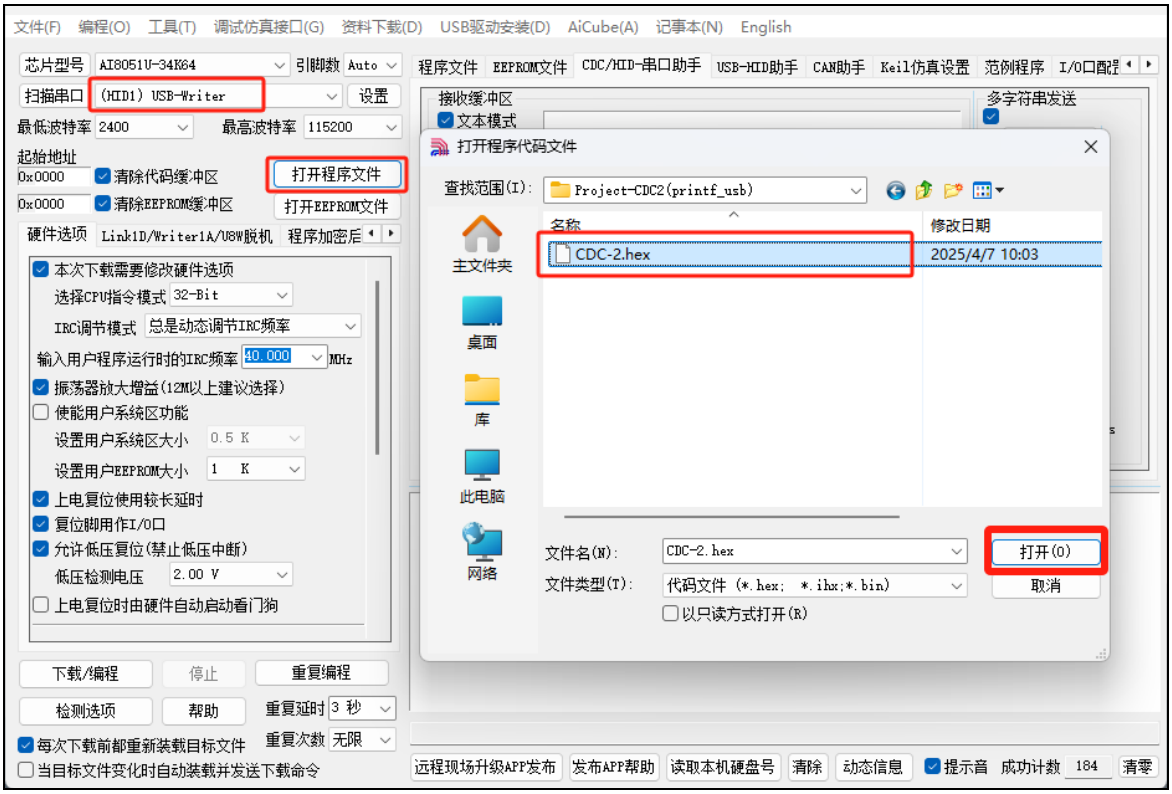
电脑端 ISP 下载软件提示：下载进行中…，下载成功！

备注：如果目标 MCU 的用户程序区已有 USB 通讯程序（含 CDC、HID 等），则通过复位脚复位进入 USB 下载模式不被支持，只能通过停电上电冷启动下载或者从用户程序区软复位到系统区下载

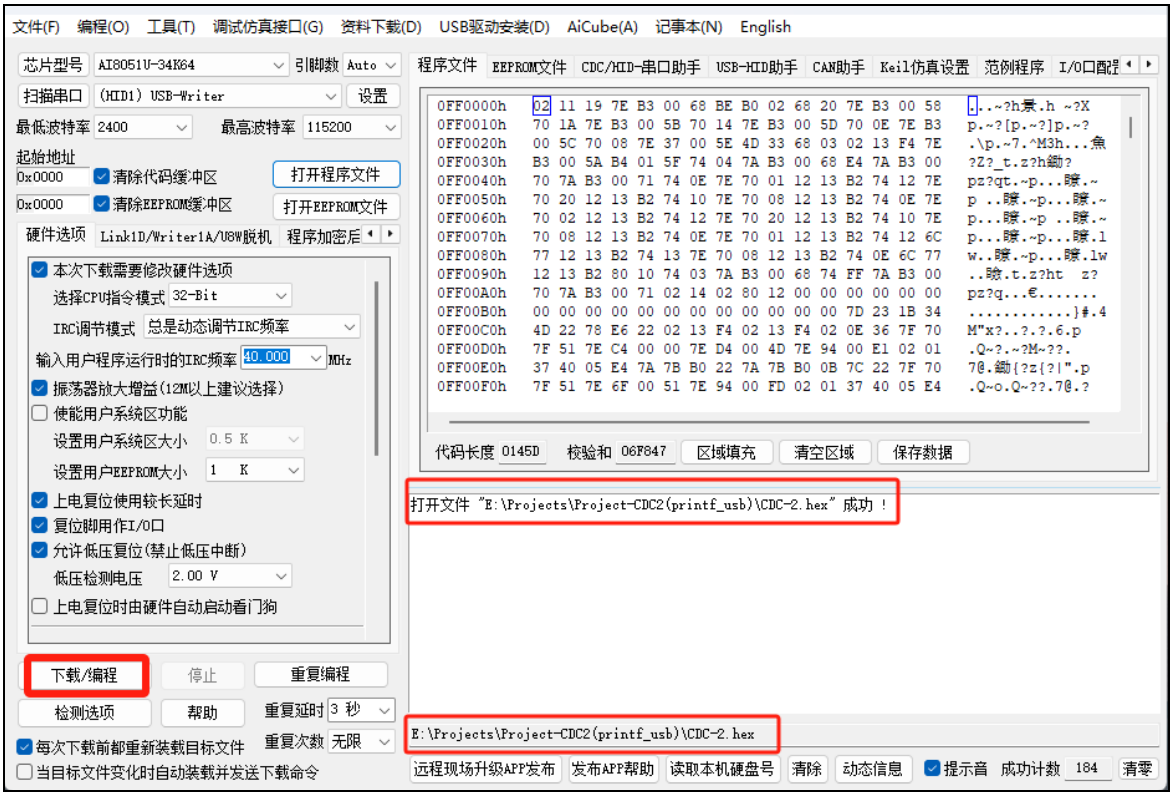


2.13.3.3 下载用户程序文件

现在就打开用户程序文件，点击 AIapp-ISP 软件界面上的“打开程序文件”（打开的是项目程序文件编译无误后生成的 HEX 文件，一般默认存放在该项目的 Objects 目录中），调入在前面小节编译好的 HEX 文件：

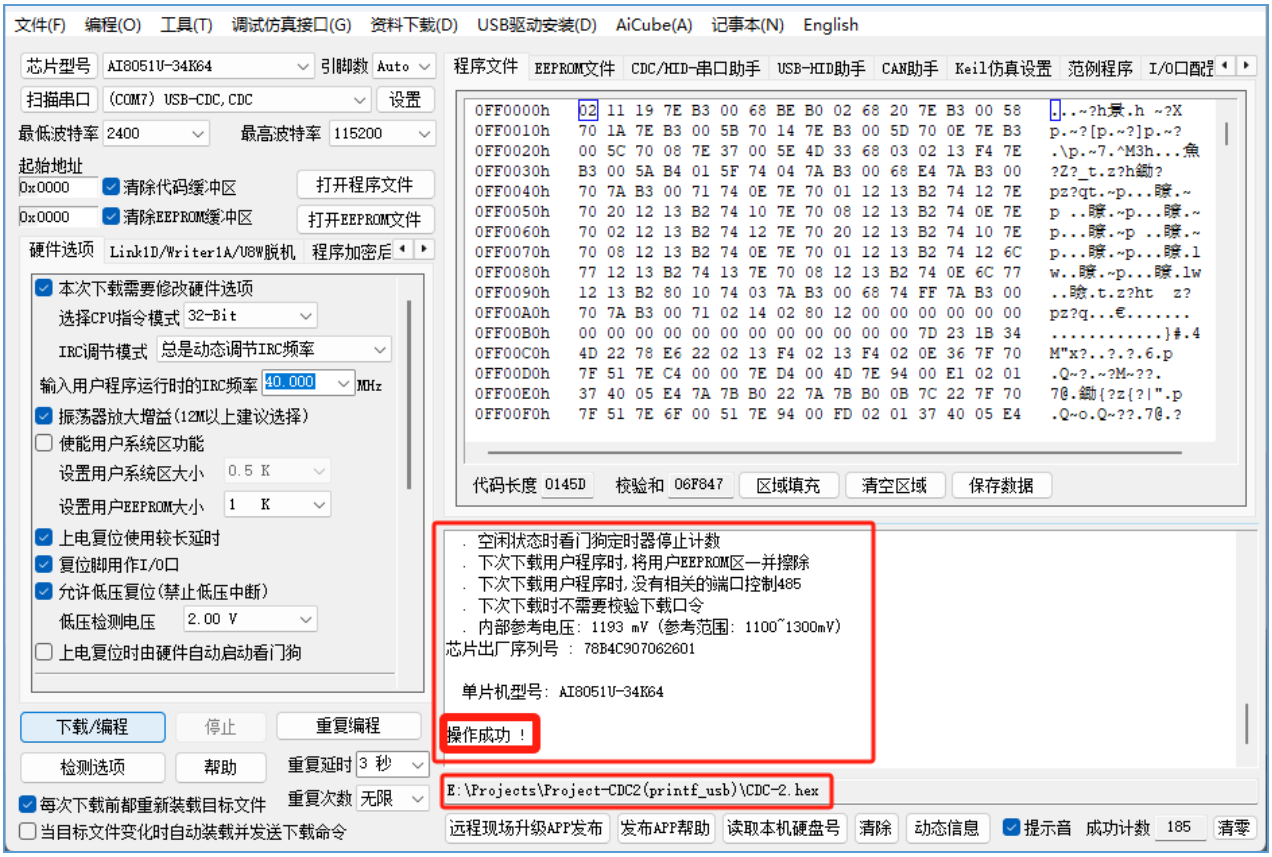


然后，AIapp-ISP 软件界面如下图所示，出现“‘打开文件***.hex’成功！”

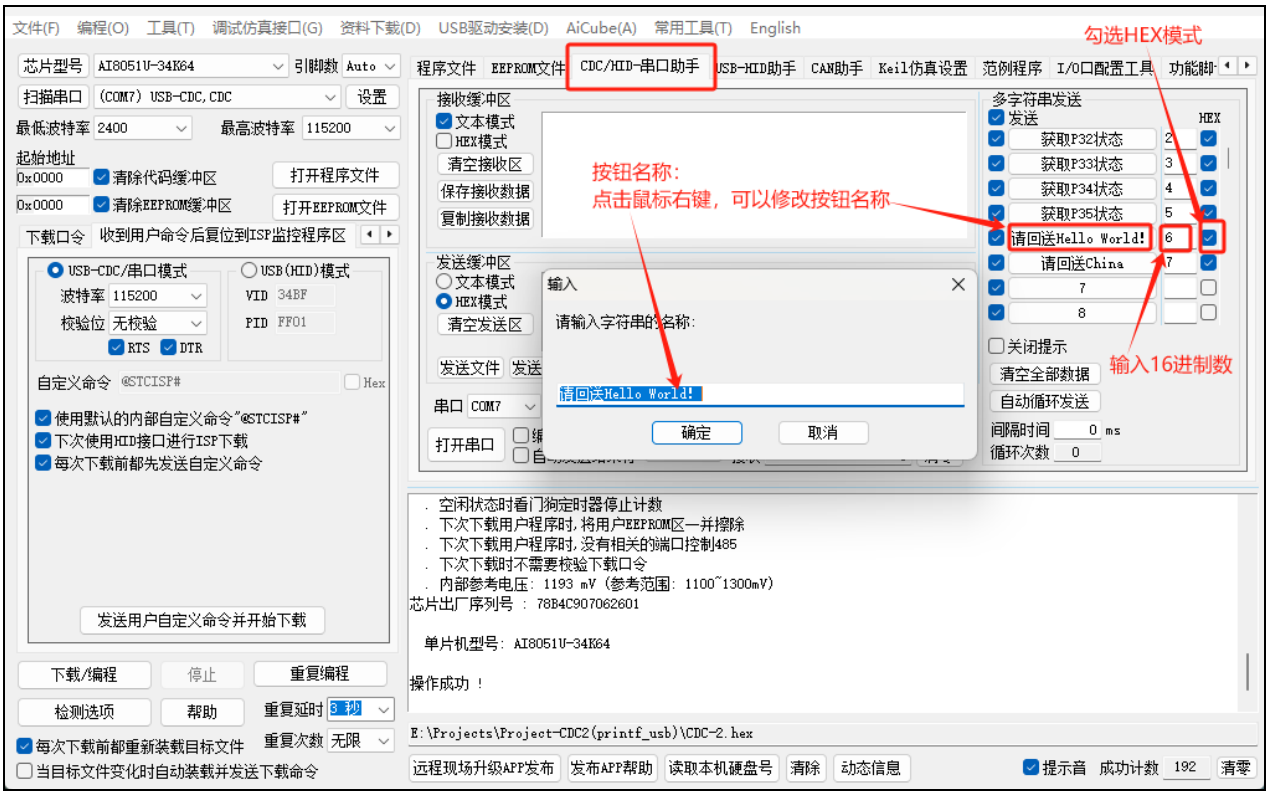


点击“下载/编程”，把 CDC-2.hex 烧录到实验箱，AIapp-ISP 软件界面出现：操作成功！

恭喜你！



我们看到此时串口显示：（COM7）USB-CDC，CDC，选择 AIapp-ISP 系统软件的“CDC/HID-串口助手”选项卡，在“多字符串发送”窗口，设置好发送控制命令，以便观察实验效果，如下图：

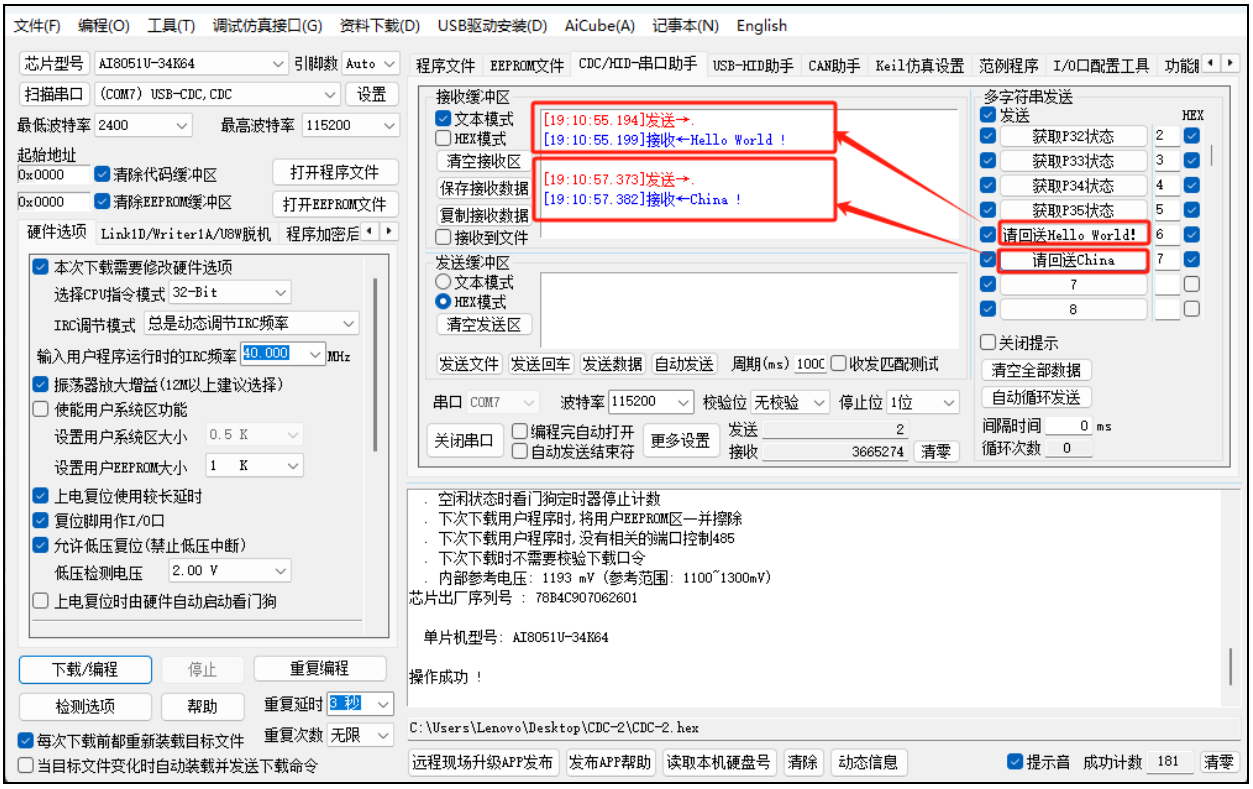


一切准备就绪！

2.13.4　查看实验效果

点击“请回送 Hello World！”按钮，单片机收到 16 进制数 6 后，回送“Hello World！”给 USB 主机显示。
点击“请回送 China！”按钮，单片机收到 16 进制数 7 后，回送“China！”给 USB 主机显示。

如下图：



完成实验验证！

2.13.5　使用【AiCube】工具创建“查询方式，查询到电脑命令后，printf_usb("Hello World !\r\n")或其他”

STCAI 在最新版的 AIapp-ISP 软件中，推出【AiCube】项目创建工具，目前支持 Ai8051U-32Bit 各功能模块的智能化配置，并生成程序框架。这项功能在程序员建立项目时，简化了项目的初始化难度，提高效率，避免因不太熟悉或不完全了解该单片机的性能而忽略个别配置，引起编译错误等问题。尤其对刚涉足单片机的初学者，此工具不仅引导其了解单片机编程流程，还极大的降低了入门门槛，缩短了摸索的时间，提高入门的兴趣和信心。现在我们使用这个工具来生成实现此实验功能的配置和代码。

a) 安装 AIapp-ISP 下载/编程/烧录/软件

安装 STCAI 公司研发的 AIapp-ISP 下载 / 编程 / 烧录 / 软件，含强大的辅助开发工具包。登录 STCAI 官网，下载最新版的 AIapp-ISP 安装包，下载链接如下：https://www.stcai.com/gjrj

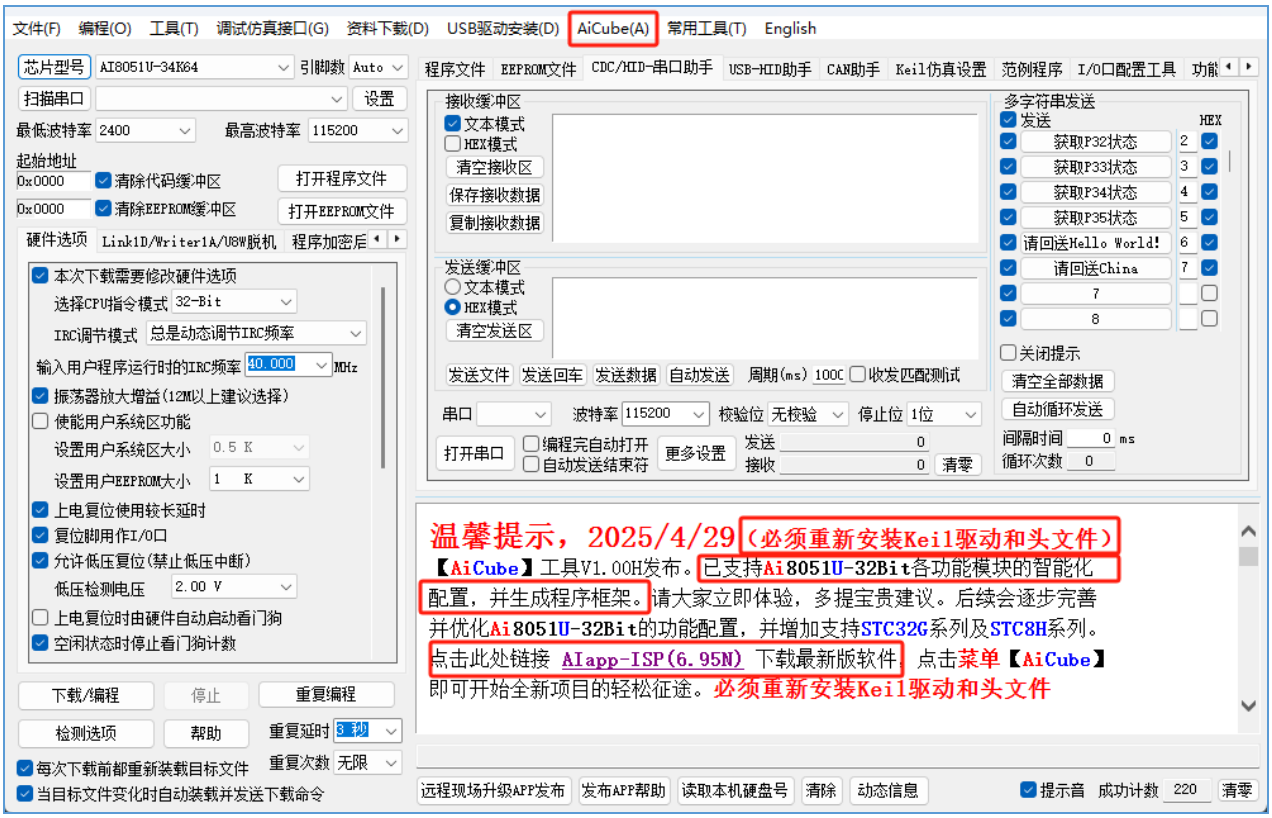
下载下来的是压缩包文件如下图：

双击后解压缩如下图：

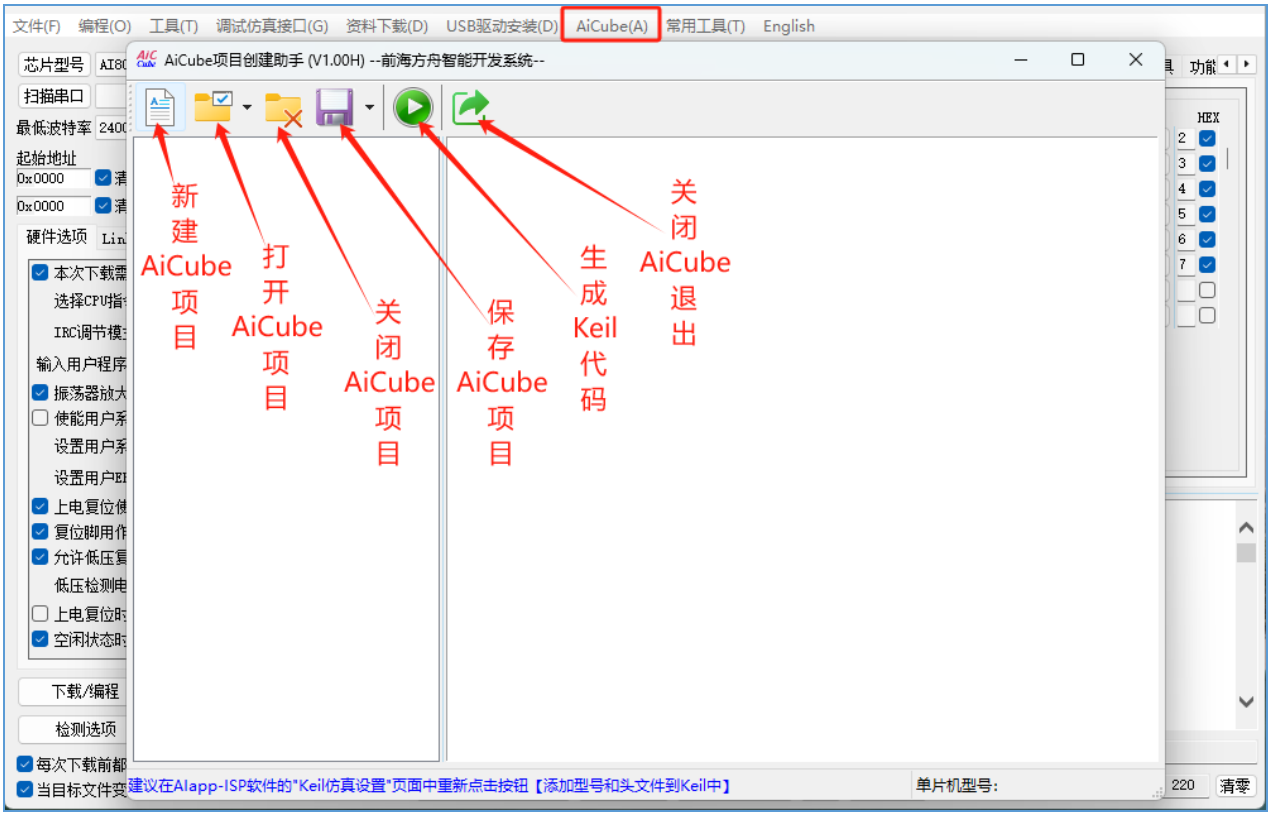
如果是解压缩到非桌面的其他目录，如 D:\STCAI-ISP，建议创建快捷方式到桌面方便打开。这实际是一个免安装的绿色软件。

b) 打开【AiCube】

双击图标，打开 AIapp-ISP 软件界面，即可直接使用，如下图，在菜单栏上找到【AiCube(A)】菜单：

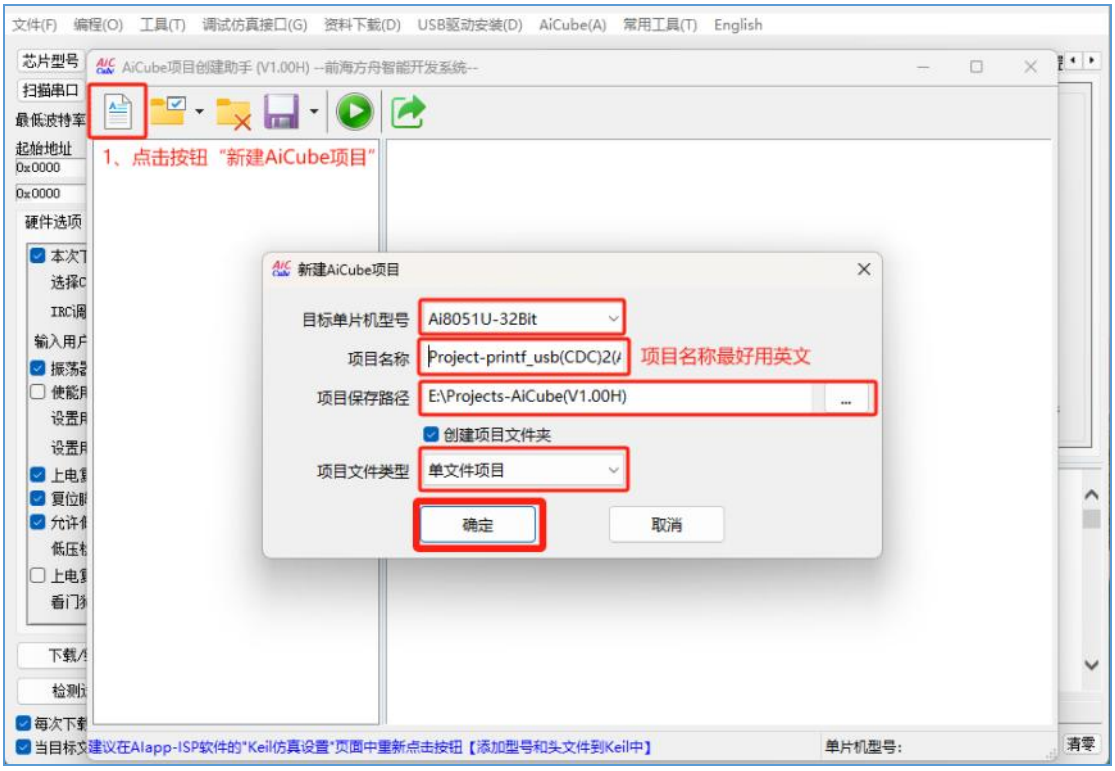


点击【AiCube(A)】，弹出如下窗口：

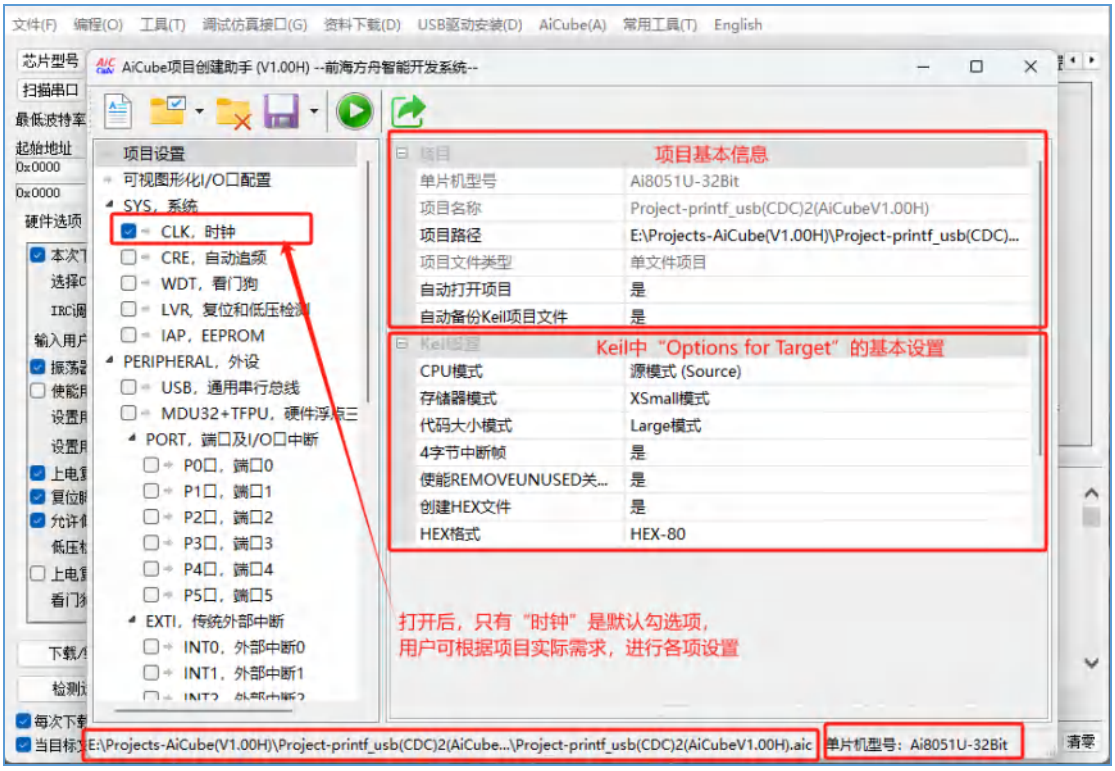


c) 使用【AiCube】创建项目

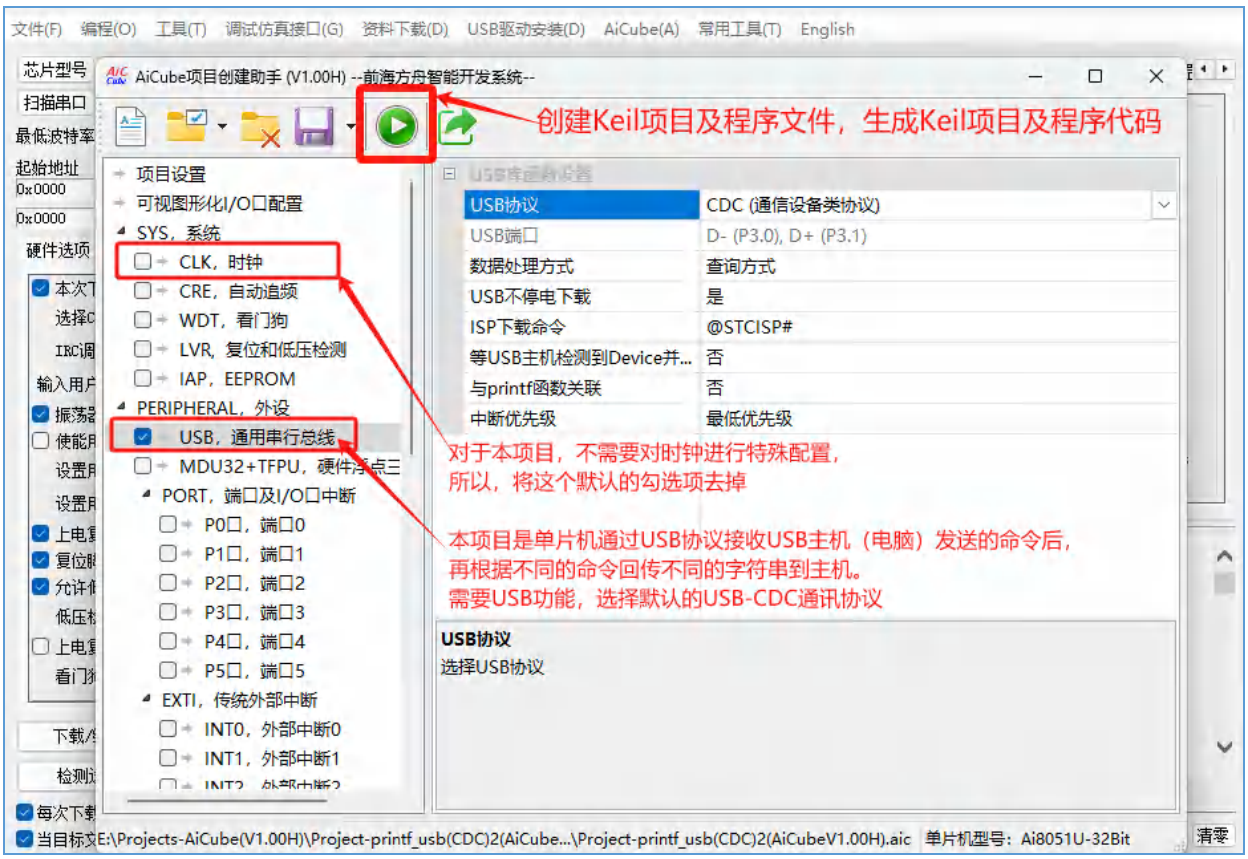
点击按钮“新建 AiCube 项目”，出现如下浮窗，选择并填写项目基本信息，建议项目名称用英文！



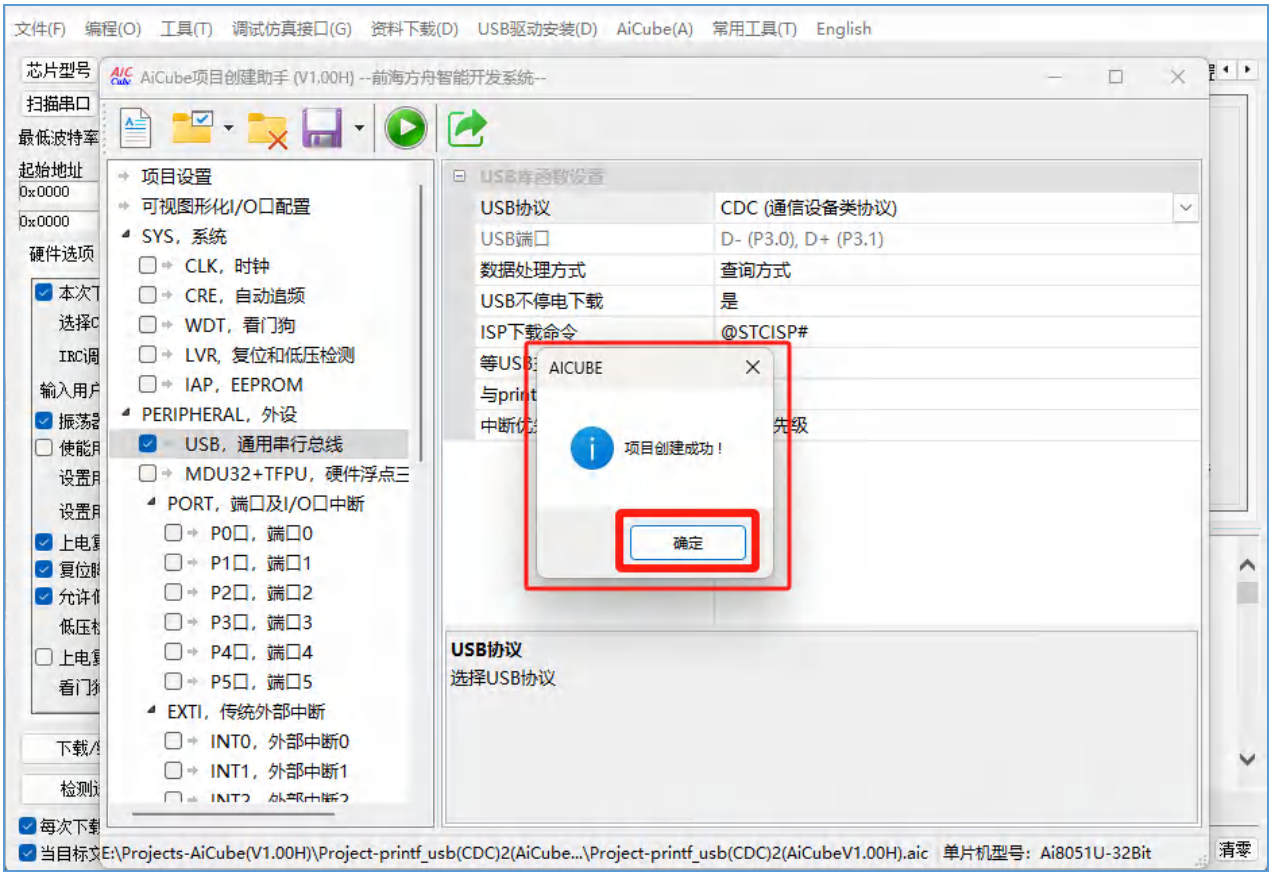
点击“确定”，出现如下界面：



因本项目用到 USB-CDC 功能，要加载 USB-CDC 库函数，对 USB 初始化等，使用【AiCube】助手根据本项目的实际需求，对配置进行如下修改：



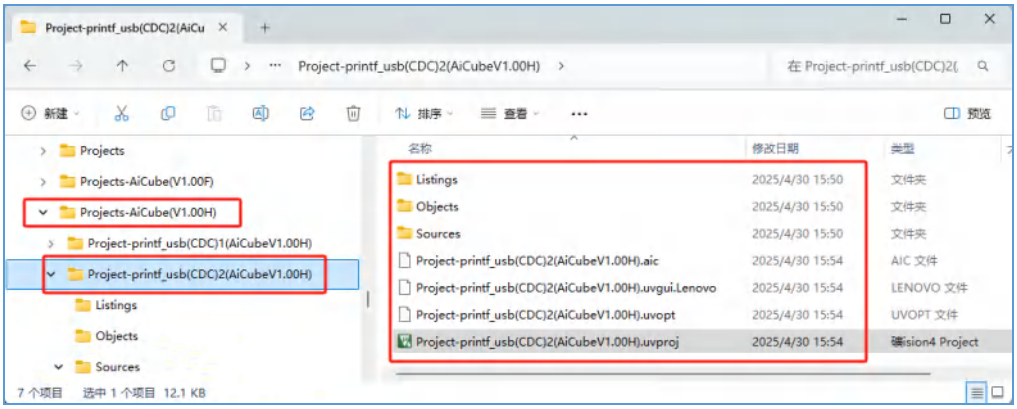
点击按钮“创建 Keil 项目及程序文件”，【AiCube】助手提示：“项目创建成功！”，如下图：



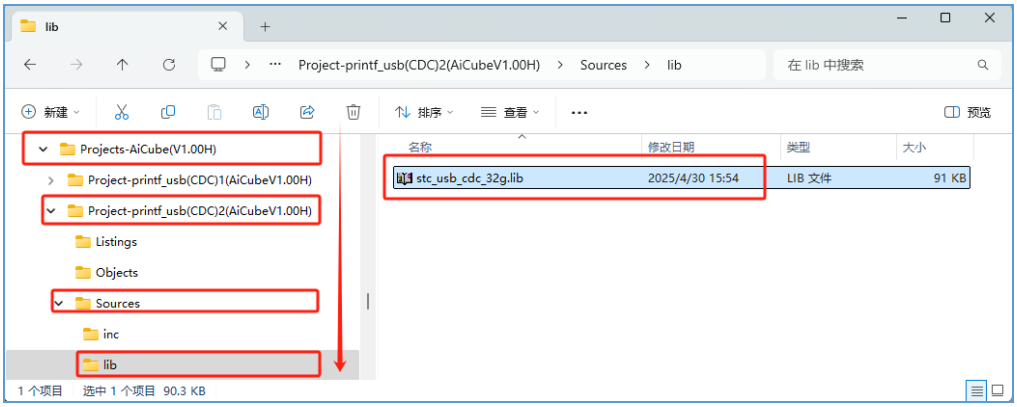
点击“确定”按钮后，系统会自动打开 Keil 软件，调入新创建的 Keil 工程项目，本实验创建的项目是“Project -printf_usb(CDC)2(AiCubeV1.00H)”。现在，可以关闭【AiCube】工具，当然，也可以不关闭。

➲ 找到并打开刚刚创建的项目目录

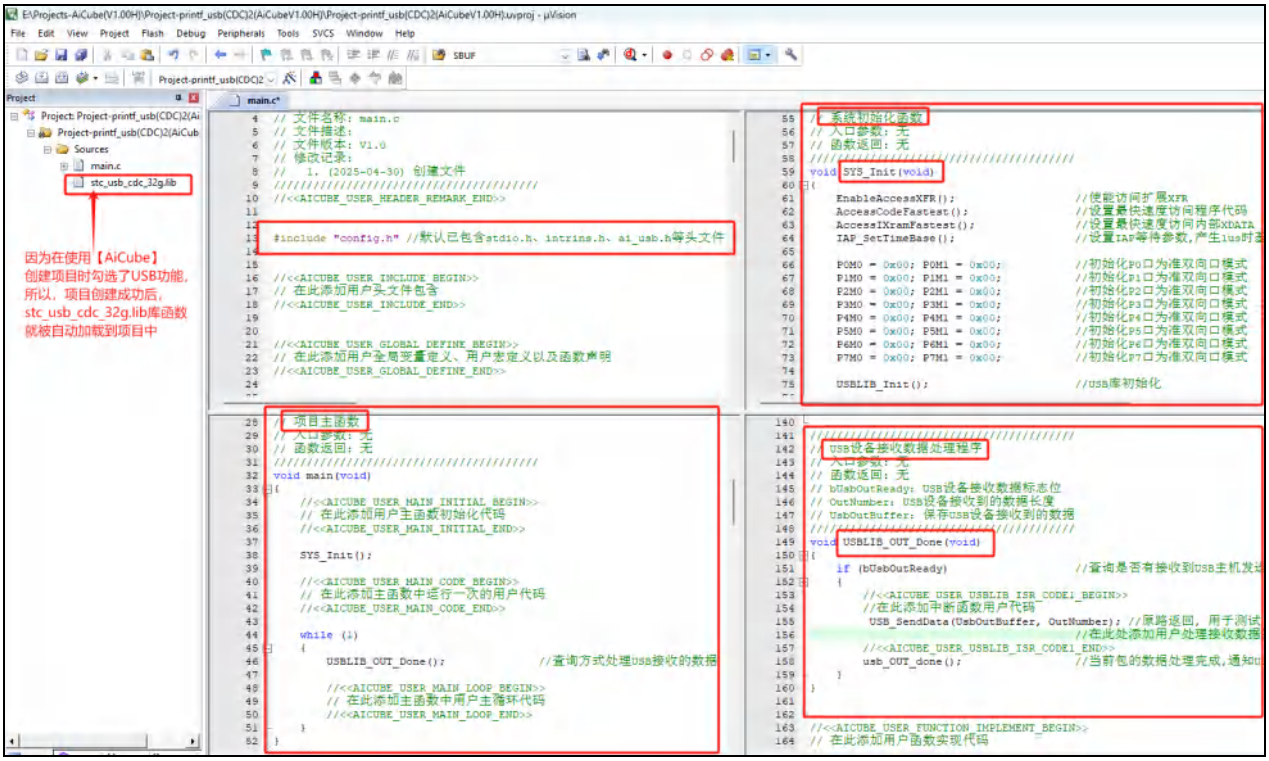
上述操作过程，【AiCube】工具已经帮我们建立了项目目录及 Project-printf_usb(CDC)2(AiCube V1.00H).uvproj 文件，如下图：



再打开 Sources>lib 子目录，发现，【AiCube】将相应的 USB 库函数加载进项目目录：



➲ 观察已打开的 Keil 项目文件：Project-printf_usb(CDC)2(AiCubeV1.00H).uvproj，如下图：



根据在【AiCube】中的配置，从上图中我们看到【AiCube】创建的本实验项目中不仅有了 main.c，还已经自动加载了 USB-CDC 库函数（stc_usb_cdc_32g.lib）。main.c 中按模块化设计，已经包含了：必要的

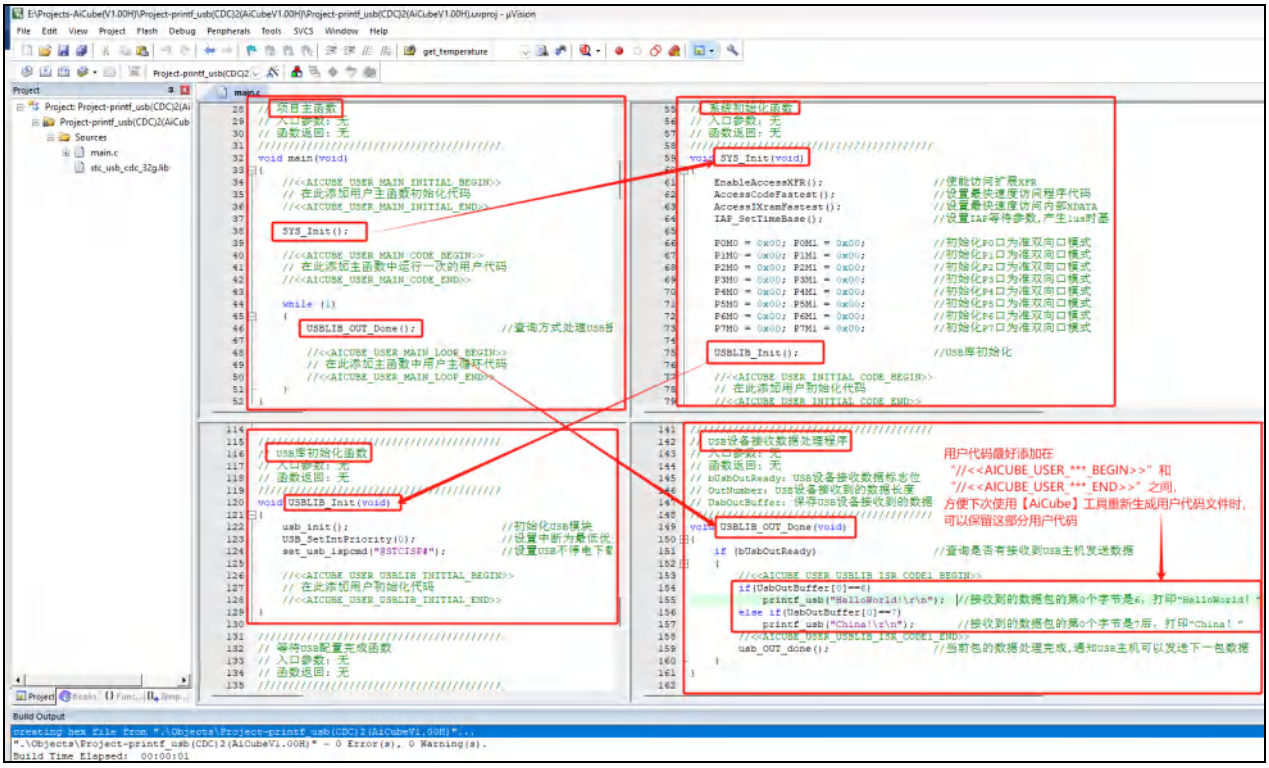
头文件、主函数、系统初始化函数、微秒延时函数、毫秒延时函数、USB 相关的配置及处理函数。

d) 增加实验项目代码

为了实现单片机通过 USB 协议接收 USB 主机（电脑）发送的命令后，再根据不同的命令回传不同的字符串到主机。我们现在只需要在“USB 设备接收数据处理程序”子函数的“查询是否有接收到 USB 主机发送数据”的 if 判断中加入如下代码即可：

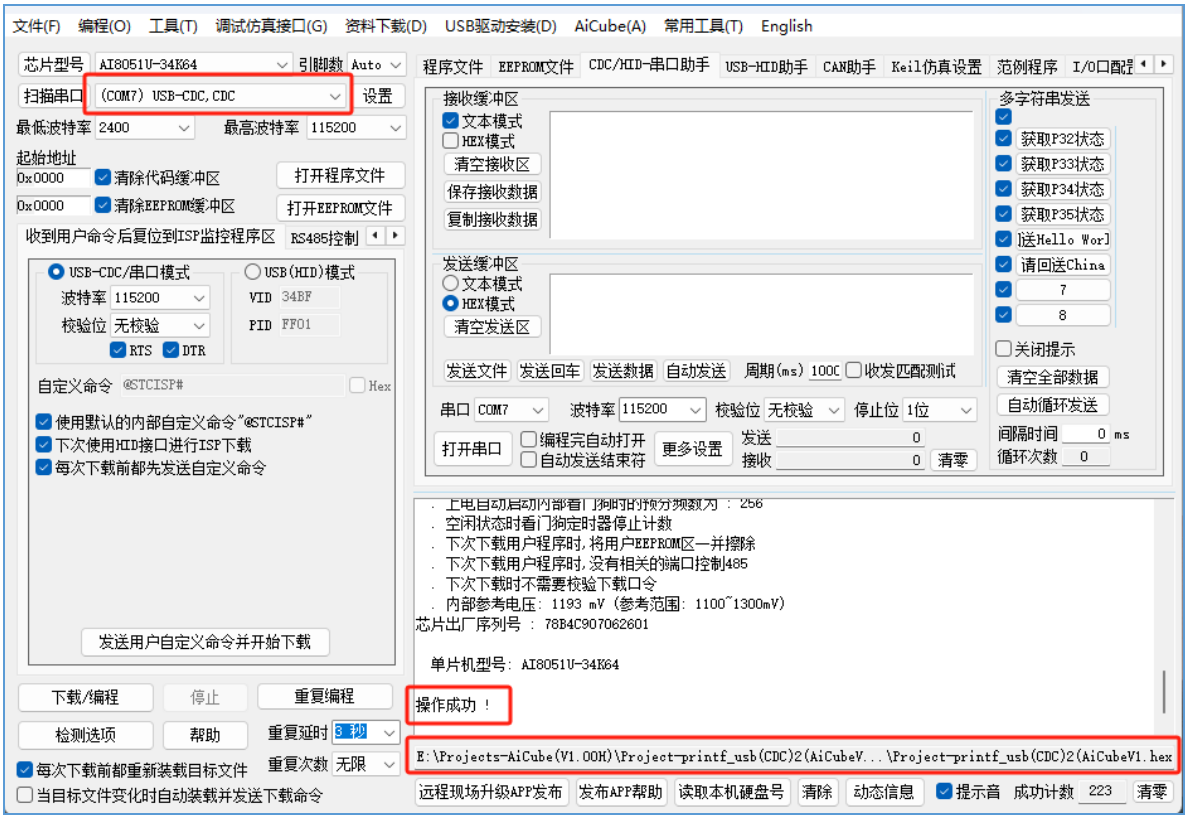
```
if(UsbOutBuffer[0]==6)
    printf_usb("HelloWorld!\r\n");      //接收到的数据包的第 0 个字节是 6，打印“HelloWorld！”
else if(UsbOutBuffer[0]==7)
    printf_usb("China!\r\n");           //接收到的数据包的第 0 个字节是 7 后，打印“China！”
```

保存，编译，如下图（为方便查看，使用菜单“Window”→“Split”将原窗口分裂出 4 个窗口，对比上图观察）：

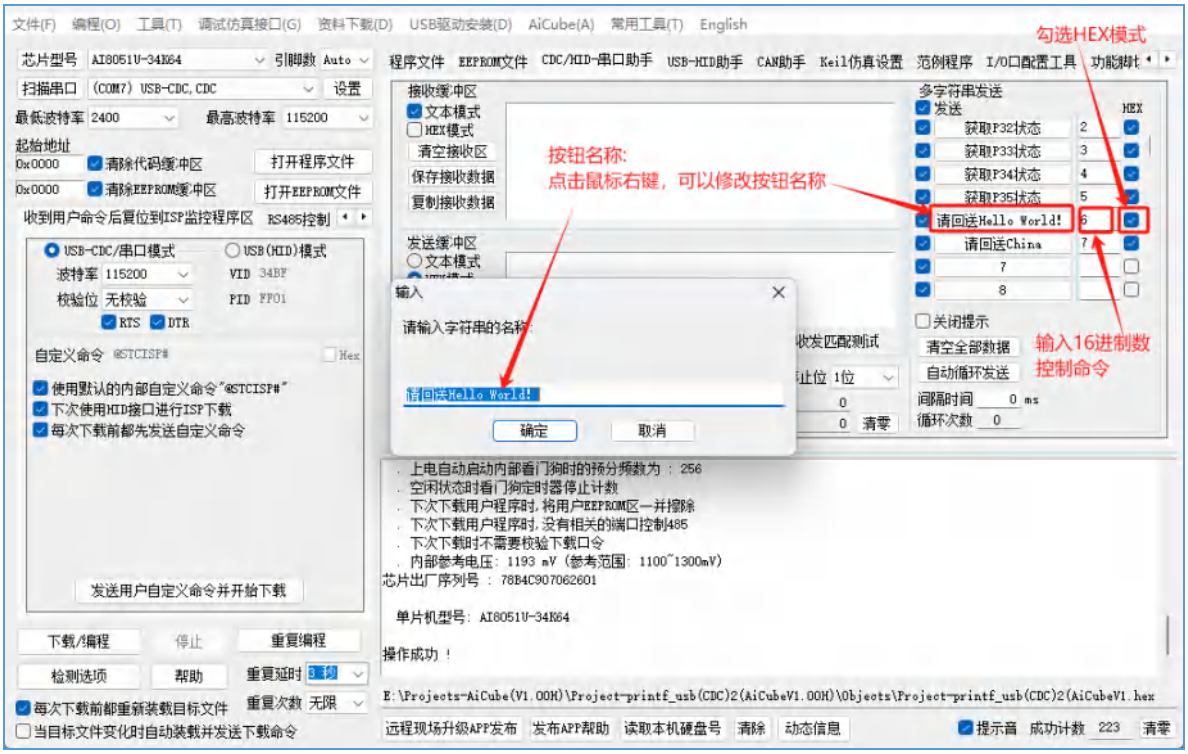


e) 使用 AIapp-ISP 软件系统和 Ai8051U 实验箱查验实验效果

连接好 Ai8051U 实验箱，运行 AIapp-ISP 软件系统，按步骤打开并下载程序“Project-printf_usb(CDC)2(AiCubeV1.00H).hex”，如下图：

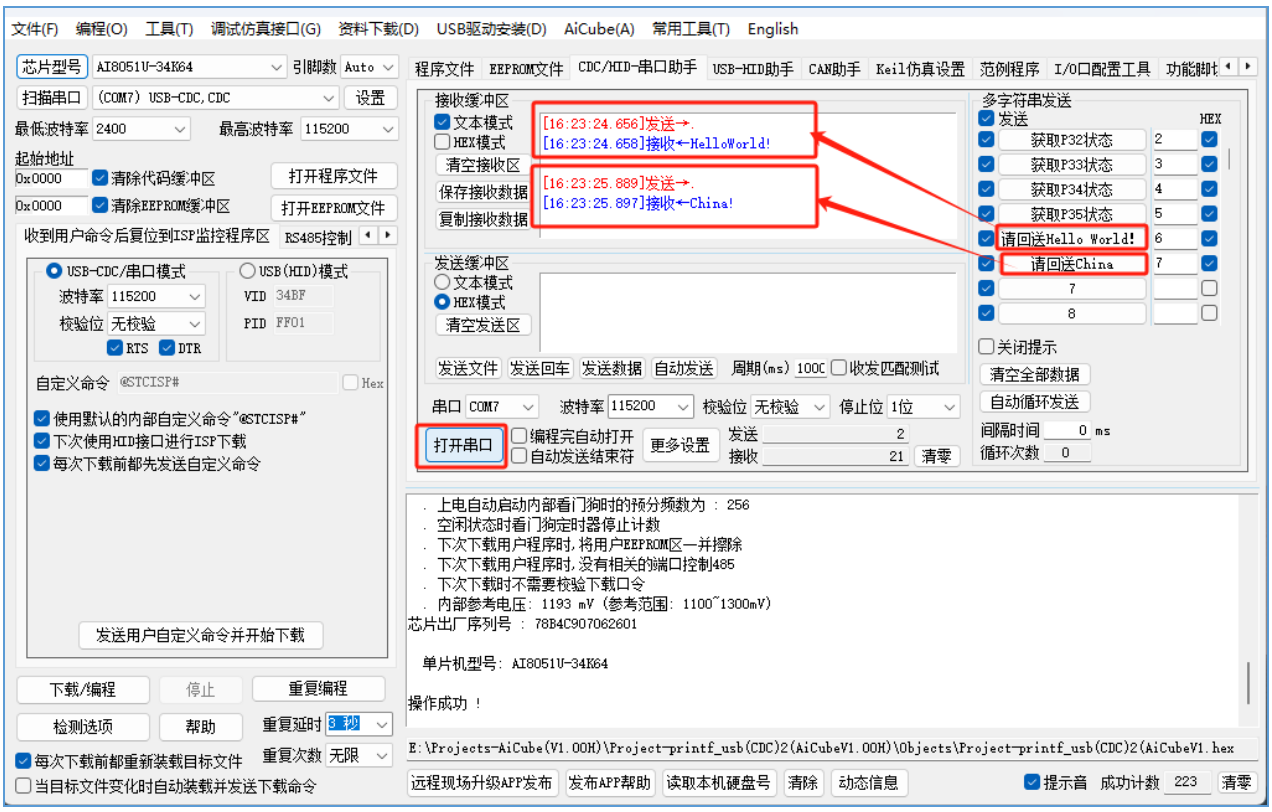


在 AIapp-ISP 系统软件，我们看到：此时串口显示“（COM7）USB-CDC，CDC”。在“CDC/HID-串口助手”选项卡的“多字符串发送”窗口，设置好发送控制命令，以便观察实验效果，如下图：



在 CDC/HID-串口助手选项卡，找到并点击“打开串口”按钮，一切准备就绪！

点击“请回送 Hello World！”按钮，单片机收到 16 进制数 6 后，回送“Hello World！”给 USB 主机显示。
点击“请回送 China！”按钮，单片机收到 16 进制数 7 后，回送“China！”给 USB 主机显示。
在 CDC/HID-串口助手的“接收缓冲区”，如下图：



完成实验验证！

有了【AiCube】助手，单片机编程就是这么简单！快用起来吧！

2.13.6　USB 不停电下载，ISP 软件如何设置

在开发过程中，程序经常需要反复修改，重新编译后生成新的 HEX 文件，就需要停电/上电进入下载模式，很不方便。其实可以利用我们提供的 USB 不停电 ISP 下载功能，进行自动下载。

在有 USB 功能的实验中，都要加载 USB 库函数，并对 USB 进行初始化。由于在 USB 库函数的代码中已经定义了不停电 ISP 下载命令"@STCISP#"，只需要在下载软件的"收到用户命令后复位到 ISP 监控程序区"的选项卡中进行如下设置，即可实现【USB 不停电下载】功能。

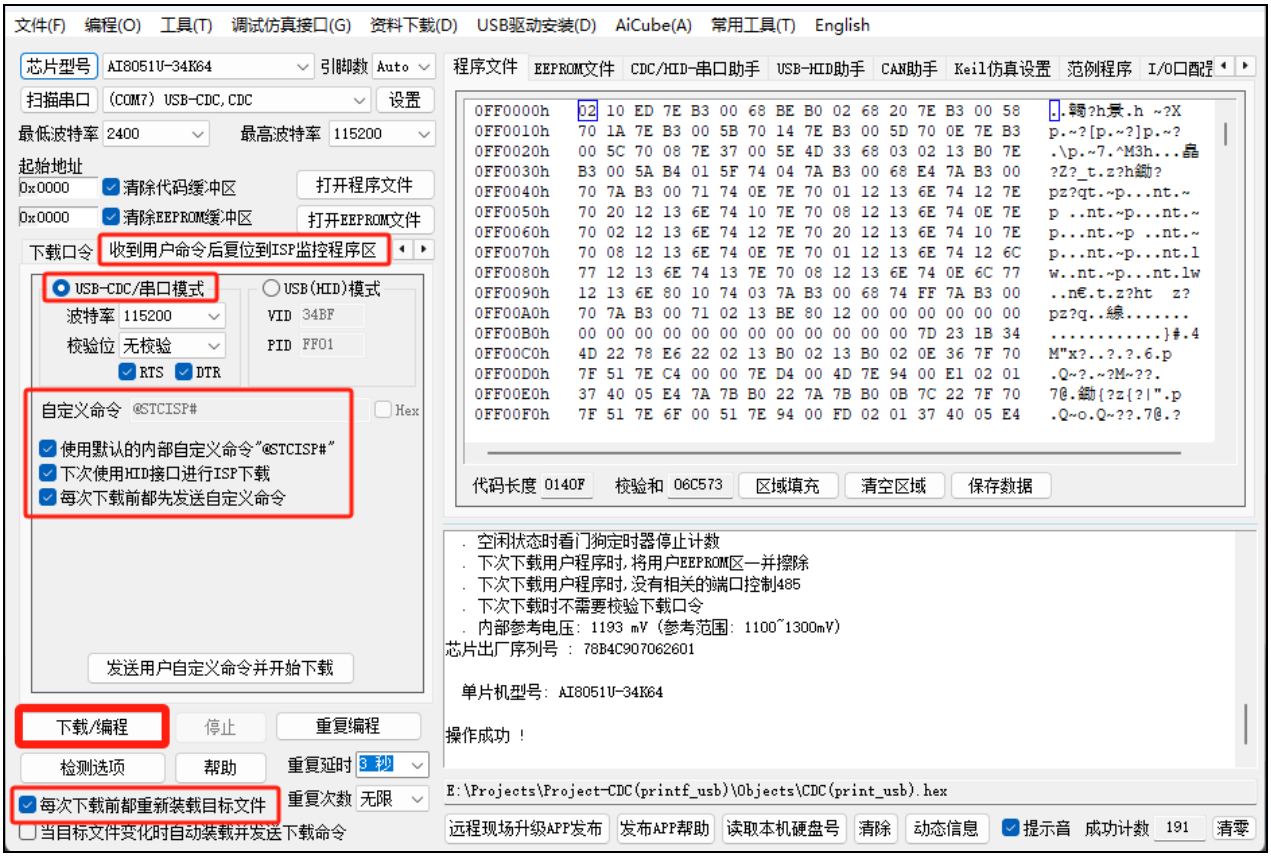
- 如使用 USB-CDC 库函数产生 USB-CDC 虚拟串口，可实现 USB-CDC 不停电自动 ISP 下载功能，并可忽略传统串口的波特率、校验位等参数的设置，下载时系统程序区还是使用的 USB-HID 通讯协议；
- 如果是传统的串口，需正确设置波特率、校验位等参数，才能实现串口的不停电自动 ISP 下载功能；
- 如果使用 USB-HID 库函数产生 USB-HID 设备，则选择"USB（HID）模式"就能实现不停电自动 ISP 下载功能，下载时系统程序区还是使用的 USB-HID 通讯协议。

ISP 软件如何设置：

2.13.6.1　手动点击【下载/编程】按钮，ISP 软件自动重新打开文件，不停电下载

如下图设置，就可以实现：

在点击"下载/编程"按钮后，ISP 软件会重新调入已打开的目标文件一次（不管目标文件有无变化），接着通过 USB-CDC 通讯协议自动发送自定义下载命令如"@STCISP#"给单片机，单片机中的用户软件收到该命令后，就会软复位到单片机的系统监控程序区，等待 ISP 下载，本设置还指定了电脑端软件后续是通过 USB-HID 通讯协议传输用户程序到目标单片机进行 ISP 下载。

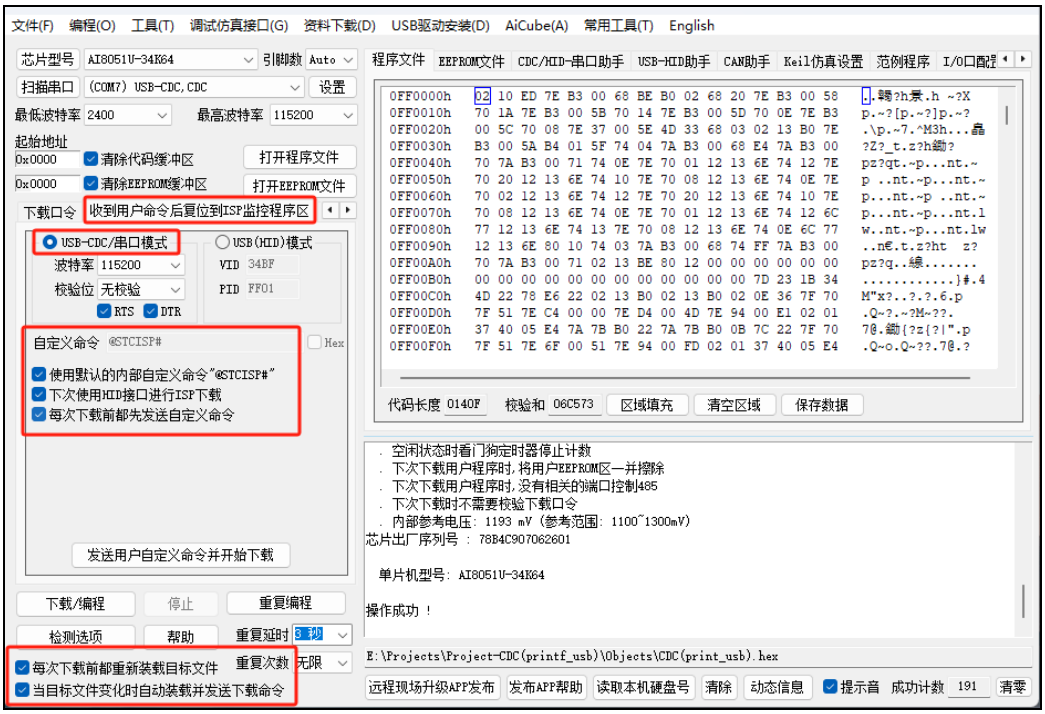


如上所示，不停电下载实验是成功的，但本设置还需要手工点击一次"下载/编程"按钮。

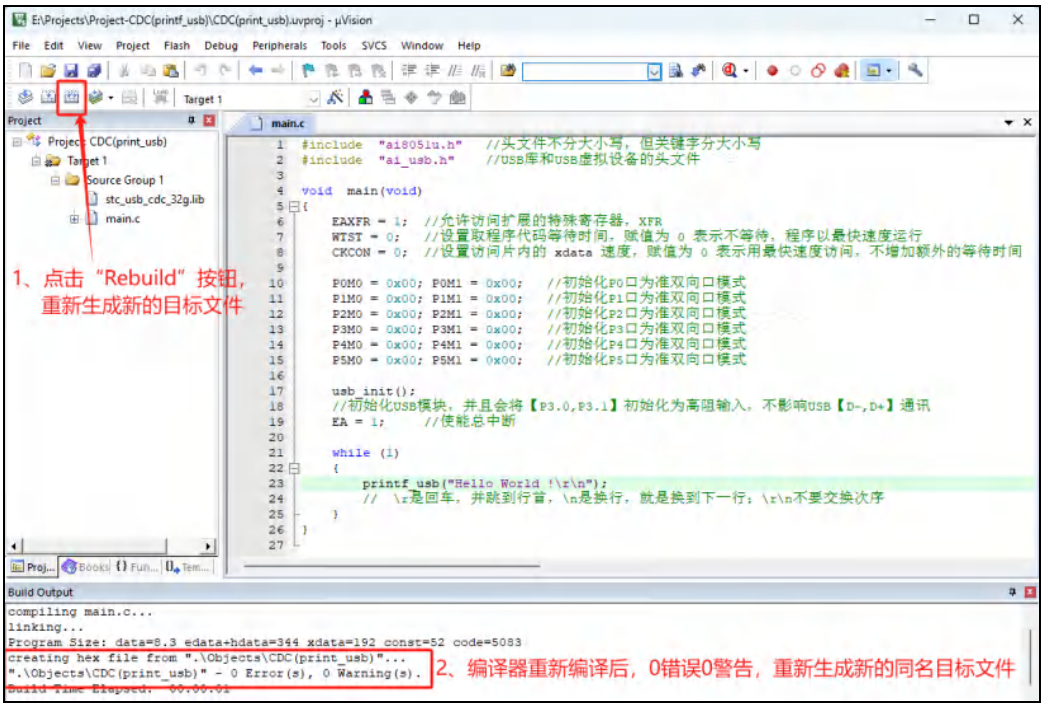
2.13.6.2 编译器重新编译后，ISP 软件会自动重新调入文件，进行不停电下载

如下图设置：

下载软件会每秒检测多次目标文件被编译的时间是否已更新，如时间已更新，则 ISP 软件会自动重新调入已打开的目标文件，接着通过 USB-CDC 通讯协议自动发送自定义下载命令如"@STCISP#"给单片机，单片机中的用户软件收到该命令后，就会软复位到单片机的系统监控程序区，等待 ISP 下载，本设置还指定了电脑端软件后续是通过 USB-HID 通讯协议传输用户程序到目标单片机进行 ISP 下载。



如上所示，不停电下载实验是成功的，但本设置的优点是不需要手工点击"下载/编程"按钮。



在编译集成环境中，点击"Rebuild"按钮，目标文件会被重新编译生成一次，ISP 软件发现目标文件被编译的时间发生变化后，ISP 软件就会执行前面描述的不停电下载动作。

2.14　实验三：中断方式，单片机 USB 中断接收服务函数收到命令后，printf_usb("Hello World !\r\n")或其他

看下面的程序，其实还是那个熟悉的标准 C 语言的 printf()，只是到了单片机 C 语言中，变成了 printf_usb()

2.14.1　printf( )，printf_usb( )实验程序代码

```
#include   "ai8051u.h"   //头文件不分大小写，但关键字分大小写
#include   "ai_usb.h"    //USB 库和 USB 虚拟设备的头文件，需要在项目文件中添加对应的 USB 库
//由于在 USB 库函数的代码中已经定义了，不停电 ISP 下载命令字"@STCISP#"，
//只需要在下载软件的“收到用户命令后复位到 ISP 监控程序区”选项卡中进行相应的设置，
//即可实现 USB 不停电 ISP 下载功能，或串口不停电 ISP 下载功能。

// USB 接收中断服务程序中，调用用户实际需要执行的服务程序，
//  简称回调函数，用户根据实际应用场景需求编写
void USBLIB_OUT_Callback(void)
{    //用户希望 USB 库函数，中断方式回调执行的用户自己编写的任务函数，简称回调函数
     if (UsbOutBuffer[0] == 6)
          printf_usb("Hello World !\r\n");
          //接收到数据 6 后，直接打印输出“Hello World !”到主机相应软件的 USB【CDC/HID-串口助手】
          //   \r 是回车，跳到行首，\n 是换行，就是换到下一行；\r\n 不要交换次序
     else if (UsbOutBuffer[0] == 7)
          printf_usb("China !\r\n");
          //接收到数据 7 后，直接打印输出 “China !” 到主机相应软件的 USB【CDC/HID-串口助手】”
}

void main(void)
{
     EAXFR = 1;   //允许访问扩展的特殊寄存器，XFR
     WTST = 0;    //设置取程序代码等待时间，赋值为 0 表示不等待，程序以最快速度运行
     CKCON = 0;  //设置访问片内的 xdata 速度，赋值为 0 表示用最快速度访问，不增加额外的等待时间

     usb_init();                          //初始化 USB 模块
     //初始化 USB 模块，并且会将【P3.0, P3.1】初始化为高阻输入，不影响 USB【D-, D+】通讯

     set_usb_OUT_callback(USBLIB_OUT_Callback);  //设置 USB 中断服务程序可以调用的回调函数
     //此处 USBLIB_OUT_Callback 是前面用户自己编写的任务函数的函数名/地址，供 USB 库函数回调

     EA = 1;                              //总中断允许位打开

     while (1);
}
```

以上是完整的第一个程序代码

printf_usb()函数：

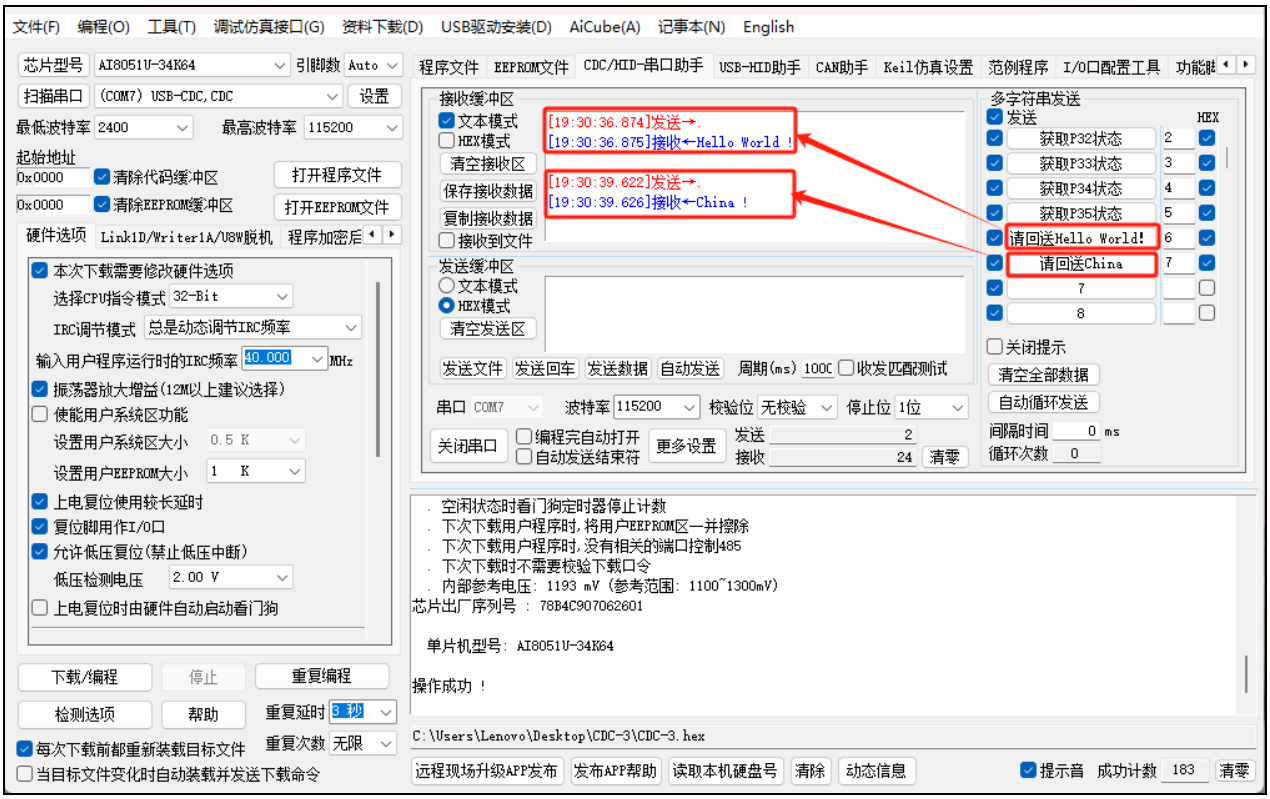
printf_usb()函数和标准 C 语言的格式完全一致，详见 printf()函数章节

进入 printf_usb()函数后，首先会判断 USB 主机是否已经对 USB 设备配置完成，

如果没有配置完成，printf_usb()函数会直接返回退出。
如果已经配置完成，还会继续判断如有发送前一笔数据，则判断是否发送完成，
如果没有发送完成，会继续等待，直到前一笔数据发送完成，
才会发送新的 printf_usb()需发送的数据后再返回退出。
但如等待超过 50ms@24MHz，会直接退出。
如果主时钟不是 24MHz，20MHz 等待时间会变长，40MHz 等待时间会变短。
如果已经配置完成且前一笔数据已经发送完成，
则会直接发送新的 printf_usb()需发送的数据并返回退出。

2.14.2　查看实验效果

按 printf_usb()实验二的步骤流程建立项目，编译通过，并将目标文件下载到 Ai8051U 实验箱。
运行效果如下图：



点击“请回送 Hello World！”按钮，单片机收到 16 进制数 6 后，回送“Hello World！”给 USB 主机显示。
点击“请回送 China！”按钮，单片机收到 16 进制数 7 后，回送“China！”给 USB 主机显示。

2.14.3　使用【AiCube】工具创建“中断方式，单片机 USB 中断接收服务函数收到命令后，printf_usb("Hello World !\r\n")或其他”

STCAI 在最新版的 AIapp-ISP 软件中，推出【AiCube】项目创建工具，目前支持 Ai8051U-32Bit 各功能模块的智能化配置，并生成程序框架。这项功能在程序员建立项目时，简化了项目的初始化难度，提高效率，避免因不太熟悉或不完全了解该单片机的性能而忽略个别配置，引起编译错误等问题。尤其对刚涉足单片机的初学者，此工具不仅引导其了解单片机编程流程，还极大的降低了入门门槛，缩短了摸索的时间，提高入门的兴趣和信心。现在我们使用这个工具来生成实现此实验功能的配置和代码。

a) 安装 AIapp-ISP 下载/编程/烧录/软件

安装 STCAI 公司研发的 AIapp-ISP 下载 / 编程 / 烧录 / 软件，含强大的辅助开发工具包。登录 STCAI 官网，下载最新版的 AIapp-ISP 安装包，下载链接如下：https://www.stcai.com/gjrj

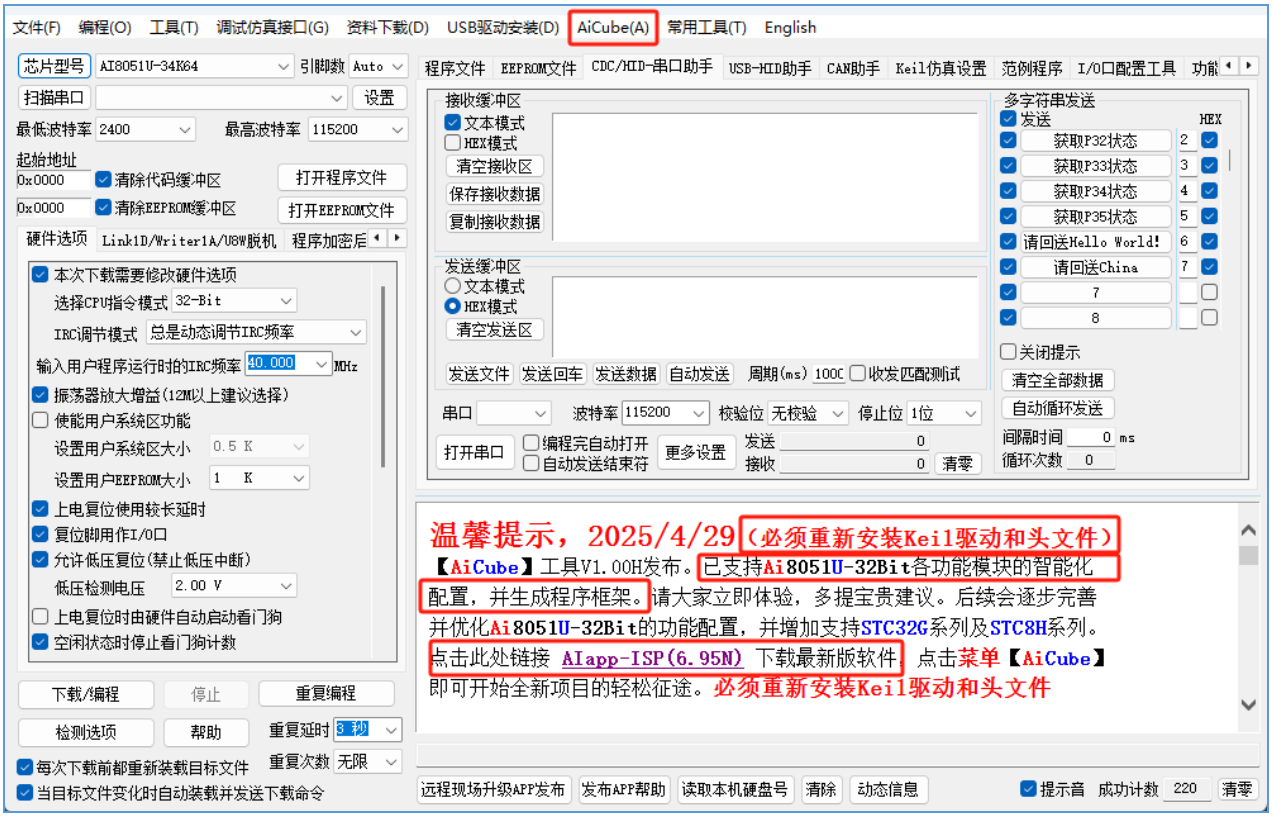
下载下来的是压缩包文件如下图：

双击后解压缩如下图：

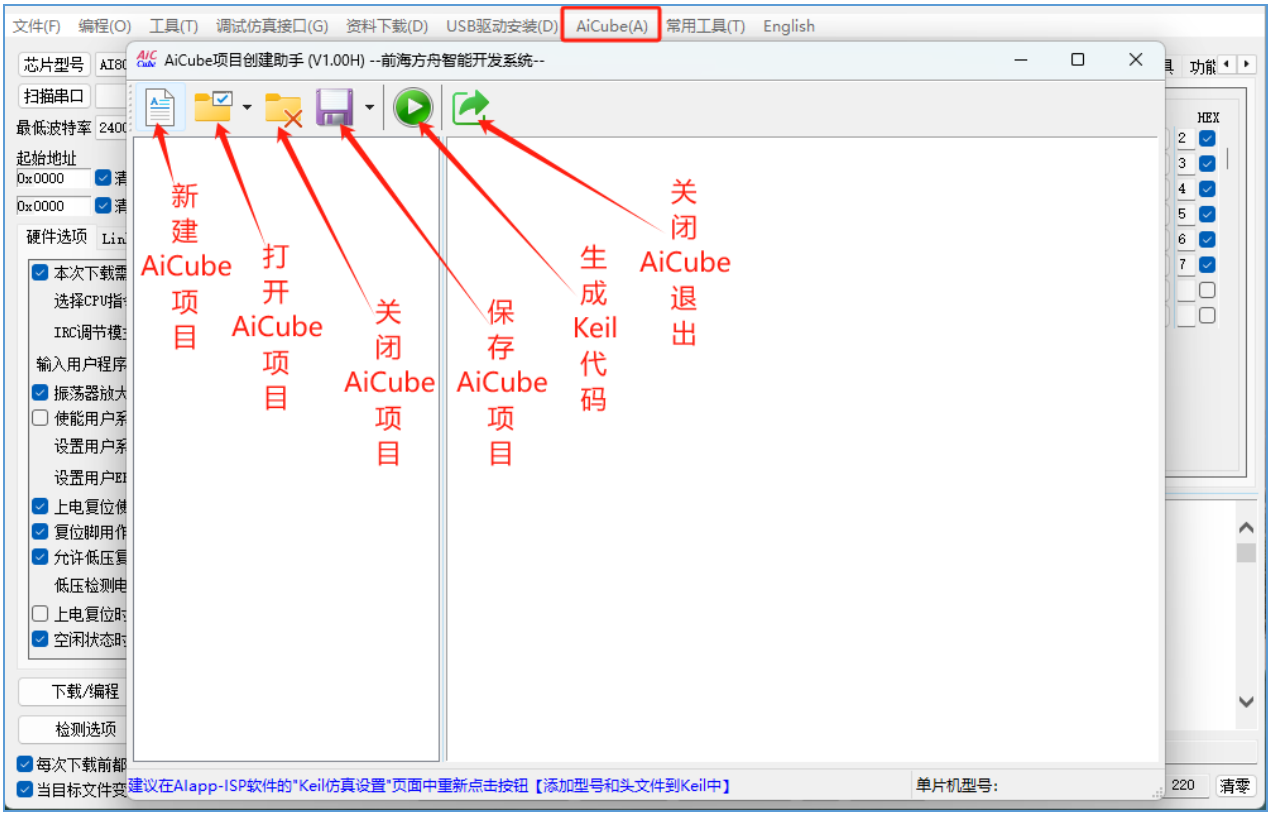
如果是解压缩到非桌面的其他目录，如 D:\STCAI-ISP，建议创建快捷方式到桌面方便打开。这实际是一个免安装的绿色软件。

b) 打开【AiCube】助手

双击图标，打开 AIapp-ISP 软件界面，即可直接使用，如下图，在菜单栏上找到【AiCube(A)】菜单：

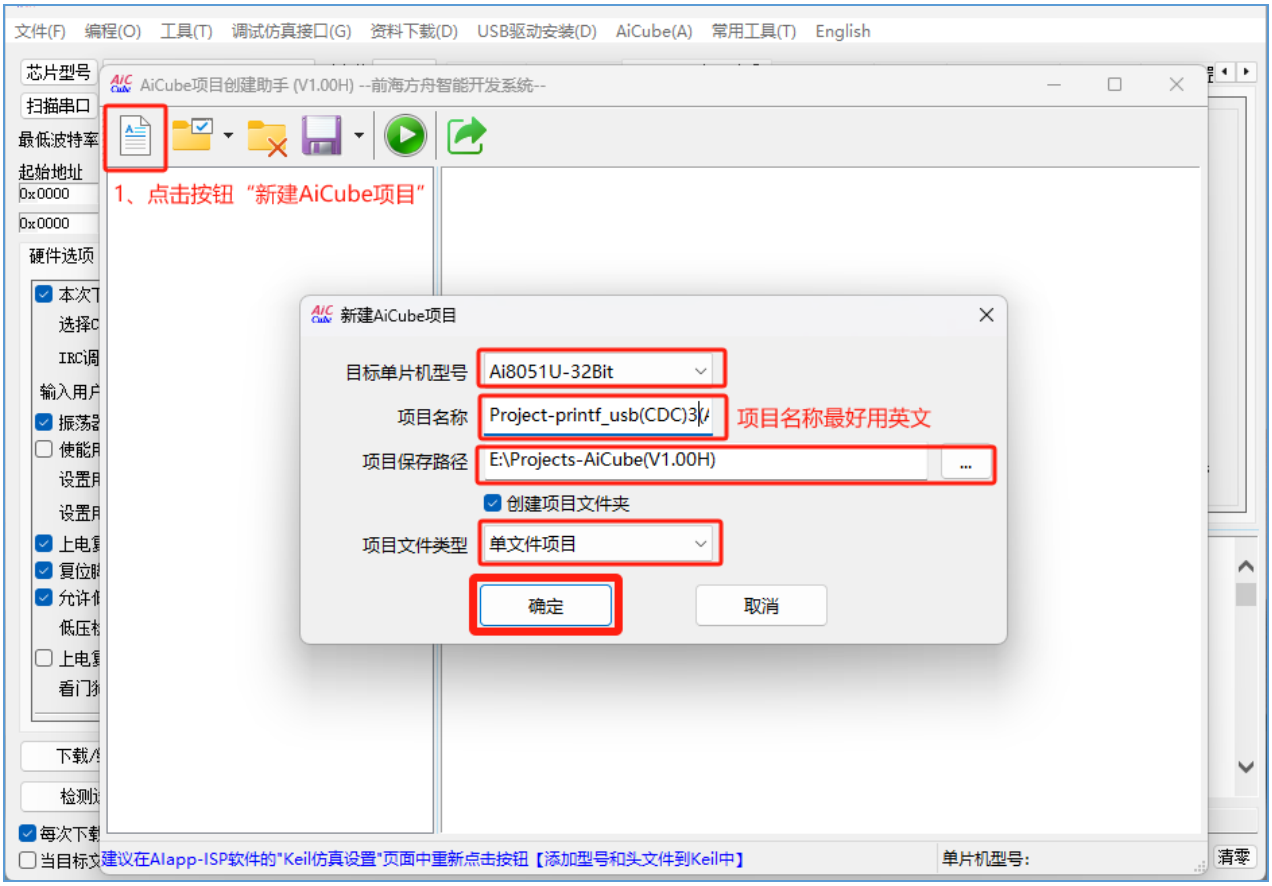


点击【AiCube(A)】，弹出如下窗口：

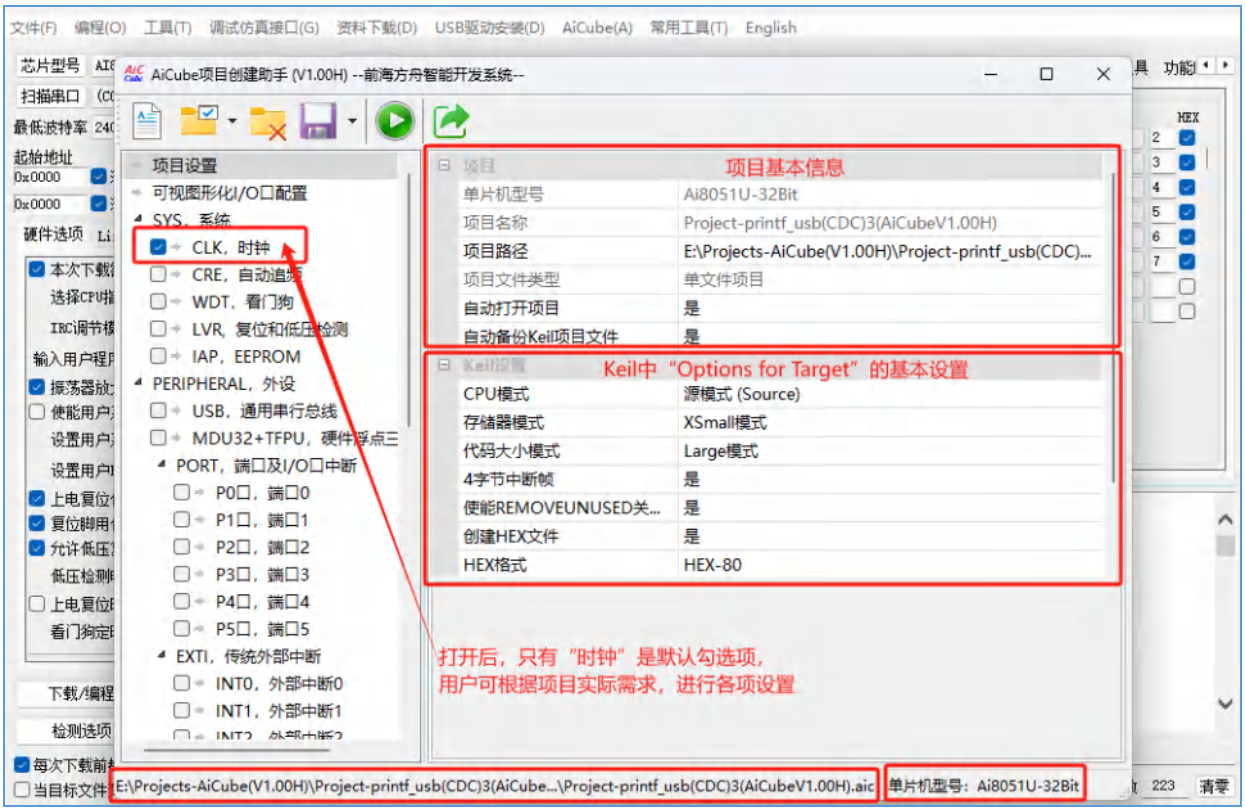


c) 使用【AiCube】创建项目

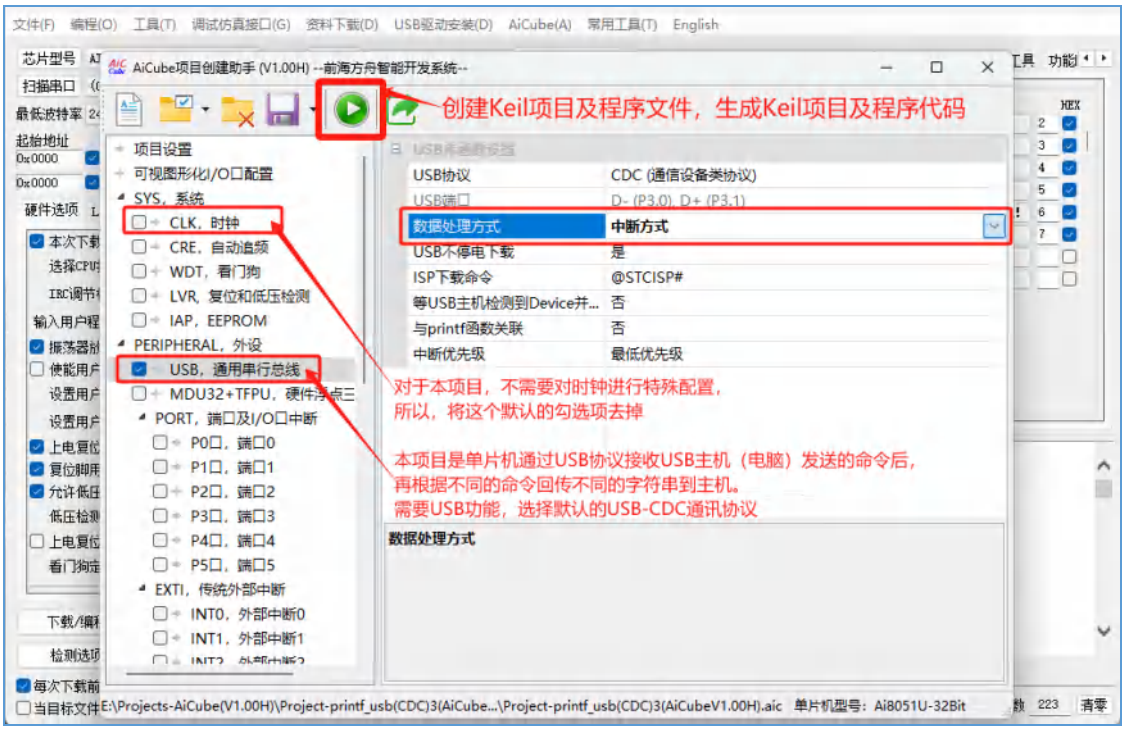
点击按钮“新建 AiCube 项目”，出现如下配置界面，选择并填写项目基本信息，建议项目名称用英文！



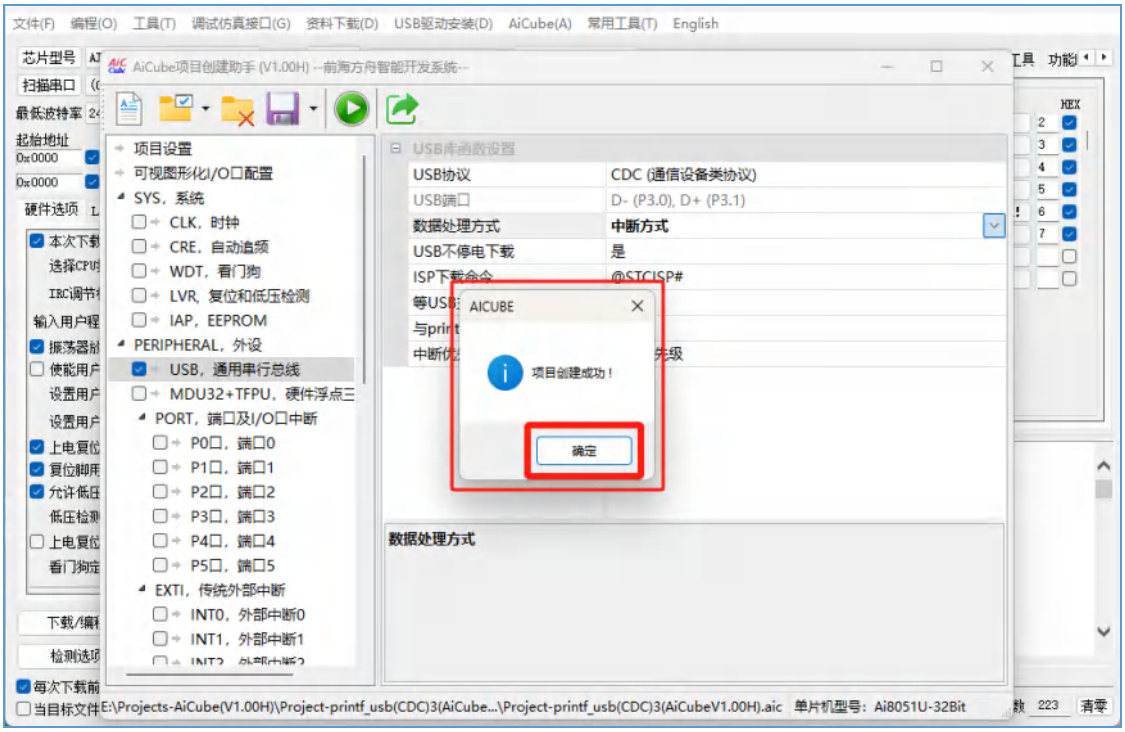
点击“确定”，出现如下界面：



因本项目用到 USB-CDC 功能，要加载 USB-CDC 库函数，对 USB 初始化等，使用【AiCube】助手根据本项目的实际需求，对配置进行如下修改：



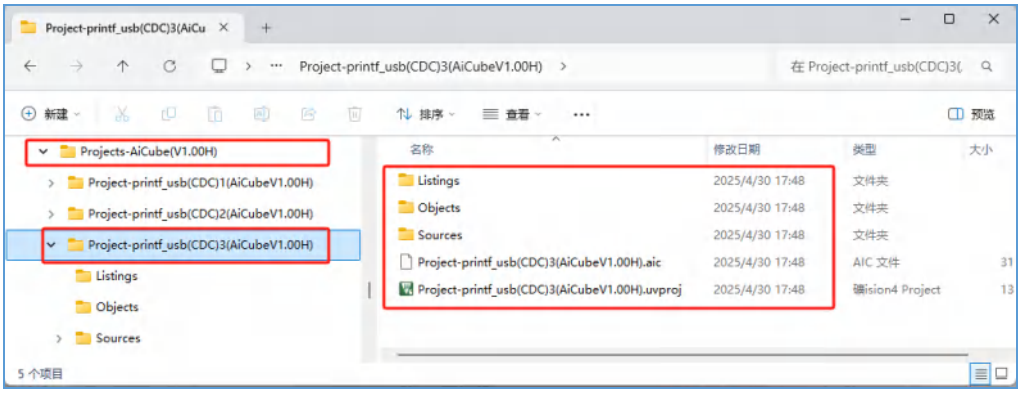
点击上图“创建 Keil 项目及程序文件”按钮，【AiCube】提示：“项目创建成功！”，如下图：



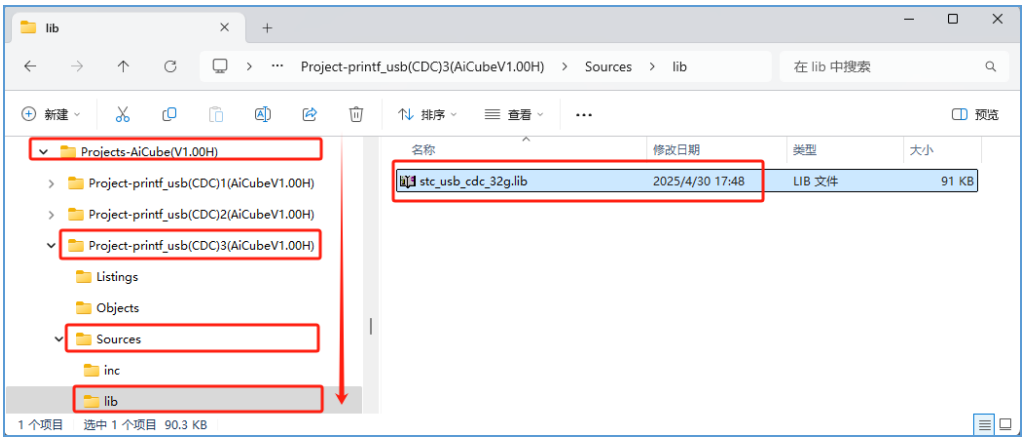
点击“确定”按钮后，系统会自动打开 Keil 软件，调入新创建的 Keil 工程项目，本实验创建的项目是“Project-printf_usb(CDC)3(AiCubeV1.00H)”。现在，可以关闭【AiCube】工具，当然，也可以不关闭。

➲ 找到并打开刚刚创建的项目目录

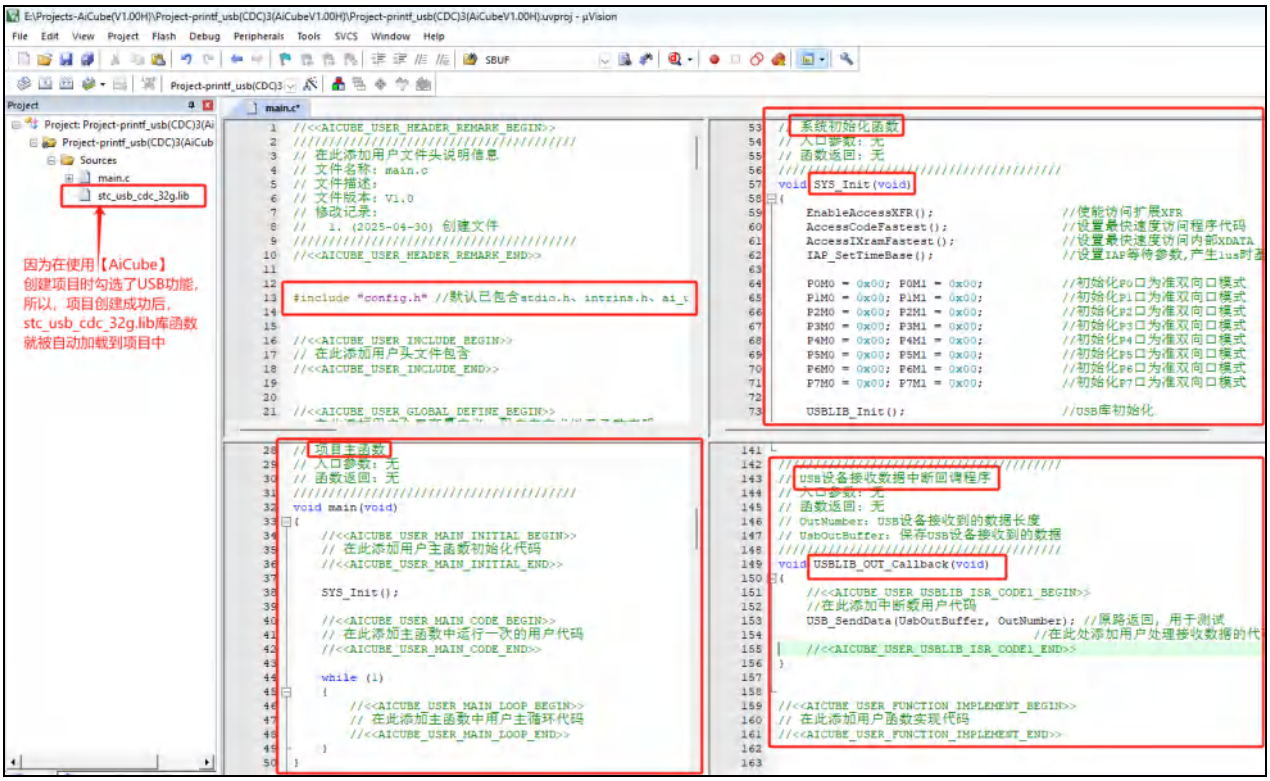
上述操作过程，【AiCube】工具已经帮我们建立了项目目录及 Project-printf_usb(CDC)3(AiCubeV1.00H).uvproj 文件，如下图：



再打开 Sources>lib 子目录，发现，【AiCube】将相应的 USB 库函数加载进项目目录：



➲ 观察已打开的 Keil 项目文件：Project-printf_usb(CDC)3(AiCubeV1.00H).uvproj，如下图：



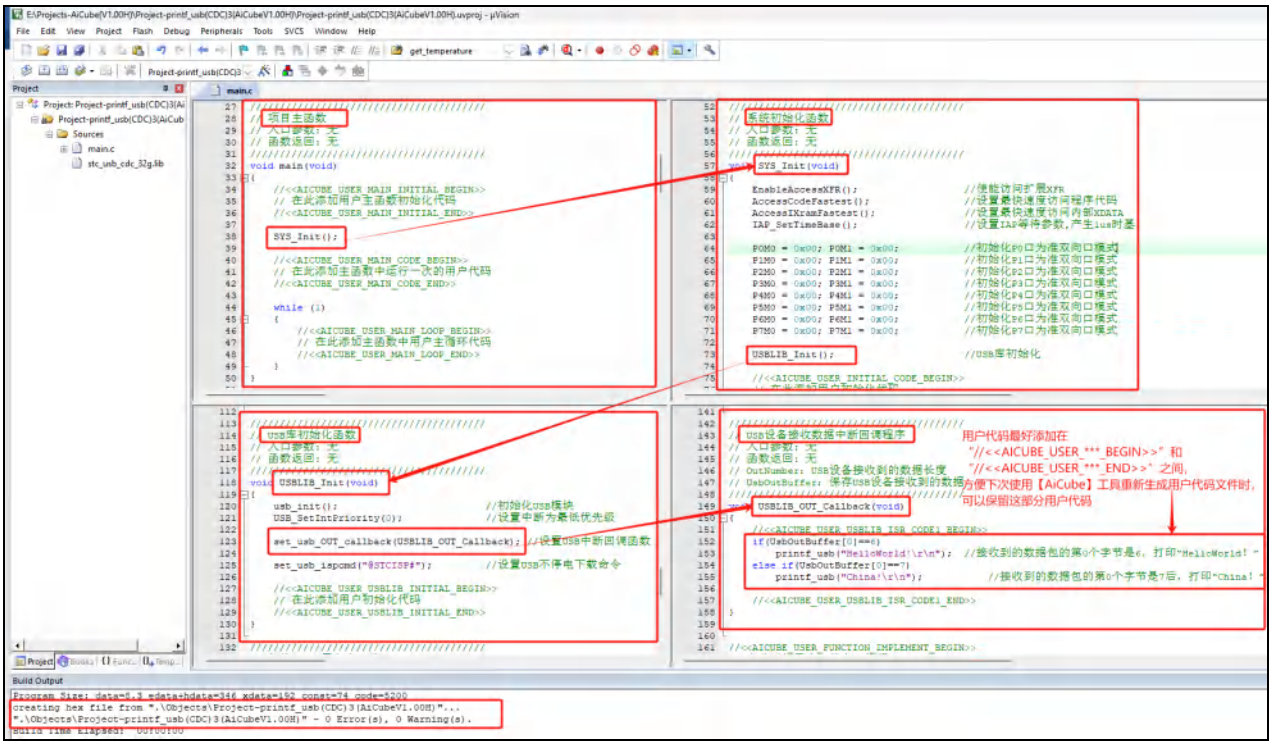
根据在【AiCube】中的配置，从上图中我们看到【AiCube】创建的本实验项目中不仅有了 main.c，还已经自动加载了 USB-CDC 库函数（stc_usb_cdc_32g.lib）。main.c 中按模块化设计，已经包含了：必要的头文件、主函数、系统初始化函数、微秒延时函数、毫秒延时函数、USB 相关的配置及处理函数。

d) 增加实验项目代码

为了实现单片机通过 USB 协议接收 USB 主机（电脑）发送的命令后，再根据不同的命令回传不同的字符串到主机。我们现在只需要在“USB 设备接收数据中断回调程序”子函数中加入如下代码即可：

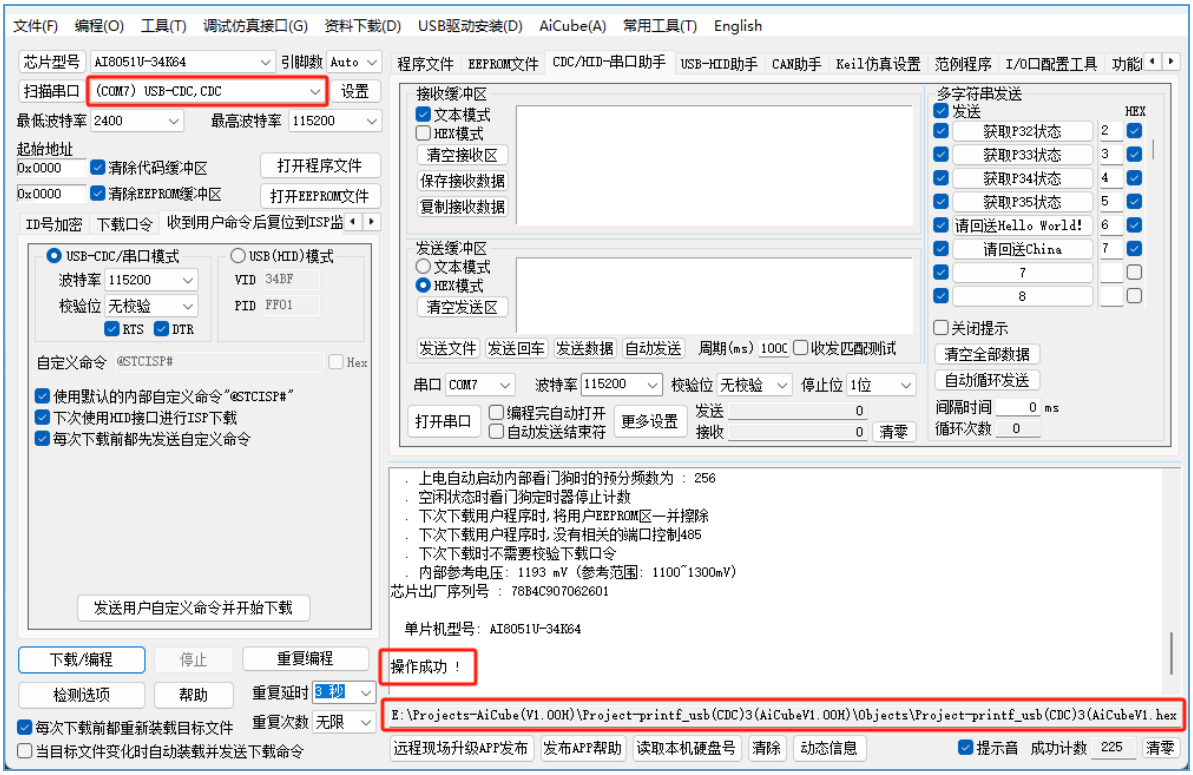
```
if(UsbOutBuffer[0]==6)
    printf_usb("HelloWorld!\r\n");      //接收到的数据包的第 0 个字节是 6，打印“HelloWorld！”
else if(UsbOutBuffer[0]==7)
    printf_usb("China!\r\n");           //接收到的数据包的第 0 个字节是 7 后，打印“China！”
```

保存，编译，如下图（为方便查看，使用菜单“Window”→“Split”将原窗口分裂出 4 个窗口，对比上图观察）：

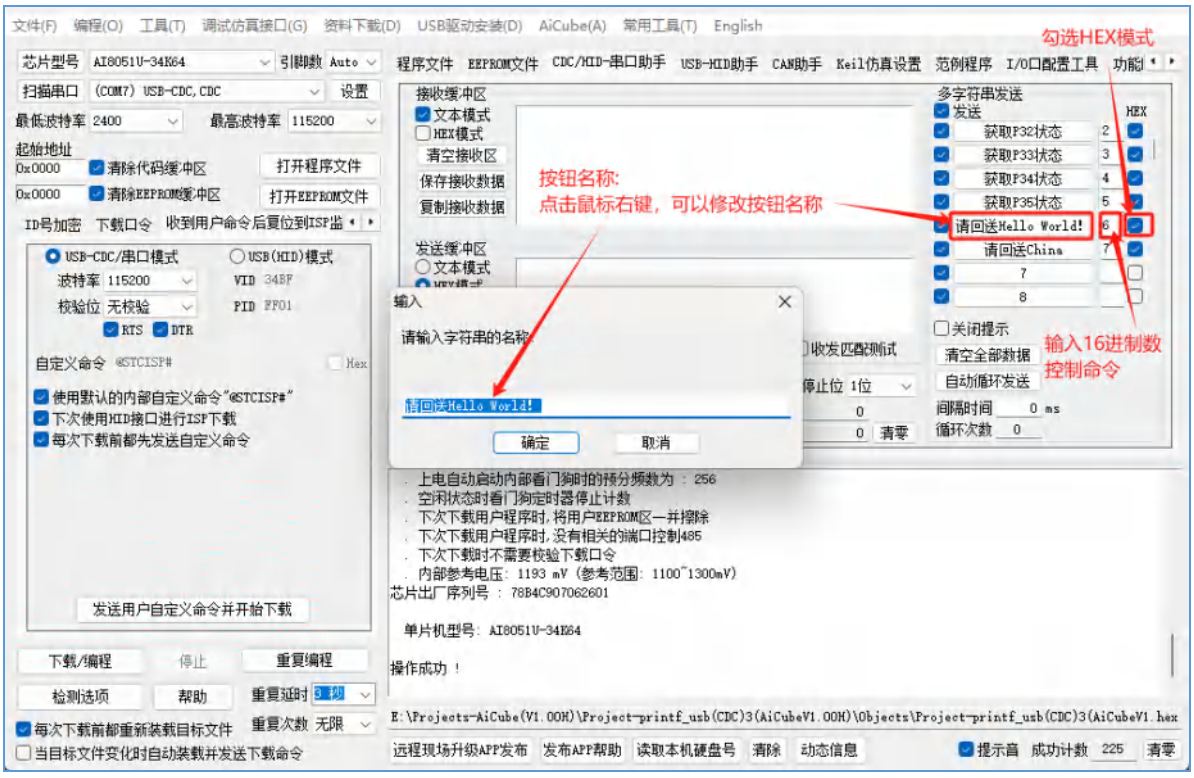


e) 使用 AIapp-ISP 软件系统和 Ai8051U 实验箱查验实验效果

连接好 Ai8051U 实验箱，运行 AIapp-ISP 软件系统，按步骤打开并下载程序“Project-printf_usb(CDC)3(AiCubeV1.00H).hex”，如下图：

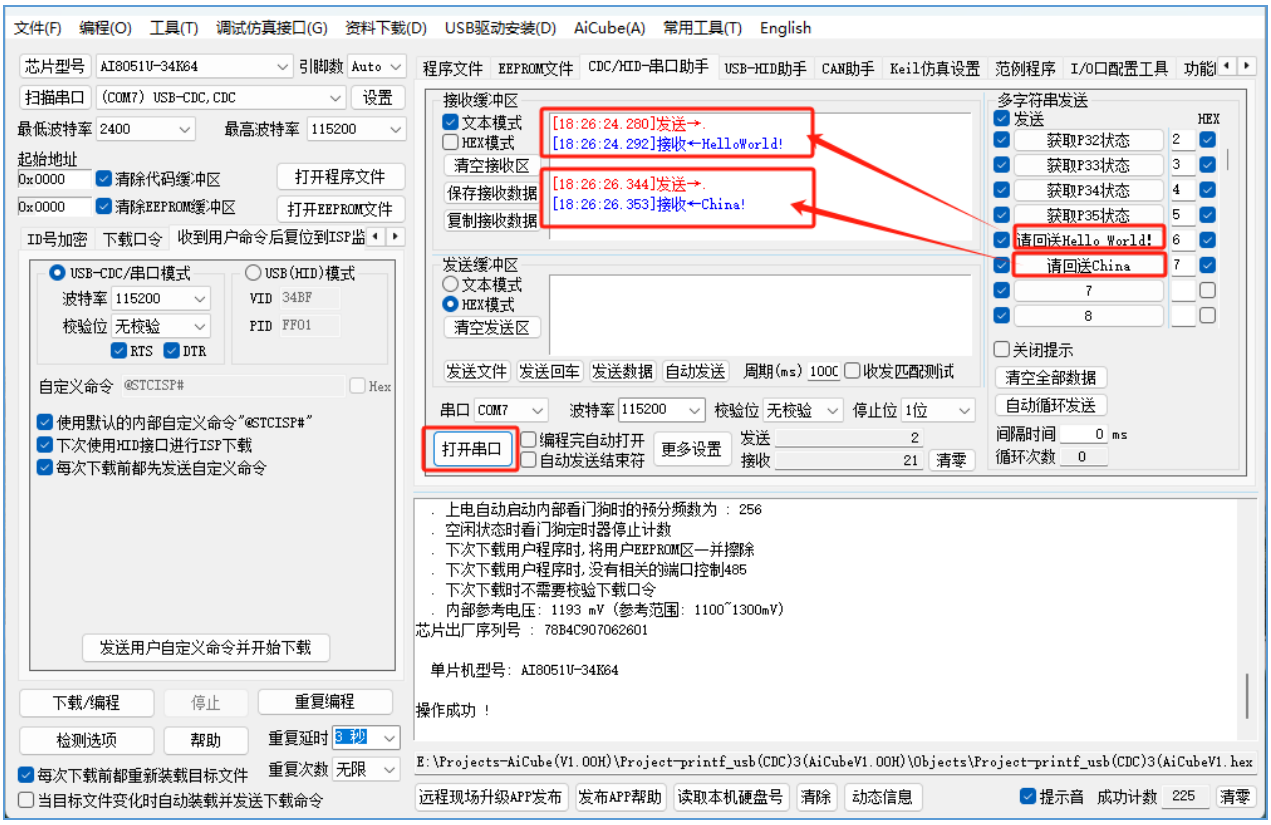


在 AIapp-ISP 系统软件，我们看到：此时串口显示“（COM7）USB-CDC，CDC”。在“CDC/HID-串口助手”选项卡的“多字符串发送”窗口，设置好发送控制命令，以便观察实验效果，如下图：



在 CDC/HID-串口助手选项卡，找到并点击“打开串口”按钮，一切准备就绪！

点击“请回送 Hello World！”按钮，单片机收到 16 进制数 6 后，回送“Hello World！”给 USB 主机显示。
点击“请回送 China！”按钮，单片机收到 16 进制数 7 后，回送“China！”给 USB 主机显示。
在 CDC/HID-串口助手的“接收缓冲区”，如下图：



完成实验验证！

有了【AiCube】助手，单片机编程就是这么简单！快用起来吧！

2.14.4　USB 不停电下载，ISP 软件如何设置

在开发过程中，程序经常需要反复修改，重新编译后生成新的 HEX 文件，就需要停电/上电进入下载模式，很不方便。其实可以利用我们提供的 USB 不停电 ISP 下载功能，进行自动下载。

在有 USB 功能的实验中，都要加载 USB 库函数，并对 USB 进行初始化。由于在 USB 库函数的代码中已经定义了不停电 ISP 下载命令"@STCISP#"，只需要在下载软件的"收到用户命令后复位到 ISP 监控程序区"的选项卡中进行如下设置，即可实现【USB 不停电下载】功能。

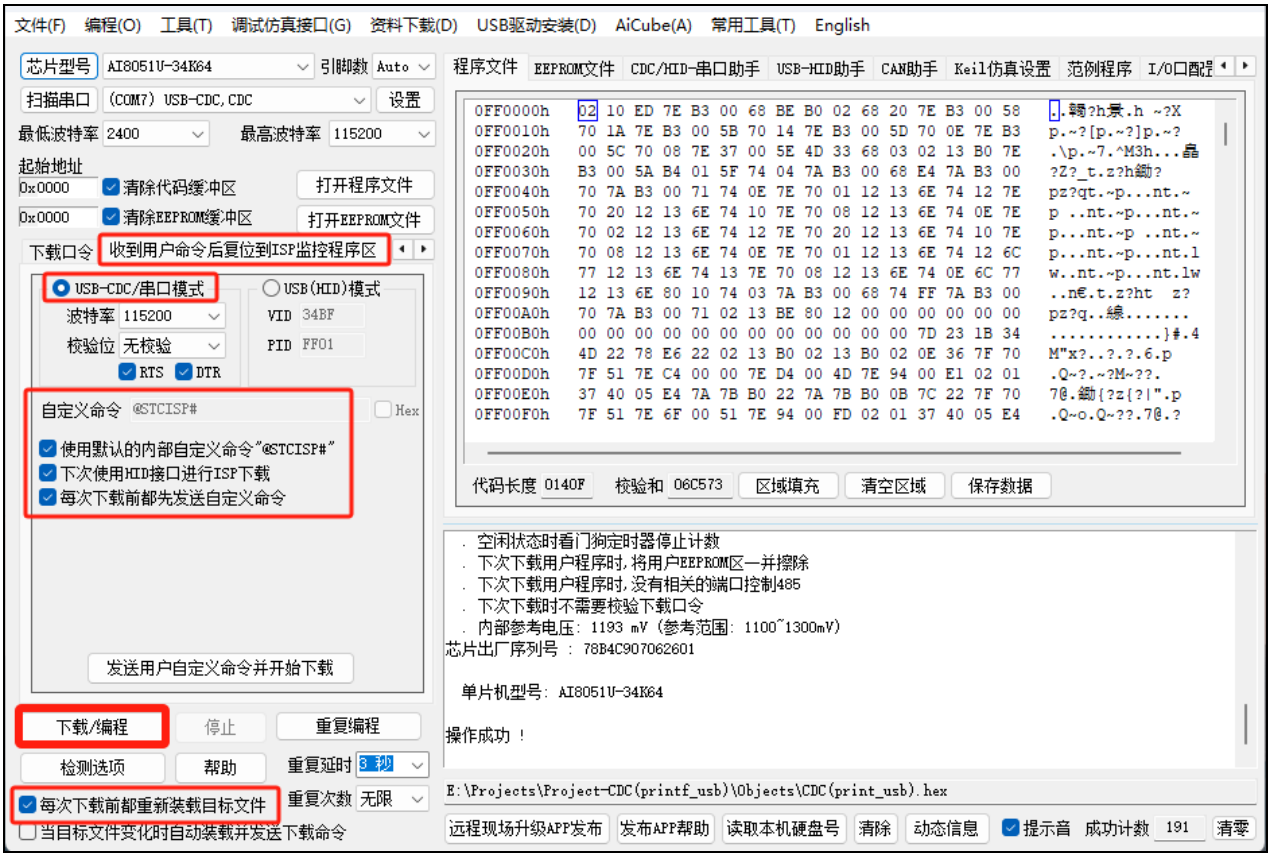
- 如使用 USB-CDC 库函数产生 USB-CDC 虚拟串口，可实现 USB-CDC 不停电自动 ISP 下载功能，并可忽略传统串口的波特率、校验位等参数的设置，下载时系统程序区还是使用的 USB-HID 通讯协议；
- 如果是传统的串口，需正确设置波特率、校验位等参数，才能实现串口的不停电自动 ISP 下载功能；
- 如果使用 USB-HID 库函数产生 USB-HID 设备，则选择"USB（HID）模式"就能实现不停电自动 ISP 下载功能，下载时系统程序区还是使用的 USB-HID 通讯协议。

ISP 软件如何设置：

2.14.4.1　手动点击【下载/编程】按钮，ISP 软件自动重新打开文件，不停电下载

如下图设置，就可以实现：

在点击"下载/编程"按钮后，ISP 软件会重新调入已打开的目标文件一次（不管目标文件有无变化），接着通过 USB-CDC 通讯协议自动发送自定义下载命令如"@STCISP#"给单片机，单片机中的用户软件收到该命令后，就会软复位到单片机的系统监控程序区，等待 ISP 下载，本设置还指定了电脑端软件后续是通过 USB-HID 通讯协议传输用户程序到目标单片机进行 ISP 下载。

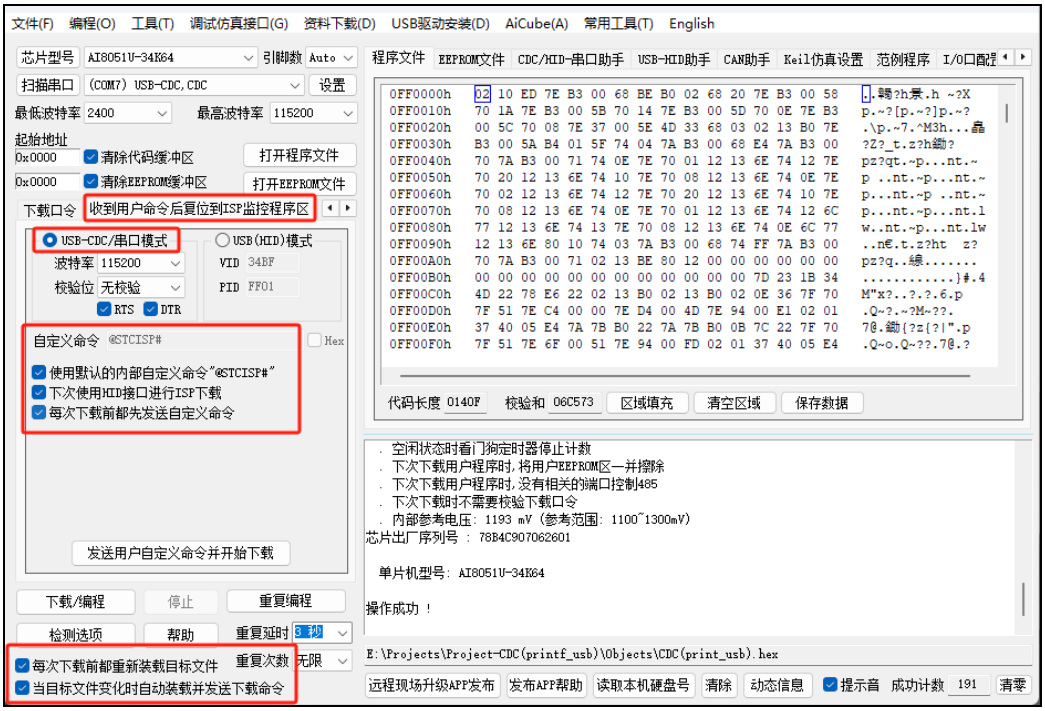


如上所示，不停电下载实验是成功的，但本设置还需要手工点击一次"下载/编程"按钮。

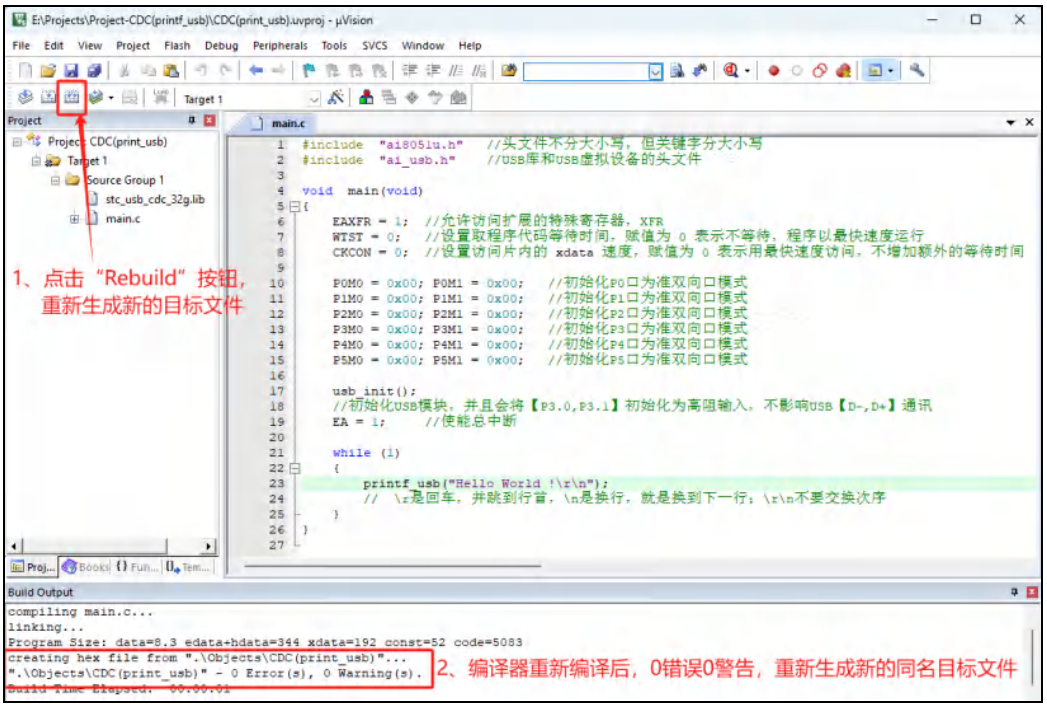
2.14.4.2 编译器重新编译后，ISP 软件会自动重新调入文件，进行不停电下载

如下图设置：

下载软件会每秒检测多次目标文件被编译的时间是否已更新，如时间已更新，则 ISP 软件会自动重新调入已打开的目标文件，接着通过 USB-CDC 通讯协议自动发送自定义下载命令如"@STCISP#"给单片机，单片机中的用户软件收到该命令后，就会软复位到单片机的系统监控程序区，等待 ISP 下载，本设置还指定了电脑端软件后续是通过 USB-HID 通讯协议传输用户程序到目标单片机进行 ISP 下载。



如上所示，不停电下载实验是成功的，但本设置的优点是不需要手工点击"下载/编程"按钮。

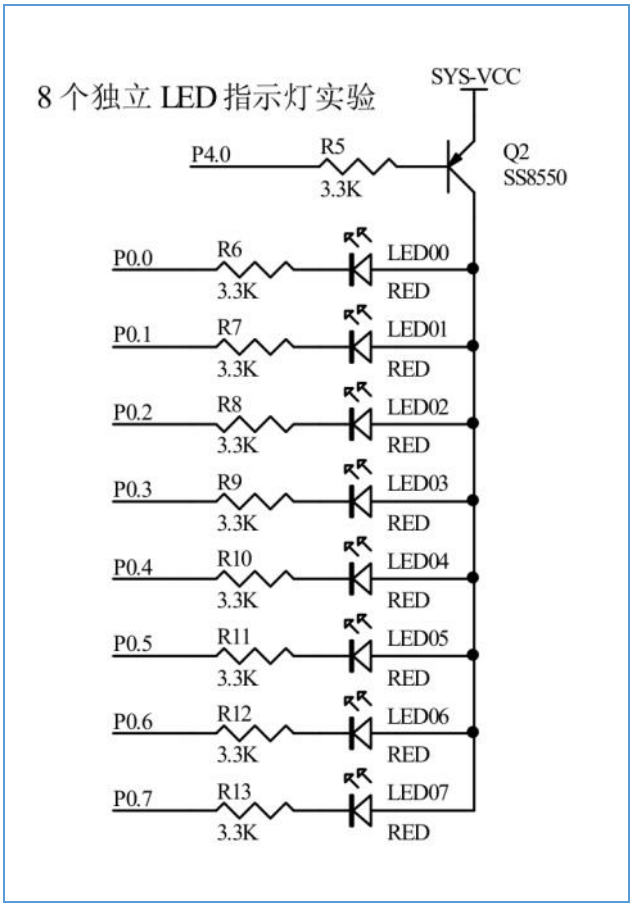


在编译集成环境中，点击"Rebuild"按钮，目标文件会被重新编译生成一次，ISP 软件发现目标文件被编译的时间发生变化后，ISP 软件就会执行前面描述的不停电下载动作。

2.15　点灯实验一：点亮 LED 灯，继续完成前面的工程项目，有视频讲解

本实验相应的视频链接：https://www.stcaimcu.com/forum.php?mod=viewthread&tid=15176#lastpost
<Ai8051U 实验箱的全部原理图可以通过点击本段文字进行查看>

2.15.1　原理介绍，在 Ai8051U 实验箱中有如下的一排 LED 指示灯



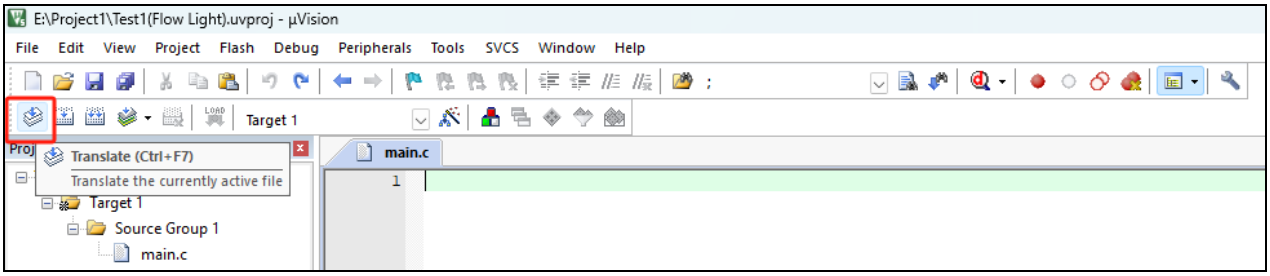
I/O 口 P4.0 控制 PNP 三极管的通断，控制这 8 个灯的整体供电。
如果 P4.0 输出 1，则 PNP 三极管不导通，这 8 个灯都不会获得电源，都不会有电流流过，都不会被点亮。
如果 P4.0 输出 0，则 PNP 三极管导通，这 8 个灯才有可能会流过电流，才有可能会被点亮。
如 P4.0 输出 0，PNP 三极管导通，特定 LED 灯的控制 I/O 口如输出 0，则该 LED 灯会有电流流过，会被点亮。
如 P0.0 输出 0，LED00 灯就会被点亮；如 P0.0 输出 1，LED00 灯就会熄灭；
如 P0.1 输出 0，LED01 灯就会被点亮；如 P0.1 输出 1，LED01 灯就会熄灭；
如 P0.2 输出 0，LED02 灯就会被点亮；如 P0.2 输出 1，LED02 灯就会熄灭；
如 P0.3 输出 0，LED03 灯就会被点亮；如 P0.3 输出 1，LED03 灯就会熄灭；
如 P0.4 输出 0，LED04 灯就会被点亮；如 P0.4 输出 1，LED04 灯就会熄灭；
如 P0.5 输出 0，LED05 灯就会被点亮；如 P0.5 输出 1，LED05 灯就会熄灭；
如 P0.6 输出 0，LED06 灯就会被点亮；如 P0.6 输出 1，LED06 灯就会熄灭；
如 P0.7 输出 0，LED07 灯就会被点亮；如 P0.7 输出 1，LED07 灯就会熄灭；

如 P0 口对外全部送 0，则这 8 个灯全亮；
如 P0 口对外全部送 1，则这 8 个灯全灭；
如果 P4.0 输出 1，则 PNP 三极管不导通，不管 P0 口对外全部送 0 或送 1，这 8 个灯都不会有电流流过，都不会亮。P4.0 控制 PNP 三极管，作为这 8 个 LED 指示灯的总电源开关。

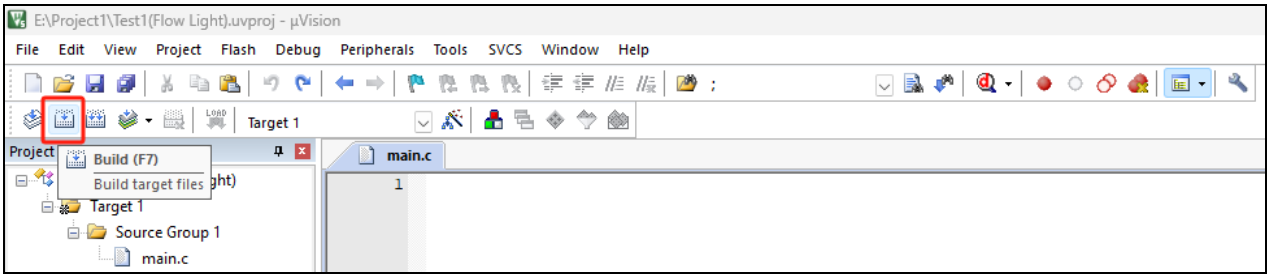
2.15.2　认识 Keil 软件的 Build Toolbar（Build 工具栏）

Keil 是一款软件开发系统，输入工程项目的程序源代码后，要进行编译（Build）。编译过程是将 C 语言源代码转换为机器可以执行的机器代码（通常是 HEX 文件），这样程序才能在目标硬件上运行。编译过程中，编译器对源代码进行预处理，处理宏定义、条件编译指令等。然后将预处理后的源代码转换成汇编语言。这一步会检查语法错误，并将高级语言转换为低级语言。下面我们认识一下 Build 工具栏的几个重要的功能按钮。

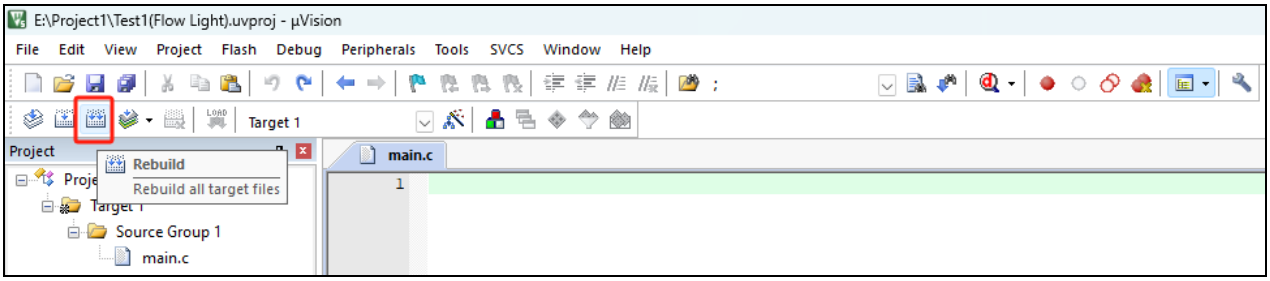
Translate（Translate the currently active file）按钮：编译当前正在编辑的源文件(main.c 仅生成 main.o)，在 Translate 过程中会检查源文件中的语法错误，但并不生成最终可执行文件。



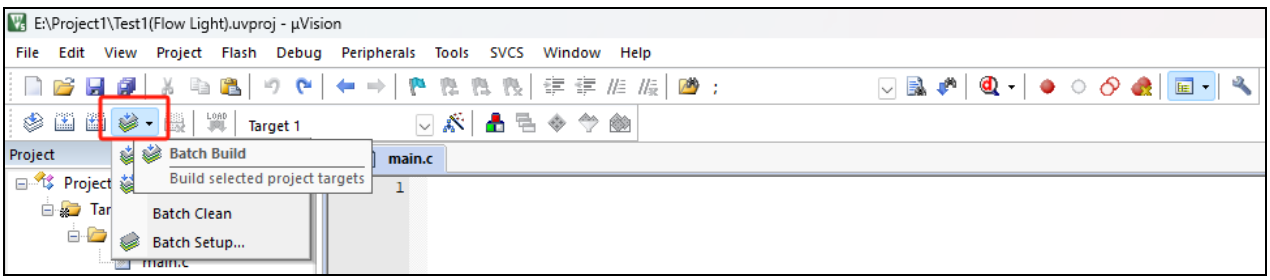
Build（Build target files）按钮：Build 在技术文档中是 increase build，所以理解 Build 按钮是编译工程中当前修改的文件及与此文件相关联的所有文件（模块），检查语法错误，并重新链接生成可执行文件。这样特别在调试大型项目时，因为不需要每次都从头开始编译整个项目，可以节省时间。平时项目开发调试验证过程中，也更多选择用 Build 编译生成可执行文件验证。如果工程是首次编译，会直接调用 Rebuild All。



Rebuild（Rebuild all target files）按钮：编译全部文件，检查语法错误，并重新链接生成可执行文件。



Batch Build（Build selected project targets）按钮：一次编译多个项目。在 Keil 中，可创建多个 projects，并通过 Batch Build 同时对这些 projects 进行编译。

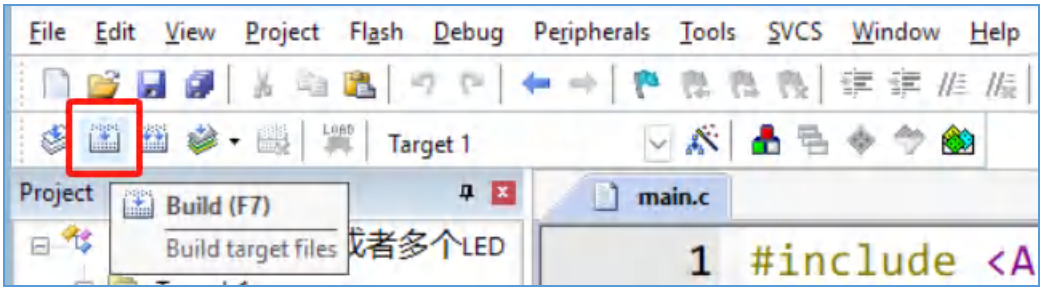


2.15.3　代码部分

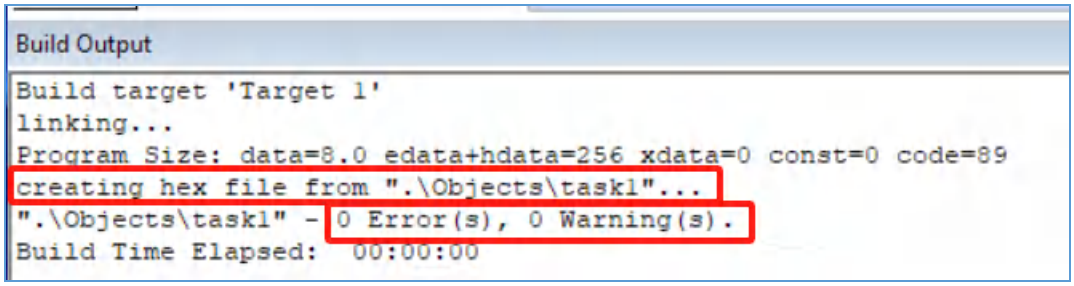
要想实现“点亮 LED 灯”，则在上面的工程项目的 main.c 文件中可输入如下代码：
//实验箱内，所有的 LED 通过一个 PNP 三极管控制总的通断。给 P40 接口一个低电平后所有的 LED 阳极就会有高电平，给对应 P0 的端口低电平即可控制对应的 LED 亮起来。

```
#include <AI8051U.H>                    //包含 AI8051U 的头文件

void main(void)
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)。
    WTST = 0;                           //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00; P0M1 = 0x00;           //设置 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x00;           //设置 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;           //设置 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x00;           //设置 P3 口为准双向口模式
    P4M0 = 0x00; P4M1 = 0x00;           //设置 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;           //设置 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;           //设置 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;           //设置 P7 口为准双向口模式
//上面的将所有 I/O 口都初始化为准双向口工作模式的程序，
//可以使用 AIapp-ISP 软件的“I/O 口配置工具”来生成。见本节后面的截图。
    P40 = 0;                            //打开 LED 部分的供电
    while(1)
    {                                   //LED 点亮部分
        P0 = 0;                         //对 P0 端口置 0，点亮整组 LED
        //P00 = 0;                      //对单个端口置 0，点亮单个 LED，这个点亮的是 LED00
        //P01 = 0;                      //对单个端口置 0，点亮单个 LED，这个点亮的是 LED01
    }
}
```

编写完成代码后，通过按键盘上的 F7 或者工具栏上的 Build 按钮进行编译



编译完成后，会在底部的 Build Output 生成编译信息



出现 creating hex file 和 0 Error(s) 0 Warning（s）即为编译成功，无报错且成功创建了 hex 文件。此 hex 文件，一般是在对应工程目录下的 Objects 文件夹里。

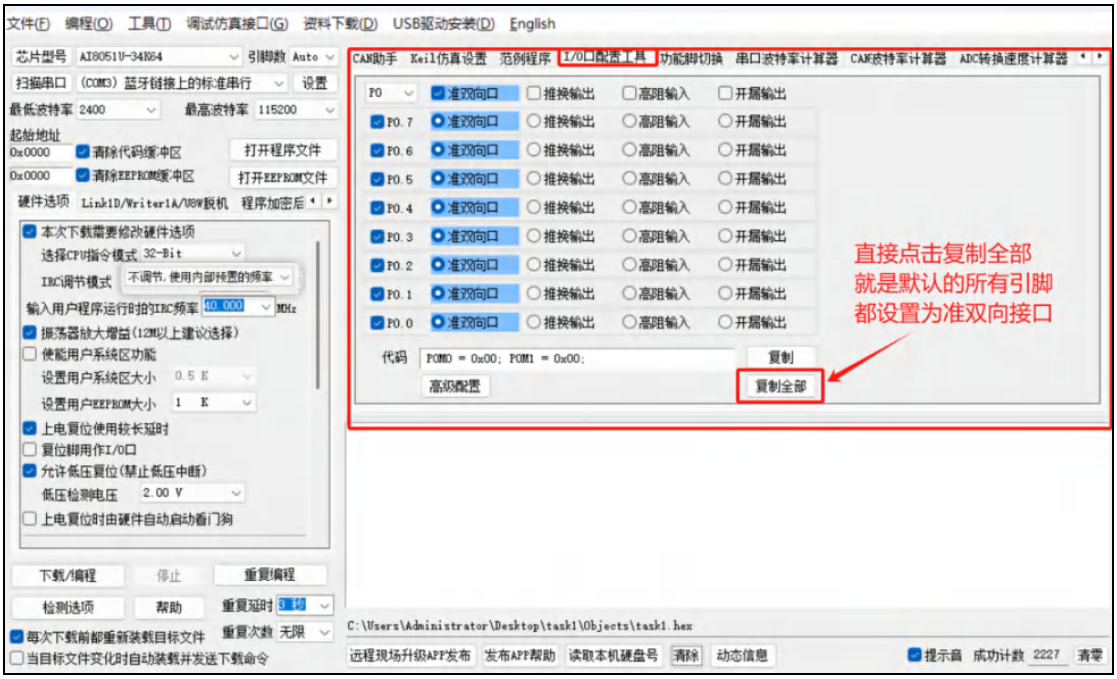
【小技巧--灵活运用 AIapp-ISP 软件的工具自动生成代码】

I/O 口的初始化部分就可以使用 **AIapp-ISP** 软件的“**I/O 口配置工具**”来生成。

例如上述代码中这一段将所有 I/O 口设置为准双向口模式的代码：

```
P0M0 = 0x00; P0M1 = 0x00;
P1M0 = 0x00; P1M1 = 0x00;
P2M0 = 0x00; P2M1 = 0x00;
P3M0 = 0x00; P3M1 = 0x00;
P4M0 = 0x00; P4M1 = 0x00;
P5M0 = 0x00; P5M1 = 0x00;
P6M0 = 0x00; P6M1 = 0x00;
P7M0 = 0x00; P7M1 = 0x00;
```

就可以使用 AIapp-ISP 的 I/O 口配置工具进行生成：

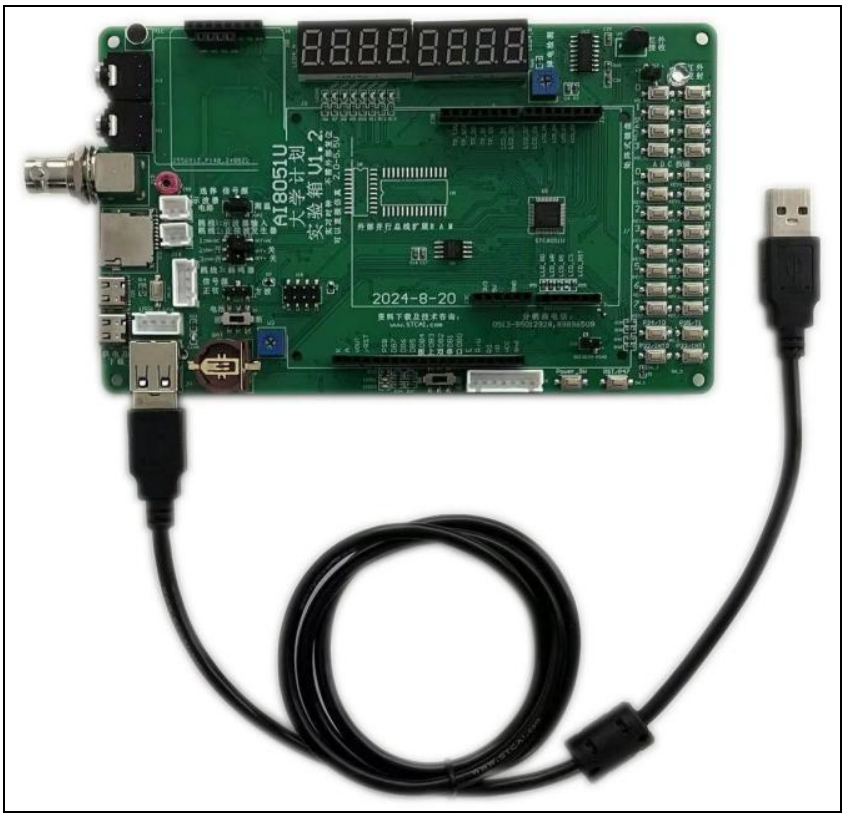


将复制出来的内容直接粘贴到代码文件中，不用手动输入。

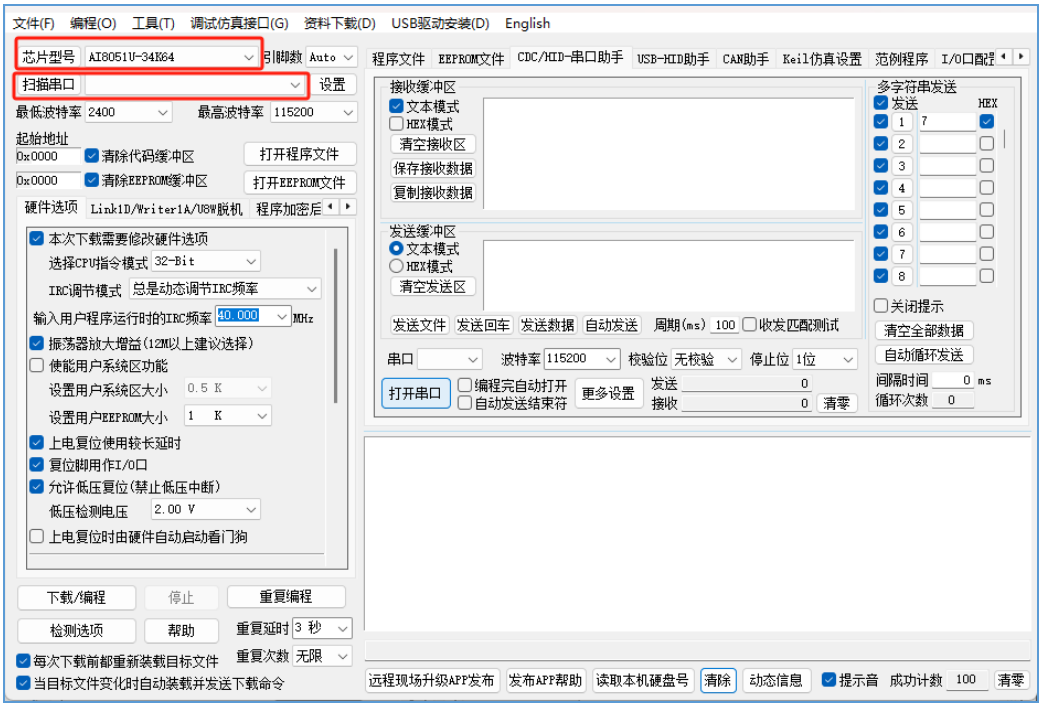
2.15.4　下载用户程序到实验并查看实验现象，实物图

2.15.4.1　连接实验箱和电脑

拿出实验箱中配置的 USB-TypeA 到 USB-TypeA 线将实验箱与电脑正确连接。

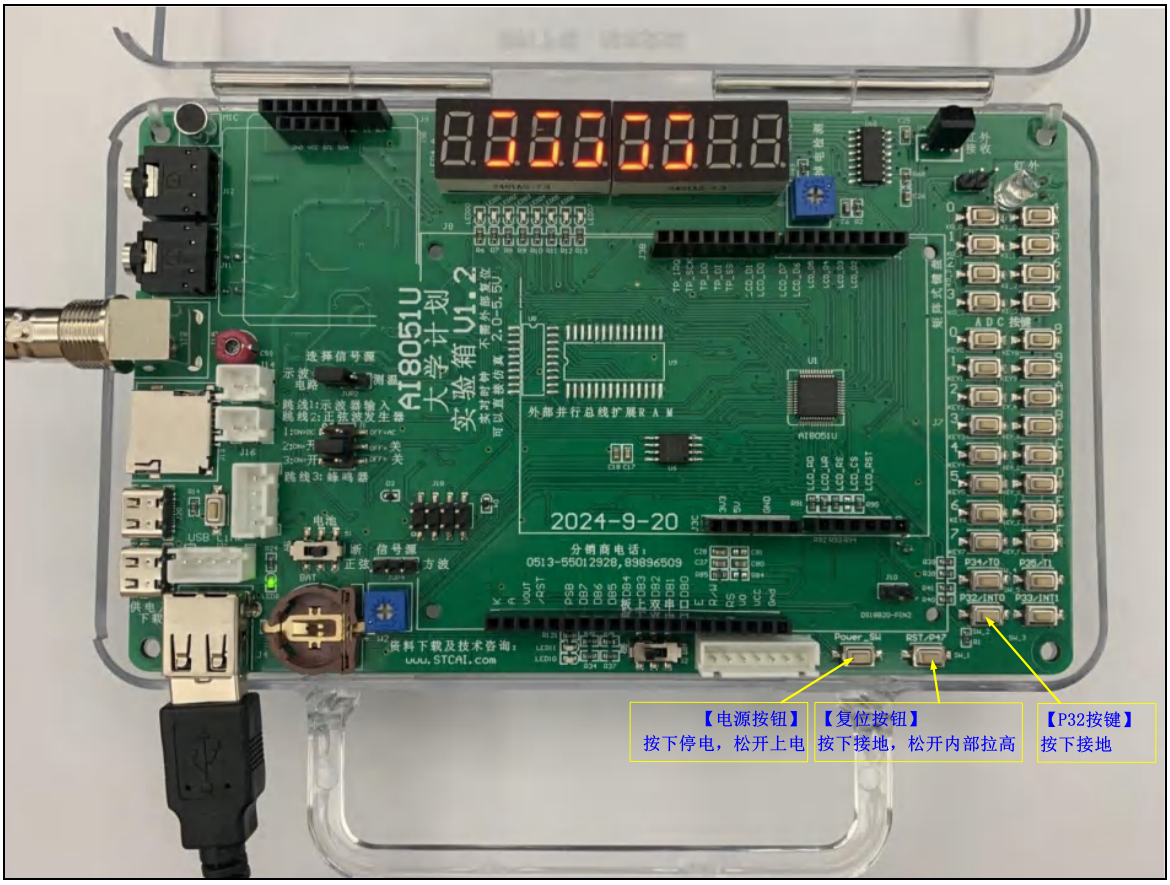


然后打开 AIapp-ISP-v6.95F 或以上版本。选择单片机型号：Ai8051U-34K64，如下图，我们看到“扫描串口”后面的端口列表框不是“（HID1）USB-Writer”，说明还不能下载程序到实验箱。

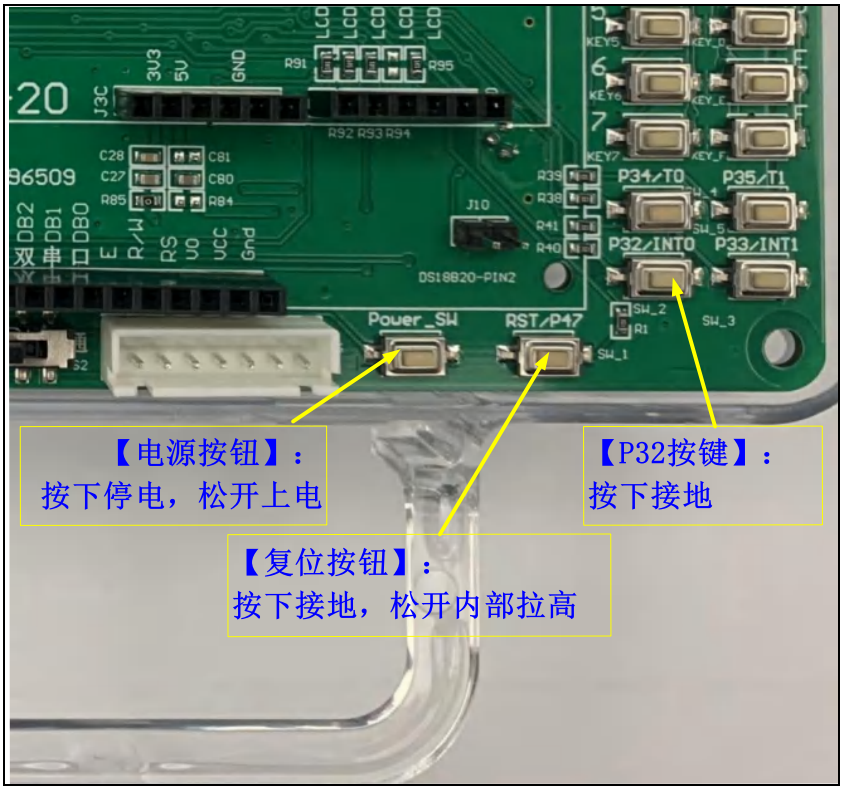


2.15.4.2 硬件 USB 下载用户程序步骤

下图是实验箱已经和电脑相连的正面图片，数码管是随机显示的，数码管下面的 8 个 LED 灯都是暗的，**此实验的目的就是点亮这 8 个灯。**



局部放大实验箱图片，其正下方有两排按键，找到 P3.2 按键和 Power_SW 按键



【方法一，P3.2 按键，再结合停电上电下载】

1、按下板子上的 P3.2 按键，就是 P3.2 接地

2、给目标芯片重新上电，不管之前是否已通电。

===本 Ai8051U 实验箱是：按下电源按钮停电，再松开电源按钮上电，

这时电脑端 ISP 下载软件会自动识别出“(HID1) USB Writer”，此时就与 P3.2 状态无关了，可以松开 P3.2 按键（P3.2 在用户程序区可做其它任意用途）

3、点击电脑端下载软件中的【下载/编程】按钮（注意：USB 下载与串口下载的操作顺序不同）

电脑端 ISP 下载软件提示：下载进行中…，几秒钟后，提示下载成功！

【方法二：复位管脚低电平复位下载】

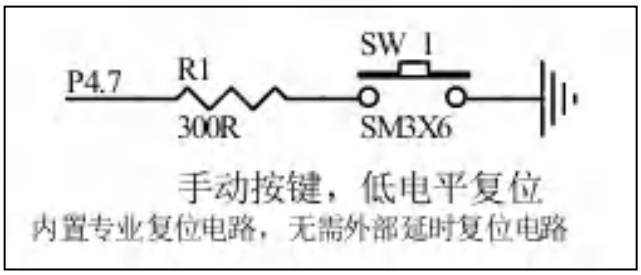
USB 连接好并已上电的情况下，外部按键复位也可进入 USB 下载模式，注意：

P4.7-nRST 出厂时默认是 P4.7-I/O 功能，要改为复位功能，需 ISP 烧录时取消设置复位脚用作 I/O 口，停电一次再上电才生效，程序区中用户程序也可改为复位脚或 I/O，这个立即生效。

1、按下复位管脚/P4.7-nRST 外接的低电平复位按键，复位 MCU，

再松开复位键，MCU 首先从系统程序区启动运行系统程序，判断是否要下载用户程序，

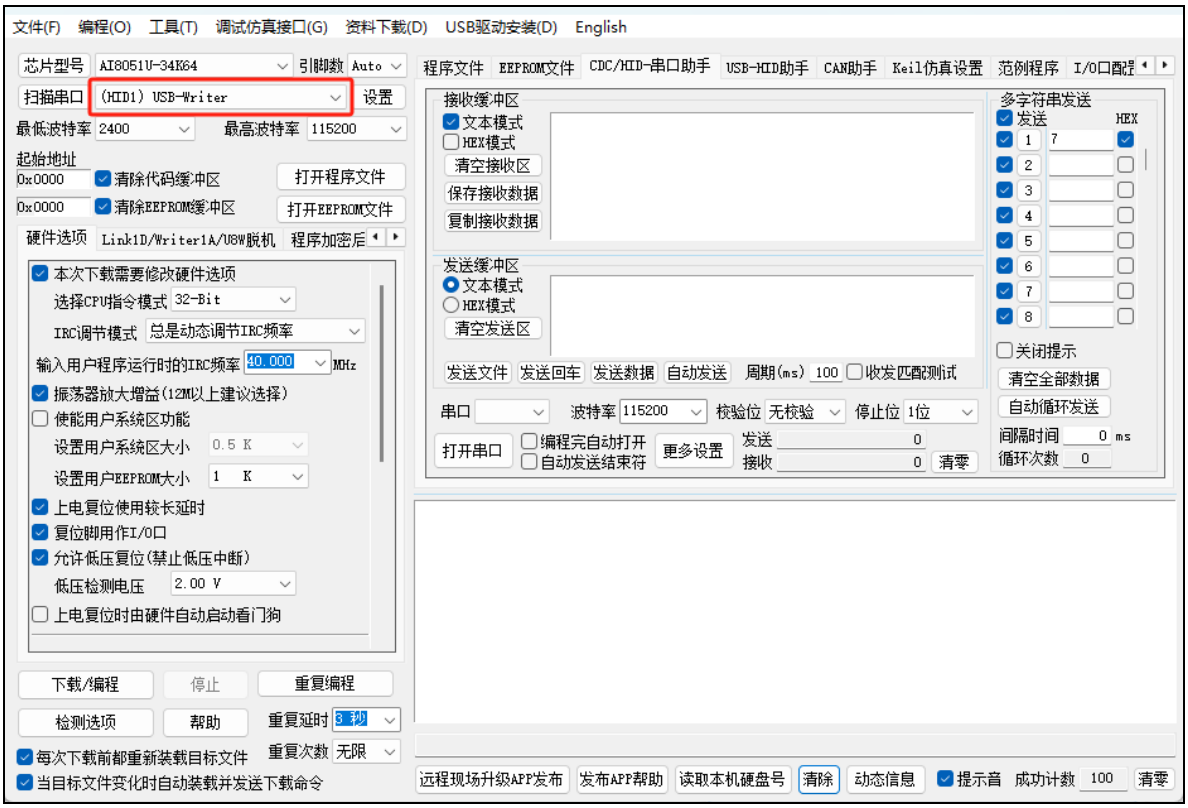
等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后



2、点击电脑端下载软件中的【下载/编程】按钮

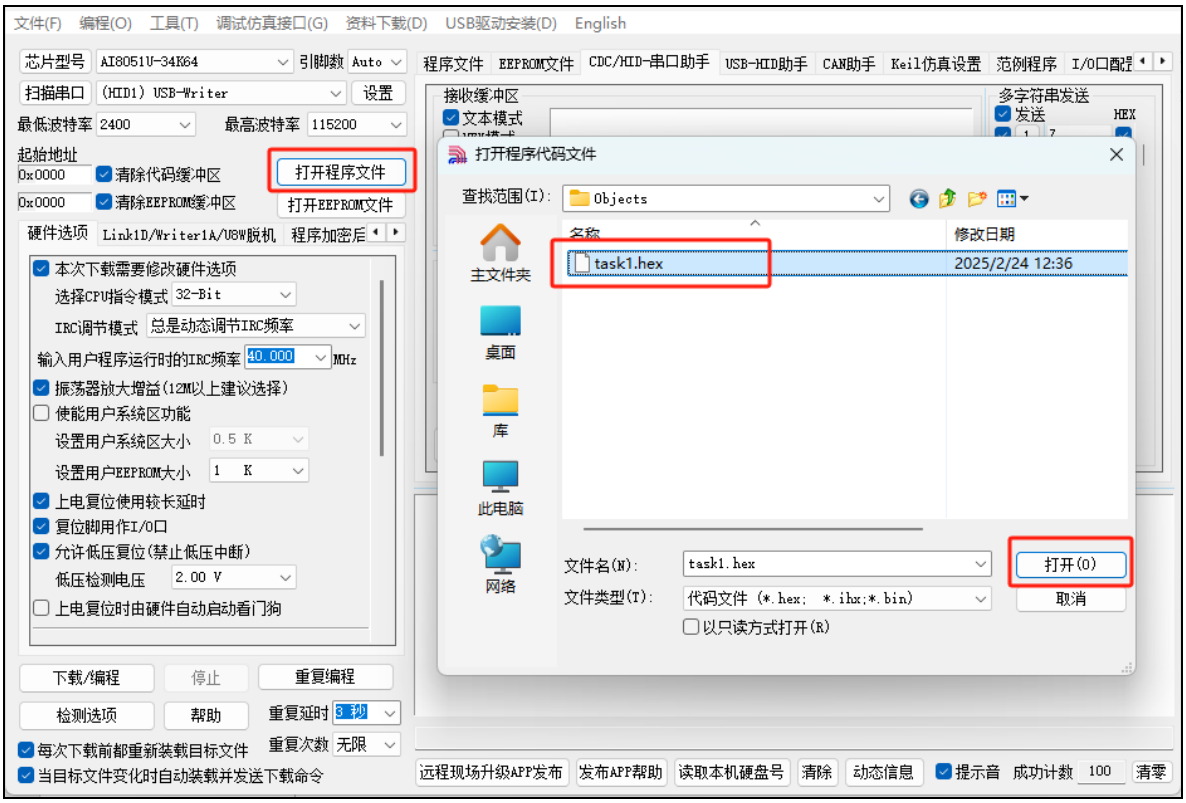
电脑端 ISP 下载软件提示：下载进行中…，下载成功！

备注：如果目标 MCU 的用户程序区已有 USB 通讯程序（含 CDC、HID 等），则通过复位脚复位进入 USB 下载模式不被支持，只能通过停电上电冷启动下载或者从用户程序区软复位到系统区下载

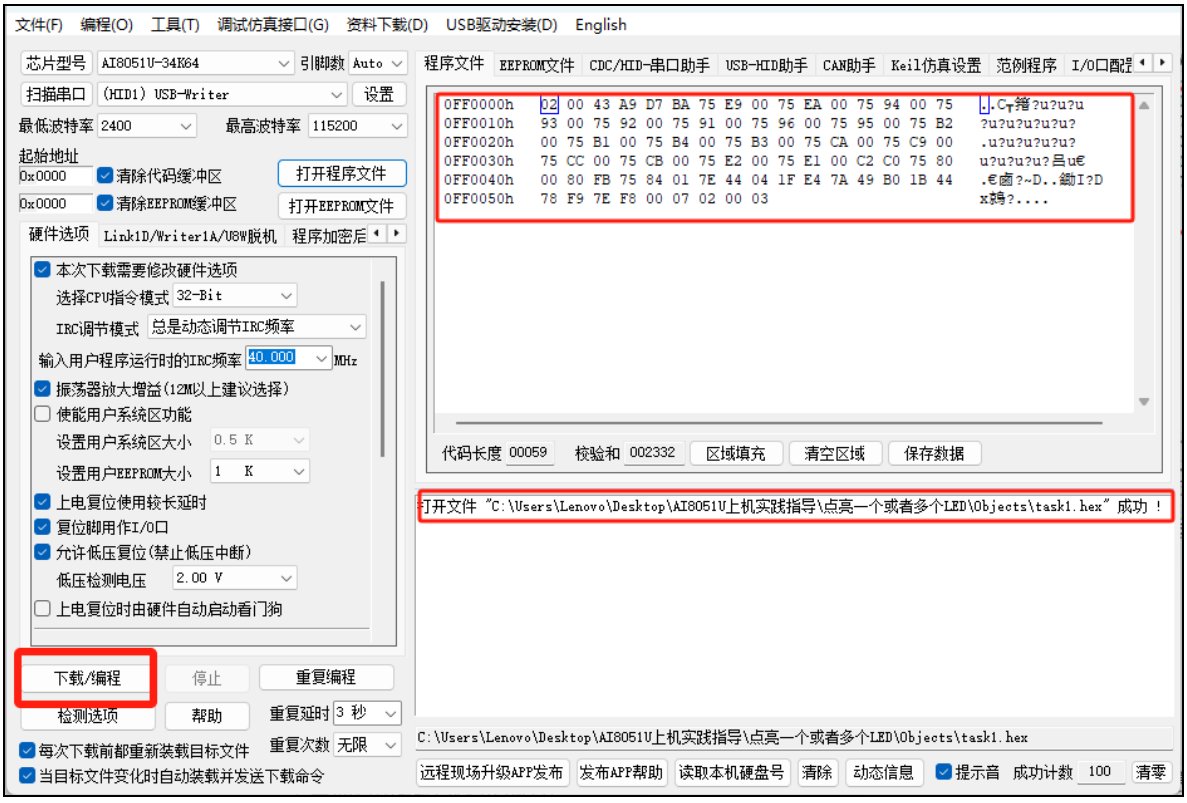


2.15.4.3 下载用户程序文件并观察实验现象

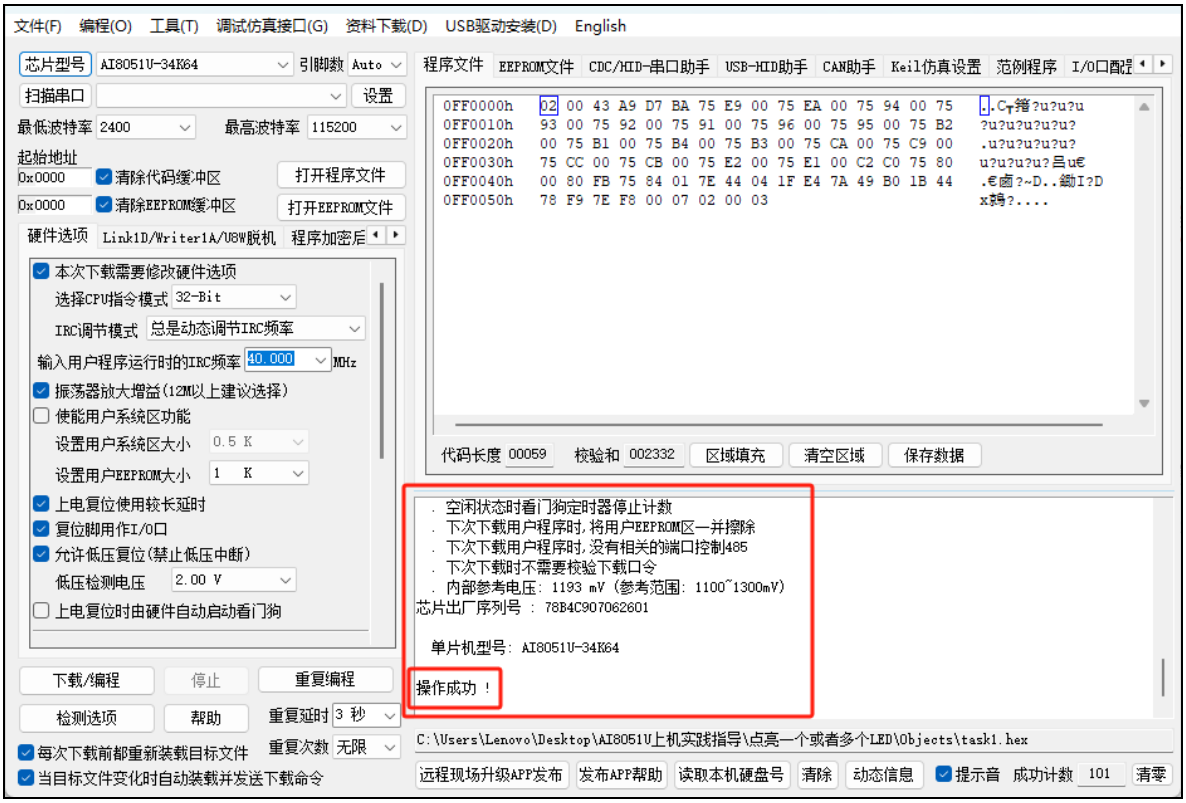
现在就打开用户程序文件，点击 AIapp-ISP 软件界面上的“打开程序文件”（打开的是项目程序文件编译无误后生成的 HEX 文件，一般默认存放在该项目的 Objects 目录中），调入在前面小节编译好的 HEX 文件：



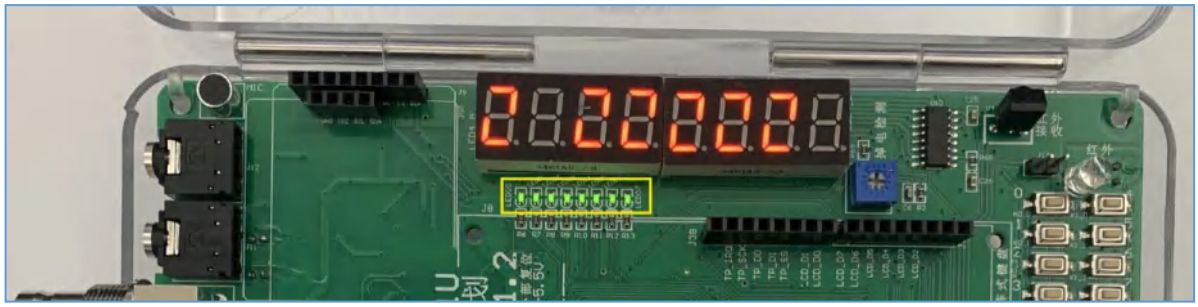
然后，AIapp-ISP 软件界面如下图所示，出现“‘打开文件***.hex’成功！”



点击“下载/编程”，把 task1.hex 烧录到实验箱，AIapp-ISP 软件界面出现：操作成功！恭喜你！



这时，再看实验箱，刚刚暗淡的 LED 灯，齐齐亮了起来！至此，实验成功！



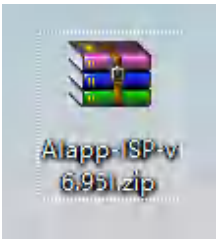
2.15.5　使用【AiCube】项目创建助手工具创建并实现点亮 LED 灯

STCAI 在最新版的 AIapp-ISP 软件中，推出【AiCube】项目创建助手工具，目前支持 Ai8051U-32Bit 各功能模块的智能化配置，并生成程序框架。这项功能在程序员建立项目时，简化了项目的初始化难度，提高效率，避免因不太熟悉或不完全了解该单片机的性能而忽略个别配置，引起编译错误等问题。尤其对刚涉足单片机的初学者，此工具不仅引导其了解单片机编程流程，还极大的降低了入门门槛，缩短了摸索的时间，提高入门的兴趣和信心。现在我们使用这个工具来生成实现此实验功能的配置和代码。

a) 安装【AIapp-ISP】下载/编程/烧录/软件

安装 STCAI 公司研发的 AIapp-ISP 下载 / 编程 / 烧录 / 软件，含强大的辅助开发工具包。登录 STCAI 官网，下载最新版的 AIapp-ISP 安装包，下载链接如下：https://www.stcai.com/gjrj

下载下来的是压缩包文件如下图：

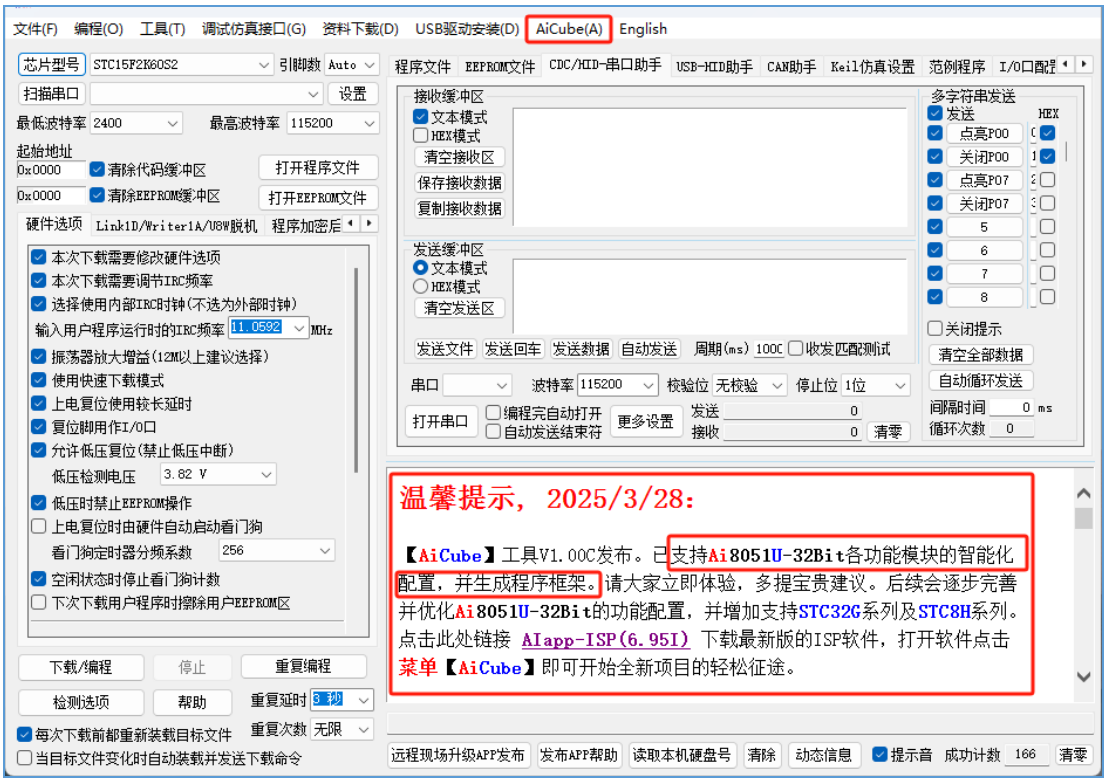


双击后解压缩如下图：

如果是解压缩到非桌面的其他目录，如 D:\STCAI-ISP，建议创建快捷方式到桌面方便打开。这实际是一个免安装的绿色软件。

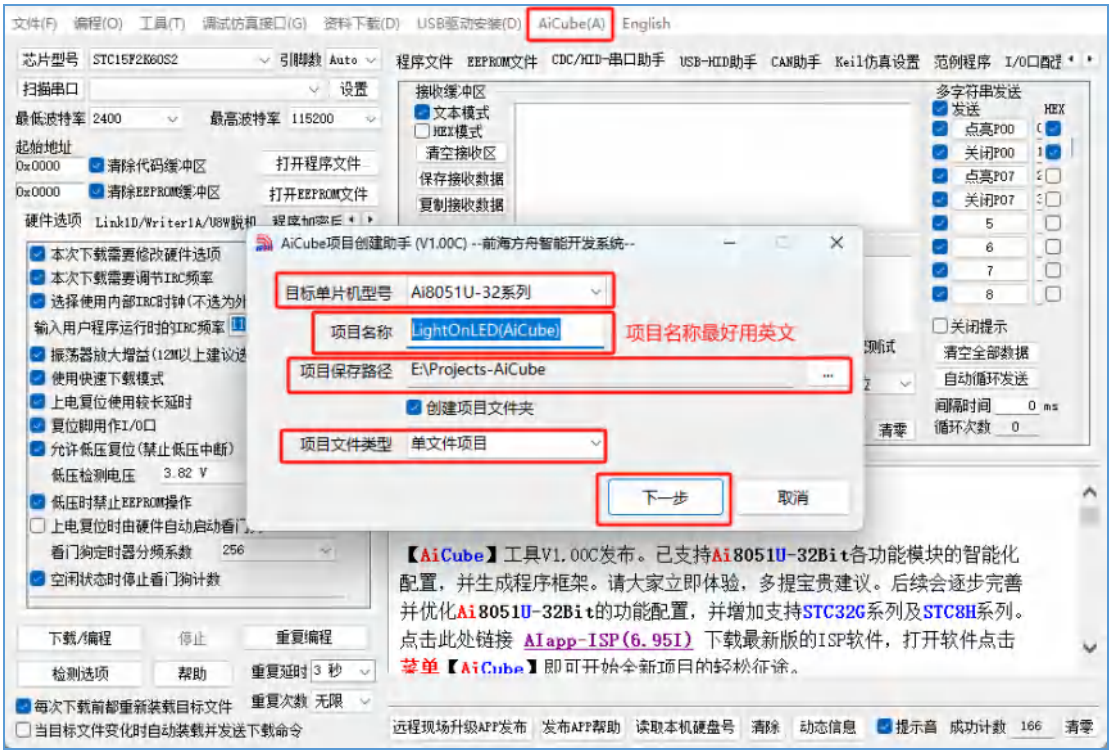
b) 打开【AiCube】助手

双击图标，即可打开 AIapp-ISP 软件界面，即可直接使用，如下图：

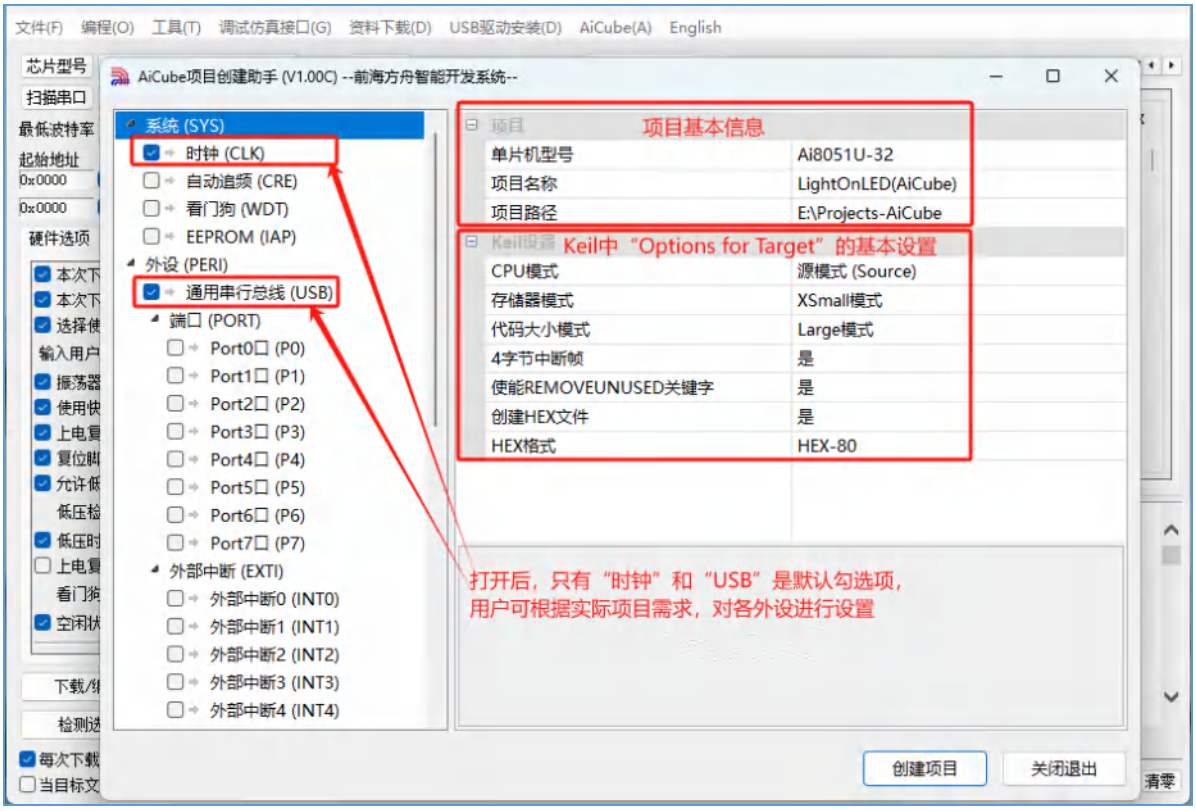


c) 使用【AiCube】助手创建项目

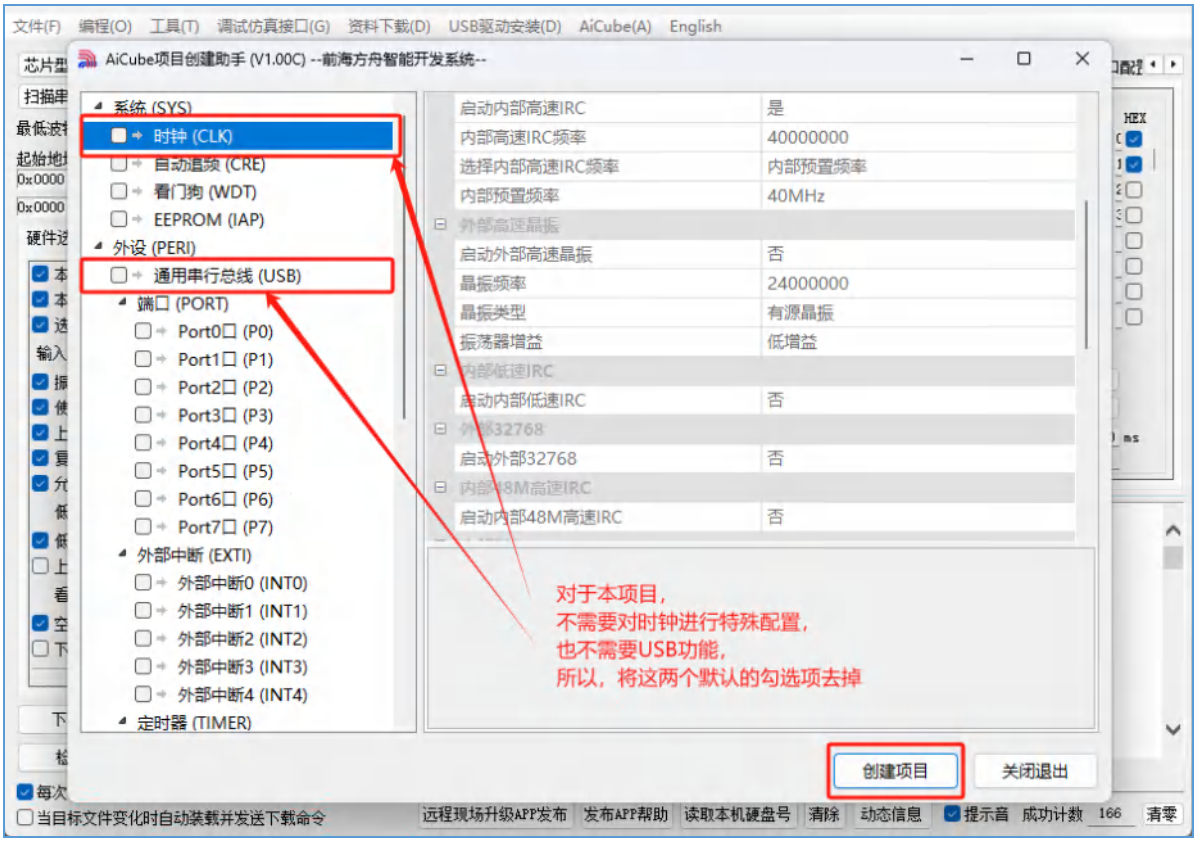
点击【AiCube（A）】，弹出如下窗口：



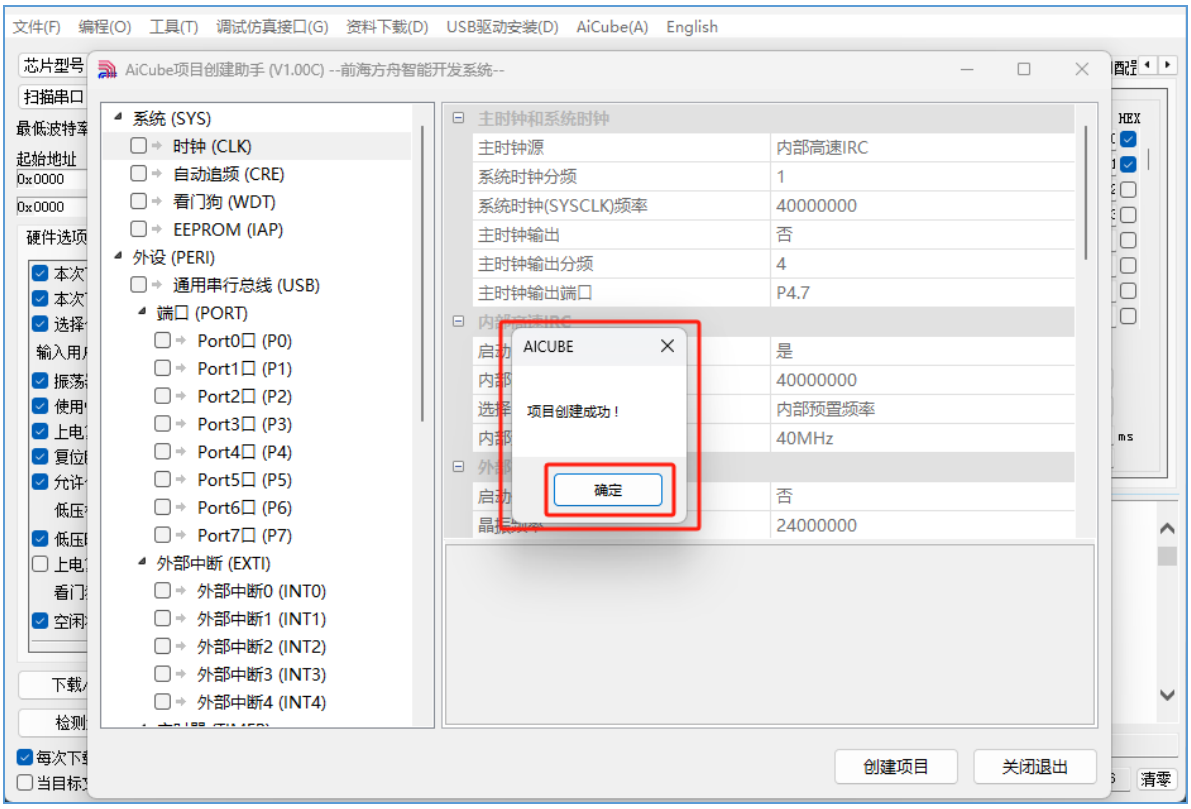
选择并填写项目基本信息，建议项目名称用英文！点击“下一步”，出现如下配置界面：



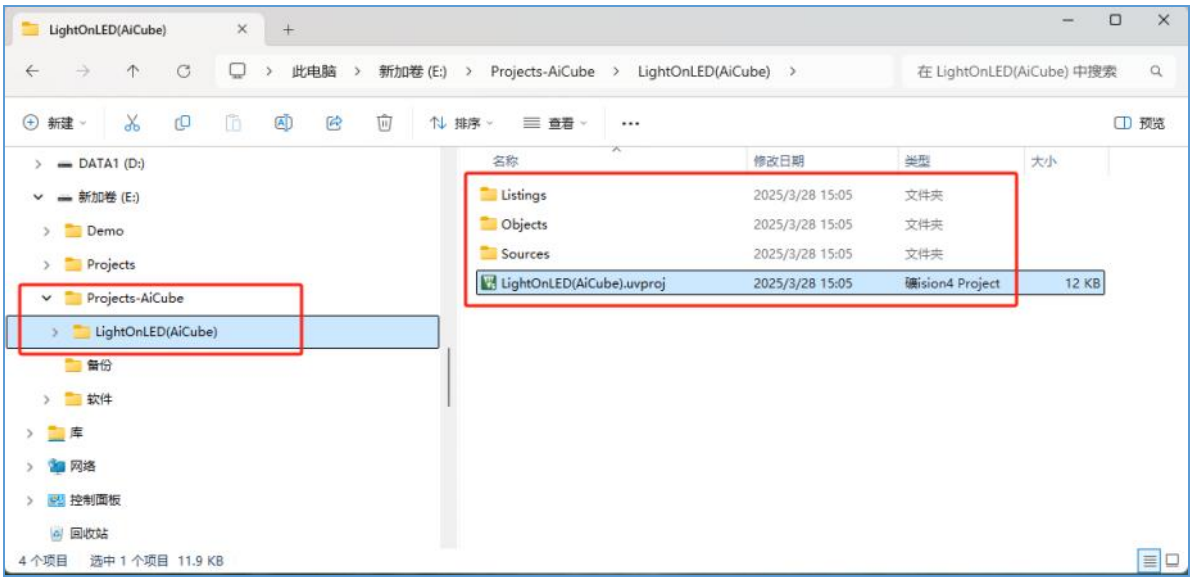
根据本项目的实际需求，对配置进行如下修改：



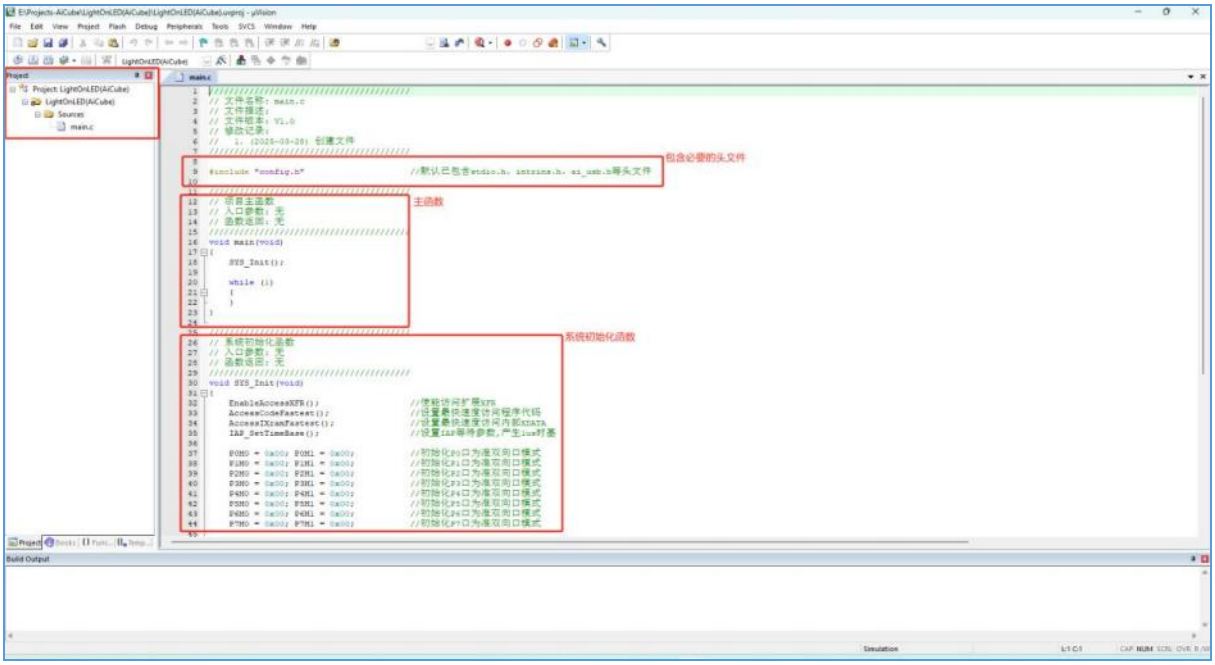
点击"创建项目"，【AiCube】助手提示："项目创建成功！"



点击“确定”，即成功创建了项目及项目文件目录“LightOnLED(AiCube)”。找到并打开刚刚创建的项目目录，发现刚刚的操作过程，【AiCube】助手已经帮我们建立了项目相关目录及 LightOnLED(AiCube).uvproj 文件，如下图：



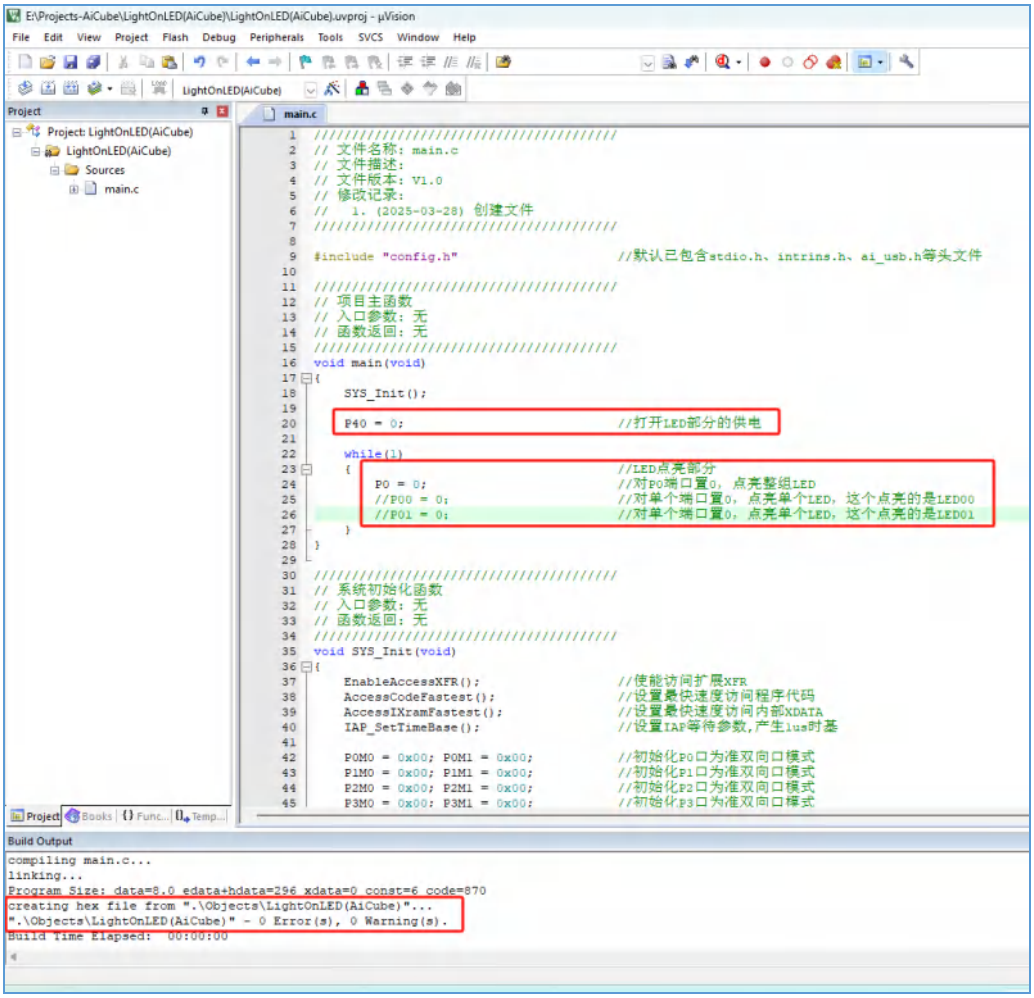
双击：LightOnLED(AiCube).uvproj 文件，Keil 会自动打开项目文件，如下图：



这是一个非常简单的入门实验，我们发现【AiCube】助手已经帮我们在项目的 main.c 中按模块化设计，并已经包含了：必要的头文件、主函数、系统初始化函数、延时函数（根据主频自动计算的延时初值）。

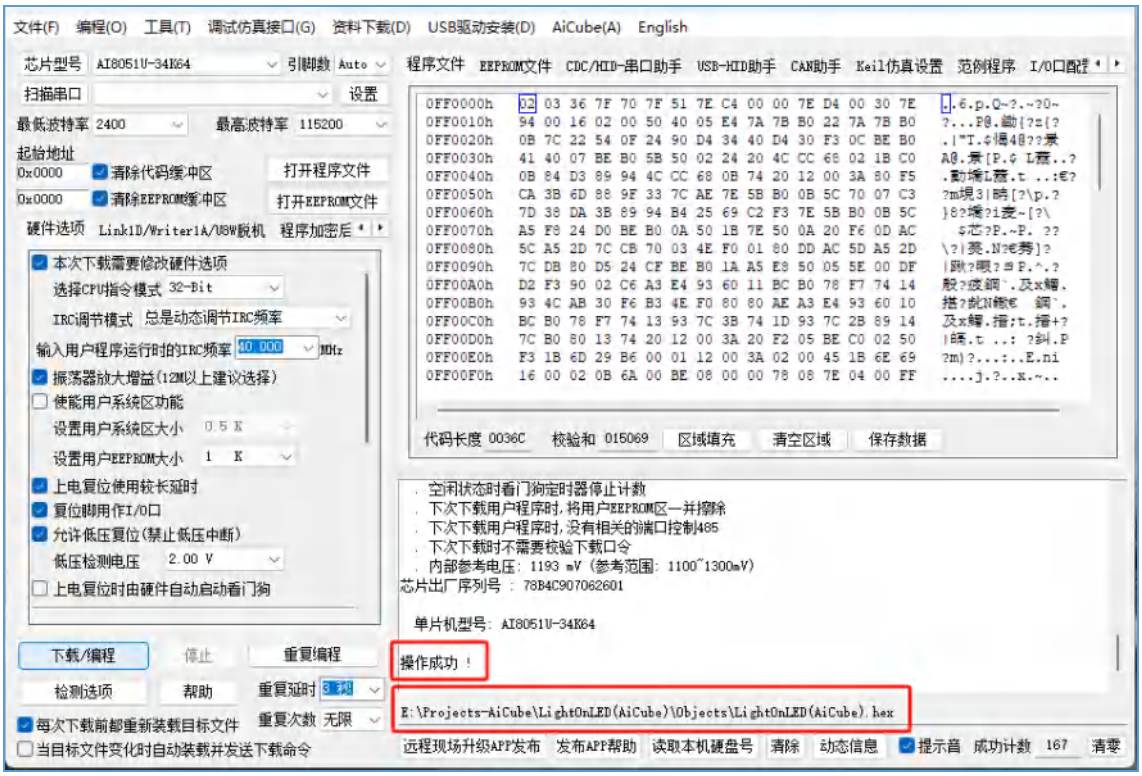
d) 增加实验项目代码

为了实现点灯的功能，我们现在**只需要在主函数中补充 LED 控制代码**，保存，编译，如下图：



e) 使用 Ai8051U 实验箱查验实验效果

连接好 Ai8051U 实验箱，运行 AIapp-ISP 软件系统，按步骤打开并下载程序“LightOnLED(AiCube).hex”，如下图：



Ai8051U 实验箱上的 LED 灯齐齐被点亮：

有了【AiCube】助手，单片机编程就是这么简单！快用起来吧！

2.15.6　USB 不停电下载，ISP 软件如何设置

在开发过程中，程序经常需要反复修改，重新编译后生成新的 HEX 文件，就需要停电/上电进入下载模式，很不方便。其实可以利用我们提供的 USB 不停电 ISP 下载功能，进行自动下载。

在有 USB 功能的实验中，都要加载 USB 库函数，并对 USB 进行初始化。由于在 USB 库函数的代码中已经定义了不停电 ISP 下载命令"@STCISP#"，只需要在下载软件的"收到用户命令后复位到 ISP 监控程序区"的选项卡中进行如下设置，即可实现【USB 不停电下载】功能。

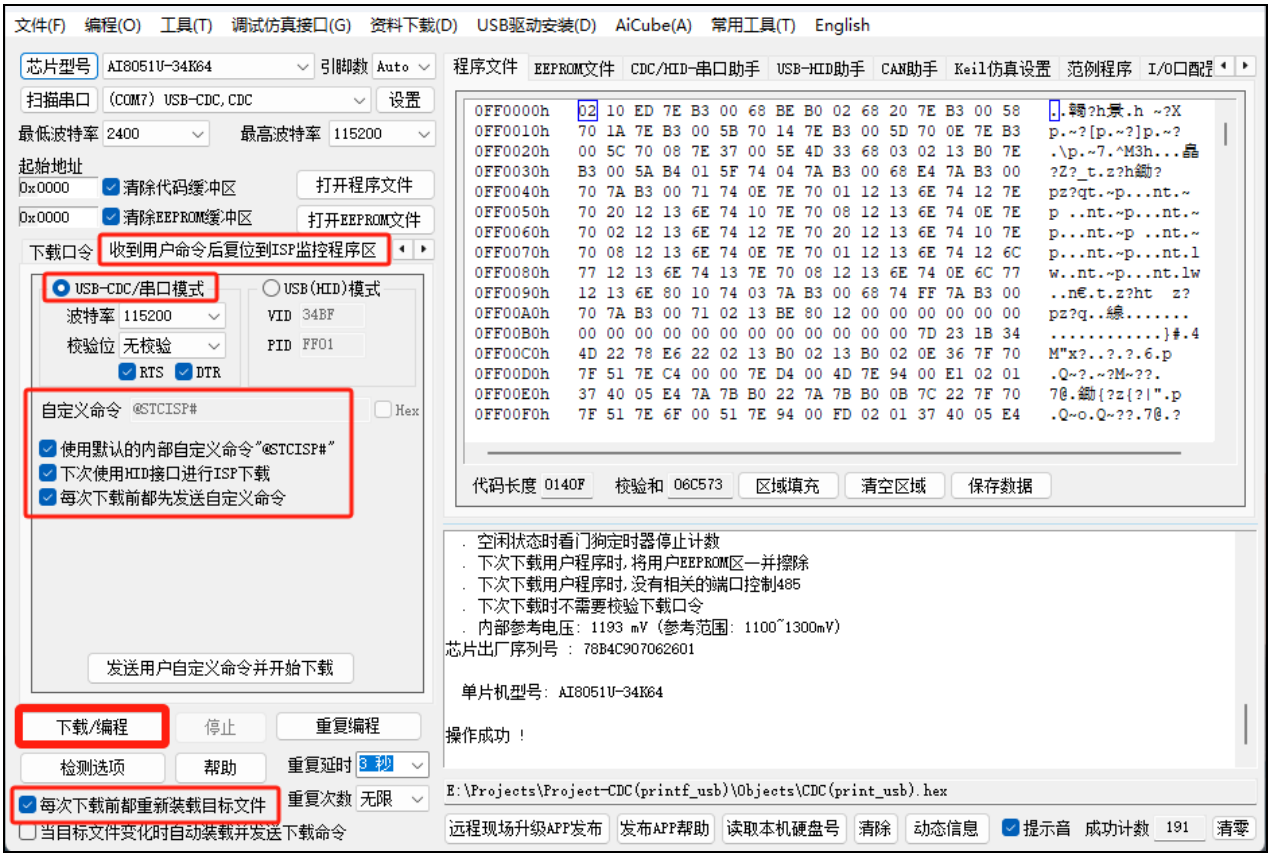
- 如使用 USB-CDC 库函数产生 USB-CDC 虚拟串口，可实现 USB-CDC 不停电自动 ISP 下载功能，并可忽略传统串口的波特率、校验位等参数的设置，下载时系统程序区还是使用的 USB-HID 通讯协议；
- 如果是传统的串口，需正确设置波特率、校验位等参数，才能实现串口的不停电自动 ISP 下载功能；
- 如果使用 USB-HID 库函数产生 USB-HID 设备，则选择"USB（HID）模式"就能实现不停电自动 ISP 下载功能，下载时系统程序区还是使用的 USB-HID 通讯协议。

ISP 软件如何设置：

2.15.6.1　手动点击【下载/编程】按钮，ISP 软件自动重新打开文件，不停电下载

如下图设置，就可以实现：

在点击"下载/编程"按钮后，ISP 软件会重新调入已打开的目标文件一次（不管目标文件有无变化），接着通过 USB-CDC 通讯协议自动发送自定义下载命令如"@STCISP#"给单片机，单片机中的用户软件收到该命令后，就会软复位到单片机的系统监控程序区，等待 ISP 下载，本设置还指定了电脑端软件后续是通过 USB-HID 通讯协议传输用户程序到目标单片机进行 ISP 下载。

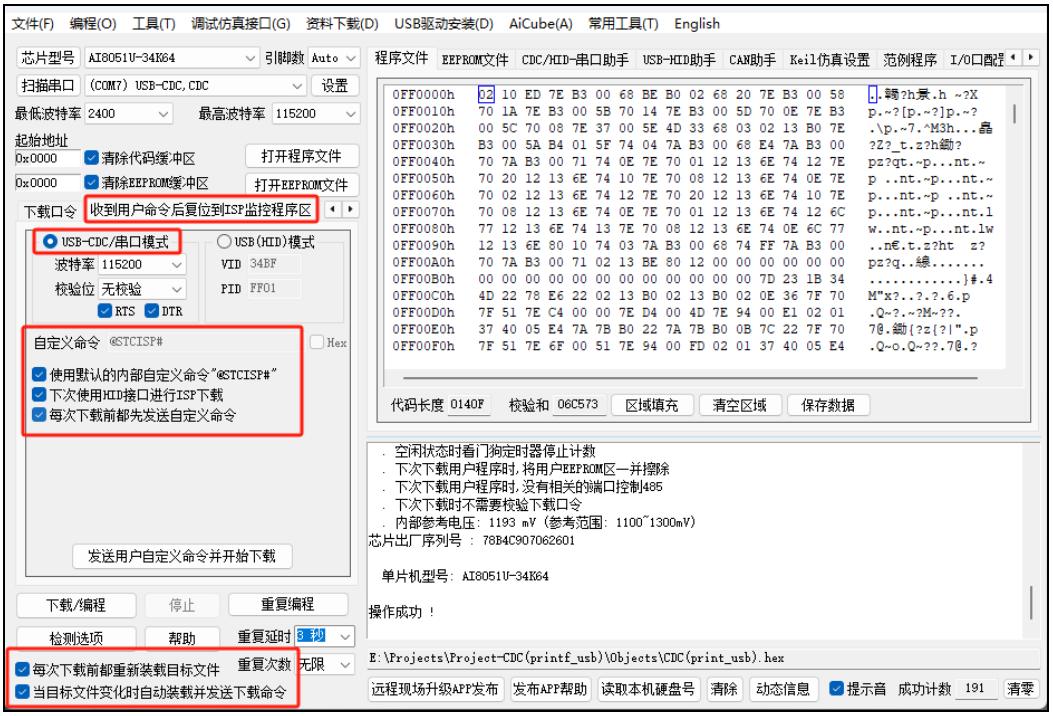


如上所示，不停电下载实验是成功的，但本设置还需要手工点击一次"下载/编程"按钮。

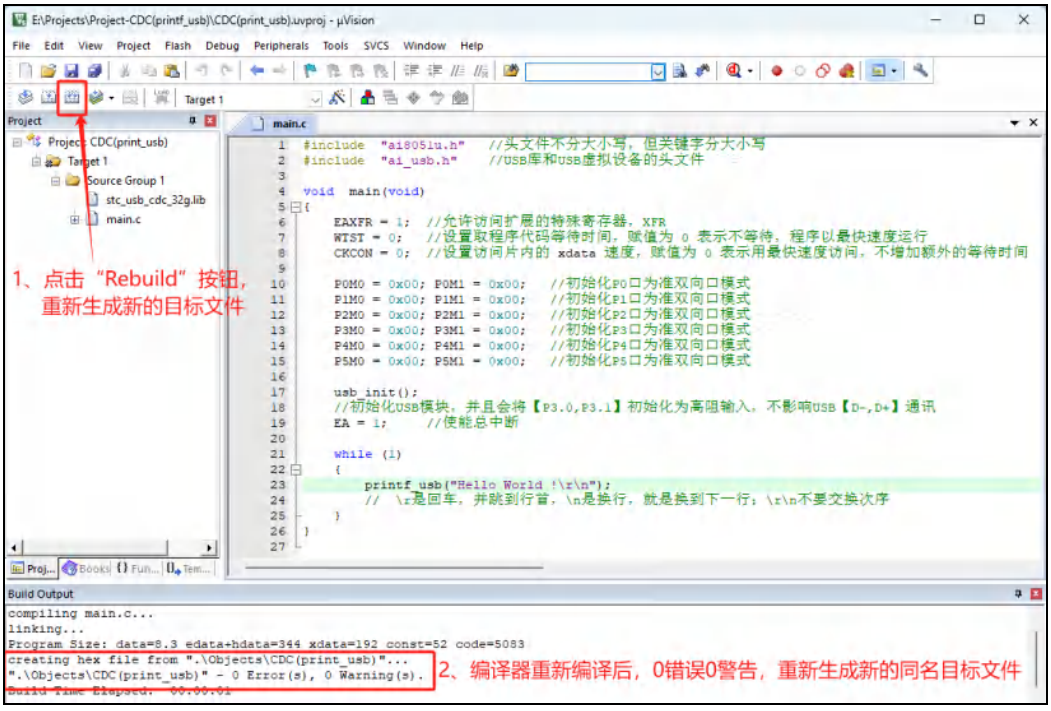
2.15.6.2　编译器重新编译后，ISP 软件会自动重新调入文件，进行不停电下载

如下图设置：

下载软件会每秒检测多次目标文件被编译的时间是否已更新，如时间已更新，则 ISP 软件会自动重新调入已打开的目标文件，接着通过 USB-CDC 通讯协议自动发送自定义下载命令如"@STCISP#"给单片机，单片机中的用户软件收到该命令后，就会软复位到单片机的系统监控程序区，等待 ISP 下载，本设置还指定了电脑端软件后续是通过 USB-HID 通讯协议传输用户程序到目标单片机进行 ISP 下载。



如上所示，不停电下载实验是成功的，但本设置的优点是不需要手工点击"下载/编程"按钮。

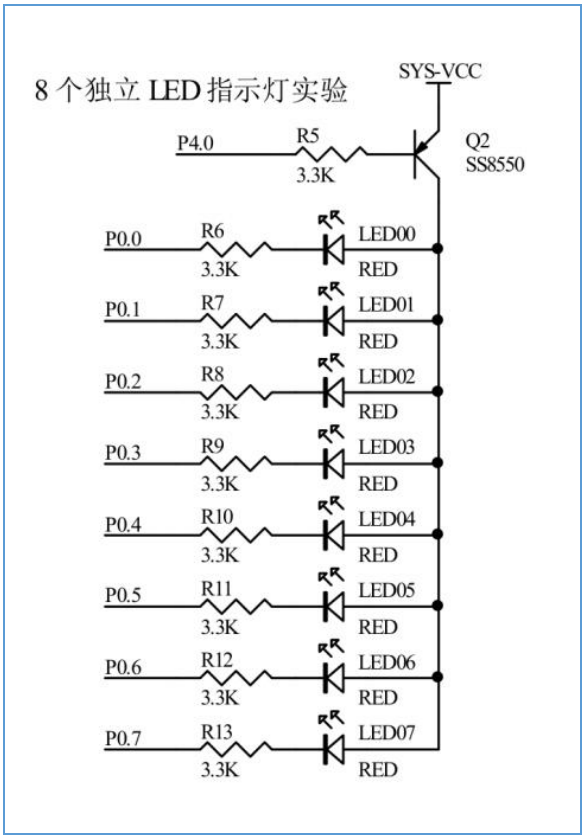


在编译集成环境中，点击"Rebuild"按钮，目标文件会被重新编译生成一次，ISP 软件发现目标文件被编译的时间发生变化后，ISP 软件就会执行前面描述的不停电下载动作。

2.16　点灯实验二：让 LED 灯闪烁起来，有视频讲解

本实验相应的视频链接：https://www.stcaimcu.com/forum.php?mod=viewthread&tid=15176#lastpost
<Ai8051U 实验箱的全部原理图可以通过点击本段文字进行查看>

2.16.1　原理介绍，在 Ai8051U 实验箱中有如下的一排 LED 指示灯



I/O 口 P4.0 控制 PNP 三极管的通断，控制这 8 个灯的整体供电。
如果 P4.0 输出 1，则 PNP 三极管不导通，这 8 个灯都不会获得电源，都不会有电流流过，都不会被点亮。
如果 P4.0 输出 0，则 PNP 三极管导通，这 8 个灯才有可能会流过电流，才有可能会被点亮。
如 P4.0 输出 0，PNP 三极管导通，特定 LED 灯的控制 I/O 口如输出 0，则该 LED 灯会有电流流过，会被点亮。
如 P0.0 输出 0，LED00 灯就会被点亮；如 P0.0 输出 1，LED00 灯就会熄灭；
如 P0.1 输出 0，LED01 灯就会被点亮；如 P0.1 输出 1，LED01 灯就会熄灭；
如 P0.2 输出 0，LED02 灯就会被点亮；如 P0.2 输出 1，LED02 灯就会熄灭；
如 P0.3 输出 0，LED03 灯就会被点亮；如 P0.3 输出 1，LED03 灯就会熄灭；
如 P0.4 输出 0，LED04 灯就会被点亮；如 P0.4 输出 1，LED04 灯就会熄灭；
如 P0.5 输出 0，LED05 灯就会被点亮；如 P0.5 输出 1，LED05 灯就会熄灭；
如 P0.6 输出 0，LED06 灯就会被点亮；如 P0.6 输出 1，LED06 灯就会熄灭；
如 P0.7 输出 0，LED07 灯就会被点亮；如 P0.7 输出 1，LED07 灯就会熄灭；

如 P0 口对外全部送 0，则这 8 个灯全亮；
如 P0 口对外全部送 1，则这 8 个灯全灭；
如果 P4.0 输出 1，则 PNP 三极管不导通，不管 P0 口对外全部送 0 或送 1，这 8 个灯都不会有电流流过，都不会亮。P4.0 控制 PNP 三极管，作为这 8 个 LED 指示灯的总电源开关。

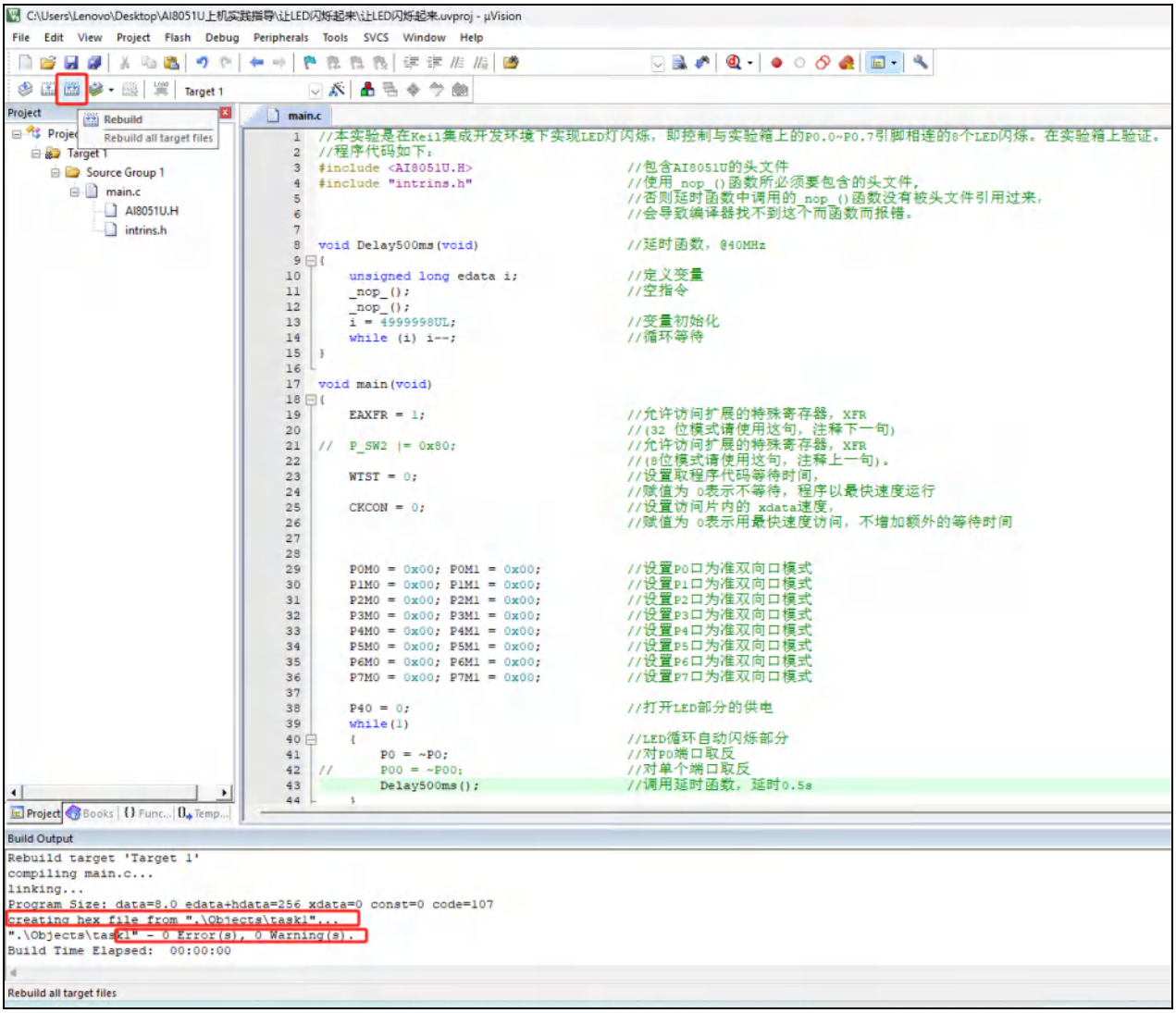
2.16.2　代码部分

要想实现“让 LED 灯闪烁起来”，则在上面的工程项目的 main.c 文件中可输入如下代码：
本实验是在 Keil 集成开发环境下实现 LED 灯闪烁，即控制与实验箱上的 P0.0~P0.7 引脚相连的 8 个 LED 闪烁。在实验箱上验证。

```
#include <AI8051U.H>                //包含 AI8051U 的头文件
#include "intrins.h"                //使用_nop_()函数所必须要包含的头文件,
                                    //否则延时函数中调用的_nop_()函数没有被头文件引用过来，
                                    //会导致编译器找不到这个而函数而报错。

void Delay500ms(void)               //延时函数，主频 40MHz
{
    unsigned long edata i;          //定义变量
    _nop_();                        //空指令
    _nop_();
    i = 4999998UL;                  //变量初始化
    while (i) i--;                  //循环等待
}
//上面延时函数部分代码可使用 AIapp-ISP 软件的“软件延时计算器”工具来生成，见本节后面的截图。

void main(void)
{
    EAXFR = 1;                      //允许访问扩展的特殊寄存器，XFR
                                    //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                  //允许访问扩展的特殊寄存器，XFR
                                    //(8 位模式请使用这句，注释上一句)。
    WTST = 0;                       //设置取程序代码等待时间，
                                    //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                      //设置访问片内的 xdata 速度，
                                    //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00; P0M1 = 0x00;       //设置 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x00;       //设置 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;       //设置 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x00;       //设置 P3 口为准双向口模式
    P4M0 = 0x00; P4M1 = 0x00;       //设置 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;       //设置 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;       //设置 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;       //设置 P7 口为准双向口模式
//上面的将所有 I/O 口都初始化为准双向口工作模式的程序，
//可以使用 AIapp-ISP 软件的“I/O 口配置工具”来生成，见本节后面的截图。

    P40 = 0;                        //打开 LED 部分的供电
    while(1)
    {                               //LED 循环自动闪烁部分
```

```
        P0 = ~P0;                   //对 P0 端口取反
//      P00 = ~P00;                 //对单个端口取反
        Delay500ms();               //调用延时函数，延时 0.5s
    }
}
```

编写完成代码后，通过工具栏上的 Rebuild 按钮进行编译：



编译完成后，会在底部的 Build Output 生成编译信息
出现 creating hex file 和 0 Error(s) 0 Warning（s）即为编译成功，无报错且成功创建了 hex 文件。
此 hex 文件，一般是在对应工程目录下的 Objects 文件夹里。

【小技巧--灵活运用 AIapp-ISP 软件的工具自动生成代码】

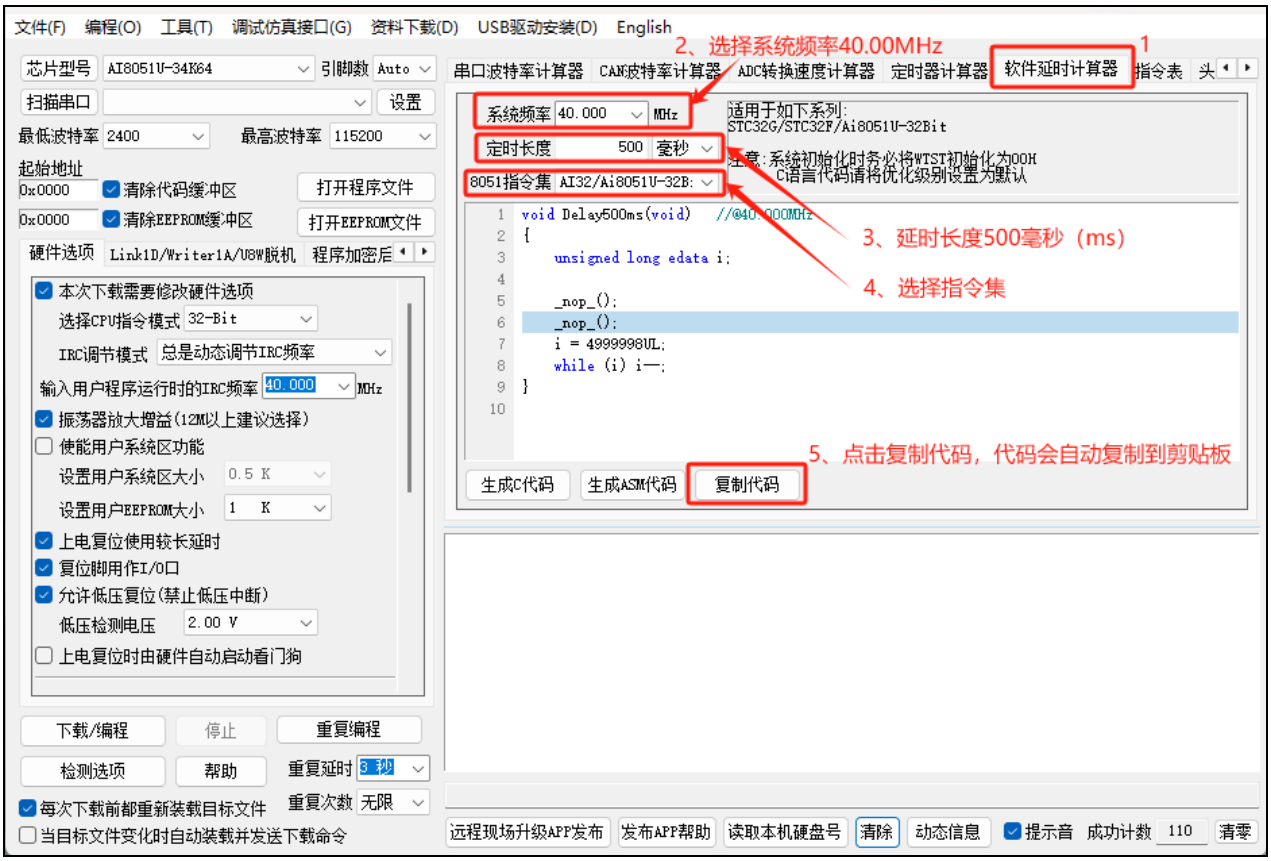
延时函数部分代码就可以使用 **AIapp-ISP** 软件的“**软件延时计算器**”来生成。

例如上述代码中这一段延时函数的代码：

```
void Delay500ms(void)               //延时函数，主频@40MHz
{
    unsigned long edata i;          //定义变量
```

```
    _nop_();                    //空指令
    _nop_();
    i = 4999998UL;              //变量初始化
    while (i) i--;              //循环等待
}
```

就可以使用 AIapp-ISP 软件的“软件延时计算器”来生成：



将复制出来的内容直接粘贴到代码文件中，不用手动输入。

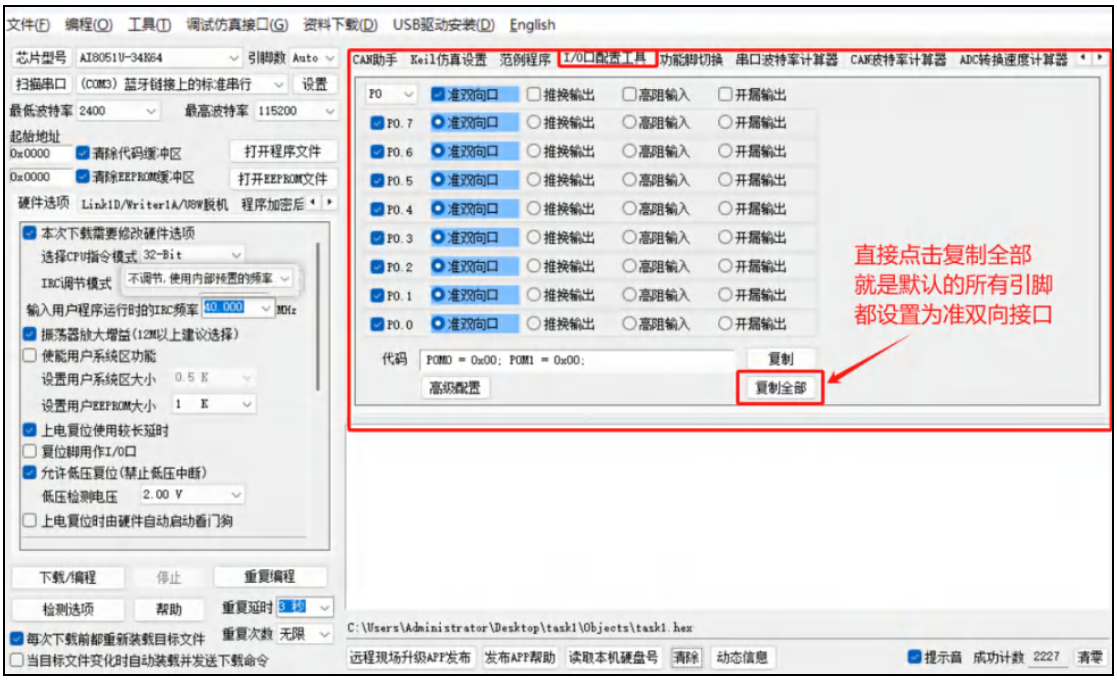
【小技巧--灵活运用 AIapp-ISP 软件的工具自动生成代码】

I/O 口的初始化部分就可以使用 **AIapp-ISP** 软件的“**I/O 口配置工具**”来生成。

例如上述代码中这一段将所有 I/O 口设置为准双向口模式的代码：

```
P0M0 = 0x00; P0M1 = 0x00;
P1M0 = 0x00; P1M1 = 0x00;
P2M0 = 0x00; P2M1 = 0x00;
P3M0 = 0x00; P3M1 = 0x00;
P4M0 = 0x00; P4M1 = 0x00;
P5M0 = 0x00; P5M1 = 0x00;
P6M0 = 0x00; P6M1 = 0x00;
P7M0 = 0x00; P7M1 = 0x00;
```

就可以使用 AIapp-ISP 的 I/O 口配置工具进行生成：

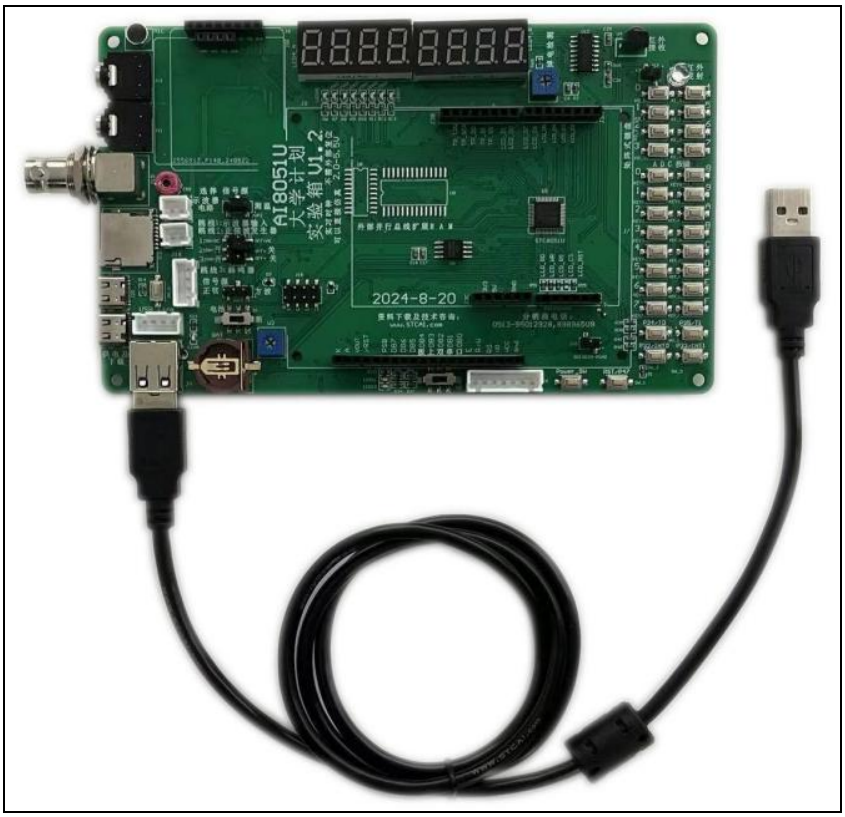


将复制出来的内容直接粘贴到代码文件中，不用手动输入。

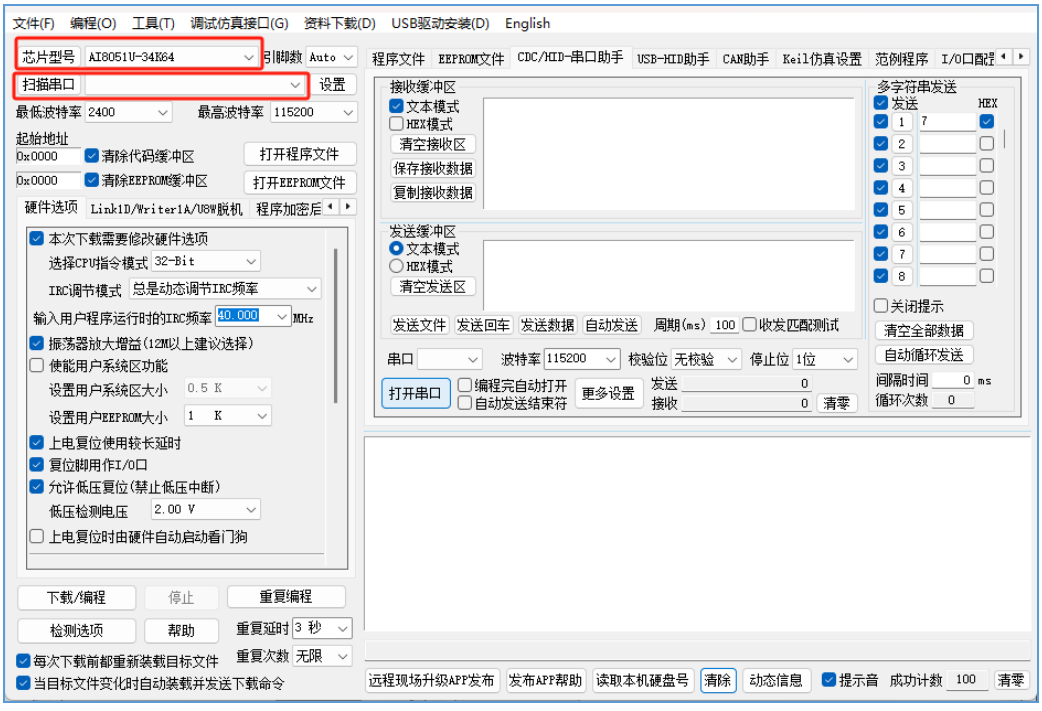
2.16.3　下载用户程序到实验箱并查看实验现象，实物图

2.16.3.1　连接实验箱和电脑

拿出实验箱中配置的 USB-TypeA 到 USB-TypeA 线将实验箱与电脑正确连接。

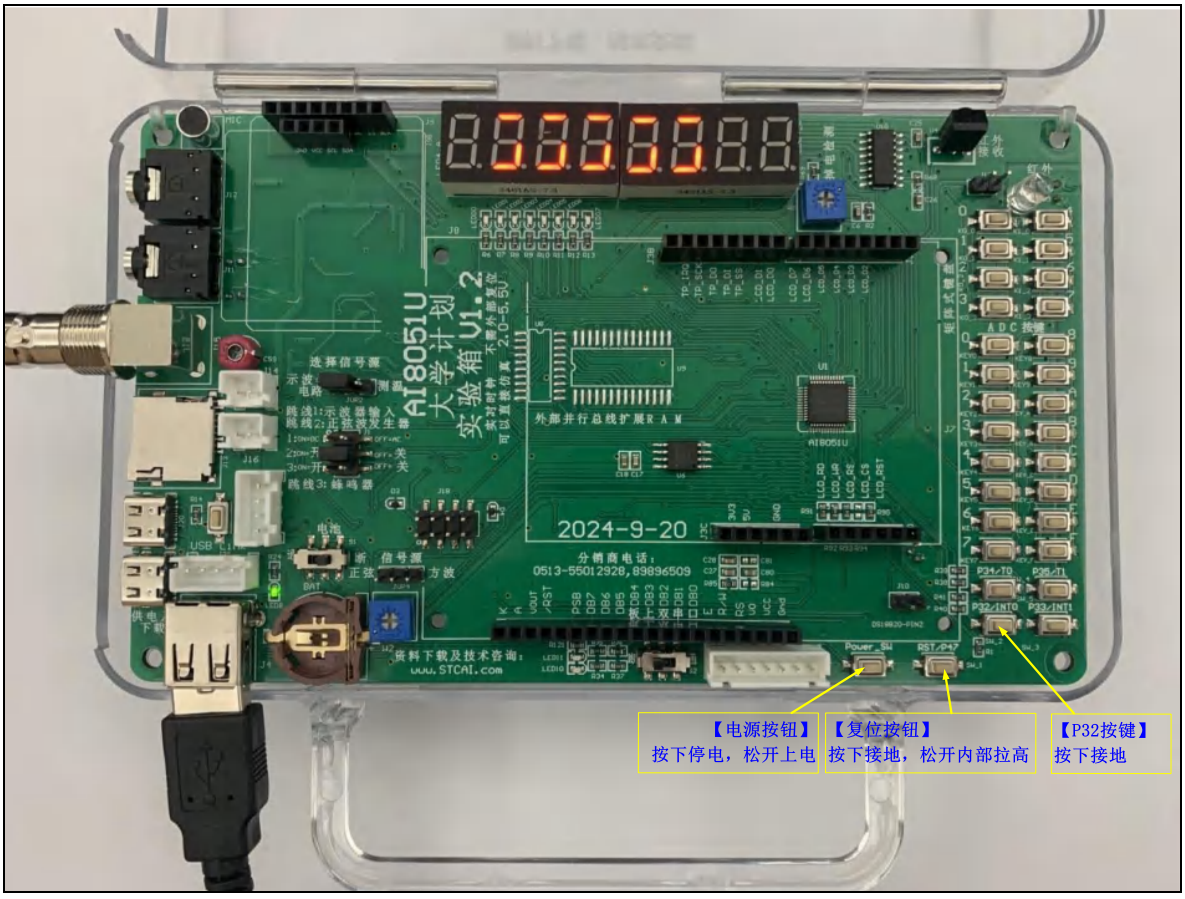


然后打开 AIapp-ISP-v6.95F 或以上版本。选择单片机型号：Ai8051U-34K64，如下图，我们看到“扫描串口”后面的端口列表框不是“（HID1）USB-Writer”，说明还不能下载程序到实验箱。

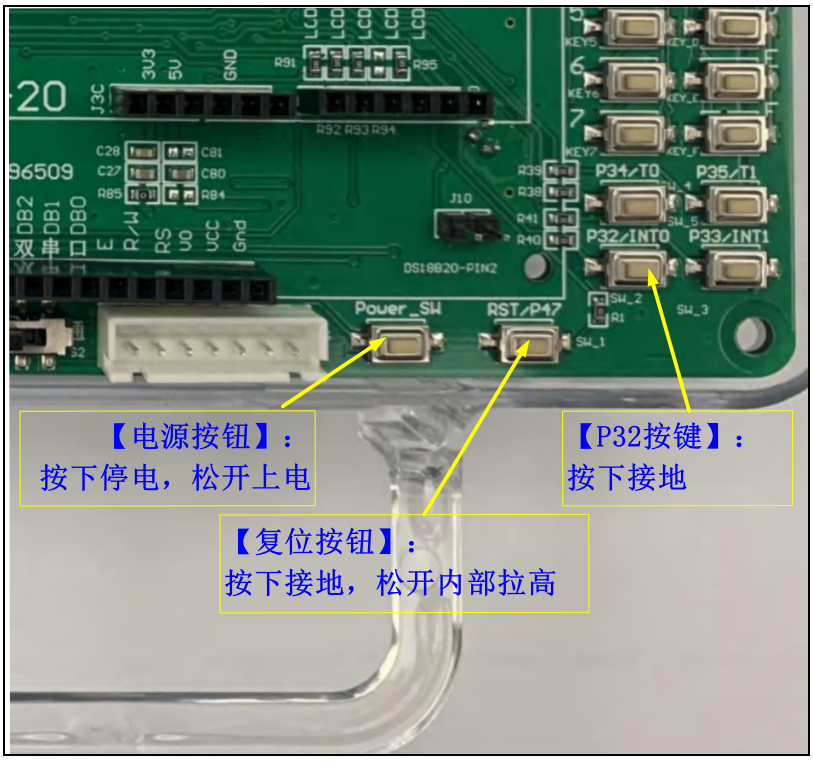


2.16.3.2 硬件 USB 下载用户程序步骤

下图是实验箱已经和电脑相连的正面图片，数码管是随机显示的，数码管下面的 8 个 LED 灯都是暗的，**此实验的目的就是让这 8 个灯闪烁起来。**



局部放大实验箱图片，其正下方有两排按键，找到 P3.2 按键和 Power_SW 按键



【方法一，P3.2 按键，再结合停电上电下载】

1、按下板子上的 P3.2 按键，就是 P3.2 接地

2、给目标芯片重新上电，不管之前是否已通电。

===本 Ai8051U 实验箱是：按下电源按钮停电，再松开电源按钮上电，

这时电脑端 ISP 下载软件会自动识别出“(HID1) USB Writer”，此时就与 P3.2 状态无关了，可以松开 P3.2 按键（P3.2 在用户程序区可做其它任意用途）

3、点击电脑端下载软件中的【下载/编程】按钮（注意：USB 下载与串口下载的操作顺序不同）

电脑端 ISP 下载软件提示：下载进行中，几秒钟后，提示下载成功！

【方法二：复位管脚低电平复位下载】

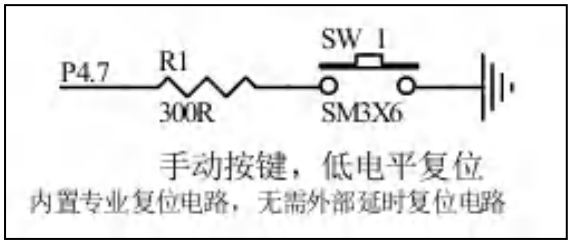
USB 连接好并已上电的情况下，外部按键复位也可进入 USB 下载模式，注意：

P4.7-nRST 出厂时默认是 P4.7-I/O 功能，要改为复位功能，需 ISP 烧录时取消设置复位脚用作 I/O 口，停电一次再上电才生效，程序区中用户程序也可改为复位脚或 I/O，这个立即生效。

1、按下复位管脚/P4.7-nRST 外接的低电平复位按键，复位 MCU，

再松开复位键，MCU 首先从系统程序区启动运行系统程序，判断是否要下载用户程序，

等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后

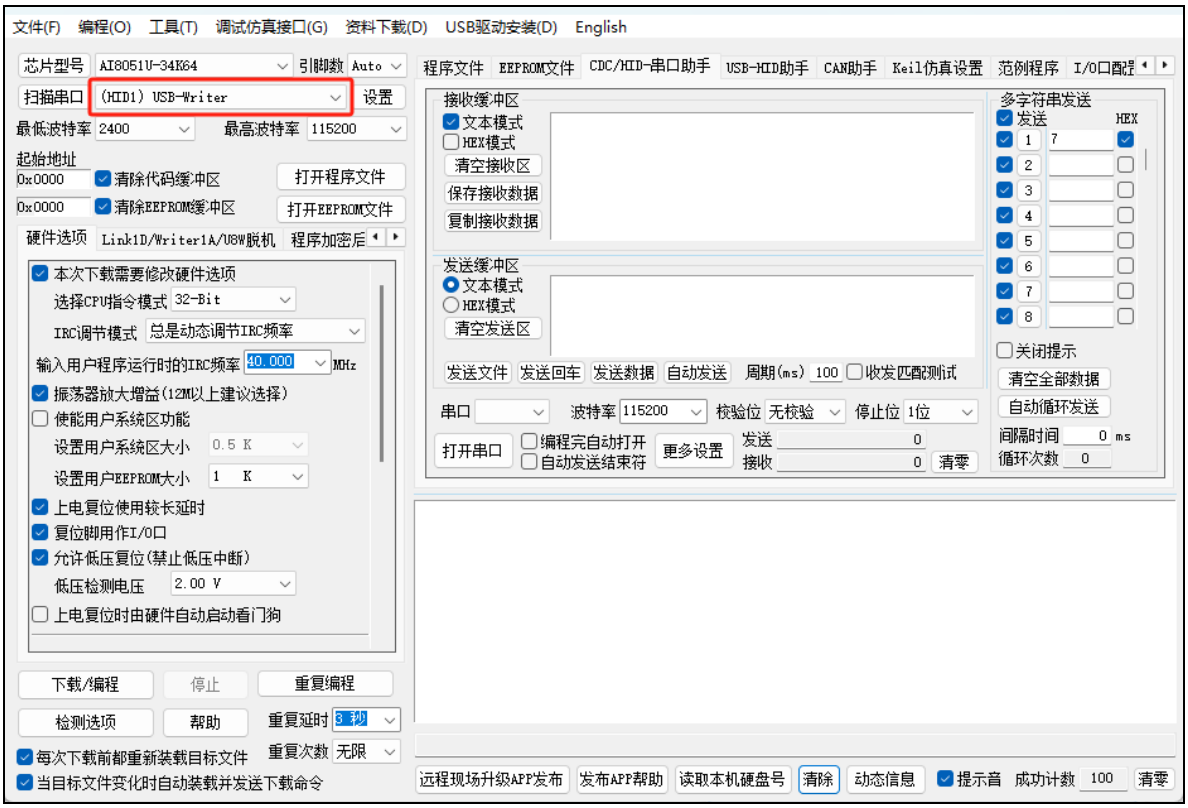


2、点击电脑端下载软件中的【下载/编程】按钮

电脑端 ISP 下载软件提示：下载进行中…，下载成功！

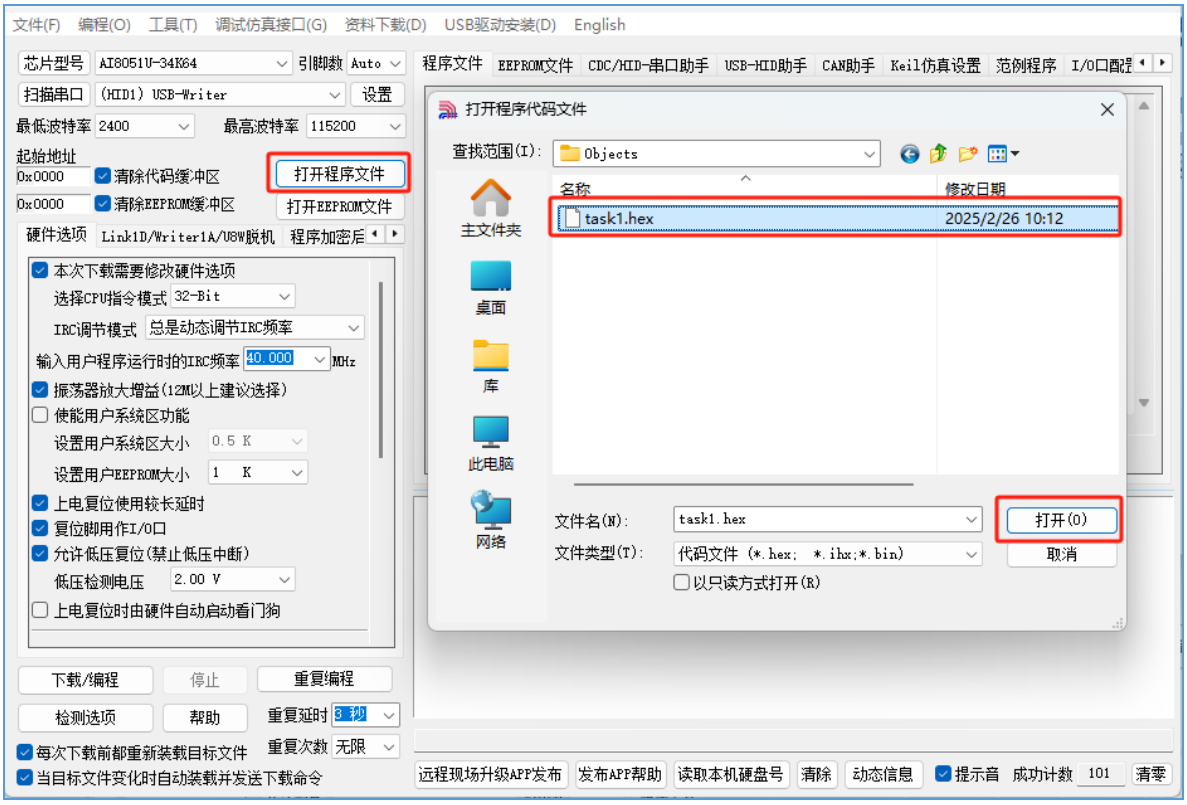
备注：如果目标 MCU 的用户程序区已有 USB 通讯程序（含 CDC、HID 等），则通过复位脚复位进入 USB 下载模式不被支持，只能通过停电上电冷启动下载或者从用户程序区软复位到系统区下载

如下图：AIapp-ISP 软件界面的“扫描串口”后面的端口列表框内显示：（HID1）USB-Writer

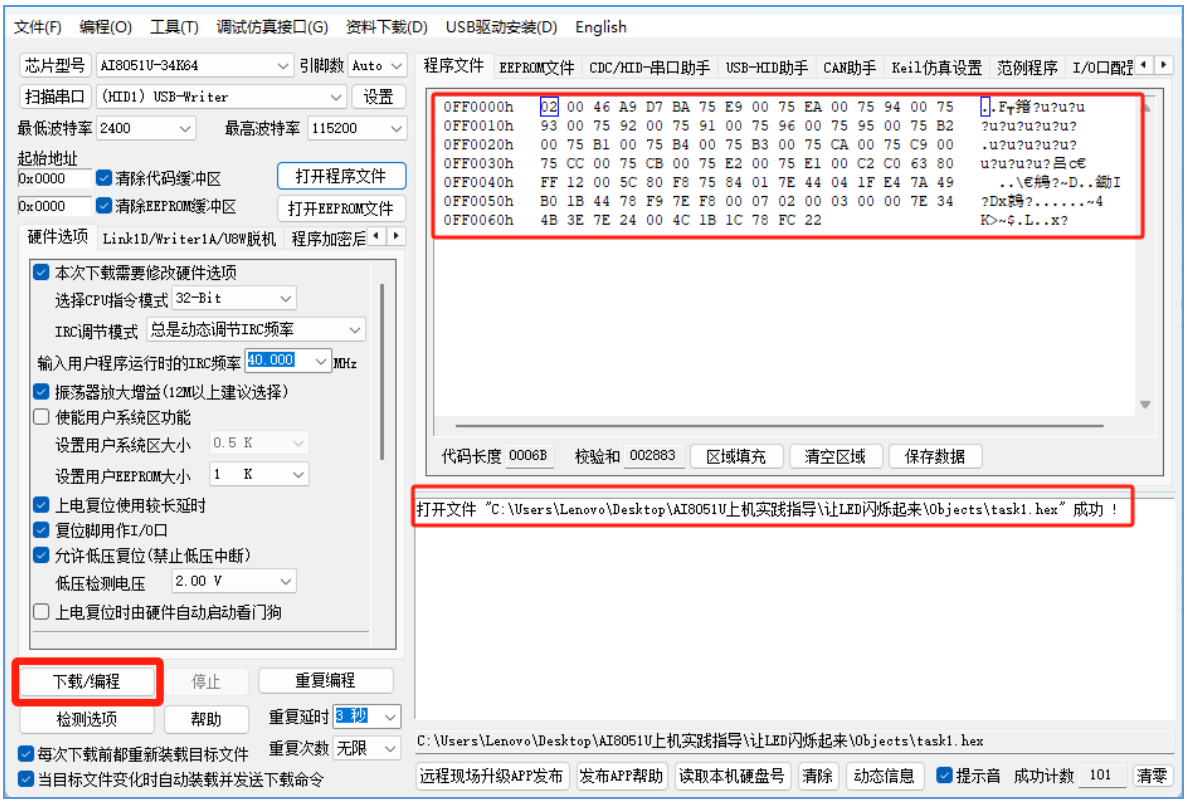


2.16.3.3 下载用户程序文件并观察实验效果

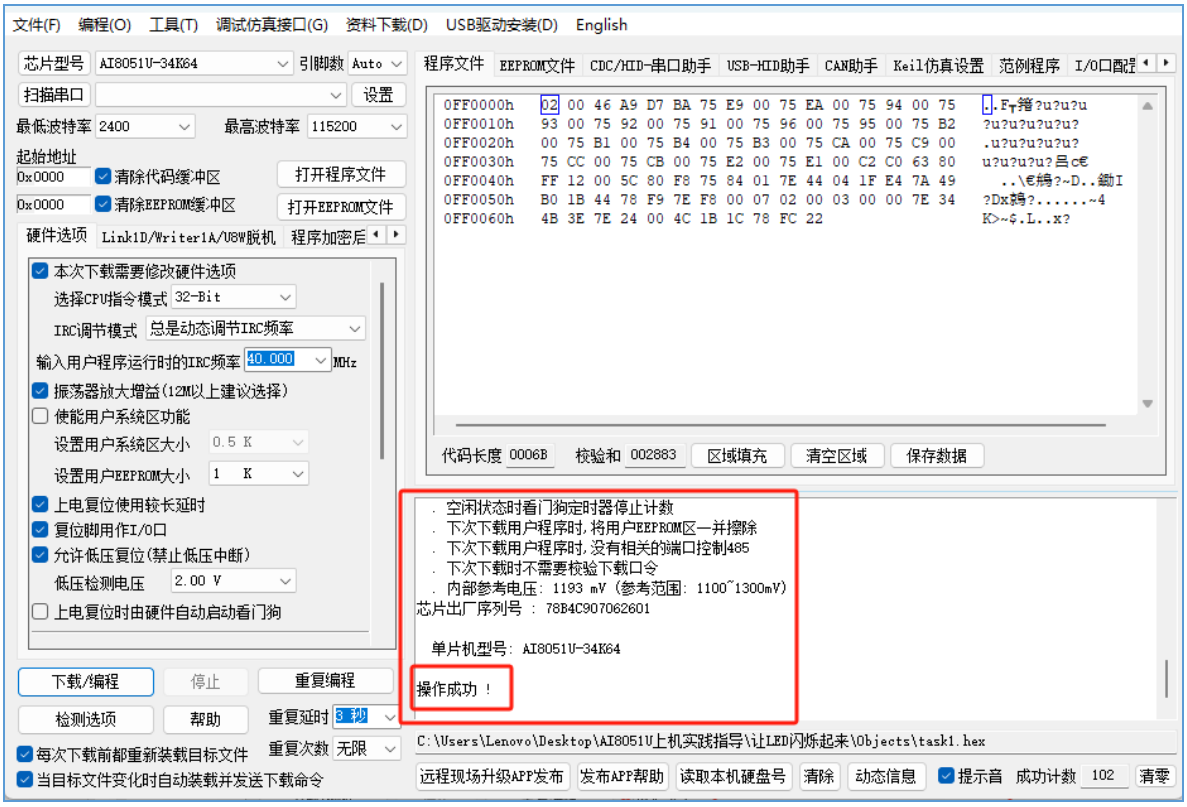
现在就打开用户程序文件，点击 AIapp-ISP 软件界面上的“打开程序文件”（打开的是项目程序文件编译无误后生成的 HEX 文件，一般默认存放在该项目的 Objects 目录中），加载在前面小节编译好的 HEX 文件：



然后，AIapp-ISP 软件界面如下图所示，出现“‘打开文件***.hex’成功!”



点击“下载/编程”，把 task1.hex 烧录到实验箱，AIapp-ISP 软件界面如下图所示，出现：操作成功！恭喜你！



这时，再看实验箱，刚刚暗淡的 LED 灯，齐齐一亮一灭闪烁起来了！至此，实验成功！

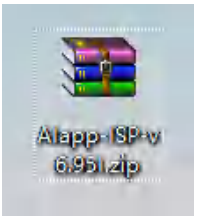
2.16.4　使用【AiCube】项目创建助手工具创建并实现 LED 灯闪烁

STCAI 在最新版的 AIapp-ISP 软件中，推出【AiCube】项目创建助手工具，目前支持 Ai8051U-32Bit 各功能模块的智能化配置，并生成程序框架。这项功能在程序员建立项目时，简化了项目的初始化难度，提高效率，避免因不太熟悉或不完全了解该单片机的性能而忽略个别配置，引起编译错误等问题。尤其对刚涉足单片机的初学者，此工具不仅引导其了解单片机编程流程，还极大的降低了入门门槛，缩短了摸索的时间，提高入门的兴趣和信心。现在我们使用这个工具来生成实现此实验功能的配置和代码。

a) 安装【AIapp-ISP】下载/编程/烧录/软件

安装 STCAI 公司研发的 AIapp-ISP 下载 / 编程 / 烧录 / 软件，含强大的辅助开发工具包。登录 STCAI 官网，下载最新版的 AIapp-ISP 安装包，下载链接如下：https://www.stcai.com/gjrj

下载下来的是压缩包文件如下图：

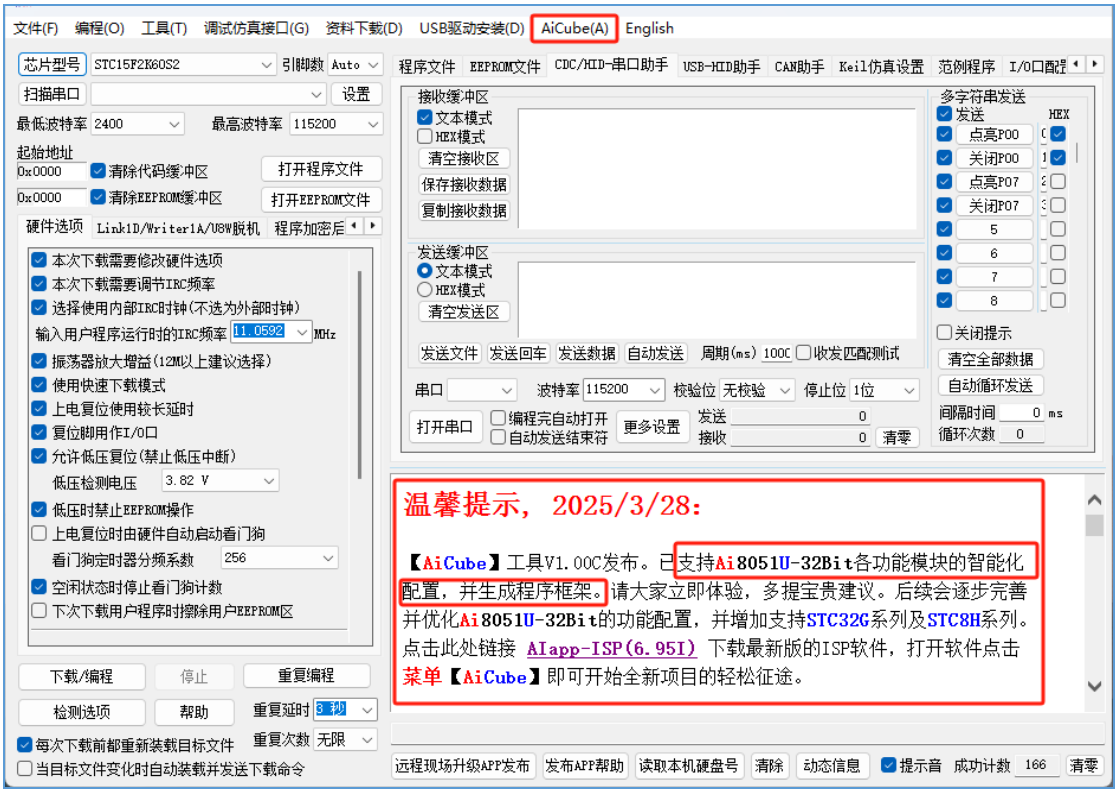


双击后解压缩如下图：

如果是解压缩到非桌面的其他目录，如 D:\STCAI-ISP，建议创建快捷方式到桌面方便打开。这实际是一个免安装的绿色软件。

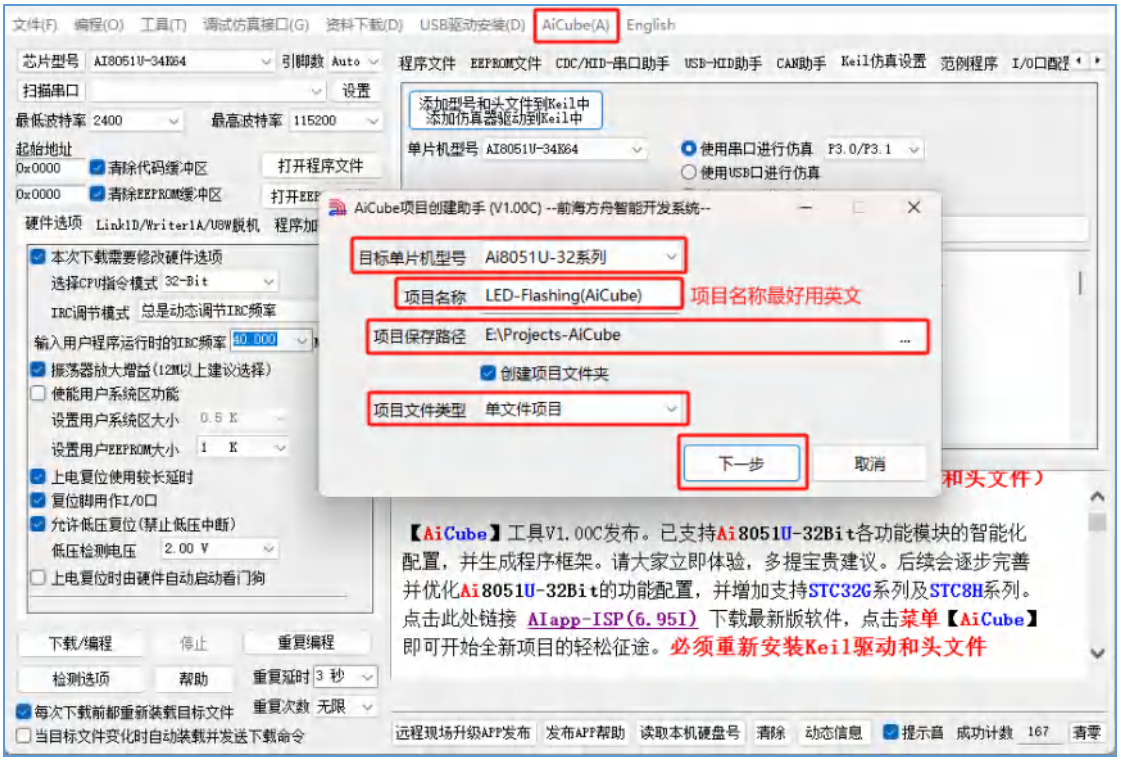
b) 打开【AiCube】助手

双击图标，即可打开 AIapp-ISP 软件界面，即可直接使用，如下图：

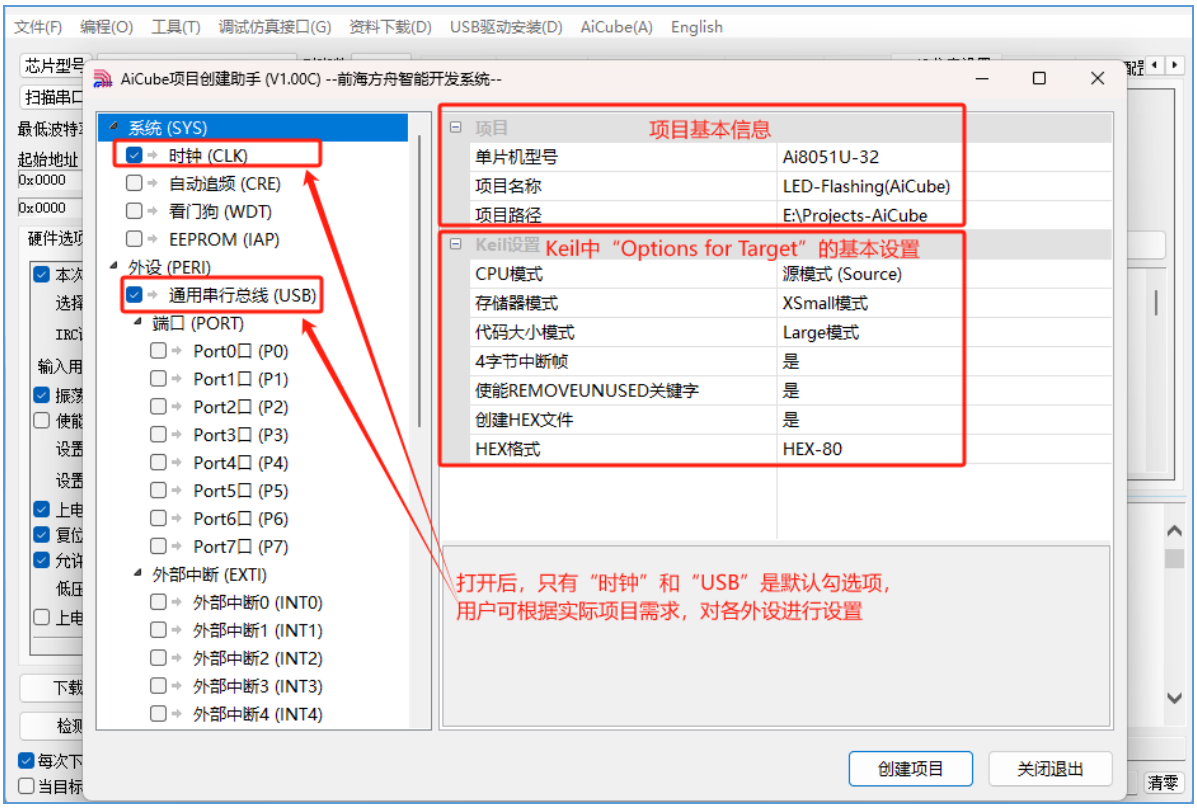


c) 使用【AiCube】助手创建项目

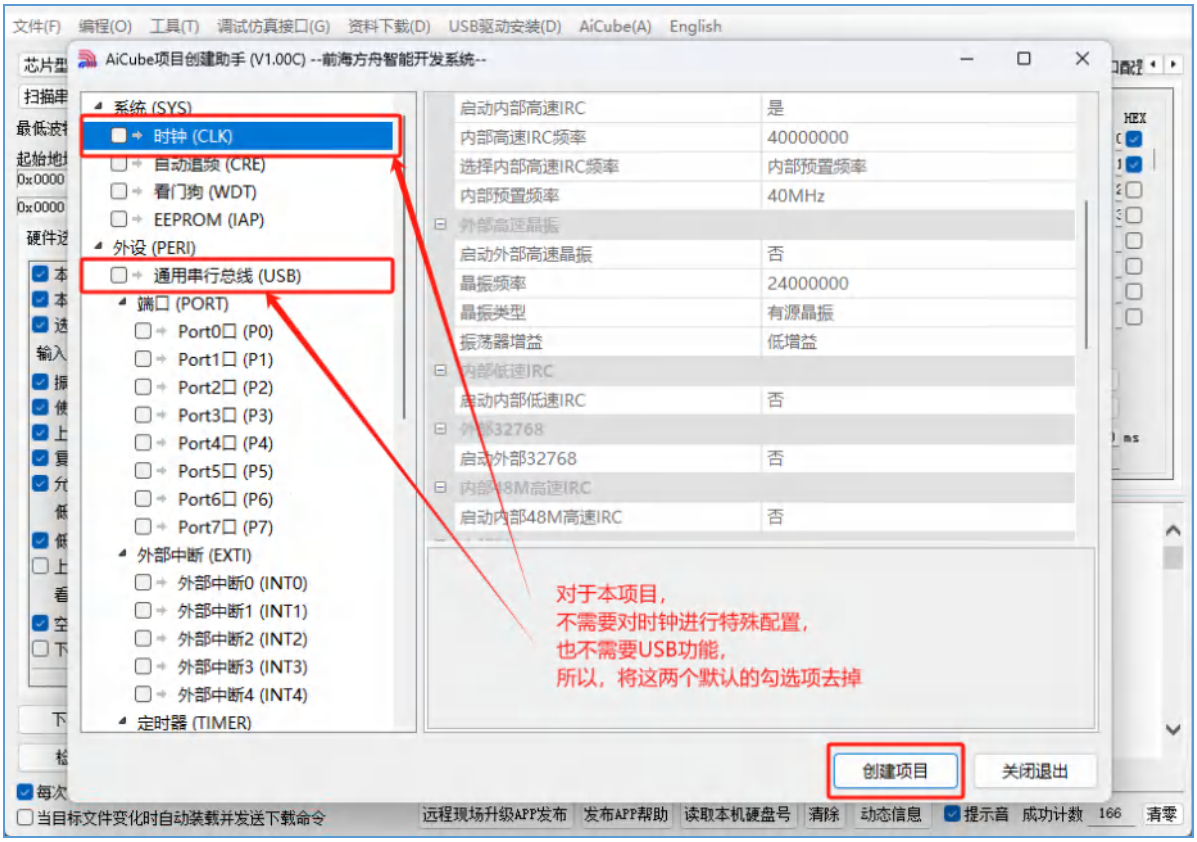
点击【AiCube（A）】，弹出如下窗口：



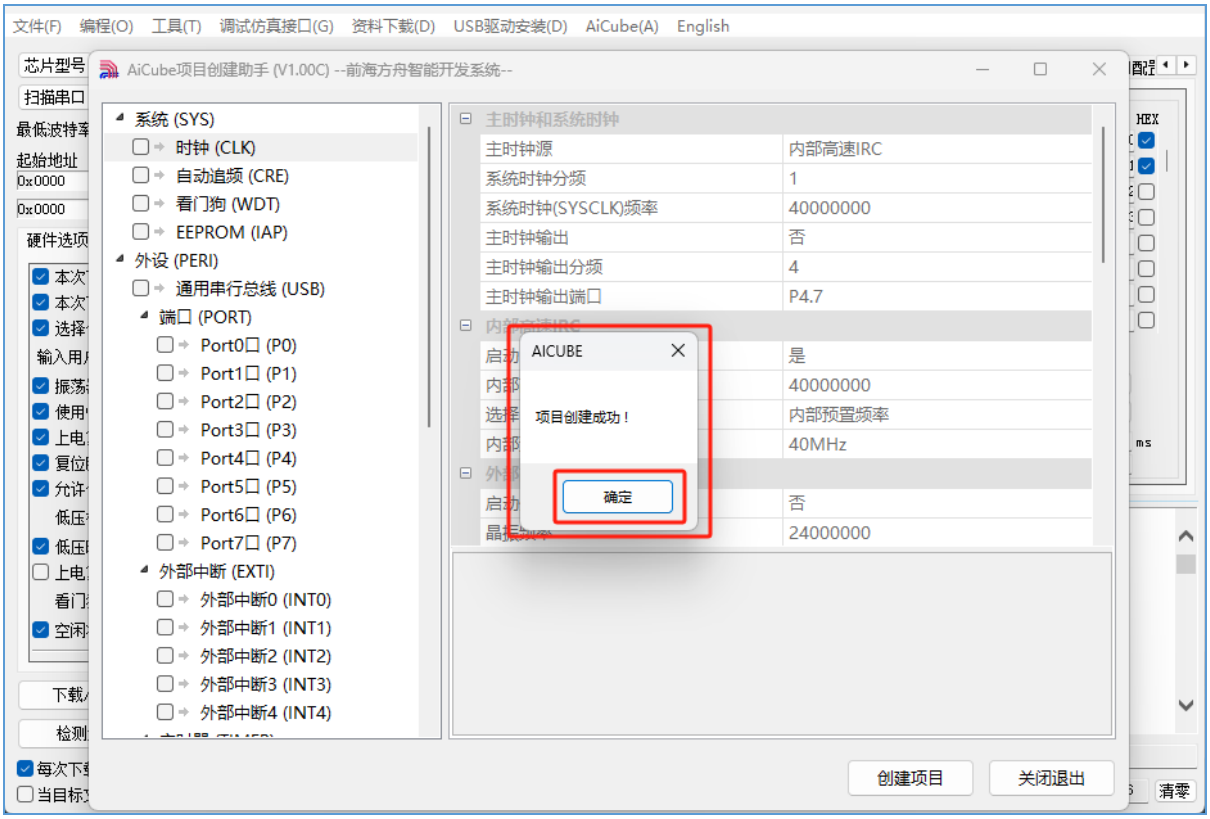
选择并填写项目基本信息，建议项目名称用英文！点击“下一步”，出现如下配置界面：



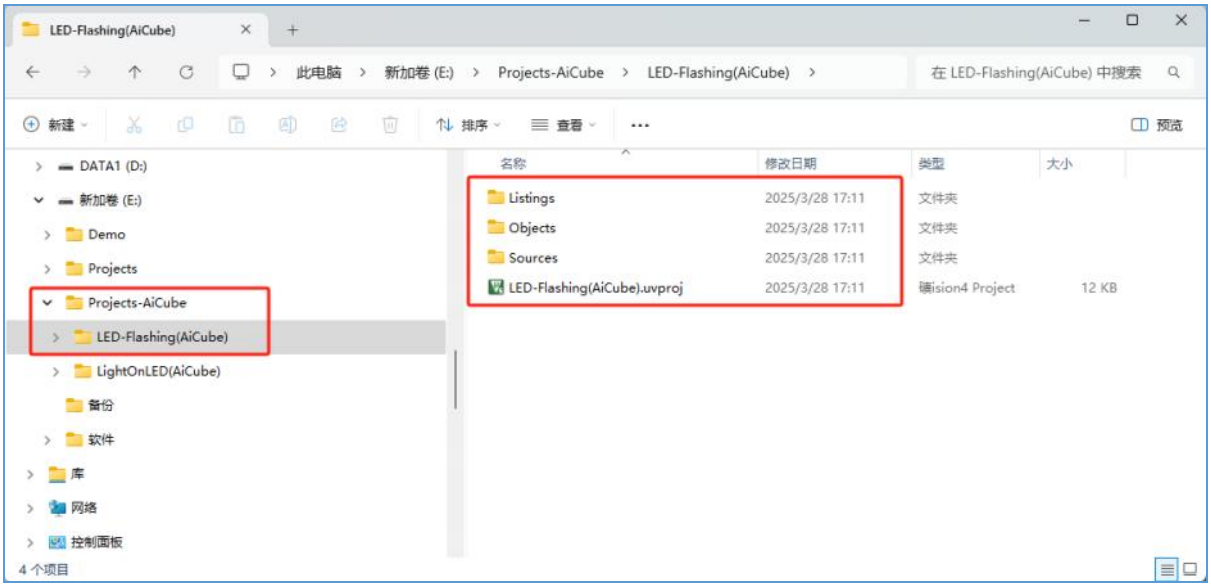
根据本项目的实际需求，对配置进行如下修改：



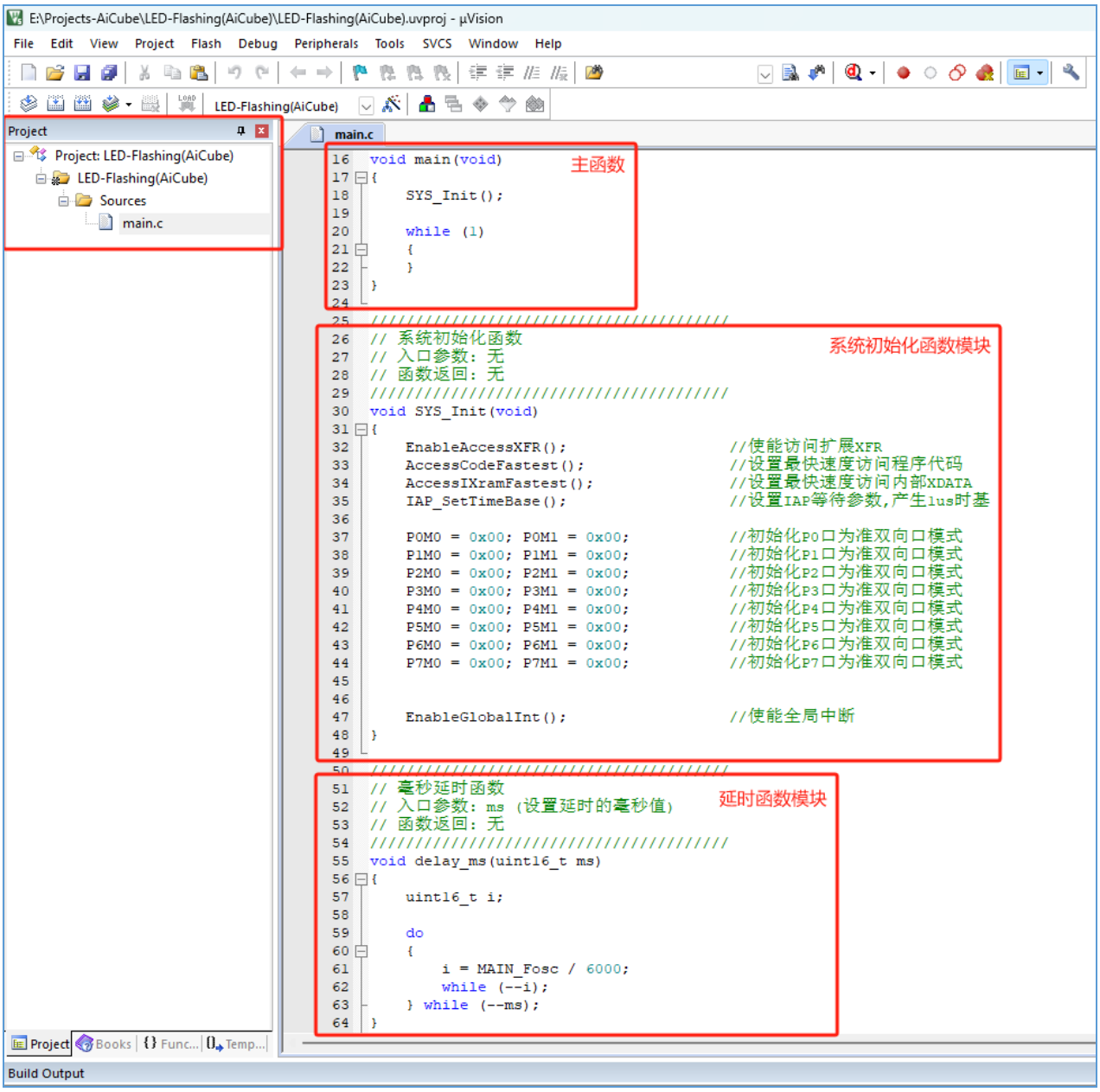
点击“创建项目”，【AiCube】助手提示：“项目创建成功！”



点击“确定”，即成功创建了项目及项目文件目录“LED-Flashing(AiCube)”，找到并打开刚刚创建的项目目录。发现刚刚的操作过程，【AiCube】助手已经帮我们建立了项目目录及 LED-Flashing(AiCube).uvproj 文件，如下图：



双击打开项目：LED-Flashing(AiCube).uvproj，如下图：

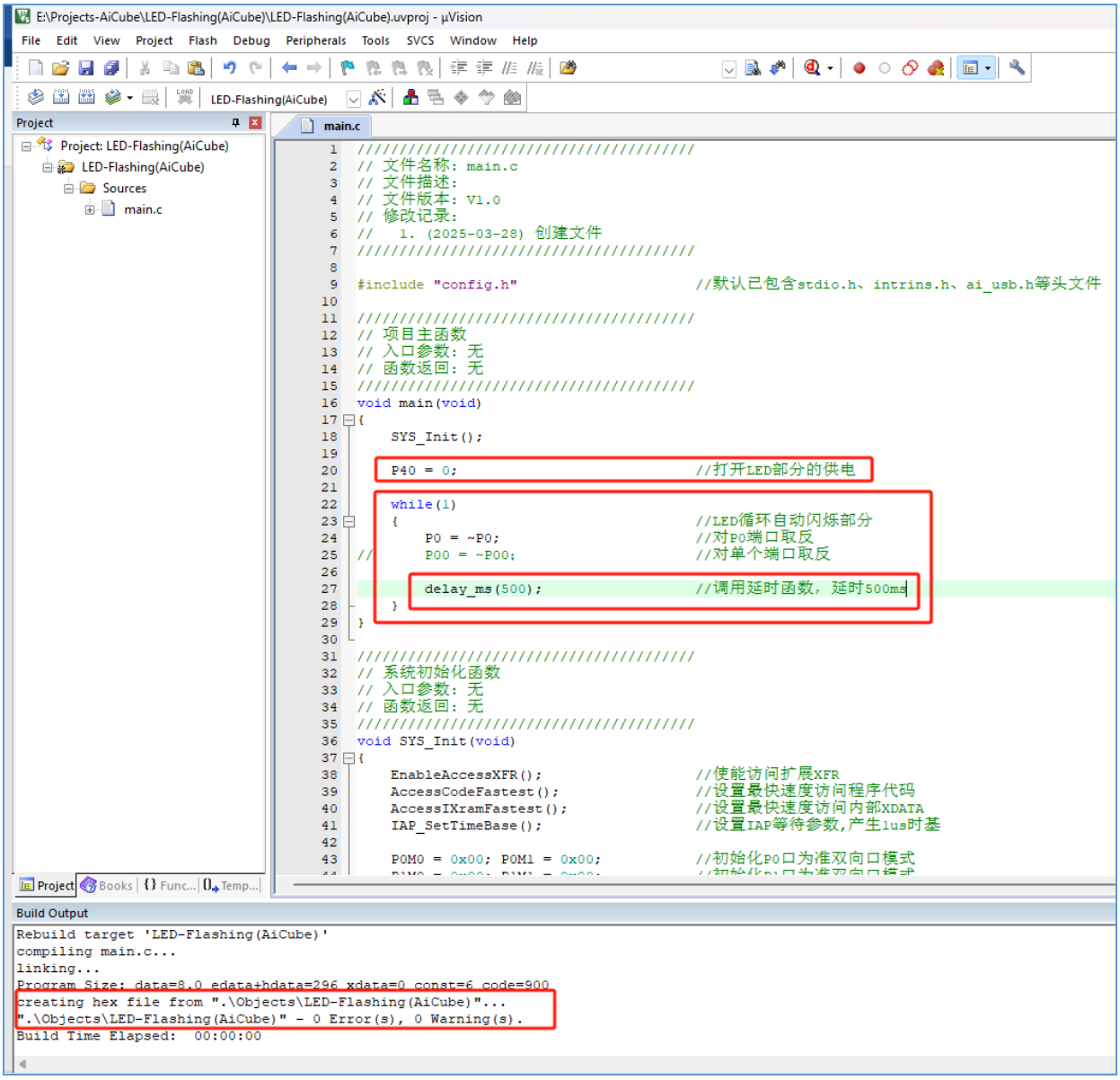



```c
void main(void)
{
    SYS_Init();

    while (1)
    {
    }
}

//////////////////////////////////////////
// 系统初始化函数
// 入口参数: 无
// 函数返回: 无
//////////////////////////////////////////
void SYS_Init(void)
{
    EnableAccessXFR();                  //使能访问扩展XFR
    AccessCodeFastest();                //设置最快速度访问程序代码
    AccessIXramFastest();               //设置最快速度访问内部XDATA
    IAP_SetTimeBase();                  //设置IAP等待参数,产生1us时基

    P0M0 = 0x00; P0M1 = 0x00;           //初始化P0口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x00;           //初始化P1口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;           //初始化P2口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x00;           //初始化P3口为准双向口模式
    P4M0 = 0x00; P4M1 = 0x00;           //初始化P4口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;           //初始化P5口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;           //初始化P6口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;           //初始化P7口为准双向口模式


    EnableGlobalInt();                  //使能全局中断
}

//////////////////////////////////////////
// 毫秒延时函数
// 入口参数: ms (设置延时的毫秒值)
// 函数返回: 无
//////////////////////////////////////////
void delay_ms(uint16_t ms)
{
    uint16_t i;

    do
    {
        i = MAIN_Fosc / 6000;
        while (--i);
    } while (--ms);
}
```

由于这是一个非常简单的入门实验，我们发现【AiCube】助手已经帮我们在项目的 main.c 中按模块化设计，并已经包含了：必要的头文件、主函数、系统初始化函数、延时函数（根据主频自动计算的延时初值）。

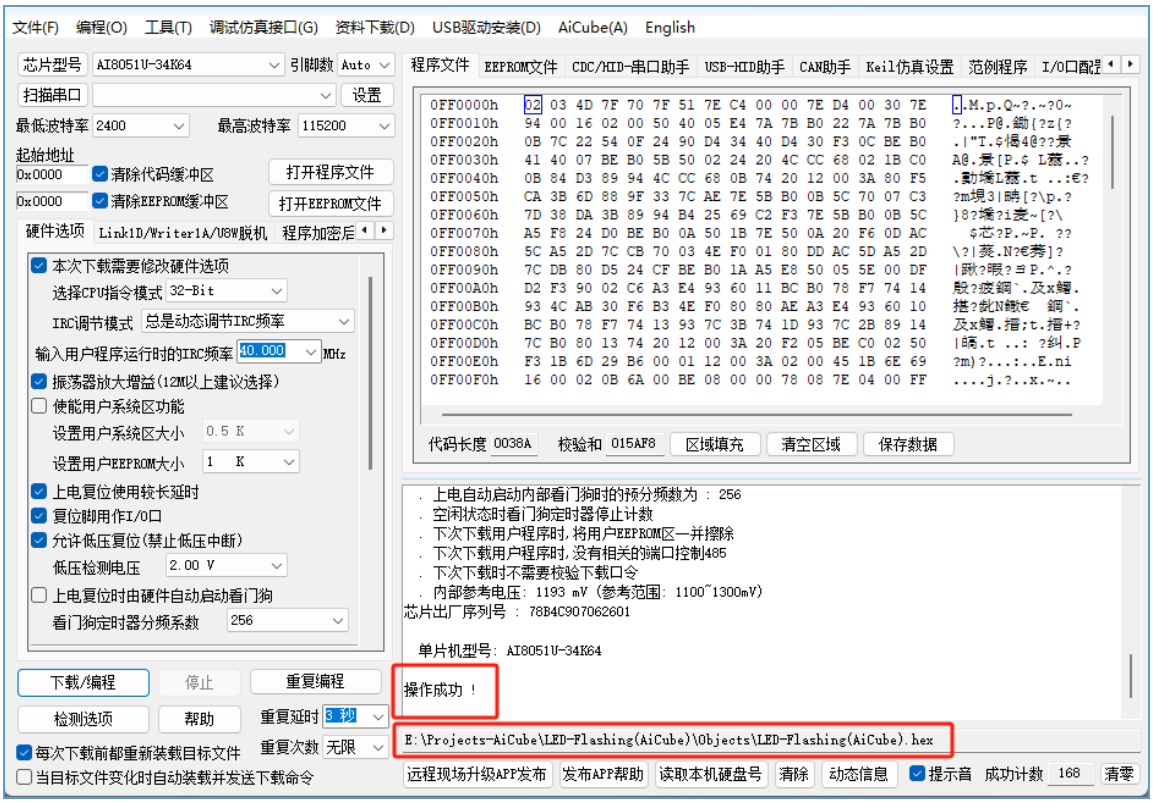
d) 增加实验项目代码

为了实现 LED 闪烁的功能，我们现在只需要在主函数中补充 LED 控制代码并调用延时函数（延时500ms），保存，编译，如下图：



e) 使用 Ai8051U 实验箱查验实验效果

连接好 Ai8051U 实验箱，运行 AIapp-ISP 软件系统，按步骤打开并下载程序“LED-Flashing(AiCube).hex”，如下图：



Ai8051U 实验箱上的 LED 灯齐齐闪烁起来：

有了【AiCube】助手，单片机编程就是这么简单！快用起来吧！

2.16.5　USB 不停电下载，ISP 软件如何设置

在开发过程中，程序经常需要反复修改，重新编译后生成新的 HEX 文件，就需要停电/上电进入下载模式，很不方便。其实可以利用我们提供的 USB 不停电 ISP 下载功能，进行自动下载。

在有 USB 功能的实验中，都要加载 USB 库函数，并对 USB 进行初始化。由于在 USB 库函数的代码中已经定义了不停电 ISP 下载命令"@STCISP#"，只需要在下载软件的"收到用户命令后复位到 ISP 监控程序区"的选项卡中进行如下设置，即可实现【USB 不停电下载】功能。

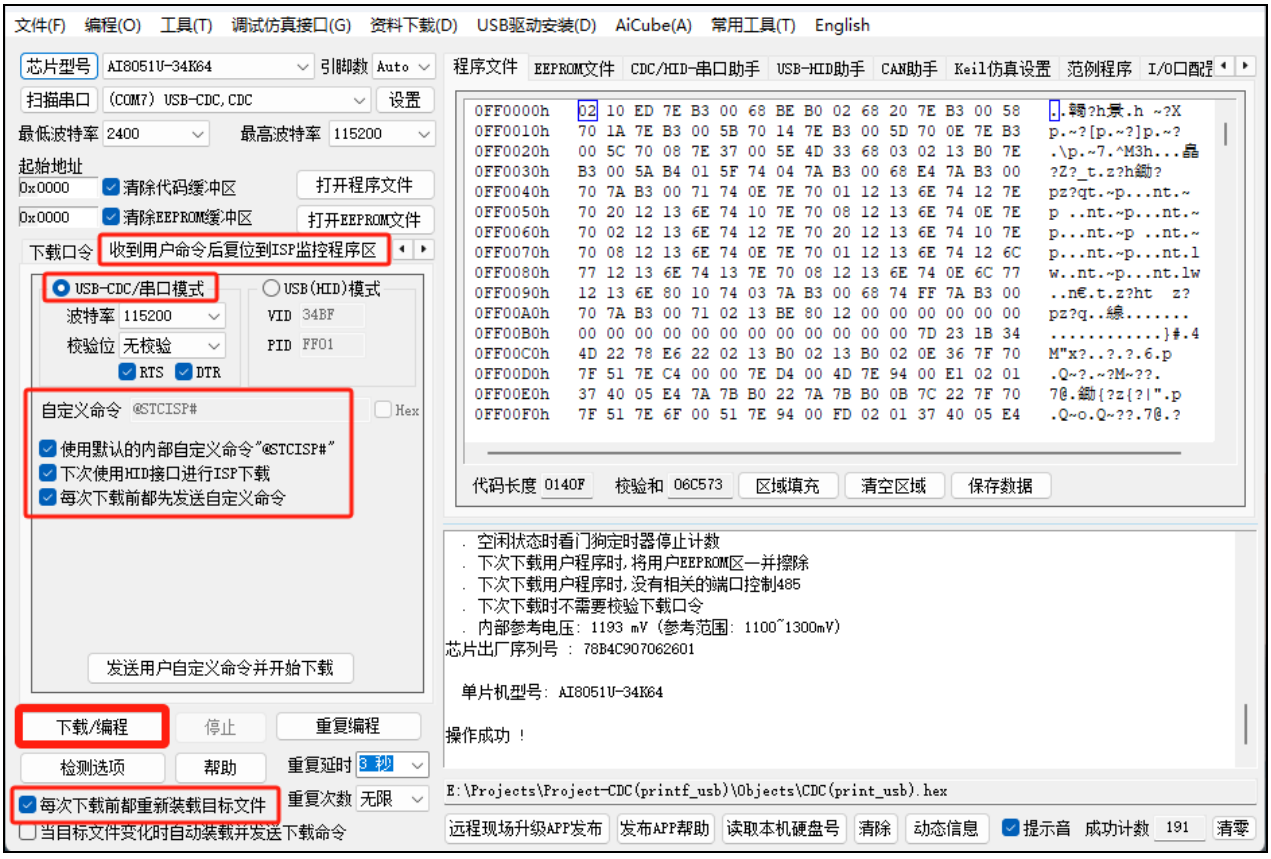
- 如使用 USB-CDC 库函数产生 USB-CDC 虚拟串口，可实现 USB-CDC 不停电自动 ISP 下载功能，并可忽略传统串口的波特率、校验位等参数的设置，下载时系统程序区还是使用的 USB-HID 通讯协议；
- 如果是传统的串口，需正确设置波特率、校验位等参数，才能实现串口的不停电自动 ISP 下载功能；
- 如果使用 USB-HID 库函数产生 USB-HID 设备，则选择"USB（HID）模式"就能实现不停电自动 ISP 下载功能，下载时系统程序区还是使用的 USB-HID 通讯协议。

ISP 软件如何设置：

2.16.5.1　手动点击【下载/编程】按钮，ISP 软件自动重新打开文件，不停电下载

如下图设置，就可以实现：

在点击"下载/编程"按钮后，ISP 软件会重新调入已打开的目标文件一次（不管目标文件有无变化），接着通过 USB-CDC 通讯协议自动发送自定义下载命令如"@STCISP#"给单片机，单片机中的用户软件收到该命令后，就会软复位到单片机的系统监控程序区，等待 ISP 下载，本设置还指定了电脑端软件后续是通过 USB-HID 通讯协议传输用户程序到目标单片机进行 ISP 下载。

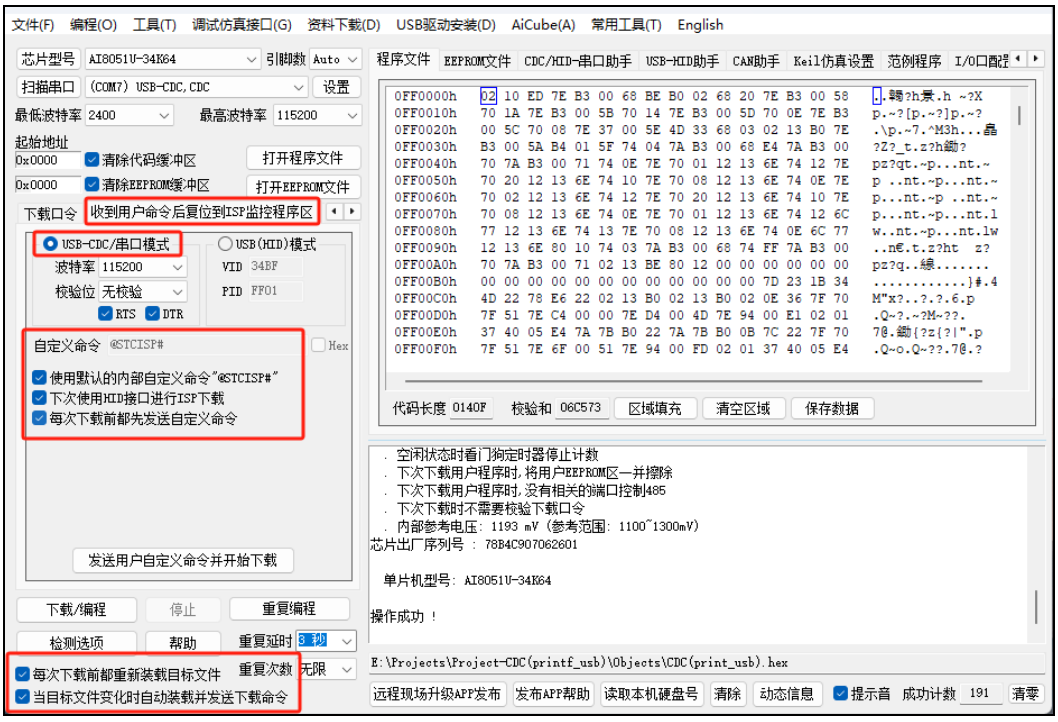


如上所示，不停电下载实验是成功的，但本设置还需要手工点击一次"下载/编程"按钮。

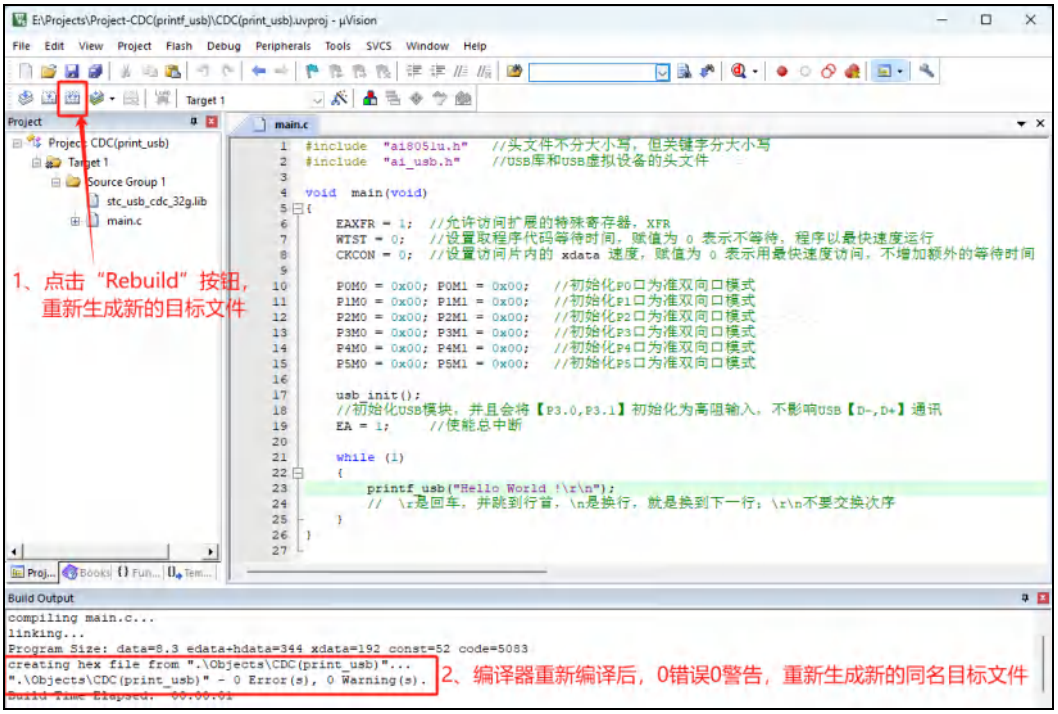
2.16.5.2 编译器重新编译后，ISP 软件会自动重新调入文件，进行不停电下载

如下图设置：

下载软件会每秒检测多次目标文件被编译的时间是否已更新，如时间已更新，则 ISP 软件会自动重新调入已打开的目标文件，接着通过 USB-CDC 通讯协议自动发送自定义下载命令如"@STCISP#"给单片机，单片机中的用户软件收到该命令后，就会软复位到单片机的系统监控程序区，等待 ISP 下载，本设置还指定了电脑端软件后续是通过 USB-HID 通讯协议传输用户程序到目标单片机进行 ISP 下载。



如上所示，不停电下载实验是成功的，但本设置的优点是不需要手工点击"下载/编程"按钮。

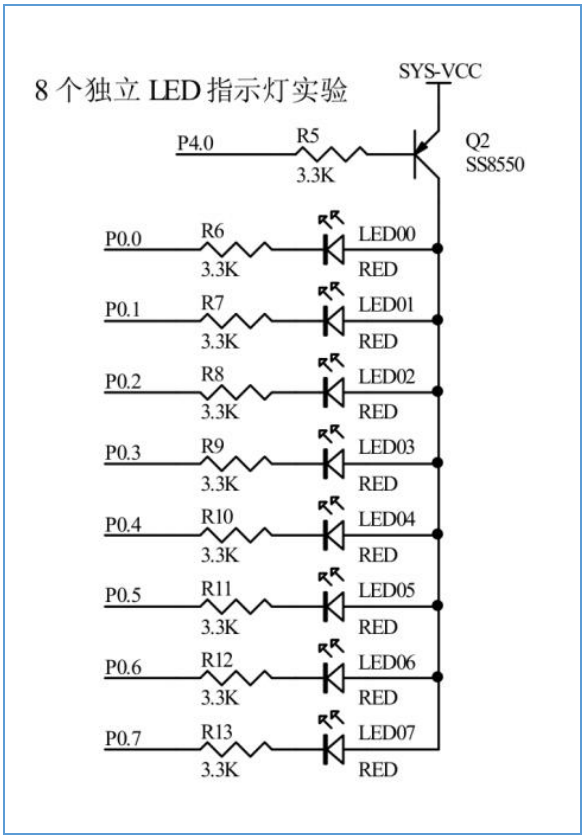


在编译集成环境中，点击"Rebuild"按钮，目标文件会被重新编译生成一次，ISP 软件发现目标文件被编译的时间发生变化后，ISP 软件就会执行前面描述的不停电下载动作。

2.17　点灯实验三：通过 USB-CDC 方式控制 Ai8051U 进行 LED 点灯

本实验相应的视频链接：https://www.stcaimcu.com/forum.php?mod=viewthread&tid=15176#lastpost
<Ai8051U 实验箱的全部原理图可以通过点击本段文字进行查看>

2.17.1　原理介绍，在 Ai8051U 实验箱中有如下的一排 LED 指示灯



I/O 口 P4.0 控制 PNP 三极管的通断，控制这 8 个灯的整体供电。
如果 P4.0 输出 1，则 PNP 三极管不导通，这 8 个灯都不会获得电源，都不会有电流流过，都不会被点亮。
如果 P4.0 输出 0，则 PNP 三极管导通，这 8 个灯才有可能会流过电流，才有可能会被点亮。
如 P4.0 输出 0，PNP 三极管导通，特定 LED 灯的控制 I/O 口如输出 0，则该 LED 灯会有电流流过，会被点亮。
如 P0.0 输出 0，LED00 灯就会被点亮；如 P0.0 输出 1，LED00 灯就会熄灭；
如 P0.1 输出 0，LED01 灯就会被点亮；如 P0.1 输出 1，LED01 灯就会熄灭；
如 P0.2 输出 0，LED02 灯就会被点亮；如 P0.2 输出 1，LED02 灯就会熄灭；
如 P0.3 输出 0，LED03 灯就会被点亮；如 P0.3 输出 1，LED03 灯就会熄灭；
如 P0.4 输出 0，LED04 灯就会被点亮；如 P0.4 输出 1，LED04 灯就会熄灭；
如 P0.5 输出 0，LED05 灯就会被点亮；如 P0.5 输出 1，LED05 灯就会熄灭；
如 P0.6 输出 0，LED06 灯就会被点亮；如 P0.6 输出 1，LED06 灯就会熄灭；
如 P0.7 输出 0，LED07 灯就会被点亮；如 P0.7 输出 1，LED07 灯就会熄灭；

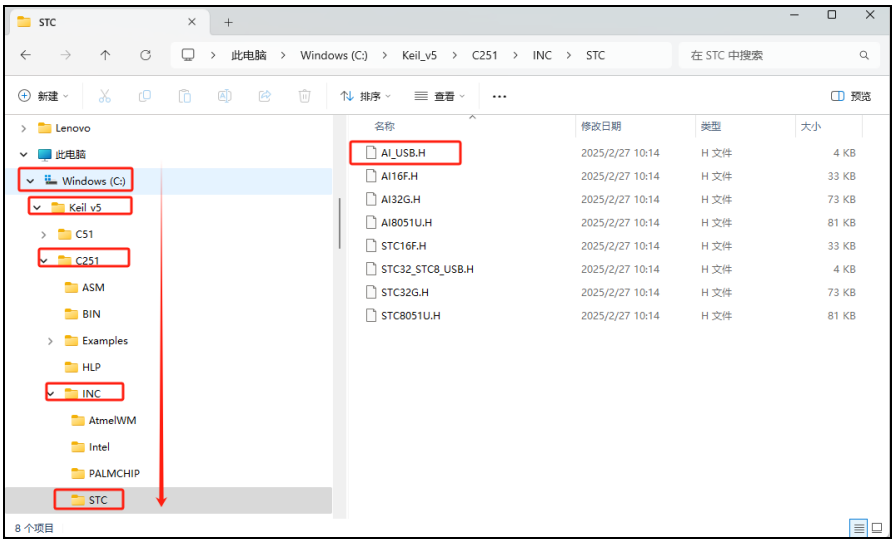
如 P0 口对外全部送 0，则这 8 个灯全亮；
如 P0 口对外全部送 1，则这 8 个灯全灭；
如果 P4.0 输出 1，则 PNP 三极管不导通，不管 P0 口对外全部送 0 或送 1，这 8 个灯都不会有电流流过，都不会亮。P4.0 控制 PNP 三极管，作为这 8 个 LED 指示灯的总电源开关。

2.17.2　准备工作

1、添加常用头文件 ai_usb.h

#include "ai_usb.h"　　//使用 usb 所需要包含的头文件。

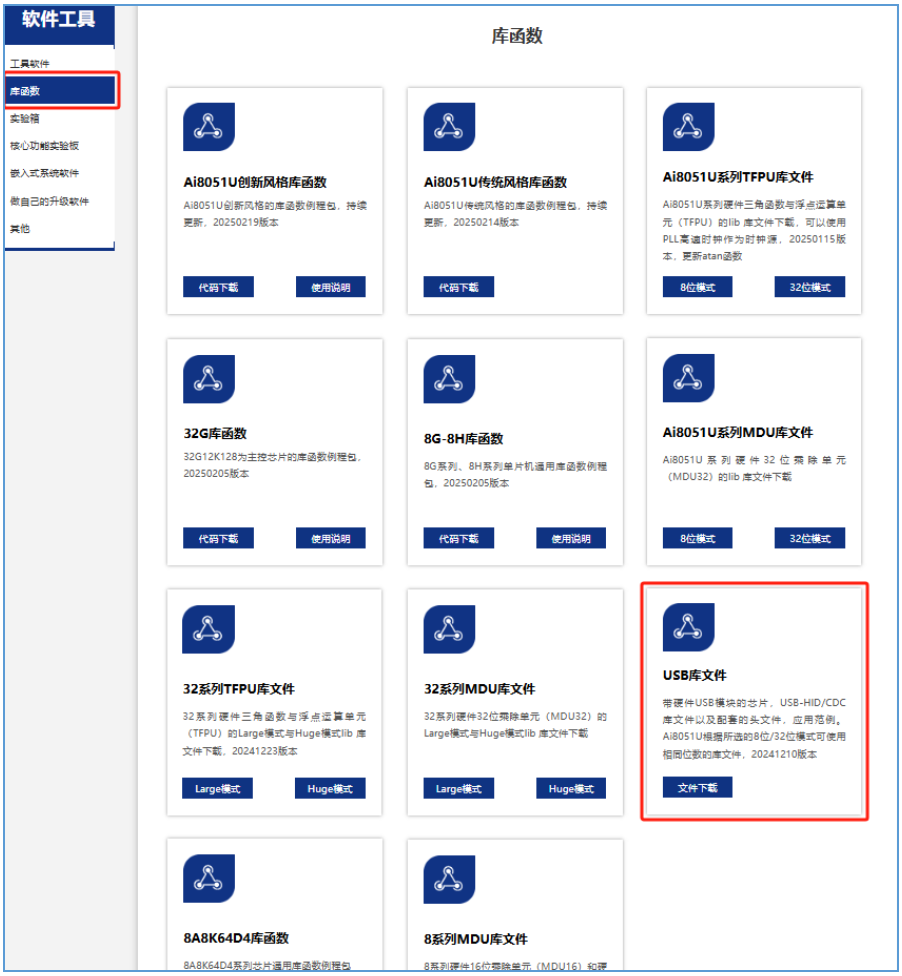
这个头文件在使用 AIapp-ISP 软件添加型号和头文件到 Keil 中时就已经一并被添加到 Keil 中了。



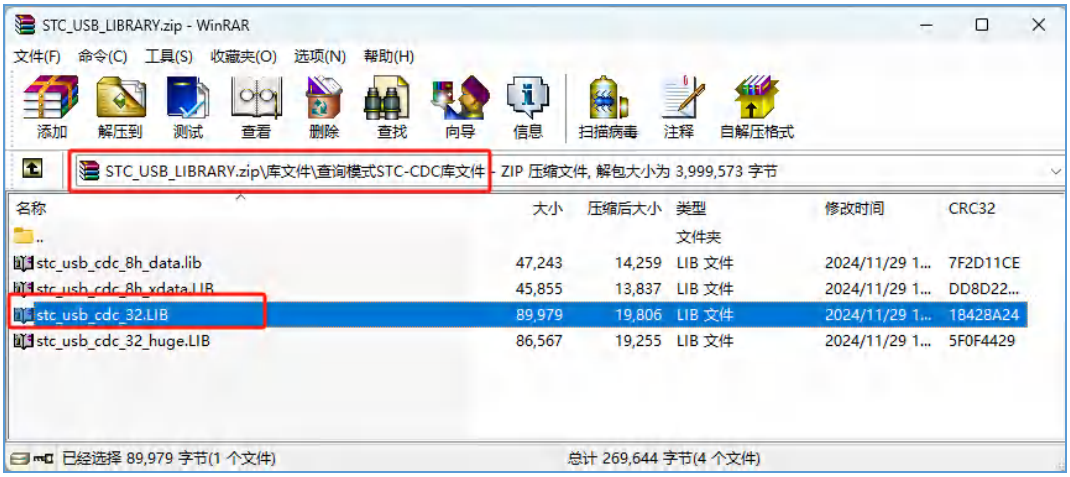
2、添加 usb_cdc 库函数文件

因为要使用到 USB 功能，必须在项目中添加 USB 库函数。这部分可以到 STCAI 的官网：

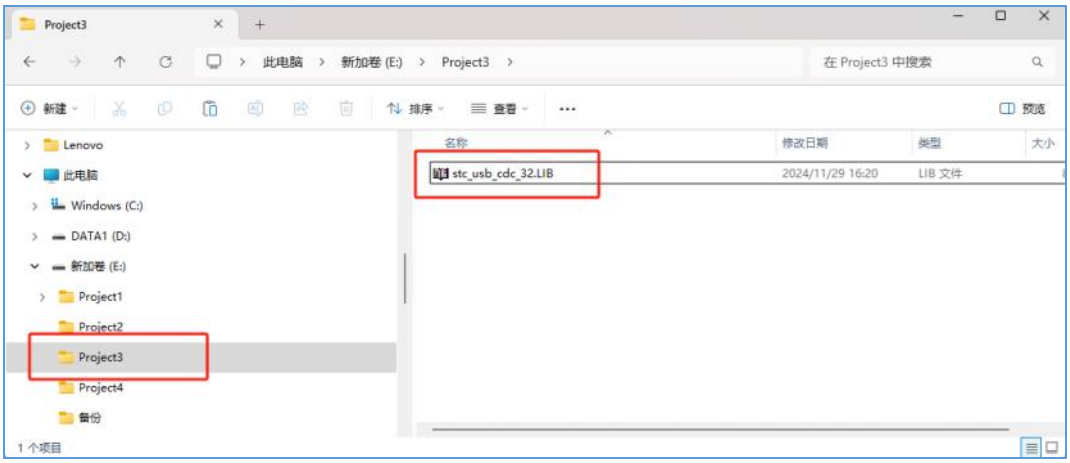
https://www.stcai.com/khs 的 USB 库文件中进行下载



下载完成后，打开压缩包“STC_USB_LIBRARY.zip”，
在’库文件\查询模式 STC-CDC 库文件’中，复制出来 stc_usb_cdc_32.LIB

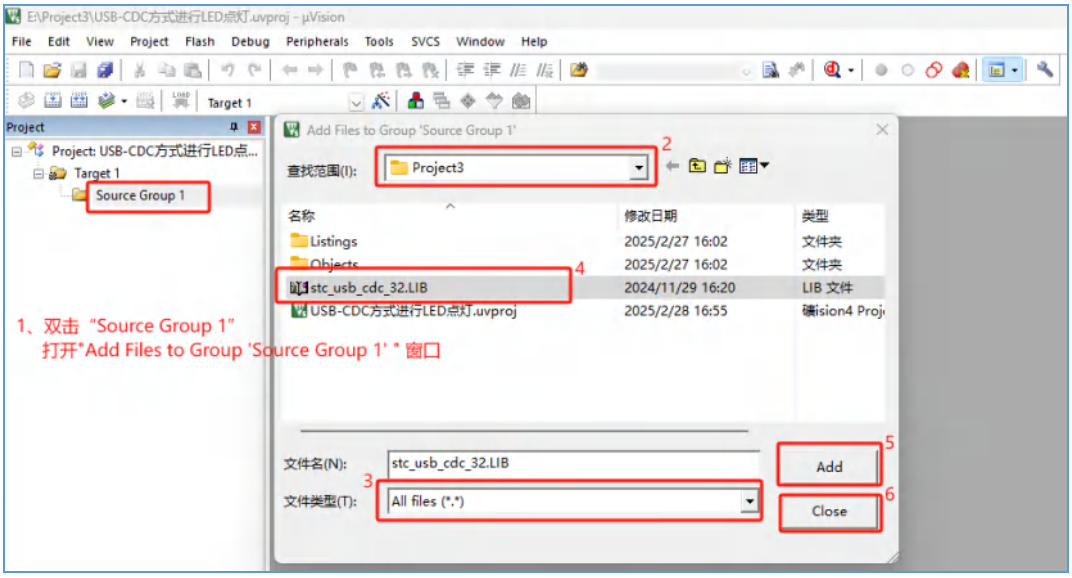


3、将库函数“stc_usb_cdc_32.LIB”复制到创建工程所在的文件夹 project3 中

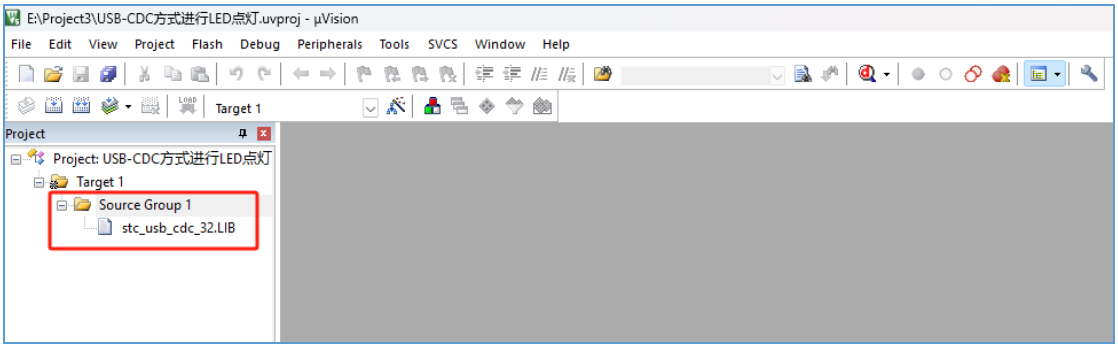


4、将库函数 stc_usb_cdc_32.LIB 添加进入此工程项目中

在 Keil 中，双击 Source Group，在弹出的 Add Files to Group 'Source Group 1 窗口中，将库函数 stc_usb_cdc_32.LIB 添加进入此工程，就可以使用 USB-CDC 功能点亮 LED 灯。



这样，Keil 界面变化如下图：



【注意】：USB-CDC 串口则更类似传统串口的形式，使用 COM 口进行通讯，但是只有 Win10 及其以上才默认携带 USB-CDC 的驱动，较低的系统版本可能要手动安装驱动才能使用。

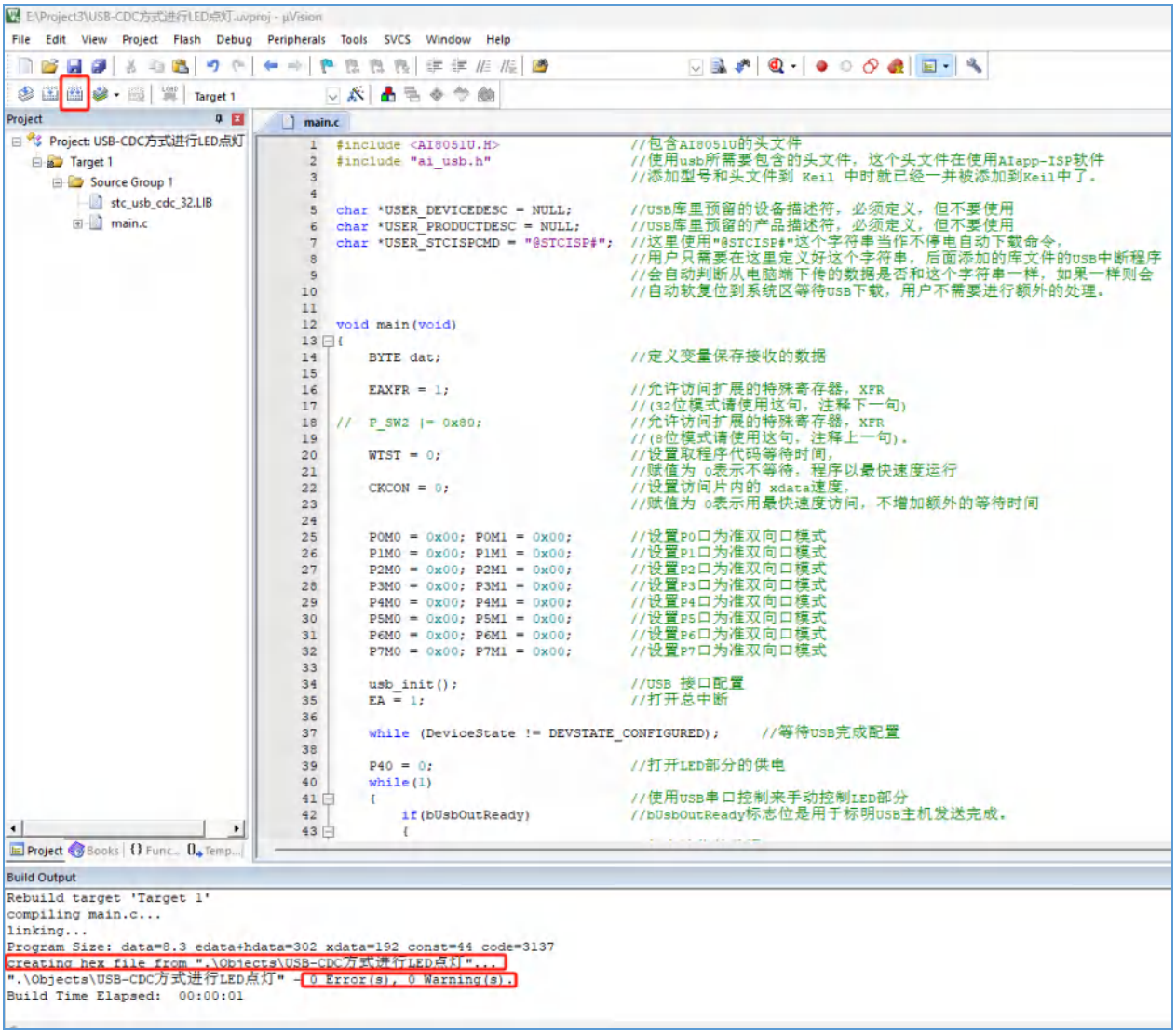
2.17.3　代码部分

在工程项目的 main.c 文件中可输入如下代码：
//通过 USB-CDC 方式控制 AI8051U 进行 LED 点灯

```
#include <AI8051U.H>        //包含 AI8051U 的头文件
#include   "ai_usb.h"        //USB 库和 USB 虚拟设备的头文件
//由于在 USB 库函数的代码中已经定义了，不停电 ISP 下载命令字"@STCISP#"，
//只需要在下载软件的“收到用户命令后复位到 ISP 监控程序区”选项卡中进行相应的设置，
//即可实现 USB 不停电 ISP 下载功能，或串口不停电 ISP 下载功能。

/*
如何获得上面这 2 个头文件，如已按照前面介绍的方法：
从 www.STCAI.com 网站，【软件工具 | 工具软件】处，下载安装最新的【AIapp-ISP 软件】，
选择其中的选项卡 【Keil 仿真设置 】，
点击【添加型号和头文件到 Keil 中，并添加仿真器驱动到 Keil 中】按钮，
则 Ai8051U.h 和 AI_usb.h 都会自动复制到 Keil 中，包含进来就可以直接使用，
大家可以打开看下其中的内容，AI_usb.h 和 stc32_stc8_usb.h 是相似的，只需用其中的 1 个头文件，
具体见 Keil_v5\C251\INC\STC 目录，或 Keil_v5\C51\INC\STC 目录。
如将对应的 USB 库文件添加到 项目中，后续用户程序中，USB 如收到命令字符串 "@STCISP#"
USB 库函数就会自动软复位到系统 ISP 程序区，实现 USB 不停电下载，具体见相应部分的说明。
*/

//char *USER_DEVICEDESC = NULL;
//char *USER_PRODUCTDESC = NULL;
//char *USER_STCISPCMD = "@STCISP#";
//使用 20250319 之前版本的库函数时需要在用户代码中增加上面 3 行定义。
//使用 20250319 及以后的版本都不再需要上面的定义

void main(void)
{
    BYTE dat;                       //定义变量保存接收的数据

    EAXFR = 1;                      //允许访问扩展的特殊寄存器，XFR
```

```
                                        //(32 位模式请使用这句，注释下一句)
//    P_SW2 |= 0x80;                    //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)。
      WTST = 0;                         //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
      CKCON = 0;                        //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

      P0M0 = 0x00; P0M1 = 0x00;         //设置 P0 口为准双向口模式
      P1M0 = 0x00; P1M1 = 0x00;         //设置 P1 口为准双向口模式
      P2M0 = 0x00; P2M1 = 0x00;         //设置 P2 口为准双向口模式
      P3M0 = 0x00; P3M1 = 0x00;         //设置 P3 口为准双向口模式
      P4M0 = 0x00; P4M1 = 0x00;         //设置 P4 口为准双向口模式
      P5M0 = 0x00; P5M1 = 0x00;         //设置 P5 口为准双向口模式
      P6M0 = 0x00; P6M1 = 0x00;         //设置 P6 口为准双向口模式
      P7M0 = 0x00; P7M1 = 0x00;         //设置 P7 口为准双向口模式
//上面的将所有 I/O 口都初始化为准双向口工作模式的程序，
//可以使用 AIapp-ISP 软件的“I/O 口配置工具”来生成，见本节后面的截图。

      usb_init();     //调用 USB 初始化函数，不需要立即判断电脑已正确识别到 USB 从设备
      /* USB 型单片机从设备，如需要主动向电脑发送数据，
         在执行 USB_SendData( )函数和 printf_usb( )函数时，
         这两个函数已增加了判断电脑是否已正确识别到 USB 从设备的程序。
         如果电脑要主动发送数据给 USB 从设备，电脑自己会主动判断与 USB 从设备是否已正确连接。
      */
      EA = 1;                           //总中断允许位打开

      P40 = 0;                          //打开 LED 部分的供电

      while(1)
      {                                 //使用 USB 串口控制来手动控制 LED 部分
          if(bUsbOutReady)              //bUsbOutReady 标志位是用于标明 USB 主机发送完成。
                //USB 库中有定义 bUsbOutReady 位变量，查询方式必须调用一次“usb_OUT_done();”
          {
              dat = UsbOutBuffer[0];    //保存接收的数据
              P0 = dat;                 //将接收的数据送到 P0 口显示
              LED40_SetPort(0,dat);     //使用 AIapp-ISP 软件菜单“调试仿真接口”中的
                                        //“擎天柱-LED-DIP40”进行模拟显示 P0 口
              usb_OUT_done();           //本包接收的数据处理完毕，准备接收下一包数据
                //当使用查询方式 if(bUsbOutReady)，查询调用 USB 库函数时，在数据处理完成后，
                //必须调用一次“usb_OUT_done();”，这样才能接收下一包数据。
          }
      }
}
```

代码编写完后，点击工具栏上的 Rebuild 按钮进行编译



出现 creating hex file 和 0 Error(s) 0 Warning（s）即为编译成功，成功创建了 hex 文件且无报错。此 hex 文件，一般是在对应工程目录下的 Objects 文件夹里。

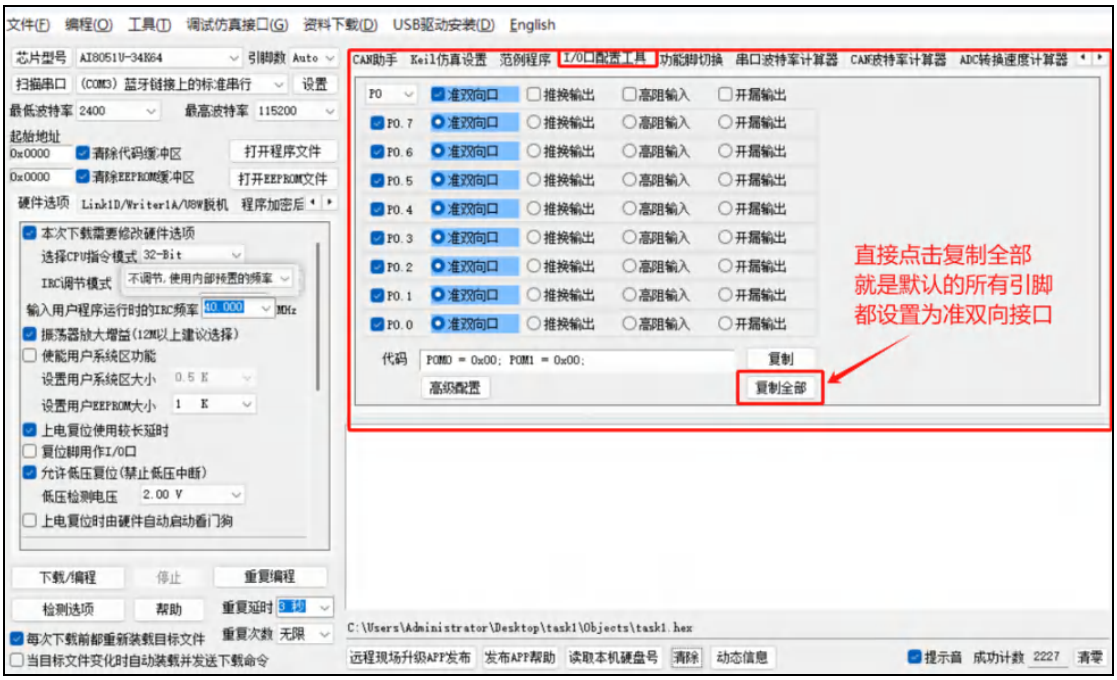
【小技巧--灵活运用 AIapp-ISP 软件的工具自动生成代码】

I/O 口的初始化部分就可以使用 **AIapp-ISP** 软件的“**I/O 口配置工具**”来生成。

例如上述代码中这一段将所有 I/O 口设置为准双向口模式的代码：

```
P0M0 = 0x00; P0M1 = 0x00;
P1M0 = 0x00; P1M1 = 0x00;
P2M0 = 0x00; P2M1 = 0x00;
P3M0 = 0x00; P3M1 = 0x00;
P4M0 = 0x00; P4M1 = 0x00;
P5M0 = 0x00; P5M1 = 0x00;
P6M0 = 0x00; P6M1 = 0x00;
P7M0 = 0x00; P7M1 = 0x00;
```

就可以使用 AIapp-ISP 的 I/O 口配置工具进行生成：

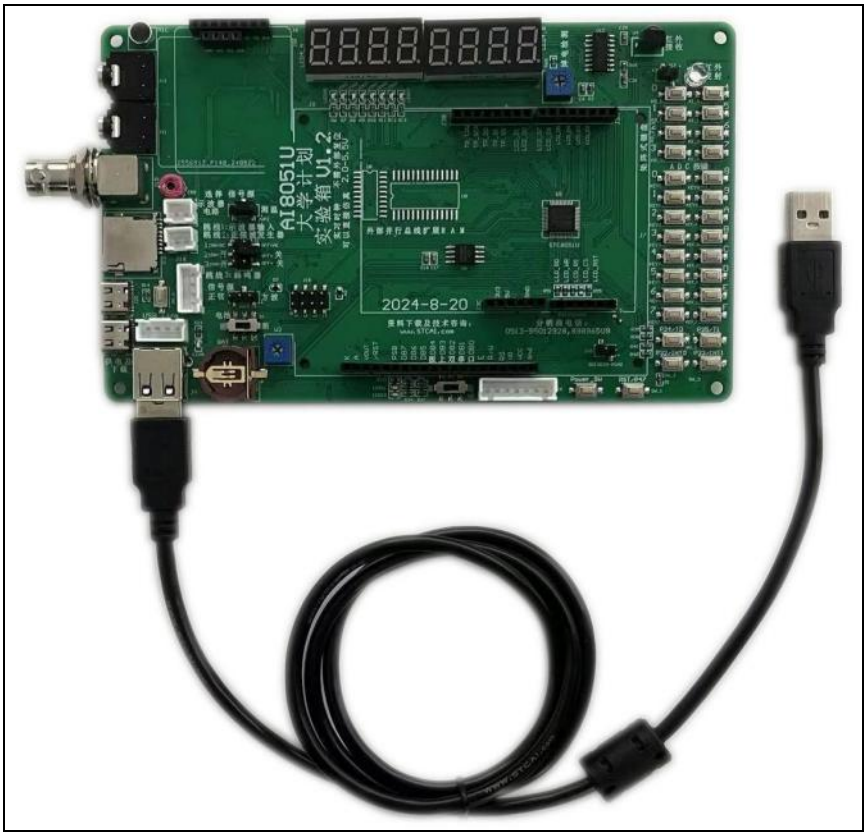


将复制出来的内容直接粘贴到代码文件中，不用手动输入。

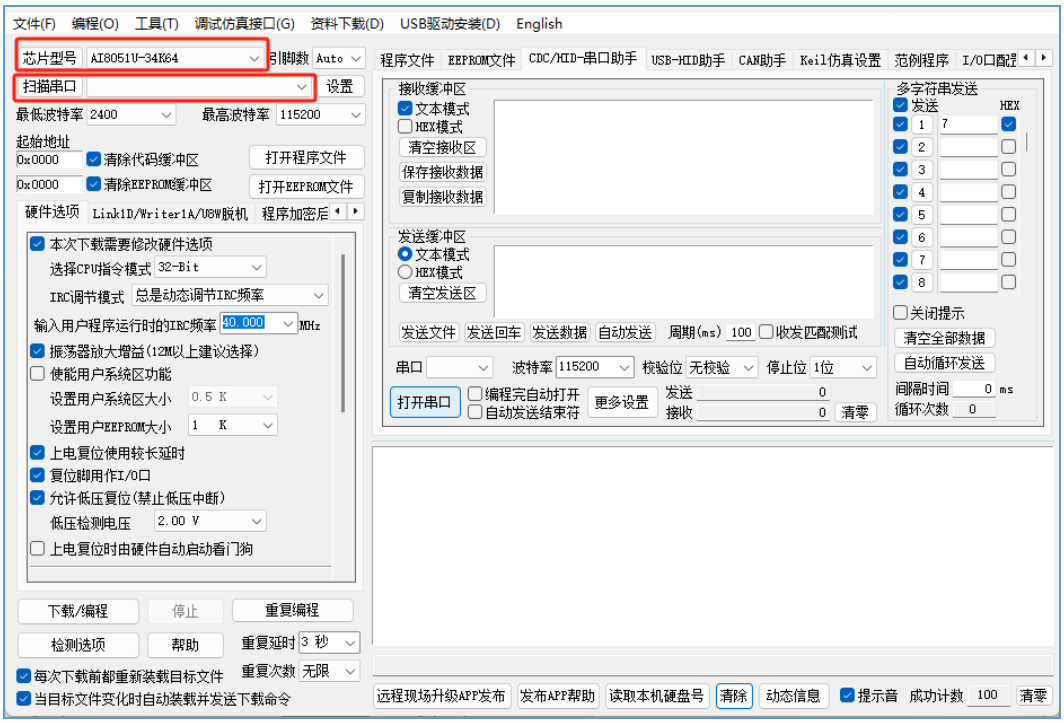
2.17.4　下载用户程序到实验箱并查看实验现象，实物图

2.17.4.1　连接到实验箱

首先使用实验箱中配置的 USB-TypeA 到 USB-TypeA 线将实验箱与电脑正确连接。

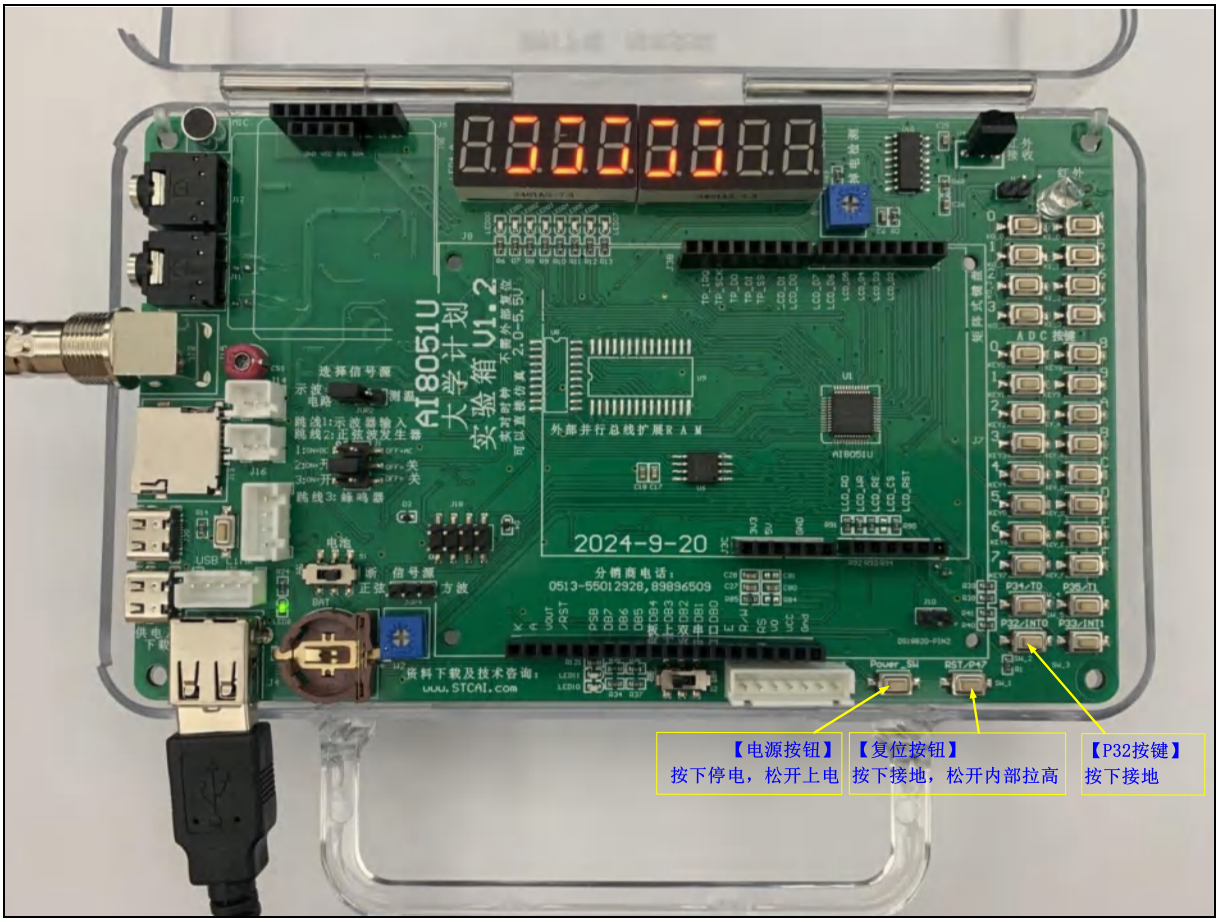


然后打开 AIapp-ISP-v6.95F 或以上版本。选择单片机型号：Ai8051U-34K64，如下图，我们看到“扫描串口”后面的端口列表框不是“（HID1）USB-Writer”，说明还不能下载程序到实验箱。

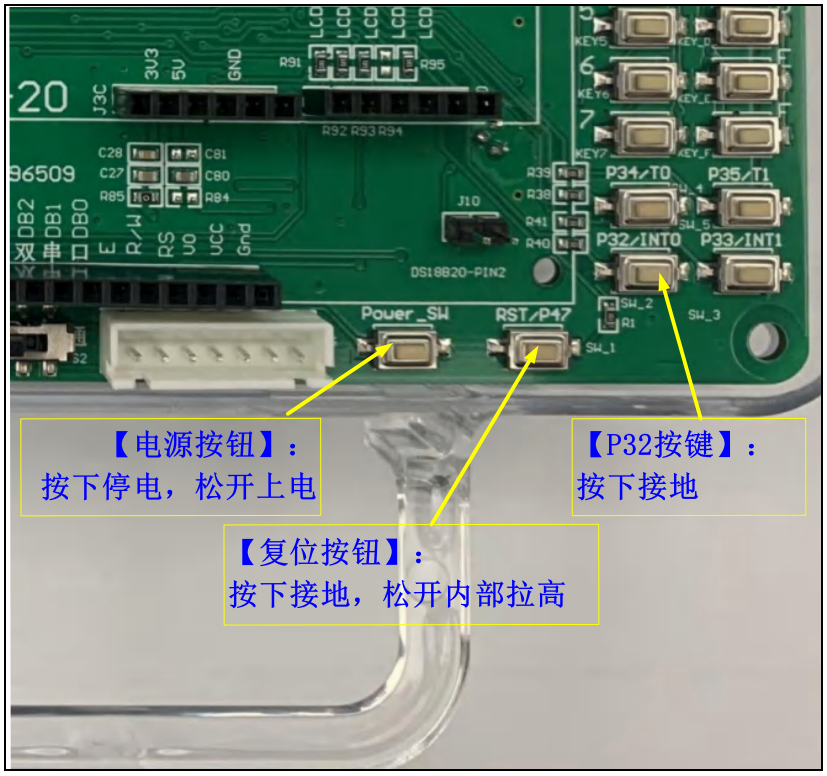


2.17.4.2 硬件 USB 下载用户程序步骤

下图是实验箱已经和电脑相连的正面图片，数码管是随机显示的，数码管下面的 8 个 LED 灯都是暗淡无光的，此实验的目的是通过 USB-CDC 方式控制 AI8051U 进行 LED 点灯。



实验箱正下方有两排按键，找到 P3.2 按键和 Power_SW 按键



【方法一，P3.2 按键，再结合停电上电下载】

1、按下板子上的 P3.2 按键，就是 P3.2 接地

2、给目标芯片重新上电，不管之前是否已通电。

===本 Ai8051U 实验箱是：按下电源按钮停电，再松开电源按钮上电，

这时电脑端 ISP 下载软件会自动识别出“(HID1) USB Writer”，此时就与 P3.2 状态无关了，可以松开 P3.2 按键（P3.2 在用户程序区可做其它任意用途）

3、点击电脑端下载软件中的【下载/编程】按钮（注意：USB 下载与串口下载的操作顺序不同）

电脑端 ISP 下载软件提示：下载进行中，几秒钟后，提示下载成功！

【方法二：复位管脚低电平复位下载】

USB 连接好并已上电的情况下，外部按键复位也可进入 USB 下载模式，注意：

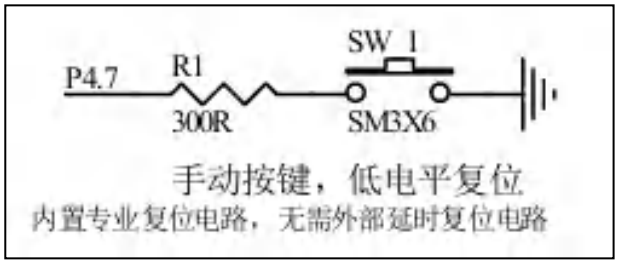
USB 连接好并已上电的情况下，外部按键复位也可进入 USB 下载模式，注意：

P4.7-nRST 出厂时默认是 P4.7-I/O 功能，要改为复位功能，需 ISP 烧录时取消设置复位脚用作 I/O 口，停电一次再上电才生效，程序区中用户程序也可改为复位脚或 I/O，这个立即生效。

1、按下复位管脚/P4.7-nRST 外接的低电平复位按键，复位 MCU，

再松开复位键，MCU 首先从系统程序区启动运行系统程序，判断是否要下载用户程序，

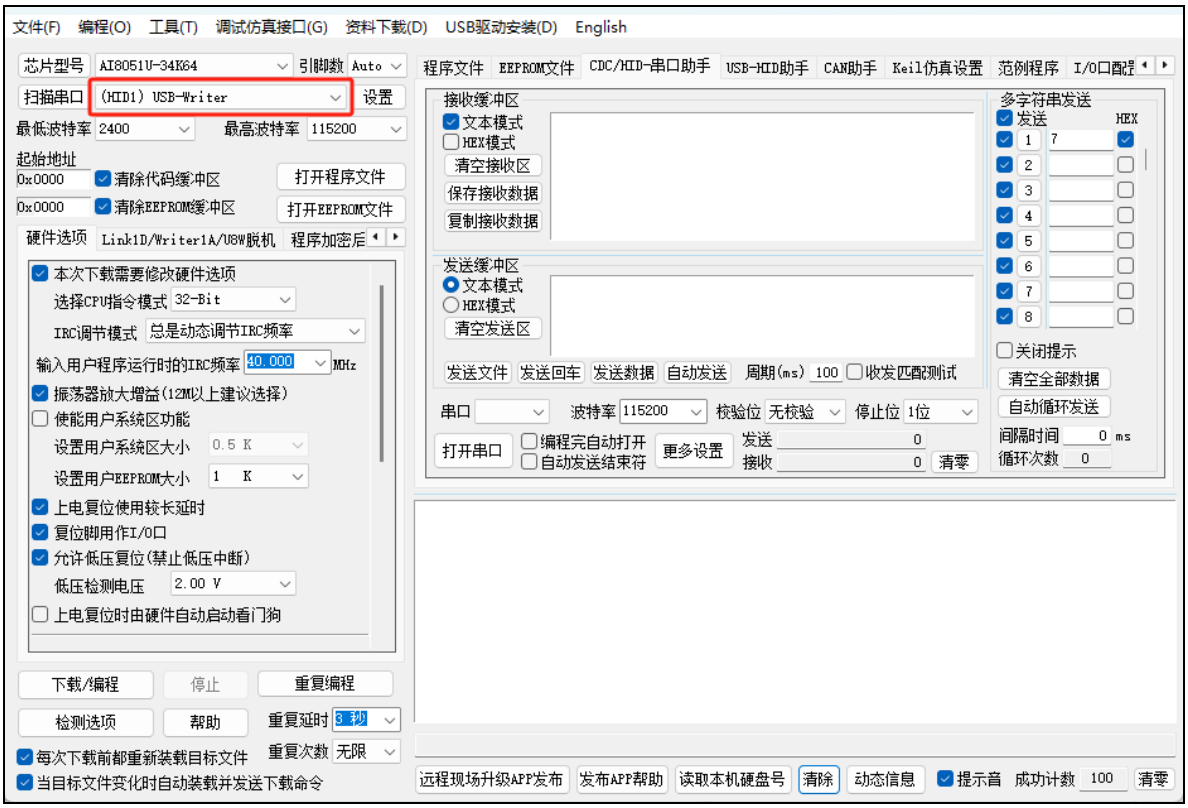
等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后



2、点击电脑端下载软件中的【下载/编程】按钮

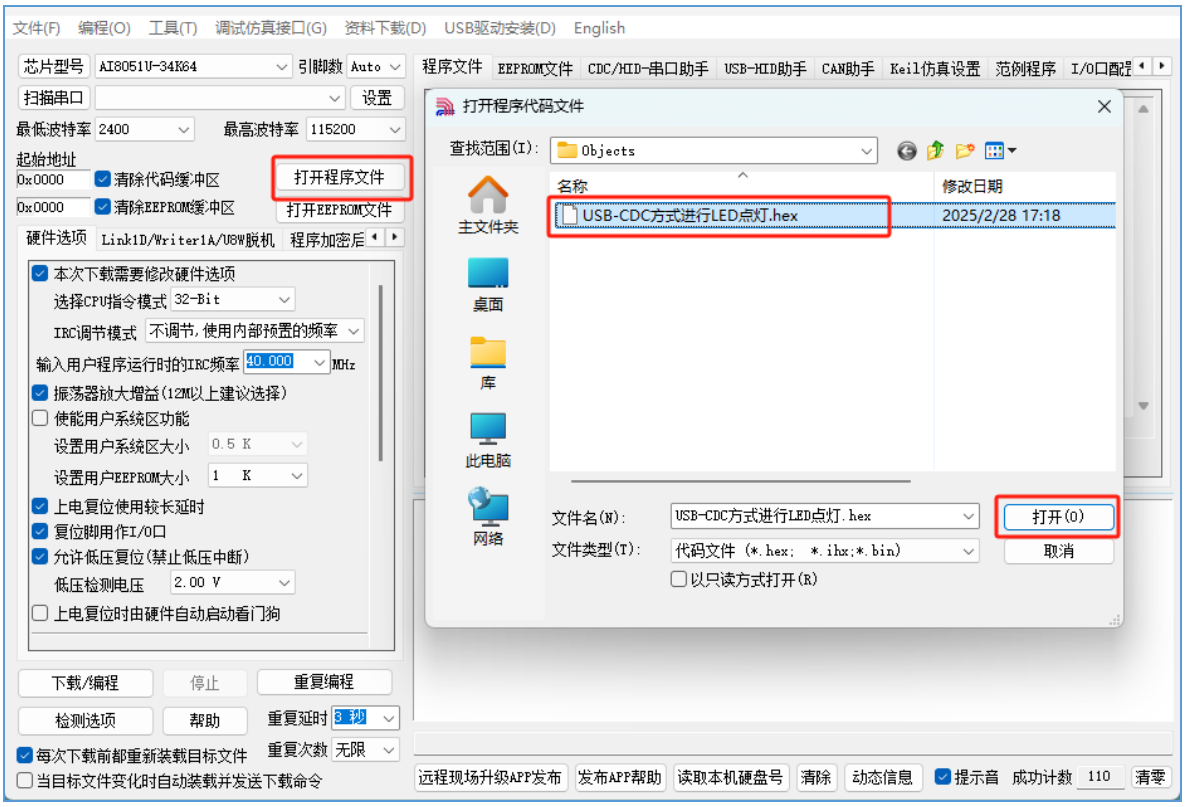
电脑端 ISP 下载软件提示：下载进行中…，下载成功！

备注：如果目标 MCU 的用户程序区已有 USB 通讯程序（含 CDC、HID 等），则通过复位脚复位进入 USB 下载模式不被支持，只能通过停电上电冷启动下载或者从用户程序区软复位到系统区下载

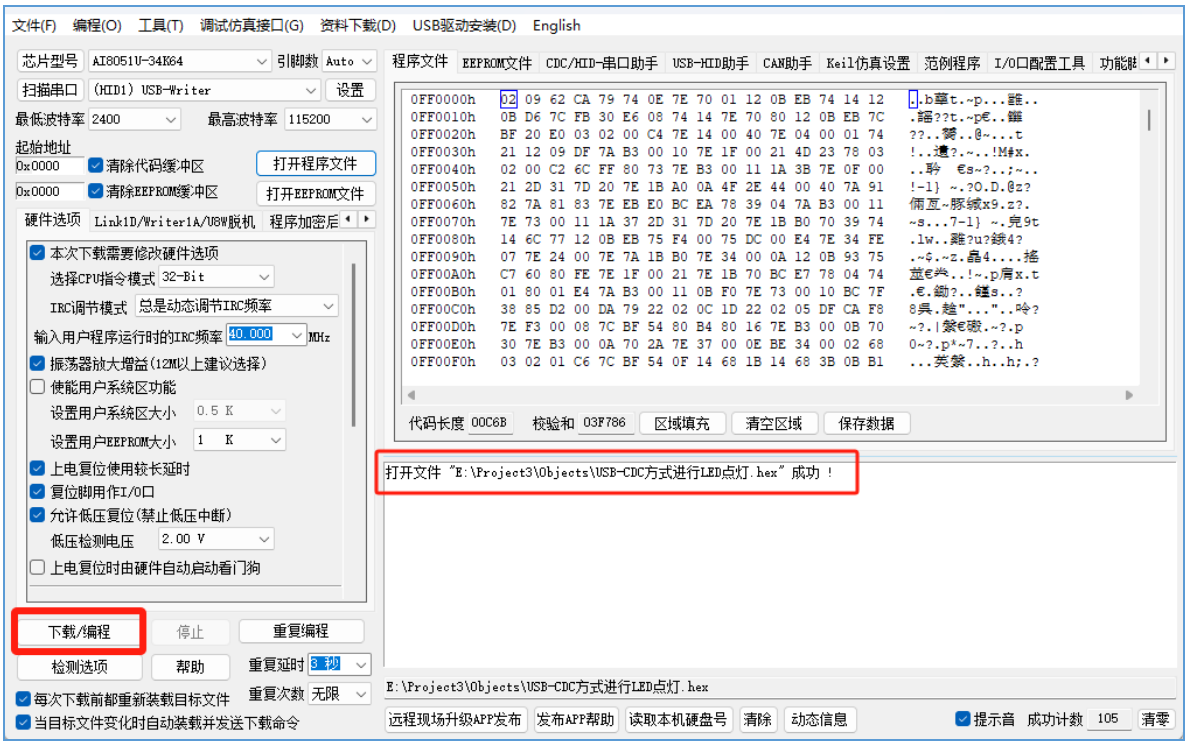


2.17.4.3 下载用户程序文件并观察实验现象

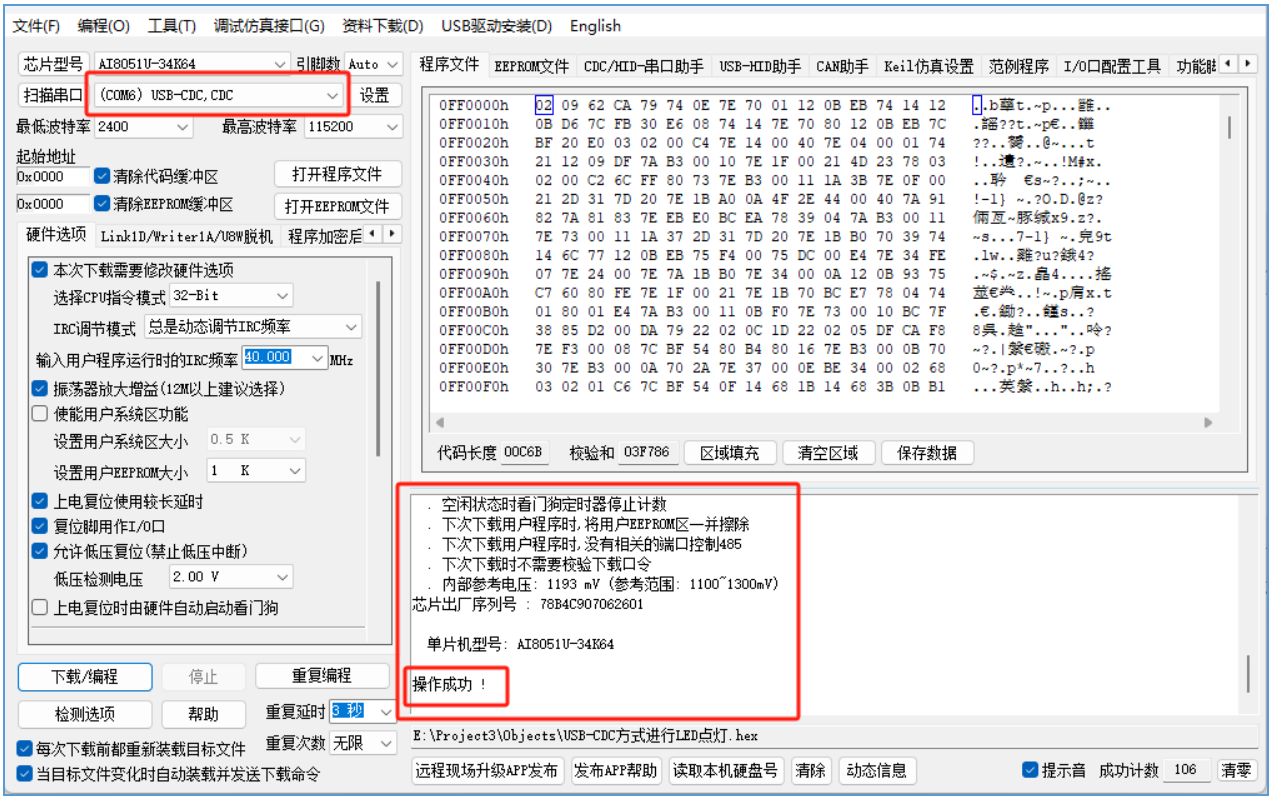
现在打开用户程序文件，通过点击界面上的“打开程序文件”（打开的是项目程序文件编译无误后生成的 HEX 文件，一般默认存放在该项目的 Objects 目录中），打开之前用 Keil 中编译好的 HEX 文件：E:\Project3\Objects\USB-CDC 方式进行 LED 点灯.hex



然后，AIapp-ISP 软件界面如下图所示，出现“‘打开文件***.hex’成功！”

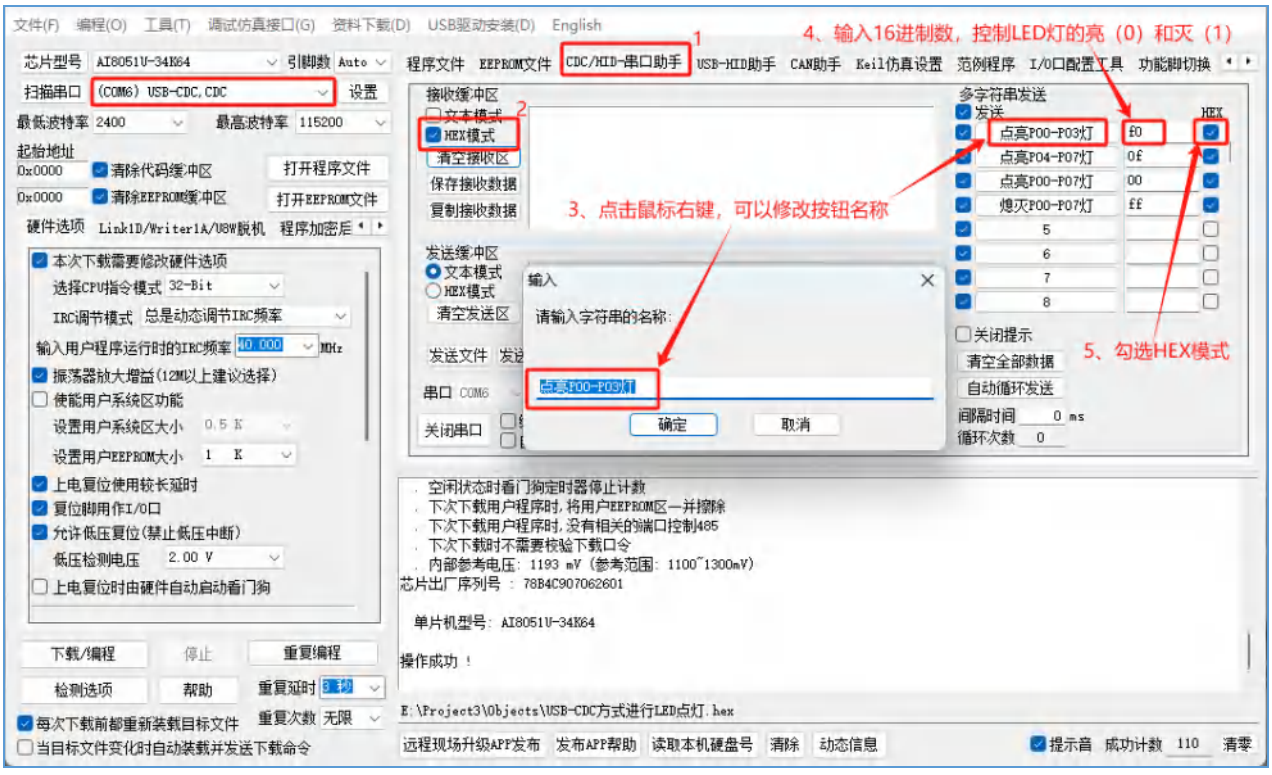


点击“下载/编程”，把 task1.hex 烧录到实验箱，AIapp-ISP 软件界面出现：操作成功！
恭喜你！

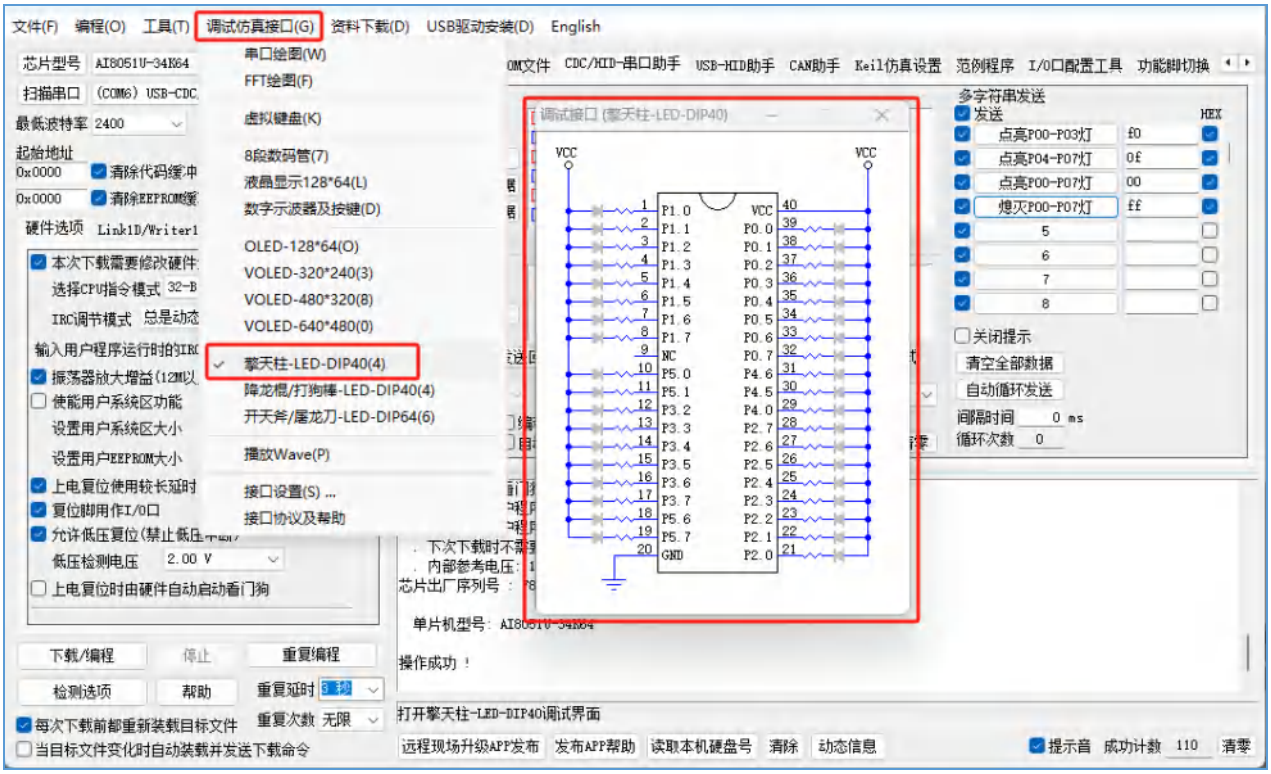


这时在 AIapp-ISP 软件界面，我们看到串口选项框中已经出现了一个(COM6)USB-CDC,CDC 串口，如果电脑系统并没有产生 CDC 端口，可能是因为系统版本较低（低于 Windows10 版本）。

在 AIapp-ISP 软件界面，看到 USB-CDC 端口后，点击 CDC/HID-串口助手选项卡，然后按照下图中步骤操作。



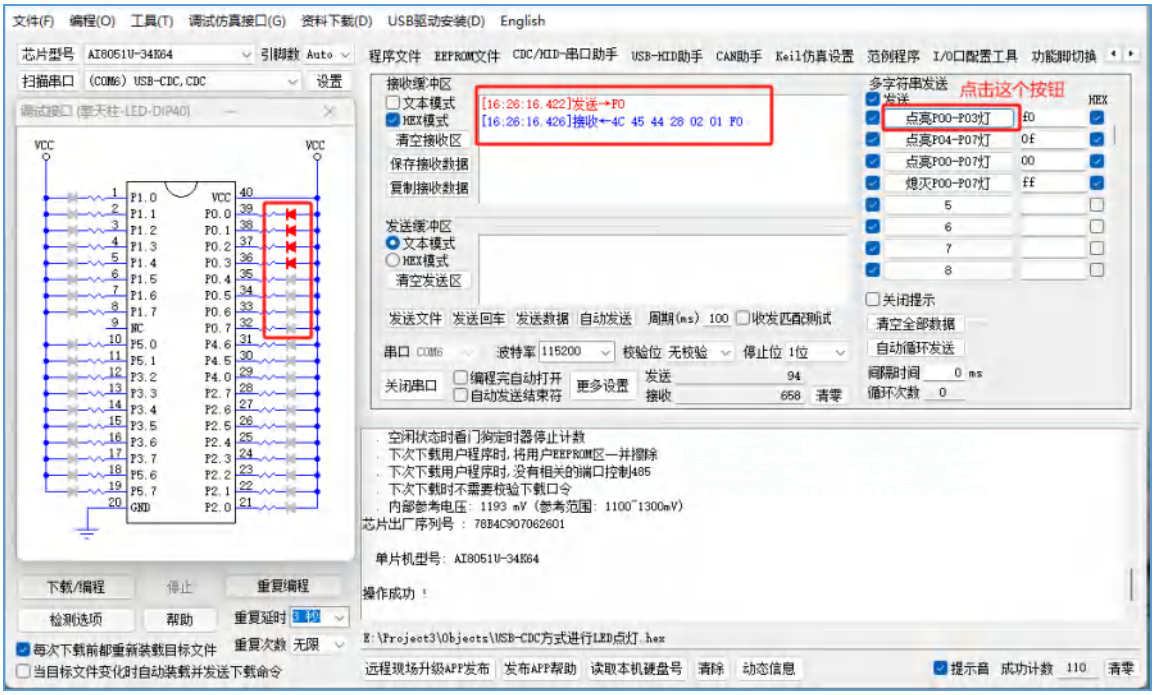
同时为了直观查看实验情况，可以打开 AIapp-ISP 软件的“调试仿真接口”菜单，选择“擎天柱-LED-DIP40”进行仿真调试：



a) 点击 CDC/HID-串口助手选项卡中的“点亮 P00-P03 灯”按钮

1、观察 AIapp-ISP 软件的 CDC/HID-串口助手选项卡界面，如下图：

- 在接收区会出现发送、接收数据；
- 擎天柱-LED-DIP40 的 P0.0-P0.3 这 4 个灯被点亮

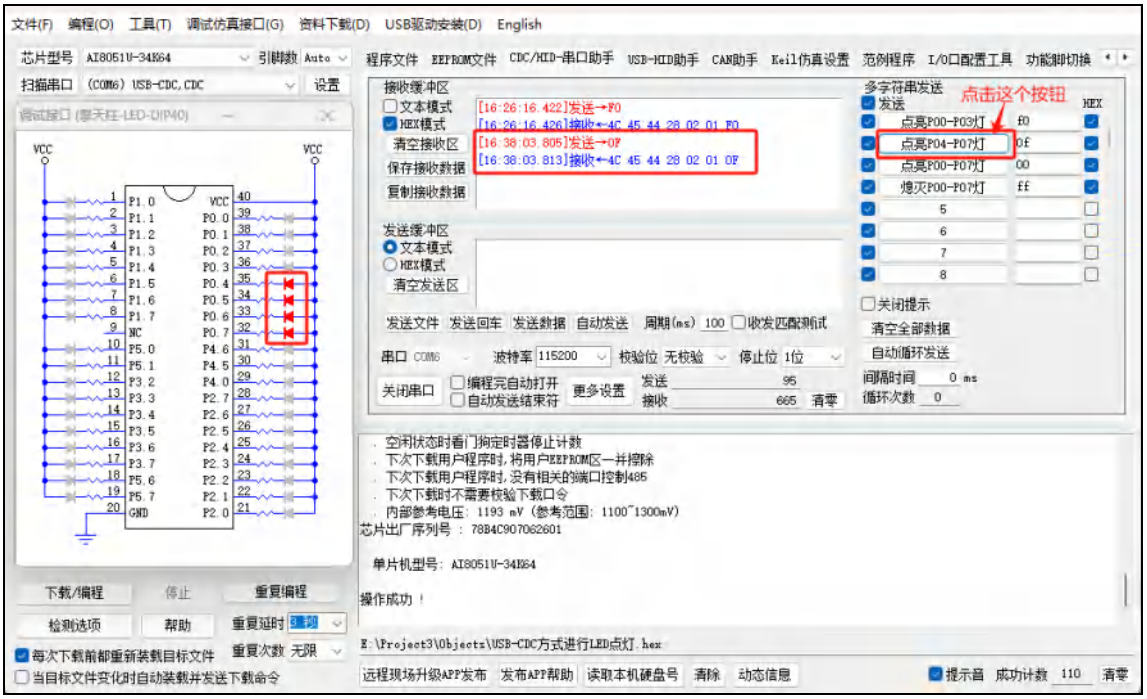


2、观察实验箱的 8 个 LED 灯，P0.0-P0.3 这 4 个 LED 灯被点亮，如下图：

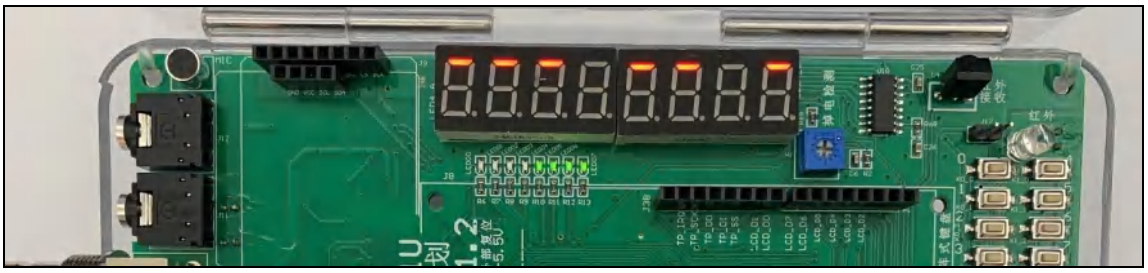
b) 点击 CDC/HID-串口助手选项卡中的“点亮 P04-P07 灯”按钮：

1、观察 AIapp-ISP 软件的 CDC/HID-串口助手选项卡界面，如下图：

- 在接收区会出现发送、接收数据；
- 擎天柱-LED-DIP40 的 P0.4-P0.7 这 4 个灯被点亮



2、观察实验箱的 8 个 LED 灯，P0.4-P0.7 这 4 个 LED 灯被点亮，如下图：



至此，实验成功！

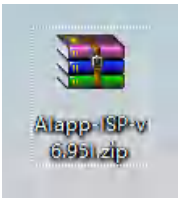
2.17.5　使用【AiCube】项目创建助手工具创建通过 USB-CDC 方式控制 Ai8051U 进行 LED 点灯

STCAI 在最新版的 AIapp-ISP 软件中，推出【AiCube】项目创建助手工具，目前支持 Ai8051U-32Bit 各功能模块的智能化配置，并生成程序框架。这项功能在程序员建立项目时，简化了项目的初始化难度，提高效率，避免因不太熟悉或不完全了解该单片机的性能而忽略个别配置，引起编译错误等问题。尤其对刚涉足单片机的初学者，此工具不仅引导其了解单片机编程流程，还极大的降低了入门门槛，缩短了摸索的时间，提高入门的兴趣和信心。现在我们使用这个工具来生成实现此实验功能的配置和代码。

a) 安装 AIapp-ISP 下载/编程/烧录/软件

安装 STCAI 公司研发的 AIapp-ISP 下载 / 编程 / 烧录 / 软件，含强大的辅助开发工具包。登录 STCAI 官网，下载最新版的 AIapp-ISP 安装包，下载链接如下：https://www.stcai.com/gjrj

下载下来的是压缩包文件如下图：

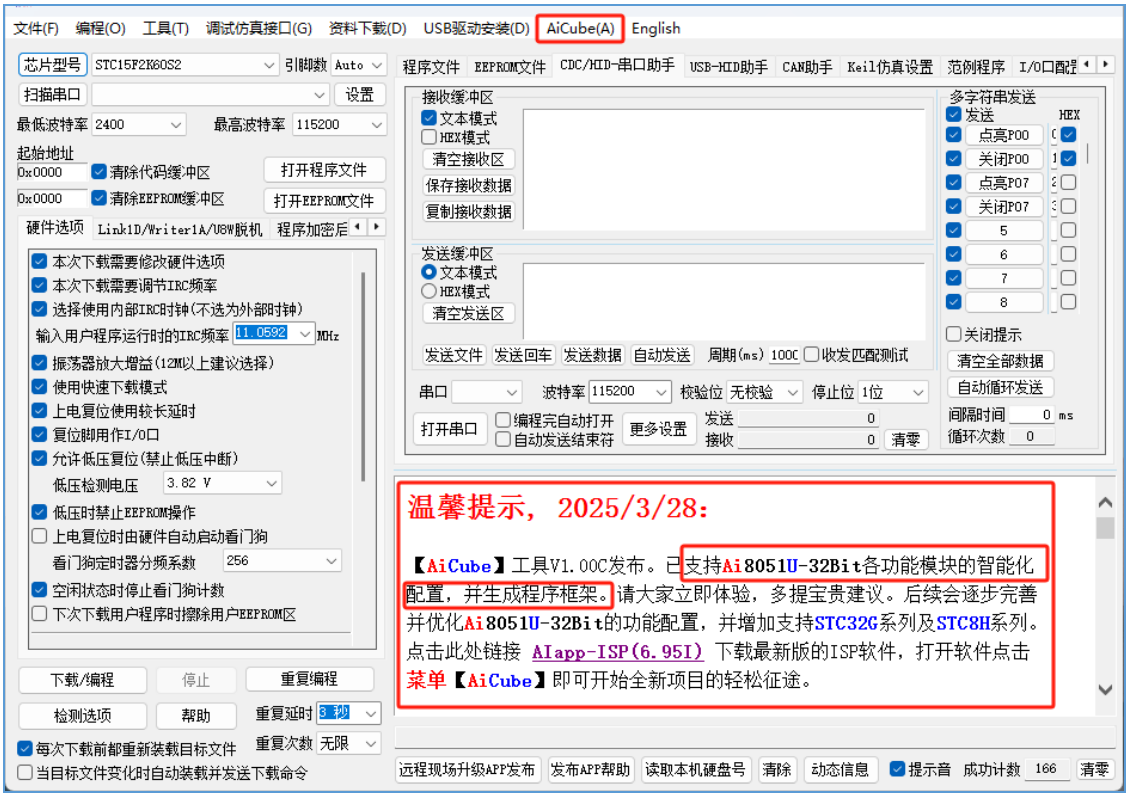


双击后解压缩如下图：

如果是解压缩到非桌面的其他目录，如 D:\STCAI-ISP，建议创建快捷方式到桌面方便打开。这实际是一个免安装的绿色软件。

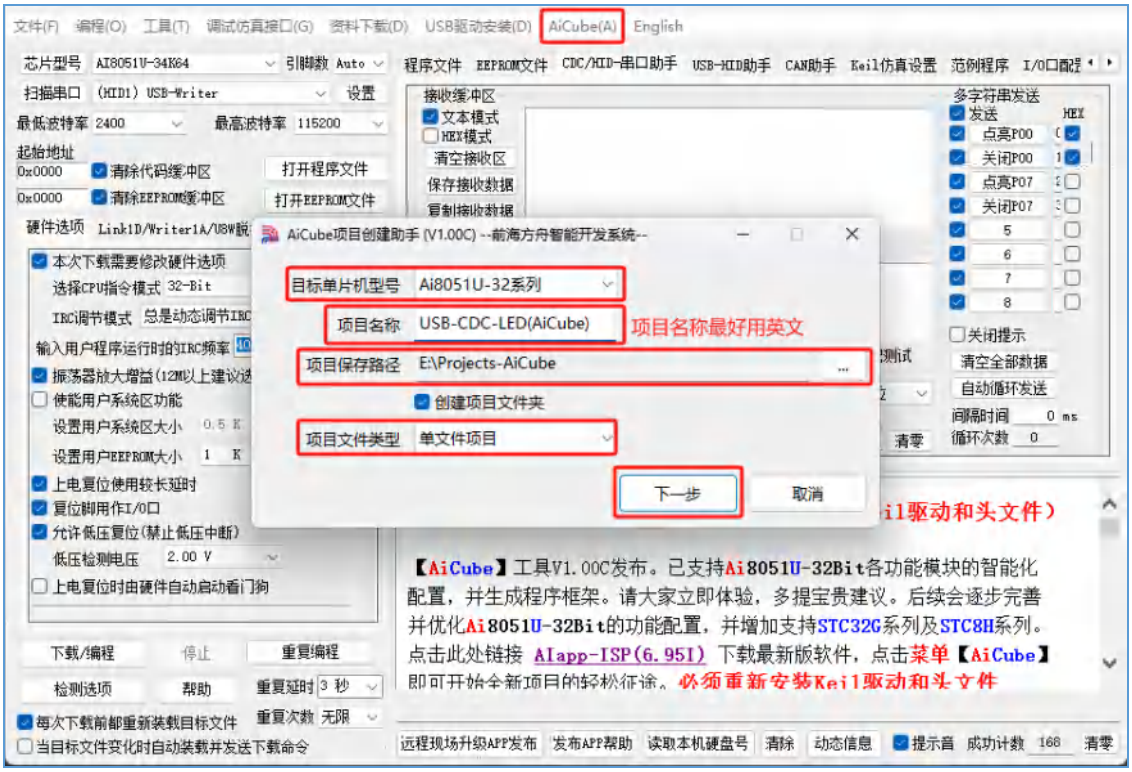
b) 打开【AiCube】助手

双击图标，即可打开 AIapp-ISP 软件界面，即可直接使用，如下图：

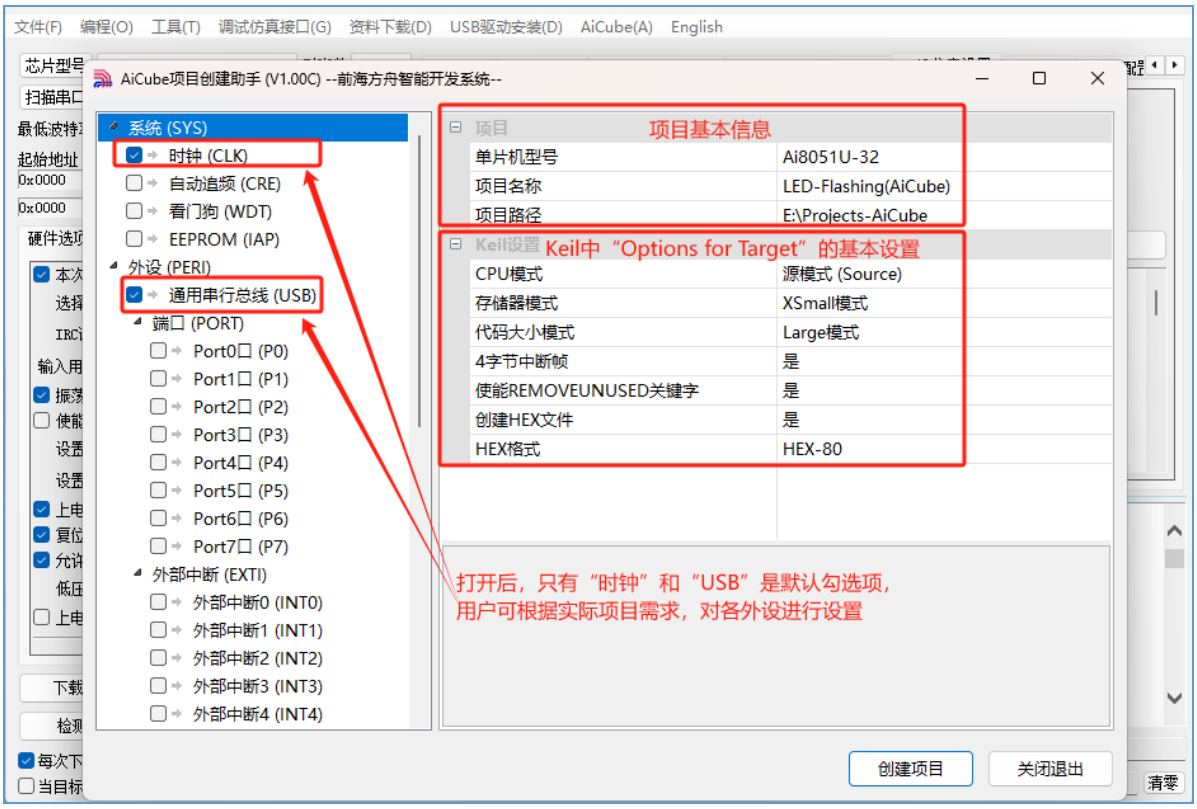


c) 使用【AiCube】助手创建项目

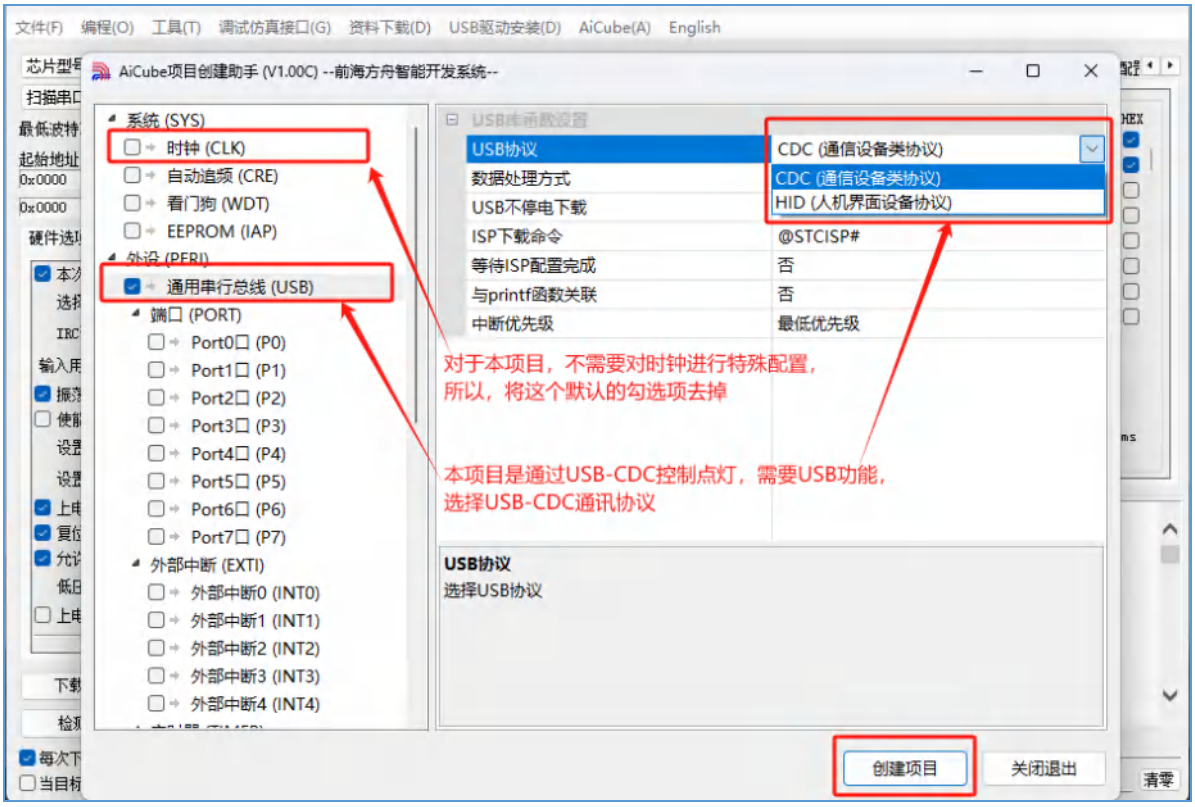
点击【AiCube（A）】，弹出如下窗口：



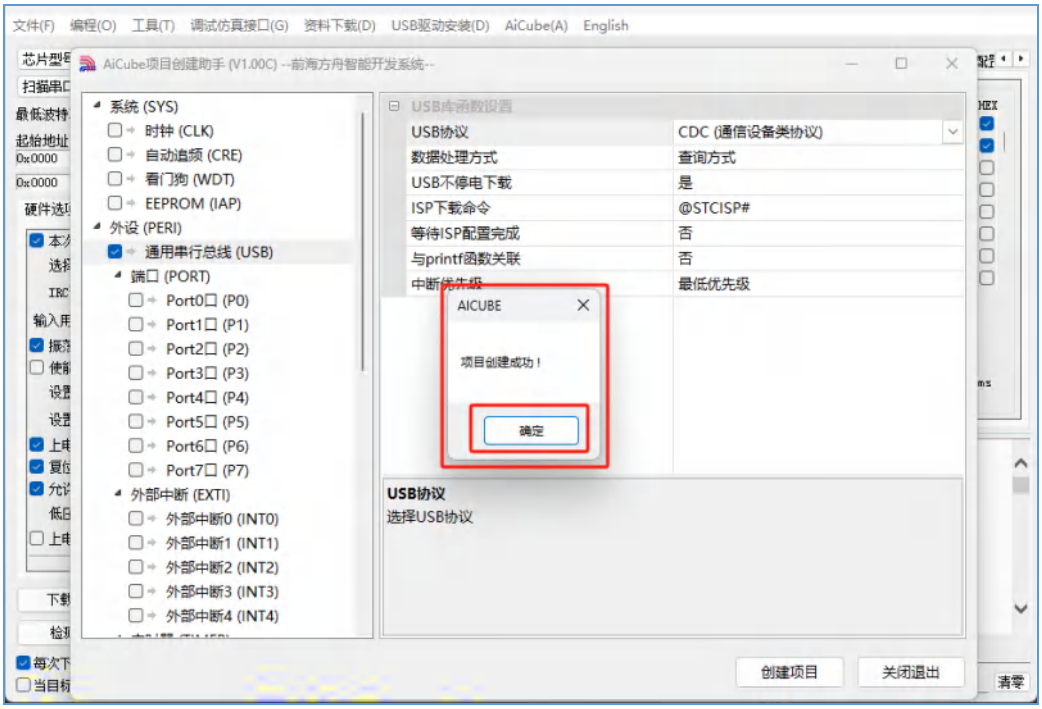
选择并填写项目基本信息，建议项目名称用英文！点击“下一步”，出现如下配置界面：



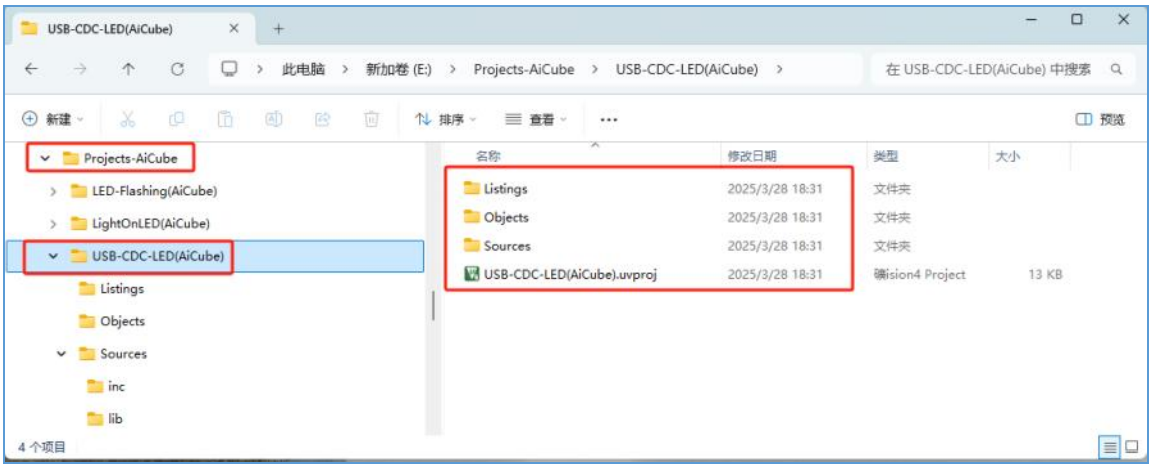
因本项目用到 USB 功能，要加载 USB 库函数、代码中要有 USB 初始化、USB-CDC 协议等，使用【AiCube】助手根据本项目的实际需求，对配置进行如下修改：



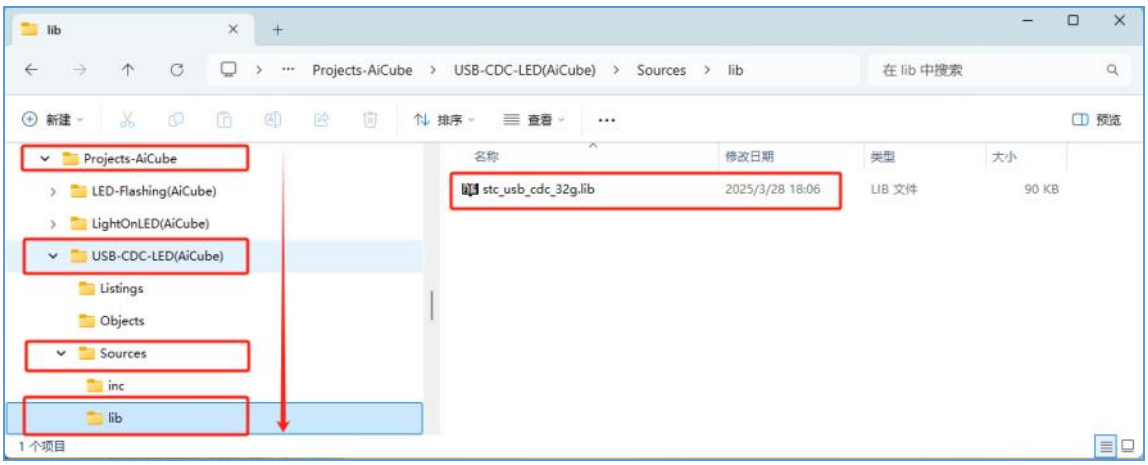
点击“创建项目”，【AiCube】助手提示：“项目创建成功！”，如下图：



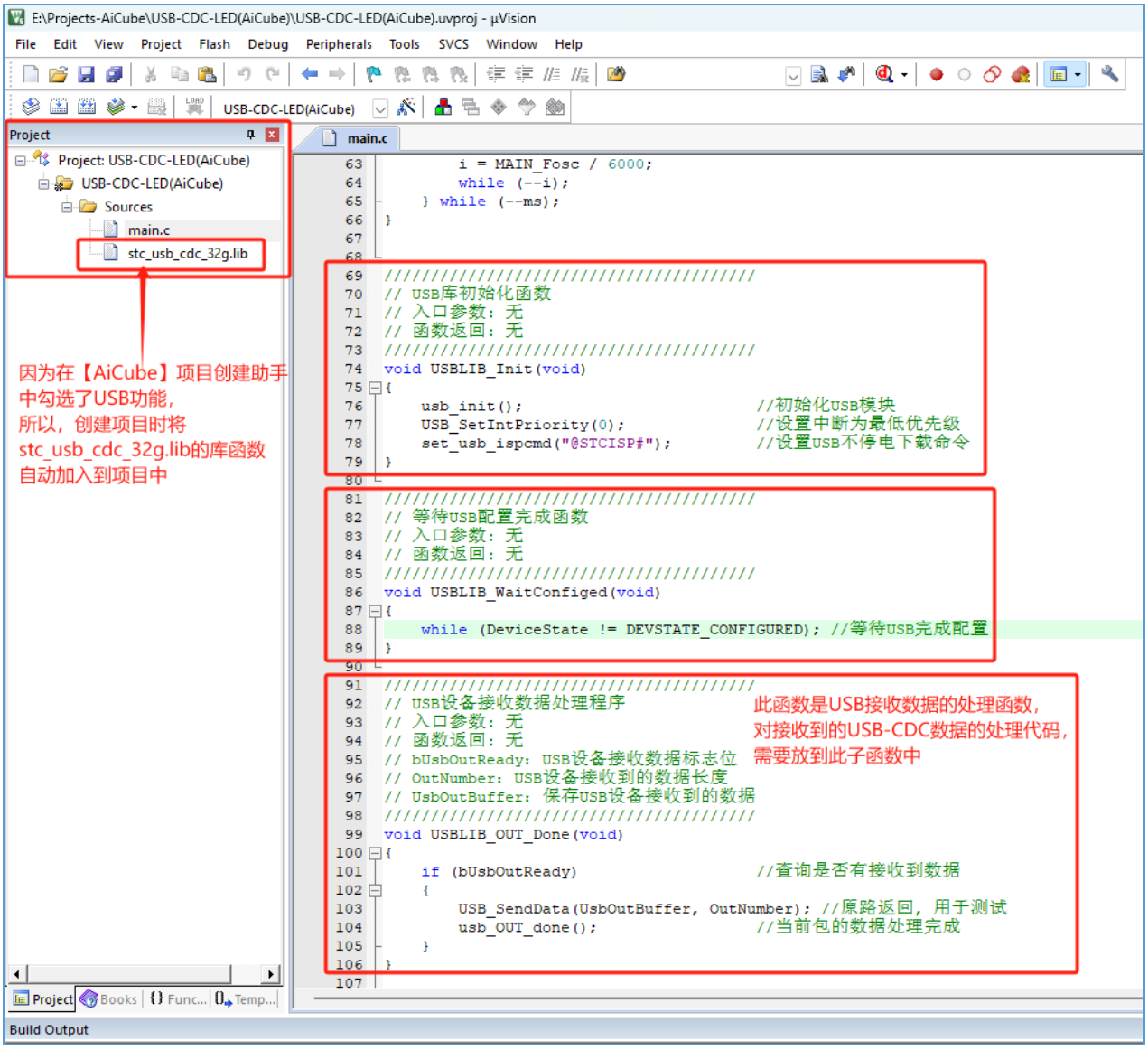
点击“确定”，即成功创建了项目及项目文件目录“USB-CDC-LED(AiCube)”，找到并打开刚刚创建的项目目录。发现刚刚的操作过程，【AiCube】助手已经帮我们建立了项目目录及 USB-CDC-LED(AiCube).uvproj 文件，如下图：



同时打开 Sources>lib 子目录，发现，【AiCube】助手连 USB 库函数也已经给加载了：



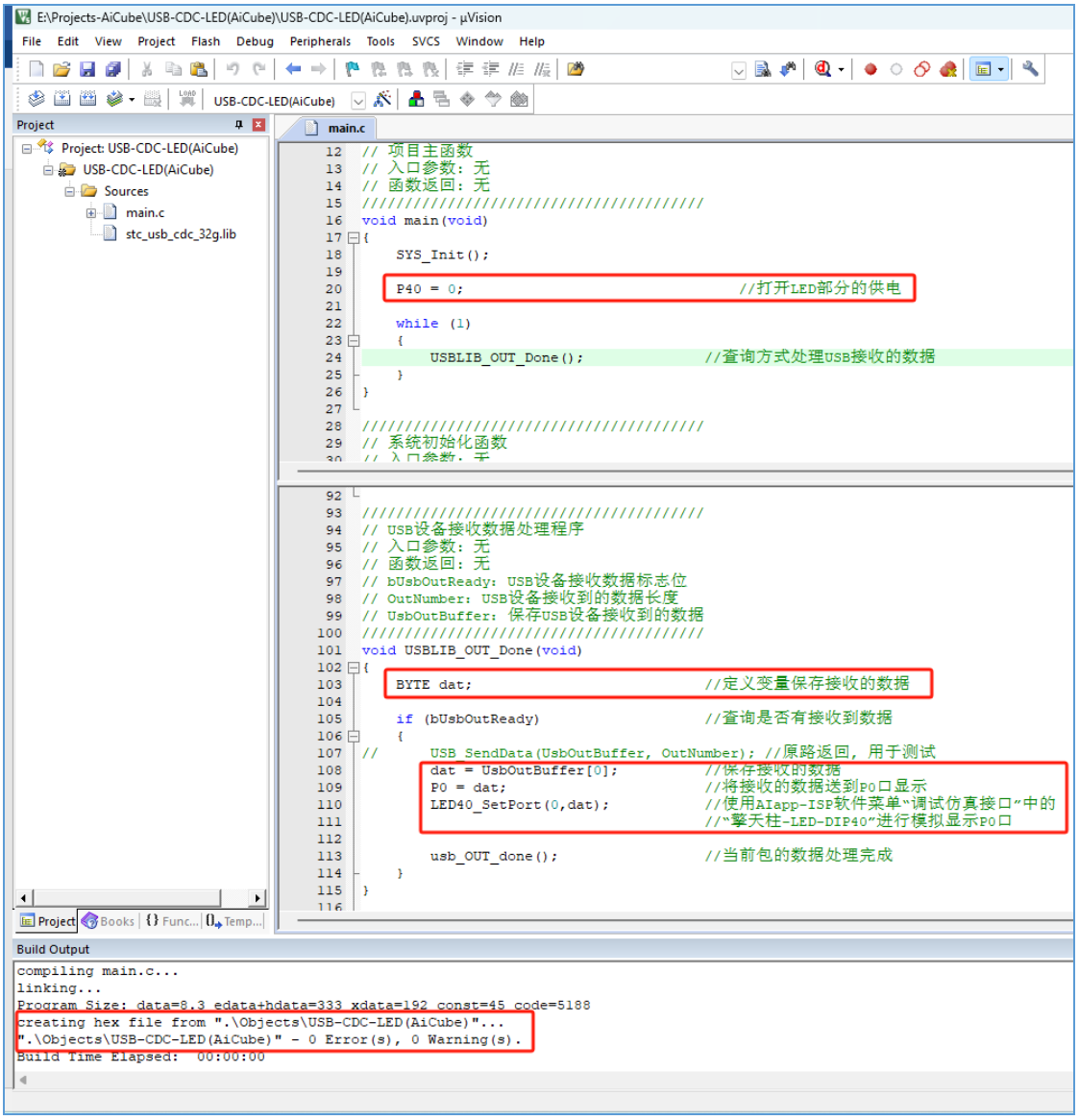
双击打开项目文件：USB-CDC-LED(AiCube).uvproj，如下图：



根据在【AiCube】助手中的配置，从上图中我们看到【AiCube】助手创建的本实验项目中不仅有了 main.c，还已经自动加载了 USB 库函数（stc_usb_cdc_32g.lib）。main.c 中按模块化设计，已经包含了：必要的头文件、主函数、系统初始化函数、延时函数、USB 相关的配置及处理函数。

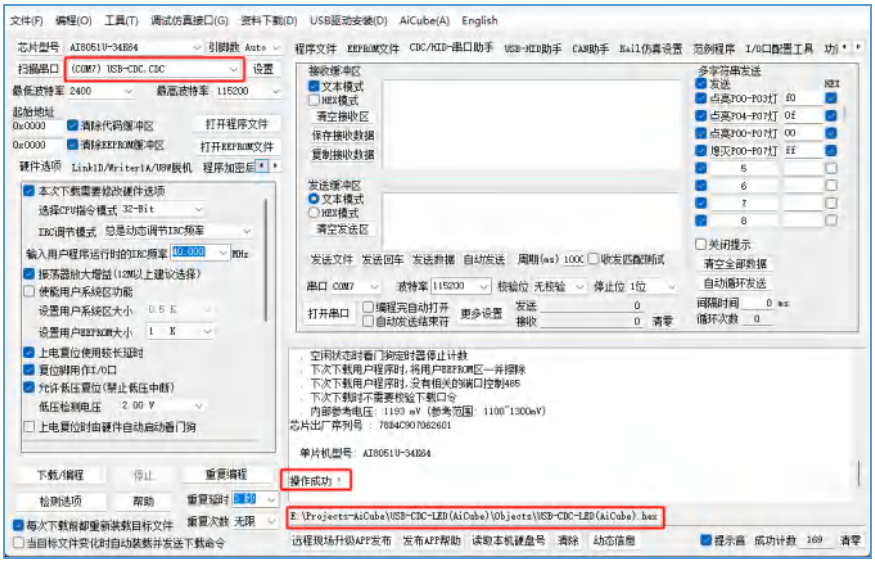
d) 增加实验项目代码

为了实现 USB-CDC 方式控制 AI8051U 进行 LED 点灯的功能，我们现在只需要在主函数中加入“P40 = 0”打开 LED 供电的代码，在“USB 设备接收数据处理程序”子函数中补充 USB 控制代码，保存，编译，如下图（为方便查看，使用菜单“Window”→“Split”将原窗口分裂出两个窗口，对比观察）：



e) 使用 AIapp-ISP 软件系统和 Ai8051U 实验箱查验实验效果

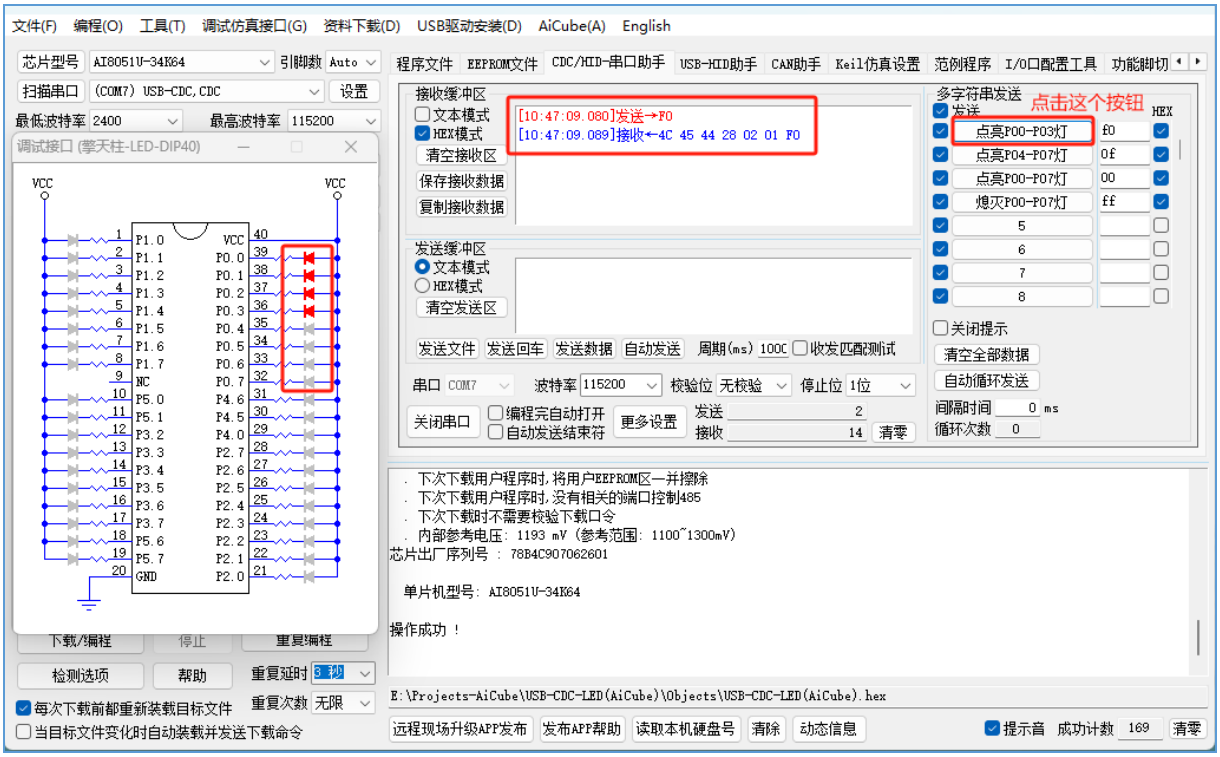
连接好 Ai8051U 实验箱，运行 AIapp-ISP 软件系统，按步骤打开并下载程序“USB-CDC-LED(AiCube).hex”，如下图：



“CDC/HID-串口助手”选项卡中，在“多字符串发送”窗口，设置好发送控制命令，同时打开 AIapp-ISP 软件的“调试仿真接口”菜单，选择“擎天柱-LED-DIP40”以便观察更直观的仿真效果：

1、点击“点亮 P00-P03 灯”这个按钮，观察实验现象：

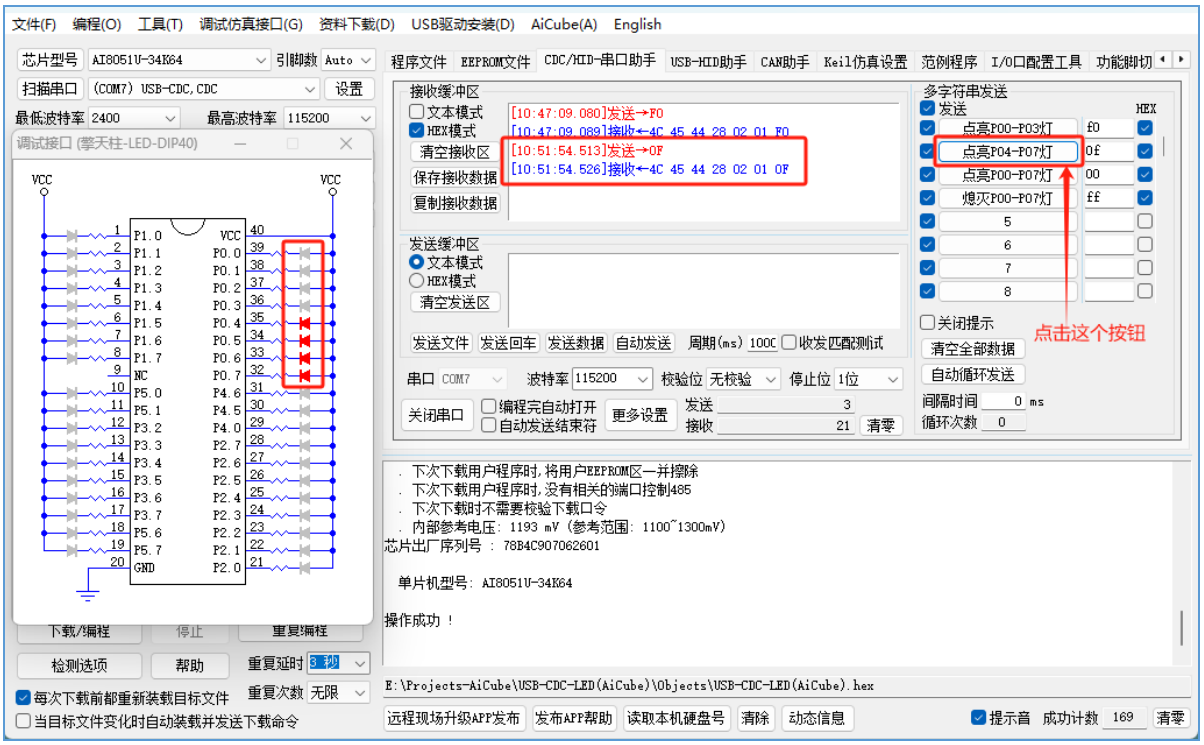
AIapp-ISP 软件的“CDC/HID-串口助手”界面，如下图：



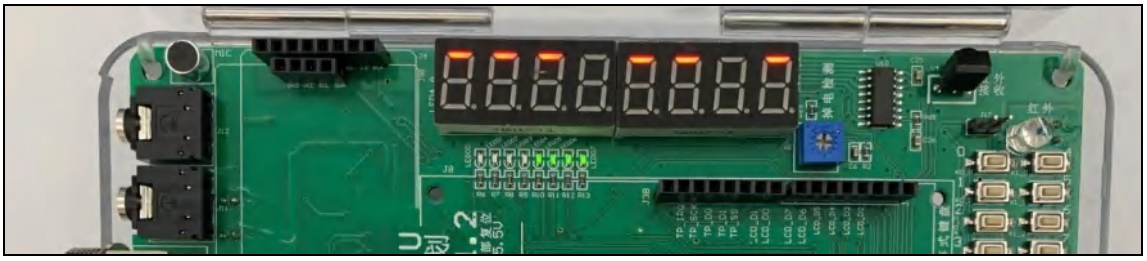
实验箱的 8 个 LED 灯，P00-P03 这 4 个 LED 灯被点亮，如下图：

2、点击“点亮 P04-P07 灯”这个按钮，观察实验现象：

AIapp-ISP 软件的“CDC/HID-串口助手”界面，如下图：



实验箱的 8 个 LED 灯，P04-P07 这 4 个 LED 灯被点亮，如下图：



有了【AiCube】助手，单片机编程就是这么简单！快用起来吧！

2.17.6　USB 不停电下载，ISP 软件如何设置

在开发过程中，程序经常需要反复修改，重新编译后生成新的 HEX 文件，就需要停电/上电进入下载模式，很不方便。其实可以利用我们提供的 USB 不停电 ISP 下载功能，进行自动下载。

在有 USB 功能的实验中，都要加载 USB 库函数，并对 USB 进行初始化。由于在 USB 库函数的代码中已经定义了不停电 ISP 下载命令"@STCISP#"，只需要在下载软件的"收到用户命令后复位到 ISP 监控程序区"的选项卡中进行如下设置，即可实现【USB 不停电下载】功能。

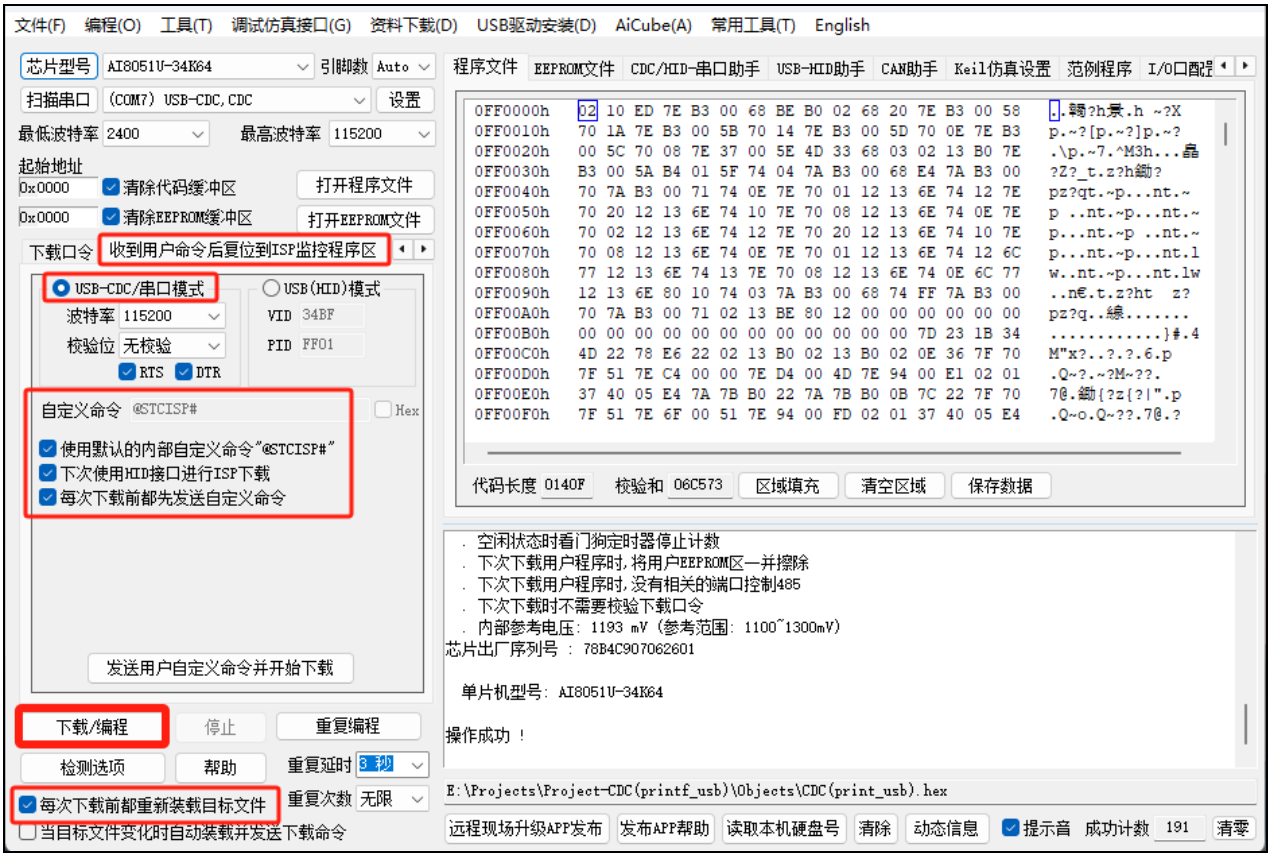
- 如使用 USB-CDC 库函数产生 USB-CDC 虚拟串口，可实现 USB-CDC 不停电自动 ISP 下载功能，并可忽略传统串口的波特率、校验位等参数的设置，下载时系统程序区还是使用的 USB-HID 通讯协议；
- 如果是传统的串口，需正确设置波特率、校验位等参数，才能实现串口的不停电自动 ISP 下载功能；
- 如果使用 USB-HID 库函数产生 USB-HID 设备，则选择"USB（HID）模式"就能实现不停电自动 ISP 下载功能，下载时系统程序区还是使用的 USB-HID 通讯协议。

ISP 软件如何设置：

2.17.6.1　手动点击【下载/编程】按钮，ISP 软件自动重新打开文件，不停电下载

如下图设置，就可以实现：

在点击"下载/编程"按钮后，ISP 软件会重新调入已打开的目标文件一次（不管目标文件有无变化），接着通过 USB-CDC 通讯协议自动发送自定义下载命令如"@STCISP#"给单片机，单片机中的用户软件收到该命令后，就会软复位到单片机的系统监控程序区，等待 ISP 下载，本设置还指定了电脑端软件后续是通过 USB-HID 通讯协议传输用户程序到目标单片机进行 ISP 下载。

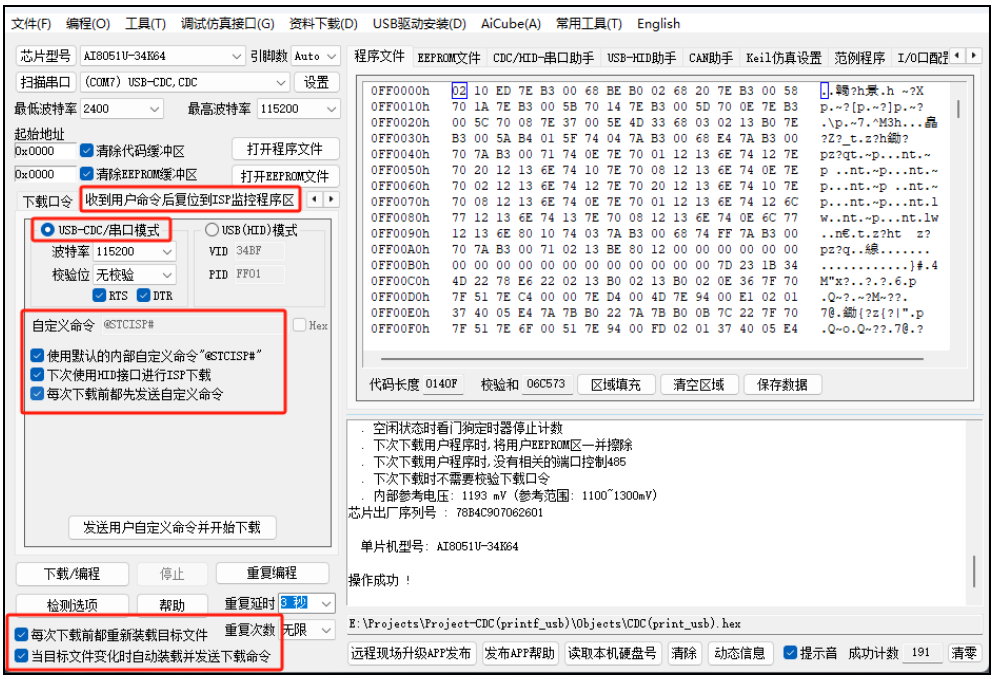


如上所示，不停电下载实验是成功的，但本设置还需要手工点击一次"下载/编程"按钮。

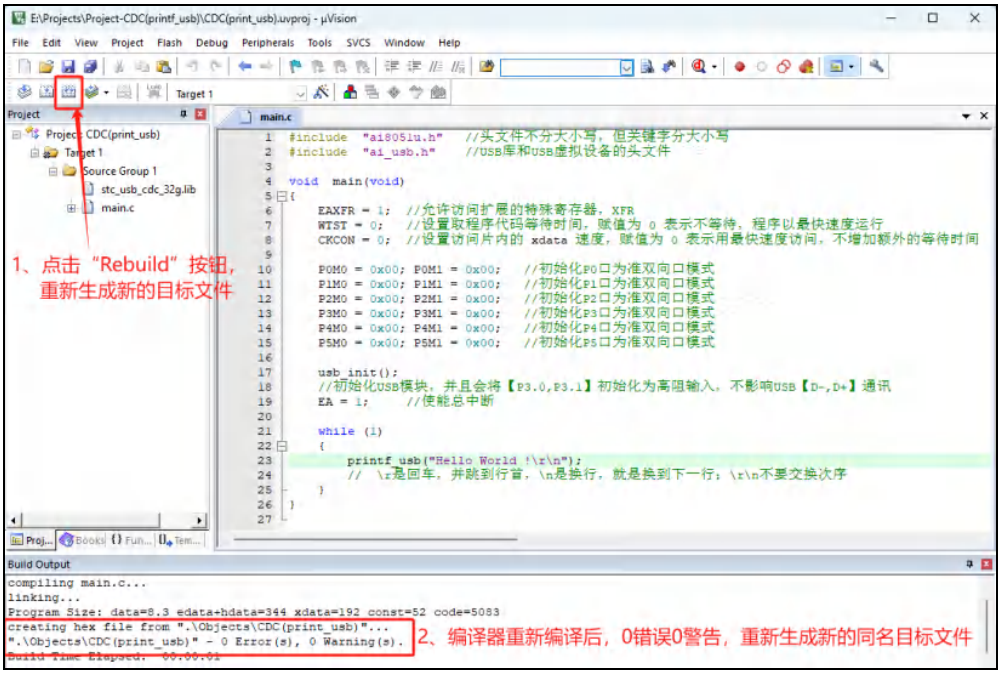
2.17.6.2 编译器重新编译后，ISP 软件会自动重新调入文件，进行不停电下载

如下图设置：

下载软件会每秒检测多次目标文件被编译的时间是否已更新，如时间已更新，则 ISP 软件会自动重新调入已打开的目标文件，接着通过 USB-CDC 通讯协议自动发送自定义下载命令如"@STCISP#"给单片机，单片机中的用户软件收到该命令后，就会软复位到单片机的系统监控程序区，等待 ISP 下载，本设置还指定了电脑端软件后续是通过 USB-HID 通讯协议传输用户程序到目标单片机进行 ISP 下载。



如上所示，不停电下载实验是成功的，但本设置的优点是不需要手工点击"下载/编程"按钮。

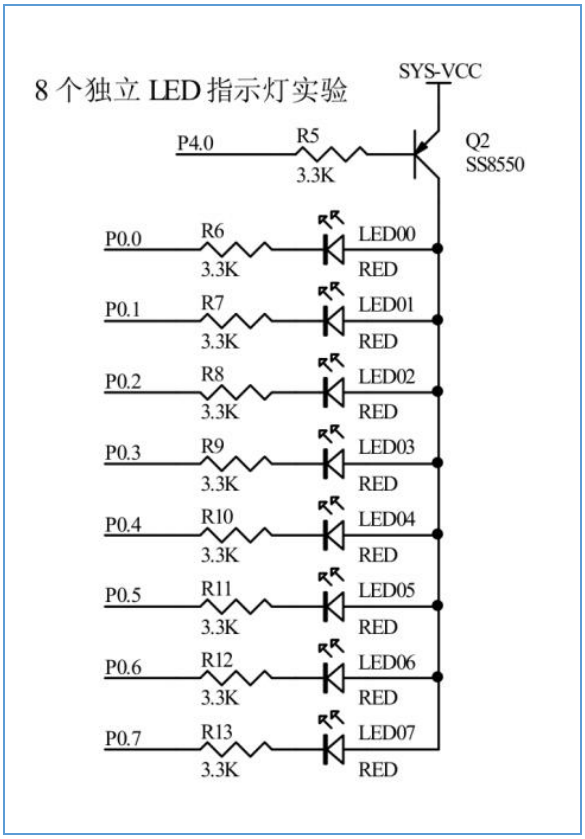


在编译集成环境中，点击"Rebuild"按钮，目标文件会被重新编译生成一次，ISP 软件发现目标文件被编译的时间发生变化后，ISP 软件就会执行前面描述的不停电下载动作。

2.18　点灯实验四：使用 Ai8051U 内置 USB 直接仿真流水灯

本实验相应的视频链接：https://www.stcaimcu.com/forum.php?mod=viewthread&tid=15176#lastpost
<Ai8051U 实验箱的全部原理图可以通过点击本段文字进行查看>

2.18.1　原理介绍，在 Ai8051U 实验箱中有如下的一排 LED 指示灯



I/O 口 P4.0 控制 PNP 三极管的通断，控制这 8 个灯的整体供电。
如果 P4.0 输出 1，则 PNP 三极管不导通，这 8 个灯都不会获得电源，都不会有电流流过，都不会被点亮。
如果 P4.0 输出 0，则 PNP 三极管导通，这 8 个灯才有可能会流过电流，才有可能会被点亮。
如 P4.0 输出 0，PNP 三极管导通，特定 LED 灯的控制 I/O 口如输出 0，则该 LED 灯会有电流流过，会被点亮。
如 P0.0 输出 0，LED00 灯就会被点亮；如 P0.0 输出 1，LED00 灯就会熄灭；
如 P0.1 输出 0，LED01 灯就会被点亮；如 P0.1 输出 1，LED01 灯就会熄灭；
如 P0.2 输出 0，LED02 灯就会被点亮；如 P0.2 输出 1，LED02 灯就会熄灭；
如 P0.3 输出 0，LED03 灯就会被点亮；如 P0.3 输出 1，LED03 灯就会熄灭；
如 P0.4 输出 0，LED04 灯就会被点亮；如 P0.4 输出 1，LED04 灯就会熄灭；
如 P0.5 输出 0，LED05 灯就会被点亮；如 P0.5 输出 1，LED05 灯就会熄灭；
如 P0.6 输出 0，LED06 灯就会被点亮；如 P0.6 输出 1，LED06 灯就会熄灭；
如 P0.7 输出 0，LED07 灯就会被点亮；如 P0.7 输出 1，LED07 灯就会熄灭；

如 P0 口对外全部送 0，则这 8 个灯全亮；
如 P0 口对外全部送 1，则这 8 个灯全灭；
如果 P4.0 输出 1，则 PNP 三极管不导通，不管 P0 口对外全部送 0 或送 1，这 8 个灯都不会有电流流过，都不会亮。P4.0 控制 PNP 三极管，作为这 8 个 LED 指示灯的总电源开关。

2.18.2　实验程序 C 语言代码

```
//使用 Ai8051U 内置 USB 直接复制流水灯实验
#include <AI8051U.H>                    //包含 AI8051U 的头文件
#include "intrins.h"                    //使用_nop_()函数所必须要包含的头文件,
                                        //否则延时函数中调用的_nop_()函数没有被头文件引用过来，
                                        //会导致编译器找不到这个而函数而报错。//unsigned char cnt;

void Delay500ms(void)                   //@40.000MHz
{
    unsigned long edata i;

    _nop_();
    _nop_();
    i = 4999998UL;
    while (i) i--;
}


void main(void)
{
    EAXFR = 1;              //允许访问扩展的特殊寄存器，XFR
    WTST = 0;               //设置取程序代码等待时间，赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;              //设置访问片内的 xdata 速度，赋值为 0 表示用最快速度访问，
                            //不增加额外的等待时间
    P0M0 = 0x00; P0M1 = 0x00;       //设置 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x00;       //设置 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;       //设置 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x00;       //设置 P3 口为准双向口模式
    P4M0 = 0x00; P4M1 = 0x00;       //设置 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;       //设置 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;       //设置 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;       //设置 P7 口为准双向口模式

    P40 = 0;                        //打开 LED 部分的供电
    while(1)
    {                               //实现 LED 流水灯
        P0 = 0;                     //对 P0 端口置 0，点亮整组 LED
        Delay500ms();

        P0 = 0xff;                  //对 P0 端口置 ff，全灭整组 LED
        Delay500ms();

        P00 = 0;                    //对单个端口 P00 置 0，点亮单个 LED，这个点亮是的 LED0
        Delay500ms();
        P00 = 1;                    //对单个端口 P00 置 1，熄灭单个 LED，这个熄灭是的 LED0
```

```
        Delay500ms();

        P01 = 0;                    //对单个端口 P01 置 0，点亮单个 LED，这个点亮是的 LED1
        Delay500ms();
        P01 = 1;                    //对单个端口 P01 置 1，熄灭单个 LED，这个熄灭是的 LED1
        Delay500ms();

        P02 = 0;                    //对单个端口 P02 置 0，点亮单个 LED，这个点亮是的 LED2
        Delay500ms();
        P02 = 1;                    //对单个端口 P02 置 1，熄灭单个 LED，这个熄灭是的 LED2
        Delay500ms();

        P03 = 0;                    //对单个端口 P03 置 0，点亮单个 LED，这个点亮是的 LED3
        Delay500ms();
        P03 = 1;                    //对单个端口 P03 置 1，熄灭单个 LED，这个熄灭是的 LED3
        Delay500ms();

        P04 = 0;                    //对单个端口 P04 置 0，点亮单个 LED，这个点亮是的 LED4
        Delay500ms();
        P04 = 1;                    //对单个端口 P04 置 1，熄灭单个 LED，这个熄灭是的 LED4
        Delay500ms();

        P05 = 0;                    //对单个端口 P05 置 0，点亮单个 LED，这个点亮是的 LED5
        Delay500ms();
        P05 = 1;                    //对单个端口 P05 置 1，熄灭单个 LED，这个熄灭是的 LED5
        Delay500ms();

        P06 = 0;                    //对单个端口 P06 置 0，点亮单个 LED，这个点亮是的 LED6
        Delay500ms();
        P06 = 1;                    //对单个端口 P06 置 1，熄灭单个 LED，这个熄灭是的 LED6
        Delay500ms();

        P07 = 0;                    //对单个端口 P07 置 0，点亮单个 LED，这个点亮是的 LED7
        Delay500ms();
        P07 = 1;                    //对单个端口 P07 置 1，熄灭单个 LED，这个熄灭是的 LED7
        Delay500ms();

//      P0 = (1<<(cnt%8));          //将对应亮起的位赋值给 P0 口
//      cnt++;                      //每次步进 1
//      Delay500ms();
    }
}
```

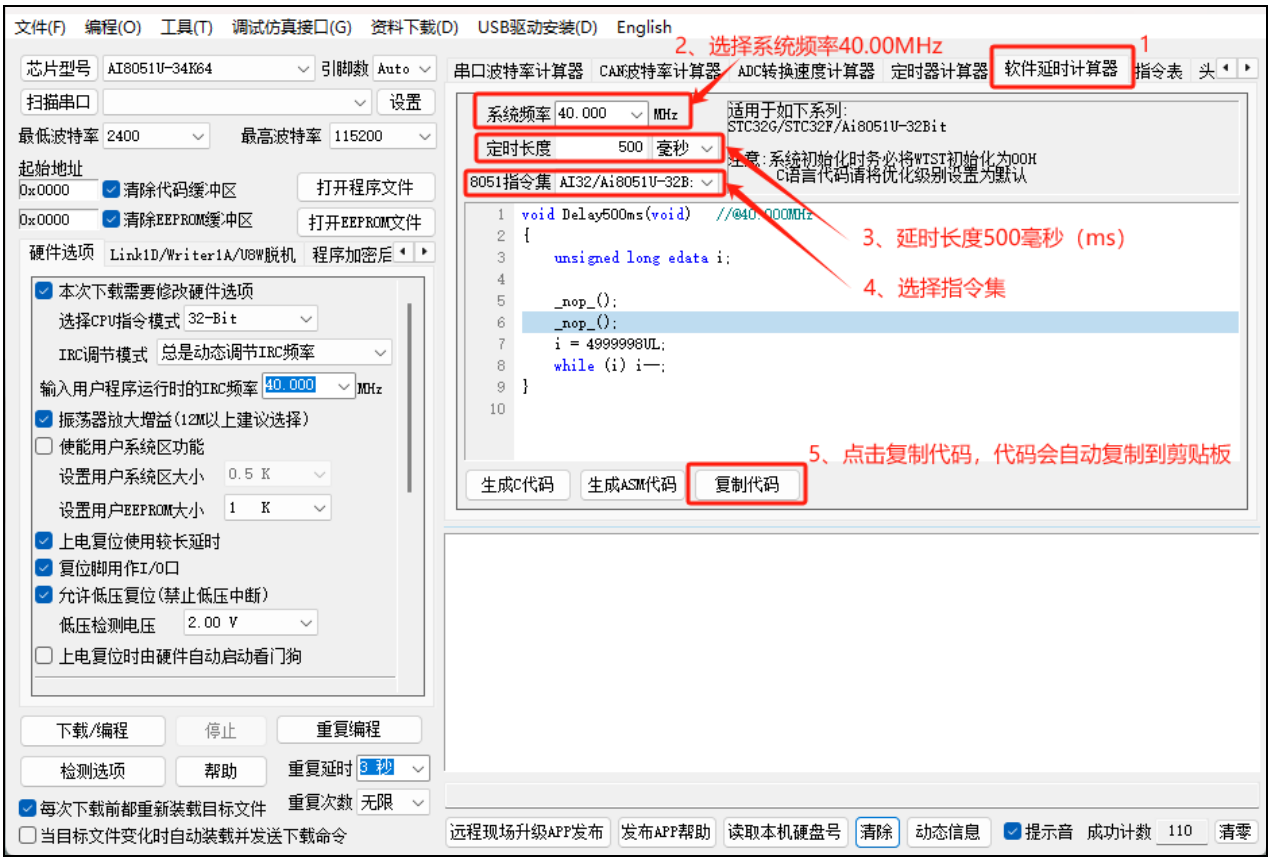
【小技巧--灵活运用 AIapp-ISP 软件的工具自动生成代码】

延时函数部分代码就可以使用 **AIapp-ISP** 软件的“**软件延时计算器**”来生成。

例如上述代码中这一段延时函数的代码：

```
void Delay500ms(void)          //延时函数，主频@40MHz
{
    unsigned long edata i;     //定义变量
    _nop_();                   //空指令
    _nop_();
    i = 4999998UL;             //变量初始化
    while (i) i--;             //循环等待
}
```

就可以使用 AIapp-ISP 软件的“软件延时计算器”来生成：



将复制出来的内容直接粘贴到代码文件中，不用手动输入。

【小技巧--灵活运用 AIapp-ISP 软件的工具自动生成代码】

I/O 口的初始化部分就可以使用 **AIapp-ISP** 软件的“**I/O 口配置工具**”来生成。

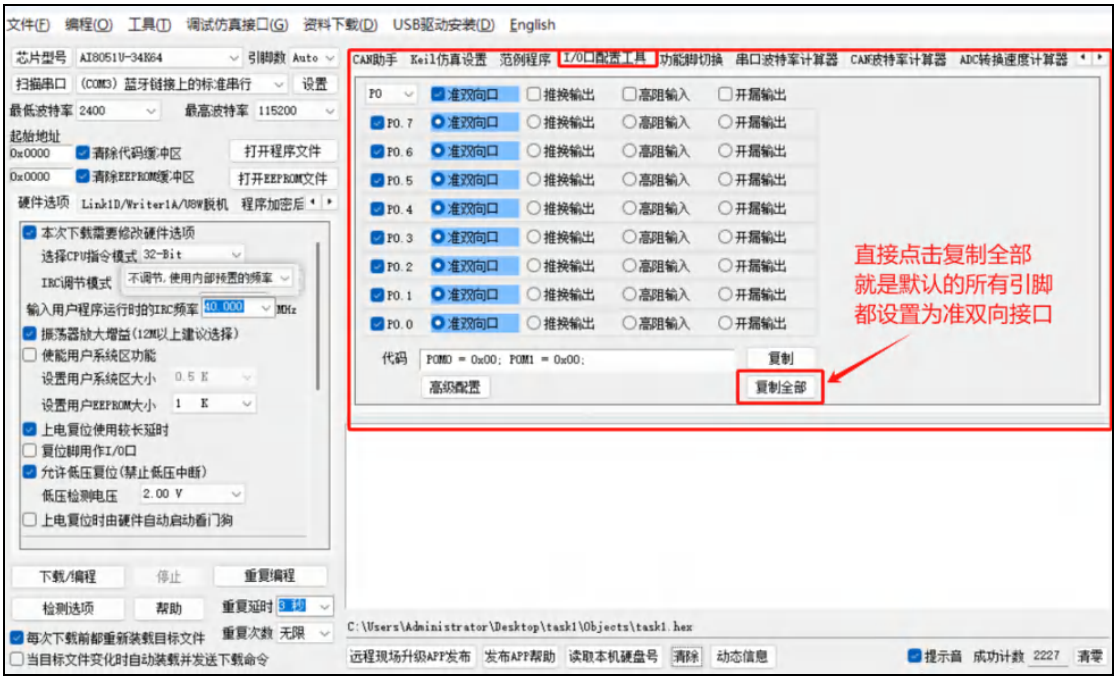
例如上述代码中这一段将所有 I/O 口设置为准双向口模式的代码：

```
P0M0 = 0x00; P0M1 = 0x00;
P1M0 = 0x00; P1M1 = 0x00;
P2M0 = 0x00; P2M1 = 0x00;
P3M0 = 0x00; P3M1 = 0x00;
P4M0 = 0x00; P4M1 = 0x00;
P5M0 = 0x00; P5M1 = 0x00;
```

P6M0 = 0x00; P6M1 = 0x00;

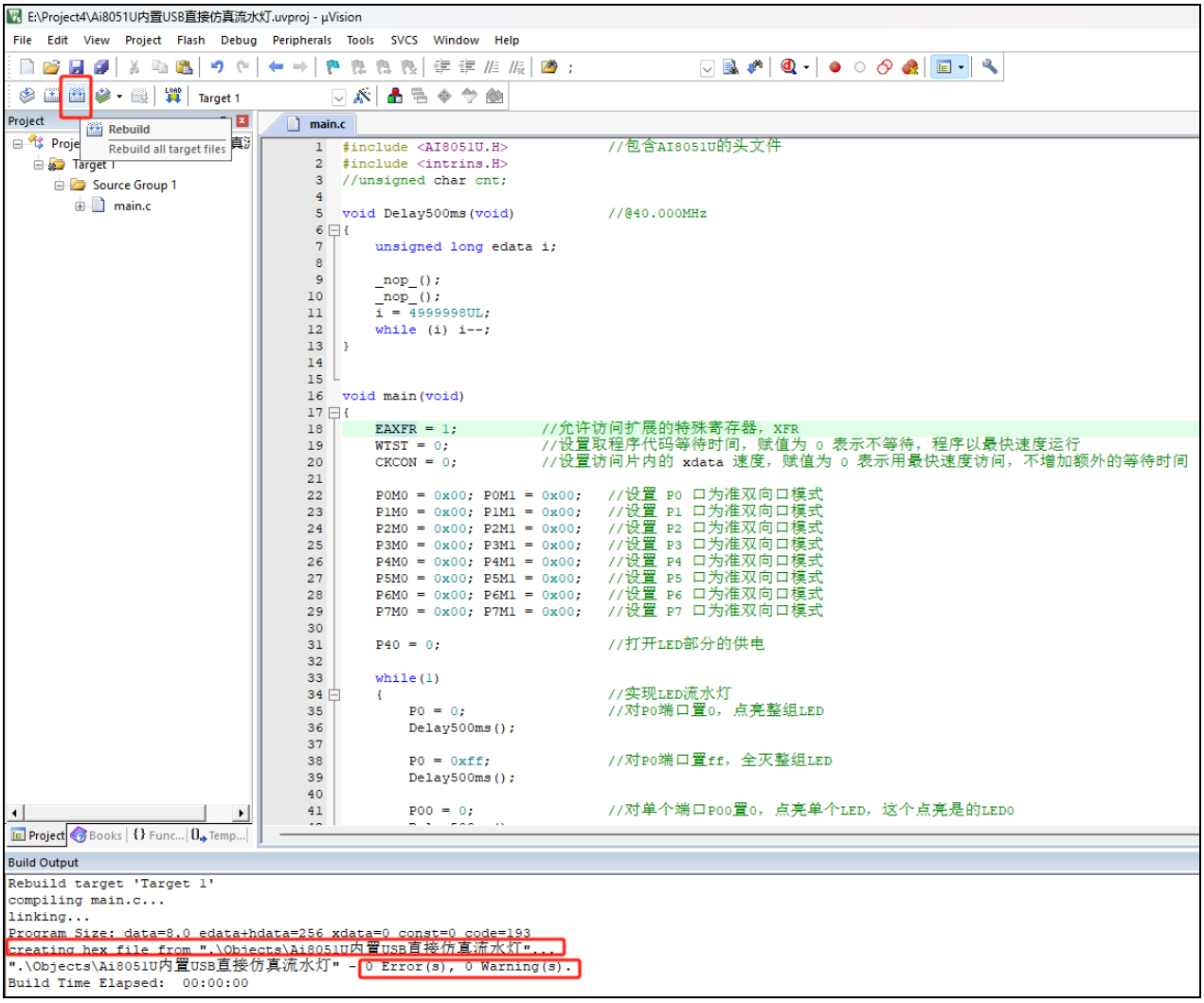
P7M0 = 0x00; P7M1 = 0x00;

就可以使用 AIapp-ISP 的 I/O 口配置工具进行生成：



将复制出来的内容直接粘贴到代码文件中，不用手动输入。

代码编写完后，点击工具栏上的 Rebuild 按钮进行编译，如下图：

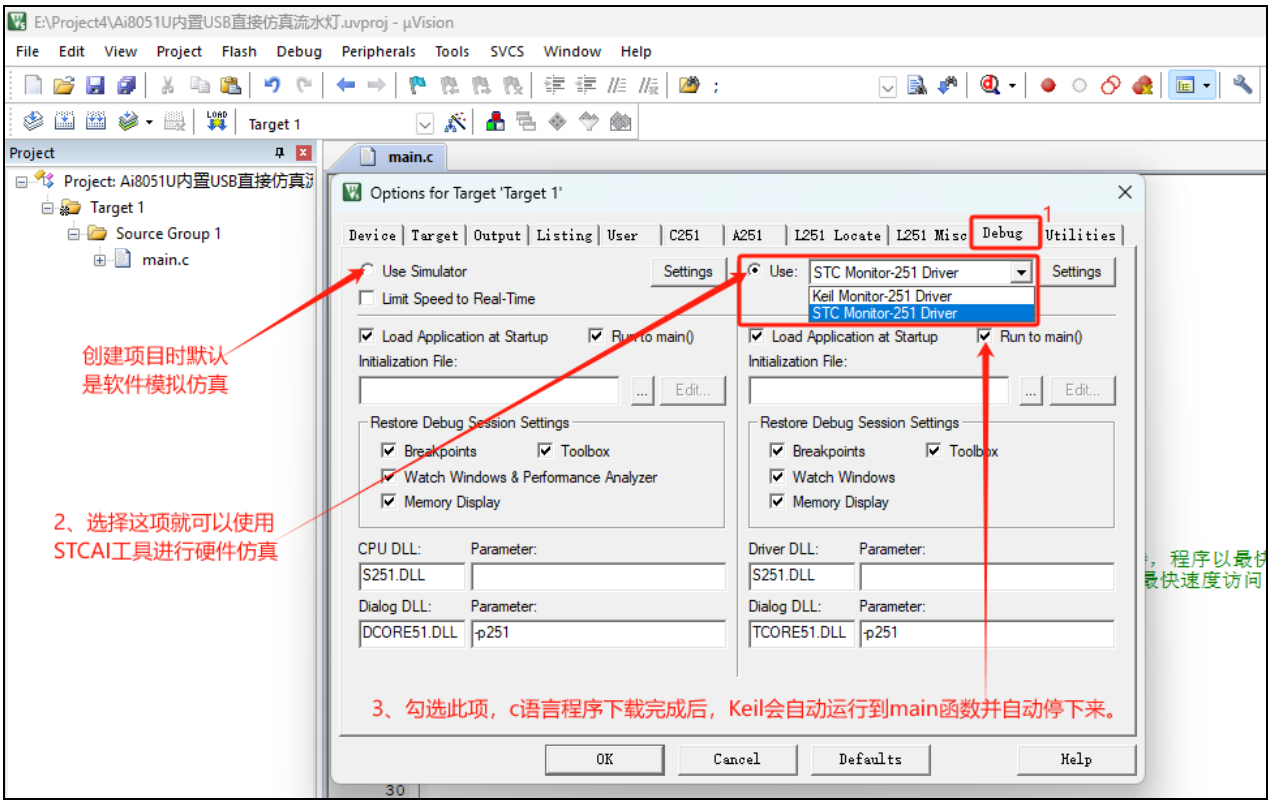


出现 creating hex file 和 0 Error(s) 0 Warning（s）即为编译成功，成功创建了 hex 文件且无报错。
此 hex 文件，一般是在对应工程目录下的 Objects 文件夹里。

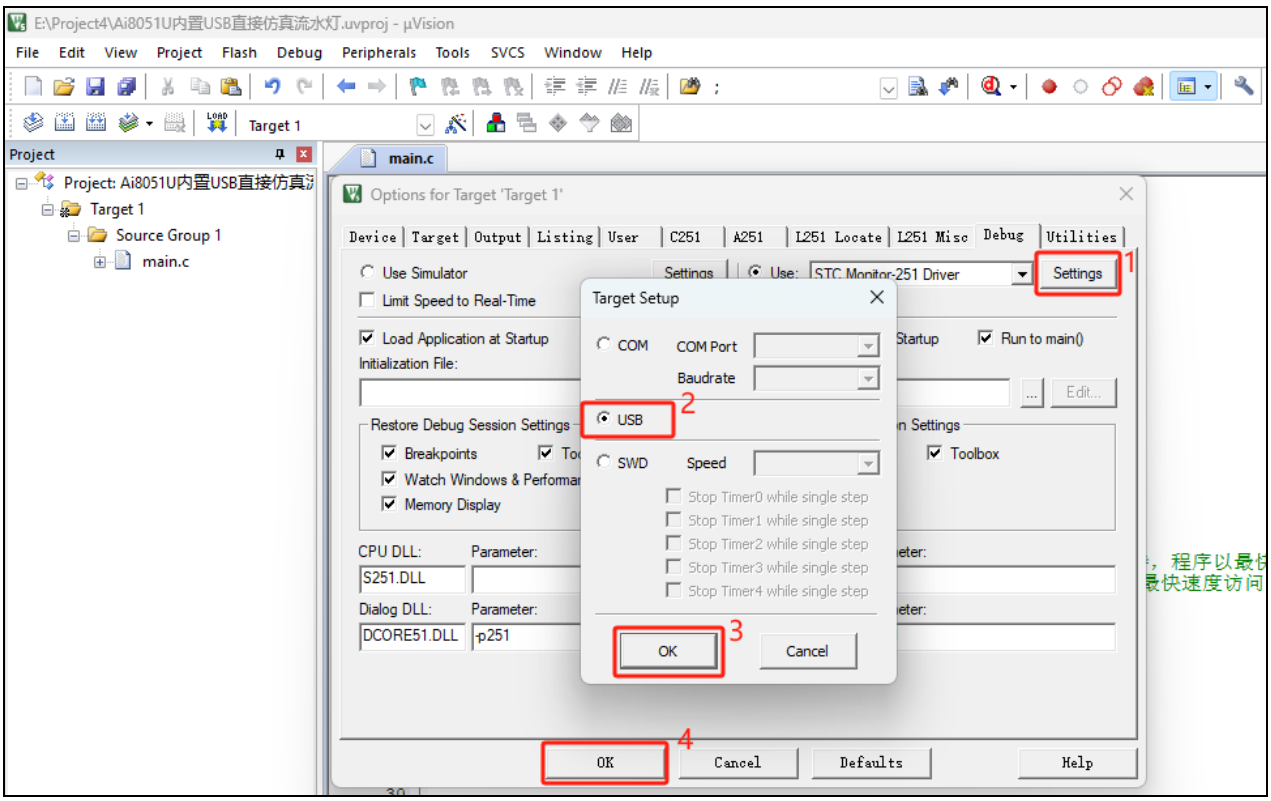
本项目是使用 AI8051U 内置 USB 直接仿真流水灯，除了新建项目时必要设置的基础选项外，还要设置：Debug 选项卡

点击工具栏上的魔术棒的工具，打开“Options for Target ‘Target 1’”任务选项面板，选择 Debug 选项卡：

选择硬件仿真：系统默认是软件模拟仿真，本项目是使用 AI8051U 内置 USB 直接仿真流水灯，属于硬件仿真，要选择“STC Monitor-251 Driver”



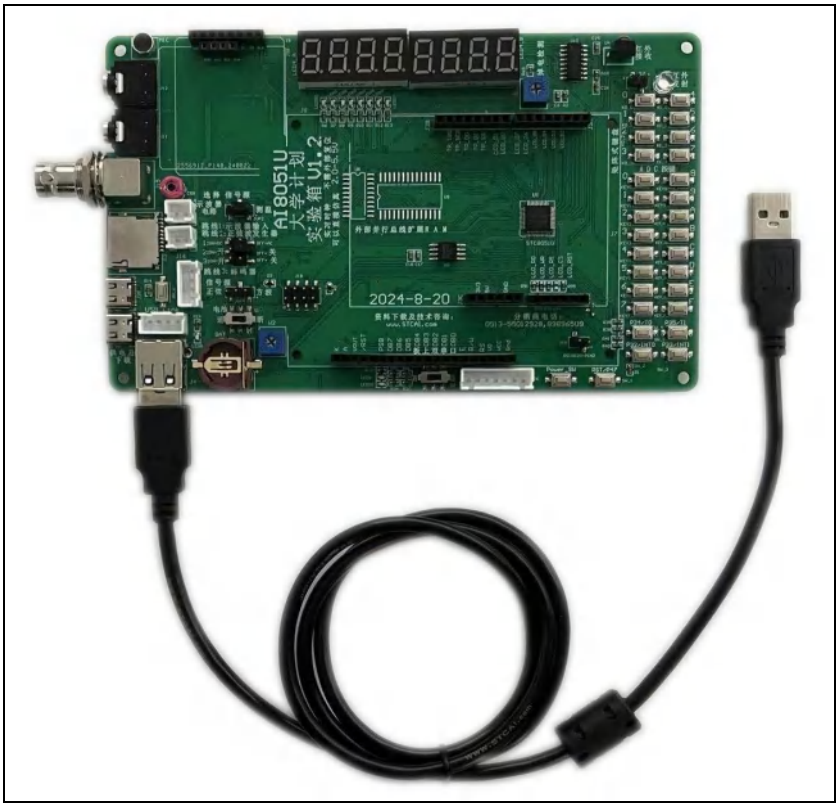
选择仿真接口：Ai8051U 芯片内置的硬件 USB 支持直接仿真，本项目要选择 USB 为仿真接口。



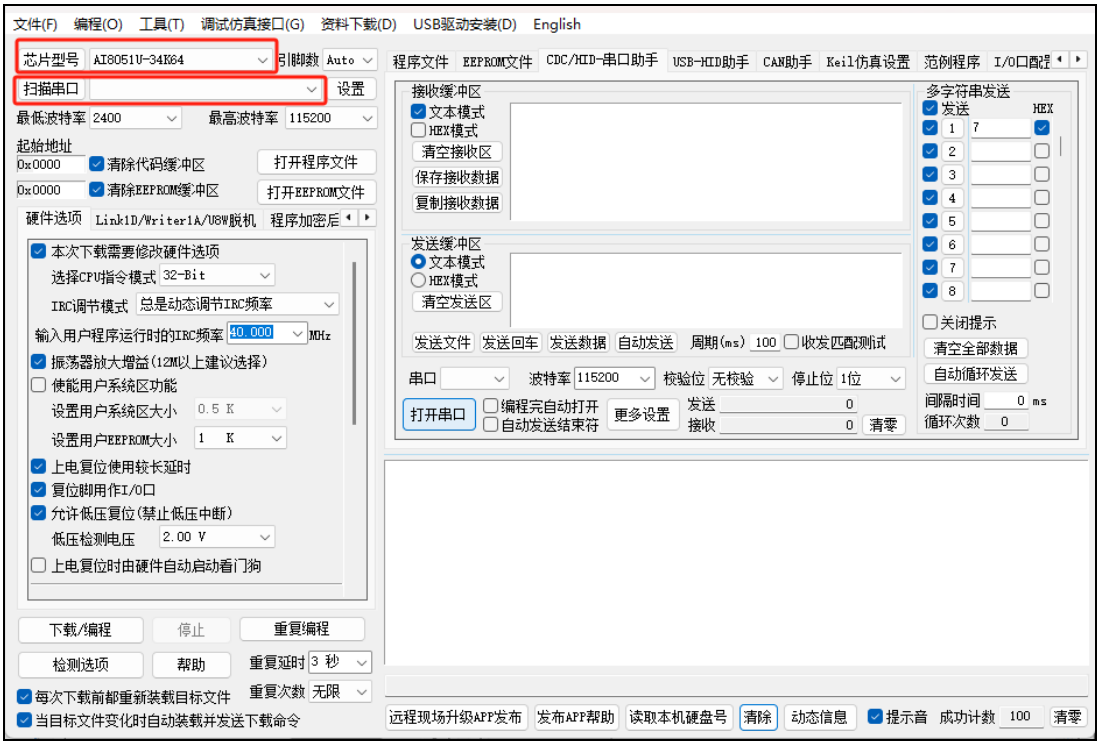
2.18.3　下载用户程序到实验箱，实物图

2.18.3.1　连接到实验箱

首先使用实验箱中配备的 USB-TypeA 到 USB-TypeA 线将实验箱与电脑正确连接。

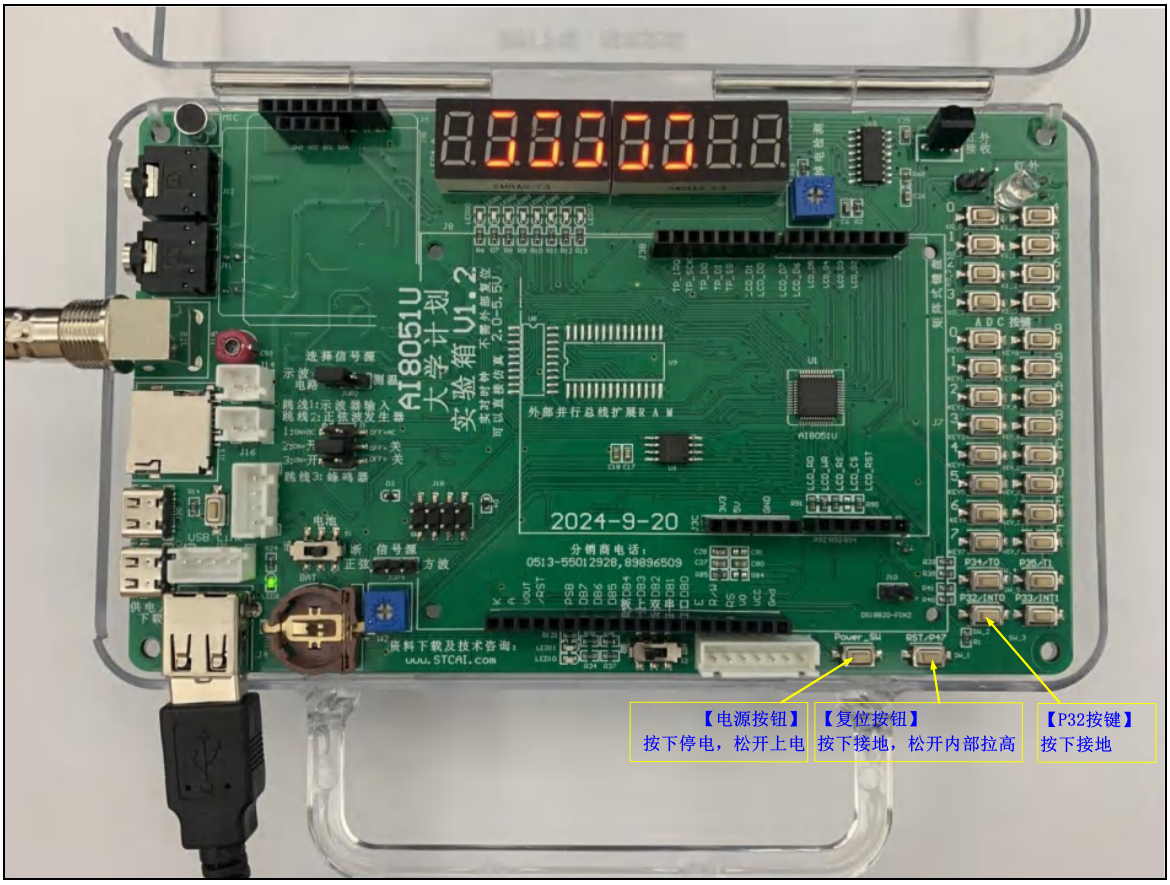


然后打开 AIapp-ISP-v6.95F 或以上版本。选择单片机型号：Ai8051U-34K64，如下图，我们看到“扫描串口”后面的端口列表框不是“（HID1）USB-Writer”，说明还不能下载程序到实验箱。

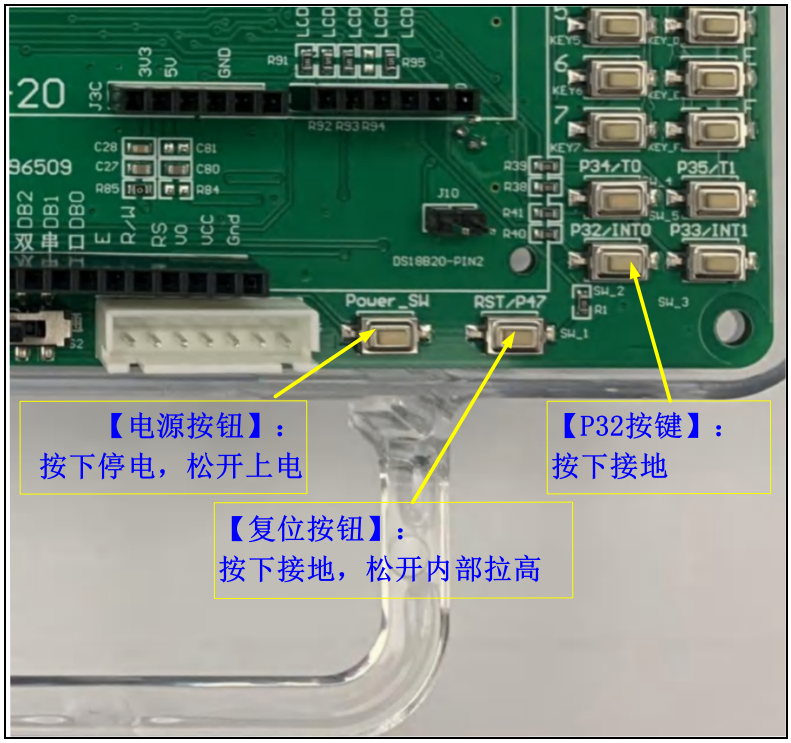


2.18.3.2 硬件 USB 下载用户程序步骤

下图是实验箱已经和电脑相连的正面图片，数码管是随机显示的，数码管下面的 8 个 LED 灯都是暗淡无光的，此实验的目的是使用 AI8051U 内置 USB 直接仿真流水灯，并观察实验现象。



实验箱正下方有两排按键，找到 P3.2 按键和 Power_SW 按键，按操作步骤，使 Ai8051U 实验箱上电，并使得 AIapp-ISP 软件界面“扫描串口”的端口列表框显示“(HID1) USB Writer”。



【方法一，P3.2 按键，再结合停电上电下载】

1、按下板子上的 P3.2 按键，就是 P3.2 接地

2、给目标芯片重新上电，不管之前是否已通电。

===本 Ai8051U 实验箱是：按下电源按钮停电，再松开电源按钮上电，

这时电脑端 ISP 下载软件会自动识别出“(HID1) USB Writer”，此时就与 P3.2 状态无关了，可以松开 P3.2 按键（P3.2 在用户程序区可做其它任意用途）

3、点击电脑端下载软件中的【下载/编程】按钮（注意：USB 下载与串口下载的操作顺序不同）

电脑端 ISP 下载软件提示：下载进行中，几秒钟后，提示下载成功！

【方法二：复位管脚低电平复位下载】

USB 连接好并已上电的情况下，外部按键复位也可进入 USB 下载模式，注意：

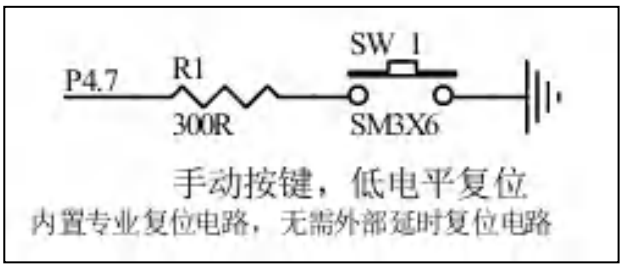
USB 连接好并已上电的情况下，外部按键复位也可进入 USB 下载模式，注意：

P4.7-nRST 出厂时默认是 P4.7-I/O 功能，要改为复位功能，需 ISP 烧录时取消设置复位脚用作 I/O 口，停电一次再上电才生效，程序区中用户程序也可改为复位脚或 I/O，这个立即生效。

1、按下复位管脚/P4.7-nRST 外接的低电平复位按键，复位 MCU，

再松开复位键，MCU 首先从系统程序区启动运行系统程序，判断是否要下载用户程序，

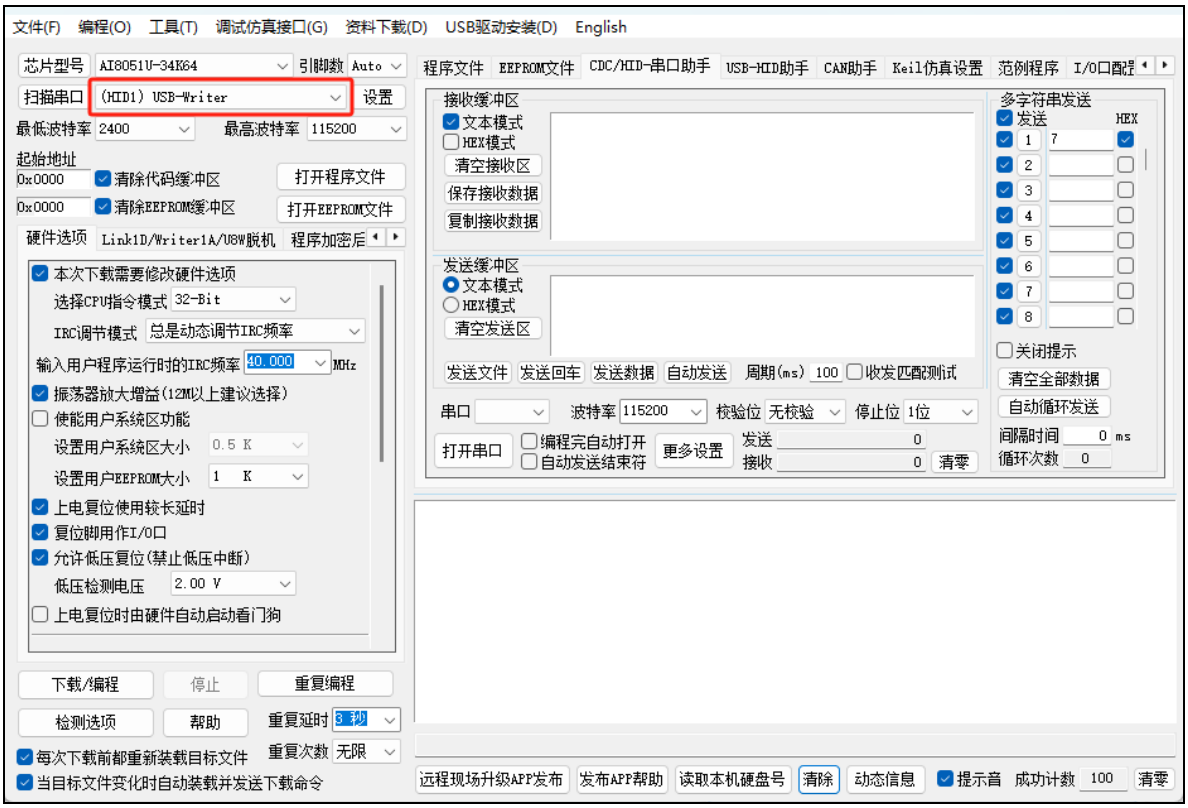
等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后



2、点击电脑端下载软件中的【下载/编程】按钮

电脑端 ISP 下载软件提示：下载进行中…，下载成功！

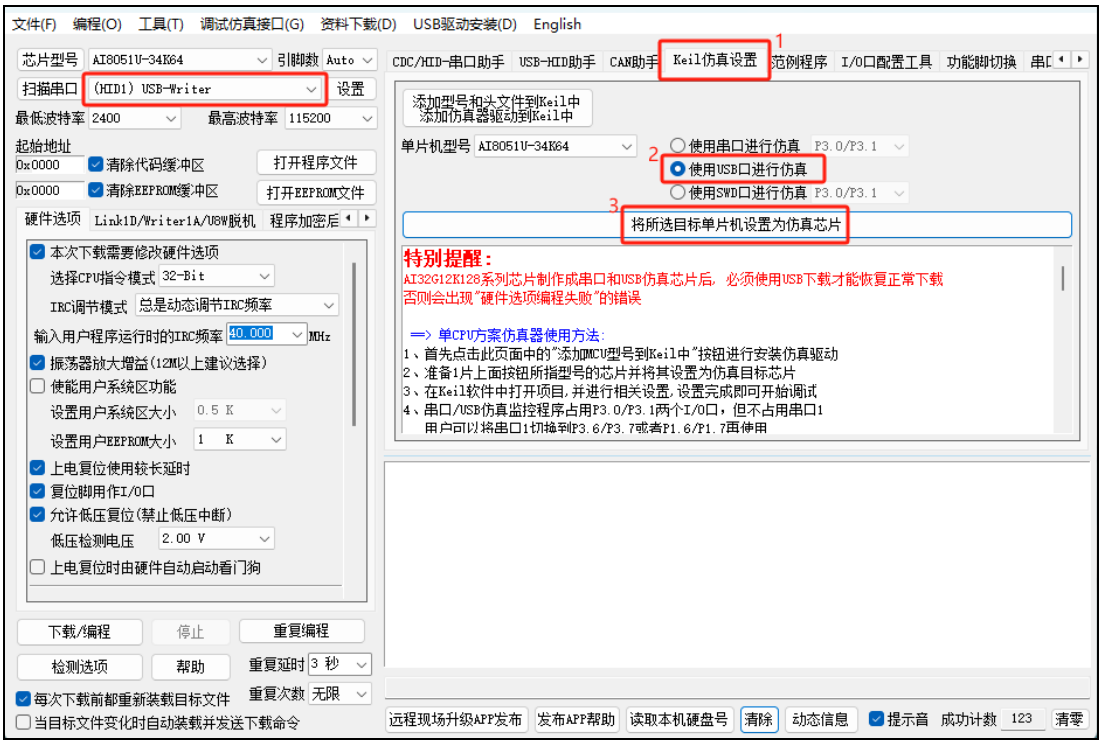
备注：如果目标 MCU 的用户程序区已有 USB 通讯程序（含 CDC、HID 等），则通过复位脚复位进入 USB 下载模式不被支持，只能通过停电上电冷启动下载或者从用户程序区软复位到系统区下载



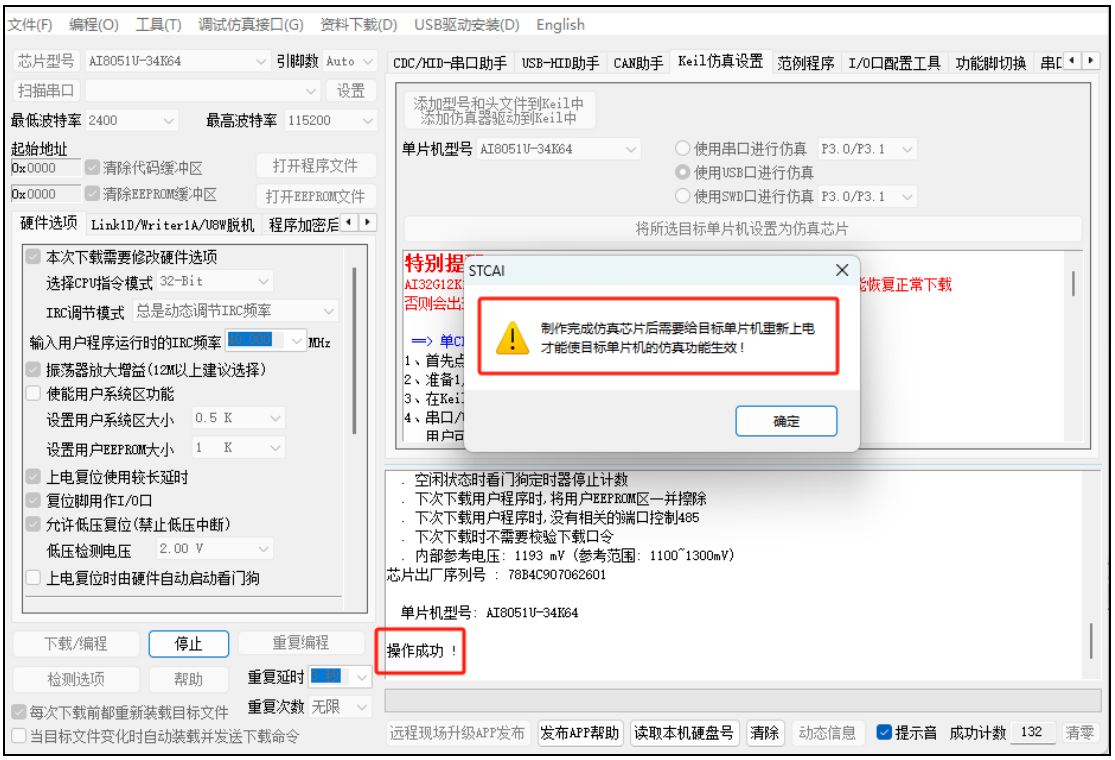
2.18.3.3 制作仿真芯片

在 AIapp-ISP 软件界面，看到扫描串口列表框显示“(HID1) USB Writer”：

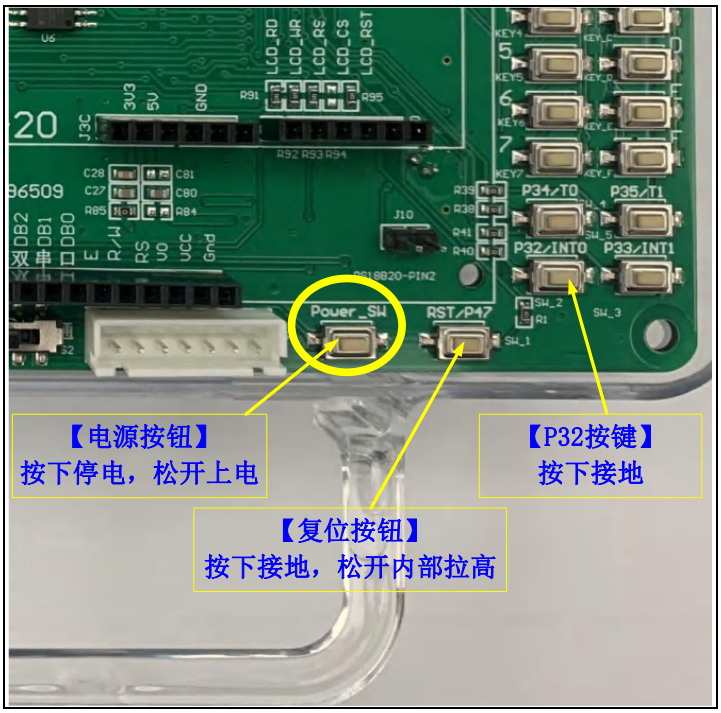
- 点击“Keil 仿真设置”标签;
- 在“Keil 仿真设置”选项卡，选择“使用 USB 口进行仿真”
- 点击“将所选目标单片机设置为仿真芯片”



然后看到 AIapp-ISP 软件在自动进行仿真芯片的设置，界面下面的进度条在推进，然后“操作成功！”界面出现特别提示：“制作完成仿真芯片后需要给目标单片机重新上电才能使目标单片机的仿真生效！”如下图：

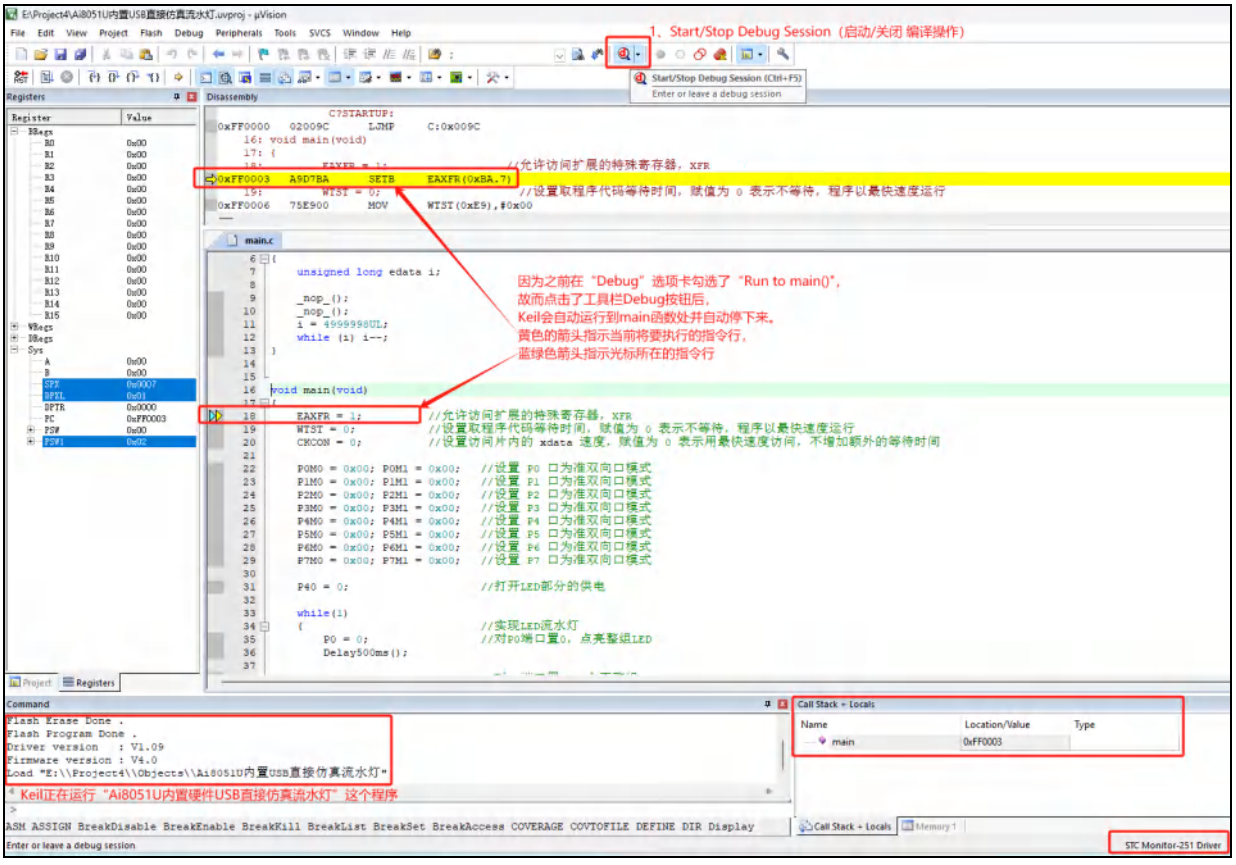


所以，按提示按一下实验箱上的“Power_SW”按钮：



2.18.4　Keil 仿真并观察实验现象

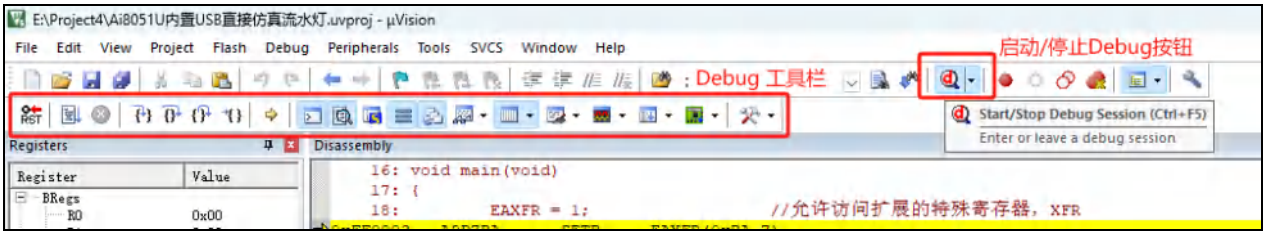
现在，返回 Keil 界面，点击工具栏的 Debug（启动/关闭编译调试操作界面）按钮后，Keil 软件进入仿真界面，如下图：



提示：退出 Debug 调试界面，同样点击工具栏的“Debug”按钮。

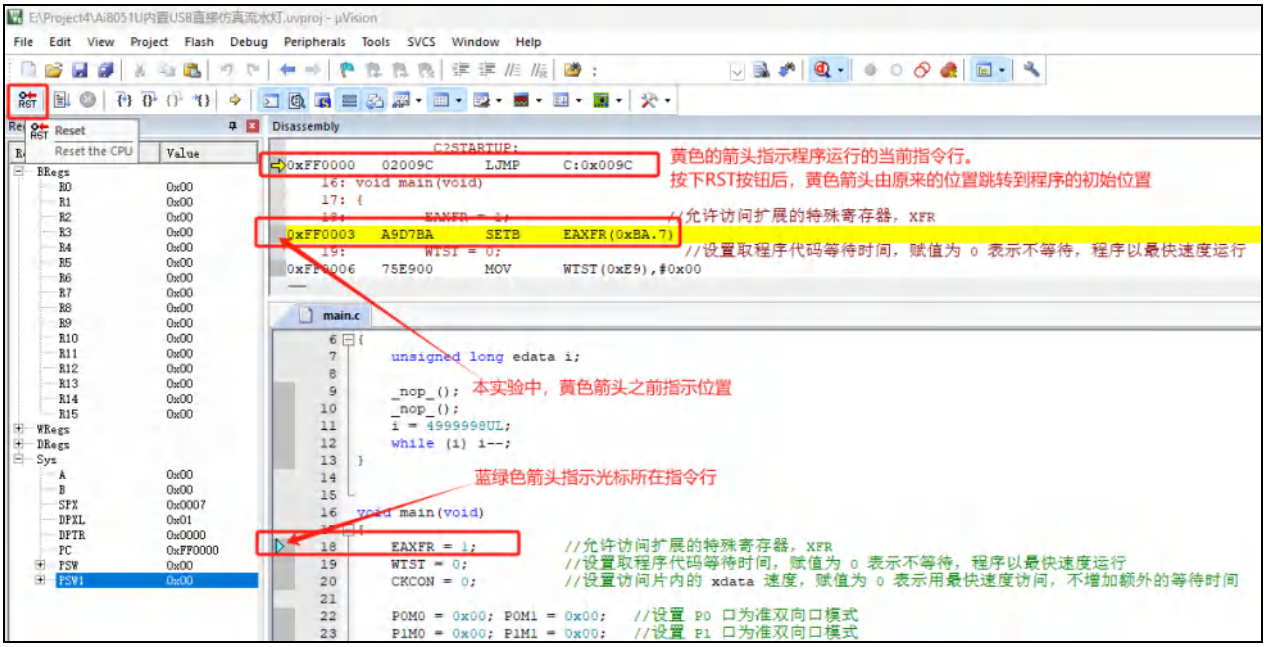
2.18.4.1 认识 Keil 软件的 Debug Toolbar（Debug 工具栏）

首先我们来认识一下 Debug Toolbar（Debug 工具栏）：

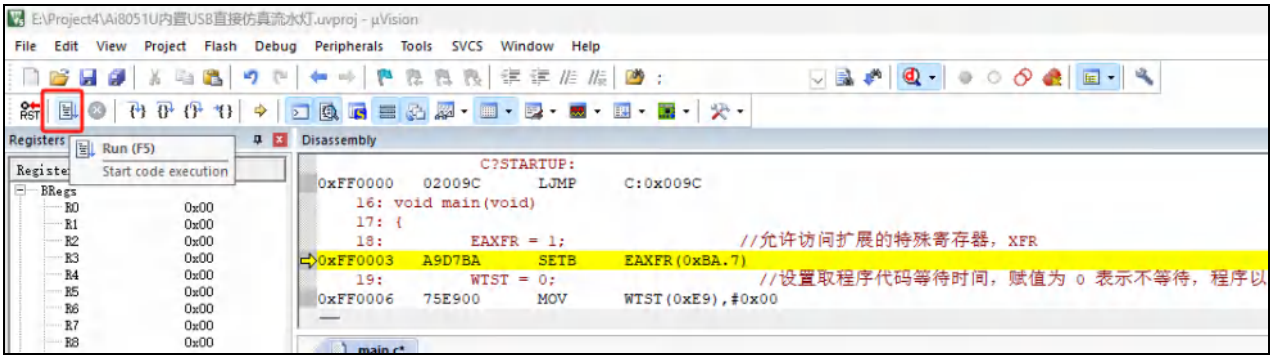


对于这个实验，我们需要初步了解 Debug Toolbar（Debug 工具栏）的几个常用按钮。

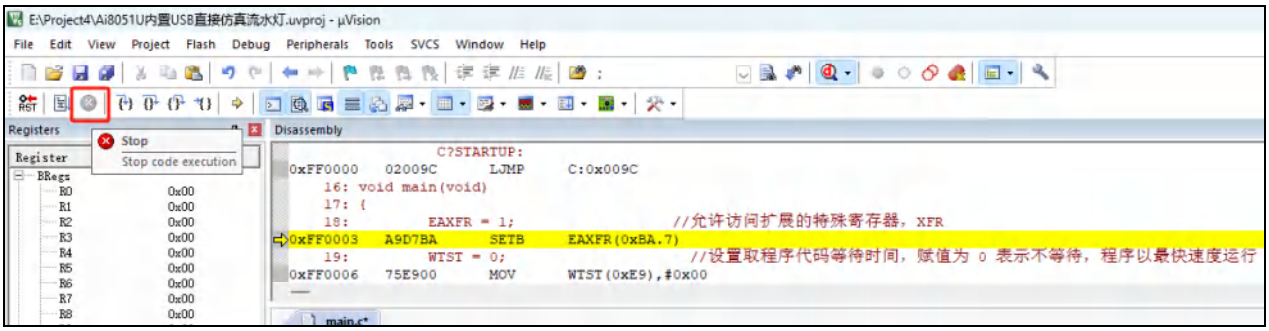
RST（Reset the CPU）按钮：类似复位按钮，相当于实现了一次硬复位。按下该按钮之后，无论程序代码执行到哪个位置，代码都会重新从头开始执行。



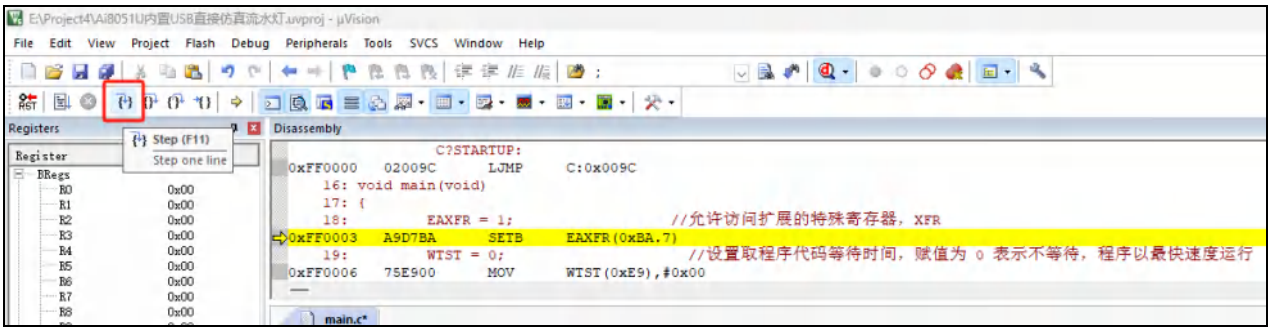
Run（Start code execution）按钮：运行按钮。点击该按钮后，程序将一直处于运行状态，或者直接快速运行到断点处（前提是你在查看的地方设置了断点）。有时候你并不需要观看程序的每步是怎么执行的，而是想快速的执行到程序的某个地方看结果，这个按钮就可以实现这样的功能。



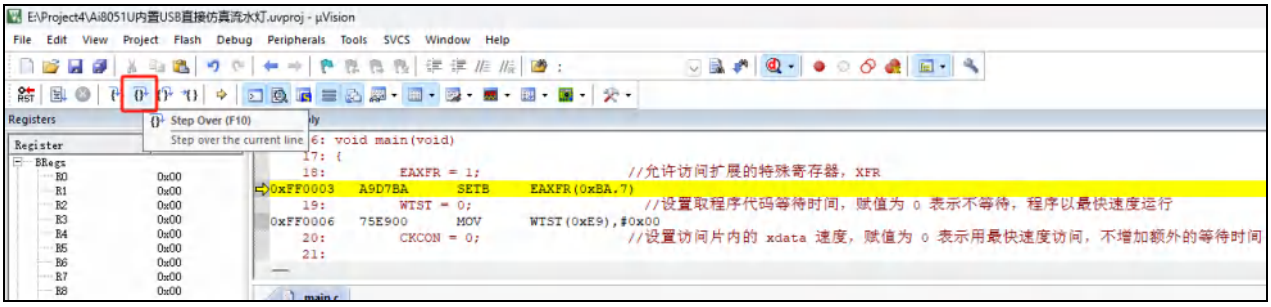
Stop（Stop code execution）按钮：停止运行按钮。此按钮在程序一直执行的时候会变为有效，通过按该按钮，就可以使程序停止下来，进入到单步调试状态。



Step（Step one line）：单步运行按钮。每点击一次该按钮，程序将执行单条语句。如果遇到函数，则会进入函数内部执行。这是调试程序时最常用的功能之一。此按钮在程序停止运行时有效。



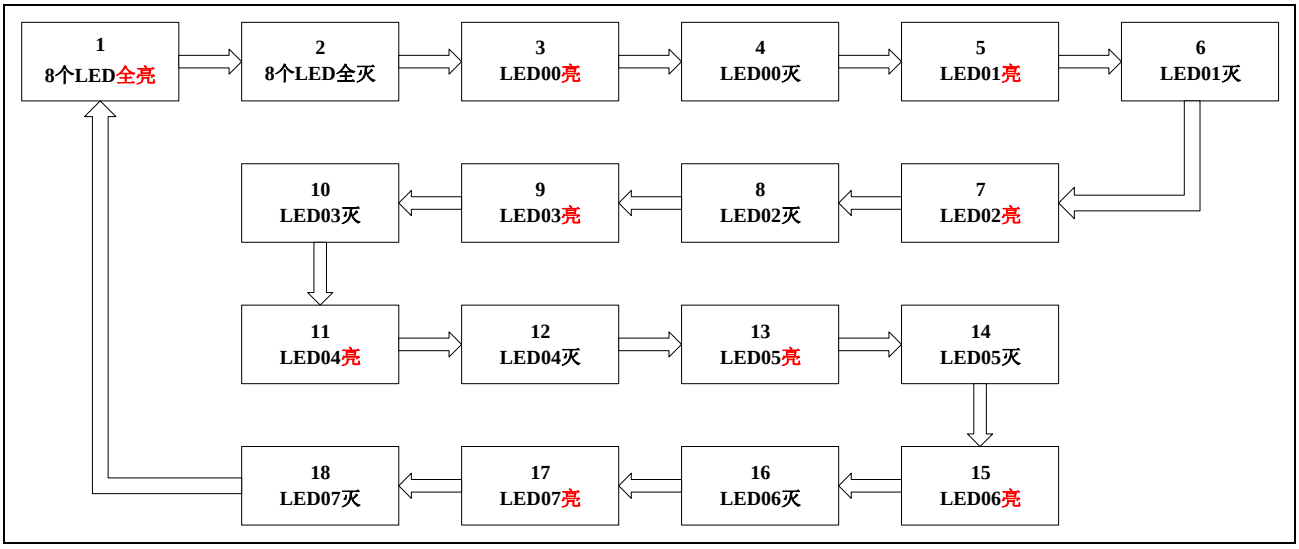
Step Over（Step over the current line）：单步跳过按钮，程序运行一行，按单条语句执行。与 Step 按钮不同的是，Step Over 按钮遇到函数不会进入函数内部，而是直接全速运行函数，并跳到下一条语句。在碰到有函数（特别是延时函数）的地方，通过该按钮就可以单步执行过这个函数，而不进入这个函数单步执行。此按钮在程序停止运行时有效。



了解上述 5 个按钮，足以观察此实验运行仿真状况。

2.18.4.2 使用 Run 按钮，观察全速运行效果的实验现象

一切准备就绪，我们先 Run 一次程序，看看程序整体执行，实验箱仿真的实验结果。

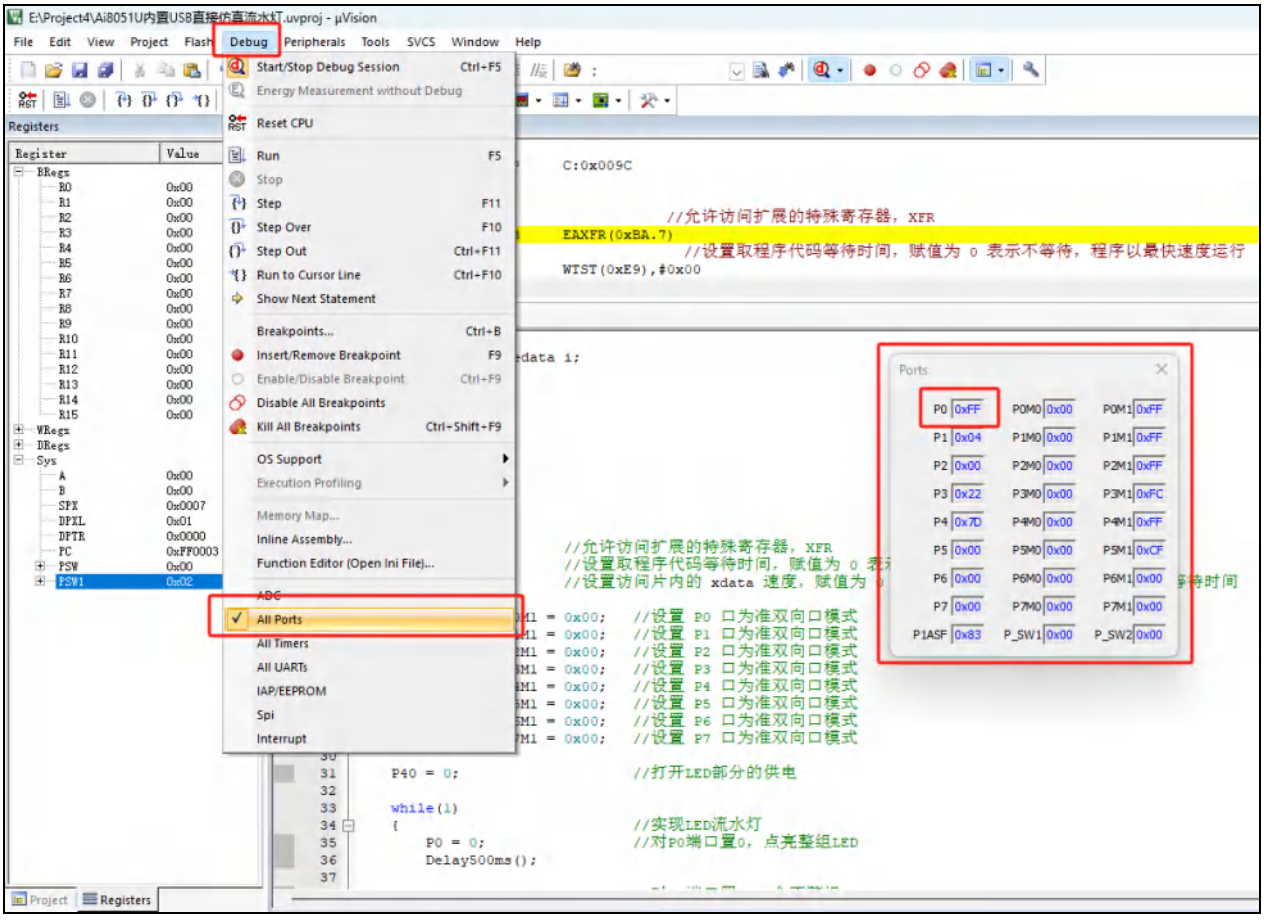


实现了我们想要的流水灯的实验现象。实验成功。

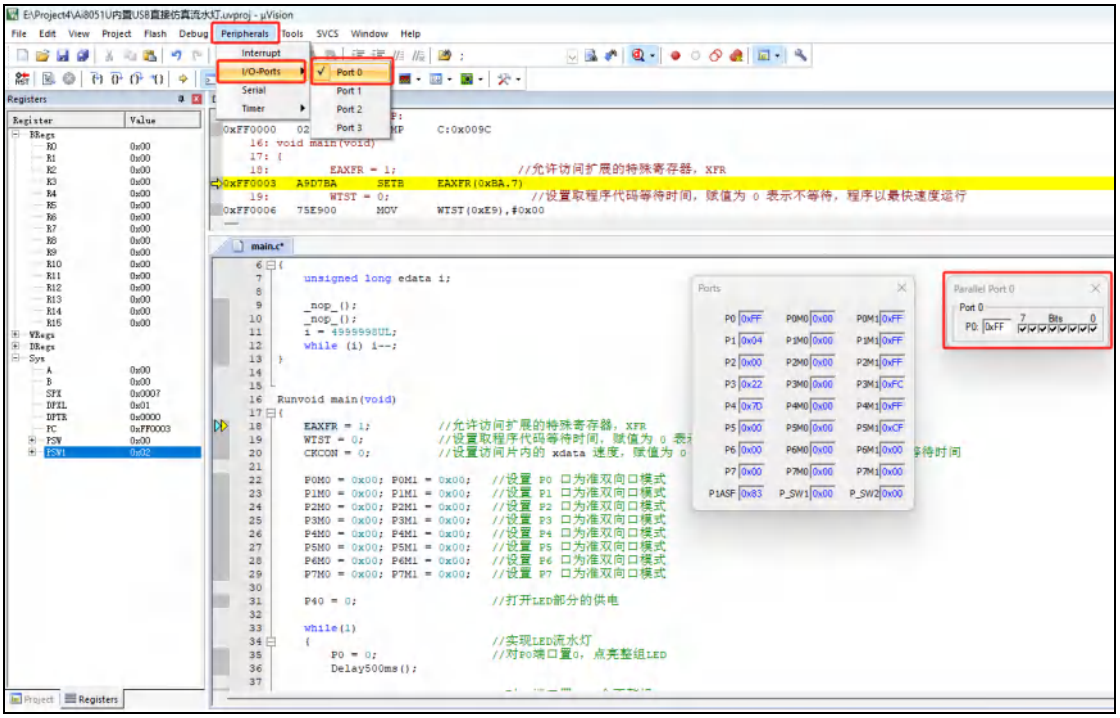
2.18.4.3 观察外设寄存器数据与实验现象

除了观察实验箱流水灯的状态，还可以用 Keil 软件来观察 P0 端口的数据，验证数据与实验箱上 LED 的亮灭现象是否对应。

1、选择菜单栏的“Debug”→勾选“All Ports”，出现“Ports”窗口



2、选择菜单栏的“Peripherals”→点击“I/O Ports”→勾选“Port 0”，出现“Parallel Port 0”窗口

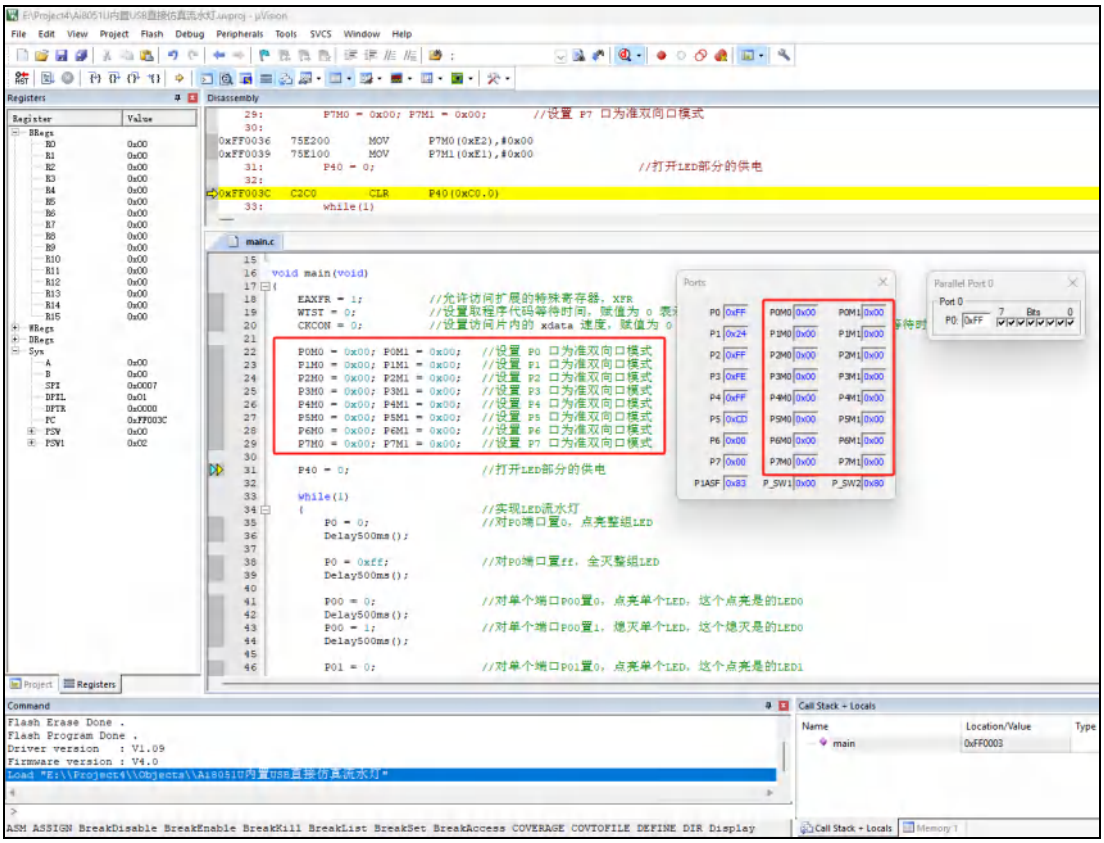


我们来单步运行程序，观察实验仿真现象。

依然将指示程序运行位置的黄色箭头停在 main 函数的第一条指令处，第 18 行：

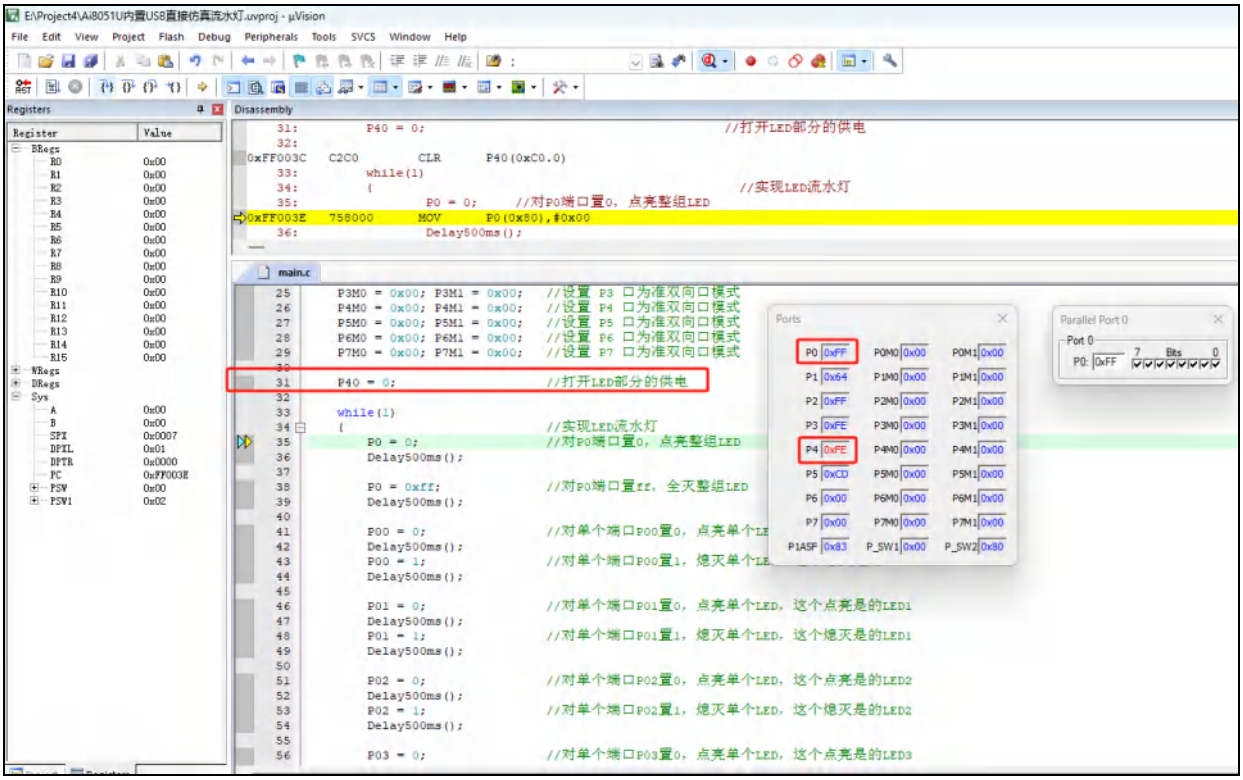
点击单步运行按钮 Step（Step one line），一直箭头指向程序到第 31 行，我们看到：

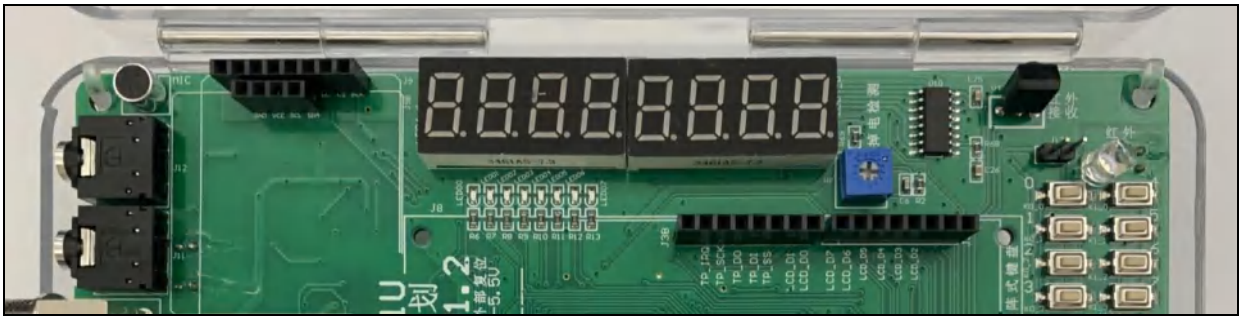
- P0 ~ P7 口依次被设置为准双向口模式，“Ports”窗口的 P0M0 ~ P0M1，…，P7M0 ~ P7M1 数据框里都显示 0x00。
- “Parallel Port 0”窗口始终不变。



点击单步运行按钮 Step（Step one line），P40 被置 0，箭头指向到程序第 35 行，这时：

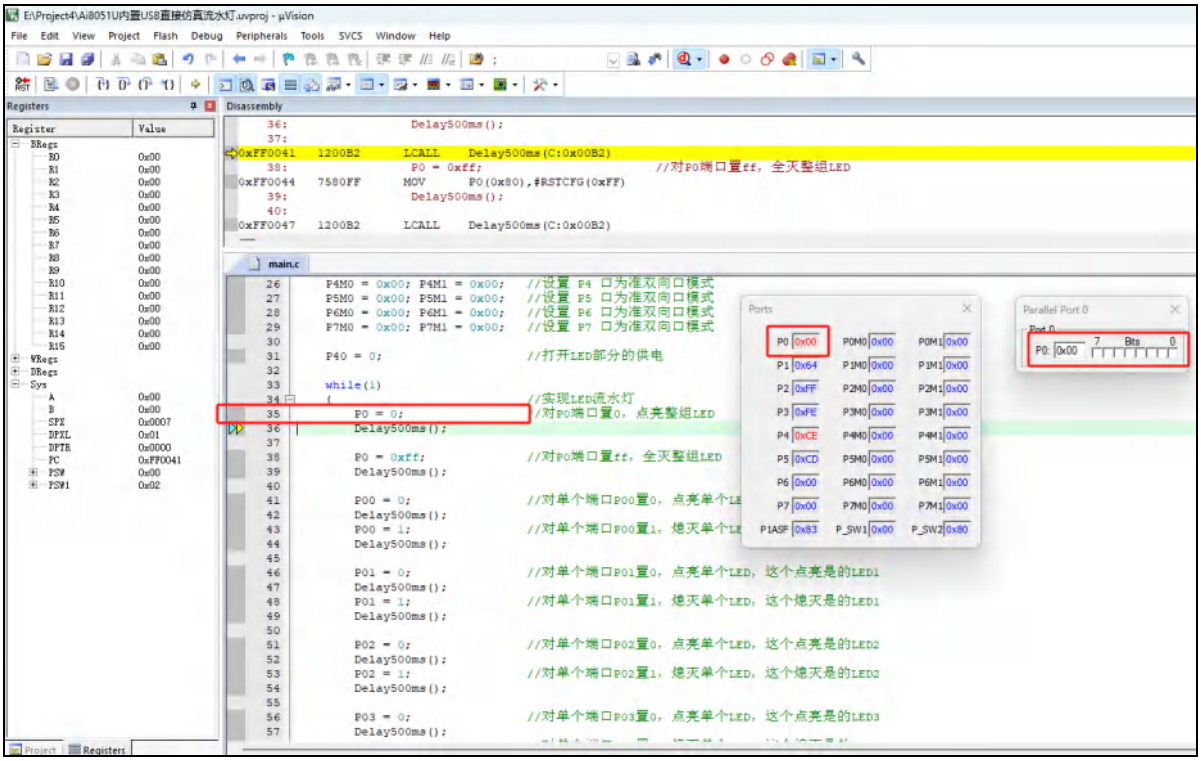
- “Ports”窗口 P4 显示 0xFE，给 LED 部分供电
- P0 依然为 0xFF，实验箱上 LED 灯保持全灭。





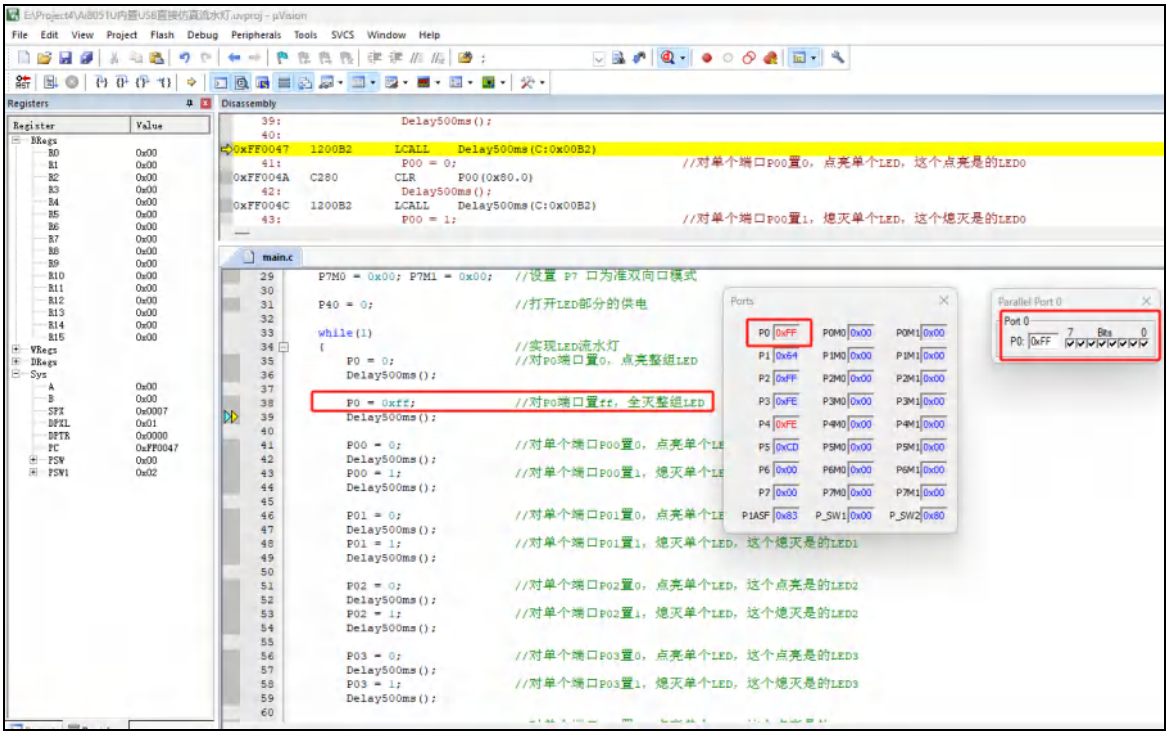
点击单步运行按钮 Step（Step one line），P0 被置 0，箭头指向到程序第 36 行：

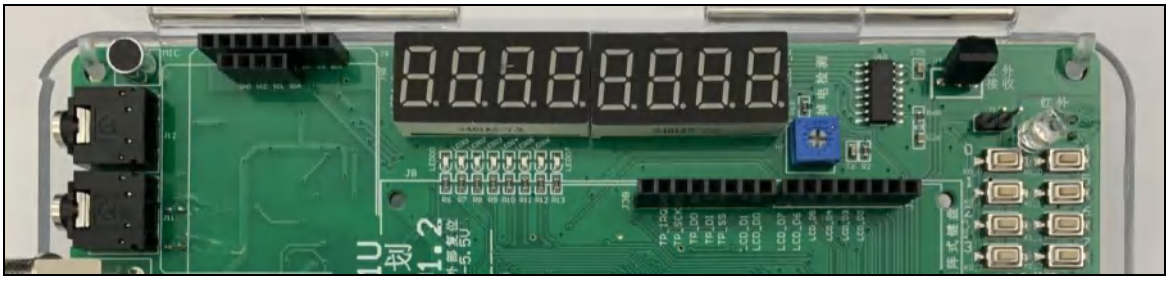
- “Ports”窗口 P0 显示 0x00
- “Parallel Port 0”窗口 P00~P07 全部置 0，显示为空
- 实验箱 LED00~LED07 全亮。



因接下来的一条指令是延时函数，**点击 Step Over 按钮**跳过延时函数，箭头指向到程序第 38 行，**再点击单步运行按钮 Step（Step one line）**，P0 被置 0xFF，箭头指向到程序第 39 行：

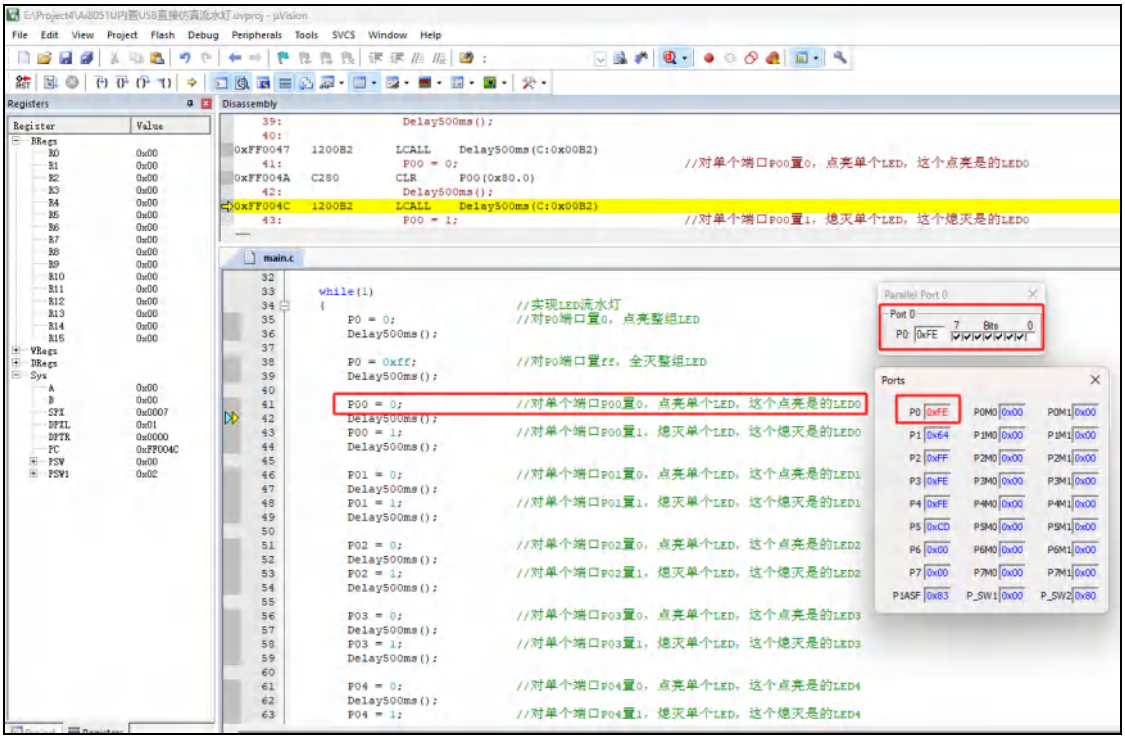
- “Ports”窗口 P0 显示 0xFF
- “Parallel Port 0”窗口 P00~P07 全部置 1
- 实验箱 LED00~LED07 全灭。





第 39 条指令是延时函数，**点击 Step Over 按钮**跳过延时函数，箭头指向到程序第 41 行，**再点击单步运行按钮 Step（Step one line）**，P00 被置 0，箭头指向到程序第 42 行：

- “Ports”窗口 P0 显示 0xFE
- “Parallel Port 0”窗口 P00 显示空，P01~P07 全部为 1
- 实验箱 LED00 灯亮，LED01~LED07 灭。



重复上述操作，观察 P0 口数据变化，及对应的实验箱上 LED 灯的变化，是否一一对应。

2.18.5　实验程序汇编语言代码

```
;以下是控制 P6.0~P6.7 引脚上一个 LED 灯循环亮灭跑马灯的汇编语言程序代码。
;特别注意延时子程序汇编语言写法。试通过每一条指令的所花的机器时钟周期计算一下软件延时时间。
;251 汇编程序
$INCLUDE(AI8051U.INC)

STACK_POIRTER     EQU     0D0H    ;堆栈开始地址

    ORG     0000H
    LJMP    START
    ORG     0100H

START:
    SETB    EAXFR           ;允许访问扩展的特殊寄存器，XFR
    MOV     WTST, #00H      ;设置取程序代码等待时间，赋值为 0 表示不等待，程序以最快速度运行
    MOV     CKCON, #00H     ;设置访问片内的 xdata 速度，赋值为 0 表示用最快速度访问，
                            ;不增加额外的等待时间
    MOV     SP, #STACK_POIRTER      ;堆栈栈顶设置
    ;端口初始化
    CLR     A
    MOV     P0M1, A     ;设置 P0 为准双向口
    MOV     P0M0, A
    MOV     P1M1, A     ;设置 P1 为准双向口
    MOV     P1M0, A
    MOV     P2M1, A     ;设置 P2 为准双向口
    MOV     P2M0, A
    MOV     P3M1, A
    MOV     P3M0, A
    MOV     P4M1, A
    MOV     P4M0, A
    MOV     P5M1, A
    MOV     P5M0, A
    MOV     P6M1, A
    MOV     P6M0, A
    MOV     P7M1, A
    MOV     P7M0, A
    MOV     P0, #0FFH
    ;变量初始化
    CLR     P40             ;流水灯总开关打开

LOOP:
    MOV     P0, #00H        ;P00 ~ P07 全亮
    ACALL   DELAY500MS      ;调用延时子程序
    MOV     P0, #0FFH       ;P00 ~ P07 全灭
    ACALL   DELAY500MS      ;调用延时子程序
```

```
    CLR     P00             ;给 P6^0 清零，低电平时 led 亮
    ACALL   DELAY500MS      ;调用延时子程序
    SETB    P00             ;给 P6^0 置 1，高电平时 led 灭
    CLR     P01
    ACALL   DELAY500MS      ;调用延时子程序
    SETB    P01
    CLR     P02
    ACALL   DELAY500MS      ;调用延时子程序
    SETB    P02
    CLR     P03
    ACALL   DELAY500MS      ;500MS;调用延时子程序
    SETB    P03
    CLR     P04
    ACALL   DELAY500MS      ;调用延时子程序
    SETB    P04
    CLR     P05
    ACALL   DELAY500MS      ;调用延时子程序
    SETB    P05
    CLR     P06
    ACALL   DELAY500MS      ;调用延时子程序
    SETB    P06
    CLR     P07
    ACALL   DELAY500MS
    SETB    P07

    SJMP    LOOP            ;无条件跳转至 LOOP 处

DELAY500MS:                 ;延时 0.5S 子程序@40.000MHz
    PUSH    DR0             ;保护现场，将双字寄存器 DR0（即 R0~R3）推入堆栈保存
    MOV     WR0, #76        ;将 WR0 赋值 76（思考 R0=？R1=？）
    MOV     WR2, #19262     ;将 WR2 赋值 19262（思考 R2=？R3=？）
NEXT:
    DEC     DR0, #1         ;DR0<-（DR0）-1
    JNE     NEXT            ;(DR0)≠0，则跳至 NEXT 标号处即继续-1，知道等于 0 则执行下一条语句
    POP     DR0             ;恢复现场
    RET                     ;子程序返回

END                         ;汇编程序结束伪指令语句
```

保存，编译，如同 C 语言代码程序一样，打开 AIapp-ISP 软件，连接好 Ai8051U 实验箱，并下载到实验箱，就能观察到一样的流水灯现象。

2.18.6 使用【AiCube】项目创建助手工具创建使用 Ai8051U 内置 USB 仿真流水灯

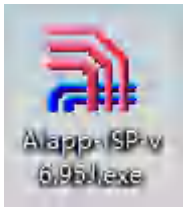
STCAI 在最新版的 AIapp-ISP 软件中，推出【AiCube】项目创建助手工具，目前支持 Ai8051U-32Bit 各功能模块的智能化配置，并生成程序框架。这项功能在程序员建立项目时，简化了项目的初始化难度，提高效率，避免因不太熟悉或不完全了解该单片机的性能而忽略个别配置，引起编译错误等问题。尤其对刚涉足单片机的初学者，此工具不仅引导其了解单片机编程流程，还极大的降低了入门门槛，缩短了摸索的时间，提高入门的兴趣和信心。现在我们使用这个工具来生成实现此实验功能的配置和代码。

a) 安装 AIapp-ISP 下载/编程/烧录/软件

安装 STCAI 公司研发的 AIapp-ISP 下载 / 编程 / 烧录 / 软件，含强大的辅助开发工具包。登录 STCAI 官网，下载最新版的 AIapp-ISP 安装包，下载链接如下：https://www.stcai.com/gjrj

下载下来的是压缩包文件如下图：

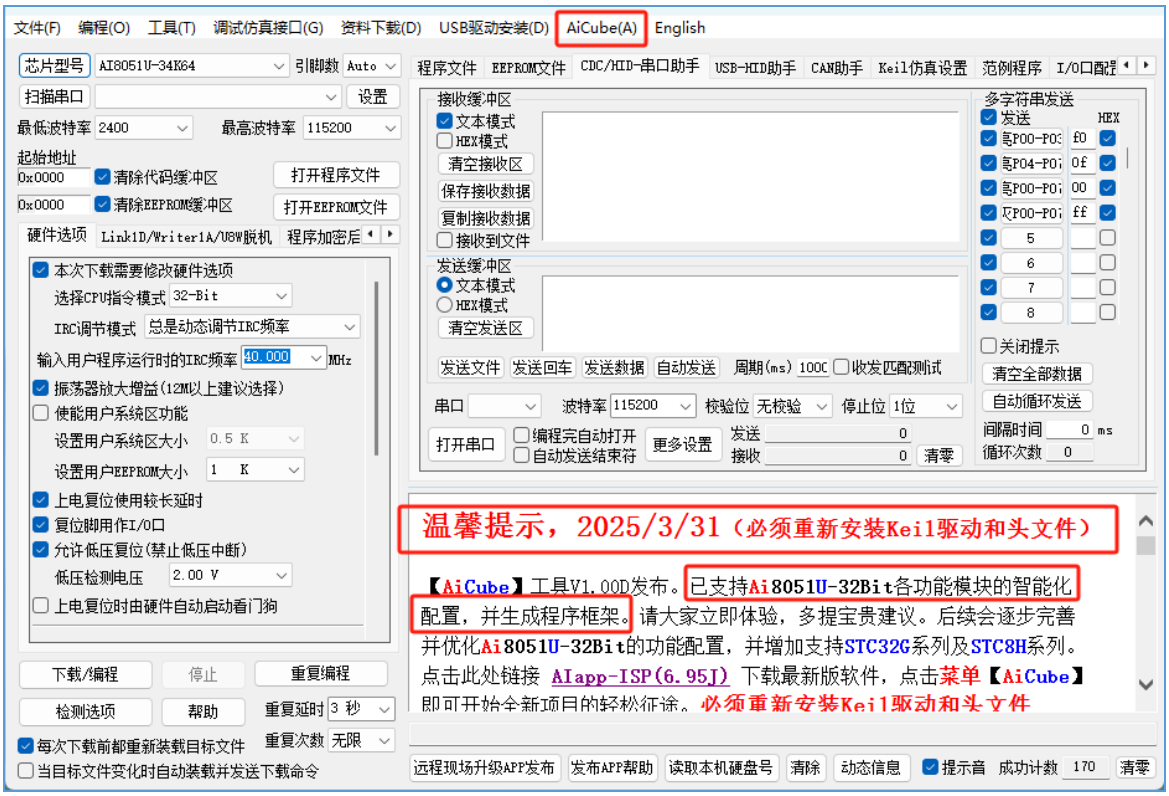
双击后解压缩如下图：



如果是解压缩到非桌面的其他目录，如 D:\STCAI-ISP，建议创建快捷方式到桌面方便打开。这实际是一个免安装的绿色软件。

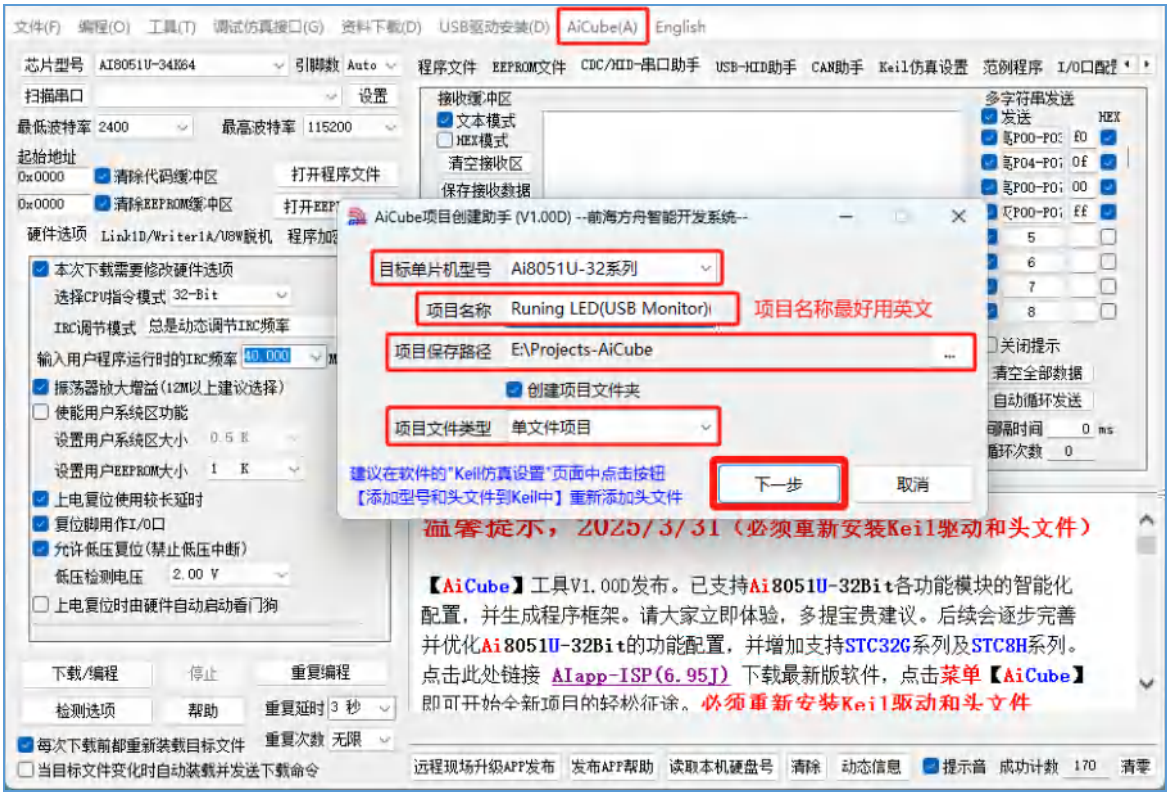
b) 打开【AiCube】助手

双击图标，即可打开 AIapp-ISP 软件界面，即可直接使用，如下图：

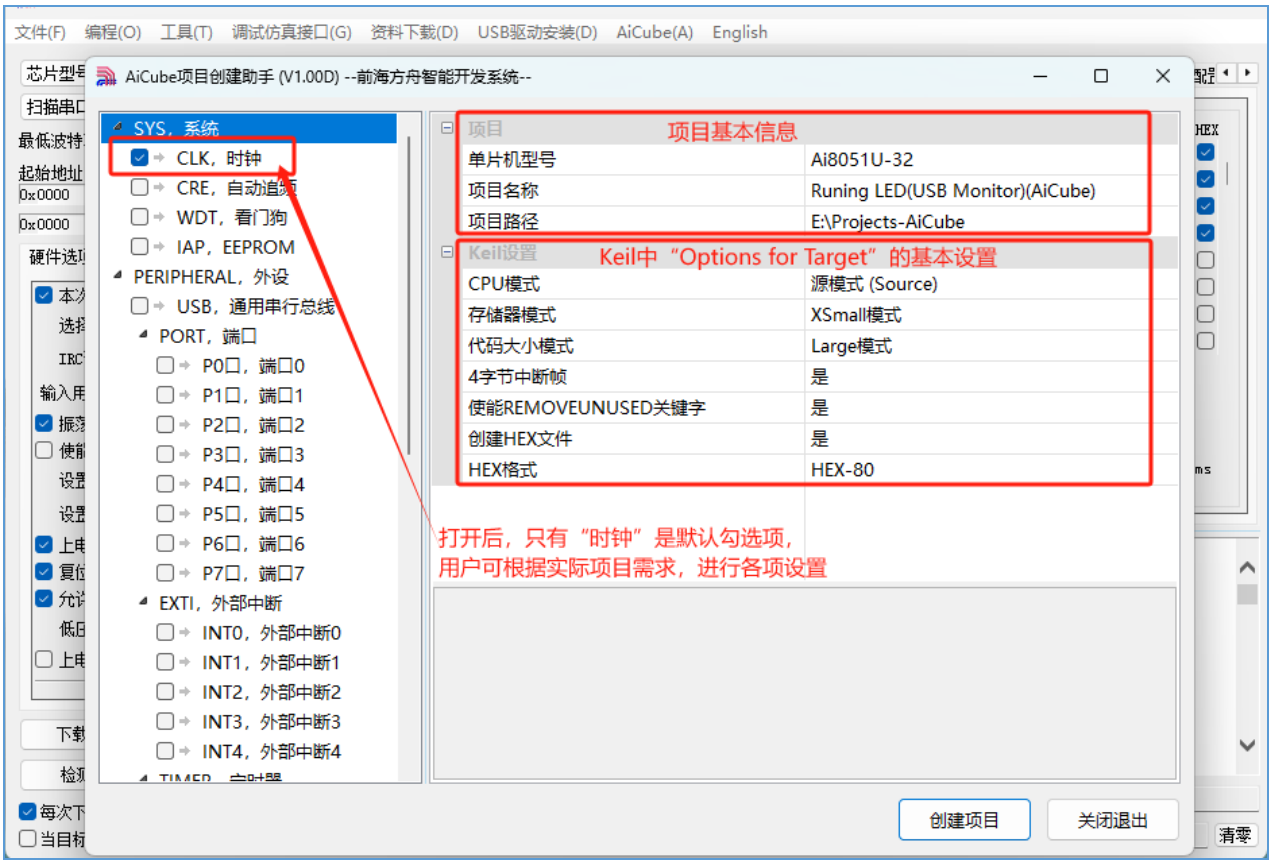


c) 使用【AiCube】助手创建项目

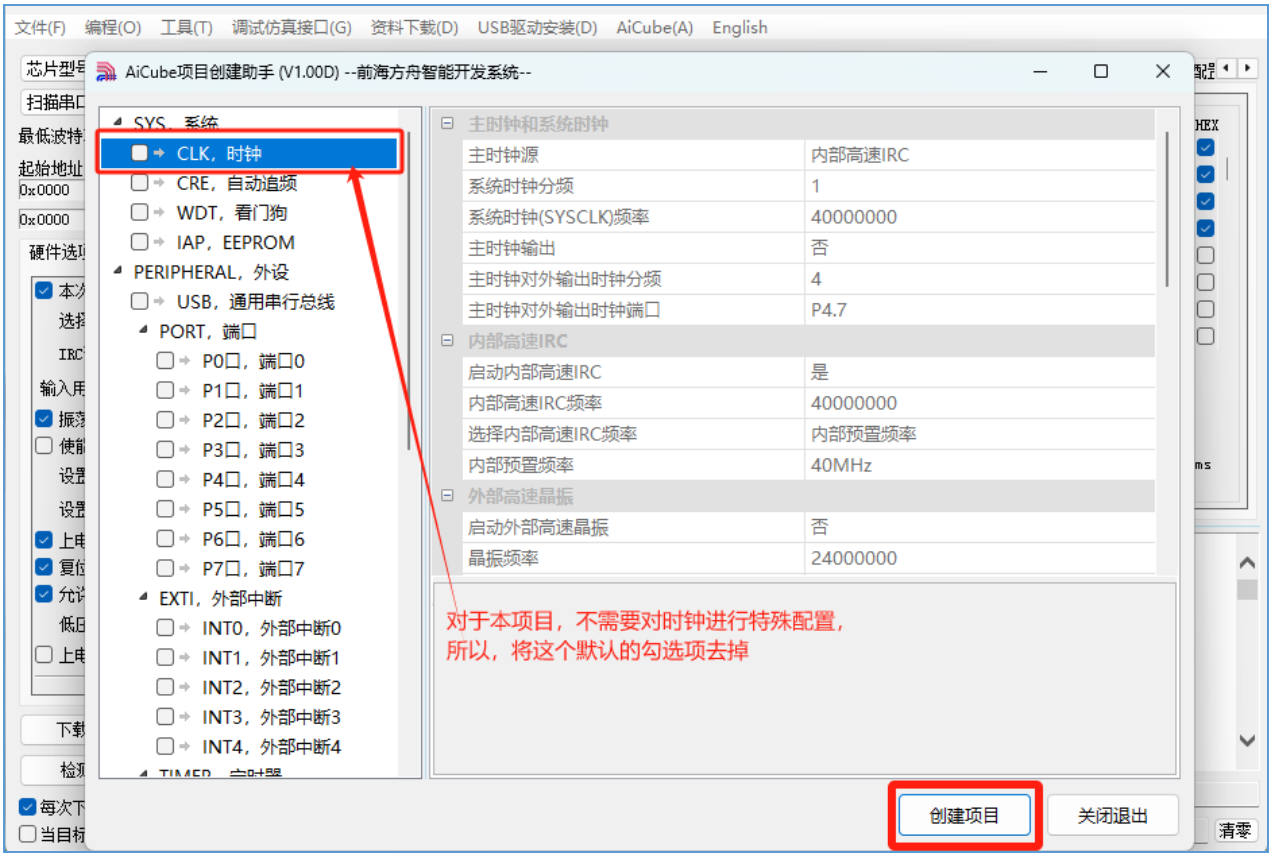
点击【AiCube（A）】，弹出如下窗口：



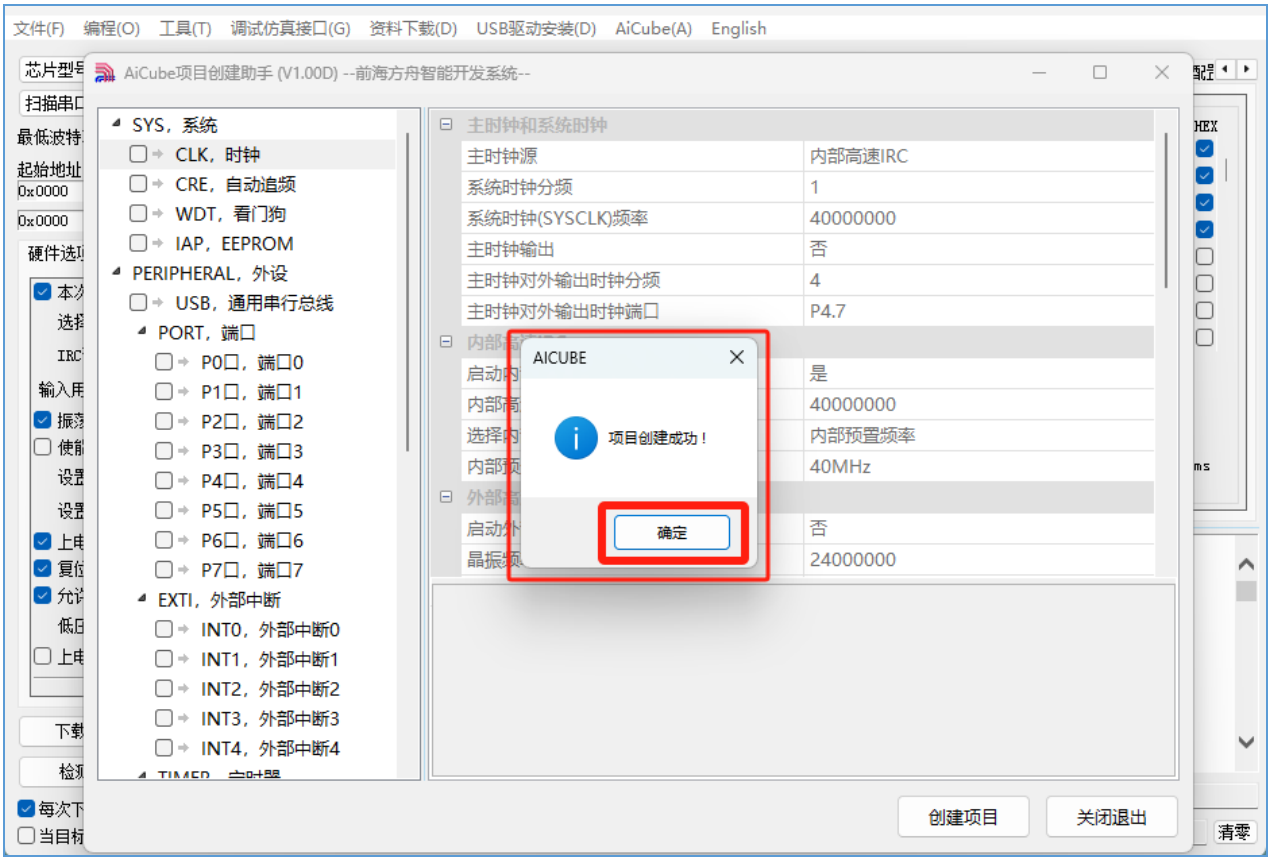
选择并填写项目基本信息，建议项目名称用英文！点击“下一步”，出现如下配置界面：



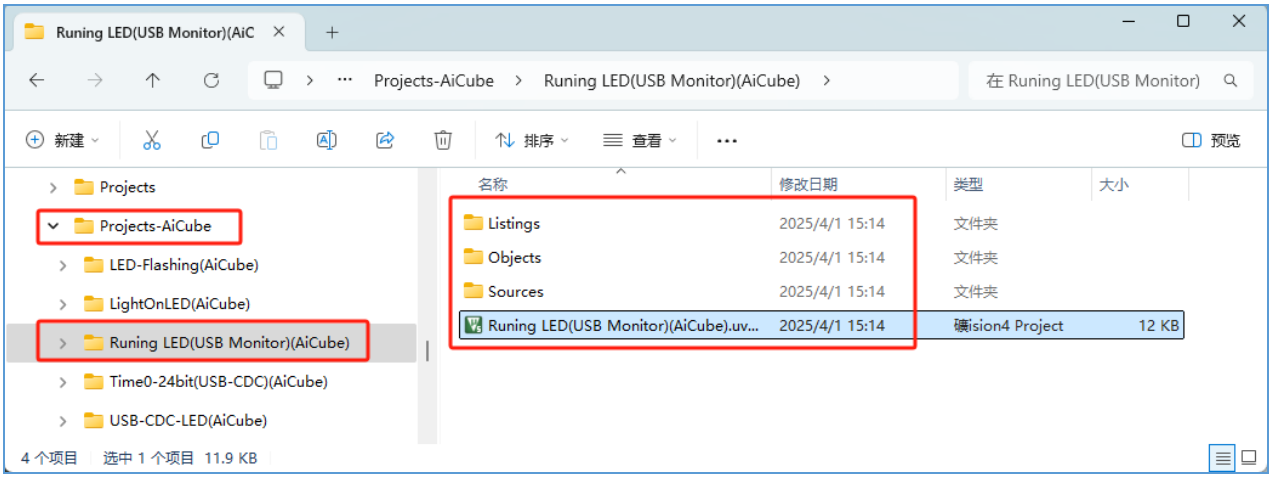
对于本项目，用 Ai8051U 内置的 USB 功能仿真实现流水灯的实验，非常简单，不需要对时钟或者其他选项做特殊配置，使用【AiCube】助手根据本项目的实际需求，对配置进行如下修改：



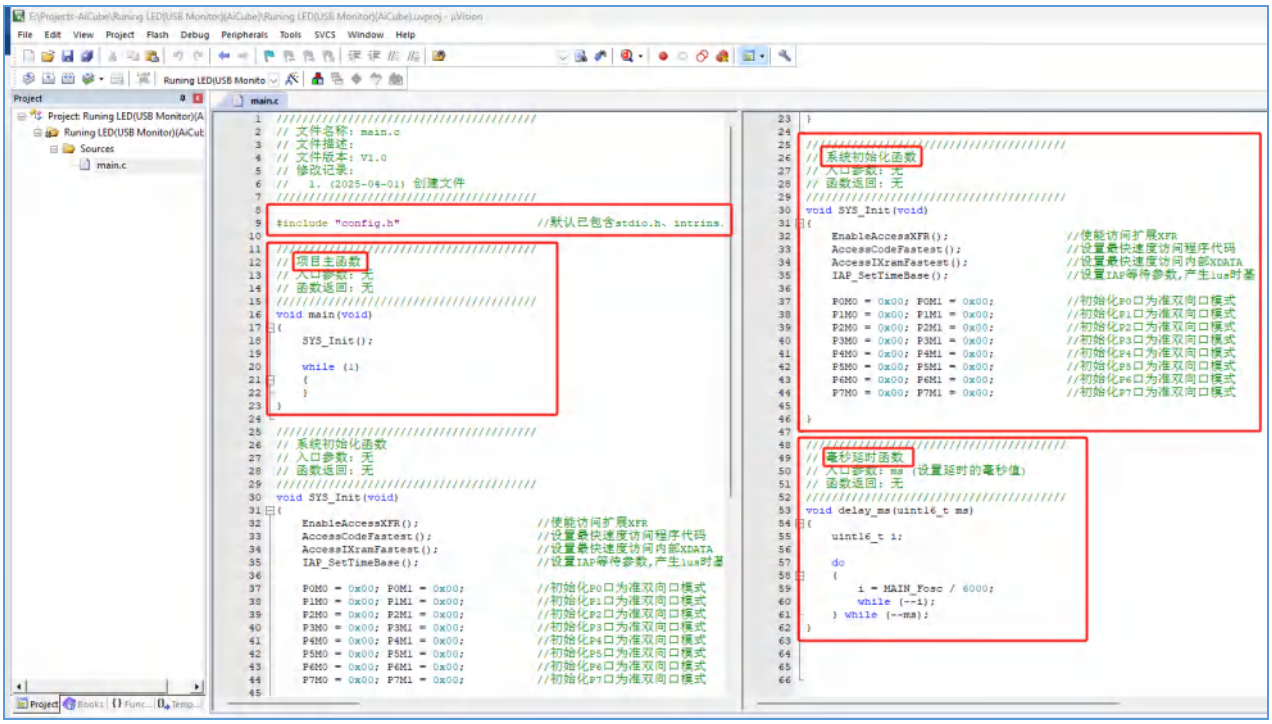
点击“创建项目”，【AiCube】助手提示：“项目创建成功！”，如下图：



点击“确定”，即成功创建了项目及项目文件目录“Runing LED(USB Monitor)(AiCube)”，找到并打开刚刚创建的项目目录。发现刚刚的操作过程，【AiCube】助手已经帮我们建立了项目目录及 Runing LED(USB Monitor)(AiCube).uvproj 文件，如下图：



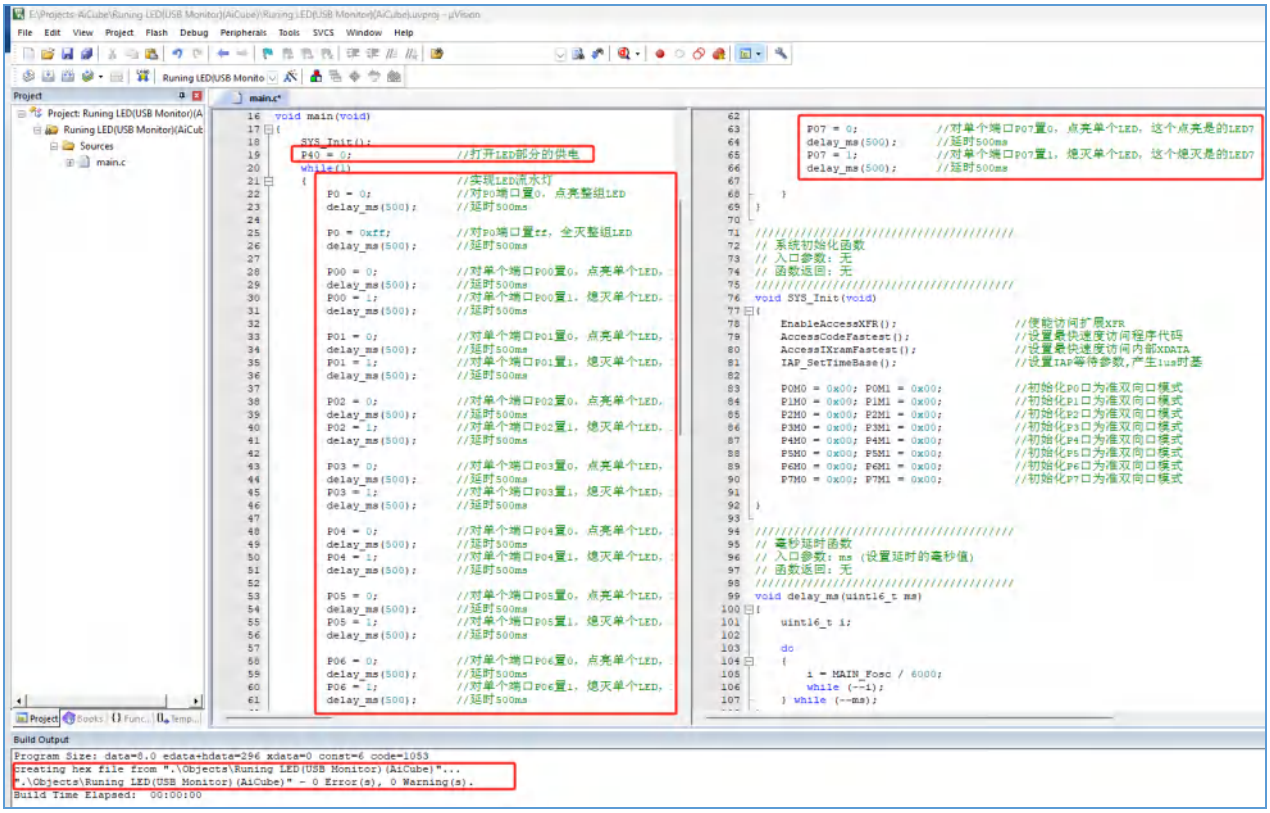
双击打开项目文件：Runing LED(USB Monitor)(AiCube).uvproj，如下图（为方便查看，使用菜单“Window”→“Split”将原窗口拆分成两个窗口，全代码观察）：



根据在【AiCube】助手中的配置，从上图中我们看到【AiCube】助手创建的本实验项目中有了 main.c。main.c 中按模块化设计，已经包含了：必要的头文件、主函数、系统初始化函数、延时函数。

d) 增加实验项目代码

为实现“用 Ai8051U 内置的 USB 功能仿真实现流水灯”的实验，我们现在只需要在主函数中加入“P40 = 0”打开 LED 供电的代码，在 while 循环中补充实现流水灯的代码，其中延时函数根据需要直接调用。保存，编译，如下图（为方便查看，使用菜单“Window”→“Split”将原窗口拆分成两个窗口）：

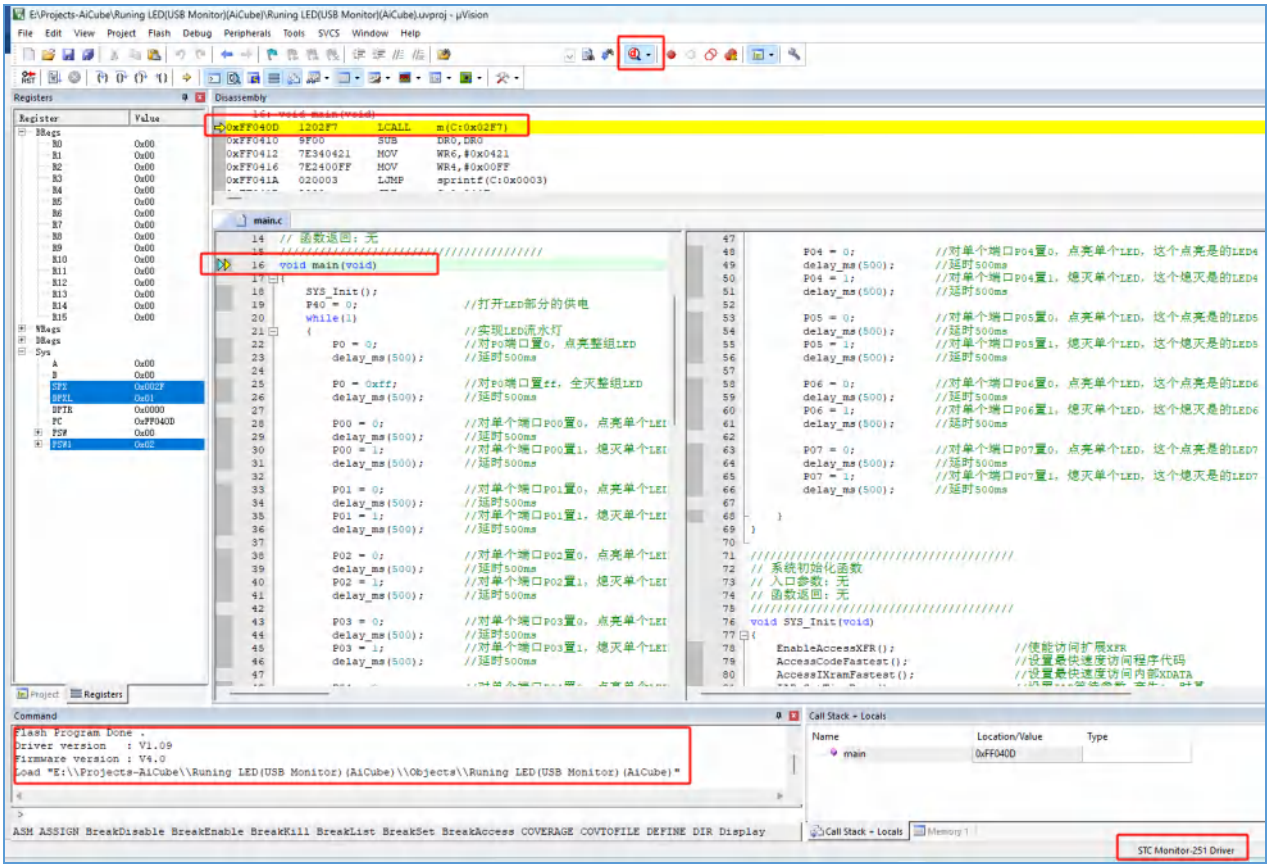


e) 使用 AIapp-ISP 软件设置仿真芯片并连接 Ai8051U 实验箱验证实验效果

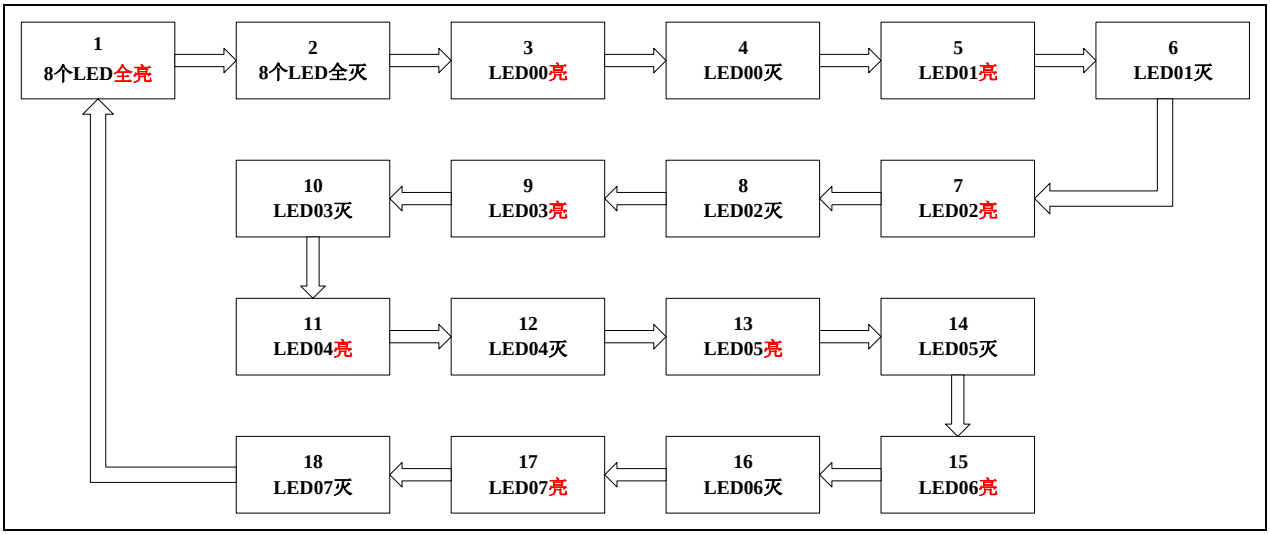
连接好 Ai8051U 实验箱，运行 AIapp-ISP 软件系统，在 Keil 仿真设置选项卡中，按步骤操作“将所选目标单片机设置为仿真芯片”。

回到 Keil 界面，在 Keil 中按步骤在“Options for Target”的 Debug 选项卡和 Utilities 选项卡中完成使用“STC Monitor-251 Driver”的 USB 仿真操作。

点击 File Toolbar 上的“Start/Stop Debug Session”按钮，进入仿真界面，如下图：



一切准备就绪，点击 Debug Toolbar 上的“Run”按钮，我们看到 Ai8051U 实验箱的 LED 灯按实验的预设跑动起来了。



有了【AiCube】助手，单片机编程就是这么简单！快用起来吧！

2.19 扩展 Keil 对中断号数量的支持，中断号大于 31 编译出错的处理

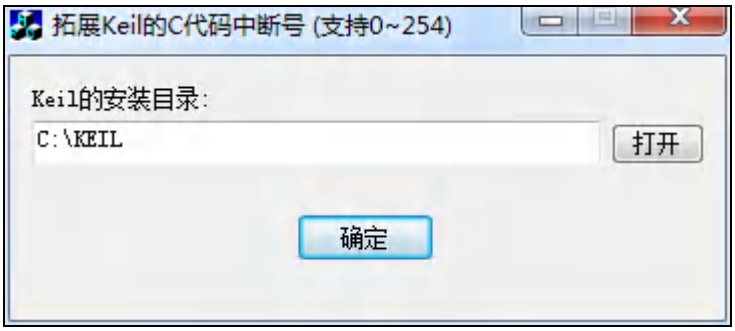
注：Keil C51/C251 编译器只支持 31 以内的中断号（0~31），超过 31 编译会报错。有热心网友提供了一个简单的 Keil 中断号拓展工具，可将中断号拓展到 254。工具界面如下：

但对于目前现有的 Keil 版本，只能使用本章节的方法进行临时解决。

2.19.1 如何下载 Keil 中断号拓展工具，及安装

软件下载地址：https://www.STCAI.com/gjrj

下载完成后，直接解压缩，运行该绿色免安装工具。



点击“打开”按钮，定位到 Keil 的安装目录后，点击“确定”即可。
根据实际的 Keil 安装目录选择地址，如部分用户安装的目录是，C:\Keil_v5
由于 Keil 的版本在不断更新，而早期版本过多，有无法收集齐，这里列举一下已测试通过的 C51.EXE 版本和 C251.EXE 版本

已测试通过的 C51.EXE 版本：

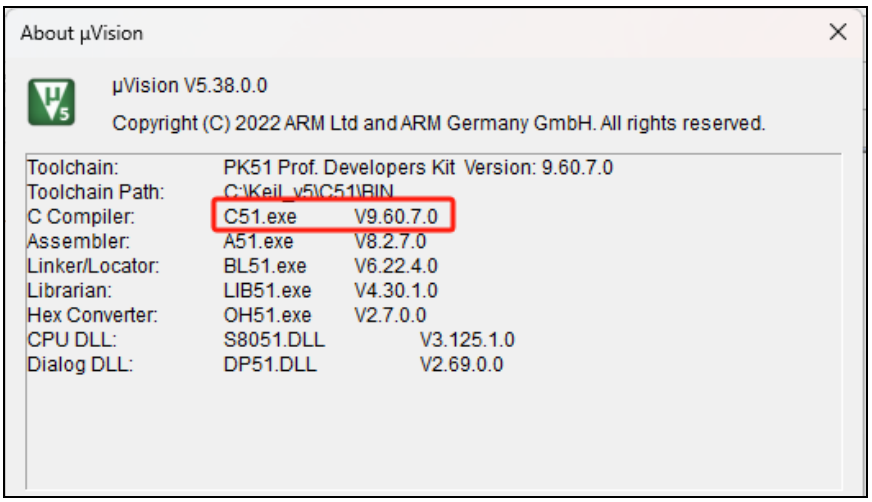
V6.12.0.1
V8.8.0.1
V9.0.0.1
V9.1.0.1
V9.53.0.0
V9.54.0.0
V9.57.0.0
V9.59.0.0
V9.60.0.0

已测试通过的 C251.EXE 版本：

V5.57.0.0
V5.60.0.0

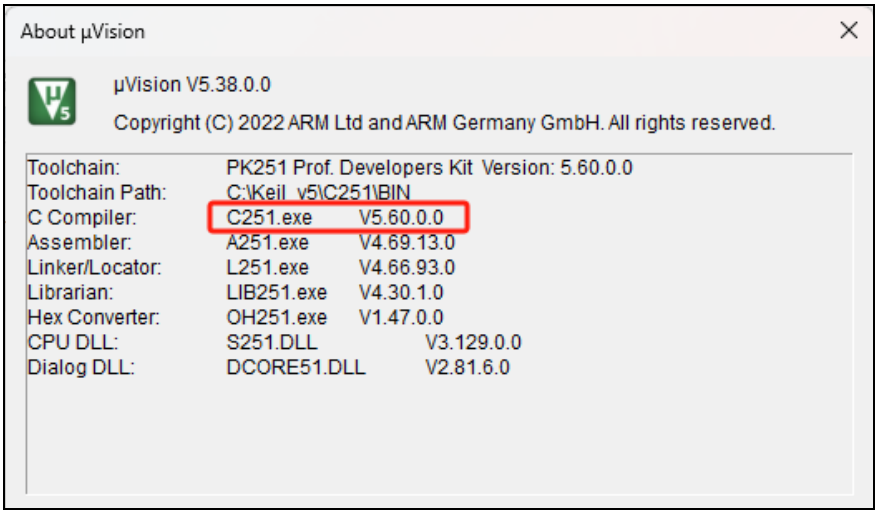
查看 C51.EXE 版本的方法：

在 keil 中打开一个基于 AI8 系列或者 AI15 系列单片机的项目，在 Keil 软件菜单项“Help”中打开“About uVision...”



查看 C251.EXE 版本的方法：

在 keil 中打开一个基于 Ai8051U 系列单片机的项目，在 Keil 软件菜单项“Help”中打开“About uVision...”



2.19.2 BL51/LX51 链接器和扩展中断号工具，REMOVEUNUSED 说明

首先，中断号扩展工具只是修改 C51 和 C251 编译器，不会修改 BL51/LX51 链接器。

Keil 的 8051 编译环境下的问题：

- 如果不扩展中断号，C51 编译器无法编译超过 31 的中断号，扩展后可以支持到 254 个中断。
- Keil C51 扩展中断号后，可以和 BL51 链接器配合使用，
  但 BL51 不支持 REMOVEUNUSED 关键字。
- 要使用 REMOVEUNUSED 关键字，则必须使用 C51+LX51 组合。
  但 LX51 不支持扩展中断号功能

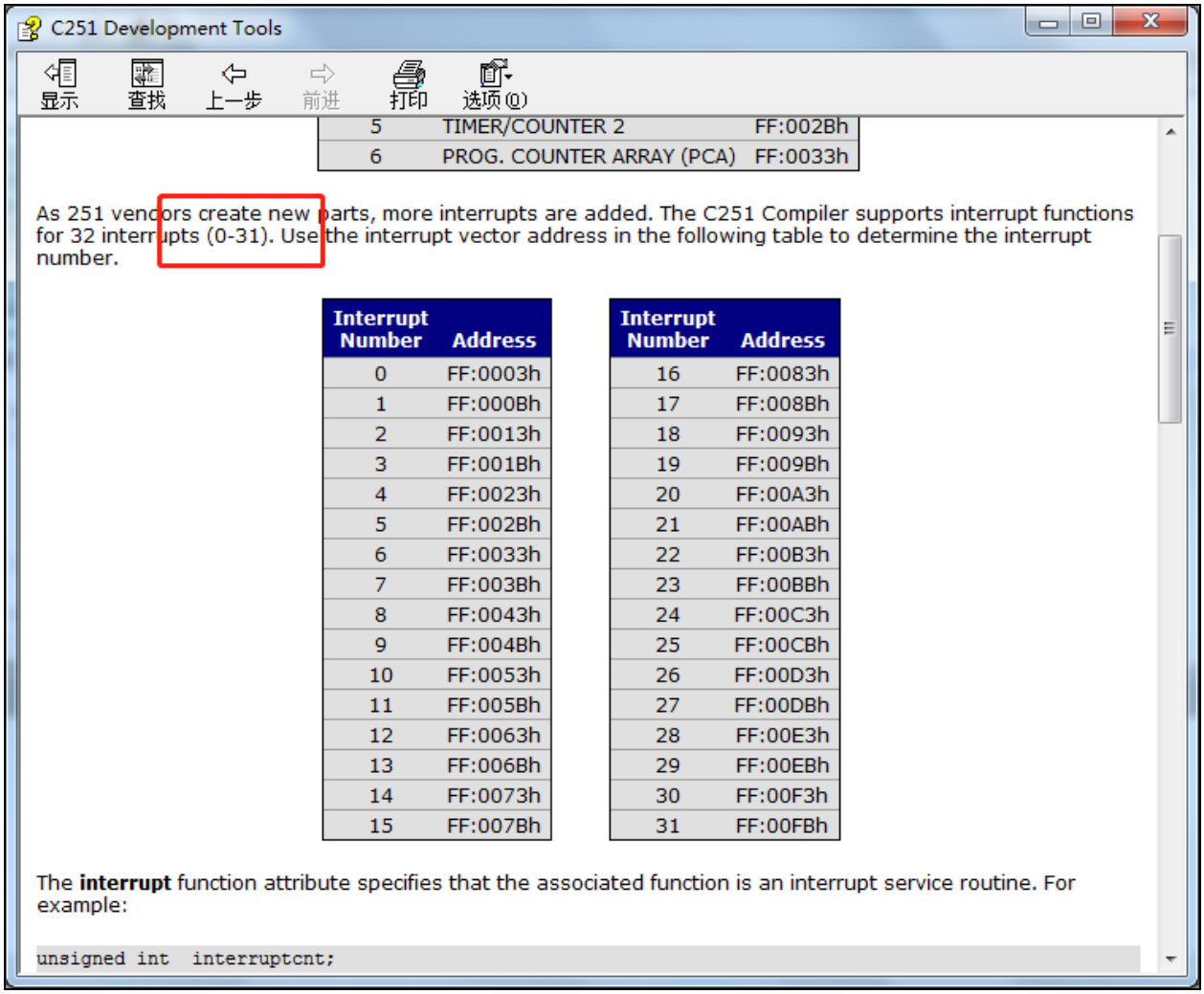
所以：

===要使用扩展的中断号，就只能使用 C51+BL51 的工具组合，
不支持 REMOVEUNUSED 关键字

===要使用 REMOVEUNUSED 关键字，
只能使用 C51+LX51 的工具组合，但不能支持扩展的中断号

2.19.3 如不使用扩展中断号工具，则需借用保留中断号进行中转

在 Keil 的 C251 编译环境下，中断号只支持 0~31，即中断向量必须小于 0100H。

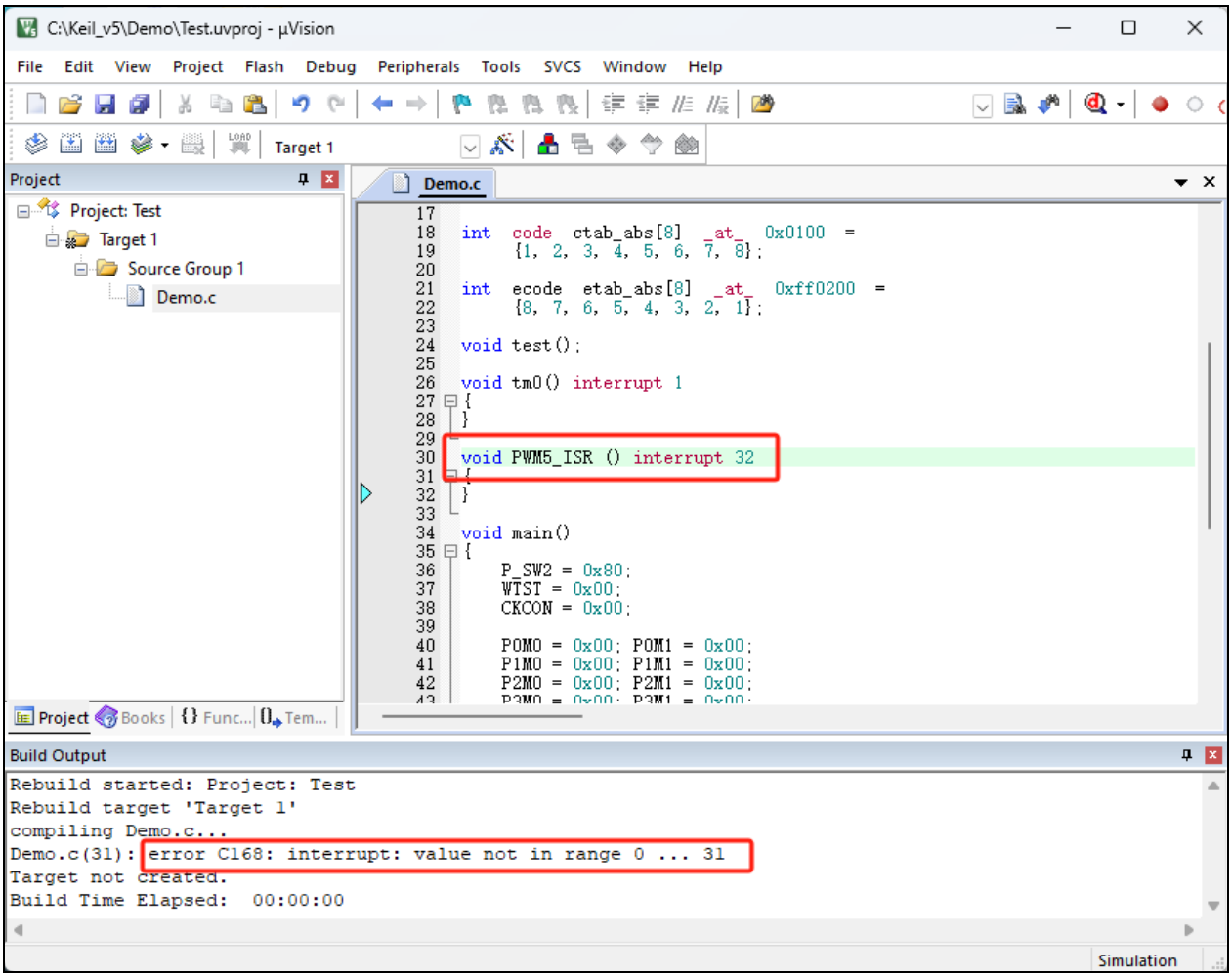


下表是目前所有系列的中断列表：

中断号	中断向量	中断类型
0	0003 H	INT0
1	000B H	定时器 0
2	0013 H	INT1
3	001B H	定时器 1
4	0023 H	串口 1
5	002B H	ADC
6	0033 H	LVD
8	0043 H	串口 2
9	004B H	SPI
10	0053 H	INT2
11	005B H	INT3
12	0063 H	定时器 2
13	**006B H**	
14	0073 H	系统内部中断
15	007B H	系统内部中断

中断号	中断向量	中断类型
16	0083 H	INT4
17	008B H	串口 3
18	0093 H	串口 4
19	009B H	定时器 3
20	00A3 H	定时器 4
21	00AB H	比较器
24	00C3 H	I2C
25	00CB H	USB
26	00D3 H	PWMA
27	00DB H	PWMB
28	00E3 H	CAN1
29	00EB H	CAN2
30	00F3 H	LIN
36	**0123 H**	**RTC**
37	**012B H**	**P0 口中断**
38	**0133 H**	**P1 口中断**
39	**013B H**	**P2 口中断**
40	**0143 H**	**P3 口中断**
41	**014B H**	**P4 口中断**
42	**0153 H**	**P5 口中断**
43	**015B H**	**P6 口中断**
44	**0163 H**	**P7 口中断**
45	**016B H**	**P8 口中断**
46	**0173 H**	**P9 口中断**
47	**017BH**	**M2M DMA 中断**
48	**0183H**	**ADC DMA 中断**
49	**018BH**	**SPI DMA 中断**
50	**0193H**	**UR1T DMA 中断**
51	**019BH**	**UR1R DMA 中断**
52	**01A3H**	**UR2T DMA 中断**
53	**01ABH**	**UR2R DMA 中断**
54	**01B3H**	**UR3T DMA 中断**
55	**01BBH**	**UR3R DMA 中断**
56	**01C3H**	**UR4T DMA 中断**
57	**01CBH**	**UR4R DMA 中断**
58	**01D3H**	**TFT 彩屏 DMA 中断**
59	**01DBH**	**TFT 彩屏中断**
60	**01E3H**	**I2CT DMA 中断**
61	**01EBH**	**I2CR DMA 中断**
62	**01F3H**	**I2S 中断**
63	**01FBH**	**I2ST DMA 中断**
64	**0203H**	**I2SR DMA 中断**

不难发现，RTC 中断开始，后面所有的中断服务程序，在 keil 中均会编译出错，如下图所示：



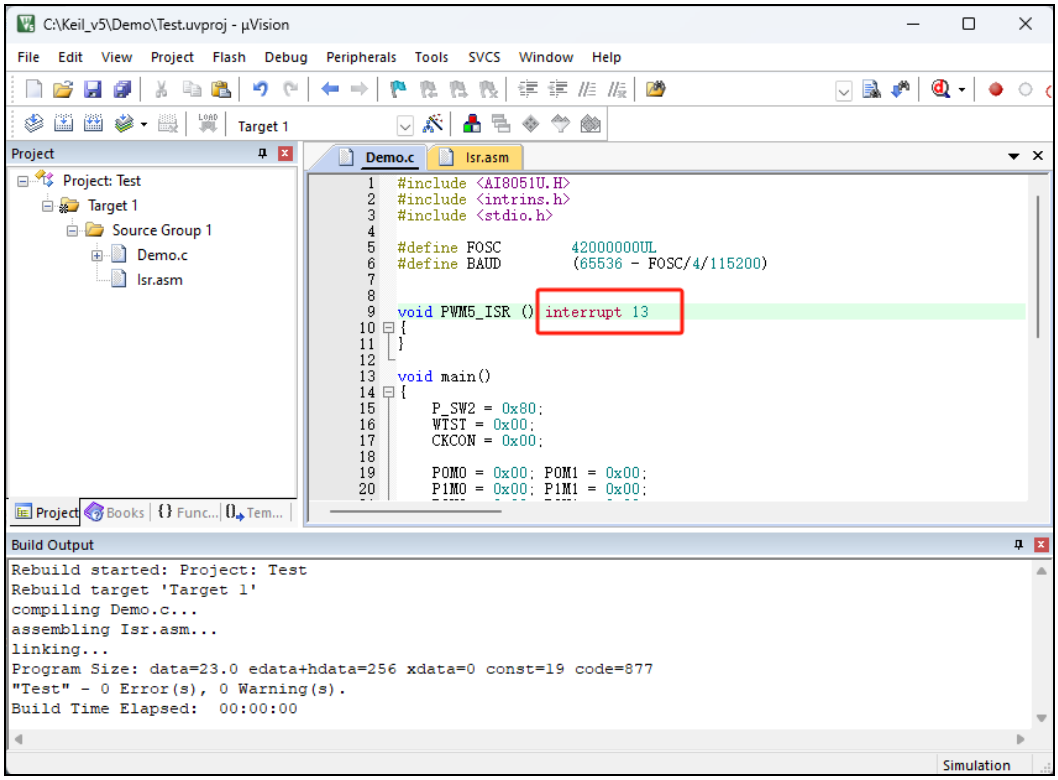
处理这种错误有如下三种方法：（均需要借助于汇编代码，优先推荐使用方法 1）

方法 1：借用 13 号中断向量

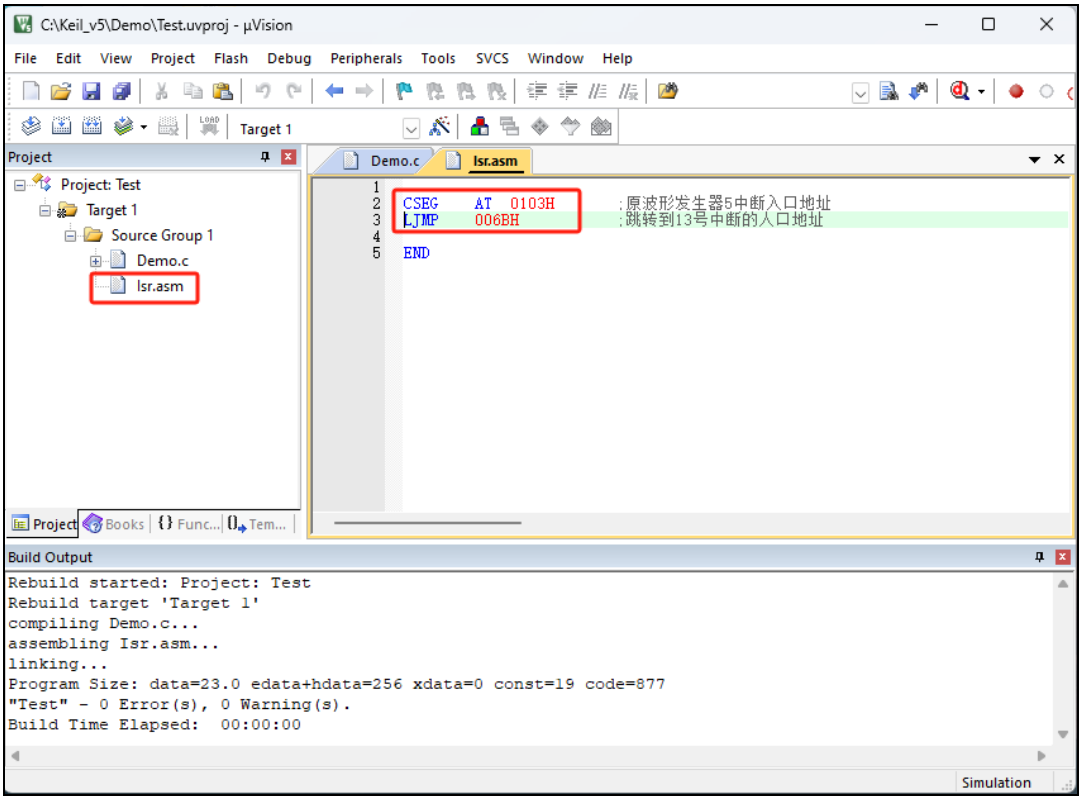
0~31 号中断中，第 13 号是保留中断号，我们可以借用此中断号

操作步骤如下：

1、将我们报错的中断号改为“13”，如下图：

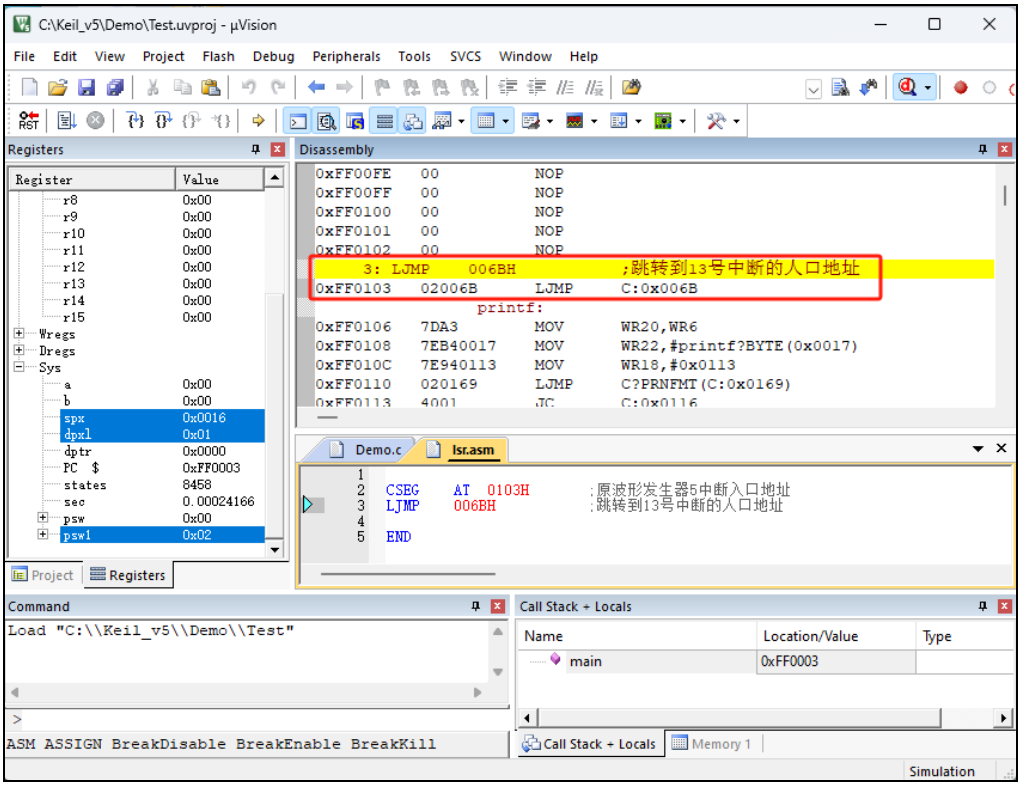


2、新建一个汇编语言文件，比如“isr.asm”，加入到项目，并在地址“0103H”的地方添加一条“LJMP 006BH”，如下图：

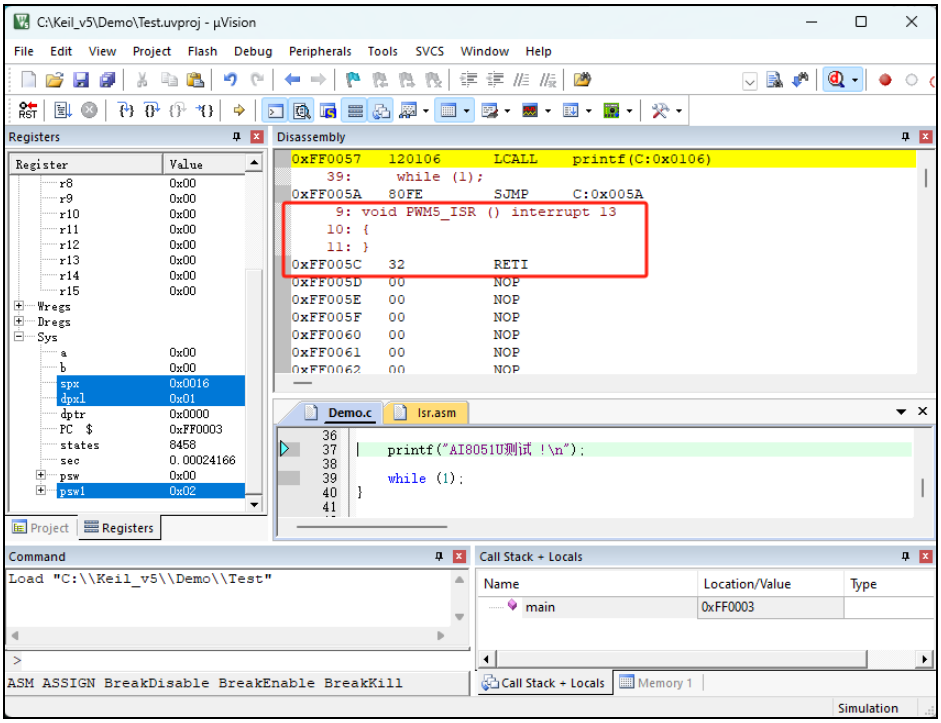


3、编译即可通过。

此时经过 Keil 的 C51 编译器编译后，在 006BH 处有一条“LJMP　　PWM5_ISR”，在 0103H 处有一条“LJMP　　006BH”，如下图：



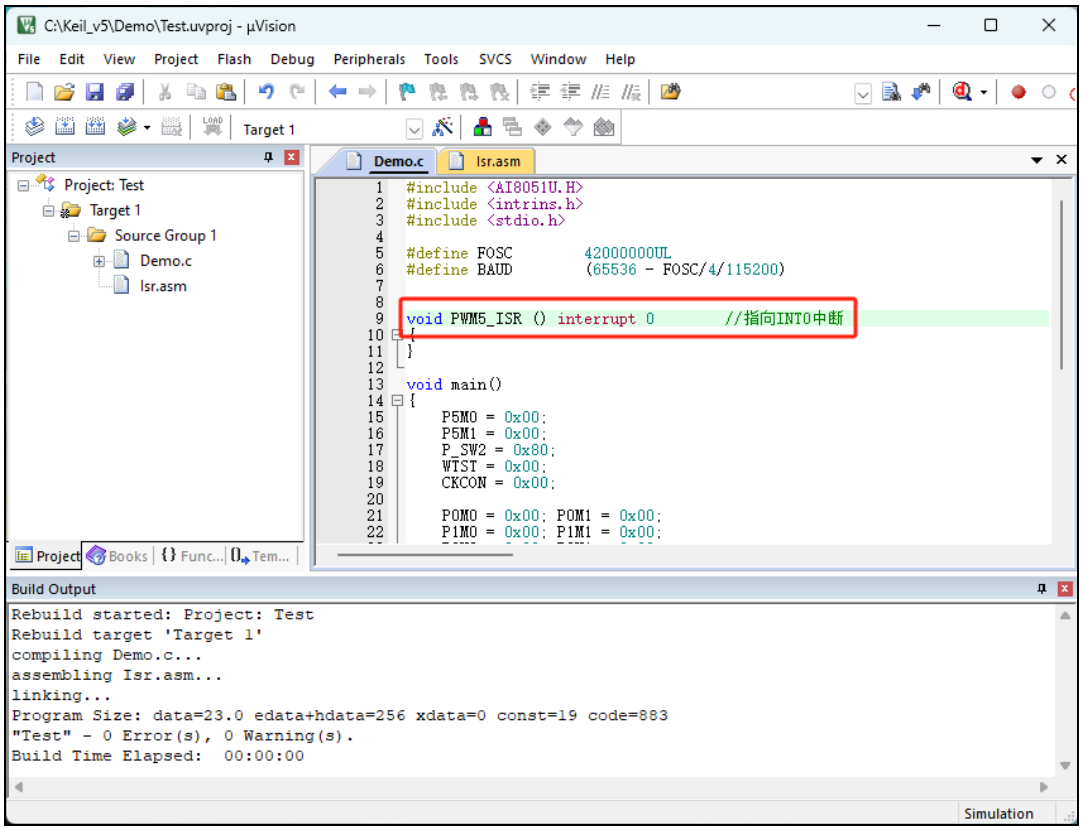
当发生 PWM5 中断时，硬件会自动跳转到 0103H 地址执行“LJMP　　006BH”，然后在 006BH 处再执行“LJMP　　PWM5_ISR”即可跳转到真正的中断服务程序，如下图：

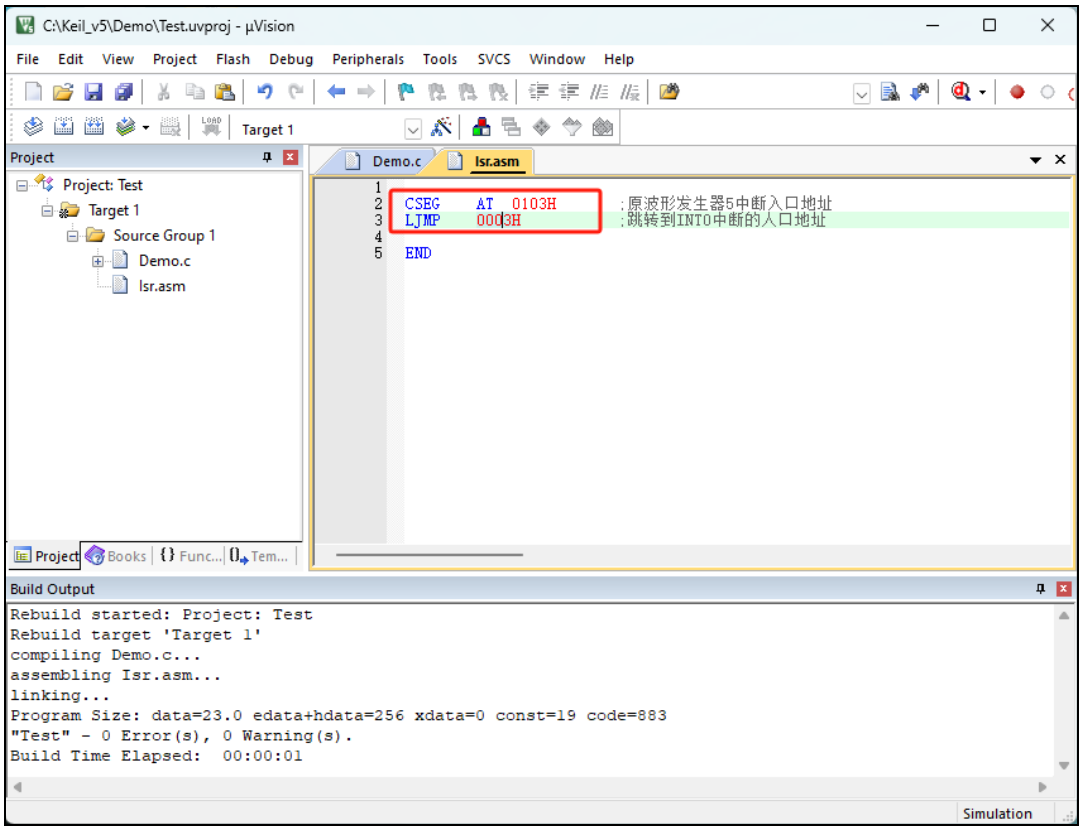


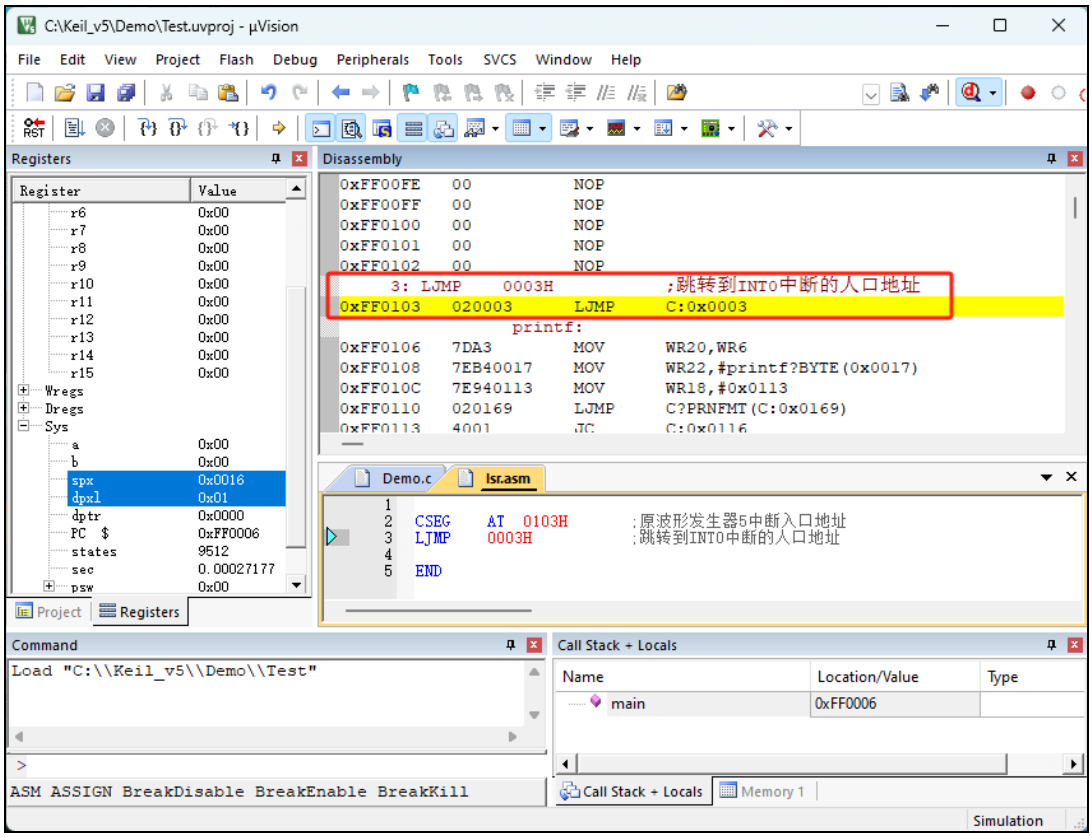
中断服务程序执行完成后，再通过 RETI 指令返回。整个中断响应过程只是多执行了一条 LJMP 语句而已。

方法 2：与方法 1 类似，借用用户程序中未使用的 0~31 的中断号

比如在用户的代码中，没有使用 INT0 中断，则可将上面的代码作类似与方法 1 的修改：





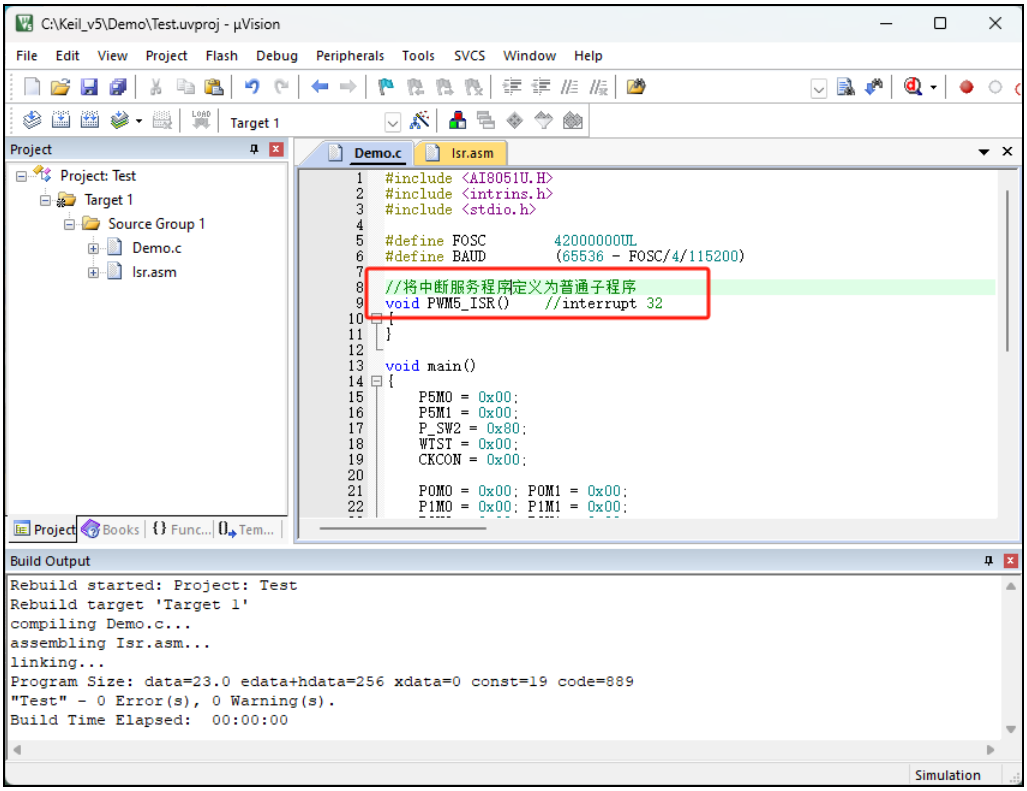


执行效果与方法 1 相同，此方法适用于需要重映射多个中断号大于 31 的情况。

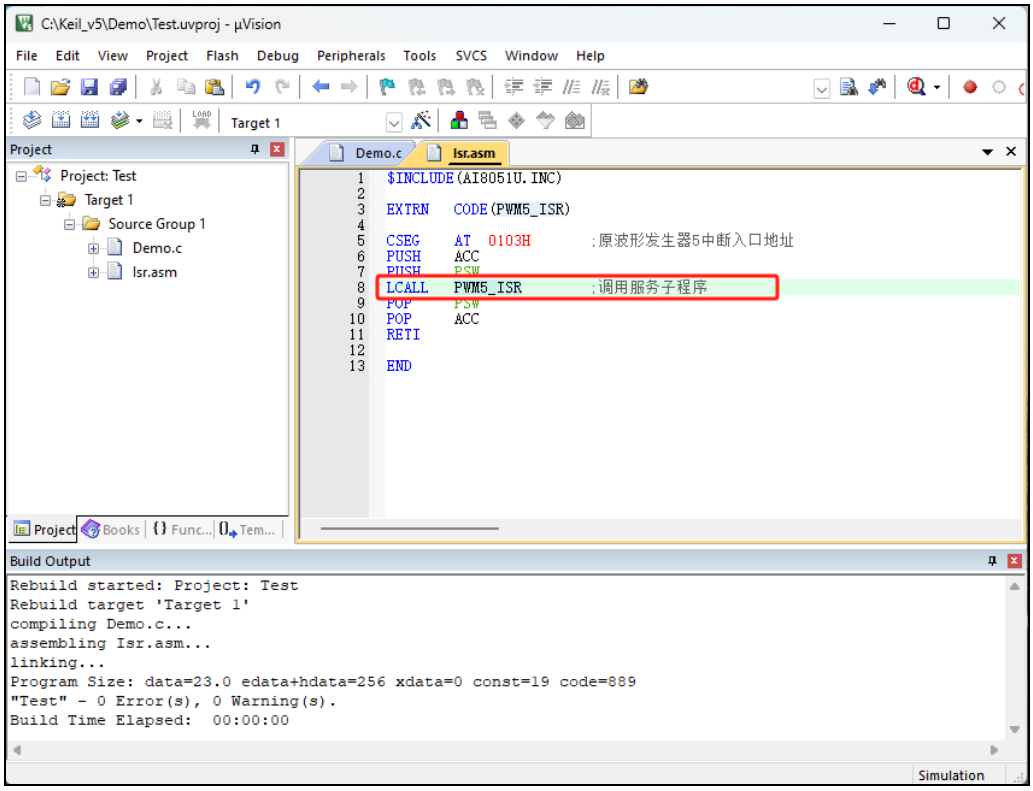
方法 3：将中断服务程序定义成子程序，然后在汇编代码中的中断入口地址中使用 LCALL 指令执行服务程序

操作步骤如下：

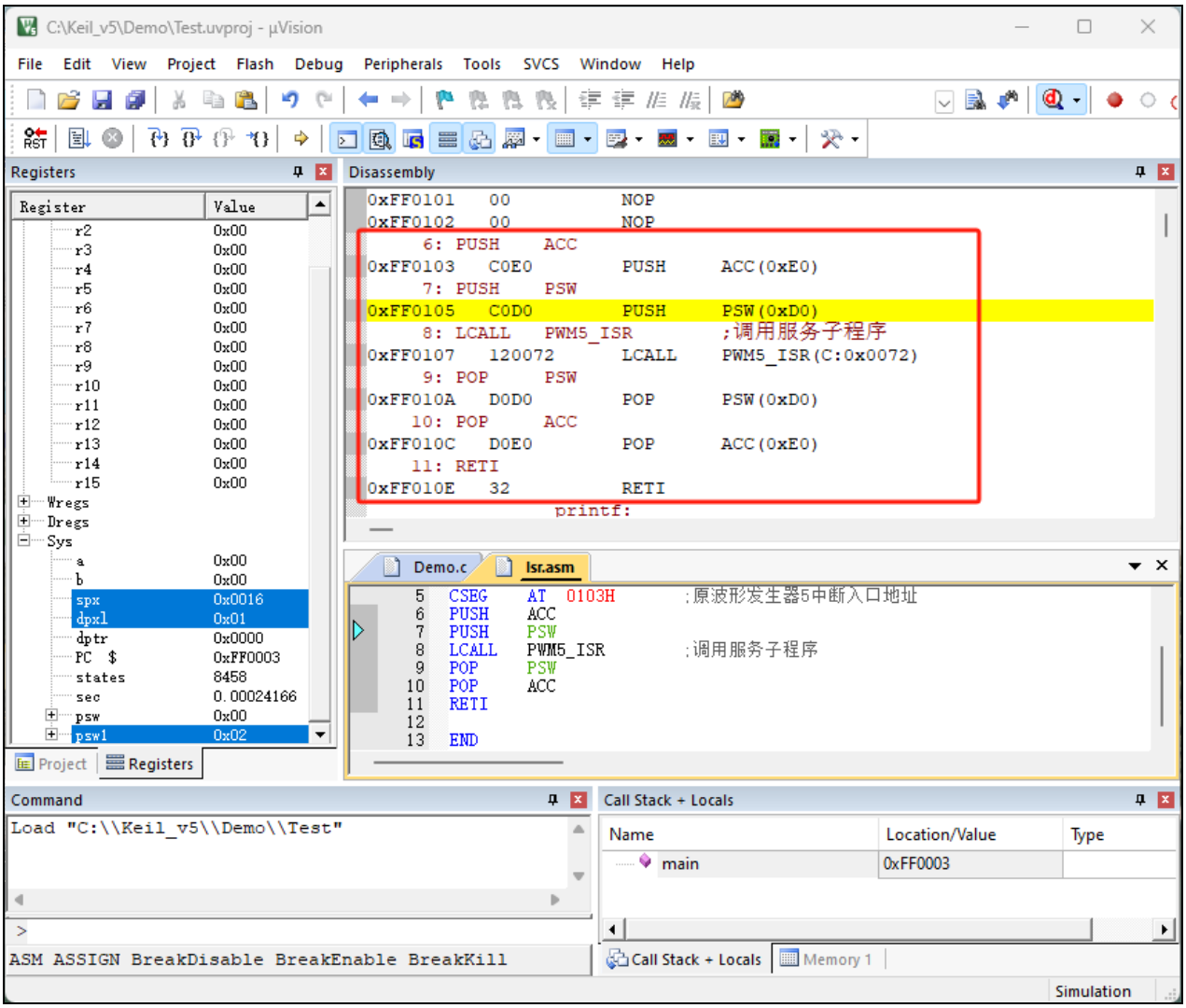
1、首先将中断服务程序去掉“interrupt”属性，定义成普通子程序



2、然后在汇编文件的 0103H 地址输入如下图所示的代码



3、编译通过后，即可发现在 0103H 地址的地方即为中断服务程序



此方法不需要重映射中断入口，不过这种方法有一个问题，在汇编文件中具体需要将哪些寄存器压入堆栈，需要用户查看 C 程序的反汇编代码来确定。一般包括 PSW、ACC、B、DPL、DPH 以及 R0~R7。除 PSW 必须压栈外，其他哪些寄存器在用户子程序中有使用，就必须将哪些寄存器压栈。

2.20　单片机程序中头文件的使用方法

c 语言中 include 用法

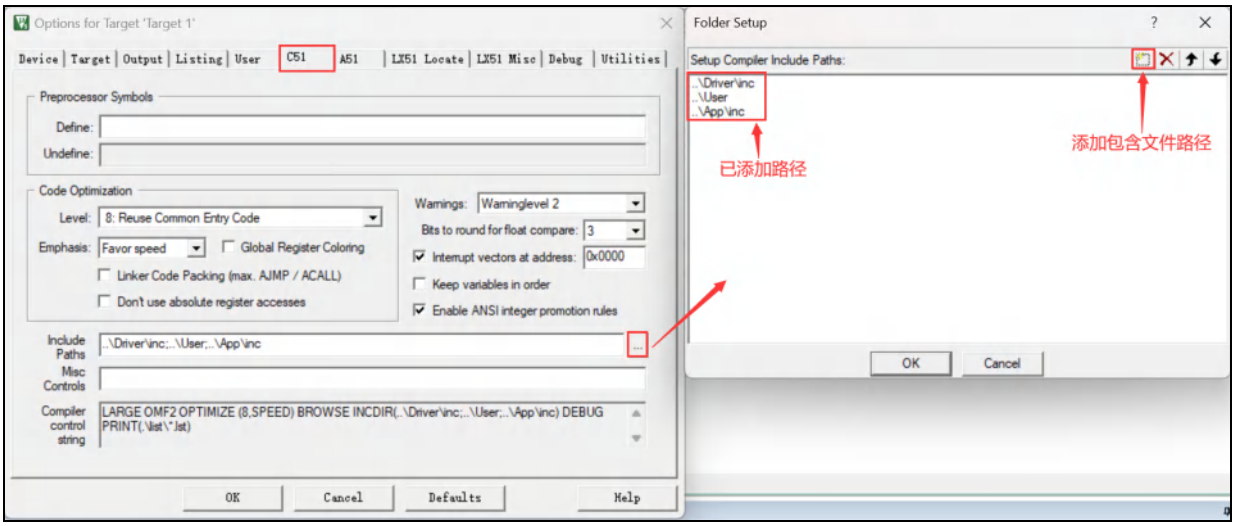
#include 命令是预处理命令的一种，预处理命令可以将别的源代码内容插入到所指定的位置。
有两种方式可以指定插入头文件：

#include <文件名.h>
#include "文件名.h"

使用尖括号< >和双引号" "的区别在于头文件的搜索路径不同：
使用尖括号< >，编译器会到系统路径下查找头文件；
使用双引号" "，编译器首先在当前目录下查找头文件，如果没有找到，再到系统路径下查找。

路径设置方式 1：

通过 keil 设置界面，添加包含文件的路径：



添加后，调用时直接使用 #include "文件名.h" 就可以将需要的文件包含进来，编译器会自动到以上路径下面寻找所包含的文件。
这种情况下，使用双引号" "包含头文件，编译器首先在当前目录下查找头文件，如果没有找到，编译器会到 keil 设置路径查找，还没有的话再到系统路径下查找。（注：系统路径是编译器安装位置存放头文件的目录）

路径设置方式 2：

在包含文件名前添加绝对路径，例如：

#include "E:\xxxx\xxxx\文件名.h"
#include "E:/xxxx/xxxx/文件名.h"

路径设置方式 3：

在包含文件名前添加相对路径，例如：

```
#include "..\comm\文件名.h"
#include "../comm/文件名.h"
```

其中 ".."是指上一级目录，以上路径是指包含文件在当前目录的上一级目录的 comm 目录下面。

汇编语言中 include 用法与 c 语言类似，将"#"换成"$"，用小括号()包含文件：

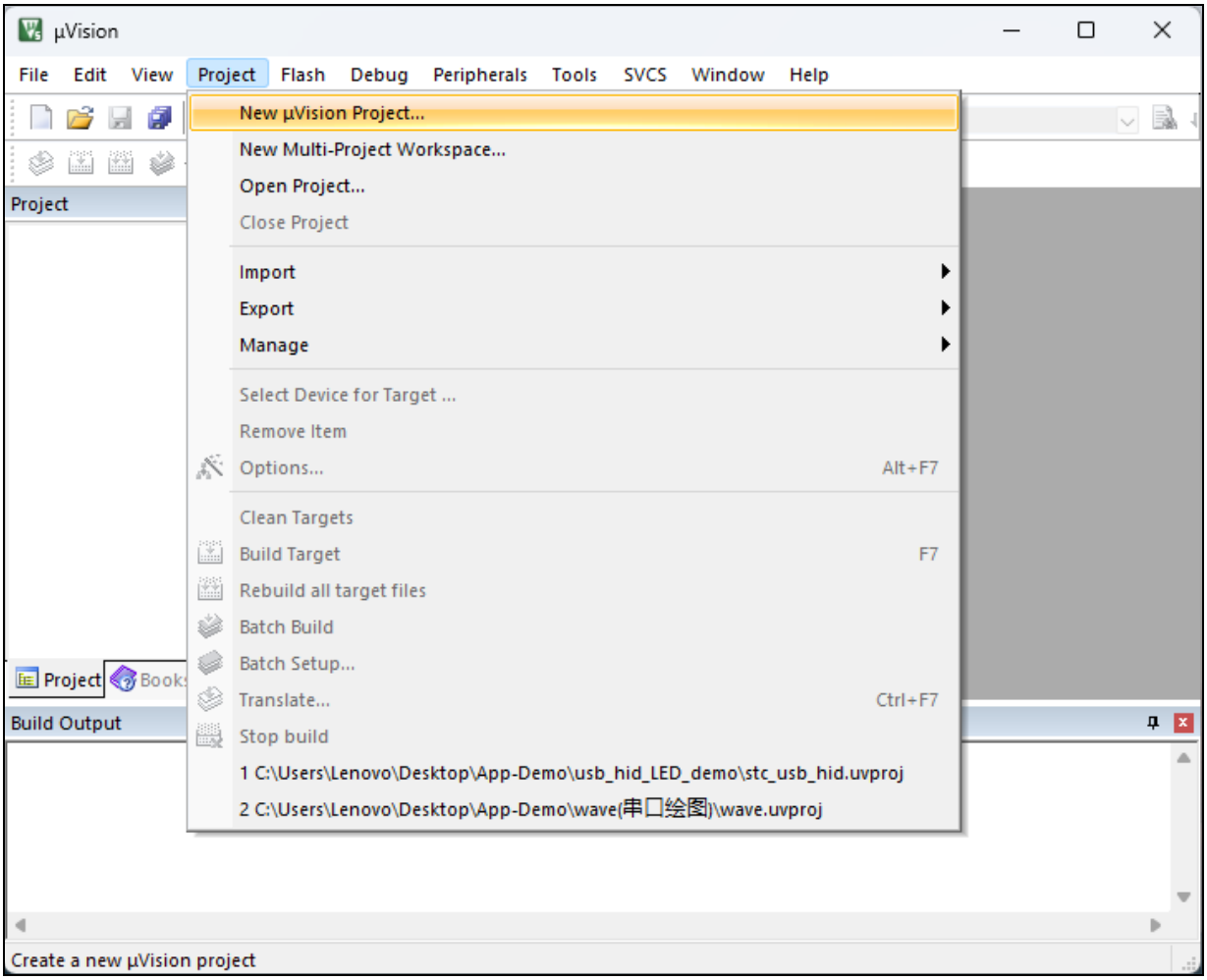
```
$include (../../comm/AI8H.INC)
```

以上指令表示要包含的文件 AI8H.INC，在当前目录的上一级目录的上一级目录的 comm 目录下面。

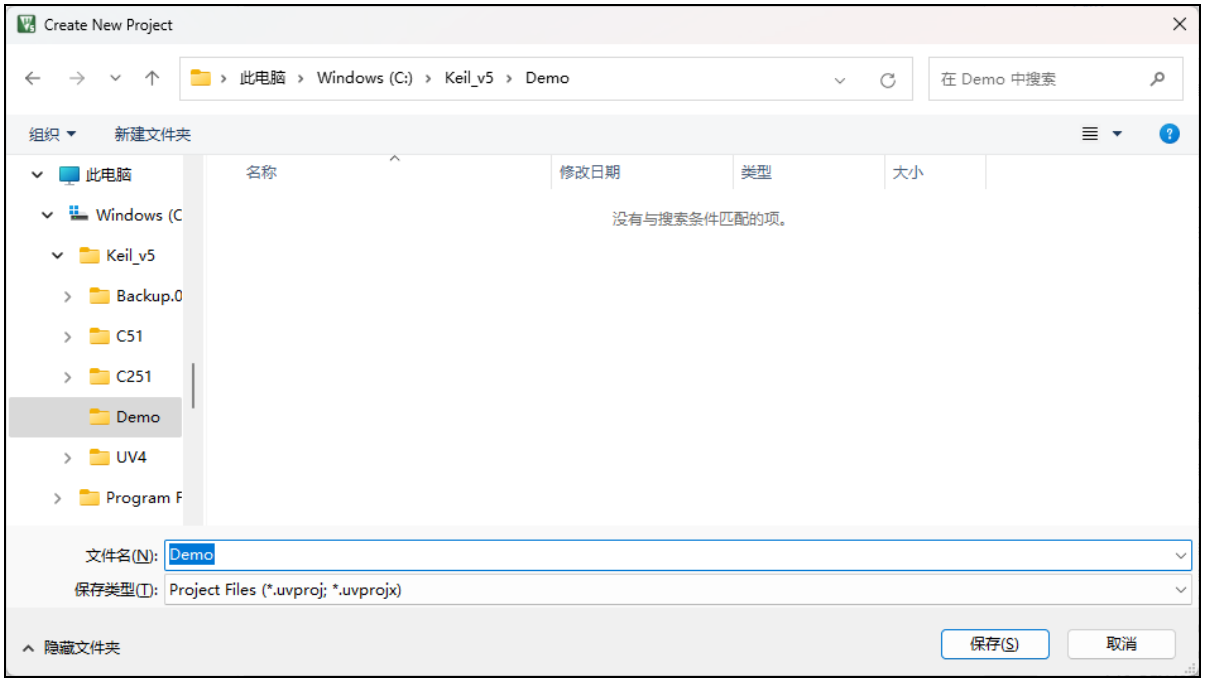
2.21 超过 64K 代码的项目（Source 模式）设置为 Huge，不超过选择 Large

2.21.1 设置项目路径和项目名称

打开 Keil 软件，并点击“Project”菜单中的“New uVision Project ...”项

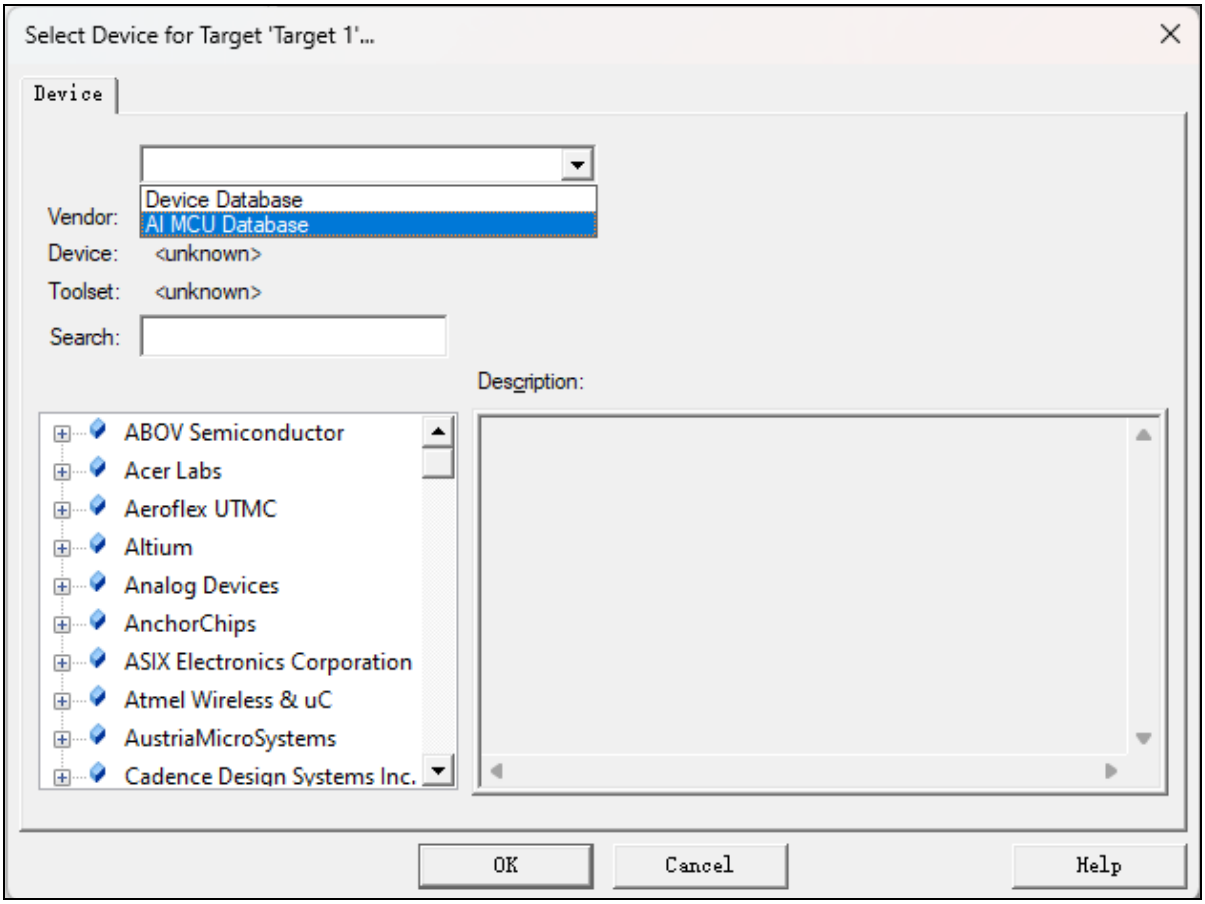


将目录定位在准备好的项目文件夹中，并输入项目名称（例如：Demo）

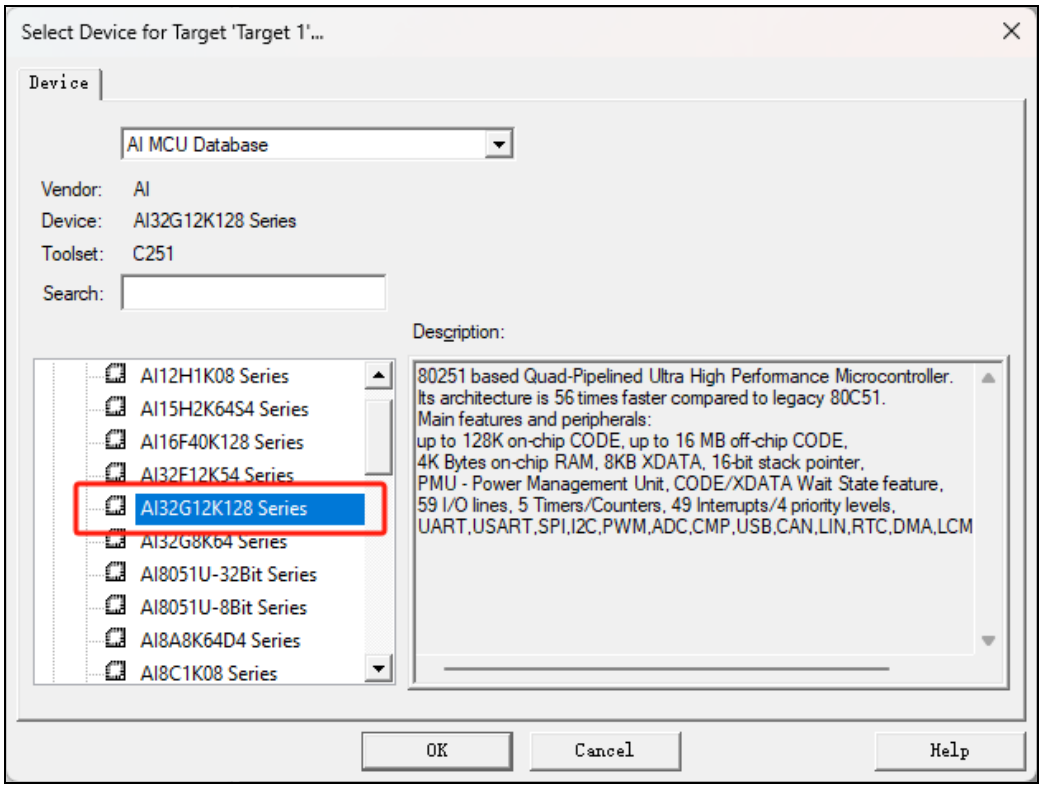


2.21.2　选择目标单片机型号

在弹出的“Select a CPU Data Base File”窗口中选择“AI MCU Database”

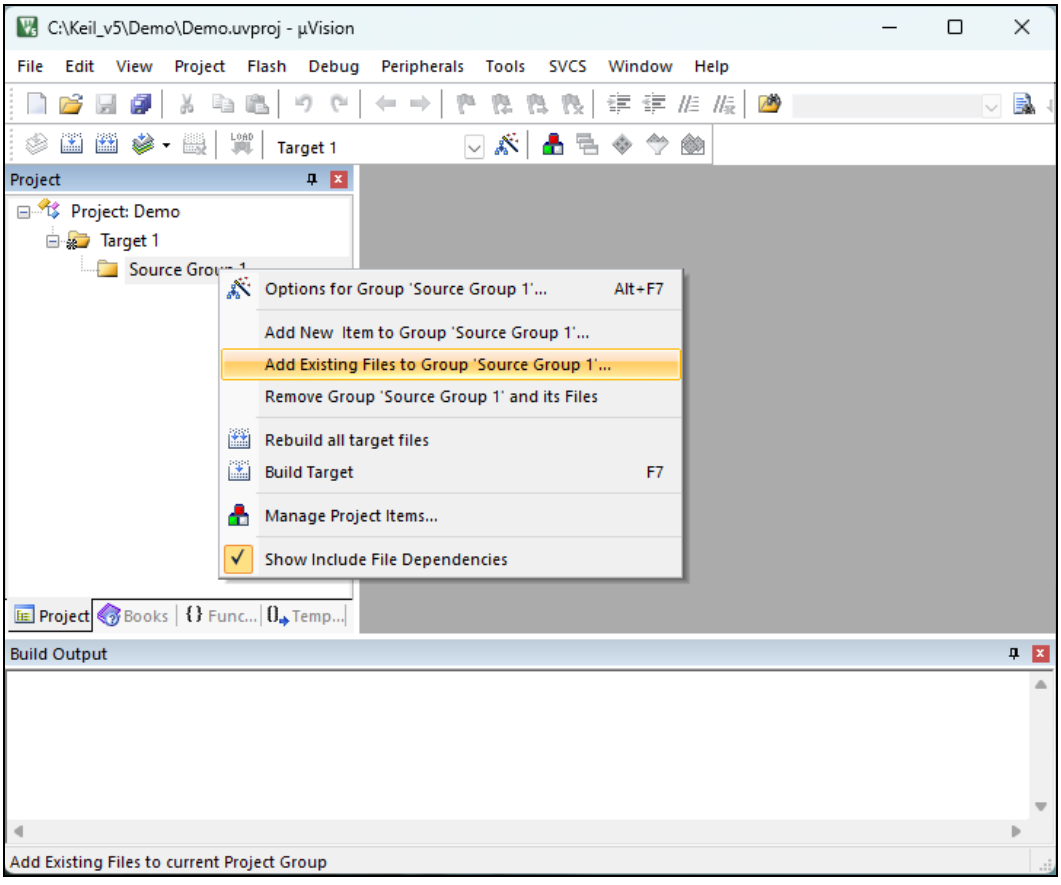


在“Select Device for Target ...”窗口中选择正确的目标单片机型号（例如：AI32G12K128）

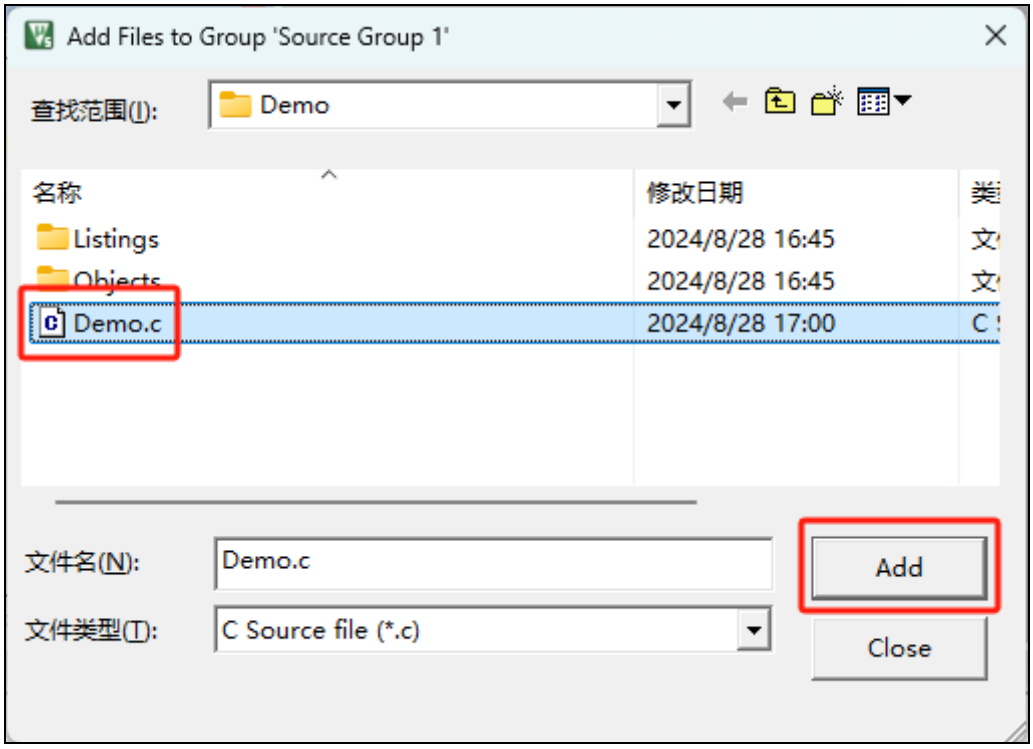


2.21.3　添加源代码文件到项目

如下图所示，在“Source Group 1”所在的图标点击鼠标右键，并选择右键菜单中的“Add Existing Files to Group ' Source Group 1'...”

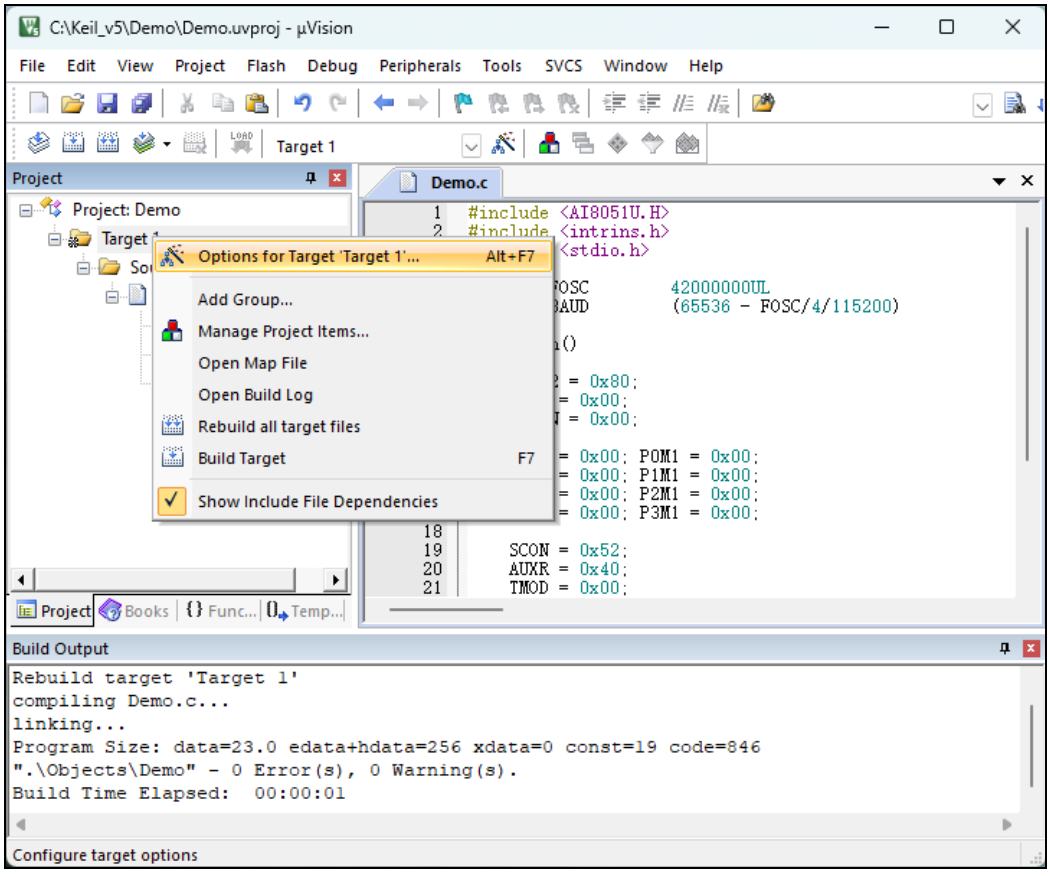


选择已编辑完成的代码文件加入到项目中

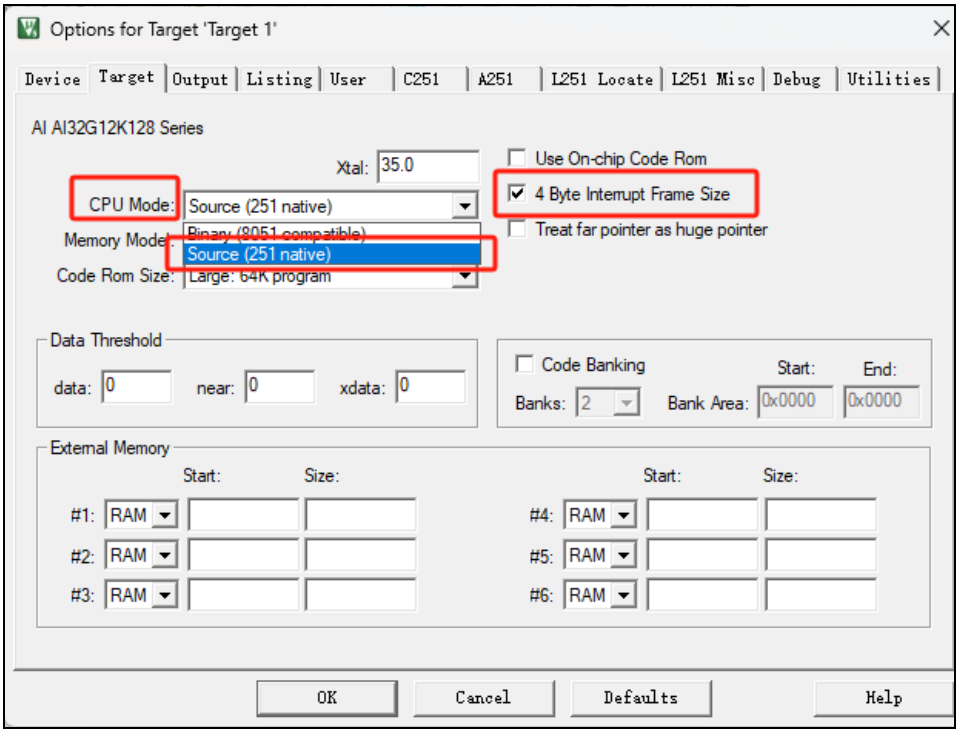


2.21.4　设置 1（“CPU Mode”选择 Source 模式）

如下图所示，在“Target1”所在的图标点击鼠标右键，并选择“Options for Target ' Target 1'...”



在弹出的“Options for Target ' Target 1'”窗口中选择“Target”选项页，在“CPU Mode”的下拉选项中选择“Source (251 Native)”

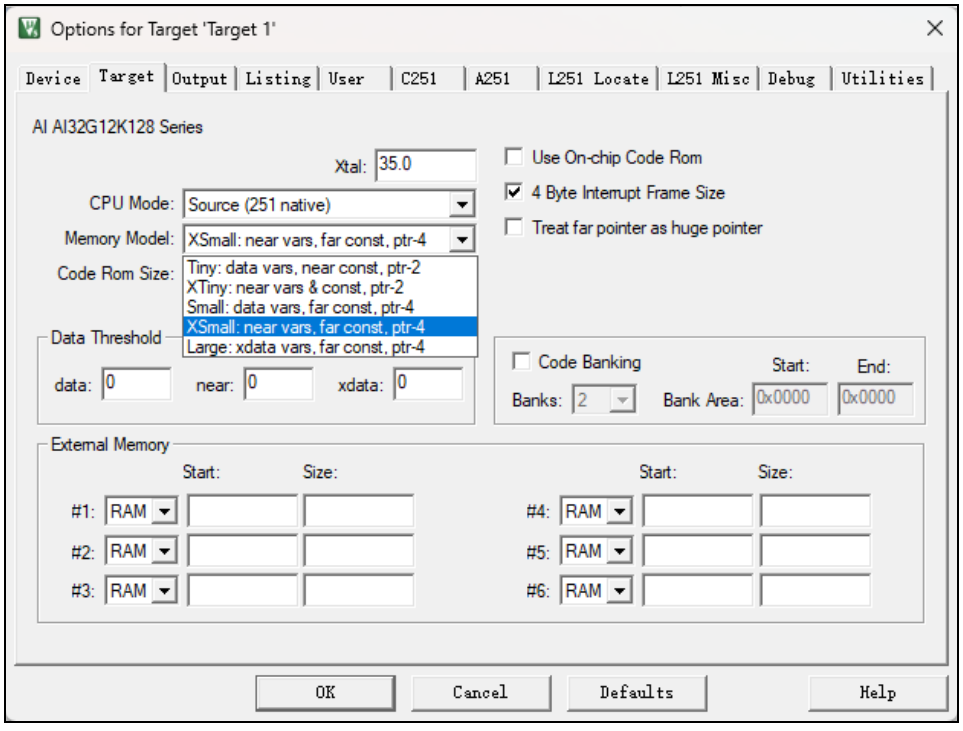


80251 的指令模式有“Binary”和“Source”两种模式，Ai8051U 系列目前只支持“Source”模式

由于 Ai8051U 系列单片机在中断中的压栈和出栈都是 4 字节模式，建议“4 Byte Interrupt Frame Size”选项也打上钩

2.21.5　设置 2（“Memory Model”选择 XSmall 模式）

在“Memory Model”的下拉选项中选择“XSmall: ...”模式。80251 的存储器模式，在 Keil 环境下有如下图所示的 5 种模式：



各种模式对比如下表：

Memory Model	默认变量类型（数据存储器）	默认常量类型（程序存储器）	默认指针变量	指针访问范围
Tiny 模式	data	near	2 字节	00:0000 ~ 00:FFFF
XTiny 模式	edata	near	2 字节	00:0000 ~ 00:FFFF
Small 模式	data	far	4 字节	00:0000 ~ FF:FFFF
XSmall 模式	**edata**	**far**	**4 字节**	**00:0000 ~ FF:FFFF**
Large 模式	xdata	far	4 字节	00:0000 ~ FF:FFFF

由于 Ai8051U 的程序逻辑地址为 FF:0000H~FF:FFFFH，需要使用 24 位地址线才能正确访问，默认的常量类型（程序存储器类型）必须使用“far”类型，默认指针变量必须为 4 字节。

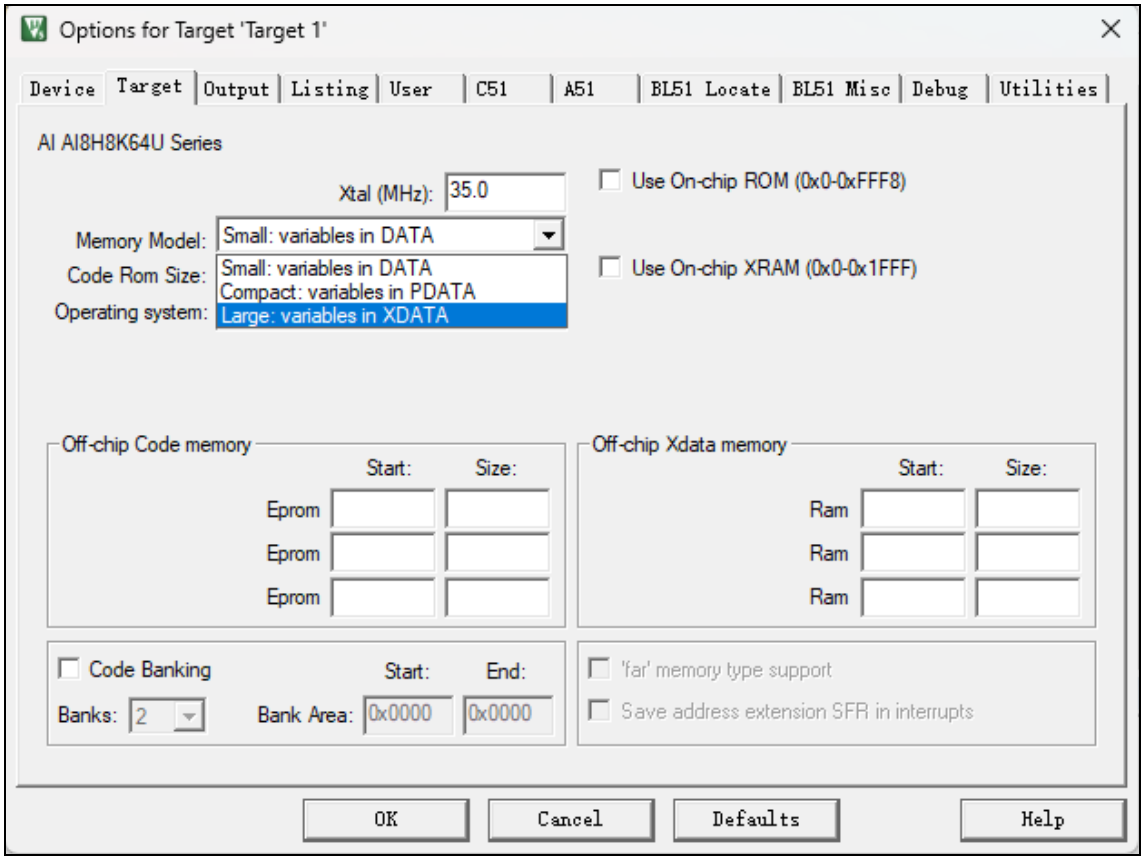
不建议使用“Small”“Tiny”和“XTiny”模式，推荐使用“XSmall”模式，这种模式默认将变量定义在内部 RAM(edata)，单时钟存取，访问速度快，且 Ai8051U 系列芯片有 2K 的 edata 可以使用；使用“Small”模式时，默认将变量定义在内部 RAM(data)，data 默认只有 128 字节，当用户对 RAM 需求超过 128 字节时，Keil 编译器会报错，data 区数量有限，容易报错，所以不建议使用；不推荐使用“Large”模式，虽然该模式也能正确访问 Ai8051U 的全部 16M 寻址空间，但“Large”模式默认将变量定义在内部扩展 RAM(xdata)里面，存取需要 2~3 个时钟，访问速度慢

注意：当项目编译时出现如下错误提示时表示 edata 已经超出当前单片机内部 edata 的容量了，则需要您强制使用 xdata 将部分变量分配到 XRAM（例如将数组强制指定为 xdata 类型：int xdata buffer[256]; ）

```
Build target 'Target 1'
compiling Test.c...
linking...
*** ERROR L107: ADDRESS SPACE OVERFLOW
    SPACE:   EDATA
    SEGMENT: ?ED?TEST
    LENGTH:  001400H
Program Size: data=8.0 edata+hdata=5416 xdata=0 const=49 code=1021
Target not created
```

Ai8051U 系列 edata 大小为 2K

与之相对应的 AI8H8K64U 系列，在 Keil 软件中的“Memory Model”有如下 3 个选择



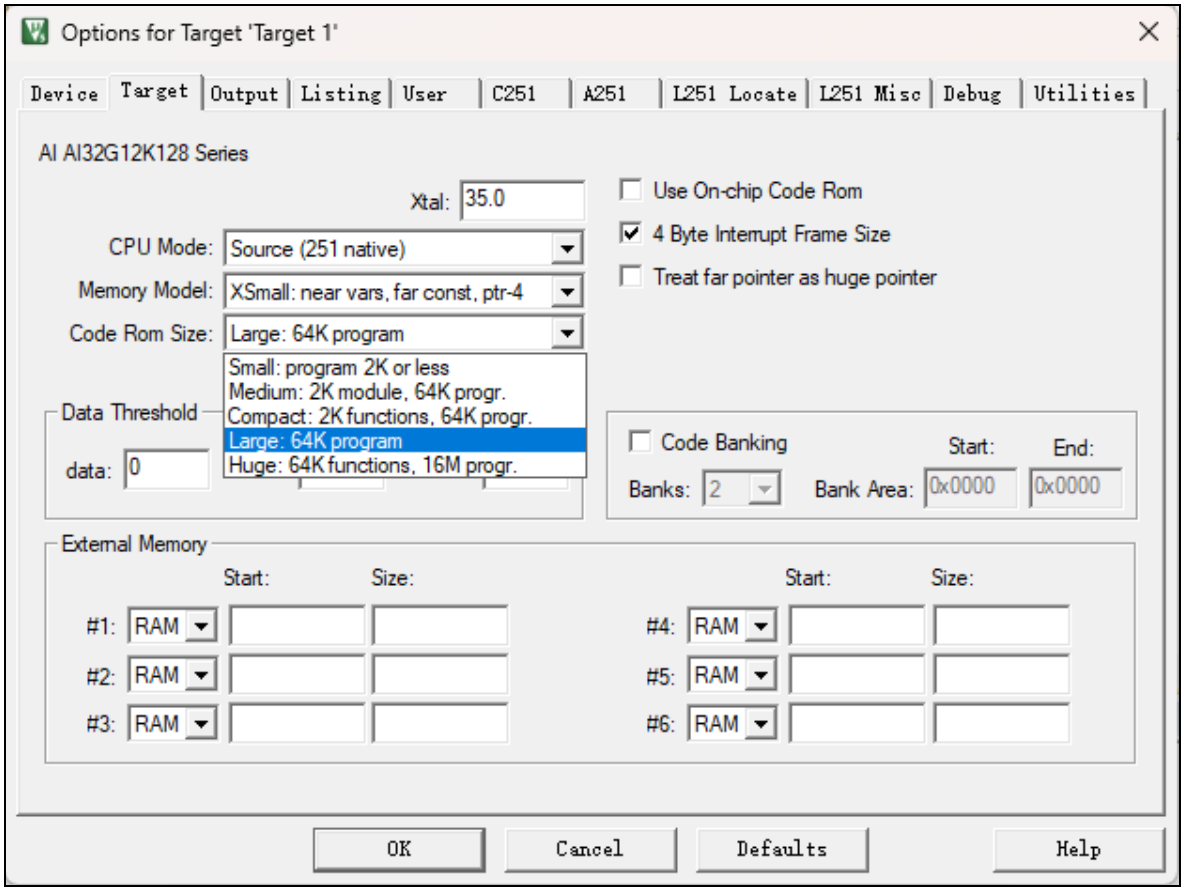
各种模式对比如下表：

Memory Model	默认变量类型（数据存储器）	存储器大小	地址范围
Small 模式	data	128 字节	D:00 ~ D:7F
Compact 模式	pdata	256 字节	X:0000 ~ X:00FF
Large 模式	**xdata**	**64K 字节（理论值）**	**X:0000 ~ X:FFFF**

为了达到比较高的效率，一般建议选择“Small”模式，当编译器出现“error C249: 'DATA': SEGMENT TOO LARGE”错误时，则需要手动将部分比较大的数组通过“xdata”强制分配到 XDATA 区域（例如：char xdata buffer[256];）

2.21.6　设置 3，Code Rom Size：超 64K 选 Huge，小于 64K 选 Large

在“Code Rom Size”的下拉选项中选择“Large: ...”或者“Huge: ...”模式 80251 的代码大小模式，在 Keil 环境下有如下图所示的 5 种模式：

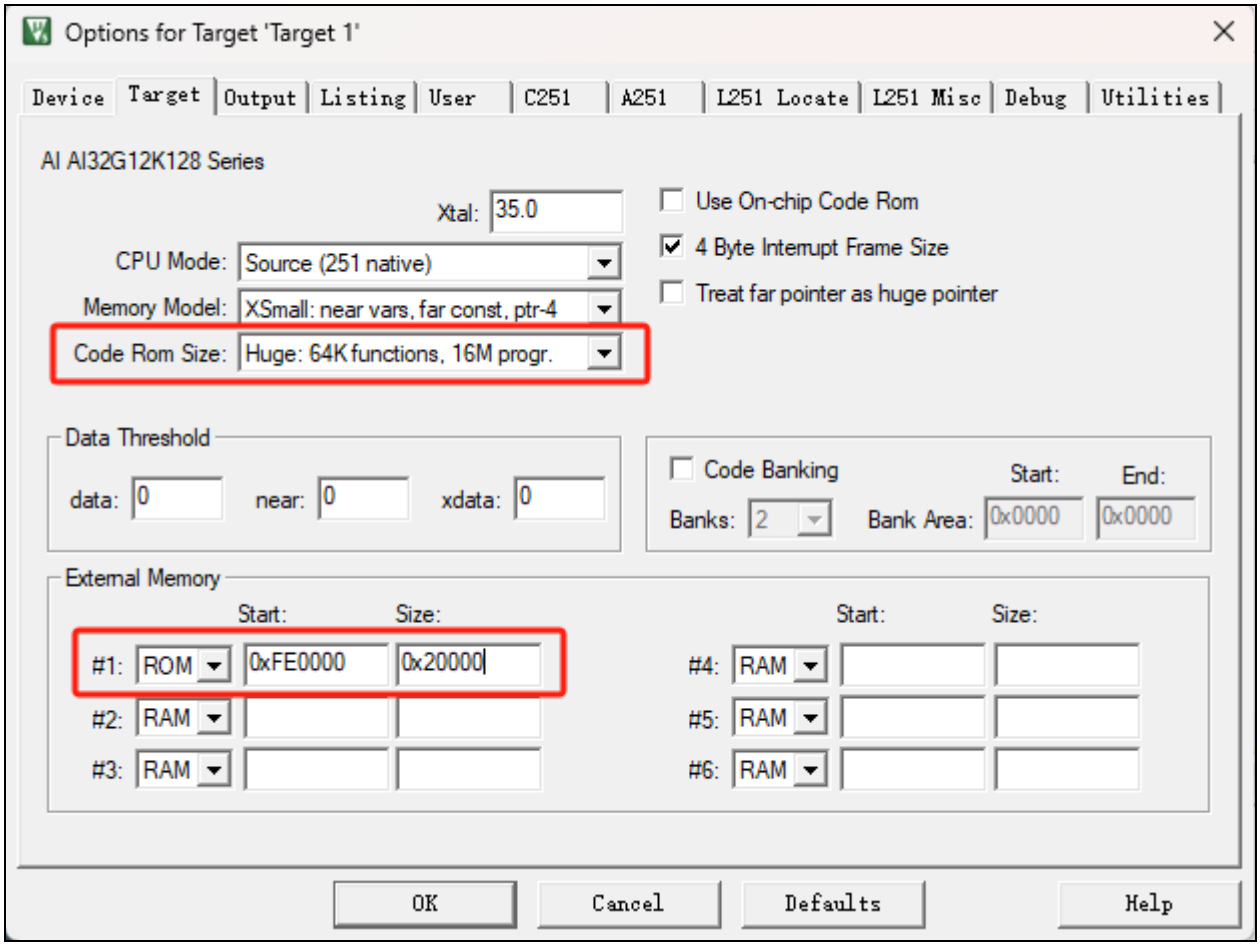


各种模式对比如下表：

Code Rom Size	跳转/调用指令	代码大小限制	
		单个函数/模块/文件的代码大小	总代码大小
Small 模式	AJMP/ACALL	2K	2K
Medium 模式	内部模块代码使用 AJMP/ACALL 外部模块代码使用 LJMP/LCALL	2K	64K
Compact 模式	LCALL/AJMP	2K	64K
Large 模式	LCALL/LJMP	64K	64K
Huge 模式	内部模块代码使用 LJMP/ECALL 外部模块代码使用 EJMP/ECALL （多文件项目中，文件内部的代码为内部模块代码，其他文件的代码为外部模块代码）	64K	16M

2.21.7　设置 4（超 64K 代码的相关设置）

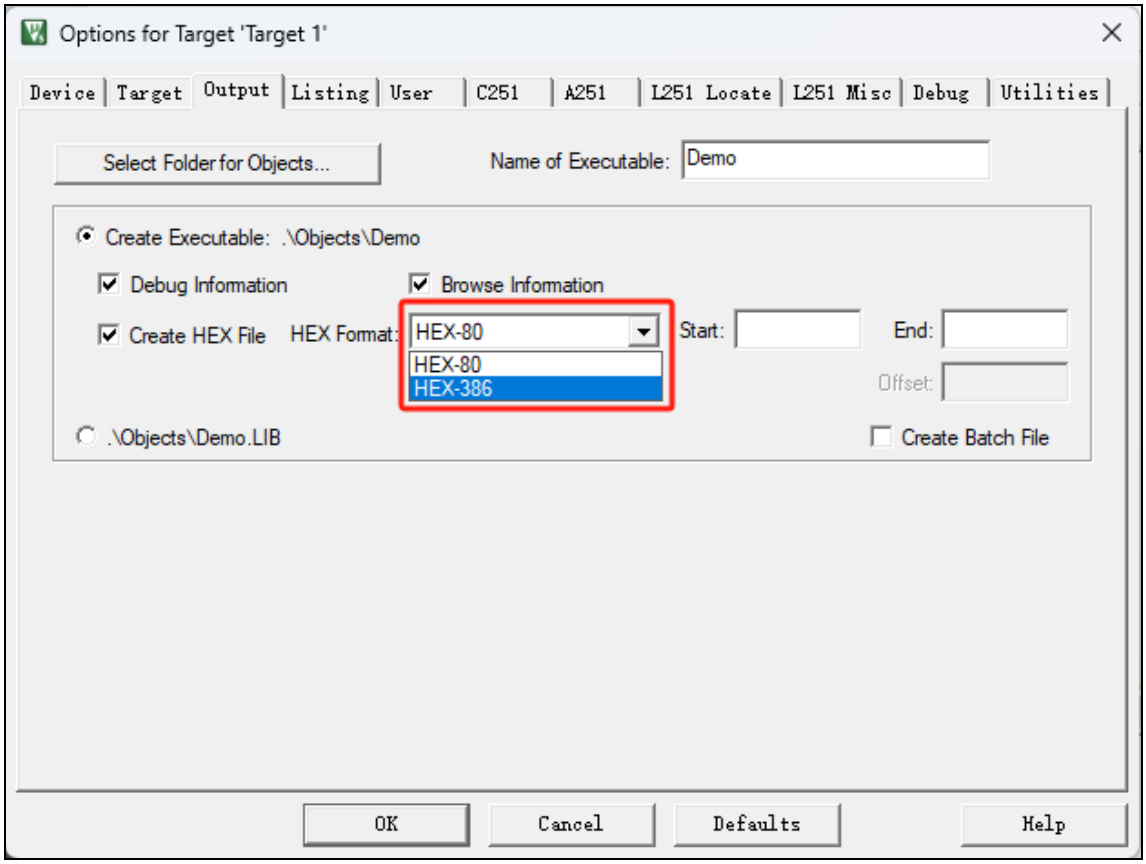
如果代码大小在 64K 以内，选择“Large”模式即可。若代码大小超过 64K，则需要选择“Huge”模式，并需要保证单个函数以及单个文件的代码大小必须在 64K 字节以内，并且单个表格的数据量也必须在 64K 字节以内。同时还需要作如下图所示的设置：



请务必注意：设置的是 ROM 区域

2.21.8　设置 5（HEX 文件格式设置）超 64K 选 Hex-386，小于 64K 可选 Hex-80

“Options for Target ' Target 1'”窗口中选择“Output”选项页，勾选其中的“Create HEX File”选项。若程序空间超过 64K，则“HEX format”必须选择“HEX-386”模式，只有程序空间在 64K 以内，“HEX format”才可选择 “HEX-80”模式；



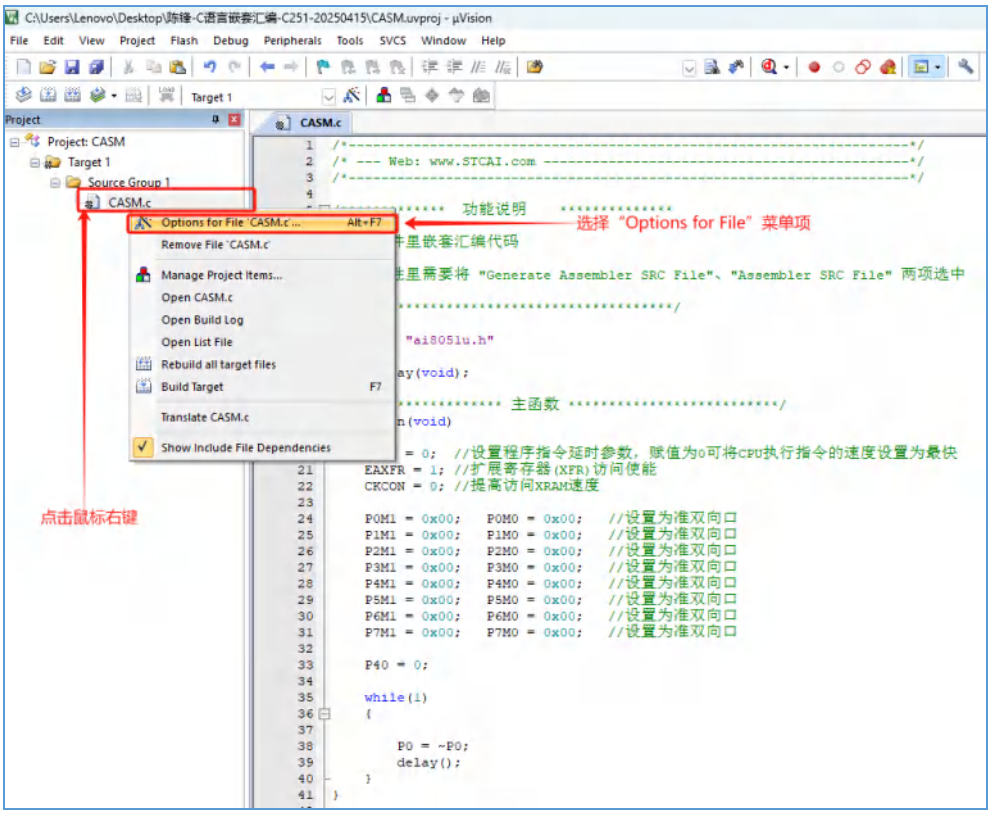
完成上面的设置后，鼠标单击如下图所示的编译按钮，如果代码没有错误，即可生成 HEX 文件

2.22　嵌入汇编代码在 C 语言中，Keil C251 编译器

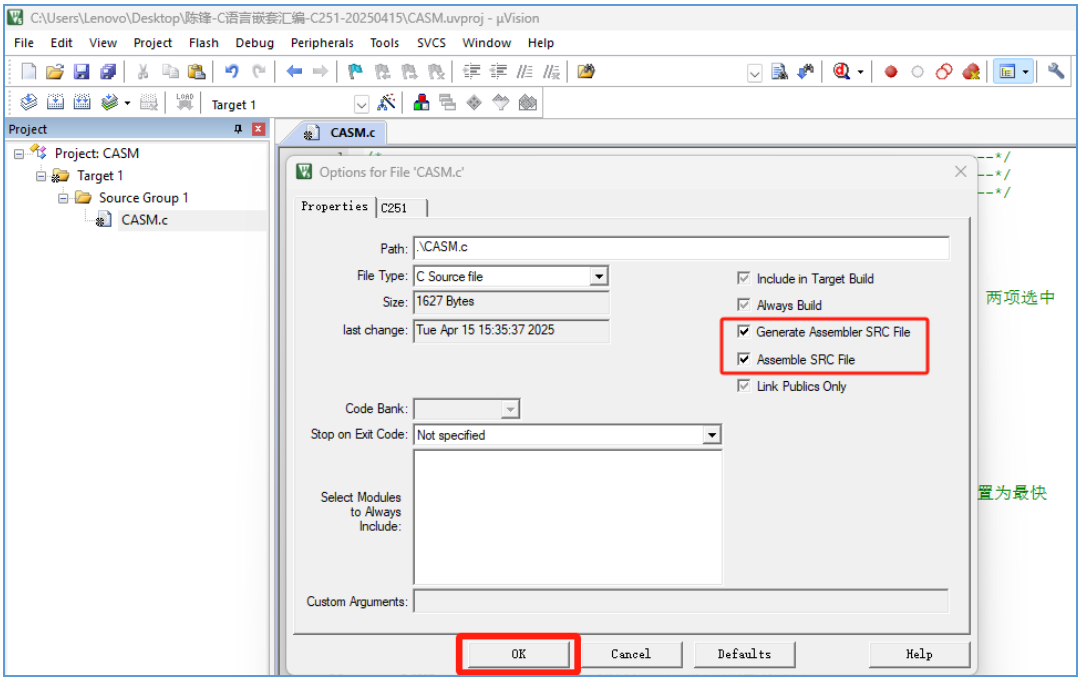
有时为了直接访问硬件资源、优化代码执行速度或精度、实现某些 C 语言本身难以直接表达的算法，需要在 C 语言文件里嵌套汇编代码，以便更直接地利用芯片特性或优化性能。

第 1 步：新建一个工程，添加 C 文件，编译通过

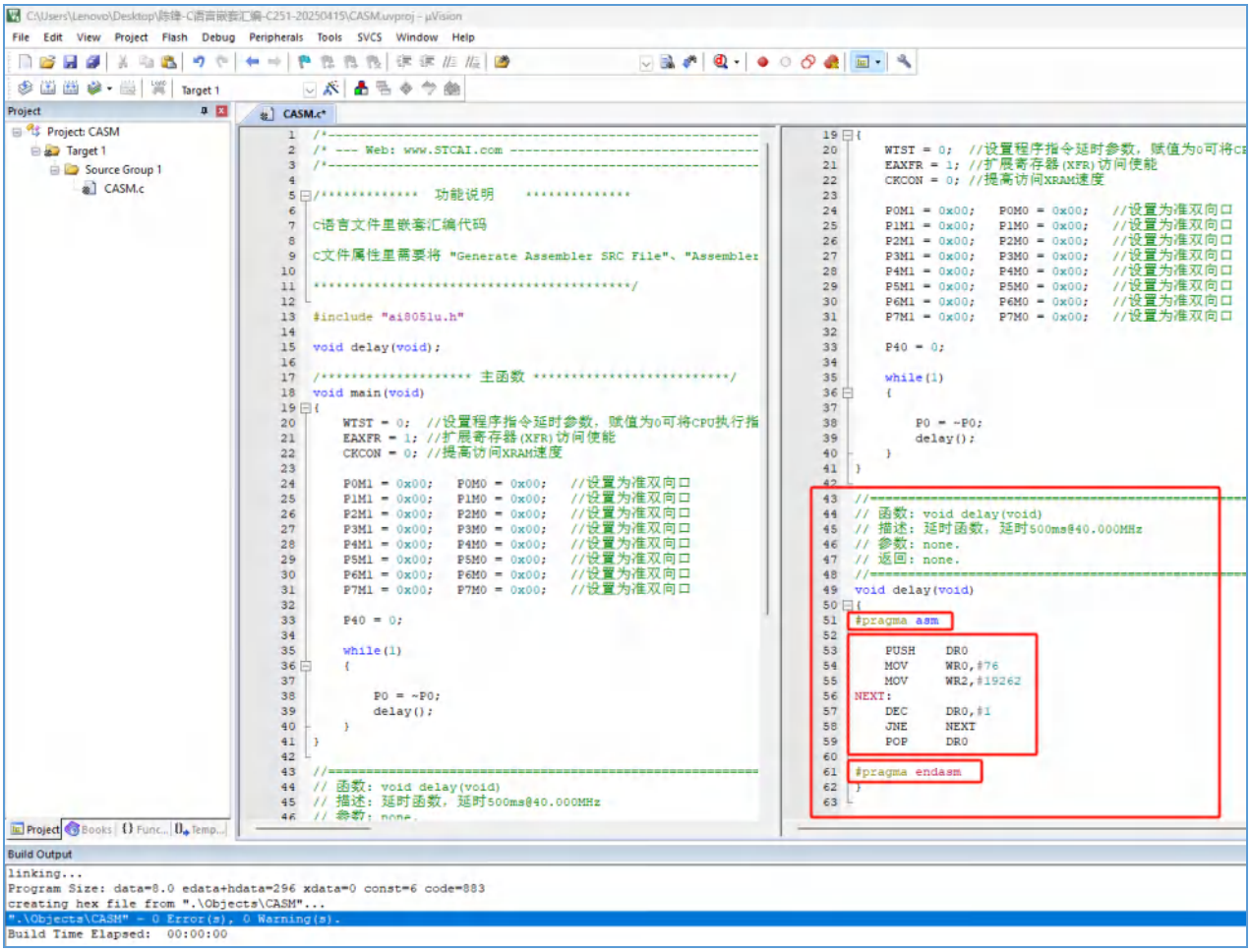
第 2 步：右击 Keil 界面中 Project Workspace 里的 C 文件名，选择‘Option for …’



第 3 步：点击‘Generate Assembler SRC File’、‘Assemble SRC File’两个三态复选框，使它们呈高亮并勾选，点击“OK”



第 4 步：添加汇编代码，以“#pragma asm”为始，以“#pragma endasm”为止。保存、编译，如下图：



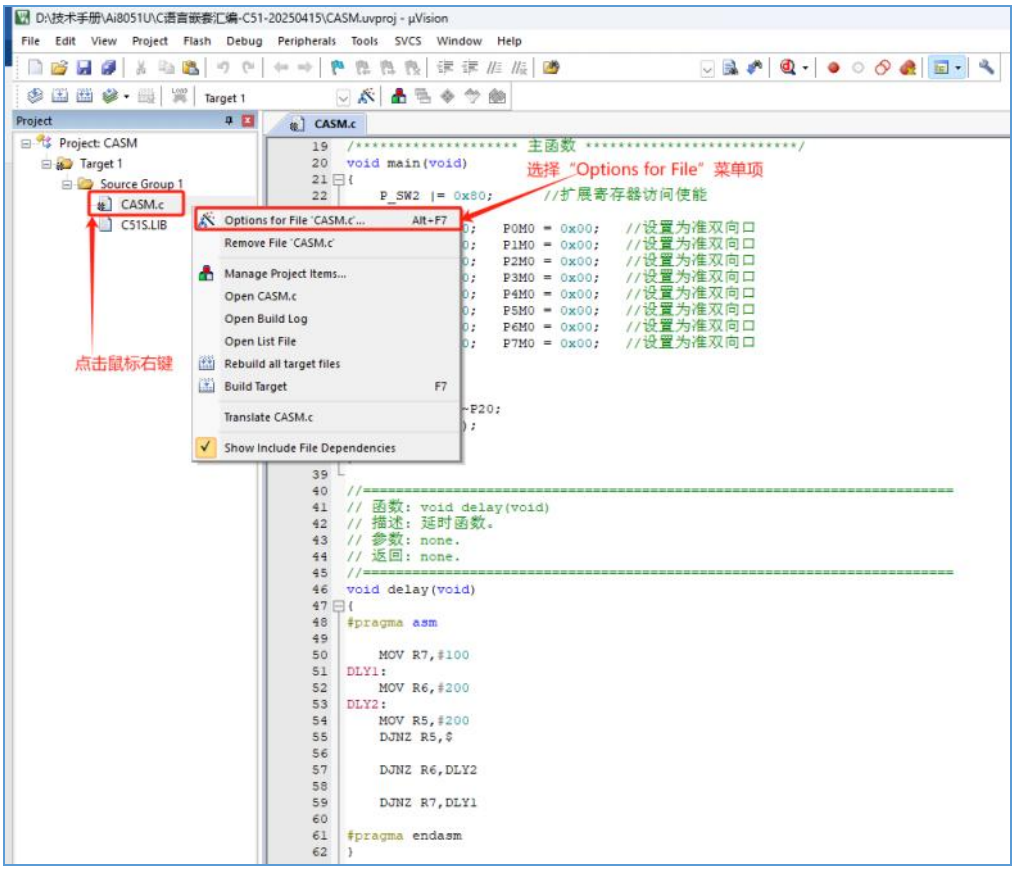
连接 Ai8051U 实验箱，打开 AIapp-ISP 系统软件，打开并下载该项目的 HEX 文件，Ai8051U 实验箱上的 LED00~LED07 就闪烁起来了。

2.23　嵌入汇编代码在 C 语言中，Keil C51 编译器

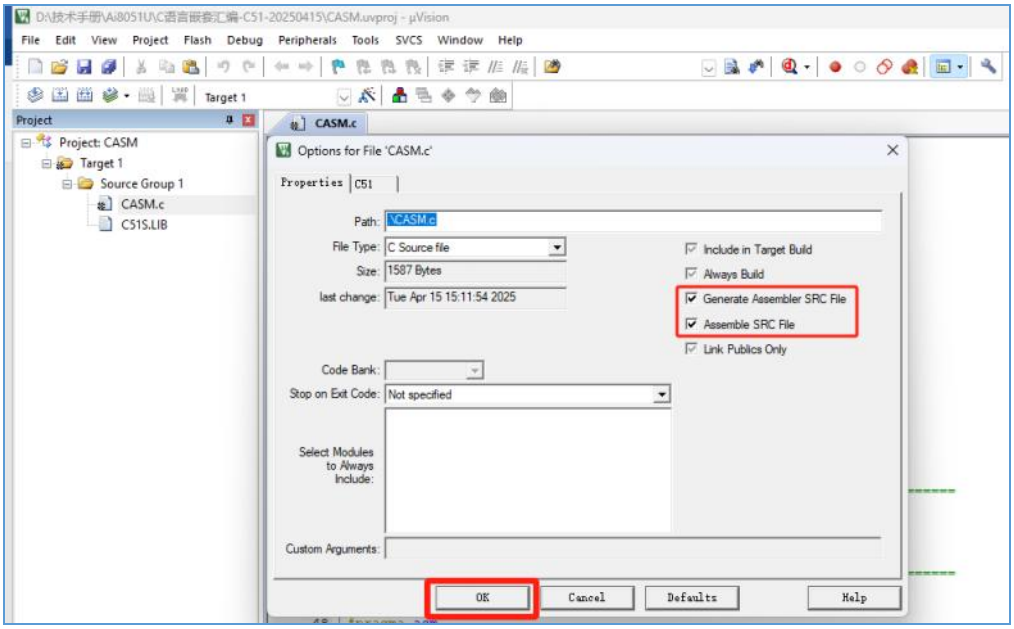
有时为了直接访问硬件资源、优化代码执行速度或精度或实现某些 C 语言本身难以直接表达的算法，需要在 C 语言文件里嵌套汇编代码。

第 1 步：新建一个工程，添加 C 文件，编译通过

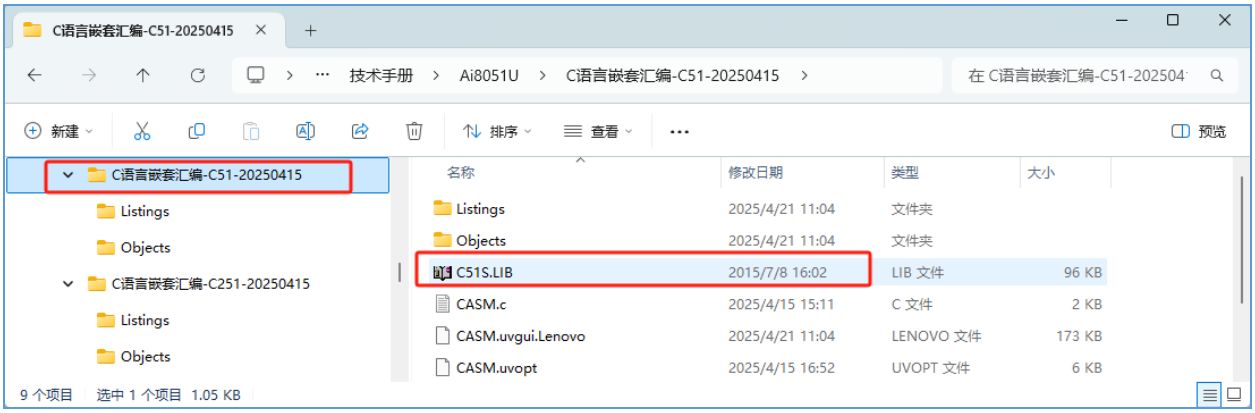
第 2 步：右击 Keil 界面中 Project Workspace 里的 C 文件名，选择‘Option for…’



第 3 步：点击‘Generate Assembler SRC File’、‘Assemble SRC File’两个三态复选框，使它们呈高亮并勾选，点击“OK”



第 4 步：将 KEIL 安装目录下 C51/LIB/C51S.LIB（如果是 SMALL 模式）文件复制到工程目录下：



第 5 步：添加库文件 C51S.LIB 到工程，即右击 ‘Source Group 1’，选择‘Add Files to Group …’，在工程目录下选择 C51S.LIB，点击打开

【因为有了第 3 步中的操作，如果在这里不添加库文件，则编译生成的 HEX 文件不能被单片机正确执行，同时编译时会出现一个警告：

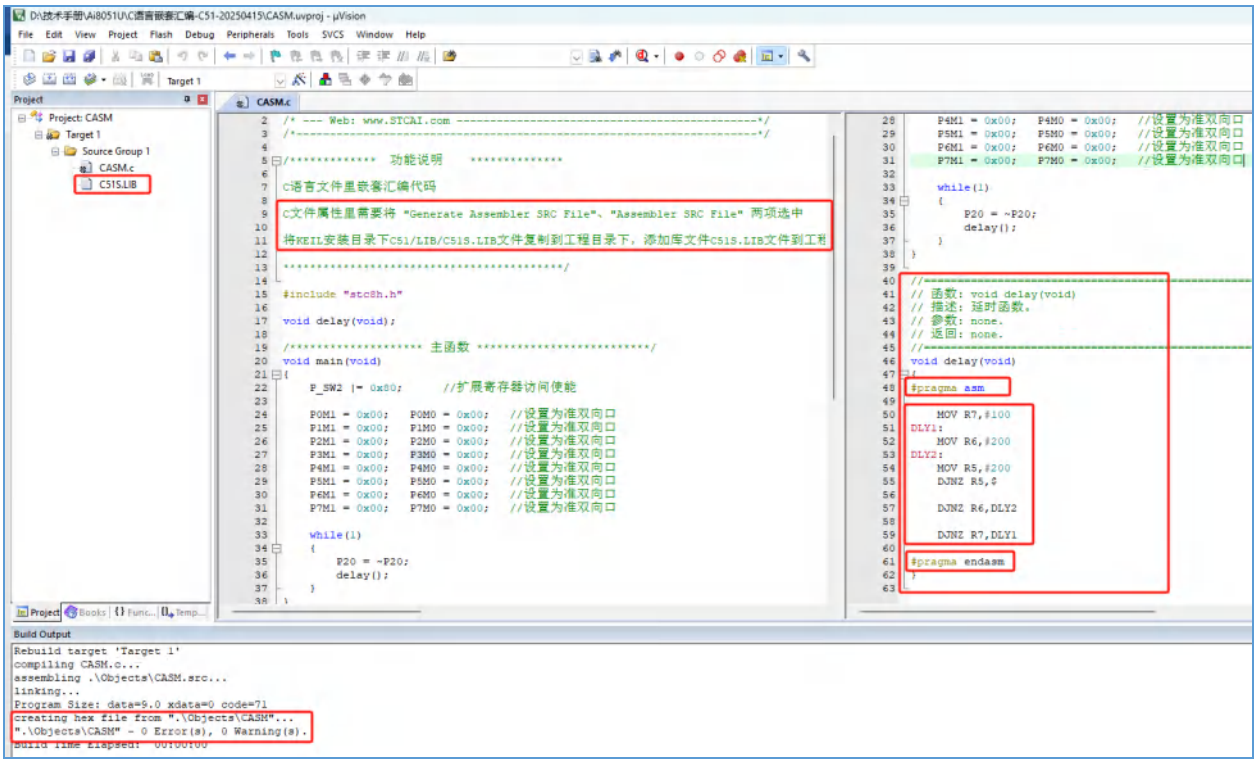
WARNING L1:UNRESOLVED EXTERNAL SYMBOL

SYMBOL:　?C_STARTUP

无法识别?C_STARTUP 这个标号，这个标号是在库文件中定义】

汇编代码，以“#pragma asm”为始，以“#pragma endasm”为止。

第 6 步：保存、重新编译，生成 HEX 文件，加载到单片机中，可以发现与 C 语言运行的现象是一致的，如下图：



连接擎天柱核心功能实验板，打开 AIapp-ISP 系统软件，打开并下载该项目的 HEX 文件，擎天柱上的 P20 的 LED 就闪烁起来了。

2.24　汇编代码演示，在 Keil 中编写 Ai8051U 汇编代码

2.24.1　代码大小在 64K 以内的汇编程序编写方法

```
P0      DATA    080H
P0M1    DATA    093H
P0M0    DATA    094H
WTST    DATA    0E9H
CKCON   DATA    0EAH

        ORG     0000H                   ;复位入口地址
        JMP     START                   ;1.  64K 程序大小的代码,可直接使用 0000H 定义地址
                                        ;    编译器会自动将代码连接到 FF:0000 开始的地方
                                        ;2.  中断向量处可使用 JMP 语句,编译器会自动
                                        ;    根据实际编译情况智能替换成 AJMP/LJMP/EJMP

        ORG     0003H                   ;中断入口地址
        JMP     INT0_ISR
        ORG     000BH
        JMP     TIMER0_ISR
        ORG     0013H
        JMP     INT1_ISR
        ORG     001BH
        JMP     TIMER1_ISR

        ORG     0200H
        NOP

INT0_ISR:                               ;中断函数
        NOP
        NOP
        RETI                            ;64K 程序大小的中断使用 RETI 返回

TIMER0_ISR
        NOP
        NOP
        RETI

INT1_ISR
        NOP
        NOP
        RETI

TIMER1_ISR
        NOP
        NOP
        RETI

START:
        MOV     SPX,#0100H              ;设置堆栈指针初始值
        MOV     WTST,#0                 ;设置程序指令延时参数，
                                        ;赋值为 0 可将 CPU 执行指令的速度设置为最快
        MOV     CKCON,#0                ;提高访问 XRAM 速度
```

```
        MOV     P0M0,#0                 ;系统初始化
        MOV     P0M1,#0
MAINLOOP:
        INC     P0
        LCALL   DELAY                   ;64K 程序大小的函数使用 LCALL 或者 ACALL 调用
        JMP     MAINLOOP

DELAY:
        MOV     WR0,#5
DELAY1:
        DEC     WR0,#1
        JNE     DELAY1
        RET                             ;64K 程序大小的函数使用 RET 返回

        END
```

2.24.2 代码大小超过 64K 的汇编程序编写方法

```
P0      DATA    080H
P0M1    DATA    093H
P0M0    DATA    094H
WTST    DATA    0E9H
CKCON   DATA    0EAH

        ORG     0FF:0000H               ;复位入口地址
        JMP     START                   ; 1.  超 64K 程序大小的代码,
                                        ;     ORG 定址必须使用 0xx:xxxx 的方式
                                        ;2.   中断向量处可使用 JMP 语句,编译器会自动
                                        ;     根据实际编译情况智能替换成 AJMP/LJMP/EJMP

        ORG     0FF:0003H               ;中断入口地址
        JMP     INT0_ISR
        ORG     0FF:000BH
        JMP     TIMER0_ISR
        ORG     0FF:0013H
        JMP     INT1_ISR
        ORG     0FF:001BH
        JMP     TIMER1_ISR

        ORG     0FF:0200H

INT0_ISR:                               ;中断函数
        NOP
        NOP
        RETI                            ;中断返回

TIMER0_ISR:
        NOP
        NOP
        RETI

INT1_ISR:
        NOP
        NOP
```

```
        RETI

TIMER1_ISR:
        NOP
        NOP
        RETI

START:
        MOV     SPX,#0100H              ;设置堆栈指针初始值
        MOV     WTST,#0                 ;设置程序指令延时参数，
                                        ;赋值为 0 可将 CPU 执行指令的速度设置为最快
        MOV     CKCON,#0                ;提高访问 XRAM 速度

        MOV     P0M0,#0                 ;系统初始化
        MOV     P0M1,#0
MAINLOOP:
        INC     P0
        ECALL   DELAY                   ;超 64K 程序大小的函数使用 ECALL 调用
        JMP     MAINLOOP

        ORG     0FE:0000H

DELAY:
        MOV     WR0,#1000
DELAY1:
        DEC     WR0,#1
        JNE     DELAY1
        ERET                            ;超 64K 程序大小的函数使用 ERET 返回

        END
```

2.25　对变量、常量、表格数据、函数指定绝对地址，在 C251 中

2.25.1　变量如何指定绝对地址，Keil C251 中

语法如下：

数据类型 [存储类型] 变量名称 **_at_** 绝对地址;

在 data 区域指定绝对地址变量的范例：

```
int  data  var_data_abs  _at_  0x50;
```

在 edata 区域指定绝对地址变量的范例：

```
int  edata  var_edata_abs  _at_  0x200;
```

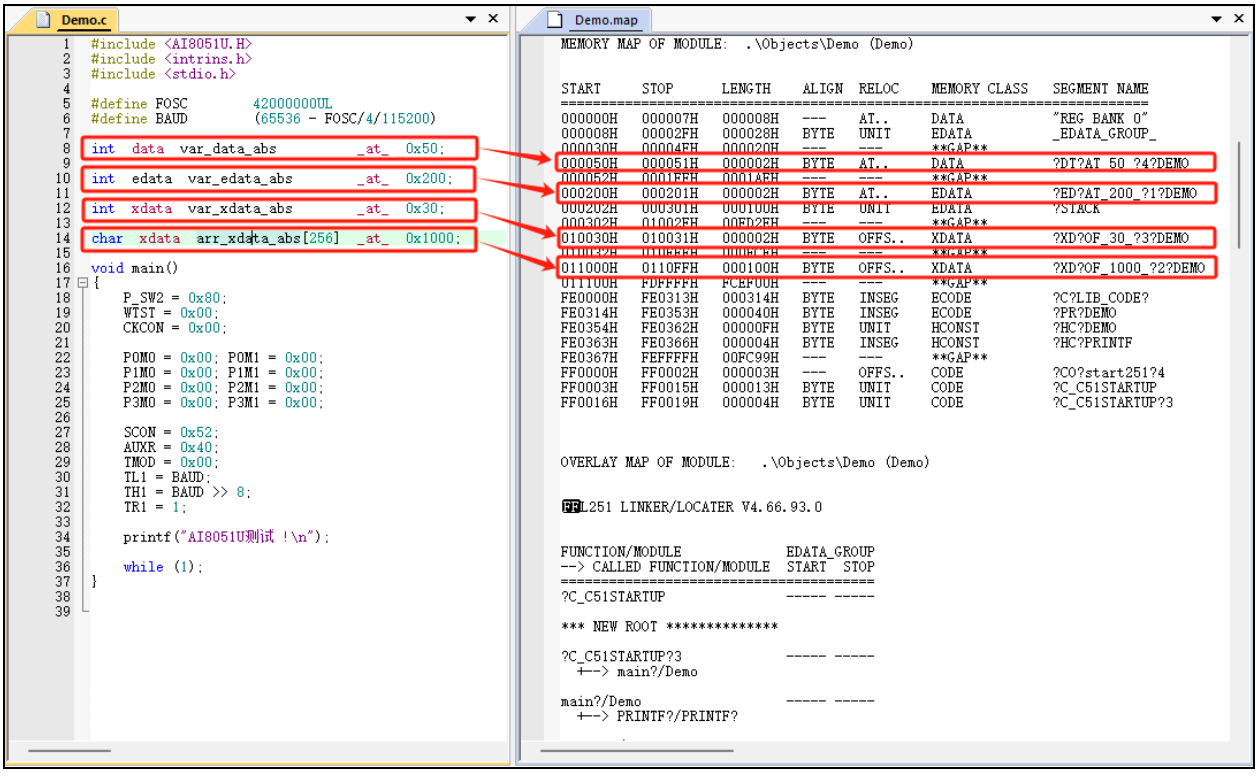
在 xdata 区域指定绝对地址变量的范例：

```
int  xdata  var_xdata_abs  _at_  0x30;
```

在 xdata 区域指定绝对地址数组的范例：

```
char  xdata  arr_xdata_abs[256]  _at_  0x1000;
```

编译完成后地址分配如下图：



Demo.c

```
#include <AI8051U.H>
#include <intrins.h>
#include <stdio.h>

#define FOSC        42000000UL
#define BAUD        (65536 - FOSC/4/115200)

int  data  var_data_abs          _at_  0x50;
int  edata  var_edata_abs        _at_  0x200;
int  xdata  var_xdata_abs        _at_  0x30;
char  xdata  arr_xdata_abs[256]  _at_  0x1000;

void main()
{
    P_SW2 = 0x80;
    WTST = 0x00;
    CKCON = 0x00;

    P0M0 = 0x00; P0M1 = 0x00;
    P1M0 = 0x00; P1M1 = 0x00;
    P2M0 = 0x00; P2M1 = 0x00;
    P3M0 = 0x00; P3M1 = 0x00;

    SCON = 0x52;
    AUXR = 0x40;
    TMOD = 0x00;
    TL1 = BAUD;
    TH1 = BAUD >> 8;
    TR1 = 1;

    printf("AI8051U测试 !\n");

    while (1);
}
```

Demo.map

```
MEMORY MAP OF MODULE:  .\Objects\Demo (Demo)


START     STOP      LENGTH    ALIGN  RELOC    MEMORY CLASS   SEGMENT NAME
=============================================================================
000000H   000007H   000008H   ---    AT..     DATA           "REG BANK 0"
000008H   00002FH   000028H   BYTE   UNIT     EDATA          _EDATA_GROUP_
000030H   00004FH   000020H   ---    ---      **GAP**
000050H   000051H   000002H   BYTE   AT..     DATA           ?DT?AT_50_?4?DEMO
000052H   0001FFH   0001ADH   ---    ---      **GAP**
000200H   000201H   000002H   BYTE   AT..     EDATA          ?ED?AT_200_?1?DEMO
000202H   000301H   000100H   BYTE   UNIT     EDATA          ?STACK
000302H   01002FH   00FD2EH   ---    ---      **GAP**
010030H   010031H   000002H   BYTE   OFFS..   XDATA          ?XD?OF_30_?3?DEMO
010032H   010FFFH   000FCEH   ---    ---      **GAP**
011000H   0110FFH   000100H   BYTE   OFFS..   XDATA          ?XD?OF_1000_?2?DEMO
011100H   FDFFFFH   FCEF00H   ---    ---      **GAP**
FE0000H   FE0313H   000314H   BYTE   INSEG    ECODE          ?C?LIB_CODE?
FE0314H   FE0353H   000040H   BYTE   INSEG    ECODE          ?PR?DEMO
FE0354H   FE0362H   00000FH   BYTE   UNIT     HCONST         ?HC?DEMO
FE0363H   FE0366H   000004H   BYTE   INSEG    HCONST         ?HC?PRINTF
FE0367H   FEFFFFH   00FC99H   ---    ---      **GAP**
FF0000H   FF0002H   000003H   ---    OFFS..   CODE           ?CO?start251?4
FF0003H   FF0015H   000013H   BYTE   UNIT     CODE           ?C_C51STARTUP
FF0016H   FF0019H   000004H   BYTE   UNIT     CODE           ?C_C51STARTUP?3


OVERLAY MAP OF MODULE:   .\Objects\Demo (Demo)


L251 LINKER/LOCATER V4.66.93.0


FUNCTION/MODULE             EDATA_GROUP
--> CALLED FUNCTION/MODULE  START  STOP
=========================================
?C_C51STARTUP               ------ ------

*** NEW ROOT **************

?C_C51STARTUP?3             ------ ------
  +--> main?/Demo

main?/Demo                  ------ ------
  +--> PRINTF?/PRINTF?
```

2.25.2 常量如何指定绝对地址，Keil C251 中

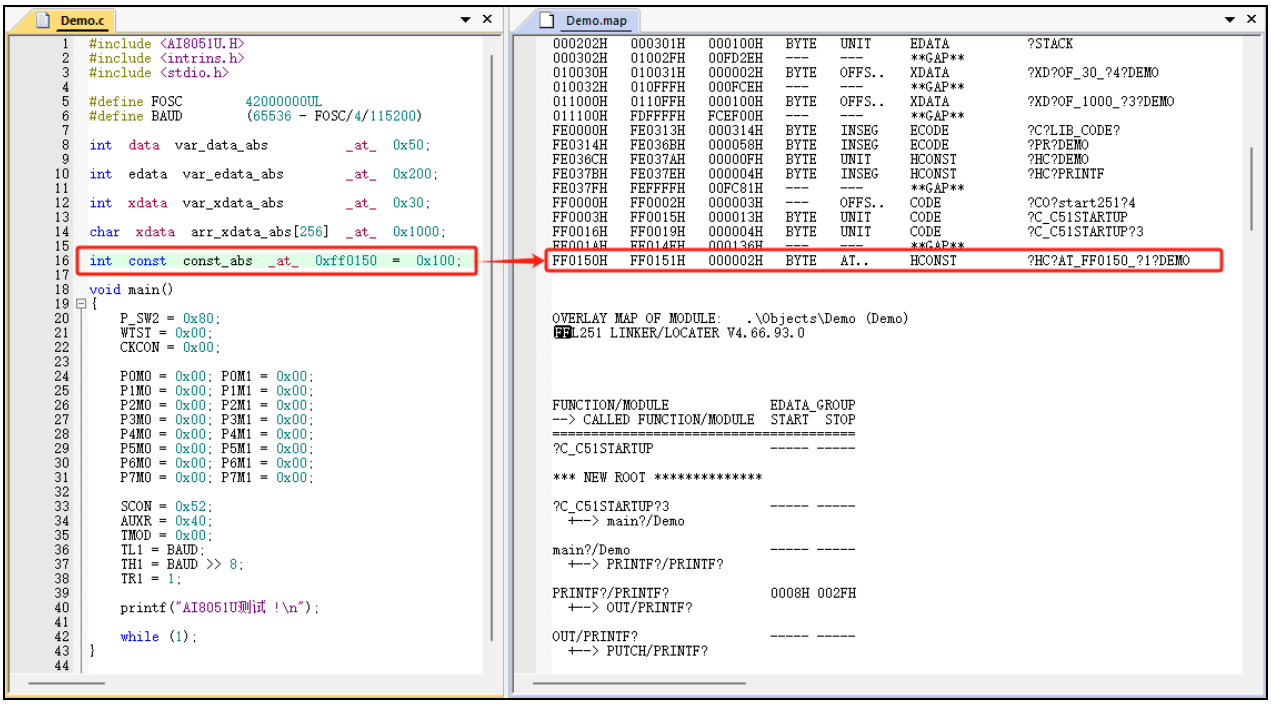
语法如下：

数据类型 **const** 变量名称 **_at_** 绝对地址;

在 code 区域指定绝对地址常量的范例：

```
int  const  const_abs  _at_  0xff0150  =  0x100;
```

编译完成后地址分配如下图：



Demo.c

```
#include <AI8051U.H>
#include <intrins.h>
#include <stdio.h>

#define FOSC         42000000UL
#define BAUD         (65536 - FOSC/4/115200)

int  data  var_data_abs          _at_  0x50;

int  edata  var_edata_abs        _at_  0x200;

int  xdata  var_xdata_abs        _at_  0x30;

char  xdata  arr_xdata_abs[256]  _at_  0x1000;

int  const  const_abs  _at_  0xff0150  =  0x100;

void main()
{
    P_SW2 = 0x80;
    WTST = 0x00;
    CKCON = 0x00;

    P0M0 = 0x00; P0M1 = 0x00;
    P1M0 = 0x00; P1M1 = 0x00;
    P2M0 = 0x00; P2M1 = 0x00;
    P3M0 = 0x00; P3M1 = 0x00;
    P4M0 = 0x00; P4M1 = 0x00;
    P5M0 = 0x00; P5M1 = 0x00;
    P6M0 = 0x00; P6M1 = 0x00;
    P7M0 = 0x00; P7M1 = 0x00;

    SCON = 0x52;
    AUXR = 0x40;
    TMOD = 0x00;
    TL1 = BAUD;
    TH1 = BAUD >> 8;
    TR1 = 1;

    printf("AI8051U测试 !\n");

    while (1);
}
```

Demo.map

```
000202H   000301H   000100H   BYTE   UNIT     EDATA          ?STACK
000302H   01002FH   00FD2EH   ---    ---      **GAP**
010030H   010031H   000002H   BYTE   OFFS..   XDATA          ?XD?OF_30_?4?DEMO
010032H   010FFFH   000FCEH   ---    ---      **GAP**
011000H   0110FFH   000100H   BYTE   OFFS..   XDATA          ?XD?OF_1000_?3?DEMO
011100H   FDFFFFH   FCEF00H   ---    ---      **GAP**
FE0000H   FE0313H   000314H   BYTE   INSEG    ECODE          ?C?LIB_CODE?
FE0314H   FE036BH   000058H   BYTE   INSEG    ECODE          ?PR?DEMO
FE036CH   FE037AH   00000FH   BYTE   UNIT     HCONST         ?HC?DEMO
FE037BH   FE037EH   000004H   BYTE   INSEG    HCONST         ?HC?PRINTF
FE037FH   FEFFFFH   00FC81H   ---    ---      **GAP**
FF0000H   FF0002H   000003H   ---    OFFS..   CODE           ?CO?start251?4
FF0003H   FF0015H   000013H   BYTE   UNIT     CODE           ?C_C51STARTUP
FF0016H   FF0019H   000004H   BYTE   UNIT     CODE           ?C_C51STARTUP?3
FF001AH   FF014FH   000136H   ---    ---      **GAP**
FF0150H   FF0151H   000002H   BYTE   AT..     HCONST         ?HC?AT_FF0150_?1?DEMO


OVERLAY MAP OF MODULE:   .\Objects\Demo (Demo)
FPL251 LINKER/LOCATER V4.66.93.0



FUNCTION/MODULE             EDATA_GROUP
--> CALLED FUNCTION/MODULE  START  STOP
=======================================
?C_C51STARTUP               ----- -----

*** NEW ROOT **************

?C_C51STARTUP?3             ----- -----
  +--> main?/Demo

main?/Demo                  ----- -----
  +--> PRINTF?/PRINTF?

PRINTF?/PRINTF?             0008H 002FH
  +--> OUT/PRINTF?

OUT/PRINTF?                 ----- -----
  +--> PUTCH/PRINTF?
```

2.25.3 表格数据如何指定绝对地址，Keil C251 中

语法如下：

数据类型 **code** 变量名称 **_at_** 绝对地址 = { 表格数据 };

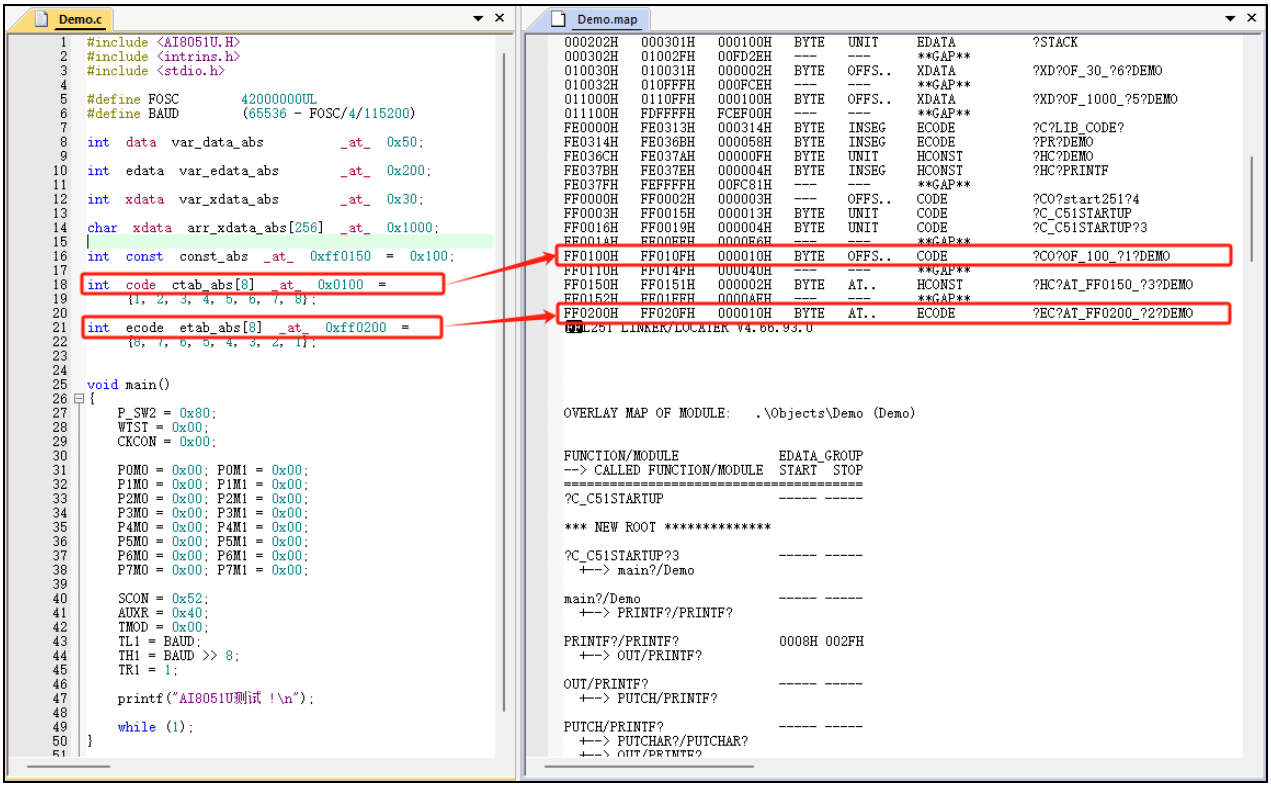
或者

数据类型 **ecode** 变量名称 **_at_** 绝对地址 = { 表格数据 };

注：使用 code 储存类型时，地址范围为 FF:0000H~FF:FFFFH，需要用 0000H~FFFFH 的 16 位地址指定的绝对地址；使用 ecode 储存类型时，地址范围为 80:0000H~FF:FFFFH，需要用 800000H~FFFFFFH 的 24 位地址指定的绝对地址

在 code 区域指定绝对地址表格的范例：

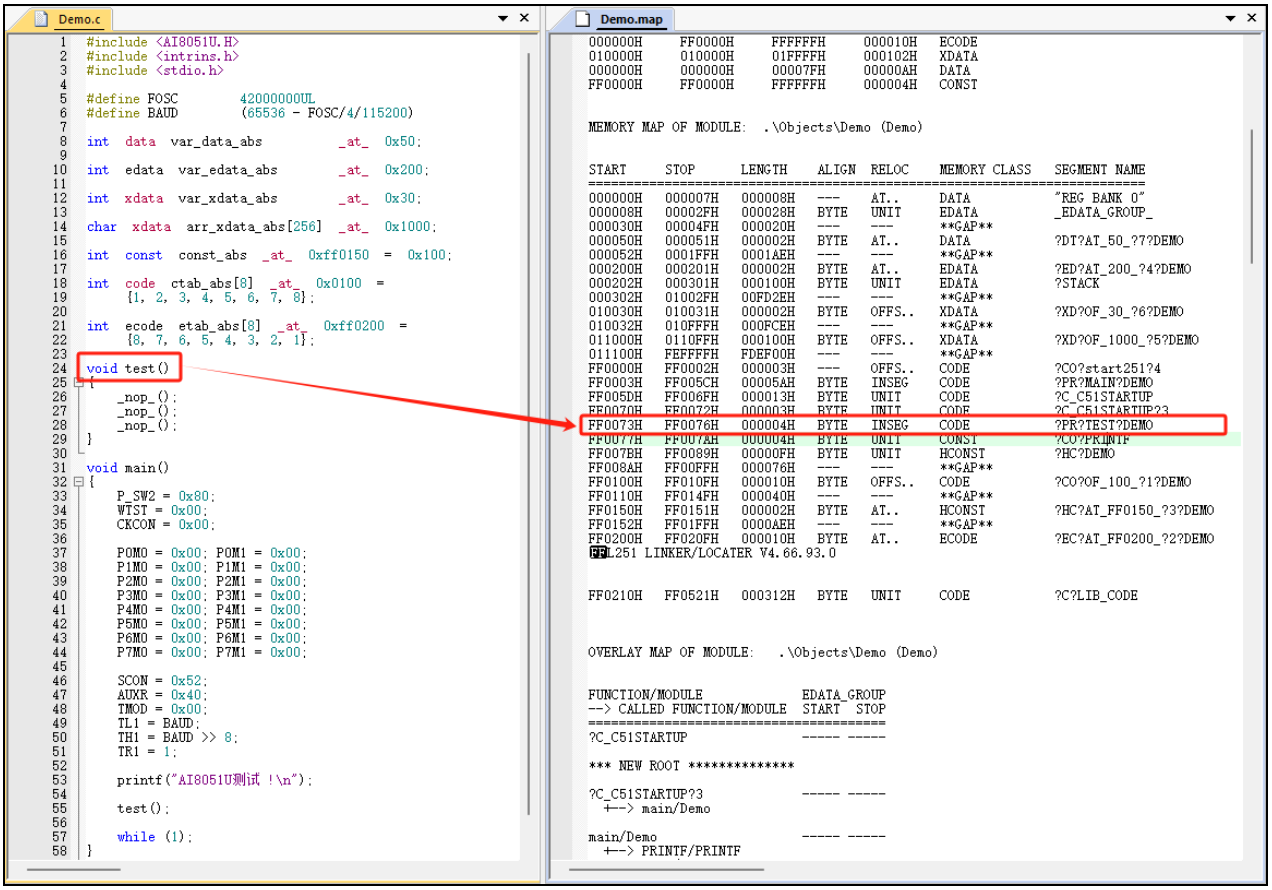
```
int   code   ctab_abs[8]   _at_   0x0100   =   {1, 2, 3, 4, 5, 6, 7, 8};
int   ecode   etab_abs[8]   _at_   0xff0200   =   {8, 7, 6, 5, 4, 3, 2, 1};
```



2.25.4 函数如何指定绝对地址，Keil C251 中

C251 中无法直接在程序中指定函数的绝对地址，需要通过如下方法实现

首先在程序编写完成函数代码，编译成功后，获取函数的链接符号（如下图为“?PR?TEST?DEMO”）



Demo.c

```
#include <AI8051U.H>
#include <intrins.h>
#include <stdio.h>

#define FOSC        42000000UL
#define BAUD        (65536 - FOSC/4/115200)

int  data  var_data_abs          _at_  0x50;

int  edata  var_edata_abs        _at_  0x200;

int  xdata  var_xdata_abs        _at_  0x30;

char  xdata  arr_xdata_abs[256]  _at_  0x1000;

int  const  const_abs  _at_  0xff0150  =  0x100;

int  code  ctab_abs[8]  _at_  0x0100  =
     {1, 2, 3, 4, 5, 6, 7, 8};

int  ecode  etab_abs[8]  _at_  0xff0200  =
     {8, 7, 6, 5, 4, 3, 2, 1};

void test()
{
    _nop_();
    _nop_();
    _nop_();
}

void main()
{
    P_SW2 = 0x80;
    WTST = 0x00;
    CKCON = 0x00;

    P0M0 = 0x00; P0M1 = 0x00;
    P1M0 = 0x00; P1M1 = 0x00;
    P2M0 = 0x00; P2M1 = 0x00;
    P3M0 = 0x00; P3M1 = 0x00;
    P4M0 = 0x00; P4M1 = 0x00;
    P5M0 = 0x00; P5M1 = 0x00;
    P6M0 = 0x00; P6M1 = 0x00;
    P7M0 = 0x00; P7M1 = 0x00;

    SCON = 0x52;
    AUXR = 0x40;
    TMOD = 0x00;
    TL1 = BAUD;
    TH1 = BAUD >> 8;
    TR1 = 1;

    printf("AI8051U测试 !\n");

    test();

    while (1);
}
```

Demo.map

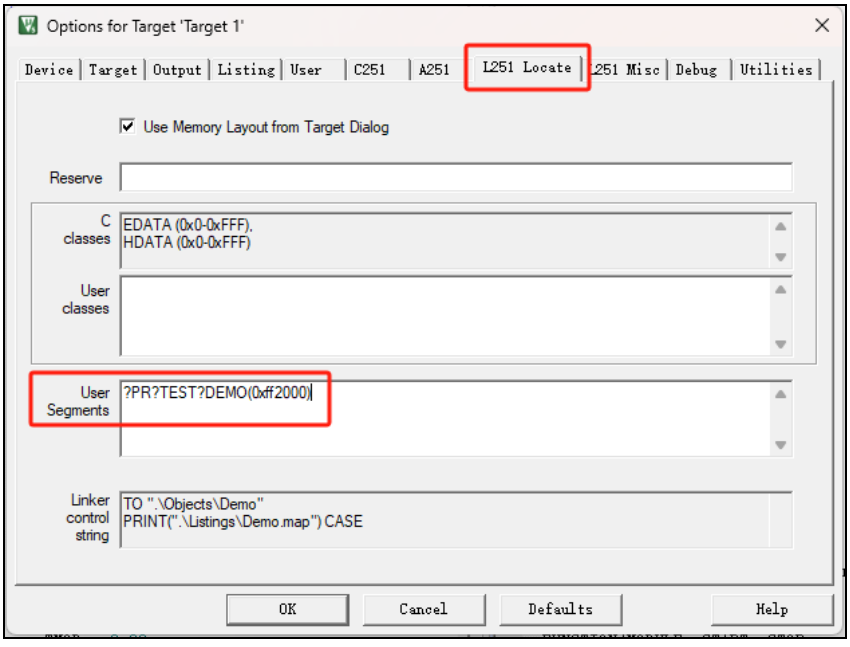
```
000000H    FF0000H    FFFFFFH    000010H   ECODE
010000H    010000H    01FFFFH    000102H   XDATA
000000H    000000H    00007FH    00000AH   DATA
FF0000H    FF0000H    FFFFFFH    000004H   CONST


MEMORY MAP OF MODULE:  .\Objects\Demo (Demo)


START     STOP      LENGTH    ALIGN  RELOC    MEMORY CLASS   SEGMENT NAME
=============================================================================
000000H   000007H   000008H   ---    AT..     DATA           "REG BANK 0"
000008H   00002FH   000028H   BYTE   UNIT     EDATA          _EDATA_GROUP_
000030H   00004FH   000020H   ---    ---      **GAP**
000050H   000051H   000002H   BYTE   AT..     DATA           ?DT?AT_50_??DEMO
000052H   0001FFH   0001AEH   ---    ---      **GAP**
000200H   000201H   000002H   BYTE   AT..     EDATA          ?ED?AT_200_?4?DEMO
000202H   000301H   000100H   BYTE   UNIT     EDATA          ?STACK
000302H   01002FH   00FD2EH   ---    ---      **GAP**
010030H   010031H   000002H   BYTE   OFFS..   XDATA          ?XD?OF_30_?6?DEMO
010032H   010FFFH   000FCEH   ---    ---      **GAP**
011000H   0110FFH   000100H   BYTE   OFFS..   XDATA          ?XD?OF_1000_?5?DEMO
011100H   FEFFFFH   FDEF00H   ---    ---      **GAP**
FF0000H   FF0002H   000003H   ---    OFFS..   CODE           ?CO?start251?4
FF0003H   FF005CH   00005AH   BYTE   INSEG    CODE           ?PR?MAIN?DEMO
FF005DH   FF006FH   000013H   BYTE   UNIT     CODE           ?C_C51STARTUP
FF0070H   FF0072H   000003H   BYTE   UNIT     CODE           ?C_C51STARTUP23
FF0073H   FF0076H   000004H   BYTE   INSEG    CODE           ?PR?TEST?DEMO
FF0077H   FF0087H   000004H   BYTE   UNIT     CONST          ?CO?PRINTF
FF007BH   FF0089H   00000FH   BYTE   UNIT     HCONST         ?HC?DEMO
FF008AH   FF00FFH   000076H   ---    ---      **GAP**
FF0100H   FF010FH   000010H   BYTE   OFFS..   CODE           ?CO?OF_100_?1?DEMO
FF0110H   FF014FH   000040H   ---    ---      **GAP**
FF0150H   FF0151H   000002H   BYTE   AT..     HCONST         ?HC?AT_FF0150_?3?DEMO
FF0152H   FF01FFH   0000AEH   ---    ---      **GAP**
FF0200H   FF020FH   000010H   BYTE   AT..     ECODE          ?EC?AT_FF0200_?2?DEMO
FF
L251 LINKER/LOCATER V4.66.93.0


FF0210H   FF0521H   000312H   BYTE   UNIT     CODE           ?C?LIB_CODE


OVERLAY MAP OF MODULE:   .\Objects\Demo (Demo)


FUNCTION/MODULE             EDATA_GROUP
--> CALLED FUNCTION/MODULE  START  STOP
=========================================
?C_C51STARTUP               -----  -----

*** NEW ROOT **************

?C_C51STARTUP?3             -----  -----
  +--> main/Demo

main/Demo                   -----  -----
  +--> PRINTF/PRINTF
```

接下来在项目设置项中打开“L251 Locate”设置页面

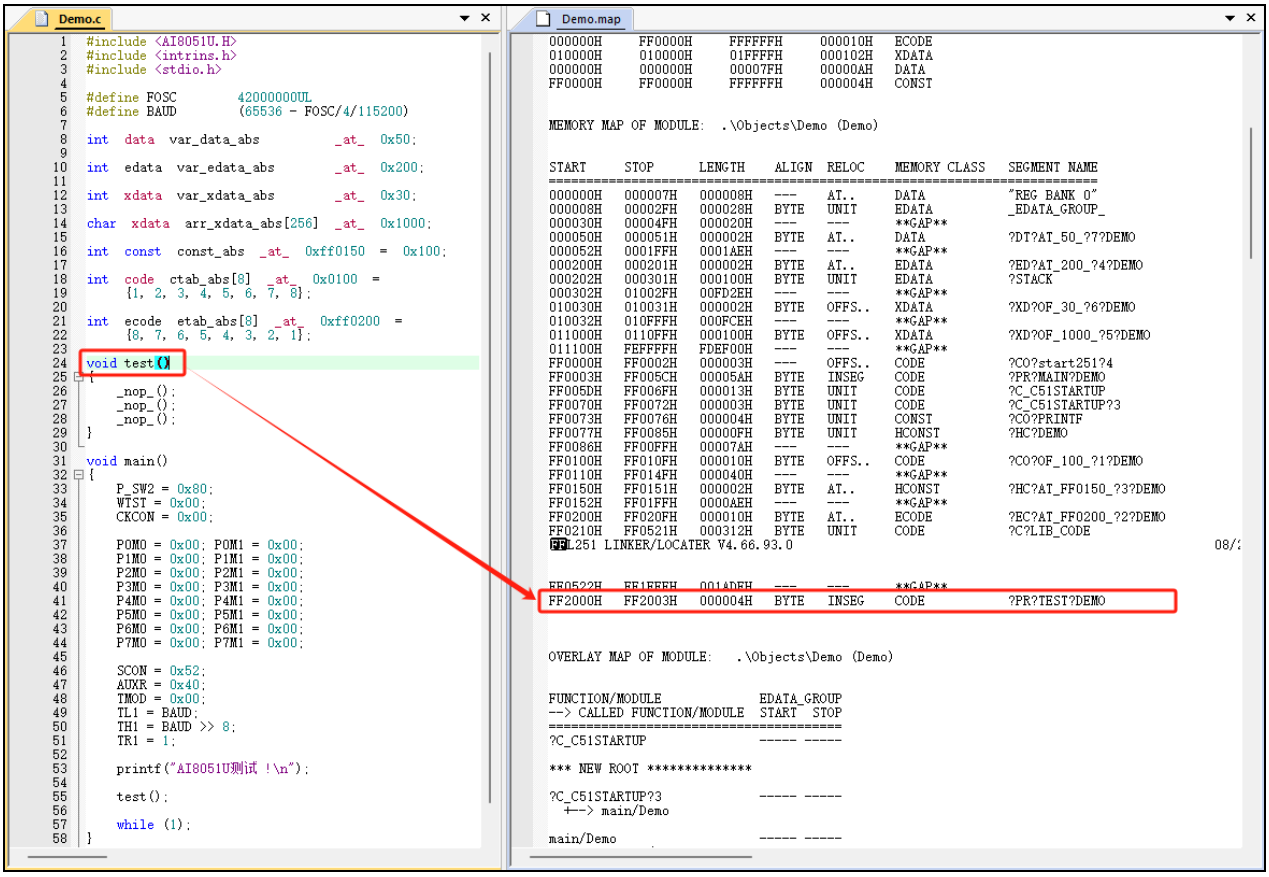
在“User Segments”一栏中按照：链接名称 (链接地址)　的格式，输入绝对地址

注：链接地址需要使用 24 位地址指定

如下图，将函数指定到代码区的 0xff2000 的绝对地址

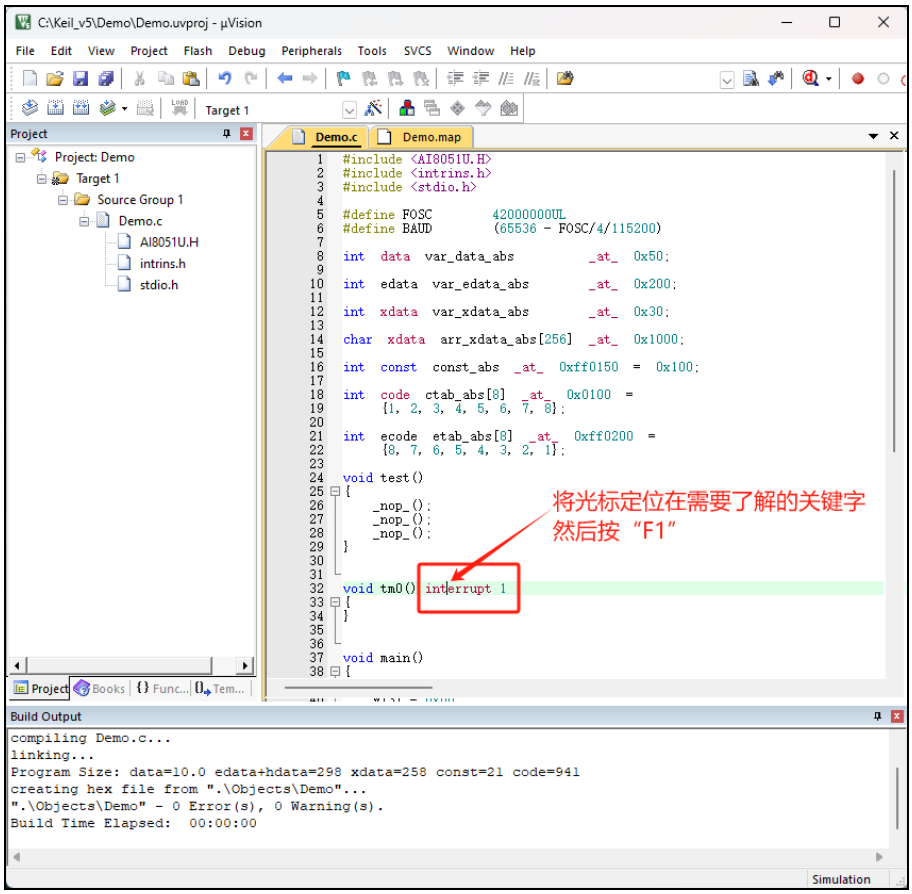


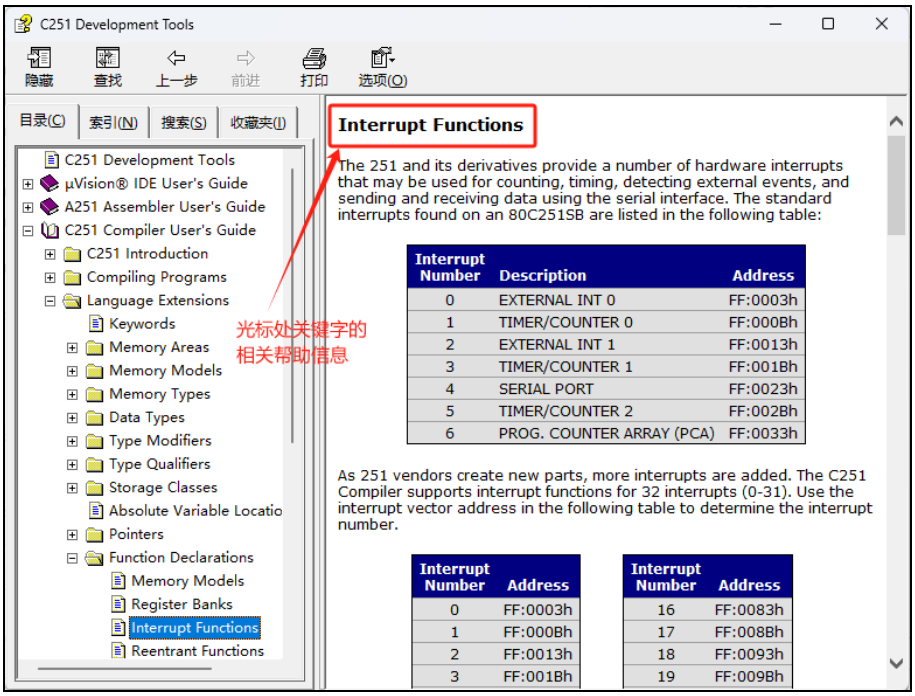
设置完成后，再次编译，函数即可被链接到指定的绝对地址，如下图：



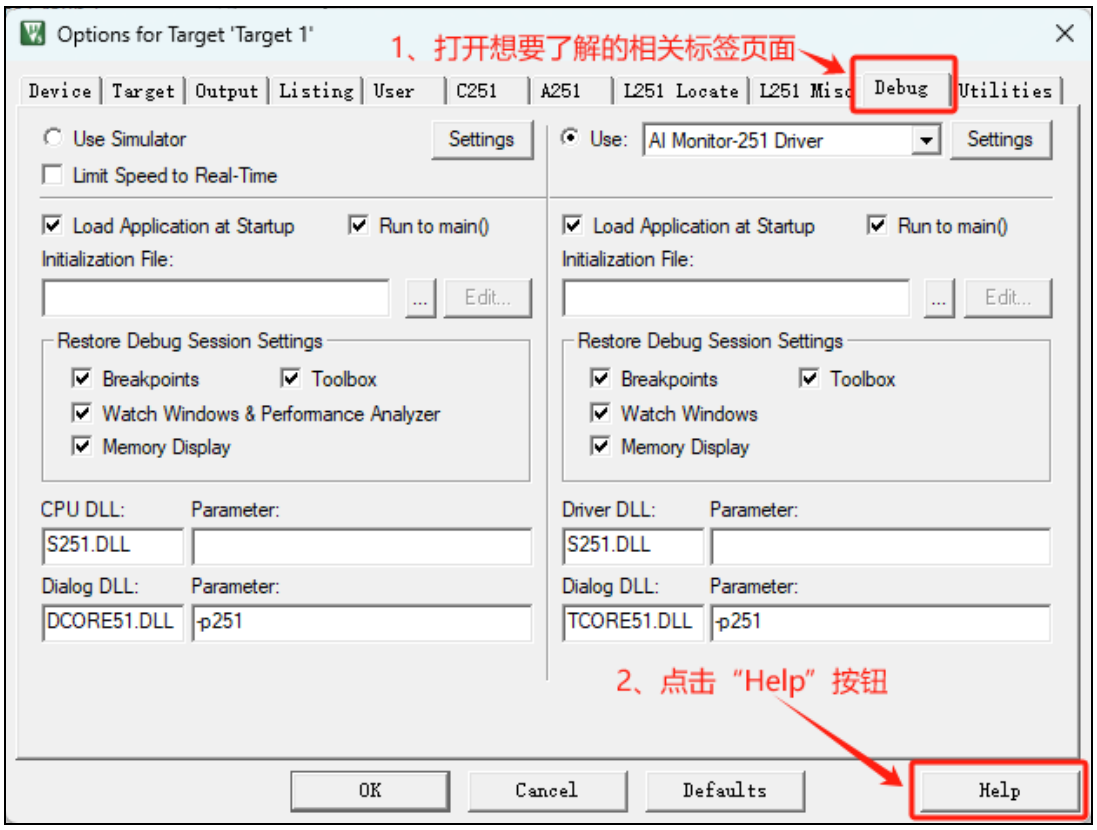
2.26　获取帮助的简单方法，Keil 软件中

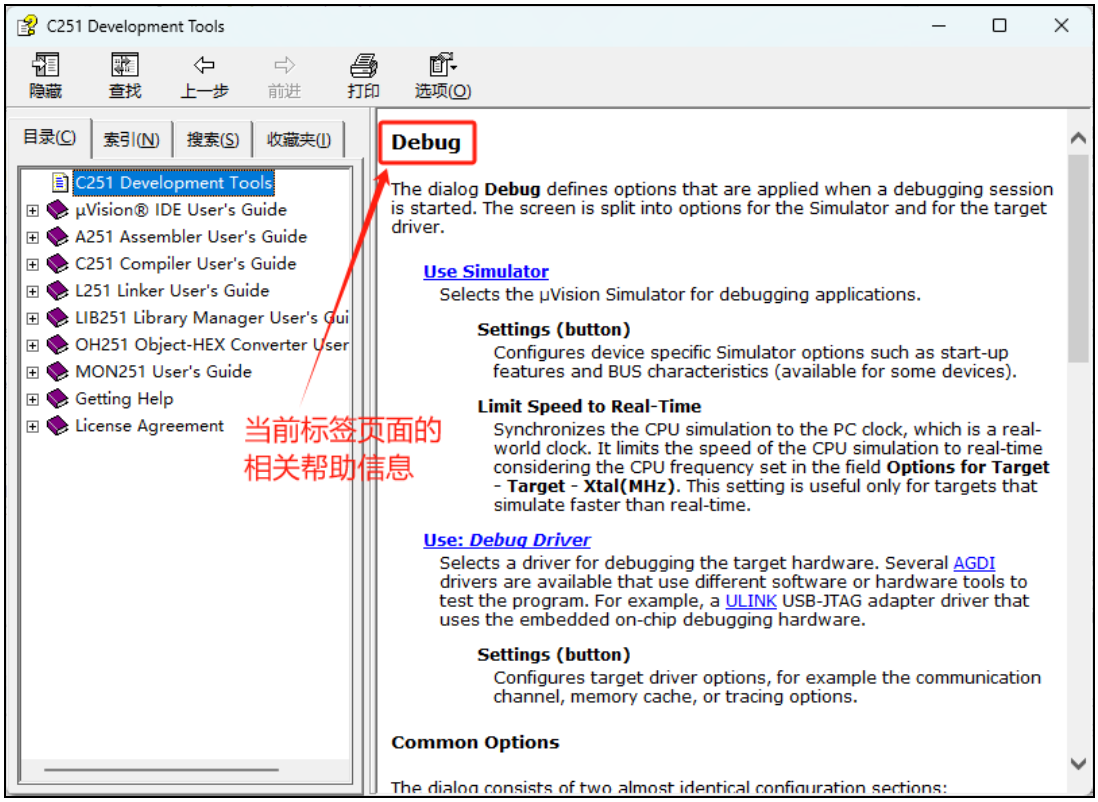
Keil 软件提供了很完整的帮助文件，对于一般的软件使用和编程问题，直接使用 Keil 软件的帮助基本都可以得到解决。如下图：



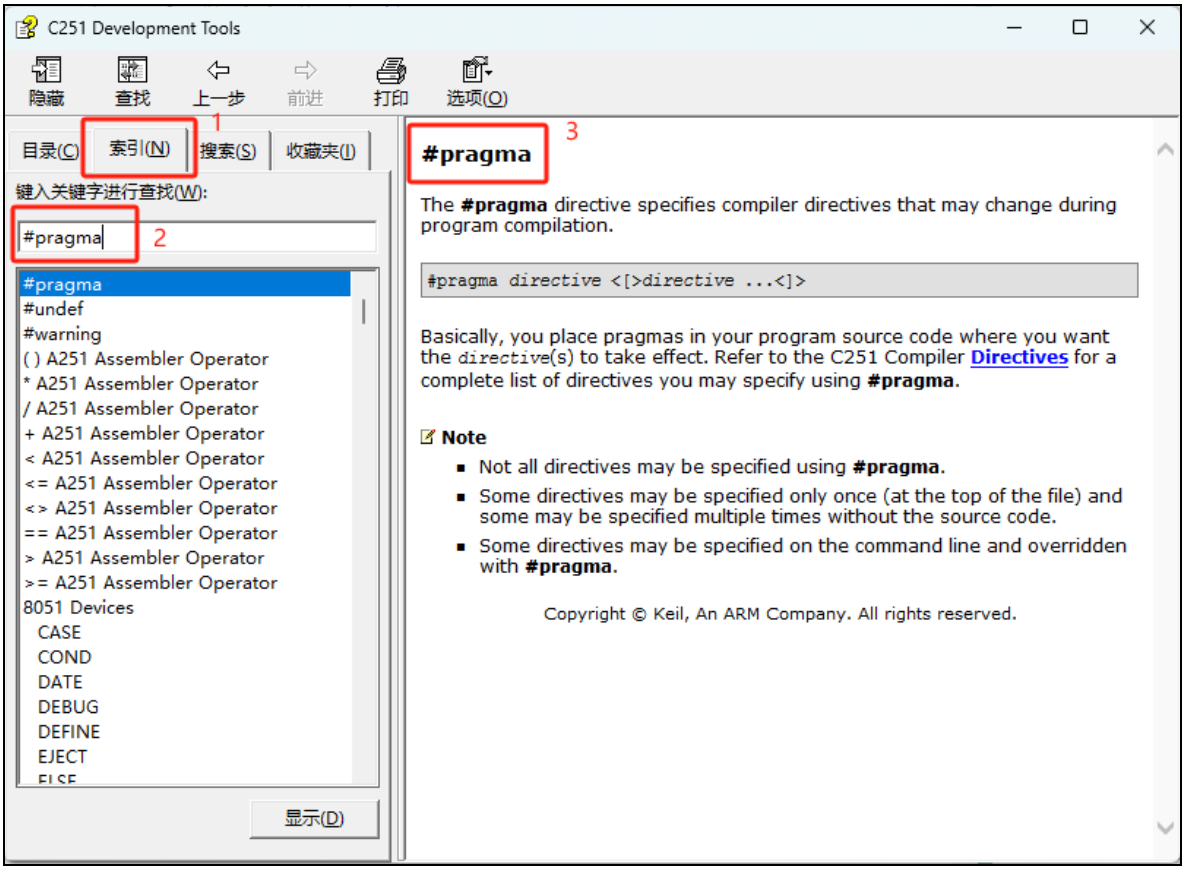


若需要了解项目设置中的相关设置的，可按下图所示的方法获取帮助





另外，也可在帮助窗口中直接输入想了解的内如。比如需要了解如何在程序中设置特殊的编译指示，可按下图所示，在搜索框输入“#pragma”即可

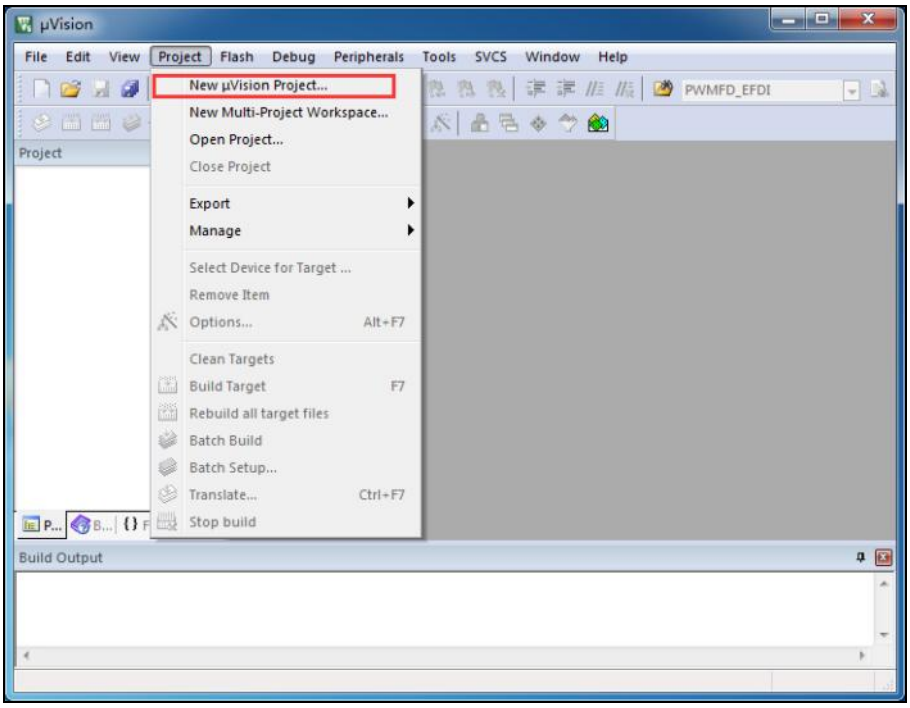


如果需要更详细的帮助详细，可登录 Keil 官网进行查询

2.27　建立多文件项目的方法，在 Keil 中

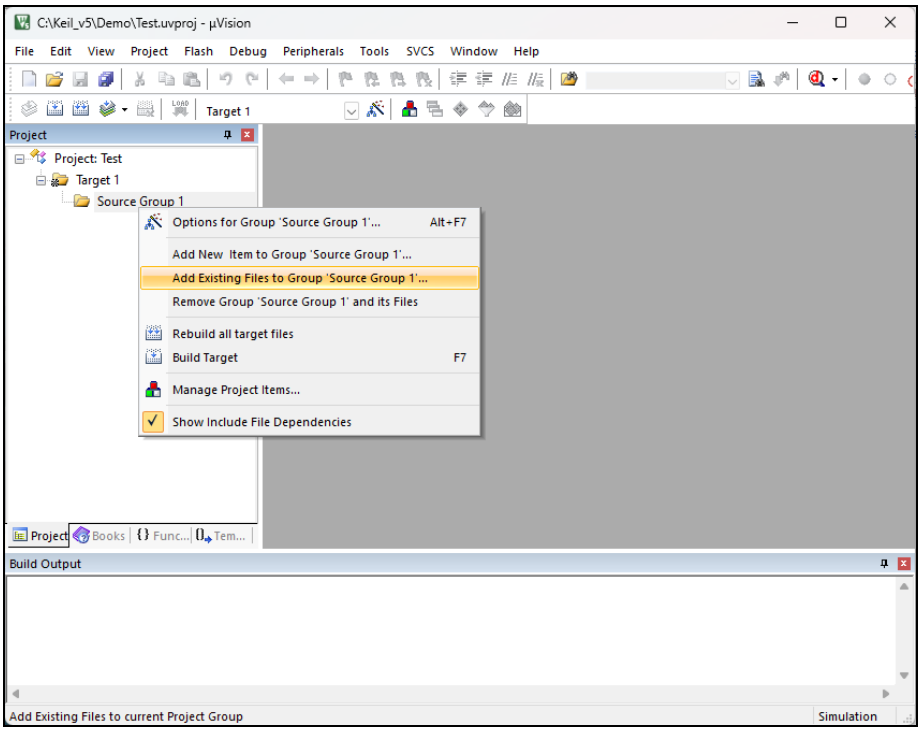
在 Keil 中，一般比较小的项目都只有一个源文件，但对于一些稍微复杂的项目往往需要多个源文件
建立多文件项目的方法如下：

1、首先打开 Keil，在菜单“Project”中选择“New uVision Project ...”

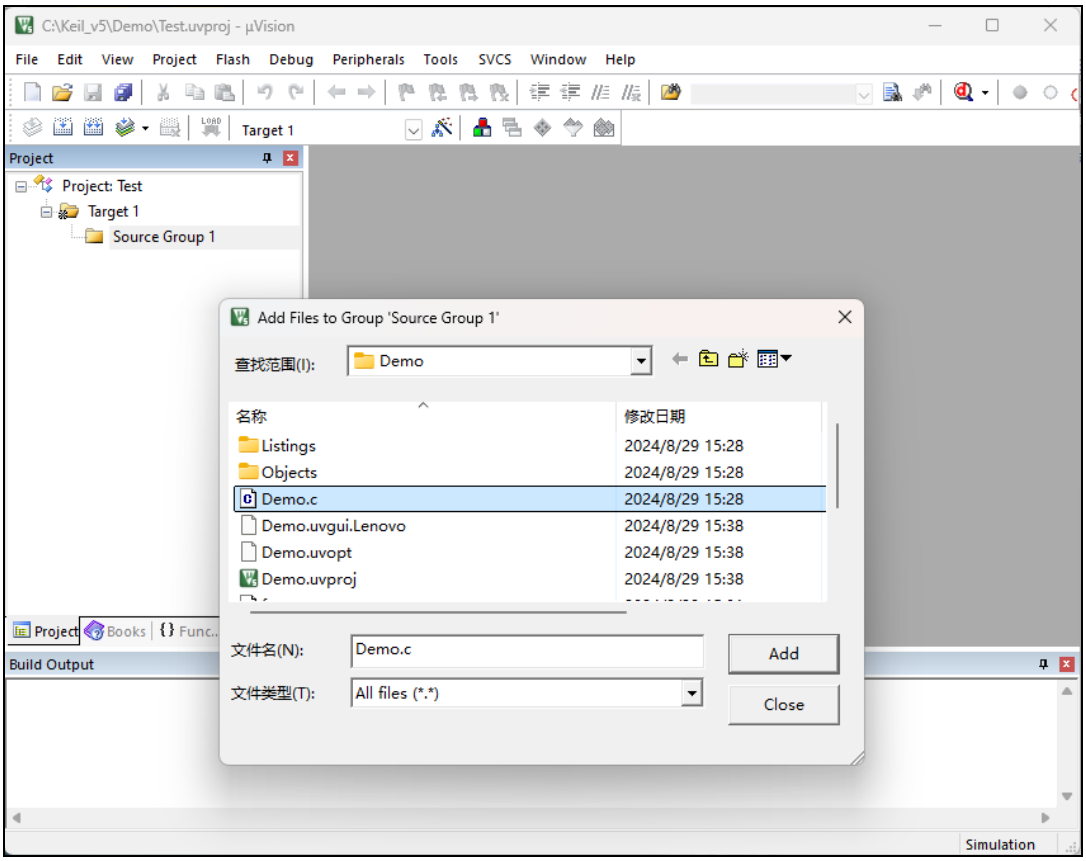


即可完成一个空项目的建立

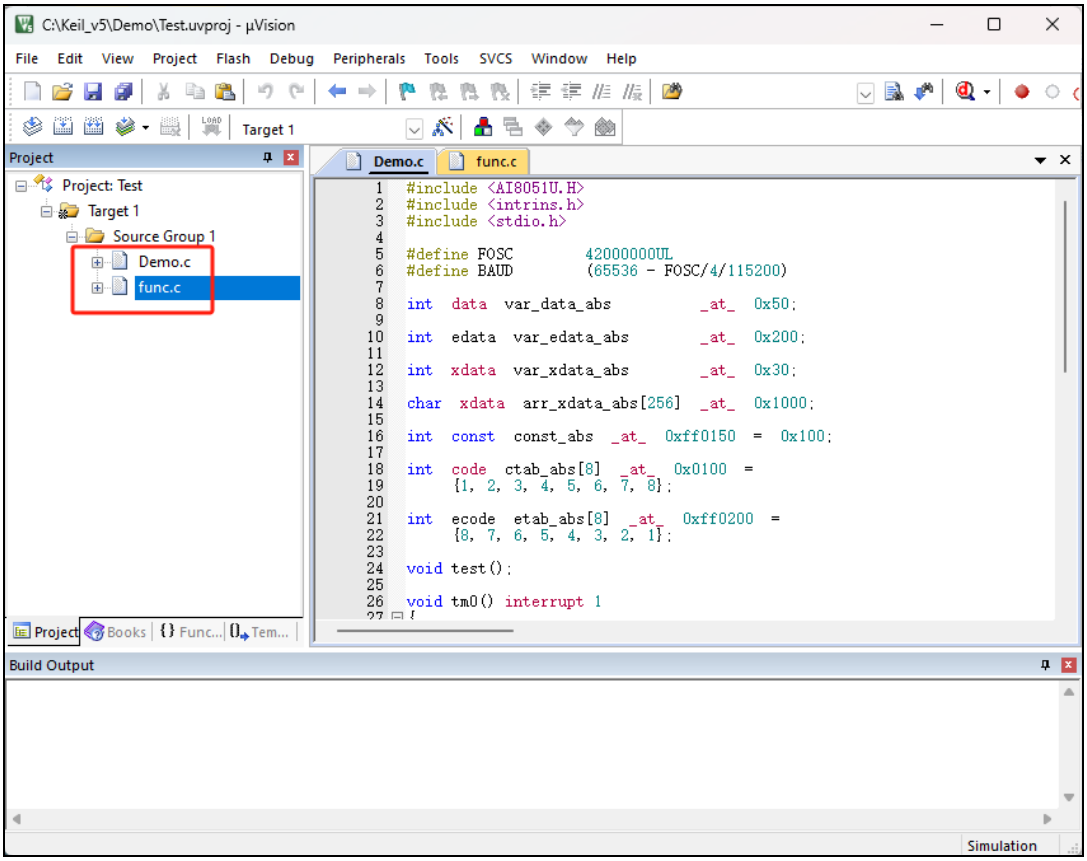
2、在空项目的项目树中，鼠标右键单击“Source Group 1”，并选择右键菜单中的“Add Existing Files to Group "Source Group 1" ...”



3、在弹出的文件对话框中，多次添加源文件



如下图所示即可完成多文件项目的建立



2.28 外部变量和外部函数的声明和使用，在多文件项目中

在 Keil 环境下开发比较大的项目时，往往需要在项目中加入多个源文件。而不同源文件内，可能会有一些变量和函数是开放为其它源文件使用的。但默认情况下，在源文件内部定义的变量和函数只在本源文件可见(可使用)，如果需要跨源文件使用则需要遵循一定的规则进行定义和声明。

对于**需要跨源文件访问变量**：

1、变量必须定义为全局变量
所谓全局变量就是在函数外部定义，在函数内部定义的变量为局部变量
由于局部变量的作用域仅在定义的函数内部，且生命周期在函数返回时就结束了
所以**要使变量能跨源文件访问，变量必须定义为全局变量**

2、变量必须定义为非静态变量
使用 static 关键字定义的变量为静态变量，静态变量作用域仅为本源文件，外部源文件不可访问。
所以要使变量能跨源文件访问，变量必须定义为非静态变量，即定义时不要使用 static 关键字

```
13
14  //定义全局变量
15  /*****************************************************
16  非静态的全局变量可以使用extern声明后跨源文件使用
17  不同的源文件不可以定义相同名称的非静态全局变量
18  *****************************************************/
19  int g_iVar1;
20
```

3、不同的源文件中不可定义名称相同的全局变量
如果在不同源文件中定义名称相同的全局变量，编译器会如下错误提示

```
linking...
*** ERROR L104: MULTIPLE PUBLIC DEFINITIONS
```

4、在需要使用外部变量的源文件内使用 extern 声明变量

```
06
07  //使用extern声明全局变量,使变量可跨文件访问
08  extern int g_iVar1;
09
```

如果需要在多个源文件中使用同一个外部变量，可将上面的外部声明语句
放在.h 文件中，在需要使用这个变量的源文件中 include 一下这个头文件即可

同样的，若需要**跨源文件调用外部函数**：

1、函数必须定义为非静态函数
使用 static 关键字定义的函数为静态函数，静态函数作用域仅为本源文件，外部源文件不可访问。
所以要使函数能跨源文件访问，必须定义为非静态函数，即定义函数时不要使用 static 关键字。

```
32
33  //定义非静态函数
34  /*************************************************
35  非静态函数可以使用extern声明后跨源文件使用
36  不同的源文件不可以定义相同名称的非静态函数
37  *************************************************/
38  int func1(int var)
39  {
40      check();
41      return (var * factor);
42  }
43
```

2、不同的源文件中不可定义名称相同的非静态函数

如果在不同源文件中定义名称相同的非静态函数时，编译器会如下错误提示

```
linking...
*** ERROR L104: MULTIPLE PUBLIC DEFINITIONS
```

3、在需要调用外部函数的源文件内直接声明一下即可

对于外部函数声明，是否使用 extern 关键字，编译时都可通过

```
03
04  //声明外部函数,使函数可跨文件调用
05  int func1(int var);
06
```

也可以 extern　　int　func1(int var);

如果需要在多个源文件中使用同一个外部函数，可将上面的外部声明语句放在.h 文件中，在需要调用这个函数的源文件中 include 一下这个头文件即可

2.29 变量的声明要求在可执行语句之前，C89 标准，KEIL C 语言

KEIL 支持的 C89 标准 C 语言，变量的声明必须遵循 C89 的严格作用域规则。

全局变量的声明需要在函数之前声明。

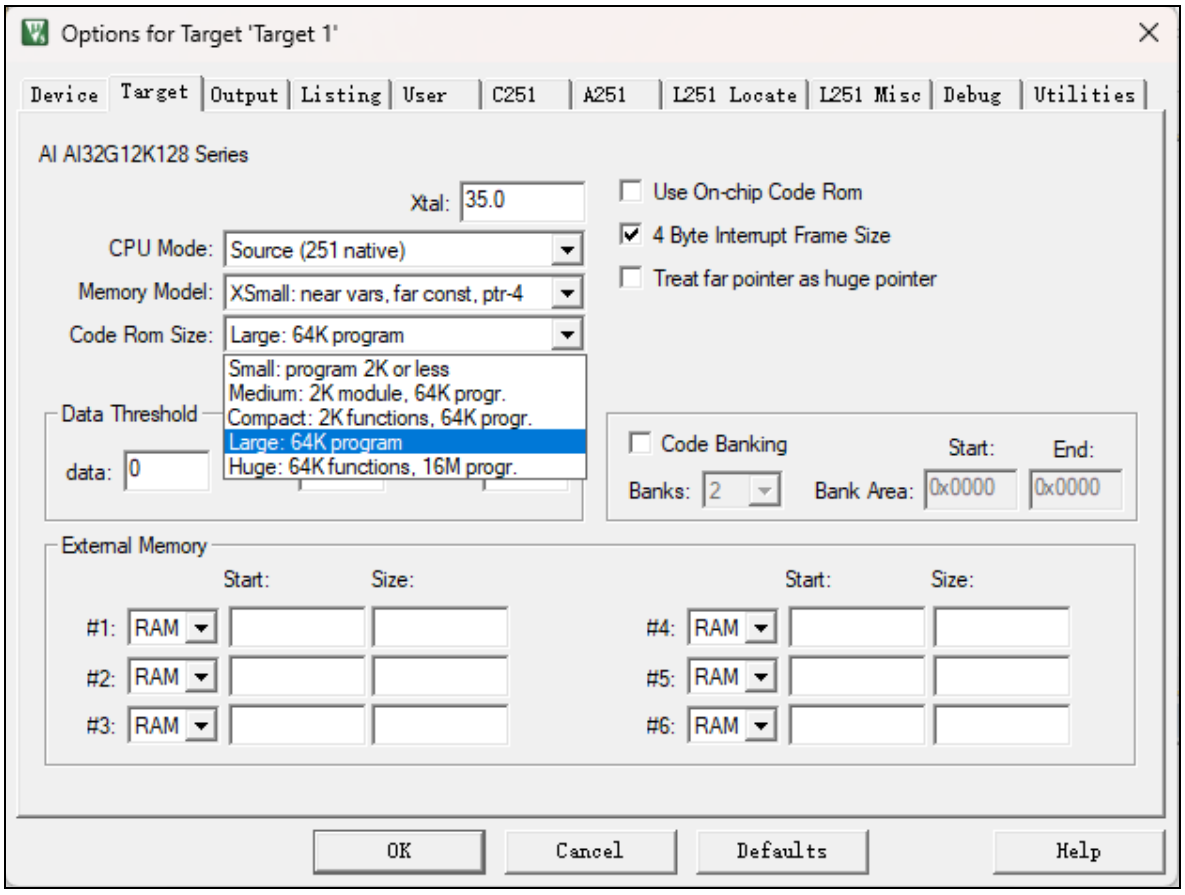
局部变量的声明必须集中在 {} 代码块的起始位置，不能与可执行语句混合，必须在代码块开头（任何可执行语句之前）声明。

循环变量：for 循环的计数器必须在循环外声明（C99 才支持 for(int i=0;...) 语法）。

中间声明：禁止在代码中间声明变量（如 if 语句之间或函数调用之后）。

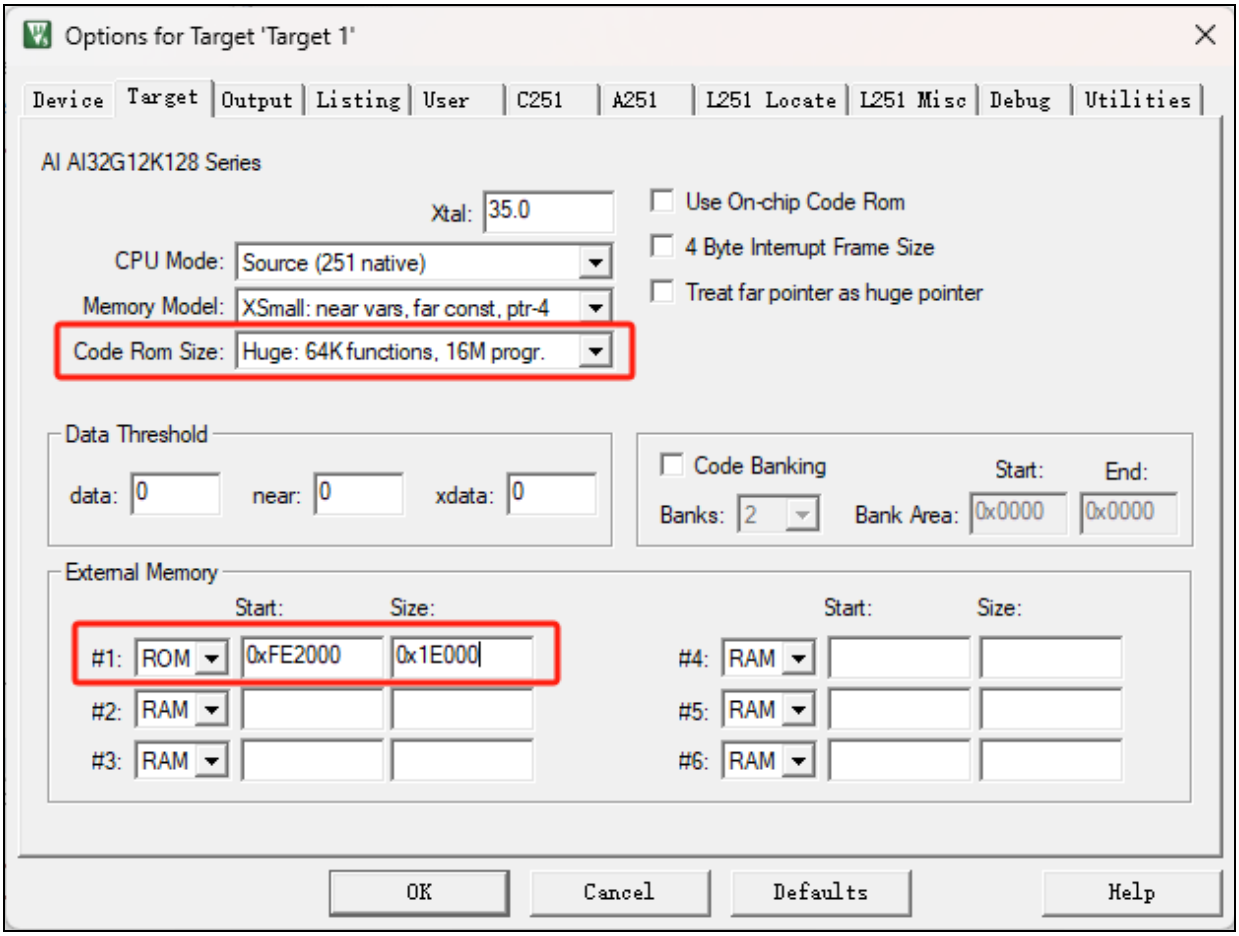
2.30 程序超 64K 时如何设置保留 EEPROM 空间，暂时用不到

若用户代码大小在 64K 以内，则可设置“Code Rom Size”为“Large”模式，Keil 编译器在链接代码块时会自动将代码全部链接在 FF:0000~FF:FFFF 的地址范围内。FE:0000~FE:FFFF 的 64K 可根据用户的 EEPROM 大小设置，任意使用

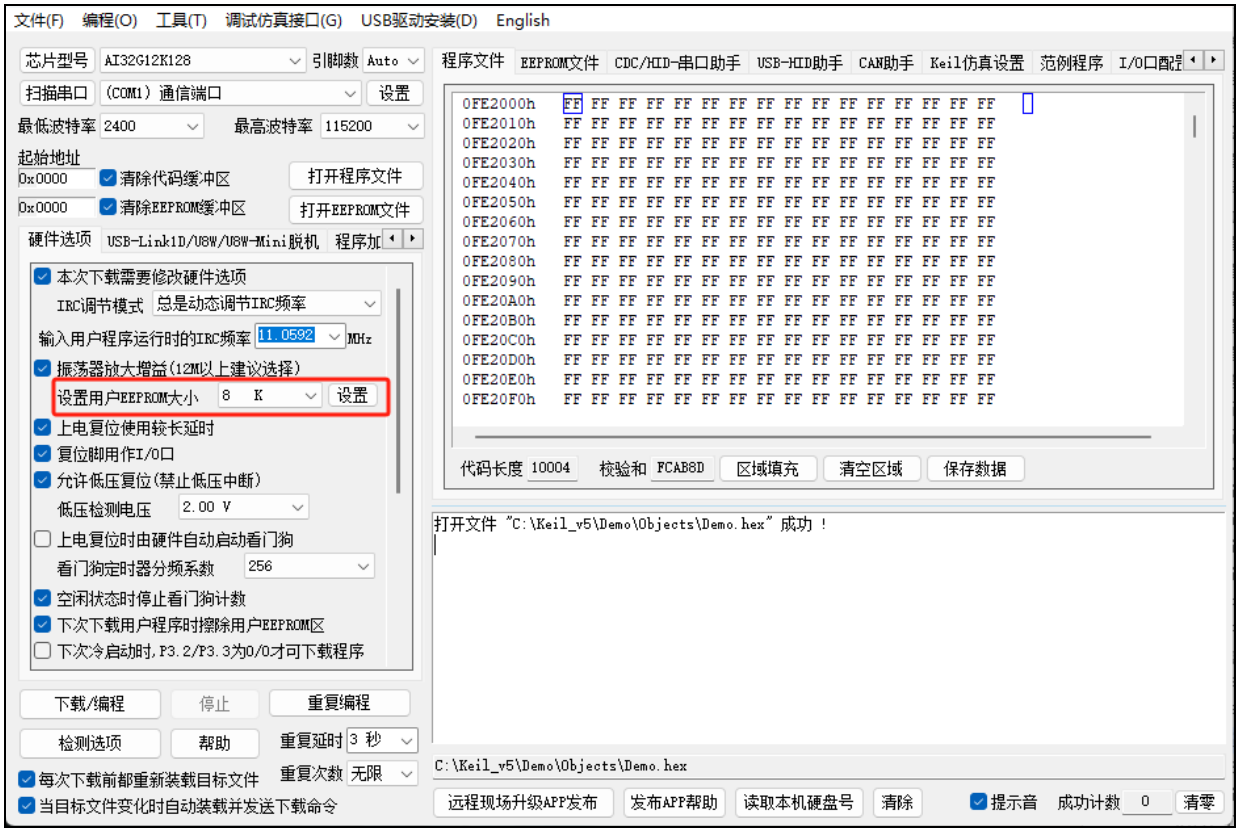


若用户代码大小超过 64K，则“Code Rom Size”必须设置为“Huge”模式，此时 Keil 编译器在链接代码块时，首先会将复位代码和中断向量代码链接在 FF:0000 开始的地址，其他代码块则会自动从用户设置的地址范例开始存放。由于 AI32G12K128 的 EEPROM 在 FLASH 中的地址固定为 FE:0000，所以为了让编译器不要将用户代码放在 EEPROM 区，则必须进行如下的相应设置：

比如用户需要 EEPROM 的大小为 8K，即 FLASH 地址的 FE:0000~FE:1FFF 区域为 EEPROM 区域，FLASH 地址的 FE:2000~FF:FFFF 区域为用户代码区。则在 Keil 中需要进行如下设置：



在 ISP 下载软件中需要进行如下设置：



2.31　用户程序复位到系统区进行 USB 模式 ISP 下载的方法（不停电）

当项目处于开发阶段时，需要反复的下载用户代码到目标芯片中进行代码验证，使用 USB 模式对单片机进行正常的 ISP 下载，需要先将 P3.2 口短路到 GND，然后对目标芯片进行重新上电，从而会使得项目在开发阶段烧录步骤比较繁琐。为此单片机增加了一个特殊功能寄存器 IAP_CONTR，当用户向此寄存器写入 0x60，即可实现软件复位到系统区，进而实现不停电就可进行 ISP 下载。

注：当用户程序软复位到系统区时，若 P3.0/D-和 P3.1/D+已经和电脑的 USB 口相连，则系统代码会自动进入 USB 下载模式等待 ISP 下载，此时不需要 P3.2 连接到地

下面介绍如下两种方法：

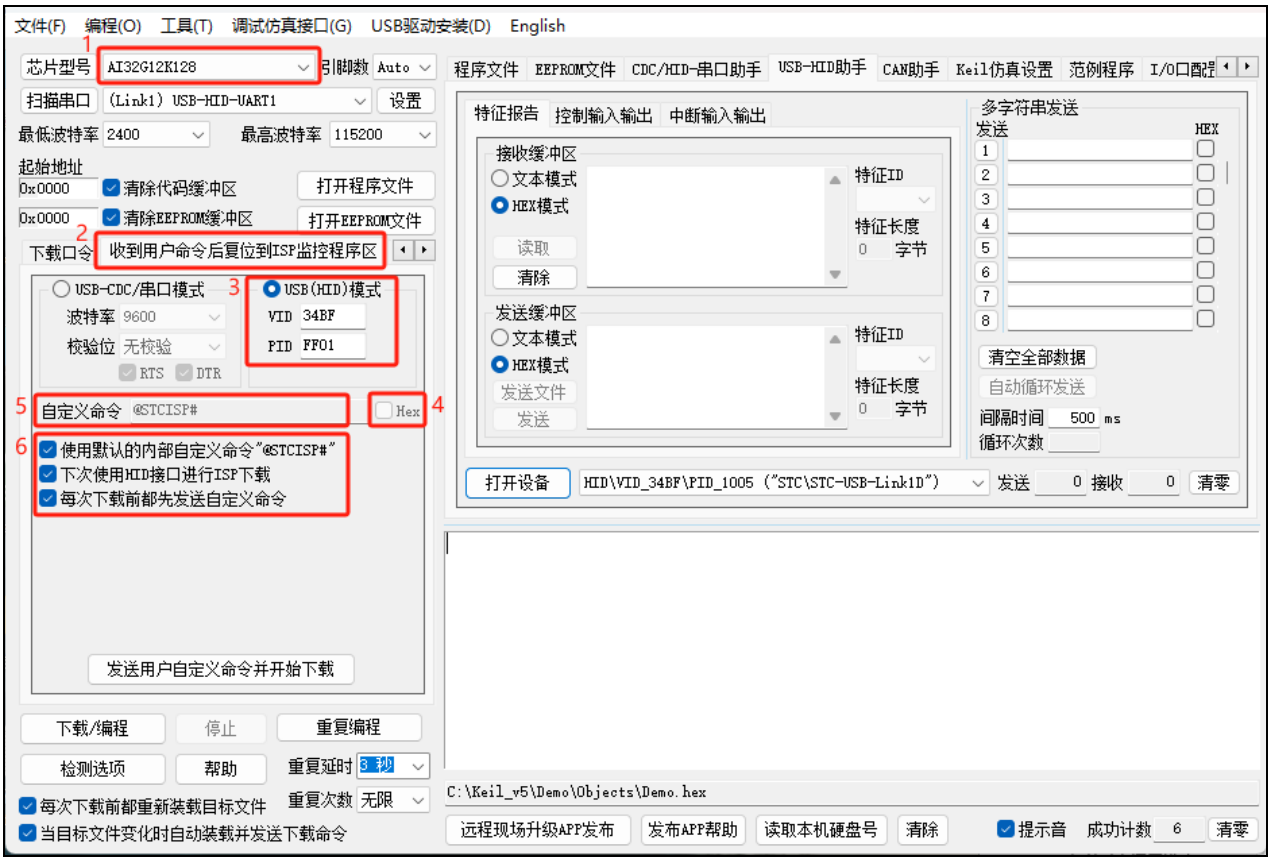
1、使用 P3.2 口的按键（非 USB 项目）

这里使用 P3.2 口的按键触发软复位和“P3.2 口短路到 GND，然后对目标芯片进行重新上电”的方法不一样。用户程序的主循环中，判断 P3.2 口电平状态，当检测到 P3.2 口电平为 0 时，触发软件复位到系统区即可进行 USB ISP 下载。P3.2 口的按键在释放状态时，用户程序从 P3.2 口读取的电平为 1，当需要复位到 ISP 进行 USB 下载时，只需手动按一下 P3.2 即可。

程序中判断 P3.2 电平的范例程序如下：

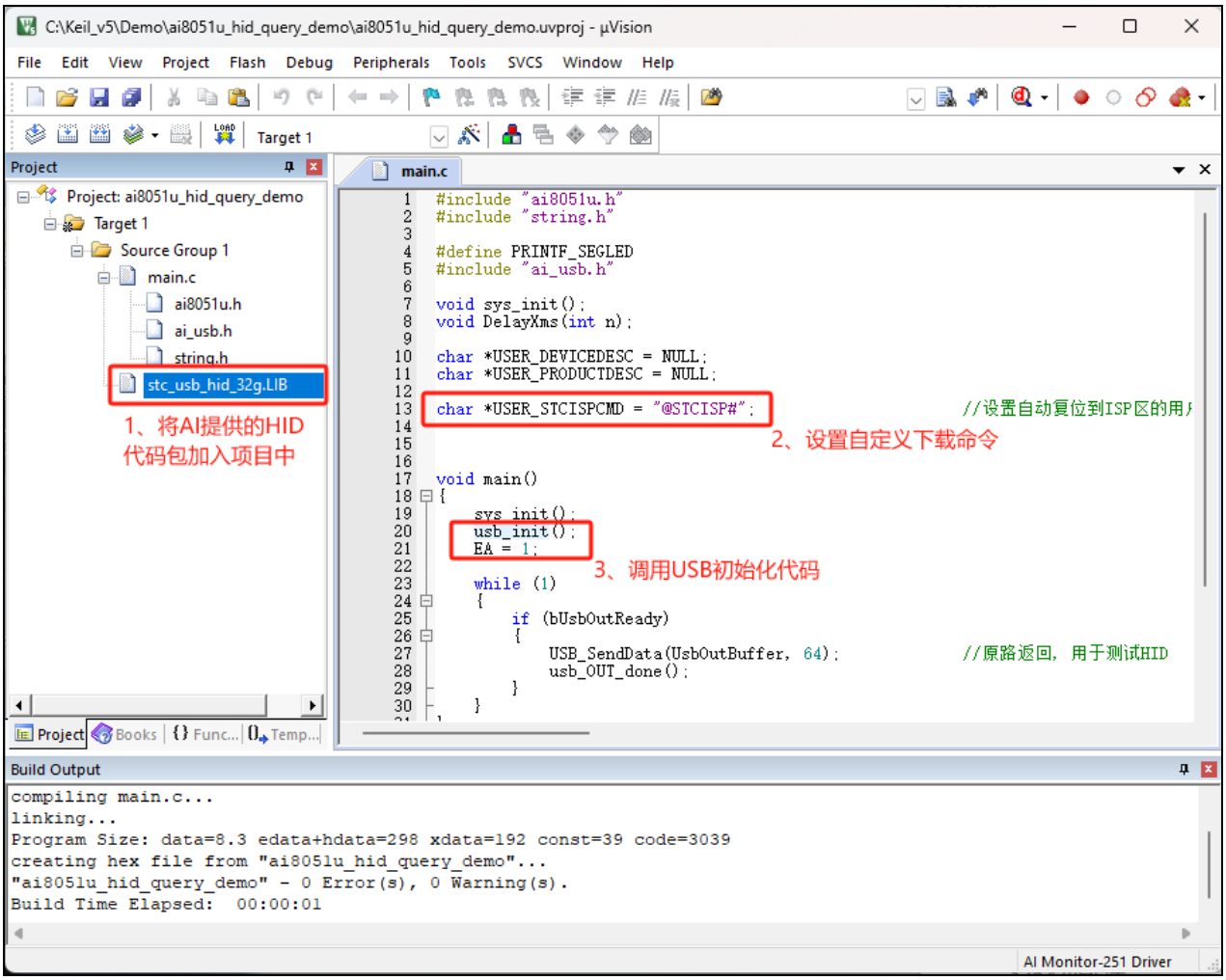
```
//测试工作频率为11.0592MHz

#include “Ai8051U.H”                    //头文件见下载软件

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                           //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P3M0 = 0x00;
    P3M1 = 0x00;
    P32 = 1;

    while (1)
    {
        if (!P32) IAP_CONTR = 0x60;     //当检测到 P3.2 的电平为低时
                                        //软件复位到系统区

        ...                             //用户代码
    }
}
```

2、使用 ISP 下载软件发送的用户下载命令（USB 项目）



1. 选择正确的单片机型号
2. 打开“收到用户命令后复位到 ISP 监控程序区”选项页
3. 选择“USB(HID)模式”，并设置 USB 设备的 VID 和 PID，范例中的 VID 为“34BF”，PID 为“FF01”
4. 选择 HEX 模式或者文本模式
5. 设置自定义下载命令，需要和代码中的自定义命令相一致
6. 选择上这两项，当目标代码重新编译后，ISP 下载软件便会自动发送复位命令，并自动开始 USB 模式的 ISP 下载

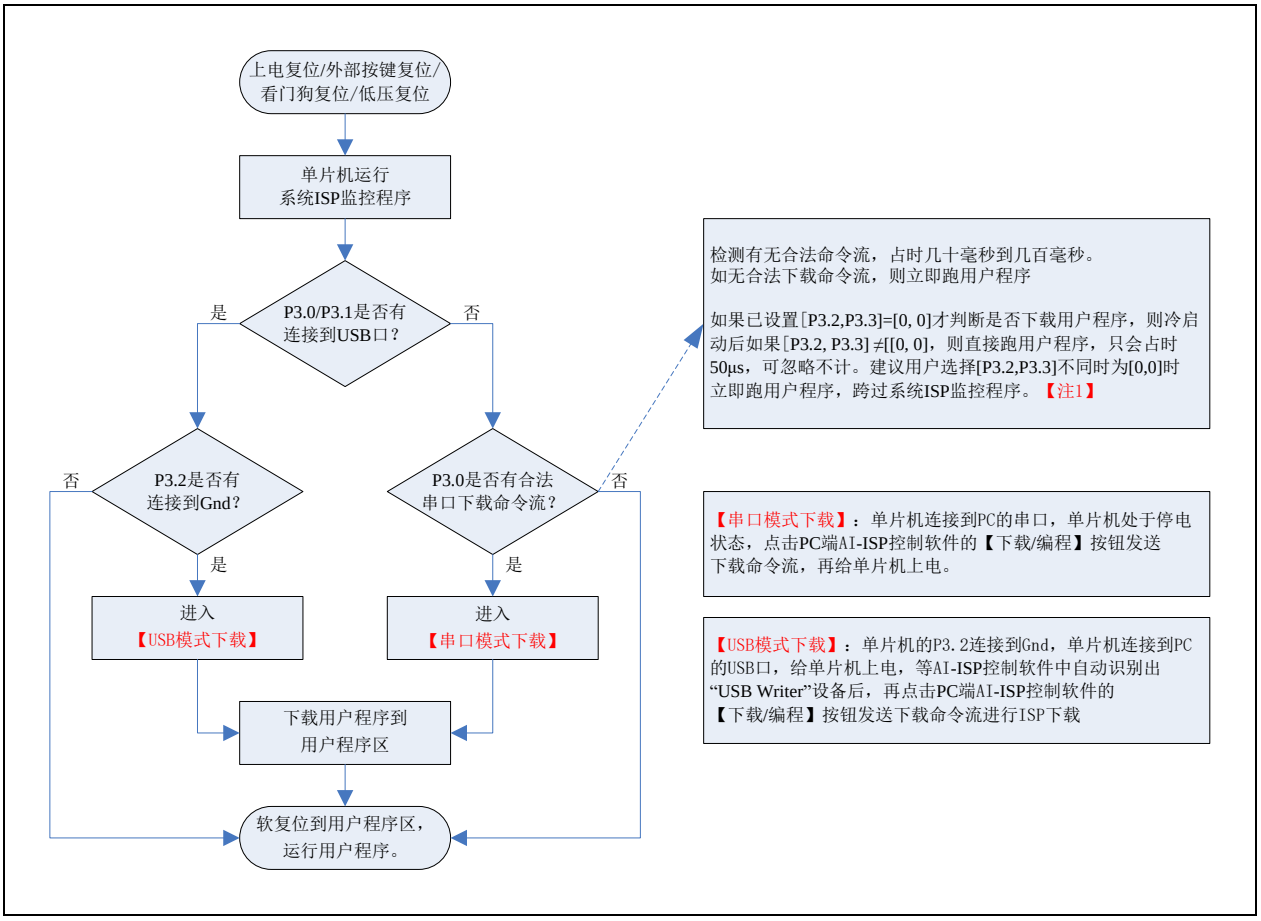
注意：若需要使用此模式，则必须将 “stc_usb_hid.lib”代码库添加到项目中，并按照下图所示的方式设置自定义下载命令。



详细代码请参考官网上的“AI32G 实验箱演示程序”包中的“76-通过 USB HID 协议打印数据信息-可用于调试”

2.32 ISP 下载流程及典型应用线路图，USB-TypeC 接口

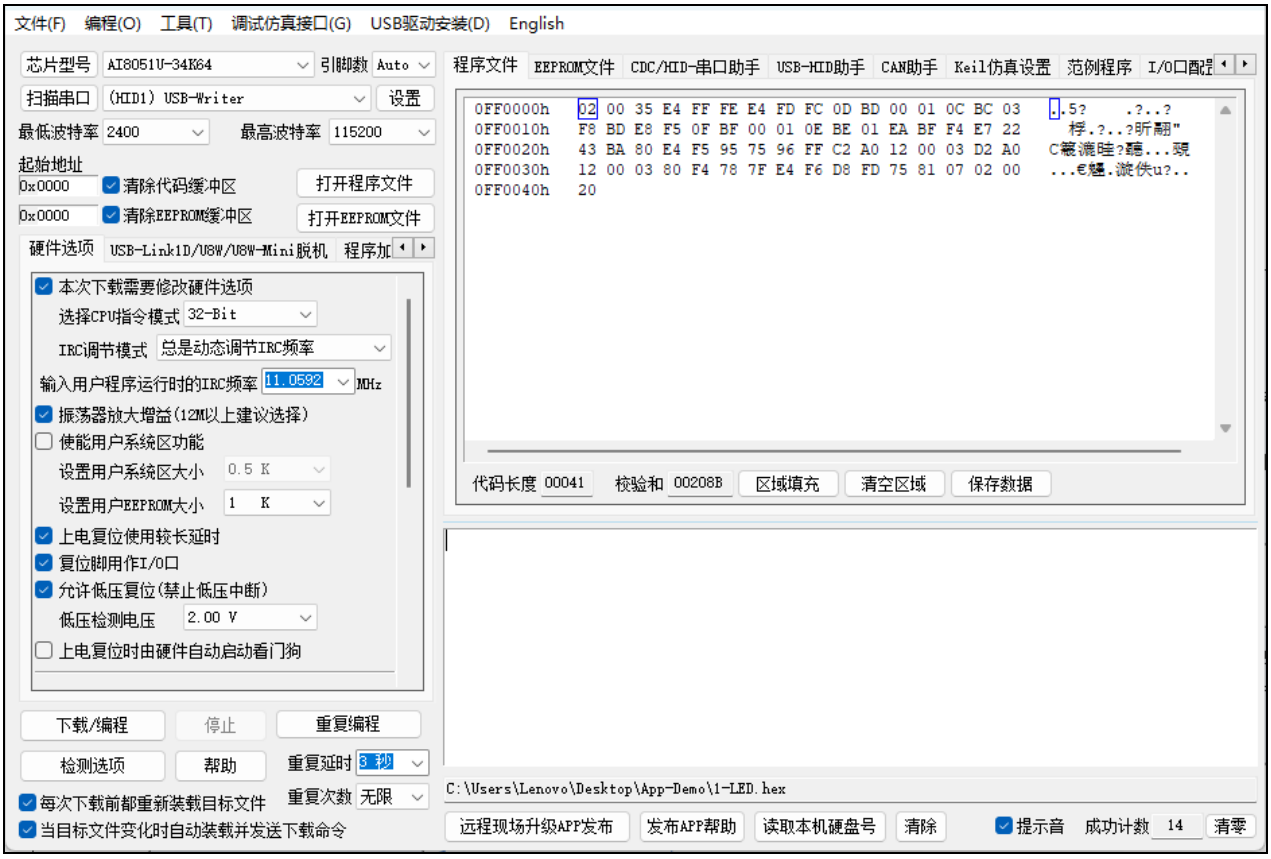
2.32.1 ISP 下载流程图（硬件/软件模拟 USB+串口模式）



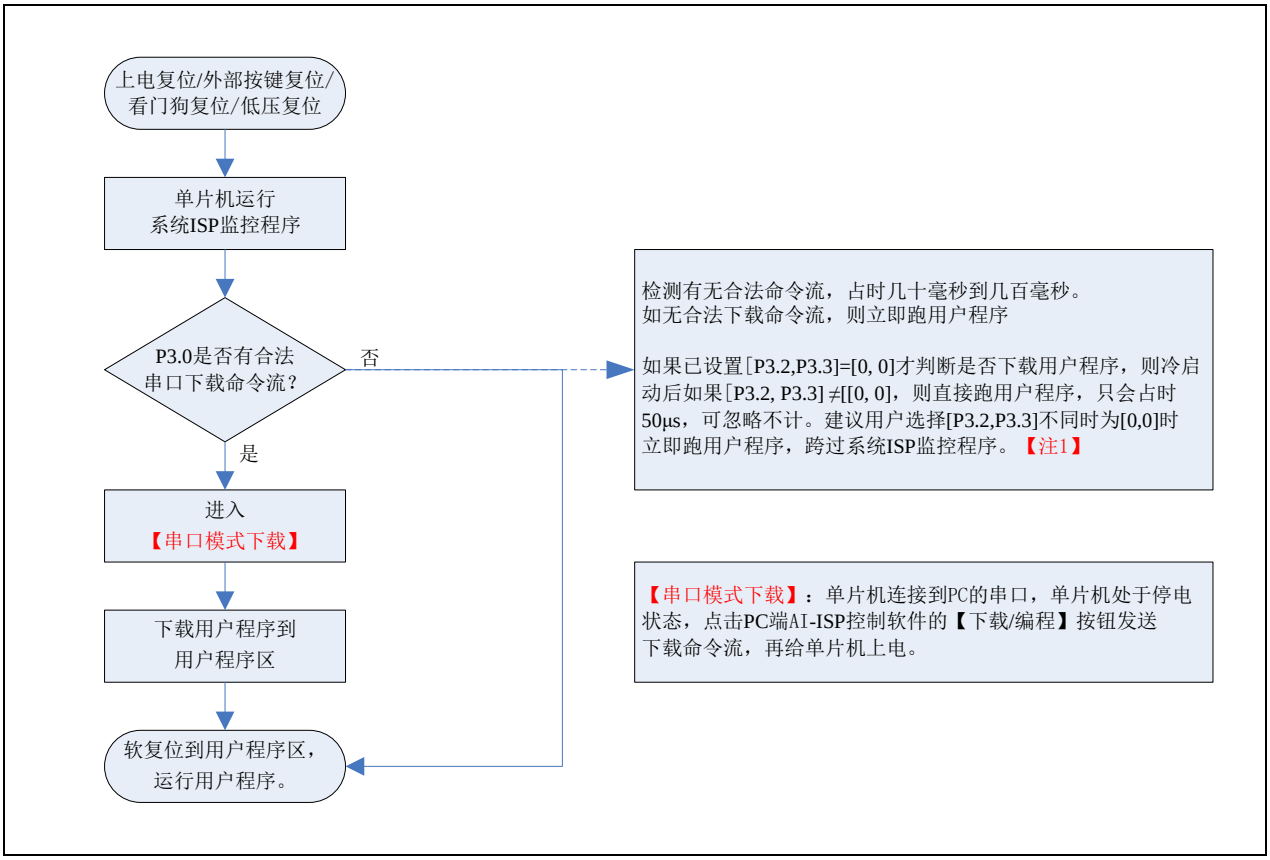
注意：因 [P3.0, P3.1] 作下载/仿真用(下载/仿真接口仅可用 [P3.0, P3.1])，故建议用户将串口 1 放在 P3.6/P3.7 或 P1.6/P1.7 ，若用户不想切换，坚持使用 P3.0/P3.1 工作或作为串口 1 进行通信，则务必在下载程序时，在软件上勾选“下次冷启动时，P3.2/P3.3 为 0/0 时才可以下载程序”。【注 1】

【注 1】：AI15，AI8 系列及以后新出的芯片的烧录保护引脚为 P3.2/P3.3，之前早期芯片的烧录保护引脚为 P1.0/P1.1。

AIapp-ISP 界面如下图所示：


文件(F) 编程(O) 工具(T) 调试仿真接口(G) USB驱动安装(D) English
芯片型号 AI8051U-34K64 引脚数 Auto
扫描串口 (HID1) USB-Writer 设置
最低波特率 2400 最高波特率 115200
起始地址
0x0000 清除代码缓冲区 打开程序文件
0x0000 清除EEPROM缓冲区 打开EEPROM文件
硬件选项 USB-Link1D/U8W/U8W-Mini脱机 程序加
本次下载需要修改硬件选项
选择CPU指令模式 32-Bit
IRC调节模式 总是动态调节IRC频率
输入用户程序运行时的IRC频率 11.0592 MHz
振荡器放大增益(12M以上建议选择)
使能用户系统区功能
设置用户系统区大小 0.5 K
设置用户EEPROM大小 1 K
上电复位使用较长延时
复位脚用作I/O口
允许低压复位(禁止低压中断)
低压检测电压 2.00 V
上电复位时由硬件自动启动看门狗
下载/编程 停止 重复编程
检测选项 帮助 重复延时 8 秒
每次下载前都重新装载目标文件 重复次数 无限
当目标文件变化时自动装载并发送下载命令
程序文件 EEPROM文件 CDC/HID-串口助手 USB-HID助手 CAN助手 Keil仿真设置 范例程序 I/O口配置
0FF0000h 02 00 35 E4 FF FE E4 FD FC 0D BD 00 01 0C BC 03
0FF0010h F8 BD E8 F5 0F BF 00 01 0E BE 01 EA BF F4 E7 22
0FF0020h 43 BA 80 E4 F5 95 75 96 FF C2 A0 12 00 03 D2 A0
0FF0030h 12 00 03 80 F4 78 7F E4 F6 D8 FD 75 81 07 02 00
0FF0040h 20
代码长度 00041 校验和 00208B 区域填充 清空区域 保存数据
C:\Users\Lenovo\Desktop\App-Demo\1-LED.hex
远程现场升级APP发布 发布APP帮助 读取本机硬盘号 清除 提示音 成功计数 14 清零

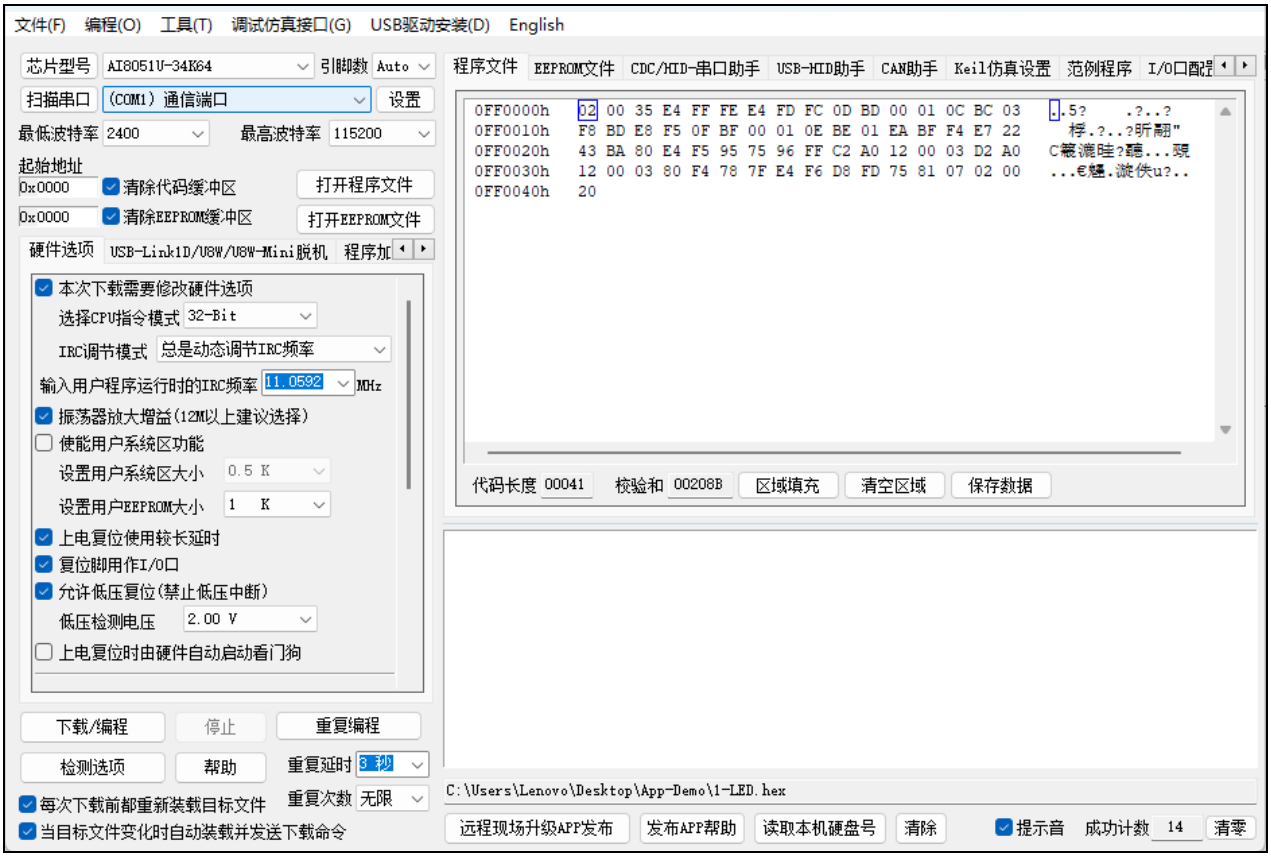
2.32.2　ISP 下载流程图（串口下载模式）



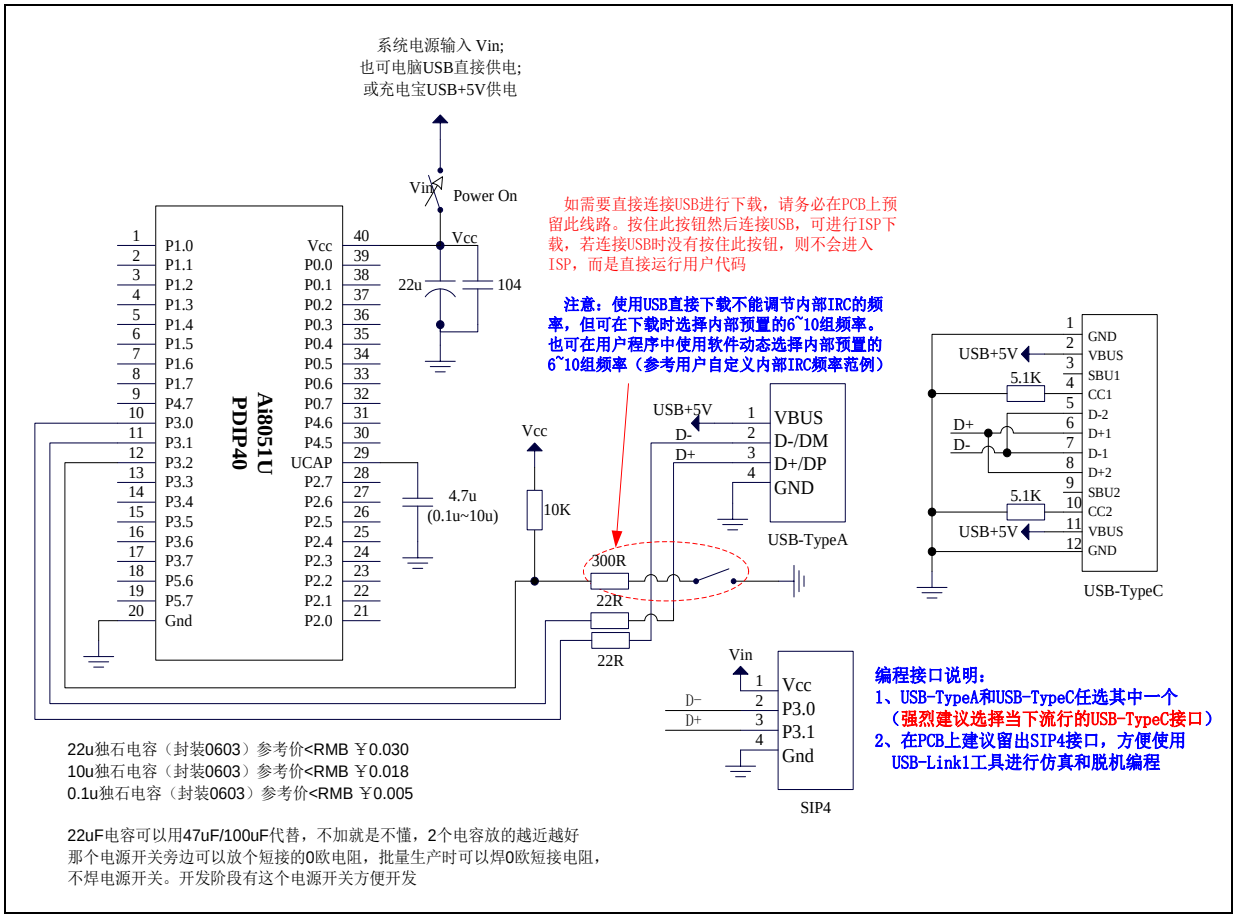
注意：因 [P3.0, P3.1] 作下载/仿真用(下载/仿真接口仅可用 [P3.0, P3.1])，故建议用户将串口 1 放在 P3.6/P3.7 或 P1.6/P1.7 ，若用户不想切换，坚持使用 P3.0/P3.1 工作或作为串口 1 进行通信，则务必在下载程序时，在软件上勾选“下次冷启动时，P3.2/P3.3 为 0/0 时才可以下载程序”。【注 1】

【注 1】：AI15，AI8 系列及以后新出的芯片的烧录保护引脚为 P3.2/P3.3，之前早期芯片的烧录保护引脚为 P1.0/P1.1。

AIapp-ISP 界面如下图所示：


文件(F) 编程(O) 工具(T) 调试仿真接口(G) USB驱动安装(D) English
芯片型号 AI8051U-34K64 引脚数 Auto
扫描串口 (COM1) 通信端口 设置
最低波特率 2400 最高波特率 115200
起始地址
0x0000 清除代码缓冲区 打开程序文件
0x0000 清除EEPROM缓冲区 打开EEPROM文件
硬件选项 USB-Link1D/U8W/U8W-Mini脱机 程序加
本次下载需要修改硬件选项
选择CPU指令模式 32-Bit
IRC调节模式 总是动态调节IRC频率
输入用户程序运行时的IRC频率 11.0592 MHz
振荡器放大增益(12M以上建议选择)
使能用户系统区功能
设置用户系统区大小 0.5 K
设置用户EEPROM大小 1 K
上电复位使用较长延时
复位脚用作I/O口
允许低压复位(禁止低压中断)
低压检测电压 2.00 V
上电复位时由硬件自动启动看门狗
下载/编程 停止 重复编程
检测选项 帮助 重复延时 3 秒
每次下载前都重新装载目标文件 重复次数 无限
当目标文件变化时自动装载并发送下载命令
程序文件 EEPROM文件 CDC/HID-串口助手 USB-HID助手 CAN助手 Keil仿真设置 范例程序 I/O口配置
0FF0000h 02 00 35 E4 FF FE E4 FD FC 0D BD 00 01 0C BC 03
0FF0010h F8 BD E8 F5 0F BF 00 01 0E BE 01 EA BF F4 E7 22
0FF0020h 43 BA 80 E4 F5 95 75 96 FF C2 A0 12 00 03 D2 A0
0FF0030h 12 00 03 80 F4 78 7F E4 F6 D8 FD 75 81 07 02 00
0FF0040h 20
代码长度 00041 校验和 00208B 区域填充 清空区域 保存数据
C:\Users\Lenovo\Desktop\App-Demo\1-LED.hex
远程现场升级APP发布 发布APP帮助 读取本机硬盘号 清除 提示音 成功计数 14 清零

2.32.3 USB-TypeC / TypeA / 普通插座都支持 USB 直接下载，5V



USB-ISP 下载程序步骤：

1、按下板子上的 P3.2/INT0 按键，就是 P3.2 接地

2、给目标芯片重新上电，不管之前是否已通电。

===电子开关是按下停电后再松开就是上电，等待 ISP 下载软件中自动识别出“USB Writer (HID1)”，识别出来后，就与 P3.2 状态无关了，这时可以松开 P3.2 按键

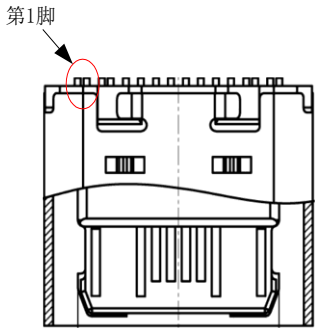
===传统的机械自锁紧开关是按上来停电，按下去是上电

3、点击下载软件中的“下载/编程”按钮（注意：USB 下载与串口下载的操作顺序不同）

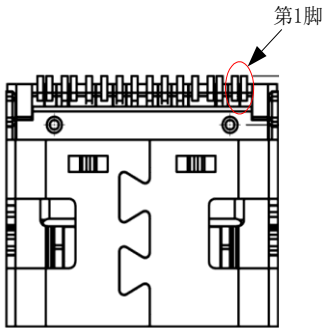
下载成功 ！

===另外从用户区软复位到系统区也是等待 USB 下载。

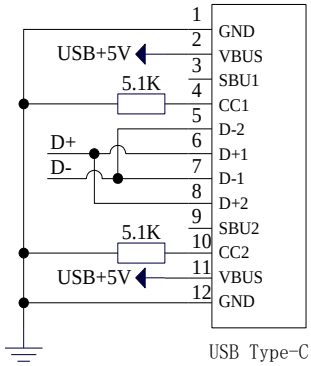
当用户使用硬件 USB 对 Ai8051U 系列进行 ISP 下载时不能调节内部 IRC 的频率，但用户可以选择芯片出厂时内部预置的 16 个频率（分别是 5.5296M、6M、11.0592M、12M、18.432M、20M、22.1184M、24M、27M、30M、33.1776M、35M、36.864M、40M、44.2368M 和 48M，不同的系列可能不一样，具体以下载软件的频率列表为准）。下载时用户只能从频率下拉列表中进行选择其中之一，而不能手动输入其他频率。（使用串口下载则可用输入 4M~48M 之间的任意频率）。



USB Type-C
实物图(正面)



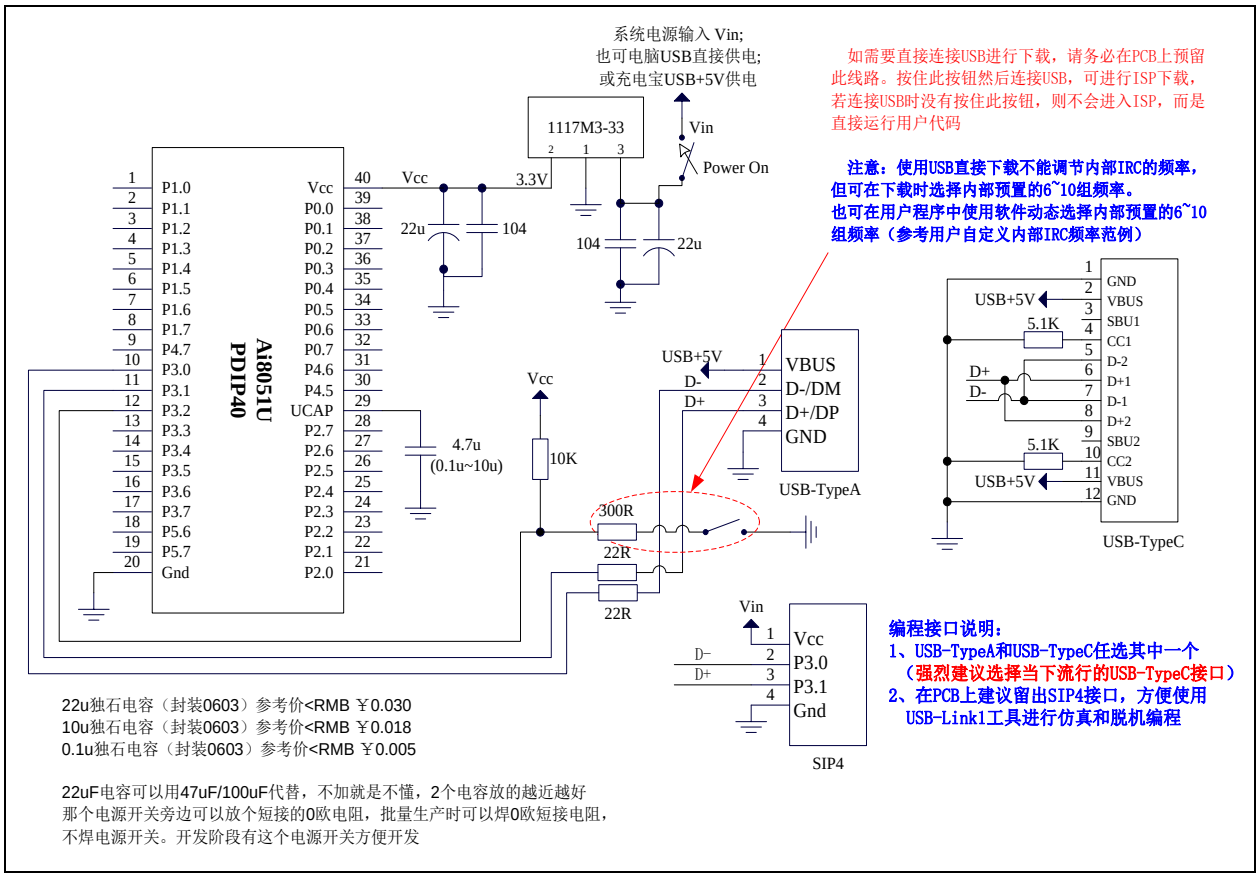
USB Type-C
实物图(反面)



USB Type-C
电路原理图

	A1	B12	A4	B9	A8	A5	B7	A6	A7	B6	B8	B5	B4	A9	B1	A12
	GND	GND	VBUS	VBUS	SBU1	CC1	D-2	D+1	D-1	D+2	SBU2	CC2	VBUS	VBUS	GND	GND
Type-C 16PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Type-C 12PIN	1		2		3	4	5	6	7	8	9	10	11		12	

2.32.4 USB-TypeC / TypeA / 普通插座都支持 USB 直接下载，3.3V



USB-ISP 下载程序步骤：

1、按下板子上的 P3.2/INT0 按键，就是 P3.2 接地

2、给目标芯片重新上电，不管之前是否已通电。

===电子开关是按下停电后再松开就是上电，等待 ISP 下载软件中自动识别出“USB Writer (HID1)”，识别出来后，就与 P3.2 状态无关了，这时可以松开 P3.2 按键

===传统的机械自锁紧开关是按上来停电，按下去是上电

3、点击下载软件中的“下载/编程”按钮（注意：USB 下载与串口下载的操作顺序不同）下载成功 ！

===另外从用户区软复位到系统区也是等待 USB 下载。

项目开发温馨提示：一般 USB 直接进行 ISP 下载是提供给您的客户升级代码时使用的，而项目开发阶段应该使用（强烈建议）我公司提供的 USB-Link1D 工具。USB-Link1D 工具给项目开发可提供如下便利：

1、ISP 下载时，工具能够自动停电和上电，可免去手动给目标芯片上电的麻烦

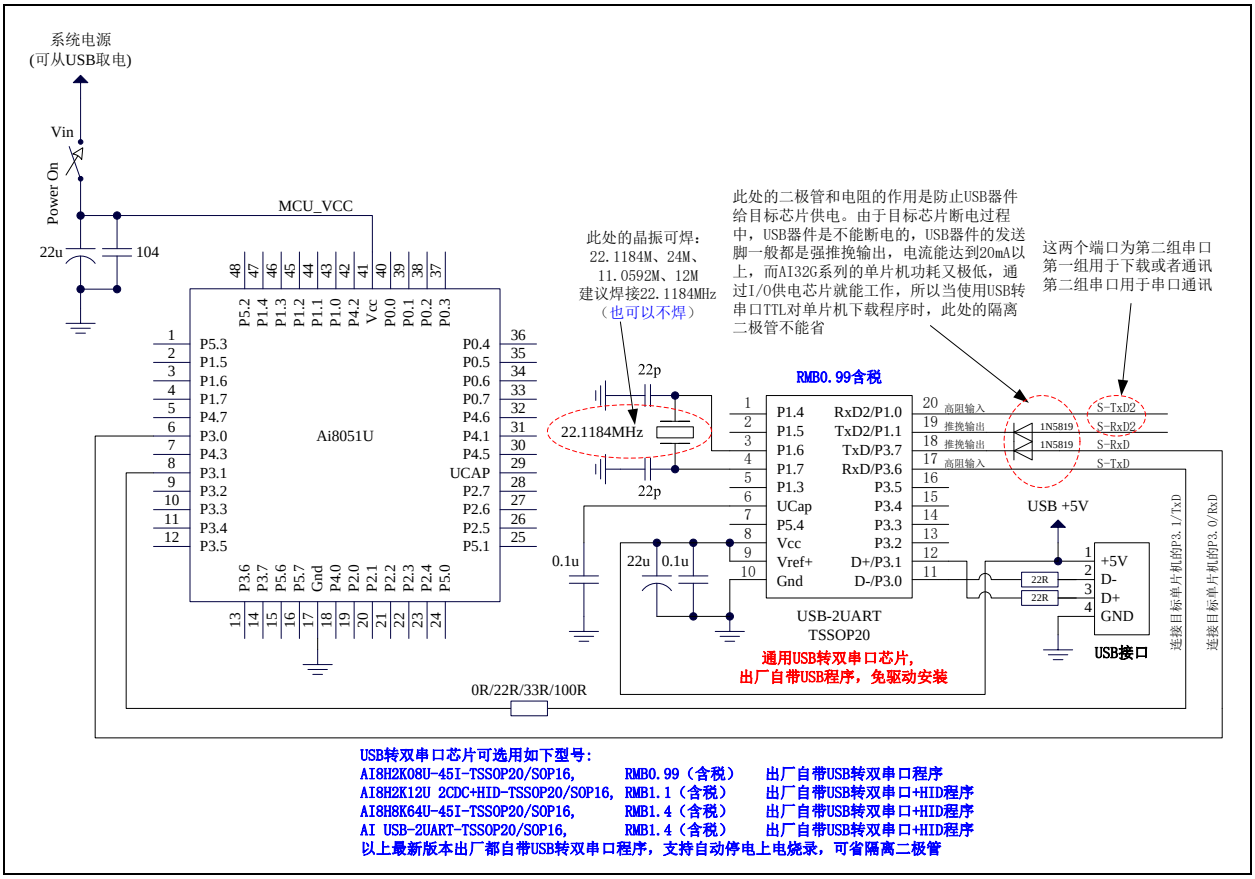
2、工具能够根据选择的目标单片机智能的提供 3.3V 或者 5V 的 VCC 电源

3、可直接使用工具对目标芯片进行串口模式仿真

4、在不进行 ISP 下载时，下载口就是一个 USB-CDC 串口 1，可协助工程师调试程序

5、另外工具还额外送一个独立的 USB-CDC 串口 2，当使用 USB-CDC 串口 1 进行仿真的同时还可以使用 USB-CDC 串口 2 调试程序中的串口模块。所以，对于一个专业的企业级公司，应给您的软件工程师人手一个 USB-Link1D 工具，从而极大提高项目开发进度。

2.32.5　使用 USB 转双串口/TTL 下载（有外部晶振）



ISP 下载步骤：

1、 给目标芯片停电，注意不能给“USB-2UART”芯片停电

2、 由于“USB-2UART”芯片的发送脚是强推挽输出，必须在目标芯片的 P3.0 口和“USB-2UART”的发送脚之间串接一个二极管，否则目标芯片无法完全断电，达不到给目标芯片停电的目标。

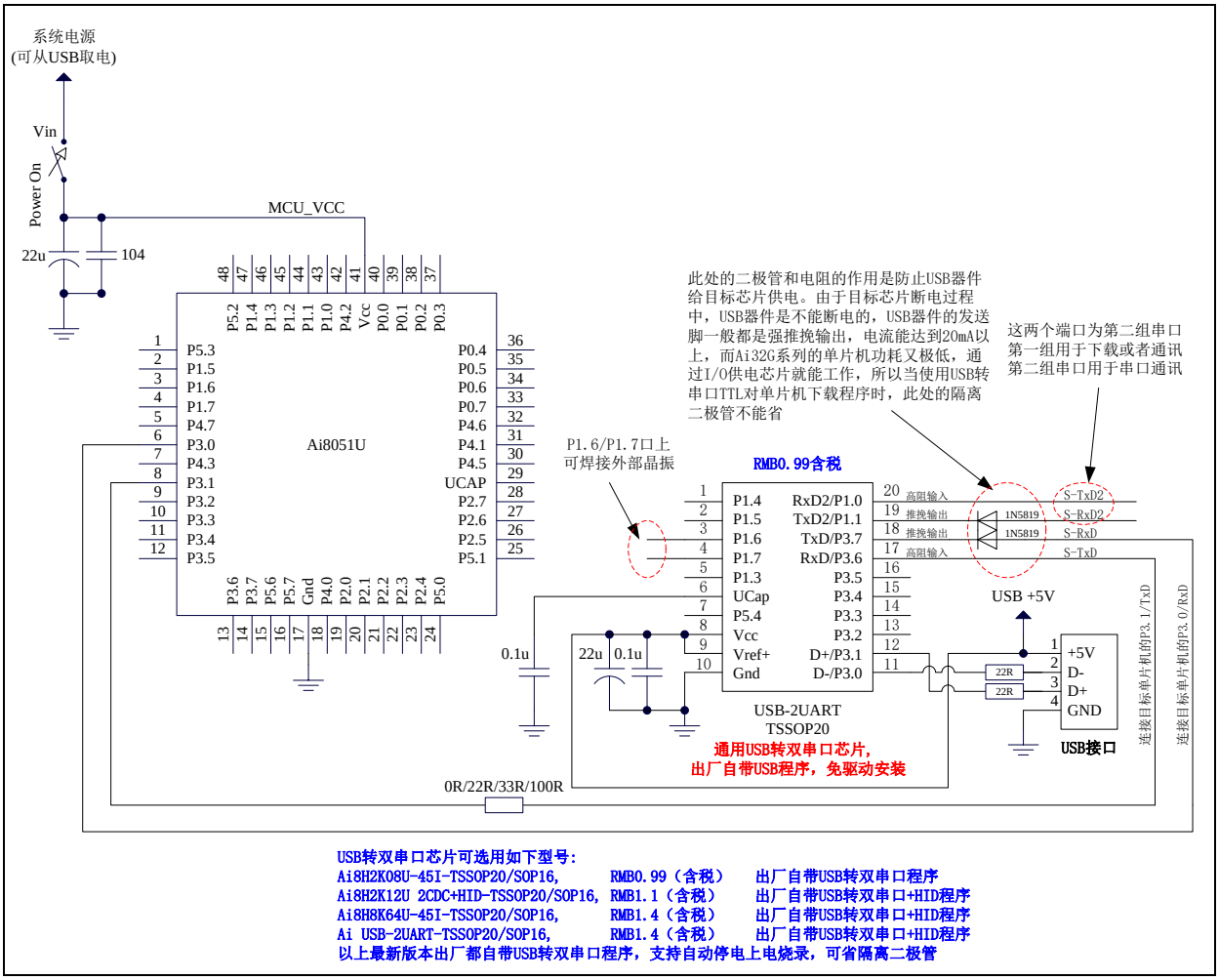
3、 点击 ISP 下载软件中的“下载/编程”按钮

4、 给目标芯片上电

5、 开始 ISP 下载

注意：目前有发现使用 USB 线供电进行 ISP 下载时，由于 USB 线太细，在 USB 线上的压降过大，导致 ISP 下载时供电不足，所以请在使用 USB 线供电进行 ISP 下载时，务必使用 USB 加强线。

2.32.6　使用 USB 转双串口/TTL 下载（无外部晶振）



ISP 下载步骤：

1、 给目标芯片停电，注意不能给“USB-2UART”芯片停电

2、 由于“USB-2UART”芯片的发送脚是强推挽输出，必须在目标芯片的 P3.0 口和“USB-2UART”的发送脚之间串接一个二极管，否则目标芯片无法完全断电，达不到给目标芯片停电的目标。

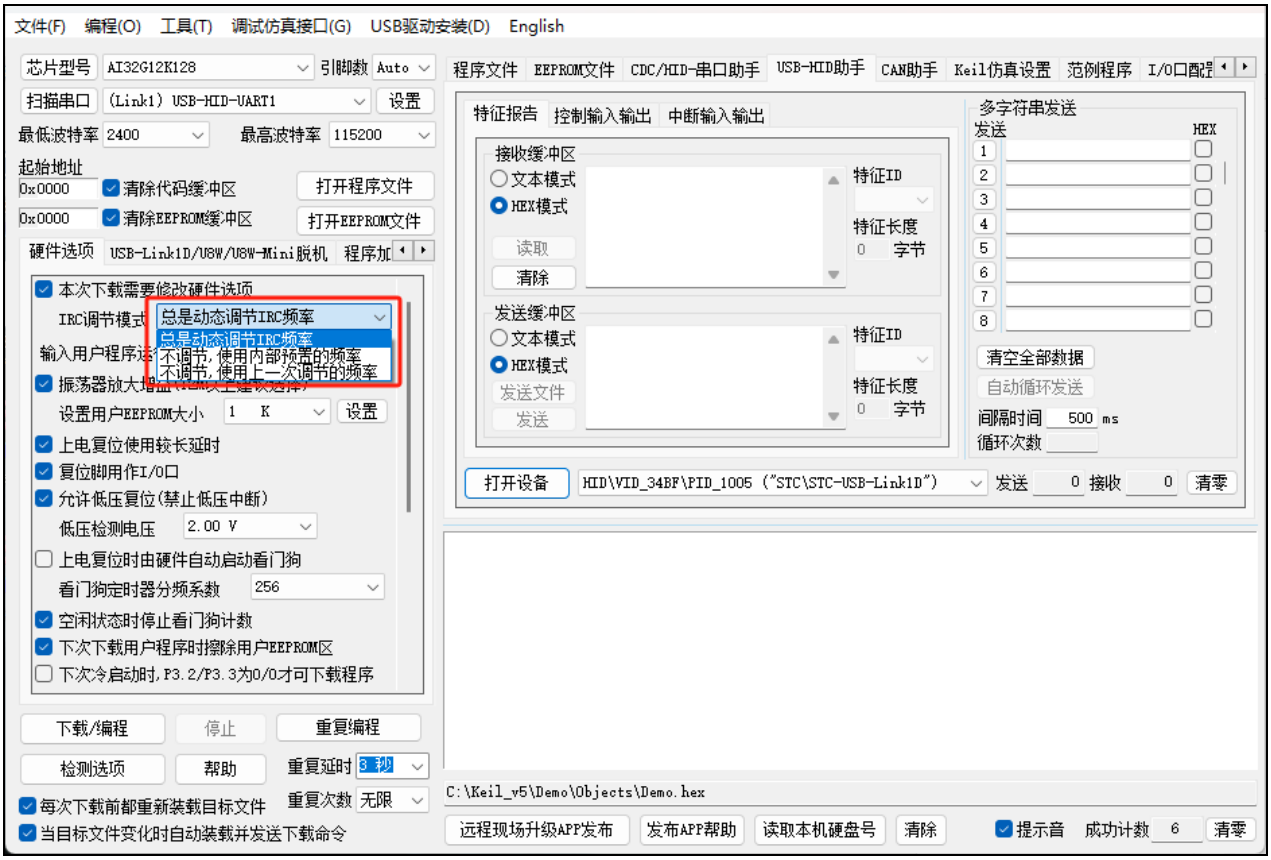
3、 点击 ISP 下载软件中的“下载/编程”按钮

4、 给目标芯片上电

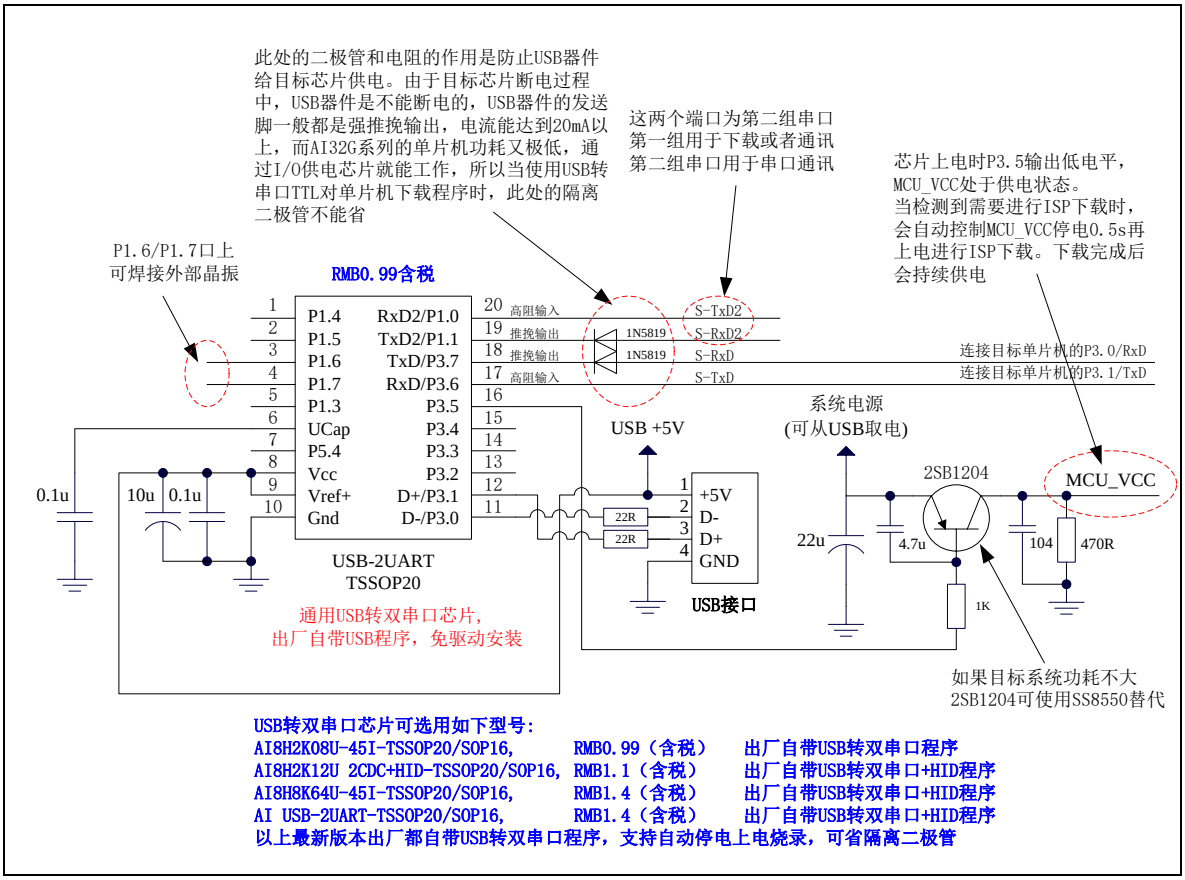
5、 开始 ISP 下载

注意：目前有发现使用 USB 线供电进行 ISP 下载时，由于 USB 线太细，在 USB 线上的压降过大，导致 ISP 下载时供电不足，所以请在使用 USB 线供电进行 ISP 下载时，务必使用 USB 加强线。

注意：如果使用无外部晶振的 USB 转双串口/TTL 下载时，强烈建议 ISP 下载选项“选择 IRC 调节模式”选择“不调节，使用内部预置的频率”选项，这样可以避免调节频率时将无外部晶振的 USB 转双串口/TTL 工具本身的频率误差代入目标芯片。如下图：



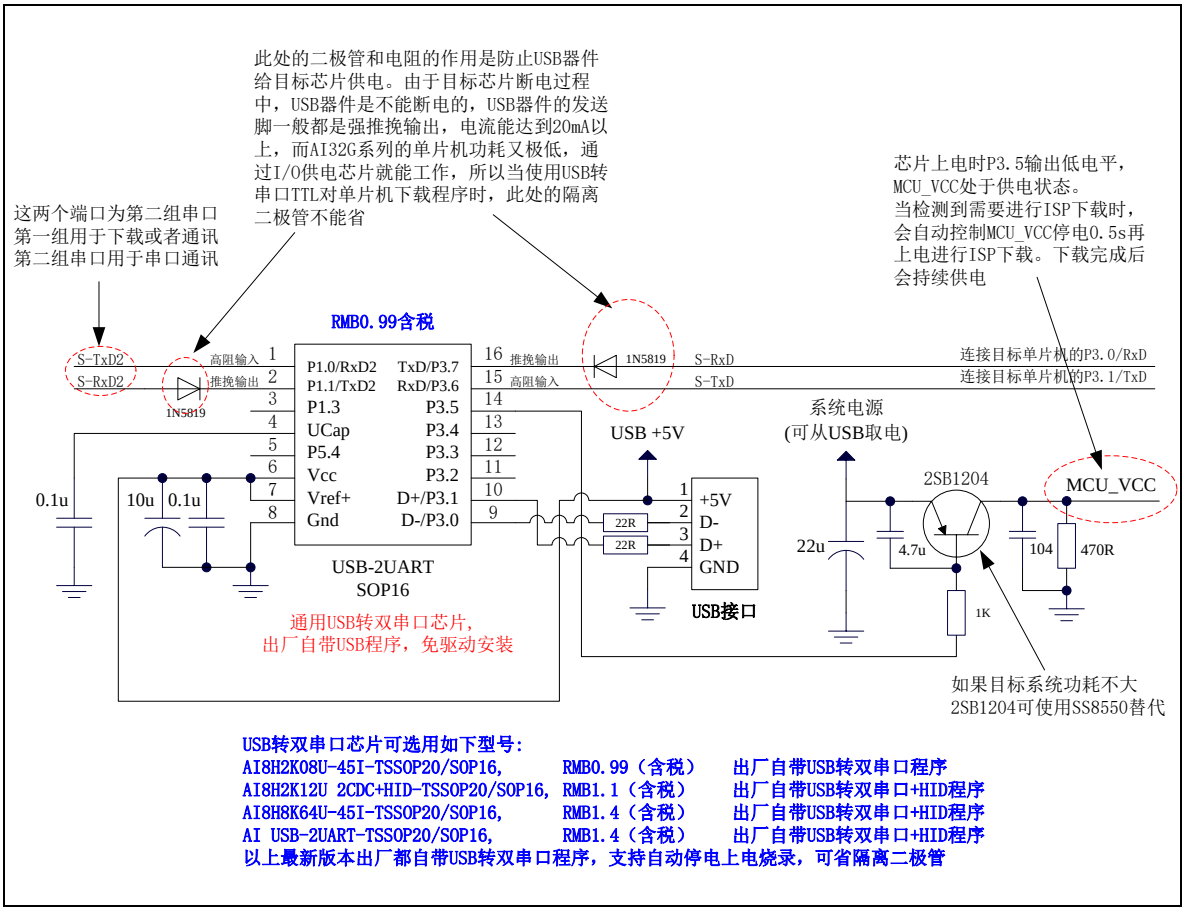
2.32.7　使用 USB 转双串口/TTL 下载（自动停电/上电）



USB转双串口芯片可选用如下型号：

AI8H2K08U-45I-TSSOP20/SOP16,	RMB0.99（含税）	出厂自带USB转双串口程序
AI8H2K12U 2CDC+HID-TSSOP20/SOP16,	RMB1.1（含税）	出厂自带USB转双串口+HID程序
AI8H8K64U-45I-TSSOP20/SOP16,	RMB1.4（含税）	出厂自带USB转双串口+HID程序
AI USB-2UART-TSSOP20/SOP16,	RMB1.4（含税）	出厂自带USB转双串口+HID程序

以上最新版本出厂都自带USB转双串口程序，支持自动停电上电烧录，可省隔离二极管

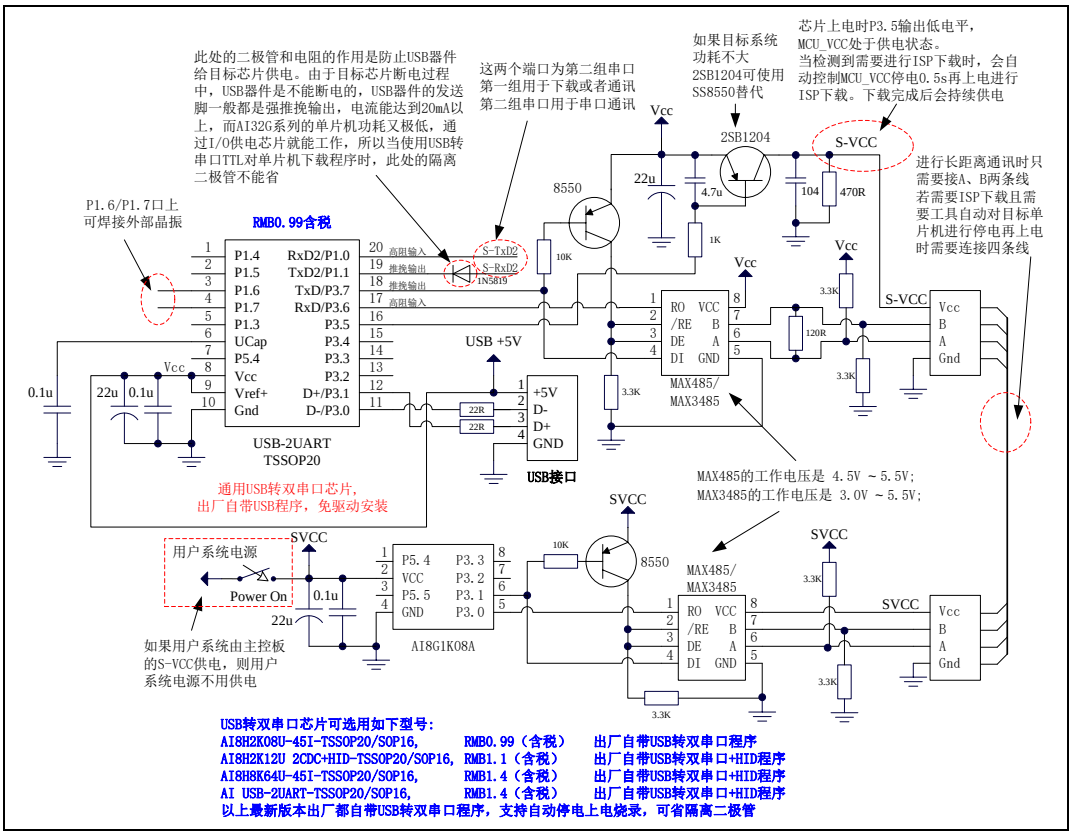


USB转双串口芯片可选用如下型号：

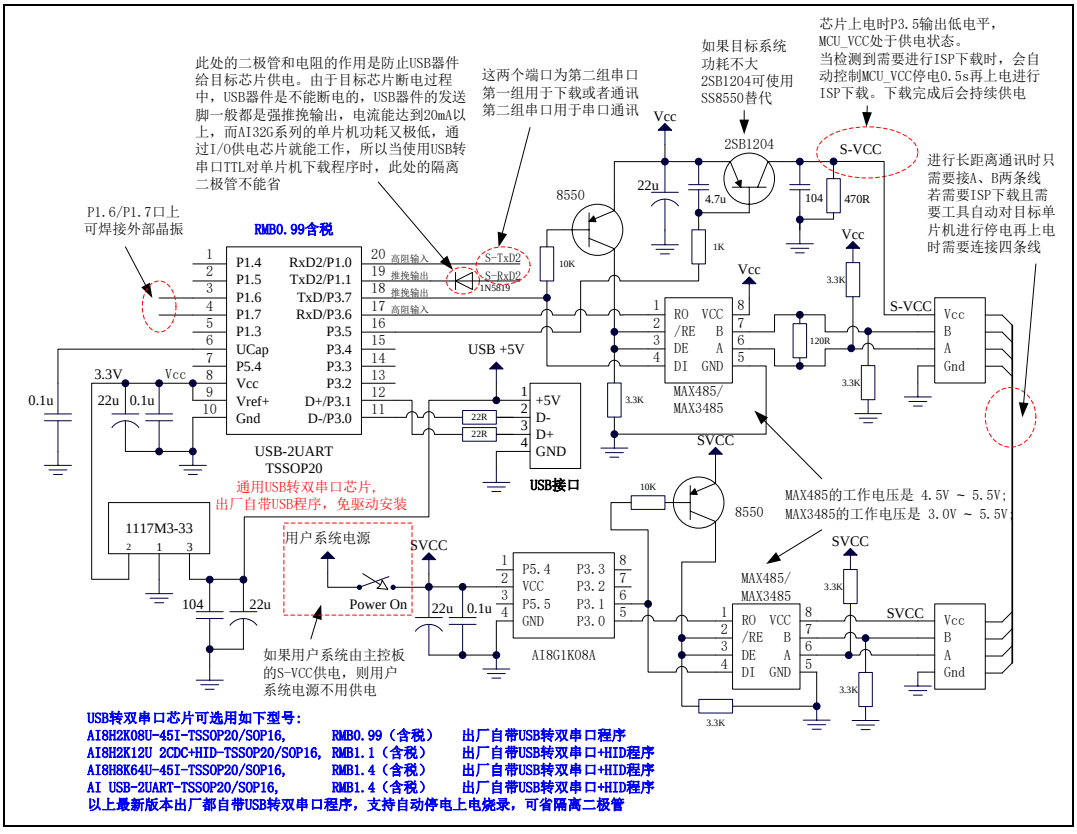
AI8H2K08U-45I-TSSOP20/SOP16,	RMB0.99（含税）	出厂自带USB转双串口程序
AI8H2K12U 2CDC+HID-TSSOP20/SOP16,	RMB1.1（含税）	出厂自带USB转双串口+HID程序
AI8H8K64U-45I-TSSOP20/SOP16,	RMB1.4（含税）	出厂自带USB转双串口+HID程序
AI USB-2UART-TSSOP20/SOP16,	RMB1.4（含税）	出厂自带USB转双串口+HID程序

以上最新版本出厂都自带USB转双串口程序，支持自动停电上电烧录，可省隔离二极管

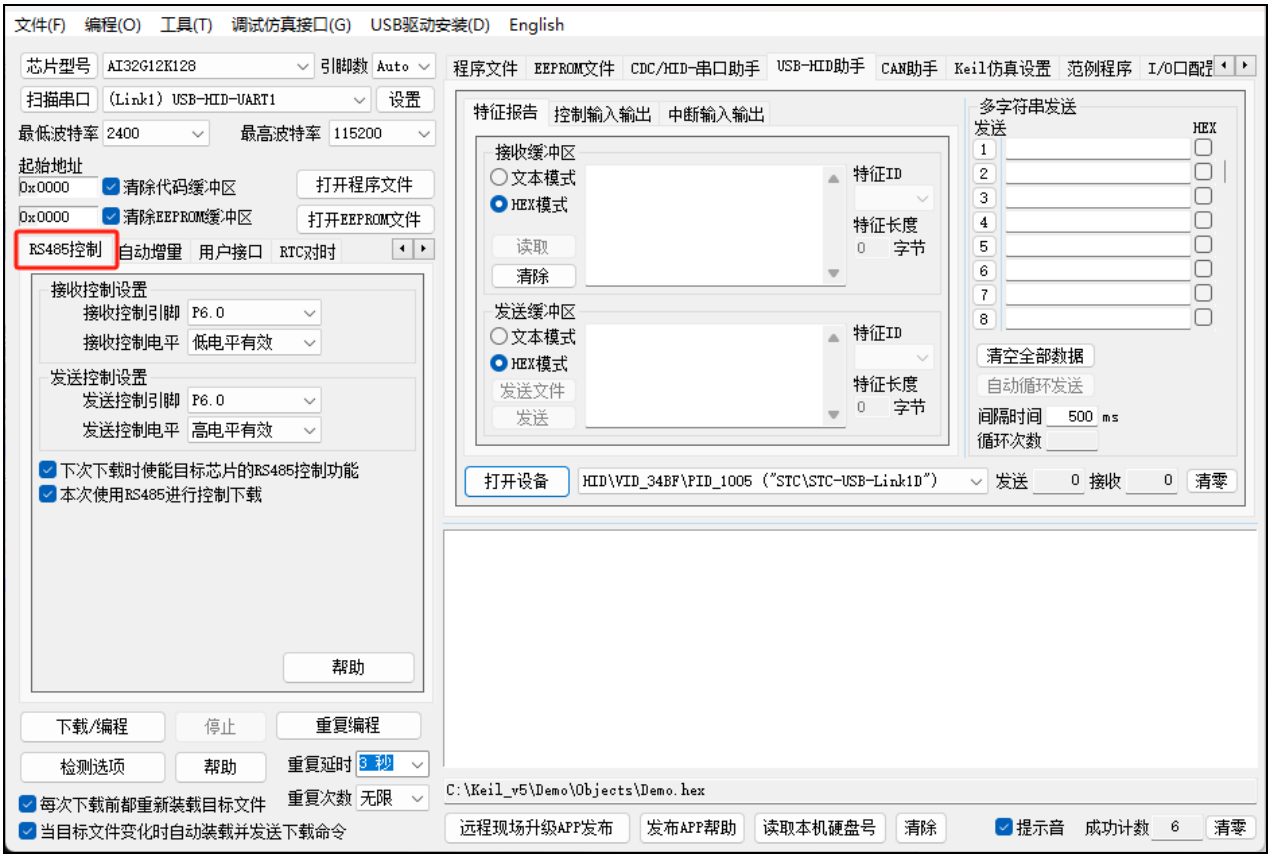
2.32.8　使用 USB 转双串口/RS485 下载（5.0V）



2.32.9　使用 USB 转双串口/RS485 下载（3.3V）



ISP 下载软件中 RS485 相关设置界面如下图：



设置项详细说明如下

“接收控制设置”：设置控制 RS485 接收脚的 I/O 口以及控制有效电平

“发送控制设置”：设置控制 RS485 发送脚的 I/O 口以及控制有效电平

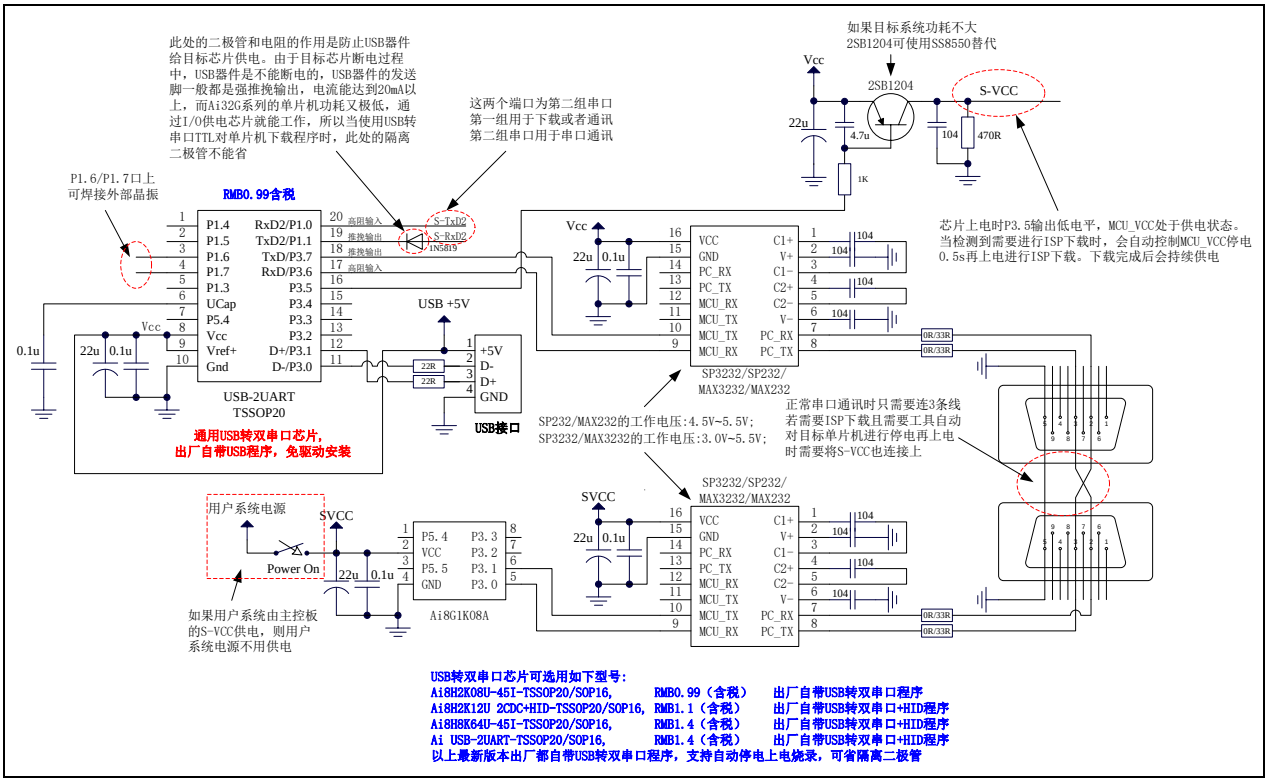
“下次下载时使能目标芯片的 RS485 控制功能”：设置目标单片机下次 ISP 下载时使能 RS485 控制（特别注意：如果用户产品需要使能 RS485 功能，则每次下载时都需要勾选此选项）

“本次使用 RS485 进行控制下载”：本次 ISP 下载软件使用 RS485 模式对目标单片机进行下载。

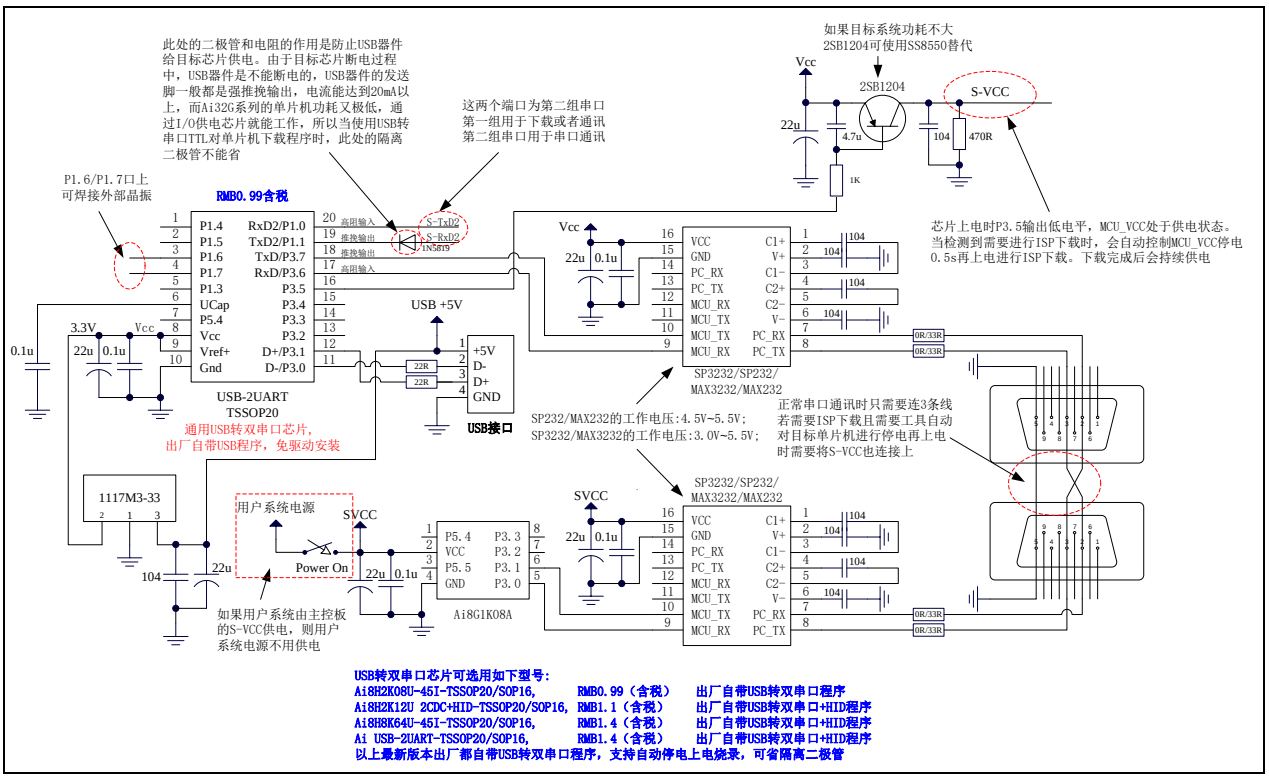
7.3.x 固件版本的单片机，需要固件版本等于或大于 7.3.12 才能很好的支持 RS485

7.4.x 固件版本的单片机都可很好的支持 RS485

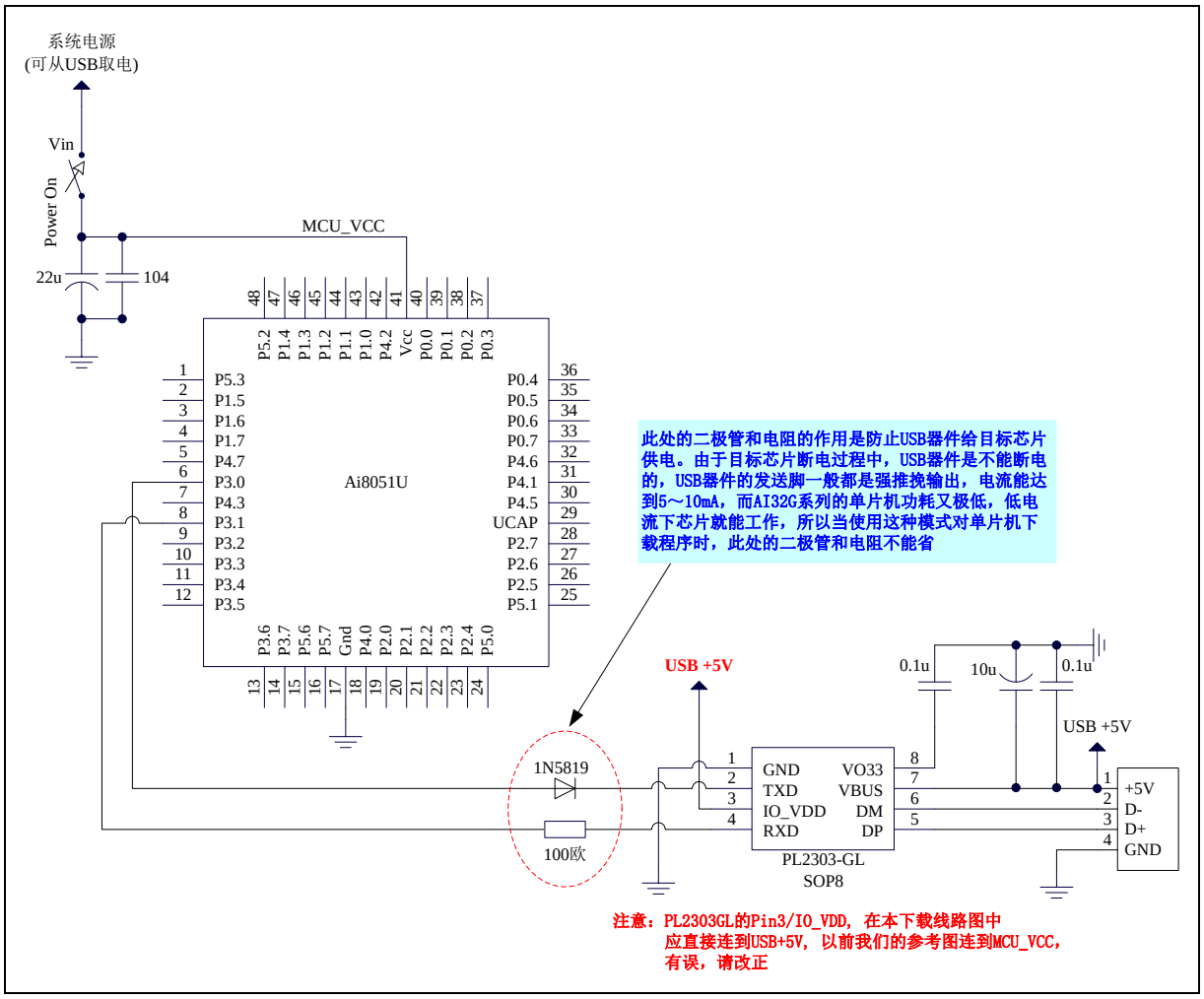
2.32.10　使用 USB 转双串口/RS232 下载（5.0V）



2.32.11　使用 USB 转双串口/RS232 下载（3.3V）



2.32.12　使用 PL2303-GL 下载



ISP 下载步骤：

1、 给目标芯片停电，注意不能给 USB 转串口芯片停电（如：CH340、PL2303-GL 等）

2、 由于 USB 转串口芯片的发送脚一般都是强推挽输出，必须在目标芯片的 P3.0 口和 USB 转串口芯片的发送脚之间串接一个二极管，否则目标芯片无法完全断电，达不到给目标芯片停电的目标。

3、 点击 ISP 下载软件中的“下载/编程”按钮

4、 给目标芯片上电

5、 开始 ISP 下载

注意：目前有发现使用 USB 线供电进行 ISP 下载时，由于 USB 线太细，在 USB 线上的压降过大，导致 ISP 下载时供电不足，所以请在使用 USB 线供电进行 ISP 下载时，务必使用 USB 加强线。

2.33 ISP 下载软件高级应用,下载需口令,控制下载次数,程序加密后传输,发布项目程序/远程现场升级 App 发布等

2.33.1 发布项目程序/远程现场升级 App 发布

发布项目程序/远程现场升级 App 发布功能主要是将用户的程序代码与相关的选项设置打包成为一个可以直接对目标芯片进行下载编程的超级简单的用户自己界面的可执行文件。

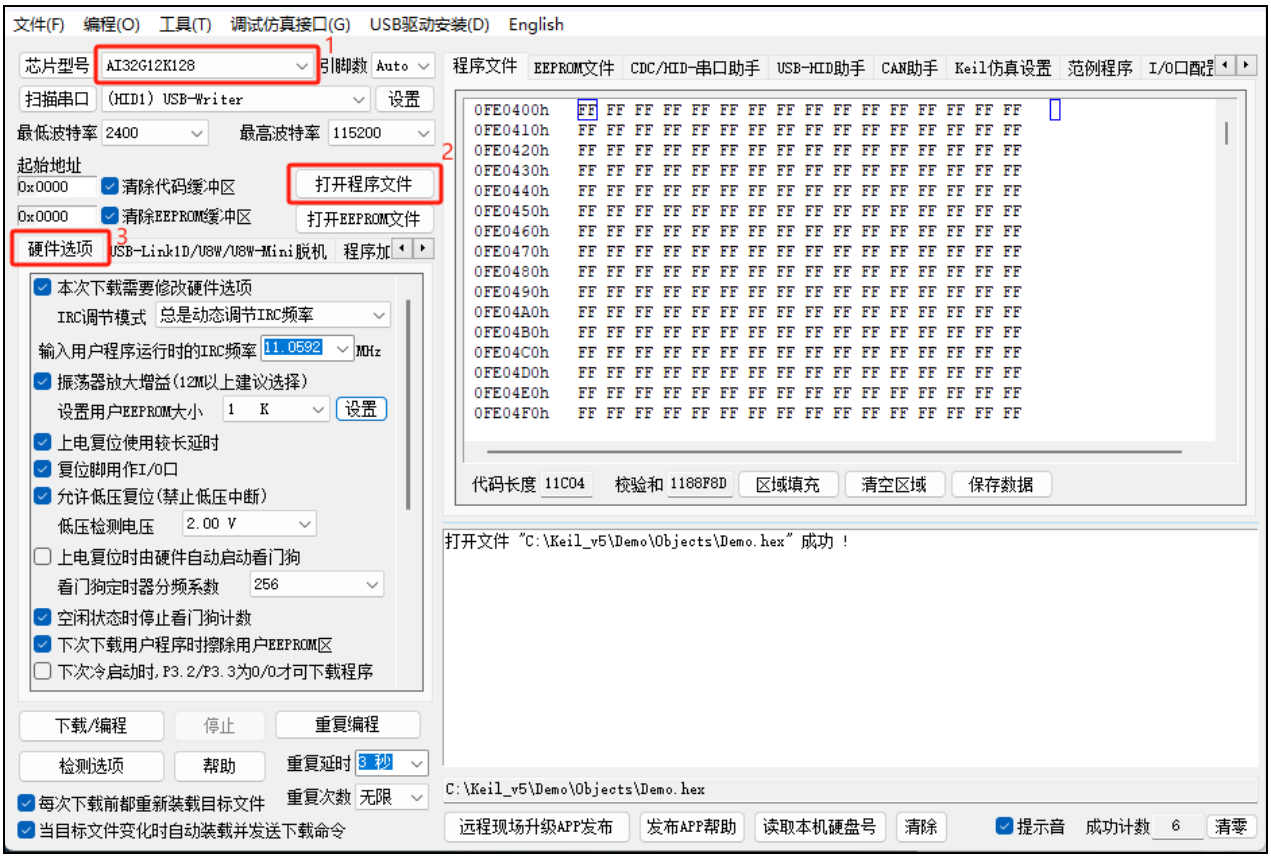
关于界面，用户可以自己进行定制（用户可以自行修改发布项目程序/远程现场升级 App 发布的标题、按钮名称以及帮助信息），同时用户还可以指定目标电脑的硬盘号和目标芯片的 ID 号，指定目标电脑的硬盘号后，便可以控制发布应用程序只能在指定的电脑上运行（防止烧录人员将程序轻易从电脑盗走,如通过网络发走,如通过 U 盘烤走,防不胜防,当然盗走你的电脑那就没办法那,所以脱机下载工具比电脑烧录安全,能限制可烧录芯片数量,让前台文员小姐烧,让老板娘烧都可以），拷贝到其它电脑，应用程序不能运行。同样的，当指定了目标芯片的 ID 号后，那么用户代码只能下载到具有相应 ID 号的目标芯片中（对于一台设备要卖几千万的产品特别有用---坦克,可以发给客户自己升级,不需冒着生命危险跑到战火纷飞的伊拉克升级软件啦），对于 ID 号不一致的其它芯片，不能进行下载编程。

发布项目程序/远程现场升级 App 发布详细的操作步骤如下：

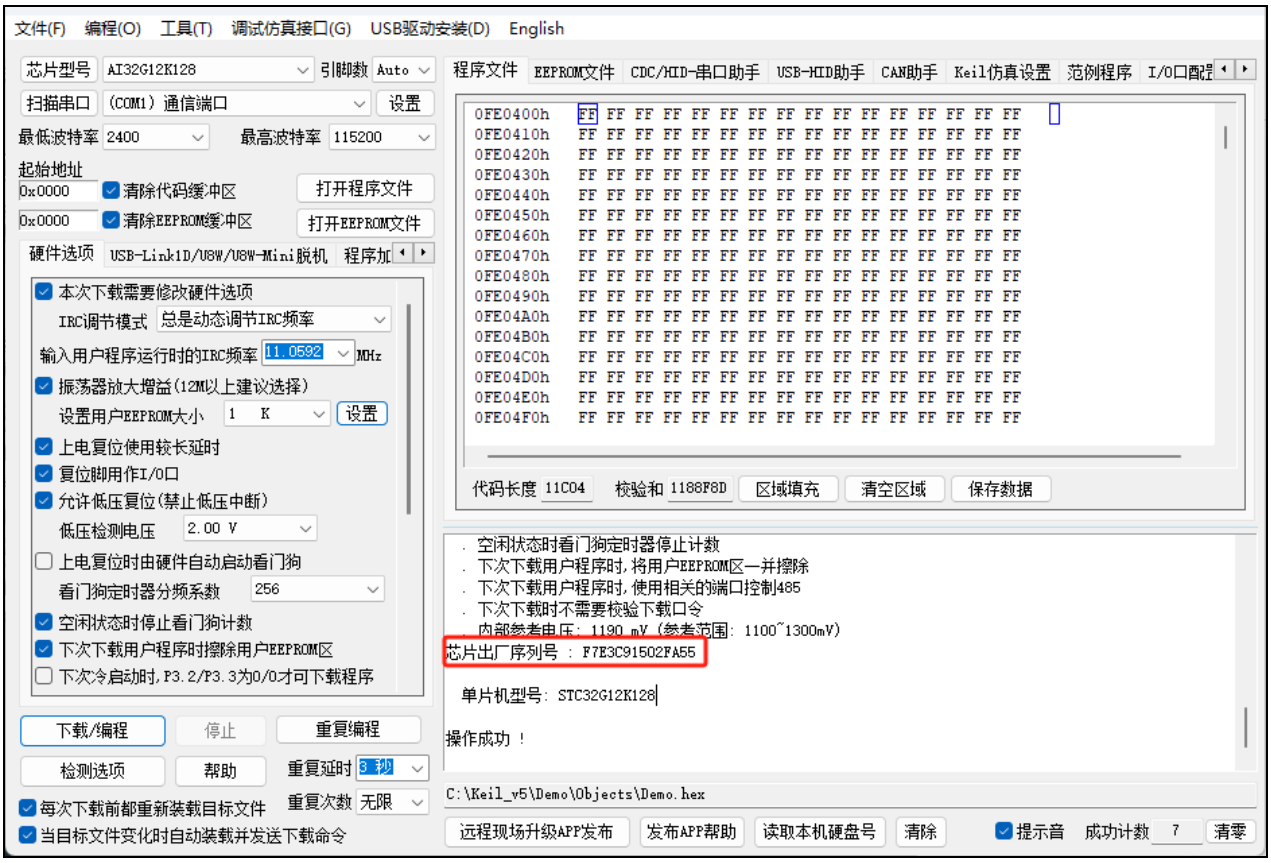
1、首先选择目标芯片的型号

2、打开程序代码文件

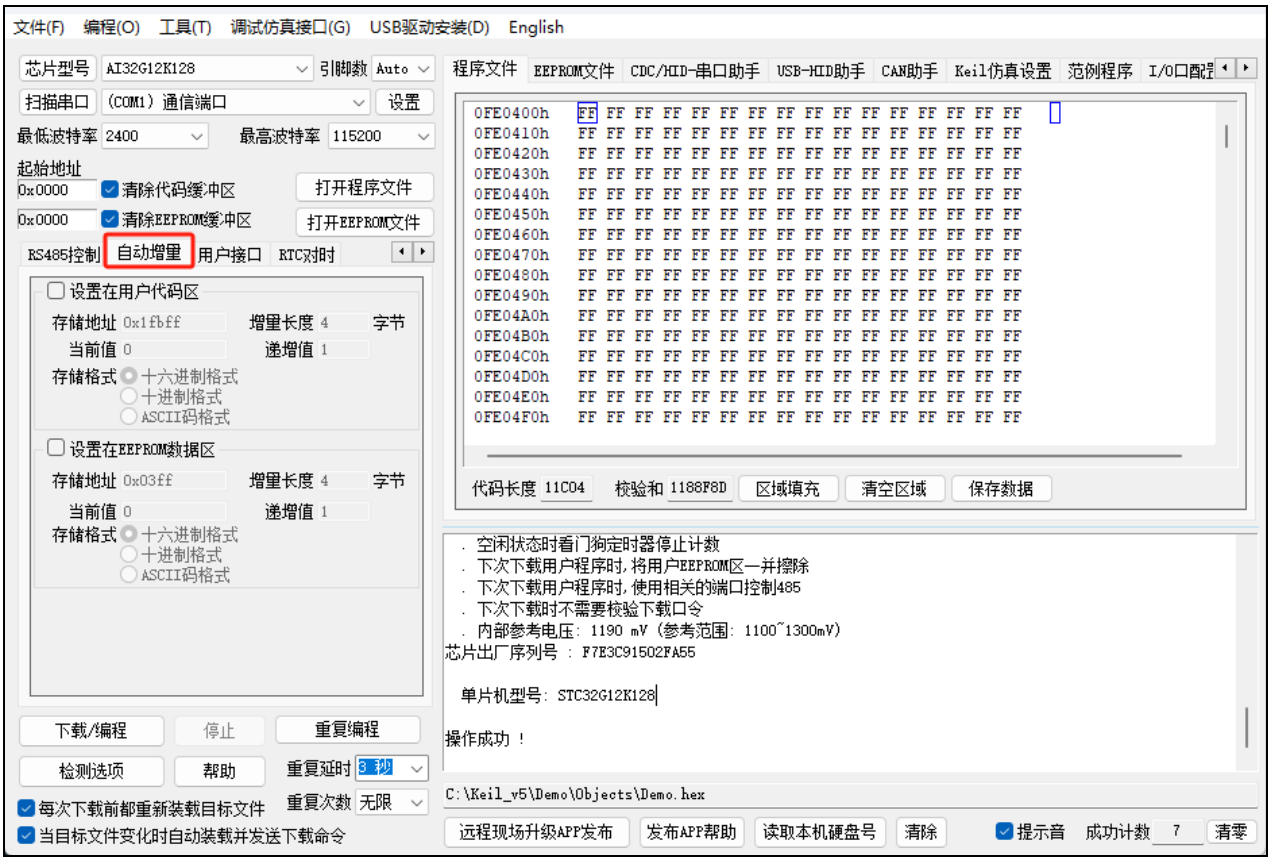
3、设置好相应的硬件选项



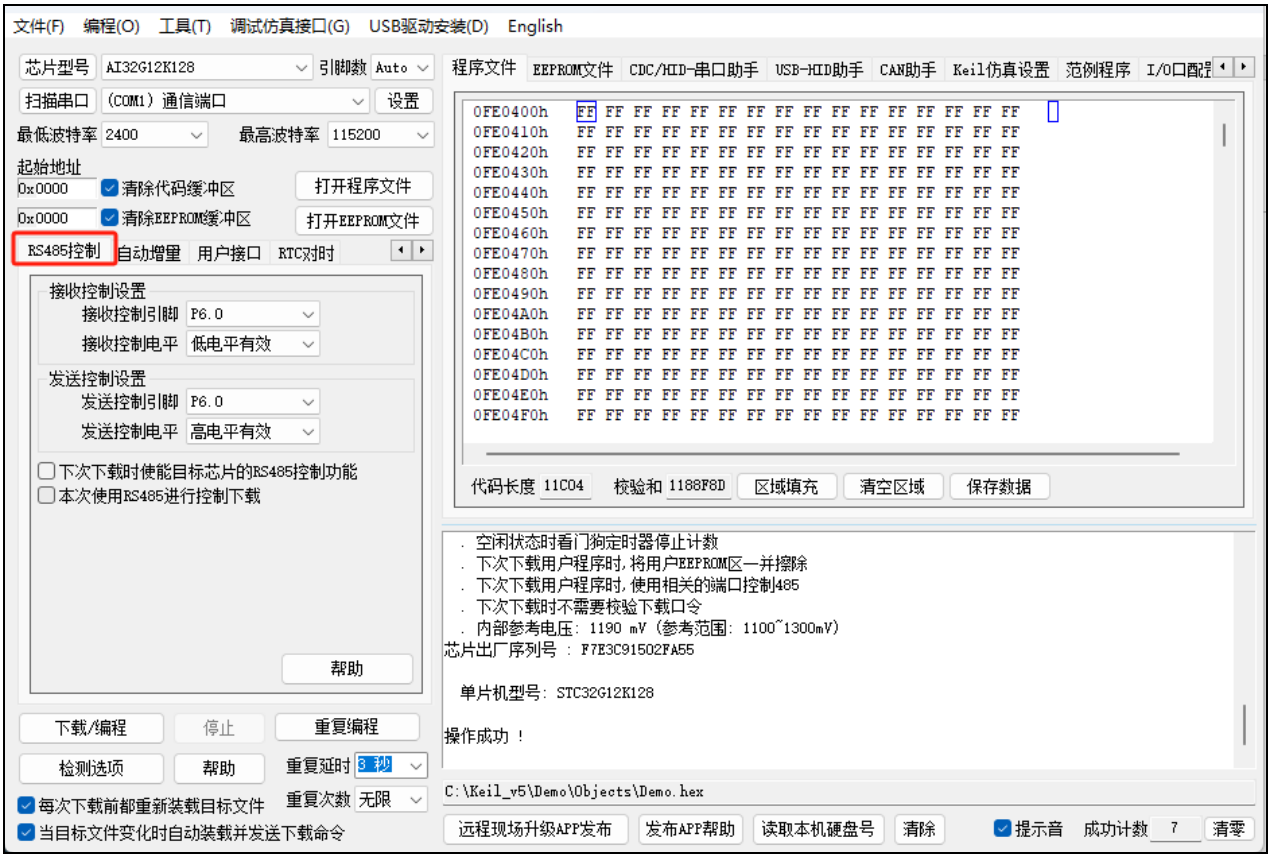
4、试烧一下芯片，并记下目标芯片的 ID 号，如下图所示（如不需要对目标芯片的 ID 号进行校验，可跳过此步）



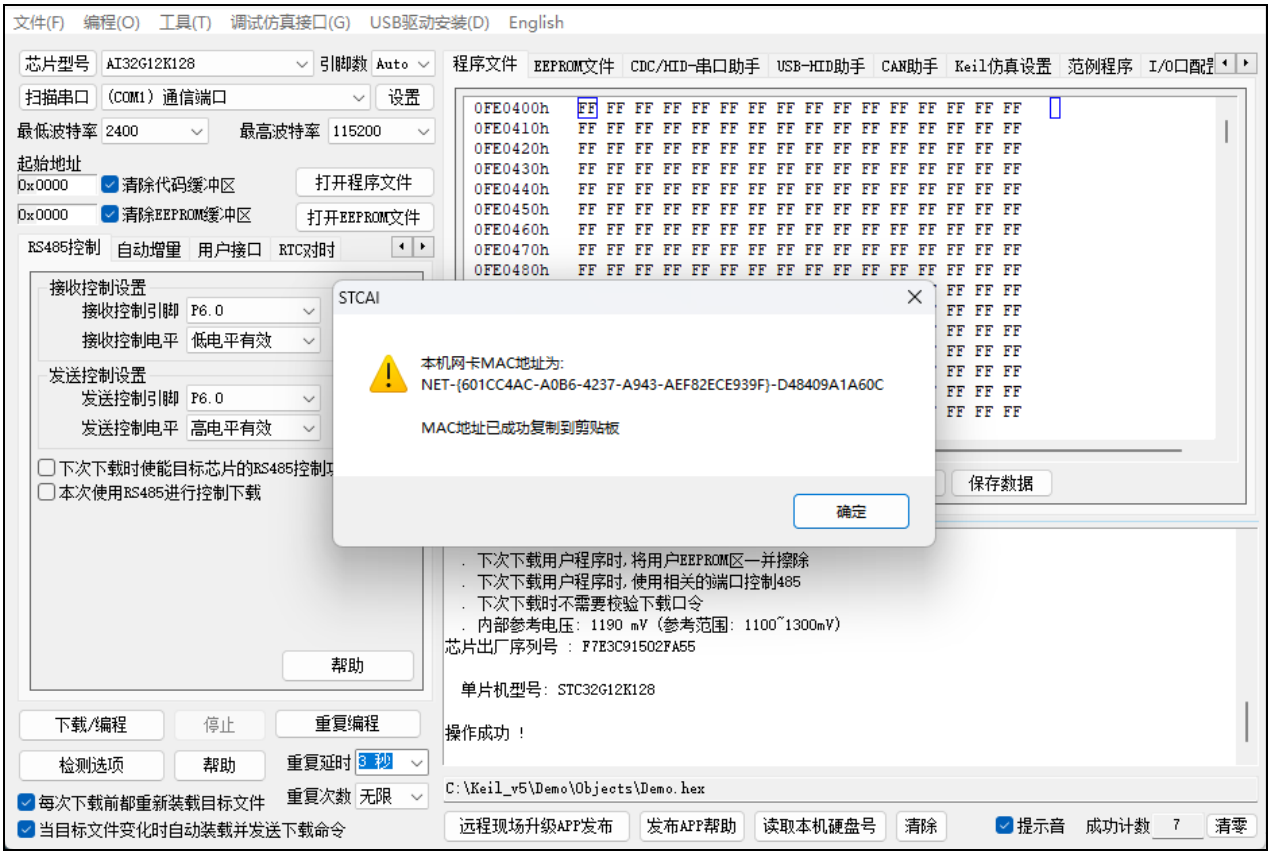
5、设置自动增量（如不需要自动增量，可跳过此步）



6、设置 RS485 控制信息（如不需要 RS485 控制，可跳过此步）



7、点击界面上的“读取本机硬盘号”按钮，并记下目标电脑的硬盘号（如不需要对目标电脑的硬盘号进行校验，可跳过此步）

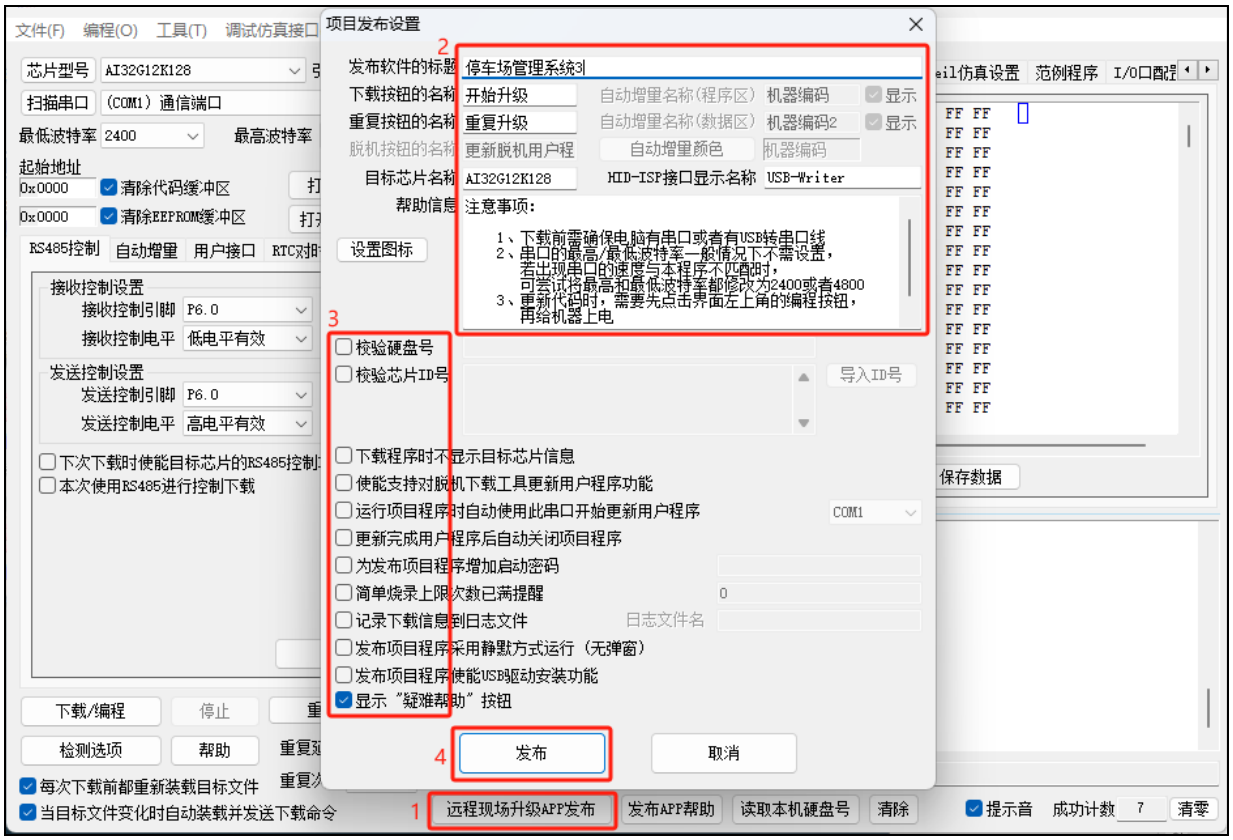


8、点击“远程现场升级 APP 发布”按钮，进入发布应用程序的设置界面。

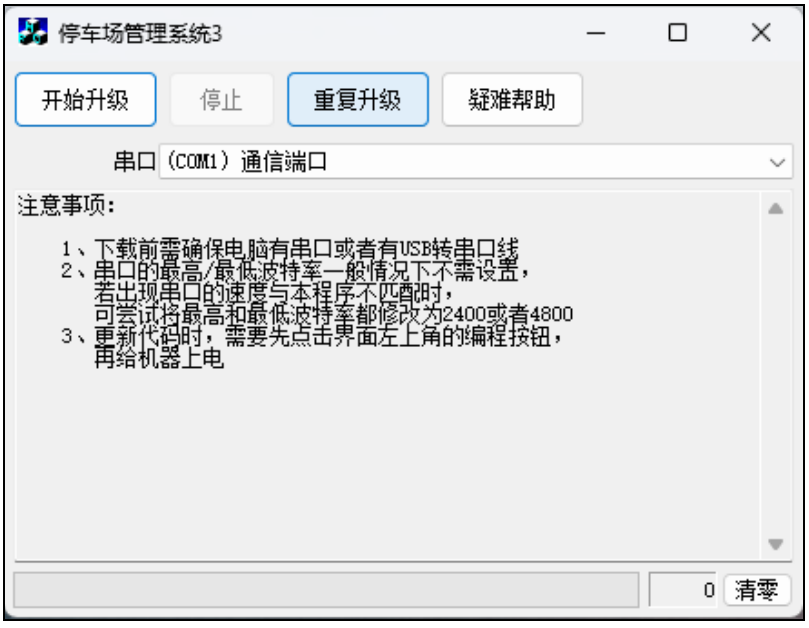
9、根据各自的需要，修改发布软件的标题、下载按钮的名称、重复下载按钮的名称、自动增量的名称以及帮助信息

10、若需要校验目标电脑的硬盘号,则需要勾选上“校验硬盘号”，并在后面的文本框内输入前面所记下的目标电脑的硬盘号

11、若需要校验目标芯片的 ID 号，则需要勾选上“校验芯片 ID 号”，并在后面的文本框内输入前面所记下的目标芯片的 ID 号



12、最后点击发布按钮，将项目发布程序保存，即可得到相应的可执行文件。发布的项目程序打界面如下图



2.33.2　程序加密后传输（防烧录时串口分析出程序）

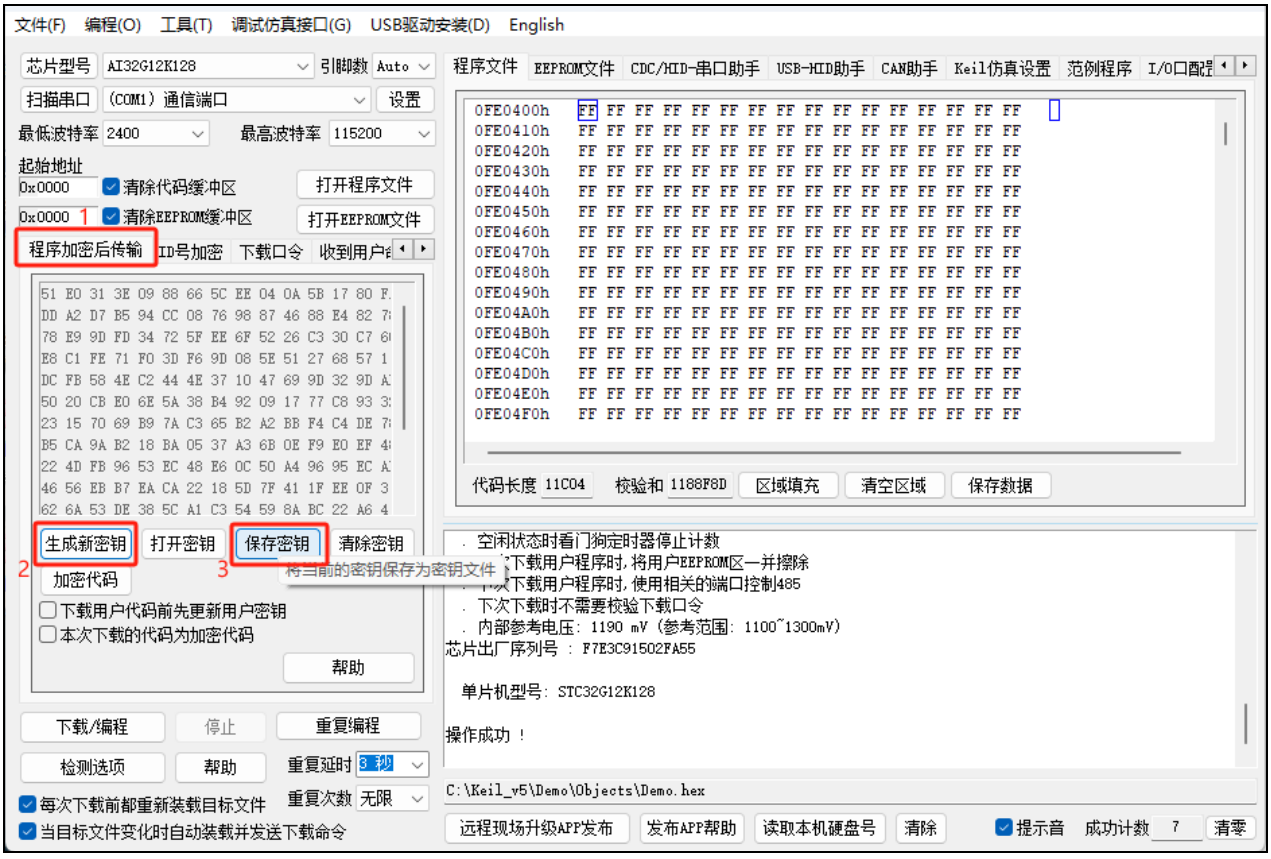
目前，所有的普通串口下载烧录编程都是采用明码通信的（电脑和目标芯片通信时，或脱机下载板和目标芯片通信时），问题：如果烧录人员通过分析下载烧录编程时串口通信的数据，高手是可以在烧录时在串口上引 2 根线出来，通过分析串口通信的数据分析出实际的用户程序代码的。当然用脱机下载板烧程序总比用电脑烧程序强（防止烧录人员将程序轻易从电脑盗走，如通过网络发走，如通过 U 盘拷走，防不胜防，当然盗走你的电脑那就没办法那，所以脱机下载工具比电脑烧录安全，让前台文员小姐烧，让老板娘烧都可以）。即使是全球首创的脱机下载工具，对于要防止天才的不法分子在脱机下载工具烧录的过程中通过分析串口通信的数据分析出实际的用户程序代码，也是没有办法达到要求的，这就需要用到最新的单片机所提供的程序加密后传输功能。

程序加密后传输下载是用户先将程序代码通过自己的一套专用密钥进行加密，然后将加密后的代码再通过串口下载，此时下载传输的是加密文件，通过串口分析出来的是加密后的乱码，如不通过派人潜入你公司盗窃你电脑里面的加密密钥，就无任何价值，便可起到防止在烧录程序时被烧录人员通过监测串口分析出代码的目的。

程序加密后传输功能的使用需要如下的几个步骤：

1、生成并保存新的密钥

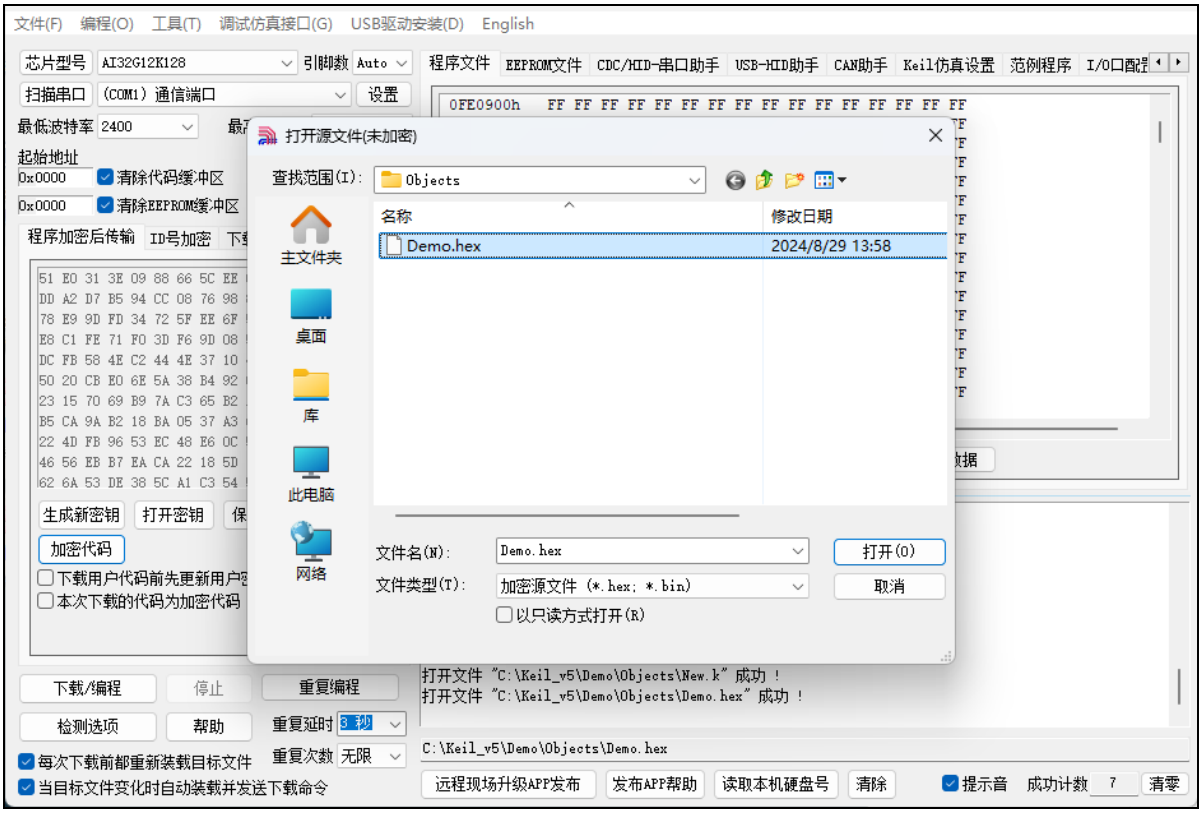
如下图，进入到“程序加密后传输”页面，点击“生成新密钥”按钮，即可在缓冲区显示新生成的 256 字节的密钥。然后点击“保存密钥”按钮，即可将生成的新密钥保存为以“.K”为扩展名的密钥文件（**注意：这个密钥文件一定要保存好，以后发布的代码文件都需要使用这个密钥加密，而且这个密钥的生成是非重复的，即任何时候都不可能生成两个完全相同的密钥，所以一旦密钥文件丢失将无法重新获得**）。例如我们将密钥保存为“New.k”。

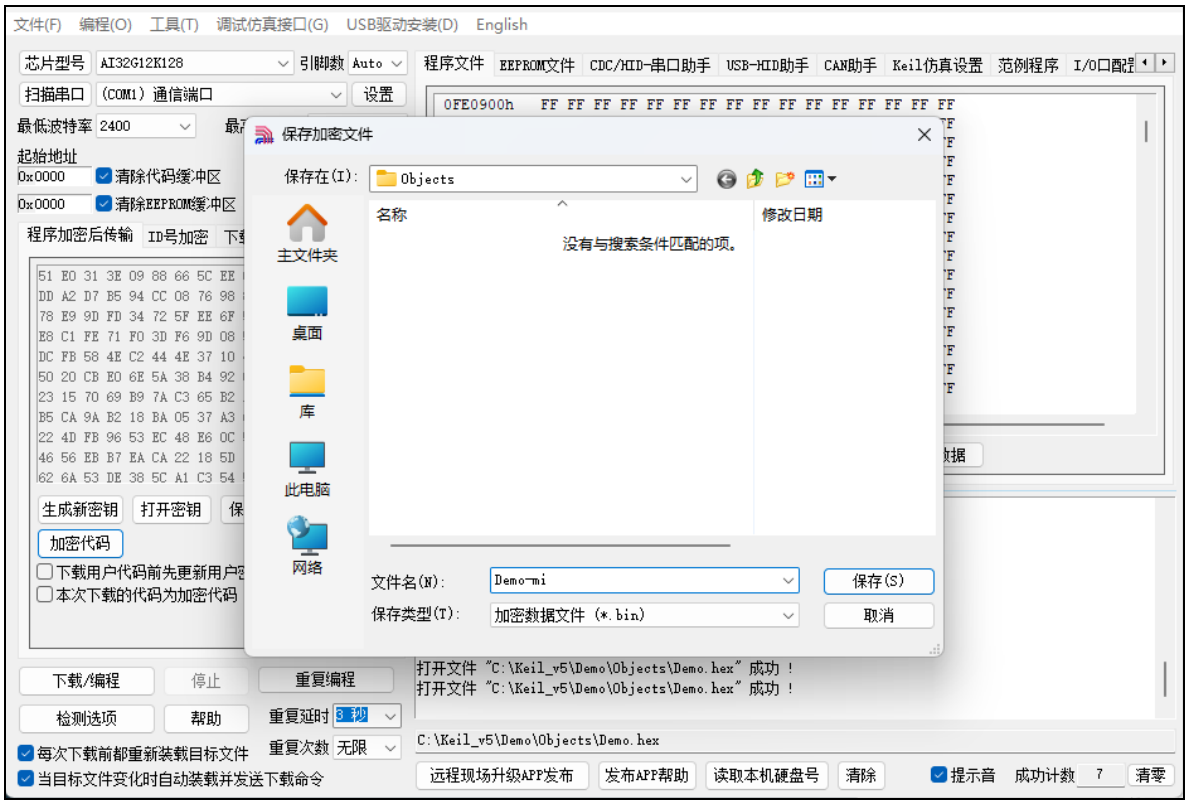


2、对代码文件加密

加密文件前，需要先打开我们自己的密钥。若缓冲区中存放的已经是我们的密钥，则不要再打开。如下图，在“程序加密后传输”页面中点击“打开密钥”按钮，打开我们之前保存的密钥文件，例如“New.k”，然后返回到“程序加密后传输”页面中点击“加密代码”按钮，如下图所示，首先会弹出“打开源文件（未加密）”的对话框，此时选择的是原始的未加密的代码文件

点击打开按钮后，马上有会弹出一个类似的对话框，但此时是对加密后的文件进行保存的对话框。如下图所示，点击保存按钮即可保存加密后的文件。



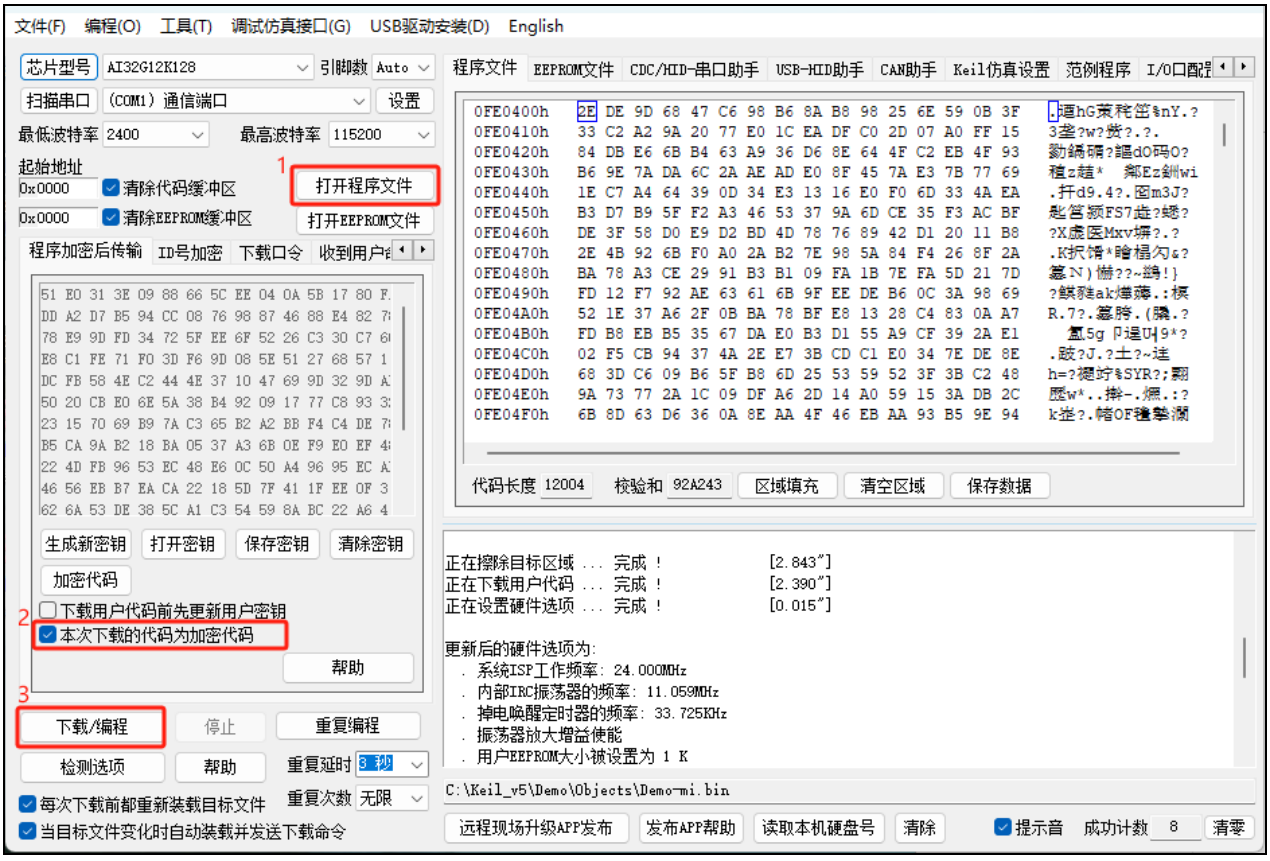


3、将用户密钥更新到目标芯片中

更新密钥前，需要先打开我们自己的密钥。若缓冲区中存放的已经是我们的密钥，则不要再打开。如下图，在“自定义加密下载”页面中点击“打开密钥”按钮，打开我们之前保存的密钥文件，例如“New.k”。密钥打开后，如下图所示，勾选上“下载用户代码前先更新用户密钥”选项和“本次下载的代码为加密代码”的选项，然后打开我们之前加密过后的文件，打开后点击界面左下角的“下载/编程”按钮，按正常方式对目标芯片下载完成即可更新用户密钥。

4、加密更新用户代码

密钥更新成功后，目标芯片便具有接收加密代码并还原的功能。此时若需要再次升级/更新代码，则只需要参考第二步的方法，将目标代码进行加密，然后如下图

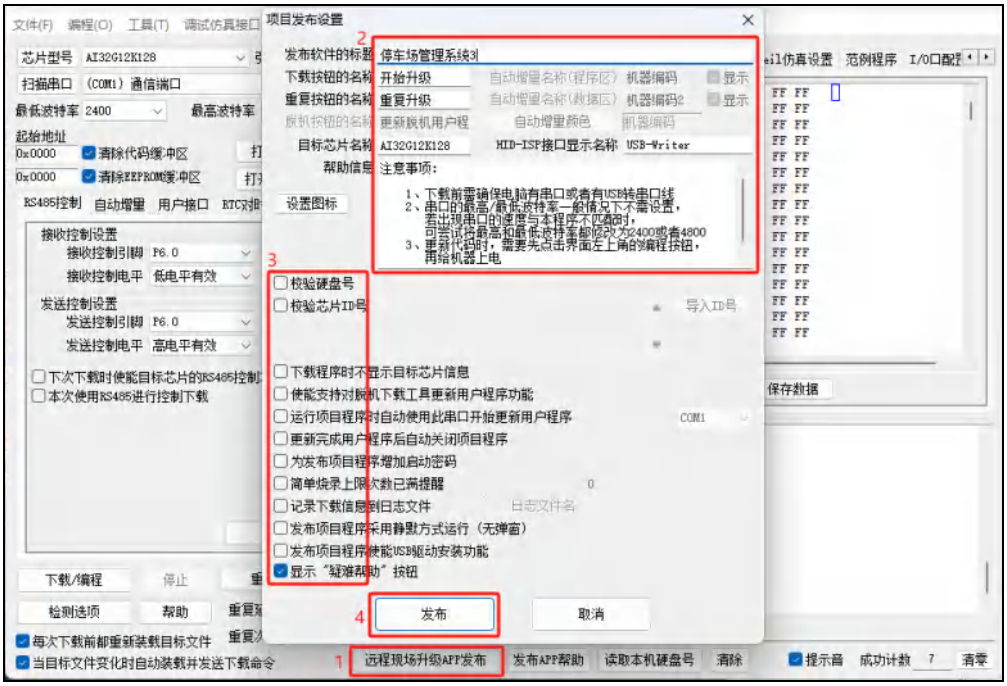


对于一片新的单片机，可将步骤 3 和步骤 4 合并完成，即将密钥更新到目标单片机的同时也可将加密后的代码一并下载到单片机中，若已经执行过步骤 3（即已经将密钥更新到目标芯片中了），则后续的代码更新就只需要按照步骤 4，只需要在“程序加密后传输”页面中选择“本次下载的代码为加密代码”的选项（“下载用户代码前先更新用户密钥”选项不需要选了），然后打开我们之前加过密后的文件，打开后点击界面左下角的“下载/编程”按钮，按正常方式对目标芯片下载即可完成用用户自己专用的加密文件更新用户代码的目的（防止在烧录程序时被烧录人员通过监测串口分析出代码的目的）。

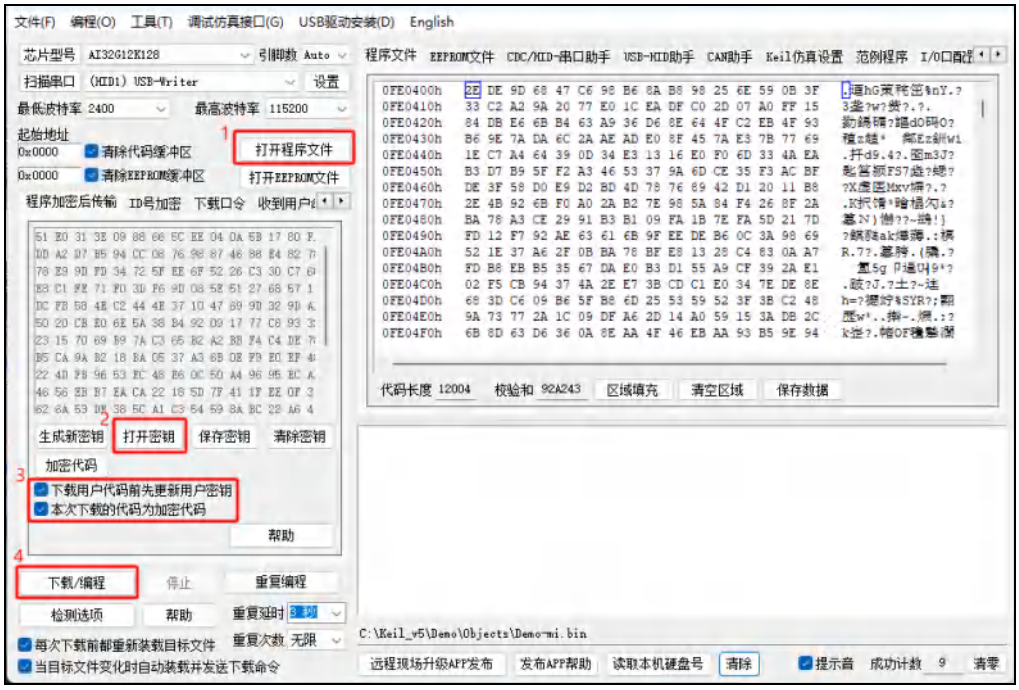
2.33.3 发布项目程序/远程现场升级 App 发布+程序加密后传输结合使用

发布项目程序/远程现场升级 App 发布与程序加密后传输两项新的特殊功能可以结合在一起使用。首先程序加密后传输可以确保用户代码在烧录编程时串口通信传输过程当中的保密性，而发布项目程序/远程现场升级 App 发布可实现让最终使用者远程升级功能（方案公司的人员不需要亲自到场）。所以两项功能结合起来使用，非常适用于方案公司/生产商在软件需要更新时，让最终使用者自己对终端产品进行软件更新的目的，又确保现场烧录人员无法通过串口分析出有用程序，强烈建议方案公司使用。

发布项目程序/远程现场升级 App 发布可参考 5.16.1 章节步骤，示意图如下：



程序加密后传输可参考 5.16.2 章节步骤，示意图如下：

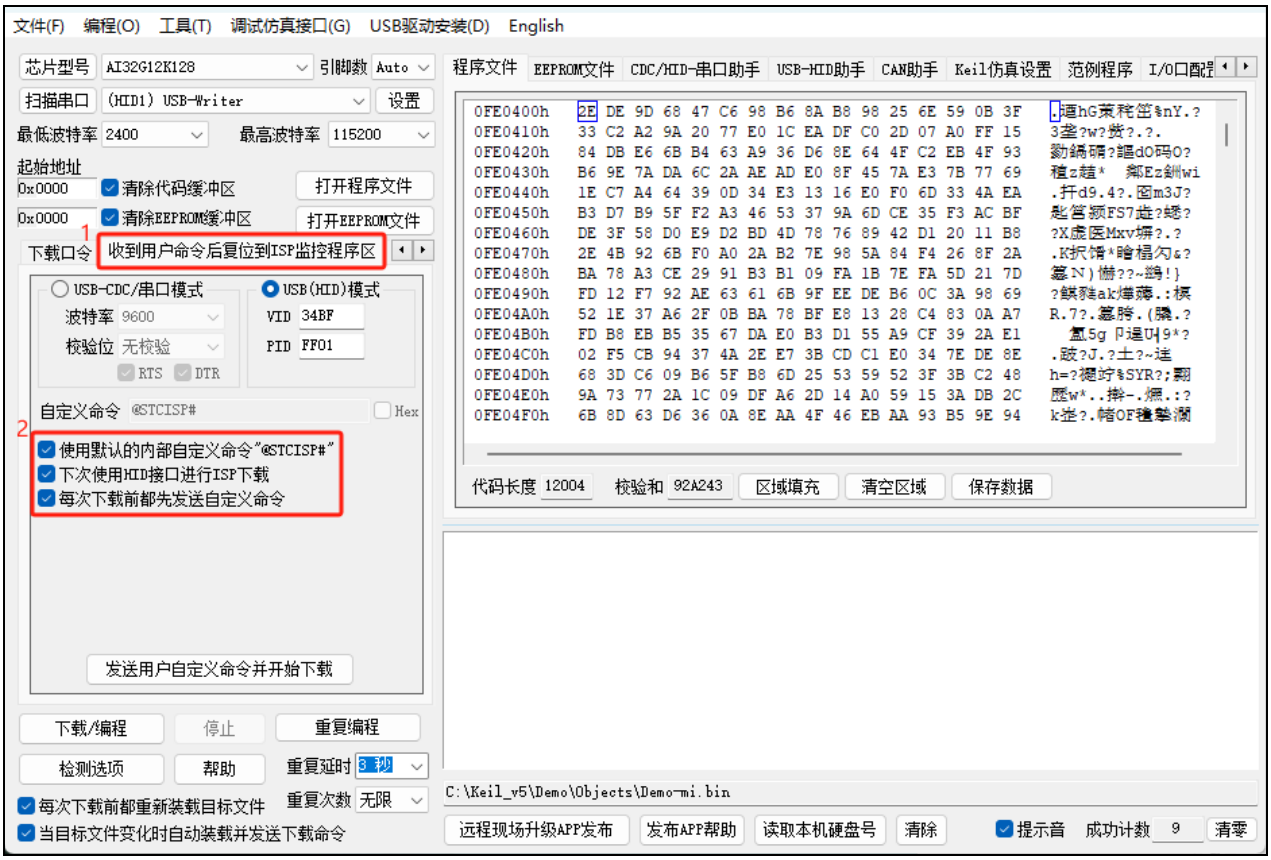


2.33.4　用户自定义下载（实现不停电下载）

将用户的目标程序下载到单片机是通过执行单片机内部的 ISP 系统代码和上位机进行串口或者 USB 通讯来实现的。但单片机内部的 ISP 系统代码只有在每次重新停电再上电时才会被执行，这就要求用户每次需要对目标单片机更新程序时就必须重新上电，而 USB 模式的 ISP，处理需要重新对目标芯片上电外，还需要在上电时将 P3.2 口下拉到 GND。对于处于开发阶段的项目，需要频繁的修改代码、更新代码，每次下载都需要重新上电会导致操作非常麻烦。

单片机在硬件设计时，增加了一个软复位寄存器（IAP_CONTR），让用户可以通过设置此寄存器来决定 CPU 复位后重新执行用户代码还是复位到 ISP 区执行 ISP 系统代码。当向 IAP_CONTR 寄存器写入 0x20 时，CPU 复位后重新执行用户代码；当向 IAP_CONTR 寄存器写入 0x60 时，CPU 复位后复位到 ISP 区执行 ISP 系统代码。

要实现不停电进行 ISP 下载，用户可以在程序中设计一段代码，例如检测一个特殊的按键、或者监控串口等待一个特殊的串口命令，当检测到满足下载条件时，就通过软件触发软复位寄存器复位到 ISP 区执行 ISP 系统代码，从而实现不停电 ISP 下载。当触发条件是外部按键时，则在用户代码中实时监控按键状态即可。若要实现 AIapp-ISP 软件和用户触发软复位完全同步，则需要使用 AIapp-ISP 软件中所提供的“收到用户命令后复位到 ISP 监控程序区”这个功能。



实现不停电 ISP 下载的步骤如下：

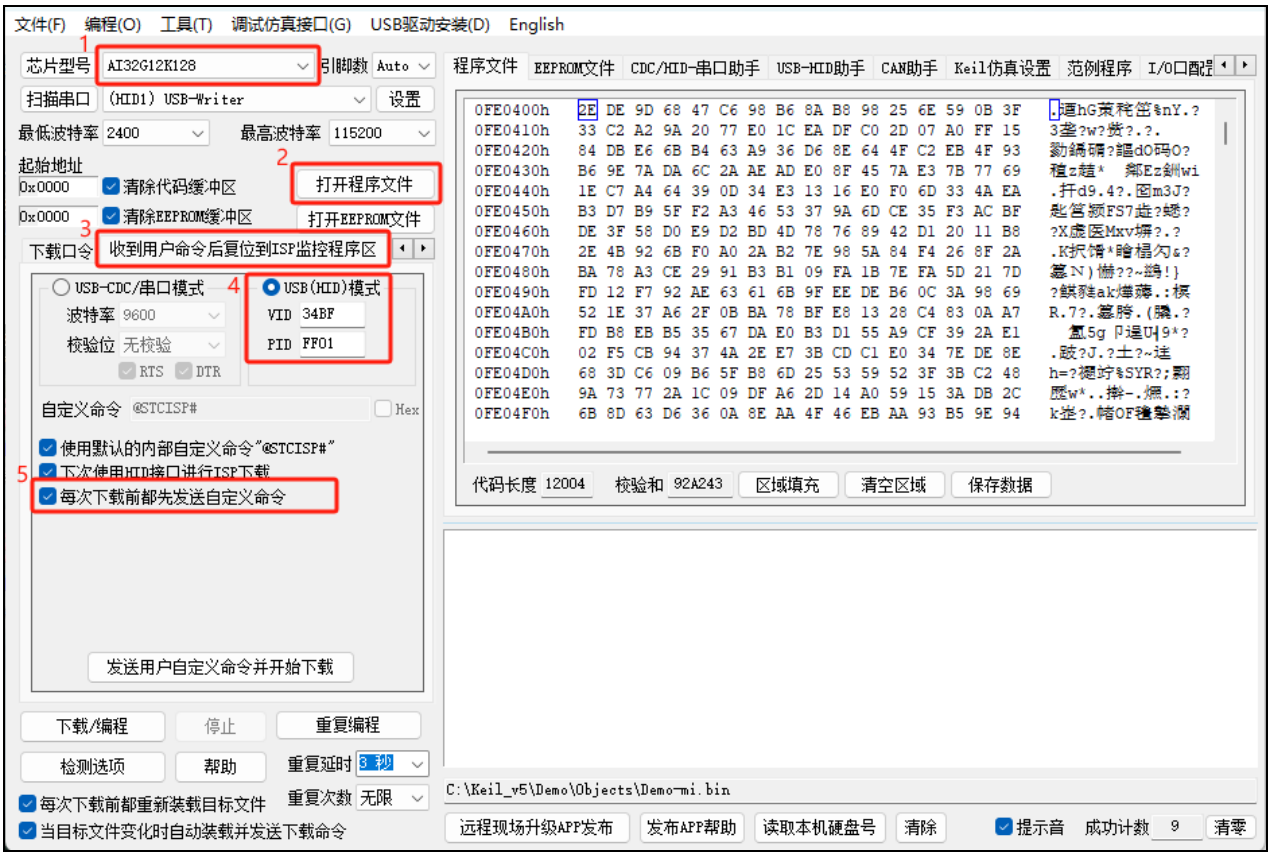
1、编写用户代码，并在用户代码中添加串口命令监控程序

（参考代码如下，测试单片机型号为 AI8H8K64U）

```
#include "AI8H.h"

#define    FOSC        11059200UL
#define    BAUD        (65536 - (FOSC/115200+2)/4)        //加 2 操作是为了让 Keil 编译器
                                                          //自动实现四舍五入运算

char code *STCISPCMD = "@STCISP#";          //自定义下载命令
char index;

void uart_isr()   interrupt 4
{
    char dat;

    if (TI)
    {
        TI = 0;
    }

    if (RI)
    {
        RI = 0;
        dat  = SBUF;                        //接收串口数据

        if (dat == STCISPCMD[index])        //判断接收的数据和当前的命令字符是否匹配
        {
            index++;                        //若匹配则索引+1
            if (STCISPCMD[index] == '\0')   //判断命令是否配完成
                IAP_CONTR = 0x60;           //若匹配完成则软复位到 ISP
        }
        else
        {
            index = 0;                      //若不匹配,则需要从头开始
            if (dat == STCISPCMD[index])
                index++;
        }
    }
}

void main()
{
    P0M0 = 0x00; P0M1 = 0x00;
    P1M0 = 0x00; P1M1 = 0x00;
```

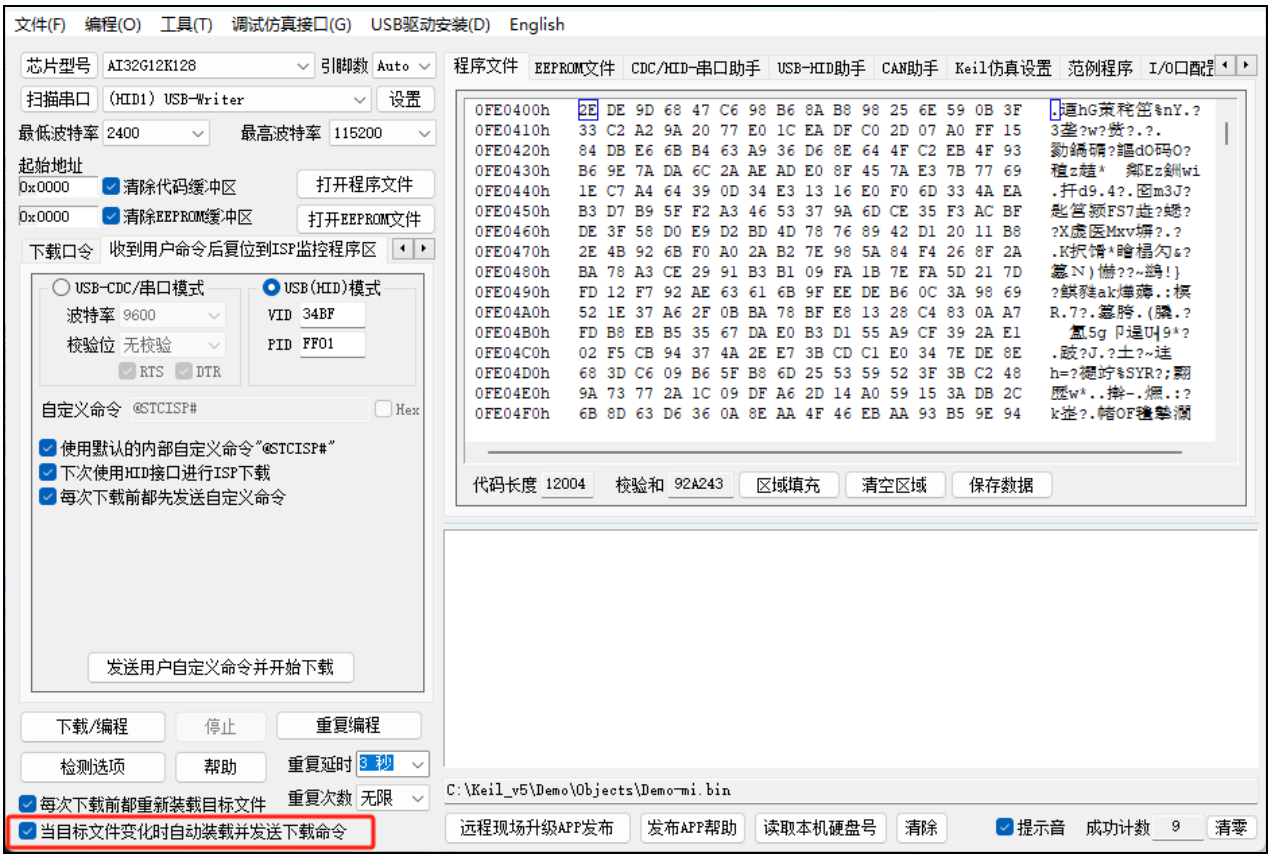
```
    P2M0 = 0x00; P2M1 = 0x00;
    P3M0 = 0x00; P3M1 = 0x00;

    SCON = 0x50;                    //串口初始化
    AUXR = 0x40;
    TMOD = 0x00;
    TH1 = BAUD >> 8;
    TL1 = BAUD;
    TR1 = 1;
    ES = 1;
    EA = 1;

    index = 0;                      //初始化命令

    while (1);
}
```

2、按下图所示的步骤进行设置自定义下载命令（范例使用默认命令“@STCISP#”）



3、第一次下载时需要对目标单片机重新上电，之后的每次更新只需要点击下载软件中的“下载/编程”按钮，下载软件自动将下载命令发送给目标单片机，目标单片机接收到命令后自动复位到系统 ISP 区，即可实现不停电更新用户代码。

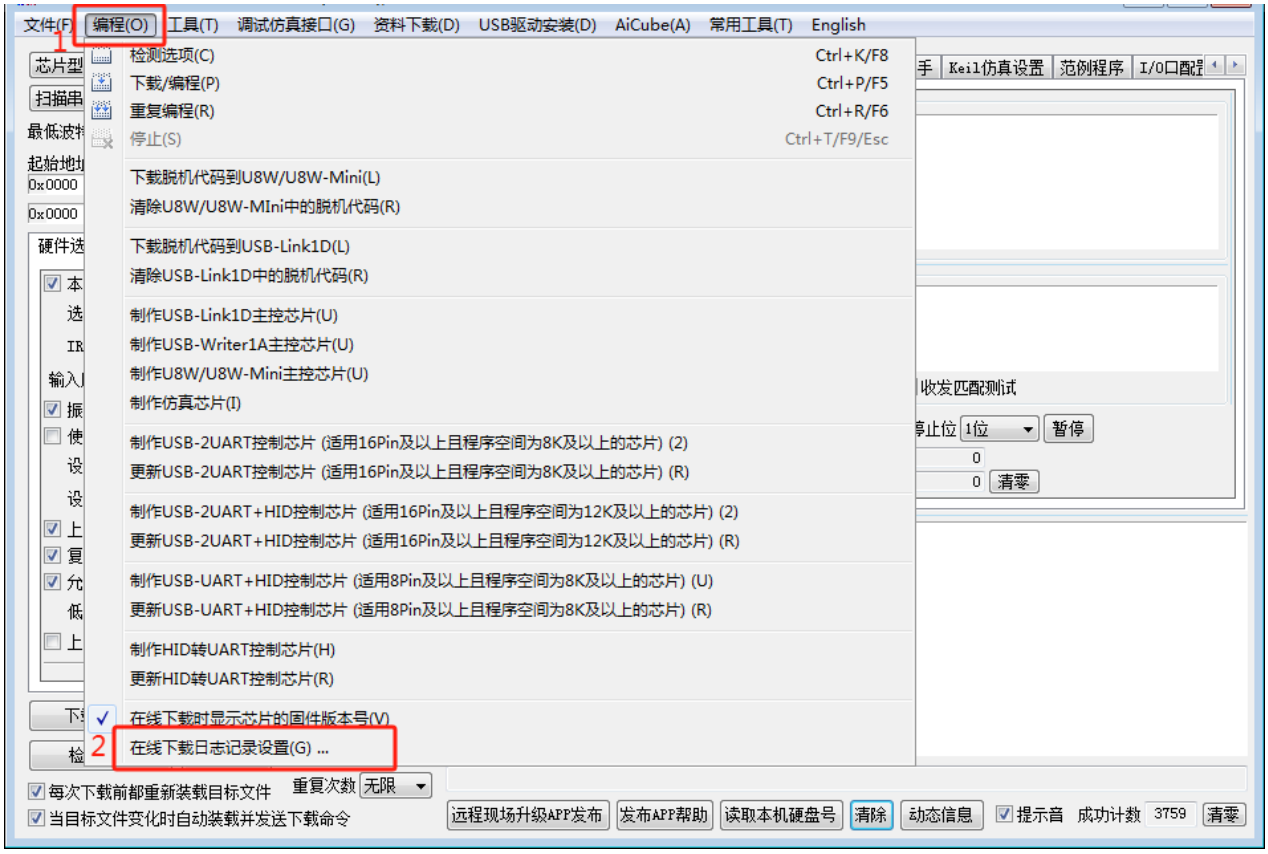
4、AIapp-ISP 还可实现项目开发阶段，完全自动下载功能，即当下载软件侦测到目标代码被更新了，就会自动发送下载命令。要实现这个功能只需要勾选下图中的选项即可



2.33.5 如何简单的控制下载次数，通过 ID 号来限制实际可以下载的 MCU 数量

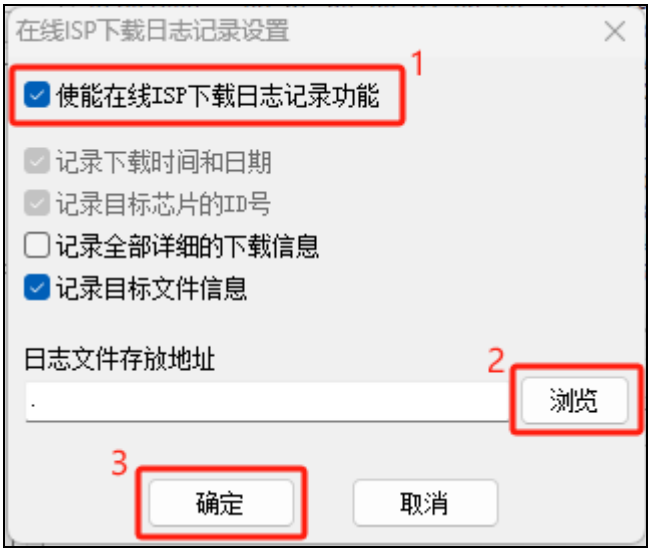
————————下载日志+发布项目高级应用

第一步、打开（使能）下载日志记录功能



1、打开“编程”菜单

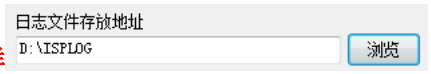
2、点击“在线下载日志记录设置”，打开下面窗口



1、勾选“使能在线 ISP 下载日志记录功能”

2、点击“浏览”按钮选择日志文件存放目录

温馨提示：当日志文件存放目录为默认的“ . ”时，表示下载日志文件存放在 ISP 下载软件所在的目录下；也可以像这样



，指定专用的存放目录

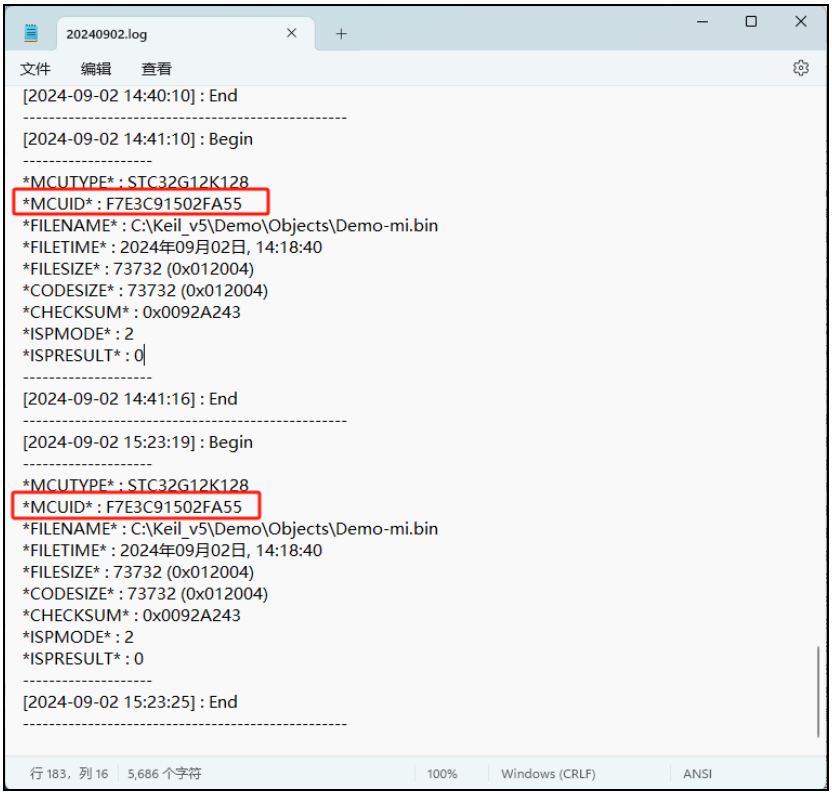
3、点击“确定”进行确认

设置完成后，接下来所有的 ISP 在线下载的下载信息都会自动记录到文件中，日志文件的文件名为当天的日期，即 ISP 下载时的年月日，扩展名为 log（**如：2025 年 5 月 20 日生成的日志为 20250520.log**）

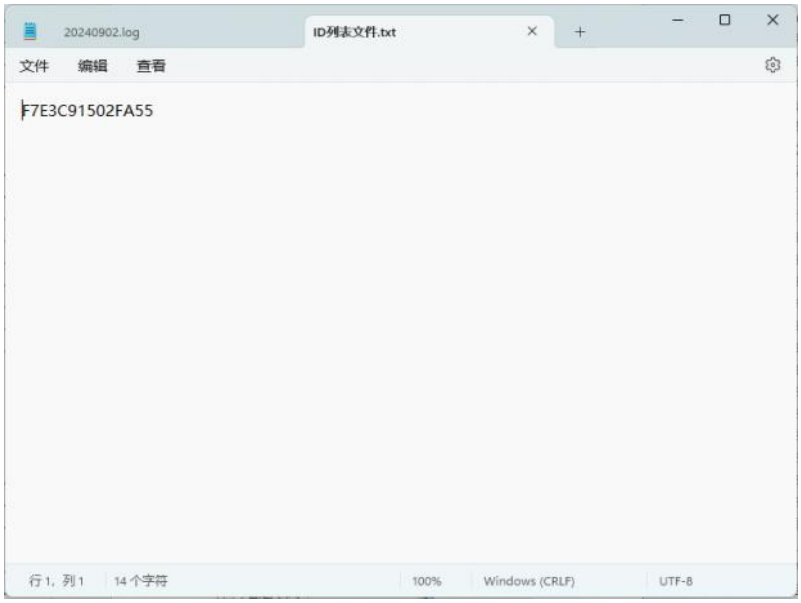
第二步、从下载日志文件中导出 ID 号列表到列表文件

（注：Ver6.92D 版本及之后的 ISP 软件可自动从列表中导入 ID 号，如需自动导入可跳过此步）

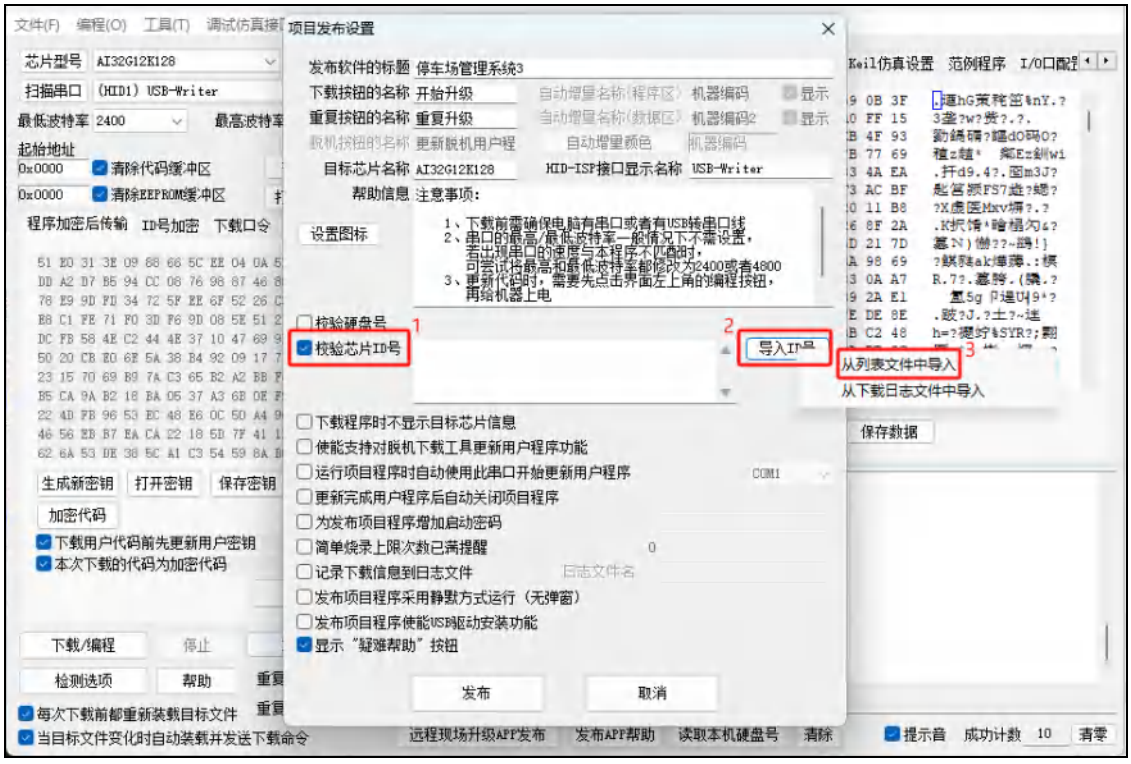
1、从日志文件存放目录中打开目标日期的日志文件（例如打开 2024 年 09 月 02 日的日志，则打开日志文件存放目录中的“20240902.log”）。日志记录格式如下图：



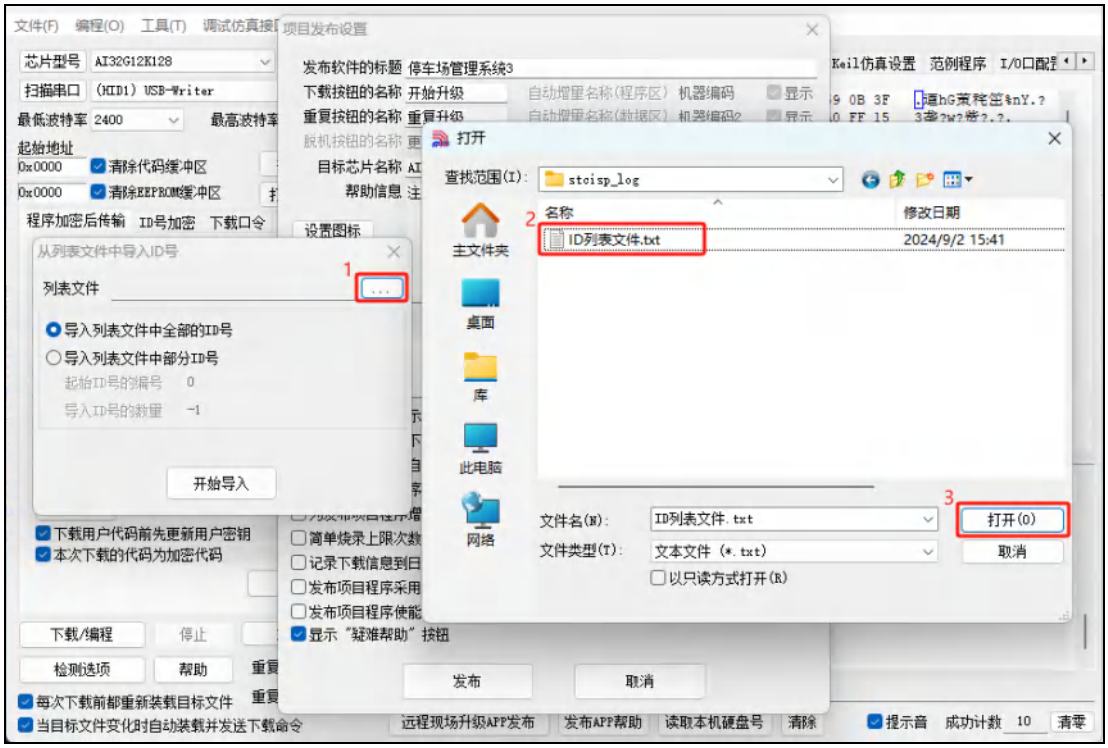
2、从日志文件中复制 ID 号到一个列表文件中，如下图



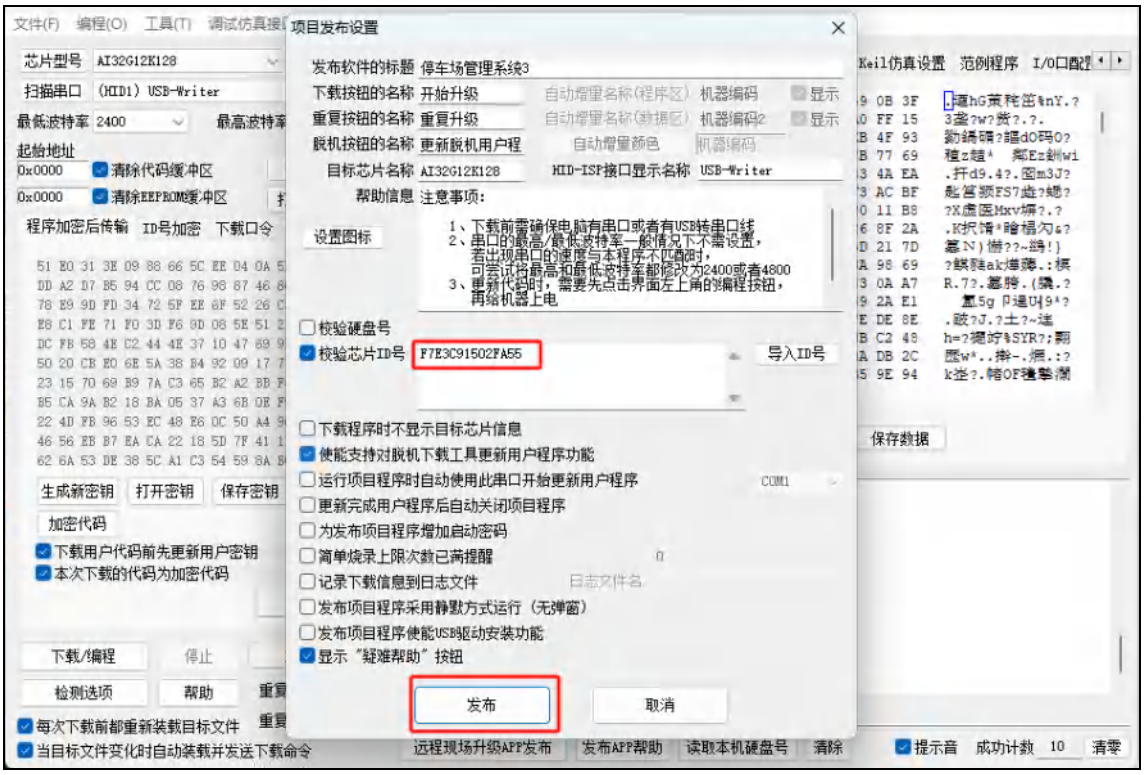
第三步、发布项目程序/远程现场升级 App 发布时导入列表文件中的 ID（如果需要从日志中自动导入，可跳到第四步）



1、点击 AIapp-ISP 下载界面中的“发布项目程序”按钮
2、勾选“校验芯片 ID 号”
3、点击“导入 ID 号”
4、选择“从列表文件中导入”
5、打开上一步导出的列表文件

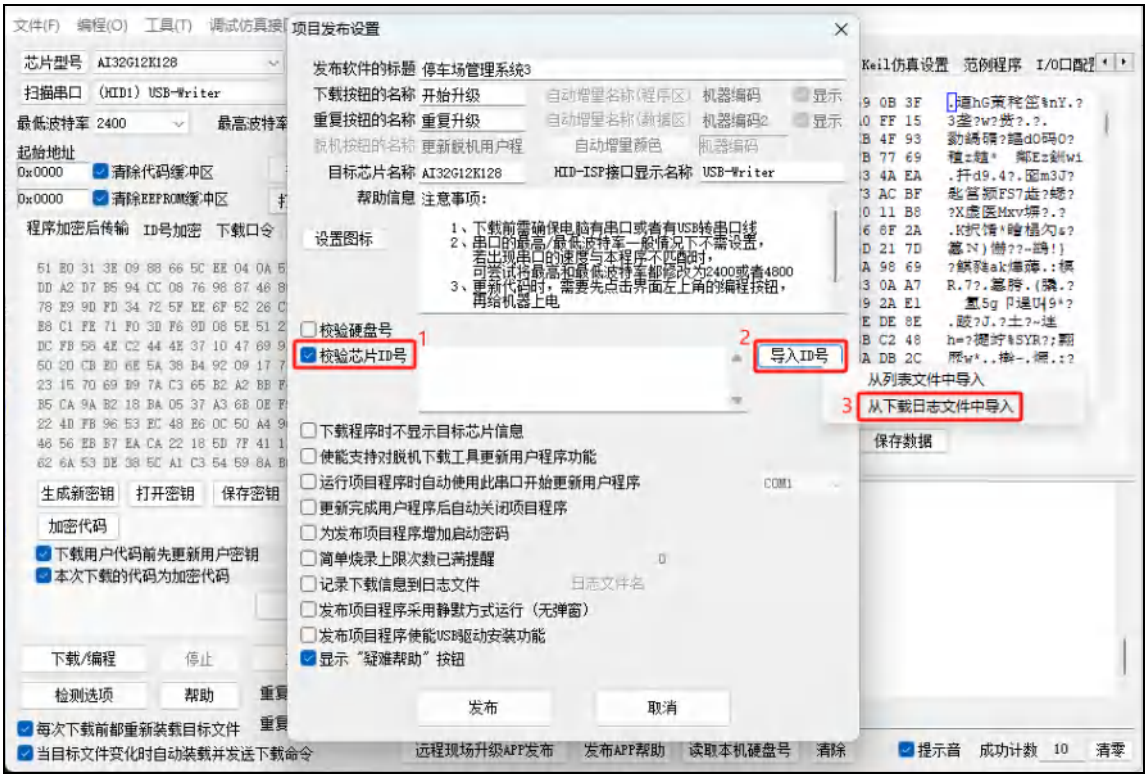

项目发布设置
发布软件的标题 停车场管理系统3
下载按钮的名称 开始升级
重复按钮的名称 重复升级
目标芯片名称
帮助信息
设置图标
发布
取消
从列表文件中导入ID号
列表文件
导入列表文件中全部的ID号
导入列表文件中部分ID号
起始ID号的编号 0
导入ID号的数量 -1
开始导入
1
打开
查找范围(I): stcisp_log
名称
修改日期
ID列表文件.txt
2024/9/2 15:41
2
主文件夹
桌面
库
此电脑
网络
文件名(N): ID列表文件.txt
文件类型(T): 文本文件 (*.txt)
以只读方式打开(R)
打开(O)
取消
3
下载用户代码前先更新用户密钥
本次下载的代码为加密代码
下载/编程
停止
检测选项
帮助
每次下载前都重新装载目标文件
当目标文件变化时自动装载并发送下载命令
显示"疑难帮助"按钮
远程现场升级APP发布
发布APP帮助
读取本机硬盘号
清除
提示音
成功计数
清零

6、列表导入成功后，在下面的 ID 号文本框内会显示刚刚导入的全部 ID 号

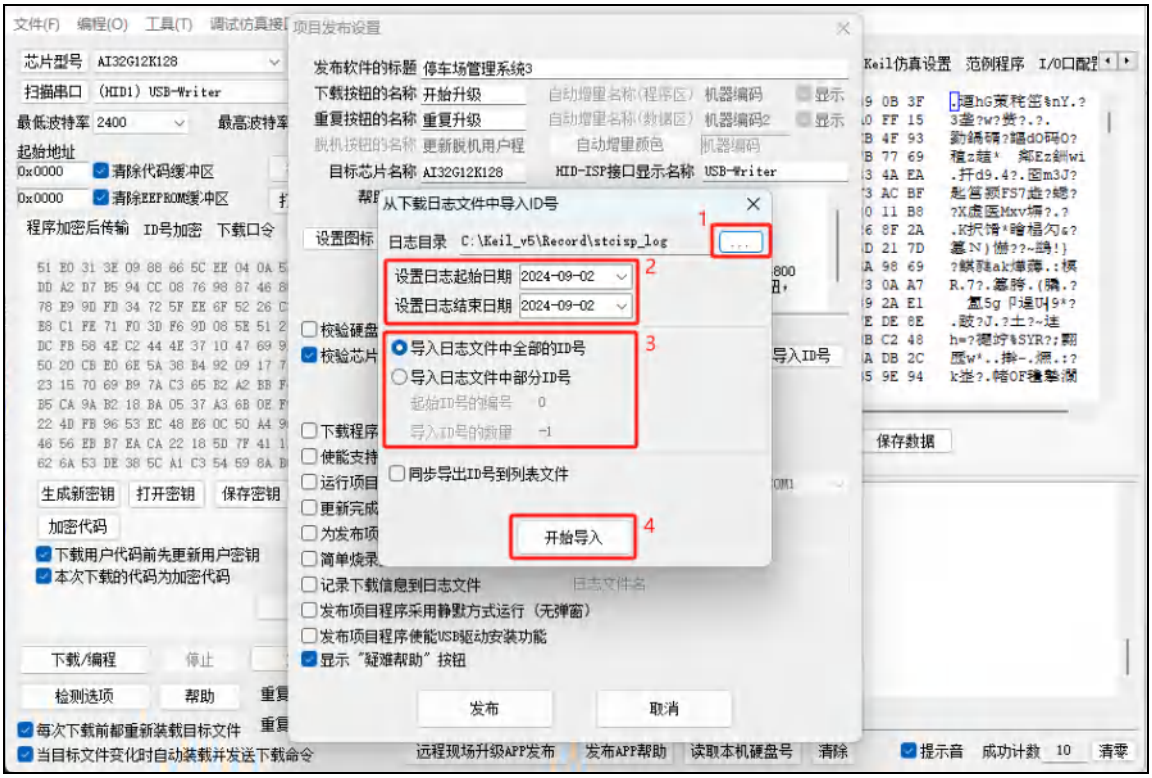


7、最后点击“发布”按钮即可发布项目。

第四步、发布项目程序时从日志文件中自动导入 ID



1、点击 AIapp-ISP 下载界面中的“发布项目程序”按钮
2、勾选“校验芯片 ID 号”
3、点击“导入 ID 号”
4、选择“从下载日志文件中导入”

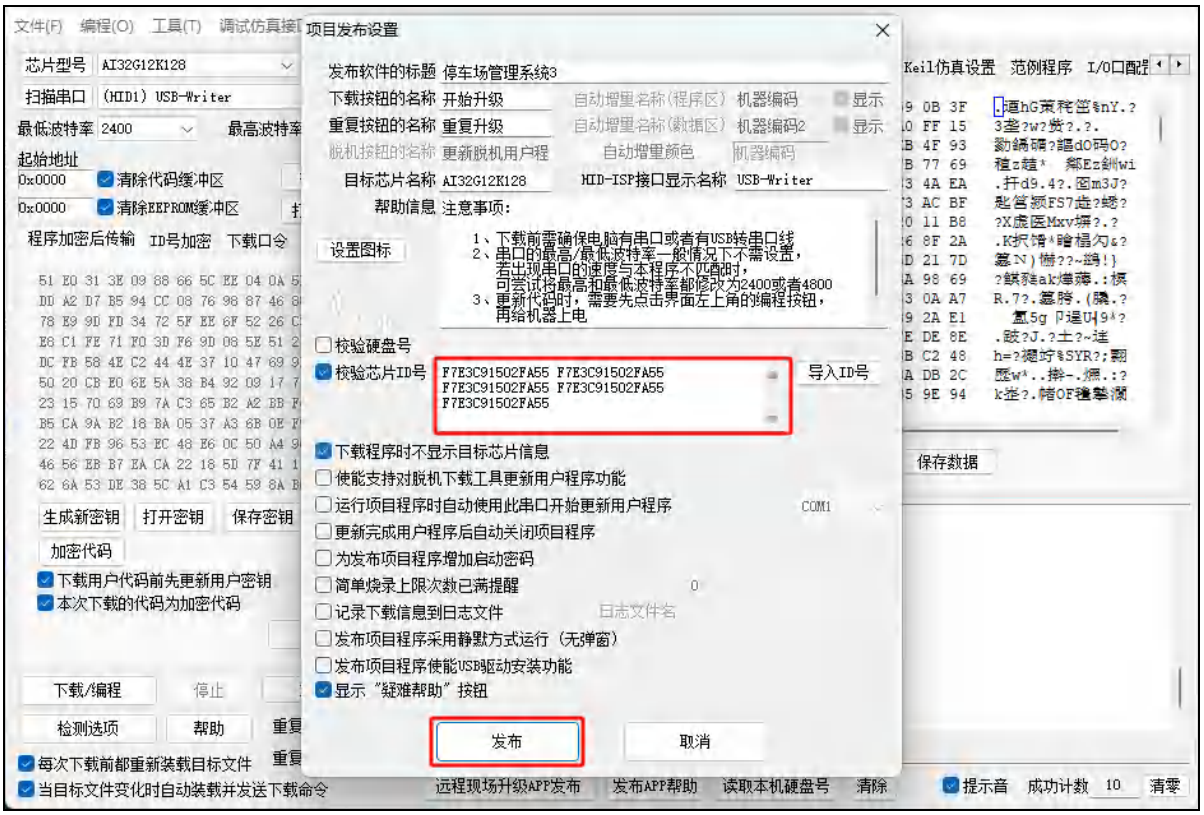


5、打开日志保存目录

6、设置需要导入日志的起始时间和结束时间

7、选择需要导入的 ID 号的序号

8、列表导入成功后，在下面的 ID 号文本框内会显示刚刚导入的全部 ID 号



9、最后点击“发布”按钮即可发布项目。

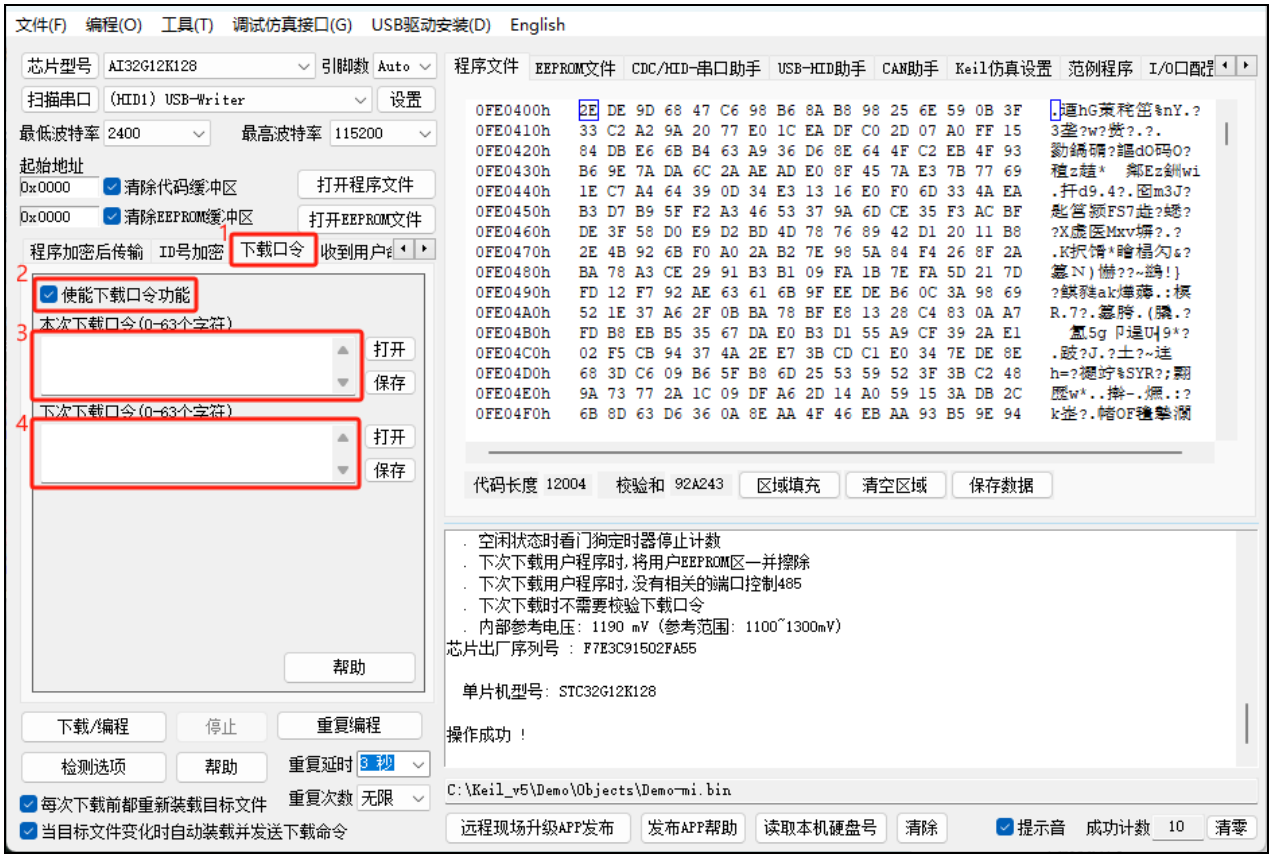
2.33.6　下载需口令高级功能

芯片提供了下载需口令的功能，此功能主要是实现将目标芯片设置下载口令，下次再下载程序【更新程序】时，必须输入正确的下载口令才可下载代码，从而防止了芯片内部的程序被恶意修改。下面详细说明一下下载需口令功能的使用。

➢ 使能下载需口令功能

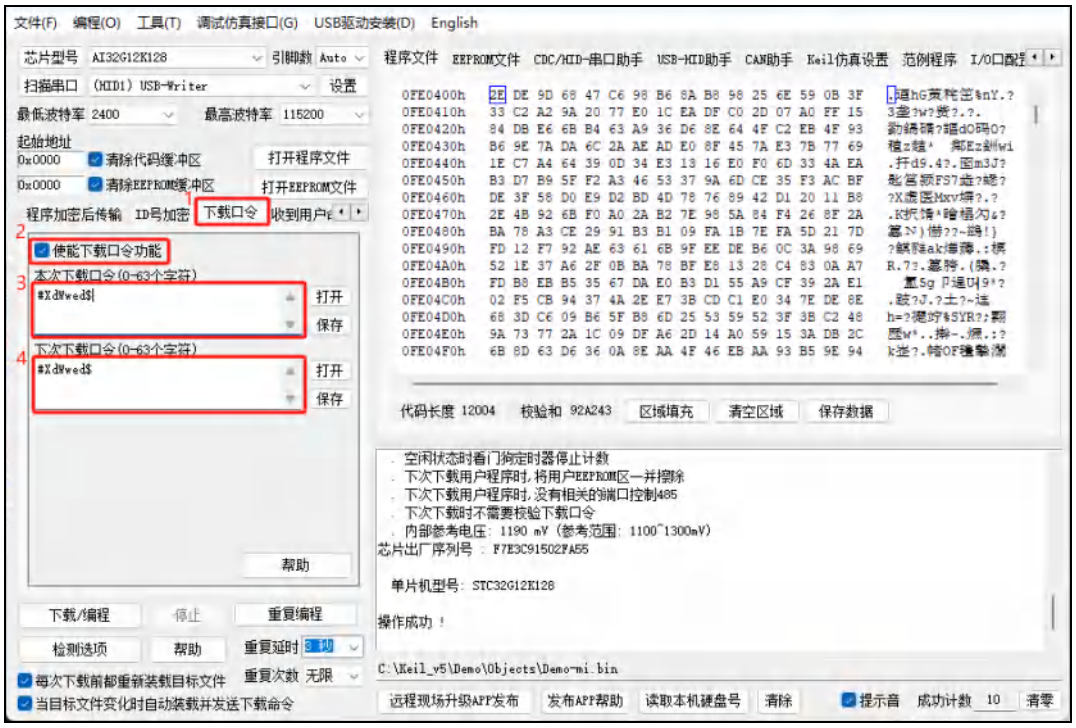
1、单片机在出厂时默认没有使能下载需口令的功能，所以第一次 ISP 下载时，需要使能此功能。使能的方法如下图的勾选（步骤②处勾选功能）。

2、对未设置下载口令的芯片进行设置下载口令在本次下载口令输入框内不需要输入（步骤③处保持为空白），在下次下载口令输入框内输入初始的下载口令（步骤④处输出下载口令），然后正常下载即可。



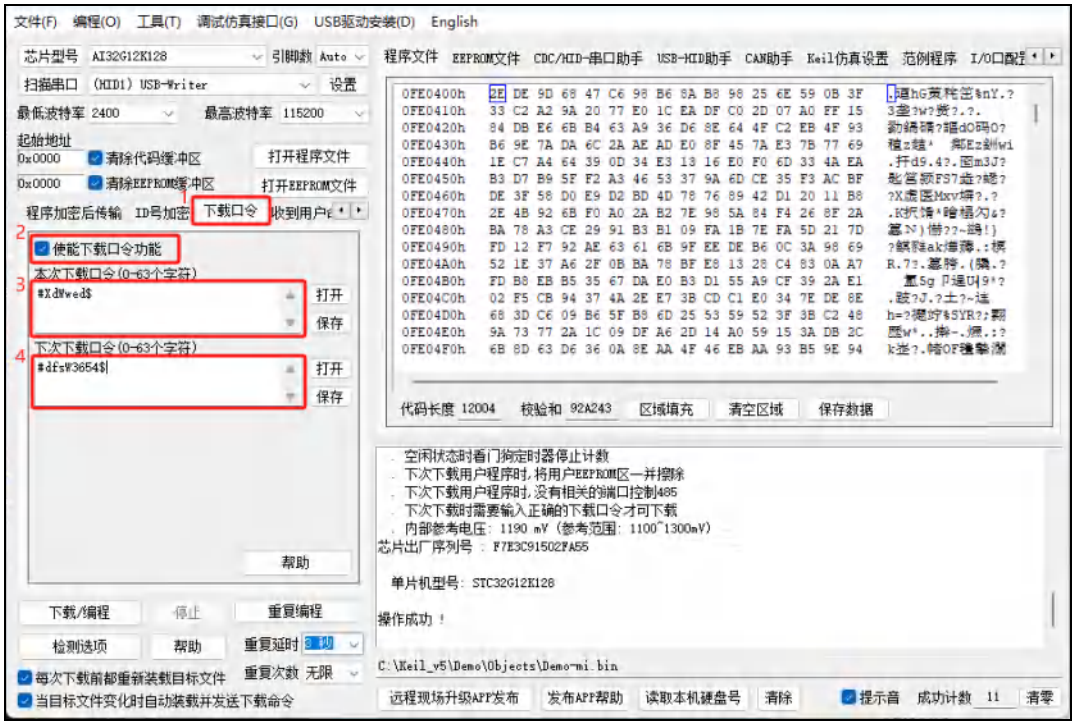
➢ 使用下载口令进行 ISP 下载

对已设置下载口令的芯片进行正常 ISP 下载时，必须要勾选下载口令功能（**步骤②处进行勾选**），在本次下载口令输入框和下次下载口令输入框内都输入之前设置的下载口令（**步骤③和步骤④均输入之前设置的下载口令**），然后正常下载即可



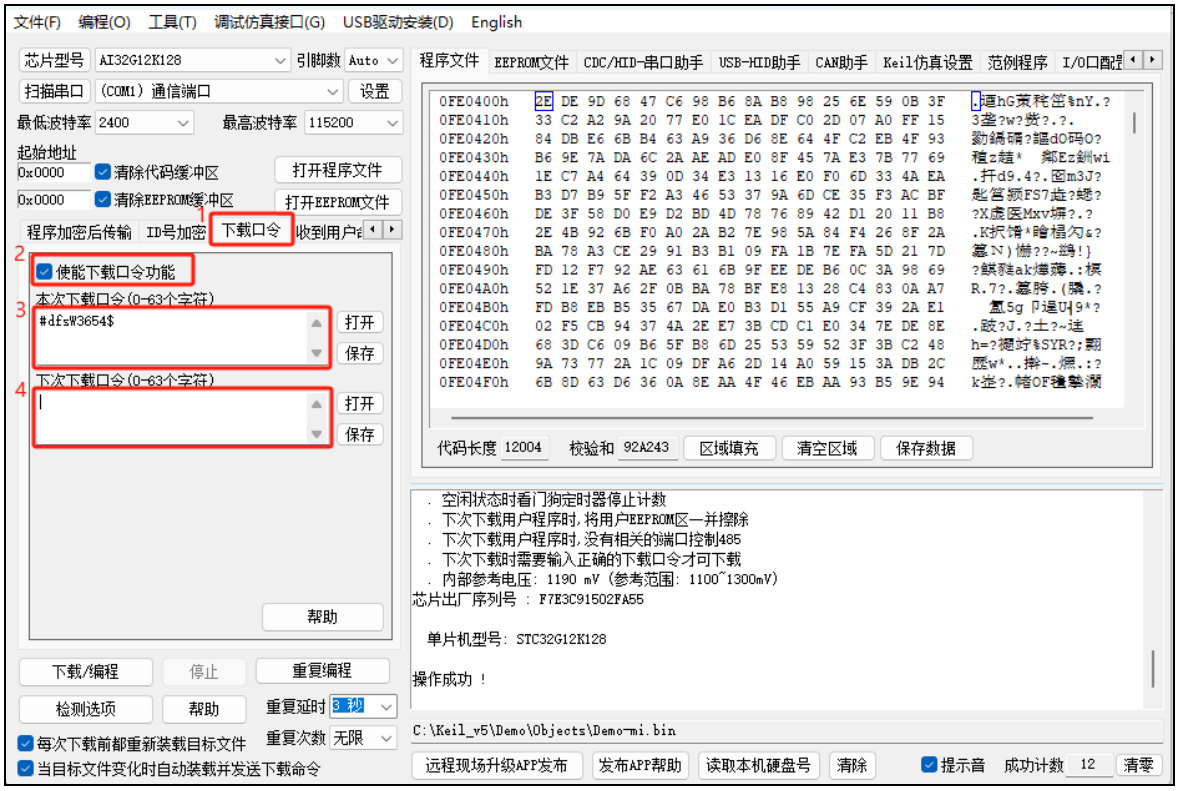
➢ 修改 ISP 下载口令

当需要修改下载口令时，可以在 ISP 下载时进行如下设置，勾选下载口令功能（**步骤②处进行勾选**）在本次下载口令输入框内输入之前设置的下载口令（**步骤③处输入之前设置的下载口令**），在下次下载口令输入框内输入新的下载口令（**步骤④处输入新的下载口令**），然后正常下载即可



➢ 取消下载需口令功能

对已设置下载口令的芯片，如果需要取消下载口令，本次下载时仍然需要勾选下载口令功能（**步骤②处进行勾选**），在本次下载口令输入框内输入之前设置的下载口令（**步骤③处输入之前设置的下载口令**），在下次下载口令输入框内不需要输入任何内容（**步骤④处处保持为空白**），然后正常下载即可。本次下载完成后，下次下载时就不需要口令，也不用勾选下载口令功能了



下载需口令功能的特别注意事项：

1、对于设置了下载口令的芯片，最多只允许进行 5 次的错误尝试（即输入 5 次错误的口令）。当尝试的次数达到 5 次后，目标芯片将被永久锁死，即使再输入正确的下载密码也不能解锁。所以一定要将设置的密码保存好。

2、口令的最大长度为 63 个字符，一个 ASCII 字符为一个字符长度，一个汉字或一个全角字符为两个字符长度。口令的字符不能包含空格、TAB 符号、回车换行符等空白字符。用户密码的格式必须以‘#’开头，以‘$’结尾。

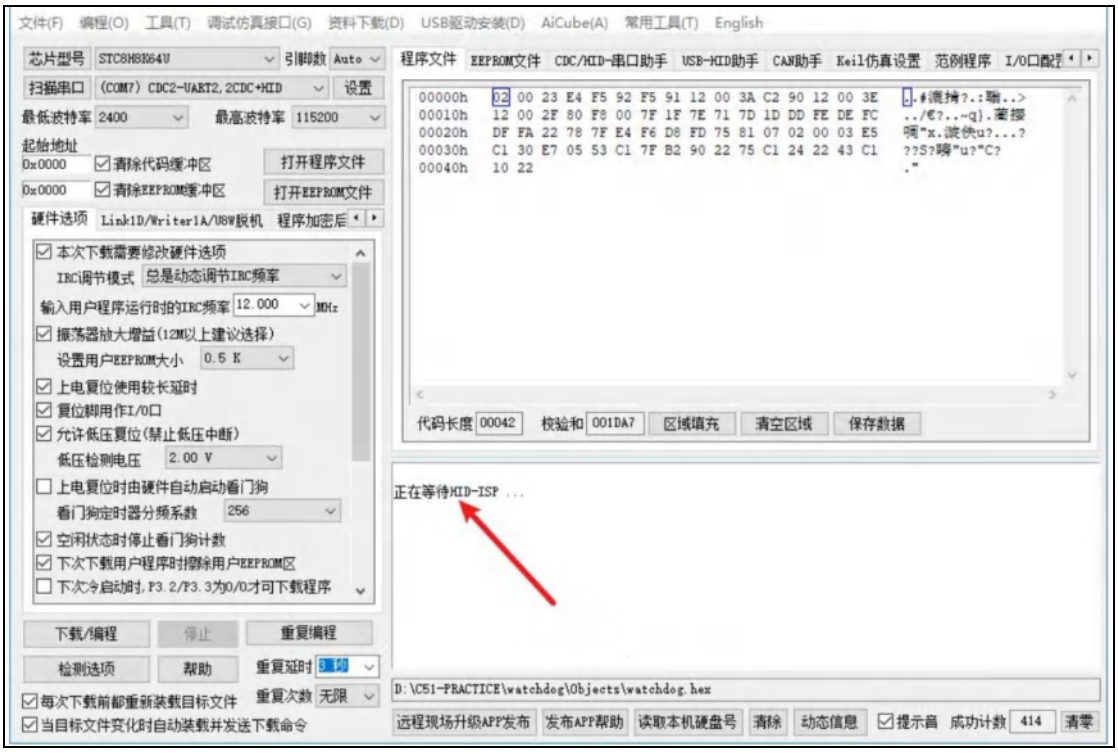
下载需口令功能的应用场景：

1、电子衡器/电子秤市场，市场监督部门在找他们要杜绝衡器出厂后，被更改程序后，变成作弊的电子秤，这个场景更新程序就需要【下载口令】的功能。

2、电子密码锁，防止上门维修人员，擅自更改程序，留后门密码，破坏厂家形象。

2.34 用【USB 转串口的下载器】下载，不要勾选【下次使用 HID 接口进行 ISP 下载】

当使用【USB 转串口的下载器】下载或者普通串口下载时，如果出现“正在等待 HID-ISP...”，如下图：

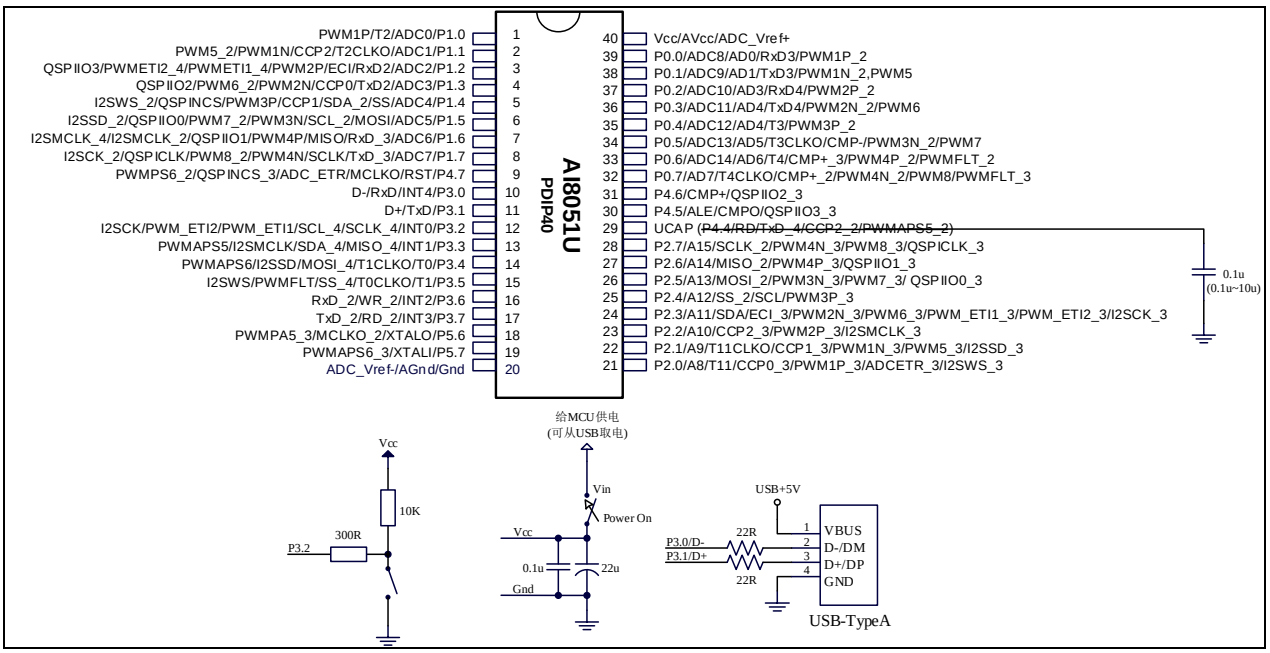


请在“收到用户命令后复位到 ISP 监控程序区”页面中，不要勾选【下次使用 HID 接口进行 ISP 下载】

3 USB 库函数使用说明，简单的 USB-CDC 虚拟串口通信，USB 不停电下载

3.1 强大而简单的 USB-CDC 虚拟串口概述

当单片机需要和电脑之间进行数据交换时，首选一般都是串口通讯。Ai8051U 系列单片机内置 USB 控制器和收发器，当用户代码中嵌入 USB-CDC 代码后，使用 USB 线将单片机与电脑直接相连接，在电脑端就可识别出【USB-CDC 虚拟串口】，【USB-CDC 虚拟串口】就是【串口】。



USB-CDC 虚拟串口和传统串口相比有如下优点：

- **数据传输更快**：USB-CDC 虚拟串口忽略传统串口的波特率，传输速度的比特率即为全速 USB 的通讯速度 12MBPS（即每秒 12M 位）
- **使用更简单便捷**：USB-CDC 虚拟串口忽略传统串口的起始位、停止位等冗余信息
- **数据传输更可靠**：USB-CDC 虚拟串口丢弃传统串口简单的软件奇偶校验机制，USB-CDC 虚拟串口数据传输时有 USB 硬件 CRC 校验，以及校验出错重传机制，保证数据 100%正确
- **自动缓存数据**：USB-CDC 虚拟串口会自动缓存数据。单片机在没有处理完成上位机下传的上一笔数据时，如果此时上位机又有新的数据下传，虚拟串口会自动将新的数据缓存，从而保证数据 100%不会丢失或被覆盖。

3.2 使用 C#/C++/VB 开发 USB-CDC 虚拟串口的应用程序与普通串口一样吗？

USB-CDC 虚拟串口/就是串口，网友问：

问题 1：使用 C#/C++/VB 编程开发上位机软件，直接调用普通的串口通讯控件与 AI32G12K128 / AI8H8K64U 的 USB-CDC 串口通信可以吗 ？

回答 1：　＝＝USB-CDC 串口在 PC 端的使用和普通串口一模一样

＝＝C#/VB 的 MSCOMM 串口控件访问 USB-CDC 虚拟串口的方式和访问普通串口一样

＝＝C++的串口相关 API 函数访问 USB-CDC 虚拟串口的方式和访问普通串口一样

＝＝如果不使用 AI32G12K128 / AI8H8K64U 的 USB-CDC 虚拟串口当 BRIDGE / USB-CDC 再转串口，则可以忽略【波特率、数据位、停止位、奇偶校验】等参数

问题 2：使用 C#/C++/VB 编程时，MSCOMM 串口控件或串口 API 如何设置波特率、数据位、停止位、奇偶校验等参数？

回答 2：　＝＝如果不使用 AI32G12K128 / AI8H8K64U 的 USB-CDC 虚拟串口当 BRIDGE / USB-CDC 再转串口，则可以忽略【波特率、数据位、停止位、奇偶校验】等参数

＝＝如果需要使用 AI32G12K128 / AI8H8K64U 的 USB-CDC 虚拟串口当 BRIDGE / USB-CDC 再转串口，则 USB-CDC 串口的【波特率、数据位、停止位、奇偶校验】等参数的设置方式和普通串口参数的设置方式相同

问题 3：AI32G12K128 系列的 USB-CDC 串口模式和 AI8H8K64U 系列一样吗？

回答 3：　＝＝一样

3.3 USB 库函数简介

STCAI 的 USB 库函数包括两大类：USB-CDC 库函数和 USB-HID 库函数。其中 USB-CDC 库函数基于 USB 通讯设备类协议，USB-HID 库函数是基于 USB 人机交互接口协议。

USB-HID 库函数主要应用于小数据量传输项目。主要特点是在所有版本的 Windows 操作系统中都是免安装驱动程序，即插即用。但数据传输速度比较低，USB-HID 设备的理论数据带宽为 64K 字节/秒，传输 64K 字节程序只需要 1 秒钟，即每毫秒收发一个 USB 数据包，每个数据包最大 64 字节。

USB-CDC 库函数主要应用于较大数据量的快速传输项目。USB-CDC 设备的理论数据带宽为 12M 位/秒（1.5M 字节/秒）。CDC 设备的有效数据传输速度一般可达到 800K 字节/秒。但 USB-CDC 有驱动程序的问题，对于 Win10（1903 版）以下的操作系统，都需要安装驱动后才能正常使用，对此 STCAI 公司有特别针对低版本的操作系统提供了 USB-CDC 驱动的安装程序。Win10（1903 版）之后的操作系统，包括最新的 Win11，对 USB-CDC 也都免安装驱动程序，实现了即插即用。

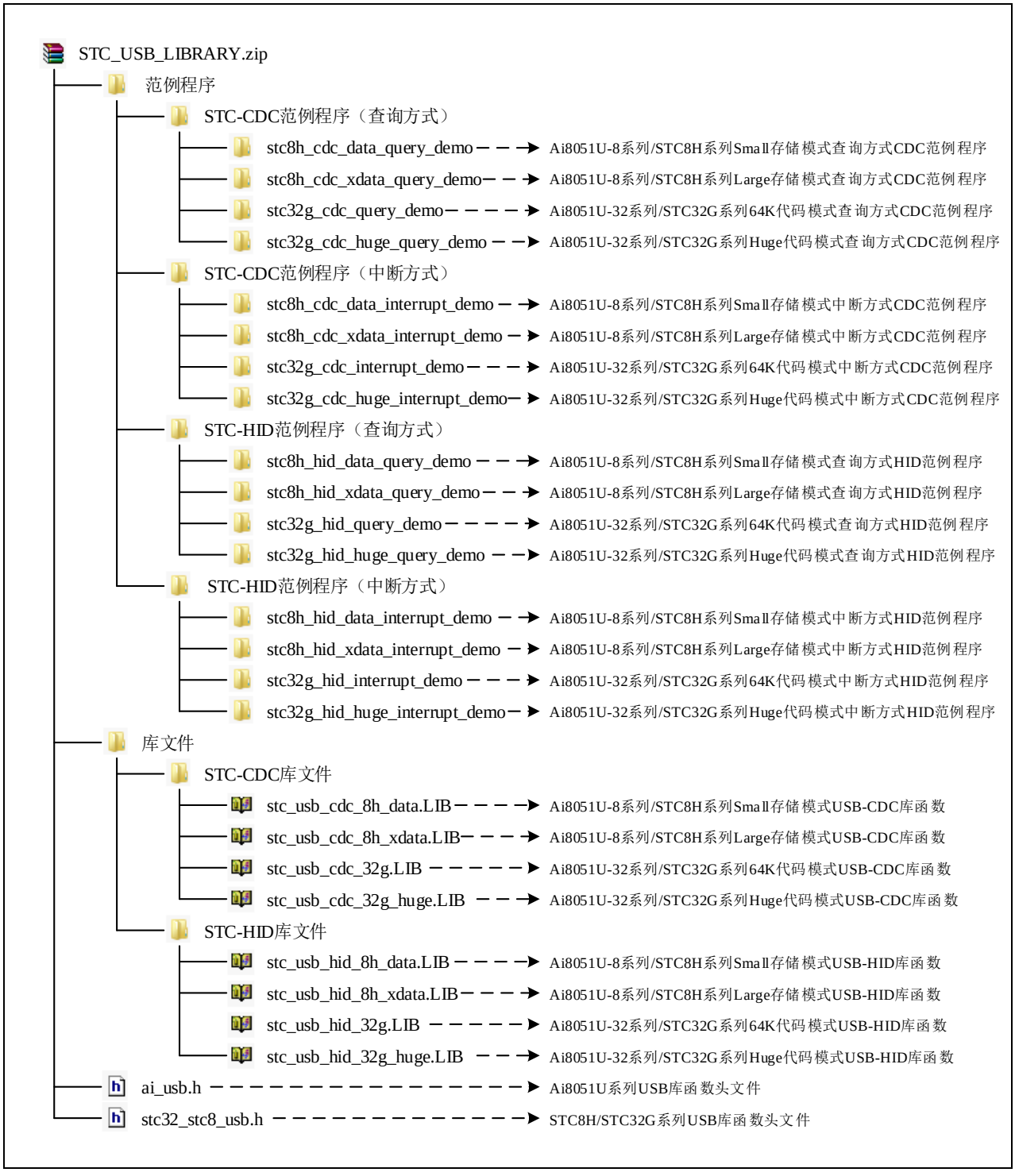
用户可以根据自己项目的实际数据传输速度需求以及软件运行的电脑操作系统，来选择使用选择 STCAI 提供的 USB-CDC 库函数和 USB-HID 库函数。

3.4 下载 USB 库函数

STCAI 的 USB 库函数可以在官网的如下地址进行下载

下载链接：https://www.stcaimcu.com/data/download/Library/STC_USB_LIBRARY.zip

“STC_USB_LIBRARY.zip”压缩包解压后的目录结构图如下：


STC_USB_LIBRARY.zip
范例程序
STC-CDC范例程序（查询方式）
stc8h_cdc_data_query_demo → Ai8051U-8系列/STC8H系列Small存储模式查询方式CDC范例程序
stc8h_cdc_xdata_query_demo → Ai8051U-8系列/STC8H系列Large存储模式查询方式CDC范例程序
stc32g_cdc_query_demo → Ai8051U-32系列/STC32G系列64K代码模式查询方式CDC范例程序
stc32g_cdc_huge_query_demo → Ai8051U-32系列/STC32G系列Huge代码模式查询方式CDC范例程序
STC-CDC范例程序（中断方式）
stc8h_cdc_data_interrupt_demo → Ai8051U-8系列/STC8H系列Small存储模式中断方式CDC范例程序
stc8h_cdc_xdata_interrupt_demo → Ai8051U-8系列/STC8H系列Large存储模式中断方式CDC范例程序
stc32g_cdc_interrupt_demo → Ai8051U-32系列/STC32G系列64K代码模式中断方式CDC范例程序
stc32g_cdc_huge_interrupt_demo → Ai8051U-32系列/STC32G系列Huge代码模式中断方式CDC范例程序
STC-HID范例程序（查询方式）
stc8h_hid_data_query_demo → Ai8051U-8系列/STC8H系列Small存储模式查询方式HID范例程序
stc8h_hid_xdata_query_demo → Ai8051U-8系列/STC8H系列Large存储模式查询方式HID范例程序
stc32g_hid_query_demo → Ai8051U-32系列/STC32G系列64K代码模式查询方式HID范例程序
stc32g_hid_huge_query_demo → Ai8051U-32系列/STC32G系列Huge代码模式查询方式HID范例程序
STC-HID范例程序（中断方式）
stc8h_hid_data_interrupt_demo → Ai8051U-8系列/STC8H系列Small存储模式中断方式HID范例程序
stc8h_hid_xdata_interrupt_demo → Ai8051U-8系列/STC8H系列Large存储模式中断方式HID范例程序
stc32g_hid_interrupt_demo → Ai8051U-32系列/STC32G系列64K代码模式中断方式HID范例程序
stc32g_hid_huge_interrupt_demo → Ai8051U-32系列/STC32G系列Huge代码模式中断方式HID范例程序
库文件
STC-CDC库文件
stc_usb_cdc_8h_data.LIB → Ai8051U-8系列/STC8H系列Small存储模式USB-CDC库函数
stc_usb_cdc_8h_xdata.LIB → Ai8051U-8系列/STC8H系列Large存储模式USB-CDC库函数
stc_usb_cdc_32g.LIB → Ai8051U-32系列/STC32G系列64K代码模式USB-CDC库函数
stc_usb_cdc_32g_huge.LIB → Ai8051U-32系列/STC32G系列Huge代码模式USB-CDC库函数
STC-HID库文件
stc_usb_hid_8h_data.LIB → Ai8051U-8系列/STC8H系列Small存储模式USB-HID库函数
stc_usb_hid_8h_xdata.LIB → Ai8051U-8系列/STC8H系列Large存储模式USB-HID库函数
stc_usb_hid_32g.LIB → Ai8051U-32系列/STC32G系列64K代码模式USB-HID库函数
stc_usb_hid_32g_huge.LIB → Ai8051U-32系列/STC32G系列Huge代码模式USB-HID库函数
ai_usb.h → Ai8051U系列USB库函数头文件
stc32_stc8_usb.h → STC8H/STC32G系列USB库函数头文件

3.5 USB 库函数的详细说明

USB 的头文件“ai_usb.h”和“stc32_stc8_usb.h”中的宏定义、变量定义以及函数声明的说明如下：

```
//#define PRINTF_SEGLED
//#define PRINTF_HID
//#define PRINTF_USB
```

PRINTF_SEGLED、PRINTF_USB 和 PRINTF_HID 是对 printf 函数打印的内容进行重定向的宏定义。3 个宏最多只能选择其中之一

PRINTF_SEGLED：当需要使用 printf 函数将打印的内容重定向到 ISP 下载软件中的 8 段数码管时，请使用此宏定义

PRINTF_USB：当需要使用 printf 函数将打印的内容重定向到 ISP 下载软件中的 CDC 串口助手或者 HID 助手时，请使用此宏定义

PRINTF_HID：意义与 PRINTF_USB 相同

```
#define DEVSTATE_ATTACHED      0
#define DEVSTATE_POWERED       1
#define DEVSTATE_DEFAULT       2
#define DEVSTATE_ADDRESS       3
#define DEVSTATE_CONFIGURED    4
#define DEVSTATE_SUSPENDED     5
```

上面的宏定义是 USB 设备的状态“DeviceState”，当 USB 主机对 USB 设备进行配置过程中，USB 设备的状态保存在“DeviceState”变量中，用户只需要关注“DEVSTATE_CONFIGURED”这个状态，当“DeviceState”为“DEVSTATE_CONFIGURED”时，USB 设备才可和主机进行数据传输。

USB 库中有定义如下变量：

- “bUsbOutReady”：
  主机发送给 HID/CDC 设备的数据状态。状态位为 1 表示，主机发送给 HID/CDC 设备数据已完成，设备已成功接收一包数据，等待对数据继续处理。
- “UsbOutBuffer”：
  主机发送给 HID/CDC 设备的数据缓冲区（HID/CDC 设备的数据**接收**缓冲区）。只有当“bUsbOutReady”状态为 1 时，“UsbOutBuffer”的内容才有效。
- “OutNumber”：
  主机发送给 HID/CDC 设备的数据字节数。只有当“bUsbOutReady”状态为 1 时，“OutNumber”的内容才有效。
- “DeviceState”：
  USB 设备的状态。只有当设备状态为“DEVSTATE_CONFIGURED”时，USB 设备才可和主机进行数据传输。特别提醒：内置的 printf_usb()和 USB_SendData()函数都内置检测 USB 状态和发送忙，当 usb 为非配置状态或者处于忙状态，则会跳过当前的 printf_usb()和 USB_SendData()操作，不需要额外判断直接调用也不会堵塞整个函数
- “UsbFeatureBuffer”：（一般不需要使用）
  主机发送给 USB 设备的 Feature 数据（特征报告数据）缓冲区。Feature 数据仅对 HID 设备有效，CDC 设备无 Feature 数据。
- “bUsbFeatureReady”：（一般不需要使用）
  主机发送给 USB 设备的 Feature 数据（特征报告数据）状态。状态位为 1 表示，HID/CDC 设备已接收完成 Feature 数据。

- “UsbInBuffer”：（一般不需要使用）

  HID/CDC 设备上传数据给主机的数据缓冲区。一般不需要（不建议）用户自己设置，而是直接使用“USB_SendData”进行自动设置。

- “bUsbInBusy”：（一般不需要使用）

  HID/CDC 设备上传数据给主机的忙标志位，状态位为 1 表示，上一包数据还未传输完成，不能继续上传数据。

USB 库中有提供如下函数

- void usb_init();：

  USB 库初始化函数，用户只需要在程序的系统初始化部分调用一次“usb_init();”，调用完 usb_init();并使能 EA（EA=1）后，硬件便会自动对 USB 设备进行初始化。

- void usb_OUT_done();：

  当使用查询方式调用 USB 库函数时，数据处理完成后，必须调用一次“usb_OUT_done();”，这样才能接收下一包数据。如果是使用中断方式调用 USB 库函数，用户程序不需要调用“usb_OUT_done();”，库函数内部会自动处理。

- void USB_SendData(BYTE *dat, int size); ：

  HID/CDC 设备上传数据给主机函数。dat 参数是数据缓冲区的起始地址，size 是数据包大小。特别注意：CDC 设备的数据包大小不固定；HID 设备的数据包大小固定为 64 字节。

- void set_usb_OUT_callback(void (*pfn)(void)); ：

  如果需要使用中断方式调用 USB 库函数，需要调用此函数来设置回调函数。若不设置，则是使用默认的查询方式调用 USB 库函数。

- void set_usb_IN_callback(void (*pfn)(void)); （一般不需要使用）：

  如果需要实现发送中断，调用此函数设置回调。

- void set_usb_ispcmd(char *cmd); ：

  设置不停电下载的用户自定义命令，不设置时默认下载命令为“@STCISP#”。

- int printf_usb (const char *fmt, ...); ：

  将格式化后的字符串通过 USB-HID/USB-CDC 输出到电脑端（USB 主机），printf_usb 支持 keil 的 printf 函数所支持的全部格式化符号。

USB 库中提供如下函数将数据显示到“擎天柱”、“降龙棍”、“打狗棒”调试仿真接口：

- void LED40_ClrBit(BYTE port, BYTE bt);　　//设置 port.bt 位为低电平
- void LED40_SetBit(BYTE port, BYTE bt);　　//设置 port.bt 位为高电平
- void LED40_SetPort(BYTE port, BYTE dat);　//设置整组 port
- void LED40_SendData(BYTE *dat, BYTE size);　（一般不需要使用）

USB 库中提供如下函数将数据显示到“开天斧”、“屠龙刀”调试仿真接口：

- void LED64_ClrBit(BYTE port, BYTE bt);　　//设置 port.bt 位为低电平
- void LED64_SetBit(BYTE port, BYTE bt);　　//设置 port.bt 位为高电平
- void LED64_SetPort(BYTE port, BYTE dat);　//设置整组 port
- void LED64_SendData(BYTE *dat, BYTE size);　（一般不需要使用）

USB 库中提供如下函数将数据显示到“8 段数码管”调试仿真接口：

- int SEG7_ShowString(const char *fmt, ...);
- void SEG7_ShowLong(long n, char radix);
- void SEG7_ShowFloat(float f);
- void SEG7_ShowCode(BYTE *cod);

USB 库中提供如下函数将数据显示到“液晶显示 128*64”调试仿真接口：

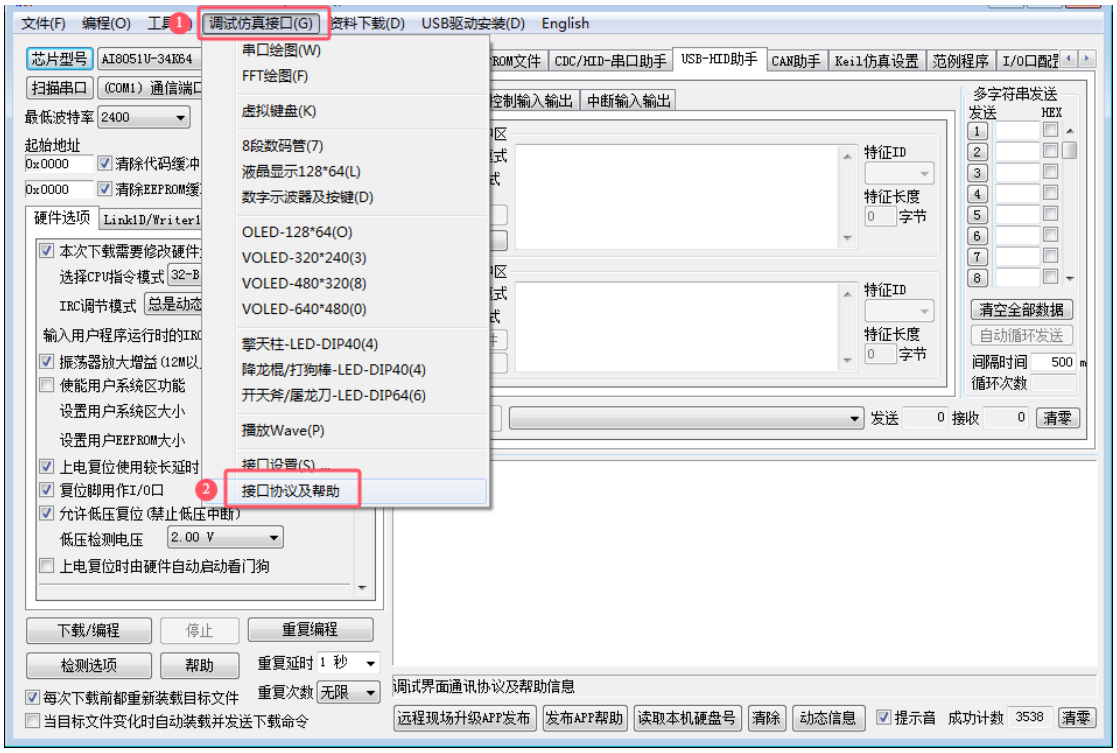
- void LCD12864_DisplayOff();
- void LCD12864_DisplayOn();
- void LCD12864_CursorOff();
- void LCD12864_CursorOn();
- void LCD12864_CursorMoveLeft();
- void LCD12864_CursorMoveRight();
- void LCD12864_CursorReturnHome();
- void LCD12864_ScrollLeft();
- void LCD12864_ScrollRight();
- void LCD12864_ScrollUp(BYTE line);
- void LCD12864_AutoWrapOff();
- void LCD12864_AutoWrapOn();
- void LCD12864_ReverseLine(BYTE line);
- void LCD12864_DisplayClear();
- void LCD12864_ShowString(BYTE x, BYTE y, char *str);
- void LCD12864_ShowPicture(BYTE x, BYTE y, BYTE cx, BYTE cy, BYTE *dat);

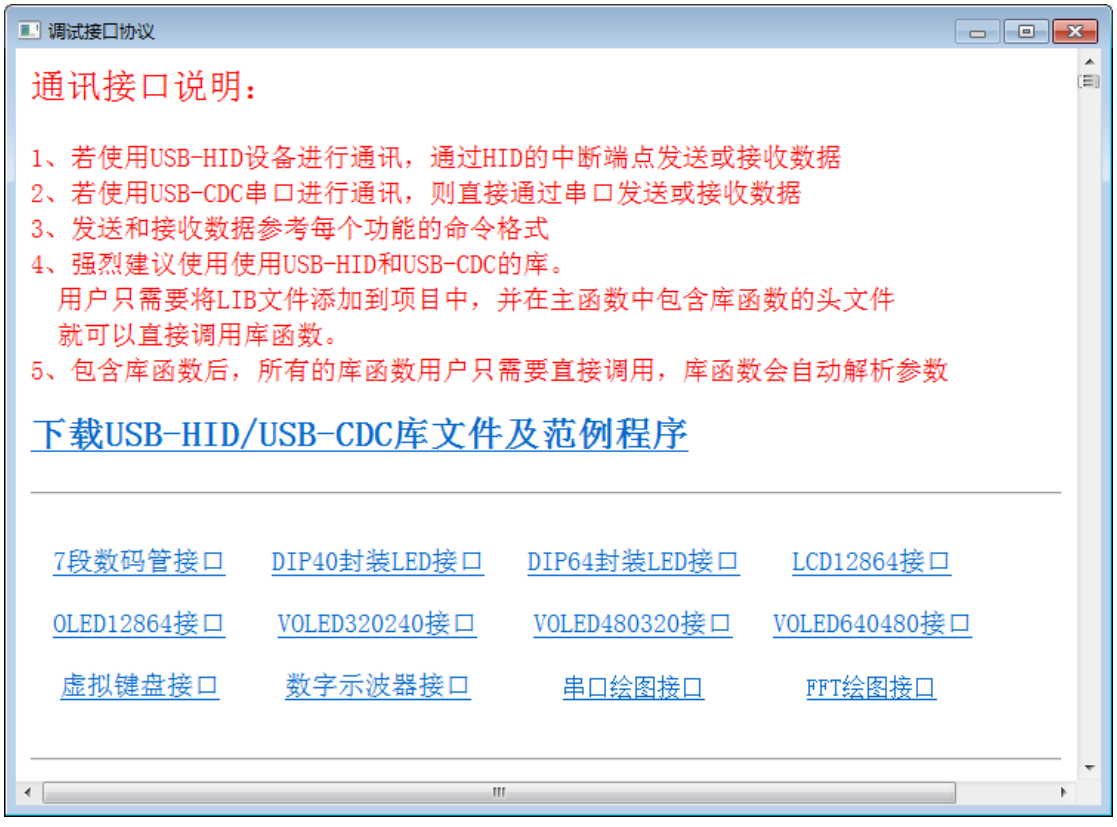
USB 库中提供如下函数将数据显示到“OLED-128*64”调试仿真接口：

- void OLED12864_DisplayOff();
- void OLED12864_DisplayOn();
- void OLED12864_DisplayContent();
- void OLED12864_DisplayEntire();
- void OLED12864_HorizontalMirror();
- void OLED12864_VerticalMirror();
- void OLED12864_DisplayReverse();
- void OLED12864_SetContrast(BYTE bContrast);

- void OLED12864_SetAddressMode(BYTE bMode);
- void OLED12864_ScrollLeft(BYTE bPageStart, BYTE bPageEnd, WORD nInterval);
- void OLED12864_ScrollRight(BYTE bPageStart, BYTE bPageEnd, WORD nInterval);
- void OLED12864_ScrollUp(BYTE bPageStart, BYTE bPageEnd, WORD nInterval);
- void OLED12864_ScrollStart();
- void OLED12864_ScrollStop();
- void OLED12864_ShowPicture(BYTE x, BYTE y, BYTE cx, BYTE cy, BYTE *dat);

上面调试仿真接口函数的详细帮助信息请参考下载软件中的“接口协议及帮助”

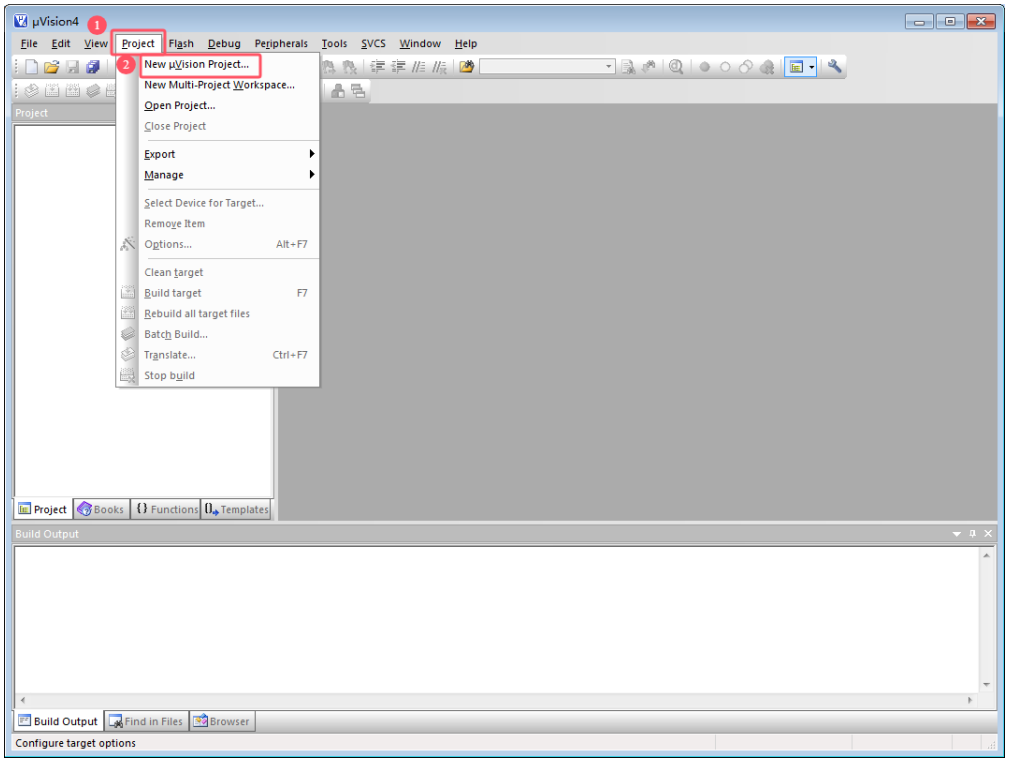




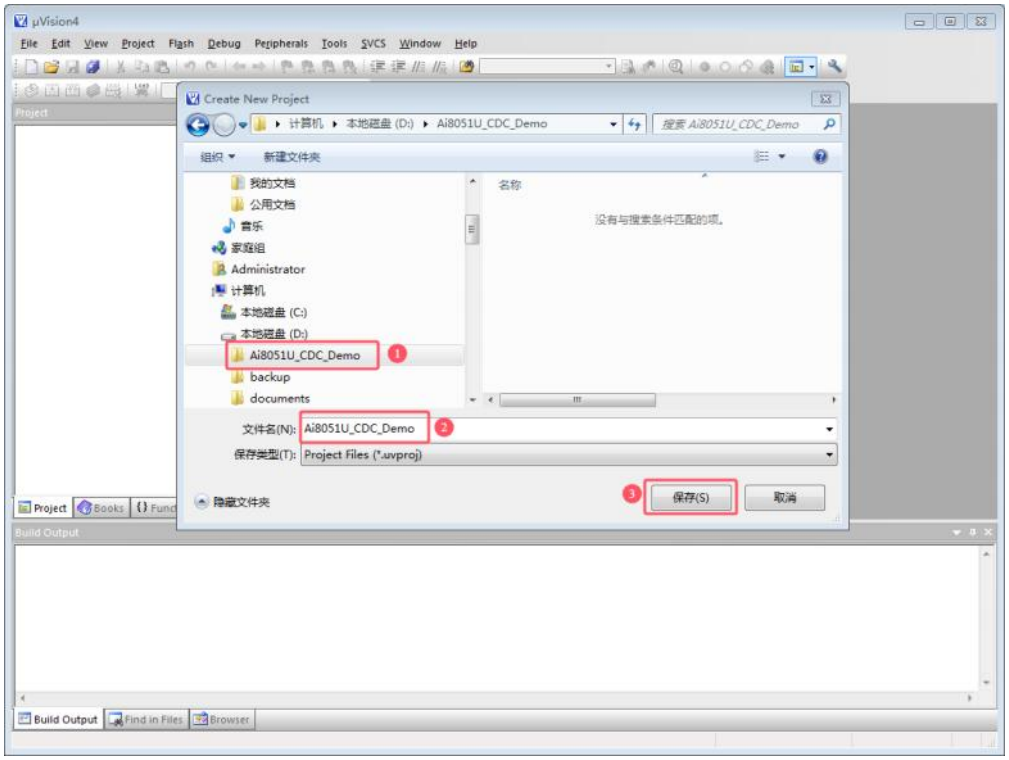
3.6　建立基于 USB 库函数的 Keil 项目

下载并了解了 STC-USB 库函数后，接下来就可以新建基于 USB 库函数的 Keil 项目程序了。目前 STCAI 有硬件 USB 功能的单片机有如下系列 STC8H8K64U、STC8H2K12U 系列、STC8H2K32U 系列、STC32G12K128 系列、Ai8051U-8Bit 系列和 Ai8051U-32Bit 系列。各个系列的单片机建立 Keil 项目的方法都大同小异，下面以 Ai8051U-32Bit 系列单片机为例，演示基于 USB-CDC 库函数的 Keil 项目。

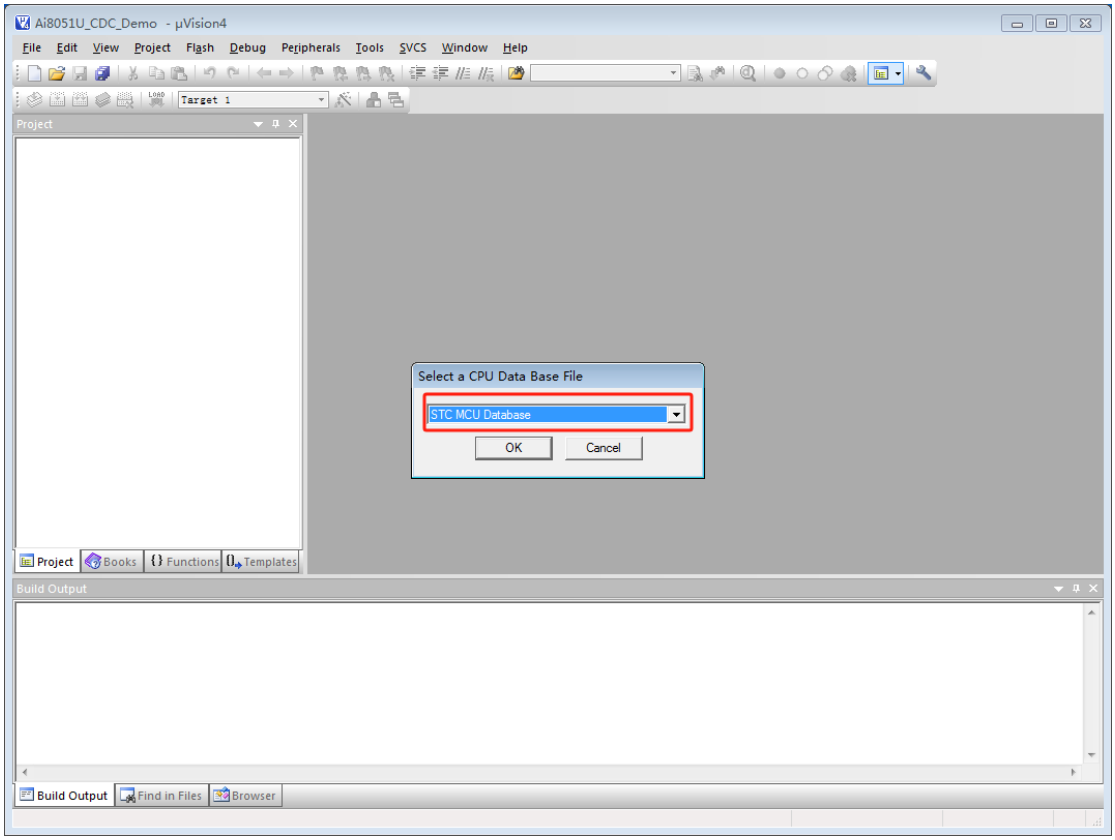
1、打开 Keil 软件，选择菜单项“Project”→“New uVision Project…”



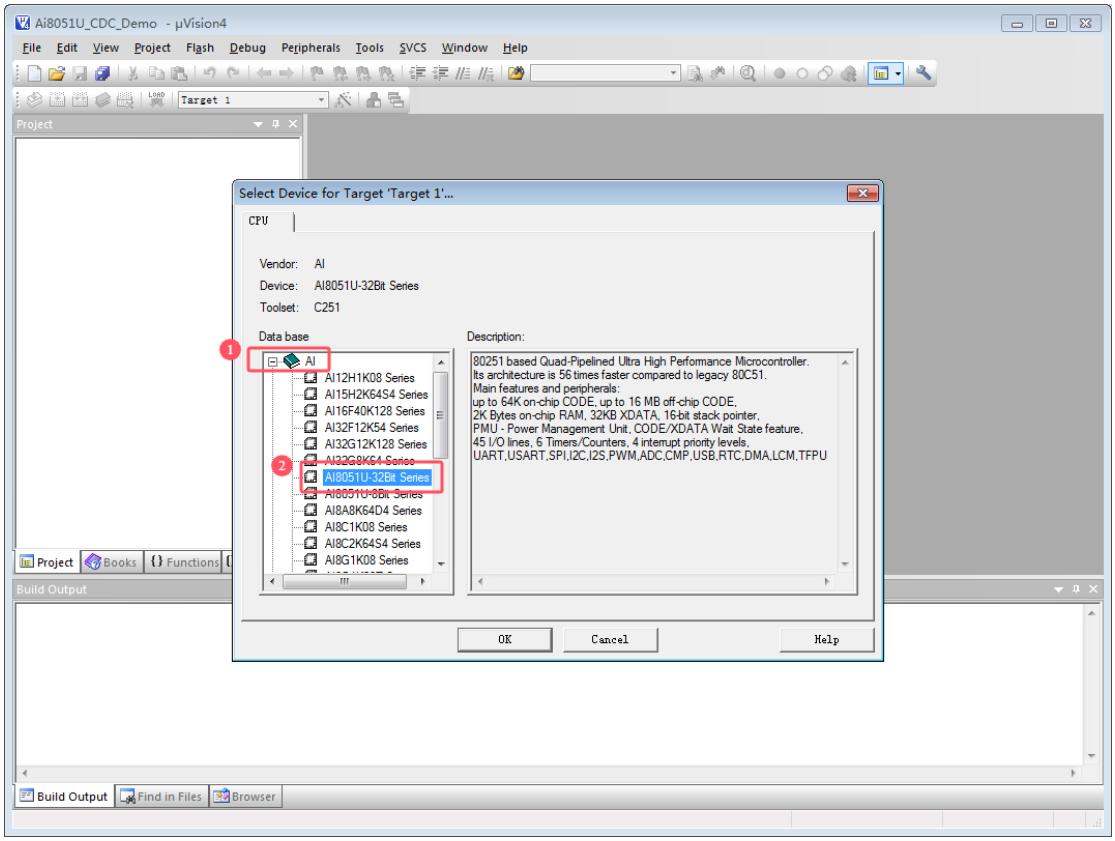
2、新建项目目录，输入项目名称，并保存



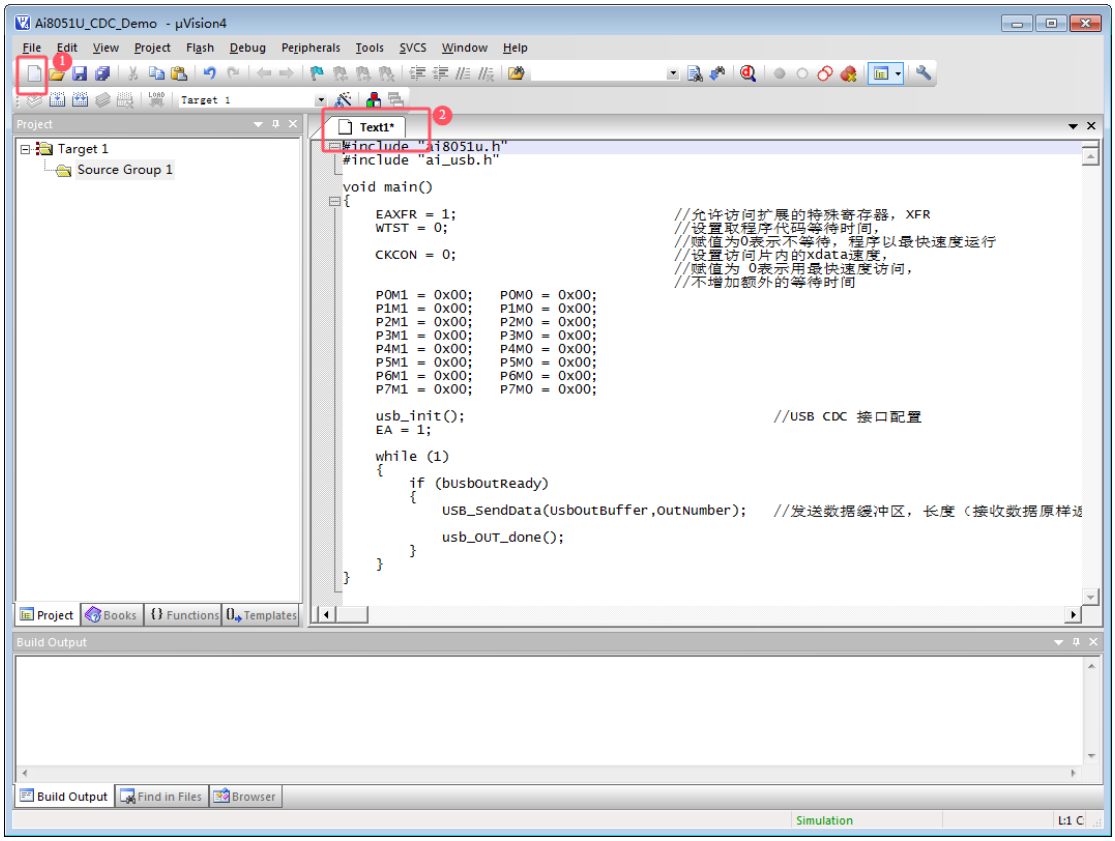
3、选择“STC MCU Database”



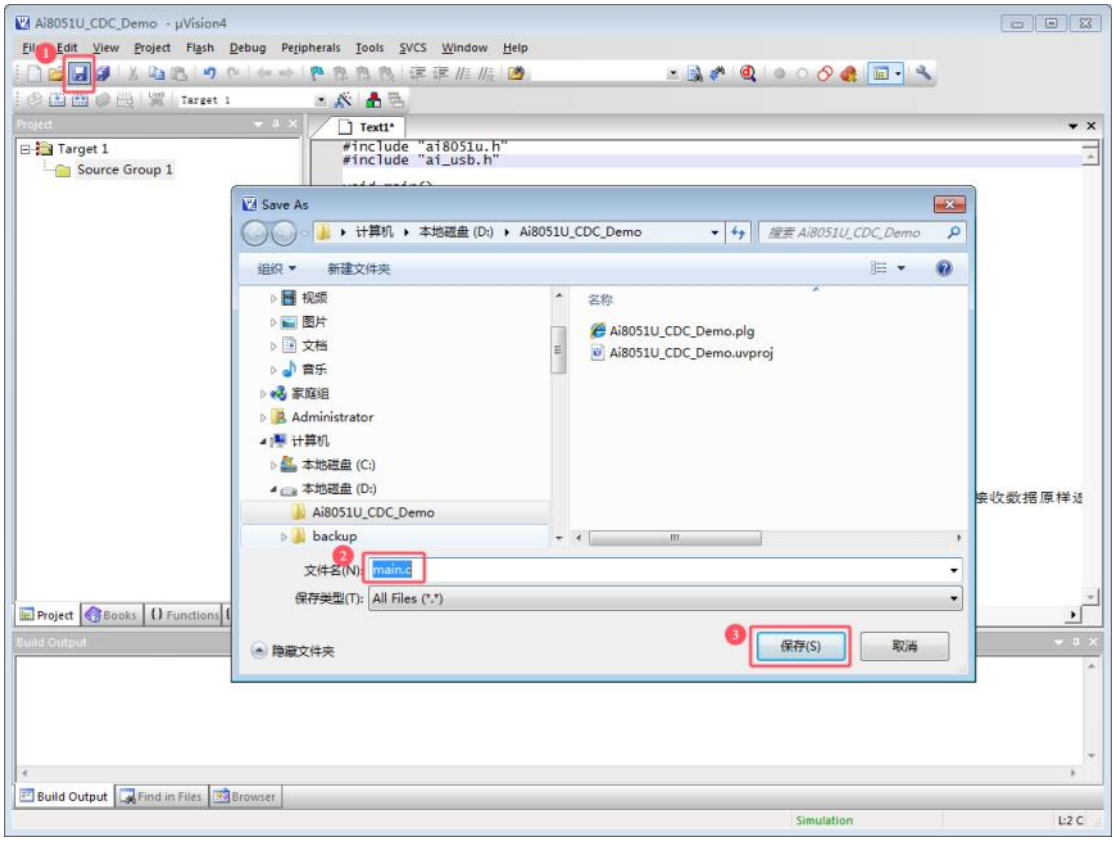
4、选择单片机型号“Ai8051U-32Bit”


Ai8051U_CDC_Demo - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Target 1
Project
Select Device for Target 'Target 1'...
CPU
Vendor: AI
Device: AI8051U-32Bit Series
Toolset: C251
Data base
Description:
AI
AI12H1K08 Series
AI15H2K64S4 Series
AI16F40K128 Series
AI32F12K54 Series
AI32G12K128 Series
AI8051U-32Bit Series
AI8A8K64D4 Series
AI8C1K08 Series
AI8C2K64S4 Series
AI8G1K08 Series
80251 based Quad-Pipelined Ultra High Performance Microcontroller.
Its architecture is 56 times faster compared to legacy 80C51.
Main features and peripherals:
up to 64K on-chip CODE, up to 16 MB off-chip CODE,
2K Bytes on-chip RAM, 32KB XDATA, 16-bit stack pointer,
PMU - Power Management Unit, CODE/XDATA Wait State feature,
45 I/O lines, 6 Timers/Counters, 4 interrupt priority levels,
UART,USART,SPI,I2C,I2S,PWM,ADC,CMP,USB,RTC,DMA,LCM,TFPU
OK
Cancel
Help
Project
Books
Functions
Build Output
Find in Files
Browser

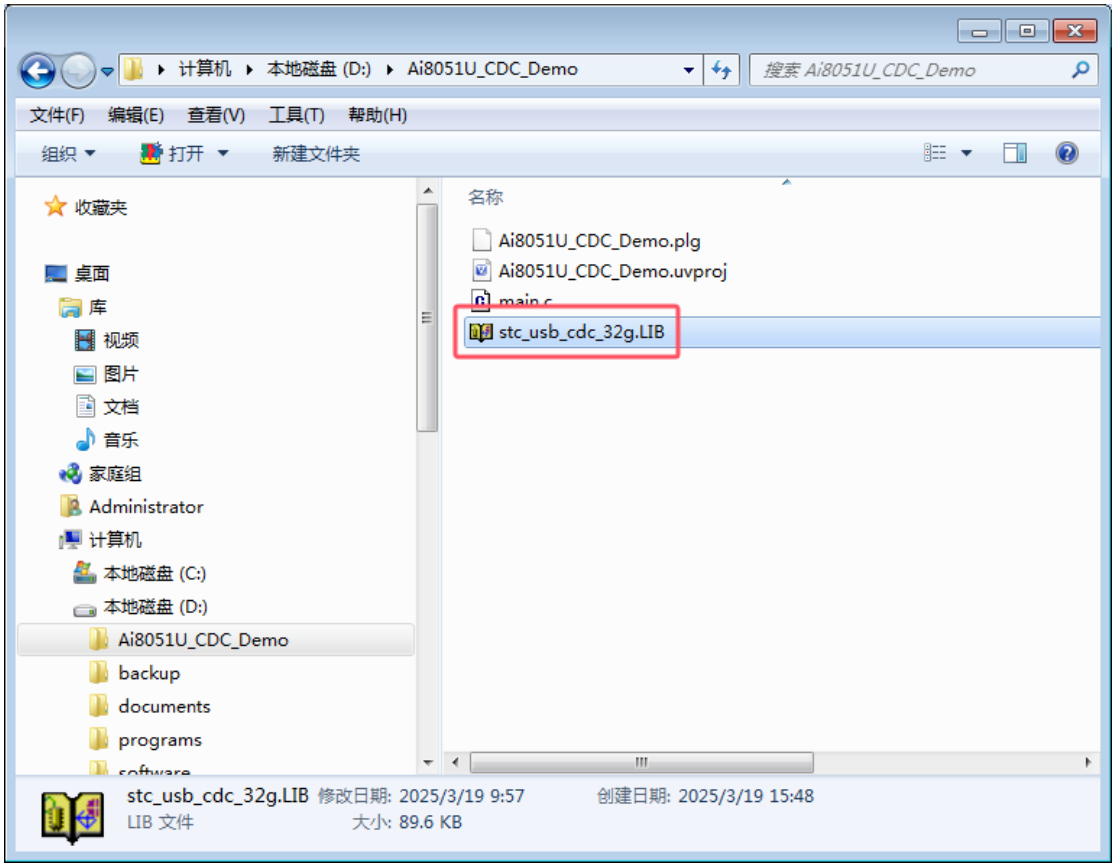
5、新建一个空白文件，并输入如下代码内容



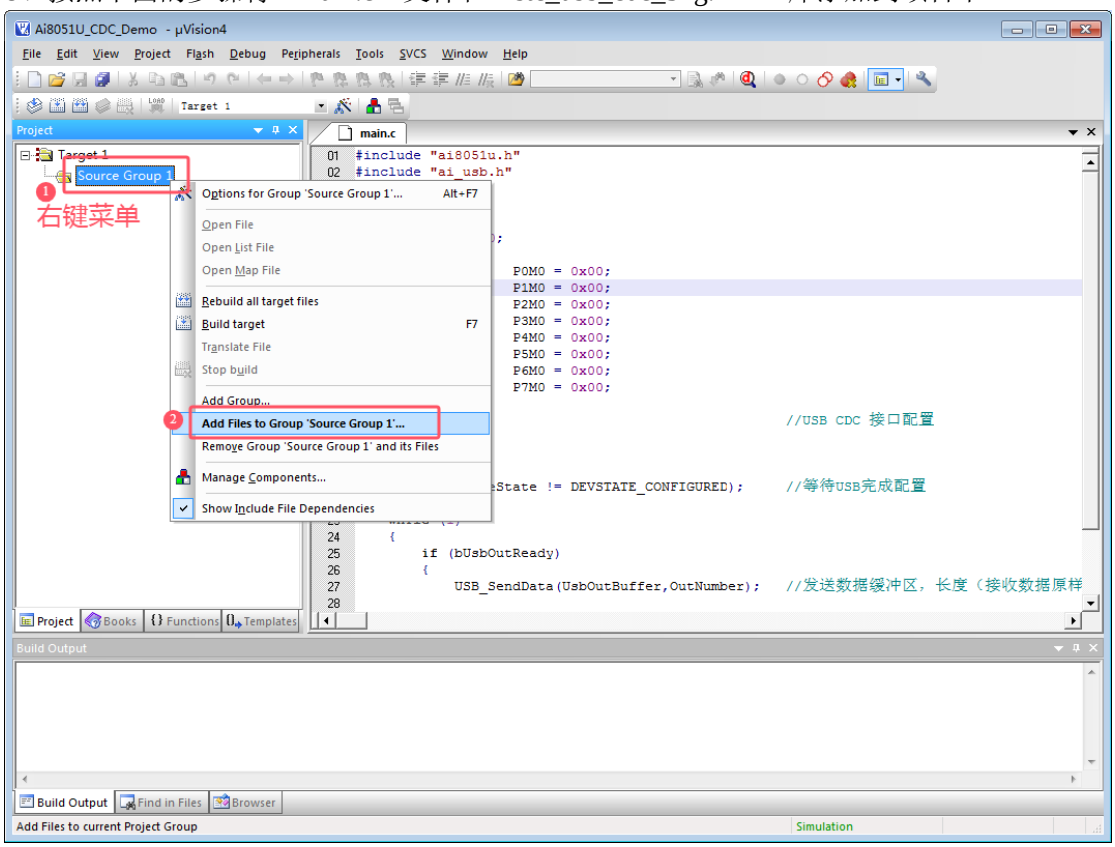
6、将文件保存到项目目录中，保存为“main.c”



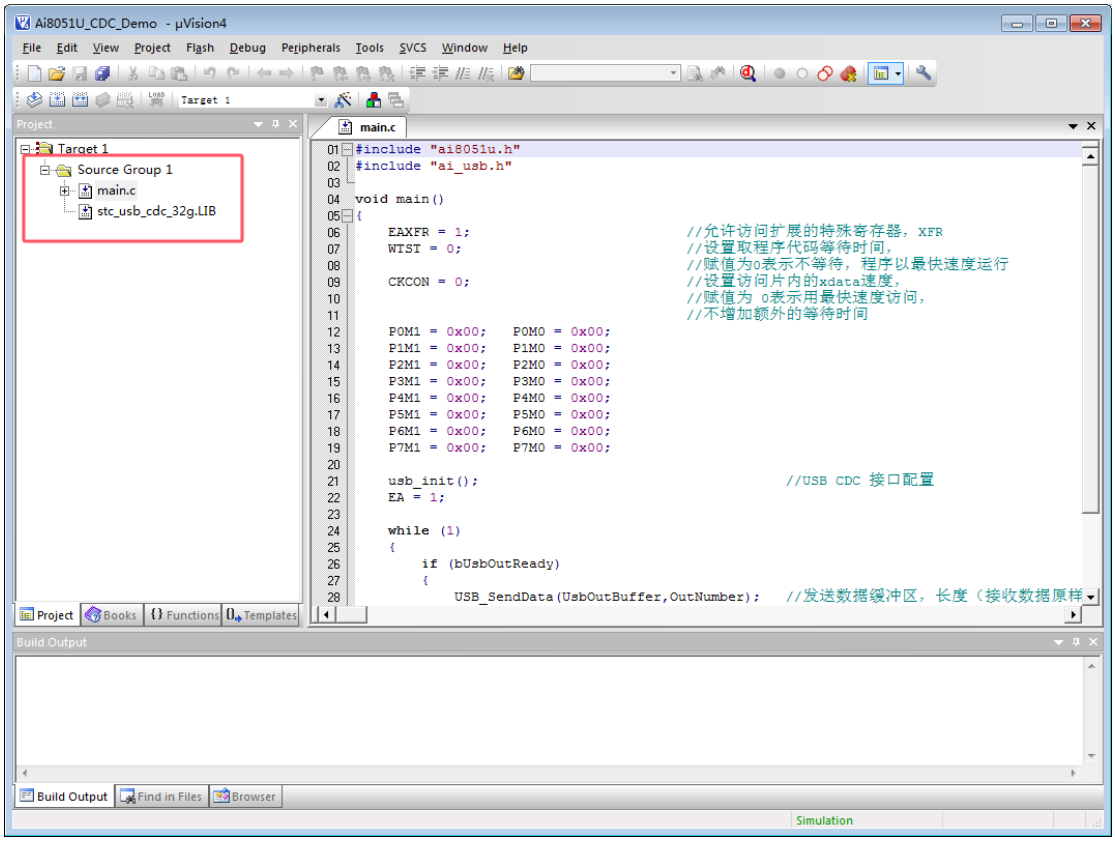
7、将前面下载的库文件包中的“stc_usb_cdc_32g.LIB”也复制到项目目录



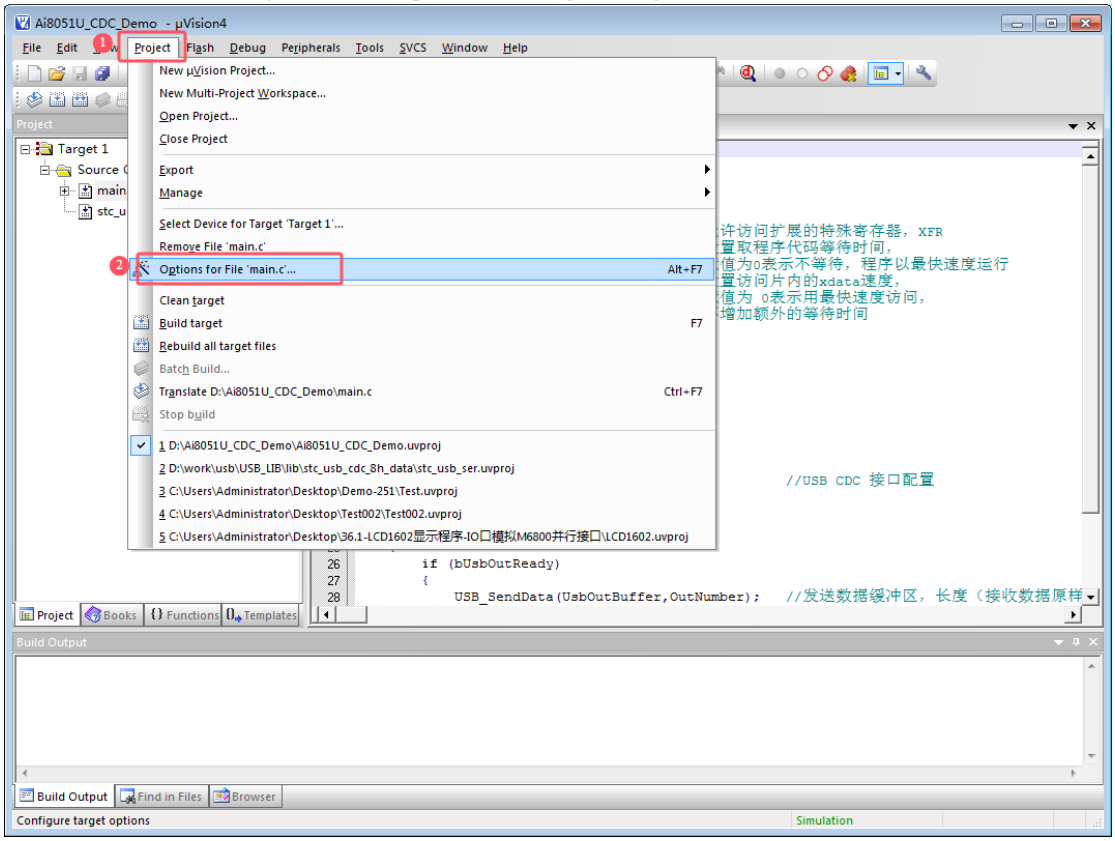
8、按照下面的步骤将“main.c”文件和“stc_usb_cdc_32g.LIB”库添加到项目中



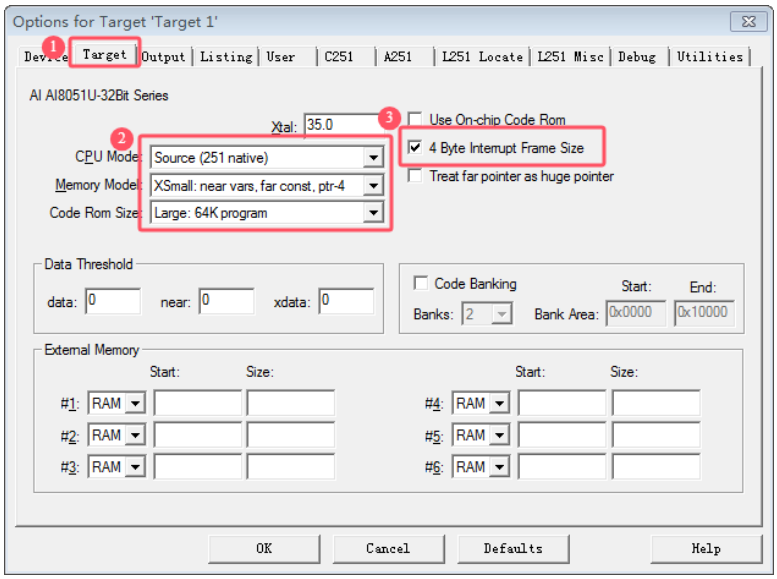
9、添加文件完成后，如下图：

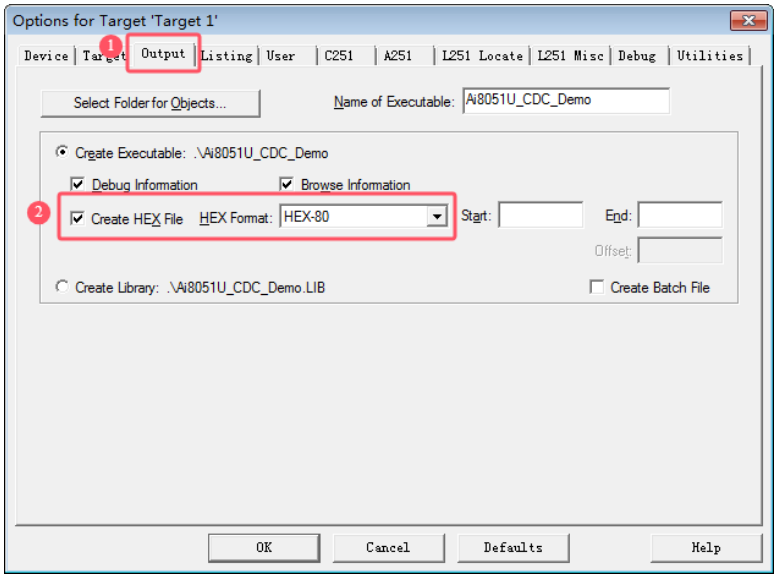


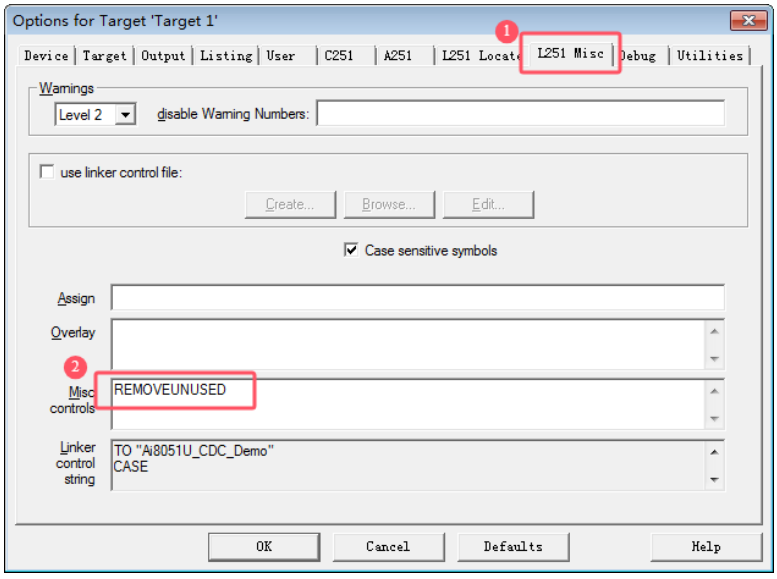
10、选择菜单项“Project”→“Options for Target ‘Target 1’”，打开项目配置界面



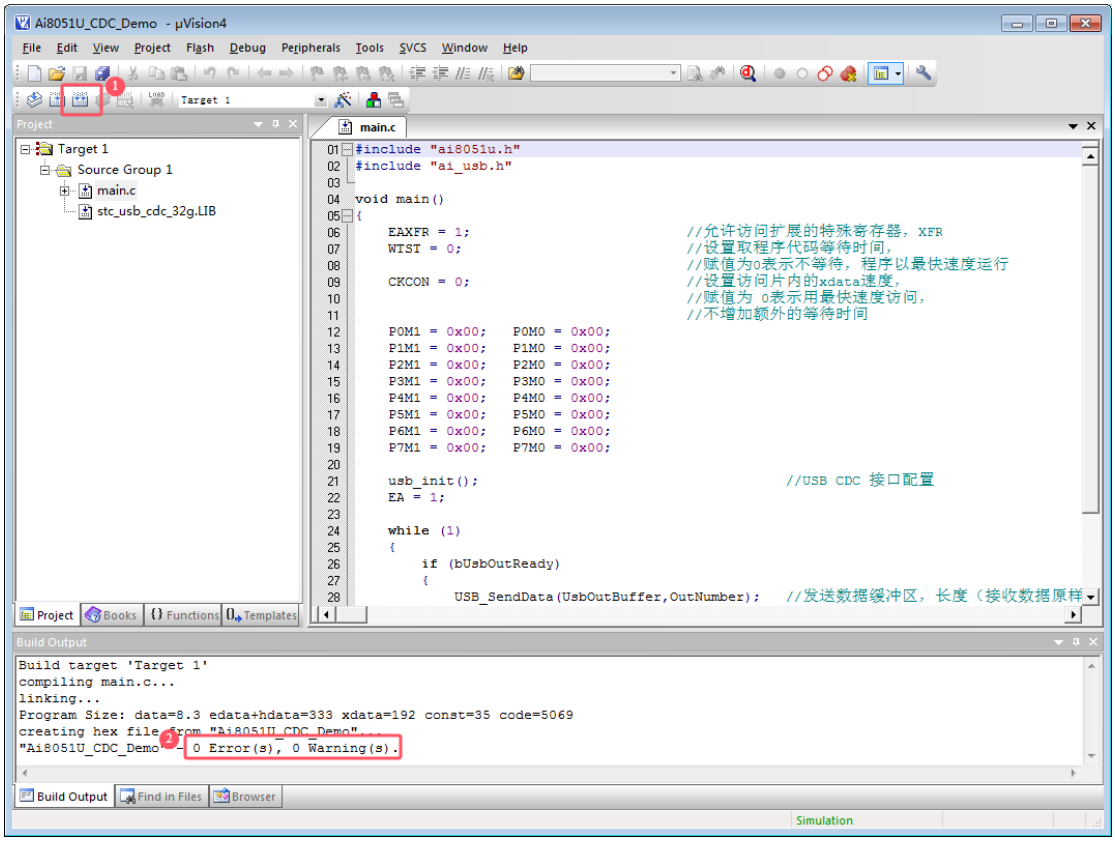
11、在项目配置的“Target”、“Output”和“L251 Misc”这 3 个页面中分别进行如下配置





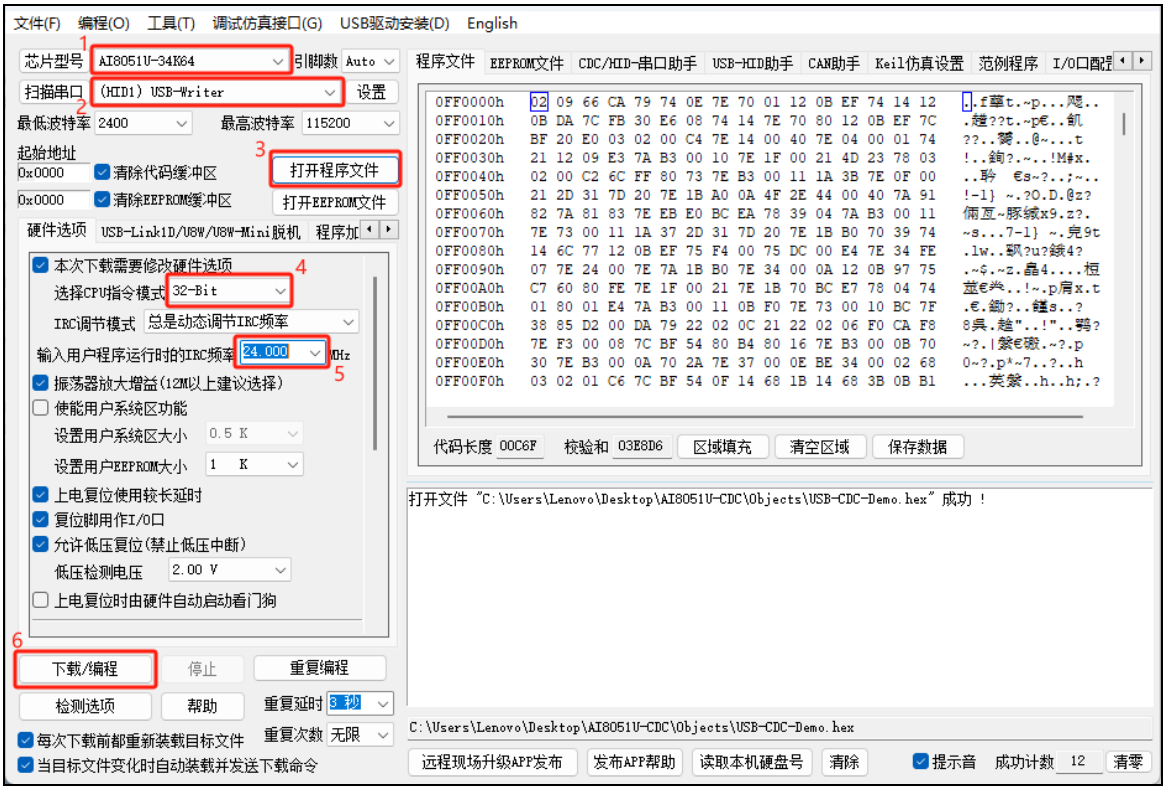


12、项目配置完成后，点击“Rebuild”按钮对代码进行编译链接

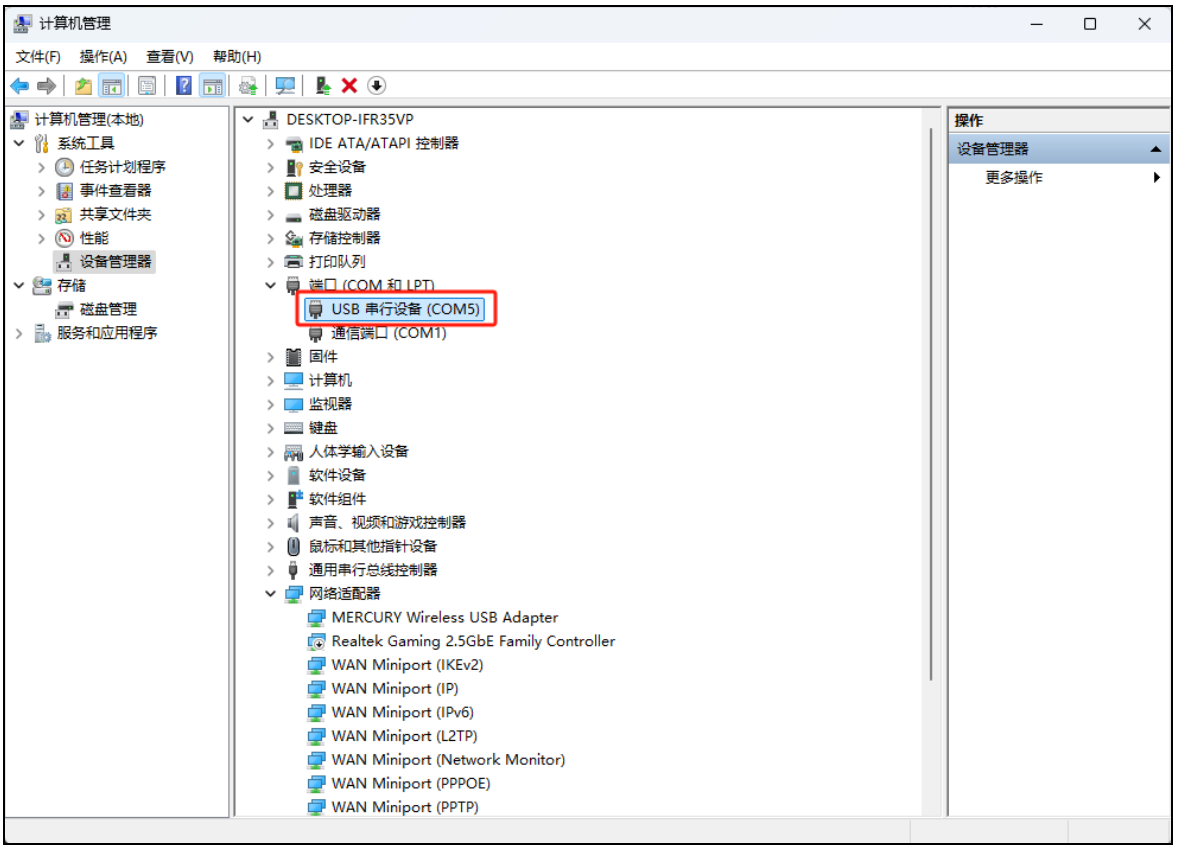


如编译结果为“0 Error(s), 0 Warning(s).”则项目创建成功！

13、使用最新的 ISP 下载软件将 HEX 下载到目标芯片

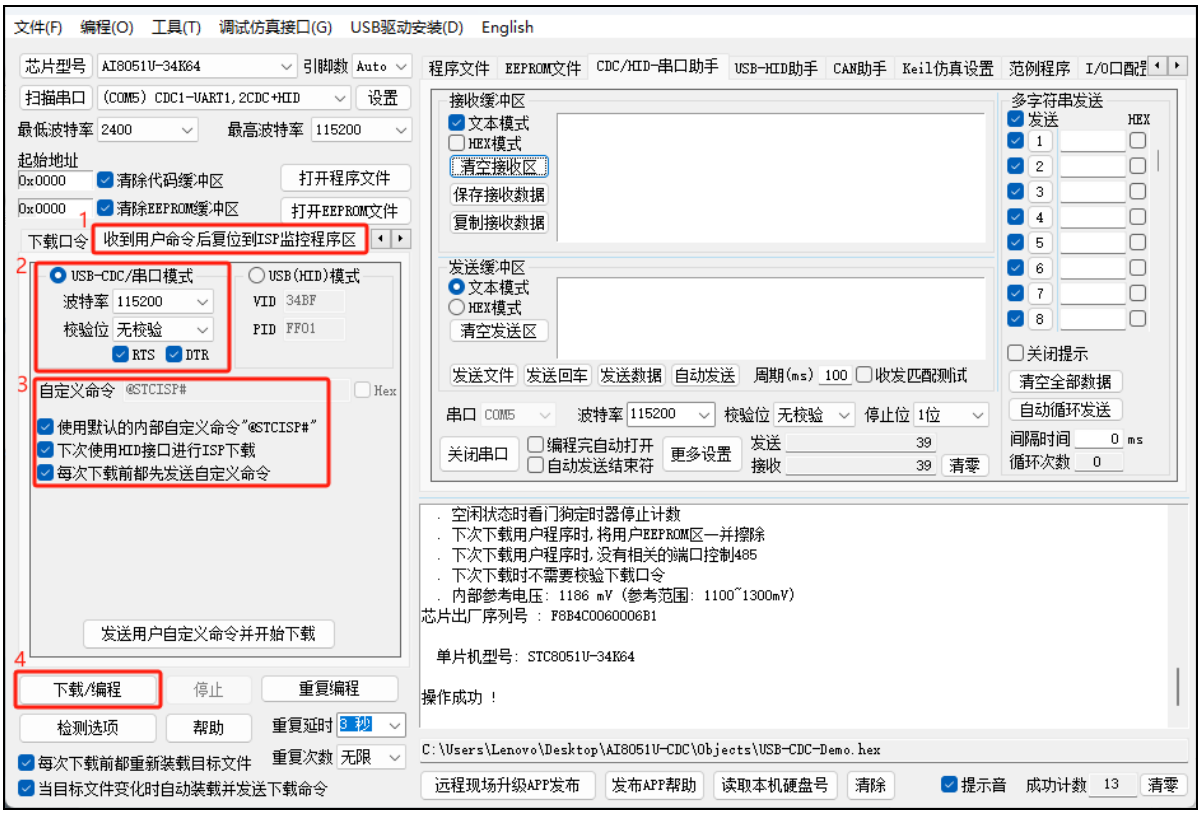


14、下载完成后，系统中就会出现一个 USB-CDC 串口



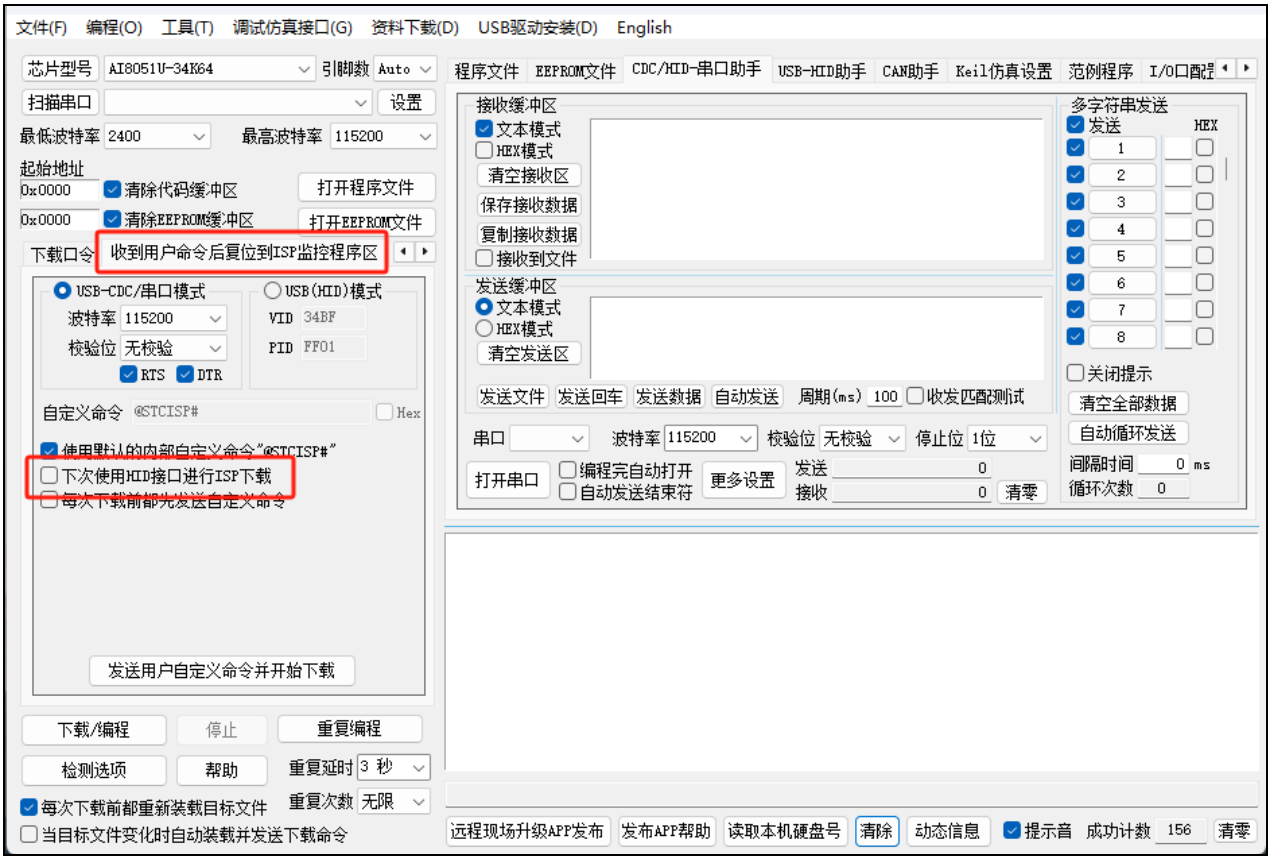
3.7 USB-CDC 虚拟串口实现不停电自动 ISP 下载，及 ISP 软件如何设置

由于在 USB 库函数的代码中已经定义了不停电自动 ISP 下载命令”@STCISP#”，只需要在下载软件的“收到用户命令后复位到 ISP 监控程序区”中的进行如下设置，即可实现不停电自动 ISP 下载功能了。



3.8 关于“下次使用 HID 接口进行 ISP 下载”的选项说明

如果使用串口进行 ISP 下载或者使用 Link1D、Writer1A、U8W 等工具对目标芯片进行 ISP 下载时，请务必去掉“下次使用 HID 接口进行 ISP 下载”的选项，如下图：



3.9　USB 范例程序，查询方式和中断方式各举一例

3.9.1　范例程序 1（查询方式调用 CDC/HID 库函数）

```
#include "ai8051u.h"
#include "ai_usb.h"
/*
如何获得上面这 2 个头文件，如已按照前面介绍的方法：
从 www.STCAI.com 网站，【软件工具 | 工具软件】处，下载安装最新的【AIapp-ISP 软件】，
选择其中的选项卡 【Keil 仿真设置 】，
点击【添加型号和头文件到 Keil 中，并添加仿真器驱动到 Keil 中】按钮，
则 Ai8051U.h 和 AI_usb.h 都会自动复制到 Keil 中，包含进来就可以直接使用，
大家可以打开看下其中的内容，AI_usb.h 和 stc32_stc8_usb.h 是相似的，只需用其中的 1 个头文件，
具体见 Keil_v5\C251\INC\STC 目录，或 Keil_v5\C51\INC\STC 目录。
如将对应的 USB 库文件添加到 项目中，后续用户程序中，USB 如收到命令字符串 "@STCISP#"
USB 库函数就会自动软复位到系统 ISP 程序区，实现 USB 不停电下载，具体见相应部分的说明。
*/

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
    WTST = 0;                           //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，
                                        //不增加额外的等待时间
    P0M0 = 0x00; P0M1 =  0x00;          //设置 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 =  0x00;          //设置 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 =  0x00;          //设置 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 =  0x00;          //设置 P3 口为准双向口模式
    P4M0 = 0x00; P4M1 =  0x00;          //设置 P4 口为准双向口模式
    P5M0 = 0x01; P5M1 =  0x00;          //设置 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 =  0x00;          //设置 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 =  0x00;          //设置 P7 口为准双向口模式

    usb_init();     //调用 USB 初始化函数，不需要立即判断电脑已正确识别到 USB 从设备
    /* USB 型单片机从设备，如需要主动向电脑发送数据，
       在执行 USB_SendData( )函数和 printf_usb( )函数时，
       这两个函数已增加了判断电脑是否已正确识别到 USB 从设备的程序。
       如果电脑要主动发送数据给 USB 从设备，电脑自己会主动判断与 USB 从设备是否已正确连接。
    */

    EA = 1;         //总中断允许位打开
```

```
    while (1)
    {
        if (bUsbOutReady)
                //USB 库中有定义 bUsbOutReady 位变量，查询方式必须调用一次“usb_OUT_done();”
        {
            USB_SendData(UsbOutBuffer,OutNumber);      //接收数据原样返回，用于测试
            usb_OUT_done();
                //当使用查询方式 if(bUsbOutReady)，查询调用 USB 库函数时，在数据处理完成后，
                //必须调用一次“usb_OUT_done();”，这样才能接收下一包数据。
        }
    }
}
```

3.9.2　范例程序 2（中断方式调用 CDC/HID 库函数）

```
#include "ai8051u.h"
#include "ai_usb.h"
/*如何获得上面这 2 个头文件，如已按照前面介绍的方法：
  从 www.STCAI.com 网站，【软件工具 | 工具软件】处，下载安装最新的【AIapp-ISP 软件】，
  选择其中的选项卡 【Keil 仿真设置 】，
  点击【添加型号和头文件到 Keil 中，并添加仿真器驱动到 Keil 中】按钮，
  则 Ai8051U.h 和 AI_usb.h 都会自动复制到 Keil 中，包含进来就可以直接使用，
  大家可以打开看下其中的内容，AI_usb.h 和 stc32_stc8_usb.h 是相似的，只需用其中的 1 个头文件，
  具体见 Keil_v5\C251\INC\STC 目录，或 Keil_v5\C51\INC\STC 目录。
  如将对应的 USB 库文件添加到 项目中，后续用户程序中，USB 如收到命令字符串 "@STCISP#"
  USB 库函数就会自动软复位到系统 ISP 程序区，实现 USB 不停电下载，具体见相应部分的说明。
*/
void usb_callback()
{
    USB_SendData(UsbOutBuffer,OutNumber);                    //接收数据原样返回，用于测试
}
void main()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，
                                                //不增加额外的等待时间
    P0M0 = 0x00; P0M1 =  0x00;                  //设置 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 =  0x00;                  //设置 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 =  0x00;                  //设置 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 =  0x00;                  //设置 P3 口为准双向口模式
    P4M0 = 0x00; P4M1 =  0x00;                  //设置 P4 口为准双向口模式
    P5M0 = 0x01; P5M1 =  0x00;                  //设置 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 =  0x00;                  //设置 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 =  0x00;                  //设置 P7 口为准双向口模式

    usb_init();     //调用 USB 初始化函数，不需要立即判断电脑已正确识别到 USB 从设备
    /* USB 型单片机从设备，如需要主动向电脑发送数据，
       在执行 USB_SendData( )函数和 printf_usb( )函数时，
       这两个函数已增加了判断电脑是否已正确识别到 USB 从设备的程序。
       如果电脑要主动发送数据给 USB 从设备，电脑自己会主动判断与 USB 从设备是否已正确连接。
    */
    set_usb_OUT_callback(usb_callback);       //设置中断回调回调函数
    EA = 1;         //总中断允许位打开
    while (1);
}
```

4 Ai8051U 系列选型简介、特性/价格/管脚图/ISP 下载/编程线路图

4.1 Ai8051U-LQFP48 总体介绍，USB 下载，烧录/仿真 线路图

4.1.1 特性及价格

➢ 选型价格（不需要外部晶振、不需要外部复位，12 位 ADC，15 通道）

单片机型号	工作电压（V）	Flash 程序存储器 10 万次 字节	edata 内部扩展 DATA RAM 可做堆栈或变量 字节	xdata 内部大容量扩展 SRAM 可做变量 字节	EEPROM 10 万次 字节	I/O 口最多数量	传统 I/O 中断（INT0/INT1/INT2/INT3/INT4）并可掉电唤醒	所有 I/O 口均支持高电平 低电平 上升沿 下降沿中断，可掉电唤醒	DMA 8080/6800 接口 TFT 彩屏模块驱动(8 位和 16 位)	RTC 实时时钟 年/月/日/时/分/秒 对内部时钟追频 掉电唤醒	DMA UART 异步串口并可掉电唤醒	DMA USART 同步/异步串口并可掉电唤醒 额外支持两组 SPI/LIN	全速 USB，支持 USB-CDC, USB-HID	DMA I2S 音频总线 掉电唤醒	DMA SPI 并可掉电唤醒	DMA QSPI	DMA I^2C 并可掉电唤醒	TFPU (108MHz)硬件浮点 / 硬件三角 / 反三角函数运算器	MDU32，硬件 32 位乘除法器	定时器/计数器（T0/T1/T2/T3/T4/T11 外部管脚也可掉电唤醒）	144MHz，16 位高级 PWM 定时器 互补对称死区控制	传统的 PCA/CCP/PWM（可当外部中断并可掉电唤醒） 支持上升沿中断、下降沿中断以及边沿中断	掉电唤醒专用定时器	DMA 15 路高速 ADC（8 路 PWM 可当 8 路 D/A 使用）	比较器（可当 1 路 A/D，可作外部掉电检测）	内部低压检测中断并可掉电唤醒	看门狗 复位定时器	内部高可靠复位（可选复位门槛电压）	内部高精准时钟（42MHz 以下可调），满足串口通信 追频	可对外输出时钟及复位	程序加密后传输（防拦截）	可设置下次更新程序需口令	支持 RS485 下载	支持硬件 USB 直接下载和硬件 USB 仿真	本身就可在线仿真	价格及封装 LQFP48	价格及封装 LQFP44	价格及封装 PDIP40	供货信息
Ai8051U-34K64	1.9-5.5	64K	2K	32K	IAP	45	有	有	有	有	2	2	有	有	3	有	有	有	有	有	有	6	8	3	有	12 位	有	有	有	4 级	有	是	有	是	是	是	是	¥1.9	¥3.9	¥4.0	
Ai8051U-34K32	1.9-5.5	32K	2K	32K	32K	45	有	有	有	有	2	2	有	有	3	有	有	有	有	有	有	6	8	3	有	12 位	有	有	有	4 级	有	是	有	是	是	是	是				
Ai8051U-34K16	1.9-5.5	16K	2K	32K	48K	45	有	有	有	有	2	2	有	有	3	有	有	有	有	有	有	6	8	3	有	12 位	有	有	有	4 级	有	是	有	是	是	是	是				

➢ **内核**

- ✓ 超高速 32 位 8051 内核（1T），比传统 8051 约快 70 倍以上
- ✓ 4 级中断优先级
- ✓ 支持在线仿真

➢ **工作电压**

- ✓ 1.9V～5.5V（当工作温度低于-40℃时，工作电压不得低于 3.0V）

➢ **工作温度**

- ✓ -20℃~65℃（内部高速 IRC 温漂-0.76%~+0.98%）
- ✓ -40℃~85℃（内部高速 IRC 温漂±1.3%）
- ✓ -40℃~125℃（内部高速 IRC 温漂±3%，当温度高于 85℃时请使用外部 35MHz 及以下的晶振）

扫码去微信小商城

➢ **Flash 存储器**

- ✓ 最大 64K 字节 FLASH 程序存储器（ROM），用于存储用户代码
- ✓ 支持用户可配置部分 FLASH 作为 DATA FLASH/EEPROM 使用，512 字节单页擦除，
  典型单个扇区擦除次数可达 10 万次，用户保守可当 2 万次使用
- ✓ 支持硬件 USB 直接下载和普通串口下载
- ✓ **Flash 数据保持时间：100 年@25℃，20 年@105℃，10 年@125℃。由于汽车及单片机的平均温度不可能长时间大于 105℃，从严考虑，Flash 数据保持时间可达 20 年以上，不保证 100 年。**

- **SRAM，共 34K 字节**
  - ✓ 2K 字节内部 SRAM（edata）
  - ✓ 32K 字节内部扩展 RAM（内部 xdata）

- **时钟控制**
  - ✓ 内部高精度、高稳定的高速 IRC（ISP 编程时可进行上下调整）
    - ⊕ 误差±0.3%（常温下 25℃）
    - ⊕ -1.35%～+1.30%温漂（全温度范围，-40℃～85℃）
    - ⊕ -0.76%～+0.98%温漂（温度范围，-20℃～65℃）
  - ✓ 内部 32KHz 低速 IRC（为了低功耗，省去了温度补偿和电压补偿电路，误差较大）
  - ✓ 外部晶振（4MHz～42MHz）和外部时钟，有专门的外部时钟干扰内部电路，可软件启动
  - ✓ 内部 PLL 输出时钟（注：PLL 输出的 144MHz/96MHz 可独立作为高速 PWM 和高速 SPI 的时钟源）
    用户可自由选择上面的 4 种时钟源
  - ✓ **注：内部 PLL 输出时钟频率最高可达 144MHz / 148MHz，若高速 PWM 使用 PLL 作为时钟源，**
    125℃时建议高速 PWM 的输入时钟控制在 138MHz 以下，
    105℃时高速 PWM 的输入时钟可用 144MHz

（芯片上电工作过程：上电复位/复位脚复位/看门狗复位/低压检测复位时，芯片默认从 ISP 系统程序开始执行代码，此时固定使用内部 24MHz 的高速 IRC 时钟，当需要下载用户程序且下载完成后复位到用户程序区或者不需要下载直接复位到用户程序区时，默认会使用上次用户下载时所调节的高速 IRC 时钟，如果用户程序需要使用外部高速晶振、外部 32.768KHz 晶振或者内部 30KHz 低速 IRC，则需要用户软件先启动相应的时钟，然后通过设置 CLKSEL 寄存器进行切换）

- **复位**
  - ✓ 硬件复位
    - ⊕ 上电复位，复位电压值为 1.7V～1.9V。（在芯片未使能低压复位功能时有效）
    - ⊕ 复位脚复位，出厂时 P4.7 默认为 I/O 口，ISP 下载时可将 P4.7 管脚设置为复位脚（注意：当设置 P4.7 管脚为复位脚时，复位电平为低电平）
    - ⊕ 看门狗溢出复位
    - ⊕ 低压检测复位，提供 4 级低压检测电压：2.0V、2.4V、2.7V、3.0V。
  - ✓ 软件复位
    - ⊕ 软件方式写复位触发寄存器

- **中断**
  - ✓ 中断源：INT0、INT1、INT2、INT3、INT4、定时器 0、定时器 1、定时器 2、定时器 3、定时器 4、USART1、USART2、UART3、UART4、ADC 模数转换、LVD 低压检测、SPI、I^2C、比较器、PWMA、PWMB、USB、TFT 彩屏接口中断、RTC 实时时钟、所有的 I/O 中断（8 组）、串口 1 的 DMA 接收和发送中断、串口 2 的 DMA 接收和发送中断、串口 3 的 DMA 接收和发送中断、串口 4 的 DMA 接收和发送中断、I2C 的 DMA 接收和发送中断、SPI 的 DMA 中断、ADC 的 DMA 中断、LCD 驱动的 DMA 中断以及存储器到存储器的 DMA 中断。
  - ✓ 提供 4 级中断优先级

- **数字外设**
  - ✓ 6 个 16 位定时器：定时器 0、定时器 1、定时器 2、定时器 3、定时器 4、定时器 11，其中定时器 0 的模式 3 具有 NMI（不可屏蔽中断）功能，定时器 0 和定时器 1 的模式 0 为 16 位自动重载模式。定时器 11：可选时钟源
  - ✓ 2 个高速同步/异步串口：串口 1（USART1）、串口 2（USART2），波特率时钟源最快可为 FOSC/4。支持同步串口模式、异步串口模式、SPI 模式、LIN 模式、红外模式（IrDA）、智能卡模式（ISO7816）
  - ✓ 2 个高速异步串口：串口 3、串口 4，波特率时钟源最快可为 FOSC/4

- ✓ 2 组高级 PWM，可实现 8 通道（4 组互补对称）带死区的控制的 PWM，并支持外部异常检测功能
- ✓ 3 组 16 位 CCP/PCA/PWM 模块：CCP0、CCP1、CCP2，可用于捕获、高速脉冲输出，及 6/7/8/10 位的 PWM 输出
- ✓ SPI：3 组硬件 SPI（一组独立 SPI，两组 USART 的 SPI 模式）支持主机模式和从机模式以及主机/从机自动切换（注：3 组 SPI 均支持 DMA）
- ✓ QSPI：支持单线、双线和四线模式
- ✓ I^2C：支持主机模式和从机模式
- ✓ ICE：硬件支持仿真
- ✓ RTC：支持年、月、日、时、分、秒、次秒（1/128 秒），并支持时钟中断和一组闹钟
- ✓ USB：USB2.0/USB1.1 兼容全速 USB，6 个双向端点，支持 4 种端点传输模式（控制传输、中断传输、批量传输和同步传输），每个端点拥有 64 字节的缓冲区
- ✓ I2S：音频总线
- ✓ MDU32：硬件 32 位乘除法器（包含 32 位除以 32 位、32 位乘以 32 位）
- ✓ TFPU：单精度浮点运算器（支持浮点加、减、乘、除以及正弦、余弦、正切和反正切等运算）
- ✓ I/O 口中断：所有的 I/O 均支持中断，每组 I/O 中断有独立的中断入口地址，所有的 I/O 中断可支持 4 种中断模式：高电平中断、低电平中断、上升沿中断、下降沿中断。I/O 口中断可以进行掉电唤醒，且有 4 级中断优先级。
- ✓ LCD 驱动模块：支持 8080 和 6800 两种接口以及 8 位和 16 位数据宽度
- ✓ DMA：支持 SPI 移位接收数据到存储器、SPI 移位发送存储器的数据、I2C 发送存储器的数据、I2C 接收数据到存储器、串口 1/2/3/4 接收数据到的存储器、串口 1/2/3/4 发送存储器的数据、ADC 自动采样数据到存储器（同时计算平均值）、LCD 驱动发送存储器的数据、以及存储器到存储器的数据复制
- ✓ 硬件数字 ID：支持 32+32 字节

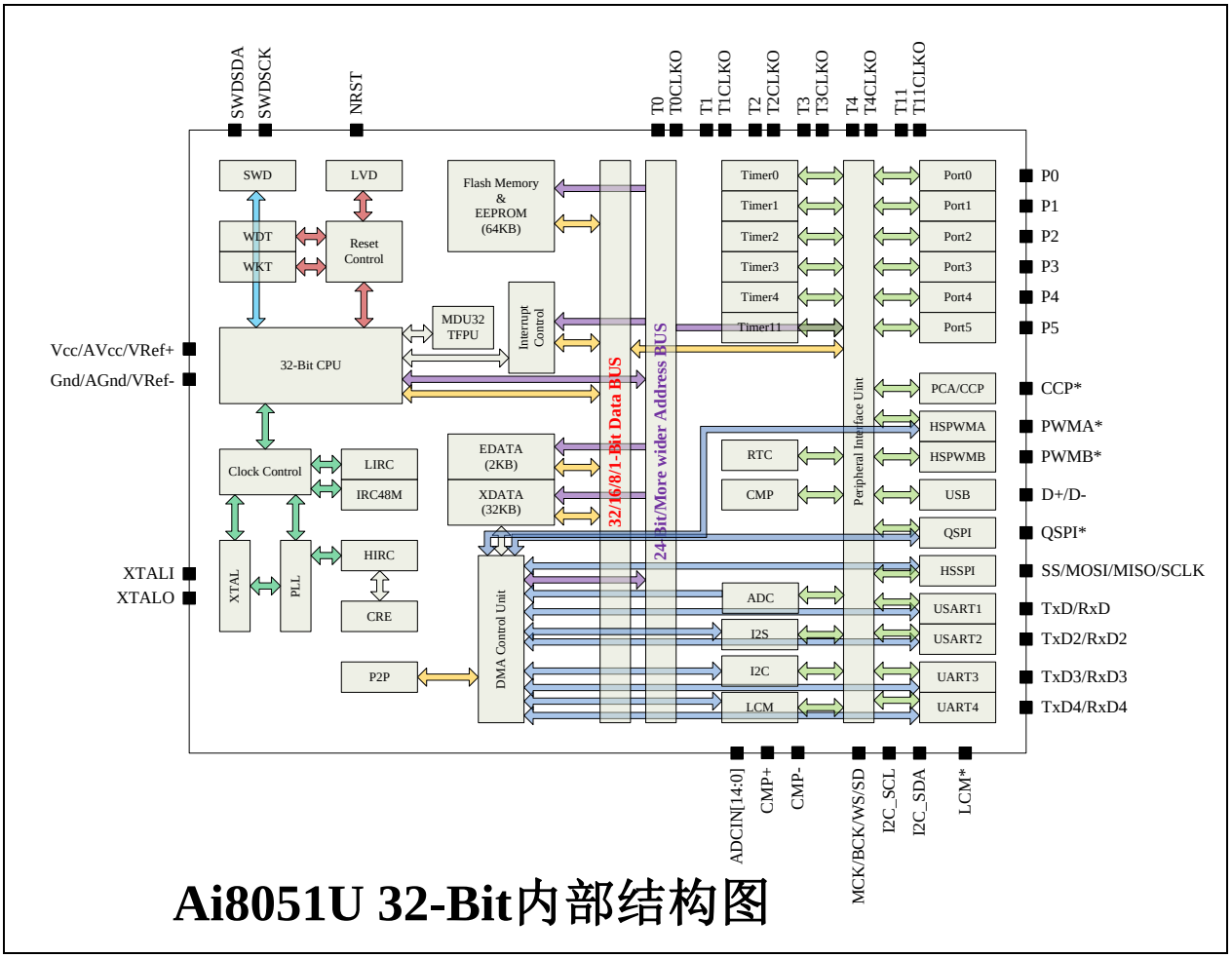
- ➢ 模拟外设
  - ✓ ADC：超高速 ADC，支持 12 位高精度 15 通道（通道 0～通道 14）的模数转换，ADC 的通道 15 用于测试内部参考电压（芯片在出厂时，内部参考电压调整为 1.19V，误差±1%）
  - ✓ 比较器：一组比较器

- ➢ GPIO
  - ✓ 最多可达 46 个 GPIO：P0.0~P0.7、P1.0~ P1.7、P2.0~P2.7、P3.0~P3.7、P4.0~P4.7、P5.0~P5.3、P5.6~P5.7
  - ✓ 所有的 GPIO 均支持如下 4 种模式：准双向口模式、强推挽输出模式、开漏模式、高阻输入模式
  - ✓ 除 P3.0 和 P3.1 外，其余所有 I/O 口上电后的状态均为高阻输入状态，用户在使用 I/O 口时必须先设置 I/O 口模式
  - ✓ 另外每个 I/O 均可独立使能内部 10K 上拉电阻和 10K 下拉电阻

- ➢ **封装**
  - ✓ LQFP48、LQFP44、PDIP40

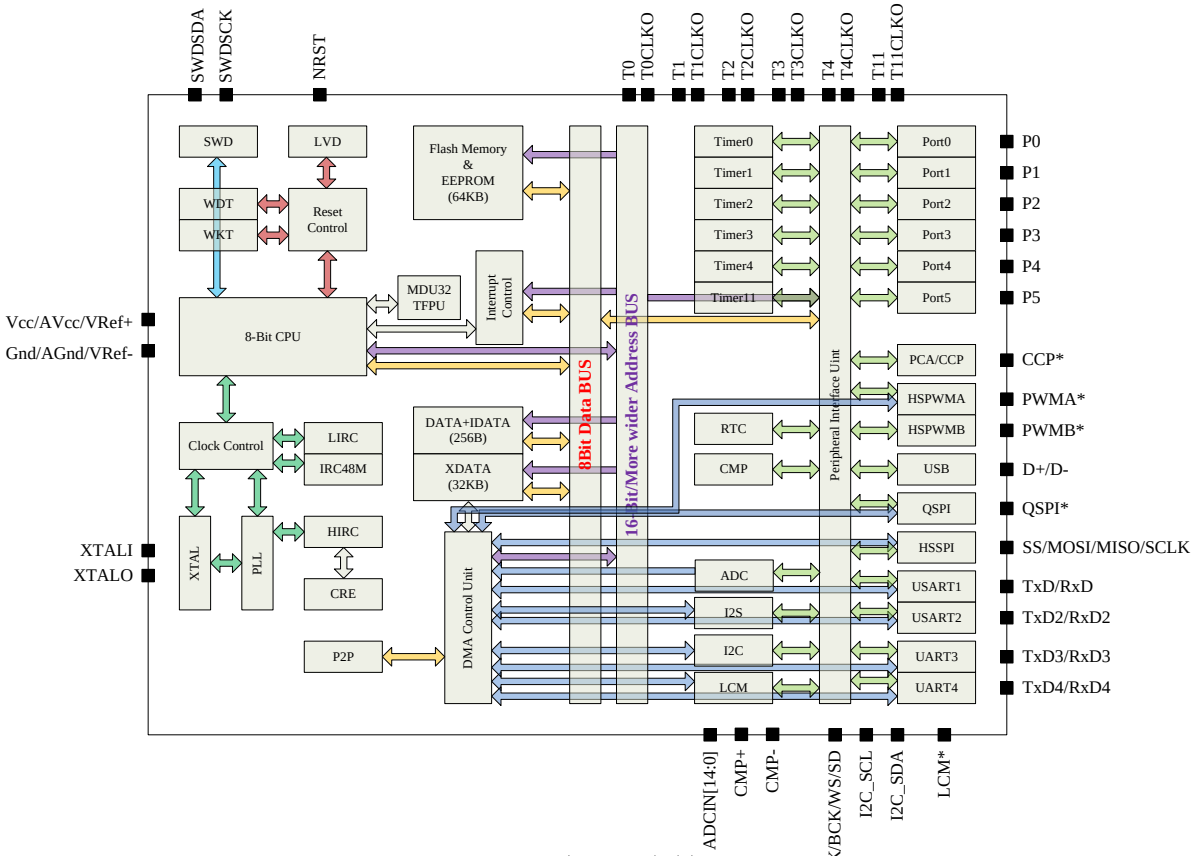
4.1.2 Ai8051U 系列内部结构图

4.1.2.1 Ai8051U-32Bit 内部结构图



Ai8051U 32-Bit内部结构图

4.1.2.2 Ai8051U-8Bit 内部结构图



Ai8051U 8-Bit内部结构图

4.1.3 LQFP48/QFN48 管脚图，5V 线路，USB-ISP 下载/烧录/仿真

备注：上图中红色管脚部分是 2024 年 10 月 15 日之后生产的芯片才有。

现在带硬件 USB 的 MCU 支持用硬件 USB 下载，因为用的是 USB-HID 通信协议，不需要安装任何驱动。只要 USB 鼠标、USB 键盘能工作，USB-HID 驱动就是好的，不要安装 USB-HID 驱动，免驱。

在 D-/P3.0，D+/P3.1 与 PC-USB 端口连接好的状况下，USB-ISP 下载程序有如下三种模式：

【USB 下载方法一，P3.2 按键，再结合停电上电下载】

1、按下板子上的 P3.2/INT0 按键，就是 P3.2 接地

2、给目标芯片重新上电，不管之前是否已通电。

===电子开关是按下停电后再松开就是上电

等待电脑端 ISP 下载软件中自动识别出"(HID1) USB Writer"后，就与 P3.2 状态无关了，这时可以松开 P3.2 按键（P3.2 在用户程序区可做其它任意用途）

===传统的机械自锁紧开关是按上来停电，按下去是上电

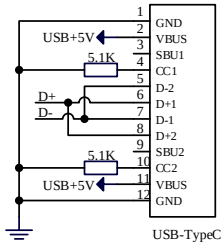
3、点击电脑端下载软件中的【下载/编程】按钮（注意：USB 下载与串口下载的操作顺序不同）

下载进行中，几秒钟后，提示下载成功！

【USB 下载方法二，复位管脚低电平复位下载】

USB 连接好并已上电的情况下，外部按键复位也可进入 USB 下载模式，注意：P4.7-nRST 出厂时默认是 P4.7-I/O 功能，要改为复位功能，需 ISP 烧录时取消设置复位脚用作 I/O 口，停电一次再上电才生效，程序区中用户程序也可改为复位脚或 I/O，这个立即生效。

比较一下流行的 USB-TypeC接口：



1、按下 P4.7-nRST 外接的低电平复位按键复位 MCU，
松开复位键，MCU 从系统程序区启动，判断是否要下载用户程序，
等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后

2、点击电脑端下载软件中的【下载/编程】按钮
下载进行中，几秒钟后，提示下载成功！

备注：如果目标 MCU 的用户程序区已有 USB 通讯程序（含 CDC、HID 等），则通过复位脚复位进入 USB 下载模式不被支持，只能通过停电上电冷启动下载或者从用户程序区软复位到系统区下载

【USB 下载方法三，从用户程序区软复位到系统区下载】

USB 连接好并已上电的情况下，从用户程序区软复位到系统区也可进入 USB 下载模式

1、在用户程序区运行软复位到系统区的程序，就是 IAP_CONTR 寄存器送 60H
等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后

2、点击电脑端下载软件中的【下载/编程】按钮
下载进行中，几秒钟后，提示下载成功！

USB 下载 注意事项：

拔插 USB 插头不能代替上面线路图中的电源开关。正常操作步骤如下：

USB 的【Gnd，D+，D-】接好的情况下，按下 P3.2 按键接地，再通过正常的电源开关给 MCU 供电或重新供电，让 MCU 冷启动进入系统程序区，判断是否需要等待电脑端 USB 下载程序。

拔插 USB 插头代替电源开关不能每次都能成功进行 USB 下载的原因：

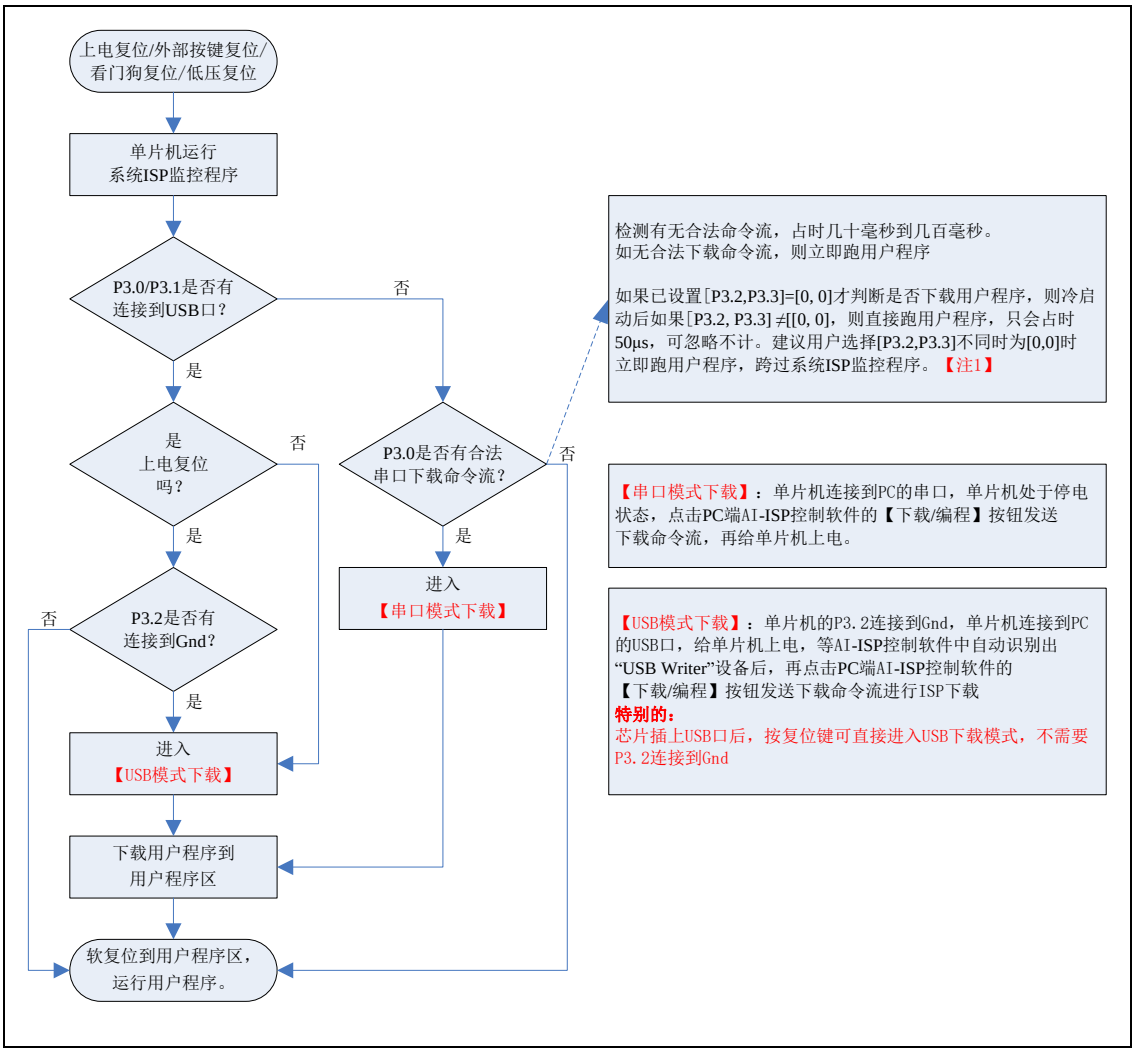
拔插 USB 插头，如【Gnd，USB+5V】已接触好，已供电，而【D+，D-】有一个甚至两个信号线还没有接触好，MCU 已上电，开始跑系统区程序时，发现 USB 还没接触好，则会从系统区软复位到用户程序区跑用户程序，不再进入等待 USB 下载模式，本次就无法顺利进行 USB 下载。

很多人经过多次插拔 USB，才能碰到 1 次【D+，D-】接触好的情况下，【Gnd，USB+5V】才开始接触好，才开始供电，才能成功进入 USB 下载。插 USB 插头代替供电，不能保证【Gnd，D+，D-，USB+5V】的接触顺序，所以，必须使用正常的电源开关，才能确保每次下载都能成功。

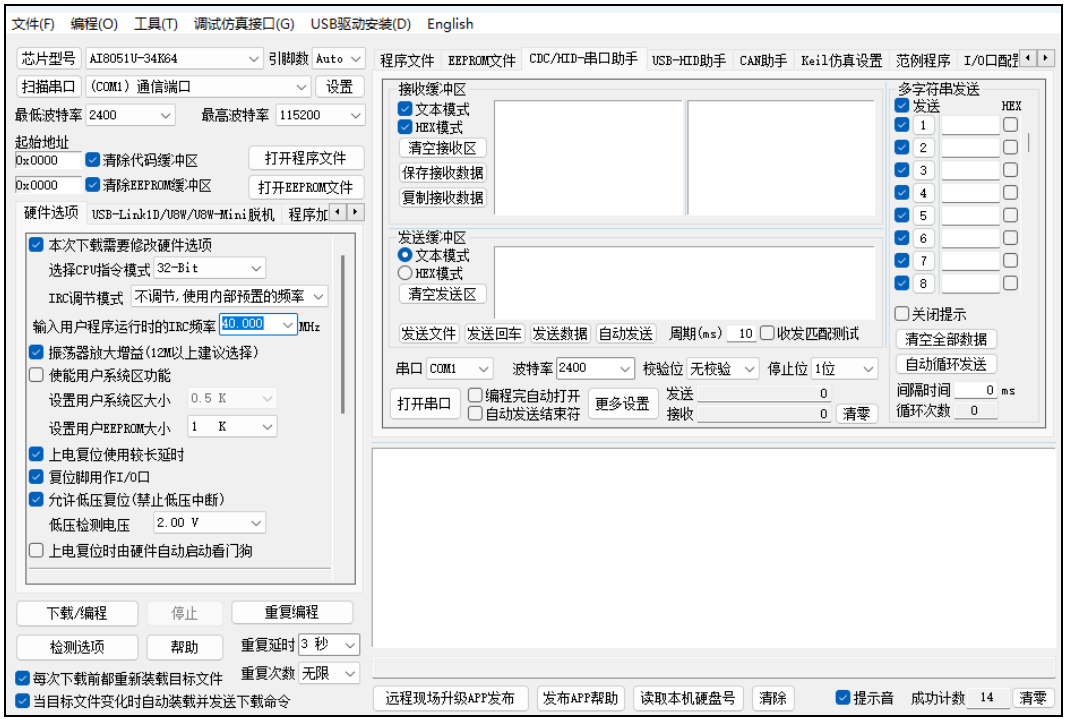
关于 I/O 的注意事项：

1、 P3.0 和 P3.1 口上电后的状态为弱上拉/准双向口模式
2、 除 P3.0 和 P3.1 外，其余所有 I/O 口上电后的状态均为高阻输入状态，用户在使用 I/O 口前必须先设置 I/O 口模式
3、 芯片上电时，若 P3.0 和 P3.1 同时为低电平，P3.2 口会短时间开启内部 4K 上拉，用以读取 P3.2 口外部状态来判断是否需要进入 USB 下载模式
4、 芯片上电时如果不需要使用 USB 进行 ISP 下载，P3.0/P3.1/P3.2 这 3 个 I/O 口不能同时为低电平，否则会进入 USB 下载模式而无法运行用户代码
5、 当使用 P4.7 当作复位脚时，这个端口内部的 4K 上拉电阻会一直打开；但 P4.7 做普通 I/O 口时，基于这个 I/O 口与复位脚共享管脚的特殊考量，端口内部的 4K 上拉电阻依然会打开大约 6.5 毫秒时间，再自动关闭（当用户的电路设计需要使用 P4.7 口驱动外部电路时，请务必考虑上电瞬间会有 6.5 毫秒时间的高电平的问题）

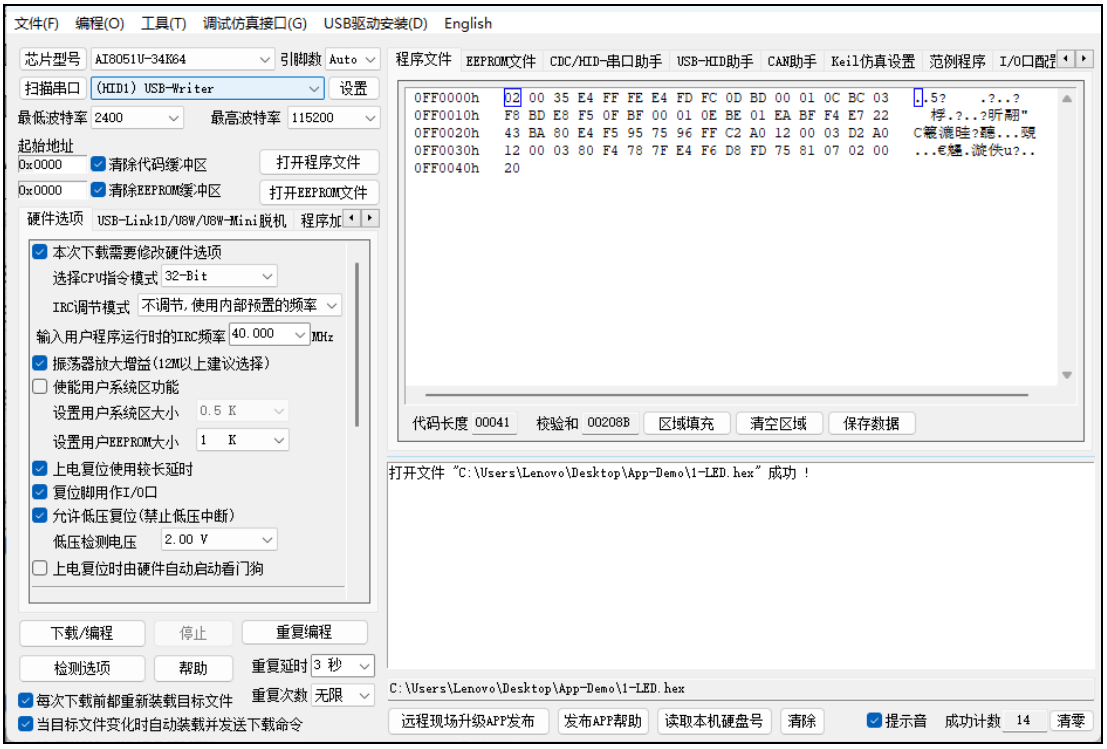
ISP 下载流程图（硬件/软件模拟 USB+串口模式）：



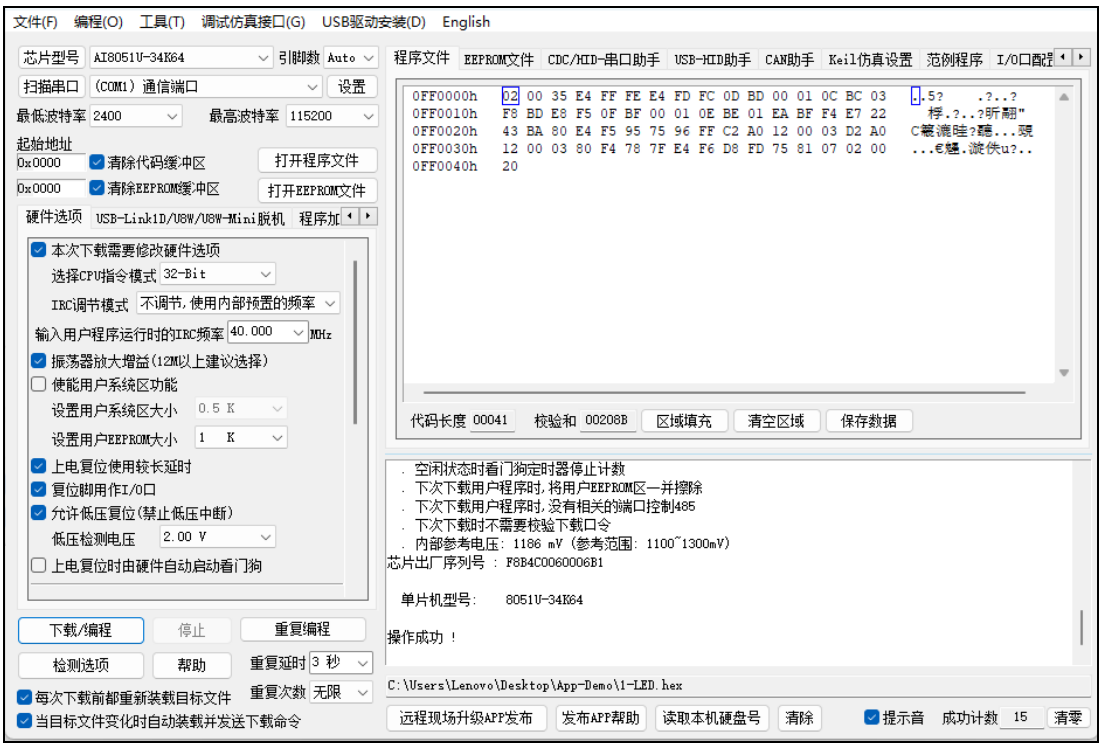
打开 AIapp-ISP-V6.94S 以上版本软件，如下图所示：



选择好对应的正确型号，打开要烧录的文件
将 USB 数据线，如前面的原理图连接好，
注意是【USB+5V，D-，D+，Gnd】USB 数据线，不是【USB+5V，NC，NC，Gnd】USB 电源线
在 P3.2 接地按键按下的状态下，给 MCU 上电，或重新上电。
则 AIapp-ISP 软件出现如下显示：



点击“下载/编程”按钮，
则会如下图显示：正在下载用户代码，操作成功！



如上硬件 USB，ISP 下载/编程 烧录成功。

QSPI

IO0	IO1	IO2	IO3	SCLK	NCS
P1.5	P1.6	P1.3	P1.2	P1.7	P1.4
P4.1	P4.2	P5.2	P5.3	P4.3	P4.0
P2.5	P2.6	P4.6	P4.5	P2.7	P4.7

独立SPI （MOSI和MISO可切换）

SS	MOSI	MISO	SCLK
P1.4	P1.5	P1.6	P1.7
P2.4	P2.5	P2.6	P2.7
P4.0	P4.1	P4.2	P4.3
P3.5	P3.4	P3.3	P3.2

USART1_SPI （MOSI和MISO不可切换）

SS	MOSI	MISO	SCLK
P1.4	P1.5	P1.6	P1.7
P2.4	P2.5	P2.6	P2.7
P4.0	P4.1	P4.2	P4.3
P3.5	P3.4	P3.3	P3.2

USART2_SPI （MOSI和MISO不可切换）

SS	MOSI	MISO	SCLK
P1.4	P1.5	P1.6	P1.7
P2.4	P2.5	P2.6	P2.7
P4.0	P4.1	P4.2	P4.3
P3.5	P3.4	P3.3	P3.2

同步/异步串口USART1, USART2可做异步串口或SPI, 分时复用另有独立的SPI共可实现3组SPI

UART1

RxD	TxD
P3.0	P3.1
P3.6	P3.7
P1.6	P1.7
P4.3	P4.4

UART2

RxD2	TxD2
P1.2	P1.3
P4.2	P4.3

UART3

RxD3	TxD3
P0.0	P0.1
P5.0	P5.1

CMP(比较器)

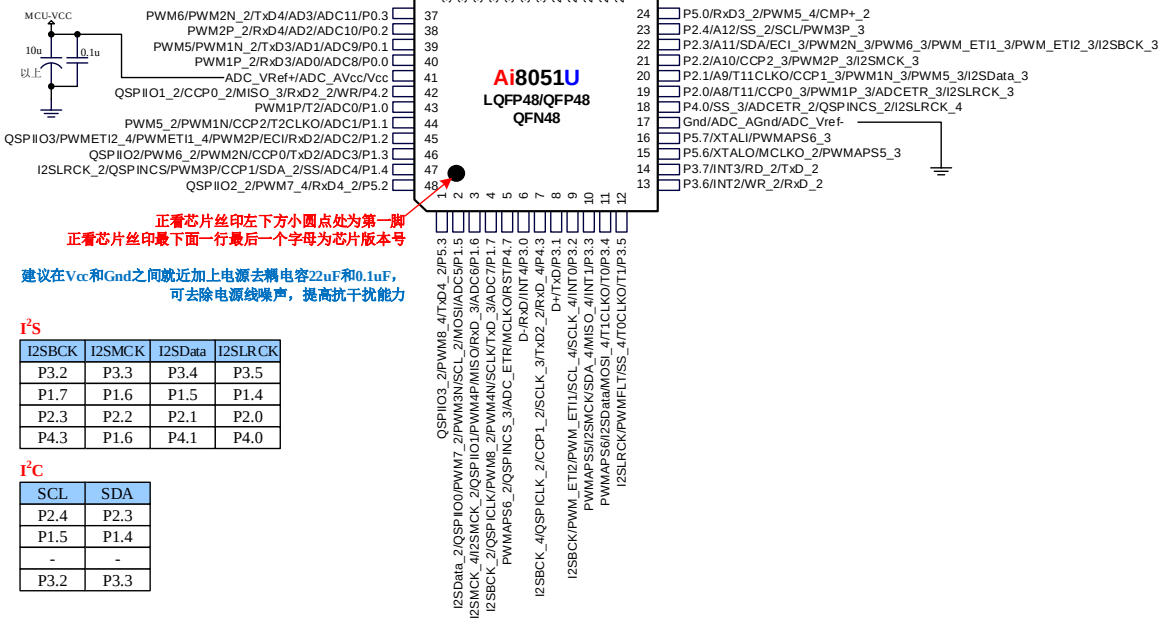
CMP+	CMP-
P4.6	P4.4
P5.0	1.19V
P5.1	-
ADCIN	-

CMPO

CMPO
P4.5
P4.1

UART4

RxD4	TxD4
P0.2	P0.3
P5.2	P5.3



I²S

I2SBCK	I2SMCK	I2SData	I2SLRCK
P3.2	P3.3	P3.4	P3.5
P1.7	P1.6	P1.5	P1.4
P2.3	P2.2	P2.1	P2.0
P4.3	P1.6	P4.1	P4.0

I²C

SCL	SDA
P2.4	P2.3
P1.5	P1.4
-	-
P3.2	P3.3

USB Link1D工具：支持全自动停电-上电在线下载 / 脱机下载 / 仿真

【应用场景一：从本工具给目标系统供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)下载编程进行中，数秒后提示下载编程成功，目标MCU会复位到用户程序区自动跑用户程序。

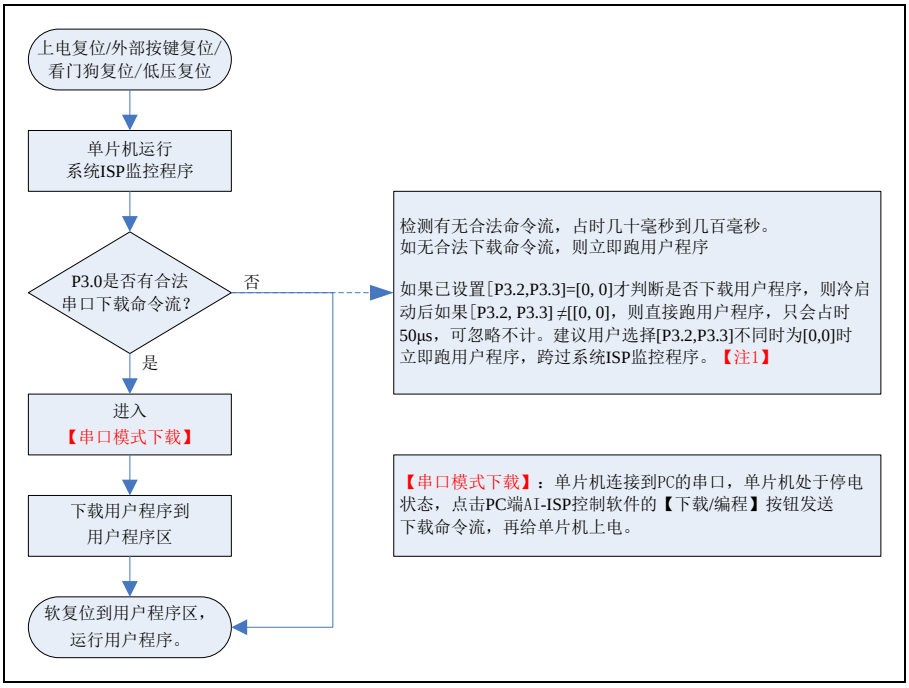
部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P4.7/nRST变成复位脚

备注：上图中红色管脚部分是 2024 年 10 月 15 日之后生产的芯片才有。

USB 转串口/TTL，全自动 ISP 下载步骤：

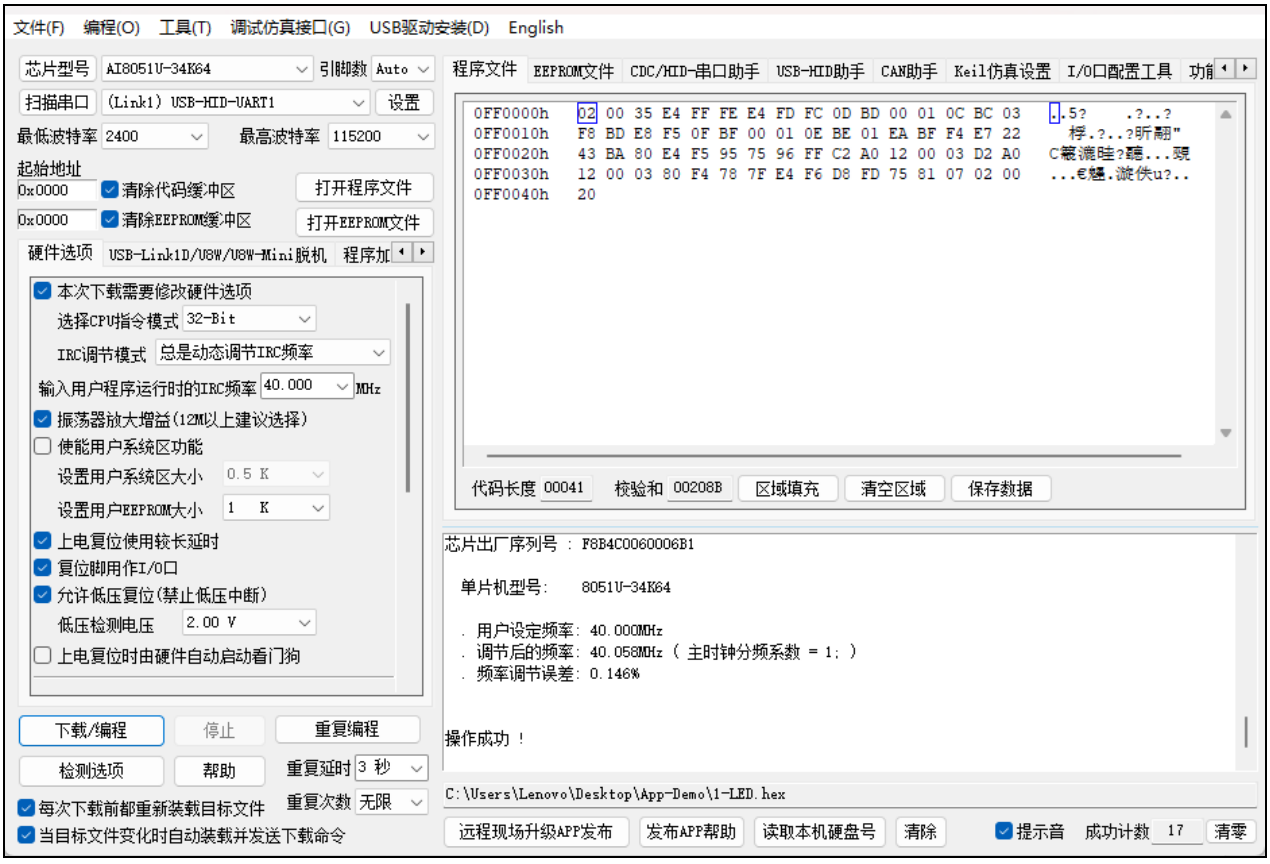
1、按照如图所示的连接方式将 USB-Link1D 和目标芯片连接
2、点击 ISP 下载软件中的“下载/编程”按钮
3、开始 ISP 下载
（注意：若是使用 USB-Link1D 给目标系统供电，目标系统的总电流不能大于 200mA，否则会导致下载失败。）

ISP 下载流程图（串口下载模式）



将【MCU-VCC，P3.0，P3.1，Gnd】，如前面的原理图连接到 USB-Link1D 工具;
将 USB-Link1D 全自动烧录工具，通过 USB 数据线连接到电脑，【USB+5V，D-，D+，Gnd】
打开 AIapp-ISP-V6.94S 以上版本软件
则 AIapp-ISP 软件显示如下：

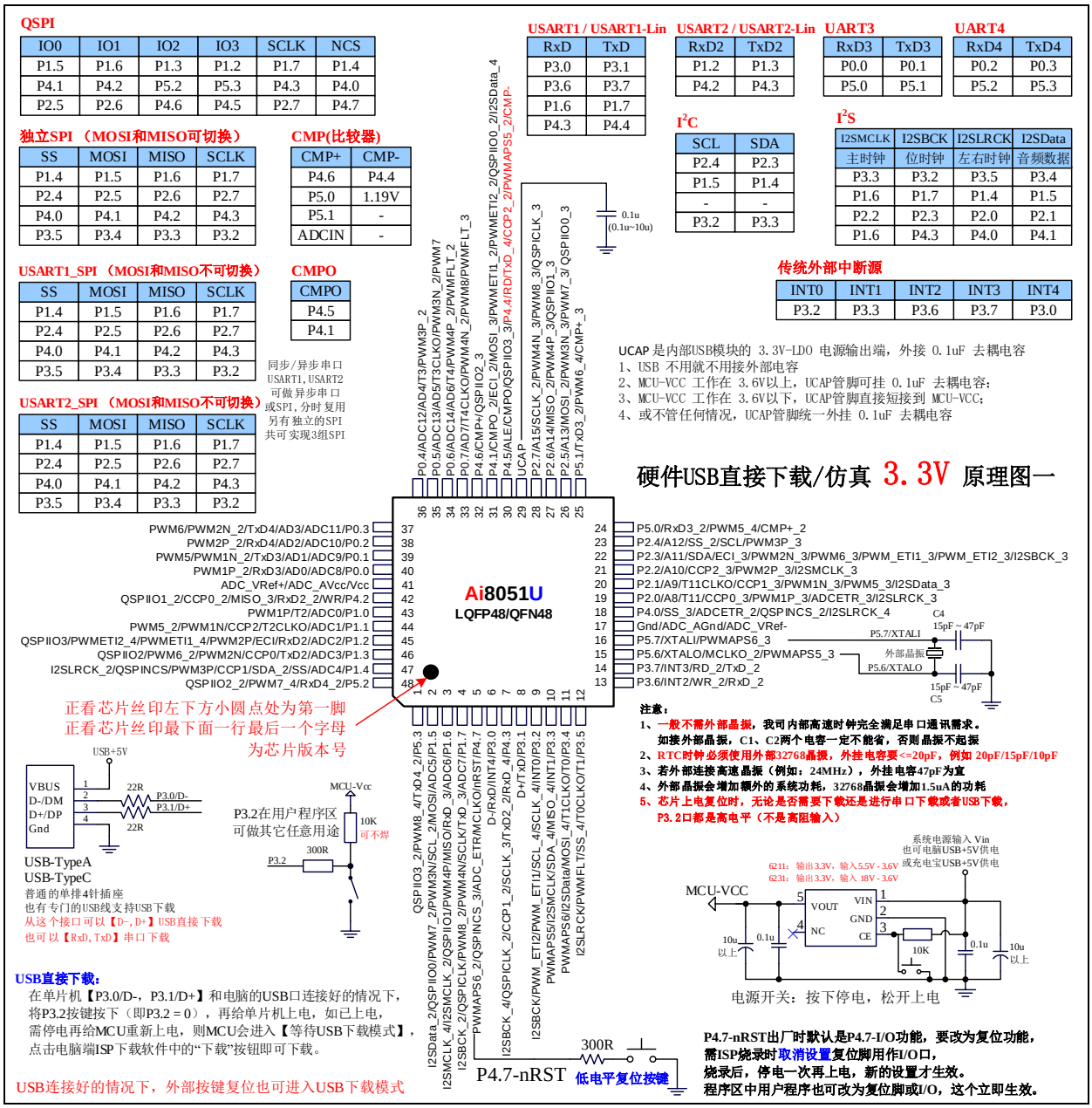
选择好对应的正确型号，打开要烧录的文件
点击“下载/编程”按钮，全自动烧录
则 AIapp-ISP 软件出现如下显示：



关于 I/O 的注意事项：

1、 P3.0 和 P3.1 口上电后的状态为弱上拉/准双向口模式
2、 除 P3.0 和 P3.1 外，其余所有 I/O 口上电后的状态均为高阻输入状态，用户在使用 I/O 口前必须先设置 I/O 口模式
3、 芯片上电时，若 P3.0 和 P3.1 同时为低电平，P3.2 口会短时间开启内部 4K 上拉，用以读取 P3.2 口外部状态来判断是否需要进入 USB 下载模式
4、 芯片上电时如果不需要使用 USB 进行 ISP 下载，P3.0/P3.1/P3.2 这 3 个 I/O 口不能同时为低电平，否则会进入 USB 下载模式而无法运行用户代码
5、 当使用 P4.7 当作复位脚时，这个端口内部的 4K 上拉电阻会一直打开；但 P4.7 做普通 I/O 口时，基于这个 I/O 口与复位脚共享管脚的特殊考量，端口内部的 4K 上拉电阻依然会打开大约 6.5 毫秒时间，再自动关闭（当用户的电路设计需要使用 P4.7 口驱动外部电路时，请务必考虑上电瞬间会有 6.5 毫秒时间的高电平的问题）

4.1.4 LQFP48/QFN48 管脚图，3.3V 线路一，USB-ISP 下载/仿真



备注：上图中红色管脚部分是 2024 年 10 月 15 日之后生产的芯片才有。

现在带硬件 USB 的 MCU 支持用硬件 USB 下载，因为用的是 USB-HID 通信协议，不需要安装任何驱动。只要 USB 鼠标、USB 键盘能工作，USB-HID 驱动就是好的，不要安装 USB-HID 驱动，免驱。

在 D-/P3.0，D+/P3.1 与 PC-USB 端口连接好的状况下，USB-ISP 下载程序有如下三种模式：

【USB 下载方法一，P3.2 按键，再结合停电上电下载】

1、按下板子上的 P3.2/INT0 按键，就是 P3.2 接地

2、给目标芯片重新上电，不管之前是否已通电。

===电子开关是按下停电后再松开就是上电

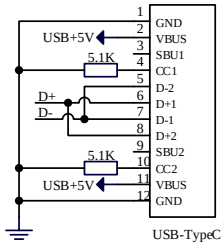
等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后，就与 P3.2 状态无关了，这时可以松开 P3.2 按键（P3.2 在用户程序区可做其它任意用途）

===传统的机械自锁紧开关是按上来停电，按下去是上电

3、点击电脑端下载软件中的【下载/编程】按钮（注意：USB 下载与串口下载的操作顺序不同）

下载进行中，几秒钟后，提示下载成功！

比较一下流行的 **USB-TypeC**接口：



【USB 下载方法二，复位管脚低电平复位下载】

USB 连接好并已上电的情况下，外部按键复位也可进入 USB 下载模式，注意：

P4.7-nRST 出厂时默认是 P4.7-I/O 功能，要改为复位功能，需 ISP 烧录时取消设置复位脚用作 I/O 口，停电一次再上电才生效，程序区中用户程序也可改为复位脚或 I/O，这个立即生效。

1、按下 P4.7-nRST 外接的低电平复位按键复位 MCU，
松开复位键，MCU 从系统程序区启动，判断是否要下载用户程序，
等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后

2、点击电脑端下载软件中的【下载/编程】按钮，下载进行中，几秒钟后，提示下载成功！

备注：如果目标 MCU 的用户程序区已有 USB 通讯程序（含 CDC、HID 等），则通过复位脚复位进入 USB 下载模式不被支持，只能通过停电上电冷启动下载或者从用户程序区软复位到系统区下载

【USB 下载方法三，从用户程序区软复位到系统区下载】

USB 连接好并已上电的情况下，从用户程序区软复位到系统区也可进入 USB 下载模式

1、在用户程序区运行软复位到系统区的程序，就是 IAP_CONTR 寄存器送 60H
等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后

2、点击电脑端下载软件中的【下载/编程】按钮，下载进行中，几秒钟后，提示下载成功！

USB 下载 注意事项：

拔插 USB 插头不能代替上面线路图中的电源开关。正常操作步骤如下：

USB 的【Gnd，D+，D-】接好的情况下，按下 P3.2 按键接地，再通过正常的电源开关给 MCU 供电或重新供电，让 MCU 冷启动进入系统程序区，判断是否需要等待电脑端 USB 下载程序。

拔插 USB 插头代替电源开关不能每次都能成功进行 USB 下载的原因：

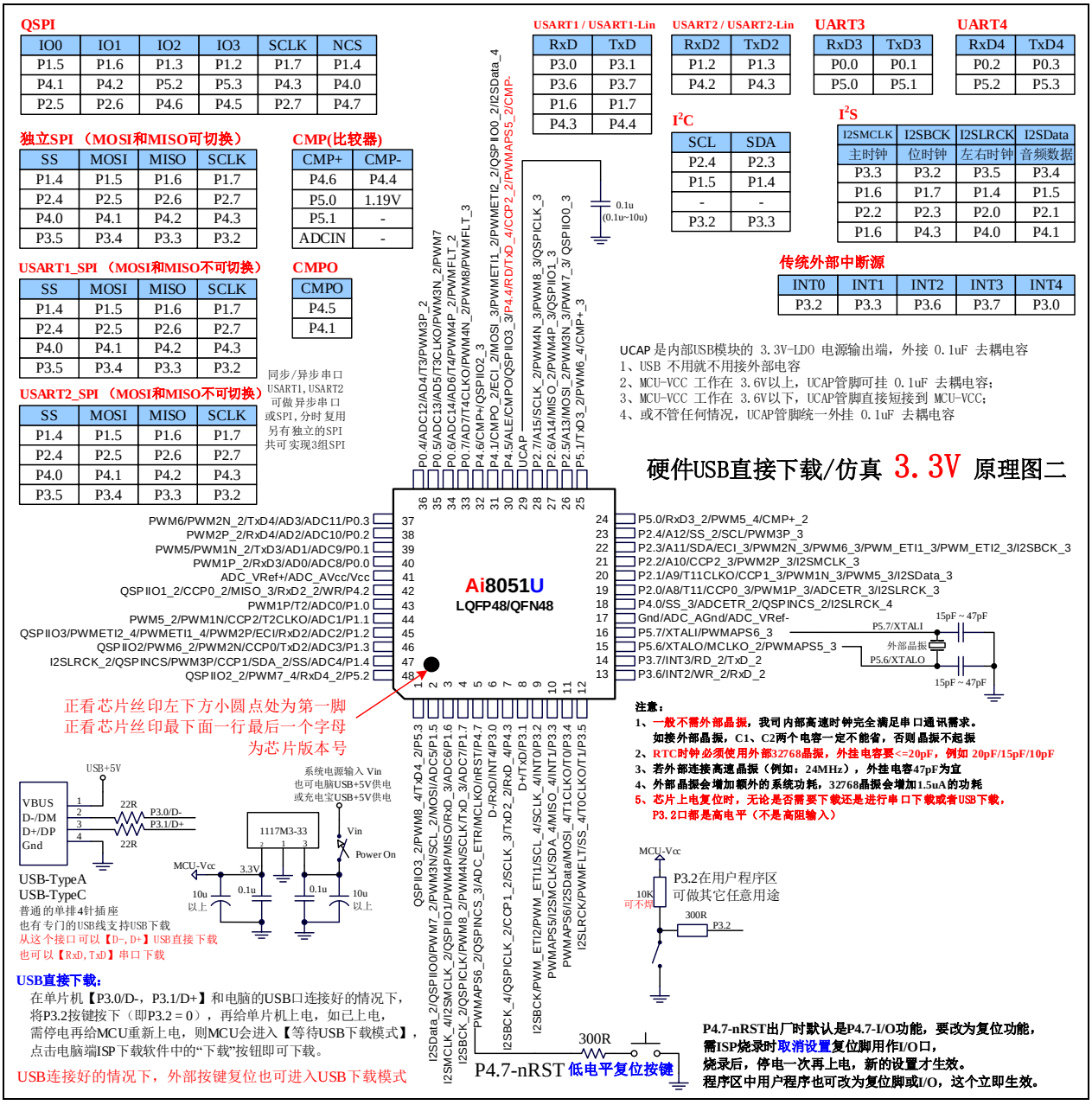
拔插 USB 插头，如【Gnd，USB+5V】已接触好，已供电，而【D+，D-】有一个甚至两个信号线还没有接触好，MCU 已上电，开始跑系统区程序时，发现 USB 还没接触好，则会从系统区软复位到用户程序区跑用户程序，不再进入等待 USB 下载模式，本次就无法顺利进行 USB 下载。

很多人经过多次插拔 USB，才能碰到 1 次【D+，D-】接触好的情况下，【Gnd，USB+5V】才开始接触好，才开始供电，才能成功进入 USB 下载。插 USB 插头代替供电，不能保证【Gnd，D+，D-，USB+5V】的接触顺序，所以，必须使用正常的电源开关，才能确保每次下载都能成功。

关于 I/O 的注意事项：

1、 P3.0 和 P3.1 口上电后的状态为弱上拉/准双向口模式
2、 除 P3.0 和 P3.1 外，其余所有 I/O 口上电后的状态均为高阻输入状态，用户在使用 I/O 口前必须先设置 I/O 口模式
3、 芯片上电时，若 P3.0 和 P3.1 同时为低电平，P3.2 口会短时间开启内部 4K 上拉，用以读取 P3.2 口外部状态来判断是否需要进入 USB 下载模式
4、 芯片上电时如果不需要使用 USB 进行 ISP 下载，P3.0/P3.1/P3.2 这 3 个 I/O 口不能同时为低电平，否则会进入 USB 下载模式而无法运行用户代码
5、 当使用 P4.7 当作复位脚时，这个端口内部的 4K 上拉电阻会一直打开；但 P4.7 做普通 I/O 口时，基于这个 I/O 口与复位脚共享管脚的特殊考量，端口内部的 4K 上拉电阻依然会打开大约 6.5 毫秒时间，再自动关闭（当用户的电路设计需要使用 P4.7 口驱动外部电路时，请务必考虑上电瞬间会有 6.5 毫秒时间的高电平的问题）

4.1.5 LQFP48/QFN48 管脚图，3.3V 线路二，USB-ISP 下载/仿真



备注：上图中红色管脚部分是 2024 年 10 月 15 日之后生产的芯片才有。

现在带硬件 USB 的 MCU 支持用硬件 USB 下载，因为用的是 USB-HID 通信协议，不需要安装任何驱动。只要 USB 鼠标、USB 键盘能工作，USB-HID 驱动就是好的，不要安装 USB-HID 驱动，免驱。

在 D-/P3.0，D+/P3.1 与 PC-USB 端口连接好的状况下，USB-ISP 下载程序有如下三种模式：

【USB 下载方法一，P3.2 按键，再结合停电上电下载】

1、按下板子上的 P3.2/INT0 按键，就是 P3.2 接地

2、给目标芯片重新上电，不管之前是否已通电。

===电子开关是按下停电后再松开就是上电

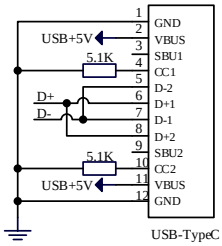
等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后，就与 P3.2 状态无关了，这时可以松开 P3.2 按键（P3.2 在用户程序区可做其它任意用途）

===传统的机械自锁紧开关是按上来停电，按下去是上电

3、点击电脑端下载软件中的【下载/编程】按钮（注意：USB 下载与串口下载的操作顺序不同）

下载进行中，几秒钟后，提示下载成功！

比较一下流行的 **USB-TypeC接口：**



【USB 下载方法二，复位管脚低电平复位下载】

USB 连接好并已上电的情况下，外部按键复位也可进入 USB 下载模式，注意：

P4.7-nRST 出厂时默认是 P4.7-I/O 功能，要改为复位功能，需 ISP 烧录时取消设置复位脚用作 I/O 口，停电一次再上电才生效，程序区中用户程序也可改为复位脚或 I/O，这个立即生效。

1、按下 P4.7-nRST 外接的低电平复位按键复位 MCU，
松开复位键，MCU 从系统程序区启动，判断是否要下载用户程序，
等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后

2、点击电脑端下载软件中的【下载/编程】按钮
下载进行中，几秒钟后，提示下载成功！

备注：如果目标 MCU 的用户程序区已有 USB 通讯程序（含 CDC、HID 等），则通过复位脚复位进入 USB 下载模式不被支持，只能通过停电上电冷启动下载或者从用户程序区软复位到系统区下载

【USB 下载方法三，从用户程序区软复位到系统区下载】

USB 连接好并已上电的情况下，从用户程序区软复位到系统区也可进入 USB 下载模式

1、在用户程序区运行软复位到系统区的程序，就是 IAP_CONTR 寄存器送 60H
等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后

2、点击电脑端下载软件中的【下载/编程】按钮，下载进行中，几秒钟后，提示下载成功！

USB 下载 注意事项：

拔插 USB 插头不能代替上面线路图中的电源开关。正常操作步骤如下：

USB 的【Gnd，D+，D-】接好的情况下，按下 P3.2 按键接地，再通过正常的电源开关给 MCU 供电或重新供电，让 MCU 冷启动进入系统程序区，判断是否需要等待电脑端 USB 下载程序。

拔插 USB 插头代替电源开关不能每次都能成功进行 USB 下载的原因：

拔插 USB 插头，如【Gnd，USB+5V】已接触好，已供电，而【D+，D-】有一个甚至两个信号线还没有接触好，MCU 已上电，开始跑系统区程序时，发现 USB 还没接触好，则会从系统区软复位到用户程序区跑用户程序，不再进入等待 USB 下载模式，本次就无法顺利进行 USB 下载。

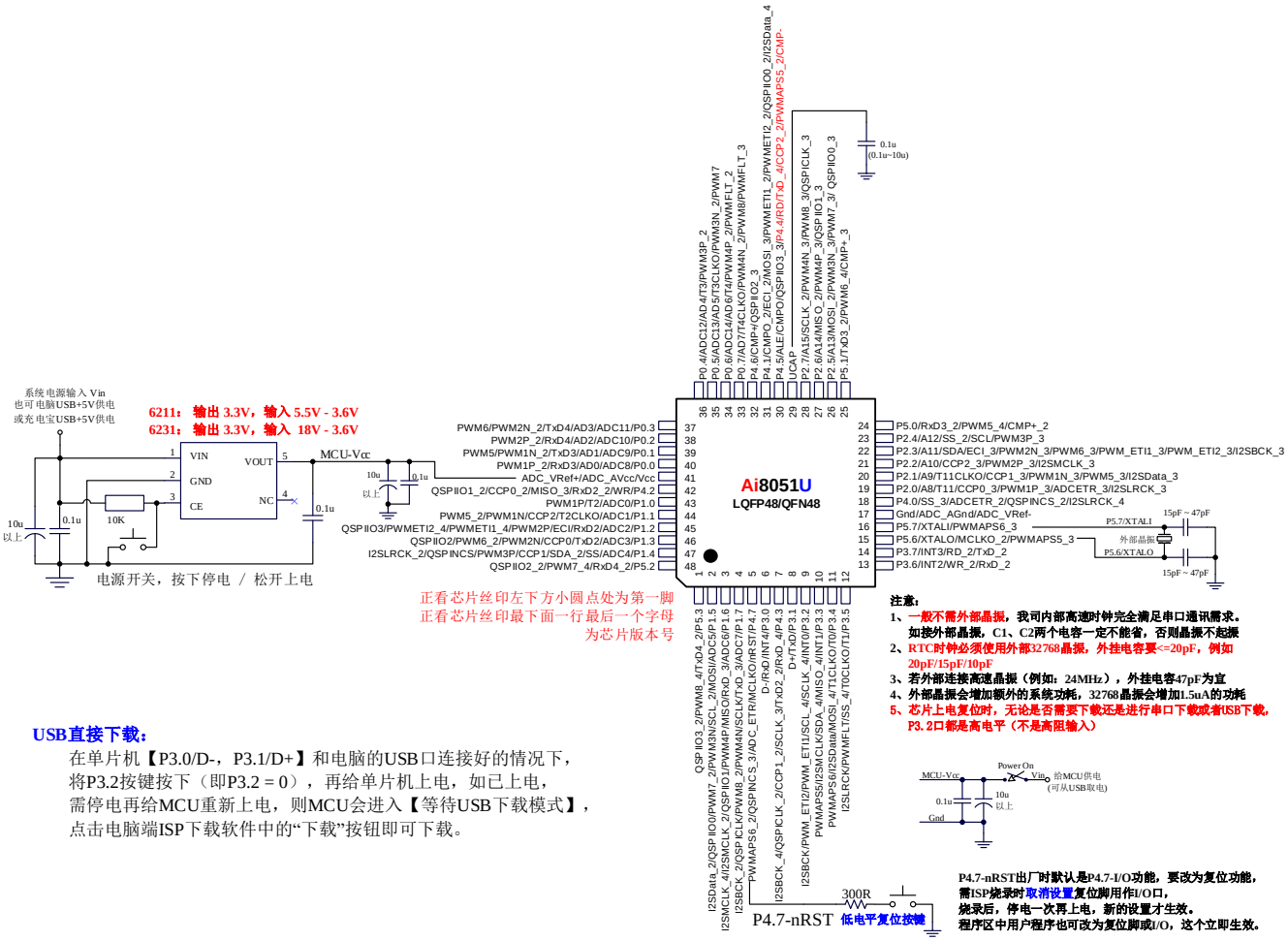
很多人经过多次插拔 USB，才能碰到 1 次【D+，D-】接触好的情况下，【Gnd，USB+5V】才开始接触好，才开始供电，才能成功进入 USB 下载。插 USB 插头代替供电，不能保证【Gnd，D+，D-，USB+5V】的接触顺序，所以，必须使用正常的电源开关，才能确保每次下载都能成功。

关于 I/O 的注意事项：

1、 P3.0 和 P3.1 口上电后的状态为弱上拉/准双向口模式
2、 除 P3.0 和 P3.1 外，其余所有 I/O 口上电后的状态均为高阻输入状态，用户在使用 I/O 口前必须先设置 I/O 口模式
3、 芯片上电时，若 P3.0 和 P3.1 同时为低电平，P3.2 口会短时间开启内部 4K 上拉，用以读取 P3.2 口外部状态来判断是否需要进入 USB 下载模式
4、 芯片上电时如果不需要使用 USB 进行 ISP 下载，P3.0/P3.1/P3.2 这 3 个 I/O 口不能同时为低电平，否则会进入 USB 下载模式而无法运行用户代码
5、 当使用 P4.7 当作复位脚时，这个端口内部的 4K 上拉电阻会一直打开；但 P4.7 做普通 I/O 口时，基于这个 I/O 口与复位脚共享管脚的特殊考量，端口内部的 4K 上拉电阻依然会打开大约 6.5 毫秒时间，再自动关闭（当用户的电路设计需要使用 P4.7 口驱动外部电路时，请务必考虑上电瞬间会有 6.5 毫秒时间的高电平的问题）

4.1.6　如果使用 3.3V-LDO 供电，建议用有电源关断功能的 6211/6231

3.3V-LDO 供电，使用有电源关断功能的 6211/6231供电 示意图



USB直接下载：

在单片机【P3.0/D-，P3.1/D+】和电脑的USB口连接好的情况下，
将P3.2按键按下（即P3.2 = 0），再给单片机上电，如已上电，
需停电再给MCU重新上电，则MCU会进入【等待USB下载模式】，
点击电脑端ISP下载软件中的“下载”按钮即可下载。

4.1.7 管脚说明

编号			名称	类型	说明
LQFP48/QFN48	LQFP44	PDIP40			
1	1	6	P5.3	I/O	标准 I/O 口
			TxD4_2	O	串口 4 的发送脚
			PWM8_4	I/O	PWM8 的捕获输入和脉冲输出
			QSPIIO3_2	I/O	QSPI 数据脚
2	1	6	P1.5	I/O	标准 I/O 口
			ADC5	I	ADC 模拟输入通道 5
			MOSI	I/O	SPI 主机输出从机输入
			SCL_2	I/O	I2C 的时钟线
			PWM3N	O	PWM3 的脉冲输出负极
			PWM7_2	I/O	PWM7 的捕获输入和脉冲输出
			QSPIIO0	I/O	QSPI 数据脚
			I2SSD_2	I/O	I2S 数据脚
3	2	7	P1.6	I/O	标准 I/O 口
			ADC6	I	ADC 模拟输入通道 6
			RxD_3	I	串口 1 的接收脚
			MISO	I/O	SPI 主机输入从机输出
			PWM4P	I/O	PWM4 的捕获输入和脉冲输出正极
			QSPIIO1	I/O	QSPI 数据脚
			I2SMCLK_2	I/O	I2S 主时钟脚
			I2SMCLK_4	I/O	I2S 主时钟脚
4	3	8	P1.7	I/O	标准 I/O 口
			ADC7	I	ADC 模拟输入通道 7
			TxD_3	O	串口 3 的发送脚
			SCLK	I/O	SPI 的时钟脚
			PWM4N	O	PWM4 的脉冲输出负极
			PWM8_2	I/O	PWM8 的捕获输入和脉冲输出
			QSPICLK	O	QSPI 时钟脚
			I2SCK_2	I/O	I2S 位时钟脚
5	4	9	P4.7	I/O	标准 I/O 口
			nRST	I	复位引脚（低电平复位）
			MCLKO	O	主时钟分频输出
			ADCETR	I	ADC 外部触发脚
			QSPINCS_3	O	QSPI 片选脚
			PWMAPS6_2	I/O	PWM 硬件移相端口
6	5	10	P3.0	I/O	标准 I/O 口
			INT4	I	外部中断 4 输入脚
			RxD	I	串口 1 的接收脚
			D-	I/O	USB 差分 D-脚

编号			名称	类型	说明
LQFP48/QFN48	LQFP44	PDIP40			
7	6	11	P4.3	I/O	标准 I/O 口
			RxD_4	I	串口 1 的接收脚
			TxD2_2	O	串口 2 的发送脚
			SCLK_3	I/O	SPI 的时钟脚
			CCP1_2	I/O	CCP1 的捕获输入和脉冲输出
			QSPICLK_2	O	QSPI 时钟脚
			I2SCK_4	I/O	I2S 位时钟脚
8	7	11	P3.1	I/O	标准 I/O 口
			TxD	O	串口 1 的发送脚
			D+	I/O	USB 差分 D+脚
9	8	12	P3.2	I/O	标准 I/O 口
			INT0	I	外部中断 0 输入脚
			SCLK_4	I/O	SPI 的时钟脚
			SCL_4	I/O	I2C 的时钟线
			PWMETI1	I	PWMA 外部触发输入脚
			PWMETI2	I	PWMB 外部触发输入脚
			I2SCK	I/O	I2S 位时钟脚
10	9	13	P3.3	I/O	标准 I/O 口
			INT1	I	外部中断 1 输入脚
			MISO_4	I/O	SPI 主机输入从机输出
			SDA_4	I/O	I2C 数据脚
			I2SMCLK	I/O	I2S 主时钟脚
			PWMAPS5	I/O	PWM 硬件移相端口
11	10	14	P3.4	I/O	标准 I/O 口
			T0	I	定时器 0 外部时钟输入
			T1CLKO	O	定时器 1 时钟分频输出
			MOSI_4	I/O	SPI 主机输出从机输入
			I2SSD	I/O	I2S 数据脚
			PWMAPS6	I/O	PWM 硬件移相端口
12	11	15	P3.5	I/O	标准 I/O 口
			T1	I	定时器 1 外部时钟输入
			T0CLKO	O	定时器 0 时钟分频输出
			SS_4	I	SPI 的从机选择脚（主机为输出）
			PWMFLT	I	PWMA/PWMB 的外部异常检测脚
			I2SWS	I/O	I2S 左右时钟脚
13	12	16	P3.6	I/O	标准 I/O 口
			INT2	I	外部中断 2 输入脚
			WR_2	O	外部总线写信号输出脚
			RxD_2	I	串口 1 的接收脚

编号			名称	类型	说明
LQFP48/QFN48	LQFP44	PDIP40			
14	13	17	P3.7	I/O	标准 I/O 口
			INT3	I	外部中断 3 输入脚
			RD_2	O	外部总线读信号输出脚
			TxD_2	O	串口 1 的发送脚
15	14	18	P5.6	I/O	标准 I/O 口
			XTALO	O	外部晶振型号输出脚
			MCLKO_2	O	主时钟分频输出
			PWMAPS5_3	I/O	PWM 硬件移相端口
16	15	19	P5.7	I/O	标准 I/O 口
			XTALI	I	外部晶振型号输入脚
			PWMAPS6_3	I/O	PWM 硬件移相端口
17	16	20	Gnd	Gnd	地线
			ADC_AGnd	Gnd	ADC 地线
			ADC_VRef-	Ref	ADC 外部参考电压源输入脚
18	17	21	P4.0	I/O	标准 I/O 口
			SS_3	I	SPI 的从机选择脚（主机为输出）
			ADCETR_2	I	ADC 外部触发脚
			QSPINCS_2	O	QSPI 片选脚
			I2SWS_4	I/O	I2S 左右时钟脚
19	18	21	P2.0	I/O	标准 I/O 口
			A8	O	地址总线
			T11	I	定时器 11 外部时钟输入
			CCP0_3	I/O	CCP0 的捕获输入和脉冲输出
			PWM1P_3	I/O	PWM1 的捕获输入和脉冲输出正极
			ADCETR_3	I	ADC 外部触发脚
			I2SWS_3	I/O	I2S 左右时钟脚
20	19	22	P2.1	I/O	标准 I/O 口
			A9	O	地址总线
			T11CLKO	O	定时器 11 时钟分频输出
			CCP1_3	I/O	CCP1 的捕获输入和脉冲输出
			PWM1N_3	O	PWM1 的脉冲输出负极
			PWM5_3	I/O	PWM5 的捕获输入和脉冲输出
			I2SSD_3	I/O	I2S 数据脚
21	20	23	P2.2	I/O	标准 I/O 口
			A10	O	地址总线
			CCP2_3	I/O	CCP2 的捕获输入和脉冲输出
			PWM2P_3	I/O	PWM2 的捕获输入和脉冲输出正极
			I2SMCLK_3	I/O	I2S 主时钟脚

编号			名称	类型	说明
LQFP48/QFN48	LQFP44	PDIP40			
22	21	24	P2.3	I/O	标准 I/O 口
			A11	O	地址总线
			SDA	I/O	I2C 数据脚
			ECI_3	I	CCP 模块的外部时钟脚
			PWM2N_3	O	PWM2 的脉冲输出负极
			PWM6_3	I/O	PWM6 的捕获输入和脉冲输出
			PWMETI_3	I	PWMA 外部触发输入脚
			PWMETI2_3	I	PWMB 外部触发输入脚
			I2SCK_3	I/O	I2S 位时钟脚
23	22	25	P2.4	I/O	标准 I/O 口
			A12	O	地址总线
			SS_2	I	SPI 的从机选择脚（主机为输出）
			SCL	I/O	I2C 的时钟线
			PWM3P_3	I/O	PWM3 的捕获输入和脉冲输出正极
24	22	25	P5.0	I/O	标准 I/O 口
			RxD3_2	I	串口 3 的接收脚
			PWM5_4	I/O	PWM5 的捕获输入和脉冲输出
			CMP+_2	I	比较器正极输入
25	23	26	P5.1	I/O	标准 I/O 口
			TxD3_2	O	串口 3 的发送脚
			PWM6_4	I/O	PWM6 的捕获输入和脉冲输出
			CMP+_3	I	比较器正极输入
26	23	26	P2.5	I/O	标准 I/O 口
			A13	O	地址总线
			MOSI_2	I/O	SPI 主机输出从机输入
			PWM3N_3	O	PWM3 的脉冲输出负极
			PWM7_3	I/O	PWM7 的捕获输入和脉冲输出
			QSPIIO0_3	I/O	QSPI 数据脚
27	24	27	P2.6	I/O	标准 I/O 口
			A14	O	地址总线
			MISO_2	I/O	SPI 主机输入从机输出
			PWM4P_3	I/O	PWM4 的捕获输入和脉冲输出正极
			QSPIIO1_3	I/O	QSPI 数据脚
28	25	28	P2.7	I/O	标准 I/O 口
			A15	O	地址总线
			SCLK_2	I/O	SPI 的时钟脚
			PWM4N_3	O	PWM4 的脉冲输出负极
			PWM8_3	I/O	PWM8 的捕获输入和脉冲输出
			QSPICLK_3	O	QSPI 时钟脚
29	26	29	UCAP	CAP	USB LDO 外接电容较

编号			名称	类型	说明
LQFP48/QFN48	LQFP44	PDIP40			
30	27	30	P4.5	I/O	标准 I/O 口
			ALE	O	地址锁存信号
			CMPO	O	比较器结果输出脚
			QSPIIO3_3	I/O	QSPI 数据脚
30	27	30	P4.4	I/O	标准 I/O 口
			RD	O	外部总线读信号输出脚
			TxD_4	O	串口 1 的发送脚
			CCP2_2	I/O	CCP2 的捕获输入和脉冲输出
			PWMAPS5	I/O	PWM 硬件移相端口
			CMP-	I	比较器负极输入
31	28	31	P4.1	I/O	标准 I/O 口
			CMPO_2	O	比较器结果输出脚
			ECI_2	I	CCP 模块的外部时钟脚
			MOSI_3	I/O	SPI 主机输出从机输入
			PWMETI_2	I	PWMA 外部触发输入脚
			PWMETI2_2	I	PWMB 外部触发输入脚
			QSPIIO0_2	I/O	QSPI 数据脚
			I2SSD_4	I/O	I2S 数据脚
32	29	31	P4.6	I/O	标准 I/O 口
			CMP+	I	比较器正极输入
			QSPIIO2_3	I/O	QSPI 数据脚
33	30	32	P0.7	I/O	标准 I/O 口
			AD7	I/O	地址/数据总线
			T4CLKO	O	定时器 4 时钟分频输出
			PWM4N_2	O	PWM4 的脉冲输出负极
			PWM8	I/O	PWM8 的捕获输入和脉冲输出
			PWMFLT_3	I	PWMA/PWMB 的外部异常检测脚
34	31	33	P0.6	I/O	标准 I/O 口
			ADC14	I	ADC 模拟输入通道 14
			AD6	I/O	地址/数据总线
			T4	I	定时器 4 外部时钟输入
			PWM4P_2	I/O	PWM4 的捕获输入和脉冲输出正极
			PWMFLT_2	I	PWMA/PWMB 的外部异常检测脚
35	32	34	P0.5	I/O	标准 I/O 口
			ADC13	I	ADC 模拟输入通道 13
			AD5	I/O	地址/数据总线
			T3CLKO	O	定时器 3 时钟分频输出
			PWM3N_2	O	PWM3 的脉冲输出负极
			PWM7	I/O	PWM7 的捕获输入和脉冲输出

编号			名称	类型	说明
LQFP48/QFN48	LQFP44	PDIP40			
36	33	35	P0.4	I/O	标准 I/O 口
			ADC12	I	ADC 模拟输入通道 12
			AD4	I/O	地址/数据总线
			T3	I	定时器 3 外部时钟输入
			PWM3P_2	I/O	PWM3 的捕获输入和脉冲输出正极
37	34	36	P0.3	I/O	标准 I/O 口
			ADC11	I	ADC 模拟输入通道 11
			AD3	I/O	地址/数据总线
			TxD4	O	串口 4 的发送脚
			PWM2N_2	O	PWM2 的脉冲输出负极
			PWM6	I/O	PWM6 的捕获输入和脉冲输出
38	35	37	P0.2	I/O	标准 I/O 口
			ADC10	I	ADC 模拟输入通道 10
			AD2	I/O	地址/数据总线
			RxD4	I	串口 4 的接收脚
			PWM2P_2	I/O	PWM2 的捕获输入和脉冲输出正极
39	36	38	P0.1	I/O	标准 I/O 口
			ADC9	I	ADC 模拟输入通道 9
			AD1	I/O	地址/数据总线
			TxD3	O	串口 3 的发送脚
			PWM1N_2	O	PWM1 的脉冲输出负极
			PWM5	I/O	PWM5 的捕获输入和脉冲输出
40	37	39	P0.0	I/O	标准 I/O 口
			ADC8	I	ADC 模拟输入通道 8
			AD0	I/O	地址/数据总线
			RxD3	I	串口 3 的接收脚
			PWM1P_2	I/O	PWM1 的捕获输入和脉冲输出正极
41	38	40	Vcc	Vcc	电源脚
			ADC_AVcc	Vcc	ADC 电源脚
			ADC_VRef+	Ref	ADC 外部参考电压源输入脚
42	39	1	P4.2	I/O	标准 I/O 口
			WR	O	外部总线写信号输出脚
			RxD2_2	I	串口 2 的接收脚
			MISO_3	I/O	SPI 主机输入从机输出
			CCP0_2	I/O	CCP0 的捕获输入和脉冲输出
			QSPIIO1_2	I/O	QSPI 数据脚
43	40	1	P1.0	I/O	标准 I/O 口
			ADC0	I	ADC 模拟输入通道 0
			T2	I	定时器 2 外部时钟输入
			PWM1P	I/O	PWM1 的捕获输入和脉冲输出正极

编号			名称	类型	说明
LQFP48/QFN48	LQFP44	PDIP40			
44	41	2	P1.1	I/O	标准 I/O 口
			ADC1	I	ADC 模拟输入通道 1
			T2CLKO	O	定时器 2 时钟分频输出
			CCP2	I/O	CCP2 的捕获输入和脉冲输出
			PWM1N	O	PWM1 的脉冲输出负极
			PWM5_2	I/O	PWM5 的捕获输入和脉冲输出
45	42	3	P1.2	I/O	标准 I/O 口
			ADC2	I	ADC 模拟输入通道 2
			RxD2	I	串口 2 的接收脚
			ECI	I	CCP 模块的外部时钟脚
			PWM2P	I/O	PWM2 的捕获输入和脉冲输出正极
			PWMETI1_4	I	PWMA 外部触发输入脚
			PWMWTI2_4	I	PWMB 外部触发输入脚
			QSPIIO3	I/O	QSPI 数据脚
46	43	4	P1.3	I/O	标准 I/O 口
			ADC3	I	ADC 模拟输入通道 3
			TxD2	O	串口 2 的发送脚
			CCP0	I/O	CCP0 的捕获输入和脉冲输出
			PWM2N	O	PWM2 的脉冲输出负极
			PWM6_2	I/O	PWM6 的捕获输入和脉冲输出
			QSPIIO2	I/O	QSPI 数据脚
47	44	5	P1.4	I/O	标准 I/O 口
			ADC4	I	ADC 模拟输入通道 4
			SS	I	SPI 的从机选择脚（主机为输出）
			SDA_2	I/O	I2C 数据脚
			CCP1	I/O	CCP1 的捕获输入和脉冲输出
			PWM3P	I/O	PWM3 的捕获输入和脉冲输出正极
			QSPINCS	O	QSPI 片选脚
			I2SWS_2	I/O	I2S 左右时钟脚
48	44	5	P5.2	I/O	标准 I/O 口
			RxD_2	I	串口 1 的接收脚
			PWM7_4	I/O	PWM7 的捕获输入和脉冲输出
			QSPIIO2_2	I/O	QSPI 数据脚

4.1.8　USB-Link1D 对 Ai8051U 自动停电/上电烧录，串口仿真+串口通讯

USB Link1D工具：支持全自动停电-上电在线下载 / 脱机下载 / 仿真

【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，下载编程进行中，数秒后提示下载编程成功，目标MCU会自动复位到用户程序区自动跑用户程序。部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P4.7/nRST变成复位脚

QSPI

IO0	IO1	IO2	IO3	SCLK	NCS
P1.5	P1.6	P1.3	P1.2	P1.7	P1.4
P4.1	P4.2	P5.2	P5.3	P4.3	P4.0
P2.5	P2.6	P4.6	P4.5	P2.7	P4.7

独立SPI （MOSI和MISO可切换）

SS	MOSI	MISO	SCLK
P1.4	P1.5	P1.6	P1.7
P2.4	P2.5	P2.6	P2.7
P4.0	P4.1	P4.2	P4.3
P3.5	P3.4	P3.3	P3.2

USART1_SPI （MOSI和MISO不可切换）

SS	MOSI	MISO	SCLK
P1.4	P1.5	P1.6	P1.7
P2.4	P2.5	P2.6	P2.7
P4.0	P4.1	P4.2	P4.3
P3.5	P3.4	P3.3	P3.2

USART2_SPI （MOSI和MISO不可切换）

SS	MOSI	MISO	SCLK
P1.4	P1.5	P1.6	P1.7
P2.4	P2.5	P2.6	P2.7
P4.0	P4.1	P4.2	P4.3
P3.5	P3.4	P3.3	P3.2

同步/异步串口USART1,USART2可做异步串口或SPI,分时复用另有独立的SPI共可实现3组SPI

USART1 / USART1-Lin

RxD	TxD
P3.0	P3.1
P3.6	P3.7
P1.6	P1.7
P4.3	P4.4

USART2 / USART2-Lin

RxD2	TxD2
P1.2	P1.3
P4.2	P4.3

UART3

RxD3	TxD3
P0.0	P0.1
P5.0	P5.1

CMP(比较器)

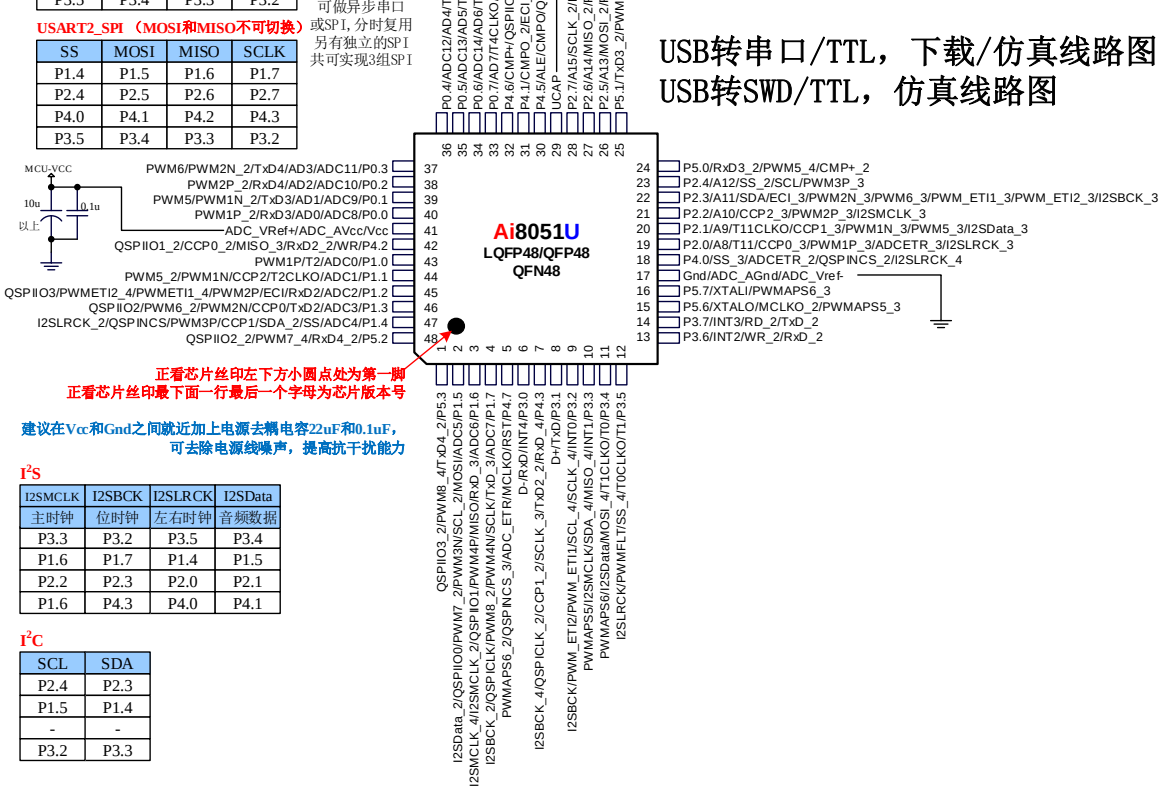
CMP+	CMP-
P4.6	P4.4
P5.0	1.19V
P5.1	-
ADCIN	-

CMPO

CMPO
P4.5
P4.1

UART4

RxD4	TxD4
P0.2	P0.3
P5.2	P5.3



I²S

I2SMCLK	I2SBCK	I2SLRCK	I2SData
主时钟	位时钟	左右时钟	音频数据
P3.3	P3.2	P3.5	P3.4
P1.6	P1.7	P1.4	P1.5
P2.2	P2.3	P2.0	P2.1
P1.6	P4.3	P4.0	P4.1

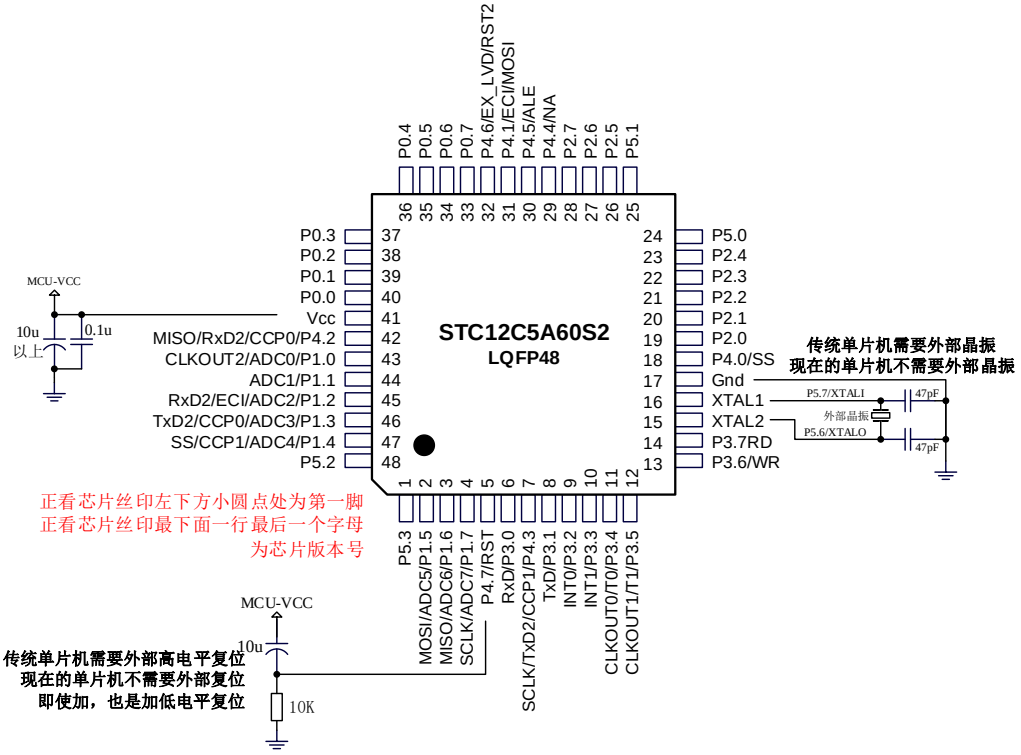
I²C

SCL	SDA
P2.4	P2.3
P1.5	P1.4
-	-
P3.2	P3.3

备注：上图中红色管脚部分是 2024 年 10 月 15 日之后生产的芯片才有。

比较下传统的 12C5A60S2 相应下载线路图：

传统单片机需要外部晶振，现在的单片机不需要外部晶振
传统单片机需要外部高电平复位，现在的不需要，并且已改为低电平复位



USB Link1D工具：支持全自动停电-上电在线下载 / 脱机下载 / 仿真

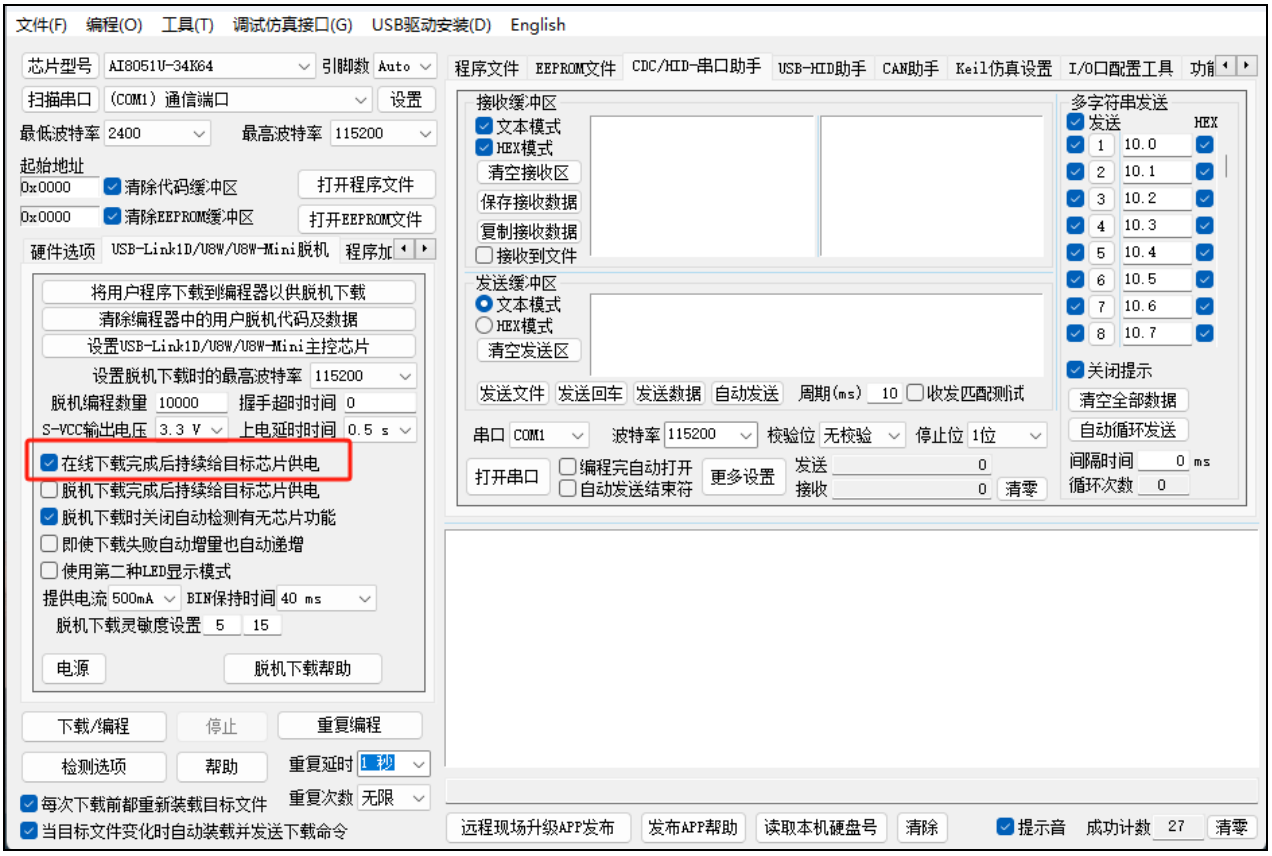
【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

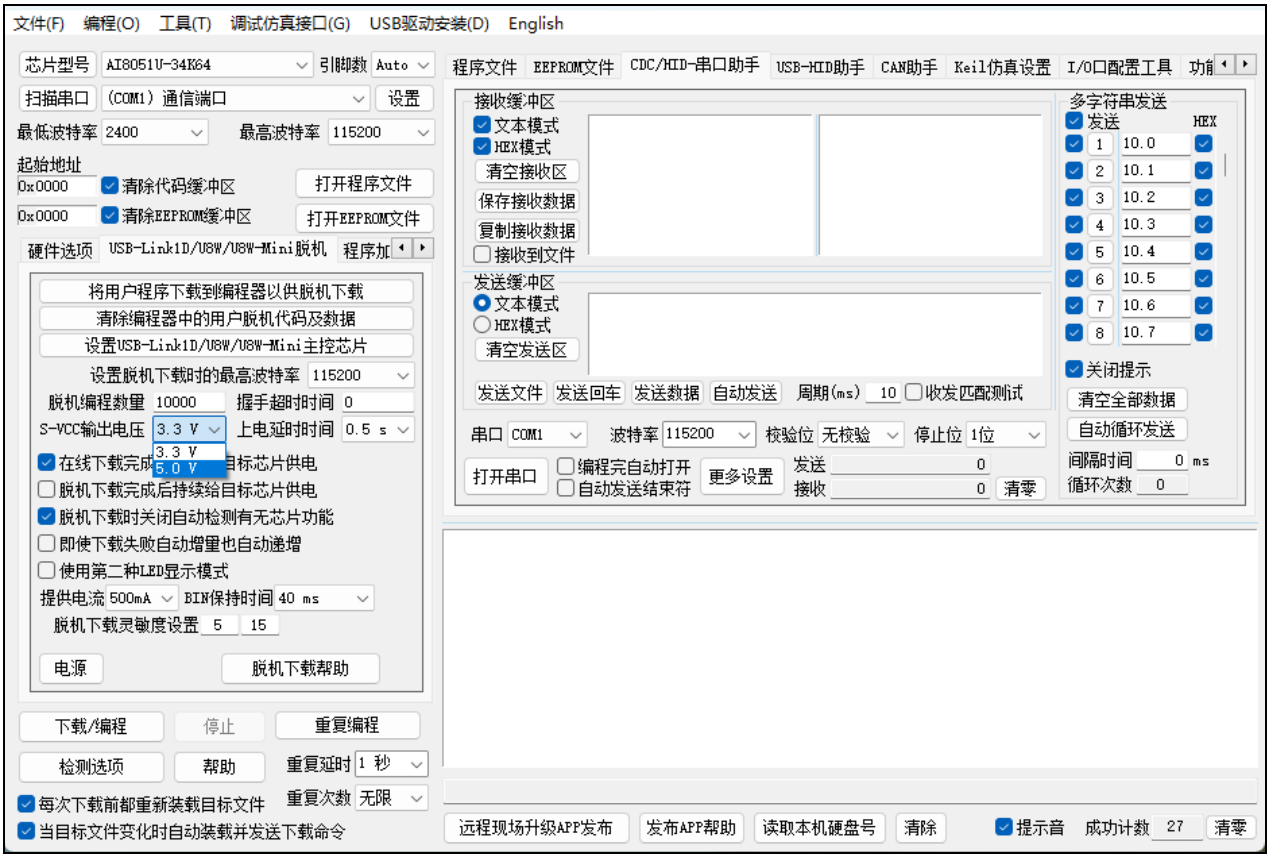
【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮
2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，
下载编程进行中，数秒后提示下载编程成功，目标MCU会自动复位到用户程序区自动跑用户程序。
部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P4.7/nRST变成复位脚

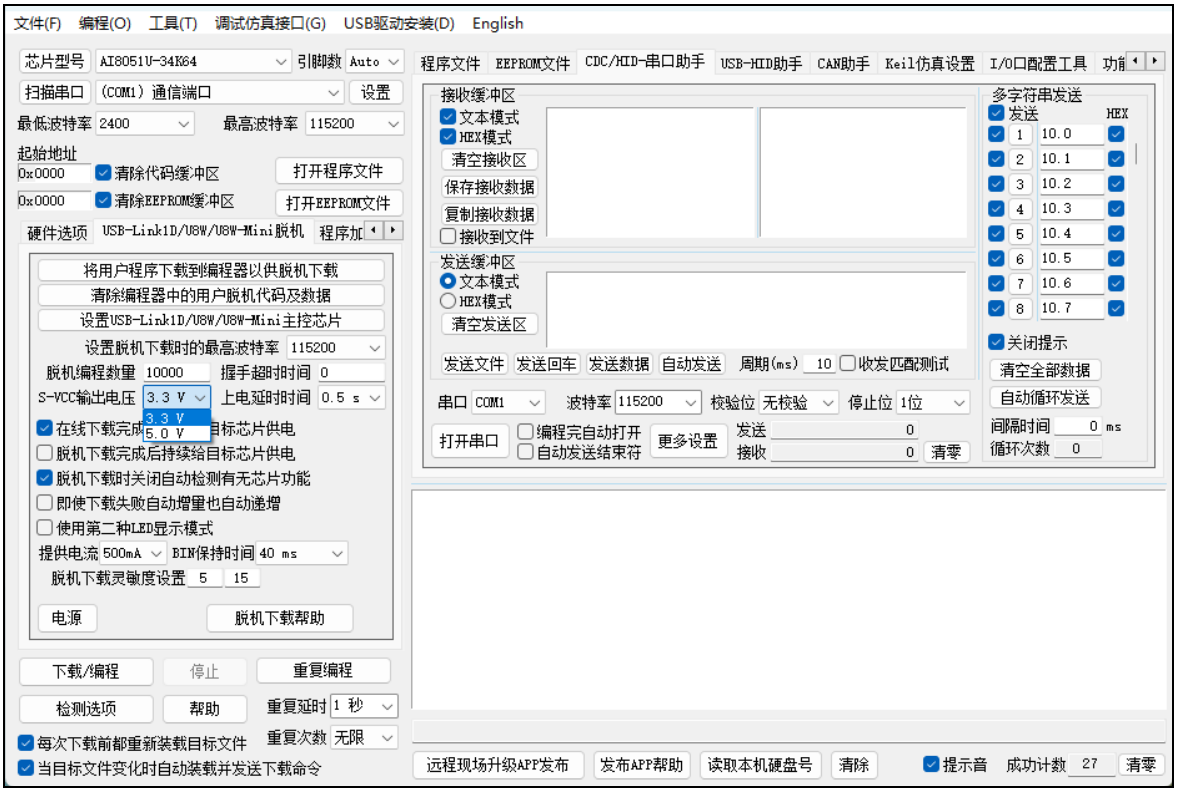
如何设置 USB-Link1D 下载完后持续给目标芯片供电



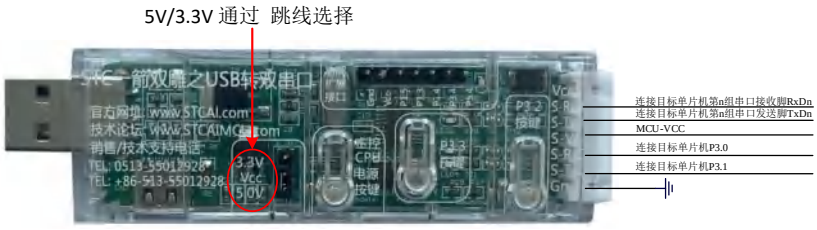
如何设置 USB-Link1D 输出 5V



如何设置 USB-Link1D 输出 3.3V



4.1.9 【一箭双雕之 USB 转双串口】工具进行烧录，串口仿真+串口通讯



一箭双雕之USB转双串口工具可支持其中一个串口仿真，另外一个串口通讯

【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，下载编程进行中，数秒后提示下载编程成功，目标MCU会自动复位到用户程序区自动跑用户程序。部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P4.7/nRST变成复位脚

QSPI

IO0	IO1	IO2	IO3	SCLK	NCS
P1.5	P1.6	P1.3	P1.2	P1.7	P1.4
P4.1	P4.2	P5.2	P5.3	P4.3	P4.0
P2.5	P2.6	P4.6	P4.5	P2.7	P4.7

独立SPI （MOSI和MISO可切换）

SS	MOSI	MISO	SCLK
P1.4	P1.5	P1.6	P1.7
P2.4	P2.5	P2.6	P2.7
P4.0	P4.1	P4.2	P4.3
P3.5	P3.4	P3.3	P3.2

USART1_SPI （MOSI和MISO不可切换）

SS	MOSI	MISO	SCLK
P1.4	P1.5	P1.6	P1.7
P2.4	P2.5	P2.6	P2.7
P4.0	P4.1	P4.2	P4.3
P3.5	P3.4	P3.3	P3.2

USART2_SPI （MOSI和MISO不可切换）

SS	MOSI	MISO	SCLK
P1.4	P1.5	P1.6	P1.7
P2.4	P2.5	P2.6	P2.7
P4.0	P4.1	P4.2	P4.3
P3.5	P3.4	P3.3	P3.2

同步/异步串口USART1,USART2可做异步串口或SPI,分时复用另有独立的SPI共可实现3组SPI

USART1 / USART1-Lin

RxD	TxD
P3.0	P3.1
P3.6	P3.7
P1.6	P1.7
P4.3	P4.4

USART2 / USART2-Lin

RxD2	TxD2
P1.2	P1.3
P4.2	P4.3

UART3

RxD3	TxD3
P0.0	P0.1
P5.0	P5.1

CMP(比较器)

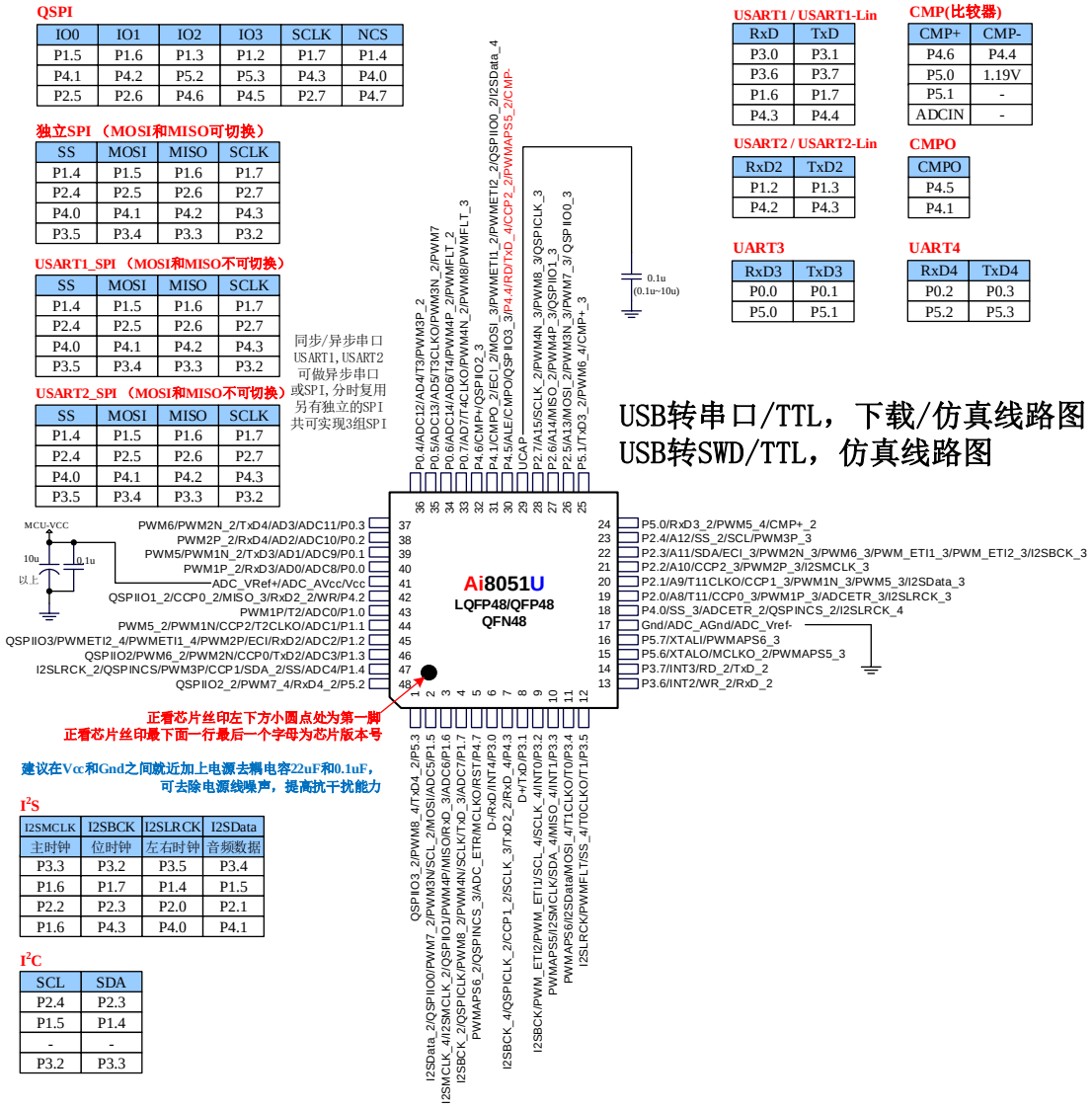
CMP+	CMP-
P4.6	P4.4
P5.0	1.19V
P5.1	-
ADCIN	-

CMPO

CMPO
P4.5
P4.1

UART4

RxD4	TxD4
P0.2	P0.3
P5.2	P5.3



I²S

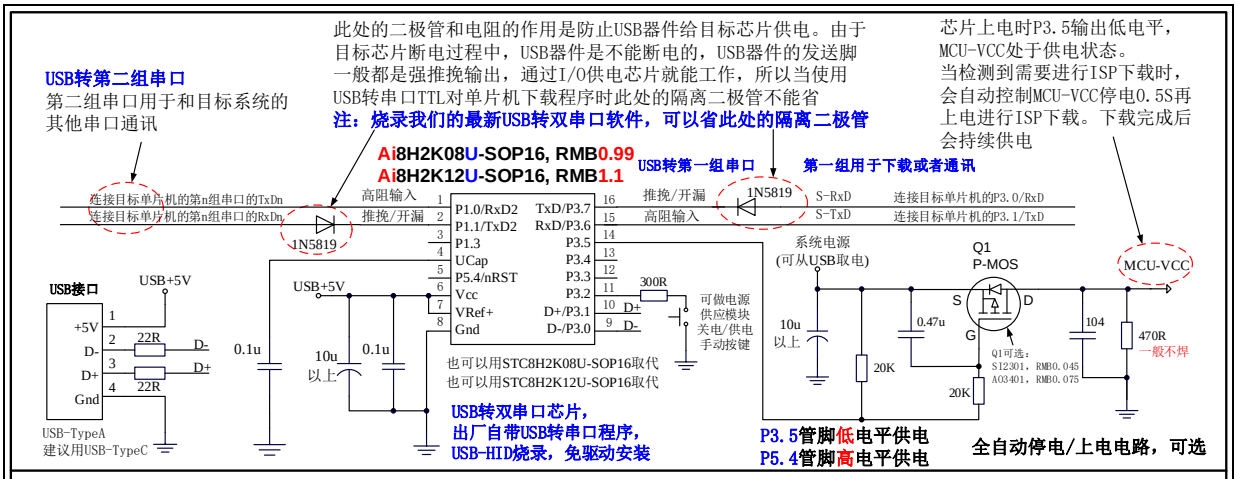
I2SMCLK	I2SBCK	I2SLRCK	I2SData
主时钟	位时钟	左右时钟	音频数据
P3.3	P3.2	P3.5	P3.4
P1.6	P1.7	P1.4	P1.5
P2.2	P2.3	P2.0	P2.1
P1.6	P4.3	P4.0	P4.1

I²C

SCL	SDA
P2.4	P2.3
P1.5	P1.4
-	-
P3.2	P3.3

备注：上图中红色管脚部分是 2024 年 10 月 15 日之后生产的芯片才有。

4.1.10 USB 转双串口芯片全自动停电/上电烧录，串口仿真+串口通讯，5V



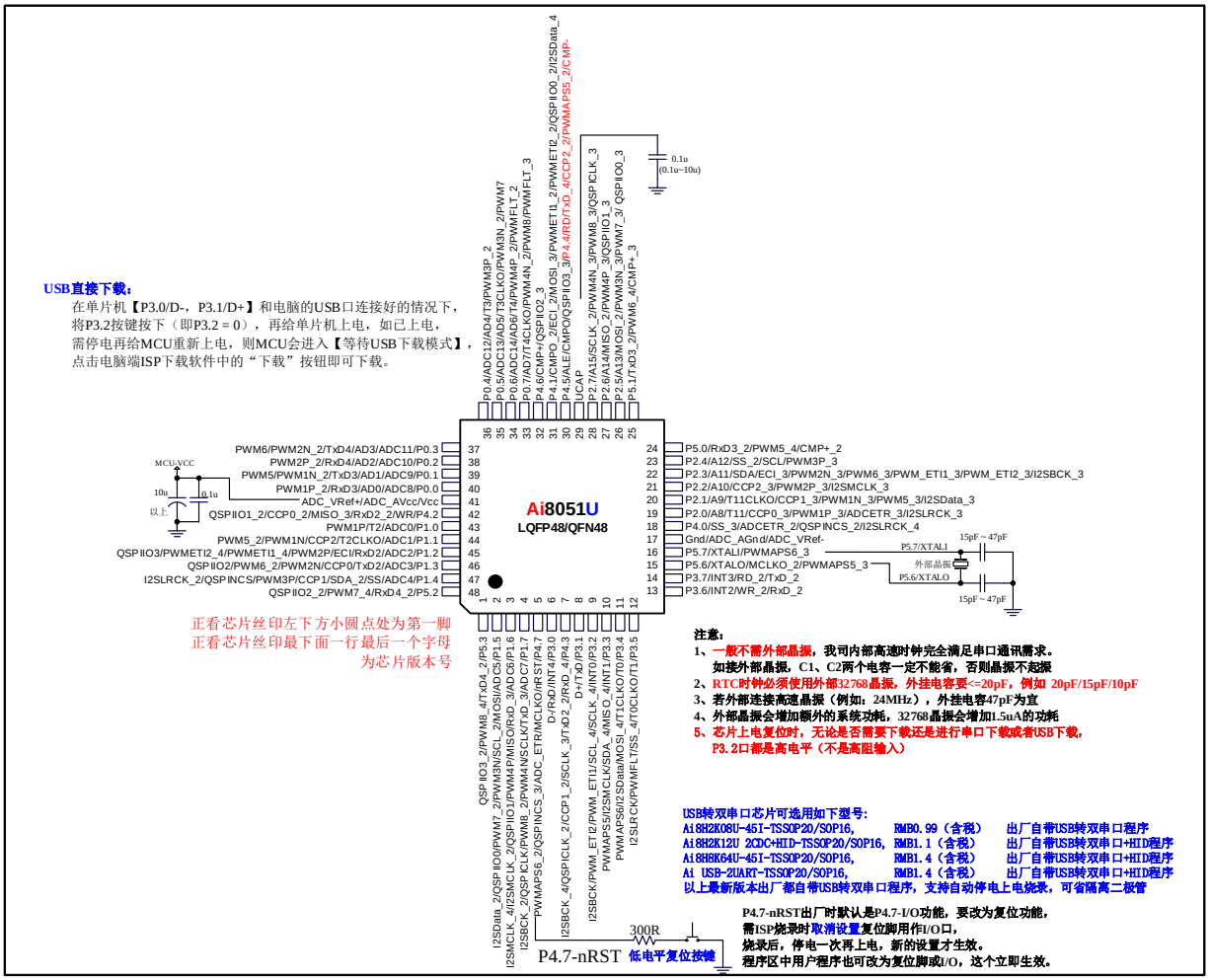
【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

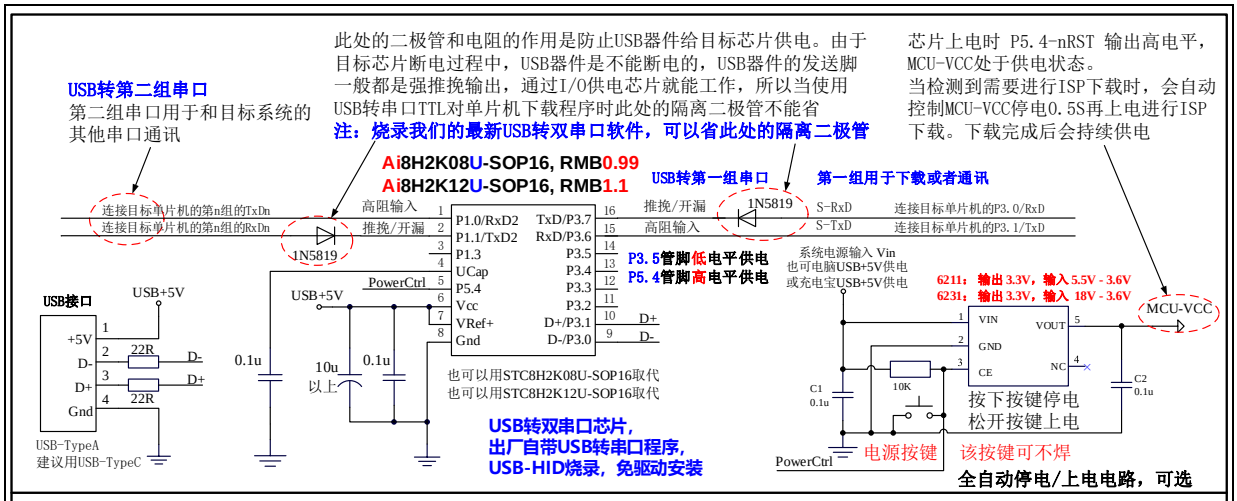
【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，下载编程进行中，数秒后提示下载编程成功，目标MCU会自动复位到用户程序区自动跑用户程序。
部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P4.7/nRST变成复位脚



4.1.11 USB 转双串口芯片全自动烧录，串口仿真+串口通讯，3.3V 原理图



【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

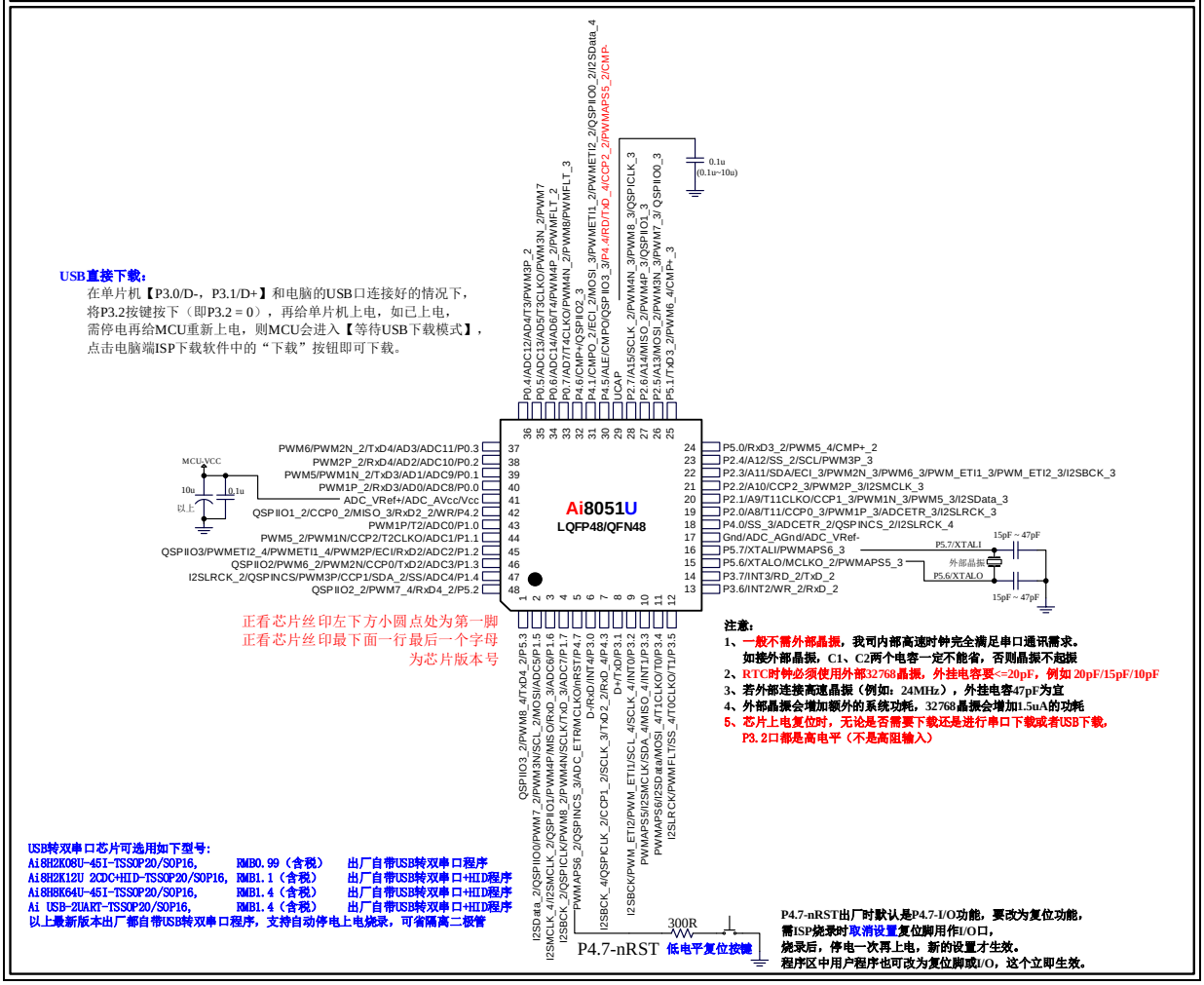
点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】

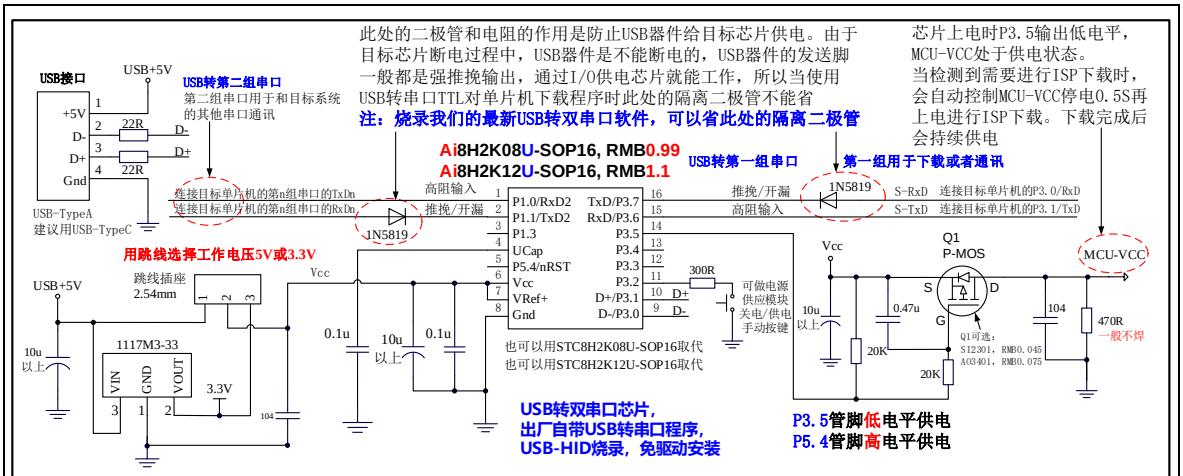
1、点击 电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，下载编程进行中，数秒后提示下载编程成功，目标MCU会自动复位到用户程序区自动跑用户程序。

部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P4.7/nRST变成复位脚



4.1.12　USB 转双串口芯片进行自动烧录/仿真+串口通讯，5V/3.3V 跳线选择



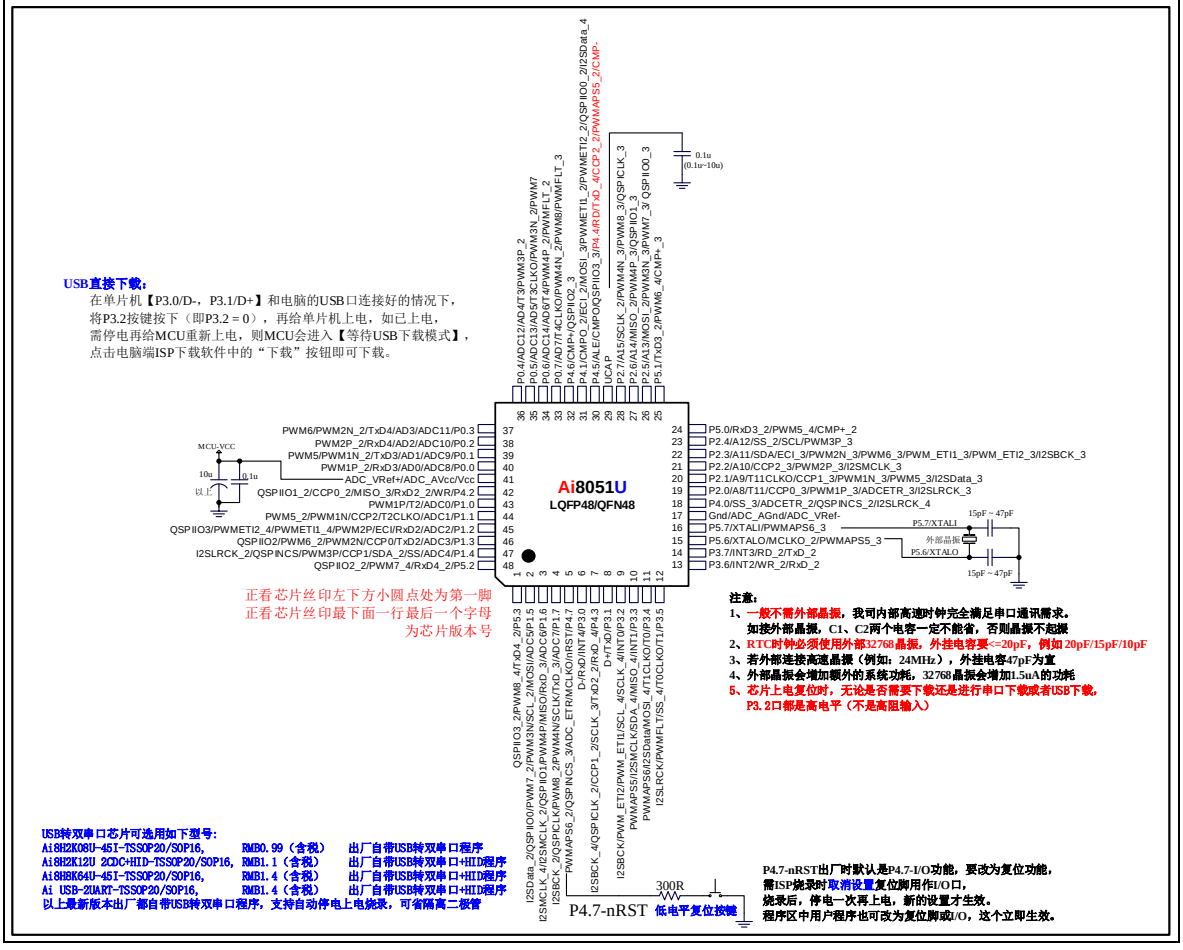
【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，
数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

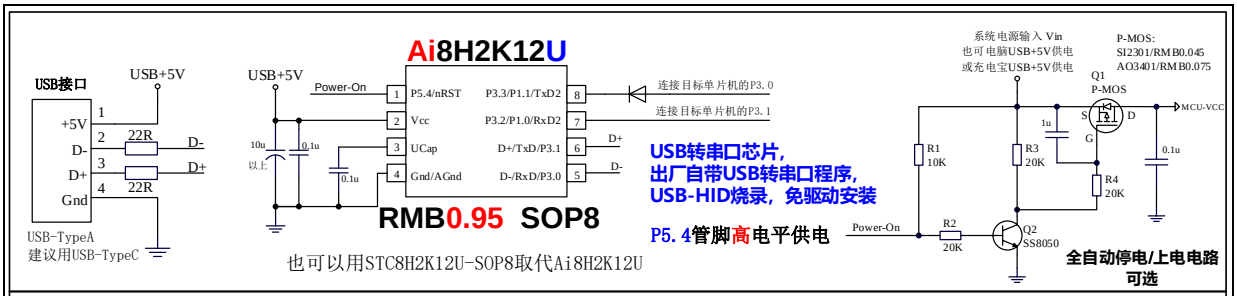
【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，
下载编程进行中，数秒后提示下载编程成功，目标MCU会自动复位到用户程序区自动跑用户程序。
部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P4.7/nRST变成复位脚



4.1.13 通用 USB 转串口芯片全自动停电/上电烧录，串口仿真，5V 原理图



【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

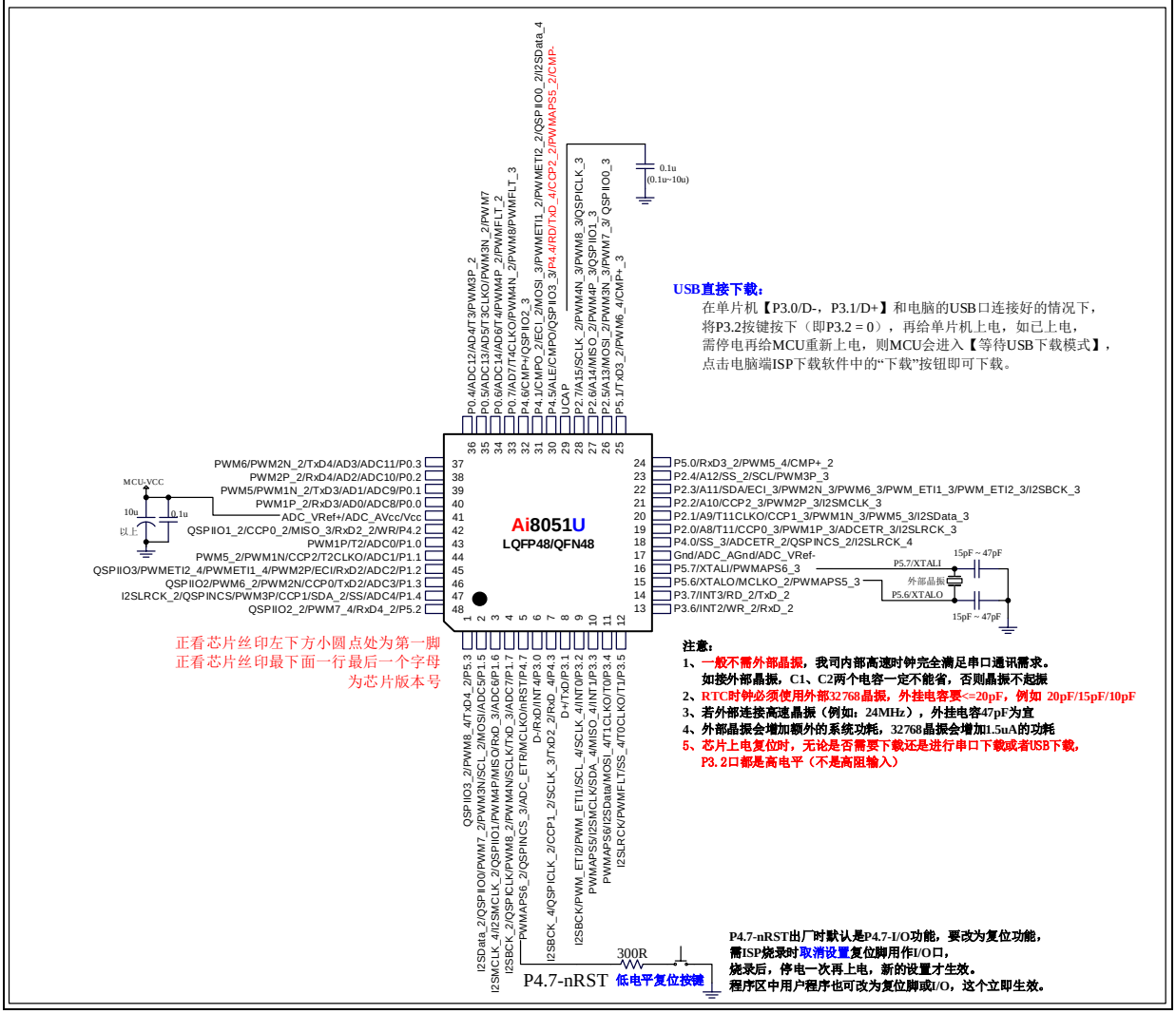
点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，下载编程进行中，数秒后提示下载编程成功，目标MCU会自动复位到用户程序区自动跑用户程序。

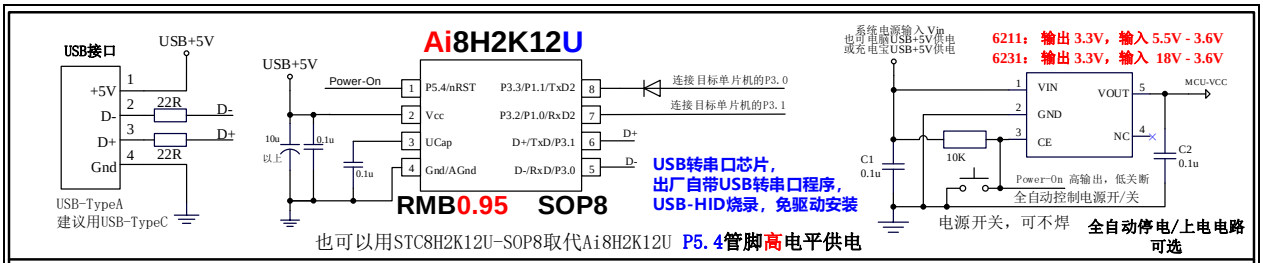
部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P4.7/nRST变成复位脚



备

注：上图中红色管脚部分是 2024 年 10 月 15 日之后生产的芯片才有。

4.1.14 通用 USB 转串口芯片全自动停电/上电烧录，串口仿真，3.3V 原理图



【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

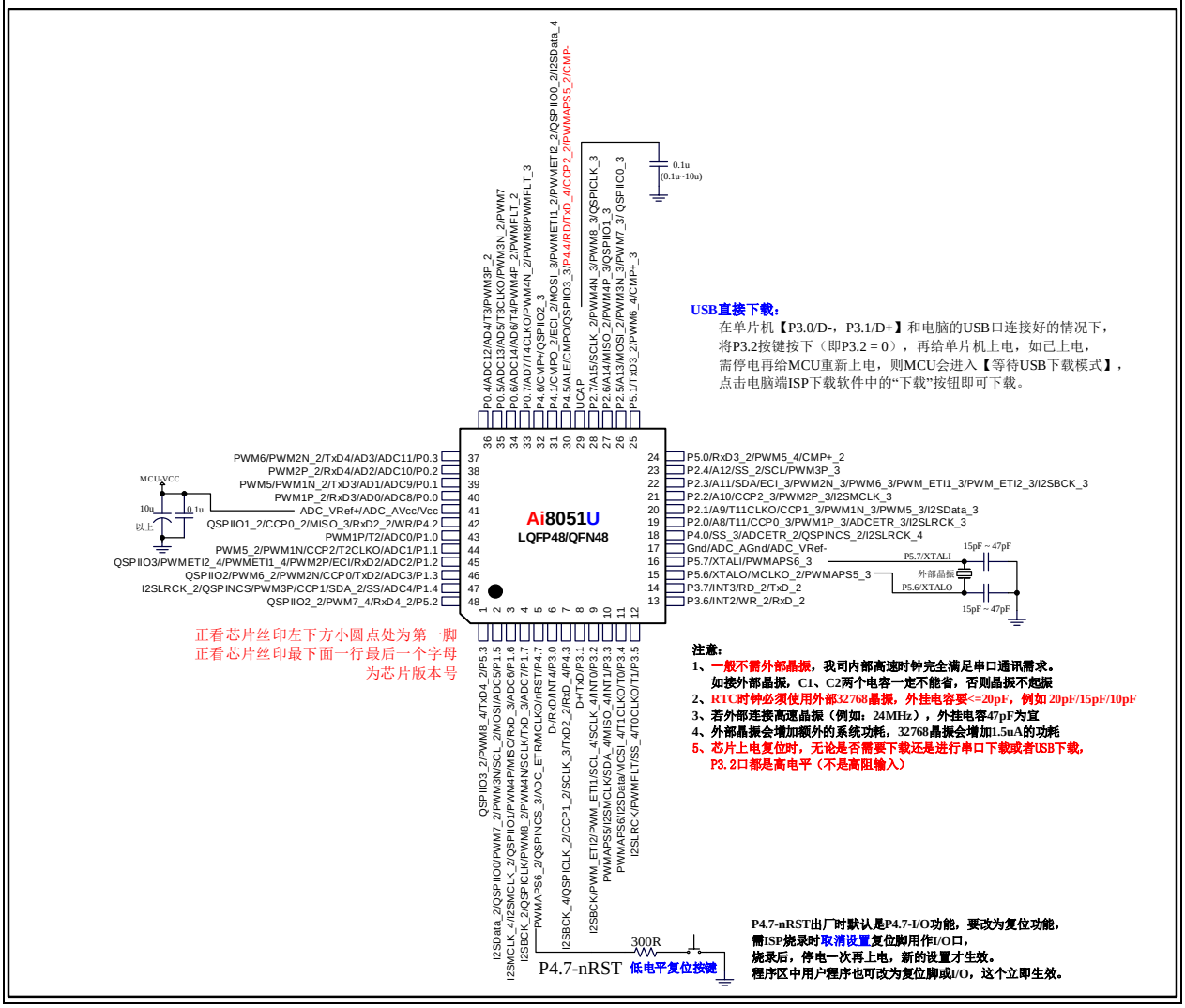
点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮

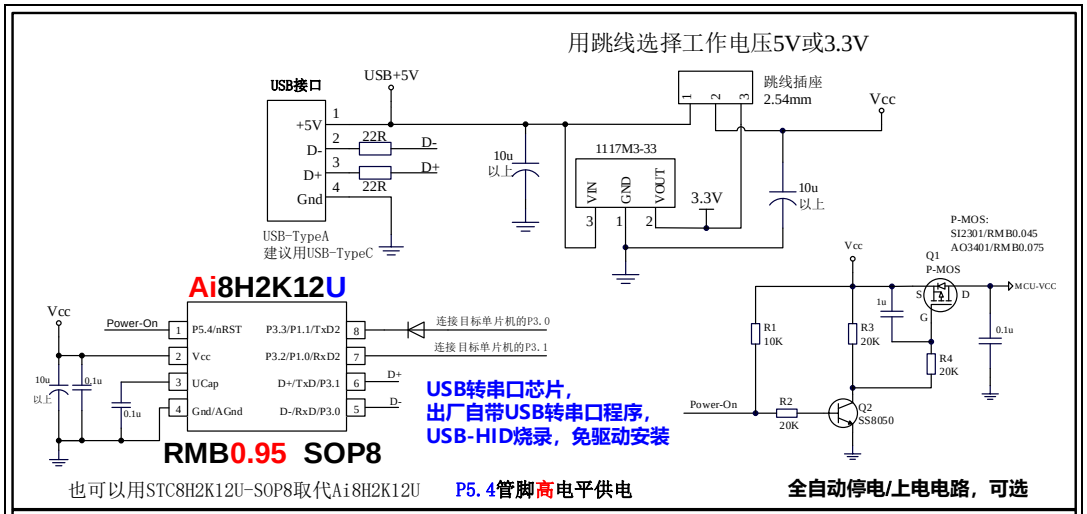
2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，下载编程进行中，数秒后提示下载编程成功，目标MCU会自动复位到用户程序区自动跑用户程序。

部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P4.7/nRST变成复位脚



备注：上图中红色管脚部分是 2024 年 10 月 15 日之后生产的芯片才有。

4.1.15 USB 转串口芯片全自动停电/上电烧录/仿真/通信，5V/3.3V 跳线选择

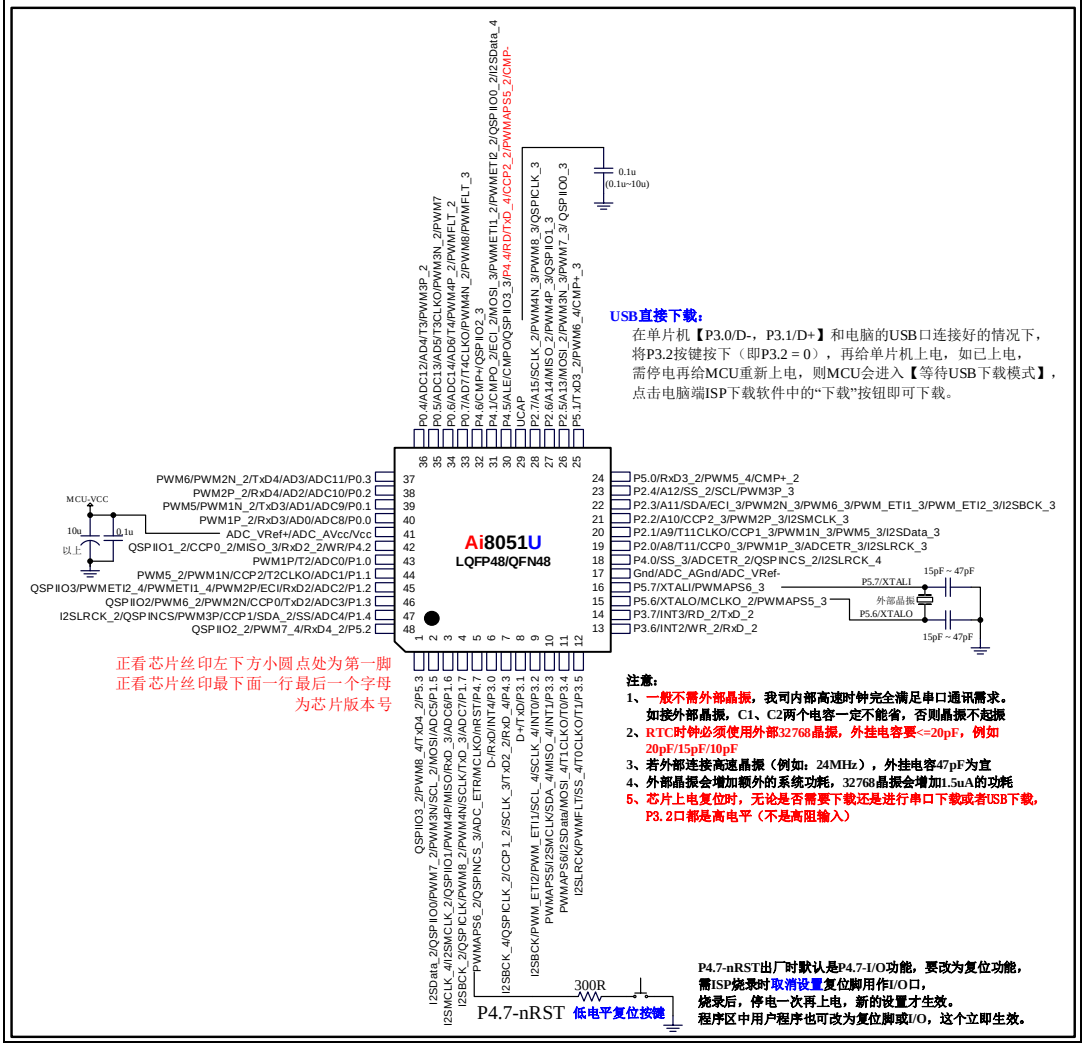


【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，
数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

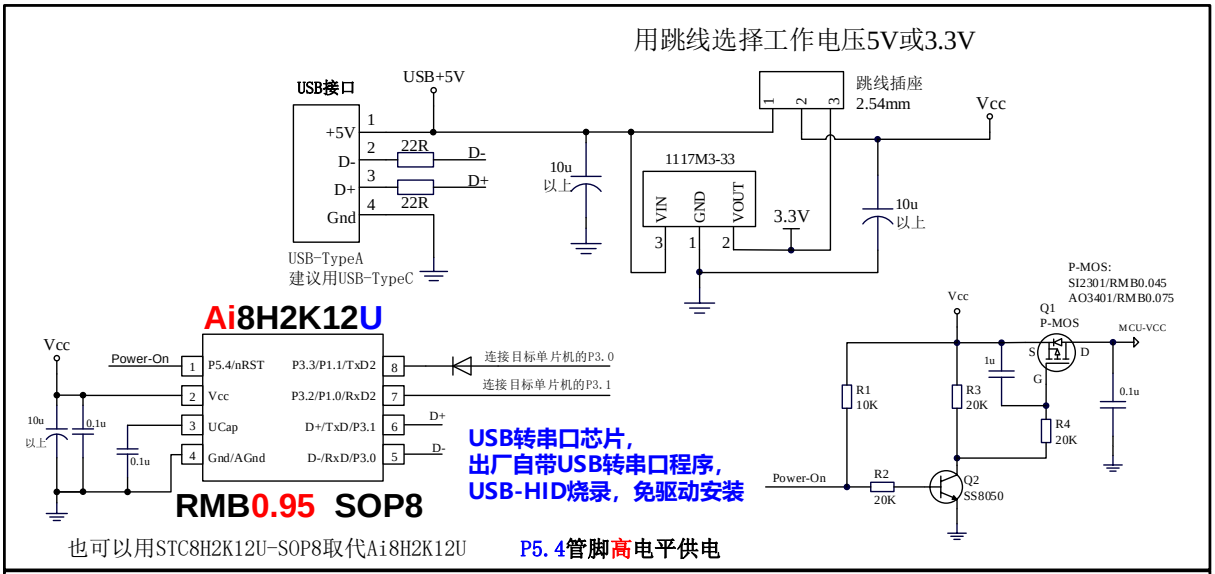
【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮
2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，
下载编程进行中，数秒后提示下载编程成功，目标MCU会自动复位到用户程序区自动跑用户程序。
部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P4.7/nRST变成复位脚



备注：上图中红色管脚部分是 2024 年 10 月 15 日之后生产的芯片才有。

比较下传统的 12C5A60S2 相应下载线路图：



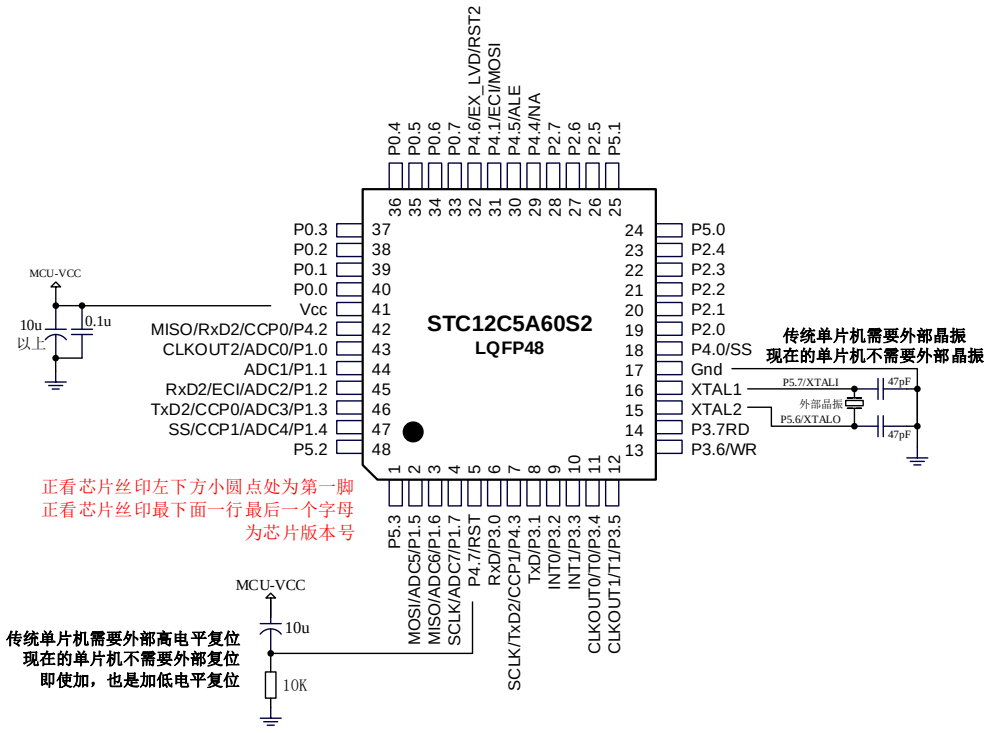
【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，
数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

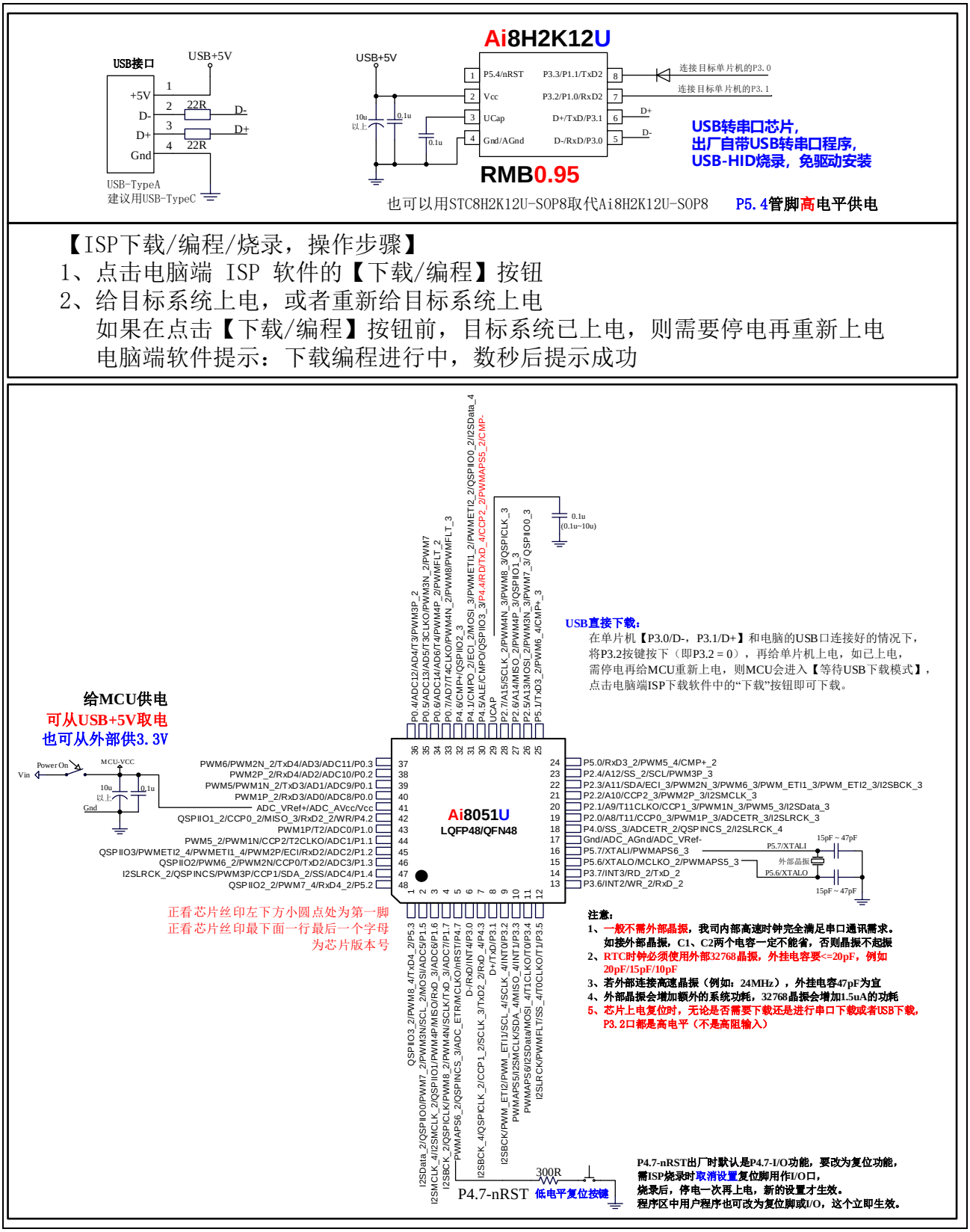
【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮
2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，
下载编程进行中，数秒后提示下载编程成功，目标MCU会自动复位到用户程序区自动跑用户程序。
部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P4.7/nRST变成复位脚

传统单片机需要外部晶振，现在的单片机不需要外部晶振
传统单片机需要外部高电平复位，现在的不需要，并且已改为低电平复位

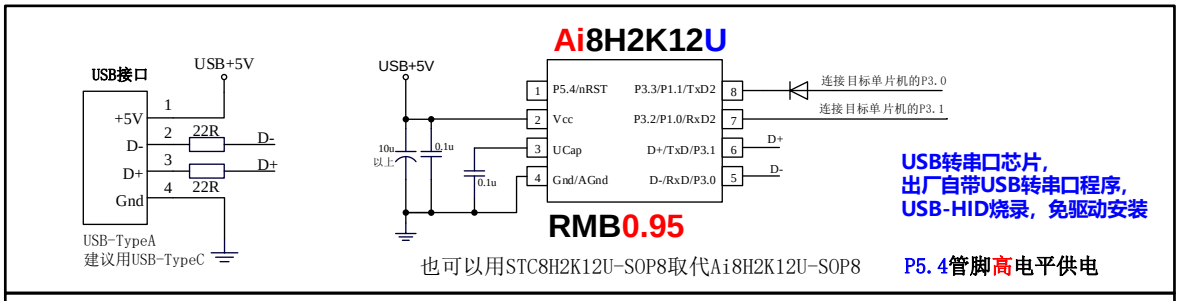


4.1.16 USB转串口芯片进行烧录/串口仿真，手动停电/上电，5V/3.3V原理图



备注：上图中红色管脚部分是 2024 年 10 月 15 日之后生产的芯片才有。

比较下传统的 12C5A60S2 相应下载线路图：



【ISP下载/编程/烧录，操作步骤】

1、点击电脑端 ISP 软件的【下载/编程】按钮

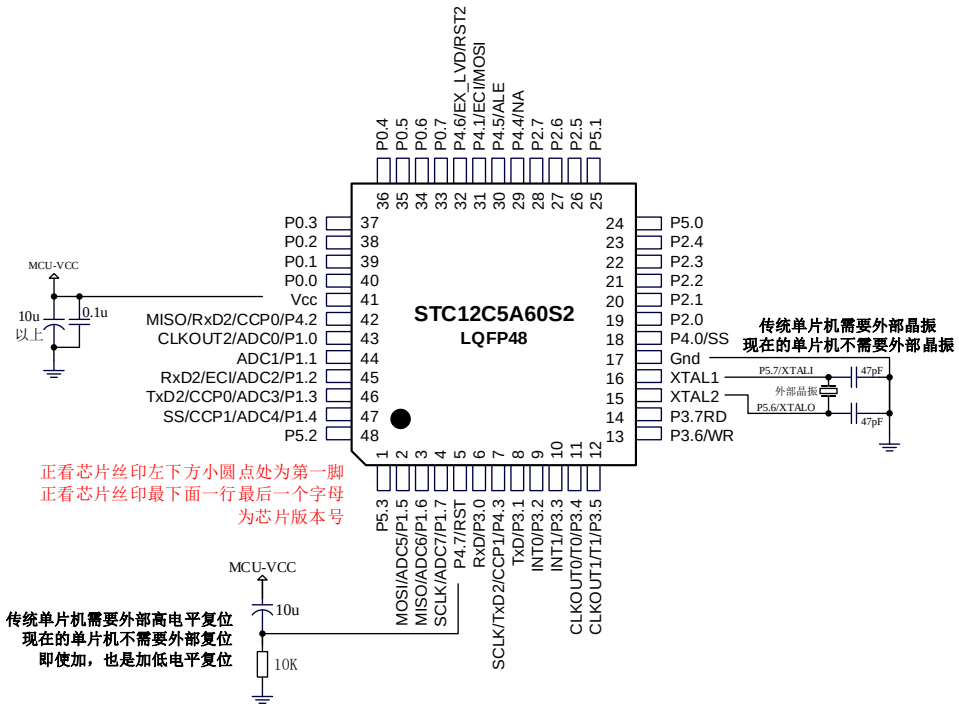
2、给目标系统上电，或者重新给目标系统上电

如果在点击【下载/编程】按钮前，目标系统已上电，则需要停电再重新上电

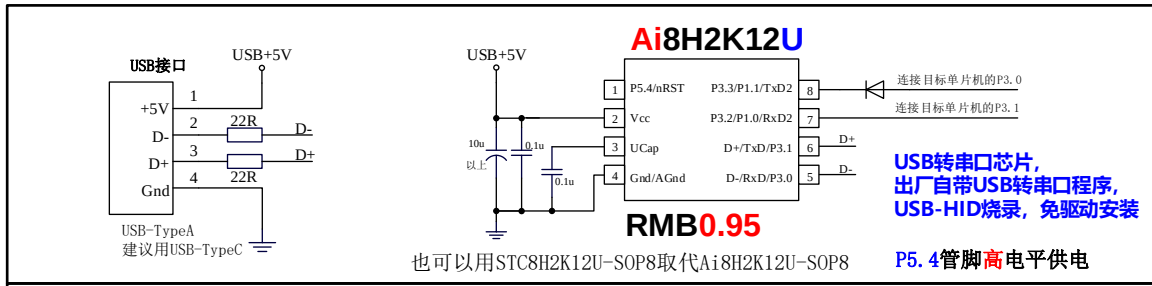
电脑端软件提示：下载编程进行中，数秒后提示成功

传统单片机需要外部晶振，现在的单片机不需要外部晶振

传统单片机需要外部高电平复位，现在的不需要，并且已改为低电平复位



4.1.17 USB 转串口芯片进行烧录，串口仿真，手动停电/上电，3.3V 原理图



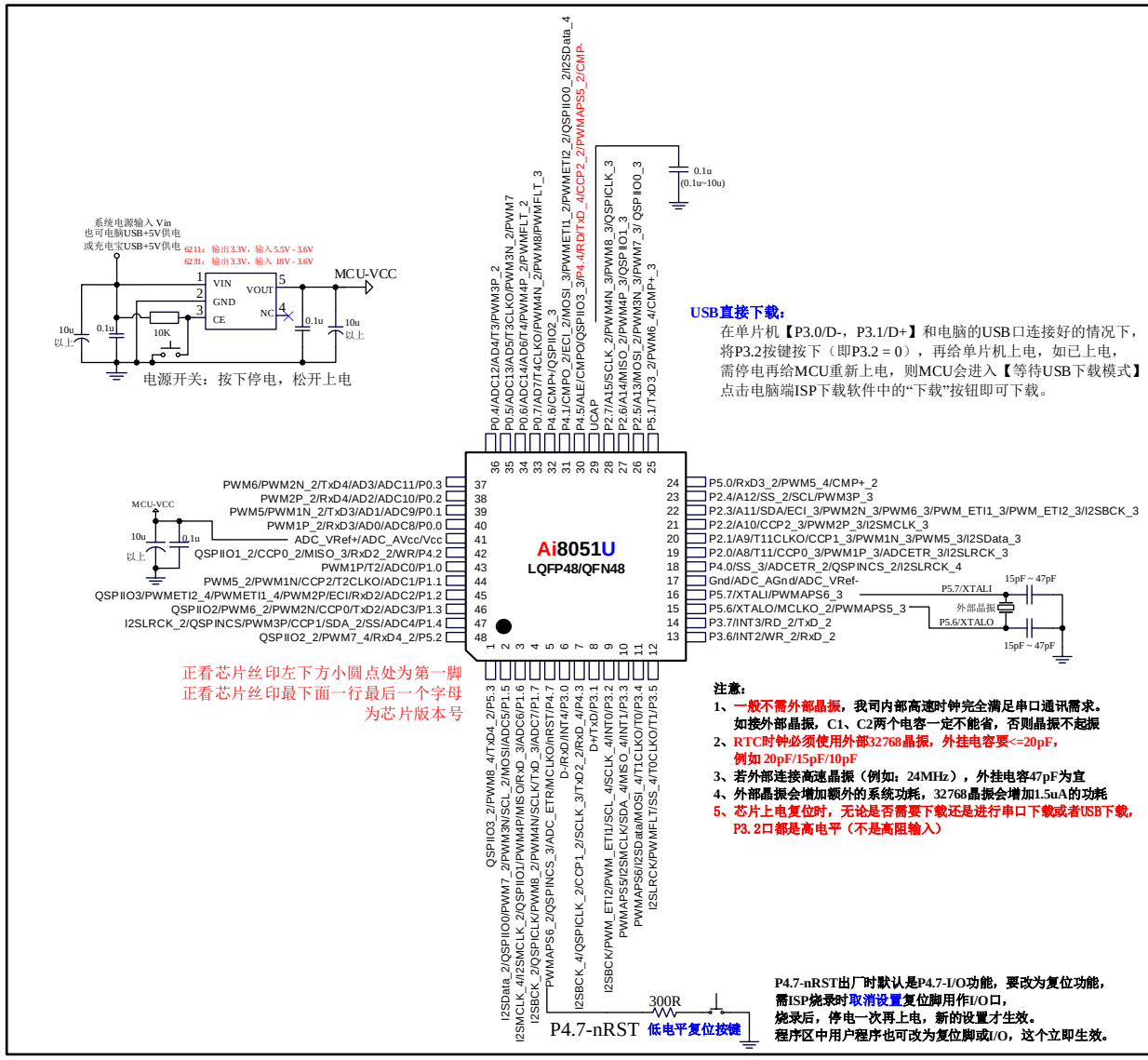
【ISP下载/编程/烧录，操作步骤】

1、点击电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新给目标系统上电

如果在点击【下载/编程】按钮前，目标系统已上电，则需要停电再重新上电

电脑端软件提示：下载编程进行中，数秒后提示成功



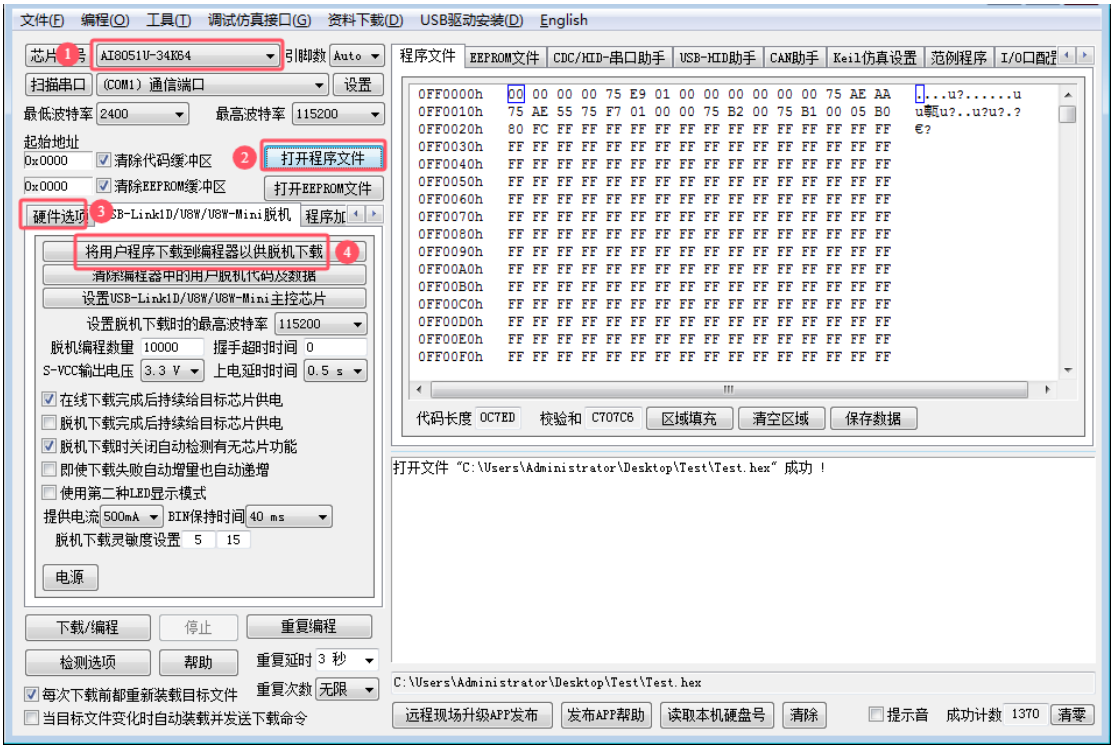
备注：上图中红色管脚部分是 2024 年 10 月 15 日之后生产的芯片才有。

4.1.18 USB-Link1D 支持 脱机下载 说明

脱机下载是指脱离电脑主机进行下载的一种方法，一般是使用下载控制芯片（又称脱机下载母片）进行控制。USB-Link1D 工具除了支持在线 ISP 下载，还支持脱机下载。USB-Link1D 工具使用外部的 22.1184MHz 晶振，可保证对目标芯片进行在线或脱机下载时，校准频率的精度。用户可将代码下载到 USB-Link1D 工具中，就可实现脱机下载。USB-Link1D 主控芯片如内部存储空间不够时，会将部分被下载的用户程序用加密的方式加密放部分到片外串行 Flash。

先将 USB-Link1D 工具使用 USB 线连接到电脑，然后按照下面的步骤进行脱机下载：

1、选择目标芯片的型号
2、打开需要下载的文件
3、设置硬件选项
4、进入“USB-Link1D 脱机”页面，点击“将用户程序下载到编程器以供脱机下载“按钮，即可将用户代码下载到 USB-Linik1D 工具中。下载完成后便使用 USB-Link1D 工具对目标芯片进行脱机下载了



4.1.19　USB-Link1D 支持 脱机下载，如何免烧录环节

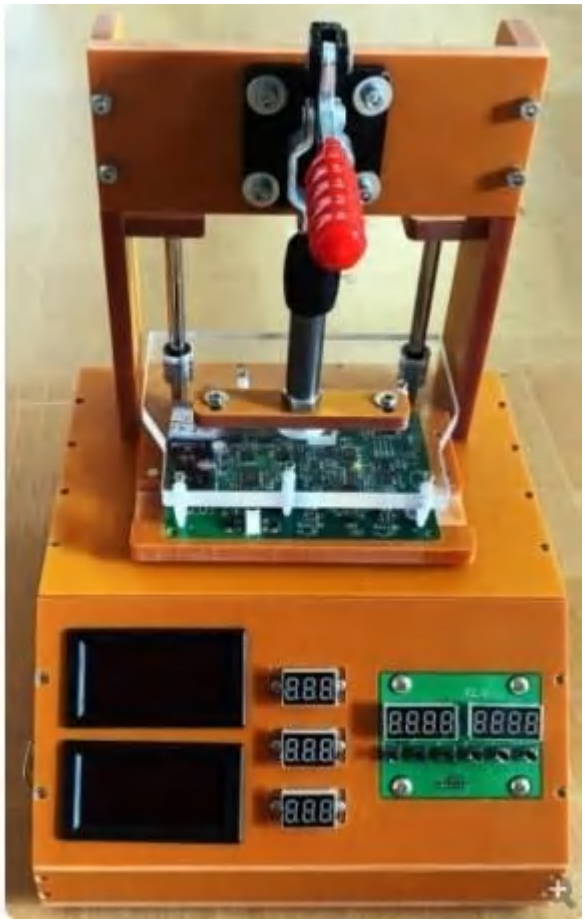
大批量生产，如何省去专门的烧录人员，如何无烧录环节
大批量生产，你在将由 STC 的 MCU 作为主控芯片的控制板组装到设备里面之前，在你将 STC MCU 贴片到你的控制板完成之后，你必须测试你的控制板的好坏。不要说 100%，直通无问题，那是抬杠，不是搞生产，只要生产，就会虚焊，短路，部分元件贴错，部分元件采购错。
所以在贴片回来后，组装到外壳里面之前，你必须要测试，你的含有 STC MCU 控制板的好坏，好的去组装，坏的去维修抢救。

控制板的测试/不是烧录！

控制板的测试环节必须有，但烧录环节可以省！

大批量生产，必须要有方便控制板测试的测试架/下面接上我们的脱机烧录工具：
USB-Link1D / U8W-Mini / U8W，还要接上其他控制部分！
1、通过 USER-VCC、P3.0、P3.1、GND 连接，要工人每次都开电源
2、通过 S-VCC、P3.0、P3.1、GND 连接，不要你开电源，STC 的脱机工具给你自动供电

外面帮你做一个测试架的成本 500 元以下，就是有机玻璃，夹具，顶针。
1 个测试你控制板是否正常的工人管理 2 - 3 个 测试架

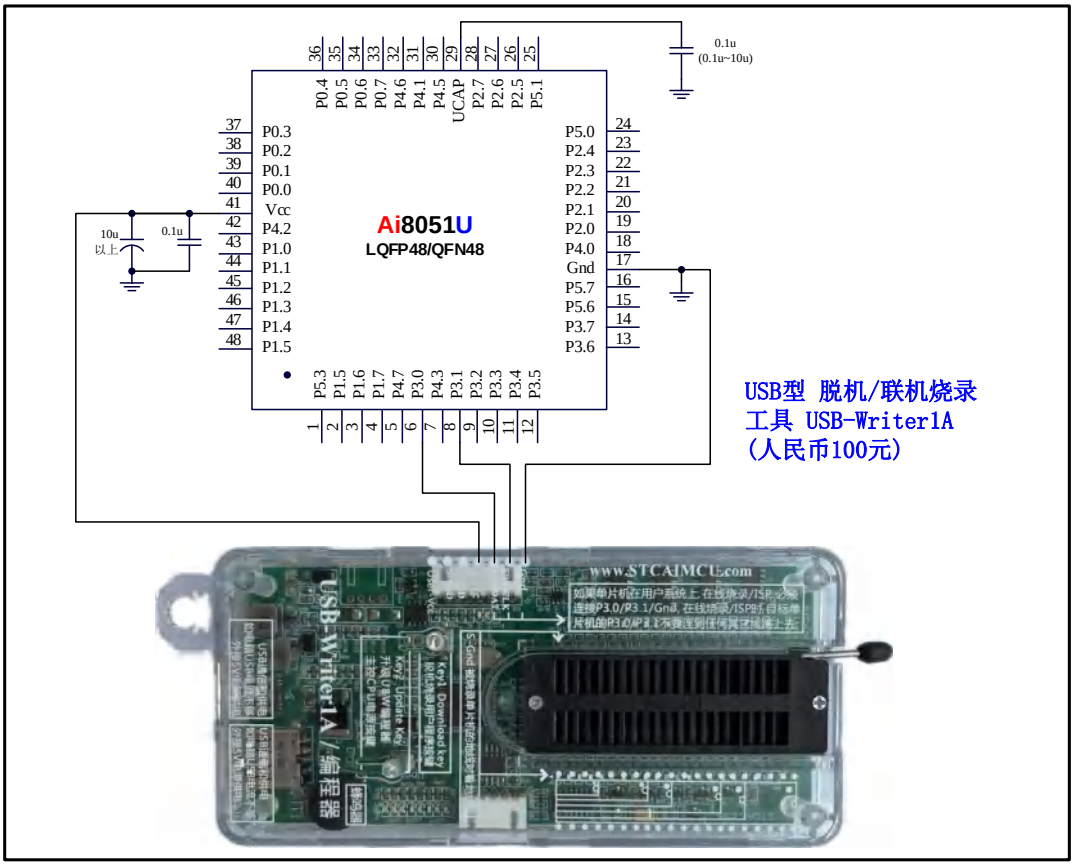


操作流程：

1、将你的 MCU 控制板 卡到测试架 1 上

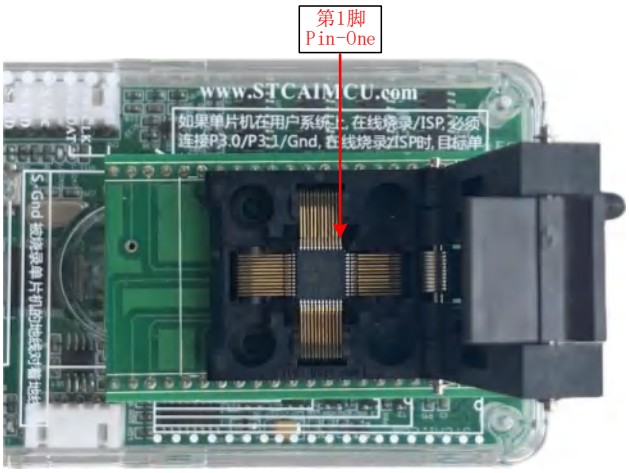
2、将你的 MCU 控制板 卡到测试架 2 上，测试架 1 上的程序已烧录完成/感觉不到烧录时间
3、测试 测试架 1 上的 MCU 主控板功能是否正常，正常放到正常区，不正常，放到不正常区
4、给测试架 1 卡上新的未测试的无程序的控制板
5、测试 测试架 2 上的未测试控制板/程序不知何时早就不知不觉的烧好了，换新的未测试未烧录的控制板
6、循环步骤 3 到步骤 5
========不需要安排烧录人员

4.1.20 USB-Writer1A 编程器/烧录器 • 支持 • 插在 • 锁紧座上 • 烧录

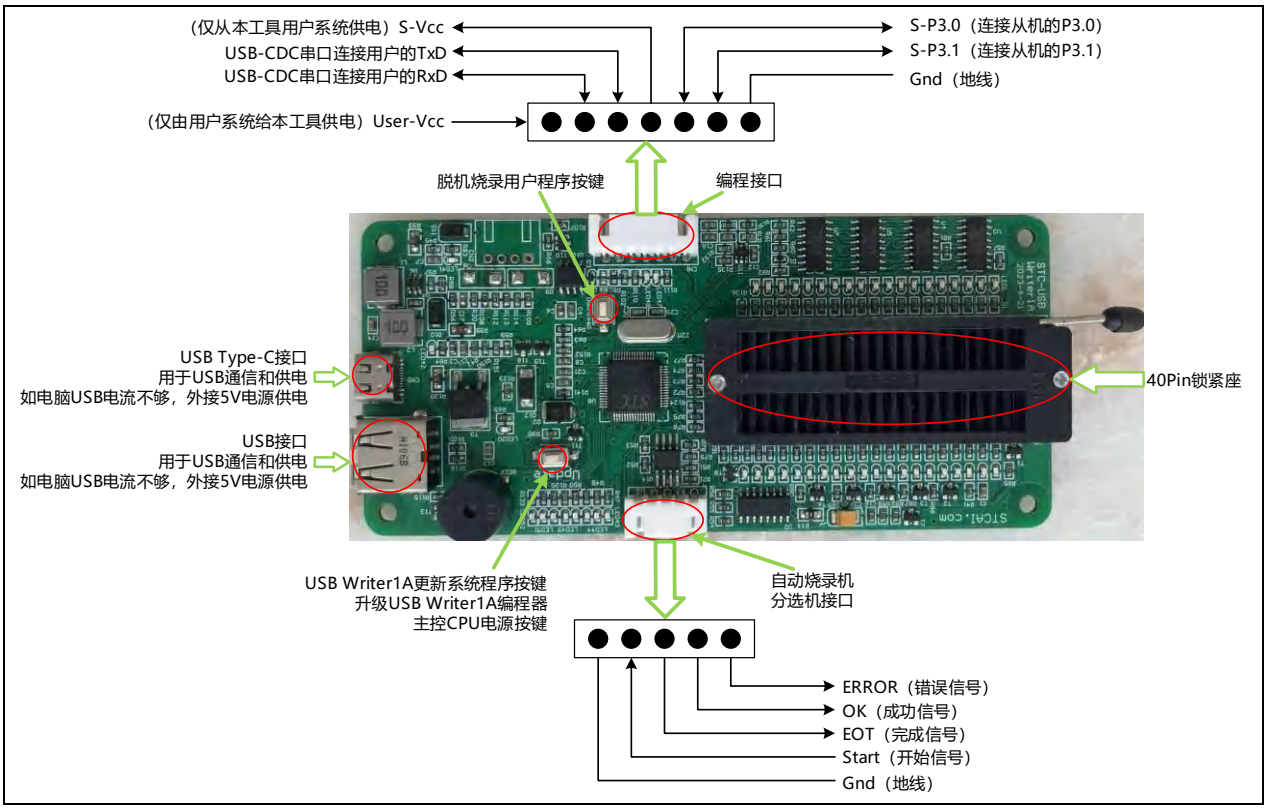


芯片可直接放在此锁紧座上进行传统方式的 ISP 编程烧录

也可如上图所示进行引线 ISP下载烧录



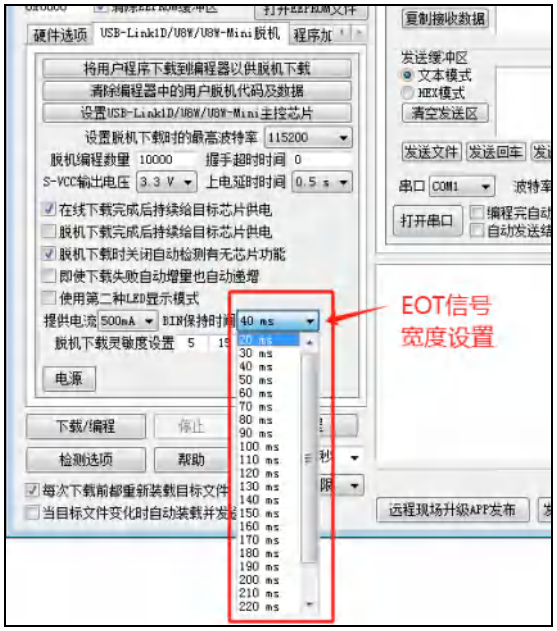
4.1.21 USB-Writer1A 支持 自动烧录机，通信协议和接口



自动烧录接口（分选机自动控制接口）协议：

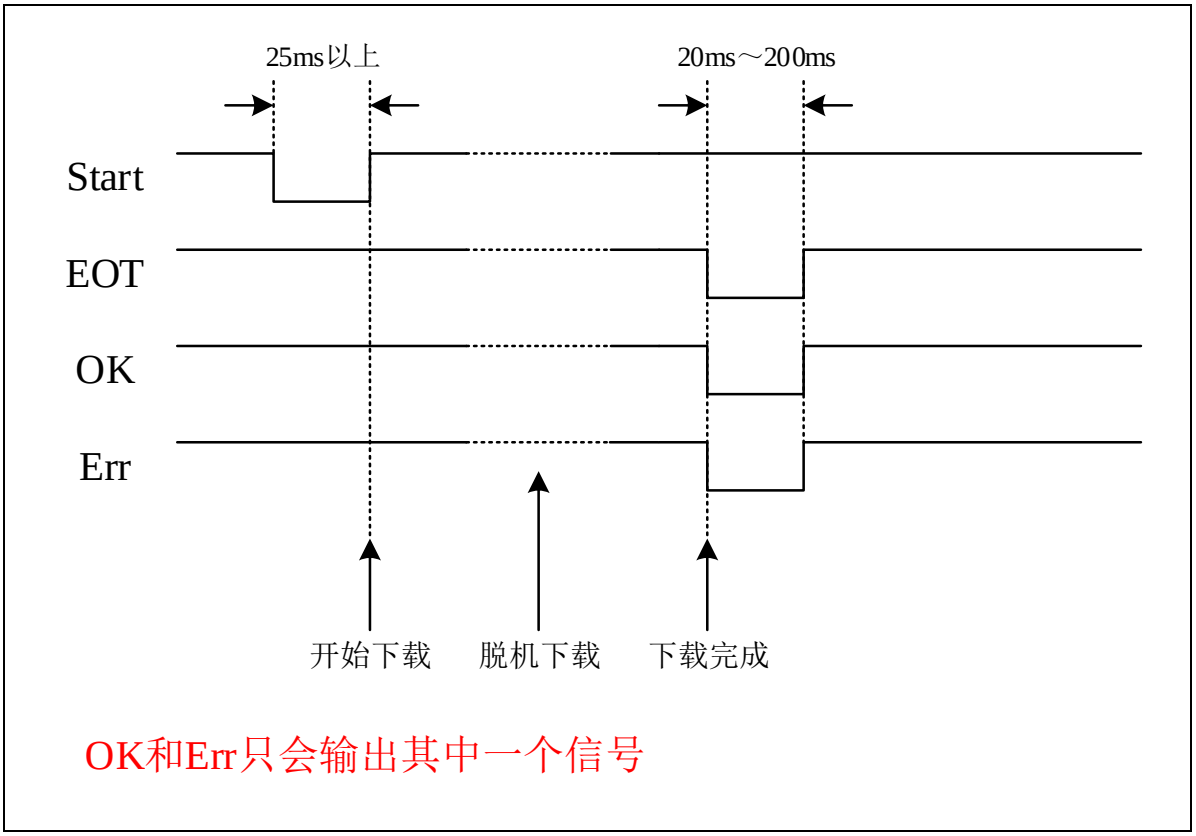
Start： 开始信号输入脚。从外部输入低电平信号触发开始脱机烧录，低电平必须维持 25 毫秒以上

EOT： 烧录完成信号输出脚。脱机烧录完成后，工具输出 20ms～250ms 的低电平 EOT 信号。电平宽度如下图所示的地方进行设置



OK： 良品信号输出脚。下载成功后工具从 OK 脚输出低电平信号，信号与 EOT 信号同步。

Err： 不良品信号输出脚。若下载失败，工具从 ERR 脚输出输出低电平信号，信号与 EOT 完成信号同步。



4.2 LQFP44 管脚图，USB-ISP 下载，各种 烧录/仿真 线路图

4.2.1 LQFP44 管脚图，5V 线路，USB-ISP 下载/烧录/仿真

备注：上图中红色管脚部分是 2024 年 10 月 15 日之后生产的芯片才有。

现在带硬件 USB 的 MCU 支持用硬件 USB 下载，因为用的是 USB-HID 通信协议，不需要安装任何驱动。只要 USB 鼠标、USB 键盘能工作，USB-HID 驱动就是好的，不要安装 USB-HID 驱动，免驱。

在 D-/P3.0，D+/P3.1 与 PC-USB 端口连接好的状况下，USB-ISP 下载程序有如下三种模式：

【USB 下载方法一，P3.2 按键，再结合停电上电下载】

1、按下板子上的 P3.2/INT0 按键，就是 P3.2 接地

2、给目标芯片重新上电，不管之前是否已通电。

===电子开关是按下停电后再松开就是上电

等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后，就与 P3.2 状态无关了，这时可以松

开 P3.2 按键（P3.2 在用户程序区可做其它任意用途）

===传统的机械自锁紧开关是按上来停电，按下去是上电

3、点击电脑端下载软件中的【下载/编程】按钮（注意：USB 下载与串口下载的操作顺序不同）

下载进行中，几秒钟后，提示下载成功！

【USB 下载方法二，复位管脚低电平复位下载】

USB 连接好并已上电的情况下，外部按键复位也可进入 USB 下载模式，注意：P4.7-nRST 出厂时默认是 P4.7-I/O 功能，要改为复位功能，需 ISP 烧录时取消设置复位脚用作 I/O 口，停电一次再上电才生效，程序区中用户程序也可改为复位脚或 I/O，这个立即生效。

1、按下 P4.7-nRST 外接的低电平复位按键复位 MCU，

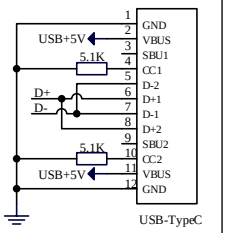
松开复位键，MCU 从系统程序区启动，判断是否要下载用户程序，

等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后

2、点击电脑端下载软件中的【下载/编程】按钮

下载进行中，几秒钟后，提示下载成功！

比较一下流行的 USB-TypeC接口：



备注：如果目标 MCU 的用户程序区已有 USB 通讯程序（含 CDC、HID 等），则通过复位脚复位进入 USB 下载模式不被支持，只能通过停电上电冷启动下载或者从用户程序区软复位到系统区下载

【USB 下载方法三，从用户程序区软复位到系统区下载】

USB 连接好并已上电的情况下，从用户程序区软复位到系统区也可进入 USB 下载模式

1、在用户程序区运行软复位到系统区的程序，就是 IAP_CONTR 寄存器送 60H

等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后

2、点击电脑端下载软件中的【下载/编程】按钮，下载进行中，几秒钟后，提示下载成功！

USB 下载 注意事项：

拔插 USB 插头不能代替上面线路图中的电源开关。正常操作步骤如下：

USB 的【Gnd，D+，D-】接好的情况下，按下 P3.2 按键接地，再通过正常的电源开关给 MCU 供电或重新供电，让 MCU 冷启动进入系统程序区，判断是否需要等待电脑端 USB 下载程序。

拔插 USB 插头代替电源开关不能每次都能成功进行 USB 下载的原因：

拔插 USB 插头，如【Gnd，USB+5V】已接触好，已供电，而【D+，D-】有一个甚至两个信号线还没有接触好，MCU 已上电，开始跑系统区程序时，发现 USB 还没接触好，则会从系统区软复位到用户程序区跑用户程序，不再进入等待 USB 下载模式，本次就无法顺利进行 USB 下载。

很多人经过多次插拔 USB，才能碰到 1 次【D+，D-】接触好的情况下，【Gnd，USB+5V】才开始接触好，才开始供电，才能成功进入 USB 下载。插 USB 插头代替供电，不能保证【Gnd，D+，D-，USB+5V】的接触顺序，所以，必须使用正常的电源开关，才能确保每次下载都能成功。

关于 I/O 的注意事项：

1、 P3.0 和 P3.1 口上电后的状态为弱上拉/准双向口模式

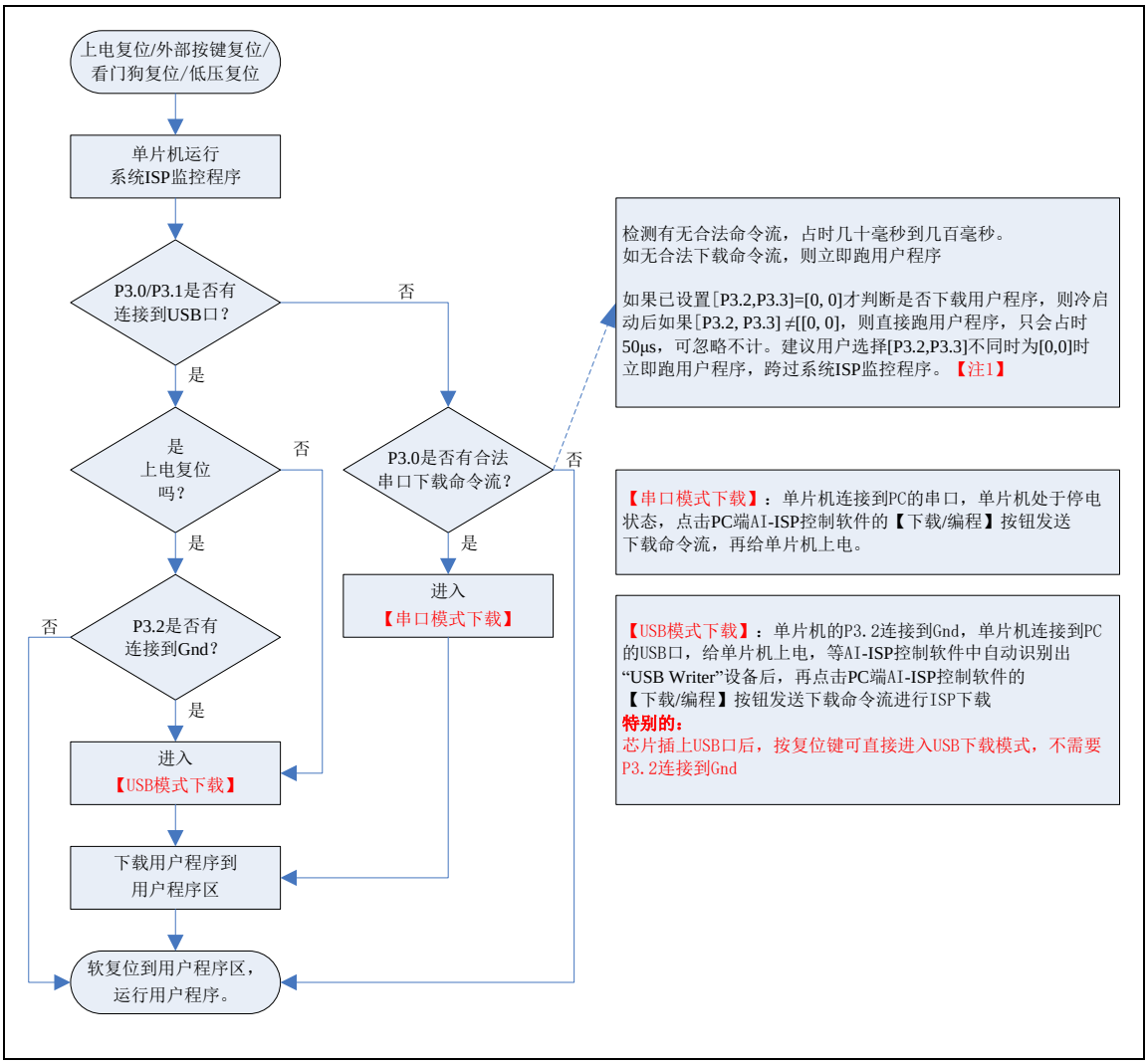
2、 除 P3.0 和 P3.1 外，其余所有 I/O 口上电后的状态均为高阻输入状态，用户在使用 I/O 口前必须先设置 I/O 口模式

3、 芯片上电时，若 P3.0 和 P3.1 同时为低电平，P3.2 口会短时间开启内部 4K 上拉，用以读取 P3.2 口外部状态来判断是否需要进入 USB 下载模式

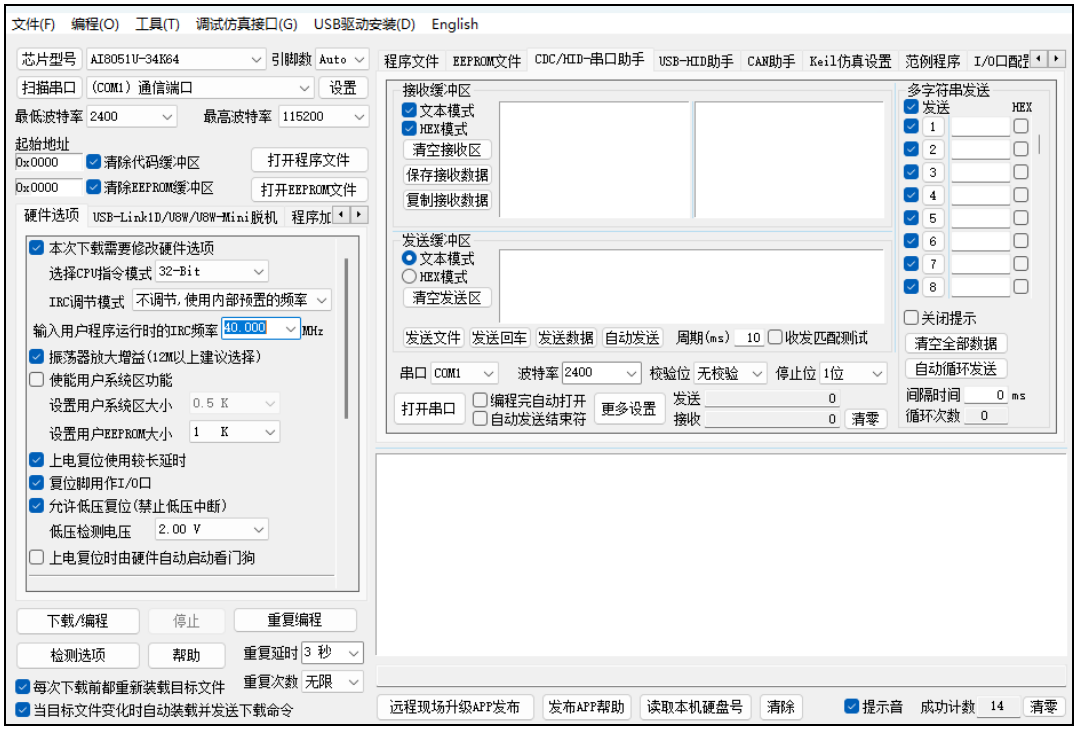
4、 芯片上电时如果不需要使用 USB 进行 ISP 下载，P3.0/P3.1/P3.2 这 3 个 I/O 口不能同时为低电平，否则会进入 USB 下载模式而无法运行用户代码

5、 当使用 P4.7 当作复位脚时，这个端口内部的 4K 上拉电阻会一直打开；但 P4.7 做普通 I/O 口时，基于这个 I/O 口与复位脚共享管脚的特殊考量，端口内部的 4K 上拉电阻依然会打开大约 6.5 毫秒时间，再自动关闭（当用户的电路设计需要使用 P4.7 口驱动外部电路时，请务必考虑上电瞬间会有 6.5 毫秒时间的高电平的问题）

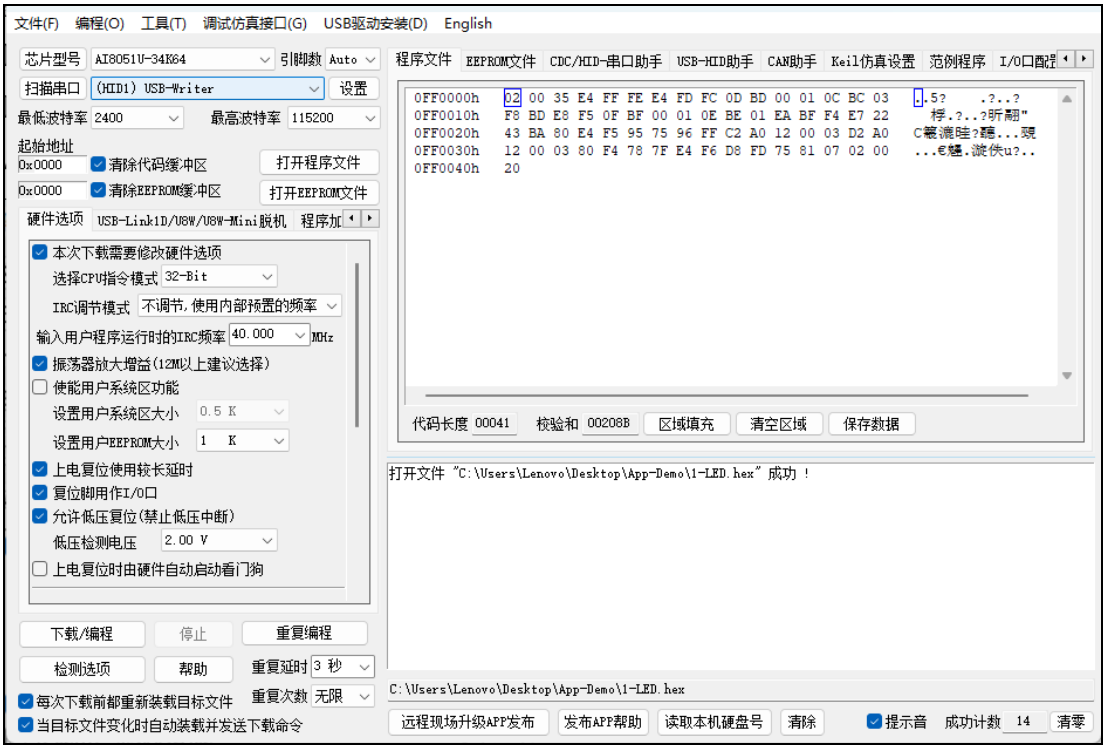
ISP 下载流程图（硬件/软件模拟 USB+串口模式）：



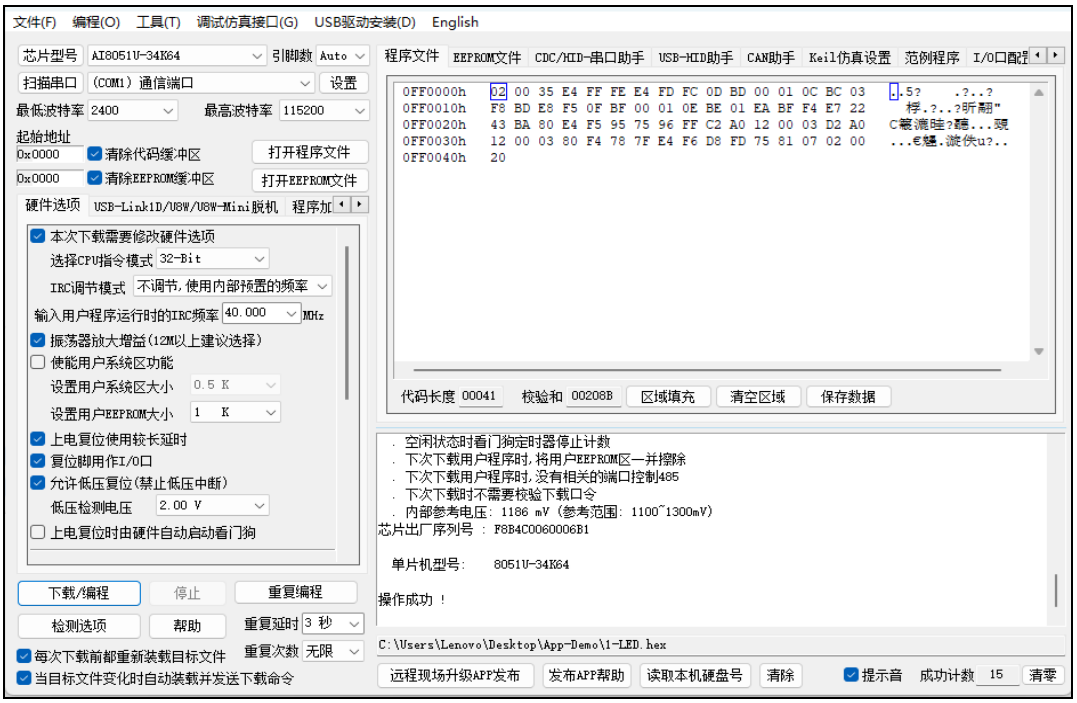
打开 AIapp-ISP-V6.94S 以上版本软件，如下图所示：



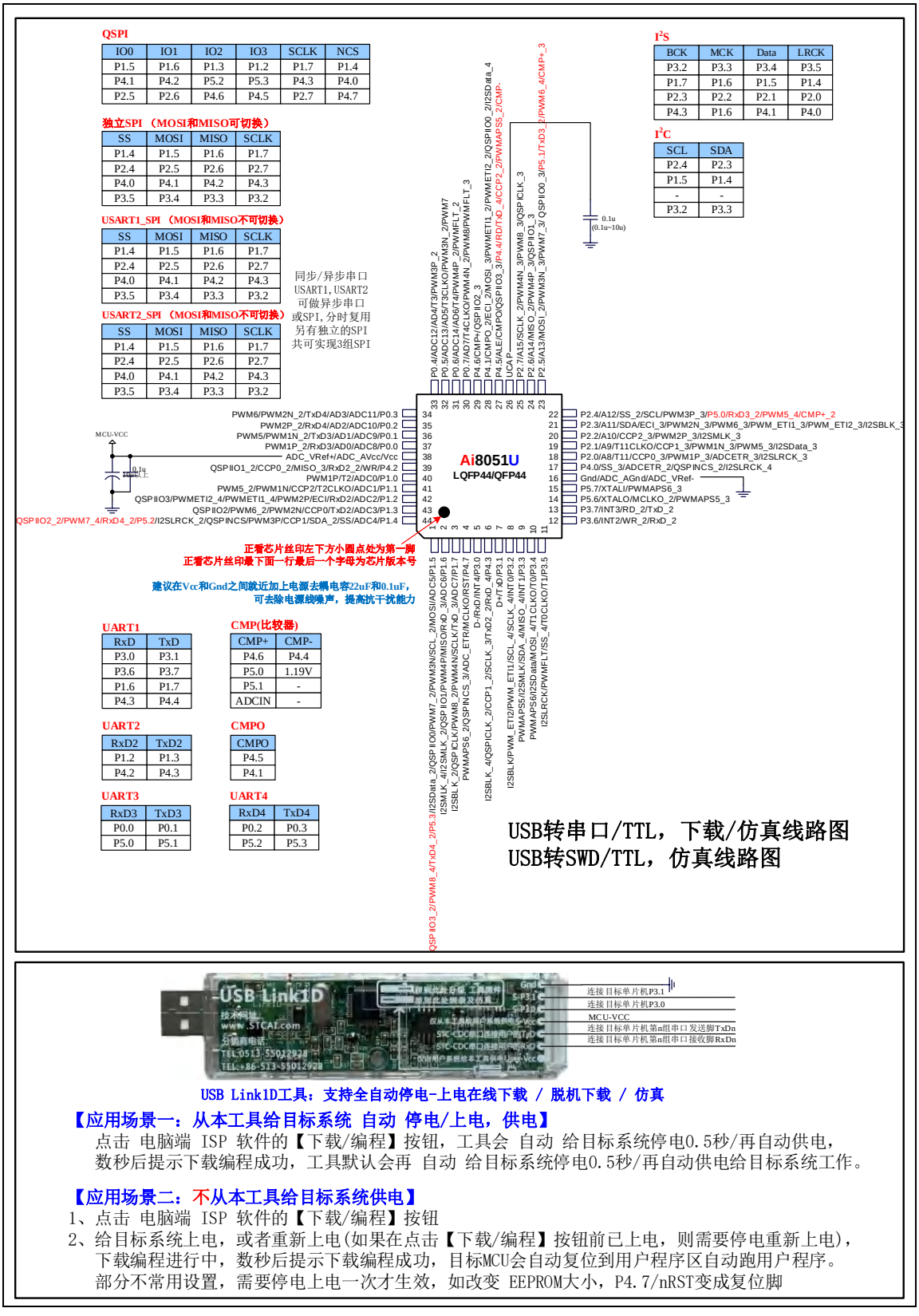
选择好对应的正确型号，打开要烧录的文件
将 USB 数据线，如前面的原理图连接好，
注意是【USB+5V，D-，D+，Gnd】USB 数据线，不是【USB+5V，NC，NC，Gnd】USB 电源线
在 P3.2 接地按键按下的状态下，
给 MCU 上电，或重新上电。
则 AIapp-ISP 软件出现如下显示：



点击“下载/编程”按钮，
则会如下图显示：正在下载用户代码，操作成功！



如上硬件 USB，ISP 下载/编程 烧录成功。



USB Link1D工具：支持全自动停电-上电在线下载 / 脱机下载 / 仿真

【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，下载编程进行中，数秒后提示下载编程成功，目标MCU会自动复位到用户程序区自动跑用户程序。

部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P4.7/nRST变成复位脚

备注：上图中红色管脚部分是 2024 年 10 月 15 日之后生产的芯片才有。

USB 转串口/TTL，全自动 ISP 下载步骤：

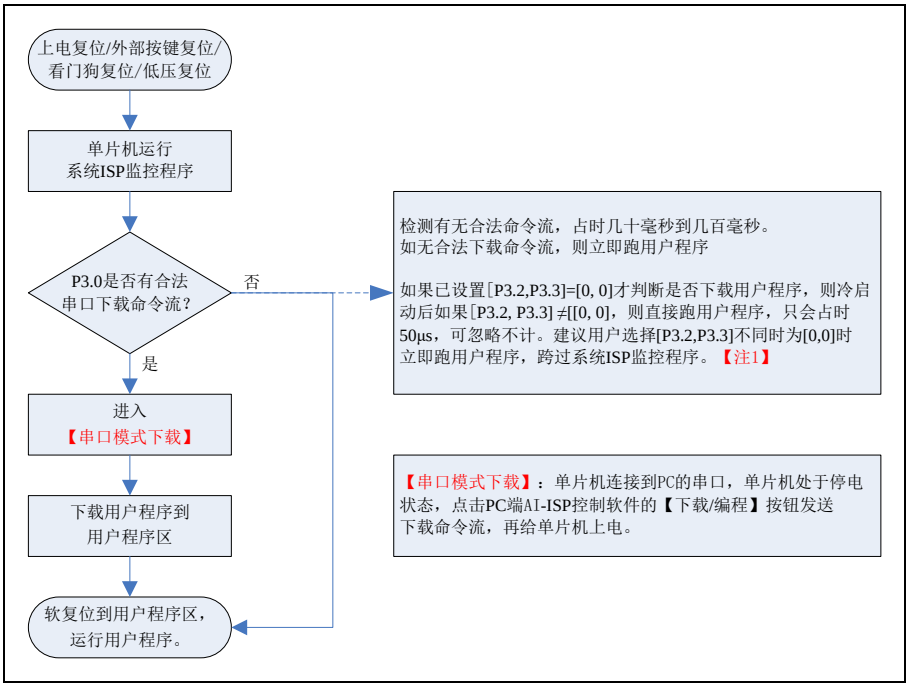
1、按照如图所示的连接方式将 USB-Link1D 和目标芯片连接

2、点击 ISP 下载软件中的“下载/编程”按钮

3、开始 ISP 下载

（注意：若是使用 USB-Link1D 给目标系统供电，目标系统的总电流不能大于 200mA，否则会导致下载失败。）

ISP 下载流程图（串口下载模式）



将【MCU-VCC，P3.0，P3.1，Gnd】，如前面的原理图连接到 USB-Link1D 工具;

将 USB-Link1D 全自动烧录工具，通过 USB 数据线连接到电脑，【USB+5V，D-，D+，Gnd】

打开 AIapp-ISP-V6.94S 以上版本软件

则 AIapp-ISP 软件显示如下：

选择好对应的正确型号，
打开要烧录的文件
点击“下载/编程”按钮，全自动烧录
则 AIapp-ISP 软件出现如下显示：

关于 I/O 的注意事项：

1、 P3.0 和 P3.1 口上电后的状态为弱上拉/准双向口模式
2、 除 P3.0 和 P3.1 外，其余所有 I/O 口上电后的状态均为高阻输入状态，用户在使用 I/O 口前必须先设置 I/O 口模式
3、 芯片上电时，若 P3.0 和 P3.1 同时为低电平，P3.2 口会短时间开启内部 4K 上拉，用以读取 P3.2 口外部状态来判断是否需要进入 USB 下载模式
4、 芯片上电时如果不需要使用 USB 进行 ISP 下载，P3.0/P3.1/P3.2 这 3 个 I/O 口不能同时为低电平，否则会进入 USB 下载模式而无法运行用户代码
5、 当使用 P4.7 当作复位脚时，这个端口内部的 4K 上拉电阻会一直打开；但 P4.7 做普通 I/O 口时，基于这个 I/O 口与复位脚共享管脚的特殊考量，端口内部的 4K 上拉电阻依然会打开大约 6.5 毫秒时间，再自动关闭（当用户的电路设计需要使用 P4.7 口驱动外部电路时，请务必考虑上电瞬间会有 6.5 毫秒时间的高电平的问题）

4.2.2 LQFP44 管脚图，3.3V 线路一，USB-ISP 下载/仿真

备注：上图中红色管脚部分是 2024 年 10 月 15 日之后生产的芯片才有。

现在带硬件 USB 的 MCU 支持用硬件 USB 下载，因为用的是 USB-HID 通信协议，不需要安装任何驱动。只要 USB 鼠标、USB 键盘能工作，USB-HID 驱动就是好的，不要安装 USB-HID 驱动，免驱。

在 D-/P3.0，D+/P3.1 与 PC-USB 端口连接好的状况下，USB-ISP 下载程序有如下三种模式：

【USB 下载方法一，P3.2 按键，再结合停电上电下载】

1、按下板子上的 P3.2/INT0 按键，就是 P3.2 接地

2、给目标芯片重新上电，不管之前是否已通电。

===电子开关是按下停电后再松开就是上电

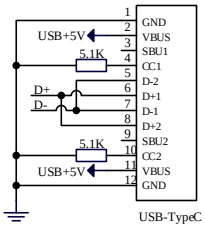
等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后，就与 P3.2 状态无关了，这时可以松开 P3.2 按键（P3.2 在用户程序区可做其它任意用途）

===传统的机械自锁紧开关是按上来停电，按下去是上电

3、点击电脑端下载软件中的【下载/编程】按钮（注意：USB 下载与串口下载的操作顺序不同）

下载进行中，几秒钟后，提示下载成功！

比较一下流行的 USB-TypeC接口：



【USB 下载方法二，复位管脚低电平复位下载】

USB 连接好并已上电的情况下，外部按键复位也可进入 USB 下载模式，注意：

P4.7-nRST 出厂时默认是 P4.7-I/O 功能，要改为复位功能，需 ISP 烧录时取消设置复位脚用作 I/O 口，停电一次再上电才生效，程序区中用户程序也可改为复位脚或 I/O，这个立即生效。

1、按下 P4.7-nRST 外接的低电平复位按键复位 MCU，
松开复位键，MCU 从系统程序区启动，判断是否要下载用户程序，
等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后

2、点击电脑端下载软件中的【下载/编程】按钮
下载进行中，几秒钟后，提示下载成功！

备注：如果目标 MCU 的用户程序区已有 USB 通讯程序（含 CDC、HID 等），则通过复位脚复位进入 USB 下载模式不被支持，只能通过停电上电冷启动下载或者从用户程序区软复位到系统区下载

【USB 下载方法三，从用户程序区软复位到系统区下载】

USB 连接好并已上电的情况下，从用户程序区软复位到系统区也可进入 USB 下载模式

1、在用户程序区运行软复位到系统区的程序，就是 IAP_CONTR 寄存器送 60H
等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后

2、点击电脑端下载软件中的【下载/编程】按钮
下载进行中，几秒钟后，提示下载成功！

USB 下载 注意事项:

拔插 USB 插头不能代替上面线路图中的电源开关。正常操作步骤如下：

USB 的【Gnd，D+，D-】接好的情况下，按下 P3.2 按键接地，再通过正常的电源开关给 MCU 供电或重新供电，让 MCU 冷启动进入系统程序区，判断是否需要等待电脑端 USB 下载程序。

拔插 USB 插头代替电源开关不能每次都能成功进行 USB 下载的原因：

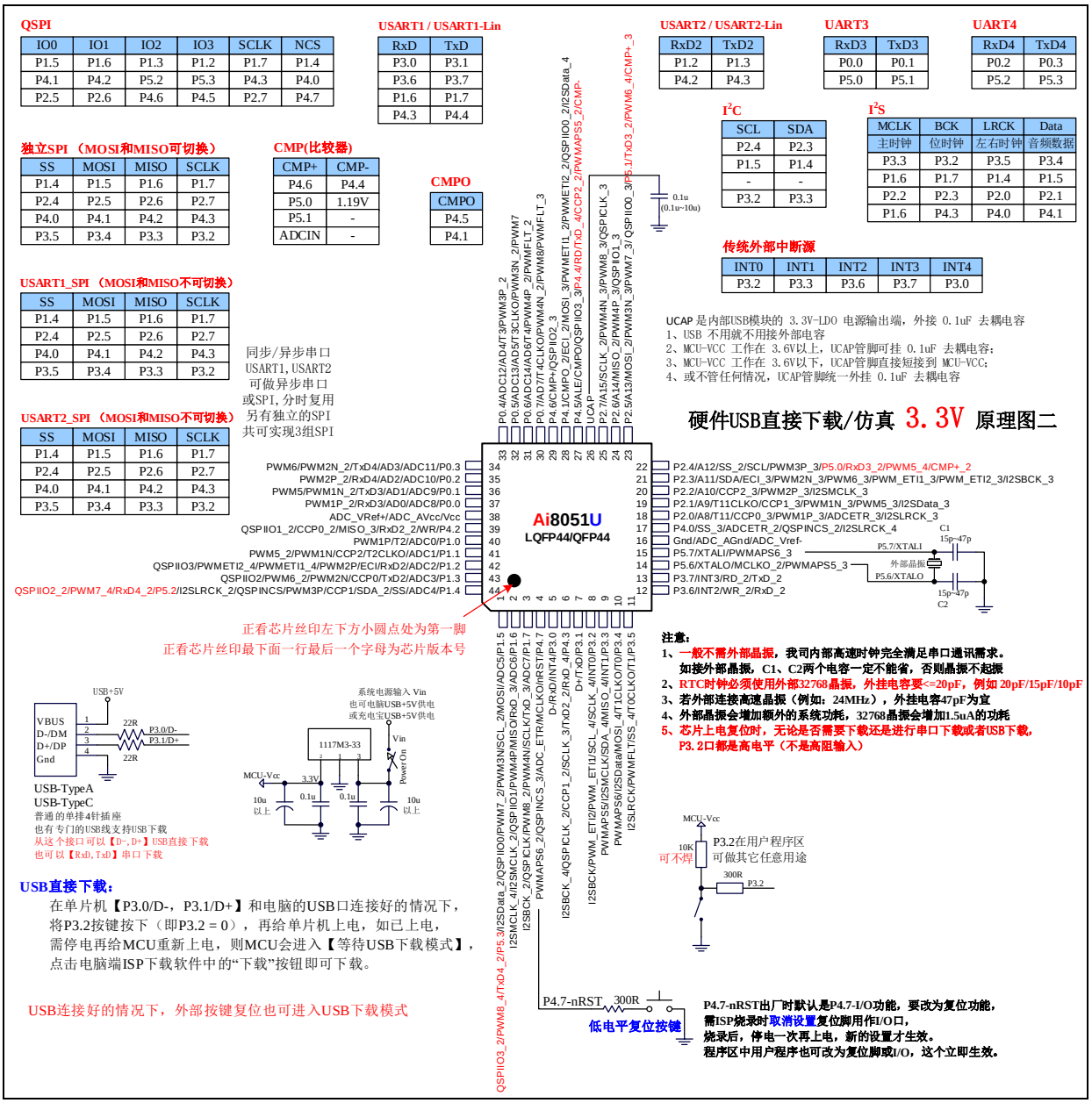
拔插 USB 插头，如【Gnd，USB+5V】已接触好，已供电，而【D+，D-】有一个甚至两个信号线还没有接触好，MCU 已上电，开始跑系统区程序时，发现 USB 还没接触好，则会从系统区软复位到用户程序区跑用户程序，不再进入等待 USB 下载模式，本次就无法顺利进行 USB 下载。

很多人经过多次插拔 USB，才能碰到 1 次【D+，D-】接触好的情况下，【Gnd，USB+5V】才开始接触好，才开始供电，才能成功进入 USB 下载。插 USB 插头代替供电，不能保证【Gnd，D+，D-，USB+5V】的接触顺序，所以，必须使用正常的电源开关，才能确保每次下载都能成功。

关于 I/O 的注意事项:

1、 P3.0 和 P3.1 口上电后的状态为弱上拉/准双向口模式
2、 除 P3.0 和 P3.1 外，其余所有 I/O 口上电后的状态均为高阻输入状态，用户在使用 I/O 口前必须先设置 I/O 口模式
3、 芯片上电时，若 P3.0 和 P3.1 同时为低电平，P3.2 口会短时间开启内部 4K 上拉，用以读取 P3.2 口外部状态来判断是否需要进入 USB 下载模式
4、 芯片上电时如果不需要使用 USB 进行 ISP 下载，P3.0/P3.1/P3.2 这 3 个 I/O 口不能同时为低电平，否则会进入 USB 下载模式而无法运行用户代码
5、 当使用 P4.7 当作复位脚时，这个端口内部的 4K 上拉电阻会一直打开；但 P4.7 做普通 I/O 口时，基于这个 I/O 口与复位脚共享管脚的特殊考量，端口内部的 4K 上拉电阻依然会打开大约 6.5 毫秒时间，再自动关闭（当用户的电路设计需要使用 P4.7 口驱动外部电路时，请务必考虑上电瞬间会有 6.5 毫秒时间的高电平的问题）

4.2.3　LQFP44 管脚图，3.3V 线路二，USB-ISP 下载/仿真



备注：上图中红色管脚部分是 2024 年 10 月 15 日之后生产的芯片才有。

现在带硬件 USB 的 MCU 支持用硬件 USB 下载，因为用的是 USB-HID 通信协议，不需要安装任何驱动。只要 USB 鼠标、USB 键盘能工作，USB-HID 驱动就是好的，不要安装 USB-HID 驱动，免驱。

在 D-/P3.0，D+/P3.1 与 PC-USB 端口连接好的状况下，USB-ISP 下载程序有如下三种模式：

【USB 下载方法一，P3.2 按键，再结合停电上电下载】

1、按下板子上的 P3.2/INT0 按键，就是 P3.2 接地

2、给目标芯片重新上电，不管之前是否已通电。

===电子开关是按下停电后再松开就是上电

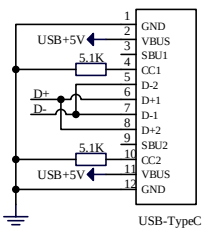
等待电脑端 ISP 下载软件中自动识别出"(HID1) USB Writer"后，就与 P3.2 状态无关了，这时可以松开 P3.2 按键（P3.2 在用户程序区可做其它任意用途）

===传统的机械自锁紧开关是按上来停电，按下去是上电

3、点击电脑端下载软件中的【下载/编程】按钮（注意：USB 下载与串口下载的操作顺序不同）

下载进行中，几秒钟后，提示下载成功！

比较一下流行的 USB-TypeC接口：



【USB 下载方法二，复位管脚低电平复位下载】

USB 连接好并已上电的情况下，外部按键复位也可进入 USB 下载模式，注意：

P4.7-nRST 出厂时默认是 P4.7-I/O 功能，要改为复位功能，需 ISP 烧录时取消设置复位脚用作 I/O 口，停电一次再上电才生效，程序区中用户程序也可改为复位脚或 I/O，这个立即生效。

1、按下 P4.7-nRST 外接的低电平复位按键复位 MCU，
松开复位键，MCU 从系统程序区启动，判断是否要下载用户程序，
等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后

2、点击电脑端下载软件中的【下载/编程】按钮
下载进行中，几秒钟后，提示下载成功！

备注：如果目标 MCU 的用户程序区已有 USB 通讯程序（含 CDC、HID 等），则通过复位脚复位进入 USB 下载模式不被支持，只能通过停电上电冷启动下载或者从用户程序区软复位到系统区下载

【USB 下载方法三，从用户程序区软复位到系统区下载】

USB 连接好并已上电的情况下，从用户程序区软复位到系统区也可进入 USB 下载模式

1、在用户程序区运行软复位到系统区的程序，就是 IAP_CONTR 寄存器送 60H
等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后

2、点击电脑端下载软件中的【下载/编程】按钮
下载进行中，几秒钟后，提示下载成功！

USB 下载 注意事项：

拔插 USB 插头不能代替上面线路图中的电源开关。正常操作步骤如下：

USB 的【Gnd，D+，D-】接好的情况下，按下 P3.2 按键接地，再通过正常的电源开关给 MCU 供电或重新供电，让 MCU 冷启动进入系统程序区，判断是否需要等待电脑端 USB 下载程序。

拔插 USB 插头代替电源开关不能每次都能成功进行 USB 下载的原因：

拔插 USB 插头，如【Gnd，USB+5V】已接触好，已供电，而【D+，D-】有一个甚至两个信号线还没有接触好，MCU 已上电，开始跑系统区程序时，发现 USB 还没接触好，则会从系统区软复位到用户程序区跑用户程序，不再进入等待 USB 下载模式，本次就无法顺利进行 USB 下载。

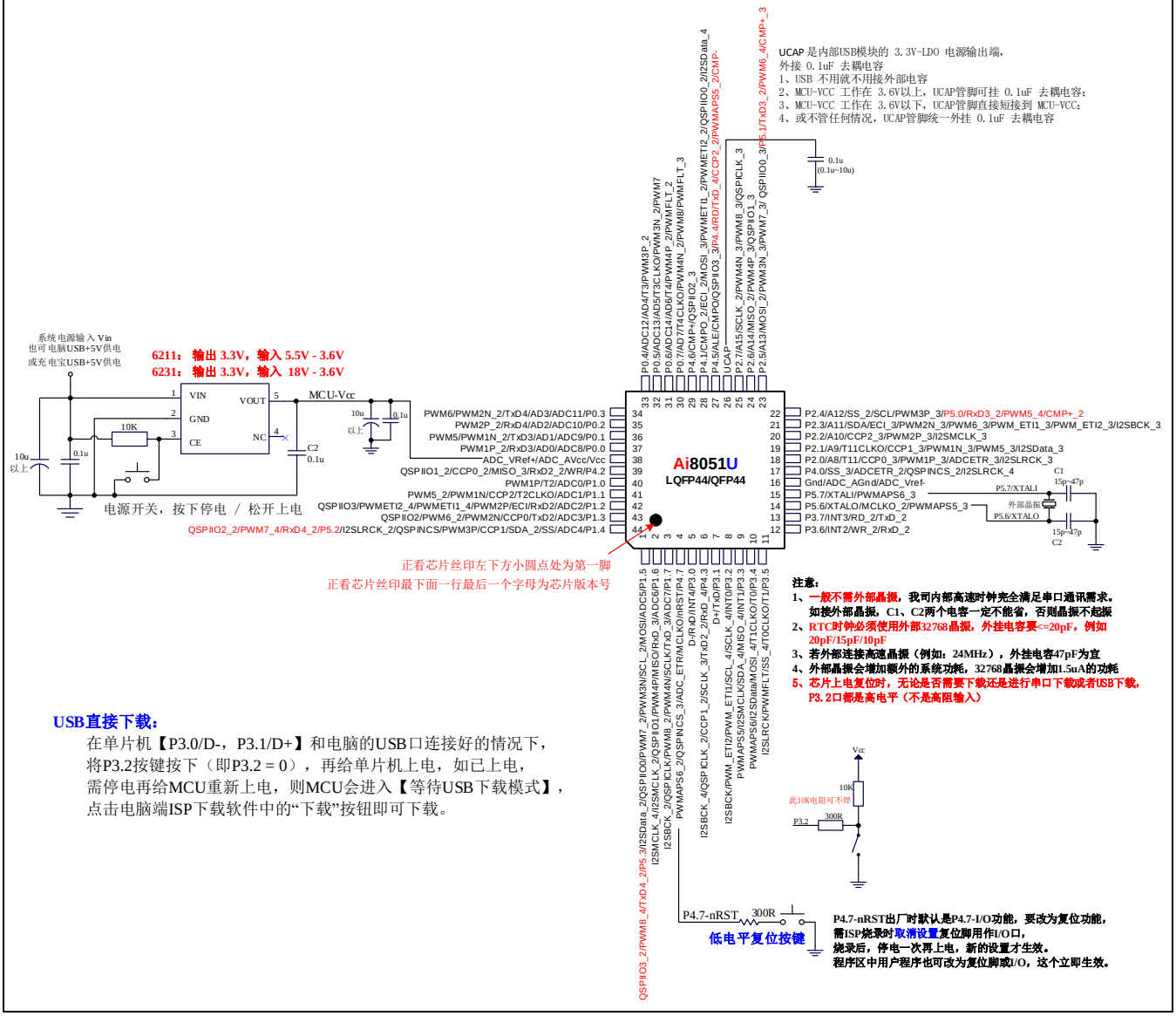
很多人经过多次插拔 USB，才能碰到 1 次【D+，D-】接触好的情况下，【Gnd，USB+5V】才开始接触好，才开始供电，才能成功进入 USB 下载。插 USB 插头代替供电，不能保证【Gnd，D+，D-，USB+5V】的接触顺序，所以，必须使用正常的电源开关，才能确保每次下载都能成功。

关于 I/O 的注意事项：

1、 P3.0 和 P3.1 口上电后的状态为弱上拉/准双向口模式
2、 除 P3.0 和 P3.1 外，其余所有 I/O 口上电后的状态均为高阻输入状态，用户在使用 I/O 口前必须先设置 I/O 口模式
3、 芯片上电时，若 P3.0 和 P3.1 同时为低电平，P3.2 口会短时间开启内部 4K 上拉，用以读取 P3.2 口外部状态来判断是否需要进入 USB 下载模式
4、 芯片上电时如果不需要使用 USB 进行 ISP 下载，P3.0/P3.1/P3.2 这 3 个 I/O 口不能同时为低电平，否则会进入 USB 下载模式而无法运行用户代码
5、 当使用 P4.7 当作复位脚时，这个端口内部的 4K 上拉电阻会一直打开；但 P4.7 做普通 I/O 口时，基于这个 I/O 口与复位脚共享管脚的特殊考量，端口内部的 4K 上拉电阻依然会打开大约 6.5 毫秒时间，再自动关闭（当用户的电路设计需要使用 P4.7 口驱动外部电路时，请务必考虑上电瞬间会有 6.5 毫秒时间的高电平的问题）

4.2.4　如果使用 3.3V-LDO 供电，建议用有电源关断功能的 6211/6231

3.3V-LDO 供电，使用有电源关断功能的 6211/6231供电 示意图



4.2.5 管脚说明

编号			名称	类型	说明
LQFP48/QFN48	LQFP44	PDIP40			
1	1	6	P5.3	I/O	标准 I/O 口
			TxD4_2	O	串口 4 的发送脚
			PWM8_4	I/O	PWM8 的捕获输入和脉冲输出
			QSPIIO3_2	I/O	QSPI 数据脚
2	1	6	P1.5	I/O	标准 I/O 口
			ADC5	I	ADC 模拟输入通道 5
			MOSI	I/O	SPI 主机输出从机输入
			SCL_2	I/O	I2C 的时钟线
			PWM3N	O	PWM3 的脉冲输出负极
			PWM7_2	I/O	PWM7 的捕获输入和脉冲输出
			QSPIIO0	I/O	QSPI 数据脚
			I2SSD_2	I/O	I2S 数据脚
3	2	7	P1.6	I/O	标准 I/O 口
			ADC6	I	ADC 模拟输入通道 6
			RxD_3	I	串口 1 的接收脚
			MISO	I/O	SPI 主机输入从机输出
			PWM4P	I/O	PWM4 的捕获输入和脉冲输出正极
			QSPIIO1	I/O	QSPI 数据脚
			I2SMCLK_2	I/O	I2S 主时钟脚
			I2SMCLK_4	I/O	I2S 主时钟脚
4	3	8	P1.7	I/O	标准 I/O 口
			ADC7	I	ADC 模拟输入通道 7
			TxD_3	O	串口 3 的发送脚
			SCLK	I/O	SPI 的时钟脚
			PWM4N	O	PWM4 的脉冲输出负极
			PWM8_2	I/O	PWM8 的捕获输入和脉冲输出
			QSPICLK	O	QSPI 时钟脚
			I2SCK_2	I/O	I2S 位时钟脚
5	4	9	P4.7	I/O	标准 I/O 口
			nRST	I	复位引脚（低电平复位）
			MCLKO	O	主时钟分频输出
			ADCETR	I	ADC 外部触发脚
			QSPINCS_3	O	QSPI 片选脚
			PWMAPS6_2	I/O	PWM 硬件移相端口
6	5	10	P3.0	I/O	标准 I/O 口
			INT4	I	外部中断 4 输入脚
			RxD	I	串口 1 的接收脚
			D-	I/O	USB 差分 D-脚

编号			名称	类型	说明
LQFP48/QFN48	LQFP44	PDIP40			
7	6	11	P4.3	I/O	标准 I/O 口
			RxD_4	I	串口 1 的接收脚
			TxD2_2	O	串口 2 的发送脚
			SCLK_3	I/O	SPI 的时钟脚
			CCP1_2	I/O	CCP1 的捕获输入和脉冲输出
			QSPICLK_2	O	QSPI 时钟脚
			I2SCK_4	I/O	I2S 位时钟脚
8	7	11	P3.1	I/O	标准 I/O 口
			TxD	O	串口 1 的发送脚
			D+	I/O	USB 差分 D+脚
9	8	12	P3.2	I/O	标准 I/O 口
			INT0	I	外部中断 0 输入脚
			SCLK_4	I/O	SPI 的时钟脚
			SCL_4	I/O	I2C 的时钟线
			PWMETI1	I	PWMA 外部触发输入脚
			PWMETI2	I	PWMB 外部触发输入脚
			I2SCK	I/O	I2S 位时钟脚
10	9	13	P3.3	I/O	标准 I/O 口
			INT1	I	外部中断 1 输入脚
			MISO_4	I/O	SPI 主机输入从机输出
			SDA_4	I/O	I2C 数据脚
			I2SMCLK	I/O	I2S 主时钟脚
			PWMAPS5	I/O	PWM 硬件移相端口
11	10	14	P3.4	I/O	标准 I/O 口
			T0	I	定时器 0 外部时钟输入
			T1CLKO	O	定时器 1 时钟分频输出
			MOSI_4	I/O	SPI 主机输出从机输入
			I2SSD	I/O	I2S 数据脚
			PWMAPS6	I/O	PWM 硬件移相端口
12	11	15	P3.5	I/O	标准 I/O 口
			T1	I	定时器 1 外部时钟输入
			T0CLKO	O	定时器 0 时钟分频输出
			SS_4	I	SPI 的从机选择脚（主机为输出）
			PWMFLT	I	PWMA/PWMB 的外部异常检测脚
			I2SWS	I/O	I2S 左右时钟脚
13	12	16	P3.6	I/O	标准 I/O 口
			INT2	I	外部中断 2 输入脚
			WR_2	O	外部总线写信号输出脚
			RxD_2	I	串口 1 的接收脚

编号			名称	类型	说明
LQFP48/QFN48	LQFP44	PDIP40			
14	13	17	P3.7	I/O	标准 I/O 口
			INT3	I	外部中断 3 输入脚
			RD_2	O	外部总线读信号输出脚
			TxD_2	O	串口 1 的发送脚
15	14	18	P5.6	I/O	标准 I/O 口
			XTALO	O	外部晶振型号输出脚
			MCLKO_2	O	主时钟分频输出
			PWMAPS5_3	I/O	PWM 硬件移相端口
16	15	19	P5.7	I/O	标准 I/O 口
			XTALI	I	外部晶振型号输入脚
			PWMAPS6_3	I/O	PWM 硬件移相端口
17	16	20	Gnd	Gnd	地线
			ADC_AGnd	Gnd	ADC 地线
			ADC_VRef-	Ref	ADC 外部参考电压源输入脚
18	17	21	P4.0	I/O	标准 I/O 口
			SS_3	I	SPI 的从机选择脚（主机为输出）
			ADCETR_2	I	ADC 外部触发脚
			QSPINCS_2	O	QSPI 片选脚
			I2SWS_4	I/O	I2S 左右时钟脚
19	18	21	P2.0	I/O	标准 I/O 口
			A8	O	地址总线
			T11	I	定时器 11 外部时钟输入
			CCP0_3	I/O	CCP0 的捕获输入和脉冲输出
			PWM1P_3	I/O	PWM1 的捕获输入和脉冲输出正极
			ADCETR_3	I	ADC 外部触发脚
			I2SWS_3	I/O	I2S 左右时钟脚
20	19	22	P2.1	I/O	标准 I/O 口
			A9	O	地址总线
			T11CLKO	O	定时器 11 时钟分频输出
			CCP1_3	I/O	CCP1 的捕获输入和脉冲输出
			PWM1N_3	O	PWM1 的脉冲输出负极
			PWM5_3	I/O	PWM5 的捕获输入和脉冲输出
			I2SSD_3	I/O	I2S 数据脚
21	20	23	P2.2	I/O	标准 I/O 口
			A10	O	地址总线
			CCP2_3	I/O	CCP2 的捕获输入和脉冲输出
			PWM2P_3	I/O	PWM2 的捕获输入和脉冲输出正极
			I2SMCLK_3	I/O	I2S 主时钟脚

编号			名称	类型	说明
LQFP48/QFN48	LQFP44	PDIP40			
22	21	24	P2.3	I/O	标准 I/O 口
			A11	O	地址总线
			SDA	I/O	I2C 数据脚
			ECI_3	I	CCP 模块的外部时钟脚
			PWM2N_3	O	PWM2 的脉冲输出负极
			PWM6_3	I/O	PWM6 的捕获输入和脉冲输出
			PWMETI_3	I	PWMA 外部触发输入脚
			PWMETI2_3	I	PWMB 外部触发输入脚
			I2SCK_3	I/O	I2S 位时钟脚
23	22	25	P2.4	I/O	标准 I/O 口
			A12	O	地址总线
			SS_2	I	SPI 的从机选择脚（主机为输出）
			SCL	I/O	I2C 的时钟线
			PWM3P_3	I/O	PWM3 的捕获输入和脉冲输出正极
24	22	25	P5.0	I/O	标准 I/O 口
			RxD3_2	I	串口 3 的接收脚
			PWM5_4	I/O	PWM5 的捕获输入和脉冲输出
			CMP+_2	I	比较器正极输入
25	23	26	P5.1	I/O	标准 I/O 口
			TxD3_2	O	串口 3 的发送脚
			PWM6_4	I/O	PWM6 的捕获输入和脉冲输出
			CMP+_3	I	比较器正极输入
26	23	26	P2.5	I/O	标准 I/O 口
			A13	O	地址总线
			MOSI_2	I/O	SPI 主机输出从机输入
			PWM3N_3	O	PWM3 的脉冲输出负极
			PWM7_3	I/O	PWM7 的捕获输入和脉冲输出
			QSPIIO0_3	I/O	QSPI 数据脚
27	24	27	P2.6	I/O	标准 I/O 口
			A14	O	地址总线
			MISO_2	I/O	SPI 主机输入从机输出
			PWM4P_3	I/O	PWM4 的捕获输入和脉冲输出正极
			QSPIIO1_3	I/O	QSPI 数据脚
28	25	28	P2.7	I/O	标准 I/O 口
			A15	O	地址总线
			SCLK_2	I/O	SPI 的时钟脚
			PWM4N_3	O	PWM4 的脉冲输出负极
			PWM8_3	I/O	PWM8 的捕获输入和脉冲输出
			QSPICLK_3	O	QSPI 时钟脚
29	26	29	UCAP	CAP	USB LDO 外接电容较

编号			名称	类型	说明
LQFP48/QFN48	LQFP44	PDIP40			
30	27	30	P4.5	I/O	标准 I/O 口
			ALE	O	地址锁存信号
			CMPO	O	比较器结果输出脚
			QSPIIO3_3	I/O	QSPI 数据脚
30	27	30	P4.4	I/O	标准 I/O 口
			RD	O	外部总线读信号输出脚
			TxD_4	O	串口 1 的发送脚
			CCP2_2	I/O	CCP2 的捕获输入和脉冲输出
			PWMAPS5	I/O	PWM 硬件移相端口
			CMP-	I	比较器负极输入
31	28	31	P4.1	I/O	标准 I/O 口
			CMPO_2	O	比较器结果输出脚
			ECI_2	I	CCP 模块的外部时钟脚
			MOSI_3	I/O	SPI 主机输出从机输入
			PWMETI_2	I	PWMA 外部触发输入脚
			PWMETI2_2	I	PWMB 外部触发输入脚
			QSPIIO0_2	I/O	QSPI 数据脚
			I2SSD_4	I/O	I2S 数据脚
32	29	31	P4.6	I/O	标准 I/O 口
			CMP+	I	比较器正极输入
			QSPIIO2_3	I/O	QSPI 数据脚
33	30	32	P0.7	I/O	标准 I/O 口
			AD7	I/O	地址/数据总线
			T4CLKO	O	定时器 4 时钟分频输出
			PWM4N_2	O	PWM4 的脉冲输出负极
			PWM8	I/O	PWM8 的捕获输入和脉冲输出
			PWMFLT_3	I	PWMA/PWMB 的外部异常检测脚
34	31	33	P0.6	I/O	标准 I/O 口
			ADC14	I	ADC 模拟输入通道 14
			AD6	I/O	地址/数据总线
			T4	I	定时器 4 外部时钟输入
			PWM4P_2	I/O	PWM4 的捕获输入和脉冲输出正极
			PWMFLT_2	I	PWMA/PWMB 的外部异常检测脚
35	32	34	P0.5	I/O	标准 I/O 口
			ADC13	I	ADC 模拟输入通道 13
			AD5	I/O	地址/数据总线
			T3CLKO	O	定时器 3 时钟分频输出
			PWM3N_2	O	PWM3 的脉冲输出负极
			PWM7	I/O	PWM7 的捕获输入和脉冲输出

编号			名称	类型	说明
LQFP48/QFN48	LQFP44	PDIP40			
36	33	35	P0.4	I/O	标准 I/O 口
			ADC12	I	ADC 模拟输入通道 12
			AD4	I/O	地址/数据总线
			T3	I	定时器 3 外部时钟输入
			PWM3P_2	I/O	PWM3 的捕获输入和脉冲输出正极
37	34	36	P0.3	I/O	标准 I/O 口
			ADC11	I	ADC 模拟输入通道 11
			AD3	I/O	地址/数据总线
			TxD4	O	串口 4 的发送脚
			PWM2N_2	O	PWM2 的脉冲输出负极
			PWM6	I/O	PWM6 的捕获输入和脉冲输出
38	35	37	P0.2	I/O	标准 I/O 口
			ADC10	I	ADC 模拟输入通道 10
			AD2	I/O	地址/数据总线
			RxD4	I	串口 4 的接收脚
			PWM2P_2	I/O	PWM2 的捕获输入和脉冲输出正极
39	36	38	P0.1	I/O	标准 I/O 口
			ADC9	I	ADC 模拟输入通道 9
			AD1	I/O	地址/数据总线
			TxD3	O	串口 3 的发送脚
			PWM1N_2	O	PWM1 的脉冲输出负极
			PWM5	I/O	PWM5 的捕获输入和脉冲输出
40	37	39	P0.0	I/O	标准 I/O 口
			ADC8	I	ADC 模拟输入通道 8
			AD0	I/O	地址/数据总线
			RxD3	I	串口 3 的接收脚
			PWM1P_2	I/O	PWM1 的捕获输入和脉冲输出正极
41	38	40	Vcc	Vcc	电源脚
			ADC_AVcc	Vcc	ADC 电源脚
			ADC_VRef+	Ref	ADC 外部参考电压源输入脚
42	39	1	P4.2	I/O	标准 I/O 口
			WR	O	外部总线写信号输出脚
			RxD2_2	I	串口 2 的接收脚
			MISO_3	I/O	SPI 主机输入从机输出
			CCP0_2	I/O	CCP0 的捕获输入和脉冲输出
			QSPIIO1_2	I/O	QSPI 数据脚
43	40	1	P1.0	I/O	标准 I/O 口
			ADC0	I	ADC 模拟输入通道 0
			T2	I	定时器 2 外部时钟输入
			PWM1P	I/O	PWM1 的捕获输入和脉冲输出正极

编号			名称	类型	说明
LQFP48/QFN48	LQFP44	PDIP40			
44	41	2	P1.1	I/O	标准 I/O 口
			ADC1	I	ADC 模拟输入通道 1
			T2CLKO	O	定时器 2 时钟分频输出
			CCP2	I/O	CCP2 的捕获输入和脉冲输出
			PWM1N	O	PWM1 的脉冲输出负极
			PWM5_2	I/O	PWM5 的捕获输入和脉冲输出
45	42	3	P1.2	I/O	标准 I/O 口
			ADC2	I	ADC 模拟输入通道 2
			RxD2	I	串口 2 的接收脚
			ECI	I	CCP 模块的外部时钟脚
			PWM2P	I/O	PWM2 的捕获输入和脉冲输出正极
			PWMETI1_4	I	PWMA 外部触发输入脚
			PWMWTI2_4	I	PWMB 外部触发输入脚
			QSPIIO3	I/O	QSPI 数据脚
46	43	4	P1.3	I/O	标准 I/O 口
			ADC3	I	ADC 模拟输入通道 3
			TxD2	O	串口 2 的发送脚
			CCP0	I/O	CCP0 的捕获输入和脉冲输出
			PWM2N	O	PWM2 的脉冲输出负极
			PWM6_2	I/O	PWM6 的捕获输入和脉冲输出
			QSPIIO2	I/O	QSPI 数据脚
47	44	5	P1.4	I/O	标准 I/O 口
			ADC4	I	ADC 模拟输入通道 4
			SS	I	SPI 的从机选择脚（主机为输出）
			SDA_2	I/O	I2C 数据脚
			CCP1	I/O	CCP1 的捕获输入和脉冲输出
			PWM3P	I/O	PWM3 的捕获输入和脉冲输出正极
			QSPINCS	O	QSPI 片选脚
			I2SWS_2	I/O	I2S 左右时钟脚
48	44	5	P5.2	I/O	标准 I/O 口
			RxD_2	I	串口 1 的接收脚
			PWM7_4	I/O	PWM7 的捕获输入和脉冲输出
			QSPIIO2_2	I/O	QSPI 数据脚

4.2.6 USB-Link1D 对 Ai8051U 自动停电/上电烧录，串口仿真+串口通讯

USB Link1D工具：支持全自动停电-上电在线下载 / 脱机下载 / 仿真

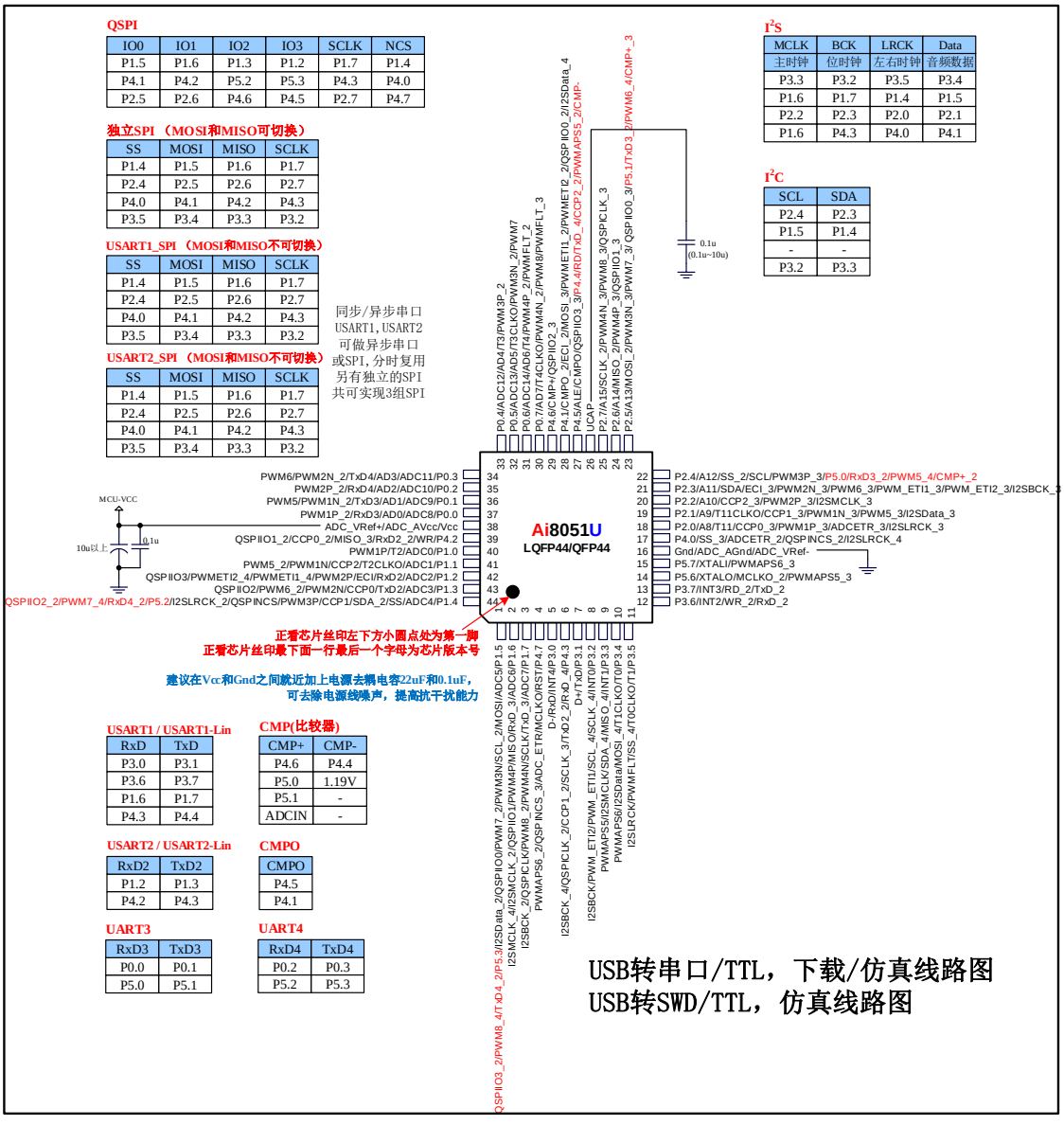
【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，下载编程进行中，数秒后提示下载编程成功，目标MCU会自动复位到用户程序区自动跑用户程序。部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P4.7/nRST变成复位脚



备注：上图中红色管脚部分是 2024 年 10 月 15 日之后生产的芯片才有。

ISP 下载步骤：

1、按照如图所示的连接方式将 USB-Link1D 和目标芯片连接

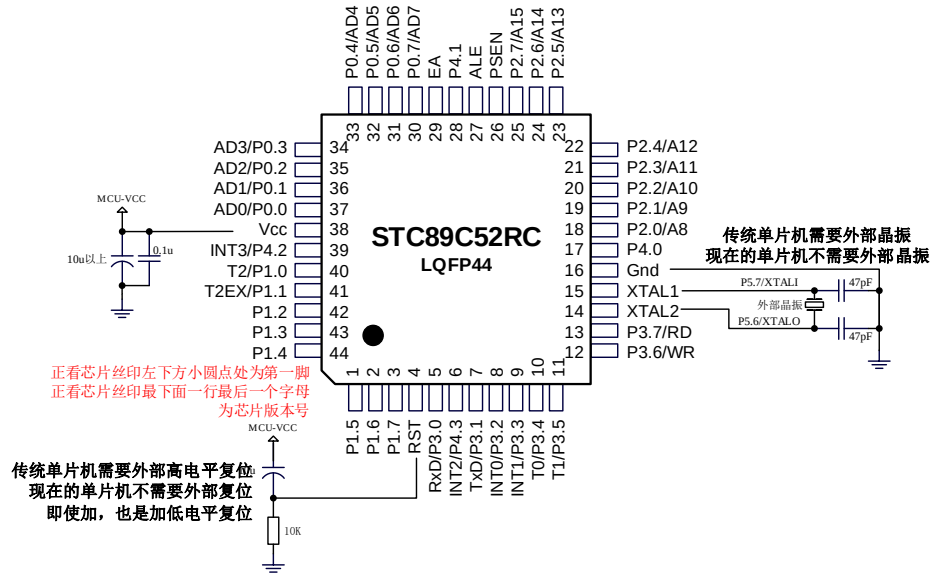
2、点击 ISP 下载软件中的“下载/编程”按钮

3、开始 ISP 下载

（注意：若是使用 USB-Link1D 给目标系统供电，目标系统的总电流不能大于 200mA，否则会导致下载失败。）

比较下传统的 89C52RC 系列相应下载线路图：

传统单片机需要外部晶振，现在的单片机不需要外部晶振
传统单片机需要外部高电平复位，现在的不需要，并且已改为低电平复位



USB Link1D工具：支持全自动停电-上电在线下载 / 脱机下载 / 仿真

【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

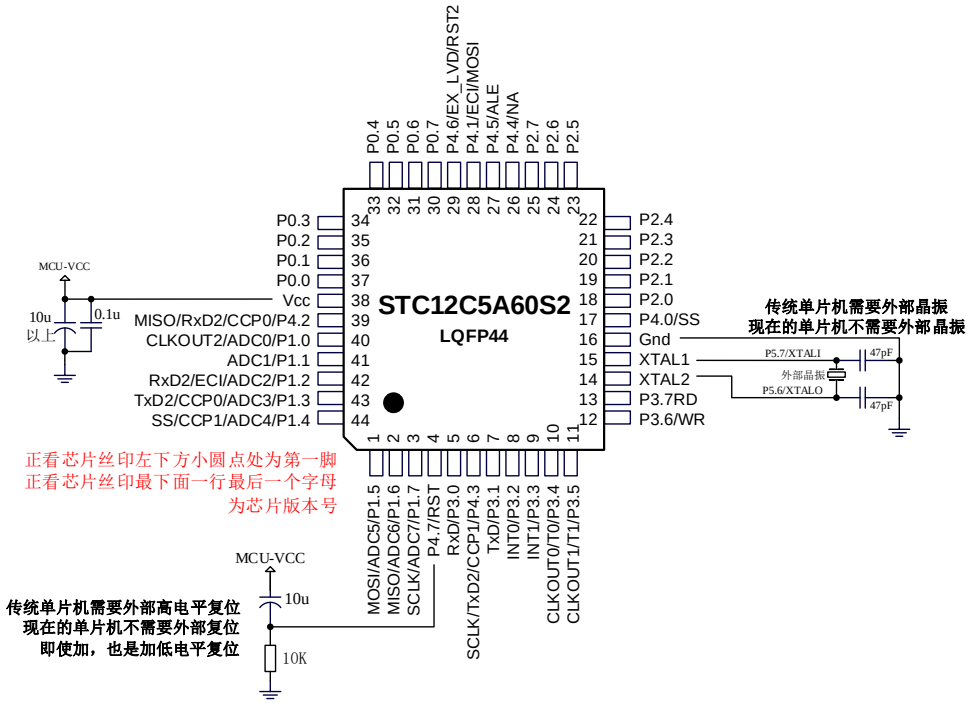
点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮
2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，下载编程进行中，数秒后提示下载编程成功，目标MCU会自动复位到用户程序区自动跑用户程序。
部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P4.7/nRST变成复位脚

比较下传统的 12C5A60S2 相应下载线路图：

传统单片机需要外部晶振，现在的单片机不需要外部晶振
传统单片机需要外部高电平复位，现在的不需要，并且已改为低电平复位



USB Link1D工具：支持全自动停电-上电在线下载 / 脱机下载 / 仿真

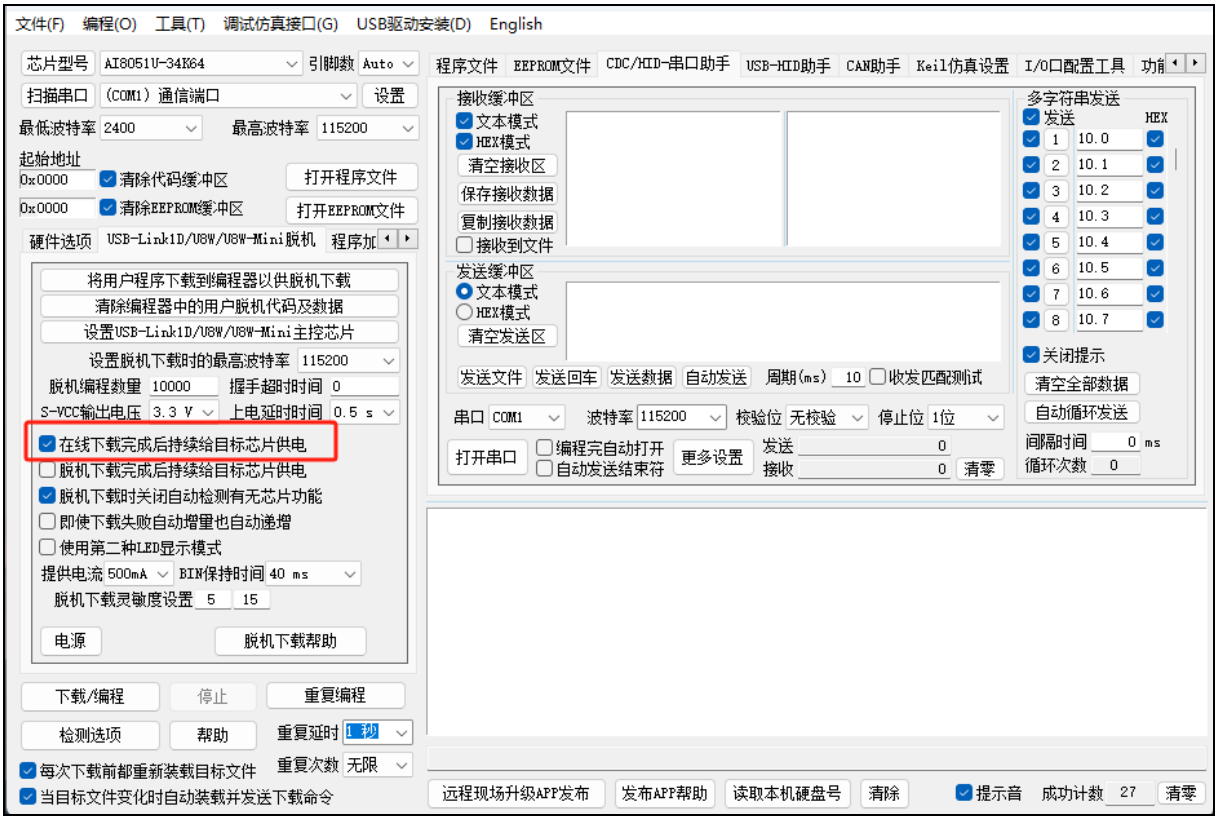
【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

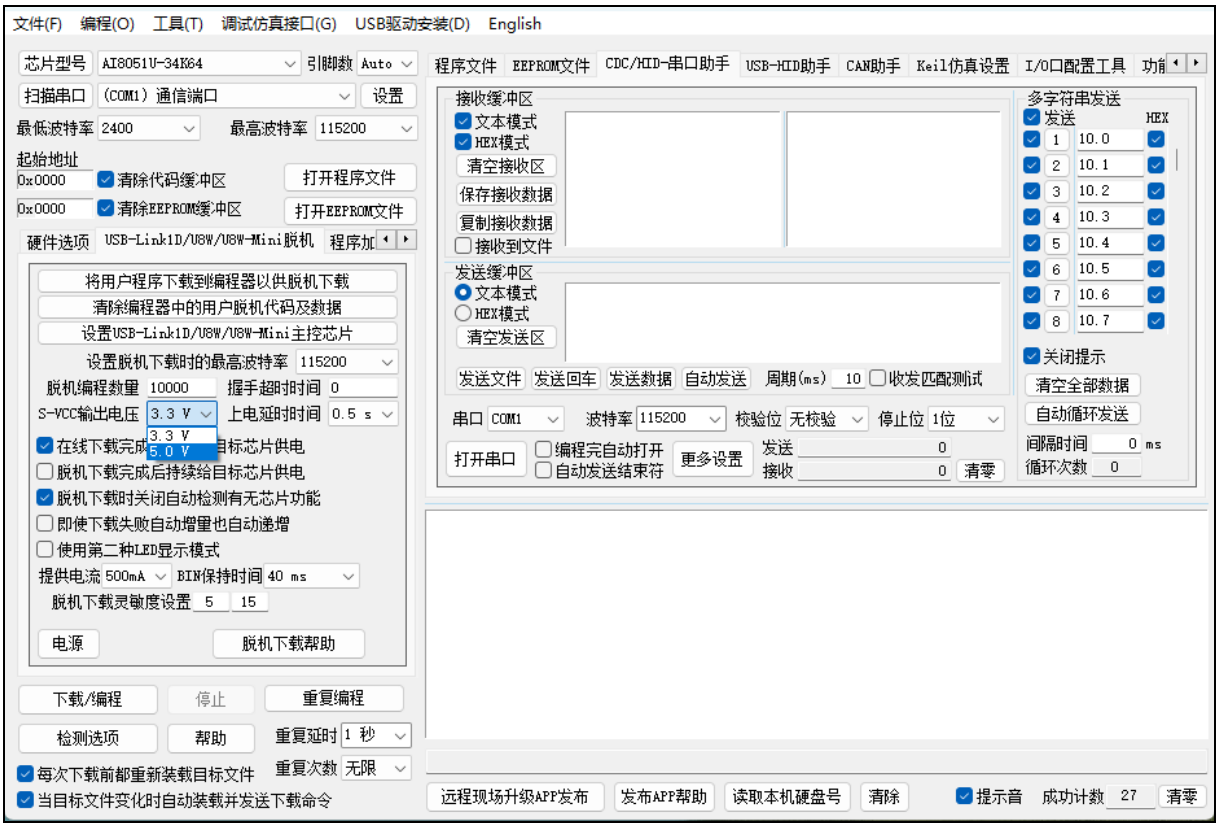
【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮
2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，下载编程进行中，数秒后提示下载编程成功，目标MCU会自动复位到用户程序区自动跑用户程序。
部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P4.7/nRST变成复位脚

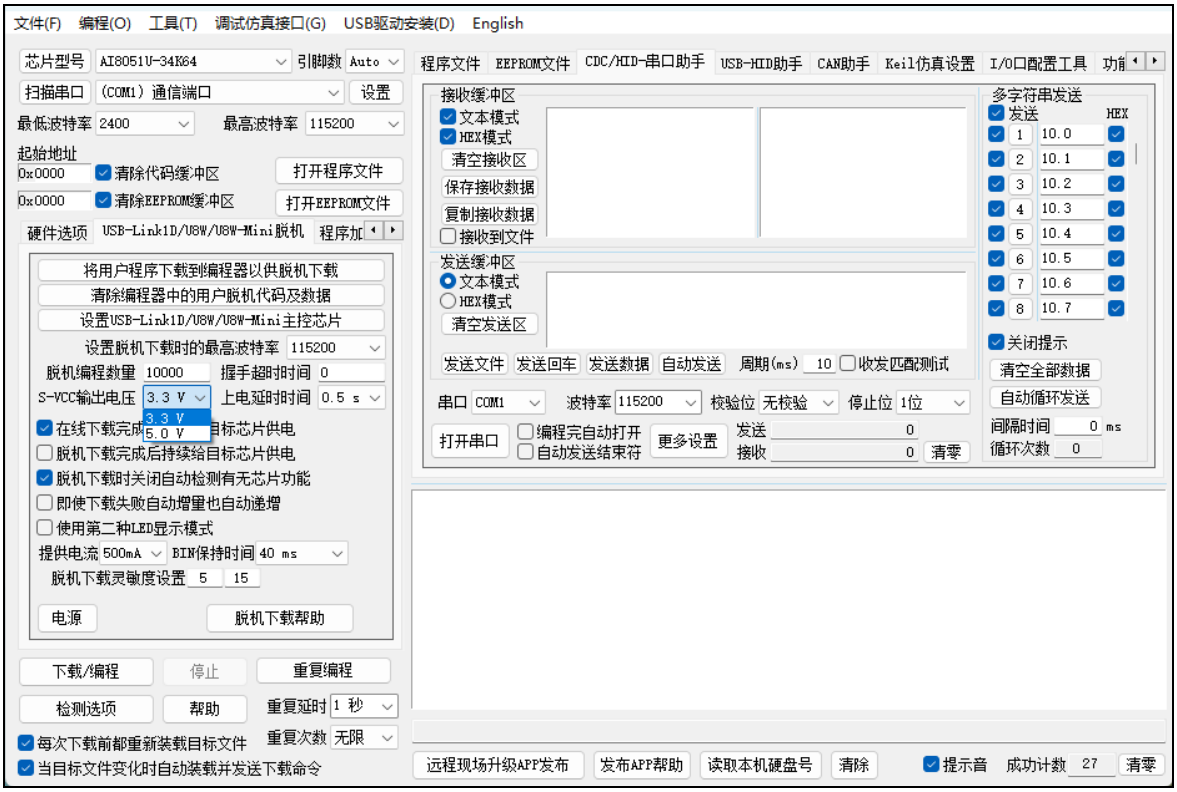
如何设置 USB-Link1D 下载完后持续给目标芯片供电



如何设置 USB-Link1D 输出 5V

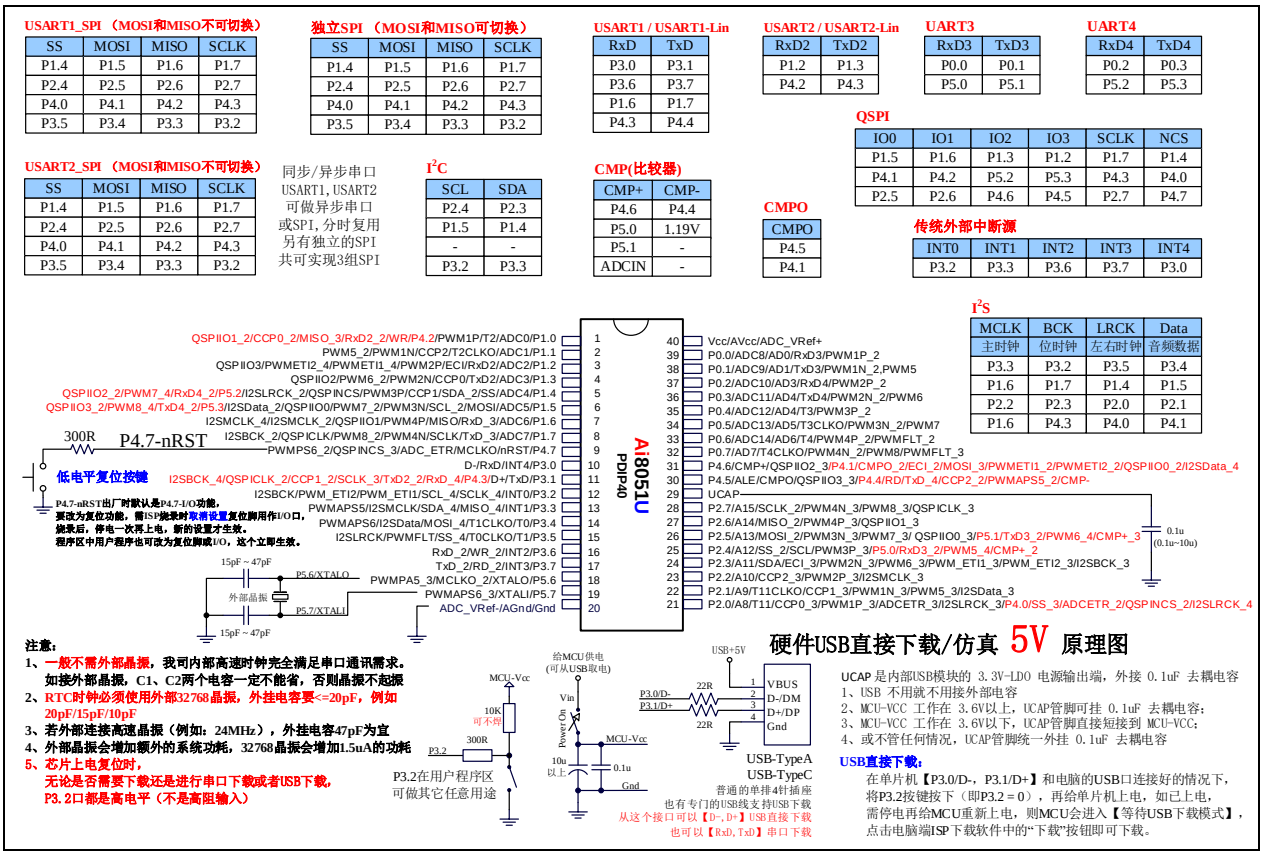


如何设置 USB-Link1D 输出 3.3V



4.3 PDIP40 管脚图，USB-ISP 下载，各种 烧录/仿真 线路图

4.3.1 PDIP40 管脚图，5V 线路，USB-ISP 下载/烧录/仿真



备注：上图中红色管脚部分是 2024 年 10 月 15 日之后生产的芯片才有。

现在带硬件 USB 的 MCU 支持用硬件 USB 下载，因为用的是 USB-HID 通信协议，不需要安装任何驱动。只要 USB 鼠标、USB 键盘能工作，USB-HID 驱动就是好的，不要安装 USB-HID 驱动，免驱。

在 D-/P3.0，D+/P3.1 与 PC-USB 端口连接好的状况下，USB-ISP 下载程序有如下三种模式：

【USB 下载方法一，P3.2 按键，再结合停电上电下载】

1、按下板子上的 P3.2/INT0 按键，就是 P3.2 接地

2、给目标芯片重新上电，不管之前是否已通电。

===电子开关是按下停电后再松开就是上电

等待电脑端 ISP 下载软件中自动识别出"(HID1) USB Writer"后，就与 P3.2 状态无关了，这时可以松开 P3.2 按键（P3.2 在用户程序区可做其它任意用途）

===传统的机械自锁紧开关是按上来停电，按下去是上电

3、点击电脑端下载软件中的【下载/编程】按钮（注意：USB 下载与串口下载的操作顺序不同）

下载进行中，几秒钟后，提示下载成功！

【USB 下载方法二，复位管脚低电平复位下载】

USB 连接好并已上电的情况下，外部按键复位也可进入 USB 下载模式，注意：

P4.7-nRST 出厂时默认是 P4.7-I/O 功能，要改为复位功能，需 ISP 烧录时取消设置复位脚用作 I/O 口，停电一次再上电才生效，程序区中用户程序也可改为复位脚或 I/O，这个立即生效。

1、按下 P4.7-nRST 外接的低电平复位按键复位 MCU，

松开复位键，MCU 从系统程序区启动，判断是否要下载用户程序，

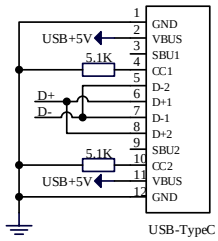
等待电脑端 ISP 下载软件中自动识别出"(HID1) USB Writer"后

2、点击电脑端下载软件中的【下载/编程】按钮

下载进行中，几秒钟后，提示下载成功！

备注：如果目标 MCU 的用户程序区已有 USB 通讯程序（含 CDC、HID 等），则通过复位脚复位进入 USB 下载模式不被支持，只能通过停电上电冷启动下载或者从用户程序区软复位到系统区下载

比较一下流行的 USB-TypeC接口：



【USB 下载方法三，从用户程序区软复位到系统区下载】

USB 连接好并已上电的情况下，从用户程序区软复位到系统区也可进入 USB 下载模式

1、在用户程序区运行软复位到系统区的程序，就是 IAP_CONTR 寄存器送 60H

等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后

2、点击电脑端下载软件中的【下载/编程】按钮

下载进行中，几秒钟后，提示下载成功！

USB 下载 注意事项：

拔插 USB 插头不能代替上面线路图中的电源开关。正常操作步骤如下：

USB 的【Gnd，D+，D-】接好的情况下，按下 P3.2 按键接地，再通过正常的电源开关给 MCU 供电或重新供电，让 MCU 冷启动进入系统程序区，判断是否需要等待电脑端 USB 下载程序。

拔插 USB 插头代替电源开关不能每次都能成功进行 USB 下载的原因：

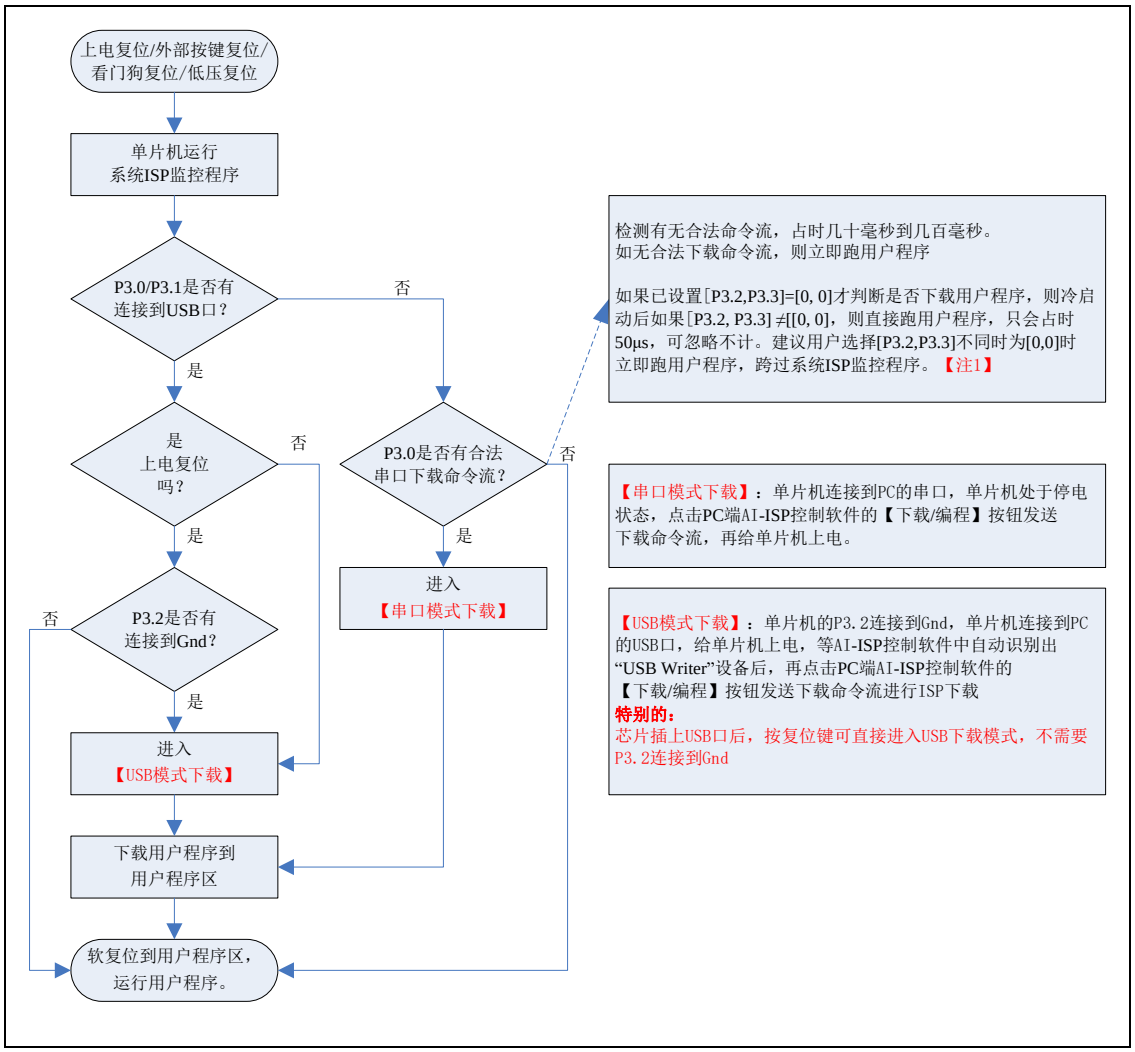
拔插 USB 插头，如【Gnd，USB+5V】已接触好，已供电，而【D+，D-】有一个甚至两个信号线还没有接触好，MCU 已上电，开始跑系统区程序时，发现 USB 还没接触好，则会从系统区软复位到用户程序区跑用户程序，不再进入等待 USB 下载模式，本次就无法顺利进行 USB 下载。

很多人经过多次插拔 USB，才能碰到 1 次【D+，D-】接触好的情况下，【Gnd，USB+5V】才开始接触好，才开始供电，才能成功进入 USB 下载。插 USB 插头代替供电，不能保证【Gnd，D+，D-，USB+5V】的接触顺序，所以，必须使用正常的电源开关，才能确保每次下载都能成功。

关于 I/O 的注意事项：

1、 P3.0 和 P3.1 口上电后的状态为弱上拉/准双向口模式
2、 除 P3.0 和 P3.1 外，其余所有 I/O 口上电后的状态均为高阻输入状态，用户在使用 I/O 口前必须先设置 I/O 口模式
3、 芯片上电时，若 P3.0 和 P3.1 同时为低电平，P3.2 口会短时间开启内部 4K 上拉，用以读取 P3.2 口外部状态来判断是否需要进入 USB 下载模式
4、 芯片上电时如果不需要使用 USB 进行 ISP 下载，P3.0/P3.1/P3.2 这 3 个 I/O 口不能同时为低电平，否则会进入 USB 下载模式而无法运行用户代码
5、 当使用 P4.7 当作复位脚时，这个端口内部的 4K 上拉电阻会一直打开；但 P4.7 做普通 I/O 口时，基于这个 I/O 口与复位脚共享管脚的特殊考量，端口内部的 4K 上拉电阻依然会打开大约 6.5 毫秒时间，再自动关闭（当用户的电路设计需要使用 P4.7 口驱动外部电路时，请务必考虑上电瞬间会有 6.5 毫秒时间的高电平的问题）

ISP 下载流程图（硬件/软件模拟 USB+串口模式）：



打开 AIapp-ISP-V6.94S 以上版本软件，如下图所示：

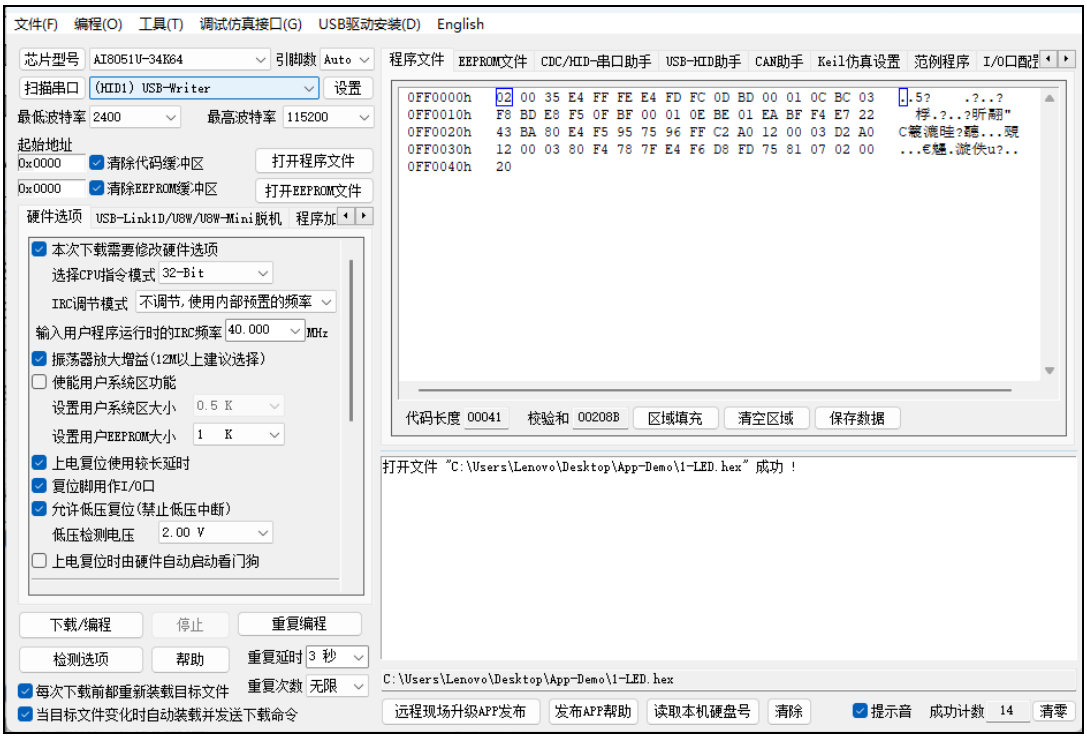
选择好对应的正确型号，打开要烧录的文件

将 USB 数据线，如前面的原理图连接好，

注意是【USB+5V，D-，D+，Gnd】USB 数据线，不是【USB+5V，NC，NC，Gnd】USB 电源线

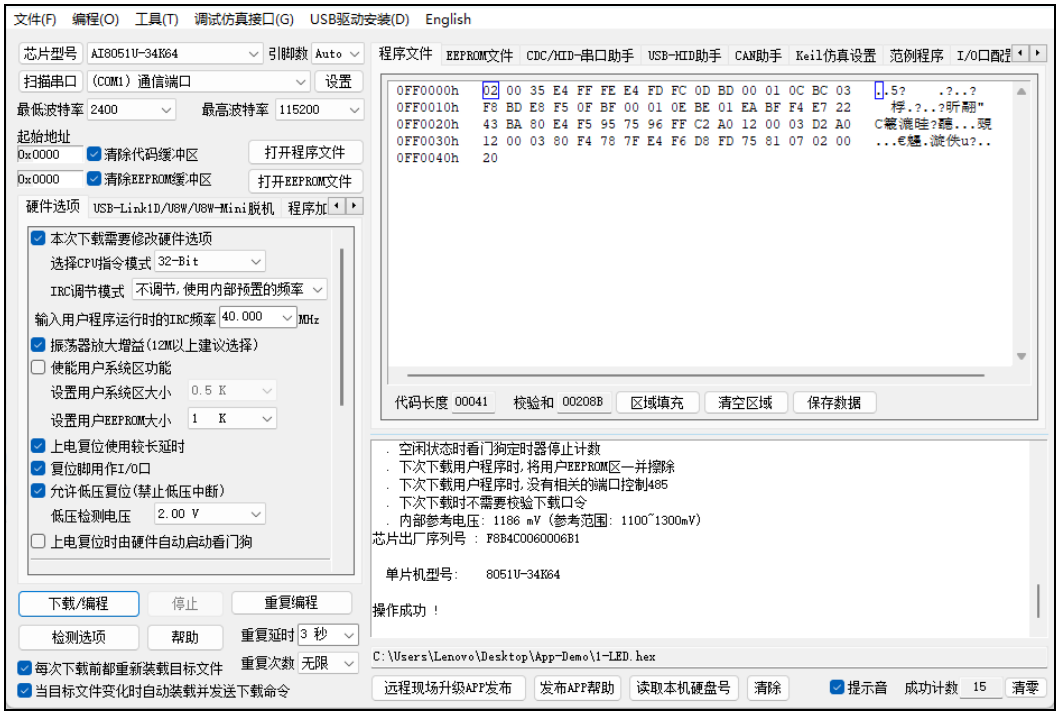
在 P3.2 接地按键按下的状态下，给 MCU 上电，或重新上电。

则 AIapp-ISP 软件出现如下显示：



点击“下载/编程”按钮，

则会如下图显示：正在下载用户代码，操作成功！



如上硬件 USB，ISP 下载/编程 烧录成功。

QSPI

IO0	IO1	IO2	IO3	SCLK	NCS
P1.5	P1.6	P1.3	P1.2	P1.7	P1.4
-	-	-	-	-	-
P2.5	P2.6	P4.6	P4.5	P2.7	P4.7

独立SPI （MOSI和MISO可切换）

SS	MOSI	MISO	SCLK
P1.4	P1.5	P1.6	P1.7
P2.4	P2.5	P2.6	P2.7
P4.0	P4.1	P4.2	P4.3
P3.5	P3.4	P3.3	P3.2

同步/异步串口
USART1,USART2
可做异步串口
或SPI,分时复用
另有独立的SPI
共可实现3组SPI

USART1_SPI （MOSI和MISO不可切换）

SS	MOSI	MISO	SCLK
P1.4	P1.5	P1.6	P1.7
P2.4	P2.5	P2.6	P2.7
P4.0	P4.1	P4.2	P4.3
P3.5	P3.4	P3.3	P3.2

USART2_SPI （MOSI和MISO不可切换）

SS	MOSI	MISO	SCLK
P1.4	P1.5	P1.6	P1.7
P2.4	P2.5	P2.6	P2.7
P4.0	P4.1	P4.2	P4.3
P3.5	P3.4	P3.3	P3.2

I²S

BCK	MCK	Data	LRCK
P3.2	P3.3	P3.4	P3.5
P1.7	P1.6	P1.5	P1.4
P2.3	P2.2	P2.1	P2.0
-	-	-	-

I²C

SCL	SDA
P2.4	P2.3
P1.5	P1.4
-	-
P3.2	P3.3

UART1

RxD	TxD
P3.0	P3.1
P3.6	P3.7
P1.6	P1.7
P4.3	P4.4

UART2

RxD2	TxD2
P1.2	P1.3
P4.2	P4.3

UART3

RxD3	TxD3
P0.0	P0.1
P5.0	P5.1

CMP(比较器)

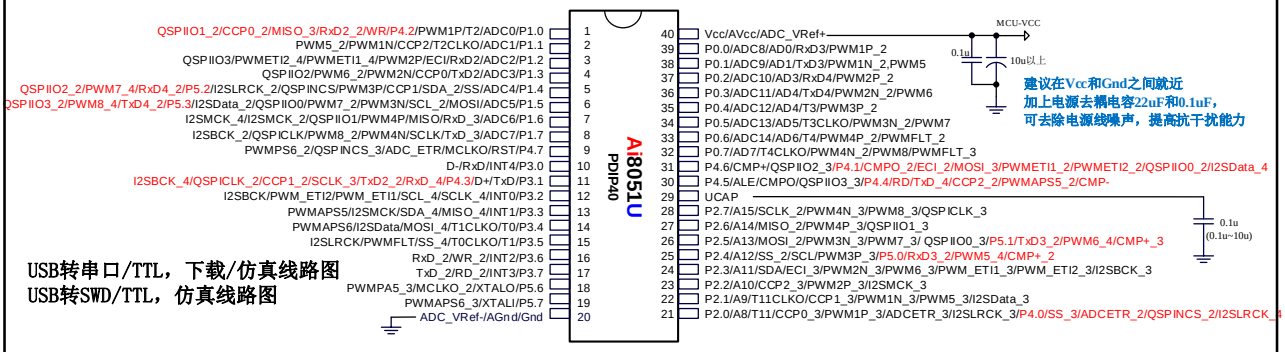
CMP+	CMP-
P4.6	P4.4
P5.0	1.19V
P5.1	-
ADCIN	-

CMPO

CMPO
P4.5
P4.1

UART4

RxD4	TxD4
P0.2	P0.3
P5.2	P5.3



USB转串口/TTL，下载/仿真线路图
USB转SWD/TTL，仿真线路图

USB Link1D工具：支持全自动停电-上电在线下载 / 脱机下载 / 仿真

【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，
数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】

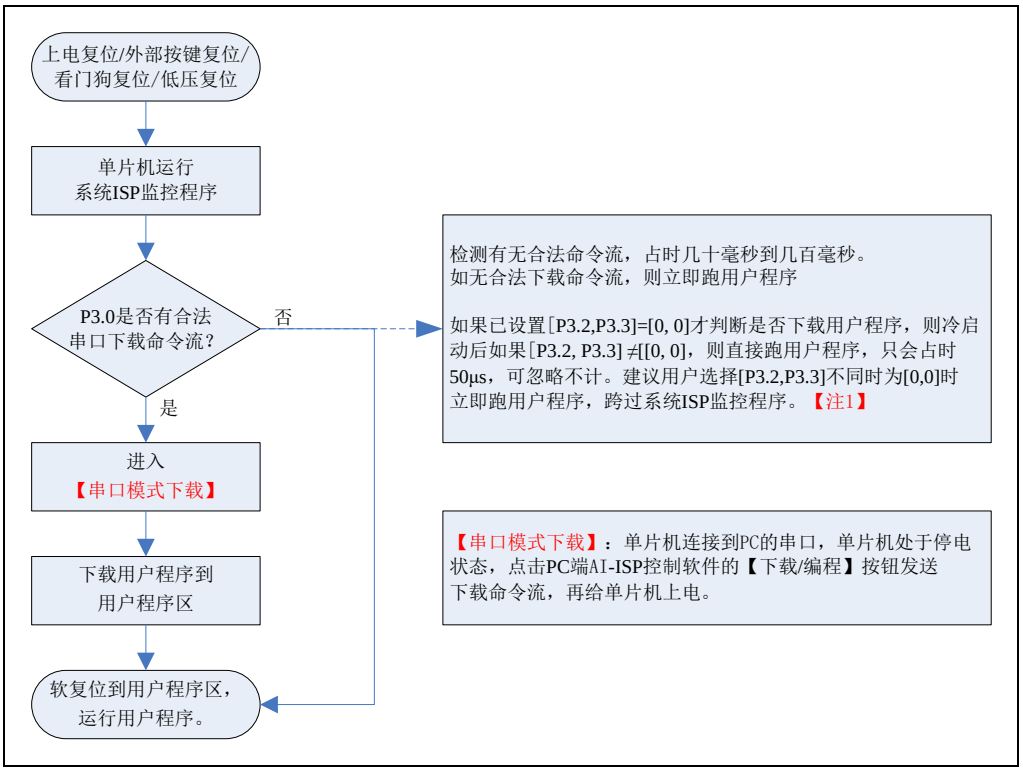
1、点击 电脑端 ISP 软件的【下载/编程】按钮
2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，
下载编程进行中，数秒后提示下载编程成功，目标MCU会自动复位到用户程序区自动跑用户程序。
部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P4.7/nRST变成复位脚

备注：上图中红色管脚部分是 2024 年 10 月 15 日之后生产的芯片才有。

USB 转串口/TTL，全自动 ISP 下载步骤：

1、按照如图所示的连接方式将 USB-Link1D 和目标芯片连接
2、点击 ISP 下载软件中的“下载/编程”按钮
3、开始 ISP 下载
（注意：若是使用 USB-Link1D 给目标系统供电，目标系统的总电流不能大于200mA，否则会导致下载失败。）

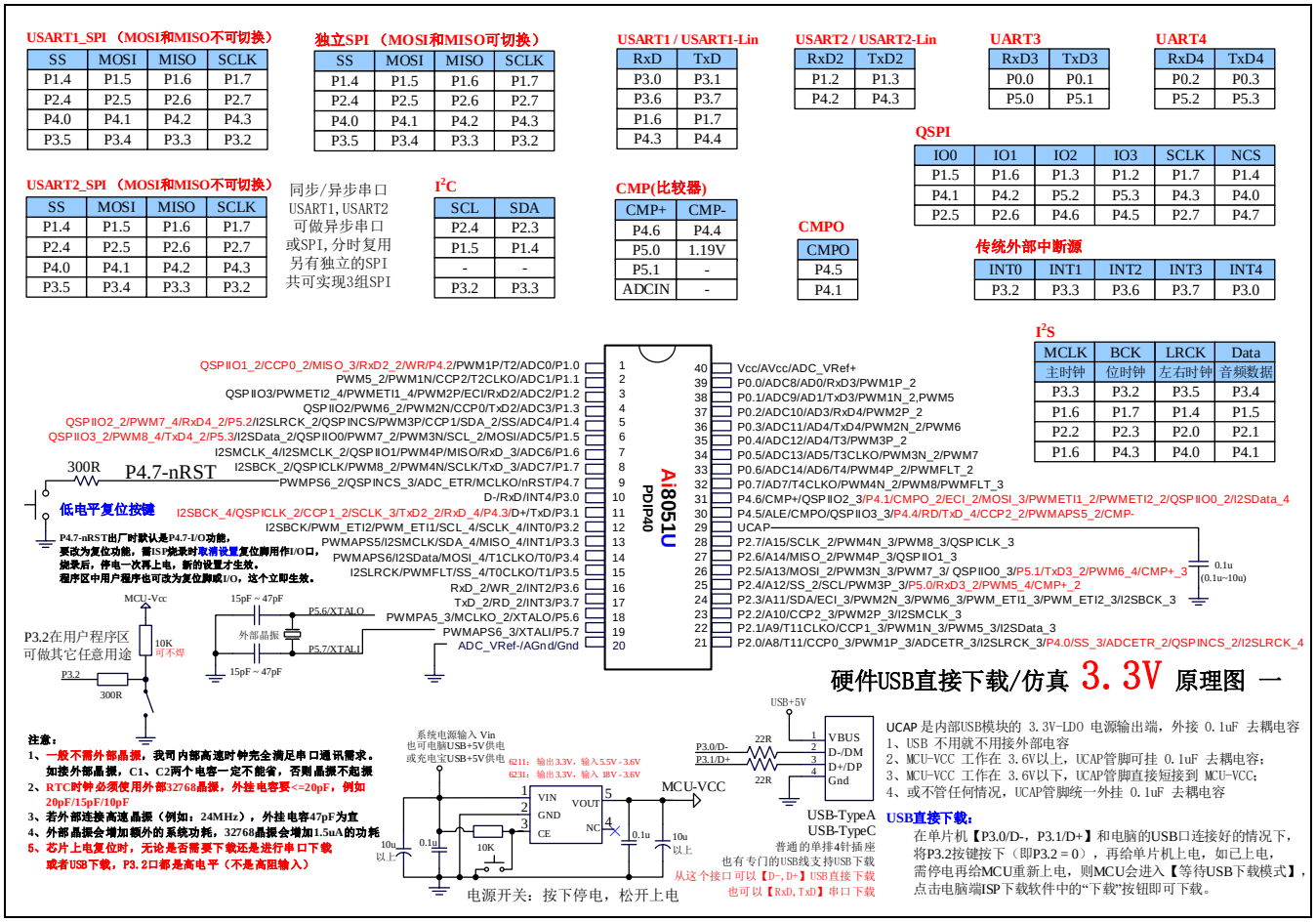
ISP 下载流程图（串口下载模式）



将【MCU-VCC，P3.0，P3.1，Gnd】，如前面的原理图连接到 USB-Link1D 工具;
将 USB-Link1D 全自动烧录工具，通过 USB 数据线连接到电脑，【USB+5V，D-，D+，Gnd】
打开 AIapp-ISP-V6.94S 以上版本软件
则 AIapp-ISP 软件显示如下：

选择好对应的正确型号，
打开要烧录的文件
点击“下载/编程”按钮，全自动烧录
则 AIapp-ISP 软件出现如下显示：

4.3.2　PDIP40 管脚图，3.3V 线路一，USB-ISP 下载/仿真



备注：上图中红色管脚部分是 2024 年 10 月 15 日之后生产的芯片才有。

现在带硬件 USB 的 MCU 支持用硬件 USB 下载，因为用的是 USB-HID 通信协议，不需要安装任何驱动。只要 USB 鼠标、USB 键盘能工作，USB-HID 驱动就是好的，不要安装 USB-HID 驱动，免驱。

在 D-/P3.0，D+/P3.1 与 PC-USB 端口连接好的状况下，USB-ISP 下载程序有如下三种模式：

【USB 下载方法一，P3.2 按键，再结合停电上电下载】

1、按下板子上的 P3.2/INT0 按键，就是 P3.2 接地

2、给目标芯片重新上电，不管之前是否已通电。

===电子开关是按下停电后再松开就是上电

等待电脑端 ISP 下载软件中自动识别出"(HID1) USB Writer"后，就与 P3.2 状态无关了，这时可以松开 P3.2 按键（P3.2 在用户程序区可做其它任意用途）

===传统的机械自锁紧开关是按上来停电，按下去是上电

3、点击电脑端下载软件中的【下载/编程】按钮（注意：USB 下载与串口下载的操作顺序不同）

下载进行中，几秒钟后，提示下载成功！

【USB 下载方法二，复位管脚低电平复位下载】

USB 连接好并已上电的情况下，外部按键复位也可进入 USB 下载模式，注意：

P4.7-nRST 出厂时默认是 P4.7-I/O 功能，要改为复位功能，需 ISP 烧录时取消设置复位脚用作 I/O 口，停电一次再上电才生效，程序区中用户程序也可改为复位脚或 I/O，这个立即生效。

1、按下 P4.7-nRST 外接的低电平复位按键复位 MCU，

松开复位键，MCU 从系统程序区启动，判断是否要下载用户程序，
等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后

2、点击电脑端下载软件中的【下载/编程】按钮
下载进行中，几秒钟后，提示下载成功！

备注：如果目标 MCU 的用户程序区已有 USB 通讯程序（含 CDC、HID 等），则通过复位脚复位进入 USB 下载模式不被支持，只能通过停电上电冷启动下载或者从用户程序区软复位到系统区下载

【USB 下载方法三，从用户程序区软复位到系统区下载】

USB 连接好并已上电的情况下，从用户程序区软复位到系统区也可进入 USB 下载模式

1、在用户程序区运行软复位到系统区的程序，就是 IAP_CONTR 寄存器送 60H
等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后

2、点击电脑端下载软件中的【下载/编程】按钮
下载进行中，几秒钟后，提示下载成功！

USB 下载 注意事项：

拔插 USB 插头不能代替上面线路图中的电源开关。正常操作步骤如下：

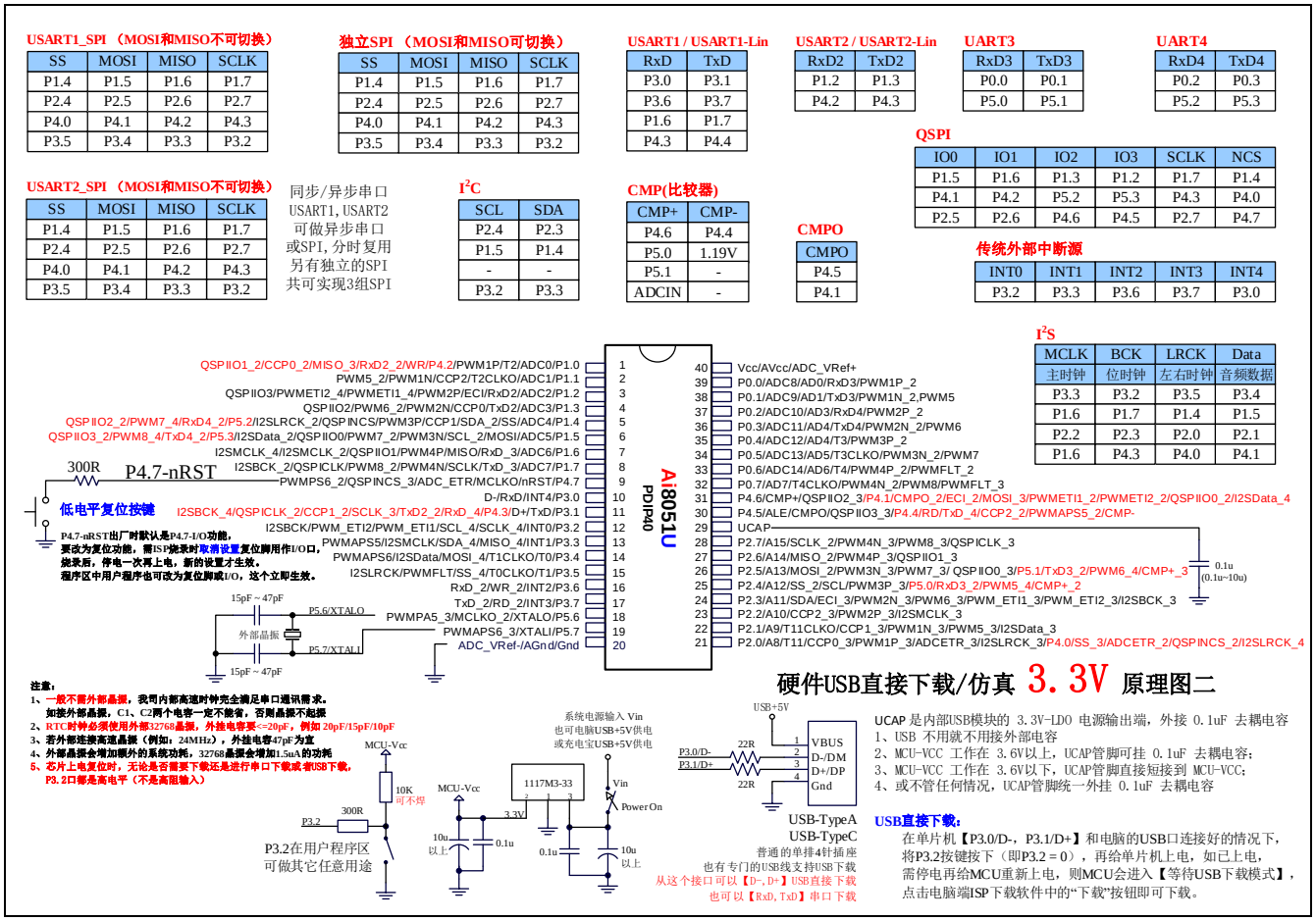
USB 的【Gnd，D+，D-】接好的情况下，按下 P3.2 按键接地，再通过正常的电源开关给 MCU 供电或重新供电，让 MCU 冷启动进入系统程序区，判断是否需要等待电脑端 USB 下载程序。

拔插 USB 插头代替电源开关不能每次都能成功进行 USB 下载的原因：

拔插 USB 插头，如【Gnd，USB+5V】已接触好，已供电，而【D+，D-】有一个甚至两个信号线还没有接触好，MCU 已上电，开始跑系统区程序时，发现 USB 还没接触好，则会从系统区软复位到用户程序区跑用户程序，不再进入等待 USB 下载模式，本次就无法顺利进行 USB 下载。

很多人经过多次插拔 USB，才能碰到 1 次【D+，D-】接触好的情况下，【Gnd，USB+5V】才开始接触好，才开始供电，才能成功进入 USB 下载。插 USB 插头代替供电，不能保证【Gnd，D+，D-，USB+5V】的接触顺序，所以，必须使用正常的电源开关，才能确保每次下载都能成功。

4.3.3　PDIP40 管脚图，3.3V 线路二，USB-ISP 下载/仿真



备注：上图中红色管脚部分是 2024 年 10 月 15 日之后生产的芯片才有。

现在带硬件 USB 的 MCU 支持用硬件 USB 下载，因为用的是 USB-HID 通信协议，不需要安装任何驱动。只要 USB 鼠标、USB 键盘能工作，USB-HID 驱动就是好的，不要安装 USB-HID 驱动，免驱。

在 D-/P3.0，D+/P3.1 与 PC-USB 端口连接好的状况下，USB-ISP 下载程序有如下三种模式：

【USB 下载方法一，P3.2 按键，再结合停电上电下载】

1、按下板子上的 P3.2/INT0 按键，就是 P3.2 接地

2、给目标芯片重新上电，不管之前是否已通电。

===电子开关是按下停电后再松开就是上电

等待电脑端 ISP 下载软件中自动识别出"(HID1) USB Writer"后，就与 P3.2 状态无关了，这时可以松开 P3.2 按键（P3.2 在用户程序区可做其它任意用途）

===传统的机械自锁紧开关是按上来停电，按下去是上电

3、点击电脑端下载软件中的【下载/编程】按钮（注意：USB 下载与串口下载的操作顺序不同）

下载进行中，几秒钟后，提示下载成功！

【USB 下载方法二，复位管脚低电平复位下载】

USB 连接好并已上电的情况下，外部按键复位也可进入 USB 下载模式，注意：

P4.7-nRST 出厂时默认是 P4.7-I/O 功能，要改为复位功能，需 ISP 烧录时取消设置复位脚用作 I/O 口，停电一次再上电才生效，程序区中用户程序也可改为复位脚或 I/O，这个立即生效。

1、按下 P4.7-nRST 外接的低电平复位按键复位 MCU，

松开复位键，MCU 从系统程序区启动，判断是否要下载用户程序，
等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后
2、点击电脑端下载软件中的【下载/编程】按钮
下载进行中，几秒钟后，提示下载成功！

备注：如果目标 MCU 的用户程序区已有 USB 通讯程序（含 CDC、HID 等），则通过复位脚复位进入 USB 下载模式不被支持，只能通过停电上电冷启动下载或者从用户程序区软复位到系统区下载

【USB 下载方法三，从用户程序区软复位到系统区下载】

USB 连接好并已上电的情况下，从用户程序区软复位到系统区也可进入 USB 下载模式
1、在用户程序区运行软复位到系统区的程序，就是 IAP_CONTR 寄存器送 60H
等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后
2、点击电脑端下载软件中的【下载/编程】按钮
下载进行中，几秒钟后，提示下载成功！

USB 下载 注意事项：

拔插 USB 插头不能代替上面线路图中的电源开关。正常操作步骤如下：

USB 的【Gnd，D+，D-】接好的情况下，按下 P3.2 按键接地，再通过正常的电源开关给 MCU 供电或重新供电，让 MCU 冷启动进入系统程序区，判断是否需要等待电脑端 USB 下载程序。

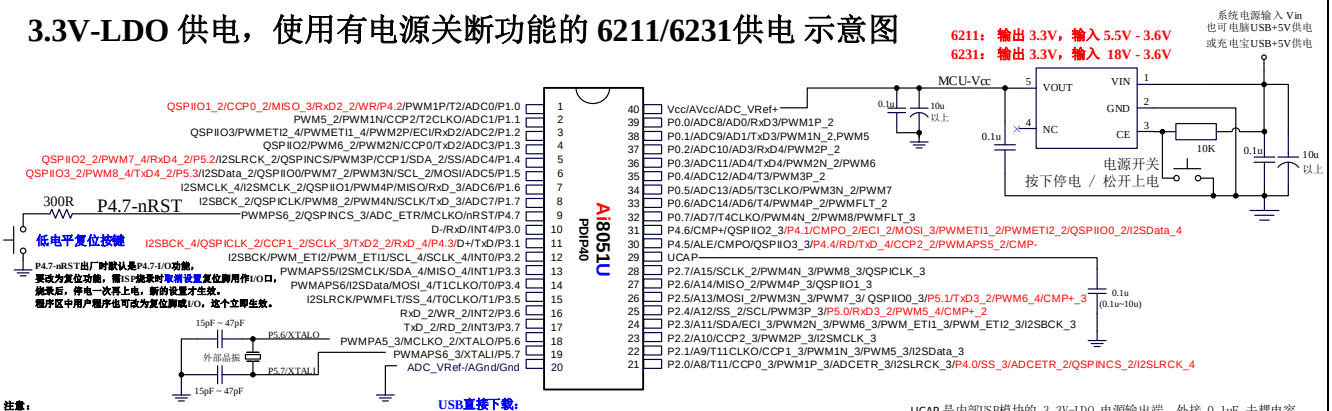
拔插 USB 插头代替电源开关不能每次都能成功进行 USB 下载的原因：

拔插 USB 插头，如【Gnd，USB+5V】已接触好，已供电，而【D+，D-】有一个甚至两个信号线还没有接触好，MCU 已上电，开始跑系统区程序时，发现 USB 还没接触好，则会从系统区软复位到用户程序区跑用户程序，不再进入等待 USB 下载模式，本次就无法顺利进行 USB 下载。

很多人经过多次插拔 USB，才能碰到 1 次【D+，D-】接触好的情况下，【Gnd，USB+5V】才开始接触好，才开始供电，才能成功进入 USB 下载。插 USB 插头代替供电，不能保证【Gnd，D+，D-，USB+5V】的接触顺序，所以，必须使用正常的电源开关，才能确保每次下载都能成功。

4.3.4　如果使用 3.3V-LDO 供电，建议用有电源关断功能的 6211/6231

3.3V-LDO 供电，使用有电源关断功能的 6211/6231供电 示意图



4.3.5 管脚说明

编号			名称	类型	说明
LQFP48/QFN48	LQFP44	PDIP40			
1	1	6	P5.3	I/O	标准 I/O 口
			TxD4_2	O	串口 4 的发送脚
			PWM8_4	I/O	PWM8 的捕获输入和脉冲输出
			QSPIIO3_2	I/O	QSPI 数据脚
2	1	6	P1.5	I/O	标准 I/O 口
			ADC5	I	ADC 模拟输入通道 5
			MOSI	I/O	SPI 主机输出从机输入
			SCL_2	I/O	I2C 的时钟线
			PWM3N	O	PWM3 的脉冲输出负极
			PWM7_2	I/O	PWM7 的捕获输入和脉冲输出
			QSPIIO0	I/O	QSPI 数据脚
			I2SSD_2	I/O	I2S 数据脚
3	2	7	P1.6	I/O	标准 I/O 口
			ADC6	I	ADC 模拟输入通道 6
			RxD_3	I	串口 1 的接收脚
			MISO	I/O	SPI 主机输入从机输出
			PWM4P	I/O	PWM4 的捕获输入和脉冲输出正极
			QSPIIO1	I/O	QSPI 数据脚
			I2SMCLK_2	I/O	I2S 主时钟脚
			I2SMCLK_4	I/O	I2S 主时钟脚
4	3	8	P1.7	I/O	标准 I/O 口
			ADC7	I	ADC 模拟输入通道 7
			TxD_3	O	串口 3 的发送脚
			SCLK	I/O	SPI 的时钟脚
			PWM4N	O	PWM4 的脉冲输出负极
			PWM8_2	I/O	PWM8 的捕获输入和脉冲输出
			QSPICLK	O	QSPI 时钟脚
			I2SCK_2	I/O	I2S 位时钟脚
5	4	9	P4.7	I/O	标准 I/O 口
			nRST	I	复位引脚（低电平复位）
			MCLKO	O	主时钟分频输出
			ADCETR	I	ADC 外部触发脚
			QSPINCS_3	O	QSPI 片选脚
			PWMAPS6_2	I/O	PWM 硬件移相端口
6	5	10	P3.0	I/O	标准 I/O 口
			INT4	I	外部中断 4 输入脚
			RxD	I	串口 1 的接收脚
			D-	I/O	USB 差分 D-脚

编号			名称	类型	说明
LQFP48/QFN48	LQFP44	PDIP40			
7	6	11	P4.3	I/O	标准 I/O 口
			RxD_4	I	串口 1 的接收脚
			TxD2_2	O	串口 2 的发送脚
			SCLK_3	I/O	SPI 的时钟脚
			CCP1_2	I/O	CCP1 的捕获输入和脉冲输出
			QSPICLK_2	O	QSPI 时钟脚
			I2SCK_4	I/O	I2S 位时钟脚
8	7	11	P3.1	I/O	标准 I/O 口
			TxD	O	串口 1 的发送脚
			D+	I/O	USB 差分 D+脚
9	8	12	P3.2	I/O	标准 I/O 口
			INT0	I	外部中断 0 输入脚
			SCLK_4	I/O	SPI 的时钟脚
			SCL_4	I/O	I2C 的时钟线
			PWMETI1	I	PWMA 外部触发输入脚
			PWMETI2	I	PWMB 外部触发输入脚
			I2SCK	I/O	I2S 位时钟脚
10	9	13	P3.3	I/O	标准 I/O 口
			INT1	I	外部中断 1 输入脚
			MISO_4	I/O	SPI 主机输入从机输出
			SDA_4	I/O	I2C 数据脚
			I2SMCLK	I/O	I2S 主时钟脚
			PWMAPS5	I/O	PWM 硬件移相端口
11	10	14	P3.4	I/O	标准 I/O 口
			T0	I	定时器 0 外部时钟输入
			T1CLKO	O	定时器 1 时钟分频输出
			MOSI_4	I/O	SPI 主机输出从机输入
			I2SSD	I/O	I2S 数据脚
			PWMAPS6	I/O	PWM 硬件移相端口
12	11	15	P3.5	I/O	标准 I/O 口
			T1	I	定时器 1 外部时钟输入
			T0CLKO	O	定时器 0 时钟分频输出
			SS_4	I	SPI 的从机选择脚（主机为输出）
			PWMFLT	I	PWMA/PWMB 的外部异常检测脚
			I2SWS	I/O	I2S 左右时钟脚
13	12	16	P3.6	I/O	标准 I/O 口
			INT2	I	外部中断 2 输入脚
			WR_2	O	外部总线写信号输出脚
			RxD_2	I	串口 1 的接收脚

编号			名称	类型	说明
LQFP48/QFN48	LQFP44	PDIP40			
14	13	17	P3.7	I/O	标准 I/O 口
			INT3	I	外部中断 3 输入脚
			RD_2	O	外部总线读信号输出脚
			TxD_2	O	串口 1 的发送脚
15	14	18	P5.6	I/O	标准 I/O 口
			XTALO	O	外部晶振型号输出脚
			MCLKO_2	O	主时钟分频输出
			PWMAPS5_3	I/O	PWM 硬件移相端口
16	15	19	P5.7	I/O	标准 I/O 口
			XTALI	I	外部晶振型号输入脚
			PWMAPS6_3	I/O	PWM 硬件移相端口
17	16	20	Gnd	Gnd	地线
			ADC_AGnd	Gnd	ADC 地线
			ADC_VRef-	Ref	ADC 外部参考电压源输入脚
18	17	21	P4.0	I/O	标准 I/O 口
			SS_3	I	SPI 的从机选择脚（主机为输出）
			ADCETR_2	I	ADC 外部触发脚
			QSPINCS_2	O	QSPI 片选脚
			I2SWS_4	I/O	I2S 左右时钟脚
19	18	21	P2.0	I/O	标准 I/O 口
			A8	O	地址总线
			T11	I	定时器 11 外部时钟输入
			CCP0_3	I/O	CCP0 的捕获输入和脉冲输出
			PWM1P_3	I/O	PWM1 的捕获输入和脉冲输出正极
			ADCETR_3	I	ADC 外部触发脚
			I2SWS_3	I/O	I2S 左右时钟脚
20	19	22	P2.1	I/O	标准 I/O 口
			A9	O	地址总线
			T11CLKO	O	定时器 11 时钟分频输出
			CCP1_3	I/O	CCP1 的捕获输入和脉冲输出
			PWM1N_3	O	PWM1 的脉冲输出负极
			PWM5_3	I/O	PWM5 的捕获输入和脉冲输出
			I2SSD_3	I/O	I2S 数据脚
21	20	23	P2.2	I/O	标准 I/O 口
			A10	O	地址总线
			CCP2_3	I/O	CCP2 的捕获输入和脉冲输出
			PWM2P_3	I/O	PWM2 的捕获输入和脉冲输出正极
			I2SMCLK_3	I/O	I2S 主时钟脚

编号			名称	类型	说明
LQFP48/QFN48	LQFP44	PDIP40			
22	21	24	P2.3	I/O	标准 I/O 口
			A11	O	地址总线
			SDA	I/O	I2C 数据脚
			ECI_3	I	CCP 模块的外部时钟脚
			PWM2N_3	O	PWM2 的脉冲输出负极
			PWM6_3	I/O	PWM6 的捕获输入和脉冲输出
			PWMETI_3	I	PWMA 外部触发输入脚
			PWMETI2_3	I	PWMB 外部触发输入脚
			I2SCK_3	I/O	I2S 位时钟脚
23	22	25	P2.4	I/O	标准 I/O 口
			A12	O	地址总线
			SS_2	I	SPI 的从机选择脚（主机为输出）
			SCL	I/O	I2C 的时钟线
			PWM3P_3	I/O	PWM3 的捕获输入和脉冲输出正极
24	22	25	P5.0	I/O	标准 I/O 口
			RxD3_2	I	串口 3 的接收脚
			PWM5_4	I/O	PWM5 的捕获输入和脉冲输出
			CMP+_2	I	比较器正极输入
25	23	26	P5.1	I/O	标准 I/O 口
			TxD3_2	O	串口 3 的发送脚
			PWM6_4	I/O	PWM6 的捕获输入和脉冲输出
			CMP+_3	I	比较器正极输入
26	23	26	P2.5	I/O	标准 I/O 口
			A13	O	地址总线
			MOSI_2	I/O	SPI 主机输出从机输入
			PWM3N_3	O	PWM3 的脉冲输出负极
			PWM7_3	I/O	PWM7 的捕获输入和脉冲输出
			QSPIIO0_3	I/O	QSPI 数据脚
27	24	27	P2.6	I/O	标准 I/O 口
			A14	O	地址总线
			MISO_2	I/O	SPI 主机输入从机输出
			PWM4P_3	I/O	PWM4 的捕获输入和脉冲输出正极
			QSPIIO1_3	I/O	QSPI 数据脚
28	25	28	P2.7	I/O	标准 I/O 口
			A15	O	地址总线
			SCLK_2	I/O	SPI 的时钟脚
			PWM4N_3	O	PWM4 的脉冲输出负极
			PWM8_3	I/O	PWM8 的捕获输入和脉冲输出
			QSPICLK_3	O	QSPI 时钟脚
29	26	29	UCAP	CAP	USB LDO 外接电容较

编号			名称	类型	说明
LQFP48/QFN48	LQFP44	PDIP40			
30	27	30	P4.5	I/O	标准 I/O 口
			ALE	O	地址锁存信号
			CMPO	O	比较器结果输出脚
			QSPIIO3_3	I/O	QSPI 数据脚
30	27	30	P4.4	I/O	标准 I/O 口
			RD	O	外部总线读信号输出脚
			TxD_4	O	串口 1 的发送脚
			CCP2_2	I/O	CCP2 的捕获输入和脉冲输出
			PWMAPS5	I/O	PWM 硬件移相端口
			CMP-	I	比较器负极输入
31	28	31	P4.1	I/O	标准 I/O 口
			CMPO_2	O	比较器结果输出脚
			ECI_2	I	CCP 模块的外部时钟脚
			MOSI_3	I/O	SPI 主机输出从机输入
			PWMETI_2	I	PWMA 外部触发输入脚
			PWMETI2_2	I	PWMB 外部触发输入脚
			QSPIIO0_2	I/O	QSPI 数据脚
			I2SSD_4	I/O	I2S 数据脚
32	29	31	P4.6	I/O	标准 I/O 口
			CMP+	I	比较器正极输入
			QSPIIO2_3	I/O	QSPI 数据脚
33	30	32	P0.7	I/O	标准 I/O 口
			AD7	I/O	地址/数据总线
			T4CLKO	O	定时器 4 时钟分频输出
			PWM4N_2	O	PWM4 的脉冲输出负极
			PWM8	I/O	PWM8 的捕获输入和脉冲输出
			PWMFLT_3	I	PWMA/PWMB 的外部异常检测脚
34	31	33	P0.6	I/O	标准 I/O 口
			ADC14	I	ADC 模拟输入通道 14
			AD6	I/O	地址/数据总线
			T4	I	定时器 4 外部时钟输入
			PWM4P_2	I/O	PWM4 的捕获输入和脉冲输出正极
			PWMFLT_2	I	PWMA/PWMB 的外部异常检测脚
35	32	34	P0.5	I/O	标准 I/O 口
			ADC13	I	ADC 模拟输入通道 13
			AD5	I/O	地址/数据总线
			T3CLKO	O	定时器 3 时钟分频输出
			PWM3N_2	O	PWM3 的脉冲输出负极
			PWM7	I/O	PWM7 的捕获输入和脉冲输出

编号			名称	类型	说明
LQFP48/QFN48	LQFP44	PDIP40			
36	33	35	P0.4	I/O	标准 I/O 口
			ADC12	I	ADC 模拟输入通道 12
			AD4	I/O	地址/数据总线
			T3	I	定时器 3 外部时钟输入
			PWM3P_2	I/O	PWM3 的捕获输入和脉冲输出正极
37	34	36	P0.3	I/O	标准 I/O 口
			ADC11	I	ADC 模拟输入通道 11
			AD3	I/O	地址/数据总线
			TxD4	O	串口 4 的发送脚
			PWM2N_2	O	PWM2 的脉冲输出负极
			PWM6	I/O	PWM6 的捕获输入和脉冲输出
38	35	37	P0.2	I/O	标准 I/O 口
			ADC10	I	ADC 模拟输入通道 10
			AD2	I/O	地址/数据总线
			RxD4	I	串口 4 的接收脚
			PWM2P_2	I/O	PWM2 的捕获输入和脉冲输出正极
39	36	38	P0.1	I/O	标准 I/O 口
			ADC9	I	ADC 模拟输入通道 9
			AD1	I/O	地址/数据总线
			TxD3	O	串口 3 的发送脚
			PWM1N_2	O	PWM1 的脉冲输出负极
			PWM5	I/O	PWM5 的捕获输入和脉冲输出
40	37	39	P0.0	I/O	标准 I/O 口
			ADC8	I	ADC 模拟输入通道 8
			AD0	I/O	地址/数据总线
			RxD3	I	串口 3 的接收脚
			PWM1P_2	I/O	PWM1 的捕获输入和脉冲输出正极
41	38	40	Vcc	Vcc	电源脚
			ADC_AVcc	Vcc	ADC 电源脚
			ADC_VRef+	Ref	ADC 外部参考电压源输入脚
42	39	1	P4.2	I/O	标准 I/O 口
			WR	O	外部总线写信号输出脚
			RxD2_2	I	串口 2 的接收脚
			MISO_3	I/O	SPI 主机输入从机输出
			CCP0_2	I/O	CCP0 的捕获输入和脉冲输出
			QSPIIO1_2	I/O	QSPI 数据脚
43	40	1	P1.0	I/O	标准 I/O 口
			ADC0	I	ADC 模拟输入通道 0
			T2	I	定时器 2 外部时钟输入
			PWM1P	I/O	PWM1 的捕获输入和脉冲输出正极

编号			名称	类型	说明
LQFP48/QFN48	LQFP44	PDIP40			
44	41	2	P1.1	I/O	标准 I/O 口
			ADC1	I	ADC 模拟输入通道 1
			T2CLKO	O	定时器 2 时钟分频输出
			CCP2	I/O	CCP2 的捕获输入和脉冲输出
			PWM1N	O	PWM1 的脉冲输出负极
			PWM5_2	I/O	PWM5 的捕获输入和脉冲输出
45	42	3	P1.2	I/O	标准 I/O 口
			ADC2	I	ADC 模拟输入通道 2
			RxD2	I	串口 2 的接收脚
			ECI	I	CCP 模块的外部时钟脚
			PWM2P	I/O	PWM2 的捕获输入和脉冲输出正极
			PWMETI1_4	I	PWMA 外部触发输入脚
			PWMWTI2_4	I	PWMB 外部触发输入脚
			QSPIIO3	I/O	QSPI 数据脚
46	43	4	P1.3	I/O	标准 I/O 口
			ADC3	I	ADC 模拟输入通道 3
			TxD2	O	串口 2 的发送脚
			CCP0	I/O	CCP0 的捕获输入和脉冲输出
			PWM2N	O	PWM2 的脉冲输出负极
			PWM6_2	I/O	PWM6 的捕获输入和脉冲输出
			QSPIIO2	I/O	QSPI 数据脚
47	44	5	P1.4	I/O	标准 I/O 口
			ADC4	I	ADC 模拟输入通道 4
			SS	I	SPI 的从机选择脚（主机为输出）
			SDA_2	I/O	I2C 数据脚
			CCP1	I/O	CCP1 的捕获输入和脉冲输出
			PWM3P	I/O	PWM3 的捕获输入和脉冲输出正极
			QSPINCS	O	QSPI 片选脚
			I2SWS_2	I/O	I2S 左右时钟脚
48	44	5	P5.2	I/O	标准 I/O 口
			RxD_2	I	串口 1 的接收脚
			PWM7_4	I/O	PWM7 的捕获输入和脉冲输出
			QSPIIO2_2	I/O	QSPI 数据脚

4.3.6　USB-Link1D 对 Ai8051U 自动停电/上电烧录，串口仿真+串口通讯

USB Link1D工具：支持全自动停电-上电在线下载 / 脱机下载 / 仿真

【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，下载编程进行中，数秒后提示下载编程成功，目标MCU会自动复位到用户程序区自动跑用户程序。

部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P4.7/nRST变成复位脚

USART1_SPI （MOSI和MISO不可切换）

SS	MOSI	MISO	SCLK
P1.4	P1.5	P1.6	P1.7
P2.4	P2.5	P2.6	P2.7
P4.0	P4.1	P4.2	P4.3
P3.5	P3.4	P3.3	P3.2

独立SPI （MOSI和MISO可切换）

SS	MOSI	MISO	SCLK
P1.4	P1.5	P1.6	P1.7
P2.4	P2.5	P2.6	P2.7
P4.0	P4.1	P4.2	P4.3
P3.5	P3.4	P3.3	P3.2

USART1 / USART1-Lin

RxD	TxD
P3.0	P3.1
P3.6	P3.7
P1.6	P1.7
P4.3	P4.4

USART2 / USART2-Lin

RxD2	TxD2
P1.2	P1.3
P4.2	P4.3

UART3

RxD3	TxD3
P0.0	P0.1
P5.0	P5.1

UART4

RxD4	TxD4
P0.2	P0.3
P5.2	P5.3

USART2_SPI （MOSI和MISO不可切换）

SS	MOSI	MISO	SCLK
P1.4	P1.5	P1.6	P1.7
P2.4	P2.5	P2.6	P2.7
P4.0	P4.1	P4.2	P4.3
P3.5	P3.4	P3.3	P3.2

同步/异步串口
USART1,USART2
可做异步串口
或SPI,分时复用
另有独立的SPI
共可实现3组SPI

I²S

MCLK	BCK	LRCK	Data
主时钟	位时钟	左右时钟	音频数据
P3.3	P3.2	P3.5	P3.4
P1.6	P1.7	P1.4	P1.5
P2.2	P2.3	P2.0	P2.1
P1.6	P4.3	P4.0	P4.1

I²C

SCL	SDA
P2.4	P2.3
P1.5	P1.4
-	-
P3.2	P3.3

CMP(比较器)

CMP+	CMP-
P4.6	P4.4
P5.0	1.19V
P5.1	-
ADCIN	-

CMPO

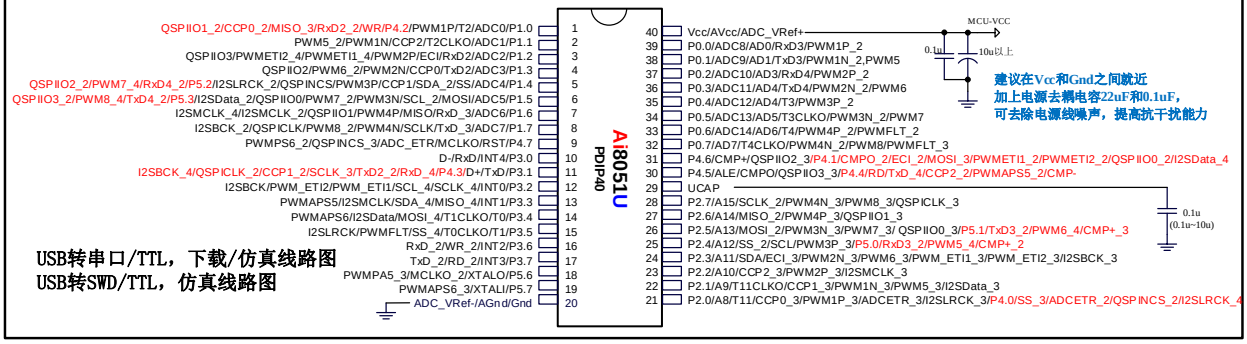
CMPO
P4.5
P4.1

传统外部中断源

INT0	INT1	INT2	INT3	INT4
P3.2	P3.3	P3.6	P3.7	P3.0

QSPI

IO0	IO1	IO2	IO3	SCLK	NCS
P1.5	P1.6	P1.3	P1.2	P1.7	P1.4
P4.1	P4.2	P5.2	P5.3	P4.3	P4.0
P2.5	P2.6	P4.6	P4.5	P2.7	P4.7



备注：上图中红色管脚部分是 2024 年 10 月 15 日之后生产的芯片才有。

ISP 下载步骤：

1、按照如图所示的连接方式将 USB-Link1D 和目标芯片连接

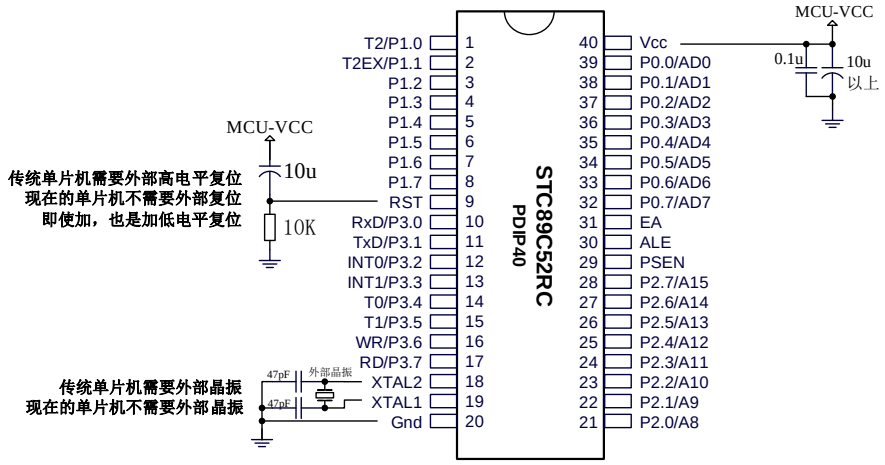
2、点击 ISP 下载软件中的“下载/编程”按钮

3、开始 ISP 下载

（注意：若是使用 USB-Link1D 给目标系统供电，目标系统的总电流不能大于 200mA，否则会导致下载失败。）

比较下传统的 89C52RC 系列相应下载线路图：

传统单片机需要外部晶振，现在的单片机不需要外部晶振
传统单片机需要外部高电平复位，现在的不需要，并且已改为低电平复位



USB Link1D工具：支持全自动停电-上电在线下载 / 脱机下载 / 仿真

【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

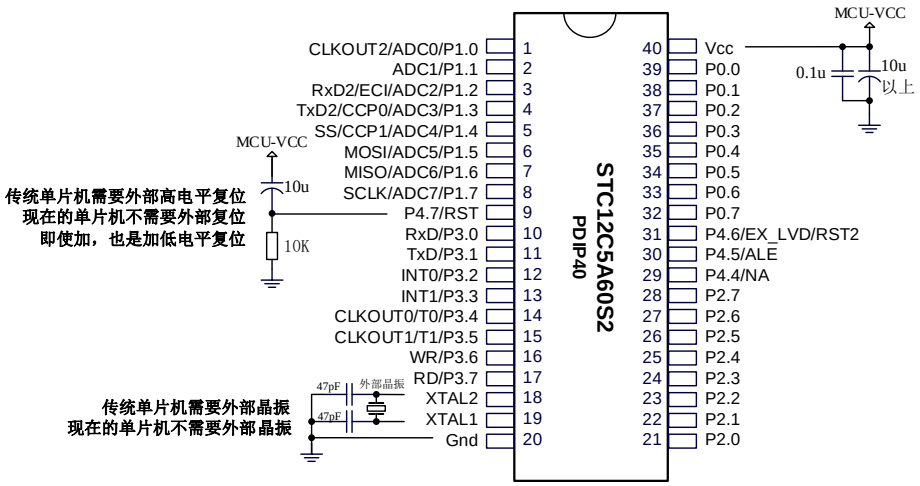
点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮
2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，下载编程进行中，数秒后提示下载编程成功，目标MCU会自动复位到用户程序区自动跑用户程序。
部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P4.7/nRST变成复位脚。

比较下传统的 12C5A60S2 相应下载线路图：

传统单片机需要外部晶振，现在的单片机不需要外部晶振
传统单片机需要外部高电平复位，现在的不需要，并且已改为低电平复位



USB Link1D工具：支持全自动停电-上电在线下载 / 脱机下载 / 仿真

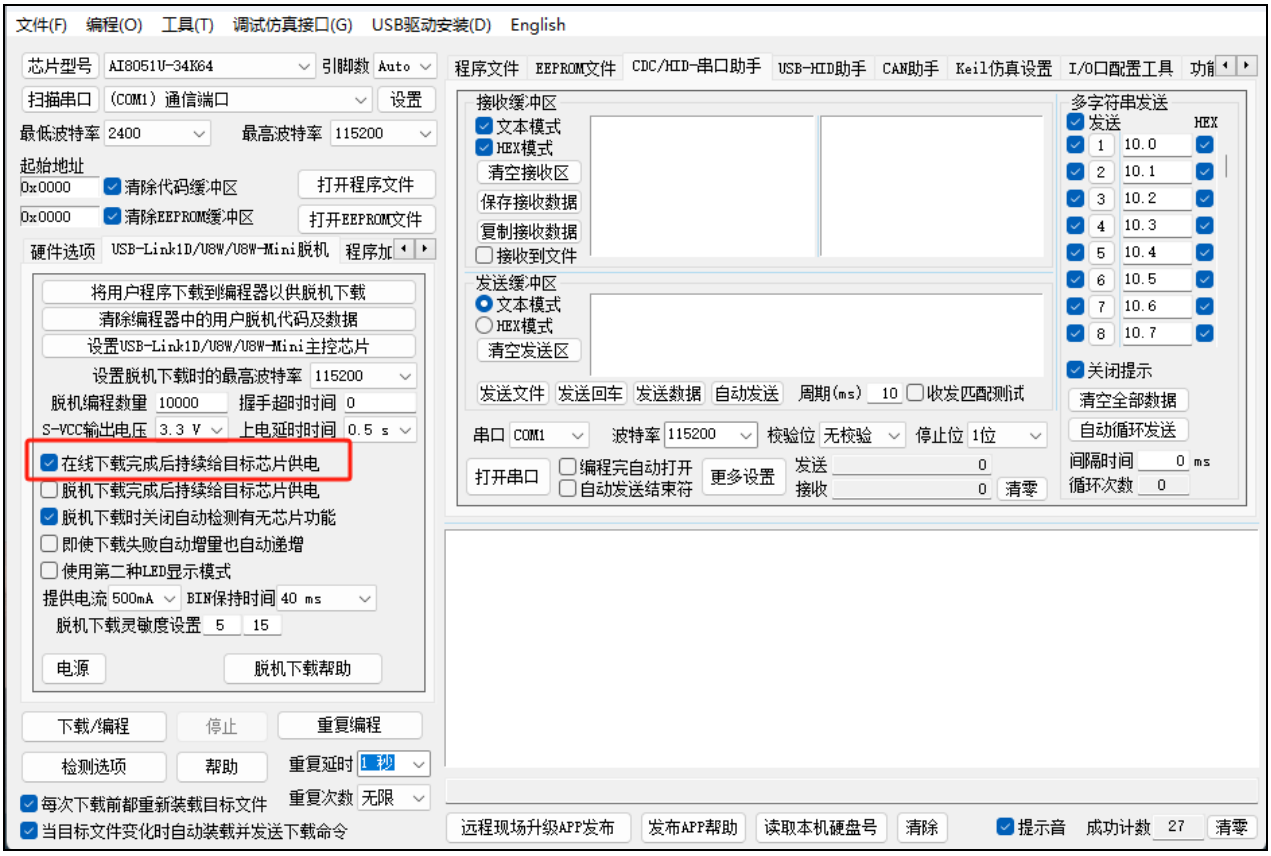
【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

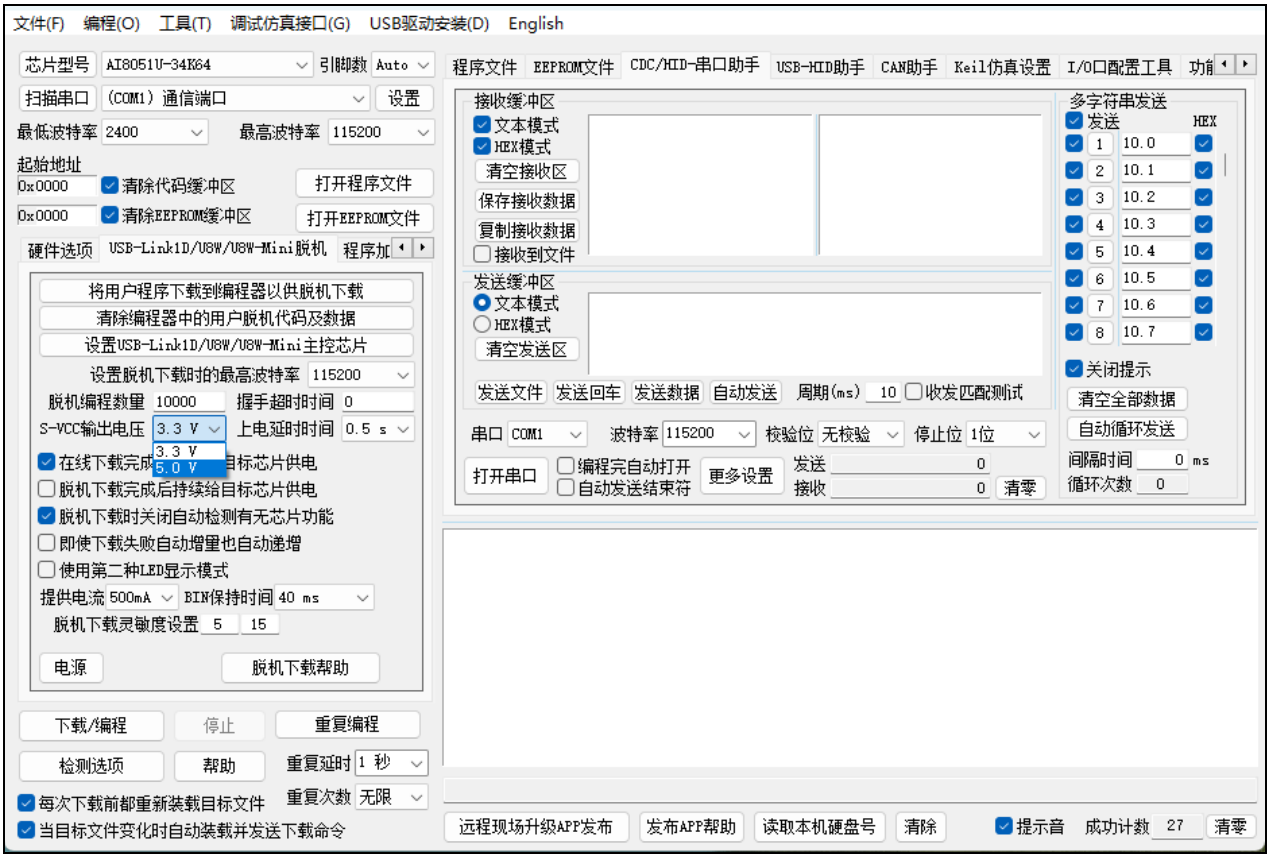
【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮
2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，下载编程进行中，数秒后提示下载编程成功，目标MCU会自动复位到用户程序区自动跑用户程序。
部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P4.7/nRST变成复位脚。

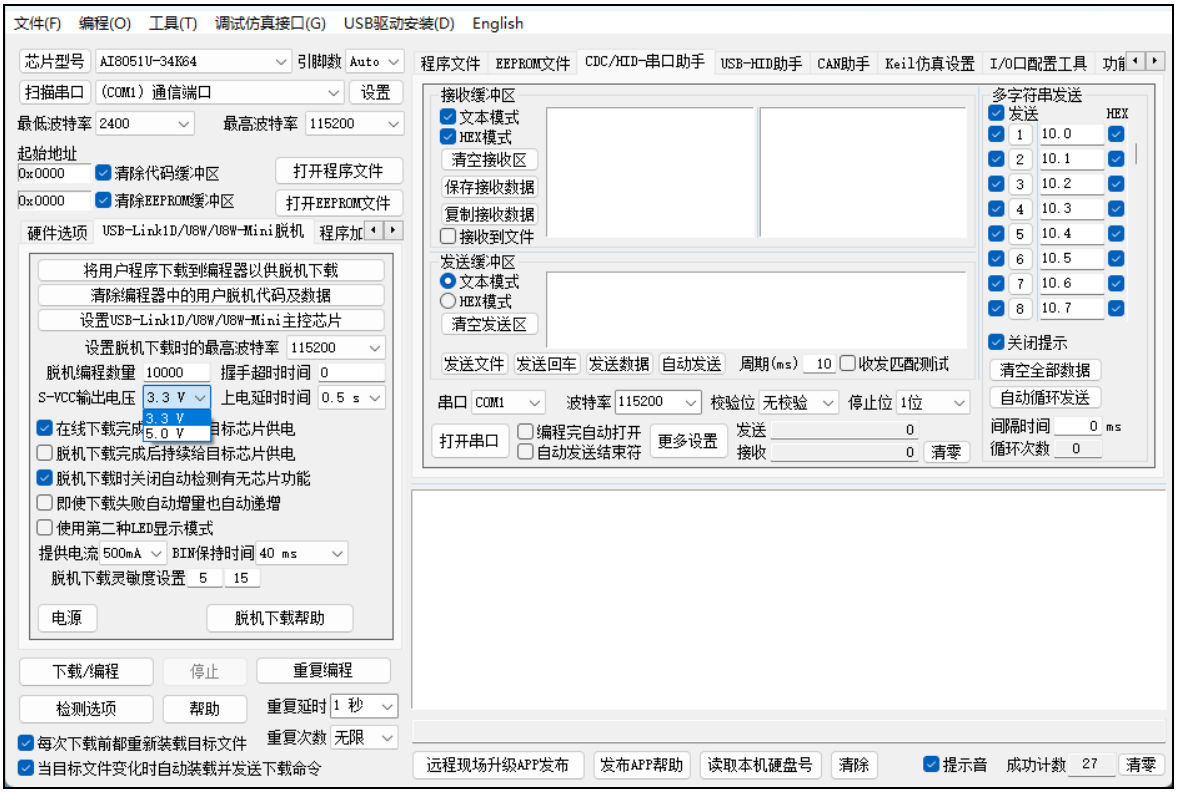
如何设置 USB-Link1D 下载完后持续给目标芯片供电



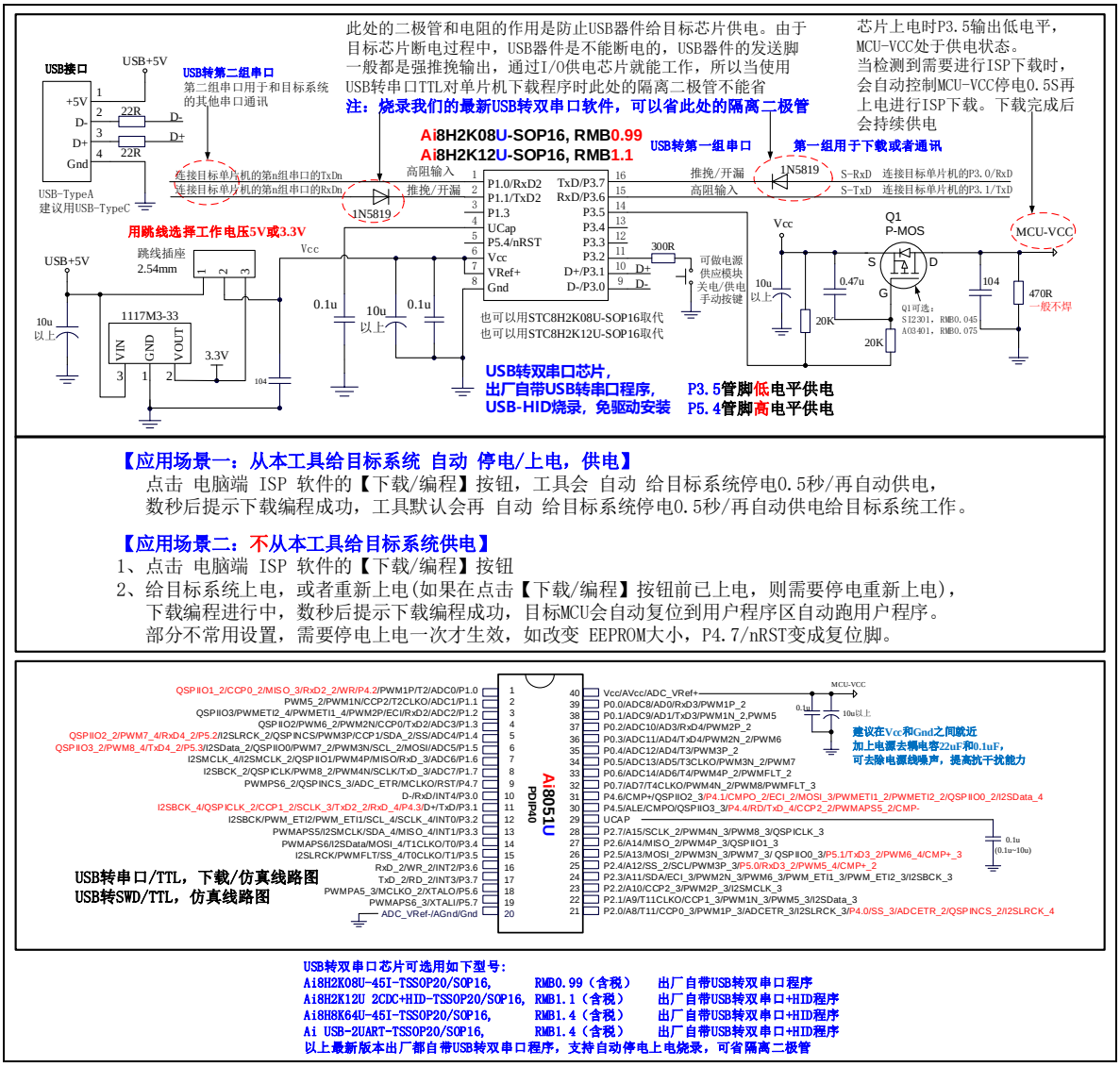
如何设置 USB-Link1D 输出 5V



如何设置 USB-Link1D 输出 3.3V



4.3.7　USB 转双串口芯片进行自动烧录/仿真+串口通讯，5V/3.3V 跳线选择



【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮

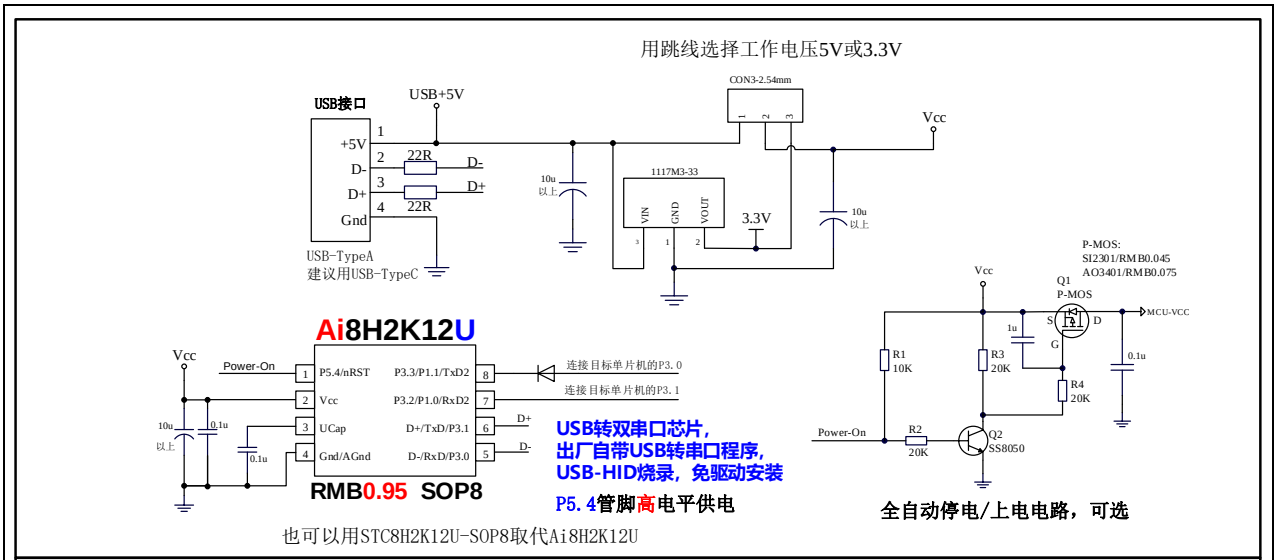
2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，下载编程进行中，数秒后提示下载编程成功，目标MCU会自动复位到用户程序区自动跑用户程序。
部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P4.7/nRST变成复位脚。

USB转双串口芯片可选用如下型号：

Ai8H2K08U-45I-TSSOP20/SOP16,	RMB0.99（含税）	出厂自带USB转双串口程序
Ai8H2K12U 2CDC+HID-TSSOP20/SOP16,	RMB1.1（含税）	出厂自带USB转双串口+HID程序
Ai8H8K64U-45I-TSSOP20/SOP16,	RMB1.4（含税）	出厂自带USB转双串口+HID程序
Ai USB-2UART-TSSOP20/SOP16,	RMB1.4（含税）	出厂自带USB转双串口+HID程序

以上最新版本出厂都自带USB转双串口程序，支持自动停电上电烧录，可省隔离二极管

4.3.8 USB 转串口芯片进行全自动烧录/仿真/通信，5V/3.3V 跳线选择



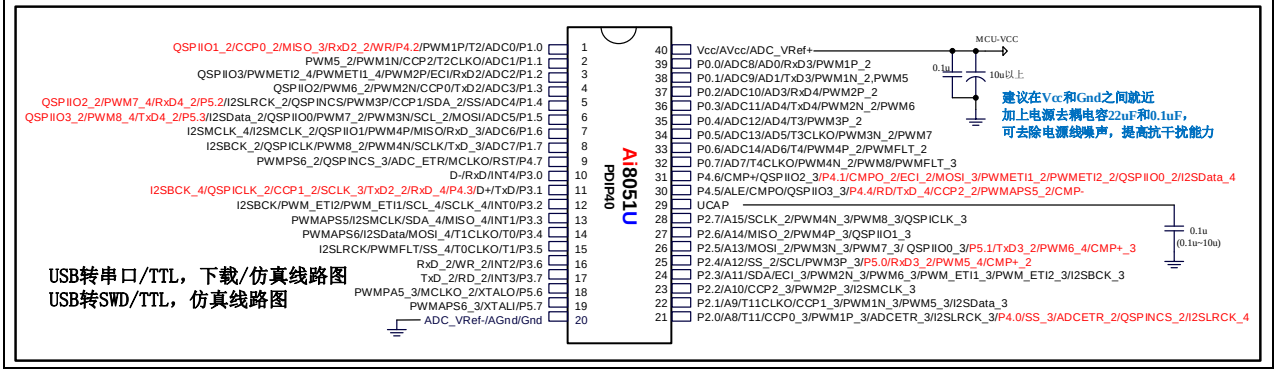
【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，
数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】

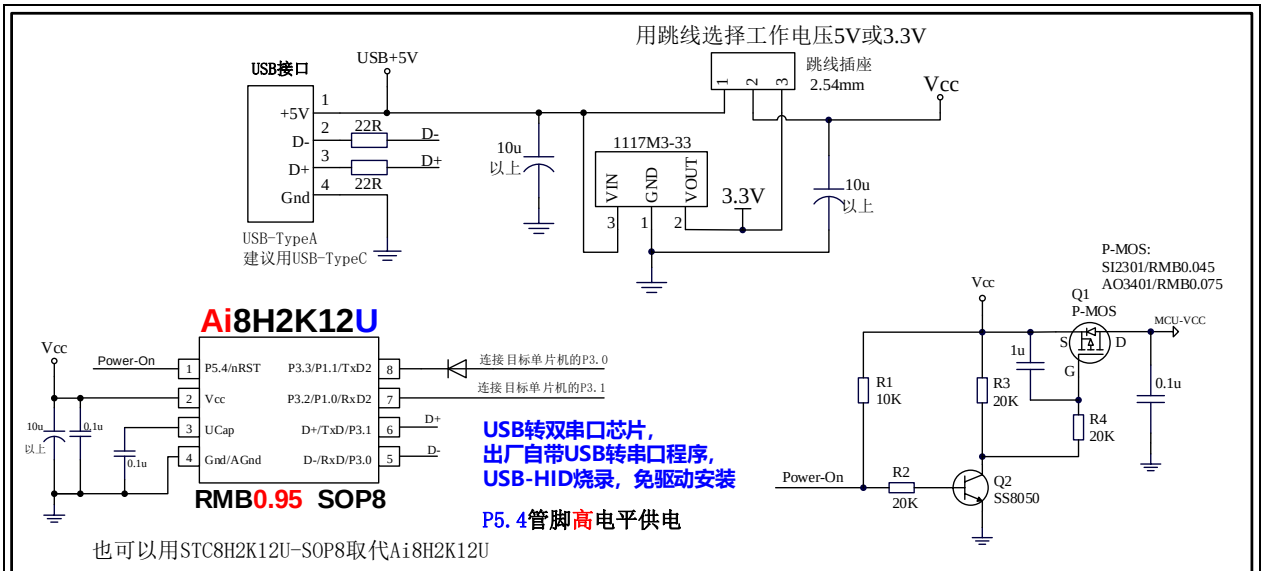
1、点击 电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，
下载编程进行中，数秒后提示下载编程成功，目标MCU会自动复位到用户程序区自动跑用户程序。
部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P4.7/nRST变成复位脚。



备注：上图中红色管脚部分是 2024 年 10 月 15 日之后生产的芯片才有。

比较下传统的 89C52RC 系列相应下载线路图：



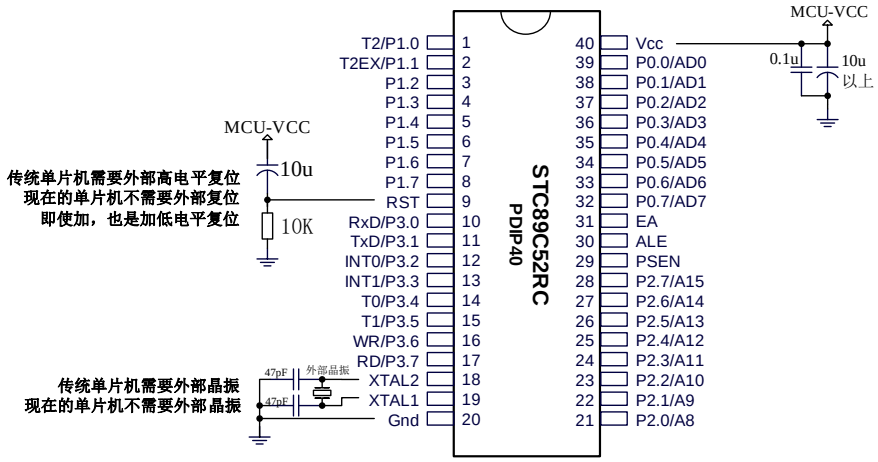
【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，
数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

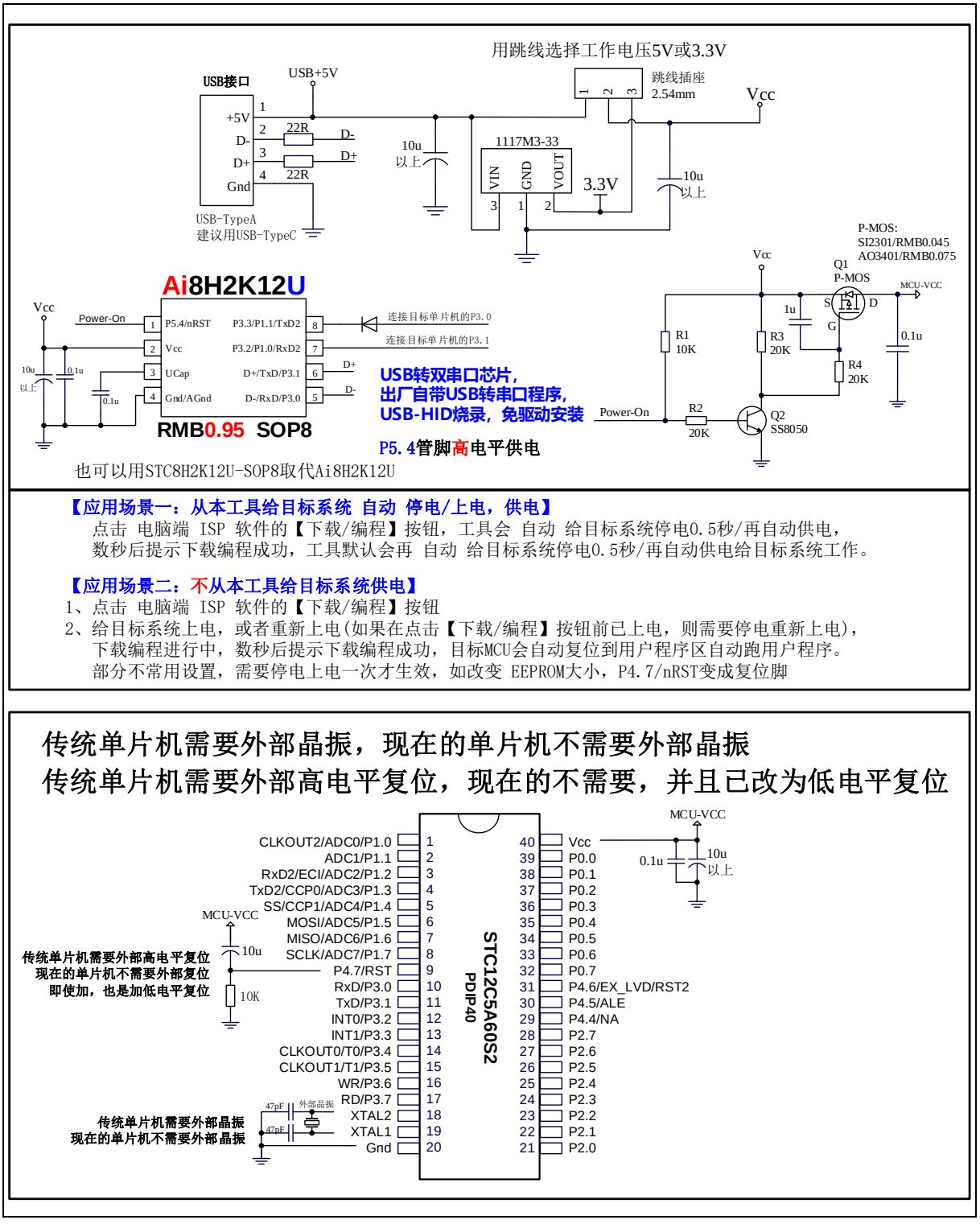
【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮
2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，
下载编程进行中，数秒后提示下载编程成功，目标MCU会自动复位到用户程序区自动跑用户程序。
部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P4.7/nRST变成复位脚

传统单片机需要外部晶振，现在的单片机不需要外部晶振
传统单片机需要外部高电平复位，现在的不需要，并且已改为低电平复位



比较下传统的 12C5A60S2 相应下载线路图：



【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

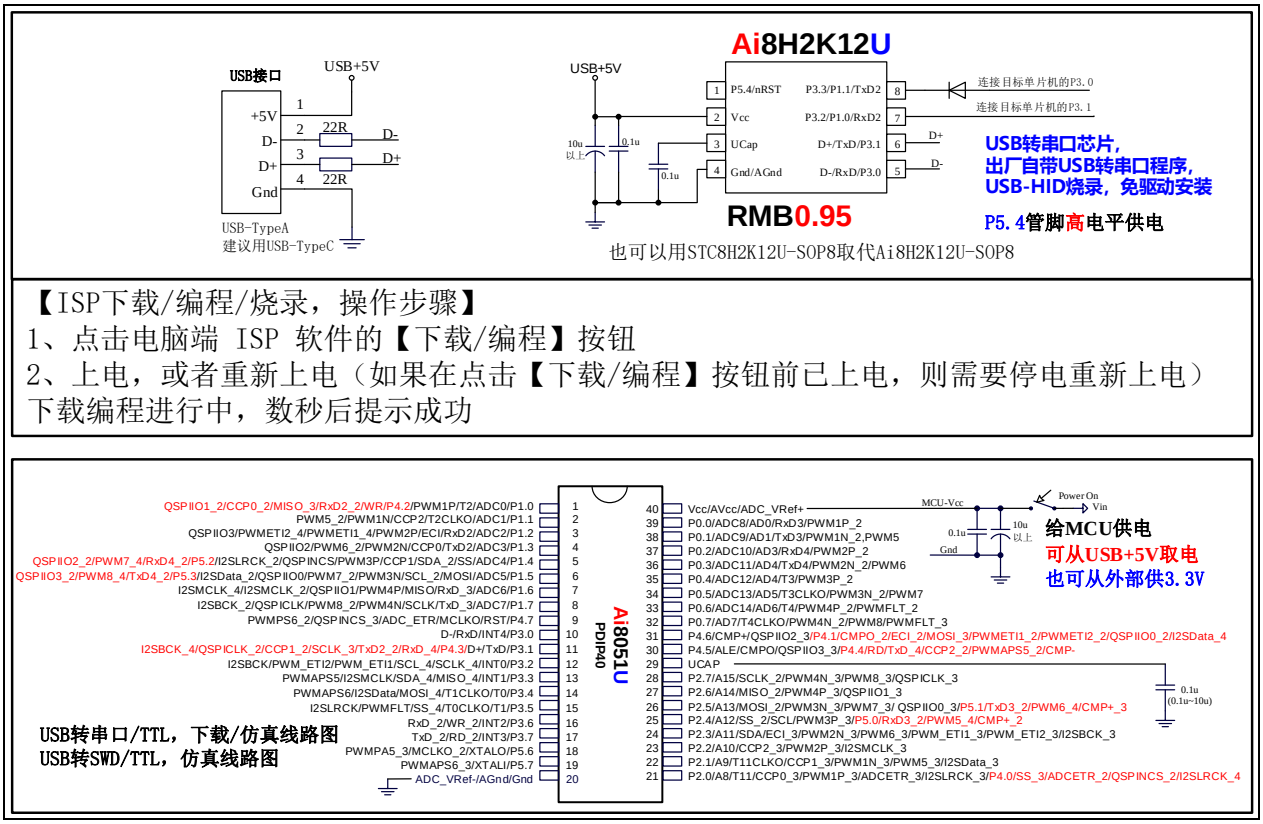
点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，
数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮
2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，
下载编程进行中，数秒后提示下载编程成功，目标MCU会自动复位到用户程序区自动跑用户程序。
部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P4.7/nRST变成复位脚

传统单片机需要外部晶振，现在的单片机不需要外部晶振
传统单片机需要外部高电平复位，现在的不需要，并且已改为低电平复位

4.3.9 USB转串口芯片进行烧录/串口仿真，手动停电/上电，5V/3.3V原理图



【ISP下载/编程/烧录，操作步骤】

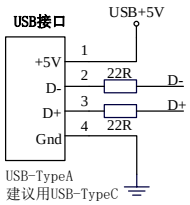
1、点击电脑端 ISP 软件的【下载/编程】按钮

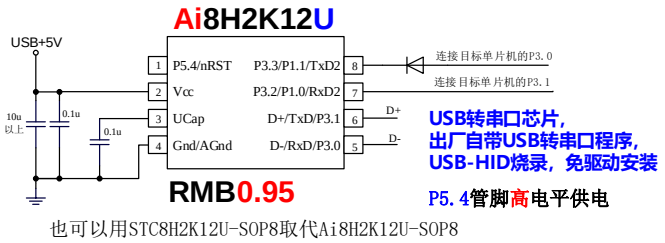
2、上电，或者重新上电（如果在点击【下载/编程】按钮前已上电，则需要停电重新上电）

下载编程进行中，数秒后提示成功

备注：上图中红色管脚部分是 2024 年 10 月 15 日之后生产的芯片才有。

比较下传统的 89C52RC 系列相应下载线路图：





也可以用STC8H2K12U-SOP8取代Ai8H2K12U-SOP8

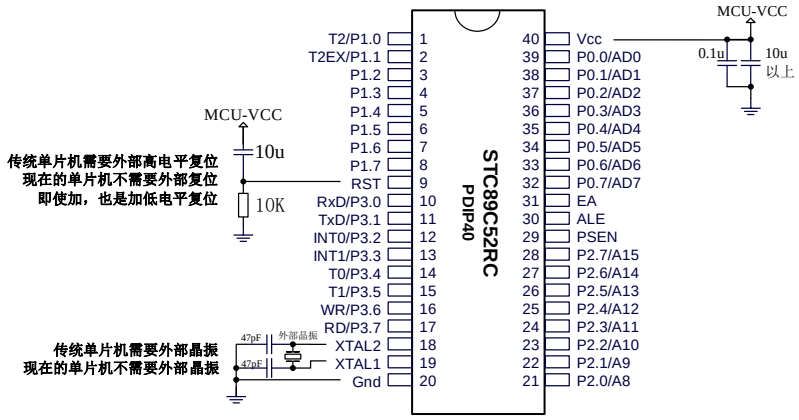
【ISP下载/编程/烧录，操作步骤】

1、点击电脑端 ISP 软件的【下载/编程】按钮

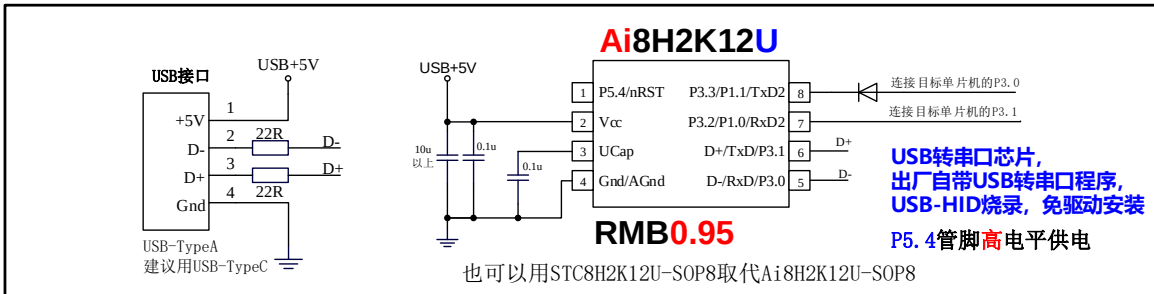
2、上电，或者重新上电（如果在点击【下载/编程】按钮前已上电，则需要停电重新上电）
下载编程进行中，数秒后提示成功

传统单片机需要外部晶振，现在的单片机不需要外部晶振

传统单片机需要外部高电平复位，现在的不需要，并且已改为低电平复位



比较下传统的 12C5A60S2 相应下载线路图，



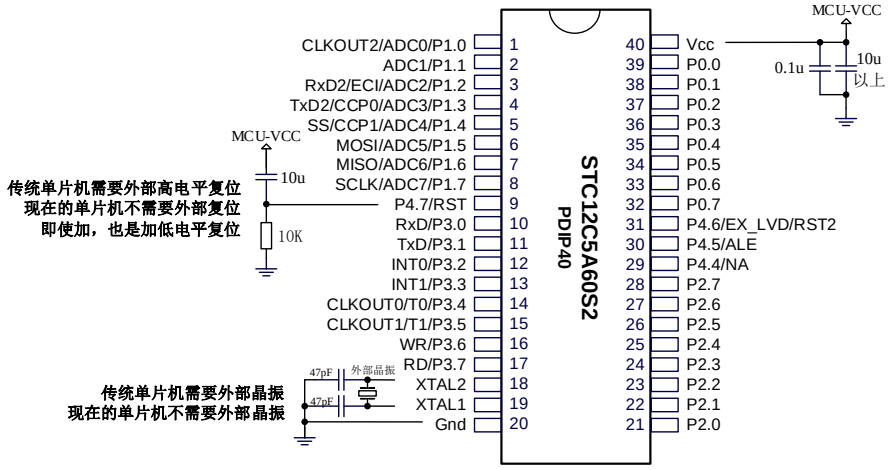
【ISP下载/编程/烧录，操作步骤】

1、点击电脑端 ISP 软件的【下载/编程】按钮

2、上电，或者重新上电（如果在点击【下载/编程】按钮前已上电，则需要停电重新上电）
下载编程进行中，数秒后提示成功

传统单片机需要外部晶振，现在的单片机不需要外部晶振

传统单片机需要外部高电平复位，现在的不需要，并且已改为低电平复位

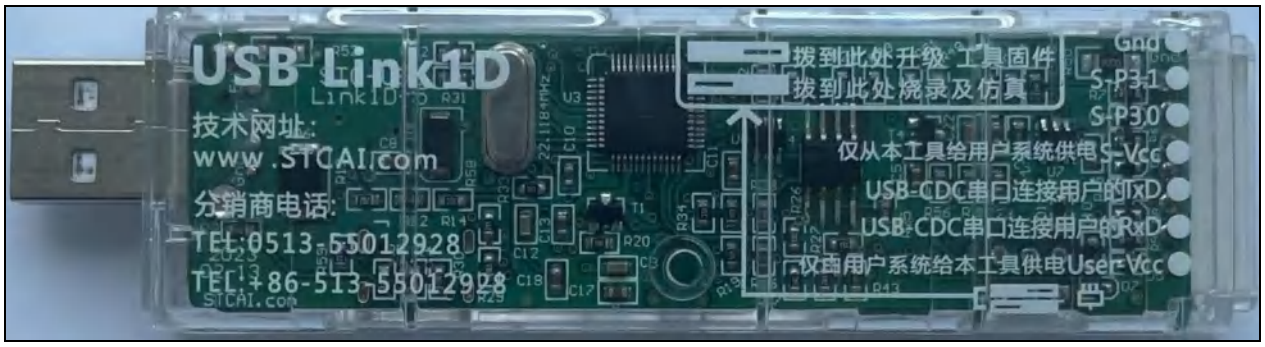


4.4 USB-Link1D 工具强大的配套多种接口豪华线，使用注意事项

4.4.1 工具接口说明

USB-Link1D 工具是 USB Link1 的升级版、功能在 USB Link1 的基础上增加为两个 USB-CDC 串口，可作为通用 USB 转串口使用，【Win10，1903 版本】以后的系统不需要安装 CDC 驱动，并且支持 ISP 烧录时默认是 USB-HID 烧录，免驱动安装。

工具 USB Link1 的使用注意事项请参考附录章节



工具正面图

工具反面图

User-Vcc，外部给工具供电

S-RxD，第2组STC-CDC串口发送脚，接用户单片机串口的接收脚

S-TxD，第2组STC-CDC串口接收脚，接用户单片机串口的发送脚

S-Vcc，本工具给外部系统供电

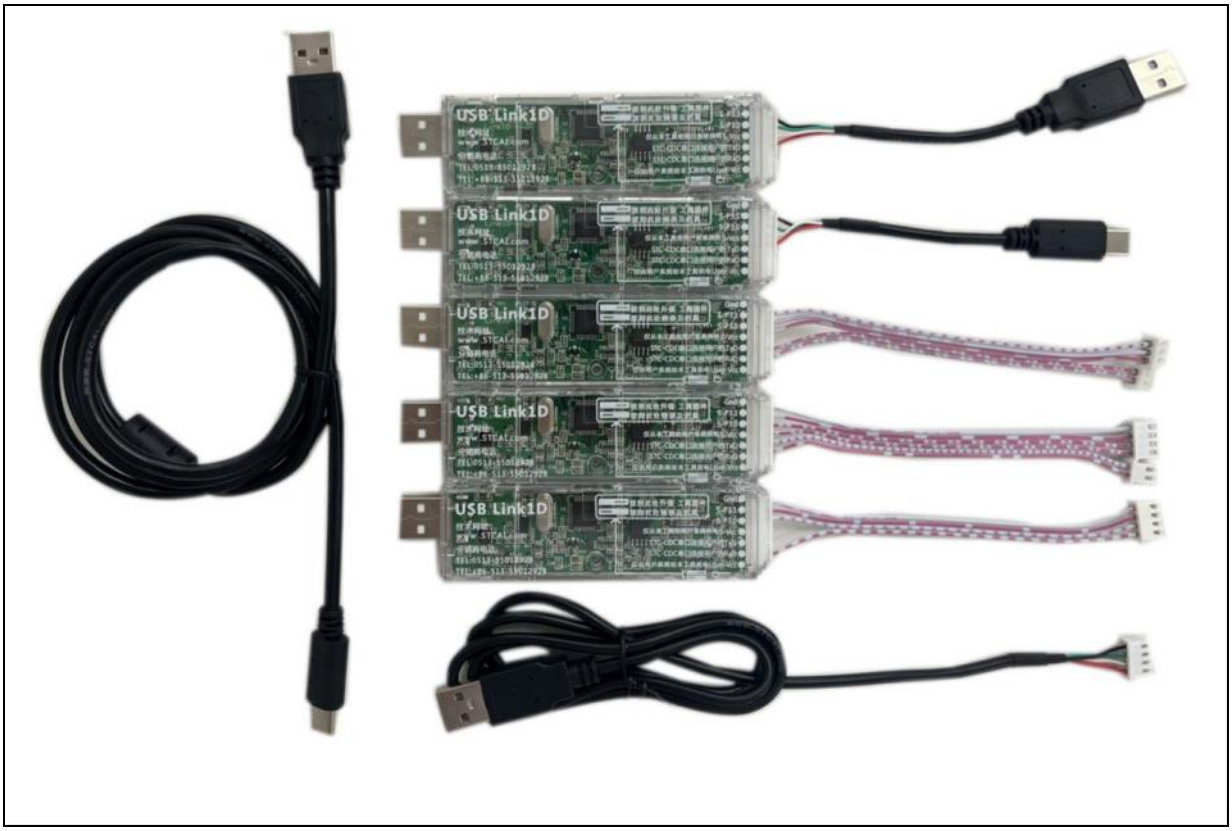
S-P3.0，Link1硬件仿真数据线、Link1进行ISP下载的串口发送脚、第1组STC-CDC串口发送脚

S-P3.1，Link1硬件仿真时钟线、Link1进行ISP下载的串口接收脚、第1组STC-CDC串口接收脚

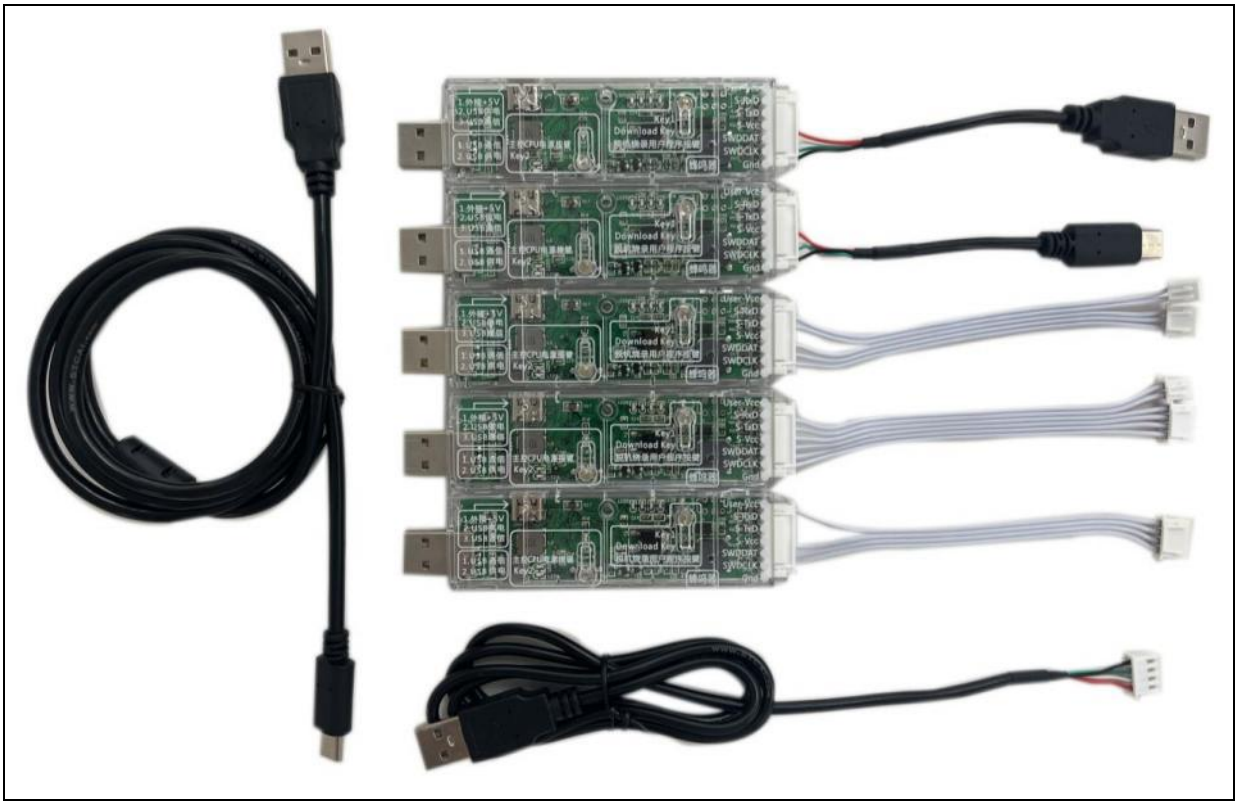
Gnd，地线

管脚编号	接口名称	接口功能
1	User-Vcc	仅由用户系统给本工具供电
2	S-RxD	第 2 组 USB-CDC 串口的发送脚，连接用户单片机串口的接收脚
3	S-TxD	第 2 组 USB-CDC 串口的接收脚，连接用户单片机串口的发送脚
4	S-Vcc	仅从本工具给用户系统供电
5	S-P3.0	使用 Link1D 进行 ISP 下载时的串口发送脚，连接目标单片机的 P3.0
		使用 Link1D 进行 SWD 硬件仿真时的数据脚，连接目标单片机的 SWDDAT
		第 1 组 USB-CDC 串口的发送脚，连接用户单片机串口的接收脚
6	S-P3.1	使用 Link1D 进行 ISP 下载时的串口接收脚，连接目标单片机的 P3.1
		使用 Link1D 进行 SWD 硬件仿真时的时钟脚，连接目标单片机的 SWDCLK
		第 1 组 USB-CDC 串口的接收脚，连接用户单片机串口的发送脚
7	Gnd	地线

4.4.2　附送的各种强大的人性化配线图片及使用说明



正面



反面

USB-Link1D 各种豪华配线的应用场景介绍

USB转双串口/TTL，连接线
不从本工具供电

连接目标用户系统的Gnd
连接目标用户系统的P3.1或串口发送脚
连接目标用户系统的P3.0或串口接收脚
不从本工具给目标系统供电，目标系统自己供电
连接目标用户系统的第n组串口的TxDn
连接目标用户系统的第n组串口的RxDn

这种连接方式为本工具和目标系统各自独立供电，ISP下载需要手动给目标系统停电再上电。
当目标系统的耗电大于300mA时，在线下载、脱机下载、在线仿真，可选这种接线方式。

USB转双串口/TTL，连接线
从本工具供电

连接目标用户系统的Gnd
连接目标用户系统的P3.1或串口发送脚
连接目标用户系统的P3.0或串口接收脚
从本工具给目标用户系统供电，300mA以内
连接目标用户系统的第n组串口的TxDn
连接目标用户系统的第n组串口的RxDn

这种连接方式为本工具给目标系统自动 停电/上电 供电，ISP下载不需要手动停电/上电。
当目标系统电流不超过300mA时，在线下载、脱机下载、在线仿真，可选择这种线方式。

USB转串口/TTL，脱机烧录连接线
从用户系统给本工具供电

连接目标用户系统的Gnd
连接目标用户系统的P3.1
连接目标用户系统的P3.0
目标用户系统给工具供电

这种连接方式为常用的脱机下载连接方式，从用户的目标系统给本脱机工具供电。
如果用户系统电流大于300mA，建议使用这种连接方式。用户系统上电之后自动脱机下载程序，下载成功之后，给用户系统停电，换下一个待烧录用户程序的新的用户系统，再上电，脱机下载程序，重复如上过程即为常见的【批量生产脱机烧录】。
特别注意：用户系统给本工具供电的电压不能超过5.3V

USB转串口/TTL，专门定制的TypeC接口线
但此处为TTL串口，不是USB

连接目标用户系统的Gnd
连接目标用户系统的P3.1或串口发送脚
连接目标用户系统的P3.0或串口接收脚
从本工具给目标用户系统供电，300mmA以内

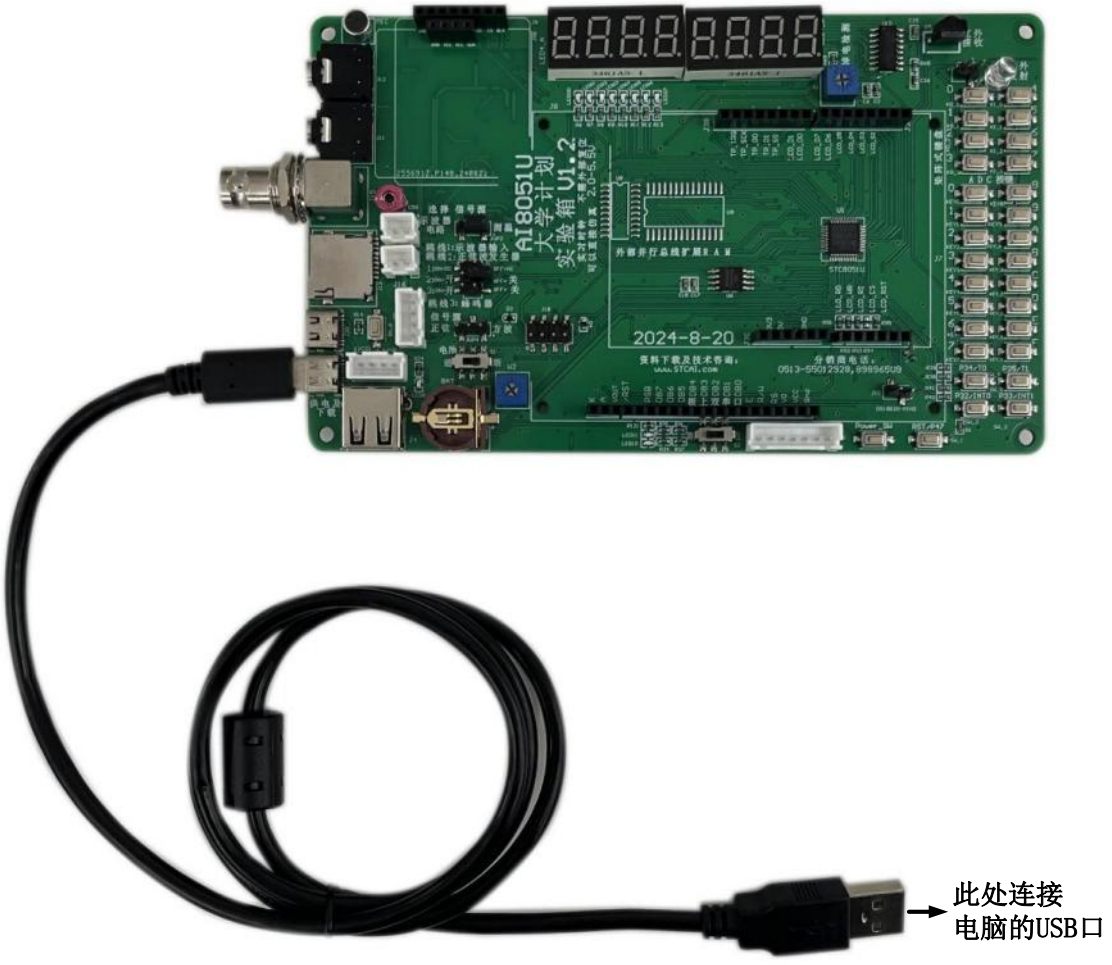
USB转串口/TTL，专门定制的TypeA接口线
但此处为TTL串口，不是USB

连接目标用户系统的Gnd
连接目标用户系统的P3.1或串口发送脚
连接目标用户系统的P3.0或串口接收脚
从本工具给目标用户系统供电，300mmA以内

USB直接下载连接示意图

使用 USB-TypeA 到 USB-TypeC 线

这种一端为 USB-TypeA ，另一端为 USB-TypeC 的标准线，STCAI.com 也会提供这种连接线，方便预留了 USB-TypeC 接口的用户系统，进行硬件USB直接下载。



USB 直接下载连接示意图

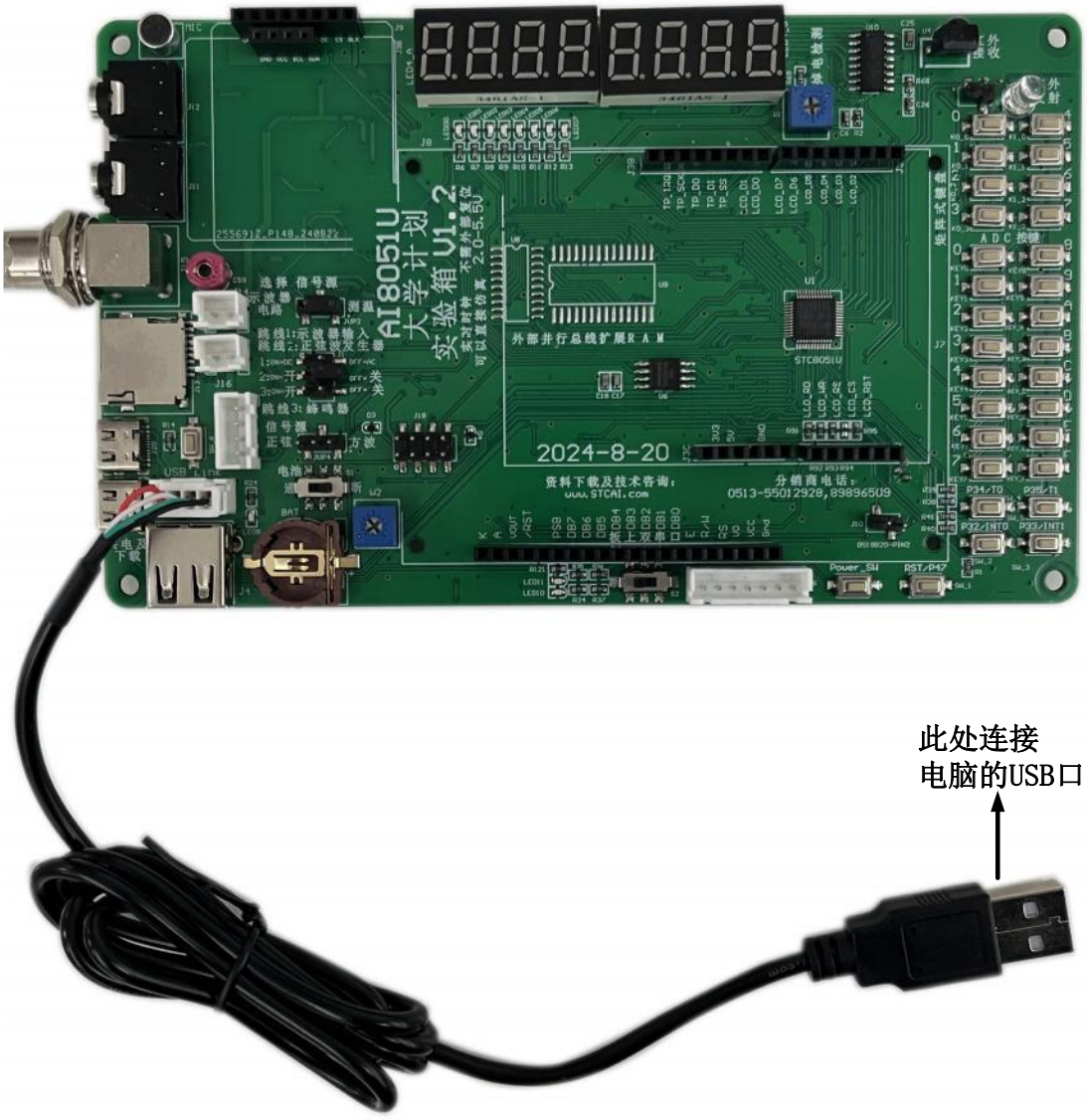
使用 USB-TypeA 到 USB-TypeA 线

这种两端同为 USB-TypeA 接口的标准线，
方便预留了 USB-TypeA 接口的用户系统，进行硬件USB直接下载

USB 直接下载连接示意图

使用 USB-TypeA 到流行的 2.54mm 间距通用插座

STCAI.com提供这种连接线，方便对只预留了2.54mm间距的通用插座目标系统芯片进行硬件USB直接下载



4.4.3 USB-Link1D 实际应用

1、使用 USB-Link1D 工具对 Ai8051U 系列单片机进行 SWD 硬件仿真

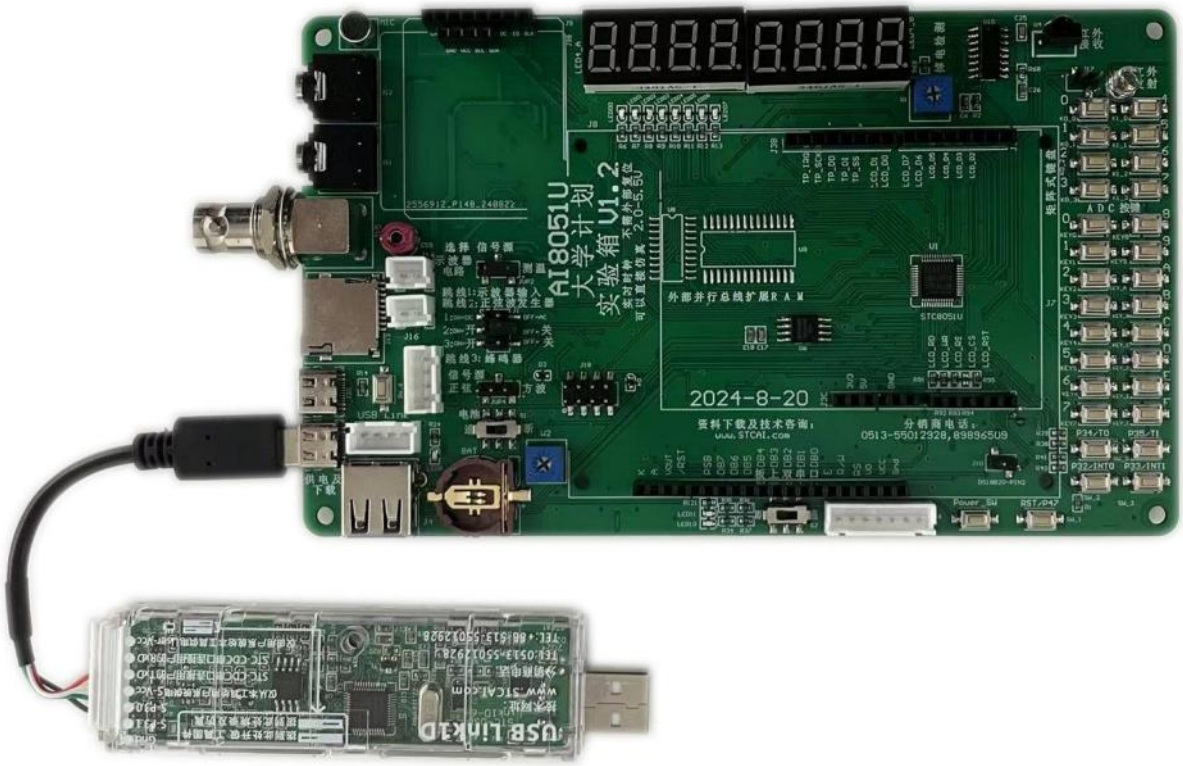
按照如下图所示的方式将工具的 S-Vcc、S-P3.0、S-P3.1、GND 分别与目标单片机的 M-Vcc、P3.0（SWDDAT）、P3.1（SWDCLK）、GND 相连接，然后参考前面章节中硬件仿真的步骤和设置即可进行 SWD 硬件仿真

使用 USB-Link1D 的 USB转串口/TTL，用单排2.54mm间距插头连接目标芯片系统，串口下载连接示意图-正面

使用 USB-Link1D 的 USB转串口/TTL，用单排2.54mm间距插头连接目标芯片系统，串口下载连接示意图-反面

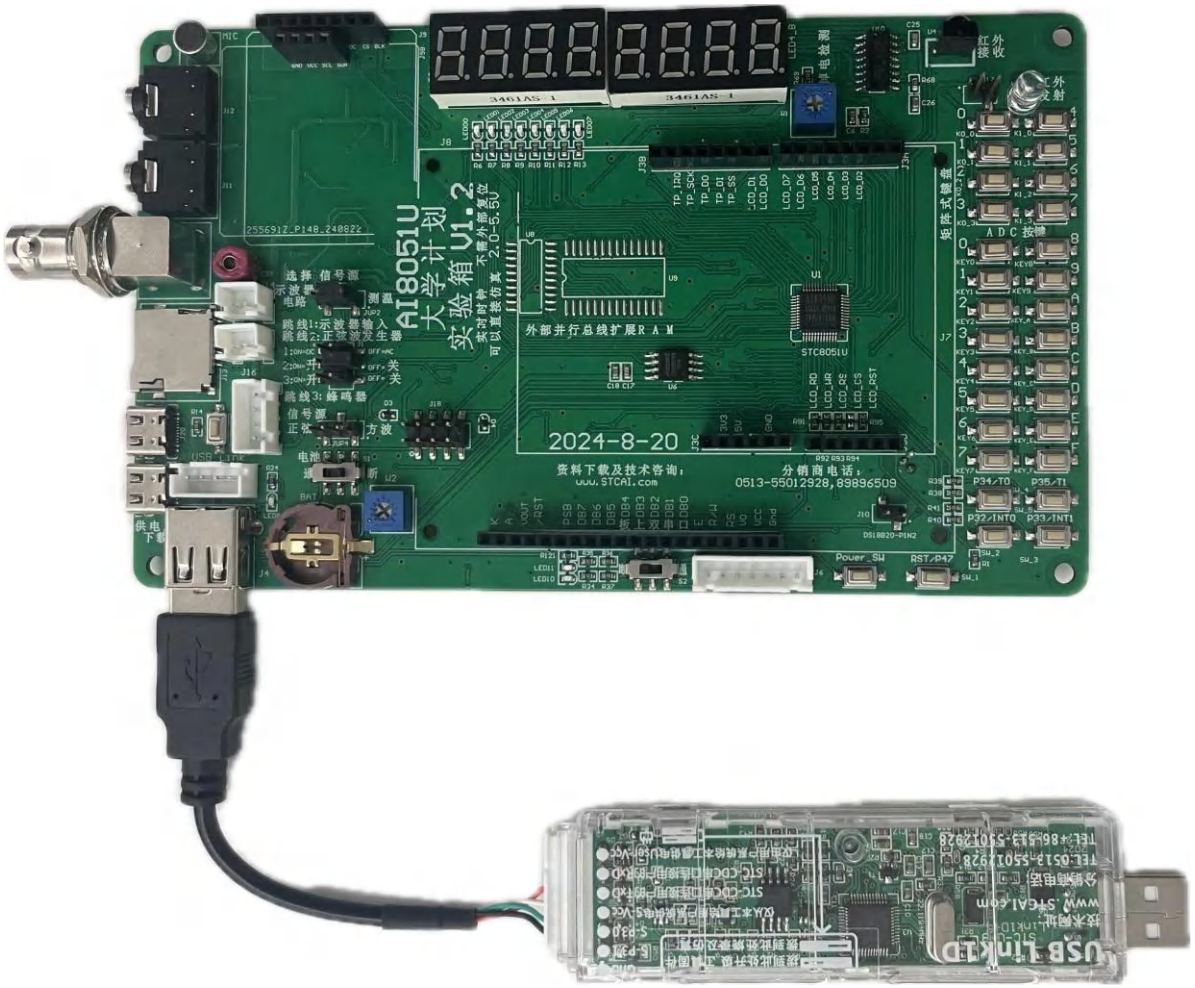
使用USB-Link1D的USB转串口对目标系统进行串口下载

注意：此处连接目标系统的 USB-TypeC接口 是 USB转串口/TTL,
对目标系统芯片 串口/TTL 下载连接示意图



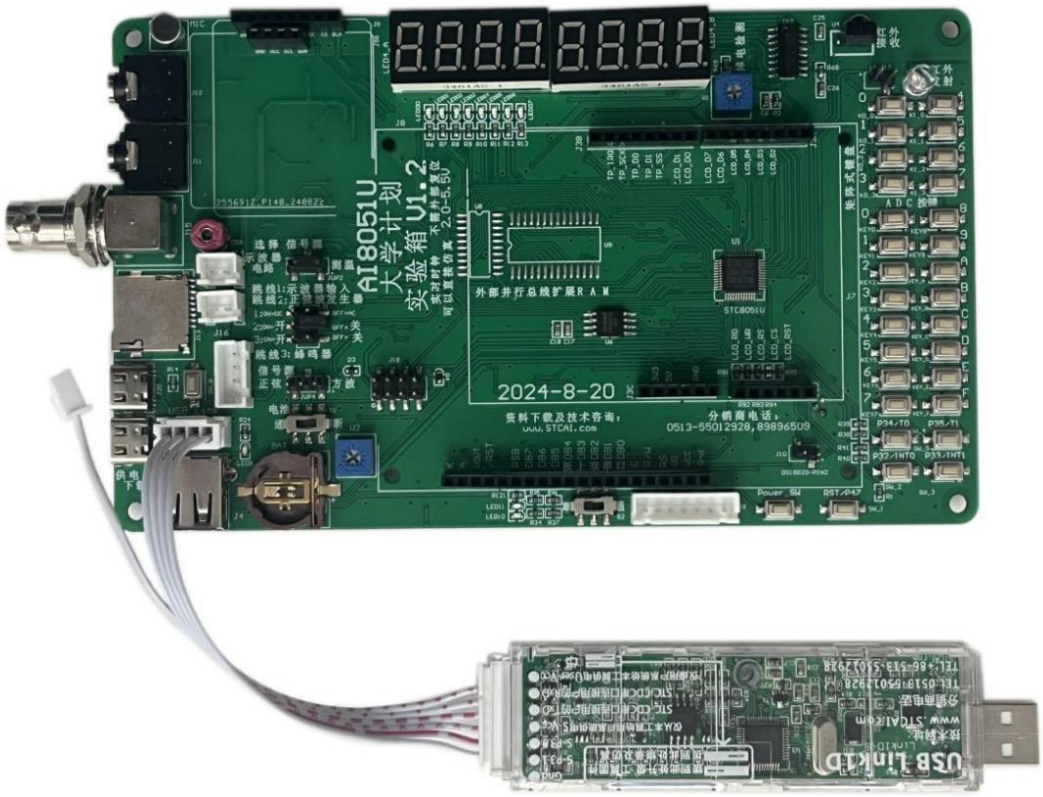
使用USB-Link1D的USB转串口对目标系统进行串口下载

注意：此处连接目标系统的 USB-TypeA接口 是 USB转串口/TTL,
对目标系统芯片 串口/TTL 下载连接示意图



使用USB-Link1D的USB转串口对目标系统进行串口下载

注意：此处连接目标系统的 普通四芯单排插座接口 是 USB转串口/TTL，
对目标系统芯片 串口/TTL 下载连接示意图

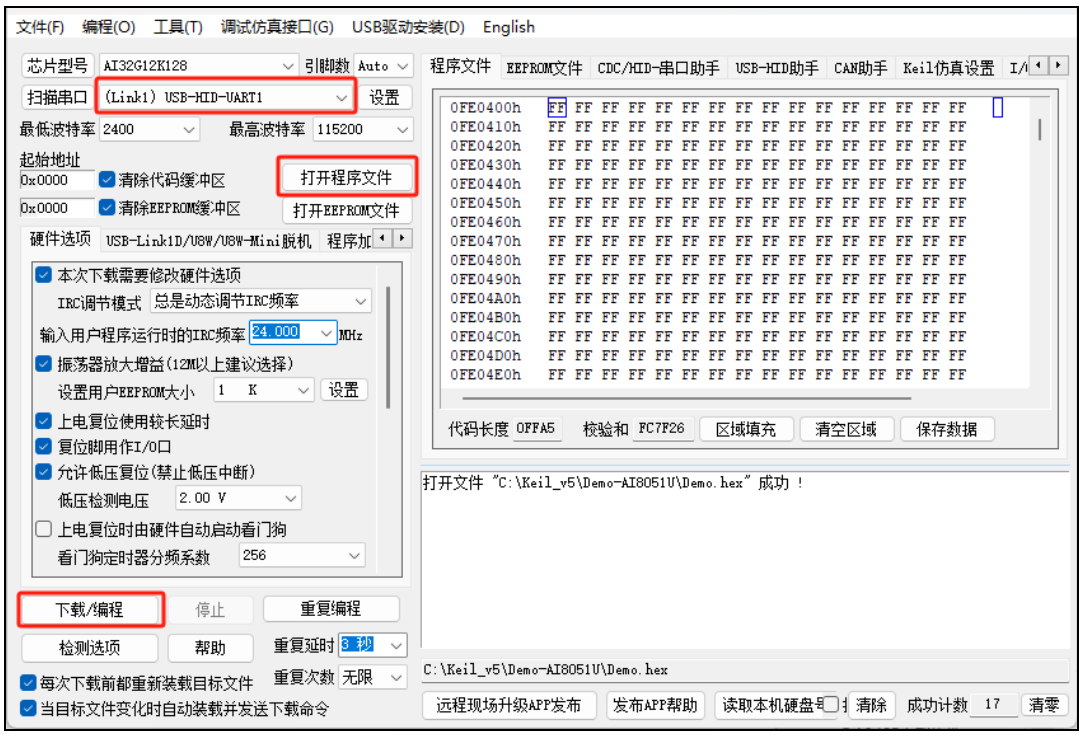


2、使用 USB-Link1D 工具对 Ai15 和 Ai8 系列进行串口仿真

工具的 S-Vcc、S-P3.0、S-P3.1、GND 分别与目标单片机的 M-Vcc、P3.0（RxD）、P3.1（TxD）、GND 相连接，然后 Keil 仿真设置中选择 USB-CDC1 所对应的串口号，然后参考 AI15/AI8 系列数据手册中的直接串口仿真章节中仿真的步骤和设置，即可进行串口仿真

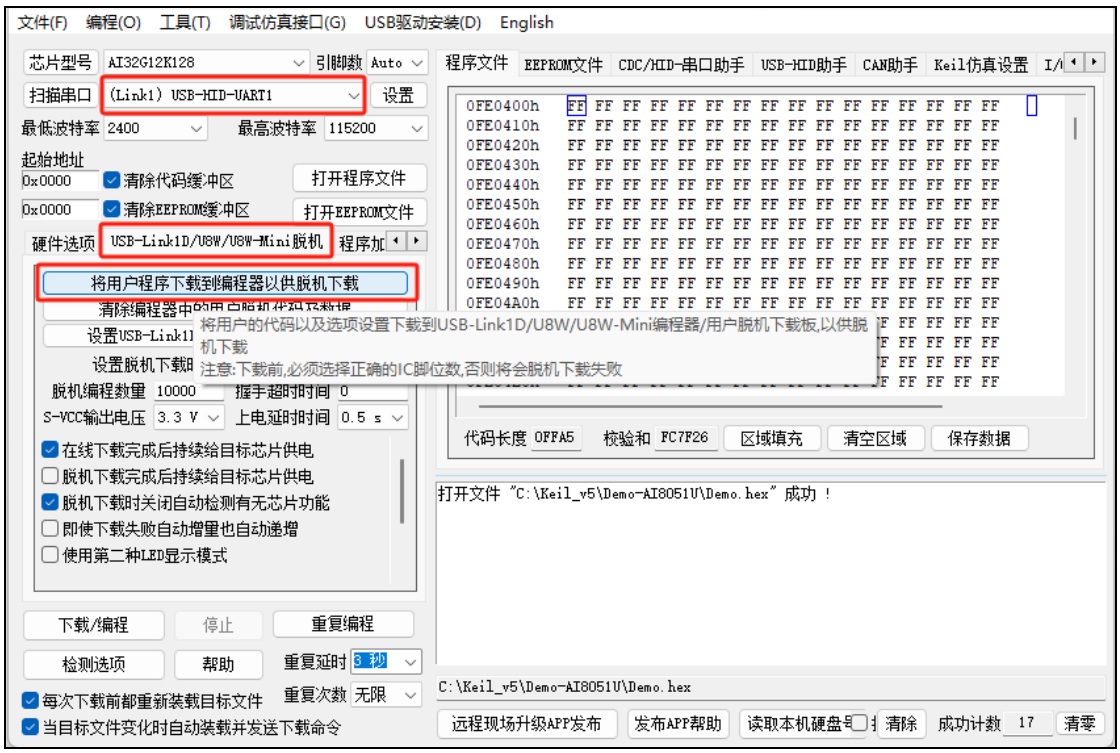
3、使用 USB-Link1D 工具对全系列单片机进行 ISP 在线下载

工具的 S-Vcc、S-P3.0、S-P3.1、GND 分别与目标单片机的 M-Vcc、P3.0（RxD）、P3.1（TxD）、GND 相连接，在 ISP 下载软件中的串口号选择“（Link1）USB-HID-UART1”，打开程序文件以及设置相关硬件选项，然后点击“下载/编程”按钮即可进行 ISP 在线下载



4、使用 USB-Link1D 工具对全系列单片机进行 ISP 脱机下载

在 ISP 下载软件中的串口号选择“（Link1）USB-HID-UART1”，打开程序文件以及设置相关硬件选项，后点击“U8W/Link1 脱机”页面中的“将用户程序下载到编程器以供脱机下载”按钮，将用户代码和相关设置下载到 USB Link1 工具上的存储器中。



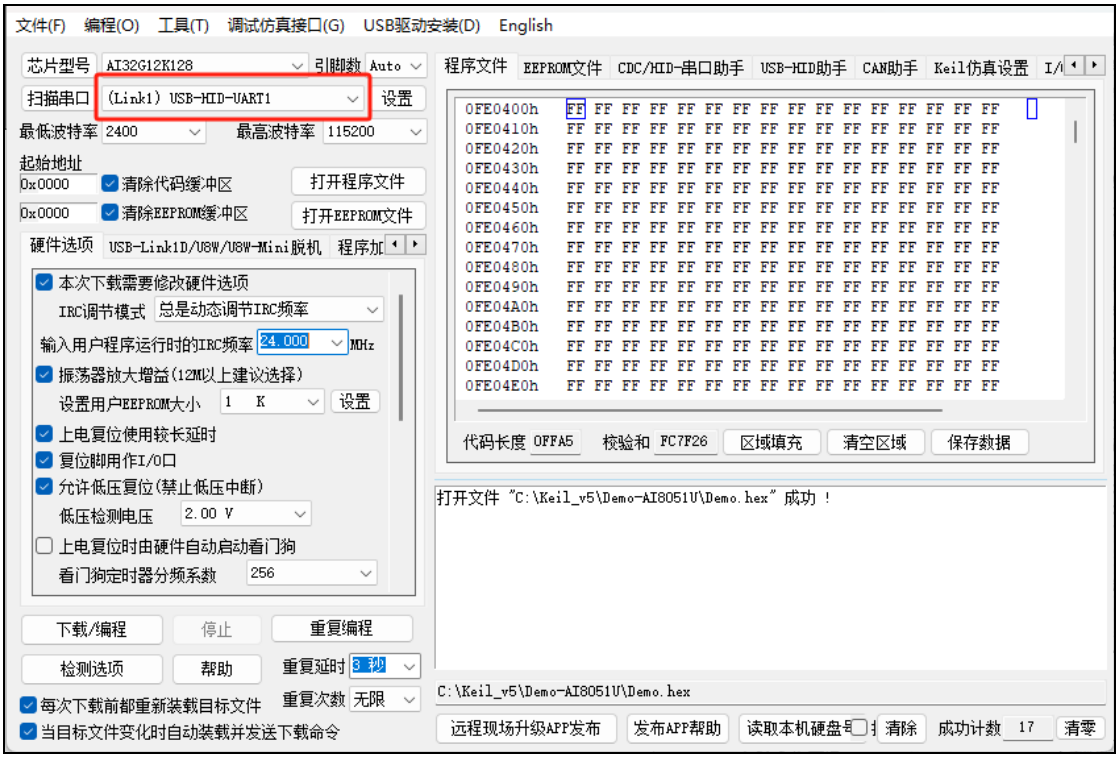
将工具的 S-Vcc、S-P3.0、S-P3.1、GND 分别与目标单片机的 M-Vcc、P3.0（RxD）、P3.1（TxD）、GND 相连接，然后按下工具上的“Key1”按键即可对目标芯片进行脱机下载（即不需要 PC 端的控制，独立进行 ISP 下载）

5、USB-Link1D 工具当作通用 USB 专串口工具使用

USB-Link1D 工具提供了两个 USB-CDC 串口，可作为通用 USB 专串口工具使用，由于第一个串口 CDC1 与硬件仿真、ISP 下载共用 S-P3.0 和 S-P3.1 端口，而第二个串口 CDC2 是独立串口，所以建议 S-P3.0 和 S-P3.1 作为仿真和 ISP 下载使用，当需要使用通用 USB 专串口工具时，使用 S-TxD 和 S-RxD 所对应的 CDC2。（注：在没有使用冲突的情况下，CDC1 和 CDC2 均可各自独立的当作通用 USB 专串口工具使用）

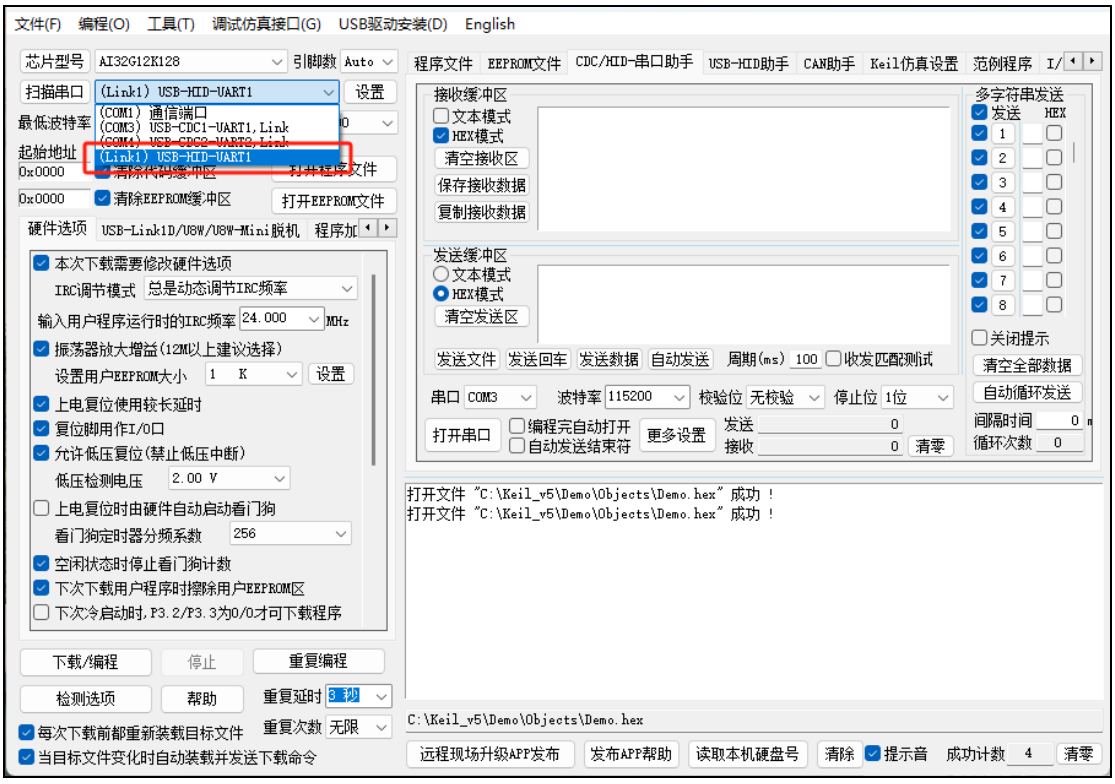
4.4.4 USB-Link1D 插上电脑并正常识别到后的显示

USB-Link1D 工具在出厂时，主控芯片内已烧录了 USB-Link1D 的控制程序。正常情况下，工具连接到电脑后，在 ISP 下载软件中会立即识别出“（Link1）USB-HID-UART1”，如下图所示



正确识别后，即可使用 USB-Link1D 进行在线 ISP 下载或者脱机 ISP 下载。

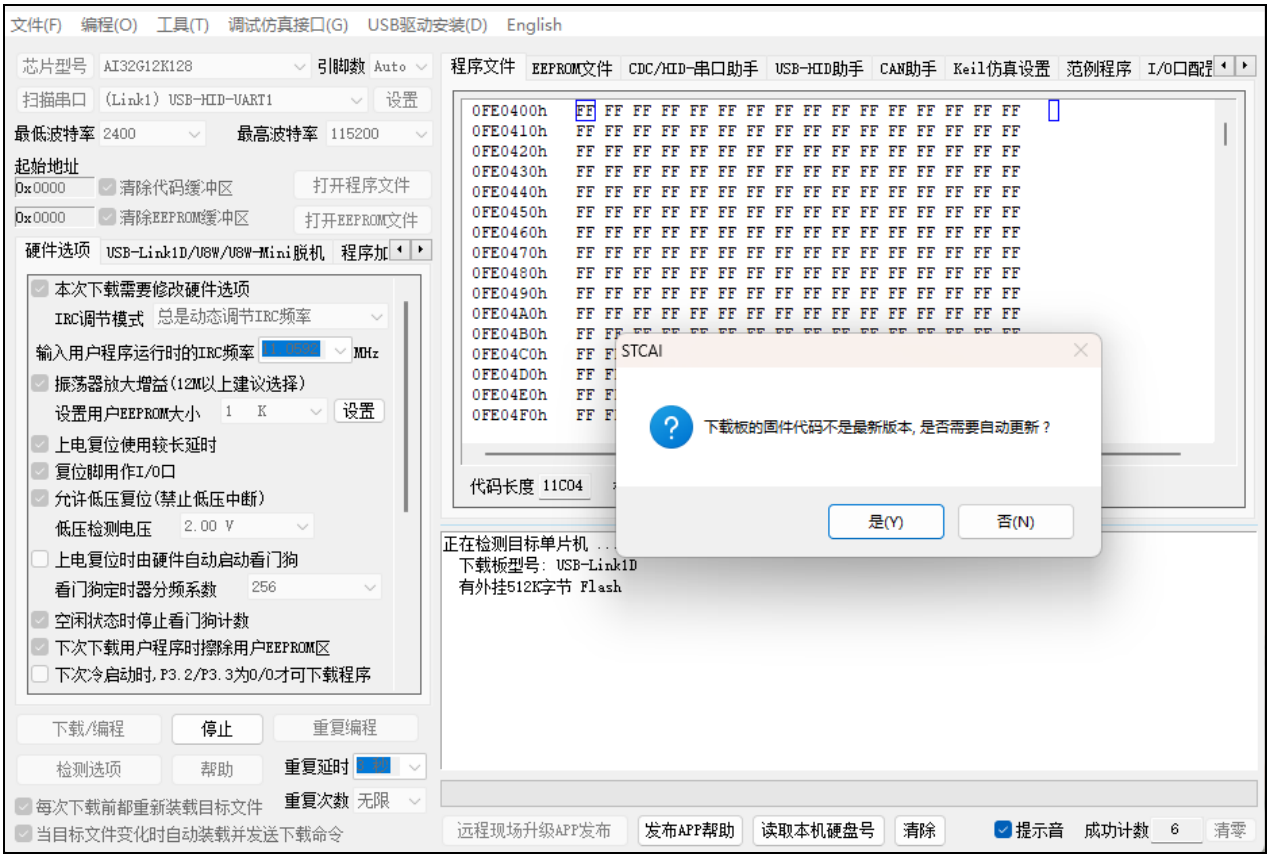
在驱动安装成功后，还会自动识别出两个 USB-CDC 串口，如下图所示：



可以当作通用 USB 转双串口工具使用。

4.5　如电脑端新版本 ISP 软件提示 USB-Link1D 工具固件需升级

当使用工具进行 ISP 下载时，如新版本软件弹出如下画面，表示工具的固件需要升级

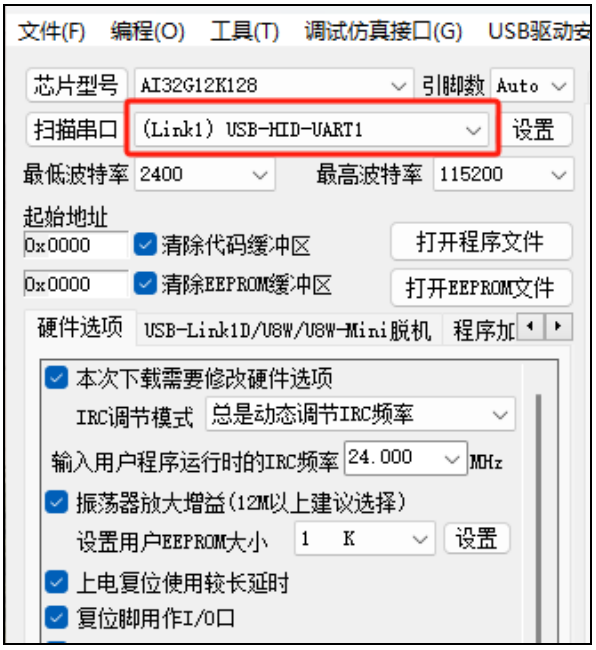


点击“是”按钮，工具便会自动开始升级。

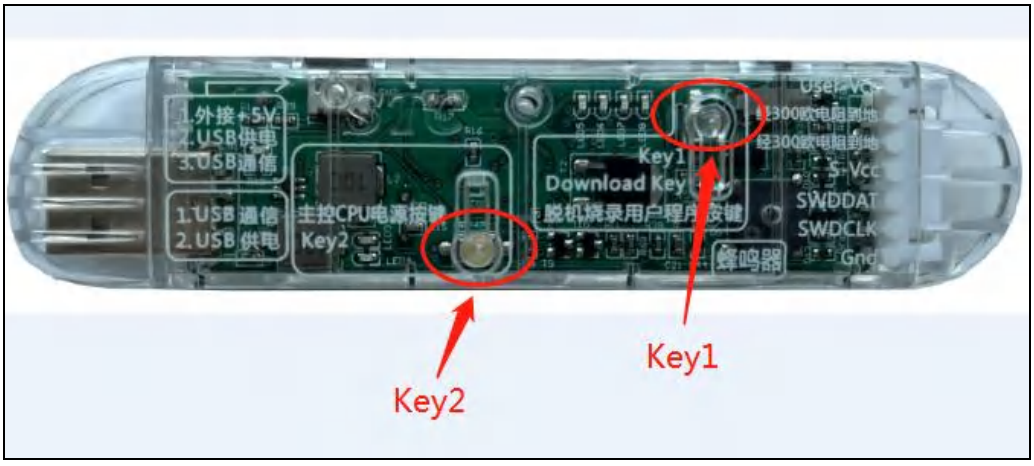
4.6　重新制作 USB-Link1D 主控芯片，主动升级或遇到异常时

USB-Link1D 工具在出厂时，主控芯片已烧录入主控程序，所以用户拿到工具后不需要自己手动再次制作 USB-Link1D 工具的主控芯片。只有在用户将工具的主控芯片更换为一颗全新的芯片才需要进行下面的步骤。

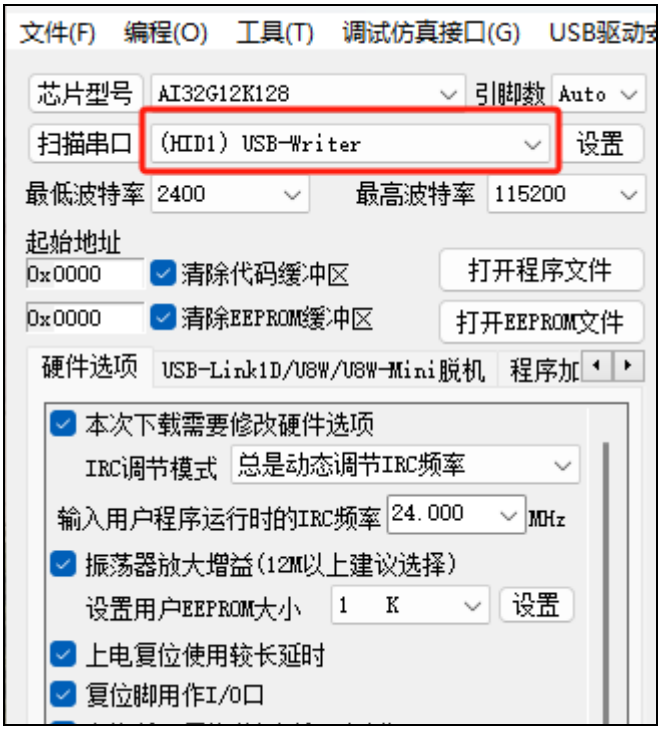
1、将 USB-Link1D 工具插入电脑的 USB 口：



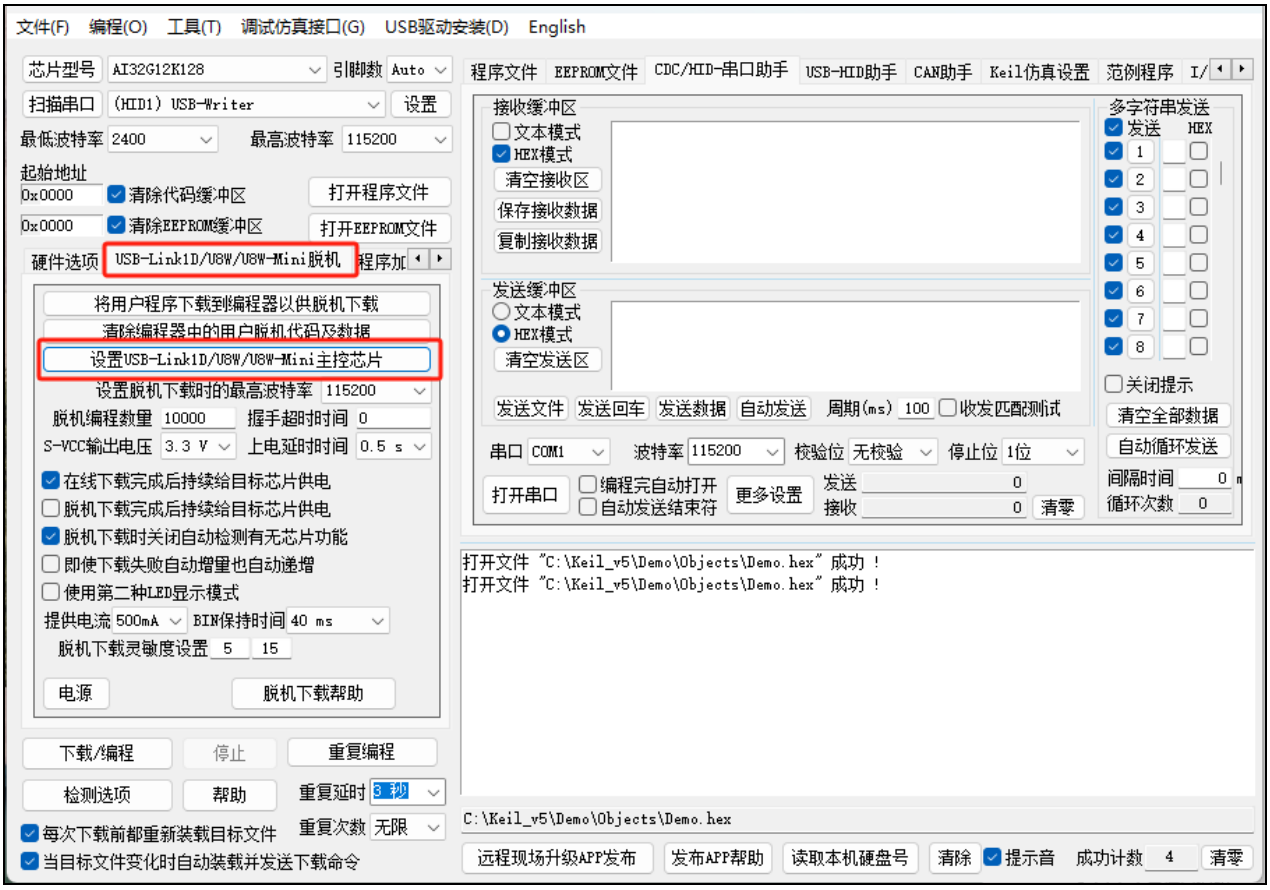
如果 AIapp-ISP 下载软件的端口列表中没有如上图显示出“（Link1）USB-HID-UART1”设备，则说明工具的主控芯片为空片，需要继续接下来的步骤



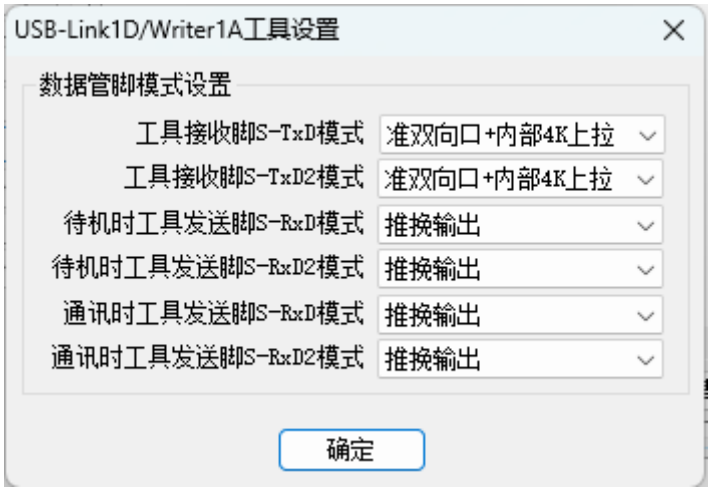
2、在工具已通过 USB 线和电脑相连的情况下，按下工具的“Key1”按键并暂时不松开，再按下“Key2”按键停电，然后松开“Key2”按键上电。等到 AIapp-ISP 下载软件的端口列表中显示出“(HID1) USB-Writer”设备（如下图）时，就可以松开“Key1”按键（注：工具上的“Key1”为主控芯片的 P3.2 口，“Key2”为工具主控芯片的电源键）



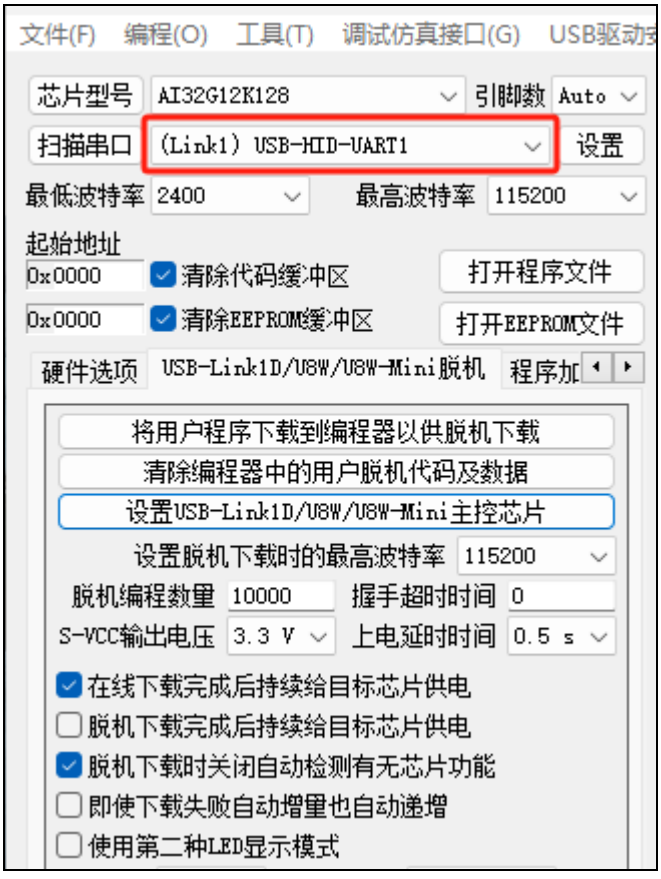
3、点击 AIapp-ISP 下载软件中“USB-Link1D/U8W/U8W-Mini 脱机”页面的“设置 USB-Link1D/U8W/U8W-Mini 主控芯片”按钮



4、会弹出如下 Link1D 工具设置窗口，用户可以根据实际需要进行设置，若无特殊需要可保持默认选项



5、接下来 ISP 软件会将主控芯片的主控程序烧录进 USB-Link1D 工具的主控芯片。烧录完成后，建议将工具的 USB 线重新插拔一次（特别是对一颗全新的芯片第一次制作主控芯片时必须重新插拔一次）。主控芯片制作完成后，拔掉连接电脑的 USB 线，再将 USB 线重新插入电脑的 USB 口后，AIapp-ISP 下载软件的端口列表框中会显示出“（Link1）USB-HID-UART1”设备（如下图），说明工具的主控芯片已成功制作。



上面的拔插 USB 线，可以用按下 Key2 电源按键停电，松开 Key2 电源按键上电来代替。

4.7 ISP 下载相关硬件选项的说明

硬件选项	选项何时生效
☑ 选择使用内部IRC时钟(不选为外部时钟)	需要重新上电才生效
输入用户程序运行时的IRC频率 11.0592 MHz	动态调整，立即生效
☑ 振荡器放大增益(12M以上建议选择)	需要重新上电才生效
☑ 使用快速下载模式	只与本次下载有关
设置用户EEPROM大小 0.5 K	需要重新上电才生效
☐ 下次冷启动时,P3.2/P3.3为0/0才可下载程序	下次下载时有效
☑ 上电复位使用较长延时	需要重新上电才生效
☑ 复位脚用作I/O口	需要重新上电才生效
☑ 允许低压复位(禁止低压中断)	需要重新上电才生效
低压检测电压 2.20 V	需要重新上电才生效
☑ 低压时禁止EEPROM操作	需要重新上电才生效
上电复位时由硬件自动启动看门狗 看门狗定时器分频系数 256 ☑ 空闲状态时停止看门狗计数	需要重新上电才生效
☑ 下次下载用户程序时擦除用户EEPROM区	下次下载时有效
☐ 在程序区的结束处添加重要测试参数	每次下载时一并写入

需要重新上电才生效：选项修改后，目标芯片需要断电一次（停电），重新再上电，新的设置才生效

动态调整，立即生效：本次 ISP 下载有效

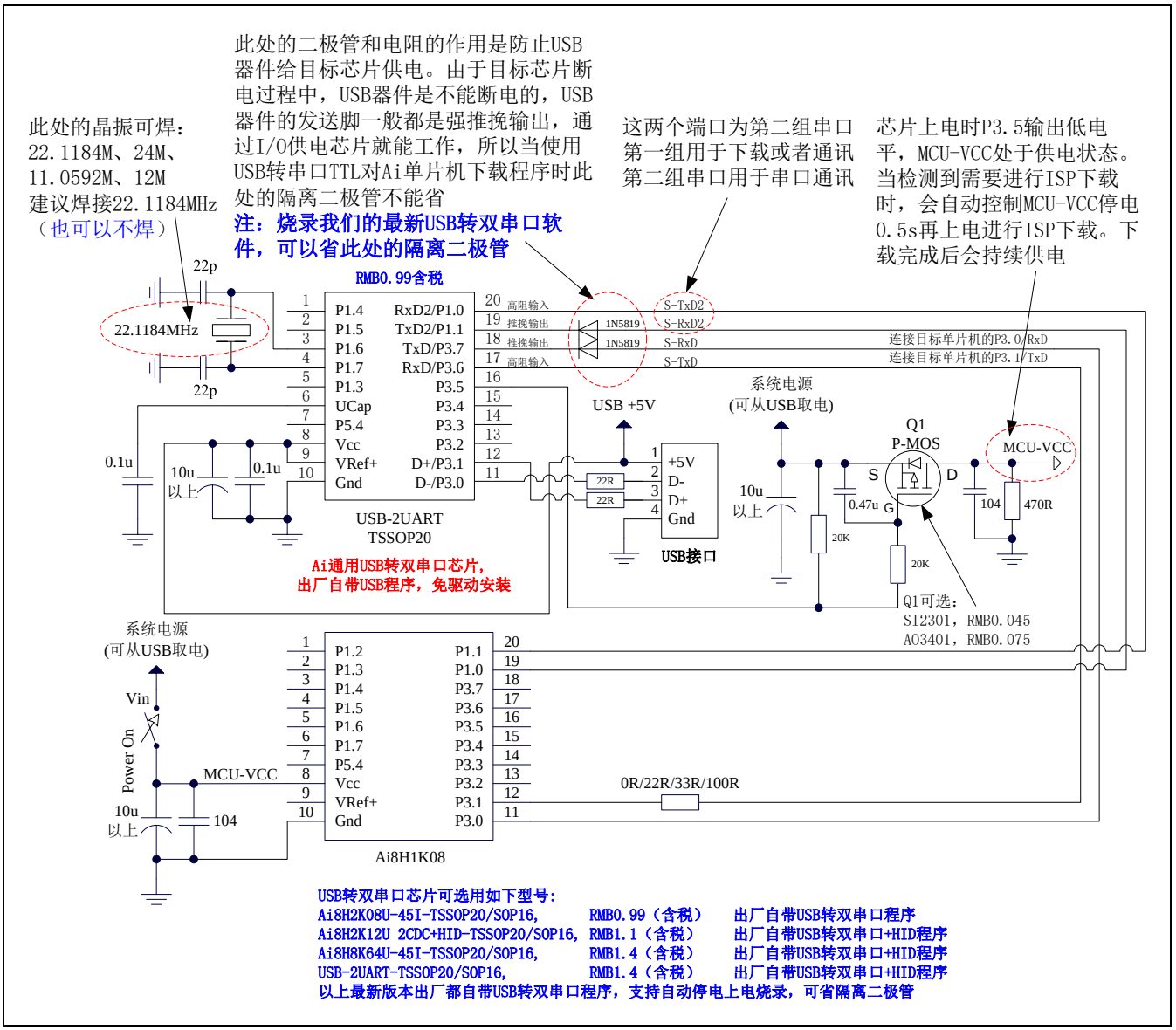
只与本次下载有关：此选项只与本次 ISP 下载有关，不影响下一次下载

下次下载时有效：选项修改后，下次下载时才生效，修改对本次 ISP 下载无效

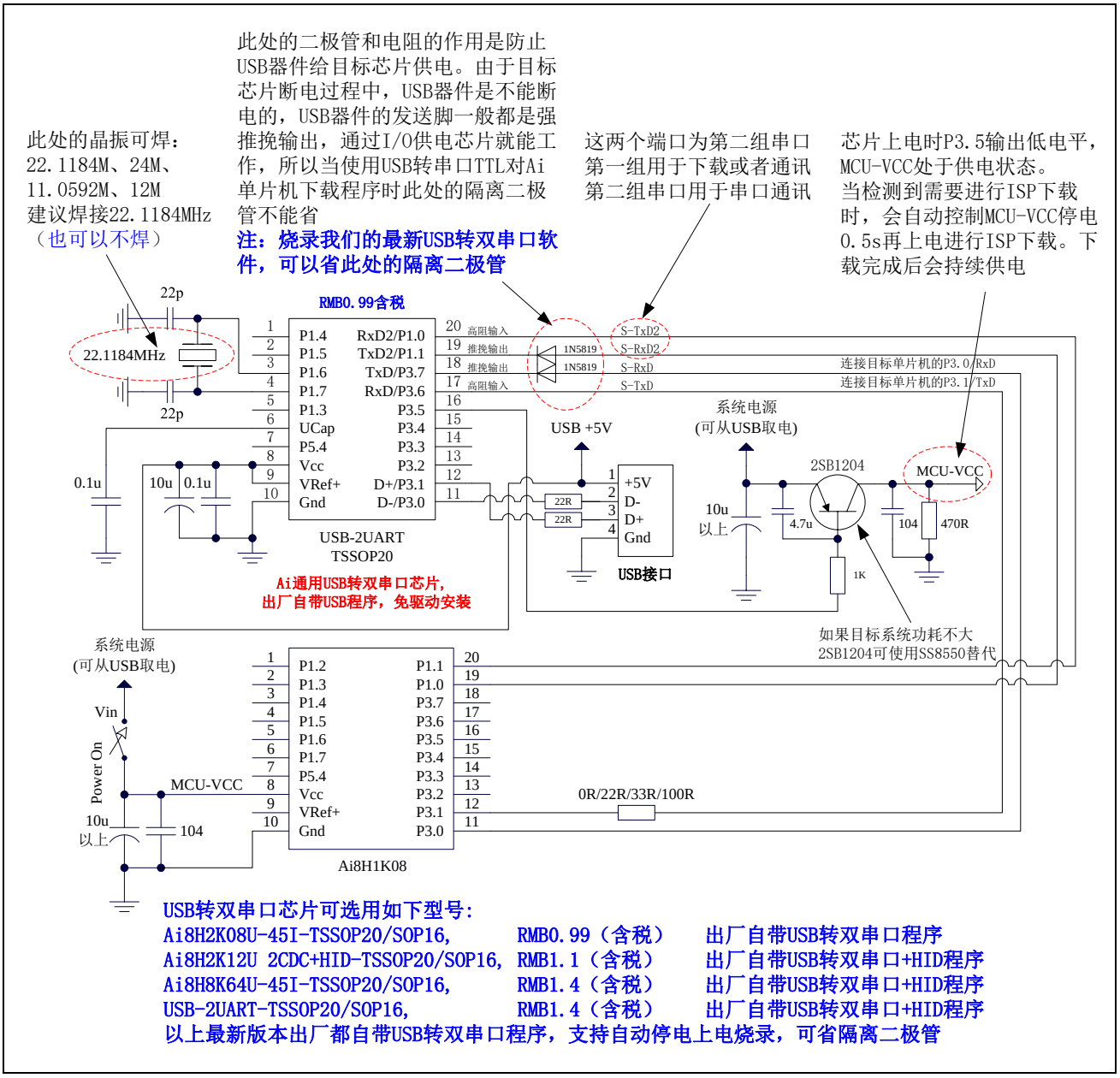
每次下载时一并写入：选择此选项后，在本次下载时将附加的数据一并写入，与下次下载无关

4.8 通用 USB 转双串口芯片：USB-2UART, TSSOP20/SOP16

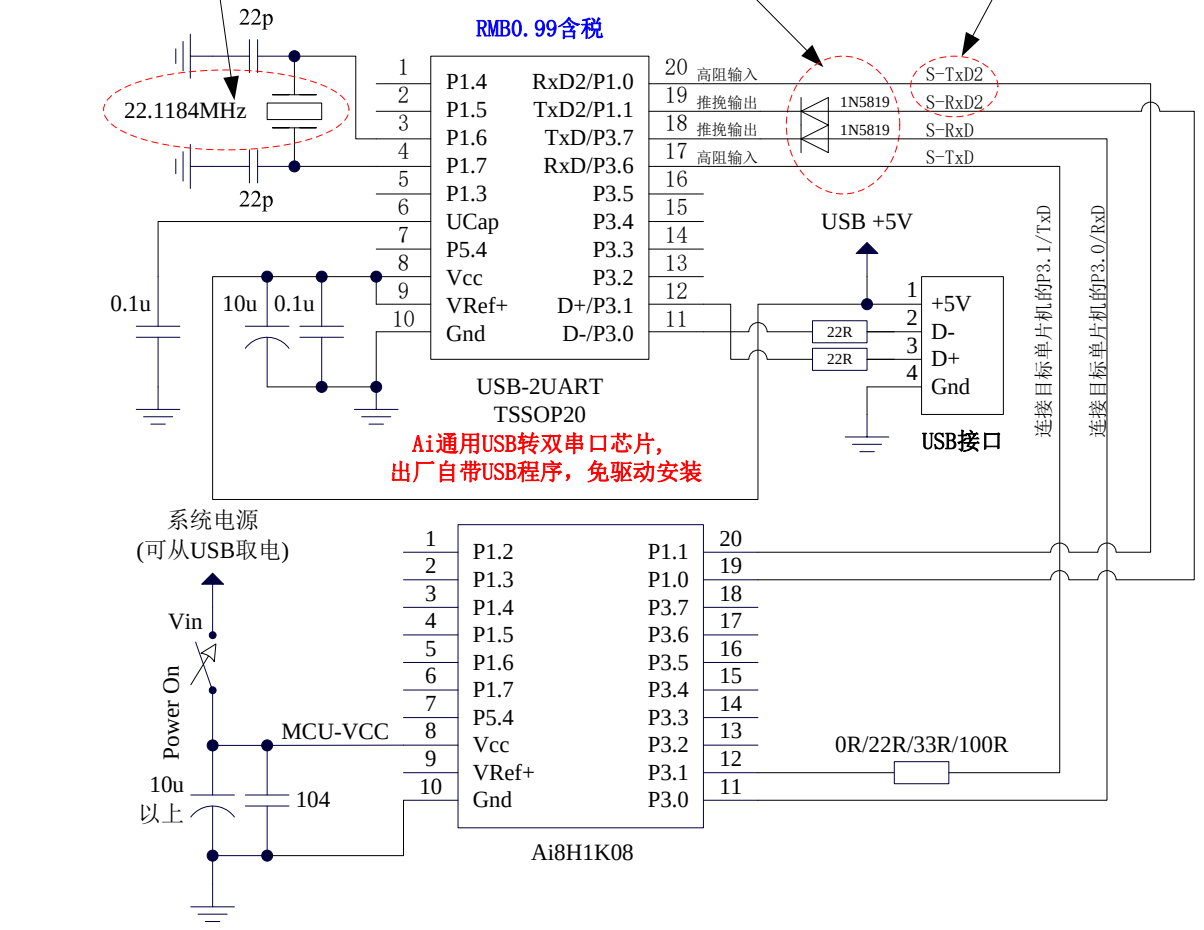
4.8.1 USB 转双串口芯片 USB-2UART-45I-TSSOP20，自动停电上电（MOS 管）



4.8.2 USB 转双串口芯片 USB-2UART-45I-TSSOP20，自动停电上电（三极管）



4.8.3 USB 转双串口芯片 USB-2UART-45I-TSSOP20，手动停电上电

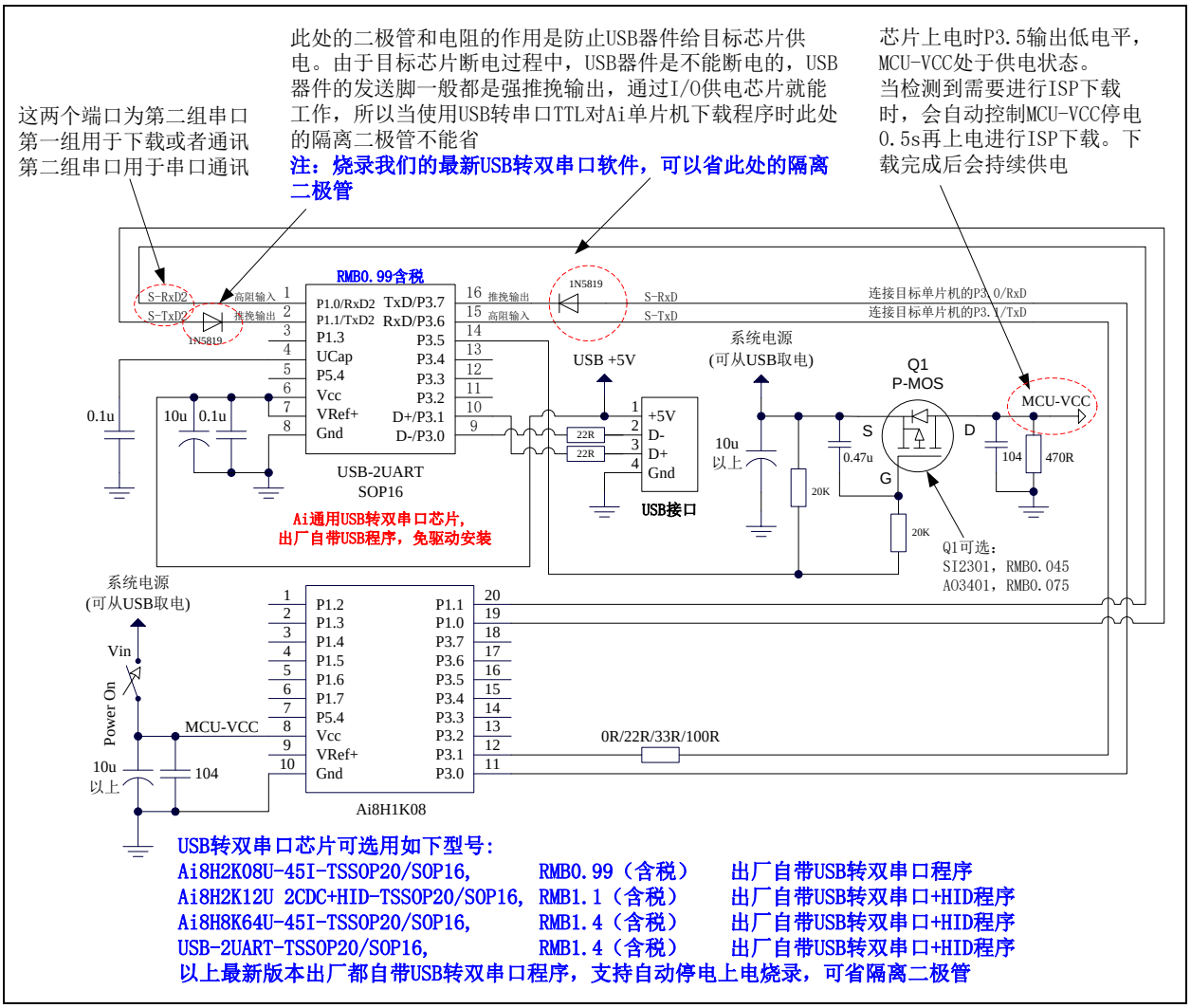


USB转双串口芯片可选用如下型号：

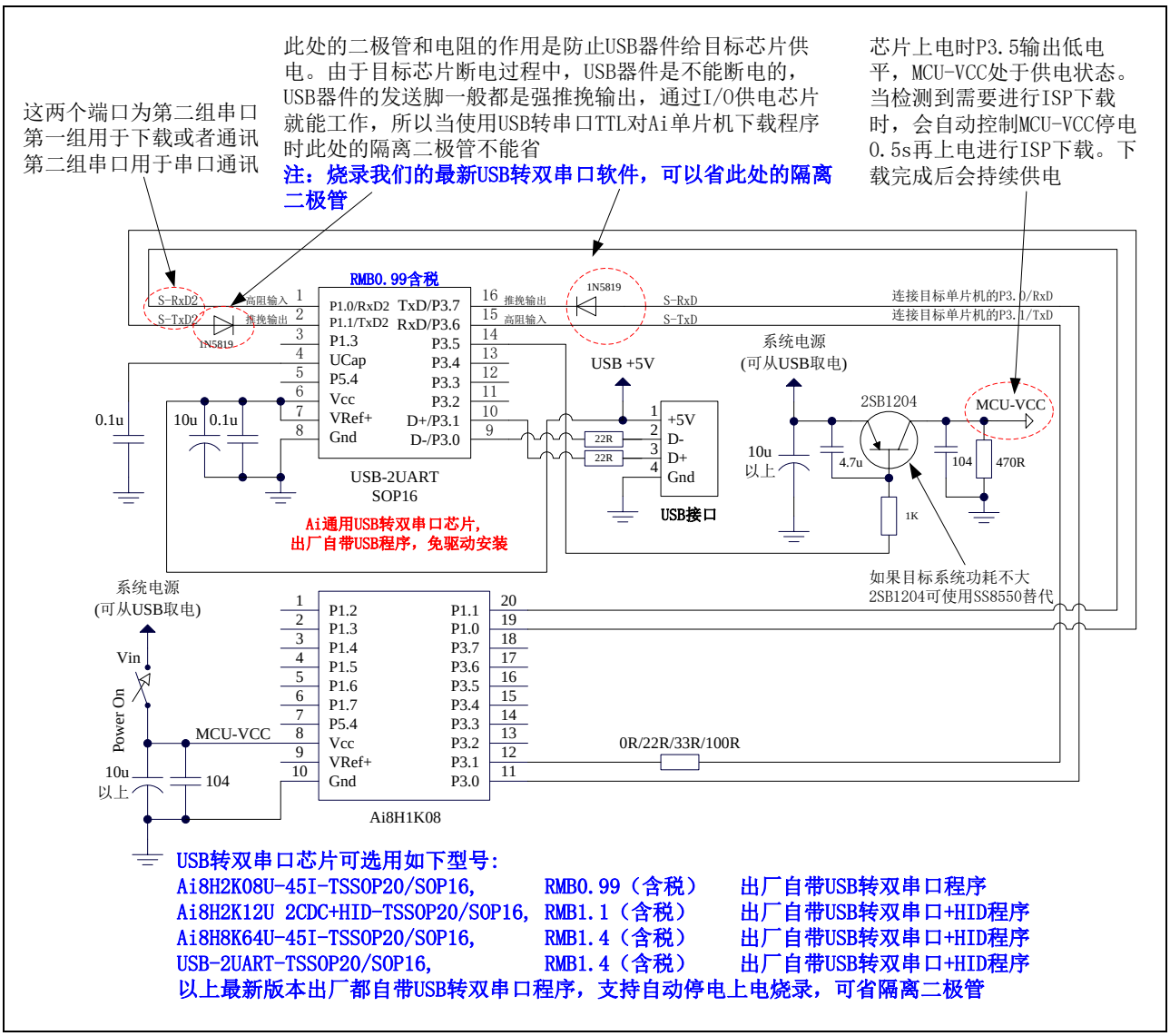
Ai8H2K08U-45I-TSSOP20/SOP16,	RMB0.99（含税）	出厂自带USB转双串口程序
Ai8H2K12U 2CDC+HID-TSSOP20/SOP16,	RMB1.1（含税）	出厂自带USB转双串口+HID程序
Ai8H8K64U-45I-TSSOP20/SOP16,	RMB1.4（含税）	出厂自带USB转双串口+HID程序
USB-2UART-TSSOP20/SOP16,	RMB1.4（含税）	出厂自带USB转双串口+HID程序

以上最新版本出厂都自带USB转双串口程序，支持自动停电上电烧录，可省隔离二极管

4.8.4 USB 转双串口芯片 USB-2UART-45I- SOP16，自动停电上电（MOS 管）



4.8.5 USB 转双串口芯片 USB-2UART-45I- SOP16，自动停电上电（三极管）



USB转双串口芯片可选用如下型号：

Ai8H2K08U-45I-TSSOP20/SOP16,	RMB0.99（含税）	出厂自带USB转双串口程序
Ai8H2K12U 2CDC+HID-TSSOP20/SOP16,	RMB1.1（含税）	出厂自带USB转双串口+HID程序
Ai8H8K64U-45I-TSSOP20/SOP16,	RMB1.4（含税）	出厂自带USB转双串口+HID程序
USB-2UART-TSSOP20/SOP16,	RMB1.4（含税）	出厂自带USB转双串口+HID程序

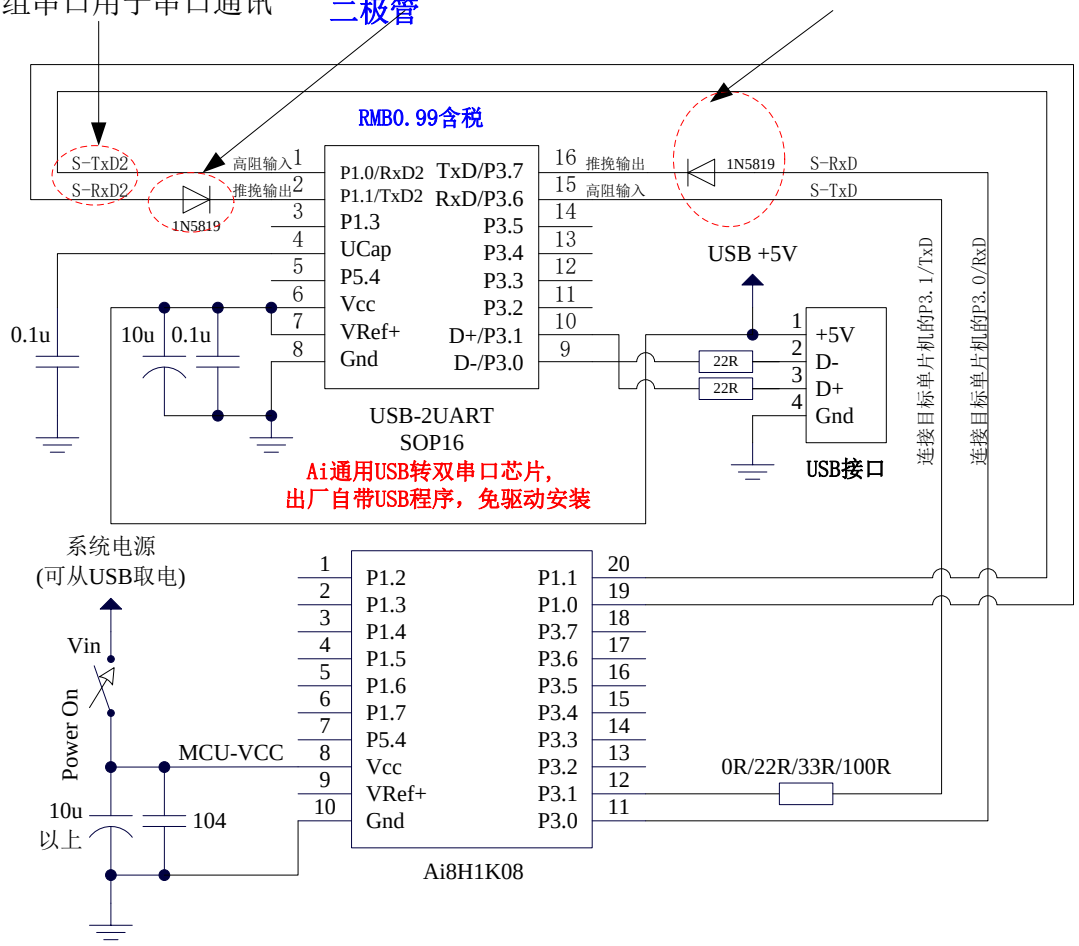
以上最新版本出厂都自带USB转双串口程序，支持自动停电上电烧录，可省隔离二极管

4.8.6 USB转双串口芯片USB-2UART-45I-SOP16，手动停电上电

这两个端口为第二组串口
第一组用于下载或者通讯
第二组串口用于串口通讯

此处的二极管和电阻的作用是防止USB器件给目标芯片供电。由于目标芯片断电过程中，USB器件是不能断电的，USB器件的发送脚一般都是强推挽输出，通过I/O供电芯片就能工作，所以当使用USB转串口TTL对Ai单片机下载程序时此处的隔离二极管不能省

注：烧录我们的最新USB转双串口软件，可以省此处的隔离二极管



USB转双串口芯片可选用如下型号:

Ai8H2K08U-45I-TSSOP20/SOP16,	RMB0.99（含税）	出厂自带USB转双串口程序
Ai8H2K12U 2CDC+HID-TSSOP20/SOP16,	RMB1.1（含税）	出厂自带USB转双串口+HID程序
Ai8H8K64U-45I-TSSOP20/SOP16,	RMB1.4（含税）	出厂自带USB转双串口+HID程序
USB-2UART-TSSOP20/SOP16,	RMB1.4（含税）	出厂自带USB转双串口+HID程序

以上最新版本出厂都自带USB转双串口程序，支持自动停电上电烧录，可省隔离二极管

5　Ai8052U 系列选型简介、特性、价格、管脚图

5.1　Ai8052U-LQFP100 总体介绍，USB 下载，烧录/仿真 线路图

5.1.1　特性及价格

- 选型价格（不需要外部晶振、不需要外部复位，两组 12 位 ADC，15+15 通道）

单片机型号	工作电压（V）	Flash 程序存储器 10 万次 字节	edata 内部扩展 DATA RAM 可做堆栈或变量 字节	xdata 内部大容量扩展 SRAM 可做变量 字节	EEPROM 10 万次 字节	I/O 口最多数量	传统 I/O 中断（INT0/INT1/INT2/INT3/INT4）并可掉电唤醒	所有 I/O 口均支持高电平 低电平 上升沿 下降沿中断，可掉电唤醒	DMA 8080/6800 接口 TFT 彩屏模块驱动(8 位和 16 位)	RTC 实时时钟 年/月/日/时/分/秒/星期 对内部时钟追频 掉电唤醒	DMA USART 同步/异步串口并可掉电唤醒 额外支持 8 组 SPI/LIN	全速 USB, 支持 USB-CDC, USB-HID	DMA I2S 音频总线 掉电唤醒（全双工录音 + 放音）	DMA SPI 并可掉电唤醒	DMA QSPI	DMA I^2C 并可掉电唤醒	DMA 汽车电子通讯 CAN-FD 总线并可掉电唤醒	汽车电子通讯 LIN 总线	TFPU (230MHz)硬件浮点 / 硬件三角 / 反三角函数运算器	DSP32 （74MHz），硬件 32 位/64 位 DSP 指令	定时器/计数器（T0～T11, T17/T18 外部管脚也可掉电唤醒）	230MHz，16 位高级 PWM 定时器 互补对称死区控制	掉电唤醒专用定时器	DMA 支持两个独立的 12 位 ADC，每个 16 通道，共 32 通道	DMA 支持两个独立的 12 位高速 DAC	比较器（6P6N 模式，6 个正极输入，6 个负极输入）	运算放大器（OPA）	内部低压检测中断并可掉电唤醒	看门狗 复位定时器	内部高可靠复位（可选复位门槛电压）	内部高精准时钟（74MHz 以下可调），满足串口通信，追频	可对外输出时钟及复位	程序加密后传输（防拦截），可设置下次更新程序需口令	支持 RS485 下载	本身就可在线仿真，支持硬件 USB 直接下载和硬件 USB 仿真	价格及封装 LQFP100	LQFP64	LQFP48	LQFP44	供货信息
Ai8052U-144K256	1.9-5.5	256K	16K	132K	IAP	96	有	有	有	有	8	有	2	3+8	有	2	2	2+8	有	有	14	24	有	2 组	2 组	4 组	4 组	有	有	4 级	有	是	是	是	是					

- **内核**
  - ✓ 超高速 32 位 8051 内核（1T），比传统 8051 约快 70 倍以上
  - ✓ 4 级中断优先级
  - ✓ 支持在线仿真
- **工作电压**
  - ✓ 1.9V～5.5V（当工作温度低于-40℃时，工作电压不得低于 3.0V）
- **工作温度**
  - ✓ -20℃~65℃（内部高速 IRC 温漂-0.76%~+0.98%）
  - ✓ -40℃~85℃（内部高速 IRC 温漂±1.3%）
  - ✓ -40℃~125℃（内部高速 IRC 温漂±3%，当温度高于 85℃时请使用外部 35MHz 及以下的晶振）
- **Flash 存储器**
  - ✓ 最大 256K 字节 FLASH 程序存储器（ROM），用于存储用户代码
  - ✓ 支持用户可配置部分 FLASH 作为 DATA FLASH/EEPROM 使用，512 字节单页擦除，
    典型单个扇区擦除次数可达 10 万次，用户保守可当 2 万次使用
  - ✓ 支持硬件 USB 直接下载和普通串口下载
  - ✓ 支持硬件 SWD 实时仿真，P3.0/P3.1（需 USB-Link1D 工具）
  - ✓ **Flash 数据保持时间：100 年@25℃，20 年@105℃，10 年@125℃。由于汽车及单片机的平均温度不可能长时间大于 105℃，从严考虑，Flash 数据保持时间可达 20 年以上，不保证 100 年。**
- **SRAM，共 148K 字节**

车规启航
扫码去微信小商城

- ✓ 16K 字节内部 SRAM（edata）
- ✓ 128K 字节内部扩展 RAM（内部 xdata）
- ✓ **4K 字节高端扩展 RAM（可映射到 80H:0000H～80H:0FFFH 区执行用户程序）**

➢ **时钟控制**

- ✓ 内部高精度、高稳定的高速 IRC（ISP 编程时可进行上下调整）
  - ⌖ 误差±0.3%（常温下 25℃）
  - ⌖ -1.35%～+1.30%温漂（全温度范围，-40℃～85℃）
  - ⌖ -0.76%～+0.98%温漂（温度范围，-20℃～65℃）
- ✓ 内部 32KHz 低速 IRC（为了低功耗，省去了温度补偿和电压补偿电路，误差较大）
- ✓ 外部晶振（4MHz～80MHz）和外部时钟，有专门的外部时钟干扰内部电路，可软件启动
- ✓ **内部两组独立的 500MHz 的 PLL，可输出 230MHz 的时钟提供给 TFPU、PWM、I2S、SPI 等高速外设**

用户可自由选择上面的 4 种时钟源

（芯片上电工作过程：上电复位/复位脚复位/看门狗复位/低压检测复位时，芯片默认从 ISP 系统程序开始执行代码，此时固定使用内部 24MHz 的高速 IRC 时钟，当需要下载用户程序且下载完成后复位到用户程序区或者不需要下载直接复位到用户程序区时，默认会使用上次用户下载时所调节的高速 IRC 时钟，如果用户程序需要使用外部高速晶振、外部 32.768KHz 晶振或者内部 30KHz 低速 IRC，则需要用户软件先启动相应的时钟，然后通过设置 CLKSEL 寄存器进行切换）

➢ **复位**

- ✓ 硬件复位
  - ⌖ 上电复位，复位电压值为 1.7V～1.9V。（在芯片未使能低压复位功能时有效）
  - ⌖ 复位脚复位，出厂时 P5.4 默认为 I/O 口，ISP 下载时可将 P5.4 管脚设置为复位脚（注意：当设置 P5.4 管脚为复位脚时，复位电平为低电平）
  - ⌖ 看门狗溢出复位
  - ⌖ 低压检测复位，提供 4 级低压检测电压：2.0V、2.4V、2.7V、3.0V。
- ✓ 软件复位
  - ⌖ 软件方式写复位触发寄存器

➢ **中断**

- ✓ 中断源：INT0、INT1、INT2、INT3、INT4、定时器 0、定时器 1、定时器 2、定时器 3、定时器 4、定时器 5、定时器 6、定时器 7、定时器 8、定时器 9、定时器 10、定时器 11、定时器 17、定时器 18、USART1、USART2、USART3、USART4、USART5、USART6、USART7、USART8、ADCA 模数转换、ADCB 模数转换、DACA 数模转换、DACB 数模转换、LVD 低压检测、SPI1、SPI2、SPI3、I^2C1、I^2C2、比较器 1、比较器 2、比较器 3、比较器 4、PWMA、PWMB、PWMC、PWMD、PWME、PWMF、USB、TFT 彩屏接口中断、RTC 实时时钟、所有的 I/O 中断（8 组）、所有的 USART 串口的 DMA 接收和发送中断（8 组）、I2C 的 DMA 接收和发送中断（两组）、I2S 的 DMA 接收和发送中断（两组）、SPI 的 DMA 中断（3 组）、QSPI 的 DMA 中断、ADC 的 DMA 中断（两组）、DAC 的 DMA 中断（两组）、LCD 驱动的 DMA 中断、、CAN-FD 的 DMA 中断（两组）、PWM 的 DMA 中断（PWMA/PWMC/PWME）、存储器到存储器的 DMA 中断以及**两组外设到外设的 DMA 中断**。
- ✓ 提供 4 级中断优先级

➢ **数字外设**

- ✓ 14 个 16 位定时器：定时器 0、定时器 1、定时器 2、定时器 3、定时器 4、定时器 5、定时器 6、定时器 7、定时器 8、定时器 9、定时器 10、定时器 11、定时器 17、定时器 18，其中定时器 0 的模式 3 具有 NMI（不可屏蔽中断）功能，定时器 0 和定时器 1 的模式 0 为 16 位自动重载模式。定时器 11：可选时钟源。**定时器 17 和定时器 18 可同步触发两组 ADC 或者同步触发两组 DAC**。
- ✓ 8 个高速同步/异步串口：串口 1（USART1）、串口 2（USART2）、串口 3（USART3）、串口 4（USART4）、

串口 5（USART5）、串口 6（USART6）、串口 7（USART7）、串口 8（USART8），波特率时钟源最快可为 FOSC/4。支持同步串口模式、异步串口模式、SPI 模式、LIN 模式、红外模式（IrDA）、智能卡模式（ISO7816）

- ✓ 24 通道高级 PWM，3+3 组高级 PWM，每组均可实现 4 通道 PWM。PWMA/PWMC/PWME 均可实现 4 对互补对称带死区控制的 PWM，PWMB/PWMD/PWMF 均可实现 4 通道 PWM 输出，并支持外部异常检测功能。PWM 时钟源的输入**速度可支持到 230MHz**
- ✓ SPI：3+8 组硬件 SPI（3 组独立 SPI，8 组 USART 的 SPI 模式）支持主机模式和从机模式以及主机/从机自动切换（注：11 组 SPI 均支持 DMA，11 组 SPI 的 MISO 和 MOSI 脚均可软件设置进行功能交换），SPI 的时钟源支持 **230MHz**
- ✓ QSPI：支持单线、双线和四线模式
- ✓ I^2C：两组独立的硬件 I^2C，均支持主机模式和从机模式
- ✓ ICE：支持硬件仿真，支持串口和 USB 口直接仿真，还支持 SWD 硬件仿真
- ✓ RTC：支持年、月、日、星期、时、分、秒、次秒（1/128 秒），并支持时钟中断和一组闹钟。芯片复位时 RTC 的时间计数值不复位。
- ✓ USB：USB2.0/USB1.1 兼容全速 USB，**16 个双向端点**，支持 4 种端点传输模式（控制传输、中断传输、批量传输和同步传输），端点拥有 64/128/256/512/1024 字节的 FIFO 缓冲区，不同的端点的 FIFO 缓冲区大小不等。
- ✓ CAN：两个独立的 CAN-FD 2.0 控制单元。CAN-FD 的 STB 脚可软件设置是否使能，若不使能 STB 脚可当作 GPIO 使用。
- ✓ LIN：2+8 组硬件 LIN（两组组独立 LIN，8 组 USART 的 LIN 模式），一个独立的 LIN 控制单元（支持 1.3 和 2.1 版本）
- ✓ I2S：两路独立的 I2S 音频总线，**可支持全双工方式的录音和放音**
- ✓ DSP32：硬件 32 位/64 位 DSP 运算指令，**速度可达 74MHz**
- ✓ TFPU：单精度浮点运算器（支持浮点加、减、乘、除以及正弦、余弦、正切和反正切等运算），**速度可达 230MHz**
- ✓ I/O 口中断：所有的 I/O 均支持中断，每组 I/O 中断有独立的中断入口地址，所有的 I/O 中断可支持 4 种中断模式：高电平中断、低电平中断、上升沿中断、下降沿中断。I/O 口中断可以进行掉电唤醒，且有 4 级中断优先级。
- ✓ LCD/TFT 彩屏驱动模块：支持 8080 和 6800 两种接口以及 8 位和 16 位数据宽度。对 16 位数据支持大小端数据格式选择。
- ✓ DMA：支持 SPI（3 组）、QSPI、I2C（两组）、I2S（两组）、串口（8 组）、PWM（3 组，PWMA/PWMC/PWME）、ADC（两组，ADCA/ADCB）、DAC（两组，DACA/DACB）、CAN-DA（两组）、LCM 的 DMA，以及外设到外设直接的 DMA
- ✓ 硬件数字 ID：支持 32+32 字节

➢ 模拟外设

- ✓ ADC：两组独立的高速 ADC，支持 12 位高精度 15 通道（通道 0～通道 14）的模数转换，ADC 的通道 15 用于测试内部参考电压（芯片在出厂时，内部参考电压调整为 1.19V，误差±1%）
- ✓ DAC：两组独立的高速 DAC，支持 12 位转换精度。两组 DAC 的模拟输出均从芯片内部直接连接到比较器和运算放大器。
- ✓ 比较器：4 组独立的 6P6N 模式比较器（CMP）
- ✓ 运算放大器：4 组独立的运算放大器（OPA）

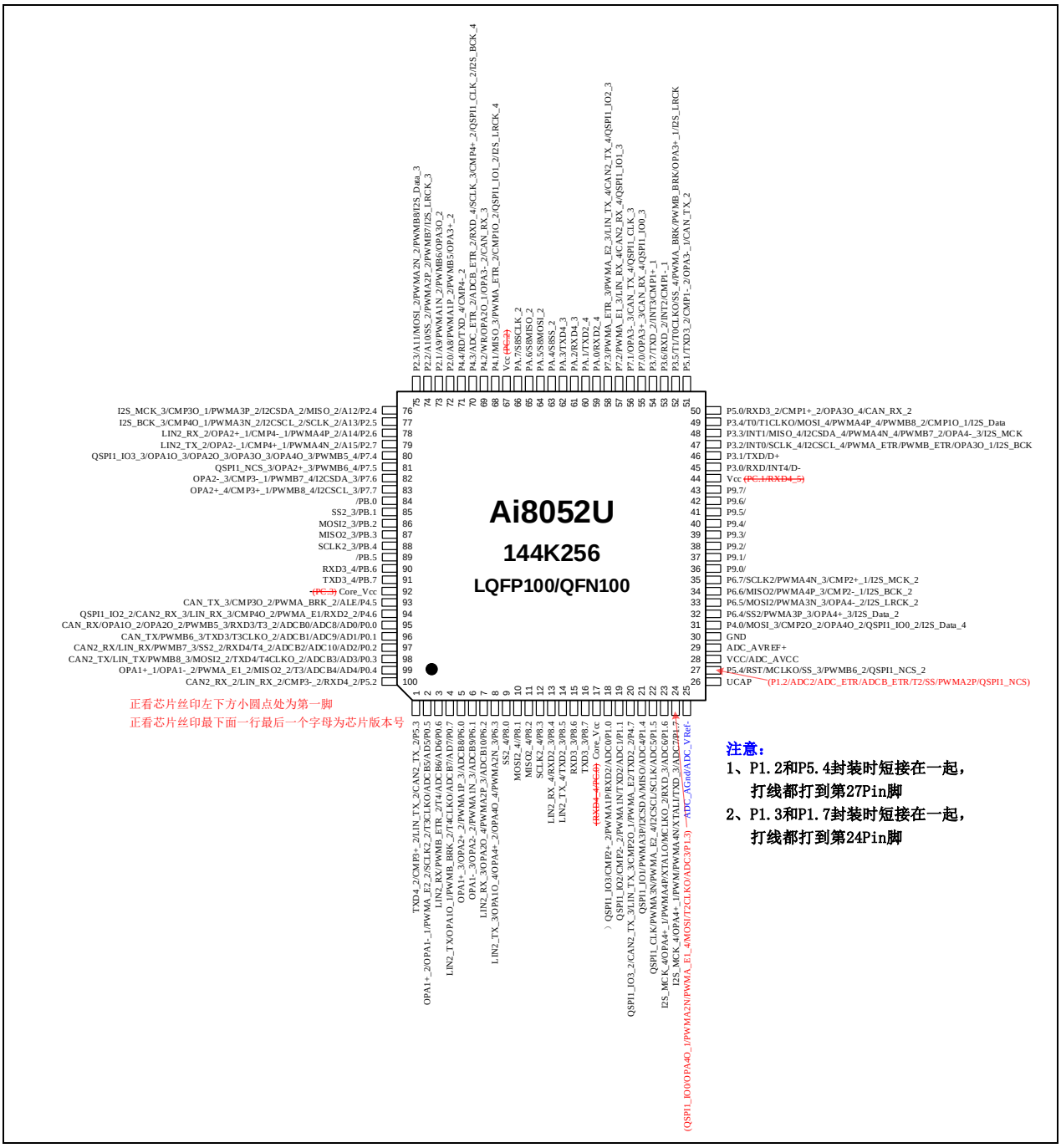
➢ GPIO

- ✓ 最多可达 96 个 GPIO：P0.0~P0.7、P1.0~ P1.7（P1.2 和 P5.4 为同一个 PIN 脚）、P2.0~P2.7、P3.0~P3.7、P4.0~P4.7、P5.0~P5.4、P6.0~P6.7、P7.0~P7.7、P8.0~P8.7、P9.0~P9.7、PA.0~PA.7、PB.0~PB.7、PC.0~PC.3
- ✓ 所有的 GPIO 均支持如下模式：~~准双向口模式~~、强推挽输出模式、开漏模式、高阻输入模式

- ✓ 所有 I/O 口上电后的状态均为高阻输入状态，用户在使用 I/O 口时必须先设置 I/O 口模式
- ✓ 另外每个 I/O 均可独立使能内部 10K 上拉电阻和 10K 下拉电阻

- ➢ **封装**
  - ✓ LQFP100、LQFP64、LQFP48、LQFP44

5.1.2　管脚图，最小系统（LQFP100/QFN100）



1、CPU，DSP 可工作在 80MHz 及以下

2、TFPU，PWM 可工作在 250MHz 及以下

3、144K SRAM，256K Flash

5.1.3 多功能管脚切换

USART1

RxD	TxD
P3.0	P3.1
P3.6	P3.7
P1.6	P1.7
P4.3	P4.4

USART1_SPI （MOSI和MISO可切换）

U1SS	U1MOSI	U1MISO	U1SCLK
P1.2	P1.3	P1.4	P1.5
P2.2	P2.3	P2.4	P2.5
P5.4	P4.0	P4.1	P4.2
P6.4	P6.5	P6.6	P6.7

USART2

RxD2	TxD2
P1.0	P1.1
P4.6	P4.7
P8.4	P8.5
PA.0	PA.1

USART2_SPI （MOSI和MISO可切换）

U2SS	U2MOSI	U2MISO	U2SCLK
P1.2	P1.3	P1.4	P1.5
P2.2	P2.3	P2.4	P2.5
P5.4	P4.0	P4.1	P4.2
P7.4	P7.5	P7.6	P7.7

USART3

RxD3	TxD3
P0.0	P0.1
P5.0	P5.1
P8.6	P8.7
PB.6	PB.7

USART3_SPI （MOSI和MISO可切换）

U3SS	U3MOSI	U3MISO	U3SCLK
P6.4	P6.5	P6.6	P6.7
P0.2	P0.3	P0.4	P0.5
PB.1	PB.2	PB.3	PB.4
P9.4	P9.5	P9.6	P9.7

USART4

RxD4	TxD4
P0.2	P0.3
P5.2	P5.3
PA.2	PA.3
PC.0	PC.1

USART4_SPI （MOSI和MISO可切换）

U4SS	U4MOSI	U4MISO	U4SCLK
P6.4	P6.5	P6.6	P6.7
P0.2	P0.3	P0.4	P0.5
PB.1	PB.2	PB.3	PB.4
P7.0	P7.1	P7.2	P7.3

USART5

RxD5	TxD5
P0.4	P0.5
P4.0	P4.1
P1.4	P1.5
P2.0	P2.1

USART5_SPI （MOSI和MISO可切换）

U5SS	U5MOSI	U5MISO	U5SCLK
P2.2	P2.3	P2.4	P2.5
P7.0	P7.1	P7.2	P7.3
P8.4	P8.5	P8.6	P8.7
P9.0	P9.1	P9.2	P9.3

USART6

RxD6	TxD6
P0.6	P0.7
P6.2	P6.3
P3.2	P3.3
P2.6	P2.7

USART6_SPI （MOSI和MISO可切换）

U6SS	U6MOSI	U6MISO	U6SCLK
P2.2	P2.3	P2.4	P2.5
P7.0	P7.1	P7.2	P7.3
P8.4	P8.5	P8.6	P8.7
P9.4	P9.5	P9.6	P9.7

USART7

RxD7	TxD7
P5.0	P5.1
PA.4	PA.5
P9.0	P9.1
P1.2	P1.3

USART7_SPI （MOSI和MISO可切换）

U7SS	U7MOSI	U7MISO	U7SCLK
P5.0	P5.1	P5.2	P5.3
PA.4	PA.5	PA.6	PA.7
P7.4	P7.5	P7.6	P7.7
PC.0	PC.1	PC.2	PC.3

USART8

RxD8	TxD8
P5.2	P5.3
PA.6	PA.7
P9.2	P9.3
P6.0	P6.1

USART8_SPI （MOSI和MISO可切换）

U8SS	U8MOSI	U8MISO	U8SCLK
P5.0	P5.1	P5.2	P5.3
PA.4	PA.5	PA.6	PA.7
P7.4	P7.5	P7.6	P7.7
P6.0	P6.1	P6.2	P6.3

CAN-FD

CANRx	CANTx	CANSTB
P0.0	P0.1	P0.6
P5.0	P5.1	P2.2
P4.2	P4.5	P5.1
P7.0	P7.1	P4.5
P3.0	P3.1	P3.3
P3.6	P3.7	P1.2
P1.4	P1.5	P3.3
P4.3	P4.4	P7.4

CAN-FD2

CAN2Rx	CAN2Tx	CAN2STB
P0.2	P0.3	P0.7
P5.2	P5.3	P2.3
P4.6	P4.7	P5.3
P7.2	P7.3	P4.7
P1.0	P1.1	P0.7
P8.4	P8.5	P8.3
PA.0	PA.1	P2.3
-	-	-

LIN

LINRx	LINTx
P0.2	P0.3
P5.2	P5.3
P4.6	P4.7
P7.2	P7.3

LIN2

LIN2Rx	LIN2Tx
P0.6	P0.7
P2.6	P2.7
P6.2	P6.3
P8.4	P8.5

DAC不能直接输出到I/O，必须经过OPA放大后才能输出到I/O

SPI （MOSI和MISO可切换）

SS	MOSI	MISO	SCLK
P1.2	P1.3	P1.4	P1.5
P2.2	P2.3	P2.4	P2.5
P5.4	P4.0	P4.1	P4.2
P3.5	P3.4	P3.3	P3.2

SPI2 （MOSI和MISO可切换）

SS2	MOSI2	MISO2	SCLK2
P6.4	P6.5	P6.6	P6.7
P0.2	P0.3	P0.4	P0.5
PB.1	PB.2	PB.3	PB.4
P8.0	P8.1	P8.2	P8.3

SPI3 （MOSI和MISO可切换）

SS3	MOSI3	MISO3	SCLK3
P2.2	P2.3	P2.4	P2.5
P7.0	P7.1	P7.2	P7.3
P8.4	P8.5	P8.6	P8.7
P9.0	P9.1	P9.2	P9.3

QSPI

NCS	IO0	IO1	SCLK	IO2	IO3
P1.2	P1.3	P1.4	P1.5	P1.1	P1.0
P5.4	P4.0	P4.1	P4.3	P4.6	P4.7
P7.5	P7.0	P7.2	P7.1	P7.3	P7.4
-	-	-	-	-	-

I2C（第1个I2C模块）

SCL	SDA
P1.5	P1.4
P2.5	P2.4
P7.7	P7.6
P3.2	P3.3

I2C2（第2个I2C模块）

SCL2	SDA2
P2.7	P2.6
P3.7	P3.6
P1.7	P1.6
P6.1	P6.0

I2S（第1个I2S模块）

BCK	MCK	Data	LRCK
P3.2	P3.3	P3.4	P3.5
P6.6	P6.7	P6.4	P6.5
P2.5	P2.4	P2.3	P2.2
P4.3	P1.6	P4.0	P4.1

I2S2（第2个I2S模块）

BCK2	MCK2	Data2	LRCK2
P5.3	P5.2	P5.1	P5.0
P6.3	P6.2	P6.1	P6.0
P8.3	P8.2	P8.1	P8.0
PB.4	PB.3	PB.2	PB.1

PWMA（可互补对称带死区输出，每路均可独立选择）

PWMA1P	PWMA1N	PWMA2P	PWMA2N	PWMA3P	PWMA3N	PWMA4P	PWMA4N	PWMAE1	PWMAE2	PWMAETR	PWMBBRK
P1.0	P1.1	P1.2	P1.3	P1.4	P1.5	P1.6	P1.7	P4.6	P4.7	P3.2	P3.5
P2.0	P2.1	P2.2	P2.3	P2.4	P2.5	P2.6	P2.7	P0.4	P0.5	P4.1	P4.5
P6.0	P6.1	P6.2	P6.3	P6.4	P6.5	P6.6	P6.7	P7.2	P7.3	P7.3	-
P9.4	P9.5	P9.6	P9.7	P9.4	P9.5	P9.6	P9.7	P1.3	P1.5	-	-

PWMC（可互补对称带死区输出，每路均可独立选择）

PWMC1P	PWMC1N	PWMC2P	PWMC2N	PWMC3P	PWMC3N	PWMC4P	PWMC4N	PWMCE1	PWMCE2	PWMCETR	PWMCBRK
P6.0	P6.1	P6.2	P6.3	P6.4	P6.5	P6.6	P6.7	P1.1	P1.7	P3.2	P3.5
P8.0	P8.1	P8.2	P8.3	P8.4	P8.5	P8.6	P8.7	P3.6	P3.7	P7.0	P7.1
P4.0	P4.1	P4.2	P4.3	P4.4	P4.5	P4.6	P4.7	P9.6	P9.7	P9.4	P9.5
PB.0	PB.1	PB.2	PB.3	PB.4	PB.5	PB.6	PB.7	P6.3	P6.5	-	-

PWME（可互补对称带死区输出，每路均可独立选择）

PWME1P	PWME1N	PWME2P	PWME2N	PWME3P	PWME3N	PWME4P	PWME4N	PWMEE1	PWMEE2	PWMEETR	PWMEBRK
P7.0	P7.1	P7.2	P7.3	P7.4	P7.5	P7.6	P7.7	P5.0	P5.1	P3.2	P3.5
P9.0	P9.1	P9.2	P9.3	P9.4	P9.5	P9.6	P9.7	P4.3	P4.4	PC.2	PC.3
PA.0	PA.1	PA.2	PA.3	PA.4	PA.5	PA.6	PA.7	PB.4	PB.5	P0.2	P0.3
-	-	-	-	-	-	-	-	P9.5	P9.7	-	-

PWMB（单端输出，每路均可独立选择）

PWMB5P	PWMB6P	PWMB7P	PWMB8P	PWMBETR	PWMBBRK
P2.0	P2.1	P2.2	P2.3	P3.2	P3.5
P1.7	P5.4	P3.3	P3.4	P0.6	P0.7
P0.0	P0.1	P0.2	P0.3	-	-
P7.4	P7.5	P7.6	P7.7	-	-

PWMD（单端输出，每路均可独立选择）

PWMD5P	PWMD6P	PWMD7P	PWMD8P	PWMDETR	PWMDBRK
P2.4	P2.5	P2.6	P2.7	P3.2	P3.5
P8.1	P8.3	P8.5	P8.7	P7.0	P7.1
P5.0	P5.1	P5.2	P5.3	PC.0	PC.1
PB.4	PB.5	PB.6	PB.7	-	-

PWMF（单端输出，每路均可独立选择）

PWMF5P	PWMF6P	PWMF7P	PWMF8P	PWMFETR	PWMFBRK
P7.0	P7.1	P7.2	P7.3	P3.2	P3.5
PA.1	PA.3	PA.5	PA.7	PB.1	PB.3
P9.0	P9.1	P9.2	P9.3	P0.0	P0.1
-	-	-	-	-	-

CMP（第1个比较器）

CMP1+	CMP1-	CMP1O
P3.7	P3.6	P3.4
P5.0	P5.1	P4.1
OP1O	OP2O	-
ADCIN	1.19V	-
DAC1O	DAC1O	-
DAC2O	DAC2O	-

CMP2（第2个比较器）

CMP2+	CMP2-	CMP2O
P6.7	P6.6	P4.7
P1.0	P1.1	P4.0
OP2O	OP3O	-
ADCIN	1.19V	-
DAC1O	DAC1O	-
DAC2O	DAC2O	-

CMP3（第3个比较器）

CMP3+	CMP3-	CMP3O
P7.7	P7.6	P2.4
P5.3	P5.2	P4.5
OP3O	OP4O	-
ADCBIN	1.19V	-
DAC1O	DAC1O	-
DAC2O	DAC2O	-

CMP4（第4个比较器）

CMP4+	CMP4-	CMP4O
P2.7	P2.6	P2.5
P4.3	P4.4	P4.6
OP4O	OP1O	-
ADCBIN	1.19V	-
DAC1O	DAC1O	-
DAC2O	DAC2O	-

比较器的正极输入端CMPn+和负极输入端CMPn-均可独立选择

OPA1（输入可独立选择）

OP1+	OP1-	OP1O
P0.4	P0.5	P0.7
P0.5	P0.4	P0.0
P6.0	P6.1	P7.4
DAC1O	DAC1O	P6.3

OPA2（输入可独立选择）

OP2+	OP2-	OP2O
P2.6	P2.7	P4.2
P6.0	P6.1	P0.0
P7.5	P7.6	P7.4
P7.7	DAC2O	P6.2

OPA3（输入可独立选择）

OP3+	OP3-	OP3O
P3.5	P5.1	P3.2
P2.0	P4.2	P2.1
P7.0	P7.1	P7.4
DAC2O	DAC2O	P5.0

OPA4（输入可独立选择）

OP4+	OP4-	OP4O
P1.6	P1.7	P1.3
P6.3	P6.5	P4.0
P6.4	P3.3	P7.4
DAC2O	DAC1O	P6.3

DAC1

输出
DAC1O

DAC2

输出
DAC2O

ADC（第1个ADC模块）

ADC0	ADC1	AIDC2	ADC3	ADC4	ADC5	ADC6	ADC7
P1.0	P1.1	P1.2	P1.3	P1.4	P1.5	P1.6	P1.7
ADC8	ADC9	ADC10	ADC11	DC12	ADC13	ADC14	ADC15
P0.0	P0.1	P0.2	OP1O	OP2O	OP3O	OP4O	1.19V

ADCB（第2个ADC模块）

ADCB0	ADCB1	ADCB2	ADCB3	ADCB4	ADCB5	ADCB6	ADCB7
P0.0	P0.1	P0.2	P0.3	P0.4	P0.5	P0.6	P0.7
ADCB8	ADCB9	ADCB10	ADCB11	ADCB12	ADCB13	ADCB14	ADCB15
P6.0	P6.1	P6.2	OP1O	OP2O	OP3O	OP4O	1.19V

TFT-i8080/M6800接口 - 8位数据

RS	WR	RD	D0	D1	D2	D3	D4	D5	D6	D7
P4.5	P4.2	P4.4	P2.0	P2.1	P2.2	P2.3	P2.4	P2.5	P2.6	P2.7
P4.5	P3.6	P3.7	P6.0	P6.1	P6.2	P6.3	P6.4	P6.5	P6.6	P6.7
P4.0	P4.2	P4.4	P2.0	P2.1	P2.2	P2.3	P2.4	P2.5	P2.6	P2.7
P4.0	P3.6	P3.7	P6.0	P6.1	P6.2	P6.3	P6.4	P6.5	P6.6	P6.7

TFT-i8080/M6800接口 - 16位数据

RS	WR	RD	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15
P4.5	P4.2	P4.4	P0.0	P0.1	P0.2	P0.3	P0.4	P0.5	P0.6	P0.7	P2.0	P2.1	P2.2	P2.3	P2.4	P2.5	P2.6	P2.7
P4.5	P3.6	P3.7	P2.0	P2.1	P2.2	P2.3	P2.4	P2.5	P2.6	P2.7	P6.0	P6.1	P6.2	P6.3	P6.4	P6.5	P6.6	P6.7
P4.0	P4.2	P4.4	P4.1	P4.3	P4.6	P4.7	P0.4	P0.5	P0.6	P0.7	P2.0	P2.1	P2.2	P2.3	P2.4	P2.5	P2.6	P2.7
P4.0	P3.6	P3.7	P7.0	P7.1	P7.2	P7.3	P7.4	P7.5	P7.6	P7.7	P6.0	P6.1	P6.2	P6.3	P6.4	P6.5	P6.6	P6.7

说明：

1、ADC为第一个ADC模块，ADCB为第二个ADC模块，每个ADC模块都有16个通道，其中11个通道的输入可以选择外部I/O，另外5个通道的输入可以选择内部的模拟信号。

2、CMP1/CMP2/CMP3/CMP4为4个比较器模块，每个比较器都是6P6N模式，都支持6个正极输入，6个负极输入。

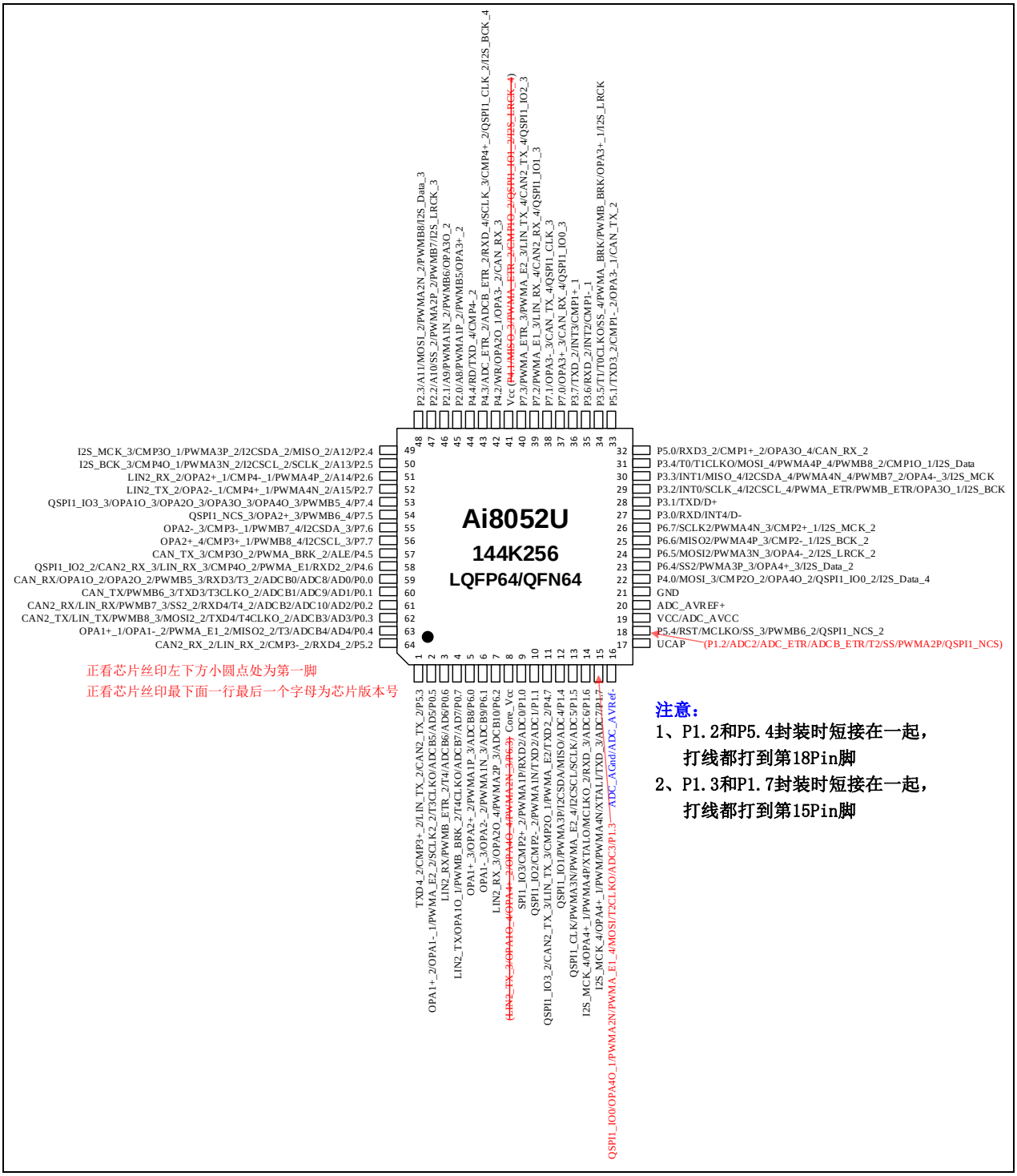
3、OPA1/OPA2/OPA3/OPA4为4个运算放大器，运算放大器的输入信号可以从外部I/O输入，也可选择内部的模拟信号。OP1O/OP2O/OP3O/OP4O分别为4个运放的输出信号。

4、Ai8052U芯片内部集成了两个数模转换模块（DAC1/DAC2），DAC1O/DAC2O分别为2个DAC的输出。
由于DAC的输出信号驱动能力弱，不能直接输出到外部管脚，芯片内部将DAC的输出直接连接到OPA（运算放大器）的输入端，驱动能力经过放大后再输出到外部管脚。

5、1.19V信号为内部BandGap电压放大后的电压，是辅助固定参考信号源，不能直接作为ADC或者DAC的参考电压。

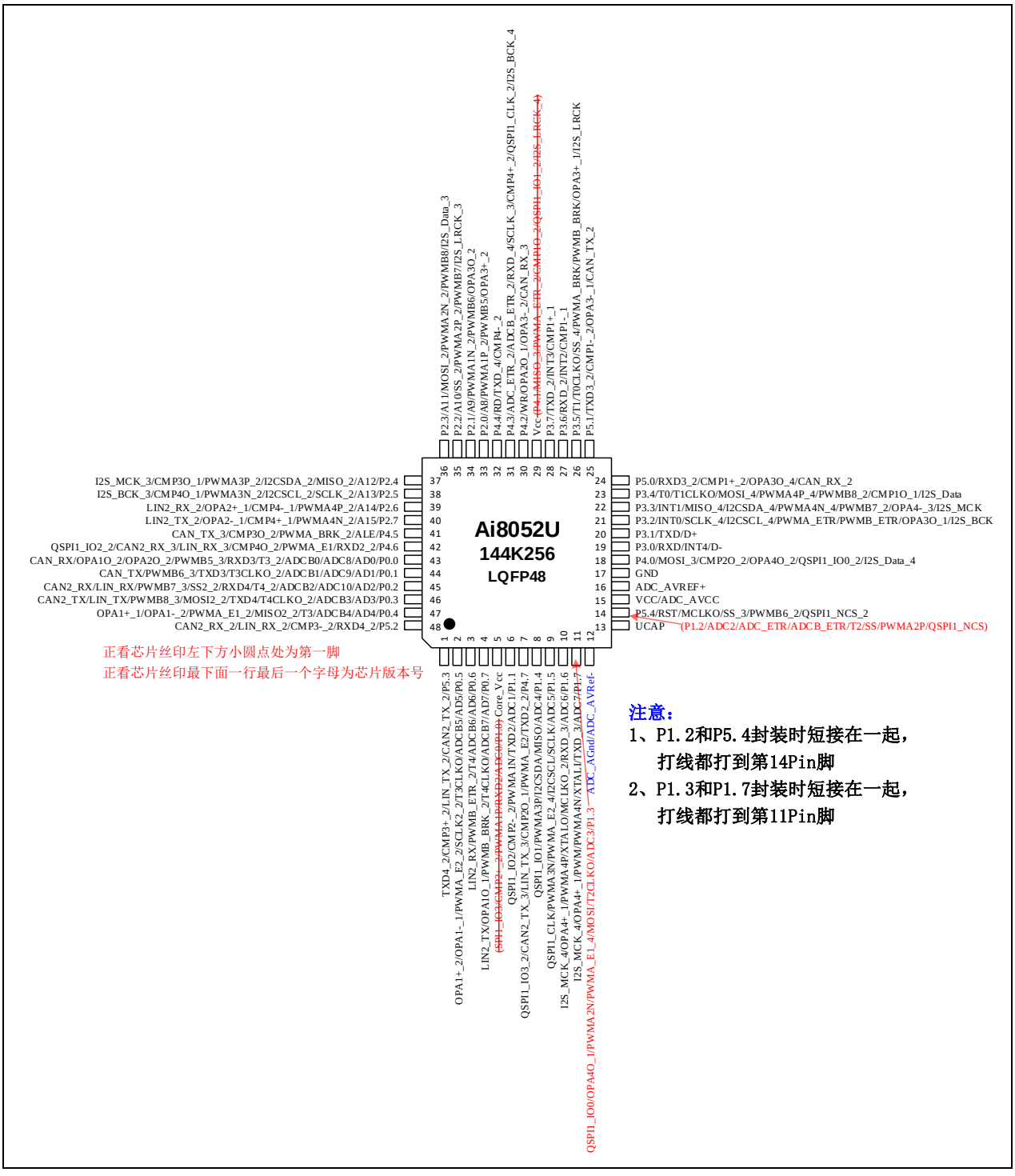
6、ADC、PWM、比较器、运算放大器的输入和输出端均可独立选择，其他的外设USART、SPI、QSPI、I2C、I2S、CAN-FD、LIN、TFT，均为整组（整行）选择。

5.1.4　管脚图，最小系统（LQFP64/QFN64）



1、CPU，DSP 可工作在 80MHz 及以下
2、TFPU，PWM 可工作在 250MHz 及以下
3、144K SRAM，256K Flash

5.1.5　管脚图，最小系统（LQFP48）



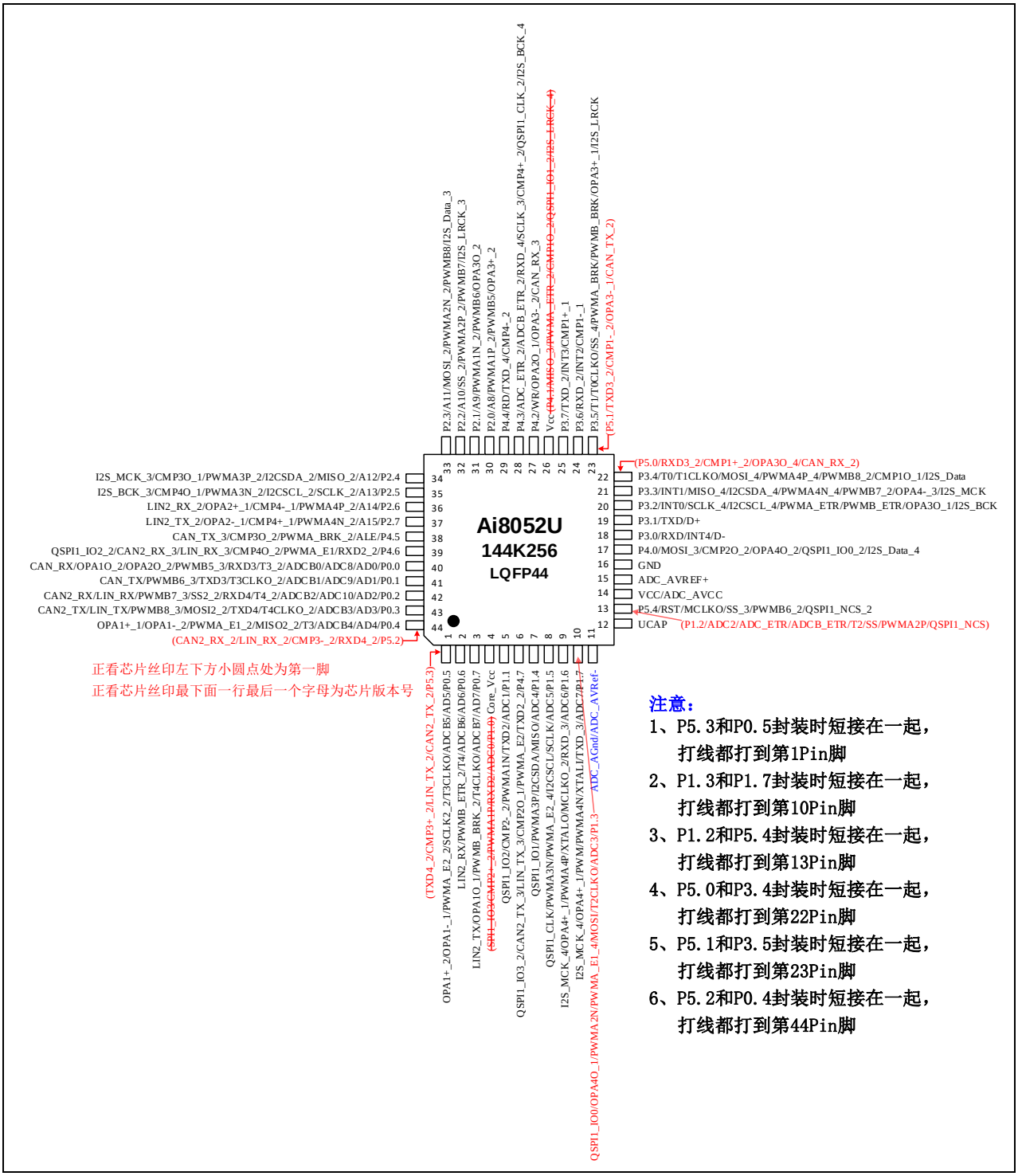
1、CPU，DSP 可工作在 80MHz 及以下

2、TFPU，PWM 可工作在 250MHz 及以下

3、144K SRAM，256K Flash

尽量选 LQFP100、LQFP64，尽量不选 LQFP48，减轻库存及生产压力

5.1.6　管脚图，最小系统（LQFP44）



正看芯片丝印左下方小圆点处为第一脚
正看芯片丝印最下面一行最后一个字母为芯片版本号

1、CPU，DSP 可工作在 80MHz 及以下
2、TFPU，PWM 可工作在 250MHz 及以下
3、144K SRAM，256K Flash

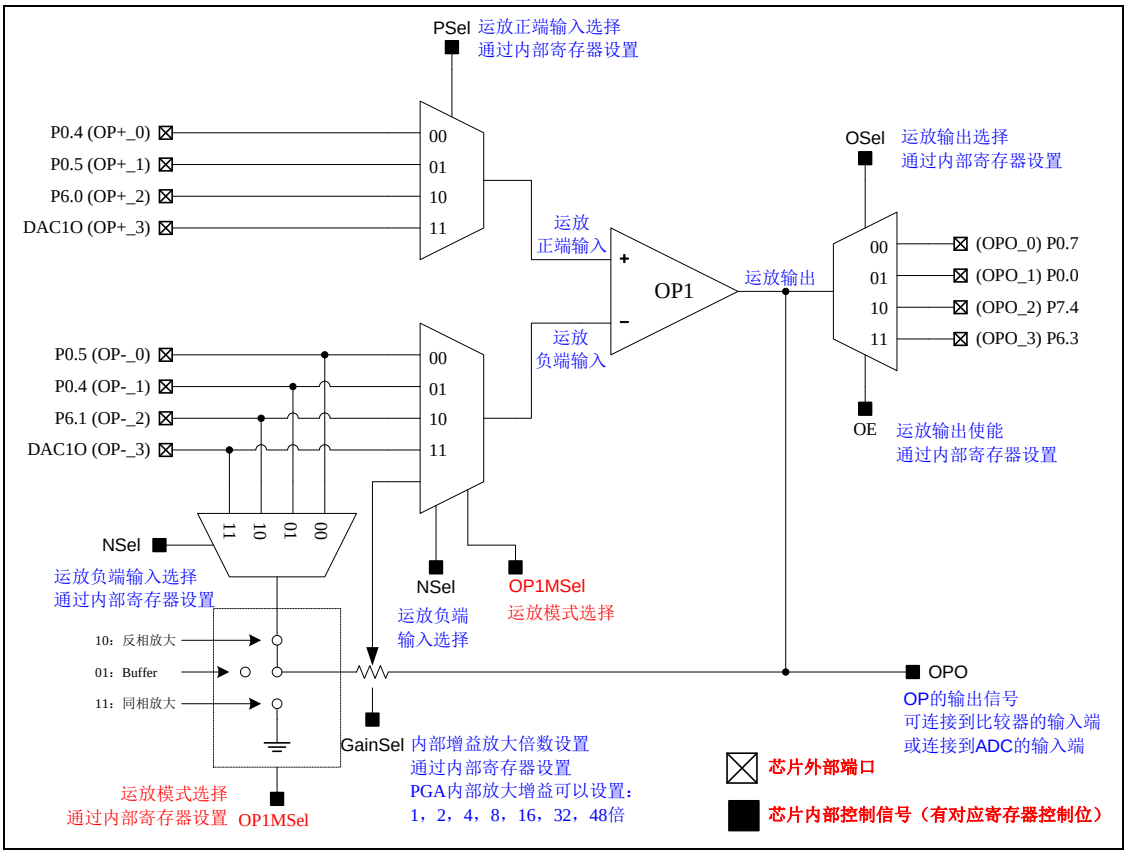
尽量选 LQFP100、LQFP64，尽量不选 LQFP44，减轻库存及生产压力

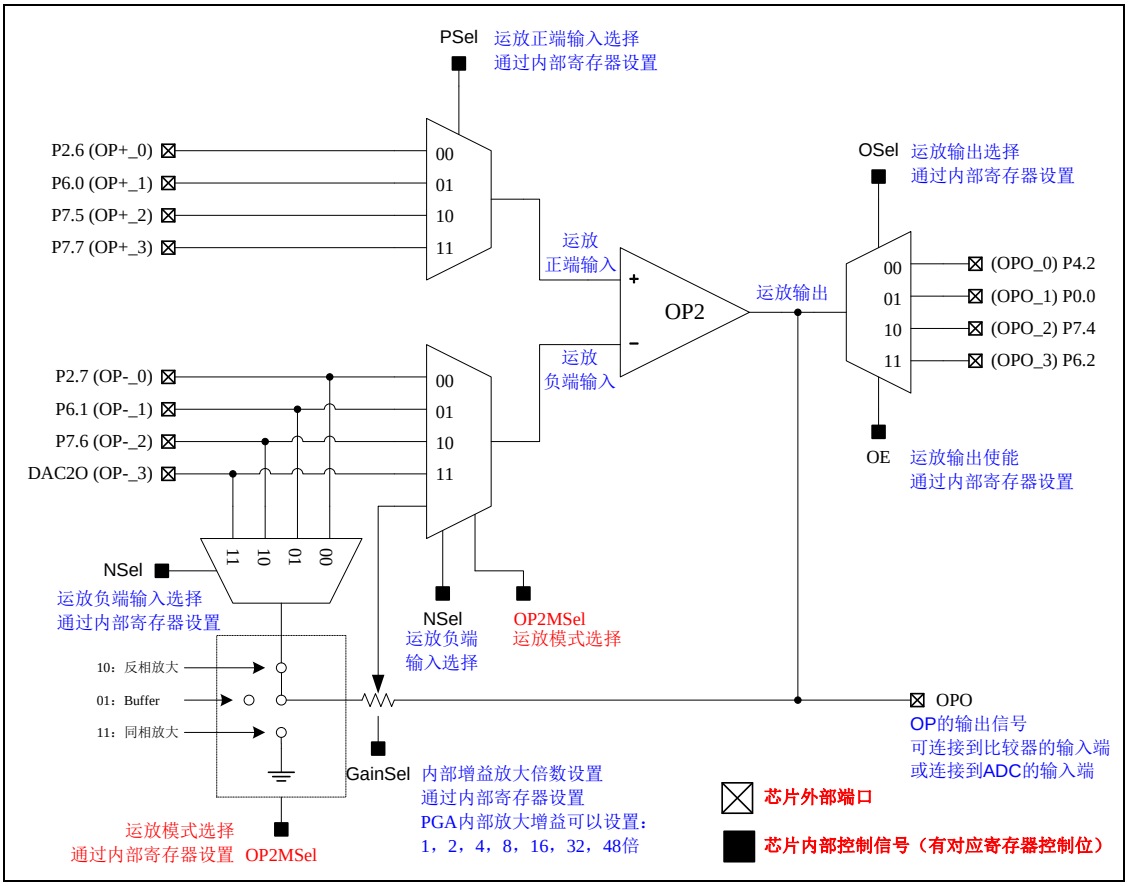
备战 2025 年 全国大学生智能汽车竞赛，全国大学生电子设计竞赛
大家先看下管脚图合不合理，10 月底流片，12 月回来，新年送样
帮设计强大的 AI8052U-144K256-LQFP100 实验箱

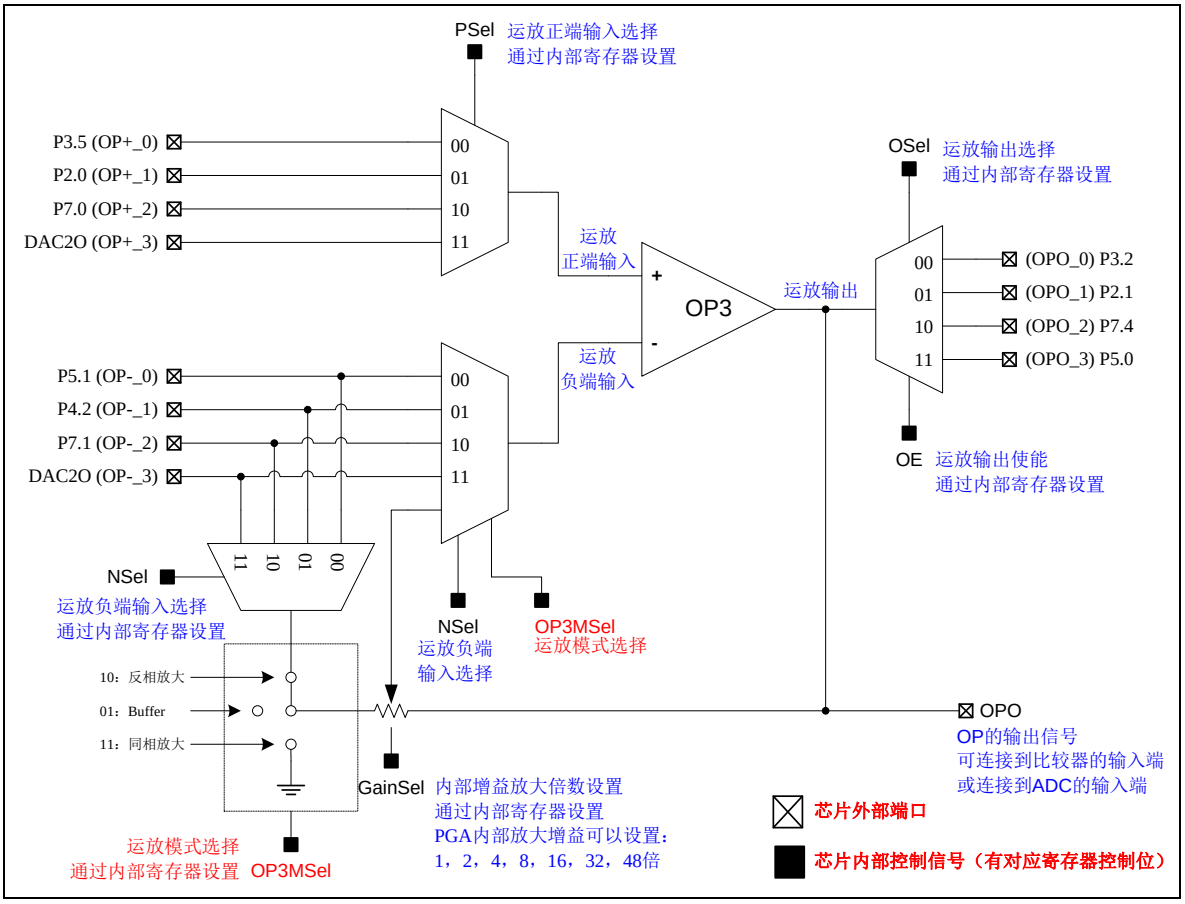
又名：AI32G144K256-250MHz-LQFP100, LQFP64, LQFP48/44
144K SRAM, 256K Flash

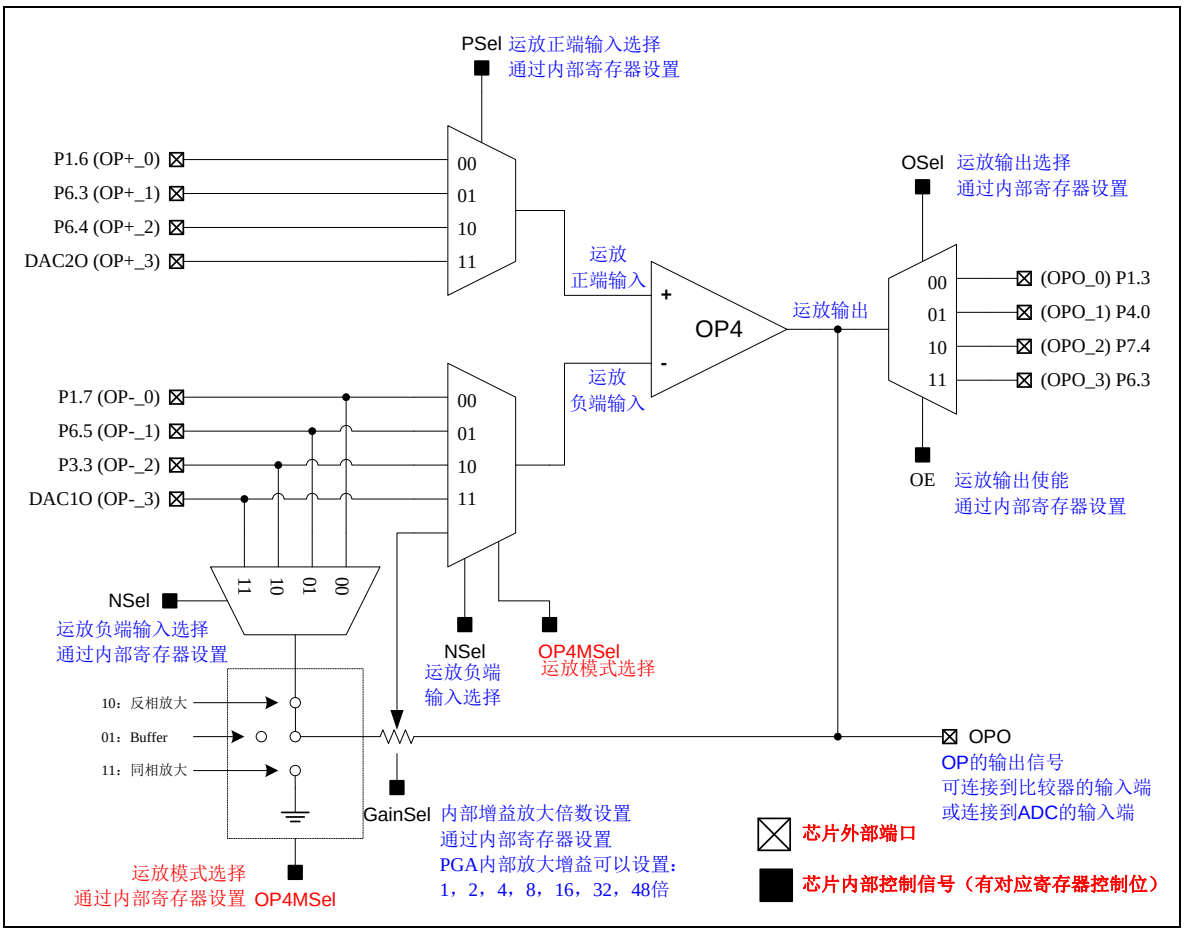
管脚兼容 8H8K64U, 32G12K128, 32G8K64, 32F12K54 系列

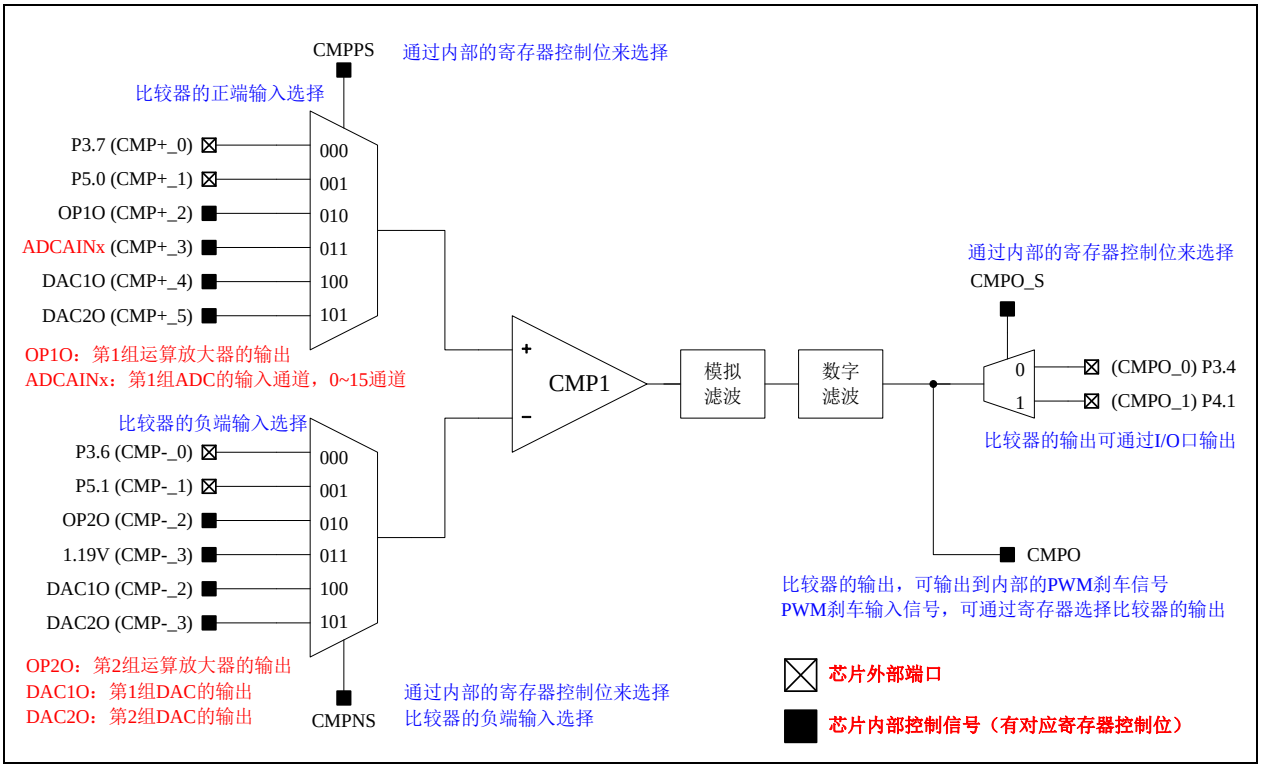
5.2　两组独立 DAC、4 组运放、4 组比较器结构及应用，整理中

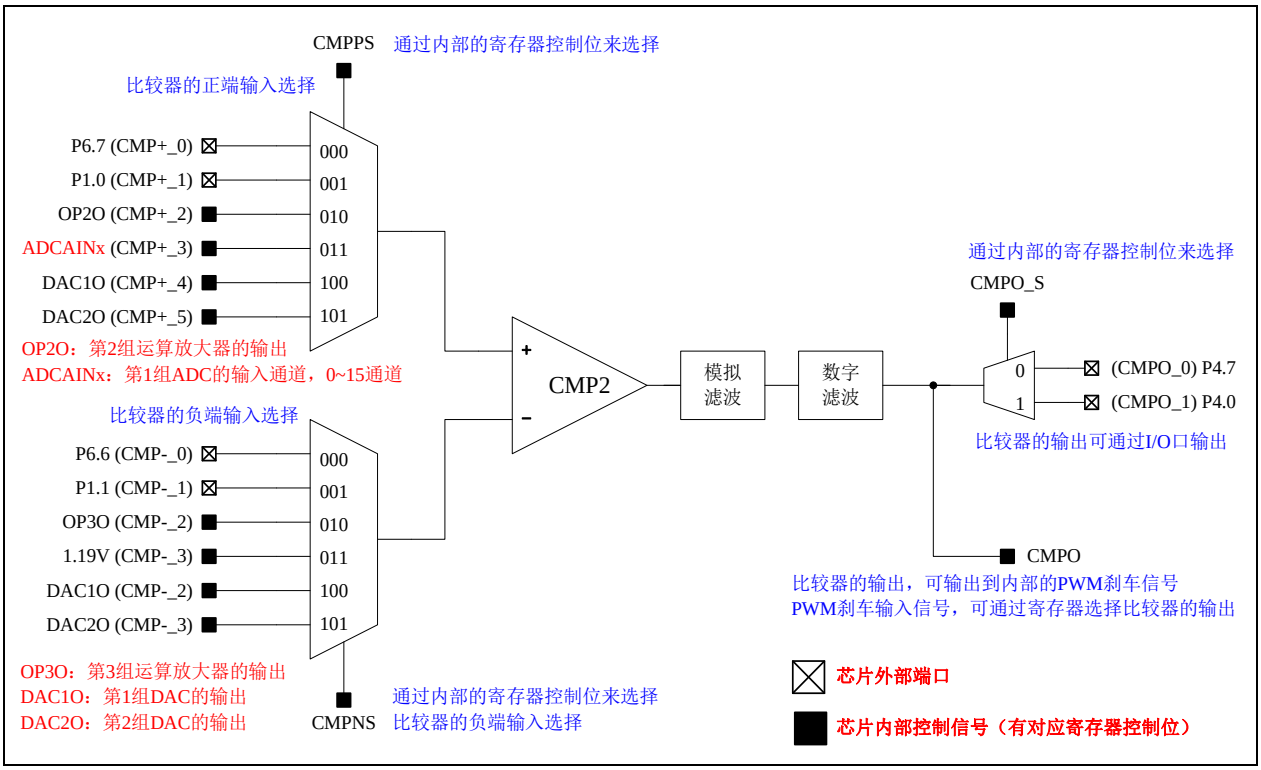


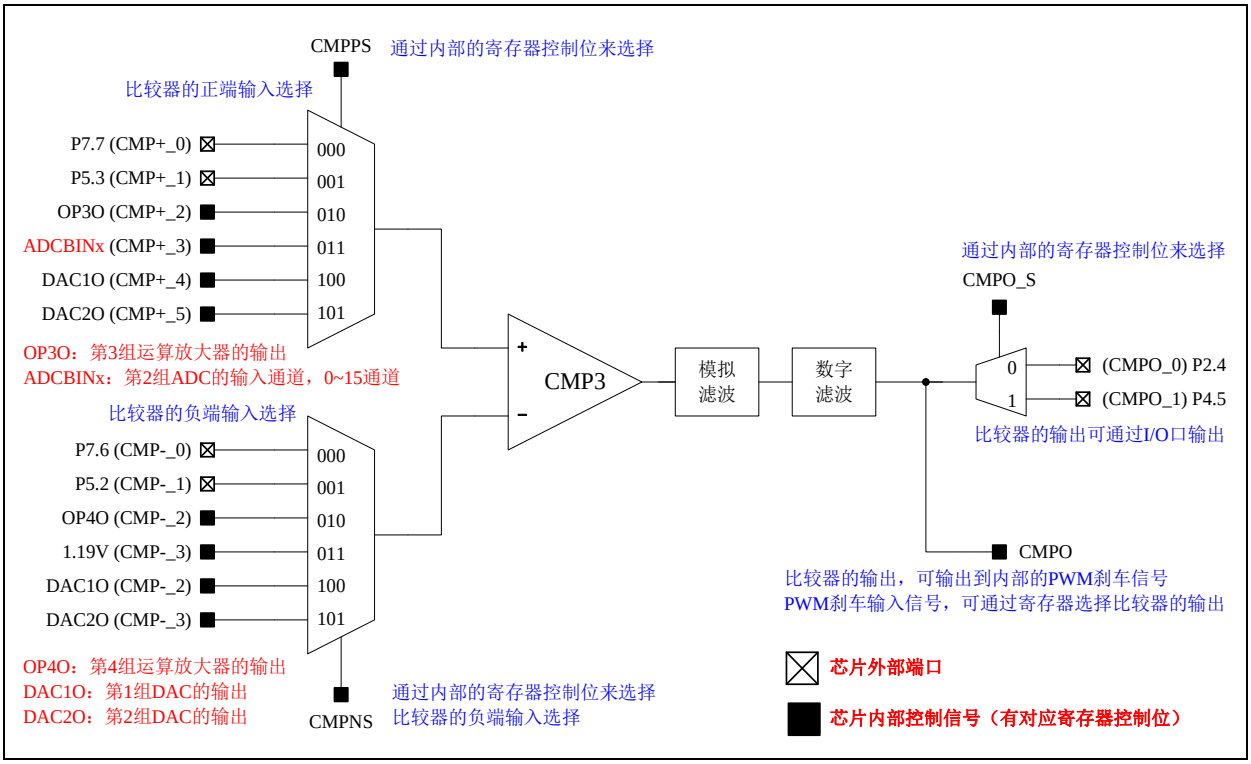


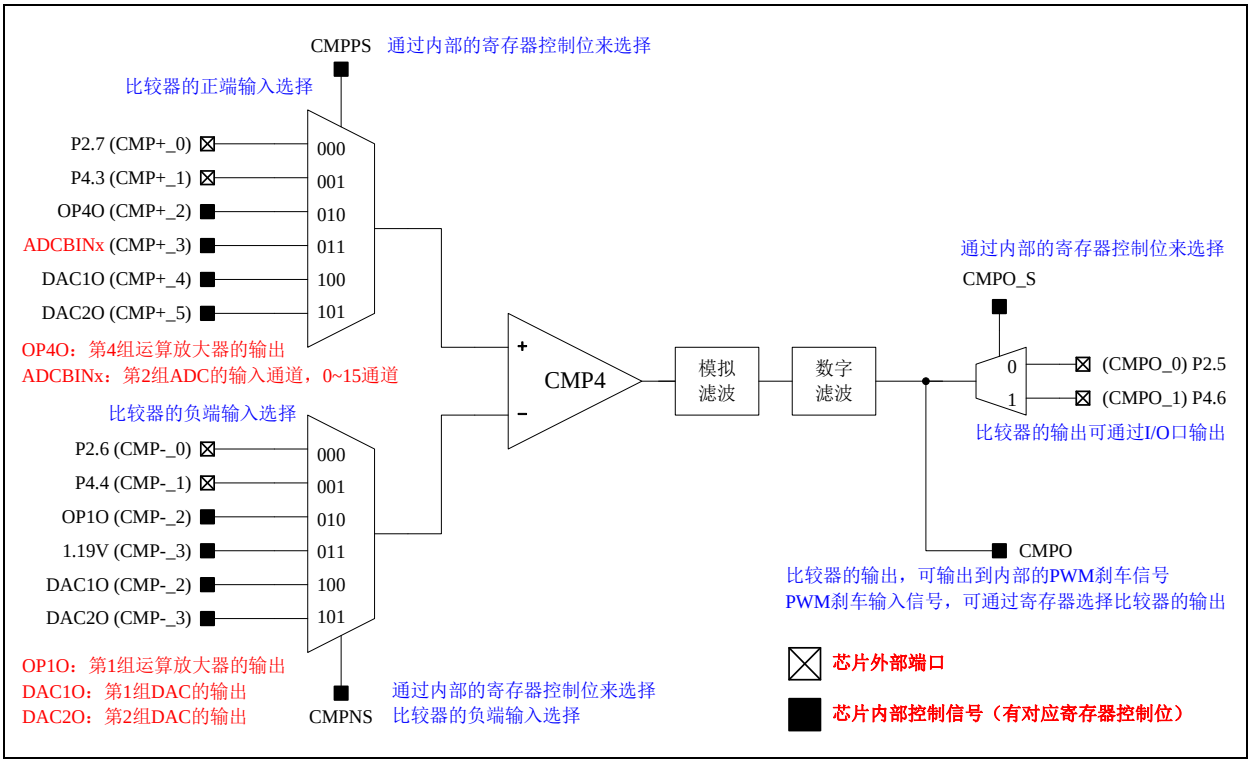

PSel 运放正端输入选择
通过内部寄存器设置
P3.5 (OP+_0)
P2.0 (OP+_1)
P7.0 (OP+_2)
DAC2O (OP+_3)
00
01
10
11
运放
正端输入
OP3
运放输出
OSel 运放输出选择
通过内部寄存器设置
(OPO_0) P3.2
(OPO_1) P2.1
(OPO_2) P7.4
(OPO_3) P5.0
P5.1 (OP-_0)
P4.2 (OP-_1)
P7.1 (OP-_2)
DAC2O (OP-_3)
运放
负端输入
OE 运放输出使能
通过内部寄存器设置
NSel
运放负端输入选择
通过内部寄存器设置
NSel
运放负端
输入选择
OP3MSel
运放模式选择
10：反相放大
01：Buffer
11：同相放大
OPO
OP的输出信号
可连接到比较器的输入端
或连接到ADC的输入端
GainSel 内部增益放大倍数设置
通过内部寄存器设置
PGA内部放大增益可以设置：
1，2，4，8，16，32，48倍
运放模式选择
通过内部寄存器设置 OP3MSel
芯片外部端口
芯片内部控制信号（有对应寄存器控制位）

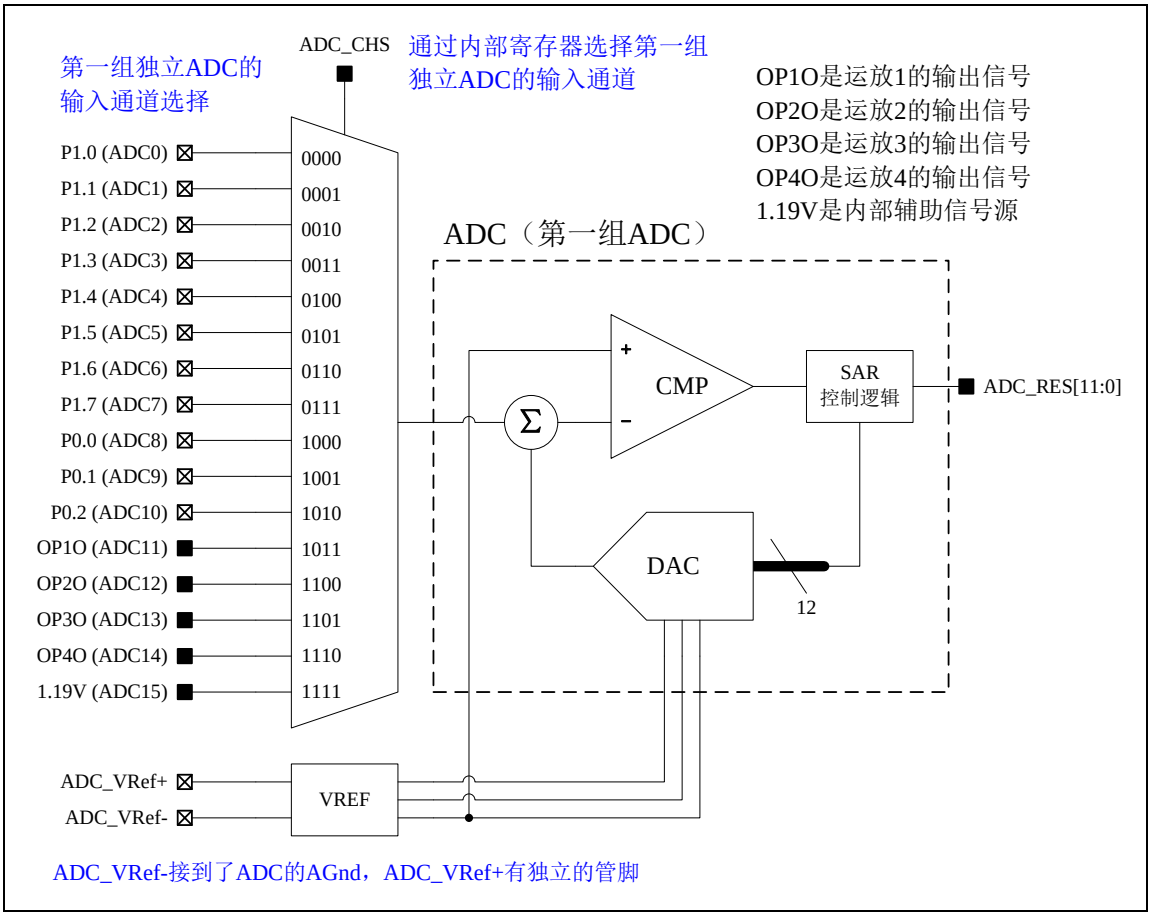

PSel 运放正端输入选择
通过内部寄存器设置
P1.6 (OP+_0)
P6.3 (OP+_1)
P6.4 (OP+_2)
DAC2O (OP+_3)
00
01
10
11
运放
正端输入
OP4
运放输出
OSel 运放输出选择
通过内部寄存器设置
(OPO_0) P1.3
(OPO_1) P4.0
(OPO_2) P7.4
(OPO_3) P6.3
P1.7 (OP-_0)
P6.5 (OP-_1)
P3.3 (OP-_2)
DAC1O (OP-_3)
运放
负端输入
OE 运放输出使能
通过内部寄存器设置
NSel
运放负端输入选择
通过内部寄存器设置
NSel
运放负端
输入选择
OP4MSel
运放模式选择
10：反相放大
01：Buffer
11：同相放大
OPO
OP的输出信号
可连接到比较器的输入端
或连接到ADC的输入端
GainSel 内部增益放大倍数设置
通过内部寄存器设置
PGA内部放大增益可以设置：
1，2，4，8，16，32，48倍
运放模式选择
通过内部寄存器设置 OP4MSel
芯片外部端口
芯片内部控制信号（有对应寄存器控制位）

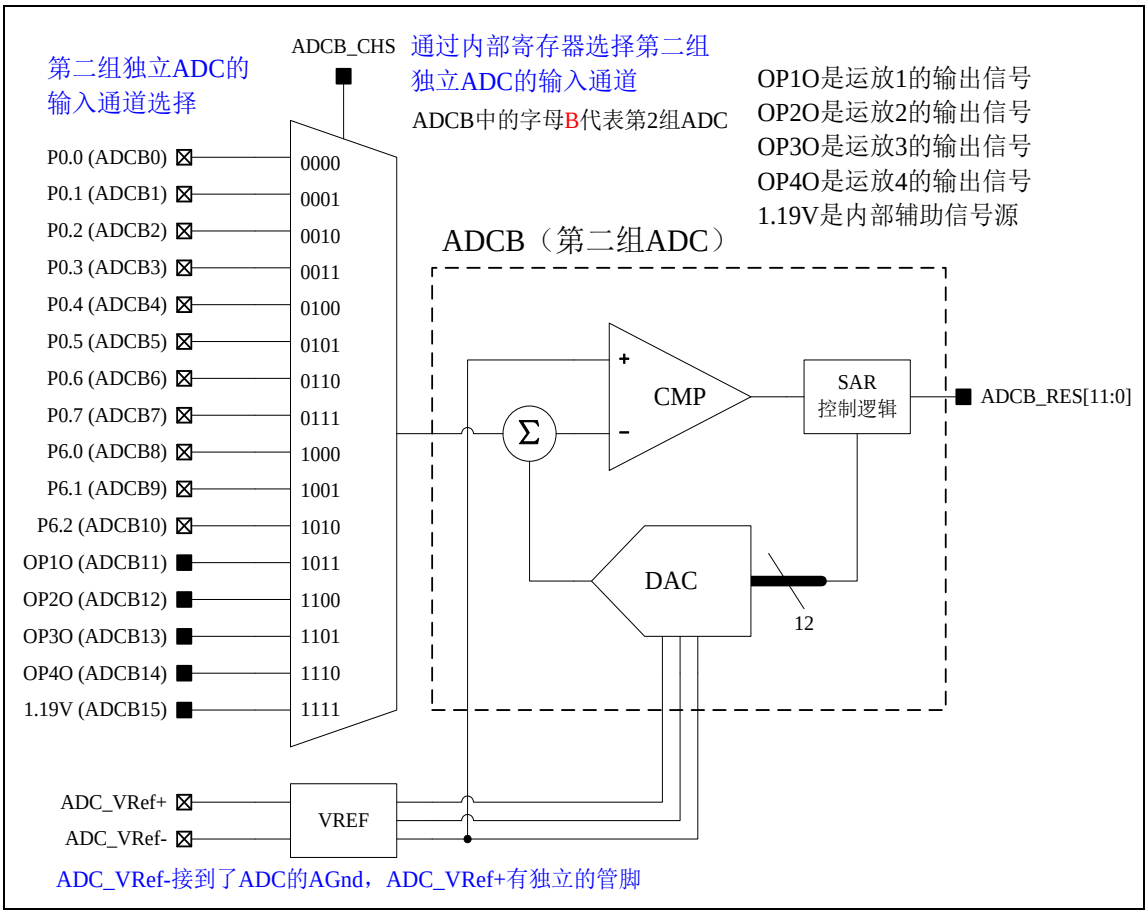

CMPPS
通过内部的寄存器控制位来选择
比较器的正端输入选择
P3.7 (CMP+_0)
P5.0 (CMP+_1)
OP1O (CMP+_2)
ADCAINx (CMP+_3)
DAC1O (CMP+_4)
DAC2O (CMP+_5)
000
001
010
011
100
101
OP1O：第1组运算放大器的输出
ADCAINx：第1组ADC的输入通道，0~15通道
比较器的负端输入选择
P3.6 (CMP-_0)
P5.1 (CMP-_1)
OP2O (CMP-_2)
1.19V (CMP-_3)
DAC1O (CMP-_2)
DAC2O (CMP-_3)
OP2O：第2组运算放大器的输出
DAC1O：第1组DAC的输出
DAC2O：第2组DAC的输出
CMPNS
通过内部的寄存器控制位来选择
比较器的负端输入选择
CMP1
模拟滤波
数字滤波
通过内部的寄存器控制位来选择
CMPO_S
0
1
(CMPO_0) P3.4
(CMPO_1) P4.1
比较器的输出可通过I/O口输出
CMPO
比较器的输出，可输出到内部的PWM刹车信号
PWM刹车输入信号，可通过寄存器选择比较器的输出
芯片外部端口
芯片内部控制信号（有对应寄存器控制位）


CMPPS
通过内部的寄存器控制位来选择
比较器的正端输入选择
P6.7 (CMP+_0)
P1.0 (CMP+_1)
OP2O (CMP+_2)
ADCAINx (CMP+_3)
DAC1O (CMP+_4)
DAC2O (CMP+_5)
000
001
010
011
100
101
OP2O：第2组运算放大器的输出
ADCAINx：第1组ADC的输入通道，0~15通道
比较器的负端输入选择
P6.6 (CMP-_0)
P1.1 (CMP-_1)
OP3O (CMP-_2)
1.19V (CMP-_3)
DAC1O (CMP-_2)
DAC2O (CMP-_3)
OP3O：第3组运算放大器的输出
DAC1O：第1组DAC的输出
DAC2O：第2组DAC的输出
CMPNS
通过内部的寄存器控制位来选择
比较器的负端输入选择
CMP2
模拟滤波
数字滤波
通过内部的寄存器控制位来选择
CMPO_S
0
1
(CMPO_0) P4.7
(CMPO_1) P4.0
比较器的输出可通过I/O口输出
CMPO
比较器的输出，可输出到内部的PWM刹车信号
PWM刹车输入信号，可通过寄存器选择比较器的输出
芯片外部端口
芯片内部控制信号（有对应寄存器控制位）


CMPPS
通过内部的寄存器控制位来选择
比较器的正端输入选择
P7.7 (CMP+_0)
P5.3 (CMP+_1)
OP3O (CMP+_2)
ADCBINx (CMP+_3)
DAC1O (CMP+_4)
DAC2O (CMP+_5)
000
001
010
011
100
101
OP3O：第3组运算放大器的输出
ADCBINx：第2组ADC的输入通道，0~15通道
比较器的负端输入选择
P7.6 (CMP-_0)
P5.2 (CMP-_1)
OP4O (CMP-_2)
1.19V (CMP-_3)
DAC1O (CMP-_2)
DAC2O (CMP-_3)
OP4O：第4组运算放大器的输出
DAC1O：第1组DAC的输出
DAC2O：第2组DAC的输出
CMPNS
通过内部的寄存器控制位来选择
比较器的负端输入选择
CMP3
模拟滤波
数字滤波
通过内部的寄存器控制位来选择
CMPO_S
0
1
(CMPO_0) P2.4
(CMPO_1) P4.5
比较器的输出可通过I/O口输出
CMPO
比较器的输出，可输出到内部的PWM刹车信号
PWM刹车输入信号，可通过寄存器选择比较器的输出
芯片外部端口
芯片内部控制信号（有对应寄存器控制位）


CMPPS
通过内部的寄存器控制位来选择
比较器的正端输入选择
P2.7 (CMP+_0)
P4.3 (CMP+_1)
OP4O (CMP+_2)
ADCBINx (CMP+_3)
DAC1O (CMP+_4)
DAC2O (CMP+_5)
000
001
010
011
100
101
OP4O：第4组运算放大器的输出
ADCBINx：第2组ADC的输入通道，0~15通道
比较器的负端输入选择
P2.6 (CMP-_0)
P4.4 (CMP-_1)
OP1O (CMP-_2)
1.19V (CMP-_3)
DAC1O (CMP-_2)
DAC2O (CMP-_3)
OP1O：第1组运算放大器的输出
DAC1O：第1组DAC的输出
DAC2O：第2组DAC的输出
CMPNS
通过内部的寄存器控制位来选择
比较器的负端输入选择
CMP4
模拟滤波
数字滤波
通过内部的寄存器控制位来选择
CMPO_S
0
1
(CMPO_0) P2.5
(CMPO_1) P4.6
比较器的输出可通过I/O口输出
CMPO
比较器的输出，可输出到内部的PWM刹车信号
PWM刹车输入信号，可通过寄存器选择比较器的输出
芯片外部端口
芯片内部控制信号（有对应寄存器控制位）


第一组独立ADC的
输入通道选择
ADC_CHS
通过内部寄存器选择第一组
独立ADC的输入通道
OP1O是运放1的输出信号
OP2O是运放2的输出信号
OP3O是运放3的输出信号
OP4O是运放4的输出信号
1.19V是内部辅助信号源
ADC（第一组ADC）
P1.0 (ADC0)
P1.1 (ADC1)
P1.2 (ADC2)
P1.3 (ADC3)
P1.4 (ADC4)
P1.5 (ADC5)
P1.6 (ADC6)
P1.7 (ADC7)
P0.0 (ADC8)
P0.1 (ADC9)
P0.2 (ADC10)
OP1O (ADC11)
OP2O (ADC12)
OP3O (ADC13)
OP4O (ADC14)
1.19V (ADC15)
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
CMP
SAR
控制逻辑
ADC_RES[11:0]
DAC
12
ADC_VRef+
ADC_VRef-
VREF
ADC_VRef-接到了ADC的AGnd，ADC_VRef+有独立的管脚

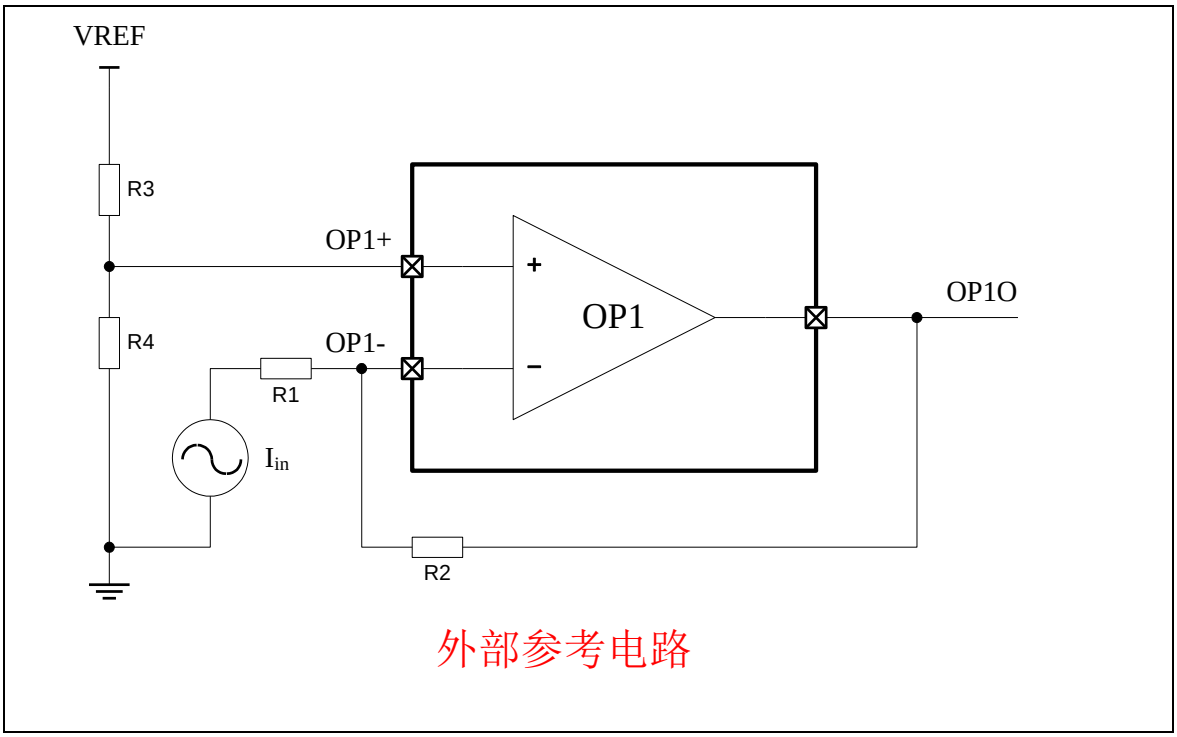

第二组独立ADC的
输入通道选择
ADCB_CHS
通过内部寄存器选择第二组
独立ADC的输入通道
ADCB中的字母B代表第2组ADC
OP1O是运放1的输出信号
OP2O是运放2的输出信号
OP3O是运放3的输出信号
OP4O是运放4的输出信号
1.19V是内部辅助信号源
ADCB（第二组ADC）
P0.0 (ADCB0)
P0.1 (ADCB1)
P0.2 (ADCB2)
P0.3 (ADCB3)
P0.4 (ADCB4)
P0.5 (ADCB5)
P0.6 (ADCB6)
P0.7 (ADCB7)
P6.0 (ADCB8)
P6.1 (ADCB9)
P6.2 (ADCB10)
OP1O (ADCB11)
OP2O (ADCB12)
OP3O (ADCB13)
OP4O (ADCB14)
1.19V (ADCB15)
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
CMP
SAR
控制逻辑
ADCB_RES[11:0]
DAC
12
ADC_VRef+
ADC_VRef-
VREF
ADC_VRef-接到了ADC的AGnd，ADC_VRef+有独立的管脚

5.2.1　反相放大

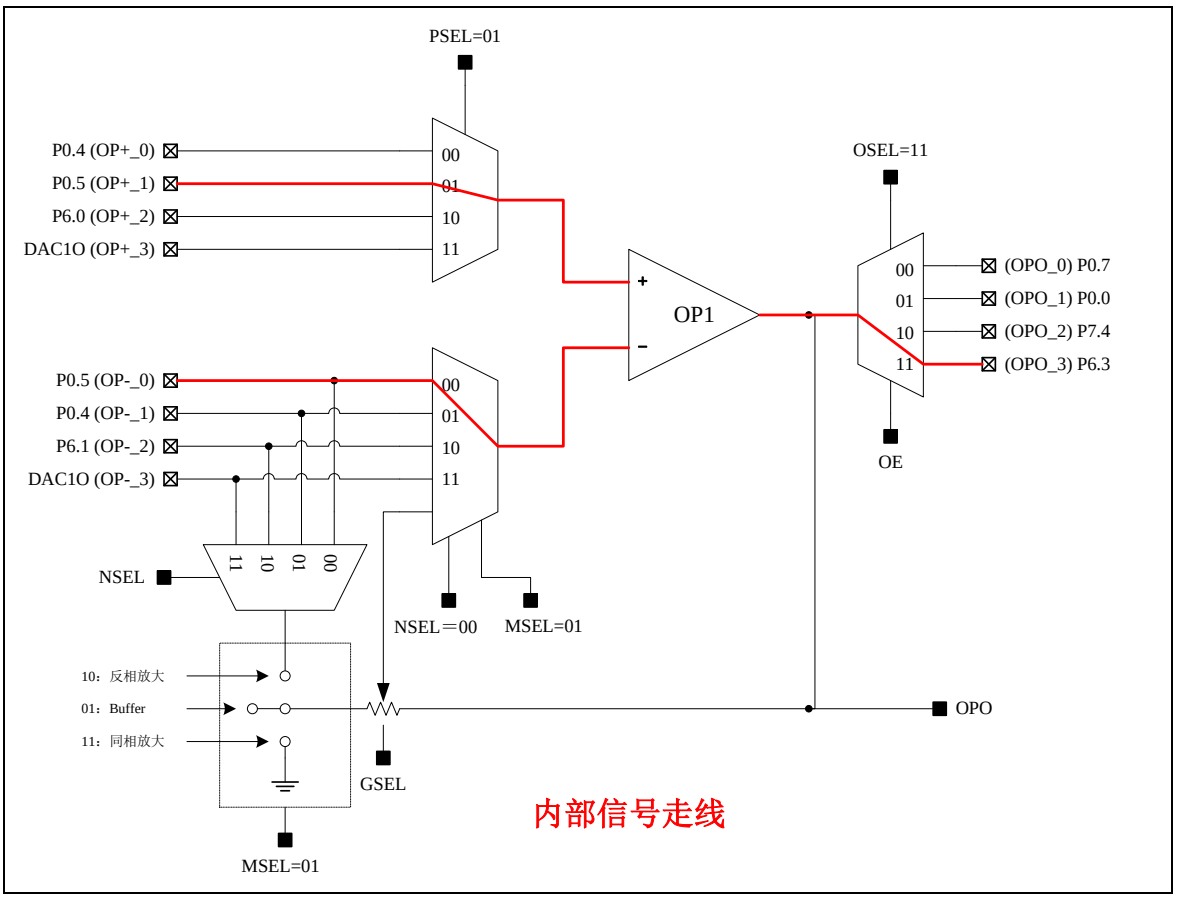
一般作电流转电压时的应用，可以用此电路 (如烟感 ….)

我们的芯片可以做以下设置

1.　OP 对外部信号进行反相放大，增益电路外接，OP 工作在缓冲模式

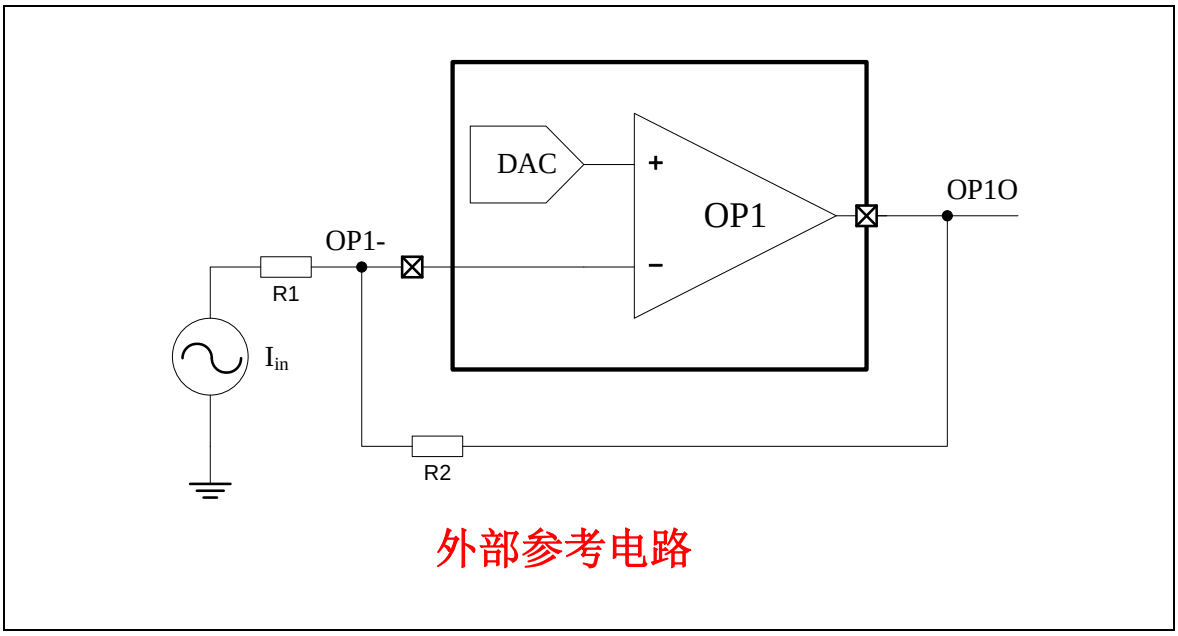


外部参考电路

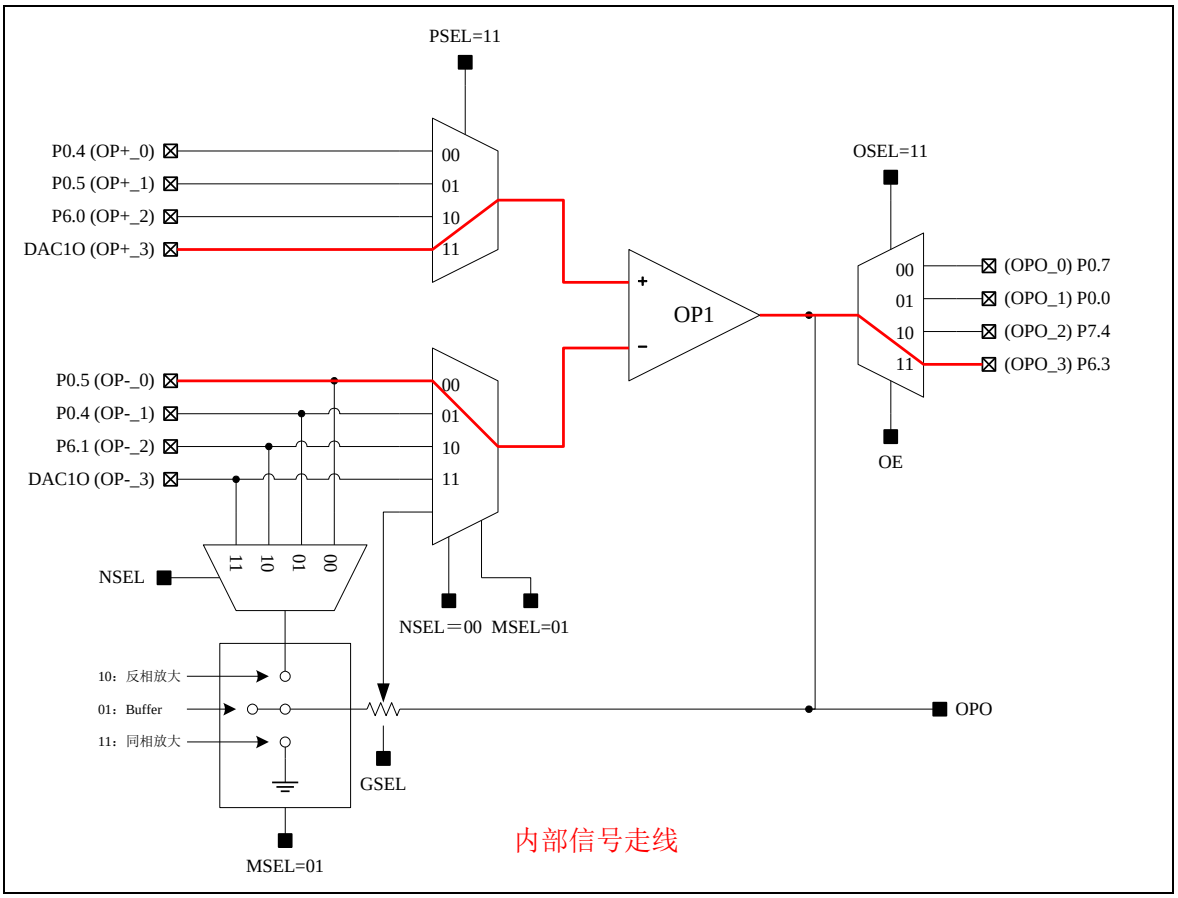


内部信号走线

2. OP 对内部 DAC 输出信号进行反相放大，增益电路外接，OP 工作在缓冲模式

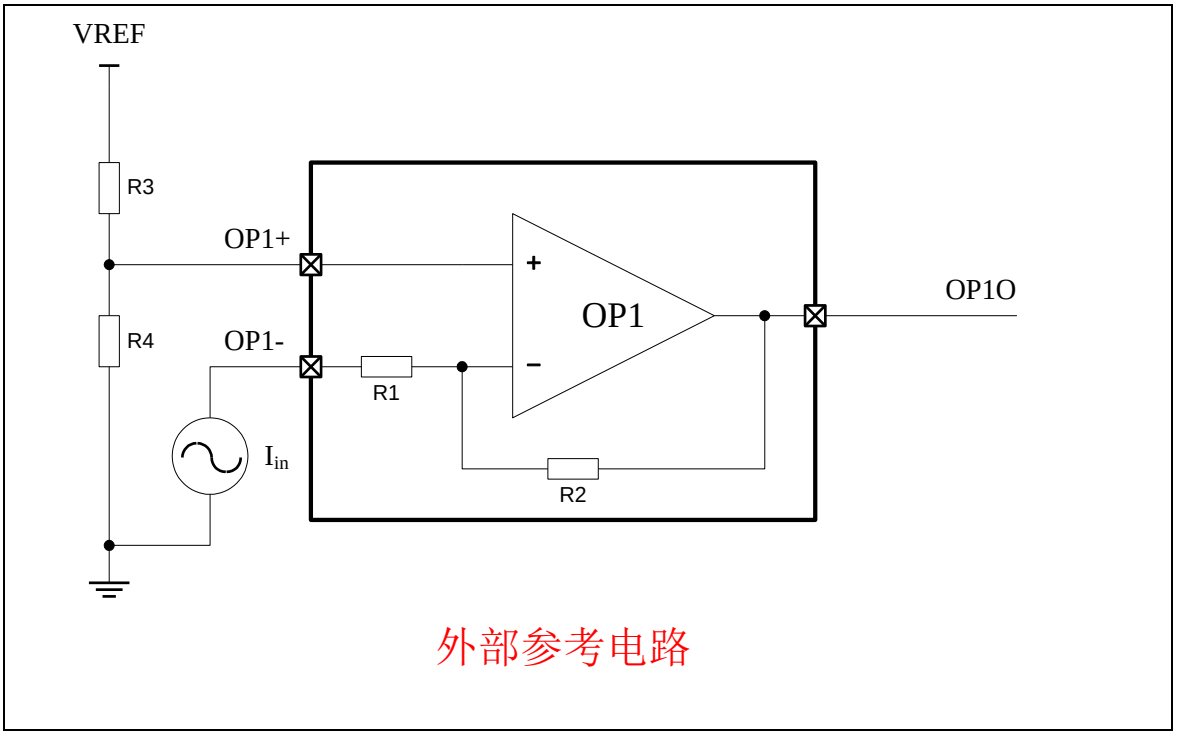


外部参考电路

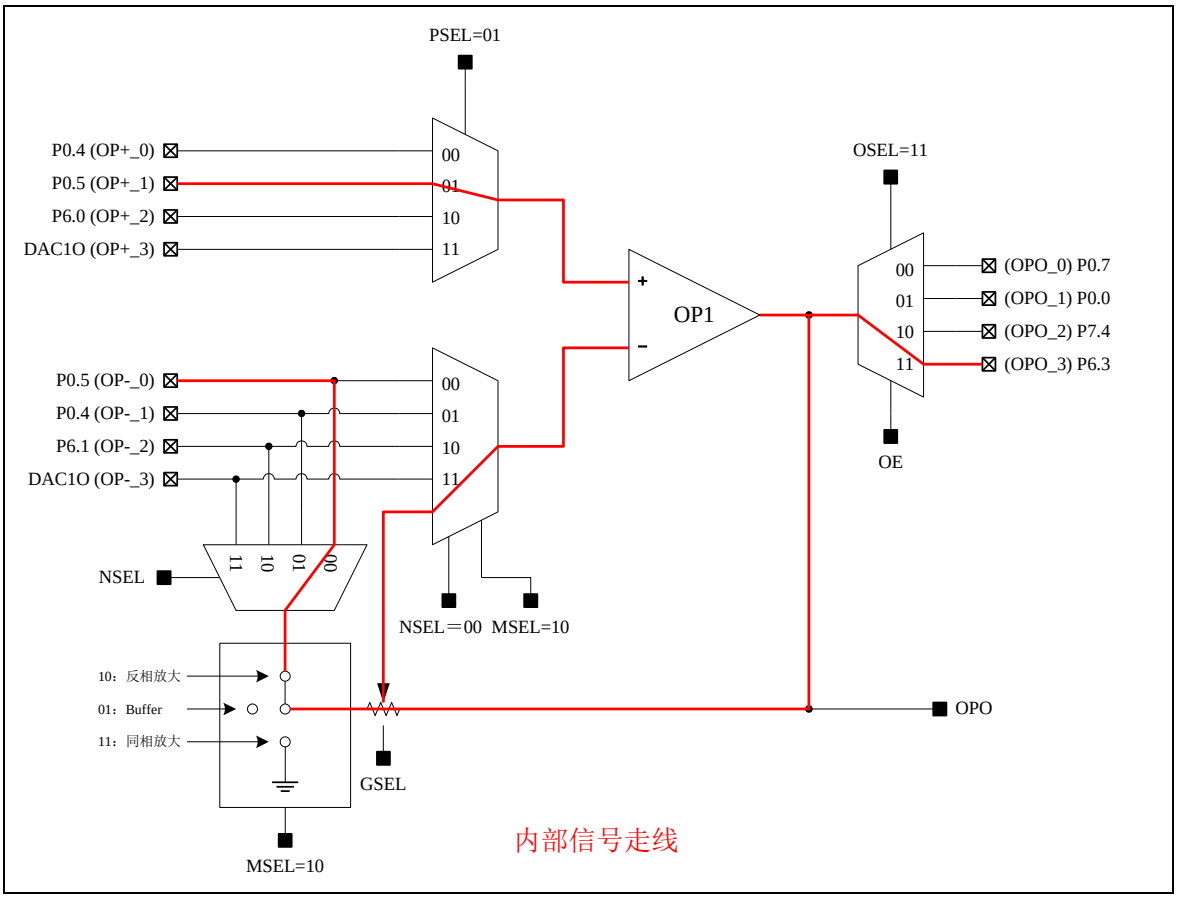


内部信号走线

3. OP 对外部信号进行反相放大，使用内部增益电路，OP 工作在反相放大模式（不建议使用，用内部 DAC 更好)

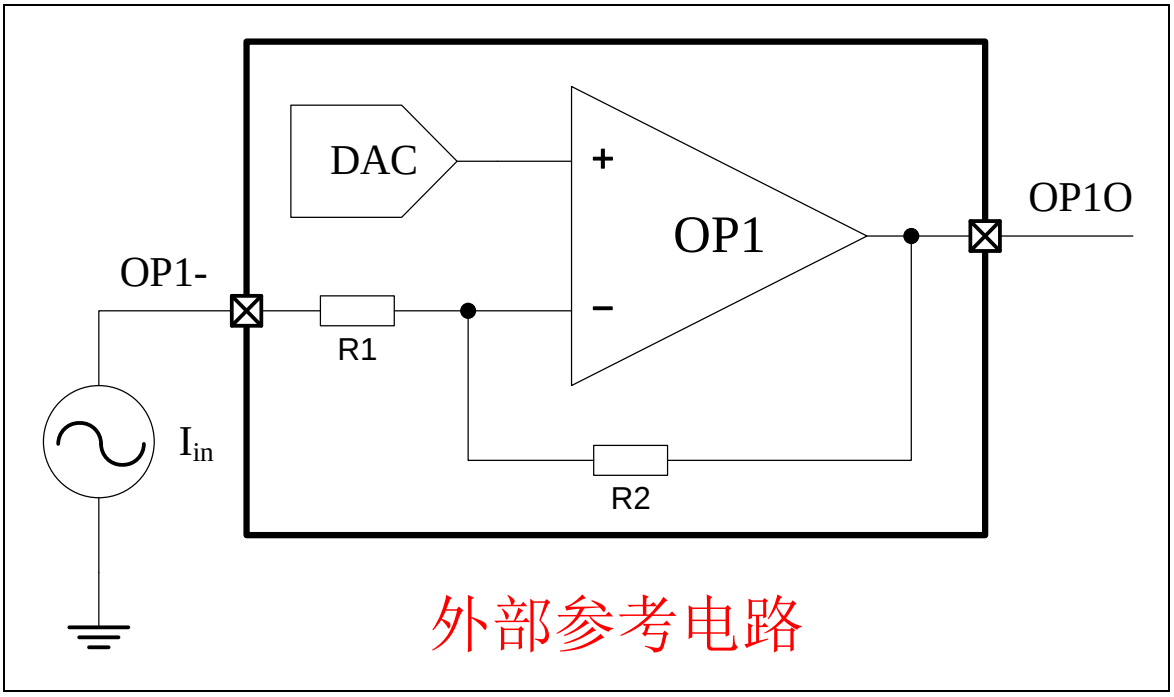


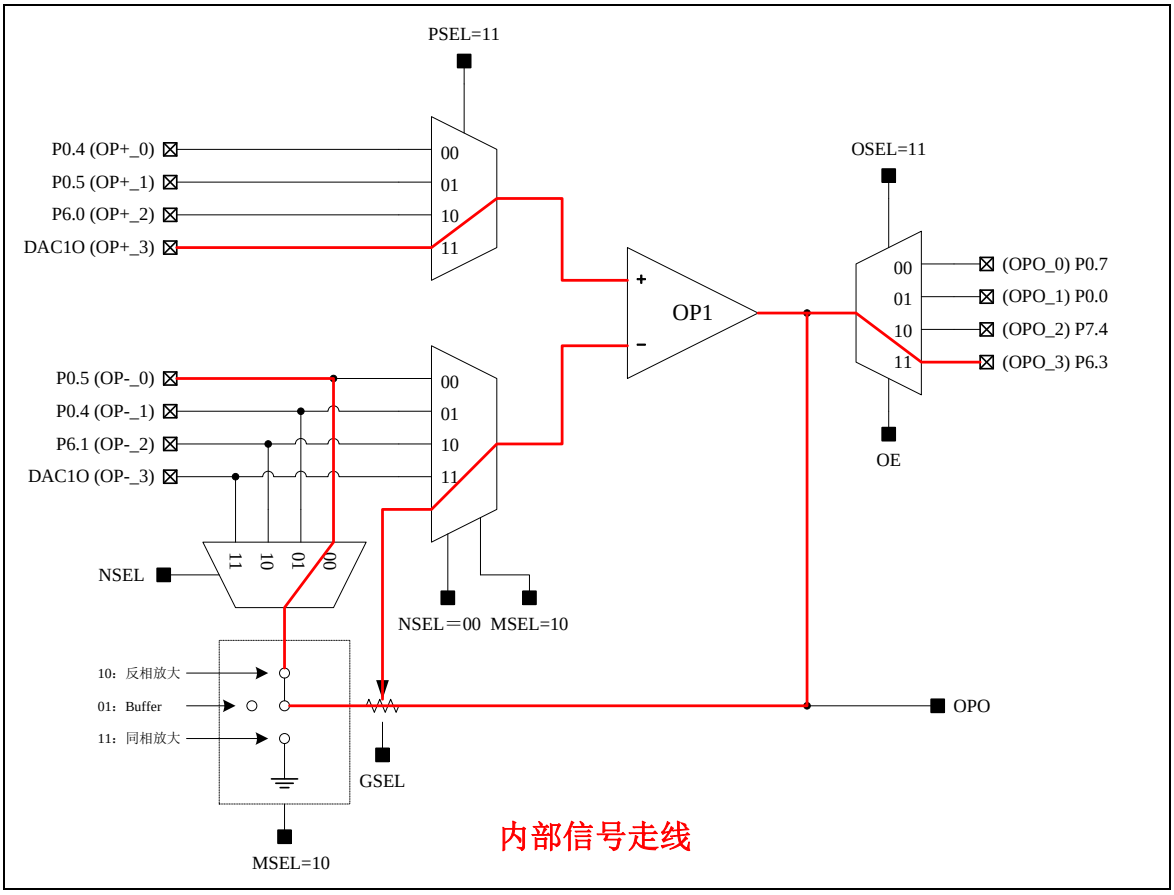
外部参考电路



内部信号走线

4. OP 对内部 DAC 输出信号进行反相放大，使用内部增益电路，OP 工作在反相放大模式

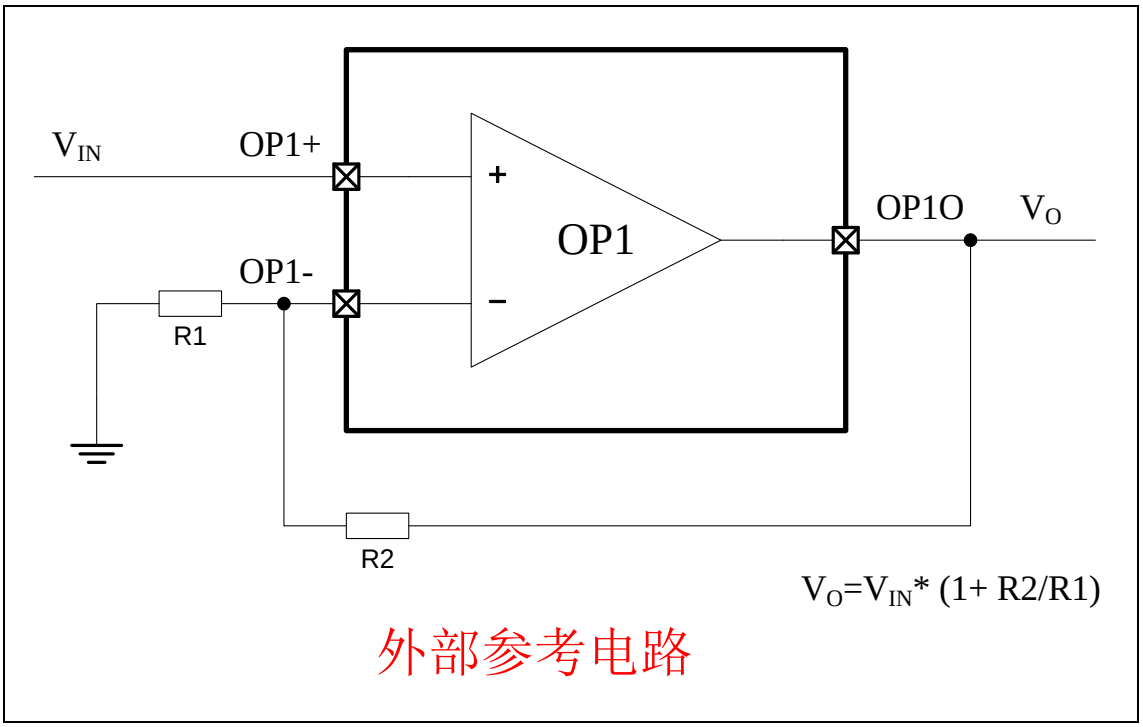


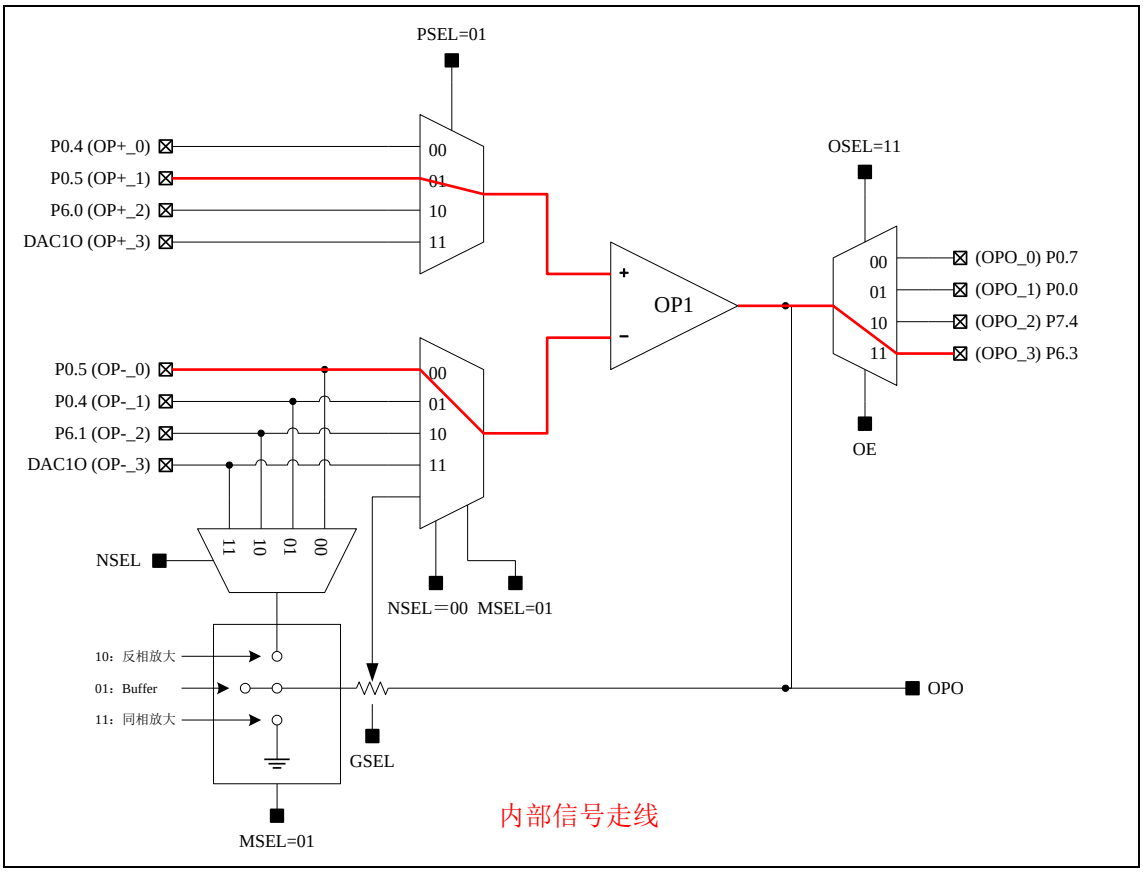


5.2.2　同相/正相放大

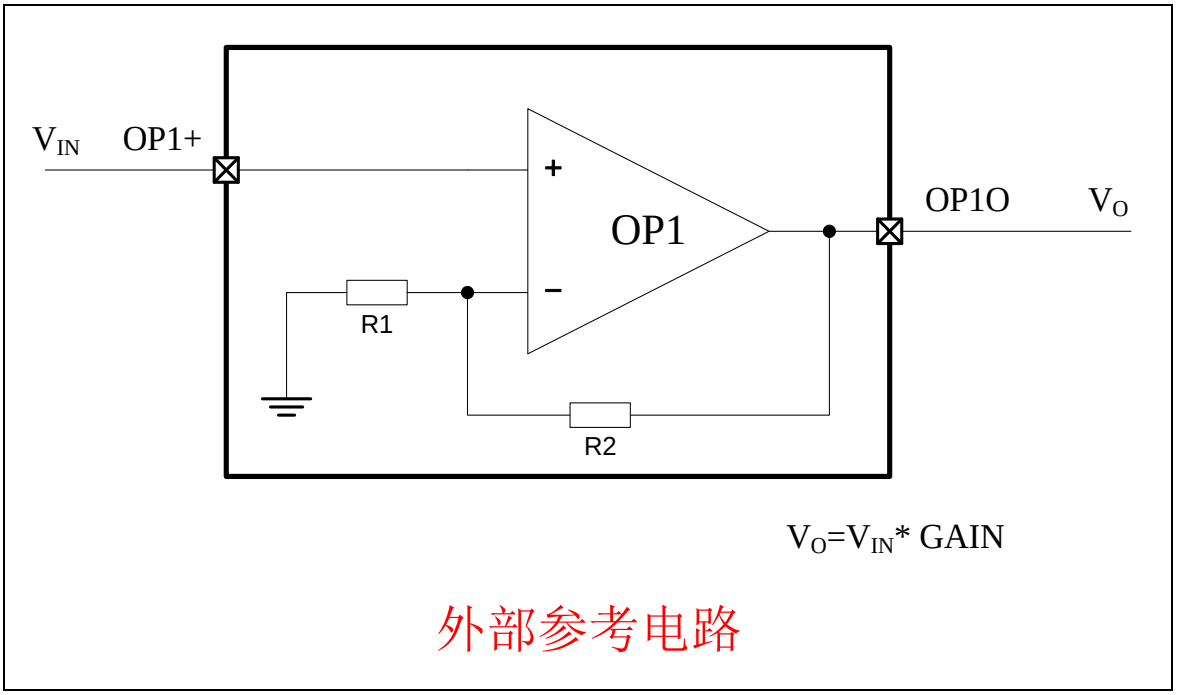
一般作小电压放大时的应用, 可以用此电路如(压力表头 ，耳温枪 ，电流侦测 ，过电流保护)
我们的芯片可以做以下设置

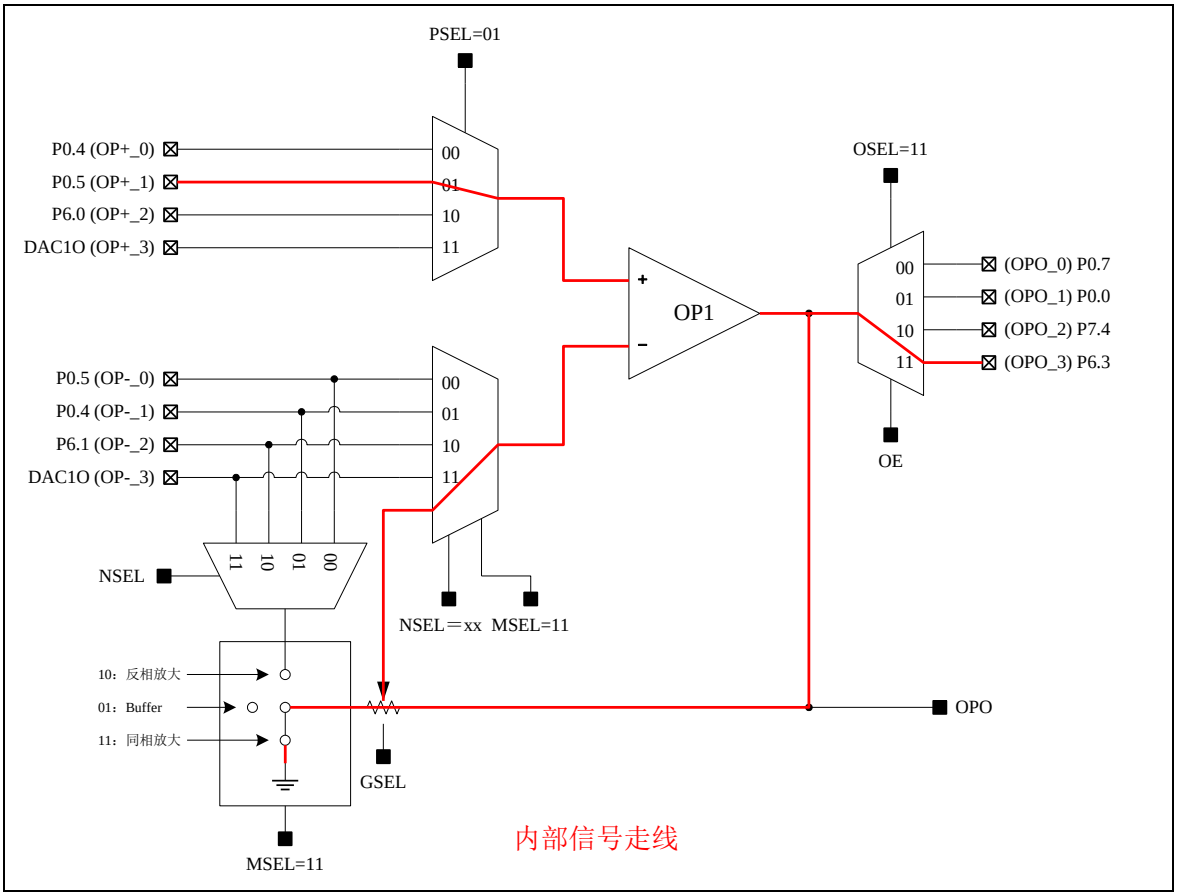
1. OP 对外部信号进行正相放大，增益电路的电阻外接，OP 工作在缓冲模式



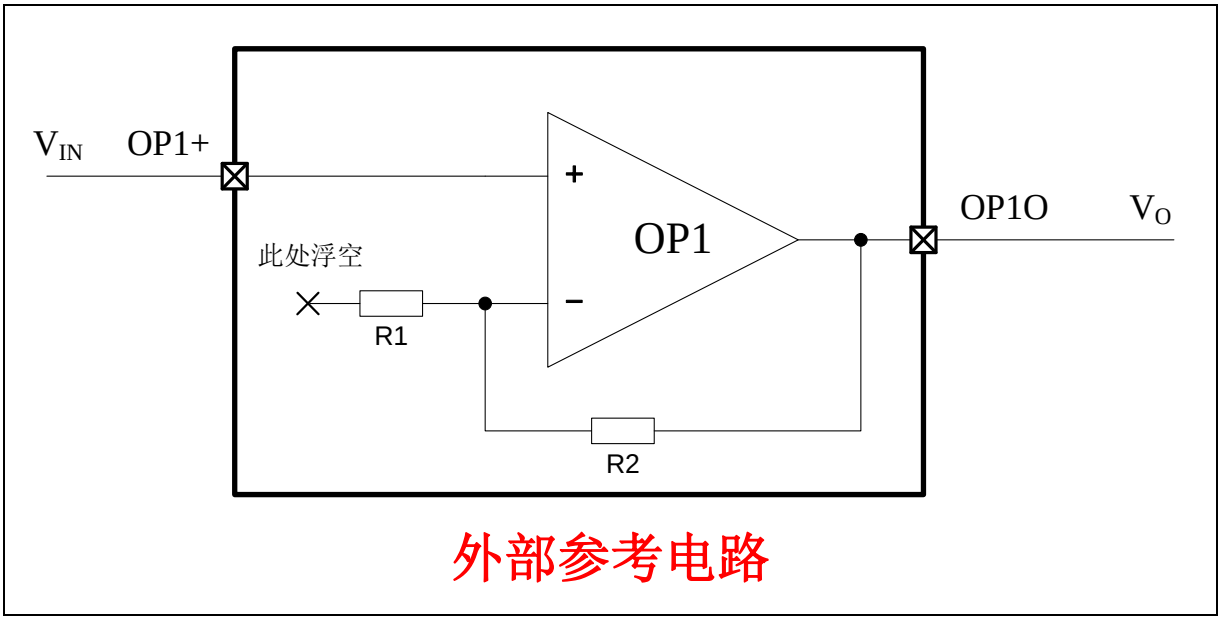


2. OP 对外部信号进行正相放大，使用内部增益电路，OP 工作在正相放大模式

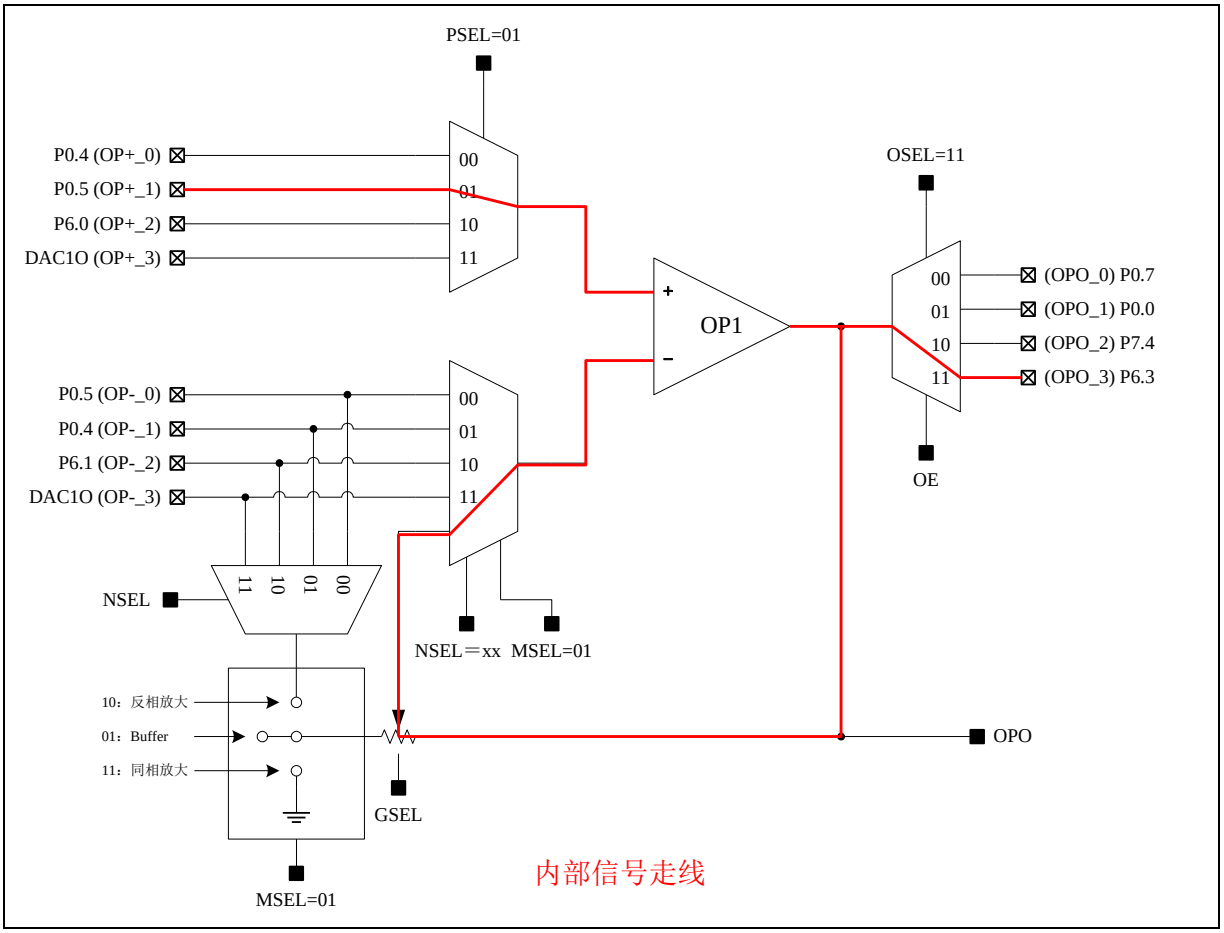




5.2.3 Input BUFFER Mode（输入缓冲模式）

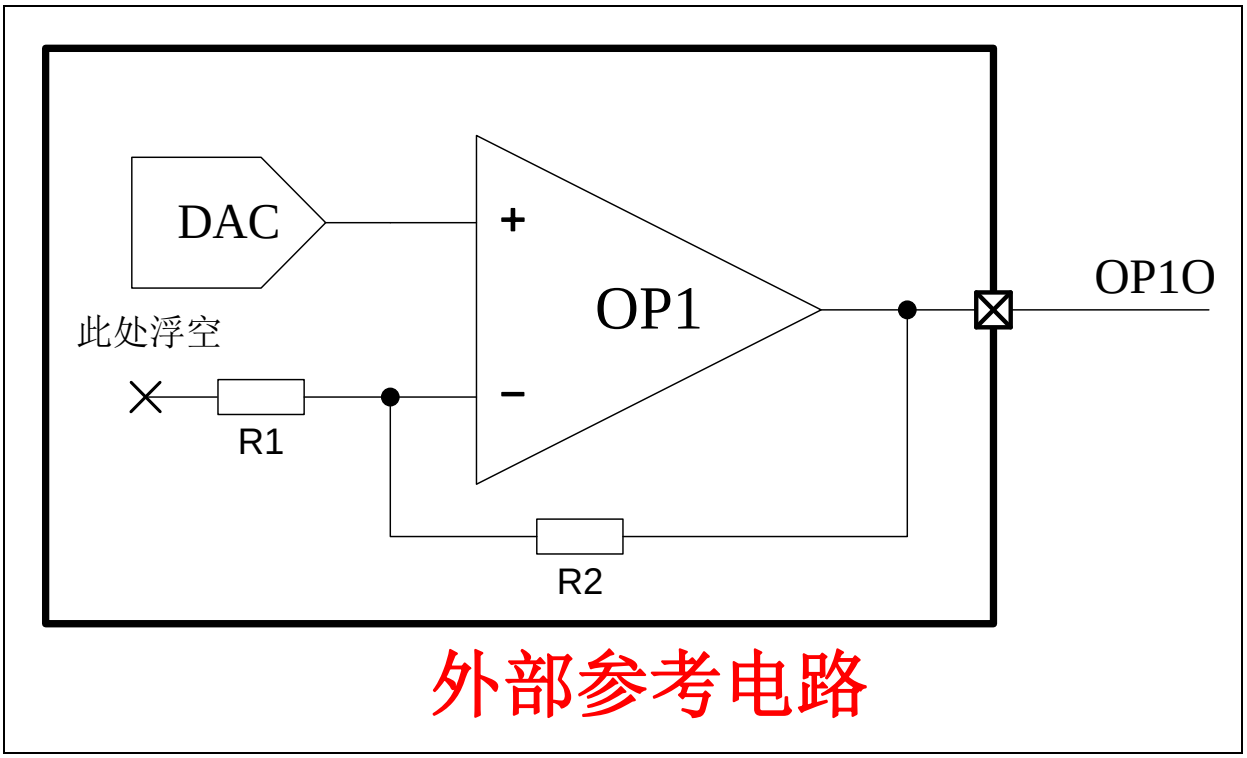


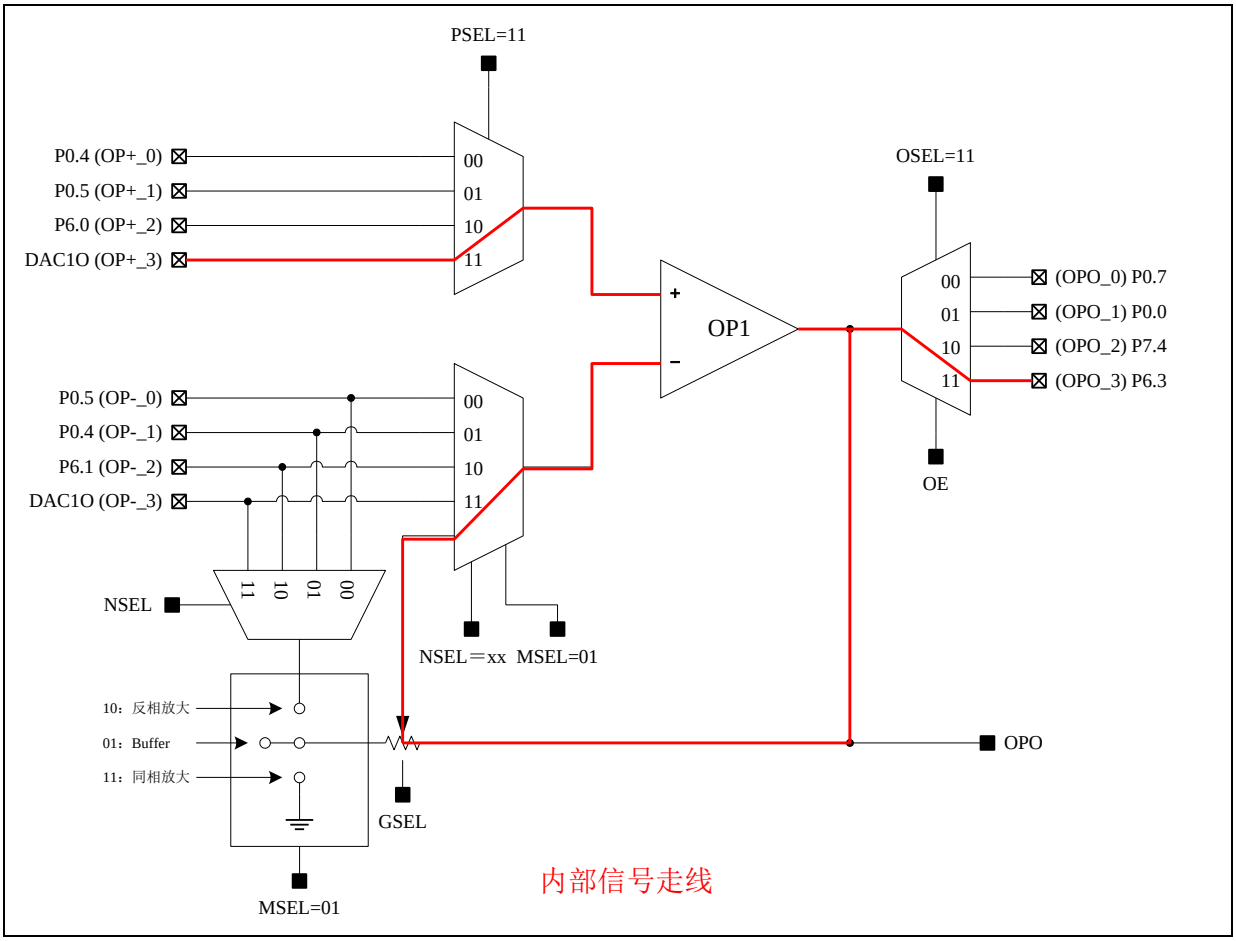
外部参考电路



内部信号走线

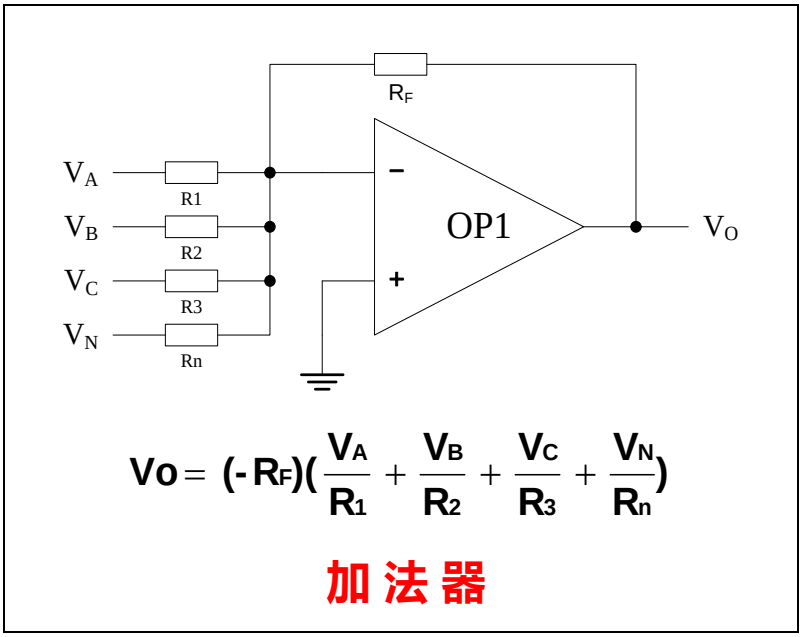
5.2.4 DAC BUFFER Mode（DAC 缓冲模式）

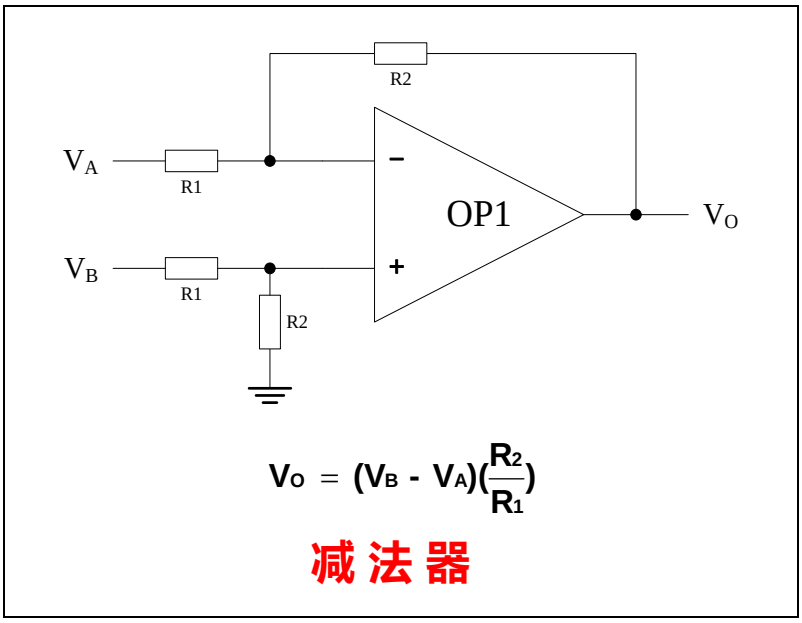


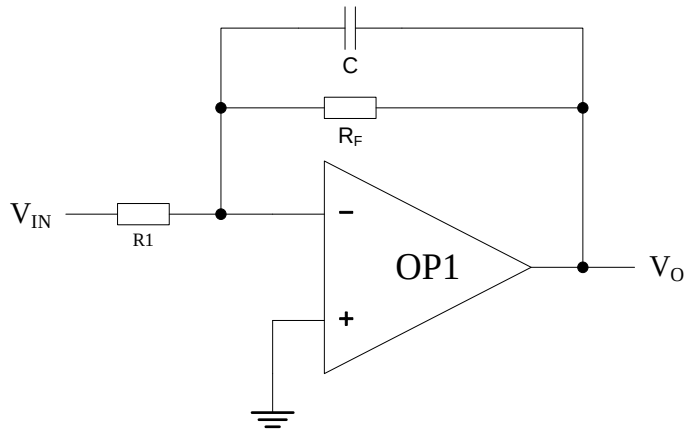


5.2.5　常见的运放应用电路，不是本器件的内部特定电路

Other mode:(建议使用 GP MODE 来外接电路)



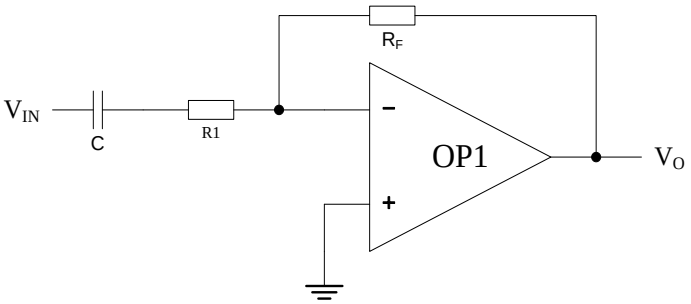




$$T = RC = R_FC$$

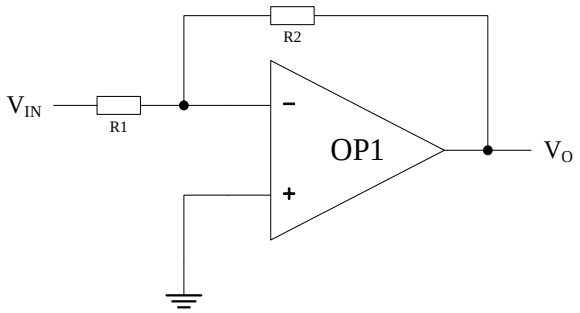
$$V_O = V_{IN}\left(-\frac{R_F}{R_1} \times \frac{1}{1 + j\omega R_FC}\right)$$

低通滤波器/积分器



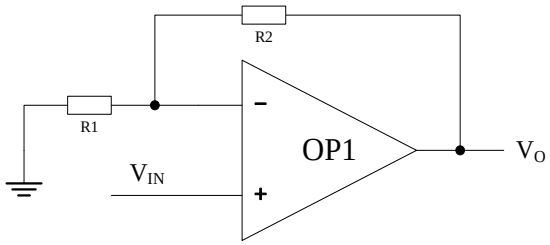
$$V_O = V_{IN}\left(-\frac{R_F}{R_1} \times \frac{1}{1 + \frac{1}{j\omega R_1C}}\right)$$

高通滤波器/微分器



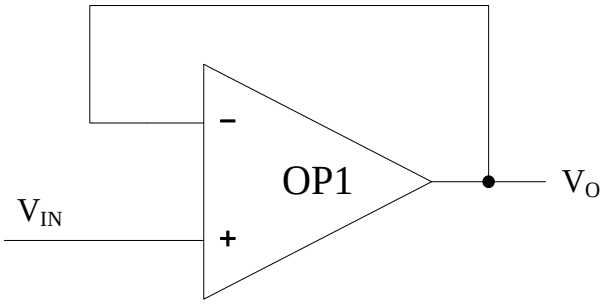
$$V_O = -\frac{R_2}{R_1} V_{IN}$$

反相放大器



$$V_O = \left(1 + \frac{R_2}{R_1}\right) V_{IN}$$

同相放大器



$$V_O = V_{IN}$$

电压随耦器

6 功能脚切换

Ai8051U 系列单片机的特殊外设串口、SPI、PWM、I²C 以及总线控制脚可以在多个 I/O 直接进行切换，以实现一个外设当作多个设备进行分时复用。

6.1 功能脚切换相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
P_SW1	外设端口切换寄存器 1	A2H	S1_S[1:0]		CAN_S[1:0]		SPI_S[1:0]		LIN_S[1:0]		nn00,0000
P_SW2	外设端口切换寄存器 2	BAH	EAXFR	-	I2C_S[1:0]		CMPO_S	S4_S	S3_S	S2_S	0x00,0000
P_SW3	外设端口切换寄存器 3	BBH	I2S_S[1:0]		S2SPI_S[1:0]		S1SPI_S[1:0]		CAN2_S[1:0]		0000,0000
P_SW4	外设端口切换寄存器 4	BFH	-		-		-		QSPI_S[1:0]		xxxx,xx00

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
MCLKOCR	主时钟输出控制寄存器	7EFE05H	MCLKO_S	MCLKODIV[6:0]							0000,0000
PWMA_PS	PWMA 切换寄存器	7EFEB2H	C4PS[1:0]		C3PS[1:0]		C2PS[1:0]		C1PS[1:0]		0000,0000
PWMA_PS2	PWMA 切换寄存器 2	7EFEB8H	-		-		AC6PS[1:0]		AC5PS[1:0]		xxxx,0000
PWMB_PS	PWMB 切换寄存器	7EFEB6H	C8PS[1:0]		C7PS[1:0]		C6PS[1:0]		C5PS[1:0]		0000,0000
PWMA_ETRPS	PWMA 的 ERT 切换寄存器	7EFEB0H						BRKAPS	ETRAPS[1:0]		xxx,x000
PWMB_ETRPS	PWMB 的 ERT 切换寄存器	7EFEB4H						BRKBPS	ETRBPS[1:0]		xxx,x000
CMOD	PCA 模式寄存器	7EFD67H	CIDL	CCP_S[1:0]		CPS[3:0]				ECF	0000,0000
ADCEXCFG	ADC 扩展配置寄存器	7EFEADH	ADCETR_PS[1:0]		ADCETRS[1:0]		-	CVTIMESEL[2:0]			0000,x000

6.1.1 串口 1/SPI，功能脚切换控制（P_SW1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW1	A2H	S1_S[1:0]		-		SPI_S[1:0]		-	

S1_S[1:0]：串口 1 功能脚选择位

S1_S[1:0]	RxD	TxD
00	P3.0	P3.1
01	P3.6	P3.7
10	P1.6	P1.7
11	P4.3	P4.4

SPI_S[1:0]：SPI 功能脚选择位

SPI_S[1:0]	SS	MOSI	MISO	SCLK
00	P1.4	P1.5	P1.6	P1.7
01	P2.4	P2.5	P2.6	P2.7
10	P4.0	P4.1	P4.2	P4.3
11	P3.5	P3.4	P3.3	P3.2

6.1.2 串口 2/3/4/I²C/比较器输出，功能脚切换控制（P_SW2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW2	BAH	EAXFR	-	I2C_S[1:0]		CMPO_S	S4_S	S3_S	S2_S

EAXFR：扩展 RAM 区特殊功能寄存器（XFR）访问控制寄存器

0：禁止访问 XFR

1：使能访问 XFR。

当需要访问 XFR 时，必须先将 EAXFR 置 1，才能对 XFR 进行正常的读写。建议上电初始化时直接设置为 1，后续不要再修改

I2C_S[1:0]：I²C 功能脚选择位

I2C_S[1:0]	SCL	SDA
00	P2.4	P2.3
01	P1.5	P1.4
10	-	-
11	P3.2	P3.3

CMPO_S：比较器输出脚选择位

CMPO_S	CMPO
0	P4.5
1	P4.1

S4_S：串口 4 功能脚选择位

S4_S	RxD4	TxD4
0	P0.2	P0.3
1	P5.2	P5.3

S3_S：串口 3 功能脚选择位

S3_S	RxD3	TxD3
0	P0.0	P0.1
1	P5.0	P5.1

S2_S：串口 2 功能脚选择位

S2_S	RxD2	TxD2
0	P1.2	P1.3
1	P4.2	P4.3

6.1.3 I2S/串口 1 的 SPI/串口 2 的 SPI，功能脚切换控制（P_SW3）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW3	BBH	I2S_S		S2SPI_S[1:0]		S1SPI_S[1:0]			

S2SPI_S[1:0]：USART2 的 SPI 功能脚选择位（适用于 USART2 的 SPI 模式）

S2SPI_S[1:0]	S2SS	S2MOSI	S2MISO	S2SCLK
00	P1.4	P1.5	P1.6	P1.7
01	P2.4	P2.5	P2.6	P2.7
10	P4.0	P4.1	P4.2	P4.3
11	P3.5	P3.4	P3.3	P3.2

S1SPI_S[1:0]：USART1 的 SPI 功能脚选择位（适用于 USART1 的 SPI 模式）

S1SPI_S[1:0]	S1SS	S1MOSI	S1MISO	S1SCLK
00	P1.4	P1.5	P1.6	P1.7
01	P2.4	P2.5	P2.6	P2.7
10	P4.0	P4.1	P4.2	P4.3
11	P3.5	P3.4	P3.3	P3.2

I2S_S[1:0]：I2S 功能脚选择位

I2S_S[1:0]	I2SBCK	I2SMCK	I2SData	I2SLRCK
00	P3.2	P3.3	P3.4	P3.5
01	P1.7	P1.6	P1.5	P1.4
10	P2.3	P2.2	P2.1	P2.0
11	P4.3	P1.6	P4.1	P4.0

6.1.4 QSPI，功能脚切换控制（P_SW4）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW4	BFH	-		-		-		QSPI_S[1:0]	

QSPI_S[1:0]：QSPI 功能脚选择位

QSPI_S[1:0]	QSPINCS	QSPIIO0	QSPIIO1	QSPIIO2	QSPIIO3	QSPICLK
00	P1.4	P1.5	P1.6	P1.3	P1.2	P1.7n
01	P4.0	P4.1	P4.2	P5.2	P5.3	P4.3
10	P4.7	P2.5	P2.6	P4.6	P4.5	P2.7
11						

6.1.5 SPI 的 MOSI 和 MISO 脚交换控制（HSSPI_CFG2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
HSSPI_CFG2	7EFBF9H	-	IOSW	HSSPIEN	FIFOEN	SS_DACT[3:0]			

IOSW：交换 MOSI 和 MISO 脚位（普通 SPI 模式和高速 SPI 模式通用）

0：不交换，维持上电默认脚位。

1：交换 MOSI 和 MISO 的脚位。

6.1.6 主时钟输出脚，输出选择寄存器（MCLKOCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
MCLKOCR	7EFE07H	MCLKO_S	MCLKODIV[6:0]						

MCLKO_S：主时钟输出脚选择位

MCLKO_S	MCLKO
0	P4.7
1	P5.6

6.1.7 PCA/CCP 输出/捕获，功能脚切换控制（CMOD）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMOD	7EFD67H	CIDL	CCP_S[1:0]		CPS[3:0]				ECF

CCP_S[1:0]：PCA 功能脚选择位

CCP_S[1:0]	ECI	CCP0	CCP1	CCP2
00	P1.2	P1.3	P1.4	P1.1
01	P4.1	P4.2	P4.3	P4.4
10	P2.3	P2.0	P2.1	P2.2
11	-	-	-	-

6.1.8 高级 PWM 输出脚/外部捕获脚，切换控制（PWMn_PS）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_PS	7EFEB2H	C4PS[1:0]		C3PS[1:0]		C2PS[1:0]		C1PS[1:0]	
PWMA_PS2	7EFEB8H	-		-		AC6PS[1:0]		AC5PS[1:0]	
PWMB_PS	7EFEB6H	C8PS[1:0]		C7PS[1:0]		C6PS[1:0]		C5PS[1:0]	

C1PS[1:0]：高级 PWM 通道 1 输出脚选择位

C1PS[1:0]	PWM1P	PWM1N
00	P1.0	P1.1
01	P0.0	P0.1
10	P2.0	P2.1
11	-	-

C2PS[1:0]：高级 PWM 通道 2 输出脚选择位

C2PS[1:0]	PWM2P	PWM2N
00	P1.2	P1.3
01	P0.2	P0.3
10	P2.2	P2.3
11	-	-

C3PS[1:0]：高级 PWM 通道 3 输出脚选择位

C3PS[1:0]	PWM3P	PWM3N
00	P1.4	P1.5
01	P0.4	P0.5
10	P2.4	P2.5
11	-	-

C4PS[1:0]：高级 PWM 通道 4 输出脚选择位

C4PS[1:0]	PWM4P	PWM4N
00	P1.6	P1.7
01	P0.6	P0.7
10	P2.6	P2.7
11	-	-

AC5PS[1:0]：高级 PWMA 通道 5 输出脚选择位

AC5PS[1:0]	PWMAPS5
00	P3.3
01	P4.4
10	P5.6
11	-

AC6PS[1:0]：高级 PWMA 通道 6 输出脚选择位

AC6PS[1:0]	PWMAPS6
00	P3.4
01	P4.7
10	P5.7
11	-

C5PS[1:0]：高级 PWM 通道 5 输出脚选择位

C5PS[1:0]	PWM5
00	P0.1
01	P1.1
10	P2.1
11	P5.0

C6PS[1:0]：高级 PWM 通道 6 输出脚选择位

C6PS[1:0]	PWM6
00	P0.3
01	P1.3
10	P2.3
11	P5.1

C7PS[1:0]：高级 PWM 通道 7 输出脚选择位

C7PS[1:0]	PWM7
00	P0.5
01	P1.5
10	P2.5
11	P5.2

C8PS[1:0]：高级 PWM 通道 8 输出脚选择位

C8PS[1:0]	PWM8
00	P0.7
01	P1.7
10	P2.7
11	P5.3

6.1.9 高级 PWM 外部触发脚/刹车脚，切换控制（PWMx_ETRPS）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ETRPS	7EFEB0H					BRKAPS[1:0]		ETRAPS[1:0]	
PWMB_ETRPS	7EFEB4H					BRKBPS[1:0]		ETRBPS[1:0]	

ETRAPS[1:0]：高级 PWMA 的外部触发脚 ERI 选择位

ETRAPS [1:0]	PWMETI
00	P3.2
01	P4.1
10	P2.3
11	P1.2

ETRBPS[1:0]：高级 PWMB 的外部触发脚 ERIB 选择位

ETRBPS [1:0]	PWMETI2
00	P3.2
01	P4.1
10	P2.3
11	P1.2

BRKAPS[1:0]：高级 PWMA 的刹车脚 PWMFLT 选择位

BRKAPS[1:0]	PWMFLT
00	P3.5
01	比较器的输出
10	P0.6
11	-

BRKBPS[1:0]：高级 PWMB 的刹车脚 PWMFLT2 选择位

BRKBPS[1:0]	PWMFLT2
00	P3.5
01	比较器的输出
10	P0.6
11	-

6.1.10 ADC 外部触发脚，切换控制（ADCEXCFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADCEXCFG	7EFEADH	ADCETR_PS[1:0]		ADCETRS[1:0]		-	CVTIMESEL[2:0]		

ADCETR_PS[1:0]：ADC 外部触发脚 ADC_ETR 功能脚选择

ADCETR_PS[1:0]	ADC_ETR
00	P4.7
01	P4.0
10	P2.0
11	-

6.2　范例程序

6.2.1　串口 1 切换

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                    //头文件见下载软件

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                           //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    S1_S1 = 0; S1_S0 = 0;               //RXD/P3.0, TXD/P3.1
//  S1_S1 = 0; S1_S0 = 1;               //RXD_2/P3.6, TXD_2/P3.7
//  S1_S1 = 1; S1_S0 = 0;               //RXD_3/P1.6, TXD_3/P1.7
//  S1_S1 = 1; S1_S0 = 1;               //RXD_4/P4.3, TXD_4/P4.4

    while (1);
}
```

6.2.2　串口 2 切换

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                    //头文件见下载软件

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                           //设置取程序代码等待时间，
```

```
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    S2_S = 0;                           //RXD2/P1.2, TXD2/P1.3
//  S2_S = 1;                           //RXD2_2/P4.2, TXD2_2/P4.3

    while (1);
}
```

6.2.3　串口 3 切换

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                   //头文件见下载软件

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                           //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    S3_S = 0;                           //RXD3/P0.0, TXD3/P0.1
//  S3_S = 1;                           //RXD3_2/P5.0, TXD3_2/P5.1
```

```
    while (1);
}
```

6.2.4 串口 4 切换

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                    //头文件见下载软件

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                           //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    S4_S = 0;                           //RXD4/P0.2, TXD4/P0.3
//  S4_S = 1;                           //RXD4_2/P5.2, TXD4_2/P5.3

    while (1);
}
```

6.2.5 SPI 切换

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                    //头文件见下载软件

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                           //设置取程序代码等待时间，
```

```
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    SPI_S1 = 0; SPI_S0 = 0;             //SS/P1.4, MOSI/P1.5, MISO/P1.6, SCLK/P1.7
//  SPI_S1 = 0; SPI_S0 = 1;             //SS_2/P2.4, MOSI_2/P2.5, MISO_2/P2.6, SCLK_2/P2.7
//  SPI_S1 = 1; SPI_S0 = 0;             //SS_3/P4.0, MOSI_3/P4.1, MISO_3/P4.2, SCLK_3/P4.3
//  SPI_S1 = 1; SPI_S0 = 1;             //SS_4/P3.5, MOSI_4/P3.4, MISO_4/P3.3, SCLK_4/P3.2

    while (1);
}
```

6.2.6　I2C 切换

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                   //头文件见下载软件

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                           //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    I2C_S1 = 0; I2C_S0 = 0;             //SCL/P2.4, SDA/P2.3
```

```
//      I2C_S1 = 0; I2C_S0 = 1;                 //SCL_2/P1.5, SDA_2/P1.4
//      I2C_S1 = 1; I2C_S0 = 1;                 //SCL_4/P3.2, SDA_4/P3.3

        while (1);
}
```

6.2.7 比较器输出切换

```
//测试工作频率为11.0592MHz

#include “Ai8051U.H”                            //头文件见下载软件

void main()
{
        EAXFR = 1;                              //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//      P_SW2 |= 0x80;                          //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
        WTST = 0;                               //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
        CKCON = 0;                              //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

        CMPO_S = 0;                             //CMPO/P4.5
//      CMPO_S = 1;                             //CMPO_2/P4.1

        while (1);
}
```

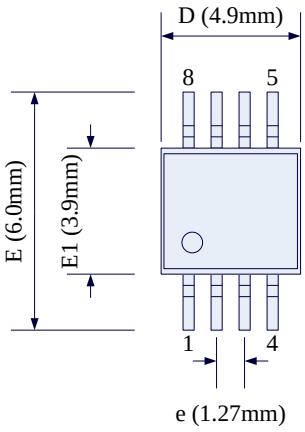
6.2.8 主时钟输出切换

```
//测试工作频率为11.0592MHz

#include “Ai8051U.H”                            //头文件见下载软件

void main()
{
        EAXFR = 1;                              //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
```

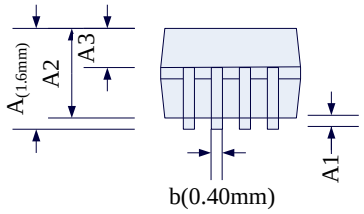
```
//      P_SW2 |= 0x80;                          //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
        WTST = 0;                               //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
        CKCON = 0;                              //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

        MCLKOCR = 0x04;                         //IRC/4 output via MCLKO/P4.7
//      MCLKOCR = 0x84;                         //IRC/4 output via MCLKO_2/P5.6

        while (1);
}
```

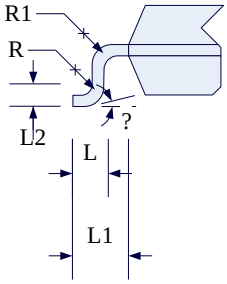
7 封装尺寸图

7.1 SOP8 封装尺寸图

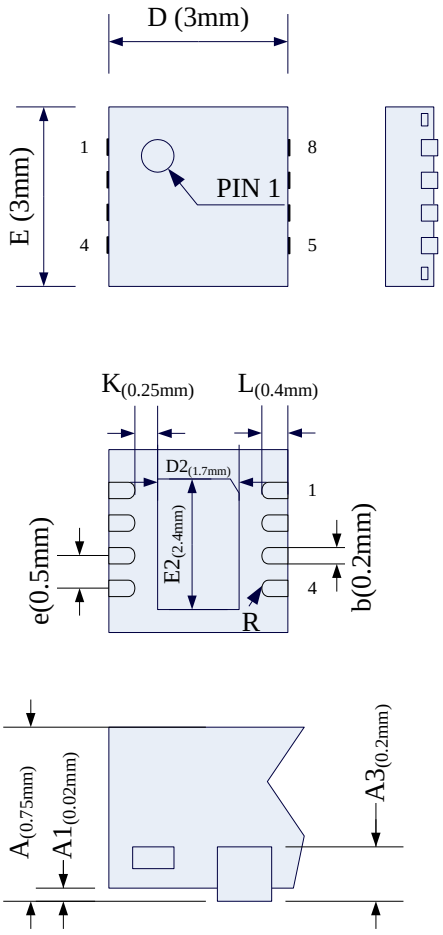


一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	1.35	1.60	1.75
A1	0.10	0.15	0.25
A2	1.25	1.45	1.65
A3	0.55	0.65	0.75
b	0.35	0.40	0.45
D	4.80	4.90	5.00
E	5.80	6.00	6.20
E1	3.80	3.90	4.00
e		1.27BSC	
L	0.45	0.60	0.80
L1		1.04REF	
L2		0.25BSC	
R1	0.07	-	-
R2	0.07	-	-





7.2 DFN8 封装尺寸图（3mm*3mm）

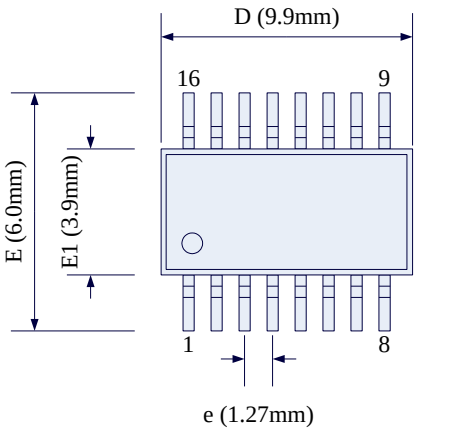


一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	0.70	0.75	0.80
A1	0	0.02	0.05
A2	0.50	0.55	0.60
A3	-	0.203REF	-
b	0.175	0.20	0.225
D	2.89	3.00	3.11
E	2.89	3.00	3.11
D2	1.65	1.70	1.75
E2	2.35	2.40	2.45
e	0.45	0.50	0.55
L	0.381	0.40	0.419
K	0.25REF		
R	0.125	-	-

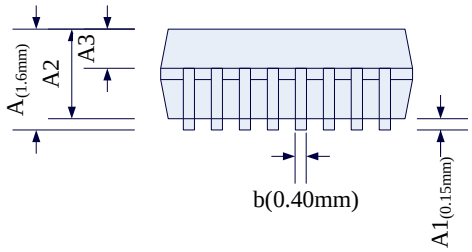
现有 DFN8 封装芯片的背面金属片（衬底），在芯片内部并未接地，在用户的 PCB 板上可以接地，也可以不接地，不会对芯片性能造成影响

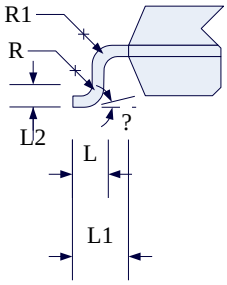
特别说明：造 PCB 的封装库时，焊盘可以向外延伸 0.1～0.2mm，但绝对不能向内延伸，必须确保 K 不能小于 0.25mm，否则会造成管脚短路

7.3 SOP16 封装尺寸图

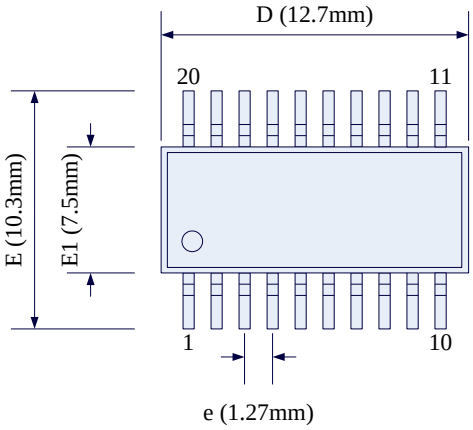


一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	1.35	1.60	1.75
A1	0.10	0.15	0.25
A2	1.25	1.45	1.65
A3	0.55	0.65	0.75
b	0.35	0.40	0.45
D	9.80	9.90	10.00
E	5.80	6.00	6.20
E1	3.80	3.90	4.00
e		1.27BSC	
L	0.45	0.60	0.80
L1		1.04REF	
L2		0.25BSC	
R1	0.07	-	-
R2	0.07	-	-

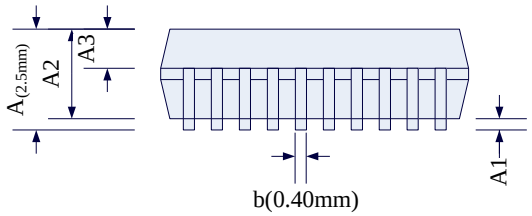


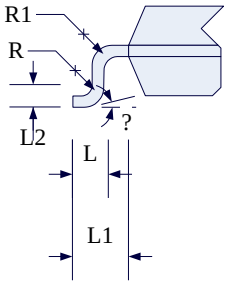


7.4 SOP20 封装尺寸图

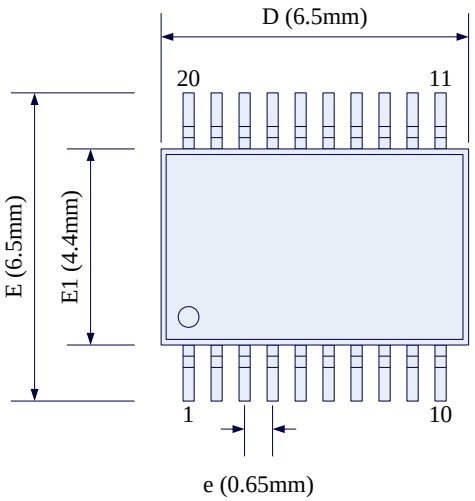


一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	2.465	2.515	2.565
A1	0.10	0.15	0.20
A2	2.10	2.30	2.50
A3	0.55	0.65	0.75
b	0.356	0.406	0.456
D	12.50	12.70	12.90
E	10.20	10.30	10.40
E1	7.45	7.50	7.55
e		1.27BSC	
L	0.80	0.86	0.90
L1		1.40REF	
L2		0.27BSC	
R1	-	0.30	-
R2	-	0.20	-

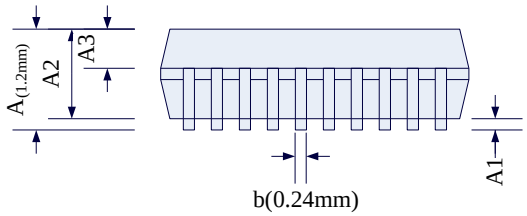


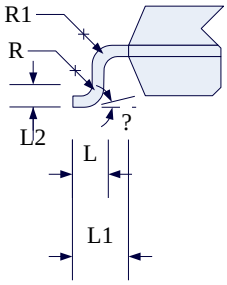


7.5 TSSOP20 封装尺寸图

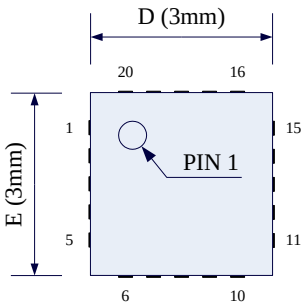


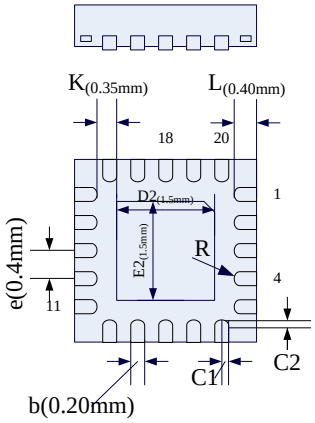
一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	-	-	1.20
A1	0.05	-	0.15
A2	0.90	1.00	1.05
A3	0.34	0.44	0.54
b	0.20	0.24	0.28
D	6.40	6.50	6.60
E	6.20	6.50	6.60
E1	4.30	4.40	4.50
e		0.65BSC	
L	0.45	0.60	0.75
L1		1.00REF	
L2		0.25BSC	
R1	0.09	-	-
R2	0.09	-	-



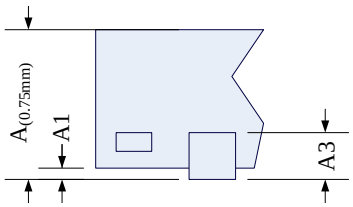


7.6 QFN20 封装尺寸图（3mm*3mm）





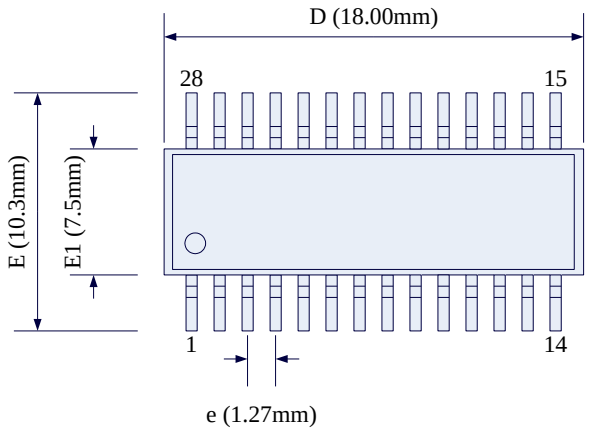
一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	0.70	0.75	0.80
A1	0	0.02	0.05
A2	0.50	0.55	0.60
A3	-	0.20REF	-
b	0.15	0.20	0.25
D	2.90	3.00	3.10
E	2.90	3.00	3.10
D2	1.40	1.50	1.60
E2	1.40	1.50	1.60
e	0.30	0.40	0.50
L	0.35	0.40	0.45
K	0.35REF		
R	0.085	-	-
C1	-	0.07	-
C2	-	0.07	-



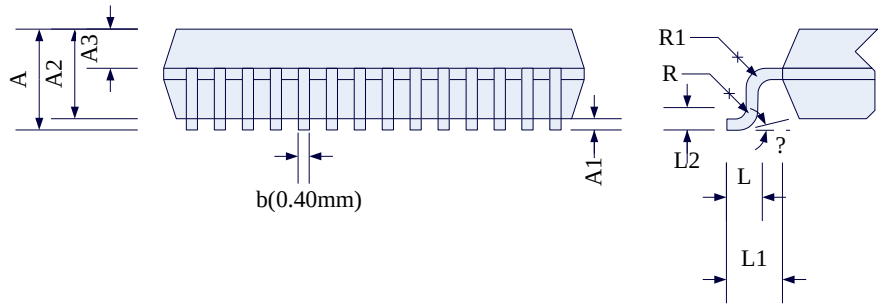
现有 QFN20 封装芯片的背面金属片（衬底），在芯片内部并未接地，在用户的 PCB 板上可以接地，也可以不接地，不会对芯片性能造成影响

特别说明：造 PCB 的封装库时，焊盘可以向外延伸 0.1～0.2mm，但绝对不能向内延伸，必须确保 K 不能小于 0.35mm，否则会造成管脚短路

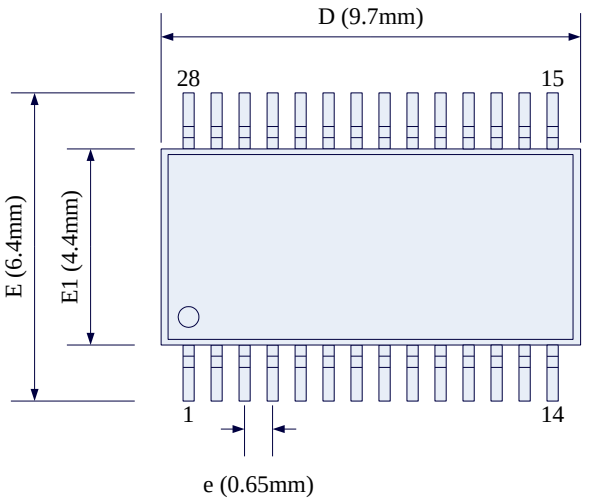
7.7 SOP28 封装尺寸图



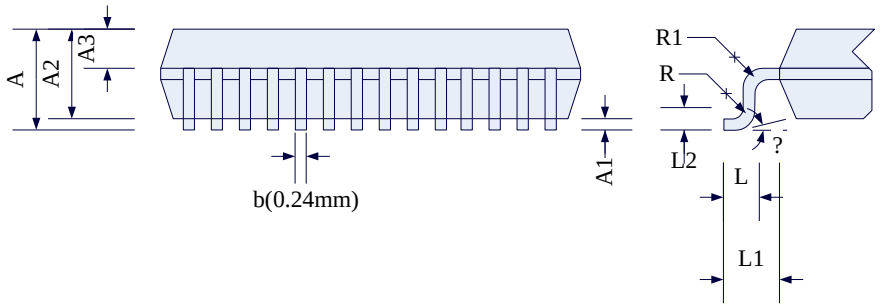
一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	-	-	2.65
A1	0.03	0.08	0.12
A2	2.15	2.20	2.25
A3	0.97	1.02	1.07
b	0.39	-	0.47
D	17.90	18.00	18.10
E	10.10	10.30	10.50
E1	7.40	7.50	7.60
e		1.27BSC	
L	0.70	-	1.00
L1		1.40REF	
L2		0.27BSC	
R1	-	0.30	-
R2	-	0.20	-



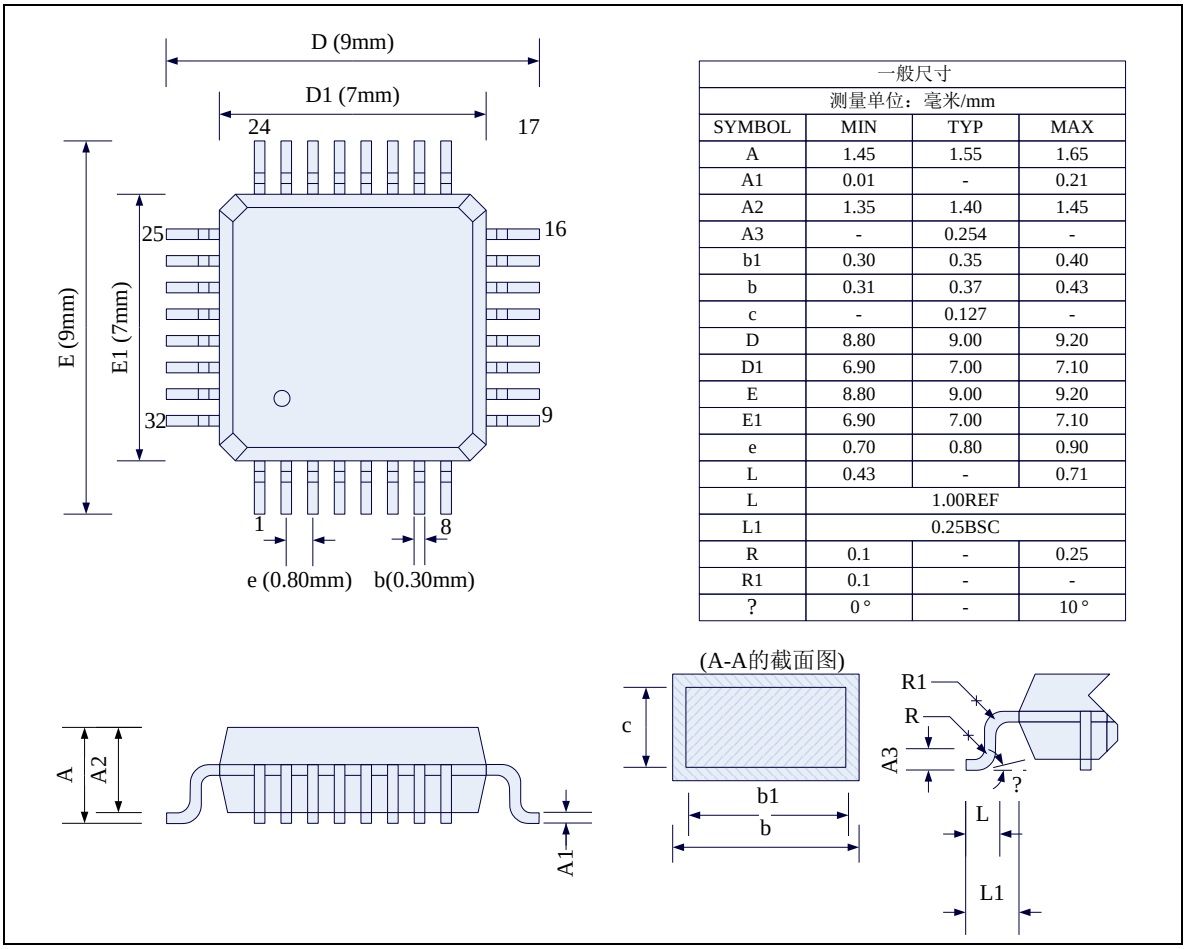
7.8 TSSOP28 封装尺寸图



一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	-	-	1.20
A1	0.03	0.08	0.12
A2	0.80	-	1.00
A3	0.39	0.44	0.49
b	0.20	-	0.29
D	9.60	9.70	9.80
E	6.20	6.40	6.60
E1	4.30	4.40	4.50
e		0.65BSC	
L	0.45	0.60	0.75
L1		1.00REF	
L2		0.25BSC	
R1	0.09	-	-
R2	0.09	-	-

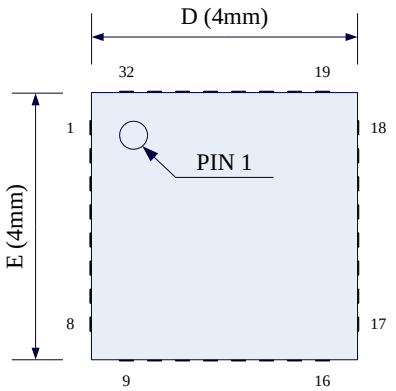


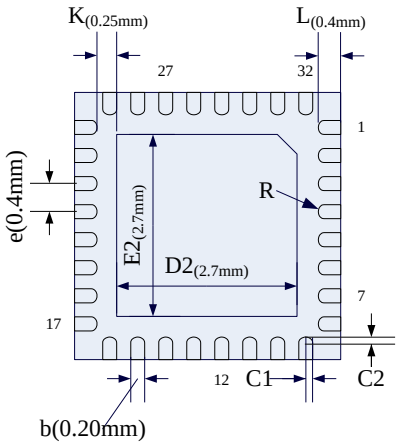
7.9 LQFP32 封装尺寸图（9mm*9mm）



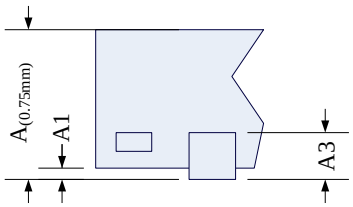
一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	1.45	1.55	1.65
A1	0.01	-	0.21
A2	1.35	1.40	1.45
A3	-	0.254	-
b1	0.30	0.35	0.40
b	0.31	0.37	0.43
c	-	0.127	-
D	8.80	9.00	9.20
D1	6.90	7.00	7.10
E	8.80	9.00	9.20
E1	6.90	7.00	7.10
e	0.70	0.80	0.90
L	0.43	-	0.71
L	1.00REF		
L1	0.25BSC		
R	0.1	-	0.25
R1	0.1	-	-
?	0°	-	10°

7.10 QFN32 封装尺寸图（4mm*4mm）





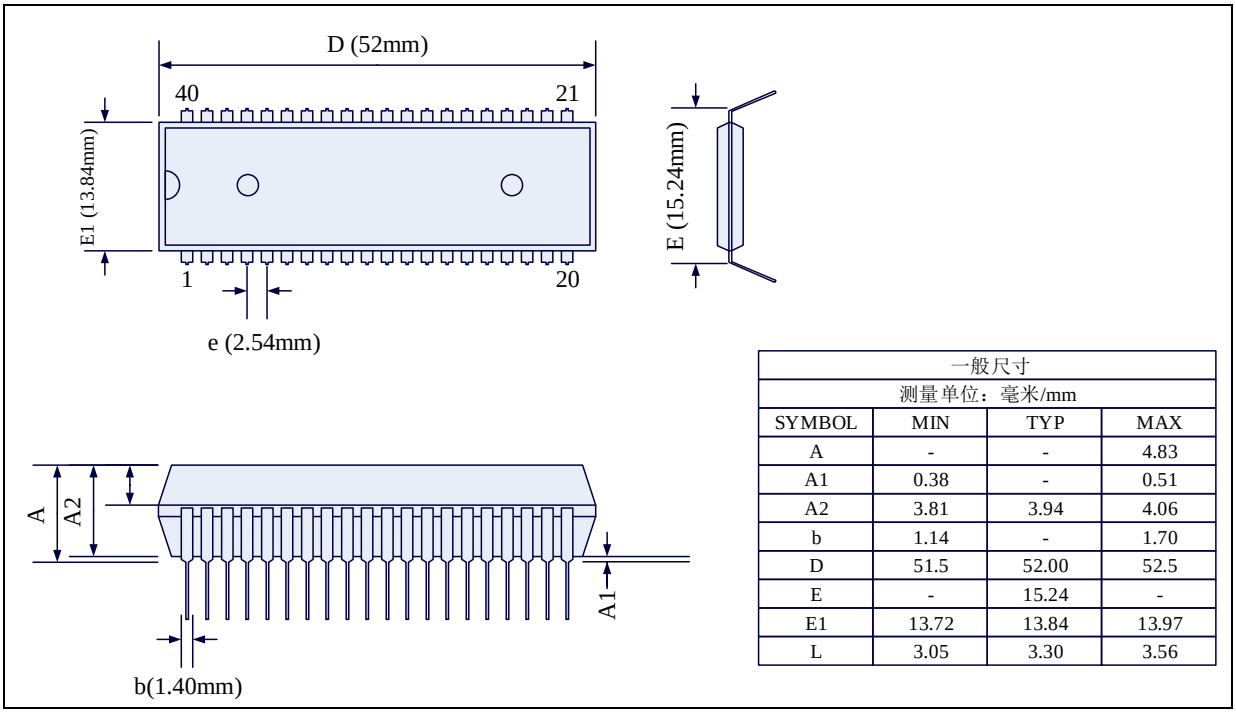
一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	0.70	0.75	0.80
A1	0	0.02	0.05
A2	0.50	0.55	0.60
A3	-	0.20REF	-
b	0.15	0.20	0.25
D	3.90	4.00	4.10
E	3.90	4.00	4.10
D2	2.60	2.70	2.80
E2	2.60	2.70	2.80
e	0.30	0.40	0.50
L	0.35	0.40	0.45
K		0.25REF	
R	0.09	-	-
C1	-	0.16	-
C2	-	0.16	-



现有 QFN32 封装芯片的背面金属片（衬底），在芯片内部并未接地，在用户的 PCB 板上可以接地，也可以不接地，不会对芯片性能造成影响

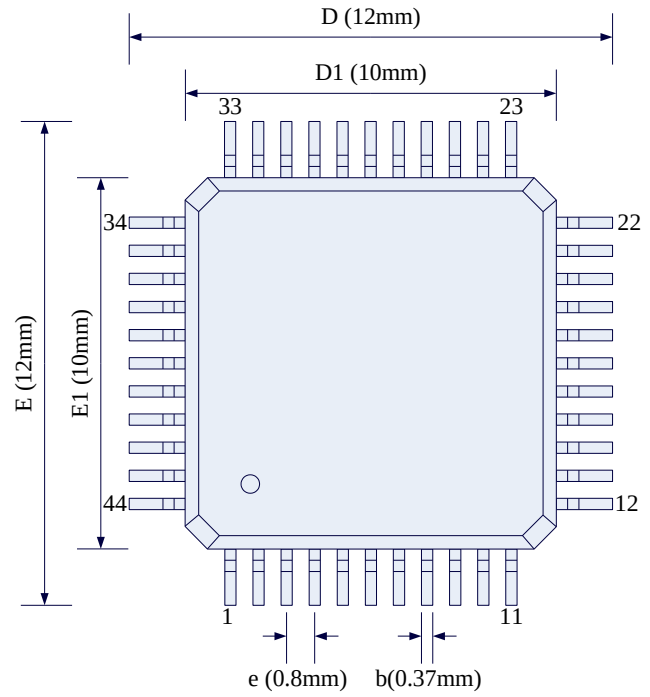
特别说明：造 PCB 的封装库时，焊盘可以向外延伸 0.1～0.2mm，但绝对不能向内延伸，必须确保 K 不能小于 0.25mm，否则会造成管脚短路

7.11 PDIP40 封装尺寸图

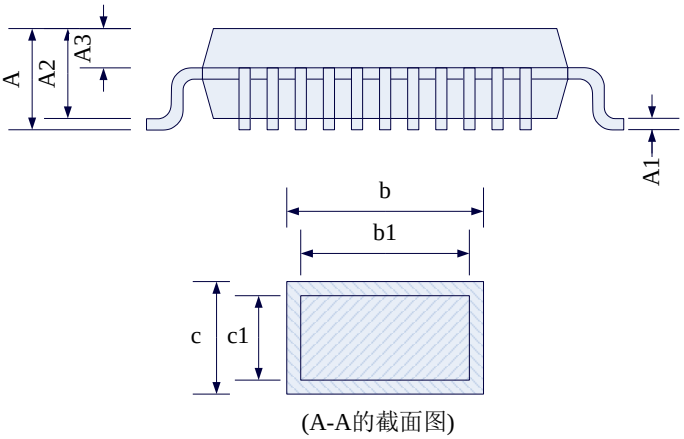


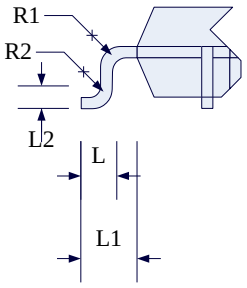
一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	-	-	4.83
A1	0.38	-	0.51
A2	3.81	3.94	4.06
b	1.14	-	1.70
D	51.5	52.00	52.5
E	-	15.24	-
E1	13.72	13.84	13.97
L	3.05	3.30	3.56

7.12 LQFP44/QFP44 封装尺寸图（12mm*12mm）

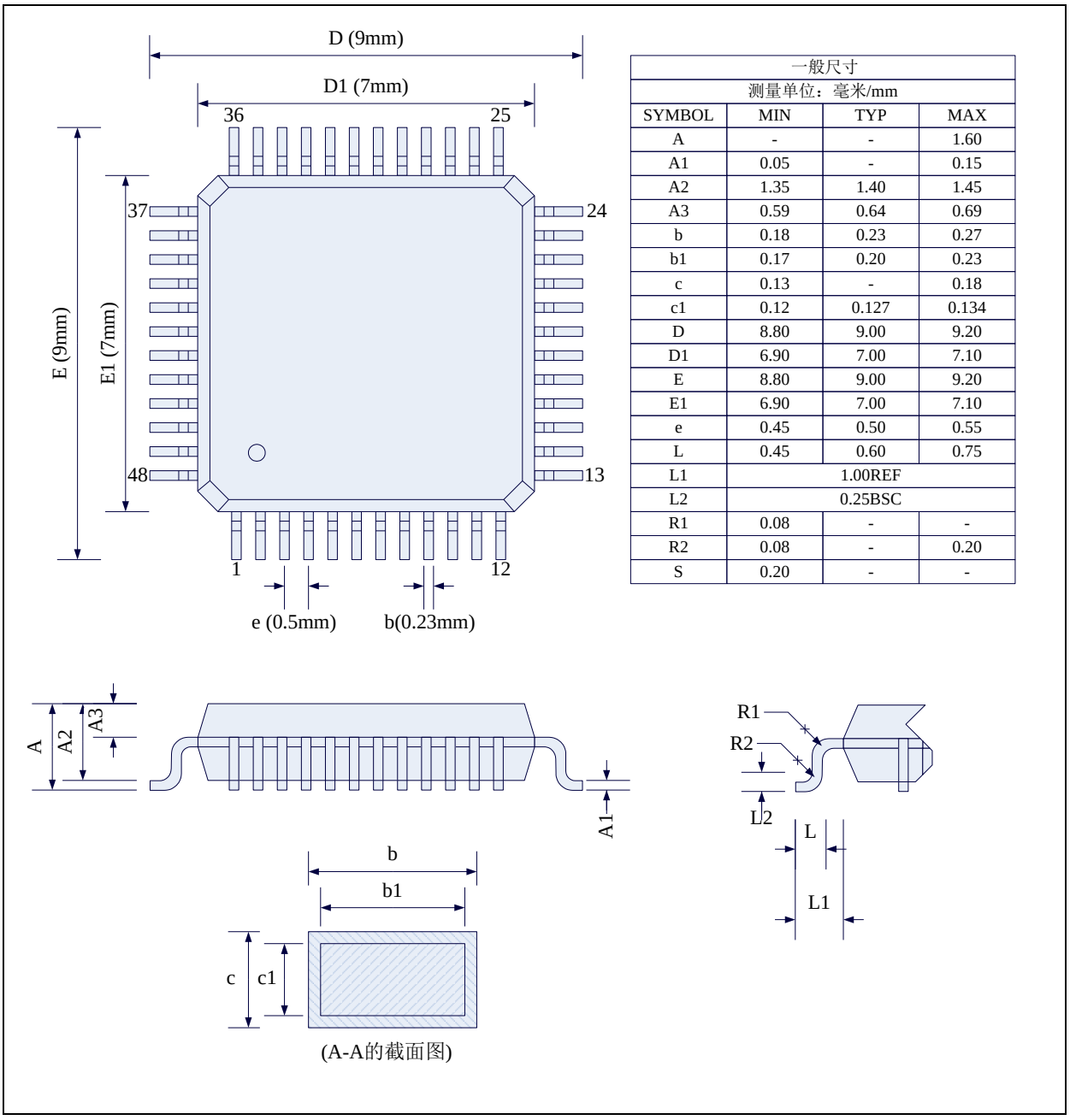


一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	-	-	1.60
A1	0.05	-	0.15
A2	1.35	1.40	1.45
A3	0.59	0.64	0.69
b	0.33	0.37	0.42
b1	0.32	0.35	0.38
c	0.13	-	0.18
c1	0.12	0.13	0.14
D	11.95	12.00	12.05
D1	9.90	10.00	10.10
E	11.95	12.00	12.05
E1	9.90	10.00	10.10
e	0.70	0.80	0.90
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25BSC		
R1	0.08	-	-
R2	0.08	-	0.20
S	0.20	-	-



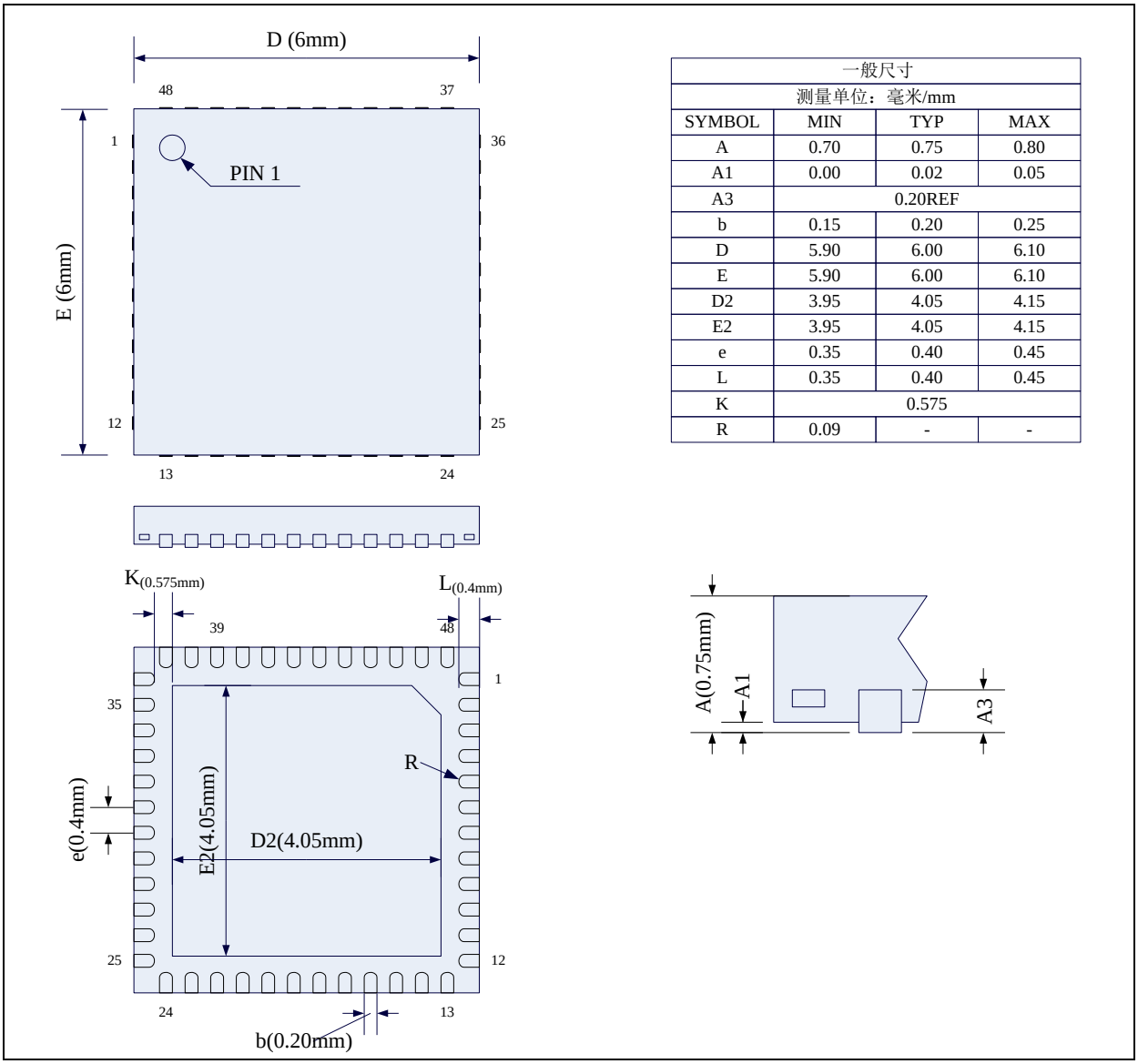


7.13 LQFP48/QFP48 封装尺寸图（9mm*9mm）



一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	-	-	1.60
A1	0.05	-	0.15
A2	1.35	1.40	1.45
A3	0.59	0.64	0.69
b	0.18	0.23	0.27
b1	0.17	0.20	0.23
c	0.13	-	0.18
c1	0.12	0.127	0.134
D	8.80	9.00	9.20
D1	6.90	7.00	7.10
E	8.80	9.00	9.20
E1	6.90	7.00	7.10
e	0.45	0.50	0.55
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25BSC		
R1	0.08	-	-
R2	0.08	-	0.20
S	0.20	-	-

7.14　QFN48 封装尺寸图（6mm*6mm）

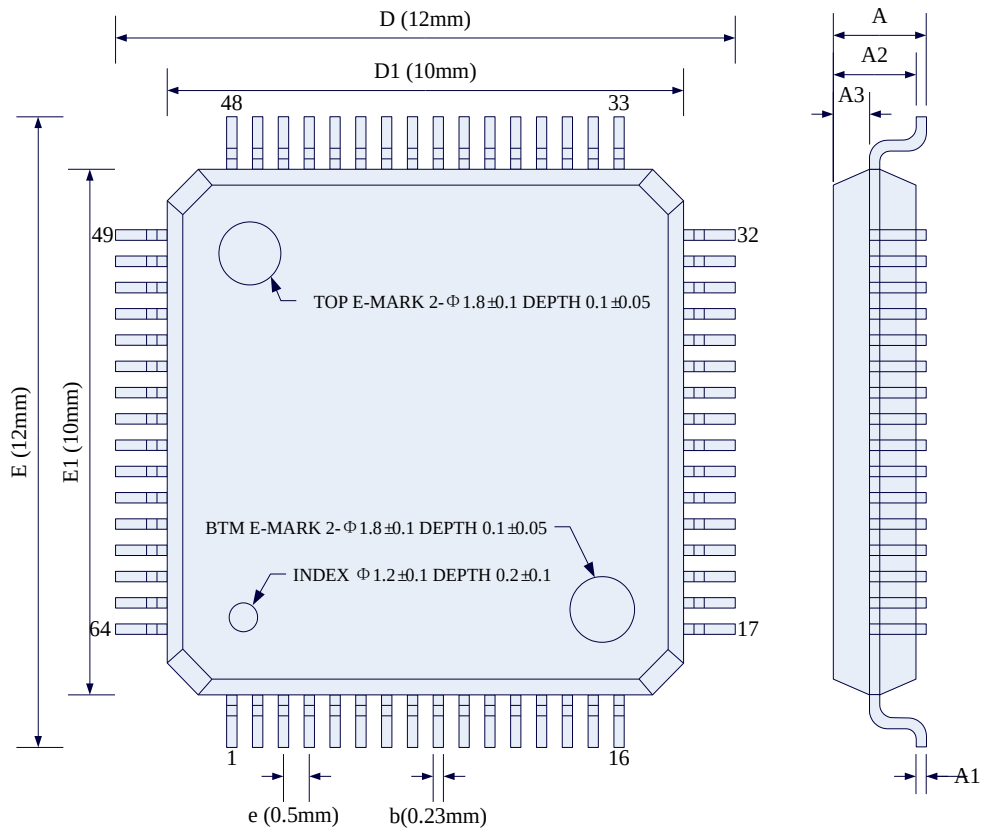


一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	0.70	0.75	0.80
A1	0.00	0.02	0.05
A3		0.20REF	
b	0.15	0.20	0.25
D	5.90	6.00	6.10
E	5.90	6.00	6.10
D2	3.95	4.05	4.15
E2	3.95	4.05	4.15
e	0.35	0.40	0.45
L	0.35	0.40	0.45
K		0.575	
R	0.09	-	-

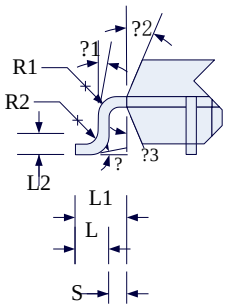
现 Ai8051U 的 QFN48 封装芯片的背面金属片（衬底），接了内部 ADC 的模拟地线，在用户的 PCB 板上可以接模拟地，也可以不接地，不会对芯片性能造成影响。

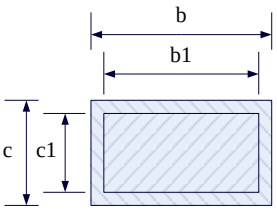
特别说明：造 PCB 的封装库时，焊盘可以向外延伸 0.1～0.2mm，但绝对不能向内延伸，必须确保 K 不能小于 0.2mm，否则会造成管脚短路

7.15 LQFP64 封装尺寸图（12mm*12mm）



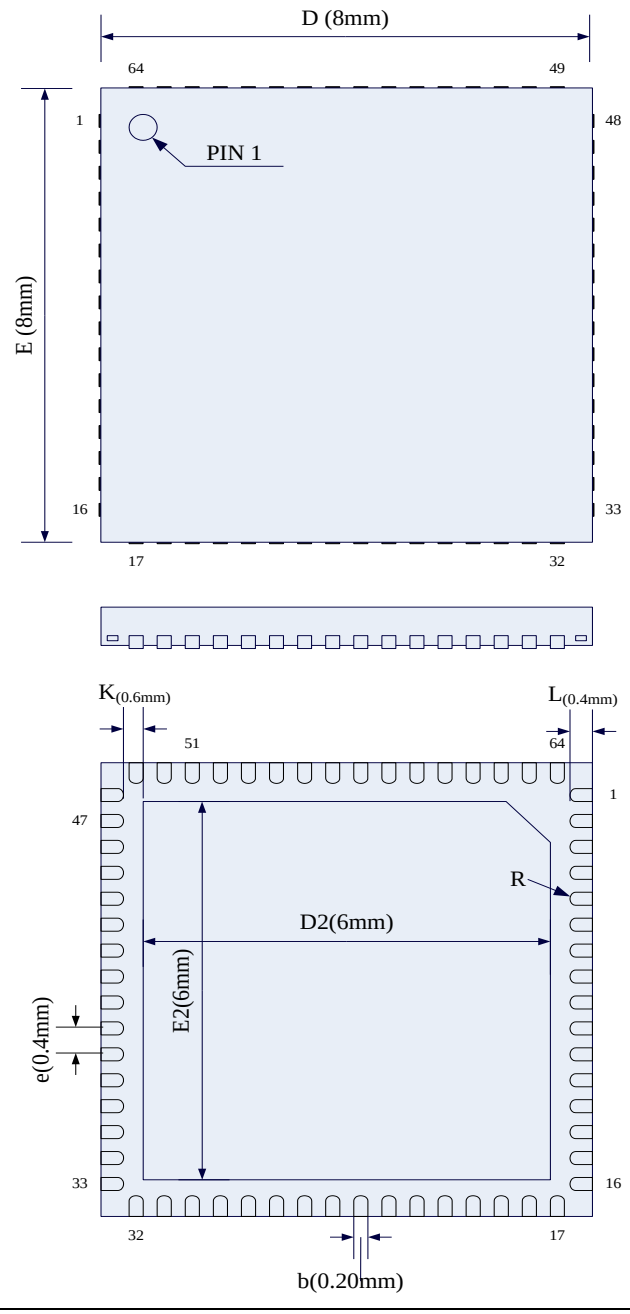
一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	-	-	1.60
A1	0.05	-	0.15
A2	1.35	1.40	1.45
A3	0.59	0.64	0.69
b	0.18	0.23	0.27
b1	0.17	0.20	0.23
c	0.13	-	0.18
c1	0.12	0.127	0.134
D	11.80	12.00	12.20
D1	9.90	10.00	10.10
E	11.80	12.00	12.20
E1	9.90	10.00	10.10
e	0.50BSC		
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25BSC		
R1	0.08	-	-
R2	0.08	-	0.20
S	0.20	-	-
?	0°	3.5°	7°
?1	0°	-	-
?2	11°	12°	13°
?3	11°	12°	13°



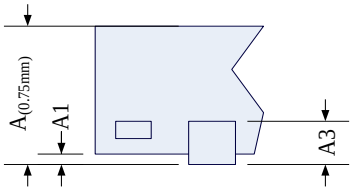


(A-A的截面图)

7.16　QFN64 封装尺寸图（8mm*8mm）



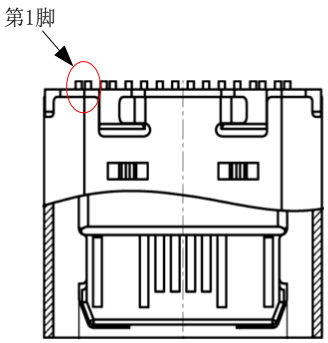
一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	0.70	0.75	0.80
A1	0.00	0.02	0.05
A2	0.50	0.55	0.60
A3		0.20REF	
b	0.15	0.20	0.25
D	7.90	8.00	8.10
E	7.90	8.00	8.10
D2	5.90	6.00	6.10
E2	5.90	6.00	6.10
e	0.30	0.40	0.50
L	0.30	0.40	0.50
K		0.60	
R	0.09	-	-



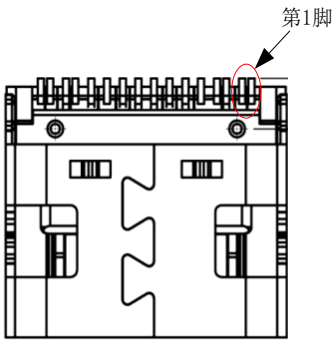
现有 QFN64 封装芯片的背面金属片（衬底），在芯片内部并未接地，在用户的 PCB 板上可以接地，也可以不接地，不会对芯片性能造成影响

特别说明：造 PCB 的封装库时，焊盘可以向外延伸 0.1～0.2mm，但绝对不能向内延伸，必须确保 K 不能小于 0.4mm，否则会造成管脚短路

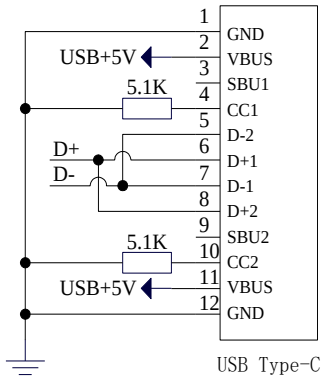
7.17 USB-TypeC 封装尺寸图，原理图



USB Type-C
实物图(正面)



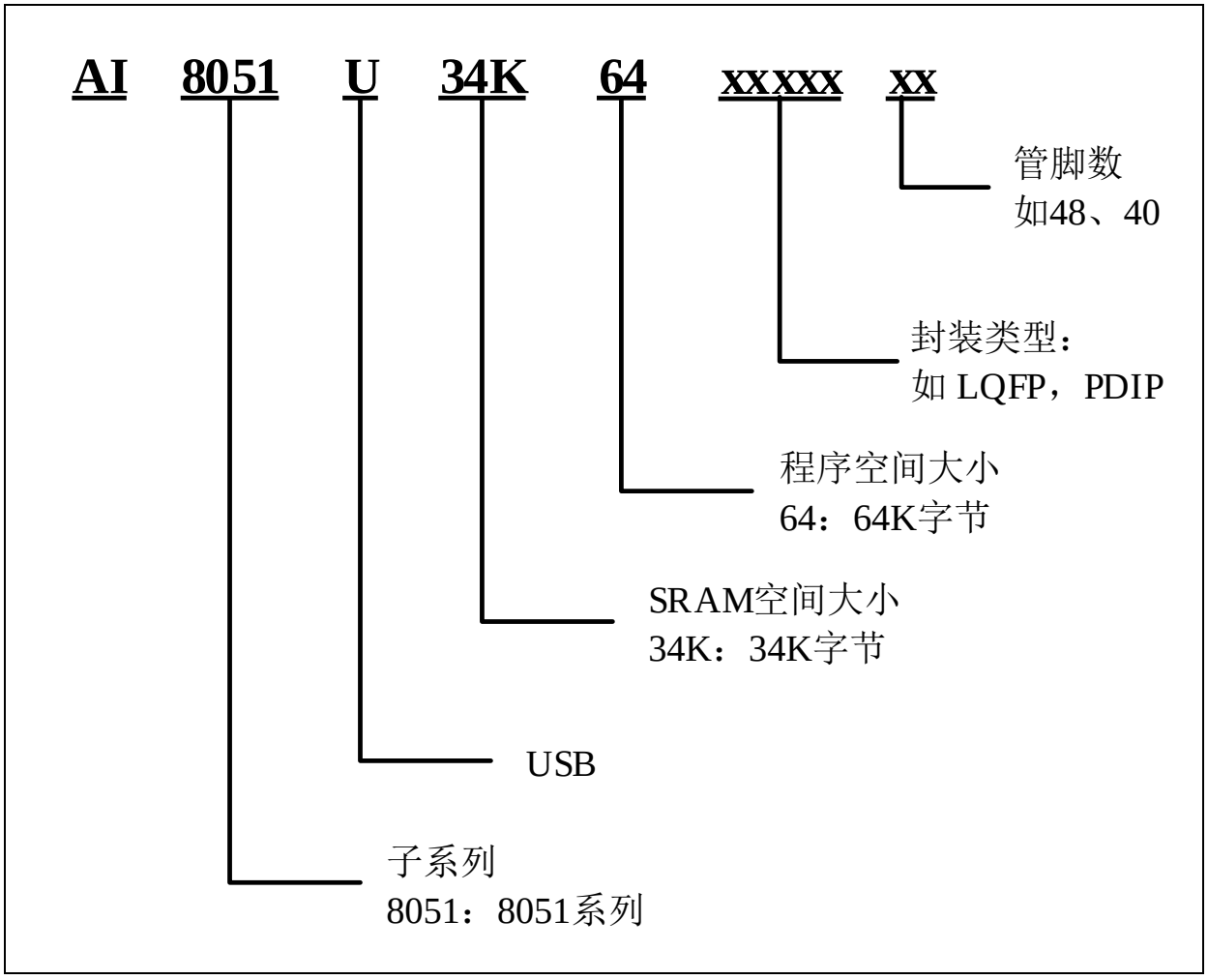
USB Type-C
实物图(反面)



USB Type-C
电路原理图

	A1	B12	A4	B9	A8	A5	B7	A6	A7	B6	B8	B5	B4	A9	B1	A12
	GND	GND	VBUS	VBUS	SBU1	CC1	D-2	D+1	D-1	D+2	SBU2	CC2	VBUS	VBUS	GND	GND
Type-C 16PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Type-C 12PIN	1		2		3	4	5	6	7	8	9	10	11		12	

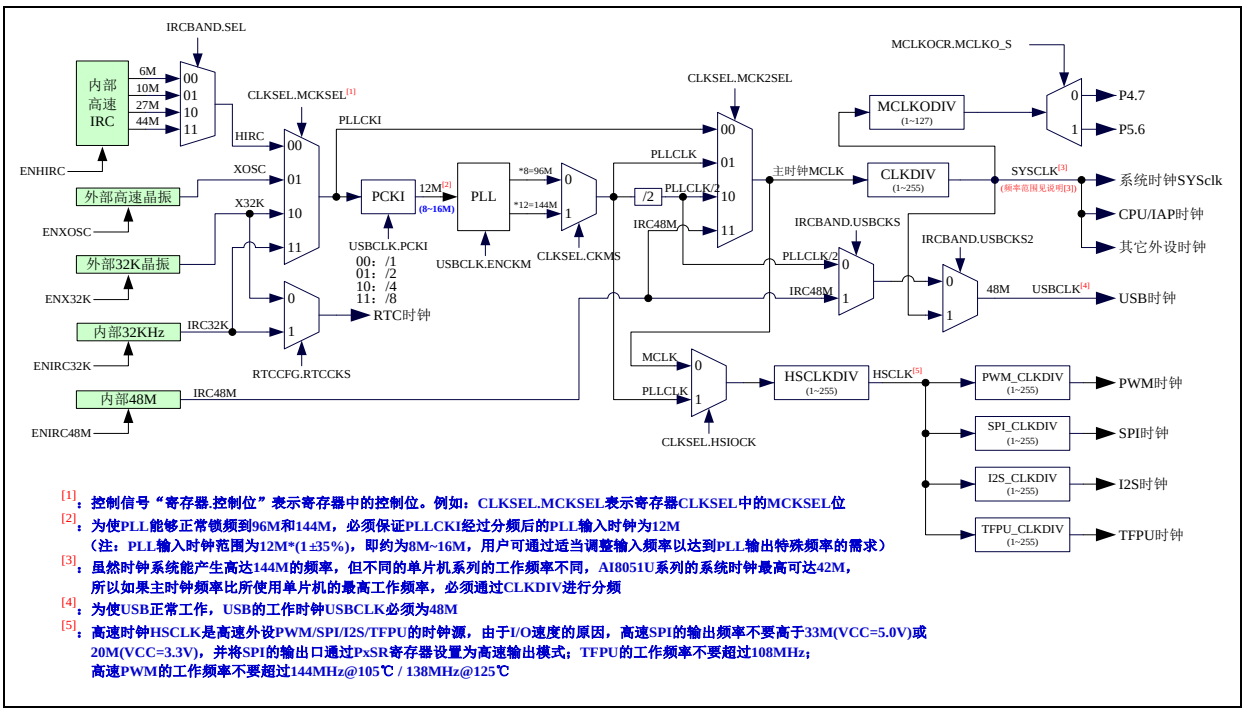
7.18 单片机命名规则


AI 8051 U 34K 64 xxxxx xx
管脚数
如48、40
封装类型：
如 LQFP，PDIP
程序空间大小
64：64K字节
SRAM空间大小
34K：34K字节
USB
子系列
8051：8051系列

8　时钟管理，芯片上电工作过程，外部晶振抗干扰

8.1　系统时钟控制

系统时钟控制器为单片机的 CPU 和所有外设系统提供时钟源，系统时钟有 4 个时钟源可供选择：内部高精度 IRC、内部 32KHz 的 IRC（误差较大）、外部晶振、内部 PLL 输出时钟。用户可通过程序分别使能和关闭各个时钟源，以及内部提供时钟分频以达到降低功耗的目的。



[1]：控制信号“寄存器.控制位”表示寄存器中的控制位。例如：CLKSEL.MCKSEL表示寄存器CLKSEL中的MCKSEL位

[2]：为使PLL能够正常锁频到96M和144M，必须保证PLLCKI经过分频后的PLL输入时钟为12M
（注：PLL输入时钟范围为12M*(1±35%)，即约为8M~16M，用户可通过适当调整输入频率以达到PLL输出特殊频率的需求）

[3]：虽然时钟系统能产生高达144M的频率，但不同的单片机系列的工作频率不同，AI8051U系列的系统时钟最高可达42M，
所以如果主时钟频率比所使用单片机的最高工作频率，必须通过CLKDIV进行分频

[4]：为使USB正常工作，USB的工作时钟USBCLK必须为48M

[5]：高速时钟HSCLK是高速外设PWM/SPI/I2S/TFPU的时钟源，由于I/O速度的原因，高速SPI的输出频率不要高于33M(VCC=5.0V)或
20M(VCC=3.3V)，并将SPI的输出口通过PxSR寄存器设置为高速输出模式；TFPU的工作频率不要超过108MHz；
高速PWM的工作频率不要超过144MHz@105℃ / 138MHz@125℃

系统及外设时钟选择参考表（详细设置参考范例程序）

主时钟选择 （MCLK）	内部 IRC	内部高速 IRC（HIRC）
		内部低速 IRC（IRC32K）
		内部 USB 专用 48M 高速 IRC（IRC48M）
	外部晶振	外部高速晶振（XOSC）
		外部低速晶振（X32K）
	PLL	内部 PLL 时钟（PLL）
		内部 PLL 时钟 2 分频（PLL/2）
高速外设时钟选择 （HSIOCK）	主时钟（MCLK）	
	内部 PLL 时钟（PLL）	
USB 时钟选择（48M）	内部 USB 专用 48M 高速 IRC（IRC48M）	
	内部 PLL 时钟 2 分频（PLL/2）	
	系统时钟（SYSCLK）	

注：系统时钟（SYSCLK）为主时钟（MCLK）通过 CLKDIV 分频所得的时钟
HSCLK 为高速外设时钟（HSIOCK）通过 HSCLKDIV 分频所得的时钟

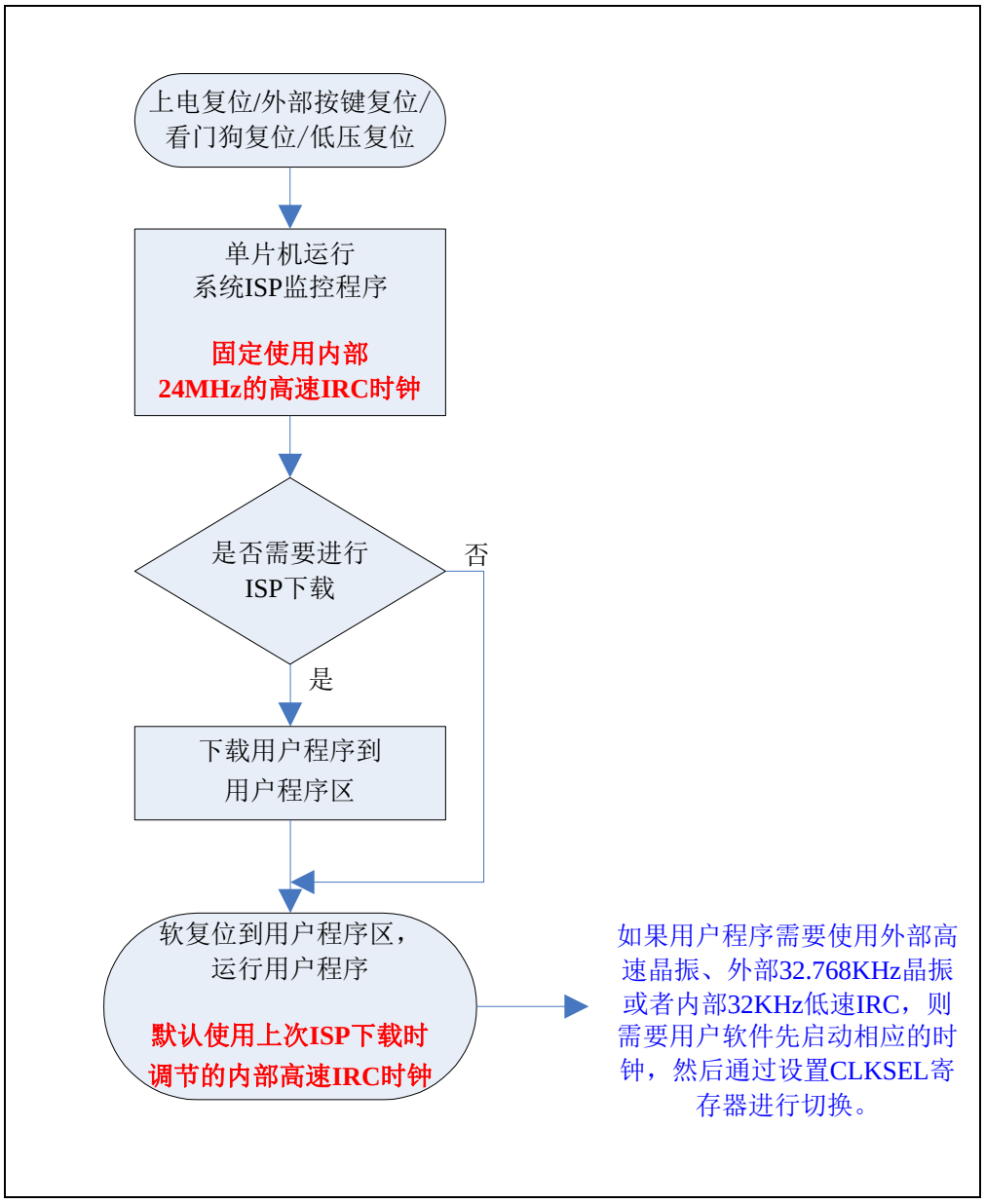
PLL 输入时钟选择参考

MCKSEL[1:0]	PLL 输入时钟源
00	内部高速 IRC（HIRC）
01	外部高速晶振（XOSC）
10	外部低速晶振（X32K）
11	内部低速 IRC（IRC32K）

8.2 芯片上电工作过程：

上电复位/复位脚复位/看门狗复位/低压检测复位时，芯片默认从 ISP 系统程序开始执行代码，此时固定使用内部 24MHz 的高速 IRC 时钟，当需要下载用户程序且下载完成后复位到用户程序区或者不需要下载直接复位到用户程序区时，默认会使用上次用户下载时所调节的高速 IRC 时钟，如果用户程序需要使用外部高速晶振、外部 32.768KHz 晶振或者内部 32KHz 低速 IRC，则需要用户软件先启动相应的时钟，然后通过设置 CLKSEL 寄存器进行切换。

启动流程如下：



8.3 时钟配置寄存器，外设，高频 PLL 时钟及分频，外部晶振抗干扰

相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
IRCBAND	IRC 频段选择	9DH	USBCKS	USBCKS2	-	-	-	-	SEL[1:0]		10xx,xxnn
IRTRIM	IRC 频率调整寄存器	9FH	IRTRIM[7:0]								nnnn,nnnn
USBCLK	USB 时钟控制寄存器	DCH	ENCKM	PCKI[1:0]		CRE	TST_USB	TST_PHY	PHYTST[1:0]		0010,0000

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CLKSEL	时钟选择寄存器	7EFE00H	CKMS	HSIOCK	-	-	MCK2SEL[1:0]		MCKSEL[1:0]		00xx,0000
CLKDIV	时钟分频寄存器	7EFE01H									nnnn,nnnn
HIRCCR	内部高速振荡器控制寄存器	7EFE02H	ENHIRC	-	-	-	-	-	-	HIRCST	1xxx,xxx0
XOSCCR	外部晶振控制寄存器	7EFE03H	ENXOSC	XITYPE	GAIN		XCFILTER[1:0]			XOSCST	000x,00x0
IRC32KCR	内部低速振荡器控制寄存器	7EFE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST	0xxx,xxx0
MCLKOCR	主时钟输出控制寄存器	7EFE05H	MCLKO_S	MCLKODIV[6:0]							0000,0000
IRCDB	内部高速振荡器稳定时间控制	7EFE06H									1000,0000
IRC48MCR	内部 48M 振荡器控制寄存器	7EFE07H	ENIRC48M	-	-	-	-	-	-	IRC48MST	0xxx,xxx0
X32KCR	外部 32K 晶振控制寄存器	7FFE08H	ENX32K	GAIN32K		-	-	-	-	X32KST	00xx,xxx0
HSCLKDIV	高速时钟分频寄存器	7EFE0BH									0000,0010
SPI_CLKDIV	SPI 时钟分频寄存器	7EFE90H									0000,0000
PWMA_CLKDIV	PWMA 时钟分频寄存器	7EFE91H									0000,0000
PWMB_CLKDIV	PWMB 时钟分频寄存器	7EFE92H									0000,0000
TFPU_CLKDIV	TFPU 时钟分频寄存器	7EFE93H									0000,0000
I2S_CLKDIV	I2S 时钟分频寄存器	7EFE94H									0000,0000

8.3.1 USB 时钟控制寄存器（USBCLK）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
USBCLK	DCH	ENCKM	PCKI[1:0]		CRE	TST_USB	TST_PHY	PHYTST[1:0]	

ENCKM：PLL 倍频控制

0：禁止 PLL 倍频

1：使能 PLL 倍频

PCKI[1:0]：PLL 时钟选择

PCKI[1:0]	PLL 时钟源
00	/1
01	/2
10	/4
11	/8

CRE：时钟追频控制位

0：禁止时钟追频

1：使能时钟追频

8.3.2 系统时钟选择寄存器（CLKSEL）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CLKSEL	7EFE00H	CKMS	HSIOCK	-	-	MCK2SEL[1:0]		MCKSEL[1:0]	

CKMS：内部 PLL 输出时钟选择

0：PLL 输出 96MHz

1：PLL 输出 144MHz

HSIOCK：高速 I/O 时钟源选择

0：主时钟 MCLK 为高速 I/O 时钟源

1：PLL 输出 96MHz/144MHz 的 PLLCLK 为高速 I/O 时钟源

MCK2SEL[1:0]：主时钟源选择

MCK2SEL[1:0]	主时钟源
00	MCKSEL 选择的时钟源
01	内部 PLL 输出
10	内部 PLL 输出/2
11	内部 48MHz 高速 IRC

MCKSEL[1:0]：主时钟源选择

MCKSEL[1:0]	主时钟源
00	内部高精度 IRC
01	外部高速晶振
10	外部 32KHz 晶振
11	内部 32KHz 低速 IRC

8.3.3 时钟分频寄存器（CLKDIV）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CLKDIV	7EFE01H								

CLKDIV：主时钟分频系数。系统时钟 SYSCLK 是对主时钟 MCLK 进行分频后的时钟信号。

CLKDIV	系统时钟频率
0	MCLK/1
1	MCLK/1
2	MCLK/2
3	MCLK/3
…	…
255	MCLK/255

8.3.4 内部高速高精度 IRC 控制寄存器（HIRCCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
HIRCCR	7EFE02H	ENHIRC	-	-	-	-	-	-	HIRCST

ENHIRC：内部高速高精度 IRC 使能位

0：关闭内部高精度 IRC

1：使能内部高精度 IRC

HIRCST：内部高速高精度 IRC 频率稳定标志位。（只读位）

当内部的 IRC 从停振状态开始使能后，必须经过一段时间，振荡器的频率才会稳定，当振荡器频率稳定后，时钟控制器会自动将 HIRCST 标志位置 1。所以当用户程序需要将时钟切换到使用内部 IRC 时，首先必须设置 ENHIRC=1 使能振荡器，然后一直查询振荡器稳定标志位 HIRCST，直到标志位变为 1 时，才可进行时钟源切换。

8.3.5 外部振荡器控制寄存器（XOSCCR），外部晶振抗干扰

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
XOSCCR	7EFE03H	ENXOSC	XITYPE	GAIN		XCFILTER[1:0]			XOSCST

ENXOSC：外部晶体振荡器使能位

0：关闭外部晶体振荡器

1：使能外部晶体振荡器

XITYPE：外部时钟源类型

0：外部时钟源是外部时钟信号（或有源晶振）。信号源只需连接单片机的 XTALI（P5.7），此时 P5.6 口为高阻输入模式，P5.6 可当作输入口使用。

1：外部时钟源是晶体振荡器。信号源连接单片机的 XTALI（P5.7）和 XTALO（P5.6）

XCFILTER[1:0]：外部晶体振荡器抗干扰控制寄存器

00：外部晶体振荡器频率在 48M～24M 之间时可选择此项

01：外部晶体振荡器频率在 24M～12M 之间时可选择此项

1x：外部晶体振荡器频率在 12M 及以下时可选择此项

GAIN：外部晶体振荡器振荡增益控制位

0：关闭振荡增益（低增益）

1：使能振荡增益（高增益）

XOSCST：外部晶体振荡器频率稳定标志位。（只读位）

当外部晶体振荡器从停振状态开始使能后，必须经过一段时间，振荡器的频率才会稳定，当振荡器频率稳定后，时钟控制器会自动将 XOSCST 标志位置 1。所以当用户程序需要将时钟切换到使用外部晶体振荡器时，首先必须设置 ENXOSC=1 使能振荡器，然后一直查询振荡器稳定标志位 XOSCST，直到标志位变为 1 时，才可进行时钟源切换。

8.3.6　内部低速 IRC 控制寄存器（IRC32KCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IRC32KCR	7EFE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST

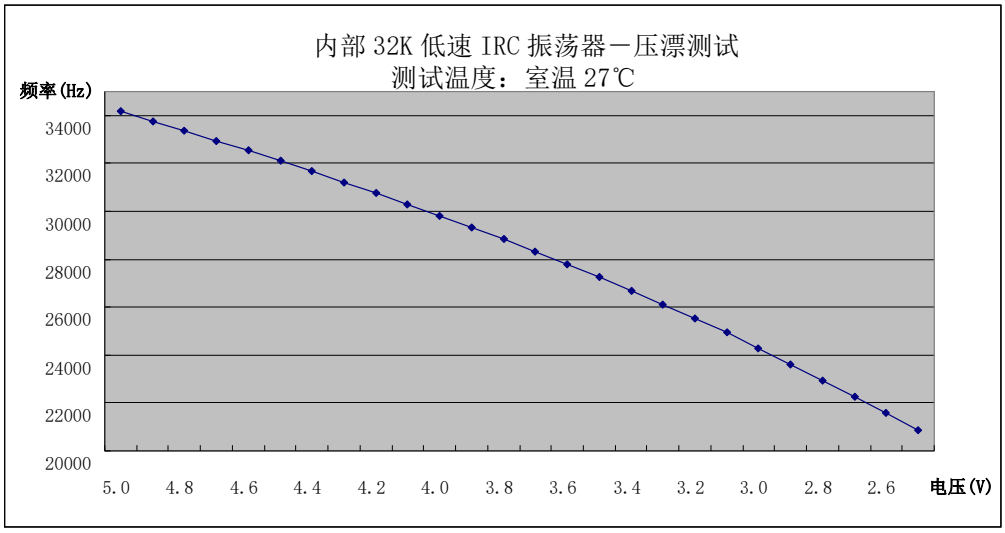
ENIRC32K：内部低速 IRC 使能位

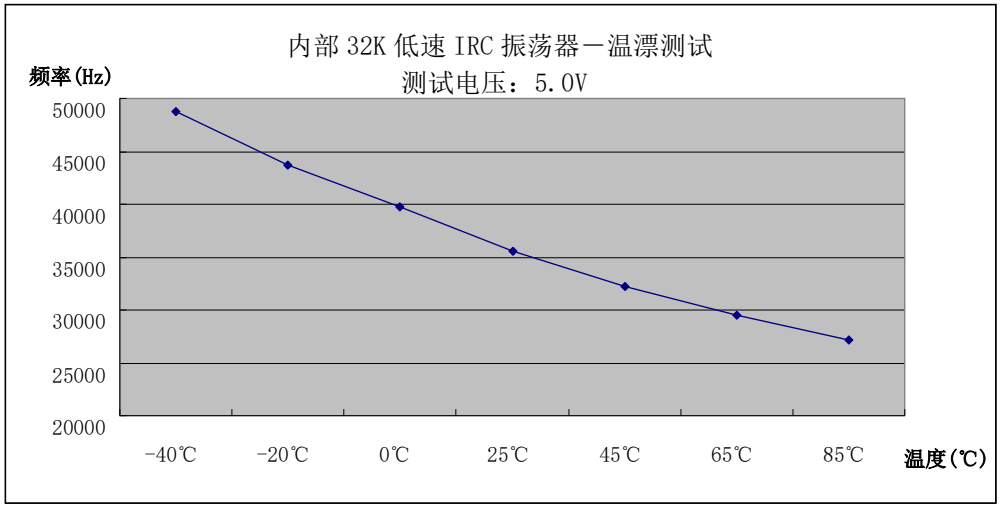
0：关闭内部低速 IRC

1：使能内部低速 IRC

IRC32KST：内部低速 IRC 频率稳定标志位。（只读位）

当内部低速 IRC 从停振状态开始使能后，必须经过一段时间，振荡器的频率才会稳定，当振荡器频率稳定后，时钟控制器会自动将 IRC32KST 标志位置 1。所以当用户程序需要将时钟切换到使用内部低速 IRC 时，首先必须设置 ENIRC32K=1 使能振荡器，然后一直查询振荡器稳定标志位 IRC32KST，直到标志位变为 1 时，才可进行时钟源切换。





8.3.7 主时钟输出控制寄存器（MCLKOCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
MCLKOCR	7EFE05H	MCLKO_S	MCLKODIV[6:0]						

MCLKODIV[6:0]：主时钟输出分频系数

MCLKODIV[6:0]	系统时钟分频输出频率
0000000	不输出时钟
0000001	SYSclk/1
0000010	SYSclk /2
0000011	SYSclk /3
...	...
1111110	SYSclk /126
1111111	SYSclk /127

MCLKO_S：系统时钟输出管脚选择

0：系统时钟分频输出到 P4.7 口

1：系统时钟分频输出到 P5.6 口

8.3.8 内部高速 IRC 时钟恢复振荡到稳定需要等待的时钟数控制寄存器（IRCDB）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IRCDB	FE06H								

IRCDB[7:0]：内部高速 IRC 时钟从主时钟停振/省电模式（掉电模式）恢复振荡到稳定需要等待的时钟数控制寄存器。**上电复位后初始值为 80H。**

IRCDB	系统时钟数
0	255 个时钟
1	0 个时钟
2	1 个时钟
3	2 个时钟
…	…
128	**127 个时钟（上电初始值）**
…	…
255	254 个时钟

【寄存器说明】：

1、芯片使用内部高速 IRC 时钟作为系统时钟时，当芯片从主时钟停振/省电模式（掉电模式）被唤醒后，系统会等待 IRCDB 寄存器设置的等待时钟数，再将时钟提供给 CPU 和系统，CPU 获得时钟后开始继续运行用户程序。建议在主时钟停振语句的下一句后面增加 7~9 个 NOP。

2、需要设置等待多少个时钟再给 CPU 供应时钟，让 CPU 开始跑程序，根据用户实际使用的工作频率来确定，目前测试发现运行在 33.1776MHz 附近或以下时，必须将 IRCDB 设置为 0x10 时，这样就很稳定。

3、如果用户的工作频率高于 33.1776MHz 时，建议用户程序在进入主时钟停振/省电模式前将内部高速 IRC 频率调低，再进入主时钟停振/省电模式。等芯片再次被唤醒后再将内部高速 IRC 调整到用户需要的工作频率。内部高速 IRC 时钟，出厂时有多种校准值，用户程序可以任意设置芯片出厂时预置的几种校准时钟频率。实际测试发现芯片运行在 40MHz 以下时，将 IRCDB 设置为 0x10 也是稳定的。

4、如果程序需要进入省电模式，程序初始化时必须加上下面这条语句：

```
IRCDB = 0x10;
//内部高速 IRC 时钟，停振后被唤醒后起振，等待多少个时钟数后给 MCU 供应时钟
```

8.3.9 内部 48MHz 高速 IRC 控制寄存器（IRC48MCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IRC48MCR	7EFE07H	ENIRC48M	-	-	-	-	-	-	IRC48MST

ENIRC48M：内部 48M 高速 IRC 使能位

0：关闭内部 48M 高速 IRC

1：使能内部 48M 高速 IRC

IRC48MST：内部 48M 高速 IRC 频率稳定标志位。（只读位）

当内部 48M 高速 IRC 从停振状态开始使能后，必须经过一段时间，振荡器的频率才会稳定，当振荡器频率稳定后，时钟控制器会自动将 IRC48MST 标志位置 1。所以当用户程序需要将时钟切换到使用内部 48M 高速 IRC 时，首先必须设置 ENIRC48M=1 使能振荡器，然后一直查询振荡器稳定标志位 IRC48MST，直到标志位变为 1 时，才可进行时钟源切换。

8.3.10 外部 32K 振荡器控制寄存器（X32KCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
X32KCR	7EFE08H	ENX32K	GAIN32K	-	-	-	-	-	X32KST

ENX32K：外部 32K 晶体振荡器使能位

0：关闭外部 32K 晶体振荡器

1：使能外部 32K 晶体振荡器

GAIN32K：外部 32K 晶体振荡器振荡增益控制位

0：关闭 32K 振荡增益（低增益）

1：使能 32K 振荡增益（高增益）

X32KST：外部 32K 晶体振荡器频率稳定标志位。（只读位）

8.3.11 高速时钟分频寄存器（HSCLKDIV）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
HSCLKDIV	7EFE0BH								

HSCLKDIV：高速 I/O 时钟分频系数。

CLKDIV	系统时钟频率
0	高速 I/O 时钟源/1
1	高速 I/O 时钟源/1
2	高速 I/O 时钟源/2
3	高速 I/O 时钟源/3
…	…
255	高速 I/O 时钟源/255

8.3.12 SPI 时钟分频寄存器（SPI_CLKDIV）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPI_CLKDIV	7EFE90H								

SPI_CLKDIV：高速 SPI 时钟分频系数。

SPI_CLKDIV	SPI 工作时钟频率
0	HSCLK /1
1	HSCLK /1
2	HSCLK /2
3	HSCLK /3
…	…
255	HSCLK /255

8.3.13 PWMA 时钟分频寄存器（PWMA_CLKDIV）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CLKDIV	7EFE91H								

PWMA_CLKDIV：高速 PWMA 时钟分频系数。（注：PWMA 最高工作频率必须在 144MHz 以下）

PWMA_CLKDIV	PWMA 工作时钟频率
0	HSCLK /1
1	HSCLK /1
2	HSCLK /2
3	HSCLK /3
…	…
255	HSCLK /255

8.3.14 PWMB 时钟分频寄存器（PWMB_CLKDIV）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMB_CLKDIV	7EFE92H								

PWMB_CLKDIV：高速 PWMB 时钟分频系数。（注：PWMB 最高工作频率必须在 144MHz 以下）

PWMB_CLKDIV	PWMB 工作时钟频率
0	HSCLK /1
1	HSCLK /1
2	HSCLK /2
3	HSCLK /3
…	…
255	HSCLK /255

8.3.15 TFPU 时钟分频寄存器（TFPU_CLKDIV）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TFPU_CLKDIV	7EFE93H								

TFPU_CLKDIV：TFPU 时钟分频系数。（注：TFPU 最高工作频率必须在 108MHz 以下）

TFPU_CLKDIV	TFPU 工作时钟频率
0	HSCLK /1
1	HSCLK /1
2	HSCLK /2
3	HSCLK /3
…	…
255	HSCLK /255

8.3.16 I2S 时钟分频寄存器（I2S_CLKDIV）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2S_CLKDIV	7EFE94H								

I2S_CLKDIV：I2S 时钟分频系数。

I2S_CLKDIV	I2S 工作时钟频率
0	HSCLK /1
1	HSCLK /1
2	HSCLK /2
3	HSCLK /3
…	…
255	HSCLK /255

8.3.17 I2C 总线速度控制

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CCFG	7EFE80H	ENI2C	MSSL	**MSSPEED[5:0]**					
I2CPSCR	7EFE889H	**MSSPEED[13:6]**							

MSSPEED[13:0]：I²C 总线速度（等待时钟数）控制，**I2C 总线速度＝SYSCLK / 2 / (MSSPEED * 2 + 4)**

（I2C 最快速度为 SYSCLK/8）

MSSPEED[13:0]	对应的时钟数
0	4
1	6
2	8
…	…
x	2x+4
…	…
62	128
63	130
…	…
16383	32770

8.4 Ai8051U 系列内部高速 IRC 时钟频率调整

Ai8051U 系列单片机内部均集成有一颗高精度内部 IRC 振荡器。在用户使用 ISP 下载软件进行下载时，ISP 下载软件会根据用户所选择/设置的频率自动进行调整，一般频率值的精度可调整到±0.3%以下，调整后的频率在全温度范围内（－40℃~85℃）的温漂可达-1.35%~1.30%。如需工作在-40℃~125℃，则内部高精度 IRC 时钟的温漂是±3%，这时如串口通讯 1 个字节分两次发送，温漂可以认为是±1.5%。当然也可以使用外部高频晶振，或者使用外部 32768Hz-RTC 晶振自动校准内部高速 IRC 时钟。

Ai8051U 系列内部 IRC 有 4 个频段，频段的中心频率分别为 6MHz、10MHz、27MHz 和 44MHz，每个频段的调节范围约为±27%（注意：不同的芯片以及不同的生成批次可能会有约 5%左右的制造误差）。

内部高速 IRC 的 4 个频段的中心频率，供参考：

单位（MHz）		样品 1	样品 2	样品 3	样品 4	平均值
6M 频段	最小频率	4.39	4.42	4.43	4.61	4.46
	中间频率	6.22	6.26	6.20	6.53	6.30
	最大频率	8.03	8.05	8.08	8.36	8.13
10M 频段	最小频率	7.20	7.29	7.34	7.74	7.39
	中间频率	10.11	10.26	10.33	10.91	10.40
	最大频率	12.99	13.11	13.26	13.90	13.32
27M 频段	最小频率	20.06	20.11	20.46	21.11	20.43
	中间频率	27.82	27.67	28.24	29.14	28.22
	最大频率	35.14	34.97	35.75	36.83	35.67
44M 频段	最小频率	27.79	28.20	28.72	29.70	28.60
	中间频率	41.45	41.74	42.55	44.04	42.45
	最大频率	54.04	54.41	55.50	57.47	55.35

注意：对于一般用户，内部 IRC 频率的调整可以不用关心，因为频率调整工作在进行 ISP 下载时已经自动完成了。所以若用户不需要自行调整频率，那么下面相关的 4 个寄存器也不能随意修改，否则可能会导致工作频率变化。

内部 IRC 频率调整主要使用下面的 4 个寄存器进行调整

8.4.1 IRC 频段选择寄存器（IRCBAND）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IRCBAND	9DH	USBCKS	USBCKS2	-	-	-	-	SEL[1:0]	

USBCKS/USBCKS2：USB 时钟选择寄存器

USBCKS	USBCKS2	USB 时钟
0	0	PLLCLK/2
1	0	IRC48M
x	1	系统时钟 SYSCLK

SEL[1:0]：频段选择

00：选择 6MHz 频段

01：选择 10MHz 频段

10：选择 27MHz 频段

11：选择 44MHz 频段

Ai8051U 系列内部 IRC 有四个频段，频段的中心频率分别为 6MHz、10MHz、27MHz 和 44MHz，每个频段的调节范围约为±27%。

Ai8051U 系列芯片出厂时校准的 10 个频率如下：

频率（MHz）	22.1184	24	27	30	33.1776	35	36.864	40	44.2368	48
所属频段	27M	27M	27M	27M	27M	44M	44M	44M	44M	44M

8.4.2 内部 IRC 频率调整寄存器（IRTRIM）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IRTRIM	9FH	IRTRIM[7:0]							

IRTRIM[7:0]：内部高精度 IRC 频率调整寄存器

IRTRIM 可对 IRC 频率进行 256 个等级的调整，每个等级所调整的频率值在整体上呈线性分布，局部会有波动。宏观上，每一级所调整的频率约为 0.24%，即 IRTRIM 为（n+1）时的频率比 IRTRIM 为（n）时的频率约快 0.24%。但由于 IRC 频率调整并非每一级都是 0.24%（每一级所调整频率的最大值约为 0.55%，最小值约为 0.02%，整体平均值约为 0.24%），所以会造成局部波动。

8.4.3 时钟分频寄存器（CLKDIV）

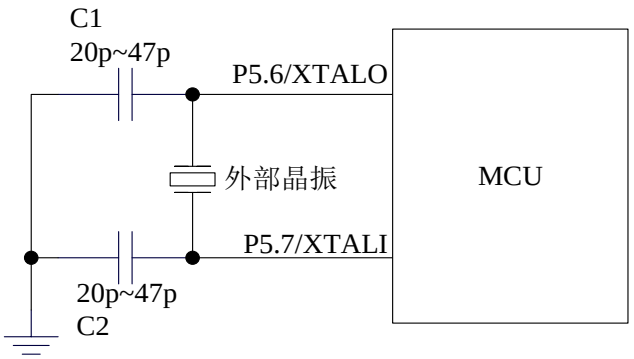
符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CLKDIV	7EFE01H								

CLKDIV：主时钟分频系数。系统时钟 SYSCLK 是对主时钟 MCLK 进行分频后的时钟信号。

CLKDIV	系统时钟频率
0	MCLK/1
1	MCLK/1
2	MCLK/2
3	MCLK/3
…	…
255	MCLK/255

8.5　外部晶振及外部时钟电路

8.5.1 外部晶振输入电路



注意：

1、一般不需外部晶振，我司内部高速HIRC时钟完全满足串口通讯需求。
如接外部晶振，C1、C2两个电容一定不能省，否则晶振可能不起振

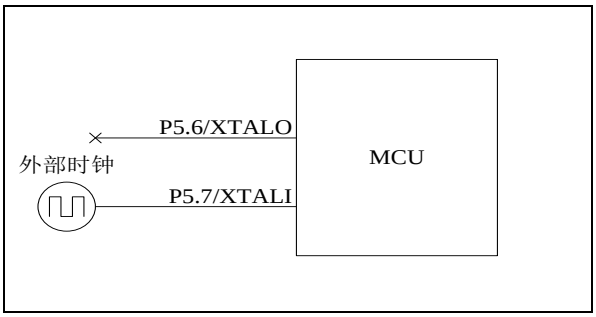
2、RTC时钟必须使用外部32768晶振，外挂电容要<=20pF，例如20p/15p/10p

3、若外部连接高速晶振（例如：24MHz），外挂电容47pF为宜

4、外部晶振会增加额外的系统功耗，32768晶振增加1.5uA的功耗，
高速晶振（例如：24MHz）会增加2mA以上的功耗

5、XTALI / XTAL1 串个 200欧的电阻，有利于外部晶振起振
XTALO / XTAL2 串个 1K的电阻，减少外部晶振振荡后对外的电磁辐射

8.5.2 外部时钟输入电路（P5.6 为高阻输入模式，可当输入口使用）



注：当使用内部时钟时，P5.6/P5.7 都可以当普通 I/O 使用。当 P5.7 口外接有源时钟或外接其他时钟源时，P5.6 口为高阻输入模式，可当输入口使用，此时 P5.6 的端口模式不可改变。有 RTC 功能的 MCU，外部 32768 时钟可从 P5.7 口输入。

8.6　范例程序

8.6.1　选择内部高速 IRC（HIRC）作为系统时钟源

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件

void main()
{
    EAXFR = 1;                                //允许访问扩展的特殊寄存器，XFR
                                              //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                            //允许访问扩展的特殊寄存器，XFR
                                              //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                 //设置取程序代码等待时间，
                                              //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                //设置访问片内的 xdata 速度，
                                              //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

                                              //注：芯片上电后系统会默启动内部高速 HIRC，
                                              //并选择为系统时钟，所以一般情况下不需要作
                                              //如下设置
    HIRCCR = 0x80;                            //启动内部高速 IRC
    while (!(HIRCCR & 1));                    //等待时钟稳定
    CLKSEL = 0x00;                            //选择内部高速 HIRC

    while (1);
}
```

8.6.2　在用户程序区用户如何动态选择不同的系统时钟频率

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件
#include "intrins.h"                          //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。
#define   T22M_ADDR     CHIPID11              //22.1184MHz
#define   T24M_ADDR     CHIPID12              //24MHz
#define   T27M_ADDR     CHIPID13              //27MHz
```

```
#define   T30M_ADDR     CHIPID14        //30MHz
#define   T33M_ADDR     CHIPID15        //33.1776MHz
#define   T35M_ADDR     CHIPID16        //35MHz
#define   T36M_ADDR     CHIPID17        //36.864MHz
#define   T40M_ADDR     CHIPID18        //40MHz
#define   T44M_ADDR     CHIPID19        //44.2368MHz
#define   T48M_ADDR     CHIPID20        //48MHz
#define   VRT6M_ADDR    CHIPID21        //VRTRIM_6M
#define   VRT10M_ADDR   CHIPID22        //VRTRIM_10M
#define   VRT27M_ADDR   CHIPID23        //VRTRIM_27M
#define   VRT44M_ADDR   CHIPID24        //VRTRIM_44M

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                           //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

//  //选择 22.1184MHz
//  CLKDIV = 0x04;
//  IRTRIM = T22M_ADDR;
//  VRTRIM = VRT27M_ADDR;
//  IRCBAND &= ~0x03;
//  IRCBAND |= 0x02;
//  CLKDIV = 0x00;

    //选择 24MHz
    CLKDIV = 0x04;
    IRTRIM = T24M_ADDR;
    VRTRIM = VRT27M_ADDR;
    IRCBAND &= ~0x03;
    IRCBAND |= 0x02;
    CLKDIV = 0x00;

//  //选择 27MHz
//  CLKDIV = 0x04;
//  IRTRIM = T27M_ADDR;
//  VRTRIM = VRT27M_ADDR;
//  IRCBAND &= ~0x03;
//  IRCBAND |= 0x02;
//  CLKDIV = 0x00;
```

```
//      //选择 30MHz
//      CLKDIV = 0x04;
//      IRTRIM = T30M_ADDR;
//      VRTRIM = VRT27M_ADDR;
//      IRCBAND &= ~0x03;
//      IRCBAND |= 0x02;
//      CLKDIV = 0x00;

//      //选择 33.1776MHz
//      CLKDIV = 0x04;
//      IRTRIM = T33M_ADDR;
//      VRTRIM = VRT27M_ADDR;
//      IRCBAND &= ~0x03;
//      IRCBAND |= 0x02;
//      CLKDIV = 0x00;

//      //选择 35MHz
//      CLKDIV = 0x04;
//      IRTRIM = T35M_ADDR;
//      VRTRIM = VRT44M_ADDR;
//      IRCBAND |= 0x03;
//      CLKDIV = 0x00;

//      //选择 44.2368MHz
//      CLKDIV = 0x04;
//      IRTRIM = T44M_ADDR;
//      VRTRIM = VRT44M_ADDR;
//      IRCBAND |= 0x03;
//      CLKDIV = 0x00;

//      //选择 48MHz
//      CLKDIV = 0x04;
//      IRTRIM = T48M_ADDR;
//      VRTRIM = VRT44M_ADDR;
//      IRCBAND |= 0x03;
//      CLKDIV = 0x00;

        while (1);
}
```

8.6.3　选择内部 IRC（IRC32K）作为系统时钟源

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件

void main()
{
    EAXFR = 1;                                //允许访问扩展的特殊寄存器，XFR
                                              //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                            //允许访问扩展的特殊寄存器，XFR
                                              //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                 //设置取程序代码等待时间，
                                              //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                //设置访问片内的 xdata 速度，
                                              //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IRC32KCR = 0x80;                          //启动内部低速 IRC
    while (!(IRC32KCR & 1));                  //等待时钟稳定
    CLKDIV = 0x00;                            //时钟不分频
    CLKSEL = 0x03;                            //选择内部低速 IRC

    while (1);
}
```

8.6.4　选择内部 48M 的 IRC（IRC48M）作为系统时钟源

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件

void main()
{
    EAXFR = 1;                                //允许访问扩展的特殊寄存器，XFR
                                              //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                            //允许访问扩展的特殊寄存器，XFR
                                              //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                 //设置取程序代码等待时间，
                                              //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                //设置访问片内的 xdata 速度，
```

```
                                                    //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IRC48MCR = 0x80;                                //启动内部 48M IRC
    while (!(IRC48MCR & 1));                        //等待时钟稳定
    CLKDIV = 0x02;                                  //时钟 2 分频
    CLKSEL = 0x0c;                                  //选择 IRC48M

    while (1);
}
```

8.6.5　选择外部高速晶振（XOSC）作为系统时钟源

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                //头文件见下载软件

void main()
{
    EAXFR = 1;                                      //允许访问扩展的特殊寄存器，XFR
                                                    //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                  //允许访问扩展的特殊寄存器，XFR
                                                    //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                       //设置取程序代码等待时间，
                                                    //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                      //设置访问片内的 xdata 速度，
                                                    //赋值为 0 表示用最快速度访问，不增加额外的等待时间


    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    XOSCCR = 0xc0;                                  //启动外部晶振
    while (!(XOSCCR & 1));                          //等待时钟稳定
```

```
    CLKDIV = 0x00;                          //时钟不分频
    CLKSEL = 0x01;                          //选择外部晶振

    while (1);
}
```

8.6.6　选择外部低速晶振（X32K）作为系统时钟源

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                       //头文件见下载软件

void main()
{
    EAXFR = 1;                              //允许访问扩展的特殊寄存器，XFR
                                            //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                          //允许访问扩展的特殊寄存器，XFR
                                            //(8 位模式请使用这句，注释上一句)
    WTST = 0;                               //设置取程序代码等待时间，
                                            //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                              //设置访问片内的 xdata 速度，
                                            //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    X32KCR = 0xc0;                          //启动外部 32K 晶振
    while (!(X32KCR & 1));                  //等待时钟稳定
    CLKDIV = 0x00;                          //时钟不分频
    CLKSEL = 0x02;                          //选择外部 32K 晶振

    while (1);
}
```

8.6.7　选择内部 PLL 作为系统时钟源

```
//测试工作频率为 12MHz

#include “Ai8051U.H”                       //头文件见下载软件

void delay()
```

```
{
    int i;

    for (i=0; i<100; i++);
}

void main()
{
    EAXFR = 1;                  //允许访问扩展的特殊寄存器，XFR
                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;              //允许访问扩展的特殊寄存器，XFR
                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                   //设置取程序代码等待时间，
                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                  //设置访问片内的 xdata 速度，
                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CLKSEL &= ~0x80;            //选择 PLL 的 96M 作为 PLL 的输出时钟
//  CLKSEL |= 0x80;             //选择 PLL 的 144M 作为 PLL 的输出时钟

    USBCLK &= ~0x60;
    USBCLK |= 0x00;             //PLL 输入时钟为 12M 则选择 1 分频
//  USBCLK |= 0x20;             //PLL 输入时钟为 24M 则选择 2 分频
//  USBCLK |= 0x40;             //PLL 输入时钟为 48M 则选择 4 分频
//  USBCLK |= 0x60;             //PLL 输入时钟为 96M 则选择 8 分频

    USBCLK |= 0x80;             //启动 PLL

    delay();                    //等待 PLL 锁频，建议 50us 以上

    CLKDIV = 0x04;              //时钟 4 分频，主时钟选择高速频率前,
                                //必须先设置分频系数,否则程序会当掉
    CLKSEL &= 0xf0;             //选择 PLL 时钟源
    CLKSEL |= 0x04;

    while (1);
}
```

8.6.8　选择主时钟（MCLK）作为高速外设时钟源

```
//测试工作频率为 11.0592MHz
```

```
#include “Ai8051U.H”                          //头文件见下载软件

void main()
{
    EAXFR = 1;                               //允许访问扩展的特殊寄存器，XFR
                                             //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                           //允许访问扩展的特殊寄存器，XFR
                                             //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                //设置取程序代码等待时间，
                                             //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                               //设置访问片内的 xdata 速度，
                                             //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    HSCLKDIV = 0x00;
    CLKSEL &= ~0x40;                         //选择主时钟（MCLK）作为高速外设时钟源

    while (1);
}
```

8.6.9　选择内部 PLL 时钟作为高速外设时钟源

```
//测试工作频率为 12MHz

#include “Ai8051U.H”                          //头文件见下载软件

void main()
{
    EAXFR = 1;                               //允许访问扩展的特殊寄存器，XFR
                                             //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                           //允许访问扩展的特殊寄存器，XFR
                                             //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                //设置取程序代码等待时间，
                                             //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                               //设置访问片内的 xdata 速度，
                                             //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
```

```
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CLKSEL &= ~0x80;                    //选择 PLL 的 96M 作为 PLL 的输出时钟
//  CLKSEL |= 0x80;                     //选择 PLL 的 144M 作为 PLL 的输出时钟

    USBCLK &= ~0x60;
    USBCLK |= 0x00;                     //PLL 输入时钟为 12M 则选择 1 分频
//  USBCLK |= 0x20;                     //PLL 输入时钟为 24M 则选择 2 分频
//  USBCLK |= 0x40;                     //PLL 输入时钟为 48M 则选择 4 分频
//  USBCLK |= 0x60;                     //PLL 输入时钟为 96M 则选择 8 分频

    USBCLK |= 0x80;                     //启动 PLL

    delay();                            //等待 PLL 锁频，建议 50us 以上

    HSCLKDIV = 0x00;
    CLKSEL |= 0x40;                     //选择 PLL 时钟作为高速外设时钟源

    while (1);
}
```

8.6.10　选择系统时钟（SYSCLK）作为 USB 时钟源

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                   //头文件见下载软件

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                           //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;
```

```
    IRCBAND |= 0x40;                        //选择系统时钟（SYSCLK）作为 USB 时钟源

    while (1);
}
```

8.6.11　选择内部 PLL 时钟作为 USB 时钟源

```
//测试工作频率为 12MHz

#include “Ai8051U.H”                       //头文件见下载软件

void main()
{
    EAXFR = 1;                              //允许访问扩展的特殊寄存器，XFR
                                            //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                          //允许访问扩展的特殊寄存器，XFR
                                            //(8 位模式请使用这句，注释上一句)
    WTST = 0;                               //设置取程序代码等待时间，
                                            //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                              //设置访问片内的 xdata 速度，
                                            //赋值为 0 表示用最快速度访问，不增加额外的等待时间
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CLKSEL &= ~0x80;                        //选择 PLL 的 96M 作为 PLL 的输出时钟
//  CLKSEL |= 0x80;                         //选择 PLL 的 144M 作为 PLL 的输出时钟

    USBCLK &= ~0x60;
    USBCLK |= 0x00;                         //PLL 输入时钟为 12M 则选择 1 分频
//  USBCLK |= 0x20;                         //PLL 输入时钟为 24M 则选择 2 分频
//  USBCLK |= 0x40;                         //PLL 输入时钟为 48M 则选择 4 分频
//  USBCLK |= 0x60;                         //PLL 输入时钟为 96M 则选择 8 分频

    USBCLK |= 0x80;                         //启动 PLL

    delay();                                //等待 PLL 锁频，建议 50us 以上

    IRCBAND &=~ 0xc0;                       //选择 PLL 时钟（96M/2=48M）作为 USB 时钟源

    while (1);
}
```

8.6.12　选择内部 USB 专用 48M 的 IRC 作为 USB 时钟源

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件

void main()
{
    EAXFR = 1;                                //允许访问扩展的特殊寄存器，XFR
                                              //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                            //允许访问扩展的特殊寄存器，XFR
                                              //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                 //设置取程序代码等待时间，
                                              //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                //设置访问片内的 xdata 速度，
                                              //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IRC48MCR = 0x80;                          //启动内部 48M IRC
    while (!(IRC48MCR & 1));                  //等待时钟稳定

    IRCBAND |= 0x80;                          //选择 IRC48M 作为 USB 时钟源
    IRCBAND &= ~0x40;

    while (1);
}
```

8.6.13　主时钟分频输出

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件

void main()
{
    EAXFR = 1;                                //允许访问扩展的特殊寄存器，XFR
                                              //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                            //允许访问扩展的特殊寄存器，XFR
                                              //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                 //设置取程序代码等待时间，
                                              //赋值为 0 表示不等待，程序以最快速度运行
```

```
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

//  MCLKOCR = 0x01;                             //主时钟输出到 P4.7 口
//  MCLKOCR = 0x02;                             //主时钟 2 分频输出到 P4.7 口
    MCLKOCR = 0x04;                             //主时钟 4 分频输出到 P4.7 口
//  MCLKOCR = 0x84;                             //主时钟 4 分频输出到 P5.6 口

    while (1);
}
```

9 校准内部 IRC 时钟，利用外部 32768Hz 晶振，降低电磁辐射

== 自动频率校准，校准内部高速 IRC 时钟，自动追频（CRE）

产品线	自动追频
Ai8051U 系列	●

Ai8051U 单片机系列内建一个频率自动校准模块（CRE），CRE 模块是使用外部的 32.768KHz 晶振对内部高速 IRC（HIRC）的 IRTRIM 寄存器进行自动调整，以达到自动频率校准的功能。

CPU 使用内部高速 IRC 的时钟，时钟对外电磁辐射低；如 CPU 使用外部高速晶振，外部高速晶振振荡后对外电磁辐射大；如对内部高速 IRC 时钟的温漂要求比较严苛，可以用外部 32768Hz-RTC 晶振来自动校准内部高速 IRC 时钟。

需要使用自动校准时，只需要根据给定的公式设置好目标频率的计数值和误差范围，然启动 CRE 模块，硬件便会进行自动频率校准，当 HIRC 的频率达到用户所设置误差范围内时，校准完成标志会被置位。

9.1 相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CRECR	CRE 控制寄存器	7EFDA8H	ENCRE	MONO	UPT[1:0]		CREHF	CREINC	CREDEC	CRERDY	0000,0000
CRECNTH	CRE 校准目标寄存器	7EFDA9H	CNT[15:8]								0000,0000
CRECNTL	CRE 校准目标寄存器	7EFDAAH	CNT[7:0]								0000,0000
CRERES	CRE 分辨率控制寄存器	7EFDABH	RES[7:0]								0000,0000

9.1.1 CRE 控制寄存器（CRECR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CRECR	7EFDA8H	ENCRE	MONO	UPT[1:0]		CREHF	CREINC	CREDEC	CRERDY

ENCRE：CRE模块控制位

0：关闭 CRE 模块。

1：使能 CRE 模块。

MONO：自动校准步幅控制

0：单步模式。每个校准周期，硬件自动将 IRTRIM 递增或递减 1。

1：双步模式。每个校准周期，硬件自动将 IRTRIM 递增或递减 2。

单步模式比双步模式校准后的 IRC 精度更高，但自动校准的时间比双步模式长。

UPT[1:0]：CRE 校准周期选择

UPT[1:0]	校准周期
00	1ms
01	4ms
10	32ms
11	64ms

CREHF：高频模式选择

0：低频模式（目标频率小于或等于 50MHz）。

1：高频模式（目标频率大于 50MHz）。

CREINC：CRE校准正处于上调状态。只读位。

CREDEC：CRE校准正处于下调状态。只读位。

CRERDY：CRE校准完成状态。只读位。

0：CRE 校准功能未启动或者未校准完成。

1：CRE 校准已完成。

9.1.2 CRE 校准计数值寄存器（CRECNT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CRECNTH	7EFDA9H	CRECNT[15:8]							
CRECNTL	7EFDAAH	CRECNT[7:0]							

CRECNT[15:0]：16位校准计数值。

目标校准值计算公式：

低频模式（CREHF=0）：CRECNT = (16 * 目标频率(Hz)) / 32768

高频模式（CREHF=1）：CRECNT = (8 * 目标频率(Hz)) / 32768

（详细设置见范例程序）

9.1.3 CRE 校准误差值寄存器（CRERES）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CRERES	7EFDABH	CRERES[7:0]							

CRERES[7:0]：8位校准误差值（解析度控制）。

由于内部高速IRC的解析度远低于外部的32.768K晶振，最终的校准值无法与CRECNT所设置的目标值完全一致，所以必须通过CRERES寄存器设定一个误差范围。

校准误差计算公式：

CRERES = 误差范围(%) * 目标校准值

（误差范围一般控制在 1%~0.3%即可，不建议超出此范围）

（详细设置见范例程序）

9.2 内部高速 IRC 的 4 个频段的中心频率，供参考

单位（MHz）		样品 1	样品 2	样品 3	样品 4	平均值
6M 频段	最小频率	4.39	4.42	4.43	4.61	4.46
	中间频率	6.22	6.26	6.20	6.53	6.30
	最大频率	8.03	8.05	8.08	8.36	8.13
10M 频段	最小频率	7.20	7.29	7.34	7.74	7.39
	中间频率	10.11	10.26	10.33	10.91	10.40
	最大频率	12.99	13.11	13.26	13.90	13.32
27M 频段	最小频率	20.06	20.11	20.46	21.11	20.43
	中间频率	27.82	27.67	28.24	29.14	28.22
	最大频率	35.14	34.97	35.75	36.83	35.67
44M 频段	最小频率	27.79	28.20	28.72	29.70	28.60
	中间频率	41.45	41.74	42.55	44.04	42.45
	最大频率	54.04	54.41	55.50	57.47	55.35

9.3　范例程序

9.3.1　自动校准内部高速 IRC（HIRC）

例如：校准的目标频率为40MHzMHz，校准误差范围为±0.5%

则需要将CREHF设置为0，CRECNT设置为(16*40000000)/32768＝19531（4C4B0H），

即将CRECNTH设置为4CH，CRECNTL设置为4BH，CRERES设置为19531 * 0.5%＝98（62H）

```
//测试工作频率为 11.0592MHz
#include “Ai8051U.H”                          //头文件见下载软件

#define   FOSC        22118400UL
#define   FOSC        35000000UL
#define   FOSC        40000000UL
#define   FOSC        42000000UL

#define   CNT         ((16 * (FOSC)) / 32768)  //校准目标频率为 22.1184M
#define   RES         (CNT *5 / 1000)          //设置校准误差为 0.5%

void main()
{
    EAXFR = 1;                                 //允许访问扩展的特殊寄存器，XFR
                                               //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                             //允许访问扩展的特殊寄存器，XFR
                                               //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                  //设置取程序代码等待时间，
                                               //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                 //设置访问片内的 xdata 速度，
                                               //赋值为 0 表示用最快速度访问，不增加额外的等待时间
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    X32KCR = 0xc0;                             //启动外部 32K 晶振
    while (!(X32KCR & 1));                     //等待时钟稳定

    IRCBAND &= ~0x03;
    IRCBAND |= 0x02;                           //选择 27M 频段
    CLKSEL = 0x00;                             //选择内部高速 HIRC 为系统时钟
    CRECNTH = CNT22M >> 8;                     //设置目标校准值
    CRECNTL = CNT22M;
    CRERES = RES22M;                           //设置校准误差
    CRECR = 0x90;                              //使能 CRE 功能，并设置校准周期为 4ms

    while (1)
    {
        if (CRECR & 0x01)
        {
```

```
                                        //频率自动校准完成
            }
        }
}
```

10 复位、看门狗、掉电唤醒专用定时器与电源管理

10.1 系统复位

Ai8051U 系列单片机的复位分为硬件复位和软件复位两种。

硬件复位时，所有的寄存器的值会复位到初始值，系统会重新读取所有的硬件选项。同时根据硬件选项所设置的上电等待时间进行上电等待。硬件复位主要包括：

- 上电复位
- 低压复位
- 复位脚复位（低电平复位）
- 看门狗复位

软件复位时，除与时钟相关的寄存器保持不变外，其余的所有寄存器的值会复位到初始值，软件复位不会重新读取所有的硬件选项。软件复位主要包括：

- 写 IAP_CONTR 的 SWRST 所触发的复位

相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
WDT_CONTR	看门狗控制寄存器	C1H	WDT_FLAG	-	EN_WDT	CLR_WDT	IDL_WDT	WDT_PS[2:0]			0x00,0000
IAP_CONTR	IAP 控制寄存器	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	SWBS2	-			0000,x000
RSTCFG	复位配置寄存器	FFH	-	ENLVR	-	P47RST	-	-	LVDS[1:0]		0000,0000

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
RSTFLAG	复位标志寄存器	7EFE99H	-	-	-	LVDRSTF	WDTRSTF	SWRSTF	ROMOVF	EXRSTF	xxx1,0100
RSTCR0	复位控制寄存器 0	7EFE9AH	-	-	-	-	RSTTM34	TSTTM2	-	RSTTM01	xxxx,00x0
RSTCR1	复位控制寄存器 1	7EFE9BH	-	-	-	-	RSTUR4	RSTUR3	RSTUR2	RSTUR1	xxxx,0000
RSTCR2	复位控制寄存器 2	7EFE9CH	RSTCAN2	RSTCAN	RSTLIN	RSTRTC	RSTPWMB	RSTPWMA	RSTI2C	RSTSPI	0000,0000
RSTCR3	复位控制寄存器 3	7EFE9DH	RSTFPU	RSTDMA	RSTLCM	RSTLCD	RSTLED	RSTTKS	RSTCMP	RSTADC	0000,0000
RSTCR4	复位控制寄存器 4	7EFE9EH	-	-	-	-	-	-	-	RSTMDU	xxxx,xxx0
RSTCR5	复位控制寄存器 5	7EFE9FH	-	-	-	-	-	-	-	-	xxxx,xxxx

10.1.1 看门狗控制寄存器（WDT_CONTR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
WDT_CONTR	C1H	WDT_FLAG	-	EN_WDT	CLR_WDT	IDL_WDT	WDT_PS[2:0]		

WDT_FLAG：看门狗溢出标志

看门狗发生溢出时，硬件自动将此位置 1，需要软件清零。

EN_WDT：看门狗使能位

0：对单片机无影响

1：启动看门狗定时器

CLR_WDT：看门狗定时器清零

0：对单片机无影响

1：清零看门狗定时器，硬件自动将此位复位

IDL_WDT：IDLE 模式时的看门狗控制位

0：IDLE 模式时看门狗停止计数

1：IDLE 模式时看门狗继续计数

WDT_PS[2:0]：看门狗定时器时钟分频系数

WDT_PS[2:0]	分频系数	12M 主频时的溢出时间	20M 主频时的溢出时间
000	2	≈ 65.5 毫秒	≈ 39.3 毫秒
001	4	≈ 131 毫秒	≈ 78.6 毫秒
010	8	≈ 262 毫秒	≈ 157 毫秒
011	16	≈ 524 毫秒	≈ 315 毫秒
100	32	≈ 1.05 秒	≈ 629 毫秒
101	64	≈ 2.10 秒	≈ 1.26 秒
110	128	≈ 4.20 秒	≈ 2.52 秒
111	256	≈ 8.39 秒	≈ 5.03 秒

看门狗溢出时间计算公式如下：

$$\text{看门狗溢出时间} = \frac{12 \times 32768 \times 2^{(\text{WDT_PS}+1)}}{\text{SYSclk}}$$

10.1.2 IAP 控制寄存器（IAP_CONTR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CONTR	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	SWBS2	-	-	-

SWBS/SWBS2：软件复位启动选择

SWBS	SWBS2	复位	说明
0	0	软件复位后从用户程序区开始执行代码	用户数据区的数据保持不变
0	1	软件复位后从用户系统区开始执行代码	用户数据区的数据保持不变
1	x	软件复位后从系统 ISP 区开始执行代码	用户数据区的数据会被初始化

SWRST：软件复位触发位

0：对单片机无影响

1：触发软件复位

范例程序：

源程序区 （从此程序区）	目标程序区 （复位到此程序区）	代码
用户程序区	系统 ISP 区	IAP_CONTR = 0x60;
用户系统区		
用户程序区	用户系统区	IAP_CONTR = 0x28;
系统 ISP 区		
用户系统区	用户程序区	IAP_CONTR = 0x20;
系统 ISP 区		

10.1.3 复位配置寄存器（RSTCFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
RSTCFG	FFH	-	ENLVR	-	P47RST	-	-	LVDS[1:0]	

ENLVR：低压复位控制位

0：禁止低压复位。当系统检测到低压事件时，会产生低压中断

1：使能低压复位。当系统检测到低压事件时，自动复位

P47RST：RST 管脚功能选择

0：RST 管脚用作普通 I/O 口（P4.7）

1：RST 管脚用作复位脚（低电平复位）

LVDS[1:0]：低压检测门槛电压设置

LVDS[1:0]	低压检测门槛电压
00	2.0V
01	2.4V
10	2.7V
11	3.0V

10.1.4　复位标志寄存器（RSTFLAG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
RSTFLAG	7EFE99H	-			LVDRSTF	WDTRSTF	SWRSTF	ROMOVF	EXRSTF

LVDRSTF：LVD 低压复位标志

读 0：无意义

读 1：当前的复位是由 LVD 低压复位所触发（上电复位默认值为 1）

写 0：无效果

写 1：清除 LVDRST 标志位

WDTRSTF：看门狗复位标志

读 0：无意义（上电复位默认值为 0）

读 1：当前的复位是由看门狗溢出所触发

写 0：无效果

写 1：清除 WDTRST 标志位

SWRSTF：软复位标志

读 0：无意义

读 1：当前的复位是由软件写 SWRST（IAP_CONTR.5）所触发（用户程序复位默认值为 1）

写 0：无效果

写 1：清除 SWRST 标志位

ROMOVF：代码区溢出标志

读 0：无意义（上电复位默认值为 0）

读 1：当前的复位是由于 CPU 执行代码到非程序区导致的代码区溢出所触发

写 0：无效果

写 1：清除 ROMOV 标志位

EXRSTF：外部复位标志

读 0：无意义（上电复位默认值为 0）

读 1：当前的复位是外部复位脚（P4.7/RST）被拉低所触发

写 0：无效果

写 1：清除 EXRST 标志位

关于用户程序软复位到系统区进行 USB-ISP 下载的说明：

上电时需要 P3.2 同时接地才可进入 USB-ISP 下载模式，为方便用户自主控制 ISP 下载，特别增加：当用户程序软复位到系统区时，可不用 P3.2 接地就可进行 USB-ISP 下载。此功能的判断方式为进入 ISP 后判断 SWRSTF（RSTFLAG.2）是否为 1，若为 1 表示是用户软复位到系统区，则不用 P3.2 接地，否则需要 P3.2 接地。若用户需要按键复位或者看门狗复位或者低压复位后需要进行 USB-ISP 下载，可保持 SWRSTF（RSTFLAG.2）为 1，否则请在用户代码初始化时将 SWRSTF（RSTFLAG.2）写 1，以清零 SWRSTF（RSTFLAG.2）。

详细各种情况说明见下表

基本情况介绍：上电冷启动后，硬件自动将 SWRSTF（RSTFLAG.2）标志位清 0，并进入到系统程序区，此时无论是否进行 ISP 下载，从系统程序区复位到用户程序区，硬件都会自动将 SWRSTF 标志位置 1。用户程序中可对 SWRSTF 标志位写 1 清 0，也可不对 SWRSTF 标志位写 1 清 0，即维持 SWRSTF 标志位的值为 1。

SWRSTF 标志位 （RSTFLAG.2）	复位情况及 USB 下载情况	USB 下载是否 需要 P3.2 接地
重新上电复位， 复位脚按键复位， 看门狗复位， LVD 低压复位 都会复位到系统程序区， 最后都会从系统程序区 软复位到用户程序区，并 将 SWRSTF 标志位置 1 如果用户程序中对 SWRSTF 写 1 则会清 0	从用户程序区软复位到系统区时，硬件自动将 SWRSTF 软件标志位重新置 1，此时系统程序判断到此位为 1， 此时 USB 下载不需要 P3.2 接地	不需要
	复位脚按键复位，看门狗复位，低压复位， 复位到系统区复位到系统区，系统程序判断到此位为 0， 此时 USB 下载需要 P3.2 接地	需要
	重新上电复位到系统区，硬件自动将 SWRSTF 复位为 0， 系统程序判断到此位为 0，此时 USB 下载需要 P3.2 接地	需要
重新上电复位， 复位脚按键复位， 看门狗复位， LVD 低压复位 都会复位到系统程序区， 最后都会从系统程序区 软复位到用户程序区，并 将 SWRSTF 标志位置 1 如果用户程序不对 SWRSTF 写 1 清 0， SWRSTF 保持为 1	从用户程序区软复位到系统区时，硬件自动将 SWRSTF 软件标志位重新置 1，此时系统程序判断到此位为 1， 此时 USB 下载不需要 P3.2 接地	不需要
	复位脚按键复位，看门狗复位，低压复位， 复位到系统区复位到系统区，系统程序判断到此位为 1， 此时 USB 下载不需要 P3.2 接地	不需要
	重新上电复位到系统区，硬件自动将 SWRSTF 复位为 0， 系统程序判断到此位为 0，此时 USB 下载需要 P3.2 接地	需要

总结：

1、用户程序软件复位到系统区，无论用户软件是否对 SWRSTF 写 1 清 0，USB 下载均不需要 P3.2 接地
2、对芯片重新上电复位到系统区，无论用户软件是否对 SWRSTF 写 1 清 0，USB 下载均需要 P3.2 接地
3、按键复位/看门狗复位/低压复位到系统区，如果用户软件对 SWRSTF 写 1 清 0，则 USB 下载需要 P3.2 接地；如果用户不对 SWRSTF 写 1 清 0，保持 SWRSTF 为 1，则 USB 下载不需要 P3.2 接地

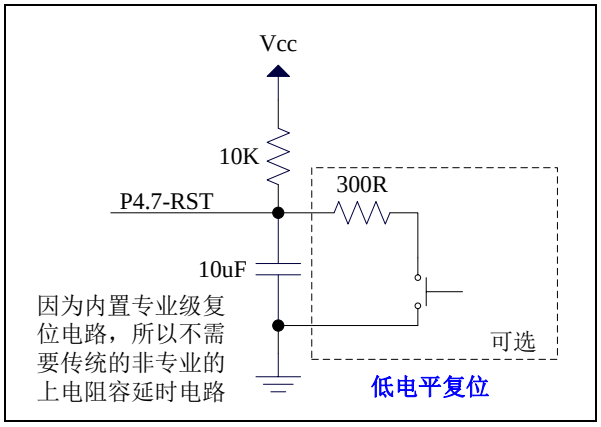
10.1.5 复位控制寄存器（RSTCRx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
RSTCR0	7EFE9AH	-	-	-	-	RSTTM34	RSTTM2	-	RSTTM01
RSTCR1	7EFE9BH	-	-	-	-	RSTUR4	RSTUR3	RSTUR2	RSTUR1
RSTCR2	7EFE9CH	RSTCAN2	RSTCAN	RSTLIN	RSTRTC	RSTPWMB	RSTPWMA	RSTI2C	RSTSPI
RSTCR3	7EFE9DH	RSTFPU	RSTDMA	RSTLCM	RSTLCD	RSTLED	RSTTKS	RSTCMP	RSTADC
RSTCR4	7EFE9EH	-	-	-	-	-	-	-	RSTMDU

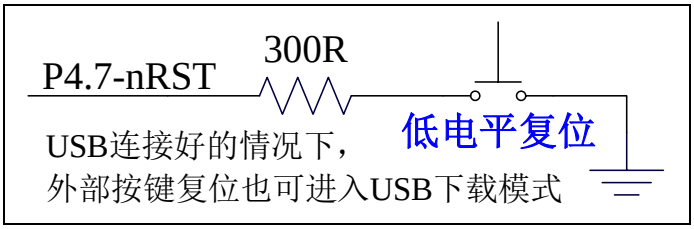
RSTTM01：TIMER0/1 复位控制位
RSTTM2：TIMER2 复位控制位
RSTTM34：TIMER3/4 复位控制位
RSTUARTn：UART1/2/3/4 复位控制位
RSTSPI：SPI 复位控制位
RSTI2C：I2C（SSB）复位控制位
RSTPWMA：PWMA 复位控制位
RSTPWMB：PWMB 复位控制位
RSTRTC：RTC 复位控制位
RSTLIN：LIN 复位控制位
RSTCAN：CAN 复位控制位
RSTCAN2：CAN2 复位控制位
RSTADC：ADC 复位控制位
RSTCMP：CMP（比较器）复位控制位
RSTTKS：TKS（TouchKey）复位控制位
RSTLED：LED 驱动复位控制位
RSTLCD：LCD 驱动复位控制位
RSTLCM：TFT 彩屏驱动复位控制位
RSTDMA：DMA 复位控制位
RSTFPU：FPU 复位控制位
RSTMDU：MDU32 复位控制位

写 1：复位相应的外设模块模块，需软件清零

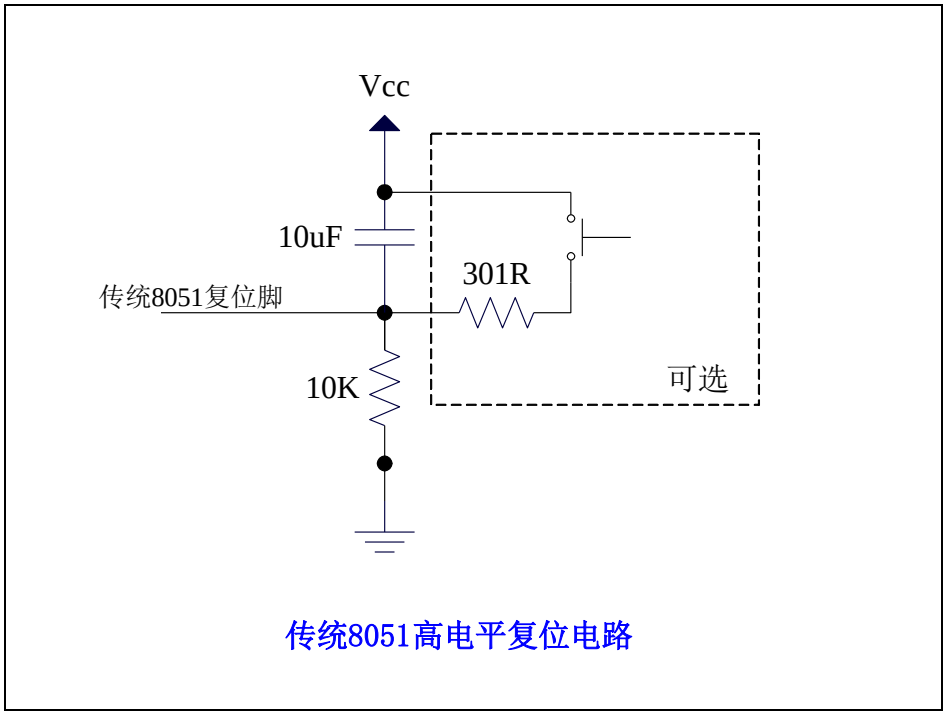
10.1.6　内置专业级复位电路，不需传统的阻容式上电延时电路



10.1.7　外部低电平按键复位电路



10.1.8　传统 8051 高电平上电复位参考电路



上图为传统 8051 的高电平复位电路，Ai8051U 的复位为低电平复位，与传统复位电路不同

10.2 主时钟停振/省电模式，系统电源管理

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
PCON	电源控制寄存器	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000

10.2.1 电源控制寄存器（PCON）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PCON	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL

LVDF：低压检测标志位。当系统检测到低压事件时，硬件自动将此位置 1，并向 CPU 提出中断请求。此位需要用户软件清零。

POF： 上电标志位。当硬件自动将此位置 1。

PD： 主时钟停振/省电模式控制位

0： 无影响

1： 单片机进入主时钟停振/省电模式，CPU 以及全部外设均停止工作。唤醒后硬件自动清零。

注： 省电模式/主时钟停振模式，俗称掉电模式：

仅主时钟停止振荡，依赖主时钟工作的 CPU/及大部分外设功能模块，

无时钟无法继续工作，耗电的这部分功能模块工作被暂停后，MCU 几乎没有功耗。

不依赖主时钟工作的低功耗外设可继续工作，如：

RTC 如选择的是外部 32768RTC 晶振，RTC 会继续正常工作，

RTC 工作，仅需要 1.5uA 的电流。

段码 LCD，如需要继续工作，功耗也极低，不含屏的功耗，也仅需 1.5uA。

IDL：IDLE（空闲）模式控制位

0：无影响

1：单片机进入 IDLE 模式，只有 CPU 停止工作，其他外设依然在运行。唤醒后硬件自动清零

【特别注意】

Ai8051U系列单片机进入主时钟停振/省电模式前，必须根据工作频率正确设置IAP_TPS寄存器（若工作频率为12MHz，则需要将IAP_TPS设置为12；若工作频率为24MHz，则需要将IAP_TPS设置为24，其他频率以此类推）。

单片机从主时钟停振/省电模式唤醒后，硬件会自动对主时钟恢复起振，当主时钟开始起振后，硬件会自动等待一定的主时钟震荡周期数，这个等待的主时钟震荡周期数由IRCDB寄存器进行设置，直到主时钟最后稳定。主时钟稳定后，在开始运行用户程序前，还需要对内部的FLASH程序存储器进行唤醒，FLASH唤醒的等待时间基准是1us，要产生1us的时间基准，软件必须根据主时钟的工作频率正确设置IAP_TPS寄存器。

强烈建议用户在进行系统初始化时对IAP_TPS进行初始化。

10.2.2 内部高速 IRC 时钟恢复振荡到稳定需要等待的时钟数控制寄存器（IRCDB）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IRCDB	FE06H								

IRCDB[7:0]：内部高速 IRC 时钟从主时钟停振/省电模式（掉电模式）恢复振荡到稳定需要等待的时钟数控制寄存器。**上电复位后初始值为 80H。**

IRCDB	系统时钟数
0	255 个时钟
1	0 个时钟
2	1 个时钟
3	2 个时钟
…	…
128	**127 个时钟（上电初始值）**
…	…
255	254 个时钟

【寄存器说明】：

1、芯片使用内部高速 IRC 时钟作为系统时钟时，当芯片从主时钟停振/省电模式（掉电模式）被唤醒后，系统会等待 IRCDB 寄存器设置的等待时钟数，再将时钟提供给 CPU 和系统，CPU 获得时钟后开始继续运行用户程序。建议在主时钟停振语句的下一句后面增加 7~9 个 NOP。

2、需要设置等待多少个时钟再给 CPU 供应时钟，让 CPU 开始跑程序，根据用户实际使用的工作频率来确定，目前测试发现运行在 33.1776MHz 附近或以下时，必须将 IRCDB 设置为 0x10 时，这样就很稳定。

3、如果用户的工作频率高于 33.1776MHz 时，建议用户程序在进入主时钟停振/省电模式前将内部高速 IRC 频率调低，再进入主时钟停振/省电模式。等芯片再次被唤醒后再将内部高速 IRC 调整到用户需要的工作频率。内部高速 IRC 时钟，出厂时有多种校准值，用户程序可以任意设置芯片出厂时预置的几种校准时钟频率。实际测试发现芯片运行在 40MHz 以下时，将 IRCDB 设置为 0x10 也是稳定的。

4、如果程序需要进入省电模式，程序初始化时必须加上下面这条语句：

```
IRCDB = 0x10;
//内部高速 IRC 时钟，停振后被唤醒后起振，等待多少个时钟数后给 MCU 供应时钟
```

10.3 可以唤醒省电模式/主时钟停振模式的中断资源

唤醒源	相应端口	边沿类型	唤醒 STOP	唤醒 IDLE
RTC 中断			✓	✓
掉电唤醒定时器唤醒，不是中断			✓	✗
定时器 11 中断			✓	✓
外部中断	INT0（P3.2） INT1（P3.3）	下降沿 上升沿+下降沿	✓	✓
	INT2（P3.6） INT3（P3.7） INT4（P3.0）	下降沿	✓	✓
所有的普通 I/O 口都支持的外部中断	所有的 I/O 口	上升沿，下降沿 注意：不支持高电平和低电平唤醒	✓	✓
PCA 的 CCP 脚	CCP0（P1.3/P4.2/P2.0） CCP1（P1.4/P4.3/P2.1） CCP2（P1.1/P4.4/P2.2）	上升沿 下降沿 上升沿+下降沿	✓	✓
串口接收脚	RXD （P3.0/P3.6/P1.6/P4.3） RXD2（P1.0/P4.6） RXD3（P0.0/P5.0） RXD4（P0.2/P5.2）	下降沿 注意：串口接收脚的下降沿唤醒 CPU，如果需要唤醒后能接收到完整的串口数据，需将串口波特率设置到 9600 或以下。	✓	✗
I2C 的数据输入脚	SDA （P1.4/P2.4/P3.3）	下降沿	✓	✗
SPI 的 SS 脚	SS （P1.4/P2.4/P4.0/P3.5）	下降沿	✓	✗
I2S 的数据输入脚	I2SData （P3.4/P1.5/P2.1/P4.1）	下降沿	✓	✗
定时器的外部计数输入管脚	T0（P3.4） T1（P3.5） T2（P1.0） T3（P0.4） T4（P0.6）	下降沿	✓	✗
比较器中断			✓	✓
低压检测中断			✓	✓

10.4 主时钟停振/省电模式，I/O 口如何设置才省电

===主时钟停振/省电模式， Ai8051U 系列如何省电

主时钟停振/省电模式，I/O 口如何设置才省电，进入主时钟停振/省电模式前：

1，不用的 I/O 口，就是浮空的 I/O，关闭数字输入

2，用作模拟输入的口，一般是配置成高阻输入，也必须关闭数字输入

＝＝＝指用作 ADCx 外部模拟输入的 I/O

＝＝＝指用作比较器外部模拟输入的 I/O

3，用作高阻输入的 I/O，也必须关闭数字输入

如你 I/O 外部的输入电平 Vx 在【不是逻辑高的电压，也不是逻辑低的电压】

这时内部数字输入电路就会有翻转，就会有几十 uA 的功耗

关闭数字输入，就不会有功耗

4，I/O 外部是高电平的，你如要工作在输出，你就置高

I/O 外部是低电平的，你如要工作在输出，你就置低

否则两边的电平电位不同，就会水往低处走，有电流流进或流出

如你 I/O 外部的输入电平 Vx 在 MCU_Gnd < Vx < MCU_VCC，这时工作在输出，也会有电流流动所以进省电模式前，必须改设置为高阻输入，并关闭数字输入

5，如有启动 RTC/实时时钟功能，在省电时工作的 MCU，【P5.7/XTALI, P5.6/XTALO】

【P5.7/XTALI，P5.6/XTALO】- 接外部 32768-RTC 晶振，

这 2 个口上电默认是高阻输入，可用户程序配置为高阻输入

省电模式时必须保持高阻输入，并必须关闭数字输入

总之，省电模式时，I/O 尽量高阻输入并关闭数字输入

10.5 掉电唤醒定时器

内部掉电唤醒定时器是一个 15 位的计数器（由{WKTCH[6:0],WKTCL[7:0]}组成 15 位）。用于唤醒处于主时钟停振/省电模式的 MCU。

10.5.1 掉电唤醒定时器计数寄存器（WKTCL，WKTCH）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
WKTCL	AAH								
WKTCH	ABH	WKTEN							

WKTEN：掉电唤醒定时器的使能控制位

0：停用掉电唤醒定时器

1：启用掉电唤醒定时器

注意：读写分别位不同的寄存器（但地址相同），写必须先写 WKTCL，再同时写 WKTCH 和 WKTEN 位，不能用读-修改-写的方式。

如果 Ai8051U 系列单片机内置掉电唤醒专用定时器被允许（通过软件将 WKTCH 寄存器中的 WKTEN 位置 1），当 MCU 进入主时钟停振/省电模式后，掉电唤醒专用定时器开始计数，当计数值与用户所设置的值相等时，掉电唤醒专用定时器将 MCU 唤醒。MCU 唤醒后，程序从上次设置单片机进入主时钟停振/省电模式语句的下一条语句开始往下执行，注意不会进入中断。掉电唤醒之后，可以通过读 WKTCH 和 WKTCL 中的内容获取单片机在主时钟停振/省电模式中的睡眠时间。

这里请注意：用户在寄存器{WKTCH[6:0],WKTCL[7:0]}中写入的值必须比实际计数值少 1。如用户需计数 10 次，则将 9 写入寄存器{WKTCH[6:0],WKTCL[7:0]}中。同样，如果用户需计数 32767 次，则应对{WKTCH[6:0],WKTCL[7:0]}写入 7FFEH（即 32766）。（**计数值 0 和计数值 32767 为内部保留值，用户不能使用**）

内部掉电唤醒定时器固定使用内部低速 IRC 时钟，时钟频率大约为 32KHz。其中掉电唤醒定时器计数一次的时间就是由该时钟决定的。出于功耗考虑，无抗温漂压漂电路，误差较大。用户可以通过读 RAM 区 F8H 和 F9H 的内容（F8H 存放频率的高字节，F9H 存放低字节）来获取内部掉电唤醒专用定时器出厂时所记录的时钟频率。注意该频率为出厂温度（25℃）下的频率，当温度不同时，存在温漂。

掉电唤醒专用定时器计数时间的计算公式如下所示：（F_{wt} 为我们从 RAM 区 F8H 和 F9H 获取到的内部掉电唤醒专用定时器的时钟频率）

$$\text{掉电唤醒定时器定时时间} = \frac{10^6 \times 16 \times \text{计数次数}}{F_{wt}} \text{(微秒)}$$

假设 F_{wt}=32KHz，则有：

{WKTCH[6:0],WKTCL[7:0]}	掉电唤醒专用定时器计数时间
~~0（内部保留）~~	
1	$10^6 \div 32K \times 16 \times (1+1) \approx 1$ 毫秒
9	$10^6 \div 32K \times 16 \times (1+9) \approx 5$ 毫秒
99	$10^6 \div 32K \times 16 \times (1+99) \approx 50$ 毫秒
999	$10^6 \div 32K \times 16 \times (1+999) \approx 0.5$ 秒
4095	$10^6 \div 32K \times 16 \times (1+4095) \approx 2$ 秒
32766	$10^6 \div 32K \times 16 \times (1+32766) \approx 16$ 秒
~~32767（内部保留）~~	

10.6　　省电模式，I/O 口如何设置才省电

省电模式，I/O 口如何设置才省电，进入主时钟停振/省电模式前：

===主时钟停振/省电模式，STC8/STC32 系列如何省电

1、不用的 I/O 口，就是浮空的 I/O，设置为高阻输入，并关闭数字输入，也不怕短路了

2、用作模拟输入的口，一般是配置成高阻输入，也必须关闭数字输入

===指用作 ADCx 外部模拟输入的 I/O

===指用作 比较器外部模拟输入的 I/O

===省电模式时，他外部来个 1.5V 附近变化的电压，

数字部分不关闭数字输入可能就会产生额外的功耗

3、用作高阻输入的 I/O，也必须关闭数字输入

如你 I/O 外部的输入电平 Vx 在 【不是逻辑高的电压，也不是逻辑低的电压】

这时内部数字输入电路就会有翻转，就会有几十 uA 的功耗

关闭数字输入，就不会有功耗

4、I/O 外部是高电平的，你如要工作在输出，你就置高

I/O 外部是低电平的，你如要工作在输出，你就置低

否则两边的电平电位不同，就会水往低处走，有电流流进或流出

如你 I/O 外部的输入电平 Vx 在 MCU_Gnd < Vx < MCU_VCC

这时工作在输出，也会有电流流动

所以进省电模式前，必须改设置为高阻输入，并关闭数字输入

5、如有启动 RTC/实时时钟功能，在省电时工作的 MCU，【P1.7/XTALI, P1.6/XTALO】

【P1.7/XTALI，P1.6/XTALO】- 接外部 32768-RTC 晶振，

这 2 个口上电默认是高阻输入，可用户程序配置为高阻输入

这个场景有外部 32768 晶振在振荡，省电模式时必须保持高阻输入，

===并必须关闭数字输入，否则浮空的口，外部不停的在变化，就会产生额外的功耗

6、MCU 如有 ADC_VRef+，ADC_VRef+不能浮空，否则也会产生额外的电流

总之，省电模式时，I/O 尽量 高阻输入并关闭数字输入

10.7　范例程序

10.7.1　看门狗定时器应用

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件

void main()
{
    EAXFR = 1;                                //允许访问扩展的特殊寄存器，XFR
                                              //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                            //允许访问扩展的特殊寄存器，XFR
                                              //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                 //设置取程序代码等待时间，
                                              //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                //设置访问片内的 xdata 速度，
                                              //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

//  WDT_CONTR = 0x23;                         //使能看门狗,溢出时间约为 0.5s
    WDT_CONTR = 0x24;                         //使能看门狗,溢出时间约为 1s
//  WDT_CONTR = 0x27;                         //使能看门狗,溢出时间约为 8s
    P32 = 0;                                  //测试端口

    while (1)
    {
//      WDT_CONTR = 0x33;                     //清看门狗,否则系统复位
        WDT_CONTR = 0x34;                     //清看门狗,否则系统复位
//      WDT_CONTR = 0x37;                     //清看门狗,否则系统复位

        Display();                            //显示模块
        Scankey();                            //按键扫描模块
        MotorDriver();                        //电机驱动模块
    }
}
```

10.7.2　软复位实现自定义下载

```
//测试工作频率为 11.0592MHz
```

```
#include “Ai8051U.H”                                    //头文件见下载软件

void main()
{
    EAXFR = 1;                                          //允许访问扩展的特殊寄存器，XFR
                                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                      //允许访问扩展的特殊寄存器，XFR
                                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                           //设置取程序代码等待时间，
                                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                          //设置访问片内的 xdata 速度，
                                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P32 = 1;                                            //测试端口
    P33 = 1;                                            //测试端口

    while (1)
    {
        if (!P32 && !P33)
        {
            IAP_CONTR |= 0x60;                          //检查到 P3.2 和 P3.3 同时为 0 时复位到 ISP
        }
    }
}
```

10.7.3　低压检测

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                    //头文件见下载软件

#define   ENLVR         0x40                          //RSTCFG.6
#define   LVD2V0        0x00                          //LVD@2.0V
#define   LVD2V4        0x01                          //LVD@2.4V
#define   LVD2V7        0x02                          //LVD@2.7V
#define   LVD3V0        0x03                          //LVD@3.0V

void Lvd_Isr() interrupt 6
{
    LVDF = 0;                                           //清中断标志
    P32 = ~P32;                                         //测试端口
```

```
}

void main()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    LVDF = 0;                                   //测试端口
//  RSTCFG = ENLVR | LVD3V0;                    //使能 3.0V 时低压复位,不产生 LVD 中断
    RSTCFG = LVD3V0;                            //使能 3.0V 时低压中断
    ELVD = 1;                                   //使能 LVD 中断
    EA = 1;

    while (1);
}
```

10.7.4　省电模式

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                           //头文件见下载软件
#include "intrins.h"                            //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void INT0_Isr() interrupt 0
{
    P34 = ~P34;                                 //测试端口
}

void main()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
```

```
                                                    //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                      //设置访问片内的 xdata 速度，
                                                    //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IRCDB = 0x10;       //内部高速 IRC 时钟，停振后被唤醒后起振，等待多少个时钟数后给 MCU 供应时钟
    IAP_TPS = 40;       //根据系统工作频率设置此寄存器
                        //（如果系统工作频率为 40MHz，则 IAP_TPS 设置为 40；
                        //  如果系统工作频率为 22.1184MHz，则 IAP_TPS 设置为 22）

    EX0 = 1;                                        //使能 INT0 中断,用于唤醒 MCU
    EA = 1;
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    IDL = 1;                                        //MCU 进入 IDLE 模式
//  PD = 1;                                         //MCU 进入主时钟停振/省电模式
    _nop_();
    _nop_();  _nop_();  _nop_();
    _nop_();  _nop_();  _nop_();
    _nop_();  _nop_();  _nop_();
    P35 = 0;

    while (1);
}
```

10.7.5　使用 INT0/INT1/INT2/INT3/INT4 管脚中断唤醒省电模式

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                //头文件见下载软件
#include "intrins.h"                                //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。
void INT0_Isr() interrupt 0
{
    P10 = !P10;                                     //测试端口
}

void INT1_Isr() interrupt 2
{
    P10 = !P10;                                     //测试端口
}
```

```
void INT2_Isr() interrupt 10
{
    P10 = !P10;                                 //测试端口
}

void INT3_Isr() interrupt 11
{
    P10 = !P10;                                 //测试端口
}

void INT4_Isr() interrupt 16
{
    P10 = !P10;                                 //测试端口
}

void main()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IRCDB = 0x10;       //内部高速 IRC 时钟，停振后被唤醒后起振，等待多少个时钟数后给 MCU 供应时钟
    IAP_TPS = 40;       //根据系统工作频率设置此寄存器
                        //（如果系统工作频率为 40MHz，则 IAP_TPS 设置为 40；
                        //  如果系统工作频率为 22.1184MHz，则 IAP_TPS 设置为 22）

    IT0 = 0;                                    //使能 INT0 上升沿和下降沿中断
//  IT0 = 1;                                    //使能 INT0 下降沿中断
    EX0 = 1;                                    //使能 INT0 中断

    IT1 = 0;                                    //使能 INT1 上升沿和下降沿中断
//  IT1 = 1;                                    //使能 INT1 下降沿中断
    EX1 = 1;                                    //使能 INT1 中断

    EX2 = 1;                                    //使能 INT2 下降沿中断
    EX3 = 1;                                    //使能 INT3 下降沿中断
    EX4 = 1;                                    //使能 INT4 下降沿中断

    EA = 1;
```

```
    PD = 1;                                 //MCU 进入主时钟停振/省电模式
    _nop_();                                //主时钟停振/省电模式被唤醒后,MCU 首先会执行此语句
                                            //然后再进入中断服务程序

    _nop_();  _nop_();  _nop_();
    _nop_();  _nop_();  _nop_();
    _nop_();  _nop_();  _nop_();

    while (1)
    {
        P11 = ~P11;
    }
}
```

10.7.6　使用 T0/T1/T2/T3/T4 管脚中断唤醒省电模式

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                       //头文件见下载软件
#include "intrins.h"                        //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void TM0_Isr() interrupt 1
{
    P10 = !P10;                             //测试端口
}

void TM1_Isr() interrupt 3
{
    P10 = !P10;                             //测试端口
}

void TM2_Isr() interrupt 12
{
    P10 = !P10;                             //测试端口
}

void TM3_Isr() interrupt 19
{
    P10 = !P10;                             //测试端口
}

void TM4_Isr() interrupt 20
{
    P10 = !P10;                             //测试端口
}

void main()
{
    EAXFR = 1;                              //允许访问扩展的特殊寄存器，XFR
                                            //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                          //允许访问扩展的特殊寄存器，XFR
                                            //(8 位模式请使用这句，注释上一句)
    WTST = 0;                               //设置取程序代码等待时间，
                                            //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                              //设置访问片内的 xdata 速度，
```

```
                                                    //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IRCDB = 0x10;       //内部高速 IRC 时钟，停振后被唤醒后起振，等待多少个时钟数后给 MCU 供应时钟
    IAP_TPS = 40;       //根据系统工作频率设置此寄存器
                        //（如果系统工作频率为 40MHz，则 IAP_TPS 设置为 40；
                        //  如果系统工作频率为 22.1184MHz，则 IAP_TPS 设置为 22）

    TMOD = 0x00;
    TL0 = 0x66;                                     //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                                        //启动定时器
    ET0 = 1;                                        //使能定时器中断

    TL1 = 0x66;                                     //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                                        //启动定时器
    ET1 = 1;                                        //使能定时器中断

    T2L = 0x66;                                     //65536-11.0592M/12/1000
    T2H = 0xfc;
    T2R = 1;                                        //启动定时器
    ET2 = 1;                                        //使能定时器中断

    T3L = 0x66;                                     //65536-11.0592M/12/1000
    T3H = 0xfc;
    T3R = 1;                                        //启动定时器
    ET3 = 1;                                        //使能定时器中断

    T4L = 0x66;                                     //65536-11.0592M/12/1000
    T4H = 0xfc;
    T4R = 1;                                        //启动定时器
    ET4 = 1;                                        //使能定时器中断

    EA = 1;

    PD = 1;                                         //MCU 进入主时钟停振/省电模式
    _nop_();                                        //掉电唤醒后不会立即进入中断服务程序,
                                                    //而是等到定时器溢出后才会进入中断服务程序

    _nop_();  _nop_();  _nop_();
    _nop_();  _nop_();  _nop_();
    _nop_();  _nop_();  _nop_();

    while (1)
    {
        P11 = ~P11;
    }
```

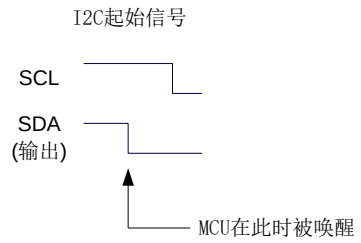
```
}
```

10.7.7　使用 RxD/RxD2/RxD3/RxD4 管脚中断唤醒省电模式

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void UART1_Isr() interrupt 4
{
}

void UART2_Isr() interrupt 8
{
}

void UART3_Isr() interrupt 17
{
}

void UART4_Isr() interrupt 18
{
}

void main()
{
    EAXFR = 1;                                          //允许访问扩展的特殊寄存器，XFR
                                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                      //允许访问扩展的特殊寄存器，XFR
                                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                           //设置取程序代码等待时间，
                                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                          //设置访问片内的 xdata 速度，
                                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IRCDB = 0x10;       //内部高速 IRC 时钟，停振后被唤醒后起振，等待多少个时钟数后给 MCU 供应时钟
    IAP_TPS = 40;       //根据系统工作频率设置此寄存器
                        //（如果系统工作频率为 40MHz，则 IAP_TPS 设置为 40；
                        //  如果系统工作频率为 22.1184MHz，则 IAP_TPS 设置为 22）
```

```
    S1_S1 = 0; S1_S0 = 0;                   //RXD 下降沿唤醒
//  S1_S1 = 0; S1_S0 = 1;                   //RXD_2 下降沿唤醒
//  S1_S1 = 1; S1_S0 = 0;                   //RXD_3 下降沿唤醒
//  S1_S1 = 1; S1_S0 = 1;                   //RXD_4 下降沿唤醒

    S2_S = 0;                               //RXD2 下降沿唤醒
//  S2_S = 1;                               //RXD2_2 下降沿唤醒

    S3_S = 0;                               //RXD3 下降沿唤醒
//  S3_S = 1;                               //RXD3_2 下降沿唤醒

    S4_S = 0;                               //RXD4 下降沿唤醒
//  S4_S = 1;                               //RXD4_2 下降沿唤醒

    ES = 1;                                 //使能串口中断
    ES2 = 1;                                //使能串口中断
    ES3 = 1;                                //使能串口中断
    ES4 = 1;                                //使能串口中断
    EA = 1;

    PD = 1;                                 //MCU 进入主时钟停振/省电模式
    _nop_();                                //掉电唤醒后不会进入中断服务程序,
    _nop_();  _nop_();  _nop_();
    _nop_();  _nop_();  _nop_();
    _nop_();  _nop_();  _nop_();

    while (1)
    {
        P11 = ~P11;
    }
}
```

10.7.8　使用 I2C 的 SDA 脚唤醒 MCU 省电模式




```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                        //头文件见下载软件
#include "intrins.h"                        //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void i2c_isr() interrupt 24
{
    I2CSLST &= ~0x40;
}

void main()
```

```
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IRCDB = 0x10;       //内部高速 IRC 时钟，停振后被唤醒后起振，等待多少个时钟数后给 MCU 供应时钟
    IAP_TPS = 40;       //根据系统工作频率设置此寄存器
                        //（如果系统工作频率为 40MHz，则 IAP_TPS 设置为 40；
                        //  如果系统工作频率为 22.1184MHz，则 IAP_TPS 设置为 22）

    I2C_S1 = 0; I2C_S0 = 0;                     //SDA 下降沿唤醒
//  I2C_S1 = 0; I2C_S0 = 1;                     //SDA_2 下降沿唤醒
//  I2C_S1 = 1; I2C_S0 = 1;                     //SDA_4 下降沿唤醒

    I2CCFG = 0x80;                              //使能 I2C 模块的从机模式
    I2CSLCR = 0x40;                             //使能起始信号中断
    EA = 1;

    PD = 1;                                     //MCU 进入主时钟停振/省电模式
    _nop_();                                    //掉电唤醒后不会进入中断服务程序
    _nop_();  _nop_();  _nop_();
    _nop_();  _nop_();  _nop_();
    _nop_();  _nop_();  _nop_();

    while (1)
    {
        P11 = ~P11;
    }
}
```

10.7.9　使用掉电唤醒定时器唤醒省电模式

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                            //头文件见下载软件
#include "intrins.h"                            //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。
```

```
void main()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IRCDB = 0x10;       //内部高速 IRC 时钟，停振后被唤醒后起振，等待多少个时钟数后给 MCU 供应时钟
    IAP_TPS = 40;       //根据系统工作频率设置此寄存器
                        //（如果系统工作频率为 40MHz，则 IAP_TPS 设置为 40；
                        //  如果系统工作频率为 22.1184MHz，则 IAP_TPS 设置为 22）

    WKTCL = 0xff;                               //设定掉电唤醒时钟约为 1 秒钟
    WKTCH = 0x87;

    while (1)
    {
        _nop_();
        _nop_();
        PD = 1;                                 //MCU 进入主时钟停振/省电模式
        _nop_();
        _nop_();  _nop_();  _nop_();
        _nop_();  _nop_();  _nop_();
        _nop_();  _nop_();  _nop_();
        P11 = ~P11;
    }
}
```

10.7.10　LVD 中断唤醒省电模式，建议配合使用掉电唤醒定时器

主时钟停振/省电模式下，不建议启动 LVD 和比较器，否则硬件系统还会自动启动内部 1.19V 的高精准参考源，这个高精准参考源有有相应的抗温漂和调校线路，大约会额外增加 300uA 的耗电，而 MCU 进入主时钟停振/省电模式后，3.3V 工作电压时只耗约 0.4uA 的电流，所以进入主时钟停振/省电模式时不建议开 LVD 和比较器。如果确实需要用，建议开启掉电唤醒定时器，掉电唤醒定时器只会增加约 1.4uA 的耗电，这个耗电一般系统是可以接受的。让掉电唤醒定时器每 5 秒唤醒一次 MCU，唤醒后可用 LVD、比较器、 ADC 检测外部电池电压，检测工作约耗时 1mS 后再进入主时钟停振/省电模式，这样增加的平均电流小于 1uA，则整体功耗大约为 2.8uA（0.4uA + 1.4uA + 1uA）。

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                  //头文件见下载软件
#include "intrins.h"                                  //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   ENLVR       0x40                            //RSTCFG.6
#define   LVD2V0      0x00                            //LVD@2.0V
#define   LVD2V4      0x01                            //LVD@2.4V
#define   LVD2V7      0x02                            //LVD@2.7V
#define   LVD3V0      0x03                            //LVD@3.0V

void LVD_Isr() interrupt 6
{
    LVDF = 0;                                         //清中断标志
    P10 = !P10;                                       //测试端口
}

void main()
{
    EAXFR = 1;                                        //允许访问扩展的特殊寄存器，XFR
                                                      //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                    //允许访问扩展的特殊寄存器，XFR
                                                      //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                         //设置取程序代码等待时间，
                                                      //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                        //设置访问片内的 xdata 速度，
                                                      //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IRCDB = 0x10;       //内部高速 IRC 时钟，停振后被唤醒后起振，等待多少个时钟数后给 MCU 供应时钟
    IAP_TPS = 40;       //根据系统工作频率设置此寄存器
                        //（如果系统工作频率为 40MHz，则 IAP_TPS 设置为 40；
                        //  如果系统工作频率为 22.1184MHz，则 IAP_TPS 设置为 22）

    LVDF = 0;                                         //上电需要清中断标志
    RSTCFG = LVD3V0;                                  //设置 LVD 电压为 3.0V
    ELVD = 1;                                         //使能 LVD 中断
    EA = 1;

    PD = 1;                                           //MCU 进入主时钟停振/省电模式
    _nop_();                                          //掉电唤醒后立即进入中断服务程序
    _nop_();  _nop_();  _nop_();
    _nop_();  _nop_();  _nop_();
    _nop_();  _nop_();  _nop_();
```

```
    while (1)
    {
        P11 = ~P11;
    }
}
```

10.7.11　比较器中断唤醒省电模式，建议配合使用掉电唤醒定时器

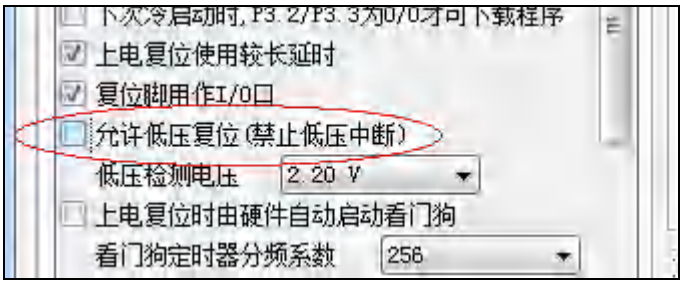
主时钟停振/省电模式下，不建议启动 LVD 和比较器，否则硬件系统还会自动启动内部 1.19V 的高精准参考源，这个高精准参考源有有相应的抗温漂和调校线路，大约会额外增加 300uA 的耗电，而 MCU 进入主时钟停振/省电模式后，3.3V 工作电压时只耗约 0.4uA 的电流，所以进入主时钟停振/省电模式时不建议开 LVD 和比较器。如果确实需要用，建议开启掉电唤醒定时器，掉电唤醒定时器只会增加约 1.4uA 的耗电，这个耗电一般系统是可以接受的。让掉电唤醒定时器每 5 秒唤醒一次 MCU，唤醒后可用 LVD、比较器、 ADC 检测外部电池电压，检测工作约耗时 1mS 后再进入主时钟停振/省电模式，这样增加的平均电流小于 1uA，则整体功耗大约为 2.8uA（0.4uA + 1.4uA + 1uA）。

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件
#include "intrins.h"                          //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void CMP_Isr() interrupt 21
{
    CMPIF = 0;                                //清中断标志
    P10 = !P10;                               //测试端口
}

void main()
{
    EAXFR = 1;                                //允许访问扩展的特殊寄存器，XFR
                                              //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                            //允许访问扩展的特殊寄存器，XFR
                                              //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                 //设置取程序代码等待时间，
                                              //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                //设置访问片内的 xdata 速度，
                                              //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IRCDB = 0x10;       //内部高速 IRC 时钟，停振后被唤醒后起振，等待多少个时钟数后给 MCU 供应时钟
    IAP_TPS = 40;       //根据系统工作频率设置此寄存器
```

```
                    //（如果系统工作频率为 40MHz，则 IAP_TPS 设置为 40；
                    //   如果系统工作频率为 22.1184MHz，则 IAP_TPS 设置为 22）

    CMPCR2 = 0x00;
    CMPEN = 1;                                  //使能比较器模块
    PIE = 1; NIE = 1;                           //使能比较器边沿中断
    CMPOE = 1;                                  //使能比较器输出
    EA = 1;

    PD = 1;                                     //MCU 进入主时钟停振/省电模式
    _nop_();                                    //掉电唤醒后立即进入中断服务程序
    _nop_();  _nop_();  _nop_();
    _nop_();  _nop_();  _nop_();
    _nop_();  _nop_();  _nop_();

    while (1)
    {
        P11 = ~P11;
    }
}
```

10.7.12　使用 LVD 功能检测工作电压（电池电压）

若需要使用 LVD 功能检测电池电压，则在 ISP 下载时需要将低压复位功能去掉，如下图 “允许低压复位（禁止低压中断）”的硬件选项的勾选项需要去掉




```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件
#include "intrins.h"                           //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define FOSC            11059200UL             //定义为无符号长整型,避免计算溢出
#define T1MS            (65536 - FOSC/4/100)

#define   LVD2V0          0x00                 //LVD@2.0V
#define   LVD2V4          0x01                 //LVD@2.4V
#define   LVD2V7          0x02                 //LVD@2.7V
#define   LVD3V0          0x03                 //LVD@3.0V

void delay()
{
    int i;

    for (i=0; i<100; i++)
```

```
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
}

void main()
{
    unsigned char power;

    CKCON = 0x00;                                   //设置外部数据总线速度为最快
    WTST = 0x00;                                    //设置程序代码等待参数，
                                                    //赋值为0 可将 CPU 执行程序的速度设置为最快

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    LVDF = 0;
    RSTCFG = LVD3V0;

    while (1)
    {
        power = 0x0f;

        RSTCFG = LVD3V0;
        delay();
        LVDF = 0;
        delay();
        if (LVDF)
        {
            power >>= 1;
            RSTCFG = LVD2V7;
            delay();
            LVDF = 0;
            delay();
            if (LVDF)
            {
                power >>= 1;
                RSTCFG = LVD2V4;
                delay();
                LVDF = 0;
                delay();
                if (LVDF)
                {
                    power >>= 1;
                    RSTCFG = LVD2V0;
                    delay();
```

```
                        LVDF = 0;
                        delay();
                        if (LVDF)
                        {
                            power >>= 1;
                        }
                    }
                }
            }
            RSTCFG = LVD3V0;
            P2 = ~power;    //P2.3~P2.0 显示电池电量
        }
}
```

10.7.13　使用 RTC 实时时钟中断唤醒省电模式

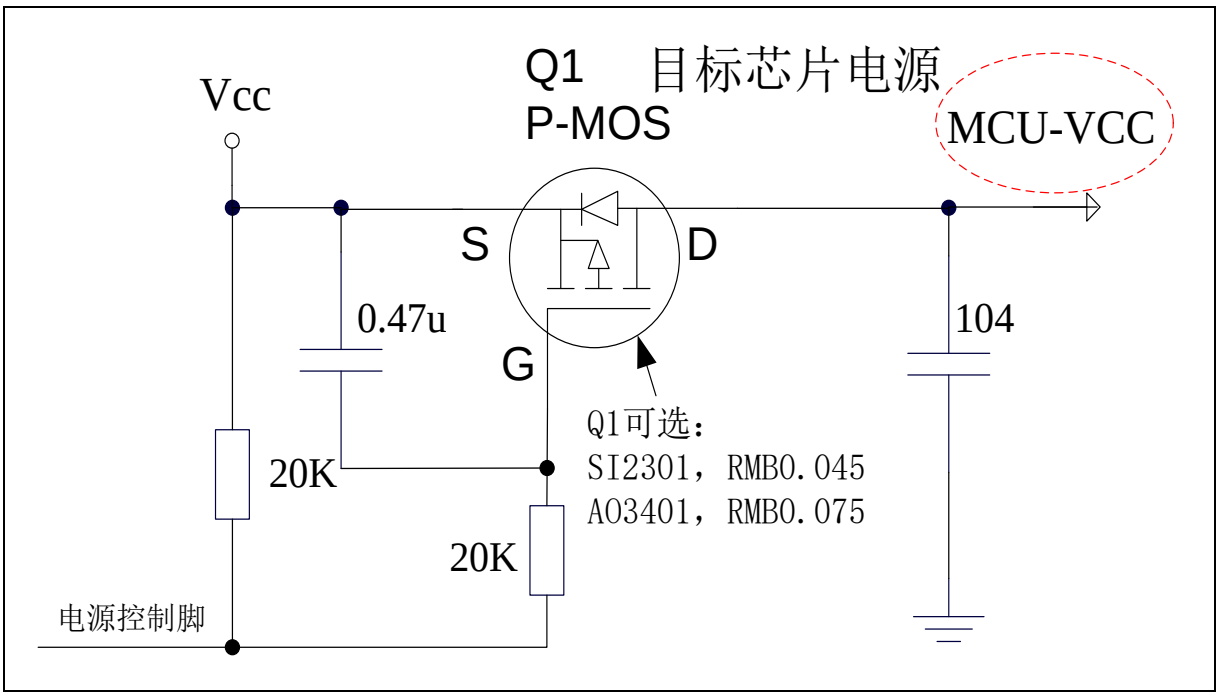
```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                  //头文件见下载软件
#include "intrins.h"                                  //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void main()
{
    EAXFR = 1;                                        //允许访问扩展的特殊寄存器，XFR
                                                      //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                    //允许访问扩展的特殊寄存器，XFR
                                                      //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                         //设置取程序代码等待时间，
                                                      //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                        //设置访问片内的 xdata 速度，
                                                      //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IRCDB = 0x10;       //内部高速 IRC 时钟，停振后被唤醒后起振，等待多少个时钟数后给 MCU 供应时钟
    IAP_TPS = 40;       //根据系统工作频率设置此寄存器
                        //（如果系统工作频率为 40MHz，则 IAP_TPS 设置为 40；
                        //  如果系统工作频率为 22.1184MHz，则 IAP_TPS 设置为 22）

    P40 = 0;

    INIYEAR = 25;
```

```
    INIMONTH = 1;
    INIDAY = 1;
    INIHOUR = 12;
    INIMIN = 0;
    INISEC = 0;
    INISSEC = 0;

    RTCIF = 0;
    RTCIEN = 0;
//  RTCIEN |= 0x01;                 //使能 1/32 秒中断
//  RTCIEN |= 0x02;                 //使能 1/8 秒中断
//  RTCIEN |= 0x04;                 //使能 1/2 秒中断
    RTCIEN |= 0x08;                 //使能 1 秒中断
//  RTCIEN |= 0x10;                 //使能 1 分钟中断
//  RTCIEN |= 0x20;                 //使能 1 小时中断
//  RTCIEN |= 0x40;                 //使能 1 天中断
//  RTCIEN |= 0x80;                 //使能闹钟中断

    IRC32KCR = 0x80;                //启动内部低速 IRC
    while (!(IRC32KCR & 0x01));     //等待时钟稳定
    RTCCFG = 0x02;

//  X32KCR = 0xc0;                  //启动外部 32768 晶振
//  while (!(X32KCR & 0x01));       //等待时钟稳定
//  RTCCFG = 0x00;

    RTCCR = 0x01;                   //启动 RTC 功能
    RTCCFG |= 0x01;
    while (RTCCFG & 0x01);          //等待 RTC 初始化完成

    EA = 1;

    PCON = 0x02;
    _nop_();                        //主时钟停振/省电模式被唤醒后,MCU 首先会执行此语句
                                    //然后再进入中断服务程序
    _nop_();  _nop_();  _nop_();
    _nop_();  _nop_();  _nop_();
    _nop_();  _nop_();  _nop_();

    while (1);
}

void RTC_ISR() interrupt RTC_VECTOR
{
    unsigned char rtcif;

    rtcif = RTCIF;
    RTCIF &= ~rtcif;

    if (rtcif & 0x08)               //RTC 秒中断
    {
        P0++;                       //测试端口
    }
}
```

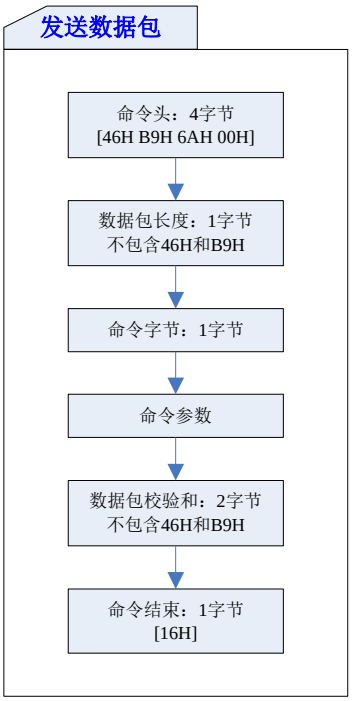
11 第三方对 Ai8051U 下载程序范例及用户程序如何选择内部不同时钟频率

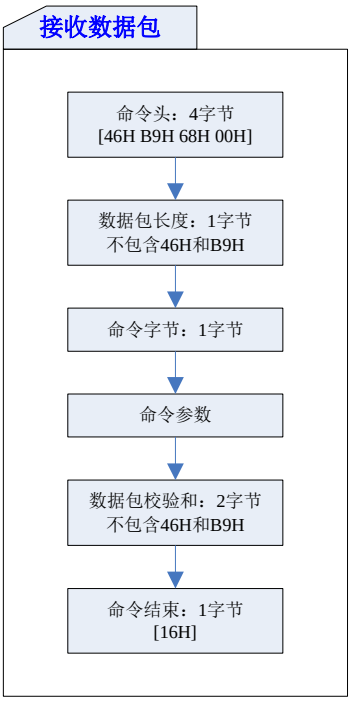
11.1 电源控制参考电路

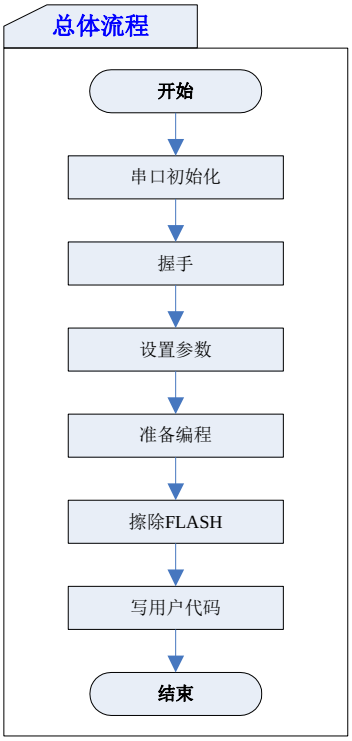
ISP 下载需要对目标进行硬件复位才能进入 ISP 下载模式。当使用第三方 MCU 对芯片进行 ISP 下载时，建议使用下面的电源控制电路来实现。

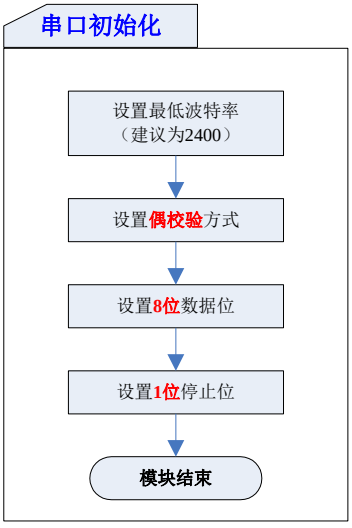


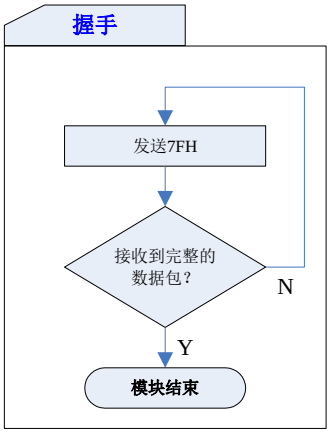
11.2 通信协议流程图

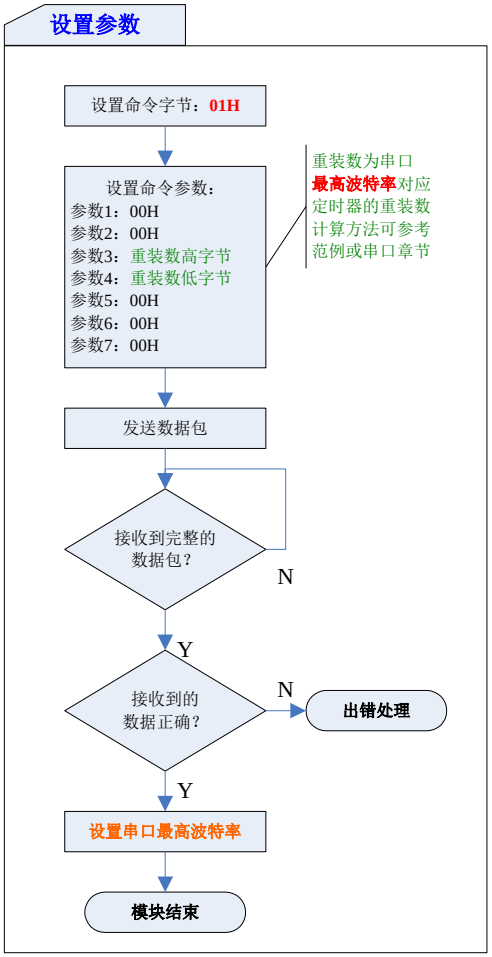

发送数据包
命令头：4字节
[46H B9H 6AH 00H]
数据包长度：1字节
不包含46H和B9H
命令字节：1字节
命令参数
数据包校验和：2字节
不包含46H和B9H
命令结束：1字节
[16H]

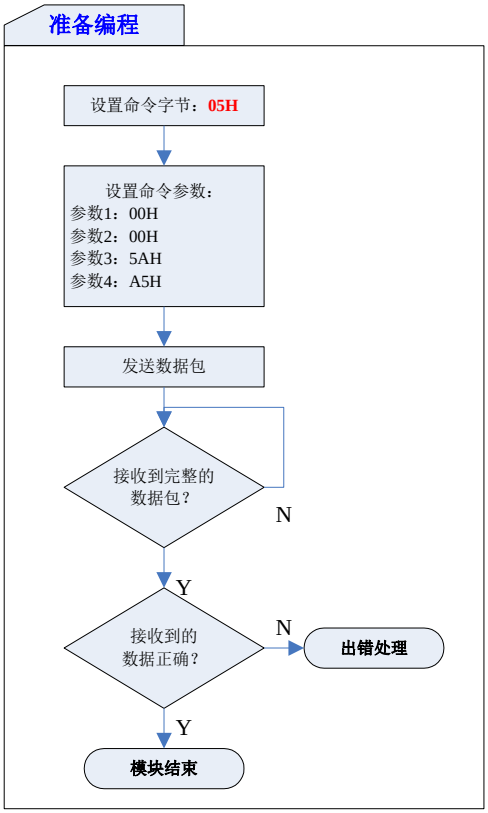

接收数据包
命令头：4字节
[46H B9H 68H 00H]
数据包长度：1字节
不包含46H和B9H
命令字节：1字节
命令参数
数据包校验和：2字节
不包含46H和B9H
命令结束：1字节
[16H]

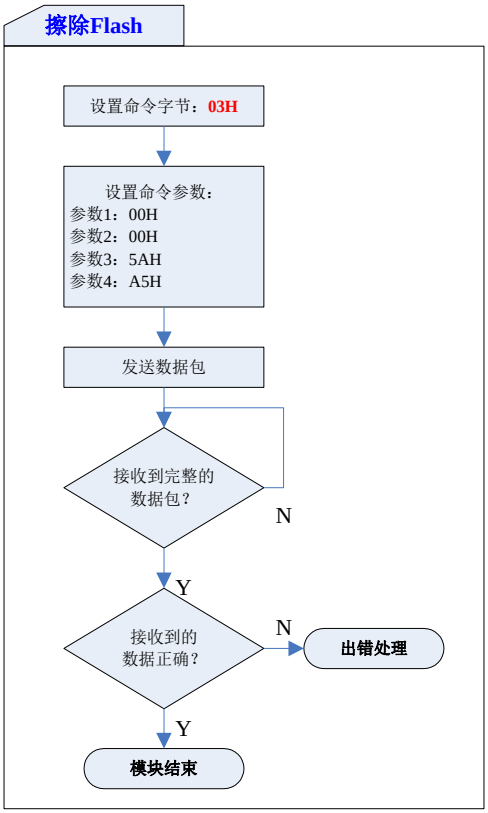

总体流程
开始
串口初始化
握手
设置参数
准备编程
擦除FLASH
写用户代码
结束

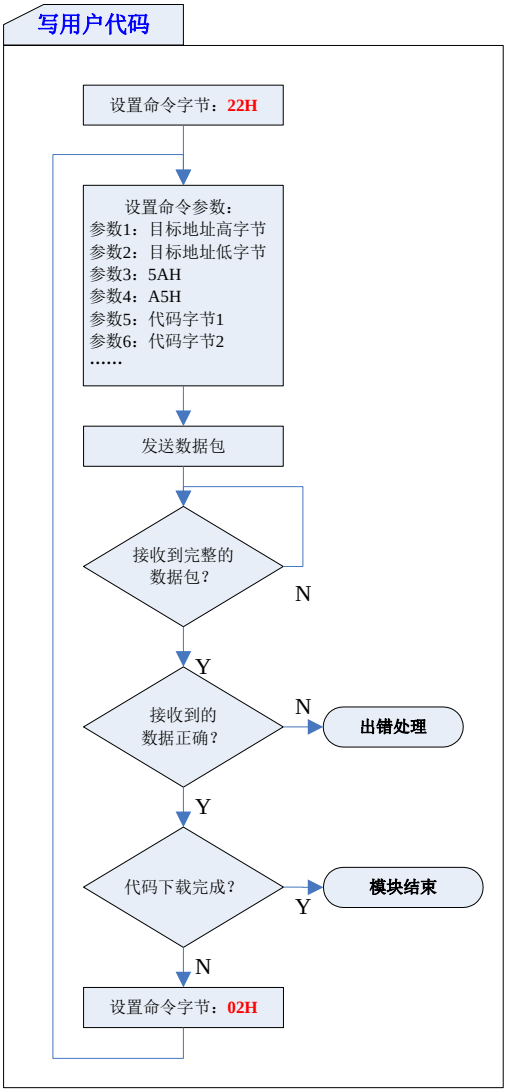

串口初始化
设置最低波特率
（建议为2400）
设置偶校验方式
设置8位数据位
设置1位停止位
模块结束


握手
发送7FH
接收到完整的
数据包？
N
Y
模块结束


设置参数
设置命令字节：01H
设置命令参数：
参数1：00H
参数2：00H
参数3：重装数高字节
参数4：重装数低字节
参数5：00H
参数6：00H
参数7：00H
重装数为串口
最高波特率对应
定时器的重装数
计算方法可参考
范例或串口章节
发送数据包
接收到完整的
数据包？
N
Y
接收到的
数据正确？
N
出错处理
Y
设置串口最高波特率
模块结束


准备编程
设置命令字节：05H
设置命令参数：
参数1：00H
参数2：00H
参数3：5AH
参数4：A5H
发送数据包
接收到完整的
数据包？
N
Y
接收到的
数据正确？
N
出错处理
Y
模块结束


擦除Flash
设置命令字节：03H
设置命令参数：
参数1：00H
参数2：00H
参数3：5AH
参数4：A5H
发送数据包
接收到完整的
数据包？
N
Y
接收到的
数据正确？
N
出错处理
Y
模块结束


写用户代码
设置命令字节：22H
设置命令参数：
参数1：目标地址高字节
参数2：目标地址低字节
参数3：5AH
参数4：A5H
参数5：代码字节1
参数6：代码字节2
……
发送数据包
接收到完整的数据包？
N
Y
接收到的数据正确？
N
出错处理
Y
代码下载完成？
模块结束
Y
N
设置命令字节：02H

11.3 参考代码（C 语言）

C 语言代码

```
//注意:使用本代码对 AI8H 系列的单片机进行下载时,必须要执行了 Download 代码之后,
//才能给目标芯片上电,否则目标芯片将无法正确下载

#include "reg51.h"

typedef     bit             BOOL;
typedef     unsigned char   BYTE;
typedef     unsigned short  WORD;

//宏、常量定义
#define     FALSE           0
#define     TRUE            1
#define     LOBYTE(w)       ((BYTE)(WORD)(w))
#define     HIBYTE(w)       ((BYTE)((WORD)(w) >> 8))

#define     MINBAUD         2400L
#define     MAXBAUD         115200L

#define     FOSC            11059200L           //主控芯片工作频率
#define     BR(n)           (65536 - (FOSC/(n)+2) /4)   //主控芯片串口波特率计算公式
                                                //加 2 操作是为了让 Keil 编译器
                                                //自动实现四舍五入运算

#define     T1MS            (65536 - FOSC/1000)         //主控芯片 1ms 定时初值

#define     FUSER           24000000L           //Ai8051U 系列目标芯片工作频率
#define     RL(n)           (65536 - (FUSER/(n)+2) /4)  //Ai8051U 系列目标芯片串口波特率计算公式
                                                //加 2 操作是为了让 Keil 编译器
                                                //自动实现四舍五入运算

sfr         AUXR = 0x8e;
sfr         P3M1 = 0xB1;
sfr         P3M0 = 0xB2;

//变量定义
BOOL f1ms;                          //1ms 标志位
BOOL UartBusy;                      //串口发送忙标志位
BOOL UartReceived;                  //串口数据接收完成标志位
BYTE UartRecvStep;                  //串口数据接收控制
BYTE TimeOut;                       //串口通讯超时计数器
BYTE xdata TxBuffer[256];           //串口数据发送缓冲区
BYTE xdata RxBuffer[256];           //串口数据接收缓冲区
char code DEMO[256];                //演示代码数据

//函数声明
void Initial(void);
void DelayXms(WORD x);
BYTE UartSend(BYTE dat);
void CommInit(void);
void CommSend(BYTE size);
BOOL Download(BYTE *pdat, long size);

//主函数入口
void main(void)
```

```
{
    P3M0 = 0x00;
    P3M1 = 0x00;

    Initial();
    if (Download(DEMO, 256))
    {
        //下载成功
        P3 = 0xff;
        DelayXms(500);
        P3 = 0x00;
        DelayXms(500);
        P3 = 0xff;
        DelayXms(500);
        P3 = 0x00;
        DelayXms(500);
        P3 = 0xff;
        DelayXms(500);
        P3 = 0x00;
        DelayXms(500);
        P3 = 0xff;
    }
    else
    {
        //下载失败
        P3 = 0xff;
        DelayXms(500);
        P3 = 0xf3;
        DelayXms(500);
        P3 = 0xff;
        DelayXms(500);
        P3 = 0xf3;
        DelayXms(500);
        P3 = 0xff;
        DelayXms(500);
        P3 = 0xf3;
        DelayXms(500);
        P3 = 0xff;
    }

    while (1);
}

//1ms 定时器中断服务程序
void tm0(void) interrupt 1
{
    static BYTE Counter100;

    f1ms = TRUE;
    if (Counter100-- == 0)
    {
        Counter100 = 100;
        if (TimeOut) TimeOut--;
    }
}

//串口中断服务程序
void uart(void) interrupt 4
{
```

```
    static WORD RecvSum;
    static BYTE RecvIndex;
    static BYTE RecvCount;
    BYTE dat;

    if (TI)
    {
        TI = 0;
        UartBusy = FALSE;
    }

    if (RI)
    {
        RI = 0;
        dat  = SBUF;
        switch (UartRecvStep)
        {
        case 1:
            if (dat != 0xb9) goto L_CheckFirst;
            UartRecvStep++;
            break;
        case 2:
            if (dat != 0x68) goto L_CheckFirst;
            UartRecvStep++;
            break;
        case 3:
            if (dat != 0x00) goto L_CheckFirst;
            UartRecvStep++;
            break;
        case 4:
            RecvSum = 0x68 + dat;
            RecvCount = dat - 6;
            RecvIndex = 0;
            UartRecvStep++;
            break;
        case 5:
            RecvSum += dat;
            RxBuffer[RecvIndex++] = dat;
            if (RecvIndex == RecvCount)     UartRecvStep++;
            break;
        case 6:
            if (dat != HIBYTE(RecvSum))     goto L_CheckFirst;
            UartRecvStep++;
            break;
        case 7:
            if (dat != LOBYTE(RecvSum))     goto L_CheckFirst;
            UartRecvStep++;
            break;
        case 8:
            if (dat != 0x16) goto L_CheckFirst;
            UartReceived = TRUE;
            UartRecvStep++;
            break;
L_CheckFirst:
        case 0:
        default:
            CommInit();
            UartRecvStep = (dat == 0x46     ? 1 : 0);
            break;
```

```
            }
        }
}

//系统初始化
void Initial(void)
{
    UartBusy = FALSE;

    SCON = 0xd0;                    //串口数据模式必须为 8 位数据+1 位偶检验
    AUXR = 0xc0;
    TMOD = 0x00;
    TH0 = HIBYTE(T1MS);
    TL0 = LOBYTE(T1MS);
    TR0 = 1;
    TH1 = HIBYTE(BR(MINBAUD));
    TL1 = LOBYTE(BR(MINBAUD));
    TR1 = 1;
    ET0 = 1;
    ES = 1;
    EA = 1;
}

//Xms 延时程序
void DelayXms(WORD x)
{
    do
    {
        f1ms = FALSE;
        while (!f1ms);
    } while (x--);
}

//串口数据发送程序
BYTE UartSend(BYTE dat)
{
    while (UartBusy);

    UartBusy = TRUE;
    ACC = dat;
    TB8 = P;
    SBUF = ACC;

    return dat;
}

//串口通讯初始化
void CommInit(void)
{
    UartRecvStep = 0;
    TimeOut = 20;
    UartReceived = FALSE;
}

//发送串口通讯数据包
void CommSend(BYTE size)
{
    WORD sum;
    BYTE i;
```

```
        UartSend(0x46);
        UartSend(0xb9);
        UartSend(0x6a);
        UartSend(0x00);
        sum = size + 6 + 0x6a;
        UartSend(size + 6);
        for (i=0; i<size; i++)
        {
            sum  += UartSend(TxBuffer[i]);
        }
        UartSend(HIBYTE(sum));
        UartSend(LOBYTE(sum));
        UartSend(0x16);
        while (UartBusy);

        CommInit();
}

//对 Ai8051U 系列的芯片进行 ISP 下载程序
BOOL Download(BYTE *pdat, long size)
{
        BYTE offset;
        BYTE cnt;
        DWORD addr;                     //超 64K 代码空间，地址需定义为 4 字节

        //握手
        CommInit();
        while (1)
        {
            if (UartRecvStep == 0)
            {
                UartSend(0x7f);
                DelayXms(10);
            }
            if (UartReceived)
            {
                if (RxBuffer[0] == 0x50) break;
                return FALSE;
            }
        }

        //设置参数(设置从芯片使用最高的波特率)
        TxBuffer[0] = 0x01;
        TxBuffer[1] = 0x00;
        TxBuffer[2] = 0x00;
        TxBuffer[3] = HIBYTE(RL(MAXBAUD));
        TxBuffer[4] = LOBYTE(RL(MAXBAUD));
        TxBuffer[5] = 0x00;
        TxBuffer[6] = 0x00;
        TxBuffer[7] = 0x97;
        CommSend(8);
        while (1)
        {
            if (TimeOut == 0) return FALSE;
            if (UartReceived)
            {
                if (RxBuffer[0] == 0x01) break;
                return FALSE;
```

```
        }
    }

    //准备
    TH1 = HIBYTE(BR(MAXBAUD));
    TL1 = LOBYTE(BR(MAXBAUD));
    DelayXms(10);
    TxBuffer[0] = 0x05;
    TxBuffer[1] = 0x00;
    TxBuffer[2] = 0x00;
    TxBuffer[3] = 0x5a;
    TxBuffer[4] = 0xa5;
    CommSend(5);
    while (1)
    {
        if (TimeOut == 0) return FALSE;
        if (UartReceived)
        {
            if (RxBuffer[0] == 0x05) break;
            return FALSE;
        }
    }

    //擦除
    DelayXms(10);
    TxBuffer[0] = 0x03;
    TxBuffer[1] = 0x00;
    TxBuffer[2] = 0x00;
    TxBuffer[3] = 0x5a;
    TxBuffer[4] = 0xa5;
    CommSend(5);
    TimeOut = 100;
    while (1)
    {
        if (TimeOut == 0) return FALSE;
        if (UartReceived)
        {
            if (RxBuffer[0] == 0x03) break;
            return FALSE;
        }
    }

    //写用户代码
    DelayXms(10);
    addr = 0;
    TxBuffer[0] = 0x22;
    TxBuffer[3] = 0x5a;
    TxBuffer[4] = 0xa5;
    offset = 5;
    while (addr < size)
    {
        if (addr < 0x10000)                 //程序代码的目标地址信息需从原 HEX 文件中获取
        {
            TxBuffer[0] |= 0x10;            //目标编程地址为 FE:0000~FE:FFFF
        }
        else
        {
            TxBuffer[0] &= ~0x10;           //目标编程地址为 FF:0000~FF:FFFF
        }
```

```
            TxBuffer[1] = HIBYTE(addr);
            TxBuffer[2] = LOBYTE(addr);
            cnt    = 0;
            while (addr < size)
            {
                TxBuffer[cnt+offset] = pdat[addr];
                addr++;
                cnt++;
                if (cnt >= 128) break;
            }
            CommSend(cnt + offset);
            while (1)
            {
                if (TimeOut == 0) return FALSE;
                if (UartReceived)
                {
                    if ((RxBuffer[0] == 0x02) && (RxBuffer[1] == 'T')) break;
                    return FALSE;
                }
            }
            TxBuffer[0] = 0x02;
        }

        //下载完成
        return TRUE;
}

char code DEMO[256] =
{
        0x80,0x00,0x75,0xB2,0xFF,0x75,0xB1,0x00,0x05,0xB0,0x11,0x0E,0x80,0xFA,0xD8,0xFE,
        0xD9,0xFC,0x22,
};
```

11.4　如何在用户程序中设置程序运行时的工作频率

11.4.1　在用户程序区用户如何动态选择不同的系统时钟频率

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                  //头文件见下载软件
#include "intrins.h"                                  //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   T22M_ADDR       CHIPID11                    //22.1184MHz
#define   T24M_ADDR       CHIPID12                    //24MHz
#define   T27M_ADDR       CHIPID13                    //27MHz
#define   T30M_ADDR       CHIPID14                    //30MHz
#define   T33M_ADDR       CHIPID15                    //33.1776MHz
#define   T35M_ADDR       CHIPID16                    //35MHz
#define   T36M_ADDR       CHIPID17                    //36.864MHz
#define   T40M_ADDR       CHIPID18                    //40MHz
#define   T44M_ADDR       CHIPID19                    //44.2368MHz
#define   T48M_ADDR       CHIPID20                    //48MHz
#define   VRT6M_ADDR      CHIPID21                    //VRTRIM_6M
#define   VRT10M_ADDR     CHIPID22                    //VRTRIM_10M
#define   VRT27M_ADDR     CHIPID23                    //VRTRIM_27M
#define   VRT44M_ADDR     CHIPID24                    //VRTRIM_44M

void main()
{
    EAXFR = 1;                                        //允许访问扩展的特殊寄存器，XFR
                                                      //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                    //允许访问扩展的特殊寄存器，XFR
                                                      //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                         //设置取程序代码等待时间，
                                                      //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                        //设置访问片内的 xdata 速度，
                                                      //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

//  //选择 22.1184MHz
//  CLKDIV = 0x04;
//  IRTRIM = T22M_ADDR;
//  VRTRIM = VRT27M_ADDR;
//  IRCBAND &= ~0x03;
//  IRCBAND |= 0x02;
//  CLKDIV = 0x00;
```

```
    //选择 24MHz
    CLKDIV = 0x04;
    IRTRIM = T24M_ADDR;
    VRTRIM = VRT27M_ADDR;
    IRCBAND &= ~0x03;
    IRCBAND |= 0x02;
    CLKDIV = 0x00;

//  //选择 27MHz
//  CLKDIV = 0x04;
//  IRTRIM = T27M_ADDR;
//  VRTRIM = VRT27M_ADDR;
//  IRCBAND &= ~0x03;
//  IRCBAND |= 0x02;
//  CLKDIV = 0x00;

//  //选择 30MHz
//  CLKDIV = 0x04;
//  IRTRIM = T30M_ADDR;
//  VRTRIM = VRT27M_ADDR;
//  IRCBAND &= ~0x03;
//  IRCBAND |= 0x02;
//  CLKDIV = 0x00;

//  //选择 33.1776MHz
//  CLKDIV = 0x04;
//  IRTRIM = T33M_ADDR;
//  VRTRIM = VRT27M_ADDR;
//  IRCBAND &= ~0x03;
//  IRCBAND |= 0x02;
//  CLKDIV = 0x00;

//  //选择 35MHz
//  CLKDIV = 0x04;
//  IRTRIM = T35M_ADDR;
//  VRTRIM = VRT44M_ADDR;
//  IRCBAND |= 0x03;
//  CLKDIV = 0x00;

//  //选择 44.2368MHz
//  CLKDIV = 0x04;
//  IRTRIM = T44M_ADDR;
//  VRTRIM = VRT44M_ADDR;
//  IRCBAND |= 0x03;
//  CLKDIV = 0x00;

//  //选择 48MHz
//  CLKDIV = 0x04;
//  IRTRIM = T48M_ADDR;
//  VRTRIM = VRT44M_ADDR;
//  IRCBAND |= 0x03;
//  CLKDIV = 0x00;

    while (1);
}
```

11.5　如何在用户程序中设置复位脚、低压检测门槛电压等参数

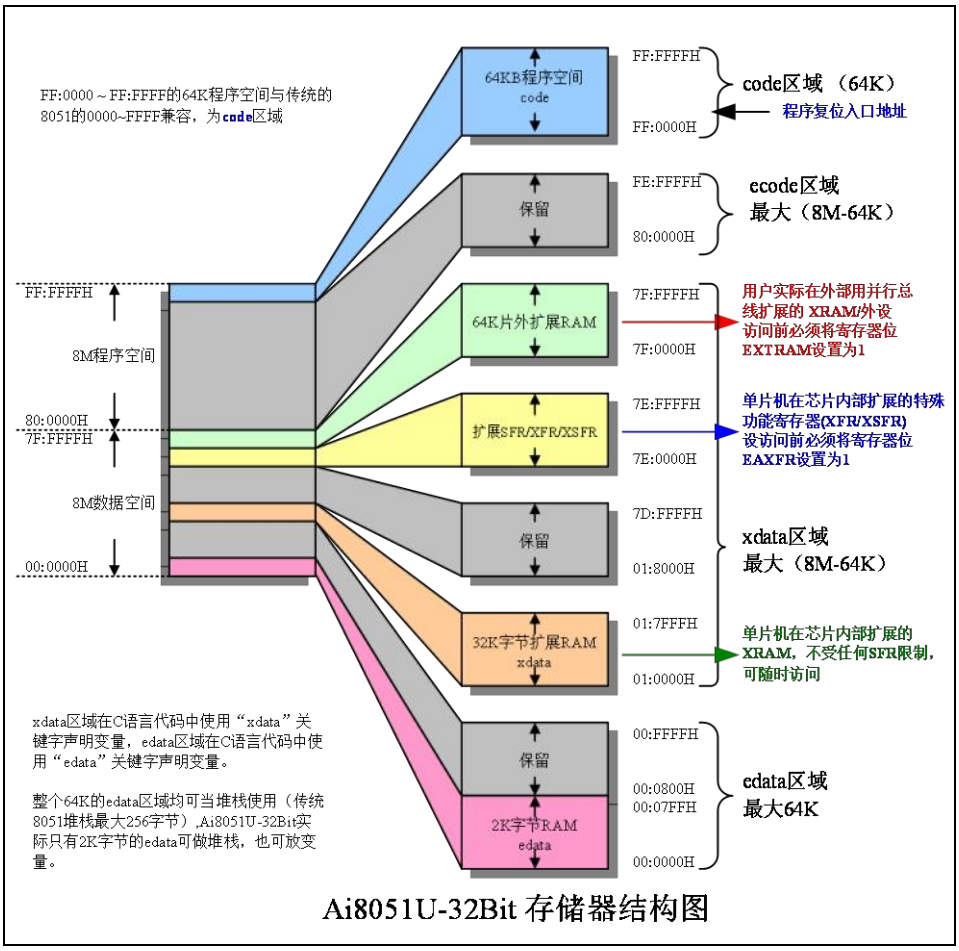
C 语言代码

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件

void main()
{
    EAXFR = 1;                                //允许访问扩展的特殊寄存器，XFR
                                              //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                            //允许访问扩展的特殊寄存器，XFR
                                              //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                 //设置取程序代码等待时间，
                                              //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                //设置访问片内的 xdata 速度，
                                              //赋值为 0 表示用最快速度访问，不增加额外的等待时间
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    RSTCFG |= 0x10;                           //设置 P4.7 为复位脚
//  RSTCFG &= ~0x10;                          //设置 P4.7 为普通 I/O 口

    RSTCFG |= 0x40;                           //使能低压复位功能
//  RSTCFG &= ~0x40;                          //禁止低压复位

    RSTCFG = (RSTCFG & ~0x03) | 0;            //选择低压复位/检测门槛电压为 2.0V
//  RSTCFG = (RSTCFG & ~0x03) | 1;            //选择低压复位/检测门槛电压为 2.4V
//  RSTCFG = (RSTCFG & ~0x03) | 2;            //选择低压复位/检测门槛电压为 2.7V
//  RSTCFG = (RSTCFG & ~0x03) | 3;            //选择低压复位/检测门槛电压为 3.0V

    while (1);
}
```

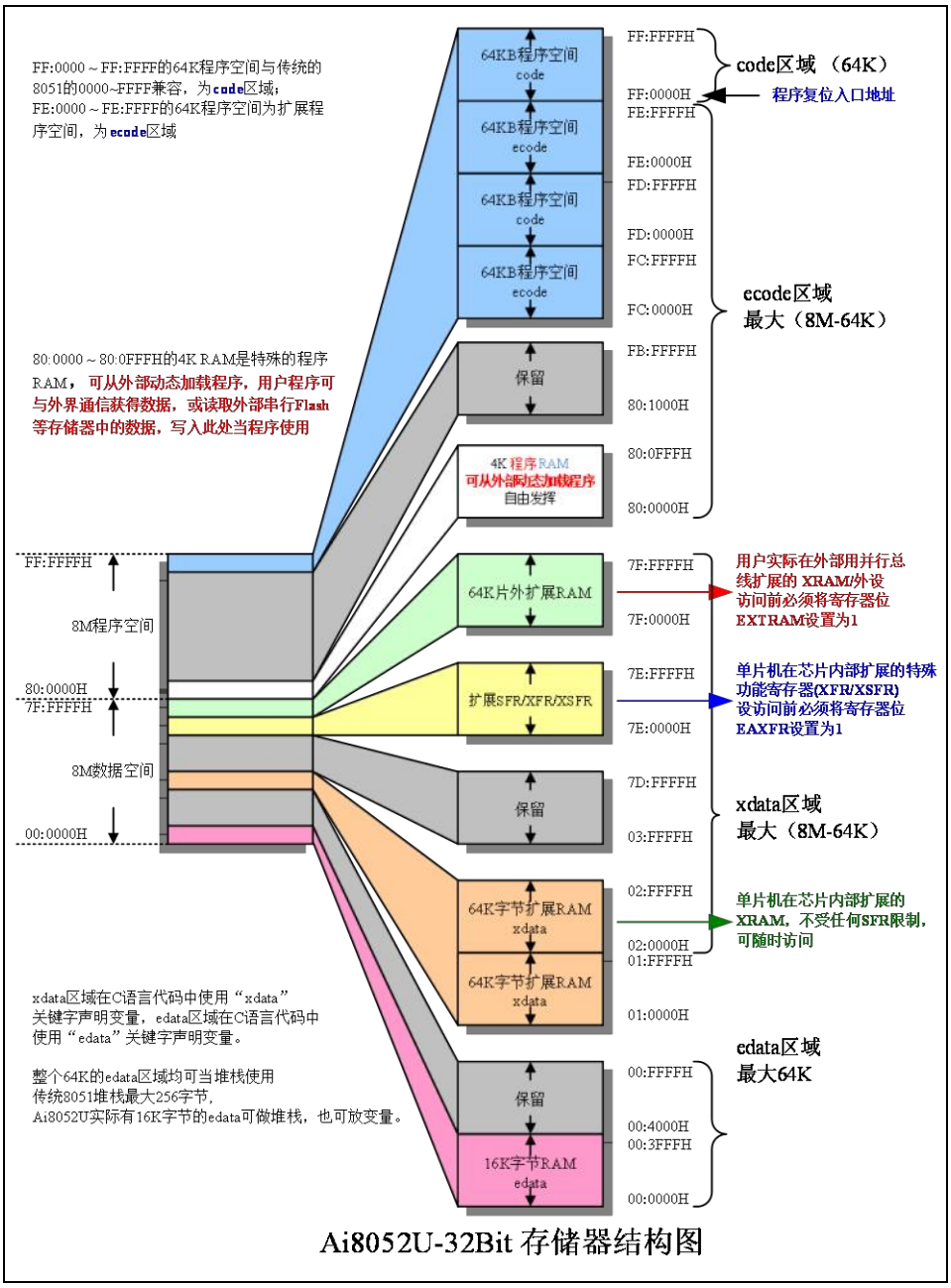
12 存储器（32/16/8 位访问），在哪存储重要参数，及全球唯一 ID 号 CHIPID

Ai8051U 系列单片机的程序存储器和数据存储器是统一编址的。Ai8051U 系列单片机提供 24 位寻址空间，最多能够访问 16M 的存储器（8M 数据存储器+8M 程序存储器）。由于没有提供访问外部程序存储器的总线，所以单片机的所有程序存储器都是片上 Flash 存储器，不能访问外部程序存储器。

Ai8051U 系列单片机内部集成了大容量的数据存储器。Ai8051U 系列单片机内部的数据存储器在物理和逻辑上都分为两个地址空间:内部 RAM（edata）和内部扩展 RAM（xdata）。



Ai8051U-32Bit 存储器结构图

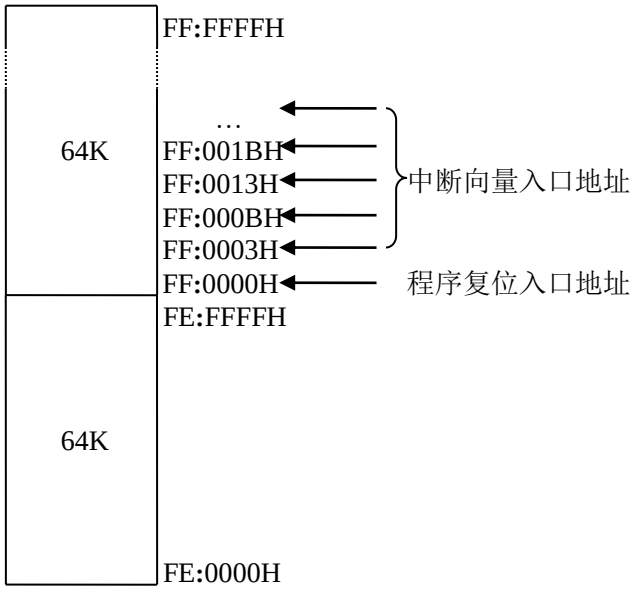


Ai8052U-32Bit 存储器结构图

12.1 程序存储器

程序存储器用于存放用户程序、固定不变的数据以及表格等信息。

单片机系列	Flash 程序存储器（ROM）	地址范围
Ai8051U 系列	64K 字节	FF:0000H~FF:FFFFH



（Ai8051U 系列与传统 8051 中断入口地址对比）

	Ai8051U 系列	传统 8051
复位入口地址	FF:0000H	0000H
INT0 中断入口地址	FF:0003H	0003H
TIMER0 中断入口地址	FF:000BH	000BH
INT1 中断入口地址	FF:0013H	0013H
TIMER1 中断入口地址	FF:001BH	001BH
UART 中断入口地址	FF:0023H	0023H

单片机复位后，程序计数器(PC)的内容为 FF:0000H，从 FF:0000H 单元开始执行程序。另外中断服务程序的入口地址(又称中断向量)也位于程序存储器单元。在程序存储器中，每个中断都有一个固定的入口地址，当中断发生并得到响应后，单片机就会自动跳转到相应的中断入口地址去执行程序。外部中断 0（INT0）的中断服务程序的入口地址是 FF:0003H，定时器/计数器 0（TIMER0）中断服务程序的入口地址是 FF:000BH，外部中断 1（INT1）的中断服务程序的入口地址是 FF:0013H，定时器/计数器 1（TIMER1）的中断服务程序的入口地址是 FF:001BH 等。更多的中断服务程序的入口地址(中断向量)请参考中断介绍章节。

由于相邻中断入口地址的间隔区间仅仅有 8 个字节，一般情况下无法保存完整的中断服务程序，因此在中断响应的地址区域存放一条无条件转移指令，指向真正存放中断服务程序的空间去执行。

Ai8051U 系列单片机中都包含有 Flash 数据存储器（EEPROM）。以字节为单位进行读/写数据，以 512 字节为页单位进行擦除，可在线反复编程擦写 10 万次以上，提高了使用的灵活性和方便性。

12.1.1 程序读取等待控制寄存器（WTST）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
WTST	E9H	WTST[7:0]							

WTST[7:0]：CPU 读取程序存储器的等待时间控制

WTST[7:0]	等待时钟数
0	0 个时钟
1	1 个时钟
…	…
7	7 个时钟
8	保留
…	保留
255	保留

每条指令的实际执行时钟数＝指令时钟数＋程序存储器的等待时钟数

12.2 数据存储器（32 位访问，16 位访问，8 位访问）

Ai8051U 的 edata 区域可对 32-BIT/16-BIT/8-BIT 的数据进行单时钟读写访问，xdata 区域可对 16-BIT/8-BIT 的数据读写访问。edata 区域的 SRAM 目前的最大存储深度已设计为 64K 字节；xdata 区域的 SRAM 最大存储深度为 8M 字节。

将来新增的特殊功能寄存器 32-BIT SFR32（如 ADC_DATA32），如将 SFR32 的逻辑地址映射在 edata 区域，就可以支持对新增特殊功能寄存器的 32-BIT/16-BIT/8-BIT 访问；

将来新增的特殊功能寄存器 16-BIT SFR16（如 ADC_DATA16），如将 SFR16 的逻辑地址映射在 xdata 区域，就可以支持对新增特殊功能寄存器的 16-BIT/8-BIT 访问

Ai8051U 系列单片机内部集成的 RAM 可用于存放程序执行的中间结果和过程数据。

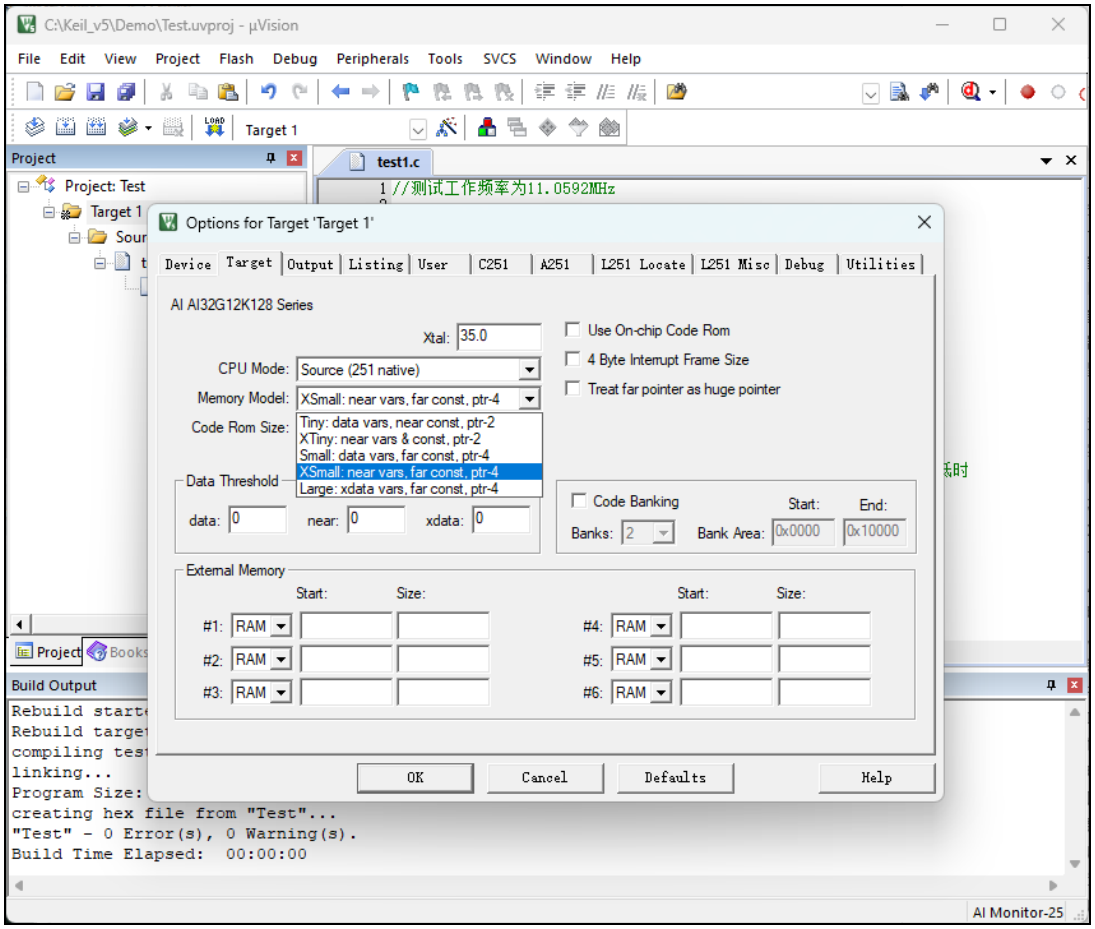
单片机系列	内部 RAM （edata）	内部扩展 RAM （xdata）
Ai8051U 系列	2K 字节	32K 字节

EDATA：单时钟存取，访问速度快，可做堆栈使用，成本高；

XDATA：存取需要 2~3 个时钟，访问速度慢，寻址空间可达 8M，成本低。

12.2.1　Keil 选项 Memory Model 设置

对于 Ai8051U 系列的项目的存储器模式，在 Keil 环境下有如下图所示的 5 种模式：



各种模式对比如下表：

Memory Model	默认变量类型（数据存储器）	默认常量类型（程序存储器）	默认指针变量	指针访问范围
Tiny 模式	data	near	2 字节	00:0000 ~ 00:FFFF
XTiny 模式	edata	near	2 字节	00:0000 ~ 00:FFFF
Small 模式	data	far	4 字节	00:0000 ~ FF:FFFF
XSmall 模式	**edata**	**far**	**4 字节**	**00:0000 ~ FF:FFFF**
Large 模式	xdata	far	4 字节	00:0000 ~ FF:FFFF

由于 AI8051U 的程序逻辑地址为 FF:0000H~FF:FFFFH，需要使用 24 位地址线才能正确访问，默认的常量类型（程序存储器类型）必须使用“far”类型，默认指针变量必须为 4 字节。

不建议使用“Small”“Tiny”和“XTiny”模式，推荐使用“XSmall”模式，这种模式默认将变量定义在内部 RAM(edata)，单时钟存取，访问速度快，且 Ai8051U 系列芯片有 2K 的 edata 可以使用；使用“Small”模式时，默认将变量定义在内部 RAM(data)，data 默认只有 128 字节，当用户对 RAM 需求超过 128 字节时，Keil 编译器会报错，data 区数量有限，容易报错，所以不建议使用；不推荐使用“Large”模式，虽然该模式也能正确访问 Ai8051U 的全部 16M 寻址空间，但“Large”模式默认将变量定义在内部扩展 RAM(xdata)里面，存取需要 2~3 个时钟，访问速度慢

Ai8051U 系列单片机的 C 语言变量声明建议：

1、 当用户变量需求量较小时，建议不要使用“edata、xdata”等关键字声明变量，而使用如下方式直接声明变量：

```
char    bCounter = 1;       //声明字节变量
int     wCounter = 100;     //声明双字节变量
long    dwCounter = 0x1234; //声明 4 字节变量
```

然后在项目选项中将“Memory Model”设置为“XSmall”，让编译器自动将声明的变量分配到 edata 区域

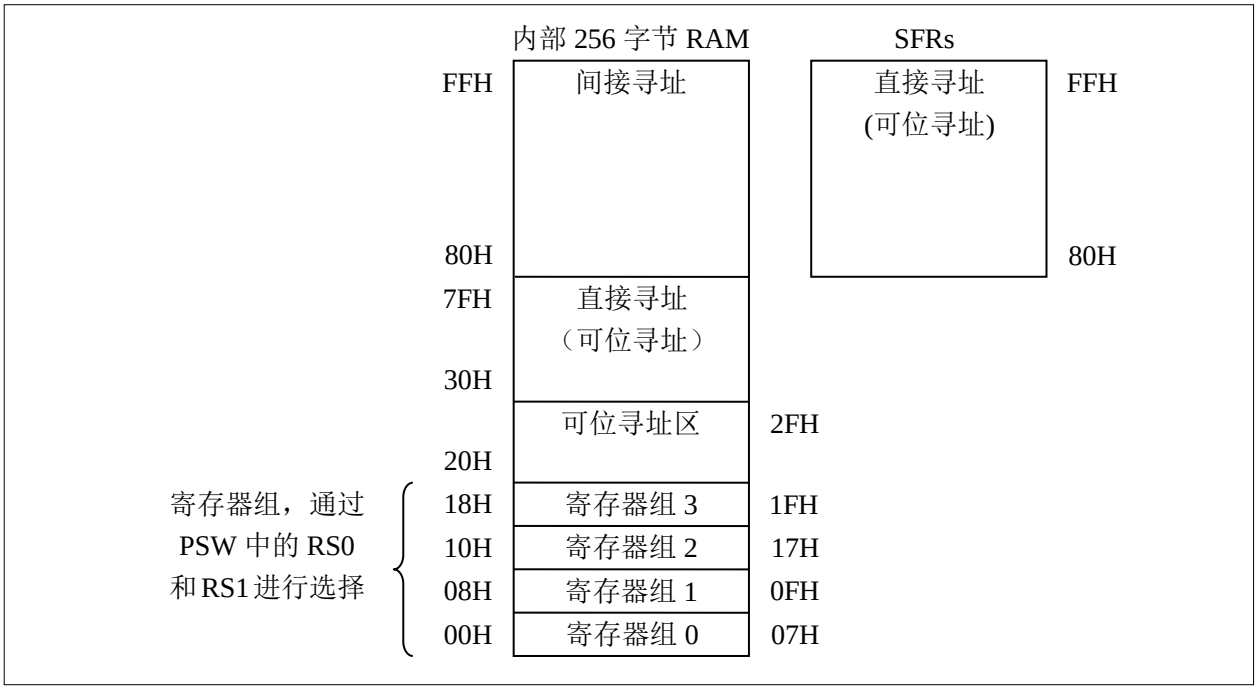
2、 当用户变量需求量接近或超过（单片机 edata 容量-1K）的大小时，建议将超出部分使用“xdata”关键字强制分配到 xdata 区域，如下所示：

```
int  xdata  pBuffer = 5;        //使用 xdata 关键字强制分配到 xdata 区域
```

12.2.2　内部 edata-RAM（C 语言声明关键字为 edata）

内部可 edata-RAM 共 2K 字节，2K 字节低端的 256 字节与 8051 的 256 字节 DATA 完全兼容，可分为 2 个部分：低 128 字节 RAM 和高 128 字节 RAM。低 128 字节的数据存储器与传统 8051 兼容，既可直接寻址也可间接寻址。高 128 字节 RAM（在 8052 中扩展了高 128 字节 RAM）只能间接寻址。特殊功能寄存器分布在 80H~FFH 区域，只可直接寻址。

低端 256 字节 RAM 的结构如下图所示：



Ai8051U 的堆栈放在 EDATA，设计上理论深度可达 64K，实际放了 2K Bytes；Ai8051U 的普通扩展 XDATA，设计上理论深度可达（8M-64K），实际放了 32K Bytes。所以 Ai8051U 的 SRAM 总共是 34K（2K edata + 32K xdata）。

在 C 语言代码中将变量声明在 EDATA 区域，即可实现单时钟进行 32 位/16 位/8 位的读写操作

```
char    edata   bCounter;      //在 EDATA 区域声明字节变量（单时钟进行 8 位读写操作）
int     edata   wCounter;      //在 EDATA 区域声明双字节变量（单时钟进行 16 位读写操作）
long    edata   dwCounter;     //在 EDATA 区域声明 4 字节变量（单时钟进行 32 位读写操作）
```

在 C 语言代码中将变量声明在 XDATA 区域，即可实现 8 位/16 位的读写操作

```
char    xdata   bCounter;      //在 XDATA 区域声明字节变量（3/2 个时钟进行 8 位读/写操作）
int     xdata   wCounter;      //在 XDATA 区域声明双字节变量（3/2 个时钟进行 16 位读/写操作）
```

12.2.3 程序状态寄存器（PSW）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PSW	D0H	CY	AC	F0	**RS1**	**RS0**	OV	-	P

CY：进位标志

AC：辅助进位标志

F0：通用标志

OV：溢出标志

P：ACC 的偶校验标志

RS1，RS0：工作寄存器选择位

RS1	RS0	工作寄存器组（R0~R7）
0	0	第 0 组（00H~07H）
0	1	第 1 组（08H~0FH）
1	0	第 2 组（10H~17H）
1	1	第 3 组（18H~1FH）

可位寻址区的地址从 20H ~ 2FH 共 16 个字节单元。20H~2FH 单元既可像普通 RAM 单元一样按字节存取，也可以对单元中的任何一位单独存取，共 128 位，所对应的逻辑位地址范围是 00H~7FH。位地址范围是 00H~7FH，内部 RAM 低 128 字节的地址也是 00H~7FH，从外表看，二者地址是一样的，实际上二者具有本质的区别；位地址指向的是一个位，而字节地址指向的是一个字节单元，在程序中使用不同的指令区分。

12.2.4 程序状态寄存器 1（PSW1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PSW1	D1H	CY	AC	N	RS1	RS0	OV	Z	-

CY：进位标志（与 PSW 中的 CY 相同）

AC：辅助进位标志（与 PSW 中的 AC 相同）

N：计算结果负标志位，运算结果的最高位为 1 时 N 为 1，否则为 0

RS1，RS0：工作寄存器选择位（与 PSW 中的 RS0、RS1 相同）

OV：溢出标志（与 PSW 中的 OV 相同）

Z：计算结果零标志位，运算结果为 0 时 Z 为 1，否则为 0

12.2.5 内部 xdata-RAM（C 语言声明关键字为 xdata）

Ai8051U 系列单片机片内部扩展了 XRAM，不是用户实际在片外扩展的 XRAM）。

访问内部扩展的 XRAM 的方法和传统 8051 单片机访问外部扩展 RAM 的方法相同，但是不影响 P0 口、P2 口、以及 RD、WR 和 ALE 等端口上的信号。

在汇编语言中，内部扩展 RAM 通过 MOVX 指令访问，

```
MOVX    A , @DPTR
MOVX    @DPTR , A
MOVX    A , @Ri
MOVX    @Ri , A
```

在 C 语言中，可使用 xdata 关键字声明存储类型即可。**（强烈建议不要使用 pdata 关键字声明变量）**

12.2.6 辅助寄存器（AUXR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	**EXTRAM**	S1BRT

EXTRAM：片外用户实际扩展的 XRAM 访问控制

0：不能访问用户实际在外部用并行总线扩展的 XRAM 或外设，这部分用户可以实际在外部并行扩展的 XRAM 或外设地址范围在：特殊的地址 7F:0000H~7F:FFFFH

1：可以访问用户实际在外部用并行总线扩展的 XRAM 或外设，这部分用户可以实际在外部并行扩展的 XRAM 或外设地址范围在：特殊的地址 7F:0000H~7F:FFFFH。当 EXTRAM 设置为 1 后，必须目标地址在 7F:0000H - 7F:FFFFH 范围之内，P0/P2/ALE/WR/RD 才会送出控制信号。

注：Ai8051U 系列 MCU 芯片内部的扩展 RAM/XRAM/xdata 的访问不受任何其他特殊功能寄存器的影响控制，用户随时都可以进行读写。片内的 XRAM 的访问不受任何 SFR 限制。

12.2.7 片外用户自己扩展的外部 XRAM/xdata

Ai8051U 系列单片机具有片外扩展 64KB 外部数据存储器 XRAM 或外设的能力（用户实际在外部用并行总线扩展的 XRAM 或外设），映射到内部的逻辑地址范围为 7F:0000H~7F:FFFFH。访问片外的外部数据存储器或外设期间，P0/P2/WR/RD/ALE 信号才有效。Ai8051U 系列单片机控制片外扩展的外部 64K 字节数据存储器或外设的总线速度的特殊功能寄存器 BUS_SPEED 说明如下：

12.2.8　片外用户扩展 RAM 的总线速度控制寄存器（BUS_SPEED）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
BUS_SPEED	A1H	RW_S[1:0]					SPEED[2:0]		

RW_S[1:0]：RD/WR 控制线选择位

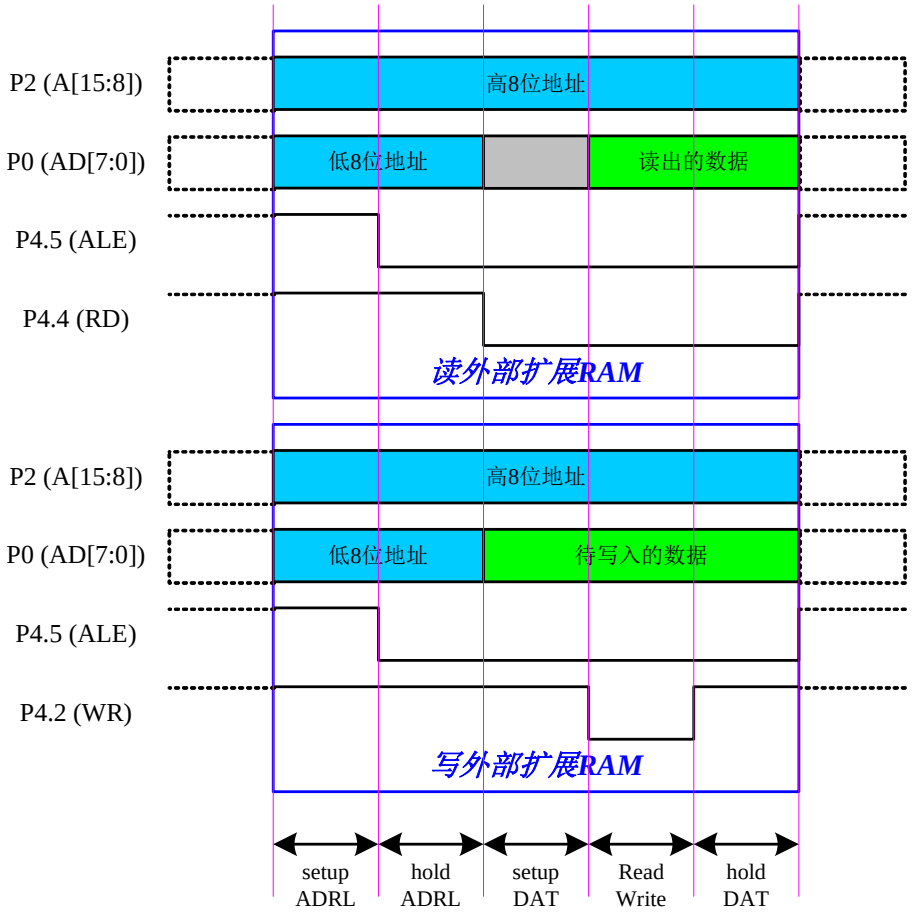
00：P4.4 为 RD，P4.2 为 WR

01：P3.7 为 RD，P3.6 为 WR

1x：保留

SPEED[2:0]：总线读写速度控制（读写数据时控制信号和数据信号的准备时间和保持时间）

读写外部扩展 RAM 时序如下图所示：



12.2.9 片内 MCU 扩展 RAM 数据总线时钟控制寄存器（CKCON）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CKCON	EAH	-	-	-	T1M	T0M	CKCON[2:0]		

T1M：保留

T0M：保留

CKCON[2:0]：外部数据总线时钟控制寄存器（上电复位值为 7，强烈建议上电初始化为 0）

CKCON[2:0]	等待时钟数
0	0 个时钟
1	1 个时钟
…	…
7	7 个时钟

12.2.10 片外扩展 RAM 地址总线扩展寄存器（MXAX）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
MXAX	EBH								

使用汇编指令 MOVX A,@Ri 和 MOVX @Ri,A 指令访问 XRAM 时，XRAM[23:16]地址由 MXAX 寄存器设置，XRAM[15:8]地址由 P2 口（P2 寄存器）设置，XRAM[7:0]地址由 Ri（R0/R1）设置。

只有在使用汇编指令 MOVX A,@Ri 和 MOVX @Ri,A 指令访问 XRAM，或者在 C 语言程序中使用 pdata 变量访问 XRAM 时，才需要使用 MXAX 寄存器（注：强烈不建议使用 pdata 定义变量）

12.2.11Ai8051U 系列单片机中可位寻址的数据存储器

Ai8051U 系列单片机内部可位寻址的数据存储器包括两部分：第一部分的地址范围为 DATA 区域的 20H~7FH，第二部分的地址范围是特殊功能寄存器 SFR：80H~FFH，下面对这两部分区域分别进行说明。

DATA 区域

DATA 区域的 00H~1FH 为寄存器 R0~R7 的映射区，不可位寻址，剩下 20H~7FH，共 96 个字节，每个字节均可位寻址。

汇编代码的定义方法：

```
BVAR1    BIT    20H.0        ;定义位变量 BVAR1
BVAR2    BIT    50H.1        ;定义位变量 BVAR2
BVAR3    BIT    70H.2        ;定义位变量 BVAR3
```

汇编代码的使用方法：

```
SETB     BVAR1                ;BVAR1 = 1
CLR      BVAR2                ;BVAR2 = 0
CPL      BVAR3                ;BVAR3 = ~BVAR3
```

C 语言代码的定义方法：

```
char  ebdata      flag;       //在可位寻址区域定义一个字节变量
sbit  bVar1   =   flag^0;     //使用 sbit 在 flag 中声明位变量 bVar1
sbit  bVar2   =   flag^7;     //使用 sbit 在 flag 中声明位变量 bVar2
bit   ebdata      bVar3;      //使用 bit ebdata 直接声明位变量 bVar3
```

C 语言代码的使用方法：

```
bVar1 = 1;                    //位变量置 1
bVar2 = 0;                    //位变量清 0
bVar3 = ~bVar3                //位变量取反
```

特殊功能寄存器（SFR）区域

全部 SFR 区域的 80H~FFH，共 128 个字节，每个字节均可位寻址。

汇编代码的定义方法：

```
BVAR1    BIT    80H.0        ;定义位变量 BVAR1
BVAR2    BIT    87H.1        ;定义位变量 BVAR2
BVAR3    BIT    FFH.2        ;定义位变量 BVAR3
```

汇编代码的使用方法：

```
SETB     BVAR1                ;BVAR1 = 1
CLR      BVAR2                ;BVAR2 = 0
CPL      BVAR3                ;BVAR3 = ~BVAR3
```

C 语言代码的定义方法：

```
sfr   P0      =   0x80;       //定义 SFR
sbit  P00     =   P0^0;       //使用 sbit 在 SFR 中声明 SFR 位
sfr   PCON    =   0x87;       //定义 SFR
sbit  PD      =   PCON^1;     //使用 sbit 在 SFR 中声明 SFR 位
sfr   ACC     =   0xE0;       //定义 SFR
sbit  ACC7    =   ACC^7;      //使用 sbit 在 SFR 中声明 SFR 位
```

C 语言代码的使用方法：

```
PD = 0;                       //位变量置 1
P00 = 1;                      //位变量清 0
```

```
ACC7 = ~ACC7                                    //位变量取反
```

注意：位变量不支持数组和指针类型的定义

12.2.12 扩展 SFR 使能寄存器 EAXFR 的使用说明

Ai8051U 系列单片机存储器地址的编址如下：

区域	地址范围	功能	说明
数据区	00:0000H — 00:FFFFH	数据区（edata）	堆栈区
	01:0000H — 01:FFFFH	内部扩展数据区（xdata）	
	02:0000H — 7D:FFFFH	数据保留区	
	7E:0000H — 7E:FFFFH	**扩展 SFR 区（XFR）**	**需使能 EAXFR（P_SW2.7）才能访问**
	7F:0000H — 7F:FFFFH	外部数据区（xdata）	需使能 EXTRAM（AUXR.1）才能访问
代码区	80:0000H — FD:FFFFH	代码保留区	
	FE:0000H — FE:FFFFH	扩展代码区（ecode）	
	FF:0000H — FF:FFFFH	代码区（code）	

如需访问 XFR 区域的扩展 SFR，需要先将 EAXFR（P_SW2.7）置 1，由于 Ai8051U 的存储器地址的编址方式是线性编址的，XFR 区域的地址是独立的一块地址区域，所以可以在**上电系统初始化时将 EAXFR 寄存器写 1（例如：EAXFR = 1;），后续一直保持为 1 不用再修改**，即可正常访问 XFR 区域。

注意：由于 EAXFR 控制位和 S2_S、S3_S 等位寄存器位共用 P_SW2，在设置 P_SW2 中的 S2_S、S3_S 时，建议直接使用位操作语句（例如 S2_S = 0; S3_S = 1;），避免直接操作 P_SW2 寄存器。如代码中确实需要直接对 P_SW2 进行修改，也一定要使用与或指令（例如 P_SW2 &= ~0x01; P_SW2 |= 0x02;）。

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW2	BAH	EAXFR	-	I2C_S[1:0]		CMPO_S	S4_S	S3_S	S2_S

EAXFR：扩展 RAM 区特殊功能寄存器（XFR）访问控制寄存器

0：禁止访问 XFR

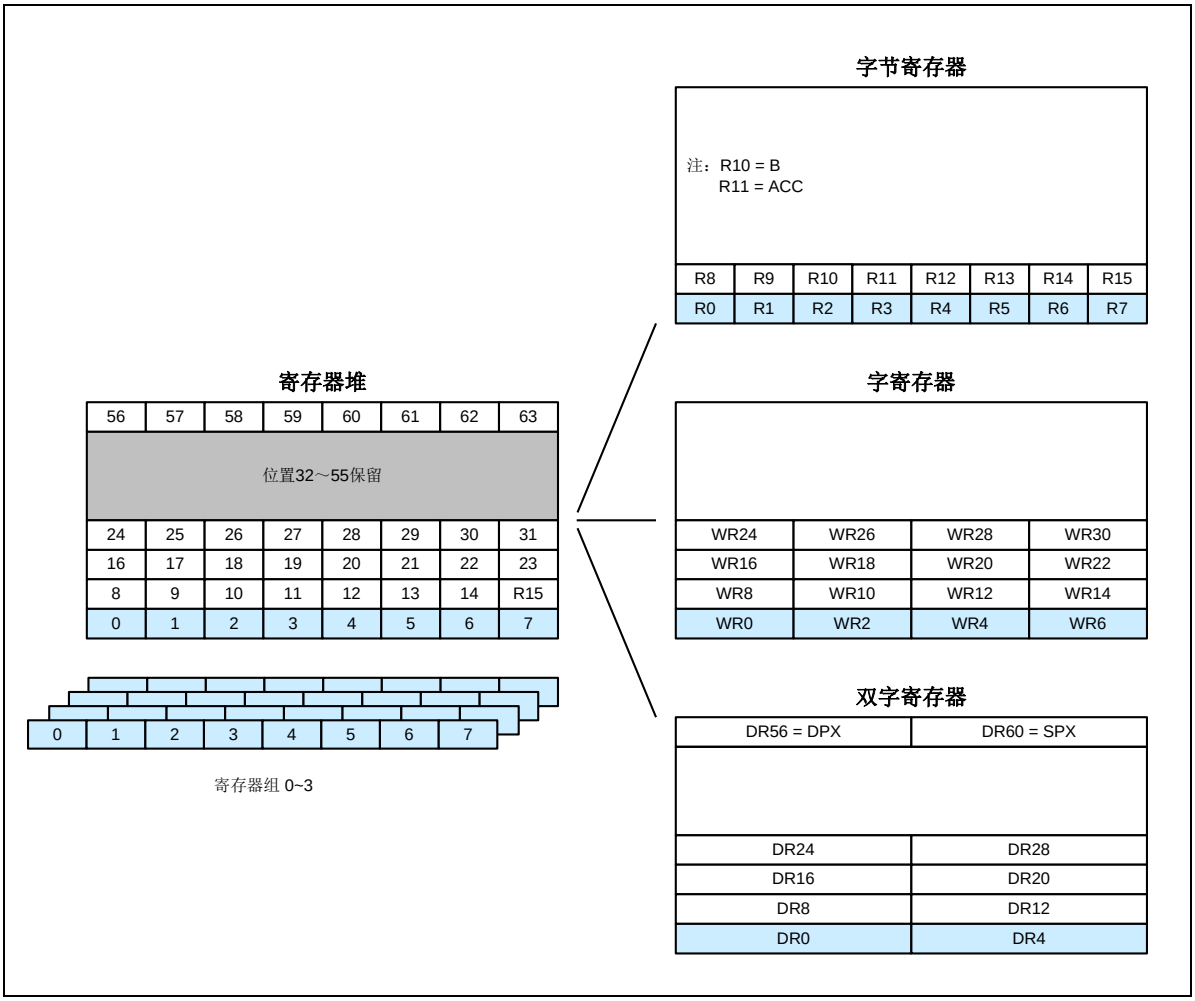
1：使能访问 XFR。

当需要访问 XFR 时，必须先将 EAXFR 置 1，才能对 XFR 进行正常的读写。建议上电初始化时直接设置为 1，后续不要再修改

12.3　寄存器堆

12.3.1 寄存器堆结构图

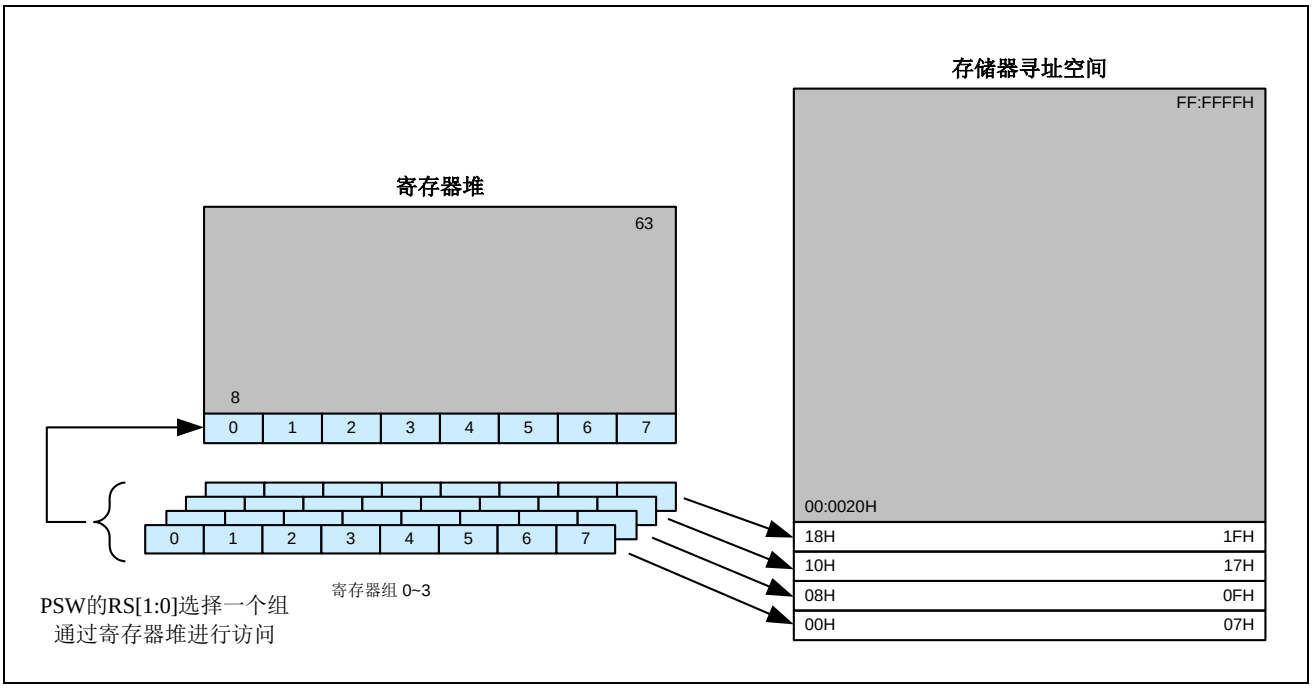
Ai8051U 系列单片机的寄存器堆由 40 个位置组成：0～31 以及 56～63，如下图所示。这些位置可作为字节、字和双字访问。



位置 0 到 7 的寄存器堆实际上由四个可切换组组成，每个组有 8 个寄存器，如下图所示。这四个组使用片上内存的前 32 个字节，并且始终可在存储地址空间的 00：0000H ～ 00:001FH 的位置访问。通过 PSW 寄存器的 RS1 和 RS0 位选择四个组中的一个进行访问。

位置 8 到 31 和 56 到 63 的寄存器组始终可以访问。这些位置在 CPU 中作为寄存器使用。这些位置在寄存器堆是专用的，所以它们不作为片上内存的一部分。

位置 32 到 55 的寄存器堆保留，不能访问。



寄存器组选择

组	地址范围	PSW 选择位	
		RS1	RS0
工作组 0	00H–07H	0	0
工作组 1	08H–0FH	0	1
工作组 2	10H–17H	1	0
工作组 3	18H–1FH	1	1

12.3.2 字节、字以及双字寄存器

根据在寄存器堆中的位置，寄存器可作为字节、字以及、或者双字寻址。寄存器以其最低编号的字节位置命名。例如：

R4 是由位置 4 组成的字节寄存器。

WR4 是由第 4 和第 5 寄存器组成的字寄存器。

DR4 是由第 4 到第 7 寄存器组成的双字寄存器。

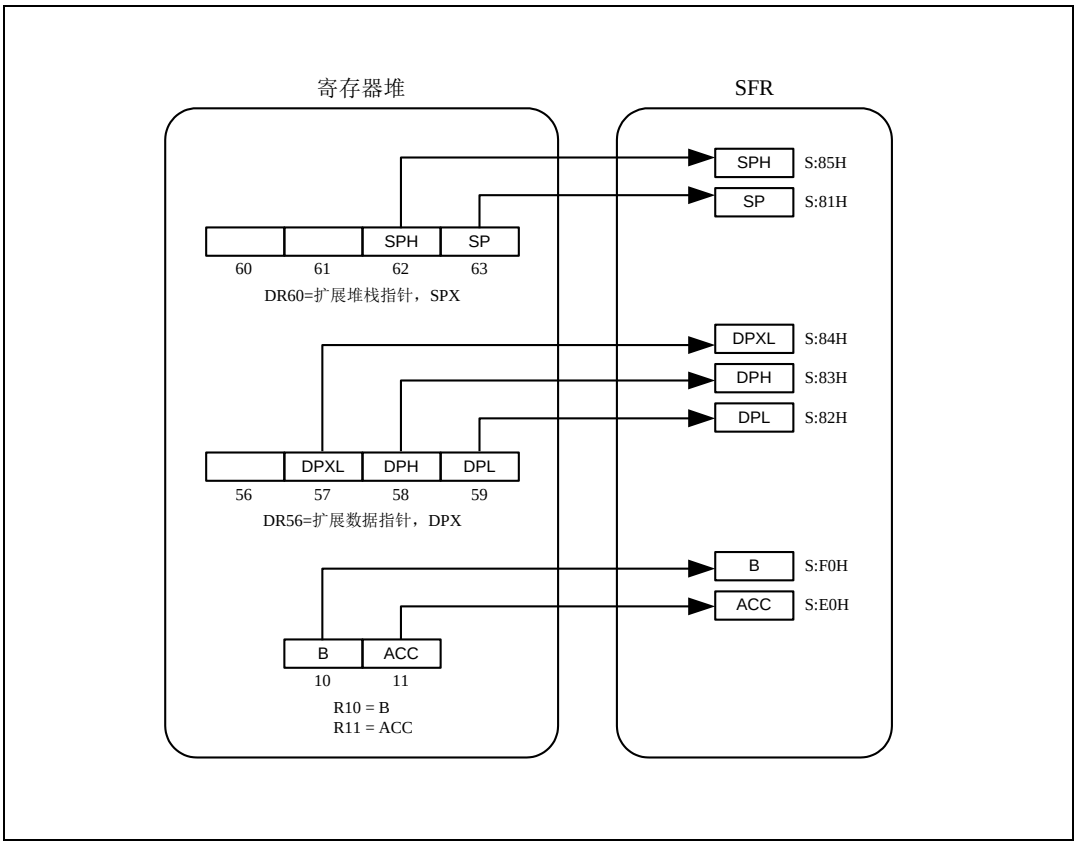
位置 R0～R15 可作为字节、字或者双字寻址。位置 16～31 只能作为字或者双字寻址。位置 56 到 63 只能作为双字寻址。

12.3.3 专用寄存器

寄存器堆有四个专用寄存器：

- R10 是寄存器 B
- R11 是累加器 ACC
- DR56 是扩展数据指针，DPX
- DR60 是扩展堆栈指针，SPX

这些寄存器位于寄存器堆中；然而 R10、R11 以及 DR56 和 DR60 的某些字节也可作为 SFR 访问。寄存器堆中的 DPX 和 SPX 字节只能通过双字寄存器寻址来访问。寄存器堆中的专用寄存器及其对应的 SFR 如下图所示



寄存器堆中的专用寄存器及其对应的 SFR

寄存器堆						SFR	
名称			助记符	寄存器	位置	助记符	地址
堆栈指针（SPX）	—		—	DR60	60	—	—
	—		—		61	—	—
	堆栈指针，高		SPH		62	SPH	S:85H
	堆栈指针，低		SP		63	SP	S:81H
数据指针（DPX）	—		—	DR56	56	—	—
	数据指针，扩展低		DPXL		57	DPXL	S:84H
	DPTR	数据指针，高	DPH		58	DPH	S:83H
		数据指针，低	DPL		59	DPL	S:82H
累加器（寄存器 A）			A	R11	11	ACC	S:E0H
寄存器 B			B	R10	10	B	S:F0H

12.3.4 扩展数据指针，DPX

双字寄存器 DR56 是扩展数据指针 DPX。DPX 的低三个字节（DPL、DPH 和 DPXL）可作为 SFR 访问。DPL 和 DPH 组成 16 位数据指针 DPTR。DPXL 的字节在位置 57 处。在用 MOVX 指令将数据搬入和搬出外部存储器时由 DPXL 寻址指定的区域。DPXL 的复位值为 01H。

12.3.5 扩展堆栈指针，SPX

双字寄存器 DR60 是堆栈指针 SPX。位置 63 的字节是 8 位堆栈指针 SP。位置 62 处的字节是堆栈指针高位 SPH。这两个字节允许堆栈扩展到内存 00 区域的顶部。SP 和 SPH 可以作为 SFR 访问。

PUSH 和 POP 两条指令直接寻址堆栈指针。子程序调用（ACALL、ECALL、LCALL）和返回（ERET、RET、RETI）也使用堆栈指针。请勿将 DR60 用作通用寄存器。

12.4 只读特殊功能寄存器中存储的唯一 ID 号和重要参数（CHIPID 和 CHIPIDX）

产品线	CHIPID
Ai8051U 系列	●

Ai8051U 系列单片机内部的只读特殊功能寄存器 CHIPID 中保存有与芯片相关的一些特殊参数，包括：全球唯一 ID 号、32K 掉电唤醒定时器的频率、内部 1.19V 参考信号源值（BGV）以及 IRC 参数。在用户程序中只能读取 CHIPID 中的内容，不可修改。使用 CHIPID 中的数据对用户程序进行加密是官方推荐的最优方案。

12.4.1 CHIPID 相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID0	硬件数字 ID00	7EFDE0H	全球唯一 ID 号（第 0 字节）								nnnn,nnnn
CHIPID1	硬件数字 ID01	7EFDE1H	全球唯一 ID 号（第 1 字节）								nnnn,nnnn
CHIPID2	硬件数字 ID02	7EFDE2H	全球唯一 ID 号（第 2 字节）								nnnn,nnnn
CHIPID3	硬件数字 ID03	7EFDE3H	全球唯一 ID 号（第 3 字节）								nnnn,nnnn
CHIPID4	硬件数字 ID04	7EFDE4H	全球唯一 ID 号（第 4 字节）								nnnn,nnnn
CHIPID5	硬件数字 ID05	7EFDE5H	全球唯一 ID 号（第 5 字节）								nnnn,nnnn
CHIPID6	硬件数字 ID06	7EFDE6H	全球唯一 ID 号（第 6 字节）								nnnn,nnnn
CHIPID7	硬件数字 ID07	7EFDE7H	内部 1.19V 参考信号源-BGV（高字节）								nnnn,nnnn
CHIPID8	硬件数字 ID08	7EFDE8H	内部 1.19V 参考信号源-BGV（低字节）								nnnn,nnnn
CHIPID9	硬件数字 ID09	7EFDE9H	32K 掉电唤醒定时器的频率（高字节）								nnnn,nnnn
CHIPID10	硬件数字 ID10	7EFDEAH	32K 掉电唤醒定时器的频率（低字节）								nnnn,nnnn
CHIPID11	硬件数字 ID11	7EFDEBH	22.1184MHz 的 IRC 参数（27M 频段）								nnnn,nnnn
CHIPID12	硬件数字 ID12	7EFDECH	24MHz 的 IRC 参数（27M 频段）								nnnn,nnnn
CHIPID13	硬件数字 ID13	7EFDEDH	27MHz 的 IRC 参数（27M 频段）								nnnn,nnnn
CHIPID14	硬件数字 ID14	7EFDEEH	30MHz 的 IRC 参数（27M 频段）								nnnn,nnnn
CHIPID15	硬件数字 ID15	7EFDEFH	33.1776MHz 的 IRC 参数（27M 频段）								nnnn,nnnn
CHIPID16	硬件数字 ID16	7EFDF0H	35MHz 的 IRC 参数（44M 频段）								nnnn,nnnn
CHIPID17	硬件数字 ID17	7EFDF1H	36.864MHz 的 IRC 参数（44M 频段）								nnnn,nnnn
CHIPID18	硬件数字 ID18	7EFDF2H	40MHz 的 IRC 参数（44M 频段）								nnnn,nnnn
CHIPID19	硬件数字 ID19	7EFDF3H	44.2368MHz 的 IRC 参数（44M 频段）								nnnn,nnnn
CHIPID20	硬件数字 ID20	7EFDF4H	45.1584MHz 的 IRC 参数（44M 频段）								nnnn,nnnn
CHIPID21	硬件数字 ID21	7EFDF5H	6M 频段的 VRTRIM 参数								nnnn,nnnn
CHIPID22	硬件数字 ID22	7EFDF6H	10M 频段的 VRTRIM 参数								nnnn,nnnn
CHIPID23	硬件数字 ID23	7EFDF7H	27M 频段的 VRTRIM 参数								nnnn,nnnn
CHIPID24	硬件数字 ID24	7EFDF8H	44M 频段的 VRTRIM 参数								nnnn,nnnn
CHIPID25	硬件数字 ID25	7EFDF9H	00H[2]								nnnn,nnnn
CHIPID26	硬件数字 ID26	7EFDFAH	用户程序空间结束地址（高字节）								nnnn,nnnn

CHIPID27	硬件数字 ID27	7EFDFBH	芯片测试时间（年）	nnnn,nnnn
CHIPID28	硬件数字 ID28	7EFDFCH	芯片测试时间（月）	nnnn,nnnn
CHIPID29	硬件数字 ID29	7EFDFDH	芯片测试时间（日）	nnnn,nnnn
CHIPID30	硬件数字 ID30	7EFDFEH	芯片封装形式编号	nnnn,nnnn
CHIPID31	硬件数字 ID31	7EFDFFH	5AH/5BH[1]	nnnn,nnnn

备注[1]：当 CHIPID31 为 5AH 时，CHIPID25 为保留值 00H；当 CHIPID31 为 5BH 时，CHIPID25 为内置 CHIPID13～CHIPID20 这 8 个 IRC 频率的频段的 bitmap 值，详情参考备注[2]

备注[2]：当 CHIPID25 的值为频段的 bitmap 值时，对应关系如下：

CHIPID25	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
频段	B_{CID20_45M}	B_{CID19_44M}	B_{CID18_40M}	B_{CID17_36M}	B_{CID16_35M}	B_{CID15_33M}	B_{CID14_30M}	B_{CID13_27M}

B_{CID20_45M}：45.1584MHz 频率的频段，0：27M 频段；1：44M 频段

B_{CID19_44M}：44.2368MHz 频率的频段，0：27M 频段；1：44M 频段

B_{CID18_40M}：40MHz 频率的频段，0：27M 频段；1：44M 频段

B_{CID17_36M}：36.864MHz 频率的频段，0：27M 频段；1：44M 频段

B_{CID16_35M}：35MHz 频率的频段，0：27M 频段；1：44M 频段

B_{CID15_33M}：33.1776MHz 频率的频段，0：27M 频段；1：44M 频段

B_{CID14_30M}：30MHz 频率的频段，0：27M 频段；1：44M 频段

B_{CID13_27M}：27MHz 频率的频段，0：27M 频段；1：44M 频段

12.4.2　CHIPIDX 相关寄存器

固件版本等于或高于 7.4.10 的 Ai8051U 系列芯片才有 CHIPIDX 功能

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPIDX0	扩展硬件数字 ID00	7EFBD0H	33.8688MHz 的 IRC 参数								nnnn,nnnn
CHIPIDX1	扩展硬件数字 ID01	7EFBD1H	33.8688MHz 的频段参数								nnnn,nnnn
CHIPIDX2	扩展硬件数字 ID02	7EFBD2H	40.96MHz 的 IRC 参数								nnnn,nnnn
CHIPIDX3	扩展硬件数字 ID03	7EFBD3H	40.96MHz 的频段参数								nnnn,nnnn
CHIPIDX4	扩展硬件数字 ID04	7EFBD4H	42MHz 的 IRC 参数								nnnn,nnnn
CHIPIDX5	扩展硬件数字 ID05	7EFBD5H	42MHz 的频段参数								nnnn,nnnn
CHIPIDX6	扩展硬件数字 ID06	7EFBD6H	保留								nnnn,nnnn
CHIPIDX7	扩展硬件数字 ID07	7EFBD7H	保留								nnnn,nnnn
CHIPIDX8	扩展硬件数字 ID08	7EFBD8H	保留								nnnn,nnnn
CHIPIDX9	扩展硬件数字 ID09	7EFBD9H	保留								nnnn,nnnn
CHIPIDX10	扩展硬件数字 ID10	7EFBDAH	保留								nnnn,nnnn
CHIPIDX11	扩展硬件数字 ID11	7EFBDBH	保留								nnnn,nnnn
CHIPIDX12	扩展硬件数字 ID12	7EFBDCH	保留								nnnn,nnnn
CHIPIDX13	扩展硬件数字 ID13	7EFBDDH	保留								nnnn,nnnn
CHIPIDX14	扩展硬件数字 ID14	7EFBDEH	保留								nnnn,nnnn
CHIPIDX15	扩展硬件数字 ID15	7EFBDFH	保留								nnnn,nnnn
CHIPIDX16	扩展硬件数字 ID16	7EFBE0H	保留								nnnn,nnnn
CHIPIDX17	扩展硬件数字 ID17	7EFBE1H	保留								nnnn,nnnn
CHIPIDX18	扩展硬件数字 ID18	7EFBE2H	保留								nnnn,nnnn

CHIPIDX19	扩展硬件数字 ID19	7EFBE3H	保留	nnnn,nnnn
CHIPIDX20	扩展硬件数字 ID20	7EFBE4H	保留	nnnn,nnnn
CHIPIDX21	扩展硬件数字 ID21	7EFBE5H	保留	nnnn,nnnn
CHIPIDX22	扩展硬件数字 ID22	7EFBE6H	保留	nnnn,nnnn
CHIPIDX23	扩展硬件数字 ID23	7EFBE7H	保留	nnnn,nnnn
CHIPIDX24	扩展硬件数字 ID24	7EFBE8H	保留	nnnn,nnnn
CHIPIDX25	扩展硬件数字 ID25	7EFBE9H	保留	nnnn,nnnn
CHIPIDX26	扩展硬件数字 ID26	7EFBEAH	保留	nnnn,nnnn
CHIPIDX27	扩展硬件数字 ID27	7EFBEBH	保留	nnnn,nnnn
CHIPIDX28	扩展硬件数字 ID28	7EFBECH	保留	nnnn,nnnn
CHIPIDX29	扩展硬件数字 ID29	7EFBEDH	保留	nnnn,nnnn
CHIPIDX30	扩展硬件数字 ID30	7EFBEEH	保留	nnnn,nnnn
CHIPIDX31	扩展硬件数字 ID31	7EFBEFH	5AH	nnnn,nnnn

12.4.3 CHIPID 之全球唯一 ID 号解读

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID0	硬件数字 ID00	7EFDE0H	全球唯一 ID 号（第 0 字节）								nnnn,nnnn
CHIPID1	硬件数字 ID01	7EFDE1H	全球唯一 ID 号（第 1 字节）								nnnn,nnnn
CHIPID2	硬件数字 ID02	7EFDE2H	全球唯一 ID 号（第 2 字节）								nnnn,nnnn
CHIPID3	硬件数字 ID03	7EFDE3H	全球唯一 ID 号（第 3 字节）								nnnn,nnnn
CHIPID4	硬件数字 ID04	7EFDE4H	全球唯一 ID 号（第 4 字节）								nnnn,nnnn
CHIPID5	硬件数字 ID05	7EFDE5H	全球唯一 ID 号（第 5 字节）								nnnn,nnnn
CHIPID6	硬件数字 ID06	7EFDE6H	全球唯一 ID 号（第 6 字节）								nnnn,nnnn

[CHIPID0, CHIPID1]：16 位 MCU ID，用于区别不同的单片机型号（高位在前）。

[CHIPID2, CHIPID3]：16 位测试机台编号（高位在前）。

[CHIPID4, CHIPID5, CHIPID6]：24 位测试流水编号（高位在前）。

12.4.4 CHIPID 之内部参考信号源解读

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID7	硬件数字 ID07	7EFDE7H	内部 1.19V 参考信号源-BGV（高字节）								nnnn,nnnn
CHIPID8	硬件数字 ID08	7EFDE8H	内部 1.19V 参考信号源-BGV（低字节）								nnnn,nnnn

[CHIPID7, CHIPID8]：16 位内部参考信号源电压值（高位在前）。

标准值为 1190（04A6H），单位为 mV，即 1.19V。但实际的芯片由于存在制造误差。内部参考信号源的电压值并不会受工作电压 VCC 的影响，所以内部参考信号源可以和 ADC 结合用于校准 ADC，也可和比较器结合用于侦测工作电压。

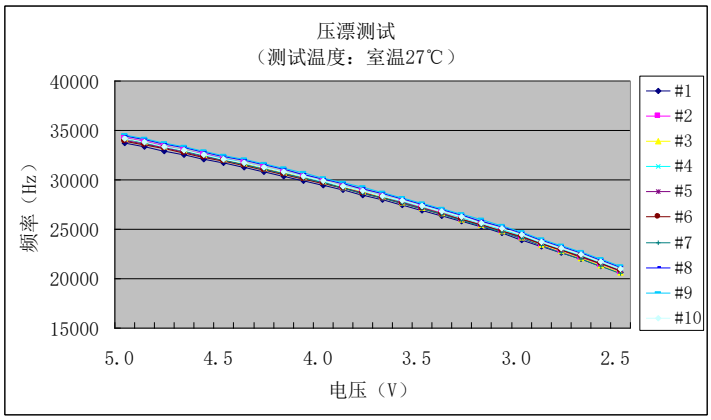
12.4.5 CHIPID 之内部 32K 的 IRC 振荡频率解读

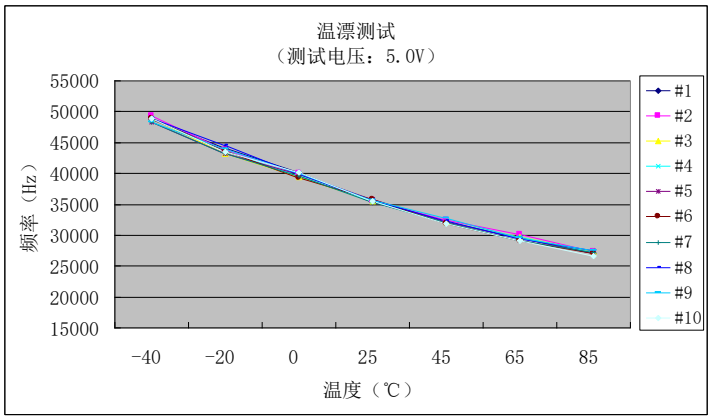
符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID9	硬件数字 ID09	7EFDE9H	32K 掉电唤醒定时器的频率（高字节）								nnnn,nnnn
CHIPID10	硬件数字 ID10	7EFDEAH	32K 掉电唤醒定时器的频率（低字节）								nnnn,nnnn

[CHIPID9, CHIPID10]：16 位 32K IRC 振荡器频率值（高位在前）。

标准值为 32768（8000H），单位为 Hz，即 32.768KHz。但实际的芯片由于存在制造误差，而且温漂和压漂均比较大。

内部 32K 振荡器的压漂测试线性图和温漂线性图如下：


压漂测试
（测试温度：室温27℃）
40000
35000
30000
25000
20000
15000
频率（Hz）
5.0
4.5
4.0
3.5
3.0
2.5
电压（V）
#1
#2
#3
#4
#5
#6
#7
#8
#9
#10


温漂测试
（测试电压：5.0V）
55000
50000
45000
40000
35000
30000
25000
20000
15000
频率（Hz）
-40
-20
0
25
45
65
85
温度（℃）
#1
#2
#3
#4
#5
#6
#7
#8
#9
#10

12.4.6 CHIPID 之高精度 IRC 参数解读

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID11	硬件数字 ID11	7EFDEBH	22.1184MHz 的 IRC 参数（27M 频段）								nnnn,nnnn
CHIPID12	硬件数字 ID12	7EFDECH	24MHz 的 IRC 参数（27M 频段）								nnnn,nnnn
CHIPID13	硬件数字 ID13	7EFDEDH	27MHz 的 IRC 参数（27M 频段）								nnnn,nnnn
CHIPID14	硬件数字 ID14	7EFDEEH	30MHz 的 IRC 参数（27M 频段）								nnnn,nnnn
CHIPID15	硬件数字 ID15	7EFDEFH	33.1776MHz 的 IRC 参数（27M 频段）								nnnn,nnnn
CHIPID16	硬件数字 ID16	7EFDF0H	35MHz 的 IRC 参数（44M 频段）								nnnn,nnnn
CHIPID17	硬件数字 ID17	7EFDF1H	36.864MHz 的 IRC 参数（44M 频段）								nnnn,nnnn
CHIPID18	硬件数字 ID18	7EFDF2H	40MHz 的 IRC 参数（44M 频段）								nnnn,nnnn
CHIPID19	硬件数字 ID19	7EFDF3H	44.2368MHz 的 IRC 参数（44M 频段）								nnnn,nnnn
CHIPID20	硬件数字 ID20	7EFDF4H	45.1584MHz 的 IRC 参数（44M 频段）								nnnn,nnnn
CHIPID21	硬件数字 ID21	7EFDF5H	6M 频段的 VRTRIM 参数								nnnn,nnnn
CHIPID22	硬件数字 ID22	7EFDF6H	10M 频段的 VRTRIM 参数								nnnn,nnnn
CHIPID23	硬件数字 ID23	7EFDF7H	27M 频段的 VRTRIM 参数								nnnn,nnnn
CHIPID24	硬件数字 ID24	7EFDF8H	44M 频段的 VRTRIM 参数								nnnn,nnnn

支持 CHIPID 功能的 Ai8051U 系列单片机，内部集成的高精度 IRC 分 4 个频段，每个频段对应的参考电压值在出厂时已进行了校准，当选择不同的频段时，只需要将相应频段的电压校准值填入 VRTRIM 寄存器即可。4 个频段的中心频率分别为 6MHz、10MHz、27MHz 和 44MHz，由于制造误差，中心频率一般可能有±5%的偏差，为了得到精确的用户频率，可使用 IRTRIM 对频率进行微调校准。使用官方提供的下载软件下载用户程序时，系统会根据用户所设定频率自动设置 VRTRIM 和 IRTRIM 寄存器。同时，在 CHIPID 也内部预置了 10 个常用频率的 IRTRIM 值以及 4 个频段的参考电压校准值，让用户可以在程序运行过程中动态的修改工作频率。

[CHIPID11 : CHIPID20]：10 个常用频率的 IRTRIM 值。括号里面的注解即为对应的频段

[CHIPID21 : CHIPID24]：4 个频段的参考电压值校准值。

用户动态修改频率时，只需要将[CHIPID11 : CHIPID20]中的某个频率校准值读出并写入 IRTRIM 寄存器，同时根据该频率所对应的频段将[CHIPID21 ： CHIPID24]中的某个电压校准值读出并写入 VRTRIM 寄存器即可。详细操作请参考后续章节的范例程序。

当 CHIPID31 为 5AH 时，CHIPID25 为保留值 00H；当 CHIPID31 为 5BH 时，CHIPID25 为内置 CHIPID13～CHIPID20 这 8 个 IRC 频率的频段的 bitmap 值，对应关系如下：

CHIPID25	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
频段	B_{CID20_45M}	B_{CID19_44M}	B_{CID18_40M}	B_{CID17_36M}	B_{CID16_35M}	B_{CID15_33M}	B_{CID14_30M}	B_{CID13_27M}

B_{CID20_45M}：45.1584MHz 频率的频段，0：27M 频段；1：44M 频段

B_{CID19_44M}：44.2368MHz 频率的频段，0：27M 频段；1：44M 频段

B_{CID18_40M}：40MHz 频率的频段，0：27M 频段；1：44M 频段

B_{CID17_36M}：36.864MHz 频率的频段，0：27M 频段；1：44M 频段

B_{CID16_35M}：35MHz 频率的频段，0：27M 频段；1：44M 频段

B_{CID15_33M}：33.1776MHz 频率的频段，0：27M 频段；1：44M 频段

B_{CID14_30M}：30MHz 频率的频段，0：27M 频段；1：44M 频段

B_{CID13_27M}：27MHz 频率的频段，0：27M 频段；1：44M 频段

12.4.7　CHIPIDX 之高精度 IRC 参数解读

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPIDX0	扩展硬件数字 ID00	7EFBD0H	33.8688MHz 的 IRC 参数								nnnn,nnnn
CHIPIDX1	扩展硬件数字 ID01	7EFBD1H	33.8688MHz 的频段参数								nnnn,nnnn
CHIPIDX2	扩展硬件数字 ID02	7EFBD2H	40.96MHz 的 IRC 参数								nnnn,nnnn
CHIPIDX3	扩展硬件数字 ID03	7EFBD3H	40.96MHz 的频段参数								nnnn,nnnn
CHIPIDX4	扩展硬件数字 ID04	7EFBD4H	42MHz 的 IRC 参数								nnnn,nnnn
CHIPIDX5	扩展硬件数字 ID05	7EFBD5H	42MHz 的频段参数								nnnn,nnnn

注：固件版本等于或高于 7.4.10 的 Ai8051U 系列芯片才有 CHIPIDX 功能

目前的 7.4.10 固件版本，在 CHIPIDX 中记录了 3 组 IRC 参数，每组参数包括两项内容：IRC 参数和频段参数，IRC 参数是该频率所对应的 IRTRIM 的校准值，频段参数是该频率所在的频段值。

例如：软件加载 33.8688MHz 的 IRC 参数的示例代码如下：

```
CLKDIV = 0x04;                          //先将系统频率降低，防止程序跑飞
IRTRIM = CHIPIDX0;                      //加载 IRC 校准参数到 IRTRIM
VRTRIM = CHIPID[21 + CHIPIDX1];         //根据 IRC 频段从 CHIPID 中加载 VR 值到 VRTRIM
IRCBAND = (IRCBAND & ~0x03) | CHIPIDX1; //配置 IRC 频段寄存器
CLKDIV = 0x01;                          //恢复系统除频寄存器
```

12.4.8 CHIPID 之测试时间参数解读

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID27	硬件数字 ID27	7EFDFBH	芯片测试时间（年）								nnnn,nnnn
CHIPID28	硬件数字 ID28	7EFDFCH	芯片测试时间（月）								nnnn,nnnn
CHIPID29	硬件数字 ID29	7EFDFDH	芯片测试时间（日）								nnnn,nnnn

测试时间的年、月、日参数均为 BCD 码。（例如：CHIPID27＝0x21，CHIPID28=0x11，CHIPID29=0x18，则目标芯片的生产测试日期为 2021 年 11 月 18 日）

12.4.9 CHIPID 之芯片封装形式编号解读

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID30	硬件数字 ID30	7EFDFEH	芯片封装形式编号								nnnn,nnnn

封装编号	封装形式		封装编号	封装形式
0x00	DIP8		0x50	SOP32
0x01	SOP8		0x51	LQFP32
0x02	DFN8		0x52	QFN32
0x10	DIP16		0x53	PLCC32
0x11	SOP16		0x54	QFN32S
0x20	DIP18		0x60	PDIP40
0x21	SOP18		0x70	LQFP44
0x30	DIP20		0x71	PLCC44
0x31	SOP20		0x72	PQFP44
0x32	TSSOP20		0x80	LQFP48
0x33	LSSOP20		0x81	QFN48
0x34	QFN20		0x90	LQFP64
0x40	SKDIP28		0x91	LQFP64S
0x41	SOP28		0x92	LQFP64L
0x42	TSSOP28		0x93	LQFP64M
0x43	QFN28		0x94	QFN64

12.5　范例程序

12.5.1　读取内部 1.19V 参考信号源值（BGV）

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                  //头文件见下载软件
#include "intrins.h"                                  //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   FOSC          11059200UL                    //定义为无符号长整型,避免计算溢出
#define   BRT           (65536 - (FOSC / 115200+2) / 4)
                                                      //加 2 操作是为了让 Keil 编译器
                                                      //自动实现四舍五入运算

#define   VREFH_ADDR    CHIPID7
#define   VREFL_ADDR    CHIPID8

bit       busy;

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    T1x12 = 1;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void main()
{
    EAXFR = 1;                                        //允许访问扩展的特殊寄存器，XFR
                                                      //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                    //允许访问扩展的特殊寄存器，XFR
                                                      //(8 位模式请使用这句，注释上一句)
```

```
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    UartInit();
    ES = 1;
    EA = 1;
    UartSend(VREF_ADDRH);                       //读取内部 1.19V 参考信号源的高字节
    UartSend(VREF_ADDRL);                       //读取内部 1.19V 参考信号源的低字节

    while (1);
}
```

12.5.2　读取全球唯一 ID 号

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                           //头文件见下载软件
#include "intrins.h"                           //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   FOSC          11059200UL            //定义为无符号长整型,避免计算溢出
#define   BRT           (65536 - (FOSC / 115200+2) / 4)
                                              //加 2 操作是为了让 Keil 编译器
                                              //自动实现四舍五入运算

#define   ID_ADDR           (&CHIPID0)

bit       busy;

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
    }
```

```
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    T1x12 = 1;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void main()
{
    char i;

    EAXFR = 1;                  //允许访问扩展的特殊寄存器，XFR
                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;              //允许访问扩展的特殊寄存器，XFR
                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                   //设置取程序代码等待时间，
                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                  //设置访问片内的 xdata 速度，
                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    UartInit();
    ES = 1;
    EA = 1;

    for (i=0; i<7; i++)
    {
        UartSend(ID_ADDR[i]);
    }

    while (1);
}
```

12.5.3　读取 32K 掉电唤醒定时器的频率

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                  //头文件见下载软件
#include "intrins.h"                                  //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   FOSC          11059200UL                    //定义为无符号长整型,避免计算溢出
#define   BRT           (65536 - (FOSC / 115200+2) / 4)
                                                      //加 2 操作是为了让 Keil 编译器
                                                      //自动实现四舍五入运算

#define   F32K_ADDRH    CHIPID9
#define   F32K_ADDRL    CHIPID10

bit       busy;

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    T1x12 = 1;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void main()
{
    EAXFR = 1;                                        //允许访问扩展的特殊寄存器，XFR
                                                      //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                    //允许访问扩展的特殊寄存器，XFR
                                                      //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                         //设置取程序代码等待时间，
                                                      //赋值为 0 表示不等待，程序以最快速度运行
```

```
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    UartInit();
    ES = 1;
    EA = 1;

    UartSend(F32K_ADDRH);                       //读取 32K 频率的高字节
    UartSend(F32K_ADDRL);                       //读取 32K 频率的低字节

    while (1);
}
```

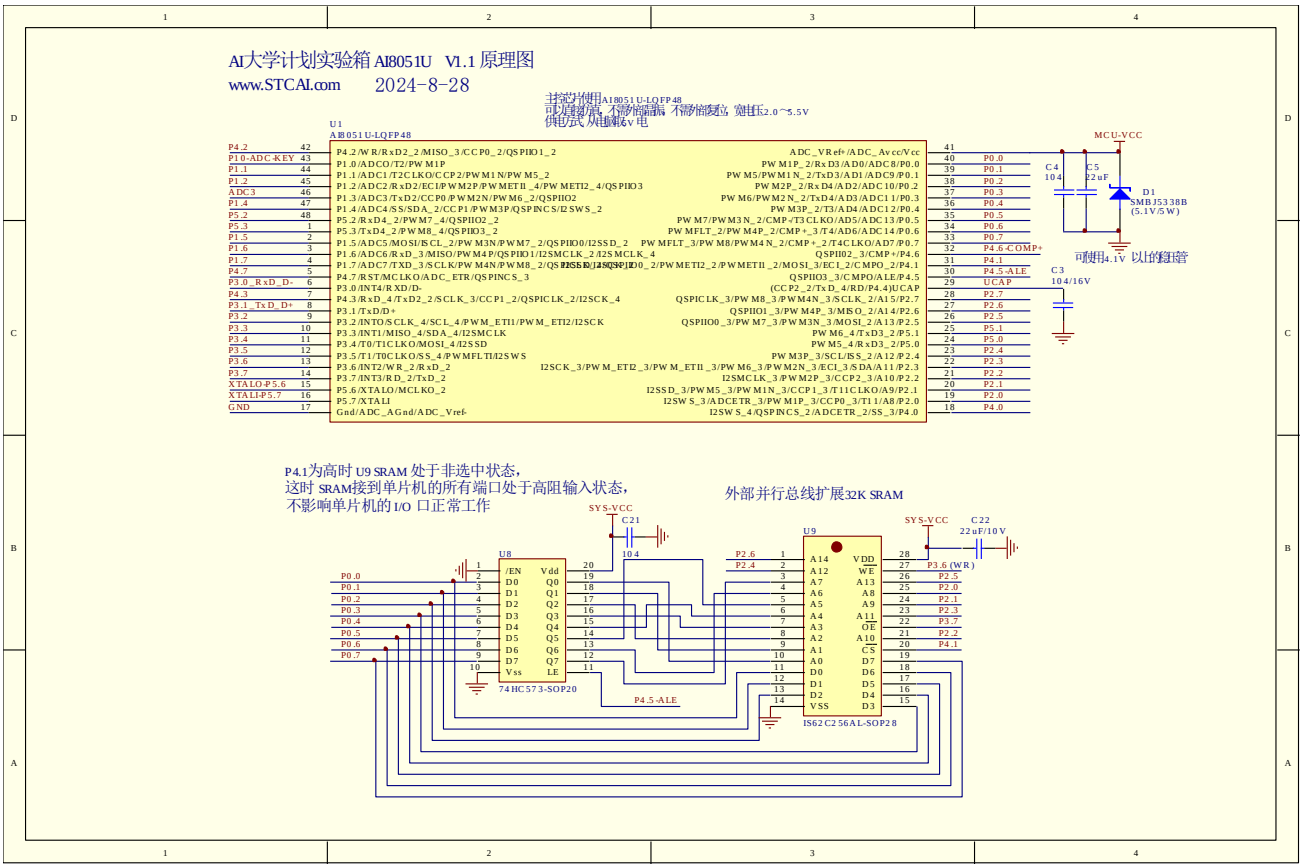
12.5.4　用户自定义内部 IRC 频率

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                           //头文件见下载软件
#include "intrins.h"                           //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   T22M_ADDR     CHIPID11              //22.1184MHz
#define   T24M_ADDR     CHIPID12              //24MHz
#define   T27M_ADDR     CHIPID13              //27MHz
#define   T30M_ADDR     CHIPID14              //30MHz
#define   T33M_ADDR     CHIPID15              //33.1776MHz
#define   T35M_ADDR     CHIPID16              //35MHz
#define   T36M_ADDR     CHIPID17              //36.864MHz
#define   T40M_ADDR     CHIPID18              //40MHz
#define   T44M_ADDR     CHIPID19              //44.2368MHz
#define   T48M_ADDR     CHIPID20              //48MHz
#define   VRT6M_ADDR    CHIPID21              //VRTRIM_6M
#define   VRT10M_ADDR   CHIPID22              //VRTRIM_10M
#define   VRT27M_ADDR   CHIPID23              //VRTRIM_27M
#define   VRT44M_ADDR   CHIPID24              //VRTRIM_44M

void main()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
```

```
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

//  //选择 22.1184MHz
//  CLKDIV = 0x04;
//  IRTRIM = T22M_ADDR;
//  VRTRIM = VRT27M_ADDR;
//  IRCBAND &= ~0x03;
//  IRCBAND |= 0x02;
//  CLKDIV = 0x00;

    //选择 24MHz
    CLKDIV = 0x04;
    IRTRIM = T24M_ADDR;
    VRTRIM = VRT27M_ADDR;
    IRCBAND &= ~0x03;
    IRCBAND |= 0x02;
    CLKDIV = 0x00;

//  //选择 27MHz
//  CLKDIV = 0x04;
//  IRTRIM = T27M_ADDR;
//  VRTRIM = VRT27M_ADDR;
//  IRCBAND &= ~0x03;
//  IRCBAND |= 0x02;
//  CLKDIV = 0x00;

//  //选择 30MHz
//  CLKDIV = 0x04;
//  IRTRIM = T30M_ADDR;
//  VRTRIM = VRT27M_ADDR;
//  IRCBAND &= ~0x03;
//  IRCBAND |= 0x02;
//  CLKDIV = 0x00;

//  //选择 33.1776MHz
//  CLKDIV = 0x04;
//  IRTRIM = T33M_ADDR;
//  VRTRIM = VRT27M_ADDR;
//  IRCBAND &= ~0x03;
//  IRCBAND |= 0x02;
//  CLKDIV = 0x00;

//  //选择 35MHz
```

```
//      CLKDIV = 0x04;
//      IRTRIM = T35M_ADDR;
//      VRTRIM = VRT44M_ADDR;
//      IRCBAND |= 0x03;
//      CLKDIV = 0x00;

//      //选择 44.2368MHz
//      CLKDIV = 0x04;
//      IRTRIM = T44M_ADDR;
//      VRTRIM = VRT44M_ADDR;
//      IRCBAND |= 0x03;
//      CLKDIV = 0x00;

//      //选择 48MHz
//      CLKDIV = 0x04;
//      IRTRIM = T48M_ADDR;
//      VRTRIM = VRT44M_ADDR;
//      IRCBAND |= 0x03;
//      CLKDIV = 0x00;

        while (1);
}
```

12.5.5　读写片外扩展 RAM




```
//测试工作频率为 11.0592MHz
```

```
#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   EXRAMB        ((unsigned char volatile far *)0x7f0000)
#define   EXRAMW        ((unsigned int volatile far *)0x7f0000)
#define   EXRAMD        ((unsigned long volatile far *)0x7f0000)

void main()
{
    char x8;
    int x16;
    long x32;

    EAXFR = 1;                                          //允许访问扩展的特殊寄存器，XFR
                                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                      //允许访问扩展的特殊寄存器，XFR
                                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                           //设置取程序代码等待时间，
                                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                          //设置访问片内的 xdata 速度，
                                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    EXTRAM = 1;                                 //使能访问片外
    BUS_SPEED = 2;                              //设置外部总线访问速度

    x8 = EXRAMB[0x0100];                        //从外部扩展 RAM 的 0x0100 地址读取 1 字节数据到变量 x8
    x16 = EXRAMW[0x0200];                       //从外部扩展 RAM 的 0x0400 地址读取 2 字节数据到变量 x16
    x32 = EXRAMD[0x0300];                       //从外部扩展 RAM 的 0x0C00 地址读取 4 字节数据到变量 x32
                                                //注意：Keil 中的多字节数据格式使用的是 BE(big-endian)格式，
                                                //即高字节存放在较低的地址，低字节存放在较高的地址

    EXRAMB[0x0101] = x8;                        //将变量 x8 的数据写入外部扩展 RAM 的 0x0101 地址
    EXRAMB[0x0205] = x16;                       //将变量 x16 的数据写入外部扩展 RAM 的 0x040A 地址
    EXRAMB[0x030c] = x32;                       //将变量 x32 的数据写入外部扩展 RAM 的 0x0C30 地址

    while (1);
}
```

13　特殊功能寄存器（SFR、XFR），EAXFR 允许访问 XFR

13.1　EAXFR 允许访问扩展的 XFR，取指令不减速，访问片内部 XRAM 不减速

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW2	BAH	EAXFR	-	I2C_S[1:0]		CMPO_S	S4_S	S3_S	S2_S

EAXFR：扩展 RAM 区特殊功能寄存器（XFR）访问控制寄存器

0：禁止访问 XFR

1：使能访问 XFR。

当需要访问 XFR 时，必须先将 EAXFR 置 1，才能对 XFR 进行正常的读写。建议上电初始化时直接设置为 1，后续不要再修改

Ai8051U 系列单片机存储器地址的编址如下：

区域	地址范围	功能	说明
数据区	00:0000H － 00:FFFFH	数据区（edata）	堆栈区
	01:0000H － 01:FFFFH	内部扩展数据区（xdata）	
	02:0000H － 7D:FFFFH	数据保留区	
	7E:0000H － 7E:FFFFH	**扩展 SFR 区（XFR）**	**需使能 EAXFR（P_SW2.7）才能访问**
	7F:0000H － 7F:FFFFH	外部数据区（xdata）	需使能 EXTRAM（AUXR.1）才能访问
代码区	80:0000H － FD:FFFFH	代码保留区	
	FE:0000H － FE:FFFFH	扩展代码区（ecode）	
	FF:0000H － FF:FFFFH	代码区（code）	

如需访问 XFR 区域的扩展 SFR，需要先将 EAXFR（P_SW2.7）置 1，由于 Ai8051U 的存储器地址的编址方式是线性编址的，XFR 区域的地址是独立的一块地址区域，所以可以在**上电系统初始化时将 EAXFR 寄存器写 1（例如：EAXFR = 1;），后续一直保持为 1 不用再修改**，即可正常访问 XFR 区域。

注意：由于 EAXFR 控制位和 S2_S、S3_S 等位寄存器位共用 P_SW2，在设置 P_SW2 中的 S2_S、S3_S 时，建议直接使用位操作语句（例如 S2_S = 0; S3_S = 1;），避免直接操作 P_SW2 寄存器。如代码中确实需要直接对 P_SW2 进行修改，也一定要使用与或指令（例如 P_SW2 &= ~0x01; P_SW2 |= 0x02;）。

13.2 Ai8051U 系列

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H					USBADR	S4CON	S4BUF	RSTCFG
F0H	B				USBCON	IAP_TPS	IAP_ADDRE	
E8H		WTST	CKCON	MXAX	USBDAT	DMAIR	IP3H	AUXINTIF
E0H	ACC			DPS			CMPCR1	CMPCR2
D8H					USBCLK	T4T3M	ADCCFG	IP3
D0H	PSW	PSW1	T4H	T4L	T3H	T3L	T2H	T2L
C8H	P5	P5M1	P5M0			SPSTAT	SPCTL	SPDAT
C0H	P4	WDT_CONTR	IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP_CONTR
B8H	IP	SADEN	P_SW2	P_SW3	ADC_CONTR	ADC_RES	ADC_RESL	P_SW4
B0H	P3	P3M1	P3M0	P4M1	P4M0	IP2	IP2H	IPH
A8H	IE	SADDR	WKTCL	WKTCH	S3CON	S3BUF	TA	IE2
A0H	P2	BUS_SPEED	P_SW1				VRTRIM	
98H	SCON	SBUF	S2CON	S2BUF		IRCBAND	-	IRTRIM
90H	P1	P1M1	P1M0	P0M1	P0M0	P2M1	P2M0	AUXR2
88H	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	INTCLKO
80H	P0	SP	DPL	DPH	DPXL	SPH		PCON

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
7EFEF8	PWMB_CCR6L	PWMB_CCR7H	PWMB_CCR7L	PWMB_CCR8H	PWMB_CCR8L	PWMB_BKR	PWMB_DTR	PWMB_OISR
7EFEF0	PWMB_PSCRH	PWMB_PSCRL	PWMB_ARRH	PWMB_ARRL	PWMB_RCR	PWMB_CCR5H	PWMB_CCR5L	PWMB_CCR6H
7EFEE8	PWMB_CCMR1	PWMB_CCMR2	PWMB_CCMR3	PWMB_CCMR4	PWMB_CCER1	PWMB_CCER2	PWMB_CNTRH	PWMB_CNTRL
7EFEE0	PWMB_CR1	PWMB_CR2	PWMB_SMCR	PWMB_ETR	PWMB_IER	PWMB_SR1	PWMB_SR2	PWMB_EGR
7EFED8	PWMA_CCR2L	PWMA_CCR3H	PWMA_CCR3L	PWMA_CCR4H	PWMA_CCR4L	PWMA_BKR	PWMA_DTR	PWMA_OISR
7EFED0	PWMA_PSCRH	PWMA_PSCRL	PWMA_ARRH	PWMA_ARRL	PWMA_RCR	PWMA_CCR1H	PWMA_CCR1L	PWMA_CCR2H
7EFEC8	PWMA_CCMR1	PWMA_CCMR2	PWMA_CCMR3	PWMA_CCMR4	PWMA_CCER1	PWMA_CCER2	PWMA_CNTRH	PWMA_CNTRL
7EFEC0	PWMA_CR1	PWMA_CR2	PWMA_SMCR	PWMA_ETR	PWMA_IER	PWMA_SR1	PWMA_SR2	PWMA_EGR
7EFEB8	PWMA_PS2	PWMA_RCRH	PWMB_RCRH					
7EFEB0	PWMA_ETRPS	PWMA_ENO	PWMA_PS	PWMA_IOAUX	PWMB_ETRPS	PWMB_ENO	PWMB_PS	PWMB_IOAUX
7EFEA8	ADCTIM					ADCEXCFG	CMPEXCFG	
7EFEA0	TM0PS	TM1PS	TM2PS	TM3PS	TM4PS			
7EFE98		RSTFLAG	RSTCR0	RSTCR1	RSTCR2	RSTCR3	RSTCR4	RSTCR5
7EFE90	SPI_CLKDIV	PWMA_CLKDIV	PWMB_CLKDIV	TFPU_CLKDIV	I2S_CLKDIV			
7EFE88	I2CMSAUX	I2CPSCR						
7EFE80	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
7EFE78	T11CR	T11PS	T11H	T11L				
7EFE70	RTCYEAR	RTCMONTH	RTCDAY	RTCHOUR	RTCMIN	RTCSEC	RTCSSEC	
7EFE68	INIYEAR	INIMONTH	INIDAY	INIHOUR	INIMIN	INISEC	INISSEC	INIWEEK RTCWEEK
7EFE60	RTCCR	RTCCFG	RTCIEN	RTCIF	ALAHOUR	ALAMIN	ALASEC	ALASSEC
7EFE50	LCMIFCFG	LCMIFCFG2	LCMIFCR	LCMIFSTA	LCMIFDATL	LCMIFDATH	LCMIFPSCR	
7EFE48	P0BP	P1BP	P2BP	P3BP	P4BP	P5BP		
7EFE40	P0PD	P1PD	P2PD	P3PD	P4PD	P5PD		
7EFE30	P0IE	P1IE	P2IE	P3IE	P4IE	P5IE		
7EFE28	P0DR	P1DR	P2DR	P3DR	P4DR	P5DR		
7EFE20	P0SR	P1SR	P2SR	P3SR	P4SR	P5SR		
7EFE18	P0NCS	P1NCS	P2NCS	P3NCS	P4NCS	P5NCS		
7EFE10	P0PU	P1PU	P2PU	P3PU	P4PU	P5PU		
7EFE08	X32KCR			HSCLKDIV				
7EFE00	CLKSEL	CLKDIV	HIRCCR	XOSCCR	IRC32KCR	MCLKOCR	IRCDB	IRC48MCR

7EFDF8	CHIPID[24]	CHIPID[25]	CHIPID[26]	CHIPID[27]	CHIPID[28]	CHIPID[29]	CHIPID[30]	CHIPID[31]
7EFDF0	CHIPID[16]	CHIPID[17]	CHIPID[18]	CHIPID[19]	CHIPID[20]	CHIPID[21]	CHIPID[22]	CHIPID[23]
7EFDE8	CHIPID[8]	CHIPID[9]	CHIPID[10]	CHIPID[11]	CHIPID[12]	CHIPID[13]	CHIPID[14]	CHIPID[15]
7EFDE0	CHIPID[0]	CHIPID[1]	CHIPID[2]	CHIPID[3]	CHIPID[4]	CHIPID[5]	CHIPID[6]	CHIPID[7]
7EFDC8	USART2CR1	USART2CR2	USART2CR3	USART2CR4	USART2CR5	USART2GTR	USART2BRH	USART2BRL
7EFDC0	USARTCR1	USARTCR2	USARTCR3	USARTCR4	USARTCR5	USARTGTR	USARTBRH	USARTBRL
7EFDB0					S2CFG	S2ADDR	S2ADEN	
7EFDA8	CRECR	CRECNTH	CRECNTL	CRERES				
7EFDA0	I2SMD	I2SMCKDIV						
7EFD98	I2SCR	I2SSR	I2SDRH	I2SDRL	I2SPRH	I2SPRL	I2SCFGH	I2SCFGL
7EFD88	UR1TOTE	UR2TOTE	UR3TOTE	UR4TOTE	SPITOTE	I2CTOTE		
7EFD80	SPITOCR	SPITOSR	SPITOTH	SPITOTL	I2CTOCR	I2CTOSR	I2CTOTH	I2CTOTL
7EFD78	UR3TOCR	UR3TOSR	UR3TOTH	UR3TOTL	UR4TOCR	UR4TOSR	UR4TOTH	UR4TOTL
7EFD70	UR1TOCR	UR1TOSR	UR1TOTH	UR1TOTL	UR2TOCR	UR2TOSR	UR2TOTH	UR2TOTL
7EFD60	PINIPL	PINIPH			CCON	CL	CH	CMOD
7EFD58	CCAPM0	CCAP0L	CCAP0H	PCA_PWM0	CCAPM1	CCAP1L	CCAP1H	PCA_PWM1
7EFD50	CCAPM2	CCAP2L	CCAP2H	PCA_PWM2	CCAPM3	CCAP3L	CCAP3H	PCA_PWM3
7EFD40	P0WKUE	P1WKUE	P2WKUE	P3WKUE	P4WKUE	P5WKUE		
7EFD30	P0IM1	P1IM1	P2IM1	P3IM1	P4IM1	P5IM1		
7EFD20	P0IM0	P1IM0	P2IM0	P3IM0	P4IM0	P5IM0		
7EFD10	P0INTF	P1INTF	P2INTF	P3INTF	P4INTF	P5INTF		
7EFD00	P0INTE	P1INTE	P2INTE	P3INTE	P4INTE	P5INTE		

7EFBF8	HSSPI_CFG	HSSPI_CFG2	HSSPI_STA	HSSPI_PSCR				
7EFBF0	HSPWMA_CFG	HSPWMA_ADR	HSPWMA_DAT	-	HSPWMB_CFG	HSPWMB_ADR	HSPWMB_DAT	-
7EFBE8	CHIPIDX[24]	CHIPIDX[25]	CHIPIDX[26]	CHIPIDX[27]	CHIPIDX[28]	CHIPIDX[29]	CHIPIDX[30]	CHIPIDX[31]
7EFBE0	CHIPIDX[16]	CHIPIDX[17]	CHIPIDX[18]	CHIPIDX[19]	CHIPIDX[20]	CHIPIDX[21]	CHIPIDX[22]	CHIPIDX[23]
7EFBD8	CHIPIDX[8]	CHIPIDX[9]	CHIPIDX[10]	CHIPIDX[11]	CHIPIDX[12]	CHIPIDX[13]	CHIPIDX[14]	CHIPIDX[15]
7EFBD0	CHIPIDX[0]	CHIPIDX[1]	CHIPIDX[2]	CHIPIDX[3]	CHIPIDX[4]	CHIPIDX[5]	CHIPIDX[6]	CHIPIDX[7]

7EFAF8	DMA_ARBCFG	DMA_ARBSTA						
7EFAF0	DMA_P2P_CR1	DMA_P2P_CR2						
7EFAD8	DMA_QSPI_RXAL			DMA_QSPI_AMTH	DMA_QSPI_DONEH		DMA_QSPI_ITVH	DMA_QSPI_ITVL
7EFAD0	DMA_QSPI_CFG	DMA_QSPI_CR	DMA_QSPI_STA	DMA_QSPI_AMT	DMA_QSPI_DONE	DMA_QSPI_TXAH	DMA_QSPI_TXAL	DMA_QSPI_RXAH
7EFAC8	DMA_UR1_ITVH	DMA_UR1_ITVL	DMA_UR2_ITVH	DMA_UR2_ITVL	DMA_UR3_ITVH	DMA_UR3_ITVL	DMA_UR4_ITVH	DMA_UR4_ITVL
7EFAC0	DMA_I2ST_AMTH	DMA_I2ST_DONEH	DMA_I2SR_AMTH	DMA_I2SR_DONEH	DMA_I2C_ITVH	DMA_I2C_ITVL	DMA_I2S_ITVH	DMA_I2S_ITVL
7EFAB8	DMA_I2SR_CFG	DMA_I2SR_CR	DMA_I2SR_STA	DMA_I2SR_AMT	DMA_I2SR_DONE	DMA_I2SR_RXAH	DMA_I2SR_RXAL	
7EFAB0	DMA_I2ST_CFG	DMA_I2ST_CR	DMA_I2ST_STA	DMA_I2ST_AMT	DMA_I2ST_DONE	DMA_I2ST_TXAH	DMA_I2ST_TXAL	
7EFAA8	DMA_I2CT_AMTH	DMA_I2CT_DONEH	DMA_I2CR_AMTH	DMA_I2CR_DONEH		DMA_I2C_CR	DMA_I2C_ST1	DMA_I2C_ST2
7EFAA0	DMA_I2CR_CFG	DMA_I2CR_CR	DMA_I2CR_STA	DMA_I2CR_AMT	DMA_I2CR_DONE	DMA_I2CR_RXAH	DMA_I2CR_RXAL	
7EFA98	DMA_I2CT_CFG	DMA_I2CT_CR	DMA_I2CT_STA	DMA_I2CT_AMT	DMA_I2CT_DONE	DMA_I2CT_TXAH	DMA_I2CT_TXAL	
7EFA90	DMA_UR3T_AMTH	DMA_UR3T_DONEH	DMA_UR3R_AMTH	DMA_UR3R_DONEH	DMA_UR4T_AMTH	DMA_UR4T_DONEH	DMA_UR4R_AMTH	DMA_UR4R_DONEH
7EFA88	DMA_UR1T_AMTH	DMA_UR1T_DONEH	DMA_UR1R_AMTH	DMA_UR1R_DONEH	DMA_UR2T_AMTH	DMA_UR2T_DONEH	DMA_UR2R_AMTH	DMA_UR2R_DONEH
7EFA80	DMA_M2M_AMTH	DMA_M2M_DONEH	DMA_ADC_AMTH	DMA_ADC_DONEH	DMA_SPI_AMTH	DMA_SPI_DONEH	DMA_LCM_AMTH	DMA_LCM_DONEH
7EFA78	DMA_LCM_RXAL						DMA_LCM_ITVH	DMA_LCM_ITVL
7EFA70	DMA_LCM_CFG	DMA_LCM_CR	DMA_LCM_STA	DMA_LCM_AMT	DMA_LCM_DONE	DMA_LCM_TXAH	DMA_LCM_TXAL	DMA_LCM_RXAH
7EFA68	DMA_UR4R_CFG	DMA_UR4R_CR	DMA_UR4R_STA	DMA_UR4R_AMT	DMA_UR4R_DONE	DMA_UR4R_RXAH	DMA_UR4R_RXAL	
7EFA60	DMA_UR4T_CFG	DMA_UR4T_CR	DMA_UR4T_STA	DMA_UR4T_AMT	DMA_UR4T_DONE	DMA_UR4T_TXAH	DMA_UR4T_TXAL	
7EFA58	DMA_UR3R_CFG	DMA_UR3R_CR	DMA_UR3R_STA	DMA_UR3R_AMT	DMA_UR3R_DONE	DMA_UR3R_RXAH	DMA_UR3R_RXAL	
7EFA50	DMA_UR3T_CFG	DMA_UR3T_CR	DMA_UR3T_STA	DMA_UR3T_AMT	DMA_UR3T_DONE	DMA_UR3T_TXAH	DMA_UR3T_TXAL	
7EFA48	DMA_UR2R_CFG	DMA_UR2R_CR	DMA_UR2R_STA	DMA_UR2R_AMT	DMA_UR2R_DONE	DMA_UR2R_RXAH	DMA_UR2R_RXAL	
7EFA40	DMA_UR2T_CFG	DMA_UR2T_CR	DMA_UR2T_STA	DMA_UR2T_AMT	DMA_UR2T_DONE	DMA_UR2T_TXAH	DMA_UR2T_TXAL	
7EFA38	DMA_UR1R_CFG	DMA_UR1R_CR	DMA_UR1R_STA	DMA_UR1R_AMT	DMA_UR1R_DONE	DMA_UR1R_RXAH	DMA_UR1R_RXAL	
7EFA30	DMA_UR1T_CFG	DMA_UR1T_CR	DMA_UR1T_STA	DMA_UR1T_AMT	DMA_UR1T_DONE	DMA_UR1T_TXAH	DMA_UR1T_TXAL	
7EFA28	DMA_SPI_RXAL	DMA_SPI_CFG2					DMA_SPI_ITVH	DMA_SPI_ITVL
7EFA20	DMA_SPI_CFG	DMA_SPI_CR	DMA_SPI_STA	DMA_SPI_AMT	DMA_SPI_DONE	DMA_SPI_TXAH	DMA_SPI_TXAL	DMA_SPI_RXAH
7EFA18	DMA_ADC_RXAL	DMA_ADC_CFG2	DMA_ADC_CHSW0	DMA_ADC_CHSW1			DMA_ADC_ITVH	DMA_ADC_ITVL
7EFA10	DMA_ADC_CFG	DMA_ADC_CR	DMA_ADC_STA	DMA_ADC_AMT	DMA_ADC_DONE			DMA_ADC_RXAH
7EFA08	DMA_M2M_RXAL							
7EFA00	DMA_M2M_CFG	DMA_M2M_CR	DMA_M2M_STA	DMA_M2M_AMT	DMA_M2M_DONE	DMA_M2M_TXAH	DMA_M2M_TXAL	DMA_M2M_RXAH

7EF998	DMA_PWMAR_RXAH	DMA_PWMAR_RXAL						
7EF990	DMA_PWMAR_CFG	DMA_PWMAR_CR	DMA_PWMAR_STA		DMA_PWMAR_AMTH	DMA_PWMAR_AMT	DMA_PWMAR_DONE_NH	DMA_PWMAR_DONE_N
7EF988	DMA_PWMAT_TXAH	DMA_PWMAT_TXAL					PWMA_BMM_ITVH	PWMA_BMM_ITVL
7EF980	DMA_PWMAT_CFG	DMA_PWMAT_CR	DMA_PWMAT_STA		DMA_PWMAT_AMTH	DMA_PWMAT_AMT	DMA_PWMAT_DONE_NH	DMA_PWMAT_DONE_N
7EF948	PWMA_DER	PWMA_DBA	PWMA_DBL	PWMA_DMACR				
7EF940	PWMA_CCR5H	PWMA_CCR5L	PWMA_CCR5X	PWMA_CCR6H	PWMA_CCR6L			
7EF938	PWMA_CCMR1X	PWMA_CCMR2X	PWMA_CCMR3X	PWMA_CCMR4X	PWMA_CCMR5	PWMAX_CCMR5X	PWMA_CCMR6	PWMA_CCMR6X
7EF930	PWMA_ENO2	PWMA_IOAUX2	PWMA_CR3	PWMA_SR3	PWMA_CCER3			
7EF928	QSPI_PSMAR1				QSPI_PIR1	QSPI_PIR2		
7EF920	QSPI_DR				QSPI_PSMKR1			
7EF918	QSPI_AR1	QSPI_AR2	QSPI_AR3	QSPI_AR4	QSPI_ABR			
7EF910	QSPI_DLR1	QSPI_DLR2			QSPI_CCR1	QSPI_CCR2	QSPI_CCR3	QSPI_CCR4
7EF908	QSPI_FCR	QSPI_HCR1	QSPI_HCR2					
7EF900	QSPI_CR1	QSPI_CR2	QSPI_CR3	QSPI_CR4	QSPI_DCR1	QSPI_DCR2	QSPI_SR1	QSPI_SR2

13.3　特殊功能寄存器列表（SFR：0x80-0xFF）

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
P0	P0 端口	80H	P07	P06	P05	P04	P03	P02	P01	P00	1111,1111
SP	堆栈指针	81H	SP[7:0]								0000,0111
DPL	数据指针（低字节）	82H	DPTR[7:0]								0000,0000
DPH	数据指针（高字节）	83H	DPTR[15:8]								0000,0000
DPXL	数据指针（最高字节）	84H	DPTR[23:16]								0000,0001
SPH	堆栈指针高字节	85H	SP[15:8]								0000,0000
PCON	电源控制寄存器	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
TCON	定时器控制寄存器	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
TMOD	定时器模式寄存器	89H	T1_GATE	T1_C/T	T1_M1	T1_M0	T0_GATE	T0_C/T	T0_M1	T0_M0	0000,0000
TL0	定时器 0 低 8 位寄存器	8AH	TMR0[7:0]								0000,0000
TL1	定时器 1 低 8 位寄存器	8BH	TMR1[7:0]								0000,0000
TH0	定时器 0 高 8 位寄存器	8CH	TMR0[15:8]								0000,0000
TH1	定时器 1 高 8 位寄存器	8DH	TMR1[15:8]								0000,0000
AUXR	辅助寄存器 1	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1BRT	0000,0001
INTCLKO	中断与时钟输出控制寄存器	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000
P1	P1 端口	90H	P17	P16	P15	P14	P13	P12	P11	P10	1111,1111
P1M1	P1 口配置寄存器 1	91H	P1M1[7:0]								1111,1111
P1M0	P1 口配置寄存器 0	92H	P1M0[7:0]								0000,0000
P0M1	P0 口配置寄存器 1	93H	P0M1[7:0]								1111,1111
P0M0	P0 口配置寄存器 0	94H	P0M0[7:0]								0000,0000
P2M1	P2 口配置寄存器 1	95H	P2M1[7:0]								1111,1111
P2M0	P2 口配置寄存器 0	96H	P2M0[7:0]								0000,0000
AUXR2	辅助寄存器 2	97H	TINY	CPUMODE	-	-	-	-	-	-	0nxx,xxxx
SCON	串口 1 控制寄存器	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
SBUF	串口 1 数据寄存器	99H	SBUF[7:0]								0000,0000
S2CON	串口 2 控制寄存器	9AH	S2SM0/FE	S2SM1	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0000,0000
S2BUF	串口 2 数据寄存器	9BH	S2BUF[7:0]								0000,0000
IRCBAND	IRC 频段选择检测	9DH	USBCKS	USBCKS2	-	-	-	-	SEL[1:0]		10xx,xxnn
IRTRIM	IRC 频率调整寄存器	9FH	IRTRIM[7:0]								nnnn,nnnn
P2	P2 端口	A0H	P27	P26	P25	P24	P23	P22	P21	P20	1111,1111
BUS_SPEED	总线速度控制寄存器	A1H	RW_S[1:0]		-	-	-	SPEED[2:0]			00xx,x000
P_SW1	外设端口切换寄存器 1	A2H	S1_S[1:0]		-		SPI_S[1:0]		-		nnxx,00xx
IE	中断允许寄存器	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000,0000
SADDR	串口 1 从机地址寄存器	A9H	SADDR[7:0]								0000,0000
WKTCL	掉电唤醒定时器低字节	AAH	WKTC[7:0]								1111,1111
WKTCH	掉电唤醒定时器高字节	ABH	WKTEN	WKTC[14:8]							0111,1111
S3CON	串口 3 控制寄存器	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000
S3BUF	串口 3 数据寄存器	ADH	S3BUF[7:0]								0000,0000
TA	DPTR 时序控制寄存器	AEH	TA[7:0]								0000,0000
IE2	中断允许寄存器 2	AFH	EUSB	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	0000,0000

P3	P3 端口	B0H	P37	P36	P35	P34	P33	P32	P31	P30	1111,1111
P3M1	P3 口配置寄存器 1	B1H	P3M1[7:0]								1111,1100
P3M0	P3 口配置寄存器 0	B2H	P3M0[7:0]								0000,0000
P4M1	P4 口配置寄存器 1	B3H	P4M1[7:0]								1111,1111
P4M0	P4 口配置寄存器 0	B4H	P4M0[7:0]								0000,0000
IP2	中断优先级控制寄存器 2	B5H	PUSB	PI2C	PCMP	PX4	PPWMB	PPWMA	PSPI	PS2	0000,0000
IP2H	高中断优先级控制寄存器 2	B6H	PUSBH	PI2CH	PCMPH	PX4H	PPWMBH	PPWMAH	PSPIH	PS2H	0000,0000
IPH	高中断优先级控制寄存器	B7H	PPCAH	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H	x000,0000
IP	中断优先级控制寄存器	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	x000,0000
SADEN	串口 1 从机地址屏蔽寄存器	B9H	SADEN[7:0]								0000,0000
P_SW2	外设端口切换寄存器 2	BAH	EAXFR	-	I2C_S[1:0]		CMPO_S	S4_S	S3_S	S2_S	0x00,0000
P_SW3	外设端口切换寄存器 3	BBH	I2S_S[1:0]		S2SPI_S[1:0]		S1SPI_S[1:0]		-		0000,00xx
ADC_CONTR	ADC 控制寄存器	BCH	ADC_POWER	ADC_START	ADC_FLAG	ADC_EPWMT	ADC_CHS[3:0]				0000,0000
ADC_RES	ADC 转换结果高位寄存器	BDH									0000,0000
ADC_RESL	ADC 转换结果低位寄存器	BEH									0000,0000
P_SW4	外设端口切换寄存器 4	BFH	-		-		-		QSPI_S[1:0]		xxxx,xx00
P4	P4 端口	C0H	P47	P46	P45	P44	P43	P42	P41	P40	1111,1111
WDT_CONTR	看门狗控制寄存器	C1H	WDT_FLAG	-	EN_WDT	CLR_WDT	IDL_WDT	WDT_PS[2:0]			0x00,0000
IAP_DATA	IAP 数据寄存器	C2H	IAP_DATA[7:0]								0000,0000
IAP_ADDRH	IAP 高地址寄存器	C3H	IAP_ADDR[15:8]								0000,0000
IAP_ADDRL	IAP 低地址寄存器	C4H	IAP_ADDR[7:0]								0000,0000
IAP_CMD	IAP 命令寄存器	C5H	-	-	-	-	-	CMD[2:0]			xxxx,x000
IAP_TRIG	IAP 触发寄存器	C6H									0000,0000
IAP_CONTR	IAP 控制寄存器	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	SWBS2	-	-	-	0000,0xxx
P5	P5 端口	C8H	P57	P56	P55	P54	P53	P52	P51	P50	1111,1111
P5M1	P5 口配置寄存器 1	C9H	P5M1[7:0]								1111,1111
P5M0	P5 口配置寄存器 0	CAH	P5M0[7:0]								xx00,0000
P6M1	P6 口配置寄存器 1	CBH	P6M1[7:0]								1111,1111
P6M0	P6 口配置寄存器 0	CCH	P6M0[7:0]								0000,0000
SPSTAT	SPI 状态寄存器	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx,xxxx
SPCTL	SPI 控制寄存器	CEH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR[1:0]		0000,0100
SPDAT	SPI 数据寄存器	CFH	SPDAT[7:0]								0000,0000
PSW	程序状态字寄存器	D0H	CY	AC	F0	RS1	RS0	OV	F1	P	0000,0000
PSW1	程序状态字 1 寄存器	D1H	CY	AC	N	RS1	RS0	OV	Z		0000,000x
T4H	定时器 4 高字节	D2H	TMR4[15:8]								0000,0000
T4L	定时器 4 低字节	D3H	TMR4[7:0]								0000,0000
T3H	定时器 3 高字节	D4H	TMR3[15:8]								0000,0000
T3L	定时器 3 低字节	D5H	TMR3[7:0]								0000,0000
T2H	定时器 2 高字节	D6H	TMR2[15:8]								0000,0000
T2L	定时器 2 低字节	D7H	TMR2[7:0]								0000,0000
USBCLK	USB 时钟控制寄存器	DCH	ENCKM	PCKI[1:0]		CRE	TST_USB	TST_PHY	PHYTST[1:0]		0010,0000
T4T3M	定时器 4/3 控制寄存器	DDH	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO	0000,0000
ADCCFG	ADC 配置寄存器	DEH	-	-	RESFMT	-	SPEED[3:0]				xx0x,0000
IP3	中断优先级控制寄存器 3	DFH	-	-	-	-	PI2S	PRTC	PS4	PS3	xxxx,0000

ACC	累加器	E0H								0000,0000
P7M1	P7 口配置寄存器 1	E1H	P7M1[7:0]							1111,1111
P7M0	P7 口配置寄存器 0	E2H	P7M0[7:0]							0000,0000
DPS	DPTR 指针选择器	E3H	ID1	ID0	TSL	AU1	AU0	-	-	SEL	0000,0xx0
CMPCR1	比较器控制寄存器 1	E6H	CMPEN	CMPIF	PIE	NIE	-	-	CMPOE	CMPRES	0000,xx00
CMPCR2	比较器控制寄存器 2	E7H	INVCMPO	DISFLT	LCDTY[5:0]						0000,0000
P6	P6 端口	E8H	P67	P66	P65	P64	P63	P62	P61	P60	1111,1111
WTST	程序读取等待控制寄存器	E9H	WTST[7:0]								0000,0111
CKCON	XRAM 控制寄存器	EAH	-	-	-	T1M	T0M	CKCON[2:0]			0000,0111
MXAX	MOVX 扩展地址寄存器	EBH	MXAX[7:0]								0000,0001
USBDAT	USB 数据寄存器	ECH	USBDAT[7:0]								0000,0000
DMAIR	FMU DMA 指令寄存器	EDH	DMAIR[7:0]								0000,0000
IP3H	高中断优先级控制寄存器 3	EEH	-	-	-	-	PI2SH	PRTCH	PS4H	PS3H	xxxx,0000
AUXINTIF	扩展外部中断标志寄存器	EFH	-	INT4IF	INT3IF	INT2IF	-	T4IF	T3IF	T2IF	x000,x000
B	B 寄存器	F0H									0000,0000
USBCON	USB 控制寄存器	F4H	ENUSB	ENUSBRST	PS2M	PUEN	PDEN	DFREC	DP	DM	0000,0000
IAP_TPS	IAP 等待时间控制寄存器	F5H	-	-	IAP_TPS[5:0]						xx00,0000
IAP_ADDRE	IAP 扩展高地址寄存器	F6H									1111,1111
P7	P7 端口	F8H	P77	P76	P75	P74	P73	P72	P71	P70	1111,1111
USBADR	USB 地址寄存器	FCH	BUSY	AUTORD	UADR[5:0]						0000,0000
S4CON	串口 4 控制寄存器	FDH	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000,0000
S4BUF	串口 4 数据寄存器	FEH	S4BUF[7:0]								0000,0000
RSTCFG	复位配置寄存器	FFH	-	ENLVR	-	P47RST	-	-	LVDS[1:0]		x0x0,xx00

13.4 扩展特殊功能寄存器列表（XFR：0x7EFE00-0x7EFEFF）

下列特殊功能寄存器为扩展 SFR（XFR），逻辑地址位于 XDATA 区域，访问前需要将 P_SW2 寄存器的最高位（EAXFR）置 1，然后使用 MOV @DRk, Rm 和 MOV Rm, @DRk 指令进行访问，例如：

```
MOV     A,#00H
MOV     WR6,#WORD0 CLKSEL       ; CLKSEL 可换为需要访问的寄存器
MOV     WR4,#WORD2 CLKSEL
MOV     @DR4,R11
```

和

```
MOV     WR6,#WORD0 CLKSEL       ; CLKSEL 可换为需要访问的寄存器
MOV     WR4,#WORD2 CLKSEL
MOV     R11,@DR4
```

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CLKSEL	时钟选择寄存器	7EFE00H	CKMS	HSIOCK	-	-	MCK2SEL[1:0]		MCKSEL[1:0]		00xx,0000
CLKDIV	时钟分频寄存器	7EFE01H	DIV[7:0]								0000,0100
HIRCCR	内部高速振荡器控制寄存器	7EFE02H	ENHIRC	-	-	-	-	-	-	HIRCST	1xxx,xxx0
XOSCCR	外部晶振控制寄存器	7EFE03H	ENXOSC	XITYPE	GAIN	-	XCFILTER[1:0]		-	XOSCST	000x,00x0
IRC32KCR	内部低速振荡器控制寄存器	7EFE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST	0xxx,xxx0
MCLKOCR	主时钟输出控制寄存器	7EFE05H	MCLKO_S	MCLKODIV[6:0]							0000,0000
IRCDB	内部高速振荡器去抖控制	7EFE06H	DB[7:0]								1000,0000
IRC48MCR	内部 48M 振荡器控制寄存器	7EFE07H	ENIRC48M	-	-	-	-	-	-	IRC48MST	1xxx,xxx0
X32KCR	外部 32K 晶振控制寄存器	7FFE08H	ENX32K	GAIN32K	-	-	-	-	-	X32KST	00xx,xxx0
HSCLKDIV	高速时钟分频寄存器	7EFE0BH	DIV[7:0]								0000,0010
P0PU	P0 口上拉电阻控制寄存器	7EFE10H	P0PU[7:0]								0000,0000
P1PU	P1 口上拉电阻控制寄存器	7EFE11H	P1PU[7:0]								0000,0000
P2PU	P2 口上拉电阻控制寄存器	7EFE12H	P2PU[7:0]								0000,0000
P3PU	P3 口上拉电阻控制寄存器	7EFE13H	P3PU[7:0]								0000,0000
P4PU	P4 口上拉电阻控制寄存器	7EFE14H	P4PU[7:0]								0000,0000
P5PU	P5 口上拉电阻控制寄存器	7EFE15H	P5PU[7:0]								0000,0000
P6PU	P6 口上拉电阻控制寄存器	7EFE16H	P6PU[7:0]								0000,0000
P7PU	P7 口上拉电阻控制寄存器	7EFE17H	P7PU[7:0]								0000,0000
P0NCS	P0 口施密特触发控制寄存器	7EFE18H	P0NCS[7:0]								0000,0000
P1NCS	P1 口施密特触发控制寄存器	7EFE19H	P1NCS[7:0]								0000,0000
P2NCS	P2 口施密特触发控制寄存器	7EFE1AH	P2NCS[7:0]								0000,0000
P3NCS	P3 口施密特触发控制寄存器	7EFE1BH	P3NCS[7:0]								0000,0000
P4NCS	P4 口施密特触发控制寄存器	7EFE1CH	P4NCS[7:0]								0000,0000
P5NCS	P5 口施密特触发控制寄存器	7EFE1DH	P5NCS[7:0]								0000,0000
P6NCS	P6 口施密特触发控制寄存器	7EFE1EH	P6NCS[7:0]								0000,0000

符号	描述	地址	位地址与符号							复位值
P7NCS	P7 口施密特触发控制寄存器	7EFE1FH	P7NCS[7:0]							0000,0000
P0SR	P0 口电平转换速率寄存器	7EFE20H	P0SR[7:0]							1111,1111
P1SR	P1 口电平转换速率寄存器	7EFE21H	P1SR[7:0]							1111,1111
P2SR	P2 口电平转换速率寄存器	7EFE22H	P2SR[7:0]							1111,1111
P3SR	P3 口电平转换速率寄存器	7EFE23H	P3SR[7:0]							1111,1111
P4SR	P4 口电平转换速率寄存器	7EFE24H	P4SR[7:0]							1111,1111
P5SR	P5 口电平转换速率寄存器	7EFE25H	P5SR[7:0]							1111,1111
P6SR	P6 口电平转换速率寄存器	7EFE26H	P6SR[7:0]							1111,1111
P7SR	P7 口电平转换速率寄存器	7EFE27H	P7SR[7:0]							1111,1111
P0DR	P0 口驱动电流控制寄存器	7EFE28H	P0DR[7:0]							1111,1111
P1DR	P1 口驱动电流控制寄存器	7EFE29H	P1DR[7:0]							1111,1111
P2DR	P2 口驱动电流控制寄存器	7EFE2AH	P2DR[7:0]							1111,1111
P3DR	P3 口驱动电流控制寄存器	7EFE2BH	P3DR[7:0]							1111,1111
P4DR	P4 口驱动电流控制寄存器	7EFE2CH	P4DR[7:0]							1111,1111
P5DR	P5 口驱动电流控制寄存器	7EFE2DH	P5DR[7:0]							1111,1111
P6DR	P6 口驱动电流控制寄存器	7EFE2EH	P6DR[7:0]							1111,1111
P7DR	P7 口驱动电流控制寄存器	7EFE2FH	P7DR[7:0]							1111,1111
P0IE	P0 口输入使能控制寄存器	7EFE30H	P0IE[7:0]							1111,1111
P1IE	P1 口输入使能控制寄存器	7EFE31H	P1IE[7:0]							1111,1111
P2IE	P2 口输入使能控制寄存器	7EFE32H	P2IE[7:0]							1111,1111
P3IE	P3 口输入使能控制寄存器	7EFE33H	P3IE[7:0]							1111,1111
P4IE	P4 口输入使能控制寄存器	7EFE34H	P4IE[7:0]							1111,1111
P5IE	P5 口输入使能控制寄存器	7EFE35H	P5IE[7:0]							1111,1111
P6IE	P6 口输入使能控制寄存器	7EFE36H	P6IE[7:0]							1111,1111
P7IE	P7 口输入使能控制寄存器	7EFE37H	P7IE[7:0]							1111,1111
P0PD	P0 口下拉电阻控制寄存器	7EFE40H	P0PD[7:0]							0000,0000
P1PD	P1 口下拉电阻控制寄存器	7EFE41H	P1PD[7:0]							0000,0000
P2PD	P2 口下拉电阻控制寄存器	7EFE42H	P2PD[7:0]							0000,0000
P3PD	P3 口下拉电阻控制寄存器	7EFE43H	P3PD[7:0]							0000,0000
P4PD	P4 口下拉电阻控制寄存器	7EFE44H	P4PD[7:0]							0000,0000
P5PD	P5 口下拉电阻控制寄存器	7EFE45H	P5PD[7:0]							0000,0000
P6PD	P6 口下拉电阻控制寄存器	7EFE46H	P6PD[7:0]							0000,0000
P7PD	P7 口下拉电阻控制寄存器	7EFE47H	P7PD[7:0]							0000,0000
P0BP	P0 口模式用户控制寄存器	7EFE48H	P0BP[7:0]							1111,1111
P1BP	P1 口模式用户控制寄存器	7EFE49H	P1BP[7:0]							1111,1111
P2BP	P2 口模式用户控制寄存器	7EFE4AH	P2BP[7:0]							1111,1111
P3BP	P3 口模式用户控制寄存器	7EFE4BH	P3BP[7:0]							1111,1111
P4BP	P4 口模式用户控制寄存器	7EFE4CH	P4BP[7:0]							1111,1111
P5BP	P5 口模式用户控制寄存器	7EFE4DH	P5BP[7:0]							1111,1111
P6BP	P6 口模式用户控制寄存器	7EFE4EH	P6BP[7:0]							1111,1111
P7BP	P7 口模式用户控制寄存器	7EFE4FH	P7BP[7:0]							1111,1111
LCMIFCFG	TFT 彩屏接口配置寄存器	7EFE50H	LCMIFIE	-	LCMIFIP[1:0]		LCMIFDPS[1:0]		D16_D8	M68_I80	0x00,0000
LCMIFCFG2	TFT 彩屏接口配置寄存器 2	7EFE51H	-	LCMIFCPS[1:0]		SETUPT[2:0]			HOLDT[1:0]		x000,0000

符号	描述	地址	位地址与符号								复位值
LCMIFCR	TFT 彩屏接口控制寄存器	7EFE52H	ENLCMIF	-	-	ST_ENDIAN	-	CMD[2:0]			0xx0,x000
LCMIFSTA	TFT 彩屏接口状态寄存器	7EFE53H	-	-	-	-	-	-	-	LCMIFIF	xxxx,xxx0
LCMIFDATL	TFT 彩屏接口低字节数据	7EFE54H	LCMIFDAT[7:0]								0000,0000
LCMIFDATH	TFT 彩屏接口高字节数据	7EFE55H	LCMIFDAT[15:8]								0000,0000
LCMIFPSCR	TFT 彩屏接口时钟预分频	7EFE56H	LCMIFPSCR[7:0]								0000,0000
RTCCR	RTC 控制寄存器	7EFE60H	-	-	-	-	-	-	-	RUNRTC	xxxx,xxx0
RTCCFG	RTC 配置寄存器	7EFE61H	-	-	-	-	-	-	RTCCKS	SETRTC	xxxx,xx00
RTCIEN	RTC 中断使能寄存器	7EFE62H	EALAI	EDAYI	EHOURI	EMINI	ESECI	ESEC2I	ESEC8I	ESEC32I	0000,0000
RTCIF	RTC 中断请求寄存器	7EFE63H	ALAIF	DAYIF	HOURIF	MINIF	SECIF	SEC2IF	SEC8IF	SEC32IF	0000,0000
ALAHOUR	RTC 闹钟的小时值	7EFE64H	-	-	-	HOUR[4:0]					xxx0,0000
ALAMIN	RTC 闹钟的分钟值	7EFE65H	-	-	MIN[5:0]						xx00,0000
ALASEC	RTC 闹钟的秒值	7EFE66H	-	-	SEC[5:0]						xx00,0000
ALASSEC	RTC 闹钟的 1/128 秒值	7EFE67H	-	SSEC[6:0]							x000,0000
INIYEAR	RTC 年初始化	7EFE68H	-	YEAR[6:0]							x000,0000
INIMONTH	RTC 月初始化	7EFE69H	-	-	-	-	MONTH[3:0]				xxxx,0000
INIDAY	RTC 日初始化	7EFE6AH	-	-	-	DAY[4:0]					xxx0,0000
INIHOUR	RTC 小时初始化	7EFE6BH	-	-	-	HOUR[4:0]					xxx0,0000
INIMIN	RTC 分钟初始化	7EFE6CH	-	-	MIN[5:0]						xx00,0000
INISEC	RTC 秒初始化	7EFE6DH	-	-	SEC[5:0]						xx00,0000
INISSEC	RTC1/128 秒初始化	7EFE6EH	-	SSEC[6:0]							x000,0000
INIWEEK	RTC 星期初始化	7EFE6FH	-	-	-	-	-	WEEK[2:0]			xxxx,x000
RTCWEEK	RTC 的星期计数值	7EFE6FH	-	-	-	-	-	WEEK[2:0]			xxxx,x000
RTCYEAR	RTC 的年计数值	7EFE70H	-	YEAR[6:0]							x000,0000
RTCMONTH	RTC 的月计数值	7EFE71H	-	-	-	-	MONTH[3:0]				xxxx,0000
RTCDAY	RTC 的日计数值	7EFE72H	-	-	-	DAY[4:0]					xxx0,0000
RTCHOUR	RTC 的小时计数值	7EFE73H	-	-	-	HOUR[4:0]					xxx0,0000
RTCMIN	RTC 的分钟计数值	7EFE74H	-	-	MIN[5:0]						xx00,0000
RTCSEC	RTC 的秒计数值	7EFE75H	-	-	SEC[5:0]						xx00,0000
RTCSSEC	RTC 的 1/128 秒计数值	7EFE76H	-	SSEC[6:0]							x000,0000
T11CR	定时器 T11 控制寄存器	7EFE78H	T11R	T11_C/T	T11CLKO	T11x12	T11CS[1:0]		ET11I	T11IF	0000,0000
T11PS	T11 时钟预分频寄存器	7EFE79H	PS[7:0]								0000,0000
T11H	定时器 T11 高字节	7EFE7AH	CNT[15:8]								0000,0000
T11L	定时器 T11 低字节	7EFE7BH	CNT[7:0]								0000,0000
I2CCFG	I²C 配置寄存器	7EFE80H	ENI2C	MSSL	MSSPEED[5:0]						0000,0000
I2CMSCR	I²C 主机控制寄存器	7EFE81H	EMSI	-	-	-	MSCMD[3:0]				0xxx,0000
I2CMSST	I²C 主机状态寄存器	7EFE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO	00xx,xx10
I2CSLCR	I²C 从机控制寄存器	7EFE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0
I2CSLST	I²C 从机状态寄存器	7EFE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000
I2CSLADR	I²C 从机地址寄存器	7EFE85H	SLADR[6:0]							MA	0000,0000
I2CTXD	I²C 数据发送寄存器	7EFE86H	TXD[7:0]								0000,0000
I2CRXD	I²C 数据接收寄存器	7EFE87H	RXD[7:0]								0000,0000
I2CMSAUX	I²C 主机辅助控制寄存器	7EFE88H	-	-	-	-	-	-	-	WDTA	xxxx,xxx0
I2CPSCR	I²C 主机时钟分频寄存器	7EFE89H	MSSPEED[13:6]								0000,0000

符号	描述	地址	位地址与符号								复位值
SPI_CLKDIV	SPI 时钟分配器	7EFE90H	DIV[7:0]								0000,0000
PWMA_CLKDIV	PWMA 时钟分配器	7EFE91H	DIV[7:0]								0000,0000
PWMB_CLKDIV	PWMB 时钟分配器	7EFE92H	DIV[7:0]								0000,0000
TFPU_CLKDIV	TFPU 时钟分配器	7EFE93H	DIV[7:0]								0000,0000
I2S_CLKDIV	I2S 时钟分配器	7EFE94H	DIV[7:0]								0000,0000
RSTFLAG	复位标志寄存器	7EFE99H	-	-	-	LVDRSTF	WDTRSTF	SWRSTF	ROMOVF	EXRSTF	xxx1,0100
RSTCR0	复位控制寄存器 0	7EFE9AH	-	-	-	-	RSTTM34	TSTTM2	-	RSTTM01	xxxx,00x0
RSTCR1	复位控制寄存器 1	7EFE9BH	-	-	-	-	RSTUR4	RSTUR3	RSTUR2	RSTUR1	xxxx,0000
RSTCR2	复位控制寄存器 2	7EFE9CH	-	-	-	RSTRTC	RSTPWMB	RSTPWMA	RSTI2C	RSTSPI	xxx0,0000
RSTCR3	复位控制寄存器 3	7EFE9DH	RSTFPU	RSTDMA	RSTLCM	RSTLCD	RSTLED	RSTTKS	RSTCMP	RSTADC	0000,0000
RSTCR4	复位控制寄存器 4	7EFE9EH	-	-	-	-	-	-	-	RSTMDU	xxxx,xxx0
RSTCR5	复位控制寄存器 5	7EFE9FH	-	-	-	-	-	-	-	-	xxxx,xxxx
TM0PS	定时器 0 时钟预分频寄存器	7EFEA0H	PS[7:0]								0000,0000
TM1PS	定时器 1 时钟预分频寄存器	7EFEA1H	PS[7:0]								0000,0000
TM2PS	定时器 2 时钟预分频寄存器	7EFEA2H	PS[7:0]								0000,0000
TM3PS	定时器 3 时钟预分频寄存器	7EFEA3H	PS[7:0]								0000,0000
TM4PS	定时器 4 时钟预分频寄存器	7EFEA4H	PS[7:0]								0000,0000
ADCTIM	ADC 时序控制寄存器	7EFEA8H	CSSETUP	CSHOLD[1:0]		SMPDUTY[4:0]					0010,1010
ADCEXCFG	ADC 扩展配置寄存器	7EFEADH	ADCETR_PS[1:0]		ADCETRS [1:0]		-	CVTIMESEL[2:0]			0000,x000
CMPEXCFG	比较器扩展配置寄存器	7EFEAEH	CHYS[1:0]			-	-	CMPNS	CMPPS[1:0]		00xx,x000
PWMA_ETRPS	PWMA 的 ETR 选择寄存器	7EFEB0H						BRKAPS	ETRAPS[1:0]		xxxx,x000
PWMA_ENO	PWMA 输出使能控制	7EFEB1H	ENO4N	ENO4P	ENO3N	ENO3P	ENO2N	ENO2P	ENO1N	ENO1P	0000,0000
PWMA_PS	PWMA 输出脚选择寄存器	7EFEB2H	C4PS[1:0]		C3PS[1:0]		C2PS[1:0]		C1PS[1:0]		0000,0000
PWMA_IOAUX	PWMA 辅助寄存器	7EFEB3H	AUX4N	AUX4P	AUX3N	AUX3P	AUX2N	AUX2P	AUX1N	AUX1P	0000,0000
PWMB_ETRPS	PWMB 的 ETR 选择寄存器	7EFEB4H						BRKBPS	ETRBPS[1:0]		xxxx,x000
PWMB_ENO	PWMB 输出使能控制	7EFEB5H	-	ENO8P	-	ENO7P	-	ENO6P	-	ENO5P	x0x0,x0x0
PWMB_PS	PWMB 输出脚选择寄存器	7EFEB6H	C8PS[1:0]		C7PS[1:0]		C6PS[1:0]		C5PS[1:0]		0000,0000
PWMB_IOAUX	PWMB 辅助寄存器	7EFEB7H	-	AUX8P	-	AUX7P	-	AUX6P	-	AUX5P	x0x0,x0x0
PWMA_PS2	PWMA 输出脚选择寄存器 2	7EFEB8H	-		-		C6PS[1:0]		C5PS[1:0]		xxxx,0000
PWMA_RCRH	PWMA 重复计数器寄存器 2	7EFEB9H	REP1[15:8]								0000,0000
PWMB_RCRH	PWMB 重复计数器寄存器 2	7EFEBAH	REP2[15:8]								0000,0000
PWMA_CR1	PWMA 控制寄存器 1	7EFEC0H	ARPE	CMS[1:0]		DIR	OPM	URS	UDIS	CEN	0000,0000
PWMA_CR2	PWMA 控制寄存器 2	7EFEC1H	-	MMS[2:0]			-	COMS	-	CCPC	x000,x0x0
PWMA_SMCR	PWMA 从模式控制寄存器	7EFEC2H	MSM	TS[2:0]			-	SMS[2:0]			0000,x000
PWMA_ETR	PWMA 外部触发寄存器	7EFEC3H	ETP	ECE	ETPS[1:0]		ETF[3:0]				0000,0000
PWMA_IER	PWMA 中断使能寄存器	7EFEC4H	BIE	TIE	COMIE	CC4IE	CC3IE	CC2IE	CC1IE	UIE	0000,0000
PWMA_SR1	PWMA 状态寄存器 1	7EFEC5H	BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	UIF	0000,0000
PWMA_SR2	PWMA 状态寄存器 2	7EFEC6H	-	-	-	CC4OF	CC3OF	CC2OF	CC1OF	-	xxx0,000x
PWMA_EGR	PWMA 事件发生寄存器	7EFEC7H	BG	TG	COMG	CC4G	CC3G	CC2G	CC1G	UG	0000,0000
PWMA_CCMR1	PWMA 捕获模式寄存器 1	7EFEC8H	OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]		0000,0000
	PWMA 比较模式寄存器 1		IC1F[3:0]				IC1PSC[1:0]		CC1S[1:0]		0000,0000
PWMA_CCMR2	PWMA 捕获模式寄存器 2	7EFEC9H	OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]		0000,0000
	PWMA 比较模式寄存器 2		IC2F[3:0]				IC2PSC[1:0]		CC2S[1:0]		0000,0000

符号	描述	地址	位地址与符号								复位值
PWMA_CCMR3	PWMA 捕获模式寄存器 3	7EFECAH	OC3CE	OC3M[2:0]			OC3PE	OC3FE	CC3S[1:0]		0000,0000
	PWMA 比较模式寄存器 3		IC3F[3:0]				IC3PSC[1:0]		CC3S[1:0]		0000,0000
PWMA_CCMR4	PWMA 捕获模式寄存器 4	7EFECBH	OC4CE	OC4M[2:0]			OC4PE	OC4FE	CC4S[1:0]		0000,0000
	PWMA 比较模式寄存器 4		IC4F[3:0]				IC4PSC[1:0]		CC4S[1:0]		0000,0000
PWMA_CCER1	PWMA 捕获比较使能寄存器 1	7EFECCH	CC2NP	CC2NE	CC2P	CC2E	CC1NP	CC1NE	CC1P	CC1E	0000,0000
PWMA_CCER2	PWMA 捕获比较使能寄存器 2	7EFECDH	CC4NP	CC4NE	CC4P	CC4E	CC3NP	CC3NE	CC3P	CC3E	0000,0000
PWMA_CNTRH	PWMA 计数器高字节	7EFECEH	CNT[15:8]								0000,0000
PWMA_CNTRL	PWMA 计数器低字节	7EFECFH	CNT[7:0]								0000,0000
PWMA_PSCRH	PWMA 预分频高字节	7EFED0H	PSC[15:8]								0000,0000
PWMA_PSCRL	PWMA 预分频低字节	7EFED1H	PSC[7:0]								0000,0000
PWMA_ARRH	PWMA 自动重装寄存器高字节	7EFED2H	ARR[15:8]								0000,0000
PWMA_ARRL	PWMA 自动重装寄存器低字节	7EFED3H	ARR[7:0]								0000,0000
PWMA_RCR	PWMA 重复计数器寄存器	7EFED4H	REP[7:0]								0000,0000
PWMA_CCR1H	PWMA 比较捕获寄存器 1 高位	7EFED5H	CCR1[15:8]								0000,0000
PWMA_CCR1L	PWMA 比较捕获寄存器 1 低位	7EFED6H	CCR1[7:0]								0000,0000
PWMA_CCR2H	PWMA 比较捕获寄存器 2 高位	7EFED7H	CCR2[15:8]								0000,0000
PWMA_CCR2L	PWMA 比较捕获寄存器 2 低位	7EFED8H	CCR2[7:0]								0000,0000
PWMA_CCR3H	PWMA 比较捕获寄存器 3 高位	7EFED9H	CCR3[15:8]								0000,0000
PWMA_CCR3L	PWMA 比较捕获寄存器 3 低位	7EFEDAH	CCR3[7:0]								0000,0000
PWMA_CCR4H	PWMA 比较捕获寄存器 4 高位	7EFEDBH	CCR4[15:8]								0000,0000
PWMA_CCR4L	PWMA 比较捕获寄存器 4 低位	7EFEDCH	CCR4[7:0]								0000,0000
PWMA_BKR	PWMA 刹车寄存器	7EFEDDH	MOE	AOE	BKP	BKE	OSSR	OSSI	LOCK[1:0]		0000,000x
PWMA_DTR	PWMA 死区控制寄存器	7EFEDEH	DTG[7:0]								0000,0000
PWMA_OISR	PWMA 输出空闲状态寄存器	7EFEDFH	OIS4N	OIS4	OIS3N	OIS3	OIS2N	OIS2	OIS1N	OIS1	0000,0000
PWMB_CR1	PWMB 控制寄存器 1	7EFEE0H	ARPE	CMS[1:0]		DIR	OPM	URS	UDIS	CEN	0000,0000
PWMB_CR2	PWMB 控制寄存器 2	7EFEE1H	-	MMS[2:0]			-	COMS	-	CCPC	x000,x0x0
PWMB_SMCR	PWMB 从模式控制寄存器	7EFEE2H	MSM	TS[2:0]			-	SMS[2:0]			0000,x000
PWMB_ETR	PWMB 外部触发寄存器	7EFEE3H	ETP	ECE	ETPS[1:0]		ETF[3:0]				0000,0000
PWMB_IER	PWMB 中断使能寄存器	7EFEE4H	BIE	TIE	COMIE	CC8IE	CC7IE	CC6IE	CC5IE	UIE	0000,0000
PWMB_SR1	PWMB 状态寄存器 1	7EFEE5H	BIF	TIF	COMIF	CC8IF	CC7IF	CC6IF	CC5IF	UIF	0000,0000
PWMB_SR2	PWMB 状态寄存器 2	7EFEE6H	-	-	-	CC8OF	CC7OF	CC6OF	CC5OF	-	xxx0,000x
PWMB_EGR	PWMB 事件发生寄存器	7EFEE7H	BG	TG	COMG	CC8G	CC7G	CC6G	CC5G	UG	0000,0000
PWMB_CCMR1	PWMB 捕获模式寄存器 1	7EFEE8H	OC5CE	OC5M[2:0]			OC5PE	OC5FE	CC5S[1:0]		0000,0000
	PWMB 比较模式寄存器 1		IC5F[3:0]				IC5PSC[1:0]		CC5S[1:0]		0000,0000
PWMB_CCMR2	PWMB 捕获模式寄存器 2	7EFEE9H	OC6CE	OC6M[2:0]			OC6PE	OC6FE	CC6S[1:0]		0000,0000
	PWMB 比较模式寄存器 2		IC6F[3:0]				IC6PSC[1:0]		CC6S[1:0]		0000,0000
PWMB_CCMR3	PWMB 捕获模式寄存器 3	7EFEEAH	OC7CE	OC7M[2:0]			OC7PE	OC7FE	CC7S[1:0]		0000,0000
	PWMB 比较模式寄存器 3		IC7F[3:0]				IC7PSC[1:0]		CC7S[1:0]		0000,0000
PWMB_CCMR4	PWMB 捕获模式寄存器 4	7EFEEBH	OC8CE	OC8M[2:0]			OC8PE	OC8FE	CC8S[1:0]		0000,0000
	PWMB 比较模式寄存器 4		IC8F[3:0]				IC8PSC[1:0]		CC8S[1:0]		0000,0000
PWMB_CCER1	PWMB 捕获比较使能寄存器 1	7EFEECH	-	-	CC6P	CC6E	-	-	CC5P	CC5E	xx00,xx00
PWMB_CCER2	PWMB 捕获比较使能寄存器 2	7EFEEDH	-	-	CC8P	CC8E	-	-	CC7P	CC7E	xx00,xx00
PWMB_CNTRH	PWMB 计数器高字节	7EFEEEH	CNT[15:8]								0000,0000

符号	描述	地址	位地址与符号								复位值
PWMB_CNTRL	PWMB 计数器低字节	7EFEEFH	CNT[7:0]								0000,0000
PWMB_PSCRH	PWMB 预分频高字节	7EFEF0H	PSC[15:8]								0000,0000
PWMB_PSCRL	PWMB 预分频低字节	7EFEF1H	PSC[7:0]								0000,0000
PWMB_ARRH	PWMB 自动重装寄存器高字节	7EFEF2H	ARR[15:8]								0000,0000
PWMB_ARRL	PWMB 自动重装寄存器低字节	7EFEF3H	ARR[7:0]								0000,0000
PWMB_RCR	PWMB 重复计数器寄存器	7EFEF4H	REP[7:0]								0000,0000
PWMB_CCR5H	PWMB 比较捕获寄存器 5 高位	7EFEF5H	CCR1[15:8]								0000,0000
PWMB_CCR5L	PWMB 比较捕获寄存器 5 低位	7EFEF6H	CCR1[7:0]								0000,0000
PWMB_CCR6H	PWMB 比较捕获寄存器 6 高位	7EFEF7H	CCR2[15:8]								0000,0000
PWMB_CCR6L	PWMB 比较捕获寄存器 6 低位	7EFEF8H	CCR2[7:0]								0000,0000
PWMB_CCR7H	PWMB 比较捕获寄存器 7 高位	7EFEF9H	CCR3[15:8]								0000,0000
PWMB_CCR7L	PWMB 比较捕获寄存器 7 低位	7EFEFAH	CCR3[7:0]								0000,0000
PWMB_CCR8H	PWMB 比较捕获寄存器 8 高位	7EFEFBH	CCR4[15:8]								0000,0000
PWMB_CCR8L	PWMB 比较捕获寄存器 8 低位	7EFEFCH	CCR4[7:0]								0000,0000
PWMB_BKR	PWMB 刹车寄存器	7EFEFDH	MOE	AOE	BKP	BKE	OSSR	OSSI	LOCK[1:0]	-	0000,000x
PWMB_DTR	PWMB 死区控制寄存器	7EFEFEH	DTG[7:0]								0000,0000
PWMB_OISR	PWMB 输出空闲状态寄存器	7EFEFFH	-	OIS8	-	OIS7	-	OIS6	-	OIS5	x0x0,x0x0

13.5 扩展特殊功能寄存器列表（XFR：0x7EFD00-0x7EFDFF）

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
P0INTE	P0 口中断使能寄存器	7EFD00H	P0INTE[7:0]								0000,0000
P1INTE	P1 口中断使能寄存器	7EFD01H	P1INTE[7:0]								0000,0000
P2INTE	P2 口中断使能寄存器	7EFD02H	P2INTE[7:0]								0000,0000
P3INTE	P3 口中断使能寄存器	7EFD03H	P3INTE[7:0]								0000,0000
P4INTE	P4 口中断使能寄存器	7EFD04H	P4INTE[7:0]								0000,0000
P5INTE	P5 口中断使能寄存器	7EFD05H	P5INTE[7:0]								0000,0000
P6INTE	P6 口中断使能寄存器	7EFD06H	P6INTE[7:0]								0000,0000
P7INTE	P7 口中断使能寄存器	7EFD07H	P7INTE[7:0]								0000,0000
P0INTF	P0 口中断标志寄存器	7EFD10H	P0INTF[7:0]								0000,0000
P1INTF	P1 口中断标志寄存器	7EFD11H	P1INTF[7:0]								0000,0000
P2INTF	P2 口中断标志寄存器	7EFD12H	P2INTF[7:0]								0000,0000
P3INTF	P3 口中断标志寄存器	7EFD13H	P3INTF[7:0]								0000,0000
P4INTF	P4 口中断标志寄存器	7EFD14H	P4INTF[7:0]								0000,0000
P5INTF	P5 口中断标志寄存器	7EFD15H	P5INTF[7:0]								0000,0000
P6INTF	P6 口中断标志寄存器	7EFD16H	P6INTF[7:0]								0000,0000
P7INTF	P7 口中断标志寄存器	7EFD17H	P7INTF[7:0]								0000,0000
P0IM0	P0 口中断模式寄存器 0	7EFD20H	P0IM0[7:0]								0000,0000
P1IM0	P1 口中断模式寄存器 0	7EFD21H	P1IM0[7:0]								0000,0000
P2IM0	P2 口中断模式寄存器 0	7EFD22H	P2IM0[7:0]								0000,0000
P3IM0	P3 口中断模式寄存器 0	7EFD23H	P3IM0[7:0]								0000,0000

符号	描述	地址	位地址与符号								复位值
P4IM0	P4 口中断模式寄存器 0	7EFD24H	P4IM0[7:0]								0000,0000
P5IM0	P5 口中断模式寄存器 0	7EFD25H	P5IM0[7:0]								0000,0000
P6IM0	P6 口中断模式寄存器 0	7EFD26H	P6IM0[7:0]								0000,0000
P7IM0	P7 口中断模式寄存器 0	7EFD27H	P7IM0[7:0]								0000,0000
P0IM1	P0 口中断模式寄存器 1	7EFD30H	P0IM1[7:0]								0000,0000
P1IM1	P1 口中断模式寄存器 1	7EFD31H	P1IM1[7:0]								0000,0000
P2IM1	P2 口中断模式寄存器 1	7EFD32H	P2IM1[7:0]								0000,0000
P3IM1	P3 口中断模式寄存器 1	7EFD33H	P3IM1[7:0]								0000,0000
P4IM1	P4 口中断模式寄存器 1	7EFD34H	P4IM1[7:0]								0000,0000
P5IM1	P5 口中断模式寄存器 1	7EFD35H	P5IM1[7:0]								0000,0000
P6IM1	P6 口中断模式寄存器 1	7EFD36H	P6IM1[7:0]								0000,0000
P7IM1	P7 口中断模式寄存器 1	7EFD37H	P7IM1[7:0]								0000,0000
P0WKUE	P0 口中断唤醒使能	7EFD40H	P0WKUE[7:0]								0000,0000
P1WKUE	P1 口中断唤醒使能	7EFD41H	P1WKUE[7:0]								0000,0000
P2WKUE	P2 口中断唤醒使能	7EFD42H	P2WKUE[7:0]								0000,0000
P3WKUE	P3 口中断唤醒使能	7EFD43H	P3WKUE[7:0]								0000,0000
P4WKUE	P4 口中断唤醒使能	7EFD44H	P4WKUE[7:0]								0000,0000
P5WKUE	P5 口中断唤醒使能	7EFD45H	P5WKUE[7:0]								0000,0000
P6WKUE	P6 口中断唤醒使能	7EFD46H	P6WKUE[7:0]								0000,0000
P7WKUE	P7 口中断唤醒使能	7EFD47H	P7WKUE[7:0]								0000,0000
CCAPM2	PCA 模块 2 模式控制寄存器	7EFD50H	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2	x000,0000
CCAP2L	PCA 模块 2 低字节	7EFD51H									0000,0000
CCAP2H	PCA 模块 2 高字节	7EFD52H									0000,0000
PCA_PWM2	PCA2 的 PWM 模式寄存器	7EFD53H	EBS2[1:0]		XCCAP2H[1:0]		XCCAP2L[1:0]		EPC2H	EPC2L	0000,0000
CCAPM3	PCA 模块 3 模式控制寄存器	7EFD54H	-	ECOM3	CCAPP3	CCAPN3	MAT3	TOG3	PWM3	ECCF3	x000,0000
CCAP3L	PCA 模块 3 低字节	7EFD55H									0000,0000
CCAP3H	PCA 模块 3 高字节	7EFD56H									0000,0000
PCA_PWM3	PCA3 的 PWM 模式寄存器	7EFD57H	EBS3[1:0]		XCCAP3H[1:0]		XCCAP3L[1:0]		EPC3H	EPC3L	0000,0000
CCAPM0	PCA 模块 0 模式控制寄存器	7EFD58H	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0	x000,0000
CCAP0L	PCA 模块 0 低字节	7EFD59H									0000,0000
CCAP0H	PCA 模块 0 高字节	7EFD5AH									0000,0000
PCA_PWM0	PCA0 的 PWM 模式寄存器	7EFD5BH	EBS0[1:0]		XCCAP0H[1:0]		XCCAP0L[1:0]		EPC0H	EPC0L	0000,0000
CCAPM1	PCA 模块 1 模式控制寄存器	7EFD5CH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1	x000,0000
CCAP1L	PCA 模块 1 低字节	7EFD5DH									0000,0000
CCAP1H	PCA 模块 1 高字节	7EFD5EH									0000,0000
PCA_PWM1	PCA1 的 PWM 模式寄存器	7EFD5FH	EBS1[1:0]		XCCAP1H[1:0]		XCCAP1L[1:0]		EPC1H	EPC1L	0000,0000
PINIPL	I/O 口中断优先级低寄存器	7EFD60H	P7IP	P6IP	P5IP	P4IP	P3IP	P2IP	P1IP	P0IP	0000,0000
PINIPH	I/O 口中断优先级高寄存器	7EFD61H	P7IPH	P6IPH	P5IPH	P4IPH	P3IPH	P2IPH	P1IPH	P0IPH	0000,0000
CCON	PCA 控制寄存器	7EFD64H	CF	CR	-	-	CCF3	CCF2	CCF1	CCF0	00xx,x000
CL	PCA 计数器低字节	7EFD65H									0000,0000
CH	PCA 计数器高字节	7EFD66H									0000,0000
CMOD	PCA 模式寄存器	7EFD67H	CIDL	CCP_S[1:0]		CPS[3:0]				ECF	0000,0000
UR1TOCR	串口 1 接收超时控制寄存器	7EFD70H	ENTO	ENTOI	SCALE	-	-	-	-	-	000x,xxxx

符号	描述	地址	位地址与符号								复位值
UR1TOSR	串口 1 接收超时状态寄存器	7EFD71H	CTOIF	-	-	-	-	-	-	TOIF	0xxx,xxx0
UR1TOTH	串口 1 接收超时长度寄存器	7EFD72H	TM[15:8]								0000,0000
UR1TOTL	串口 1 接收超时长度寄存器	7EFD73H	TM[7:0]								0000,0000
UR2TOCR	串口 2 接收超时控制寄存器	7EFD74H	ENTO	ENTOI	SCALE	-	-	-	-	-	000x,xxxx
UR2TOSR	串口 2 接收超时状态寄存器	7EFD75H	CTOIF	-	-	-	-	-	-	TOIF	0xxx,xxx0
UR2TOTH	串口 2 接收超时长度寄存器	7EFD76H	TM[15:8]								0000,0000
UR2TOTL	串口 2 接收超时长度寄存器	7EFD77H	TM[7:0]								0000,0000
UR3TOCR	串口 3 接收超时控制寄存器	7EFD78H	ENTO	ENTOI	SCALE	-	-	-	-	-	000x,xxxx
UR3TOSR	串口 3 接收超时状态寄存器	7EFD79H	CTOIF	-	-	-	-	-	-	TOIF	0xxx,xxx0
UR3TOTH	串口 3 接收超时长度寄存器	7EFD7AH	TM[15:8]								0000,0000
UR3TOTL	串口 3 接收超时长度寄存器	7EFD7BH	TM[7:0]								0000,0000
UR4TOCR	串口 4 接收超时控制寄存器	7EFD7CH	ENTO	ENTOI	SCALE	-	-	-	-	-	000x,xxxx
UR4TOSR	串口 4 接收超时状态寄存器	7EFD7DH	CTOIF	-	-	-	-	-	-	TOIF	0xxx,xxx0
UR4TOTH	串口 4 接收超时长度寄存器	7EFD7EH	TM[15:8]								0000,0000
UR4TOTL	串口 4 接收超时长度寄存器	7EFD7FH	TM[7:0]								0000,0000
SPITOCR	SPI 从机超时控制寄存器	7EFD80H	ENTO	ENTOI	SCALE	-	-	-	-	-	000x,xxxx
SPITOSR	SPI 从机超时状态寄存器	7EFD81H	CTOIF	-	-	-	-	-	-	TOIF	0xxx,xxx0
SPITOTH	SPI 从机超时长度寄存器	7EFD82H	TM[15:8]								0000,0000
SPITOTL	SPI 从机超时长度寄存器	7EFD83H	TM[7:0]								0000,0000
I2CTOCR	I2C 从机超时控制寄存器	7EFD84H	ENTO	ENTOI	SCALE	-	-	-	-	-	000x,xxxx
I2CTOSR	I2C 从机超时状态寄存器	7EFD85H	CTOIF	-	-	-	-	-	-	TOIF	0xxx,xxx0
I2CTOTH	I2C 从机超时长度寄存器	7EFD86H	TM[15:8]								0000,0000
I2CTOTL	I2C 从机超时长度寄存器	7EFD87H	TM[7:0]								0000,0000
UR1TOTE	串口 1 接收超时长度寄存器	7EFD88H	TM[23:16]								0000,0000
UR2TOTE	串口 2 接收超时长度寄存器	7EFD89H	TM[23:16]								0000,0000
UR3TOTE	串口 3 接收超时长度寄存器	7EFD8AH	TM[23:16]								0000,0000
UR4TOTE	串口 4 接收超时长度寄存器	7EFD8BH	TM[23:16]								0000,0000
SPITOTE	SPI 接收超时长度寄存器	7EFD8CH	TM[23:16]								0000,0000
I2CTOTE	I2C 接收超时长度寄存器	7EFD8DH	TM[23:16]								0000,0000
I2SCR	I2S 控制寄存器	7EFD98H	TXEIE	RXNEIE	ERRIE	FRF	-	-	TXDMAEN	RXDMAEN	0000,xx00
I2SSR	I2S 状态寄存器	7EFD99H	-	FRE	BUY	OVR	UDR	CHSID	TXE	RXNE	x000,0000
I2SDRH	I2S 数据寄存器高字节	7EFD9AH	DR[15:8]								0000,0000
I2SDRL	I2S 数据寄存器低字节	7EFD9BH	DR[7:0]								0000,0000
I2SPRH	I2S 分频寄存器高字节	7EFD9CH	-	-	-	-	-	-	MCKOE	ODD	xxxx,xx00
I2SPRL	I2S 分频寄存器低字节	7EFD9DH	DIV[7:0]								0000,0000
I2SCFGH	I2S 配置寄存器高字节	7EFD9EH	-	-	-	-	-	I2SE	I2SCFG[1:0]		xxxx,x000
I2SCFGL	I2S 配置寄存器低字节	7EFD9FH	PCMSYNC	-	STD[1:0]		CKPOL	DATLEN[1:0]		CHLEN	0x00,0000
I2SMD	I2S 从模式控制寄存器	7EFDA0H	MODE[7:0]								0000,0000
I2SMCKDIV	I2S 主时钟分配器	7EFDA1H	DIV[7:0]								0000,0000
CRECR	CRE 控制寄存器	7EFDA8H	ENCRE	MONO	UPT[1:0]		CREHF	CREINC	CREDEC	CRERDY	0000,0000
CRECNTH	CRE 校准目标寄存器	7EFDA9H	CNT[15:8]								0000,0000
CRECNTL	CRE 校准目标寄存器	7EFDAAH	CNT[7:0]								0000,0000
CRERES	CRE 分辨率控制寄存器	7EFDABH	RES[7:0]								0000,0000

符号	描述	地址	位地址与符号								复位值
S2CFG	串口 2 配置寄存器	7EFDB4H	-	S2MOD0	S2M0x6	-	-	-	-	W1	000x,xxx0
S2ADDR.	串口 2 从机地址寄存器	7EFDB5H									0000,0000
S2ADEN	串口 2 从机地址屏蔽寄存器	7EFDB6H									0000,0000
USARTCR1	串口 1 控制寄存器 1	7EFDC0H	LINEN	DORD	CLKEN	SPMOD	SPIEN	SPSLV	CPOL	CPHA	0000,0000
USARTCR2	串口 1 控制寄存器 2	7EFDC1H	IREN	IRLP	SCEN	NACK	HDSEL	PCEN	PS	PE	0000,0000
USARTCR3	串口 1 控制寄存器 3	7EFDC2H	IrDA_LPBAUD[7:0]								0000,0111
USARTCR4	串口 1 控制寄存器 4	7EFDC3H	-	-	-	-	SCCKS[1:0]		SPICKS[1:0]		xxxx,0000
USARTCR5	串口 1 控制寄存器 5	7EFDC4H	BRKDET	HDRER	SLVEN	ASYNC	TXCF	SENDBK	HDRDET	SYNC	0000,0000
USARTGTR	串口 1 保护时间寄存器	7EFDC5H									0000,0000
USARTBRH	串口 1 波特率寄存器	7EFDC6H	USARTBR[15:8]								0000,0000
USARTBRL	串口 1 波特率寄存器	7EFDC7H	USARTBR[7:0]								0000,0000
USART2CR1	串口 2 控制寄存器 1	7EFDC8H	LINEN	DORD	CLKEN	SPMOD	SPIEN	SPSLV	CPOL	CPHA	0000,0000
USART2CR2	串口 2 控制寄存器 2	7EFDC9H	IREN	IRLP	SCEN	NACK	HDSEL	PCEN	PS	PE	0000,0000
USART2CR3	串口 2 控制寄存器 3	7EFDCAH	IrDA_LPBAUD[7:0]								0000,0000
USART2CR4	串口 2 控制寄存器 4	7EFDCBH	-	-	-	-	SCCKS[1:0]		SPICKS[1:0]		xxxx,0000
USART2CR5	串口 2 控制寄存器 5	7EFDCCH	BRKDET	HDRER	SLVEN	ASYNC	TXCF	SENDBK	HDRDET	SYNCAN	0000,0000
USART2GTR	串口 2 保护时间寄存器	7EFDCDH									0000,0000
USART2BRH	串口 2 波特率寄存器	7EFDCEH	USART2BR[15:8]								0000,0000
USART2BRL	串口 2 波特率寄存器	7EFDCFH	USART2BR[7:0]								0000,0000
CHIPID0	硬件数字 ID00	7EFDE0H	全球唯一 ID 号（第 0 字节）								nnnn,nnnn
CHIPID1	硬件数字 ID01	7EFDE1H	全球唯一 ID 号（第 1 字节）								nnnn,nnnn
CHIPID2	硬件数字 ID02	7EFDE2H	全球唯一 ID 号（第 2 字节）								nnnn,nnnn
CHIPID3	硬件数字 ID03	7EFDE3H	全球唯一 ID 号（第 3 字节）								nnnn,nnnn
CHIPID4	硬件数字 ID04	7EFDE4H	全球唯一 ID 号（第 4 字节）								nnnn,nnnn
CHIPID5	硬件数字 ID05	7EFDE5H	全球唯一 ID 号（第 5 字节）								nnnn,nnnn
CHIPID6	硬件数字 ID06	7EFDE6H	全球唯一 ID 号（第 6 字节）								nnnn,nnnn
CHIPID7	硬件数字 ID07	7EFDE7H	内部 1.19V 参考信号源-BGV（高字节）								nnnn,nnnn
CHIPID8	硬件数字 ID08	7EFDE8H	内部 1.19V 参考信号源-BGV（低字节）								nnnn,nnnn
CHIPID9	硬件数字 ID09	7EFDE9H	32K 掉电唤醒定时器的频率（高字节）								nnnn,nnnn
CHIPID10	硬件数字 ID10	7EFDEAH	32K 掉电唤醒定时器的频率（低字节）								nnnn,nnnn
CHIPID11	硬件数字 ID11	7EFDEBH	22.1184MHz 的 IRC 参数（27M 频段）								nnnn,nnnn
CHIPID12	硬件数字 ID12	7EFDECH	24MHz 的 IRC 参数（27M 频段）								nnnn,nnnn
CHIPID13	硬件数字 ID13	7EFDEDH	27MHz 的 IRC 参数（27M 频段）								nnnn,nnnn
CHIPID14	硬件数字 ID14	7EFDEEH	30MHz 的 IRC 参数（27M 频段）								nnnn,nnnn
CHIPID15	硬件数字 ID15	7EFDEFH	33.1776MHz 的 IRC 参数（27M 频段）								nnnn,nnnn
CHIPID16	硬件数字 ID16	7EFDF0H	35MHz 的 IRC 参数（44M 频段）								nnnn,nnnn
CHIPID17	硬件数字 ID17	7EFDF1H	36.864MHz 的 IRC 参数（44M 频段）								nnnn,nnnn
CHIPID18	硬件数字 ID18	7EFDF2H	40MHz 的 IRC 参数（44M 频段）								nnnn,nnnn
CHIPID19	硬件数字 ID19	7EFDF3H	44.2368MHz 的 IRC 参数（44M 频段）								nnnn,nnnn
CHIPID20	硬件数字 ID20	7EFDF4H	48MHz 的 IRC 参数（44M 频段）								nnnn,nnnn
CHIPID21	硬件数字 ID21	7EFDF5H	6M 频段的 VRTRIM 参数								nnnn,nnnn
CHIPID22	硬件数字 ID22	7EFDF6H	10M 频段的 VRTRIM 参数								nnnn,nnnn
CHIPID23	硬件数字 ID23	7EFDF7H	27M 频段的 VRTRIM 参数								nnnn,nnnn

符号	描述	地址	位地址与符号	复位值
CHIPID24	硬件数字 ID24	7EFDF8H	44M 频段的 VRTRIM 参数	nnnn,nnnn
CHIPID25	硬件数字 ID25	7EFDF9H	00H	nnnn,nnnn
CHIPID26	硬件数字 ID26	7EFDFAH	用户程序空间结束地址（高字节）	nnnn,nnnn
CHIPID27	硬件数字 ID27	7EFDFBH	芯片测试时间（年）	nnnn,nnnn
CHIPID28	硬件数字 ID28	7EFDFCH	芯片测试时间（月）	nnnn,nnnn
CHIPID29	硬件数字 ID29	7EFDFDH	芯片测试时间（日）	nnnn,nnnn
CHIPID30	硬件数字 ID30	7EFDFEH	芯片封装形式编号	nnnn,nnnn
CHIPID31	硬件数字 ID31	7EFDFFH	5AH	nnnn,nnnn

13.6　扩展特殊功能寄存器列表（XFR：0x7EFB00-0x7EFBFF）

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPIDX0	扩展硬件数字 ID00	7EFBD0H	33.8688MHz 的 IRC 参数								nnnn,nnnn
CHIPIDX1	扩展硬件数字 ID01	7EFBD1H	33.8688MHz 的频段参数								nnnn,nnnn
CHIPIDX2	扩展硬件数字 ID02	7EFBD2H	40.96MHz 的 IRC 参数								nnnn,nnnn
CHIPIDX3	扩展硬件数字 ID03	7EFBD3H	40.96MHz 的频段参数								nnnn,nnnn
CHIPIDX4	扩展硬件数字 ID04	7EFBD4H	42MHz 的 IRC 参数								nnnn,nnnn
CHIPIDX5	扩展硬件数字 ID05	7EFBD5H	42MHz 的频段参数								nnnn,nnnn
CHIPIDX6	扩展硬件数字 ID06	7EFBD6H	保留								nnnn,nnnn
CHIPIDX7	扩展硬件数字 ID07	7EFBD7H	保留								nnnn,nnnn
CHIPIDX8	扩展硬件数字 ID08	7EFBD8H	保留								nnnn,nnnn
CHIPIDX9	扩展硬件数字 ID09	7EFBD9H	保留								nnnn,nnnn
CHIPIDX10	扩展硬件数字 ID10	7EFBDAH	保留								nnnn,nnnn
CHIPIDX11	扩展硬件数字 ID11	7EFBDBH	保留								nnnn,nnnn
CHIPIDX12	扩展硬件数字 ID12	7EFBDCH	保留								nnnn,nnnn
CHIPIDX13	扩展硬件数字 ID13	7EFBDDH	保留								nnnn,nnnn
CHIPIDX14	扩展硬件数字 ID14	7EFBDEH	保留								nnnn,nnnn
CHIPIDX15	扩展硬件数字 ID15	7EFBDFH	保留								nnnn,nnnn
CHIPIDX16	扩展硬件数字 ID16	7EFBE0H	保留								nnnn,nnnn
CHIPIDX17	扩展硬件数字 ID17	7EFBE1H	保留								nnnn,nnnn
CHIPIDX18	扩展硬件数字 ID18	7EFBE2H	保留								nnnn,nnnn
CHIPIDX19	扩展硬件数字 ID19	7EFBE3H	保留								nnnn,nnnn
CHIPIDX20	扩展硬件数字 ID20	7EFBE4H	保留								nnnn,nnnn
CHIPIDX21	扩展硬件数字 ID21	7EFBE5H	保留								nnnn,nnnn
CHIPIDX22	扩展硬件数字 ID22	7EFBE6H	保留								nnnn,nnnn
CHIPIDX23	扩展硬件数字 ID23	7EFBE7H	保留								nnnn,nnnn
CHIPIDX24	扩展硬件数字 ID24	7EFBE8H	保留								nnnn,nnnn
CHIPIDX25	扩展硬件数字 ID25	7EFBE9H	保留								nnnn,nnnn
CHIPIDX26	扩展硬件数字 ID26	7EFBEAH	保留								nnnn,nnnn
CHIPIDX27	扩展硬件数字 ID27	7EFBEBH	保留								nnnn,nnnn
CHIPIDX28	扩展硬件数字 ID28	7EFBECH	保留								nnnn,nnnn
CHIPIDX29	扩展硬件数字 ID29	7EFBEDH	保留								nnnn,nnnn
CHIPIDX30	扩展硬件数字 ID30	7EFBEEH	保留								nnnn,nnnn
CHIPIDX31	扩展硬件数字 ID31	7EFBEFH	5AH								nnnn,nnnn
HSPWMA_CFG	高速 PWMA 配置寄存器	7EFBF0H	-	-	-	EXTN	-	INTEN	ASYNCEN	1	xxx0,0001
HSPWMA_ADR	高速 PWMA 地址寄存器	7EFBF1H	RW/BUSY	ADDR[6:0]							0000,0000
HSPWMA_DAT	高速 PWMA 数据寄存器	7EFBF2H	DATA[7:0]								0000,0000
HSPWMB_CFG	高速 PWMB 配置寄存器	7EFBF4H	-	-	-	EXTN	-	INTEN	ASYNCEN	1	xxx0,0001
HSPWMB_ADR	高速 PWMB 地址寄存器	7EFBF5H	RW/BUSY	ADDR[6:0]							0000,0000
HSPWMB_DAT	高速 PWMB 数据寄存器	7EFBF6H	DATA[7:0]								0000,0000
HSSPI_CFG	高速 SPI 配置寄存器	7EFBF8H	SS_HLD[3:0]				SS_SETUP[3:0]				0011,0011

HSSPI_CFG2	高速 SPI 配置寄存器 2	7EFBF9H	-	IOSW	HSSPIEN	FIFOEN	SS_DACT[3:0]				x000,0011
HSSPI_STA	高速 SPI 状态寄存器	7EFBFAH	-	-	-	-	TXFULL	TXEMPT	RXFULL	RXEMPT	xxxx,0000
HSSPI_PSCR	高速 SPI 时钟分配器	7EFBFBH	DIV[7:0]								0000,0000

13.7　扩展特殊功能寄存器列表（XFR：0x7EFA00-0x7EFAFF）

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
DMA_M2M_CFG	M2M_DMA 配置寄存器	7EFA00H	M2MIE	-	TXACO	RXACO	M2MIP[1:0]		M2MPTY[1:0]		0x00,0000
DMA_M2M_CR	M2M_DMA 控制寄存器	7EFA01H	ENM2M	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_M2M_STA	M2M_DMA 状态寄存器	7EFA02H	-	-	-	-	-	-	-	M2MIF	xxxx,xxx0
DMA_M2M_AMT	M2M_DMA 传输总字节数	7EFA03H	AMT[7:0]								0000,0000
DMA_M2M_DONE	M2M_DMA 传输完成字节数	7EFA04H	DONE[7:0]								0000,0000
DMA_M2M_TXAH	M2M_DMA 发送高地址	7EFA05H	ADR[15:0]								0000,0000
DMA_M2M_TXAL	M2M_DMA 发送低地址	7EFA06H	ADR[7:0]								0000,0000
DMA_M2M_RXAH	M2M_DMA 接收高地址	7EFA07H	ADR[15:0]								0000,0000
DMA_M2M_RXAL	M2M_DMA 接收低地址	7EFA08H	ADR[7:0]								0000,0000
DMA_ADC_CFG	ADC_DMA 配置寄存器	7EFA10H	ADCIE	-	-	-	ADCMIP[1:0]		ADCPTY[1:0]		0xxx,0000
DMA_ADC_CR	ADC_DMA 控制寄存器	7EFA11H	ENADC	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_ADC_STA	ADC_DMA 状态寄存器	7EFA12H	-	-	-	-	-	-	-	ADCIF	xxxx,xxx0
DMA_ADC_AMT	ADC_DMA 传输总字节数	7EFA13H	AMT[7:0]								0000,0000
DMA_ADC_DONE	ADC_DMA 传输完成字节数	7EFA14H	DONE[7:0]								0000,0000
DMA_ADC_RXAH	ADC_DMA 接收高地址	7EFA17H	ADR[15:0]								0000,0000
DMA_ADC_RXAL	ADC_DMA 接收低地址	7EFA18H	ADR[7:0]								0000,0000
DMA_ADC_CFG2	ADC_DMA 配置寄存器 2	7EFA19H	-	-	-	-	CVTIMESEL[3:0]				xxxx,0000
DMA_ADC_CHSW0	ADC_DMA 通道使能	7EFA1AH	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0	0000,0001
DMA_ADC_CHSW1	ADC_DMA 通道使能	7EFA1BH	CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8	1000,0000
DMA_ADC_ITVH	ADC_DMA 时间间隔寄存器	7EFA1EH	ITV[15:8]								0000,0000
DMA_ADC_ITVL	ADC_DMA 时间间隔寄存器	7EFA1FH	ITV[7:0]								0000,0000
DMA_SPI_CFG	SPI_DMA 配置寄存器	7EFA20H	SPIIE	ACT_TX	ACT_RX	-	SPIIP[1:0]		SPIPTY[1:0]		000x,0000
DMA_SPI_CR	SPI_DMA 控制寄存器	7EFA21H	ENSPI	TRIG_M	TRIG_S	-	-	-	-	CLRFIFO	000x,xxx0
DMA_SPI_STA	SPI_DMA 状态寄存器	7EFA22H	-	-	-	-	-	TXOVW	RXLOSS	SPIIF	xxxx,x000
DMA_SPI_AMT	SPI_DMA 传输总字节数	7EFA23H	AMT[7:0]								0000,0000
DMA_SPI_DONE	SPI_DMA 传输完成字节数	7EFA24H	DONE[7:0]								0000,0000
DMA_SPI_TXAH	SPI_DMA 发送高地址	7EFA25H	ADR[15:0]								0000,0000
DMA_SPI_TXAL	SPI_DMA 发送低地址	7EFA26H	ADR[7:0]								0000,0000
DMA_SPI_RXAH	SPI_DMA 接收高地址	7EFA27H	ADR[15:0]								0000,0000
DMA_SPI_RXAL	SPI_DMA 接收低地址	7EFA28H	ADR[7:0]								0000,0000
DMA_SPI_CFG2	SPI_DMA 配置寄存器 2	7EFA29H	-	-	-	-	-	WRPSS	SSS[1:0]		xxxx,x000
DMA_SPI_ITVH	SPI_DMA 时间间隔寄存器	7EFA2EH	ITV[15:8]								0000,0000
DMA_SPI_ITVL	SPI_DMA 时间间隔寄存器	7EFA2FH	ITV[7:0]								0000,0000
DMA_UR1T_CFG	UR1T_DMA 配置寄存器	7EFA30H	UR1TIE	-	-	-	UR1TIP[1:0]		UR1TPTY[1:0]		0xxx,0000
DMA_UR1T_CR	UR1T_DMA 控制寄存器	7EFA31H	ENUR1T	TRIG	-	-	-	-	-	-	00xx,xxxx

DMA_UR1T_STA	UR1T_DMA 状态寄存器	7EFA32H	-	-	-	-	-	TXOVW	-	UR1TIF	xxxx,x0x0
DMA_UR1T_AMT	UR1T_DMA 传输总字节数	7EFA33H	AMT[7:0]								0000,0000
DMA_UR1T_DONE	UR1T_DMA 传输完成字节数	7EFA34H	DONE[7:0]								0000,0000
DMA_UR1T_TXAH	UR1T_DMA 发送高地址	7EFA35H	ADR[15:0]								0000,0000
DMA_UR1T_TXAL	UR1T_DMA 发送低地址	7EFA36H	ADR[7:0]								0000,0000
DMA_UR1R_CFG	UR1R_DMA 配置寄存器	7EFA38H	UR1RIE	-	-	-	UR1RIP[1:0]		UR1RPTY[1:0]		0xxx,0000
DMA_UR1R_CR	UR1R_DMA 控制寄存器	7EFA39H	ENUR1R	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0
DMA_UR1R_STA	UR1R_DMA 状态寄存器	7EFA3AH	-	-	-	-	-	-	RXLOSS	UR1RIF	xxxx,xx00
DMA_UR1R_AMT	UR1R_DMA 传输总字节数	7EFA3BH	AMT[7:0]								0000,0000
DMA_UR1R_DONE	UR1R_DMA 传输完成字节数	7EFA3CH	DONE[7:0]								0000,0000
DMA_UR1R_RXAH	UR1R_DMA 接收高地址	7EFA3DH	ADR[15:0]								0000,0000
DMA_UR1R_RXAL	UR1R_DMA 接收低地址	7EFA3EH	ADR[7:0]								0000,0000
DMA_UR2T_CFG	UR2T_DMA 配置寄存器	7EFA40H	UR2TIE	-	-	-	UR2TIP[1:0]		UR2TPTY[1:0]		0xxx,0000
DMA_UR2T_CR	UR2T_DMA 控制寄存器	7EFA41H	ENUR2T	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_UR2T_STA	UR2T_DMA 状态寄存器	7EFA42H	-	-	-	-	-	TXOVW	-	UR2TIF	xxxx,x0x0
DMA_UR2T_AMT	UR2T_DMA 传输总字节数	7EFA43H	AMT[7:0]								0000,0000
DMA_UR2T_DONE	UR2T_DMA 传输完成字节数	7EFA44H	DONE[7:0]								0000,0000
DMA_UR2T_TXAH	UR2T_DMA 发送高地址	7EFA45H	ADR[15:0]								0000,0000
DMA_UR2T_TXAL	UR2T_DMA 发送低地址	7EFA46H	ADR[7:0]								0000,0000
DMA_UR2R_CFG	UR2R_DMA 配置寄存器	7EFA48H	UR2RIE	-	-	-	UR2RIP[1:0]		UR2RPTY[1:0]		0xxx,0000
DMA_UR2R_CR	UR2R_DMA 控制寄存器	7EFA49H	ENUR2R	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0
DMA_UR2R_STA	UR2R_DMA 状态寄存器	7EFA4AH	-	-	-	-	-	-	RXLOSS	UR2RIF	xxxx,xx00
DMA_UR2R_AMT	UR2R_DMA 传输总字节数	7EFA4BH	AMT[7:0]								0000,0000
DMA_UR2R_DONE	UR2R_DMA 传输完成字节数	7EFA4CH	DONE[7:0]								0000,0000
DMA_UR2R_RXAH	UR2R_DMA 接收高地址	7EFA4DH	ADR[15:0]								0000,0000
DMA_UR2R_RXAL	UR2R_DMA 接收低地址	7EFA4EH	ADR[7:0]								0000,0000
DMA_UR3T_CFG	UR3T_DMA 配置寄存器	7EFA50H	UR3TIE	-	-	-	UR3TIP[1:0]		UR3TPTY[1:0]		0xxx,0000
DMA_UR3T_CR	UR3T_DMA 控制寄存器	7EFA51H	ENUR3T	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_UR3T_STA	UR3T_DMA 状态寄存器	7EFA52H	-	-	-	-	-	TXOVW	-	UR3TIF	xxxx,x0x0
DMA_UR3T_AMT	UR3T_DMA 传输总字节数	7EFA53H	AMT[7:0]								0000,0000
DMA_UR3T_DONE	UR3T_DMA 传输完成字节数	7EFA54H	DONE[7:0]								0000,0000
DMA_UR3T_TXAH	UR3T_DMA 发送高地址	7EFA55H	ADR[15:0]								0000,0000
DMA_UR3T_TXAL	UR3T_DMA 发送低地址	7EFA56H	ADR[7:0]								0000,0000
DMA_UR3R_CFG	UR3R_DMA 配置寄存器	7EFA58H	UR3RIE	-	-	-	UR3RIP[1:0]		UR3RPTY[1:0]		0xxx,0000
DMA_UR3R_CR	UR3R_DMA 控制寄存器	7EFA59H	ENUR3R	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0
DMA_UR3R_STA	UR3R_DMA 状态寄存器	7EFA5AH	-	-	-	-	-	-	RXLOSS	UR3RIF	xxxx,xx00
DMA_UR3R_AMT	UR3R_DMA 传输总字节数	7EFA5BH	AMT[7:0]								0000,0000
DMA_UR3R_DONE	UR3R_DMA 传输完成字节数	7EFA5CH	DONE[7:0]								0000,0000
DMA_UR3R_RXAH	UR3R_DMA 接收高地址	7EFA5DH	ADR[15:0]								0000,0000
DMA_UR3R_RXAL	UR3R_DMA 接收低地址	7EFA5EH	ADR[7:0]								0000,0000
DMA_UR4T_CFG	UR4T_DMA 配置寄存器	7EFA60H	UR4TIE	-	-	-	UR4TIP[1:0]		UR4TPTY[1:0]		0xxx,0000
DMA_UR4T_CR	UR4T_DMA 控制寄存器	7EFA61H	ENUR4T	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_UR4T_STA	UR4T_DMA 状态寄存器	7EFA62H	-	-	-	-	-	TXOVW	-	UR4TIF	xxxx,x0x0
DMA_UR4T_AMT	UR4T_DMA 传输总字节数	7EFA63H	AMT[7:0]								0000,0000

DMA_UR4T_DONE	UR4T_DMA 传输完成字节数	7EFA64H	DONE[7:0]								0000,0000
DMA_UR4T_TXAH	UR4T_DMA 发送高地址	7EFA65H	ADR[15:0]								0000,0000
DMA_UR4T_TXAL	UR4T_DMA 发送低地址	7EFA66H	ADR[7:0]								0000,0000
DMA_UR4R_CFG	UR4R_DMA 配置寄存器	7EFA68H	UR4RIE	-	-	-	UR4RIP[1:0]		UR4RPTY[1:0]		0xxx,0000
DMA_UR4R_CR	UR4R_DMA 控制寄存器	7EFA69H	ENUR4R	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0
DMA_UR4R_STA	UR4R_DMA 状态寄存器	7EFA6AH	-	-	-	-	-	-	RXLOSS	UR4RIF	xxxx,xx00
DMA_UR4R_AMT	UR4R_DMA 传输总字节数	7EFA6BH	AMT[7:0]								0000,0000
DMA_UR4R_DONE	UR4R_DMA 传输完成字节数	7EFA6CH	DONE[7:0]								0000,0000
DMA_UR4R_RXAH	UR4R_DMA 接收高地址	7EFA6DH	ADR[15:0]								0000,0000
DMA_UR4R_RXAL	UR4R_DMA 接收低地址	7EFA6EH	ADR[7:0]								0000,0000
DMA_LCM_CFG	TFT 彩屏 DMA 配置寄存器	7EFA70H	LCMIE	-	-	-	LCMIP[1:0]		LCMPTY[1:0]		0xxx,0000
DMA_LCM_CR	TFT 彩屏 DMA 控制寄存器	7EFA71H	ENLCM	TRIGWC	TRIGWD	TRIGRC	TRIGRD	-	-	-	0000,0xxx
DMA_LCM_STA	TFT 彩屏 DMA 状态寄存器	7EFA72H	-	-	-	-	-	-	TXOVW	LCMIF	xxxx,xx00
DMA_LCM_AMT	TFT 彩屏 DMA 传输总字节数	7EFA73H	AMT[7:0]								0000,0000
DMA_LCM_DONE	TFT 彩屏 DMA 传输完成字节数	7EFA74H	DONE[7:0]								0000,0000
DMA_LCM_TXAH	TFT 彩屏 DMA 发送高地址	7EFA75H	ADR[15:0]								0000,0000
DMA_LCM_TXAL	TFT 彩屏 DMA 发送低地址	7EFA76H	ADR[7:0]								0000,0000
DMA_LCM_RXAH	TFT 彩屏 DMA 接收高地址	7EFA77H	ADR[15:0]								0000,0000
DMA_LCM_RXAL	TFT 彩屏 DMA 接收低地址	7EFA78H	ADR[7:0]								0000,0000
DMA_LCM_ITVH	TFT 彩屏 DMA 时间间隔寄存器	7EFA7EH	ITV[15:8]								0000,0000
DMA_LCM_ITVL	TFT 彩屏 DMA 时间间隔寄存器	7EFA7FH	ITV[7:0]								0000,0000
DMA_M2M_AMTH	M2M_DMA 传输总字节数	7EFA80H	AMT[15:8]								0000,0000
DMA_M2M_DONEH	M2M_DMA 传输完成字节数	7EFA81H	DONE[15:8]								0000,0000
DMA_ADC_AMTH	ADC_DMA 传输总字节数	7EFA82H	AMT[15:8]								0000,0000
DMA_ADC_DONEH	ADC_DMA 传输完成字节数	7EFA83H	DONE[15:8]								0000,0000
DMA_SPI_AMTH	SPI_DMA 传输总字节数	7EFA84H	AMT[15:8]								0000,0000
DMA_SPI_DONEH	SPI_DMA 传输完成字节数	7EFA85H	DONE[15:8]								0000,0000
DMA_LCM_AMTH	TFT 彩屏 DMA 传输总字节数	7EFA86H	AMT[15:8]								0000,0000
DMA_LCM_DONEH	TFT 彩屏 DMA 传输完成字节数	7EFA87H	DONE[15:8]								0000,0000
DMA_UR1T_AMTH	UR1T_DMA 传输总字节数	7EFA88H	AMT[15:8]								0000,0000
DMA_UR1T_DONEH	UR1T_DMA 传输完成字节数	7EFA89H	DONE[15:8]								0000,0000
DMA_UR1R_AMTH	UR1R_DMA 传输总字节数	7EFA8AH	AMT[15:8]								0000,0000
DMA_UR1R_DONEH	UR1R_DMA 传输完成字节数	7EFA8BH	DONE[15:8]								0000,0000
DMA_UR2T_AMTH	UR2T_DMA 传输总字节数	7EFA8CH	AMT[15:8]								0000,0000
DMA_UR2T_DONEH	UR2T_DMA 传输完成字节数	7EFA8DH	DONE[15:8]								0000,0000
DMA_UR2R_AMTH	UR2R_DMA 传输总字节数	7EFA8EH	AMT[15:8]								0000,0000
DMA_UR2R_DONEH	UR2R_DMA 传输完成字节数	7EFA8FH	DONE[15:8]								0000,0000
DMA_UR3T_AMTH	UR3T_DMA 传输总字节数	7EFA90H	AMT[15:8]								0000,0000
DMA_UR3T_DONEH	UR3T_DMA 传输完成字节数	7EFA91H	DONE[15:8]								0000,0000
DMA_UR3R_AMTH	UR3R_DMA 传输总字节数	7EFA92H	AMT[15:8]								0000,0000
DMA_UR3R_DONEH	UR3R_DMA 传输完成字节数	7EFA93H	DONE[15:8]								0000,0000
DMA_UR4T_AMTH	UR4T_DMA 传输总字节数	7EFA94H	AMT[15:8]								0000,0000
DMA_UR4T_DONEH	UR4T_DMA 传输完成字节数	7EFA95H	DONE[15:8]								0000,0000
DMA_UR4R_AMTH	UR4R_DMA 传输总字节数	7EFA96H	AMT[15:8]								0000,0000

DMA_UR4R_DONEH	UR4R_DMA 传输完成字节数	7EFA97H	DONE[15:8]							0000,0000
DMA_I2CT_CFG	I2CT_DMA 配置寄存器	7EFA98H	I2CTIE	-	-	-	I2CTIP[1:0]		I2CTPTY[1:0]	0xxx,0000
DMA_I2CT_CR	I2CT_DMA 控制寄存器	7EFA99H	ENI2CT	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_I2CT_STA	I2CT_DMA 状态寄存器	7EFA9AH	-	-	-	-	-	TXOVW	-	I2CTIF	xxxx,x0x0
DMA_I2CT_AMT	I2CT_DMA 传输总字节数	7EFA9BH	AMT[7:0]							0000,0000
DMA_I2CT_DONE	I2CT_DMA 传输完成字节数	7EFA9CH	DONE[7:0]							0000,0000
DMA_I2CT_TXAH	I2CT_DMA 发送高地址	7EFA9DH	ADR[15:0]							0000,0000
DMA_I2CT_TXAL	I2CT_DMA 发送低地址	7EFA9EH	ADR[7:0]							0000,0000
DMA_I2CR_CFG	I2CR_DMA 配置寄存器	7EFAA0H	I2CRIE	-	-	-	I2CRIP[1:0]		I2CRPTY[1:0]	0xxx,0000
DMA_I2CR_CR	I2CR_DMA 控制寄存器	7EFAA1H	ENI2CR	TRIG	-	-	-	-	-	CLRFIFO	00xx,xxx0
DMA_I2CR_STA	I2CR_DMA 状态寄存器	7EFAA2H	-	-	-	-	-	-	RXLOSS	I2CRIF	xxxx,xx00
DMA_I2CR_AMT	I2CR_DMA 传输总字节数	7EFAA3H	AMT[7:0]							0000,0000
DMA_I2CR_DONE	I2CR_DMA 传输完成字节数	7EFAA4H	DONE[7:0]							0000,0000
DMA_I2CR_RXAH	I2CR_DMA 接收高地址	7EFAA5H	ADR[15:0]							0000,0000
DMA_I2CR_RXAL	I2CR_DMA 接收低地址	7EFAA6H	ADR[7:0]							0000,0000
DMA_I2CT_AMTH	I2CT_DMA 传输总字节数	7EFAA8H	AMT[15:8]							0000,0000
DMA_I2CT_DONEH	I2CT_DMA 传输完成字节数	7EFAA9H	DONE[15:8]							0000,0000
DMA_I2CR_AMTH	I2CR_DMA 传输总字节数	7EFAAAH	AMT[15:8]							0000,0000
DMA_I2CR_DONEH	I2CR_DMA 传输完成字节数	7EFAABH	DONE[15:8]							0000,0000
DMA_I2C_CR	I2C_DMA 控制寄存器	7EFAADH	RDSEL	-	-	-	-	ACKERR	INTEN	BMMEN	0xxx,x000
DMA_I2C_ST1	I2C_DMA 状态寄存器	7EFAAEH	COUNT[7:0]							0000,0000
DMA_I2C_ST2	I2C_DMA 状态寄存器	7EFAAFH	COUNT[15:8]							0000,0000
DMA_I2ST_CFG	I2ST_DMA 配置寄存器	7EFAB0H	I2STIE	-	-	-	I2STIP[1:0]		I2STPTY[1:0]	0xxx,0000
DMA_I2ST_CR	I2ST_DMA 控制寄存器	7EFAB1H	ENI2ST	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_I2ST_STA	I2ST_DMA 状态寄存器	7EFAB2H	-	-	-	-	-	TXOVW	-	I2STIF	xxxx,x0x0
DMA_I2ST_AMT	I2ST_DMA 传输总字节数	7EFAB3H	AMT[7:0]							0000,0000
DMA_I2ST_DONE	I2ST_DMA 传输完成字节数	7EFAB4H	DONE[7:0]							0000,0000
DMA_I2ST_TXAH	I2ST_DMA 发送高地址	7EFAB5H	ADR[15:0]							0000,0000
DMA_I2ST_TXAL	I2ST_DMA 发送低地址	7EFAB6H	ADR[7:0]							0000,0000
DMA_I2SR_CFG	I2SR_DMA 配置寄存器	7EFAB8H	I2SRIE	-	-	-	I2SRIP[1:0]		I2SRPTY[1:0]	0xxx,0000
DMA_I2SR_CR	I2SR_DMA 控制寄存器	7EFAB9H	ENI2SR	TRIG	-	-	-	-	-	CLRFIFO	00xx,xxx0
DMA_I2SR_STA	I2SR_DMA 状态寄存器	7EFABAH	-	-	-	-	-	-	RXLOSS	I2SRIF	xxxx,xx00
DMA_I2SR_AMT	I2SR_DMA 传输总字节数	7EFABBH	AMT[7:0]							0000,0000
DMA_I2SR_DONE	I2SR_DMA 传输完成字节数	7EFABCH	DONE[7:0]							0000,0000
DMA_I2SR_RXAH	I2SR_DMA 接收高地址	7EFABDH	ADR[15:0]							0000,0000
DMA_I2SR_RXAL	I2SR_DMA 接收低地址	7EFABEH	ADR[7:0]							0000,0000
DMA_I2ST_AMTH	I2ST_DMA 传输总字节数	7EFAC0H	AMT[15:8]							0000,0000
DMA_I2ST_DONEH	I2ST_DMA 传输完成字节数	7EFAC1H	DONE[15:8]							0000,0000
DMA_I2SR_AMTH	I2SR_DMA 传输总字节数	7EFAC2H	AMT[15:8]							0000,0000
DMA_I2SR_DONEH	I2SR_DMA 传输完成字节数	7EFAC3H	DONE[15:8]							0000,0000
DMA_I2C_ITVH	I2C_DMA 时间间隔寄存器	7EFAC4H	ITV[15:8]							0000,0000
DMA_I2C_ITVL	I2C_DMA 时间间隔寄存器	7EFAC5H	ITV[7:0]							0000,0000
DMA_I2S_ITVH	I2S_DMA 时间间隔寄存器	7EFAC6H	ITV[15:8]							0000,0000
DMA_I2S_ITVL	I2S_DMA 时间间隔寄存器	7EFAC7H	ITV[7:0]							0000,0000

DMA_UR1_ITVH	UR1_DMA 时间间隔寄存器	7EFAC8H	ITV[15:8]							0000,0000
DMA_UR1_ITVL	UR1_DMA 时间间隔寄存器	7EFAC9H	ITV[7:0]							0000,0000
DMA_UR2_ITVH	UR2_DMA 时间间隔寄存器	7EFACAH	ITV[15:8]							0000,0000
DMA_UR2_ITVL	UR2_DMA 时间间隔寄存器	7EFACBH	ITV[7:0]							0000,0000
DMA_UR3_ITVH	UR3_DMA 时间间隔寄存器	7EFACCH	ITV[15:8]							0000,0000
DMA_UR3_ITVL	UR3_DMA 时间间隔寄存器	7EFACDH	ITV[7:0]							0000,0000
DMA_UR4_ITVH	UR4_DMA 时间间隔寄存器	7EFACEH	ITV[15:8]							0000,0000
DMA_UR4_ITVL	UR4_DMA 时间间隔寄存器	7EFACFH	ITV[7:0]							0000,0000
DMA_QSPI_CFG	QSPI_DMA 配置寄存器	7EFAD0H	QSPIIE	ACT_WR	ACT_RD	-	QSPIIP[1:0]		QSPIPTY[1:0]	000x,0000
DMA_QSPI_CR	QSPI_DMA 控制寄存器	7EFAD1H	ENQSPI	TRIG_W	TRIG_R	-	-	-	CLRWFIFO	CLRRFIFO	000x,xx00
DMA_QSPI_STA	QSPI_DMA 状态寄存器	7EFAD2H	-	-	-	-	-	-	-	QSPIIF	xxxx,xxx0
DMA_QSPI_AMT	QSPI_DMA 传输总字节数	7EFAD3H	AMT[7:0]							0000,0000
DMA_QSPI_DONE	QSPI_DMA 传输完成字节数	7EFAD4H	DONE[7:0]							0000,0000
DMA_QSPI_TXAH	QSPI_DMA 发送高地址	7EFAD5H	ADR[15:0]							0000,0000
DMA_QSPI_TXAL	QSPI_DMA 发送低地址	7EFAD6H	ADR[7:0]							0000,0000
DMA_QSPI_RXAH	QSPI_DMA 接收高地址	7EFAD7H	ADR[15:0]							0000,0000
DMA_QSPI_RXAL	QSPI_DMA 接收低地址	7EFAD8H	ADR[7:0]							0000,0000
DMA_QSPI_AMTH	QSPI_DMA 传输总字节数	7EFADBH	AMT[15:8]							0000,0000
DMA_QSPI_DONEH	QSPI_DMA 传输完成字节数	7EFADCH	DONE[15:8]							0000,0000
DMA_QSPI_ITVH	QSPI_DMA 时间间隔寄存器	7EFADEH	ITV[15:8]							0000,0000
DMA_QSPI_ITVL	QSPI_DMA 时间间隔寄存器	7EFADFH	ITV[7:0]							0000,0000
DMA_P2P_CR1	P2P DMA 控制寄存器 1	7EFAF0H	SRC1[3:0]				DEST1[3:0]			0000,0000
DMA_P2P_CR2	P2P DMA 控制寄存器 2	7EFAF1H	SRC2[3:0]				DEST2[3:0]			0000,0000
DMA_ARB_CFG	DMA 总裁配置寄存器	7EFAF8H	WTRREN	-	-	-	STASEL[3:0]-			0xxx,0000
DMA_ARB_STA	DMA 总裁状态寄存器	7EFAF9H	STA[7:0]							0000,0000

13.8 扩展特殊功能寄存器列表（XFR：0x7EF900-0x7EF9FF）

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
QSPI_CR1	QSPI 控制寄存器 1	7EF900H	-	-	-	-	-	-	ABORT	EN	xxxx,xx00
QSPI_CR2	QSPI 控制寄存器 2	7EF901H	-	-	-	FTHRES[4:0]					xxx0,0000
QSPI_CR3	QSPI 控制寄存器 3	7EF902H	PMM	APMS	-	-	SMIE	FTIE	TCIE	TEIE	00xx,0000
QSPI_CR4	QSPI 控制寄存器 4	7EF903H	PSCR[7:0]								0000,0000
QSPI_DCR1	QSPI 器件配置寄存器 1	7EF904H	-	CSHT[2:0]			QSPI_IP[1:0]		-	CKMODE	x000,00x0
QSPI_DCR2	QSPI 器件配置寄存器 2	7EF905H	-	-	-	FSIZE[4:0]					0000,0000
QSPI_SR1	QSPI 状态寄存器 1	7EF906H	-	-	BUSY	-	SMF	FTF	TCF	TEF	xx0x,0100
QSPI_SR2	QSPI 状态寄存器 2	7EF907H	-	-	FLEVEL[5:0]						xx00,0000
QSPI_FCR	QSPI 标志清零寄存器	7EF908H	-	-	-	-	CSMF	-	CTCF	CTEF	xxxx,0x00
~~QSPI_HCR1~~	~~QSPI 控制寄存器 1~~	~~7EF909H~~	~~HLD1[7:0]~~								~~0000,0000~~
~~QSPI_HCR2~~	~~QSPI 控制寄存器 2~~	~~7EF90AH~~	~~HLD2[7:0]~~								~~0000,0000~~
QSPI_DLR1	QSPI 数据长度寄存器 1	7EF910H	DL[7:0]								0000,0000
QSPI_DLR2	QSPI 数据长度寄存器 2	7EF911H	DL[15:8]								0000,0000
QSPI_CCR1	QSPI 通信配置寄存器 1	7EF914H	INSTRUCTION[7:0]								0000,0000
QSPI_CCR2	QSPI 通信配置寄存器 2	7EF915H	ABMODE[1:0]		ADSIZE[1:0]		ADMODE[1:0]		IMODE[1:0]		0000,0000
QSPI_CCR3	QSPI 通信配置寄存器 3	7EF916H	-	DCYC[4:0]					ABSIZE[1:0]		x000,0000
QSPI_CCR4	QSPI 通信配置寄存器 4	7EF917H	-	-	-	SIOO	FMODE[1:0]		DMODE[1:0]		xxx0,0000
QSPI_AR1	QSPI 地址寄存器 1	7EF918H	ADR[7:0]								0000,0000
QSPI_AR2	QSPI 地址寄存器 2	7EF919H	ADR[15:8]								0000,0000
QSPI_AR3	QSPI 地址寄存器 3	7EF91AH	ADR[23:16]								0000,0000
QSPI_AR4	QSPI 地址寄存器 4	7EF91BH	ADR[31:24]								0000,0000
QSPI_ABR	QSPI 交替字节寄存器 1	7EF91CH	ALTERNATED[7:0]								0000,0000
QSPI_DR	QSPI 数据寄存器	7EF920H	DATA[7:0]								0000,0000
QSPI_PSMKR1	QSPI 状态屏蔽寄存器	7EF924H	MASK[7:0]								0000,0000
QSPI_PSMAR1	QSPI 状态匹配寄存器	7EF928H	MATCH[7:0]								0000,0000
QSPI_PIR1	QSPI 轮询间隔寄存器 1	7EF92CH	INTERVAL[7:0]								0000,0000
QSPI_PIR2	QSPI 轮询间隔寄存器 2	7EF92DH	INTERVAL[15:8]								0000,0000
PWMA_ENO2	PWMA 输出使能寄存器 2	7EF930H	-	-	-	-	-	ENO6	-	ENO5	xxxx,x0x0
PWMA_IOAUX2	PWMA 输出附加寄存器 2	7EF931H	-	-	-	-	-	AUX6	-	AUX6	xxxx,x0x0
PWMA_CR3	PWMA 控制寄存器 3	7EF932H	MMS2[3:0]				-	OIS6	-	OIS5	0000,x0x0
PWMA_SR3	PWMA 状态寄存器 3	7EF933H	-	-	-	-	-	-	CC6IF	CC5IF	xxxx,xx00
PWMA_CCER3	PWMA 捕获比较使能寄存器 3	7EF934H	-	-	CC6P	CC6E	-	-	CC5P	CC5E	xx00,xx00
PWMA_CCMR1X	PWMA 捕获比较模式寄存器 1x	7EF938H	-	-	-	-	-	-	-	OC1M[3]	xxx,xxx0
PWMA_CCMR2X	PWMA 捕获比较模式寄存器 2x	7EF939H	-	-	-	-	-	-	-	OC2M[3]	xxx,xxx0
PWMA_CCMR3X	PWMA 捕获比较模式寄存器 3x	7EF93AH	-	-	-	-	-	-	-	OC3M[3]	xxx,xxx0
PWMA_CCMR4X	PWMA 捕获比较模式寄存器 4x	7E93BH	-	-	-	-	-	-	-	OC4M[3]	0000,0000
PWMA_CCMR5	PWMA 捕获比较模式寄存器 5	7EF93CH	OC5CE	OC5M[2:0]			OC5PE	-	-	-	0000,0xxx
PWMA_CCMR5X	PWMA 捕获比较模式寄存器 5x	7EF93DH	-	-	-	-	-	-	-	OC5M[3]	xxxx,xxx0
PWMA_CCMR6	PWMA 捕获比较模式寄存器 6	7EF93EH	OC6CE	OC6M[2:0]	OC6PE	-	-	-	-	-	000x,xxxx

PWMA_CCMR6X	PWMA 捕获比较模式寄存器 6x	7EF93FH	-	-	-	-	-	-	-	OC6M[3]	xxx,xxx0
PWMA_CCR5H	PWMA 捕获比较寄存器 5	7EF940H	CCR5[15:8]								0000,0000
PWMA_CCR5L	PWMA 捕获比较寄存器 5	7EF941H	CCR5[7:0]								0000,0000
PWMA_CCR5X	PWMA 捕获比较寄存器 5	7EF942H	GC5C3	GC5C2	GC5C1	-	-	-	-	-	000x,xxx
PWMA_CCR6H	PWMA 捕获比较寄存器 6	7EF943H	CCR6[15:8]								0000,0000
PWMA_CCR6L	PWMA 捕获比较寄存器 6	7EF944H	CCR6[7:0]								0000,0000
PWMA_DER	PWMA_DMA 事件使能寄存器	7EF948H	-	TDE	COMDE	CC4DE	CC3DE	CC2DE	CC1DE	UDE	0000,0000
PWMA_DBA	PWMA_DMA 基址寄存器	7EF949H	-	-	-	DBA[4:0]					xxx0,0000
PWMA_DBL	PWMA_DMA 传输长度寄存器	7EF94AH	-	-	-	DBL[4:0]					xxx0,0000
PWMA_DMACR	PWMA_DMA 控制寄存器	7EF94BH	-	-	-	DSKIP	DDIR	DMAEN	SIZE[1:0]		xxx0,0000
DMA_PWMAT_CFG	PWMAT_DMA 配置寄存器	7EF980H	PWMATIE	-	-	-	PWMATMIP[1:0]		PWMATBAP[1:0]		0xxx,0000
DMA_PWMAT_CR	PWMAT_DMA 控制寄存器	7EF981H	ENPWMAT	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_PWMAT_STA	PWMAT_DMA 状态寄存器	7EF982H	RDOV	-	-	-	-	TXOVW	-	PWMATIF	0xxx,x0x0
DMA_PWMAT_AMTH	PWMAT_DMA 传输总字节数	7EF984H	AMT[15:8]								0000,0000
DMA_PWMAT_AMT	PWMAT_DMA 传输总字节数	7EF985H	AMT[7:0]								0000,0000
DMA_PWMAT_DONEH	PWMAT_DMA 传输完成字节数	7EF986H	DONE[15:8]								0000,0000
DMA_PWMAT_DONE	PWMAT_DMA 传输完成字节数	7EF987H	DONE [7:0]								0000,0000
DMA_PWMAT_TXAH	PWMAT_DMA 发送高地址	7EF988H	ADR[15:8]								0000,0000
DMA_PWMAT_TXAL	PWMAT_DMA 发送低地址	7EF989H	ADR[7:0]								0000,0000
DMA_PWMA_ITVH	PWMAT_DMA 时间间隔寄存器	7EF98EH	ITV[15:8]								0000,0000
DMA_PWMA_ITVL	PWMAT _DMA 时间间隔寄存器	7EF98FH	ITV[7:0]								0000,0000
DMA_PWMAR_CFG	PWMAR_DMA 配置寄存器	7EF990H	PWMARIE	-	-	-	PWMARMIP[1:0]		PWMARBAP[1:0]		0xxx,0000
DMA_PWMAR_CR	PWMAR_DMA 控制寄存器	7EF991H	ENPWMAR	TRIG	-	-	-	-	-	CLRFIFO	00xx,xxx0
DMA_PWMAR_STA	PWMAR_DMA 状态寄存器	7EF992H	WROV	-	-	-	-	-	RXLOSS	PWMARIF	0xxx,xx00
DMA_PWMAR_AMTH	PWMAR_DMA 传输总字节数	7EF994H	AMT[15:8]								0000,0000
DMA_PWMAR_AMT	PWMAR_DMA 传输总字节数	7EF995H	AMT[7:0]								0000,0000
DMA_PWMAR_DONEH	PWMAR_DMA 传输完成字节数	7EF996H	DONE[15:8]								0000,0000
DMA_PWMAR_DONE	PWMAR_DMA 传输完成字节数	7EF997H	DONE [7:0]								0000,0000
DMA_PWMAR_RXAH	PWMAR_DMA 接收高地址	7EF998H	ADR[15:8]								0000,0000
DMA_PWMAR_RXAL	PWMAR_DMA 接收低地址	7EF999H	ADR[7:0]								0000,0000

14 I/O 口

产品线	最多 I/O 口数量
Ai8051U 系列	**45**

Ai8051U 系列单片机所有的 I/O 口均有 4 种工作模式：准双向口/弱上拉（标准 8051 输出口模式）、推挽输出/强上拉、高阻输入（电流既不能流入也不能流出）、开漏模式。可使用软件对 I/O 口的工作模式进行各种工作模式的设置。

关于 I/O 的注意事项：

1、 P3.0 和 P3.1 口上电后的状态为弱上拉/准双向口模式

2、 除 P3.0 和 P3.1 外，其余所有 I/O 口上电后的状态均为高阻输入状态，用户在使用 I/O 口前必须先设置 I/O 口模式

3、 芯片上电时如果不需要使用 USB 进行 ISP 下载，P3.0/P3.1/P3.2 这 3 个 I/O 口不能同时为低电平，否则会进入 USB 下载模式而无法运行用户代码

4、 芯片上电时，若 P3.0 和 P3.1 同时为低电平，系统程序会短时间将上电初始状态是高阻输入的 P3.2 口对应的内部上拉电阻打开，并延时 1mS，来准确读取上电时 P3.2 口的外部状态，如果读到 P3.2 外部状态是低电平，则进入等待 USB 下载模式；如果读到 P3.2 外部状态是高电平，则不进入 USB 下载模式，并立即将 P3.2 口的内部上拉电阻关闭

5、 当使用 P4.7 当作复位脚时，这个端口内部的 4K 上拉电阻会一直打开；但 P4.7 做普通 I/O 口时，基于这个 I/O 口与复位脚共享管脚的特殊考量，端口内部的 4K 上拉电阻依然会打开大约 6.5 毫秒时间，再自动关闭（当用户的电路设计需要使用 P4.7 口驱动外部电路时，请务必考虑上电瞬间会有 6.5 毫秒时间的高电平的问题）

6、 内部上拉电阻和外部上拉电阻的部分区别：

外部上拉电阻随系统电源从 0V 上升到 MCU-VCC，到 3.3V 或 5V，这个时间是足够的长，系统上电复位后到初始化运行用户程序时，外部上拉电阻已将外部拉高到高电平。

内部上拉电阻是运行用户程序时才使能打开的，由于外部电路的分布电容效应，要等待 0.7RC 时间（R:电阻，C:电容）才能将外部拉高到 0.7MCU-VCC 高电平。建议将需要打开的内部上拉电阻使能后，直接多等待一会，如等待 1ms 后程序才认为外部已被拉高。

7、 特别注意，有硬件 USB 的 MCU：

P3.2 口是高电平（是 高阻输入 + 内部 4K 上拉电阻被打开）@系统程序区时
芯片上电复位后，首先跑的是系统程序，而不是用户程序。

为了判断是否要等待硬件 USB 下载，后来软件为了支持硬件 USB 下载做了优化，
无论是否需要下载还是进行串口下载或者 USB 下载，
P3.2 口都是高电平（是 高阻输入 + 内部 4K 上拉电阻被打开），
在系统区 P3.2 的 4K 上拉电阻被打开了。

等从系统区复位到用户区后，P3.2 的内部上拉电阻才被复位关闭。

14.1 I/O 口相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
P0	P0 端口	80H	P07	P06	P05	P04	P03	P02	P01	P00	1111,1111
P1	P1 端口	90H	P17	P16	P15	P14	P13	P12	P11	P10	1111,1111
P2	P2 端口	A0H	P27	P26	P25	P24	P23	P22	P21	P20	1111,1111
P3	P3 端口	B0H	P37	P36	P35	P34	P33	P32	P31	P30	1111,1111
P4	P4 端口	C0H	P47	P46	P45	P44	P43	P42	P41	P40	1111,1111
P5	P5 端口	C8H	P57	P56	P55	P54	P53	P52	P51	P50	1111,1111
P6	P6 端口	E8H	P67	P66	P65	P64	P63	P62	P61	P60	1111,1111
P7	P7 端口	F8H	P77	P76	P75	P74	P73	P72	P71	P70	1111,1111
P0M1	P0 口配置寄存器 1	93H	P0M1[7:0]								1111,1111
P0M0	P0 口配置寄存器 0	94H	P0M0[7:0]								0000,0000
P1M1	P1 口配置寄存器 1	91H	P1M1[7:0]								1111,1111
P1M0	P1 口配置寄存器 0	92H	P1M0[7:0]								0000,0000
P2M1	P2 口配置寄存器 1	95H	P2M1[7:0]								1111,1111
P2M0	P2 口配置寄存器 0	96H	P2M0[7:0]								0000,0000
P3M1	P3 口配置寄存器 1	B1H	P3M1[7:0]								1111,1100
P3M0	P3 口配置寄存器 0	B2H	P3M0[7:0]								0000,0000
P4M1	P4 口配置寄存器 1	B3H	P4M1[7:0]								1111,1111
P4M0	P4 口配置寄存器 0	B4H	P4M0[7:0]								0000,0000
P5M1	P5 口配置寄存器 1	C9H	P5M1[7:0]								1111,1111
P5M0	P5 口配置寄存器 0	CAH	P5M0[7:0]								xx00,0000
P6M1	P6 口配置寄存器 1	CBH	P6M1[7:0]								1111,1111
P6M0	P6 口配置寄存器 0	CCH	P6M0[7:0]								0000,0000
P7M1	P7 口配置寄存器 1	E1H	P7M1[7:0]								1111,1111
P7M0	P7 口配置寄存器 0	E2H	P7M0[7:0]								0000,0000

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
P0PU	P0 口上拉电阻控制寄存器	7EFE10H	P0PU[7:0]								0000,0000
P1PU	P1 口上拉电阻控制寄存器	7EFE11H	P1PU[7:0]								0000,0000
P2PU	P2 口上拉电阻控制寄存器	7EFE12H	P2PU[7:0]								0000,0000
P3PU	P3 口上拉电阻控制寄存器	7EFE13H	P3PU[7:0]								0000,0000
P4PU	P4 口上拉电阻控制寄存器	7EFE14H	P4PU[7:0]								0000,0000
P5PU	P5 口上拉电阻控制寄存器	7EFE15H	P5PU[7:0]								0000,0000
P6PU	P6 口上拉电阻控制寄存器	7EFE16H	P6PU[7:0]								0000,0000
P7PU	P7 口上拉电阻控制寄存器	7EFE17H	P7PU[7:0]								0000,0000
P0NCS	P0 口施密特触发控制寄存器	7EFE18H	P0NCS[7:0]								0000,0000
P1NCS	P1 口施密特触发控制寄存器	7EFE19H	P1NCS[7:0]								0000,0000
P2NCS	P2 口施密特触发控制寄存器	7EFE1AH	P2NCS[7:0]								0000,0000
P3NCS	P3 口施密特触发控制寄存器	7EFE1BH	P3NCS[7:0]								0000,0000
P4NCS	P4 口施密特触发控制寄存器	7EFE1CH	P4NCS[7:0]								0000,0000

P5NCS	P5 口施密特触发控制寄存器	7EFE1DH	P5NCS[7:0]	0000,0000
P6NCS	P6 口施密特触发控制寄存器	7EFE1EH	P6NCS[7:0]	0000,0000
P7NCS	P7 口施密特触发控制寄存器	7EFE1FH	P7NCS[7:0]	0000,0000
P0SR	P0 口电平转换速率寄存器	7EFE20H	P0SR[7:0]	1111,1111
P1SR	P1 口电平转换速率寄存器	7EFE21H	P1SR[7:0]	1111,1111
P2SR	P2 口电平转换速率寄存器	7EFE22H	P2SR[7:0]	1111,1111
P3SR	P3 口电平转换速率寄存器	7EFE23H	P3SR[7:0]	1111,1111
P4SR	P4 口电平转换速率寄存器	7EFE24H	P4SR[7:0]	1111,1111
P5SR	P5 口电平转换速率寄存器	7EFE25H	P5SR[7:0]	1111,1111
P6SR	P6 口电平转换速率寄存器	7EFE26H	P6SR[7:0]	1111,1111
P7SR	P7 口电平转换速率寄存器	7EFE27H	P7SR[7:0]	1111,1111
P0DR	P0 口驱动电流控制寄存器	7EFE28H	P0DR[7:0]	1111,1111
P1DR	P1 口驱动电流控制寄存器	7EFE29H	P1DR[7:0]	1111,1111
P2DR	P2 口驱动电流控制寄存器	7EFE2AH	P2DR[7:0]	1111,1111
P3DR	P3 口驱动电流控制寄存器	7EFE2BH	P3DR[7:0]	1111,1111
P4DR	P4 口驱动电流控制寄存器	7EFE2CH	P4DR[7:0]	1111,1111
P5DR	P5 口驱动电流控制寄存器	7EFE2DH	P5DR[7:0]	1111,1111
P6DR	P6 口驱动电流控制寄存器	7EFE2EH	P6DR[7:0]	1111,1111
P7DR	P7 口驱动电流控制寄存器	7EFE2FH	P7DR[7:0]	1111,1111
P0IE	P0 口输入使能控制寄存器	7EFE30H	P0IE[7:0]	1111,1111
P1IE	P1 口输入使能控制寄存器	7EFE31H	P1IE[7:0]	1111,1111
P2IE	P2 口输入使能控制寄存器	7EFE32H	P2IE[7:0]	1111,1111
P3IE	P3 口输入使能控制寄存器	7EFE33H	P3IE[7:0]	1111,1111
P4IE	P4 口输入使能控制寄存器	7EFE34H	P4IE[7:0]	1111,1111
P5IE	P5 口输入使能控制寄存器	7EFE35H	P5IE[7:0]	1111,1111
P6IE	P6 口输入使能控制寄存器	7EFE36H	P6IE[7:0]	1111,1111
P7IE	P7 口输入使能控制寄存器	7EFE37H	P7IE[7:0]	1111,1111
P0PD	P0 口下拉电阻控制寄存器	7EFE40H	P0PD[7:0]	0000,0000
P1PD	P1 口下拉电阻控制寄存器	7EFE41H	P1PD[7:0]	0000,0000
P2PD	P2 口下拉电阻控制寄存器	7EFE42H	P2PD[7:0]	0000,0000
P3PD	P3 口下拉电阻控制寄存器	7EFE43H	P3PD[7:0]	0000,0000
P4PD	P4 口下拉电阻控制寄存器	7EFE44H	P4PD[7:0]	0000,0000
P5PD	P5 口下拉电阻控制寄存器	7EFE45H	P5PD[7:0]	0000,0000
P6PD	P6 口下拉电阻控制寄存器	7EFE46H	P6PD[7:0]	0000,0000
P7PD	P7 口下拉电阻控制寄存器	7EFE47H	P7PD[7:0]	0000,0000
P0BP	P0 口模式用户控制寄存器	7EFE48H	P0BP[7:0]	1111,1111
P1BP	P1 口模式用户控制寄存器	7EFE49H	P1BP[7:0]	1111,1111
P2BP	P2 口模式用户控制寄存器	7EFE4AH	P2BP[7:0]	1111,1111
P3BP	P3 口模式用户控制寄存器	7EFE4BH	P3BP[7:0]	1111,1111
P4BP	P4 口模式用户控制寄存器	7EFE4CH	P4BP[7:0]	1111,1111
P5BP	P5 口模式用户控制寄存器	7EFE4DH	P5BP[7:0]	1111,1111
P6BP	P6 口模式用户控制寄存器	7EFE4EH	P6BP[7:0]	1111,1111
P7BP	P7 口模式用户控制寄存器	7EFE4FH	P7BP[7:0]	1111,1111

14.1.1 端口数据寄存器（Px）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0	80H	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0
P1	90H	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
P2	A0H	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0
P3	B0H	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0
P4	C0H	P4.7	P4.6	P4.5	P4.4	P4.3	P4.2	P4.1	P4.0
P5	C8H	P5.7	P5.6	P5.5	P5.4	P5.3	P5.2	P5.1	P5.0
P6	E8H	P6.7	P6.6	P6.5	P6.4	P6.3	P6.2	P6.1	P6.0
P7	F8H	P7.7	P7.6	P7.5	P7.4	P7.3	P7.2	P7.1	P7.0

读写端口状态

写 0：输出低电平到端口缓冲区

写 1：输出高电平到端口缓冲区

读：直接读端口管脚上的电平

14.1.2 端口模式配置寄存器（PxM0，PxM1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0M0	94H	P07M0	P06M0	P05M0	P04M0	P03M0	P02M0	P01M0	P00M0
P0M1	93H	P07M1	P06M1	P05M1	P04M1	P03M1	P02M1	P01M1	P00M1
P1M0	92H	P17M0	P16M0	P15M0	P14M0	P13M0	P12M0	P11M0	P10M0
P1M1	91H	P17M1	P16M1	P15M1	P14M1	P13M1	P12M1	P11M1	P10M1
P2M0	96H	P27M0	P26M0	P25M0	P24M0	P23M0	P22M0	P21M0	P20M0
P2M1	95H	P27M1	P26M1	P25M1	P24M1	P23M1	P22M1	P21M1	P20M1
P3M0	B2H	P37M0	P36M0	P35M0	P34M0	P33M0	P32M0	P31M0	P30M0
P3M1	B1H	P37M1	P36M1	P35M1	P34M1	P33M1	P32M1	P31M1	P30M1
P4M0	B4H	P47M0	P46M0	P45M0	P44M0	P43M0	P42M0	P41M0	P40M0
P4M1	B3H	P47M1	P46M1	P45M1	P44M1	P43M1	P42M1	P41M1	P40M1
P5M0	CAH	P57M0	P56M0	P55M0	P54M0	P53M0	P52M0	P51M0	P50M0
P5M1	C9H	P57M1	P56M1	P55M1	P54M1	P53M1	P52M1	P51M1	P50M1
P6M0	CCH	P67M0	P66M0	P65M0	P64M0	P63M0	P62M0	P61M0	P60M0
P6M1	CBH	P67M1	P66M1	P65M1	P64M1	P63M1	P62M1	P61M1	P60M1
P7M0	E2H	P77M0	P76M0	P75M0	P74M0	P73M0	P72M0	P71M0	P70M0
P7M1	E1H	P77M1	P76M1	P75M1	P74M1	P73M1	P72M1	P71M1	P70M1

配置端口的模式

PnM1.x	PnM0.x	Pn.x 口工作模式
0	0	准双向口
0	1	推挽输出
1	0	高阻输入
1	1	开漏模式

14.1.3 端口上拉电阻控制寄存器（PxPU）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0PU	7EFE10H	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU
P1PU	7EFE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU
P2PU	7EFE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU
P3PU	7EFE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU
P4PU	7EFE14H	P47PU	P46PU	P45PU	P44PU	P43PU	P42PU	P41PU	P40PU
P5PU	7EFE15H	P57PU	P56PU	P55PU	P54PU	P53PU	P52PU	P51PU	P50PU
P6PU	7EFE16H	P67PU	P66PU	P65PU	P64PU	P63PU	P62PU	P61PU	P60PU
P7PU	7EFE17H	P77PU	P76PU	P75PU	P74PU	P73PU	P72PU	P71PU	P70PU

端口内部上拉电阻控制位（注：P3.0和P3.1口上的上拉电阻可能会略小一些）

0：禁止端口内部的上拉电阻

1：使能端口内部的上拉电阻

上拉电阻 / 电压	设计目标	实际测量
5.0V	4.0K	4.2K
3.3V	6.3K	5.8K

14.1.4 端口施密特触发控制寄存器（PxNCS）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0NCS	7EFE18H	P07NCS	P06NCS	P05NCS	P04NCS	P03NCS	P02NCS	P01NCS	P00NCS
P1NCS	7EFE19H	P17NCS	P16NCS	P15NCS	P14NCS	P13NCS	P12NCS	P11NCS	P10NCS
P2NCS	7EFE1AH	P27NCS	P26NCS	P25NCS	P24NCS	P23NCS	P22NCS	P21NCS	P20NCS
P3NCS	7EFE1BH	P37NCS	P36NCS	P35NCS	P34NCS	P33NCS	P32NCS	P31NCS	P30NCS
P4NCS	7EFE1CH	P47NCS	P46NCS	P45NCS	P44NCS	P43NCS	P42NCS	P41NCS	P40NCS
P5NCS	7EFE1DH	P57NCS	P56NCS	P55NCS	P54NCS	P53NCS	P52NCS	P51NCS	P50NCS
P6NCS	7EFE1EH	P67NCS	P66NCS	P65NCS	P64NCS	P63NCS	P62NCS	P61NCS	P60NCS
P7NCS	7EFE1FH	P77NCS	P76NCS	P75NCS	P74NCS	P73NCS	P72NCS	P71NCS	P70NCS

端口施密特触发控制位

0：使能端口的施密特触发功能。（上电复位后默认使能施密特触发）

1：禁止端口的施密特触发功能。

14.1.5 端口电平转换速度控制寄存器（PxSR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0SR	7EFE20H	P07SR	P06SR	P05SR	P04SR	P03SR	P02SR	P01SR	P00SR
P1SR	7EFE21H	P17SR	P16SR	P15SR	P14SR	P13SR	P12SR	P11SR	P10SR
P2SR	7EFE22H	P27SR	P26SR	P25SR	P24SR	P23SR	P22SR	P21SR	P20SR
P3SR	7EFE23H	P37SR	P36SR	P35SR	P34SR	P33SR	P32SR	P31SR	P30SR
P4SR	7EFE24H	P47SR	P46SR	P45SR	P44SR	P43SR	P42SR	P41SR	P40SR
P5SR	7EFE25H	P57SR	P56SR	P55SR	P54SR	P53SR	P52SR	P51SR	P50SR
P6SR	7EFE26H	P67SR	P66SR	P65SR	P64SR	P63SR	P62SR	P61SR	P60SR

P7SR	7EFE27H	P77SR	P76SR	P75SR	P74SR	P73SR	P72SR	P71SR	P70SR

控制端口电平转换的速度

0：电平转换速度快，相应的上下冲会比较大

1：电平转换速度慢，相应的上下冲比较小

14.1.6 端口驱动电流控制寄存器（PxDR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0DR	7EFE28H	P07DR	P06DR	P05DR	P04DR	P03DR	P02DR	P01DR	P00DR
P1DR	7EFE29H	P17DR	P16DR	P15DR	P14DR	P13DR	P12DR	P11DR	P10DR
P2DR	7EFE2AH	P27DR	P26DR	P25DR	P24DR	P23DR	P22DR	P21DR	P20DR
P3DR	7EFE2BH	P37DR	P36DR	P35DR	P34DR	P33DR	P32DR	P31DR	P30DR
P4DR	7EFE2CH	P47DR	P46DR	P45DR	P44DR	P43DR	P42DR	P41DR	P40DR
P5DR	7EFE2DH	P57DR	P56DR	P55DR	P54DR	P53DR	P52DR	P51DR	P50DR
P6DR	7EFE2EH	P67DR	P66DR	P65DR	P64DR	P63DR	P62DR	P61DR	P60DR
P7DR	7EFE2FH	P77DR	P76DR	P75DR	P74DR	P73DR	P72DR	P71DR	P70DR

控制端口的驱动能力

0：增强驱动能力

1：一般驱动能力

14.1.7 端口数字信号输入使能控制寄存器（PxIE）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0IE	7EFE30H	P07IE	P06IE	P05IE	P04IE	P03IE	P02IE	P01IE	P00IE
P1IE	7EFE31H	P17IE	P16IE	P15IE	P14IE	P13IE	P12IE	P11IE	P10IE
P2IE	7EFE32H	P27IE	P26IE	P25IE	P24IE	P23IE	P22IE	P21IE	P20IE
P3IE	7EFE33H	P37IE	P36IE	P35IE	P34IE	P33IE	P32IE	P31IE	P30IE
P4IE	7EFE34H	P47IE	P46IE	P45IE	P44IE	P43IE	P42IE	P41IE	P40IE
P5IE	7EFE35H	P57IE	P56IE	P55IE	P54IE	P53IE	P52IE	P51IE	P50IE
P6IE	7EFE36H	P67IE	P66IE	P65IE	P64IE	P63IE	P62IE	P61IE	P60IE
P7IE	7EFE37H	P77IE	P76IE	P75IE	P74IE	P73IE	P72IE	P71IE	P70IE

数字信号输入使能控制

0：禁止数字信号输入。若 I/O 被当作比较器输入口、ADC 输入口或者触摸按键输入口等模拟口时，进入主时钟停振/省电模式前，必须设置为 0，否则会有额外的耗电。

1：使能数字信号输入。若 I/O 被当作数字口时，必须设置为 1，否 MCU 无法读取外部端口的电平。

单片机系列	P0IE	P1IE	P2IE	P3IE	P4IE	P5IE	P6IE	P7IE
Ai8051U 系列	✓	✓	✓	✓	✓	✓	-	-

14.1.8 端口下拉电阻控制寄存器（PxPD）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0PD	7EFE40H	P07PD	P06PD	P05PD	P04PD	P03PD	P02PD	P01PD	P00PD
P1PD	7EFE41H	P17PD	P16PD	P15PD	P14PD	P13PD	P12PD	P11PD	P10PD
P2PD	7EFE42H	P27PD	P26PD	P25PD	P24PD	P23PD	P22PD	P21PD	P20PD
P3PD	7EFE43H	P37PD	P36PD	P35PD	P34PD	P33PD	P32PD	P31PD	P30PD
P4PD	7EFE44H	P47PD	P46PD	P45PD	P44PD	P43PD	P42PD	P41PD	P40PD
P5PD	7EFE45H	P57PD	P56PD	P55PD	P54PD	P53PD	P52PD	P51PD	P50PD
P6PD	7EFE46H	P67PD	P66PD	P65PD	P64PD	P63PD	P62PD	P61PD	P60PD
P7PD	7EFE47H	P77PD	P76PD	P75PD	P74PD	P73PD	P72PD	P71PD	P70PD

端口内部下拉电阻控制位

0：禁止端口内部的下拉电阻

1：使能端口内部的下拉电阻

下拉电阻 / 电压	设计目标	实际测量
5.0V	47K	46K
3.3V	32K	35K

14.1.9 端口模式用户控制寄存器（PxBP）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0BP	7EFE48H	P07BP	P06BP	P05BP	P04BP	P03BP	P02BP	P01BP	P00BP
P1BP	7EFE49H	P17BP	P16BP	P15BP	P14BP	P13BP	P12BP	P11BP	P10BP
P2BP	7EFE4AH	P27BP	P26BP	P25BP	P24BP	P23BP	P22BP	P21BP	P20BP
P3BP	7EFE4BH	P37BP	P36BP	P35BP	P34BP	P33BP	P32BP	P31BP	P30BP
P4BP	7EFE4CH	P47BP	P46BP	P45BP	P44BP	P43BP	P42BP	P41BP	P40BP
P5BP	7EFE4DH	P57BP	P56BP	P55BP	P54BP	P53BP	P52BP	P51BP	P50BP
P6BP	7EFE4EH	P67BP	P66BP	P65BP	P64BP	P63BP	P62BP	P61BP	P60BP
P7BP	7EFE4FH	P77BP	P76BP	P75BP	P74BP	P73BP	P72BP	P71BP	P70BP

端口模式用户控制位

0：外设模块对所用 I/O 的模式进行自动配置，忽略用户使用 PxM0/PxM1 对相应 I/O 进行的配置。

1：外设模块对所用 I/O 的模式不自动配置，需要用户使用 PxM0/PxM1 寄存器对 I/O 进行配置。

14.2　配置 I/O 口

每个 I/O 的配置都需要使用两个寄存器进行设置。

以 P0 口为例，配置 P0 口需要使用 P0M0 和 P0M1 两个寄存器进行配置，如下图所示：

即 P0M0 的第 0 位和 P0M1 的第 0 位组合起来配置 P0.0 口的模式
即 P0M0 的第 1 位和 P0M1 的第 1 位组合起来配置 P0.1 口的模式
其他所有 I/O 的配置都与此类似。

PnM0 与 PnM1 的组合方式如下表所示

PnM1	PnM0	I/O 口工作模式
0	0	准双向口（传统8051端口模式，弱上拉） 灌电流可达20mA，拉电流为270~150μA（存在制造误差）
0	1	推挽输出（强上拉输出，可达20mA，要加限流电阻）
1	0	高阻输入（电流既不能流入也不能流出）
1	1	开漏模式（Open-Drain），内部上拉电阻断开 开漏模式既可读外部状态也可对外输出（高电平或低电平）。如要正确读外部状态或需要对外输出高电平，需外加上拉电阻，否则读不到外部状态，也对外输不出高电平。 **===【开漏工作模式】，对外设置输出为 1，等同于 【高阻输入】** **===【开漏工作模式】，【打开内部上拉电阻	或外部加上拉电阻】，简单等同于 【准双向口】**

注：n = 0, 1, 2, 3, 4, 5, 6, 7

注意：

虽然每个 I/O 口在弱上拉（准双向口）/强推挽输出/开漏模式时都能承受 20mA 的灌电流（还是要加限流电阻，如 1K、560Ω、472Ω 等），在强推挽输出时能输出 20mA 的拉电流（也要加限流电阻），但整个芯片的工作电流推荐不要超过 90mA，即从 VCC 流入的电流建议不要超过 90mA，从 GND 流出电流建议不要超过 90mA，整体流入/流出电流建议都不要超过 90mA。

14.3 I/O 的结构图

14.3.1 准双向口（弱上拉）

准双向口（弱上拉）输出类型可用作输出和输入功能而不需重新配置端口输出状态。这是因为当端口输出为 1 时驱动能力很弱，允许外部装置将其拉低。当引脚输出为低时，它的驱动能力很强，可吸收相当大的电流。准双向口有 3 个上拉晶体管适应不同的需要。

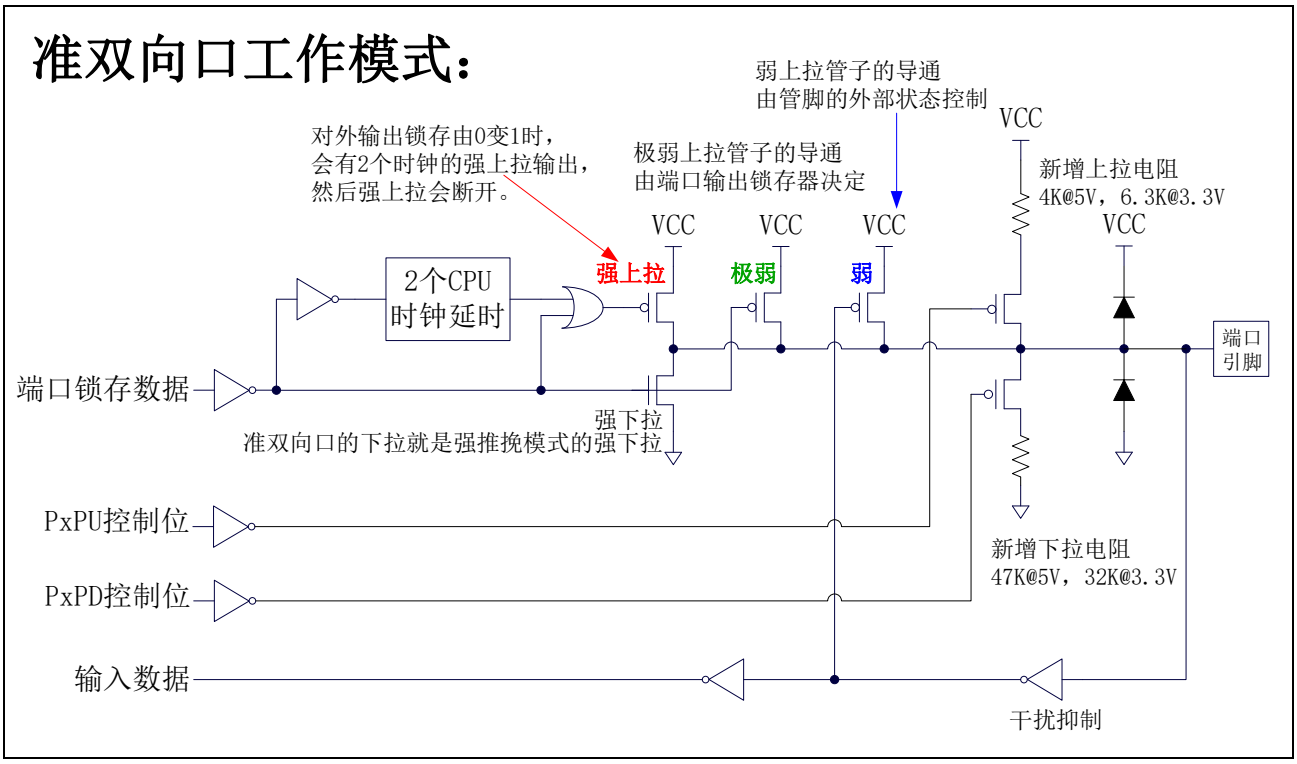
在 3 个上拉晶体管中，有 1 个上拉晶体管称为“弱上拉”，当端口寄存器为 1 且引脚本身也为 1 时打开。此上拉提供基本驱动电流使准双向口输出为 1。如果一个引脚输出为 1 而由外部装置下拉到低时，弱上拉关闭而“极弱上拉”维持开状态，为了把这个引脚强拉为低，外部装置必须有足够的灌电流能力使引脚上的电压降到门槛电压以下。对于 5V 单片机，“弱上拉”晶体管的电流约 250uA；对于 3.3V 单片机，“弱上拉”晶体管的电流约 150uA。

第 2 个上拉晶体管，称为“极弱上拉”，当端口锁存为 1 时打开。当引脚悬空时，这个极弱的上拉源产生很弱的上拉电流将引脚上拉为高电平。对于 5V 单片机，“极弱上拉”晶体管的电流约 18uA；对于 3.3V 单片机，“极弱上拉”晶体管的电流约 5uA。

第 3 个上拉晶体管称为“强上拉”。当端口锁存器由 0 到 1 跳变时，这个上拉用来加快准双向口由逻辑 0 到逻辑 1 转换。当发生这种情况时，强上拉打开约 2 个时钟以使引脚能够迅速地上拉到高电平。

准双向口（弱上拉）带有一个施密特触发输入以及一个干扰抑制电路。准双向口（弱上拉）读外部状态前,要先锁存为 ‘1’,才可读到外部正确的状态.

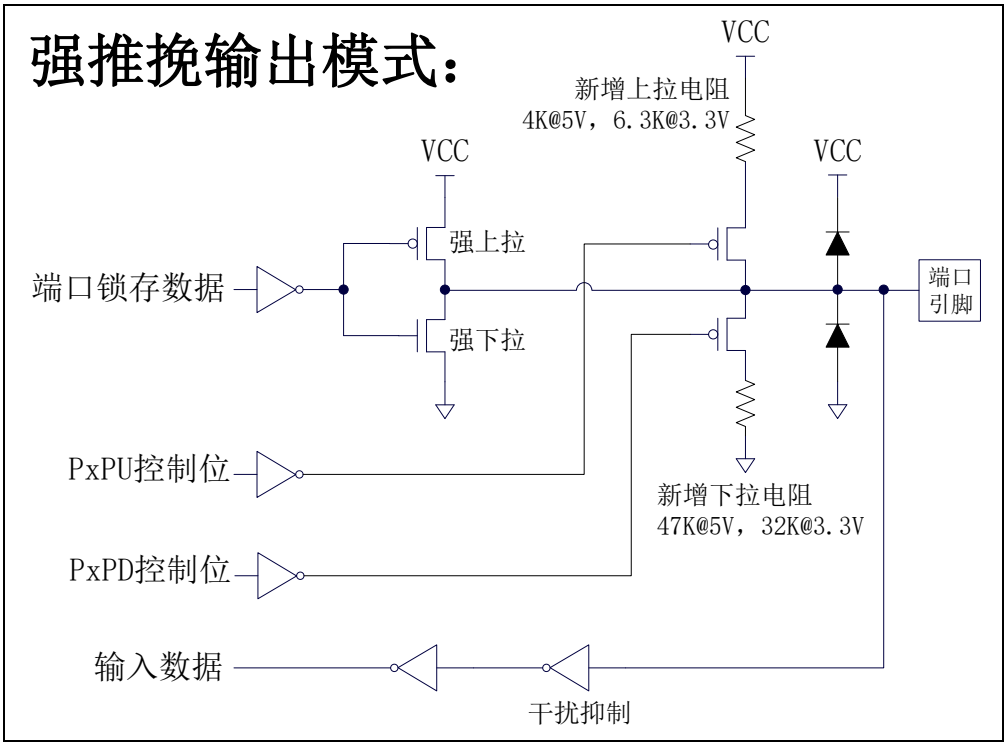
准双向口（弱上拉）输出如下图所示：



14.3.2　强推挽输出

强推挽输出配置的下拉结构与开漏模式以及准双向口的下拉结构相同，但当锁存器为 1 时提供持续的强上拉。推挽模式一般用于需要更大驱动电流的情况。

强推挽引脚配置如下图所示：

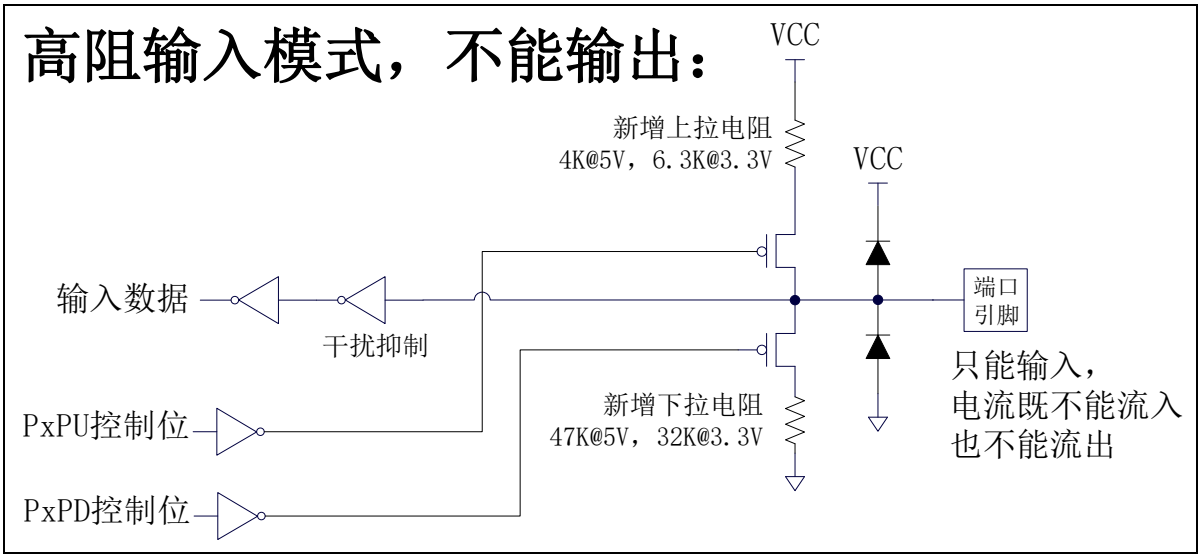


14.3.3　高阻输入

电流既不能流入也不能流出

输入口带有一个施密特触发输入以及一个干扰抑制电路

高阻输入引脚配置如下图所示：



14.3.4　开漏模式

＝＝＝【开漏工作模式】，对外设置输出为 1，等同于 【高阻输入】

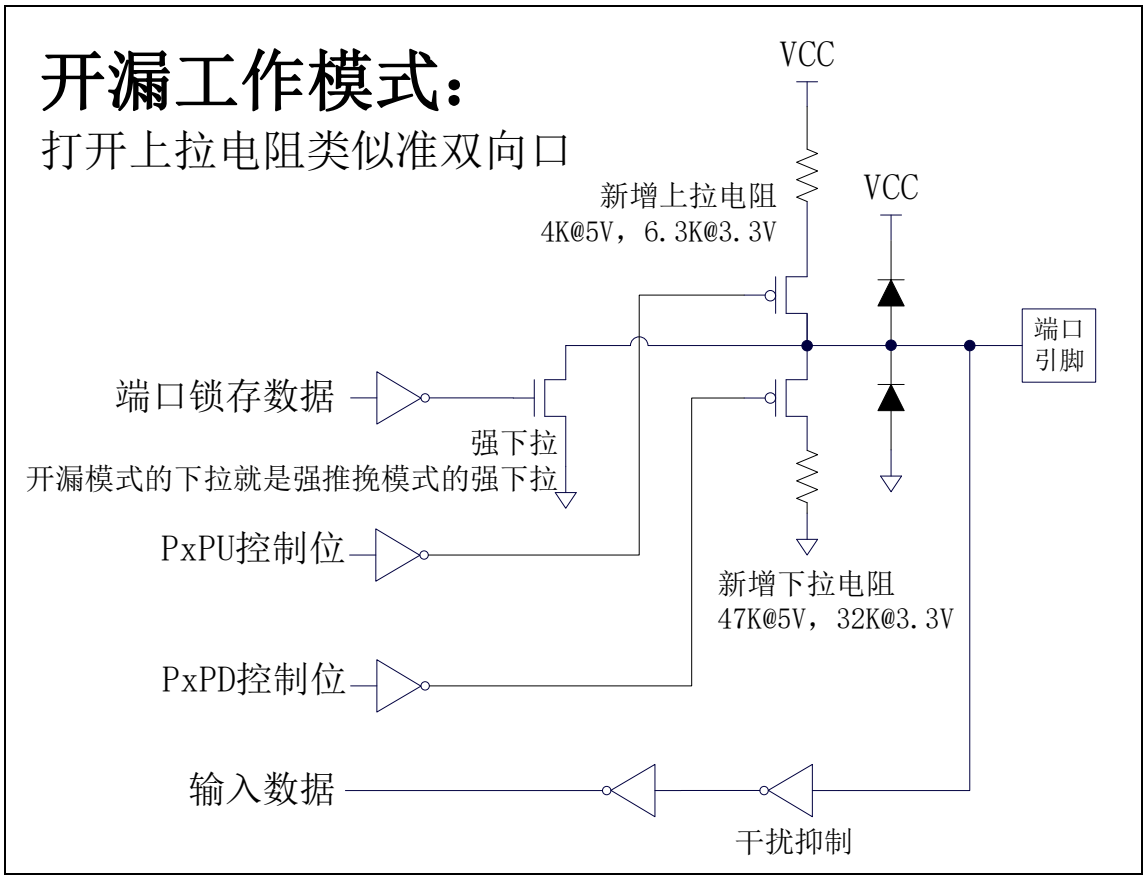
＝＝＝【开漏工作模式】，【打开内部上拉电阻 | 或外部加上拉电阻】，简单等同于 【准双向口】

开漏模式既可读外部状态也可对外输出（高电平或低电平）。如要正确读外部状态或需要对外输出高电平，需外加上拉电阻。

当端口锁存器为 0 时，开漏模式关闭所有上拉晶体管。当作为一个逻辑输出高电平时，这种配置方式必须有外部上拉，一般通过电阻外接到 VCC。如果外部有上拉电阻，开漏的 I/O 口还可读外部状态，即此时被配置为开漏模式的 I/O 口还可作为输入 I/O 口。这种方式的下拉与准双向口相同。

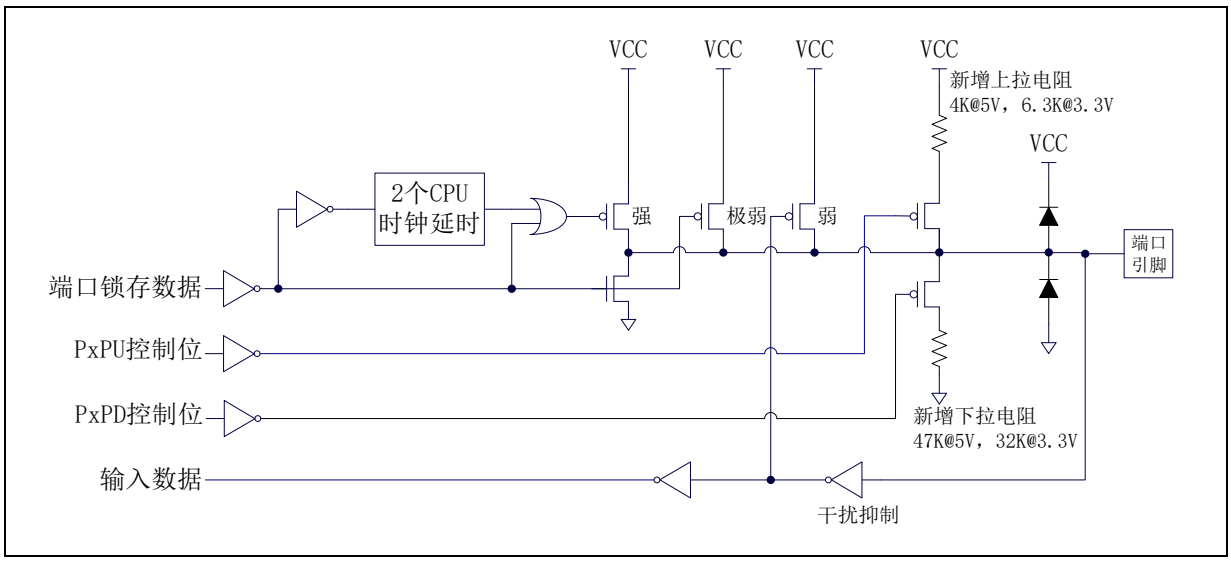
开漏端口带有一个施密特触发输入以及一个干扰抑制电路。

输出端口配置如下图所示：



14.3.5 新增 4K@5V/6.3K@3.3V 上拉电阻和 47K@5V/32K@3.3V 下拉电阻

Ai8051U 系列所有的 I/O 口内部均可使能一个 4K@5.0V/6.3K@3.3V 的上拉电阻和一个 47K@5V/32K@3.3V 的下拉电阻，由于制造原因，实际的电阻值可能有一定的误差



上拉电阻 / 电压	设计目标	实际测量
5.0V	4.0K	4.2K
3.3V	6.3K	5.8K

下拉电阻 / 电压	设计目标	实际测量
5.0V	47K	46K
3.3V	32K	35K

端口上拉电阻控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0PU	7EFE10H	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU
P1PU	7EFE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU
P2PU	7EFE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU
P3PU	7EFE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU
P4PU	7EFE14H	P47PU	P46PU	P45PU	P44PU	P43PU	P42PU	P41PU	P40PU
P5PU	7EFE15H	P57PU	P56PU	P55PU	P54PU	P53PU	P52PU	P51PU	P50PU
P6PU	7EFE16H	P67PU	P66PU	P65PU	P64PU	P63PU	P62PU	P61PU	P60PU
P7PU	7EFE17H	P77PU	P76PU	P75PU	P74PU	P73PU	P72PU	P71PU	P70PU

端口内部10K上拉电阻控制位（注：P3.0和P3.1口上的上拉电阻可能会略小一些）

0：禁止端口内部的 10K 上拉电阻

1：使能端口内部的 10K 上拉电阻

端口下拉电阻控制寄存器（PxPD）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0PD	7EFE40H	P07PD	P06PD	P05PD	P04PD	P03PD	P02PD	P01PD	P00PD
P1PD	7EFE41H	P17PD	P16PD	P15PD	P14PD	P13PD	P12PD	P11PD	P10PD
P2PD	7EFE42H	P27PD	P26PD	P25PD	P24PD	P23PD	P22PD	P21PD	P20PD
P3PD	7EFE43H	P37PD	P36PD	P35PD	P34PD	P33PD	P32PD	P31PD	P30PD
P4PD	7EFE44H	P47PD	P46PD	P45PD	P44PD	P43PD	P42PD	P41PD	P40PD
P5PD	7EFE45H	P57PD	P56PD	P55PD	P54PD	P53PD	P52PD	P51PD	P50PD
P6PD	7EFE46H	P67PD	P66PD	P65PD	P64PD	P63PD	P62PD	P61PD	P60PD
P7PD	7EFE47H	P77PD	P76PD	P75PD	P74PD	P73PD	P72PD	P71PD	P70PD

端口内部10K下拉电阻控制位

0：禁止端口内部的 10K 下拉电阻

1：使能端口内部的 10K 下拉电阻

14.3.6 如何设置 I/O 口对外输出速度

当用户需要 I/O 口对外输出较快的频率时，可通过加大 I/O 口驱动电流以及增加 I/O 口电平转换速度以达到提高 I/O 口对外输出速度

设置 PxSR 寄存器，可用于控制 I/O 口电平转换速度，设置为 0 时相应的 I/O 口为快速翻转，设置为 1 时为慢速翻转。

设置 PxDR 寄存器，可用于控制 I/O 口驱动电流大小，设置为 1 时 I/O 输出为一般驱动电流，设置为 0 时为强驱动电流

14.3.7 如何设置 I/O 口电流驱动能力

若需要改变 I/O 口的电流驱动能力，可通过设置 PxDR 寄存器来实现

设置 PxDR 寄存器，可用于控制 I/O 口驱动电流大小，设置为 1 时 I/O 输出为一般驱动电流，设置为 0 时为强驱动电流

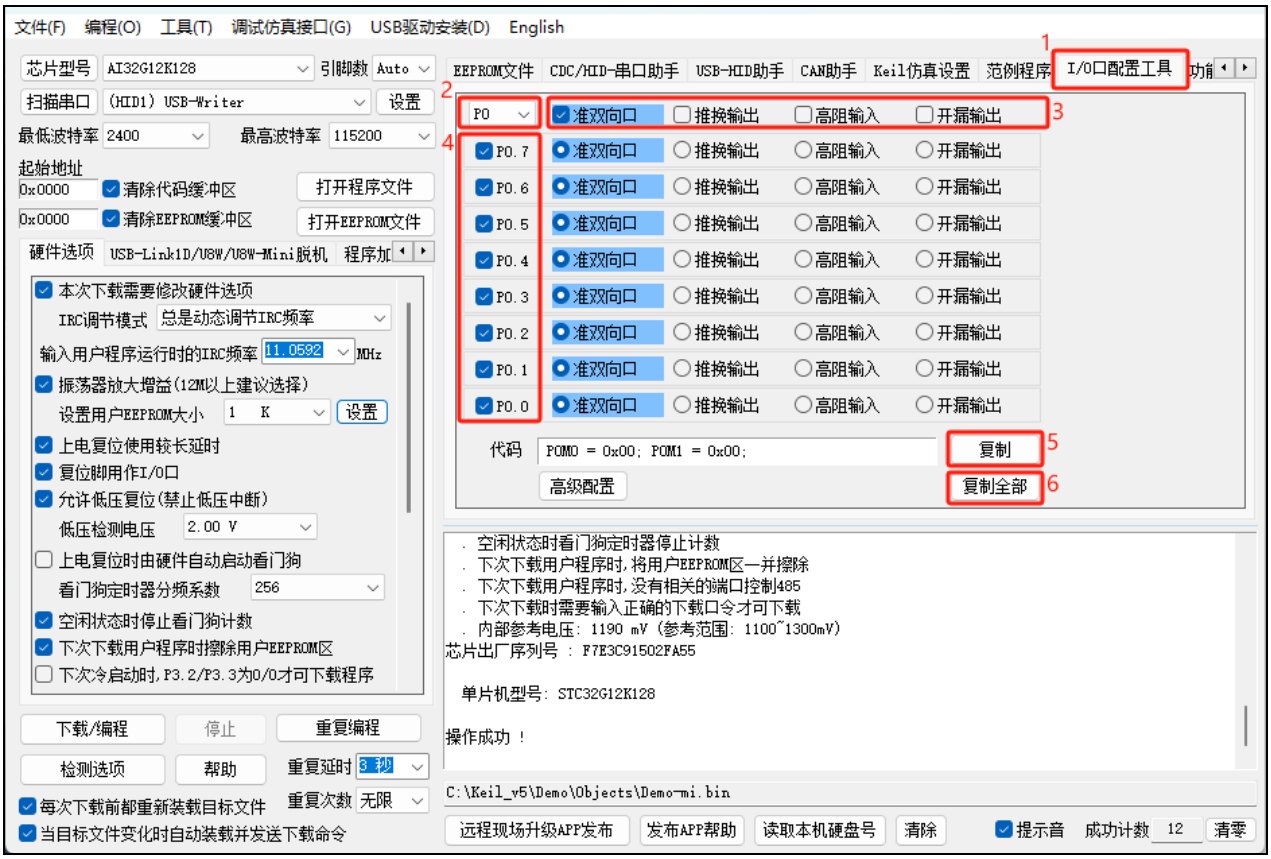
14.3.8 如何降低 I/O 口对外辐射

由于设置 PxSR 寄存器，可用于控制 I/O 口电平转换速度，设置 PxDR 寄存器，可用于控制 I/O 口驱动电流大小

当需要降低 I/O 口对外的辐射时，需要将 PxSR 寄存器设置为 1 以降低 I/O 口电平转换速度，同时需要将 PxDR 寄存器设为 1 以降低 I/O 驱动电流，最终达到降低 I/O 口对外辐射

14.4 AIapp-ISP | I/O 口配置工具

14.4.1 普通配置模式

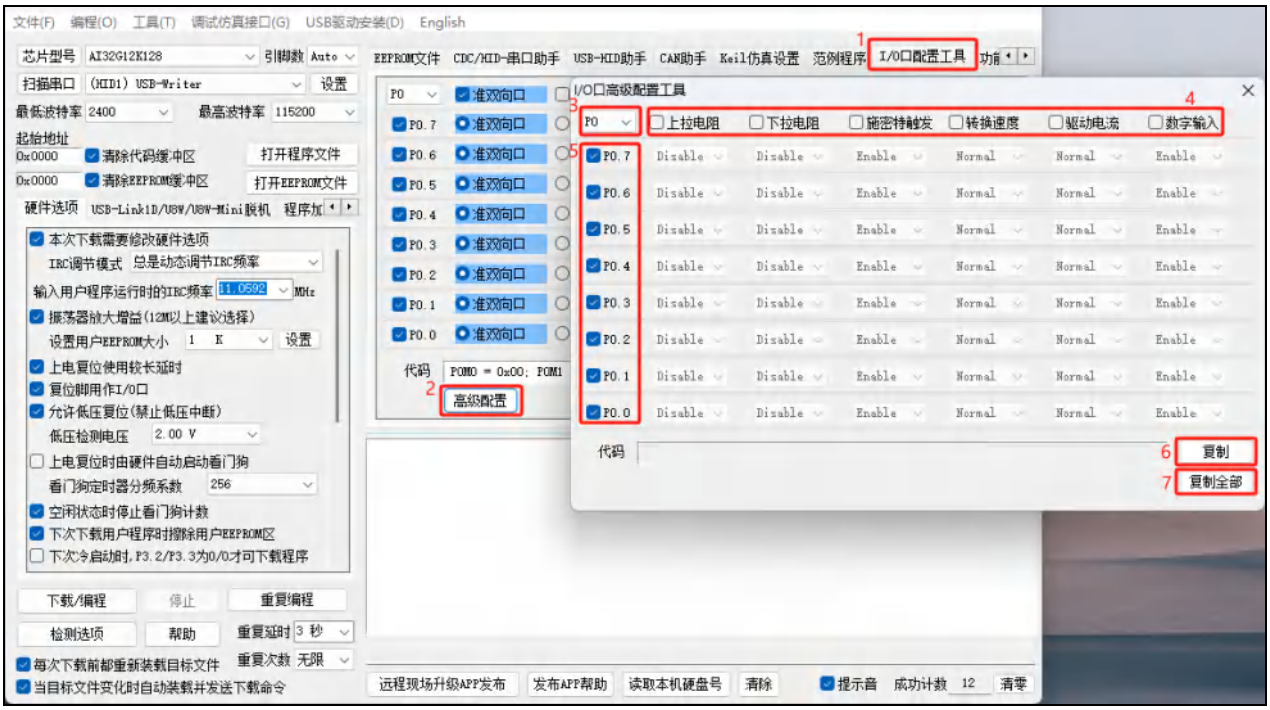


普通模式可以配置所有 I/O 的 4 中模式：

准双向口模式
推挽输出模式
高阻输入模式
开漏模式

①：在下载软件中选择“I/O 口配置工具”功能页，进入 I/O 口配置界面
②：选择需要配置的 I/O 口组，支持 P0~P7
③：整组 I/O 进行设置（将整组 P0/P1/…/P7 设置为）准双向口模式、推挽输出模式等
④：选择使能配置每组 I/O 中的端口
⑤：复制当前组 I/O 的配置代码
⑥：复制 P0~P7 口的全部配置代码

14.4.2　高级配置模式

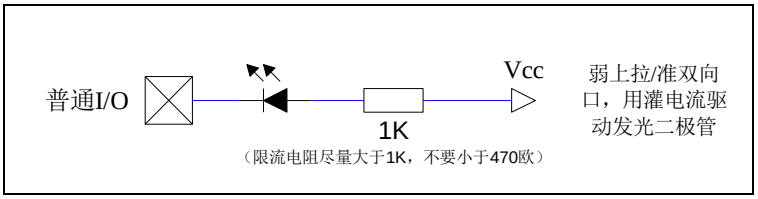


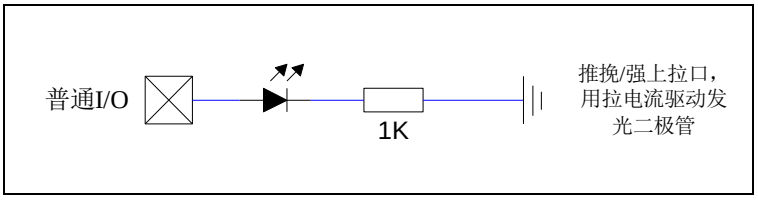
高级模式可以配置 I/O：

使能/关闭上拉电阻
使能/关闭下拉电阻
使能/关闭施密特触发功能
选择 I/O 转换速度
选择 I/O 口驱动电流
使能/关闭 I/O 的数组输入功能

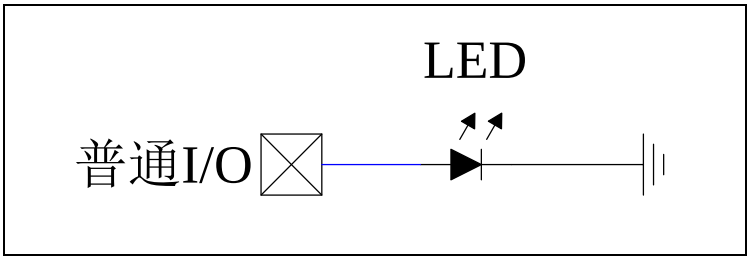
①：在下载软件中选择“I/O 口配置工具”功能页，进入 I/O 口配置界面
②：点击界面中的“高级配置”按钮进入 I/O 口高级配置界面
③：选择需要配置的 I/O 口组，支持 P0~P7
④：选择需要的配置模式
⑤：选择使能配置每组 I/O 中的端口
⑥：复制当前组 I/O 的配置代码
⑦：复制 P0~P7 口的全部配置代码

14.5　典型发光二极管控制电路



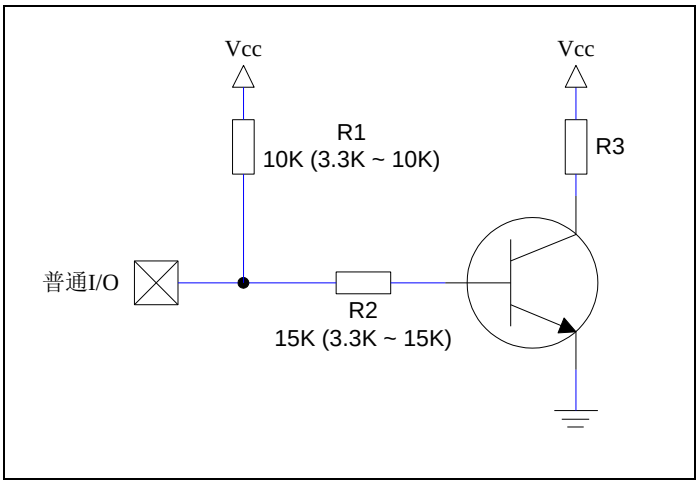


I/O 口，不接限流电阻，直接驱动 LED 灯，网友说不管亮度，只要省成本
要省这个 1K 限流电阻，那就用内部粗糙的 4K 上拉电阻代替
AI8H/Ai8051U 系列 I/O 口 上电是【高阻输入 + 4K 上拉电阻关闭】
可 I/O 口直接接 LED 灯到地，保持高阻输入：
打开内部 4K 上拉电阻，灯亮；关闭内部 4K 上拉电阻，灯灭

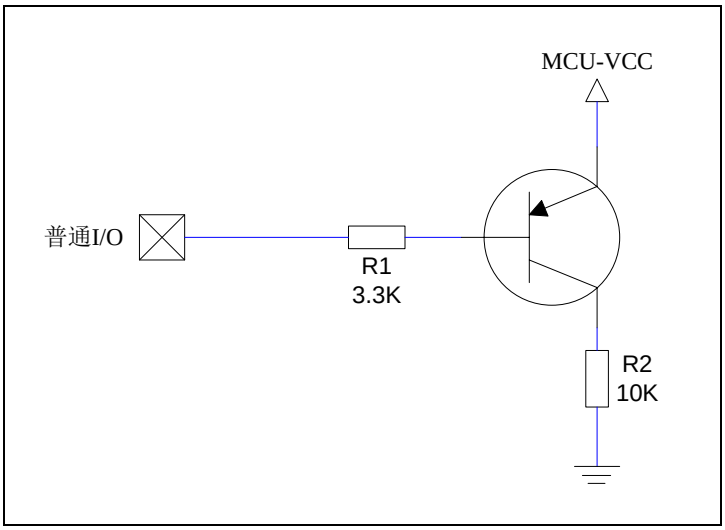


有网友说不管亮度，只要省成本，I/O 口，不接限流电阻，直接驱动 LED 灯
要省外部这个 1K~5K 的限流电阻，那就用内部粗糙的 4K 上拉电阻代替外部的限流电阻

14.6　一种典型三极管控制电路



上图中，如果使用弱上拉控制，建议加上拉电阻 R1(3.3K~10K)，如果不加上拉电阻 R1(3.3K~10K)，建议 R2 的值在 15K 以上，或用强推挽输出。

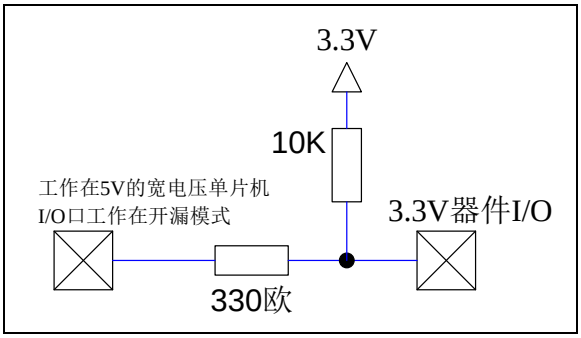


上图中，不接限流电阻 R1，I/O 口直接控制 PNP 三级管，网友说只要省成本
要省这个 3.3K 限流电阻，那就用内部粗糙的 47K 下拉电阻代替
Ai8051U 的 I/O 口上电是：【高阻输入+ 4K 上拉电阻关闭+ 47K 下拉电阻关闭】
可 I/O 口直接控制 PNP 三级管，保持高阻输入：
===打开内部 47K 下拉电阻，PNP 三级管导通
===关闭内部 47K 下拉电阻，PNP 三级管关闭

14.7　混合电压供电系统 3V/5V 器件 I/O 口互连

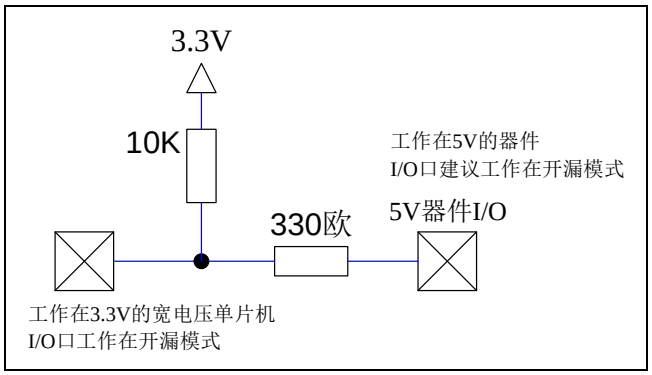
宽电压单片机工作在 5V 时:

如需要直接连接 3.3V 器件时，为防止 3.3V 器件承受不了 5V，可将相应的单片机 I/O 口先串一个 330Ω 的限流电阻到 3.3V 器件 I/O 口，程序初始化时将单片机的 I/O 口设置成开漏工作模式，断开内部上拉电阻，相应的 3.3V 器件 I/O 口外部加 10K 上拉电阻到 3.3V 器件的 Vcc，这样高电平是 3.3V，低电平是 0V，输入输出一切正常。宽电压单片机 2.2V 以上肯定是高电平。



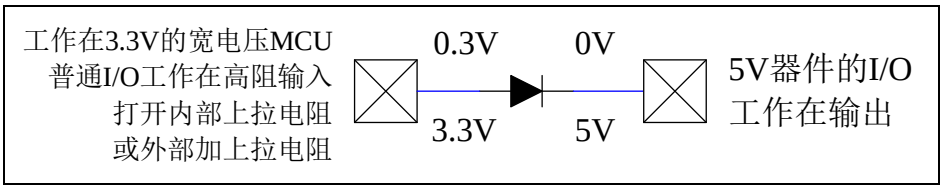
宽电压 MCU 如工作在 3.3V 时，如需要连接到 5V 器件:

1、如果对方 5V 器件可以工作在开漏模式:

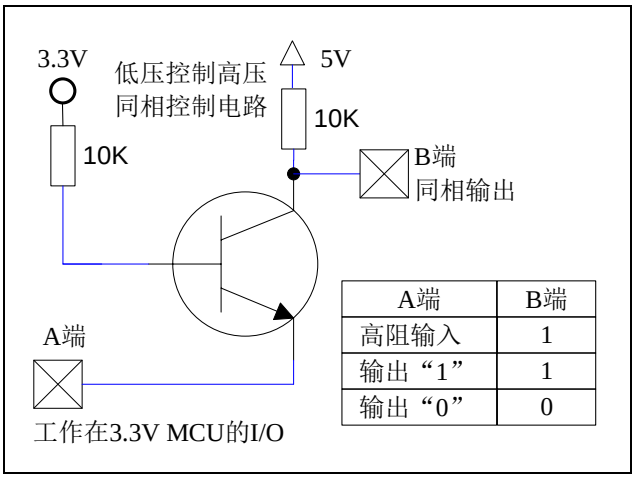


2、如是连接到 5V 器件的高阻输入，则可以直接相连，或串个 300 欧电阻相连。工作在 3.3V 的宽电压单片机的 I/O 可以工作在强推挽输出模式，或者工作在准双向口/弱上拉模式。

3、如果对方是 5V 输出，我方可工作在高阻输入模式，打开内部上拉电阻，外部串接一个锗二极管隔离，隔离高压 5V。外部电压高于单片机工作电压时锗二极管截止，I/O 口因内部上拉到高电平，所以读 I/O 口状态是高电平；外部电压为低时隔离的锗二极管导通，I/O 口被钳位在 0.3V，小于 0.8V 时单片机读 I/O 口状态是低电平。锗二极管可以使用 1N5817/1N5819，不要使用硅二极管。2.2V 以上是高电平，0.8V 以下是低电平。

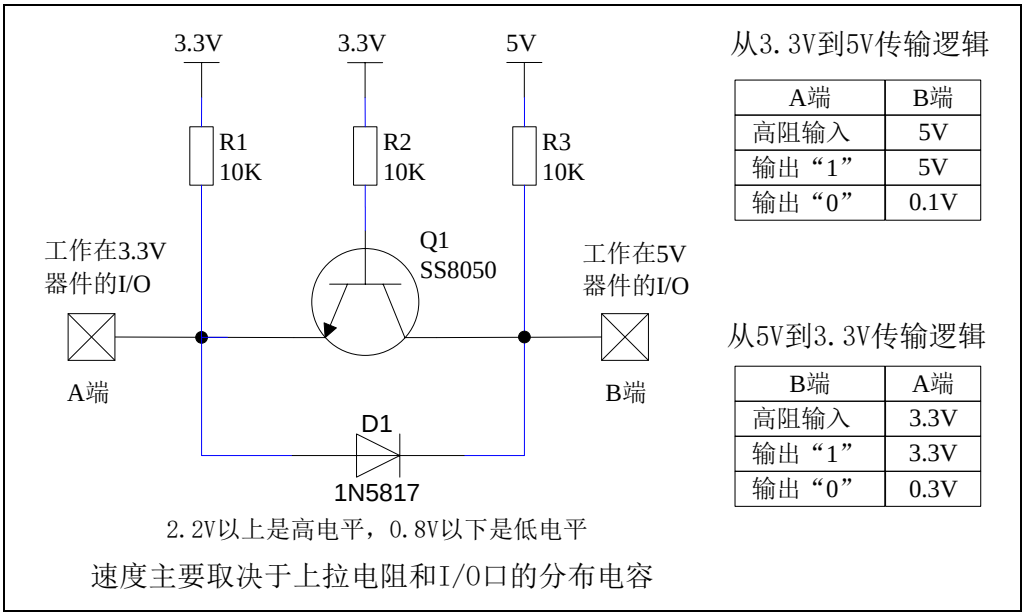


4、宽电压单片机工作在 3.3V 时，如需要控制 5V 器件，还可以用下图同相控制电路：



A端	B端
高阻输入	1
输出“1”	1
输出“0”	0

5、工作在 3.3V 的器件和工作在 5V 的器件打交道，还可以用如下【三极管+二极管】双向电路：



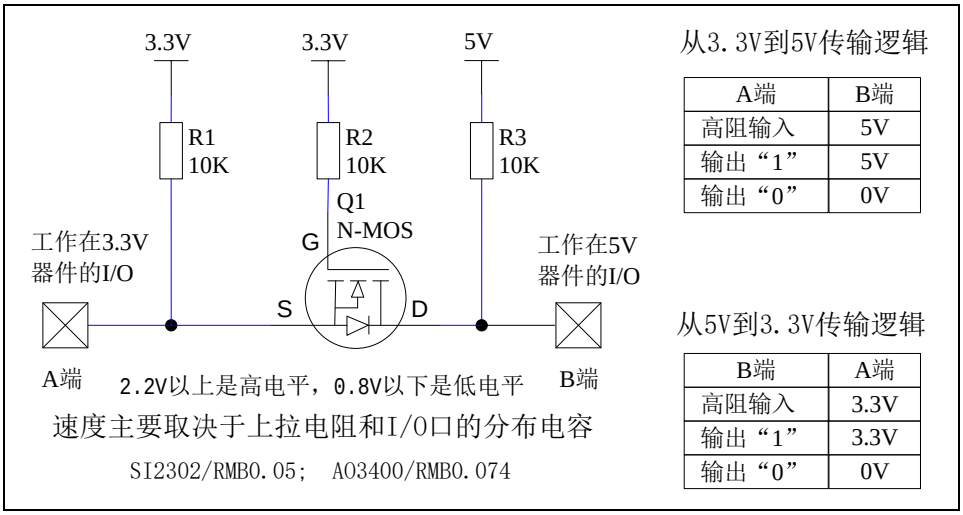
从3.3V到5V传输逻辑

A端	B端
高阻输入	5V
输出“1”	5V
输出“0”	0.1V

从5V到3.3V传输逻辑

B端	A端
高阻输入	3.3V
输出“1”	3.3V
输出“0”	0.3V

6、工作在 3.3V 的器件和工作在 5V 的器件打交道，还可以用如下 MOS 管双向电路：



从3.3V到5V传输逻辑

A端	B端
高阻输入	5V
输出“1”	5V
输出“0”	0V

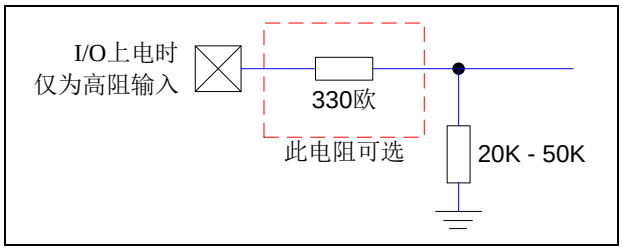
从5V到3.3V传输逻辑

B端	A端
高阻输入	3.3V
输出“1”	3.3V
输出“0”	0V

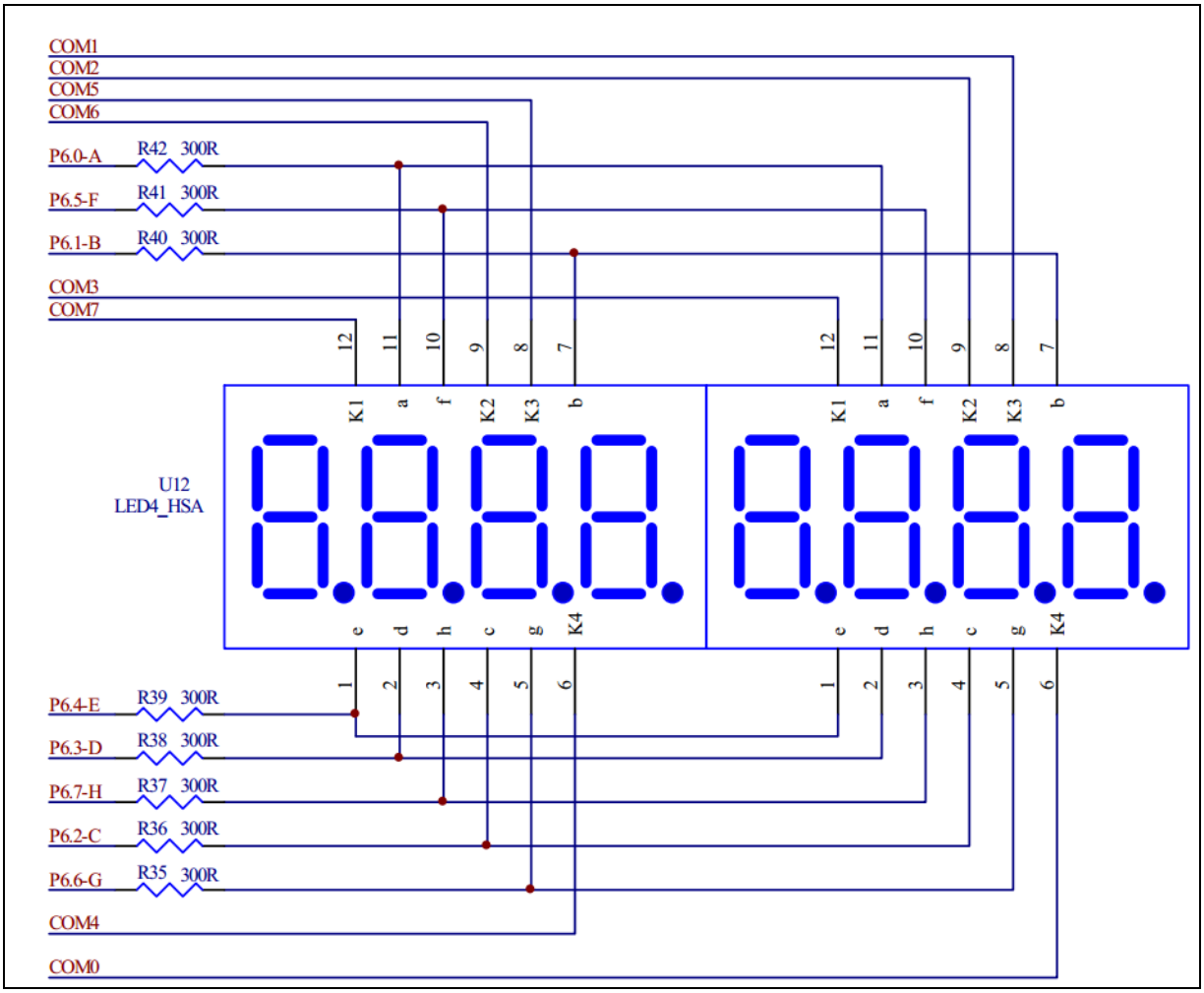
14.8　如何让 I/O 口上电复位时为低电平

普通 8051 单片机上电复位时普通 I/O 口为弱上拉(准双向口)高电平输出，而很多实际应用要求上电时某些 I/O 口为低电平输出，否则所控制的系统(如马达)就会误动作，现 Ai8051U/8H/32G 系列单片机上电复位后 I/O 口为高阻输入。如外部加下拉电阻【20K－50K】，则上电后就是低电平。如外部加上拉电阻【20K－50K】，则上电后就是高电平。由于 I/O 口可以工作在准双向口/弱上拉，又可以工作在强推挽模式，后续可以很容易拉高或拉低，很简单的解决了这个问题。

如下图可在单片机 I/O 口上加一个下拉电阻(20K 左右)，这样上电复位时，除了下载口 P3.0 和 P3.1 为弱上拉(准双向口)外，其他 I/O 口均为高阻输入模式，而外部加下拉电阻，所以该 I/O 口上电复位时外部为低电平。如果要将此 I/O 口驱动为高电平，可将此 I/O 口设置为强推挽输出，而强推挽输出时，I/O 口驱动电流可达 20mA，故肯定可以将该口驱动为高电平输出。

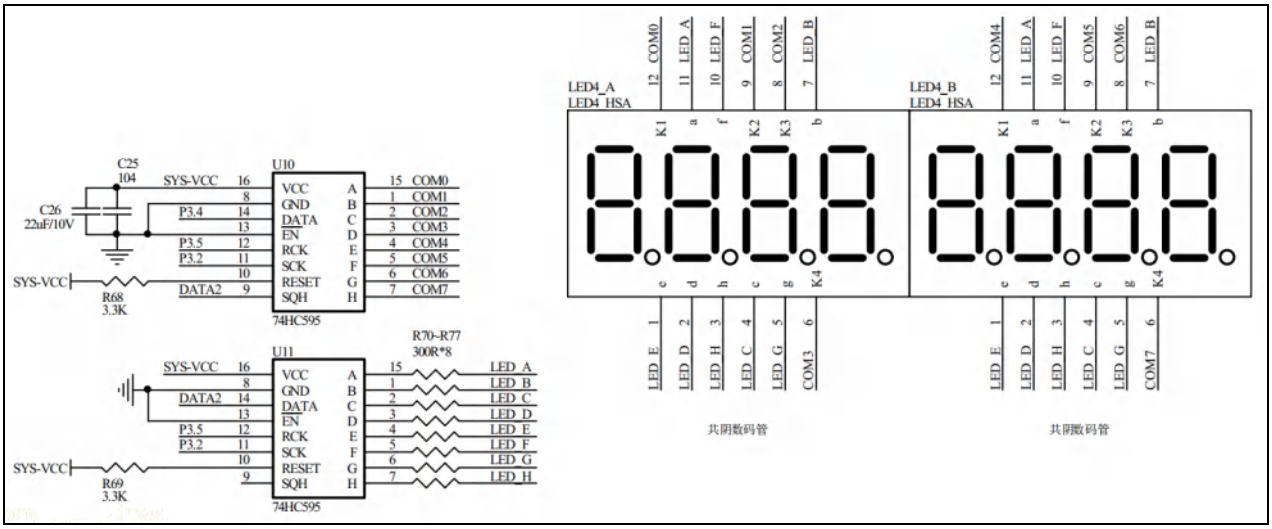


14.9　I/O 口直接驱动 LED 数码管应用线路图

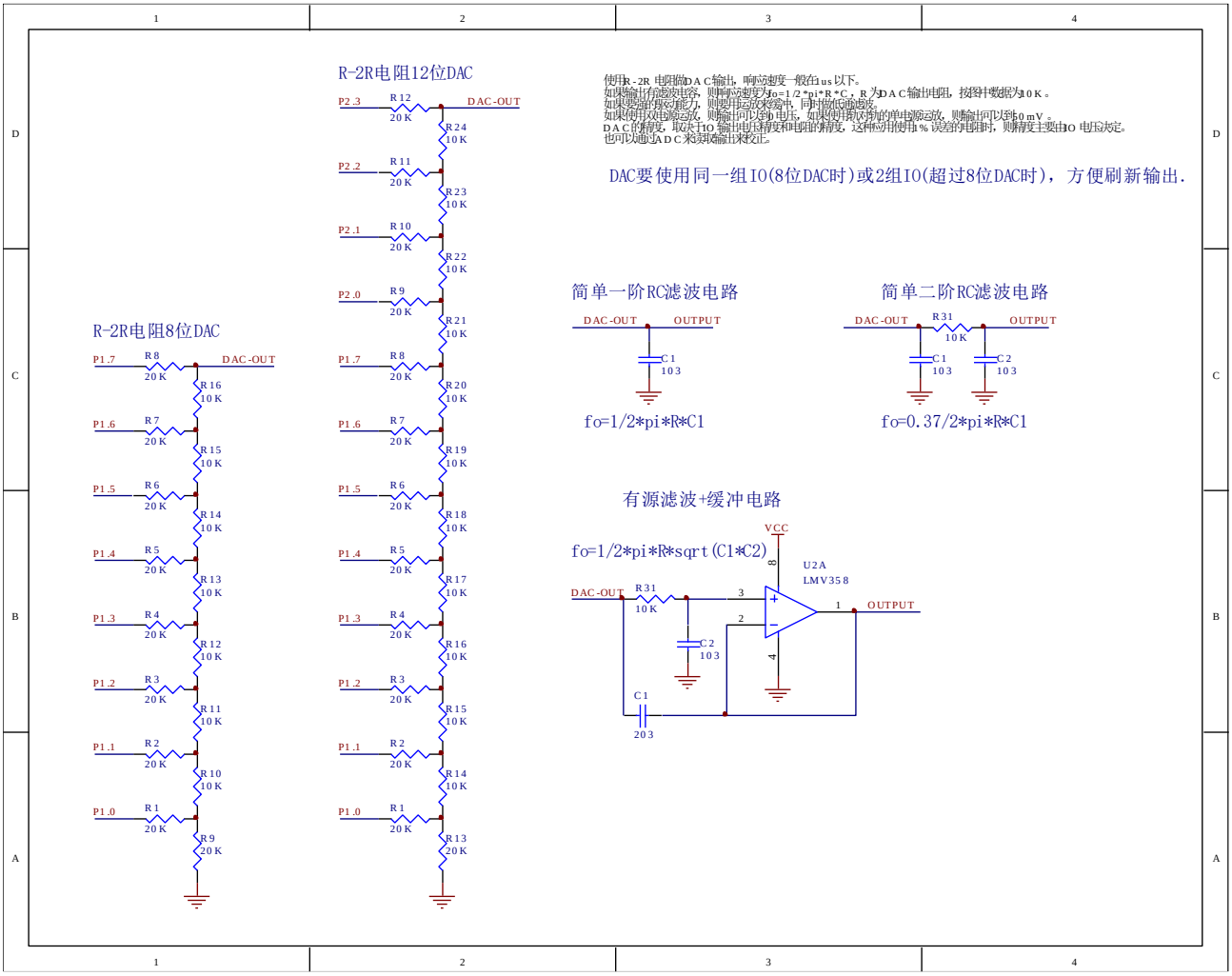


注：上图的 COM0 ～ COM7 公共端可以使用 I/O 口来控制（例如：P7.0~P7.7），不需要加限流电阻。段码限流电阻使用 470 欧 ～ 1K，大些好

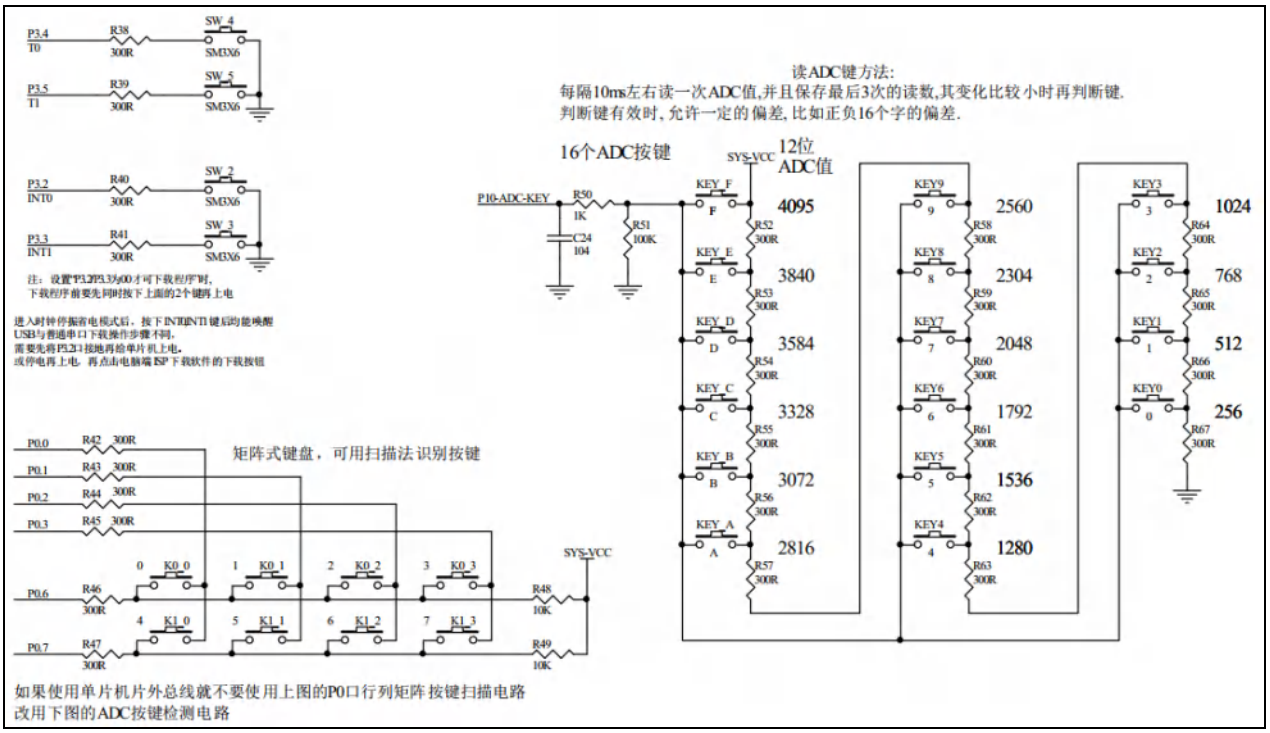
14.10 利用 74HC595 驱动 8 个数码管(串行扩展,3 根线)，节省 I/O



14.11　使用 I/O 和 R-2R 电阻分压实现 DAC 的经典线路图



14.12　利用 ADC 做按键扫描节省 I/O 口



14.13　范例程序

14.13.1　端口模式设置（适用于所有的 I/O）

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件
#include "intrins.h"                          //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void main()
{
    EAXFR = 1;                                //允许访问扩展的特殊寄存器，XFR
                                              //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                            //允许访问扩展的特殊寄存器，XFR
                                              //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                 //设置取程序代码等待时间，
                                              //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                //设置访问片内的 xdata 速度，
                                              //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;                              //设置 P0.0~P0.7 为双向口模式
    P0M1 = 0x00;
    P1M0 = 0xff;                              //设置 P1.0~P1.7 为推挽输出模式
    P1M1 = 0x00;
    P2M0 = 0x00;                              //设置 P2.0~P2.7 为高阻输入模式
    P2M1 = 0xff;

    P3M0 = 0xcc;                              //设置 P3.0~P3.1 为双向口模式
    P3M1 = 0xf0;                              //设置 P3.2~P3.3 为推挽输出模式
                                              //设置 P3.4~P3.5 为高阻输入模式
                                              //设置 P3.6~P3.7 为开漏模式

    while (1);
}
```

14.13.2　双向口读写操作（适用于所有的 I/O）

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件
#include "intrins.h"                          //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void main()
{
    EAXFR = 1;                                //允许访问扩展的特殊寄存器，XFR
                                              //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                            //允许访问扩展的特殊寄存器，XFR
                                              //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                 //设置取程序代码等待时间，
                                              //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                //设置访问片内的 xdata 速度，
                                              //赋值为 0 表示用最快速度访问，不增加额外的等待时间
```

```
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P0M0 = 0x00;                    //设置 P0.0~P0.7 为双向口模式
    P0M1 = 0x00;

    P0 = 0xff;                      //P0 口全部输出高电平
    _nop_();                        //
    _nop_();                        //
    P0 = 0x00;                      //P0 口全部输出低电平
    _nop_();                        //
    _nop_();                        //

    P00 = 1;                        //P0.0 口输出高电平
    _nop_();                        //
    _nop_();                        //
    P00 = 0;                        //P0.0 口输出低电平
    _nop_();                        //
    _nop_();                        //

    P00 = 1;                        //读取端口前先使能内部弱上拉电阻
    _nop_();                        //等待两个时钟
    _nop_();                        //
    CY = P00;                       //读取端口状态

    while (1);
}
```

14.13.3　打开 I/O 口内部上拉电阻（适用于所有的 I/O）

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                //头文件见下载软件
#include "intrins.h"                //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void main()
{
    EAXFR = 1;                      //允许访问扩展的特殊寄存器，XFR
                                    //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                  //允许访问扩展的特殊寄存器，XFR
                                    //(8 位模式请使用这句，注释上一句)
    WTST = 0;                       //设置取程序代码等待时间，
                                    //赋值为 0 表示不等待，程序以最快速度运行
```

```
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P0PU = 0x0f;                                //打开 P0.0~P0.3 口的内部上拉电阻

    P1PU = 0xf0;                                //打开 P1.4~P1.7 口的内部上拉电阻

    while (1);
}
```

14.13.4　将端口设置成传统 8051 I/O 模式（适用于所有的 I/O）

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                            //头文件见下载软件
#include "intrins.h"                            //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void main()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;                                //设置 P0.0~P0.7 为双向口模式
    P0M1 = 0x00;
    P1M0 = 0x00;                                //设置 P1.0~P1.7 为双向口模式
    P1M1 = 0x00;
    P2M0 = 0x00;                                //设置 P2.0~P2.7 为双向口模式
    P2M1 = 0x00;
    P3M0 = 0x00;                                //设置 P3.0~P3.7 为双向口模式
    P3M1 = 0x00;
    P4M0 = 0x00;                                //设置 P4.0~P4.7 为双向口模式
    P4M1 = 0x00;
    P5M0 = 0x00;                                //设置 P5.0~P5.7 为双向口模式
    P5M1 = 0x00;

//  delay();                                    //如需要立即读取外部口的状态建议延时 1ms，否则
                                                //如果外部口是浮空状态则可能无法正确读取状态
```

```
    while (1);
}
```

14.14 深圳大学--I/O 口实验：I/O 口读取按键状态并点灯

14.14.1 实验要求

可以通过正确设置 I/O 口模式来读取实验箱上按键的电平，然后根据对应按键的电平来设置 LED 灯的点亮/熄灭状态

14.14.2 简单介绍

Ai8051U 系列单片机所有的 I/O 口均有 4 种工作模式：

- ✓ 准双向口/弱上拉（标准 8051 输出口模式）
- ✓ 强推挽输出/强上拉
- ✓ 高阻输入（电流既不能流入也不能流出）
- ✓ 开漏模式（如果打开内部上拉电阻，就相当于准双向口；如果不加上拉并对外置 1，就相当于高阻输入）

14.14.3 实验建议

1）高阻输入模式，内部配有 6K 上拉电阻；
为了读取外部状态，需要打开内部的 6K 上拉电阻。

2）开漏模式，内部同样配有 6K 上拉电阻；
打开内部 6K 上拉电阻后，其功能与准双向口/弱上拉模式相似。

3）读取 I/O 口的外部状态时，可以采用以下三种模式：

准双向口/弱上拉模式:

===要确保对外输出锁存为 1，关闭对内部对外强输出低的 MOS 管，
才能准确读到外部状态
===如对外输出锁存为 0，则外部被内部对外强输出低的 MOS 管拉低，
读到的是低电平，不能准确读到外部来的高输入/被内部拉低了

高阻输入模式，每个 I/O 口都有独立的 6K 上拉电阻:
===为了准确读取外部状态，需要打开内部的 6K 上拉电阻，或外部有上拉能力
===内部的对外输出锁存器对此模式无效，对外无输出高/低的能力
===电流流不进来，也留不出去，
高阻输入，无输出能力，不影响外部来的的输入状态

开漏模式，每个 I/O 口都有独立的 6K 上拉电阻：
启用内部 6K 上拉电阻后，其功能与准双向口/弱上拉模式相似
===要确保对外输出锁存为 1，关闭对内部对外强输出低的 MOS 管，才能准确读到外部状态
===如对外输出锁存为 0，则外部被内部对外强输出低的 MOS 管拉低，
读到的是低电平，不能准确读到外部来的高输入/被内部拉低了

4）I/O 口对外输出高或低电平时，可以采用以下三种模式：

- ➢ 准双向口/弱上拉模式：
  对外输出低时，相当于强推挽输出低

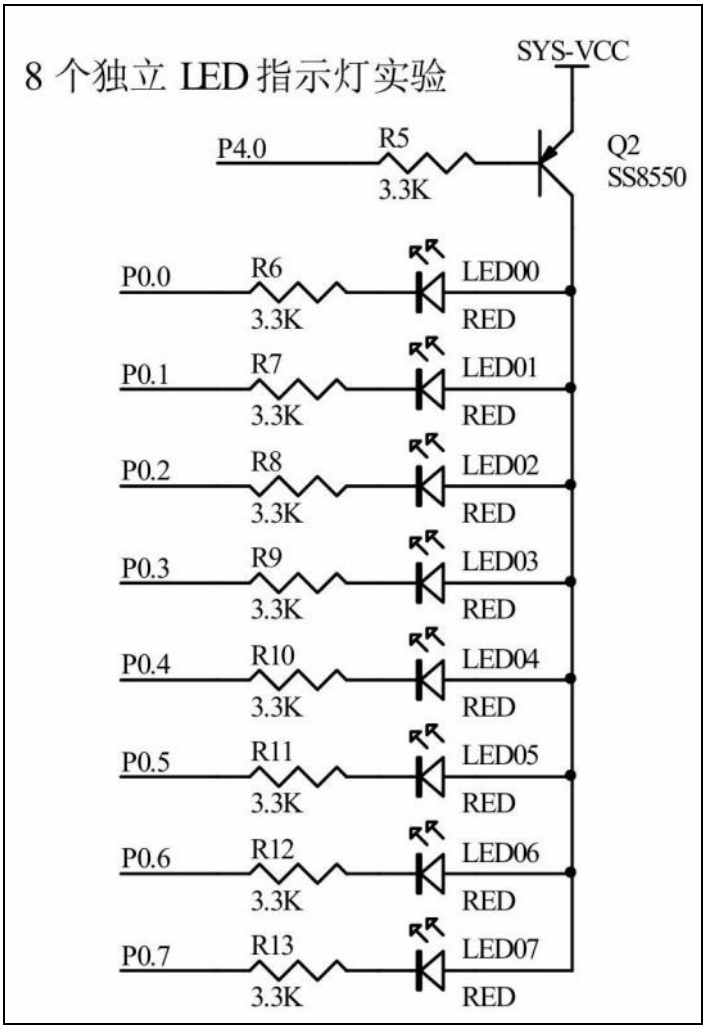
- 强推挽输出模式；
- 开漏模式；

  在对外输出低电平时，表现为强推挽输出的低电平状态。
  在对外输出高电平时，表现为高阻输入状态，此时电流流不进来也流不出去。
  如果用来控制 LED 灯，因为无电流流动，所以 LED 灯不亮。

5）P3.2/P3.3/P3.4/P3.5 端口均有按键，按下按键可接地，建议：
读取这些 I/O 口的状态，并将结果输出到 P0.2/P0.3/P0.4/P0.5 对应的指示灯上。

14.14.4 原理图

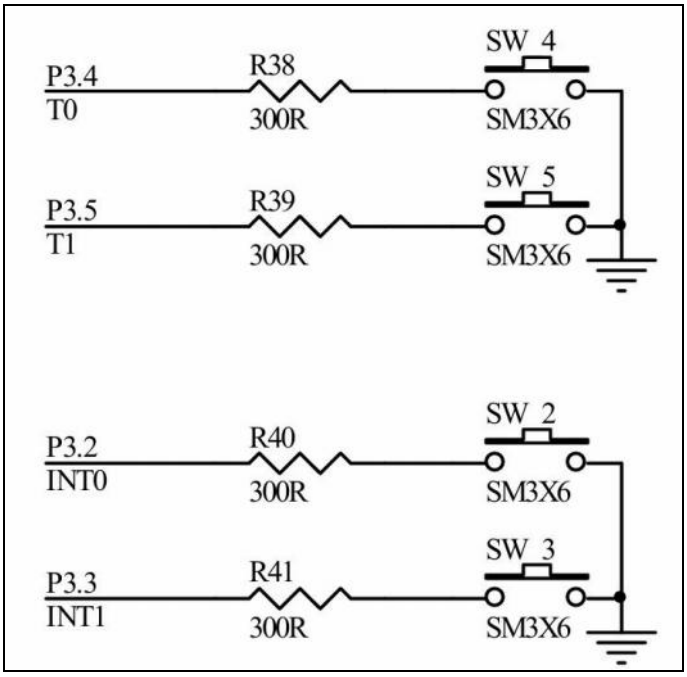
P0 口 都有单独的指示灯



I/O 口 P4.0 控制 PNP 三极管的通断，控制这 8 个灯的整体供电。
如果 P4.0 输出 1，则 PNP 三极管不导通，这 8 个灯都不会获得电源，都不会有电流流过，都不会被点亮。
如果 P4.0 输出 0，则 PNP 三极管导通，这 8 个灯才有可能会流过电流，才有可能会被点亮。
如 P4.0 输出 0，PNP 三极管导通，特定 LED 灯的控制 I/O 口如输出 0，则该 LED 灯会有电流流过，会被点亮。
如 P0.0 输出 0，LED00 灯就会被点亮；如 P0.0 输出 1，LED00 灯就会熄灭；
如 P0.1 输出 0，LED01 灯就会被点亮；如 P0.1 输出 1，LED01 灯就会熄灭；
如 P0.2 输出 0，LED02 灯就会被点亮；如 P0.2 输出 1，LED02 灯就会熄灭；
如 P0.3 输出 0，LED03 灯就会被点亮；如 P0.3 输出 1，LED03 灯就会熄灭；

如 P0.4 输出 0，LED03 灯就会被点亮；如 P0.4 输出 1，LED03 灯就会熄灭；
如 P0.5 输出 0，LED03 灯就会被点亮；如 P0.5 输出 1，LED03 灯就会熄灭；
如 P0.6 输出 0，LED03 灯就会被点亮；如 P0.6 输出 1，LED03 灯就会熄灭；
如 P0.7 输出 0，LED03 灯就会被点亮；如 P0.7 输出 1，LED03 灯就会熄灭；

如 P0 口对外全部送 0，则这 8 个灯全亮；
如 P0 口对外全部送 1，则这 8 个灯全灭；
如果 P4.0 输出 1，则 PNP 三极管不导通，不管 P0 口对外全部送 0 或送 1，这 8 个灯都不会有电流流过，都不会亮。P4.0 控制 PNP 三极管，作为这 8 个 LED 指示灯的总电源开关。



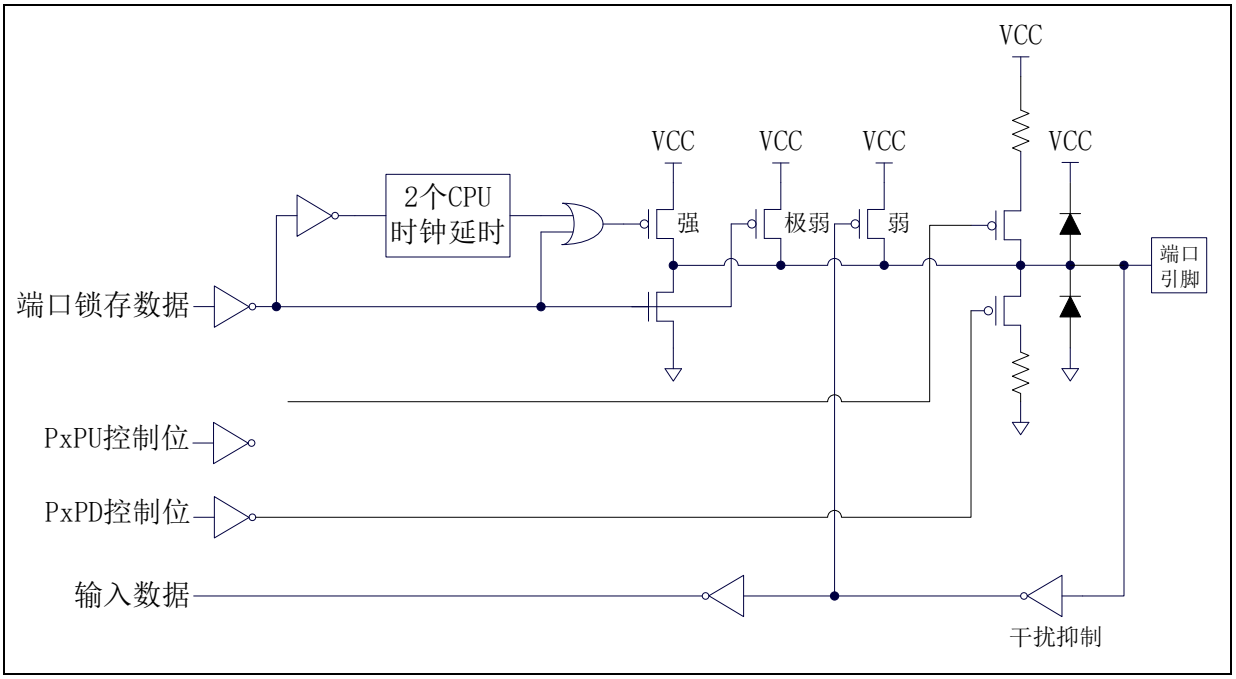
14.14.5　I/O 口工作模式和内部上拉/下拉电阻介绍

a) 内部上拉电阻和外部上拉/下拉电阻的部分区别：

外部上拉电阻随系统电源从 0V 上升到 MCU-VCC，到 3.3V 或 5V，这个时间是足够的长，系统上电复位后到初始化运行用户程序时，外部上拉电阻已将外部拉高到高电平。
内部上拉电阻是运行用户程序时才使能打开的，由于外部电路的分布电容效应，要等待 0.7RC 时间（R:电阻，C:电容）才能将外部拉高到 0.7MCU-VCC 高电平。建议将需要打开的内部上拉电阻使能后，直接多等待一会，如等待 1ms 后，就可认为外部已被拉高。

b) 准双向口介绍

准双向口（弱上拉）输出类型可用作输出和输入功能而不需重新配置端口输出状态。这是因为当端口输出为 1 时驱动能力很弱，允许外部装置将其拉低。当引脚输出为低时，它的驱动能力很强，可吸收相当大的电流。准双向口输出低电平的驱动能力和强推挽的驱动能力是一样的。此处的上拉电阻和下拉电阻是独立的，和 I/O 口 4 种功能模式无关。

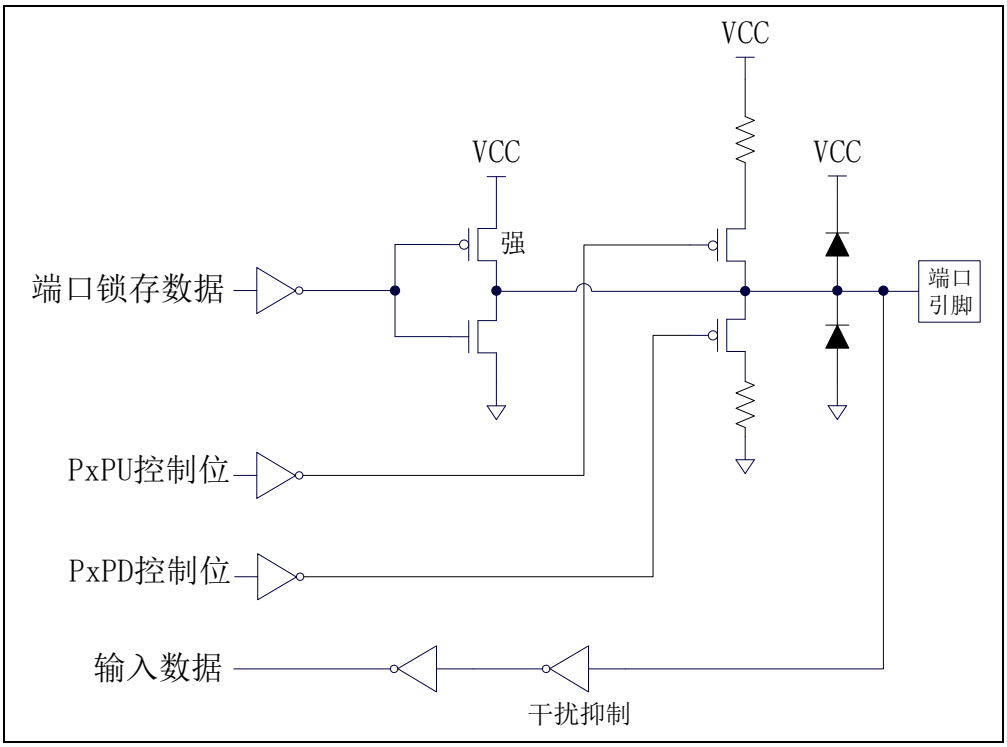


上拉电阻 / 电压	设计目标	实际测量
5.0V	4.0K	4.2K
3.3V	6.3K	5.8K

下拉电阻 / 电压	设计目标	实际测量
5.0V	47K	46K
3.3V	32K	35K

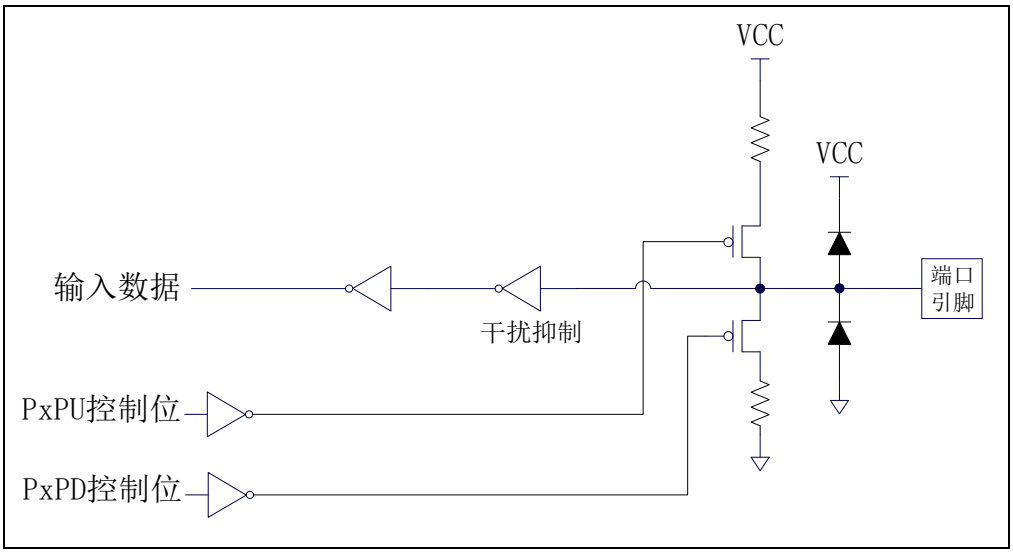
c) 强推挽输出

强推挽输出配置的下拉结构与开漏模式以及准双向口的下拉结构相同。
当对外输出锁存器设置为 0 时，强推挽的输出配置为强下拉结构，与开漏模式及准双向口的下拉是同一个 MOS 管，驱动能力是一样的。当对外输出锁存器设置为 1 时，对外输出强上拉。强推挽模式一般用于需要大驱动电流的情况。



d) 高阻输入

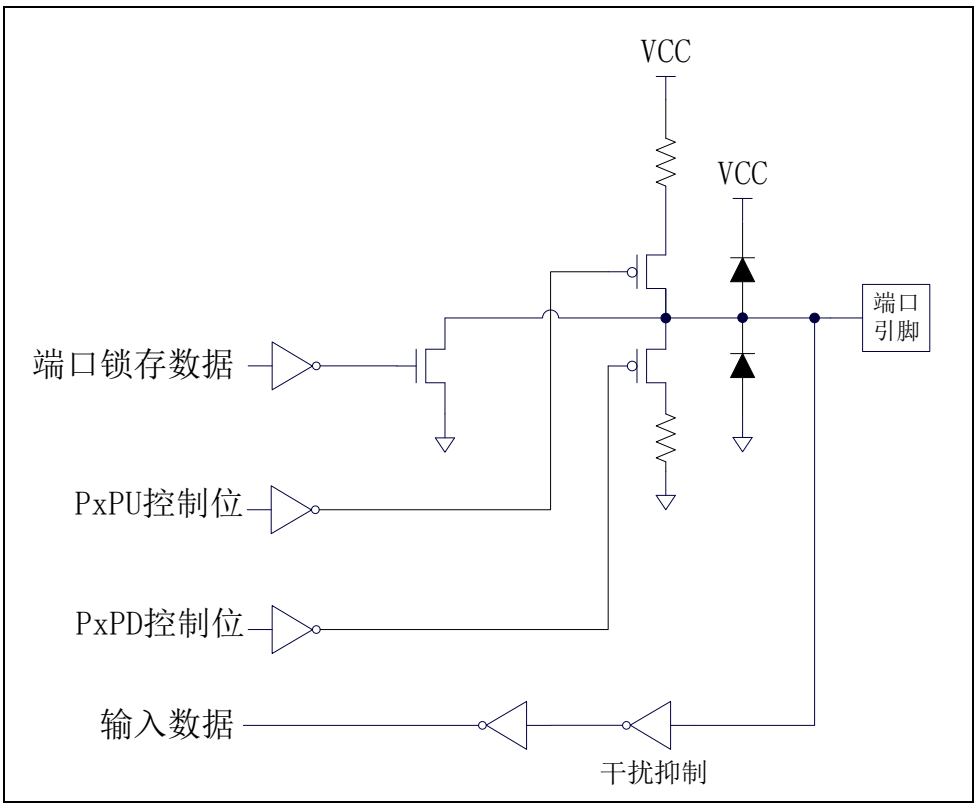
电流既不能流入也不能流出
输入口带有一个施密特触发输入以及一个抗干扰抑制电路



e) 开漏模式

开漏工作模式，对外设置输出为 1，等同于高阻输入
开漏工作模式，打开内部上拉电阻 | 或外部加上拉电阻，简单等同于 准双向口

开漏模式如果打开内部上拉电阻或外接上拉电阻，可准确的读到外部状态。开漏模式对外可直接输出 0，但如果要输出 1，需打开内部上拉电阻或外接上拉电阻。



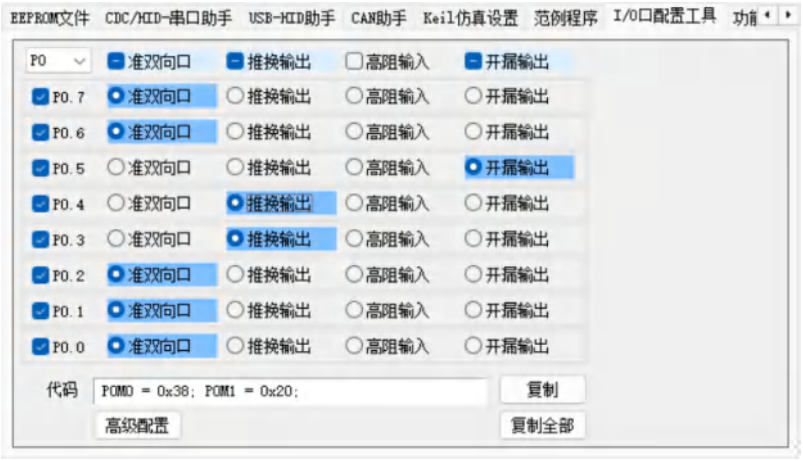
14.14.6　实验 1：位操作，用 USB 直接仿真来调试

单独读这几个独立 I/O 的状态，【P3.2 / P3.3 / P3.4 / P3.5】
输出到【P0.2 / P0.3 / P0.4 / P0.5】对应的指示灯

a) I/O 口设置

这里我们将不同的 I/O 口设置为不同的模式，可以尝试不同情况下依然可以正常输入/输出。
打开 AIapp-ISP 软件：

配置输出端口的 P03、P04 为推挽输出，P05 为开漏输出（可以输出 0 点亮 LED）



输入配置 P30、P31 为高阻输入(因为跟 USB 复用，建议设置为高阻输入，不干扰 USB 的通讯控制)
设置 P32 为准双向口、P33 为高阻输入、P34 和 P35 为开漏模式（设置为 1 等效高阻输入）

然后开启 P33、P34、P35 的内部上拉电阻以稳定电平状态（不开启内部上拉电阻读外部状态，高阻输入/开漏模式，外部都需要有上拉电阻配合）

设置对应 I/O 口时，也可以通过打开/关闭端口左侧的复选框来选择本次生成的代码修改哪些部分 I/O 的状态，如果取消选择，则生成的代码就不会更改未选择 I/O 的模式，保留之前的值。

b) 实验程序 C 语言代码

```
#include <AI8051U.H>            //包含 AI8051U 的头文件
#include "intrins.h"            //使用_nop_()函数所必须要包含的头文件,
                                //否则延时函数中调用的_nop_()函数没有被头文件引用过来，
                                //会导致编译器找不到这个而函数而报错。

void Delay1ms(void)             //@40.000MHz
{
    unsigned long edata i;

    _nop_();
    _nop_();
    _nop_();
    i = 9998UL;
    while (i) i--;
}

void main(void)
{
    EAXFR = 1;          //允许访问扩展的特殊寄存器，XFR
    WTST = 0;           //设置取程序代码等待时间，赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;
    //设置访问片内的 xdata 速度，赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x38; P0M1 = 0x20;
    //设置 P03、P04 为推挽输出，P05 为开漏输出，其余为准双向口模式
    P1M0 = 0x00; P1M1 = 0x00;       //设置 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;       //设置 P2 口为准双向口模式
    P3M0 = 0x30; P3M1 = 0x3b;
    //P30、P31 为高阻输入，P32 为准双向口、P33 为高阻输入、P34 和 P35 为开漏输出
    P3PU |= 0x38;
    //打开 P33、P34、P35 的内部上拉电阻，并且仅修改这三个 I/O 的上拉电阻打开状态
```

```
    P3 |= 0x3c;
    //需要读的 I/O，对外输出锁存器先设置为 1，0011 1100B（0x3c）先和 P3 口逻辑或一下
    //P3M0 = 0x00; P3M1 = 0x00;     //设置 P3 口为准双向口模式
    //P3M0 = 0x00; P3M1 = 0x3c;     //设置 P32、P33、P34、P35 为高阻输入，其余引脚为准双向口模式
    //P3PU = 0x3c;                  //需要同时开启对应端口的上拉电阻，0011 1100B
    P4M0 = 0x00; P4M1 = 0x00;       //设置 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;       //设置 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;       //设置 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;       //设置 P7 口为准双向口模式

    Delay1ms();                     //延时 1ms，等待 I/O 口电平稳定，因为前面刚改变了模式
    //立刻读可能会读到错误的电平，所以需要先等待电平变化稳定后再读

    EA = 1;                         //总中断允许位打开

    P40 = 0;                        //打开 LED 部分的供电

    while(1)
    {
        if(P32 == 0) P02 = 0;       //P32 为低电平，则 LED02 亮
        else P02 = 1;               //P32 为高电平，则 LED02 灭

        if(P33 == 0) P03 = 0;       //P33 为低电平，则 LED03 亮
        else P03 = 1;               //P33 为高电平，则 LED03 灭

        if(P34 == 0) P04 = 0;       //P34 为低电平，则 LED04 亮
        else P04 = 1;               //P34 为高电平，则 LED04 灭

        if(P35 == 0) P05 = 0;       //P35 为低电平，则 LED05 亮
        else P05 = 1;               //P35 为高电平，则 LED05 灭
    }
}
```

c) 实验程序汇编语言代码

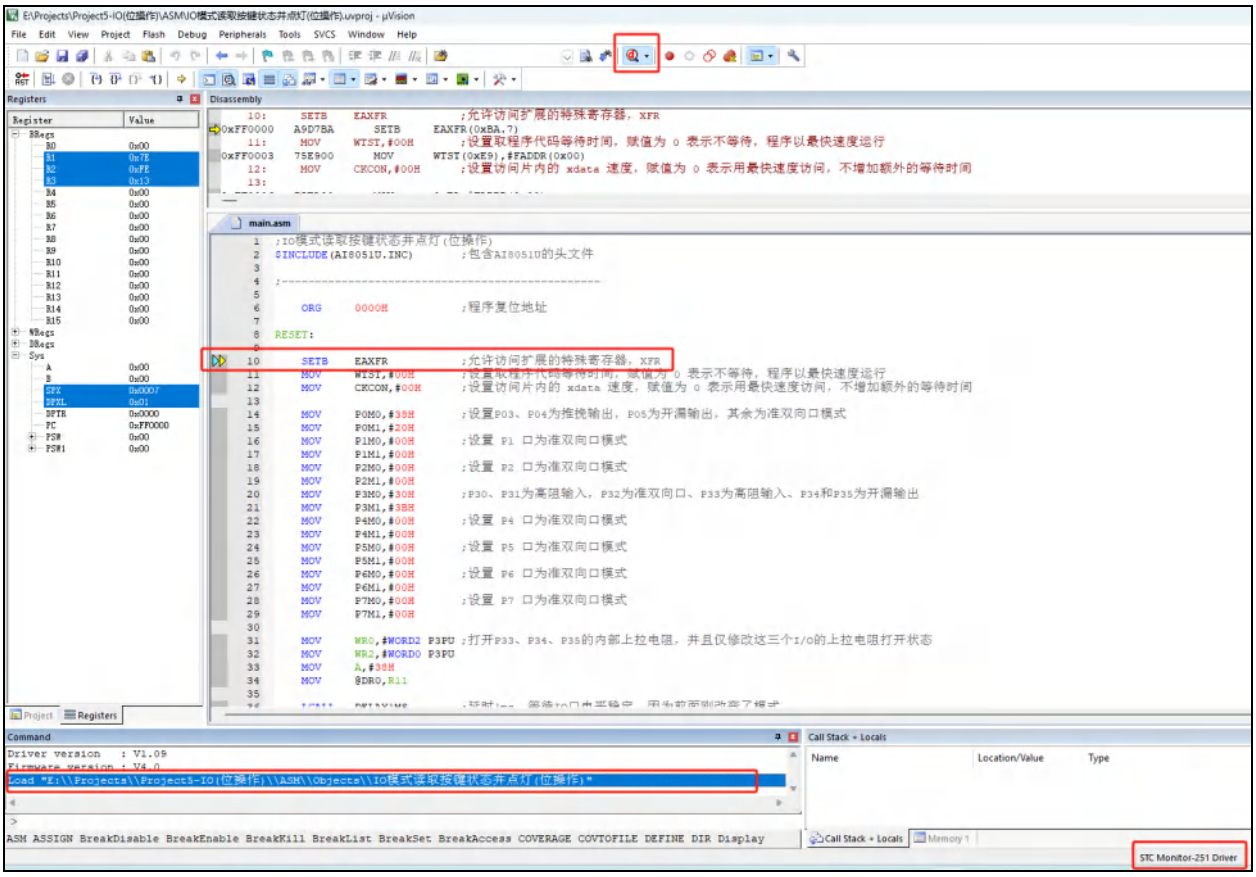
```
;IO 模式读取按键状态并点灯(位操作)
$INCLUDE(AI8051U.INC)           ;包含 AI8051U 的头文件

;-----------------------------------------------

    ORG     0000H               ;程序复位地址

START:
```

```
    SETB    EAXFR           ;允许访问扩展的特殊寄存器，XFR
    MOV     WTST, #00H      ;设置取程序代码等待时间，赋值为 0 表示不等待，程序以最快速度运行
    MOV     CKCON, #00H
            ;设置访问片内的 xdata 速度，赋值为 0 表示用最快速度访问，不增加额外的等待时间

    MOV     P0M0, #38H      ;设置 P03、P04 为推挽输出，P05 为开漏输出，其余为准双向口模式
    MOV     P0M1, #20H
    MOV     P1M0, #00H      ;设置 P1 口为准双向口模式
    MOV     P1M1, #00H
    MOV     P2M0, #00H      ;设置 P2 口为准双向口模式
    MOV     P2M1, #00H
    MOV     P3M0, #30H
            ;P30、P31 为高阻输入，P32 为准双向口、P33 为高阻输入、P34 和 P35 为开漏输出
    MOV     P3M1, #3BH
    MOV     P4M0, #00H      ;设置 P4 口为准双向口模式
    MOV     P4M1, #00H
    MOV     P5M0, #00H      ;设置 P5 口为准双向口模式
    MOV     P5M1, #00H
    MOV     P6M0, #00H      ;设置 P6 口为准双向口模式
    MOV     P6M1, #00H
    MOV     P7M0, #00H      ;设置 P7 口为准双向口模式
    MOV     P7M1, #00H

    MOV     WR0, #WORD2 P3PU
            ;打开 P33、P34、P35 的内部上拉电阻，并且仅修改这三个 I/O 的上拉电阻打开状态
    MOV     WR2, #WORD0 P3PU
    MOV     A, #38H
    MOV     @DR0, R11

    LCALL   DELAY1MS        ;延时 1ms，等待 IO 口电平稳定，因为前面刚改变了模式
                            ;立刻读可能会读到错误的电平，所以需要先等待电平变化稳定后再读

    CLR     P40             ;打开 LED 部分的供电

;---------------------------------------------

MAINLOOP:

    JNB     P3.2, P32ISLOW  ;判断 P32 的电平

P32ISHIGH:
    SETB    P0.2            ;P32 为高电平，则 LED02 灭
    SJMP    CHECKP33        ;跳转到判断 P33

P32ISLOW:
    CLR     P0.2            ;P32 为低电平，则 LED02 亮
```

```
CHECKP33:
    JNB     P3.3, P33ISLOW   ;判断 P33 的电平

P33ISHIGH:
    SETB    P0.3             ;P33 为高电平，则 LED03 灭
    SJMP    CHECKP34         ;跳转到判断 P34

P33ISLOW:
    CLR     P0.3             ;P33 为低电平，则 LED03 亮

CHECKP34:
    JNB     P3.4, P34ISLOW   ;判断 P34 的电平

P34ISHIGH:
    SETB    P0.4             ;P34 为高电平，则 LED04 灭
    SJMP    CHECKP35         ;跳转到判断 P35

P34ISLOW:
    CLR     P0.4             ;P34 为低电平，则 LED04 亮

CHECKP35:
    JNB     P3.5, P35ISLOW   ;判断 P35 的电平

P35ISHIGH:
    SETB    P0.5             ;P35 为高电平，则 LED05 灭
    SJMP    MAINLOOP         ;跳回到主循环开始的地方继续运行程序

P35ISLOW:
    CLR     P0.5             ;P35 为低电平，则 LED05 亮

    SJMP    MAINLOOP         ;跳回到主循环开始的地方继续运行程序

;----------------------------------------------

DELAY1MS:                    ;@40.000MHz
    PUSH    DR0
    MOV     WR0, #0
    MOV     WR2, #9998

NEXT:
    DEC     DR0, #1
    JNE     NEXT
    POP     DR0
    RET
```

```
;-------------------------------------------------
    END
```

使用 Aiapp-ISP 系统软件及 Ai8051U 实验箱制作仿真芯片，上述代码编译无误，点击“Debug”按钮，进入仿真界面，如下图：



实验操作步骤及实验现象与 C 语言代码编写的该项目完全相同。

14.14.7　实验 1A--使用【AiCube】项目创建助手实现“位操作，用 USB 仿真来点灯”

STCAI 在最新版的 AIapp-ISP 软件中，推出【AiCube】项目创建助手工具，目前支持 Ai8051U-32Bit 各功能模块的智能化配置，并生成程序框架。这项功能在程序员建立项目时，简化了项目的初始化难度，提高效率，避免因不太熟悉或不完全了解该单片机的性能而忽略个别配置，引起编译错误等问题。尤其对刚涉足单片机的初学者，此工具不仅引导其了解单片机编程流程，还极大的降低了入门门槛，缩短了摸索的时间，提高入门的兴趣和信心。现在我们使用这个工具来生成实现此实验功能的配置和代码。

a) 安装 AIapp-ISP 下载/编程/烧录/软件

安装 STCAI 公司研发的 AIapp-ISP 下载 / 编程 / 烧录 / 软件，含强大的辅助开发工具包。登录 STCAI 官网，下载最新版的 AIapp-ISP 安装包，下载链接如下：https://www.stcai.com/gjrj

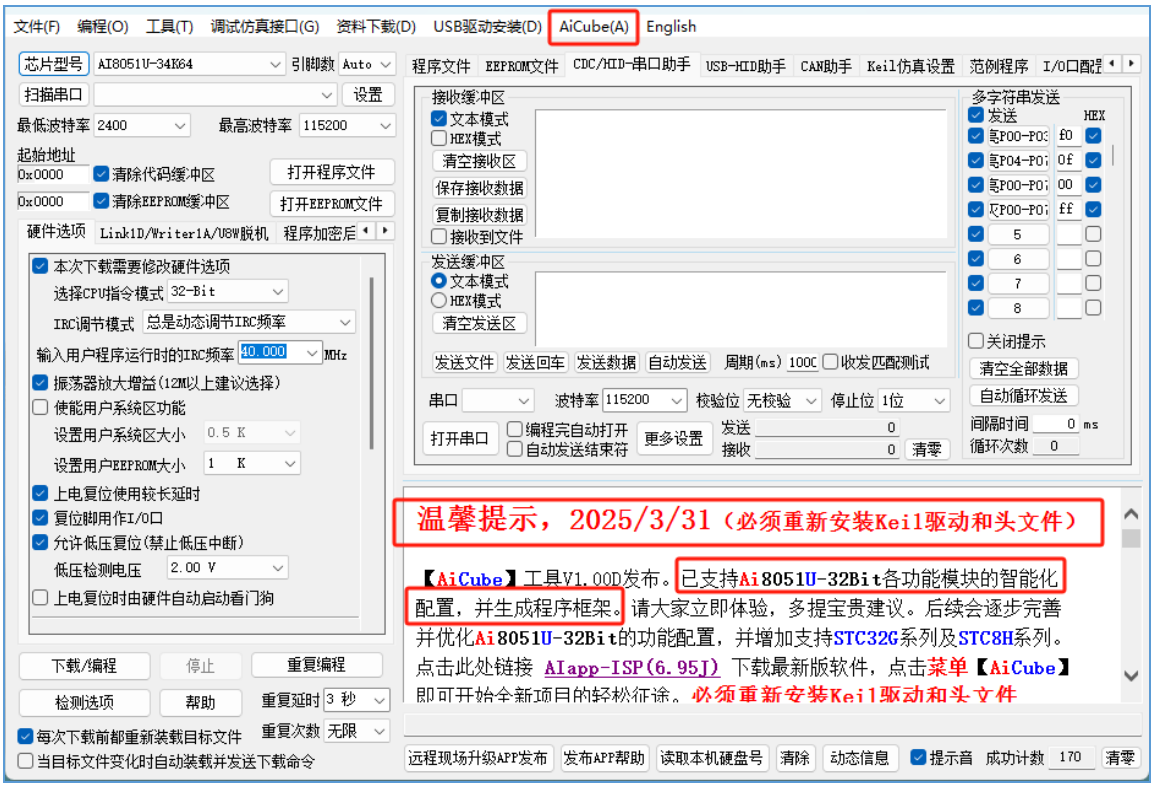
下载下来的是压缩包文件如下图：

双击后解压缩如下图：

如果是解压缩到非桌面的其他目录，如 D:\STCAI-ISP，建议创建快捷方式到桌面方便打开。这实际是一个免安装的绿色软件。

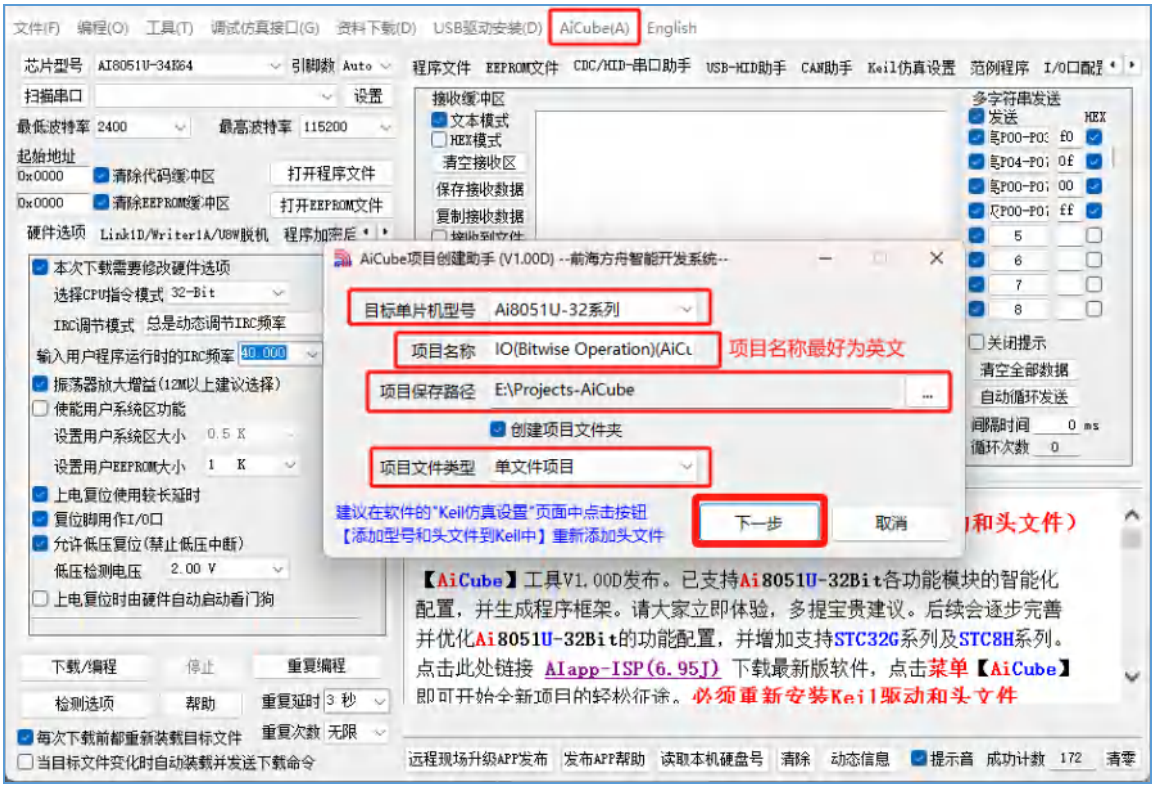
b) 打开【AiCube】助手

双击图标，即可打开 AIapp-ISP 软件界面，即可直接使用，如下图：

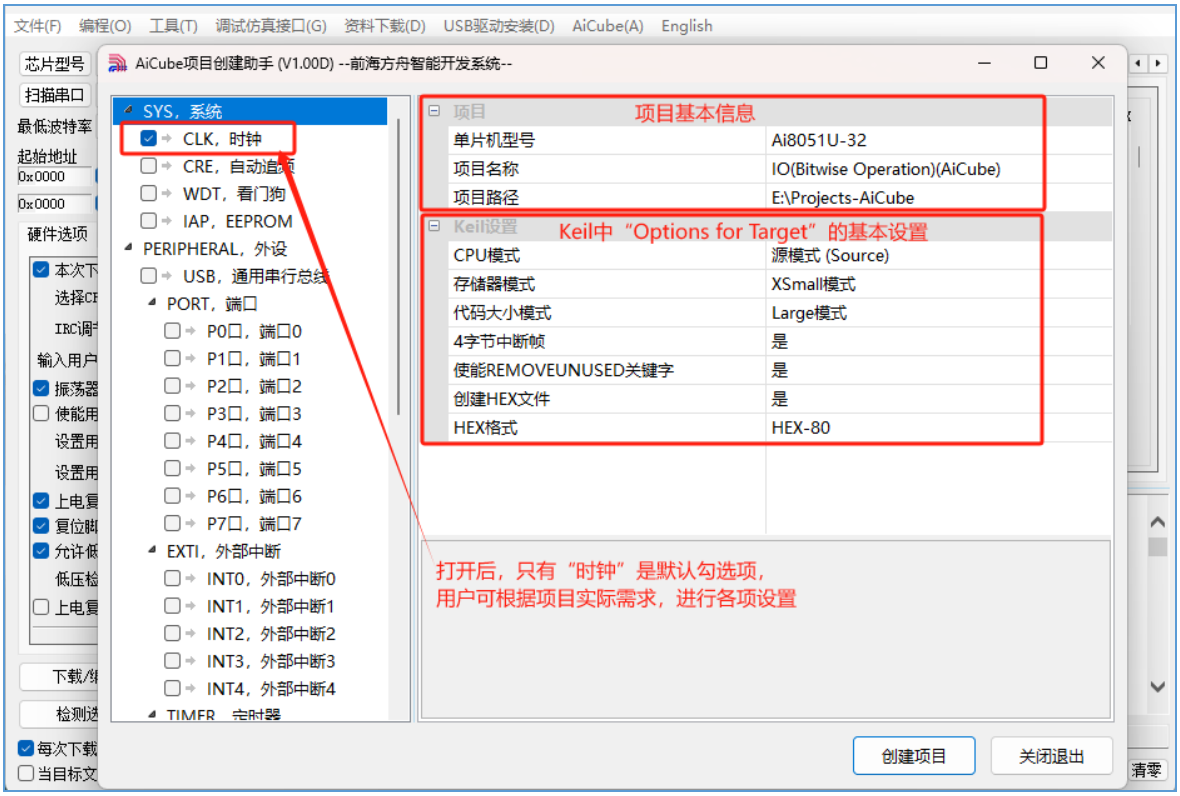


c) 使用【AiCube】助手创建项目

点击【AiCube（A）】，弹出如下窗口：



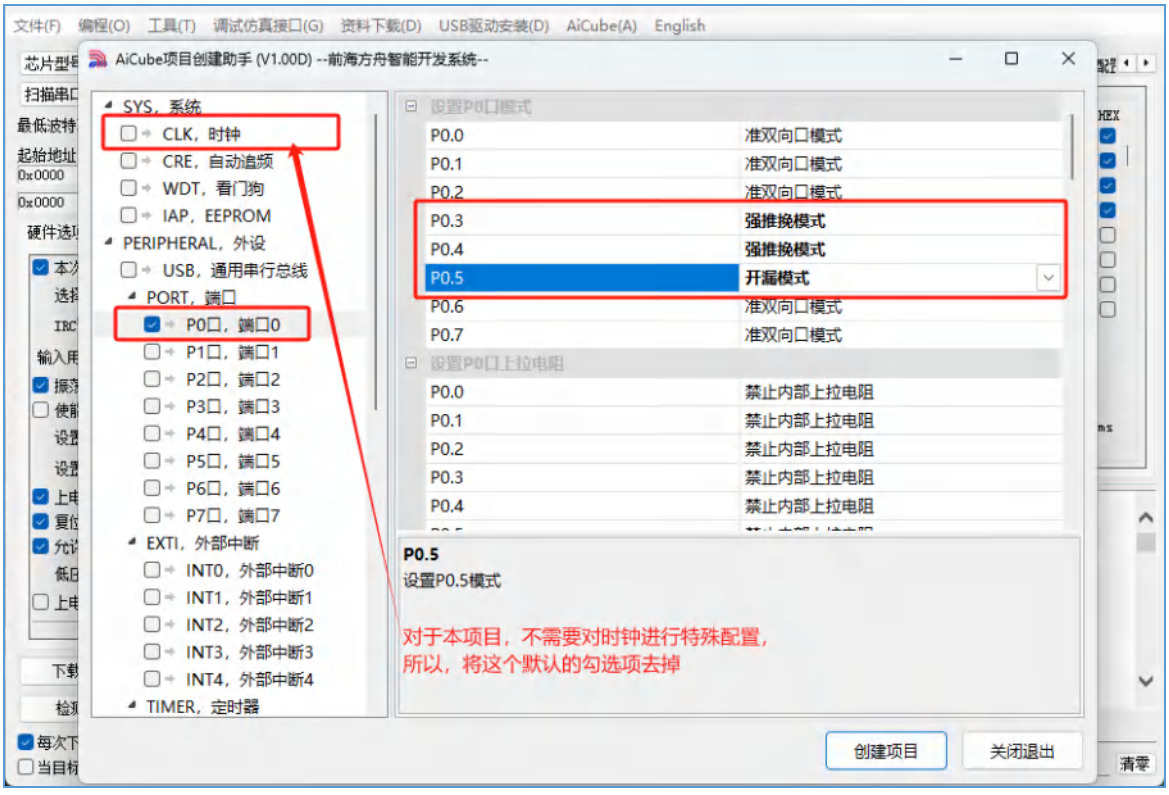
选择并填写项目基本信息，建议项目名称用英文！点击“下一步”，出现如下配置界面：



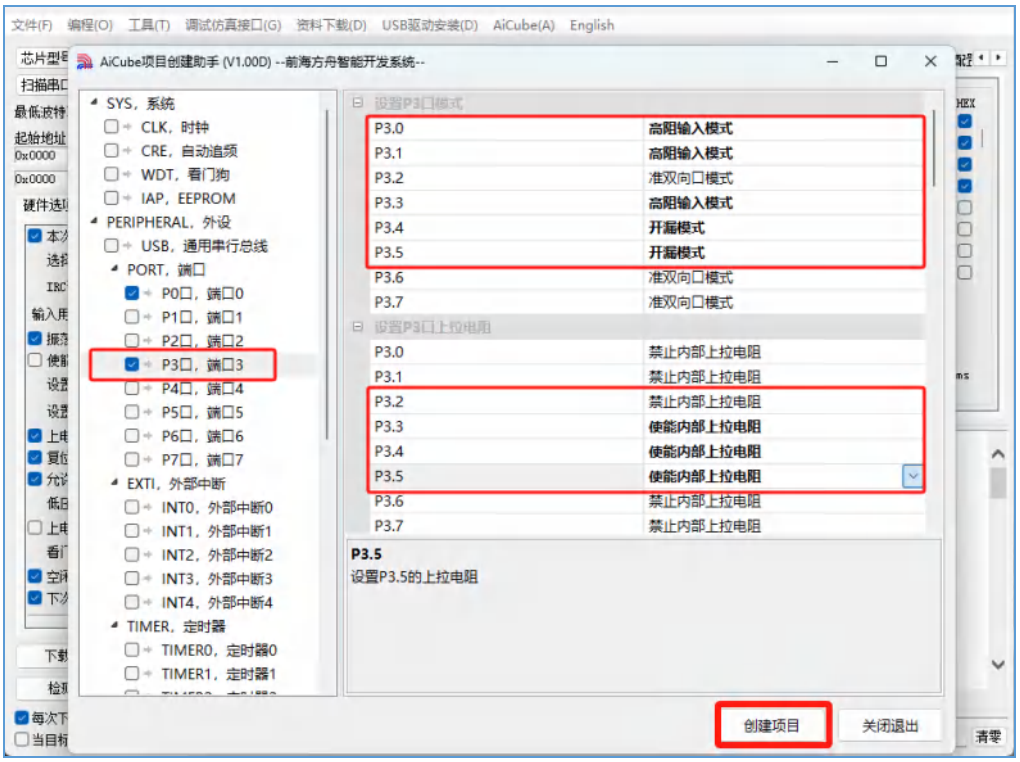
对于本项目，需要单片机读【P3.2 / P3.3 / P3.4 / P3.5】这 4 个 I/O 的状态，
输出到【P0.2 / P0.3 / P0.4 / P0.5】对应的指示灯

针对 I/O 有 4 种不同的模式，我们将【P3.2 / P3.3 / P3.4 / P3.5】这 4 个不同的 I/O 口设置为不同的模式，以验证不同情况下依然可以正常输入/输出。

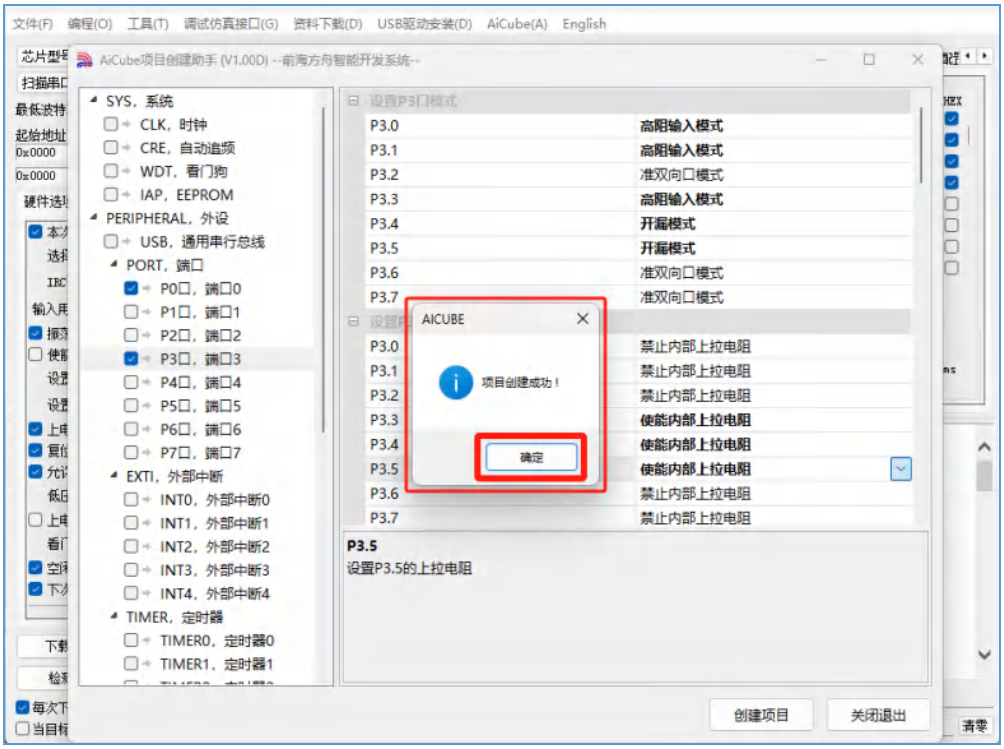
【AiCube】助手中，端口默认是准双向口模式，我们配置**输出端口**的 P02 保持准双向口模式，P03、P04 为强推挽输出，P05 为开漏输出（可以输出 0 点亮 LED），如下图：



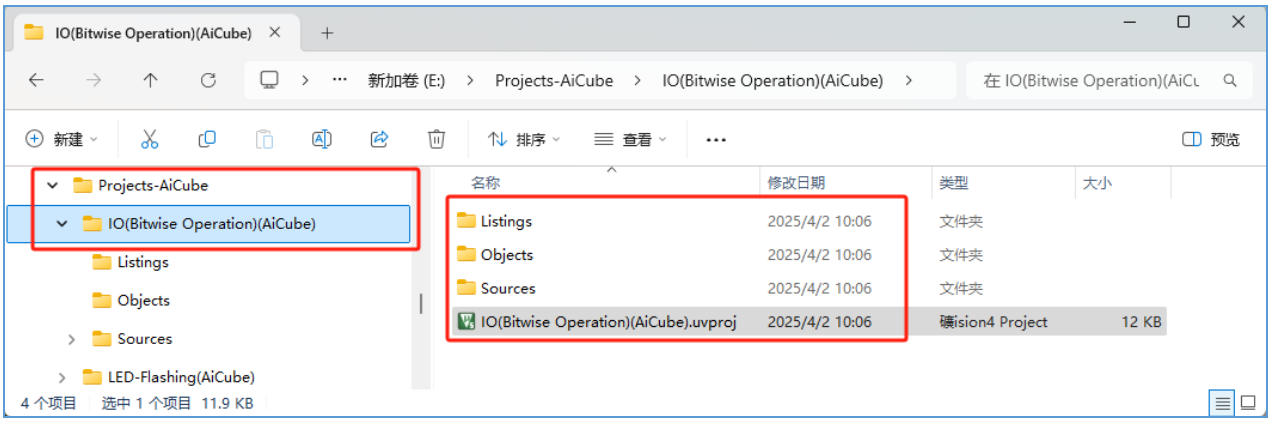
输入配置 P30、P31 为高阻输入(因为跟 USB 复用，建议设置为高阻输入，不干扰 USB 的通讯控制)
设置 P32 为准双向口、P33 为高阻输入、P34 和 P35 为开漏模式（设置为 1 等效高阻输入）
然后开启 P33、P34、P35 的内部上拉电阻以稳定电平状态（不开启内部上拉电阻读外部状态，高阻输入/开漏模式，外部都需要有上拉电阻配合）
【AiCube】助手中，端口默认是准双向口模式，内部上拉电阻默认是禁用设置。现在我们按上述要求配置 P30、P31 为高阻输入；P32 为保持准双向口；P33 为高阻输入；P34、P35 为开漏模式；打开 P33、P34、P35 的内部上拉电阻。如下图：



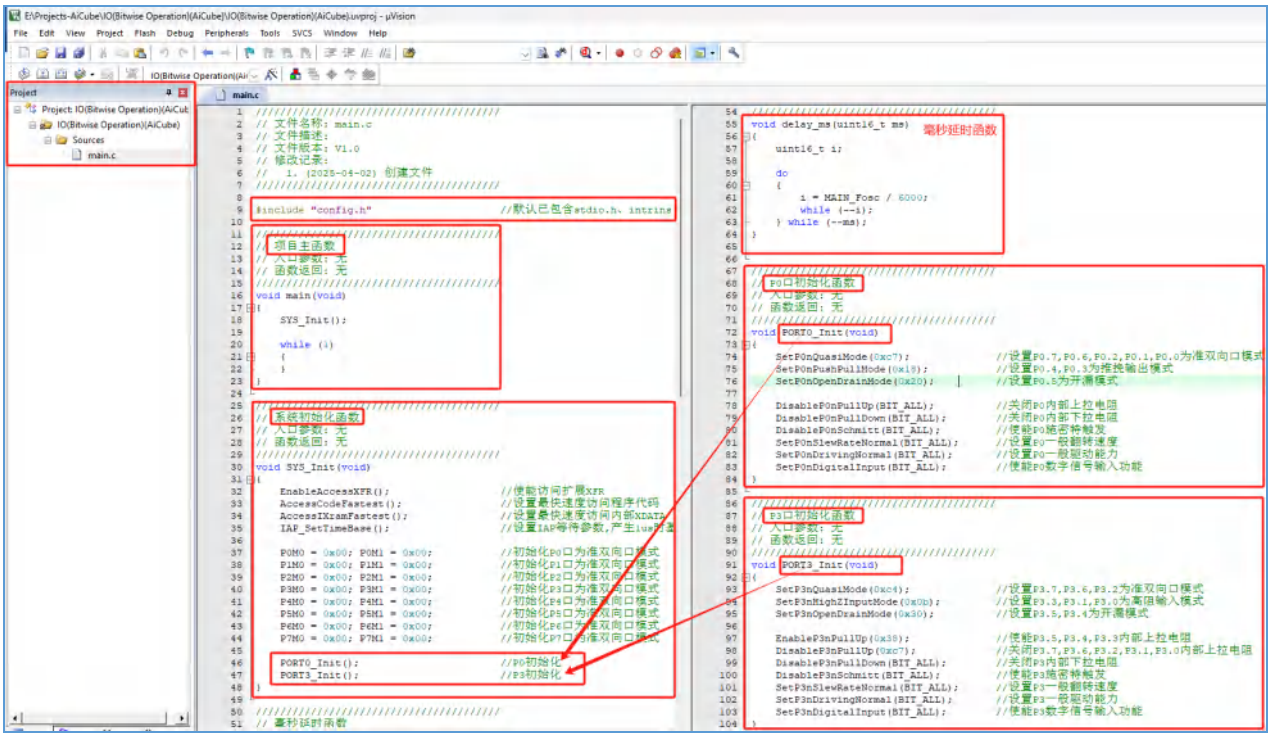
点击“创建项目”，【AiCube】助手提示：“项目创建成功！”，如下图：



点击“确定”，即成功创建了项目及项目文件目录“IO(Bitwise Operation)(AiCube)”，找到并打开刚刚创建的项目目录。发现刚刚的操作过程，【AiCube】助手已经帮我们建立了项目目录及 IO(Bitwise Operation)(AiCube).uvproj 文件，如下图：



双击打开项目文件：IO(Bitwise Operation)(AiCube).uvproj，如下图（为方便查看，使用菜单“Window”→“Split”将原窗口拆分成两个窗口，全代码观察）：

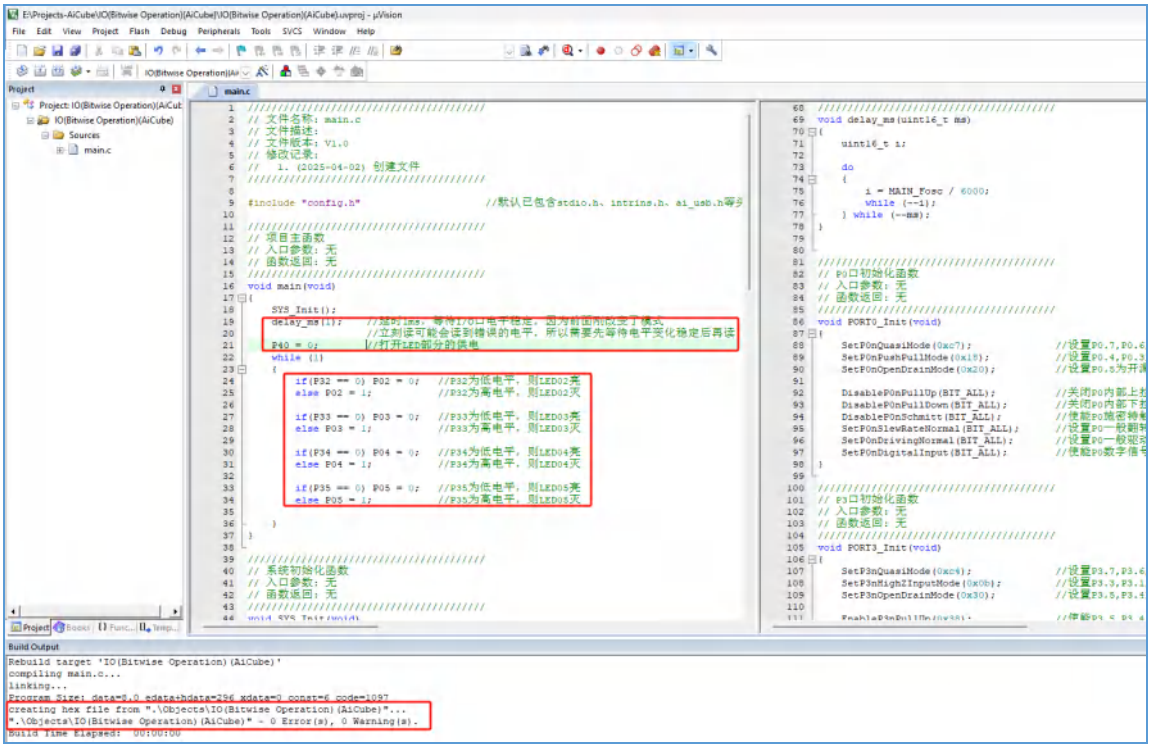


根据在【AiCube】助手中的配置，从上图中我们看到【AiCube】助手创建的本实验项目中有了 main.c。main.c 中按模块化设计，已经包含了：必要的头文件、主函数、系统初始化函数、延时函数、P0 口初始化函数、P3 口初始化函数。

d) 增加实验项目代码

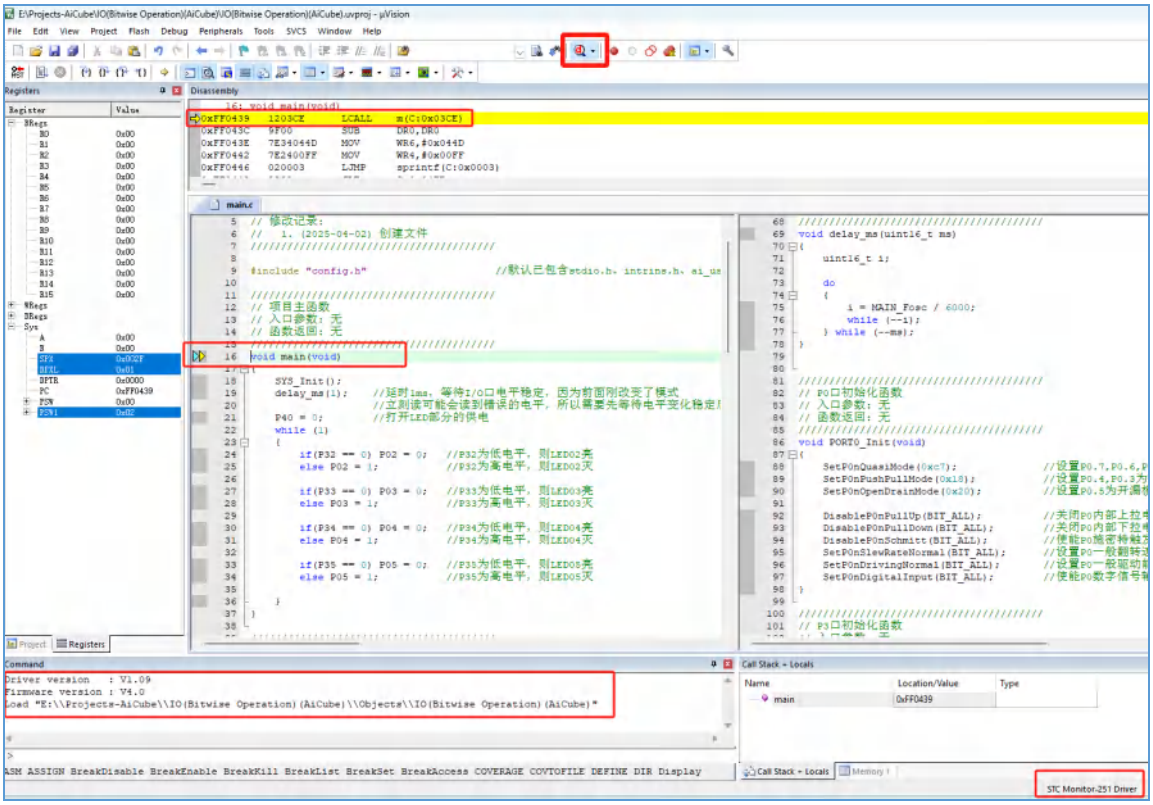
为实现“I/O 口位操作点灯，用 USB 仿真点灯”的实验，我们现在需要在主函数中：

- 加入调用延时函数“delay_ms(1)”延时 1ms（等待 I/O 口电平稳定）；
- 加入“P40 = 0”打开 LED 供电的代码；
- 在 while 循环中加入 I/O 口位操作点灯代码。
- 保存，编译，如下图（为方便查看，使用菜单“Window”→“Split”将原窗口拆分成两个窗口，对比观察）：



e) 使用 AIapp-ISP 软件设置仿真芯片并连接 Ai8051U 实验箱验证实验效果

- 连接好 Ai8051U 实验箱，运行 AIapp-ISP 软件系统，在 Keil 仿真设置选项卡中，按步骤操作“将所选目标单片机设置为仿真芯片”。
- 回到 Keil 界面，在 Keil 中按步骤在“Options for Target”的 Debug 选项卡和 Utilities 选项卡中完成使用“STC Monitor-251 Driver”的 USB 仿真设置。
- 点击 File Toolbar 上的“Start/Stop Debug Session”按钮，进入仿真界面，如下图：



一切准备就绪，点击 Debug Toolbar 上的“Run”按钮，

- 按下 Ai8051U 实验箱上的 P3.2 按键，对应的 P02 灯就亮了；松开 P3.2 按键，P02 灯就灭了。
- 按下 Ai8051U 实验箱上的 P3.3 按键，对应的 P03 灯就亮了；松开 P3.3 按键，P02 灯就灭了。
- 按下 Ai8051U 实验箱上的 P3.4 按键，对应的 P04 灯就亮了；松开 P3.4 按键，P02 灯就灭了。
- 按下 Ai8051U 实验箱上的 P3.5 按键，对应的 P05 灯就亮了；松开 P3.5 按键，P02 灯就灭了。

有了【AiCube】助手，单片机编程就是这么简单！快用起来吧！

14.14.8　实验 2：整组 8 位端口操作，用 USB 直接仿真来调试

整组 读这几个 I/O 的状态【P3.2/P3.3/P3.4/P3.5】，一次性读全部 P3 口
整体输出到 【P0.2/P0.3/P0.4/P0.5】在的 P0 口对应的指示灯

a) 实验程序代码

```
#include <AI8051U.H>          //包含 AI8051U 的头文件
#include "intrins.h"          //使用_nop_()函数所必须要包含的头文件,
                              //否则延时函数中调用的_nop_()函数没有被头文件引用过来，
                              //会导致编译器找不到这个而函数而报错。

void Delay1ms(void)           //@40.000MHz
{
    unsigned long edata i;

    _nop_();
    _nop_();
    _nop_();
    i = 9998UL;
    while (i) i--;
}

void main(void)
{
    EAXFR = 1;      //允许访问扩展的特殊寄存器，XFR
    WTST = 0;       //设置取程序代码等待时间，赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;
        //设置访问片内的 xdata 速度，赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x38; P0M1 = 0x20;
        //设置 P03、P04 为推挽输出，P05 为开漏输出，其余为准双向口模式

    P1M0 = 0x00; P1M1 = 0x00;     //设置 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;     //设置 P2 口为准双向口模式

    P3M0 = 0x30; P3M1 = 0x3b;
        //P30、P31 为高阻输入，P32 为准双向口、P33 为高阻输入、P34 和 P35 为开漏输出

    P3PU |= 0x38;
        //打开 P33、P34、P35 的内部上拉电阻，并且仅修改这三个 I/O 的上拉电阻打开状态

    P3 |= 0x3c;
    //需要读的 I/O，对外输出锁存器先设置为 1，0011 1100B（0x3c）先和 P3 口逻辑或一下

//  P3M0 = 0x00; P3M1 = 0x00;     //设置 P3 口为准双向口模式
```

```
//    P3M0 = 0x00; P3M1 = 0x3c;      //设置 P32、P33、P34、P35 为高阻输入，其余引脚为准双向口模式
//    P3PU = 0x3c;                   //需要同时开启对应端口的上拉电阻，0011 1100B

    P4M0 = 0x00; P4M1 = 0x00;      //设置 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;      //设置 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;      //设置 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;      //设置 P7 口为准双向口模式

    Delay1ms();                    //延时 1ms，等待 I/O 口电平稳定，因为前面刚改变了模式
        //立刻读可能会读到错误的电平，所以需要先等待电平变化稳定后再读

    EA = 1;                        //总中断允许位打开
    P40 = 0;                       //打开 LED 部分的供电

    while(1)
    {
        P0 = P3|0xc3;              //同步端口状态，并将其他不需要的位置 1（不显示），11000011B
    }
}
```

14.14.9　实验 2A--使用【AiCube】项目创建助手实现“整组 8 位端口操作，用 USB 仿真来点灯”

STCAI 在最新版的 AIapp-ISP 软件中，推出【AiCube】项目创建助手工具，目前支持 Ai8051U-32Bit 各功能模块的智能化配置，并生成程序框架。这项功能在程序员建立项目时，简化了项目的初始化难度，提高效率，避免因不太熟悉或不完全了解该单片机的性能而忽略个别配置，引起编译错误等问题。尤其对刚涉足单片机的初学者，此工具不仅引导其了解单片机编程流程，还极大的降低了入门门槛，缩短了摸索的时间，提高入门的兴趣和信心。现在我们使用这个工具来生成实现此实验功能的配置和代码。

a) 安装 AIapp-ISP 下载/编程/烧录/软件

安装 STCAI 公司研发的 AIapp-ISP 下载 / 编程 / 烧录 / 软件，含强大的辅助开发工具包。登录 STCAI 官网，下载最新版的 AIapp-ISP 安装包，下载链接如下：https://www.stcai.com/gjrj

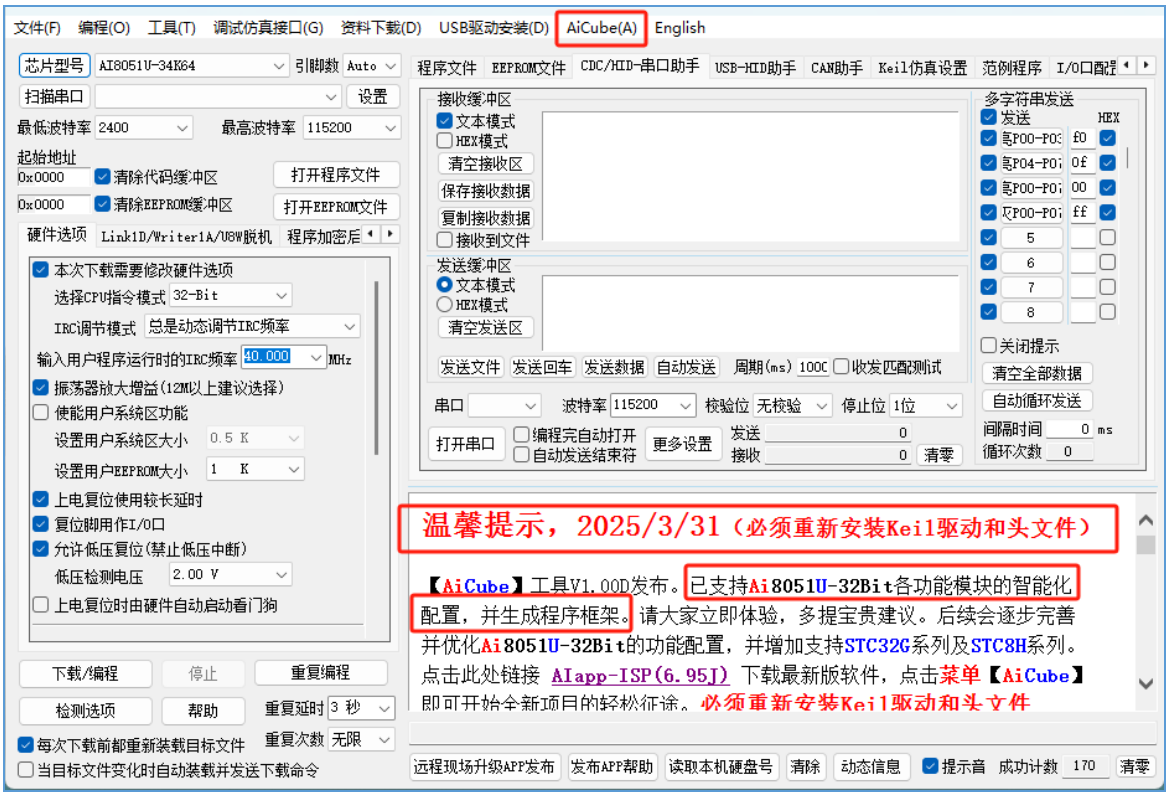
下载下来的是压缩包文件如下图：

双击后解压缩如下图：

如果是解压缩到非桌面的其他目录，如 D:\STCAI-ISP，建议创建快捷方式到桌面方便打开。这实际是一个免安装的绿色软件。

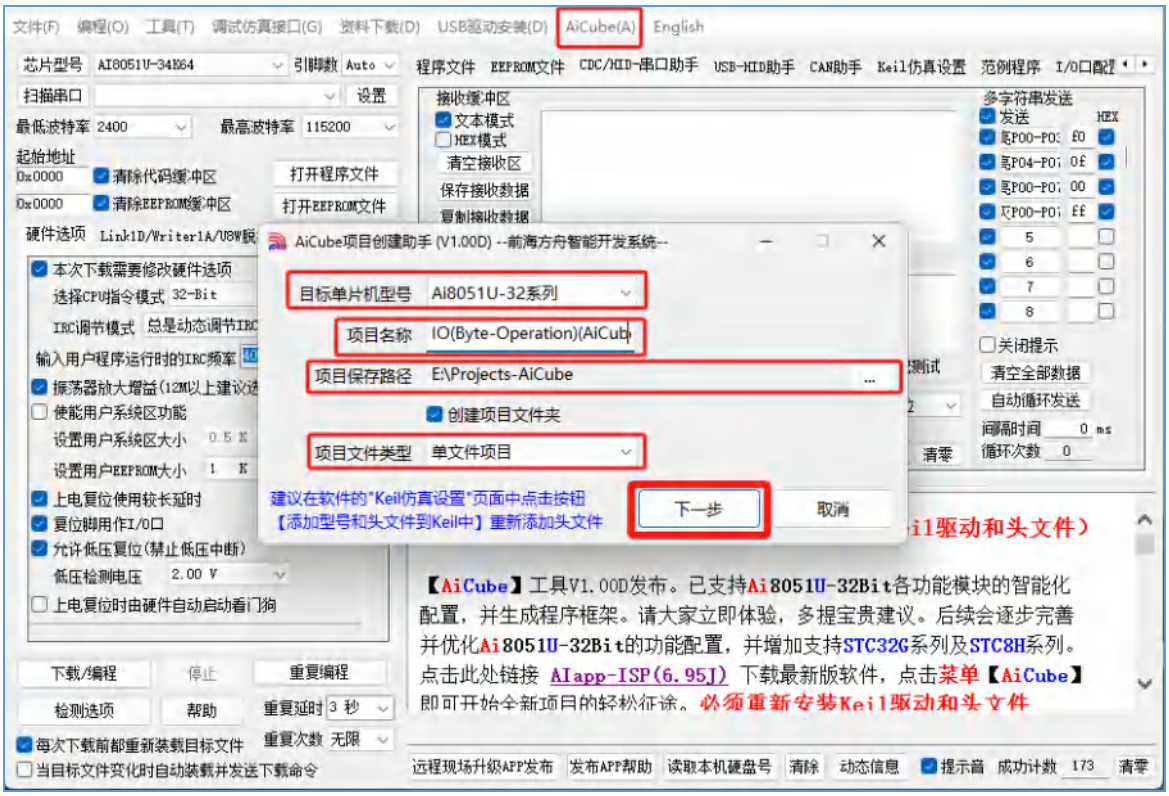
b) 打开【AiCube】助手

双击图标，即可打开 AIapp-ISP 软件界面，即可直接使用，如下图：

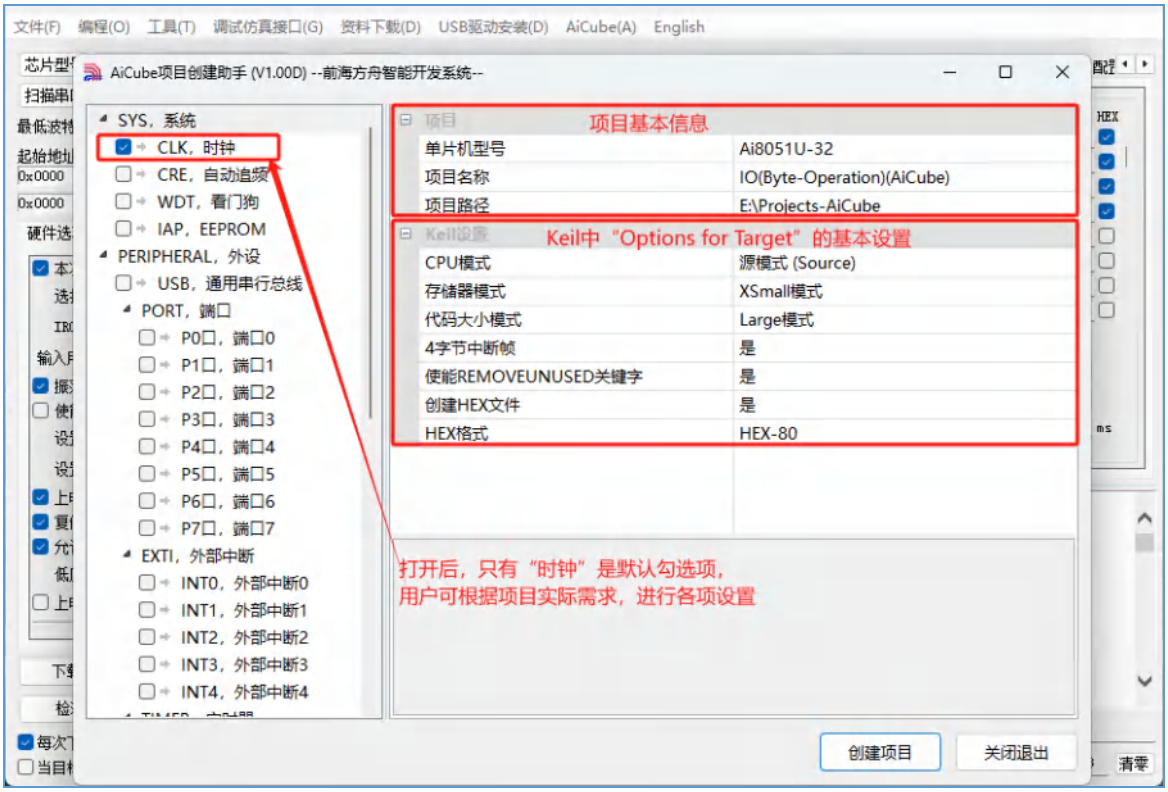


c) 使用【AiCube】助手创建项目

点击【AiCube（A）】，弹出如下窗口：



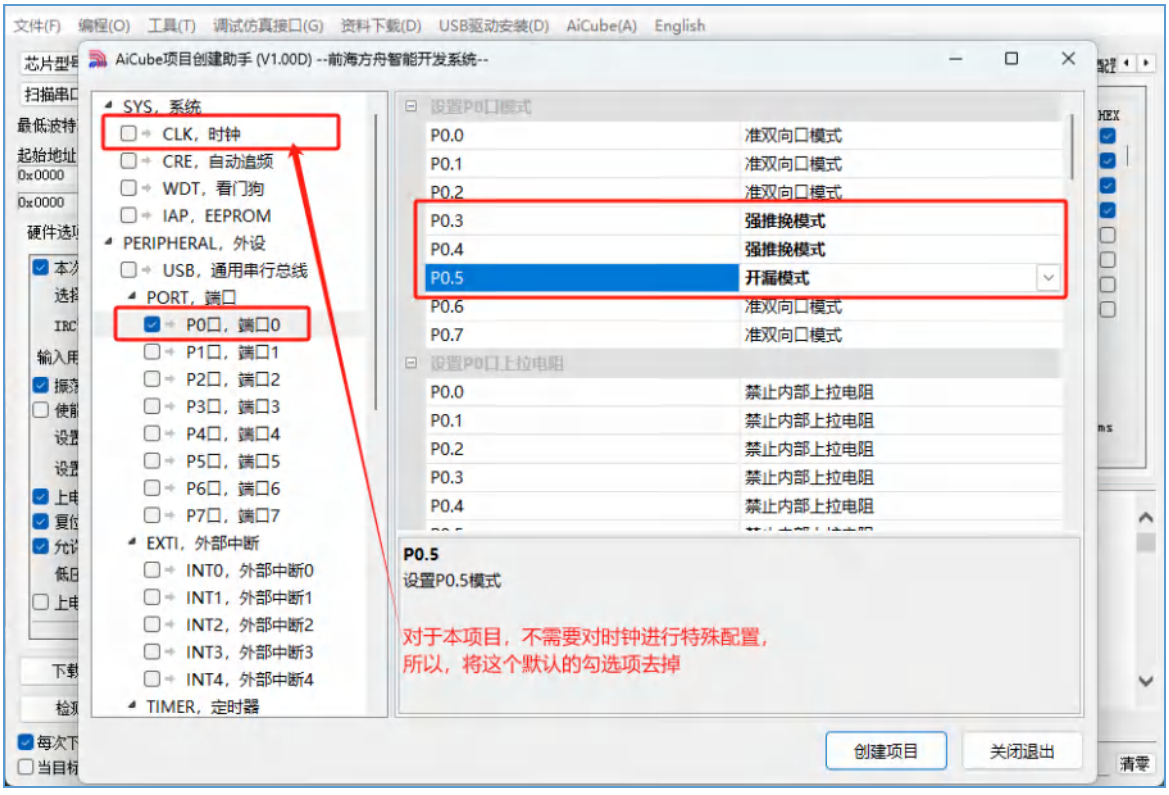
选择并填写项目基本信息，建议项目名称用英文！点击“下一步”，出现如下配置界面：



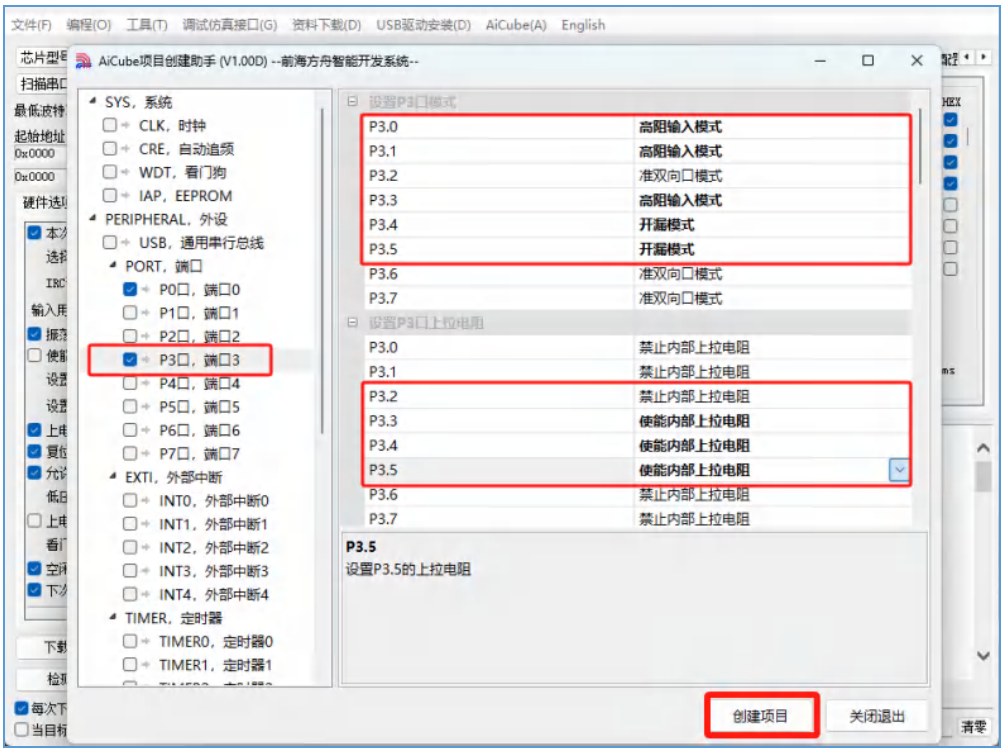
对于本项目，需要单片机读【P3.2 / P3.3 / P3.4 / P3.5】这 4 个 I/O 的状态，
输出到【P0.2 / P0.3 / P0.4 / P0.5】对应的指示灯

针对 I/O 有 4 种不同的模式，我们将【P3.2 / P3.3 / P3.4 / P3.5】这 4 个不同的 I/O 口设置为不同的模式，以验证不同情况下依然可以正常输入/输出。

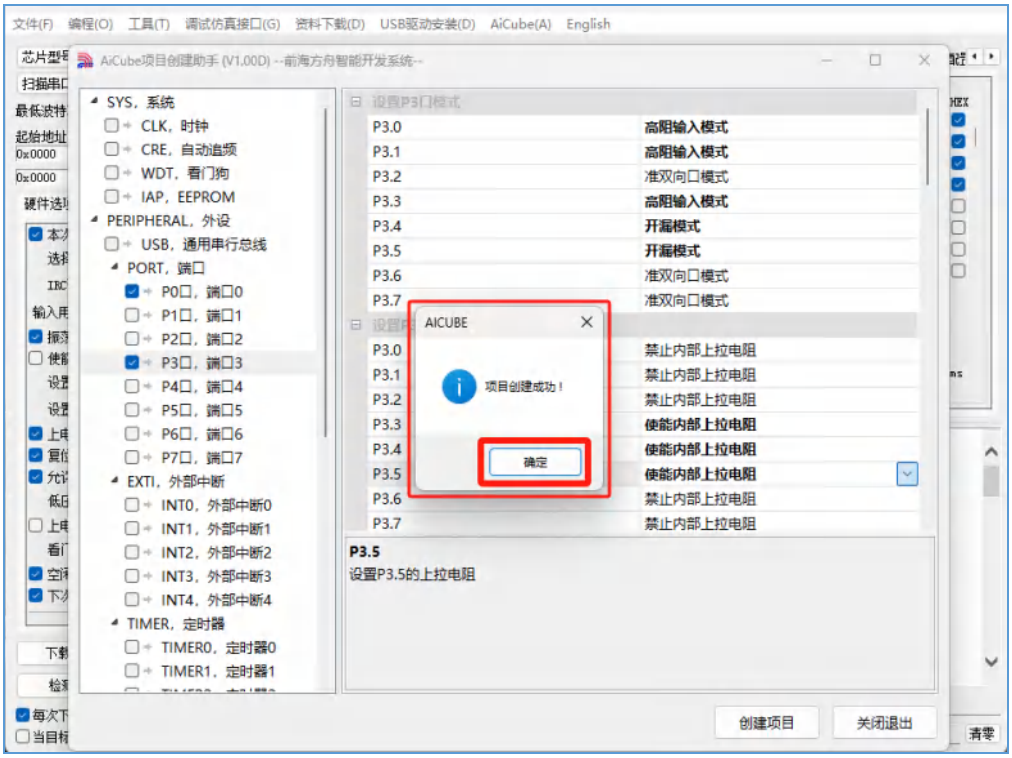
【AiCube】助手中，端口默认是准双向口模式，我们配置**输出端口**的 P02 保持准双向口模式，P03、P04 为强推挽输出，P05 为开漏输出（可以输出 0 点亮 LED）。如下图：



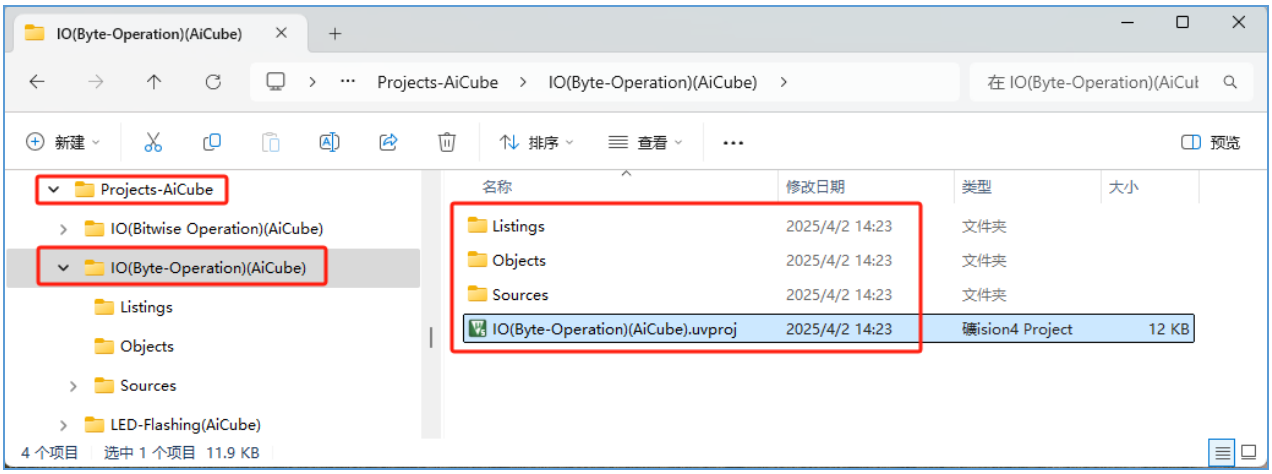
输入配置 P30、P31 为高阻输入(因为跟 USB 复用，建议设置为高阻输入，不干扰 USB 的通讯控制)
设置 P32 为准双向口、P33 为高阻输入、P34 和 P35 为开漏模式（设置为 1 等效高阻输入）
然后开启 P33、P34、P35 的内部上拉电阻以稳定电平状态（不开启内部上拉电阻读外部状态，高阻输入/开漏模式，外部都需要有上拉电阻配合）
【AiCube】助手中，端口默认是准双向口模式，内部上拉电阻默认是禁用设置。现在我们按上述要求配置 P30、P31 为高阻输入；P32 为保持准双向口；P33 为高阻输入；P34、P35 为开漏模式；打开 P33、P34、P35 的内部上拉电阻。如下图：



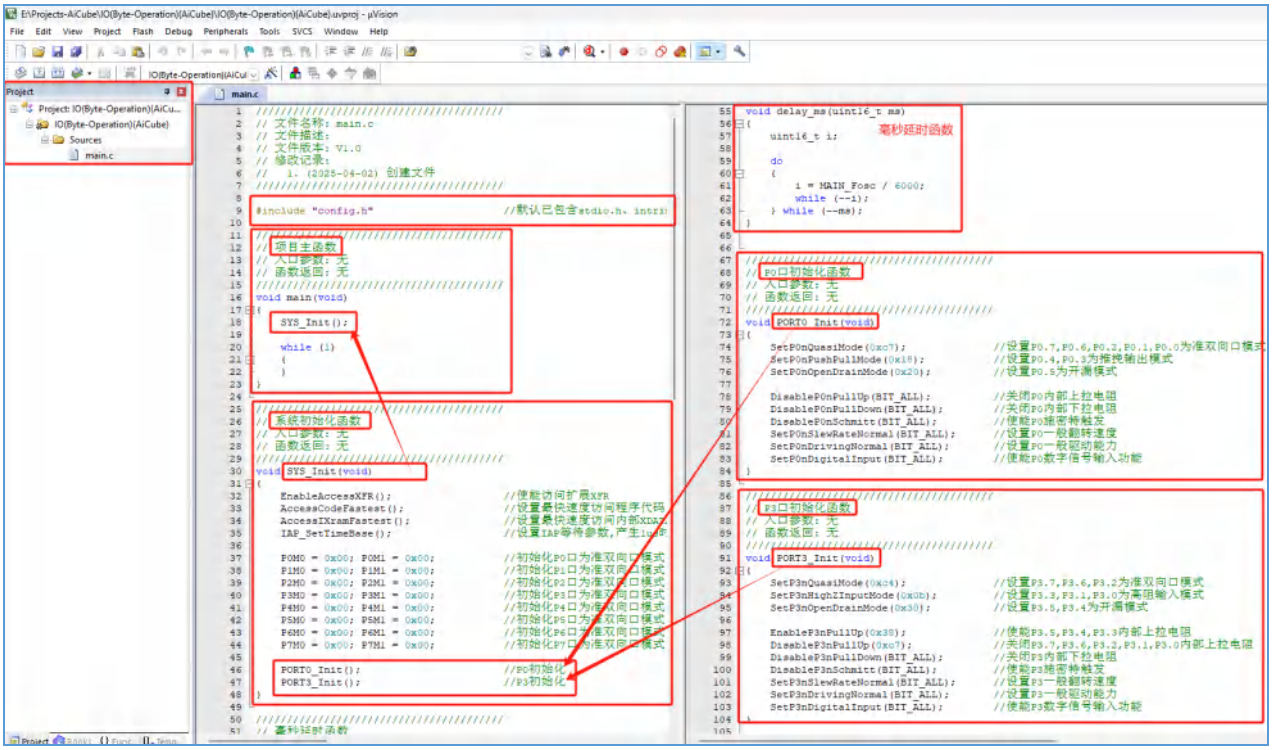
点击“创建项目”，【AiCube】助手提示：“项目创建成功！”，如下图：



点击“确定”，即成功创建了项目及项目文件目录“IO(Byte-Operation)(AiCube)”，找到并打开刚刚创建的项目目录。发现刚刚的操作过程，【AiCube】助手已经帮我们建立了项目目录及 IO(Byte-Operation)(AiCube).uvproj 文件，如下图：



双击打开项目文件：IO(Byte-Operation)(AiCube).uvproj，如下图（为方便查看，使用菜单“Window”→“Split”将原窗口拆分成两个窗口，全代码观察）：



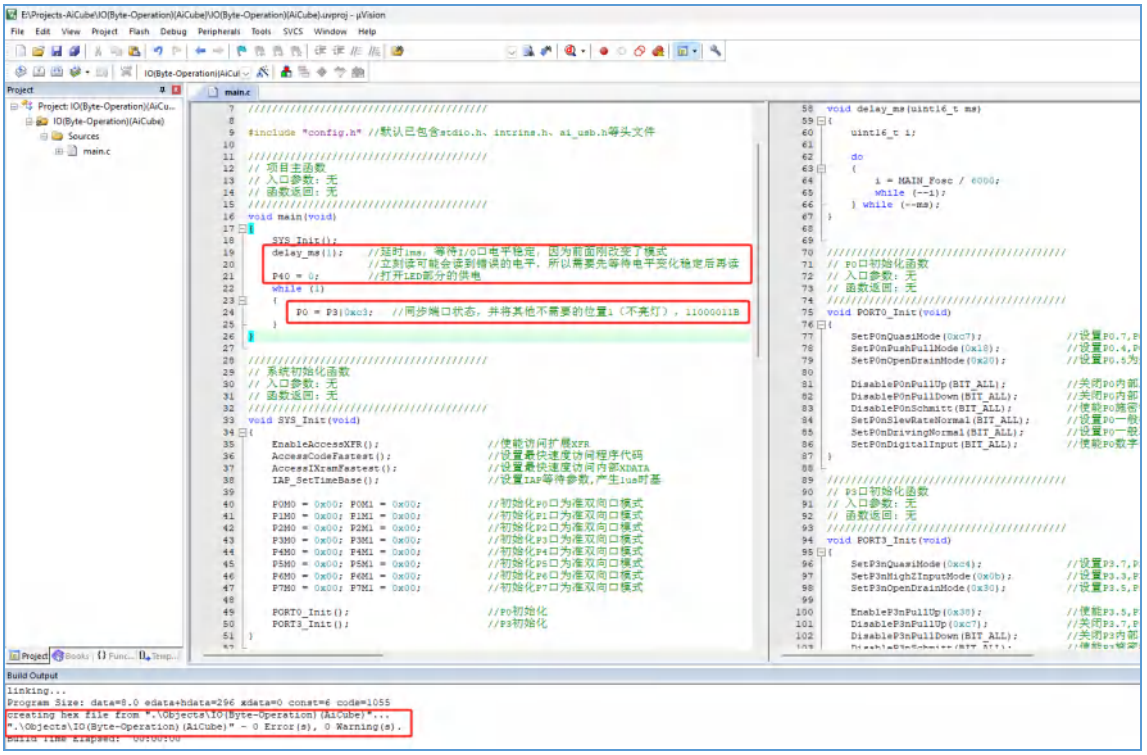
根据在【AiCube】助手中的配置，从上图中我们看到【AiCube】助手创建的本实验项目中有了 main.c。main.c 中按模块化设计，已经包含了：必要的头文件、主函数、系统初始化函数、延时函数、P0 口初始化函数、P3 口初始化函数。

d) 增加实验项目代码

为实现“I/O 整组 8 位操作点灯，用 USB 仿真调试”的实验，我们现在需要在主函数中：

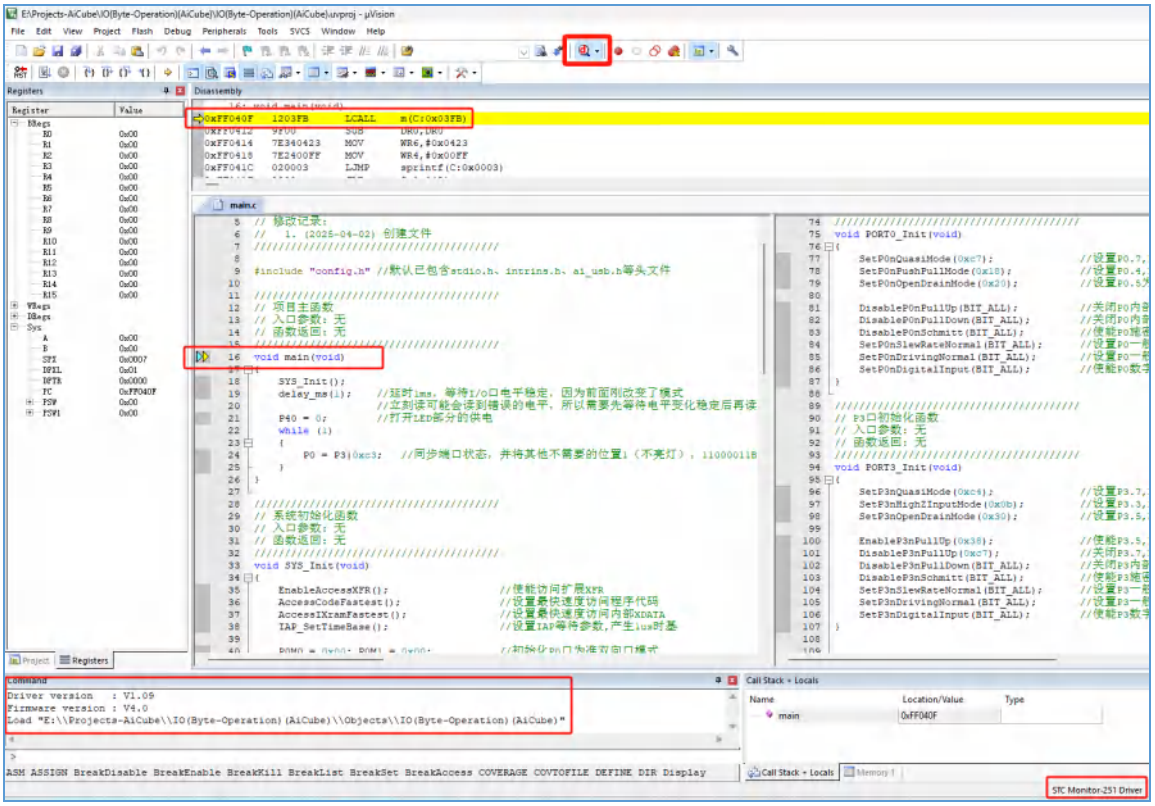
- 调用延时函数“delay_ms(1);”，延时 1ms（等待 I/O 口电平稳定）；
- 加入“P40 = 0;”，打开 LED 供电的代码；
- 在 while 循环中加入 I/O 口 8 位整组操作点灯代码“P0 = P3|0xc3;”。
- 保存，编译，如下图（为方便查看，使用菜单“Window”→“Split”将原窗口拆分成两个窗口，对

比观察）：



e) 使用 AIapp-ISP 软件设置仿真芯片并连接 Ai8051U 实验箱验证实验效果

- 连接好 Ai8051U 实验箱，运行 AIapp-ISP 软件系统，在 Keil 仿真设置选项卡中，按步骤操作“将所选目标单片机设置为仿真芯片”。
- 回到 Keil 界面，在 Keil 中按步骤在“Options for Target”的 Debug 选项卡和 Utilities 选项卡中完成使用“STC Monitor-251 Driver”的 USB 仿真设置。
- 点击 File Toolbar 上的“Start/Stop Debug Session”按钮，进入仿真界面，如下图：



一切准备就绪，点击 Debug Toolbar 上的“Run”按钮，

- 按下 Ai8051U 实验箱上的 P3.2 按键，对应的 P02 灯就亮了；松开 P3.2 按键，P02 灯就灭了。
- 按下 Ai8051U 实验箱上的 P3.3 按键，对应的 P03 灯就亮了；松开 P3.3 按键，P02 灯就灭了。
- 按下 Ai8051U 实验箱上的 P3.4 按键，对应的 P04 灯就亮了；松开 P3.4 按键，P02 灯就灭了。
- 按下 Ai8051U 实验箱上的 P3.5 按键，对应的 P05 灯就亮了；松开 P3.5 按键，P02 灯就灭了。

有了【AiCube】助手，单片机编程就是这么简单！快用起来吧！

14.14.10 实验 3：用 USB-CDC 读取按键状态在串口助手显示并点亮 LED 灯

电脑 AIapp-ISP 软件中的，USB-CDC 串口助手通知单片机读【P3.2/P3.3/P3.4/P3.5】的状态，送回 USB-CDC 串口助手显示，同时将相应的状态送到 LED【P0.2/P0.3/P0.4/P0.5】显示（点亮/熄灭）

a) 实验介绍：

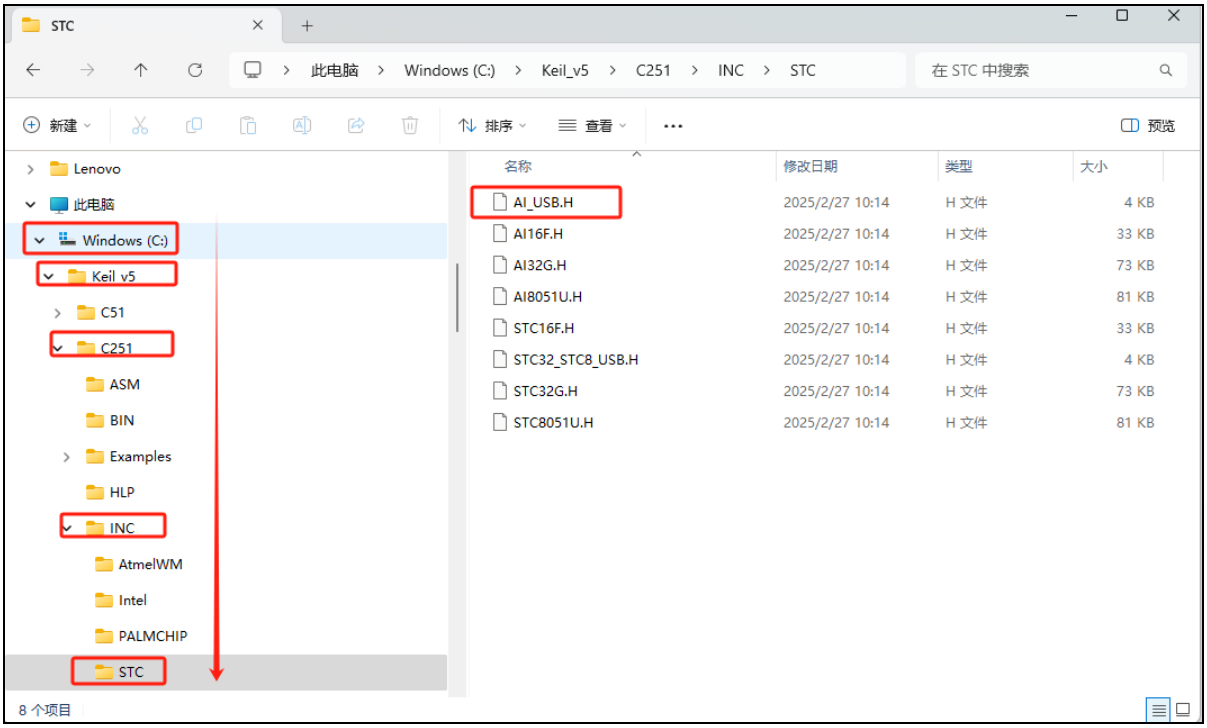
通过 USB-CDC 接口在串口助手接收窗口能直观的查看到对应端口的状态，并且提示点亮/熄灭了对应的 LED 灯。

b) 准备工作--添加头文件和库函数

1、添加常用头文件 ai_usb.h

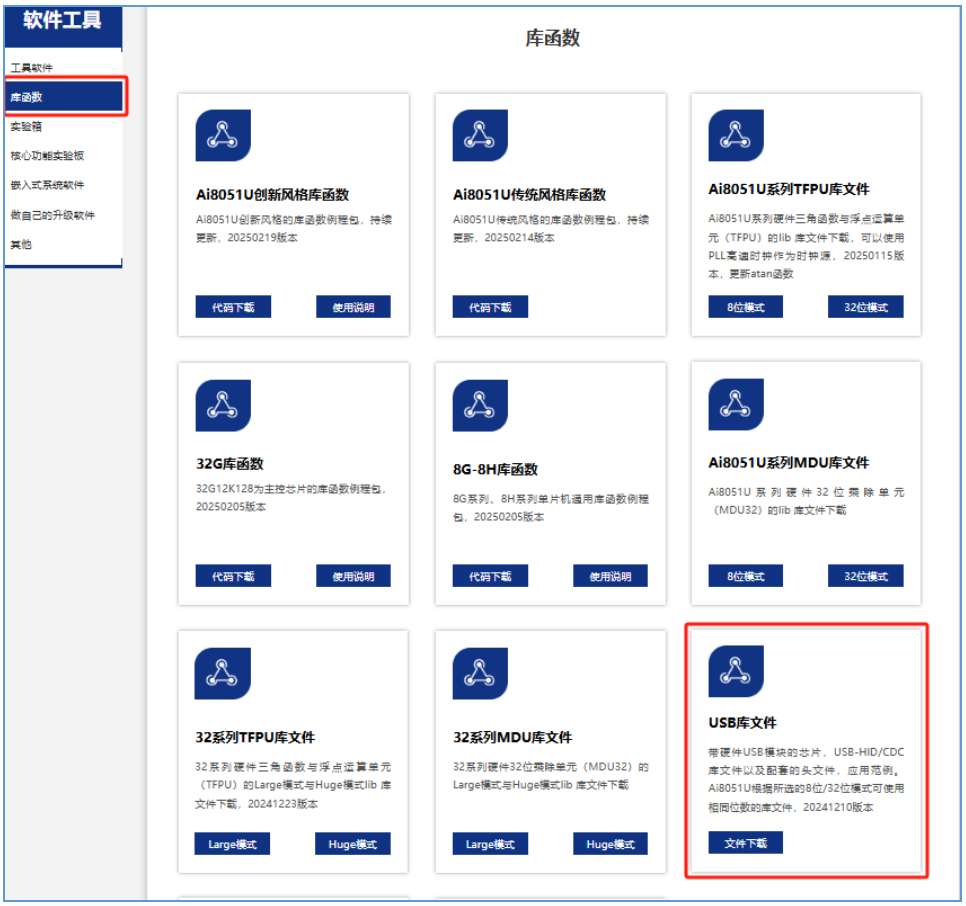
#include "ai_usb.h"　　//使用 usb 所需要包含的头文件。

这个头文件在使用 AIapp-ISP 软件添加型号和头文件到 Keil 中时就已经一并被添加到 Keil 中了。



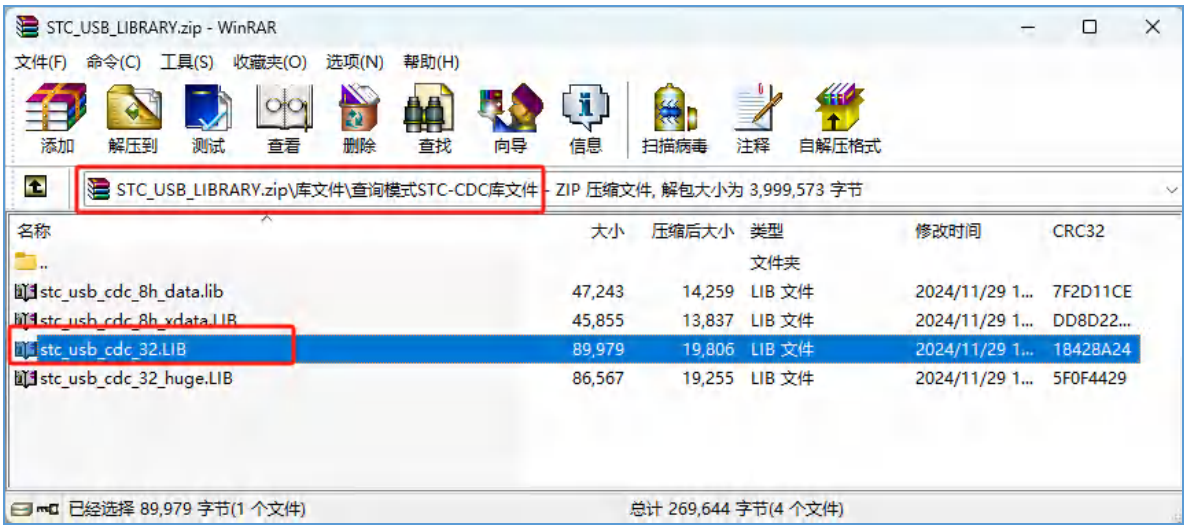
2、添加 usb_cdc 库函数文件

这部分可以到 STCAI 的官网：https://www.stcai.com/khs 的 USB 库文件中进行下载

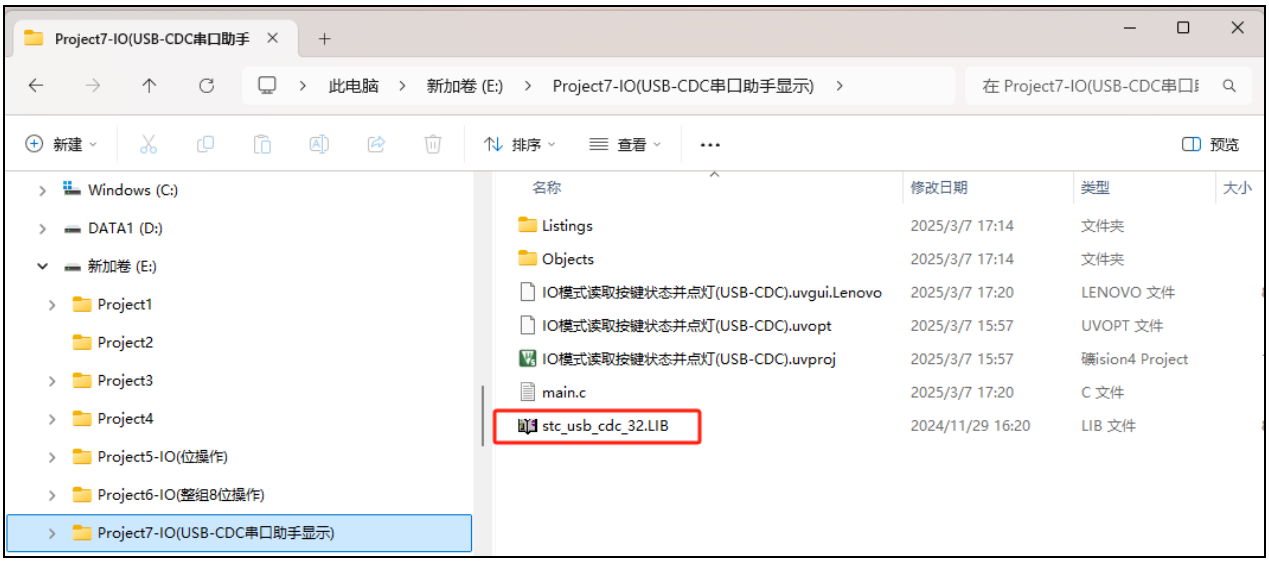


下载完成后，打开压缩包“STC_USB_LIBRARY.zip”，

在’库文件\查询模式 STC-CDC 库文件’中，复制出来 stc_usb_cdc_32.LIB

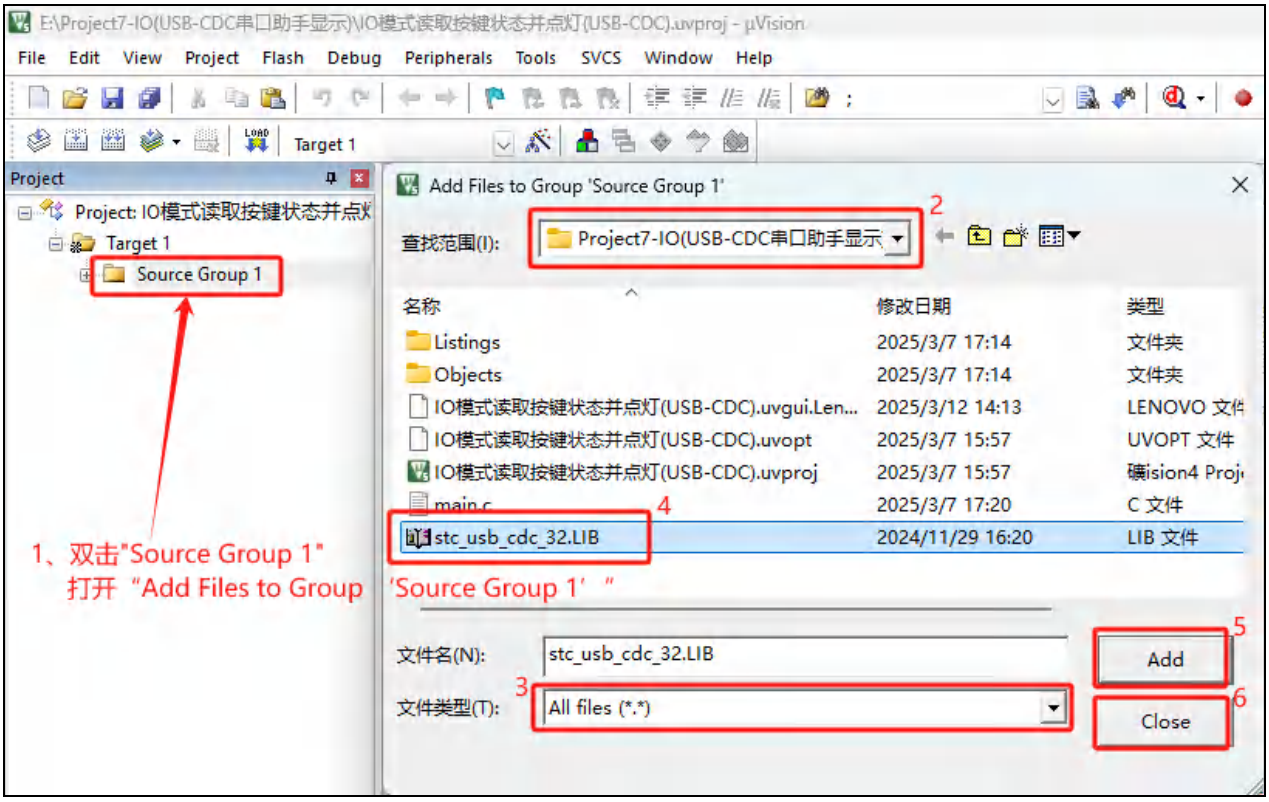


3、将库函数“stc_usb_cdc_32.LIB”复制到创建工程所在的文件夹 project7 中

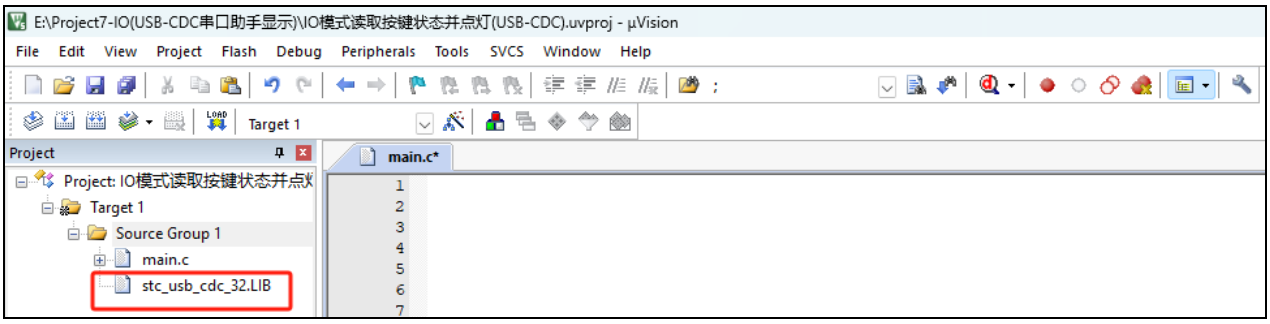


4、将库函数 stc_usb_cdc_32.LIB 添加进入此工程项目中

在 Keil 中，双击 Source Group，在弹出的 Add Files to Group 'Source Group 1 窗口中，将库函数 stc_usb_cdc_32.LIB 添加进入此工程，就可以使用 USB-CDC 功能进行端口通讯。



这样，Keil 界面变化如下图：



【注意】：USB-CDC 串口则更类似传统串口的形式，使用 COM 口进行通讯，但是只有 Win10 及其以上才默认携带 USB-CDC 的驱动，较低的系统版本可能要手动安装驱动才能使用。

c) 实验程序代码

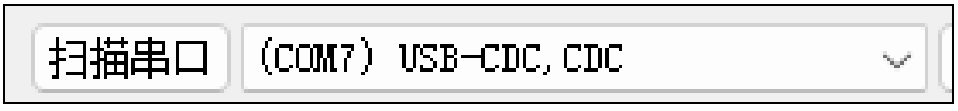
```
#include <AI8051U.H>          //包含 AI8051U 的头文件
#include   "ai_usb.h"         //USB 库和 USB 虚拟设备的头文件
//由于在 USB 库函数的代码中已经定义了，不停电 ISP 下载命令字"@STCISP#"，
//只需要在下载软件的“收到用户命令后复位到 ISP 监控程序区”选项卡中进行相应的设置，
//即可实现 USB 不停电 ISP 下载功能，或串口不停电 ISP 下载功能。
/*
如何获得上面这 2 个头文件，如已按照前面介绍的方法：
从 www.STCAI.com 网站，【软件工具 | 工具软件】处，下载安装最新的【AIapp-ISP 软件】，
选择其中的选项卡 【Keil 仿真设置 】，
点击【添加型号和头文件到 Keil 中，并添加仿真器驱动到 Keil 中】按钮，
则 Ai8051U.h 和 AI_usb.h 都会自动复制到 Keil 中，包含进来就可以直接使用，
大家可以打开看下其中的内容，AI_usb.h 和 stc32_stc8_usb.h 是相似的，只需用其中的 1 个头文件，
具体见 Keil_v5\C251\INC\STC 目录，或 Keil_v5\C51\INC\STC 目录。
如将对应的 USB 库文件添加到 项目中，后续用户程序中，USB 如收到命令字符串 "@STCISP#"
USB 库函数就会自动软复位到系统 ISP 程序区，实现 USB 不停电下载，具体见相应部分的说明。
*/

#include "intrins.h"          //使用_nop_()函数所必须要包含的头文件,
                              //否则延时函数中调用的_nop_()函数没有被头文件引用过来，
                              //会导致编译器找不到这个而函数而报错。

//char *USER_DEVICEDESC = NULL;
//char *USER_PRODUCTDESC = NULL;
//char *USER_STCISPCMD = "@STCISP#";
//使用 20250319 之前版本的库函数时需要在用户代码中增加上面 3 行定义。
//使用 20250319 及以后的版本都不再需要上面的定义

void Delay1ms(void)           //@40.000MHz
{
    unsigned long edata i;

    _nop_();
    _nop_();
    _nop_();
```

```
    i = 9998UL;
    while (i) i--;
}

void main(void)
{
    EAXFR = 1;          //允许访问扩展的特殊寄存器，XFR
    WTST = 0;           //设置取程序代码等待时间，赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;
        //设置访问片内的 xdata 速度，赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x38; P0M1 = 0x20;
        //设置 P03、P04 为推挽输出，P05 为开漏输出，其余为准双向口模式

    P1M0 = 0x00; P1M1 = 0x00;       //设置 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;       //设置 P2 口为准双向口模式

    P3M0 = 0x30; P3M1 = 0x3b;
        //P30、P31 为高阻输入，P32 为准双向口、P33 为高阻输入、P34 和 P35 为开漏输出

    P3PU |= 0x38;
        //打开 P33、P34、P35 的内部上拉电阻，并且仅修改这三个 I/O 的上拉电阻打开状态

    P3 |= 0x3c;
    //需要读的 I/O，对外输出锁存器先设置为 1，0011 1100B（0x3c）先和 P3 口逻辑或一下

//  P3M0 = 0x00; P3M1 = 0x00;       //设置 P3 口为准双向口模式
//  P3M0 = 0x00; P3M1 = 0x3c;       //设置 P32、P33、P34、P35 为高阻输入，其余引脚为准双向口模式
//  P3PU = 0x3c;                    //需要同时开启对应端口的上拉电阻，0011 1100B

    P4M0 = 0x00; P4M1 = 0x00;       //设置 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;       //设置 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;       //设置 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;       //设置 P7 口为准双向口模式

    Delay1ms();                     //延时 1ms，等待 I/O 口电平稳定，因为前面刚改变了模式
        //立刻读可能会读到错误的电平，所以需要先等待电平变化稳定后再读

    usb_init();     //调用 USB 初始化函数，不需要立即判断电脑已正确识别到 USB 从设备
    /* USB 型单片机从设备，如需要主动向电脑发送数据，
       在执行 USB_SendData( )函数和 printf_usb( )函数时，
       这两个函数已增加了判断电脑是否已正确识别到 USB 从设备的程序。
       如果电脑要主动发送数据给 USB 从设备，电脑自己会主动判断与 USB 从设备是否已正确连接。
    */

    EA = 1;                         //总中断允许位打开
```

```
    P40 = 0;                            //打开 LED 部分的供电

    while(1)
    {                                   //使用 USB 串口控制来手动控制 LED 部分
        if(bUsbOutReady)
            //USB 库中有定义 bUsbOutReady 位变量，查询方式必须调用一次“usb_OUT_done();”
        {
            printf_usb("P3%d 当前的状态为:%d\r\n",(int)UsbOutBuffer[0],(int)(P3>>UsbOutBuffer[0])&1);
            //传入 hex 的位号来标识需要读取哪个位的状态，
            //读取后通过对整组 I/O 口数据的移位操作读取到对应 I/O 的状态
            //直接打印输出 “P3%d 当前的状态为: %d” 到主机相应软件的 USB【CDC/HID-串口助手】

//          P0 = P3|~(1<<UsbOutBuffer[0]);      //同步端口状态，并将其他不需要的位置 1（不显示）

//下面的 switch 等同于上一句，上一句 C 语言高手才能看懂
            switch (UsbOutBuffer[0])            //判断接收的第一个字节数据
            {
                case 2: P0 = P3 | 0xfb; break;  //如果接收到 2 则将 P3.2 状态送 P0.2，
                                                //其他不需要的位置 1（不显示），1111 1011B
                case 3: P0 = P3 | 0xf7; break;  //如果接收到 3 则将 P3.3 状态送 P0.3，
                                                //其他不需要的位置 1（不显示），1111 0111B
                case 4: P0 = P3 | 0xef; break;  //如果接收到 4 则将 P3.4 状态送 P0.4，
                                                //其他不需要的位置 1（不显示），1110 1111B
                case 5: P0 = P3 | 0xdf; break;  //如果接收到 5 则将 P3.5 状态送 P0.5，
                                                //其他不需要的位置 1（不显示），1101 1111B
            }
            printf_usb("已将状态同步给 LED0%d\r\n",(int)UsbOutBuffer[0]);       //显示同步完成
            //直接打印输出 “已将状态同步给 LED0%d” 到主机相应软件的 USB【CDC/HID-串口助手】

            usb_OUT_done();                 // 本包接收的数据处理完毕，准备接收下一包数据
                //当使用查询方式 if(bUsbOutReady)，查询调用 USB 库函数时，在数据处理完成后，
                //必须调用一次“usb_OUT_done();”，这样才能接收下一包数据。
        }
    }
}
```

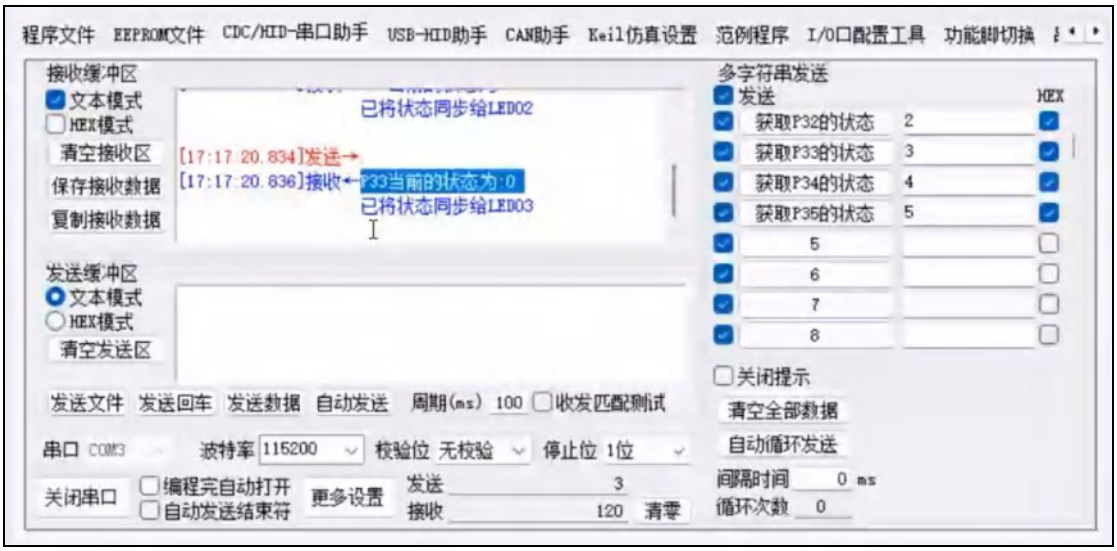
d) 程序下载

程序下载完成以后，就可以看到出现了一个 USB-CDC 端口，此时就可以通过这个 USB-CDC 端口进行数据通讯了。



e) 程序实现效果

程序编写好以后，编译无误后加载到实验箱，就可以通过 AIapp-ISP 右侧的多字符串发送部分，通过设置对应的文字和需要发送的内容。实现**按下对应按键后，**读取对应端口的状态并能进行回传，在接收窗口就能看到相应的接收内容。



14.14.11 实验 3A--使用【AiCube】项目创建助手实现“用 USB-CDC 读取按键状态在串口助手显示并点亮 LED 灯”

STCAI 在最新版的 AIapp-ISP 软件中，推出【AiCube】项目创建助手工具，目前支持 Ai8051U-32Bit 各功能模块的智能化配置，并生成程序框架。这项功能在程序员建立项目时，简化了项目的初始化难度，提高效率，避免因不太熟悉或不完全了解该单片机的性能而忽略个别配置，引起编译错误等问题。尤其对刚涉足单片机的初学者，此工具不仅引导其了解单片机编程流程，还极大的降低了入门门槛，缩短了摸索的时间，提高入门的兴趣和信心。现在我们使用这个工具来生成实现此实验功能的配置和代码。

a) 安装 AIapp-ISP 下载/编程/烧录/软件

安装 STCAI 公司研发的 AIapp-ISP 下载 / 编程 / 烧录 / 软件，含强大的辅助开发工具包。登录 STCAI 官网，下载最新版的 AIapp-ISP 安装包，下载链接如下：https://www.stcai.com/gjrj

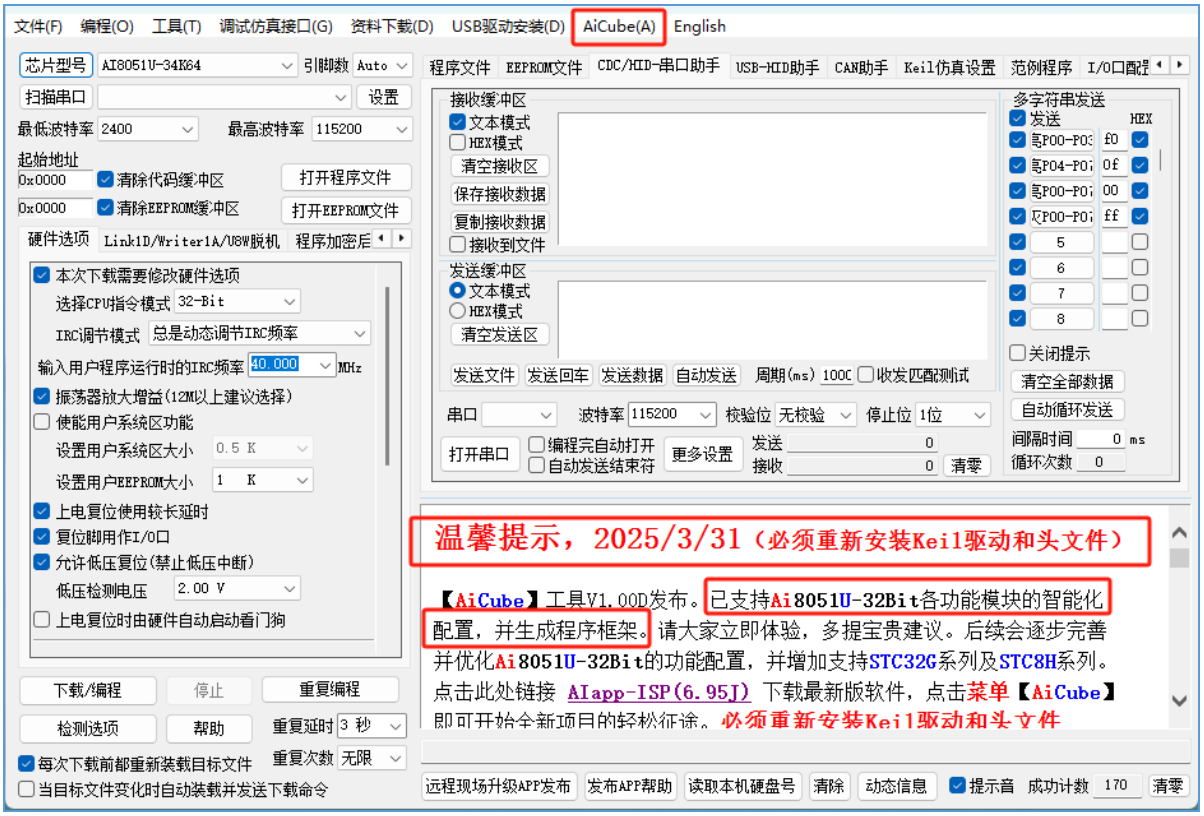
下载下来的是压缩包文件如下图：

双击后解压缩如下图：

如果是解压缩到非桌面的其他目录，如 D:\STCAI-ISP，建议创建快捷方式到桌面方便打开。这实际是一个免安装的绿色软件。

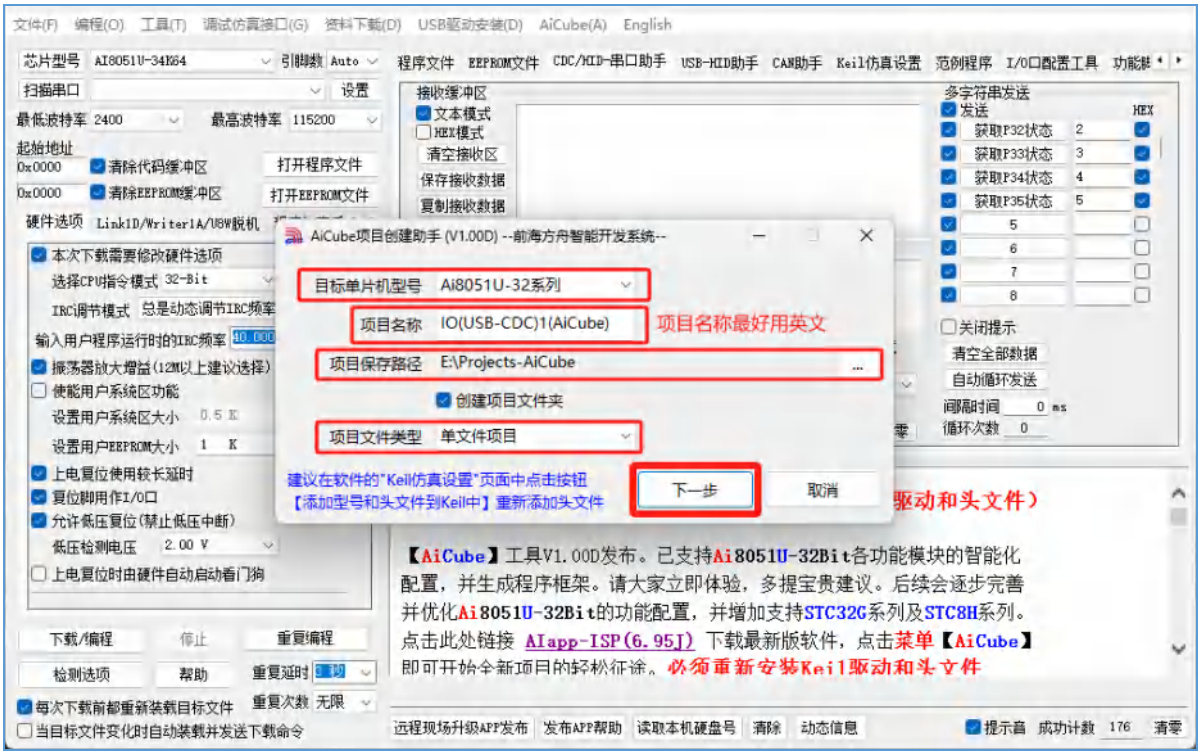
b) 打开【AiCube】助手

双击图标，即可打开 AIapp-ISP 软件界面，即可直接使用，如下图：

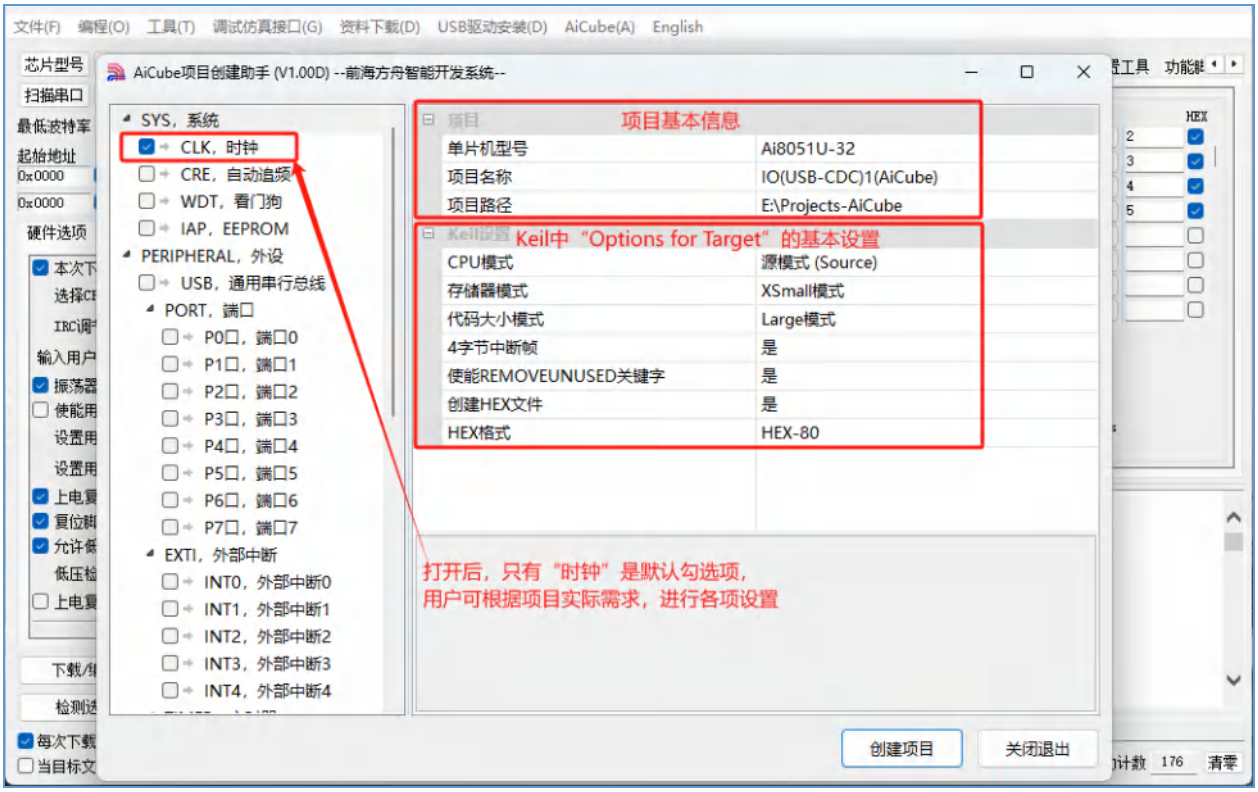


c) 使用【AiCube】助手创建项目

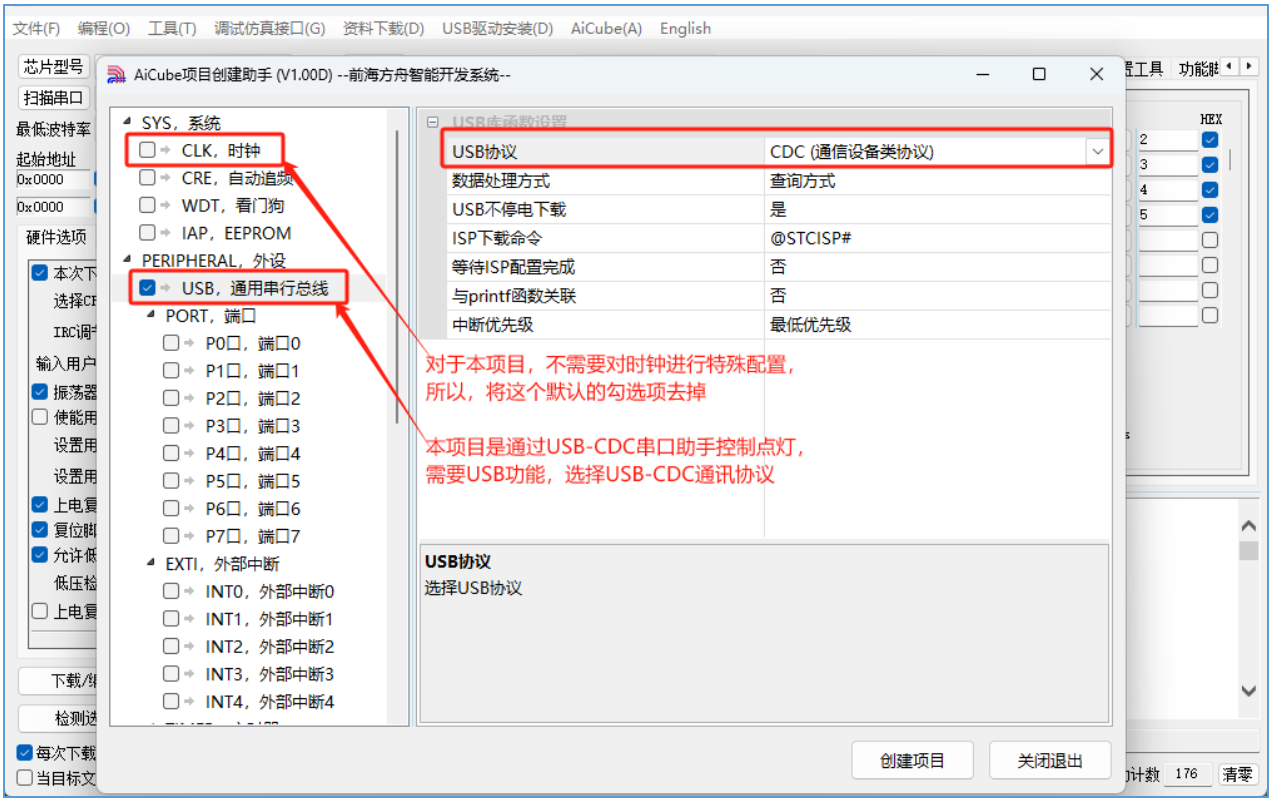
点击【AiCube（A）】，弹出如下窗口：



选择并填写项目基本信息，建议项目名称用英文！点击“下一步”，出现如下配置界面：



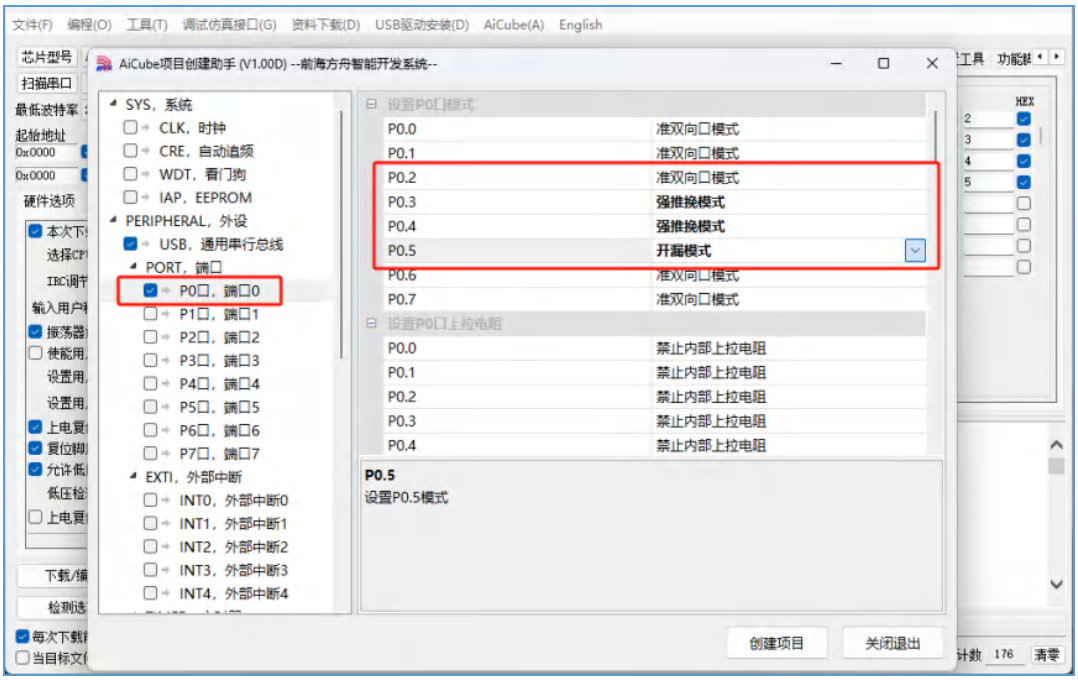
本实验是通过【AIapp-ISP】系统软件中的 USB-CDC 串口助手通知单片机读【P3.2/P3.3/P3.4/P3.5】的状态，送回 USB-CDC 串口助手显示，同时将相应的状态送到 LED【P0.2/P0.3/P0.4/P0.5】显示（点亮/熄灭）。显然要勾选 USB 选项，选择“CDC（通信设备类协议）”，如下图：



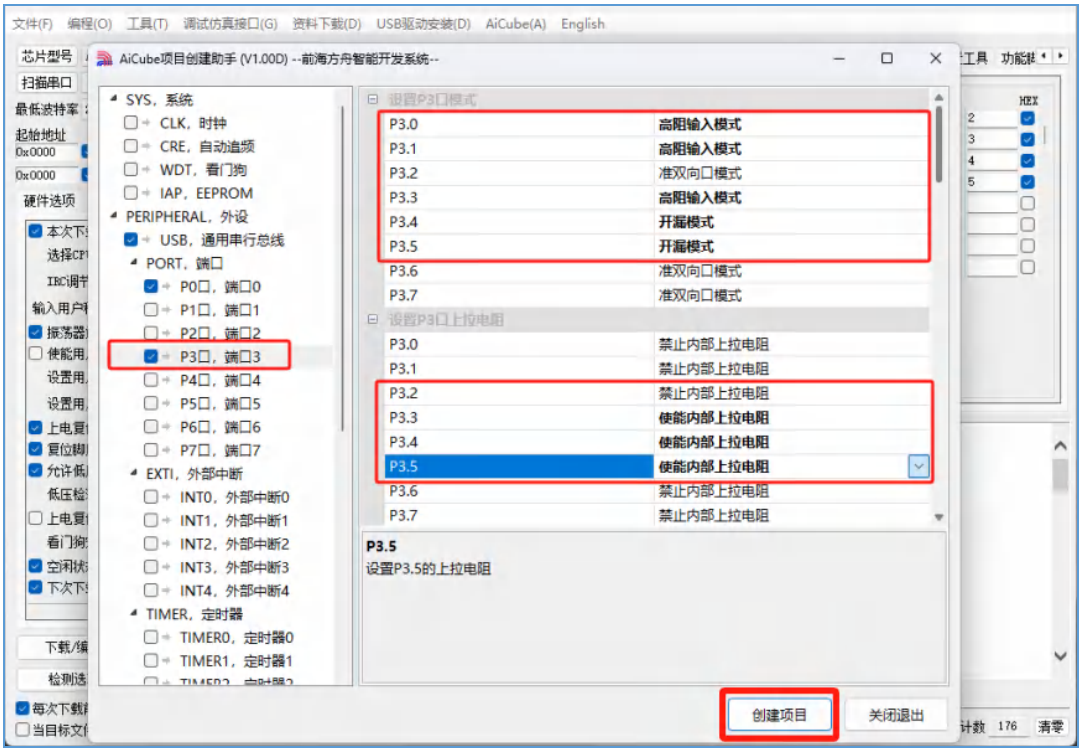
对于本项目，需要单片机读【P3.2 / P3.3 / P3.4 / P3.5】这 4 个 I/O 的状态，
输出到【P0.2 / P0.3 / P0.4 / P0.5】对应的指示灯

针对 I/O 有 4 种不同的模式，我们将【P3.2 / P3.3 / P3.4 / P3.5】不同的 I/O 口设置为不同的模式，验证不同情况下依然可以正常输入/输出。

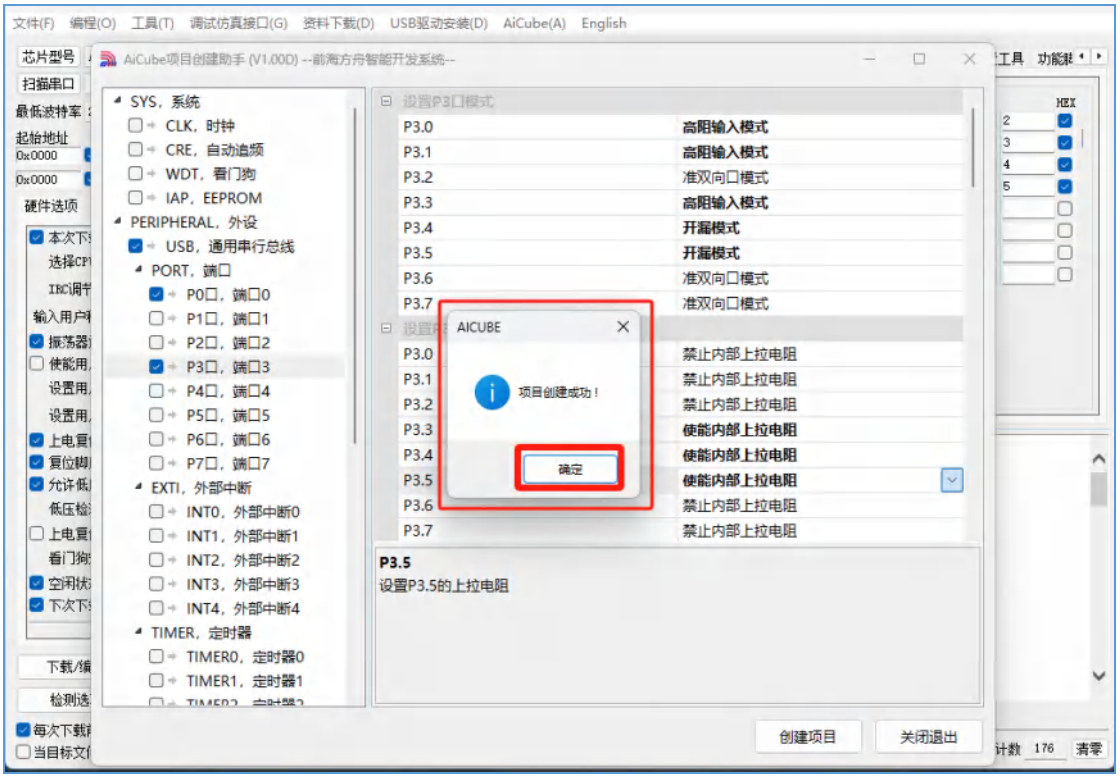
【AiCube】助手中，端口默认是准双向口模式，我们配置输出端口的 P02 保持准双向口模式，P03、P04 为强推挽输出，P05 为开漏输出（可以输出 0 点亮 LED）。如下图：



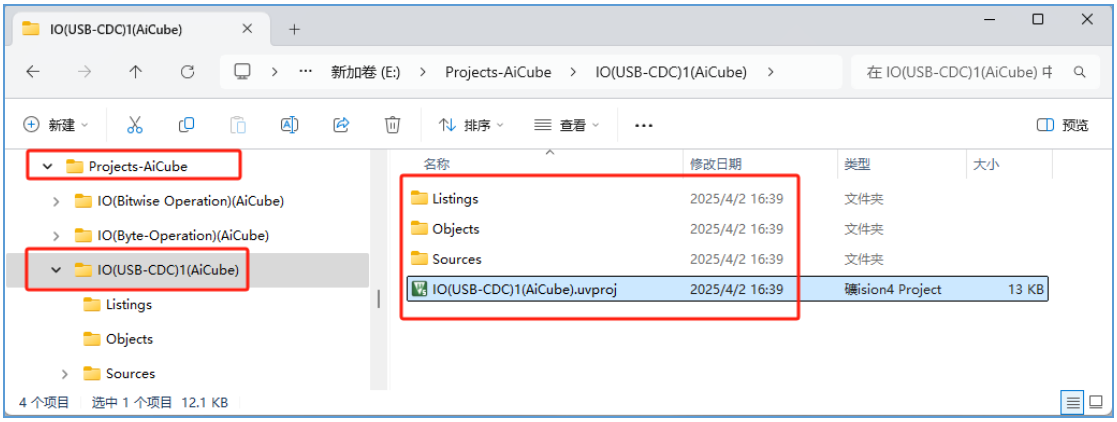
输入配置 P30、P31 为高阻输入(因为跟 USB 复用，建议设置为高阻输入，不干扰 USB 的通讯控制)
设置 P32 为准双向口、P33 为高阻输入、P34 和 P35 为开漏模式（设置为 1 等效高阻输入）
然后开启 P33、P34、P35 的内部上拉电阻以稳定电平状态（不开启内部上拉电阻读外部状态，高阻输入/开漏模式，外部都需要有上拉电阻配合）
【AiCube】助手中，端口默认是准双向口模式，内部上拉电阻默认是禁用设置。现在我们按上述要求配置 P30、P31 为高阻输入；P32 为保持准双向口；P33 为高阻输入；P34、P35 为开漏模式；打开 P33、P34、P35 的内部上拉电阻。如下图：



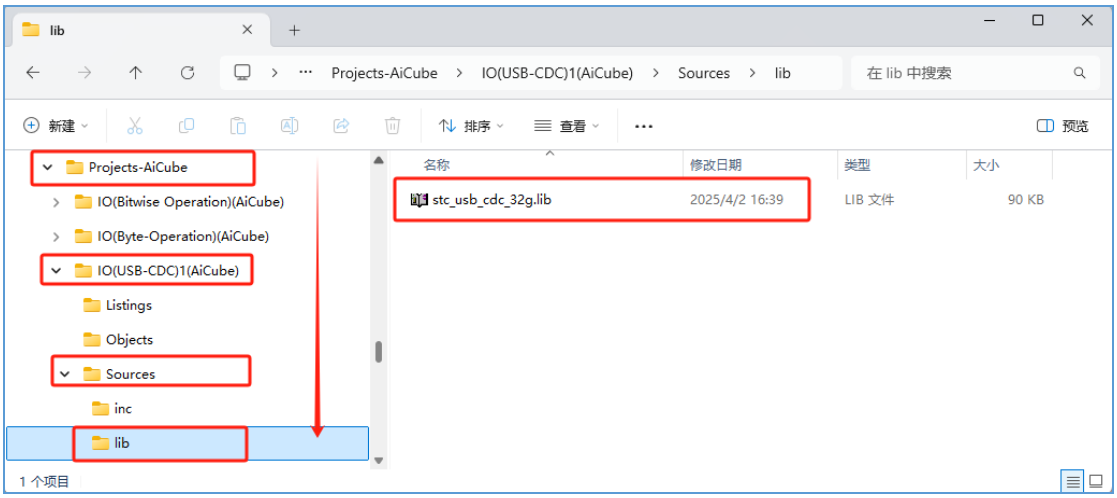
点击“创建项目”，【AiCube】助手提示：“项目创建成功！”，如下图：



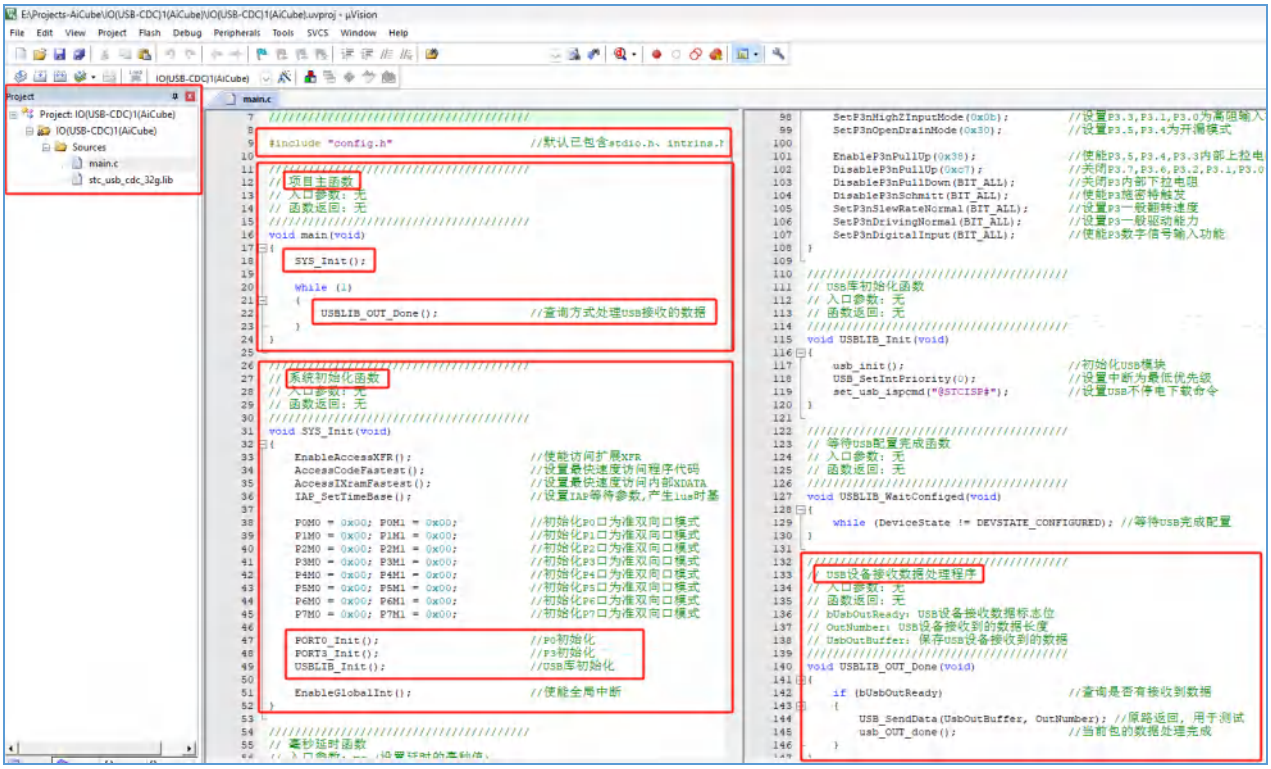
点击“确定”，即成功创建了项目及项目文件目录“IO(USB-CDC)1(AiCube)”，找到并打开刚刚创建的项目目录。发现刚刚的操作过程，【AiCube】助手已经帮我们建立了项目目录及 IO(USB-CDC)1(AiCube).uvproj 文件，如下图：



因为勾选了 USB 选项，打开 Sources>lib 子目录，发现，【AiCube】助手连 USB 库函数也已经加载了：



双击打开项目文件：IO(USB-CDC)1(AiCube).uvproj，如下图（为方便查看，使用菜单“Window”→“Split”将原窗口拆分成两个窗口，方便观察）：

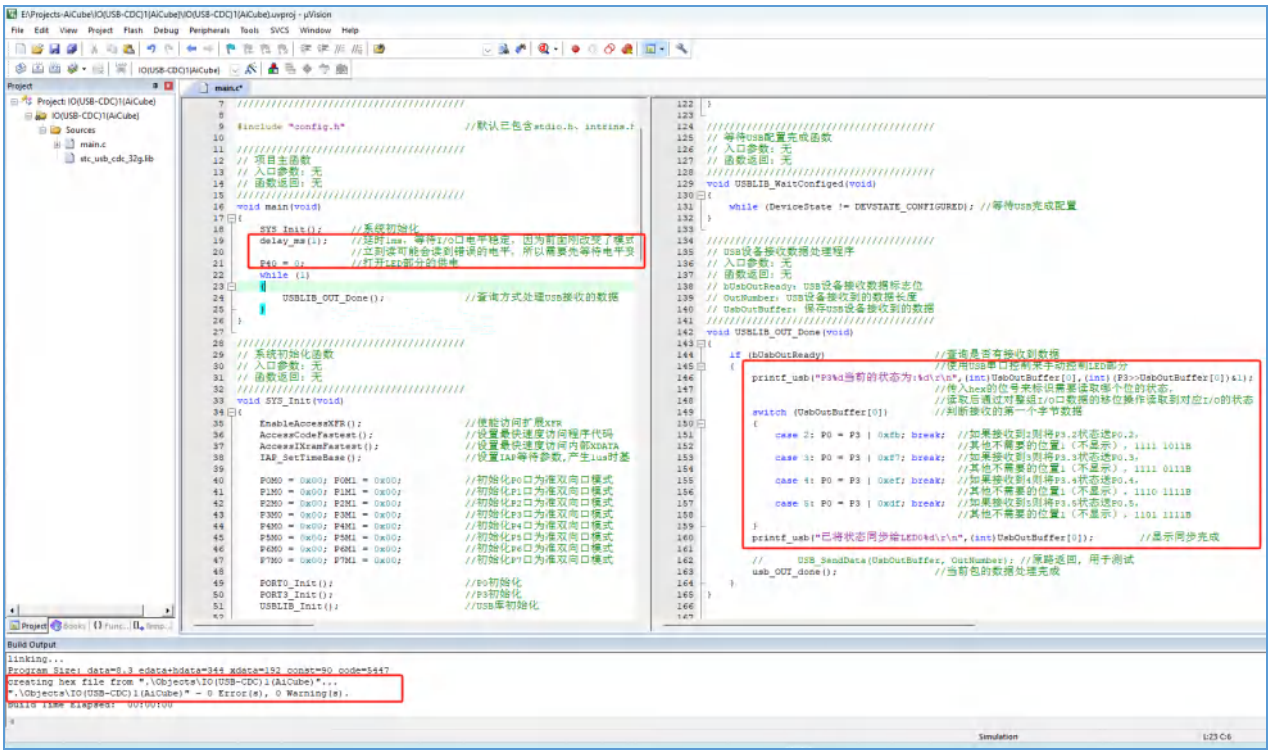


根据在【AiCube】助手中的配置，从上图中我们看到【AiCube】助手创建的本实验项目中有了 main.c，并自动加载 USB 库函数（stc_usb_cdc_32g.lib）。main.c 中按模块化设计，已经包含了：必要的头文件、主函数、系统初始化函数、延时函数、P0 口初始化函数、P3 口初始化函数、USB 库初始化函数、等待 USB 配置完成函数、USB 设备接收数据处理程序。

d) 增加实验项目代码

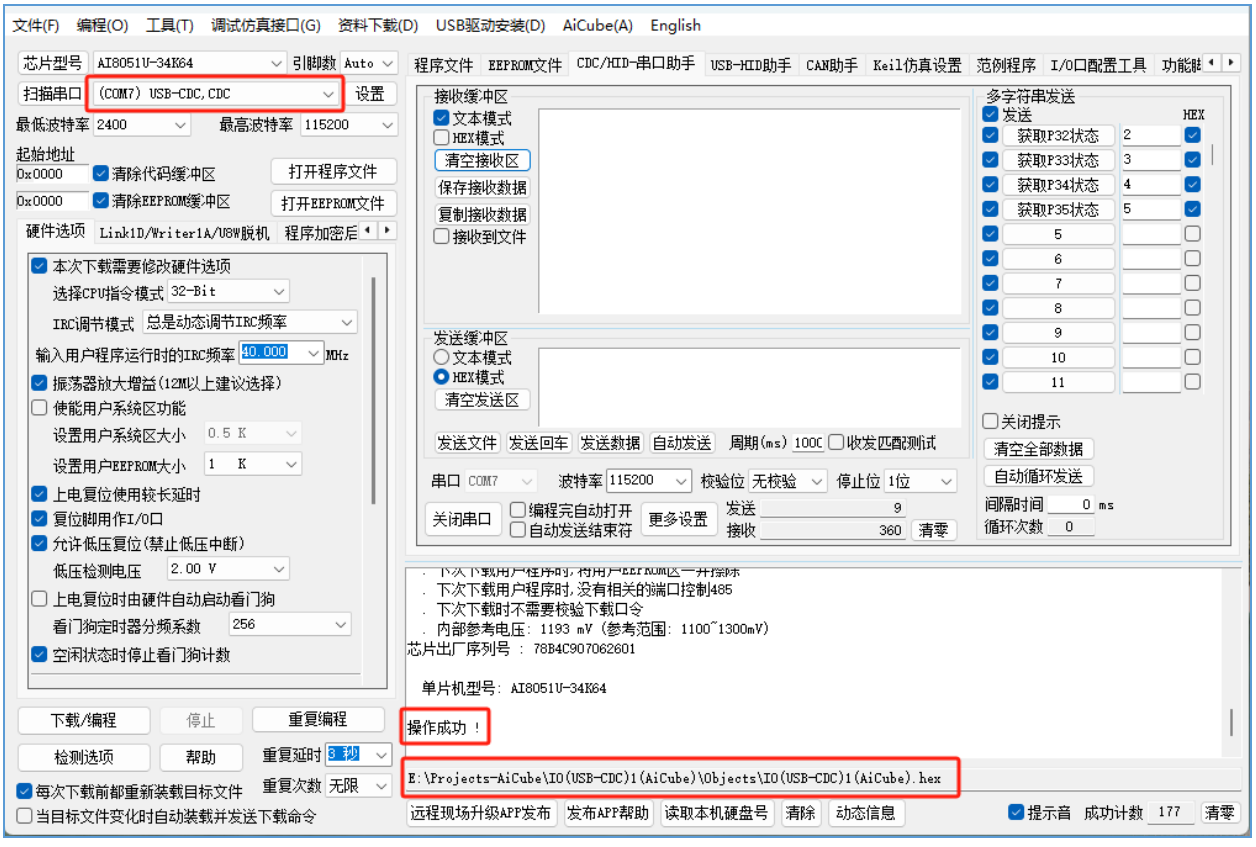
为实现通过 AIapp-ISP 系统软件中的 USB-CDC 串口助手读取【P3.2/P3.3/P3.4/P3.5】的状态，送回 USB-CDC 串口助手显示，同时将相应的状态送到 LED 的【P0.2/P0.3/P0.4/P0.5】显示（点亮/熄灭）。我们现在需要：

- 在主函数中，调用延时函数“delay_ms(1);”，延时 1ms（等待 I/O 口电平稳定）；
- 在主函数中，加入“P40 = 0;”，打开 LED 供电的代码；
- 在“USB 设备接收数据处理程序”子函数中，补充 USB 控制代码；
- 保存，编译，如下图（为方便查看，使用菜单“Window”→“Split”将原窗口拆分成两个窗口，对比观察）：

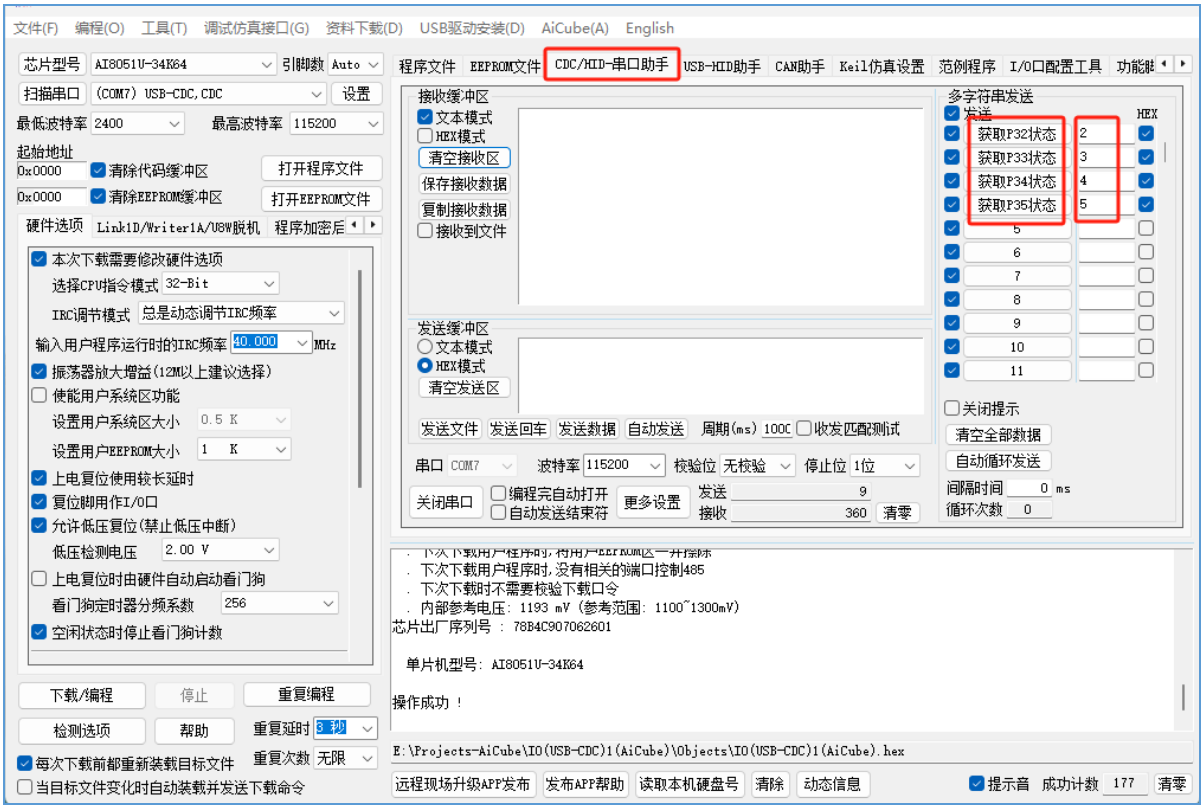


e) 使用 AIapp-ISP 软件系统并连接 Ai8051U 实验箱验证实验效果

连接好 Ai8051U 实验箱，运行 AIapp-ISP 软件系统，按步骤打开并下载程序“IO(USB-CDC)1(AiCube).hex”到实验箱，如下图：



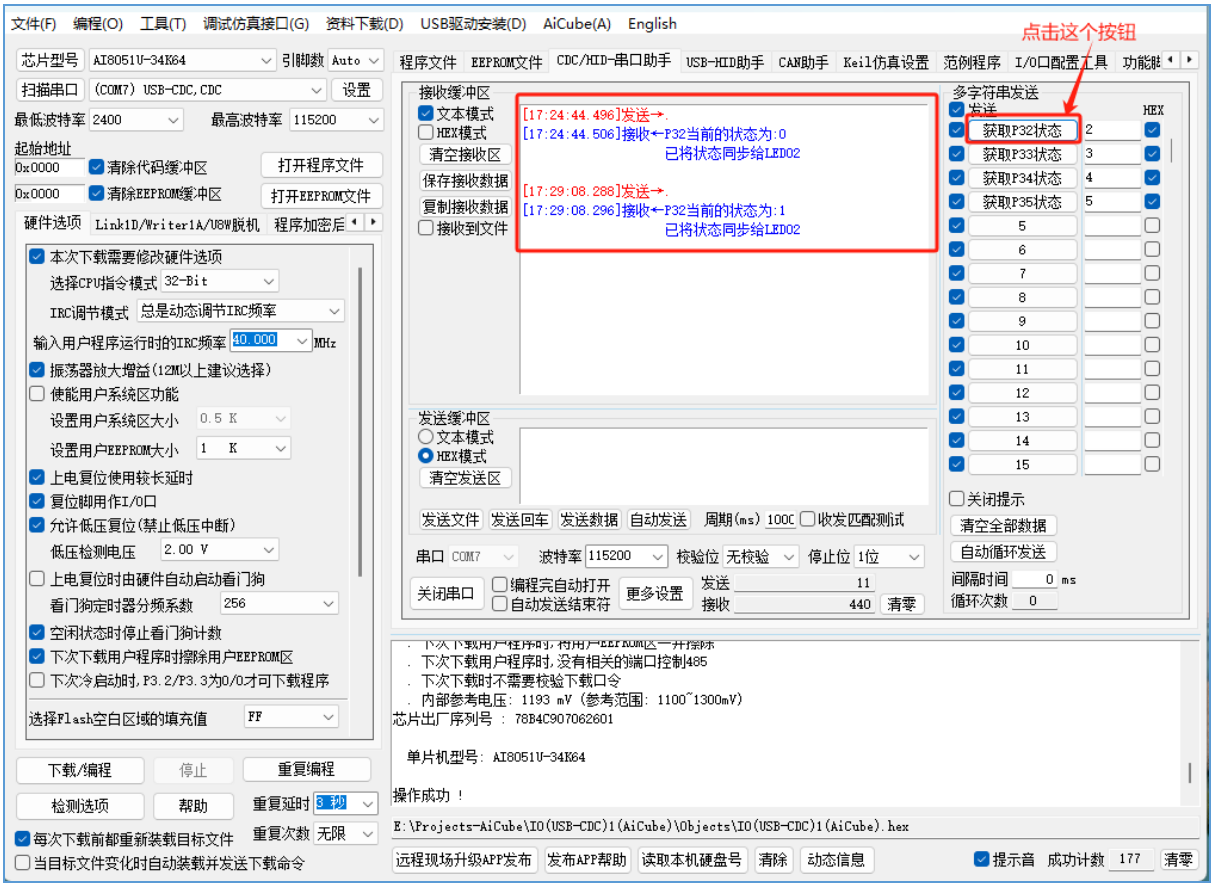
“CDC/HID-串口助手”选项卡中，在“多字符串发送”窗口，设置好发送控制命令，以便观察实验效果，如下图：



一切准备就绪！

按住 Ai8051U 实验箱上的 P3.2 按键，点击“获取 P32 状态”按钮，控制实验箱上的 P02 灯亮了；松开 P3.2 按键，P02 灯依然是亮的，再点击“获取 P32 状态”按钮，P02 灯灭。

在 USB-CDC 接收窗口就能看到相应的接收内容，如下图：

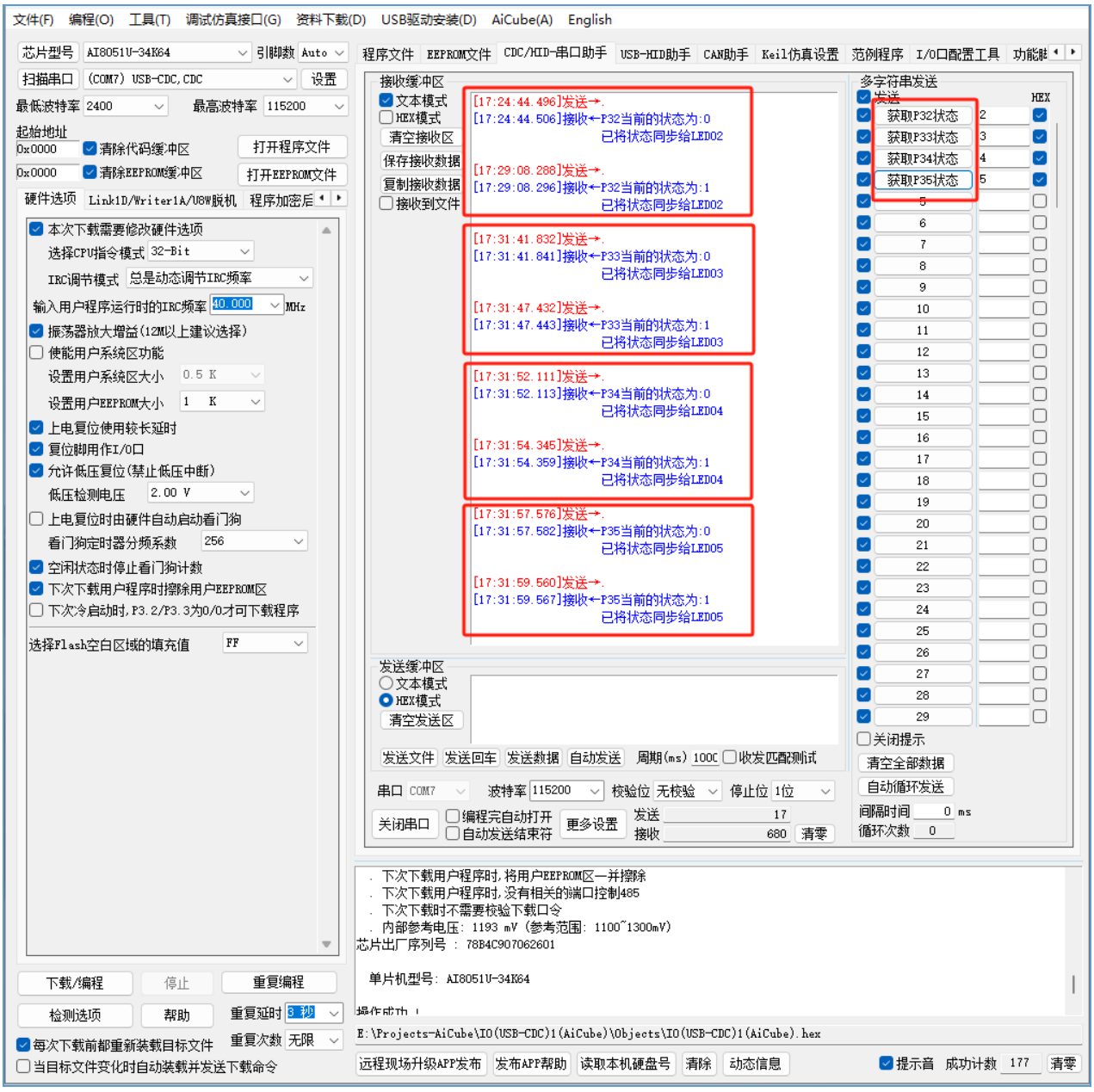


按住 Ai8051U 实验箱上的 P3.3 按键，点击“获取 P33 状态”按钮，控制实验箱上的 P03 灯亮了；松开 P3.3 按键，P03 灯依然是亮的，再点击“获取 P33 状态”按钮，P03 灯灭。

按住 Ai8051U 实验箱上的 P3.4 按键，点击“获取 P34 状态”按钮，控制实验箱上的 P04 灯亮了；松开 P3.4 按键，P04 灯依然是亮的，再点击“获取 P34 状态”按钮，P04 灯灭。

按住 Ai8051U 实验箱上的 P3.5 按键，点击“获取 P35 状态”按钮，控制实验箱上的 P05 灯亮了；松开 P3.5 按键，P05 灯依然是亮的，再点击“获取 P35 状态”按钮，P05 灯灭。

在 USB-CDC 接收窗口就能看到相应的接收内容，如下图：



有了【AiCube】助手，单片机编程就是这么简单！快用起来吧！

14.14.12　实验 4：用 USB-CDC 读取按键状态，在 ISP 软件的 【调试仿真接口 | 擎天柱-LED-DIP40】 虚拟显示

电脑 AIapp-ISP 软件中的，USB-CDC 串口助手通知单片机读【P3.2/P3.3/P3.4/P3.5】的状态，送到 AIapp-ISP 软件中的【调试仿真接口 | 擎天柱-LED-PID40】虚拟显示，同时将相应的状态送到 LED【P0.2/P0.3/P0.4/P0.5】显示（点亮/熄灭）

a) 通信协议

AIapp-ISP 软件界面的“调试仿真接口”→“接口协议及帮助”→“DIP40 封装 LED 接口”

功能2：控制DIP40的指定组管脚上LED的状态

命令格式： 4CH 45H 44H 28H 02H x1 px

命令说明：

第1～4字节：命令头

第5字节：后面有效数据长度

第6字节：P0～P5端口的屏蔽位
（01H:P0, 02H:P1, 04H:P2, 08H:P3, 10H:P4, 20H:P5）

第7字节：端口的状态

使用串口通讯请参考下面的示例发送串口数据

示例1： 发送4CH 45H 44H 28H 02H 01H 55H
设置P0口的状态为55H

示例2： 发送4CH 45H 44H 28H 02H 04H AAH
设置P2口的状态为AAH

库函数声明：
```
void LED40_SetPort(BYTE port, BYTE dat);
```
*** 包含库函数的头文件后，不用另外声明 ***

项目中包含库函数后，可直接调用下面的库函数

库函数调用：
```
LED40_SetPort(0, 0x55);     //设置P0口输出55H
LED40_SetPort(1, 0xaa);     //设置P1口输出AAH
```

b) 实验介绍

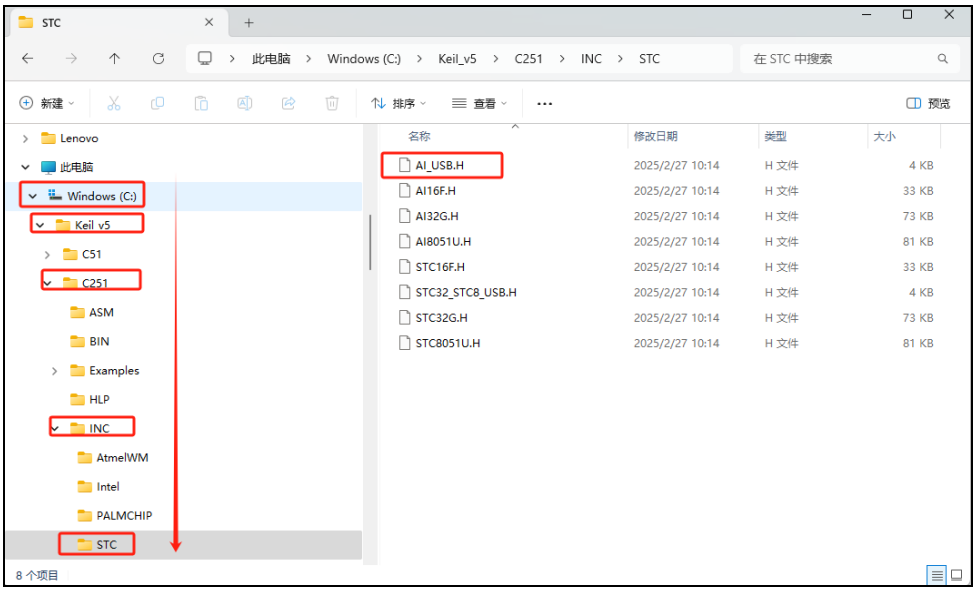
通过 USB-CDC 接口在“擎天柱调试接口”能直观的查看到对应端口的状态，并且点亮/熄灭了对应的 LED 灯。

c) 准备工作--添加头文件和库函数

1、添加常用头文件 ai_usb.h

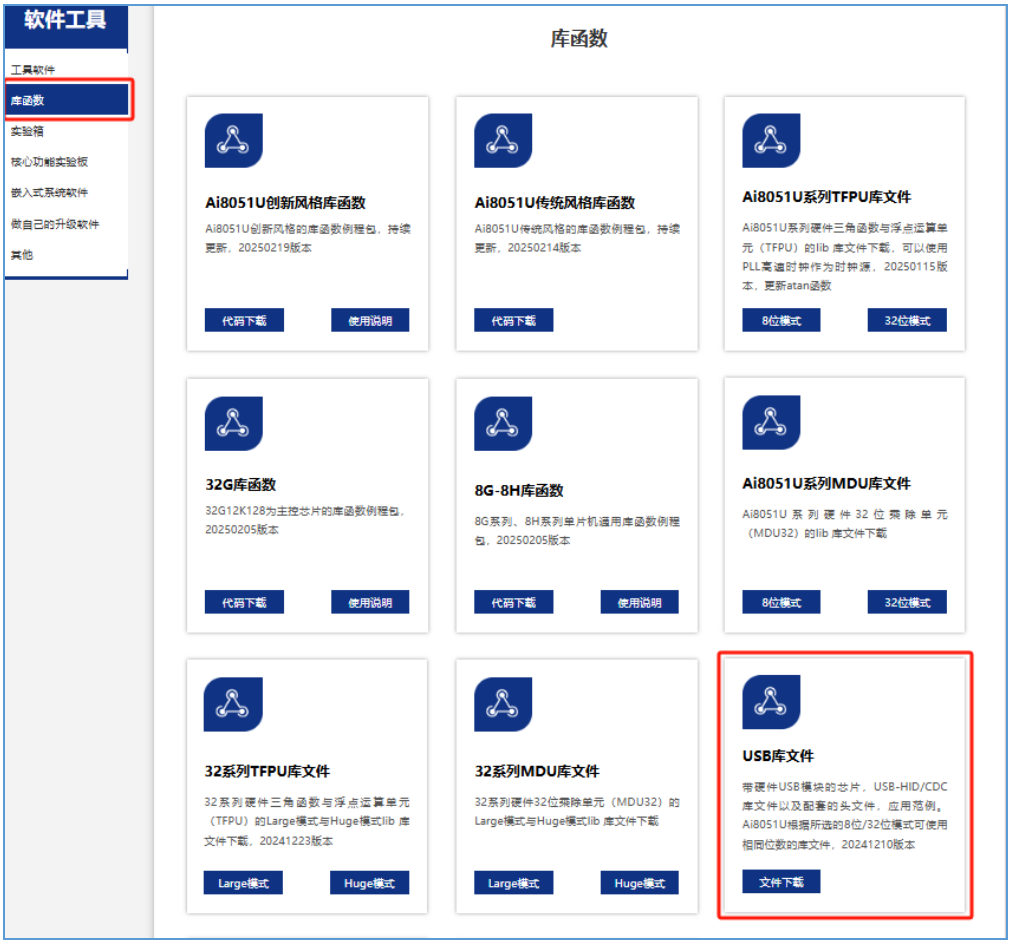
```
#include "ai_usb.h"                    //使用 usb 所需要包含的头文件。
```

这个头文件在使用 AIapp-ISP 软件添加型号和头文件到 Keil 中时就已经一并被添加到 Keil 中了。



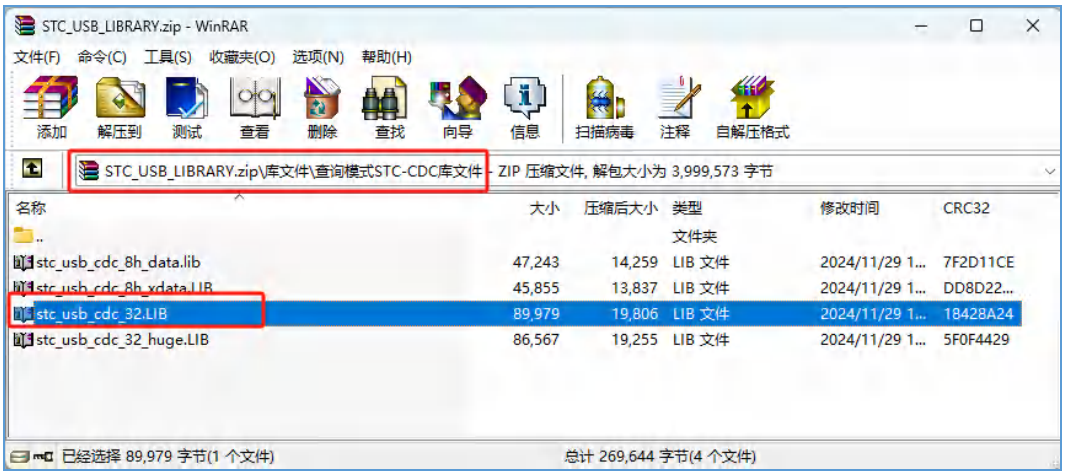
2、添加 usb_cdc 库函数文件

这部分可以到 STCAI 的官网：https://www.stcai.com/khs 的 USB 库文件中进行下载

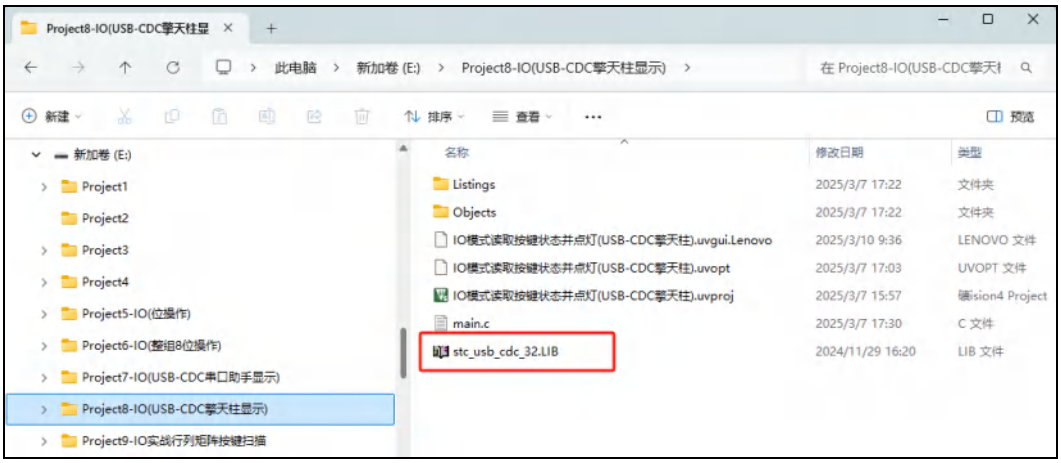


下载完成后，打开压缩包“STC_USB_LIBRARY.zip”，

在’库文件\查询模式 STC-CDC 库文件’中，复制出来 stc_usb_cdc_32.LIB

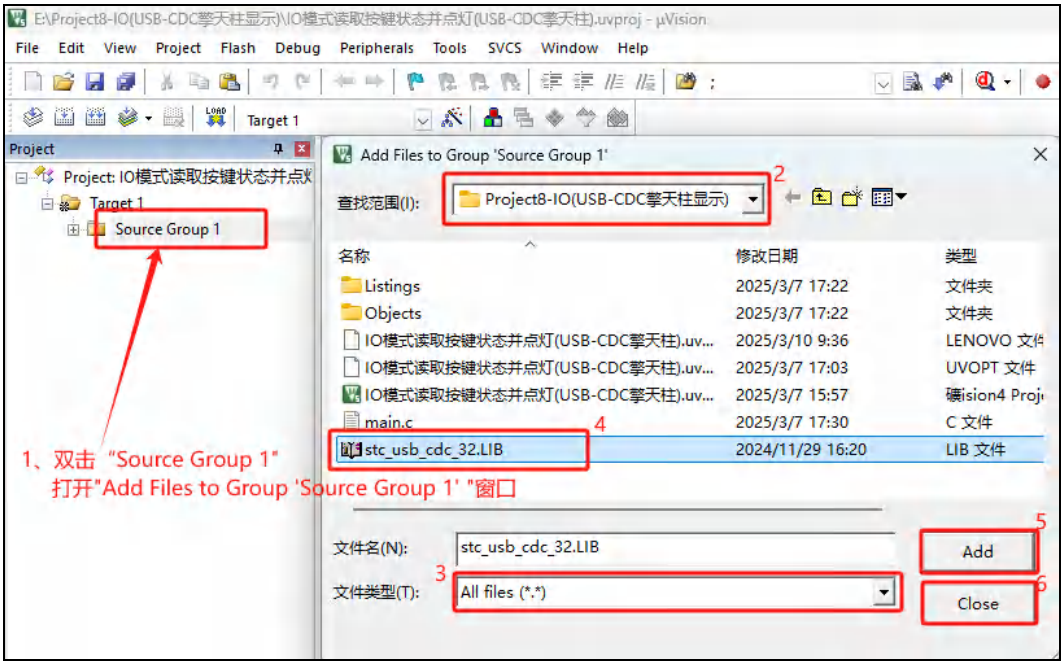


3、将库函数“stc_usb_cdc_32.LIB”复制到创建工程所在的文件夹 project8 中

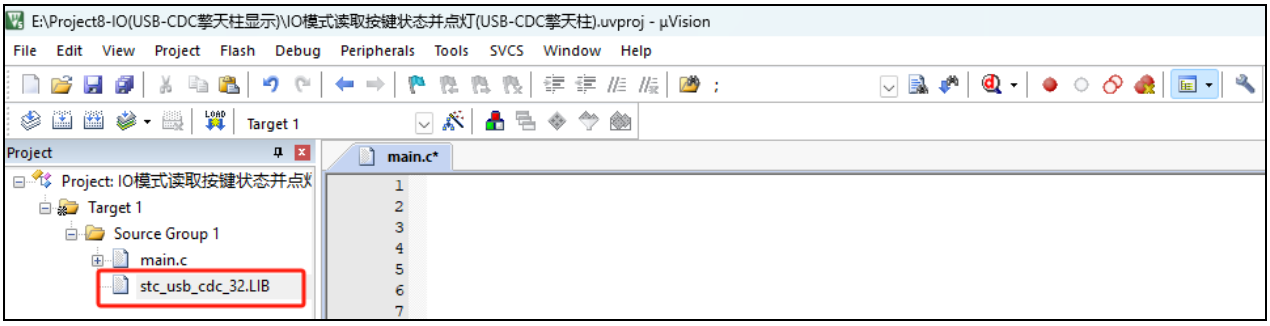


4、将库函数 stc_usb_cdc_32.LIB 添加进入此工程项目中

在 Keil 中，双击 Source Group，在弹出的 Add Files to Group 'Source Group 1 窗口中，将库函数 stc_usb_cdc_32.LIB 添加进入此工程，就可以使用 USB-CDC 功能进行端口通讯。



这样，Keil 界面变化如下图：



【注意】：USB-CDC 串口则更类似传统串口的形式，使用 COM 口进行通讯，但是只有 Win10 及其以上才默认携带 USB-CDC 的驱动，较低的系统版本可能要手动安装驱动才能使用。

d) 实验程序代码

```
#include <AI8051U.H>          //包含 AI8051U 的头文件
#include   "ai_usb.h"          //USB 库和 USB 虚拟设备的头文件
//由于在 USB 库函数的代码中已经定义了，不停电 ISP 下载命令字"@STCISP#"，
//只需要在下载软件的“收到用户命令后复位到 ISP 监控程序区”选项卡中进行相应的设置，
//即可实现 USB 不停电 ISP 下载功能，或串口不停电 ISP 下载功能。
/*
如何获得上面这 2 个头文件，如已按照前面介绍的方法：
从 www.STCAI.com 网站，【软件工具 | 工具软件】处，下载安装最新的【AIapp-ISP 软件】，
选择其中的选项卡 【Keil 仿真设置 】，
点击【添加型号和头文件到 Keil 中，并添加仿真器驱动到 Keil 中】按钮，
则 Ai8051U.h 和 AI_usb.h 都会自动复制到 Keil 中，包含进来就可以直接使用，
大家可以打开看下其中的内容，AI_usb.h 和 stc32_stc8_usb.h 是相似的，只需用其中的 1 个头文件，
具体见 Keil_v5\C251\INC\STC 目录，或 Keil_v5\C51\INC\STC 目录。
如将对应的 USB 库文件添加到 项目中，后续用户程序中，USB 如收到命令字符串 "@STCISP#"
USB 库函数就会自动软复位到系统 ISP 程序区，实现 USB 不停电下载，具体见相应部分的说明。
*/

#include "intrins.h"          //使用_nop_()函数所必须要包含的头文件,
                              //否则延时函数中调用的_nop_()函数没有被头文件引用过来，
                              //会导致编译器找不到这个而函数而报错。
//char *USER_DEVICEDESC = NULL;
//char *USER_PRODUCTDESC = NULL;
//char *USER_STCISPCMD = "@STCISP#";
//使用 20250319 之前版本的库函数时需要在用户代码中增加上面 3 行定义。
//使用 20250319 及以后的版本都不再需要上面的定义

void Delay1ms(void)           //@40.000MHz
{
    unsigned long edata i;

    _nop_();
    _nop_();
    _nop_();
```

```
    i = 9998UL;
    while (i) i--;
}

void main(void)
{
    EAXFR = 1;          //允许访问扩展的特殊寄存器，XFR
    WTST = 0;           //设置取程序代码等待时间，赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;
        //设置访问片内的 xdata 速度，赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x38; P0M1 = 0x20;
        //设置 P03、P04 为推挽输出，P05 为开漏输出，其余为准双向口模式

    P1M0 = 0x00; P1M1 = 0x00;       //设置 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;       //设置 P2 口为准双向口模式

    P3M0 = 0x30; P3M1 = 0x3b;
        //P30、P31 为高阻输入，P32 为准双向口、P33 为高阻输入、P34 和 P35 为开漏输出
    P3PU |= 0x38;
        //打开 P33、P34、P35 的内部上拉电阻，并且仅修改这三个 I/O 的上拉电阻打开状态

    P3 |= 0x3c;
    //需要读的 I/O，对外输出锁存器先设置为 1，0011 1100B（0x3c）先和 P3 口逻辑或一下

//  P3M0 = 0x00; P3M1 = 0x00;       //设置 P3 口为准双向口模式
//  P3M0 = 0x00; P3M1 = 0x3c;       //设置 P32、P33、P34、P35 为高阻输入，其余引脚为准双向口模式
//  P3PU = 0x3c;                    //需要同时开启对应端口的上拉电阻，0011 1100B
    P4M0 = 0x00; P4M1 = 0x00;       //设置 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;       //设置 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;       //设置 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;       //设置 P7 口为准双向口模式

    Delay1ms();                     //延时 1ms，等待 I/O 口电平稳定，因为前面刚改变了模式
        //立刻读可能会读到错误的电平，所以需要先等待电平变化稳定后再读

    usb_init();     //调用 USB 初始化函数，不需要立即判断电脑已正确识别到 USB 从设备
    /* USB 型单片机从设备，如需要主动向电脑发送数据，
       在执行 USB_SendData( )函数和 printf_usb( )函数时，
       这两个函数已增加了判断电脑是否已正确识别到 USB 从设备的程序。
       如果电脑要主动发送数据给 USB 从设备，电脑自己会主动判断与 USB 从设备是否已正确连接。
    */
    EA = 1;                         //总中断允许位打开

    P40 = 0;                        //打开 LED 部分的供电
```

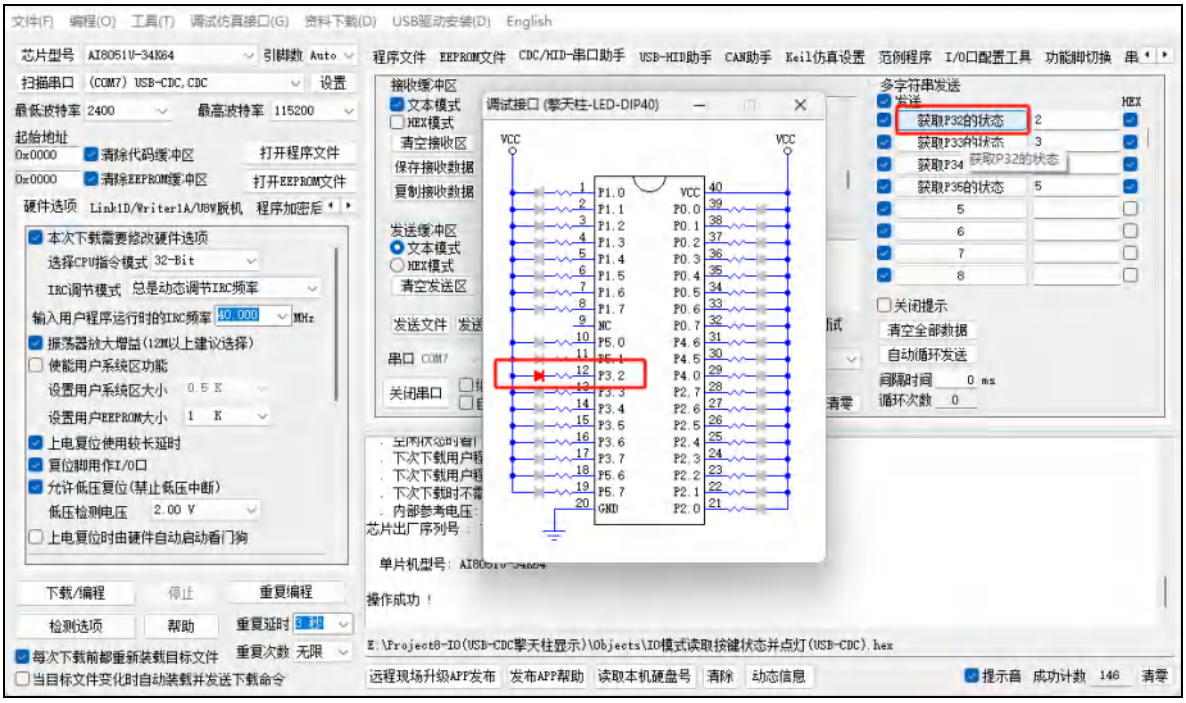
```
    while(1)
    {                                   //使用 USB 串口控制来手动控制 LED 部分
        if(bUsbOutReady)
            //USB 库中有定义 bUsbOutReady 位变量，查询方式必须调用一次“usb_OUT_done();”
        {
            LED40_SetPort(3, P3 | ~ (1<<UsbOutBuffer[0]));
                                    //使用 AIapp-ISP 软件菜单“调试仿真接口”中的
                                    //“擎天柱-LED-DIP40”进行模拟显示 P0 口

//          P0 = P3 | ~(1<<UsbOutBuffer[0]);      //同步端口状态，并将其他不需要的位置 1（不显示）

//下面的 switch 等同于上一句，上一句 C 语言高手才能看懂
            switch (UsbOutBuffer[0])            //判断接收的第一个字节数据
            {
                case 2: P0 = P3 | 0xfb; break;      //如果接收到 2 则将 P3.2 状态送 P0.2，
                                                    //其他不需要的位置 1（不显示），1111 1011B
                case 3: P0 = P3 | 0xf7; break;      //如果接收到 3 则将 P3.3 状态送 P0.3，
                                                    //其他不需要的位置 1（不显示），1111 0111B
                case 4: P0 = P3 | 0xef; break;      //如果接收到 4 则将 P3.4 状态送 P0.4，
                                                    //其他不需要的位置 1（不显示），1110 1111B
                case 5: P0 = P3 | 0xdf; break;      //如果接收到 5 则将 P3.5 状态送 P0.5，
                                                    //其他不需要的位置 1（不显示），1101 1111B
            }
            usb_OUT_done();         //本包接收的数据处理完毕，准备接收下一包数据
                //当使用查询方式 if(bUsbOutReady)，查询调用 USB 库函数时，在数据处理完成后，
                //必须调用一次“usb_OUT_done();”，这样才能接收下一包数据。
        }
    }
}
```

e) 程序下载

程序下载完成以后，就可以看到出现了一个 USB-CDC 端口，此时就可以通过这个 USB-CDC 端口进行数据通讯了。

f) 程序实现效果

程序编写好以后，编译无误后加载到实验箱，就可以通过 AIapp-ISP 软件界面的“调试仿真接口”→“擎天柱-LED-DIP40”虚拟显示对应端口的状态。实现**按下对应按键后，**“擎天柱-LED-DIP40”显示点亮/熄灭了对应的 LED 灯。


调试接口 (擎天柱-LED-DIP40)
VCC
VCC
P1.0
P1.1
P1.2
P1.3
P1.4
P1.5
P1.6
P1.7
NC
P5.0
P3.2
P3.3
P3.4
P3.5
P3.6
P3.7
P5.6
P5.7
GND
P4.6
P4.5
P4.0
P2.7
P2.6
P2.5
P2.4
P2.3
P2.2
P2.1
P2.0
P0.0
P0.1
P0.2
P0.3
P0.4
P0.5
P0.6
P0.7
获取P32的状态
获取P33的状态
获取P34
获取P35的状态
关闭提示
清空全部数据
自动循环发送
操作成功 !

14.14.13　实验 4A--使用【AiCube】项目助手实现“用 USB-CDC 读取按键状态并在【调试接口 | 擎天柱-LED-DIP40】虚拟显示”

STCAI 在最新版的 AIapp-ISP 软件中，推出【AiCube】项目创建助手工具，目前支持 Ai8051U-32Bit 各功能模块的智能化配置，并生成程序框架。这项功能在程序员建立项目时，简化了项目的初始化难度，提高效率，避免因不太熟悉或不完全了解该单片机的性能而忽略个别配置，引起编译错误等问题。尤其对刚涉足单片机的初学者，此工具不仅引导其了解单片机编程流程，还极大的降低了入门门槛，缩短了摸索的时间，提高入门的兴趣和信心。现在我们使用这个工具来生成实现此实验功能的配置和代码。

a) 安装 AIapp-ISP 下载/编程/烧录/软件

安装 STCAI 公司研发的 AIapp-ISP 下载 / 编程 / 烧录 / 软件，含强大的辅助开发工具包。登录 STCAI 官网，下载最新版的 AIapp-ISP 安装包，下载链接如下：https://www.stcai.com/gjrj

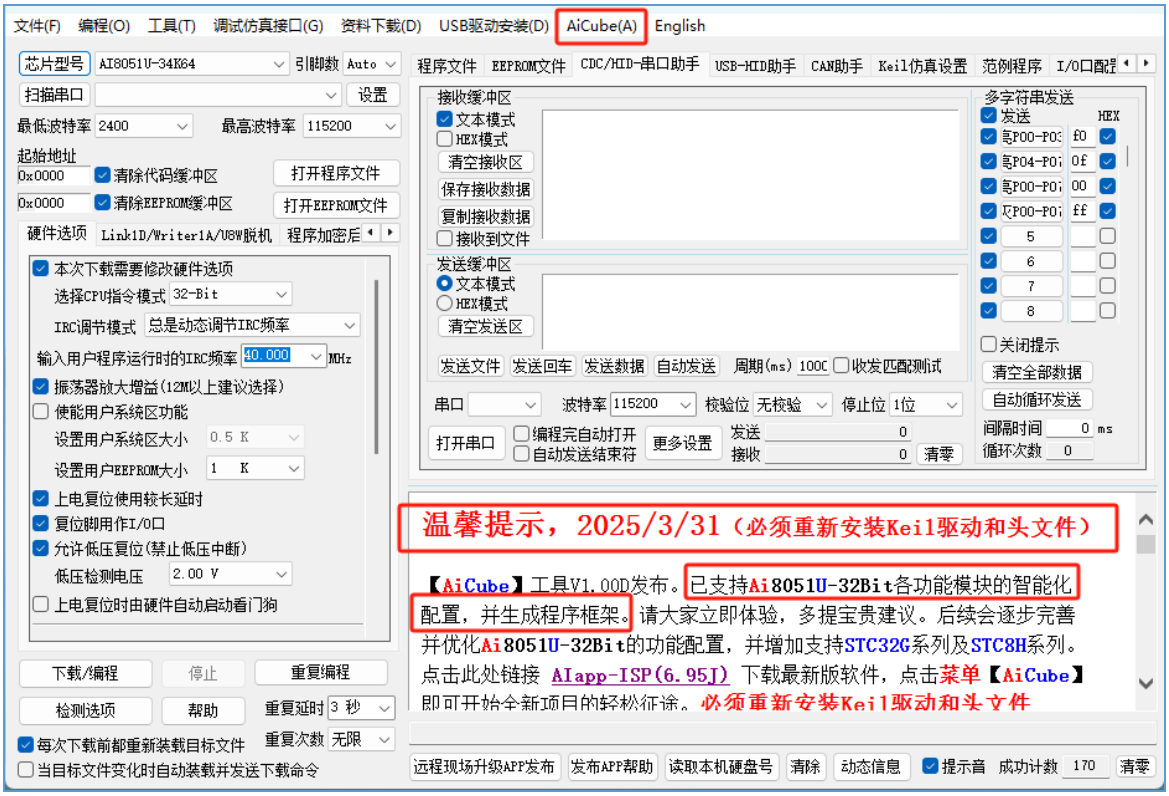
下载下来的是压缩包文件如下图：

双击后解压缩如下图：

如果是解压缩到非桌面的其他目录，如 D:\STCAI-ISP，建议创建快捷方式到桌面方便打开。这实际是一个免安装的绿色软件。

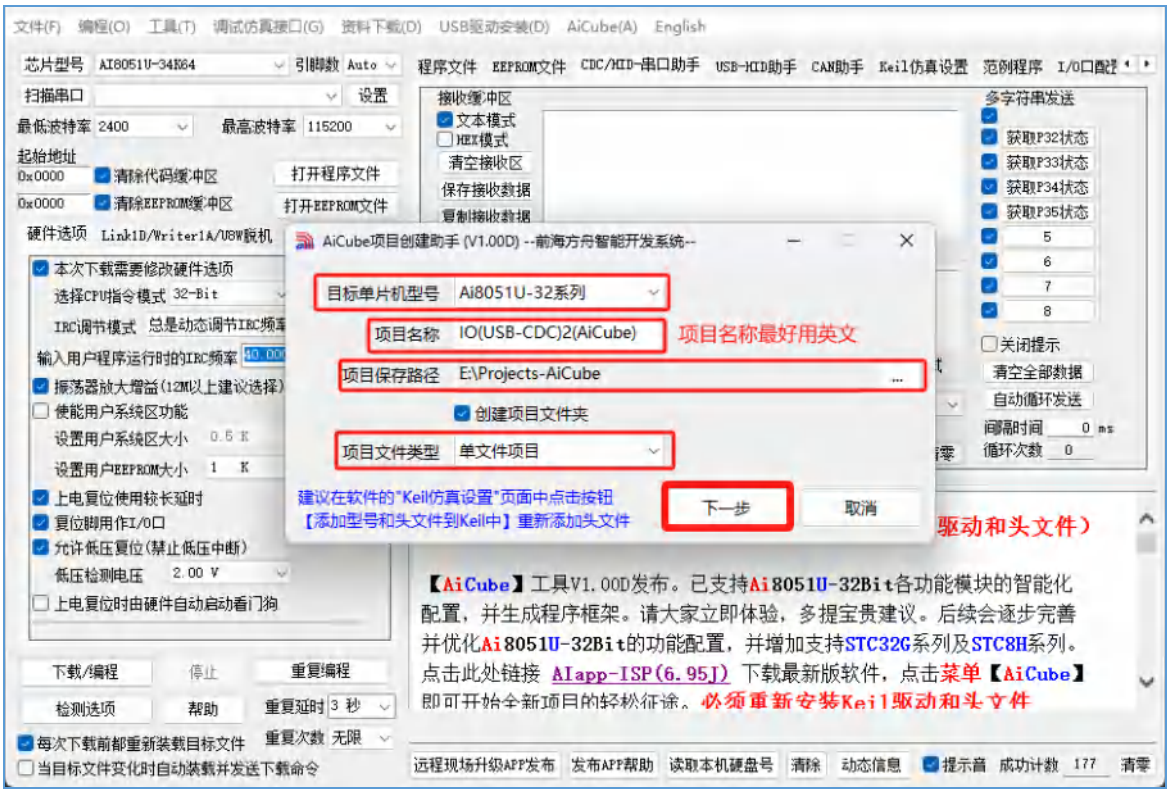
b) 打开【AiCube】助手

双击图标，即可打开 AIapp-ISP 软件界面，即可直接使用，如下图：

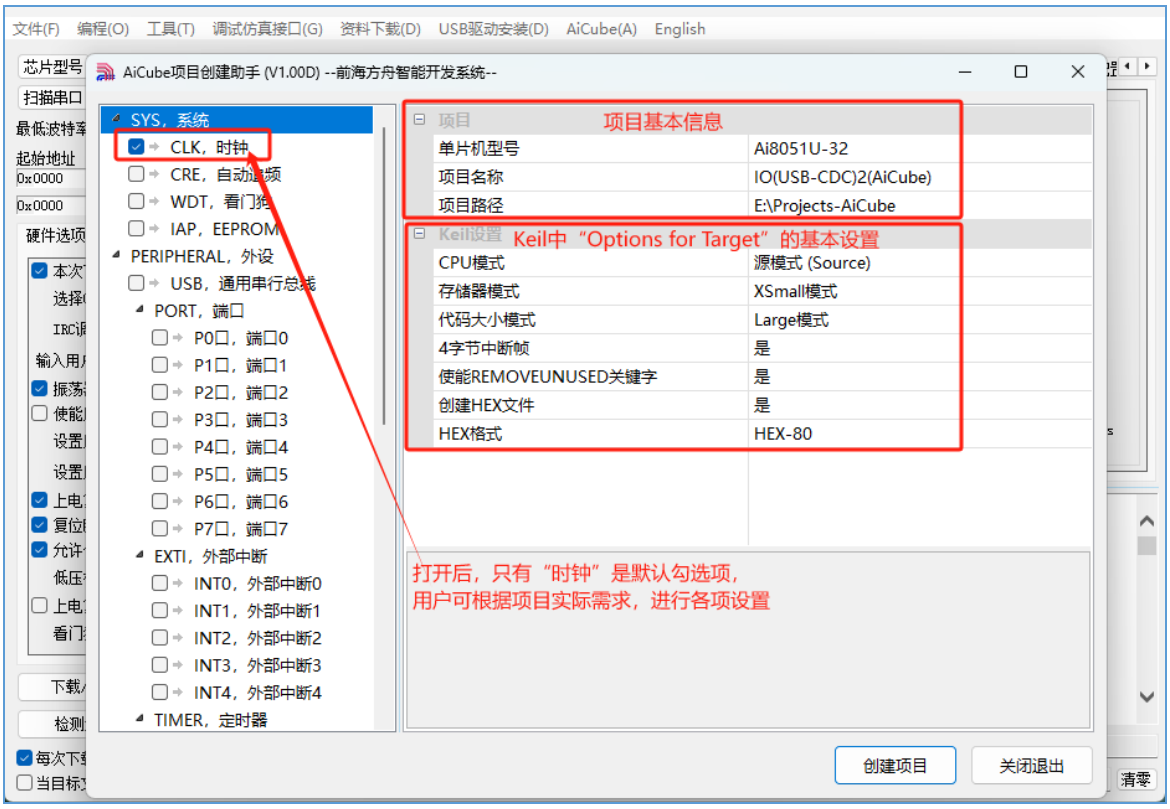


c) 使用【AiCube】助手创建项目

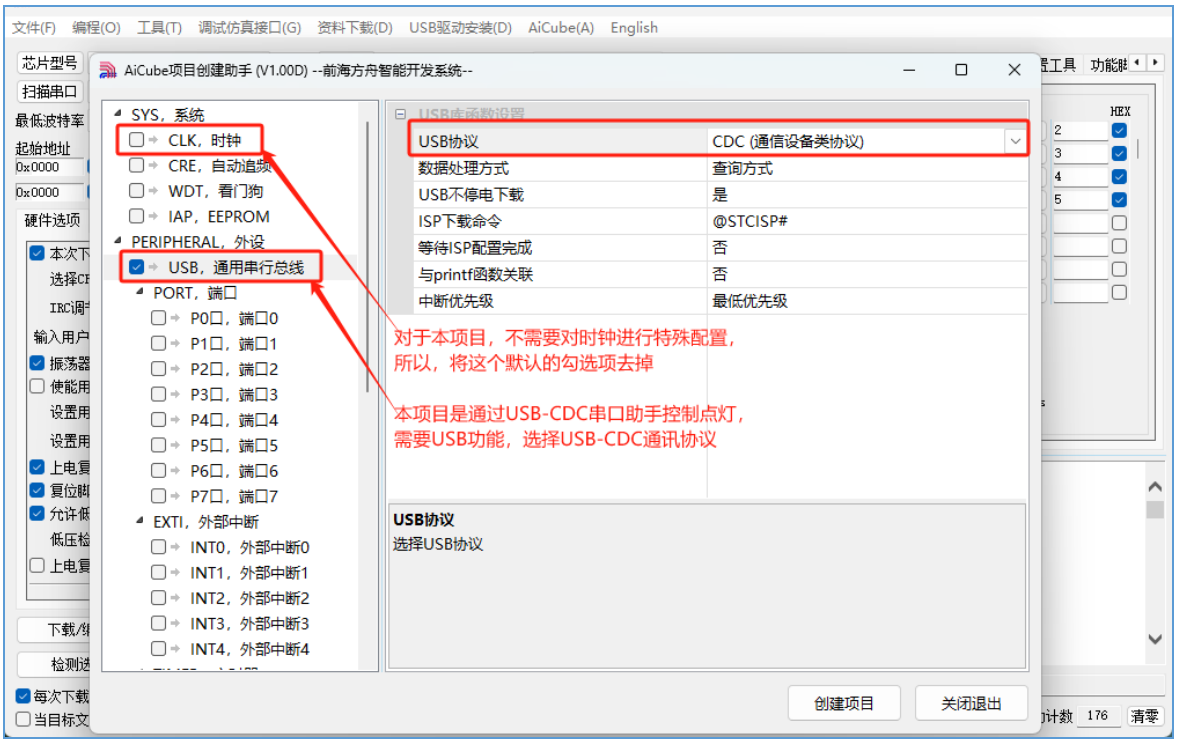
点击【AiCube（A）】，弹出如下窗口：



选择并填写项目基本信息，建议项目名称用英文！点击“下一步”，出现如下配置界面：



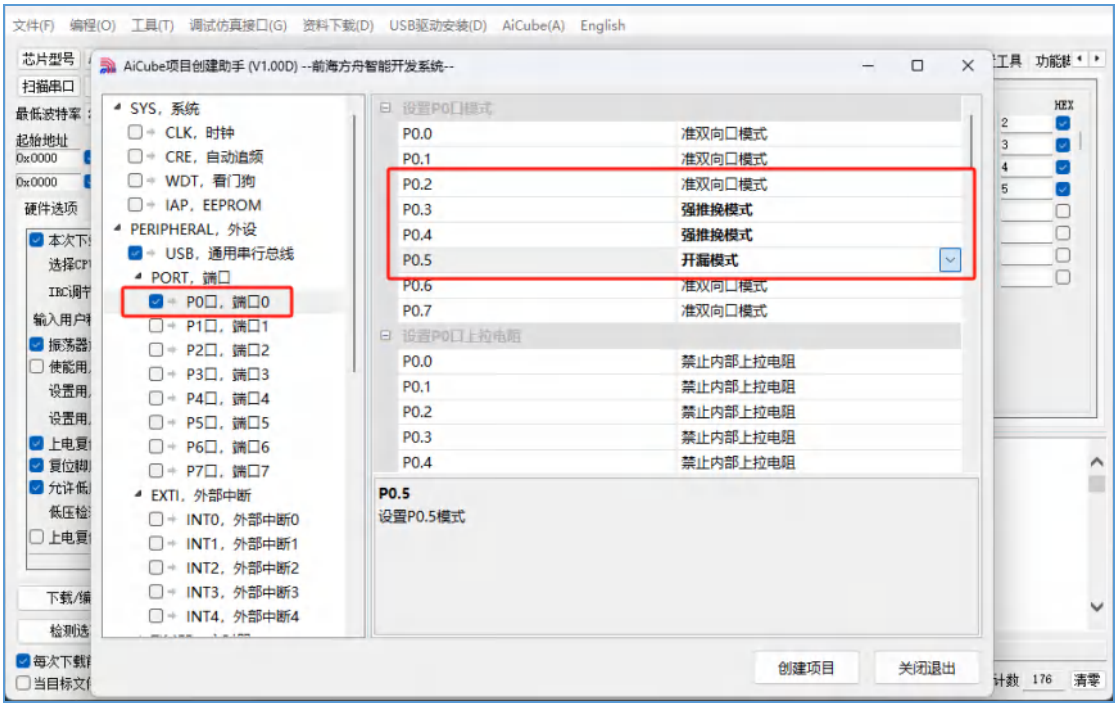
本实验是通过【AIapp-ISP】系统软件中的 USB-CDC 串口助手通知单片机读【P3.2/P3.3/P3.4/P3.5】的状态，送到【调试仿真接口 | 擎天柱-LED-PID40】虚拟显示，同时将相应的状态送到 LED【P0.2/P0.3/P0.4/P0.5】显示（点亮/熄灭）。显然要勾选 USB 选项，选择“CDC（通信设备类协议）”，如下图：



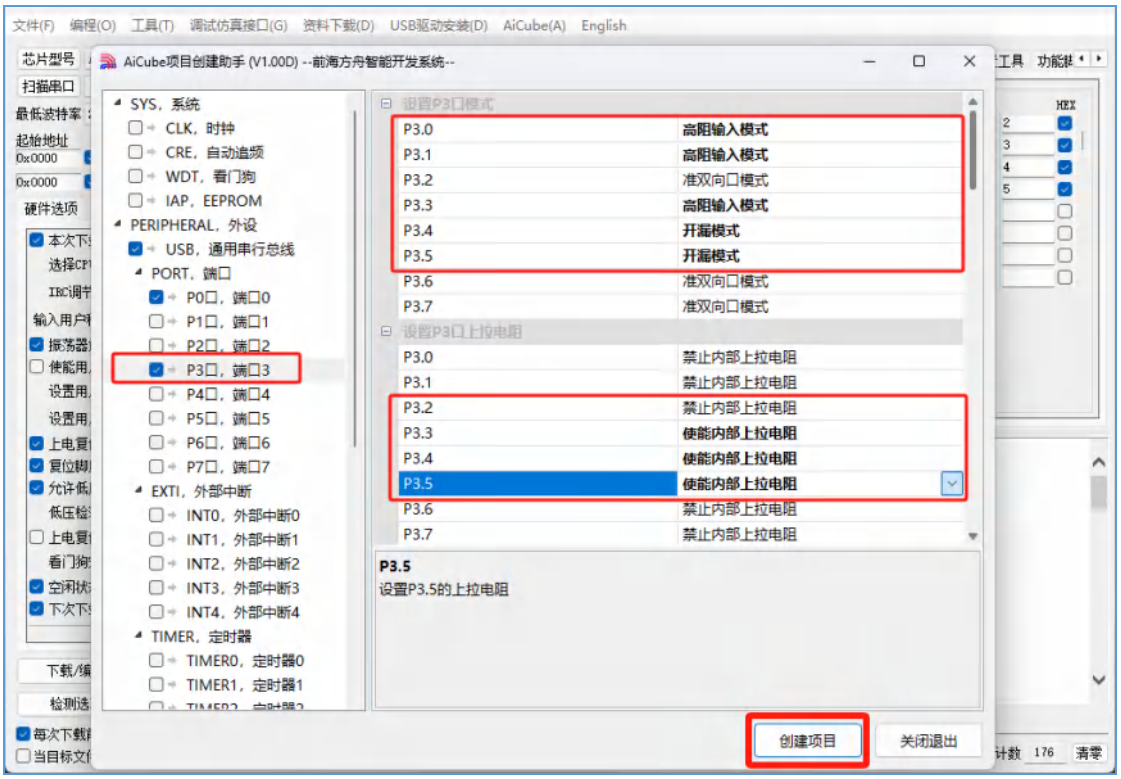
对于本项目，需要单片机读【P3.2 / P3.3 / P3.4 / P3.5】这 4 个 I/O 的状态，
输出到【P0.2 / P0.3 / P0.4 / P0.5】对应的指示灯

针对 I/O 有 4 种不同的模式，我们将【P3.2 / P3.3 / P3.4 / P3.5】不同的 I/O 口设置为不同的模式，验证不同情况下依然可以正常输入/输出。

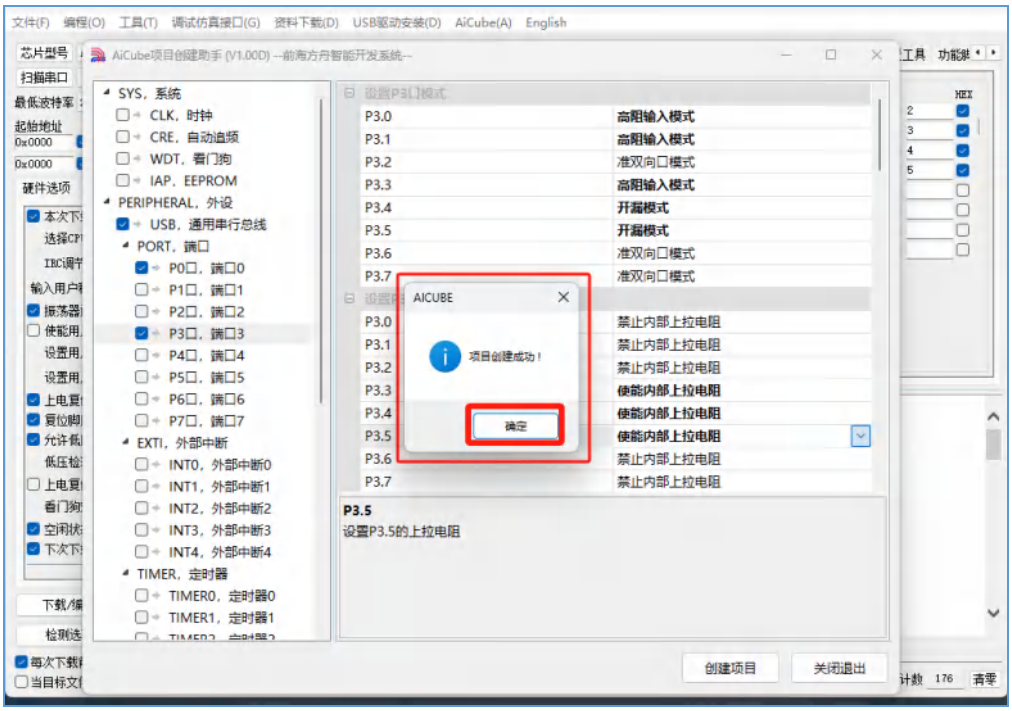
【AiCube】助手中，端口默认是准双向口模式，我们配置输出端口的 P02 保持准双向口模式，P03、P04 为强推挽输出，P05 为开漏输出（可以输出 0 点亮 LED）。如下图：



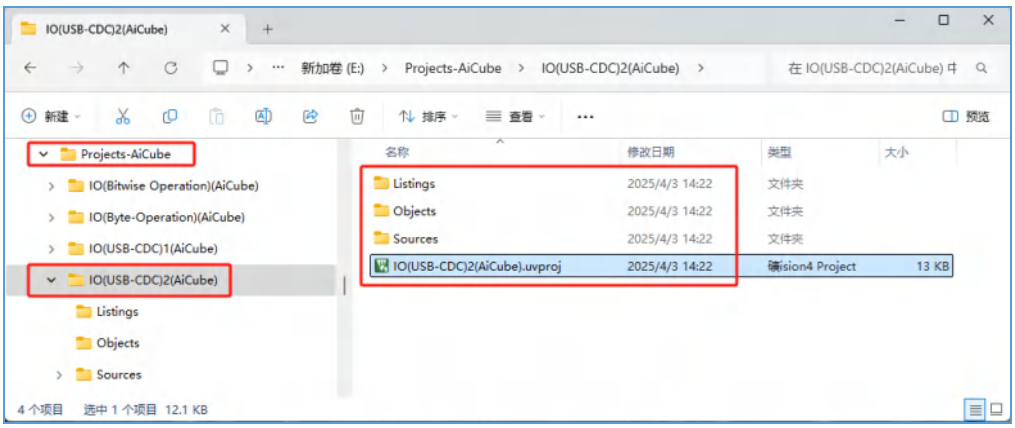
输入配置 P30、P31 为高阻输入(因为跟 USB 复用，建议设置为高阻输入，不干扰 USB 的通讯控制)
设置 P32 为准双向口、P33 为高阻输入、P34 和 P35 为开漏模式（设置为 1 等效高阻输入）
然后开启 P33、P34、P35 的内部上拉电阻以稳定电平状态（不开启内部上拉电阻读外部状态，高阻输入/开漏模式，外部都需要有上拉电阻配合）
【AiCube】助手中，端口默认是准双向口模式，内部上拉电阻默认是禁用设置。现在我们按上述要求配置 P30、P31 为高阻输入；P32 为保持准双向口；P33 为高阻输入；P34、P35 为开漏模式；打开 P33、P34、P35 的内部上拉电阻。如下图：



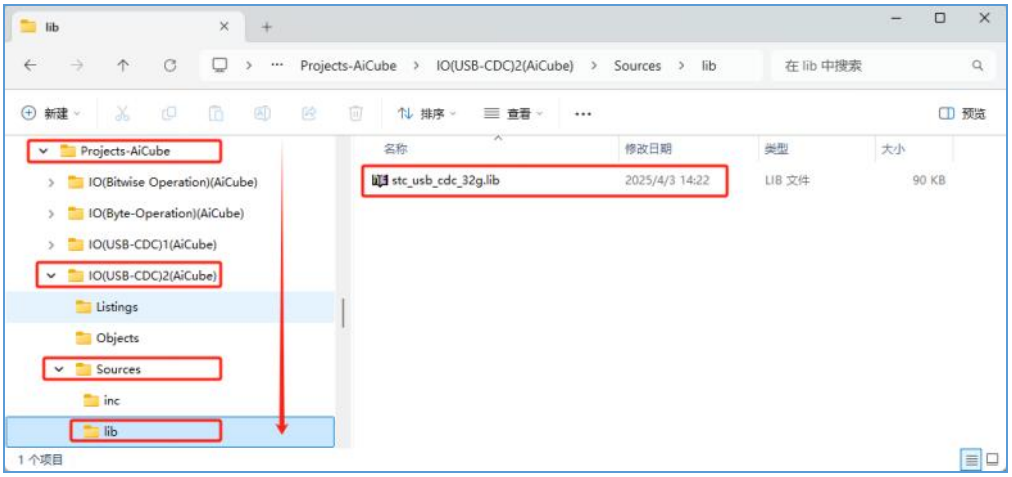
点击“创建项目”，【AiCube】助手提示：“项目创建成功！”，如下图：



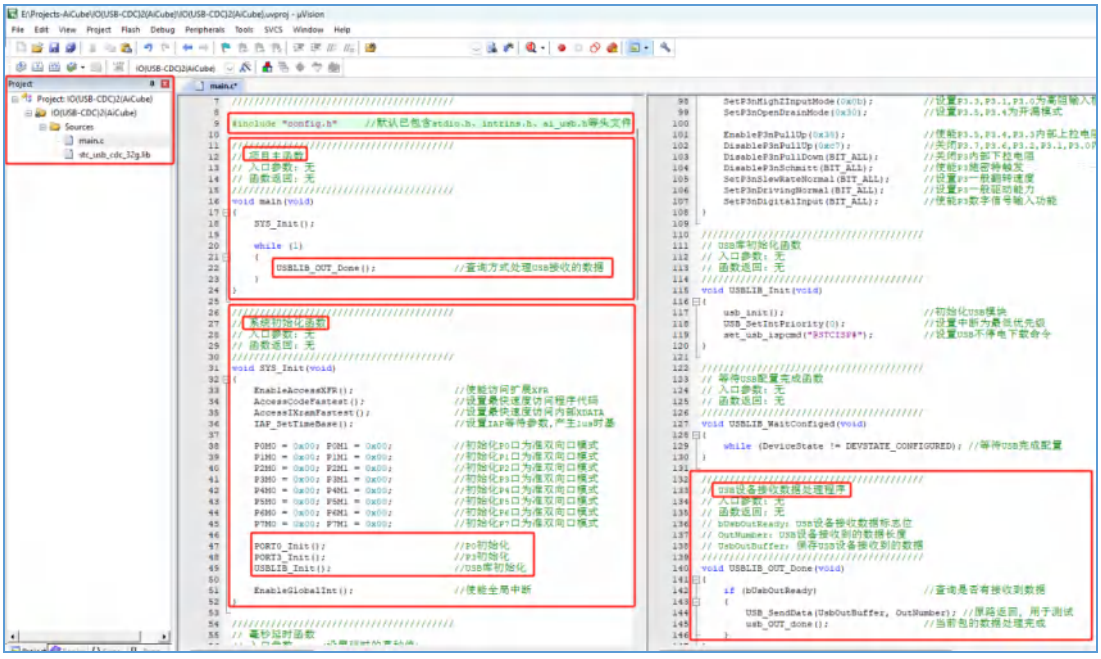
点击“确定”，即成功创建了项目及项目文件目录“IO(USB-CDC)2(AiCube)”，找到并打开刚刚创建的项目目录。发现刚刚的操作过程，【AiCube】助手已经帮我们建立了项目目录及 IO(USB-CDC)2(AiCube).uvproj 文件，如下图：



因为勾选了 USB 选项，打开 Sources>lib 子目录，发现，【AiCube】助手连 USB 库函数也已经加载了：



双击打开项目文件：IO(USB-CDC)2(AiCube).uvproj，如下图（为方便查看，使用菜单“Window”→“Split”将原窗口拆分成两个窗口，方便观察）：

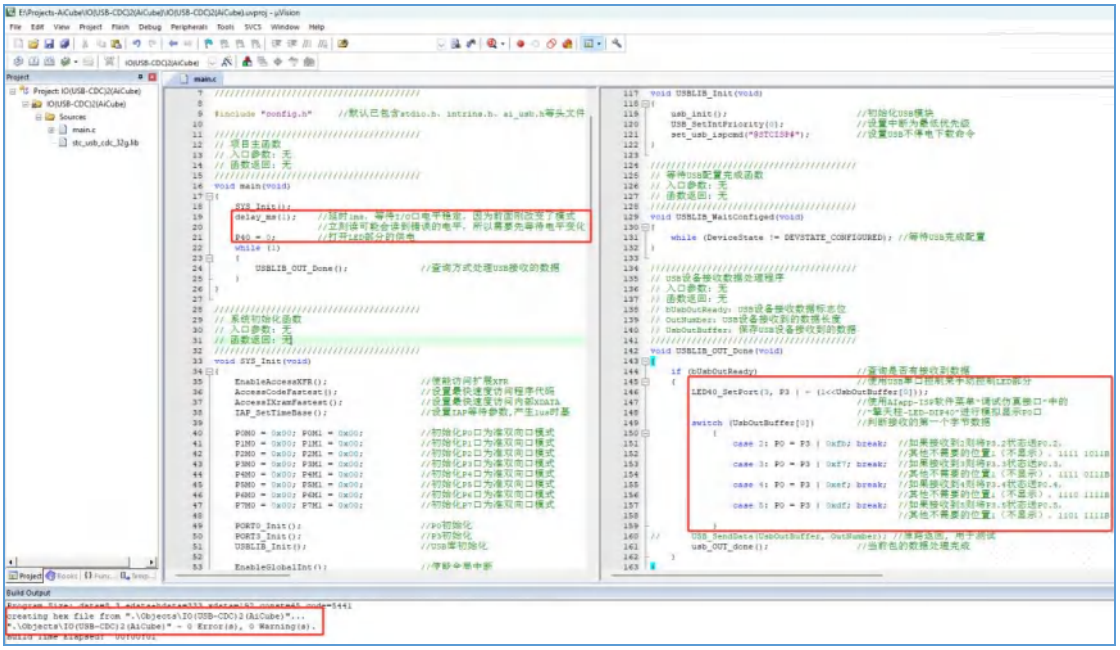


根据在【AiCube】助手中的配置，从上图中我们看到【AiCube】助手创建的本实验项目中有了 main.c，还已经自动加载了 USB 库函数（stc_usb_cdc_32g.lib）。main.c 中按模块化设计，已经包含了：必要的头文件、主函数、系统初始化函数、延时函数、P0 口初始化函数、P3 口初始化函数、USB 库初始化函数、等待 USB 配置完成函数、USB 设备接收数据处理程序。

d) 增加实验项目代码

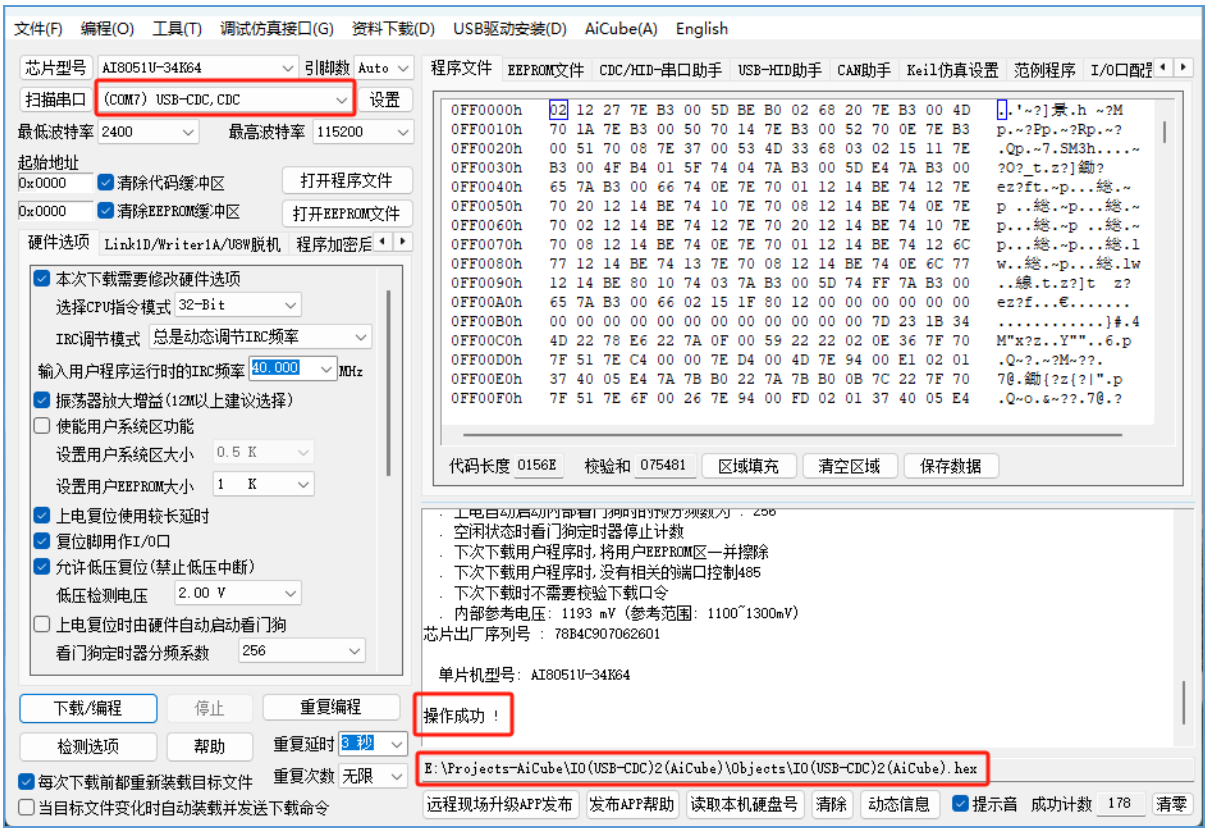
为实现通过【AIapp-ISP】系统软件中的 USB-CDC 串口助手通知单片机读【P3.2/P3.3/P3.4/P3.5】的状态，送到【调试仿真接口 | 擎天柱-LED-PID40】虚拟显示，同时将相应的状态送到 LED【P0.2/P0.3/P0.4/P0.5】显示（点亮/熄灭）。我们现在需要：

- 在主函数中，调用延时函数“delay_ms(1);”，延时 1ms（等待 I/O 口电平稳定）；
- 在主函数中，加入“P40 = 0;”，打开 LED 供电的代码；
- 在“USB 设备接收数据处理程序”子函数中，补充 USB 控制代码；
- 保存，编译，如下图（为方便查看，使用菜单“Window”→“Split”将原窗口拆分成两个窗口，对比观察）：

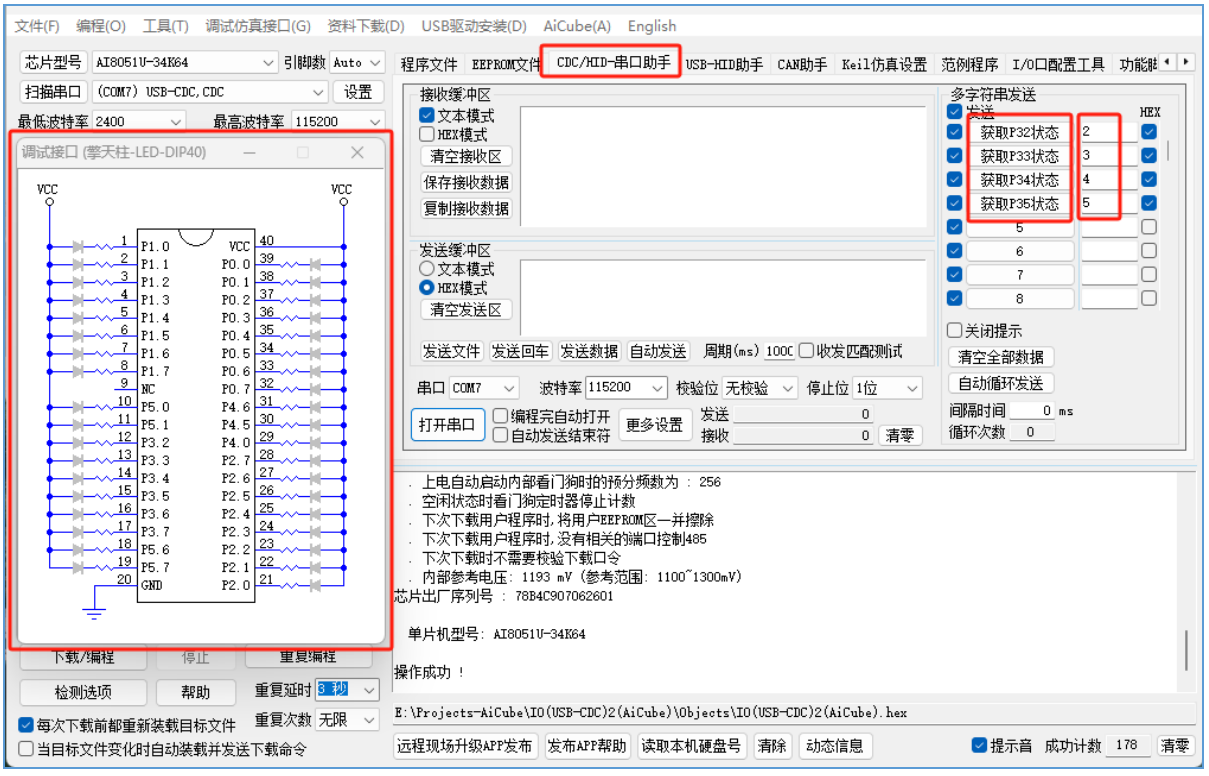


e) 使用 AIapp-ISP 软件系统并连接 Ai8051U 实验箱验证实验效果

连接好 Ai8051U 实验箱，运行 AIapp-ISP 软件系统，按步骤打开并下载程序“IO(USB-CDC)2(AiCube).hex”到实验箱，如下图：



“CDC/HID-串口助手”选项卡中，在“多字符串发送”窗口，设置好发送控制命令，同时打开 AIapp-ISP 软件的“调试仿真接口”菜单→选择“擎天柱-LED-DIP40”以便观察更直观的仿真效果：

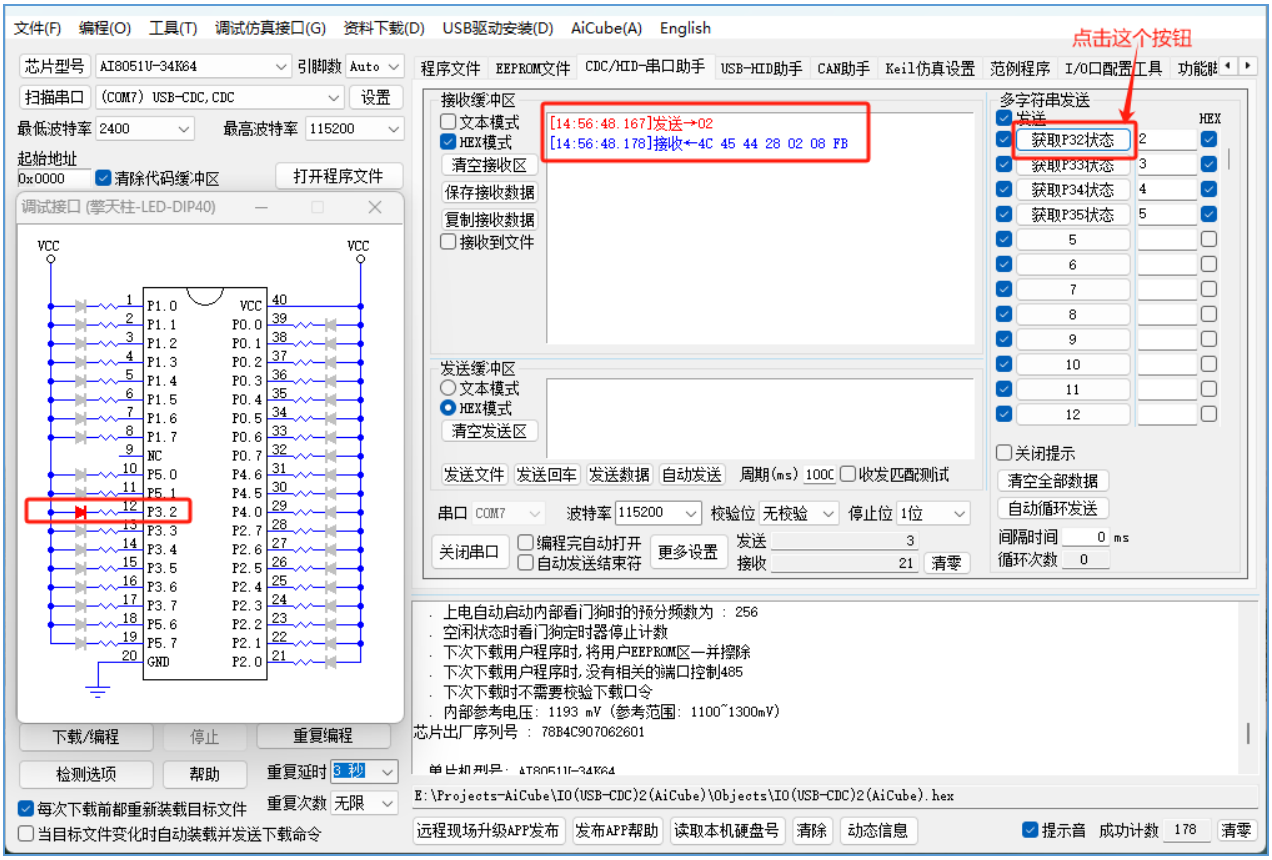


一切准备就绪，

按住 Ai8051U 实验箱上的 P3.2 按键，点击“获取 P32 状态”按钮，控制实验箱上的 P02 灯亮，调试接口（擎天柱-LED-DIP40）的 P3.2 端口的 LED 亮；

松开 P3.2 按键，P02 灯依然是亮的，再点击“获取 P32 状态”按钮，P02 灯灭，调试接口（擎天柱-LED-DIP40）的 P3.2 端口的 LED 灭。

在 USB-CDC 接收窗口就能看到相应的接收内容，调试接口（擎天柱-LED-DIP40）显示如下图：



按住 Ai8051U 实验箱上的 P3.3 按键，点击“获取 P33 状态”按钮，控制实验箱上的 P03 灯亮了，调试接口（擎天柱-LED-DIP40）的 P3.3 端口的 LED 亮；

松开 P3.3 按键，P03 灯依然是亮的，再点击“获取 P33 状态”按钮，P03 灯灭，调试接口（擎天柱-LED-DIP40）的 P3.3 端口的 LED 灭。

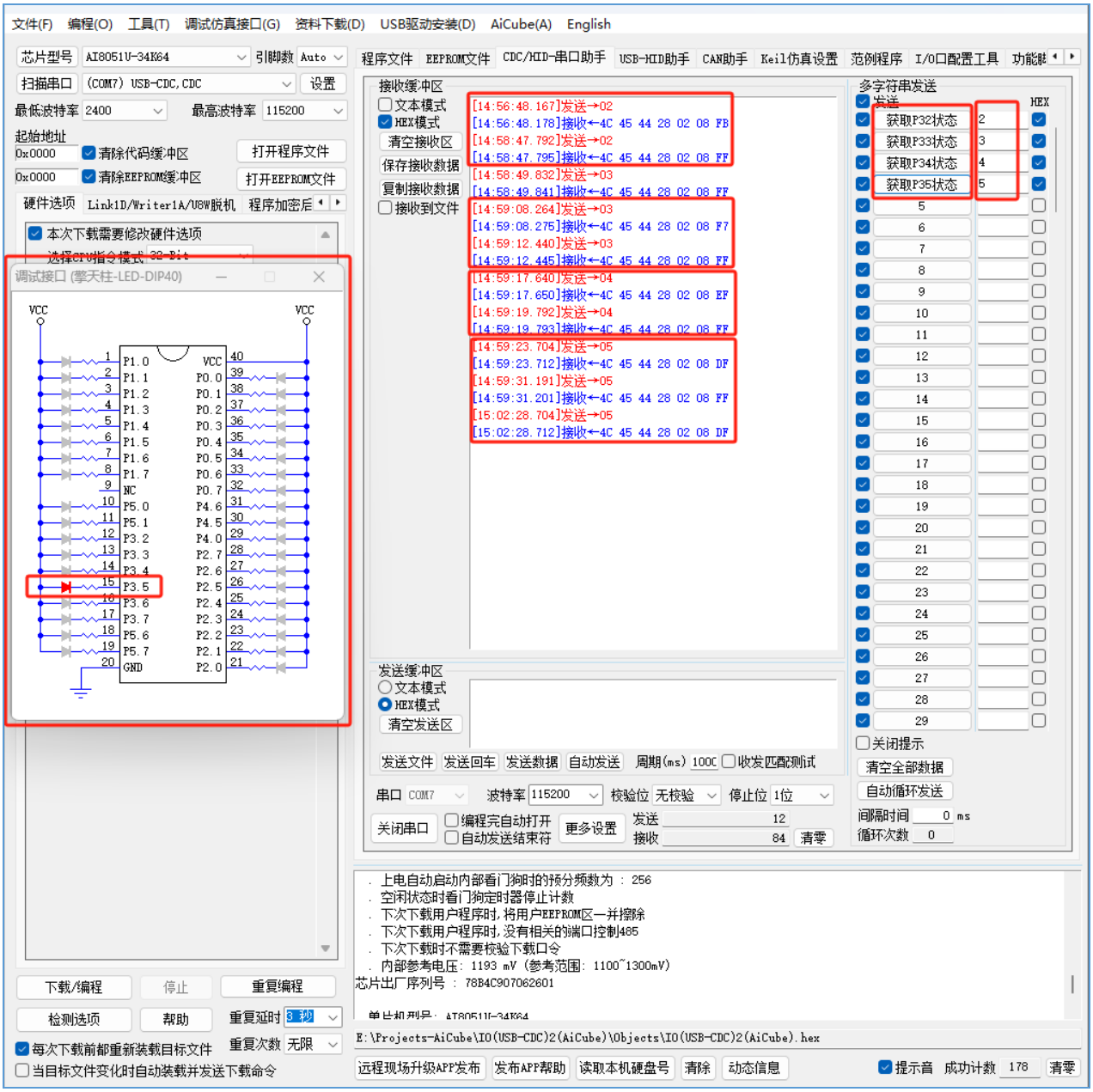
按住 Ai8051U 实验箱上的 P3.4 按键，点击“获取 P34 状态”按钮，控制实验箱上的 P04 灯亮了，调试接口（擎天柱-LED-DIP40）的 P3.4 端口的 LED 亮；

松开 P3.4 按键，P04 灯依然是亮的，再点击“获取 P34 状态”按钮，P04 灯灭，调试接口（擎天柱-LED-DIP40）的 P3.4 端口的 LED 灭。

按住 Ai8051U 实验箱上的 P3.5 按键，点击“获取 P35 状态”按钮，控制实验箱上的 P05 灯亮了，调试接口（擎天柱-LED-DIP40）的 P3.5 端口的 LED 亮；

松开 P3.5 按键，P05 灯依然是亮的，再点击“获取 P35 状态”按钮，P05 灯灭，调试接口（擎天柱-LED-DIP40）的 P3.5 端口的 LED 灭。

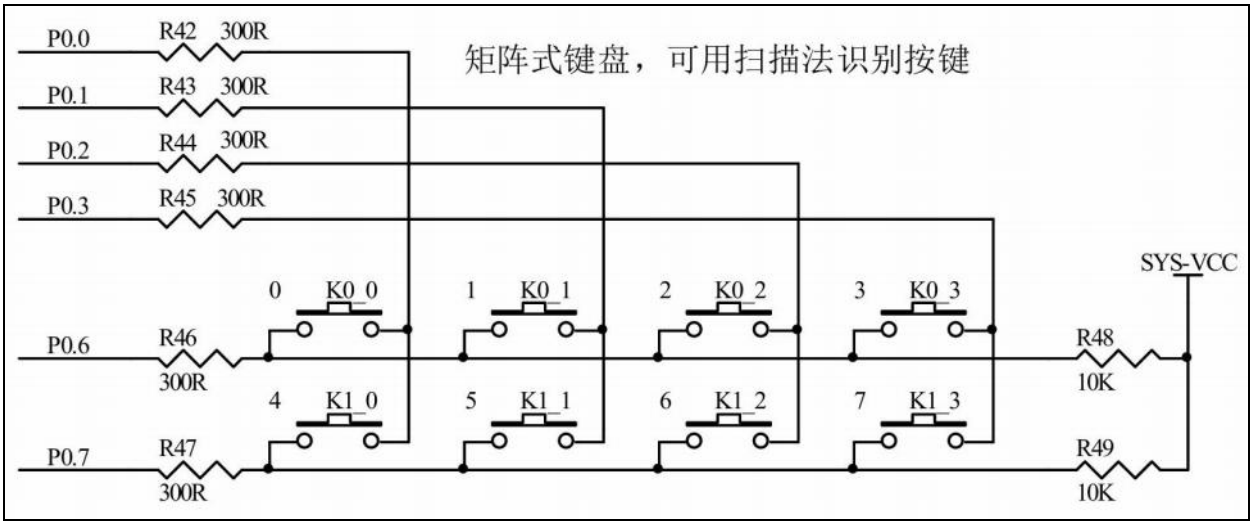
在 USB-CDC 接收窗口就能看到相应的接收内容，如下图：

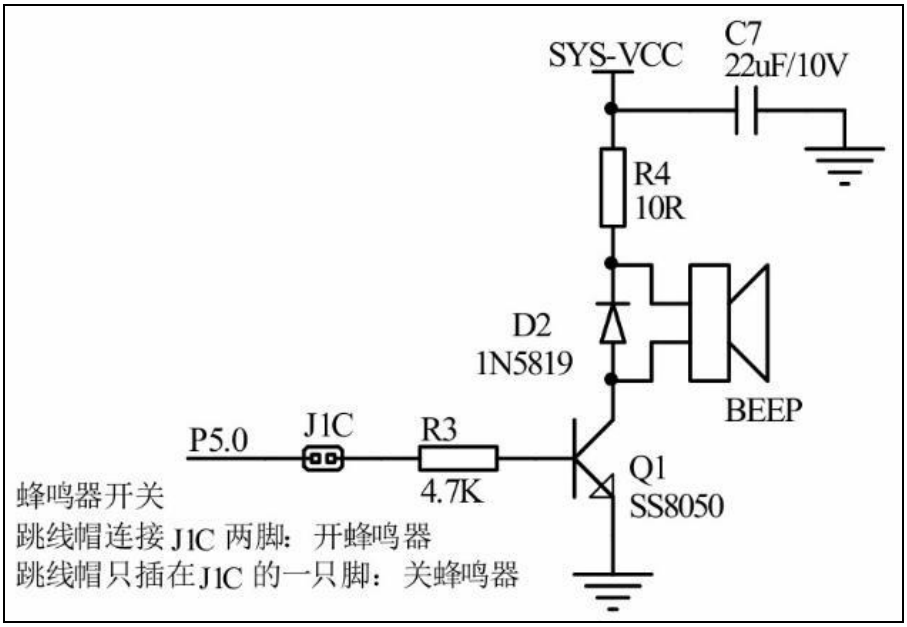


有了【AiCube】助手，单片机编程就是这么简单！快用起来吧！

14.14.14　实验 5：I/O 口行列按键扫描，蜂鸣器发声大型实战

a) 原理图





b) 实验介绍

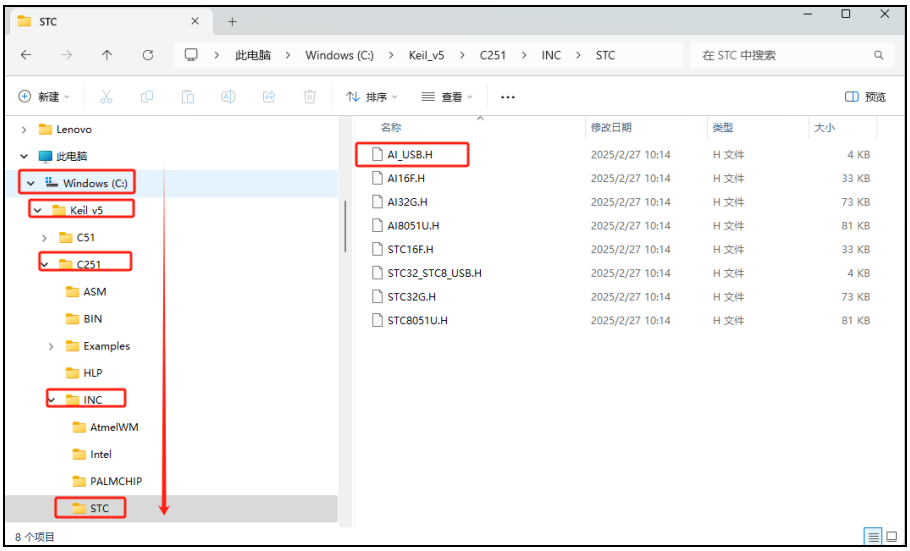
按键扫描说明：
如 1 个按键要判断为正常按下，而不是抖动，
假定需要该按键保持按下持续保持为低状态时间是 50mS ~ 200mS；
假定 1 个主循环的运行时间大概 10mS ~ 30mS
则如发现某个按键按下，需要连续 6 次主循环都能读到该按键按下
===如发现多个按键按下，则归 0，重新开始
调整按键的灵敏度只需要修改，
===【判断按键已按下的时间，即读到为低经历了多少次主循环的次数】

c) 准备工作--添加头文件和库函数

1、添加常用头文件 ai_usb.h

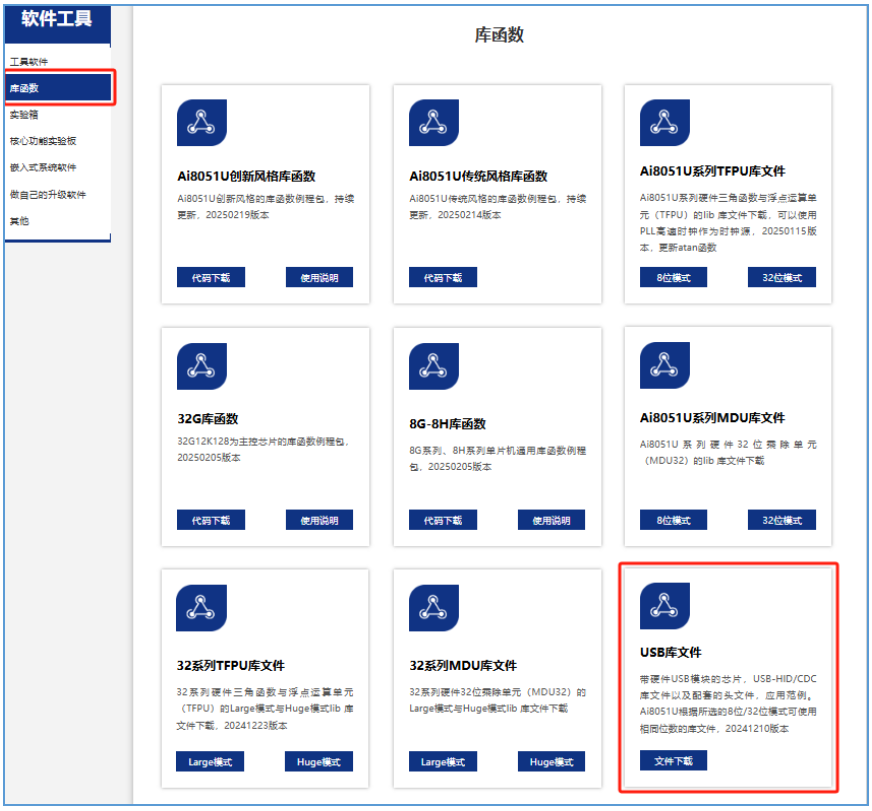
#include "ai_usb.h"　　//使用 usb 所需要包含的头文件。

这个头文件在使用 AIapp-ISP 软件添加型号和头文件到 Keil 中时就已经一并被添加到 Keil 中了。



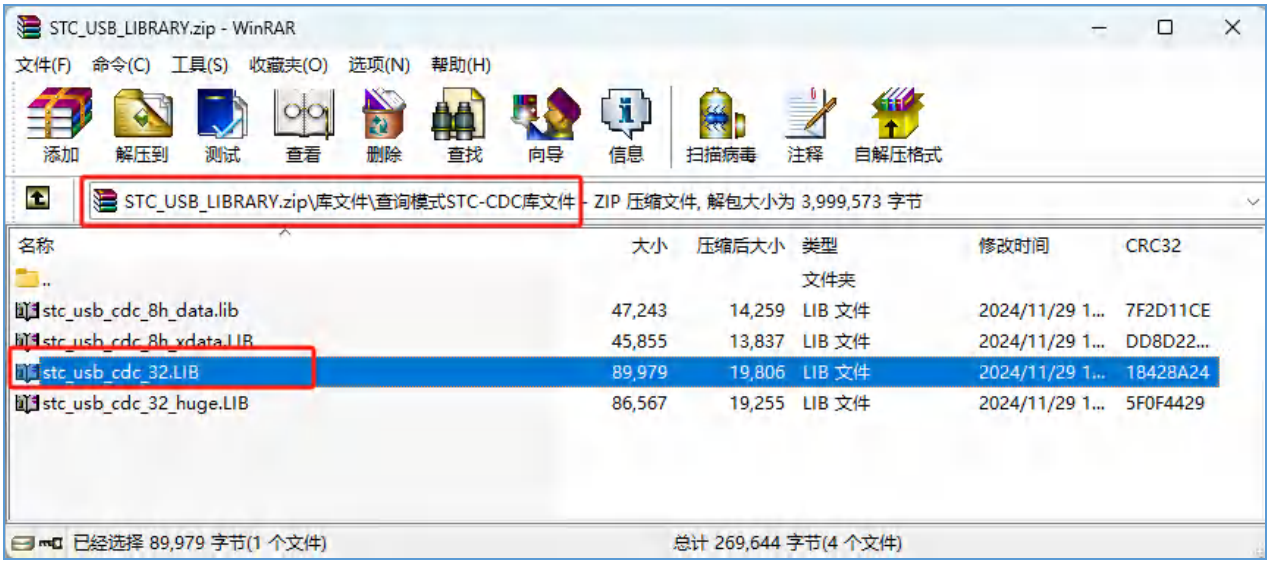
2、添加 usb_cdc 库函数文件

这部分可以到 STCAI 的官网：https://www.stcai.com/khs 的 USB 库文件中进行下载

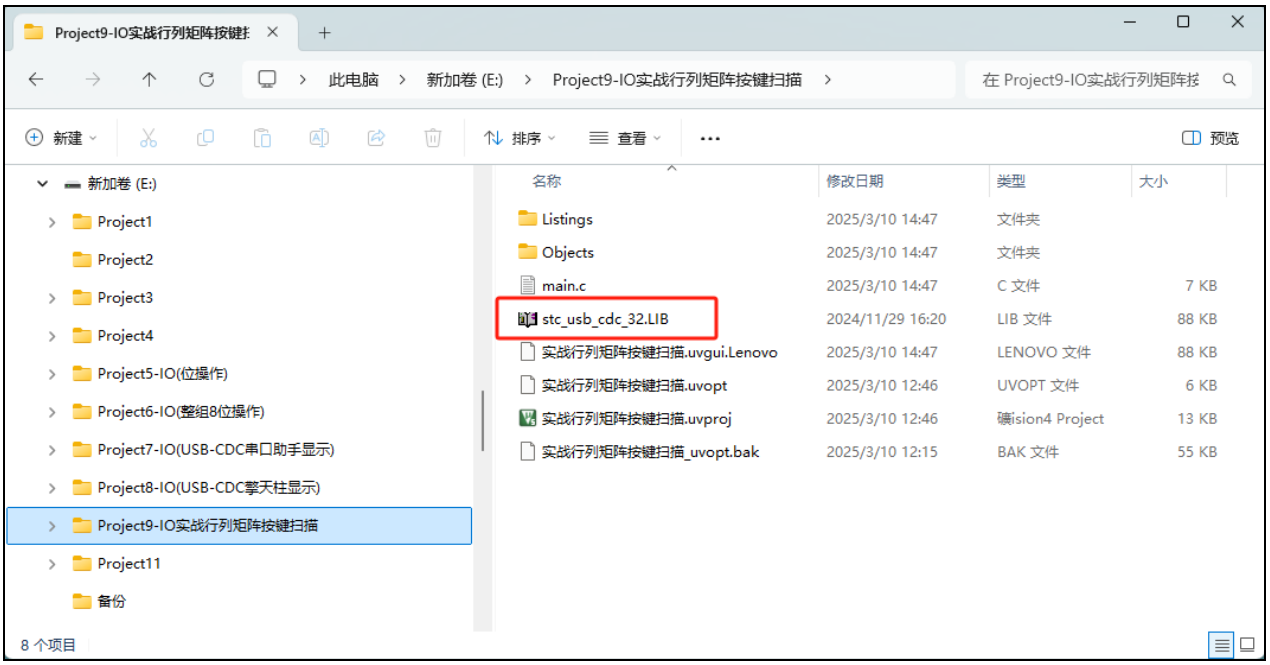


下载完成后，打开压缩包“STC_USB_LIBRARY.zip”，

在’库文件\查询模式 STC-CDC 库文件’中，复制出来 stc_usb_cdc_32.LIB

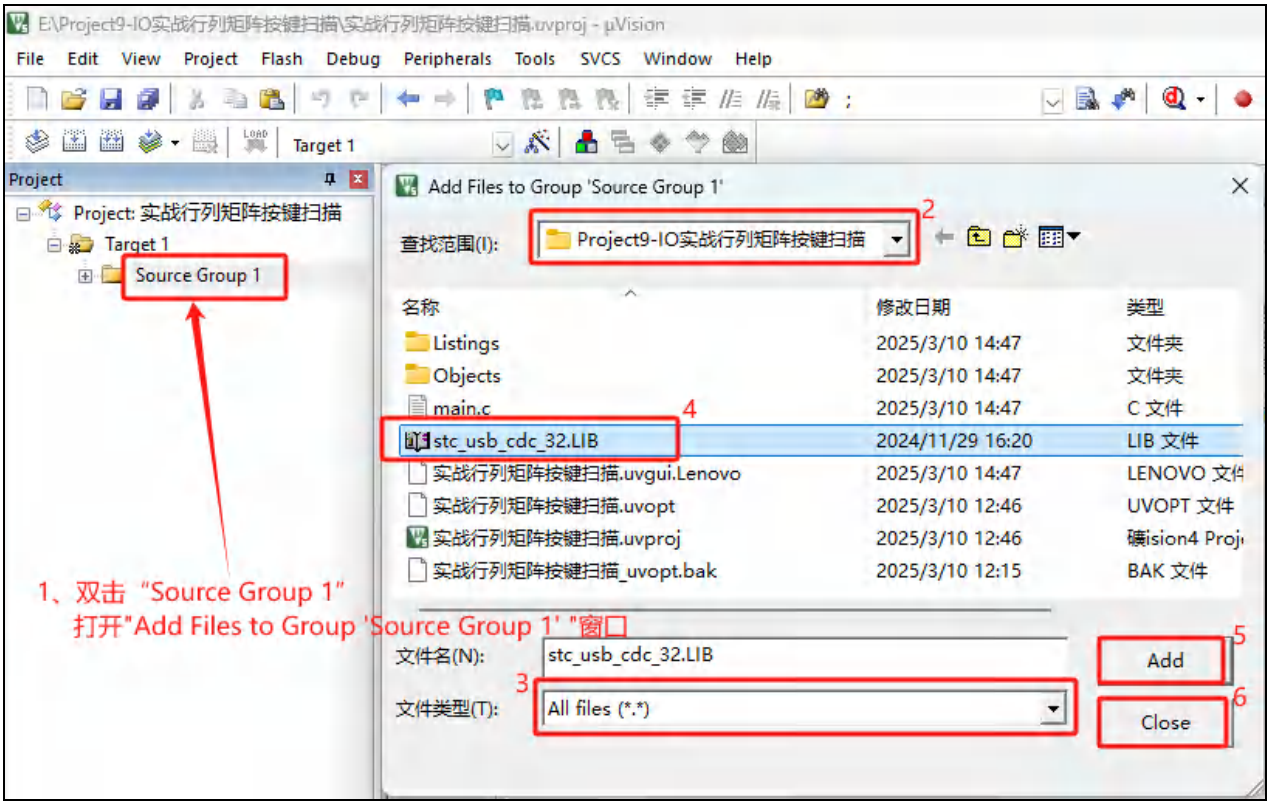


3、将库函数“stc_usb_cdc_32.LIB”复制到创建工程所在的文件夹 project9 中

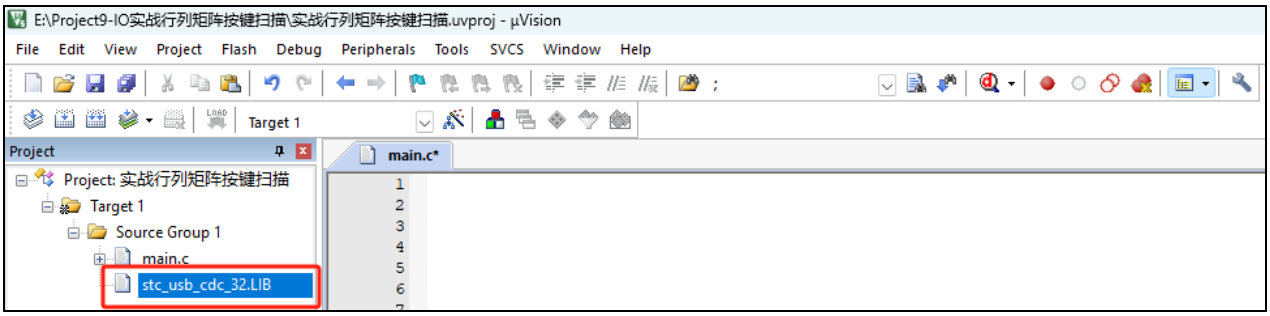


4、将库函数 stc_usb_cdc_32.LIB 添加进入此工程项目中

在 Keil 中，双击 Source Group，在弹出的 Add Files to Group 'Source Group 1 窗口中，将库函数 stc_usb_cdc_32.LIB 添加进入此工程，就可以使用 USB-CDC 功能进行端口通讯。



这样，Keil 界面变化如下图：



【注意】：USB-CDC 串口则更类似传统串口的形式，使用 COM 口进行通讯，但是只有 Win10 及其以上才默认携带 USB-CDC 的驱动，较低的系统版本可能要手动安装驱动才能使用。

d) 实验程序代码

```
#include <AI8051U.H>                    //包含 AI8051U 的头文件
#include "ai_usb.h"                     //使用 usb 所需要包含的头文件
/*
如何获得上面这 2 个头文件，如已按照前面介绍的方法：
从 www.STCAI.com 网站，【软件工具 | 工具软件】处，下载安装最新的【AIapp-ISP 软件】，
选择其中的选项卡 【Keil 仿真设置 】，
点击【添加型号和头文件到 Keil 中，并添加仿真器驱动到 Keil 中】按钮，
则 Ai8051U.h 和 AI_usb.h 都会自动复制到 Keil 中，包含进来就可以直接使用，
大家可以打开看下其中的内容，AI_usb.h 和 stc32_stc8_usb.h 是相似的，只需用其中的 1 个头文件，
具体见 Keil_v5\C251\INC\STC 目录，或 Keil_v5\C51\INC\STC 目录。
```

如将对应的 USB 库文件添加到 项目中，后续用户程序中，USB 如收到命令字符串 "@STCISP#" USB 库函数就会自动软复位到系统 ISP 程序区，实现 USB 不停电下载，具体见相应部分的说明。

```
//char *USER_DEVICEDESC = NULL;
//char *USER_PRODUCTDESC = NULL;
//char *USER_STCISPCMD = "@STCISP#";
//使用 20250319 之前版本的库函数时需要在用户代码中增加上面 3 行定义。
//使用 20250319 及以后的版本都不再需要上面的定义
*/

#include "intrins.h"                    //使用_nop_()函数所必须要包含的头文件,
                                        //否则延时函数中调用的_nop_()函数没有被头文件引用过来,
                                        //会导致编译器找不到这个而函数而报错。

bit    keyok;                           //按键按下标志
unsigned char keycode;                  //定义变量,保存按键键码
unsigned char debounce;                 //按键去抖动

void Delay1ms(void)                     //@40.000MHz，延时 1ms
{
    unsigned long edatai;

    _nop_();
    _nop_();
    _nop_();
    i =    9998UL;
    while (i) i--;
}

void DelayXms(int n)                    //延时函数，延时 x(ms)
{
    while (n--)
        Delay1ms();
}

void DelayXus(int n)                    //@40.000MHz,延时 x(us)
{
    unsigned long edatai;

    while (n--)
    {
        i =    8UL;
        while (i) i--;
    }
}

void BootLogo(void)                     //开机画面，LED00-LED07 闪烁 3 次
```

```
{
    P0 = 0x00;                  //8 个 LED 全部点亮
    DelayXms(200);              //延时 200ms
    P0 = 0xff;                  //8 个 LED 全部熄灭
    DelayXms(200);              //延时 200ms
    P0 = 0x00;                  //8 个 LED 全部点亮
    DelayXms(200);              //延时 200ms
    P0 = 0xff;                  //8 个 LED 全部熄灭
    DelayXms(200);              //延时 200ms
    P0 = 0x00;                  //8 个 LED 全部点亮
    DelayXms(200);              //延时 200ms
    P0 = 0xff;                  //8 个 LED 全部熄灭
    DelayXms(200);              //延时 200ms
}

void DetectTemp(void)           //检测当前温度
{
}

void ControlTemp(void)          //控制当前温度
{
}

void OtherEvent(void)           //其他事件
{
    DelayXms(10);               //假定其他事件处理时间为 10ms
}

void Beep(int tone)             //蜂鸣器程序
{                               //实验箱上的是无源蜂鸣器,需要持续输出高低电平蜂鸣器才会发声
    int   i;

    for   (i = 0;     i <   50;   i++)
    {
        P50  = 1;               //打开蜂鸣器控制三极管
        DelayXus(tone);         //延时(修改延时时间可改变蜂鸣器的音调)
        P50  = 0;               //关闭蜂鸣器控制三极管
        DelayXus(tone);         //延时
    }
}

void KeyEvents(void)            //执行有效按键对应的任务
{
    if (keyok)                  //判断按键按下标志
    {
        keyok =   0;            //清除标志
```

```
        switch (keycode)            //处理不同的按键键码
        {
        case 0:    Beep(956); break;   //Key0 事件,使用蜂鸣器发 Do
                                       //这里的参数为 956，即产生一个低电平和高电平均为 956us 的方波
                                       //得到的方波频率为 1000000/(956+956)=523Hz
                                       //即为高 8 度的 Do 的频率（261.6Hz*2）
        case 1:    Beep(851); break;   //Key1 事件,使用蜂鸣器发 Re
                                       //这里的参数为 851，即产生一个低电平和高电平均为 851us 的方波
                                       //得到的方波频率为 1000000/(851+851)=587Hz
                                       //即为高 8 度的 Re 的频率（293.6Hz*2）
        case 2:    Beep(758); break;   //Key2 事件,使用蜂鸣器发 Mi
                                       //这里的参数为 758，即产生一个低电平和高电平均为 758us 的方波
                                       //得到的方波频率为 1000000/(758+758)=660Hz
                                       //即为高 8 度的 Mi 的频率（329.6Hz*2）
        case 3:    Beep(716); break;   //Key3 事件,使用蜂鸣器发 Fa
                                       //这里的参数为 716，即产生一个低电平和高电平均为 716us 的方波
                                       //得到的方波频率为 1000000/(716+716)=698Hz
                                       //即为高 8 度的 Fa 的频率（349.2Hz*2）
        case 4:    Beep(638); break;   //Key4 事件,使用蜂鸣器发 So
                                       //这里的参数为 638，即产生一个低电平和高电平均为 638us 的方波
                                       //得到的方波频率为 1000000/(638+638)=784Hz
                                       //即为高 8 度的 So 的频率（392Hz*2）
        case 5:    Beep(568); break;   //Key5 事件,使用蜂鸣器发 La
                                       //这里的参数为 568，即产生一个低电平和高电平均为 568us 的方波
                                       //得到的方波频率为 1000000/(568+568)=880Hz
                                       //即为高 8 度的 La 的频率（440Hz*2）
        case 6:    Beep(506); break;   //Key6 事件,使用蜂鸣器发 Si
                                       //这里的参数为 506，即产生一个低电平和高电平均为 506us 的方波
                                       //得到的方波频率为 1000000/(506+506)=988Hz
                                       //即为高 8 度的 Si 的频率（493.8Hz*2）
        case 7:    Beep(478); break;   //Key7 事件,使用蜂鸣器发 Do
                                       //这里的参数为 478，即产生一个低电平和高电平均为 478us 的方波
                                       //得到的方波频率为 1000000/(478+478)=1046Hz
                                       //即为高两个 8 度的 Do 的频率（261.6Hz*2*2）
        }
    }
}

void ScanKey(void)                  //检测按键状态
{
    unsigned char key;              //按键扫描键码

    key  = 0xff;                    //初始化扫描变量
    P0 = 0xff;                      //P0 输出高使能内部若上拉,准备读取按键状态
    P06  = 0; P07 = 1;              //准备扫描第 1 行
```

```
    NOP12();                        //等待 12 个 nop 再读取按键状态，
                                    //NOP12()这个宏已在 AI8051U.H 的头文件中定义
    if (!P00) key =0;               //检查第 1 行第 1 列按键是否被按下
    else if     (!P01) key = 1;     //检查第 1 行第 2 列按键是否被按下
    else if     (!P02) key = 2;     //检查第 1 行第 3 列按键是否被按下
    else if     (!P03) key = 3;     //检查第 1 行第 4 列按键是否被按下
    else                            //如果第一行没有按键按下则扫描下一行
    {
        P07  = 0;                   //准备扫描第 2 行
        P06  = 1;
        NOP12();                    //等待 12 个 nop 再读取按键状态
                                    //NOP12()这个宏已在 AI8051U.H 的头文件中定义
        if (!P00) key =4;           //检查第 2 行第 1 列按键是否被按下
        else if     (!P01) key = 5; //检查第 2 行第 2 列按键是否被按下
        else if     (!P02) key = 6; //检查第 2 行第 3 列按键是否被按下
        else if     (!P03) key = 7; //检查第 2 行第 4 列按键是否被按下
    }
    P0 = 0xff;                      //恢复 P0 口状态

    if (key     != keycode)         //判断按键状态是否有变化
    {
        keycode   = key;            //保存键码
        debounce = 5;               //初始化去抖动变量
    }
    else
    {
        if (debounce !=      0)     //是否
        {
            if (--debounce == 0)    //判断去抖动是否已经完成
            {
                if (keycode     != 0xff)
//判断是松开按键还是按下按键,键码为 0xff 表示是松开按键,非 0xff 为按下按键
                {
                    keyok =   1;    //设置按键按下标志

                    printf_usb("Key%d 被按下\r\n", keycode);
//直接打印输出 “Key%d 被按下” 到主机相应软件的 USB【CDC/HID-串口助手】
                }
            }
        }
    }
}

void main(void)
{
    EAXFR =      1;                 //允许访问扩展的特殊寄存器，XFR
```

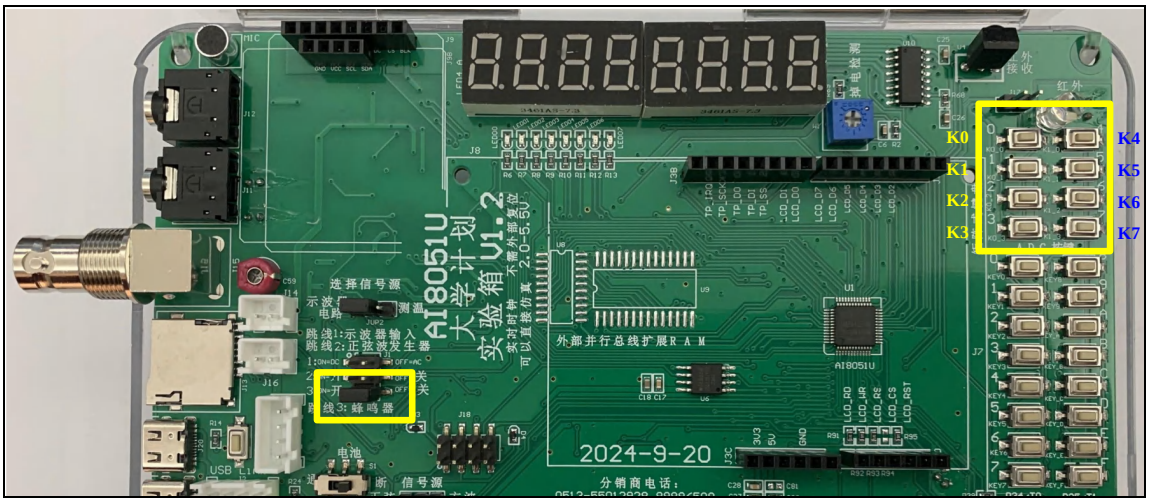
```
    WTST = 0;
//设置取程序代码等待时间，赋值为 0 表示不等待，程序以最快速度运行
    CKCON =    0;
//设置访问片内的 xdata 速度，赋值为 0  表示用最快速度访问，不增加额外的等待时间
    P0M0 = 0x00; P0M1 =  0x00;    //设置 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 =  0x00;    //设置 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 =  0x00;    //设置 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 =  0x00;    //设置 P3 口为准双向口模式
    P4M0 = 0x00; P4M1 =  0x00;    //设置 P4 口为准双向口模式
    P5M0 = 0x01; P5M1 =  0x00;    //设置 P5.0  口为强推挽，其他设置为准双向口模式
    P6M0 = 0x00; P6M1 =  0x00;    //设置 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 =  0x00;    //设置 P7 口为准双向口模式
    keycode  = 0xff;              //初始化键码(0~7 为有效键码,0xff 为无效键码)
    keyok =  0;                   //初始化按键标志

    usb_init();     //调用 USB 初始化函数，不需要立即判断电脑已正确识别到 USB 从设备
    /* USB 型单片机从设备，如需要主动向电脑发送数据，
       在执行 USB_SendData( )函数和 printf_usb( )函数时，
       这两个函数已增加了判断电脑是否已正确识别到 USB 从设备的程序。
       如果电脑要主动发送数据给 USB 从设备，电脑自己会主动判断与 USB 从设备是否已正确连接。
    */
    EA = 1;                       //总中断允许位打开

    P50  = 0;                     //关闭蜂鸣器
    P40  = 0;                     //打开 LED 部分的供电
    BootLogo();                   //显示开机画面,  8 个 LED 同时闪烁 3 次

    while(1)
    {                             //假定整个主循环的执行时间为 10ms 左右
        ScanKey();                //按键扫描
        KeyEvents();              //按键事件处理

        DetectTemp();             //温度检查任务
        ControlTemp();            //温度控制任务
        OtherEvent();             //其他事件
    }
}
```

e) 程序下载

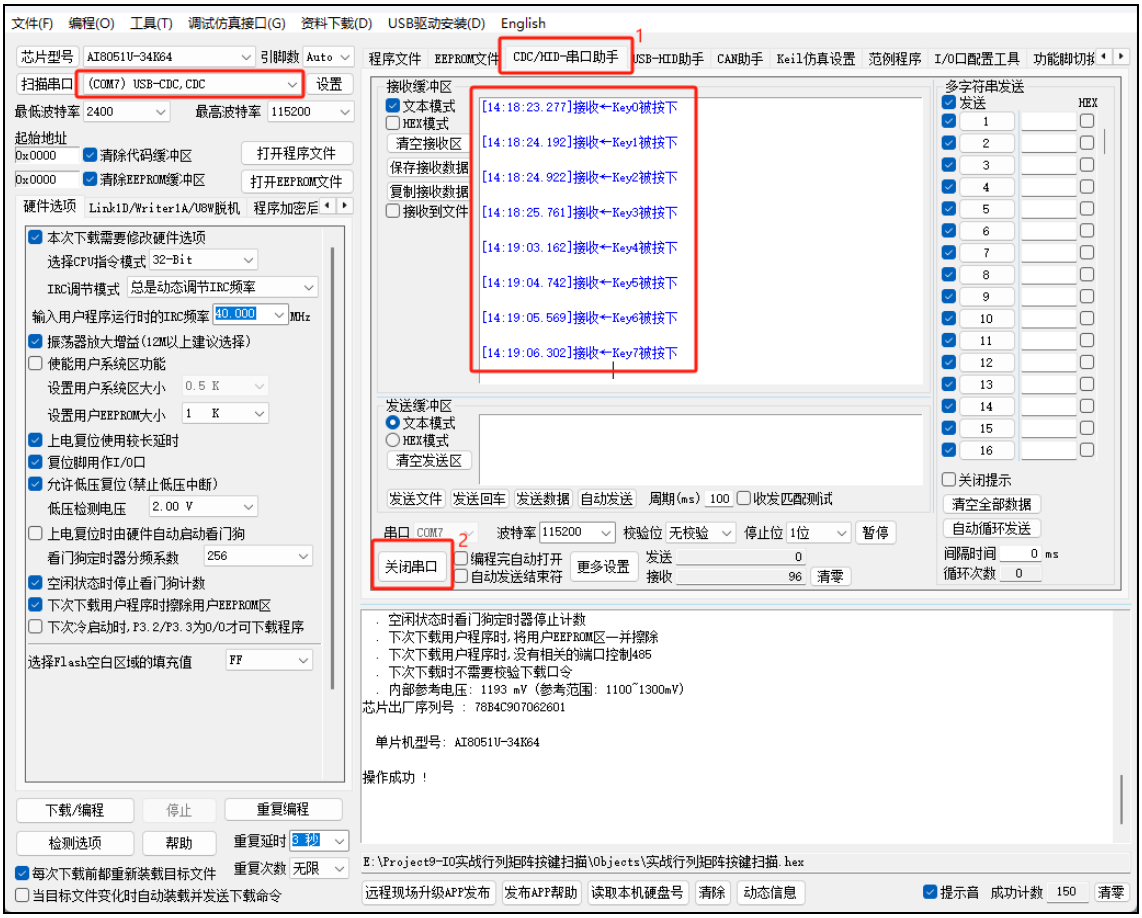
程序下载完成以后，就可以看到出现了一个 USB-CDC 端口，此时就可以通过这个 USB-CDC 端口进行数据通讯了。

f) 程序实现效果

程序编写好以后，编译无误后加载到实验箱，8 个 LED 灯就会同时闪烁 3 次，按下对应 K0_0 ~K0_3、K1_0 ~ K1_3 这 8 个按键（如下图）后，蜂鸣器会发出“Do、Re、Mi、Fa、So、La、Si、Do”的音符声：



AIapp-ISP 软件“CDC/HID-串口助手”选项卡会显示如下效果：



15 中断系统及外部中断 INT0/1/2/3/4 实验

中断系统是为使 CPU 具有对外界紧急事件的实时处理能力而设置的。

当中央处理机 CPU 正在处理某件事的时候外界发生了紧急事件请求，要求 CPU 暂停当前的工作，转而去处理这个紧急事件，处理完以后，再回到原来被中断的地方，继续原来的工作，这样的过程称为中断。实现这种功能的部件称为中断系统，请示 CPU 中断的请求源称为中断源。微型机的中断系统一般允许多个中断源，当几个中断源同时向 CPU 请求中断，要求为它服务的时候，这就存在 CPU 优先响应哪一个中断源请求的问题。通常根据中断源的轻重缓急排队，优先处理最紧急事件的中断请求源，即规定每一个中断源有一个优先级别。CPU 总是先响应优先级别最高的中断请求。

当 CPU 正在处理一个中断源请求的时候（执行相应的中断服务程序），发生了另外一个优先级比它还高的中断源请求。如果 CPU 能够暂停对原来中断源的服务程序，转而去处理优先级更高的中断请求源，处理完以后，再回到原低级中断服务程序，这样的过程称为中断嵌套。这样的中断系统称为多级中断系统，没有中断嵌套功能的中断系统称为单级中断系统。

用户可以用关总中断允许位（EA/IE.7）或相应中断的允许位屏蔽相应的中断请求，也可以用打开相应的中断允许位来使 CPU 响应相应的中断申请，每一个中断源可以用软件独立地控制为开中断或关中断状态，部分中断的优先级别均可用软件设置。高优先级的中断请求可以打断低优先级的中断，反之，低优先级的中断请求不可以打断高优先级的中断。当两个相同优先级的中断同时产生时，将由查询次序来决定系统先响应哪个中断。

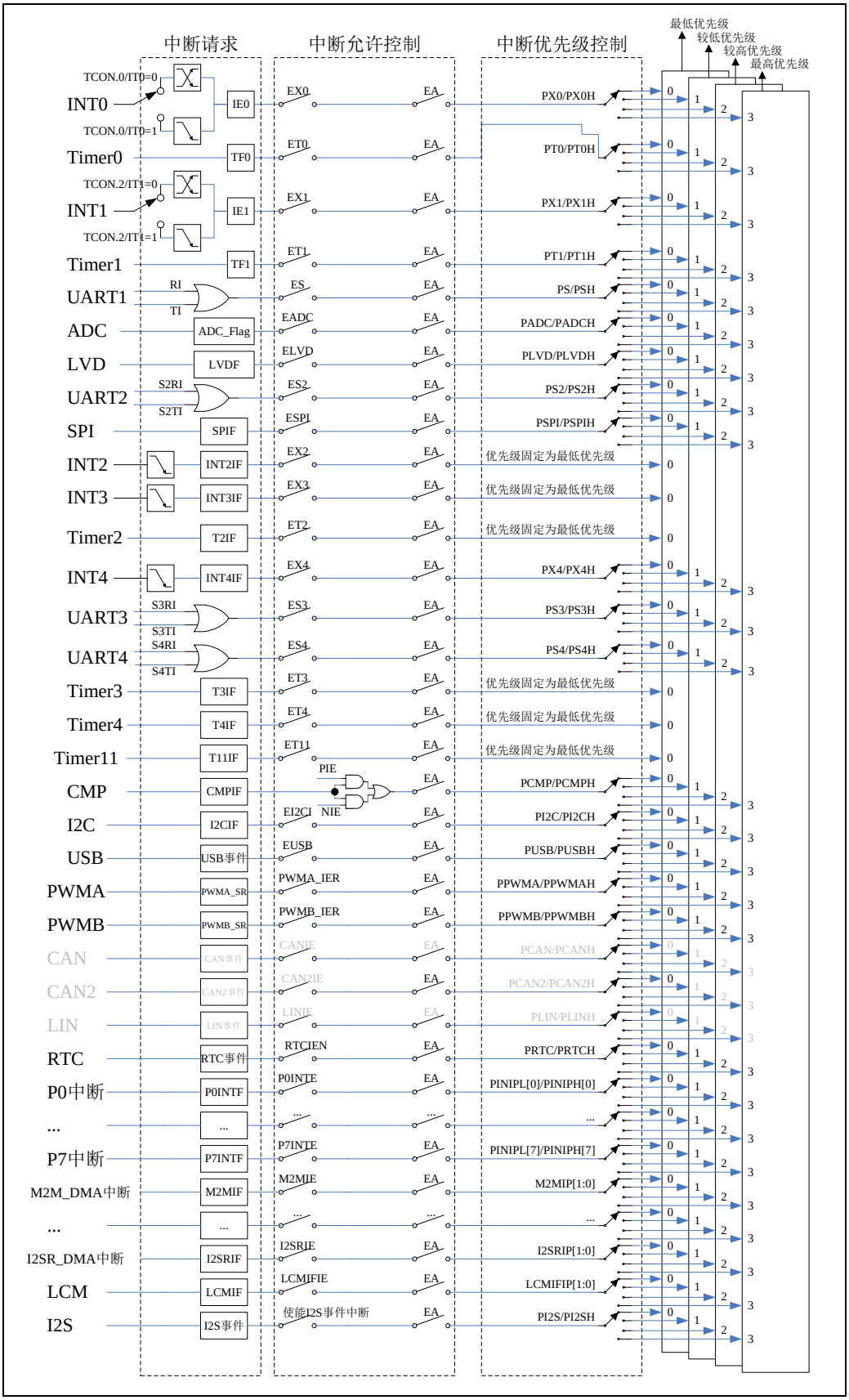
15.1 Ai8051U 系列中断源

中断源	Ai8051U系列
外部中断 0 中断（INT0，从 P3.2 管脚输入）	√
定时器 0 中断（Timer0）	√
外部中断 1 中断（INT1，从 P3.3 管脚输入）	√
定时器 1 中断（Timer1）	√
串口 1 中断（UART1）	√
模数转换中断（ADC）	√
低压检测中断（LVD）	√
CCP/PCA/PWM 中断	√
串口 2 中断（UART2）	√
串行外设接口中断（SPI）	√
外部中断 2 中断（INT2，从 P3.6 管脚输入）	√
外部中断 3 中断（INT3，从 P3.7 管脚输入）	√
定时器 2 中断（Timer2）	√
外部中断 4 中断（INT4，从 P3.0 管脚输入）	√
串口 3 中断（UART3）	√
串口 4 中断（UART4）	√
定时器 3 中断（Timer3）	√
定时器 4 中断（Timer4）	√

比较器中断（CMP）	√
I2C 总线中断	√
USB 中断	√
PWMA	√
PWMB	√
RTC 中断	√
P0 口中断	√
P1 口中断	√
P2 口中断	√
P3 口中断	√
P4 口中断	√
P5 口中断	√
M2M_DMA 中断	√
ADC_DMA 中断	√
SPI_DMA 中断	√
串口 1 发送 DMA 中断	√
串口 1 接收 DMA 中断	√
串口 2 发送 DMA 中断	√
串口 2 接收 DMA 中断	√
串口 3 发送 DMA 中断	√
串口 3 接收 DMA 中断	√
串口 4 发送 DMA 中断	√
串口 4 接收 DMA 中断	√
TFT 彩屏 DMA 中断	√
TFT 彩屏中断	√
I2C 发送 DMA 中断	√
I2C 接收 DMA 中断	√
I2S 中断	√
I2S 发送 DMA 中断	√
I2S 接收 DMA 中断	√
QSPI_DMA 中断	√
QSPI 中断	√
定时器 11 中断（Timer11）	√
PWMA 发送 DMA 中断	√
PWMA 接收 DMA 中断	√

15.2 Ai8051U 中断及中断优先级结构图

注：高优先级中断可以打断正在执行中的低优先级中断，跟传统 STC89C52 一样



15.3 Ai8051U 系列中断向量地址及同级中断优先级中断查询次序表

中断源	中断向量		相同优先级内的查询次序	优先级设置	优先级 0（最低）	优先级 1	优先级 2	优先级 3（最高）	中断请求位	中断允许位
	Ai8051U	AI8G/H								
INT0/P3.2 中断	FF:0003H	0003H	0（最高）	PX0, PX0H	0	1	2	3	IE0	EX0
Timer0 中断	FF:000BH	000BH	1	PT0, PT0H	0	1	2	3	TF0	ET0
INT1/P3.3 中断	FF:0013H	0013H	2	PX1, PX1H	0	1	2	3	IE1	EX1
Timer1 中断	FF:001BH	001BH	3	PT1, PT1H	0	1	2	3	TF1	ET1
UART1 中断	FF:0023H	0023H	4	PS, PSH	0	1	2	3	RI || TI	ES
ADC 中断	FF:002BH	002BH	5	PADC, PADCH	0	1	2	3	ADC_FLAG	EADC
LVD 中断	FF:0033H	0033H	6	PLVD, PLVDH	0	1	2	3	LVDF	ELVD
PCA 中断	FF:003BH	003BH	7	PPCA, PPCAH	0	1	2	3	CF	ECF
									CCF0	ECCF0
									CCF1	ECCF1
									CCF2	ECCF2
UART2 中断	FF:0043H	0043H	8	PS2, PS2H	0	1	2	3	S2RI || S2TI	ES2
SPI 中断	FF:004BH	004BH	9	PSPI, PSPIH	0	1	2	3	SPIF	ESPI
INT2/P3.6 中断	FF:0053H	0053H	10		0	0	0	0	INT2IF	EX2
INT3/P3.7 中断	FF:005BH	005BH	11		0	0	0	0	INT3IF	EX3
Timer2 中断	FF:0063H	0063H	12		0	0	0	0	T2IF	ET2
INT4/P3.0 中断	FF:0083H	0083H	16	PX4, PX4H	0	1	2	3	INT4IF	EX4
UART3 中断	FF:008BH	008BH	17	PS3, PS3H	0	1	2	3	S3RI || S3TI	ES3
UART4 中断	FF:0093H	0093H	18	PS4, PS4H	0	1	2	3	S4RI || S4TI	ES4
Timer3 中断	FF:009BH	009BH	19		0	0	0	0	T3IF	ET3
Timer4 中断	FF:00A3H	00A3H	20		0	0	0	0	T4IF	ET4
CMP 中断	FF:00ABH	00ABH	21	PCMP, PCMPH	0	1	2	3	CMPIF	PIE|NIE
I2C 中断	FF:00C3H	00C3H	24	PI2C, PI2CH	0	1	2	3	MSIF	EMSI
					0	1	2	3	STAIF	ESTAI
					0	1	2	3	RXIF	ERXI
					0	1	2	3	TXIF	ETXI
					0	1	2	3	STOIF	ESTOI
USB 中断	FF:00CBH	00CBH	25	PUSB, PUSBH	0	1	2	3	USB Events	EUSB
PWMA 中断	FF:00D3H	00D3H	26	PPWMA, PPWMAH	0	1	2	3	PWMA_SR	PWMA_IER
PWMB 中断	FF:00DBH	00DBH	27	PPWMB, PPWMBH	0	1	2	3	PWMB_SR	PWMB_IER
RTC 中断	FF:0123H	0123H	36	PRTC, PRTCH	0	1	2	3	ALAIF	EALAI
									DAYIF	EDAYI
									HOURIF	EHOURI
									MINIF	EMINI
									SECIF	ESECI
									SEC2IF	ESEC2I
									SEC8IF	ESEC8I
									SEC32IF	ESEC32I

中断源	中断向量		相同优先级内的查询次序	优先级设置	优先级 0（最低）	优先级 1	优先级 2	优先级 3（最高）	中断请求位	中断允许位
	Ai8051U	AI8G/H								
P0 中断	FF:012BH	012BH	37	PINIPL[0], PINIPH[0]	0	1	2	3	P0INTF	P0INTE
P1 中断	FF:0133H	0133H	38	PINIPL[1], PINIPH[1]	0	1	2	3	P1INTF	P1INTE
P2 中断	FF:013BH	013BH	39	PINIPL[2], PINIPH[2]	0	1	2	3	P2INTF	P2INTE
P3 中断	FF:0143H	0143H	40	PINIPL[3], PINIPH[3]	0	1	2	3	P3INTF	P3INTE
P4 中断	FF:014BH	014BH	41	PINIPL[4], PINIPH[4]	0	1	2	3	P4INTF	P4INTE
P5 中断	FF:0153H	0153H	42	PINIPL[5], PINIPH[5]	0	1	2	3	P5INTF	P5INTE
P6 中断	FF:015BH	015BH	43	PINIPL[6], PINIPH[6]	0	1	2	3	P6INTF	P6INTE
P7 中断	FF:0163H	0163H	44	PINIPL[7], PINIPH[7]	0	1	2	3	P7INTF	P7INTE
DMA_M2M 中断	FF:017BH	017BH	47	M2MIP[1:0]	0	1	2	3	M2MIF	M2MIE
DMA_ADC 中断	FF:0183H	0183H	48	ADCIP[1:0]	0	1	2	3	ADCIF	ADCIE
DMA_SPI 中断	FF:018BH	018BH	49	SPIIP[1:0]	0	1	2	3	SPIIF	SPIIE
DMA_UR1T 中断	FF:0193H	0193H	50	UR1TIP[1:0]	0	1	2	3	UR1TIF	UR1TIE
DMA_UR1R 中断	FF:019BH	019BH	51	UR1RIP[1:0]	0	1	2	3	UR1RIF	UR1RIE
DMA_UR2T 中断	FF:01A3H	01A3H	52	UR2TIP[1:0]	0	1	2	3	UR2TIF	UR2TIE
DMA_UR2R 中断	FF:01ABH	01ABH	53	UR2RIP[1:0]	0	1	2	3	UR2RIF	UR2RIE
DMA_UR3T 中断	FF:01B3H	01B3H	54	UR3TIP[1:0]	0	1	2	3	UR3TIF	UR3TIE
DMA_UR3R 中断	FF:01BBH	01BBH	55	UR3RIP[1:0]	0	1	2	3	UR3RIF	UR3RIE
DMA_UR4T 中断	FF:01C3H	01C3H	56	UR4TIP[1:0]	0	1	2	3	UR4TIF	UR4TIE
DMA_UR4R 中断	FF:01CBH	01CBH	57	UR4RIP[1:0]	0	1	2	3	UR4RIF	UR3RIE
TFT 彩屏 DMA 中断	FF:01D3H	01D3H	58	LCMIP[1:0]	0	1	2	3	LCMIF	LCMIE
TFT 彩屏中断	FF:01DBH	01DBH	59	LCMIFIP[1:0]	0	1	2	3	LCMIFIF	LCMIFIE
DMA_I2CT 中断	FF:01E3H	01E3H	60	I2CTIP[1:0]	0	1	2	3	I2CTIF	I2CTIE
DMA_I2CR 中断	FF:01EBH	01EBH	61	I2CRIP[1:0]	0	1	2	3	I2CRIF	I2CRIE
I2S 中断	FF:01F3H	01F3H	62	PI2S,PI2SH	0	1	2	3	TXE	TXEIE
					0	1	2	3	RXNE	RXNEIE
					0	1	2	3	FRE	ERRIE
									OVR	
									UDR	
DMA_I2ST 中断	FF:01FBH	01FBH	63	I2STIP[1:0]	0	1	2	3	I2STIF	I2STIE
DMA_I2SR 中断	FF:0203H	0203H	64	I2SRIP[1:0]	0	1	2	3	I2SRIF	I2SRIE
DMA_QSPI 中断	FF:020BH	020BH	65	QSPIIP[1:0]	0	1	2	3	QSPIIF	QSPIIE
QSPI 中断	FF:0213H	0213H	66	QSPI_IP[1:0]	0	1	2	3	SMF	SMIE
									FTF	FTIE
									TCF	TCIE
									TEF	TEIE

中断源	中断向量		相同优先级内的查询次序	优先级设置	优先级 0（最低）	优先级 1	优先级 2	优先级 3（最高）	中断请求位	中断允许位
	Ai8051U	AI8G/H								
Timer11	FF:021BH	021BH	67		0	0	0	0	T11IF	ET11I
DMA_PWMAT 中断	FF:0243H	0243H	72	PWMATIP[1:0]	0	1	2	3	PWMATIF	PWMATIE
DMA_PWMAR 中断	FF:024BH	024BH	73（最低）	PWMARIP[1:0]	0	1	2	3	PWMARIF	PWMARIE

此处的查询次序是指：

同一个中断优先级中，如有多个中断源产生，内部硬件逻辑会按从 0～N 的顺序，优先响应查询次序小的中断源，即优先执行查询次序小的中断源的中断服务程序

在 C 语言中声明中断服务程序

```
void    INT0_Routine(void)      interrupt  0;
void    TM0_Rountine(void)      interrupt  1;
void    INT1_Routine(void)      interrupt  2;
void    TM1_Rountine(void)      interrupt  3;
void    UART1_Routine(void)     interrupt  4;
void    ADC_Routine(void)       interrupt  5;
void    LVD_Routine(void)       interrupt  6;
void    PCA_Routine(void)       interrupt  7;
void    UART2_Routine(void)     interrupt  8;
void    SPI_Routine(void)       interrupt  9;
void    INT2_Routine(void)      interrupt  10;
void    INT3_Routine(void)      interrupt  11;
void    TM2_Routine(void)       interrupt  12;
void    INT4_Routine(void)      interrupt  16;
void    UART3_Routine(void)     interrupt  17;
void    UART4_Routine(void)     interrupt  18;
void    TM3_Routine(void)       interrupt  19;
void    TM4_Routine(void)       interrupt  20;
void    CMP_Routine(void)       interrupt  21;
void    I2C_Routine(void)       interrupt  24;
void    USB_Routine(void)       interrupt  25;
void    PWMA_Routine(void)      interrupt  26;
void    PWMB_Routine(void)      interrupt  27;
```

中断号超过31的C语言中断服务程序不能直接用interrupt声明，请参考“开发环境的建立与ISP下载”章节中的“关于中断号大于31在Keil中编译出错的处理”小节的处理方法。汇编语言不受影响

15.4 中断相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
IE	中断允许寄存器	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000,0000
IE2	中断允许寄存器 2	AFH	EUSB	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	0000,0000
IP	中断优先级控制寄存器	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	x000,0000
IPH	高中断优先级控制寄存器	B7H	PPCAH	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H	x000,0000
IP2	中断优先级控制寄存器 2	B5H	PUSB	PI2C	PCMP	PX4	PPWMB	PPWMA	PSPI	PS2	0000,0000
IP2H	高中断优先级控制寄存器 2	B6H	PUSBH	PI2CH	PCMPH	PX4H	PPWMBH	PPWMAH	PSPIH	PS2H	0000,0000
IP3	中断优先级控制寄存器 3	DFH	-	-	-	-	PI2S	PRTC	PS4	PS3	xxxx,0000
IP3H	高中断优先级控制寄存器 3	EEH	-	-	-	-	PI2SH	PRTCH	PS4H	PS3H	xxxx,0000
PCON	电源控制寄存器	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
TCON	定时器控制寄存器	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
INTCLKO	中断与时钟输出控制寄存器	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000
AUXINTIF	扩展外部中断标志寄存器	EFH	-	INT4IF	INT3IF	INT2IF	-	T4IF	T3IF	T2IF	x000,x000
SCON	串口 1 控制寄存器	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
S2CON	串口 2 控制寄存器	9AH	S2SM0/FE	S2SM1	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0000,0000
S3CON	串口 3 控制寄存器	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000
S4CON	串口 4 控制寄存器	FDH	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000,0000
ADC_CONTR	ADC 控制寄存器	BCH	ADC_POWER	ADC_START	ADC_FLAG	ADC_EPWMT	ADC_CHS[3:0]				0000,0000
SPSTAT	SPI 状态寄存器	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx,xxxx
CMPCR1	比较器控制寄存器 1	E6H	CMPEN	CMPIF	PIE	NIE	-	-	CMPOE	CMPRES	0000,xx00

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
LCMIFCFG	TFT 彩屏接口配置寄存器	7EFE50H	LCMIFIE	-	LCMIFIP[1:0]		LCMIFDPS[1:0]		D16_D8	M68_I80	0x00,0000
LCMIFSTA	TFT 彩屏接口状态寄存器	7EFE53H	-	-	-	-	-	-	-	LCMIFIF	xxxx,xxx0
RTCIEN	RTC 中断使能寄存器	7EFE62H	EALAI	EDAYI	EHOURI	EMINI	ESECI	ESEC2I	ESEC8I	ESEC32I	0000,0000
RTCIF	RTC 中断请求寄存器	7EFE63H	ALAIF	DAYIF	HOURIF	MINIF	SECIF	SEC2IF	SEC8IF	SEC32IF	0000,0000
T11CR	定时器 T11 控制寄存器	7EFE78H	T11R	T11_C/T	T11CLKO	T11x12	T11CS[1:0]		ET11I	T11IF	0000,0000
I2CMSCR	I²C 主机控制寄存器	7EFE81H	EMSI	-	-	-	MSCMD[3:0]				0xxx,0000
I2CMSST	I²C 主机状态寄存器	7EFE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO	00xx,xx10
I2CSLCR	I²C 从机控制寄存器	7EFE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0
I2CSLST	I²C 从机状态寄存器	7EFE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000
PWMA_IER	PWMA 中断使能寄存器	7EFEC4H	BIE	TIE	COMIE	CC4IE	CC3IE	CC2IE	CC1IE	UIE	0000,0000
PWMA_SR1	PWMA 状态寄存器 1	7EFEC5H	BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	UIF	0000,0000
PWMA_SR2	PWMA 状态寄存器 2	7EFEC6H	-	-	-	CC4OF	CC3OF	CC2OF	CC1OF	-	xxx0,000x
PWMB_IER	PWMB 中断使能寄存器	7EFEE4H	BIE	TIE	COMIE	CC8IE	CC7IE	CC6IE	CC5IE	UIE	0000,0000
PWMB_SR1	PWMB 状态寄存器 1	7EFEE5H	BIF	TIF	COMIF	CC8IF	CC7IF	CC6IF	CC5IF	UIF	0000,0000
PWMB_SR2	PWMB 状态寄存器 2	7EFEE6H	-	-	-	CC8OF	CC7OF	CC6OF	CC5OF	-	xxx0,000x

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
P0INTE	P0 口中断使能寄存器	7EFD00H	P0INTE[7:0]								0000,0000
P1INTE	P1 口中断使能寄存器	7EFD01H	P1INTE[7:0]								0000,0000
P2INTE	P2 口中断使能寄存器	7EFD02H	P2INTE[7:0]								0000,0000
P3INTE	P3 口中断使能寄存器	7EFD03H	P3INTE[7:0]								0000,0000
P4INTE	P4 口中断使能寄存器	7EFD04H	P4INTE[7:0]								0000,0000
P5INTE	P5 口中断使能寄存器	7EFD05H	P5INTE[7:0]								0000,0000
P6INTE	P6 口中断使能寄存器	7EFD06H	P6INTE[7:0]								0000,0000
P7INTE	P7 口中断使能寄存器	7EFD07H	P7INTE[7:0]								0000,0000
P0INTF	P0 口中断标志寄存器	7EFD10H	P0INTF[7:0]								0000,0000
P1INTF	P1 口中断标志寄存器	7EFD11H	P1INTF[7:0]								0000,0000
P2INTF	P2 口中断标志寄存器	7EFD12H	P2INTF[7:0]								0000,0000
P3INTF	P3 口中断标志寄存器	7EFD13H	P3INTF[7:0]								0000,0000
P4INTF	P4 口中断标志寄存器	7EFD14H	P4INTF[7:0]								0000,0000
P5INTF	P5 口中断标志寄存器	7EFD15H	P5INTF[7:0]								0000,0000
P6INTF	P6 口中断标志寄存器	7EFD16H	P6INTF[7:0]								0000,0000
P7INTF	P7 口中断标志寄存器	7EFD17H	P7INTF[7:0]								0000,0000
CCAPM2	PCA 模块 2 模式控制寄存器	7EFD50H	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2	x000,0000
CCAPM3	PCA 模块 3 模式控制寄存器	7EFD54H	-	ECOM3	CCAPP3	CCAPN3	MAT3	TOG3	PWM3	ECCF3	x000,0000
CCAPM0	PCA 模块 0 模式控制寄存器	7EFD58H	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0	x000,0000
CCAPM1	PCA 模块 1 模式控制寄存器	7EFD5CH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1	x000,0000
PINIPL	I/O 口中断优先级低寄存器	7EFD60H	P7IP	P6IP	P5IP	P4IP	P3IP	P2IP	P1IP	P0IP	0000,0000
PINIPH	I/O 口中断优先级高寄存器	7EFD61H	P7IPH	P6IPH	P5IPH	P4IPH	P3IPH	P2IPH	P1IPH	P0IPH	0000,0000
CCON	PCA 控制寄存器	7EFD64H	CF	CR	-	-	-	CCF2	CCF1	CCF0	00xx,x000
CMOD	PCA 模式寄存器	7EFD67H	CIDL	CCP_S[1:0]		CPS[3:0]				ECF	0xxx,0000
UR1TOCR	串口 1 接收超时控制寄存器	7EFD70H	ENTO	ENTOI	SCALE	-	-	-	-	-	000x,xxxx
UR1TOSR	串口 1 接收超时状态寄存器	7EFD71H	CTOIF	-	-	-	-	-	-	TOIF	0xxx,xxx0
UR2TOCR	串口 2 接收超时控制寄存器	7EFD74H	ENTO	ENTOI	SCALE	-	-	-	-	-	000x,xxxx
UR2TOSR	串口 2 接收超时状态寄存器	7EFD75H	CTOIF	-	-	-	-	-	-	TOIF	0xxx,xxx0
UR3TOCR	串口 3 接收超时控制寄存器	7EFD78H	ENTO	ENTOI	SCALE	-	-	-	-	-	000x,xxxx
UR3TOSR	串口 3 接收超时状态寄存器	7EFD79H	CTOIF	-	-	-	-	-	-	TOIF	0xxx,xxx0
UR4TOCR	串口 4 接收超时控制寄存器	7EFD7CH	ENTO	ENTOI	SCALE	-	-	-	-	-	000x,xxxx
UR4TOSR	串口 4 接收超时状态寄存器	7EFD7DH	CTOIF	-	-	-	-	-	-	TOIF	0xxx,xxx0
SPITOCR	SPI 从机超时控制寄存器	7EFD80H	ENTO	ENTOI	SCALE	-	-	-	-	-	000x,xxxx
SPITOSR	SPI 从机超时状态寄存器	7EFD81H	CTOIF	-	-	-	-	-	-	TOIF	0xxx,xxx0
I2CTOCR	I2C 从机超时控制寄存器	7EFD84H	ENTO	ENTOI	SCALE	-	-	-	-	-	000x,xxxx
I2CTOSR	I2C 从机超时状态寄存器	7EFD85H	CTOIF	-	-	-	-	-	-	TOIF	0xxx,xxx0
I2SCR	I2S 控制寄存器	7EFD98H	TXEIE	RXNEIE	ERRIE	FRF	-	-	TXDMAEN	RXDMAEN	0000,xx00
I2SSR	I2S 状态寄存器	7EFD99H	-	FRE	BUY	OVR	UDR	CHSID	TXE	RXNE	x000,0000

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
DMA_M2M_CFG	M2M_DMA 配置寄存器	7EFA00H	M2MIE	-	TXACO	RXACO	M2MIP[1:0]		M2MPTY[1:0]		0x00,0000
DMA_M2M_STA	M2M_DMA 状态寄存器	7EFA02H	-	-	-	-	-	-	-	M2MIF	xxxx,xxx0
DMA_ADC_CFG	ADC_DMA 配置寄存器	7EFA10H	ADCIE	-	-	-	ADCMIP[1:0]		ADCPTY[1:0]		0xxx,0000
DMA_ADC_STA	ADC_DMA 状态寄存器	7EFA12H	-	-	-	-	-	-	-	ADCIF	xxxx,xxx0
DMA_SPI_CFG	SPI_DMA 配置寄存器	7EFA20H	SPIIE	ACT_TX	ACT_RX	-	SPIIP[1:0]		SPIPTY[1:0]		000x,0000
DMA_SPI_STA	SPI_DMA 状态寄存器	7EFA22H	-	-	-	-	-	TXOVW	RXLOSS	SPIIF	xxxx,x000
DMA_UR1T_CFG	UR1T_DMA 配置寄存器	7EFA30H	UR1TIE	-	-	-	UR1TIP[1:0]		UR1TPTY[1:0]		0xxx,0000
DMA_UR1T_STA	UR1T_DMA 状态寄存器	7EFA32H	-	-	-	-	-	TXOVW	-	UR1TIF	xxxx,x0x0
DMA_UR1R_CFG	UR1R_DMA 配置寄存器	7EFA38H	UR1RIE	-	-	-	UR1RIP[1:0]		UR1RPTY[1:0]		0xxx,0000
DMA_UR1R_STA	UR1R_DMA 状态寄存器	7EFA3AH	-	-	-	-	-	-	RXLOSS	UR1RIF	xxxx,xx00
DMA_UR2T_CFG	UR2T_DMA 配置寄存器	7EFA40H	UR2TIE	-	-	-	UR2TIP[1:0]		UR2TPTY[1:0]		0xxx,0000
DMA_UR2T_STA	UR2T_DMA 状态寄存器	7EFA42H	-	-	-	-	-	TXOVW	-	UR2TIF	xxxx,x0x0
DMA_UR2R_CFG	UR2R_DMA 配置寄存器	7EFA48H	UR2RIE	-	-	-	UR2RIP[1:0]		UR2RPTY[1:0]		0xxx,0000
DMA_UR2R_STA	UR2R_DMA 状态寄存器	7EFA4AH	-	-	-	-	-	-	RXLOSS	UR2RIF	xxxx,xx00
DMA_UR3T_CFG	UR3T_DMA 配置寄存器	7EFA50H	UR3TIE	-	-	-	UR3TIP[1:0]		UR3TPTY[1:0]		0xxx,0000
DMA_UR3T_STA	UR3T_DMA 状态寄存器	7EFA52H	-	-	-	-	-	TXOVW	-	UR3TIF	xxxx,x0x0
DMA_UR3R_CFG	UR3R_DMA 配置寄存器	7EFA58H	UR3RIE	-	-	-	UR3RIP[1:0]		UR3RPTY[1:0]		0xxx,0000
DMA_UR3R_STA	UR3R_DMA 状态寄存器	7EFA5AH	-	-	-	-	-	-	RXLOSS	UR3RIF	xxxx,xx00
DMA_UR4T_CFG	UR4T_DMA 配置寄存器	7EFA60H	UR4TIE	-	-	-	UR4TIP[1:0]		UR4TPTY[1:0]		0xxx,0000
DMA_UR4T_STA	UR4T_DMA 状态寄存器	7EFA62H	-	-	-	-	-	TXOVW	-	UR4TIF	xxxx,x0x0
DMA_UR4R_CFG	UR4R_DMA 配置寄存器	7EFA68H	UR4RIE	-	-	-	UR4RIP[1:0]		UR4RPTY[1:0]		0xxx,0000
DMA_UR4R_STA	UR4R_DMA 状态寄存器	7EFA6AH	-	-	-	-	-	-	RXLOSS	UR4RIF	xxxx,xx00
DMA_LCM_CFG	TFT 彩屏 DMA 配置寄存器	7EFA70H	LCMIE	-	-	-	LCMIP[1:0]		LCMPTY[1:0]		0xxx,0000
DMA_LCM_STA	TFT 彩屏 DMA 状态寄存器	7EFA72H	-	-	-	-	-	-	TXOVW	LCMIF	xxxx,xx00
DMA_I2CT_CFG	I2CT_DMA 配置寄存器	7EFA98H	I2CTIE	-	-	-	I2CTIP[1:0]		I2CTPTY[1:0]		0xxx,0000
DMA_I2CT_STA	I2CT_DMA 状态寄存器	7EFA9AH	-	-	-	-	-	TXOVW	-	I2CTIF	xxxx,x0x0
DMA_I2CR_CFG	I2CR_DMA 配置寄存器	7EFAA0H	I2CRIE	-	-	-	I2CRIP[1:0]		I2CRPTY[1:0]		0xxx,0000
DMA_I2CR_STA	I2CR_DMA 状态寄存器	7EFAA2H	-	-	-	-	-	-	RXLOSS	I2CRIF	xxxx,xx00
DMA_I2ST_CFG	I2ST_DMA 配置寄存器	7EFAB0H	I2STIE	-	-	-	I2STIP[1:0]		I2STPTY[1:0]		0xxx,0000
DMA_I2ST_STA	I2ST_DMA 状态寄存器	7EFAB2H	-	-	-	-	-	TXOVW	-	I2STIF	xxxx,x0x0
DMA_I2SR_CFG	I2SR_DMA 配置寄存器	7EFAB8H	I2SRIE	-	-	-	I2SRIP[1:0]		I2SRPTY[1:0]		0xxx,0000
DMA_I2SR_STA	I2SR_DMA 状态寄存器	7EFABAH	-	-	-	-	-	-	RXLOSS	I2SRIF	xxxx,xx00
DMA_QSPI_CFG	QSPI_DMA 配置寄存器	7EFAD0H	QSPIIE	ACT_WR	ACT_RD	-	QSPIIP[1:0]		QSPIPTY[1:0]		000x,0000
DMA_QSPI_STA	QSPI_DMA 状态寄存器	7EFAD2H	-	-	-	-	-	-	-	QSPIIF	xxxx,xxx0

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
QSPI_CR3	QSPI 控制寄存器 3	7EF902H	PMM	APMS	-	-	SMIE	FTIE	TCIE	TEIE	00xx,0000
QSPI_DCR1	QSPI 器件配置寄存器 1	7EF904H	-	CSHT[2:0]			QSPI_IP[1:0]		-	CKMODE	x000,00x0
QSPI_SR1	QSPI 状态寄存器 1	7EF906H	-	-	BUSY	-	SMF	FTF	TCF	TEF	xx0x,0100
DMA_PWMAT_CFG	PWMAT_DMA 配置寄存器	7EF980H	PWMATIE	-	-	-	PWMATMIP[1:0]		PWMATBAP[1:0]		0xxx,0000
DMA_PWMAT_STA	PWMAT_DMA 状态寄存器	7EF982H	RDOV	-	-	-	-	TXOVW	-	PWMATTIF	0xxx,x0x0
DMA_PWMAR_CFG	PWMAR_DMA 配置寄存器	7EF990H	PWMARIE	-	-	-	PWMARMIP[1:0]		PWMARBAP[1:0]		0xxx,0000
DMA_PWMAR_STA	PWMAR_DMA 状态寄存器	7EF992H	WROV	-	-	-	-	-	RXLOSS	PWMATRIF	0xxx,xx00

15.4.1 中断使能寄存器（中断允许位）

IE（中断使能寄存器）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

EA：总中断允许控制位。EA 的作用是使中断允许形成多级控制。即各中断源首先受 EA 控制;其次还受各中断源自己的中断允许控制位控制。

0：CPU 屏蔽所有的中断申请

1：CPU 开放中断

ELVD：低压检测中断允许位。

0：禁止低压检测中断

1：允许低压检测中断

EADC：A/D 转换中断允许位。

0：禁止 A/D 转换中断

1：允许 A/D 转换中断

ES：串行口 1 中断允许位。

0：禁止串行口 1 中断

1：允许串行口 1 中断

ET1：定时/计数器 T1 的溢出中断允许位。

0：禁止 T1 中断

1：允许 T1 中断

EX1：外部中断 1 中断允许位。

0：禁止 INT1 中断

1：允许 INT1 中断

ET0：定时/计数器 T0 的溢出中断允许位。

0：禁止 T0 中断

1：允许 T0 中断

EX0：外部中断 0 中断允许位。

0：禁止 INT0 中断

1：允许 INT0 中断

IE2（中断使能寄存器 2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IE2	AFH	EUSB	ET4	ET3	ES4	ES3	ET2	ESPI	ES2

EUSB：USB 中断允许位。

0：禁止 USB 中断

1：允许 USB 中断

ET4：定时/计数器 T4 的溢出中断允许位。

0：禁止 T4 中断

1：允许 T4 中断

ET3：定时/计数器 T3 的溢出中断允许位。

0：禁止 T3 中断

1：允许 T3 中断

ES4：串行口 4 中断允许位。

0：禁止串行口 4 中断

1：允许串行口 4 中断

ES3：串行口 3 中断允许位。

0：禁止串行口 3 中断

1：允许串行口 3 中断

ET2：定时/计数器 T2 的溢出中断允许位。

0：禁止 T2 中断

1：允许 T2 中断

ESPI：SPI 中断允许位。

0：禁止 SPI 中断

1：允许 SPI 中断

ES2：串行口 2 中断允许位。

0：禁止串行口 2 中断

1：允许串行口 2 中断

INTCLKO（外部中断与时钟输出控制寄存器）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INTCLKO	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO

EX4：外部中断 4 中断允许位。

0：禁止 INT4 中断

1：允许 INT4 中断

EX3：外部中断 3 中断允许位。

0：禁止 INT3 中断

1：允许 INT3 中断

EX2：外部中断 2 中断允许位。

0：禁止 INT2 中断

1：允许 INT2 中断

CMPCR1（比较器控制寄存器 1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR1	E6H	CMPEN	CMPIF	PIE	NIE	-	-	CMPOE	CMPRES

PIE：比较器上升沿中断允许位。

0：禁止比较器上升沿中断

1：允许比较器上升沿中断

NIE：比较器下降沿中断允许位。

0：禁止比较器下降沿中断

1：允许比较器下降沿中断

RTC 中断使能寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
RTCIEN	7EFE62H	EALAI	EDAYI	EHOURI	EMINI	ESECI	ESEC2I	ESEC8I	ESEC32I

EALAI：闹钟中断使能位

0：关闭闹钟中断

1：使能闹钟中断

EDAYI：一日（24 小时）中断使能位

0：关闭一日中断

1：使能一日中断

EHOURI：一小时（60 分钟）中断使能位

0：关闭小时中断

1：使能小时中断

EMINI：一分钟（60 秒）中断使能位

0：关闭分钟中断

1：使能分钟中断

ESECI：一秒中断使能位

0：关闭秒中断

1：使能秒中断

ESEC2I：1/2 秒中断使能位

0：关闭 1/2 秒中断

1：使能 1/2 秒中断

ESEC8I：1/8 秒中断使能位

0：关闭 1/8 秒中断

1：使能 1/8 秒中断

ESEC32I：1/32 秒中断使能位

0：关闭 1/32 秒中断

1：使能 1/32 秒中断

TFT 彩屏接口配置寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFCFG	7EFE50H	LCMIFIE	-	LCMIFIP[1:0]		LCMIFDPS[1:0]		D16_D8	M68_I80

LCMIFIE：TFT彩屏接口中断允许位。

0：禁止 TFT 彩屏接口中断

1：允许 TFT 彩屏接口中断

T11 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T11CR	7EFE78H	T11R	T11_C/T	T11CLKO	T11x12	T11CS[1:0]		ET11I	T11IF

ET11I：T11中断允许位。

0：禁止 T11 中断

1：允许 T11 中断

I2C 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSCR	7EFE81H	EMSI	-	-	-	MSCMD[3:0]			
I2CSLCR	7EFE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST

EMSI：I^2C主机模式中断允许位。

0：禁止 I^2C 主机模式中断

1：允许 I^2C 主机模式中断

ESTAI：I^2C从机接收START事件中断允许位。

0：禁止 I^2C 从机接收 START 事件中断

1：允许 I^2C 从机接收 START 事件中断

ERXI：I^2C从机接收数据完成事件中断允许位。

0：禁止 I^2C 从机接收数据完成事件中断

1：允许 I^2C 从机接收数据完成事件中断

ETXI：I^2C从机发送数据完成事件中断允许位。

0：禁止 I^2C 从机发送数据完成事件中断

1：允许 I^2C 从机发送数据完成事件中断

ESTOI：I^2C从机接收STOP事件中断允许位。

0：禁止 I^2C 从机接收 STOP 事件中断

1：允许 I^2C 从机接收 STOP 事件中断

PWMA 中断使能寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_IER	7EFEC4H	BIE	TIE	COMIE	CC4IE	CC3IE	CC2IE	CC1IE	UIE

BIE：PWMA刹车中断允许位。

0：禁止 PWMA 刹车中断

1：允许 PWMA 刹车中断

TIE：PWMA触发中断允许位。

0：禁止 PWMA 触发中断

1：允许 PWMA 触发中断

COMIE：PWMA比较中断允许位。

0：禁止 PWMA 比较中断

1：允许 PWMA 比较中断

CC4IE：PWMA捕获比较通道4中断允许位。

0：禁止 PWMA 捕获比较通道 4 中断

1：允许 PWMA 捕获比较通道 4 中断

CC3IE：PWMA捕获比较通道3中断允许位。

0：禁止 PWMA 捕获比较通道 3 中断

1：允许 PWMA 捕获比较通道 3 中断

CC2IE：PWMA捕获比较通道2中断允许位。

0：禁止 PWMA 捕获比较通道 2 中断

1：允许 PWMA 捕获比较通道 2 中断

CC1IE：PWMA捕获比较通道1中断允许位。

0：禁止 PWMA 捕获比较通道 1 中断

1：允许 PWMA 捕获比较通道 1 中断

UIE：PWMA更新中断允许位。

0：禁止 PWMA 更新中断

1：允许 PWMA 更新中断

PWMB 中断使能寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMB_IER	7EFEE4H	BIE	TIE	COMIE	CC8IE	CC7IE	CC6IE	CC5IE	UIE

BIE：PWMB刹车中断允许位。

0：禁止 PWMB 刹车中断

1：允许 PWMB 刹车中断

TIE：PWMB触发中断允许位。

0：禁止 PWMB 触发中断

1：允许 PWMB 触发中断

COMIE：PWMB比较中断允许位。

0：禁止 PWMB 比较中断

1：允许 PWMB 比较中断

CC8IE：PWMB捕获比较通道8中断允许位。

0：禁止 PWMB 捕获比较通道 8 中断

1：允许 PWMB 捕获比较通道 8 中断

CC7IE：PWMB捕获比较通道7中断允许位。

0：禁止 PWMB 捕获比较通道 7 中断

1：允许 PWMB 捕获比较通道 7 中断

CC6IE：PWMB捕获比较通道6中断允许位。

0：禁止 PWMB 捕获比较通道 6 中断

1：允许 PWMB 捕获比较通道 6 中断

CC5IE：PWMB捕获比较通道5中断允许位。

0：禁止 PWMB 捕获比较通道 5 中断

1：允许 PWMB 捕获比较通道 5 中断

UIE：PWMB更新中断允许位。

0：禁止 PWMB 更新中断

1：允许 PWMB 更新中断

端口中断使能寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0INTE	7EFD00H	P07INTE	P06INTE	P05INTE	P04INTE	P03INTE	P02INTE	P01INTE	P00INTE
P1INTE	7EFD01H	P17INTE	P16INTE	P15INTE	P14INTE	P13INTE	P12INTE	P11INTE	P10INTE
P2INTE	7EFD02H	P27INTE	P26INTE	P25INTE	P24INTE	P23INTE	P22INTE	P21INTE	P20INTE
P3INTE	7EFD03H	P37INTE	P36INTE	P35INTE	P34INTE	P33INTE	P32INTE	P31INTE	P30INTE
P4INTE	7EFD04H	P47INTE	P46INTE	P45INTE	P44INTE	P43INTE	P42INTE	P41INTE	P40INTE
P5INTE	7EFD05H	P57INTE	P56INTE	P55INTE	P54INTE	P53INTE	P52INTE	P51INTE	P50INTE
P6INTE	7EFD06H	P67INTE	P66INTE	P65INTE	P64INTE	P63INTE	P62INTE	P61INTE	P60INTE
P7INTE	7EFD07H	P77INTE	P76INTE	P75INTE	P74INTE	P73INTE	P72INTE	P71INTE	P70INTE

PnINTE.x：端口中断使能控制位（n=0~7，x=0~7）

0：关闭 Pn.x 口中断功能

1：使能 Pn.x 口中断功能

串 PCA/CCP/PWM 中断控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMOD	7EFD67H	CIDL	CCP_S[1:0]		CPS[3:0]				ECF
CCAPM0	7EFD58H	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0
CCAPM1	7EFD5CH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1
CCAPM2	7EFD50H	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2

ECF：PCA 计数器中断允许位。

0：禁止 PCA 计数器中断

1：允许 PCA 计数器中断

ECCF0：PCA 模块 0 中断允许位。

0：禁止 PCA 模块 0 中断

1：允许 PCA 模块 0 中断

ECCF1：PCA 模块 1 中断允许位。

0：禁止 PCA 模块 1 中断

1：允许 PCA 模块 1 中断

ECCF2：PCA 模块 2 中断允许位。

0：禁止 PCA 模块 2 中断

1：允许 PCA 模块 2 中断

串口 1 超时控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
UR1TOCR	7EFD70H	ENTO	ENTOI	SCALE	-	-	-	-	-

ENTOI：串口1接收超时中断允许位。

0：禁止串口 1 接收超时中断

1：允许串口 1 接收超时中断

串口 2 超时控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
UR2TOCR	7EFD74H	ENTO	ENTOI	SCALE	-	-	-	-	-

ENTOI：串口2接收超时中断允许位。

0：禁止串口 2 接收超时中断

1：允许串口 2 接收超时中断

串口 3 超时控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
UR3TOCR	7EFD78H	ENTO	ENTOI	SCALE	-	-	-	-	-

ENTOI：串口3接收超时中断允许位。

0：禁止串口 3 接收超时中断

1：允许串口 3 接收超时中断

串口 4 超时控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
UR4TOCR	7EFD7CH	ENTO	ENTOI	SCALE	-	-	-	-	-

ENTOI：串口4超时中断允许位。

0：禁止串口 4 接收超时中断

1：允许串口 4 接收超时中断

SPI 超时控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPITOCR	7EFD80H	ENTO	ENTOI	SCALE	-	-	-	-	-

ENTOI：SPI从机超时中断允许位。

0：禁止 SPI 从机超时中断

1：允许 SPI 从机超时中断

I2C 超时控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CTOCR	7EFD84H	ENTO	ENTOI	SCALE	-	-	-	-	-

ENTOI：I2C从机超时中断允许位。

0：禁止 I2C 从机超时中断

1：允许 I2C 从机超时中断

I2S 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2SCR	7EFD98H	TXEIE	RXNEIE	ERRIE	FRF	-	-	TXDMAEN	RXDMAEN

TXEIE：I2S发送缓冲区空中断允许位。

0：禁止 I2S 发送缓冲区空中断

1：允许 I2S 发送缓冲区空中断

RXNEIE：I2S接收缓冲区非空中断允许位。

0：禁止 I2S 接收缓冲区非空中断

1：允许 I2S 接收缓冲区非空中断

ERRIE：I2S错误中断允许位。

0：禁止 I2S 错误中断

1：允许 I2S 错误中断

QSPI 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
QSPICR3	7EF902H	PMM	APMS	-	-	SMIE	FTIE	TCIE	TEIE

SMIE：QSPI状态匹配中断允许位。

0：禁止 QSPI 状态匹配中断

1：允许 QSPI 状态匹配中断

FTIE：QSPI FIFO阈值中断允许位。

0：禁止 QSPI FIFO 阈值中断

1：允许 QSPI FIFO 阈值中断

TCIE：QSPI传输完成中断允许位。

0：禁止 QSPI 传输完成中断

1：允许 QSPI 传输完成中断

TEIE：QSPI传输错误中断允许位。

0：禁止 QSPI 传输错误中断

1：允许 QSPI 传输错误中断

DMA 中断使能寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_CFG	7EFA00H	M2MIE	-	TXACO	RXACO	M2MIP[1:0]		M2MPTY[1:0]	
DMA_ADC_CFG	7EFA10H	ADCIE	-	-	-	ADCMIP[1:0]		ADCPTY[1:0]	
DMA_SPI_CFG	7EFA20H	SPIIE	ACT_TX	ACT_RX	-	SPIIP[1:0]		SPIPTY[1:0]	
DMA_UR1T_CFG	7EFA30H	UR1TIE	-	-	-	UR1TIP[1:0]		UR1TPTY[1:0]	
DMA_UR1R_CFG	7EFA38H	UR1RIE	-	-	-	UR1RIP[1:0]		UR1RPTY[1:0]	
DMA_UR2T_CFG	7EFA40H	UR2TIE	-	-	-	UR2TIP[1:0]		UR2TPTY[1:0]	
DMA_UR2R_CFG	7EFA48H	UR2RIE	-	-	-	UR2RIP[1:0]		UR2RPTY[1:0]	
DMA_UR3T_CFG	7EFA50H	UR3TIE	-	-	-	UR3TIP[1:0]		UR3TPTY[1:0]	
DMA_UR3R_CFG	7EFA58H	UR3RIE	-	-	-	UR3RIP[1:0]		UR3RPTY[1:0]	
DMA_UR4R_CFG	7EFA60H	UR4TIE	-	-	-	UR4TIP[1:0]		UR4TPTY[1:0]	
DMA_UR4R_CFG	7EFA68H	UR4RIE	-	-	-	UR4RIP[1:0]		UR4RPTY[1:0]	
DMA_LCM_CFG	7EFA70H	LCMIE	-	-	-	LCMIP[1:0]		LCMPTY[1:0]	
DMA_I2CT_CFG	7EFA98H	I2CTIE	-	-	-	I2CTIP[1:0]		I2CTPTY[1:0]	
DMA_I2CR_CFG	7EFAA0H	I2CRIE	-	-	-	I2CRIP[1:0]		I2CRPTY[1:0]	
DMA_I2ST_CFG	7EFAB0H	I2STIE	-	-	-	I2STIP[1:0]		I2STPTY[1:0]	
DMA_I2SR_CFG	7EFAB8H	I2SRIE	-	-	-	I2SRIP[1:0]		I2SRPTY[1:0]	
DMA_QSPI_CFG	7EFAD0H	QSPIIE	ACT_WR	ACT_RD	-	QSPIIP[1:0]		QSPIPTY[1:0]	
DMA_PWMAT_CFG	7EF980H	PWMATIE	-	-	-	PWMATIP[1:0]		PWMATPTY[1:0]	
DMA_PWMAR_CFG	7EF990H	PWMARIE	-	-	-	PWMARIP[1:0]		PWMARPTY[1:0]	

M2MIE： DMA_M2M（存储器到存储器DMA）中断允许位。

0：禁止 DMA_M2M 中断

1：允许 DMA_M2M 中断

ADCIE：DMA_ADC（ADC DMA）中断允许位。

0：禁止 DMA_ADC 中断

1：允许 DMA_ADC 中断

SPIIE：DMA_SPI（SPI DMA）中断允许位。

0：禁止 DMA_SPI 中断

1：允许 DMA_SPI 中断

UR1TIE：DMA_UR1T（串口1发送DMA）中断允许位。

0：禁止 DMA_UR1T 中断

1：允许 DMA_UR1T 中断

UR1RIE：DMA_UR1R（串口1接收DMA）中断允许位。

0：禁止 DMA_UR1R 中断

1：允许 DMA_UR1R 中断

UR2TIE：DMA_UR2T（串口2发送DMA）中断允许位。

0：禁止 DMA_UR2T 中断

1：允许 DMA_UR2T 中断

UR2RIE：DMA_UR2R（串口2接收DMA）中断允许位。

0：禁止 DMA_UR2R 中断

1：允许 DMA_UR2R 中断

UR3TIE：DMA_UR3T（串口3发送DMA）中断允许位。

0：禁止 DMA_UR3T 中断

1：允许 DMA_UR3T 中断

UR3RIE：DMA_UR3R（串口3接收DMA）中断允许位。

0：禁止 DMA_UR3R 中断

1：允许 DMA_UR3R 中断

UR4TIE：DMA_UR4T（串口4发送DMA）中断允许位。

0：禁止 DMA_UR4T 中断

1：允许 DMA_UR4T 中断

UR4RIE：DMA_UR4R（串口4接收DMA）中断允许位。

0：禁止 DMA_UR4R 中断

1：允许 DMA_UR4R 中断

LCMIE：DMA_LCM（TFT彩屏接口DMA）中断允许位。

0：禁止 DMA_LCM 中断

1：允许 DMA_LCM 中断

I2CTIE：DMA_I2CT（I2C发送DMA）中断允许位。

0：禁止 DMA_I2CT 中断

1：允许 DMA_I2CT 中断

I2CRIE：DMA_I2CR（I2C接收DMA）中断允许位。

0：禁止 DMA_I2CR 中断

1：允许 DMA_I2CR 中断

I2STIE：DMA_I2ST（I2S发送DMA）中断允许位。

0：禁止 DMA_I2ST 中断

1：允许 DMA_I2ST 中断

I2SRIE：DMA_I2SR（I2S接收DMA）中断允许位。

0：禁止 DMA_I2SR 中断

1：允许 DMA_I2SR 中断

QSPIIE：DMA_QSPI（QSPI DMA）中断允许位。

0：禁止 DMA_QSPI 中断

1：允许 DMA_QSPI 中断

PWMATIE：DMA_PWMAT（PWMA发送DMA）中断允许位。

0：禁止 DMA_PWMAT 中断

1：允许 DMA_PWMAT 中断

PWMARIE：DMA_PWMAR（PWMA接收DMA）中断允许位。

0：禁止 DMA_PWMAR 中断

1：允许 DMA_PWMAR 中断

15.4.2 中断请求寄存器（中断标志位）

定时器控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TCON	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1：定时器1溢出中断标志。中断服务程序中，硬件自动清零。

TF0：定时器0溢出中断标志。中断服务程序中，硬件自动清零。

IE1：外部中断1中断请求标志。中断服务程序中，硬件自动清零。

IE0：外部中断0中断请求标志。中断服务程序中，硬件自动清零。

中断标志辅助寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AUXINTIF	EFH	-	INT4IF	INT3IF	INT2IF	-	T4IF	T3IF	T2IF

INT4IF：外部中断4中断请求标志。中断服务程序中，硬件自动清零。

INT3IF：外部中断3中断请求标志。中断服务程序中，硬件自动清零。

INT2IF：外部中断2中断请求标志。中断服务程序中，硬件自动清零。

T4IF：定时器4溢出中断标志。中断服务程序中，硬件自动清零。

T3IF：定时器3溢出中断标志。中断服务程序中，硬件自动清零。

T2IF：定时器2溢出中断标志。中断服务程序中，硬件自动清零。

串口控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SCON	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI
S2CON	9AH	S2SM0/FE	S2SM1	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI
S3CON	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI
S4CON	FDH	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI

TI：串口1发送完成中断请求标志。需要软件清零。

RI：串口1接收完成中断请求标志。需要软件清零。

S2TI：串口2发送完成中断请求标志。需要软件清零。

S2RI：串口2接收完成中断请求标志。需要软件清零。

S3TI：串口3发送完成中断请求标志。需要软件清零。

S3RI：串口3接收完成中断请求标志。需要软件清零。

S4TI：串口4发送完成中断请求标志。需要软件清零。

S4RI：串口4接收完成中断请求标志。需要软件清零。

电源管理寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PCON	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL

LVDF：低压检测中断请求标志。需要软件清零。

ADC 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADC_CONTR	BCH	ADC_POWER	ADC_START	ADC_FLAG	ADC_EPWMT	ADC_CHS[3:0]			

ADC_FLAG：ADC转换完成中断请求标志。需要软件清零。

SPI 状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPSTAT	CDH	SPIF	WCOL	-	-	-	-	-	-

SPIF：SPI数据传输完成中断请求标志。**需要软件写“1”清零**。

比较器控制寄存器 1

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR1	E6H	CMPEN	CMPIF	PIE	NIE	-	-	CMPOE	CMPRES

CMPIF：比较器中断请求标志。需要软件清零。

TFT 彩屏接口状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFSTA	7EFE53H	-	-	-	-	-	-	-	LCMIFIF

LCMIFIF：TFT彩屏接口中断请求标志。需要软件清零。

RTC 中断请求标志寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
RTCIF	7EFE63H	ALAIF	DAYIF	HOURIF	MINIF	SECIF	SEC2IF	SEC8IF	SEC32IF

ALAIF：闹钟中断请求位。需软件清零。

DAYIF：一日（24 小时）中断请求位。需软件清零。

HOURIF：一小时（60 分钟）中断请求位。需软件清零。

MINIF：一分钟（60 秒）中断请求位。需软件清零。

SECIF：一秒中断请求位。需软件清零。

SEC2IF：1/2 秒中断请求位。需软件清零

SEC8IF：1/8 秒中断请求位。需软件清零。

SEC32IF：1/32 秒中断请求位。需软件清零。

T11 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T11CR	7EFE78H	T11R	T11_C/T	T11CLKO	T11x12	T11CS[1:0]		ET11I	T11IF

T11IF：定时器T11溢出中断标志。中断服务程序中，硬件自动清零。

I2C 状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSST	7EFE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO
I2CSLST	7EFE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO

MSIF：I^2C主机模式中断请求标志。需要软件清零。

STAIF：I^2C从机接收START事件中断请求标志。需要软件清零。

RXIF：I^2C从机接收数据完成事件中断请求标志。需要软件清零。

TXIF：I^2C从机发送数据完成事件中断请求标志。需要软件清零。

STOIF：I^2C从机接收STOP事件中断请求标志。需要软件清零。

PWMA 状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_SR1	7EFEC5H	BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	UIF
PWMA_SR2	7EFEC6H	-	-	-	CC4OF	CC3OF	CC2OF	CC1OF	-

BIF：PWMA刹车中断请求标志。需要软件清零。

TIF：PWMA触发中断请求标志。需要软件清零。

COMIF：PWMA比较中断请求标志。需要软件清零。

CC4IF：PWMA通道4发生捕获比较中断请求标志。需要软件清零。

CC3IF：PWMA通道3发生捕获比较中断请求标志。需要软件清零。

CC2IF：PWMA通道2发生捕获比较中断请求标志。需要软件清零。

CC1IF：PWMA通道1发生捕获比较中断请求标志。需要软件清零。

UIF：PWMA更新中断请求标志。需要软件清零。

CC4OF：PWMA通道4发生重复捕获中断请求标志。需要软件清零。

CC3OF：PWMA通道3发生重复捕获中断请求标志。需要软件清零。

CC2OF：PWMA通道2发生重复捕获中断请求标志。需要软件清零。

CC1OF：PWMA通道1发生重复捕获中断请求标志。需要软件清零。

PWMB 状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMB_SR1	7EFEE5H	BIF	TIF	COMIF	CC8IF	CC7IF	CC6IF	CC5IF	UIF
PWMB_SR2	7EFEE6H	-	-	-	CC8OF	CC7OF	CC6OF	CC5OF	-

BIF：PWMB刹车中断请求标志。需要软件清零。

TIF：PWMB触发中断请求标志。需要软件清零。

COMIF：PWMB比较中断请求标志。需要软件清零。

CC8IF：PWMB通道8发生捕获比较中断请求标志。需要软件清零。

CC7IF：PWMB通道7发生捕获比较中断请求标志。需要软件清零。

CC6IF：PWMB通道6发生捕获比较中断请求标志。需要软件清零。

CC5IF：PWMB通道5发生捕获比较中断请求标志。需要软件清零。

UIF：PWMB更新中断请求标志。需要软件清零。

CC8OF：PWMB通道8发生重复捕获中断请求标志。需要软件清零。

CC7OF：PWMB通道7发生重复捕获中断请求标志。需要软件清零。

CC6OF：PWMB通道6发生重复捕获中断请求标志。需要软件清零。

CC5OF：PWMB通道5发生重复捕获中断请求标志。需要软件清零。

端口中断标志寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0INTF	7EFD10H	P07INTF	P06INTF	P05INTF	P04INTF	P03INTF	P02INTF	P01INTF	P00INTF
P1INTF	7EFD11H	P17INTF	P16INTF	P15INTF	P14INTF	P13INTF	P12INTF	P11INTF	P10INTF
P2INTF	7EFD12H	P27INTF	P26INTF	P25INTF	P24INTF	P23INTF	P22INTF	P21INTF	P20INTF
P3INTF	7EFD13H	P37INTF	P36INTF	P35INTF	P34INTF	P33INTF	P32INTF	P31INTF	P30INTF
P4INTF	7EFD14H	P47INTF	P46INTF	P45INTF	P44INTF	P43INTF	P42INTF	P41INTF	P40INTF
P5INTF	7EFD15H	-	-	P55INTF	P54INTF	P53INTF	P52INTF	P51INTF	P50INTF
P6INTF	7EFD16H	P67INTF	P66INTF	P65INTF	P64INTF	P63INTF	P62INTF	P61INTF	P60INTF
P7INTF	7EFD17H	P77INTF	P76INTF	P75INTF	P74INTF	P73INTF	P72INTF	P71INTF	P70INTF

PnINTF.x：端口中断请求标志位（n=0~7，x=0~7）

0：Pn.x 口没有中断请求

1：Pn.x 口有中断请求，若使能中断，则会进入中断服务程序。需要软件清零。

PCA 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CCON	7EFD64H	CF	CR	-	-	-	CCF2	CCF1	CCF0

CF：PCA计数器中断请求标志。需要软件清零。

CCF2：PCA模块2中断请求标志。需要软件清零。

CCF1：PCA模块1中断请求标志。需要软件清零。

CCF0：PCA模块0中断请求标志。需要软件清零。

串口 1 超时状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
UR1TOSR	7EFD71H	CTOIF	-	-	-	—	—	—	TOIF

TOIF：串口1超时中断请求标志。需要软件清零。

串口 2 超时状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
UR2TOSR	7EFD75H	CTOIF	-	-	-	—	—	—	TOIF

TOIF：串口2超时中断请求标志。需要软件清零。

串口 3 超时状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
UR3TOSR	7EFD79H	CTOIF	-	-	-	—	—	—	TOIF

TOIF：串口3超时中断请求标志。需要软件清零。

串口 4 超时状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
UR4TOSR	7EFD7DH	CTOIF	-	-	-	—	—	—	TOIF

TOIF：串口4超时中断请求标志。需要软件清零。

SPI 超时状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPITOSR	7EFD81H	CTOIF	-	-	-	—	—	—	TOIF

TOIF：SPI超时中断请求标志。需要软件清零。

I2C 超时状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CTOSR	7EFD85H	CTOIF	-	-	-	—	—	—	TOIF

TOIF：I2C超时中断请求标志。需要软件清零。

I2S 状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2SSR	7EFD99H	-	FRE	BUY	OVR	UDR	CHSID	TXE	RXNE

TXE：I2S发送缓冲区空中断中断请求标志。不需要软件清零。

RXNE：I2S接收缓冲区非空中断请求标志。不需要软件清零。

QSPI 状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
QSPI_SR1	7EF906H	-	-	BUSY	-	SMF	FTF	TCF	TEF

SMF：QSPI状态匹配中断请求标志。需要软件清零。

FTF：QSPI FIFO阈值中断请求标志。需要软件清零。

TCF：QSPI传输完成中断请求标志。需要软件清零。

TEF：QSPI传输错误中断请求标志。需要软件清零。

DMA 中断标志寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_STA	7EFA02H	-	-	-	-	-	-	-	M2MIF
DMA_ADC_STA	7EFA12H	-	-	-	-	-	-	-	ADCIF
DMA_SPI_STA	7EFA22H	-	-	-	-	-	TXOVW	RXLOSS	SPIIF
DMA_UR1T_STA	7EFA32H	-	-	-	-	-	TXOVW	-	UR1TIF
DMA_UR1R_STA	7EFA3AH	-	-	-	-	-	-	RXLOSS	UR1RIF
DMA_UR2T_STA	7EFA42H	-	-	-	-	-	TXOVW	-	UR2TIF
DMA_UR2R_STA	7EFA4AH	-	-	-	-	-	-	RXLOSS	UR2RIF
DMA_UR3T_STA	7EFA52H	-	-	-	-	-	TXOVW	-	UR3TIF
DMA_UR3R_STA	7EFA5AH	-	-	-	-	-	-	RXLOSS	UR3RIF
DMA_UR4T_STA	7EFA62H	-	-	-	-	-	TXOVW	-	UR4TIF
DMA_UR4R_STA	7EFA6AH	-	-	-	-	-	-	RXLOSS	UR4RIF
DMA_LCM_STA	7EFA72H	-	-	-	-	-	-	TXOVW	LCMIF
DMA_I2CT_STA	7EFA9AH	-	-	-	-	-	TXOVW	-	I2CTIF
DMA_I2CR_STA	7EFAA2H	-	-	-	-	-	-	RXLOSS	I2CRIF
DMA_I2ST_STA	7EFAB2H	-	-	-	-	-	TXOVW	-	I2STIF
DMA_I2SR_STA	7EFABAH	-	-	-	-	-	-	RXLOSS	I2SRIF
DMA_QSPI_STA	7EFAD2H	-	-	-	-	-	-	-	QSPIIF
DMA_PWMAT_STA	7EF982H	-	-	-	-	-	TXOVW	-	PWMATTIF
DMA_PWMAR_STA	7EF992H	-	-	-	-	-	-	RXLOSS	PWMATRIF

M2MIF： DMA_M2M（存储器到存储器DMA）中断请求标志。需要软件清零。

ADCIF：DMA_ADC（ADC DMA）中断请求标志。需要软件清零。

SPIIF：DMA_SPI（SPI DMA）中断请求标志。需要软件清零。

UR1TIF：DMA_UR1T（串口1发送DMA）中断请求标志。需要软件清零。

UR1RIF：DMA_UR1R（串口1接收DMA）中断请求标志。需要软件清零。

UR2TIF：DMA_UR2T（串口2发送DMA）中断请求标志。需要软件清零。

UR2RIF：DMA_UR2R（串口2接收DMA）中断请求标志。需要软件清零。

UR3TIF：DMA_UR3T（串口3发送DMA）中断请求标志。需要软件清零。

UR3RIF：DMA_UR3R（串口3接收DMA）中断请求标志。需要软件清零。

UR4TIF：DMA_UR4T（串口4发送DMA）中断请求标志。需要软件清零。

UR4RIF：DMA_UR4R（串口4接收DMA）中断请求标志。需要软件清零。

LCMIF：DMA_LCM（TFT彩屏接口DMA）中断请求标志。需要软件清零。

I2CTIF：DMA_I2CT（I2C发送DMA）中断请求标志。需要软件清零。

I2CRIF：DMA_I2CR（I2C接收DMA）中断请求标志。需要软件清零。

I2STIF：DMA_I2ST（I2S发送DMA）中断请求标志。需要软件清零。

I2SRIF：DMA_I2SR（I2S接收DMA）中断请求标志。需要软件清零。

QSPIIF：DMA_QSPI（QSPI DMA）中断请求标志。需要软件清零。

PWMATIF：DMA_PWMAT（PWMA发送DMA）中断请求标志。需要软件清零。

PWMARIF：DMA_PWMAR（PWMA接收DMA）中断请求标志。需要软件清零。

15.4.3 中断优先级寄存器

中断优先级控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IP	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0
IPH	B7H	PPCAH	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H
IP2	B5H	PUSB	PI2C	PCMP	PX4	PPWMB	PPWMA	PSPI	PS2
IP2H	B6H	PUSBH	PI2CH	PCMPH	PX4H	PPWMBH	PPWMAH	PSPIH	PS2H
IP3	DFH	-	-	-	-	PI2S	PRTC	PS4	PS3
IP3H	EEH	-	-	-	-	PI2SH	PRTCH	PS4H	PS3H

PX0H,PX0：外部中断0中断优先级控制位

00：INT0 中断优先级为 0 级（最低级）
01：INT0 中断优先级为 1 级（较低级）
10：INT0 中断优先级为 2 级（较高级）
11：INT0 中断优先级为 3 级（最高级）

PT0H,PT0：定时器0中断优先级控制位

00：定时器 0 中断优先级为 0 级（最低级）
01：定时器 0 中断优先级为 1 级（较低级）
10：定时器 0 中断优先级为 2 级（较高级）
11：定时器 0 中断优先级为 3 级（最高级）

PX1H,PX1：外部中断1中断优先级控制位

00：INT1 中断优先级为 0 级（最低级）
01：INT1 中断优先级为 1 级（较低级）
10：INT1 中断优先级为 2 级（较高级）
11：INT1 中断优先级为 3 级（最高级）

PT1H,PT1：定时器1中断优先级控制位

00：定时器 1 中断优先级为 0 级（最低级）
01：定时器 1 中断优先级为 1 级（较低级）
10：定时器 1 中断优先级为 2 级（较高级）
11：定时器 1 中断优先级为 3 级（最高级）

PSH,PS：串口1中断优先级控制位

00：串口 1 中断优先级为 0 级（最低级）
01：串口 1 中断优先级为 1 级（较低级）
10：串口 1 中断优先级为 2 级（较高级）
11：串口 1 中断优先级为 3 级（最高级）

PADCH,PADC：ADC中断优先级控制位

00：ADC 中断优先级为 0 级（最低级）
01：ADC 中断优先级为 1 级（较低级）
10：ADC 中断优先级为 2 级（较高级）
11：ADC 中断优先级为 3 级（最高级）

PLVDH,PLVD：低压检测中断优先级控制位

00：LVD 中断优先级为 0 级（最低级）
01：LVD 中断优先级为 1 级（较低级）

10：LVD 中断优先级为 2 级（较高级）
11：LVD 中断优先级为 3 级（最高级）

PPCAH,PPCA：CCP/PCA/PWM中断优先级控制位

00：CCP/PCA/PWM 中断优先级为 0 级（最低级）
01：CCP/PCA/PWM 中断优先级为 1 级（较低级）
10：CCP/PCA/PWM 中断优先级为 2 级（较高级）
11：CCP/PCA/PWM 中断优先级为 3 级（最高级）

PS2H,PS2：串口2中断优先级控制位

00：串口 2 中断优先级为 0 级（最低级）
01：串口 2 中断优先级为 1 级（较低级）
10：串口 2 中断优先级为 2 级（较高级）
11：串口 2 中断优先级为 3 级（最高级）

PS3H,PS3：串口3中断优先级控制位

00：串口 3 中断优先级为 0 级（最低级）
01：串口 3 中断优先级为 1 级（较低级）
10：串口 3 中断优先级为 2 级（较高级）
11：串口 3 中断优先级为 3 级（最高级）

PS4H,PS4：串口4中断优先级控制位

00：串口 4 中断优先级为 0 级（最低级）
01：串口 4 中断优先级为 1 级（较低级）
10：串口 4 中断优先级为 2 级（较高级）
11：串口 4 中断优先级为 3 级（最高级）

PSPIH,PSPI：SPI中断优先级控制位

00：SPI 中断优先级为 0 级（最低级）
01：SPI 中断优先级为 1 级（较低级）
10：SPI 中断优先级为 2 级（较高级）
11：SPI 中断优先级为 3 级（最高级）

PPWMAH,PPWMA：高级PWMA中断优先级控制位

00：高级 PWMA 中断优先级为 0 级（最低级）
01：高级 PWMA 中断优先级为 1 级（较低级）
10：高级 PWMA 中断优先级为 2 级（较高级）
11：高级 PWMA 中断优先级为 3 级（最高级）

PPWMBH,PPWMB：高级PWMB中断优先级控制位

00：高级 PWMB 中断优先级为 0 级（最低级）
01：高级 PWMB 中断优先级为 1 级（较低级）
10：高级 PWMB 中断优先级为 2 级（较高级）
11：高级 PWMB 中断优先级为 3 级（最高级）

PX4H,PX4：外部中断4中断优先级控制位

00：INT4 中断优先级为 0 级（最低级）
01：INT4 中断优先级为 1 级（较低级）
10：INT4 中断优先级为 2 级（较高级）
11：INT4 中断优先级为 3 级（最高级）

PCMPH,PCMP：比较器中断优先级控制位

00：CMP 中断优先级为 0 级（最低级）

01：CMP 中断优先级为 1 级（较低级）
10：CMP 中断优先级为 2 级（较高级）
11：CMP 中断优先级为 3 级（最高级）

PI2CH,PI2C：I2C中断优先级控制位
00：I2C 中断优先级为 0 级（最低级）
01：I2C 中断优先级为 1 级（较低级）
10：I2C 中断优先级为 2 级（较高级）
11：I2C 中断优先级为 3 级（最高级）

PUSBH,PUSB：USB中断优先级控制位
00：USB 中断优先级为 0 级（最低级）
01：USB 中断优先级为 1 级（较低级）
10：USB 中断优先级为 2 级（较高级）
11：USB 中断优先级为 3 级（最高级）

PRTCH,PRTC：RTC中断优先级控制位
00：RTC 中断优先级为 0 级（最低级）
01：RTC 中断优先级为 1 级（较低级）
10：RTC 中断优先级为 2 级（较高级）
11：RTC 中断优先级为 3 级（最高级）

PI2SH,PI2S：I2S中断优先级控制位
00：I2S 中断优先级为 0 级（最低级）
01：I2S 中断优先级为 1 级（较低级）
10：I2S 中断优先级为 2 级（较高级）
11：I2S 中断优先级为 3 级（最高级）

TFT 彩屏接口配置寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFCFG	7EFE50H	LCMIFIE	-	LCMIFIP[1:0]		LCMIFDPS[1:0]		D16_D8	M68_I80

LCMIFIP[1:0]：TFT彩屏接口中断优先级控制位
00：TFT 彩屏接口中断优先级为 0 级（最低级）
01：TFT 彩屏接口中断优先级为 1 级（较低级）
10：TFT 彩屏接口中断优先级为 2 级（较高级）
11：TFT 彩屏接口中断优先级为 3 级（最高级）

端口中断优先级控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PINIPL	7EFD60H	P7IP	P6IP	P5IP	P4IP	P3IP	P2IP	P1IP	P0IP
PINIPH	7EFD61H	P7IPH	P6IPH	P5IPH	P4IPH	P3IPH	P2IPH	P1IPH	P0IPH

P0IPH,P0IP：P0口中断优先级控制位
00：P0 口中断优先级为 0 级（最低级）
01：P0 口中断优先级为 1 级（较低级）
10：P0 口中断优先级为 2 级（较高级）
11：P0 口中断优先级为 3 级（最高级）

P1IPH,P1IP：P1口中断优先级控制位

00：P1 口中断优先级为 0 级（最低级）
01：P1 口中断优先级为 1 级（较低级）
10：P1 口中断优先级为 2 级（较高级）
11：P1 口中断优先级为 3 级（最高级）

P2IPH,P2IP：P2口中断优先级控制位

00：P2 口中断优先级为 0 级（最低级）
01：P2 口中断优先级为 1 级（较低级）
10：P2 口中断优先级为 2 级（较高级）
11：P2 口中断优先级为 3 级（最高级）

P3IPH,P3IP：P3口中断优先级控制位

00：P3 口中断优先级为 0 级（最低级）
01：P3 口中断优先级为 1 级（较低级）
10：P3 口中断优先级为 2 级（较高级）
11：P3 口中断优先级为 3 级（最高级）

P4IPH,P4IP：P4口中断优先级控制位

00：P4 口中断优先级为 0 级（最低级）
01：P4 口中断优先级为 1 级（较低级）
10：P4 口中断优先级为 2 级（较高级）
11：P4 口中断优先级为 3 级（最高级）

P5IPH,P5IP：P5口中断优先级控制位

00：P5 口中断优先级为 0 级（最低级）
01：P5 口中断优先级为 1 级（较低级）
10：P5 口中断优先级为 2 级（较高级）
11：P5 口中断优先级为 3 级（最高级）

P6IPH,P6IP：P6口中断优先级控制位

00：P6 口中断优先级为 0 级（最低级）
01：P6 口中断优先级为 1 级（较低级）
10：P6 口中断优先级为 2 级（较高级）
11：P6 口中断优先级为 3 级（最高级）

P7IPH,P7IP：P7口中断优先级控制位

00：P7 口中断优先级为 0 级（最低级）
01：P7 口中断优先级为 1 级（较低级）
10：P7 口中断优先级为 2 级（较高级）
11：P7 口中断优先级为 3 级（最高级）

DMA 中断优先级控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_CFG	7EFA00H	M2MIE	-	TXACO	RXACO	M2MIP[1:0]		M2MPTY[1:0]	
DMA_ADC_CFG	7EFA10H	ADCIE	-	-	-	ADCMIP[1:0]		ADCPTY[1:0]	
DMA_SPI_CFG	7EFA20H	SPIIE	ACT_TX	ACT_RX	-	SPIIP[1:0]		SPIPTY[1:0]	
DMA_UR1T_CFG	7EFA30H	UR1TIE	-	-	-	UR1TIP[1:0]		UR1TPTY[1:0]	
DMA_UR1R_CFG	7EFA38H	UR1RIE	-	-	-	UR1RIP[1:0]		UR1RPTY[1:0]	
DMA_UR2T_CFG	7EFA40H	UR2TIE	-	-	-	UR2TIP[1:0]		UR2TPTY[1:0]	
DMA_UR2R_CFG	7EFA48H	UR2RIE	-	-	-	UR2RIP[1:0]		UR2RPTY[1:0]	
DMA_UR3T_CFG	7EFA50H	UR3TIE	-	-	-	UR3TIP[1:0]		UR3TPTY[1:0]	
DMA_UR3R_CFG	7EFA58H	UR3RIE	-	-	-	UR3RIP[1:0]		UR3RPTY[1:0]	
DMA_UR4T_CFG	7EFA60H	UR4TIE	-	-	-	UR4TIP[1:0]		UR4TPTY[1:0]	
DMA_UR4R_CFG	7EFA68H	UR4RIE	-	-	-	UR4RIP[1:0]		UR4RPTY[1:0]	
DMA_LCM_CFG	7EFA70H	LCMIE	-	-	-	LCMIP[1:0]		LCMPTY[1:0]	
DMA_I2CT_CFG	7EFA98H	I2CTIE	-	-	-	I2CTIP[1:0]		I2CTPTY[1:0]	
DMA_I2CR_CFG	7EFAA0H	I2CRIE	-	-	-	I2CRIP[1:0]		I2CRPTY[1:0]	
DMA_I2ST_CFG	7EFAB0H	I2STIE	-	-	-	I2STIP[1:0]		I2STPTY[1:0]	
DMA_I2SR_CFG	7EFAB8H	I2SRIE	-	-	-	I2SRIP[1:0]		I2SRPTY[1:0]	
DMA_QSPI_CFG	7EFAD0H	QSPIIE	ACT_WR	ACT_RD	-	QSPIIP[1:0]		QSPIPTY[1:0]	
DMA_PWMAT_CFG	7EF980H	PWMATIE	-	-	-	PWMATIP[1:0]		PWMATPTY[1:0]	
DMA_PWMAR_CFG	7EF990H	PWMARIE	-	-	-	PWMARIP[1:0]		PWMARPTY[1:0]	

M2MIP： DMA_M2M（存储器到存储器DMA）中断优先级控制位

00：DMA_M2M 中断优先级为 0 级（最低级）
01：DMA_M2M 中断优先级为 1 级（较低级）
10：DMA_M2M 中断优先级为 2 级（较高级）
11：DMA_M2M 中断优先级为 3 级（最高级）

ADCIP：DMA_ADC（ADC DMA）中断优先级控制位

00：DMA_ADC 中断优先级为 0 级（最低级）
01：DMA_ADC 中断优先级为 1 级（较低级）
10：DMA_ADC 中断优先级为 2 级（较高级）
11：DMA_ADC 中断优先级为 3 级（最高级）

SPIIP：DMA_SPI（SPI DMA）中断优先级控制位

00：DMA_SPI 中断优先级为 0 级（最低级）
01：DMA_SPI 中断优先级为 1 级（较低级）
10：DMA_SPI 中断优先级为 2 级（较高级）
11：DMA_SPI 中断优先级为 3 级（最高级）

UR1TIP：DMA_UR1T（串口1发送DMA）中断优先级控制位

00：DMA_UR1T 中断优先级为 0 级（最低级）
01：DMA_UR1T 中断优先级为 1 级（较低级）
10：DMA_UR1T 中断优先级为 2 级（较高级）
11：DMA_UR1T 中断优先级为 3 级（最高级）

UR1RIP：DMA_UR1R（串口1接收DMA）中断优先级控制位

00：DMA_UR1R 中断优先级为 0 级（最低级）

01：DMA_UR1R 中断优先级为 1 级（较低级）
10：DMA_UR1R 中断优先级为 2 级（较高级）
11：DMA_UR1R 中断优先级为 3 级（最高级）

UR2TIP：DMA_UR2T（串口2发送DMA）中断优先级控制位

00：DMA_UR2T 中断优先级为 0 级（最低级）
01：DMA_UR2T 中断优先级为 1 级（较低级）
10：DMA_UR2T 中断优先级为 2 级（较高级）
11：DMA_UR2T 中断优先级为 3 级（最高级）

UR2RIP：DMA_UR2R（串口2接收DMA）中断优先级控制位

00：DMA_UR2R 中断优先级为 0 级（最低级）
01：DMA_UR2R 中断优先级为 1 级（较低级）
10：DMA_UR2R 中断优先级为 2 级（较高级）
11：DMA_UR2R 中断优先级为 3 级（最高级）

UR3TIP：DMA_UR3T（串口3发送DMA）中断优先级控制位

00：DMA_UR3T 中断优先级为 0 级（最低级）
01：DMA_UR3T 中断优先级为 1 级（较低级）
10：DMA_UR3T 中断优先级为 2 级（较高级）
11：DMA_UR3T 中断优先级为 3 级（最高级）

UR3RIP：DMA_UR3R（串口3接收DMA）中断优先级控制位

00：DMA_UR3R 中断优先级为 0 级（最低级）
01：DMA_UR3R 中断优先级为 1 级（较低级）
10：DMA_UR3R 中断优先级为 2 级（较高级）
11：DMA_UR3R 中断优先级为 3 级（最高级）

UR4TIP：DMA_UR4T（串口4发送DMA）中断优先级控制位

00：DMA_UR4T 中断优先级为 0 级（最低级）
01：DMA_UR4T 中断优先级为 1 级（较低级）
10：DMA_UR4T 中断优先级为 2 级（较高级）
11：DMA_UR4T 中断优先级为 3 级（最高级）

UR4RIP：DMA_UR4R（串口4接收DMA）中断优先级控制位

00：DMA_UR4R 中断优先级为 0 级（最低级）
01：DMA_UR4R 中断优先级为 1 级（较低级）
10：DMA_UR4R 中断优先级为 2 级（较高级）
11：DMA_UR4R 中断优先级为 3 级（最高级）

LCMIP：DMA_LCM（TFT彩屏接口DMA）中断优先级控制位

00：DMA_LCM 中断优先级为 0 级（最低级）
01：DMA_LCM 中断优先级为 1 级（较低级）
10：DMA_LCM 中断优先级为 2 级（较高级）
11：DMA_LCM 中断优先级为 3 级（最高级）

I2CTIP：DMA_I2CT（I2C发送DMA）中断优先级控制位

00：DMA_I2CT 中断优先级为 0 级（最低级）
01：DMA_I2CT 中断优先级为 1 级（较低级）
10：DMA_I2CT 中断优先级为 2 级（较高级）
11：DMA_I2CT 中断优先级为 3 级（最高级）

I2CRIP：DMA_I2CR（I2C接收DMA）中断优先级控制位

00：DMA_I2CR 中断优先级为 0 级（最低级）
01：DMA_I2CR 中断优先级为 1 级（较低级）
10：DMA_I2CR 中断优先级为 2 级（较高级）
11：DMA_I2CR 中断优先级为 3 级（最高级）

I2STIP：DMA_I2ST（I2S发送DMA）中断优先级控制位

00：DMA_I2ST 中断优先级为 0 级（最低级）
01：DMA_I2ST 中断优先级为 1 级（较低级）
10：DMA_I2ST 中断优先级为 2 级（较高级）
11：DMA_I2ST 中断优先级为 3 级（最高级）

I2SRIP：DMA_I2SR（I2S接收DMA）中断优先级控制位

00：DMA_I2SR 中断优先级为 0 级（最低级）
01：DMA_I2SR 中断优先级为 1 级（较低级）
10：DMA_I2SR 中断优先级为 2 级（较高级）
11：DMA_I2SR 中断优先级为 3 级（最高级）

QSPIIP：DMA_QSPI（QSPI DMA）中断优先级控制位

00：DMA_QSPI 中断优先级为 0 级（最低级）
01：DMA_QSPI 中断优先级为 1 级（较低级）
10：DMA_QSPI 中断优先级为 2 级（较高级）
11：DMA_QSPI 中断优先级为 3 级（最高级）

PWMATIP：DMA_PWMAT（PWMA发送DMA）中断优先级控制位

00：DMA_PWMAT 中断优先级为 0 级（最低级）
01：DMA_PWMAT 中断优先级为 1 级（较低级）
10：DMA_PWMAT 中断优先级为 2 级（较高级）
11：DMA_PWMAT 中断优先级为 3 级（最高级）

PWMARIP：DMA_PWMAR（PWMA接收DMA）中断优先级控制位

00：DMA_PWMAR 中断优先级为 0 级（最低级）
01：DMA_PWMAR 中断优先级为 1 级（较低级）
10：DMA_PWMAR 中断优先级为 2 级（较高级）
11：DMA_PWMAR 中断优先级为 3 级（最高级）

QSPI 器件配置寄存器 1

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
QSPI_DCR1	7EF904H	-	CSHT[2:0]			QSPI_IP[1:0]		-	CKMODE

QSPI_IP[1:0]：QSPI中断优先级控制位

00：QSPI 中断优先级为 0 级（最低级）
01：QSPI 中断优先级为 1 级（较低级）
10：QSPI 中断优先级为 2 级（较高级）
11：QSPI 中断优先级为 3 级（最高级）

15.5　范例程序

15.5.1　INT0 中断（上升沿和下降沿），可同时支持上升沿和下降沿

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void INT0_Isr() interrupt 0
{
    if (P32)                                            //判断上升沿和下降沿
    {
        P10 = !P10;                                     //测试端口
    }
    else
    {
        P11 = !P11;                                     //测试端口
    }
}

void main()
{
    EAXFR = 1;                                          //允许访问扩展的特殊寄存器，XFR
                                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                      //允许访问扩展的特殊寄存器，XFR
                                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                           //设置取程序代码等待时间，
                                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                          //设置访问片内的 xdata 速度，
                                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IT0 = 0;                                            //使能 INT0 上升沿和下降沿中断
    EX0 = 1;                                            //使能 INT0 中断
    EA = 1;

    while (1);
}
```

15.5.2　INT0 中断（下降沿）

```
//测试工作频率为11.0592MHz

#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void INT0_Isr() interrupt 0
{
    P10 = !P10;                                         //测试端口
}

void main()
{
    EAXFR = 1;                                          //允许访问扩展的特殊寄存器，XFR
                                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                      //允许访问扩展的特殊寄存器，XFR
                                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                           //设置取程序代码等待时间，
                                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                          //设置访问片内的 xdata 速度，
                                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IT0 = 1;                                            //使能 INT0 下降沿中断
    EX0 = 1;                                            //使能 INT0 中断
    EA = 1;

    while (1);
}
```

15.5.3　INT1 中断（上升沿和下降沿），可同时支持上升沿和下降沿

```
//测试工作频率为11.0592MHz

#include “Ai8051U.H”                                    //头文件见下载软件
```

```
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void INT1_Isr() interrupt 2
{
    if (P33)                                            //判断上升沿和下降沿
    {
        P10 = !P10;                                     //测试端口
    }
    else
    {
        P11 = !P11;                                     //测试端口
    }
}

void main()
{
    EAXFR = 1;                                          //允许访问扩展的特殊寄存器，XFR
                                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                      //允许访问扩展的特殊寄存器，XFR
                                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                           //设置取程序代码等待时间，
                                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                          //设置访问片内的 xdata 速度，
                                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IT1 = 0;                                            //使能 INT1 上升沿和下降沿中断
    EX1 = 1;                                            //使能 INT1 中断
    EA = 1;

    while (1);
}
```

15.5.4　INT1 中断（下降沿）

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                   //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void INT1_Isr() interrupt 2
```

```
{
    P10 = !P10;                                 //测试端口
}

void main()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IT1 = 1;                                    //使能 INT1 下降沿中断
    EX1 = 1;                                    //使能 INT1 中断
    EA = 1;

    while (1);
}
```

15.5.5　INT2 中断（下降沿），只支持下降沿中断

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                           //头文件见下载软件
#include "intrins.h"                            //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void INT2_Isr() interrupt 10
{
    P10 = !P10;                                 //测试端口
}

void main()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
```

```
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    EX2 = 1;                            //使能 INT2 中断
    EA = 1;

    while (1);
}
```

15.5.6　INT3 中断（下降沿），只支持下降沿中断

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                    //头文件见下载软件
#include "intrins.h"                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void INT3_Isr() interrupt 11
{
    P10 = !P10;                         //测试端口
}

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                           //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
```

```
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    EX3 = 1;                                    //使能 INT3 中断
    EA = 1;

    while (1);
}
```

15.5.7 INT4 中断（下降沿），只支持下降沿中断

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                            //头文件见下载软件
#include "intrins.h"                            //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void INT4_Isr() interrupt 16
{
    P10 = !P10;                                 //测试端口
}

void main()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    EX4 = 1;                                    //使能 INT4 中断
    EA = 1;

    while (1);
}
```

15.5.8　定时器 0 中断

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件
#include "intrins.h"                          //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void TM0_Isr() interrupt 1
{
    P10 = !P10;                               //测试端口
}

void main()
{
    EAXFR = 1;                                //允许访问扩展的特殊寄存器，XFR
                                              //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                            //允许访问扩展的特殊寄存器，XFR
                                              //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                 //设置取程序代码等待时间，
                                              //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                //设置访问片内的 xdata 速度，
                                              //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x00;
    TL0 = 0x66;                               //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                                  //启动定时器
    ET0 = 1;                                  //使能定时器中断
    EA = 1;

    while (1);
}
```

15.5.9　定时器 1 中断

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件
#include "intrins.h"                          //使用_nop_()函数所必须要包含的头文件,
```

```
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void TM1_Isr() interrupt 3
{
    P10 = !P10;                                 //测试端口
}

void main()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x00;
    TL1 = 0x66;                                 //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                                    //启动定时器
    ET1 = 1;                                    //使能定时器中断
    EA = 1;

    while (1);
}
```

15.5.10　定时器 2 中断

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                            //头文件见下载软件
#include "intrins.h"                            //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void TM2_Isr() interrupt 12
{
    P10 = !P10;                                 //测试端口
}

void main()
```

```
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T2L = 0x66;                                 //65536-11.0592M/12/1000
    T2H = 0xfc;
    T2R = 1;                                    //启动定时器
    ET2 = 1;                                    //使能定时器中断
    EA = 1;

    while (1);
}
```

15.5.11　定时器 3 中断

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                           //头文件见下载软件
#include "intrins.h"                            //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void TM3_Isr() interrupt 19
{
    P10 = !P10;                 //测试端口
}

void main()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间
```

```
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T3L = 0x66;                 //65536-11.0592M/12/1000
    T3H = 0xfc;
    T3R = 1;                    //启动定时器
    ET3 = 1;                    //使能定时器中断
    EA = 1;

    while (1);
}
```

15.5.12　定时器 4 中断

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                  //头文件见下载软件
#include "intrins.h"                                  //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void TM4_Isr() interrupt 20
{
    P10 = !P10;                 //测试端口
}

void main()
{
    EAXFR = 1;                                        //允许访问扩展的特殊寄存器，XFR
                                                      //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                    //允许访问扩展的特殊寄存器，XFR
                                                      //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                         //设置取程序代码等待时间，
                                                      //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                        //设置访问片内的 xdata 速度，
                                                      //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
```

```
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T4L = 0x66;                 //65536-11.0592M/12/1000
    T4H = 0xfc;
    T4R = 1;                    //启动定时器
    ET4 = 1;                    //使能定时器中断
    EA = 1;

    while (1);
}
```

15.5.13　UART1 中断

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                  //头文件见下载软件
#include "intrins.h"                                  //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void UART1_Isr() interrupt 4
{
    if (TI)
    {
        TI = 0;                                       //清中断标志
        P10 = !P10;                                   //测试端口
    }
    if (RI)
    {
        RI = 0;                                       //清中断标志
        P11 = !P11;                                   //测试端口
    }
}

void main()
{
    EAXFR = 1;                                        //允许访问扩展的特殊寄存器，XFR
                                                      //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                    //允许访问扩展的特殊寄存器，XFR
                                                      //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                         //设置取程序代码等待时间，
                                                      //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                        //设置访问片内的 xdata 速度，
                                                      //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
```

```
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    SCON = 0x50;
    T2L = 0xe8;                         //65536-11059200/115200/4=0FFE8H
    T2H = 0xff;
    S1BRT = 1;
    T2x12 = 1;
    T2R = 1;                            //启动定时器
    ES = 1;                             //使能串口中断
    EA = 1;
    SBUF = 0x5a;                        //发送测试数据

    while (1);
}
```

15.5.14　UART2 中断

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                    //头文件见下载软件
#include "intrins.h"                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void UART2_Isr() interrupt 8
{
    if (S2TI)
    {
        S2TI = 0;                       //清中断标志
        P12 = !P12;                     //测试端口
    }
    if (S2RI)
    {
        S2RI = 0;                       //清中断标志
        P13 = !P13;                     //测试端口
    }
}

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                           //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
```

```
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    S2CON = 0x50;
    T2L = 0xe8;                         //65536-11059200/115200/4=0FFE8H
    T2H = 0xff;
    T2x12 = 1; T2R = 1;                 //启动定时器
    ES2 = 1;                            //使能串口中断
    EA = 1;
    S2BUF = 0x5a;                       //发送测试数据

    while (1);
}
```

15.5.15　UART3 中断

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                    //头文件见下载软件
#include "intrins.h"                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void UART3_Isr() interrupt 17
{
    if (S3TI)
    {
        S3TI = 0;                       //清中断标志
        P12 = !P12;                     //测试端口
    }
    if (S3RI)
    {
        S3RI = 0;                       //清中断标志
        P13 = !P13;                     //测试端口
    }
}

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                           //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
```

```
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    S3CON = 0x10;
    T2L = 0xe8;                         //65536-11059200/115200/4=0FFE8H
    T2H = 0xff;
    T2x12 = 1; T2R = 1;                 //启动定时器
    ES3 = 1;                            //使能串口中断
    EA = 1;
    S3BUF = 0x5a;                       //发送测试数据

    while (1);
}
```

15.5.16　UART4 中断

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                   //头文件见下载软件
#include "intrins.h"                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void UART4_Isr() interrupt 18
{
    if (S4TI)
    {
        S4TI = 0;                       //清中断标志
        P12 = !P12;                     //测试端口
    }
    if (S4RI)
    {
        S4RI = 0;                       //清中断标志
        P13 = !P13;                     //测试端口
    }
}

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                           //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
```

```
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    S4CON = 0x10;
    T2L = 0xe8;                         //65536-11059200/115200/4=0FFE8H
    T2H = 0xff;
    T2x12 = 1; T2R = 1;                 //启动定时器
    ES4 = 1;                            //使能串口中断
    EA = 1;
    S4BUF = 0x5a;                       //发送测试数据

    while (1);
}
```

15.5.17　ADC 中断

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                   //头文件见下载软件
#include "intrins.h"                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void ADC_Isr() interrupt 5
{
    ADC_FLAG = 0;                       //清中断标志
    P0 = ADC_RES;                       //测试端口
    P2 = ADC_RESL;                      //测试端口
}

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                           //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
```

```
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    ADCCFG = 0x00;
    ADC_CONTR = 0xc0;                   //使能并启动 ADC 模块
    EADC = 1;                           //使能 ADC 中断
    EA = 1;

    while (1);
}
```

15.5.18　LVD 中断

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                    //头文件见下载软件
#include "intrins.h"                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   ENLVR       0x40              //RSTCFG.6
#define   LVD2V2      0x00              //LVD@2.2V
#define   LVD2V4      0x01              //LVD@2.4V
#define   LVD2V7      0x02              //LVD@2.7V
#define   LVD3V0      0x03              //LVD@3.0V

void LVD_Isr() interrupt 6
{
    LVDF = 0;                           //清中断标志
    P10 = !P10;                         //测试端口
}

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                           //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
```

```
    P5M0 = 0x00;
    P5M1 = 0x00;

    LVDF = 0;                                   //上电需要清中断标志
    RSTCFG = LVD3V0;                            //设置 LVD 电压为 3.0V
    ELVD = 1;                                   //使能 LVD 中断
    EA = 1;

    while (1);
}
```

15.5.19　比较器中断

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                            //头文件见下载软件
#include "intrins.h"                            //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void CMP_Isr() interrupt 21
{
    CMPIF = 0;                                  //清中断标志
    P10 = !P10;                                 //测试端口
}

void main()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CMPCR2 = 0x00;
    CMPEN = 1;                                  //使能比较器模块
    PIE = NIE = 1;                              //使能比较器边沿中断
    EA = 1;

    while (1);
```

```
}
```

15.5.20　SPI 中断

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void SPI_Isr() interrupt 9
{
    SPIF = 1;                                           //清中断标志
    P10 = !P10;                                         //测试端口
}

void main()
{
    EAXFR = 1;                                          //允许访问扩展的特殊寄存器，XFR
                                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                      //允许访问扩展的特殊寄存器，XFR
                                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                           //设置取程序代码等待时间，
                                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                          //设置访问片内的 xdata 速度，
                                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    SPCTL = 0x50;                                       //使能 SPI 主机模式
    SPSTAT = 0xc0;                                      //清中断标志
    ESPI = 1;                                           //使能 SPI 中断
    EA = 1;
    SPDAT = 0x5a;                                       //发送测试数据

    while (1);
}
```

15.5.21　I2C 中断

```
//测试工作频率为11.0592MHz

#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void I2C_Isr() interrupt 24
{
    if (I2CMSST & 0x40)
    {
        I2CMSST &= ~0x40;                               //清中断标志
        P10 = !P10;                                     //测试端口
    }
}

void main()
{
    EAXFR = 1;                                          //允许访问扩展的特殊寄存器，XFR
                                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                      //允许访问扩展的特殊寄存器，XFR
                                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                           //设置取程序代码等待时间，
                                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                          //设置访问片内的 xdata 速度，
                                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    I2CCFG = 0xc0;                                      //使能 I2C 主机模式
    I2CMSCR = 0x80;                                     //使能 I2C 中断;

    EA = 1;

    I2CMSCR = 0x81;                                     //发送起始命令

    while (1);
}
```

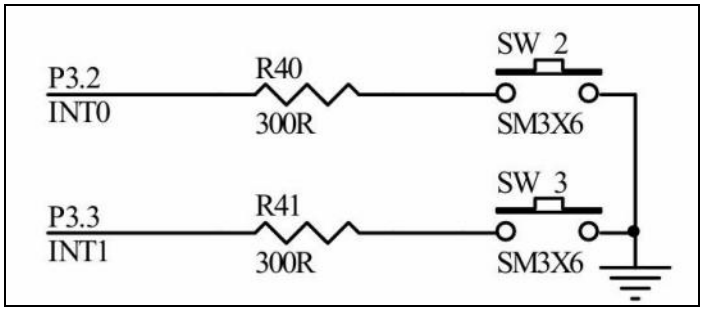
15.6　深圳大学--传统外部中断实验：INT0，INT1，INT2，INT3

15.6.1　深大--中断实验 1(外部中断 INT0/P3.2，INT1/P3.3 控制点灯并 USB 仿真)

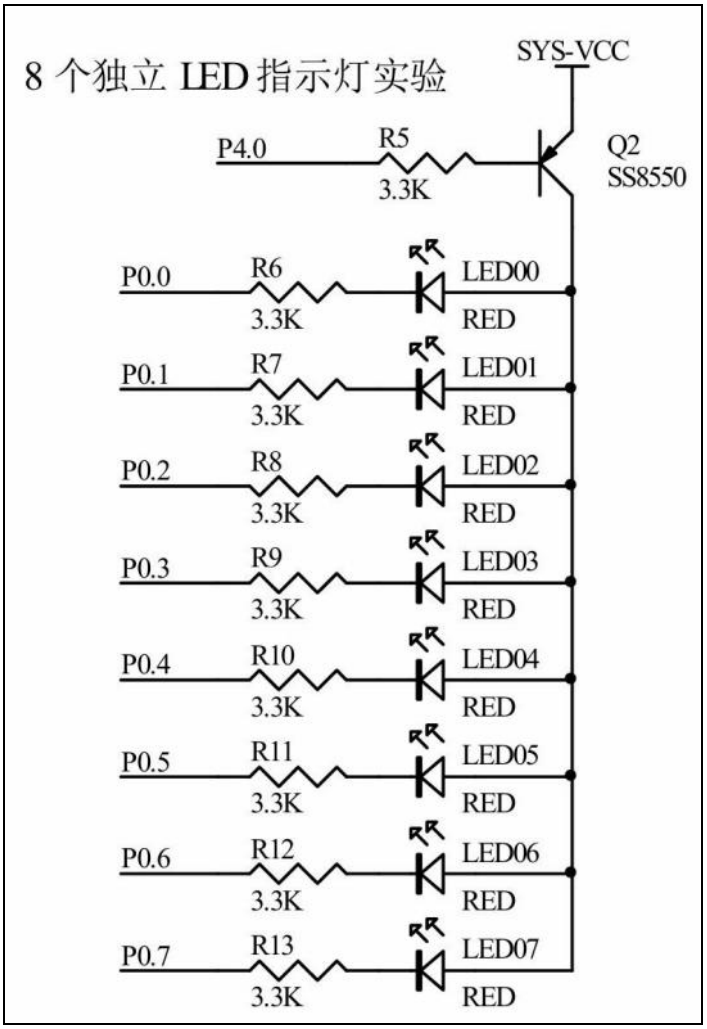
a)　原理图

- **外部中断【INT0/P3.2，INT1/P3.3】输入原理图@Ai8051U 实验箱**

  将[INT0/P3.2，INT1/P3.3]设置为：高阻输入，打开内部上拉电阻
  或　　准双向口，打开内部上拉电阻
  或　　开漏模式，打开内部上拉电阻
  [INT0/P3.2，INT1/P3.3]设置为边缘触发中断。



- **外部中断实验辅助显示 LED 灯原理图**



I/O 口 P4.0 控制 PNP 三极管的通断，控制这 8 个灯的整体供电。
如果 P4.0 输出 1，则 PNP 三极管不导通，这 8 个灯都不会获得电源，都不会有电流流过，都不会被点亮。
如果 P4.0 输出 0，则 PNP 三极管导通，这 8 个灯才有可能会流过电流，才有可能会被点亮。
如 P4.0 输出 0，PNP 三极管导通，特定 LED 灯的控制 I/O 口如输出 0，则该 LED 灯会有电流流过，会被点亮。
如 P0.0 输出 0，LED00 灯就会被点亮；如 P0.0 输出 1，LED00 灯就会熄灭；
如 P0.1 输出 0，LED01 灯就会被点亮；如 P0.1 输出 1，LED01 灯就会熄灭；
如 P0.2 输出 0，LED02 灯就会被点亮；如 P0.2 输出 1，LED02 灯就会熄灭；
如 P0.3 输出 0，LED03 灯就会被点亮；如 P0.3 输出 1，LED03 灯就会熄灭；
如 P0.4 输出 0，LED03 灯就会被点亮；如 P0.4 输出 1，LED03 灯就会熄灭；
如 P0.5 输出 0，LED03 灯就会被点亮；如 P0.5 输出 1，LED03 灯就会熄灭；
如 P0.6 输出 0，LED03 灯就会被点亮；如 P0.6 输出 1，LED03 灯就会熄灭；
如 P0.7 输出 0，LED03 灯就会被点亮；如 P0.7 输出 1，LED03 灯就会熄灭；

如 P0 口对外全部送 0，则这 8 个灯全亮；
如 P0 口对外全部送 1，则这 8 个灯全灭；
如果 P4.0 输出 1，则 PNP 三极管不导通，不管 P0 口对外全部送 0 或送 1，这 8 个灯都不会有电流流过，都不会亮。P4.0 控制 PNP 三极管，作为这 8 个 LED 指示灯的总电源开关。

b) 实验介绍

1，边缘触发中断，INT0/P3.2 外部中断实验演示程序：
　　P3.2 设置为高阻输入，打开内部上拉电阻
　或　P3.2 设置为准双向口，打开内部上拉电阻
　　INT0/P3.2 设置为边缘触发中断。
【实验步骤】：
- 先点击 Run（运行）按钮，按下 P3.2 按键，保持不松开（为了验证下一步的实验），
  进 INT0 中断后，读到 INT0/P3.2 口是 低，下降沿，点亮 P0.7，退出中断服务程序。
- 点击 Run（运行）按钮，松开 P3.2 按键
  进 INT0 中断后，读到 INT0/P3.2 口是 高，上升沿，点亮 P0.0，退出中断服务程序。

利用仿真，可以设置运行到断点处，捕捉当前程序运行到哪，是下降沿，还是上升沿

2，下降沿触发中断，INT1/P3.3 外部中断实验演示程序：
　　P3.3 设置为高阻输入，打开内部上拉电阻
　或　P3.3 设置为准双向口，打开内部上拉电阻
　　INT1/P3.3 设置为下降沿触发中断。
【实验步骤】：
- 先点击 Run（运行）按钮，按一下 P3.3 按键
  进 INT1 中断后，点亮 P0.4，退出中断服务程序。

利用仿真，可以设置运行到断点处，捕捉当前程序运行到哪。

c) 实验程序 C 语言代码

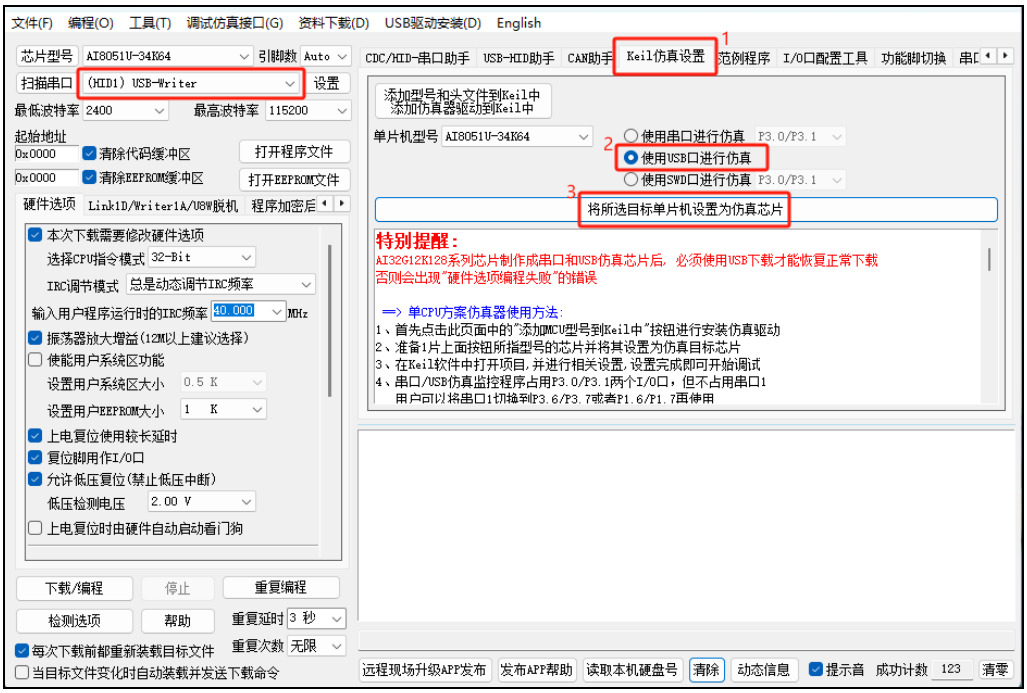
```
#include <AI8051U.H>            //包含 AI8051U 的头文件
#include "intrins.h"            //使用_nop_()函数所必须要包含的头文件,
                                //否则延时函数中调用的_nop_()函数没有被头文件引用过来，
                                //会导致编译器找不到这个而函数而报错。
bit   int0_flag = 0;  //定义 1 个位变量，INT0 事件位变量标志，记录 INT0 已产生中断
//  供主循环查询 INT0 是否已产生中断，在主循环中处理 INT0 的中断事件任务，不堵塞其他中断
bit   int1_flag = 0;  //定义 1 个位变量，INT1 事件位变量标志，记录 INT1 已产生中断
//  供主循环查询 INT1 是否已产生中断，在主循环中处理 INT1 的中断事件任务，不堵塞其他中断

void main (void)
{
    EAXFR = 1;                  //允许访问扩展的特殊寄存器，XFR
    WTST = 0;                   //设置取程序代码等待时间，赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                  //设置访问片内的 xdata 速度，赋值为 0 表示用最快速度访问，
                                //不增加额外的等待时间
    P0M0 = 0x00; P0M1 = 0x00;
    P1M0 = 0x00; P1M1 = 0x00;
    P2M0 = 0x00; P2M1 = 0x00;
//  P3M0 = 0x00; P3M1 = 0x00;       //P32、P33 设置为准双向口
    P3M0 = 0x00; P3M1 = 0xff;       //P32、P33 设置为高阻输入(需要同步开启上拉电阻)
    P4M0 = 0x00; P4M1 = 0x00;
    P5M0 = 0x00; P5M1 = 0x00;
    P6M0 = 0x00; P6M1 = 0x00;
    P7M0 = 0x00; P7M1 = 0x00;
    P3PU = 0x0c;                    //P32、P33 打开上拉电阻

    IT0 = 0;                        //使能 INT0 上升沿和下降沿中断
//  IT0 = 1;                        //使能 INT0 下降沿中断
    EX0 = 1;                        //使能 INT0 中断

//  IT1 = 0;                        //使能 INT1 上升沿和下降沿中断
    IT1 = 1;                        //使能 INT1 下降沿中断
    EX1 = 1;                        //使能 INT1 中断

    IE0 = 0;                        //清 INT0 中断标志
    IE1 = 0;                        //清 INT1 中断标志

    EA = 1;                         //总中断允许位打开
    P40 = 0;                        //打开 LED 灯供电

    int0_flag = 0;                  //初始化用户标志位
    int1_flag = 0;                  //初始化用户标志位
```

```
    while(1)
    {
        if(int0_flag)   //主循环中查询，INT0 是否已产生中断，是否有需要处理的 INT 0 事件
        {
            int0_flag = 0;      //清 0，INT0 事件位变量标志
            _nop_();            //用户在此添加需要处理的事件
            _nop_();
        }
        if(int1_flag)   //主循环中查询，INT1 是否已产生中断，是否有需要处理的 INT1 事件
        {
            int1_flag = 0;      //清 0，INT1 事件位变量标志
            _nop_();            //用户在此添加需要处理的事件
            _nop_();
        }
    }
}

void int0_isr(void) interrupt INT0_VECTOR
{
    int0_flag = 1;  // int0_flag 置 1 是通知主循环处理部分 INT0 中断事件不需要特急处理的任务
                    //置 1，记录 INT0 已产生中断，供主循环查询判断有无需处理的 INT0 任务

    if(INT0)                //边沿中断，进入后再次判断电平从而判断是什么样的电平
    {
        P00 = 0;            //判断为高电平，则当前为上升沿，点亮 P00 端口上的 LED 灯
        _nop_();            //可以在这里插入断点进行观察现象
        P00 = 1;            //关闭 LED 灯
    }
    else
    {
        P07 = 0;            //判断为低电平，则当前为下降沿，点亮 P07 端口上的 LED 灯
        _nop_();            //可以在这里插入断点进行观察现象
        P07 = 1;            //关闭 LED 灯
    }
}

void int1_isr(void) interrupt INT1_VECTOR
{
    int1_flag = 1;  // int1_flag 置 1 是通知主循环处理部分 INT1 中断事件不需要特急处理的任务
                    //置 1，记录 INT1 已产生中断，供主循环查询判断有无需处理的 INT1 任务

    P04 = 0;                //点亮 P04 端口上的 LED 灯
    _nop_();                //可以在这里插入断点观察现象
    P04 = 1;                //关闭 LED 灯
}
```

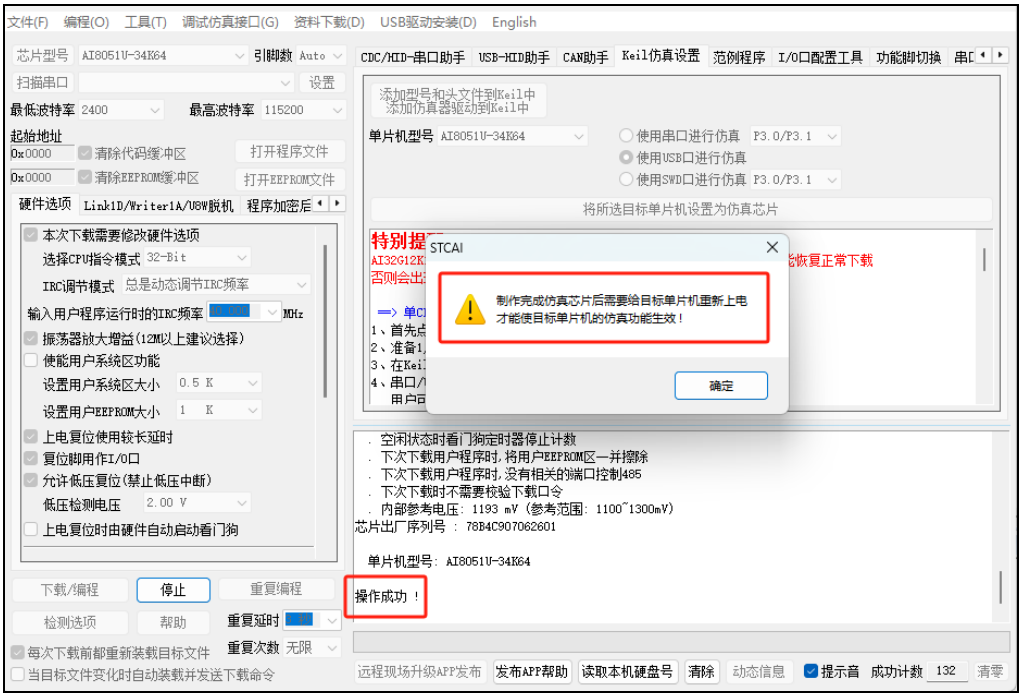
d) 制作仿真芯片

在 AIapp-ISP 软件界面，看到扫描串口列表框显示“(HID1) USB Writer”：

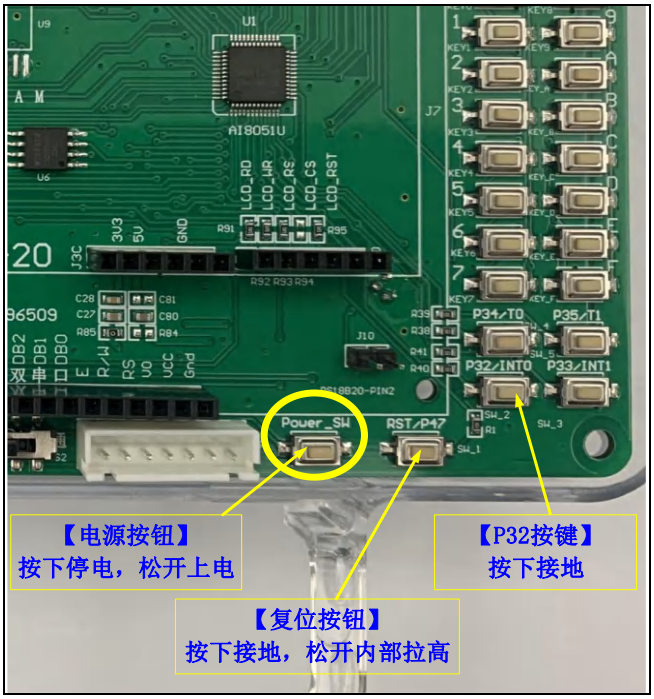
- 点击“Keil 仿真设置”标签；
- 在“Keil 仿真设置”选项卡，选择“使用 USB 口进行仿真”
- 点击“将所选目标单片机设置为仿真芯片”



然后看到 AIapp-ISP 软件在自动进行仿真芯片的设置，界面下面的进度条在推进，然后“操作成功！”界面出现特别提示：“制作完成仿真芯片后需要给目标单片机重新上电才能使目标单片机的仿真生效！”如下图：

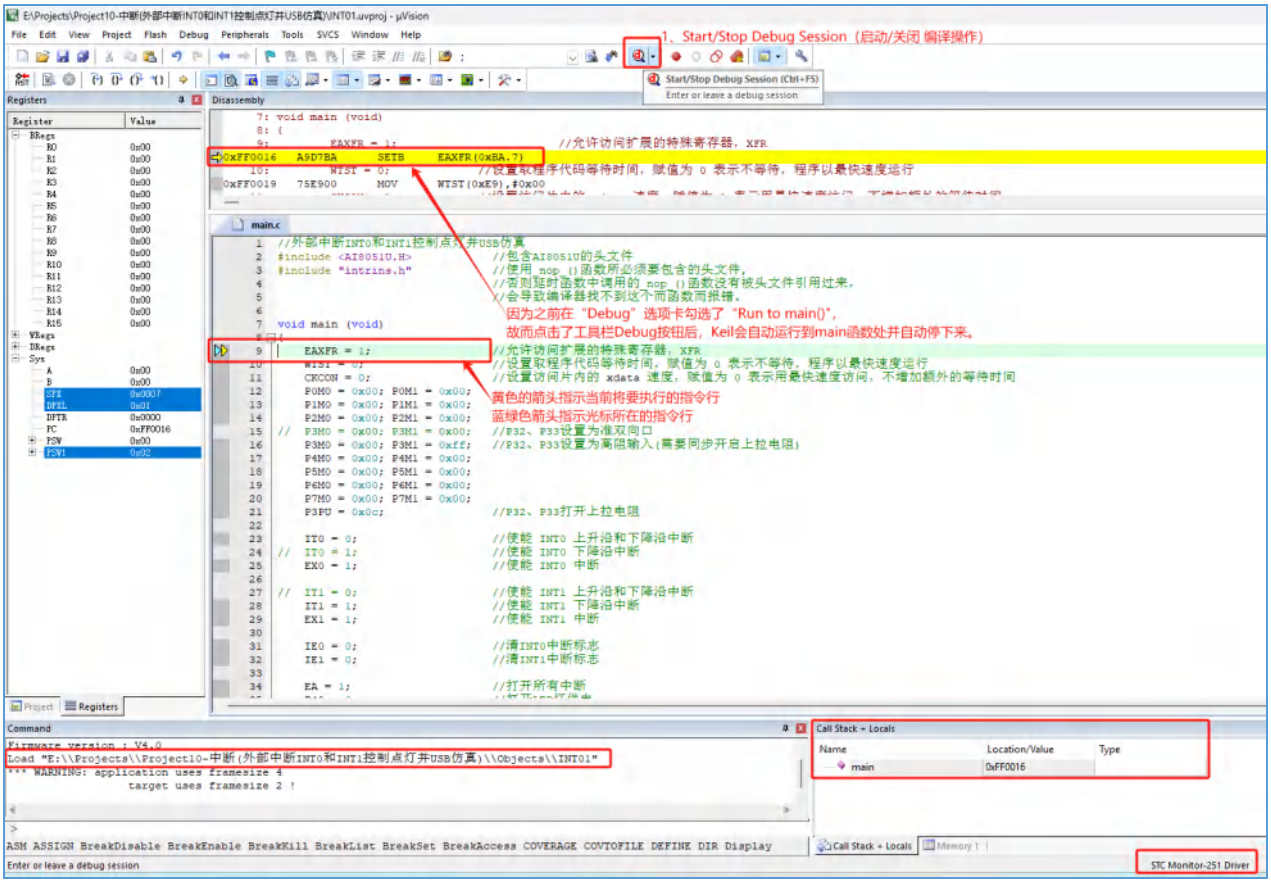


所以，按提示按一下实验箱上的“Power_SW”按钮：



e) Keil 仿真

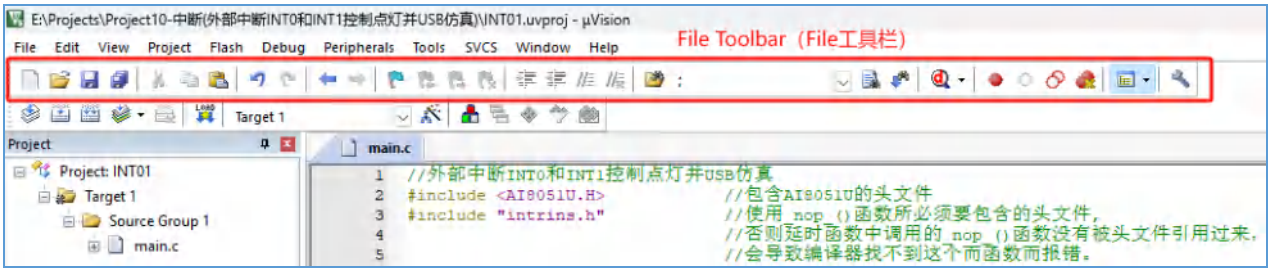
现在，返回 Keil 界面，上述代码程序编译无误后，点击工具栏的 Debug（启动/关闭编译调试操作界面）按钮后，Keil 软件进入仿真界面，如下图：



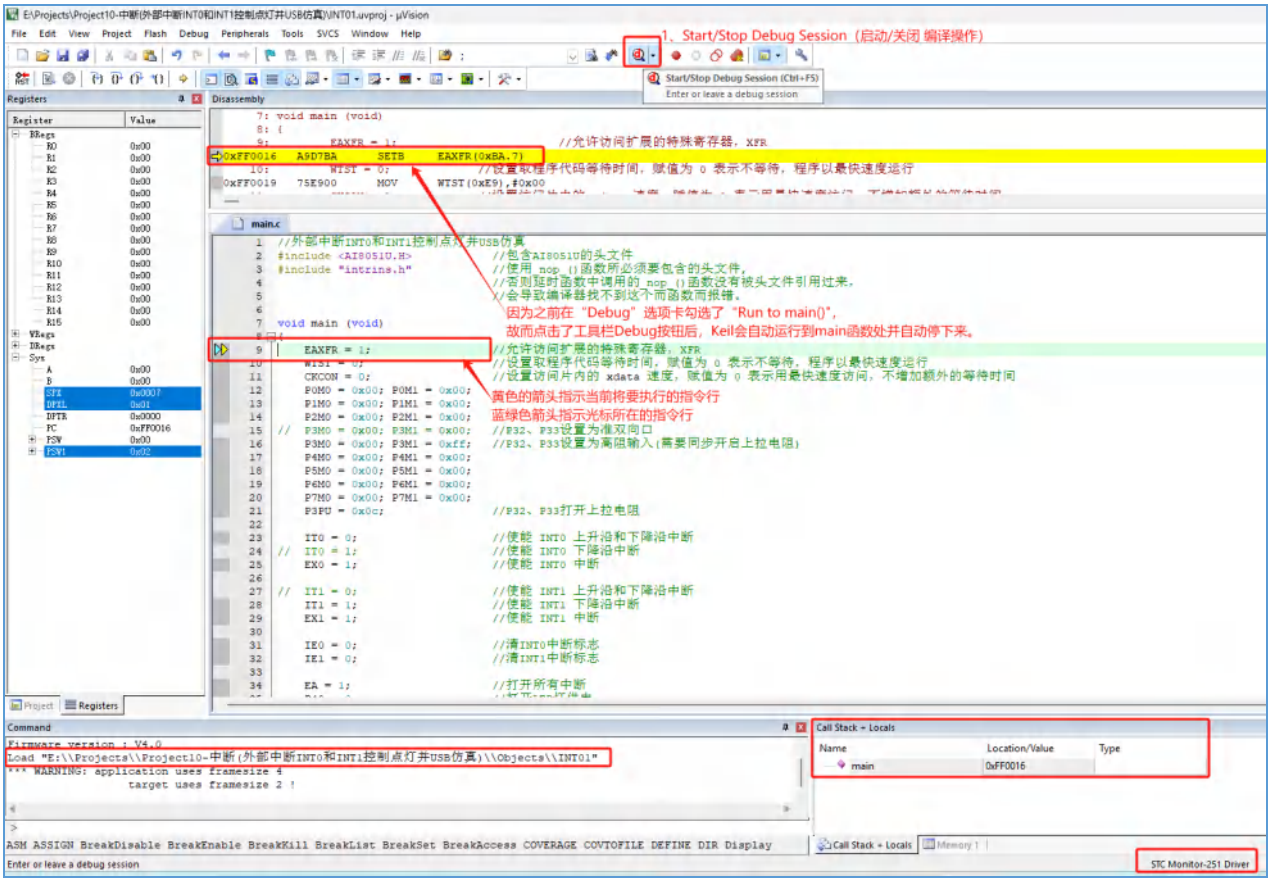
提示：退出 Debug 调试界面，同样点击工具栏的“Debug”按钮。

f) 认识 Keil 软件的 File Toolbar（File 工具栏）

在集成开发环境的使用中，介绍了 Debug 工具栏的常用按钮。本实验，还需要我们认识 File Toolbar（File 工具栏）及几个常用调试/仿真按钮：



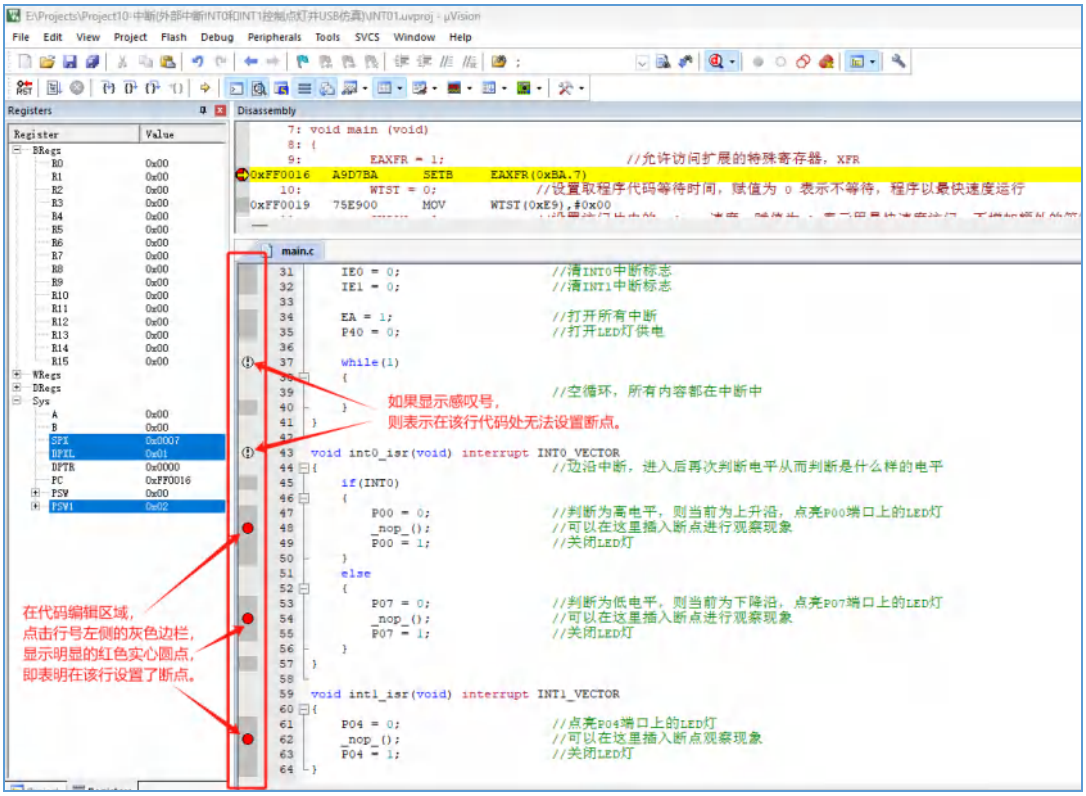
Start/Stop Debug Session（Enter or leave a debug session）按钮：启动/停止 Debug 按钮。在软件开发过程中，调试是不可或缺的一环。Keil 提供了丰富的 Debug 调试工具，帮助我们快速定位和解决程序中的问题。点击此按钮，即刻进入 Debug 界面，如下图。再次点击此按钮，就会退出 Debug 界面。



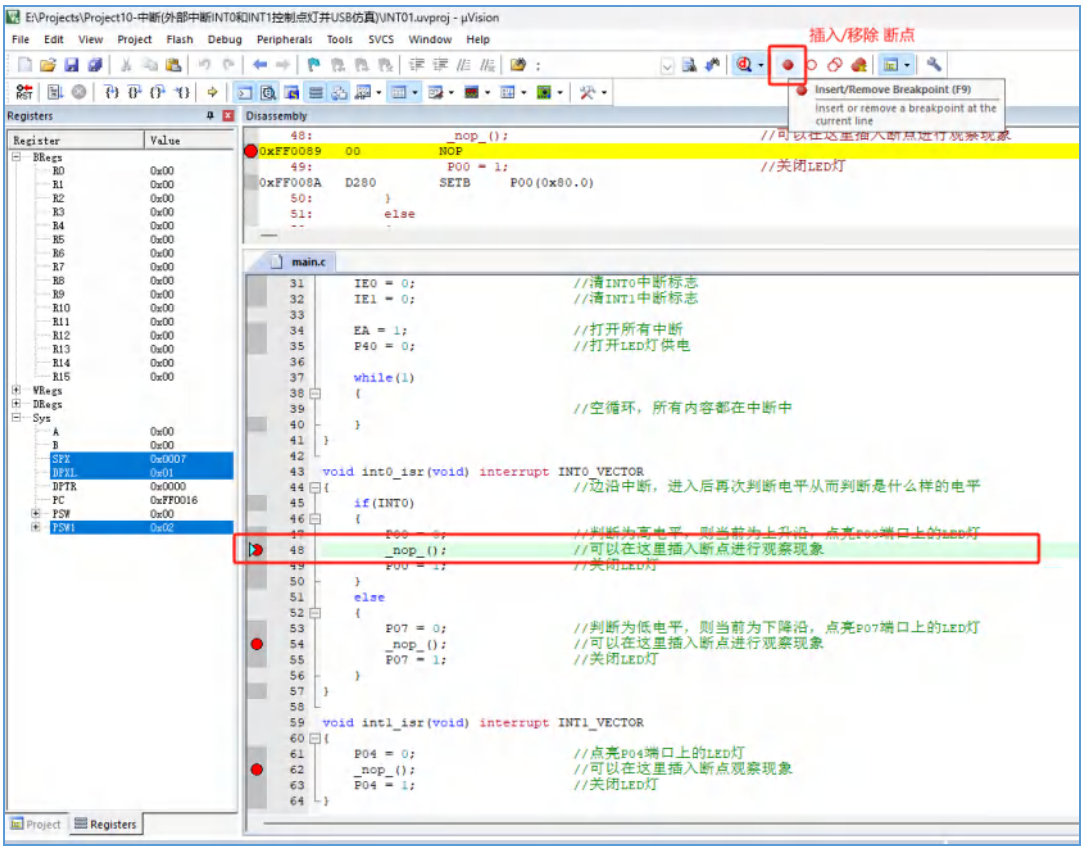
在调试/仿真程序时，会涉及一个很重要的工具：Breakpoint（断点）。

Breakpoint：断点。在 Keil 中，断点是一种强大的调试工具。断点本身是程序中的一个位置标记，当程序执行到断点位置时，程序会暂停执行，从而方便我们观察实验现象，定位、分析和排查程序中的问题等。在 Keil 环境中，设置断点有多种方式，下面我们介绍最简单的 2 种断点设置方法。

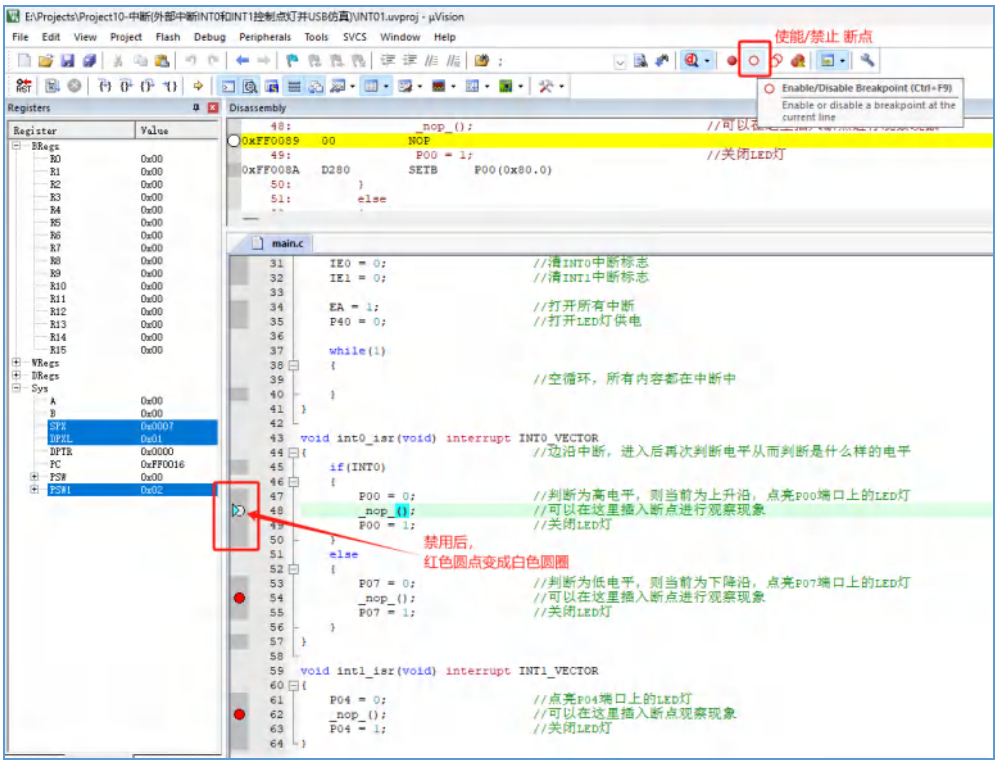
1、在代码编辑区域，点击行号左侧的灰色边栏，显示明显的红色实心圆点，即表明在该行代码处插入了断点。如果显示感叹号，则表示在该行代码处无法设置断点。在断点处点击断点（红色实心圆点）即为取消断点。



2、Insert/Remove Breakpoint（Insert or remove a breakpoint at the current line）按钮：在当前光标所在行插入/移除断点。在当前光标所在行，该按钮如果显示红色，则表示可在该处插入断点。点击该按钮，即在当前行插入断点，当前行左侧就会显示一个红色圆点表示断点位置。该按钮如果显示灰色，则无法在该处插入断点。定位光标在断点所在代码行，点击该按钮，即移除该处断点，红色圆点消失。

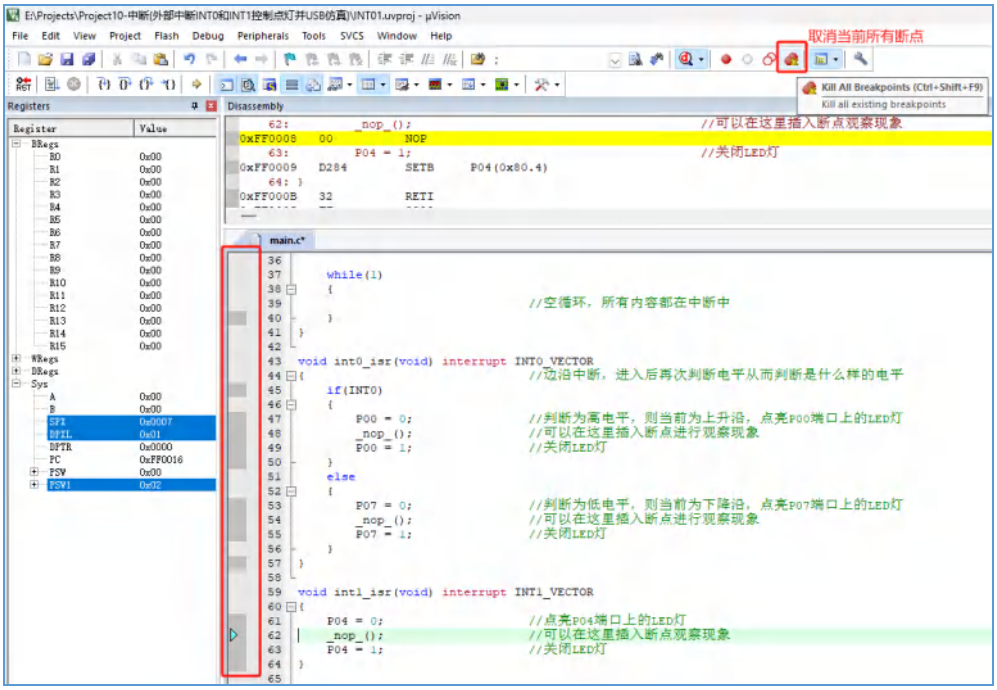


Enable/Disable Breakpoint（Enable or disable a breakpoint at the current line）：使能/禁止断点。启用/禁用当前光标所在行的断点。断点在启用时，红色圆点显示。禁用后，红色圆点变成白色圆圈，在调试中就不起作用了。断点使能/禁止按钮，在设置断点后，你可以使用此功能按钮来启用或禁用断点，从而控制代码的暂停与继续。



Disable All Breakpoints（Disable all existing breakpoints）：禁用当前所有断点。点击该按钮后，当前所有断点，由红色圆点变成白色圆圈，在调试中就不起作用了。

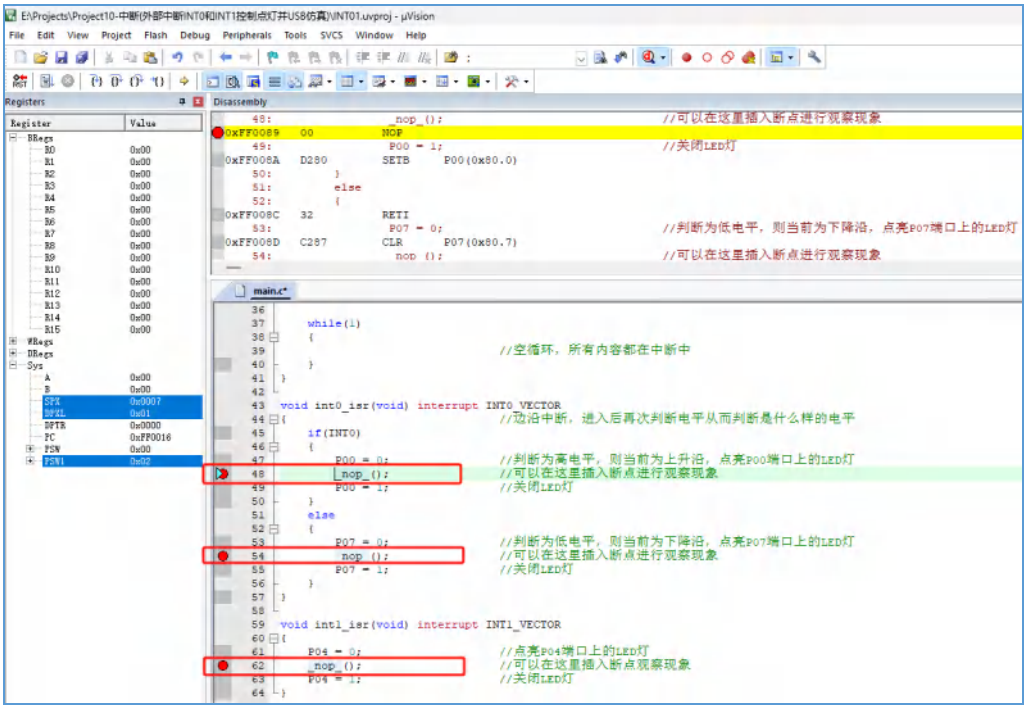
Kill All Breakpoints（Kill all existing breakpoints）：取消当前所有断点。点击该按钮后，所有断点标记消失。



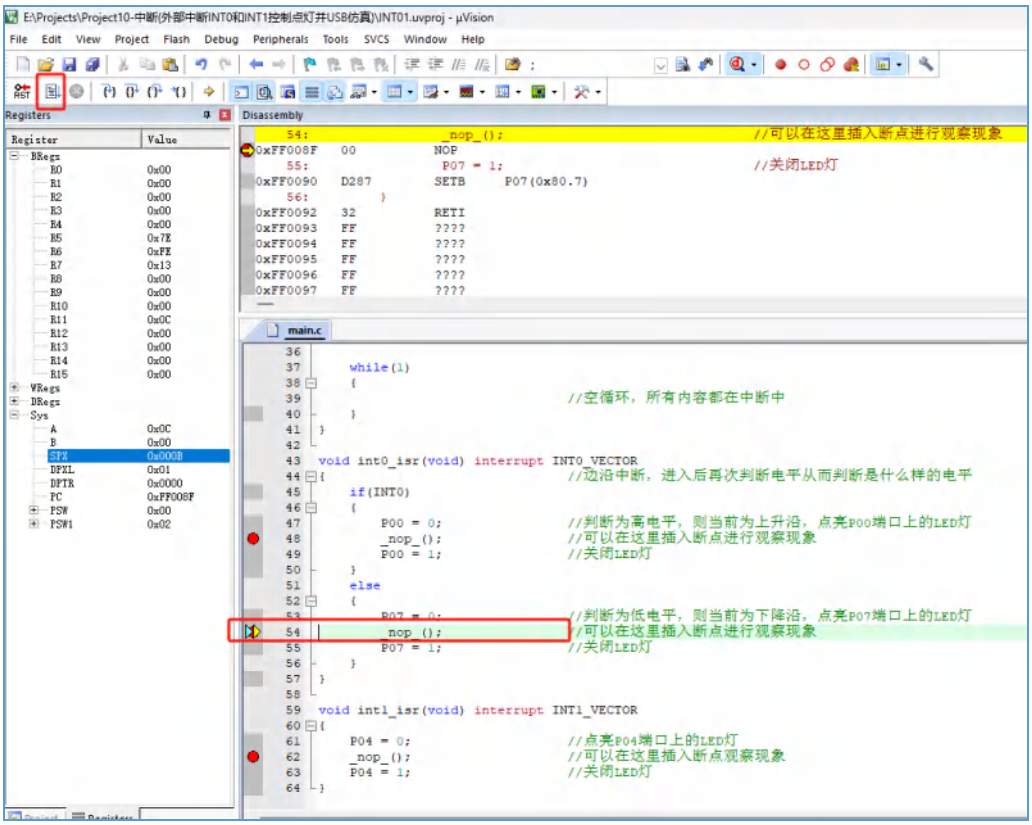
g) 程序实现效果

步骤一：利用仿真，可以设置运行到断点处，捕捉当前程序运行到哪，INT0/P3.2 是下降沿还是上升沿

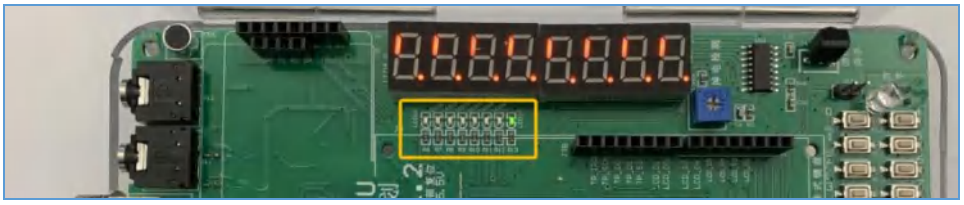
为方便观察实验现象，需在中断服务程序中设置如下 3 个断点：



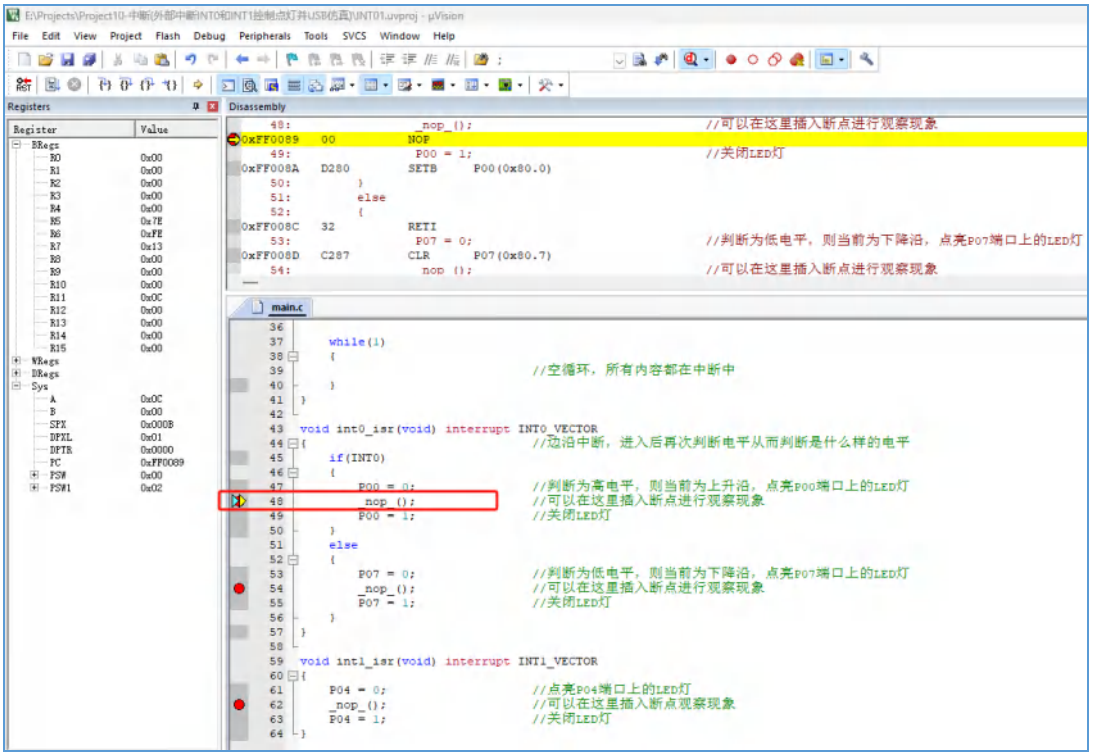
先点击 Run（运行）按钮，按下 P3.2 按键，保持不松开（为了验证下一步的实验），程序会停在如下图所示断点处：



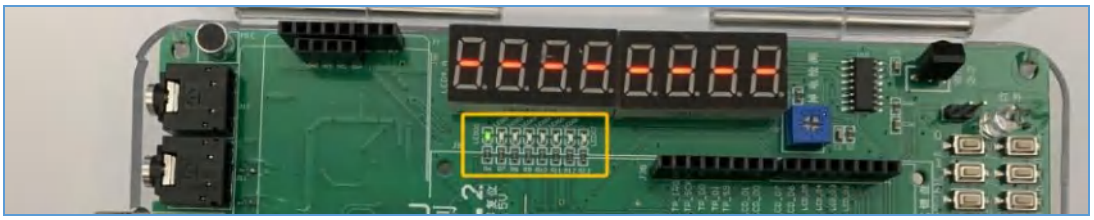
进 INT0 中断后，读到 INT0/P3.2 口是 低，下降沿，点亮 P0.7，实验箱如图：



继续点击 Run（运行）按钮运行，退出中断服务程序。再松开 P3.2 按键，程序会停在如下图所示断点处：



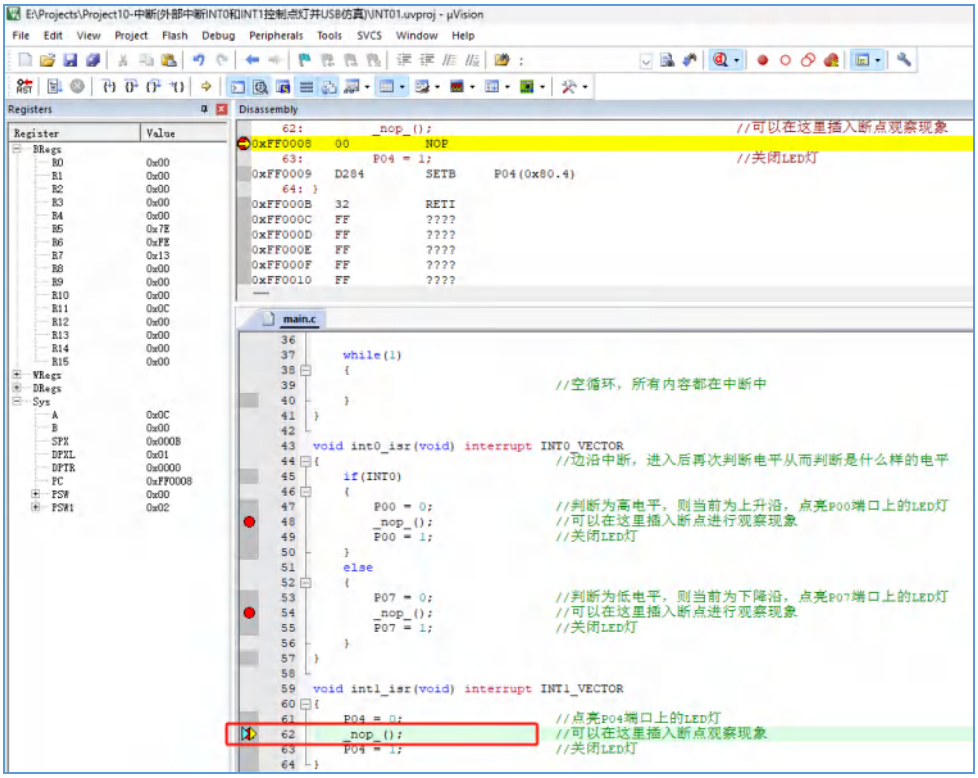
进 INT0 中断后，读到 INT0/P3.2 口是 高，上升沿，点亮 P0.0，实验箱如图：



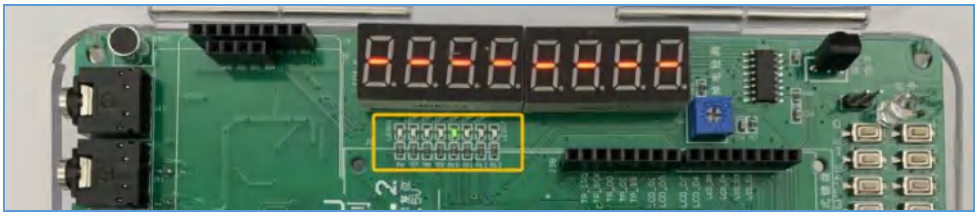
继续点击 Run（运行）按钮运行，退出中断服务程序。

步骤二：利用仿真，可以设置运行到 INT1/P3.3 外部中断的断点处

如果程序没有在运行，需先点击 Run（运行）按钮，按一下 P3.3 按键，程序会停在如下图所示断点处：



进 INT1 中断后，点亮 P0.4，实验箱如图：



继续点击 Run（运行）按钮运行，退出中断服务程序。

完成本实验目的。

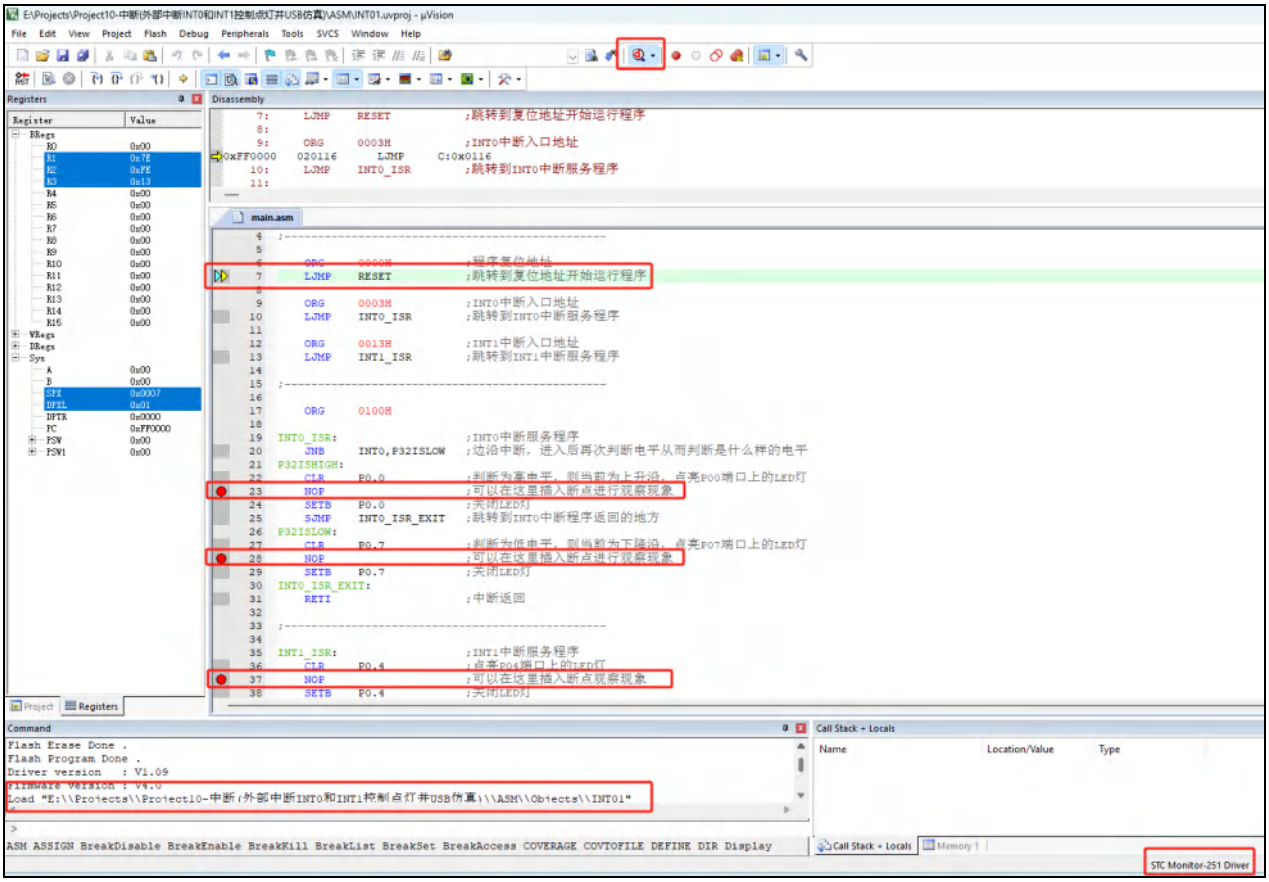
h) 实验程序汇编语言代码

```
;外部中断 INT0 和 INT1 控制点灯并 USB 仿真
$INCLUDE(AI8051U.INC)          ;包含 AI8051U 的头文件
;-----------------------------------------------

    ORG     0000H               ;程序复位地址
    LJMP    START               ;跳转到复位地址开始运行程序

    ORG     0003H               ;INT0 中断入口地址
    LJMP    INT0_ISR            ;跳转到 INT0 中断服务程序

    ORG     0013H               ;INT1 中断入口地址
    LJMP    INT1_ISR            ;跳转到 INT1 中断服务程序

;-----------------------------------------------

    ORG     0100H

INT0_ISR:                       ;INT0 中断服务程序
    JNB     INT0, P32ISLOW      ;边沿中断，进入后再次判断电平从而判断是什么样的电平

P32ISHIGH:
    CLR     P0.0                ;判断为高电平，则当前为上升沿，点亮 P00 端口上的 LED 灯
    NOP                         ;可以在这里插入断点进行观察现象
    SETB    P0.0                ;关闭 LED 灯
    SJMP    INT0_ISR_EXIT       ;跳转到 INT0 中断程序返回的地方

P32ISLOW:
    CLR     P0.7                ;判断为低电平，则当前为下降沿，点亮 P07 端口上的 LED 灯
    NOP                         ;可以在这里插入断点进行观察现象
    SETB    P0.7                ;关闭 LED 灯

INT0_ISR_EXIT:
    RETI                        ;中断返回

;-----------------------------------------------

INT1_ISR:                       ;INT1 中断服务程序
    CLR     P0.4                ;点亮 P04 端口上的 LED 灯
    NOP                         ;可以在这里插入断点观察现象
    SETB    P0.4                ;关闭 LED 灯
    RETI                        ;中断返回

;-----------------------------------------------
```

```
START:

    SETB    EAXFR           ;允许访问扩展的特殊寄存器，XFR
    MOV     WTST, #00H      ;设置取程序代码等待时间，赋值为 0 表示不等待，程序以最快速度运行
    MOV     CKCON, #00H
            ;设置访问片内的 xdata 速度，赋值为 0 表示用最快速度访问，不增加额外的等待时间

    MOV     P0M0, #38H      ;设置 P03、P04 为推挽输出，P05 为开漏输出，其余为准双向口模式
    MOV     P0M1, #20H
    MOV     P1M0, #00H      ;设置 P1 口为准双向口模式
    MOV     P1M1, #00H
    MOV     P2M0, #00H      ;设置 P2 口为准双向口模式
    MOV     P2M1, #00H
    MOV     P3M0, #00H      ;P32、P33 设置为高阻输入(需要同步开启上拉电阻)
    MOV     P3M1, #0CH
    MOV     P4M0, #00H      ;设置 P4 口为准双向口模式
    MOV     P4M1, #00H
    MOV     P5M0, #00H      ;设置 P5 口为准双向口模式
    MOV     P5M1, #00H
    MOV     P6M0, #00H      ;设置 P6 口为准双向口模式
    MOV     P6M1, #00H
    MOV     P7M0, #00H      ;设置 P7 口为准双向口模式
    MOV     P7M1, #00H

    MOV     WR0, #WORD2 P3PU        ;P32、P33 打开上拉电阻
    MOV     WR2, #WORD0 P3PU
    MOV     A, #0CH
    MOV     @DR0, R11

    CLR     IT0             ;使能 INT0 上升沿和下降沿中断
;   SETB    IT0             ;使能 INT0 下降沿中断
    SETB    EX0             ;使能 INT0 中断

;   CLR     IT1             ;使能 INT1 上升沿和下降沿中断
    SETB    IT1             ;使能 INT1 下降沿中断
    SETB    EX1             ;使能 INT1 中断

    CLR     IE0             ;清 INT0 中断标志
    CLR     IE1             ;清 INT1 中断标志

    SETB    EA              ;打开总中断开关
    CLR     P40             ;打开 LED 灯供电

;----------------------------------------------
```

```
MAINLOOP:
    SJMP    MAINLOOP        ;主循环为空循环，所有内容都在中断中

;------------------------------------------------

END
```

使用 Aiapp-ISP 系统软件及 Ai8051U 实验箱制作仿真芯片，上述代码编译无误，点击“Debug”按钮，进入仿真界面。为方便观察实验现象，需在中断服务程序中设置如下 3 个断点，如下图：



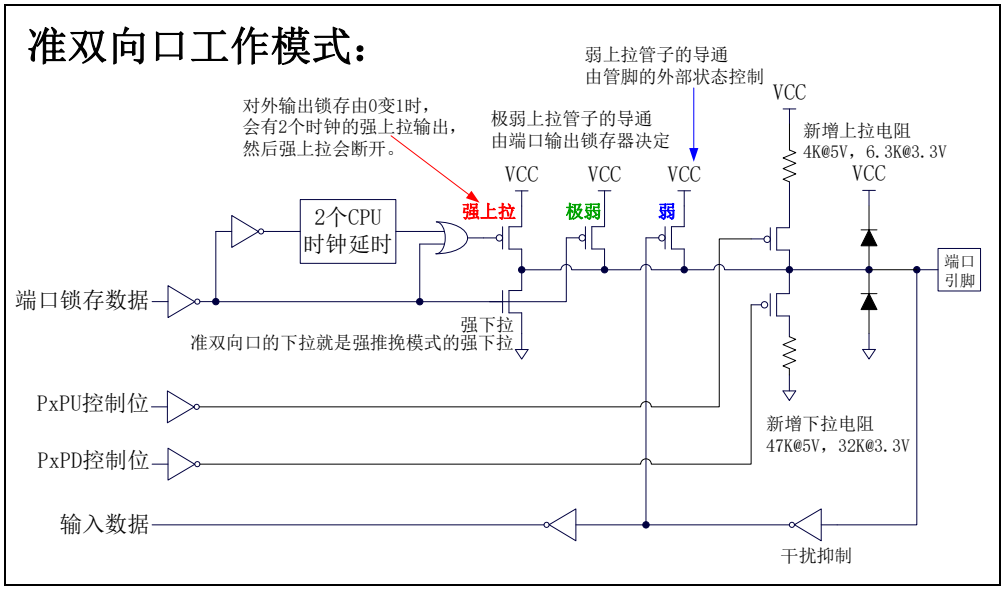
实验步骤及实验现象与 C 语言代码编写的该项目完全相同。

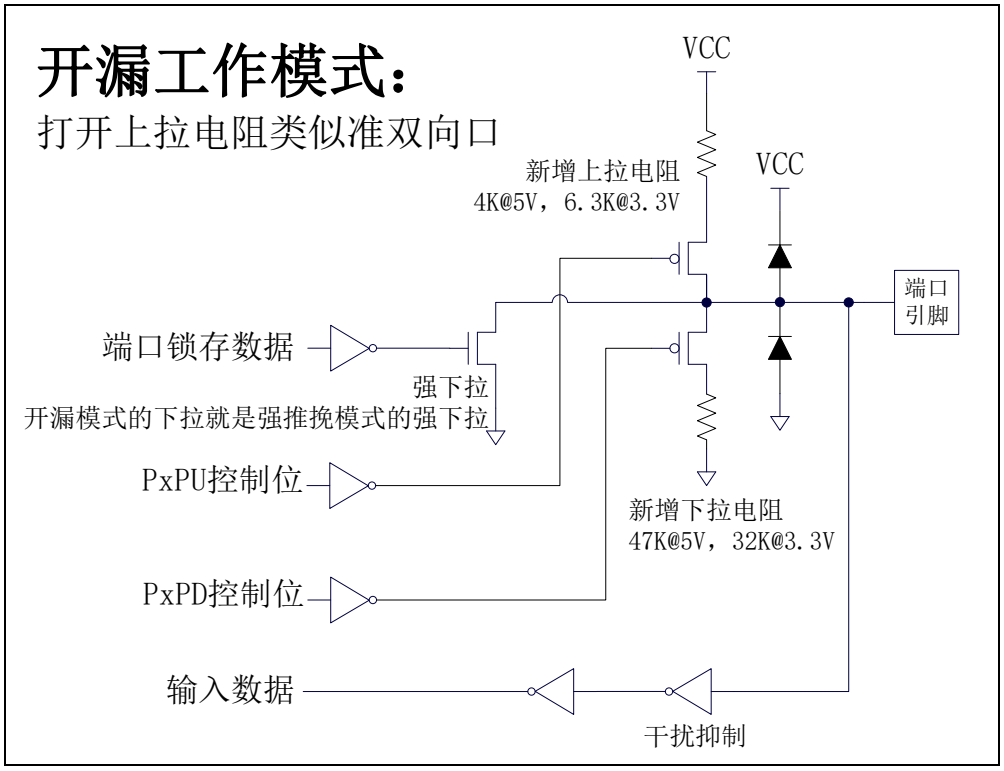
15.6.2 深大--中断实验 2(单片机主动触发外部中断 INT2/P3.6，INT3/P3.7 控制点灯并 USB 仿真)

a) 原理图

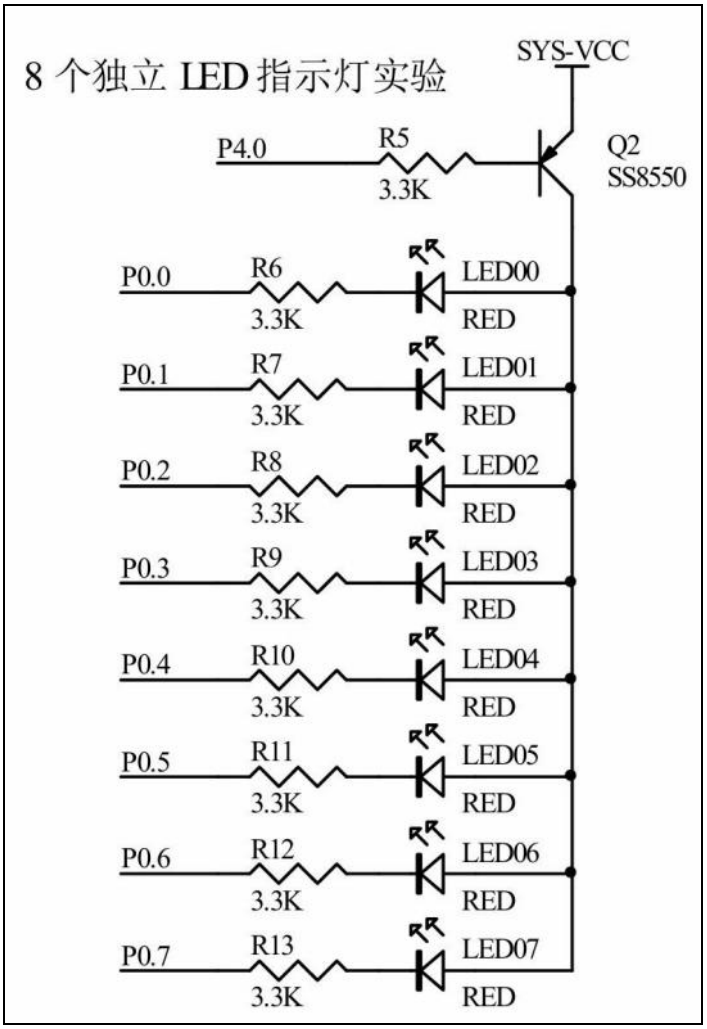
- **外部中断【INT2/P3.6，INT3/P3.7】实验原理图@Ai8051U 实验箱**

  在本实验中，【INT2/P3.6，INT3/P3.7】没有设计外部按键输入，而采用内部将【INT2/P3.6，INT3/P3.7】拉低产生下降沿的方式产生中断的方法。【INT2/P3.6，INT3/P3.7】工作在准双向口/开漏模式时，输出是独立的，输入也是独立的，【INT2/P3.6，INT3/P3.7】对外输出低后，【INT2/P3.6，INT3/P3.7】读到的外部状态就是低；输出高，在打开内部上拉电阻或有上拉电阻的状态下，读到的就是高，除非外界有强行拉低的电路。





➢ **外部中断实验辅助显示 LED 灯原理图：**



I/O 口 P4.0 控制 PNP 三极管的通断，控制这 8 个灯的整体供电。
如果 P4.0 输出 1，则 PNP 三极管不导通，这 8 个灯都不会获得电源，都不会有电流流过，都不会被点亮。
如果 P4.0 输出 0，则 PNP 三极管导通，这 8 个灯才有可能会流过电流，才有可能会被点亮。
如 P4.0 输出 0，PNP 三极管导通，特定 LED 灯的控制 I/O 口如输出 0，则该 LED 灯会有电流流过，会被点亮。
如 P0.0 输出 0，LED00 灯就会被点亮；如 P0.0 输出 1，LED00 灯就会熄灭；
如 P0.1 输出 0，LED01 灯就会被点亮；如 P0.1 输出 1，LED01 灯就会熄灭；
如 P0.2 输出 0，LED02 灯就会被点亮；如 P0.2 输出 1，LED02 灯就会熄灭；
如 P0.3 输出 0，LED03 灯就会被点亮；如 P0.3 输出 1，LED03 灯就会熄灭；
如 P0.4 输出 0，LED03 灯就会被点亮；如 P0.4 输出 1，LED03 灯就会熄灭；
如 P0.5 输出 0，LED03 灯就会被点亮；如 P0.5 输出 1，LED03 灯就会熄灭；
如 P0.6 输出 0，LED03 灯就会被点亮；如 P0.6 输出 1，LED03 灯就会熄灭；
如 P0.7 输出 0，LED03 灯就会被点亮；如 P0.7 输出 1，LED03 灯就会熄灭；

如 P0 口对外全部送 0，则这 8 个灯全亮；
如 P0 口对外全部送 1，则这 8 个灯全灭；
如果 P4.0 输出 1，则 PNP 三极管不导通，不管 P0 口对外全部送 0 或送 1，这 8 个灯都不会有电流流过，都不会亮。P4.0 控制 PNP 三极管，作为这 8 个 LED 指示灯的总电源开关。

b) 实验介绍

1，下降沿触发中断，单片机内部主动触发 INT2/P3.6 口，产生中断实验演示程序：

P3.6 设置为高阻输入，打开内部上拉电阻

或　P3.6 设置为准双向口，打开内部上拉电阻

INT2/P3.6 固定为下降沿触发中断

2，下降沿触发中断，单片机内部主动触发 INT3/P3.7 口，产生中断实验演示程序：

P3.7 设置为高阻输入，打开内部上拉电阻

或　P3.7 设置为准双向口，打开内部上拉电阻

INT3/P3.7 固定为下降沿触发中断

【实验步骤】：

- 先点击 Run（运行）按钮，运行到单片机内部主动触发 INT2/P3.6 时，会进入 INT2 中断，点亮 P0.0 灯，退出中断服务程序，继续运行。
- 运行到单片机内部主动触发 INT3/P3.7 口时，会进入 INT3 中断，点亮 P0.7 灯，退出中断服务程序。

利用仿真，可以设置运行到断点处，捕捉当前程序运行到哪。

c) 实验程序代码

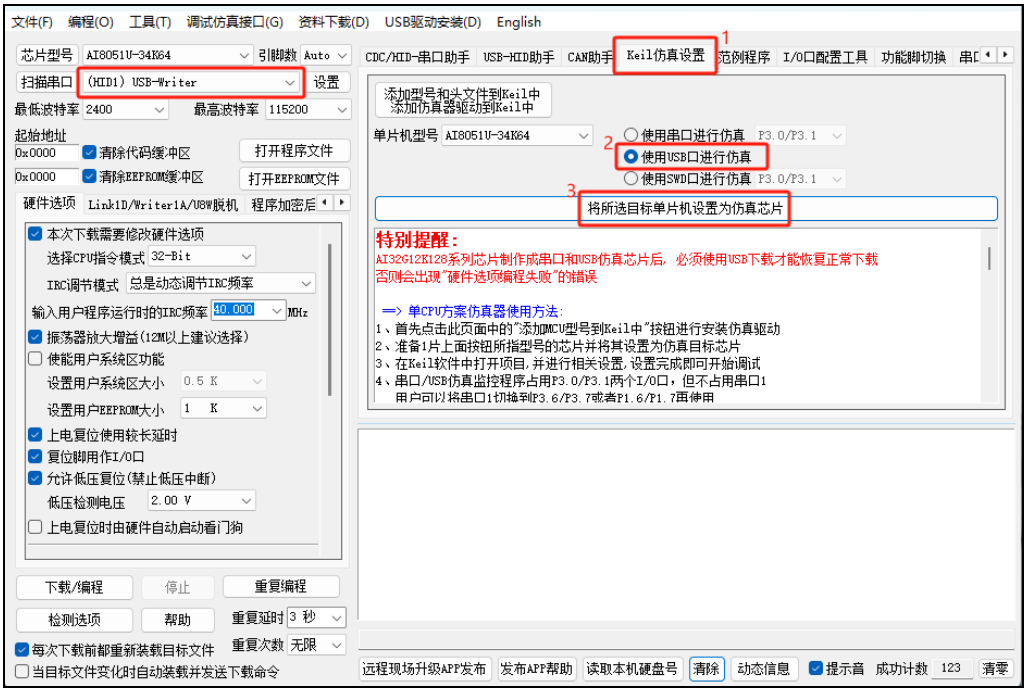
```
#include <AI8051U.H>
#include "intrins.h"            //使用_nop_()函数所必须要包含的头文件,
                                //否则延时函数中调用的_nop_()函数没有被头文件引用过来，
                                //会导致编译器找不到这个而函数而报错。

bit   int2_flag = 0;  //定义 1 个位变量，INT2 事件位变量标志，记录 INT2 已产生中断
//  供主循环查询 INT2 是否已产生中断，在主循环中处理 INT2 的中断事件任务，不堵塞其他中断
bit   int3_flag = 0;  //定义 1 个位变量，INT3 事件位变量标志，记录 INT3 已产生中断
//  供主循环查询 INT3 是否已产生中断，在主循环中处理 INT3 的中断事件任务，不堵塞其他中断

void main (void)
{
    EAXFR = 1;              //允许访问扩展的特殊寄存器，XFR
    WTST = 0;               //设置取程序代码等待时间，赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;              //设置访问片内的 xdata 速度，赋值为 0 表示用最快速度访问，
                            //不增加额外的等待时间
    P0M0 = 0x00; P0M1 = 0x00;
    P1M0 = 0x00; P1M1 = 0x00;
    P2M0 = 0x00; P2M1 = 0x00;
    P3M0 = 0x00; P3M1 = 0x00;       //P36、P37 设置为准双向口
    P4M0 = 0x00; P4M1 = 0x00;
    P5M0 = 0x00; P5M1 = 0x00;
    P6M0 = 0x00; P6M1 = 0x00;
    P7M0 = 0x00; P7M1 = 0x00;
    P3PU = 0xc0;            //P36、P37 打开上拉电阻
    AUXINTIF = 0x00;        //清空 INT2、INT3 的中断标志位，防止仿真的时候被错误触发一次
    EX2 = 1;                //使能 INT2 中断，固定只有下降沿中断
    EX3 = 1;                //使能 INT3 中断，固定只有下降沿中断

    EA = 1;                 //总中断允许位打开
    P40 = 0;                //打开 LED 灯供电

    P36 = 1;                //INT2 所在 I/O 端口
    NOP12();                //可以作为断点插入，用于观察现象
    P36 = 0;                //手动创造下降沿，这里就会直接跳转到 INT2 中断了
    NOP12();                //芯片内部对下降沿信号进行采样，此处不可设置断点
    NOP12();                //可以作为断点插入，用于观察现象

    P37 = 1;                //INT3 所在 I/O 端口
    NOP12();                //可以作为断点插入，用于观察现象
    P37 = 0;                //手动创造下降沿，这里就会直接跳转到 INT3 中断了
    NOP12();                //芯片内部对下降沿信号进行采样，此处不可设置断点
    NOP12();                //可以作为断点插入，用于观察现象
```

```
    P37 = 1;                //手动创造上升沿，此时不会进入中断
    NOP12();                //不会进入中断，用于观察现象
    NOP12();                //可以作为断点插入，用于观察现象

    while(1)
    {
        if(int2_flag)   //主循环中查询，INT2 是否已产生中断，是否有需要处理的 INT2 事件
        {
            int2_flag = 0;      //清 0，INT2 事件位变量标志
            _nop_();            //用户在此添加需要处理的事件
            _nop_();
        }
        if(int3_flag)   //主循环中查询，INT3 是否已产生中断，是否有需要处理的 INT3 事件
        {
            int3_flag = 0;      //清 0，INT3 事件位变量标志
            _nop_();            //用户在此添加需要处理的事件
            _nop_();
        }
    }
}

void int2_isr(void) interrupt INT2_VECTOR
{
    int2_flag = 1;  // int2_flag 置 1 是通知主循环处理部分 INT2 中断事件不需要特急处理的任务
                    //置 1，记录 INT2 已产生中断，供主循环查询判断有无需处理的 INT2 任务

    P00 = 0;                //点亮 P00 端口上的 LED 灯
    NOP12();                //可以在这里插入断点观察现象
    P00 = 1;                //关闭 LED 灯
}

void int3_isr(void) interrupt INT3_VECTOR
{
    int3_flag = 1;  // int3_flag 置 1 是通知主循环处理部分 INT3 中断事件不需要特急处理的任务
                    //置 1，记录 INT3 已产生中断，供主循环查询判断有无需处理的 INT3 任务

    P07 = 0;                //点亮 P07 端口上的 LED 灯
    _nop_();                //可以在这里插入断点观察现象
    P07 = 1;                //关闭 LED 灯
}
```

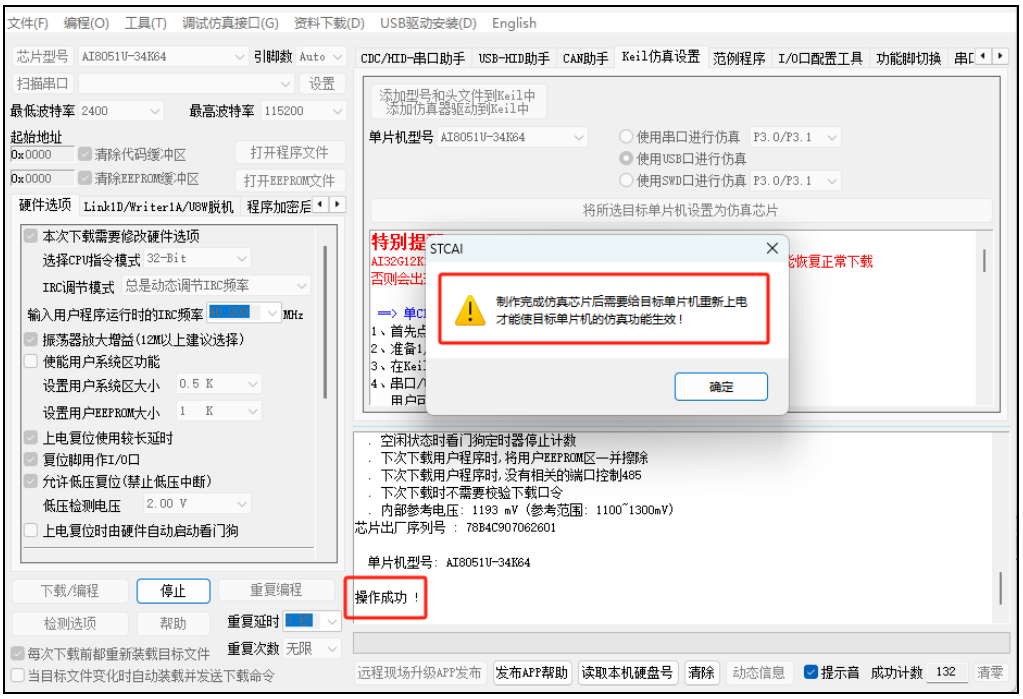
d) 制作仿真芯片

在 AIapp-ISP 软件界面，看到扫描串口列表框显示“(HID1) USB Writer”：

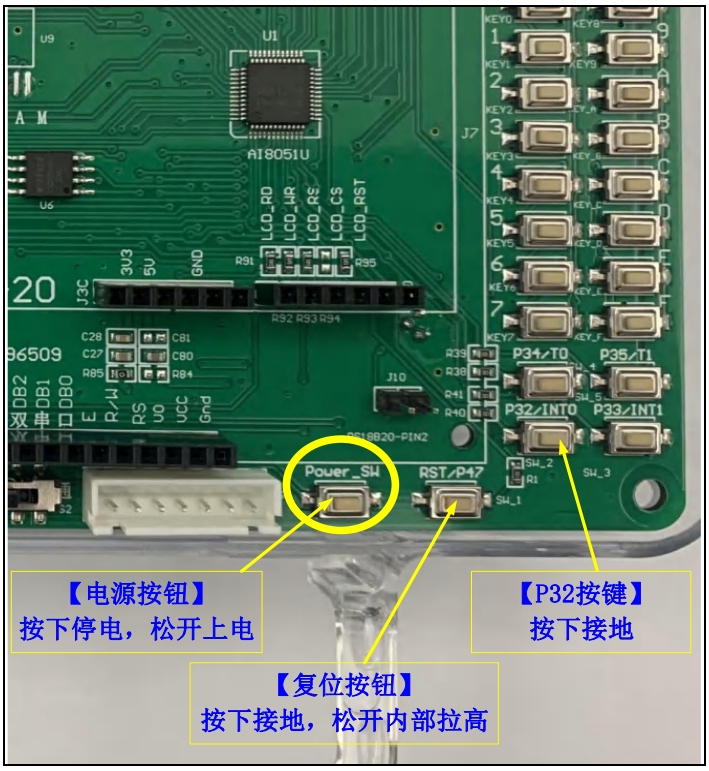
- 点击“Keil 仿真设置”标签；
- 在“Keil 仿真设置”选项卡，选择“使用 USB 口进行仿真”
- 点击“将所选目标单片机设置为仿真芯片”



然后看到 AIapp-ISP 软件在自动进行仿真芯片的设置，界面下面的进度条在推进，然后“操作成功！”界面出现特别提示：“制作完成仿真芯片后需要给目标单片机重新上电才能使目标单片机的仿真生效！”如下图：

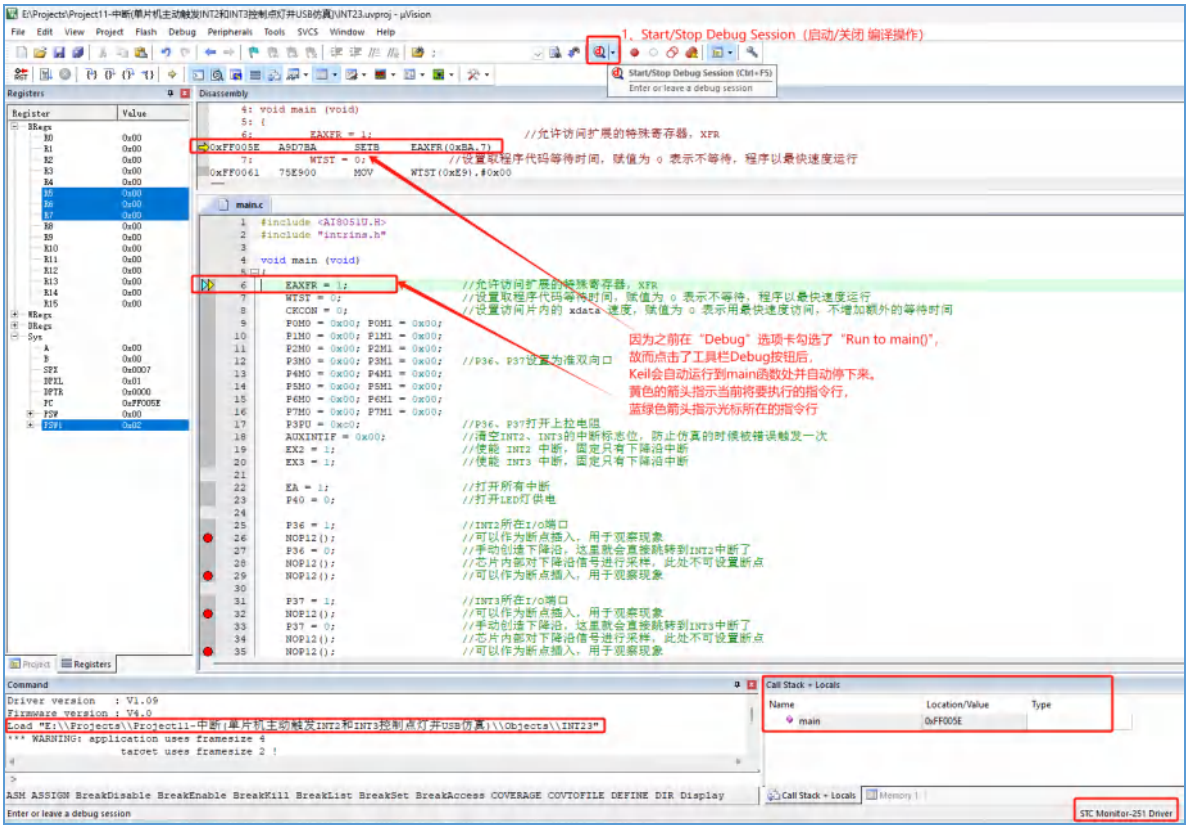


所以，按提示按一下实验箱上的“Power_SW”按钮：



e) Keil 仿真

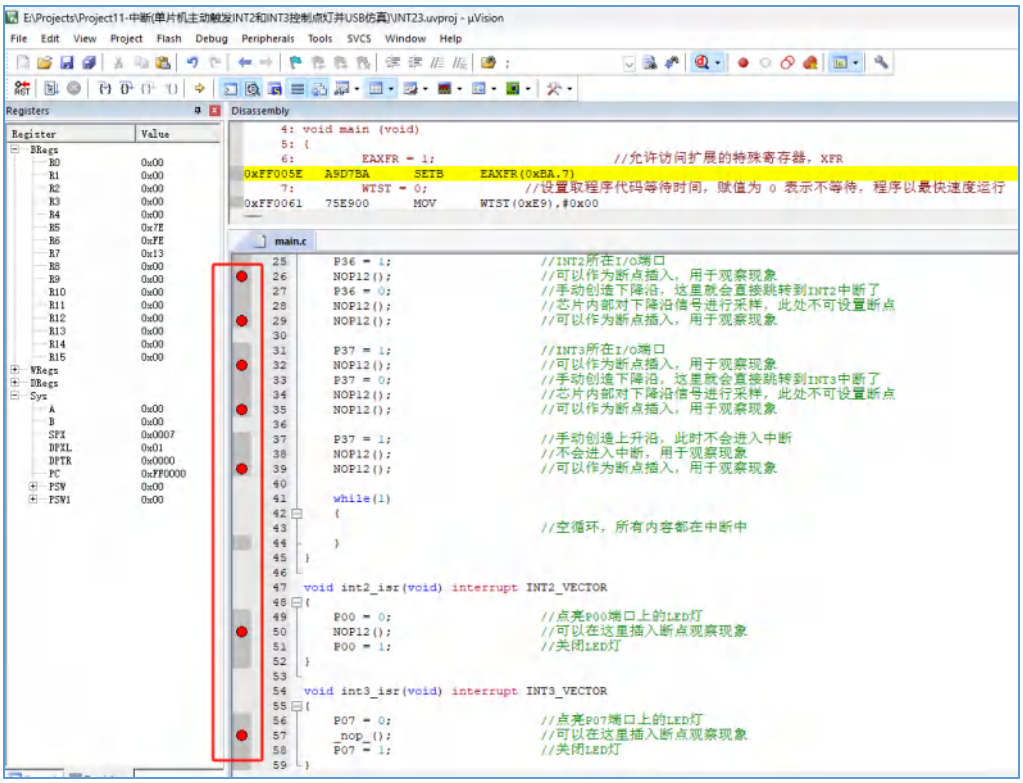
现在，返回 Keil 界面，上述代码程序编译无误后，点击工具栏的 Debug（启动/关闭编译调试操作界面）按钮后，Keil 软件进入仿真界面，如下图：



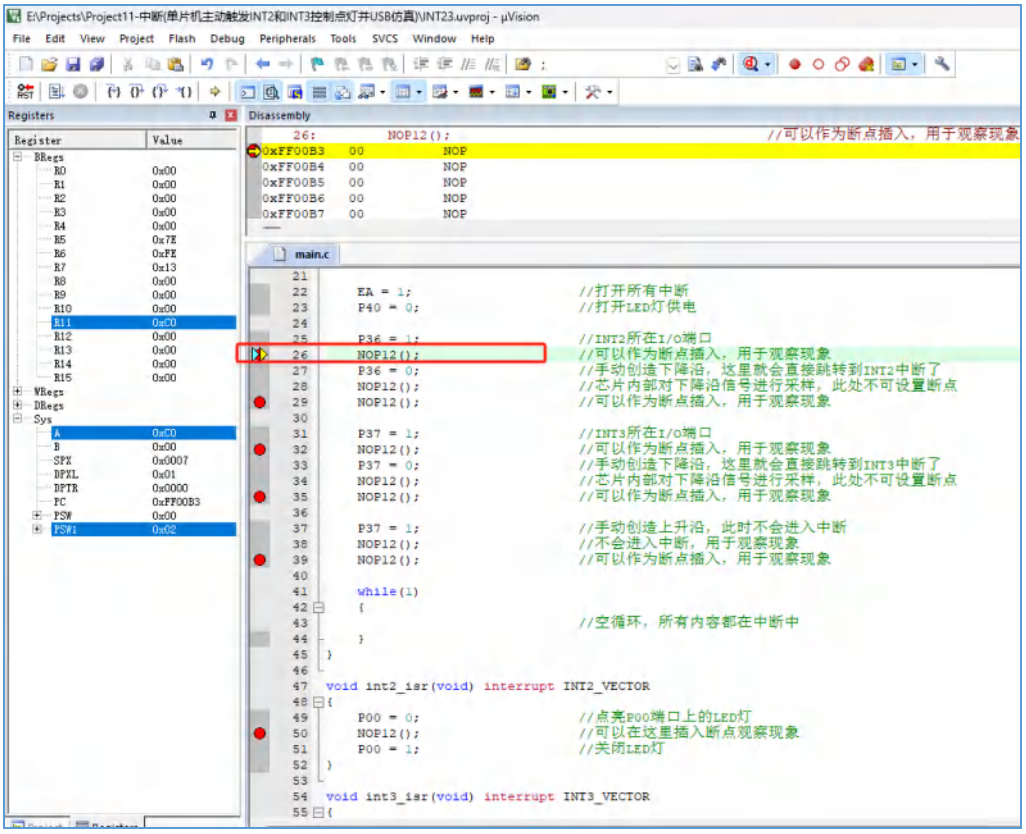
提示：退出 Debug 调试界面，同样点击工具栏的“Debug”按钮。

f) 程序实现效果

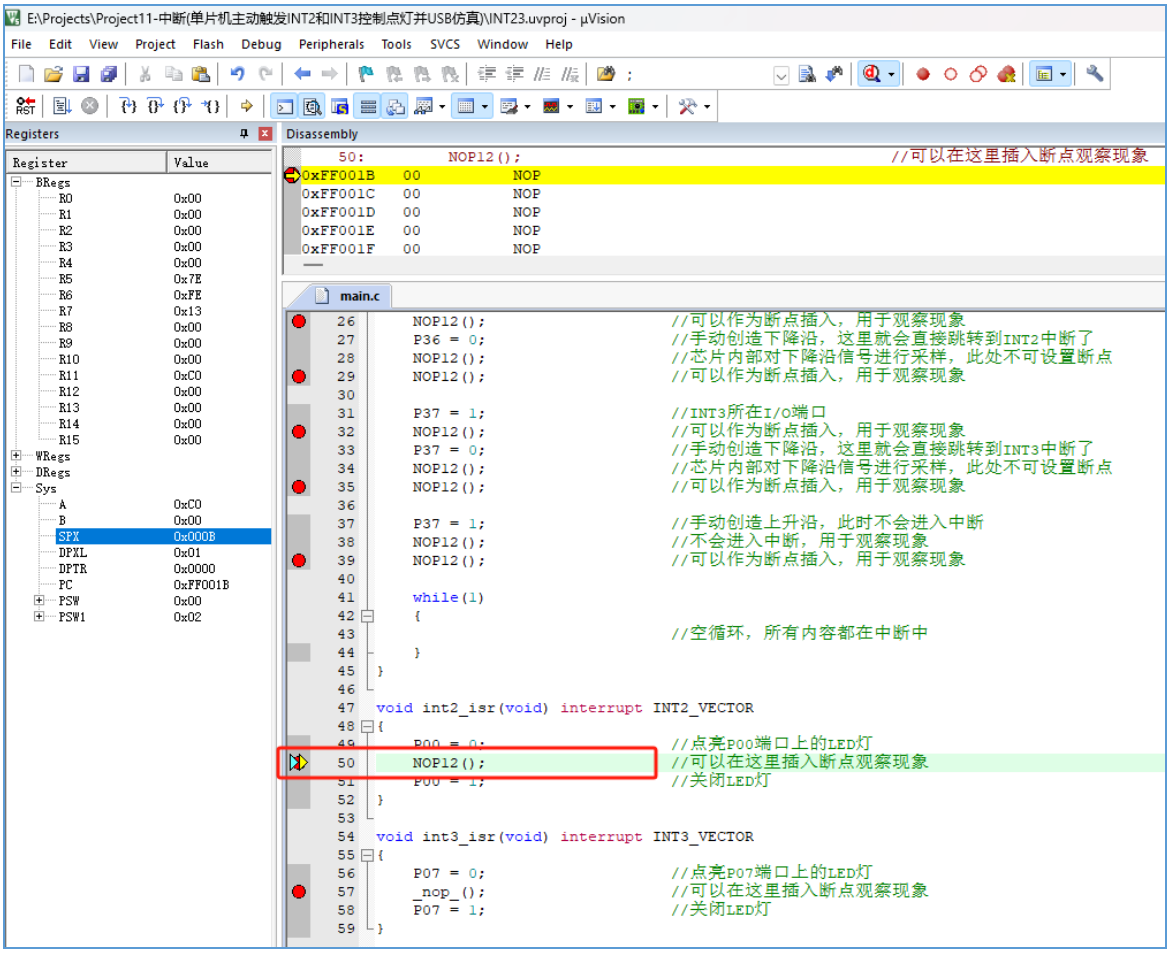
利用仿真，可以设置运行到断点处，捕捉当前程序运行到哪
为方便观察，需在中断服务程序中设置如下 7 个断点：



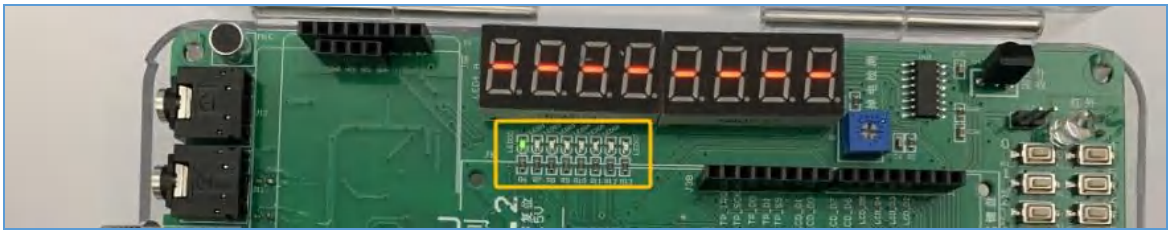
1、先点击 Run（运行）按钮，程序会停在如下图所示断点处（准备内部自动触发 INT2）：



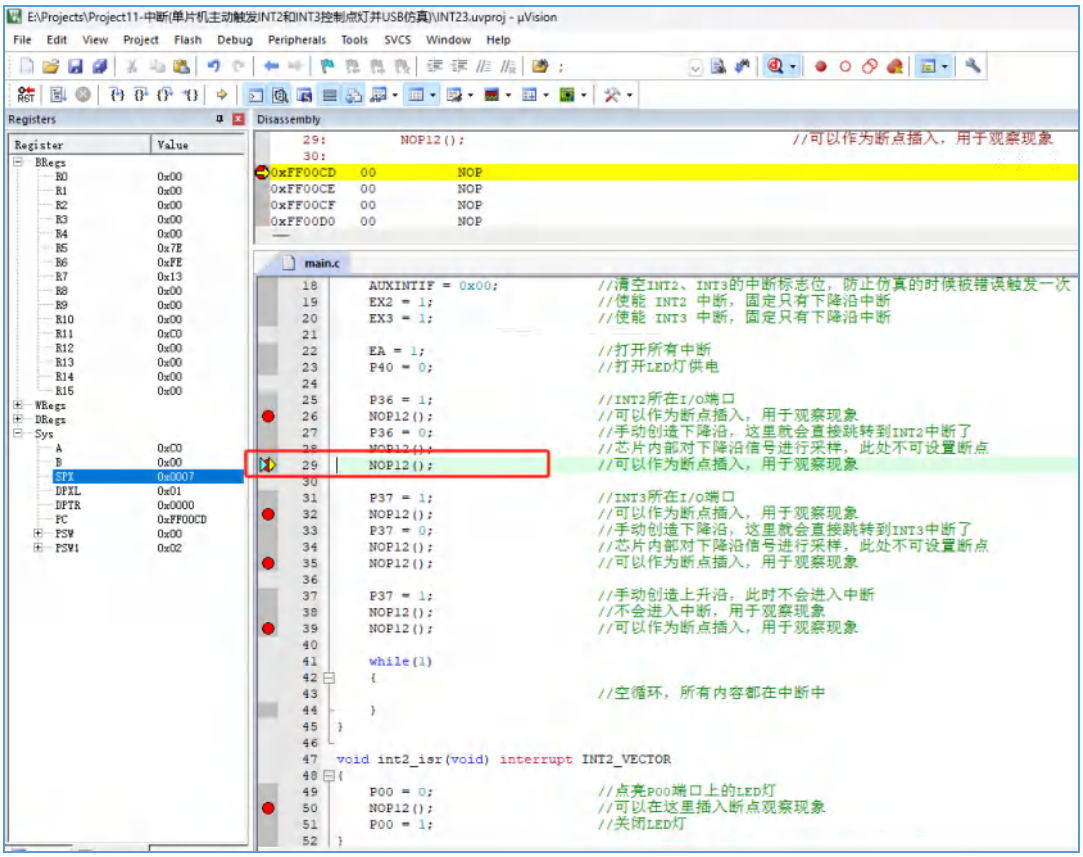
2、点击 Run（运行）按钮，继续运行，单片机内部主动触发 INT2/P3.6，会进入 INT2 中断，点亮 P0.0 灯，程序会停在如下图所示断点处：



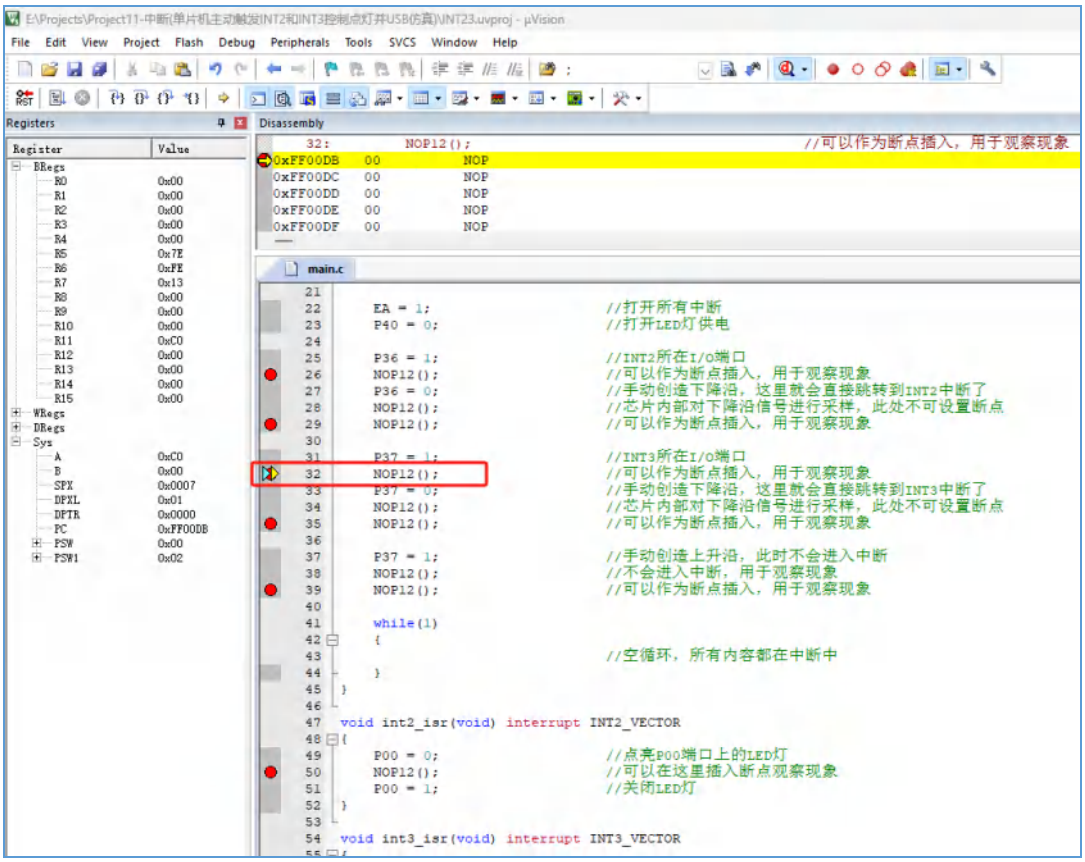
实验箱如图，点亮 P0.0 灯：



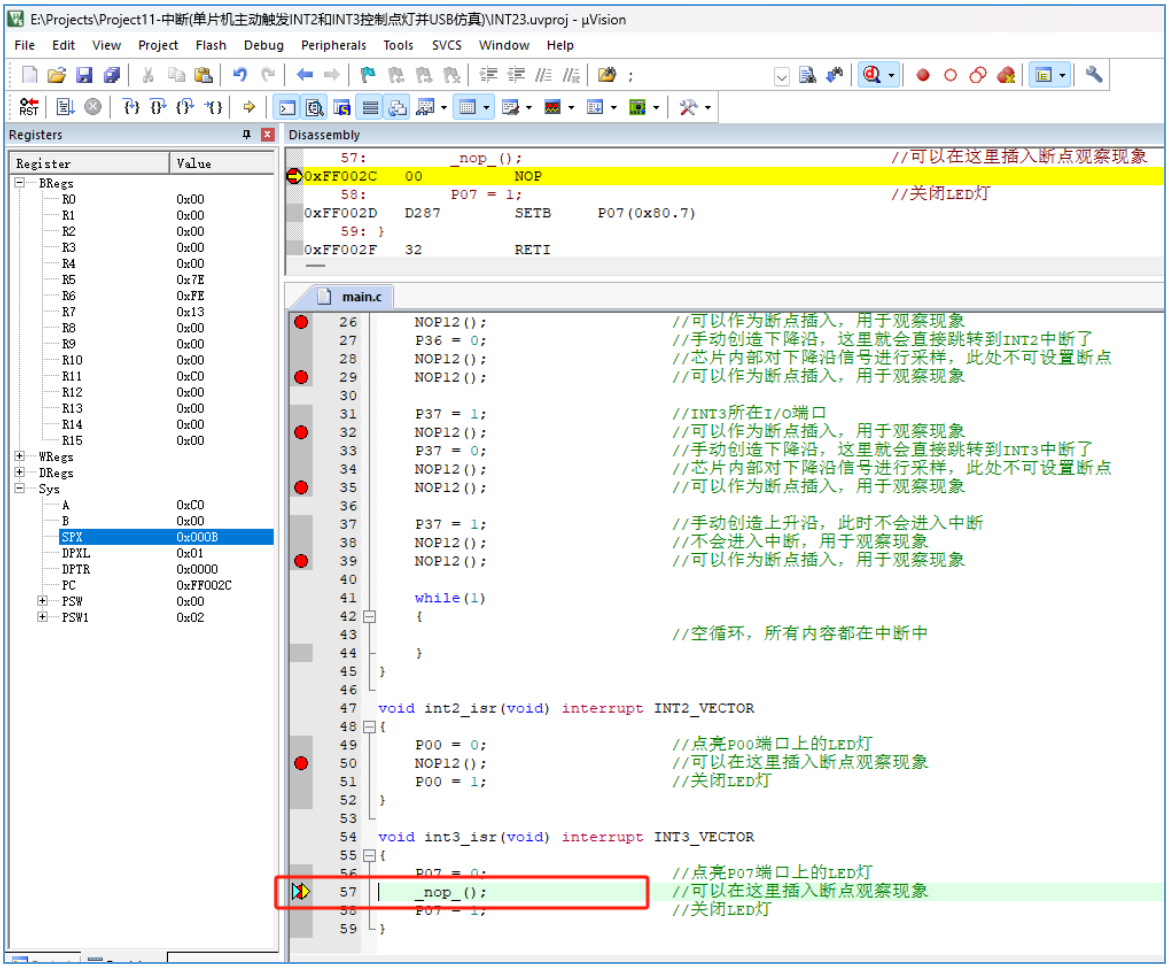
3、点击 Run（运行）按钮，继续运行。实验箱上 P0.0 灯灭，程序会停在如下图所示断点处（中断返回）：



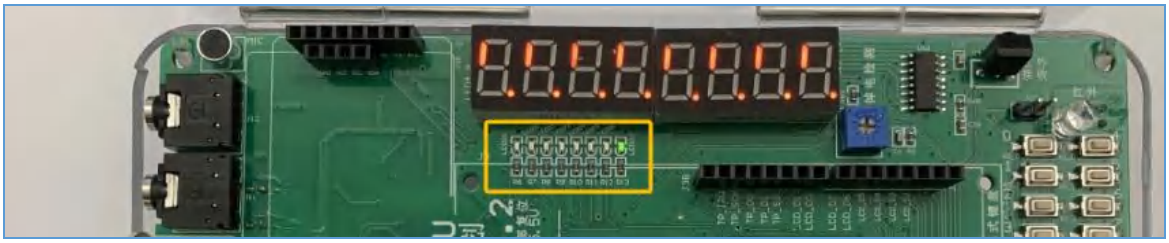
4、点击 Run（运行）按钮，继续运行，程序会停在如下图所示断点处（准备内部自动触发 INT3）：



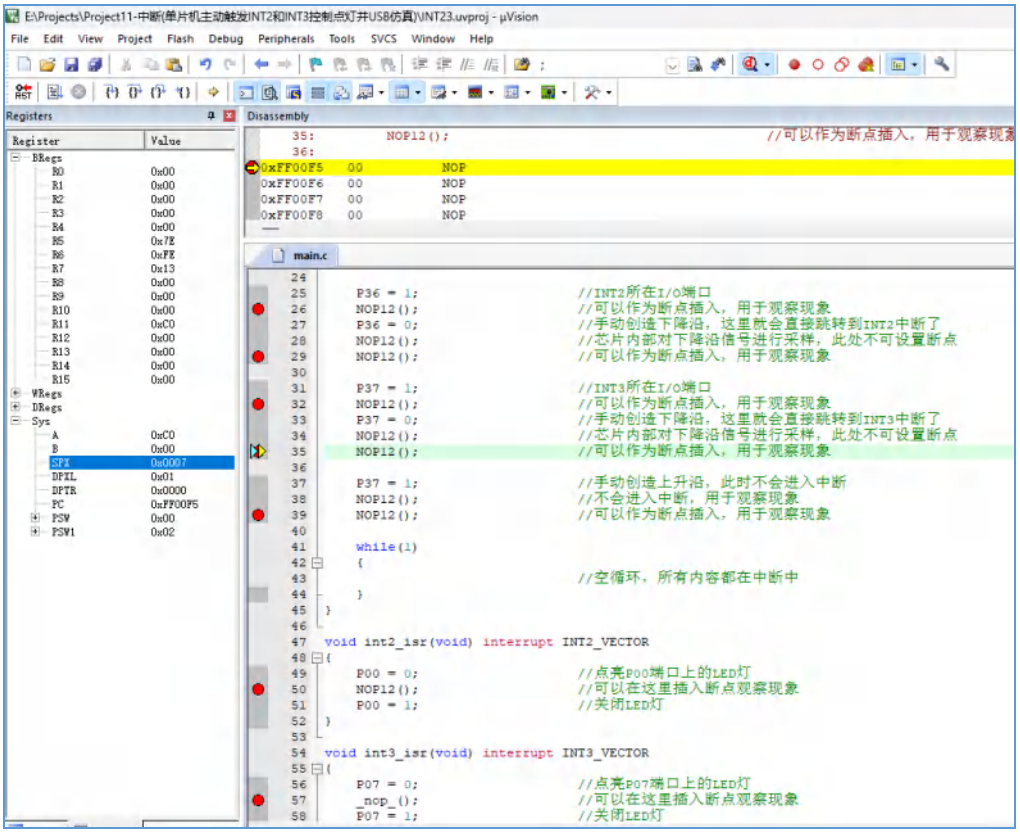
5、点击 Run（运行）按钮，继续运行，单片机内部主动触发 INT3/P3.7，会进入 INT3 中断，点亮 P0.7 灯，程序会停在如下图所示断点处：



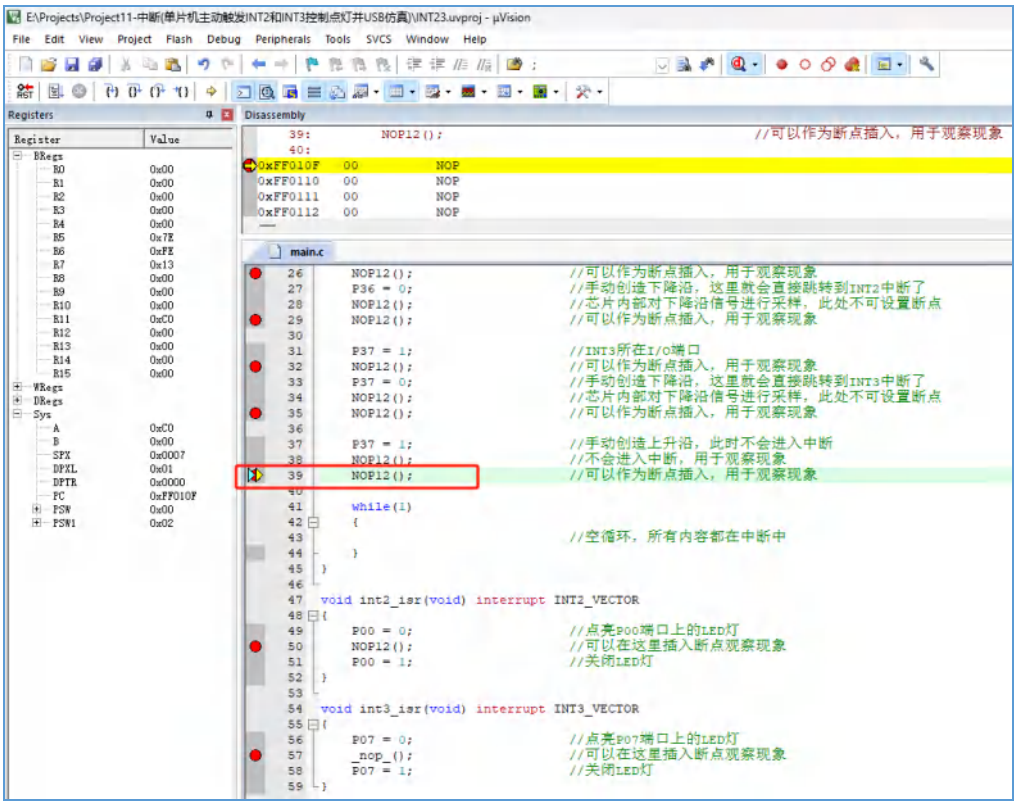
实验箱如图，点亮 P0.7 灯：



6、点击 Run（运行）按钮，继续运行，实验箱上 P0.7 灯灭，程序会停在如下图所示断点处（中断返回）：



7、点击 Run（运行）按钮，继续运行，程序会停在如下图所示断点处（测试上升沿不触发中断）：



点击 Run（运行）按钮，实验结束。

16　普通 I/O 口均可中断，不是传统外部中断

产品线	I/O 中断	I/O 中断优先级	I/O 中断唤醒功能
Ai8051U 系列	●	4 级	●

Ai8051U 系列支持所有的 I/O 中断，且支持 4 种中断模式：下降沿中断、上升沿中断、低电平中断、高电平中断。每组 I/O 口都有独立的中断入口地址，且每个 I/O 可独立设置中断模式。

特别的，如果需要同时检测某个信号的上升沿和下降沿，可将此信号同时连接到两个 I/O，其中一个 I/O 检测此信号的上升沿、另外一个 I/O 检测此信号的下降沿即可。

16.1　I/O 口中断相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
P0INTE	P0 口中断使能寄存器	7EFD00H	P07INTE	P06INTE	P05INTE	P04INTE	P03INTE	P02INTE	P01INTE	P00INTE	0000,0000
P1INTE	P1 口中断使能寄存器	7EFD01H	P17INTE	P16INTE	P15INTE	P14INTE	P13INTE	P12INTE	P11INTE	P10INTE	0000,0000
P2INTE	P2 口中断使能寄存器	7EFD02H	P27INTE	P26INTE	P25INTE	P24INTE	P23INTE	P22INTE	P21INTE	P20INTE	0000,0000
P3INTE	P3 口中断使能寄存器	7EFD03H	P37INTE	P36INTE	P35INTE	P34INTE	P33INTE	P32INTE	P31INTE	P30INTE	0000,0000
P4INTE	P4 口中断使能寄存器	7EFD04H	P47INTE	P46INTE	P45INTE	P44INTE	P43INTE	P42INTE	P41INTE	P40INTE	0000,0000
P5INTE	P5 口中断使能寄存器	7EFD05H	P57INTE	P56INTE	P55INTE	P54INTE	P53INTE	P52INTE	P51INTE	P50INTE	xx00,0000
P6INTE	P6 口中断使能寄存器	7EFD06H	P67INTE	P66INTE	P65INTE	P64INTE	P63INTE	P62INTE	P61INTE	P60INTE	0000,0000
P7INTE	P7 口中断使能寄存器	7EFD07H	P77INTE	P76INTE	P75INTE	P74INTE	P73INTE	P72INTE	P71INTE	P70INTE	0000,0000
P0INTF	P0 口中断标志寄存器	7EFD10H	P07INTF	P06INTF	P05INTF	P04INTF	P03INTF	P02INTF	P01INTF	P00INTF	0000,0000
P1INTF	P1 口中断标志寄存器	7EFD11H	P17INTF	P16INTF	P15INTF	P14INTF	P13INTF	P12INTF	P11INTF	P10INTF	0000,0000
P2INTF	P2 口中断标志寄存器	7EFD12H	P27INTF	P26INTF	P25INTF	P24INTF	P23INTF	P22INTF	P21INTF	P20INTF	0000,0000
P3INTF	P3 口中断标志寄存器	7EFD13H	P37INTF	P36INTF	P35INTF	P34INTF	P33INTF	P32INTF	P31INTF	P30INTF	0000,0000
P4INTF	P4 口中断标志寄存器	7EFD14H	P47INTF	P46INTF	P45INTF	P44INTF	P43INTF	P42INTF	P41INTF	P40INTF	0000,0000
P5INTF	P5 口中断标志寄存器	7EFD15H	P57INTF	P56INTF	P55INTF	P54INTF	P53INTF	P52INTF	P51INTF	P50INTF	xx00,0000
P6INTF	P6 口中断标志寄存器	7EFD16H	P67INTF	P66INTF	P65INTF	P64INTF	P63INTF	P62INTF	P61INTF	P60INTF	0000,0000
P7INTF	P7 口中断标志寄存器	7EFD17H	P77INTF	P76INTF	P75INTF	P74INTF	P73INTF	P72INTF	P71INTF	P70INTF	0000,0000
P0IM0	P0 口中断模式寄存器 0	7EFD20H	P07IM0	P06IM0	P05IM0	P04IM0	P03IM0	P02IM0	P01IM0	P00IM0	0000,0000
P1IM0	P1 口中断模式寄存器 0	7EFD21H	P17IM0	P16IM0	P15IM0	P14IM0	P13IM0	P12IM0	P11IM0	P10IM0	0000,0000
P2IM0	P2 口中断模式寄存器 0	7EFD22H	P27IM0	P26IM0	P25IM0	P24IM0	P23IM0	P22IM0	P21IM0	P20IM0	0000,0000
P3IM0	P3 口中断模式寄存器 0	7EFD23H	P37IM0	P36IM0	P35IM0	P34IM0	P33IM0	P32IM0	P31IM0	P30IM0	0000,0000
P4IM0	P4 口中断模式寄存器 0	7EFD24H	P47IM0	P46IM0	P45IM0	P44IM0	P43IM0	P42IM0	P41IM0	P40IM0	0000,0000
P5IM0	P5 口中断模式寄存器 0	7EFD25H	P57IM0	P56IM0	P55IM0	P54IM0	P53IM0	P52IM0	P51IM0	P50IM0	xx00,0000
P6IM0	P6 口中断模式寄存器 0	7EFD26H	P67IM0	P66IM0	P65IM0	P64IM0	P63IM0	P62IM0	P61IM0	P60IM0	0000,0000
P7IM0	P7 口中断模式寄存器 0	7EFD27H	P77IM0	P76IM0	P75IM0	P74IM0	P73IM0	P72IM0	P71IM0	P70IM0	0000,0000
P0IM1	P0 口中断模式寄存器 1	7EFD30H	P07IM1	P06IM1	P05IM1	P04IM1	P03IM1	P02IM1	P01IM1	P00IM1	0000,0000
P1IM1	P1 口中断模式寄存器 1	7EFD31H	P17IM1	P16IM1	P15IM1	P14IM1	P13IM1	P12IM1	P11IM1	P10IM1	0000,0000
P2IM1	P2 口中断模式寄存器 1	7EFD32H	P27IM1	P26IM1	P25IM1	P24IM1	P23IM1	P22IM1	P21IM1	P20IM1	0000,0000

P3IM1	P3 口中断模式寄存器 1	7EFD33H	P37IM1	P36IM1	P35IM1	P34IM1	P33IM1	P32IM1	P31IM1	P30IM1	0000,0000
P4IM1	P4 口中断模式寄存器 1	7EFD34H	P47IM1	P46IM1	P45IM1	P44IM1	P43IM1	P42IM1	P41IM1	P40IM1	0000,0000
P5IM1	P5 口中断模式寄存器 1	7EFD35H	P57IM1	P56IM1	P55IM1	P54IM1	P53IM1	P52IM1	P51IM1	P50IM1	xx00,0000
P6IM1	P6 口中断模式寄存器 1	7EFD36H	P67IM1	P66IM1	P65IM1	P64IM1	P63IM1	P62IM1	P61IM1	P60IM1	0000,0000
P7IM1	P7 口中断模式寄存器 1	7EFD37H	P77IM1	P76IM1	P75IM1	P74IM1	P73IM1	P72IM1	P71IM1	P70IM1	0000,0000
PINIPL	I/O 口中断优先级低寄存器	7EFD40H	P7IP	P6IP	P5IP	P4IP	P3IP	P2IP	P1IP	P0IP	0000,0000
PINIPH	I/O 口中断优先级高寄存器	7EFD41H	P7IPH	P6IPH	P5IPH	P4IPH	P3IPH	P2IPH	P1IPH	P0IPH	0000,0000
P0WKUE	P0 口中断唤醒使能寄存器	7EFD42H	P07WKUE	P06WKUE	P05WKUE	P04WKUE	P03WKUE	P02WKUE	P01WKUE	P00WKUE	0000,0000
P1WKUE	P1 口中断唤醒使能寄存器	7EFD43H	P17WKUE	P16WKUE	P15WKUE	P14WKUE	P13WKUE	P12WKUE	P11WKUE	P10WKUE	0000,0000
P2WKUE	P2 口中断唤醒使能寄存器	7EFD44H	P27WKUE	P26WKUE	P25WKUE	P24WKUE	P23WKUE	P22WKUE	P21WKUE	P20WKUE	0000,0000
P3WKUE	P3 口中断唤醒使能寄存器	7EFD45H	P37WKUE	P36WKUE	P35WKUE	P34WKUE	P33WKUE	P32WKUE	P31WKUE	P30WKUE	0000,0000
P4WKUE	P4 口中断唤醒使能寄存器	7EFD46H	P47WKUE	P46WKUE	P45WKUE	P44WKUE	P43WKUE	P42WKUE	P41WKUE	P40WKUE	0000,0000
P5WKUE	P5 口中断唤醒使能寄存器	7EFD47H	P57WKUE	P56WKUE	P55WKUE	P54WKUE	P53WKUE	P52WKUE	P51WKUE	P50WKUE	xx00,0000
P6WKUE	P6 口中断唤醒使能寄存器	7EFD60H	P67WKUE	P66WKUE	P65WKUE	P64WKUE	P63WKUE	P62WKUE	P61WKUE	P60WKUE	0000,0000
P7WKUE	P7 口中断唤醒使能寄存器	7EFD61H	P77WKUE	P76WKUE	P75WKUE	P74WKUE	P73WKUE	P72WKUE	P71WKUE	P70WKUE	0000,0000

16.1.1 端口中断使能寄存器（PxINTE）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0INTE	7EFD00H	P07INTE	P06INTE	P05INTE	P04INTE	P03INTE	P02INTE	P01INTE	P00INTE
P1INTE	7EFD01H	P17INTE	P16INTE	P15INTE	P14INTE	P13INTE	P12INTE	P11INTE	P10INTE
P2INTE	7EFD02H	P27INTE	P26INTE	P25INTE	P24INTE	P23INTE	P22INTE	P21INTE	P20INTE
P3INTE	7EFD03H	P37INTE	P36INTE	P35INTE	P34INTE	P33INTE	P32INTE	P31INTE	P30INTE
P4INTE	7EFD04H	P47INTE	P46INTE	P45INTE	P44INTE	P43INTE	P42INTE	P41INTE	P40INTE
P5INTE	7EFD05H	P57INTE	P56INTE	P55INTE	P54INTE	P53INTE	P52INTE	P51INTE	P50INTE
P6INTE	7EFD06H	P67INTE	P66INTE	P65INTE	P64INTE	P63INTE	P62INTE	P61INTE	P60INTE
P7INTE	7EFD07H	P77INTE	P76INTE	P75INTE	P74INTE	P73INTE	P72INTE	P71INTE	P70INTE

PnINTE.x：端口中断使能控制位（n=0~7，x=0~7）

0：关闭 Pn.x 口中断功能

1：使能 Pn.x 口中断功能

16.1.2 端口中断标志寄存器（PxINTF）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0INTF	7EFD10H	P07INTF	P06INTF	P05INTF	P04INTF	P03INTF	P02INTF	P01INTF	P00INTF
P1INTF	7EFD11H	P17INTF	P16INTF	P15INTF	P14INTF	P13INTF	P12INTF	P11INTF	P10INTF
P2INTF	7EFD12H	P27INTF	P26INTF	P25INTF	P24INTF	P23INTF	P22INTF	P21INTF	P20INTF
P3INTF	7EFD13H	P37INTF	P36INTF	P35INTF	P34INTF	P33INTF	P32INTF	P31INTF	P30INTF
P4INTF	7EFD14H	P47INTF	P46INTF	P45INTF	P44INTF	P43INTF	P42INTF	P41INTF	P40INTF
P5INTF	7EFD15H	P57INTF	P56INTF	P55INTF	P54INTF	P53INTF	P52INTF	P51INTF	P50INTF
P6INTF	7EFD16H	P67INTF	P66INTF	P65INTF	P64INTF	P63INTF	P62INTF	P61INTF	P60INTF
P7INTF	7EFD17H	P77INTF	P76INTF	P75INTF	P74INTF	P73INTF	P72INTF	P71INTF	P70INTF

PnINTF.x：端口中断请求标志位（n=0~7，x=0~7）

0：Pn.x 口没有中断请求

1：Pn.x 口有中断请求，若使能中断，则会进入中断服务程序。标志位需软件清 0。

16.1.3 端口中断模式配置寄存器（PxIM0，PxIM1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0IM0	7EFD20H	P07IM0	P06IM0	P05IM0	P04IM0	P03IM0	P02IM0	P01IM0	P00IM0
P0IM1	7EFD30H	P07IM1	P06IM1	P05IM1	P04IM1	P03IM1	P02IM1	P01IM1	P00IM1
P1IM0	7EFD21H	P17IM0	P16IM0	P15IM0	P14IM0	P13IM0	P12IM0	P11IM0	P10IM0
P1IM1	7EFD31H	P17IM1	P16IM1	P15IM1	P14IM1	P13IM1	P12IM1	P11IM1	P10IM1
P2IM0	7EFD22H	P27IM0	P26IM0	P25IM0	P24IM0	P23IM0	P22IM0	P21IM0	P20IM0
P2IM1	7EFD32H	P27IM1	P26IM1	P25IM1	P24IM1	P23IM1	P22IM1	P21IM1	P20IM1
P3IM0	7EFD23H	P37IM0	P36IM0	P35IM0	P34IM0	P33IM0	P32IM0	P31IM0	P30IM0
P3IM1	7EFD33H	P37IM1	P36IM1	P35IM1	P34IM1	P33IM1	P32IM1	P31IM1	P30IM1
P4IM0	7EFD24H	P47IM0	P46IM0	P45IM0	P44IM0	P43IM0	P42IM0	P41IM0	P40IM0
P4IM1	7EFD34H	P47IM1	P46IM1	P45IM1	P44IM1	P43IM1	P42IM1	P41IM1	P40IM1
P5IM0	7EFD25H	P57IM0	P56IM0	P55IM0	P54IM0	P53IM0	P52IM0	P51IM0	P50IM0
P5IM1	7EFD35H	P57IM1	P56IM1	P55IM1	P54IM1	P53IM1	P52IM1	P51IM1	P50IM1
P6IM0	7EFD26H	P67IM0	P66IM0	P65IM0	P64IM0	P63IM0	P62IM0	P61IM0	P60IM0
P6IM1	7EFD36H	P67IM1	P66IM1	P65IM1	P64IM1	P63IM1	P62IM1	P61IM1	P60IM1
P7IM0	7EFD27H	P77IM0	P76IM0	P75IM0	P74IM0	P73IM0	P72IM0	P71IM0	P70IM0
P7IM1	7EFD37H	P77IM1	P76IM1	P75IM1	P74IM1	P73IM1	P72IM1	P71IM1	P70IM1

配置端口的模式

PnIM1.x	PnIM0.x	Pn.x 口中断模式	掉电唤醒支持
0	0	下降沿中断	支持
0	1	上升沿中断	支持
1	0	低电平中断	不支持
1	1	高电平中断	不支持

16.1.4 端口中断优先级控制寄存器（PINIPL，PINIPH）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PINIPL	7EFD60H	P7IP	P6IP	P5IP	P4IP	P3IP	P2IP	P1IP	P0IP
PINIPH	7EFD61H	P7IPH	P6IPH	P5IPH	P4IPH	P3IPH	P2IPH	P1IPH	P0IPH

PxIPH,PxIP：Px口中断优先级控制位

00：Px 口中断优先级为 0 级（最低级）

01：Px 口中断优先级为 1 级（较低级）

10：Px 口中断优先级为 2 级（较高级）

11：Px 口中断优先级为 3 级（最高级）

16.1.5 端口中断掉电唤醒使能寄存器（PxWKUE）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0WKUE	7EFD40H	P07WKUE	P06WKUE	P05WKUE	P04WKUE	P03WKUE	P02WKUE	P01WKUE	P00WKUE
P1WKUE	7EFD41H	P17WKUE	P16WKUE	P15WKUE	P14WKUE	P13WKUE	P12WKUE	P11WKUE	P10WKUE
P2WKUE	7EFD42H	P27WKUE	P26WKUE	P25WKUE	P24WKUE	P23WKUE	P22WKUE	P21WKUE	P20WKUE
P3WKUE	7EFD43H	P37WKUE	P36WKUE	P35WKUE	P34WKUE	P33WKUE	P32WKUE	P31WKUE	P30WKUE
P4WKUE	7EFD44H	P47WKUE	P46WKUE	P45WKUE	P44WKUE	P43WKUE	P42WKUE	P41WKUE	P40WKUE
P5WKUE	7EFD45H	P57WKUE	P56WKUE	P55WKUE	P54WKUE	P53WKUE	P52WKUE	P51WKUE	P50WKUE
P6WKUE	7EFD46H	P67WKUE	P66WKUE	P65WKUE	P64WKUE	P63WKUE	P62WKUE	P61WKUE	P60WKUE
P7WKUE	7EFD47H	P77WKUE	P76WKUE	P75WKUE	P74WKUE	P73WKUE	P72WKUE	P71WKUE	P70WKUE

PnxWKUE：端口中断掉电唤醒使能控制位（n=0~7，x=0~7）

0：关闭 Pn.x 口中断掉电唤醒功能

1：使能 Pn.x 口中断掉电唤醒功能**（注意：I/O 口中断模式必须选择下降沿中断模式或者上升沿中断模式才能实现掉电唤醒/主时钟停振被唤醒，高电平中断模式或者低电平中断模式无法掉电唤醒/主时钟停振被唤醒）**

16.2　范例程序

16.2.1　P0 口下降沿中断

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件
#include "intrins.h"                          //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void main()
{
    EAXFR = 1;                                //允许访问扩展的特殊寄存器，XFR
                                              //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                            //允许访问扩展的特殊寄存器，XFR
                                              //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                 //设置取程序代码等待时间，
                                              //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                //设置访问片内的 xdata 速度，
                                              //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P0IM0 = 0x00;                             //下降沿中断
    P0IM1 = 0x00;
    P0INTE = 0xff;                            //使能 P0 口中断

    EA = 1;

    while (1);
}

//由于中断向量大于 31，在 KEIL 中无法直接编译
//必须借用第 13 号中断入口地址
void common_isr() interrupt 13
{
    unsigned char intf;

    intf = P0INTF;
    if (intf)
    {
        P0INTF = 0x00;
        if (intf & 0x01)
        {
                                              //P0.0 口中断
        }
```

```
        if (intf & 0x02)
        {
                                                    //P0.1 口中断
        }
        if (intf & 0x04)
        {
                                                    //P0.2 口中断
        }
        if (intf & 0x08)
        {
                                                    //P0.3 口中断
        }
        if (intf & 0x10)
        {
                                                    //P0.4 口中断
        }
        if (intf & 0x20)
        {
                                                    //P0.5 口中断
        }
        if (intf & 0x40)
        {
                                                    //P0.6 口中断
        }
        if (intf & 0x80)
        {
                                                    //P0.7 口中断
        }
    }
}

// ISR.ASM
//将下面的代码保存为 ISR.ASM，然后将文件加入到项目中即可

            CSEG        AT 012BH                ;P0 口中断入口地址
            JMP         P0INT_ISR
P0INT_ISR:
            JMP         006BH                   ;借用 13 号中断的入口地址
            END
```

16.2.2　P1 口上升沿中断

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                             //头文件见下载软件
#include "intrins.h"                            //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void main()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
```

```
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P1IM0 = 0xff;                               //上升沿中断
    P1IM1 = 0x00;
    P1INTE = 0xff;                              //使能 P1 口中断

    EA = 1;

    while (1);
}

//由于中断向量大于 31，在 KEIL 中无法直接编译
//必须借用第 13 号中断入口地址
void common_isr() interrupt 13
{
    unsigned char intf;

    intf = P1INTF;
    if (intf)
    {
        P1INTF = 0x00;
        if (intf & 0x01)
        {
                                                //P1.0 口中断
        }
        if (intf & 0x02)
        {
                                                //P1.1 口中断
        }
        if (intf & 0x04)
        {
                                                //P1.2 口中断
        }
        if (intf & 0x08)
        {
                                                //P1.3 口中断
        }
        if (intf & 0x10)
        {
                                                //P1.4 口中断
        }
        if (intf & 0x20)
        {
```

```
                                                    //P1.5 口中断
        }
        if (intf & 0x40)
        {
                                                    //P1.6 口中断
        }
        if (intf & 0x80)
        {
                                                    //P1.7 口中断
        }
    }
}

// ISR.ASM
//将下面的代码保存为ISR.ASM，然后将文件加入到项目中即可

            CSEG        AT 0133H                    ;P1 口中断入口地址
            JMP         P1INT_ISR
P1INT_ISR:
            JMP         006BH                       ;借用 13 号中断的入口地址
            END
```

16.2.3 P2 口低电平中断

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                //头文件见下载软件
#include "intrins.h"                                //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void main()
{
    EAXFR = 1;                                      //允许访问扩展的特殊寄存器，XFR
                                                    //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                  //允许访问扩展的特殊寄存器，XFR
                                                    //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                       //设置取程序代码等待时间，
                                                    //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                      //设置访问片内的 xdata 速度，
                                                    //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;
```

```
    P2IM0 = 0x00;                                   //低电平中断
    P2IM1 = 0xff;
    P2INTE = 0xff;                                  //使能 P2 口中断

    EA = 1;

    while (1);
}

//由于中断向量大于 31，在 KEIL 中无法直接编译
//必须借用第 13 号中断入口地址
void common_isr() interrupt 13
{
    unsigned char intf;

    intf = P2INTF;
    if (intf)
    {
        P2INTF = 0x00;
        if (intf & 0x01)
        {
                                                    //P2.0 口中断
        }
        if (intf & 0x02)
        {
                                                    //P2.1 口中断
        }
        if (intf & 0x04)
        {
                                                    //P2.2 口中断
        }
        if (intf & 0x08)
        {
                                                    //P2.3 口中断
        }
        if (intf & 0x10)
        {
                                                    //P2.4 口中断
        }
        if (intf & 0x20)
        {
                                                    //P2.5 口中断
        }
        if (intf & 0x40)
        {
                                                    //P2.6 口中断
        }
        if (intf & 0x80)
        {
                                                    //P2.7 口中断
        }
    }
}

// ISR.ASM
//将下面的代码保存为 ISR.ASM，然后将文件加入到项目中即可

            CSEG        AT 013BH                    ;P2 口中断入口地址
            JMP         P2INT_ISR
```

```
P2INT_ISR:
        JMP     006BH                   ;借用 13 号中断的入口地址
        END
```

16.2.4 P3 口高电平中断

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                    //头文件见下载软件
#include "intrins.h"                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                           //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P3IM0 = 0xff;                       //高电平中断
    P3IM1 = 0xff;
    P3INTE = 0xff;                      //使能 P3 口中断

    EA = 1;

    while (1);
}

//由于中断向量大于 31，在 KEIL 中无法直接编译
//必须借用第 13 号中断入口地址
void common_isr() interrupt 13
{
    unsigned char intf;

    intf = P3INTF;
    if (intf)
    {
```

```
        P3INTF = 0x00;
        if (intf & 0x01)
        {
                                                    //P3.0 口中断
        }
        if (intf & 0x02)
        {
                                                    //P3.1 口中断
        }
        if (intf & 0x04)
        {
                                                    //P3.2 口中断
        }
        if (intf & 0x08)
        {
                                                    //P3.3 口中断
        }
        if (intf & 0x10)
        {
                                                    //P3.4 口中断
        }
        if (intf & 0x20)
        {
                                                    //P3.5 口中断
        }
        if (intf & 0x40)
        {
                                                    //P3.6 口中断
        }
        if (intf & 0x80)
        {
                                                    //P3.7 口中断
        }
    }
}

// ISR.ASM
//将下面的代码保存为 ISR.ASM，然后将文件加入到项目中即可

            CSEG        AT 0143H                ;P3 口中断入口地址
            JMP         P3INT_ISR
P3INT_ISR:
            JMP         006BH                   ;借用 13 号中断的入口地址
            END
```

17　定时器/计数器，16 位自动重装载定时器 + 8 位预分频 = 24 位

产品线	定时器					
	T0	T1	T2	T3	T4	T11
Ai8051U 系列	●24	●24	●24	●24	●24	●24

●16：定时器支持 16 位模式

●24：定时器支持 24 位模式（8 位预分频+16 位定时）

Ai8051U 系列单片机内部设置了 6 个 24 位定时器/计数器（8 位预分频+16 位计数）。6 个 16 位定时器 T0、T1、T2、T3、T4 和 T11 都具有计数方式和定时方式两种工作方式。对定时器/计数器 T0 和 T1，用它们在特殊功能寄存器 TMOD 中相对应的控制位 C/T 来选择 T0 或 T1 为定时器还是计数器。对定时器/计数器 T2，用特殊功能寄存器 AUXR 中的控制位 T2_C/T 来选择 T2 为定时器还是计数器。对定时器/计数器 T3，用特殊功能寄存器 T4T3M 中的控制位 T3_C/T 来选择 T3 为定时器还是计数器。对定时器/计数器 T4，用特殊功能寄存器 T4T3M 中的控制位 T4_C/T 来选择 T4 为定时器还是计数器。对定时器/计数器 T11，用特殊功能寄存器 T11CR 中的控制位 T11_C/T 来选择 T11 为定时器还是计数器。定时器/计数器的核心部件是一个加法计数器，其本质是对脉冲进行计数。只是计数脉冲来源不同：如果计数脉冲来自系统时钟，则为定时方式，此时定时器/计数器每 12 个时钟或者每 1 个时钟得到一个计数脉冲，计数值加 1；如果计数脉冲来自单片机外部引脚，则为计数方式，每来一个脉冲加 1。

当定时器/计数器 T0、T1 及 T2 工作在定时模式时，特殊功能寄存器 AUXR 中的 T0x12、T1x12 和 T2x12 分别决定是系统时钟/12 还是系统时钟/1（不分频）后让 T0、T1 和 T2 进行计数。当定时器/计数器 T3 和 T4 工作在定时模式时，特殊功能寄存器 T4T3M 中的 T3x12 和 T4x12 分别决定是系统时钟/12 还是系统时钟/1（不分频）后让 T3 和 T4 进行计数。当定时器/计数器 T11 工作在定时模式时，特殊功能寄存器 T11CR 中的 T11x12 决定是系统时钟/12 还是系统时钟/1（不分频）后让 T11 进行计数。当定时器/计数器工作在计数模式时，对外部脉冲计数不分频。

定时器/计数器 0 有 4 种工作模式：模式 0（16 位自动重装载模式），模式 1（16 位不可重装载模式），模式 2（8 位自动重装模式），模式 3（不可屏蔽中断的 16 位自动重装载模式）。定时器/计数器 1 除模式 3 外，其他工作模式与定时器/计数器 0 相同。T1 在模式 3 时无效，停止计数。定时器 T2 的工作模式固定为 16 位自动重装载模式。T2 可以当定时器使用，也可以当串口的波特率发生器和可编程时钟输出。定时器 3、定时器 4、定时器 11 与定时器 T2 一样，它们的工作模式固定为 16 位自动重装载模式。T3/T4 可以当定时器使用，也可以当串口的波特率发生器和可编程时钟输出。定时器 11 的工作模式固定为 16 位自动重装载模式。T11 可以当定时器使用，也可编程时钟输出。

17.1 定时器的相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
TCON	定时器控制寄存器	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
TMOD	定时器模式寄存器	89H	T1_GATE	T1_C/T	T1_M1	T1_M0	T0_GATE	T0_C/T	T0_M1	T0_M0	0000,0000
TL0	定时器 0 低 8 位寄存器	8AH									0000,0000
TL1	定时器 1 低 8 位寄存器	8BH									0000,0000
TH0	定时器 0 高 8 位寄存器	8CH									0000,0000
TH1	定时器 1 高 8 位寄存器	8DH									0000,0000
AUXR	辅助寄存器 1	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1BRT	0000,0001
INTCLKO	中断与时钟输出控制寄存器	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000
WKTCL	掉电唤醒定时器低字节	AAH									1111,1111
WKTCH	掉电唤醒定时器高字节	ABH	WKTEN								0111,1111
T4T3M	定时器 4/3 控制寄存器	DDH	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO	0000,0000
T4H	定时器 4 高字节	D2H									0000,0000
T4L	定时器 4 低字节	D3H									0000,0000
T3H	定时器 3 高字节	D4H									0000,0000
T3L	定时器 3 低字节	D5H									0000,0000
T2H	定时器 2 高字节	D6H									0000,0000
T2L	定时器 2 低字节	D7H									0000,0000

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
TM0PS	定时器 0 时钟预分频寄存器	7EFEA0H									0000,0000
TM1PS	定时器 1 时钟预分频寄存器	7EFEA1H									0000,0000
TM2PS	定时器 2 时钟预分频寄存器	7EFEA2H									0000,0000
TM3PS	定时器 3 时钟预分频寄存器	7EFEA3H									0000,0000
TM4PS	定时器 4 时钟预分频寄存器	7EFEA4H									0000,0000
T11CR	定时器 T11 控制寄存器	7EFE78H	T11R	T11_C/T	T11CLKO	T11x12	T11CS[1:0]		ET11I	T11IF	0000,0000
T11PS	定时器 T11 时钟预分频寄存器	7EFE79H									0000,0000
T11H	定时器 T11 高字节	7EFE7AH									0000,0000
T11L	定时器 T11 低字节	7EFE7BH									0000,0000

17.2 定时器 0/1，16 位自动重装载，+ 8 位预分频就是 24 位定时器

17.2.1 定时器 0/1 控制寄存器（TCON）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TCON	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1：T1溢出中断标志。T1被允许计数以后，从初值开始加1计数。当产生溢出时由硬件将TF1位置“1”，并向CPU请求中断，一直保持到CPU响应中断时，才由硬件清“0”（也可由查询软件清“0”）。

TR1：定时器T1的运行控制位。该位由软件置位和清零。当GATE（TMOD.7）=0，TR1=1时就允许T1开始计数，TR1=0时禁止T1计数。当GATE（TMOD.7）=1，TR1=1且INT1输入高电平时，才允许T1计数。

TF0：T0溢出中断标志。T0被允许计数以后，从初值开始加1计数，当产生溢出时，由硬件置“1”TF0，向CPU请求中断，一直保持CPU响应该中断时，才由硬件清0（也可由查询软件清0）。

TR0：定时器T0的运行控制位。该位由软件置位和清零。当GATE（TMOD.3）=0，TR0=1时就允许T0开始计数，TR0=0时禁止T0计数。当GATE（TMOD.3）=1，TR0=1且INT0输入高电平时，才允许T0计数，TR0=0时禁止T0计数。

IE1：外部中断1请求源（INT1/P3.3）标志。IE1=1，外部中断向CPU请求中断，当CPU响应该中断时由硬件清“0”IE1。

IT1：外部中断源1触发控制位。IT1=0，上升沿或下降沿均可触发外部中断1。IT1=1，外部中断1程控为下降沿触发方式。

IE0：外部中断0请求源（INT0/P3.2）标志。IE0=1外部中断0向CPU请求中断，当CPU响应外部中断时，由硬件清“0”IE0（边沿触发方式）。

IT0：外部中断源0触发控制位。IT0=0，上升沿或下降沿均可触发外部中断0。IT0=1，外部中断0程控为下降沿触发方式。

17.2.2　定时器 0/1 模式寄存器（TMOD）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TMOD	89H	T1_GATE	T1_C/T	T1_M1	T1_M0	T0_GATE	T0_C/T	T0_M1	T0_M0

T1_GATE：控制定时器1，置1时只有在INT1脚为高及TR1控制位置1时才可打开定时器/计数器1。

T0_GATE：控制定时器0，置1时只有在INT0脚为高及TR0控制位置1时才可打开定时器/计数器0。

T1_C/T：控制定时器1用作定时器或计数器，清0则用作定时器（对内部系统时钟进行计数），置1用作计数器（对引脚T1/P3.5外部脉冲进行计数）。

T0_C/T：控制定时器0用作定时器或计数器，清0则用作定时器（对内部系统时钟进行计数），置1用作计数器（对引脚T0/P3.4外部脉冲进行计数）。

T1_M1/T1_M0：定时器定时器/计数器1模式选择

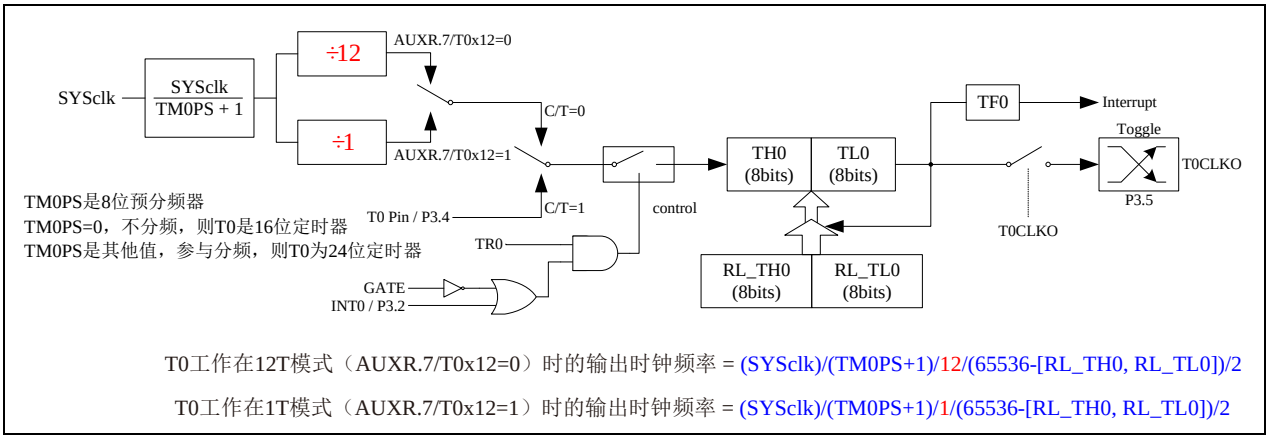
T1_M1	T1_M0	定时器/计数器1工作模式
0	0	16位自动重载模式 当[TH1,TL1]中的16位计数值溢出时，系统会自动将内部16位重载寄存器中的重载值装入[TH1,TL1]中。
0	1	16位不自动重载模式 当[TH1,TL1]中的16位计数值溢出时，定时器1将从0开始计数
1	0	8位自动重载模式 当TL1中的8位计数值溢出时，系统会自动将TH1中的重载值装入TL1中。
1	1	T1停止工作

T0_M1/T0_M0：定时器定时器/计数器0模式选择

T0_M1	T0_M0	定时器/计数器0工作模式
0	0	16位自动重载模式 当[TH0,TL0]中的16位计数值溢出时，系统会自动将内部16位重载寄存器中的重载值装入[TH0,TL0]中。
0	1	16位不自动重载模式 当[TH0,TL0]中的16位计数值溢出时，定时器0将从0开始计数
1	0	8位自动重载模式 当TL0中的8位计数值溢出时，系统会自动将TH0中的重载值装入TL0中。
1	1	不可屏蔽中断的16位自动重载模式 与模式0相同，不可屏蔽中断，中断优先级最高，高于其他所有中断的优先级，并且不可关闭，可用作操作系统的系统节拍定时器，或者系统监控定时器。

17.2.3 定时器 0 模式 0，16 位自动重装载，启动 8 位预分频就是 24 位定时器

此模式下定时器/计数器 0 作为可自动重装载的 16 位计数器，如下图所示：



定时器/计数器 0 的模式 0：16 位自动重装载模式，如果启动 8 位预分频 TM0PS 就是 24 位定时器

当 GATE=0（TMOD.3）时，如 TR0=1，则定时器计数。GATE=1 时，允许由外部输入 INT0 控制定时器 0，这样可实现脉宽测量。TR0 为 TCON 寄存器内的控制位，TCON 寄存器各位的具体功能描述见上节 TCON 寄存器的介绍。

当 C/T=0 时，多路开关连接到系统时钟的分频输出，T0 对内部系统时钟计数，T0 工作在定时方式。当 C/T=1 时，多路开关连接到外部脉冲输入 P3.4/T0，即 T0 工作在计数方式。

单片机的定时器 0 有两种计数速率：一种是 12T 模式，每 12 个时钟加 1，与传统 8051 单片机相同；另外一种是 1T 模式，每个时钟加 1，速度是传统 8051 单片机的 12 倍。T0 的速率由特殊功能寄存器 AUXR 中的 T0x12 决定，如果 T0x12=0，T0 则工作在 12T 模式；如果 T0x12=1，T0 则工作在 1T 模式

定时器 0 有两个隐藏的寄存器 RL_TH0 和 RL_TL0。RL_TH0 与 TH0 共有同一个地址，RL_TL0 与 TL0 共有同一个地址。当 TR0=0 即定时器/计数器 0 被禁止工作时，对 TL0 写入的内容会同时写入 RL_TL0，对 TH0 写入的内容也会同时写入 RL_TH0。当 TR0=1 即定时器/计数器 0 被允许工作时，对 TL0 写入内容，实际上不是写入当前寄存器 TL0 中，而是写入隐藏的寄存器 RL_TL0 中，对 TH0 写入内容，实际上也不是写入当前寄存器 TH0 中，而是写入隐藏的寄存器 RL_TH0，这样可以巧妙地实现 16 位重装载定时器。当读 TH0 和 TL0 的内容时，所读的内容就是 TH0 和 TL0 的内容，而不是 RL_TH0 和 RL_TL0 的内容。

当定时器 0 工作在模式 0（TMOD[1:0]/[M1,M0]=00B）时，[TH0,TL0]的溢出不仅置位 TF0，而且会自动将[RL_TH0,RL_TL0]的内容重新装入[TH0,TL0]。

当 T0CLKO/INT_CLKO.0=1 时，P3.5/T1 管脚配置为定时器 0 的时钟输出 T0CLKO。输出时钟频率为 T0 溢出率/2。

如果 C/T=0，定时器/计数器 T0 对内部系统时钟计数，则：

T0 工作在 1T 模式（AUXR.7/T0x12=1）时的输出时钟频率 = (SYSclk)/(TM0PS+1)/1/(65536-[RL_TH0, RL_TL0])/2

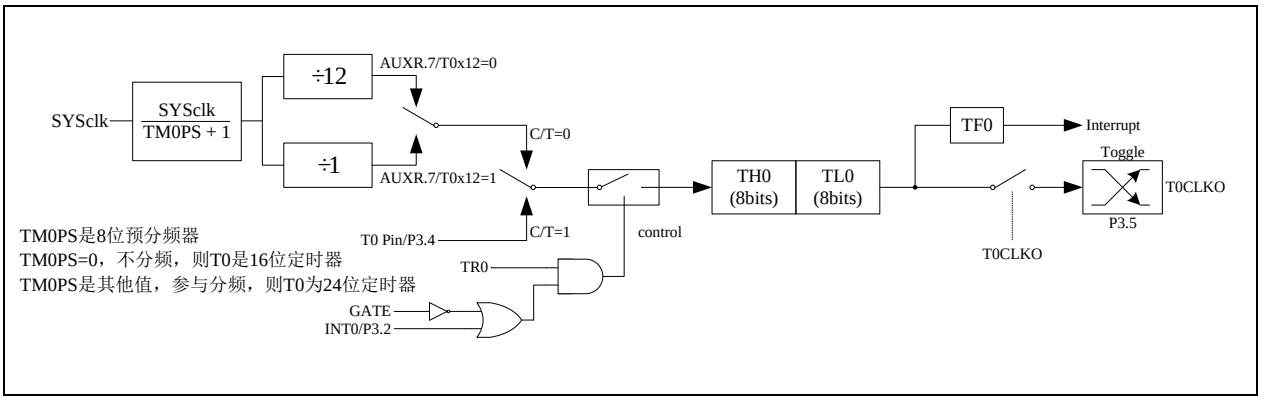
T0 工作在 12T 模式（AUXR.7/T0x12=0）时的输出时钟频率 = (SYSclk)/(TM0PS+1)/12/(65536-[RL_TH0, RL_TL0])/2

如果 C/T=1，定时器/计数器 T0 是对外部脉冲输入(P3.4/T0)计数，则：

输出时钟频率 = (T0_Pin_CLK) / (65536-[RL_TH0, RL_TL0])/2

17.2.4　定时器 0 模式 1（16 位不可重装载模式）

此模式下定时器/计数器 0 工作在 16 位不可重装载模式，如下图所示：



定时器/计数器 0 的模式 1：16 位不可重装载模式

此模式下，定时器/计数器 0 配置为 16 位不可重装载模式，由 TL0 的 8 位和 TH0 的 8 位所构成。TL0 的 8 位溢出向 TH0 进位，TH0 计数溢出置位 TCON 中的溢出标志位 TF0。

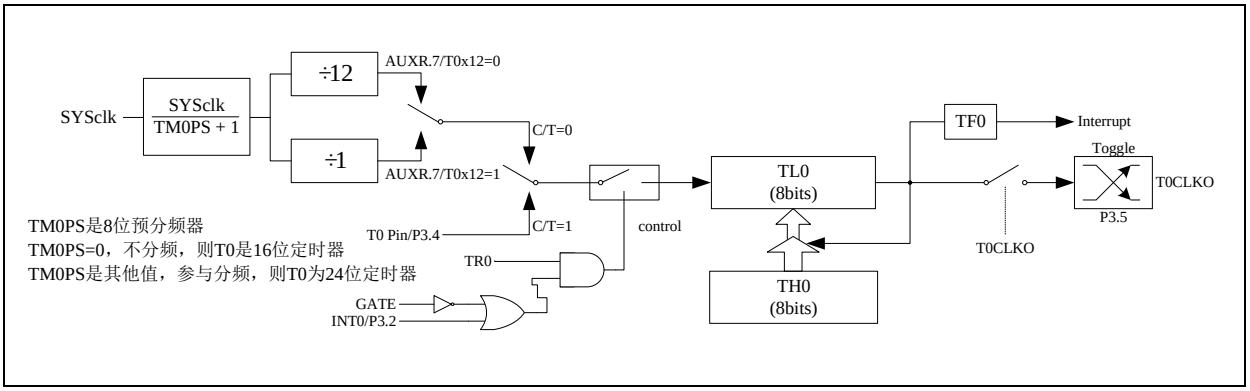
当 GATE=0(TMOD.3)时，如 TR0=1，则定时器计数。GATE=1 时，允许由外部输入 INT0 控制定时器 0，这样可实现脉宽测量。TR0 为 TCON 寄存器内的控制位，TCON 寄存器各位的具体功能描述见上节 TCON 寄存器的介绍。

当 C/T=0 时，多路开关连接到系统时钟的分频输出，T0 对内部系统时钟计数，T0 工作在定时方式。当 C/T=1 时，多路开关连接到外部脉冲输入 P3.4/T0，即 T0 工作在计数方式。

单片机的定时器 0 有两种计数速率：一种是 12T 模式，每 12 个时钟加 1，与传统 8051 单片机相同；另外一种是 1T 模式，每个时钟加 1，速度是传统 8051 单片机的 12 倍。T0 的速率由特殊功能寄存器 AUXR 中的 T0x12 决定，如果 T0x12=0，T0 则工作在 12T 模式；如果 T0x12=1，T0 则工作在 1T 模式。

17.2.5　定时器 0 模式 2（8 位自动重装载模式）

此模式下定时器/计数器 0 作为可自动重装载的 8 位计数器，如下图所示：



定时器/计数器 0 的模式 2：8 位自动重装载模式

TL0 的溢出不仅置位 TF0，而且将 TH0 的内容重新装入 TL0，TH0 内容由软件预置，重装时 TH0 内容不变。

当 T0CLKO/INT_CLKO.0=1 时，P3.5/T1 管脚配置为定时器 0 的时钟输出 T0CLKO。输出时钟频率为 T0 溢出率/2。

如果 C/T=0，定时器/计数器 T0 对内部系统时钟计数，则：

T0 工作在 1T 模式（AUXR.7/T0x12=1）时的输出时钟频率 = (SYSclk)/(TM0PS+1)/(256-TH0)/2

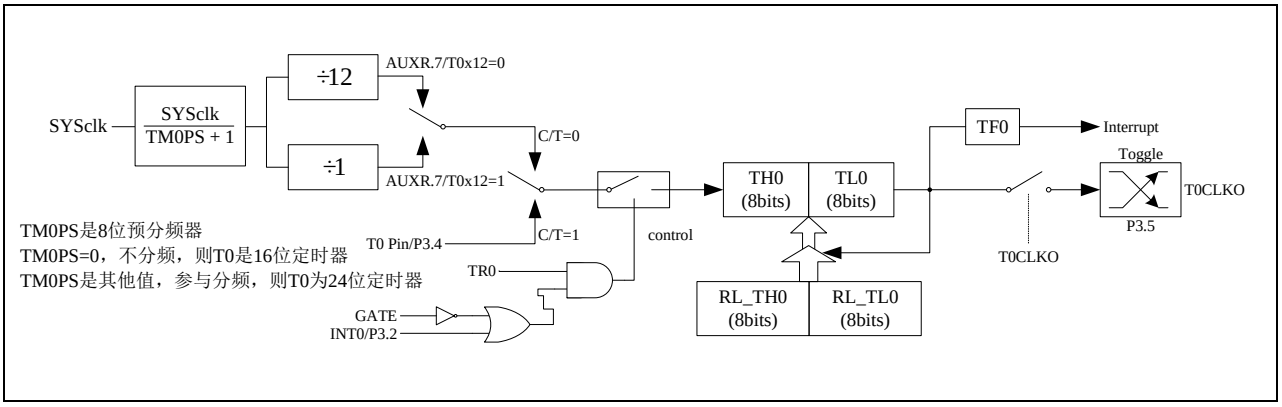
T0 工作在 12T 模式（AUXR.7/T0x12=0）时的输出时钟频率 = (SYSclk)/(TM0PS+1)/12/(256-TH0)/2

如果 C/T=1，定时器/计数器 T0 是对外部脉冲输入(P3.4/T0)计数，则：

输出时钟频率 = (T0_Pin_CLK) / (256-TH0)/2

17.2.6 定时器 0 模式 3（不可屏蔽中断 16 位自动重装载，实时操作系统节拍器）

对定时器/计数器 0，其工作模式模式 3 与工作模式 0 是一样的（下图定时器模式 3 的原理图，与工作模式 0 是一样的）。唯一不同的是：当定时器/计数器 0 工作在模式 3 时，只需允许 ET0/IE.1(定时器/计数器 0 中断允许位)，不需要允许 EA/IE.7(总中断使能位)就能打开定时器/计数器 0 的中断，此模式下的定时器/计数器 0 中断与总中断使能位 EA 无关，一旦工作在模式 3 下的定时器/计数器 0 中断被打开(ET0=1)，那么该中断是不可屏蔽的，该中断的优先级是最高的，即该中断不能被任何中断所打断，而且该中断打开后既不受 EA/IE.7 控制也不再受 ET0 控制，当 EA=0 或 ET0=0 时都不能屏蔽此中断。故将此模式称为不可屏蔽中断的 16 位自动重装载模式。

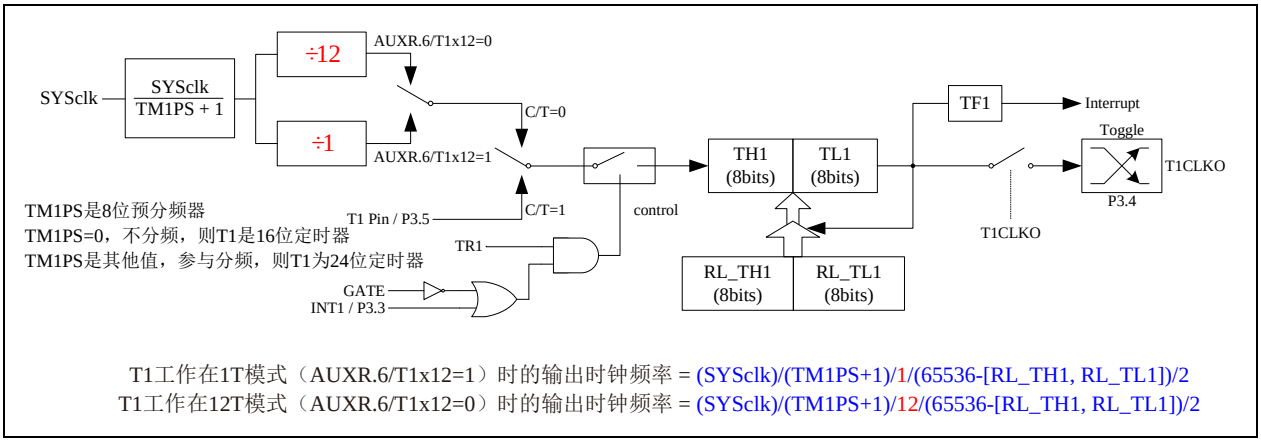


定时器/计数器 0 的模式 3：不可屏蔽中断的 16 位自动重装载模式

注意：当定时器/计数器 0 工作在模式 3(不可屏蔽中断的 16 位自动重装载模式)时，不需要允许 EA/IE.7(总中断使能位)，只需允许 ET0/IE.1(定时器/计数器 0 中断允许位)就能打开定时器/计数器 0 的中断，此模式下的定时器/计数器 0 中断与总中断使能位 EA 无关。一旦此模式下的定时器/计数器 0 中断被打开后，该定时器/计数器 0 中断优先级就是最高的，它不能被其它任何中断所打断(不管是比定时器/计数器 0 中断优先级低的中断还是比其优先级高的中断，都不能打断此时的定时器/计数器 0 中断)，而且该中断打开后既不受 EA/IE.7 控制也不再受 ET0 控制了，清零 EA 或 ET0 都不能关闭此中断。

17.2.7　定时器 1 模式 0，16 位自动重装载，启动 8 位预分频就是 24 位定时器

此模式下定时器/计数器 1 作为可自动重装载的 16 位计数器，如下图所示：



定时器/计数器 1 的模式 0：16 位自动重装载模式，如果启动 8 位预分频 TM1PS 就是 24 位定时器

当 GATE=0（TMOD.7）时，如 TR1=1，则定时器计数。GATE=1 时，允许由外部输入 INT1 控制定时器 1，这样可实现脉宽测量。TR1 为 TCON 寄存器内的控制位，TCON 寄存器各位的具体功能描述见上节 TCON 寄存器的介绍。

当 C/T=0 时，多路开关连接到系统时钟的分频输出，T1 对内部系统时钟计数，T1 工作在定时方式。当 C/T=1 时，多路开关连接到外部脉冲输入 P3.5/T1，即 T1 工作在计数方式。

单片机的定时器 1 有两种计数速率：一种是 12T 模式，每 12 个时钟加 1，与传统 8051 单片机相同；另外一种是 1T 模式，每个时钟加 1，速度是传统 8051 单片机的 12 倍。T1 的速率由特殊功能寄存器 AUXR 中的 T1x12 决定，如果 T1x12=0，T1 则工作在 12T 模式；如果 T1x12=1，T1 则工作在 1T 模式

定时器 1 有两个隐藏的寄存器 RL_TH1 和 RL_TL1。RL_TH1 与 TH1 共有同一个地址，RL_TL1 与 TL1 共有同一个地址。当 TR1=0 即定时器/计数器 1 被禁止工作时，对 TL1 写入的内容会同时写入 RL_TL1，对 TH1 写入的内容也会同时写入 RL_TH1。当 TR1=1 即定时器/计数器 1 被允许工作时，对 TL1 写入内容，实际上不是写入当前寄存器 TL1 中，而是写入隐藏的寄存器 RL_TL1 中，对 TH1 写入内容，实际上也不是写入当前寄存器 TH1 中，而是写入隐藏的寄存器 RL_TH1，这样可以巧妙地实现 16 位重装载定时器。当读 TH1 和 TL1 的内容时，所读的内容就是 TH1 和 TL1 的内容，而不是 RL_TH1 和 RL_TL1 的内容。

当定时器 1 工作在模式 1（TMOD[5:4]/[M1,M0]=00B）时，[TH1,TL1]的溢出不仅置位 TF1，而且会自动将[RL_TH1,RL_TL1]的内容重新装入[TH1,TL1]。

当 T1CLKO/INT_CLKO.1=1 时，P3.4/T0 管脚配置为定时器 1 的时钟输出 T1CLKO。输出时钟频率为 T1 溢出率/2。

如果 C/T=0，定时器/计数器 T1 对内部系统时钟计数，则：

T1 工作在 1T 模式（AUXR.6/T1x12=1）时的输出时钟频率 = (SYSclk)/(TM1PS+1)/1/(65536-[RL_TH1, RL_TL1])/2

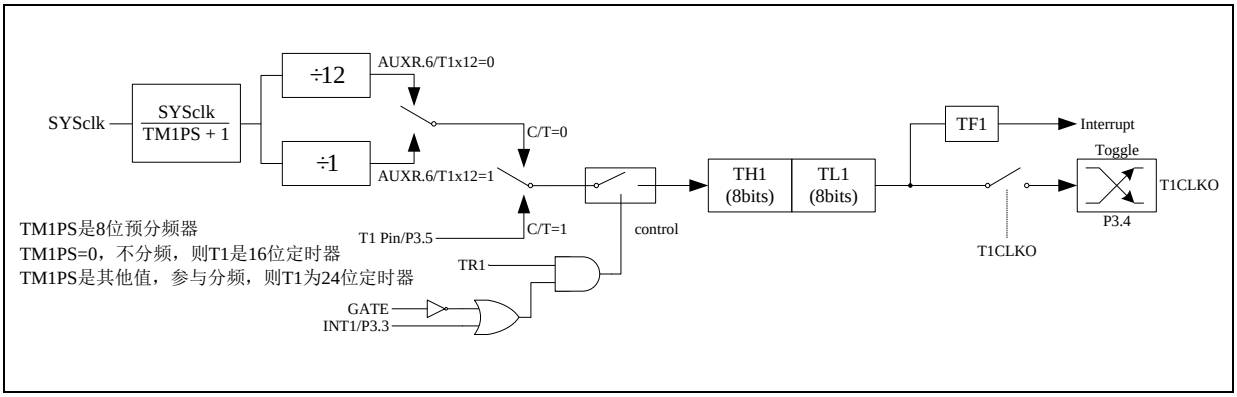
T1 工作在 12T 模式（AUXR.6/T1x12=0）时的输出时钟频率 = (SYSclk)/(TM1PS+1)/12/(65536-[RL_TH1, RL_TL1])/2

如果 C/T=1，定时器/计数器 T1 是对外部脉冲输入(P3.5/T1)计数，则：

输出时钟频率 = (T1_Pin_CLK) / (65536-[RL_TH1, RL_TL1])/2

17.2.8　定时器 1 模式 1（16 位不可重装载模式）

此模式下定时器/计数器 1 工作在 16 位不可重装载模式，如下图所示：



定时器/计数器 1 的模式 1：16 位不可重装载模式

此模式下，定时器/计数器 1 配置为 16 位不可重装载模式，由 TL1 的 8 位和 TH1 的 8 位所构成。TL1 的 8 位溢出向 TH1 进位，TH1 计数溢出置位 TCON 中的溢出标志位 TF1。

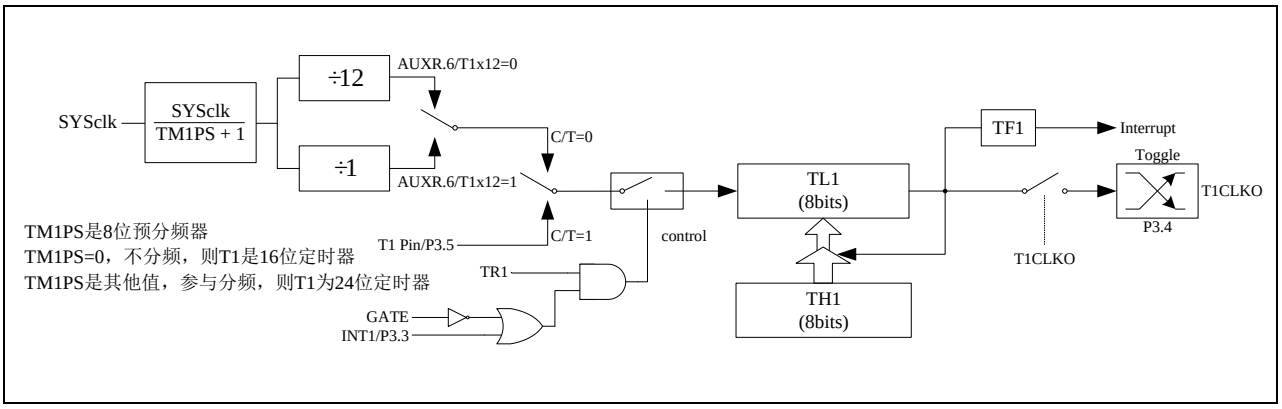
当 GATE=0(TMOD.7)时，如 TR1=1，则定时器计数。GATE=1 时，允许由外部输入 INT1 控制定时器 1，这样可实现脉宽测量。TR1 为 TCON 寄存器内的控制位，TCON 寄存器各位的具体功能描述见上节 TCON 寄存器的介绍。

当 C/T=0 时，多路开关连接到系统时钟的分频输出，T1 对内部系统时钟计数，T1 工作在定时方式。当 C/T=1 时，多路开关连接到外部脉冲输入 P3.5/T1，即 T1 工作在计数方式。

单片机的定时器 1 有两种计数速率：一种是 12T 模式，每 12 个时钟加 1，与传统 8051 单片机相同；另外一种是 1T 模式，每个时钟加 1，速度是传统 8051 单片机的 12 倍。T1 的速率由特殊功能寄存器 AUXR 中的 T1x12 决定，如果 T1x12=0，T1 则工作在 12T 模式；如果 T1x12=1，T1 则工作在 1T 模式。

17.2.9　定时器 1 模式 2（8 位自动重装载模式）

此模式下定时器/计数器 1 作为可自动重装载的 8 位计数器，如下图所示：



定时器/计数器 1 的模式 2：8 位自动重装载模式

TL1 的溢出不仅置位 TF1，而且将 TH1 的内容重新装入 TL1，TH1 内容由软件预置，重装时 TH1 内容不变。

当 T1CLKO/INT_CLKO.1=1 时，P3.4/T0 管脚配置为定时器 1 的时钟输出 T1CLKO。输出时钟频率为 T1 溢出率/2。

如果 C/T=0，定时器/计数器 T1 对内部系统时钟计数，则：

T1 工作在 1T 模式（AUXR.6/T1x12=1）时的输出时钟频率 = (SYSclk)/(TM1PS+1)/(256-TH1)/2

T1 工作在 12T 模式（AUXR.6/T1x12=0）时的输出时钟频率 = (SYSclk)/(TM1PS+1)/12/(256-TH1)/2

如果 C/T=1，定时器/计数器 T1 是对外部脉冲输入(P3.5/T1)计数，则：

输出时钟频率 = (T1_Pin_CLK) / (256-TH1)/2

17.2.10 定时器 0 计数寄存器（TL0，TH0）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TL0	8AH								
TH0	8CH								

当定时器/计数器0工作在16位模式（模式0、模式1、模式3）时，TL0和TH0组合成为一个16位寄存器，TL0为低字节，TH0为高字节。若为8位模式（模式2）时，TL0和TH0为两个独立的8位寄存器。

17.2.11 定时器 1 计数寄存器（TL1，TH1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TL1	8BH								
TH1	8DH								

当定时器/计数器1工作在16位模式（模式0、模式1）时，TL1和TH1组合成为一个16位寄存器，TL1为低字节，TH1为高字节。若为8位模式（模式2）时，TL1和TH1为两个独立的8位寄存器。

17.2.12 辅助寄存器 1（AUXR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1BRT

T0x12：定时器0速度控制位

0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）

1：1T 模式，即 CPU 时钟不分频分频（FOSC/1）

T1x12：定时器1速度控制位

0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）

1：1T 模式，即 CPU 时钟不分频分频（FOSC/1）

17.2.13 中断与时钟输出控制寄存器（INTCLKO）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INTCLKO	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO

T0CLKO：定时器0时钟输出控制

0：关闭时钟输出

1：使能 P3.5 口的是定时器 0 时钟输出功能

当定时器 0 计数发生溢出时，P3.5 口的电平自动发生翻转。

T1CLKO：定时器1时钟输出控制

0：关闭时钟输出

1：使能 P3.4 口的是定时器 1 时钟输出功能

当定时器 1 计数发生溢出时，P3.4 口的电平自动发生翻转。

17.2.14 定时器 0 的 8 位预分频寄存器（TM0PS）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TM0PS	7EFEA0H								

定时器0的时钟 ＝ 系统时钟SYSclk ÷ (TM0PS + 1)

17.2.15 定时器 1 的 8 位预分频寄存器（TM1PS）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TM1PS	7EFEA1H								

定时器1的时钟 ＝ 系统时钟SYSclk ÷ (TM1PS + 1)

17.2.16 定时器 0 计算公式

定时器模式	定时器速度	周期计算公式
模式0/3 （16位自动重载）	1T	定时器周期 = $\frac{65536 - [TH0, TL0]}{SYSclk/(TM0PS+1)}$ （自动重载）
	12T	定时器周期 = $\frac{65536 - [TH0, TL0]}{SYSclk/(TM0PS+1)} \times 12$ （自动重载）
模式1 （16位不自动重载）	1T	定时器周期 = $\frac{65536 - [TH0, TL0]}{SYSclk/(TM0PS+1)}$ （需软件装载）
	12T	定时器周期 = $\frac{65536 - [TH0, TL0]}{SYSclk/(TM0PS+1)} \times 12$ （需软件装载）
模式2 （8位自动重载）	1T	定时器周期 = $\frac{256 - TH0}{SYSclk/(TM0PS+1)}$ （自动重载）
	12T	定时器周期 = $\frac{256 - TH0}{SYSclk/(TM0PS+1)} \times 12$ （自动重载）

17.2.17　定时器 1 计算公式

定时器模式	定时器速度	周期计算公式
模式0 （16位自动重载）	1T	定时器周期 = $\frac{65536 - [TH1, TL1]}{SYSclk/(TM1PS+1)}$　（自动重载）
	12T	定时器周期 = $\frac{65536 - [TH1, TL1]}{SYSclk/(TM1PS+1)} \times 12$　（自动重载）
模式1 （16位不自动重载）	1T	定时器周期 = $\frac{65536 - [TH1, TL1]}{SYSclk/(TM1PS+1)}$　（需软件装载）
	12T	定时器周期 = $\frac{65536 - [TH1, TL1]}{SYSclk/(TM1PS+1)} \times 12$　（需软件装载）
模式2 （8位自动重载）	1T	定时器周期 = $\frac{256 - TH1}{SYSclk/(TM1PS+1)}$　（自动重载）
	12T	定时器周期 = $\frac{256 - TH1}{SYSclk/(TM1PS+1)} \times 12$　（自动重载）

17.3 定时器 2，16 位自动重装载，+ 8 位预分频就是 24 位定时器

17.3.1 辅助寄存器 1（AUXR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1BRT

T2R：定时器2的运行控制位

0：定时器 2 停止计数

1：定时器 2 开始计数

T2_C/T：控制定时器2用作定时器或计数器，清0则用作定时器（对内部系统时钟进行计数），置1用作计数器（对引脚T2/P1.0外部脉冲进行计数）。

T2x12：定时器2速度控制位

0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）

1：1T 模式，即 CPU 时钟不分频分频（FOSC/1）

17.3.2 中断与时钟输出控制寄存器（INTCLKO）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INTCLKO	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO

T2CLKO：定时器2时钟输出控制

0：关闭时钟输出

1：使能 P1.1 口的是定时器 2 时钟输出功能

当定时器 2 计数发生溢出时，P1.1 口的电平自动发生翻转。

17.3.3 定时器 2 计数寄存器（T2L，T2H）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T2L	D7H								
T2H	D6H								

定时器/计数器2的工作模式固定为16位重载模式，T2L和T2H组合成为一个16位寄存器，T2L为低字节，T2H为高字节。当[T2H,T2L]中的16位计数值溢出时，系统会自动将内部16位重载寄存器中的重载值装入[T2H,T2L]中。

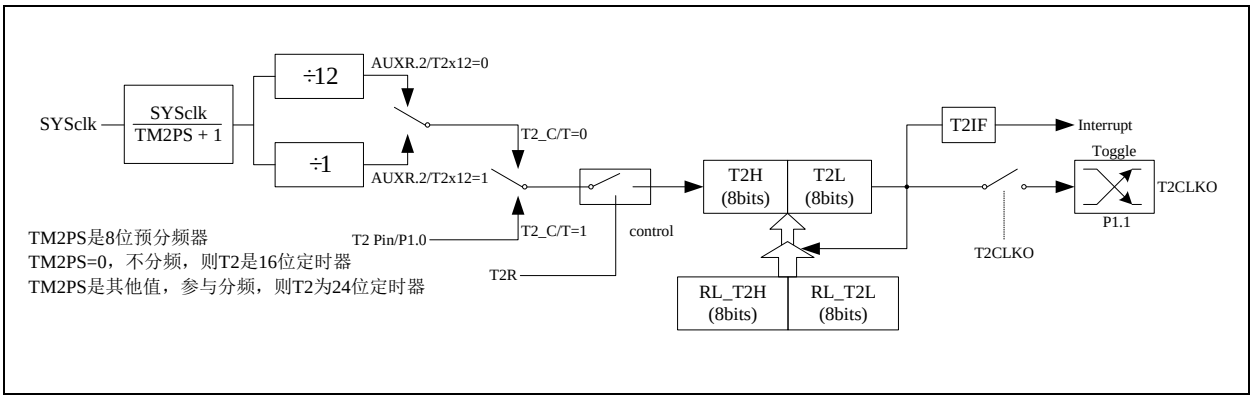
17.3.4 定时器 2 的 8 位预分频寄存器（TM2PS）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TM2PS	7EFEA2H								

定时器2的时钟 ＝ 系统时钟SYSclk ÷ (TM2PS + 1)

17.3.5　定时器 2 工作模式

定时器/计数器2的原理框图如下：



定时器/计数器 2 的工作模式：16 位自动重装载模式

T2R/AUXR.4 为 AUXR 寄存器内的控制位，AUXR 寄存器各位的具体功能描述见上节 AUXR 寄存器的介绍。

当 T2_C/T=0 时，多路开关连接到系统时钟输出，T2 对内部系统时钟计数，T2 工作在定时方式。当 T2_C/T=1 时，多路开关连接到外部脉冲输 T2，即 T2 工作在计数方式。

单片机的定时器 2 有两种计数速率：一种是 12T 模式，每 12 个时钟加 1，与传统 8051 单片机相同；另外一种是 1T 模式，每个时钟加 1，速度是传统 8051 单片机的 12 倍。T2 的速率由特殊功能寄存器 AUXR 中的 T2x12 决定，如果 T2x12=0，T2 则工作在 12T 模式；如果 T2x12=1，T2 则工作在 1T 模式

定时器 2 有两个隐藏的寄存器 RL_T2H 和 RL_T2L。RL_T2H 与 T2H 共有同一个地址，RL_T2L 与 T2L 共有同一个地址。当 T2R=0 即定时器/计数器 2 被禁止工作时，对 T2L 写入的内容会同时写入 RL_T2L，对 T2H 写入的内容也会同时写入 RL_T2H。当 T2R=1 即定时器/计数器 2 被允许工作时，对 T2L 写入内容，实际上不是写入当前寄存器 T2L 中，而是写入隐藏的寄存器 RL_T2L 中，对 T2H 写入内容，实际上也不是写入当前寄存器 T2H 中，而是写入隐藏的寄存器 RL_T2H，这样可以巧妙地实现 16 位重装载定时器。当读 T2H 和 T2L 的内容时，所读的内容就是 T2H 和 T2L 的内容，而不是 RL_T2H 和 RL_T2L 的内容。

[T2H,T2L]的溢出不仅置位中断请求标志位（T2IF），使 CPU 转去执行定时器 2 的中断程序，而且会自动将[RL_T2H,RL_T2L]的内容重新装入[T2H,T2L]。

17.3.6　定时器 2 计算公式

定时器速度	周期计算公式
1T	定时器周期 = $\frac{65536 - [T2H, T2L]}{SYSclk/(TM2PS+1)}$　（自动重载）
12T	定时器周期 = $\frac{65536 - [T2H, T2L]}{SYSclk/(TM2PS+1)} \times 12$　（自动重载）

17.4 定时器 3/4，24 位定时器，8 位预分频+16 位自动重装载定时器

17.4.1 定时器 3/4 功能脚切换

Ai805U 无 T3T4PIN/T3T4PS 寄存器。Ai8051U 的 T3 脚固定在 P0.4、T3CLKO 脚固定在 P0.5、T4 脚固定在 P0.6、T4CLKO 脚固定在 P0.7，均不能切换

17.4.2 定时器 4/3 控制寄存器（T4T3M）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	DDH	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO

T4R：定时器4的运行控制位

0：定时器 4 停止计数

1：定时器 4 开始计数

T4_C/T：控制定时器4用作定时器或计数器，清0则用作定时器（对内部系统时钟进行计数），置1用作计数器（对引脚T4/P0.6外部脉冲进行计数）。

T4x12：定时器4速度控制位

0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）

1：1T 模式，即 CPU 时钟不分频分频（FOSC/1）

T4CLKO：定时器4时钟输出控制

0：关闭时钟输出

1：使能 P0.7 口的是定时器 4 时钟输出功能

当定时器 4 计数发生溢出时，P0.7 口的电平自动发生翻转。

T3R：定时器3的运行控制位

0：定时器 3 停止计数

1：定时器 3 开始计数

T3_C/T：控制定时器3用作定时器或计数器，清0则用作定时器（对内部系统时钟进行计数），置1用作计数器（对引脚T3/P0.4外部脉冲进行计数）。

T3x12：定时器3速度控制位

0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）

1：1T 模式，即 CPU 时钟不分频分频（FOSC/1）

T3CLKO：定时器3时钟输出控制

0：关闭时钟输出

1：使能 P0.5 口的是定时器 3 时钟输出功能

当定时器 3 计数发生溢出时，P0.5 口的电平自动发生翻转。

17.4.3 定时器 3 计数寄存器（T3L，T3H）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T3L	D5H								
T3H	D4H								

定时器/计数器3的工作模式固定为16位重载模式，T3L和T3H组合成为一个16位寄存器，T3L为低字节，T3H为高字节。当[T3H,T3L]中的16位计数值溢出时，系统会自动将内部16位重载寄存器中的重载值装入[T3H,T3L]中。

17.4.4 定时器 4 计数寄存器（T4L，T4H）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T4L	D3H								
T4H	D2H								

定时器/计数器4的工作模式固定为16位重载模式，T4L和T4H组合成为一个16位寄存器，T4L为低字节，T4H为高字节。当[T4H,T4L]中的16位计数值溢出时，系统会自动将内部16位重载寄存器中的重载值装入[T4H,T4L]中。

17.4.5 定时器 3 的 8 位预分频寄存器（TM3PS）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TM3PS	7EFEA3H								

定时器3的时钟 ＝ 系统时钟SYSclk ÷ (TM3PS + 1)

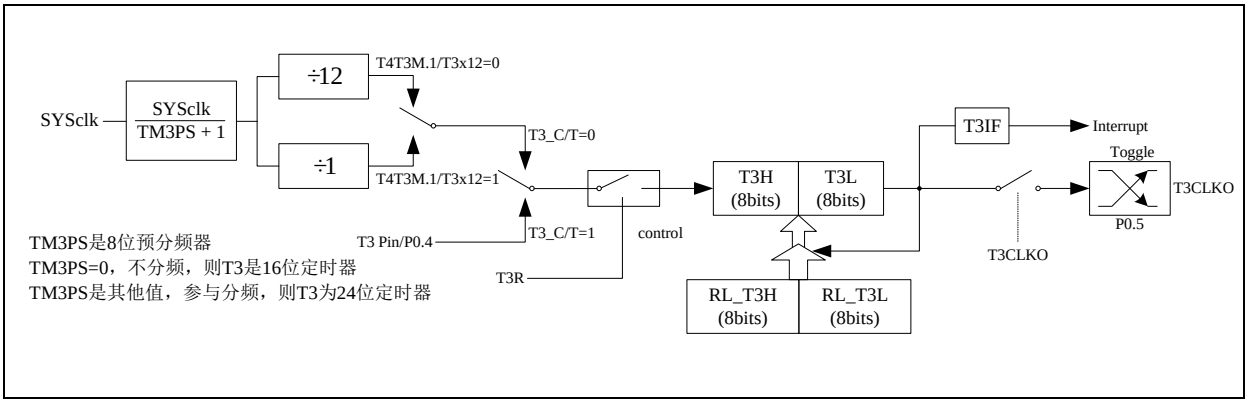
17.4.6 定时器 4 的 8 位预分频寄存器（TM4PS）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TM4PS	7EFEA4H								

定时器4的时钟 ＝ 系统时钟SYSclk ÷ (TM4PS + 1)

17.4.7　定时器 3 工作模式

定时器/计数器3的原理框图如下：



定时器/计数器 3 的工作模式：16 位自动重装载模式

T3R/T4T3M.3 为 T4T3M 寄存器内的控制位，T4T3M 寄存器各位的具体功能描述见上节 T4T3M 寄存器的介绍。

当 T3_C/T=0 时，多路开关连接到系统时钟输出，T3 对内部系统时钟计数，T3 工作在定时方式。当 T3_C/T=1 时，多路开关连接到外部脉冲输 T3，即 T3 工作在计数方式。

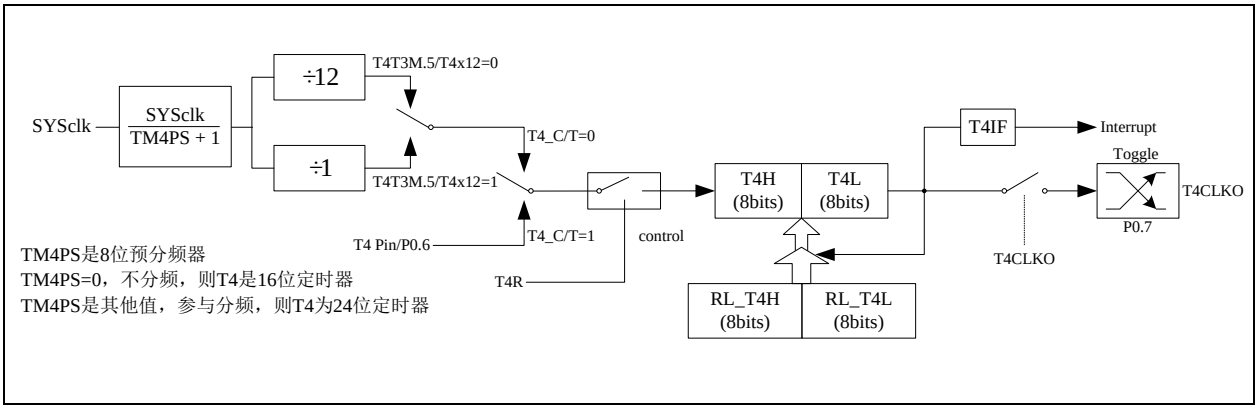
单片机的定时器 3 有两种计数速率：一种是 12T 模式，每 12 个时钟加 1，与传统 8051 单片机相同；另外一种是 1T 模式，每个时钟加 1，速度是传统 8051 单片机的 12 倍。T3 的速率由特殊功能寄存器 T4T3M 中的 T3x12 决定，如果 T3x12=0，T3 则工作在 12T 模式；如果 T3x12=1，T3 则工作在 1T 模式

定时器 3 有两个隐藏的寄存器 RL_T3H 和 RL_T3L。RL_T3H 与 T3H 共有同一个地址，RL_T3L 与 T3L 共有同一个地址。当 T3R=0 即定时器/计数器 3 被禁止工作时，对 T3L 写入的内容会同时写入 RL_T3L，对 T3H 写入的内容也会同时写入 RL_T3H。当 T3R=1 即定时器/计数器 3 被允许工作时，对 T3L 写入内容，实际上不是写入当前寄存器 T3L 中，而是写入隐藏的寄存器 RL_T3L 中，对 T3H 写入内容，实际上也不是写入当前寄存器 T3H 中，而是写入隐藏的寄存器 RL_T3H，这样可以巧妙地实现 16 位重装载定时器。当读 T3H 和 T3L 的内容时，所读的内容就是 T3H 和 T3L 的内容，而不是 RL_T3H 和 RL_T3L 的内容。

[T3H,T3L]的溢出不仅置位中断请求标志位（T3IF），使 CPU 转去执行定时器 3 的中断程序，而且会自动将[RL_T3H,RL_T3L]的内容重新装入[T3H,T3L]。

17.4.8　定时器 4 工作模式

定时器/计数器 4 的原理框图如下：



定时器/计数器 4 的工作模式：16 位自动重装载模式

T4R/T4T3M.7 为 T4T3M 寄存器内的控制位，T4T3M 寄存器各位的具体功能描述见上节 T4T3M 寄存器的介绍。

当 T4_C/T=0 时，多路开关连接到系统时钟输出，T4 对内部系统时钟计数，T4 工作在定时方式。当 T4_C/T=1 时，多路开关连接到外部脉冲输 T4，即 T4 工作在计数方式。

单片机的定时器 4 有两种计数速率：一种是 12T 模式，每 12 个时钟加 1，与传统 8051 单片机相同；另外一种是 1T 模式，每个时钟加 1，速度是传统 8051 单片机的 12 倍。T4 的速率由特殊功能寄存器 T4T3M 中的 T4x12 决定，如果 T4x12=0，T4 则工作在 12T 模式；如果 T4x12=1，T4 则工作在 1T 模式

定时器 4 有两个隐藏的寄存器 RL_T4H 和 RL_T4L。RL_T4H 与 T4H 共有同一个地址，RL_T4L 与 T4L 共有同一个地址。当 T4R=0 即定时器/计数器 4 被禁止工作时，对 T4L 写入的内容会同时写入 RL_T4L，对 T4H 写入的内容也会同时写入 RL_T4H。当 T4R=1 即定时器/计数器 4 被允许工作时，对 T4L 写入内容，实际上不是写入当前寄存器 T4L 中，而是写入隐藏的寄存器 RL_T4L 中，对 T4H 写入内容，实际上也不是写入当前寄存器 T4H 中，而是写入隐藏的寄存器 RL_T4H，这样可以巧妙地实现 16 位重装载定时器。当读 T4H 和 T4L 的内容时，所读的内容就是 T4H 和 T4L 的内容，而不是 RL_T4H 和 RL_T4L 的内容。

[T4H,T4L]的溢出不仅置位中断请求标志位（T4IF），使 CPU 转去执行定时器 4 的中断程序，而且会自动将[RL_T4H,RL_T4L]的内容重新装入[T4H,T4L]。

17.4.9 定时器 3 计算公式

定时器速度	周期计算公式
1T	定时器周期 = $\frac{65536 - [T3H, T3L]}{SYSclk/(TM3PS+1)}$ （自动重载）
12T	定时器周期 = $\frac{65536 - [T3H, T3L]}{SYSclk/(TM3PS+1)} \times 12$ （自动重载）

17.4.10 定时器 4 计算公式

定时器速度	周期计算公式
1T	定时器周期 = $\frac{65536 - [T4H, T4L]}{SYSclk/(TM4PS+1)}$ （自动重载）
12T	定时器周期 = $\frac{65536 - [T4H, T4L]}{SYSclk/(TM4PS+1)} \times 12$ （自动重载）

17.5 定时器 T11（24 位定时器，8 位预分频+16 位自动重装载定时器）

17.5.1 定时器 T11 控制寄存器（T11CR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T11CR	7EFE78H	T11R	T11_C/T	T11CLKO	T11x12	T11CS[1:0]		ET11I	T11IF

T11R：定时器T11的运行控制位

0：定时器 T11 停止计数

1：定时器 T11 开始计数

T11_C/T：控制定时器T11用作定时器或计数器，清0则用作定时器（对内部系统时钟进行计数），置1用作计数器（对引脚T11/P2.0外部脉冲进行计数）。

T11CLKO：定时器T11时钟输出控制

0：关闭时钟输出

1：使能 P2.1 口的是定时器 T11 时钟输出功能

当定时器 T11 计数发生溢出时，P2.1 口的电平自动发生翻转。

T11x12：定时器T11速度控制位

0：12T 模式，即 T11 时钟源 12 分频（T11CLK/12）

1：1T 模式，即 T11 时钟源不分频分频（T11CLK /1）

T11CS[1:0]：定时器T11时钟源（T11CLK）选择

T11CS[1:0]	定时器T11时钟源（T11CLK）
00	系统时钟SYSclk
01	内部高速IRC
10	外部32K晶振
11	内部低速IRC

特别说明：若定时器 T11 选择外部 32K 或者内部低速 IRC 作为时钟源，当 MCU 进入主时钟停振/省电模式时，定时器 T11 会继续工作，发送 T11 定时中断时可唤醒主时钟停振/省电模式

ET11I：定时/计数器 T11 的溢出中断允许位。

0：禁止 T11 中断

1：允许 T11 中断

T11IF：T11溢出中断标志。T11被允许计数以后，从初值开始加1计数，当产生溢出时，由硬件置“1”T11IF，向CPU请求中断，一直保持CPU响应该中断时，才由硬件清0（也可由查询软件清0）。

17.5.2 定时器 T11 的 8 位预分频寄存器（T11PS）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T11PS	7EFE79H								

定时器T11的时钟 ＝ 系统时钟SYSclk ÷ (T11PS + 1)

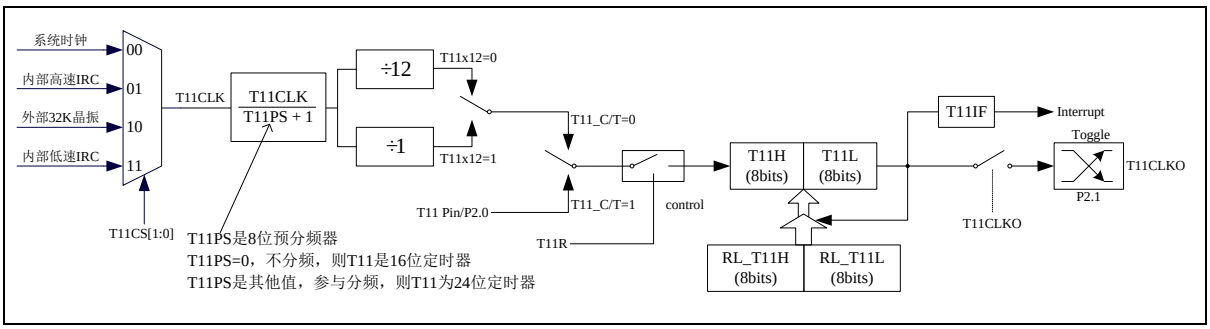
17.5.3 定时器 T11 计数寄存器（T11L，T11H）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T11L	7EFE7BH								
T11H	7EFE7AH								

定时器/计数器T11的工作模式固定为16位重载模式，T11L和T11H组合成为一个16位寄存器，T11L为低字节，T11H为高字节。当[T11H,T11L]中的16位计数值溢出时，系统会自动将内部16位重载寄存器中的重载值装入[T11H,T11L]中。

17.5.4　定时器 T11 工作模式

定时器/计数器 T11 的原理框图如下：



定时器/计数器 T11 的工作模式：16 位自动重装载模式

当 T11_C/T=0 时，多路开关连接到内部时钟 T11CLK，T11 对内部 T11CLK 时钟计数，T11 工作在定时方式。当 T11_C/T=1 时，多路开关连接到外部脉冲输 T11（P1.4.），即 T11 工作在计数方式。

单片机的定时器 T11 有两种计数速率：一种是 12T 模式，每 12 个时钟加 1，与传统 8051 单片机相同；另外一种是 1T 模式，每个时钟加 1，速度是传统 8051 单片机的 12 倍。T11 的速率由特殊功能寄存器 T11x12 决定，如果 T11x12=0，T11 则工作在 12T 模式；如果 T11x12=1，T11 则工作在 1T 模式

定时器 T11 有两个隐藏的寄存器 RL_T11H 和 RL_T11L。RL_T11H 与 T11H 共有同一个地址，RL_T11L 与 T11L 共有同一个地址。当 T11R=0 即定时器/计数器 T11 被禁止工作时，对 T11L 写入的内容会同时写入 RL_T11L，对 T11H 写入的内容也会同时写入 RL_T11H。当 T11R=1 即定时器/计数器 T11 被允许工作时，对 T11L 写入内容，实际上不是写入当前寄存器 T11L 中，而是写入隐藏的寄存器 RL_T11L 中，对 T11H 写入内容，实际上也不是写入当前寄存器 T11H 中，而是写入隐藏的寄存器 RL_T11H，这样可以巧妙地实现 16 位重装载定时器。当读 T11H 和 T11L 的内容时，所读的内容就是 T11H 和 T11L 的内容，而不是 RL_T11H 和 RL_T11L 的内容。

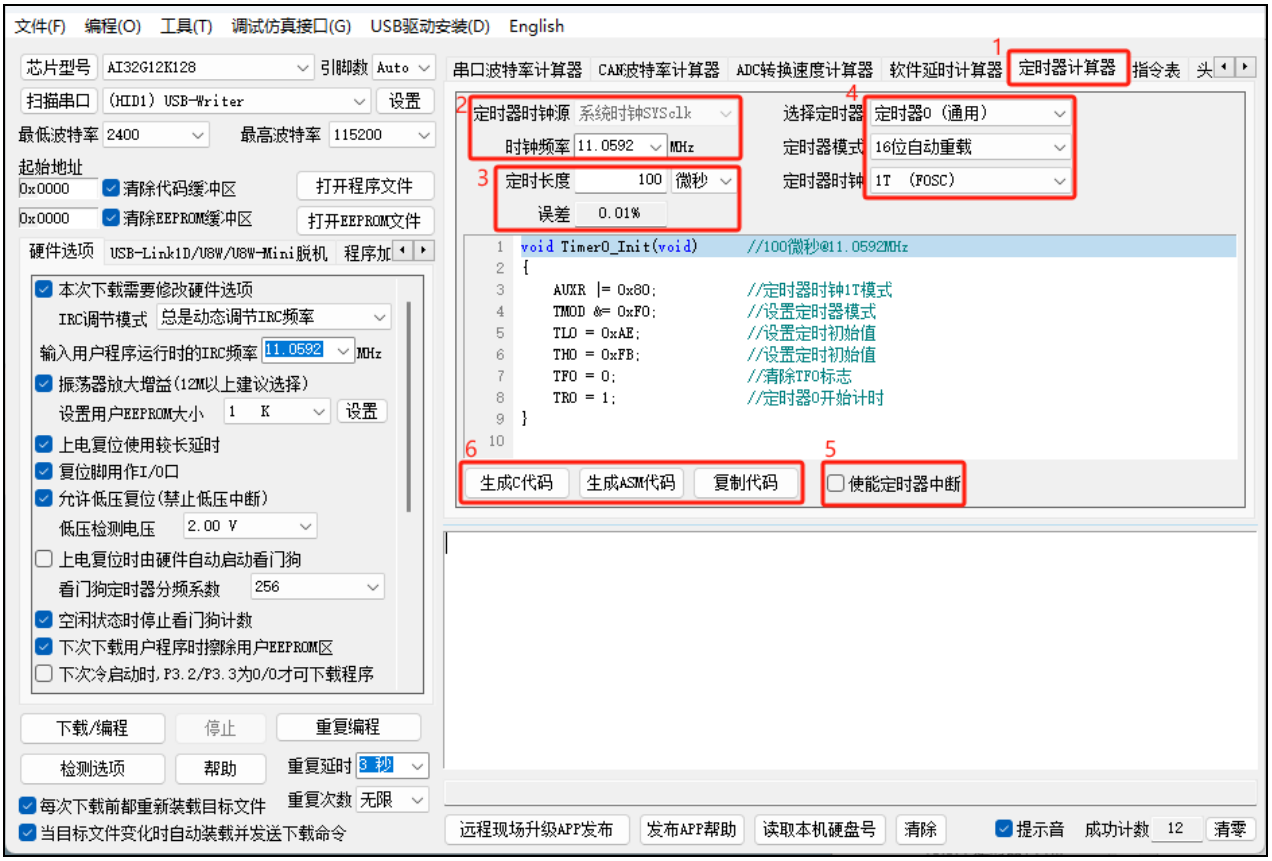
[T11H,T11L]的溢出不仅置位中断请求标志位（T11IF），使 CPU 转去执行定时器 T11 的中断程序，而且会自动将[RL_T11H,RL_T11L]的内容重新装入[T11H,T11L]。

特别的，定时器 T11 的时钟源可通过 T11CS 寄存器进行选择，可选择：系统时钟、内部高速 IRC 时钟、外部 32K 晶振以及内部低速 IRC 时钟

17.5.5 定时器 T11 计算公式

定时器速度	周期计算公式
1T	定时器周期 = $\frac{65536 - [T11H, T11L]}{T11CLK/(T11PS+1)}$ （自动重载）
12T	定时器周期 = $\frac{65536 - [T11H, T11L]}{T11CLK/(T11PS+1)} \times 12$ （自动重载）

17.6 AIapp-ISP | 定时器计算器工具



①：在下载软件中选择“定时器计算器”功能页，进定时器代码生成界面

②：设置系统工作频率（单位：MHz）

③：设置定时时间长度（单位：毫秒/微秒）

④：选择目标定时器，并设置定时器工作模式

⑤：选择是否需要使能定时器中断

⑥：手动生成 C 代码或者 ASM 代码，复制范例

17.7　范例程序

17.7.1　定时器 0（模式 0－16 位自动重载），用作定时

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件
#include "intrins.h"                          //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void TM0_Isr() interrupt 1
{
    P10 = !P10;                               //测试端口
}

void main()
{
    EAXFR = 1;                                //允许访问扩展的特殊寄存器，XFR
                                              //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                            //允许访问扩展的特殊寄存器，XFR
                                              //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                 //设置取程序代码等待时间，
                                              //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                //设置访问片内的 xdata 速度，
                                              //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x00;                              //模式 0
    TL0 = 0x66;                               //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                                  //启动定时器
    ET0 = 1;                                  //使能定时器中断
    EA = 1;

    while (1);
}
```

17.7.2　定时器 0（模式 1－16 位不自动重载），用作定时

```
//测试工作频率为 11.0592MHz
```

```
#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void TM0_Isr() interrupt 1
{
    TL0 = 0x66;                                         //重设定时参数
    TH0 = 0xfc;
    P10 = !P10;                                         //测试端口
}

void main()
{
    EAXFR = 1;                                          //允许访问扩展的特殊寄存器，XFR
                                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                      //允许访问扩展的特殊寄存器，XFR
                                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                           //设置取程序代码等待时间，
                                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                          //设置访问片内的 xdata 速度，
                                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x01;                                        //模式 1
    TL0 = 0x66;                                         //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                                            //启动定时器
    ET0 = 1;                                            //使能定时器中断
    EA = 1;

    while (1);
}
```

17.7.3　定时器 0（模式 2—8 位自动重载），用作定时

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void TM0_Isr() interrupt 1
```

```
{
    P10 = !P10;                                 //测试端口
}

void main()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x02;                                //模式 2
    TL0 = 0xf4;                                 //256-11.0592M/12/76K
    TH0 = 0xf4;
    TR0 = 1;                                    //启动定时器
    ET0 = 1;                                    //使能定时器中断
    EA = 1;

    while (1);
}
```

17.7.4　定时器 0（模式 3－16 位自动重载不可屏蔽中断），用作定时

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                            //头文件见下载软件
#include "intrins.h"                            //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void TM0_Isr() interrupt 1
{
    P10 = !P10;                                 //测试端口
}

void main()
{
```

```
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x03;                                //模式 3
    TL0 = 0x66;                                 //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                                    //启动定时器
    ET0 = 1;                                    //使能定时器中断
//  EA = 1;                                     //不受 EA 控制

    while (1);
}
```

17.7.5　定时器 0（外部计数—扩展 T0 为外部下降沿中断）

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                            //头文件见下载软件
#include "intrins.h"                            //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void TM0_Isr() interrupt 1
{
    P10 = !P10;                                 //测试端口
}

void main()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间
```

```
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x04;                                    //外部计数模式
    TL0 = 0xff;
    TH0 = 0xff;
    TR0 = 1;                                        //启动定时器
    ET0 = 1;                                        //使能定时器中断
    EA = 1;

    while (1);
}
```

17.7.6　定时器 0（测量脉宽－INT0 高电平宽度）

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                //头文件见下载软件
#include "intrins.h"                                //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void INT0_Isr() interrupt 0
{
    P0 = TL0;                                       //TL0 为测量值低字节
    P1 = TH0;                                       //TH0 为测量值高字节
    TL0 = 0x00;
    TH0 = 0x00;
}

void main()
{
    EAXFR = 1;                                      //允许访问扩展的特殊寄存器，XFR
                                                    //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                  //允许访问扩展的特殊寄存器，XFR
                                                    //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                       //设置取程序代码等待时间，
                                                    //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                      //设置访问片内的 xdata 速度，
                                                    //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
```

```
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T0x12 = 1;                          //1T 模式
    TMOD = 0x08;                        //使能 GATE,INT0 为 1 时使能计时
    TL0 = 0x00;
    TH0 = 0x00;
    while (P32);                        //等待 INT0 为低
    TR0 = 1;                            //启动定时器
    IT0 = 1;                            //使能 INT0 下降沿中断
    EX0 = 1;
    EA = 1;

    while (1);
}
```

17.7.7　定时器 0（模式 0），时钟分频输出

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                   //头文件见下载软件
#include "intrins.h"                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                           //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间


    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x00;                        //模式 0
```

```
    TL0 = 0x66;                                 //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                                    //启动定时器
    T0CLKO = 1;                                 //使能时钟输出

    while (1);
}
```

17.7.8　定时器 1（模式 0－16 位自动重载），用作定时

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                            //头文件见下载软件
#include "intrins.h"                            //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void TM1_Isr() interrupt 3
{
    P10 = !P10;                                 //测试端口
}

void main()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x00;                                //模式 0
    TL1 = 0x66;                                 //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                                    //启动定时器
    ET1 = 1;                                    //使能定时器中断
    EA = 1;

    while (1);
}
```

17.7.9　定时器 1（模式 1－16 位不自动重载），用作定时

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件
#include "intrins.h"                          //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void TM1_Isr() interrupt 3
{
    TL1 = 0x66;                               //重设定时参数
    TH1 = 0xfc;
    P10 = !P10;                               //测试端口
}

void main()
{
    EAXFR = 1;                                //允许访问扩展的特殊寄存器，XFR
                                              //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                            //允许访问扩展的特殊寄存器，XFR
                                              //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                 //设置取程序代码等待时间，
                                              //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                //设置访问片内的 xdata 速度，
                                              //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x10;                              //模式 1
    TL1 = 0x66;                               //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                                  //启动定时器
    ET1 = 1;                                  //使能定时器中断
    EA = 1;

    while (1);
}
```

17.7.10　定时器 1（模式 2－8 位自动重载），用作定时

```
//测试工作频率为 11.0592MHz
```

```
#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void TM1_Isr() interrupt 3
{
    P10 = !P10;                                         //测试端口
}

void main()
{
    EAXFR = 1;                                          //允许访问扩展的特殊寄存器，XFR
                                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                      //允许访问扩展的特殊寄存器，XFR
                                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                           //设置取程序代码等待时间，
                                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                          //设置访问片内的 xdata 速度，
                                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x20;                                        //模式 2
    TL1 = 0xf4;                                         //256-11.0592M/12/76K
    TH1 = 0xf4;
    TR1 = 1;                                            //启动定时器
    ET1 = 1;                                            //使能定时器中断
    EA = 1;

    while (1);
}
```

17.7.11　定时器 1（外部计数—扩展 T1 为外部下降沿中断）

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void TM1_Isr() interrupt 3
{
    P10 = !P10;                                         //测试端口
```

```
}

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                           //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x40;                        //外部计数模式
    TL1 = 0xff;
    TH1 = 0xff;
    TR1 = 1;                            //启动定时器
    ET1 = 1;                            //使能定时器中断
    EA = 1;

    while (1);
}
```

17.7.12　定时器 1（测量脉宽－INT1 高电平宽度）

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                   //头文件见下载软件
#include "intrins.h"                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void INT1_Isr() interrupt 2
{
    P0 = TL1;                           //TL1 为测量值低字节
    P1 = TH1;                           //TH1 为测量值高字节
    TL1 = 0x00;
    TH1= 0x00;
}

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
```

```
                                        //(32 位模式请使用这句，注释下一句)
//      P_SW2 |= 0x80;                  //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
        WTST = 0;                       //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
        CKCON = 0;                      //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

        T1x12 = 1;                      //1T 模式
        TMOD = 0x80;                    //使能 GATE,INT1 为 1 时使能计时
        TL1 = 0x00;
        TH1 = 0x00;
        while (P33);                    //等待 INT1 为低
        TR1 = 1;                        //启动定时器
        IT1 = 1;                        //使能 INT1 下降沿中断
        EX1 = 1;
        EA = 1;

        while (1);
}
```

17.7.13　定时器 1（模式 0），时钟分频输出

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                   //头文件见下载软件
#include "intrins.h"                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void main()
{
        EAXFR = 1;                      //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//      P_SW2 |= 0x80;                  //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
        WTST = 0;                       //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
        CKCON = 0;                      //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

        P0M0 = 0x00;
        P0M1 = 0x00;
```

```
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x00;                                    //模式 0
    TL1 = 0x66;                                     //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                                        //启动定时器
    T1CLKO = 1;                                     //使能时钟输出

    while (1);
}
```

17.7.14　定时器 1（模式 0）做串口 1 波特率发生器

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                //头文件见下载软件
#include "intrins.h"                                //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   FOSC          11059200UL                  //定义为无符号长整型,避免计算溢出
#define   BRT           (65536 – (FOSC / 115200 + 2) / 4)

                                                    //加 2 操作是为了让 Keil 编译器
                                                    //自动实现四舍五入运算

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

void UartInit()
{
```

```
    SCON = 0x50;
    S1BRT = 0;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    T1x12 = 1;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void UartSendStr(char *p)
{
    while (*p)
    {
        UartSend(*p++);
    }
}

void main()
{
    EAXFR = 1;                  //允许访问扩展的特殊寄存器，XFR
                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;              //允许访问扩展的特殊寄存器，XFR
                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                   //设置取程序代码等待时间，
                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                  //设置访问片内的 xdata 速度，
                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
```

```
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

17.7.15　定时器 1（模式 2）做串口 1 波特率发生器

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   FOSC        11059200UL                        //定义为无符号长整型,避免计算溢出
#define   BRT         (256 - (FOSC / 115200+16) / 32)
                                                        //加 16 操作是为了让 Keil 编译器
                                                        //自动实现四舍五入运算

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

void UartInit()
{
    SCON = 0x50;
    S1BRT = 0;
    TMOD = 0x20;
    TL1 = BRT;
    TH1 = BRT;
    TR1 = 1;
    T1x12 = 1;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void UartSend(char dat)
{
```

```
        while (busy);
        busy = 1;
        SBUF = dat;
}

void UartSendStr(char *p)
{
        while (*p)
        {
                UartSend(*p++);
        }
}

void main()
{
        EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                    //(32 位模式请使用这句，注释下一句)
//      P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                    //(8 位模式请使用这句，注释上一句)
        WTST = 0;                                   //设置取程序代码等待时间，
                                                    //赋值为 0 表示不等待，程序以最快速度运行
        CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                    //赋值为 0 表示用最快速度访问，不增加额外的等待时间

        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

        UartInit();
        ES = 1;
        EA = 1;
        UartSendStr("Uart Test !\r\n");

        while (1)
        {
                if (rptr != wptr)
                {
                        UartSend(buffer[rptr++]);
                        rptr &= 0x0f;
                }
        }
}
```

17.7.16　定时器 2（16 位自动重载），用作定时

```
//测试工作频率为 11.0592MHz
```

```
#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void TM2_Isr() interrupt 12
{
    P10 = !P10;                                         //测试端口
}

void main()
{
    EAXFR = 1;                                          //允许访问扩展的特殊寄存器，XFR
                                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                      //允许访问扩展的特殊寄存器，XFR
                                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                           //设置取程序代码等待时间，
                                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                          //设置访问片内的 xdata 速度，
                                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T2L = 0x66;                                         //65536-11.0592M/12/1000
    T2H = 0xfc;
    T2R = 1;                                            //启动定时器
    ET2 = 1;                                            //使能定时器中断
    EA = 1;

    while (1);
}
```

17.7.17　定时器 2（外部计数－扩展 T2 为外部下降沿中断）

```
//测试工作频率为11.0592MHz

#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void TM2_Isr() interrupt 12
{
    P10 = !P10;                                         //测试端口
}
```

```
void main()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T2L = 0xff;
    T2H = 0xff;
    T2_CT = 1; T2R = 1;                         //设置外部计数模式并启动定时器
    ET2 = 1;                                    //使能定时器中断
    EA = 1;

    while (1);
}
```

17.7.18　定时器 2，时钟分频输出

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                            //头文件见下载软件
#include "intrins.h"                            //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void main()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
```

```
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T2L = 0x66;                                 //65536-11.0592M/12/1000
    T2H = 0xfc;
    T2R = 1;                                    //启动定时器
    T2CLKO = 1;                                 //使能时钟输出

    while (1);
}
```

17.7.19　定时器 2 做串口 1 波特率发生器

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                            //头文件见下载软件
#include "intrins.h"                            //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   FOSC        11059200UL                //定义为无符号长整型,避免计算溢出
#define   BRT         (65536 - (FOSC / 115200+2) / 4)
                                                //加 2 操作是为了让 Keil 编译器
                                                //自动实现四舍五入运算

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

void UartInit()
{
    SCON = 0x50;
```

```
    T2L = BRT;
    T2H = BRT >> 8;
    S1BRT = 1;
    T2x12 = 1;
    T2R = 1;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void UartSendStr(char *p)
{
    while (*p)
    {
        UartSend(*p++);
    }
}

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                           //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
```

```
            rptr &= 0x0f;
        }
    }
}
```

17.7.20　定时器 2 做串口 2 波特率发生器

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   FOSC        11059200UL                        //定义为无符号长整型,避免计算溢出
#define   BRT         (65536 - (FOSC / 115200+2) / 4)
                                                        //加 2 操作是为了让 Keil 编译器
                                                        //自动实现四舍五入运算

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void Uart2Isr() interrupt 8
{
    if (S2TI)
    {
        S2TI = 0;
        busy = 0;
    }
    if (S2RI)
    {
        S2RI = 0;
        buffer[wptr++] = S2BUF;
        wptr &= 0x0f;
    }
}

void Uart2Init()
{
    S2CON = 0x50;
    T2L = BRT;
    T2H = BRT >> 8;
    T2x12 = 1;
    T2R = 1;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart2Send(char dat)
{
    while (busy);
    busy = 1;
    S2BUF = dat;
}
```

```
void Uart2SendStr(char *p)
{
    while (*p)
    {
        Uart2Send(*p++);
    }
}

void main()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart2Init();
    IE2 = 0x01;
    EA = 1;
    Uart2SendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart2Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

17.7.21　定时器 2 做串口 3 波特率发生器

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                           //头文件见下载软件
#include "intrins.h"                            //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。
```

```
#define    FOSC        11059200UL                  //定义为无符号长整型,避免计算溢出
#define    BRT         (65536 - (FOSC / 115200+2) / 4)
                                                   //加 2 操作是为了让 Keil 编译器
                                                   //自动实现四舍五入运算

bit        busy;
char       wptr;
char       rptr;
char       buffer[16];

void Uart3Isr() interrupt 17
{
    if (S3TI)
    {
        S3TI = 0;
        busy = 0;
    }
    if (S3RI)
    {
        S3RI = 0;
        buffer[wptr++] = S3BUF;
        wptr &= 0x0f;
    }
}

void Uart3Init()
{
    S3CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    T2x12 = 1;
    T2R = 1;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart3Send(char dat)
{
    while (busy);
    busy = 1;
    S3BUF = dat;
}

void Uart3SendStr(char *p)
{
    while (*p)
    {
        Uart3Send(*p++);
    }
}

void main()
{
    EAXFR = 1;                                     //允许访问扩展的特殊寄存器，XFR
                                                   //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                 //允许访问扩展的特殊寄存器，XFR
                                                   //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                      //设置取程序代码等待时间，
```

```
                                                    //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                      //设置访问片内的 xdata 速度，
                                                    //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart3Init();
    IE2 = 0x08;
    EA = 1;
    Uart3SendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart3Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

17.7.22　定时器 2 做串口 4 波特率发生器

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                //头文件见下载软件
#include "intrins.h"                                //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   FOSC        11059200UL                    //定义为无符号长整型,避免计算溢出
#define   BRT         (65536 - (FOSC / 115200+2) / 4)
                                                    //加 2 操作是为了让 Keil 编译器
                                                    //自动实现四舍五入运算

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void Uart4Isr() interrupt 18
{
    if (S4TI)
    {
        S4TI = 0;
```

```
            busy = 0;
        }
        if (S4RI)
        {
            S4RI = 0;
            buffer[wptr++] = S4BUF;
            wptr &= 0x0f;
        }
}

void Uart4Init()
{
        S4CON = 0x10;
        T2L = BRT;
        T2H = BRT >> 8;
        T2x12 = 1;
        T2R = 1;
        wptr = 0x00;
        rptr = 0x00;
        busy = 0;
}

void Uart4Send(char dat)
{
        while (busy);
        busy = 1;
        S4BUF = dat;
}

void Uart4SendStr(char *p)
{
        while (*p)
        {
            Uart4Send(*p++);
        }
}

void main()
{
        EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                            //(32 位模式请使用这句，注释下一句)
//      P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                            //(8 位模式请使用这句，注释上一句)
        WTST = 0;                           //设置取程序代码等待时间，
                                            //赋值为 0 表示不等待，程序以最快速度运行
        CKCON = 0;                          //设置访问片内的 xdata 速度，
                                            //赋值为 0 表示用最快速度访问，不增加额外的等待时间

        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
```

```
    P5M1 = 0x00;

    Uart4Init();
    IE2 = 0x10;
    EA = 1;
    Uart4SendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart4Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

17.7.23　定时器 3（16 位自动重载），用作定时

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   ET2         0x04
#define   ET3         0x20
#define   ET4         0x40
#define   T2IF        0x01
#define   T3IF        0x02
#define   T4IF        0x04

void TM3_Isr() interrupt 19
{
    P10 = !P10;                         //测试端口
}

void main()
{
    EAXFR = 1;                                          //允许访问扩展的特殊寄存器，XFR
                                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                      //允许访问扩展的特殊寄存器，XFR
                                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                           //设置取程序代码等待时间，
                                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                          //设置访问片内的 xdata 速度，
                                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
```

```
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T3L = 0x66;                 //65536-11.0592M/12/1000
    T3H = 0xfc;
    T4T3M = 0x08;               //启动定时器
    IE2 = ET3;                  //使能定时器中断
    EA = 1;

    while (1);
}
```

17.7.24　定时器 3（外部计数－扩展 T3 为外部下降沿中断）

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                  //头文件见下载软件
#include "intrins.h"                                  //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   ET2           0x04
#define   ET3           0x20
#define   ET4           0x40
#define   T2IF          0x01
#define   T3IF          0x02
#define   T4IF          0x04

void TM3_Isr() interrupt 19
{
    P10 = !P10;                 //测试端口
}

void main()
{
    EAXFR = 1;                                        //允许访问扩展的特殊寄存器，XFR
                                                      //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                    //允许访问扩展的特殊寄存器，XFR
                                                      //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                         //设置取程序代码等待时间，
                                                      //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                        //设置访问片内的 xdata 速度，
                                                      //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
```

```
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T3L = 0xff;
    T3H = 0xff;
    T4T3M = 0x0c;                   //设置外部计数模式并启动定时器
    IE2 = ET3;                      //使能定时器中断
    EA = 1;

    while (1);
}
```

17.7.25　定时器 3，时钟分频输出

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                  //头文件见下载软件
#include "intrins.h"                                  //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void main()
{
    EAXFR = 1;                                        //允许访问扩展的特殊寄存器，XFR
                                                      //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                    //允许访问扩展的特殊寄存器，XFR
                                                      //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                         //设置取程序代码等待时间，
                                                      //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                        //设置访问片内的 xdata 速度，
                                                      //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T3L = 0x66;                                       //65536-11.0592M/12/1000
    T3H = 0xfc;
    T4T3M = 0x09;                                     //使能时钟输出并启动定时器

    while (1);
}
```

17.7.26　定时器 3 做串口 3 波特率发生器

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                  //头文件见下载软件
#include "intrins.h"                                  //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   FOSC          11059200UL                    //定义为无符号长整型,避免计算溢出
#define   BRT           (65536 - (FOSC / 115200+2) / 4)
                                                      //加 2 操作是为了让 Keil 编译器
                                                      //自动实现四舍五入运算

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void Uart3Isr() interrupt 17
{
    if (S3TI)
    {
        S3TI = 0;
        busy = 0;
    }
    if (S3RI)
    {
        S3RI = 0;
        buffer[wptr++] = S3BUF;
        wptr &= 0x0f;
    }
}

void Uart3Init()
{
    S3CON = 0x50;
    T3L = BRT;
    T3H = BRT >> 8;
    T4T3M = 0x0a;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart3Send(char dat)
{
    while (busy);
    busy = 1;
    S3BUF = dat;
}

void Uart3SendStr(char *p)
{
    while (*p)
    {
        Uart3Send(*p++);
    }
```

```
}

void main()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart3Init();
    IE2 = 0x08;
    EA = 1;
    Uart3SendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart3Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

17.7.27　定时器 4（16 位自动重载），用作定时

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                           //头文件见下载软件
#include "intrins.h"                            //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   ET2           0x04
#define   ET3           0x20
#define   ET4           0x40
#define   T2IF          0x01
#define   T3IF          0x02
#define   T4IF          0x04
```

```
void TM4_Isr() interrupt 20
{
    P10 = !P10;                         //测试端口
}

void main()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T4L = 0x66;                         //65536-11.0592M/12/1000
    T4H = 0xfc;
    T4T3M = 0x80;                       //启动定时器
    IE2 = ET4;                          //使能定时器中断
    EA = 1;

    while (1);
}
```

17.7.28　定时器 4（外部计数－扩展 T4 为外部下降沿中断）

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                 //头文件见下载软件
#include "intrins.h"                                //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   ET2       0x04
#define   ET3       0x20
#define   ET4       0x40
#define   T2IF      0x01
#define   T3IF      0x02
#define   T4IF      0x04

void TM4_Isr() interrupt 20
```

```
{
    P10 = !P10;                         //测试端口
}

void main()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T4L = 0xff;
    T4H = 0xff;
    T4T3M = 0xc0;                       //设置外部计数模式并启动定时器
    IE2 = ET4;                          //使能定时器中断
    EA = 1;

    while (1);
}
```

17.7.29　定时器 4，时钟分频输出

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                              //头文件见下载软件
#include "intrins.h"                               //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void main()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间
```

```
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P3M1 = 0x00;

    T4L = 0x66;                                 //65536-11.0592M/12/1000
    T4H = 0xfc;
    T4T3M = 0x90;                               //使能时钟输出并启动定时器

    while (1);
}
```

17.7.30　定时器 4 做串口 4 波特率发生器

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                            //头文件见下载软件
#include "intrins.h"                            //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   FOSC        11059200UL                //定义为无符号长整型,避免计算溢出
#define   BRT         (65536 - (FOSC / 115200+2) / 4)
                                                //加 2 操作是为了让 Keil 编译器
                                                //自动实现四舍五入运算

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void Uart4Isr() interrupt 18
{
    if (S4TI)
    {
        S4TI = 0;
        busy = 0;
    }
    if (S4RI)
    {
        S4RI = 0;
        buffer[wptr++] = S4BUF;
        wptr &= 0x0f;
    }
}

void Uart4Init()
```

```
{
    S4CON = 0x50;
    T4L = BRT;
    T4H = BRT >> 8;
    T4T3M = 0xa0;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart4Send(char dat)
{
    while (busy);
    busy = 1;
    S4BUF = dat;
}

void Uart4SendStr(char *p)
{
    while (*p)
    {
        Uart4Send(*p++);
    }
}

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                           //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart4Init();
    IE2 = 0x10;
    EA = 1;
    Uart4SendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart4Send(buffer[rptr++]);
```

```
            rptr &= 0x0f;
        }
    }
}
```

17.7.31 定时器 T11 应用范例

C 语言代码

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                  //头文件见下载软件
#include "intrins.h"                                  //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   MAIN_Fosc       11059200UL
#define   T11MS12T        (65536 - MAIN_Fosc / 12 / 1000)
#define   T11MS1T         (65536 - MAIN_Fosc / 1000)

void tm11isr() interrupt 13                           //借用 13 号中断向量地址，在 isr.asm 进行中断映射
{
//  T11CR &= ~0x01;                                   //清中断标志 T11IF，进中断时硬件自动清除
    P10 = ~P10;
}

void main()
{
    P_SW2 |= 0x80;                                    //扩展寄存器(XFR)访问使能

    P0M1 = 0x00;   P0M0 = 0x00;                       //设置为准双向口
    P1M1 = 0x00;   P1M0 = 0x00;                       //设置为准双向口
    P2M1 = 0x00;   P2M0 = 0x00;                       //设置为准双向口
    P3M1 = 0x00;   P3M0 = 0x00;                       //设置为准双向口
    P4M1 = 0x00;   P4M0 = 0x00;                       //设置为准双向口
    P5M1 = 0x00;   P5M0 = 0x00;                       //设置为准双向口
    P6M1 = 0x00;   P6M0 = 0x00;                       //设置为准双向口
    P7M1 = 0x00;   P7M0 = 0x00;                       //设置为准双向口

    IRCDB = 0x10;       //内部高速 IRC 时钟，停振后被唤醒后起振，等待多少个时钟数后给 MCU 供应时钟
    IAP_TPS = 40;       //根据系统工作频率设置此寄存器
                        //（如果系统工作频率为 40MHz，则 IAP_TPS 设置为 40；
                        //  如果系统工作频率为 22.1184MHz，则 IAP_TPS 设置为 22）

    //定时器(12T)
//  T11CR = 0x00;                                     //做定时器关闭时钟输出,12T 模式,系统时钟做时钟源
//  T11PS = 0;                                        //分频系数：(T11PS+1)分频
//  T11L = T11MS12T;
//  T11H = T11MS12T >> 8;
//  T11CR |= 0x82;                                    //定时器 11 开始计数，允许中断
//  EA = 1;

    //定时器(1T)
//  T11CR = 0x10;                                     //做定时器，关闭时钟输出，1T 模式，系统时钟做时钟源
//  T11PS = 0;                                        //分频系数：(T11PS+1)分频
//  T11L = T11MS1T;
//  T11H = T11MS1T >> 8;
```

```
//      T11CR |= 0x82;                                  //定时器 11 开始计数，允许中断
//      EA = 1;

        //计数模式 - T11 脚(P14)输入脉冲计数
//      T11CR = 0x50;                                  //做计数器，关闭时钟输出，1T 模式，系统时钟做时钟源
//      T11PS = 0;                                     //分频系数：(T11PS+1)分频
//      T11L = 0xff;
//      T11H = 0xff;
//      T11CR |= 0x82;                                 //定时器 11 开始计数，允许中断
//      EA = 1;


//      X32KCR = 0x80 + 0x40;                          //启动外部 32K 晶振，低增益+0x00，高增益+0x40.
//      while (!(X32KCR & 1));                         //等待时钟稳定

        //选择时钟源
//      T11CR = 0x10;                                  //做定时器关闭时钟输出，1T 模式，系统时钟做时钟源
        T11CR = 0x1c;                                  //做定时器关闭时钟输出，1T 模式，
                                                       //内部低速 IRC 做时钟源(自动启动内部低速 IRC)
//      T11CR = 0x18;                                  //做定时器关闭时钟输出，1T 模式，
                                                       //外部 32K 晶振做时钟源
        T11PS = 0;                                     //分频系数：(T11PS+1)分频
        T11L = T11MS1T;
        T11H = T11MS1T >> 8;
        T11CR |= 0x82;                                 //定时器 11 开始计数，允许中断
        EA = 1;

        //时钟输出
        T11CR |= 0x20;                                 //使能 P1.5 口定时器 11 时钟输出

        while (1)
        {
            P11 = ~P11;

            //唤醒 PD 休眠模式
            _nop_();
            _nop_();
            _nop_();
            PCON = 0x02;
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
        }
}
```

17.8 深圳大学--定时器实验：定时器 0/1，24 位定时器

17.8.1 深大--定时器 0/1，24 位定时器实验（定时器中断点灯并通过 USB-CDC 端口输出信息）

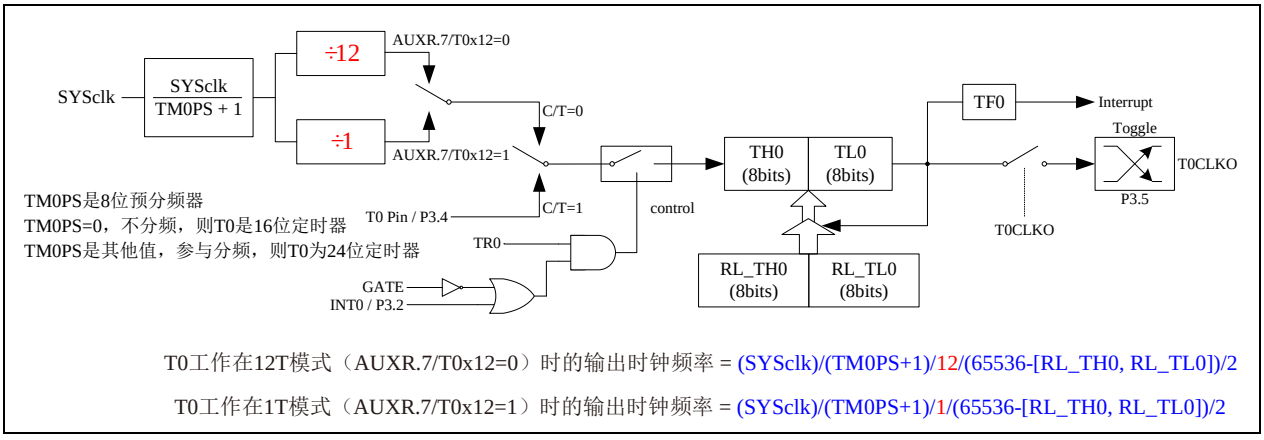
a) 实验介绍

定时器 0（Timer0）定时 2s 产生中断，P00 灯闪烁，CDC 窗口显示“Timer0！”
定时器 1（Timer1）定时 500ms 产生中断，P07 灯闪烁，CDC 窗口显示“Timer1！”
观察到：

- AIapp-ISP 软件的 CDC 窗口每显示 4 次“Timer1”，显示 1 次“Timer0”
- 实验箱，P07 亮灭各 2 次，P00 亮 1 次

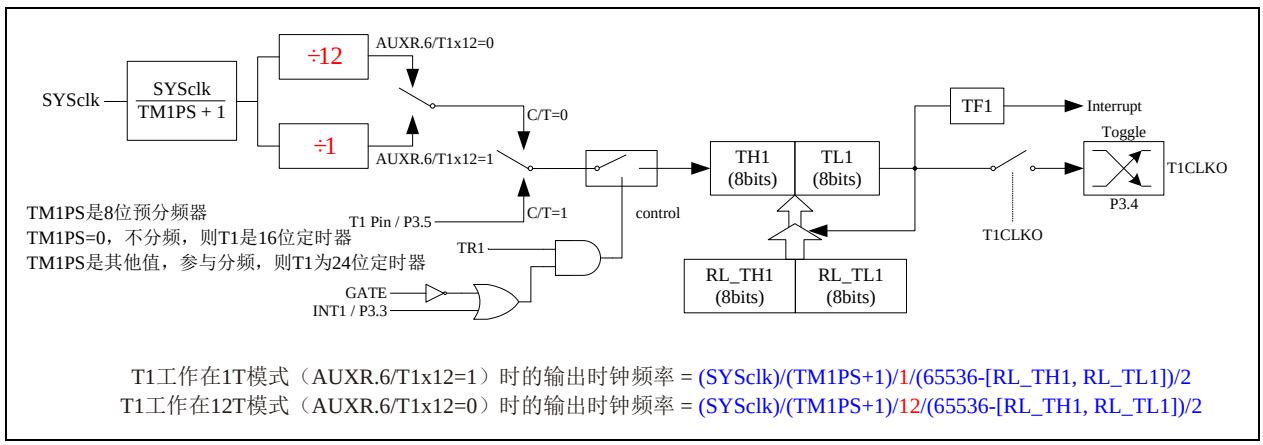
b) 定时器 0 模式 0 结构图，定时器 1 模式 0 结构图

定时器 0 模式 0 结构图：此模式下定时器/计数器 0 作为可自动重装载的 16 位计数器，如果启动 8 位预分频 TM0PS 就是 24 位定时器。如下图所示：



定时器/计数器 0 的模式 0：16 位自动重装载模式，如果启动 8 位预分频 TM0PS 就是 24 位定时器

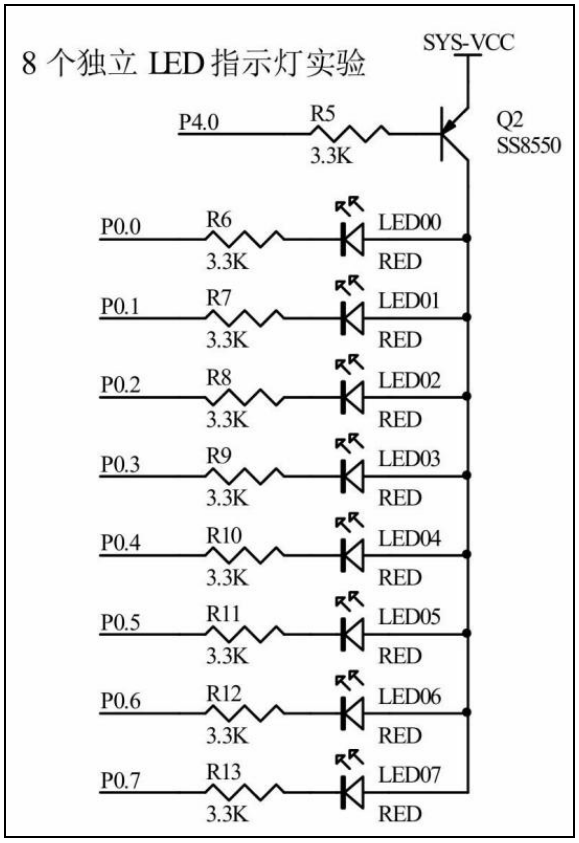
定时器 1 模式 0 结构图：此模式下定时器/计数器 1 作为可自动重装载的 16 位计数器，如果启动 8 位预分频 TM1PS 就是 24 位定时器。如下图所示：



定时器/计数器 1 的模式 0：16 位自动重装载模式，如果启动 8 位预分频 TM1PS 就是 24 位定时器

c) 配合本实验的 LED 指示灯原理图

LED 指示灯原理图
P0 口都有单独的指示灯



I/O 口 P4.0 控制 PNP 三极管的通断，控制这 8 个灯的整体供电。
如果 P4.0 输出 1，则 PNP 三极管不导通，这 8 个灯都不会获得电源，都不会有电流流过，都不会被点亮。
如果 P4.0 输出 0，则 PNP 三极管导通，这 8 个灯才有可能会流过电流，才有可能会被点亮。
如 P4.0 输出 0，PNP 三极管导通，特定 LED 灯的控制 I/O 口如输出 0，则该 LED 灯会有电流流过，会被点亮。
如 P0.0 输出 0，LED00 灯就会被点亮；如 P0.0 输出 1，LED00 灯就会熄灭；
如 P0.1 输出 0，LED01 灯就会被点亮；如 P0.1 输出 1，LED01 灯就会熄灭；
如 P0.2 输出 0，LED02 灯就会被点亮；如 P0.2 输出 1，LED02 灯就会熄灭；
如 P0.3 输出 0，LED03 灯就会被点亮；如 P0.3 输出 1，LED03 灯就会熄灭；
如 P0.4 输出 0，LED03 灯就会被点亮；如 P0.4 输出 1，LED03 灯就会熄灭；
如 P0.5 输出 0，LED03 灯就会被点亮；如 P0.5 输出 1，LED03 灯就会熄灭；
如 P0.6 输出 0，LED03 灯就会被点亮；如 P0.6 输出 1，LED03 灯就会熄灭；
如 P0.7 输出 0，LED03 灯就会被点亮；如 P0.7 输出 1，LED03 灯就会熄灭；

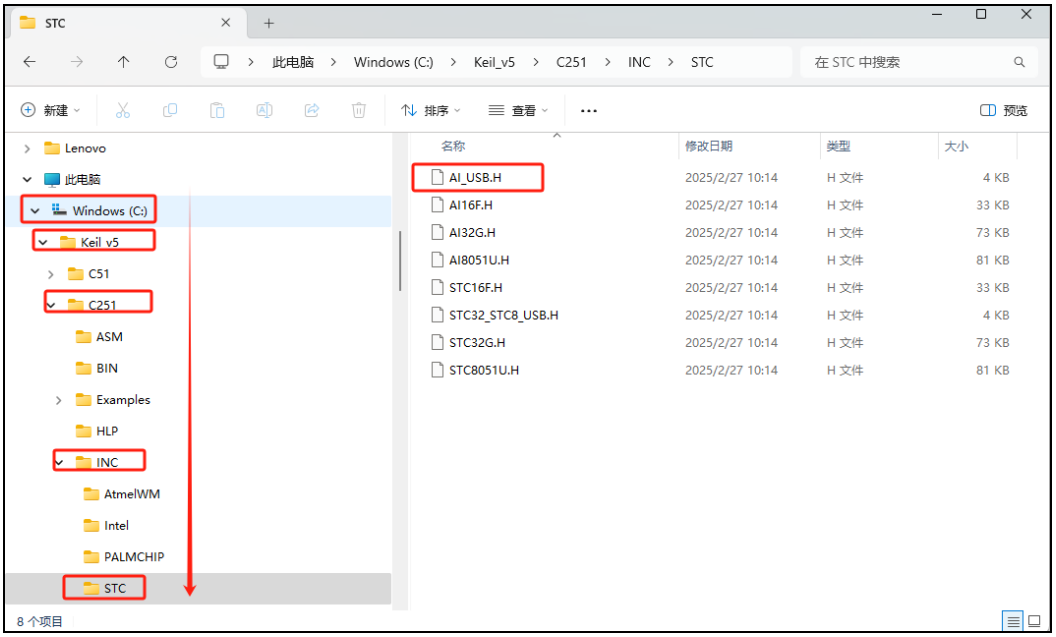
如 P0 口对外全部送 0，则这 8 个灯全亮；
如 P0 口对外全部送 1，则这 8 个灯全灭；
如果 P4.0 输出 1，则 PNP 三极管不导通，不管 P0 口对外全部送 0 或送 1，这 8 个灯都不会有电流流过，都不会亮。P4.0 控制 PNP 三极管，作为这 8 个 LED 指示灯的总电源开关。

d) 准备工作--添加头文件和库函数

1、添加常用头文件 ai_usb.h

#include "ai_usb.h"　　//使用 usb 所需要包含的头文件。

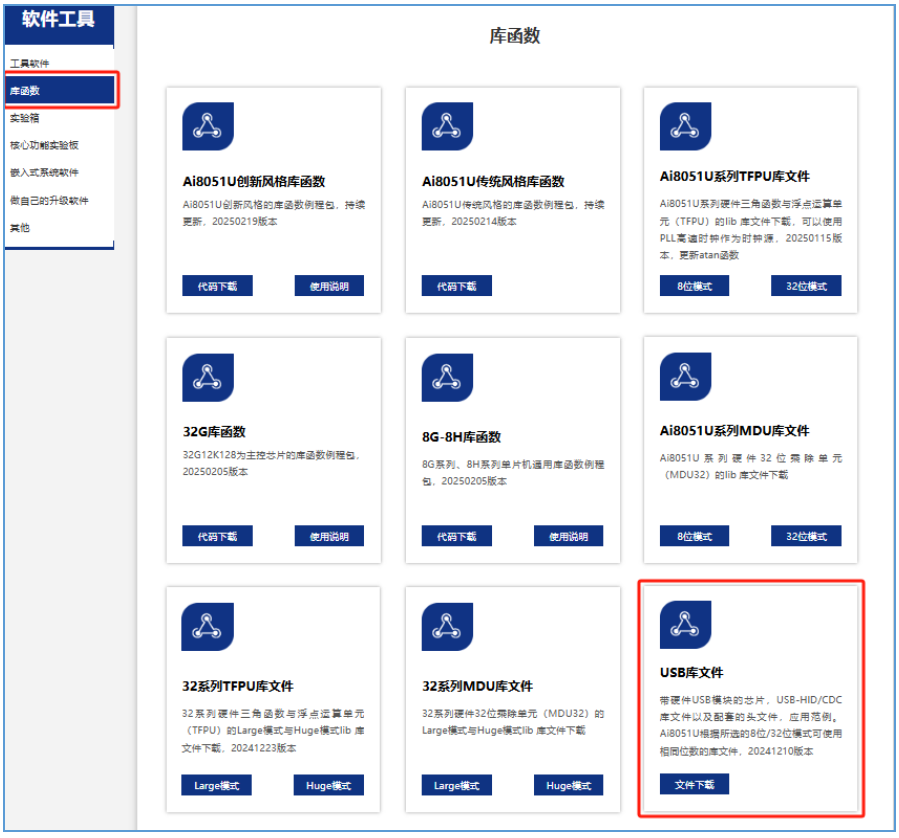
这个头文件在使用 AIapp-ISP 软件添加型号和头文件到 Keil 中时就已经一并被添加到 Keil 中了。



2、添加 usb_cdc 库函数文件

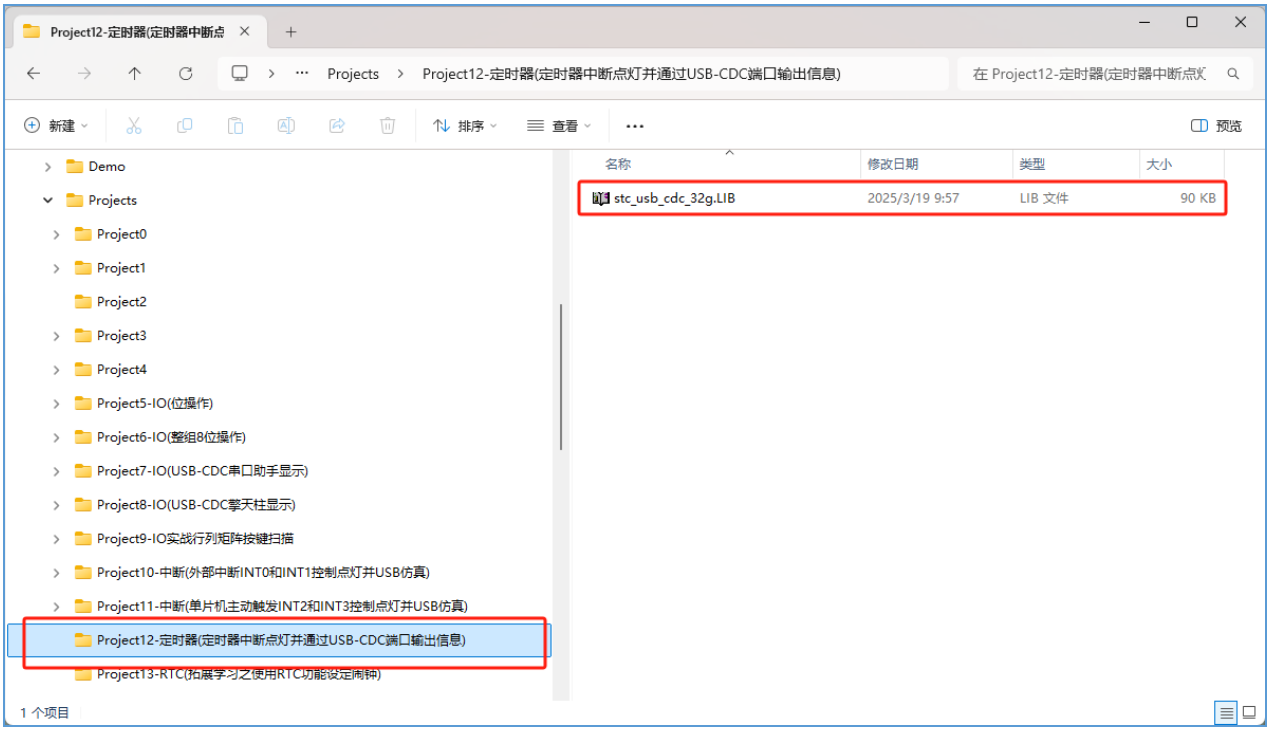
因为要使用到 USB 功能，必须在项目中添加 USB 库函数。这部分可以到 STCAI 的官网：

https://www.stcai.com/khs 的 USB 库文件中进行下载

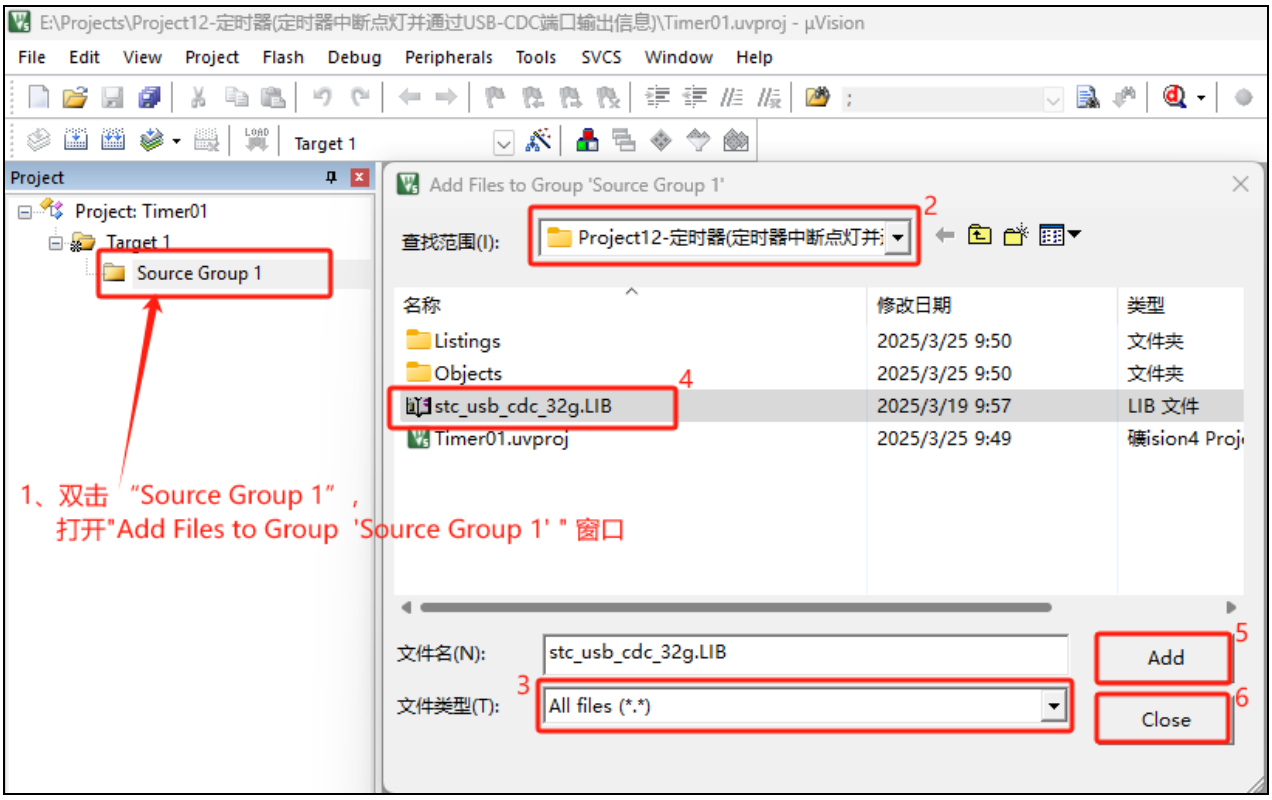


下载完成后，打开压缩包“STC_USB_LIBRARY.zip”，
在’库文件\查询模式 STC-CDC 库文件’中，复制出来 stc_usb_cdc_32g.LIB

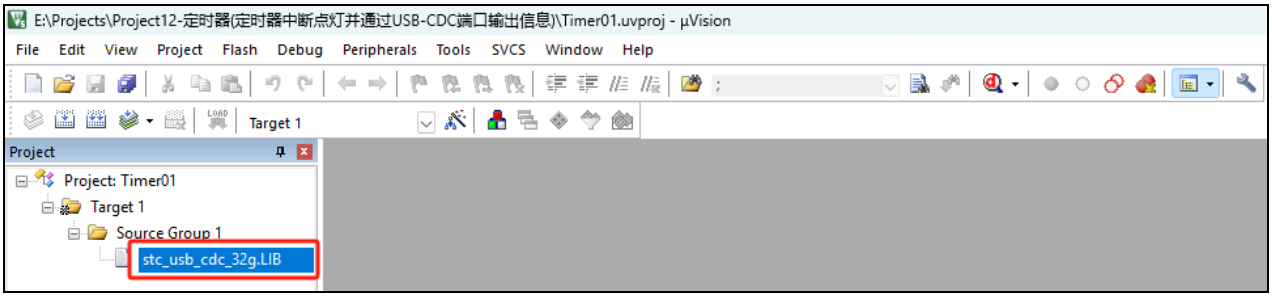
将库函数“stc_usb_cdc_32g.LIB”复制到创建工程所在的文件夹 project12 中



将库函数 stc_usb_cdc_32g.LIB 添加进入此工程项目中
在 Keil 中，双击 Source Group，在弹出的 Add Files to Group 'Source Group 1 窗口中，将库函数 stc_usb_cdc_32g.LIB 添加进入此工程，就可以使用 USB-CDC 进行端口通讯。



这样，Keil 界面变化如下图：



【注意】：

USB-CDC 串口则更类似传统串口的形式，类似 COM 口通讯，但不管串口波特率设置，Ai8051U 默认是 USB2.0-FS，12Mbps 在通讯。

手机不需要安装任何 USB 驱动，如：USB-CDC 驱动，USB-HID 驱动 / 任何系统都不需要安装。

现在的 Windows 操作系统也不需要安装 USB-CDC 驱动，只有 Win10(1903 版)之前的操作系统才需要安装 USB-CDC 驱动。安装也只是在 AIapp-ISP V6.95G 或更新的版本中执行【USB 驱动安装】菜单中的【安装 USB-CDC 驱动】即可。

e) 实验程序代码

```
#include <AI8051U.H>        //AI8051U 的通用头文件，头文件不分大小写
#include   "ai_usb.h"    //USB 库和 USB 虚拟设备的头文件
//由于在 USB 库函数的代码中已经定义了，不停电 ISP 下载命令字"@STCISP#"，
//只需要在下载软件的“收到用户命令后复位到 ISP 监控程序区”选项卡中进行相应的设置，
//即可实现 USB 不停电 ISP 下载功能，或串口不停电 ISP 下载功能。
/*
```

如何获得上面这 2 个头文件，如已按照前面介绍的方法：

从 www.STCAI.com 网站，【软件工具 | 工具软件】处，下载安装最新的【AIapp-ISP 软件】，
选择其中的选项卡 【Keil 仿真设置 】，
点击【添加型号和头文件到 Keil 中，并添加仿真器驱动到 Keil 中】按钮，
则 Ai8051U.h 和 AI_usb.h 都会自动复制到 Keil 中，包含进来就可以直接使用，
大家可以打开看下其中的内容，AI_usb.h 和 stc32_stc8_usb.h 是相似的，只需用其中的 1 个头文件，
具体见 Keil_v5\C251\INC\STC 目录，或 Keil_v5\C51\INC\STC 目录。
如将对应的 USB 库文件添加到 项目中，后续用户程序中，USB 如收到命令字符串 "@STCISP#"
USB 库函数就会自动软复位到系统 ISP 程序区，实现 USB 不停电下载，具体见相应部分的说明。
//char *USER_DEVICEDESC = NULL;
//char *USER_PRODUCTDESC = NULL;
//char *USER_STCISPCMD = "@STCISP#";
//使用 20250319 之前版本的库函数时需要在用户代码中增加上面 3 行定义。
//使用 20250319 及以后的版本都不再需要上面的定义
*/

```
bit    t0_flag = 0;    //定义 1 个位变量，T0 事件位变量标志，记录定时器 0 已产生中断
//  供主循环查询定时器 0 是否已产生中断，在主循环中处理定时器 0 的中断事件任务，不堵塞其他中断
bit    t1_flag = 0;    //定义 1 个位变量，T1 事件位变量标志，记录定时器 1 已产生中断
//  供主循环查询定时器 1 是否已产生中断，在主循环中处理定时器 1 的中断事件任务，不堵塞其他中断

void Timer0_Isr(void) interrupt TMR0_VECTOR      //定时器 0 中断服务程序
{
    P00 = ~P00;   //P00 灯闪烁，中断服务程序中尽量少执行长的任务，防止堵塞其他中断
                  //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
                  //但时间不要太长，否则会影响其他中断事件的实时响应速度
    t0_flag = 1;  // t0_flag 置 1 是通知主循环处理部分 T0 中断事件不需要特急处理的任务
                  //置 1，记录定时器 0 已产生中断，供主循环查询判断有无需处理的定时器 0 任务
}
//定时器 0 中断服务程序，TMR0_VECTOR 在 AI8051U.H 头文件中已宏定义为 1
void Timer0_Init(void)        //定时器 0 初始化，2 秒@40.000MHz
{
    TM0PS = 0x65;         //设置定时器时钟预分频 ( 注意:并非所有系列都有此寄存器,详情请查看数据手册 )
    AUXR &= 0x7F;         //定时器时钟 12T 模式
    TMOD &= 0xF0;         //设置定时器模式
    TL0 = 0xB1;           //设置定时初始值
    TH0 = 0x00;           //设置定时初始值
    TF0 = 0;              //清除 TF0 标志
    TR0 = 1;              //定时器 0 开始计时
    ET0 = 1;              //使能定时器 0 中断
}

void    Timer1_Isr(void)    interrupt    TMR1_VECTOR
{
    P07 = ~P07;       //P07 灯闪烁，中断服务程序中尽量少执行长的任务，防止堵塞其他中断
                      //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
                      //但时间不要太长，否则会影响其他中断事件的实时响应速度
    t1_flag = 1;      // t1_flag 置 1 是通知主循环处理部分 T1 中断事件不需要特急处理的任务
```

```
                                    //置 1，记录定时器 1 已产生中断，供主循环查询判断有无需处理的定时器 1 任务
}
//定时器 1 中断服务程序，TMR1_VECTOR 在 AI8051U.H 头文件中已宏定义为 3
void Timer1_Init(void)        //定时器 1 初始化，500 毫秒@40.000MHz
{
    TM1PS = 0x19;             //设置定时器时钟预分频 ( 注意:并非所有系列都有此寄存器,详情请查看数据手册 )
    AUXR &= 0xBF;             //定时器时钟 12T 模式
    TMOD &= 0x0F;             //设置定时器模式
    TL1 = 0x99;               //设置定时初始值
    TH1 = 0x05;               //设置定时初始值
    TF1 = 0;                  //清除 TF1 标志
    TR1 = 1;                  //定时器 1 开始计时
    ET1 = 1;                  //使能定时器 1 中断
}

void   sys_init(void)         //系统初始化
{
    EAXFR = 1;                //允许访问扩展的特殊寄存器，XFR
    WTST = 0;                 //设置取程序代码等待时间，赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                //设置访问片内的 xdata 速度，赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00; P0M1 = 0x00;     //设置 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x00;     //设置 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;     //设置 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x00;     //设置 P3 口为准双向口模式
    P4M0 = 0x00; P4M1 = 0x00;     //设置 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;     //设置 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;     //设置 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;     //设置 P7 口为准双向口模式
}

void   main(void)
{
    sys_init();     //调用系统初始化函数
    usb_init();     //调用 USB 初始化函数，不需要立即判断电脑已正确识别到 USB 从设备
    /* USB 型单片机从设备，如需要主动向电脑发送数据，
       在执行 USB_SendData( )函数和 printf_usb( )函数时，
       这两个函数已增加了判断电脑是否已正确识别到 USB 从设备的程序。
       如果电脑要主动发送数据给 USB 从设备，电脑自己会主动判断与 USB 从设备是否已正确连接。
    */
    EA = 1;                       //总中断允许位打开

    Timer0_Init();                //调用定时器 0 初始化函数
    Timer1_Init();                //调用定时器 1 初始化函数

    P40 = 0;                      //给 LED 灯供电
```

```
    while(1)            //主循环中查询需要处理的各种事件，如 T0/T1 中断事件，并打印输出状态
    {
        if(t0_flag)     //主循环中查询，定时器 0 是否已产生中断，是否有需要处理的定时器 0 事件
        {
            t0_flag = 0;            //清 0，T0 事件位变量标志
            printf_usb("Timer0！\r\n");
            //向电脑 USB-CDC 串口助手输出“Timer0！”字符串，代表主循环在处理 T0 不急的任务
        }
        if(t1_flag)     //主循环中查询，定时器 1 是否已产生中断，是否有需要处理的定时器 1 事件
        {
            t1_flag = 0;            //清 0，T1 事件位变量标志
            printf_usb("Timer1！\r\n");
            //向电脑 USB-CDC 串口助手输出“Timer1！”字符串，代表主循环在处理 T1 不急的任务
        }
    }
}
```

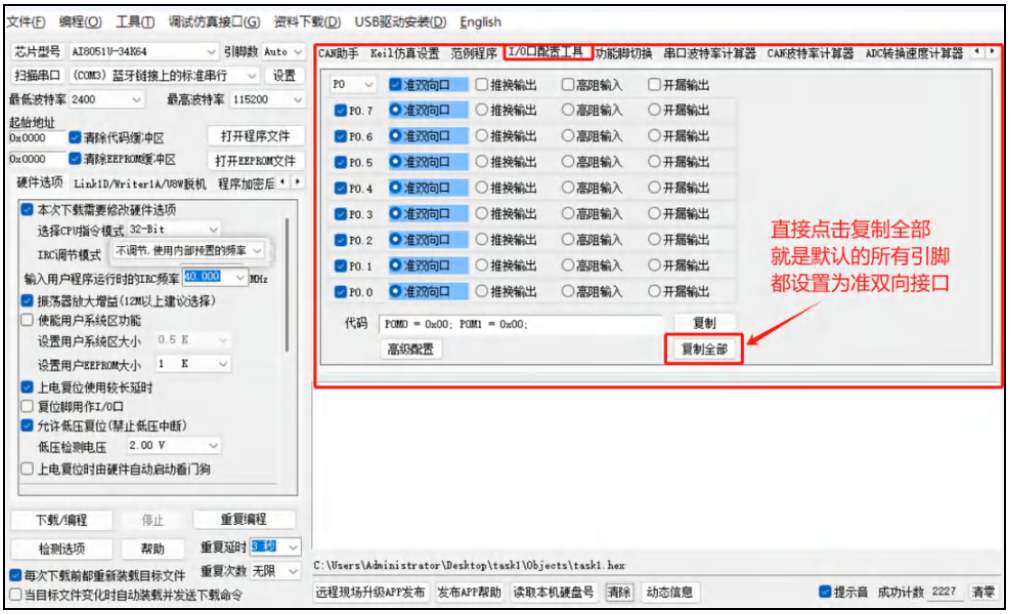
【小技巧（一）-- 灵活运用 AIapp-ISP 软件的工具自动生成代码】

I/O 口初始化代码部分就可以使用 **AIapp-ISP** 软件的“**I/O 口配置工具**”来生成。

例如上述代码中这一段将所有 I/O 口设置为准双向口模式的代码：

```
P0M0 = 0x00; P0M1 = 0x00;
P1M0 = 0x00; P1M1 = 0x00;
P2M0 = 0x00; P2M1 = 0x00;
P3M0 = 0x00; P3M1 = 0x00;
P4M0 = 0x00; P4M1 = 0x00;
P5M0 = 0x00; P5M1 = 0x00;
P6M0 = 0x00; P6M1 = 0x00;
P7M0 = 0x00; P7M1 = 0x00;
```

就可以使用 AIapp-ISP 的 I/O 口配置工具进行生成：



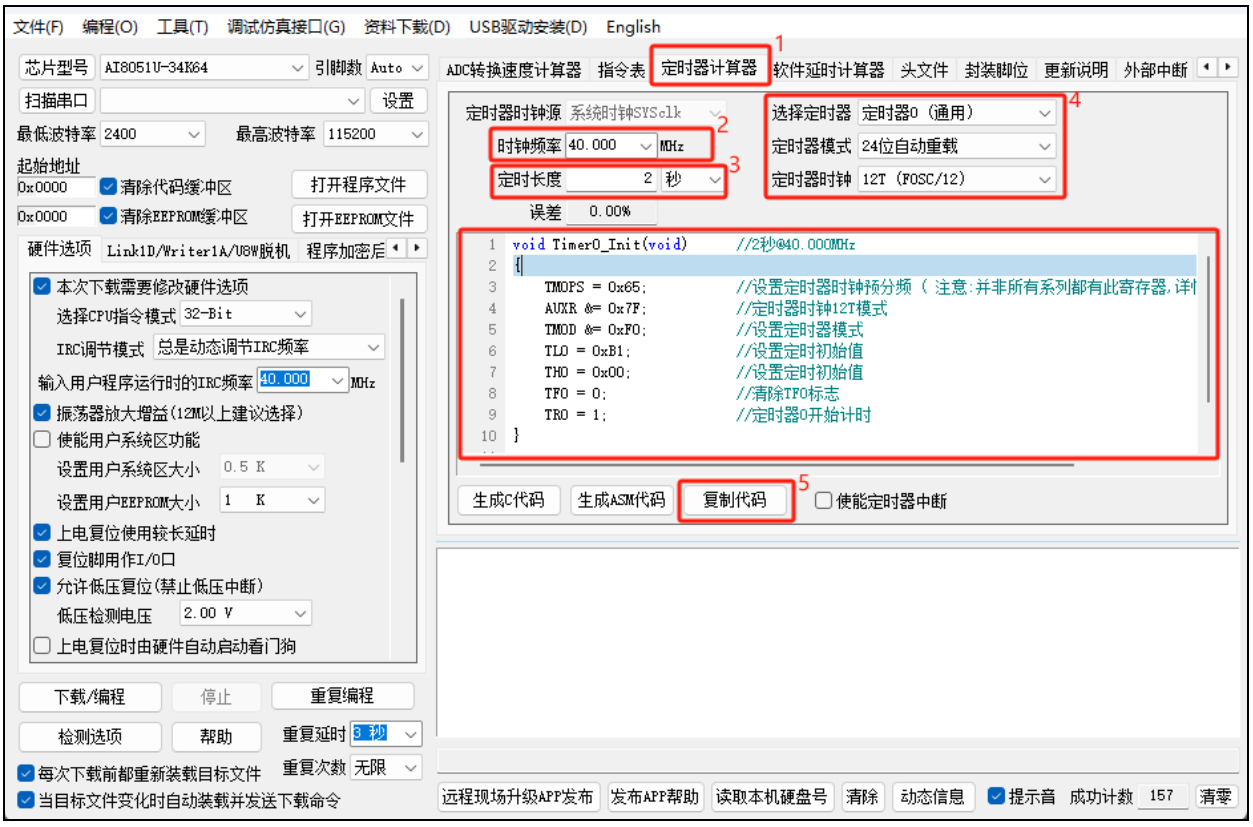
将复制出来的内容直接粘贴到代码文件中，不用手动输入。

【小技巧（二）-- 灵活运用 AIapp-ISP 软件的工具自动生成代码】

定时器 0 初始化代码部分就可以使用 **AIapp-ISP** 软件的“**定时器计算器**”来生成。

上述代码中 Time0_Init(void)这一段代码，按要求设置：时钟频率 40.00Hz，定时长度 2s，选择定时器 0、24 位自动重载、定时器时钟 12T。**AIapp-ISP 软件“定时器计算器”窗口就会自动生成如下代码：**

```
TM0PS = 0x65;    //设置定时器时钟预分频( 注意:并非所有系列都有此寄存器,详情请查看数据手册)
AUXR &= 0x7F;    //定时器时钟 12T 模式
TMOD &= 0xF0;    //设置定时器模式
TL0 = 0xB1;      //设置定时初始值
TH0 = 0x00;      //设置定时初始值
TF0 = 0;         //清除 TF0 标志
TR0 = 1;         //定时器 0 开始计时
ET0 = 1;         //使能定时器 0 中断就可以使用 AIapp-ISP 的 I/O 口配置工具进行生成：
```



将复制出来的内容直接粘贴到代码文件中，不用手动输入。

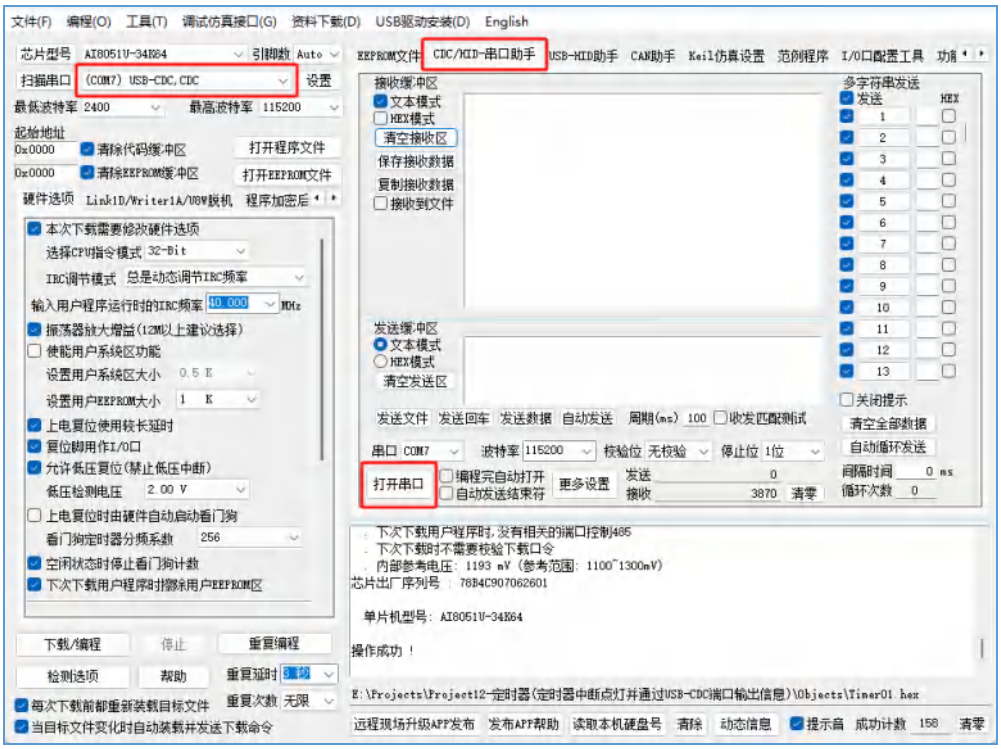
定时器 1 的初始化代码部分获取方法同上操作。

f) 程序下载

程序编译，并下载实验箱完成以后，就可以看到出现了一个 USB-CDC 端口，此时就可以通过这个 USB-CDC 端口进行数据通讯了。

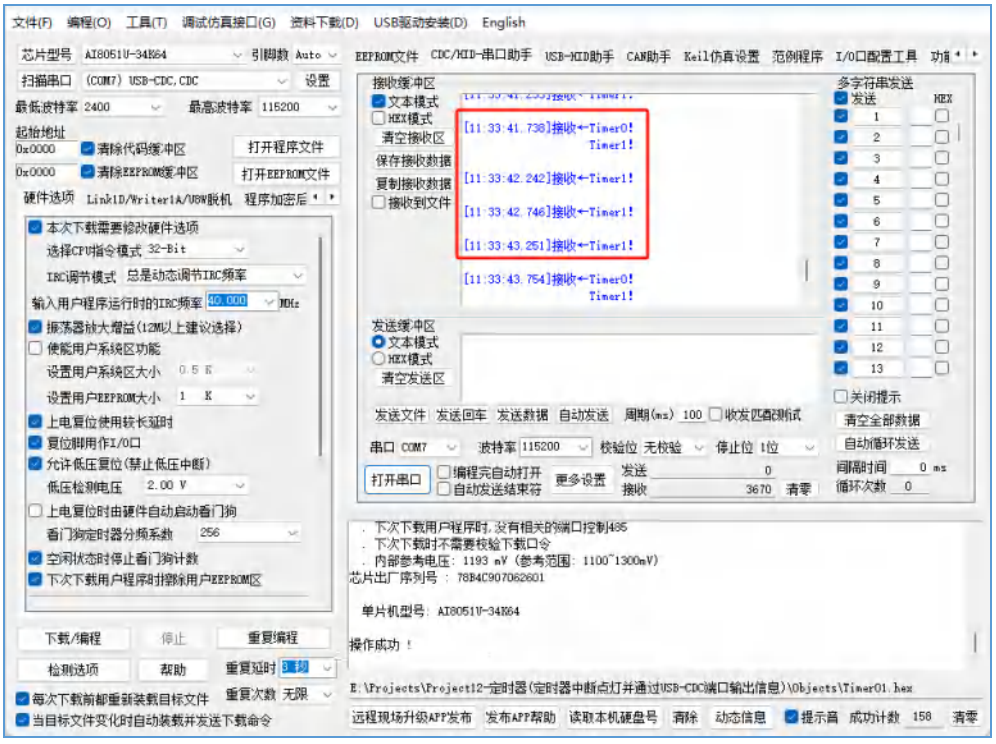
g) 程序实现效果

这时，在 AIapp-ISP 软件的 CDC 串口助手选项卡中找到“打开串口”：



点击“打开串口”

我们观察到：在 AIapp-ISP 软件的 CDC 串口助手窗口，一直显示接收时间及接收的字符串内容“Time0”和“Time1”，并且如本实验代码中设置的每触发 4 次 Time1 中断，触发 1 次 Time0。如下图：



实验箱上，P07 灯每亮灭各 2 次，P00 灯亮 1 次。

达成实验预设，实验结束。

17.8.2　深大--使用【AiCube】工具实现：定时器 0/1，24 位定时器实验（定时器中断点灯并通过 USB-CDC 端口输出信息）

STCAI 在最新版的 AIapp-ISP 软件中，推出【AiCube】项目创建助手工具，目前支持 Ai8051U-32Bit 各功能模块的智能化配置，并生成程序框架。这项功能在程序员建立项目时，简化了项目的初始化难度，提高效率，避免因不太熟悉或不完全了解该单片机的性能而忽略个别配置，引起编译错误等问题。尤其对刚涉足单片机的初学者，此工具不仅引导其了解单片机编程流程，还极大的降低了入门门槛，缩短了摸索的时间，提高入门的兴趣和信心。现在我们使用这个工具来生成实现此实验功能的配置和代码。

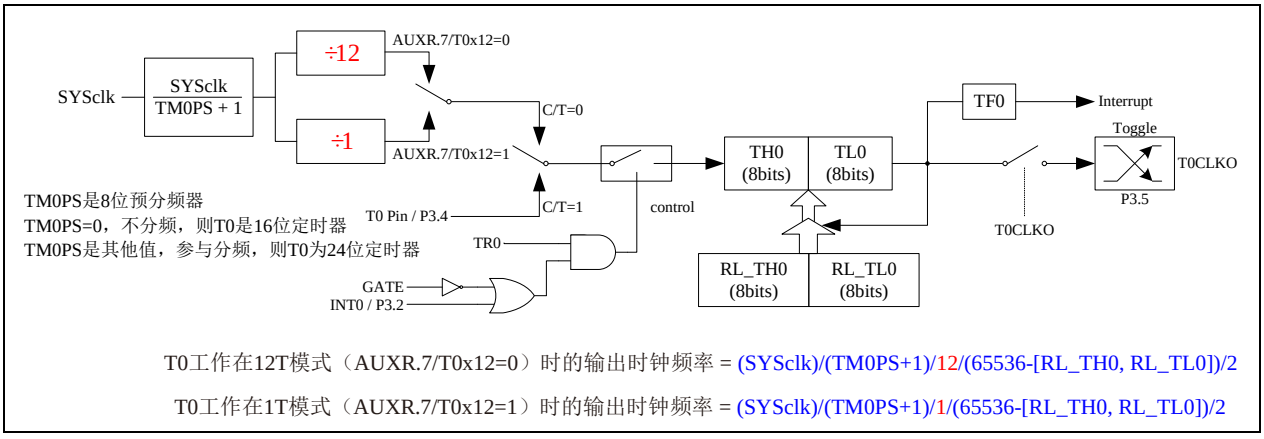
a) 实验介绍

定时器 0（Timer0）定时 2s 产生中断，P00 灯闪烁，CDC 窗口显示“Timer0！”
定时器 1（Timer1）定时 500ms 产生中断，P07 灯闪烁，CDC 窗口显示“Timer1！”
观察到：

- AIapp-ISP 软件的 CDC 窗口每显示 4 次“Timer1”，显示 1 次“Timer0”
- 实验箱，P07 亮灭各 2 次，P00 亮 1 次

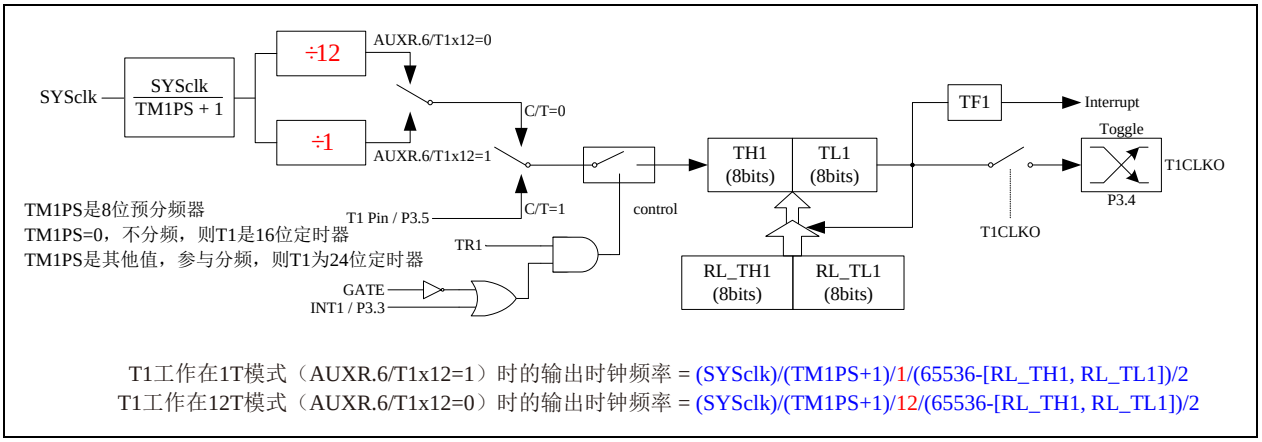
b) 定时器 0 模式 0 结构图，定时器 1 模式 0 结构图

定时器 0 模式 0 结构图：此模式下定时器/计数器 0 作为可自动重装载的 16 位计数器，如果启动 8 位预分频 TM0PS 就是 24 位定时器。如下图所示：



定时器/计数器 0 的模式 0：16 位自动重装载模式，如果启动 8 位预分频 TM0PS 就是 24 位定时器

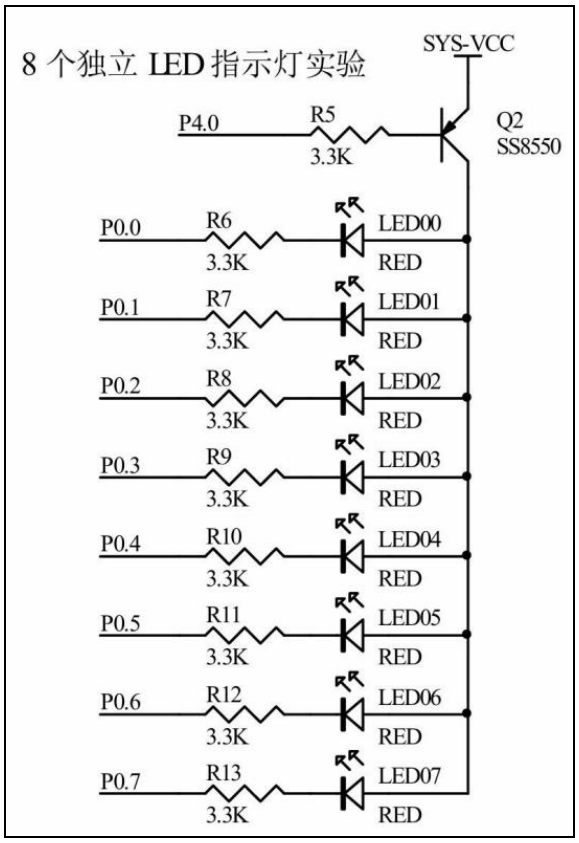
定时器 1 模式 0 结构图：此模式下定时器/计数器 1 作为可自动重装载的 16 位计数器，如果启动 8 位预分频 TM1PS 就是 24 位定时器。如下图所示：



定时器/计数器 1 的模式 0：16 位自动重装载模式，如果启动 8 位预分频 TM1PS 就是 24 位定时器

c) 配合本实验的 LED 指示灯原理图

LED 指示灯原理图
P0 口都有单独的指示灯



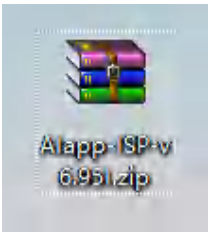
I/O 口 P4.0 控制 PNP 三极管的通断，控制这 8 个灯的整体供电。
如果 P4.0 输出 1，则 PNP 三极管不导通，这 8 个灯都不会获得电源，都不会有电流流过，都不会被点亮。
如果 P4.0 输出 0，则 PNP 三极管导通，这 8 个灯才有可能会流过电流，才有可能会被点亮。
如 P4.0 输出 0，PNP 三极管导通，特定 LED 灯的控制 I/O 口如输出 0，则该 LED 灯会有电流流过，会被点亮。
如 P0.0 输出 0，LED00 灯就会被点亮；如 P0.0 输出 1，LED00 灯就会熄灭；
如 P0.1 输出 0，LED01 灯就会被点亮；如 P0.1 输出 1，LED01 灯就会熄灭；
如 P0.2 输出 0，LED02 灯就会被点亮；如 P0.2 输出 1，LED02 灯就会熄灭；
如 P0.3 输出 0，LED03 灯就会被点亮；如 P0.3 输出 1，LED03 灯就会熄灭；
如 P0.4 输出 0，LED03 灯就会被点亮；如 P0.4 输出 1，LED03 灯就会熄灭；
如 P0.5 输出 0，LED03 灯就会被点亮；如 P0.5 输出 1，LED03 灯就会熄灭；
如 P0.6 输出 0，LED03 灯就会被点亮；如 P0.6 输出 1，LED03 灯就会熄灭；
如 P0.7 输出 0，LED03 灯就会被点亮；如 P0.7 输出 1，LED03 灯就会熄灭；

如 P0 口对外全部送 0，则这 8 个灯全亮；
如 P0 口对外全部送 1，则这 8 个灯全灭；
如果 P4.0 输出 1，则 PNP 三极管不导通，不管 P0 口对外全部送 0 或送 1，这 8 个灯都不会有电流流过，都不会亮。P4.0 控制 PNP 三极管，作为这 8 个 LED 指示灯的总电源开关。

d) 安装 AIapp-ISP 下载/编程/烧录/软件

安装 STCAI 公司研发的 AIapp-ISP 下载 / 编程 / 烧录 / 软件，含强大的辅助开发工具包。登录 STCAI 官网，下载最新版的 AIapp-ISP 安装包，下载链接如下：https://www.stcai.com/gjrj

下载下来的是压缩包文件如下图：

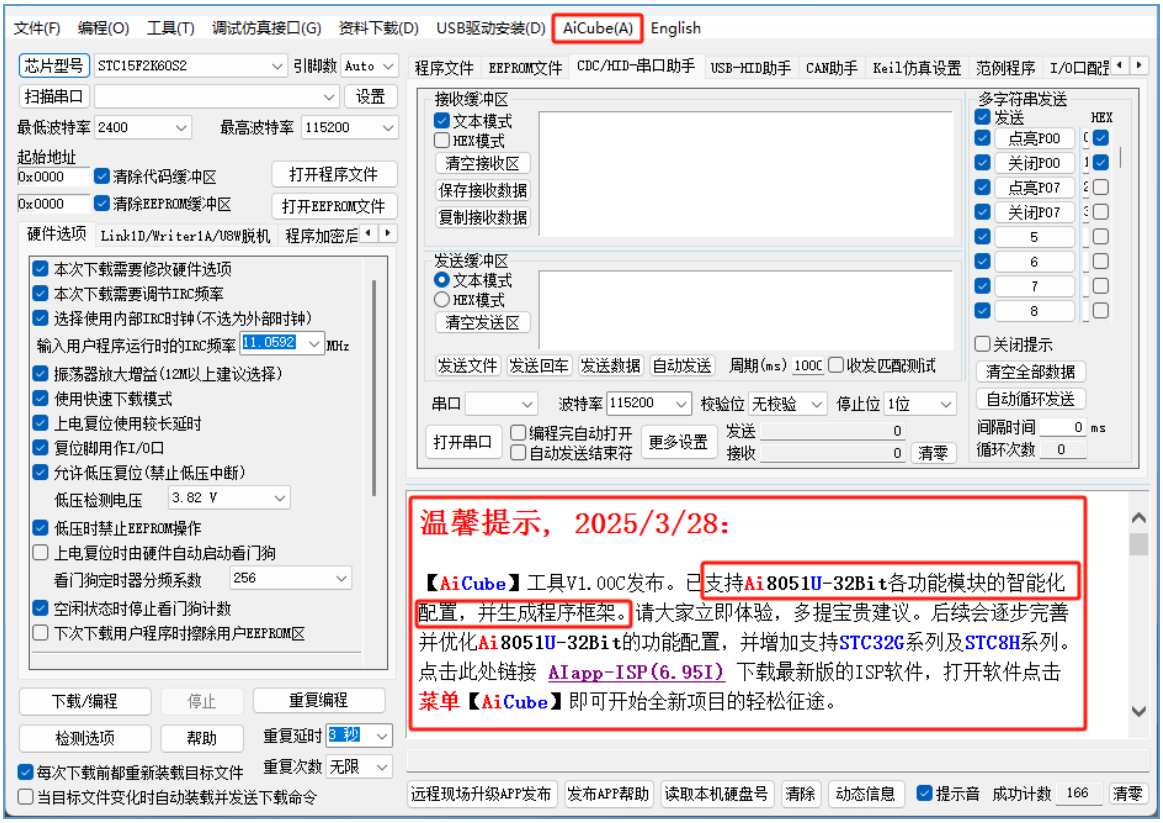


双击后解压缩如下图：

如果是解压缩到非桌面的其他目录，如 D:\STCAI-ISP，建议创建快捷方式到桌面方便打开。这实际是一个免安装的绿色软件。

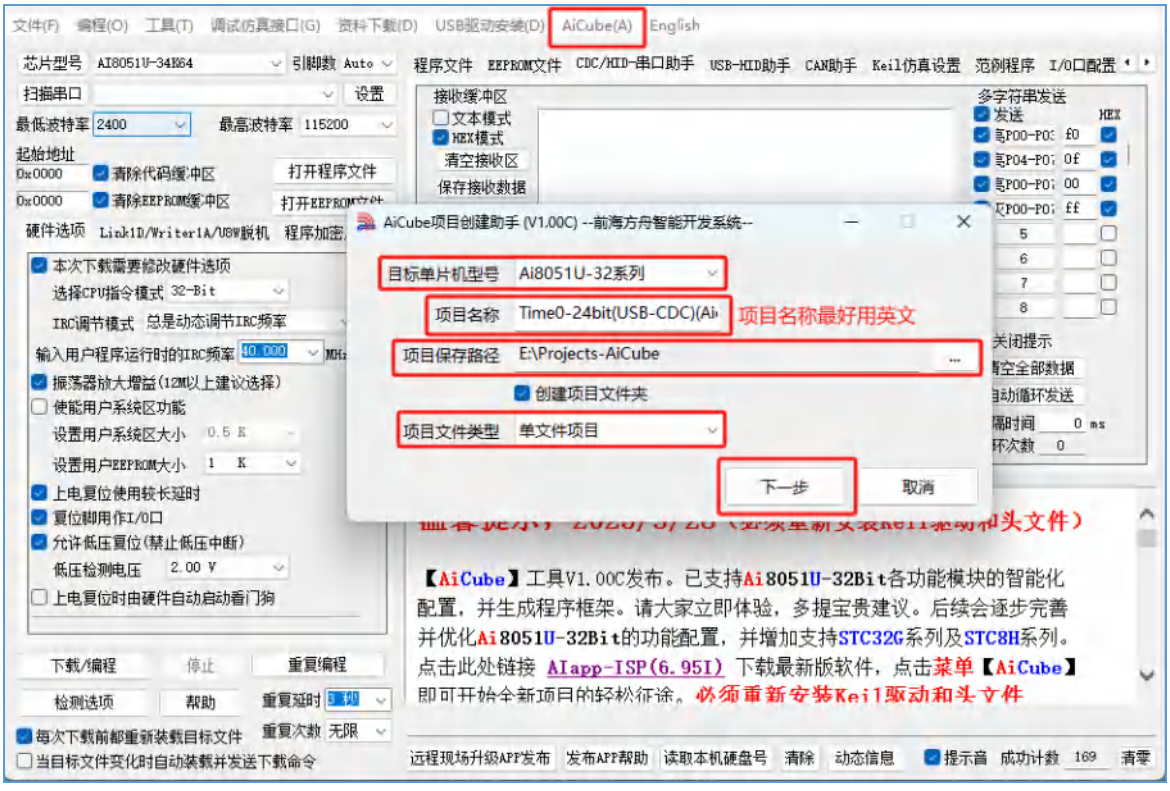
e) 打开【AiCube】

双击图标，即可打开 AIapp-ISP 软件界面，找到“AiCube(A)”菜单，如下图：

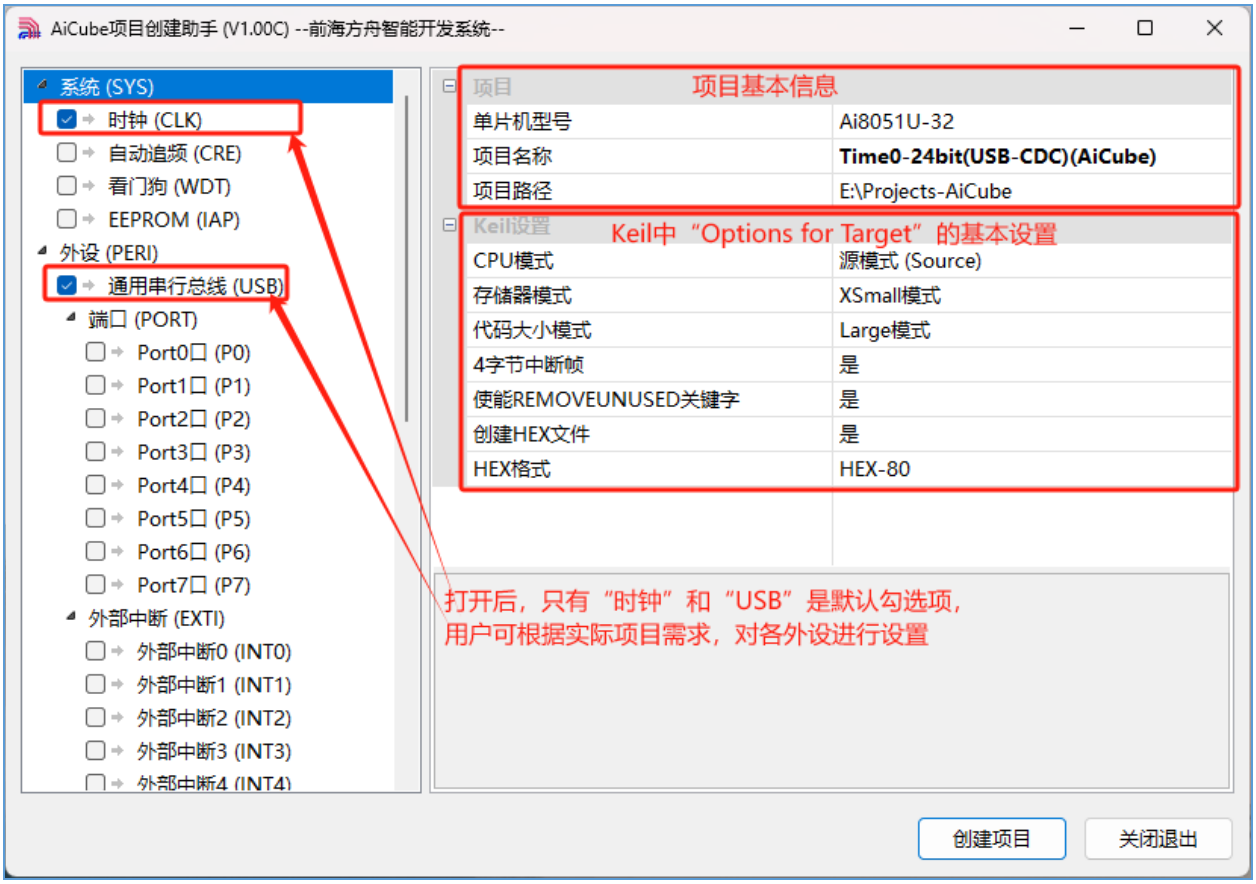


f) 使用【AiCube】创建项目

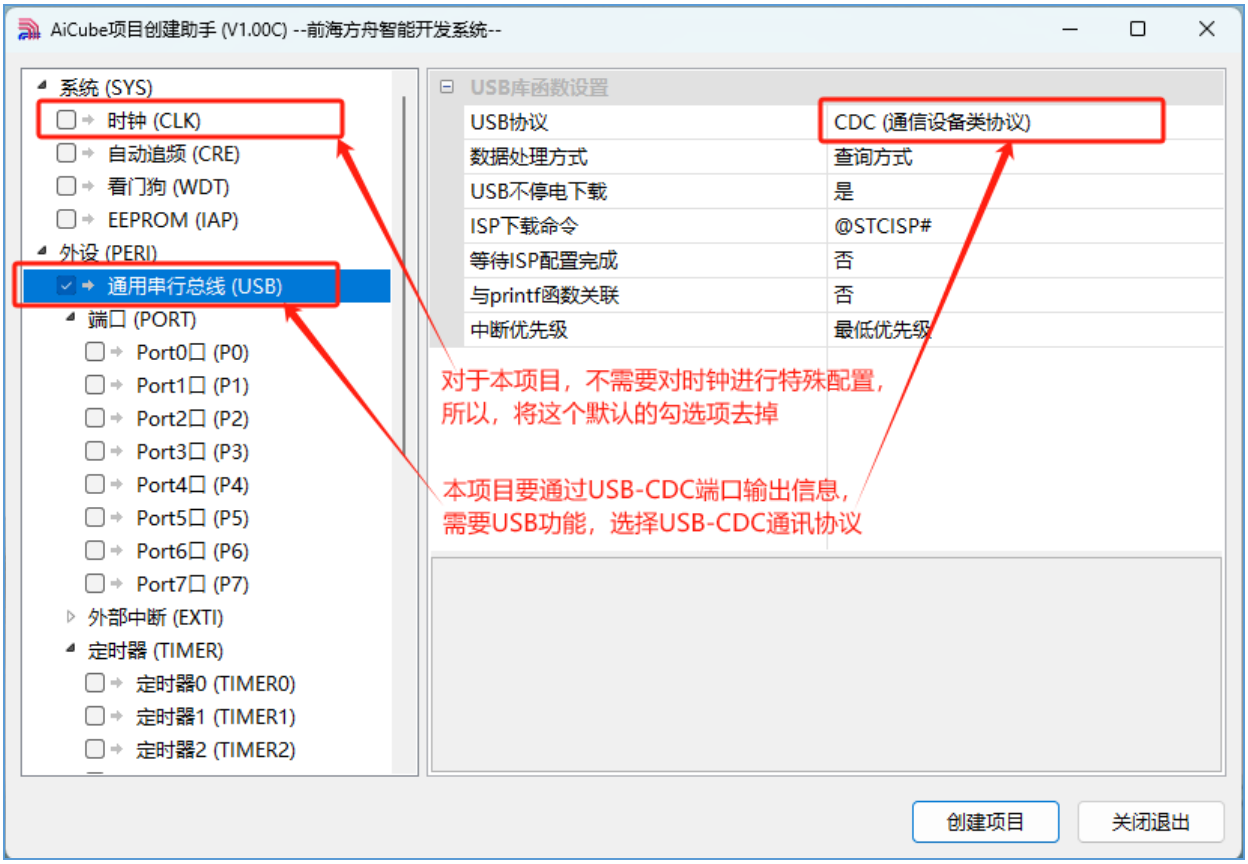
点击【AiCube（A）】，弹出如下窗口：



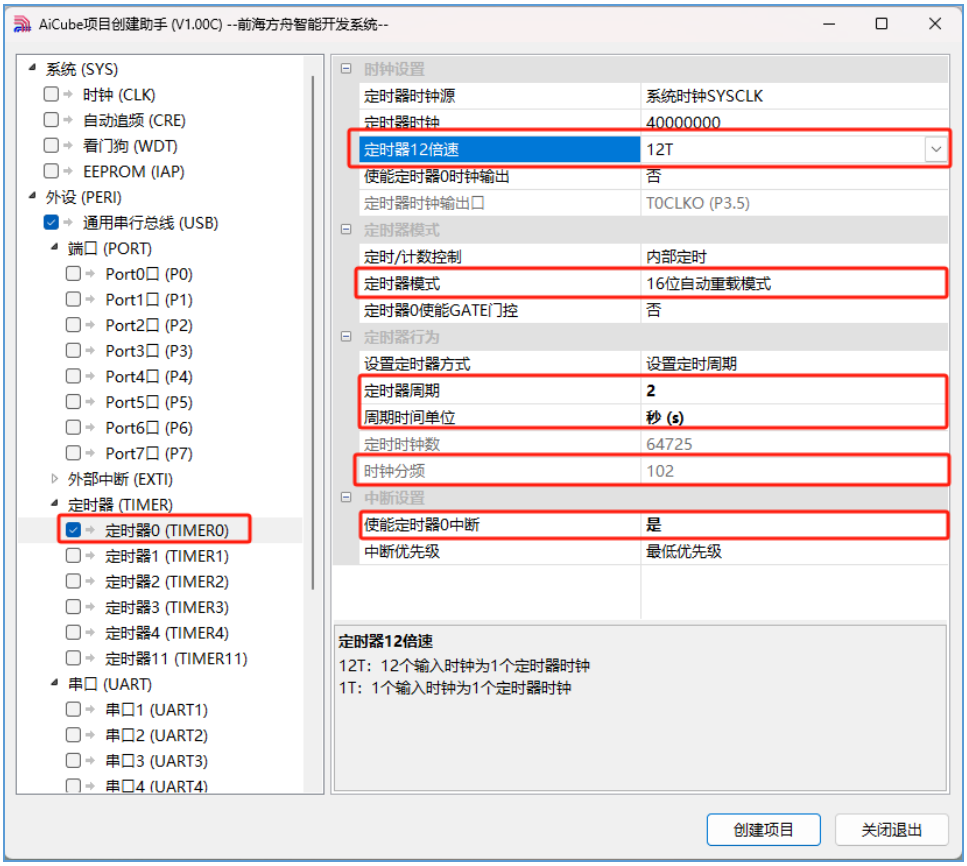
选择并填写项目基本信息，建议项目名称用英文！点击“下一步”，出现如下配置界面：



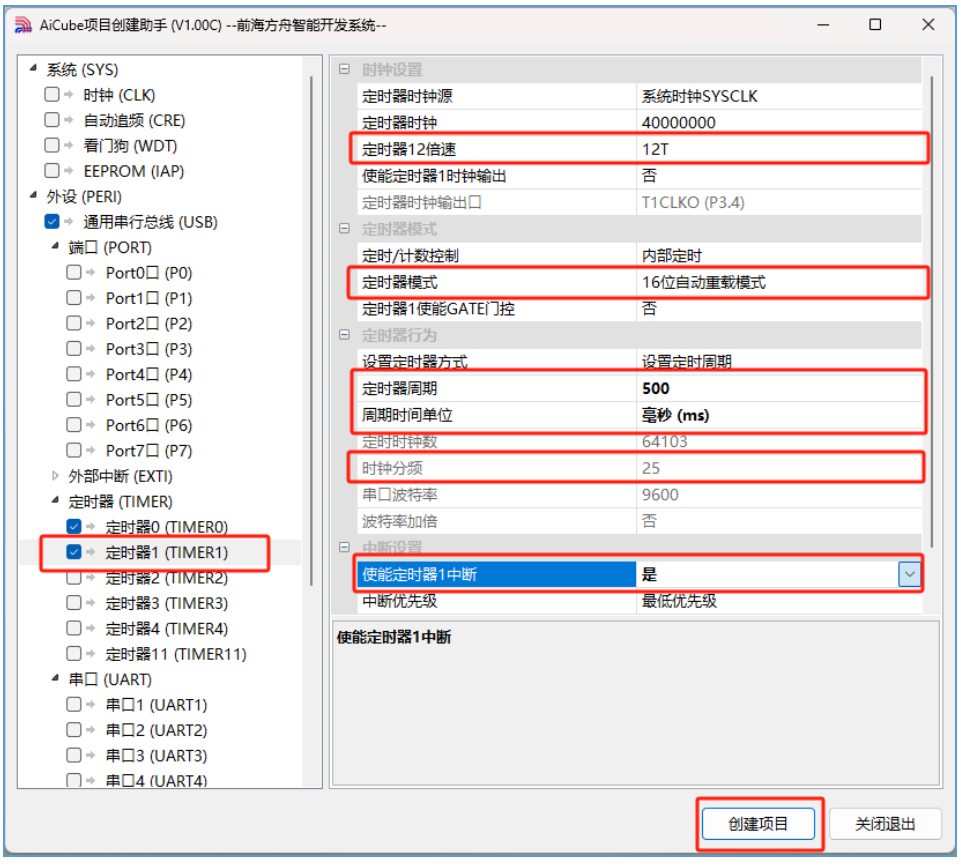
因本项目用到 USB 功能，要加载 USB 库函数、代码中要有 USB 初始化、USB-CDC 协议等，使用【AiCube】助手根据本项目的实际需求，对配置进行如下修改：



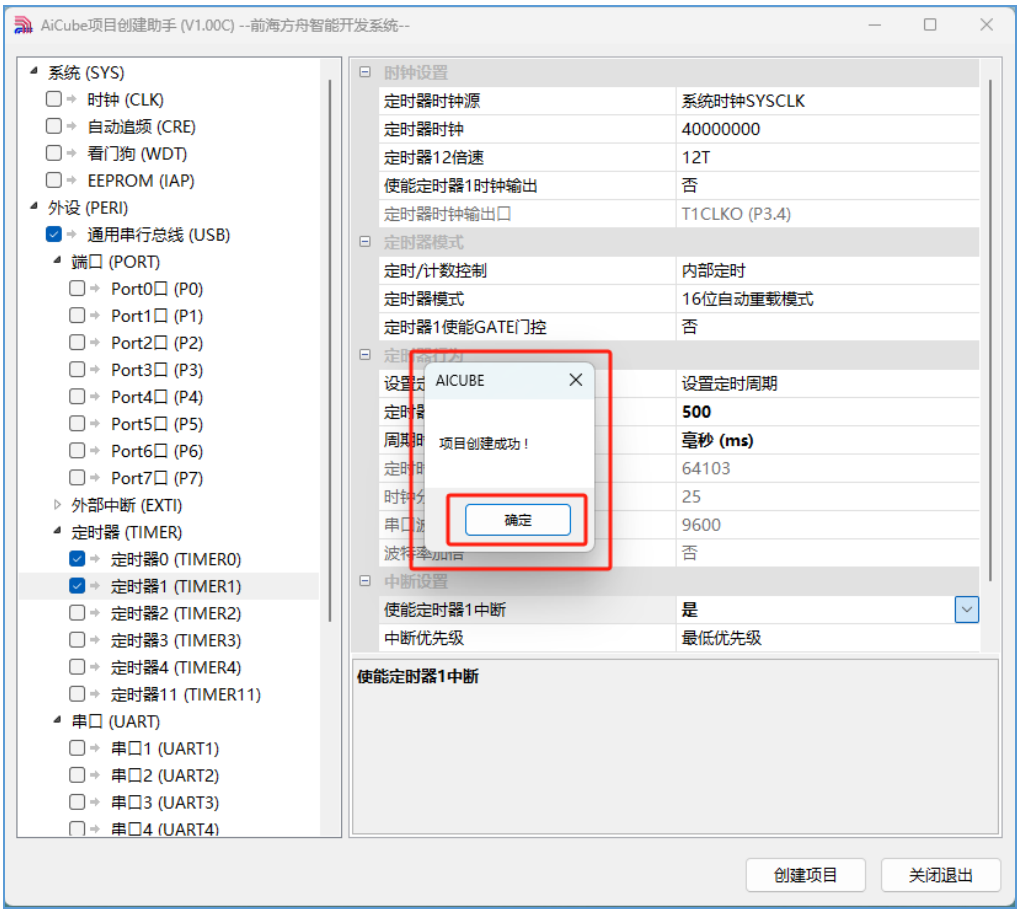
项目中需要实现【定时器 0（Timer0）定时 2s 产生中断，P00 灯闪烁，CDC 窗口显示“Timer0！”】，所以，需要【AiCube】助手对定时器 0 进行配置，如下图：



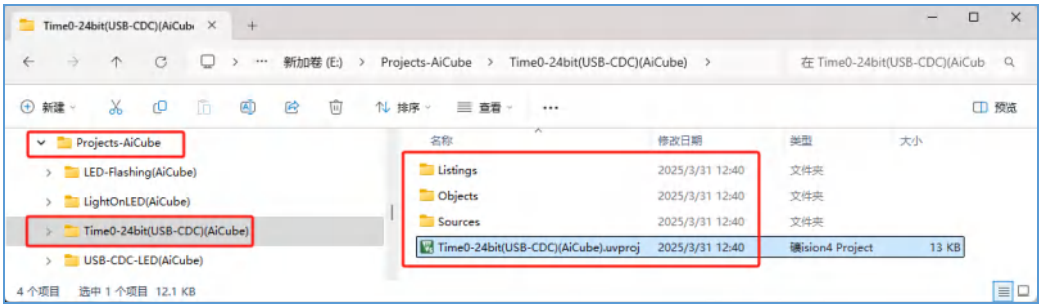
项目中需要实现【定时器 1（Timer1）定时 500ms 产生中断，P07 灯闪烁，CDC 窗口显示“Timer1！”】，所以，需要【AiCube】助手对定时器进行配置：



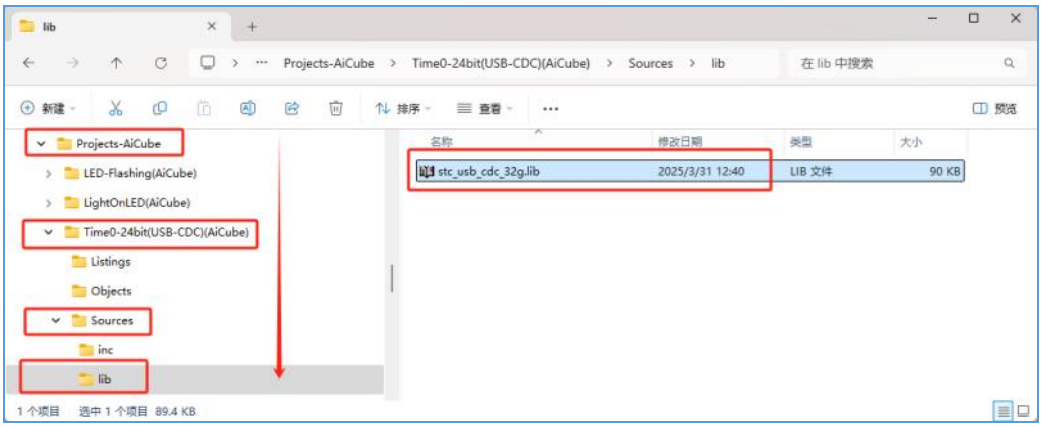
点击“创建项目”，【AiCube】助手提示：“项目创建成功！”，如下图：



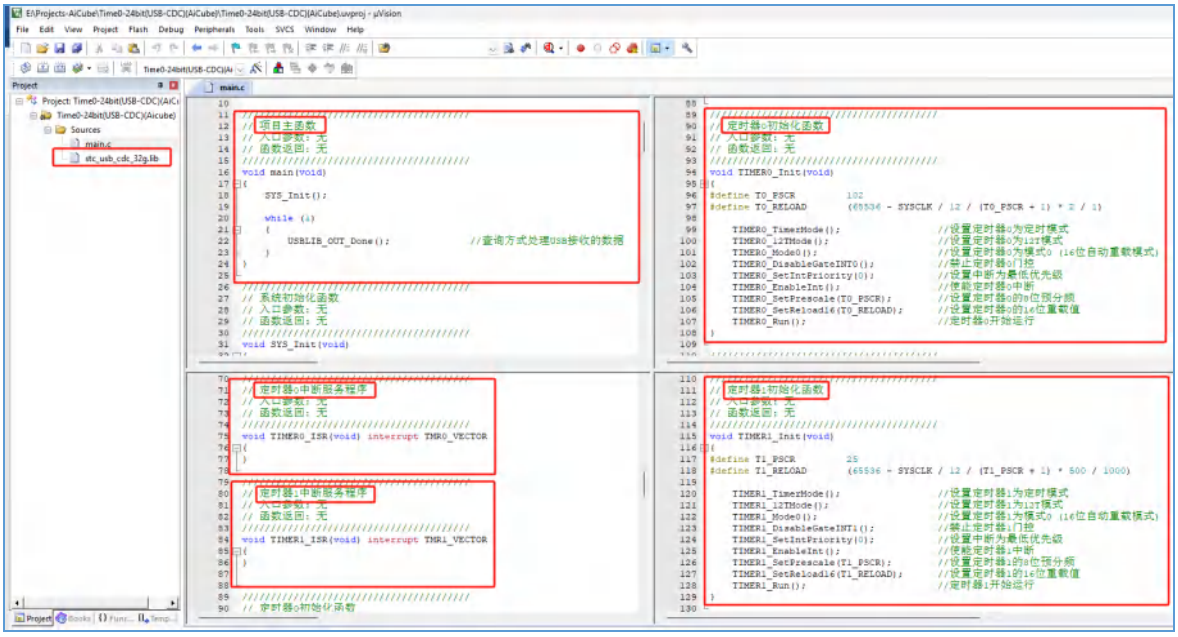
点击“确定”，即成功创建了项目及项目文件目录“Time0-24bit(USB-CDC)(AiCube)”，找到并打开刚刚创建的项目目录。发现刚刚的操作过程，【AiCube】助手已经帮我们建立了项目目录及 Time0-24bit(USB-CDC)(AiCube).uvproj 文件，如下图：



同时打开 Sources>lib 子目录，发现，【AiCube】助手连 USB 库函数也已经给加载了：



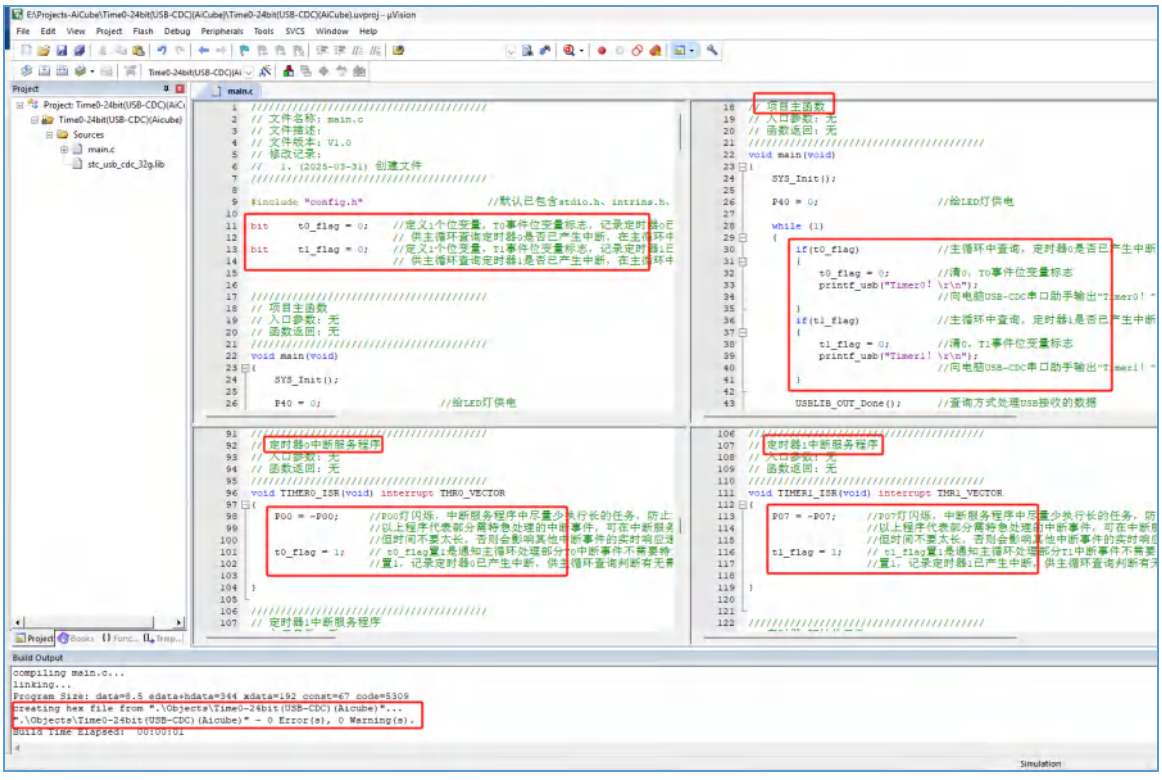
双击打开项目文件：Time0-24bit(USB-CDC)(AiCube).uvproj，如下图（为方便查看，使用菜单“Window”→“Split”将原窗口拆分出四个窗口，对比观察）：



根据在【AiCube】助手中的配置，从上图中我们看到【AiCube】助手创建的本实验项目中不仅有了 main.c，还已经自动加载了 USB 库函数（stc_usb_cdc_32g.lib）。main.c 中按模块化设计，已经包含了：必要的头文件、主函数、系统初始化函数、延时函数、定时器 Time0 初始化函数、定时器 Time0 中断服务函数、定时器 Time1 初始化函数、定时器 Time1 中断服务函数、USB 相关的配置及处理函数。

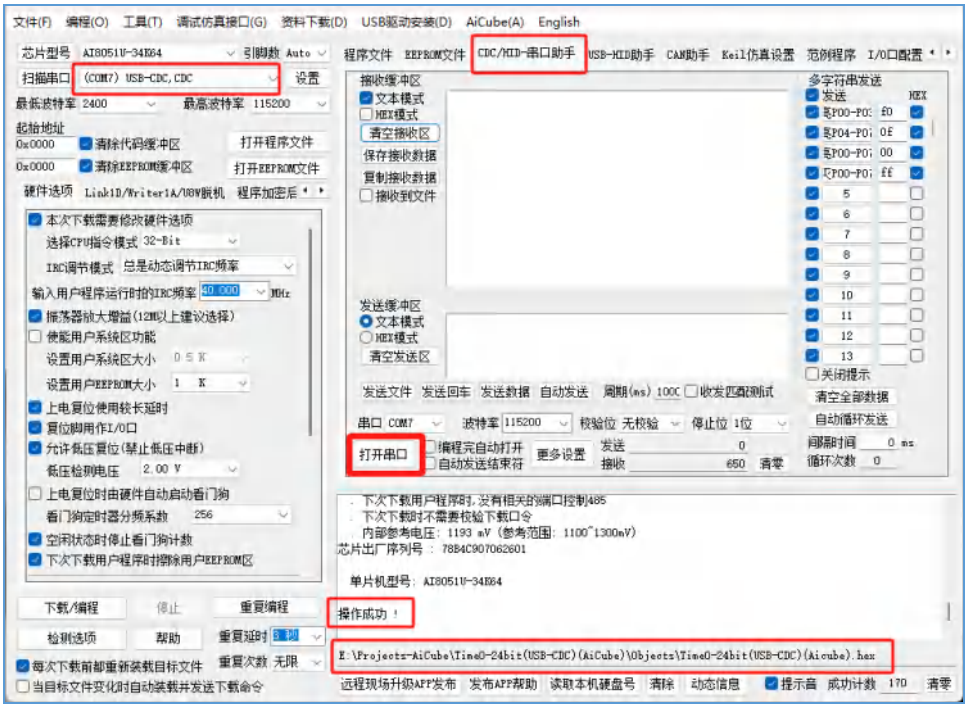
g) 增加实验项目代码

【AiCube】助手帮我们搭建了程序框架，为了实现本实验效果，我们现在需要在主函数、定时器 Time0 中断服务函数、定时器 Time1 中断服务函数中补充相应代码，保存，编译，如下图（为方便查看，使用菜单“Window”→“Split”将原窗口拆分出四个窗口，对比观察）：



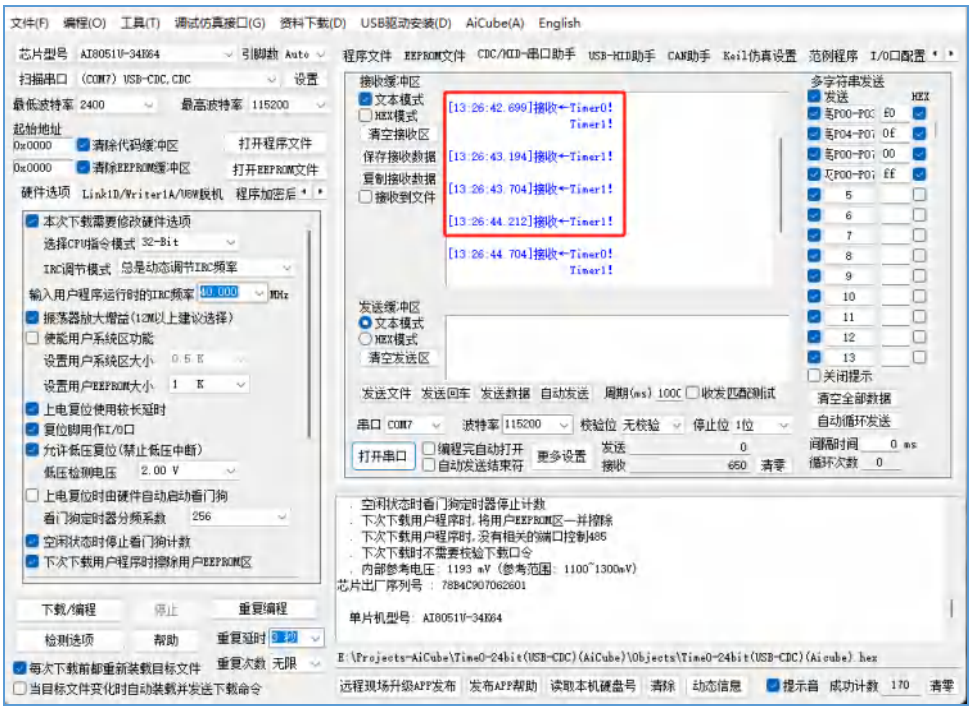
h) 使用 Ai8051U 实验箱查验实验效果

连接好 Ai8051U 实验箱，运行 AIapp-ISP 软件系统，按步骤打开并下载程序“Time0-24bit(USB-CDC)(AiCube).hex”，如下图：



点击“打开串口”

我们观察到：在 AIapp-ISP 软件的 CDC 串口助手窗口，一直显示接收时间及接收的字符串内容“Time0”和“Time1”，并且如本实验代码中设置的每触发 4 次 Time1 中断，触发 1 次 Time0。如下图：



实验箱上，P07 灯每亮灭各 2 次，P00 灯亮 1 次。

达成实验预设，实验结束。

有了【AiCube】助手，单片机编程就是这么简单！快用起来吧！

17.8.3 深大--定时器 0/1，INT0/INT1，中断组合应用实验，演示程序

```
//C 语言程序代码
#include <AI8051U.H>        //包含 AI8051U 的头文件

#include   "ai_usb.h"    //USB 库和 USB 虚拟设备的头文件，需要在项目文件中添加对应的 USB 库
           //由于在 USB 库函数的代码中已经定义了，不停电 ISP 下载命令字”@STCISP#”，
           //只需要在下载软件的“收到用户命令后复位到 ISP 监控程序区”选项卡中进行相应的设置，
           //即可实现 USB 不停电 ISP 下载功能，或串口不停电 ISP 下载功能。

#include "intrins.h"        //使用_nop_()函数所必须要包含的头文件,
                            //否则延时函数中调用的_nop_()函数没有被头文件引用过来，
                            //会导致编译器找不到这个而函数而报错。

bit   int0_flag = 0;  //定义 1 个位变量，INT0 事件位变量标志，记录 INT0 已产生中断
    //  供主循环查询 INT0 是否已产生中断，在主循环中处理 INT0 的中断事件任务，不堵塞其他中断
bit   int1_flag = 0;  //定义 1 个位变量，INT1 事件位变量标志，记录 INT1 已产生中断
    //  供主循环查询 INT1 是否已产生中断，在主循环中处理 INT1 的中断事件任务，不堵塞其他中断
bit   t0_flag = 0;    //定义 1 个位变量，T0 事件位变量标志，记录定时器 0 已产生中断
    //  供主循环查询定时器 0 是否已产生中断，在主循环中处理定时器 0 的中断事件任务，不堵塞其他中断
bit   t1_flag = 0;    //定义 1 个位变量，T1 事件位变量标志，记录定时器 1 已产生中断
    //  供主循环查询定时器 1 是否已产生中断，在主循环中处理定时器 1 的中断事件任务，不堵塞其他中断

void Timer0_Init(void)        //定时器 0 初始化，2 秒@40.000MHz
{
    TM0PS = 0x65;         //设置定时器时钟预分频 ( 注意:并非所有系列都有此寄存器,详情请查看数据手册 )
    AUXR &= 0x7F;         //定时器时钟 12T 模式
    TMOD &= 0xF0;         //设置定时器模式
    TL0 = 0xB1;           //设置定时初始值
    TH0 = 0x00;           //设置定时初始值
    TF0 = 0;              //清除 TF0 标志
    TR0 = 1;              //定时器 0 开始计时
    ET0 = 1;              //使能定时器 0 中断
}

void Timer1_Init(void)        //定时器 1 初始化，500 毫秒@40.000MHz
{
    TM1PS = 0x19;         //设置定时器时钟预分频 ( 注意:并非所有系列都有此寄存器,详情请查看数据手册 )
    AUXR &= 0xBF;         //定时器时钟 12T 模式
    TMOD &= 0x0F;         //设置定时器模式
    TL1 = 0x99;           //设置定时初始值
    TH1 = 0x05;           //设置定时初始值
    TF1 = 0;              //清除 TF1 标志
    TR1 = 1;              //定时器 1 开始计时
    ET1 = 1;              //使能定时器 1 中断
}
```

```
void main (void)
{
    EAXFR = 1;        //允许访问扩展的特殊寄存器，XFR
    WTST = 0;         //设置取程序代码等待时间，赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;        //设置访问片内的 xdata 速度，赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00; P0M1 = 0x00;     //设置 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x00;     //设置 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;     //设置 P2 口为准双向口模式
//  P3M0 = 0x00; P3M1 = 0x00;     //设置 P3 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x0c;     //P32、P33 设置为高阻输入(需要同步开启上拉电阻)
    P4M0 = 0x00; P4M1 = 0x00;     //设置 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;     //设置 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;     //设置 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;     //设置 P7 口为准双向口模式

    P3PU = 0x0c;      //P32、P33 打开上拉电阻

    IT0 = 0;          //使能 INT0 上升沿和下降沿中断
//  IT0 = 1;          //使能 INT0 下降沿中断
    EX0 = 1;          //使能 INT0 中断

//  IT1 = 0;          //使能 INT1 上升沿和下降沿中断
    IT1 = 1;          //使能 INT1 下降沿中断
    EX1 = 1;          //使能 INT1 中断

    IE0 = 0;          //清 INT0 中断标志
    IE1 = 0;          //清 INT1 中断标志

    int0_flag = 0;    //初始化用户标志位
    int1_flag = 0;    //初始化用户标志位

    Timer0_Init();    //调用定时器 0 初始化函数
    Timer1_Init();    //调用定时器 1 初始化函数

    usb_init();    //调用 USB 初始化函数，不需要立即判断电脑已正确识别到 USB 从设备
    /* USB 型单片机从设备，如需要主动向电脑发送数据，
       在执行 USB_SendData( )函数和 printf_usb( )函数时，
       这两个函数已增加了判断电脑是否已正确识别到 USB 从设备的程序。
       如果电脑要主动发送数据给 USB 从设备，电脑自己会主动判断与 USB 从设备是否已正确连接。
    */

    EA = 1;           //总中断允许位打开
    P40 = 0;          //打开 LED 灯供电
    while(1)      //主循环中查询需要处理的各种事件，如 T0/T1 中断事件，并打印输出状态
    {    /*  本演示程序中，主循环查询各中断有无需要继续处理的事件的次序，
```

```
            依次是 INT0/INT1/T0/T1, 用户可以自己根据实际情况，
            调整查询各中断有无需要继续处理的事件的优先次序  */

        if(int0_flag)   //主循环中查询，INT0 是否已产生中断，是否有需要处理的 INT 0 事件
        {
            int0_flag = 0;      //清 0，INT0 事件位变量标志
            _nop_();            //用户在此添加需要处理的事件
            _nop_();
        }

        if(int1_flag)   //主循环中查询，INT1 是否已产生中断，是否有需要处理的 INT1 事件
        {
            int1_flag = 0;      //清 0，INT1 事件位变量标志
            _nop_();            //用户在此添加需要处理的事件
            _nop_();
        }

        if(t0_flag)     //主循环中查询，定时器 0 是否已产生中断，是否有需要处理的定时器 0 事件
        {
            t0_flag = 0;        //清 0，T0 事件位变量标志
            printf_usb("Timer0！\r\n");
            //向电脑 USB-CDC 串口助手输出“Timer0！”字符串，代表主循环在处理 T0 不急的任务
        }

        if(t1_flag)     //主循环中查询，定时器 1 是否已产生中断，是否有需要处理的定时器 1 事件
        {
            t1_flag = 0;        //清 0，T1 事件位变量标志
            printf_usb("Timer1！\r\n");
            //向电脑 USB-CDC 串口助手输出“Timer1！”字符串，代表主循环在处理 T1 不急的任务
        }
    }
}

void int0_isr(void) interrupt INT0_VECTOR
{
    int0_flag = 1;  // int0_flag 置 1 是通知主循环处理部分 INT0 中断事件不需要特急处理的任务
                    //置 1，记录 INT0 已产生中断，供主循环查询判断有无需处理的 INT0 任务

    if(INT0)                //边沿中断，进入后再次判断电平从而判断是什么样的电平
    {
    //  P01 = 0;            //判断为高电平，则当前为上升沿，点亮 P01 端口上的 LED 灯
    //  _nop_();            //可以在这里插入断点进行观察现象
    //  P01 = 1;            //关闭 LED 灯
        P01 = ~P01;         //如果用外部按键触发 INT0，则需另外添加去抖动处理
    }
    else
```

```
    {
    //  P06 = 0;            //判断为低电平，则当前为下降沿，点亮 P06 端口上的 LED 灯
    //  _nop_();            //可以在这里插入断点进行观察现象
    //  P06 = 1;            //关闭 LED 灯
        P06 = ~P06;         //如果用外部按键触发 INT0，则需另外添加去抖动处理
    }
}
//INT0 中断服务程序，INT0_VECTOR 在 AI8051U.H 头文件中已宏定义为 0

void int1_isr(void) interrupt INT1_VECTOR
{
    int1_flag = 1;  // int1_flag 置 1 是通知主循环处理部分 INT1 中断事件不需要特急处理的任务
                    //置 1，记录 INT1 已产生中断，供主循环查询判断有无需处理的 INT1 任务

//  P02= 0;             //点亮 P02 端口上的 LED 灯
//  _nop_();            //可以在这里插入断点观察现象
//  P02 = 1;            //关闭 LED 灯
    P02 = ~P02;         //如果用外部按键触发 INT1，则需另外添加去抖动处理

}
//INT1 中断服务程序，INT1_VECTOR 在 AI8051U.H 头文件中已宏定义为 2

void Timer0_Isr(void) interrupt TMR0_VECTOR      //定时器 0 中断服务程序
{
    P00 = ~P00;   //P00 灯闪烁，中断服务程序中尽量少执行长的任务，防止堵塞其他中断
                  //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
                  //但时间不要太长，否则会影响其他中断事件的实时响应速度
    t0_flag = 1;  // t0_flag 置 1 是通知主循环处理部分 T0 中断事件不需要特急处理的任务
                  //置 1，记录定时器 0 已产生中断，供主循环查询判断有无需处理的定时器 0 任务
}
//定时器 0 中断服务程序，TMR0_VECTOR 在 AI8051U.H 头文件中已宏定义为 1

void Timer1_Isr(void) interrupt TMR1_VECTOR
{
    P07 = ~P07;   //P07 灯闪烁，中断服务程序中尽量少执行长的任务，防止堵塞其他中断
                  //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
                  //但时间不要太长，否则会影响其他中断事件的实时响应速度
    t1_flag = 1;  // t1_flag 置 1 是通知主循环处理部分 T1 中断事件不需要特急处理的任务
                  //置 1，记录定时器 1 已产生中断，供主循环查询判断有无需处理的定时器 1 任务
}
//定时器 1 中断服务程序，TMR1_VECTOR 在 AI8051U.H 头文件中已宏定义为 3
```

17.9　大型综合实战--深大课程设计：时钟+数码管+矩阵按键+蜂鸣器发声

17.9.1 实验介绍

1、本项目主要熟悉如何管理多文件项目
2、熟悉矩阵式按键扫描方法
3、了解使用 74HC595 控制 LED 数码管
4、软件模拟 RTC，实现 RTC 的时、分、秒控制和设置
5、I/O 控制无源蜂鸣器发声，Key0 ~ Key7 按键按下，会发出对应的 Do/Re/Mi/Fa/So/La/Si/Do 音符
6、扩展部分：如何实现长按键

a) 按键扫描说明：

如按键要判断为正常按下，而不是抖动，需要该按键保持按下持续保持为低状态时间是 20ms ~ 50ms；主循环中有个 1ms 的时间基准，将按键扫描程序作为 1ms 事件，按键状态维持 20ms 不变，即可对按键去抖动，得到稳定的键码。

b) 实验现象和操作说明：

将本项目的 HEX 文件下载到 Ai8051U 的实验箱后

1、在 LED 数码管上面低 6 位会动态显示当前的 RTC 时间

2、按下两行 4 列的 8 个矩阵按键后：
- 在 LED 数码管的最高位会显示当前的按键键码
- 同时会根据不同的按键发出不同按键音

（特别注意：需要将实验箱上“**跳线 3：蜂鸣器 / 原理图标号是 J1C**”的跳线连上，蜂鸣器才会发声）

3、矩阵按键中的按键 0，是 RTC 模式设置键
- 上电后 RTC 为正常工作模式，此时数码管动态显示 RTC 时间
- 按一次按键 0，RTC 会切换为小时设置模式，此时 RTC 的小时值会 0.5 秒闪烁一次
- 再按一次按键 0，RTC 会切换为分钟设置模式，此时 RTC 的分钟值会 0.5 秒闪烁一次
- 再按一次按键 0，RTC 会切换为秒钟设置模式，此时 RTC 的秒钟值会 0.5 秒闪烁一次
- 在按一次按键 0，恢复为 RTC 正常工作模式

4、矩阵按键中的按键 4 和 5 分别是上调键和下调键
- 只有在 RTC 的时/分/秒设置模式时，上调键和下调键才有效

5、时钟设置模式时，长按按键 4 和 5，可实现快速设置功能

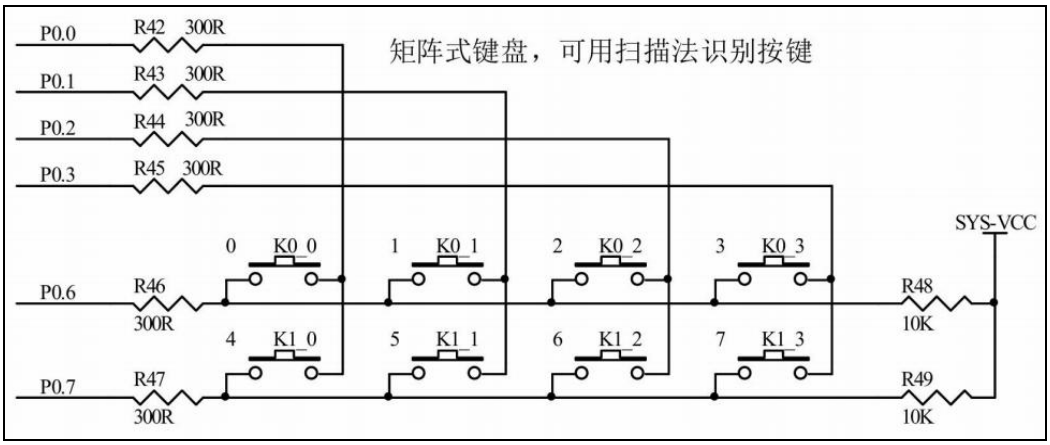
c) Keil 环境下多文件项目管理说明

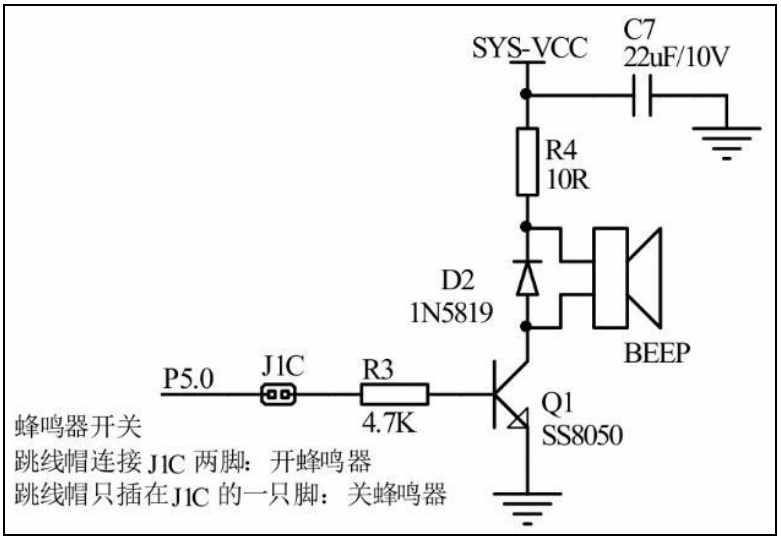
当项目的功能比较复杂时，就需要在 Keil 中建立多文件项目，以方便分工合作、代码复用、模块化管理、增强可读性和可维护性。

- 比较好的建议是将项目功能模块化，不同模块的实现代码放在不同的.c 文件中。
- 一般建议是一个模块对应一个.c 程序文件和一个.h 头文件
- 模块的初始化函数以及相关的数据处理函数都在.c 文件中实现
- 与模块相关的全局变量也必须在.c 文件中进行定义，一定不能在.h 文件中定义变量
- 如果有其他模块需要使用本模块定义的变量或函数，则这些函数和变量都需要在.h 文件中声明。
- 特别提醒：在.h 文件中声明外部变量必须使用 **extern** 关键字，否则就变成变量定义了，这样会出现变量重复定义的错误
- 为防止头文件被多次包含而产生错误或者警告，在头文件中使用类似如下的条件编译组合语句，可避免在同一个.c 文件中对同一个.h 头文件进行多次包含：

```
#ifndef XXXX
#define XXXX
...
#endif
```

17.9.2 原理图



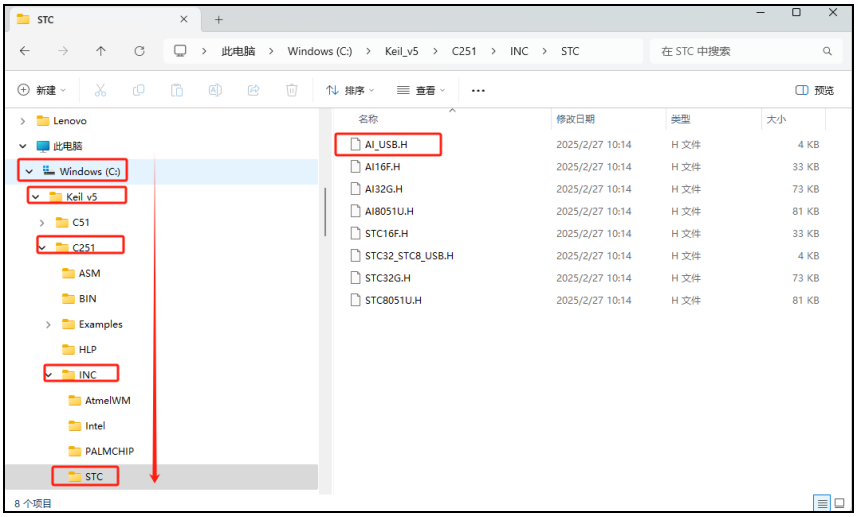


17.9.3 准备工作--添加头文件和库函数

a) 添加常用头文件 ai_usb.h

#include "ai_usb.h"　　//使用 usb 所需要包含的头文件。

这个头文件在使用 AIapp-ISP 软件添加型号和头文件到 Keil 中时就已经一并被添加到 Keil 中了。

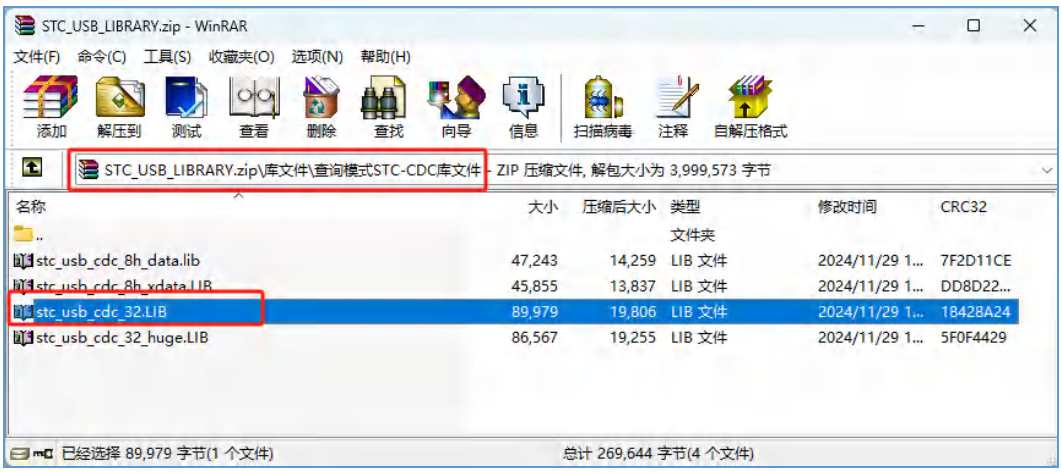


b) 添加 usb_cdc 库函数文件

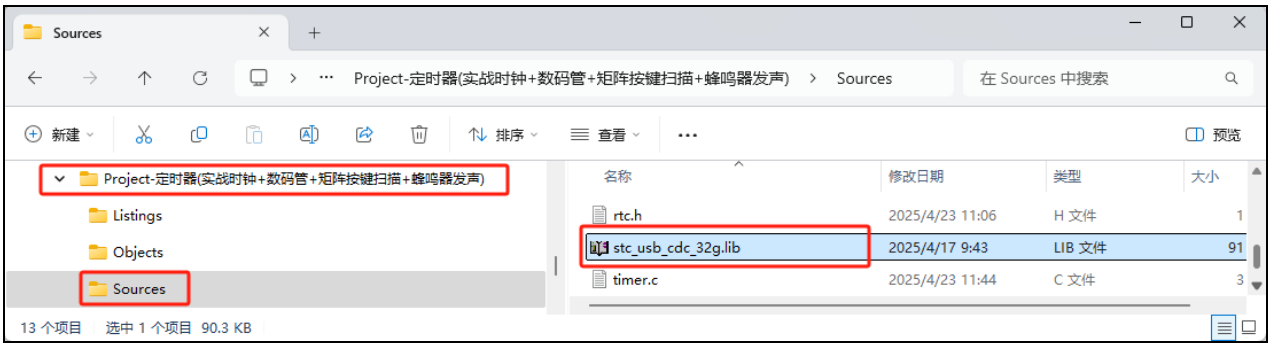
这部分可以到 STCAI 的官网：https://www.stcai.com/khs 的 USB 库文件中进行下载

下载完成后，打开压缩包“STC_USB_LIBRARY.zip”，

在’库文件\查询模式 STC-CDC 库文件’中，复制出来 stc_usb_cdc_32.LIB

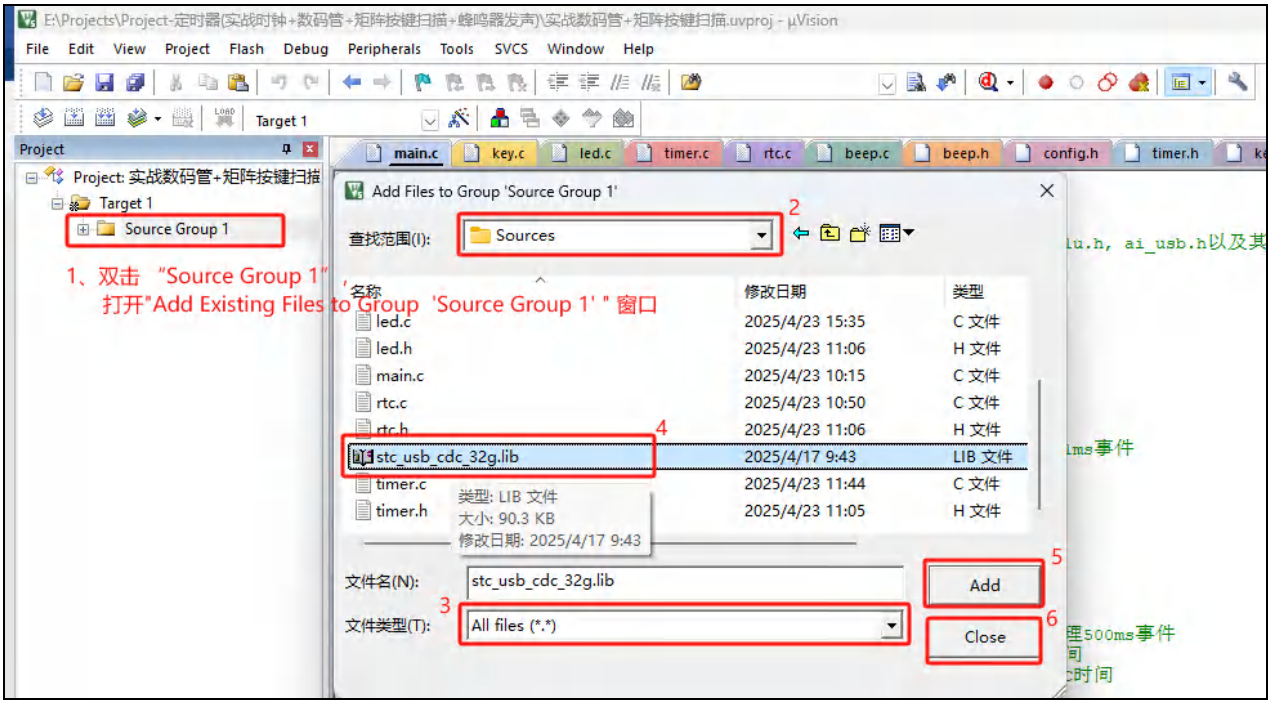


将库函数“stc_usb_cdc_32.LIB”复制到工程所在的文件夹 Project-定时器(实战时钟+数码管+矩阵按键扫描+蜂鸣器发声)>Sources 目录下：

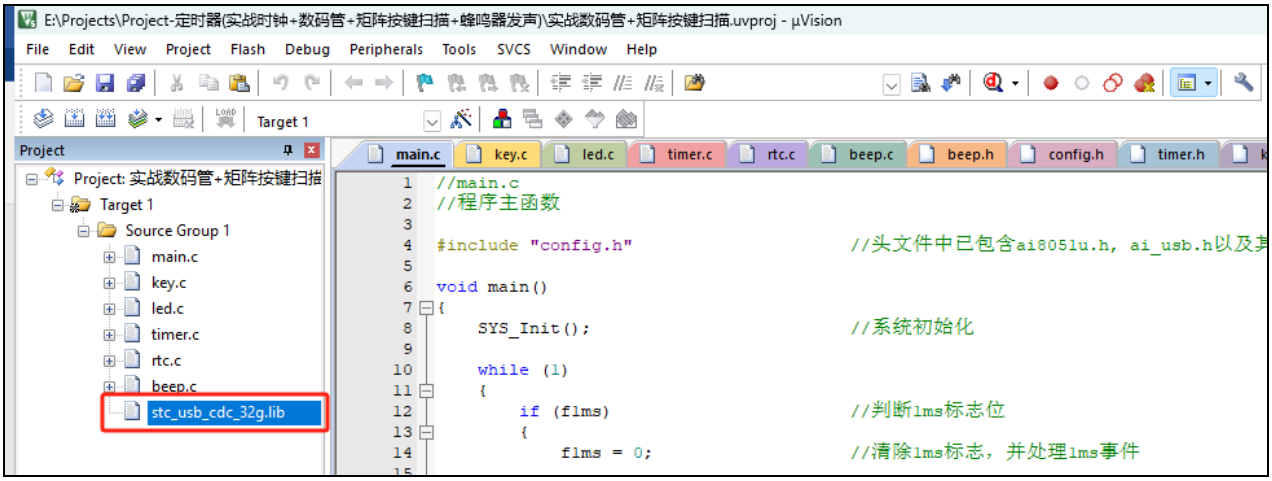


将库函数 stc_usb_cdc_32.LIB 添加进入此工程项目中

在 Keil 中，双击 Source Group，在弹出的 Add Files to Group 'Source Group 1 窗口中，将库函数 stc_usb_cdc_32.LIB 添加进入此工程，就可以使用 USB-CDC 功能。

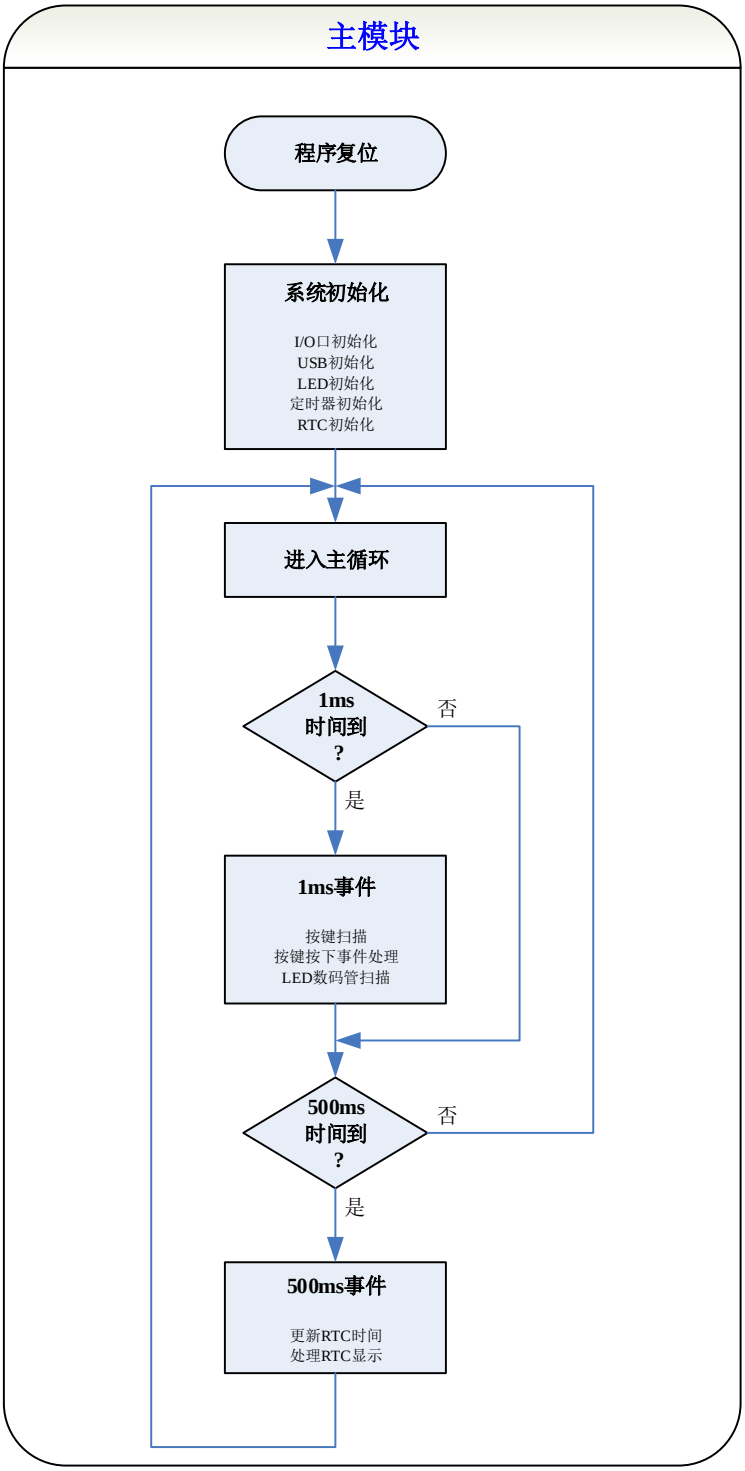


这样，Keil 界面变化如下图：

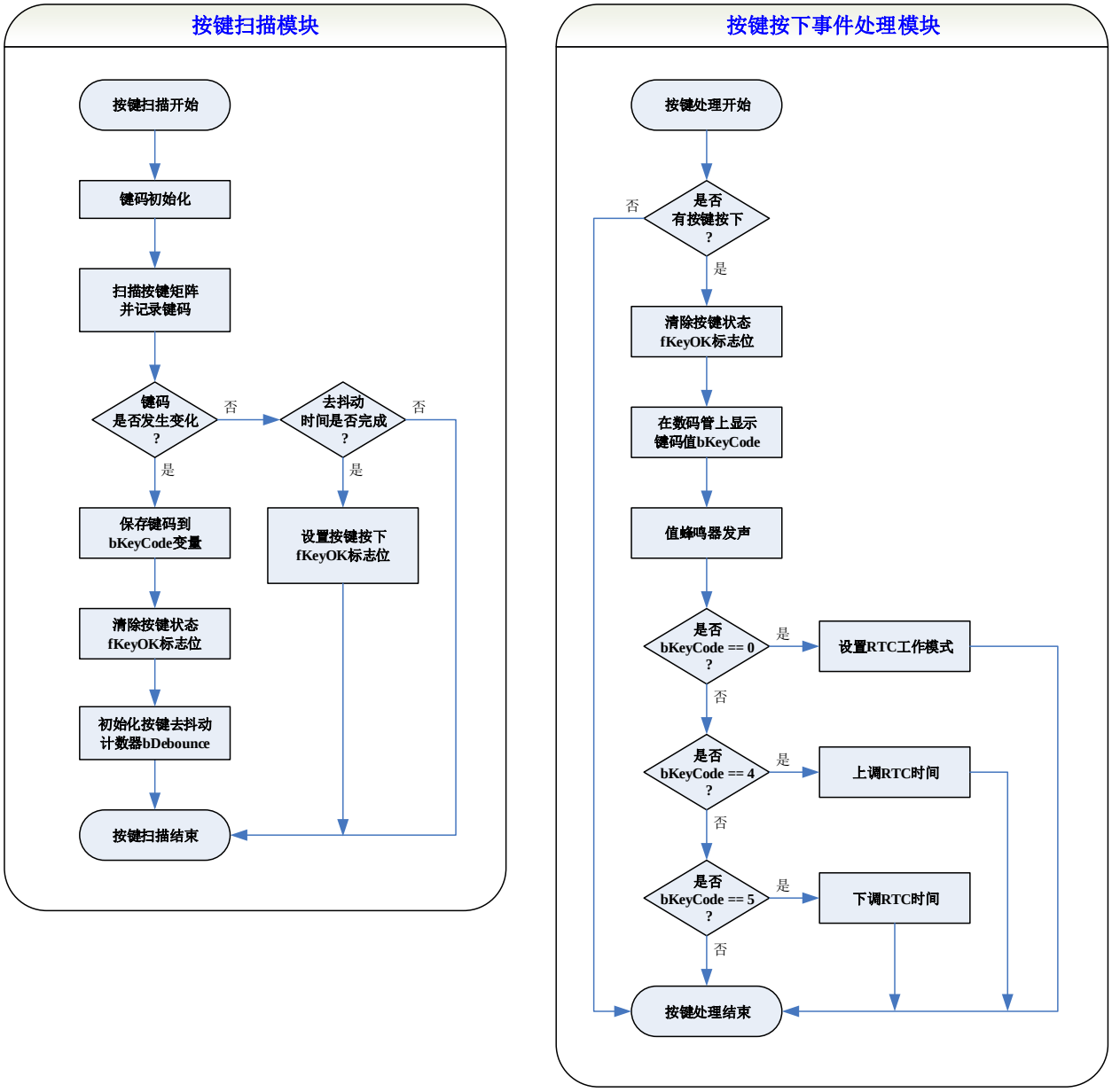


17.9.4 流程图

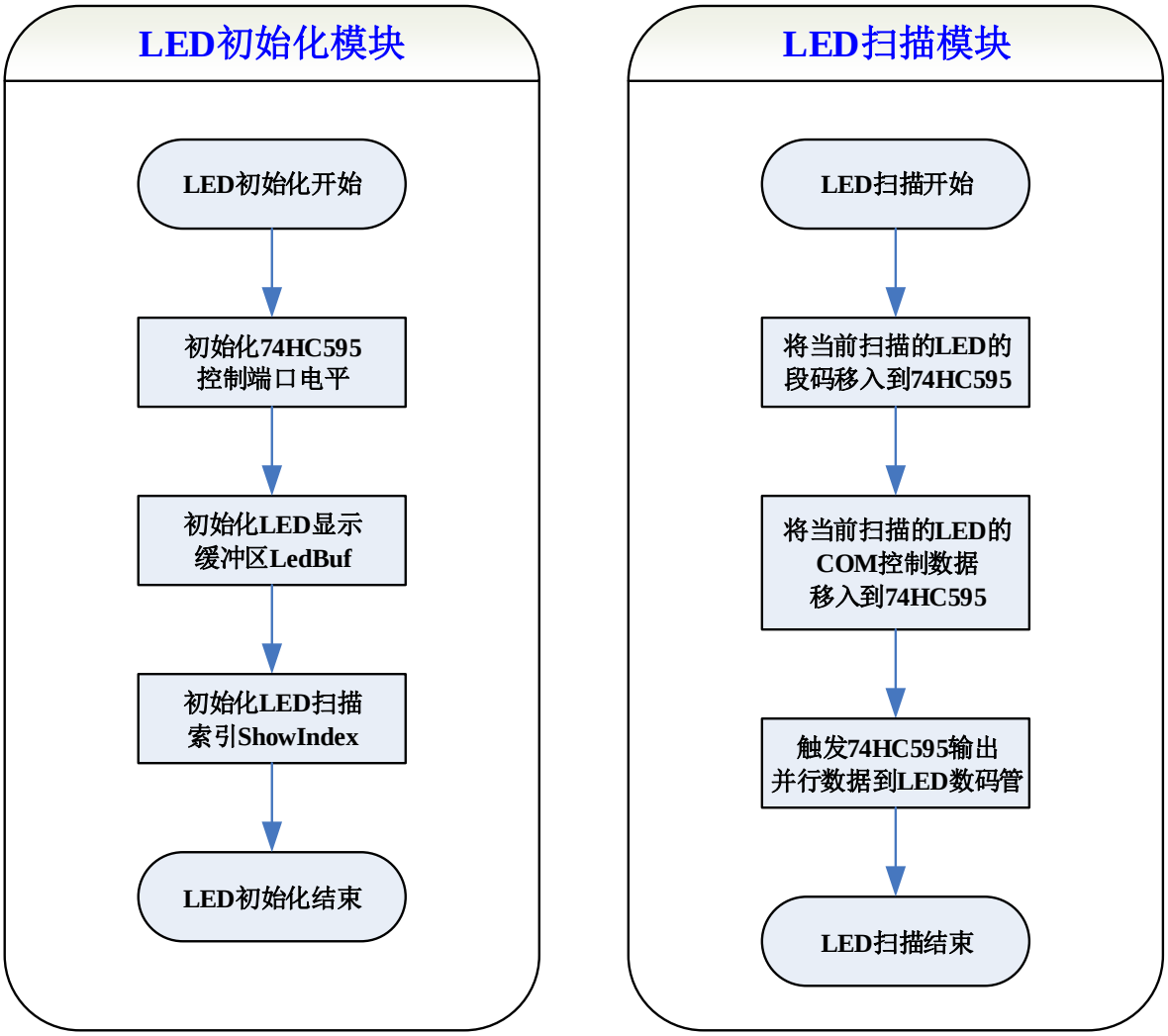
a) main -- 程序主函数 流程图



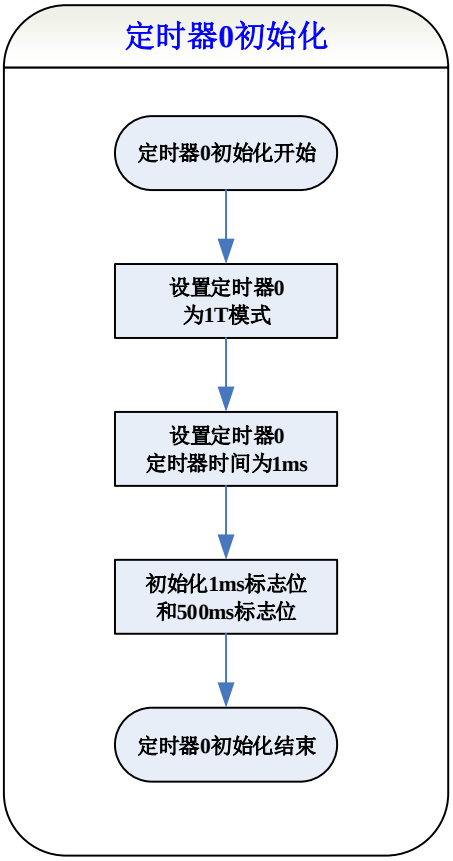
b) key -- 按键扫描以及按键事件处理 流程图

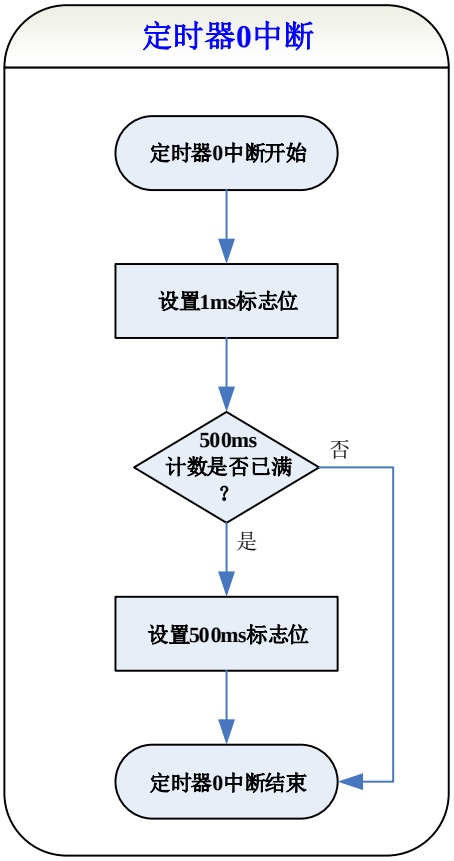


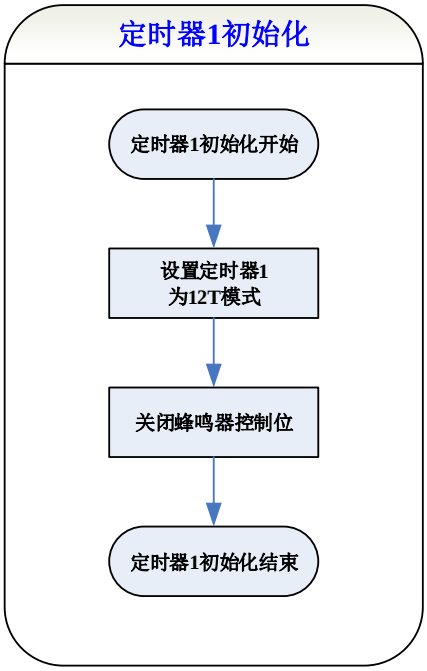
c) led -- 扫描 LED 数码管 流程图

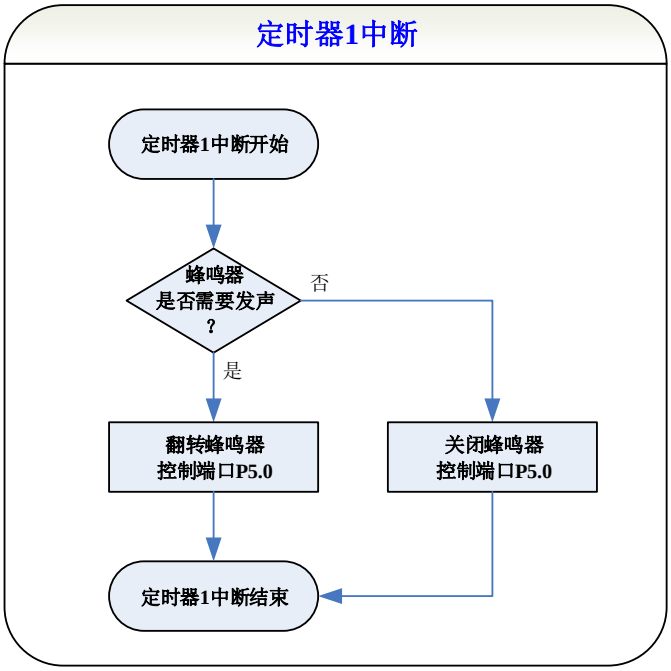


d) timer -- 定时器初始化以及定时器中断服务程序 流程图

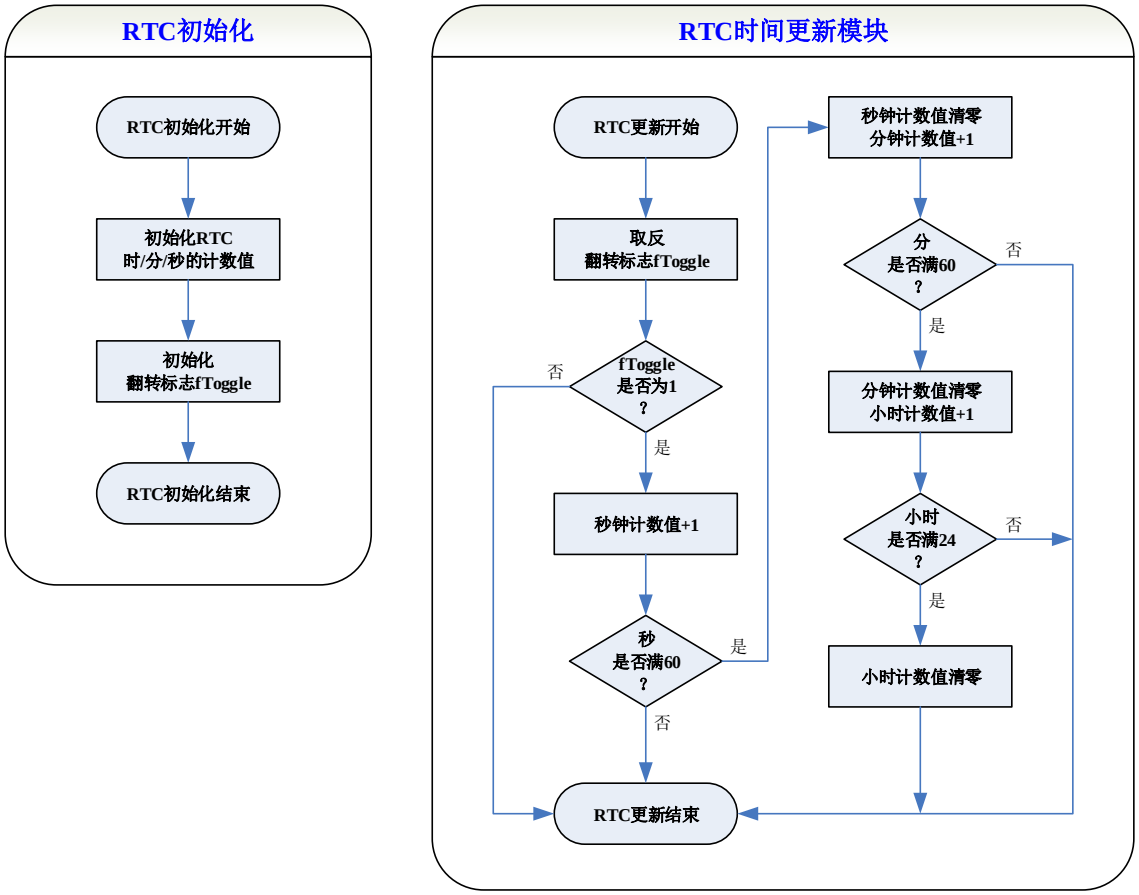


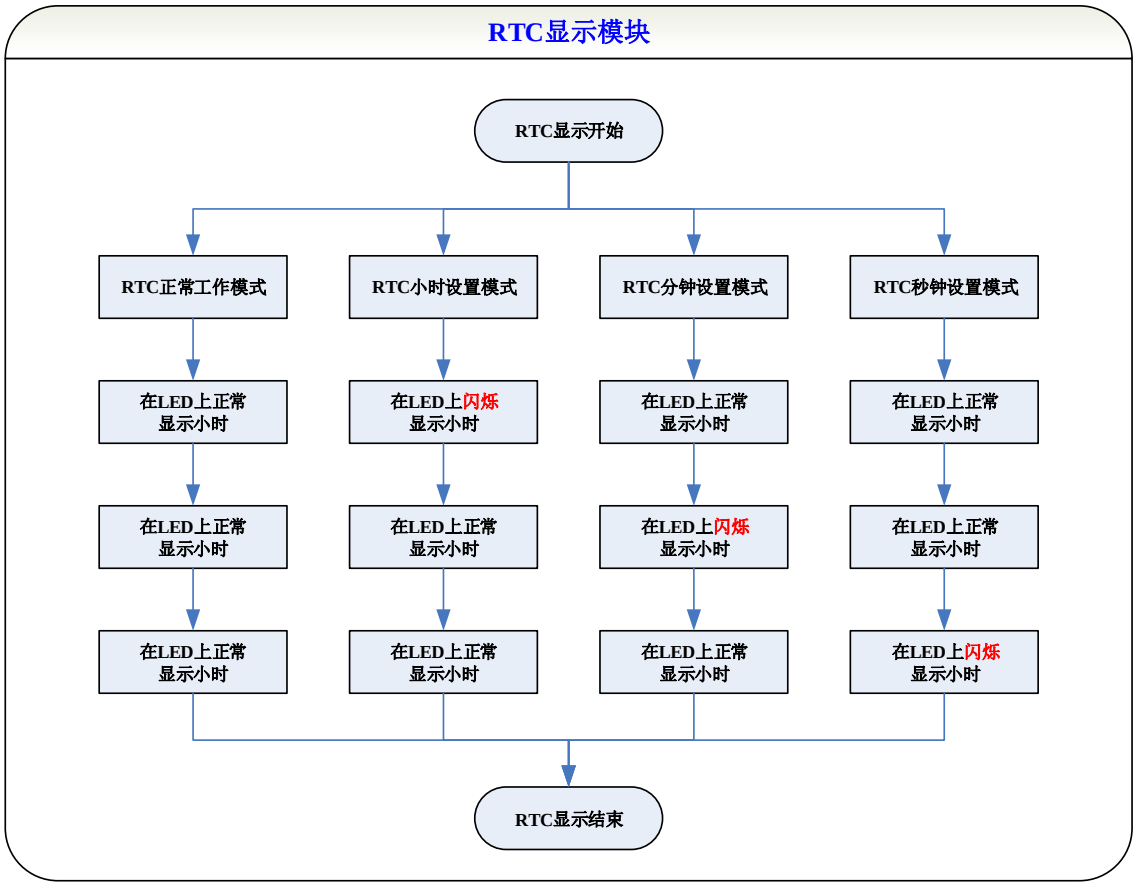




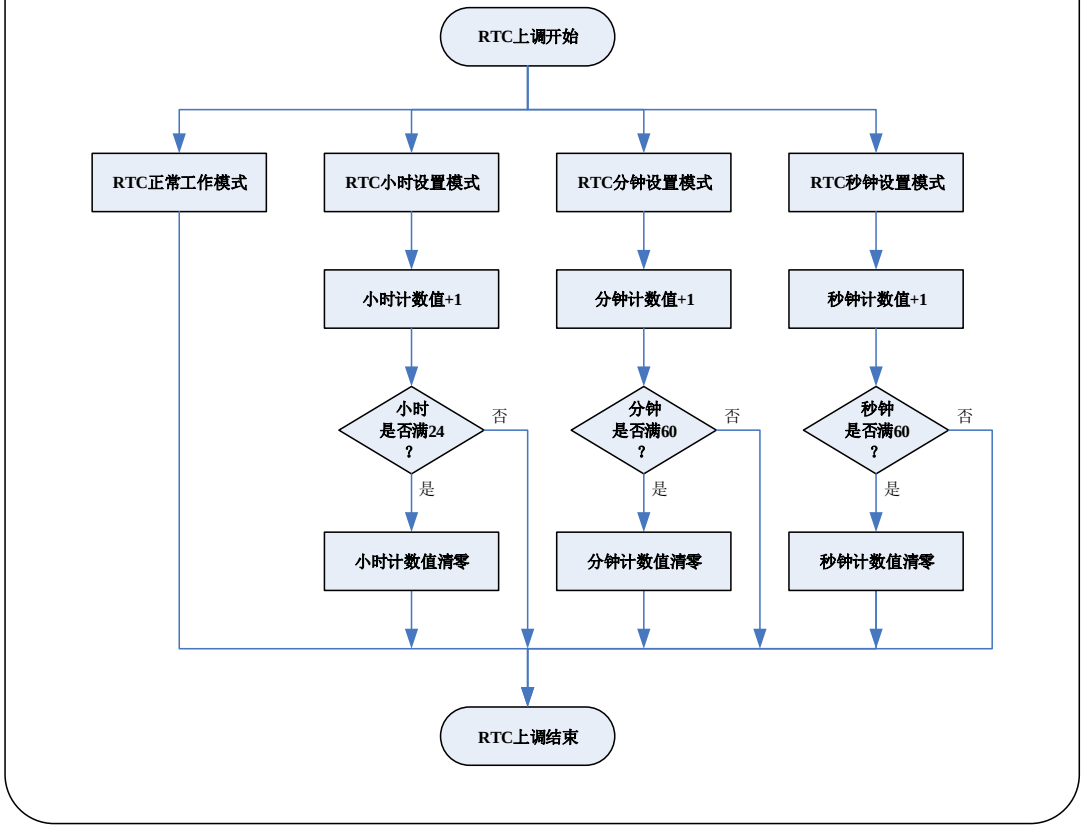


e) rtc -- 软件模拟 RTC 程序 流程图

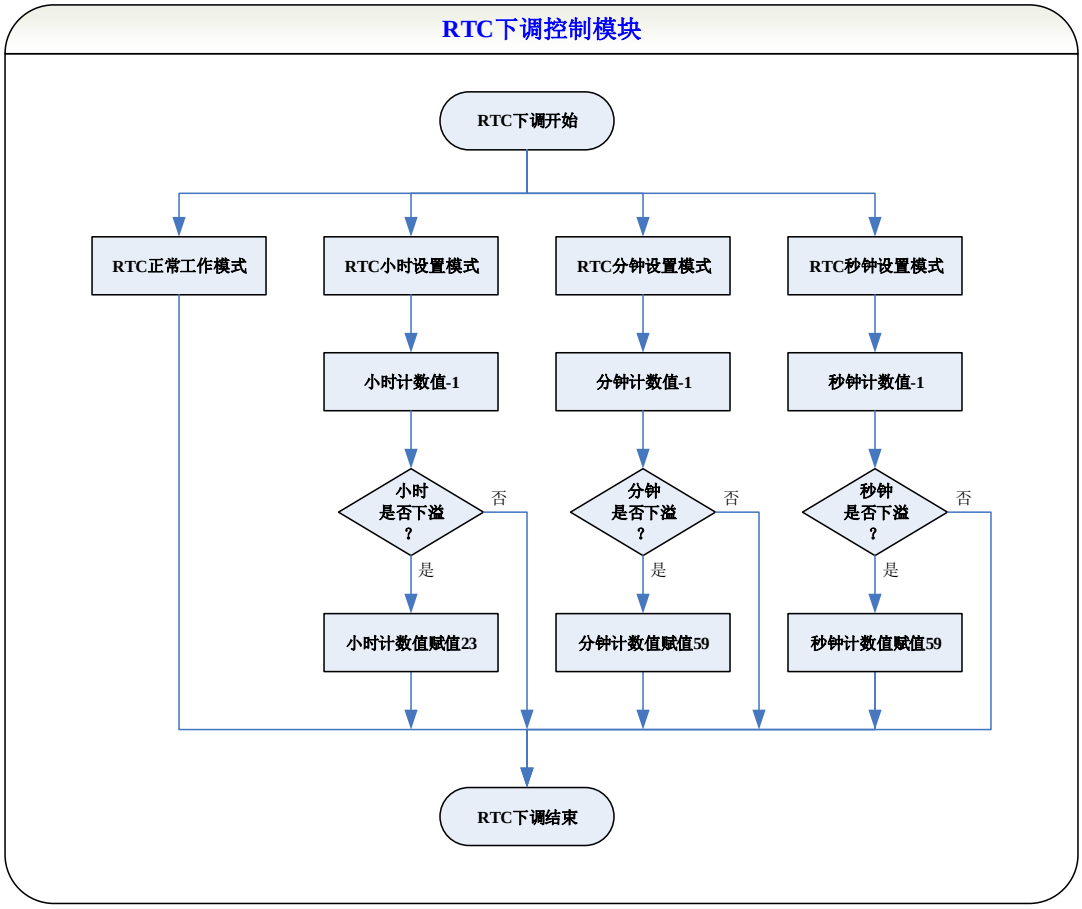




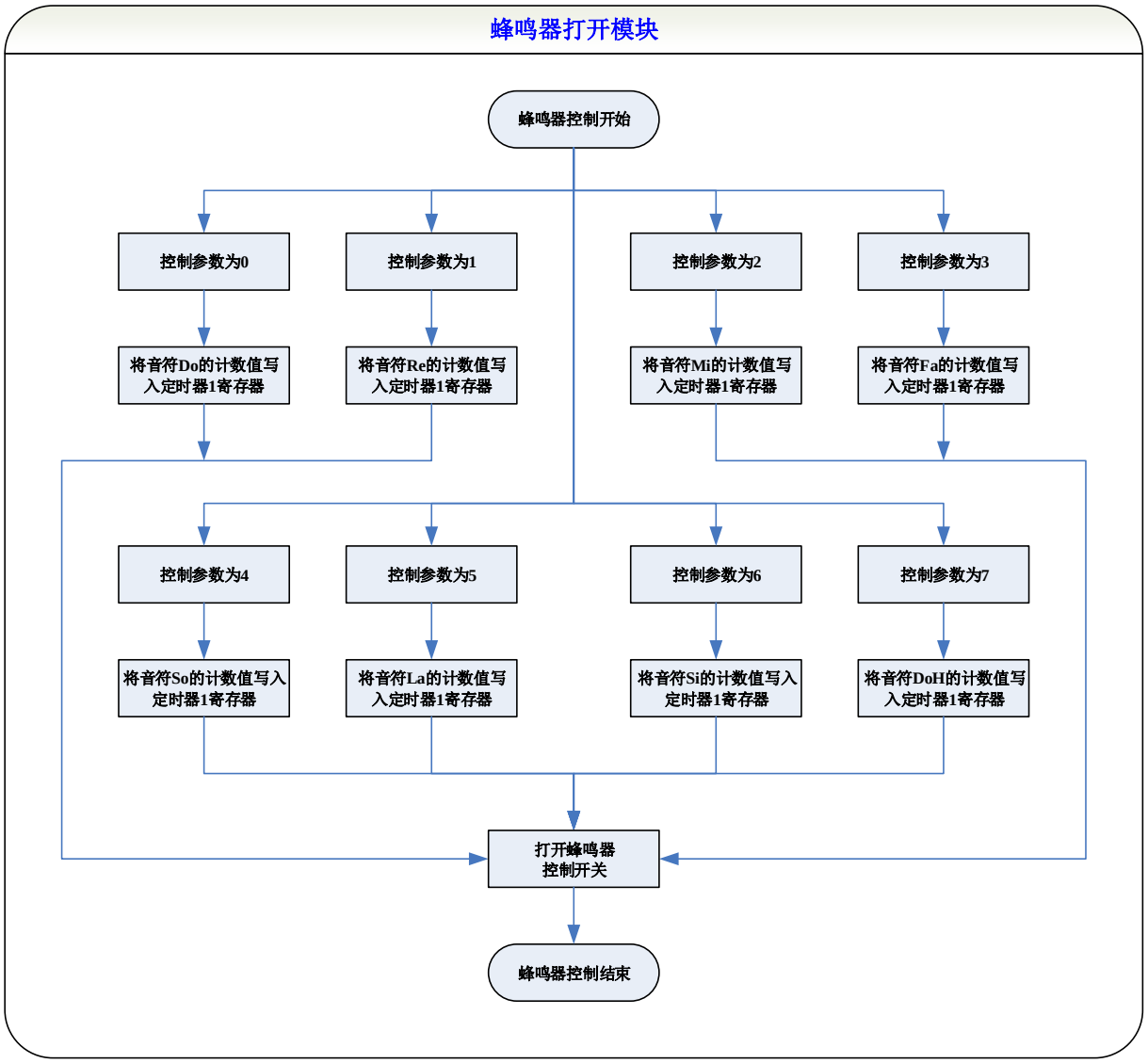
RTC上调控制模块

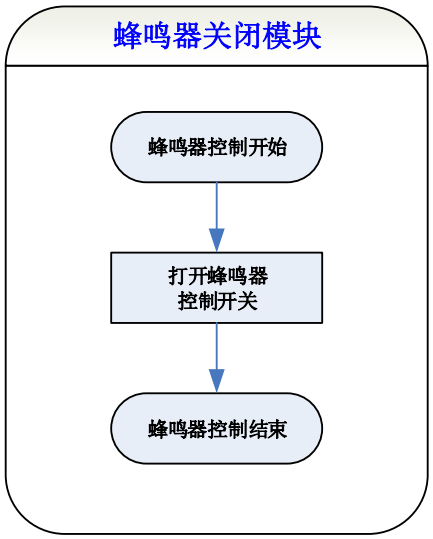


RTC下调控制模块



f)　beep -- 蜂鸣器发声控制程序 流程图





17.9.5 实验程序代码

a) main.c -- 程序主函数

```
//main.c
//程序主函数
#include "config.h"             //头文件中已包含 ai8051u.h, ai_usb.h 以及其他头文件

void main()
{
    SYS_Init();                 //系统初始化
    while (1)
    {
        if (f1ms)               //判断 1ms 标志位
        {
            f1ms = 0;           //清除 1ms 标志，并处理 1ms 事件
            ScanKey();          //按键扫描
            KeyEvents();        //按键处理事件
            ScanLed();          //LED 数码管扫描
        }
        if (f500ms)             //判断 500ms 标志位
        {
            f500ms = 0;         //清除 500ms 标志，并处理 500ms 事件
            RTC_Update();       //更新软件模拟的 RTC 时间
            RTC_Show();         //在 LED 数模管上显示 RTC 时间
        }
    }
}

void SYS_Init(void)             //系统初始化函数
{
    EAXFR = 1;                  //使能访问扩展 XFR
    WTST = 0x00;                //设置最快速度访问程序代码
    CKCON = 0x00;               //设置最快速度访问内部 XDATA

    P0M0 = 0x00; P0M1 = 0x00;   //初始化 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x00;   //初始化 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;   //初始化 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x00;   //初始化 P3 口为准双向口模式
    P4M0 = 0x00; P4M1 = 0x00;   //初始化 P4 口为准双向口模式
    P5M0 = 0x01; P5M1 = 0x00;   //设置 P5.0 口为强推挽，其他设置为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;   //初始化 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;   //初始化 P7 口为准双向口模式

    usb_init();                 //USB 库函数初始化
```

```
    Led_Init();                 //LED 数码管初始化
    Timer0_Init();              //定时器 0 初始化,用于产生 1ms 和 500ms 的时间基准
    Timer1_Init();              //定时器 1 初始化,用于驱动无源蜂鸣器,使蜂鸣器发声
    RTC_Init();                 //软件模拟 RTC 时间初始化

    EA = 1;                     //总中断允许位打开
}
```

b) key.c -- 按键扫描以及按键事件处理

```
//key.c
//按键扫描以及按键事件处理
#include "config.h"             //头文件中已包含 ai8051u.h, ai_usb.h 以及其他头文件

bit fKeyOK;                     //按键按下标志
bit fKeyHold;                   //按键长按标志,长按 2 秒后有效,用于快速设置时间
BYTE bKeyCode;                  //定义变量,保存按键键码
BYTE bDebounce;                 //按键去抖动
WORD wHoldCount;                //按键长按计数器

void ScanKey()                  //按键扫描程序  (需每 1ms 执行一次!!!)
{
    BYTE key;                   //按键扫描键码

    key = 0xff;                 //初始化扫描变量
    P0 = 0xff;                  //P0 输出输出高使能内部若上拉,准备读取按键状态
    P06 = 0; P07 = 1;           //准备扫描第 1 行(P0.6)
    _nop_();                    //等待 2 个 nop 再读取按键状态
    _nop_();
    if (!P00) key = 0;          //检查第 1 行第 1 列按键是否被按下
    else if (!P01) key = 1;     //检查第 1 行第 2 列按键是否被按下
    else if (!P02) key = 2;     //检查第 1 行第 3 列按键是否被按下
    else if (!P03) key = 3;     //检查第 1 行第 4 列按键是否被按下
    else                        //如果第一行没有按键按下则扫描下一行
    {
        P07 = 0; P06 = 1;       //准备扫描第 2 行(P0.7)
        _nop_();                //等待 2 个 nop 再读取按键状态
        _nop_();
        if (!P00) key = 4;      //检查第 2 行第 1 列按键是否被按下
        else if (!P01) key = 5; //检查第 2 行第 2 列按键是否被按下
        else if (!P02) key = 6; //检查第 2 行第 3 列按键是否被按下
        else if (!P03) key = 7; //检查第 2 行第 4 列按键是否被按下
    }
    P0 = 0xff;                  //恢复 P0 口状态
```

```
    if (key != bKeyCode)            //判断按键状态是否有变化
    {
        fKeyOK = 0;                 //清除按键按下标志
        fKeyHold = 0;               //清除按键长按标志
        bKeyCode = key;             //保存键码
        bDebounce = 20;             //初始化去抖动变量(去抖动时间设置为 20ms)
        wHoldCount = 2000;          //初始化按键长按计数器(按键按住不放维持时间超过 2000ms,
                                    //按键长按状态有效)
    }
    else
    {
        if (bDebounce != 0)         //判断按键扫描是否完成
        {
            if (--bDebounce == 0)   //判断去抖动是否已经完成
            {
                fKeyOK = 1;         //设置按键扫描完成标志
            }
        }

        if (--wHoldCount == 0)      //判断长按是否完成
        {
            wHoldCount = 100;       //长按有效后,每 100ms 自动触发一次按键处理
            fKeyHold = 1;           //设置按键长按标志位
            fKeyOK = 1;             //设置按键扫描完成标志
        }
    }
}

void KeyEvents()                    //按键事件处理程序  (需每 1ms 执行一次!!!)
{
    if (fKeyOK)                     //判断按键按下标志
    {
        fKeyOK = 0;                 //清除标志

        SetLed(0, bKeyCode, 0);     //在 LED 数码管的第 0 位显示按键键码
        BeepOn(bKeyCode);           //根据不同的按键播放按键音

        if (fKeyHold)               //判断是否按键长按有效
        {
            fKeyHold = 0;           //清除按键长按标志位
            switch (bKeyCode)       //处理不同的按键键码
            {
            case 4:
                RTC_SetUp();        //长按按键 4,快速向上调整 RTC 时间
                break;
            case 5:
```

```
                RTC_SetDown();  //长按按键 5,快速向下调整 RTC 时间
                break;
            }
        }
        else                    //否则为普通按键按下
        {
            switch (bKeyCode)   //处理不同的按键键码
            {
            case 0:
                RTC_SetMode();  //按下按键 0,设置 RTC 工作模式
                break;
            case 4:
                RTC_SetUp();    //按下按键 4,向上调整 RTC 时间
                break;
            case 5:
                RTC_SetDown();  //按下按键 5,向上调整 RTC 时间
                break;
            }
        }
    }
}
```

c) led.c -- 扫描 LED 数码管

```
//led.c
//使用 74HC595 动态扫描 LED 数码管
//由于 LED 数码管是由 8 个 SEG 和 8 个 COM 组成的矩阵 LED
//所以 LED 数码管也需要扫描显示

#include "config.h"             //头文件中已包含 ai8051u.h, ai_usb.h 以及其他头文件

sbit  DAT = P3^4;               //74HC595 串行数据输入脚
sbit  SCK = P3^2;               //74HC595 串行数据移入时钟脚,上升沿有效
sbit  RCK = P3^5;               //74HC595 并行数据输出时钟脚,上升沿有效

#define LED_OFF         0x00    //定义关闭 LED 数码管的段码数据,LED 数码管为共阴极,
                                //所以 SEG 线全部输出 0 即可关闭一个 8 段数码管
#define LED_POINT       0x80    //定义 LED 数码管小数点的段码数据

BYTE ShowIndex;                 //LED 数码管扫描显示索引(8 个数码管轮流扫描显示)
BYTE LedBuf[8];                 //8 个 LED 数码管显示数据缓冲区

BYTE code LED_PATTERN[10] =     //定义 LED 数码管显示数据 0~9 的段码数据
{
    0x3F,   //0                 //此表为共阴极数码管的段码数据
```

```
    0x06,    //1
    0x5B,    //2
    0x4F,    //3
    0x66,    //4
    0x6D,    //5
    0x7D,    //6
    0x07,    //7
    0x7F,    //8
    0x6F,    //9
};

void Shift_74HC595(BYTE dat)        //串行移入一个字节的数据到 74HC595
{
    BYTE i;

    for (i = 0; i < 8; i++)         //移动次数控制(移动 8Bit)
    {
        dat <<= 1;                  //将数据的最高位移出到 CY
        DAT = CY;                   //将移出的最高位数据输出到 74HC595 的数据输入口
        SCK = 0;                    //74HC595 的串行数据移入时钟脚输出低电平
        _nop_();                    //等待 2 个时钟
        _nop_();
        SCK = 1;                    //74HC595 的串行数据移入时钟脚输出高电平,上升沿移入数据
    }
}

void Output_74HC595(void)           //74HC595 并行数据输出控制
{
    RCK = 0;                        //74HC595 的并行数据输出时钟脚输出低电平
    _nop_();                        //等待 2 个时钟
    _nop_();
    RCK = 1;                        //74HC595 的并行数据输出时钟脚输出高电平,上升沿输出数据
}

void Led_Init(void)                 //LED 数码管初始化程序
{
    DAT = 0;                        //初始化 74HC595 时钟线和数据线
    RCK = 0;
    SCK = 0;

    LedBuf[0] = LED_OFF;            //初始化 LED 数码管显示数据缓冲区,上电关闭显示
    LedBuf[1] = LED_OFF;
    LedBuf[2] = LED_OFF;
    LedBuf[3] = LED_OFF;
    LedBuf[4] = LED_OFF;
    LedBuf[5] = LED_OFF;
```

```
    LedBuf[6] = LED_OFF;
    LedBuf[7] = LED_OFF;

    ShowIndex = 0;                  //初始化扫描显示索引
}

void ScanLed(void)                  //LED 数码管扫描显示程序,  (需每 1ms 执行一次!!!)
{
    Shift_74HC595(LedBuf[ShowIndex]); //将当前扫描到的 LED 数码管的段码移入到 74HC595
    Shift_74HC595(~(1 << ShowIndex));  //将当前扫描到的 LED 数码管的 COM 数据移入到 74HC595
    Output_74HC595();               //将 74HC595 中的段码数据和 COM 数据输出到 LED 数码管的管脚

    if (++ShowIndex >= 8)           //扫描索引值递增
    ShowIndex = 0;                  //扫描完一轮后索引值归零
}

void SetLed    (BYTE nIndex,        //设置 LED 显示函数,nIndex:设置目标数码管,0~7 分别对应 8 个数码
                BYTE nNum,          //nNum:设置显示的数据,有效值 0~9,其他值则关闭显示
                BOOL bShowPoint     //bShowPoint:设置是否显示小数点
                )
{
    if (nIndex < 8)                 //目标数码管的有效值为 0~7
    {
        if (nNum < 10)              //判断显示的值是否为 0~9
            LedBuf[nIndex] = LED_PATTERN[nNum];  //将相应数据的段码赋值到 LED 数码管显示数据缓冲区
        else
            LedBuf[nIndex] = LED_OFF;           //如果是其他值则关闭显示

        if (bShowPoint)                         //判断是否需要显示小数点
            LedBuf[nIndex] |= LED_POINT;        //如果需要显示,则增加小数点的段码数据
    }
}
```

d) timer.c -- 定时器初始化以及定时器中断服务程序

```
//timer.c
//定时器初始化以及定时器中断服务程序
//定时器 0:1ms 定时，产生 1ms 和 500ms 的标志位
//定时器 1:用于翻转蜂鸣器的控制口 P5.0
//动态改变定时器 1 定时时长可控制蜂鸣器发出不同的声音

#include "config.h"             //头文件中已包含 ai8051u.h, ai_usb.h 以及其他头文件

bit f1ms;                       //1ms 标志位
bit f500ms;                     //500ms 标志位
```

```
int cnt500;                     //500 次计数器,用于产生 500ms 的时间

void Timer0_Init(void)          //1 毫秒@40.000MHz
{
    AUXR |= 0x80;               //定时器 0 时钟 1T 模式
    TMOD &= 0xF0;               //设置定时器 0 为模式 0
    TL0 = 0xC0;                 //设置定时器 0 初始值
    TH0 = 0x63;                 //设置定时器 0 初始值
    TF0 = 0;                    //清除 TF0 标志
    TR0 = 1;                    //定时器 0 开始计时
    ET0 = 1;                    //使能定时器 0 中断

    cnt500 = 0;                 //初始化 500 次计数器
    f1ms = 0;                   //初始化 1m 标志位
    f500ms = 0;                 //初始化 500ms 标志位
}

void Timer1_Init(void)          //@40.000MHz
{
    AUXR &= ~0x40;              //定时器 1 时钟 12T 模式
    TMOD &= 0x0F;               //设置定时器 1 为模式 0
    TL1 = 0x00;                 //设置定时器 1 初始值
    TH1 = 0x00;                 //设置定时器 1 初始值
    TF1 = 0;                    //清除 TF1 标志
    TR1 = 1;                    //定时器 1 开始计时
    ET1 = 1;                    //使能定时器 1 中断

    fBeepEn = 0;                //初始化蜂鸣器发声使能位
}

void Timer0_Isr(void) interrupt 1  //定时器 0 中断服务程序
{
    f1ms = 1;                   //设置 1ms 标志位

    if (cnt500++ >= 500)        //判断 500 此计数是否已满
    {
        cnt500 = 0;             //清除计数值,开始开始计数
        f500ms = 1;             //设置 500ms 标志位
    }
}

void Timer1_Isr(void) interrupt 3  //定时器 1 中断的服务程序
{
    if (fBeepEn)                //判断是否使能蜂鸣器发声
        P50 = !P50;             //如果需要则翻转蜂鸣器控制端口 P5.0
    else
```

```
    P50 = 0;                    //如果不需要,则关闭蜂鸣器控制端口 P5.0
}
```

e) rtc.c -- 软件模拟 RTC 程序

```
//rtc.c
//软件模拟 RTC 程序
//包括 RTC 时间更新控制
//RTC 显示控制以及 RTC 时间调整控制

#include "config.h"             //头文件中已包含 ai8051u.h, ai_usb.h 以及其他头文件

bit fToggle;                    //0.5 秒->1 秒的控制位(两次计数器),用于在设置时间时进行闪烁控制
BYTE bHour;                     //RTC 小时计数值
BYTE bMinute;                   //RTC 分钟计数值
BYTE bSecond;                   //RTC 秒钟计数值

BYTE bMode;                     //RTC 工作模式控制
                                //0:正常工作模式
                                //1:小时设置模式
                                //2:分钟设置模式
                                //3:秒钟设置模式

void RTC_Init()                 //RTC 初始化函数
{
    bHour = 12;                 //初始化时间为 12:30:00
    bMinute = 30;
    bSecond = 0;
    bMode = 0;                  //初始化 RTC 工作模式为正常工作模式
    fToggle = 0;                //初始化翻转控制
}

void RTC_Update()               //RTC 时间更新函数,  (需每 500ms 执行一次!!!)
{
    fToggle = !fToggle;         //翻转控制位

    if ((fToggle == 1) &&       //如果 1 秒时间到并且为正常工作模式,则更新 RTC 时间
       (bMode == 0))
    {
        if (++bSecond >= 60)        //秒钟递增 1,并判断秒钟是否记满 60
        {
            bSecond = 0;            //秒钟归零
            if (++bMinute >= 60)    //分钟递增 1,并判断分钟是否记满 60
            {
                bMinute = 0;        //分钟归零
```

```
                if (++bHour >= 24)      //小时递增 1,并判断小时是否记满 24
                {
                    bHour = 0;          //小时归零
                }
            }
        }
    }
}

void RTC_Show()                         //RTC 显示控制程序,不同的模式显示方式不一样
{
    switch (bMode)                      //对 RTC 工作模式进行分支处理
    {
    case 0:                                         //正常工作模式
        SetLed(2, (BYTE)(bHour / 10), 0);           //在数码 2 上显示小时的十位
        SetLed(3, (BYTE)(bHour % 10), 1);           //在数码 3 上显示小时的个位
        SetLed(4, (BYTE)(bMinute / 10), 0);         //在数码 4 上显示分钟的十位
        SetLed(5, (BYTE)(bMinute % 10), 1);         //在数码 5 上显示分钟的个位
        SetLed(6, (BYTE)(bSecond / 10), 0);         //在数码 6 上显示秒钟的十位
        SetLed(7, (BYTE)(bSecond % 10), 0);         //在数码 7 上显示秒钟的个位
        break;
    case 1:                                         //设置 RTC 的小时模式
        if (fToggle)                                //闪烁控制
        {
            SetLed(2, (BYTE)(bHour / 10), 0);       //在数码 2 上显示小时的十位
            SetLed(3, (BYTE)(bHour % 10), 1);       //在数码 3 上显示小时的个位
        }
        else
        {
            SetLed(2, -1, 0);                       //在数码 2 上关闭显示,产生闪烁效果
            SetLed(3, -1, 1);                       //在数码 3 上关闭显示
        }
        SetLed(4, (BYTE)(bMinute / 10), 0);         //在数码 4 上显示分钟的十位
        SetLed(5, (BYTE)(bMinute % 10), 1);         //在数码 5 上显示分钟的个位
        SetLed(6, (BYTE)(bSecond / 10), 0);         //在数码 6 上显示秒钟的十位
        SetLed(7, (BYTE)(bSecond % 10), 0);         //在数码 7 上显示秒钟的个位
        break;
    case 2:                                         //设置 RTC 的分钟模式
        SetLed(2, (BYTE)(bHour / 10), 0);           //在数码 2 上显示小时的十位
        SetLed(3, (BYTE)(bHour % 10), 1);           //在数码 3 上显示小时的个位
        if (fToggle)                                //闪烁控制
        {
            SetLed(4, (BYTE)(bMinute / 10), 0);     //在数码 4 上显示分钟的十位
            SetLed(5, (BYTE)(bMinute % 10), 1);     //在数码 5 上显示分钟的个位
        }
        else
```

```
        {
            SetLed(4, -1, 0);                       //在数码 4 上关闭显示,产生闪烁效果
            SetLed(5, -1, 1);                       //在数码 5 上关闭显示
        }
        SetLed(6, (BYTE)(bSecond / 10), 0);         //在数码 6 上显示秒钟的十位
        SetLed(7, (BYTE)(bSecond % 10), 0);         //在数码 7 上显示秒钟的个位
        break;
    case 3:                                         //设置 RTC 的秒钟模式
        SetLed(2, (BYTE)(bHour / 10), 0);           //在数码 2 上显示小时的十位
        SetLed(3, (BYTE)(bHour % 10), 1);           //在数码 3 上显示小时的个位
        SetLed(4, (BYTE)(bMinute / 10), 0);         //在数码 4 上显示分钟的十位
        SetLed(5, (BYTE)(bMinute % 10), 1);         //在数码 5 上显示分钟的个位
        if (fToggle)                                //闪烁控制
        {
            SetLed(6, (BYTE)(bSecond / 10), 0);     //在数码 6 上显示秒钟的十位
            SetLed(7, (BYTE)(bSecond % 10), 0);     //在数码 7 上显示秒钟的个位
        }
        else
        {
            SetLed(6, -1, 0);                       //在数码 6 上关闭显示,产生闪烁效果
            SetLed(7, -1, 0);                       //在数码 7 上关闭显示
        }
        break;
    }
}

void RTC_SetMode()              //RTC 工作模式控制
{
    if (++bMode > 3)            //RTC 工作模式转换(0:正常工作模式->1:小时设置模式->
                                //2:分钟设置模式->3:秒钟设置模式->0:正常工作模式->...)
    bMode = 0;                  //如果 RTC 模式大于 3,则 RTC 重归正常工作模式
}

void RTC_SetUp()                //RTC 时间上调控制
{
    if (bMode == 0)             //如果是正常工作模式,上调无意义
        return;                 //直接返回

    switch (bMode)              //根据工作模式调整时/分/秒
    {
    case 1:                     //小时设置模式
        if (++bHour >= 24)      //小时计数值向上递增
            bHour = 0;          //溢出后自动归零
        break;
    case 2:                     //分钟设置模式
        if (++bMinute >= 60)    //分钟计数值向上递增
```

```
            bMinute = 0;            //溢出后自动归零
        break;
    case 3:                         //秒钟设置模式
        if (++bSecond >= 60)        //秒钟计数值向上递增
            bSecond = 0;            //溢出后自动归零
        break;
    }

    fToggle = 1;                    //调整完后立即强制显示,不用等待 500ms
    RTC_Show();                     //显示调整的时间
}

void RTC_SetDown()                  //RTC 时间下调控制
{
    if (bMode == 0)                 //如果是正常工作模式,下调无意义
    return;                         //直接返回

    switch (bMode)                  //根据工作模式调整时/分/秒
    {
    case 1:                         //小时设置模式
        if (--bHour >= 24)          //小时计数值向下递减
            bHour = 23;             //溢出后自动设置上限值
        break;
    case 2:                         //分钟设置模式
        if (--bMinute >= 60)        //分钟计数值向下递减
            bMinute = 59;           //溢出后自动设置上限值
        break;
    case 3:                         //秒钟设置模式
        if (--bSecond >= 60)        //秒钟计数值向下递减
            bSecond = 59;           //溢出后自动设置上限值
        break;
    }

    fToggle = 1;                    //调整完后立即强制显示,不用等待 500ms
    RTC_Show();                     //显示调整的时间
}
```

f) beep.c -- 蜂鸣器发声控制程序

```
//beep.c
//蜂鸣器发声控制程序
//实验箱上的蜂鸣器为无源蜂鸣器
//通过蜂鸣器的控制口 P5.0 上输入不同频率的方波
//即可控制蜂鸣器发出不同频率的声音
//从而达到播放不同音调的效果
```

```
#include "config.h"             //头文件中已包含 ai8051u.h, ai_usb.h 以及其他头文件

//根据 Do/Re/Mi/Fa/So/La/Si/Do 这 8 个音符的频率定义定时器 1 的定时重载值
//由于定时器是翻转模式,所以需要将音符的频率值*2
#define   BEEP_DO     (WORD)(65536 - (float)FOSC / 12 / (261.6 * 2))
#define   BEEP_RE     (WORD)(65536 - (float)FOSC / 12 / (293.6 * 2))
#define   BEEP_MI     (WORD)(65536 - (float)FOSC / 12 / (329.6 * 2))
#define   BEEP_FA     (WORD)(65536 - (float)FOSC / 12 / (349.2 * 2))
#define   BEEP_SO     (WORD)(65536 - (float)FOSC / 12 / (392.0 * 2))
#define   BEEP_LA     (WORD)(65536 - (float)FOSC / 12 / (440.0 * 2))
#define   BEEP_SI     (WORD)(65536 - (float)FOSC / 12 / (493.8 * 2))
#define   BEEP_DOH    (WORD)(65536 - (float)FOSC / 12 / (523.2 * 2))

bit fBeepEn;                    //蜂鸣器发声使能控制

void BeepOn(BYTE bBeep)         //蜂鸣器发声控制程序
{
    switch (bBeep)              //根据函数的入口参数,控制蜂鸣器发不同的声音
    {
    case 0:                     //入口参数为 0 时,蜂鸣器发 Do 的声音
        TL1 = BEEP_DO;          //将 Do 的定时器重装值低 8 位加载到定时器 1 低位寄存器
        TH1 = BEEP_DO >> 8;     //将 Do 的定时器重装值高 8 位加载到定时器 1 高位寄存器
        break;
    case 1:                     //入口参数为 1 时,蜂鸣器发 Re 的声音
        TL1 = BEEP_RE;          //将 Re 的定时器重装值低 8 位加载到定时器 1 低位寄存器
        TH1 = BEEP_RE >> 8;     //将 Re 的定时器重装值高 8 位加载到定时器 1 高位寄存器
        break;
    case 2:                     //入口参数为 2 时,蜂鸣器发 Mi 的声音
        TL1 = BEEP_MI;          //将 Mi 的定时器重装值低 8 位加载到定时器 1 低位寄存器
        TH1 = BEEP_MI >> 8;     //将 Mi 的定时器重装值高 8 位加载到定时器 1 高位寄存器
        break;
    case 3:                     //入口参数为 3 时,蜂鸣器发 Fa 的声音
        TL1 = BEEP_FA;          //将 Fa 的定时器重装值低 8 位加载到定时器 1 低位寄存器
        TH1 = BEEP_FA >> 8;     //将 Fa 的定时器重装值高 8 位加载到定时器 1 高位寄存器
        break;
    case 4:                     //入口参数为 4 时,蜂鸣器发 So 的声音
        TL1 = BEEP_SO;          //将 So 的定时器重装值低 8 位加载到定时器 1 低位寄存器
        TH1 = BEEP_SO >> 8;     //将 So 的定时器重装值高 8 位加载到定时器 1 高位寄存器
        break;
    case 5:                     //入口参数为 5 时,蜂鸣器发 La 的声音
        TL1 = BEEP_LA;          //将 La 的定时器重装值低 8 位加载到定时器 1 低位寄存器
        TH1 = BEEP_LA >> 8;     //将 La 的定时器重装值高 8 位加载到定时器 1 高位寄存器
        break;
    case 6:                     //入口参数为 6 时,蜂鸣器发 Si 的声音
        TL1 = BEEP_SI;          //将 Si 的定时器重装值低 8 位加载到定时器 1 低位寄存器
```

```
        TH1 = BEEP_SI >> 8;         //将 Si 的定时器重装值高 8 位加载到定时器 1 高位寄存器
        break;
    case 7:                         //入口参数为 70 时,蜂鸣器发 DoH 的声音
        TL1 = BEEP_DOH;             //将 DoH 的定时器重装值低 8 位加载到定时器 1 低位寄存器
        TH1 = BEEP_DOH >> 8;        //将 DoH 的定时器重装值高 8 位加载到定时器 1 高位寄存器
        break;
    default:                        //入口参数为其他值时
        BeepOff();                  //关闭蜂鸣器
        return;                     //直接返回
    }
    fBeepEn = 1;                    //是蜂鸣器发声
}

void BeepOff()                      //关闭蜂鸣器发声
{
    fBeepEn = 0;                    //关闭蜂鸣器发声控制位
}
```

g) config.h -- 项目配置的头文件

```
#ifndef __CONFIG_H__            //防止头文件被重复包含
#define __CONFIG_H__

#define HIRC            40000000UL
#define FOSC            40000000UL
#define SYSCLK          FOSC
#define MAIN_Fosc       FOSC

#include <ai8051u.h>            //包含外部头文件
#include <stdio.h>
#include <intrins.h>

#include "def.h"                //包含项目头文件
#include "ai_usb.h"
#include "timer.h"
#include "key.h"
#include "led.h"
#include "rtc.h"
#include "beep.h"

void SYS_Init(void);            //函数声明

#endif
```

h) key.h -- 项目配置文件的头文件

```
#ifndef __KEY_H__          //防止头文件被重复包含
#define __KEY_H__

void ScanKey();            //函数声明
void KeyEvents();          //函数声明

#endif
```

i) led.h -- 项目配置文件的头文件

```
#ifndef __LED_H__          //防止头文件被重复包含
#define __LED_H__

void Led_Init(void);       //函数声明
void ScanLed(void);        //函数声明
void SetLed(BYTE nIndex, BYTE nNum, BOOL bShowPoint);

#endif
```

j) timer.h -- 项目配置文件的头文件

```
#ifndef __TIMER_H__        //防止头文件被重复包含
#define __TIMER_H__

void Timer0_Init(void);    //函数声明
void Timer1_Init(void);    //函数声明

extern bit f1ms;           //外部变量声明
extern bit f500ms;         //外部变量声明

#endif
```

k) rtc.h -- 项目配置文件的头文件

```
#ifndef __RTC_H__          //防止头文件被重复包含
#define __RTC_H__

void RTC_Init();           //函数声明
void RTC_Update();         //函数声明
void RTC_Show();           //函数声明
void RTC_SetMode();        //函数声明
void RTC_SetUp();          //函数声明
void RTC_SetDown();        //函数声明
```

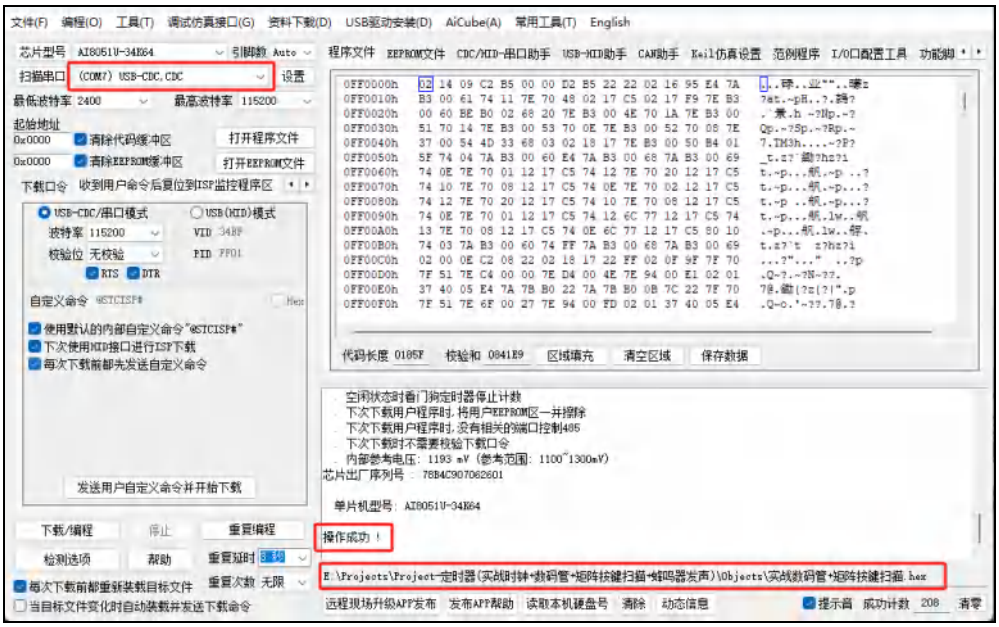
```
#endif
```

l) beep.h -- 蜂鸣器发声控制程序的头文件

```
#ifndef __BEEP_H__
#define __BEEP_H__

void BeepOn(BYTE bBeep);        //函数声明
void BeepOff();                 //函数声明

extern bit fBeepEn;             //外部变量声明

#endif
```

17.9.6 程序下载

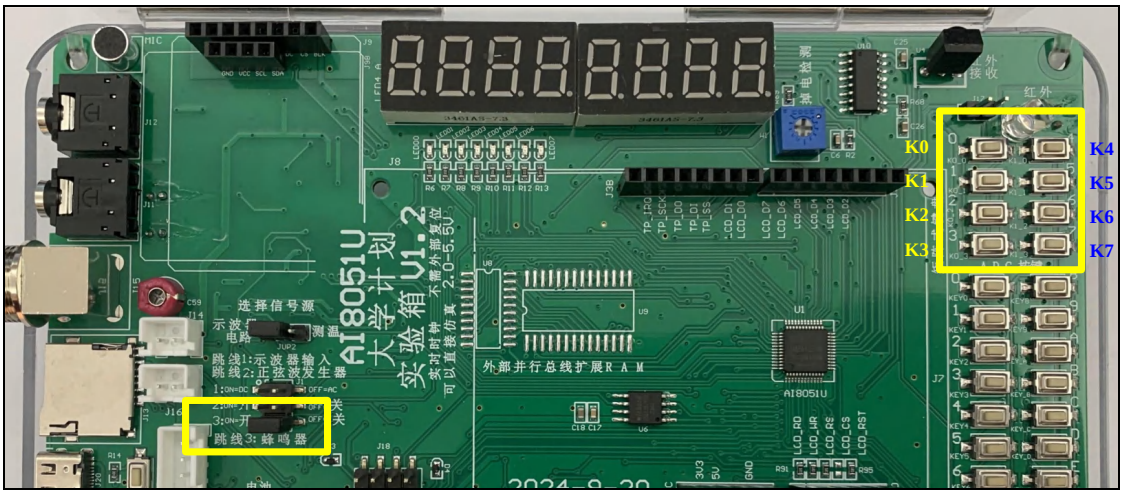
录入代码，保存，编译。连接好 Ai8051U 实验箱，运行 AIapp-ISP 软件系统，按步骤打开并下载程序“实战数码管+矩阵按键扫描.hex”，如下图：



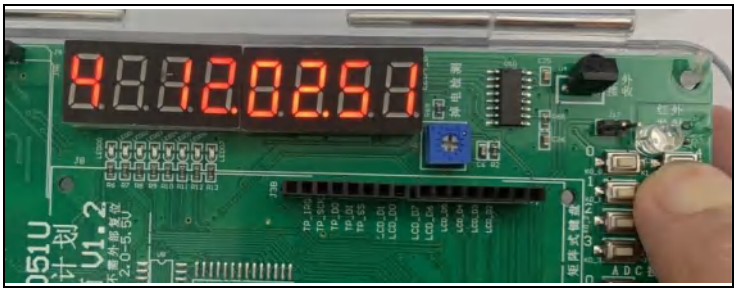
再观察 Ai8051U 实验箱，因本实验 RTC 的初始化时间为 12:30:00，那么 Ai8051U 实验箱数码低 6 位显示计时从 12:30:00 开始：

17.9.7 程序实现效果

将 Ai8051U 实验箱上“跳线 3：蜂鸣器”的跳线连上。
找到 K0_0 ~K0_3、K1_0 ~ K1_3 这 8 个按键（如下图）：



分别按下 K0_0 ~K0_3、K1_0 ~ K1_3 这 8 个按键，在 Ai8051U 实验箱数码管最高位会显示当前的按键键码，并且蜂鸣器会发出对应的“Do、Re、Mi、Fa、So、La、Si、Do”的音符声。如下图，按下 K1_0 按键，数码管最高位会显示 4，蜂鸣器发出 Fa 音符声：



矩阵按键中的按键 0，是 RTC 模式设置键

- 上电后 RTC 为正常工作模式，此时数码管动态显示 RTC 时间
- 按一次按键 0，RTC 会切换为小时设置模式，此时 RTC 的小时值会 0.5 秒闪烁一次
- 再按一次按键 0，RTC 会切换为分钟设置模式，此时 RTC 的分钟值会 0.5 秒闪烁一次
- 再按一次按键 0，RTC 会切换为秒钟设置模式，此时 RTC 的秒钟值会 0.5 秒闪烁一次
- 在按一次按键 0，恢复为 RTC 正常工作模式

矩阵按键中的按键 4 和 5 分别是上调键和下调键

- 只有在 RTC 的时/分/秒设置模式式，上调键和下调键才有效

时钟设置模式时，长按按键 4 和 5，可实现快速设置功能

18 RTC 实时时钟，年/月/日/时/分/秒

产品线	RTC	支持星期	芯片复位时 RTC 寄存器不复位[1]
Ai8051U 系列	●		●

[1]：包括硬件复位和软件复位

Ai8051U 系列部分单片机内部集成一个实时时钟控制电路，主要有如下特性：

- 低功耗：RTC 模块工作电流低至 **2uA@VCC=3.3V、3uA@VCC=5.0V（典型值）**
- 长时间跨度：支持 2000 年~2099 年，并自动判断闰年
- 闹钟：支持一组闹钟设置
- 支持多个中断：
  - 一组闹钟中断（每天中断一次，中断的时间点为闹钟寄存器所设置的任意时/分/秒）
  - 日中断（每天中断一次，中断的时间点为每天的 0 时 0 分 0 秒）
  - 小时中断（每小时中断一次，中断的时间点为分/秒均为 0，即整点时）
  - 分钟中断（每分钟中断一次，中断的时间点为秒为 0，即分钟寄存器发生变化时）
  - 秒中断（每秒中断一次，中断的时间点为秒寄存器发生变化时）
  - 1/2 秒中断（每 1/2 秒中断一次）
  - 1/8 秒中断（每 1/8 秒中断一次）
  - 1/32 秒中断（每 1/32 秒中断一次）
- 支持掉电唤醒
- **特别注意：Ai8051U 系列单片机的 RTC 无星期功能**

18.1 RTC 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
RTCCR	RTC 控制寄存器	7EFE60H	-	-	-	-	-	-	-	RUNRTC	xxxx,xxx0
RTCCFG	RTC 配置寄存器	7EFE61H	-	-	-	-	-	-	RTCCKS	SETRTC	xxxx,xx00
RTCIEN	RTC 中断使能寄存器	7EFE62H	EALAI	EDAYI	EHOURI	EMINI	ESECI	ESEC2I	ESEC8I	ESEC32I	0000,0000
RTCIF	RTC 中断请求寄存器	7EFE63H	ALAIF	DAYIF	HOURIF	MINIF	SECIF	SEC2IF	SEC8IF	SEC32IF	0000,0000
ALAHOUR	RTC 闹钟的小时值	7EFE64H	-	-	-						xxx0,0000
ALAMIN	RTC 闹钟的分钟值	7EFE65H	-	-							xx00,0000
ALASEC	RTC 闹钟的秒值	7EFE66H	-	-							xx00,0000
ALASSEC	RTC 闹钟的 1/128 秒值	7EFE67H	-								x000,0000
INIYEAR	RTC 年初始化	7EFE68H	-								x000,0000
INIMONTH	RTC 月初始化	7EFE69H	-	-	-	-					xxxx,0000
INIDAY	RTC 日初始化	7EFE6AH	-	-	-						xxx0,0000
INIHOUR	RTC 小时初始化	7EFE6BH	-	-	-						xxx0,0000
INIMIN	RTC 分钟初始化	7EFE6CH	-	-							xx00,0000
INISEC	RTC 秒初始化	7EFE6DH	-	-							xx00,0000
INISSEC	RTC1/128 秒初始化	7EFE6EH	-								x000,0000
INIWEEK RTCWEEK	RTC 星期初始化 RTC 的星期计数值	7EFE6FH	-	-	-	-	-				xxxx,x000
RTCYEAR	RTC 的年计数值	7EFE70H	-								x000,0000
RTCMONTH	RTC 的月计数值	7EFE71H	-	-	-	-					xxxx,0000
RTCDAY	RTC 的日计数值	7EFE72H	-	-	-						xxx0,0000
RTCHOUR	RTC 的小时计数值	7EFE73H	-	-	-						xxx0,0000
RTCMIN	RTC 的分钟计数值	7EFE74H	-	-							xx00,0000
RTCSEC	RTC 的秒计数值	7EFE75H	-	-							xx00,0000
RTCSSEC	RTC 的 1/128 秒计数值	7EFE76H	-								x000,0000

18.1.1 RTC 控制寄存器（RTCCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
RTCCR	7EFE60H	-	-	-	-	-	-	-	RUNRTC

RUNRTC：RTC 模块控制位

0：关闭 RTC，RTC 停止计数

1：使能 RTC，并开始 RTC 计数

18.1.2 RTC 配置寄存器（RTCCFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
RTCCFG	7EFE61H	-	-	-	-	-	-	RTCCKS	SETRTC

RTCCKS：RTC 时钟源选择

0：选择外部 32.768KHz 时钟源（需先软件启动外部 32K 晶振）

1：选择内部 32K 时钟源（需先软件启动内部 32K 振荡器）

SETRTC：设置 RTC 初始值

写 0：无意义

写 1:触发 RTC 寄存器初始化。当 SETRTC 设置为 1 时，硬件会自动将寄存器 INIYEAR、INIMONTH、INIDAY、INIWEEK、INIHOUR、INIMIN、INISEC、INISSEC 中的值复制到寄存器 RTCYEAR、RTCMONTH、RTCDAY、RTCWEEK、RTCHOUR、RTCMIN、RTCSEC、RTCSSEC 中。初始完成后，硬件会自动将 SETRTC 位清 0。

读 0：设置 RTC 相关时间寄存器已完成

读 1：硬件正在进行设置 RTC，还未完成

注：等待初始化完成，需要在"RTC 使能"之后判断。设置 RTC 时间需要 32768Hz 的 1 个周期时间，大约 30.5us。由于同步，所以实际等待时间是 0~30.5us，如果不等待设置完成就睡眠，则 RTC 会由于设置没完成，停止计数，唤醒后才继续完成设置并继续计数，如果此时设置的是使用 RTC 中断进行唤醒，则会出现无法唤醒 MCU 的情况。

18.1.3 RTC 中断使能寄存器（RTCIEN）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
RTCIEN	7EFE62H	EALAI	EDAYI	EHOURI	EMINI	ESECI	ESEC2I	ESEC8I	ESEC32I

EALAI：闹钟中断使能位

0：关闭闹钟中断

1：使能闹钟中断

EDAYI：一日（24 小时）中断使能位

0：关闭一日中断

1：使能一日中断

EHOURI：一小时（60 分钟）中断使能位

0：关闭小时中断

1：使能小时中断

EMINI：一分钟（60 秒）中断使能位

0：关闭分钟中断

1：使能分钟中断

ESECI：一秒中断使能位

0：关闭秒中断

1：使能秒中断

ESEC2I：1/2 秒中断使能位

0：关闭 1/2 秒中断

1：使能 1/2 秒中断

ESEC8I：1/8 秒中断使能位

0：关闭 1/8 秒中断

1：使能 1/8 秒中断

ESEC32I：1/32 秒中断使能位

0：关闭 1/32 秒中断

1：使能 1/32 秒中断

18.1.4 RTC 中断请求寄存器（RTCIF）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
RTCIF	7EFE63H	ALAIF	DAYIF	HOURIF	MINIF	SECIF	SEC2IF	SEC8IF	SEC32IF

ALAIF：闹钟中断请求位。需软件写 0 清 0。
DAYIF：一日（24 小时）中断请求位。需软件写 0 清 0。
HOURIF：一小时（60 分钟）中断请求位。需软件写 0 清 0。
MINIF：一分钟（60 秒）中断请求位。需软件写 0 清 0。
SECIF：一秒中断请求位。需软件写 0 清 0。
SEC2IF：1/2 秒中断请求位。需软件写 0 清 0。
SEC8IF：1/8 秒中断请求位。需软件写 0 清 0。
SEC32IF：1/32 秒中断请求位。需软件写 0 清 0。

18.1.5 RTC 闹钟设置寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ALAHOUR	7EFE64H	-	-	-					
ALAMIN	7EFE65H	-	-						
ALASEC	7EFE66H	-	-						
ALASSEC	7EFE67H	-							

ALAHOUR：设置每天闹钟的小时值。

注意：设置的值不是 BCD 码，而是 HEX 码，比如需要设置小时值 20 到 ALAHOUR，则需使用如下代码进行设置

```
MOV     WR6,#WORD0 ALAHOUR
MOV     WR4,#WORD2 ALAHOUR
MOV     A,#14H
MOV     @DR4,R11
```

ALAMIN：设置每天闹钟的分钟值。数字编码与 ALAHOUR 相同。
ALASEC：设置每天闹钟的秒值。数字编码与 ALAHOUR 相同。
ALASSEC：设置每天闹钟的 1/128 秒值。数字编码与 ALAHOUR 相同。

18.1.6 RTC 实时时钟初始值设置寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INIYEAR	7EFE68H	-							
INIMONTH	7EFE69H								
INIDAY	7EFE6AH								
INIHOUR	7EFE6BH	-	-	-					
INIMIN	7EFE6CH	-	-						
INISEC	7EFE6DH	-	-						
INISSEC	7EFE6EH	-							
INIWEEK RTCWEEK	7EFE6FH	-	-	-	-	-			

INIYEAR：设置当前实时时间的年值。有效值范围 00~99。对应 2000 年~2099 年

注意：设置的值不是 BCD 码，而是 HEX 码，比如需要设置 20 到 INIYEAR，则需使用如下代码进行设置

```
MOV     WR6,#WORD0 INIYEAR
MOV     WR4,#WORD2 INIYEAR
MOV     A,#14H
MOV     @DR4,R11
```

INIMONTH：设置当前实时时间的月值。有效值范围 1~12。数字编码与 INIYEAR 相同。

INIDAY：设置当前实时时间的日值。有效值范围 1~31。数字编码与 INIYEAR 相同。

INIHOUR：设置当前实时时间的小时值。有效值范围 00~23。数字编码与 INIYEAR 相同。

INIMIN：设置当前实时时间的分钟值。有效值范围 00~59。数字编码与 INIYEAR 相同。

INISEC：设置当前实时时间的秒值。有效值范围 00~59。数字编码与 INIYEAR 相同。

INISSEC：设置当前实时时间的 1/128 秒值。有效值范围 00~127。数字编码与 INIYEAR 相同。

INIWEEK：设置当前实时时间的星期。有效值范围 0~6。（星期计数值也从此寄存器读取）

当用户设置完成上面的初始值寄存器后，用户还需要向 SETRTC 位（RTCCFG.0）写 1 来触发硬件将初始值装载到 RTC 实时计数器中

另需注意：硬件不会对初始化数据的有效性进行检查，需要用户在设置初始值时，必须保证数据的有效性，不能超出其有效范围。

另需注意：硬件不会对初始化数据的有效性进行检查，需要用户在设置初始值时，必须保证数据的有效性，不能超出其有效范围。

18.1.7 RTC 实时时钟计数寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
RTCYEAR	7EFE70H	-							
RTCMONTH	7EFE71H								
RTCDAY	7EFE72H								
RTCHOUR	7EFE73H	-	-	-					
RTCMIN	7EFE74H	-	-						
RTCSEC	7EFE75H	-	-						
RTCSSEC	7EFE76H	-							

RTCYEAR：当前实时时间的年值。**注意：寄存器的值不是 BCD 码，而是 HEX 码**

RTCMONTH：当前实时时间的月值。数字编码与 YEAR 相同。

RTCDAY：当前实时时间的日值。数字编码与 YEAR 相同。

RTCHOUR：当前实时时间的小时值。数字编码与 YEAR 相同。

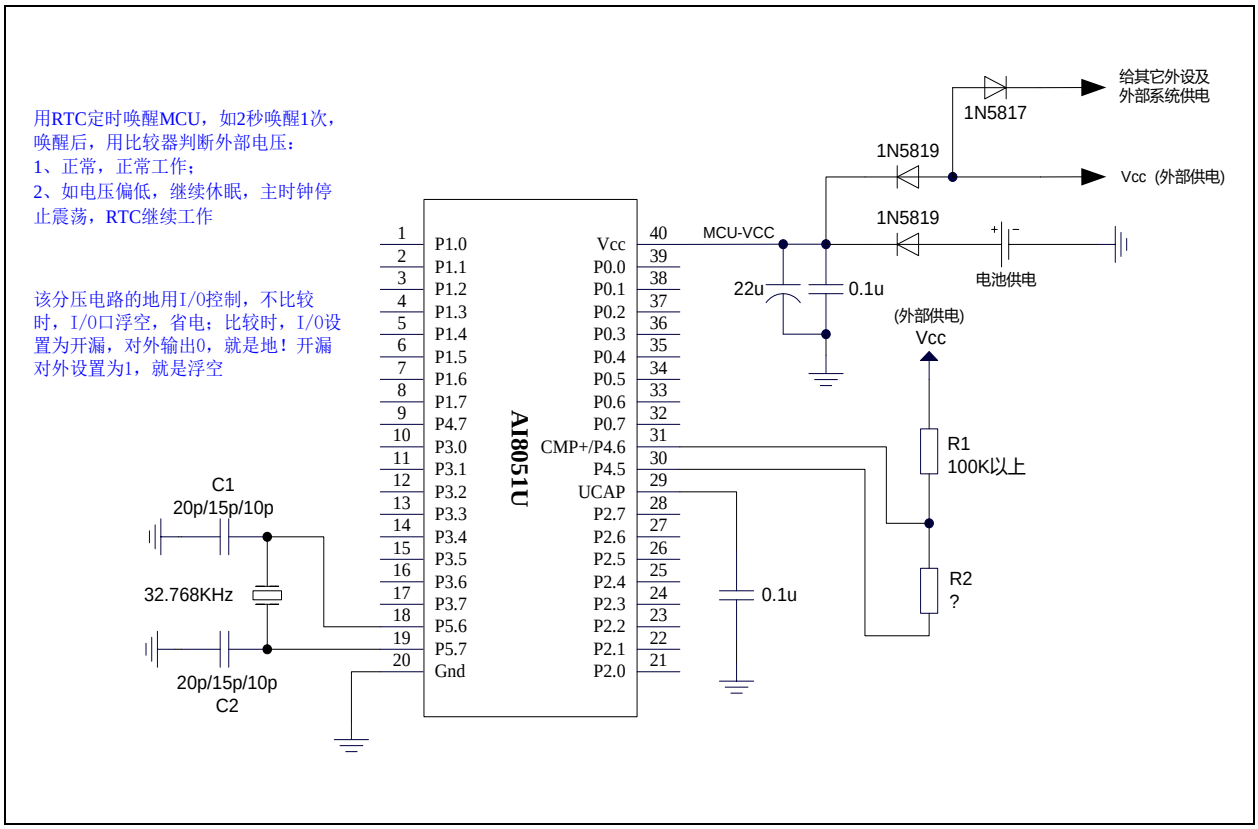
RTCMIN：当前实时时间的分钟值。数字编码与 YEAR 相同。

RTCSEC：当前实时时间的秒值。数字编码与 YEAR 相同。

RTCSSEC：当前实时时间的 1/128 秒值。数字编码与 YEAR 相同。

注意：RTCYEAR、RTCMONTH、RTCDAY、RTCHOUR、RTCMIN、RTCSEC 和 RTCSSEC 均为只读寄存器，若需要对这些寄存器执行写操作，必须通过寄存器 INIYEAR、INIMONTlH、INIDAT、INIHOU、INIMIN、INISEC、INISSEC 和 SETRTC 来实现。

18.2 RTC 实战线路图



18.3　范例程序

18.3.1　串口打印 RTC 时钟范例

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。
#include "stdio.h"

#define   MAIN_Fosc       22118400L
#define   Baudrate        115200L
#define   TM              (65536 -(MAIN_Fosc/Baudrate+2)/4)

bit       B1S_Flag;
void RTC_config(void);

void UartInit(void)
{
    SCON = (SCON & 0x3f) | 0x40;
    T2L  = TM;
    T2H  = TM>>8;
    S1BRT = 1;
    T2x12 = 1;
    T2R = 1;
}

void UartPutc(unsigned char dat)
{
    SBUF = dat;
    while(TI==0);
    TI = 0;
}

char putchar(char c)
{
    UartPutc(c);
    return c;
}

void RTC_Isr() interrupt 13
{
    if(RTCIF & 0x08)        //判断是否秒中断
    {
        RTCIF &= ~0x08;                                 //清中断标志
        B1S_Flag = 1;
    }
}

void main(void)
{
    EAXFR = 1;                                          //允许访问扩展的特殊寄存器，XFR
                                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                      //允许访问扩展的特殊寄存器，XFR
                                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                           //设置取程序代码等待时间，
```

```
                                                    //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                      //设置访问片内的 xdata 速度，
                                                    //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M1 = 0;   P0M0 = 0;   //设置为准双向口
    P1M1 = 0;   P1M0 = 0;   //设置为准双向口
    P2M1 = 0;   P2M0 = 0;   //设置为准双向口
    P3M1 = 0;   P3M0 = 0;   //设置为准双向口
    P4M1 = 0;   P4M0 = 0;   //设置为准双向口
    P5M1 = 0;   P5M0 = 0;   //设置为准双向口

    UartInit();
    RTC_config();
    EA = 1;
    printf("RTC Test Programme!\r\n");          //UART 发送一个字符串

    while (1)
    {
        if(B1S_Flag)
        {
            B1S_Flag = 0;

            printf("Year=20%bd    ", RTCYEAR);
            printf("Month=%bd    ", RTCMONTH);
            printf("Day=%bd    ", RTCDAY);
            printf("Hour=%bd    ", RTCHOUR);
            printf("Minute=%bd    ", RTCMIN);
            printf("Second=%bd    ", RTCSEC);
            printf("\r\n");
        }
    }
}

void RTC_config(void)
{
//  选择内部低速 IRC
//  IRC32KCR = 0x80;                            //启动内部低速 IRC 振荡器
//  while (!(IRC32KCR & 0x01));                 //等待时钟稳定
//  RTCCFG |= 0x02;                             //选择内部低速 IRC 作为 RTC 时钟源

    //选择外部 32K
    X32KCR = 0xc0;                              //启动外部 32K 晶振
    while (!(X32KCR & 0x01));                   //等待时钟稳定
    RTCCFG &= ~0x02;                            //选择外部 32K 作为 RTC 时钟源

    INIYEAR = 21;                               //Y:2021
    INIMONTH = 12;                              //M:12
    INIDAY = 31;                                //D:31
    INIHOUR = 23;                               //H:23
    INIMIN = 59;                                //M:59
    INISEC = 50;                                //S:50
    INISSEC = 0;                                //S/128:0
    RTCIF = 0;                                  //清中断标志
    RTCIEN = 0x08;                              //使能 RTC 秒中断
    RTCCR = 0x01;                               // RTC 使能
    RTCCFG |= 0x01;                             //触发 RTC 寄存器初始化
    while(RTCCFG & 0x01);                       //等待初始化完成,需要在 "RTC 使能" 之后判断.
                                                //设置 RTC 时间需要 32768Hz 的 1 个周期时间,
                                                //大约 30.5us. 由于同步, 所以实际等待时间是 0~30.5us.
```

```
                                        //如果不等待设置完成就睡眠，则 RTC 会由于设置没
                                        //完成，停止计数，唤醒后才继续完成设置并继续计数.
}
```

汇编代码

```
;将以下代码保存为 ASM 格式文件，一起加载到项目里，例如：isr.asm

        CSEG    AT  0123H
        JMP         006BH
        END
```

18.3.2 利用 ISP 软件的用户接口实现不停电下载保持 RTC 参数

```
//测试工作频率为 11.0592MHz

/*************  功能说明  ***************************************************************************
现有单片机系列的 RTC 模块，在单片机复位后 RTC 相关的特殊功能寄存器也会复位
本例程主要用于解决 ISP 下载后用户的 RTC 参数丢失的问题

解决思路：ISP 下载前，先将 RTC 相关参数通过 ISP 下载软件的用户接口上传到 PC 保存，等待 ISP 下载完成后，下载软件再将保存的相关参数写入到 FLASH 的指定地址（范例中指定的地址为 FE0000H）。ISP 下载完成后会立即运行用户代码，用户程序在初始化 RTC 寄存器时，可从 FLASH 的指定地址中读取之前上传的 RTC 相关参数对 RTC 寄存器进行初始化，即可实现不停电下载保持 RTC 参数的目的。

下载时，选择时钟 11.0592MHZ
*****************************************************************************************************/

#include “Ai8051U.H”
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。
#include "stdio.h"

#define   FOSC              11059200UL
#define   BAUD              (65536 – (FOSC/115200+2)/4)

typedef   bit               BOOL;
typedef   unsigned char     BYTE;
typedef   unsigned int      WORD;

struct RTC_INIT
{
    BYTE    bValidTag;                                  //数据有效标志(0x5a)
    BYTE    bIniYear;                                   //年(RTC 初始化值)
    BYTE    bIniMonth;                                  //月
    BYTE    bIniDay;                                    //日
    BYTE    bIniHour;                                   //时
    BYTE    bIniMinute;                                 //分
    BYTE    bIniSecond;                                 //秒
    BYTE    bIniSSecond;                                //次秒
    BYTE    bAlaHour;                                   //时(RTC 闹钟设置值)
    BYTE    bAlaMinute;                                 //分
    BYTE    bAlaSecond;                                 //秒
    BYTE    bAlaSSecond;                                //次秒
};

struct RTC_INIT ecode InitBlock _at_ 0xfe0000;

void SysInit();
void UartInit();
void RTCInit();
void SendUart(BYTE dat);
void UnpackCmd(BYTE dat);
void IapProgram(WORD addr, BYTE dat);

BOOL fUartBusy;
BOOL fFetchRtc;
BOOL fReset2Isp;
BYTE bUartStage;
```

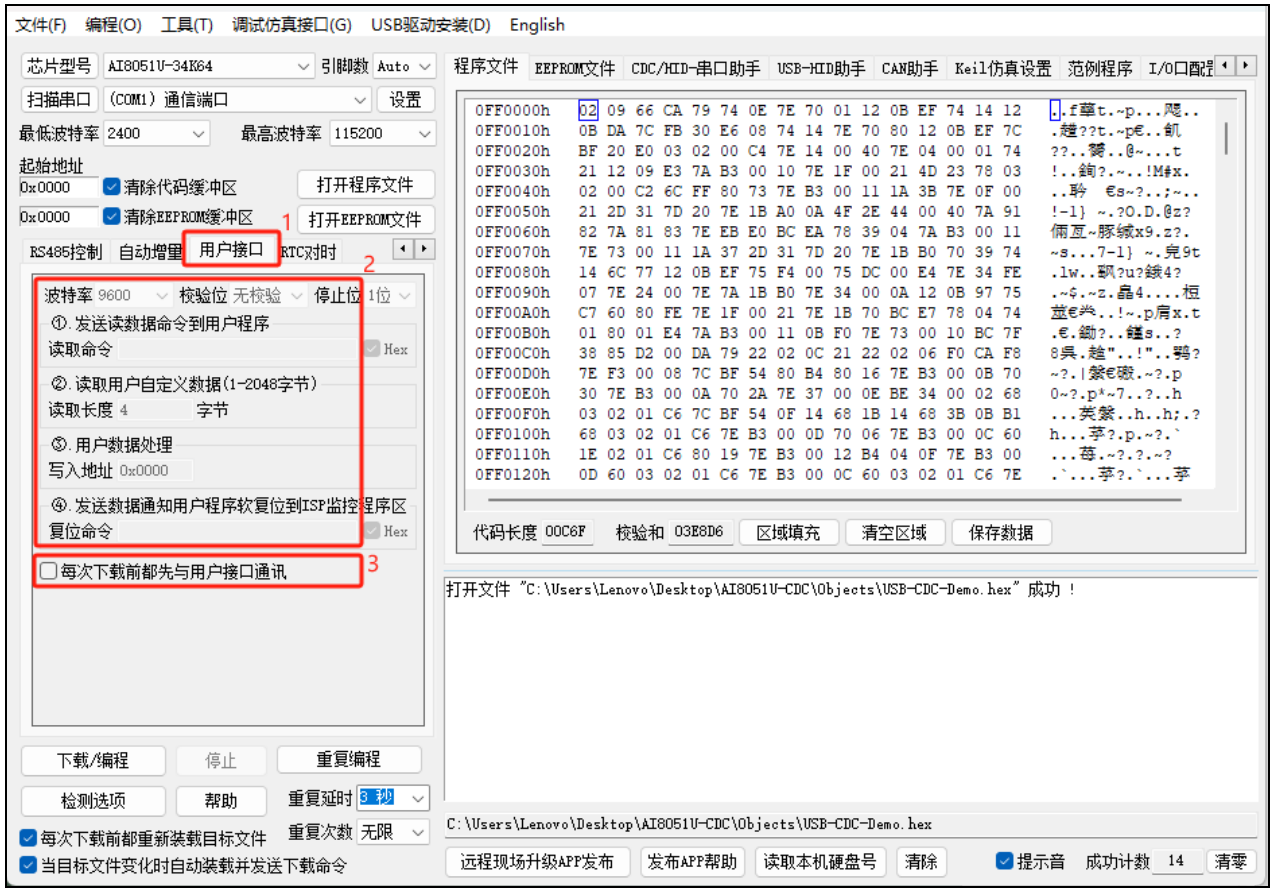
```
BYTE bDump[7];

void main()
{
    SysInit();                                  //系统初始化
    UartInit();
    RTCInit();
    EA = 1;

    fUartBusy = 0;
    fFetchRtc = 0;
    fReset2Isp = 0;
    bUartStage = 0;

    while (1)
    {
        if (fFetchRtc)                          //获取 RTC 数据请求
        {
            fFetchRtc = 0;

            RTCCR = 0;                          //上传当前的 RTC 值时,必须临时停止 RTC
                                                //以免发生进位错误
            bDump[0] = RTCYEAR;                 //快速将当前的 RTC 值缓存,
                                                //以缩短 RTC 暂停的时间,减小误差
            bDump[1] = RTCMONTH;
            bDump[2] = RTCDAY;
            bDump[3] = RTCHOUR;
            bDump[4] = RTCMIN;
            bDump[5] = RTCSEC;
            bDump[6] = RTCSSEC;
            RTCCR = 1;

            SendUart(0x5a);                     //上传 12 字节 RTC 参数
            SendUart(bDump[0]);
            SendUart(bDump[1]);
            SendUart(bDump[2]);
            SendUart(bDump[3]);
            SendUart(bDump[4]);
            SendUart(bDump[5]);
            SendUart(bDump[6]);
            SendUart(ALAHOUR);
            SendUart(ALAMIN);
            SendUart(ALASEC);
            SendUart(ALASSEC);
        }

        if (fReset2Isp)                         //重启请求
        {
            fReset2Isp = 0;

            IAP_CONTR = 0x60;                   //软件触发复位到系统 ISP 区
        }
    }
}

void uart_isr() interrupt UART1_VECTOR
{
    BYTE dat;
```

```
    if (TI)
    {
        TI = 0;

        fUartBusy =     0;
    }

    if (RI)
    {
        RI = 0;

        dat = SBUF;
        switch (bUartStage++)                       //解析串口命令
        {
        default:
        case 0:
L_Check1st:
            if (dat == '@') bUartStage = 1;
            else bUartStage = 0;
            break;
        case 1:
            if (dat == 'F') bUartStage = 2;
            else if (dat == 'R') bUartStage = 7;
            else goto L_Check1st;
            break;
        case 2:
            if (dat != 'E') goto L_Check1st;
            break;
        case 3:
            if (dat != 'T') goto L_Check1st;
            break;
        case 4:
            if (dat != 'C') goto L_Check1st;
            break;
        case 5:
            if (dat != 'H') goto L_Check1st;
            break;
        case 6:
            if (dat != '#') goto L_Check1st;
            bUartStage = 0;
            fFetchRtc = 1;                          //当前命令序列为获取 RTC 数据命令:"@FETCH#"
            break;
        case 7:
            if (dat != 'E') goto L_Check1st;
            break;
        case 8:
            if (dat != 'B') goto L_Check1st;
            break;
        case 9:
        case 10:
            if (dat != 'O') goto L_Check1st;
            break;
        case 11:
            if (dat != 'T') goto L_Check1st;
            break;
        case 12:
            if (dat != '#') goto L_Check1st;
            bUartStage = 0;
```

```
                fReset2Isp = 1;                         //当前命令序列为重启命令:"@REBOOT#"
                break;
            }
        }
    }

void rtc_isr() interrupt RTC_VECTOR                     //RTC 中断复位程序
{
    RTCIF = 0x00;                                       //清 RTC 中断标志

    P20 = !P20;                                         //P2.0 口每秒闪烁一次,测试用
}

void SysInit()
{
    EAXFR = 1;                                          //允许访问扩展的特殊寄存器，XFR
                                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                      //允许访问扩展的特殊寄存器，XFR
                                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                           //设置取程序代码等待时间，
                                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                          //设置访问片内的 xdata 速度，
                                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00; P0M1 = 0x00;
    P1M0 = 0x00; P1M1 = 0x00;
    P2M0 = 0x00; P2M1 = 0x00;
    P3M0 = 0x00; P3M1 = 0x00;
    P4M0 = 0x00; P4M1 = 0x00;
    P5M0 = 0x00; P5M1 = 0x00;
    P6M0 = 0x00; P6M1 = 0x00;
    P7M0 = 0x00; P7M1 = 0x00;
}

void UartInit()                                         //串口初始化函数
{
    SCON = 0x50;
    AUXR = 0x40;
    TMOD = 0x00;
    TL1 = BAUD;
    TH1 = BAUD >> 8;
    TR1 = 1;
    ES = 1;
}

void RTCInit()                                          //RTC 初始化函数
{
//  IRC32KCR = 0x80;
//  while (!(IRC32KCR &0x01));
//  RTCCFG |= 0x02;                                     //选择内部低速 IRC 为 RTC 时钟源

    X32KCR = 0xc0;
    while (!(X32KCR & 0x01));
    RTCCFG &= ~0x02;                                    //选择外部部 32K 为 RTC 时钟源

    if (InitBlock.bValidTag == 0x5a)
    {
        INIYEAR = InitBlock.bIniYear;                   //如果初始化数据块有效,则使用数据块初始化 RTC
        INIMONTH = InitBlock.bIniMonth;
```

```
            INIDAY = InitBlock.bIniDay;
            INIHOUR = InitBlock.bIniHour;
            INIMIN = InitBlock.bIniMinute;
            INISEC = InitBlock.bIniSecond;
            INISSEC = InitBlock.bIniSSecond;
            ALAHOUR = InitBlock.bAlaHour;
            ALAMIN = InitBlock.bAlaMinute;
            ALASEC = InitBlock.bAlaSecond;
            ALASSEC = InitBlock.bAlaSSecond;

            IapProgram(0x0000, 0x00);                   //销毁初始化数据块,以免重复初始化
        }
        else
        {
            INIYEAR = 23;                               //否则初始化 RTC 为默认值
            INIMONTH = 1;
            INIDAY = 29;
            INIHOUR = 12;
            INIMIN = 0;
            INISEC = 0;
            INISSEC = 0;
            ALAHOUR = 0;
            ALAMIN = 0;
            ALASEC = 0;
            ALASSEC = 0;
        }
        RTCCFG |= 0x01;                                 //写入 RTC 初始值
        RTCCR = 0x01;                                   //RTC 开始运行
        while (RTCCFG & 0x01);                          //等待 RTC 初始化完成
        RTCIF = 0x00;
        RTCIEN = 0x08;                                  //使能 RTC 秒中断
}

void SendUart(BYTE dat)                                 //串口发送函数
{
        while (fUartBusy);
        SBUF = dat;
        fUartBusy = 1;
}

void IapProgram(WORD addr, BYTE dat)                    //EEPROM 编程函数
{
        IAP_CONTR = 0x80;
        IAP_TPS = 11;
        IAP_CMD = 2;
        IAP_ADDRL = addr;
        IAP_ADDRH = addr >> 8;
        IAP_DATA = dat;
        IAP_TRIG = 0x5a;
        IAP_TRIG = 0xa5;
        _nop_();
        _nop_();
        _nop_();
        _nop_();}
}
```

ISP 下载软件中“用户接口”的设置如下：（注意，首次下载不能使能用户接口）



18.3.3 内部 RTC 时钟低功耗休眠唤醒-比较器检测电压程序

```
/************* 本程序功能说明 **************
本例程基于 AI8H8K64U 为主控芯片的实验箱 9 进行编写测试，AI8H 系列带 RTC 模块的芯片可通用参考.
读写芯片内部集成的 RTC 模块.
电路连接参考规格书 RTC 章节-RTC 实战线路图.
用 RTC 定时唤醒 MCU，如 1 秒唤醒 1 次，唤醒后用比较器判断外部电压：1，正常，正常工作；2，如电压偏低，继续休眠，主时钟停止震荡，RTC 继续工作.
比较器正极通过电阻分压后输入到 P3.7 口，比较器负极使用内部 1.19V 参考电压.
该分压电路的地用 I/O(P3.5)控制，I/O 设置为开漏，不比较时，对外设置为 1，I/O 口浮空，省电；比较时，对外输出 0，就是地！

下载时，  选择时钟 24MHZ (用户可自行修改频率).
*****************************************/

#include “Ai8051U.H”
#include "stdio.h"
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

typedef     unsigned char   u8;
typedef     unsigned int    u16;
typedef     unsigned long   u32;

/****************************** 用户定义宏 ***********************************/

#define   MAIN_Fosc     24000000L                       //定义主时钟
#define   PrintUart     2                               //1:printf 使用 UART1; 2:printf 使用 UART2
#define   Baudrate      115200L
#define   TM            (65536 –(MAIN_Fosc/Baudrate+2)/4)

/*****************************************************************************/


/************* 本地常量声明 **************/

u8 code   ledNum[]={0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};

/************* 本地变量声明 **************/

bit   B_1s;
bit   B_Alarm;                                          //闹钟标志
u8    ledIndex;

/************* 本地函数声明 **************/

void RTC_config(void);
void CMP_config(void);
void Ext_Vcc_Det(void);

/********************** 串口打印函数 ************************/
void UartInit(void)
{
#if(PrintUart == 1)
    SCON = (SCON & 0x3f) | 0x40;
    AUXR |= 0x40;                                       //定时器时钟 1T 模式
    AUXR &= 0xFE;                                       //串口 1 选择定时器 1 为波特率发生器
    TL1 = TM;
```

```
    TH1 = TM>>8;
    TR1 = 1;                                    //定时器 1 开始计时

//  SCON = (SCON & 0x3f) | 0x40;
//  T2L = TM;
//  T2H = TM>>8;
//  AUXR |= 0x15;                               //串口 1 选择定时器 2 为波特率发生器
#else
    P_SW2 |= 1;                                 //UART2   switch to: 0: P1.0 P1.1,   1: P4.6 P4.7
    S2CON &= ~(1<<7);                           //8 位数据,1 位起始位,1 位停止位,  无校验
    T2L = TM;
    T2H = TM>>8;
    AUXR |= 0x14;                               //定时器 2 时钟 1T 模式,开始计时
#endif
}

void UartPutc(unsigned char dat)
{
#if(PrintUart == 1)
    SBUF = dat;
    while(TI == 0);
    TI = 0;
#else
    S2BUF = dat;
    while((S2CON & 2) == 0);
    S2CON &= ~2;                                //Clear Tx flag
#endif
}

char putchar(char c)
{
    UartPutc(c);
    return c;
}

/*********************************************/
void main(void)
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M1 = 0x00;    P0M0 = 0x00;                //设置为准双向口
    P1M1 = 0x00;    P1M0 = 0x00;                //设置为准双向口
    P2M1 = 0x00;    P2M0 = 0x00;                //设置为准双向口
    P3M1 = 0xa0;    P3M0 = 0x20;                //设置为准双向口
                                                // P3.5 设置开漏模式, P3.7 设置高阻输入
    P4M1 = 0x00;    P4M0 = 0x00;                //设置为准双向口
    P5M1 = 0x00;    P5M0 = 0x00;                //设置为准双向口
    P6M1 = 0x00;    P6M0 = 0x00;                //设置为准双向口
    P7M1 = 0x00;    P7M0 = 0x00;                //设置为准双向口

    IRCDB = 0x10;       //内部高速 IRC 时钟，停振后被唤醒后起振，等待多少个时钟数后给 MCU 供应时钟
    IAP_TPS = 40;       //根据系统工作频率设置此寄存器
```

```
                        //（如果系统工作频率为 40MHz，则 IAP_TPS 设置为 40；
                        //  如果系统工作频率为 22.1184MHz，则 IAP_TPS 设置为 22）

    UartInit();
    CMP_config();
    RTC_config();
    EA = 1;                                         //打开总中断

    while(1)
    {
        if(B_1s)
        {
            B_1s = 0;
            printf("Year=20%bd,Month=%bd,Day=%bd,Hour=%bd,Minute=%bd,Second=%bd\r\n",
                RTCYEAR, RTCMONTH, RTCDAY, RTCHOUR, RTCMIN, RTCSEC);

            Ext_Vcc_Det();                          //每秒钟检测一次外部电源，
                                                    //如果外部电源连接则工作，
                                                    外部电源断开则进入休眠模式
        }

        if(B_Alarm)
        {
            B_Alarm   = 0;
            printf("RTC      Alarm!\r\n");
        }
    }
}

//========================================================================
// 函数: void Ext_Vcc_Det(void)
// 描述: 外部电源检测函数。
// 参数: 无.
// 返回: 无.
//========================================================================
void Ext_Vcc_Det(void)
{
    P35 = 0;                                        //比较时，对外输出 0，做比较电路的地线
    CMPCR1 |= 0x80;                                 //使能比较器模块
    _nop_();
    _nop_();
    _nop_();
    if(CMPCR1 & 0x01)                               //判断是否 CMP+电平高于 CMP-，外部电源连接
    {
        P40 = 0;                                    //LED Power   On
        P6 = ~ledNum[ledIndex];                     //输出低驱动
        ledIndex++;
        if(ledIndex> 7)
        {
            ledIndex = 0;
        }
    }
    else
    {
        CMPCR1 &= ~0x80;                            //关闭比较器模块
        P35 = 1;                                    //不比较时，对外设置为 1，I/O 口浮空，省电
        P40 = 1;                                    //LED Power Off
        _nop_();
        _nop_();
```

```
            PCON = 0x02;                                //AI8H8K64U B 版本芯片使用内部 32K 时钟，
                                                        休眠无法唤醒

            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
        }
}

//========================================================================
// 函数: void CMP_config(void)
// 描述: 比较器初始化函数。
// 参数: 无.
// 返回: 无.
//========================================================================
void CMP_config(void)
{
        CMPEXCFG = 0x00;
//      CMPEXCFG |= 0x40;                               //比较器 DC 迟滞输入选择，0:0mV;
                                                        // 0x40:10mV; 0x80:20mV; 0xc0:30mV

//      CMPEXCFG &=       ~0x04;                        //P4.4 为 CMP-输入脚
        CMPEXCFG |=  0x04;                              //内部 1.19V 参考电压为 CMP-输入脚

        CMPEXCFG &=       ~0x03;                        //P4.6 为 CMP+输入脚
//      CMPEXCFG |=  0x01;                              //P5.0 为 CMP+输入脚
//      CMPEXCFG |=  0x02;                              //P5.1 为 CMP+输入脚
//      CMPEXCFG |=  0x03;                              //ADC 输入脚为 CMP+输入脚


        CMPCR2 = 0x00;
        CMPCR2 &= ~0x80;                                //比较器正向输出
//      CMPCR2 |= 0x80;                                 //比较器反向输出
        CMPCR2 &= ~0x40;                                //使能 0.1us 滤波
//      CMPCR2 |= 0x40;                                 //禁止 0.1us 滤波
        CMPCR2 &= ~0x3f;                                //比较器结果直接输出
//      CMPCR2 |= 0x10;                                 //比较器结果经过 16 个去抖时钟后输出

        CMPCR1 = 0x00;
//      CMPCR1 |= 0x30;                                 //使能比较器边沿中断
//      CMPCR1 &= ~0x20;                                //禁止比较器上升沿中断
//      CMPCR1 |= 0x20;                                 //使能比较器上升沿中断
//      CMPCR1 &= ~0x10;                                //禁止比较器下降沿中断
//      CMPCR1 |= 0x10;                                 //使能比较器下降沿中断

        CMPCR1 &= ~0x02;                                //禁止比较器输出
//      CMPCR1 |= 0x02;                                 //使能比较器输出

//      P_SW2 &= ~0x08;                                 //选择 P4.5 作为比较器输出脚
        P_SW2 |= 0x08;                                  //选择 P4.1 作为比较器输出脚
        CMPCR1 |= 0x80;                                 //使能比较器模块
}

void RTC_config(void)
```

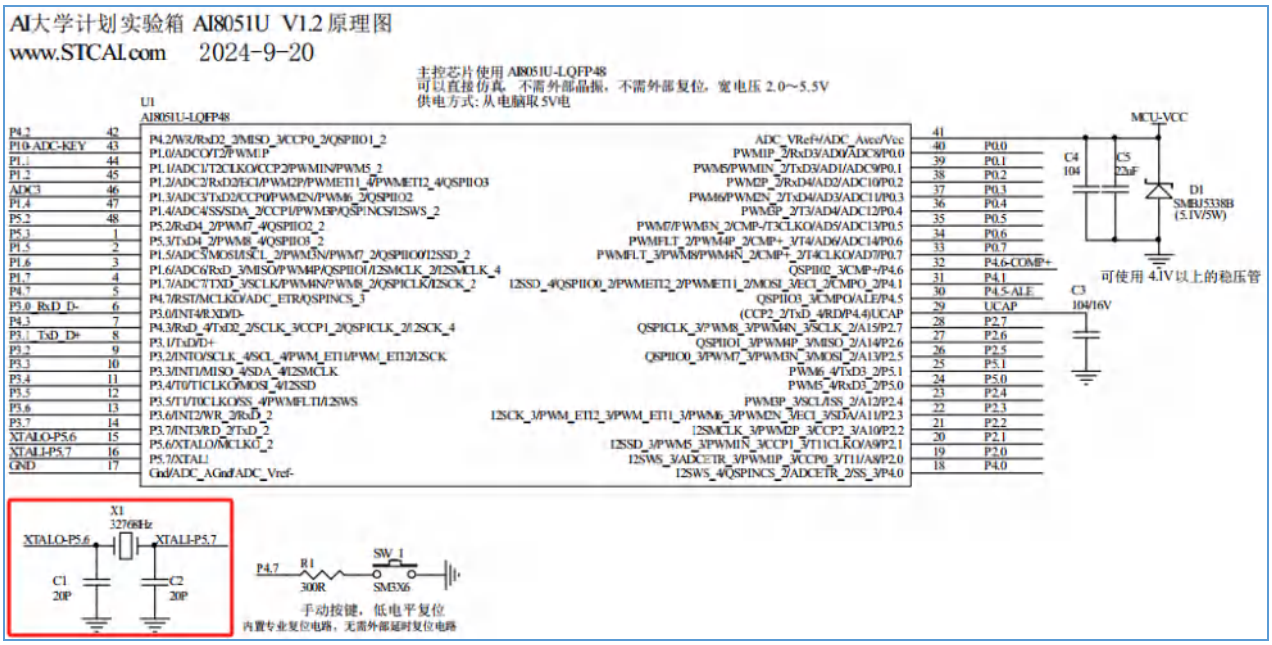
```
{
    INIYEAR  = 21;                          //Y:2021
    INIMONTH = 12;                          //M:12
    INIDAY = 31;                            //D:31
    INIHOUR = 23;                           //H:23
    INIMIN = 59;                            //M:59
    INISEC = 50;                            //S:50
    INISSEC = 0;                            //S/128:0

    ALAHOUR = 0;                            //闹钟小时
    ALAMIN = 0;                             //闹钟分钟
    ALASEC = 0;                             //闹钟秒
    ALASSEC = 0;                            //闹钟 1/128 秒

//AI8H8K64U B 版本芯片使用内部低速 IRC 时钟，休眠无法唤醒
//  IRC32KCR = 0x80;                        //启动内部低速 IRC.
//  while (!(IRC32KCR & 1));                //等待时钟稳定
//  RTCCFG = 0x03;                          //选择内部低速 IRC 时钟源，触发 RTC 寄存器初始化

    X32KCR = 0x80 + 0x40;                   //启动外部 32K 晶振, 低增益+0x00,  高增益+0x40.
    while (!(X32KCR & 1));                  //等待时钟稳定
    RTCCFG = 0x01;                          //选择外部 32K 时钟源，触发 RTC 寄存器初始化

    RTCIF = 0x00;                           //清中断标志
    RTCIEN = 0x88;                          //中断使能, 0x80:闹钟中断, 0x40:日中断, 0x20:小时中断,
                                            //0x10:分钟中断, 0x08:秒中断, 0x04:1/2 秒中断,
                                            //0x02:1/8 秒中断, 0x01:1/32 秒中断
    RTCCR = 0x01;                           //RTC 使能

    while(RTCCFG & 0x01);                   //等待初始化完成,需要在 "RTC 使能" 之后判断.
                                            //设置 RTC 时间需要 32768Hz 的 1 个周期时间,
                                            //大约 30.5us./由于同步, 所以实际等待时间是 0~30.5us.
                                            //如果不等待设置完成就睡眠,则 RTC 会由于设置没完成,
                                            //停止计数, 唤醒后才继续完成设置并继续计数.
}

/********************* RTC 中断函数 *********************/
void RTC_Isr() interrupt 13
{
    if(RTCIF & 0x80)                        //闹钟中断
    {
//      P01 = !P01;
        RTCIF &= ~0x80;
        B_Alarm = 1;
    }

    if(RTCIF & 0x08)                        //秒中断
    {
//      P00 = !P00;
        RTCIF &= ~0x08;
        B_1s = 1;
    }
}
/**************************************************************************/
//如果开启了比较器中断就需要编写对应的中断函数
void CMP_Isr() interrupt 21
{
    CMPCR1 &= ~0x40;                        //清中断标志
//  P10 = CMPCR1 & 0x01;                    //中断方式读取比较器比较结果
```

}

18.4　深圳大学--RTC 实验

18.4.1　深大-RTC 实验(拓展学习之使用 RTC 功能设定闹钟)

a) 原理图



b) 实验介绍

程序运行约 5s 后，AIapp-ISP 软件的 CDC 窗口显示“RTC Start”，RTC 闹钟 10s 产生中断，打印数据在 CDC 窗口显示“Alarm！”

CDC 串口发送任意数据，读取 RTC 的秒值，并在 CDC 窗口显示

观察到：

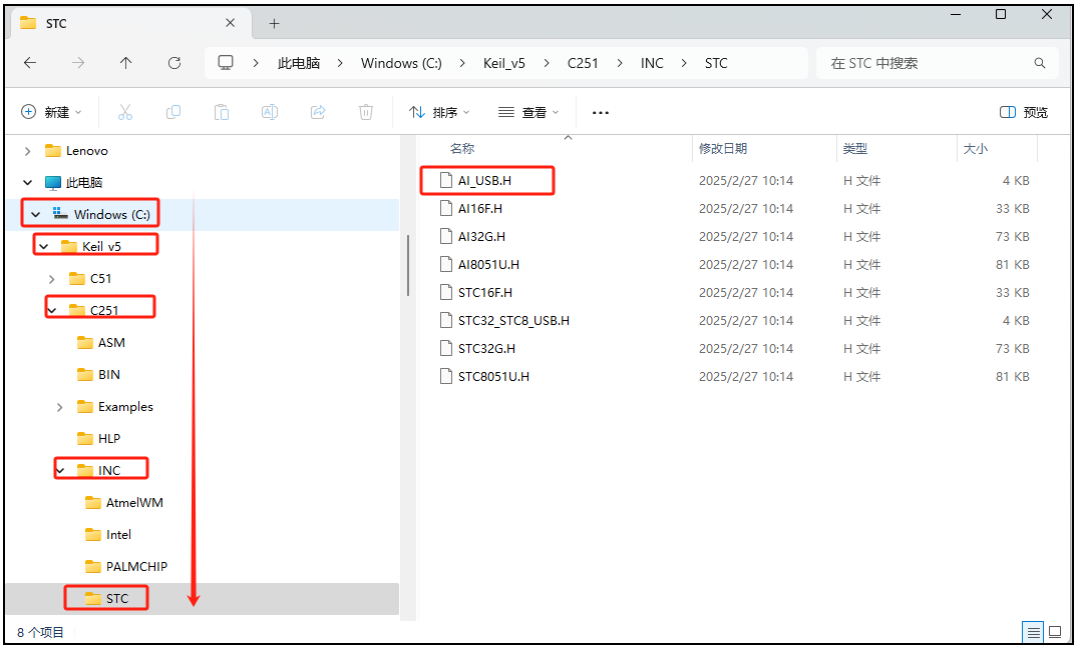
- 程序运行 5s 后，AIapp-ISP 软件的 CDC 窗口显示“RTC Start”，再 10s 后，CDC 窗口显示“Alarm！”
- CDC 串口发送任意数据，在 CDC 窗口显示当前的 RTC 秒值

c) 准备工作--添加头文件和库函数

1、添加常用头文件 ai_usb.h

```
#include "ai_usb.h"                    //使用 usb 所需要包含的头文件。
```

这个头文件在使用 AIapp-ISP 软件添加型号和头文件到 Keil 中时就已经一并被添加到 Keil 中了。



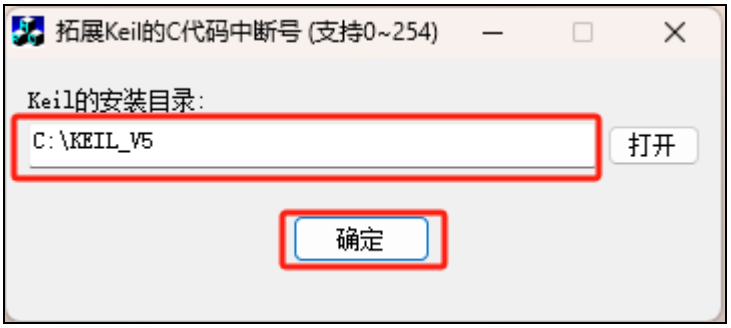
2、添加 Keil 中断拓展插件

Keil C51/C251 编译器只支持 31 以内的中断号（0~31），超过 31 编译会报错。RTC 的中断向量 RTC_VECTOR 为 36，这就需要 Keil 中断号拓展工具，可将中断号拓展到 254。

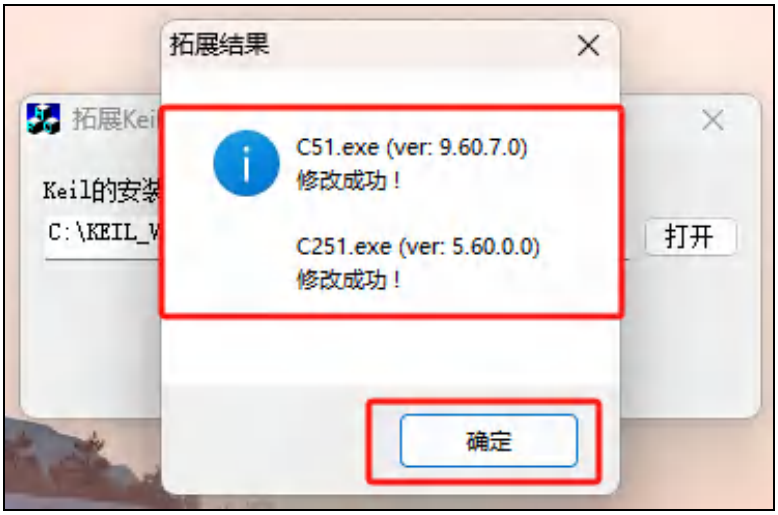
这部分可以到 STCAI 的官网：https://www.stcai.com/gjrj 的软件工具➔工具软件中“Keil 中断拓展插件”进行下载。

下载完成后，打开压缩包“拓展 Keil 的 C 代码中断号.zip”，找到“拓展 Keil 的 C 代码中断号.exe”，这是绿色免安装软件。

直接双击“拓展 Keil 的 C 代码中断号.exe”，出现如下界面，选择安装路径，按照默认直接安装到 C:\Keil_V5 目录下，点击“确定”



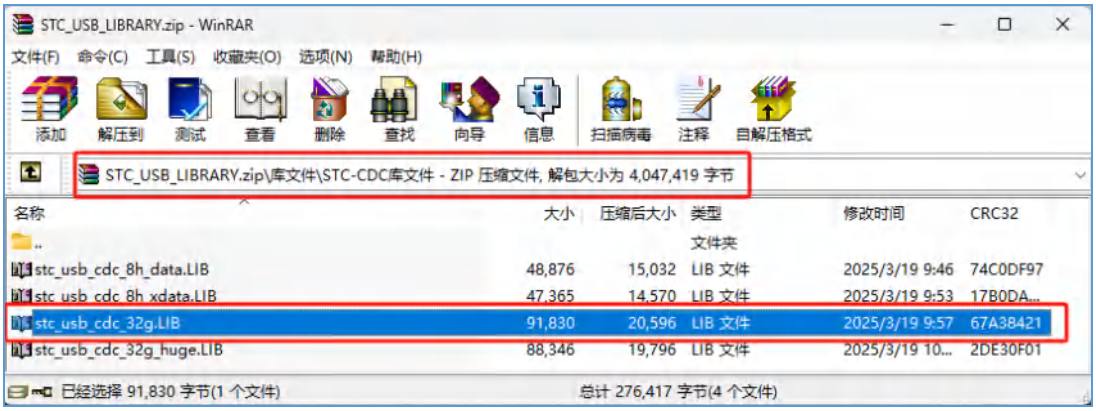
如下图，点击“确定”，Keil 的 C 代码中断号到 254



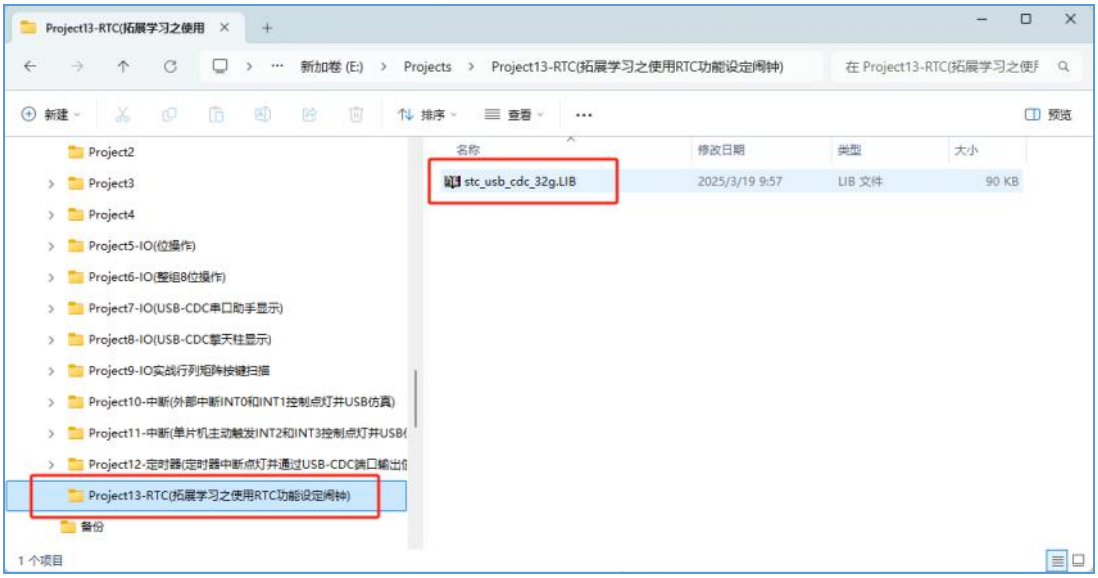
3、添加 usb_cdc 库函数文件

因为要使用到 USB 功能，必须在项目中添加 USB 库函数。这部分可以到 STCAI 的官网：https://www.stcai.com/khs 的 USB 库文件中进行下载

下载完成后，打开压缩包“STC_USB_LIBRARY.zip”，
在’库文件\查询模式 STC-CDC 库文件’中，复制出来 stc_usb_cdc_32g.LIB

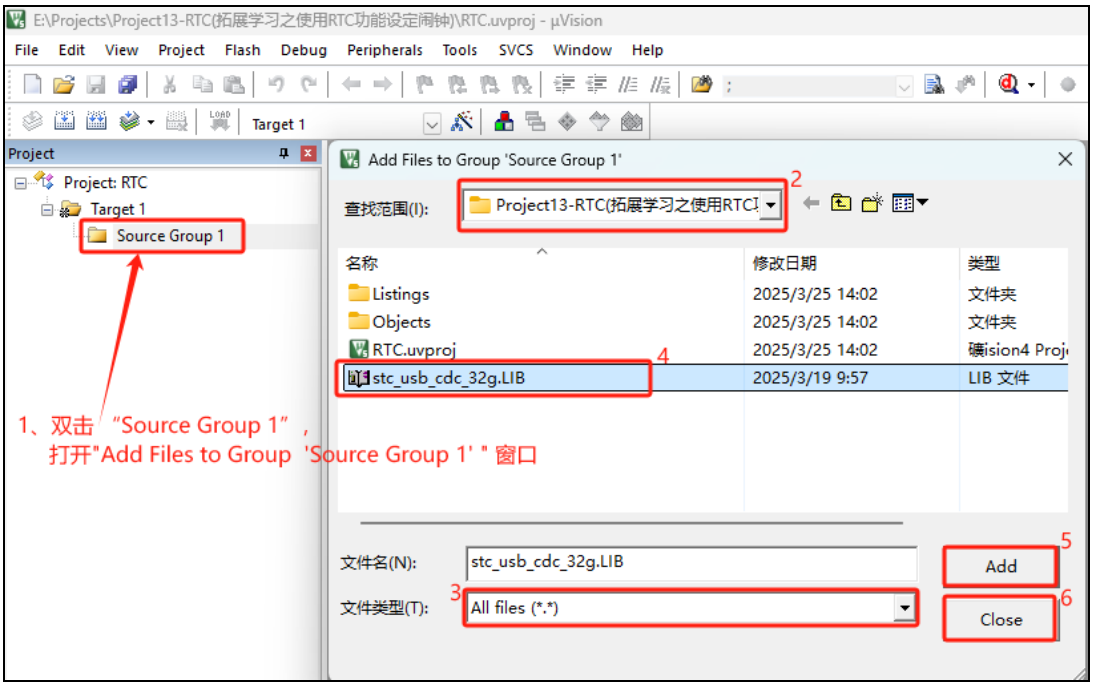


将库函数“stc_usb_cdc_32g.LIB”复制到创建工程所在的文件夹 project13 中

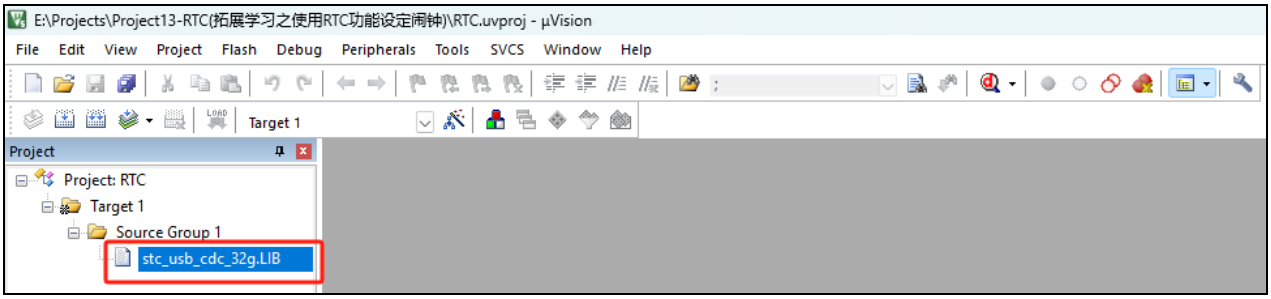


将库函数 stc_usb_cdc_32g.LIB 添加进入此工程项目中

在 Keil 中，双击 Source Group，在弹出的 Add Files to Group 'Source Group 1 窗口中，将库函数 stc_usb_cdc_32g.LIB 添加进入此工程，就可以使用 USB-CDC 功能点亮 LED 灯。



这样，Keil 界面变化如下图：



【注意】：

USB-CDC 串口则更类似传统串口的形式，类似 COM 口通讯，但不管串口波特率设置，Ai8051U 默认是 USB2.0-FS，12Mbps 在通讯。

手机不需要安装任何 USB 驱动，如：USB-CDC 驱动，USB-HID 驱动/任何系统都不需要安装。

现在的 Windows 操作系统也不需要安装 USB-CDC 驱动，只有 Win10(1903 版)之前的操作系统才需要安装 USB-CDC 驱动。安装也只是在 AIapp-ISP V6.95G 或更新的版本中执行【USB 驱动安装】菜单中的【安装 USB-CDC 驱动】即可。

d) 实验程序代码

```
#include <AI8051U.H>
#include   "ai_usb.h"    //USB 库和 USB 虚拟设备的头文件
//由于在 USB 库函数的代码中已经定义了，不停电 ISP 下载命令字"@STCISP#"，
//只需要在下载软件的“收到用户命令后复位到 ISP 监控程序区”选项卡中进行相应的设置，
//即可实现 USB 不停电 ISP 下载功能，或串口不停电 ISP 下载功能。
/*
如何获得上面这 2 个头文件，如已按照前面介绍的方法：
从 www.STCAI.com 网站，【软件工具 | 工具软件】处，下载安装最新的【AIapp-ISP 软件】，
选择其中的选项卡 【Keil 仿真设置 】，
点击【添加型号和头文件到 Keil 中，并添加仿真器驱动到 Keil 中】按钮，
则 Ai8051U.h 和 AI_usb.h 都会自动复制到 Keil 中，包含进来就可以直接使用，
大家可以打开看下其中的内容，AI_usb.h 和 stc32_stc8_usb.h 是相似的，只需用其中的 1 个头文件，
具体见 Keil_v5\C251\INC\STC 目录，或 Keil_v5\C51\INC\STC 目录。
如将对应的 USB 库文件添加到 项目中，后续用户程序中，USB 如收到命令字符串 "@STCISP#"
USB 库函数就会自动软复位到系统 ISP 程序区，实现 USB 不停电下载，具体见相应部分的说明。
//char *USER_DEVICEDESC = NULL;
//char *USER_PRODUCTDESC = NULL;
//char *USER_STCISPCMD = "@STCISP#";
//使用 20250319 之前版本的库函数时需要在用户代码中增加上面 3 行定义。
//使用 20250319 及以后的版本都不再需要上面的定义
*/
#include "intrins.h"          //使用_nop_()函数所必须要包含的头文件,
                              //否则延时函数中调用的_nop_()函数没有被头文件引用过来，
                              //会导致编译器找不到这个而函数而报错。

bit alarm_flag = 0;           //定义 RTC 中断标志
```

```
void rtc_isr(void) interrupt RTC_VECTOR  //RTC 中断服务程序
{
    alarm_flag = 1;
    RTCIF = 0;              //清除 RTC 中断标志
}

void sys_init(void)         //系统初始化
{
    EAXFR = 1;              //允许访问扩展的特殊寄存器，XFR
    WTST = 0;               //设置取程序代码等待时间，赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;              //设置访问片内的 xdata 速度，赋值为 0 表示用最快速度访问，不增加额外的等待时间
    P0M0 = 0x00; P0M1 = 0x00;     //设置 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x00;     //设置 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;     //设置 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x00;     //设置 P3 口为准双向口模式
    P4M0 = 0x00; P4M1 = 0x00;     //设置 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;     //设置 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;     //设置 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;     //设置 P7 口为准双向口模式
}

void Delay5000ms(void)          //延时函数（延时 5s）@40.000MHz
{
    unsigned long edata i;

    _nop_();
    _nop_();
    i = 49999998UL;
    while (i) i--;
}

void main(void)
{
    sys_init();                 //调用系统初始化函数
    usb_init();                 //调用 USB 初始化函数
    EA = 1;                     //总中断允许位打开
    P40 = 0;                    //给 LED 灯供电

    Delay5000ms();              //调用延时 5s，否则根本看不到 USB-CDC 窗口显示“RTC Start”

    X32KCR = 0xc0;              //启动外部 32K 晶振
    while (!(X32KCR & 0x01));   //等待时钟稳定
    RTCCFG &= ~0x02;            //选择外部 32K 作为 RTC 时钟源

    RTCIF = 0;                  //清除 RTC 中断标志
```

```
    RTCIEN = 0x80;                  //使能 RTC 闹钟中断
    ALASEC = 10;                    //闹钟秒设定，这里设定 10s
                                    //因为没有设置初始值，所以初始值默认都是从 00 开始走的
                                    //这里设置 10 就相当于从 RTC 初始化完成后开始 10s 就会接收到闹钟中断

    RTCCR = 0x01;               // RTC 模块使能
    RTCCFG |= 0x01;             //触发 RTC 寄存器初始化
    while(RTCCFG & 0x01);       //等待 RTC 初始化完成

    printf_usb("RTC Start\r\n");
        //直接打印输出 “RTC Start” 到主机相应软件的 USB【CDC/HID-串口助手】

    while(1)                        //判断 RTC 闹钟中断，打印数据
    {
        if(alarm_flag)              //闹钟中断到了
        {
            alarm_flag = 0;
            printf_usb("Alarm!\r\n");           //打印一下闹钟到达的信息
        }

        if(bUsbOutReady)        //读取是否发送消息
            //USB 库中有定义 bUsbOutReady 位变量，查询方式必须调用一次“usb_OUT_done();”
        {
            printf_usb("This_sec:%d\r\n",(int)SEC);         //发送消息的时候返回当前的秒数
                //直接打印输出 “This_sec:%d” 到主机相应软件的 USB【CDC/HID-串口助手】

            usb_OUT_done();         //每次处理完 OUT 信息，最后都要调用这个函数
                //当使用查询方式 if(bUsbOutReady)，查询调用 USB 库函数时，在数据处理完成后，
                //必须调用一次“usb_OUT_done();”，这样才能接收下一包数据。
        }
    }
}
```

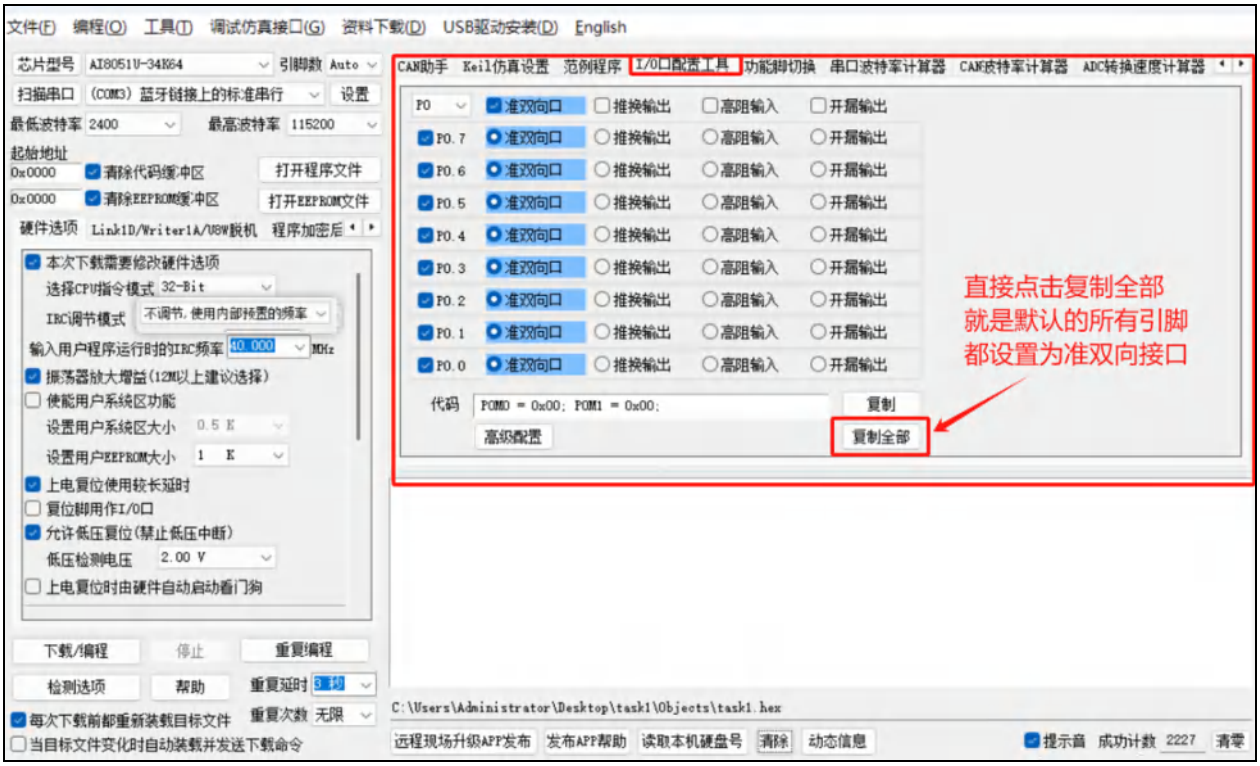
【小技巧（一）-- 灵活运用 AIapp-ISP 软件的工具自动生成代码】

I/O 口初始化代码部分就可以使用 **AIapp-ISP** 软件的“**I/O 口配置工具**”来生成。

例如上述代码中这一段将所有 I/O 口设置为准双向口模式的代码：

```
    P0M0 = 0x00; P0M1 = 0x00;
    P1M0 = 0x00; P1M1 = 0x00;
    P2M0 = 0x00; P2M1 = 0x00;
    P3M0 = 0x00; P3M1 = 0x00;
    P4M0 = 0x00; P4M1 = 0x00;
    P5M0 = 0x00; P5M1 = 0x00;
    P6M0 = 0x00; P6M1 = 0x00;
    P7M0 = 0x00; P7M1 = 0x00;
```

就可以使用 AIapp-ISP 的 I/O 口配置工具进行生成：



将复制出来的内容直接粘贴到代码文件中，不用手动输入。

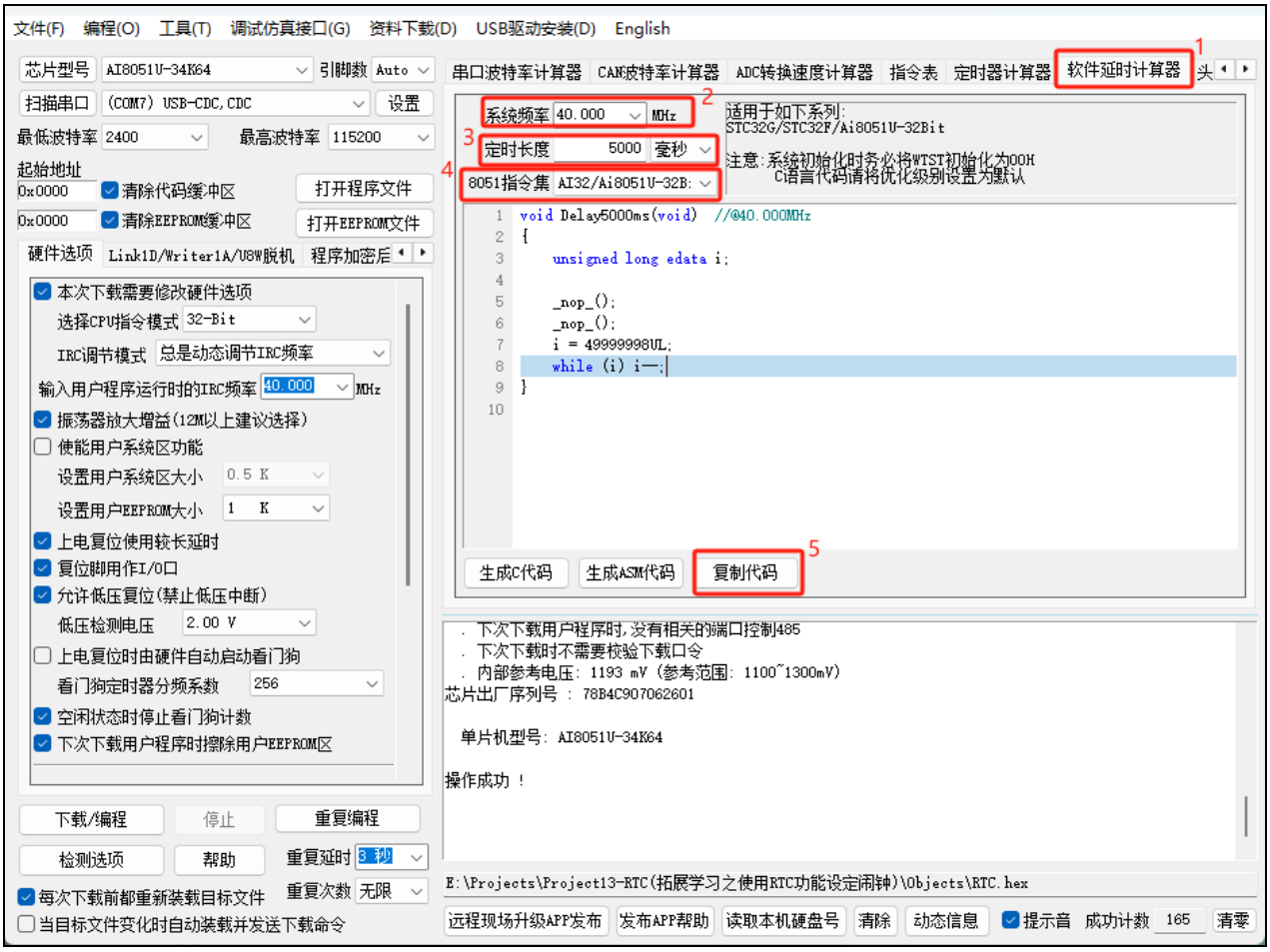
【小技巧（二）-- 灵活运用 AIapp-ISP 软件的工具自动生成代码】

延时函数部分代码就可以使用 **AIapp-ISP** 软件的“**软件延时计算器**”来生成。

例如上述代码中这一段延时函数的代码：

```
void Delay5000ms(void)	//@40.000MHz
{
	unsigned long edata i;

	_nop_();
	_nop_();
	i = 49999998UL;
	while (i) i--;
}
```

就可以使用 AIapp-ISP 软件的“软件延时计算器”来生成：



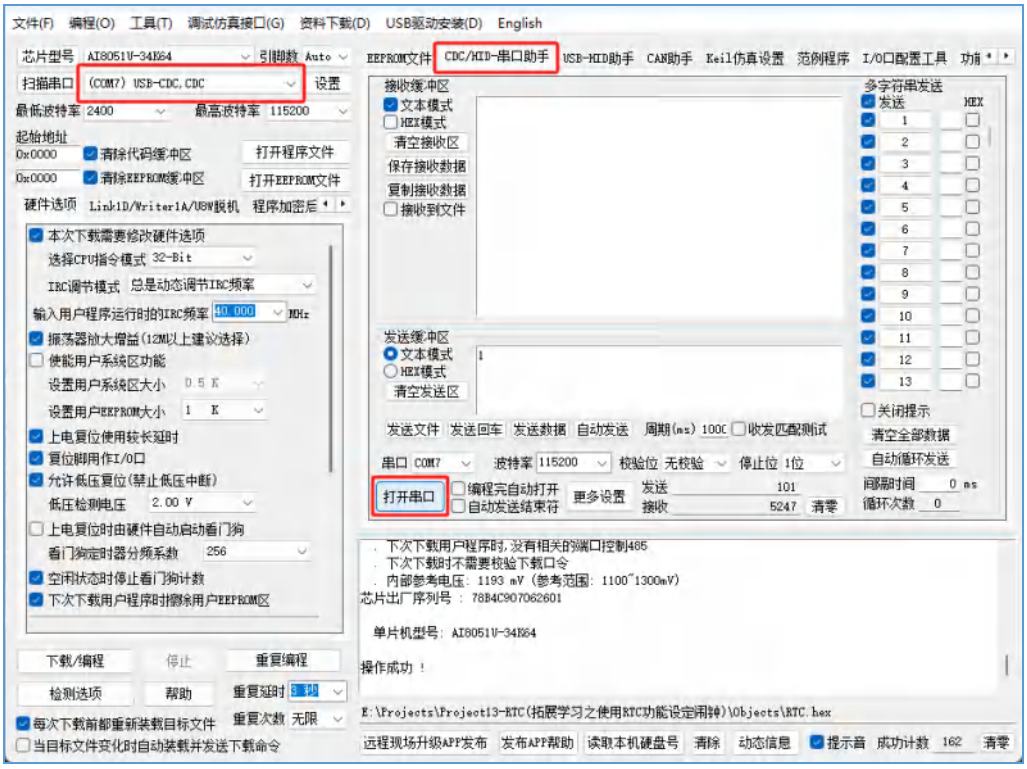
将复制出来的内容直接粘贴到代码文件中，不用手动输入。

e) 程序下载

程序编译，并下载实验箱完成以后，就可以看到出现了一个 USB-CDC 端口，此时就可以通过这个 USB-CDC 端口进行数据通讯了。

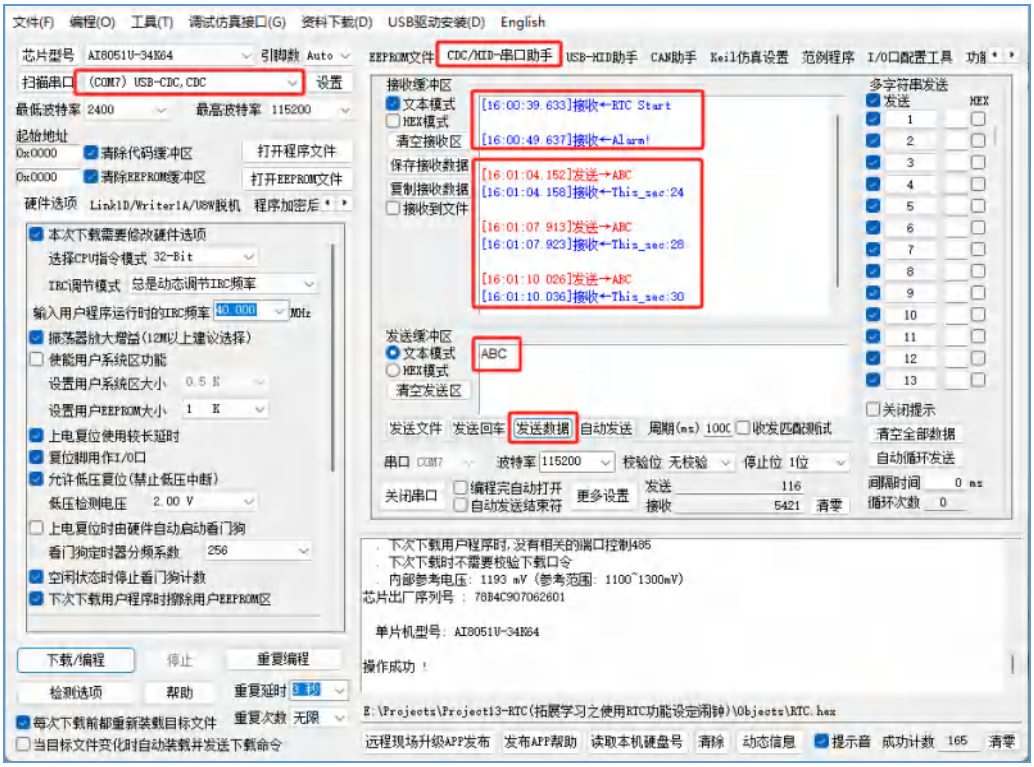
f) 程序实现效果

这时，在 AIapp-ISP 软件的 CDC 串口助手选项卡中找到“打开串口”：



点击“打开串口”

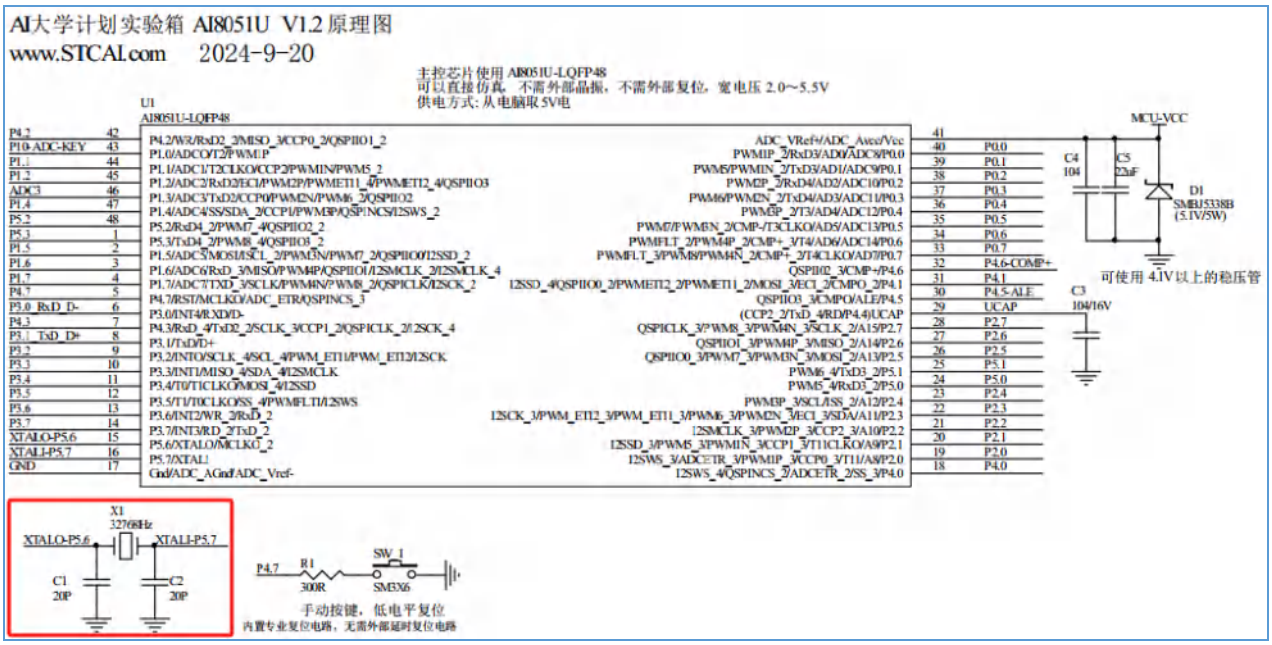
我们观察到：在 AIapp-ISP 软件的 CDC 串口助手窗口，会显示“RTC start”，10s 后显示及“Alarm！”。并显示发送的内容及发送当时的秒值。如下图：



达成实验预设，实验结束。

18.4.2　深大-使用【AiCube】助手实现 RTC 实验(测试 RTC，使用 USB-CDC 打印年/月/日/时/分/秒)

a) 原理图



b) 实验介绍

测试 Ai8051U 芯片的 RTC 功能，产生闹钟中断、天中断、小时中断、分钟中断、秒钟中断，并使用 AIapp-ISP 系统软件的 USB-CDC 功能打印中断数据及年/月/日、时/分/秒数据。

实验观察到的现象：

- 程序运行后，在 AIapp-ISP 系统软件的 USB-CDC 窗口显示闹钟中断，每秒/分/小时/天 产生的中断，并显示当前 RTC 的时间。

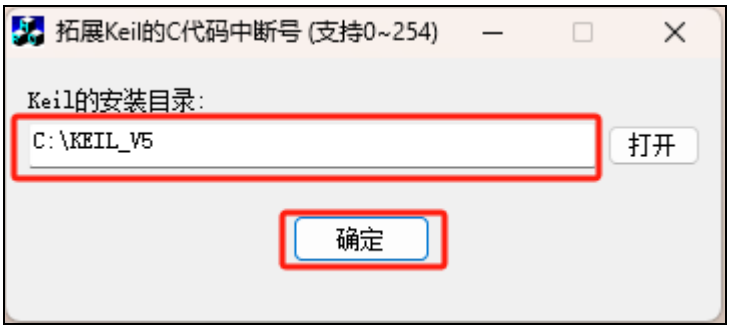
c) 准备工作--添加 Keil 中断拓展插件

Keil C51/C251 编译器只支持 31 以内的中断号（0~31），超过 31 编译会报错。RTC 的中断向量 RTC_VECTOR 为 36，这就需要 Keil 中断号拓展工具，可将中断号拓展到 254。

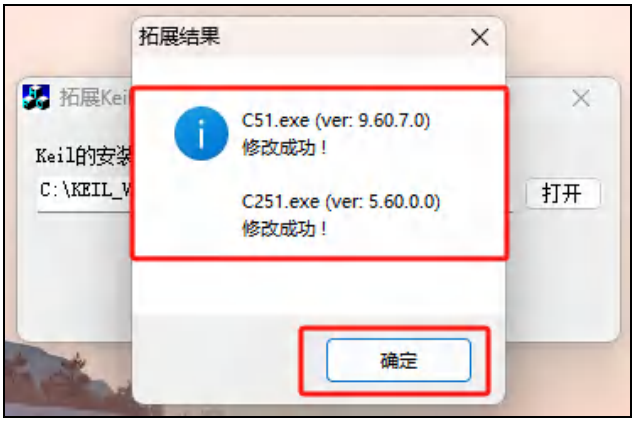
这部分可以到 STCAI 的官网：https://www.stcai.com/gjrj 的软件工具➔工具软件中“Keil 中断拓展插件”进行下载。

下载完成后，打开压缩包“拓展 Keil 的 C 代码中断号.zip”，找到“拓展 Keil 的 C 代码中断号.exe”，这是绿色免安装软件。

直接双击“拓展 Keil 的 C 代码中断号.exe”，出现如下界面，选择安装路径，按照默认直接安装到 C:\Keil_V5 目录下，点击“确定”



如下图，点击“确定”，Keil 的 C 代码中断号到 254



d) 安装 AIapp-ISP 下载/编程/烧录/软件

安装 STCAI 公司研发的 AIapp-ISP 下载 / 编程 / 烧录 / 软件，含强大的辅助开发工具包。登录 STCAI 官网，下载最新版的 AIapp-ISP 安装包，下载链接如下：https://www.stcai.com/gjrj

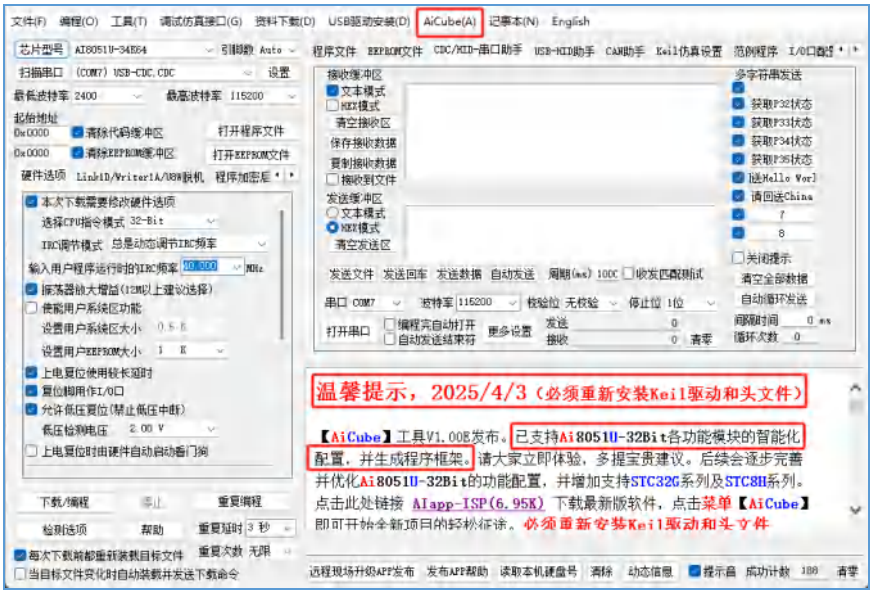
下载下来的是压缩包文件如下图：

双击后解压缩如下图：

如果是解压缩到非桌面的其他目录，如 D:\STCAI-ISP，建议创建快捷方式到桌面方便打开。这实际是一个免安装的绿色软件。

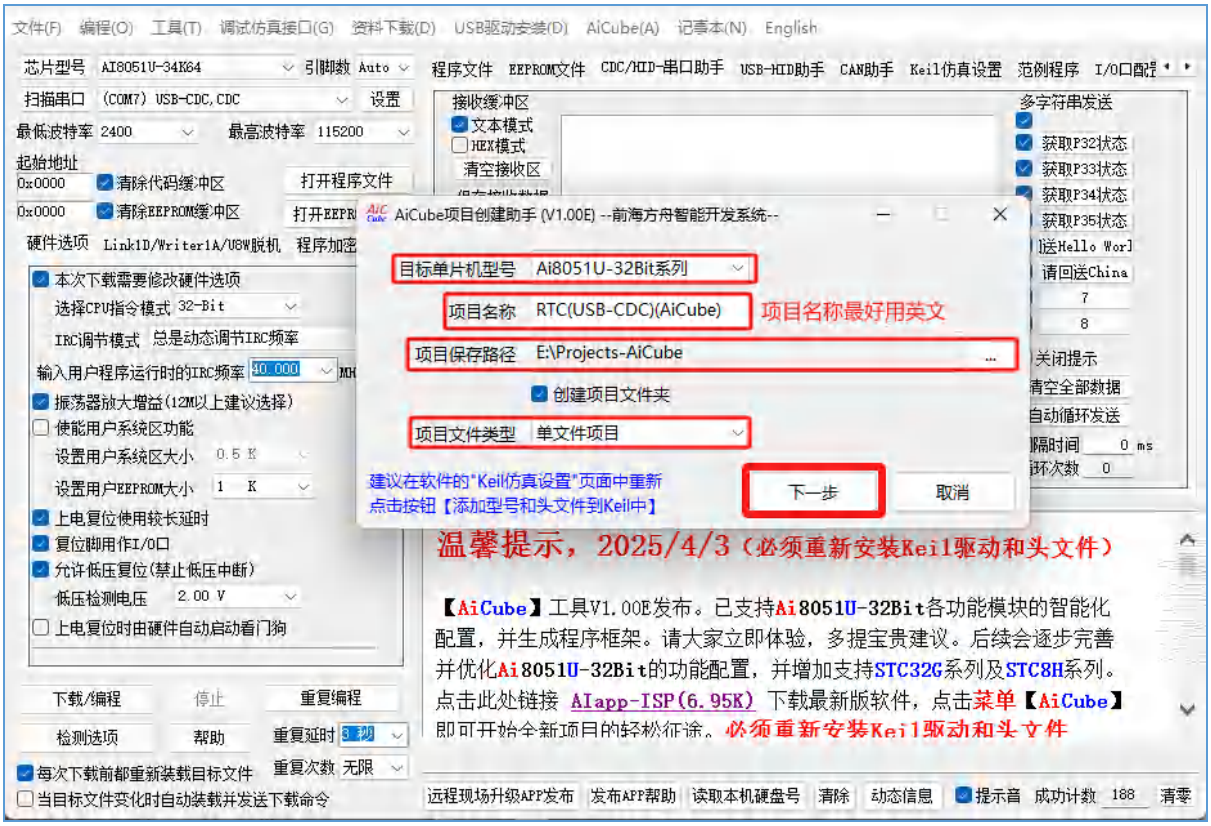
e) 打开【AiCube】助手

双击图标，即可打开 AIapp-ISP 软件界面，即可直接使用，如下图：

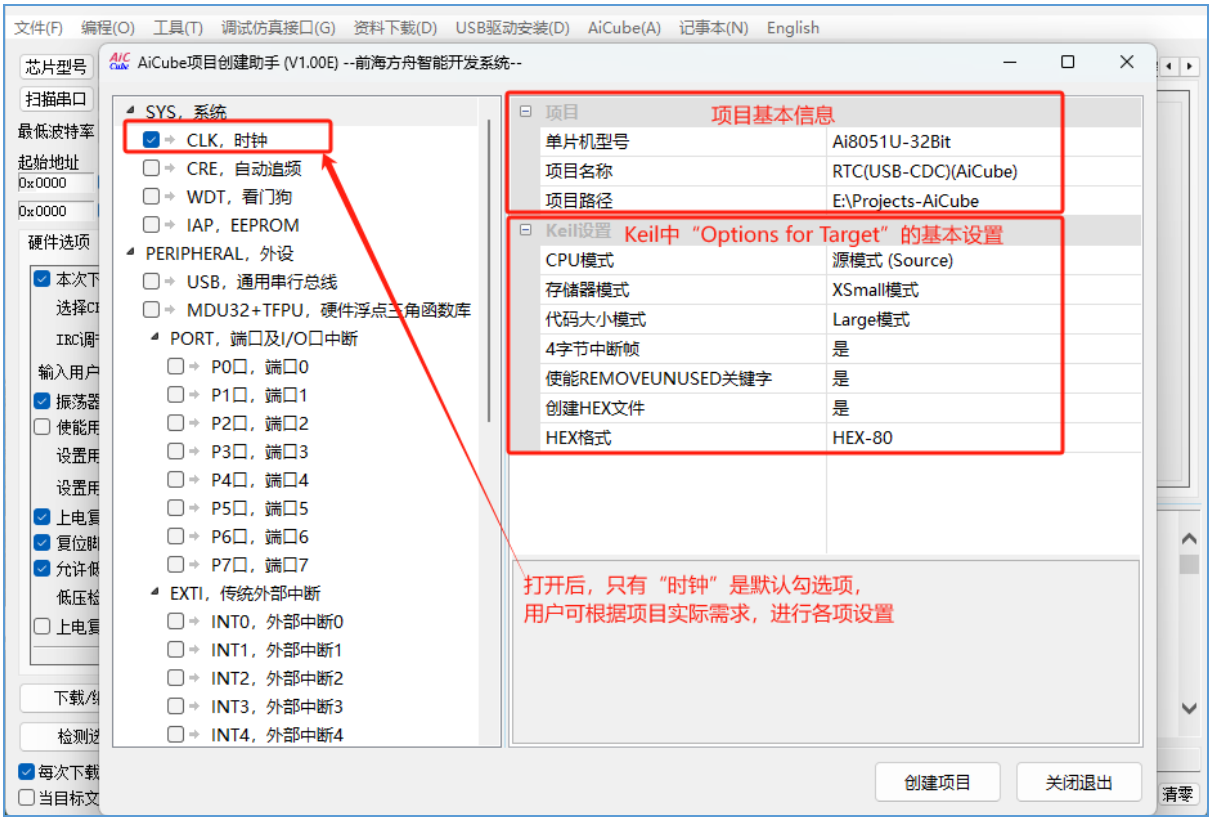


f) 使用【AiCube】助手创建项目

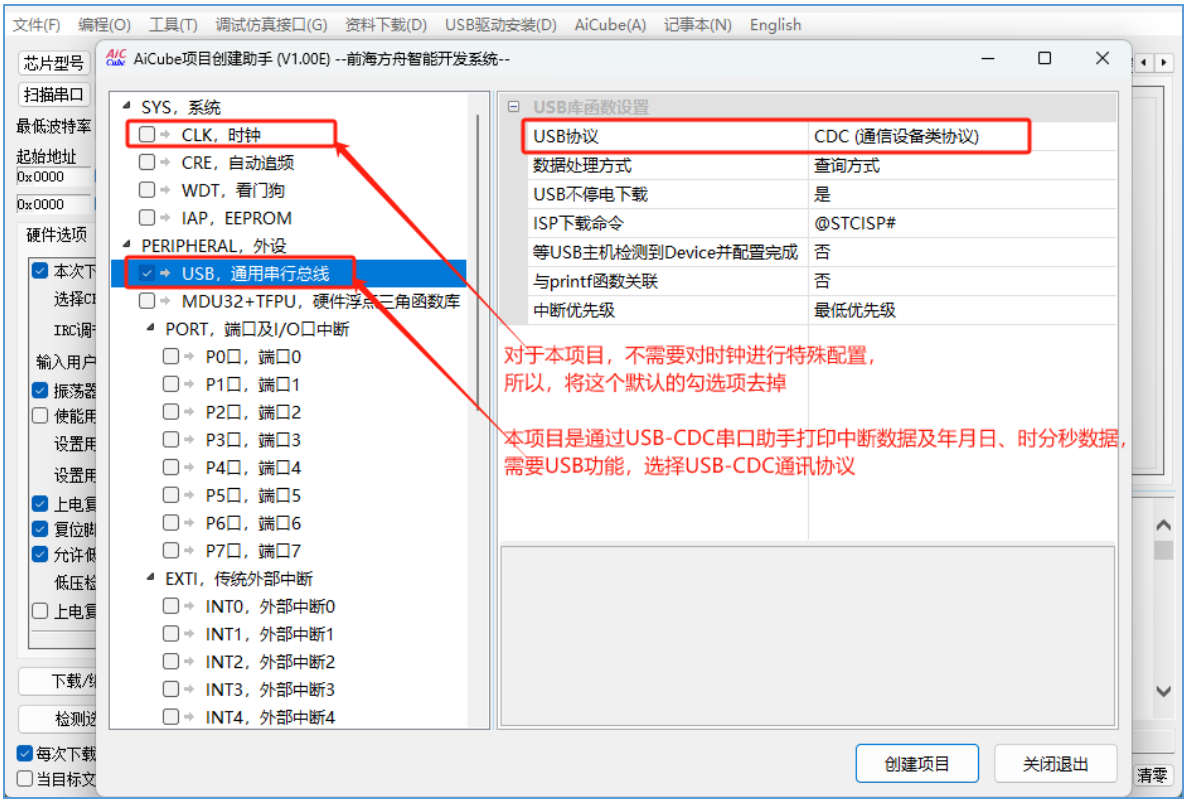
点击【AiCube（A）】，弹出如下窗口：



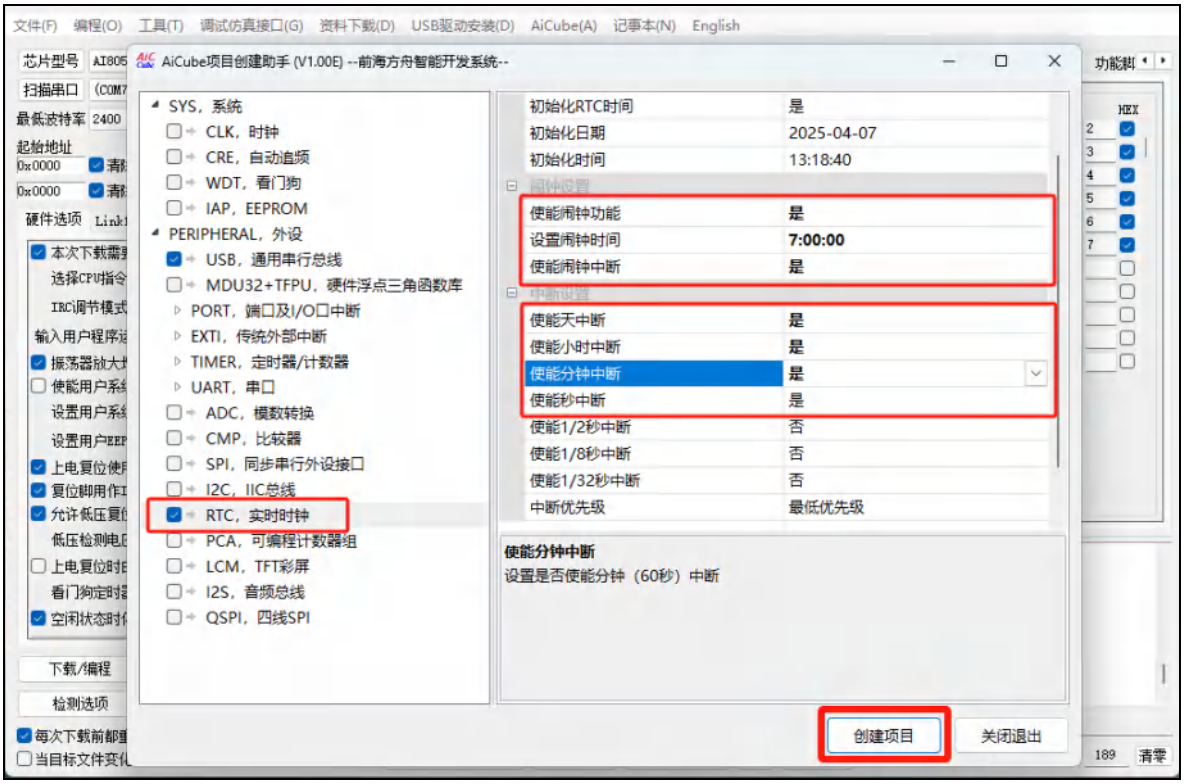
选择并填写项目基本信息，建议项目名称用英文！点击“下一步”，出现如下配置界面：



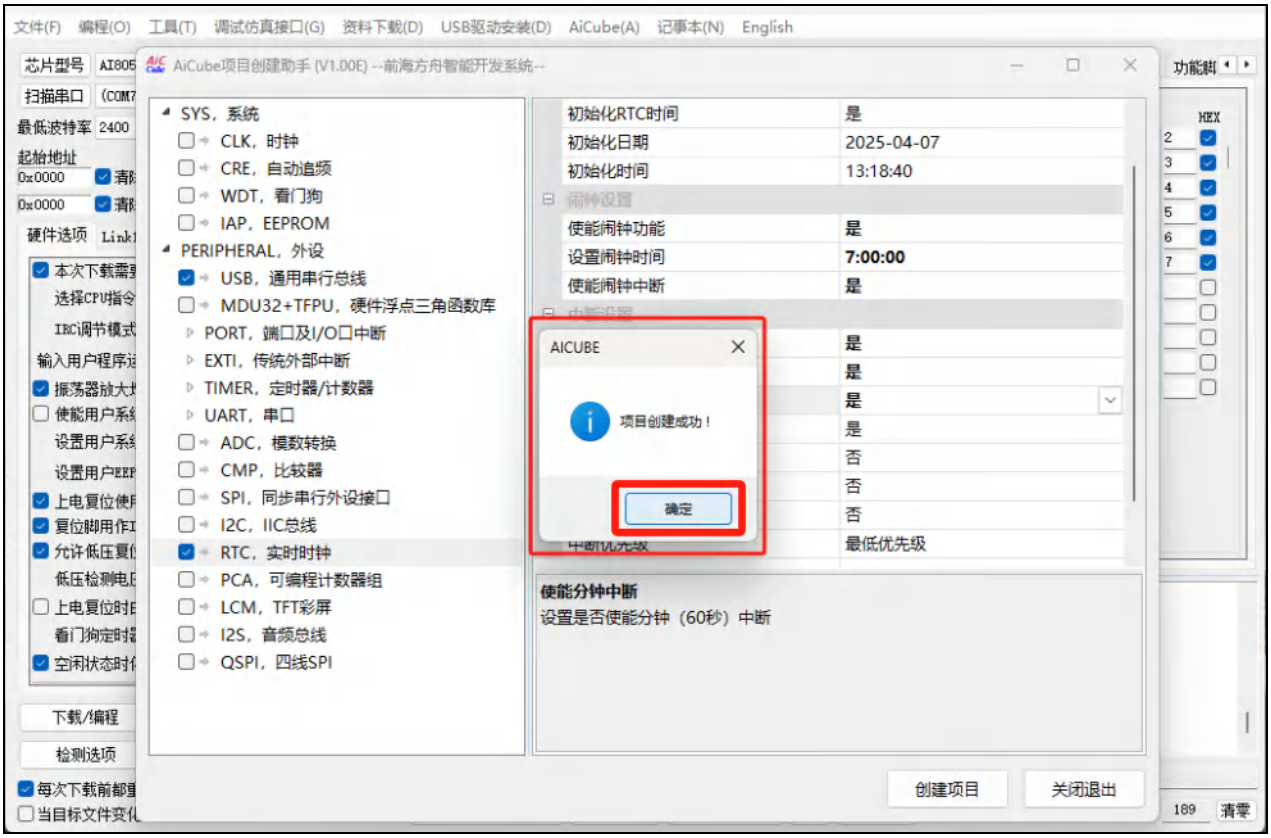
本实验是测试 Ai8051U 芯片的 RTC 功能，产生闹钟中断、天中断、小时中断、分钟中断、秒钟中断，并使用 USB-CDC 打印中断数据及年月日、时分秒数据。
显然要勾选 USB 选项，选择“CDC（通信设备类协议）”，如下图：



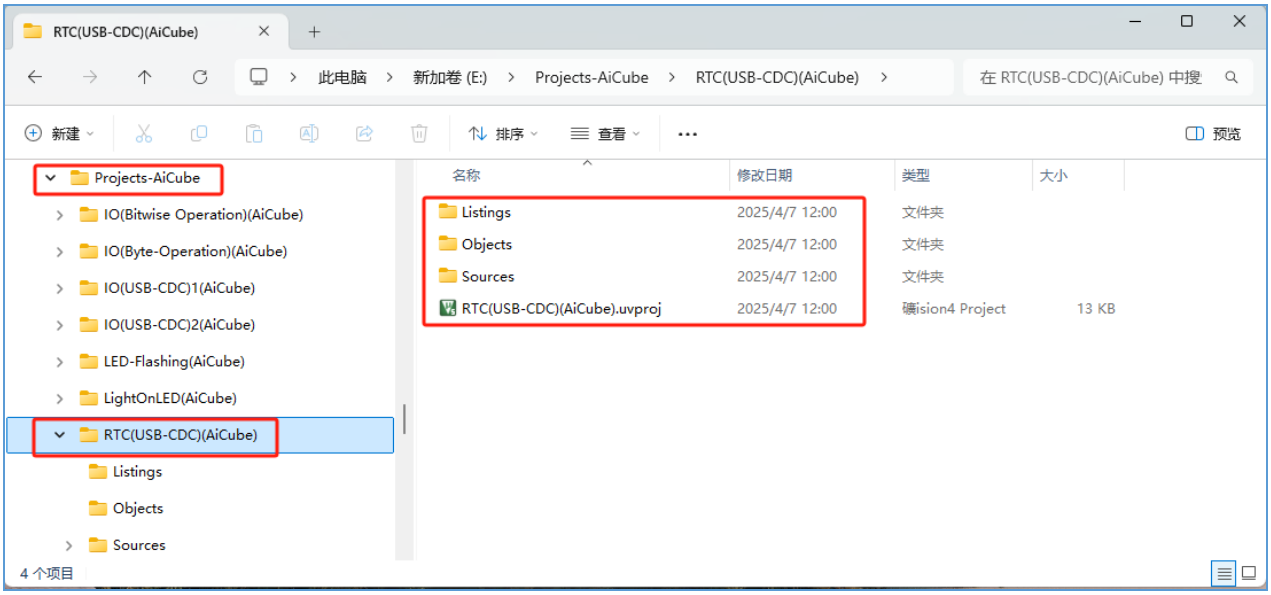
本实验是测试 Ai8051U 芯片的 RTC 功能，产生闹钟中断、天中断、小时中断、分钟中断、秒钟中断，并使用 USB-CDC 打印中断数据及年月日、时分秒数据。
当然要设置 RTC 模块，设置闹钟中断、天中断、小时中断、分钟中断、秒钟中断为“是”，如下图：



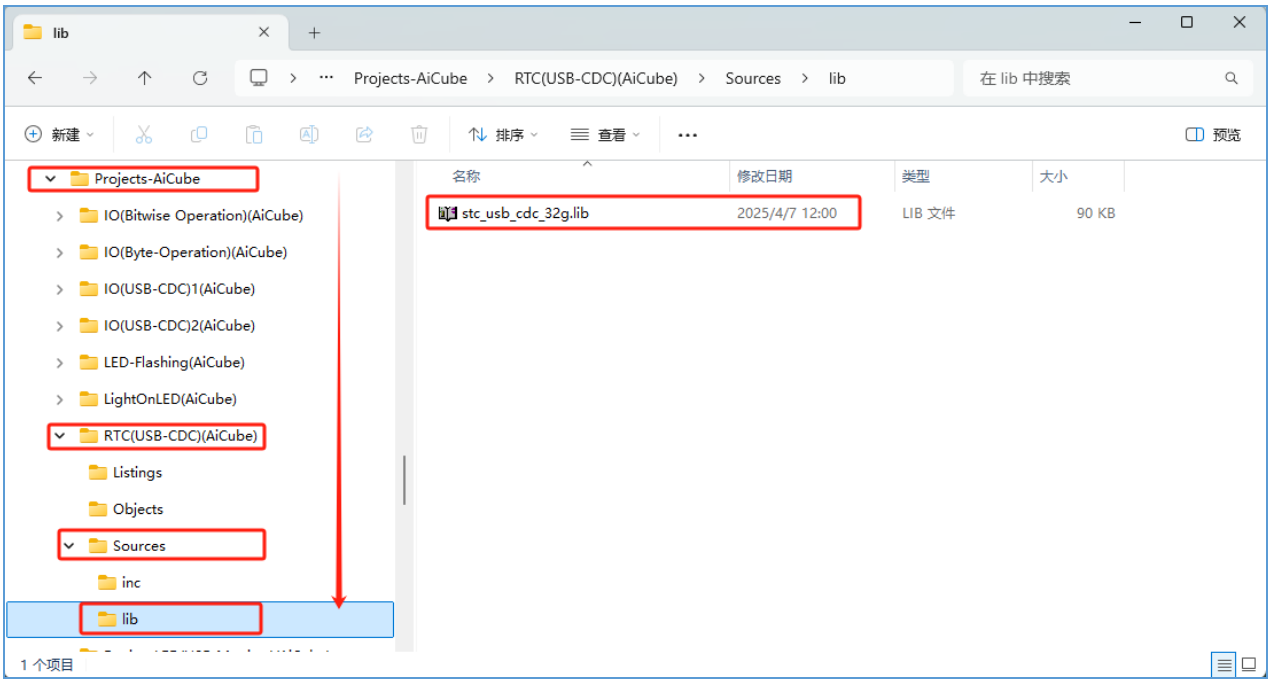
点击“创建项目”，【AiCube】助手提示：“项目创建成功！”，如下图：



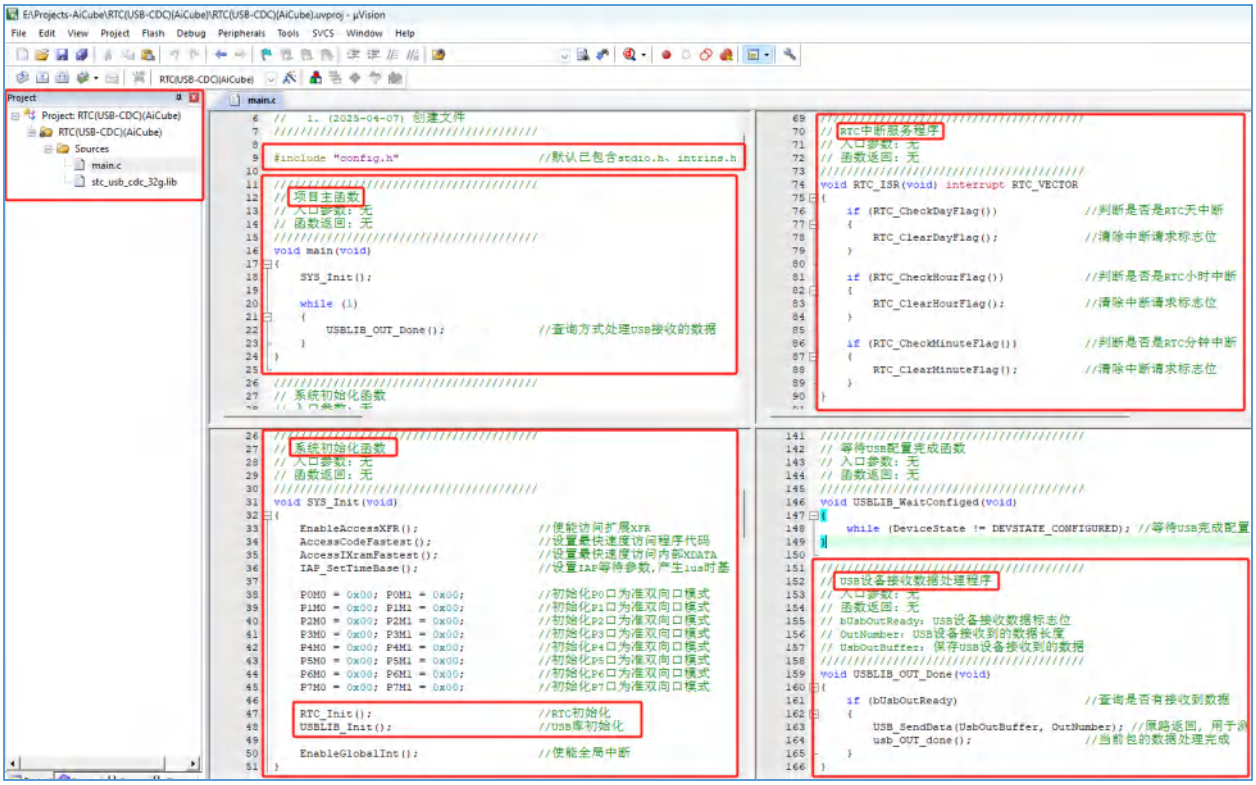
点击“确定”，即成功创建了项目及项目文件目录“RTC(USB-CDC)(AiCube)”，找到并打开刚刚创建的项目目录。发现刚刚的操作过程，【AiCube】助手已经帮我们建立了项目目录及 RTC(USB-CDC)(AiCube).uvproj 文件，如下图：



因为勾选了 USB 选项，打开 Sources>lib 子目录，发现，【AiCube】助手连 USB 库函数也已经加载了：



双击打开项目文件：RTC(USB-CDC)(AiCube).uvproj，如下图（为方便查看，使用菜单“Window”→“Split”将原窗口拆分成 4 个窗口，方便观察）：



根据在【AiCube】助手中的配置，从上图中我们看到【AiCube】助手创建的本实验项目中有了 main.c，还已经自动加载了 USB 库函数（stc_usb_cdc_32g.lib）。main.c 中按模块化设计，已经包含了：必要的头文件、主函数、系统初始化函数、延时函数、RTC 中断服务程序、RTC 初始化函数、USB 库初始化函数、等待 USB 配置完成函数、USB 设备接收数据处理程序。

g) 增加实验项目代码

为测试 Ai8051U 芯片的 RTC 功能，产生闹钟中断、天中断、小时中断、分钟中断、秒钟中断，并使用 USB-CDC 打印中断数据及年月日、时分秒数据。我们现在需要：

➲ 在主函数前定义中断标志位（闹钟中断、天中断、小时中断、分钟中断、秒钟中断）；

```
bit fAlarm = 0;          //RTC 闹钟查询标志，在中断程序中设置，主循环中查询并清除
bit fDay = 0;            //RTC 天查询标志，在中断程序中设置，主循环中查询并清除
bit fHour = 0;           //RTC 小时查询标志，在中断程序中设置，主循环中查询并清除
bit fMinute = 0;         //RTC 分钟查询标志，在中断程序中设置，主循环中查询并清除
bit fSecond = 0;         //RTC 秒钟查询标志，在中断程序中设置，主循环中查询并清除
```

➲ 在主函数的 while 循环中，加入 RTC 各种中断判断代码；

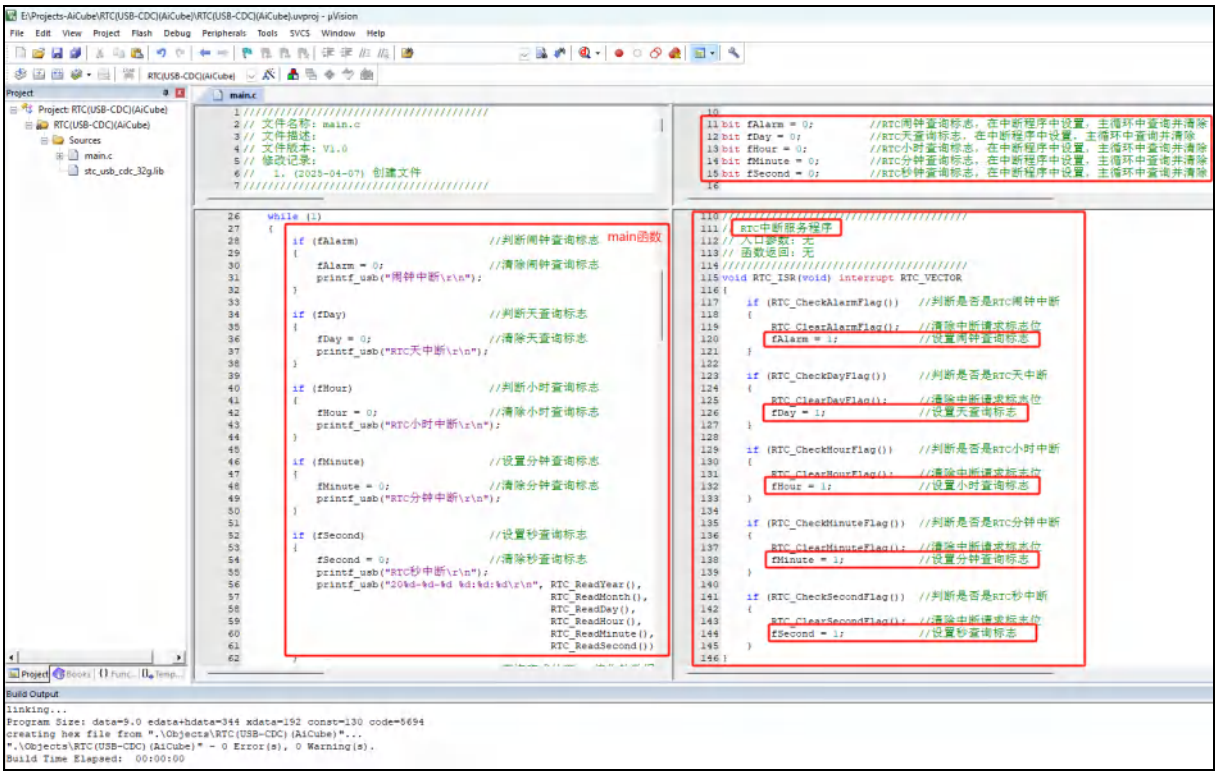
```
if (fAlarm)          //判断闹钟查询标志
{
    fAlarm = 0;    //清除闹钟查询标志
    printf_usb("闹钟中断\r\n");
}
if (fDay)            //判断天查询标志
{
    fDay = 0;      //清除天查询标志
    printf_usb("RTC 天中断\r\n");
}
if (fHour)           //判断小时查询标志
{
    fHour = 0;         //清除小时查询标志
    printf_usb("RTC 小时中断\r\n");
}
if (fMinute)         //设置分钟查询标志
{
    fMinute = 0;   //清除分钟查询标志
    printf_usb("RTC 分钟中断\r\n");
}
if (fSecond)         //设置秒查询标志
{
    fSecond = 0;       //清除秒查询标志
    printf_usb("RTC 秒中断\r\n");
    printf_usb("20%d-%d-%d %d:%d:%d\r\n",RTC_ReadYear(),
                                         RTC_ReadMonth(),
                                         RTC_ReadDay(),
                                         RTC_ReadHour(),
                                         RTC_ReadMinute(),
                                         RTC_ReadSecond());
}
```

➲ 在“RTC 中断服务程序”子函数中，补充设置中断标志的代码；

```
fAlarm = 1;          //设置闹钟查询标志
fDay = 1;            //设置天查询标志
```

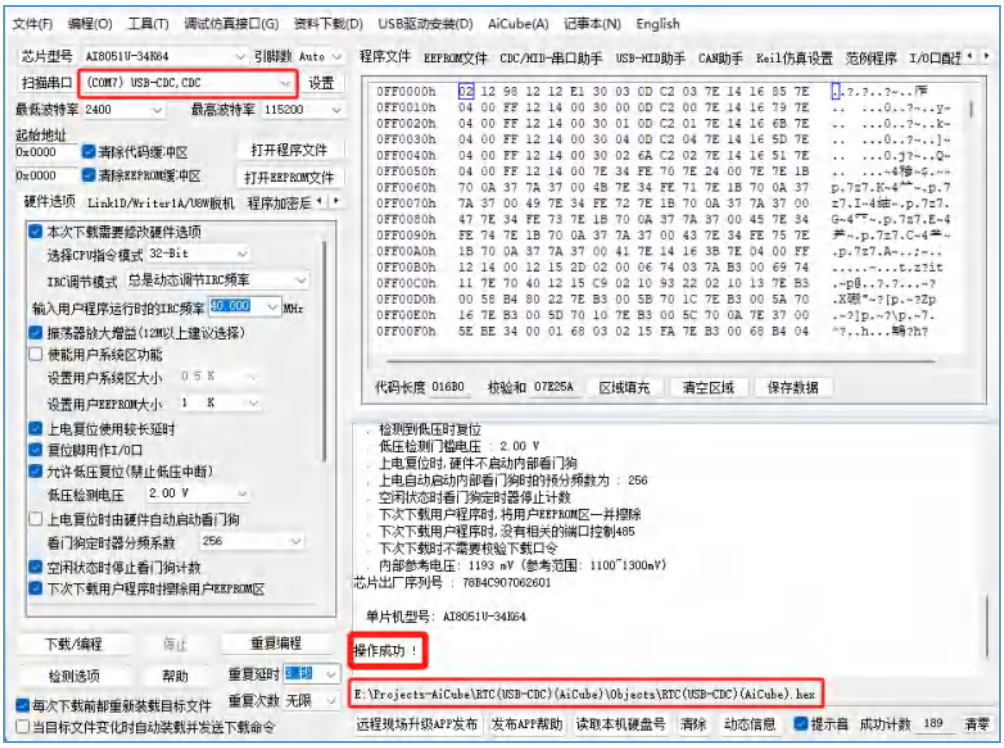
```
fHour = 1;          //设置小时查询标志
fMinute = 1;        //设置分钟查询标志
fSecond = 1;        //设置秒查询标志
```

➲ 保存，编译，如下图（使用菜单“Window”→“Split”将原窗口拆分成 4 个窗口，对比观察）：

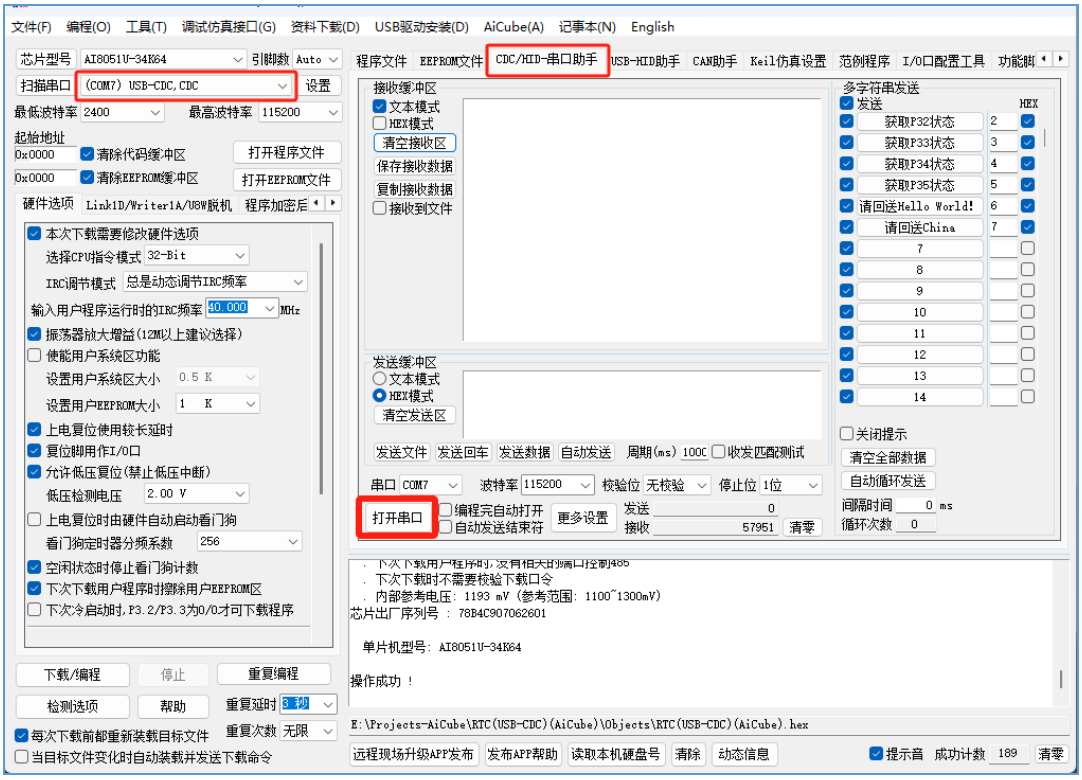


h) 使用 AIapp-ISP 软件系统并连接 Ai8051U 实验箱验证实验效果

连接好 Ai8051U 实验箱，运行 AIapp-ISP 软件系统，按步骤打开并下载程序“RTC(USB-CDC)(AiCube).hex”到实验箱，如下图：

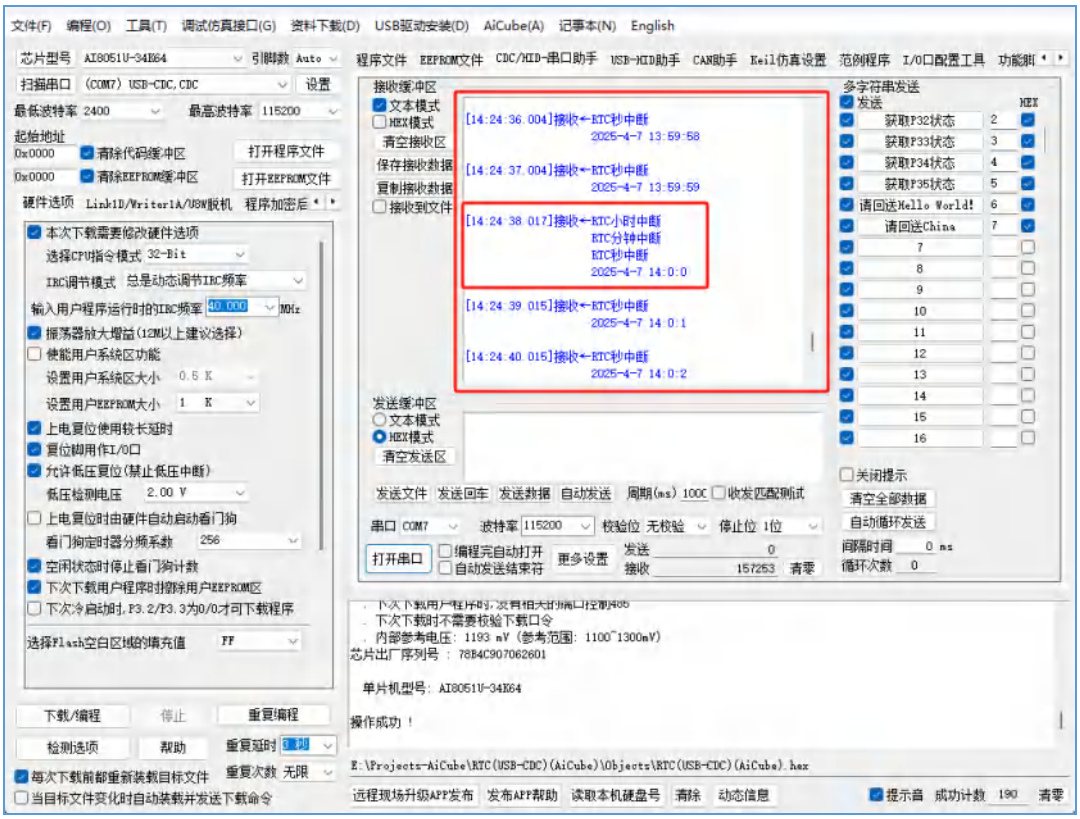


这时，在 AIapp-ISP 软件的 CDC 串口助手选项卡中找到“打开串口”按钮：



点击“打开串口”

我们观察到：在 AIapp-ISP 软件的 CDC 串口助手窗口，会显示年月日及时钟中断、分钟中断、秒钟中断。并显示发送的内容及发送当时的秒值。如下图：



达成实验预设，实验结束。

有了【AiCube】助手，单片机编程就是这么简单！快用起来吧！

19 同步/异步串口通信，硬件奇偶校验（USART1、USART2）

产品线	同步/异步串口数量
Ai8051U 系列	**2（U1~U2）**

Ai8051U 系列单片机具有 2 个全双工同步/异步串行通信接口（USART1 和 USART2）。每个串行口由 2 个数据缓冲器、一个移位寄存器、一个串行控制寄存器和一个波特率发生器等组成。每个串行口的数据缓冲器由 2 个互相独立的接收、发送缓冲器构成，可以同时发送和接收数据。

Ai8051U 系列单片机的串口 1、串口 2 均有 4 种工作方式，其中两种方式的波特率是可变的，另两种是固定的，以供不同应用场合选用。用户可用软件设置不同的波特率和选择不同的工作方式。主机可通过查询或中断方式对接收/发送进行程序处理，使用十分灵活。

串口 1、串口 2 的通讯口均可以通过功能管脚的切换功能切换到多组端口，从而可以将一个通讯口分时复用为多个通讯口。

19.1 串口功能脚切换

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW1	A2H	**S1_S[1:0]**		-		SPI_S[1:0]		-	

S1_S[1:0]：串口 1 功能脚选择位

S1_S[1:0]	RxD	TxD
00	P3.0	P3.1
01	P3.6	P3.7
10	P1.6	P1.7
11	P4.3	P4.4

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW2	BAH	EAXFR	-	I2C_S[1:0]		CMPO_S	S4_S	S3_S	S2_S

S2_S：串口 2 功能脚选择位

S2_S	RxD2	TxD2
0	P1.2	P1.3
1	P4.2	P4.3

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW3	BBH	I2S_S		S2SPI_S[1:0]		S1SPI_S[1:0]			

S2SPI_S[1:0]：USART2 的 SPI 功能脚选择位

S2SPI_S[1:0]	S2SS	S2MOSI	S2MISO	S2SCLK
00	P1.4	P1.5	P1.6	P1.7
01	P2.4	P2.5	P2.6	P2.7
10	P4.0	P4.1	P4.2	P4.3
11	P3.5	P3.4	P3.3	P3.2

S1SPI_S[1:0]：USART1 的 SPI 功能脚选择位

S1SPI_S[1:0]	S1SS	S1MOSI	S1MISO	S1SCLK
00	P1.4	P1.5	P1.6	P1.7
01	P2.4	P2.5	P2.6	P2.7
10	P4.0	P4.1	P4.2	P4.3
11	P3.5	P3.4	P3.3	P3.2

19.2 串口相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
SCON	串口 1 控制寄存器	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
SBUF	串口 1 数据寄存器	99H									0000,0000
S2CON	串口 2 控制寄存器	9AH	S2SM0/FE	S2SM1	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0000,0000
S2BUF	串口 2 数据寄存器	9BH									0000,0000
PCON	电源控制寄存器	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
AUXR	辅助寄存器 1	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1BRT	0000,0001
SADDR	串口 1 从机地址寄存器	A9H									0000,0000
SADEN	串口 1 从机地址屏蔽寄存器	B9H									0000,0000

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
S2CFG	串口 2 配置寄存器	7EFDB4H	-	S2MOD0	S2M0x6	-	-	-	-	W1	000x,xxx0
S2ADDR.	串口 2 从机地址寄存器	7EFDB5H									0000,0000
S2ADEN	串口 2 从机地址屏蔽寄存器	7EFDB6H									0000,0000
USARTCR1	串口 1 控制寄存器 1	7EFDC0H	LINEN	DORD	CLKEN	SPMOD	SPIEN	SPSLV	CPOL	CPHA	0000,0000
USARTCR2	串口 1 控制寄存器 2	7EFDC1H	IREN	IRLP	SCEN	NACK	HDSEL	PCEN	PS	PE	0000,0000
USARTCR3	串口 1 控制寄存器 3	7EFDC2H	IrDA_LPBAUD[7:0]								0000,0111
USARTCR4	串口 1 控制寄存器 4	7EFDC3H	-	-	-	-	SCCKS[1:0]		SPICKS[1:0]		xxxx,0000
USARTCR5	串口 1 控制寄存器 5	7EFDC4H	BRKDET	HDRER	SLVEN	ASYNC	TXCF	SENDBK	HDRDET	SYNC	0000,0000
USARTGTR	串口 1 保护时间寄存器	7EFDC5H									0000,0000
USARTBRH	串口 1 波特率寄存器	7EFDC6H	USARTBR[15:8]								0000,0000
USARTBRL	串口 1 波特率寄存器	7EFDC7H	USARTBR[7:0]								0000,0000
USART2CR1	串口 2 控制寄存器 1	7EFDC8H	LINEN	DORD	CLKEN	SPMOD	SPIEN	SPSLV	CPOL	CPHA	0000,0000
USART2CR2	串口 2 控制寄存器 2	7EFDC9H	IREN	IRLP	SCEN	NACK	HDSEL	PCEN	PS	PE	0000,0000
USART2CR3	串口 2 控制寄存器 3	7EFDCAH	IrDA_LPBAUD[7:0]								0000,0000
USART2CR4	串口 2 控制寄存器 4	7EFDCBH	-	-	-	-	SCCKS[1:0]		SPICKS[1:0]		xxxx,0000
USART2CR5	串口 2 控制寄存器 5	7EFDCCH	BRKDET	HDRER	SLVEN	ASYNC	TXCF	SENDBK	HDRDET	SYNCAN	0000,0000
USART2GTR	串口 2 保护时间寄存器	7EFDCDH									0000,0000
USART2BRH	串口 2 波特率寄存器	7EFDCEH	USART2BR[15:8]								0000,0000
USART2BRL	串口 2 波特率寄存器	7EFDCFH	USART2BR[7:0]								0000,0000

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
UR1TOCR	串口 1 接收超时控制寄存器	7EFD70H	ENTO	ENTOI	SCALE	-	-	-	-	-	000x,xxxx
UR1TOSR	串口 1 接收超时状态寄存器	7EFD71H	CTOIF	-	-	-	-	-	-	TOIF	0xxx,xxx0
UR1TOTH	串口 1 接收超时长度寄存器	7EFD72H	TM[15:8]								0000,0000
UR1TOTL	串口 1 接收超时长度寄存器	7EFD73H	TM[7:0]								0000,0000
UR2TOCR	串口 2 接收超时控制寄存器	7EFD74H	ENTO	ENTOI	SCALE	-	-	-	-	-	000x,xxxx
UR2TOSR	串口 2 接收超时状态寄存器	7EFD75H	CTOIF	-	-	-	-	-	-	TOIF	0xxx,xxx0
UR2TOTH	串口 2 接收超时长度寄存器	7EFD76H	TM[15:8]								0000,0000
UR2TOTL	串口 2 接收超时长度寄存器	7EFD77H	TM[7:0]								0000,0000
UR1TOTE	串口 1 接收超时长度寄存器	7EFD88H	TM[23:16]								0000,0000
UR2TOTE	串口 2 接收超时长度寄存器	7EFD89H	TM[23:16]								0000,0000

19.3 串口 1（同步/异步串口 USART）

19.3.1 串口 1 控制寄存器（SCON）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SCON	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI

SM0/FE：当PCON寄存器中的SMOD0位为1时，该位为帧错误检测标志位。当UART在接收过程中检测到一个无效停止位时，通过UART接收器将该位置1，必须由软件清零。当PCON寄存器中的SMOD0位为0时，该位和SM1一起指定串口1的通信工作模式，如下表所示：

SM0	SM1	串口1工作模式	功能说明
0	0	模式0	同步移位串行方式
0	1	模式1	可变波特率8位数据方式
1	0	模式2	固定波特率9位数据方式
1	1	模式3	可变波特率9位数据方式

SM2：允许模式 2 或模式 3 多机通信控制位。当串口 1 使用模式 2 或模式 3 时，如果 SM2 位为 1 且 REN 位为 1，则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位（即 RB8）来筛选地址帧，若 RB8=1，说明该帧是地址帧，地址信息可以进入 SBUF，并使 RI 为 1，进而在中断服务程序中再进行地址号比较；若 RB8=0，说明该帧不是地址帧，应丢掉且保持 RI=0。在模式 2 或模式 3 中，如果 SM2 位为 0 且 REN 位为 1，接收收机处于地址帧筛选被禁止状态，不论收到的 RB8 为 0 或 1，均可使接收到的信息进入 SBUF，并使 RI=1，此时 RB8 通常为校验位。模式 1 和模式 0 为非多机通信方式，在这两种方式时，SM2 应设置为 0。

REN：允许/禁止串口接收控制位

0：禁止串口接收数据

1：允许串口接收数据

TB8：当串口 1 使用模式 2 或模式 3 时，TB8 为要发送的第 9 位数据，按需要由软件置位或清 0。在模式 0 和模式 1 中，该位不用。

RB8：当串口 1 使用模式 2 或模式 3 时，RB8 为接收到的第 9 位数据，一般用作校验位或者地址帧/数据帧标志位。在模式 0 和模式 1 中，该位不用。

TI：串口 1 发送中断请求标志位。在模式 0 中，当串口发送数据第 8 位结束时，由硬件自动将 TI 置 1，向主机请求中断，响应中断后 TI 必须用软件清零。在其他模式中，则在停止位开始发送时由硬件自动将 TI 置 1，向 CPU 发请求中断，响应中断后 TI 必须用软件清零。

RI：串口 1 接收中断请求标志位。在模式 0 中，当串口接收第 8 位数据结束时，由硬件自动将 RI 置 1，向主机请求中断，响应中断后 RI 必须用软件清零。在其他模式中，串行接收到停止位的中间时刻由硬件自动将 RI 置 1，向 CPU 发中断申请，响应中断后 RI 必须由软件清零。

19.3.2 串口 1 数据寄存器（SBUF）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SBUF	99H								

SBUF：串口 1 数据接收/发送缓冲区。SBUF 实际是 2 个缓冲器，读缓冲器和写缓冲器，两个操作分别对应两个不同的寄存器，1 个是只写寄存器（写缓冲器），1 个是只读寄存器（读缓冲器）。对 SBUF 进行读操作，实际是读取串口接收缓冲区，对 SBUF 进行写操作则是触发串口开始发送数据。

19.3.3 电源管理寄存器（PCON）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PCON	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL

SMOD：串口 1 波特率控制位

0：串口 1 的各个模式的波特率都不加倍

1：串口 1 模式 1（定时器 1 的模式 2 作为波特率发生器时有效）、模式 2、模式 3（定时器 1 的模式 2 作为波特率发生器时有效）的波特率加倍

SMOD0：帧错误检测控制位

0：无帧错检测功能

1：使能帧错误检测功能。此时 SCON 的 SM0/FE 为 FE 功能，即为帧错误检测标志位。

19.3.4 辅助寄存器 1（AUXR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1BRT

UART_M0x6：串口 1 模式 0 的通讯速度控制

0：串口 1 模式 0 的波特率不加倍，固定为 Fosc/12

1：串口 1 模式 0 的波特率 6 倍速，即固定为 Fosc/12*6 ＝ Fosc/2

S1BRT：串口 1 波特率发生器选择位

0：选择定时器 1 作为波特率发生器

1：选择定时器 2 作为波特率发生器**（默认值）**

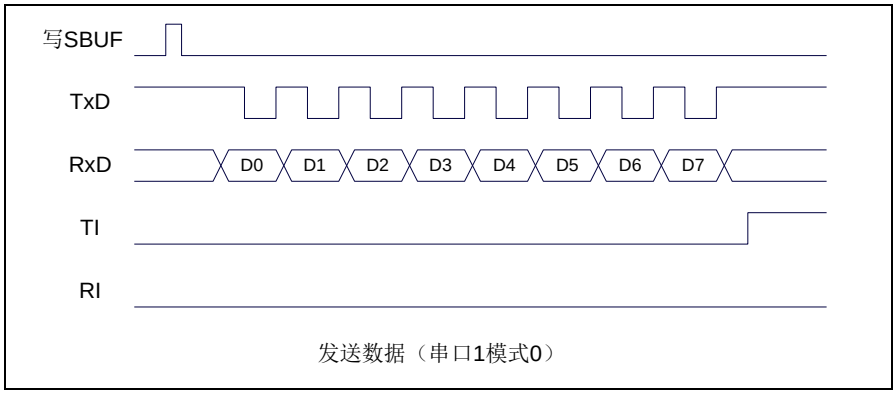
注意：串口 1 默认是使用定时器 2 做波特率发生器，不建议使用定时器 1。定时器 2 可同时共享作为串口 1、串口 2、串口 3 和串口 4 的波特率发生器

19.3.5　串口 1 模式 0，模式 0 波特率计算公式

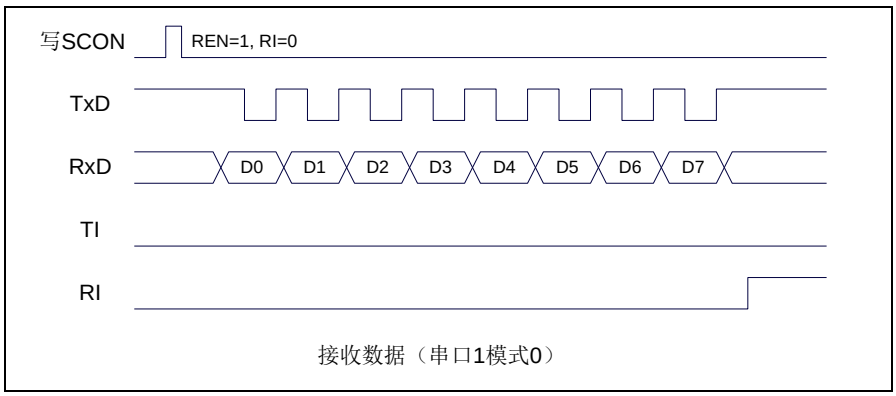
当串口 1 选择工作模式为模式 0 时，串行通信接口工作在同步移位寄存器模式，当串行口模式 0 的通信速度设置位 UART_M0x6 为 0 时，其波特率固定为系统时钟时钟的 12 分频（SYSclk/12）；当设置 UART_M0x6 为 1 时，其波特率固定为系统时钟频率的 2 分频（SYSclk/2）。RxD 为串行通讯的数据口，TxD 为同步移位脉冲输出脚，发送、接收的是 8 位数据，低位在先。

模式 0 的发送过程：当主机执行将数据写入发送缓冲器 SBUF 指令时启动发送，串行口即将 8 位数据以 SYSclk/12 或 SYSclk/2（由 UART_M0x6 确定是 12 分频还是 2 分频）的波特率从 RxD 管脚输出(从低位到高位)，发送完中断标志 TI 置 1，TxD 管脚输出同步移位脉冲信号。当写信号有效后，相隔一个时钟，发送控制端 SEND 有效(高电平)，允许 RxD 发送数据，同时允许 TxD 输出同步移位脉冲。一帧(8 位)数据发送完毕时，各控制端均恢复原状态，只有 TI 保持高电平，呈中断申请状态。在再次发送数据前，必须用软件将 TI 清 0。

模式 0 的接收过程：首先将接收中断请求标志 RI 清零并置位允许接收控制位 REN 时启动模式 0 接收过程。启动接收过程后，RxD 为串行数据输入端，TxD 为同步脉冲输出端。串行接收的波特率为 SYSclk/12 或 SYSclk/2（由 UART_M0x6 确定是 12 分频还是 2 分频）。当接收完成一帧数据（8 位）后，控制信号复位，中断标志 RI 被置 1，呈中断申请状态。当再次接收时，必须通过软件将 RI 清 0



发送数据（串口1模式0）



接收数据（串口1模式0）

工作于模式 0 时，必须清 0 多机通信控制位 SM2，使之不影响 TB8 位和 RB8 位。由于波特率固定为 SYSclk/12 或 SYSclk/2，无需定时器提供，直接由单片机的时钟作为同步移位脉冲。

串口 1 模式 0 的波特率计算公式如下表所示（SYSclk 为系统工作频率）：

UART_M0x6	波特率计算公式
0	波特率 $= \frac{\text{SYSclk}}{12}$
1	波特率 $= \frac{\text{SYSclk}}{2}$

19.3.6　串口 1 模式 1，模式 1 波特率计算公式

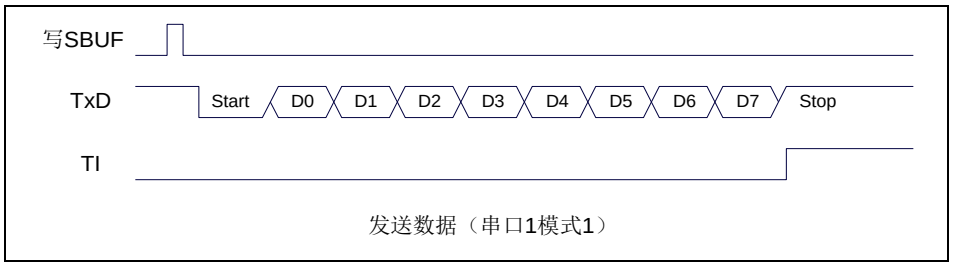
当软件设置 SCON 的 SM0、SM1 为“01”时，串行口 1 则以模式 1 进行工作。此模式为 8 位 UART 格式，一帧信息为 10 位：1 位起始位，8 位数据位（低位在先）和 1 位停止位。波特率可变，即可根据需要进行设置波特率。TxD 为数据发送口，RxD 为数据接收口，串行口全双工接受/发送。

模式 1 的发送过程：串行通信模式发送时，数据由串行发送端 TxD 输出。当主机执行一条写 SBUF 的指令就启动串行通信的发送，写“SBUF”信号还把“1”装入发送移位寄存器的第 9 位，并通知 TX 控制单元开始发送。移位寄存器将数据不断右移送 TxD 端口发送，在数据的左边不断移入“0”作补充。当数据的最高位移到移位寄存器的输出位置，紧跟其后的是第 9 位“1”，在它的左边各位全为“0”，这个状态条件，使 TX 控制单元作最后一次移位输出，然后使允许发送信号“SEND”失效，完成一帧信息的发送，并置位中断请求位 TI，即 TI=1，向主机请求中断处理。

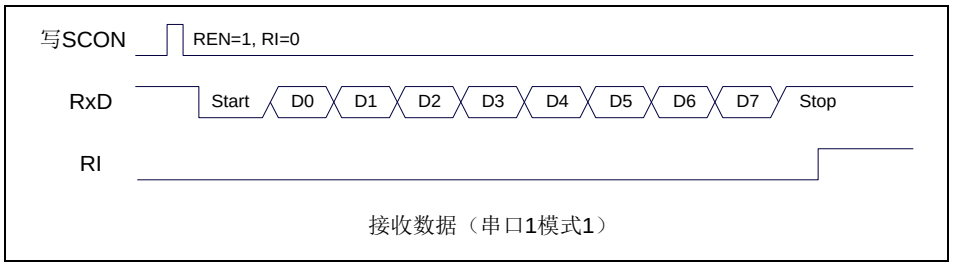
模式 1 的接收过程：当软件置位接收允许标志位 REN，即 REN=1 时，接收器便对 RxD 端口的信号进行检测，当检测到 RxD 端口发送从“1”→“0”的下降沿跳变时就启动接收器准备接收数据，并立即复位波特率发生器的接收计数器，将 1FFH 装入移位寄存器。接收的数据从接收移位寄存器的右边移入，已装入的 1FFH 向左边移出，当起始位"0"移到移位寄存器的最左边时，使 RX 控制器作最后一次移位，完成一帧的接收。若同时满足以下两个条件：

- RI=0；
- SM2=0 或接收到的停止位为 1。

则接收到的数据有效，实现装载入 SBUF，停止位进入 RB8，RI 标志位被置 1，向主机请求中断，若上述两条件不能同时满足，则接收到的数据作废并丢失，无论条件满足与否，接收器重又检测 RxD 端口上的"1"→"0"的跳变，继续下一帧的接收。接收有效，在响应中断后，RI 标志位必须由软件清 0。通常情况下，串行通信工作于模式 1 时，SM2 设置为"0"。



发送数据（串口1模式1）



接收数据（串口1模式1）

串口 1 的波特率是可变的，其波特率可由定时器 1 或者定时器 2 产生。当定时器采用 1T 模式时（12 倍速），相应的波特率的速度也会相应提高 12 倍。

串口 1 模式 1 的波特率计算公式如下表所示：（SYSclk 为系统工作频率）

选择定时器	定时器速度	重装值计算公式	波特率
定时器2	1T	定时器2重载值 = $65536 - \frac{SYSclk}{4\times波特率}$	波特率 = $\frac{SYSclk}{4\times(65536 - 定时器重装数)}$
	12T	定时器2重载值 = $65536 - \frac{SYSclk}{12\times4\times波特率}$	波特率 = $\frac{SYSclk}{12\times4\times(65536 - 定时器重装数)}$
定时器1模式0	1T	定时器1重载值 = $65536 - \frac{SYSclk}{4\times波特率}$	波特率 = $\frac{SYSclk}{4\times(65536 - 定时器重装数)}$
	12T	定时器1重载值 = $65536 - \frac{SYSclk}{12\times4\times波特率}$	波特率 = $\frac{SYSclk}{12\times4\times(65536 - 定时器重装数)}$
定时器1模式2	1T	定时器1重载值 = $256 - \frac{2^{SMOD}\times SYSclk}{32\times波特率}$	波特率 = $\frac{2^{SMOD}\times SYSclk}{32\times(256 - 定时器重装数)}$
	12T	定时器1重载值 = $256 - \frac{2^{SMOD}\times SYSclk}{12\times32\times波特率}$	波特率 = $\frac{2^{SMOD}\times SYSclk}{12\times32\times(256 - 定时器重装数)}$

下面为常用频率与常用波特率所对应定时器的重载值

频率(MHz)	波特率	定时器 2		定时器 1 模式 0		定时器 1 模式 2			
						SMOD=1		SMOD=0	
		1T 模式	12T 模式	1T 模式	12T 模式	1T 模式	12T 模式	1T 模式	12T 模式
11.0592	115200	FFE8H	FFFEH	FFE8H	FFFEH	FAH	-	FDH	-
	57600	FFD0H	FFFCH	FFD0H	FFFCH	F4H	FFH	FAH	-
	38400	FFB8H	FFFAH	FFB8H	FFFAH	EEH	-	F7H	-
	19200	FF70H	FFF4H	FF70H	FFF4H	DCH	FDH	EEH	-
	9600	FEE0H	FFE8H	FEE0H	FFE8H	B8H	FAH	DCH	FDH
18.432	115200	FFD8H	-	FFD8H	-	F6H	-	FBH	-
	57600	FFB0H	-	FFB0H	-	ECH	-	F6H	-
	38400	FF88H	FFF6H	FF88H	FFF6H	E2H	-	F1H	-
	19200	FF10H	FFECH	FF10H	FFECH	C4H	FBH	E2H	-
	9600	FE20H	FFD8H	FE20H	FFD8H	88H	F6H	C4H	FBH
22.1184	115200	FFD0H	FFFCH	FFD0H	FFFCH	F4H	FFH	FAH	-
	57600	FFA0H	FFF8H	FFA0H	FFF8H	E8H	FEH	F4H	FFH
	38400	FF70H	FFF4H	FF70H	FFF4H	DCH	FDH	EEH	-
	19200	FEE0H	FFE8H	FEE0H	FFE8H	B8H	FAH	DCH	FDH
	9600	FDC0H	FFD0H	FDC0H	FFD0H	70H	F4H	B8H	FAH

19.3.7　串口 1 模式 2，模式 2 波特率计算公式

当 SM0、SM1 两位为 10 时，串行口 1 工作在模式 2。串行口 1 工作模式 2 为 9 位数据异步通信 UART 模式，其一帧的信息由 11 位组成：1 位起始位，8 位数据位（低位在先），1 位可编程位（第 9 位数据）和 1 位停止位。发送时可编程位（第 9 位数据）由 SCON 中的 TB8 提供，可软件设置为 1 或 0，或者可将 PSW 中的奇/偶校验位 P 值装入 TB8（TB8 既可作为多机通信中的地址数据标志位，又可作为数据的奇偶校验位）。接收时第 9 位数据装入 SCON 的 RB8。TxD 为发送端口，RxD 为接收端口，以全双工模式进行接收/发送。

模式 2 的波特率固定为系统时钟的 64 分频或 32 分频（取决于 PCON 中 SMOD 的值）

串口 1 模式 2 的波特率计算公式如下表所示（SYSclk 为系统工作频率）：

SMOD	波特率计算公式
0	波特率 $= \frac{SYSclk}{64}$
1	波特率 $= \frac{SYSclk}{32}$

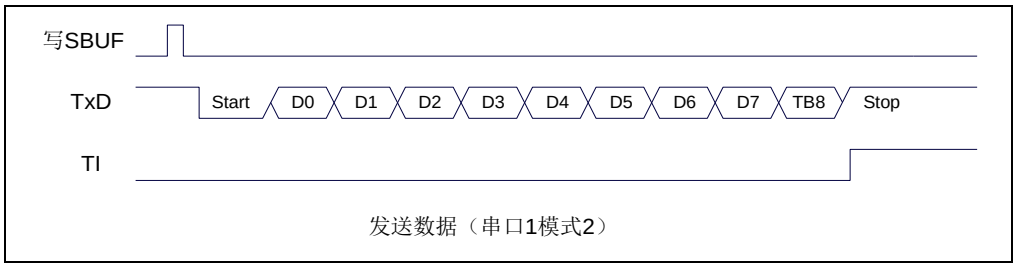
模式 2 和模式 1 相比，除波特率发生源略有不同，发送时由 TB8 提供给移位寄存器第 9 数据位不同外，其余功能结构均基本相同，其接收/发送操作过程及时序也基本相同。

当接收器接收完一帧信息后必须同时满足下列条件：

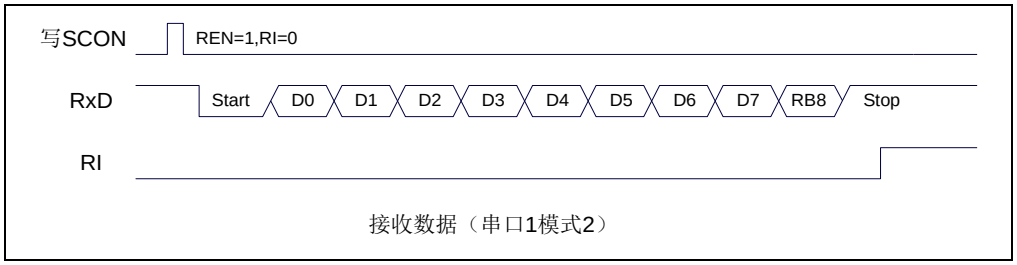
- RI=0
- SM2=0 或者 SM2=1 且接收到的第 9 数据位 RB8=1。

当上述两条件同时满足时，才将接收到的移位寄存器的数据装入 SBUF 和 RB8 中，RI 标志位被置 1，并向主机请求中断处理。如果上述条件有一个不满足，则刚接收到移位寄存器中的数据无效而丢失，也不置位 RI。无论上述条件满足与否，接收器又重新开始检测 RxD 输入端口的跳变信息，接收下一帧的输入信息。在模式 2 中，接收到的停止位与 SBUF、RB8 和 RI 无关。

通过软件对 SCON 中的 SM2、TB8 的设置以及通信协议的约定，为多机通信提供了方便。



发送数据（串口1模式2）



接收数据（串口1模式2）

19.3.8　串口 1 模式 3，模式 3 波特率计算公式

当 SM0、SM1 两位为 11 时，串行口 1 工作在模式 3。串行通信模式 3 为 9 位数据异步通信 UART 模式，其一帧的信息由 11 位组成：1 位起始位，8 位数据位（低位在先），1 位可编程位（第 9 位数据）和 1 位停止位。发送时可编程位（第 9 位数据）由 SCON 中的 TB8 提供，可软件设置为 1 或 0，或者可将 PSW 中的奇/偶校验位 P 值装入 TB8（TB8 既可作为多机通信中的地址数据标志位，又可作为数据的奇偶校验位）。接收时第 9 位数据装入 SCON 的 RB8。TxD 为发送端口，RxD 为接收端口，以全双工模式进行接收/发送。

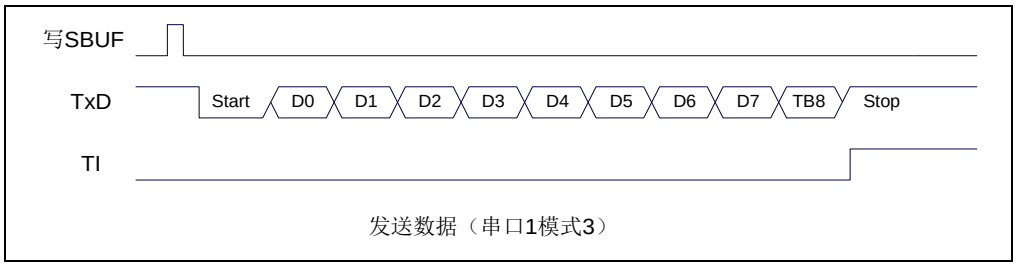
模式 3 和模式 1 相比，除发送时由 TB8 提供给移位寄存器第 9 数据位不同外，其余功能结构均基本相同，其接收‘发送操作过程及时序也基本相同。

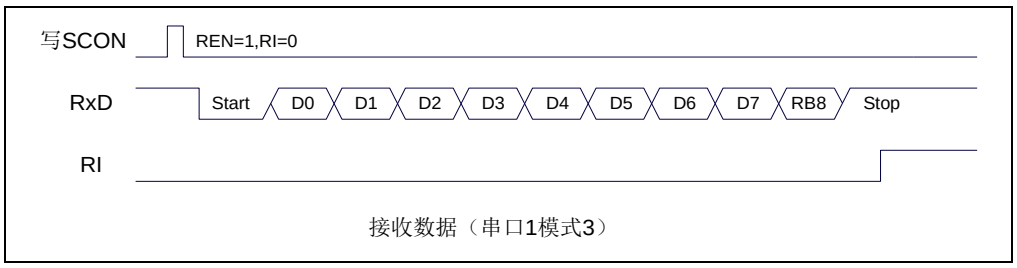
当接收器接收完一帧信息后必须同时满足下列条件：

- RI=0
- SM2=0 或者 SM2=1 且接收到的第 9 数据位 RB8=1。

当上述两条件同时满足时，才将接收到的移位寄存器的数据装入 SBUF 和 RB8 中，RI 标志位被置 1，并向主机请求中断处理。如果上述条件有一个不满足，则刚接收到移位寄存器中的数据无效而丢失，也不置位 RI。无论上述条件满足与否，接收器又重新开始检测 RxD 输入端口的跳变信息，接收下一帧的输入信息。在模式 3 中，接收到的停止位与 SBUF、RB8 和 RI 无关。

通过软件对 SCON 中的 SM2、TB8 的设置以及通信协议的约定，为多机通信提供了方便。





串口 1 模式 3 的波特率计算公式与模式 1 是完全相同的。请参考模式 1 的波特率计算公式。

19.3.9　自动地址识别，从机地址控制寄存器（SADDR，SADEN）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SADDR	A9H								
SADEN	B9H								

SADDR：从机地址寄存器

SADEN：从机地址屏蔽位寄存器

自动地址识别功能典型应用在多机通讯领域，其主要原理是从机系统通过硬件比较功能来识别来自于主机串口数据流中的地址信息，通过寄存器 SADDR 和 SADEN 设置的本机的从机地址，硬件自动对从机地址进行过滤，当来自于主机的从机地址信息与本机所设置的从机地址相匹配时，硬件产生串口中断；否则硬件自动丢弃串口数据，而不产生中断。当众多处于空闲模式的从机链接在一起时，只有从机地址相匹配的从机才会从空闲模式唤醒，从而可以大大降低从机 MCU 的功耗，即使从机处于正常工作状态也可避免不停地进入串口中断而降低系统执行效率。

要使用串口的自动地址识别功能，首先需要将参与通讯的 MCU 的串口通讯模式设置为模式 2 或者模式 3（通常都选择波特率可变的模式 3，因为模式 2 的波特率是固定的，不便于调节），并开启从机的 SCON 的 SM2 位。对于串口模式 2 或者模式 3 的 9 位数据位中，第 9 位数据（存放在 RB8 中）为地址/数据的标志位，当第 9 位数据为 1 时，表示前面的 8 位数据（存放在 SBUF 中）为地址信息。当 SM2 被设置为 1 时，从机 MCU 会自动过滤掉非地址数据（第 9 位为 0 的数据），而对 SBUF 中的地址数据（第 9 位为 1 的数据）自动与 SADDR 和 SADEN 所设置的本机地址进行比较，若地址相匹配，则会将 RI 置“1”，并产生中断，否则不予处理本次接收的串口数据。

从机地址的设置是通过 SADDR 和 SADEN 两个寄存器进行设置的。SADDR 为从机地址寄存器，里面存放本机的从机地址。SADEN 为从机地址屏蔽位寄存器，用于设置地址信息中的忽略位，设置方法如下：

例如

SADDR ＝　　11001010

SADEN ＝　　10000001

则匹配地址为　1xxxxxx0

即，只要主机送出的地址数据中的 bit0 为 0 且 bit7 为 1 就可以和本机地址相匹配

再例如

SADDR ＝　　11001010

SADEN ＝　　00001111

则匹配地址为　xxxx1010

即，只要主机送出的地址数据中的低 4 位为 1010 就可以和本机地址相匹配，而高 4 为被忽略，可以为任意值。

主机可以使用广播地址（FFH）同时选中所有的从机来进行通讯。

19.3.10 串口 1 同步模式控制寄存器 1（USARTCR1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
USARTCR1	7EFDC0H	LINEN	DORD	CLKEN	SPMOD	SPIEN	SPSLV	CPOL	CPHA

LINEN：LIN 模式使能位

0：禁止 LIN 模式

1：使能 LIN 模式

DORD：SPI 模式数据位发送/接收的顺序

0：先发送/接收数据的高位（MSB）

1：先发送/接收数据的低位（LSB）

CLKEN：SmartCard 模式时钟输出控制位

0：禁止时钟输出

1：使能时钟输出

SPMOD：SPI 模式使能位

0：禁止 SPI 模式

1：使能 SPI 模式

SPIEN：SPI 使能位

0：禁止 SPI 功能

1：使能 SPI 功能

当 USART1 需要工作在 SPI 模式时，必须先将 SPI 相关的模式寄存器、速度寄存器、时钟极性寄存器设置完成后，最后再设置 SPIEN 寄存器。

SPSLV：SPI 从机模式使能位

0：SPI 为主机模式

1：SPI 为从机模式

CPOL：SPI 时钟极性控制

0：SCLK 空闲时为低电平，SCLK 的前时钟沿为上升沿，后时钟沿为下降沿

1：SCLK 空闲时为高电平，SCLK 的前时钟沿为下降沿，后时钟沿为上升沿

CPHA：SPI 时钟相位控制

0：数据 SS 管脚为低电平驱动第一位数据并在 SCLK 的后时钟沿改变数据，前时钟沿采样数据

1：数据在 SCLK 的前时钟沿驱动，后时钟沿采样

19.3.11 串口 1 同步模式控制寄存器 2，硬件奇偶校验位设置（USARTCR2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
USARTCR2	7EFDC1H	IREN	IRLP	SCEN	NACK	HDSEL	PCEN	PS	PE

IREN：IrDA 模式使能位

0：禁止 IrDA 模式

1：使能 IrDA 模式

IRLP：IrDA 低电模式控制位

0：IrDA 为普通模式

1：IrDA 为低电模式

SCEN：SmartCard 模式使能位

0：禁止 SmartCard 模式

1：使能 SmartCard 模式

NACK：SmartCard 模式 NACK 输出使能位

0：禁止输出 NACK 信号

1：使能输出 NACK 信号

HDSEL：单线半双工模式使能位

0：禁止单线半双工模式

1：使能单线半双工模式

PCEN：硬件自动产生硬件奇偶校验位控制使能

0：禁止自动产生硬件奇偶校验位（串口的校验位为 TB8 设置的值）

1：使能自动产生硬件奇偶校验位

PS：硬件奇偶校验位模式选择

0：硬件根据 SBUF 的值自动产生偶校验位

1：硬件根据 SBUF 的值自动产生奇校验位

PE：校验位错误标志（必须软件清零）

0：无检验错误

1：有校验错误（串口接收 DMA 过程中如果发生接收数据校验位错误，DMA 不会停止，但校验位错误标志会一直保持直到 DMA 完成）

19.3.12 串口 1 同步模式控制寄存器 3（USARTCR3）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
USARTCR3	7EFDC2H	IrDA_LPBAUD[7:0]							

IrDA_LPBAUD：IrDA 低电模式波特率控制寄存器

IrDA_LPBAUD[7:0]	IrDA 低电模式波特率
0	SYSclk/16/256
1	SYSclk/16/1
2	SYSclk/16/2
...	...
255	SYSclk/16/255

19.3.13 串口 1 同步模式控制寄存器 4（USARTCR4）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
USARTCR4	7EFDC3H	-	-	-	-	SCCKS[1:0]		SPICKS[1:0]	

SCCKS[1:0]：SmartCard 模式时钟选择

SCCKS[1:0]	SmartCard 模式时钟
00	SYSclk/64
01	SYSclk/32
10	SYSclk/16
11	SYSclk/8

SPICKS[1:0]：SPI 模式时钟选择

SPICKS[1:0]	SPI 模式时钟
00	SYSclk/4
01	SYSclk/8
10	SYSclk/16
11	SYSclk/2

19.3.14 串口 1 同步模式控制寄存器 5（USARTCR5）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
USARTCR5	7EFDC4H	BRKDET	HDRER	SLVEN	ASYNC	TXCF	SENDBK	HDRDET	SYNC

BRKDET：LIN 从模式间隔场（BREAK）检测标志

0：未检测到 LIN 间隔场

1：检测到 LIN 间隔场，需要软件清零

HDRER：LIN 从模式报文头（HEADER）错误

0：未检测到 LIN 报文头错误

1：检测到 LIN 报文头错误，需要软件清零

SLVEN：LIN 从机模式使能位

0：LIN 为主机模式

1：LIN 为从机模式

ASYNC：LIN 自动同步功能使能位

0：禁止自动同步

1：使能自动同步

TXCF：传输冲突标志位。单线半双工模式、LIN 模式和 SmartCard 模式有效

0：未检测到传输冲突（发送的数据与接收的数据相同）

1：检测到传输冲突（发送的数据与接收的数据不同）

SENDBK：发送间隔场。软件写 1 触发发送间隔场，发送完成后硬件自动清 0

HDRDET：LIN 从模式报文头（HEADER）检测标志

0：未检测到 LIN 报文头

1：检测到 LIN 报文头，需要软件清零

SYNC：LIN 从模式同步场检测标志。

正确分析到同步场后，硬件将 SYNC 标志为置 1。在接收标志符场时硬件会自动清零 SYNC

19.3.15 串口 1 同步模式保护时间寄存器（USARTGTR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
USARTGTR	7EFDC5H								

USARTGTR 寄存器仅在 SmartCard 模式有效。该寄存器是根据波特率时钟数给出保护时间值。TI 标志在 SmartCard 发送的数据位等于 USARTGTR 寄存器所设置的保护时间值时被硬件置 1。注：USARTGTR 寄存器的值应大于 11

19.3.16 串口 1 同步模式波特率寄存器（USARTBR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
USARTBRH	7EFDC6H	USARTBR[15:8]							
USARTBRL	7EFDC7H	USARTBR[7:0]							

LIN 模式、IrDA 模式和 SmartCard 模式时的波特率计算公式

$$\text{同步波特率} = \frac{\text{SYSclk}}{16 * \text{USARTBR[15:0]}}$$

19.3.17 串口 1 接收超时控制寄存器（UR1TOCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
UR1TOCR	7EFD70H	ENTO	ENTOI	SCALE	-	-	-	-	-

ENTO：串口 1 接收超时功能控制位

0：禁止串口 1 接收超时功能

1：使能串口 1 接收超时功能

ENTOI：串口 1 接收超时中断控制位

0：禁止串口 1 接收超时中断

1：使能串口 1 接收超时中断

SCALE：串口 1 超时计数时钟源选择

0：串口数据位率（波特率）

1：系统时钟

19.3.18 串口 1 超时状态寄存器（UR1TOSR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
UR1TOSR	7EFD71H	CTOIF	-	-	-	-	-	-	TOIF

CTOIF：写“1”清除串口 1 超时中断标志位 TOIF。（只写）

TOIF：串口 1 超时中断请求标志位。（只读）

当发生串口 1 超时时，TOIF 会被硬件置“1”。如果 ENTOI 为 1 时会产生串口中断，中断入口地址为原串口 1 的中断入口地址。标志位需要软件向 CTOIF 位写“1”清零

19.3.19 串口 1 超时长度控制寄存器（UR1TOTE/H/L）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
UR1TOTE	7EFD88H	TM[23:16]							
UR1TOTH	7EFD72H	TM[15:8]							
UR1TOTL	7EFD73H	TM[7:0]							

TM[23:0]：串口 1 超时时间控制位。

当串口 1 处于接收空闲状态时，内部超时计数器根据 SCALE 寄存器所选择的时钟源进行计数，当计数时间达到 TM 所设置的超时时间时，便会产生超时中断。当串口 1 接收数据完成时，复位内部超时计数器，重新进行超时计数。

注：

1、如果需要使能接收超时中断功能，则 TM 不可设置为 0，且 UR1TOTL、UR1TOTH、UR1TOTE 寄存器的设置必须先设置 UR1TOTL 和 UR1TOTH，最后设置 UR1TOTE

2、必须使能串口接收才有接收超时功能

3、接收超时功能必须在正确接收到一字节数据后才能触发

4、正在接收过程中，无接收超时功能

19.4 串口 2（同步/异步串口 USART2）

19.4.1 串口 2 控制寄存器（S2CON）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S2CON	9AH	S2SM0/FE	S2SM1	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI

S2SM0/FE：当S2CFG寄存器中的S2MOD0位为1时，该位为帧错误检测标志位。当UART2在接收过程中检测到一个无效停止位时，通过UART2接收器将该位置1，必须由软件清零。当S2CFG寄存器中的S2MOD0位为0时，该位和S2SM1一起指定串口2的通信工作模式，如下表所示：

S2SM0	S2SM1	串口2工作模式	功能说明
0	0	模式0	同步移位串行方式
0	1	模式1	可变波特率8位数据方式
1	0	模式2	固定波特率9位数据方式
1	1	模式3	可变波特率9位数据方式

S2SM2：允许模式 2 或模式 3 多机通信控制位。当串口 2 使用模式 2 或模式 3 时，如果 S2SM2 位为 1 且 S2REN 位为 1，则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位（即 S2RB8）来筛选地址帧，若 S2RB8=1，说明该帧是地址帧，地址信息可以进入 S2BUF，并使 S2RI 为 1，进而在中断服务程序中再进行地址号比较；若 S2RB8=0，说明该帧不是地址帧，应丢掉且保持 S2RI=0。在模式 2 或模式 3 中，如果 S2SM2 位为 0 且 S2REN 位为 1，接收收机处于地址帧筛选被禁止状态，不论收到的 S2RB8 为 0 或 1，均可使接收到的信息进入 S2BUF，并使 S2RI=1，此时 S2RB8 通常为校验位。模式 1 和模式 0 为非多机通信方式，在这两种方式时，S2SM2 应设置为 0。

S2REN：允许/禁止串口接收控制位

0：禁止串口接收数据

1：允许串口接收数据

S2TB8：当串口 2 使用模式 2 或模式 3 时，S2TB8 为要发送的第 9 位数据，按需要由软件置位或清 0。在模式 0 和模式 1 中，该位不用。

S2RB8：当串口 2 使用模式 2 或模式 3 时，S2RB8 为接收到的第 9 位数据，一般用作校验位或者地址帧/数据帧标志位。在模式 0 和模式 1 中，该位不用。

S2TI：串口 2 发送中断请求标志位。在模式 0 中，当串口发送数据第 8 位结束时，由硬件自动将 S2TI 置 1，向主机请求中断，响应中断后 S2TI 必须用软件清零。在其他模式中，则在停止位开始发送时由硬件自动将 S2TI 置 1，向 CPU 发请求中断，响应中断后 S2TI 必须用软件清零。

S2RI：串口 2 接收中断请求标志位。在模式 0 中，当串口接收第 8 位数据结束时，由硬件自动将 S2RI 置 1，向主机请求中断，响应中断后 S2RI 必须用软件清零。在其他模式中，串行接收到停止位的中间时刻由硬件自动将 S2RI 置 1，向 CPU 发中断申请，响应中断后 S2RI 必须由软件清零。

19.4.2 串口 2 数据寄存器（S2BUF）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S2BUF	9BH								

S2BUF：串口 1 数据接收/发送缓冲区。S2BUF 实际是 2 个缓冲器，读缓冲器和写缓冲器，两个操作分别对应两个不同的寄存器，1 个是只写寄存器（写缓冲器），1 个是只读寄存器（读缓冲器）。对 S2BUF 进行读操作，实际是读取串口接收缓冲区，对 S2BUF 进行写操作则是触发串口开始发送数据。

19.4.3 串口 2 配置寄存器（S2CFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S2CFG	7EFDB4H	S2MOD	S2MOD0	S2M0x6					W1

S2MOD：串口 2 波特率控制位

0：串口 2 之模式 0/1/2/3 的波特率都不加倍

1：串口 2 之模式 2 的波特率加倍

S2MOD0：帧错误检测控制位

0：无帧错检测功能

1：使能帧错误检测功能。此时 S2CON 的 S2SM0/FE 为 FE 功能，即为帧错误检测标志位。

S2M0x6：串口 2 模式 0 的通讯速度控制

0：串口 2 模式 0 的波特率不加倍，固定为 Fosc/12

1：串口 2 模式 0 的波特率 6 倍速，即固定为 Fosc/12*6 ＝ Fosc/2

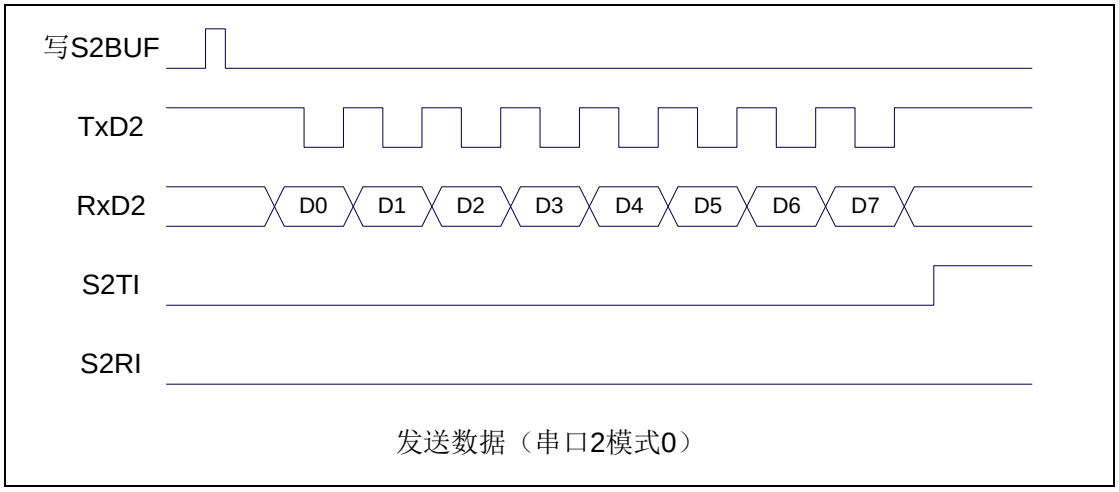
W1：当需要使用串口 2 时，此位必须设置为“1”，否则可能会产生不可预期的错误。若不需使用串口 2，则不用特别设置 W1。

19.4.4　串口 2 模式 0，模式 0 波特率计算公式

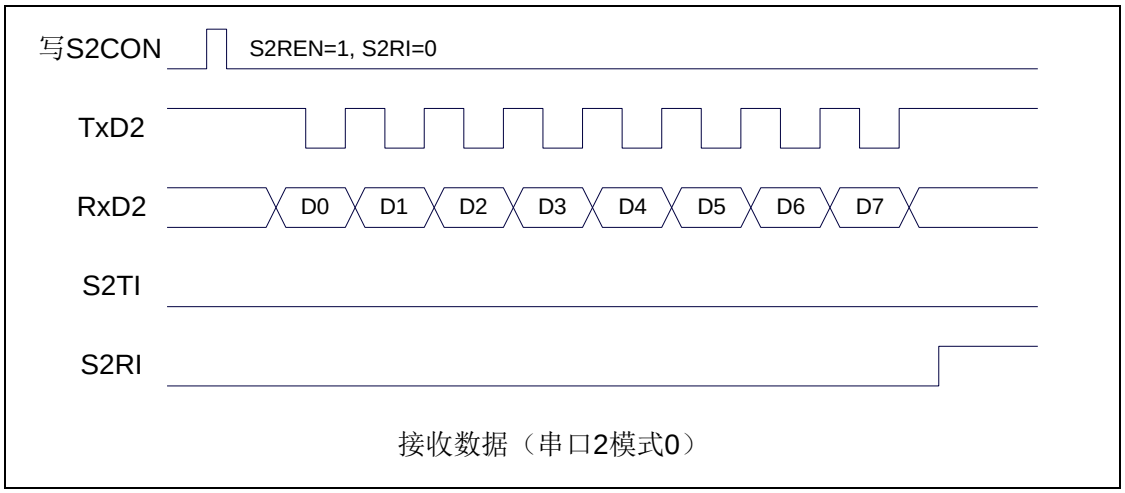
当串口 2 选择工作模式为模式 0 时，串行通信接口工作在同步移位寄存器模式，当串行口模式 0 的通信速度设置位 S2M0x6 为 0 时，其波特率固定为系统时钟时钟的 12 分频（SYSclk/12）；当设置 S2M0x6 为 1 时，其波特率固定为系统时钟频率的 2 分频（SYSclk/2）。RxD2 为串行通讯的数据口，TxD2 为同步移位脉冲输出脚，发送、接收的是 8 位数据，低位在先。

模式 0 的发送过程：当主机执行将数据写入发送缓冲器 S2BUF 指令时启动发送，串行口即将 8 位数据以 SYSclk/12 或 SYSclk/2（由 S2M0x6 确定是 12 分频还是 2 分频）的波特率从 RxD2 管脚输出(从低位到高位)，发送完中断标志 S2TI 置 1，TxD2 管脚输出同步移位脉冲信号。当写信号有效后，相隔一个时钟，发送控制端 SEND 有效(高电平)，允许 RxD2 发送数据，同时允许 TxD2 输出同步移位脉冲。一帧(8 位)数据发送完毕时，各控制端均恢复原状态，只有 S2TI 保持高电平，呈中断申请状态。在再次发送数据前，必须用软件将 S2TI 清 0。

模式 0 的接收过程：首先将接收中断请求标志 S2RI 清零并置位允许接收控制位 S2REN 时启动模式 0 接收过程。启动接收过程后，RxD2 为串行数据输入端，TxD2 为同步脉冲输出端。串行接收的波特率为 SYSclk/12 或 SYSclk/2（由 S2M0x6 确定是 12 分频还是 2 分频）。当接收完成一帧数据（8 位）后，控制信号复位，中断标志 S2RI 被置 1，呈中断申请状态。当再次接收时，必须通过软件将 S2RI 清 0



发送数据（串口2模式0）



接收数据（串口2模式0）

工作于模式 0 时，必须清 0 多机通信控制位 S2SM2，使之不影响 S2TB8 位和 S2RB8 位。由于波特率固定为 SYSclk/12 或 SYSclk/2，无需定时器提供，直接由单片机的时钟作为同步移位脉冲。

串口 2 模式 0 的波特率计算公式如下表所示（SYSclk 为系统工作频率）：

S2M0x6	波特率计算公式
0	$\text{波特率} = \frac{\text{SYSclk}}{12}$
1	$\text{波特率} = \frac{\text{SYSclk}}{2}$

19.4.5　串口 2 模式 1，模式 1 波特率计算公式

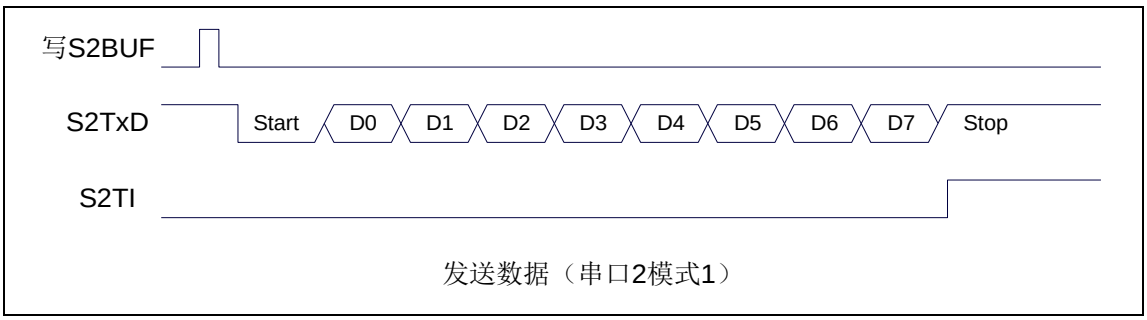
当软件设置 S2CON 的 S2SM0、S2SM1 为“01”时，串行口 2 则以模式 1 进行工作。此模式为 8 位 UART 格式，一帧信息为 10 位：1 位起始位，8 位数据位（低位在先）和 1 位停止位。波特率可变，即可根据需要进行设置波特率。TxD2 为数据发送口，RxD2 为数据接收口，串行口全双工接受/发送。

模式 1 的发送过程：串行通信模式发送时，数据由串行发送端 TxD2 输出。当主机执行一条写 S2BUF 的指令就启动串行通信的发送，写“S2BUF”信号还把“1”装入发送移位寄存器的第 9 位，并通知 TX 控制单元开始发送。移位寄存器将数据不断右移送 TxD 端口发送，在数据的左边不断移入“0”作补充。当数据的最高位移到移位寄存器的输出位置，紧跟其后的是第 9 位“1”，在它的左边各位全为“0”，这个状态条件，使 TX 控制单元作最后一次移位输出，然后使允许发送信号“SEND”失效，完成一帧信息的发送，并置位中断请求位 S2TI，即 S2TI=1，向主机请求中断处理。

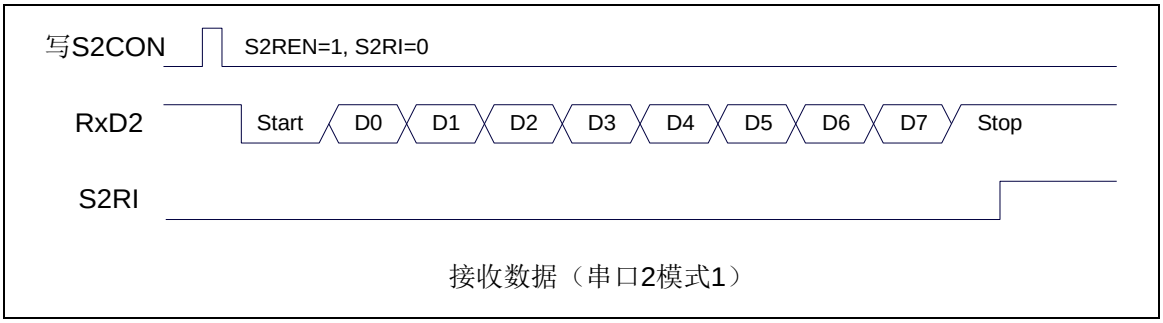
模式 1 的接收过程：当软件置位接收允许标志位 S2REN，即 S2REN=1 时，接收器便对 RxD2 端口的信号进行检测，当检测到 RxD2 端口发送从“1”→“0”的下降沿跳变时就启动接收器准备接收数据，并立即复位波特率发生器的接收计数器，将 1FFH 装入移位寄存器。接收的数据从接收移位寄存器的右边移入，已装入的 1FFH 向左边移出，当起始位"0"移到移位寄存器的最左边时，使 RX 控制器作最后一次移位，完成一帧的接收。若同时满足以下两个条件：

- S2RI=0；
- S2SM2=0 或接收到的停止位为 1。

则接收到的数据有效，实现装载入 S2BUF，停止位进入 S2RB8，S2RI 标志位被置 1，向主机请求中断，若上述两条件不能同时满足，则接收到的数据作废并丢失，无论条件满足与否，接收器重又检测 RxD2 端口上的"1"→"0"的跳变，继续下一帧的接收。接收有效，在响应中断后，S2RI 标志位必须由软件清 0。通常情况下，串行通信工作于模式 1 时，S2SM2 设置为"0"。



发送数据（串口2模式1）



接收数据（串口2模式1）

串口 2 的波特率是可变的，其波特率固定由定时器 2 产生。当定时器采用 1T 模式时（12 倍速），相应的波特率的速度也会相应提高 12 倍。

串口 2 模式 1 的波特率计算公式如下表所示：（SYSclk 为系统工作频率）

选择定时器	定时器速度	重装值计算公式	波特率
定时器2	1T	定时器2重载值 = 65536 - $\frac{SYSclk}{4\times波特率}$	波特率 = $\frac{SYSclk}{4\times(65536-定时器重装数)}$
	12T	定时器2重载值 = 65536 - $\frac{SYSclk}{12\times4\times波特率}$	波特率 = $\frac{SYSclk}{12\times4\times(65536-定时器重装数)}$

下面为常用频率与常用波特率所对应定时器的重载值

频率 (MHz)	波特率	定时器 2	
		1T 模式	12T 模式
11.0592	115200	FFE8H	FFFEH
	57600	FFD0H	FFFCH
	38400	FFB8H	FFFAH
	19200	FF70H	FFF4H
	9600	FEE0H	FFE8H
18.432	115200	FFD8H	-
	57600	FFB0H	-
	38400	FF88H	FFF6H
	19200	FF10H	FFECH
	9600	FE20H	FFD8H
22.1184	115200	FFD0H	FFFCH
	57600	FFA0H	FFF8H
	38400	FF70H	FFF4H
	19200	FEE0H	FFE8H
	9600	FDC0H	FFD0H

19.4.6　串口 2 模式 2，模式 2 波特率计算公式

当 S2SM0、S2SM1 两位为 10 时，串行口 2 工作在模式 2。串行口 2 工作模式 2 为 9 位数据异步通信 UART 模式，其一帧的信息由 11 位组成：1 位起始位，8 位数据位（低位在先），1 位可编程位（第 9 位数据）和 1 位停止位。发送时可编程位（第 9 位数据）由 S2CON 中的 S2TB8 提供，可软件设置为 1 或 0，或者可将 PSW 中的奇/偶校验位 P 值装入 S2TB8（S2TB8 既可作为多机通信中的地址数据标志位，又可作为数据的奇偶校验位）。接收时第 9 位数据装入 S2CON 的 S2RB8。TxD2 为发送端口，RxD2 为接收端口，以全双工模式进行接收/发送。

模式 2 的波特率固定为系统时钟的 64 分频或 32 分频（取决于 S2CFG 中 S2MOD 的值）

串口 2 模式 2 的波特率计算公式如下表所示（SYSclk 为系统工作频率）：

SMOD	波特率计算公式
0	波特率 $= \frac{SYSclk}{64}$
1	波特率 $= \frac{SYSclk}{32}$

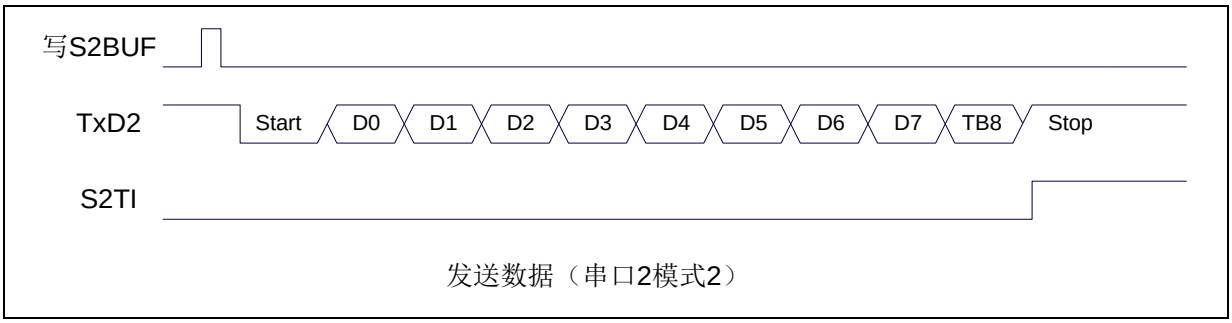
模式 2 和模式 1 相比，除波特率发生源略有不同，发送时由 S2TB8 提供给移位寄存器第 9 数据位不同外，其余功能结构均基本相同，其接收/发送操作过程及时序也基本相同。

当接收器接收完一帧信息后必须同时满足下列条件：

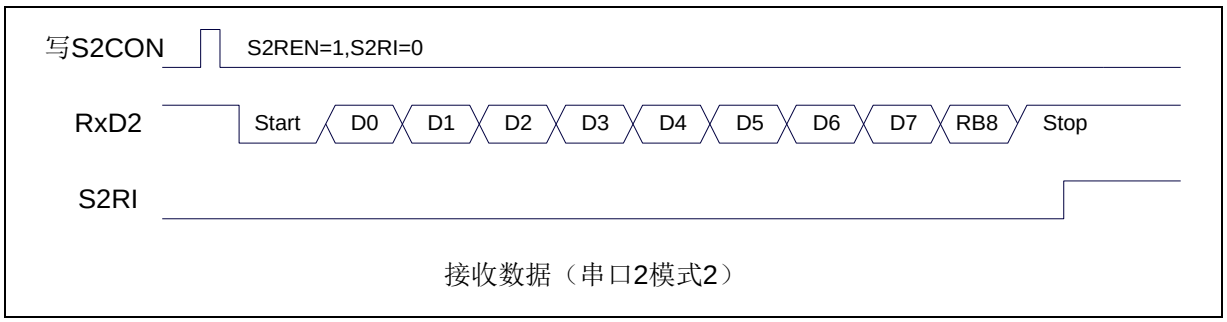
- S2RI=0
- S2SM2=0 或者 S2SM2=1 且接收到的第 9 数据位 S2RB8=1。

当上述两条件同时满足时，才将接收到的移位寄存器的数据装入 S2BUF 和 S2RB8 中，S2RI 标志位被置 1，并向主机请求中断处理。如果上述条件有一个不满足，则刚接收到移位寄存器中的数据无效而丢失，也不置位 S2RI。无论上述条件满足与否，接收器又重新开始检测 RxD2 输入端口的跳变信息，接收下一帧的输入信息。在模式 2 中，接收到的停止位与 S2BUF、S2RB8 和 S2RI 无关。

通过软件对 S2CON 中的 S2SM2、S2TB8 的设置以及通信协议的约定，为多机通信提供了方便。



发送数据（串口2模式2）



接收数据（串口2模式2）

19.4.7 串口 2 模式 3，模式 3 波特率计算公式

当 S2SM0、S2SM1 两位为 11 时，串行口 2 工作在模式 3。串行通信模式 3 为 9 位数据异步通信 UART 模式，其一帧的信息由 11 位组成：1 位起始位，8 位数据位（低位在先），1 位可编程位（第 9 位数据）和 1 位停止位。发送时可编程位（第 9 位数据）由 S2CON 中的 S2TB8 提供，可软件设置为 1 或 0，或者可将 PSW 中的奇/偶校验位 P 值装入 S2TB8（S2TB8 既可作为多机通信中的地址数据标志位，又可作为数据的奇偶校验位）。接收时第 9 位数据装入 S2CON 的 S2RB8。TxD2 为发送端口，RxD2 为接收端口，以全双工模式进行接收/发送。

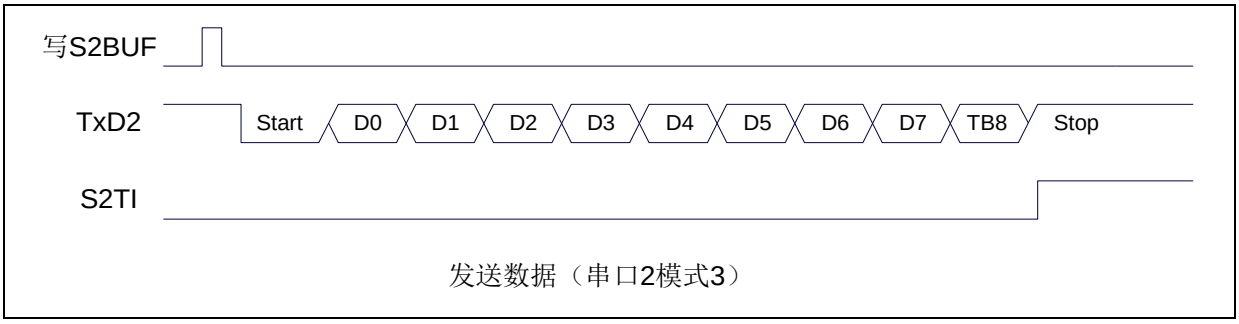
模式 3 和模式 1 相比，除发送时由 S2TB8 提供给移位寄存器第 9 数据位不同外，其余功能结构均基本相同，其接收‘发送操作过程及时序也基本相同。

当接收器接收完一帧信息后必须同时满足下列条件：

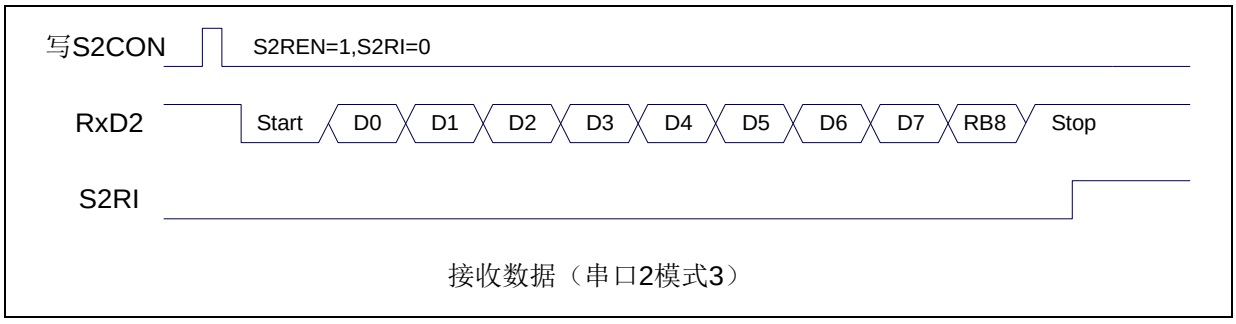
- S2RI=0
- S2SM2=0 或者 S2SM2=1 且接收到的第 9 数据位 S2RB8=1。

当上述两条件同时满足时，才将接收到的移位寄存器的数据装入 S2BUF 和 S2RB8 中，S2RI 标志位被置 1，并向主机请求中断处理。如果上述条件有一个不满足，则刚接收到移位寄存器中的数据无效而丢失，也不置位 S2RI。无论上述条件满足与否，接收器又重新开始检测 RxD2 输入端口的跳变信息，接收下一帧的输入信息。在模式 3 中，接收到的停止位与 S2BUF、S2RB8 和 S2RI 无关。

通过软件对 S2CON 中的 S2SM2、S2TB8 的设置以及通信协议的约定，为多机通信提供了方便。



发送数据（串口2模式3）



接收数据（串口2模式3）

串口 2 模式 3 的波特率计算公式与模式 1 是完全相同的。请参考模式 1 的波特率计算公式。

19.4.8　串口 2 自动地址识别

19.4.9　串口 2 从机地址控制寄存器（S2ADDR，S2ADEN）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S2ADDR	7EFDB5H								
S2ADEN	7EFDB6H								

S2ADDR：从机地址寄存器

S2ADEN：从机地址屏蔽位寄存器

自动地址识别功能典型应用在多机通讯领域，其主要原理是从机系统通过硬件比较功能来识别来自于主机串口 2 数据流中的地址信息，通过寄存器 S2ADDR 和 S2ADEN 设置的本机的从机地址，硬件自动对从机地址进行过滤，当来自于主机的从机地址信息与本机所设置的从机地址相匹配时，硬件产生串口 2 中断；否则硬件自动丢弃串口 2 数据，而不产生中断。当众多处于空闲模式的从机链接在一起时，只有从机地址相匹配的从机才会从空闲模式唤醒，从而可以大大降低从机 MCU 的功耗，即使从机处于正常工作状态也可避免不停地进入串口 2 中断而降低系统执行效率。

要使用串口 2 的自动地址识别功能，首先需要将参与通讯的 MCU 的串口 2 通讯模式设置为模式 1，并开启从机的 S2CON 的 S2SM2 位。对于串口 2 模式 1 的 9 位数据位中，第 9 位数据（存放在 S2RB8 中）为地址/数据的标志位，当第 9 位数据为 1 时，表示前面的 8 位数据（存放在 S2BUF 中）为地址信息。当 S2SM2 被设置为 1 时，从机 MCU 会自动过滤掉非地址数据（第 9 位为 0 的数据），而对 S2BUF 中的地址数据（第 9 位为 1 的数据）自动与 S2ADDR 和 S2ADEN 所设置的本机地址进行比较，若地址相匹配，则会将 S2RI 置“1”，并产生中断，否则不予处理本次接收的串口 2 数据。

从机地址的设置是通过 S2ADDR 和 S2ADEN 两个寄存器进行设置的。S2ADDR 为从机地址寄存器，里面存放本机的从机地址。S2ADEN 为从机地址屏蔽位寄存器，用于设置地址信息中的忽略位，设置方法如下：

例如
S2ADDR ＝　11001010
S2ADEN ＝　10000001
则匹配地址为　1xxxxxx0
即，只要主机送出的地址数据中的 bit0 为 0 且 bit7 为 1 就可以和本机地址相匹配

再例如
S2ADDR ＝　11001010
S2ADEN ＝　00001111
则匹配地址为　xxxx1010
即，只要主机送出的地址数据中的低 4 位为 1010 就可以和本机地址相匹配，而高 4 为被忽略，可以为任意值。

主机可以使用广播地址（FFH）同时选中所有的从机来进行通讯。

19.4.10 串口 2 同步模式控制寄存器 1（USART2CR1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
USART2CR1	7EFDC8H	LINEN	DORD	CLKEN	SPMOD	SPIEN	SPSLV	CPOL	CPHA

LINEN：LIN 模式使能位

0：禁止 LIN 模式

1：使能 LIN 模式

DORD：SPI 模式数据位发送/接收的顺序

0：先发送/接收数据的高位（MSB）

1：先发送/接收数据的低位（LSB）

CLKEN：SmartCard 模式时钟输出控制位

0：禁止时钟输出

1：使能时钟输出

SPMOD：SPI 模式使能位

0：禁止 SPI 模式

1：使能 SPI 模式

SPIEN：SPI 使能位

0：禁止 SPI 功能

1：使能 SPI 功能

当 USART2 需要工作在 SPI 模式时，必须先将 SPI 相关的模式寄存器、速度寄存器、时钟极性寄存器设置完成后，最后再设置 SPIEN 寄存器。

SPSLV：SPI 从机模式使能位

0：SPI 为主机模式

1：SPI 为从机模式

CPOL：SPI 时钟极性控制

0：SCLK 空闲时为低电平，SCLK 的前时钟沿为上升沿，后时钟沿为下降沿

1：SCLK 空闲时为高电平，SCLK 的前时钟沿为下降沿，后时钟沿为上升沿

CPHA：SPI 时钟相位控制

0：数据 SS 管脚为低电平驱动第一位数据并在 SCLK 的后时钟沿改变数据，前时钟沿采样数据

1：数据在 SCLK 的前时钟沿驱动，后时钟沿采样

19.4.11 串口 2 同步模式控制寄存器 2（USART2CR2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
USART2CR2	7EFDC9H	IREN	IRLP	SCEN	NACK	HDSEL	PCEN	PS	PE

IREN：IrDA 模式使能位

0：禁止 IrDA 模式

1：使能 IrDA 模式

IRLP：IrDA 低电模式控制位

0：IrDA 为普通模式

1：IrDA 为低电模式

SCEN：SmartCard 模式使能位

0：禁止 SmartCard 模式

1：使能 SmartCard 模式

NACK：SmartCard 模式 NACK 输出使能位

0：禁止输出 NACK 信号

1：使能输出 NACK 信号

HDSEL：单线半双工模式使能位

0：禁止单线半双工模式

1：使能单线半双工模式

PCEN：硬件自动产生校验位控制使能

0：禁止硬件自动产生校验位（串口的校验位为 S2TB8 设置的值）

1：使能硬件自动产生校验位

PS：硬件校验位模式选择

0：硬件根据 S2BUF 的值自动产生偶校验位

1：硬件根据 S2BUF 的值自动产生奇校验位

PE：校验位错误标志（必须软件清零）

0：无检验错误

1：有校验错误（串口接收 DMA 过程中如果发生接收数据校验位错误，DMA 不会停止，但校验位错误标志会一直保持直到 DMA 完成）

19.4.12 串口 2 同步模式控制寄存器 3（USART2CR3）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
USART2CR3	7EFDCAH	IrDA_LPBAUD[7:0]							

IrDA_LPBAUD：IrDA 低电模式波特率控制寄存器

IrDA_LPBAUD[7:0]	IrDA 低电模式波特率
0	SYSclk/16/256
1	SYSclk/16/1
2	SYSclk/16/2
...	...
255	SYSclk/16/255

19.4.13 串口 2 同步模式控制寄存器 4（USART2CR4）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
USART2CR4	7EFDCBH	-	-	-	-	SCCKS[1:0]		SPICKS[1:0]	

SCCKS[1:0]：SmartCard 模式时钟选择

SCCKS[1:0]	SmartCard 模式时钟
00	SYSclk/64
01	SYSclk/32
10	SYSclk/16
11	SYSclk/8

SPICKS[1:0]：SPI 模式时钟选择

SPICKS[1:0]	SPI 模式时钟
00	SYSclk/4
01	SYSclk/8
10	SYSclk/16
11	SYSclk/2

19.4.14 串口 2 同步模式控制寄存器 5（USART2CR5）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
USART2CR5	7EFDCCH	BRKDET	HDRER	SLVEN	ASYNC	TXCF	SENDBK	HDRDET	SYNC

BRKDET：LIN 从模式间隔场（BREAK）检测标志

0：未检测到 LIN 间隔场

1：检测到 LIN 间隔场，需要软件清零

HDRER：LIN 从模式报文头（HEADER）错误

0：未检测到 LIN 报文头错误

1：检测到 LIN 报文头错误，需要软件清零

SLVEN：LIN 从机模式使能位

0：LIN 为主机模式

1：LIN 为从机模式

ASYNC：LIN 自动同步功能使能位

0：禁止自动同步

1：使能自动同步

TXCF：传输冲突标志位。单线半双工模式、LIN 模式和 SmartCard 模式有效

0：未检测到传输冲突（发送的数据与接收的数据相同）

1：检测到传输冲突（发送的数据与接收的数据不同）

SENDBK：发送间隔场。软件写 1 触发发送间隔场，发送完成后硬件自动清 0

HDRDET：LIN 从模式报文头（HEADER）检测标志

0：未检测到 LIN 报文头

1：检测到 LIN 报文头，需要软件清零

SYNC：LIN 从模式同步场检测标志。

正确分析到同步场后，硬件将 SYNC 标志为置 1。在接收标志符场时硬件会自动清零 SYNC

19.4.15 串口 2 同步模式保护时间寄存器（USART2GTR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
USART2GTR	7EFDCDH								

USARTGTR 寄存器仅在 SmartCard 模式有效。该寄存器是根据波特率时钟数给出保护时间值。S2TI 标志在 SmartCard 发送的数据位等于 USARTGTR 寄存器所设置的保护时间值时被硬件置 1。注：USARTGTR 寄存器的值应大于 11

19.4.16 串口 2 同步模式波特率寄存器（USART2BR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
USART2BRH	7EFDCEH	USART2BR[15:8]							
USART2BRL	7EFDCFH	USART2BR[7:0]							

LIN 模式、IrDA 模式和 SmartCard 模式时的波特率计算公式

$$同步波特率 = \frac{SYSclk}{16 * USART2BR[15:0]}$$

19.4.17 串口 2 接收超时控制寄存器（UR2TOCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
UR2TOCR	7EFD74H	ENTO	ENTOI	SCALE	-	-	-	-	-

ENTO：串口 2 接收超时功能控制位

0：禁止串口 2 接收超时功能

1：使能串口 2 接收超时功能

ENTOI：串口 2 接收超时中断控制位

0：禁止串口 2 接收超时中断

1：使能串口 2 接收超时中断

SCALE：串口 2 超时计数时钟源选择

0：串口数据位率（波特率）

1：系统时钟

19.4.18 串口 2 超时状态寄存器（UR2TOSR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
UR2TOSR	7EFD75H	CTOIF	-	-	-	-	-	-	TOIF

CTOIF：写“1”清除串口超时中断标志位 TOIF。（只写）

TOIF：串口 2 超时中断请求标志位。（只读）

当发生串口 2 超时时，TOIF 会被硬件置“1”。如果 ENTOI 为 1 时会产生串口中断，中断入口地址为原串口 2 的中断入口地址。标志位需要软件向 CTOIF 位写“1”清零

19.4.19 串口 2 超时长度控制寄存器（UR2TOTE/H/L）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
UR2TOTE	7EFD89H	TM[23:16]							
UR2TOTH	7EFD76H	TM[15:8]							
UR2TOTL	7EFD77H	TM[7:0]							

TM[23:0]：串口 2 超时时间控制位。

当串口 2 处于接收空闲状态时，内部超时计数器根据 SCALE 寄存器所选择的时钟源进行计数，当计数时间达到 TM 所设置的超时时间时，便会产生超时中断。当串口 2 接收数据完成时，复位内部超时计数器，重新进行超时计数。

注：

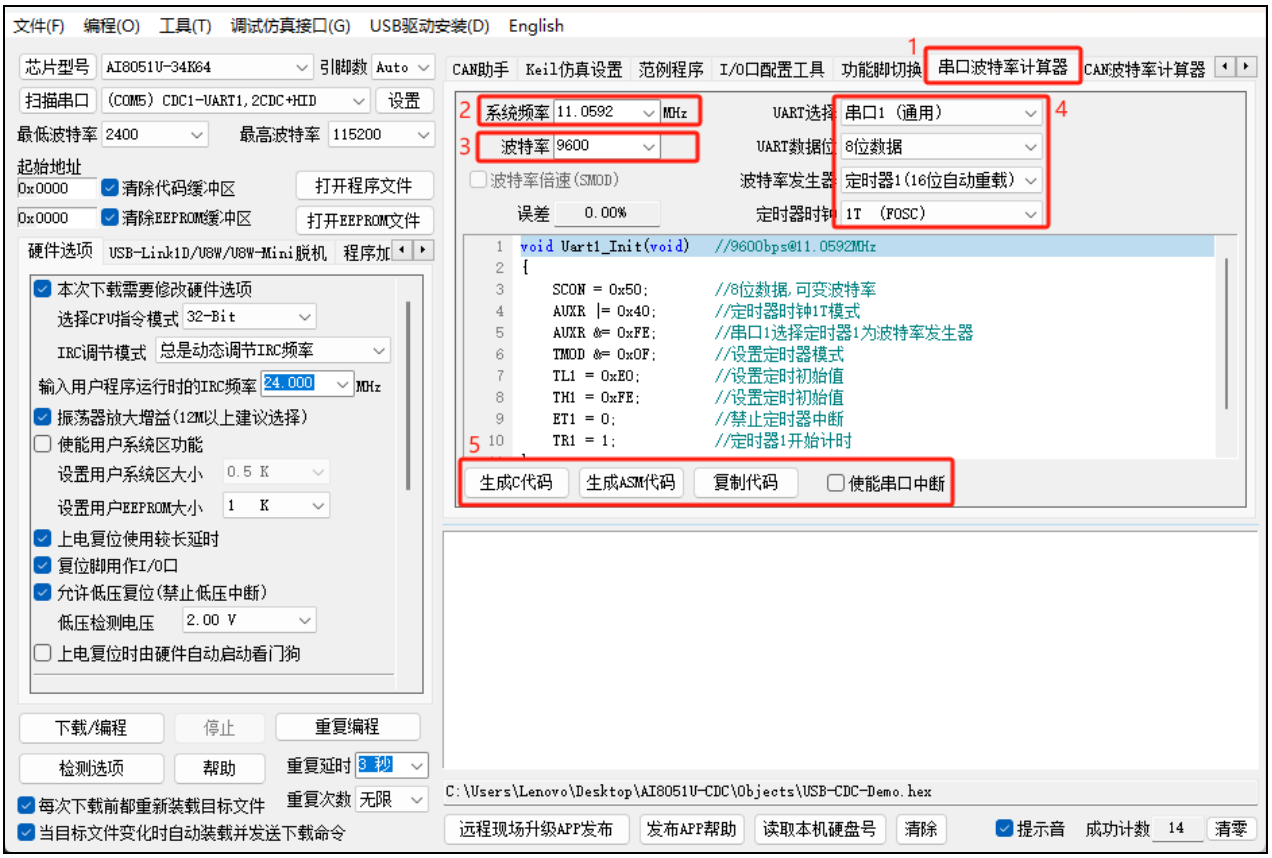
1、如果需要使能接收超时中断功能，则 TM 不可设置为 0，且 UR2TOTL、UR2TOTH、UR2TOTE 寄存器的设置必须先设置 UR2TOTL 和 UR2TOTH，最后设置 UR2TOTE

2、必须使能串口接收才有接收超时功能

3、接收超时功能必须在正确接收到一字节数据后才能触发

4、正在接收过程中，无接收超时功能

19.5 AIapp-ISP | 串口波特率计算器工具



①：在下载软件中选择“串口波特率计算器”功能页，进入串口代码生成界面

②：设置系统工作频率（单位：MHz）

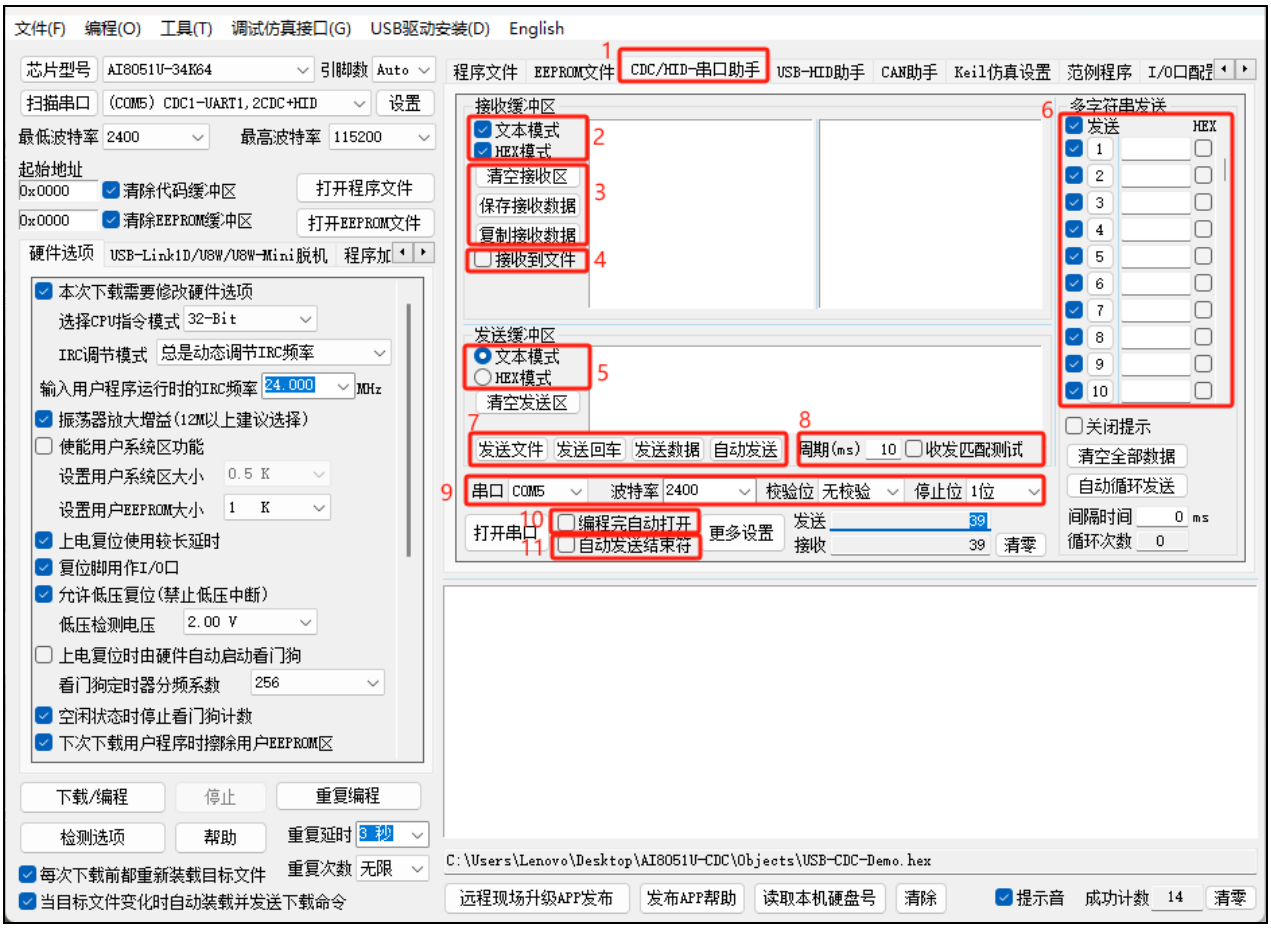
③：设置串口波特率

④：选择目标串口，并设置串口模式、波特率发生器等参数

⑤：手动生成 C 代码或者 ASM 代码，复制范例

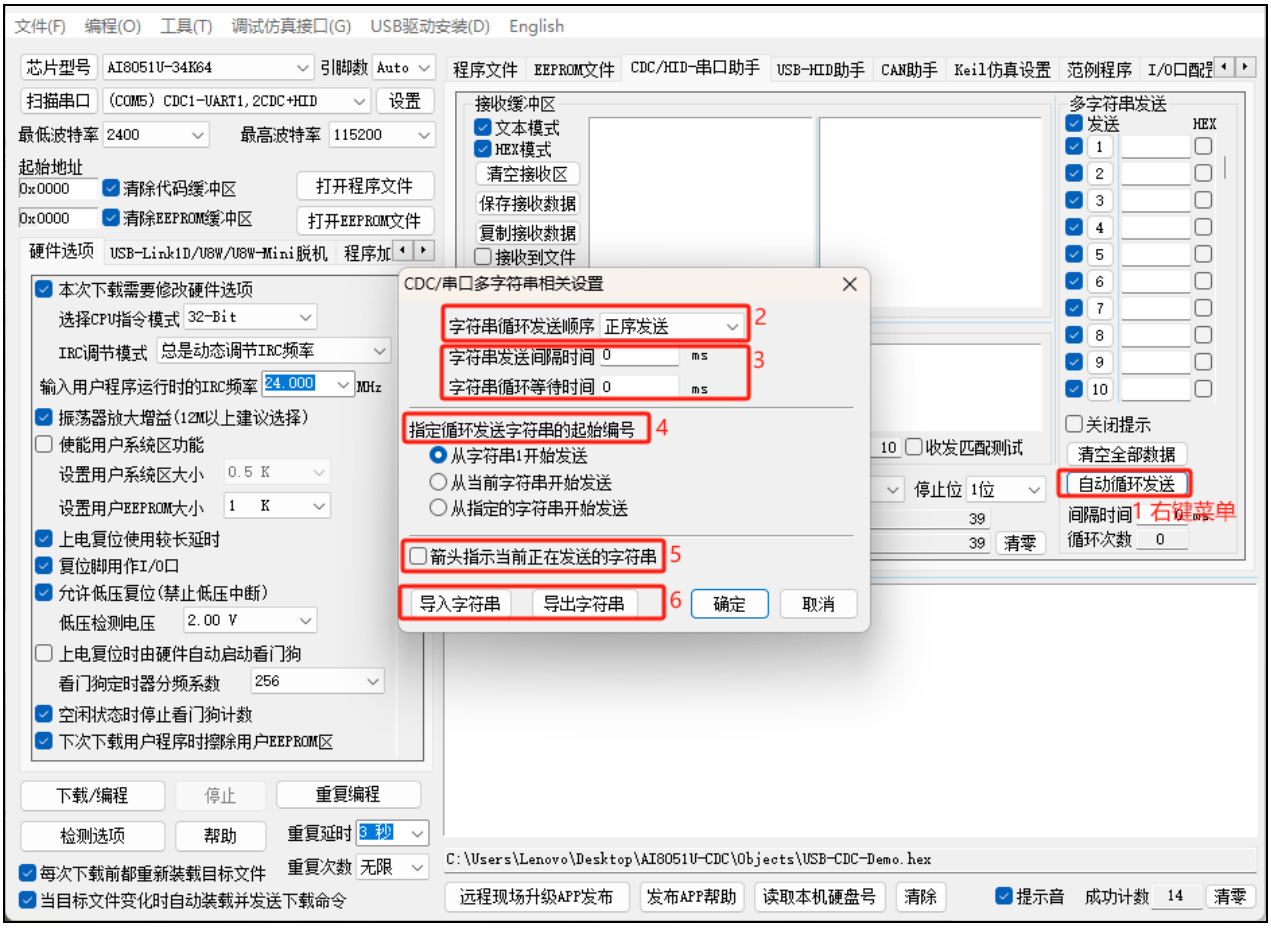
19.6　AIapp-ISP | 串口助手/USB-CDC

串口助手主界面



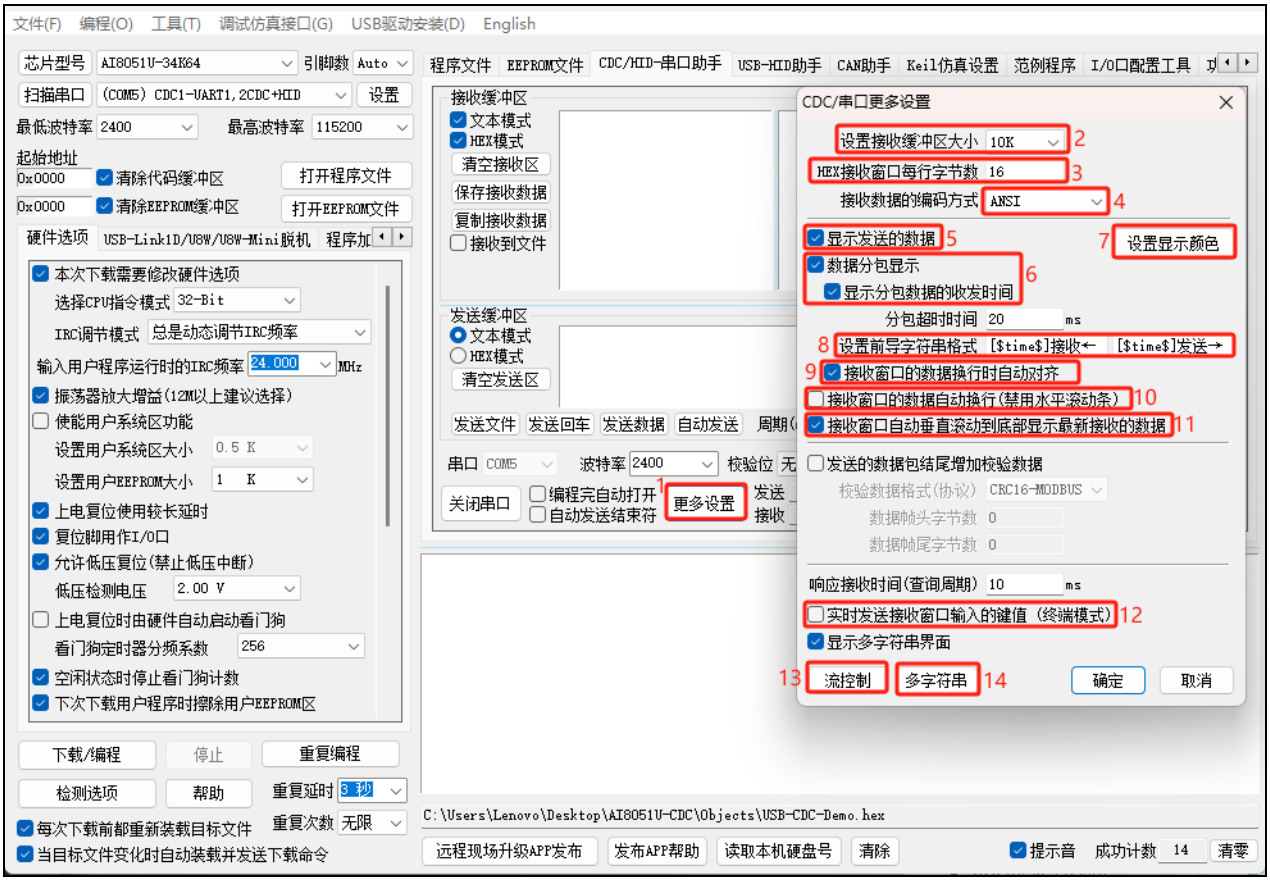
①：在下载软件中选择“USB-CDC/串口助手”功能页，进入串口助手界面
②：选择接收数据的显示格式
③：保存/复制接收的数据
④：设置接收数据自动存储到文件
⑤：选择发送数据的格式
⑥：“多字符串”控制界面
⑦：数据/文件发送按钮
⑧：设置重复下载的周期
⑨：选择串口号，设置串口参数
⑩：设置 ISP 下载完成后是否自动打开串口
⑪：设置发送完成数据后，是否自动发送结束符

多字符串设置界面



①：鼠标右键点击“自动循环发送”按钮，点击右键菜单“设置...”进入多字符串设置界面
②：设置多字符串循环发送顺序
③：设置多字符串循环发送间隔时间
④：设置多字符串循环发送的起始编号
⑤：设置是否需要用箭头指示当前正在发送的字符串
⑥：多字符串导出到文件或从文件导入

串口助手更多设置



①：点击“更多设置”按钮，进入串口助手更多设置界面

②：设置接收缓存大小（缓存越大，软件反应越慢）

③：设置 HEX 接收数据每行显示的数据个数

④：设置接收数据的汉字编码格式

支持如下汉字编码格式：

ANSI：GB2312 汉字编码

UTF8：UNICODE 互联网常用编码

UTF16-LE：小端 UTF16 编码

UTF16=BE：大端 UTF16 编码

⑤：设置是否显示发送的数据

⑥：设置数据是否自动分包显示

⑦：设置界面的显示颜色

⑧：设置接收窗口数据显示的前导字符

⑨：设置接收数据的换行显示模式

⑩：设置接收窗口是否自动换行

⑪：设置发送数据是否自动追加校验数据

支持如下校验格式：

ADD8：字节校验和

ADD8N：字节校验和补码

ADD16-LE：小端双字节校验和

ADD16-BE：大端双字节校验和

XOR8：字节异或

CRC16-MODBUS：MODBUS 协议的 CRC16

CRC16-USB：USB 协议的 CRC16

CRC16-XMODEM：XMODEM 协议的 CRC16

CRC32：32 位 CRC 校验

⑫：设置是否使能终端模式（终端模式：将光标定位到接收窗口，按下键盘按键实时发送相应的键码）

⑬：进入流控制设置界面

⑭：进入多字符串界面设置界面

19.7　范例程序

19.7.1　串口 1 使用定时器 2 做波特率发生器

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                  //头文件见下载软件
#include "intrins.h"                                  //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   FOSC        11059200UL                      //定义为无符号长整型,避免计算溢出
#define   BRT         (65536 - (FOSC / 115200+2) / 4)
                                                      //加 2 操作是为了让 Keil 编译器
                                                      //自动实现四舍五入运算

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

void UartInit()
{
    SCON = 0x50;
    T2L = BRT;
    T2H = BRT >> 8;
    S1BRT = 1;
    T2x12 = 1;
    T2R = 1;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void UartSendStr(char *p)
{
```

```
        while (*p)
        {
            UartSend(*p++);
        }
}

void main()
{
        EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                    //(32 位模式请使用这句，注释下一句)
//      P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                    //(8 位模式请使用这句，注释上一句)
        WTST = 0;                                   //设置取程序代码等待时间，
                                                    //赋值为 0 表示不等待，程序以最快速度运行
        CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                    //赋值为 0 表示用最快速度访问，不增加额外的等待时间

        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

        UartInit();
        ES = 1;
        EA = 1;
        UartSendStr("Uart Test !\r\n");

        while (1)
        {
            if (rptr != wptr)
            {
                UartSend(buffer[rptr++]);
                rptr &= 0x0f;
            }
        }
}
```

19.7.2　串口 1 使用定时器 1（模式 0）做波特率发生器

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                              //头文件见下载软件
#include "intrins.h"                               //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   FOSC          11059200UL                 //定义为无符号长整型,避免计算溢出
#define   BRT           (65536 - (FOSC / 115200+2) / 4)
```

```
                                                    //加 2 操作是为了让 Keil 编译器
                                                    //自动实现四舍五入运算

bit         busy;
char        wptr;
char        rptr;
char        buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

void UartInit()
{
    SCON = 0x50;
    S1BRT = 0;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    T1x12 = 1;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void UartSendStr(char *p)
{
    while (*p)
    {
        UartSend(*p++);
    }
}

void main()
{
    EAXFR = 1;                                      //允许访问扩展的特殊寄存器，XFR
                                                    //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                  //允许访问扩展的特殊寄存器，XFR
                                                    //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                       //设置取程序代码等待时间，
```

```
                                                    //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                      //设置访问片内的 xdata 速度，
                                                    //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

19.7.3　串口 1 使用定时器 1（模式 2）做波特率发生器

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                //头文件见下载软件
#include "intrins.h"                                //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   FOSC        11059200UL                    //定义为无符号长整型,避免计算溢出
#define   BRT         (256 - (FOSC / 115200+16) / 32)
                                                    //加 16 操作是为了让 Keil 编译器
                                                    //自动实现四舍五入运算

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
```

```
            busy = 0;
        }
        if (RI)
        {
            RI = 0;
            buffer[wptr++] = SBUF;
            wptr &= 0x0f;
        }
}

void UartInit()
{
        SCON = 0x50;
        S1BRT = 0;
        TMOD = 0x20;
        TL1 = BRT;
        TH1 = BRT;
        TR1 = 1;
        T1x12 = 1;
        wptr = 0x00;
        rptr = 0x00;
        busy = 0;
}

void UartSend(char dat)
{
        while (busy);
        busy = 1;
        SBUF = dat;
}

void UartSendStr(char *p)
{
        while (*p)
        {
            UartSend(*p++);
        }
}

void main()
{
        EAXFR = 1;                  //允许访问扩展的特殊寄存器，XFR
                                    //(32 位模式请使用这句，注释下一句)
//      P_SW2 |= 0x80;              //允许访问扩展的特殊寄存器，XFR
                                    //(8 位模式请使用这句，注释上一句)
        WTST = 0;                   //设置取程序代码等待时间，
                                    //赋值为 0 表示不等待，程序以最快速度运行
        CKCON = 0;                  //设置访问片内的 xdata 速度，
                                    //赋值为 0 表示用最快速度访问，不增加额外的等待时间

        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
```

```
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

19.7.4　串口 2 使用定时器 2 做波特率发生器

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                  //头文件见下载软件
#include "intrins.h"                                  //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   FOSC        11059200UL                      //定义为无符号长整型,避免计算溢出
#define   BRT         (65536 - (FOSC / 115200+2) / 4)
                                                      //加 2 操作是为了让 Keil 编译器
                                                      //自动实现四舍五入运算

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void Uart2Isr() interrupt 8
{
    if (S2TI)
    {
        S2TI = 0;
        busy = 0;
    }
    if (S2RI)
    {
        S2RI = 0;
        buffer[wptr++] = S2BUF;
        wptr &= 0x0f;
    }
}

void Uart2Init()
{
    EAXFR = 1;                                        //允许访问扩展的特殊寄存器，XFR
```

```
                                        //(32 位模式请使用这句，注释下一句)
//      P_SW2 |= 0x80;                  //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
        WTST = 0;                       //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
        CKCON = 0;                      //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间
        S2CFG = 0x01;

        S2CON = 0x50;
        T2L = BRT;
        T2H = BRT >> 8;
        T2x12 = 1;
        T2R = 1;
        wptr = 0x00;
        rptr = 0x00;
        busy = 0;
}

void Uart2Send(char dat)
{
        while (busy);
        busy = 1;
        S2BUF = dat;
}

void Uart2SendStr(char *p)
{
        while (*p)
        {
                Uart2Send(*p++);
        }
}

void main()
{
        EAXFR = 1;                      //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//      P_SW2 |= 0x80;                  //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
        WTST = 0;                       //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
        CKCON = 0;                      //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间


        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

        Uart2Init();
```

```
    IE2 = 0x01;
    EA = 1;
    Uart2SendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart2Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

19.7.5　使用 USART1 的 SPI 接口访问串行 FLASH（DMA 方式）

```
//测试工作频率为 24MHz

#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。
#include "stdio.h"


#define   FOSC    24000000UL                          //系统工作频率
#define   BAUD    (65536 - (FOSC/115200+2)/4)         //调试串口波特率
                                                      //加 2 操作是为了让 Keil 编译器
                                                      //自动实现四舍五入运算

typedef   bit             BOOL;
typedef   unsigned char   BYTE;
typedef   unsigned int    WORD;
typedef   unsigned long   DWORD;

sbit  S1SS    =   P2^2;
sbit  S1MOSI  =   P2^3;
sbit  S1MISO  =   P2^4;
sbit  S1SCLK  =   P2^5;

void sys_init();
void usart1_spi_init();
void usart1_tx_dma(WORD size, BYTE xdata *pdat);
void usart1_rx_dma(WORD size, BYTE xdata *pdat);
BOOL flash_is_busy();
void flash_read_id();
void flash_read_data(DWORD addr, WORD size, BYTE xdata *pdat);
void flash_write_enable();
void flash_write_data(DWORD addr, WORD size, BYTE xdata *pdat);
void flash_erase_sector(DWORD addr);

BYTE xdata buffer1[256];                              //定义缓冲区
BYTE xdata buffer2[256];                              //注意:如果需要使用 DMA 发送数据
                                                      //则缓冲区必须定义在 xdata 区域内


void main()
{
```

```
    int    i;

    sys_init();                                    //系统初始化
    usart1_spi_init();                             //USART1 使能 SPI 模式初始化

    printf("\r\nUSART_SPI_DMA test !\r\n");
    flash_read_id();
    flash_read_data(0x0000, 0x80, buffer1);        //读取外挂 FLASH 的数据
    flash_erase_sector(0x0000);                    //擦除外挂 FLASH 的一个扇区
    flash_read_data(0x0000, 0x80, buffer1);
    for (i=0; i<128; i++)
        buffer2[i] = i;
    flash_write_data(0x0000, 0x80, buffer2);       //数据到外挂 FLASH
    flash_read_data(0x0000, 0x80, buffer1);

    while (1);
}

char putchar(char dat)
{
    while (!S2TI);
    S2TI = 0;
    S2BUF = dat;

    return dat;
}

void sys_init()
{
    EAXFR = 1;                                     //允许访问扩展的特殊寄存器，XFR
                                                   //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                 //允许访问扩展的特殊寄存器，XFR
                                                   //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                      //设置取程序代码等待时间，
                                                   //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                     //设置访问片内的 xdata 速度，
                                                   //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00; P0M1 = 0x00;
    P1M0 = 0x00; P1M1 = 0x00;
    P2M0 = 0x00; P2M1 = 0x00;
    P3M0 = 0x00; P3M1 = 0x00;
    P4M0 = 0x00; P4M1 = 0x00;
    P5M0 = 0x00; P5M1 = 0x00;
    P6M0 = 0x00; P6M1 = 0x00;
    P7M0 = 0x00; P7M1 = 0x00;

    S2_S = 1;                                      //初始化串口 2,用于调试程序
    S2CON =  0x52;
    T2L  = BAUD;
    T2H  = BAUD >> 8;
    T2x12 =    1;
    T2R  = 1;
}

void usart1_spi_init()
{
    S1SPI_S0 = 1;                                  //切换 S1SPI 到
                                                   //P2.2/S1SS,P2.3/S1MOSI,P2.4/S1MISO,P2.5/S1SCLK
```

```
    S1SPI_S1 = 0;
    SCON = 0x10;                        //使能接收,必须设置为串口模式 0

    USARTCR1 = 0x10;                    //使能 USART1 的 SPI 模式
//  USARTCR1 |=  0x40;                  //DORD=1
    USARTCR1 &= ~0x40;                  //DORD=0
//  USARTCR1 |=  0x04;                  //从机模式
    USARTCR1 &= ~0x04;                  //主机模式
    USARTCR1 |=  0x00;                  //CPOL=0, CPHA=0
//  USARTCR1 |=  0x01;                  //CPOL=0, CPHA=1
//  USARTCR1 |=  0x02 ;                 //CPOL=1, CPHA=0
//  USARTCR1 |=  0x03;                  //CPOL=1, CPHA=1
    USARTCR4 = 0x00;                    //SPI 速度为 SYSCLK/4
//  USARTCR4 = 0x01;                    //SPI 速度为 SYSCLK/8
//  USARTCR4 = 0x02;                    //SPI 速度为 SYSCLK/16
//  USARTCR4 = 0x03;                    //SPI 速度为 SYSCLK/2
    USARTCR1 |=  0x08;                  //使能 SPI 功能
}

BYTE usart1_spi_shift(BYTE dat)
{
    TI = 0;
    SBUF = dat;                         //发送数据
    while (!TI);                        //TI 标志是主机模式发送/接收数据完成标志

    return SBUF;                        //读取接收的数据
}

BOOL flash_is_busy()
{
    BYTE dat;

    S1SS = 0;

    usart1_spi_shift(0x05);             //发送读取状态寄存器命令
    dat = usart1_spi_shift(0);          //读取状态寄存器

    S1SS = 1;

    return (dat & 0x01);                //检测 FLASH 的忙标志
}

void flash_read_id()
{
    BYTE id[3];

    S1SS = 0;

    usart1_spi_shift(0x9f);             //发送读取 FLASH    ID 命令
    id[0] = usart1_spi_shift(0);        //读取 ID1
    id[1] = usart1_spi_shift(0);        //读取 ID2
    id[2] = usart1_spi_shift(0);        //读取 ID3

    S1SS = 1;

    printf("ReadID : ");
    printf("%02bx", id[0]);
    printf("%02bx", id[1]);
    printf("%02bx\r\n", id[2]);
```

```
}

void flash_read_data(DWORD addr, WORD size, BYTE xdata *pdat)
{
    WORD sz;
    BYTE *ptr;

    while (flash_is_busy());

    S1SS = 0;

    usart1_spi_shift(0x03);                         //发送读取 FLASH 数据命令
    usart1_spi_shift((BYTE)(addr >> 16));
    usart1_spi_shift((BYTE)(addr >> 8));
    usart1_spi_shift((BYTE)(addr));                 //设置目标地址

//  sz = size;
//  ptr = pdat;
//  while (sz--)
//      *ptr++ = usart1_spi_shift(0);               //寄存器方式读数据

    usart1_rx_dma(size, pdat);                      //DMA 方式读数据

    S1SS = 1;

    printf("ReadData : ");
    sz = size;
    ptr  = pdat;
    for  (sz=0; sz<size; sz++)
    {
        printf("%02bx ", *ptr++);                   //将读到的数据发送到串口,调试使用
        if ((sz% 16) == 15)
        {
            printf("\r\n              ");
        }
    }
    printf("\r\n");
}

void flash_write_enable()
{
    while (flash_is_busy());

    S1SS = 0;

    usart1_spi_shift(0x06);                         //发送写使能命令

    S1SS = 1;
}

void flash_write_data(DWORD addr, WORD size, BYTE xdata *pdat)
{
    WORD sz;

    sz = size;
    while (sz)
    {
        flash_write_enable();
```

```
        S1SS = 0;

        usart1_spi_shift(0x02);                         //发送写数据命令
        usart1_spi_shift((BYTE)(addr >> 16));
        usart1_spi_shift((BYTE)(addr >> 8));
        usart1_spi_shift((BYTE)(addr));

//      do
//      {
//          usart1_spi_shift(*pdat++);                  //寄存器方式写数据
//          addr++;
//
//          if ((BYTE)(addr) == 0x00)
//              break;
//      } while     (--sz);

        usart1_tx_dma(sz, pdat);                        //DMA 方式写数据(注意:数据必须在一个 page 之内)
        sz = 0;

        S1SS = 1;
    }

    printf("Program !\r\n");
}

void flash_erase_sector(DWORD addr)
{
    flash_write_enable();

    S1SS = 0;
    usart1_spi_shift(0x20);                             //发送擦除命令
    usart1_spi_shift((BYTE)(addr >> 16));
    usart1_spi_shift((BYTE)(addr >> 8));
    usart1_spi_shift((BYTE)(addr));
    S1SS = 1;

    printf("Erase Sector !\r\n");
}

void usart1_tx_dma(WORD size, BYTE xdata *pdat)
{
    size--;                                             //DMA 传输字节数比实际少 1

    DMA_UR1T_CFG = 0x00;                                //关闭 DMA 中断
    DMA_UR1T_STA = 0x00;                                //清除 DMA 状态
    DMA_UR1T_AMT = size;                                //设置 DMA 传输字节数
    DMA_UR1T_AMTH = size >> 8;
    DMA_UR1T_TXAL = (BYTE)pdat;                         //设置缓冲区地址(注意:缓冲区必须是 xdata 类型)
    DMA_UR1T_TXAH = (WORD)pdat >> 8;
    DMA_UR1T_CR = 0xc0;                                 //使能 DMA,触发串口 1 发送数据

    while (!(DMA_UR1T_STA & 0x01));                     //等待 DMA 数据传输完成
    DMA_UR1T_STA = 0x00;                                //清除 DMA 状态
    DMA_UR1T_CR = 0x00;                                 //关闭 DMA
}

void usart1_rx_dma(WORD size, BYTE xdata *pdat)
{
    size--;                                             //DMA 传输字节数比实际少 1
```

```
    DMA_UR1R_CFG = 0x00;                    //关闭 DMA 中断
    DMA_UR1R_STA = 0x00;                    //清除 DMA 状态
    DMA_UR1R_AMT = size;                    //设置 DMA 传输字节数
    DMA_UR1R_AMTH = size >> 8;
    DMA_UR1R_RXAL = (BYTE)pdat;             //设置缓冲区地址(注意:缓冲区必须是 xdata 类型)
    DMA_UR1R_RXAH = (WORD)pdat >> 8;
    DMA_UR1R_CR = 0xa1;                     //使能 DMA,清空接收 FIFO,触发串口 1 接收数据

                                            //!!!!!!!!!!!!!
    usart1_tx_dma(size+1, pdat);            //注意:接收数据时必须同时启动发送 DMA
                                            //!!!!!!!!!!!!!

    while (!(DMA_UR1R_STA & 0x01));         //等待 DMA 数据传输完成
    DMA_UR1R_STA = 0x00;                    //清除 DMA 状态
    DMA_UR1R_CR = 0x00;                     //关闭 DMA
}
```

19.7.6　USART1 和 USART2 的 SPI 接口相互传输数据（中断方式）

```
//测试工作频率为 24MHz

#include “Ai8051U.H”                       //头文件见下载软件
#include "intrins.h"                        //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。
#include "stdio.h"

#define   FOSC    24000000UL                //系统工作频率

typedef   bit             BOOL;
typedef   unsigned char   BYTE;
typedef   unsigned int    WORD;
typedef   unsigned long   DWORD;

sbit S1SS   =   P2^2;
sbit S1MOSI =   P2^3;
sbit S1MISO =   P2^4;
sbit S1SCLK =   P2^5;

sbit S2SS   =   P2^2;
sbit S2MOSI =   P2^3;
sbit S2MISO =   P2^4;
sbit S2SCLK =   P2^5;

void sys_init();
void usart1_spi_init();
void usart2_spi_init();
void test();

BYTE xdata buffer1[256];                    //定义缓冲区
BYTE xdata buffer2[256];                    //定义缓冲区
BYTE rptr;
BYTE wptr;
bit  over;

void main()
```

```
{
    int    i;

    sys_init();                                    //系统初始化
    usart1_spi_init();                             //USART1 使能 SPI 主模式初始化
    usart2_spi_init();                             //USART2 使能 SPI 从模式初始化
    EA = 1;

    for    (i=0; i<128; i++)
    {
        buffer1[i] = i;                            //初始化缓冲区
        buffer2[i] = 0;
    }
    test();

    while (1);
}

void uart1_isr() interrupt UART1_VECTOR
{
    if (TI)
    {
        TI = 0;

        if (rptr < 128)
        {
            SBUF = buffer1[rptr++];
        }
        else
        {
            over = 1;
        }
    }

    if (RI)
    {
        RI = 0;
    }
}

void uart2_isr() interrupt UART2_VECTOR
{
    if (S2TI)
    {
        S2TI = 0;
    }

    if (S2RI)
    {
        S2RI = 0;
        buffer2[wptr++] = S2BUF;
    }
}

void sys_init()
{
    WTST = 0x00;
    CKCON =  0x00;
    EAXFR =  1;
```

```
    P0M0 = 0x00; P0M1 = 0x00;
    P1M0 = 0x00; P1M1 = 0x00;
    P2M0 = 0x00; P2M1 = 0x00;
    P3M0 = 0x00; P3M1 = 0x00;
    P4M0 = 0x00; P4M1 = 0x00;
    P5M0 = 0x00; P5M1 = 0x00;
    P6M0 = 0x00; P6M1 = 0x00;
    P7M0 = 0x00; P7M1 = 0x00;

    P40 = 0;
    P6 = 0xff;
}

void usart1_spi_init()
{
    S1SPI_S0 = 1;                   //切换 S1SPI 到
                                    //P2.2/S1SS,P2.3/S1MOSI,P2.4/S1MISO,P2.5/S1SCLK
    S1SPI_S1 = 0;
    SCON = 0x10;                    //使能接收,必须设置为串口模式 0

    USARTCR1 = 0x10;                //使能 USART1 的 SPI 模式
//  USARTCR1 |= 0x40;               //DORD=1
    USARTCR1 &= ~0x40;              //DORD=0
//  USARTCR1 |= 0x04;               //从机模式
    USARTCR1 &= ~0x04;              //主机模式
    USARTCR1 |= 0x00;               //CPOL=0, CPHA=0
//  USARTCR1 |= 0x01;               //CPOL=0, CPHA=1
//  USARTCR1 |= 0x02 ;              //CPOL=1, CPHA=0
//  USARTCR1 |= 0x03;               //CPOL=1, CPHA=1
    USARTCR4 = 0x00;                //SPI 速度为 SYSCLK/4
//  USARTCR4 = 0x01;                //SPI 速度为 SYSCLK/8
//  USARTCR4 = 0x02;                //SPI 速度为 SYSCLK/16
//  USARTCR4 = 0x03;                //SPI 速度为 SYSCLK/2
    USARTCR1 |= 0x08;               //使能 SPI 功能

    ES = 1;
}

void usart2_spi_init()
{
    S2SPI_S0 = 1;                   //切换 S2SPI 到
                                    //P2.2/S2SS,P2.3/S2MOSI,P2.4/S2MISO,P2.5/S2SCLK
    S2SPI_S1 = 0;
    S2CON = 0x10;                   //使能接收,必须设置为串口模式 0

    USART2CR1 = 0x10;               //使能 USART2 的 SPI 模式
//  USART2CR1 |= 0x40;              //DORD=1
    USART2CR1 &= ~0x40;             //DORD=0
    USART2CR1 |= 0x04;              //从机模式
//  USART2CR1 &= ~0x04;             //主机模式
    USART2CR1 |= 0x00;              //CPOL=0, CPHA=0
//  USART2CR1 |= 0x01;              //CPOL=0, CPHA=1
//  USART2CR1 |= 0x02 ;             //CPOL=1, CPHA=0
//  USART2CR1 |= 0x03;              //CPOL=1, CPHA=1
    USART2CR4 = 0x00;               //SPI 速度为 SYSCLK/4
//  USART2CR4 = 0x01;               //SPI 速度为 SYSCLK/8
//  USART2CR4 = 0x02;               //SPI 速度为 SYSCLK/16
//  USART2CR4 = 0x03;               //SPI 速度为 SYSCLK/2
```

```
    USART2CR1 |= 0x08;                          //使能 SPI 功能

    ES2 = 1;
}

void test()
{
    BYTE i;
    BYTE ret;

    wptr = 0;
    rptr = 0;
    over = 0;

    S1SS = 0;
    SBUF = buffer1[rptr++];                     //启动数据传输
    while (!over);                              //等待 128 个数据传输完成
    S1SS = 1;

    ret    = 0x5a;
    for    (i=0; i<128; i++)
    {
        if (buffer1[i] != buffer2[i])           //校验数据
        {
            ret    = 0xfe;
            break;
        }
    }
    P6 = ret;                                   //P6=0x5a 表示数据传输正确
}
```

19.7.7　串口多机通讯（主机模式）

```
/*************  功能说明    **************
本例程基于 AI32G 核心实验板（屠龙刀）进行编写测试。
串口多机通信-主机参考程序。
串口 1(P1.6,P1.7) 设置为可变波特率 9 位数据模式。第 9 位数据作为地址帧(1)，数据帧(0) 的标志。
通过 USB-CDC 接口向 MCU 发送数据，MCU 将收到的数据从串口 1 发送出去，其中第一个字节数据作为从机地址，其它作为数据内容。
SM2 置 1 后，只能接收地址帧内容，自动过滤数据帧内容。如果需要接收指定地址从机返回的数据，需要在收到指定地址帧后将 SM2 置 0。
用定时器做波特率发生器，建议使用 1T 模式(除非低波特率用 12T)，并选择可被波特率整除的时钟频率，以提高精度。
此外程序演示两种复位进入 USB 下载模式的方法：
1. 通过每 1 毫秒执行一次“KeyResetScan”函数，实现长按 P3.2 口按键触发 MCU 复位，进入 USB 下载模式。
   (如果不希望复位进入 USB 下载模式的话，可在复位代码里将　IAP_CONTR 的 bit6 清 0，选择复位进用户程序区)
2. 通过加载“stc_usb_cdc_32g.lib”库函数，实现使用 AIapp-ISP 软件发送指令触发 MCU 复位，进入 USB 下载模式并自动下载。 (注意：使用 CDC 接口触发 MCU 复位并自动下载功能，需要勾选端口设置：下次强制使用”USB Writer (HID1)”进行 ISP 下载)
下载时，选择时钟 22.1184MHZ (用户可自行修改频率).
*****************************************/

#include "../../comm/AI32G.h"                   //包含此头文件后，不需要再包含"reg51.h"头文件
#include "../../comm/usb.h"                     //USB 调试及复位所需头文件

#define     MAIN_Fosc    22118400L              //定义主时钟（精确计算 115200 波特率）
#define     Timer0_Reload (65536UL -(MAIN_Fosc / 1000))     //Timer 0 中断频率, 1000 次/秒

#define     Baudrate1    115200L
#define     UART1_BUF_LENGTH 64

bit B_1ms;                                      //1ms 标志
bit B_TX1_Busy;                                 //发送忙标志
u8 TX1_Cnt;                                     //发送计数
u8 RX1_Cnt;                                     //接收计数
u8 RX1_TimeOut;

u8 RX1_Buffer[UART1_BUF_LENGTH];                //接收缓冲

//char *USER_DEVICEDESC = NULL;
//char *USER_PRODUCTDESC = NULL;
//char *USER_STCISPCMD = "@STCISP#";
//使用 20250319 之前版本的库函数时需要在用户代码中增加上面 3 行定义。
//使用 20250319 及以后的版本都不再需要上面的定义

//P3.2 口按键复位所需变量
bit    Key_Flag;
u16   Key_cnt;

void Timer0_config(void);
void UART1_config(u8 brt);                      //选择波特率,    2: 使用 Timer2 做波特率,
                                                //其它值: 使用 Timer1 做波特率.
void UART1_TxByte(u8 dat);
void UART1_Send(u8 addr,u8 *dat,u8 len);
void KeyResetScan(void);

//========================================================================
// 函数: void main(void)
// 描述: 主函数。
// 参数: none.
// 返回: none.
```

```
// 版本: VER1.0
// 备注:
//=========================================================================
void main(void)
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    RSTFLAG |= 0x04;                            //设置硬件复位后需要检测 P3.2 的状态选择运行区域，
                                                //否则硬件复位后进入 USB 下载模式

    P0M1 = 0x00;    P0M0 = 0x00;                //设置为准双向口
    P1M1 = 0x00;    P1M0 = 0x00;                //设置为准双向口
    P2M1 = 0x00;    P2M0 = 0x00;                //设置为准双向口
    P3M1 = 0x00;    P3M0 = 0x00;                //设置为准双向口
    P4M1 = 0x00;    P4M0 = 0x00;                //设置为准双向口
    P5M1 = 0x00;    P5M0 = 0x00;                //设置为准双向口
    P6M1 = 0x00;    P6M0 = 0x00;                //设置为准双向口
    P7M1 = 0x00;    P7M0 = 0x00;                //设置为准双向口

    usb_init();     //调用 USB 初始化函数，不需要立即判断电脑已正确识别到 USB 从设备
   /* USB 型单片机从设备，如需要主动向电脑发送数据，
      在执行 USB_SendData( )函数和 printf_usb( )函数时，
      这两个函数已增加了判断电脑是否已正确识别到 USB 从设备的程序。
      如果电脑要主动发送数据给 USB 从设备，电脑自己会主动判断与 USB 从设备是否已正确连接。
   */
    Timer0_config();
    UART1_config(1);                            //选择波特率, 2: 使用 Timer2 做波特率,
                                                //其它值: 使用 Timer1 做波特率.
    EUSB = 1    ;                               //IE2 相关的中断位操作使能后，需要重新设置 EUSB
    EA = 1;                                     //允许总中断

    while (1)
    {
        if(B_1ms)
        {
            B_1ms = 0;
            KeyResetScan();                     //P3.2 口按键触发软件复位，进入 USB 下载模式，
                                                //不需要此功能可删除本行代码

            if(RX1_TimeOut > 0)                 //超时计数
            {
                if(--RX1_TimeOut == 0)
                {
                    USB_SendData(RX1_Buffer,RX1_Cnt);   //把收到的数据通过 CDC 串口输出
                    RX1_Cnt   = 0;              //清除字节数
                }
            }
        }

        if (bUsbOutReady)
```

```
            {
                //UART1_Send(0x53,UsbOutBuffer,OutNumber);  //地址,数据,长度
                UART1_Send(UsbOutBuffer[0],&UsbOutBuffer[1],(u8)(OutNumber-1));       //地址,数据,长度

                //USB_SendData(UsbOutBuffer,OutNumber);     //发送数据缓冲区,长度

                usb_OUT_done();                             //接收应答（固定格式）
            }
        }
}

void timer0(void) interrupt 1
{
    B_1ms = 1;
}

//========================================================================
// 函数: void Timer0_config(void)
// 描述: Timer0 初始化函数。
// 参数: none.
// 返回: none.
// 版本: VER1.0
// 备注:
//========================================================================
void Timer0_config(void)
{
    TMOD &= 0xf0;                                   //16 bits timer auto-reload
    T0x12 = 1;                                      //Timer0 set as 1T
    TH0 = (u8)(Timer0_Reload / 256);
    TL0 = (u8)(Timer0_Reload % 256);
    ET0 = 1;                                        //Timer0 interrupt enable
    TR0 = 1;                                        //Tiner0 run
}

//========================================================================
// 函数: void UART1_TxByte(u8 dat)
// 描述:  发送一个字节.
// 参数:  无.
// 返回:  无.
//========================================================================
void UART1_TxByte(u8 dat)
{
    B_TX1_Busy = 1;
    SBUF = dat;
    while(B_TX1_Busy);
}

//========================================================================
// 函数: void UART1_Send(u8 addr,u8 *dat,u8 len)
// 描述:  发送一个字节.
// 参数:  无.
// 返回:  无.
//========================================================================
void UART1_Send(u8 addr,u8 *dat,u8 len)
{
    u8 i;

    TB8 = 1;                                        //地址帧
    UART1_TxByte(addr);
```

```
    TB8 = 0;                                    //数据帧
    For (i=0;i<len;i++)
    {
        UART1_TxByte(dat[i]);
    }
}

//========================================================================
// 函数: SetTimer2Baudraye(u32 dat)
// 描述: 设置 Timer2 做波特率发生器。
// 参数: dat: Timer2 的重装值.
// 返回: none.
// 版本: VER1.0
// 备注:
//========================================================================
void SetTimer2Baudraye(u32 dat)                 //选择波特率,  2: 使用 Timer2 做波特率,
                                                //其它值: 使用 Timer1 做波特率.
{
    T2R = 0;                                    //Timer stop
    T2_CT = 0;                                  //Timer2 set As Timer
    T2x12 = 1;                                  //Timer2 set as 1T mode
    T2H = (u8)(dat / 256);
    T2L = (u8)(dat % 256);
    ET2 = 0;                                    //禁止中断
    T2R = 1;                                    //Timer run enable
}

//========================================================================
// 函数: void UART1_config(u8 brt)
// 描述: UART1 初始化函数。
// 参数: brt: 选择波特率, 2: 使用 Timer2 做波特率, 其它值: 使用 Timer1 做波特率.
// 返回: none.
// 版本: VER1.0
// 备注:
//========================================================================
void UART1_config(u8 brt)                       //选择波特率,  2: 使用 Timer2 做波特率,
                                                //其它值: 使用 Timer1 做波特率.
{
    /********** 波特率使用定时器 2 **********/
    if(brt == 2)
    {
        S1BRT = 1;                              //S1 BRT Use Timer2;
        SetTimer2Baudraye(65536UL - (MAIN_Fosc / 4) / Baudrate1);
    }

    /********** 波特率使用定时器 1 **********/
    else
    {
        TR1  = 0;
        S1BRT = 0;                              //S1 BRT Use Timer1;
        T1_CT = 0;                              //Timer1 set As   Timer
        T1x12 = 1;                              //Timer1 set as   1T mode
        TMOD &= ~0x30;                          //Timer1_16bitAutoReload;
        TH1 = (u8)((65536UL - (MAIN_Fosc / 4) / Baudrate1) / 256);
        TL1 = (u8)((65536UL - (MAIN_Fosc / 4) / Baudrate1) % 256);
        ET1 = 0;                                //禁止中断
        TR1 = 1;
    }
```

```
    /*********************************************/

    SCON = (SCON & 0x3f) | 0xc0;                    //UART1 模式, 0x00: 同步移位输出,
                                                    //0x40: 8 位数据,可变波特率,
                                                    //0x80: 9 位数据,固定波特率,
                                                    //0xc0: 9 位数据,可变波特率
    SM2 = 1;                                        //允许多机通信
//  PS = 1;                                         //高优先级中断
    ES = 1;                                         //允许中断
    REN = 1;                                        //允许接收
    P_SW1 &= 0x3f;
    P_SW1 |= 0x80;                                  //UART1    switch to, 0x00: P3.0 P3.1,
                                                    //0x40: P3.6 P3.7, 0x80: P1.6       P1.7,
                                                    //0xC0:      P4.3 P4.4

    RX1_TimeOut = 0;
    B_TX1_Busy = 0;
    TX1_Cnt = 0;
    RX1_Cnt = 0;
}


//========================================================================
// 函数: void UART1_int (void) interrupt UART1_VECTOR
// 描述: UART1 中断函数。
// 参数: nine.
// 返回: none.
// 版本: VER1.0
// 备注:
//========================================================================
void UART1_int (void) interrupt 4
{
    if(RI)
    {
        RI = 0;
        RX1_Buffer[RX1_Cnt] = SBUF;
        if(++RX1_Cnt >= UART1_BUF_LENGTH)
            RX1_Cnt = 0;                            //防溢出
        RX1_TimeOut = 5;
    }

    if(TI)
    {
        TI = 0;
        B_TX1_Busy = 0;
    }
}

//========================================================================
// 函数: void delay_ms(u8 ms)
// 描述:  延时函数。
// 参数: ms,要延时的 ms 数,  这里只支持 1~255ms.  自动适应主时钟.
// 返回: none.
// 版本: VER1.0
// 备注:
//========================================================================
void delay_ms(u8 ms)
{
    u16 i;
```

```
    do{
        i = MAIN_Fosc / 6000;
        while(--i);                                 //6T per loop
    }while(--ms);
}

//========================================================================
// 函数: void KeyResetScan(void)
// 描述: P3.2 口按键长按 1 秒触发软件复位，进入 USB 下载模式。
// 参数: none.
// 返回: none.
// 版本: VER1.0
// 备注:
//========================================================================
void KeyResetScan(void)
{
    if(!P32)
    {
        if(!Key_Flag)
        {
            Key_cnt++;
            if(Key_cnt >= 1000)                     //连续 1000ms 有效按键检测
            {
                Key_Flag = 1;                       //设置按键状态，防止重复触发

                USBCON = 0x00;                      //清除 USB 设置
                USBCLK = 0x00;
                IRC48MCR = 0x00;

                delay_ms(10);
                IAP_CONTR = 0x60;                   //触发软件复位，从 ISP 开始执行
                while (1);
            }
        }
    }
    else
    {
        Key_cnt = 0;
        Key_Flag = 0;
    }
}
```

19.7.8　串口多机通讯（从机模式）

```
/************ 功能说明 **************
本例程基于 AI32G 核心实验板（屠龙刀）进行编写测试。
串口多机通信-从机参考程序。
串口 1(P1.6,P1.7)设置为可变波特率 9 位数据模式。第 9 位数据作为地址帧(1)，数据帧(0)的标志。
通过 SADDR 从机地址寄存器设置本机的从机地址，其中 0xFF 是广播地址。
对 SADEN 从机屏蔽地址寄存器进行配置，SADEN 置 1 的位所对应的从机地址位与主机发送的地址帧进行对比，只有匹配的地址帧才能触发串口中断。
例如：SADDR=0x53, SADEN=0xf0，那么只有高 4 位是"5"的地址帧才会触发从机的串口接收中断。
从机将串口接收到的内容通过 USB-CDC 接口对外发送，可通过串口助手打开 CDC 串口打印接收到的数据。
SM2 置 1 后，只能接收地址帧内容，自动过滤数据帧内容。需要接收数据时需要将 SM2 置 0，收完后再置 1。
用定时器做波特率发生器，建议使用 1T 模式(除非低波特率用 12T)，并选择可被波特率整除的时钟频率，以提高精度。
此外程序演示两种复位进入 USB 下载模式的方法：
1. 通过每 1 毫秒执行一次“KeyResetScan”函数，实现长按 P3.2 口按键触发 MCU 复位，进入 USB 下载模式。
   (如果不希望复位进入 USB 下载模式的话，可在复位代码里将　IAP_CONTR 的 bit6 清 0，选择复位进用户程序区)
```

```
2. 通过加载“stc_usb_cdc_32g.lib”库函数，实现使用 AIapp-ISP 软件发送指令触发 MCU 复位，进入 USB 下载模式并自动下载。　(注意：使用 CDC 接口触发 MCU 复位并自动下载功能，需要勾选端口设置：下次强制使用”USB Writer (HID1)”进行 ISP 下载)
下载时，选择时钟 22.1184MHZ (用户可自行修改频率).
*****************************************/

#include "../../comm/AI32G.h"                   //包含此头文件后，不需要再包含"reg51.h"头文件
#include "../../comm/usb.h"                     //USB 调试及复位所需头文件

#define     MAIN_Fosc    22118400L              //定义主时钟（精确计算 115200 波特率）
#define     Timer0_Reload (65536UL -(MAIN_Fosc / 1000))     //Timer  0  中断频率,  1000 次/秒

#define     Baudrate1 115200L
#define     UART1_BUF_LENGTH 64

bit B_1ms;                                      //1ms 标志
bit B_TX1_Busy;                                 //发送忙标志
u8 TX1_Cnt;                                     //发送计数
u8 RX1_Cnt;                                     //接收计数
u8 RX1_TimeOut;

u8    RX1_Buffer[UART1_BUF_LENGTH];             //接收缓冲

//char *USER_DEVICEDESC = NULL;
//char *USER_PRODUCTDESC = NULL;
//char *USER_STCISPCMD = "@STCISP#";
//使用 20250319 之前版本的库函数时需要在用户代码中增加上面 3 行定义。
//使用 20250319 及以后的版本都不再需要上面的定义

//P3.2 口按键复位所需变量
bit    Key_Flag;
u16   Key_cnt;

void Timer0_config(void);
void UART1_config(u8 brt);                      //选择波特率,  2: 使用 Timer2 做波特率,
                                                //其它值: 使用 Timer1 做波特率.
void UART1_TxByte(u8 dat);
void UART1_Send(u8 addr,u8 *dat,u8 len);
void KeyResetScan(void);

//========================================================================
// 函数: void main(void)
// 描述: 主函数。
// 参数: none.
// 返回: none.
// 版本: VER1.0
// 备注:
//========================================================================
void main(void)
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间
```

```
    RSTFLAG |= 0x04;                                //设置硬件复位后需要检测 P3.2 的状态选择运行区域，
                                                    //否则硬件复位后进入 USB 下载模式

    P0M1 = 0x00;    P0M0 = 0x00;                    //设置为准双向口
    P1M1 = 0x00;    P1M0 = 0x00;                    //设置为准双向口
    P2M1 = 0x00;    P2M0 = 0x00;                    //设置为准双向口
    P3M1 = 0x00;    P3M0 = 0x00;                    //设置为准双向口
    P4M1 = 0x00;    P4M0 = 0x00;                    //设置为准双向口
    P5M1 = 0x00;    P5M0 = 0x00;                    //设置为准双向口
    P6M1 = 0x00;    P6M0 = 0x00;                    //设置为准双向口
    P7M1 = 0x00;    P7M0 = 0x00;                    //设置为准双向口

    usb_init();       //调用 USB 初始化函数，不需要立即判断电脑已正确识别到 USB 从设备
    /* USB 型单片机从设备，如需要主动向电脑发送数据，
       在执行 USB_SendData( )函数和 printf_usb( )函数时，
       这两个函数已增加了判断电脑是否已正确识别到 USB 从设备的程序。
       如果电脑要主动发送数据给 USB 从设备，电脑自己会主动判断与 USB 从设备是否已正确连接。
    */

    Timer0_config();
    UART1_config(1);                                // 选择波特率, 2: 使用 Timer2 做波特率,
                                                    //其它值: 使用 Timer1 做波特率.
    EUSB = 1;                                       //IE2 相关的中断位操作使能后，需要重新设置 EUSB
    EA = 1;                                         //允许总中断

    while (1)
    {
        if(B_1ms)
        {
            B_1ms = 0;
            KeyResetScan();                         //P3.2 口按键触发软件复位，进入 USB 下载模式，
                                                    //不需要此功能可删除本行代码

            if(RX1_TimeOut > 0)                     //超时计数
            {
                if(--RX1_TimeOut == 0)
                {
                    SM2 = 1;                        //数据接收完毕，重新使能地址匹配
                    USB_SendData(RX1_Buffer,RX1_Cnt);       //把收到的数据通过 CDC 串口输出
                    RX1_Cnt   = 0;                  //清除字节数
                }
            }
        }

        if (bUsbOutReady)
        {
            //USB_SendData(UsbOutBuffer,OutNumber);       //发送数据缓冲区，长度
            UART1_Send(UsbOutBuffer[0],&UsbOutBuffer[1],(u8)(OutNumber-1));//地址,数据,长度

            usb_OUT_done();                         //接收应答（固定格式）
        }
    }
}

void timer0(void) interrupt 1
{
    B_1ms = 1;
```

```
}

//========================================================================
// 函数: void Timer0_config(void)
// 描述: Timer0 初始化函数。
// 参数: none.
// 返回: none.
// 版本: VER1.0
// 备注:
//========================================================================
void Timer0_config(void)
{
    TMOD &= 0xf0;                       //16 bits timer auto-reload
    T0x12 = 1;                          //Timer0 set as 1T
    TH0 = (u8)(Timer0_Reload / 256);
    TL0 = (u8)(Timer0_Reload % 256);
    ET0 = 1;                            //Timer0 interrupt enable
    TR0 = 1;                            //Tiner0 run
}

//========================================================================
// 函数: void UART1_TxByte(u8 dat)
// 描述: 发送一个字节.
// 参数: 无.
// 返回: 无.
// 版本: V1.0
//========================================================================
void UART1_TxByte(u8 dat)
{
    B_TX1_Busy = 1;
    SBUF = dat;
    while(B_TX1_Busy);
}

//========================================================================
// 函数: void UART1_Send(u8 addr,u8 *dat,u8 len)
// 描述: 发送一个字节.
// 参数: 无.
// 返回: 无.
// 版本: V1.0
//========================================================================
void UART1_Send(u8 addr,u8 *dat,u8 len)
{
    u8 i;

    TB8 = 1;                            //地址帧
    UART1_TxByte(addr);

    TB8 = 0;                            //数据帧
    For (i=0;i<len;i++)
    {
        UART1_TxByte(dat[i]);
    }
}

//========================================================================
// 函数: SetTimer2Baudraye(u32 dat)
// 描述: 设置 Timer2 做波特率发生器。
// 参数: dat: Timer2 的重装值.
```

```
// 返回: none.
// 版本: VER1.0
// 备注:
//=========================================================================
void SetTimer2Baudraye(u32 dat)                 //选择波特率,   2: 使用 Timer2 做波特率,
                                                //其它值: 使用 Timer1 做波特率.
{
    T2R = 0;                                    //Timer stop
    T2_CT = 0;                                  //Timer2 set As Timer
    T2x12 = 1;                                  //Timer2 set as 1T mode
    T2H = (u8)(dat / 256);
    T2L = (u8)(dat % 256);
    ET2 = 0;                                    //禁止中断
    T2R = 1;                                    //Timer run enable
}

//=========================================================================
// 函数: void UART1_config(u8 brt)
// 描述: UART1 初始化函数。
// 参数: brt: 选择波特率, 2: 使用 Timer2 做波特率, 其它值: 使用 Timer1 做波特率.
// 返回: none.
// 版本: VER1.0
// 备注:
//=========================================================================
void UART1_config(u8 brt)                       //选择波特率, 2: 使用 Timer2 做波特率,
                                                //其它值: 使用 Timer1 做波特率.
{
    /********** 波特率使用定时器 2 *****************/
    if(brt == 2)
    {
        S1BRT =   1;                            //S1 BRT Use Timer2;
        SetTimer2Baudraye(65536UL - (MAIN_Fosc / 4) / Baudrate1);
    }

    /********** 波特率使用定时器 1 *****************/
    else
    {
        TR1 = 0;
        S1BRT = 0;                              //S1 BRT Use Timer1;
        T1_CT = 0;                              //Timer1 set As Timer
        T1x12 = 1;                              //Timer1 set as 1T mode
        TMOD &= ~0x30;                          //Timer1_16bitAutoReload;
        TH1 = (u8)((65536UL - (MAIN_Fosc / 4) / Baudrate1) / 256);
        TL1 = (u8)((65536UL - (MAIN_Fosc / 4) / Baudrate1) % 256);
        ET1 = 0;                                //禁止中断
        TR1 = 1;
    }
    /*************************************************/

    SCON = (SCON & 0x3f) | 0xc0;                //UART1 模式, 0x00: 同步移位输出,
                                                //0x40: 8 位数据,可变波特率,
                                                //0x80: 9 位数据,固定波特率,
                                                //0xc0: 9 位数据,可变波特率
    SM2 = 1;                                    //允许多机通信
//  PS = 1;                                     //高优先级中断
    ES = 1;                                     //允许中断
    REN = 1;                                    //允许接收
    SADDR = 0x53;                               //从机地址
    SADEN = 0xf0;                               //高 4 位匹配
```

```
    P_SW1 &= 0x3f;
    P_SW1 |= 0x80;                              //UART1    switch to, 0x00: P3.0 P3.1,
                                                //0x40: P3.6 P3.7, 0x80: P1.6 P1.7,
                                                //0xC0: P4.3 P4.4

    RX1_TimeOut = 0;
    B_TX1_Busy = 0;
    TX1_Cnt = 0;
    RX1_Cnt  = 0;
}


//========================================================================
// 函数: void UART1_int (void) interrupt UART1_VECTOR
// 描述: UART1 中断函数。
// 参数: nine.
// 返回: none.
// 版本: VER1.0
// 备注:
//========================================================================
void UART1_int (void) interrupt 4
{
    if(RI)
    {
        RI = 0;
        if(RB8) SM2 = 0;                        //收到匹配地址，SM2 置 0 后接收后续数据

        RX1_Buffer[RX1_Cnt] = SBUF;
        if(++RX1_Cnt >= UART1_BUF_LENGTH)
            RX1_Cnt = 0;                        //防溢出
        RX1_TimeOut = 5;
    }

    if(TI)
    {
        TI = 0;
        B_TX1_Busy = 0;
    }
}

//========================================================================
// 函数: void delay_ms(u8 ms)
// 描述:  延时函数。
// 参数: ms,要延时的 ms 数,  这里只支持 1~255ms.  自动适应主时钟.
// 返回: none.
// 版本: VER1.0
// 备注:
//========================================================================
void delay_ms(u8 ms)
{
    u16  i;
    do{
        i = MAIN_Fosc / 6000;
        while(--i);                             //6T per loop
    }while(--ms);
}

//========================================================================
// 函数: void KeyResetScan(void)
```

```
// 描述: P3.2 口按键长按 1 秒触发软件复位，进入 USB 下载模式。
// 参数: none.
// 返回: none.
// 版本: VER1.0
// 备注:
//==========================================================================
void KeyResetScan(void)
{
    if(!P32)
    {
        if(!Key_Flag)
        {
            Key_cnt++;
            if(Key_cnt >= 1000)                 //连续 1000ms 有效按键检测
            {
                Key_Flag = 1;                   //设置按键状态，防止重复触发

                USBCON = 0x00;                  //清除 USB 设置
                USBCLK = 0x00;
                IRC48MCR = 0x00;

                delay_ms(10);
                IAP_CONTR = 0x60;               //触发软件复位，从 ISP 开始执行
                while (1);
            }
        }
    }
    else
    {
        Key_cnt = 0;
        Key_Flag = 0;
    }
}
```

19.8 深大实验--串口通讯(硬件 UART1 与电脑收发通讯)

19.8.1 实验介绍

1、 熟悉串口 1/UART1 通讯原理
2、 了解 Ai8051U 实验箱原理图
3、 熟悉如何管理多文件项目

详细实验内容及实验步骤，请查看：
https://www.stcaimcu.com/forum.php?mod=redirect&goto=findpost&ptid=15176&pid=166981

19.9 深大实验--串口通讯(硬件 UART1 与 UART2 收发通讯)

19.9.1 实验介绍

1、熟悉串口 1/UART1 与串口 2/UART2 通讯原理
2、了解 Ai8051U 实验箱原理图
3、熟悉如何管理多文件项目

详细实验内容及实验步骤，请查看：
https://www.stcaimcu.com/forum.php?mod=redirect&goto=findpost&ptid=15176&pid=166716

19.10 深大实验--串口通讯(硬件 UART1 与 UART2 数据透传)

19.10.1 实验介绍

1、 熟悉双串口数据透传的应用
2、 了解 Ai8051U 实验箱原理图与 AIapp-ISP 系统软件的应用
3、 熟悉如何管理多文件项目

详细实验内容及实验步骤，请查看：
https://www.stcaimcu.com/forum.php?mod=redirect&goto=findpost&ptid=15176&pid=166939

19.11 大型综合实战--深大课程设计：硬件 UART1 发送按键码，接收端蜂鸣器根据收到的键码发声

19.11.1 实验介绍

1、熟悉矩阵式按键扫描方法
2、熟悉串口 1/UART1 互联收发功能的使用

3、I/O 控制无源蜂鸣器发声，Key0 ~ Key7 按键按下，会发出对应的 Do/Re/Mi/Fa/So/La/Si/Do 音符
4、熟悉如何管理多文件项目

详细实验内容及实验步骤，请查看：
https://www.stcaimcu.com/forum.php?mod=redirect&goto=findpost&ptid=15176&pid=166545

19.12 深大实验--串口通讯(串口通讯，串口绘图，ADC 检测按键，ADC 热敏电阻测温)

19.12.1 实验介绍

1、本实验项目主要目的：
- 掌握串口通讯，了解串口绘图工具
- 继续熟悉 ADC 的应用，熟悉 ADC 检测按键，熟悉 ADC 热敏电阻（NTC）测温

2、掌握 Ai8051U 实验箱原理图中的串口通信部分电路，ADC 应用部分电路
3、继续熟悉 AIapp-ISP 系统软件中串口助手工具，了解串口绘图工具
4、继续熟悉如何管理多文件项目

详细实验内容及实验步骤，请查看：
https://www.stcaimcu.com/forum.php?mod=redirect&goto=findpost&ptid=15176&pid=167587

19.13 深大实验--串口通讯(硬件 UART1 与 UART2 收发通讯-奇偶校验)

19.13.1 实验介绍

1、 熟悉串口 1/UART 和串口 2/UART2 的通讯原理
2、 了解数据奇偶校验原理（本实验使用偶校验）
奇偶校验(Parity Check)是一种校验代码传输正确性的方法。根据被传输的一组二进制代码的数位中“1”的个数是奇数或偶数来进行校验。采用奇数的称为奇校验，反之，称为偶校验。采用何种校验是事先规定好的。通常专门设置一个奇偶校验位，用它反映这组代码中“1”的个数为奇数或偶数。
若用奇校验，则当接收端收到这组代码时，校验“1”的个数是否为奇数，从而确定传输代码的正确性。
若用偶校验，则当接收端收到这组代码时，校验“1”的个数是否为偶数，从而确定传输代码的正确性。
3、 了解 Ai8051U 实验箱原理图
4、 熟悉如何管理多文件项目

详细实验内容及实验步骤，请查看：
https://www.stcaimcu.com/forum.php?mod=redirect&goto=findpost&ptid=15176&pid=167709

19.14 深大实验--串口通讯(USB-CDC 发命令控制硬件 UART1 与 UART2 多机通讯)

19.14.1 实验介绍

1、 熟悉串口 1/UART1 和串口 2/UART2 的通讯原理
2、 了解多机通讯原理

多机通讯的过程如下：

① 置全部从机的 SM2=1，处于只接收地址帧状态。
② 主机首先发送呼叫地址帧信息，将第 9 位 TB8 设置为 1，以表示发送的是呼叫地址帧。
③ 所有从机接收到呼叫地址帧后，各自将接收到的主机呼叫的地址与本机的地址相比较：
- 若比较结果相等，则为被寻址从机，清除 SM2=0，准备接收从主机发送的数据帧，直至全部数据传输完；
- 若比较不相等，则为非寻址从机，仍维持 SM2=1 不变，对其后发来的数据帧不予理睬，即接收到的数据帧内容不装入 SBUF，不置位，RI=0，不会产生中断请求，直至被寻址为止。

④ 主机在发送完呼叫地址帧后，接着发送一连串的数据帧，其中的第 9 位 TB8=0，以表示为数据帧。只有那些 SM2 位为 0 的从机（即已接收地址帧的从机）才会接收数据帧。
⑤ 当主机改变从机通信时间则再发呼叫地址帧，寻呼其他从机，原先被寻址的从机经分析得知主机在寻呼其他从机时，恢复其 SM2=1，对其后主机发送的数据帧不予理睬。

上述过程均在软件控制下实现。

3、 了解 Ai8051U 实验箱原理图
4、 熟悉如何管理多文件项目

详细实验内容及实验步骤，请查看：
https://www.stcaimcu.com/forum.php?mod=redirect&goto=findpost&ptid=15176&pid=167867

19.15 大学实验--串口 1_2_3_4(中断设置标志位，主循环查询)，演示实验

```
/*演示串口 1、串口 2、串口 3、串口 4 中断同时打开时，
在各自的中断程序中设置中断事件标志位，
标志已产生相应的中断，在主循环中查询中断事件标志位，
来处理不是特急处理的任务，不堵塞其他中断*/

#include <AI8051U.H>          //包含 AI8051U 的头文件
#include "intrins.h"          //使用_nop_()函数所必须要包含的头文件,
                              //否则延时函数中调用的_nop_()函数没有被头文件引用过来，
                              //会导致编译器找不到这个而函数而报错。

unsigned char uart1_txflag = 0;//定义 1 个位变量，UART1 事件位变量标志，记录 UART1 已产生发送中断
// 供主循环查询 UART1 是否已产生发送中断，在主循环中处理 UART1 的中断事件任务，不堵塞其他中断
unsigned char uart1_rxflag = 0;       //定义 1 个位变量，UART1 事件位变量标志，记录 UART1 已产生接收中断
// 供主循环查询 UART1 是否已产生接收中断，在主循环中处理 UART1 的中断事件任务，不堵塞其他中断

unsigned char uart2_txflag = 0;//定义 1 个位变量，UART2 事件位变量标志，记录 UART2 已产生发送中断
// 供主循环查询 UART2 是否已产生发送中断，在主循环中处理 UART2 的中断事件任务，不堵塞其他中断
```

```
unsigned char uart2_rxflag = 0;       //定义 1 个位变量，UART2 事件位变量标志，记录 UART2 已产生接收中断
// 供主循环查询 UART2 是否已产生接收中断，在主循环中处理 UART2 的中断事件任务，不堵塞其他中断

unsigned char uart3_txflag = 0;//定义 1 个位变量，UART3 事件位变量标志，记录 UART3 已产生发送中断
// 供主循环查询 UART3 是否已产生发送中断，在主循环中处理 UART3 的中断事件任务，不堵塞其他中断
unsigned char uart3_rxflag = 0;       //定义 1 个位变量，UART3 事件位变量标志，记录 UART3 已产生接收中断
// 供主循环查询 UART3 是否已产生接收中断，在主循环中处理 UART3 的中断事件任务，不堵塞其他中断

unsigned char uart4_txflag = 0;//定义 1 个位变量，UART4 事件位变量标志，记录 UART4 已产生发送中断
// 供主循环查询 UART4 是否已产生发送中断，在主循环中处理 UART1 的中断事件任务，不堵塞其他中断
unsigned char uart4_rxflag = 0;       //定义 1 个位变量，UART4 事件位变量标志，记录 UART4 已产生接收中断
// 供主循环查询 UART4 是否已产生接收中断，在主循环中处理 UART4 的中断事件任务，不堵塞其他中断

void Uart1_Init(void)                 //115200bps@40.000MHz
{
    SCON = 0x50;                      //8 位数据,可变波特率
    AUXR |= 0x40;                     //定时器时钟 1T 模式
    AUXR &= 0xFE;                     //串口 1 选择定时器 1 为波特率发生器
    TMOD &= 0x0F;                     //设置定时器模式
    TL1 = 0xA9;                       //设置定时初始值
    TH1 = 0xFF;                       //设置定时初始值
    ET1 = 0;                          //禁止定时器中断
    TR1 = 1;                          //定时器 1 开始计时
    ES = 1;                           //使能串口 1 中断
}

void Uart2_Init(void)                 //115200bps@40.000MHz
{
    S2CON = 0x50;                     //8 位数据,可变波特率
    AUXR |= 0x04;                     //定时器时钟 1T 模式
    T2L = 0xA9;                       //设置定时初始值
    T2H = 0xFF;                       //设置定时初始值
    AUXR |= 0x10;                     //定时器 2 开始计时
    IE2 |= 0x01;                      //使能串口 2 中断
}

void Uart3_Init(void)                 //115200bps@40.000MHz
{
    S3CON = 0x10;                     //8 位数据,可变波特率
    S3CON |= 0x40;                    //串口 3 选择定时器 3 为波特率发生器
    T4T3M |= 0x02;                    //定时器时钟 1T 模式
    T3L = 0xA9;                       //设置定时初始值
    T3H = 0xFF;                       //设置定时初始值
    T4T3M |= 0x08;                    //定时器 3 开始计时
    IE2 |= 0x08;                      //使能串口 3 中断
}
```

```
void Uart4_Init(void)                   //115200bps@40.000MHz
{
    S4CON = 0x10;                       //8 位数据,可变波特率
    S4CON |= 0x40;                      //串口 4 选择定时器 4 为波特率发生器
    T4T3M |= 0x20;                      //定时器时钟 1T 模式
    T4L = 0xA9;                         //设置定时初始值
    T4H = 0xFF;                         //设置定时初始值
    T4T3M |= 0x80;                      //定时器 4 开始计时
    IE2 |= 0x10;                        //使能串口 4 中断
}

void main (void)
{
    EAXFR = 1;              //允许访问扩展的特殊寄存器，XFR
    WTST = 0;               //设置取程序代码等待时间，赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;              //设置访问片内的 xdata 速度，赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00; P0M1 = 0x00;       //设置 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x00;       //设置 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;       //设置 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x00;       //设置 P3 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x0c;       //P32、P33 设置为高阻输入(需要同步开启上拉电阻)
    P4M0 = 0x00; P4M1 = 0x00;       //设置 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;       //设置 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;       //设置 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;       //设置 P7 口为准双向口模式

    uart1_txflag = 0;               //初始化用户标志位
    uart1_rxflag = 0;               //初始化用户标志位
    uart2_txflag = 0;               //初始化用户标志位
    uart2_rxflag = 0;               //初始化用户标志位
    uart3_txflag = 0;               //初始化用户标志位
    uart3_rxflag = 0;               //初始化用户标志位
    uart4_txflag = 0;               //初始化用户标志位
    uart4_rxflag = 0;               //初始化用户标志位

    Uart1_Init();                   //调用 UART1 初始化函数
    Uart2_Init();                   //调用 UART2 初始化函数
    Uart3_Init();                   //调用 UART3 初始化函数
    Uart4_Init();                   //调用 UART4 初始化函数

    EA = 1;                         //总中断允许位打开
    P40 = 0;                        //打开 LED 灯供电
    while(1)                        //主循环中查询需要处理的各种事件
    {
```

```
/*  本演示程序中，主循环查询各中断有无需要继续处理的事件的次序，
依次是 UART/UART2/UART3/UART4, 用户可以自己根据实际情况，
调整查询各中断有无需要继续处理的事件的优先次序  */

if(uart1_txflag)
{   //主循环中查询，UART1 是否已产生发送中断，是否有需要处理的 UART1 发送事件
    uart1_txflag = 0;   //清 0，UART1 发送事件位变量标志
    _nop_();            //用户在此添加需要处理的事件
    _nop_();
}

if(uart1_rxflag)
{   //主循环中查询，UART1 是否已产生接收中断，是否有需要处理的 UART1 接收事件
    uart1_rxflag = 0;   //清 0，UART1 接收事件位变量标志
    _nop_();            //用户在此添加需要处理的事件
    _nop_();
}

if(uart2_txflag)
{   //主循环中查询，UART2 是否已产生发送中断，是否有需要处理的 UART2 发送事件
    uart2_txflag = 0;   //清 0，UART2 发送事件位变量标志
    _nop_();            //用户在此添加需要处理的事件
    _nop_();
}

if(uart2_rxflag)
{   //主循环中查询，UART2 是否已产生接收中断，是否有需要处理的 UART2 接收事件
    uart2_rxflag = 0;   //清 0，UART2 接收事件位变量标志
    _nop_();            //用户在此添加需要处理的事件
    _nop_();
}

if(uart3_txflag)
{   //主循环中查询，UART3 是否已产生发送中断，是否有需要处理的 UART3 发送事件
    uart3_txflag = 0;   //清 0，UART3 发送事件位变量标志
    _nop_();            //用户在此添加需要处理的事件
    _nop_();
}

if(uart3_rxflag)
{   //主循环中查询，UART3 是否已产生接收中断，是否有需要处理的 UART3 接收事件
    uart3_rxflag = 0;   //清 0，UART3 接收事件位变量标志
    _nop_();            //用户在此添加需要处理的事件
    _nop_();
}
```

```
        if(uart4_txflag)
        {   //主循环中查询，UART4 是否已产生发送中断，是否有需要处理的 UART4 发送事件
            uart4_txflag = 0;   //清 0，UART4 发送事件位变量标志
            _nop_();            //用户在此添加需要处理的事件
            _nop_();
        }

        if(uart4_rxflag)
        {   //主循环中查询，UART4 是否已产生接收中断，是否有需要处理的 UART4 接收事件
            uart4_rxflag = 0;   //清 0，UART4 接收事件位变量标志
            _nop_();            //用户在此添加需要处理的事件
            _nop_();
        }
    }
}

void Uart1_Isr(void) interrupt UART1_VECTOR
{
    if (TI)                     //检测串口 1 发送中断
    {
        TI = 0;                 //清除串口 1 发送中断请求位
        _nop_();                //特急处理
        //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
        //但时间不要太长，否则会影响其他中断事件的实时响应速度
        uart1_txflag = 1;
        // uart1_txflag 置 1 是通知主循环处理部分串口 1 发送中断事件不需要特急处理的任务
        //置 1，记录 UART1 已产生发送中断，供主循环查询判断有无需处理的 UART1 发送任务
    }
    if (RI)                     //检测串口 1 接收中断
    {
        RI = 0;                 //清除串口 1 接收中断请求位
        _nop_();                //特急处理
        //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
        //但时间不要太长，否则会影响其他中断事件的实时响应速度
        uart1_rxflag = 1;
        // uart1_rxflag 置 1 是通知主循环处理部分串口 1 接收中断事件不需要特急处理的任务
        //置 1，记录 UART1 已产生接收中断，供主循环查询判断有无需处理的 UART1 接收任务
    }
}
//UART1 中断服务程序，UART1_VECTOR 在 AI8051U.H 头文件中已宏定义为 4

void Uart2_Isr(void) interrupt UART2_VECTOR
{
    if (S2CON & 0x02)           //检测串口 2 发送中断
    {
        S2CON &= ~0x02;         //清除串口 2 发送中断请求位
```

```
            _nop_();                    //特急处理
            //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
            //但时间不要太长，否则会影响其他中断事件的实时响应速度
            uart2_txflag = 1;
            // uart2_txflag 置 1 是通知主循环处理部分串口 2 发送中断事件不需要特急处理的任务
            //置 1，记录 UART2 已产生发送中断，供主循环查询判断有无需处理的 UART2 发送任务
        }

        if (S2CON & 0x01)           //检测串口 2 接收中断
        {
            S2CON &= ~0x01;         //清除串口 2 接收中断请求位
            _nop_();                    //特急处理
            //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
            //但时间不要太长，否则会影响其他中断事件的实时响应速度
            uart2_rxflag = 1;
            // uart2_rxflag 置 1 是通知主循环处理部分串口 2 接收中断事件不需要特急处理的任务
            //置 1，记录 UART2 已产生接收中断，供主循环查询判断有无需处理的 UART2 接收任务
        }
}
//UART2 中断服务程序，UART2_VECTOR 在 AI8051U.H 头文件中已宏定义为 8

void Uart3_Isr(void) interrupt UART3_VECTOR
{
        if (S3CON & 0x02)           //检测串口 3 发送中断
        {
            S3CON &= ~0x02;         //清除串口 3 发送中断请求位
            _nop_();                    //特急处理
            //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
            //但时间不要太长，否则会影响其他中断事件的实时响应速度
            uart3_txflag = 1;
            // uart3_txflag 置 1 是通知主循环处理部分串口 3 发送中断事件不需要特急处理的任务
            //置 1，记录 UART3 已产生发送中断，供主循环查询判断有无需处理的 UART3 发送任务
        }

        if (S3CON & 0x01)           //检测串口 3 接收中断
        {
            S3CON &= ~0x01;         //清除串口 3 接收中断请求位
            _nop_();                    //特急处理
            //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
            //但时间不要太长，否则会影响其他中断事件的实时响应速度
            uart3_rxflag = 1;
            // uart3_rxflag 置 1 是通知主循环处理部分串口 3 接收中断事件不需要特急处理的任务
            //置 1，记录 UART3 已产生接收中断，供主循环查询判断有无需处理的 UART3 接收任务
        }
}
//UART3 中断服务程序，UART3_VECTOR 在 AI8051U.H 头文件中已宏定义为 17
```

```
void Uart4_Isr(void) interrupt UART4_VECTOR
{
    if (S4CON & 0x02)           //检测串口 4 发送中断
    {
        S4CON &= ~0x02;         //清除串口 4 发送中断请求位
        _nop_();                //特急处理
        //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
        //但时间不要太长，否则会影响其他中断事件的实时响应速度
        uart4_txflag = 1;
        // uart4_txflag 置 1 是通知主循环处理部分串口 4 发送中断事件不需要特急处理的任务
        //置 1，记录 UART4 已产生发送中断，供主循环查询判断有无需处理的 UART4 发送任务
    }

    if (S4CON & 0x01)           //检测串口 4 接收中断
    {
        S4CON &= ~0x01;         //清除串口 4 接收中断请求位
        _nop_();                //特急处理
        //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
        //但时间不要太长，否则会影响其他中断事件的实时响应速度
        uart4_rxflag = 1;
        // uart4_rxflag 置 1 是通知主循环处理部分串口 4 接收中断事件不需要特急处理的任务
        //置 1，记录 UART4 已产生接收中断，供主循环查询判断有无需处理的 UART4 接收任务
    }
}
//UART4 中断服务程序，UART4_VECTOR 在 AI8051U.H 头文件中已宏定义为 18
```

19.16 大学实验--串口 1_2_3_4 + 定时器 0_1_3_4 + 外部中断 INT0_1_2_3 (中断设置标志位，主循环查询)，演示实验

```
/*演示串口 1/串口 2/串口 3/串口 4 +  定时器 0/1/3/4 + INT0/1/2/3 中断同时打开时，
在各自的中断程序中设置中断事件标志位，
标志已产生相应的中断，在主循环中查询中断事件标志位，
来处理不是特急处理的任务，不堵塞其他中断*/

#include <AI8051U.H>            //包含 AI8051U 的头文件

#include "intrins.h"            //使用_nop_()函数所必须要包含的头文件,
                                //否则延时函数中调用的_nop_()函数没有被头文件引用过来，
                                //会导致编译器找不到这个而函数而报错。

unsigned char int0_flag = 0;  //定义 1 个位变量，INT0 事件位变量标志，记录 INT0 已产生中断
//  供主循环查询 INT0 是否已产生中断，在主循环中处理 INT0 的中断事件任务，不堵塞其他中断
unsigned char int1_flag = 0;  //定义 1 个位变量，INT1 事件位变量标志，记录 INT1 已产生中断
```

```
// 供主循环查询 INT1 是否已产生中断，在主循环中处理 INT1 的中断事件任务，不堵塞其他中断
unsigned char int2_flag = 0;  //定义 1 个位变量，INT2 事件位变量标志，记录 INT2 已产生中断
// 供主循环查询 INT2 是否已产生中断，在主循环中处理 INT2 的中断事件任务，不堵塞其他中断
unsigned char int3_flag = 0;  //定义 1 个位变量，INT3 事件位变量标志，记录 INT3 已产生中断
// 供主循环查询 INT3 是否已产生中断，在主循环中处理 INT3 的中断事件任务，不堵塞其他中断

unsigned char t0_flag = 0;          //定义 1 个位变量，T0 事件位变量标志，记录定时器 0 已产生中断
// 供主循环查询定时器 0 是否已产生中断，在主循环中处理定时器 0 的中断事件任务，不堵塞其他中断
unsigned char t1_flag = 0;          //定义 1 个位变量，T1 事件位变量标志，记录定时器 1 已产生中断
// 供主循环查询定时器 1 是否已产生中断，在主循环中处理定时器 1 的中断事件任务，不堵塞其他中断
unsigned char t3_flag = 0;          //定义 1 个位变量，T3 事件位变量标志，记录定时器 3 已产生中断
// 供主循环查询定时器 3 是否已产生中断，在主循环中处理定时器 3 的中断事件任务，不堵塞其他中断
unsigned char t4_flag = 0;          //定义 1 个位变量，T4 事件位变量标志，记录定时器 4 已产生中断
// 供主循环查询定时器 4 是否已产生中断，在主循环中处理定时器 4 的中断事件任务，不堵塞其他中断

unsigned char uart1_txflag = 0;    //定义 1 个位变量，UART1 事件位变量标志，记录 UART1 已产生发送中断
// 供主循环查询 UART1 是否已产生发送中断，在主循环中处理 UART1 的中断事件任务，不堵塞其他中断
unsigned char uart1_rxflag = 0;    //定义 1 个位变量，UART1 事件位变量标志，记录 UART1 已产生接收中断
// 供主循环查询 UART1 是否已产生接收中断，在主循环中处理 UART1 的中断事件任务，不堵塞其他中断

unsigned char uart2_txflag = 0;    //定义 1 个位变量，UART2 事件位变量标志，记录 UART2 已产生发送中断
// 供主循环查询 UART2 是否已产生发送中断，在主循环中处理 UART2 的中断事件任务，不堵塞其他中断
unsigned char uart2_rxflag = 0;    //定义 1 个位变量，UART2 事件位变量标志，记录 UART2 已产生接收中断
// 供主循环查询 UART2 是否已产生接收中断，在主循环中处理 UART2 的中断事件任务，不堵塞其他中断

unsigned char uart3_txflag = 0;    //定义 1 个位变量，UART3 事件位变量标志，记录 UART3 已产生发送中断
// 供主循环查询 UART3 是否已产生发送中断，在主循环中处理 UART3 的中断事件任务，不堵塞其他中断
unsigned char uart3_rxflag = 0;    //定义 1 个位变量，UART3 事件位变量标志，记录 UART3 已产生接收中断
// 供主循环查询 UART3 是否已产生接收中断，在主循环中处理 UART3 的中断事件任务，不堵塞其他中断

unsigned char uart4_txflag = 0;    //定义 1 个位变量，UART4 事件位变量标志，记录 UART4 已产生发送中断
// 供主循环查询 UART4 是否已产生发送中断，在主循环中处理 UART1 的中断事件任务，不堵塞其他中断
unsigned char uart4_rxflag = 0;    //定义 1 个位变量，UART4 事件位变量标志，记录 UART4 已产生接收中断
// 供主循环查询 UART4 是否已产生接收中断，在主循环中处理 UART4 的中断事件任务，不堵塞其他中断

void Timer0_Init(void)              //定时器 0 初始化，2 秒@40.000MHz
{
    TM0PS = 0x65;    //设置定时器时钟预分频 ( 注意:并非所有系列都有此寄存器,详情请查看数据手册 )
    AUXR &= 0x7F;               //定时器时钟 12T 模式
    TMOD &= 0xF0;               //设置定时器模式
    TL0 = 0xB1;                 //设置定时初始值
    TH0 = 0x00;                 //设置定时初始值
    TF0 = 0;                    //清除 TF0 标志
    TR0 = 1;                    //定时器 0 开始计时
    ET0 = 1;                    //使能定时器 0 中断
}
```

```
void Timer1_Init(void)          //定时器 1 初始化，500 毫秒@40.000MHz
{
    TM1PS = 0x19;    //设置定时器时钟预分频 ( 注意:并非所有系列都有此寄存器,详情请查看数据手册 )
    AUXR &= 0xBF;           //定时器时钟 12T 模式
    TMOD &= 0x0F;           //设置定时器模式
    TL1 = 0x99;             //设置定时初始值
    TH1 = 0x05;             //设置定时初始值
    TF1 = 0;                //清除 TF1 标志
    TR1 = 1;                //定时器 1 开始计时
    ET1 = 1;                //使能定时器 1 中断
}

void Timer3_Init(void)          //定时器 3 初始化，100 毫秒@40.000MHz
{
    TM3PS = 0x3D;    //设置定时器时钟预分频 ( 注意:并非所有系列都有此寄存器,详情请查看数据手册 )
    T4T3M |= 0x02;          //定时器时钟 1T 模式
    T3L = 0xFC;             //设置定时初始值
    T3H = 0x03;             //设置定时初始值
    T4T3M |= 0x08;          //定时器 3 开始计时
    IE2 |= 0x20;            //使能定时器 3 中断
}

void Timer4_Init(void)          //定时器 4 初始化，200 毫秒@40.000MHz
{
    TM4PS = 0x7A;    //设置定时器时钟预分频 ( 注意:并非所有系列都有此寄存器,详情请查看数据手册 )
    T4T3M |= 0x20;          //定时器时钟 1T 模式
    T4L = 0xEF;             //设置定时初始值
    T4H = 0x01;             //设置定时初始值
    T4T3M |= 0x80;          //定时器 4 开始计时
    IE2 |= 0x40;            //使能定时器 4 中断
}

void Uart1_Init(void)           //串口 1 初始化，115200bps@40.000MHz
{
    SCON = 0x50;            //8 位数据,可变波特率
    AUXR |= 0x01;           //串口 1 选择定时器 2 为波特率发生器
    AUXR |= 0x04;           //定时器时钟 1T 模式
    T2L = 0xA9;             //设置定时初始值
    T2H = 0xFF;             //设置定时初始值
    AUXR |= 0x10;           //定时器 2 开始计时
    ES = 1;                 //使能串口 1 中断
}

void Uart2_Init(void)           //串口 2 初始化，115200bps@40.000MHz
{
```

```
    S2CON = 0x50;               //8 位数据,可变波特率
    AUXR |= 0x04;               //定时器时钟 1T 模式
    T2L = 0xA9;                 //设置定时初始值
    T2H = 0xFF;                 //设置定时初始值
    AUXR |= 0x10;               //定时器 2 开始计时
    IE2 |= 0x01;                //使能串口 2 中断
}

void Uart3_Init(void)           //串口 3 初始化，115200bps@40.000MHz
{
    S3CON = 0x10;               //8 位数据,可变波特率
    S3CON &= 0xBF;              //串口 3 选择定时器 2 为波特率发生器
    AUXR |= 0x04;               //定时器时钟 1T 模式
    T2L = 0xA9;                 //设置定时初始值
    T2H = 0xFF;                 //设置定时初始值
    AUXR |= 0x10;               //定时器 2 开始计时
    IE2 |= 0x08;                //使能串口 3 中断
}

void Uart4_Init(void)           //串口 4 初始化，115200bps@40.000MHz
{
    S4CON = 0x10;               //8 位数据,可变波特率
    S4CON &= 0xBF;              //串口 4 选择定时器 2 为波特率发生器
    AUXR |= 0x04;               //定时器时钟 1T 模式
    T2L = 0xA9;                 //设置定时初始值
    T2H = 0xFF;                 //设置定时初始值
    AUXR |= 0x10;               //定时器 2 开始计时
    IE2 |= 0x10;                //使能串口 4 中断
}

void main (void)
{
    EAXFR = 1;      //允许访问扩展的特殊寄存器，XFR
    WTST = 0;       //设置取程序代码等待时间，赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;      //设置访问片内的 xdata 速度，赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00; P0M1 = 0x00;       //设置 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x00;       //设置 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;       //设置 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x00;       //设置 P3 口为准双向口模式
    P4M0 = 0x00; P4M1 = 0x00;       //设置 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;       //设置 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;       //设置 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;       //设置 P7 口为准双向口模式

    int0_flag = 0;                  //初始化用户标志位
```

```
    int1_flag = 0;              //初始化用户标志位
    int2_flag = 0;              //初始化用户标志位
    int3_flag = 0;              //初始化用户标志位

    t0_flag = 0;                //初始化用户标志位
    t1_flag = 0;                //初始化用户标志位
    t3_flag = 0;                //初始化用户标志位
    t4_flag = 0;                //初始化用户标志位

    uart1_txflag = 0;           //初始化用户标志位
    uart1_rxflag = 0;           //初始化用户标志位
    uart2_txflag = 0;           //初始化用户标志位
    uart2_rxflag = 0;           //初始化用户标志位
    uart3_txflag = 0;           //初始化用户标志位
    uart3_rxflag = 0;           //初始化用户标志位
    uart4_txflag = 0;           //初始化用户标志位
    uart4_rxflag = 0;           //初始化用户标志位

    IT0 = 0;                    //使能 INT0 上升沿和下降沿中断
    //IT0 = 1;                  //使能 INT0 下降沿中断
    EX0 = 1;                    //使能 INT0 中断
    IE0 = 0;                    //清 INT0 中断标志

    //IT1 = 0;                  //使能 INT1 上升沿和下降沿中断
    IT1 = 1;                    //使能 INT1 下降沿中断
    EX1 = 1;                    //使能 INT1 中断
    IE1 = 0;                    //清 INT1 中断标志
    INTCLKO |= 0x10;            //使能 INT2 中断

    INTCLKO |= 0x20;            //使能 INT3 中断

    Timer0_Init();              //调用定时器 0 初始化函数
    Timer1_Init();              //调用定时器 1 初始化函数
    Timer3_Init();              //调用定时器 3 初始化函数
    Timer4_Init();              //调用定时器 4 初始化函数

    Uart1_Init();               //调用 UART1 初始化函数
    Uart2_Init();               //调用 UART2 初始化函数
    Uart3_Init();               //调用 UART3 初始化函数
    Uart4_Init();               //调用 UART4 初始化函数

    EA = 1;                     //总中断允许位打开

    while(1)                    //主循环中查询需要处理的各种事件
    {
        /*  本演示程序中，主循环查询各中断有无需要继续处理的事件的次序，
```

```
依次是 INTx/TIMERx/UARTx, 用户可以自己根据实际情况，
调整查询各中断有无需要继续处理的事件的优先次序  */

//查询外部中断事件
if(int0_flag)           //主循环中查询，INT0 是否已产生中断，是否有需要处理的 INT0 事件
{
    int0_flag = 0;      //清 0，INT0 事件位变量标志
    _nop_();            //用户在此添加需要处理的事件
    _nop_();
}

if(int1_flag)           //主循环中查询，INT1 是否已产生中断，是否有需要处理的 INT1 事件
{
    int1_flag = 0;      //清 0，INT1 事件位变量标志
    _nop_();            //用户在此添加需要处理的事件
    _nop_();
}

if(int2_flag)           //主循环中查询，INT2 是否已产生中断，是否有需要处理的 INT2 事件
{
    int2_flag = 0;      //清 0，INT2 事件位变量标志
    _nop_();            //用户在此添加需要处理的事件
    _nop_();
}

if(int3_flag)           //主循环中查询，INT3 是否已产生中断，是否有需要处理的 INT3 事件
{
    int3_flag = 0;      //清 0，INT3 事件位变量标志
    _nop_();            //用户在此添加需要处理的事件
    _nop_();
}

//查询定时中断事件
if(t0_flag)     //主循环中查询，定时器 0 是否已产生中断，是否有需要处理的定时器 0 事件
{
    t0_flag = 0;        //清 0，T0 事件位变量标志
    _nop_();            //用户在此添加需要处理的事件
    _nop_();
}

if(t1_flag)     //主循环中查询，定时器 1 是否已产生中断，是否有需要处理的定时器 1 事件
{
    t1_flag = 0;        //清 0，T1 事件位变量标志
    _nop_();            //用户在此添加需要处理的事件
    _nop_();
}
```

```
if(t3_flag)     //主循环中查询，定时器 3 是否已产生中断，是否有需要处理的定时器 3 事件
{
    t3_flag = 0;        //清 0，T3 事件位变量标志
    _nop_();            //用户在此添加需要处理的事件
    _nop_();
}

if(t4_flag)     //主循环中查询，定时器 4 是否已产生中断，是否有需要处理的定时器 4 事件
{
    t4_flag = 0;        //清 0，T4 事件位变量标志
    _nop_();            //用户在此添加需要处理的事件
    _nop_();
}

//查询串口中断事件
if(uart1_txflag)    //主循环中查询，UART1 是否已产生发送中断，是否有需要处理的 UART1 发送事件
{
    uart1_txflag = 0;   //清 0，UART1 发送事件位变量标志
    _nop_();            //用户在此添加需要处理的事件
    _nop_();
}

if(uart1_rxflag)    //主循环中查询，UART1 是否已产生接收中断，是否有需要处理的 UART1 接收事件
{
    uart1_rxflag = 0;   //清 0，UART1 接收事件位变量标志
    _nop_();            //用户在此添加需要处理的事件
    _nop_();
}

if(uart2_txflag)    //主循环中查询，UART2 是否已产生发送中断，是否有需要处理的 UART2 发送事件
{
    uart2_txflag = 0;   //清 0，UART2 发送事件位变量标志
    _nop_();            //用户在此添加需要处理的事件
    _nop_();
}

if(uart2_rxflag)    //主循环中查询，UART2 是否已产生接收中断，是否有需要处理的 UART2 接收事件
{
    uart2_rxflag = 0;   //清 0，UART2 接收事件位变量标志
    _nop_();            //用户在此添加需要处理的事件
    _nop_();
}

if(uart3_txflag)    //主循环中查询，UART3 是否已产生发送中断，是否有需要处理的 UART3 发送事件
{
```

```
            uart3_txflag = 0;    //清 0，UART3 发送事件位变量标志
            _nop_();             //用户在此添加需要处理的事件
            _nop_();
        }

        if(uart3_rxflag)     //主循环中查询，UART3 是否已产生接收中断，是否有需要处理的 UART3 接收事件
        {
            uart3_rxflag = 0;    //清 0，UART3 接收事件位变量标志
            _nop_();             //用户在此添加需要处理的事件
            _nop_();
        }

        if(uart4_txflag)     //主循环中查询，UART4 是否已产生发送中断，是否有需要处理的 UART4 发送事件
        {
            uart4_txflag = 0;    //清 0，UART4 发送事件位变量标志
            _nop_();             //用户在此添加需要处理的事件
            _nop_();
        }

        if(uart4_rxflag)     //主循环中查询，UART4 是否已产生接收中断，是否有需要处理的 UART4 接收事件
        {
            uart4_rxflag = 0;    //清 0，UART4 接收事件位变量标志
            _nop_();             //用户在此添加需要处理的事件
            _nop_();
        }
    }
}
void int0_isr(void) interrupt INT0_VECTOR
{
    _nop_();             //特急处理，中断服务程序中尽量少执行长的任务，防止堵塞其他中断
    //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
    //但时间不要太长，否则会影响其他中断事件的实时响应速度
    int0_flag = 1;       // int0_flag 置 1 是通知主循环处理部分 INT0 中断事件不需要特急处理的任务
                         //置 1，记录 INT0 已产生中断，供主循环查询判断有无需处理的 INT0 任务

    if(INT0)             //边沿中断，进入后再次判断电平从而判断是什么样的电平
    {
        _nop_();         //判断为高电平，则当前为上升沿
        _nop_();         //可以在这里插入断点进行观察现象
    }
    else
    {
        _nop_();         //判断为低电平，则当前为下降沿
        _nop_();         //可以在这里插入断点进行观察现象
    }
}
```

```
//INT0 中断服务程序，INT0_VECTOR 在 AI8051U.H 头文件中已宏定义为 0

void int1_isr(void) interrupt INT1_VECTOR
{
    _nop_();            //特急处理，中断服务程序中尽量少执行长的任务，防止堵塞其他中断
    //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
    //但时间不要太长，否则会影响其他中断事件的实时响应速度
    int1_flag = 1;      // int1_flag 置 1 是通知主循环处理部分 INT1 中断事件不需要特急处理的任务
                        //置 1，记录 INT1 已产生中断，供主循环查询判断有无需处理的 INT1 任务
}
//INT1 中断服务程序，INT1_VECTOR 在 AI8051U.H 头文件中已宏定义为 2

void int2_isr(void) interrupt INT2_VECTOR
{
    _nop_();            //特急处理，中断服务程序中尽量少执行长的任务，防止堵塞其他中断
    //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
    //但时间不要太长，否则会影响其他中断事件的实时响应速度
    int2_flag = 1;      // int2_flag 置 1 是通知主循环处理部分 INT2 中断事件不需要特急处理的任务
                        //置 1，记录 INT2 已产生中断，供主循环查询判断有无需处理的 INT2 任务
}
//INT2 中断服务程序，INT2_VECTOR 在 AI8051U.H 头文件中已宏定义为 10

void int3_isr(void) interrupt INT3_VECTOR
{
    _nop_();            //特急处理，中断服务程序中尽量少执行长的任务，防止堵塞其他中断
    //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
    //但时间不要太长，否则会影响其他中断事件的实时响应速度
    int3_flag = 1;      // int3_flag 置 1 是通知主循环处理部分 INT3 中断事件不需要特急处理的任务
                        //置 1，记录 INT3 已产生中断，供主循环查询判断有无需处理的 INT3 任务
}
//INT3 中断服务程序，INT3_VECTOR 在 AI8051U.H 头文件中已宏定义为 11

void Timer0_Isr(void) interrupt TMR0_VECTOR      //定时器 0 中断服务程序
{
    _nop_();            //特急处理，中断服务程序中尽量少执行长的任务，防止堵塞其他中断
    //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
    //但时间不要太长，否则会影响其他中断事件的实时响应速度
    t0_flag = 1;        // t0_flag 置 1 是通知主循环处理部分 T0 中断事件不需要特急处理的任务
                        //置 1，记录定时器 0 已产生中断，供主循环查询判断有无需处理的定时器 0 任务
}
//定时器 0 中断服务程序，TMR0_VECTOR 在 AI8051U.H 头文件中已宏定义为 1

void Timer1_Isr(void) interrupt TMR1_VECTOR
{
    _nop_();            //特急处理，中断服务程序中尽量少执行长的任务，防止堵塞其他中断
    //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
```

```
    //但时间不要太长，否则会影响其他中断事件的实时响应速度
    t1_flag = 1;          // t1_flag 置 1 是通知主循环处理部分 T1 中断事件不需要特急处理的任务
                          //置 1，记录定时器 1 已产生中断，供主循环查询判断有无需处理的定时器 1 任务
}
//定时器 1 中断服务程序，TMR1_VECTOR 在 AI8051U.H 头文件中已宏定义为 3

void Timer3_Isr(void) interrupt TMR3_VECTOR
{
    _nop_();              //特急处理，中断服务程序中尽量少执行长的任务，防止堵塞其他中断
    //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
    //但时间不要太长，否则会影响其他中断事件的实时响应速度
    t3_flag = 1;          // t3_flag 置 1 是通知主循环处理部分 T3 中断事件不需要特急处理的任务
                          //置 1，记录定时器 3 已产生中断，供主循环查询判断有无需处理的定时器 3 任务
}
//定时器 3 中断服务程序，TMR3_VECTOR 在 AI8051U.H 头文件中已宏定义为 19

void Timer4_Isr(void) interrupt TMR4_VECTOR
{
    _nop_();              //特急处理，中断服务程序中尽量少执行长的任务，防止堵塞其他中断
    //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
    //但时间不要太长，否则会影响其他中断事件的实时响应速度
    t4_flag = 1;          // t4_flag 置 4 是通知主循环处理部分 T4 中断事件不需要特急处理的任务
                          //置 1，记录定时器 4 已产生中断，供主循环查询判断有无需处理的定时器 4 任务
}
//定时器 4 中断服务程序，TMR4_VECTOR 在 AI8051U.H 头文件中已宏定义为 20

void Uart1_Isr(void) interrupt UART1_VECTOR
{
    if (TI)               //检测串口 1 发送中断
    {
        TI = 0;           //清除串口 1 发送中断请求位
        _nop_();          //特急处理
        //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
        //但时间不要太长，否则会影响其他中断事件的实时响应速度
        uart1_txflag = 1; // uart1_txflag 置 1 是通知主循环处理部分串口 1 发送中断事件不需要特急处理的任务
                          //置 1，记录 UART1 已产生发送中断，供主循环查询判断有无需处理的 UART1 发送任务
    }
    if (RI)               //检测串口 1 接收中断
    {
        RI = 0;           //清除串口 1 接收中断请求位
        _nop_();          //特急处理
        //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
        //但时间不要太长，否则会影响其他中断事件的实时响应速度
        uart1_rxflag = 1; // uart1_rxflag 置 1 是通知主循环处理部分串口 1 接收中断事件不需要特急处理的任务
                          //置 1，记录 UART1 已产生接收中断，供主循环查询判断有无需处理的 UART1 接收任务
    }
```

```
}
//UART1 中断服务程序，UART1_VECTOR 在 AI8051U.H 头文件中已宏定义为 4

void Uart2_Isr(void) interrupt UART2_VECTOR
{
    if (S2CON & 0x02)           //检测串口 2 发送中断
    {
        S2CON &= ~0x02;         //清除串口 2 发送中断请求位
        _nop_();                //特急处理
        //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
        //但时间不要太长，否则会影响其他中断事件的实时响应速度
        uart2_txflag = 1;   // uart2_txflag 置 1 是通知主循环处理部分串口 2 发送中断事件不需要特急处理的任务
                            //置 1，记录 UART2 已产生发送中断，供主循环查询判断有无需处理的 UART2 发送任务
    }
    if (S2CON & 0x01)           //检测串口 2 接收中断
    {
        S2CON &= ~0x01;         //清除串口 2 接收中断请求位
        _nop_();                //特急处理
        //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
        //但时间不要太长，否则会影响其他中断事件的实时响应速度
        uart2_rxflag = 1;   // uart2_rxflag 置 1 是通知主循环处理部分串口 2 接收中断事件不需要特急处理的任务
                            //置 1，记录 UART2 已产生接收中断，供主循环查询判断有无需处理的 UART2 接收任务
    }
}
//UART2 中断服务程序，UART2_VECTOR 在 AI8051U.H 头文件中已宏定义为 8

void Uart3_Isr(void) interrupt UART3_VECTOR
{
    if (S3CON & 0x02)           //检测串口 3 发送中断
    {
        S3CON &= ~0x02;         //清除串口 3 发送中断请求位
        _nop_();                //特急处理
        //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
        //但时间不要太长，否则会影响其他中断事件的实时响应速度
        uart3_txflag = 1;   // uart3_txflag 置 1 是通知主循环处理部分串口 3 发送中断事件不需要特急处理的任务
                            //置 1，记录 UART3 已产生发送中断，供主循环查询判断有无需处理的 UART3 发送任务
    }
    if (S3CON & 0x01)           //检测串口 3 接收中断
    {
        S3CON &= ~0x01;         //清除串口 3 接收中断请求位
        _nop_();                //特急处理
        //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
        //但时间不要太长，否则会影响其他中断事件的实时响应速度
        uart3_rxflag = 1;   // uart3_rxflag 置 1 是通知主循环处理部分串口 3 接收中断事件不需要特急处理的任务
                            //置 1，记录 UART3 已产生接收中断，供主循环查询判断有无需处理的 UART3 接收任务
    }
```

```
}
//UART3 中断服务程序，UART3_VECTOR 在 AI8051U.H 头文件中已宏定义为 17

void Uart4_Isr(void) interrupt UART4_VECTOR
{
    if (S4CON & 0x02)           //检测串口 4 发送中断
    {
        S4CON &= ~0x02;         //清除串口 4 发送中断请求位
        _nop_();                //特急处理
        //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
        //但时间不要太长，否则会影响其他中断事件的实时响应速度
        uart4_txflag = 1;   // uart4_txflag 置 1 是通知主循环处理部分串口 4 发送中断事件不需要特急处理的任务
                            //置 1，记录 UART4 已产生发送中断，供主循环查询判断有无需处理的 UART4 发送任务
    }
    if (S4CON & 0x01)           //检测串口 4 接收中断
    {
        S4CON &= ~0x01;         //清除串口 4 接收中断请求位
        _nop_();                //特急处理
        //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
        //但时间不要太长，否则会影响其他中断事件的实时响应速度
        uart4_rxflag = 1;   // uart4_rxflag 置 1 是通知主循环处理部分串口 4 接收中断事件不需要特急处理的任务
                            //置 1，记录 UART4 已产生接收中断，供主循环查询判断有无需处理的 UART4 接收任务
    }
}
//UART4 中断服务程序，UART4_VECTOR 在 AI8051U.H 头文件中已宏定义为 18
```

20 异步串口通信（UART3、UART4）

产品线	异步串口数量
Ai8051U 系列	**2（U3~U4）**

Ai8051U 系列单片机具有 2 个全双工异步串行通信接口（UART3 和 UART4）。每个串行口由 2 个数据缓冲器、一个移位寄存器、一个串行控制寄存器和一个波特率发生器等组成。每个串行口的数据缓冲器由 2 个互相独立的接收、发送缓冲器构成，可以同时发送和接收数据。

Ai8051U 系列单片机的串口 3/串口 4 都有两种工作方式，这两种方式的波特率都是可变的。用户可用软件设置不同的波特率和选择不同的工作方式。主机可通过查询或中断方式对接收/发送进行程序处理，使用十分灵活。

串口 3、串口 4 的通讯口均可以通过功能管脚的切换功能切换到多组端口，从而可以将一个通讯口分时复用为多个通讯口。

20.1 串口功能脚切换

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW2	BAH	EAXFR	-	I2C_S[1:0]		CMPO_S	**S4_S**	**S3_S**	**S2_S**

S4_S：串口 4 功能脚选择位

S4_S	RxD4	TxD4
0	P0.2	P0.3
1	P5.2	P5.3

S3_S：串口 3 功能脚选择位

S3_S	RxD3	TxD3
0	P0.0	P0.1
1	P5.0	P5.1

20.2 串口相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
S3CON	串口 3 控制寄存器	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000
S3BUF	串口 3 数据寄存器	ADH									0000,0000
S4CON	串口 4 控制寄存器	FDH	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000,0000
S4BUF	串口 4 数据寄存器	FEH									0000,0000

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
UR3TOCR	串口 3 接收超时控制寄存器	7EFD78H	ENTO	ENTOI	SCALE	-	-	-	-	-	000x,xxxx
UR3TOSR	串口 3 接收超时状态寄存器	7EFD79H	CTOIF	-	-	-	-	-	-	TOIF	0xxx,xxx0
UR3TOTH	串口 3 接收超时长度寄存器	7EFD7AH	TM[15:8]								0000,0000
UR3TOTL	串口 3 接收超时长度寄存器	7EFD7BH	TM[7:0]								0000,0000
UR4TOCR	串口 4 接收超时控制寄存器	7EFD7CH	ENTO	ENTOI	SCALE	-	-	-	-	-	000x,xxxx
UR4TOSR	串口 4 接收超时状态寄存器	7EFD7DH	CTOIF	-	-	-	-	-	-	TOIF	0xxx,xxx0
UR4TOTH	串口 4 接收超时长度寄存器	7EFD7EH	TM[15:8]								0000,0000
UR4TOTL	串口 4 接收超时长度寄存器	7EFD7FH	TM[7:0]								0000,0000
UR3TOTE	串口 3 接收超时长度寄存器	7EFD8AH	TM[23:16]								0000,0000
UR4TOTE	串口 4 接收超时长度寄存器	7EFD8BH	TM[23:16]								0000,0000

20.3 串口 3（异步串口 UART3）

20.3.1 串口 3 控制寄存器（S3CON）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S3CON	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI

S3SM0：指定串口3的通信工作模式，如下表所示：

S3SM0	串口3工作模式	功能说明
0	模式0	可变波特率8位数据方式
1	模式1	可变波特率9位数据方式

S3ST3：选择串口 3 的波特率发生器

0：选择定时器 2 为串口 3 的波特率发生器

1：选择定时器 3 为串口 3 的波特率发生器

S3SM2：允许串口 3 在模式 1 时允许多机通信控制位。在模式 1 时，如果 S3SM2 位为 1 且 S3REN 位为 1，则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位（即 S3RB8）来筛选地址帧：若 S3RB8=1，说明该帧是地址帧，地址信息可以进入 S3BUF，并使 S3RI 为 1，进而在中断服务程序中再进行地址号比较；若 S3RB8=0，说明该帧不是地址帧，应丢掉且保持 S3RI=0。在模式 1 中，如果 S3SM2 位为 0 且 S3REN 位为 1，接收收机处于地址帧筛选被禁止状态。不论收到的 S3RB8 为 0 或 1，均可使接收到的信息进入 S3BUF，并使 S3RI=1，此时 S3RB8 通常为校验位。模式 0 为非多机通信方式，在这种方式时，要设置 S3SM2 应为 0。

S3REN：允许/禁止串口接收控制位

0：禁止串口接收数据

1：允许串口接收数据

S3TB8：当串口 3 使用模式 1 时，S3TB8 为要发送的第 9 位数据，一般用作校验位或者地址帧/数据帧标志位，按需要由软件置位或清 0。在模式 0 中，该位不用。

S3RB8：当串口 3 使用模式 1 时，S3RB8 为接收到的第 9 位数据，一般用作校验位或者地址帧/数据帧标志位。在模式 0 中，该位不用。

S3TI：串口 3 发送中断请求标志位。在停止位开始发送时由硬件自动将 S3TI 置 1，向 CPU 发请求中断，响应中断后 S3TI 必须用软件清零。

S3RI：串口 3 接收中断请求标志位。串行接收到停止位的中间时刻由硬件自动将 S3RI 置 1，向 CPU 发中断申请，响应中断后 S3RI 必须由软件清零。

20.3.2 串口 3 数据寄存器（S3BUF）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S3BUF	ADH								

S3BUF：串口 1 数据接收/发送缓冲区。S3BUF 实际是 2 个缓冲器，读缓冲器和写缓冲器，两个操作分别对应两个不同的寄存器，1 个是只写寄存器（写缓冲器），1 个是只读寄存器（读缓冲器）。对 S3BUF 进行读操作，实际是读取串口接收缓冲区，对 S3BUF 进行写操作则是触发串口开始发送数据。

20.3.3 串口 3 接收超时控制寄存器（UR3TOCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
UR3TOCR	7EFD78H	ENTO	ENTOI	SCALE	-	-	-	-	-

ENTO：串口 3 接收超时功能控制位

0：禁止串口 3 接收超时功能

1：使能串口 3 接收超时功能

ENTOI：串口 3 接收超时中断控制位

0：禁止串口 3 接收超时中断

1：使能串口 3 接收超时中断

SCALE：串口 3 超时计数时钟源选择

0：串口数据位率（波特率）

1：系统时钟

20.3.4 串口 3 超时状态寄存器（UR3TOSR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
UR3TOSR	7EFD79H	CTOIF	-	-	-	-	-	-	TOIF

CTOIF：写“1”清除串口超时中断标志位 TOIF。（只写）

TOIF：串口 3 超时中断请求标志位。（只读）

当发生串口 3 超时时，TOIF 会被硬件置“1”。如果 ENTOI 为 1 时会产生串口中断，中断入口地址为原串口 3 的中断入口地址。标志位需要软件向 CTOIF 位写“1”清零

20.3.5 串口 3 超时长度控制寄存器（UR3TOTE/H/L）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
UR3TOTE	7EFD8AH	TM[23:16]							
UR3TOTH	7EFD7AH	TM[15:8]							
UR3TOTL	7EFD7BH	TM[7:0]							

TM[23:0]：串口 3 超时时间控制位。

当串口 3 处于接收空闲状态时，内部超时计数器根据 SCALE 寄存器所选择的时钟源进行计数，当计数时间达到 TM 所设置的超时时间时，便会产生超时中断。当串口 3 接收数据完成时，复位内部超时计数器，重新进行超时计数。

注：

1、如果需要使能接收超时中断功能，则 TM 不可设置为 0，且 UR3TOTL、UR3TOTH、UR3TOTE 寄存器的设置必须先设置 UR3TOTL 和 UR3TOTH，最后设置 UR3TOTE

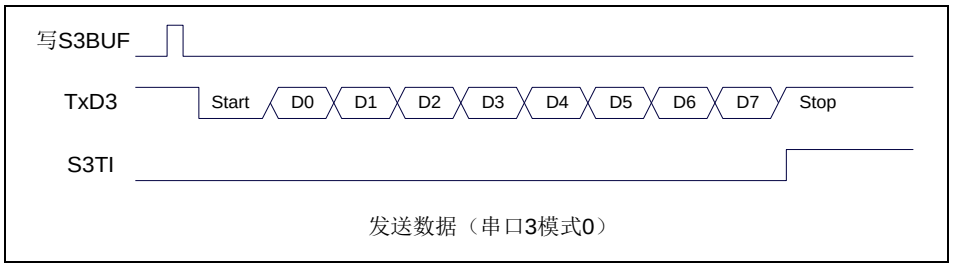
2、必须使能串口接收才有接收超时功能

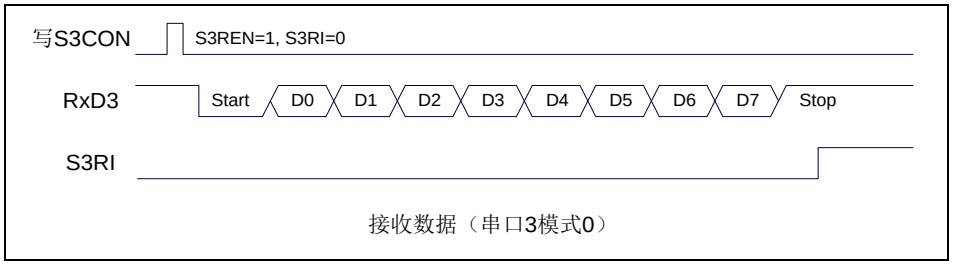
3、接收超时功能必须在正确接收到一字节数据后才能触发

4、正在接收过程中，无接收超时功能

20.3.6　串口 3 模式 0，模式 0 波特率计算公式

串行口 3 的模式 0 为 8 位数据位可变波特率 UART 工作模式。此模式一帧信息为 10 位：1 位起始位，8 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD3 为数据发送口，RxD3 为数据接收口，串行口全双工接受/发送。





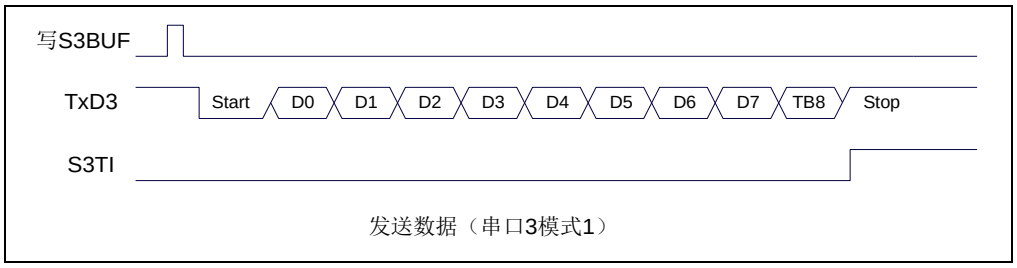
串口 3 的波特率是可变的，其波特率可由定时器 2 或定时器 3 产生。当定时器采用 1T 模式时（12 倍速），相应的波特率的速度也会相应提高 12 倍。

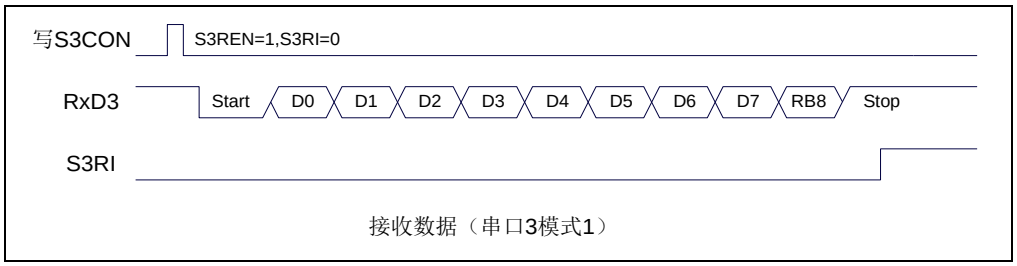
串口 3 模式 0 的波特率计算公式如下表所示：（SYSclk 为系统工作频率）

选择定时器	定时器速度	重装值计算公式	波特率
定时器2	1T	定时器2重载值 = $65536 - \frac{SYSclk}{4\times 波特率}$	波特率 = $\frac{SYSclk}{4\times(65536 - 定时器重装数)}$
	12T	定时器2重载值 = $65536 - \frac{SYSclk}{12\times 4\times 波特率}$	波特率 = $\frac{SYSclk}{12\times 4\times(65536 - 定时器重装数)}$
定时器3	1T	定时器3重载值 = $65536 - \frac{SYSclk}{4\times 波特率}$	波特率 = $\frac{SYSclk}{4\times(65536 - 定时器重装数)}$
	12T	定时器3重载值 = $65536 - \frac{SYSclk}{12\times 4\times 波特率}$	波特率 = $\frac{SYSclk}{12\times 4\times(65536 - 定时器重装数)}$

20.3.7　串口 3 模式 1，模式 1 波特率计算公式

串行口 3 的模式 1 为 9 位数据位可变波特率 UART 工作模式。此模式一帧信息为 11 位：1 位起始位，9 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD3 为数据发送口，RxD3 为数据接收口，串行口全双工接受/发送。





串口 3 模式 1 的波特率计算公式与模式 0 是完全相同的。请参考模式 0 的波特率计算公式。

20.4 串口 4（异步串口 UART4）

20.4.1 串口 4 控制寄存器（S4CON）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S4CON	FDH	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI

S4SM0：指定串口4的通信工作模式，如下表所示：

S4SM0	串口4工作模式	功能说明
0	模式0	可变波特率8位数据方式
1	模式1	可变波特率9位数据方式

S4ST4：选择串口 4 的波特率发生器

0：选择定时器 2 为串口 4 的波特率发生器

1：选择定时器 4 为串口 4 的波特率发生器

S4SM2：允许串口 4 在模式 1 时允许多机通信控制位。在模式 1 时，如果 S4SM2 位为 1 且 S4REN 位为 1，则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位（即 S4RB8）来筛选地址帧：若 S4RB8=1，说明该帧是地址帧，地址信息可以进入 S4BUF，并使 S4RI 为 1，进而在中断服务程序中再进行地址号比较；若 S4RB8=0，说明该帧不是地址帧，应丢掉且保持 S4RI=0。在模式 1 中，如果 S4SM2 位为 0 且 S4REN 位为 1，接收收机处于地址帧筛选被禁止状态。不论收到的 S4RB8 为 0 或 1，均可使接收到的信息进入 S4BUF，并使 S4RI=1，此时 S4RB8 通常为校验位。模式 0 为非多机通信方式，在这种方式时，要设置 S4SM2 应为 0。

S4REN：允许/禁止串口接收控制位

0：禁止串口接收数据

1：允许串口接收数据

S4TB8：当串口 4 使用模式 1 时，S4TB8 为要发送的第 9 位数据，一般用作校验位或者地址帧/数据帧标志位，按需要由软件置位或清 0。在模式 0 中，该位不用。

S4RB8：当串口 4 使用模式 1 时，S4RB8 为接收到的第 9 位数据，一般用作校验位或者地址帧/数据帧标志位。在模式 0 中，该位不用。

S4TI：串口 4 发送中断请求标志位。在停止位开始发送时由硬件自动将 S4TI 置 1，向 CPU 发请求中断，响应中断后 S4TI 必须用软件清零。

S4RI：串口 4 接收中断请求标志位。串行接收到停止位的中间时刻由硬件自动将 S4RI 置 1，向 CPU 发中断申请，响应中断后 S4RI 必须由软件清零。

20.4.2 串口 4 数据寄存器（S4BUF）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S4BUF	FEH								

S4BUF：串口 1 数据接收/发送缓冲区。S4BUF 实际是 2 个缓冲器，读缓冲器和写缓冲器，两个操作分别对应两个不同的寄存器，1 个是只写寄存器（写缓冲器），1 个是只读寄存器（读缓冲器）。对 S4BUF 进行读操作，实际是读取串口接收缓冲区，对 S4BUF 进行写操作则是触发串口开始发送数据。

20.4.3 串口 4 接收超时控制寄存器（UR4TOCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
UR4TOCR	7EFD7CH	ENTO	ENTOI	SCALE	-	-	-	-	-

ENTO：串口 4 接收超时功能控制位

0：禁止串口 4 接收超时功能

1：使能串口 4 接收超时功能

ENTOI：串口 4 接收超时中断控制位

0：禁止串口 4 接收超时中断

1：使能串口 4 接收超时中断

SCALE：串口 4 超时计数时钟源选择

0：串口数据位率（波特率）

1：系统时钟

20.4.4 串口 4 超时状态寄存器（UR4TOSR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
UR4TOSR	7EFD7DH	CTOIF	-	-	-	-	-	-	TOIF

CTOIF：写“1”清除串口超时中断标志位 TOIF。（只写）

TOIF：串口 4 超时中断请求标志位。（只读）

当发生串口 4 超时时，TOIF 会被硬件置“1”。如果 ENTOI 为 1 时会产生串口中断，中断入口地址为原串口 4 的中断入口地址。标志位需要软件向 CTOIF 位写“1”清零

20.4.5 串口 4 超时长度控制寄存器（UR4TOTE/H/L）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
UR4TOTE	7EFD8BH	**TM[23:16]**							
UR4TOTH	7EFD7EH	TM[15:8]							
UR4TOTL	7EFD7FH	TM[7:0]							

TM[23:0]：串口 4 超时时间控制位。

当串口 4 处于接收空闲状态时，内部超时计数器根据 SCALE 寄存器所选择的时钟源进行计数，当计数时间达到 TM 所设置的超时时间时，便会产生超时中断。当串口 4 接收数据完成时，复位内部超时计数器，重新进行超时计数。

注：

1、如果需要使能接收超时中断功能，则 TM 不可设置为 0，且 UR4TOTL、UR4TOTH、UR4TOTE 寄存器的设置必须先设置 UR4TOTL 和 UR4TOTH，最后设置 UR4TOTE

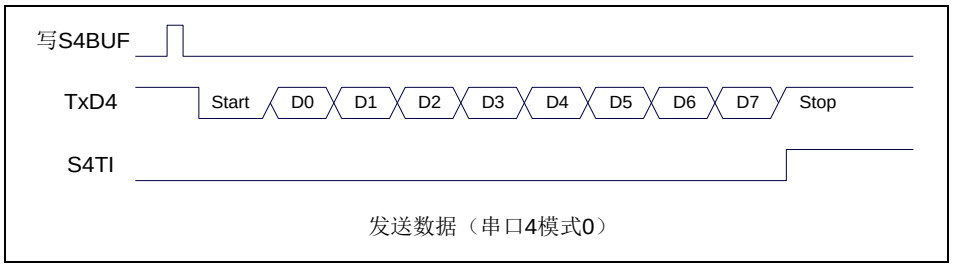
2、必须使能串口接收才有接收超时功能

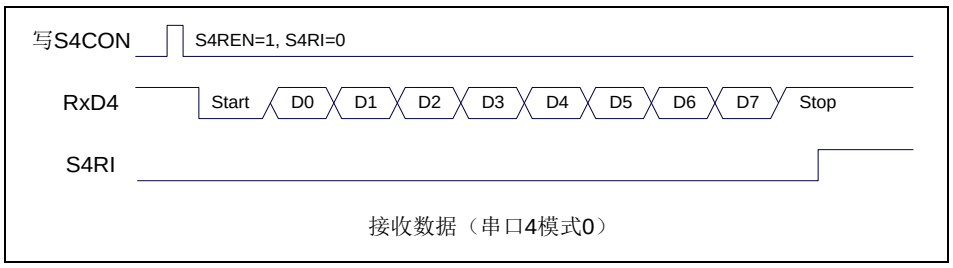
3、接收超时功能必须在正确接收到一字节数据后才能触发

4、正在接收过程中，无接收超时功能

20.4.6　串口 4 模式 0，模式 0 波特率计算公式

串行口 4 的模式 0 为 8 位数据位可变波特率 UART 工作模式。此模式一帧信息为 10 位：1 位起始位，8 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD4 为数据发送口，RxD4 为数据接收口，串行口全双工接受/发送。





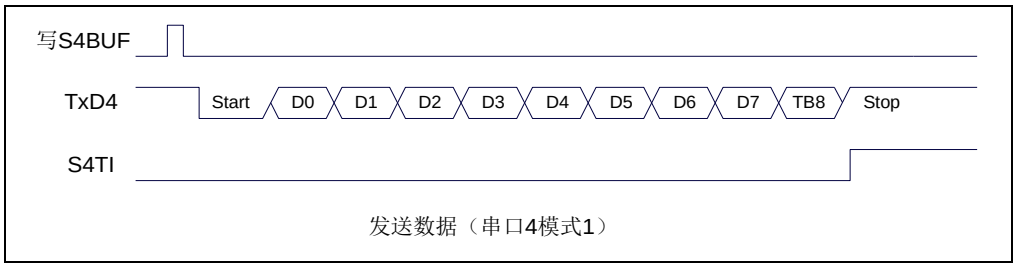
串口 4 的波特率是可变的，其波特率可由定时器 2 或定时器 4 产生。当定时器采用 1T 模式时（12 倍速），相应的波特率的速度也会相应提高 12 倍。

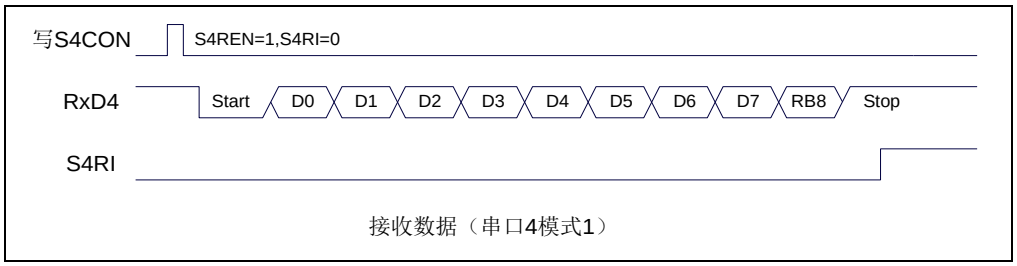
串口 4 模式 0 的波特率计算公式如下表所示：（SYSclk 为系统工作频率）

选择定时器	定时器速度	重装值计算公式	波特率
定时器2	1T	定时器2重载值 = $65536 - \frac{SYSclk}{4\times波特率}$	波特率 = $\frac{SYSclk}{4\times(65536 - 定时器重装数)}$
	12T	定时器2重载值 = $65536 - \frac{SYSclk}{12\times4\times波特率}$	波特率 = $\frac{SYSclk}{12\times4\times(65536 - 定时器重装数)}$
定时器4	1T	定时器4重载值 = $65536 - \frac{SYSclk}{4\times波特率}$	波特率 = $\frac{SYSclk}{4\times(65536 - 定时器重装数)}$
	12T	定时器4重载值 = $65536 - \frac{SYSclk}{12\times4\times波特率}$	波特率 = $\frac{SYSclk}{12\times4\times(65536 - 定时器重装数)}$

20.4.7　串口 4 模式 1，模式 1 波特率计算公式

串行口 4 的模式 1 为 9 位数据位可变波特率 UART 工作模式。此模式一帧信息为 11 位：1 位起始位，9 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD4 为数据发送口，RxD4 为数据接收口，串行口全双工接受/发送。



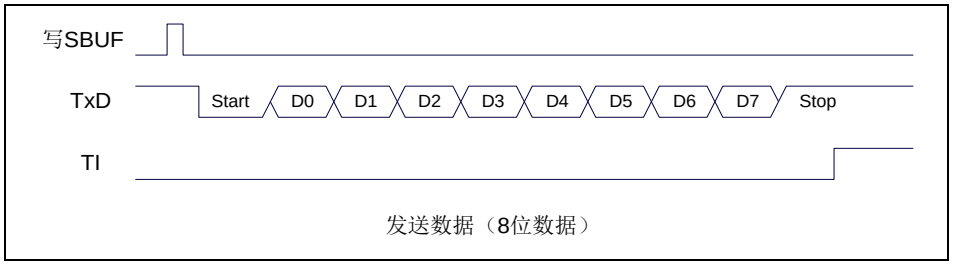


串口 4 模式 1 的波特率计算公式与模式 0 是完全相同的。请参考模式 0 的波特率计算公式。

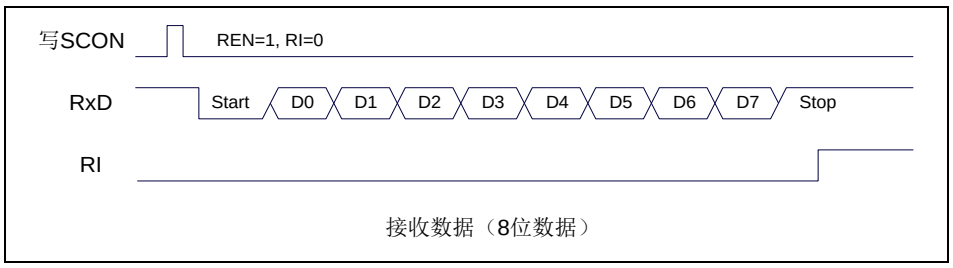
20.5 串口注意事项

关于串口中断请求有如下问题需要注意：（串口 1、串口 2、串口 3、串口 4 均类似，下面以串口 1 为例进行说明）

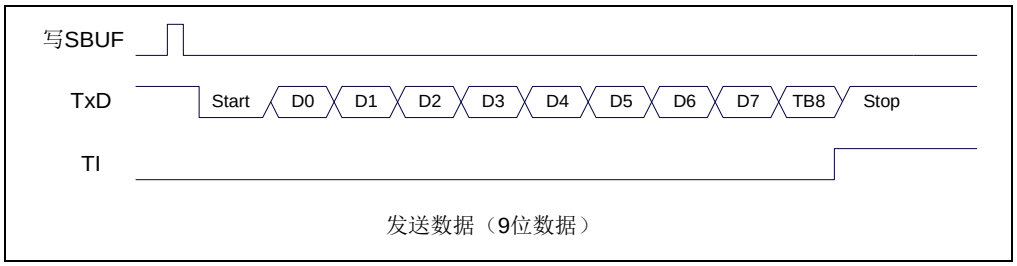
8 位数据模式时，发送完成整个停止位后产生 TI 中断请求，如下图所示：



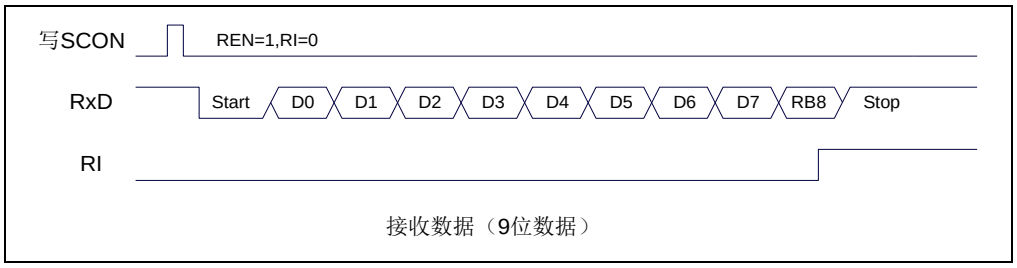
8 位数据模式时，接收完成一半个停止位后产生 RI 中断请求，如下图所示：



9 位数据模式时，发送完成整个第 9 位数据位后产生 TI 中断请求，如下图所示：



9 位数据模式时，接收完成一半个第 9 位数据位后产生 RI 中断请求，如下图所示：



20.6 范例程序

20.6.1 串口 3 使用定时器 2 做波特率发生器

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                  //头文件见下载软件
#include "intrins.h"                                  //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   FOSC        11059200UL                      //定义为无符号长整型,避免计算溢出
#define   BRT         (65536 - (FOSC / 115200+2) / 4)
                                                      //加 2 操作是为了让 Keil 编译器
                                                      //自动实现四舍五入运算

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void Uart3Isr() interrupt 17
{
    if (S3TI)
    {
        S3TI = 0;
        busy = 0;
    }
    if (S3RI)
    {
        S3RI = 0;
        buffer[wptr++] = S3BUF;
        wptr &= 0x0f;
    }
}

void Uart3Init()
{
    S3CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    T2x12 = 1;
    T2R = 1;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart3Send(char dat)
{
    while (busy);
    busy = 1;
    S3BUF = dat;
}

void Uart3SendStr(char *p)
{
    while (*p)
```

```
    {
        Uart3Send(*p++);
    }
}

void main()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart3Init();
    ES3 = 1;
    EA = 1;
    Uart3SendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart3Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

20.6.2　串口 3 使用定时器 3 做波特率发生器

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                           //头文件见下载软件
#include "intrins.h"                            //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   FOSC          11059200UL              //定义为无符号长整型,避免计算溢出
#define   BRT           (65536 - (FOSC / 115200+2) / 4)
                                                //加 2 操作是为了让 Keil 编译器
```

```
                                                    //自动实现四舍五入运算

bit         busy;
char        wptr;
char        rptr;
char        buffer[16];

void Uart3Isr() interrupt 17
{
    if (S3TI)
    {
        S3TI = 0;
        busy = 0;
    }
    if (S3RI)
    {
        S3RI = 0;
        buffer[wptr++] = S3BUF;
        wptr &= 0x0f;
    }
}

void Uart3Init()
{
    S3CON = 0x50;
    T3L = BRT;
    T3H = BRT >> 8;
    T3x12 = 1;
    T3R = 1;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart3Send(char dat)
{
    while (busy);
    busy = 1;
    S3BUF = dat;
}

void Uart3SendStr(char *p)
{
    while (*p)
    {
        Uart3Send(*p++);
    }
}

void main()
{
    EAXFR = 1;                                      //允许访问扩展的特殊寄存器，XFR
                                                    //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                  //允许访问扩展的特殊寄存器，XFR
                                                    //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                       //设置取程序代码等待时间，
                                                    //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                      //设置访问片内的 xdata 速度，
                                                    //赋值为 0 表示用最快速度访问，不增加额外的等待时间
```

```
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart3Init();
    ES3 = 1;
    EA = 1;
    Uart3SendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart3Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

20.6.3　串口 4 使用定时器 2 做波特率发生器

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件
#include "intrins.h"                          //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   FOSC        11059200UL              //定义为无符号长整型,避免计算溢出
#define   BRT         (65536 - (FOSC / 115200+2) / 4)
                                              //加 2 操作是为了让 Keil 编译器
                                              //自动实现四舍五入运算

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void Uart4Isr() interrupt 18
{
    if (S4TI)
    {
        S4TI = 0;
        busy = 0;
    }
    if (S4RI)
```

```
    {
        S4RI = 0;
        buffer[wptr++] = S4BUF;
        wptr &= 0x0f;
    }
}

void Uart4Init()
{
    S4CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    T2x12 = 1;
    T2R = 1;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart4Send(char dat)
{
    while (busy);
    busy = 1;
    S4BUF = dat;
}

void Uart4SendStr(char *p)
{
    while (*p)
    {
        Uart4Send(*p++);
    }
}

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                           //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart4Init();
```

```
    ES4 = 1;
    EA = 1;
    Uart4SendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart4Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

20.6.4　串口 4 使用定时器 4 做波特率发生器

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件
#include "intrins.h"                          //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   FOSC        11059200UL              //定义为无符号长整型,避免计算溢出
#define   BRT         (65536 - (FOSC / 115200+2) / 4)
                                              //加 2 操作是为了让 Keil 编译器
                                              //自动实现四舍五入运算

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void Uart4Isr() interrupt 18
{
    if (S4TI)
    {
        S4TI = 0;
        busy = 0;
    }
    if (S4RI)
    {
        S4RI = 0;
        buffer[wptr++] = S4BUF;
        wptr &= 0x0f;
    }
}

void Uart4Init()
{
    S4CON = 0x50;
    T4L = BRT;
    T4H = BRT >> 8;
    T4x12 = 1;
    T4R = 1;
    wptr = 0x00;
```

```
    rptr = 0x00;
    busy = 0;
}

void Uart4Send(char dat)
{
    while (busy);
    busy = 1;
    S4BUF = dat;
}

void Uart4SendStr(char *p)
{
    while (*p)
    {
        Uart4Send(*p++);
    }
}

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                           //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart4Init();
    ES4 = 1;
    EA = 1;
    Uart4SendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart4Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

21　比较器，掉电检测，内部固定比较电压

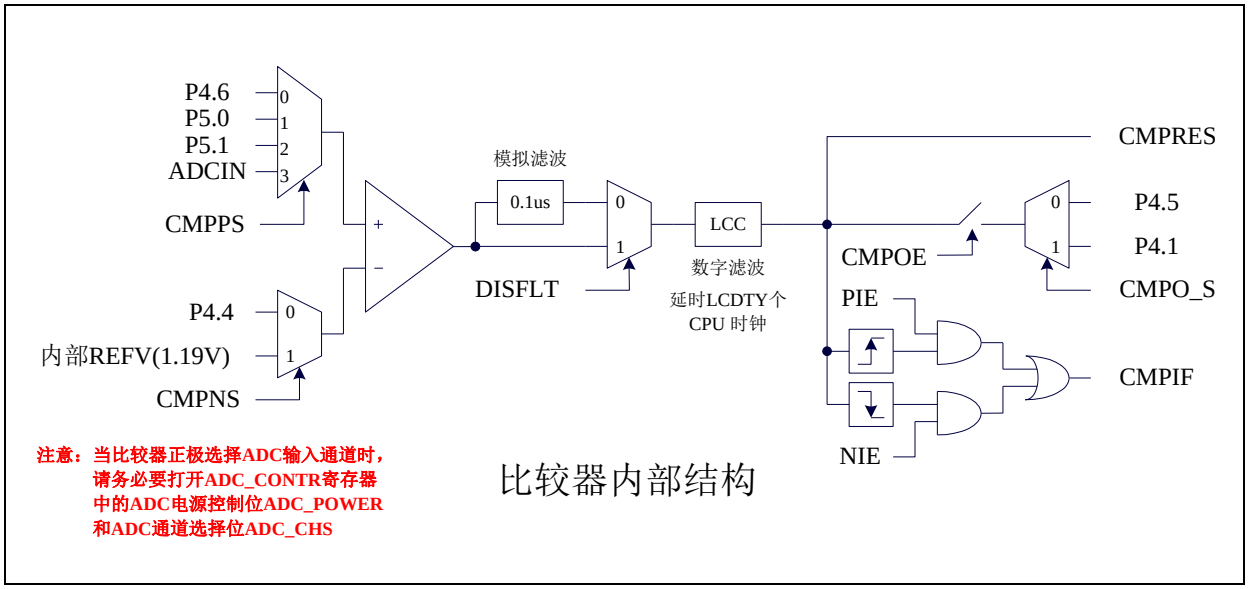
产品线	比较器（4P+2N）
Ai8051U 系列	●

Ai8051U 系列单片机内部集成了一个比较器。比较器的正极可以是 P4.6 端口、P5.0 端口、P5.1 端口或者 ADC 的模拟输入通道，而负极可以 P4.4 端口或者是内部 BandGap 经过 OP 后的 REFV 电压（内部固定比较电压）。通过多路选择器和分时复用可实现多个比较器的应用。

比较器内部有可程序控制的两级滤波：模拟滤波和数字滤波。模拟滤波可以过滤掉比较输入信号中的毛刺信号，数字滤波可以等待输入信号更加稳定后再进行比较。比较结果可直接通过读取内部寄存器位获得，也可将比较器结果正向或反向输出到外部端口。将比较结果输出到外部端口可用作外部事件的触发信号和反馈信号，可扩大比较的应用范围。

特别说明：由于 USB 的 UCAP 管脚占用了 P4.4 的 Pin 脚，所以目前的 LQFP48 和 LQFP44 脚都没有 P4.4 口，所以比较器的负端只能选择内部 1.19V 的参考电压

21.1　比较器内部结构图



21.2　比较器功能脚切换

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW2	BAH	EAXFR	-	I2C_S[1:0]		**CMPO_S**	S4_S	S3_S	S2_S

CMPO_S：比较器输出脚选择位

CMPO_S	CMPO
0	P4.5
1	P4.1

21.3 比较器相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CMPCR1	比较器控制寄存器 1	E6H	CMPEN	CMPIF	PIE	NIE	-	-	CMPOE	CMPRES	0000,xx00
CMPCR2	比较器控制寄存器 2	E7H	INVCMPO	DISFLT	LCDTY[5:0]						0000,0000
CMPEXCFG	比较器扩展配置寄存器	7EFEAEH	CHYS[1:0]		-	-	-	CMPNS	CMPPS[1:0]		00xx,x000

21.3.1 比较器控制寄存器 1（CMPCR1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR1	E6H	CMPEN	CMPIF	PIE	NIE	-	-	CMPOE	CMPRES

CMPEN：比较器模块使能位

0：关闭比较功能

1：使能比较功能

CMPIF：比较器中断标志位。当 PIE 或 NIE 被使能后，若产生相应的中断信号，硬件自动将 CMPIF 置 1，并向 CPU 提出中断请求。此标志位必须用户软件清零。

PIE：比较器上升沿中断使能位。

0：禁止比较器上升沿中断。

1：使能比较器上升沿中断。使能比较器的比较结果由 0 变成 1 时产生中断请求。

NIE：比较器下降沿中断使能位。

0：禁止比较器下降沿中断。

1：使能比较器下降沿中断。使能比较器的比较结果由 1 变成 0 时产生中断请求。

CMPOE：比较器结果输出控制位

0：禁止比较器结果输出

1：使能比较器结果输出。比较器结果输出到 P4.5 或者 P4.1（由 P_SW2 中的 CMPO_S 进行设定）

CMPRES：比较器的比较结果。此位为只读。

0：表示 CMP+的电平低于 CMP-的电平

1：表示 CMP+的电平高于 CMP-的电平

CMPRES 是经过数字滤波后的输出信号，而不是比较器的直接输出结果。

21.3.2　比较器控制寄存器 2（CMPCR2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR2	E7H	INVCMPO	DISFLT	LCDTY[5:0]					

INVCMPO：比较器结果输出控制

0：比较器结果正向输出。若 CMPRES 为 0，则 P4.5/P4.1 输出低电平，反之输出高电平。

1：比较器结果反向输出。若 CMPRES 为 0，则 P4.5/P4.1 输出高电平，反之输出低电平。

DISFLT：模拟滤波功能控制

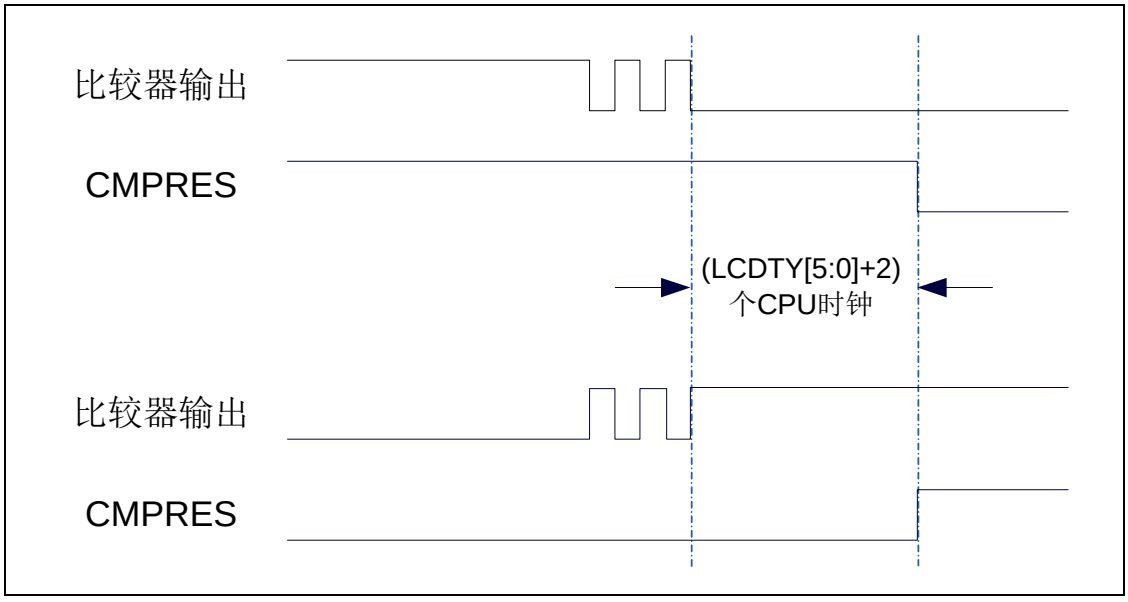
0：使能 0.1us 模拟滤波功能

1：关闭 0.1us 模拟滤波功能，可略微提高比较器的比较速度。

LCDTY[5:0]：数字滤波功能控制

数字滤波功能即为数字信号去抖动功能。当比较结果发生上升沿或者下降沿变化时，比较器侦测变化后的信号必须维持 LCDTY 所设置的 CPU 时钟数不发生变化，才认为数据变化是有效的；否则将视同信号无变化。

注意：当使能数字滤波功能后，芯片内部实际的等待时钟需额外增加两个状态机切换时间，即若 LCDTY 设置为 0 时，为关闭数字滤波功能；若 LCDTY 设置为非 0 值 n（n=1~63）时，则实际的数字滤波时间为（n+2）个系统时钟



21.3.3 比较器扩展配置寄存器（CMPEXCFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMPEXCFG	7EFEAEH	CHYS[1:0]		-	-	-	CMPNS	CMPPS[1:0]	

CHYS[1:0]：比较器 DC 迟滞输入选择

CHYS [1:0]	比较器 DC 迟滞输入选择
00	0mV
01	10mV
10	20mV
11	30mV

CMPNS：比较器负端输入选择

0：P4.4

1：选择内部 BandGap 经过 OP 后的电压 REFV 作为比较器负极输入源 （芯片在出厂时，内部参考电压调整为 **1.19V**）

CMPPS[1:0]：比较器正端输入选择

CMPPS[1:0]	比较器正端
00	P4.6
01	P5.0
10	P5.1
11	ADCIN

（注意 1：当比较器正极选择 ADC 输入通道时，请务必要打开 ADC_CONTR 寄存器中的 ADC 电源控制位 ADC_POWER 和 ADC 通道选择位 ADC_CHS）

21.4　范例程序

21.4.1　比较器的使用（中断方式）

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件
#include "intrins.h"                          //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void CMP_Isr() interrupt 21
{
    CMPIF = 0;                                //清中断标志
    if (CMPRES)
    {
        P10 = !P10;                           //下降沿中断测试端口
    }
    else
    {
        P11 = !P11;                           //上升沿中断测试端口
    }
}

void main()
{
    EAXFR = 1;                                //允许访问扩展的特殊寄存器，XFR
                                              //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                            //允许访问扩展的特殊寄存器，XFR
                                              //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                 //设置取程序代码等待时间，
                                              //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                //设置访问片内的 xdata 速度，
                                              //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CMPEXCFG = 0x00;
//  CMPEXCFG |= 0x40;                         //比较器 DC 迟滞输入选择
                                              //0:0mV; 0x40:10mV; 0x80:20mV; 0xc0:30mV

    CMPEXCFG &= ~0x03;                        //P4.6 为 CMP+输入脚
//  CMPEXCFG |= 0x01;                         //P5.0 为 CMP+输入脚
//  CMPEXCFG |= 0x02;                         //P5.1 为 CMP+输入脚
//  CMPEXCFG |= 0x03;                         //ADC 输入脚为 CMP+输入脚
    CMPEXCFG &= ~0x04;                        //P4.4 为 CMP-输入脚
//  CMPEXCFG |= 0x04;                         //内部 1.19V 参考电压为 CMP-输入脚
```

```
    CMPCR1 = 0x00;
    CMPCR2= 0x00;
    INVCMPO = 0;                    //比较器正向输出
//  INVCMPO = 1;                    //比较器反向输出
    DISFLT = 0;                     //使能 0.1us 滤波
//  DISFLT = 1;                     //禁止 0.1us 滤波
//  CMPCR2 &= ~0x3f;                //比较器结果直接输出
    CMPCR2 |= 0x10;                 //比较器结果经过 16 个去抖时钟后输出
    PIE = NIE = 1;                  //使能比较器边沿中断
//  PIE = 0;                        //禁止比较器上升沿中断
//  PIE = 1;                        //使能比较器上升沿中断
//  NIE = 0;                        //禁止比较器下降沿中断
//  NIE = 1;                        //使能比较器下降沿中断
//  CMPOE = 0;                      //禁止比较器输出
    CMPOE = 1;                      //使能比较器输出
    CMPEN = 1;                      //使能比较器模块

    EA = 1;

    while (1);
}
```

21.4.2　比较器的使用（查询方式）

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                 //头文件见下载软件
#include "intrins.h"                //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void main()
{
    EAXFR = 1;                      //允许访问扩展的特殊寄存器，XFR
                                    //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                  //允许访问扩展的特殊寄存器，XFR
                                    //(8 位模式请使用这句，注释上一句)
    WTST = 0;                       //设置取程序代码等待时间，
                                    //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                      //设置访问片内的 xdata 速度，
                                    //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;
```

```
    CMPEXCFG = 0x00;
//  CMPEXCFG |= 0x40;                   //比较器 DC 迟滞输入选择
                                        //0:0mV; 0x40:10mV; 0x80:20mV; 0xc0:30mV

    CMPEXCFG &= ~0x03;                  //P4.6 为 CMP+输入脚
//  CMPEXCFG |= 0x01;                   //P5.0 为 CMP+输入脚
//  CMPEXCFG |= 0x02;                   //P5.1 为 CMP+输入脚
//  CMPEXCFG |= 0x03;                   //ADC 输入脚为 CMP+输入脚
    CMPEXCFG &= ~0x04;                  //P4.4 为 CMP-输入脚
//  CMPEXCFG |= 0x04;                   //内部 1.19V 参考电压为 CMP-输入脚

    CMPCR1 = 0x00;
    CMPCR2= 0x00;
    INVCMPO = 0;                        //比较器正向输出
//  INVCMPO = 1;                        //比较器反向输出
    DISFLT = 0;                         //使能 0.1us 滤波
//  DISFLT = 1;                         //禁止 0.1us 滤波
//  CMPCR2 &= ~0x3f;                    //比较器结果直接输出
    CMPCR2 |= 0x10;                     //比较器结果经过 16 个去抖时钟后输出
    PIE = NIE = 1;                      //使能比较器边沿中断
//  PIE = 0;                            //禁止比较器上升沿中断
//  PIE = 1;                            //使能比较器上升沿中断
//  NIE = 0;                            //禁止比较器下降沿中断
//  NIE = 1;                            //使能比较器下降沿中断
//  CMPOE = 0;                          //禁止比较器输出
    CMPOE = 1;                          //使能比较器输出
    CMPEN = 1;                          //使能比较器模块

    while (1)
    {
        P10 = CMPRES;                   //读取比较器比较结果
    }
}
```

21.4.3　比较器的多路复用应用（比较器+ADC 输入通道）

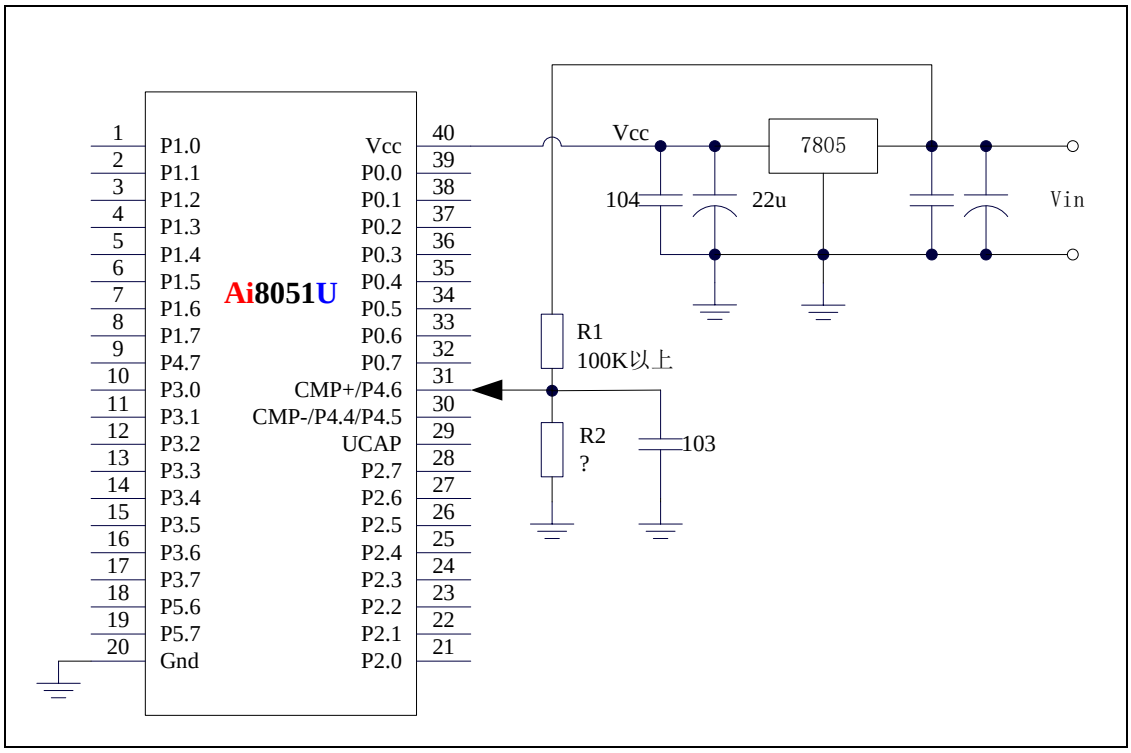
由于比较器的正极可以选择 ADC 的模拟输入通道，因此可以通过多路选择器和分时复用可实现多个比较器的应用。

注意：当比较器正极选择 ADC 输入通道时，请务必要打开 ADC_CONTR 寄存器中的 ADC 电源控制位 ADC_POWER 和 ADC 通道选择位 ADC_CHS

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                   //头文件见下载软件
#include "intrins.h"                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                           //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
```

```
                                                    //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P1M0 &= 0xfe;                                   //设置 P1.0 为输入口
    P1M1 |= 0x01;
    ADC_CONTR = 0x80;                               //使能 ADC 模块并选择 P1.0 为 ADC 输入脚

    CMPEXCFG = 0x00;
//  CMPEXCFG &= ~0x03;                              //P4.6 为 CMP+输入脚
//  CMPEXCFG |= 0x01;                               //P5.0 为 CMP+输入脚
//  CMPEXCFG |= 0x02;                               //P5.1 为 CMP+输入脚
    CMPEXCFG |= 0x03;                               //ADC 输入脚为 CMP+输入脚
    CMPEXCFG &= ~0x04;                              //P4.4 为 CMP-输入脚
//  CMPEXCFG |= 0x04;                               //内部 1.19V 参考电压为 CMP-输入脚

    CMPCR2 = 0x00;
    CMPCR1 = 0x00;
    CMPOE = 1;                                      //使能比较器输出
    CMPEN = 1;                                      //使能比较器模块

    while (1);
}
```

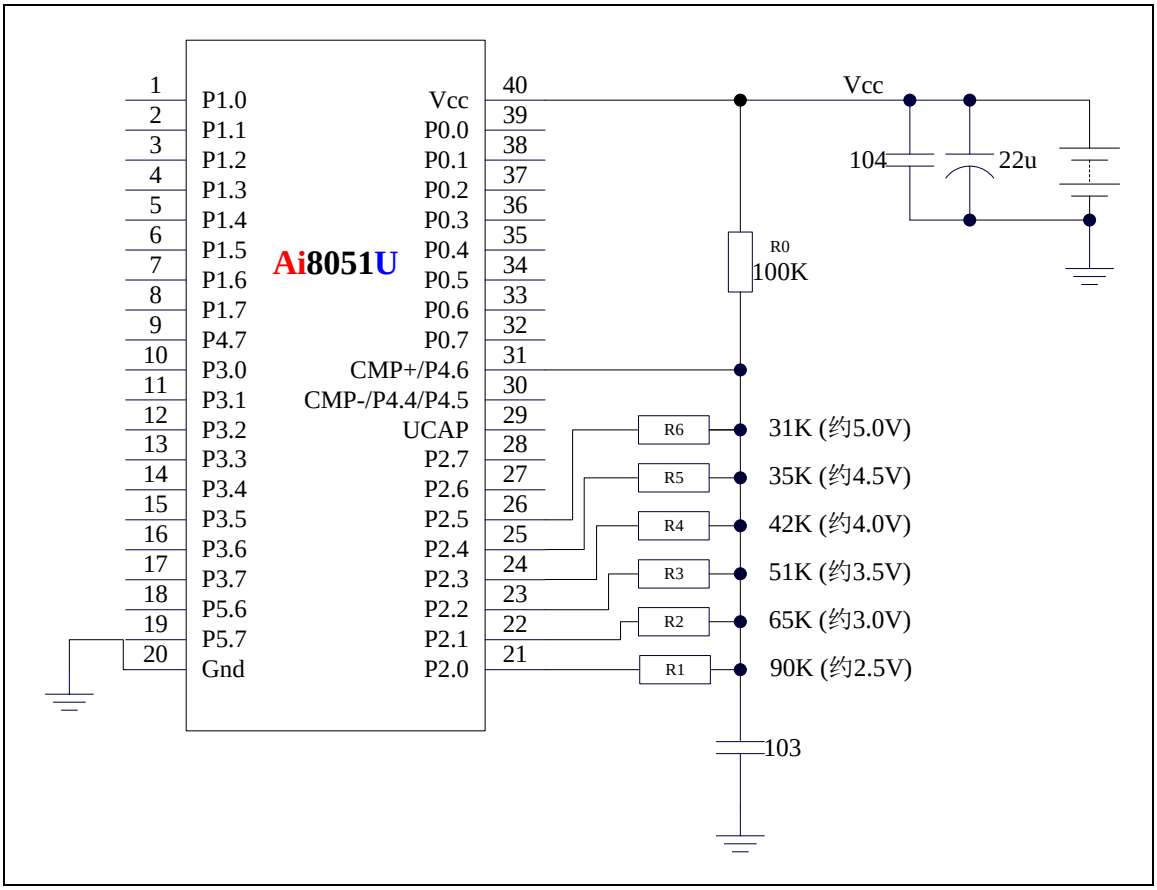
21.4.4 比较器作外部掉电检测(掉电过程中应及时保存用户数据到 EEPROM 中)



上图中电阻 R1 和 R2 对稳压块 7805 的前端电压进行分压，分压后的电压作为比较器 CMP+的外部输入与内部 1.19V 参考信号源进行比较。

一般当交流电在 220V 时，稳压块 7805 前端的直流电压为 11V，但当交流电压降到 160V 时，稳压块 7805 前端的直流电压为 8.5V。当稳压块 7805 前端的直流电压低于或等于 8.5V 时，该前端输入的直流电压被电阻 R1 和 R2 分压到比较器正极输入端 CMP+，CMP+端输入电压低于内部 1.19V 参考信号源，此时可产生比较器中断，这样在掉电检测时就有充足的时间将数据保存到 EEPROM 中。当稳压块 7805 前端的直流电压高于 8.5V 时，该前端输入的直流电压被电阻 R1 和 R2 分压到比较器正极输入端 CMP+，CMP+端输入电压高于内部 1.19V 参考信号源，此时 CPU 可继续正常工作。

21.4.5　比较器检测工作电压（电池电压）



上图中，利用电阻分压的原理可以近似的测量出 MCU 的工作电压（选通的通道，MCU 的 IO 口输出低电平，端口电压值接近 Gnd，未选通的通道，MCU 的 IO 口输出开漏模式的高，不影响其他通道）。

比较器的负端选择内部 1.19V 参考信号源，正端选择通过电阻分压后输入到 CMP+管脚的电压值。

初始化时 P2.5~P2.0 口均设置为开漏模式，并输出高。首先 P2.0 口输出低电平，此时若 Vcc 电压低于 2.5V 则比较器的比较值为 0，反之若 Vcc 电压高于 2.5V 则比较器的比较值为 1；

若确定 Vcc 高于 2.5V，则将 P2.0 口输出高，P2.1 口输出低电平，此时若 Vcc 电压低于 3.0V 则比较器的比较值为 0，反之若 Vcc 电压高于 3.0V 则比较器的比较值为 1；

若确定 Vcc 高于 3.0V，则将 P2.1 口输出高，P2.2 口输出低电平，此时若 Vcc 电压低于 3.5V 则比较器的比较值为 0，反之若 Vcc 电压高于 3.5V 则比较器的比较值为 1；

若确定 Vcc 高于 3.5V，则将 P2.2 口输出高，P2.3 口输出低电平，此时若 Vcc 电压低于 4.0V 则比较器的比较值为 0，反之若 Vcc 电压高于 4.0V 则比较器的比较值为 1；

若确定 Vcc 高于 4.0V，则将 P2.3 口输出高，P2.4 口输出低电平，此时若 Vcc 电压低于 4.5V 则比较器的比较值为 0，反之若 Vcc 电压高于 4.5V 则比较器的比较值为 1；

若确定 Vcc 高于 4.5V，则将 P2.4 口输出高，P2.5 口输出低电平，此时若 Vcc 电压低于 5.0V 则比较器的比较值为 0，反之若 Vcc 电压高于 5.0V 则比较器的比较值为 1。

```
//测试工作频率为11.0592MHz

#include “Ai8051U.H”                    //头文件见下载软件
#include "intrins.h"                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。
```

```
void delay ()
{
    char i;

    for (i=0; i<20; i++);
}

void main()
{
    unsigned char v;

    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                           //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P2M0 = 0x3f;                        //P2.5~P2.0 初始化为开漏模式
    P2M1 = 0x3f;
    P2 = 0xff;

    CMPEXCFG = 0x00;
    CMPEXCFG &= ~0x03;                  //P4.6 为 CMP+输入脚
//  CMPEXCFG |= 0x01;                   //P5.0 为 CMP+输入脚
//  CMPEXCFG |= 0x02;                   //P5.1 为 CMP+输入脚
//  CMPEXCFG |= 0x03;                   //ADC 输入脚为 CMP+输入脚
//  CMPEXCFG &= ~0x04;                  //P4.4 为 CMP-输入脚
    CMPEXCFG |= 0x04;                   //内部 1.19V 参考电压为 CMP-输入脚

    CMPCR2 = 0x10;                      //比较器结果经过 16 个去抖时钟后输出
    CMPCR1 = 0x00;
    CMPEN = 1;                          //使能比较器模块

    while (1)
    {
        v = 0x00;                       //电压<2.5V
        P2 = 0xfe;                      //P2.0 输出 0
        delay();
        if (!(CMPRES)) goto ShowVol;
        v = 0x01;                       //电压>2.5V
        P2 = 0xfd;                      //P2.1 输出 0
        delay();
```

```
            if (!(CMPRES)) goto ShowVol;
            v = 0x03;                                   //电压>3.0V
            P2 = 0xfb;                                  //P2.2 输出 0
            delay();
            if (!(CMPRES)) goto ShowVol;
            v = 0x07;                                   //电压>3.5V
            P2 = 0xf7;                                  //P2.3 输出 0
            delay();
            if (!(CMPRES)) goto ShowVol;
            v = 0x0f;                                   //电压>4.0V
            P2 = 0xef;                                  //P2.4 输出 0
            delay();
            if (!(CMPRES)) goto ShowVol;
            v = 0x1f;                                   //电压>4.5V
            P2 = 0xdf;                                  //P2.5 输出 0
            delay();
            if (!(CMPRES)) goto ShowVol;
            v = 0x3f;                                   //电压>5.0V
ShowVol:
            P2 = 0xff;
            P0 = ~v;
        }
}
```

22 IAP/EEPROM

Ai8051U 系列单片机内部集成了大容量的 EEPROM。利用 ISP/IAP 技术可将内部 Data Flash 当 EEPROM，擦写次数在 10 万次以上。EEPROM 可分为若干个扇区，每个扇区包含 512 字节。

注意：EEPROM 的写操作只能将字节中的 1 写为 0，当需要将字节中的 0 写为 1，则必须执行扇区擦除操作。EEPROM 的读/写操作是以 1 字节为单位进行，而 EEPROM 擦除操作是以 1 扇区（512 字节）为单位进行，在执行擦除操作时，如果目标扇区中有需要保留的数据，则必须预先将这些数据读取到 RAM 中暂存，待擦除完成后再将保存的数据和需要更新的数据一起再写回 EEPROM/DATA-FLASH。

所以在使用 EEPROM 时，建议同一次修改的数据放在同一个扇区，不是同一次修改的数据放在不同的扇区，不一定要用满。数据存储器的擦除操作是按扇区进行的（每扇区 512 字节）。

EEPROM 可用于保存一些需要在应用过程中修改并且掉电不丢失的参数数据。在用户程序中，可以对 EEPROM 进行字节读/字节编程/扇区擦除操作。在工作电压偏低时，建议不要进行 EEPROM 操作，以免发送数据丢失的情况。

22.1 EEPROM 操作时间

- 读取 1 字节：4 个系统时钟（使用 MOVC 指令读取更方便快捷）
- 编程 1 字节：约 30~40us（实际的编程时间为 6~7.5us，但还需要加上状态转换时间和各种控制信号的 SETUP 和 HOLD 时间）
- 擦除 1 扇区（512 字节）：约 4~6ms

EEPROM 操作所需时间是硬件自动控制的，用户只需要正确设置 IAP_TPS 寄存器即可。

IAP_TPS＝系统工作频率/1000000（小数部分四舍五入进行取整）

例如：系统工作频率为 12MHz，则 IAP_TPS 设置为 12
系统工作频率为 22.1184MHz，则 IAP_TPS 设置为 22
系统工作频率为 5.5296MHz，则 IAP_TPS 设置为 6

22.2 关于 EEPROM 操作时是否需要关闭中断的问题

- 如果只有主循环的代码中或者只有中断代码中才使用 IAP 方式对 EEPROM 进行操作，则 EEPROM 操作时不需要关闭中断。
- 如果主循环代码和中断代码中都有使用 IAP 方式对 EEPROM 进行操作，则 EEPROM 操作时必须关闭中断。因为如果在主循环代码中设置完成 IAP 相关寄存器后在 IAP 触发前或 IAP 触发一半时，产生了中断，而在中断中进行 EEPROM 操作时，又对 IAP 相关寄存器进行重新设置，中断返回后 IAP 相关寄存器已经发生了改变，回到主循环中继续之前的 IAP 操作必然会出错。

建议程序代码按照如下方式编写：

```
…                                ;设置 ISP/IAP 的其他相关寄存器
CLR     EA                       ;触发 ISP/IAP 前，先关闭总中断
MOV     IAP_TRIG, #5AH           ;先送 5Ah，到 ISP/IAP 触发寄存器
MOV     IAP_TRIG, #0A5H          ;再送 A5h，到 ISP/IAP 触发寄存器，每次都需如此
                                 ;送完 A5h 后，ISP/IAP 命令立即被触发启动
                                 ;IAP 命令执行完成前，CPU 不会向下继续执行程序
SETB    EA                       ;IAP 完成后再开总中断，防止【IAP 触发命令序列】被任意中断打断
                                 ;而这个任意中断服务程序中又有【IAP 触发命令序列】
                                 ;确保操作的【原子性】
```

22.3 EEPROM 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
IAP_DATA	IAP 数据寄存器	C2H	DATA[7:0]								0000,0000
IAP_ADDRE	IAP 扩展地址寄存器	F6H	ADDR[23:16]								1111,1111
IAP_ADDRH	IAP 高地址寄存器	C3H	ADDR[15:8]								0000,0000
IAP_ADDRL	IAP 低地址寄存器	C4H	ADDR[7:0]								0000,0000
IAP_CMD	IAP 命令寄存器	C5H	-	-	-	-	-	CMD[2:0]			xxxx,x000
IAP_TRIG	IAP 触发寄存器	C6H									0000,0000
IAP_CONTR	IAP 控制寄存器	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	SWBS2	-	-	-	0000,0xxx
IAP_TPS	IAP 等待时间控制寄存器	F5H	-	-	IAP_TPS[5:0]						xx00,0000

22.3.1 EEPROM 数据寄存器（IAP_DATA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_DATA	C2H	DATA[7:0]							

在进行 EEPROM 的读操作时，命令执行完成后读出的 EEPROM 数据保存在 IAP_DATA 寄存器中。在进行 EEPROM 的写操作时，在执行写命令前，必须将待写入的数据存放在 IAP_DATA 寄存器中，再发送写命令。擦除 EEPROM 命令与 IAP_DATA 寄存器无关。

22.3.2 EEPROM 地址寄存器（IAP_ADDR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_ADDRE	F6H	ADDR[23:16]							
IAP_ADDRH	C3H	ADDR[15:8]							
IAP_ADDRL	C4H	ADDR[7:0]							

EEPROM 进行读、写、擦除操作的目标地址寄存器。IAP_ADDRH 保存地址的高字节，IAP_ADDRL 保存地址的低字节

22.3.3 EEPROM 命令寄存器（IAP_CMD）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CMD	C5H	-	-	-	-	-	CMD[2:0]		

CMD[2:0]：发送EEPROM操作命令

000（CMD0）：空操作

001（CMD1）：读 EEPROM 字节命令。读取目标地址 ADDR[23:0]所在的 1 字节。

010（CMD2）：写 EEPROM 字节命令。写目标地址 ADDR[23:0]所在的 1 字节。

011（CMD3）：擦除 EEPROM 扇区。擦除目标地址 ADDR[23:9]所在的 1 扇区（512 字节）。

22.3.4 EEPROM 触发寄存器（IAP_TRIG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_TRIG	C6H								

设置完成 EEPROM 读、写、擦除的命令寄存器、地址寄存器、数据寄存器以及控制寄存器后，需要向触发寄存器 IAP_TRIG 依次写入 5AH、A5H（顺序不能交换）两个触发命令来触发相应的读、写、擦除操作。操作完成后，EEPROM 地址寄存器 IAP_ADDRE、IAP_ADDRH、IAP_ADDRL 和 EEPROM 命令寄存器 IAP_CMD 的内容不变。如果接下来要对下一个地址的数据进行操作，需手动更新地址寄存器 IAP_ADDRH 和寄存器 IAP_ADDRL 的值。

注意：每次 EEPROM 操作时，都要对 IAP_TRIG 先写入 5AH，再写入 A5H，相应的命令才会生效。写完触发命令后，CPU 会处于 IDLE 等待状态，直到相应的 IAP 操作执行完成后 CPU 才会从 IDLE 状态返回正常状态继续执行 CPU 指令。

如果主循环代码和中断代码中都有使用 IAP 方式对 EEPROM 进行操作，则 EEPROM 操作时必须关闭中断。因为如果在主循环代码中设置完成 IAP 相关寄存器后在 IAP 触发前或 IAP 触发一半时，产生了中断，而在中断中进行 EEPROM 操作时，又对 IAP 相关寄存器进行重新设置，中断返回后 IAP 相关寄存器已经发生了改变，回到主循环中继续之前的 IAP 操作必然会出错。

建议程序代码按照如下方式编写：

```
…                       ;设置 ISP/IAP 的其他相关寄存器
CLR     EA              ;触发 ISP/IAP 前，先关闭总中断
MOV     IAP_TRIG, #5AH  ;先送 5Ah，到 ISP/IAP 触发寄存器
MOV     IAP_TRIG, #0A5H ;再送 A5h，到 ISP/IAP 触发寄存器，每次都需如此
                        ;送完 A5h 后，ISP/IAP 命令立即被触发启动
                        ;IAP 命令执行完成前，CPU 不会向下继续执行程序
SETB    EA              ;IAP 完成后再开总中断，防止【IAP 触发命令序列】被任意中断打断
                        ;而这个任意中断服务程序中又有【IAP 触发命令序列】
                        ;确保操作的【原子性】
```

22.3.5 EEPROM 控制寄存器（IAP_CONTR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CONTR	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	SWBS2	-	-	-

IAPEN：EEPROM操作使能控制位

0：禁止 EEPROM 操作

1：使能 EEPROM 操作

CMD_FAIL：EEPROM操作失败状态位，需要软件清零

0：EEPROM 操作正确

1：EEPROM 操作失败

SWBS/SWBS2：软件复位启动选择

SWBS	SWBS2	复位	说明
0	0	软件复位后从用户程序区开始执行代码	用户数据区的数据保持不变
0	1	软件复位后从用户系统区开始执行代码	用户数据区的数据保持不变
1	x	软件复位后从系统 ISP 区开始执行代码	用户数据区的数据会被初始化

SWRST：软件复位触发位

0：对单片机无影响

1：触发软件复位

范例程序：

源程序区 （从此程序区）	目标程序区 （复位到此程序区）	代码
用户程序区	系统 ISP 区	IAP_CONTR = 0x60;
用户系统区		
用户程序区	用户系统区	IAP_CONTR = 0x28;
系统 ISP 区		
用户系统区	用户程序区	IAP_CONTR = 0x20;
系统 ISP 区		

22.3.6 EEPROM 擦除等待时间控制寄存器（IAP_TPS）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_TPS	F5H	-	-	IAP_TPS[5:0]					

需要根据工作频率进行设置

若工作频率为12MHz，则需要将IAP_TPS设置为12；若工作频率为24MHz，则需要将IAP_TPS设置为24，其他频率以此类推。

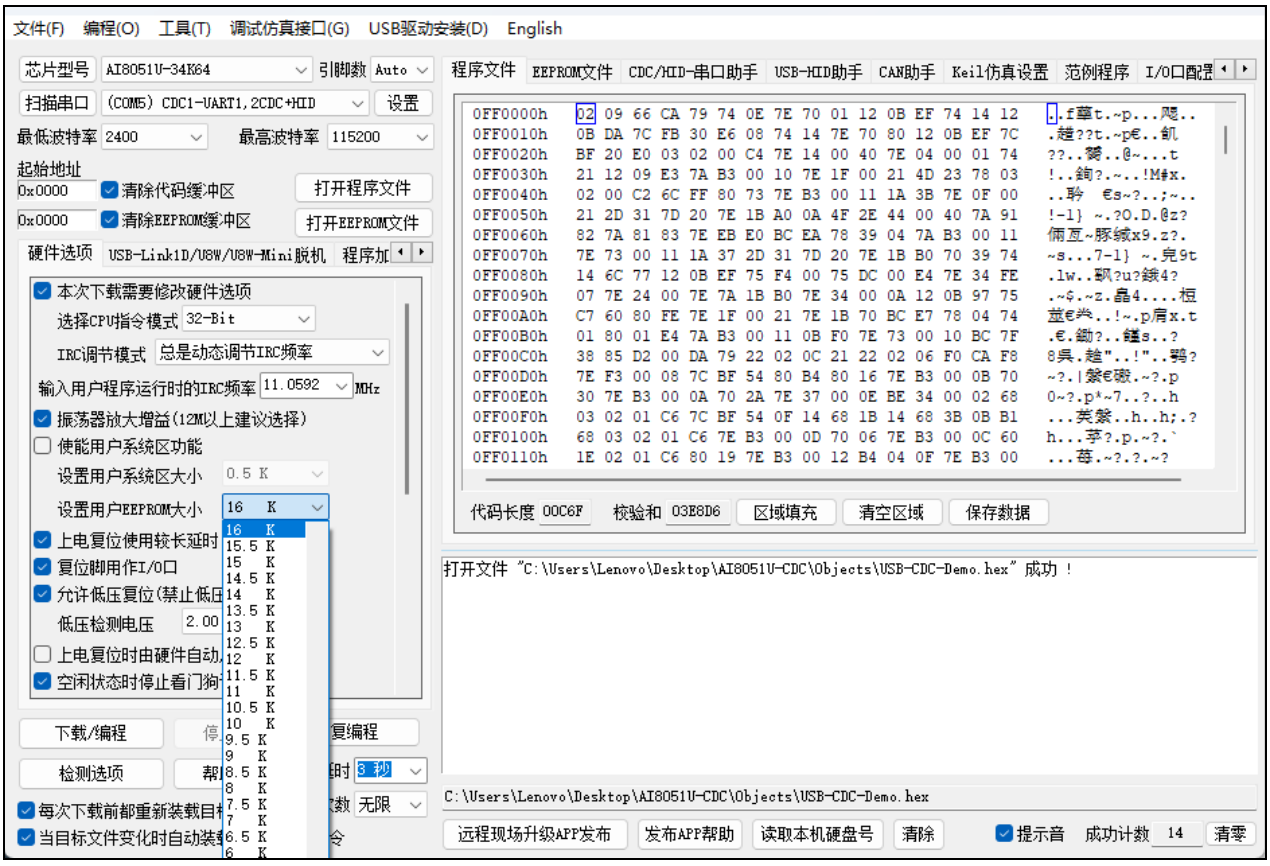
22.4 EEPROM 大小及地址

Ai8051U 系列单片机内部均有用于保存用户数据的 EEPROM。内部的 EEPROM 有 3 操作方式：读、写和擦除，其中擦除操作是以扇区为单位进行操作，每扇区为 512 字节，即每执行一次擦除命令就会擦除一个扇区，而读数据和写数据都是以字节为单位进行操作的，即每执行一次读或者写命令时只能读出或者写入一个字节。

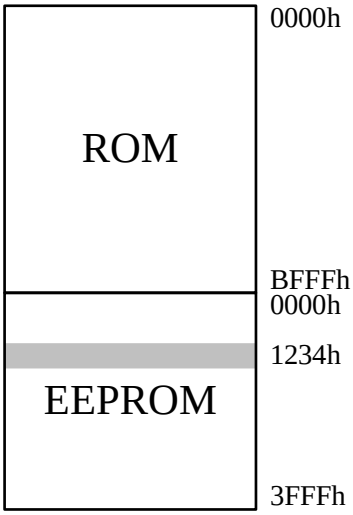
Ai8051U 系列单片机内部的 EEPROM 的访问方式有两种：IAP 方式和 MOV 方式。IAP 方式可对 EEPROM 执行读、写、擦除操作，但 MOV 只能对 EEPROM 进行读操作，而不能进行写和擦除操作。无论是使用 IAP 方式还是使用 MOV 方式访问 EEPROM，首先都需要设置正确的目标地址。

22.4.1 Ai8051U 系列 EEPROM 操作

Ai8051U 的 EEPROM 大小是需要在 ISP 下载时进行设置的，如下图所示，将 EEPROM 设置为 16K



EEPROM 在 64K 的 Flash 存储空间中位于 FF:C000h~FF:FFFFh（注意：无论 EEPROM 设置为多少，Ai8051U 系列的 EEPROM 在 Flash 存储空间中结束地址始终为 FF:FFFFh，即 **EEPROM 总是从后向前进行规划的**），如下图所示：



对于 Ai8051U 系列，使用 IAP 方式时，地址数据为 EEPROM 的目标地址，地址从 0000 开始，若使用 MOV 指令读取 EEPROM 数据，则地址数据为：**基地址 FF:0000h+程序大小的偏移+EEPROM 的目标地址**。

例如：现在需要对 EEPROM 地址 1234h 的单元进行读、写、擦除时，若使用 IAP 方式进行访问时，设置的目标地址为 1234h，即 IAP_ADDRE 设置为 00h，IAP_ADDRH 设置 12h，IAP_ADDRL 设置 34h，然后设置相应的触发命令即可对 1234h 单元进行正确操作了。但若是使用 MOV 方式读取 EEPROM 的 1234h 单元，则目标地址为**基地址 FF:0000h，**加上程序大小 0C000h，再加上 1234h，即必须将 32 位寄存器 DRx 设置为 FF:D234h，才能使用 MOV 指令正确读取。

下表列出了设置不同 EEPROM 大小时，IAP 方式和 MOV 方式访问 EEPROM 的情况

型号	EEPROM 大小	扇区数	IAP方式读/写/擦除		MOV读取	
			起始地址	结束地址	起始地址	结束地址
Ai8051U	1K	2	0:0000h	0:03FFh	FF:FC00h	FF:FFFFh
	2K	4	0:0000h	0:07FFh	FF:F800h	FF:FFFFh
	4K	8	0:0000h	0:0FFFh	FF:F000h	FF:FFFFh
	8K	16	0:0000h	0:1FFFh	FF:E000h	FF:FFFFh
	16K	32	0:0000h	0:3FFFh	FF:C000h	FF:FFFFh
	32K	64	0:0000h	0:7FFFh	FF:8000h	FF:FFFFh

用户可用根据自己的需要在整个 FLASH 空间中规划出任意不超过 FLASH 大小的 EEPROM 空间，但需要注意：**EEPROM 总是从后向前进行规划的（与 AI32GH 系列类似）**。

22.5　范例程序

22.5.1　EEPROM 基本操作

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件
#include "intrins.h"                          //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void IapIdle()
{
    IAP_CONTR = 0;                            //关闭 IAP 功能
    IAP_CMD = 0;                              //清除命令寄存器
    IAP_TRIG = 0;                             //清除触发寄存器
    IAP_ADDRE = 0x00;
    IAP_ADDRH = 0x00;
    IAP_ADDRL = 0x00;
}

char IapRead(unsigned long addr)
{
    char dat;
    bit flag;

    flag = EA;                                //保存 EA 状态
    EA = 0;                                   //关闭中断
    IAP_CONTR = 0x80;                         //使能 IAP
    IAP_TPS = 11;                             //设置等待参数 11MHz
    IAP_CMD = 1;                              //设置 IAP 读命令
    IAP_ADDRL = addr;                         //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;                    //设置 IAP 高地址
    IAP_ADDRE = addr >> 16;                   //设置 IAP 最高地址
    IAP_TRIG = 0x5a;                          //写触发命令(0x5a)
    IAP_TRIG = 0xa5;                          //写触发命令(0xa5)
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    dat = IAP_DATA;                           //读 IAP 数据
    IapIdle();                                //关闭 IAP 功能
    EA = flag;                                //恢复 EA 状态

    return dat;
}

void IapProgram(unsigned long addr, char dat)
{
    bit flag;

    flag = EA;                                //保存 EA 状态
    EA = 0;                                   //关闭中断
    IAP_CONTR = 0x80;                         //使能 IAP
    IAP_TPS = 11;                             //设置等待参数 11MHz
    IAP_CMD = 2;                              //设置 IAP 写命令
    IAP_ADDRL = addr;                         //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;                    //设置 IAP 高地址
```

```
    IAP_ADDRE = addr >> 16;                 //设置 IAP 最高地址
    IAP_DATA = dat;                         //写 IAP 数据
    IAP_TRIG = 0x5a;                        //写触发命令(0x5a)
    IAP_TRIG = 0xa5;                        //写触发命令(0xa5)
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    IapIdle();                              //关闭 IAP 功能
    EA = flag;                              //恢复 EA 状态
}

void IapErase(unsigned long addr)
{
    bit flag;

    flag = EA;                              //保存 EA 状态
    EA = 0;                                 //关闭中断
    IAP_CONTR = 0x80;                       //使能 IAP
    IAP_TPS = 11;                           //设置等待参数 11MHz
    IAP_CMD = 3;                            //设置 IAP 擦除命令
    IAP_ADDRL = addr;                       //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;                  //设置 IAP 高地址
    IAP_ADDRE = addr >> 16;                 //设置 IAP 最高地址
    IAP_TRIG = 0x5a;                        //写触发命令(0x5a)
    IAP_TRIG = 0xa5;                        //写触发命令(0xa5)
    _nop_();
    _nop_();
    _nop_();
    _nop_();                                //
    IapIdle();                              //关闭 IAP 功能
    EA = flag;                              //恢复 EA 状态
}

void main()
{
    EAXFR = 1;                              //允许访问扩展的特殊寄存器，XFR
                                            //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                          //允许访问扩展的特殊寄存器，XFR
                                            //(8 位模式请使用这句，注释上一句)
    WTST = 0;                               //设置取程序代码等待时间，
                                            //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                              //设置访问片内的 xdata 速度，
                                            //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IapErase(0x0400);
```

```
    P0 = IapRead(0x0400);                       //P0=0xff
    IapProgram(0x0400, 0x12);
    P1 = IapRead(0x0400);                       //P1=0x12

    while (1);
}
```

22.5.2　使用 MOV 读取 EEPROM（Ai8051U 系列）

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                            //头文件见下载软件
#include "intrins.h"                            //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   IAP_BASE          0xff0000
#define   ROM_OFFSET        0xc000              //假设设置 16K 的 EEPROM，则程序区为（64K-16K），
                                                //即为 48K（0xc000）

void IapIdle()
{
    IAP_CONTR = 0;                              //关闭 IAP 功能
    IAP_CMD = 0;                                //清除命令寄存器
    IAP_TRIG = 0;                               //清除触发寄存器
    IAP_ADDRE = 0x00;
    IAP_ADDRH = 0x00;
    IAP_ADDRL = 0x00;
}

unsigned char IapRead(unsigned long addr)
{
    Addr = (addr & 0xffff) + IAP_BASE + ROM_OFFSET;  //使用 MOV 读取 EEPROM 需要加上基地址和程序偏移
    return *( unsigned char far *)(addr);       //使用 MOV 读取数据
}

void IapProgram(unsigned long addr, unsigned char dat)
{
    bit flag;

    flag = EA;                                  //保存 EA 状态
    EA = 0;                                     //关闭中断
    IAP_CONTR = 0x80;                           //使能 IAP
    IAP_TPS = 11;                               //设置等待参数 11MHz
    IAP_CMD = 2;                                //设置 IAP 写命令
    IAP_ADDRL = addr;                           //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;                      //设置 IAP 高地址
    IAP_ADDRE = addr >> 16;                     //设置 IAP 最高地址
    IAP_DATA = dat;                             //写 IAP 数据
    IAP_TRIG = 0x5a;                            //写触发命令(0x5a)
    IAP_TRIG = 0xa5;                            //写触发命令(0xa5)
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    IapIdle();                                  //关闭 IAP 功能
    EA = flag;                                  //恢复 EA 状态
```

```
}

void IapErase(unsigned long addr)
{
    bit flag;

    flag = EA;                          //保存 EA 状态
    EA = 0;                             //关闭中断
    IAP_CONTR = 0x80;                   //使能 IAP
    IAP_TPS = 11;                       //设置等待参数 11MHz
    IAP_CMD = 3;                        //设置 IAP 擦除命令
    IAP_ADDRL = addr;                   //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;              //设置 IAP 高地址
    IAP_ADDRE = addr >> 16;             //设置 IAP 最高地址
    IAP_TRIG = 0x5a;                    //写触发命令(0x5a)
    IAP_TRIG = 0xa5;                    //写触发命令(0xa5)
    _nop_();
    _nop_();
    _nop_();
    _nop_();                            //
    IapIdle();                          //关闭 IAP 功能
    EA = flag;                          //恢复 EA 状态
}

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                           //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IapErase(0x0400);
    P0 = IapRead(0x0400);               //P0=0xff
    IapProgram(0x0400, 0x12);
    P1 = IapRead(0x0400);               //P1=0x12

    while (1);
}
```

22.5.3　使用串口送出 EEPROM 数据

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   FOSC          11059200UL
#define   BRT           (65536 - (FOSC / 115200+2) / 4)
                                                        //加 2 操作是为了让 Keil 编译器
                                                        //自动实现四舍五入运算

void UartInit()
{
    SCON = 0x5a;
    T2L = BRT;
    T2H = BRT >> 8;
    S1BRT = 1;
    T2x12 = 1;
    T2R = 1;
}

void UartSend(char dat)
{
    while (!TI);
    TI = 0;
    SBUF = dat;
}

void IapIdle()
{
    IAP_CONTR = 0;                                      //关闭 IAP 功能
    IAP_CMD = 0;                                        //清除命令寄存器
    IAP_TRIG = 0;                                       //清除触发寄存器
    IAP_ADDRE = 0x00;
    IAP_ADDRH = 0x00;
    IAP_ADDRL = 0x00;
}

char IapRead(unsigned long addr)
{
    char dat;
    bit flag;

    flag = EA;                                          //保存 EA 状态
    EA = 0;                                             //关闭中断

    IAP_CONTR = 0x80;                                   //使能 IAP
    IAP_TPS = 11;                                       //设置等待参数 11MHz
    IAP_CMD = 1;                                        //设置 IAP 读命令
    IAP_ADDRL = addr;                                   //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;                              //设置 IAP 高地址
    IAP_ADDRE = addr >> 16;                             //设置 IAP 最高地址
    IAP_TRIG = 0x5a;                                    //写触发命令(0x5a)
    IAP_TRIG = 0xa5;                                    //写触发命令(0xa5)
    _nop_();
```

```
    _nop_();
    _nop_();
    _nop_();
    dat = IAP_DATA;                 //读 IAP 数据
    IapIdle();                      //关闭 IAP 功能
    EA = flag;                      //恢复 EA 状态

    return dat;
}

void IapProgram(unsigned long addr, char dat)
{
    bit flag;

    flag = EA;                      //保存 EA 状态
    EA = 0;                         //关闭中断
    IAP_CONTR = 0x80;               //使能 IAP
    IAP_TPS = 11;                   //设置等待参数 11MHz
    IAP_CMD = 2;                    //设置 IAP 写命令
    IAP_ADDRL = addr;               //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;          //设置 IAP 高地址
    IAP_ADDRE = addr >> 16;         //设置 IAP 最高地址
    IAP_DATA = dat;                 //写 IAP 数据
    IAP_TRIG = 0x5a;                //写触发命令(0x5a)
    IAP_TRIG = 0xa5;                //写触发命令(0xa5)
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    IapIdle();                      //关闭 IAP 功能
    EA = flag;                      //恢复 EA 状态
}

void IapErase(unsigned long addr)
{
    bit flag;

    flag = EA;                      //保存 EA 状态
    EA = 0;                         //关闭中断
    IAP_CONTR = 0x80;               //使能 IAP
    IAP_TPS = 11;                   //设置等待参数 11MHz
    IAP_CMD = 3;                    //设置 IAP 擦除命令
    IAP_ADDRL = addr;               //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;          //设置 IAP 高地址
    IAP_ADDRE = addr >> 16;         //设置 IAP 最高地址
    IAP_TRIG = 0x5a;                //写触发命令(0x5a)
    IAP_TRIG = 0xa5;                //写触发命令(0xa5)
    _nop_();
    _nop_();
    _nop_();
    _nop_();                        //
    IapIdle();                      //关闭 IAP 功能
    EA = flag;                      //恢复 EA 状态
}

void main()
{
    EAXFR = 1;                      //允许访问扩展的特殊寄存器，XFR
                                    //(32 位模式请使用这句，注释下一句)
```

```
//      P_SW2 |= 0x80;                          //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
        WTST = 0;                               //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
        CKCON = 0;                              //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

        UartInit();
        IapErase(0x0400);
        UartSend(IapRead(0x0400));
        IapProgram(0x0400, 0x12);
        UartSend(IapRead(0x0400));

        while (1);
}
```

22.5.4　串口 1 读写 EEPROM-带 MOV 读

（main.c）

```
//测试工作频率为 11.0592MHz

/*       本程序经过测试完全正常，不提供电话技术支持，如不能理解，请自行补充相关基础。  */

/************ 本程序功能说明       **************
Ai8051U 系列 EEPROM 通用测试程序.
请先别修改程序，直接下载测试.，下载时选择主频 11.0592MHZ.
PC 串口设置：波特率 115200,8,n,1.
对 EEPROM 做扇区擦除、写入 64 字节、读出 64 字节的操作。

命令例子:
E 0       对 EEPROM 进行扇区擦除操作，E 表示擦除，数字 0 为 0 扇区(十进制, 0~126，看具体 IC).
W 0       对 EEPROM 进行写入操作，W 表示写入，数字 0 为 0 扇区(十进制, 0~126，看具体 IC). 从扇区的开始地址连续写 64 字节.
R 0       对 EEPROM 进行 IAP 读出操作 R 表示读出，数字 0 为 0 扇区(十进制, 0~126，看具体 IC). 从扇区的开始地址连续读 64 字节.
M 0       对 EEPROM 进行 MOVC 读出操作(操作地址为扇区*512+偏移地址) 数字 0 为 0 扇区(十进制, 0~126，看具体 IC). 从扇区的开始地址连续读 64 字节.


注意：为了通用，程序不识别扇区是否有效，用户自己根据具体的型号来决定。

*****************************************/
```

```
#include    "config.H"
#include    "EEPROM.h"

#define     Baudrate1                   115200L
#define     UART1_BUF_LENGTH            10
#define     EEADDR_OFFSET               0xff0000            //定义 EEPROM 用 MOV 访问时加的基地址,
#define     TimeOutSet1                 5

/************* 本地常量声明    **************/
u8    code  T_Strings[]={"去年今日此门中，人面桃花相映红。人面不知何处去，桃花依旧笑春风。"};

/************* 本地变量声明    **************/
u8    xdata tmp[70];
u8    xdata RX1_Buffer[UART1_BUF_LENGTH];
u8    RX1_Cnt;
u8    RX1_TimeOut;
bit   B_TX1_Busy;

/************* 本地函数声明    **************/
void  UART1_config(void);
void  TX1_write2buff(u8 dat);                               //写入发送缓冲
void  PrintString1(u8 *puts);                               //发送一个字符串

/*************    外部函数和变量声明 *****************/

/*****************************************************/

u8    CheckData(u8 dat)
{
    if((dat >= '0') && (dat <= '9'))    return (dat-'0');
    if((dat >= 'A') && (dat <= 'F'))    return (dat-'A'+10);
    if((dat >= 'a') && (dat <= 'f'))    return (dat-'a'+10);
    return 0xff;
}

u16   GetAddress(void)
{
    u16   address;
    u8    i;

    address = 0;
    if(RX1_Cnt <   3)       return 65535;                   //error
    if(RX1_Cnt <= 5)                                        //5 个字节以内是扇区操作，十进制,
                                                            //支持命令:      E 0, E 12, E 120
                                                            //                W 0, W 12, W 120
                                                            //                R 0, R 12, R 120
    {
        for(i=2; i<RX1_Cnt; i++)
        {
            if(CheckData(RX1_Buffer[i]) > 9)
                return 65535;                               //error
            address = address * 10 + CheckData(RX1_Buffer[i]);
        }
        if(address < 124)                                   //限制在 0~123 扇区
        {
            address <<= 9;
            return (address);
        }
    }
```

```
    else if(RX1_Cnt == 8)                                   //8 个字节直接地址操作，十六进制,
                                                            //支持命令: E 0x1234, W 0x12b3, R 0x0A00
    {
        if((RX1_Buffer[2] == '0') && ((RX1_Buffer[3] == 'x') || (RX1_Buffer[3] == 'X')))
        {
            for(i=4; i<8; i++)
            {
                if(CheckData(RX1_Buffer[i]) > 0x0F)
                    return 65535;                           //error
                address = (address << 4) + CheckData(RX1_Buffer[i]);
            }
            if(address < 63488)
                return (address);                           //限制在 0~123 扇区
        }
    }

     return    65535;                                       //error
}

//========================================================================
// 函数: void   delay_ms(u8 ms)
// 描述:  延时函数。
// 参数: ms,要延时的ms 数,  这里只支持 1~255ms.  自动适应主时钟.
// 返回: none.
// 版本: VER1.0
// 备注:
//========================================================================
void   delay_ms(u8 ms)
{
    u16 i;
    do
    {
        i = MAIN_Fosc / 10000;
        while(--i)   ;
    }while(--ms);
}

//使用 MOV 读 EEPROM
void EEPROM_MOV_read_n(u16 EE_address, u8 *DataAddress, u16 number)
{
    u8    far *pc;

    pc = EE_address + EEADDR_OFFSET;
    do
    {
        *DataAddress = *pc;                                 //读出的数据
        DataAddress++;
        pc++;
    }while(--number);
}

/**********************  主函数  *************************/
void main(void)
{
    u8    i;
    u16 addr;

    UART1_config();                                         //  选择波特率, 2:  使用 Timer2 做波特率,
                                                            //其它值:  使用 Timer1 做波特率.
```

```
    EA = 1;                                             //允许总中断

    PrintString1("AI MCU 用串口 1 测试 EEPROM 程序!\r\n"); //UART1 发送一个字符串

    while(1)
    {
        delay_ms(1);
        if(RX1_TimeOut > 0)                             //超时计数
        {
            if(--RX1_TimeOut == 0)
            {
                if(RX1_Buffer[1] == ' ')
                {
                    addr = GetAddress();
                    if(addr < 63488)                    //限制在 0~123 扇区
                    {
                        if(RX1_Buffer[0] == 'E')        //PC 请求擦除一个扇区
                        {
                            EEPROM_SectorErase(addr);
                            PrintString1("扇区擦除完成!\r\n");
                        }

                        else if(RX1_Buffer[0] == 'W')   //PC 请求写入 EEPROM 64 字节数据
                        {
                            EEPROM_write_n(addr,T_Strings,64);
                            PrintString1("写入操作完成!\r\n");
                        }

                        else if(RX1_Buffer[0] == 'R')   //PC 请求返回 64 字节 EEPROM 数据
                        {
                            PrintString1("IAP 读出的数据如下：\r\n");
                            EEPROM_read_n(addr,tmp,64);
                            for(i=0; i<64; i++)
                                TX1_write2buff(tmp[i]);     //将数据返回给串口
                            TX1_write2buff(0x0d);
                            TX1_write2buff(0x0a);
                        }
                        else if(RX1_Buffer[0] == 'M')   //PC 请求返回 64 字节 EEPROM 数据
                        {
                            PrintString1("MOVC 读出的数据如下：\r\n");
                            EEPROM_MOVC_read_n(addr,tmp,64);
                            for(i=0; i<64; i++)
                                TX1_write2buff(tmp[i]);     //将数据返回给串口
                            TX1_write2buff(0x0d);
                            TX1_write2buff(0x0a);
                        }
                        else   PrintString1("命令错误!\r\n");
                    }
                    else   PrintString1("命令错误!\r\n");
                }

                RX1_Cnt = 0;
            }
        }
    }
}
/*------------------------------------------------*/

/*************  发送一个字节 ****************************/
```

```
void TX1_write2buff(u8 dat)                     //写入发送缓冲
{
    B_TX1_Busy = 1;                             //标志发送忙
    SBUF = dat;                                 //发送一个字节
    while(B_TX1_Busy);                          //等待发送完毕
}

//========================================================================
// 函数: void PrintString1(u8 *puts)
// 描述: 串口 1 发送字符串函数。
// 参数: puts:   字符串指针.
// 返回: none.
// 版本: VER1.0
// 备注:
//========================================================================
void PrintString1(u8 *puts)                     //发送一个字符串
{
    for (; *puts != 0;  puts++)                 //遇到停止符 0 结束
    {
        TX1_write2buff(*puts);
    }
}

//========================================================================
// 函数: void    UART1_config(void)
// 描述: UART1 初始化函数。
// 参数: none.
// 返回: none.
// 版本: VER1.0
// 备注:
//========================================================================
void  UART1_config(void)
{
    TR1 = 0;
    AUXR &= ~0x01;                              //S1 BRT Use Timer1;
    AUXR |=  (1<<6);                            //Timer1 set as 1T mode
    TMOD &= ~(1<<6);                            //Timer1 set As Timer
    TMOD &= ~0x30;                              //Timer1_16bitAutoReload;
    TH1 = (u8)((65536L-(MAIN_Fosc / 4) / Baudrate1) >> 8);
    TL1 = (u8)(65536L-(MAIN_Fosc / 4) / Baudrate1);
    ET1 = 0;                                    // 禁止 Timer1 中断
    INT_CLKO &= ~0x02;                          // Timer1 不输出高速时钟
    TR1  = 1;                                   // 运行 Timer1

    S1_USE_P30P31();    P3n_standard(0x03);     //切换到 P3.0 P3.1
    //S1_USE_P36P37();  P3n_standard(0xc0);     //切换到 P3.6 P3.7
    //S1_USE_P16P17();  P1n_standard(0xc0);     //切换到 P1.6 P1.7

    SCON = (SCON & 0x3f) | 0x40;                //UART1 模式, 0x00: 同步移位输出,
                                                //            0x40: 8 位数据,可变波特率,
                                                //            0x80: 9 位数据,固定波特率,
                                                //            0xc0: 9 位数据,可变波特率
//  PS  = 1;                                    //高优先级中断
    ES  = 1;                                    //允许中断
    REN = 1;                                    //允许接收

    B_TX1_Busy = 0;
    RX1_Cnt = 0;
}
```

```
//========================================================================
// 函数: void UART1_int (void) interrupt UART1_VECTOR
// 描述: UART1 中断函数。
// 参数: nine.
// 返回: none.
// 版本: VER1.0
// 备注:
//========================================================================
void UART1_int (void) interrupt 4
{
    if(RI)
    {
        RI = 0;
        if(RX1_Cnt >= UART1_BUF_LENGTH)
            RX1_Cnt = 0;                                    //防溢出
        RX1_Buffer[RX1_Cnt++] = SBUF;
        RX1_TimeOut = TimeOutSet1;
    }

    if(TI)
    {
        TI = 0;
        B_TX1_Busy = 0;
    }
}
```

（EEPROM.c）

```
//测试工作频率为11.0592MHz

//    本程序是内置 EEPROM 读写程序。

#include "config.h"
#include "eeprom.h"

//========================================================================
// 函数: void    IAP_Disable(void)
// 描述:  禁止访问 ISP/IAP.
// 参数: non.
// 返回: non.
// 版本: V1.0
//========================================================================
void  DisableEEPROM(void)
{
    IAP_CONTR = 0;                                  //禁止 ISP/IAP 操作
    IAP_TPS   = 0;
    IAP_CMD   = 0;                                  //去除 ISP/IAP 命令
    IAP_TRIG  = 0;                                      //防止 ISP/IAP 命令误触发
    IAP_ADDRE = 0xff;                               //清 0 地址高字节
    IAP_ADDRH = 0xff;                               //清 0 地址高字节
    IAP_ADDRL = 0xff;                               //清 0 地址低字节，指向非 EEPROM 区，防止误操作
}

//========================================================================
// 函数: void EEPROM_read_n(u16 EE_address,u8 *DataAddress,u16 number)
// 描述:  从指定 EEPROM 首地址读出 n 个字节放指定的缓冲.
// 参数: EE_address:   读出 EEPROM 的首地址.
//          DataAddress:  读出数据放缓冲的首地址.
```

```
//          number:         读出的字节长度.
// 返回: non.
// 版本: V1.0
//=========================================================================
void EEPROM_read_n(u32 EE_address,u8 *DataAddress,u16 number)
{
    EA = 0;                                         //禁止中断
    IAP_CONTR = IAP_EN;                             //允许 ISP/IAP 操作
    IAP_TPS = (u8)(MAIN_Fosc / 1000000L);           //工作频率设置
    IAP_READ();                                     //送字节读命令，命令不需改变时，不需重新送命令
    do
    {
        IAP_ADDRE = EE_address / 65536;             //送地址高字节（地址需要改变时才需重新送地址）
        IAP_ADDRH = (EE_address / 256) % 256;       //送地址高字节（地址需要改变时才需重新送地址）
        IAP_ADDRL = EE_address % 256;               //送地址低字节
        IAP_TRIG();                                 //先送 5AH，再送 A5H 到 ISP/IAP 触发寄存器，
                                                    //每次都需要如此
                                                    //送完 A5H 后，ISP/IAP 命令立即被触发启动
                                                    //CPU 等待 IAP 完成后，才会继续执行程序。
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        *DataAddress = IAP_DATA;                        //读出的数据送往
        EE_address++;
        DataAddress++;
    }while(--number);

    DisableEEPROM();
    EA = 1;                                         //重新允许中断
}

/********************  扇区擦除函数  *****************/
//=========================================================================
// 函数: void EEPROM_SectorErase(u16 EE_address)
// 描述:  把指定地址的 EEPROM 扇区擦除.
// 参数: EE_address:    要擦除的扇区 EEPROM 的地址.
// 返回: non.
// 版本: V1.0
//=========================================================================
void EEPROM_SectorErase(u32 EE_address)
{
    EA = 0;                                         //禁止中断
                                                    //只有扇区擦除，没有字节擦除，512 字节/扇区。
                                                    //扇区中任意一个字节地址都是扇区地址。
    IAP_ADDRE = EE_address / 65536;                 //送地址高字节（地址需要改变时才需重新送地址）
    IAP_ADDRH = (EE_address / 256) % 256;           //送地址高字节（地址需要改变时才需重新送地址）
    IAP_ADDRL = EE_address % 256;                   //送地址低字节
    IAP_CONTR = IAP_EN;                             //允许 ISP/IAP 操作
    IAP_TPS = (u8)(MAIN_Fosc / 1000000L);           //工作频率设置
    IAP_ERASE();                                    //送扇区擦除命令，命令不需改变时，不需重新送命令
    IAP_TRIG();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    DisableEEPROM();
    EA = 1;                                         //重新允许中断
}
```

```
//=========================================================================
// 函数: void EEPROM_write_n(u16 EE_address,u8 *DataAddress,u16 number)
// 描述: 把缓冲的 n 个字节写入指定首地址的 EEPROM.
// 参数: EE_address:  写入 EEPROM 的首地址.
//       DataAddress: 写入源数据的缓冲的首地址.
//       number:      写入的字节长度.
// 返回: non.
// 版本: V1.0
//=========================================================================
void EEPROM_write_n(u32 EE_address,u8 *DataAddress,u16 number)
{
    EA = 0;                                     //禁止中断

    IAP_CONTR = IAP_EN;                         //允许 ISP/IAP 操作
    IAP_TPS = (u8)(MAIN_Fosc / 1000000L);       //工作频率设置
    IAP_WRITE();                                //送字节写命令，命令不需改变时，不需重新送命令
    do
    {
        IAP_ADDRE = EE_address / 65536;         //送地址高字节（地址需要改变时才需重新送地址）
        IAP_ADDRH = (EE_address / 256) % 256;   //送地址高字节（地址需要改变时才需重新送地址）
        IAP_ADDRL = EE_address % 256;           //送地址低字节
        IAP_DATA  = *DataAddress;               //送数据到 IAP_DATA，只有数据改变时才需重新送
        IAP_TRIG();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        EE_address++;
        DataAddress++;
    }while(--number);

    DisableEEPROM();
    EA = 1;                                     //重新允许中断
}
```

22.5.5　口令擦除写入-多扇区备份-串口 1 操作

（main.c）

```
//测试工作频率为 11.0592MHz

/*      本程序经过测试完全正常，不提供电话技术支持，如不能理解，请自行补充相关基础。  */

/************ 本程序功能说明       **************
Ai8051U 系列 EEPROM 通用测试程序，演示多扇区备份、有扇区错误则用正确扇区数据写入、全部扇区错误(比如第一次运行程序)则写入默认值.

每次写都写入 3 个扇区，即冗余备份.
每个扇区写一条记录，写入完成后读出保存的数据和校验值跟源数据和校验值比较，并从串口 1(P3.0  P3.1)返回结果(正确或错误提示).
每条记录自校验, 64 字节数据, 2 字节校验值，校验值 = 64 字节数据累加和 ^ 0x5555. ^0x5555 是为了保证写入的 66 个数据不全部为 0.
如果有扇区错误，则将正确扇区的数据写入错误扇区，如果 3 个扇区都错误，则均写入默认值.
擦除、写入、读出操作前均需要设置口令，如果口令不对则退出操作，并且每次退出操作都会清除口令.

下载时选择主频 11.0592MHZ.
```

```
PC 串口设置: 波特率 115200,8,n,1.
对 EEPROM 做扇区擦除、写入 64 字节、读出 64 字节的操作。

命令例子:
使用串口助手发单个字符, 大小写均可.

发 E 或 e:  对 EEPROM 进行扇区擦除操作，E 表示擦除, 会擦除扇区 0 、1 、2.

发 W 或 w:  对 EEPROM 进行写入操作，W 表示写入, 会写入扇区 0 、1 、2, 每个扇区连续写 64 字节, 扇区 0 写入 0x0000~0x003f, 扇区 1 写入 0x0200~0x023f, 扇区 0 写入 0x0400~0x043f.

发 R 或 r:  对 EEPROM 进行读出操作，R 表示读出, 会读出扇区 0 、1 、2, 每个扇区连续读 64 字节, 扇区 0 读出 0x0000~0x003f, 扇区 1 读出 0x0200~0x023f, 扇区 0 读出 0x0400~0x043f.

注意：为了通用，程序不识别扇区是否有效，用户自己根据具体的型号来决定。

*****************************************/

#include    "config.H"
#include    "EEPROM.h"

#define         Baudrate1           115200L

/************* 本地常量声明    **************/
u8    code  T_StringD[]={"去年今日此门中，人面桃花相映红。人面不知何处去，桃花依旧笑春风。"};
u8    code  T_StringW[]={"横看成岭侧成峰，远近高低各不同。不识庐山真面目，只缘身在此山中。"};


/************* 本地变量声明    **************/
u8    xdata tmp[70];                                    //通用数据
u8    xdata SaveTmp[70];                                //要写入的数组

bit   B_TX1_Busy;
u8    cmd;                                              //串口单字符命令


/************* 本地函数声明    **************/
void  UART1_config(void);
void  TX1_write2buff(u8 dat);                           //写入发送缓冲
void  PrintString1(u8 *puts);                           //发送一个字符串


/*************  外部函数和变量声明 *****************/

/*************  读取 EEPROM 记录, 并且校验, 返回校验结果, 0 为正确, 1 为错误 *****************/
u8    ReadRecord(u16 addr)
{
    u8    i;
    u16   ChckSum;                                      //计算的累加和
    u16   j;                                            //读取的累加和

    for(i=0; i<66; i++)     tmp[i] = 0;                 //清除缓冲
    PassWord = D_PASSWORD;                              //给定口令
    EEPROM_read_n(addr,tmp,66);                         //读出扇区 0
    for(ChckSum=0, i=0; i<64; i++)
        ChckSum += tmp[i];                              //计算累加和
    j = ((u16)tmp[64]<<8) + (u16)tmp[65];               //读取记录的累加和
    j ^= 0x5555;                                        //隔位取反, 避免全 0
```

```
    if(ChckSum != j)return 1;                       //累加和错误，返回1
    return      0;                                  //累加和正确，返回0
}

/*************  写入 EEPROM 记录，并且校验，返回校验结果, 0 为正确, 1 为错误  *****************/
u8  SaveRecord(u16 addr)
{
    u8  i;
    u16 ChckSum;                                    //计算的累加和

    for(ChckSum=0, i=0; i<64; i++)
        ChckSum += SaveTmp[i];                      //计算累加和
    ChckSum ^= 0x5555;                              //隔位取反，避免全0
    SaveTmp[64] = (u8)(ChckSum >> 8);
    SaveTmp[65] = (u8)ChckSum;

    PassWord = D_PASSWORD;                          //给定口令
    EEPROM_SectorErase(addr);                       //擦除一个扇区
    PassWord = D_PASSWORD;                          //给定口令
    EEPROM_write_n(addr, SaveTmp, 66);              //写入扇区

    for(i=0; i<66; i++)
        tmp[i] = 0;                                 //清除缓冲
    PassWord = D_PASSWORD;                          //给定口令
    EEPROM_read_n(addr,tmp,66);                     //读出扇区0
    for(i=0; i<66; i++)                             //数据比较
    {
        if(SaveTmp[i] != tmp[i])
            return 1;                               //数据有错误，返回1
    }
    return 0;                                       //累加和正确，返回0
}

/**********************  主函数  *************************/
void main(void)
{
    u8  i;
    u8  status;                                     //状态

    UART1_config();                                 //  选择波特率, 2:  使用 Timer2 做波特率,
                                                    //其它值:  使用 Timer1 做波特率.
    EA = 1;                                         //允许总中断

    PrintString1("AI32G-8H-8C 系列 MCU 用串口 1 测试 EEPROM 程序!\r\n");      //UART1 发送一个字符串

                                                    //上电读取 3 个扇区并校验，如果有扇区错误则将正确的
                                                    //扇区写入错误区，如果 3 个扇区都错误，则写入默认值
    status = 0;
    if(ReadRecord(0x0000) == 0)                     //读扇区0
    {
        status |= 0x01;                             //正确则标记 status.0=1
        for(i=0; i<64; i++)
            SaveTmp[i] = tmp[i];                    //保存在写缓冲
    }
    if(ReadRecord(0x0200) == 0)                     //读扇区1
    {
        status |= 0x02;                             //正确则标记 status.1=1
        for(i=0; i<64; i++)
            SaveTmp[i] = tmp[i];                    //保存在写缓冲
```

```
    }
    if(ReadRecord(0x0400) == 0)                     //读扇区 2
    {
        status |= 0x04;                             //正确则标记 status.2=1
        for(i=0; i<64; i++)
            SaveTmp[i] = tmp[i];                    //保存在写缓冲
    }

    if(status == 0)                                 //所有扇区都错误, 则写入默认值
    {
        for(i=0; i<64; i++)
            SaveTmp[i] = T_StringD[i];              //读取默认值
    }
    else PrintString1("上电读取 3 个扇区数据均正确!\r\n");  //UART1 发送一个字符串提示

    if((status & 0x01) == 0)                        //扇区 0 错误, 则写入默认值
    {
        if(SaveRecord(0x0000) == 0)
            PrintString1("写入扇区 0 正确!\r\n");    //写入记录 0 扇区正确
        else
            PrintString1("写入扇区 0 错误!\r\n");    //写入记录 0 扇区错误
    }
    if((status & 0x02) == 0)                        //扇区 1 错误, 则写入默认值
    {
        if(SaveRecord(0x0200) == 0)
            PrintString1("写入扇区 1 正确!\r\n");    //写入记录 1 扇区正确
        else
            PrintString1("写入扇区 1 错误!\r\n");    //写入记录 1 扇区错误
    }
    if((status & 0x04) == 0)                        //扇区 2 错误, 则写入默认值
    {
        if(SaveRecord(0x0400) == 0)
            PrintString1("写入扇区 2 正确!\r\n");    //写入记录 2 扇区正确
        else
            PrintString1("写入扇区 2 错误!\r\n");    //写入记录 2 扇区错误
    }


    while(1)
    {
        if(cmd != 0)                                //有串口命令
        {
            if((cmd >= 'a') && (cmd <= 'z'))
                cmd -= 0x20;                        //小写转大写

            if(cmd == 'E')                          //PC 请求擦除一个扇区
            {
                PassWord = D_PASSWORD;              //给定口令
                EEPROM_SectorErase(0x0000);         //擦除一个扇区
                PassWord = D_PASSWORD;              //给定口令
                EEPROM_SectorErase(0x0200);         //擦除一个扇区
                PassWord = D_PASSWORD;              //给定口令
                EEPROM_SectorErase(0x0400);         //擦除一个扇区
                PrintString1("扇区擦除完成!\r\n");
            }

            else if(cmd == 'W')                     //PC 请求写入 EEPROM 64 字节数据
            {
                for(i=0; i<64; i++)
```

```
                SaveTmp[i] = T_StringW[i];          //写入数值
            if(SaveRecord(0x0000) == 0)
                PrintString1("写入扇区 0 正确!\r\n");      //写入记录 0 扇区正确
            else
                PrintString1("写入扇区 0 错误!\r\n");      //写入记录 0 扇区错误
            if(SaveRecord(0x0200) == 0)
                PrintString1("写入扇区 1 正确!\r\n");      //写入记录 1 扇区正确
            else
                PrintString1("写入扇区 1 错误!\r\n");      //写入记录 1 扇区错误
            if(SaveRecord(0x0400) == 0)
                PrintString1("写入扇区 2 正确!\r\n");      //写入记录 2 扇区正确
            else
                PrintString1("写入扇区 2 错误!\r\n");      //写入记录 2 扇区错误
        }

        else if(cmd == 'R')                         //PC 请求返回 64 字节 EEPROM 数据
        {
            if(ReadRecord(0x0000) == 0)             //读出扇区 0 的数据
            {
                PrintString1("读出扇区 0 的数据如下：\r\n");
                for(i=0; i<64; i++)
                    TX1_write2buff(tmp[i]);         //将数据返回给串口
                TX1_write2buff(0x0d);               //回车换行
                TX1_write2buff(0x0a);
            }
            else   PrintString1("读出扇区 0 的数据错误!\r\n");

            if(ReadRecord(0x0200) == 0)             //读出扇区 1 的数据
            {
                PrintString1("读出扇区 1 的数据如下：\r\n");
                for(i=0; i<64; i++)
                    TX1_write2buff(tmp[i]);         //将数据返回给串口
                TX1_write2buff(0x0d);               //回车换行
                TX1_write2buff(0x0a);
            }
            else   PrintString1("读出扇区 1 的数据错误!\r\n");

            if(ReadRecord(0x0400) == 0)             //读出扇区 2 的数据
            {
                PrintString1("读出扇区 2 的数据如下：\r\n");
                for(i=0; i<64; i++)
                    TX1_write2buff(tmp[i]);         //将数据返回给串口
                TX1_write2buff(0x0d);               //回车换行
                TX1_write2buff(0x0a);
            }
            else   PrintString1("读出扇区 2 的数据错误!\r\n");
        }
        else   PrintString1("命令错误!\r\n");
        cmd = 0;
    }
  }
}
/*********************************************/

/**************  发送一个字节  *****************************/
void TX1_write2buff(u8 dat)                         //写入发送缓冲
{
    B_TX1_Busy = 1;                                 //标志发送忙
    SBUF = dat;                                     //发送一个字节
```

```
    while(B_TX1_Busy);                                  //等待发送完毕
}

//========================================================================
// 函数: void PrintString1(u8 *puts)
// 描述: 串口 1 发送字符串函数。
// 参数: puts:   字符串指针.
// 返回: none.
// 版本: VER1.0
// 备注:
//========================================================================
void PrintString1(u8 *puts)                             //发送一个字符串
{
    for (; *puts != 0;  puts++)                         //遇到停止符 0 结束
    {
        TX1_write2buff(*puts);
    }
}

//========================================================================
// 函数: void    UART1_config(void)
// 描述: UART1 初始化函数。
// 参数: none.
// 返回: none.
// 版本: VER1.0
// 备注:
//========================================================================
void  UART1_config(void)
{
    TR1 = 0;
    AUXR &= ~0x01;                                      //S1 BRT Use Timer1;
    AUXR |=  (1<<6);                                    //Timer1 set as 1T mode
    TMOD &= ~(1<<6);                                    //Timer1 set As Timer
    TMOD &= ~0x30;                                      //Timer1_16bitAutoReload;
    TH1 = (u8)((65536L-(MAIN_Fosc / 4) / Baudrate1) >> 8);
    TL1 = (u8)(65536L-(MAIN_Fosc / 4) / Baudrate1);
    ET1 = 0;                                            // 禁止 Timer1 中断
    INT_CLKO &= ~0x02;                                  // Timer1 不输出高速时钟
    TR1  = 1;                                           // 运行 Timer1

    S1_USE_P30P31();    P3n_standard(0x03);             //切换到 P3.0 P3.1
    //S1_USE_P36P37();  P3n_standard(0xc0);             //切换到 P3.6 P3.7
    //S1_USE_P16P17();  P1n_standard(0xc0);             //切换到 P1.6 P1.7

    SCON = (SCON & 0x3f) | 0x40;                        //UART1 模式,   0x00: 同步移位输出,
                                                        //              0x40: 8 位数据,可变波特率,
                                                        //              0x80: 9 位数据,固定波特率,
                                                        //              0xc0: 9 位数据,可变波特率
//  PS  = 1;                                            //高优先级中断
    ES  = 1;                                            //允许中断
    REN = 1;                                            //允许接收

    B_TX1_Busy = 0;
}

//========================================================================
// 函数: void UART1_int (void) interrupt UART1_VECTOR
// 描述: UART1 中断函数。
// 参数: nine.
```

```
// 返回: none.
// 版本: VER1.0
// 备注:
//==========================================================================
void UART1_int (void) interrupt 4
{
    if(RI)
    {
        RI = 0;
        cmd = SBUF;
    }

    if(TI)
    {
        TI = 0;
        B_TX1_Busy = 0;
    }
}
```

（EEPROM.c）

```
//测试工作频率为 11.0592MHz

//    本程序是内置 EEPROM 读写程序。

#include    "config.h"
#include    "EEPROM.h"

u32         PassWord;                                   //擦除 写入时需要的口令

//==========================================================================
// 函数: void    IAP_Disable(void)
// 描述: 禁止访问 ISP/IAP.
// 参数: non.
// 返回: non.
// 版本: V1.0
//==========================================================================
void  DisableEEPROM(void)
{
    IAP_CONTR = 0;                                  //禁止 ISP/IAP 操作
    IAP_TPS   = 0;
    IAP_CMD   = 0;                                  //去除 ISP/IAP 命令
    IAP_TRIG  = 0;                                   //防止 ISP/IAP 命令误触发
    IAP_ADDRE = 0xff;                               //清 0 地址高字节
    IAP_ADDRH = 0xff;                               //清 0 地址高字节
    IAP_ADDRL = 0xff;                               //清 0 地址低字节，指向非 EEPROM 区，防止误操作
}

//==========================================================================
// 函数: void EEPROM_read_n(u16 EE_address,u8 *DataAddress,u16 number)
// 描述: 从指定 EEPROM 首地址读出 n 个字节放指定的缓冲.
// 参数: EE_address:   读出 EEPROM 的首地址.
//       DataAddress:  读出数据放缓冲的首地址.
//       number:       读出的字节长度.
// 返回: non.
// 版本: V1.0
//==========================================================================
void EEPROM_read_n(u32 EE_address,u8 *DataAddress,u16 number)
{
```

```
    if(PassWord == D_PASSWORD)                  //口令正确才会操作 EEPROM
    {
        EA = 0;                                 //禁止中断
        IAP_CONTR = IAP_EN;                     //允许 ISP/IAP 操作
        IAP_TPS = (u8)(MAIN_Fosc / 1000000L);   //工作频率设置
        IAP_READ();                             //送字节读命令，命令不需改变时，不需重新送命令
        do
        {
        IAP_ADDRE = EE_address / 65536;         //送地址高字节（地址需要改变时才需重新送地址）
        IAP_ADDRH = (EE_address / 256) % 256;   //送地址高字节（地址需要改变时才需重新送地址）
        IAP_ADDRL = EE_address % 256;           //送地址低字节
            if(PassWord == D_PASSWORD)          //口令口令正确才触发操作
            {
                IAP_TRIG = 0x5A;                //先送 5AH，再送 A5H 到 ISP/IAP 触发寄存器，
                                                //每次都需要如此
                IAP_TRIG = 0xA5;                //送完 A5H 后，ISP/IAP 命令立即被触发启动
            }                                   //CPU 等待 IAP 完成后，才会继续执行程序。
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            *DataAddress = IAP_DATA;             //读出的数据送往
            EE_address++;
            DataAddress++;
        }while(--number);

        DisableEEPROM();
        EA = 1;                                 //重新允许中断
    }
    PassWord = 0;                               //清除口令
}

/********************  扇区擦除函数  *****************/
//=========================================================================
//  函数: void EEPROM_SectorErase(u16 EE_address)
//  描述:  把指定地址的 EEPROM 扇区擦除.
//  参数: EE_address:    要擦除的扇区 EEPROM 的地址.
//  返回: non.
//  版本: V1.0
//=========================================================================
void EEPROM_SectorErase(u32 EE_address)
{
    if(PassWord == D_PASSWORD)                  //口令正确才会操作 EEPROM
    {
        EA = 0;                                 //禁止中断
                                                //只有扇区擦除，没有字节擦除，512 字节/扇区。
                                                //扇区中任意一个字节地址都是扇区地址。
        IAP_ADDRE = EE_address / 65536;         //送地址高字节（地址需要改变时才需重新送地址）
        IAP_ADDRH = (EE_address / 256) % 256;   //送地址高字节（地址需要改变时才需重新送地址）
        IAP_ADDRL = EE_address % 256;           //送地址低字节
        IAP_CONTR = IAP_EN;                     //允许 ISP/IAP 操作
        IAP_TPS = (u8)(MAIN_Fosc / 1000000L);   //工作频率设置
        IAP_ERASE();                            //送扇区擦除命令，命令不需改变时，不需重新送命令
            if(PassWord == D_PASSWORD)          //口令口令正确才触发操作
            {
                IAP_TRIG = 0x5A;                //先送 5AH，再送 A5H 到 ISP/IAP 触发寄存器，
                                                //每次都需要如此
                IAP_TRIG = 0xA5;                //送完 A5H 后，ISP/IAP 命令立即被触发启动
            }                                   //CPU 等待 IAP 完成后，才会继续执行程序。
```

```
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            DisableEEPROM();
            EA = 1;                                 //重新允许中断
        }
        PassWord = 0;                               //清除口令
}

//=========================================================================
// 函数: void EEPROM_write_n(u16 EE_address,u8 *DataAddress,u16 number)
// 描述:  把缓冲的 n 个字节写入指定首地址的 EEPROM.
// 参数: EE_address:    写入 EEPROM 的首地址.
//          DataAddress:  写入源数据的缓冲的首地址.
//          number:         写入的字节长度.
// 返回: non.
// 版本: V1.0
//=========================================================================
void EEPROM_write_n(u32 EE_address,u8 *DataAddress,u16 number)
{
        if(PassWord == D_PASSWORD)              //口令正确才会操作 EEPROM
        {
            EA = 0;                             //禁止中断

            IAP_CONTR = IAP_EN;                 //允许 ISP/IAP 操作
            IAP_TPS = (u8)(MAIN_Fosc / 1000000L);   //工作频率设置
            IAP_WRITE();                        //送字节写命令，命令不需改变时，不需重新送命令
            do
            {
            IAP_ADDRE = EE_address / 65536;     //送地址高字节（地址需要改变时才需重新送地址）
            IAP_ADDRH = (EE_address / 256) % 256;   //送地址高字节（地址需要改变时才需重新送地址）
            IAP_ADDRL = EE_address % 256;       //送地址低字节
                IAP_DATA    = *DataAddress;     //送数据到 IAP_DATA，只有数据改变时才需重新送
                if(PassWord == D_PASSWORD)      //口令正确才触发操作
                {
                    IAP_TRIG = 0x5A;            //先送 5AH，再送 A5H 到 ISP/IAP 触发寄存器，
                                                //每次都需要如此
                    IAP_TRIG = 0xA5;            //送完 A5H 后，ISP/IAP 命令立即被触发启动
                }                               //CPU 等待 IAP 完成后，才会继续执行程序。
                _nop_();
                _nop_();
                _nop_();
                _nop_();
                EE_address++;
                DataAddress++;
            }while(--number);

            DisableEEPROM();
            EA = 1;                             //重新允许中断
        }
        PassWord = 0;                           //清除口令
}
```

23 ADC 模数转换、传统 DAC 实现

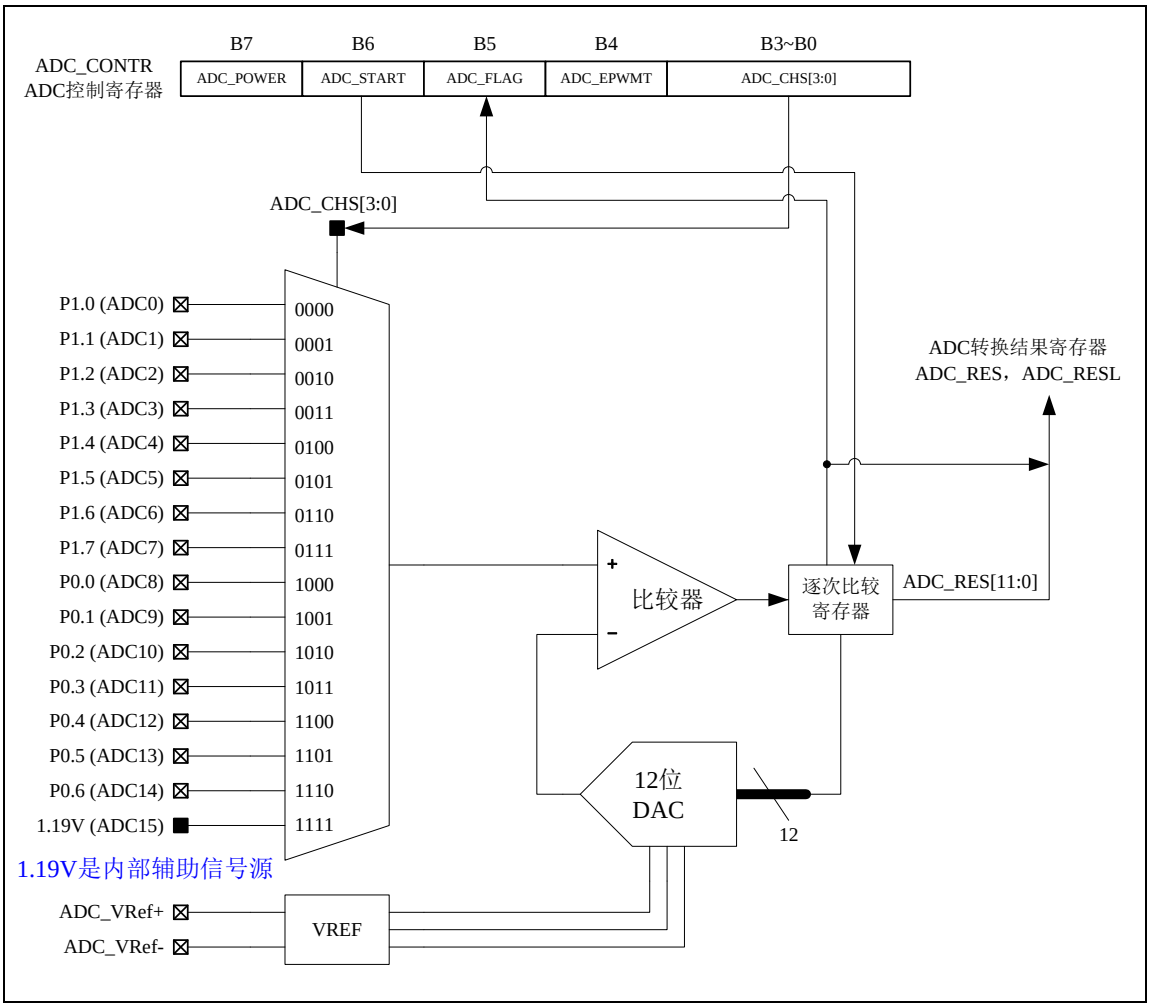
产品线	ADC 分辨率	ADC 通道数	ADCEXCFG
Ai8051U 系列	**12 位**	**15 通道**	●

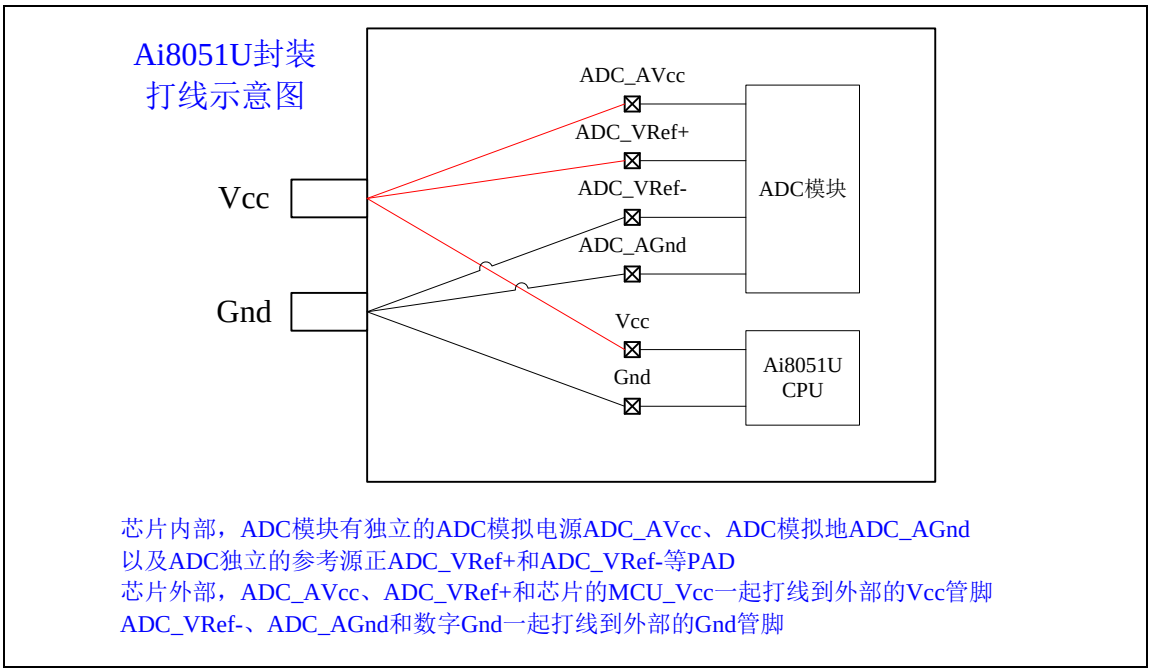
Ai8051U 系列单片机内部集成了一个 12 位高速 A/D 转换器。ADC 的时钟频率为系统频率 2 分频再经过用户设置的分频系数进行再次分频（ADC 的时钟频率范围为 SYSclk/2/1~SYSclk/2/16）。

ADC 转换结果的数据格式有两种：左对齐和右对齐。可方便用户程序进行读取和引用。

注意：ADC 的第 15 通道是专门测量内部 1.19V 参考信号源的通道，参考信号源值出厂时校准为 1.19V，由于制造误差以及测量误差，导致实际的内部参考信号源相比 1.19V，大约有±1%的误差。如果用户需要知道每一颗芯片的准确内部参考信号源值，可外接精准参考信号源，然后利用 ADC 的第 15 通道进行测量标定。

23.1 ADC 内部结构图（ADC 框图）





芯片内部，ADC模块有独立的ADC模拟电源ADC_AVcc、ADC模拟地ADC_AGnd
以及ADC独立的参考源正ADC_VRef+和ADC_VRef-等PAD
芯片外部，ADC_AVcc、ADC_VRef+和芯片的MCU_Vcc一起打线到外部的Vcc管脚
ADC_VRef-、ADC_AGnd和数字Gnd一起打线到外部的Gnd管脚

23.2 ADC 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
ADC_CONTR	ADC 控制寄存器	BCH	ADC_POWER	ADC_START	ADC_FLAG	ADC_EPWMT	ADC_CHS[3:0]				0000,0000
ADC_RES	ADC 转换结果高位寄存器	BDH									0000,0000
ADC_RESL	ADC 转换结果低位寄存器	BEH									0000,0000
ADCCFG	ADC 配置寄存器	DEH	-	-	RESFMT	-	SPEED[3:0]				xx0x,0000

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
ADCTIM	ADC 时序控制寄存器	7EFEA8H	CSSETUP	CSHOLD[1:0]		SMPDUTY[4:0]					0010,1010
ADCEXCFG	ADC 扩展配置寄存器	7EFEADH	ADCETR_PS[1:0]		ADCETRS[1:0]		-	CVTIMESEL[2:0]			0000,x000

23.2.1 ADC 控制寄存器（ADC_CONTR），PWM 触发 ADC 控制

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADC_CONTR	BCH	ADC_POWER	ADC_START	ADC_FLAG	ADC_EPWMT	ADC_CHS[3:0]			

ADC_POWER：ADC 电源控制位

0：关闭 ADC 电源

1：打开 ADC 电源。

建议进入空闲模式和主时钟停振/省电模式前将 ADC 电源关闭，以降低功耗

特别注意：

1、给 MCU 的 内部 ADC 模块电源打开后，需等待约 1ms, 等 MCU 内部的 ADC 电源稳定后再让 ADC 工作；

2、适当加长对外部信号的采样时间，就是对 ADC 内部采样保持电容的充电或放电时间,时间够，内部才能和外部电势相等。

ADC_START：ADC 转换启动控制位。写入 1 后开始 ADC 转换，转换完成后硬件自动将此位清零。

0：无影响。即使 ADC 已经开始转换工作，写 0 也不会停止 A/D 转换。

1：开始 ADC 转换，转换完成后硬件自动将此位清零。

ADC_FLAG：ADC 转换结束标志位。当 ADC 完成一次转换后，硬件会自动将此位置 1，并向 CPU 提出中断请求。此标志位必须软件清零。

ADC_EPWMT：使能 PWM 实时触发 ADC 功能。详情请参考 16 位高级 PWM 定时器章节

ADC_CHS[3:0]：ADC 模拟通道选择位

（注意：被选择为 ADC 输入通道的 I/O 口，必须设置 PxM0/PxM1 寄存器将 I/O 口模式设置为高阻输入模式。另外如果 MCU 进入主时钟停振/省电模式后，仍需要使能 ADC 通道，则需要设置 PxIE 寄存器关闭数字输入通道，以防止外部模拟输入信号忽高忽低而产生额外的功耗）

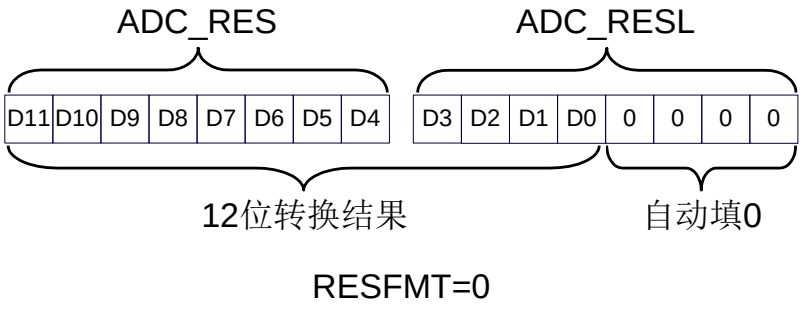
ADC_CHS[3:0]	ADC 通道	ADC_CHS[3:0]	ADC 通道
0000	P1.0	1000	P0.0
0001	P1.1	1001	P0.1
0010	P1.2	1010	P0.2
0011	P1.3	1011	P0.3
0100	P1.4	1100	P0.4
0101	P1.5	1101	P0.5
0110	P1.6	1110	P0.6
0111	P1.7	1111	测试内部 1.19V

23.2.2 ADC 配置寄存器（ADCCFG）

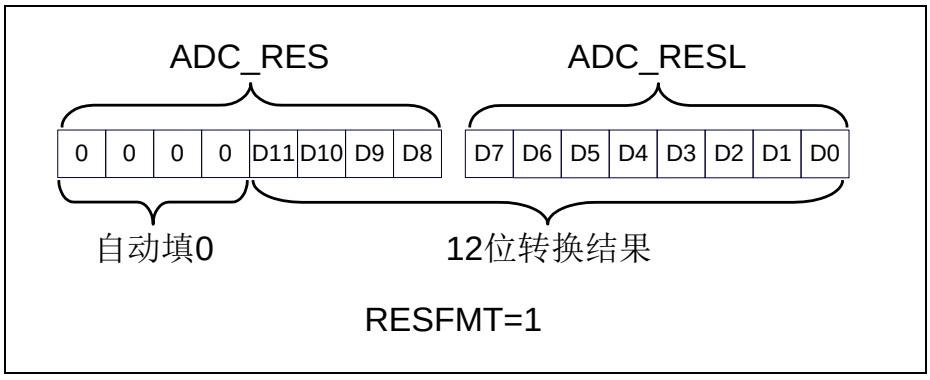
符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADCCFG	DEH	-	-	RESFMT	-	SPEED[3:0]			

RESFMT：ADC 转换结果格式控制位

0：转换结果左对齐。ADC_RES 保存结果的高 8 位，ADC_RESL 保存结果的低 4 位。格式如下：



1：转换结果右对齐。ADC_RES 保存结果的高 4 位，ADC_RESL 保存结果的低 8 位。格式如下：



SPEED[3:0]：设置 ADC 时钟｛FADC＝SYSclk/2/(SPEED+1)｝

SPEED[3:0]	ADC 时钟频率
0000	SYSclk/2/1
0001	SYSclk/2/2
0010	SYSclk/2/3
...	. . .
1101	SYSclk/2/14
1110	SYSclk/2/15
1111	SYSclk/2/16

23.2.3 ADC 转换结果寄存器（ADC_RES，ADC_RESL）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADC_RES	BDH								
ADC_RESL	BEH								

当 A/D 转换完成后，转换结果会自动保存到 ADC_RES 和 ADC_RESL 中。保存结果的数据格式请参考 ADC_CFG 寄存器中的 RESFMT 设置。

23.2.4　ADC 时序控制寄存器（ADCTIM）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADCTIM	7EFEA8H	CSSETUP	CSHOLD[1:0]		SMPDUTY[4:0]				

CSSETUP：ADC 通道选择时间控制 Tsetup

CSSETUP	ADC 时钟数
0	**1（默认值）**
1	2

CSHOLD[1:0]：ADC 通道选择保持时间控制 Thold

CSHOLD[1:0]	ADC 时钟数
00	1
01	**2（默认值）**
10	3
11	4

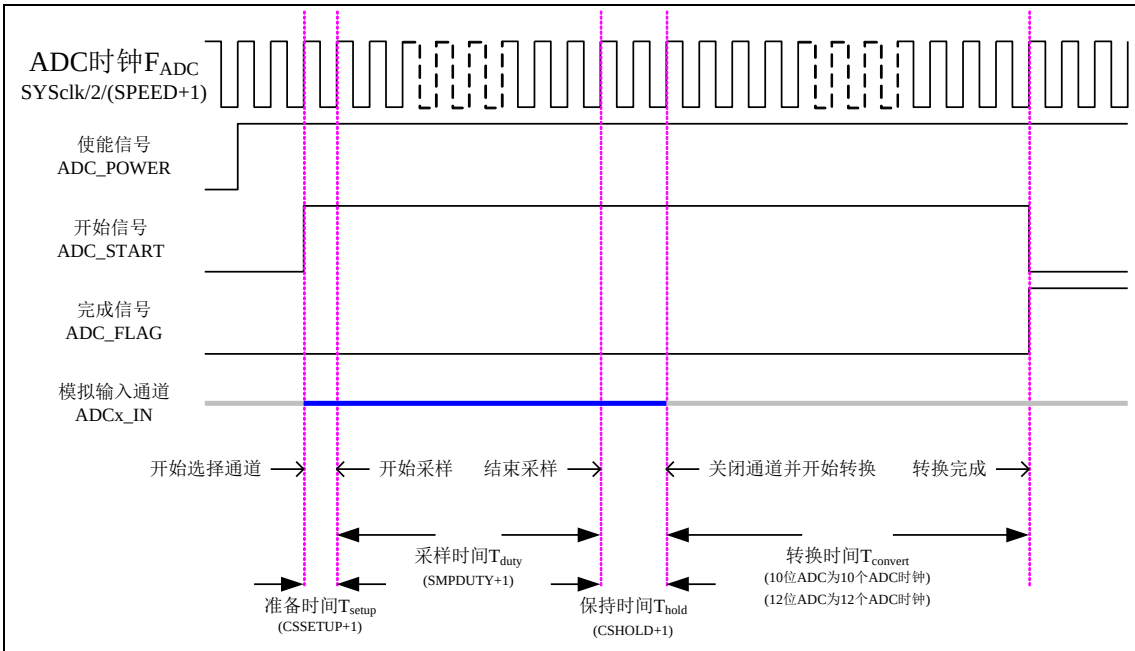
SMPDUTY[4:0]：ADC 模拟信号采样时间控制 Tduty（**注意：SMPDUTY 一定不能设置小于 01010B**）

SMPDUTY[4:0]	ADC 时钟数
00000	1
00001	2
...	...
01010	**11（默认值）**
...	...
11110	31
11111	32

ADC 数模转换时间：$T_{convert}$

12 位 ADC 的转换时间固定为 12 个 ADC 工作时钟

一个完整的 ADC 转换时间为：　$T_{setup} + T_{duty} + T_{hold} + T_{convert}$，如下图所示



ADC整体转换时序图

23.2.5 ADC 扩展配置寄存器（ADCEXCFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADCEXCFG	7EFEADH	ADCETR_PS[1:0]		ADCETRS[1:0]		-	CVTIMESEL[2:0]		

ADCETR_PS[1:0]：ADC 外部触发脚 ADC_ETR 功能脚选择

ADCETR_PS[1:0]	ADC_ETR
00	P4.7
01	P4.0
10	P2.0
11	-

ADCETRS[1:0]：ADC 外部触发脚 ADC_ETR 控制位

ADCETRS[1:0]	ADC_ETR 设置
0x	禁止 ETR 功能
10	使能 ADC_ETR 的上升沿触发 ADC
11	使能 ADC_ETR 的下降沿触发 ADC

注：使用此功能前，必须打开 ADC_CONTR 中的 ADC 电源开关，并设置好相应的 ADC 通道

CVTIMESEL[2:0]：ADC 自动转换次数选择

CVTIMESEL [2:0]	ADC 自动转换次数
0xx	转换 1 次
100	转换 2 次并取平均值
101	转换 4 次并取平均值
110	转换 8 次并取平均值
111	转换 16 次并取平均值

注：

1、当使能 ADC 自动转换多次功能后，ADC 中断标志只会在 ADC 自动转换到设置的次数后，才会被置 1（例如：设置 CVTIMESEL 为 101B，即 ADC 自动转换 4 次并取平均值，则 ADC 中断标志位每完成 4 次 ADC 转换才会被置 1）

2、当 ADC 处于 DMA 模式下时，ADCEXCFG 设置的多次转换次数无效。
ADCEXCFG 设置的重复转换次数只有在非 DMA 模式下才有效。

23.3 ADC 静态特性

符号	描述	最小值	典型值	最大值	单位
RES	分辨率	-	12	-	Bits
E_T	整体误差	-	0.5	1	LSB
E_O	偏移误差	-	-0.1	1	LSB
E_G	增益误差	-	0	1	LSB
E_D	微分非线性误差	-	0.7	1.5	LSB
E_I	积分非线性误差	-	1	2	LSB
R_{AIN}	通道等效电阻	-	∞	-	ohm
R_{ESD}	采样保持电容前串接的抗静电电阻	-	700	-	ohm
C_{ADC}	内部采样保持电容	-	16.5	-	pF

23.4 ADC 相关计算公式

23.4.1 ADC 速度计算公式

ADC 的转换速度由 ADCCFG 寄存器中的 SPEED 和 ADCTIM 寄存器共同控制。转换速度的计算公式如下所示：

$$12\text{位ADC转换速度} = \frac{\text{MCU工作频率SYSclk}}{2\times(\text{SPEED[3:0]} + 1)\times[(\text{CSSETUP} + 1) + (\text{CSHOLD} + 1) + (\text{SMPDUTY} + 1) + 12]}$$

注意：

- 12 位 ADC 的速度不能高于 800KHz
- SMPDUTY 的值不能小于 10，建议设置为 15
- CSSETUP 可使用上电默认值 0
- CHOLD 可使用上电默认值 1（ADCTIM 建议设置为 3FH）

23.4.2 ADC 转换结果计算公式

$$12\text{位ADC转换结果} = 4096 \times \frac{\text{ADC被转换通道的输入电压Vin}}{\text{AVcc电压}}$$

23.4.3 反推 ADC 输入电压计算公式

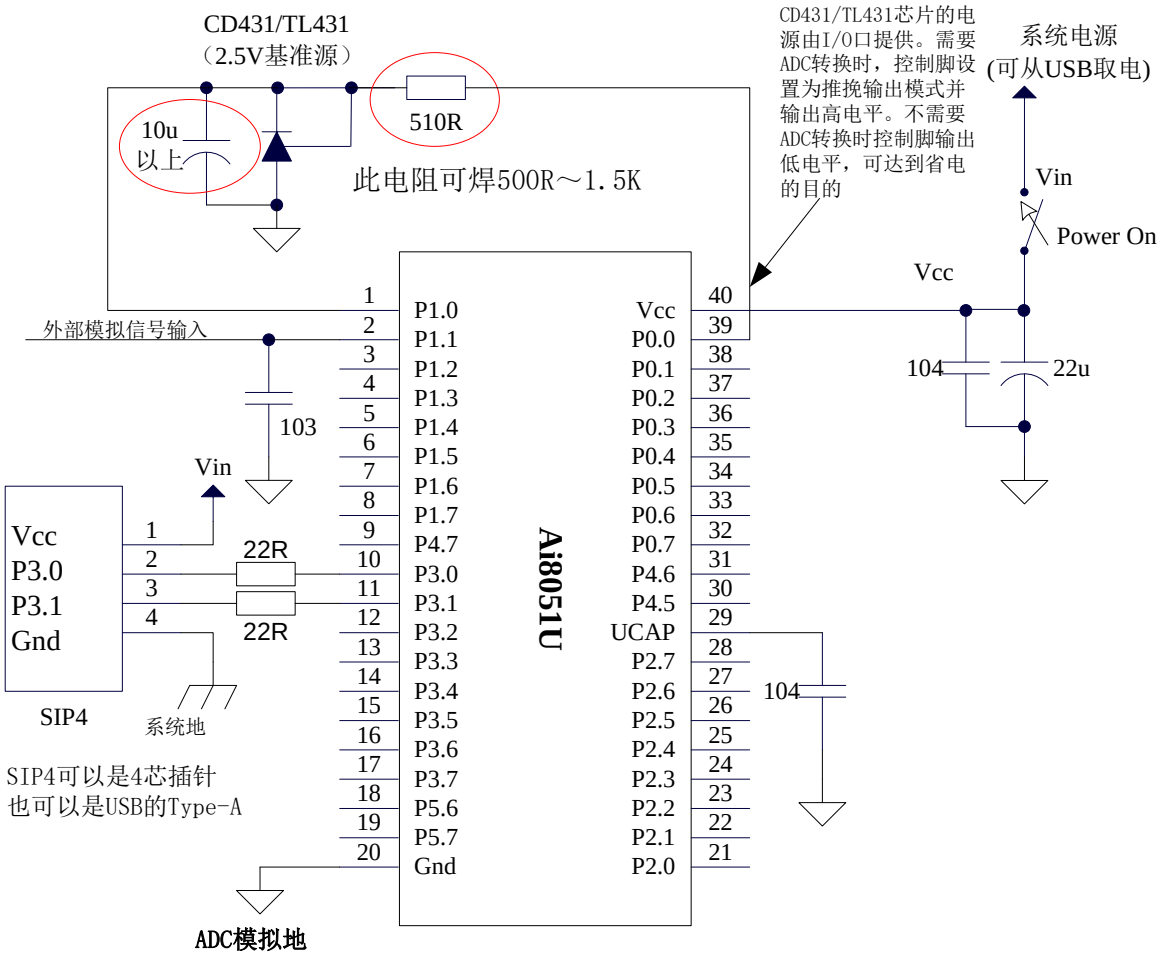
$$\text{ADC被转换通道的输入电压Vin} = \text{AVcc电压} \times \frac{12\text{位ADC转换结果}}{4096}$$

23.4.4 反推工作电压计算公式

$$\text{MCU工作电压Vcc} = 4096 \times \frac{\text{ADC被转换通道的输入电压Vin}}{12\text{位ADC转换结果}}$$

23.5 ADC 应用参考线路图

23.5.1 ADC 参考线路图，输入信号与内部 1.19V 或外接 2.5V 辅助信号源进行比较，只需计算一次



此处请当ADC模拟地处理;
请将被测量信号源的模拟地直接接到本AGnd/VRef-/Gnd管脚再去系统地

注意：

1、最后必须将系统地和ADC模拟地用短而粗的走线相连接

2、假定2次采样转换期间【ADC_VREF+ = ADC_VCC = MCU_VCC】，电压不变
使用内部1.19V或外部2.5V辅助信号源，对外部采样转换一次就知道转换后的值
使用内部1.19V辅助信号源：当前已获取了内部参考信号源电压为BGV(单位:mV)，
从CHIPID中读取，或STC-ISP烧录时指定将重要参数烧录入程序Flash
内部参考信号源的ADC15测量值为resbg，对ADC15/1.19V采样转换一次就知道转换后的值，外部通道输入电压的ADCx测量值为resx,对ADCx的外部输入信号采样转换一次就知道转换后的值，转换公式如下：
【公式1】外部通道输入电压**Vx = BGV / resbg * resx**；（Vx单位：mV）
使用外部2.5V辅助信号源：分别对输入信号和2.5V进行两次ADC转换，
外部通道输入电压的ADCx测量值为resx,外部2.5V参考信号源的ADC测量值为res25，然后计算一次就可以知道转换后的值，转换公式如下：
【公式2】外部通道输入电压**Vx = 2500mV / res25 * resx**；（Vx单位：mV）

3、详细的使用方法请参考第22.6.4章节的参考代码

利用 ADC15 通道在内部固定接的 1.19V 辅助固定信号源！

反推其他 ADCx 通道的外部输入电压，ADC0 ~ ADC14

反推 VCC，【ADC_VREF+/ADC_AVCC/MCU_VCC】

采样转换二次，只需要计算一次

===1，假定 ADC 采样转换足够快

===2，假定 2 次 ADC 转换期间，【ADC_VREF+/ADC_VCC/MCU_VCC】不变，变的误差也可以接受

===3，应用场景，【ADC_VREF+/ADC_AVCC/MCU_VCC】 这 3 条重要的电源线直接接在一起

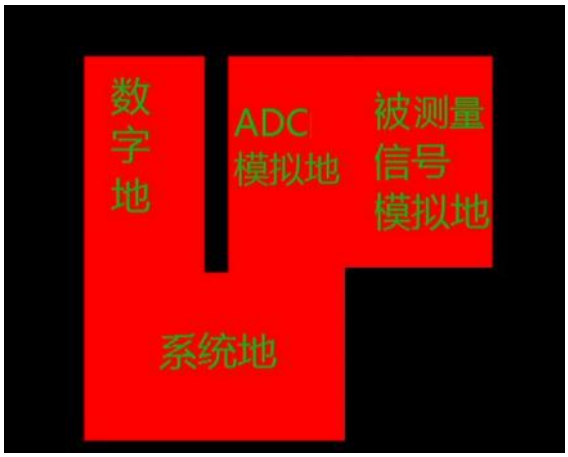
Ai8051U 系列单片机内部集成了一个 10 位/12 位高速 A/D 转换器。ADC 的时钟频率为系统频率 2 分频再经过用户设置的分频系数进行再次分频（ADC 的工作时钟频率范围为 SYSclk/2/1 到 SYSclk/2/16）。
STC8H 系列/Ai8 系列/STC32G 系列的 ADC 最快速度：12 位 ADC 为 800K（每秒进行 80 万次 ADC 转换），10 位 ADC 为 500K（每秒进行 50 万次 ADC 转换）
ADC 转换结果的数据格式有两种：左对齐和右对齐。可方便用户程序进行读取和引用。

注意：ADC 的第 15 通道是专门测量内部 1.19V 参考信号源的通道，参考信号源值出厂时校准为 1.19V，由于制造误差以及测量误差，导致实际的内部参考信号源相比 1.19V，大约有 ±1％的误差。如果用户需要知道每一颗芯片的准确内部参考信号源值，可外接精准参考信号源，然后利用 ADC 的第 15 通道进行测量标定。ADC_VRef+脚外接参考电源时，可利用 ADC 的第 15 通道可以反推 ADC_VRef+脚外接参考电源的电压；如将 ADC_VREF+短接到 MCU-VCC，就可以反推 MCU-VCC 的电压。
如果芯片有 ADC 的外部参考电源管脚 ADC_VRef+，则一定不能浮空，必须接外部参考电源或者直接连到 VCC

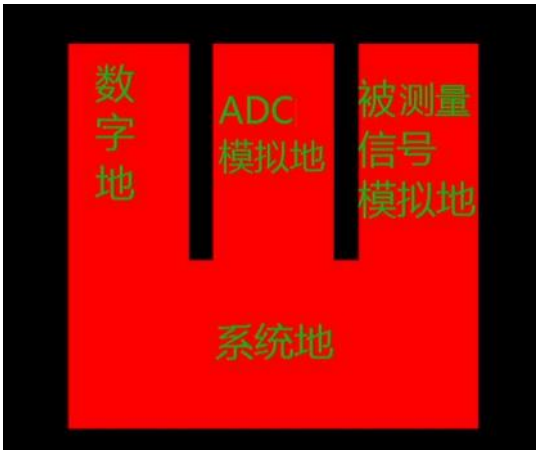
==========

使用 **ADC** 的第 **15** 通道固定接的 1.19V 辅助信号源，反推外部通道输入电压，假设：
当前已获取了内部参考信号源电压为 **BGV**，从 CHIP 中读取，或 STC-ISP 烧录时指定将重要参数烧录入程序 Flash，
内部参考信号源 **ADC15** 测量值为 **resbg**，对 ADC15/1.19V 采样转换一次就知道转换后的值；
外部通道输入电压 **ADCx** 测量值为 **resx**，对 ADCx 的外部输入信号采样转换一次就知道转换后的值
则外部通道输入电压 **Vx＝BGV / resbg * resx**；
采样转换二次，只需要计算一次
注意，是假定 2 次采样转换期间 【ADC_VREF+ = ADC_VCC = MCU_VCC】不变
===所以对外部采样转换一次，也要对内部 ADC15 接的信号源立即采样转换一次

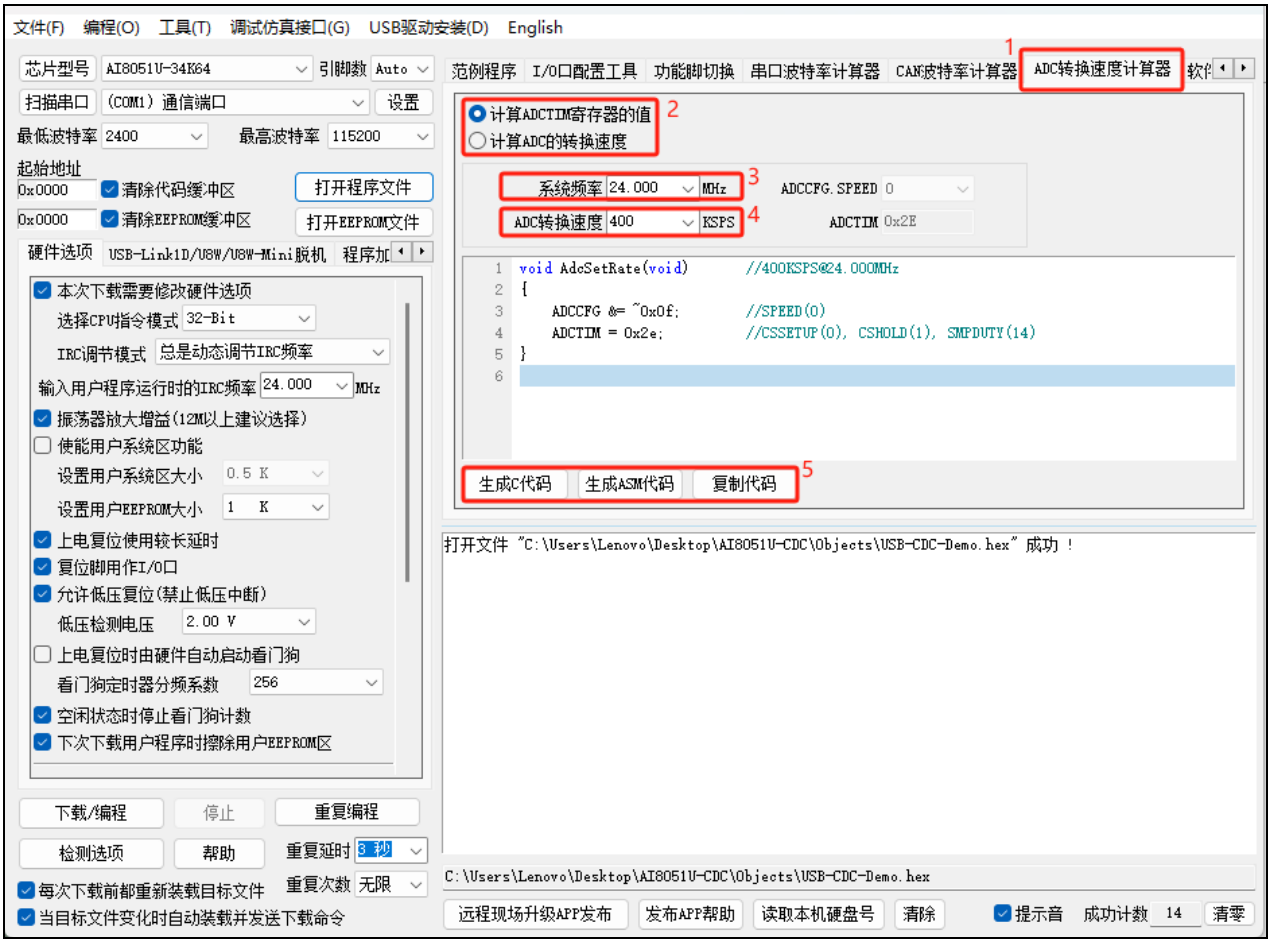
PCB 布线示意图



或者



23.6 AIapp-ISP | ADC 转换速度计算器工具



①：在下载软件中选择“ADC 转换速度计算器”功能页，进入 ADC 代码生成界面

②：选择“根据速度计算配置寄存器功能”还是“根据寄存器配置反推转换速度”

③：设置系统工作频率

④：设置 ADC 转换速度

⑤：手动生成 C 代码或者 ASM 代码，复制范例

23.7　范例程序

23.7.1　ADC 基本操作（查询方式）

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件
#include "intrins.h"                          //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void main()
{
    EAXFR = 1;                                //允许访问扩展的特殊寄存器，XFR
                                              //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                            //允许访问扩展的特殊寄存器，XFR
                                              //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                 //设置取程序代码等待时间，
                                              //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                //设置访问片内的 xdata 速度，
                                              //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P1M0 = 0x00;                              //设置 P1.0 为 ADC 口
    P1M1 = 0x01;

    ADCTIM = 0x3f;                            //设置 ADC 内部时序
    ADCCFG = 0x0f;                            //设置 ADC 时钟为系统时钟/2/16/16
    ADC_POWER = 1;                            //使能 ADC 模块

    while (1)
    {
        ADC_START = 1;                        //启动 AD 转换
        _nop_();
        _nop_();
        while (!ADC_FLAG);                    //查询 ADC 完成标志
        ADC_FLAG = 0;                         //清完成标志
        P2 = ADC_RES;                         //读取 ADC 结果
    }
}
```

23.7.2　ADC 基本操作（中断方式）

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件
#include "intrins.h"                          //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void ADC_Isr() interrupt 5
{
    ADC_FLAG = 0;                             //清中断标志
    P2 = ADC_RES;                             //读取 ADC 结果
    ADC_START = 1;                            //继续 AD 转换
}

void main()
{
    EAXFR = 1;                                //允许访问扩展的特殊寄存器，XFR
                                              //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                            //允许访问扩展的特殊寄存器，XFR
                                              //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                 //设置取程序代码等待时间，
                                              //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                //设置访问片内的 xdata 速度，
                                              //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P1M0 = 0x00;                              //设置 P1.0 为 ADC 口
    P1M1 = 0x01;

    ADCTIM = 0x3f;                            //设置 ADC 内部时序
    ADCCFG = 0x0f;                            //设置 ADC 时钟为系统时钟/2/16/16
    ADC_POWER =1;                             //使能 ADC 模块
    EADC = 1;                                 //使能 ADC 中断
    EA = 1;
    ADC_START = 1;                            //启动 AD 转换

    while (1);
}
```

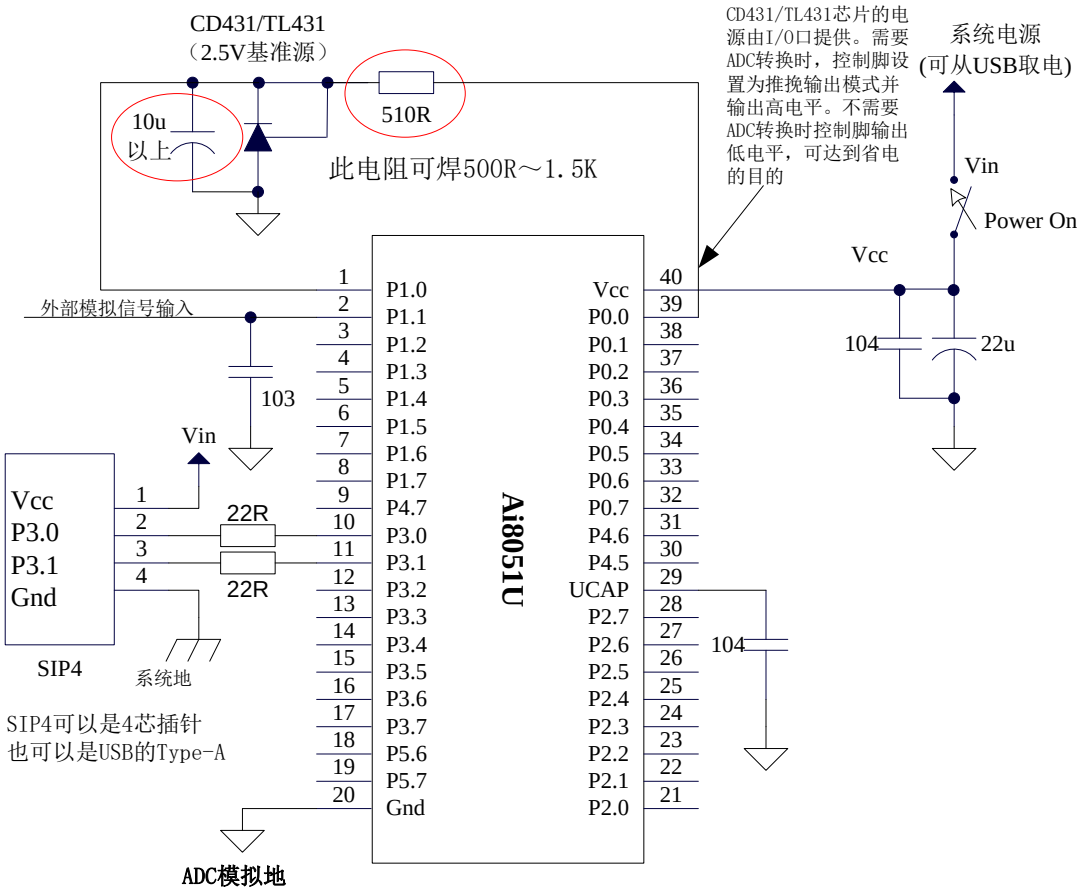
23.7.3　格式化 ADC 转换结果

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件
#include "intrins.h"                          //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void main()
{
    EAXFR = 1;                                //允许访问扩展的特殊寄存器，XFR
                                              //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                            //允许访问扩展的特殊寄存器，XFR
                                              //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                 //设置取程序代码等待时间，
                                              //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                //设置访问片内的 xdata 速度，
                                              //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P1M0 = 0x00;                              //设置 P1.0 为 ADC 口
    P1M1 = 0x01;

    ADCTIM = 0x3f;                            //设置 ADC 内部时序
    ADCCFG = 0x0f;                            //设置 ADC 时钟为系统时钟/2/16/16
    ADC_POWER = 1;                            //使能 ADC 模块
    ADC_START = 1;                            //启动 AD 转换
    _nop_();
    _nop_();
    while (!ADC_FLAG);                        //查询 ADC 完成标志
    ADC_FLAG = 0;                             //清完成标志

    ADCCFG = 0x00;                            //设置结果左对齐
    ACC = ADC_RES;                            // A 存储 ADC 的 12 位结果的高 8 位
    B = ADC_RESL;                             // B[7:4]存储 ADC 的 12 位结果的低 4 位,B[3:0]为 0

//  ADCCFG = 0x20;                            //设置结果右对齐
//  ACC = ADC_RES;                            // A[3:0]存储 ADC 的 12 位结果的高 4 位,A[7:4]为 0
//  B = ADC_RESL;                             // B 存储 ADC 的 12 位结果的低 8 位

    while (1);
}
```

23.7.4　利用 ADC15 通道在内部固定接的 1.19V 辅助固定信号源，反推其他通道的输入电压或 VCC，只需计算一次

注意：这里的 1.19V 是 ADC15 通道的固定输入信号源，1.19V

Ai8051U 系列 ADC 的第 15 通道用于测量内部参考信号源，由于内部参考信号源很稳定，约为 1.19V，且不会随芯片的工作电压的改变而变化，所以可以通过测量内部 1.19V 参考信号源，然后通过 ADC 的值便可反推出外部电压或外部电池电压。下图为参考线路图:



此处请当ADC模拟地处理;
请将被测量信号源的模拟地直接接到本AGnd/VRef-/Gnd管脚再去系统地

注意：

1、最后必须将**系统地**和**ADC模拟地**用**短而粗的走线相连接**

2、假定2次采样转换期间【ADC_VREF+ = ADC_VCC = MCU_VCC】，电压不变
使用内部1.19V或外部2.5V辅助信号源，对外部采样转换一次就知道转换后的值
使用内部1.19V辅助信号源：当前已获取了内部参考信号源电压为**BGV**(单位:mV)，
从CHIPID中读取，或STC-ISP烧录时指定将重要参数烧录入程序Flash
内部参考信号源的ADC15测量值为**resbg**，对ADC15/1.19V采样转换一次就知道转换后的值，外部通道输入电压的ADCx测量值为**resx**,对ADCx的外部输入信号采样转换一次就知道转换后的值，转换公式如下：
【公式1】外部通道输入电压**Vx = BGV / resbg * resx**；（Vx单位：mV）
使用外部2.5V辅助信号源：分别对输入信号和2.5V进行两次ADC转换，
外部通道输入电压的ADCx测量值为**resx**,外部2.5V参考信号源的ADC测量值为**res25**，然后计算一次就可以知道转换后的值，转换公式如下：
【公式2】外部通道输入电压**Vx = 2500mV / res25 * resx**；（Vx单位：mV）

3、详细的使用方法请参考第22.6.4章节的参考代码

利用 ADC15 通道在内部固定接的 1.19V 辅助固定信号源！

反推其他 ADCx 通道的外部输入电压，ADC0 ~ ADC14

反推 VCC，【ADC_VREF+/ADC_AVCC/MCU_VCC】

采样转换二次，只需要计算一次

===1，假定 ADC 采样转换足够快

===2，假定 2 次 ADC 转换期间，【ADC_VREF+/ADC_VCC/MCU_VCC】不变，变的误差也可以接受

===3，应用场景，【ADC_VREF+/ADC_AVCC/MCU_VCC】 这 3 条重要的电源线直接接在一起

Ai8051U 系列单片机内部集成了一个 10 位/12 位高速 A/D 转换器。ADC 的时钟频率为系统频率 2 分频再经过用户设置的分频系数进行再次分频（ADC 的工作时钟频率范围为 SYSclk/2/1 到 SYSclk/2/16）。
STC8H 系列的 ADC 最快速度：12 位 ADC 为 800K（每秒进行 80 万次 ADC 转换），10 位 ADC 为 500K（每秒进行 50 万次 ADC 转换）
ADC 转换结果的数据格式有两种：左对齐和右对齐。可方便用户程序进行读取和引用。

注意：ADC 的第 15 通道是专门测量内部 1.19V 参考信号源的通道，参考信号源值出厂时校准为 1.19V，由于制造误差以及测量误差，导致实际的内部参考信号源相比 1.19V，大约有 ±1％的误差。如果用户需要知道每一颗芯片的准确内部参考信号源值，可外接精准参考信号源，然后利用 ADC 的第 15 通道进行测量标定。ADC_VRef+脚外接参考电源时，可利用 ADC 的第 15 通道可以反推 ADC_VRef+脚外接参考电源的电压；如将 ADC_VREF+短接到 MCU-VCC，就可以反推 MCU-VCC 的电压。
如果芯片有 ADC 的外部参考电源管脚 ADC_VRef+，则一定不能浮空，必须接外部参考电源或者直接连到 VCC
==========
使用 **ADC** 的第 **15** 通道固定接的 1.19V 辅助信号源，反推外部通道输入电压，假设：
当前已获取了内部参考信号源电压为 **BGV**，从 CHIP 中读取，或 STC-ISP 烧录时指定将重要参数烧录入程序 Flash，
内部参考信号源 **ADC15** 测量值为 **resbg**，对 ADC15/1.19V 采样转换一次就知道转换后的值；
外部通道输入电压 **ADCx** 测量值为 **resx**，对 ADCx 的外部输入信号采样转换一次就知道转换后的值
则外部通道输入电压 **Vx＝BGV / resbg * resx**；
采样转换二次，只需要计算一次
注意，是假定 2 次采样转换期间 【ADC_VREF+ = ADC_VCC = MCU_VCC】不变
===所以对外部采样转换一次，也要对内部 ADC15 接的信号源立即采样转换一次

下面的范例使用的是内部 1.19V 作为辅助信号源，如果需要使用外部 2.5V 作为辅助信号源，计算方法可参考上图中的【公式 2】，程序代码请自行修改

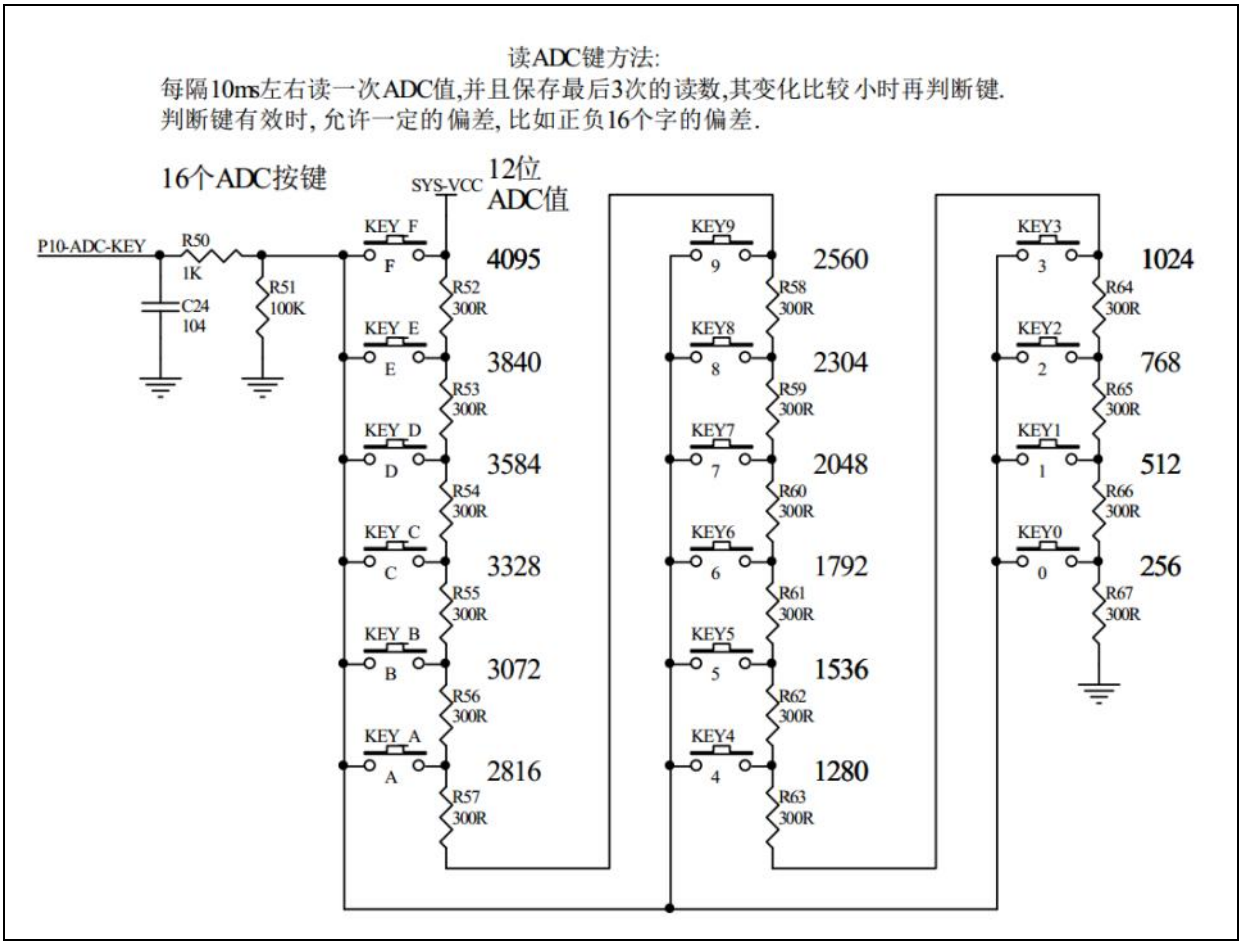
```
//测试工作频率为11.0592MHz

#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   FOSC          11059200UL
#define   BRT           (65536 - (FOSC / 115200+2) / 4)
                                                        //加2 操作是为了让 Keil 编译器
                                                        //自动实现四舍五入运算

#define   VREFH_ADDR    CHIPID7
#define   VREFL_ADDR    CHIPID8

bit       busy;
int       BGV;

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    T1x12 = 1;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void ADCInit()
{
    ADCTIM = 0x3f;                                      //设置 ADC 内部时序
    ADCCFG = 0x2f;                                      //设置 ADC 时钟为系统时钟/2/16
    ADC_CONTR = 0x8f;                                   //使能 ADC 模块,并选择第 15 通道
}

int   ADCRead()
```

```
{
    int res;

    ADC_START = 1;                          //启动 AD 转换
    _nop_();
    _nop_();
    while (!ADC_FLAG);                      //查询 ADC 完成标志
    ADC_FLAG = 0;                           //清完成标志

    res = (ADC_RES << 8) | ADC_RESL;        //读取 ADC 结果

    return res;
}

void main()
{
    int res;
    int vcc;
    int i;

    EAXFR = 1;                              //允许访问扩展的特殊寄存器，XFR
                                            //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                          //允许访问扩展的特殊寄存器，XFR
                                            //(8 位模式请使用这句，注释上一句)
    WTST = 0;                               //设置取程序代码等待时间，
                                            //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                              //设置访问片内的 xdata 速度，
                                            //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    BGV = (VREFH_ADDR << 8) + VREFL_ADDR;   //从 CHIPID 中读取内部参考电压值

    ADCInit();                              //ADC 初始化
    UartInit();                             //串口初始化

    ES = 1;
    EA = 1;

//  ADCRead();
//  ADCRead();                              //前两个数据建议丢弃

    res = 0;
    for (i=0; i<8; i++)
    {
        res += ADCRead();                   //读取 8 次数据
    }
    res >>= 3;                              //取平均值
```

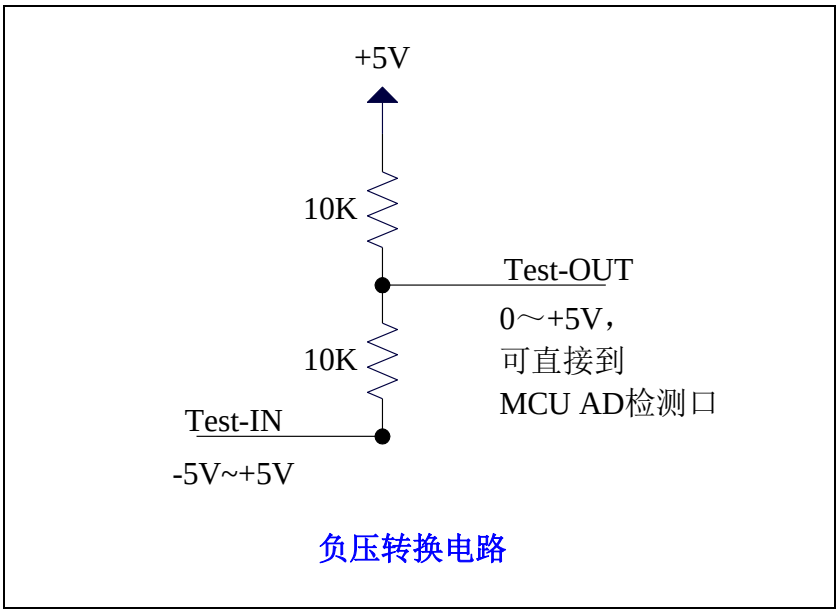
```
    vcc = (int)(4096L * BGV / res);                 //(12 位 ADC 算法)计算 VREF 管脚电压,即电池电压
                                                    //注意,此电压的单位为毫伏(mV)
    UartSend(vcc >> 8);                             //输出电压值到串口
    UartSend(vcc);

    while (1);
}
```

上面的方法是使用 ADC 的第 15 通道反推外部电池电压的。在 ADC 测量范围内，ADC 的外部测量电压与 ADC 的测量值是成正比例的，所以也可以使用 ADC 的第 15 通道反推外部通道输入电压，假设当前已获取了内部参考信号源电压为 BGV，内部参考信号源的 ADC 测量值为 res_{bg}，外部通道输入电压的 ADC 测量值为 res_x，则外部通道输入电压 **$V_x = BGV / res_{bg} * res_x$**;

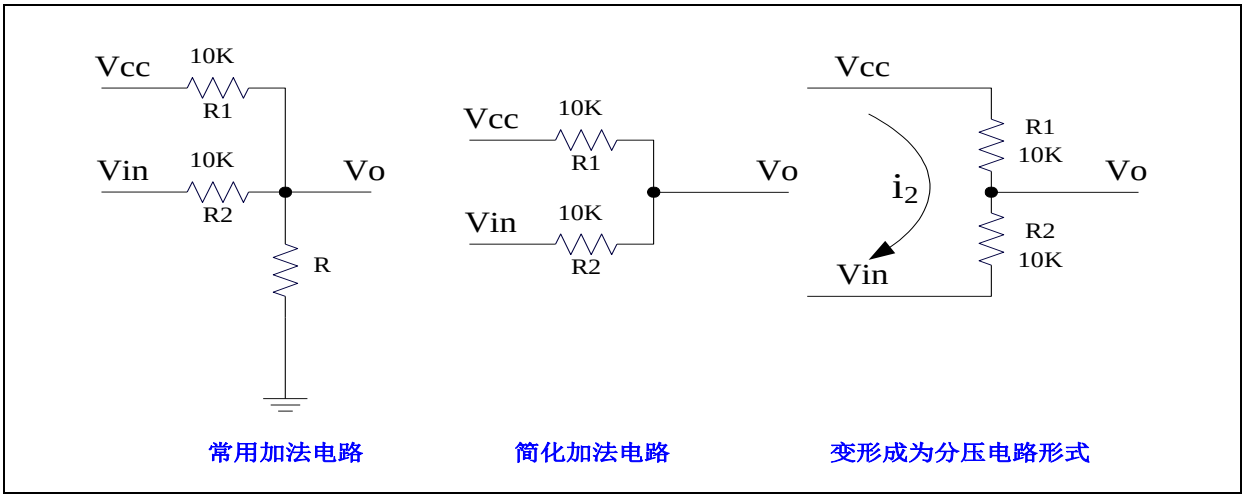
23.7.5 ADC 作按键扫描应用线路图



23.7.6 检测负电压参考线路图



23.7.7　常用加法电路在 ADC 中的应用



参照分压电路得到公式 1

公式 1：Vo = Vin + i_2 * R2

公式 2：i_2 = (Vcc - Vin) / (R1 + R2)｛条件：流向 Vo 的电流 ≈0｝

将 R1=R2 代入公式 2 得公式 3

公式 3：i_2 = (Vcc - Vin) / 2R2

将公式 3 代入公式 1 得公式 4

公式 4：Vo = (Vcc + Vin) / 2

根据公式 4，可以将以上电路看成加法电路。

在单片机的模数转换测量中，要求被测电压大于 0 并且小于 VCC。如果被测电压小于 0V，可以利用加法电路将被测电压提升到 0V 以上。此时对被测电压的变化范围有一定的要求：

把上述条件代入公式 4 可得到下面 2 式

(Vcc + Vin) / 2 > 0　　即 Vin > -Vcc

(Vcc + Vin) / 2 < Vcc　　即 Vin < Vcc

上面 2 式可以合起来：　**-Vcc < Vin < Vcc**

23.8　大型综合实战 -- 深大课程设计：模拟 RTC 时钟+数码管+ADC 查询方式检测按键

23.8.1 实验介绍

1、本项目主要熟悉如何管理多文件项目
2、熟悉 ADC 按键扫描方法
3、了解使用 74HC595 控制 LED 数码管
4、软件模拟 RTC，实现 RTC 的时、分、秒控制和设置

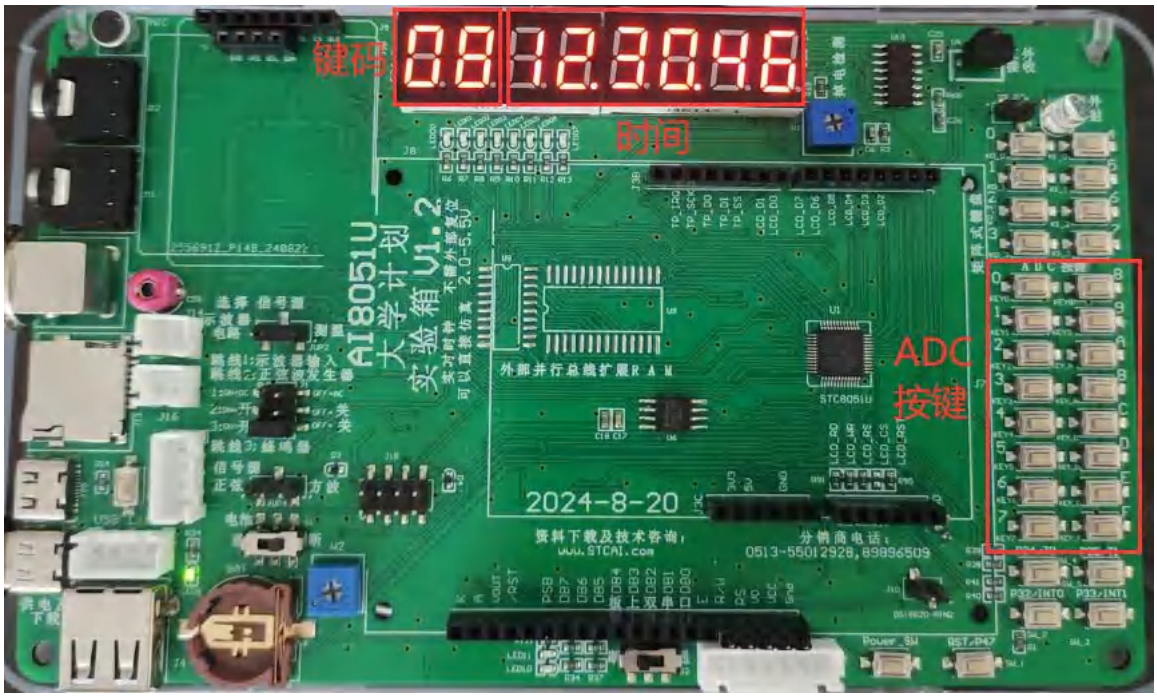
a) 按键扫描说明：

如按键要判断为正常按下，而不是抖动，需要该按键保持按下持续时间是 20ms ~ 50ms；
主循环中有个 10ms 的时间基准，将按键扫描程序作为 10ms 事件，按键的值检测 3 次都在有效范围内，即 30ms 去抖动处理，得到稳定的键码。

b) ADC 按键实验现象和操作说明：

将本项目的 HEX 文件下载到 Ai8051U 的实验箱后

1、在 LED 数码管上面低 6 位会动态显示当前的 RTC 时间，高 2 位显示按下的 ADC 按键的键码：



2、按下 16 个 ADC 按键之一后：
- 在 LED 数码管的高 2 位会显示当前的按键键码

3、ADC 按键中的按键 F，是 RTC 模式设置键
- 上电后 RTC 为正常工作模式，此时数码管动态显示 RTC 时间

- 按一次按键 F，RTC 会切换为小时设置模式，此时 RTC 的小时值会 0.5 秒闪烁一次
- 再按一次按键 F，RTC 会切换为分钟设置模式，此时 RTC 的分钟值会 0.5 秒闪烁一次
- 再按一次按键 F，RTC 会切换为秒钟设置模式，此时 RTC 的秒钟值会 0.5 秒闪烁一次
- 再按一次按键 F，恢复为 RTC 正常工作模式

4、矩阵按键中的按键 D 和 E 分别是上调键和下调键

- 只有在 RTC 的时/分/秒设置模式时，上调键和下调键才有效

5、时钟设置模式时，长按按键 D 或 E，可实现快速设置功能

c) Keil 环境下多文件项目管理说明

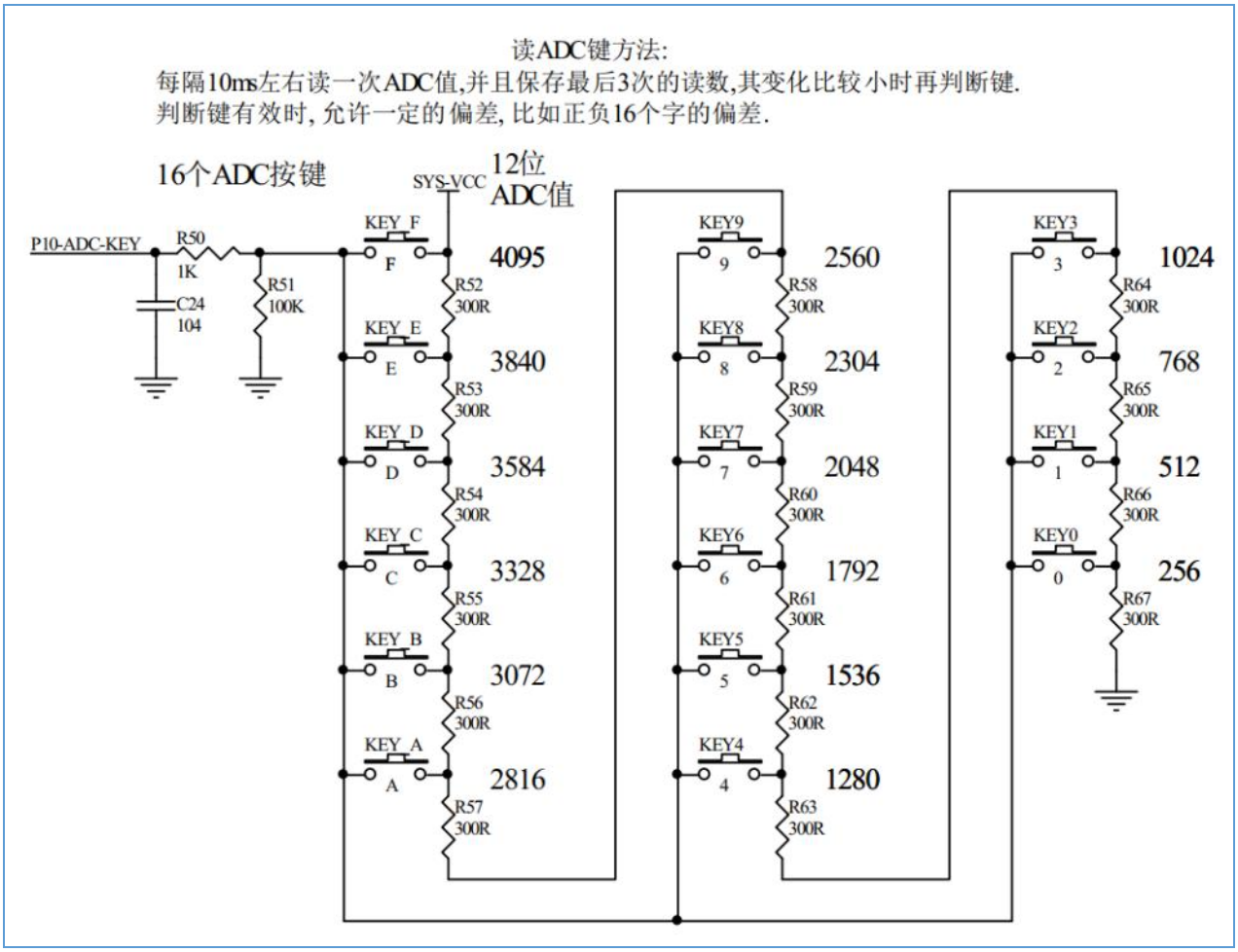
当项目的功能比较复杂时，就需要在 Keil 中建立多文件项目，以方便分工合作、代码复用、模块化管理、增强可读性和可维护性。

- 比较好的建议是将项目功能模块化，不同模块的实现代码放在不同的.c 文件中。
- 一般建议是一个模块对应一个.c 程序文件和一个.h 头文件
- 模块的初始化函数以及相关的数据处理函数都在.c 文件中实现
- 与模块相关的全局变量也必须在.c 文件中进行定义，一定不能在.h 文件中定义变量
- 如果有其他模块需要使用本模块定义的变量或函数，则这些函数和变量都需要在.h 文件中声明。
- 特别提醒：在.h 文件中声明外部变量必须使用 **extern** 关键字，否则就变成变量定义了，这样会出现变量重复定义的错误
- 为防止头文件被多次包含而产生错误或者警告，在头文件中使用类似如下的条件编译组合语句，可避免在同一个.c 文件中对同一个.h 头文件进行多次包含：

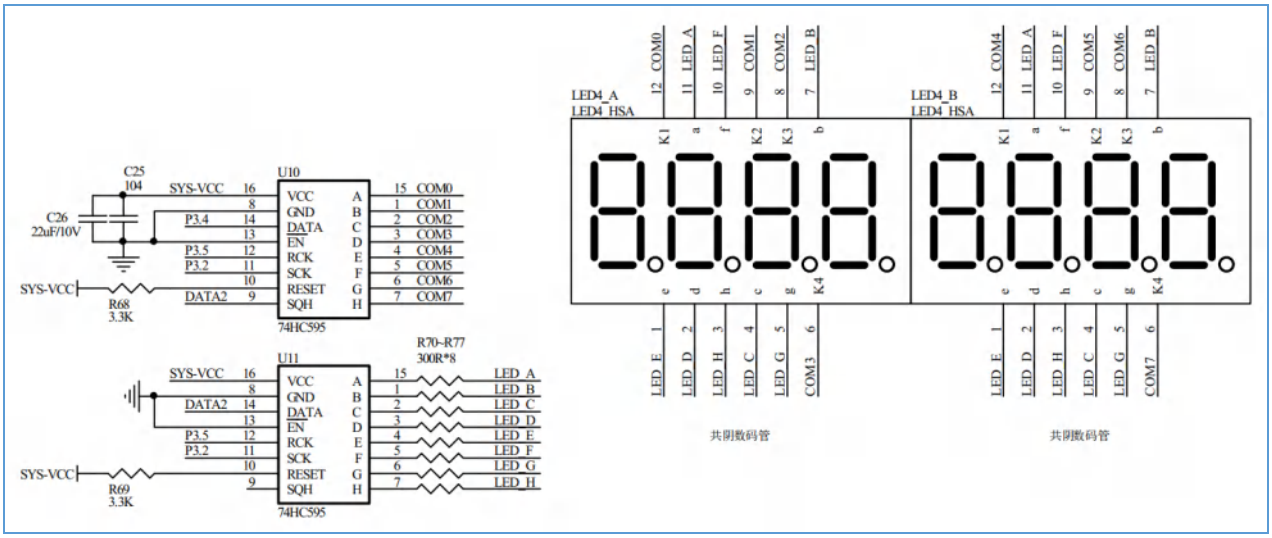
```
#ifndef XXXX
#define XXXX
...
#endif
```

23.8.2 原理图

a) ADC 按键原理图



b) 74HC595 驱动数码管原理图

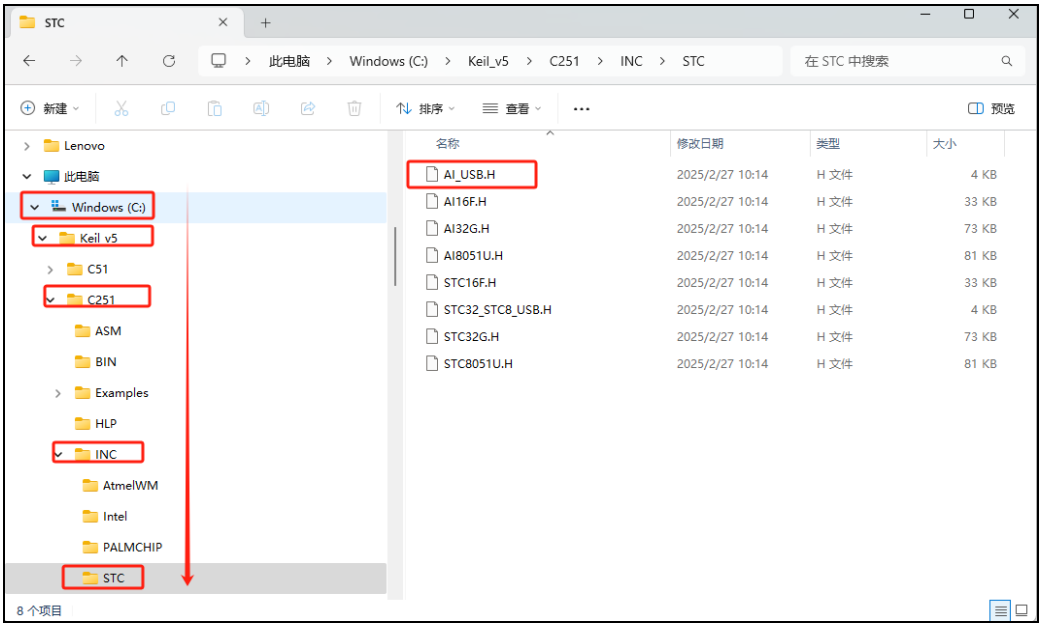


23.8.3 准备工作--添加头文件和库函数

a) 添加常用头文件 ai_usb.h

#include "ai_usb.h"　　//使用 usb 所需要包含的头文件。

这个头文件在使用 AIapp-ISP 软件添加型号和头文件到 Keil 中时就已经一并被添加到 Keil 中了。

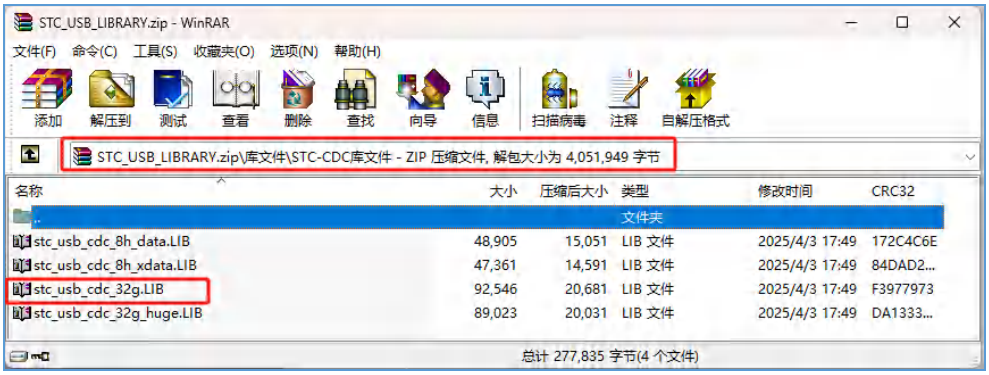


b) 添加 usb_cdc 库函数文件

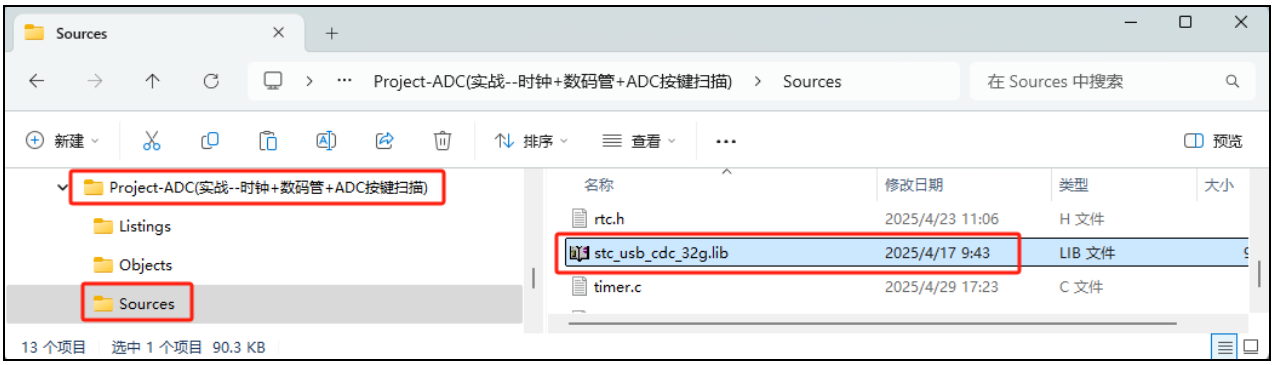
这部分可以到 STCAI 的官网：https://www.stcai.com/khs 的 USB 库文件中进行下载

下载完成后，打开压缩包“STC_USB_LIBRARY.zip”，

在‘库文件\查询模式 STC-CDC 库文件’中，复制出来 stc_usb_cdc_32g.LIB

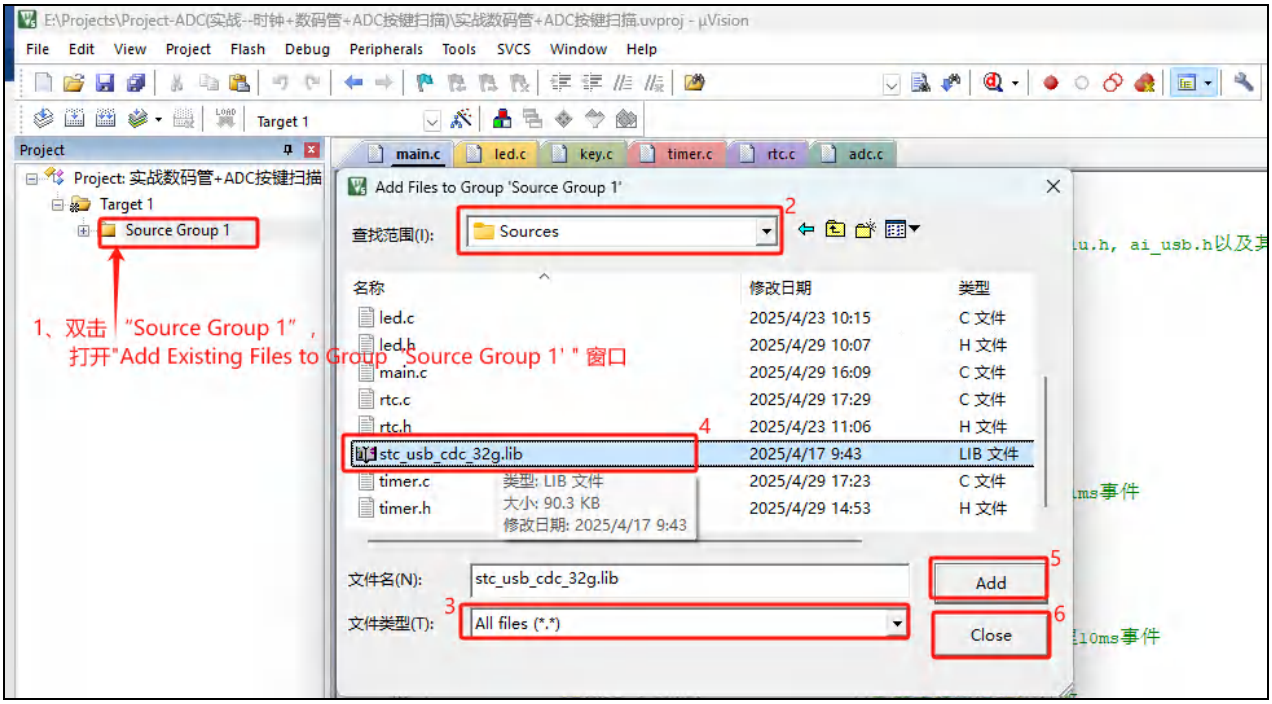


将库函数“stc_usb_cdc_32g.LIB”复制到工程所在的文件夹 Project-ADC(实战--时钟+数码管+ADC 按键扫描)>Sources 目录下：

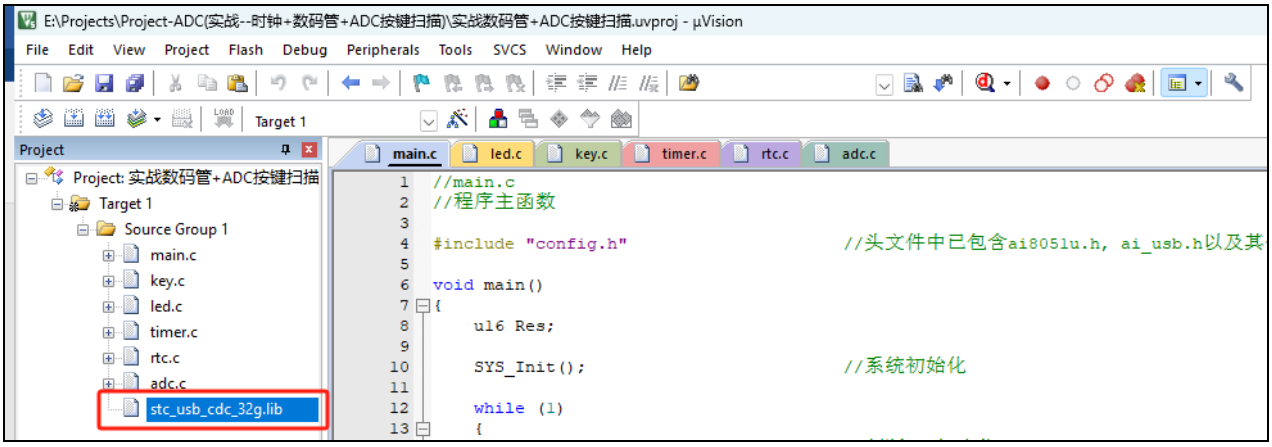


将库函数 stc_usb_cdc_32g.LIB 添加进入此工程项目中

在 Keil 中，双击 Source Group，在弹出的 Add Files to Group 'Source Group 1 窗口中，将库函数 stc_usb_cdc_32g.LIB 添加进入此工程，就可以使用 USB-CDC 功能。



这样，Keil 界面变化如下图：



23.8.4 实验程序代码

a) main.c -- 程序主函数

```
//main.c
//程序主函数

#include "config.h"              //头文件中已包含 ai8051u.h, ai_usb.h 以及其他头文件

void main()
{
    u16 Res;

    SYS_Init();                  //系统初始化

    while (1)
    {
        if (f1ms)                //判断 1ms 标志位
        {
            f1ms = 0;            //清除 1ms 标志，并处理 1ms 事件
            ScanLed();           //LED 数码管扫描
        }

        if (f10ms)               //判断 10ms 标志位
        {
            f10ms = 0;           //清除 10ms 标志，并处理 10ms 事件
            Res = Get_ADC12bitResult(0);    //检测 ADC 按键
            if(Res < 4096)       //判断采样数值是否有效
            {
                CalculateAdcKey(Res);       //判断按键
            }
            KeyEvents();         //按键处理事件
        }

        if (f500ms)              //判断 500ms 标志位
        {
            f500ms = 0;          //清除 500ms 标志，并处理 500ms 事件
            RTC_Update();        //更新软件模拟的 RTC 时间
            RTC_Show();          //在 LED 数码管上显示 RTC 时间
        }
    }
}

void SYS_Init(void) //系统初始化函数
{
```

```
    EAXFR = 1;          //使能访问扩展 XFR
    WTST = 0x00;        //设置最快速度访问程序代码
    CKCON = 0x00;       //设置最快速度访问内部 XDATA

    P0M0 = 0x00; P0M1 = 0x00;       //初始化 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x01;       //初始化 P1 口为准双向口模式, P1.0 高阻输入
    P2M0 = 0x00; P2M1 = 0x00;       //初始化 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x00;       //初始化 P3 口为准双向口模式
    P4M0 = 0x00; P4M1 = 0x00;       //初始化 P4 口为准双向口模式
    P5M0 = 0x01; P5M1 = 0x00;       //设置 P5.0 口为强推挽, 其他设置为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;       //初始化 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;       //初始化 P7 口为准双向口模式

    usb_init();         //USB 库函数初始化
    Led_Init();         //LED 数码管初始化
    Timer0_Init();      //定时器 0 初始化,用于产生 1ms 和 500ms 的时间基准
    RTC_Init();         //软件模拟 RTC 时间初始化
    ADC_Init();         //ADC 初始化

    EA = 1;             //总中断允许位打开
}
```

b) key.c -- 按键扫描以及按键事件处理

```
//key.c
//按键扫描以及按键事件处理

#include "config.h"             //头文件中已包含 ai8051u.h, ai_usb.h 以及其他头文件

u8      ADC_KeyState=0;
u8      ADC_KeyState1=0;
u8      ADC_KeyState2=0;
u8      ADC_KeyState3=0;    //键状态
u16     ADC_KeyHoldCnt=0;   //键按下计时
u8      KeyCode=0;          //给用户使用的键码, 1~16 有效

/****************** ADC 键盘计算键码 *****************************
本 ADC 键盘方案在很多实际产品设计中, 验证了其稳定可靠, 即使按键使用导电膜,都很可靠.
16 个键,理论上各个键对应的 ADC 值为 (4096 / 16) * k = 256 * k, k = 1 ~ 16, 特别的, k=16 时,对应的 ADC 值是 4095.
但是实际会有偏差,则判断时限制这个偏差, ADC_OFFSET 为 +/- 偏差, 则 ADC 值在 (256*k-ADC_OFFSET) 与 (256*k+ADC_OFFSET)之间为键有效.
间隔一定的时间,就采样一次 ADC,比如 10ms.
为了避免偶然的 ADC 值误判, 或者避免 ADC 在上升或下降时误判, 使用连续 3 次 ADC 值均在偏差范围内时, ADC 值才认为有效.
```

```
以上算法, 能保证读键非常可靠.
**********************************************/
#define    ADC_OFFSET      64
void CalculateAdcKey(u16 adc)
{
    u8 i;
    u16 j;

    if(adc < (256-ADC_OFFSET))      //判断按键是否处于释放状态
    {
        ADC_KeyState = 0;           //键状态归 0
        ADC_KeyHoldCnt = 0;
    }

    j = 256;                        //设置初始值，KEY0 理论 ADC 值为 256
    for(i=1; i<=16; i++)            //判断当前 ADC 值是否在 16 个按键有效范围之内
    {
        if((adc >= (j - ADC_OFFSET)) && (adc <= (j + ADC_OFFSET)))    break;
                                    //判断是否在偏差范围内，是的话退出循环
        j += 256;                   //否则判断是否在下一个按键有效范围
    }

    ADC_KeyState3 = ADC_KeyState2;          //保存上次状态
    ADC_KeyState2 = ADC_KeyState1;          //保存当前状态

    if(i > 16)    ADC_KeyState1 = 0;        //键无效
    else                                    //键有效
    {
        ADC_KeyState1 = i;                  //保存当前键码
        if((ADC_KeyState3 == ADC_KeyState2) && (ADC_KeyState2 == ADC_KeyState1) &&
           (ADC_KeyState3 > 0) && (ADC_KeyState2 > 0) && (ADC_KeyState1 > 0))
                                            //三次状态相同，并且键码非 0，才认为按键有效
        {
            if(ADC_KeyState == 0)           //第一次检测到
            {
                KeyCode    = i;             //保存键码
                ADC_KeyState = i;           //保存键状态
                ADC_KeyHoldCnt = 0;
            }

            if(ADC_KeyState == i)           //连续检测到同一键按着
            {
                if(++ADC_KeyHoldCnt >= 100)       //按下 1 秒后,以 10 次每秒的速度 Repeat Key
                {
                    ADC_KeyHoldCnt = 90;
                                        //重设长按计数器，后续每  10*10ms=100ms  设置一次有效键码
```

```
                    KeyCode   = i;          //保存键码
                }
            }
            else ADC_KeyHoldCnt = 0;        //按下时间计数归 0
        }
    }
}

void KeyEvents()        //按键事件处理程序 (需每 1ms 执行一次!!!)
{
    if(KeyCode > 0)     //有键按下
    {
        SetLed(0, (BYTE)(KeyCode / 10), 0);         //在数码 0 上显示键码的十位
        SetLed(1, (BYTE)(KeyCode % 10), 0);         //在数码 1 上显示键码的个位

        switch (KeyCode)                            //处理不同的按键键码
        {
            case 16:
                RTC_SetMode();                      //按下按键 F,设置 RTC 工作模式
                break;
            case 14:
                RTC_SetUp();                        //按下按键 E,向上调整 RTC 时间
                break;
            case 15:
                RTC_SetDown();                      //按下按键 D,向上调整 RTC 时间
                break;
        }

        KeyCode = 0;        //清除键码，防止重复处理
    }
}
```

c) led.c -- 扫描 LED 数码管

```
//led.c
//使用 74HC595 动态扫描 LED 数码管
//由于 LED 数码管是由 8 个 SEG 和 8 个 COM 组成的矩阵 LED
//所以 LED 数码管也需要扫描显示

#include "config.h"             //头文件中已包含 ai8051u.h, ai_usb.h 以及其他头文件

sbit    DAT = P3^4;     //74HC595 串行数据输入脚
sbit    SCK = P3^2;     //74HC595 串行数据移入时钟脚,上升沿有效
sbit    RCK = P3^5;     //74HC595 并行数据输出时钟脚,上升沿有效
```

```
#define   LED_OFF     0x00
//定义关闭 LED 数码管的段码数据,LED 数码管为共阴极,所以 SEG 线全部输出 0 即可关闭一个 8 段数码管
#define   LED_POINT   0x80      //定义 LED 数码管小数点的段码数据

BYTE    ShowIndex;          //LED 数码管扫描显示索引(8 个数码管轮流扫描显示)
BYTE    LedBuf[8];          //8 个 LED 数码管显示数据缓冲区

BYTE    code LED_PATTERN[10] =      //定义 LED 数码管显示数据 0~9 的段码数据
{
    0x3F,     //0                   //此表为共阴极数码管的段码数据
    0x06,     //1
    0x5B,     //2
    0x4F,     //3
    0x66,     //4
    0x6D,     //5
    0x7D,     //6
    0x07,     //7
    0x7F,     //8
    0x6F,     //9
};

void Shift_74HC595(BYTE dat)            //串行移入一个字节的数据到 74HC595
{
    BYTE i;

    for (i = 0; i < 8; i++)     //移动次数控制(移动 8Bit)
    {
        dat <<= 1;              //将数据的最高位移出到 CY
        DAT = CY;               //将移出的最高位数据输出到 74HC595 的数据输入口
        SCK = 0;                //74HC595 的串行数据移入时钟脚输出低电平
        _nop_();                //等待 2 个时钟
        _nop_();
        SCK = 1;                //74HC595 的串行数据移入时钟脚输出高电平,上升沿移入数据
    }
}

void Output_74HC595(void)       //74HC595 并行数据输出控制
{
    RCK = 0;            //74HC595 的并行数据输出时钟脚输出低电平
    _nop_();            //等待 2 个时钟
    _nop_();
    RCK = 1;            //74HC595 的并行数据输出时钟脚输出高电平,上升沿输出数据
}

void Led_Init(void)     //LED 数码管初始化程序
{
```

```
    DAT = 0;            //初始化 74HC595 时钟线和数据线
    RCK = 0;
    SCK = 0;

    LedBuf[0] = LED_OFF;    //初始化 LED 数码管显示数据缓冲区,上电关闭显示
    LedBuf[1] = LED_OFF;
    LedBuf[2] = LED_OFF;
    LedBuf[3] = LED_OFF;
    LedBuf[4] = LED_OFF;
    LedBuf[5] = LED_OFF;
    LedBuf[6] = LED_OFF;
    LedBuf[7] = LED_OFF;

    ShowIndex = 0;              //初始化扫描显示索引
}

void ScanLed(void)              //LED 数码管扫描显示程序,  (需每 1ms 执行一次!!!)
{
    Shift_74HC595(LedBuf[ShowIndex]);       //将当前扫描到的 LED 数码管的段码移入到 74HC595
    Shift_74HC595(~(1 << ShowIndex));       //将当前扫描到的 LED 数码管的 COM 数据移入到 74HC595
    Output_74HC595();           //将 74HC595 中的段码数据和 COM 数据输出到 LED 数码管的管脚

    if (++ShowIndex >= 8)       //扫描索引值递增
        ShowIndex = 0;          //扫描完一轮后索引值归零
}

void SetLed (BYTE nIndex,       //设置 LED 显示函数,nIndex:设置目标数码管,0~7 分别对应 8 个数码
             BYTE nNum,         //nNum:设置显示的数据,有效值 0~9,其他值则关闭显示
             BOOL bShowPoint    //bShowPoint:设置是否显示小数点
            )
{
    if (nIndex < 8)             //目标数码管的有效值为 0~7
    {
        if (nNum < 10)          //判断显示的值是否为 0~9
            LedBuf[nIndex] = LED_PATTERN[nNum];
                                //将相应数据的段码赋值到 LED 数码管显示数据缓冲区
        else
            LedBuf[nIndex] = LED_OFF;   //如果是其他值则关闭显示

        if (bShowPoint)         //判断是否需要显示小数点
            LedBuf[nIndex] |= LED_POINT;    //如果需要显示,则增加小数点的段码数据
    }
}
```

d) timer.c -- 定时器初始化以及定时器中断服务程序

```
//timer.c
//定时器初始化以及定时器中断服务程序
//定时器 0:1ms 定时,产生 1ms 和 500ms 的标志位

#include "config.h"       //头文件中已包含 ai8051u.h, ai_usb.h 以及其他头文件

bit     f1ms;       //1ms 标志位
bit     f10ms;      //10ms 标志位
bit     f500ms;     //500ms 标志位
char    cnt10;      //10 次计数器,用于产生 10ms 的时间
int     cnt500;     //500 次计数器,用于产生 500ms 的时间

void Timer0_Init(void)      //1 毫秒@40.000MHz
{
    AUXR |= 0x80;           //定时器 0 时钟 1T 模式
    TMOD &= 0xF0;           //设置定时器 0 为模式 0
    TL0 = 0xC0;             //设置定时器 0 初始值
    TH0 = 0x63;             //设置定时器 0 初始值
    TF0 = 0;                //清除 TF0 标志
    TR0 = 1;                //定时器 0 开始计时
    ET0 = 1;                //使能定时器 0 中断

    cnt10 = 0;              //初始化 10 次计数器
    cnt500 = 0;             //初始化 500 次计数器
    f1ms = 0;               //初始化 1m 标志位
    f10ms = 0;              //初始化 10m 标志位
    f500ms = 0;             //初始化 500ms 标志位
}

void Timer0_Isr(void) interrupt 1       //定时器 0 中断服务程序
{
    f1ms = 1;               //设置 1ms 标志位

    if (cnt10++ >= 10)      //判断 10 次计数是否已满
    {
        cnt10 = 0;          //清除计数值,开始开始计数
        f10ms = 1;          //设置 10ms 标志位
    }

    if (cnt500++ >= 500)    //判断 500 次计数是否已满
    {
        cnt500 = 0;         //清除计数值,开始开始计数
        f500ms = 1;         //设置 500ms 标志位
    }
```

```
}
```

e) rtc.c -- 软件模拟 RTC 程序

```
//rtc.c
//软件模拟 RTC 程序
//包括 RTC 时间更新控制
//RTC 显示控制以及 RTC 时间调整控制

#include "config.h"       //头文件中已包含 ai8051u.h, ai_usb.h 以及其他头文件

bit     fToggle;          //0.5 秒->1 秒的控制位(两次计数器),用于在设置时间时进行闪烁控制
BYTE    bHour;            //RTC 小时计数值
BYTE    bMinute;          //RTC 分钟计数值
BYTE    bSecond;          //RTC 秒钟计数值

BYTE     bMode;           //RTC 工作模式控制
                          //0:正常工作模式
                          //1:小时设置模式
                          //2:分钟设置模式
                          //3:秒钟设置模式

void RTC_Init()           //RTC 初始化函数
{
    bHour = 12;           //初始化时间为 12:30:00
    bMinute = 30;
    bSecond = 0;
    bMode = 0;            //初始化 RTC 工作模式为正常工作模式
    fToggle = 0;          //初始化翻转控制
}

void RTC_Update()                 //RTC 时间更新函数,  (需每 500ms 执行一次!!!)
{
    fToggle = !fToggle;           //翻转控制位

    if ((fToggle == 1) &&         //如果 1 秒时间到并且为正常工作模式,则更新 RTC 时间
       (bMode == 0))
    {
        if (++bSecond >= 60)              //秒钟递增 1,并判断秒钟是否记满 60
        {
            bSecond = 0;                  //秒钟归零
            if (++bMinute >= 60)          //分钟递增 1,并判断分钟是否记满 60
            {
                bMinute = 0;              //分钟归零
                if (++bHour >= 24)        //小时递增 1,并判断小时是否记满 24
                {
```

```
                    bHour = 0;          //小时归零
                }
            }
        }
    }
}

void RTC_Show()                 //RTC 显示控制程序,不同的模式显示方式不一样
{
    switch (bMode)              //对 RTC 工作模式进行分支处理
    {
        case 0:                 //正常工作模式
            SetLed(2, (BYTE)(bHour / 10), 0);       //在数码 2 上显示小时的十位
            SetLed(3, (BYTE)(bHour % 10), 1);       //在数码 3 上显示小时的个位
            SetLed(4, (BYTE)(bMinute / 10), 0);     //在数码 4 上显示分钟的十位
            SetLed(5, (BYTE)(bMinute % 10), 1);     //在数码 5 上显示分钟的个位
            SetLed(6, (BYTE)(bSecond / 10), 0);     //在数码 6 上显示秒钟的十位
            SetLed(7, (BYTE)(bSecond % 10), 0);     //在数码 7 上显示秒钟的个位
            break;

        case 1:                 //设置 RTC 的小时模式
            if (fToggle)        //闪烁控制
            {
                SetLed(2, (BYTE)(bHour / 10), 0);   //在数码 2 上显示小时的十位
                SetLed(3, (BYTE)(bHour % 10), 1);   //在数码 3 上显示小时的个位
            }
            else
            {
                SetLed(2, -1, 0);                   //在数码 2 上关闭显示,产生闪烁效果
                SetLed(3, -1, 1);                   //在数码 3 上关闭显示
            }
            SetLed(4, (BYTE)(bMinute / 10), 0);     //在数码 4 上显示分钟的十位
            SetLed(5, (BYTE)(bMinute % 10), 1);     //在数码 5 上显示分钟的个位
            SetLed(6, (BYTE)(bSecond / 10), 0);     //在数码 6 上显示秒钟的十位
            SetLed(7, (BYTE)(bSecond % 10), 0);     //在数码 7 上显示秒钟的个位
            break;

        case 2:                                     //设置 RTC 的分钟模式
            SetLed(2, (BYTE)(bHour / 10), 0);       //在数码 2 上显示小时的十位
            SetLed(3, (BYTE)(bHour % 10), 1);       //在数码 3 上显示小时的个位
            if (fToggle)                            //闪烁控制
            {
                SetLed(4, (BYTE)(bMinute / 10), 0); //在数码 4 上显示分钟的十位
                SetLed(5, (BYTE)(bMinute % 10), 1); //在数码 5 上显示分钟的个位
            }
            else
```

```
            {
                SetLed(4, -1, 0);                       //在数码 4 上关闭显示,产生闪烁效果
                SetLed(5, -1, 1);                       //在数码 5 上关闭显示
            }
            SetLed(6, (BYTE)(bSecond / 10), 0);         //在数码 6 上显示秒钟的十位
            SetLed(7, (BYTE)(bSecond % 10), 0);         //在数码 7 上显示秒钟的个位
            break;

        case 3:                                         //设置 RTC 的秒钟模式
            SetLed(2, (BYTE)(bHour / 10), 0);           //在数码 2 上显示小时的十位
            SetLed(3, (BYTE)(bHour % 10), 1);           //在数码 3 上显示小时的个位
            SetLed(4, (BYTE)(bMinute / 10), 0);         //在数码 4 上显示分钟的十位
            SetLed(5, (BYTE)(bMinute % 10), 1);         //在数码 5 上显示分钟的个位
            if (fToggle)                                //闪烁控制
            {
                SetLed(6, (BYTE)(bSecond / 10), 0);     //在数码 6 上显示秒钟的十位
                SetLed(7, (BYTE)(bSecond % 10), 0);     //在数码 7 上显示秒钟的个位
            }
            else
            {
                SetLed(6, -1, 0);                       //在数码 6 上关闭显示,产生闪烁效果
                SetLed(7, -1, 0);                       //在数码 7 上关闭显示
            }
            break;
    }
}

void RTC_SetMode()              //RTC 工作模式控制
{
    if (++bMode > 3)            //RTC 工作模式转换(0:正常工作模式->1:小时设置模式->
                                //2:分钟设置模式->3:秒钟设置模式->0:正常工作模式->...)
    bMode = 0;                  //如果 RTC 模式大于 3,则 RTC 重归正常工作模式
}

void RTC_SetUp()                //RTC 时间上调控制
{
    if (bMode == 0)             //如果是正常工作模式,上调无意义
        return;                 //直接返回

    switch (bMode)              //根据工作模式调整时/分/秒
    {
        case 1:                     //小时设置模式
            if (++bHour >= 24)      //小时计数值向上递增
                bHour = 0;          //溢出后自动归零
            break;
        case 2:                     //分钟设置模式
```

```
            if (++bMinute >= 60)    //分钟计数值向上递增
                bMinute = 0;        //溢出后自动归零
            break;
        case 3:                     //秒钟设置模式
            if (++bSecond >= 60)    //秒钟计数值向上递增
                bSecond = 0;        //溢出后自动归零
            break;
    }

    fToggle = 1;                    //调整完后立即强制显示,不用等待 500ms
    RTC_Show();                     //显示调整的时间
}

void RTC_SetDown()                  //RTC 时间下调控制
{
    if (bMode == 0)                 //如果是正常工作模式,下调无意义
    return;                         //直接返回

    switch (bMode)                  //根据工作模式调整时/分/秒
    {
        case 1:                     //小时设置模式
            if (--bHour >= 24)      //小时计数值向下递减
                bHour = 23;         //溢出后自动设置上限值
            break;
        case 2:                     //分钟设置模式
            if (--bMinute >= 60)    //分钟计数值向下递减
                bMinute = 59;       //溢出后自动设置上限值
            break;
        case 3:                     //秒钟设置模式
            if (--bSecond >= 60)    //秒钟计数值向下递减
                bSecond = 59;       //溢出后自动设置上限值
            break;
    }

    fToggle = 1;                    //调整完后立即强制显示,不用等待 500ms
    RTC_Show();                     //显示调整的时间
}
```

f) adc.c -- ADC 初始化和 ADC 查询方式检测按键

```
//adc.c
//ADC 程序
//包括 ADC 初始化和 ADC 数据采集

#include "config.h"       //头文件中已包含 ai8051u.h, ai_usb.h 以及其他头文件
```

```
void ADC_Init()          //ADC 初始化函数
{
    ADCTIM = 0x3f;       //设置 ADC 内部时序，ADC 采样时间建议设最大值
    ADCCFG = 0x2f;       //设置 ADC 时钟为系统时钟/2/16
    ADC_CONTR = 0x80;    //使能 ADC 模块
}

u16 Get_ADC12bitResult(u8 channel)        //channel = 0~15
{
    ADC_RES = 0;
    ADC_RESL = 0;

    ADC_CONTR = (ADC_CONTR & 0xf0) | channel;          //设置 ADC 转换通道
    ADC_START = 1;              //启动 ADC 转换
    _nop_();
    _nop_();
    _nop_();
    _nop_();

    while(ADC_FLAG == 0);     //wait for ADC finish
    ADC_FLAG = 0;             //清除 ADC 结束标志
    return    (((u16)ADC_RES << 8) | ADC_RESL);
}
```

g) config.h -- 项目配置的头文件

```
#ifndef __CONFIG_H__          //防止头文件被重复包含
#define __CONFIG_H__

#define   HIRC              40000000UL
#define   FOSC              40000000UL
#define   SYSCLK            FOSC
#define   MAIN_Fosc         FOSC

#include <ai8051u.h>          //包含外部头文件
#include <stdio.h>
#include <intrins.h>

#include "def.h"              //包含项目头文件
#include "ai_usb.h"
#include "timer.h"
#include "key.h"
#include "led.h"
#include "rtc.h"
```

```
#include "adc.h"

void SYS_Init(void);                    //函数声明

#endif
```

h) key.h -- 项目配置文件的头文件

```
#ifndef __KEY_H__           //防止头文件被重复包含
#define __KEY_H__

void CalculateAdcKey(u16 adc);  //函数声明
void KeyEvents();               //函数声明

#endif
```

i) led.h -- 项目配置文件的头文件

```
#ifndef __LED_H__           //防止头文件被重复包含
#define __LED_H__

extern BYTE LedBuf[8];

void Led_Init(void);            //函数声明
void ScanLed(void);             //函数声明
void SetLed(BYTE nIndex, BYTE nNum, BOOL bShowPoint);

#endif
```

j) timer.h -- 项目配置文件的头文件

```
#ifndef __TIMER_H__         //防止头文件被重复包含
#define __TIMER_H__

void Timer0_Init(void);         //函数声明
void Timer1_Init(void);         //函数声明

extern bit f1ms;                //外部变量声明
extern bit f10ms;               //外部变量声明
extern bit f500ms;              //外部变量声明

#endif
```

k) rtc.h -- 项目配置文件的头文件

```
#ifndef __RTC_H__           //防止头文件被重复包含
```

```
#define __RTC_H__

void RTC_Init();          //函数声明
void RTC_Update();        //函数声明
void RTC_Show();          //函数声明
void RTC_SetMode();       //函数声明
void RTC_SetUp();         //函数声明
void RTC_SetDown();       //函数声明

#endif
```

l)　adc.h -- 项目配置文件的头文件

```
#ifndef __ADC_H__                 //防止头文件被重复包含
#define __ADC_H__

void ADC_Init();                  //函数声明
u16 Get_ADC12bitResult(u8 channel);   //函数声明

#endif
```

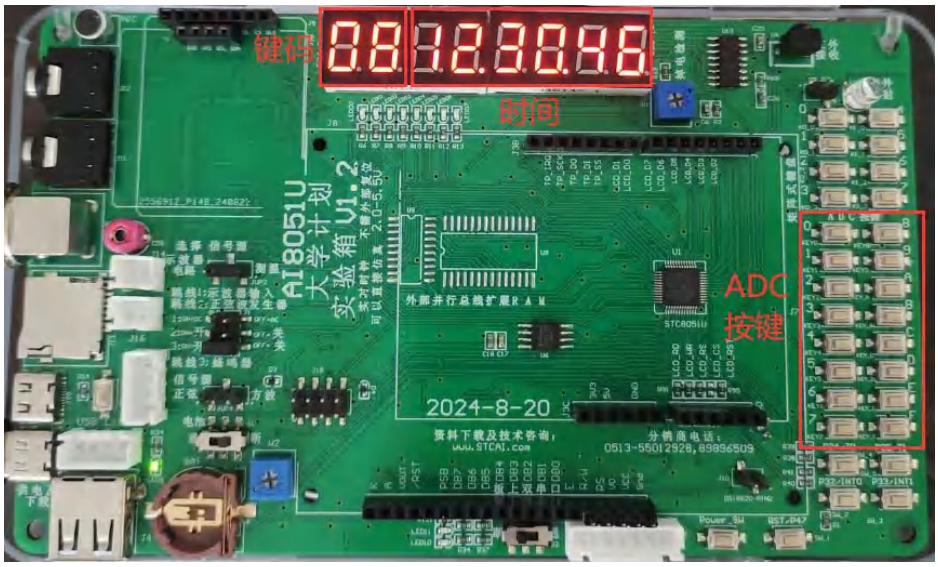
23.8.5 程序下载

录入代码，保存，编译。连接好 Ai8051U 实验箱，运行 AIapp-ISP 软件系统，按步骤打开并下载程序“实战数码管+ADC 按键扫描.hex”，如下图：

再观察 Ai8051U 实验箱，因本实验 RTC 的初始化时间为 12:30:00，那么 Ai8051U 实验箱数码低 6 位显示计时从 12:30:00 开始。

23.8.6 程序实现效果

1、在 LED 数码管上面低 6 位会动态显示当前的 RTC 时间，高 2 位显示按下的 ADC 按键的键码：



2、按下 16 个 ADC 按键之一后：

- 在 LED 数码管的最高位会显示当前的按键键码

3、ADC 按键中的按键 F，是 RTC 模式设置键

- 上电后 RTC 为正常工作模式，此时数码管动态显示 RTC 时间
- 按一次按键 F，RTC 会切换为小时设置模式，此时 RTC 的小时值会 0.5 秒闪烁一次
- 再按一次按键 F，RTC 会切换为分钟设置模式，此时 RTC 的分钟值会 0.5 秒闪烁一次
- 再按一次按键 F，RTC 会切换为秒钟设置模式，此时 RTC 的秒钟值会 0.5 秒闪烁一次
- 再按一次按键 F，恢复为 RTC 正常工作模式

4、矩阵按键中的按键 D 和 E 分别是上调键和下调键

- 只有在 RTC 的时/分/秒设置模式时，上调键和下调键才有效

5、时钟设置模式时，长按按键 D 或 E，可实现快速设置功能。

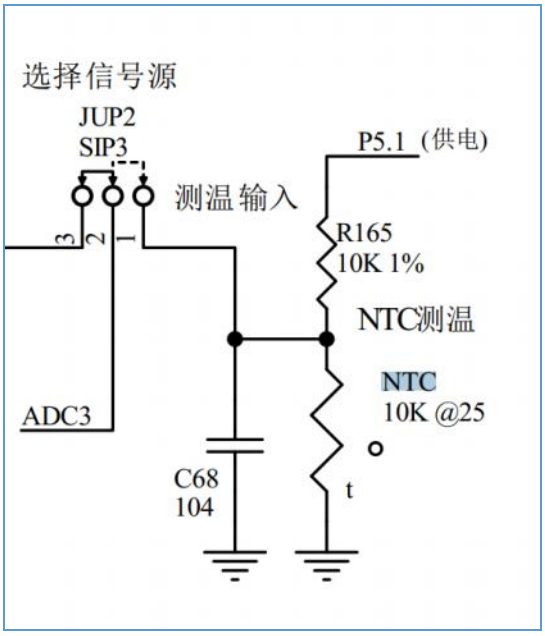
23.9 深大实验 -- ADC 中断方式检测 NTC 热敏电阻测温+数码管显示

23.9.1 实验介绍

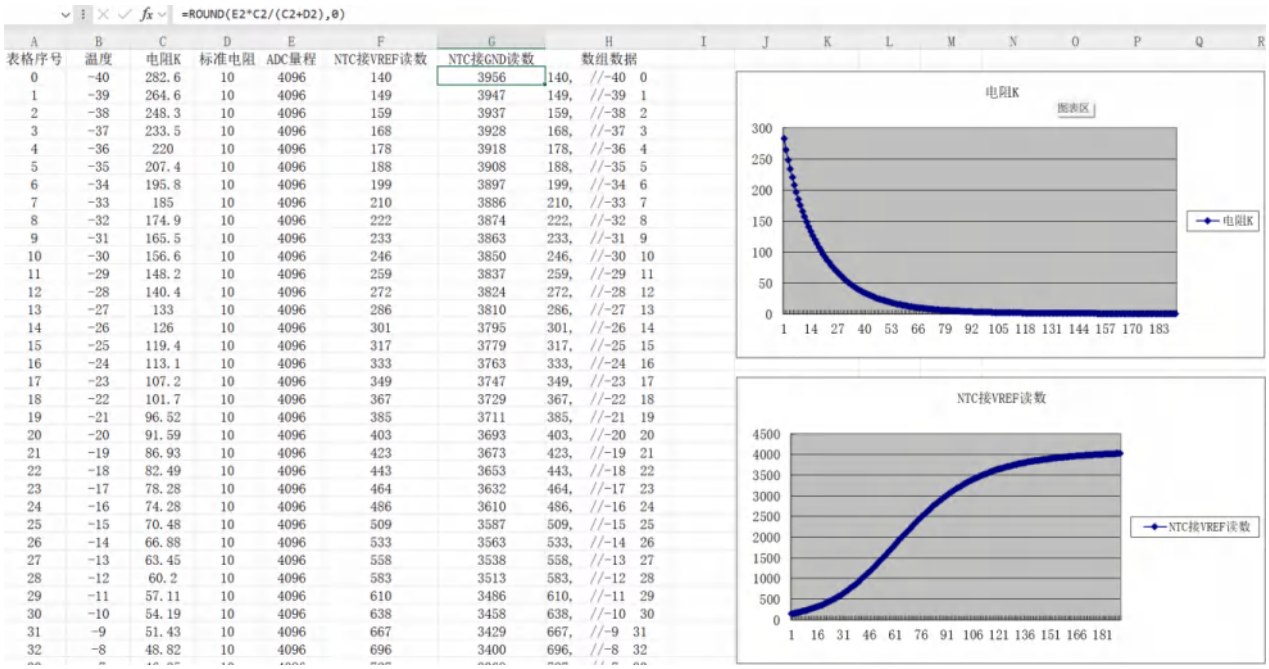
1、本项目主要熟悉如何管理多文件项目
2、熟悉 ADC 数据采集热敏电阻 NTC 方法
3、了解使用 74HC595 控制 LED 数码管

23.9.2 ADC 通过采集 NTC 热敏电阻的电压值进行测温

NTC thermistor，即 Negative Temperature Coefficient thermistor，即“负温度系数热敏电阻”，其电阻值随温度升高而减小，ADC 测温通过 ADC3 通道采集 NTC 热敏电阻的电压值来计算出温度值。如图所示，NTC 热敏电阻与 10K 精密电阻 R165 串联分压后的电压输入到 ADC3 脚位，而热敏电阻随温度变化，阻值会产生变化，从而导致分压电压跟着改变，于是通过 ADC 采集分压电压值，再根据热敏电阻的温度与阻值变化特性关系就能计算出温度值。



本实验程序通过查表法计算温度值，使用 Excel 软件生成温度与 ADC 读数对应表，放到程序 temp_table 数组里，供查表计算使用：



=ROUND(E2*C2/(C2+D2),0)

表格序号	温度	电阻K	标准电阻	ADC量程	NTC接VREF读数	NTC接GND读数	数组数据
0	-40	282.6	10	4096	140	3956	140, //-40 0
1	-39	264.6	10	4096	149	3947	149, //-39 1
2	-38	248.3	10	4096	159	3937	159, //-38 2
3	-37	233.5	10	4096	168	3928	168, //-37 3
4	-36	220	10	4096	178	3918	178, //-36 4
5	-35	207.4	10	4096	188	3908	188, //-35 5
6	-34	195.8	10	4096	199	3897	199, //-34 6
7	-33	185	10	4096	210	3886	210, //-33 7
8	-32	174.9	10	4096	222	3874	222, //-32 8
9	-31	165.5	10	4096	233	3863	233, //-31 9
10	-30	156.6	10	4096	246	3850	246, //-30 10
11	-29	148.2	10	4096	259	3837	259, //-29 11
12	-28	140.4	10	4096	272	3824	272, //-28 12
13	-27	133	10	4096	286	3810	286, //-27 13
14	-26	126	10	4096	301	3795	301, //-26 14
15	-25	119.4	10	4096	317	3779	317, //-25 15
16	-24	113.1	10	4096	333	3763	333, //-24 16
17	-23	107.2	10	4096	349	3747	349, //-23 17
18	-22	101.7	10	4096	367	3729	367, //-22 18
19	-21	96.52	10	4096	385	3711	385, //-21 19
20	-20	91.59	10	4096	403	3693	403, //-20 20
21	-19	86.93	10	4096	423	3673	423, //-19 21
22	-18	82.49	10	4096	443	3653	443, //-18 22
23	-17	78.28	10	4096	464	3632	464, //-17 23
24	-16	74.28	10	4096	486	3610	486, //-16 24
25	-15	70.48	10	4096	509	3587	509, //-15 25
26	-14	66.88	10	4096	533	3563	533, //-14 26
27	-13	63.45	10	4096	558	3538	558, //-13 27
28	-12	60.2	10	4096	583	3513	583, //-12 28
29	-11	57.11	10	4096	610	3486	610, //-11 29
30	-10	54.19	10	4096	638	3458	638, //-10 30
31	-9	51.43	10	4096	667	3429	667, //-9 31
32	-8	48.82	10	4096	696	3400	696, //-8 32

表格存放从 -40~120 度对应的 ADC 数据，分度值为 1 度，ADC 采集到的 NTC 数据与表格进行比较，从表格里找到最接近的数据所对应的温度就是当前 NTC 感应到的温度。

为了从表格里快速查找最接近的数据，我们采用二分法查表加快查找速度。例如 -40 度数据存放在数组 temp_table[0]，120 度数据存放在数组 temp_table[160]，取中间的 temp_table[80]数据跟 ADC 采样数据进行对比，如果 ADC 采样数据小于 temp_table[80]，就可以排除后面一半表格的数据，从前面一半数据里面再次进行对分比较，依此类推：

160 / 2 = 80

80 / 2 = 40

40 / 2 = 20

20 / 2 = 10

10 / 2 = 5

经过 5 次对分查表后，剩下的数据只有 temp_table[n]~temp_table[n+5] 这几个数据，然后进行逐个比较，如果 ADC 采样数据跟其中一个数据正好相等的话，那么当前温度值就是这个数据所对应的温度。如果 ADC 采样数据在两个表格数据之间，则进行线性插补。

例如 ADC 采样数据为 2063，经过比较后发现在表格数据的 2048(25 度)与 2093(26 度)之间，则计算其百分比值作为温度的小数值。为了方便计算，我们将最终数据放大 10 倍，这样就可以使用整数计算，提高计算速度：

1 度的差值：2093 - 2048 = 45

当前读数不满 1 度的值：2063 - 2048 = 15

计算小数部分，放大 10 倍：15*10 / 45 = 3　　//取整

从而得出当前温度值为 25.3 度。

为了避免浮点计算，本实验使用整数来处理，人为规定一个 0 点（即定点计算），并且温度值扩大 10 倍（即分辨率为 0.1 度），表格中 0 度对应的数据序号是 40，扩大 10 倍为 400，即 400 表示 0 度，大于 400 为正的温度，小于 400 为负的温度。

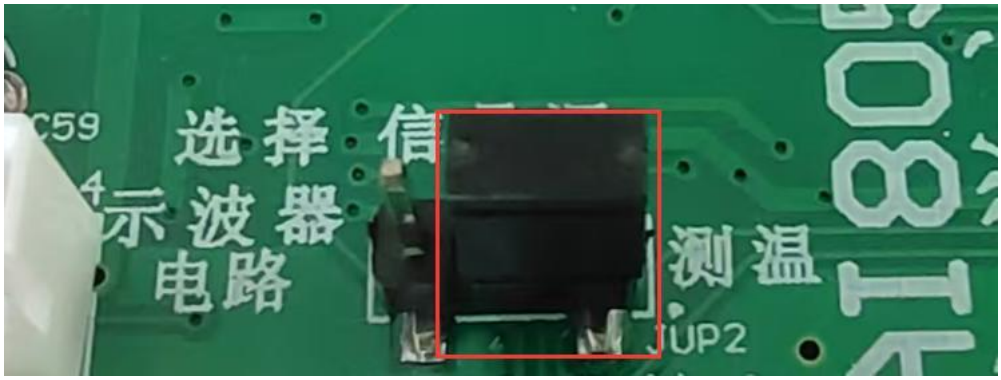
比如查表计算的结果是 653，则实际温度为（653-400）/10 = 25.3 度。

比如查表计算的结果是 300，则实际温度为（300-400）/10 = -10.0 度。

a) 操作说明和实验现象：

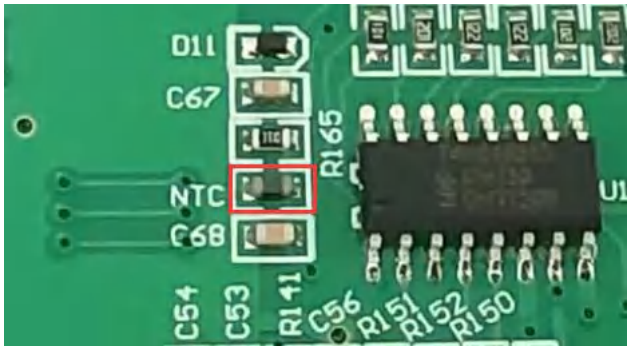
将本项目的 HEX 文件下载到 Ai8051U 的实验箱后

1、测量 NTC 温度，需要短接试验箱 JUP2 右侧两个跳线，才能连接 NTC 到芯片 ADC 脚：



2、在 LED 数码管上面低 4 位会动态显示当前的 NTC 测量的温度，高 4 位显示 NTC 的 ADC 值补码：

3、NTC 热敏电阻在实验箱背面，用手指触摸，或者向其吹气，可改变测量的温度值：



b) Keil 环境下多文件项目管理说明

当项目的功能比较复杂时，就需要在 Keil 中建立多文件项目，以方便分工合作、代码复用、模块化管理、增强可读性和可维护性。

- 比较好的建议是将项目功能模块化，不同模块的实现代码放在不同的.c 文件中。
- 一般建议是一个模块对应一个.c 程序文件和一个.h 头文件
- 模块的初始化函数以及相关的数据处理函数都在.c 文件中实现
- 与模块相关的全局变量也必须在.c 文件中进行定义，一定不能在.h 文件中定义变量
- 如果有其他模块需要使用本模块定义的变量或函数，则这些函数和变量都需要在.h 文件中声明。
- 特别提醒：在.h 文件中声明外部变量必须使用 **extern** 关键字，否则就变成变量定义了，这样会

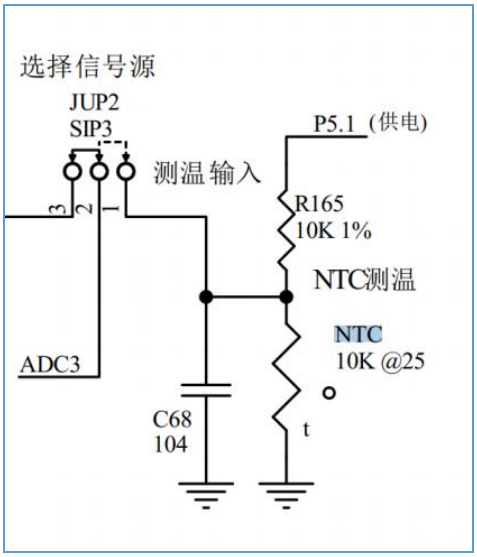
出现变量重复定义的错误

- 为防止头文件被多次包含而产生错误或者警告，在头文件中使用类似如下的条件编译组合语句，可避免在同一个.c 文件中对同一个.h 头文件进行多次包含：

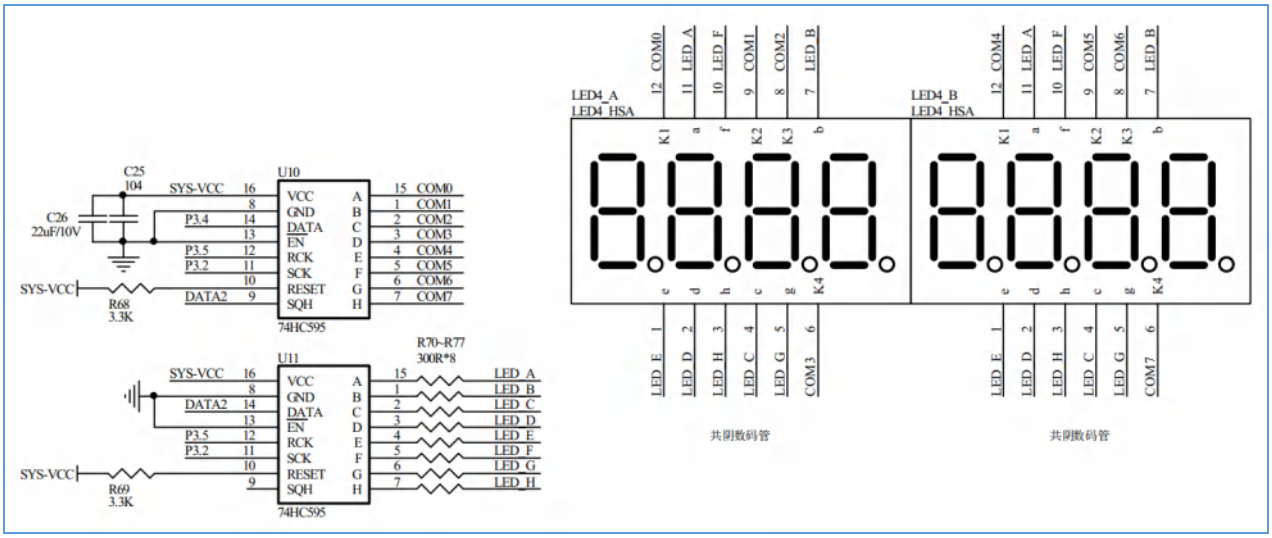
```
#ifndef XXXX
#define XXXX
...
#endif
```

23.9.3 原理图

a) NTC 温度检测原理图



b) 74HC595 驱动数码管原理图

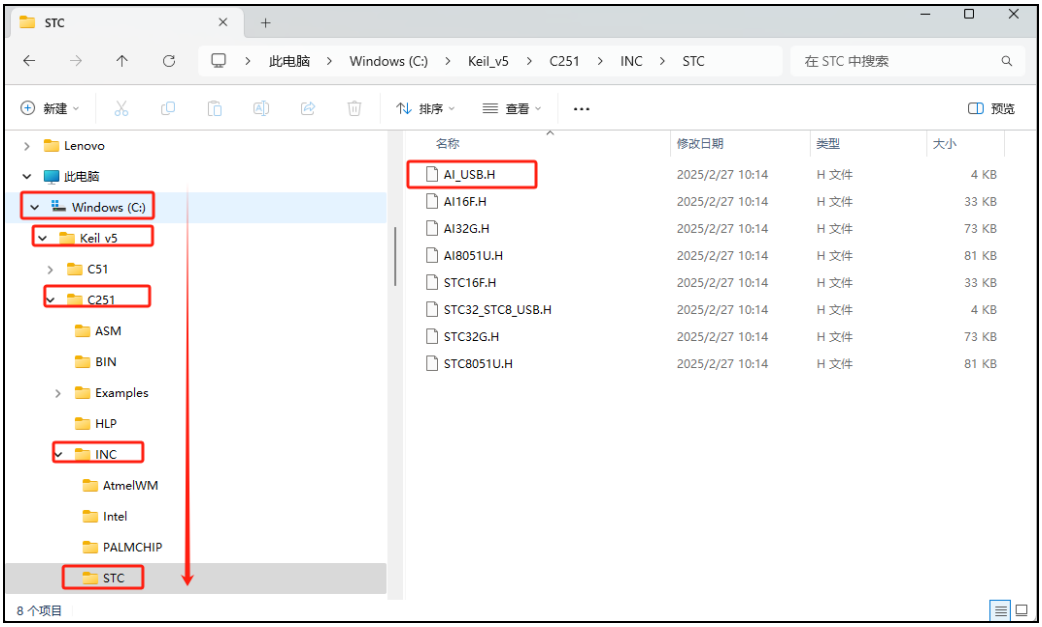


23.9.4 准备工作--添加头文件和库函数

a) 添加常用头文件 ai_usb.h

#include "ai_usb.h"　　//使用 usb 所需要包含的头文件。

这个头文件在使用 AIapp-ISP 软件添加型号和头文件到 Keil 中时就已经一并被添加到 Keil 中了。

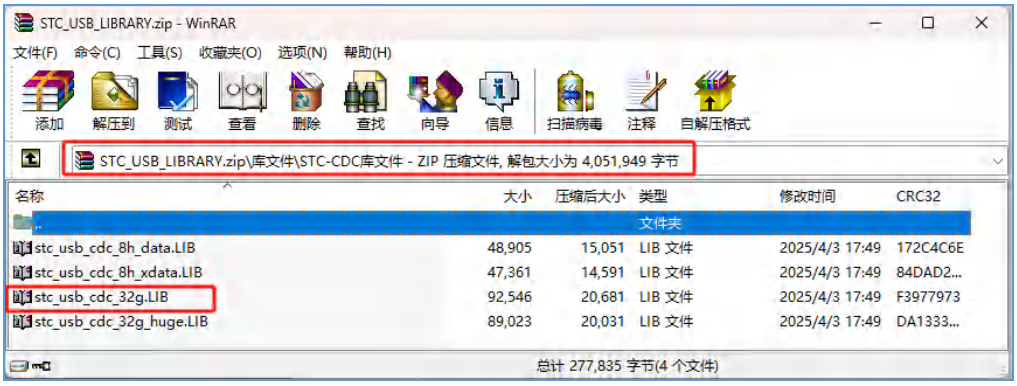


b) 添加 usb_cdc 库函数文件

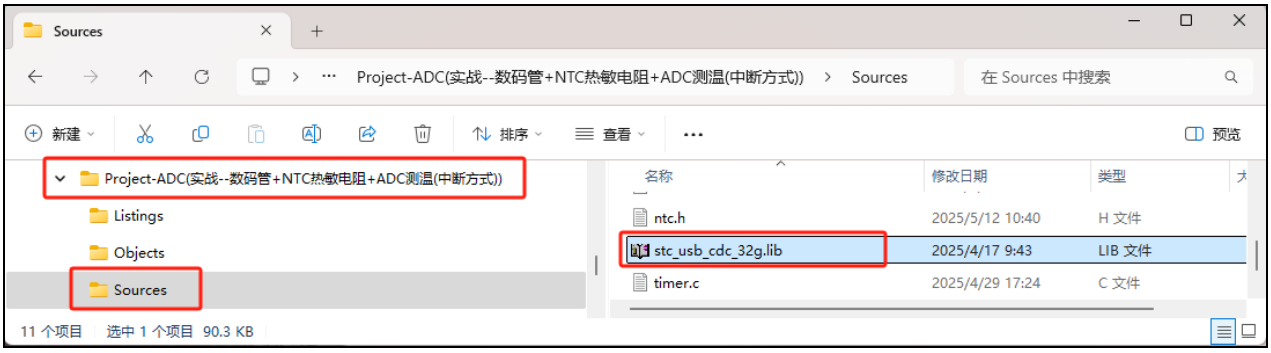
这部分可以到 STCAI 的官网：https://www.stcai.com/khs 的 USB 库文件中进行下载

下载完成后，打开压缩包“STC_USB_LIBRARY.zip”，

在‘库文件\查询模式 STC-CDC 库文件’中，复制出来 stc_usb_cdc_32g.LIB

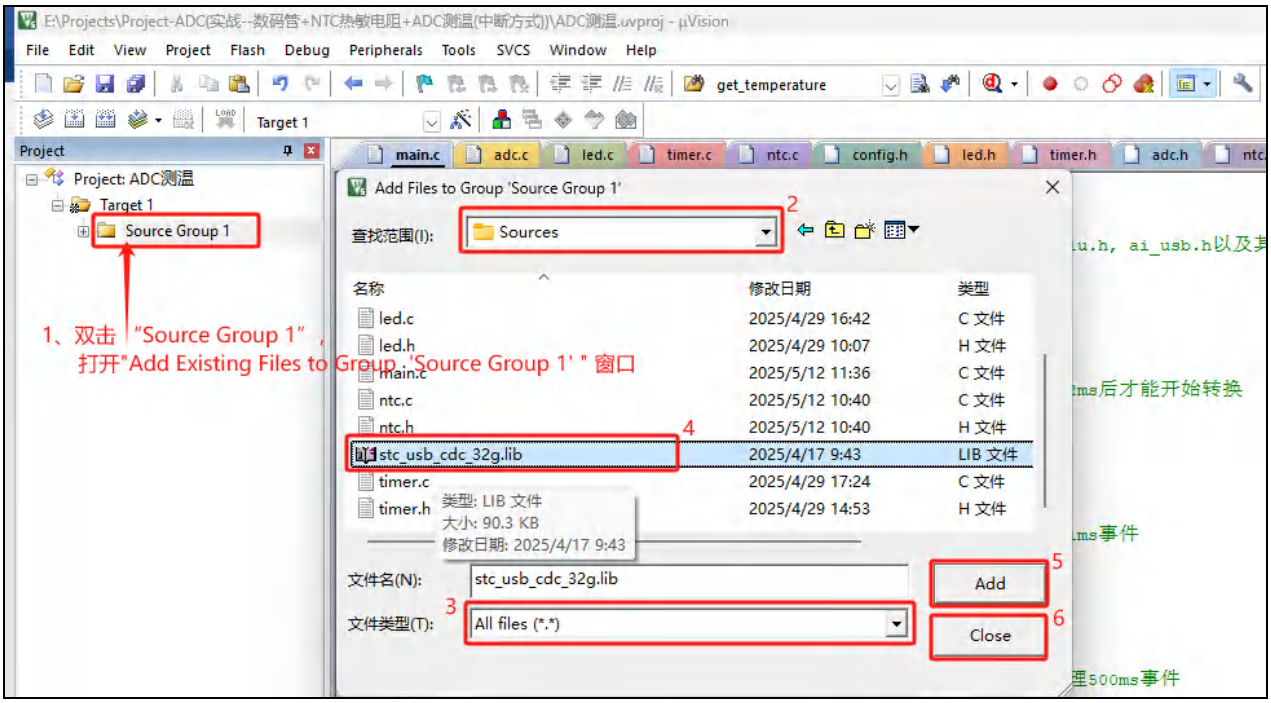


将库函数“stc_usb_cdc_32g.LIB”复制到工程所在的文件夹 Project-ADC(实战--数码管+NTC 热敏电阻+ADC 测温(查询方式))>Sources 目录下：

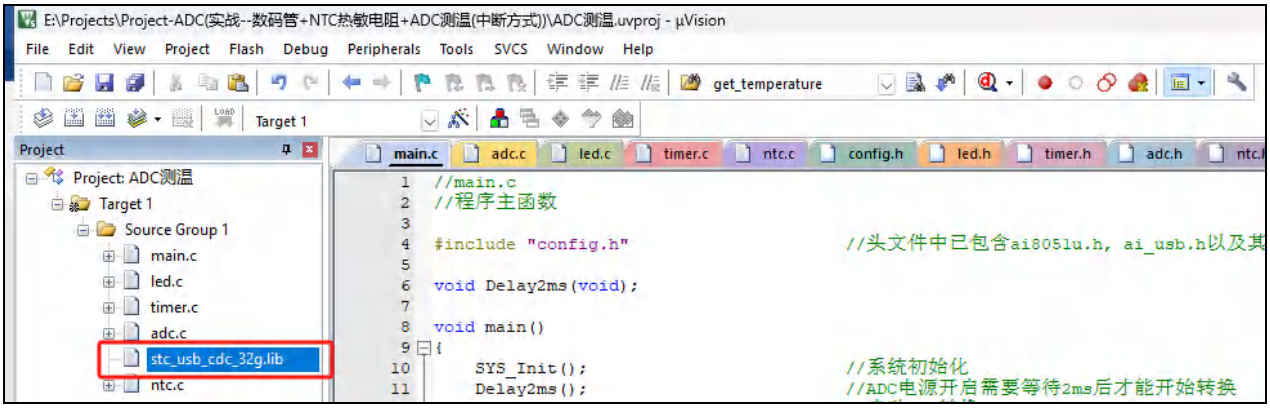


将库函数 stc_usb_cdc_32g.LIB 添加进入此工程项目中

在 Keil 中，双击 Source Group，在弹出的 Add Files to Group 'Source Group 1 窗口中，将库函数 stc_usb_cdc_32g.LIB 添加进入此工程，就可以使用 USB-CDC 功能。



这样，Keil 界面变化如下图：



23.9.5 实验程序代码

a) main.c -- 程序主函数

```
//main.c
//程序主函数

#include "config.h"       //头文件中已包含 ai8051u.h, ai_usb.h 以及其他头文件

void Delay2ms(void);

void main()
{
    SYS_Init();         //系统初始化
    Delay2ms();         //ADC 电源开启需要等待 2ms 后才能开始转换
    Get_ADC(3);         //启动 ADC 转换

    while (1)
    {
        if (f1ms)           //判断 1ms 标志位
        {
            f1ms = 0;       //清除 1ms 标志，并处理 1ms 事件
            ScanLed();      //LED 数码管扫描
        }

        if (f500ms)         //判断 500ms 标志位
        {
            f500ms = 0;     //清除 500ms 标志，并处理 500ms 事件
            NTC_Show(ADC_Data);     //在 LED 数码管上显示 NTC 温度
            Get_ADC(3);  //启动 ADC 转换
        }
    }
}

void SYS_Init(void)     //系统初始化函数
{
    EAXFR = 1;          //使能访问扩展 XFR
    WTST = 0x00;        //设置最快速度访问程序代码
    CKCON = 0x00;       //设置最快速度访问内部 XDATA

    P0M0 = 0x00; P0M1 = 0x00;       //初始化 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x08;       //初始化 P1 口为准双向口模式, P1.3 高阻输入
    P2M0 = 0x00; P2M1 = 0x00;       //初始化 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x00;       //初始化 P3 口为准双向口模式
    P4M0 = 0x00; P4M1 = 0x00;       //初始化 P4 口为准双向口模式
```

```
    P5M0 = 0x02; P5M1 = 0x00;       //设置 P5.1 口为强推挽, 其他设置为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;       //初始化 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;       //初始化 P7 口为准双向口模式

    usb_init();             //USB 库函数初始化
    Led_Init();             //LED 数码管初始化
    Timer0_Init();          //定时器 0 初始化,用于产生 1ms 和 500ms 的时间基准
    ADC_Init();             //ADC 初始化

    P51 = 1;                //给 NTC 供电
    EA = 1;                 //总中断允许位打开
}

void Delay2ms(void)     //@40.000MHz
{
    unsigned long edata i;

    _nop_();
    _nop_();
    _nop_();
    i = 19998UL;
    while (i) i--;
}
```

b) led.c -- 扫描 LED 数码管

```
//led.c
//使用 74HC595 动态扫描 LED 数码管
//由于 LED 数码管是由 8 个 SEG 和 8 个 COM 组成的矩阵 LED
//所以 LED 数码管也需要扫描显示

#include "config.h"             //头文件中已包含 ai8051u.h, ai_usb.h 以及其他头文件

sbit    DAT = P3^4;         //74HC595 串行数据输入脚
sbit    SCK = P3^2;         //74HC595 串行数据移入时钟脚,上升沿有效
sbit    RCK = P3^5;         //74HC595 并行数据输出时钟脚,上升沿有效

#define LED_OFF     0x00
//定义关闭 LED 数码管的段码数据,LED 数码管为共阴极,所以 SEG 线全部输出 0 即可关闭一个 8 段数码管
#define LED_        0x11        //定义 LED 数码管中杠的段码数据
#define LED_POINT   0x80        //定义 LED 数码管小数点的段码数据

BYTE    ShowIndex;          //LED 数码管扫描显示索引(8 个数码管轮流扫描显示)
BYTE    LedBuf[8];          //8 个 LED 数码管显示数据缓冲区
```

```
BYTE    code LED_PATTERN[10] =      //定义 LED 数码管显示数据 0~9 的段码数据
{
    0x3F,    //0                    //此表为共阴极数码管的段码数据
    0x06,    //1
    0x5B,    //2
    0x4F,    //3
    0x66,    //4
    0x6D,    //5
    0x7D,    //6
    0x07,    //7
    0x7F,    //8
    0x6F,    //9
};

void Shift_74HC595(BYTE dat)            //串行移入一个字节的数据到 74HC595
{
    BYTE i;

    for (i = 0; i < 8; i++)     //移动次数控制(移动 8Bit)
    {
        dat <<= 1;              //将数据的最高位移出到 CY
        DAT = CY;               //将移出的最高位数据输出到 74HC595 的数据输入口
        SCK = 0;                //74HC595 的串行数据移入时钟脚输出低电平
        _nop_();                //等待 2 个时钟
        _nop_();
        SCK = 1;                //74HC595 的串行数据移入时钟脚输出高电平,上升沿移入数据
    }
}

void Output_74HC595(void)       //74HC595 并行数据输出控制
{
    RCK = 0;            //74HC595 的并行数据输出时钟脚输出低电平
    _nop_();            //等待 2 个时钟
    _nop_();
    RCK = 1;            //74HC595 的并行数据输出时钟脚输出高电平,上升沿输出数据
}

void Led_Init(void)     //LED 数码管初始化程序
{
    DAT = 0;            //初始化 74HC595 时钟线和数据线
    RCK = 0;
    SCK = 0;

    LedBuf[0] = LED_OFF;        //初始化 LED 数码管显示数据缓冲区,上电关闭显示
    LedBuf[1] = LED_OFF;
    LedBuf[2] = LED_OFF;
```

```
    LedBuf[3] = LED_OFF;
    LedBuf[4] = LED_OFF;
    LedBuf[5] = LED_OFF;
    LedBuf[6] = LED_OFF;
    LedBuf[7] = LED_OFF;

    ShowIndex = 0;                  //初始化扫描显示索引
}

void ScanLed(void)                  //LED 数码管扫描显示程序,  (需每 1ms 执行一次!!!)
{
    Shift_74HC595(LedBuf[ShowIndex]);       //将当前扫描到的 LED 数码管的段码移入到 74HC595
    Shift_74HC595(~(1 << ShowIndex));       //将当前扫描到的 LED 数码管的 COM 数据移入到 74HC595
    Output_74HC595();               //将 74HC595 中的段码数据和 COM 数据输出到 LED 数码管的管脚

    if (++ShowIndex >= 8)           //扫描索引值递增
        ShowIndex = 0;              //扫描完一轮后索引值归零
}

void SetLed (BYTE nIndex,           //设置 LED 显示函数,nIndex:设置目标数码管,0~7 分别对应 8 个数码
             BYTE nNum,             //nNum:设置显示的数据,有效值 0~9,其他值则关闭显示
             BOOL bShowPoint        //bShowPoint:设置是否显示小数点
            )
{
    if (nIndex < 8)                 //目标数码管的有效值为 0~7
    {
        if (nNum < 10)              //判断显示的值是否为 0~9
            LedBuf[nIndex] = LED_PATTERN[nNum];
                                    //将相应数据的段码赋值到 LED 数码管显示数据缓冲区
        else if (nNum == 11)        //判断显示的值是否为"-"
            LedBuf[nIndex] = LED_;          //显示"-"
        else
            LedBuf[nIndex] = LED_OFF;       //如果是其他值则关闭显示

        if (bShowPoint)             //判断是否需要显示小数点
            LedBuf[nIndex] |= LED_POINT;  //如果需要显示,则增加小数点的段码数据
    }
}
```

c) timer.c -- 定时器初始化以及定时器中断服务程序

```
//timer.c
//定时器初始化以及定时器中断服务程序
//定时器 0:1ms 定时,产生 1ms 和 500ms 的标志位
```

```
#include "config.h"       //头文件中已包含 ai8051u.h, ai_usb.h 以及其他头文件

bit       f1ms;           //1ms 标志位
bit       f500ms;         //500ms 标志位
int       cnt500;         //500 次计数器,用于产生 500ms 的时间

void Timer0_Init(void)        //1 毫秒@40.000MHz
{
    AUXR |= 0x80;             //定时器 0 时钟 1T 模式
    TMOD &= 0xF0;             //设置定时器 0 为模式 0
    TL0 = 0xC0;               //设置定时器 0 初始值
    TH0 = 0x63;               //设置定时器 0 初始值
    TF0 = 0;                  //清除 TF0 标志
    TR0 = 1;                  //定时器 0 开始计时
    ET0 = 1;                  //使能定时器 0 中断

    cnt500 = 0;               //初始化 500 次计数器
    f1ms = 0;                 //初始化 1m 标志位
    f500ms = 0;               //初始化 500ms 标志位
}

void Timer0_Isr(void) interrupt 1        //定时器 0 中断服务程序
{
    f1ms = 1;                 //设置 1ms 标志位

    if (cnt500++ >= 500)      //判断 500 次计数是否已满
    {
        cnt500 = 0;           //清除计数值,开始开始计数
        f500ms = 1;           //设置 500ms 标志位
    }
}
```

d) adc.c -- ADC 初始化和 ADC 中断方式数据采集程序

```
//adc.c
//ADC 程序
//包括 ADC 初始化和 ADC 数据采集

#include "config.h"           //头文件中已包含 ai8051u.h, ai_usb.h 以及其他头文件

u16 ADC_Data;

void ADC_Init()               //ADC 初始化函数
{
    ADCTIM = 0x3f;            //设置 ADC 内部时序，ADC 采样时间建议设最大值
```

```
    ADCCFG = 0x2f;          //设置 ADC 时钟为系统时钟/2/16
    ADC_CONTR = 0x80;  //使能 ADC 模块
    EADC = 1;                   //使能 ADC 中断
}

void Get_ADC(u8 channel)   //channel = 0~15
{
    ADC_CONTR = (ADC_CONTR & 0xf0) | channel;          //设置 ADC 转换通道
    ADC_START = 1;          //启动 ADC 转换
}

void ADC_Isr() interrupt 5
{
    ADC_FLAG = 0;           //清中断标志
    ADC_Data = ((u16)ADC_RES << 8) | ADC_RESL;          //读取 ADC 结果
}
```

e) ntc.c -- ADC 中断方式检测 NTC 热敏电阻测温程序

```
//ntc.c
//ADC 采集热敏电阻 NTC 程序

#include "config.h" //头文件中已包含 ai8051u.h, ai_usb.h 以及其他头文件

//    MF52E 10K at 25, B = 3950, ADC = 12 bits
u16 code temp_table[]={
                                140,        //;-40      0
                                149,        //;-39      1
                                159,        //;-38      2
                                168,        //;-37      3
                                178,        //;-36      4
                                188,        //;-35      5
                                199,        //;-34      6
                                210,        //;-33      7
                                222,        //;-32      8
                                233,        //;-31      9
                                246,        //;-30      10
                                259,        //;-29      11
                                272,        //;-28      12
                                286,        //;-27      13
                                301,        //;-26      14
                                317,        //;-25      15
                                333,        //;-24      16
                                349,        //;-23      17
                                367,        //;-22      18
```

```
385,    //;-21    19
403,    //;-20    20
423,    //;-19    21
443,    //;-18    22
464,    //;-17    23
486,    //;-16    24
509,    //;-15    25
533,    //;-14    26
558,    //;-13    27
583,    //;-12    28
610,    //;-11    29
638,    //;-10    30
667,    //;-9     31
696,    //;-8     32
727,    //;-7     33
758,    //;-6     34
791,    //;-5     35
824,    //;-4     36
858,    //;-3     37
893,    //;-2     38
929,    //;-1     39
965,    //;0      40
1003,   //;1      41
1041,   //;2      42
1080,   //;3      43
1119,   //;4      44
1160,   //;5      45
1201,   //;6      46
1243,   //;7      47
1285,   //;8      48
1328,   //;94     9
1371,   //;10     50
1414,   //;11     51
1459,   //;12     52
1503,   //;13     53
1548,   //;14     54
1593,   //;15     55
1638,   //;16     56
1684,   //;17     57
1730,   //;18     58
1775,   //;19     59
1821,   //;20     60
1867,   //;21     61
1912,   //;22     62
1958,   //;23     63
2003,   //;24     64
```

```
2048,   //;25   65
2093,   //;26   66
2137,   //;27   67
2182,   //;28   68
2225,   //;29   69
2269,   //;30   70
2312,   //;31   71
2354,   //;32   72
2397,   //;33   73
2438,   //;34   74
2479,   //;35   75
2519,   //;36   76
2559,   //;37   77
2598,   //;38   78
2637,   //;39   79
2675,   //;40   80
2712,   //;41   81
2748,   //;42   82
2784,   //;43   83
2819,   //;44   84
2853,   //;45   85
2887,   //;46   86
2920,   //;47   87
2952,   //;48   88
2984,   //;49   89
3014,   //;50   90
3044,   //;51   91
3073,   //;52   92
3102,   //;53   93
3130,   //;54   94
3157,   //;55   95
3183,   //;56   96
3209,   //;57   97
3234,   //;58   98
3259,   //;59   99
3283,   //;60   100
3306,   //;61   101
3328,   //;62   102
3351,   //;63   103
3372,   //;64   104
3393,   //;65   105
3413,   //;66   106
3432,   //;67   107
3452,   //;68   108
3470,   //;69   109
3488,   //;70   110
```

```
3506,   //;71    111
3523,   //;72    112
3539,   //;73    113
3555,   //;74    114
3571,   //;75    115
3586,   //;76    116
3601,   //;77    117
3615,   //;78    118
3628,   //;79    119
3642,   //;80    120
3655,   //;81    121
3667,   //;82    122
3679,   //;83    123
3691,   //;84    124
3702,   //;85    125
3714,   //;86    126
3724,   //;87    127
3735,   //;88    128
3745,   //;89    129
3754,   //;90    130
3764,   //;91    131
3773,   //;92    132
3782,   //;93    133
3791,   //;94    134
3799,   //;95    135
3807,   //;96    136
3815,   //;97    137
3822,   //;98    138
3830,   //;99    139
3837,   //;100   140
3844,   //;101   141
3850,   //;102   142
3857,   //;103   143
3863,   //;104   144
3869,   //;105   145
3875,   //;106   146
3881,   //;107   147
3887,   //;108   148
3892,   //;109   149
3897,   //;110   150
3902,   //;111   151
3907,   //;112   152
3912,   //;113   153
3917,   //;114   154
3921,   //;115   155
3926,   //;116   156
```

```
                    3930,     //;117     157
                    3934,     //;118     158
                    3938,     //;119     159
                    3942      //;120     160
                    };

/*********************    计算温度  ***********************************************/
//  计算结果: 0 对应-40.0 度, 400 对应 0 度, 650 对应 25.0 度,  最大 1600 对应 120.0 度.
//  为了通用, ADC 输入为 12bit 的 ADC 值.
/*********************************************/

#define    D_SCALE 10        //结果放大倍数, 放大 10 倍就是保留一位小数
u16 get_temperature(u16 adc)
{
    u16       code *p;
    u16       i;
    u8        j,k,min,max;

    adc = 4096 - adc;         //Rt 接地
    p = temp_table;           //指针指向温度表格

    if(adc < p[0])        return (0xfffe);
    if(adc > p[160])      return (0xffff);

    min = 0;                  //-40 度
    max = 160;                //120 度

    for(j=0; j<5; j++)        //对分查表 5 次，缩小查表范围
    {
        k = min / 2 + max / 2;
        if(adc <= p[k])       max = k;
        else                  min = k;
    }

    if(adc == p[min])         i = min * D_SCALE;
                              //如果 ADC 值正好等于最小指针对应值，则取其序号作为温度参数并放大 10 倍
    else if(adc == p[max])    i = max * D_SCALE;
                              //如果 ADC 值正好等于最大指针对应值，则取其序号作为温度参数并放大 10 倍
    else                      // min < temp < max
    {
        while(min <= max)         //剩下部分逐个查表对比
        {
            min++;
            if(adc == p[min])    {i = min * D_SCALE; break;}
                                  //如果 ADC 值正好等于  p[min] ，则取其序号作为温度参数并放大 10 倍
            else if(adc < p[min])      //如果 ADC 值在  p[min-1]  与  p[min]  之间，则增加补偿
```

```
                {
                    min--;
                    i = p[min];             //min
                    j = (adc - i) * D_SCALE / (p[min+1] - i);
                    i = min;
                    i *= D_SCALE;
                    i += j;
                    break;
                }
            }
        }
        return i;
}

void NTC_Show(u16 adc)     //NTC 温度计算显示程序
{
        u16 temp;

        if(temp > 4095) return;  //判断 ADC 值是否有效，无效则直接退出

        //在数码管左侧显示 ADC 采样数据
        if((adc / 1000) == 0)
        {
            SetLed(0, 12, 0);    //千位为 0，在数码 0 上显示空白
        }
        else
        {
            SetLed(0, (BYTE)(adc / 1000), 0);       //在数码 0 上显示 ADC 的千位
        }
        SetLed(1, (BYTE)((adc % 1000) / 100), 0);       //在数码 1 上显示 ADC 的百位
        SetLed(2, (BYTE)((adc % 100) / 10), 0);         //在数码 2 上显示 ADC 的十位
        SetLed(3, (BYTE)(adc % 10), 0);                 //在数码 3 上显示 ADC 的个位

        temp = get_temperature(adc);            //计算温度值

        if(temp >= 400)     F0 = 0, temp -= 400;        //温度  >= 0 度
        else                F0 = 1, temp = 400 - temp;  //温度  <   0 度

        if(F0)
        {
            SetLed(4, 11, 0);           //温度为零下，在数码 4 上显示"-"
        }
        else if((temp / 1000) == 0)
        {
            SetLed(4, 12, 0);           //千位为 0，在数码 4 上显示空白
        }
```

```
    else
    {
        SetLed(4, (BYTE)(temp / 1000), 0);          //在数码 4 上显示 ADC 的千位
    }
    SetLed(5, (BYTE)((temp % 1000) / 100), 0);      //在数码 5 上显示 ADC 的百位
    SetLed(6, (BYTE)((temp % 100) / 10), 1);        //在数码 6 上显示 ADC 的十位，并显示小数点
    SetLed(7, (BYTE)(temp % 10), 0);                //在数码 7 上显示 ADC 的个位
}
```

f) config.h -- 项目配置的头文件

```
#ifndef __CONFIG_H__            //防止头文件被重复包含
#define __CONFIG_H__

#define   HIRC              40000000UL
#define   FOSC              40000000UL
#define   SYSCLK            FOSC
#define   MAIN_Fosc         FOSC

#include <ai8051u.h>            //包含外部头文件
#include <stdio.h>
#include <intrins.h>

#include "def.h"                //包含项目头文件
#include "ai_usb.h"
#include "timer.h"
#include "led.h"
#include "ntc.h"
#include "adc.h"

void SYS_Init(void);            //函数声明

#endif
```

g) led.h -- 项目配置文件的头文件

```
#ifndef __LED_H__           //防止头文件被重复包含
#define __LED_H__

extern BYTE LedBuf[8];

void Led_Init(void);        //函数声明
void ScanLed(void);         //函数声明
void SetLed(BYTE nIndex, BYTE nNum, BOOL bShowPoint);
```

```
#endif
```

h) timer.h -- 项目配置文件的头文件

```
#ifndef __TIMER_H__        //防止头文件被重复包含
#define __TIMER_H__

void Timer0_Init(void);    //函数声明
void Timer1_Init(void);    //函数声明

extern bit f1ms;           //外部变量声明
extern bit f10ms;          //外部变量声明
extern bit f500ms;         //外部变量声明

#endif
```

i) adc.h -- 项目配置文件的头文件

```
#ifndef __ADC_H__          //防止头文件被重复包含
#define __ADC_H__

extern u16 ADC_Data;

void ADC_Init();           //函数声明
void Get_ADC(u8 channel);  //函数声明
#endif
```

j) ntc.h -- 项目配置文件的头文件

```
#ifndef __NTC_H__          //防止头文件被重复包含
#define __NTC_H__

void NTC_Show(u16 adc);            //函数声明

#endif
```

23.9.6 程序下载

录入代码，保存，编译。连接好 Ai8051U 实验箱，运行 AIapp-ISP 软件系统，按步骤打开并下载程序“ADC 测温.hex”，如下图：

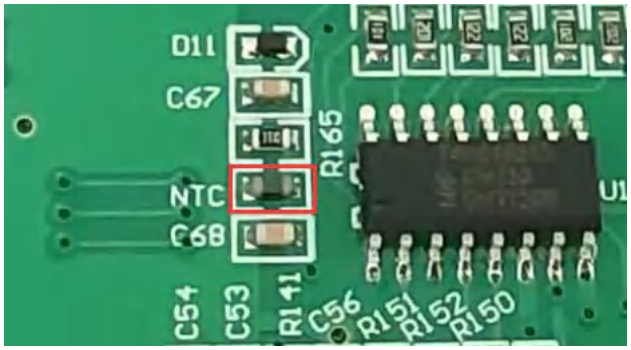
下载/烧录成功！

23.9.7 实验现象

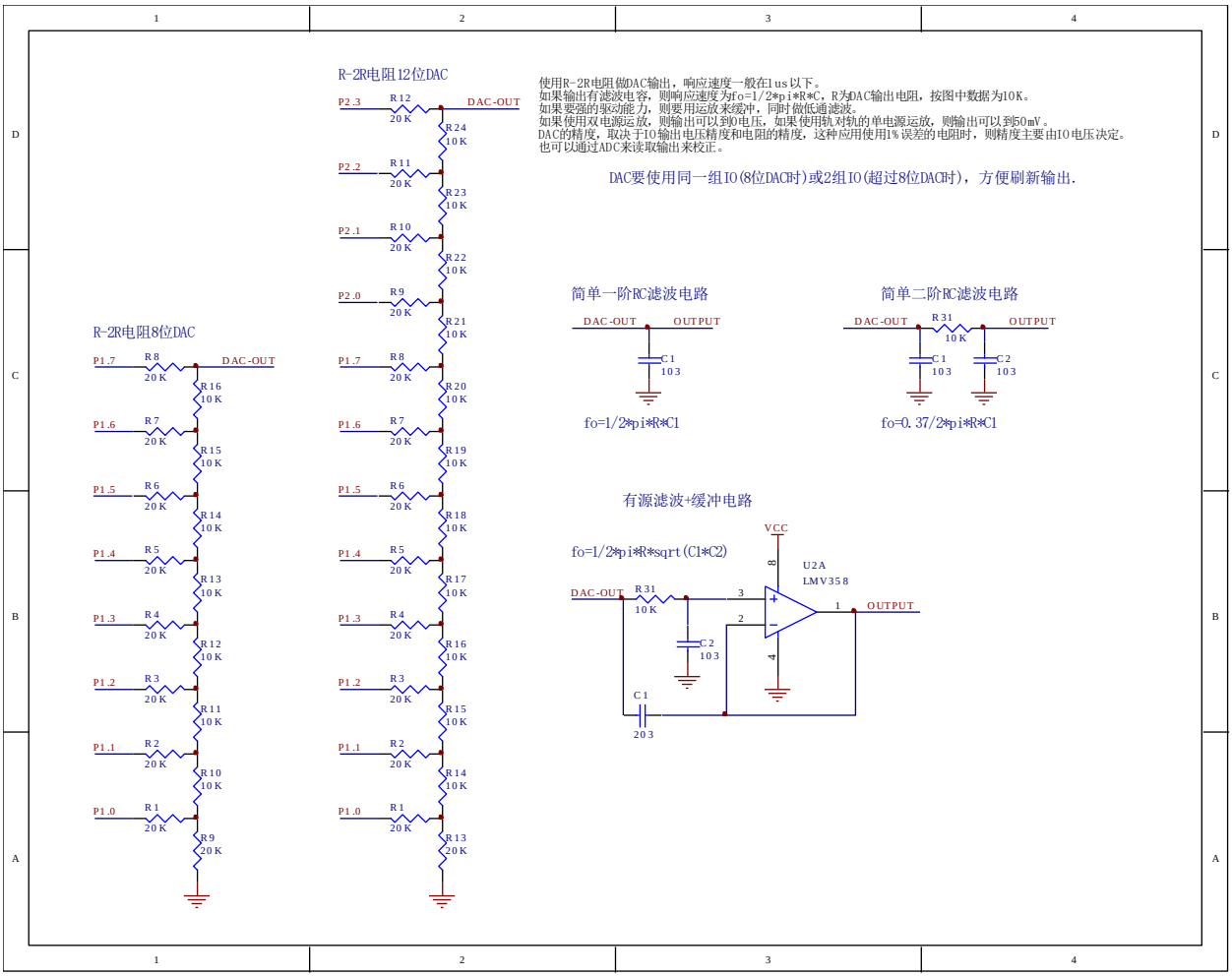
现在，观察 Ai8051U 实验箱：

1、在 LED 数码管上面低 4 位会动态显示当前的 NTC 测量的温度，高 4 位显示 NTC 的 ADC 值补码：

2、NTC 热敏电阻在实验箱背面，用手指触摸，或者向其吹气，可改变测量的温度值：

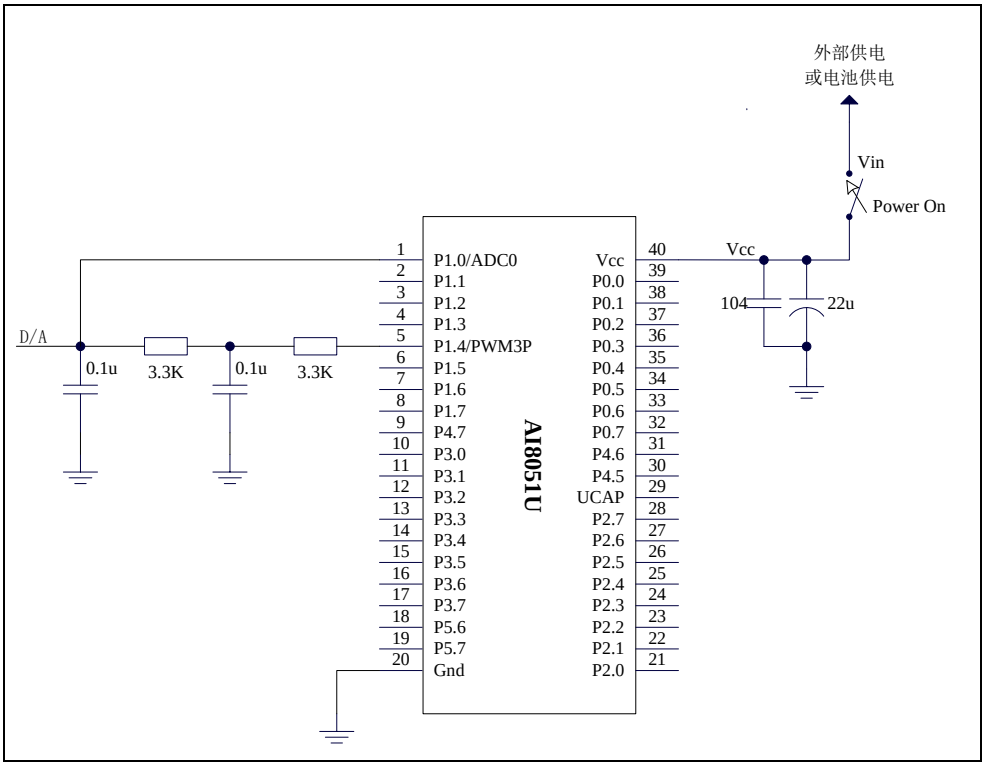


23.10　使用 I/O 和 R-2R 电阻分压实现 DAC 的经典线路图



23.11　利用 PWM 实现 16 位 DAC 的参考线路图

Ai8051U 系列单片机的高级 PWM 定时器可输出 16 位的 PWM 波形，再经过两级低通滤波即可产生 16 位的 DAC 信号，通过调节 PWM 波形的高电平占空比即可实现 DAC 信号的改变。应用线路图如下图所示，输出的 DAC 信号可输入到 MCU 的 ADC 进行反馈测量。



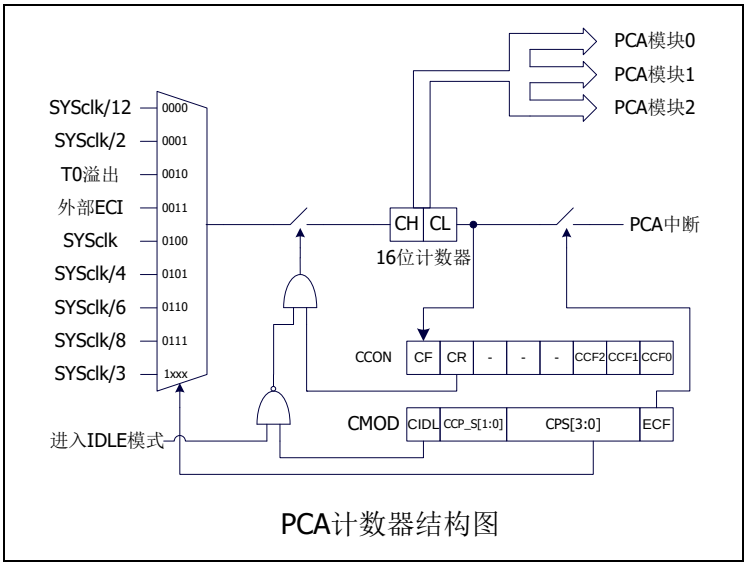
24 PCA/CCP/PWM 应用

产品线	PCA
Ai8051U 系列	●

Ai8051U 系列单片机内部集成了 3 组可编程计数器阵列（PCA/CCP/PWM）模块，可用于软件定时器、外部脉冲捕获、高速脉冲输出和 PWM 脉宽调制输出。

特别提示：如果使能 Ai8051U 系列的 PCA 中断时，PCA 的中断优先级固定为最高优先级，软件不能设置 PPCA（IP.7）和 PPCAH（IPH.7）来调整 PCA 的中断优先级，否则会导致无法产生 PCA 中断。

PCA 内部含有一个特殊的 16 位计数器，3 组 PCA 模块均与之相连接。PCA 计数器的结构图如下：



PCA计数器结构图

24.1 PCA 功能脚切换

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMOD	7EFD67H	CIDL	CCP_S[1:0]		CPS[3:0]				ECF

CCP_S[1:0]：PCA 功能脚选择位

CCP_S[1:0]	ECI	CCP0	CCP1	CCP2
00	P1.2	P1.3	P1.4	P1.1
01	P4.1	P4.2	P4.3	P4.4
10	P2.3	P2.0	P2.1	P2.2
11	-	-	-	-

24.2 PCA 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CCON	PCA 控制寄存器	7EFD64H	CF	CR	-	-	-	CCF2	CCF1	CCF0	00xx,x000
CL	PCA 计数器低字节	7EFD65H									0000,0000
CH	PCA 计数器高字节	7EFD66H									0000,0000
CMOD	PCA 模式寄存器	7EFD67H	CIDL	CCP_S[1:0]		CPS[3:0]				ECF	0xxx,0000
CCAPM0	PCA 模块 0 模式控制寄存器	7EFD58H	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0	x000,0000
CCAP0L	PCA 模块 0 低字节	7EFD59H									0000,0000
CCAP0H	PCA 模块 0 高字节	7EFD5AH									0000,0000
PCA_PWM0	PCA0 的 PWM 模式寄存器	7EFD5BH	EBS0[1:0]		XCCAP0H[1:0]		XCCAP0L[1:0]		EPC0H	EPC0L	0000,0000
CCAPM1	PCA 模块 1 模式控制寄存器	7EFD5CH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1	x000,0000
CCAP1L	PCA 模块 1 低字节	7EFD5DH									0000,0000
CCAP1H	PCA 模块 1 高字节	7EFD5EH									0000,0000
PCA_PWM1	PCA1 的 PWM 模式寄存器	7EFD5FH	EBS1[1:0]		XCCAP1H[1:0]		XCCAP1L[1:0]		EPC1H	EPC1L	0000,0000
CCAPM2	PCA 模块 2 模式控制寄存器	7EFD50H	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2	x000,0000
CCAP2L	PCA 模块 2 低字节	7EFD51H									0000,0000
CCAP2H	PCA 模块 2 高字节	7EFD52H									0000,0000
PCA_PWM2	PCA2 的 PWM 模式寄存器	7EFD53H	EBS2[1:0]		XCCAP2H[1:0]		XCCAP2L[1:0]		EPC2H	EPC2L	0000,0000

24.2.1 PCA 控制寄存器（CCON）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CCON	7EFD64H	CF	CR	-	-	-	CCF2	CCF1	CCF0

CF：PCA 计数器溢出中断标志。当 PCA 的 16 位计数器计数发生溢出时，硬件自动将此位置 1，并向 CPU 提出中断请求。此标志位需要软件清零。

CR：PCA 计数器允许控制位。

0：停止 PCA 计数

1：启动 PCA 计数

CCFn（n=0,1,2）：PCA 模块中断标志。当 PCA 模块发生匹配或者捕获时，硬件自动将此位置 1，并向 CPU 提出中断请求。此标志位需要软件清零。

24.2.2 PCA 模式寄存器（CMOD）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMOD	7EFD67H	CIDL	CCP_S[1:0]		CPS[3:0]				ECF

CIDL：空闲模式下是否停止 PCA 计数。

0：空闲模式下 PCA 继续计数

1：空闲模式下 PCA 停止计数

CCP_S[1:0]：PCA 功能脚选择位

CCP_S[1:0]	ECI	CCP0	CCP1	CCP2
00	P1.2	P1.3	P1.4	P1.1
01	P4.1	P4.2	P4.3	P4.4
10	P2.3	P2.0	P2.1	P2.2
11	-	-	-	-

CPS[3:0]：PCA 计数脉冲源选择位

CPS[3:0]	PCA 的输入时钟源	注意事项
0000	系统时钟/12	
0001	系统时钟/2	
0010	定时器 0 的溢出脉冲	
0011	ECI 脚的外部输入时钟	**外部时钟频率不能高于系统频率的 1/2**
0100	系统时钟	
0101	系统时钟/4	
0110	系统时钟/6	
0111	系统时钟/8	
1xxx	系统时钟/3	

ECF：PCA 计数器溢出中断允许位。

0：禁止 PCA 计数器溢出中断

1：使能 PCA 计数器溢出中断

24.2.3 PCA 计数器寄存器（CL，CH）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CL	7EFD65H								
CH	7EFD66H								

由 CL 和 CH 两个字节组合成一个 16 位计数器，CL 为低 8 位计数器，CH 为高 8 位计数器。每个 PCA 时钟 16 位计数器自动加 1。

24.2.4 PCA 模块模式控制寄存器（CCAPMn）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CCAPM0	7EFD58H	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0
CCAPM1	7EFD5CH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1
CCAPM2	7EFD50H	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2

ECOMn：允许 PCA 模块 n 的比较功能
CCAPPn：允许 PCA 模块 n 进行上升沿捕获
CCAPNn：允许 PCA 模块 n 进行下降沿捕获
MATn：允许 PCA 模块 n 的匹配功能
TOGn：允许 PCA 模块 n 的高速脉冲输出功能
PWMn：允许 PCA 模块 n 的脉宽调制输出功能
ECCFn：允许 PCA 模块 n 的匹配/捕获中断

24.2.5 PCA 模块模式捕获值/比较值寄存器（CCAPnL，CCAPnH）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CCAP0L	7EFD59H								
CCAP0H	7EFD5AH								
CCAP1L	7EFD5DH								
CCAP1H	7EFD5EH								
CCAP2L	7EFD51H								
CCAP2H	7EFD52H								

当 PCA 模块捕获功能使能时，CCAPnL 和 CCAPnH 用于保存发生捕获时的 PCA 的计数值（CL 和 CH）；当 PCA 模块比较功能使能时，PCA 控制器会将当前 CL 和 CH 中的计数值与保存在 CCAPnL 和 CCAPnH 中的值进行比较，并给出比较结果；当 PCA 模块匹配功能使能时，PCA 控制器会将当前 CL 和 CH 中的计数值与保存在 CCAPnL 和 CCAPnH 中的值进行比较，看是否匹配（相等），并给出匹配结果。

24.2.6 PCA 模块 PWM 模式控制寄存器（PCA_PWMn）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PCA_PWM0	7EFD5BH	EBS0[1:0]		XCCAP0H[1:0]		XCCAP0L[1:0]		EPC0H	EPC0L
PCA_PWM1	7EFD5FH	EBS1[1:0]		XCCAP1H[1:0]		XCCAP1L[1:0]		EPC1H	EPC1L
PCA_PWM2	7EFD53H	EBS2[1:0]		XCCAP2H[1:0]		XCCAP2L[1:0]		EPC2H	EPC2L

EBSn[1:0]：PCA 模块 n 的 PWM 位数控制

EBSn[1:0]	PWM 位数	重载值	比较值
00	8 位 PWM	{EPCnH, CCAPnH[7:0]}	{EPCnL, CCAPnL[7:0]}
01	7 位 PWM	{EPCnH, CCAPnH[6:0]}	{EPCnL, CCAPnL[6:0]}
10	6 位 PWM	{EPCnH, CCAPnH[5:0]}	{EPCnL, CCAPnL[5:0]}
11	10 位 PWM	{EPCnH, XCCAPnH[1:0], CCAPnH[7:0]}	{EPCnL, XCCAPnL[1:0], CCAPnL[7:0]}

XCCAPnH[1:0]：10 位 PWM 的第 9 位和第 10 位的重载值

XCCAPnL[1:0]：10 位 PWM 的第 9 位和第 10 位的比较值

EPCnH：PWM 模式下，重载值的最高位（8 位 PWM 的第 9 位，7 位 PWM 的第 8 位，6 位 PWM 的第 7 位，10 位 PWM 的第 11 位）

EPCnL：PWM 模式下，比较值的最高位（8 位 PWM 的第 9 位，7 位 PWM 的第 8 位，6 位 PWM 的第 7 位，10 位 PWM 的第 11 位）

注意：在更新 10 位 PWM 的重载值时，必须先写高两位 XCCAPnH[1:0]，再写低 8 位 CCAPnH[7:0]。

24.3 PCA 工作模式

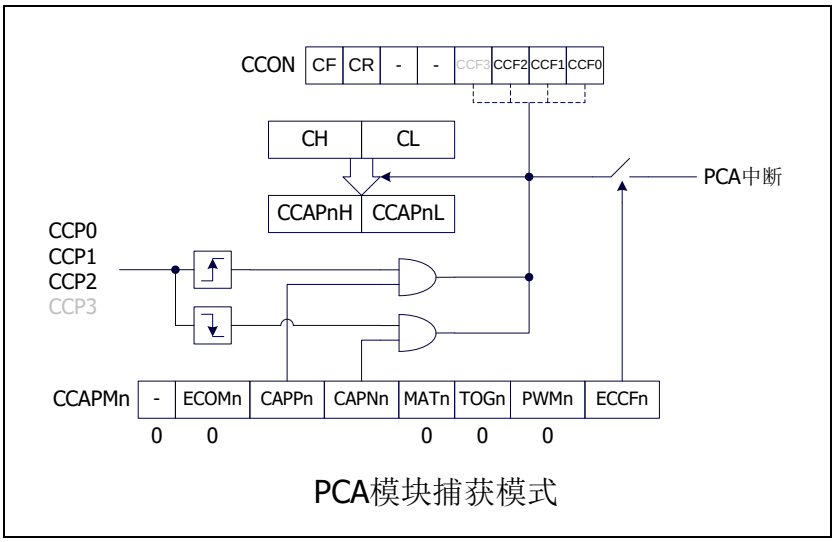
AI8 系列单片机共有 4 组 PCA 模块，每组模块都可独立设置工作模式。模式设置如下所示：

CCAPMn								模块功能
-	ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn	
-	0	0	0	0	0	0	0	无操作
-	1	0	0	0	0	1	0	6/7/8/10 位 PWM 模式，无中断
-	1	1	0	0	0	1	1	6/7/8/10 位 PWM 模式，产生上升沿中断
-	1	0	1	0	0	1	1	6/7/8/10 位 PWM 模式，产生下降沿中断
-	1	1	1	0	0	1	1	6/7/8/10 位 PWM 模式，产生边沿中断
-	0	1	0	0	0	0	x	16 位上升沿捕获
-	0	0	1	0	0	0	x	16 位下降沿捕获
-	0	1	1	0	0	0	x	16 位边沿捕获
-	1	0	0	1	0	0	x	16 位软件定时器
-	1	0	0	1	1	0	x	16 位高速脉冲输出

24.3.1 捕获模式

要使一个 PCA 模块工作在捕获模式，寄存器 CCAPMn 中的 CAPNn 和 CAPPn 至少有一位必须置 1（也可两位都置 1）。PCA 模块工作于捕获模式时，对模块的外部 CCP0/CCP1/CCP2 管脚的输入跳变进行采样。当采样到有效跳变时，PCA 控制器立即将 PCA 计数器 CH 和 CL 中的计数值装载到模块的捕获寄存器中 CCAPnL 和 CCAPnH，同时将 CCON 寄存器中相应的 CCFn 置 1。若 CCAPMn 中的 ECCFn 位被设置为 1，将产生中断。由于所有 PCA 模块的中断入口地址是共享的，所以在中断服务程序中需要判断是哪一个模块产生了中断，并注意中断标志位需要软件清零。

PCA 模块工作于捕获模式的结构图如下图所示：

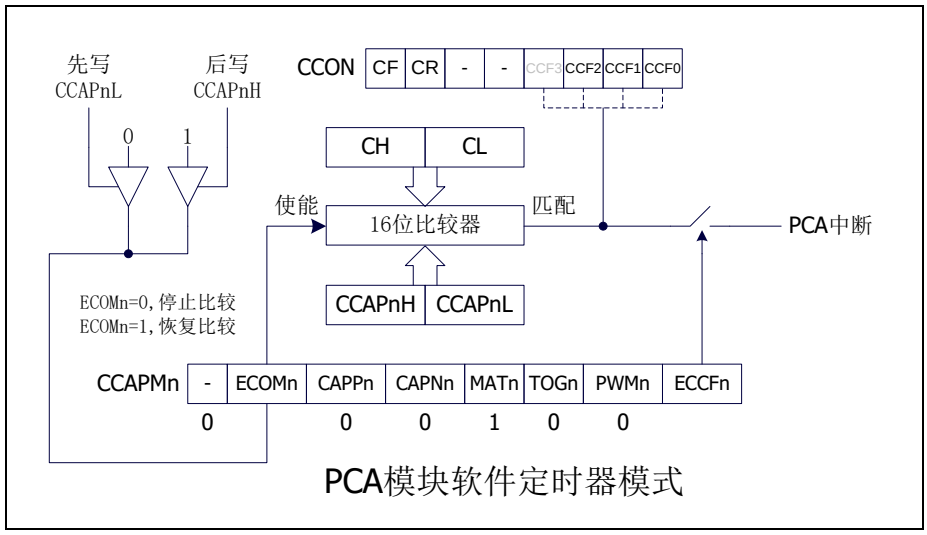


PCA模块捕获模式

24.3.2　软件定时器模式

通过置位 CCAPMn 寄存器的 ECOM 和 MAT 位，可使 PCA 模块用作软件定时器。PCA 计数器值 CL 和 CH 与模块捕获寄存器的值 CCAPnL 和 CCAPnH 相比较，当两者相等时，CCON 中的 CCFn 会被置 1，若 CCAPMn 中的 ECCFn 被设置为 1 时将产生中断。CCFn 标志位需要软件清零。

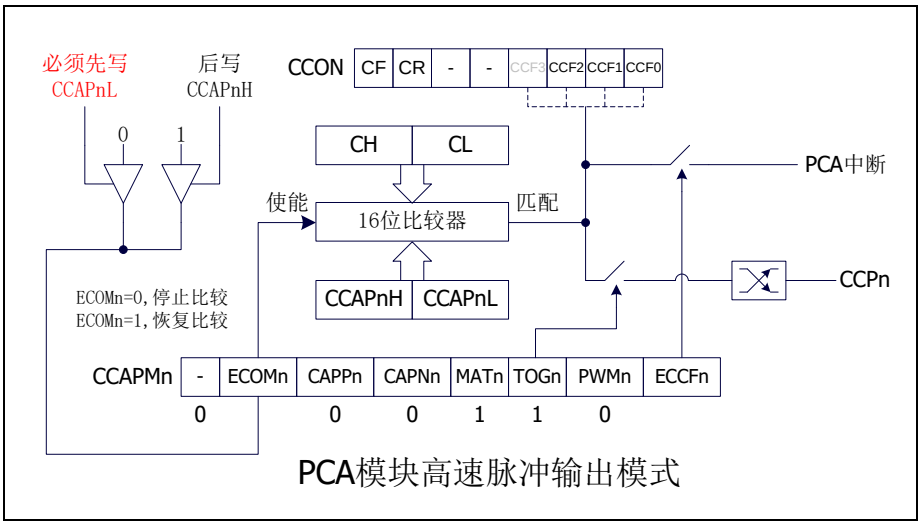
PCA 模块工作于软件定时器模式的结构图如下图所示：



24.3.3　高速脉冲输出模式

当 PCA 计数器的计数值与模块捕获寄存器的值相匹配时,PCA 模块的 CCPn 输出将发生翻转转。要激活高速脉冲输出模式，CCAPMn 寄存器的 TOGn、MATn 和 ECOMn 位必须都置 1。

PCA 模块工作于高速脉冲输出模式的结构图如下图所示：



24.3.4 PWM 脉宽调制模式及频率计算公式

24.3.4.1 8 位 PWM 模式

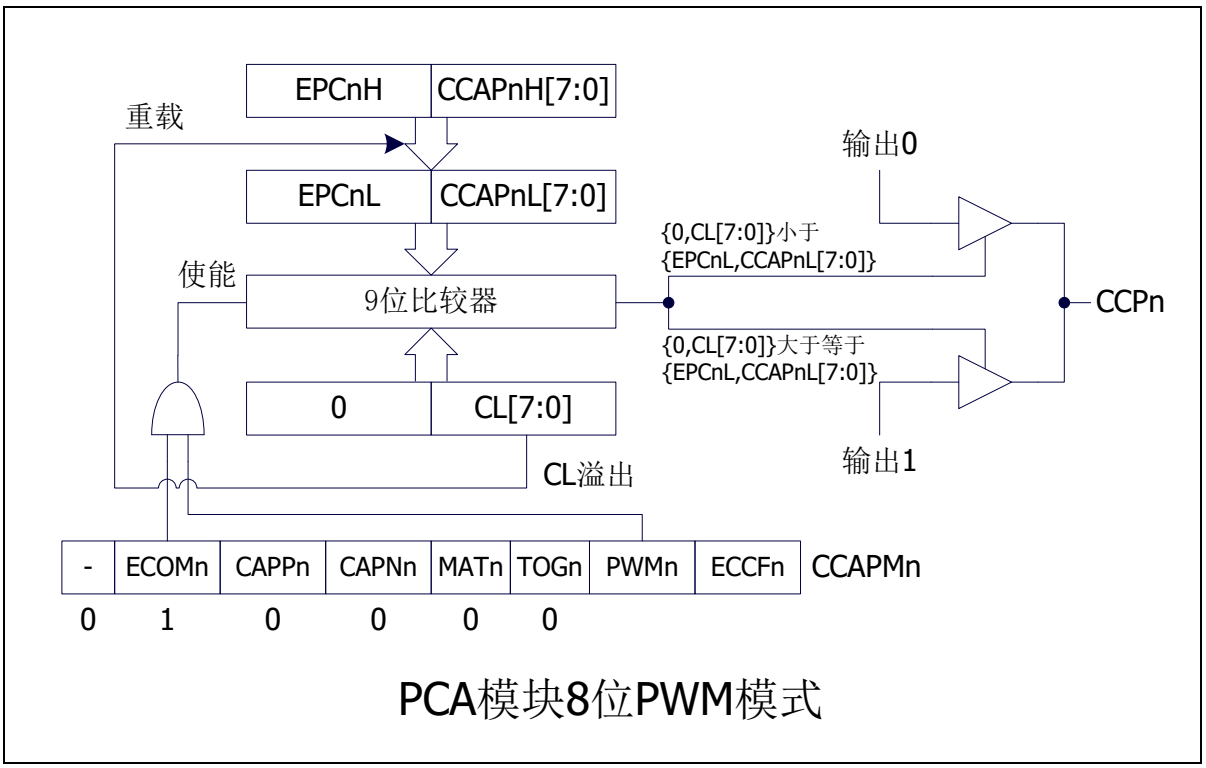
脉宽调制是使用程序来控制波形的占空比、周期、相位波形的一种技术，在三相电机驱动、D/A 转换等场合有广泛的应用。AI8 系列单片机的 PCA 模块可以通过设定各自的 PCA_PWMn 寄存器使其工作于 8 位 PWM 或 7 位 PWM 或 6 位 PWM 或 10 位 PWM 模式。要使能 PCA 模块的 PWM 功能，模块寄存器 CCAPMn 的 PWMn 和 ECOMn 位必须置 1。

PCA_PWMn 寄存器中的 EBSn[1:0]设置为 00 时，PCA 模块 n 工作于 8 位 PWM 模式，此时将{0,CL[7:0]}与捕获寄存器{EPCnL,CCAPnL[7:0]}进行比较。当 PCA 模块工作于 8 位 PWM 模式时，由于所有模块共用一个 PCA 计数器，所有它们的输出频率相同。各个模块的输出占空比使用寄存器{EPCnL,CCAPnL[7:0]}进行设置。当{0,CL[7:0]}的值小于{EPCnL,CCAPnL[7:0]}时，输出为低电平；当{0,CL[7:0]}的值等于或大于{EPCnL,CCAPnL[7:0]}时，输出为高电平。当 CL[7:0]的值由 FF 变为 00 溢出时，{EPCnH,CCAPnH[7:0]}的内容重新装载到{EPCnL,CCAPnL[7:0]}中。这样就可实现无干扰地更新 PWM。

$$8\text{位模式的PWM频率} = \frac{\text{PCA时钟输入源频率}}{256}$$

当EPCnH=0及CCAPnH=00H时，PWM固定输出高
当EPCnH=1及CCAPnH=FFH时，PWM固定输出低

PCA 模块工作于 8 位 PWM 模式的结构图如下图所示：



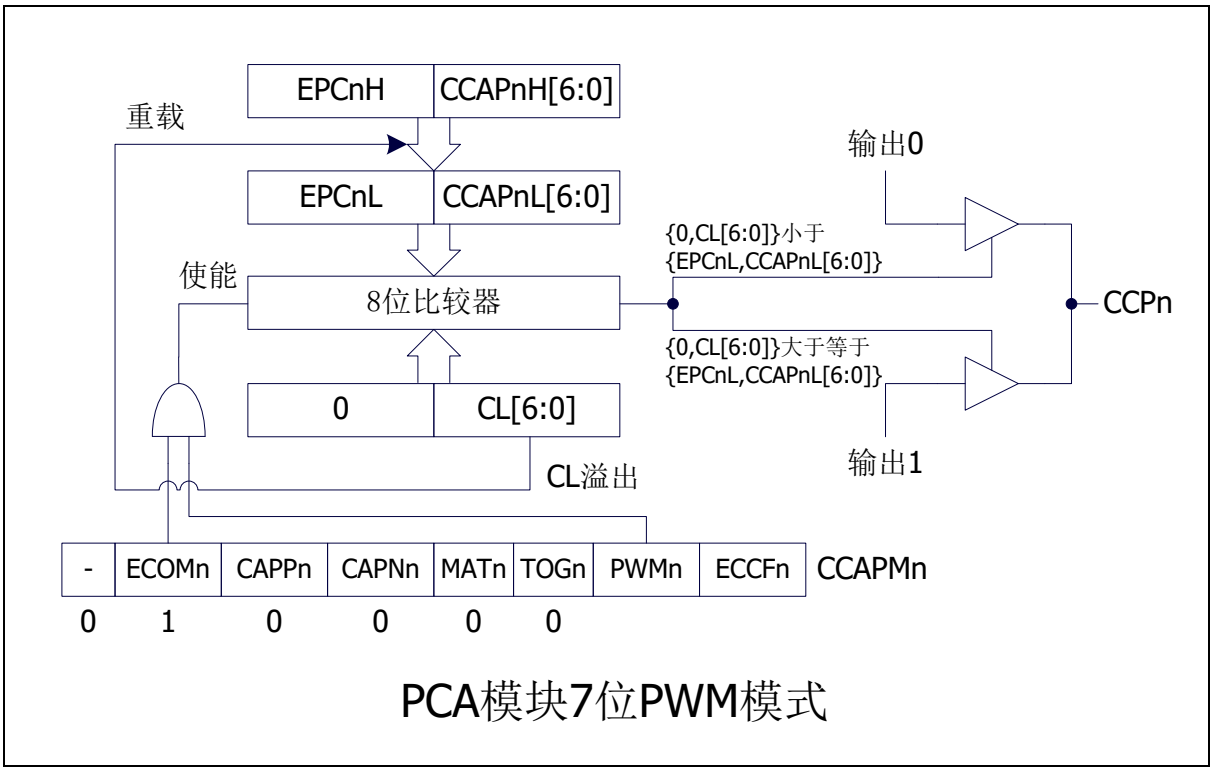
24.3.4.2　7 位 PWM 模式

PCA_PWMn 寄存器中的 EBSn[1:0]设置为 01 时，PCA 模块 n 工作于 7 位 PWM 模式，此时将{0,CL[6:0]}与捕获寄存器{EPCnL,CCAPnL[6:0]}进行比较。当 PCA 模块工作于 7 位 PWM 模式时，由于所有模块共用一个 PCA 计数器，所有它们的输出频率相同。各个模块的输出占空比使用寄存器{EPCnL,CCAPnL[6:0]}进行设置。当{0,CL[6:0]}的值小于{EPCnL,CCAPnL[6:0]}时，输出为低电平；当{0,CL[6:0]}的值等于或大于{EPCnL,CCAPnL[6:0]}时，输出为高电平。当 CL[6:0]的值由 7F 变为 00 溢出时，{EPCnH,CCAPnH[6:0]}的内容重新装载到{EPCnL,CCAPnL[6:0]}中。这样就可实现无干扰地更新 PWM。

$$7\text{位模式的PWM频率} = \frac{\text{PCA时钟输入源频率}}{128}$$

当EPCnH=0及CCAPnH=00H时，PWM固定输出高
当EPCnH=1及CCAPnH=FFH时，PWM固定输出低

PCA 模块工作于 7 位 PWM 模式的结构图如下图所示：



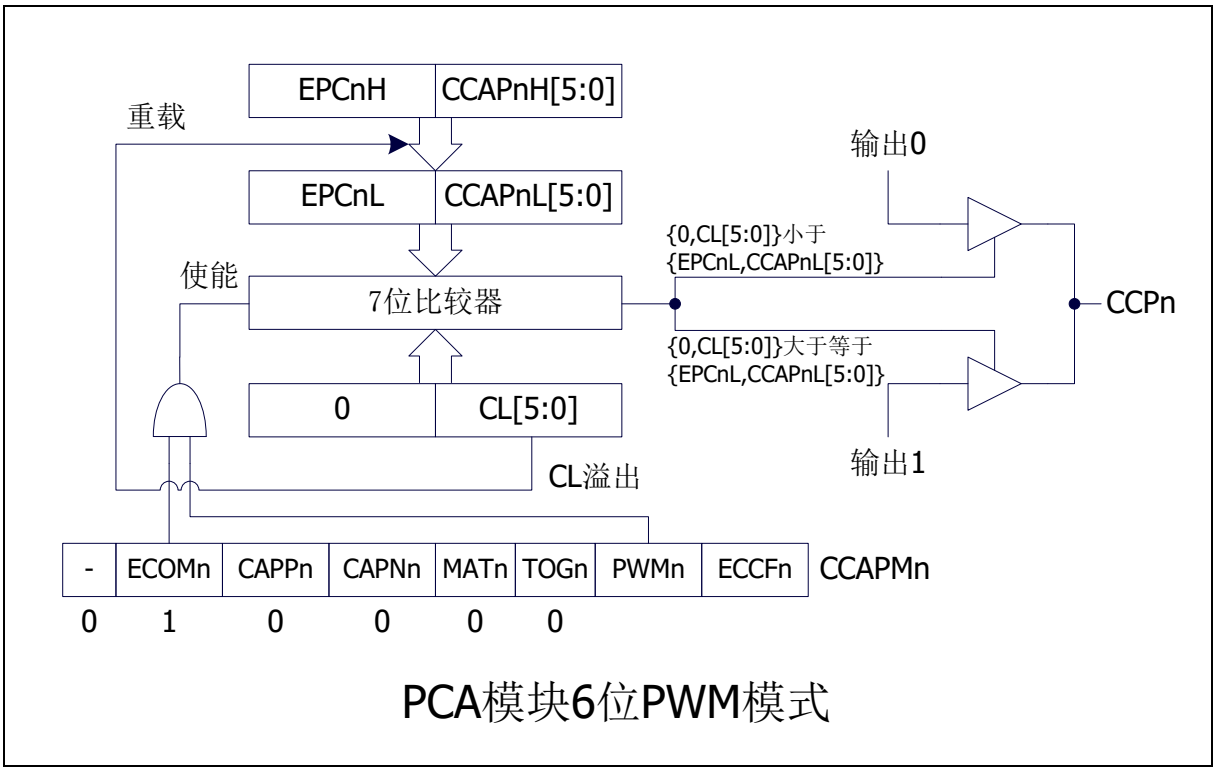
24.3.4.3　6 位 PWM 模式

PCA_PWMn 寄存器中的 EBSn[1:0]设置为 10 时，PCA 模块 n 工作于 6 位 PWM 模式，此时将{0,CL[5:0]}与捕获寄存器{EPCnL,CCAPnL[5:0]}进行比较。当 PCA 模块工作于 6 位 PWM 模式时，由于所有模块共用一个 PCA 计数器，所有它们的输出频率相同。各个模块的输出占空比使用寄存器{EPCnL,CCAPnL[5:0]}进行设置。当{0,CL[5:0]}的值小于{EPCnL,CCAPnL[5:0]}时，输出为低电平；当{0,CL[5:0]}的值等于或大于{EPCnL,CCAPnL[5:0]}时，输出为高电平。当 CL[5:0]的值由 3F 变为 00 溢出时，{EPCnH,CCAPnH[5:0]}的内容重新装载到{EPCnL,CCAPnL[5:0]}中。这样就可实现无干扰地更新 PWM。

$$6\text{位模式的PWM频率} = \frac{\text{PCA时钟输入源频率}}{64}$$

当EPCnH=0及CCAPnH=00H时，PWM固定输出高
当EPCnH=1及CCAPnH=FFH时，PWM固定输出低

PCA 模块工作于 6 位 PWM 模式的结构图如下图所示：



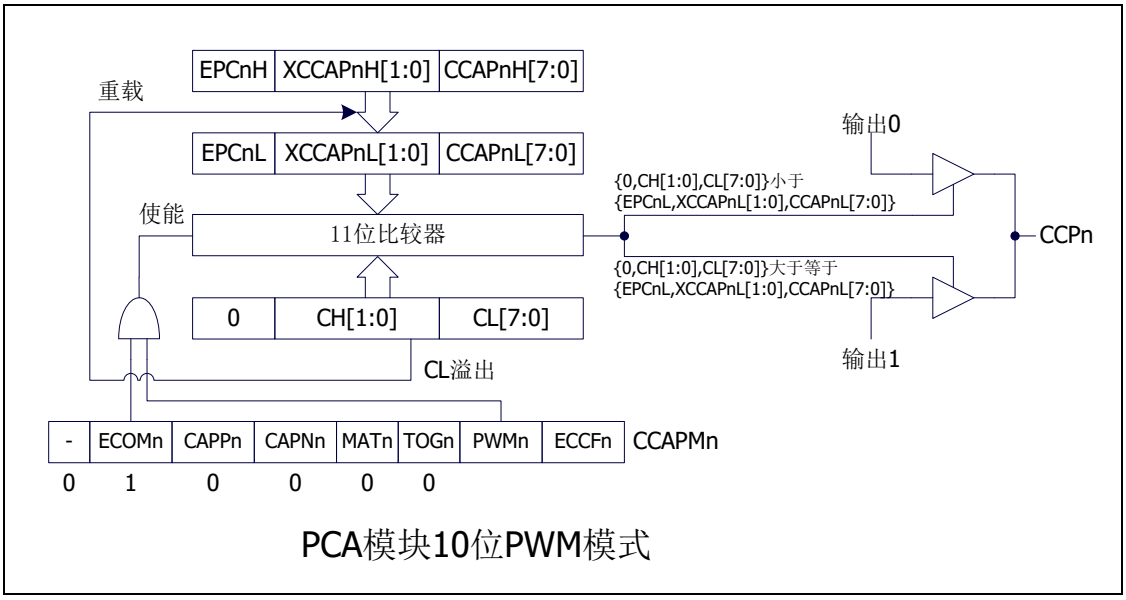
24.3.4.4　10 位 PWM 模式

PCA_PWMn 寄存器中的 EBSn[1:0]设置为 11 时，PCA 模块 n 工作于 10 位 PWM 模式，此时将{CH[1:0],CL[7:0]}与捕获寄存器{EPCnL,XCCAPnL[1:0],CCAPnL[7:0]}进行比较。当 PCA 模块工作于 10 位 PWM 模式时，由于所有模块共用一个 PCA 计数器，所有它们的输出频率相同。各个模块的输出占空比使用寄存器{EPCnL,XCCAPnL[1:0],CCAPnL[7:0]}进行设置。当{CH[1:0],CL[7:0]}的值小于{EPCnL,XCCAPnL[1:0],CCAPnL[7:0]}时，输出为低电平；当{CH[1:0],CL[7:0]}的值等于或大于{EPCnL,XCCAPnL[1:0],CCAPnL[7:0]}时，输出为高电平。当{CH[1:0],CL[7:0]}的值由 3FF 变为 00 溢出时，{EPCnH,XCCAPnH[1:0],CCAPnH[7:0]}的内容重新装载到{EPCnL,XCCAPnL[1:0],CCAPnL[7:0]}中。这样就可实现无干扰地更新 PWM。

$$10\text{位模式的PWM频率} = \frac{\text{PCA时钟输入源频率}}{1024}$$

当EPCnH=0, XCCAP0H=0及CCAPnH=00H时，PWM固定输出高
当EPCnH=1, XCCAP0H=3及CCAPnH=FFH时，PWM固定输出低

PCA 模块工作于 10 位 PWM 模式的结构图如下图所示：



PCA模块10位PWM模式

24.3.4.5　如何控制 PWM 固定输出高电平/低电平

当 PCA_PWMn &= 0xC0，CCAPnH = 0x00 时，PWM 固定输出高电平
当 PCA_PWMn |= 0x3F，CCAPnH = 0xFF 时，PWM 固定输出低电平

24.4 范例程序

24.4.1 PCA 输出 PWM（6/7/8/10 位）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件
#include "intrins.h"                          //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void main()
{
    EAXFR = 1;                                //允许访问扩展的特殊寄存器，XFR
                                              //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                            //允许访问扩展的特殊寄存器，XFR
                                              //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                 //设置取程序代码等待时间，
                                              //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                //设置访问片内的 xdata 速度，
                                              //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CCON = 0x00;
    CMOD = 0x08;                              //PCA 时钟为系统时钟
    CL = 0x00;
    CH = 0x00;
//——6 位 PWM——
    CCAPM0 = 0x42;                            //PCA 模块 0 为 PWM 工作模式
    PCA_PWM0 = 0x80;                          //PCA 模块 0 输出 6 位 PWM
    CCAP0L = 0x20;                            //PWM 占空比为 50%[(40H-20H)/40H]
    CCAP0H = 0x20;
//——7 位 PWM——
    CCAPM1 = 0x42;                            //PCA 模块 1 为 PWM 工作模式
    PCA_PWM1 = 0x40;                          //PCA 模块 1 输出 7 位 PWM
    CCAP1L = 0x20;                            //PWM 占空比为 75%[(80H-20H)/80H]
    CCAP1H = 0x20;
//——8 位 PWM——
//  CCAPM2 = 0x42;                            //PCA 模块 2 为 PWM 工作模式
//  PCA_PWM2 = 0x00;                          //PCA 模块 2 输出 8 位 PWM
//  CCAP2L = 0x20;                            //PWM 占空比为 87.5%[(100H-20H)/100H]
//  CCAP2H = 0x20;
//——10 位 PWM——
    CCAPM2 = 0x42;                            //PCA 模块 2 为 PWM 工作模式
```

```
    PCA_PWM2 = 0xc0;                        //PCA 模块 2 输出 10 位 PWM
    CCAP2L = 0x20;                          //PWM 占空比为 96.875%[(400H-20H)/400H]
    CCAP2H = 0x20;
    CCON |= CR;                             //启动 PCA 计时器

    while (1);
}
```

24.4.2　PCA 捕获测量脉冲宽度

C 语言代码

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                        //头文件见下载软件
#include "intrins.h"                        //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

unsigned char       cnt;                    //存储 PCA 计时溢出次数
unsigned long       count0;                 //记录上一次的捕获值
unsigned long       count1;                 //记录本次的捕获值
unsigned long       length;                 //存储信号的时间长度

void PCA_Isr() interrupt 7
{
    if (CCON & CF)
    {
        CCON &= ~CF;
        cnt++;                              //PCA 计时溢出次数+1
    }
    if (CCON & CCF0)
    {
        CCON &= ~CCF0;
        count0 = count1;                    //备份上一次的捕获值
        ((unsigned char *)&count1)[3] = CCAP0L;
        ((unsigned char *)&count1)[2] = CCAP0H;
        ((unsigned char *)&count1)[1] = cnt;
        ((unsigned char *)&count1)[0] = 0;
        length = count1 - count0;           //length 保存的即为捕获的脉冲宽度
    }
}

void main()
{
    EAXFR = 1;                              //允许访问扩展的特殊寄存器，XFR
                                            //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                          //允许访问扩展的特殊寄存器，XFR
                                            //(8 位模式请使用这句，注释上一句)
    WTST = 0;                               //设置取程序代码等待时间，
                                            //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                              //设置访问片内的 xdata 速度，
                                            //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
```

```
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    cnt = 0;                                    //用户变量初始化
    count0 = 0;
    count1 = 0;
    length = 0;
    CCON = 0x00;
    CMOD = 0x09;                                //PCA 时钟为系统时钟,使能 PCA 计时中断
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x11;                              //PCA 模块 0 为 16 位捕获模式（下降沿捕获）
//  CCAPM0 = 0x21;                              //PCA 模块 0 为 16 位捕获模式（上升沿捕获）
//  CCAPM0 = 0x31;                              //PCA 模块 0 为 16 位捕获模式（边沿捕获）
    CCAP0L = 0x00;
    CCAP0H = 0x00;
    CCON |= CR;                                     //启动 PCA 计时器
    EA = 1;

    while (1);
}
```

24.4.3　PCA 实现 16 位软件定时

C 语言代码

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                             //头文件见下载软件
#include "intrins.h"                              //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define     T50HZ               (11059200L / 12 / 2 / 50)

unsigned int                value;

void PCA_Isr() interrupt 7
{
    CCON &= ~CCF0;
    CCAP0L = value;
    CCAP0H = value >> 8;
    value += T50HZ;

    P10 = !P10;                                 //测试端口
}

void main()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
```

```
                                                    //(32 位模式请使用这句，注释下一句)
//      P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                    //(8 位模式请使用这句，注释上一句)
        WTST = 0;                                   //设置取程序代码等待时间，
                                                    //赋值为 0 表示不等待，程序以最快速度运行
        CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                    //赋值为 0 表示用最快速度访问，不增加额外的等待时间

        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

        CCON = 0x00;
        CMOD = 0x00;                                //PCA 时钟为系统时钟/12
        CL = 0x00;
        CH = 0x00;
        CCAPM0 = 0x49;                              //PCA 模块 0 为 16 位定时器模式
        value = T50HZ;
        CCAP0L = value;
        CCAP0H = value >> 8;
        value += T50HZ;
        CCON |= CR;                                 //启动 PCA 计时器
        EA = 1;

        while (1);
}
```

24.4.4　PCA 实现 16 位软件定时（ECI 外部时钟模式）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                               //头文件见下载软件
#include "intrins.h"                                //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define     T50HZ               (11059200L / 12 / 2 / 50)

unsigned int                value;

void PCA_Isr() interrupt 7
{
        CCON &= ~CCF0;
        CCAP0L = value;
        CCAP0H = value >> 8;
        value += T50HZ;
```

```
    P10 = !P10;                                 //测试端口
}

void main()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CCON = 0x00;
    CMOD = 0x06;                                //PCA 时钟为从 ECI 端口输入的外部时钟
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x49;                              //PCA 模块 0 为 16 位定时器模式
    value = T50HZ;
    CCAP0L = value;
    CCAP0H = value >> 8;
    value += T50HZ;
    CCON |= CR;                                 //启动 PCA 计时器
    EA = 1;

    while (1);
}
```

24.4.5　PCA 输出高速脉冲

C 语言代码

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                            //头文件见下载软件
#include "intrins.h"                            //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define     T38K4HZ             (11059200L / 2 / 38400)

unsigned int                value;
```

```
void PCA_Isr() interrupt 7
{
    CCON &= ~CCF0;
    CCAP0L = value;
    CCAP0H = value >> 8;
    value += T38K4HZ;
}

void main()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CCON = 0x00;
    CMOD = 0x08;                                //PCA 时钟为系统时钟
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x4d;                              //PCA 模块 0 为 16 位定时器模式并使能脉冲输出
    value = T38K4HZ;
    CCAP0L = value;
    CCAP0H = value >> 8;
    value += T38K4HZ;
    CCON |= CR;                                 //启动 PCA 计时器
    EA = 1;

    while (1);
}
```

24.4.6　PCA 扩展外部中断

C 语言代码

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                            //头文件见下载软件
#include "intrins.h"                            //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。
```

```
void PCA_Isr() interrupt 7
{
    CCON &= ~CCF0;
    P10 = !P10;
}

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                           //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CCON = 0x00;
    CMOD = 0x08;                        //PCA 时钟为系统时钟
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x11;                      //扩展外部端口 CCP0 为下降沿中断口
//  CCAPM0 = 0x21;                      //扩展外部端口 CCP0 为上升沿中断口
//  CCAPM0 = 0x31;                      //扩展外部端口 CCP0 为边沿中断口
    CCAP0L = 0;
    CCAP0H = 0;
    CCON |= CR;                             //启动 PCA 计时器
    EA = 1;

    while (1);
}
```

25 同步串行外设接口（SPI）

产品线	SPI
Ai8051U 系列	●

Ai8051U 系列单片机内部集成了一种高速串行通信接口——SPI 接口。SPI 是一种全双工的高速同步通信总线。Ai8051U 系列集成的 SPI 接口提供了两种操作模式：主模式和从模式。

注：USART1 和 USART2 的 SPI 模式可支持两组完整的 SPI，详细情况请参考 USART 章节

25.1 SPI 的 MOSI 和 MISO 脚交换控制

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
HSSPI_CFG2	7EFBF9H	-	IOSW	HSSPIEN	FIFOEN	SS_DACT[3:0]			

IOSW：交换 MOSI 和 MISO 脚位（普通 SPI 模式和高速 SPI 模式通用）

0：不交换，维持上电默认脚位。

1：交换 MOSI 和 MISO 的脚位。

25.2 SPI 功能脚切换

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW1	A2H	S1_S[1:0]		-		SPI_S[1:0]		-	

SPI_S[1:0]：SPI 功能脚选择位

SPI_S[1:0]	SS	MOSI	MISO	SCLK
00	P1.4	P1.5	P1.6	P1.7
01	P2.4	P2.5	P2.6	P2.7
10	P4.0	P4.1	P4.2	P4.3
11	P3.5	P3.4	P3.3	P3.2

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW3	BBH	I2S_S		S2SPI_S[1:0]		S1SPI_S[1:0]			

S2SPI_S[1:0]：USART2 的 SPI 功能脚选择位

S2SPI_S[1:0]	S2SS	S2MOSI	S2MISO	S2SCLK
00	P1.4	P1.5	P1.6	P1.7
01	P2.4	P2.5	P2.6	P2.7
10	P4.0	P4.1	P4.2	P4.3
11	P3.5	P3.4	P3.3	P3.2

S1SPI_S[1:0]：USART1 的 SPI 功能脚选择位

S1SPI_S[1:0]	S1SS	S1MOSI	S1MISO	S1SCLK
00	P1.4	P1.5	P1.6	P1.7
01	P2.4	P2.5	P2.6	P2.7
10	P4.0	P4.1	P4.2	P4.3
11	P3.5	P3.4	P3.3	P3.2

25.3 SPI 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
SPSTAT	SPI 状态寄存器	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx,xxxx
SPCTL	SPI 控制寄存器	CEH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR[1:0]		0000,0100
SPDAT	SPI 数据寄存器	CFH									0000,0000

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
SPITOCR	SPI 从机超时控制寄存器	7EFD80H	ENTO	ENTOI	SCALE	-	-	-	-	-	000x,xxxx
SPITOSR	SPI 从机超时状态寄存器	7EFD81H	CTOIF	-	-	-	-	-	-	TOIF	0xxx,xxx0
SPITOTH	SPI 从机超时长度寄存器	7EFD82H	TM[15:8]								0000,0000
SPITOTL	SPI 从机超时长度寄存器	7EFD83H	TM[7:0]								0000,0000
SPITOTE	SPI 接收超时长度寄存器	7EFD8CH	TM[23:16]								0000,0000

25.3.1 SPI 状态寄存器（SPSTAT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPSTAT	CDH	SPIF	WCOL	-	-	-	-	-	-

SPIF：SPI 中断标志位。

当发送/接收完成 1 字节的数据后，硬件自动将此位置 1，并向 CPU 提出中断请求。当 SSIG 位被设置为 0 时，由于 SS 管脚电平的变化而使得设备的主/从模式发生改变时，此标志位也会被硬件自动置 1，以标志设备模式发生变化。

注意：此标志位必须用户通过软件方式向此位写 1 进行清零。

WCOL：SPI 写冲突标志位。

当 SPI 在进行数据传输的过程中写 SPDAT 寄存器时，硬件将此位置 1。

注意：此标志位必须用户通过软件方式向此位写 1 进行清零。

25.3.2 SPI 控制寄存器（SPCTL），SPI 速度控制

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPCTL	CEH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR[1:0]	

SSIG：SS 引脚功能控制位

0：SS 引脚确定器件是主机还是从机

1：忽略 SS 引脚功能，使用 MSTR 确定器件是主机还是从机

SPEN：SPI 使能控制位

0：关闭 SPI 功能

1：使能 SPI 功能

DORD：SPI 数据位发送/接收的顺序

0：先发送/接收数据的高位（MSB）

1：先发送/接收数据的低位（LSB）

MSTR：器件主/从模式选择位

设置主机模式：

若 SSIG＝0，则 SS 管脚必须为高电平且设置 MSTR 为 1

若 SSIG＝1，则只需要设置 MSTR 为 1（忽略 SS 管脚的电平）

设置从机模式：

若 SSIG＝0，则 SS 管脚必须为低电平（与 MSTR 位无关）

若 SSIG＝1，则只需要设置 MSTR 为 0（忽略 SS 管脚的电平）

CPOL：SPI 时钟极性控制

0：SCLK 空闲时为低电平，SCLK 的前时钟沿为上升沿，后时钟沿为下降沿

1：SCLK 空闲时为高电平，SCLK 的前时钟沿为下降沿，后时钟沿为上升沿

CPHA：SPI 时钟相位控制

0：数据 SS 管脚为低电平驱动第一位数据并在 SCLK 的后时钟沿改变数据，前时钟沿采样数据（必须 SSIG＝0）

1：数据在 SCLK 的前时钟沿驱动，后时钟沿采样

SPR[1:0]：SPI 时钟频率选择

SPR[1:0]	SCLK 频率
00	SPI 输入时钟/4
01	SPI 输入时钟/8
10	SPI 输入时钟/16
11	SPI 输入时钟/2

25.3.3 SPI 数据寄存器（SPDAT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPDAT	CFH								

SPI 发送/接收数据缓冲器。

25.3.4 SPI 从机超时控制寄存器（SPITOCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPITOCR	7EFD80H	ENTO	ENTOI	SCALE	-	-	-	-	-

ENTO：SPI 从机超时功能控制位

0：禁止 SPI 从机超时功能

1：使能 SPI 从机超时功能（**注：SPI 的主机模式不要使能接收超时功能**）

ENTOI：SPI 从机超时中断控制位

0：禁止 SPI 从机超时中断

1：使能 SPI 从机超时中断

SCALE：SPI 从机计数时钟源选择

0：1us 时钟（1MHz 时钟）。（注意：如需要使用此时钟源，用户必须先正确设置 IAP_TPS 寄存器。例如：若系统时钟为 12MHz，则需要将 IAP_TPS 设置为 12；若系统时钟为 22.1184MHz，则需要将 IAP_TPS 设置为 22；其他频率以此类推。特别注意，此 1us 的时钟并不是精准时间）

1：系统时钟

25.3.5 SPI 从机超时状态寄存器（SPITOSR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPITOSR	7EFD81H	CTOIF	-	-	-	-	-	-	TOIF

CTOIF：写“1”清除 SPI 超时中断标志位 TOIF。（只写）

TOIF：SPI 超时中断请求标志位。（只读）

当发生 SPI 超时时，TOIF 会被硬件置“1”。如果 ENTOI 为 1 时会产生 SPI 中断，中断入口地址为原 SPI 的中断入口地址。标志位需要软件向 CTOIF 位写“1”清零

25.3.6 SPI 从机超时长度控制寄存器（SPITOTE/H/L）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPITOTE	7EFD8CH	TM[23:16]							
SPITOTH	7EFD82H	TM[15:8]							
SPITOTL	7EFD83H	TM[7:0]							

TM[23:0]：SPI 超时时间控制位。

当 SPI 处于接收空闲状态时，内部超时计数器根据 SCALE 寄存器所选择的时钟源进行计数，当计数时间达到 TM 所设置的超时时间时，便会产生超时中断。当 SPI 接收数据完成时，复位内部超时计数器，重新进行超时计数。

注：

1、如果需要使能接收超时中断功能，则 TM 不可设置为 0，且 SPITOTL、SPITOTH、SPITOTE 寄存器的设置必须先设置 SPITOTL 和 SPITOTH，最后设置 SPITOTE

2、必须设置 SPI 为从机模式才有接收超时功能

3、接收超时功能必须在正确接收到一字节数据后才能触发

4、正在接收过程中，无接收超时功能

25.4 SPI 通信方式

SPI 的通信方式通常有 3 种：单主单从（一个主机设备连接一个从机设备）、互为主从（两个设备连接，设备和互为主机和从机）、单主多从（一个主机设备连接多个从机设备）

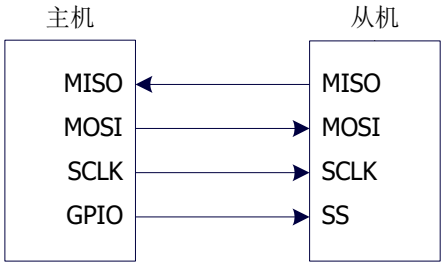
25.4.1 单主单从

两个设备相连，其中一个设备固定作为主机，另外一个固定作为从机。

主机设置：SSIG 设置为 1，MSTR 设置为 1，固定为主机模式。主机可以使用任意端口连接从机的 SS 管脚，拉低从机的 SS 脚即可使能从机

从机设置：SSIG 设置为 0，SS 管脚作为从机的片选信号。

单主单从连接配置图如下所示：



单主单从配置

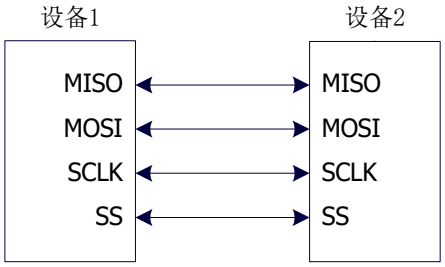
25.4.2 互为主从

两个设备相连，主机和从机不固定。

设置方法 1：两个设备初始化时都设置为 SSIG 设置为 0，MSTR 设置为 1，且将 SS 脚设置为双向口模式输出高电平。此时两个设备都是不忽略 SS 的主机模式。当其中一个设备需要启动传输时，可将自己的 SS 脚设置为输出模式并输出低电平，拉低对方的 SS 脚，这样另一个设备就被强行设置为从机模式了。

设置方法 2：两个设备初始化时都将自己设置成忽略 SS 的从机模式，即将 SSIG 设置为 1，MSTR 设置为 0。当其中一个设备需要启动传输时，先检测 SS 管脚的电平，如果时候高电平，就将自己设置成忽略 SS 的主模式，即可进行数据传输了。

互为主从连接配置图如下所示：



互为主从配置

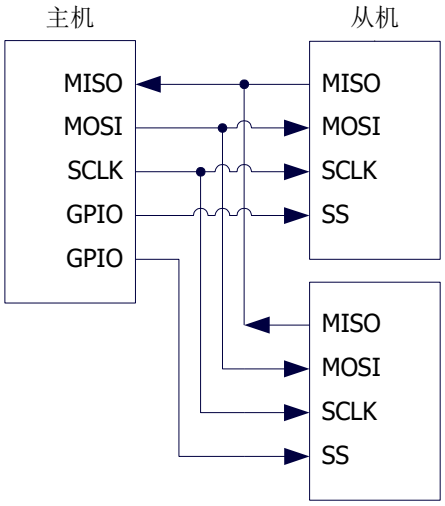
25.4.3 单主多从

多个设备相连，其中一个设备固定作为主机，其他设备固定作为从机。

主机设置：SSIG 设置为 1，MSTR 设置为 1，固定为主机模式。主机可以使用任意端口分别连接各个从机的 SS 管脚，拉低其中一个从机的 SS 脚即可使能相应的从机设备

从机设置：SSIG 设置为 0，SS 管脚作为从机的片选信号。

单主多从连接配置图如下所示：



单主多从配置

25.5 配置 SPI

控制位			通信端口				说明
SPEN	SSIG	MSTR	SS	MISO	MOSI	SCLK	
0	x	x	x	输入	输入	输入	关闭 SPI 功能，SS/MOSI/MISO/SCLK 均为普通 IO
1	0	0	0	输出	输入	输入	**从机模式**，且被选中
1	0	0	1	高阻	输入	输入	**从机模式**，但未被选中
1	0	1→0	0	输出	输入	输入	**从机模式**，不忽略 SS 且 MSTR 为 1 的主机模式，当 SS 管脚被拉低时，MSTR 将被硬件自动清零，工作模式将被被动设置为从机模式
1	0	1	1	输入	高阻	高阻	**主机模式**，空闲状态
					输出	输出	**主机模式**，激活状态
1	1	0	x	输出	输入	输入	**从机模式**
1	1	1	x	输入	输出	输出	**主机模式**

从机模式的注意事项：

当 CPHA＝0 时，SSIG 必须为 0（即不能忽略 SS 脚）。在每次串行字节开始还发送前 SS 脚必须拉低，并且在串行字节发送完后须重新设置为高电平。SS 管脚为低电平时不能对 SPDAT 寄存器执行写操作，否则将导致一个写冲突错误。CPHA=0 且 SSIG=1 时的操作未定义。

当 CPHA＝1 时，SSIG 可以置 1（即可以忽略脚）。如果 SSIG＝0，SS 脚可在连续传输之间保持低有效（即一直固定为低电平）。这种方式适用于固定单主单从的系统。

主机模式的注意事项：

在 SPI 中，传输总是由主机启动的。如果 SPI 使能（SPEN=1）并选择作为主机时，主机对 SPI 数据寄存器 SPDAT 的写操作将启动 SPI 时钟发生器和数据的传输。在数据写入 SPDAT 之后的半个到一个 SPI 位时间后，数据将出现在 MOSI 脚。写入主机 SPDAT 寄存器的数据从 MOSI 脚移出发送到从机的 MOSI 脚。同时从机 SPDAT 寄存器的数据从 MISO 脚移出发送到主机的 MISO 脚。

传输完一个字节后，SPI 时钟发生器停止，传输完成标志（SPIF）置位，如果 SPI 中断使能则会产生一个 SPI 中断。主机和从机 CPU 的两个移位寄存器可以看作是一个 16 位循环移位寄存器。当数据从主机移位传送到从机的同时，数据也以相反的方向移入。这意味着在一个移位周期中，主机和从机的数据相互交换。

通过 SS 改变模式

如果 SPEN=1，SSIG=0 且 MSTR=1，SPI 使能为主机模式，并将 SS 脚可配置为输入模式化或准双向口模式。这种情况下，另外一个主机可将该脚驱动为低电平，从而将该器件选择为 SPI 从机并向其发送数据。为了避免争夺总线，SPI 系统将该从机的 MSTR 清零，MOSI 和 SCLK 强制变为输入模式，而 MISO 则变为输出模式，同时 SPSTAT 的 SPIF 标志位置 1。

用户软件必须一直对 MSTR 位进行检测，如果该位被一个从机选择动作而被动清零，而用户想继续将 SPI 作为主机，则必须重新设置 MSTR 位，否则将一直处于从机模式。

写冲突

SPI 在发送时为单缓冲，在接收时为双缓冲。这样在前一次发送尚未完成之前，不能将新的数据写入移位寄存器。当发送过程中对数据寄存器 SPDAT 进行写操作时，WCOL 位将被置 1 以指示发生数据写冲突错误。在这种情况下，当前发送的数据继续发送，而新写入的数据将丢失。

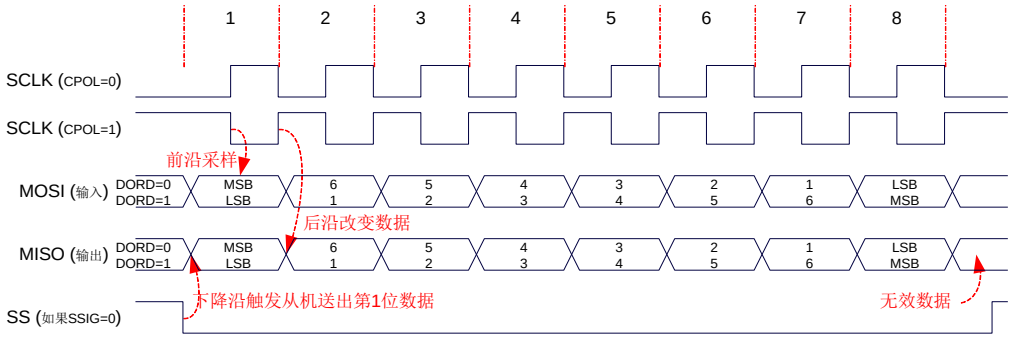
当对主机或从机进行写冲突检测时，主机发生写冲突的情况是很罕见的，因为主机拥有数据传输的完全控制权。但从机有可能发生写冲突，因为当主机启动传输时，从机无法进行控制。

接收数据时，接收到的数据传送到一个并行读数据缓冲区，这样将释放移位寄存器以进行下一个数据的接收。但必须在下个字符完全移入之前从数据寄存器中读出接收到的数据，否则，前一个接收数据将丢失。

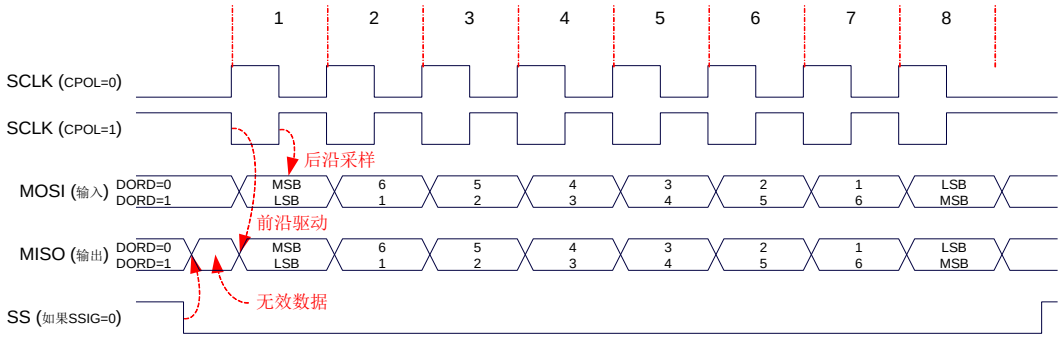
WCOL 可通过软件向其写入“1”清零。

25.6 数据模式

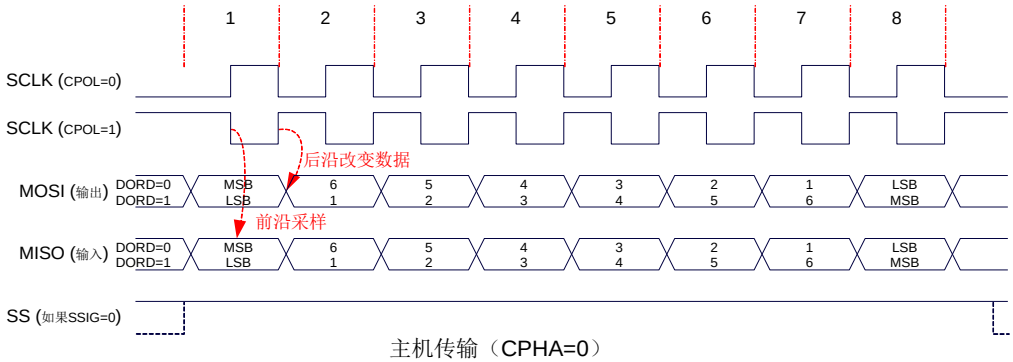
SPI 的时钟相位控制位 CPHA 可以让用户设定数据采样和改变时的时钟沿。时钟极性位 CPOL 可以让用户设定时钟极性。下面图例显示了不同时钟相位、极性设置下 SPI 通讯时序。



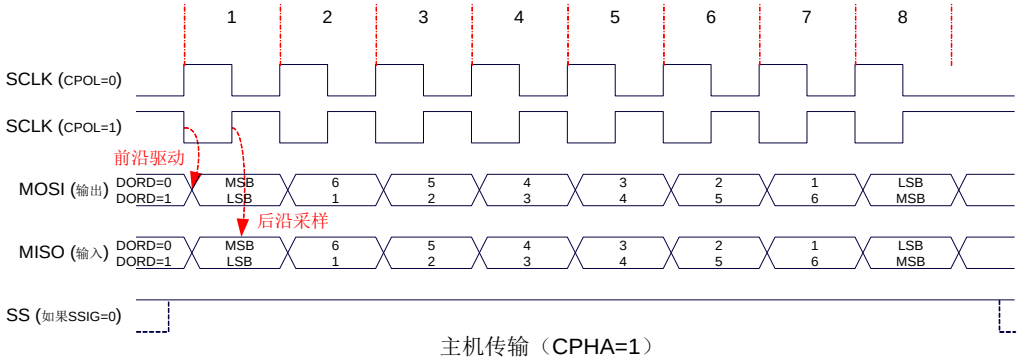
从机传输（CPHA=0）



从机传输（CPHA=1）



主机传输（CPHA=0）



主机传输（CPHA=1）

25.7　范例程序

25.7.1　SPI 单主单从系统主机程序（中断方式）

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

sbit    SS      =   P1^0;
sbit    LED     =   P1^1;

bit     busy;

void SPI_Isr() interrupt 9
{
    SPIF = 1;                                           //清中断标志
    SS = 1;                                             //拉高从机的 SS 管脚
    busy = 0;
    LED = !LED;                                         //测试端口
}

void main()
{
    EAXFR = 1;                                          //允许访问扩展的特殊寄存器，XFR
                                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                      //允许访问扩展的特殊寄存器，XFR
                                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                           //设置取程序代码等待时间，
                                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                          //设置访问片内的 xdata 速度，
                                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    LED = 1;
    SS = 1;
    busy = 0;

    SPCTL = 0x50;                                       //使能 SPI 主机模式
    SPSTAT = 0xc0;                                      //清中断标志
    ESPI = 1;                                           //使能 SPI 中断
    EA = 1;

    while (1)
```

```
    {
        while (busy);
        busy = 1;
        SS = 0;                                 //拉低从机 SS 管脚
        SPDAT = 0x5a;                           //发送测试数据
    }
}
```

25.7.2　SPI 单主单从系统从机程序（中断方式）

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                            //头文件见下载软件
#include "intrins.h"                            //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

sbit    LED     =   P1^1;

void SPI_Isr() interrupt 9
{
    SPIF = 1;                                   //清中断标志
    SPDAT = SPDAT;                              //将接收到的数据回传给主机
    LED = !LED;                                 //测试端口
}

void main()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    SPCTL = 0x40;                               //使能 SPI 从机模式
    SPSTAT = 0xc0;                              //清中断标志
    ESPI = 1;                                   //使能 SPI 中断
    EA = 1;

    while (1);
```

```
}
```

25.7.3　SPI 单主单从系统主机程序（查询方式）

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

sbit    SS      =   P1^0;
sbit    LED     =   P1^1;

void main()
{
    EAXFR = 1;                                          //允许访问扩展的特殊寄存器，XFR
                                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                      //允许访问扩展的特殊寄存器，XFR
                                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                           //设置取程序代码等待时间，
                                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                          //设置访问片内的 xdata 速度，
                                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    LED = 1;
    SS = 1;

    SPCTL = 0x50;                                       //使能 SPI 主机模式
    SPSTAT = 0xc0;                                      //清中断标志

    while (1)
    {
        SS = 0;                                         //拉低从机 SS 管脚
        SPDAT = 0x5a;                                   //发送测试数据
        while (!SPIF);                                  //查询完成标志
        SPIF = 1;                                       //清中断标志
        SS = 1;                                         //拉高从机的 SS 管脚
        LED = !LED;                                     //测试端口
    }
}
```

25.7.4　SPI 单主单从系统从机程序（查询方式）

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

sbit        LED             =       P1^1;

void main()
{
    EAXFR = 1;                                          //允许访问扩展的特殊寄存器，XFR
                                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                      //允许访问扩展的特殊寄存器，XFR
                                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                           //设置取程序代码等待时间，
                                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                          //设置访问片内的 xdata 速度，
                                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    SPCTL = 0x40;                                       //使能 SPI 从机模式
    SPSTAT = 0xc0;                                      //清中断标志

    while (1)
    {
        while (!SPIF);                                  //查询完成标志
        SPIF = 1;                                       //清中断标志
        SPDAT = SPDAT;                                  //将接收到的数据回传给主机
        LED = !LED;                                     //测试端口
    }
}
```

25.7.5　SPI 互为主从系统程序（中断方式）

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。
```

```
sbit    SS      =   P1^0;
sbit    LED     =   P1^1;
sbit    KEY     =   P0^0;

void SPI_Isr() interrupt 9
{
    SPIF = 1;                       //清中断标志
    if (SPCTL & 0x10)
    {                               //主机模式
        SS = 1;                     //拉高从机的 SS 管脚
        SPCTL = 0x40;               //重新设置为从机待机
    }
    else
    {                               //从机模式
        SPDAT = SPDAT;              //将接收到的数据回传给主机
    }
    LED = !LED;                     //测试端口
}

void main()
{
    EAXFR = 1;                      //允许访问扩展的特殊寄存器，XFR
                                    //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                  //允许访问扩展的特殊寄存器，XFR
                                    //(8 位模式请使用这句，注释上一句)
    WTST = 0;                       //设置取程序代码等待时间，
                                    //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                      //设置访问片内的 xdata 速度，
                                    //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    LED = 1;
    KEY = 1;
    SS = 1;

    SPCTL = 0x40;                   //使能 SPI 从机模式进行待机
    SPSTAT = 0xc0;                  //清中断标志
    ESPI = 1;                       //使能 SPI 中断
    EA = 1;

    while (1)
    {
        if (!KEY)                   //等待按键触发
        {
            SPCTL = 0x50;           //使能 SPI 主机模式
            SS = 0;                 //拉低从机 SS 管脚
```

```
            SPDAT = 0x5a;                   //发送测试数据
            while (!KEY);                   //等待按键释放
        }
    }
}
```

25.7.6 SPI 互为主从系统程序（查询方式）

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                        //头文件见下载软件
#include "intrins.h"                        //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

sbit    SS      =   P1^0;
sbit    LED     =   P1^1;
sbit    KEY     =   P0^0;

void main()
{
    EAXFR = 1;                              //允许访问扩展的特殊寄存器，XFR
                                            //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                          //允许访问扩展的特殊寄存器，XFR
                                            //(8 位模式请使用这句，注释上一句)
    WTST = 0;                               //设置取程序代码等待时间，
                                            //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                              //设置访问片内的 xdata 速度，
                                            //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    LED = 1;
    KEY = 1;
    SS = 1;

    SPCTL = 0x40;                           //使能 SPI 从机模式进行待机
    SPSTAT = 0xc0;                          //清中断标志

    while (1)
    {
        if (!KEY)                           //等待按键触发
        {
            SPCTL = 0x50;                   //使能 SPI 主机模式
            SS = 0;                         //拉低从机 SS 管脚
```

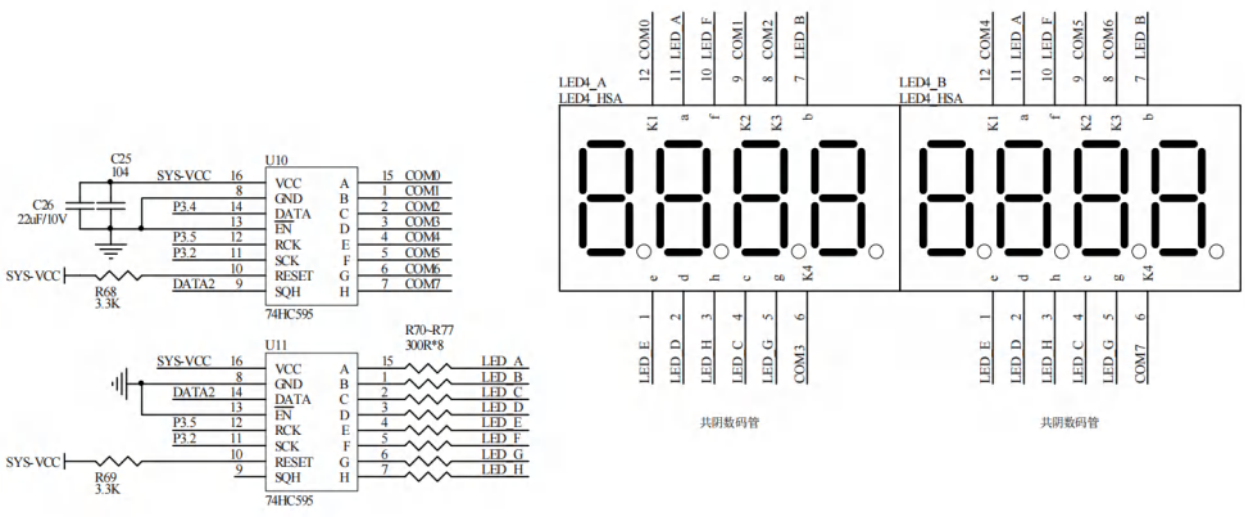
```
            SPDAT = 0x5a;                   //发送测试数据
            while (!KEY);                   //等待按键释放
        }
        if (SPIF)
        {
            SPIF = 1;                       //清中断标志
            if (SPCTL & 0x10)
            {                               //主机模式
                SS = 1;                     //拉高从机的 SS 管脚
                SPCTL = 0x40;               //重新设置为从机待机
            }
            else
            {                               //从机模式
                SPDAT = SPDAT;              //将接收到的数据回传给主机
            }
            LED = !LED;                     //测试端口
        }
    }
}
```

25.7.7　实战硬件 SPI 控制 74HC595 驱动数码管显示模拟 RTC 时钟

AI8051U 实验箱通过 2 个 74HC595 芯片驱动 8 位数码管，使用 SPI 接口可以控制 74HC595 输出的数据内容。

74HC595 是一个 8 位串行输入、并行输出的位移缓存器。并行输出为三态输出。在 SCK 的上升沿，串行数据由 DATA 输入到内部的 8 位位移缓存器，并由 SQH 脚输出，而并行输出则是在 RCK 的上升沿将在 8 位位移缓存器的数据存入到 8 位并行输出缓存器。当串行数据输入端 EN 的控制信号为低使能时，并行输出端的输出值等于并行输出缓存器所存储的值。

参考电路图：



引脚说明：

符号	引脚	描述
A~H	第 15 脚，第 1-7 脚	8 位并行数据输出
GND	第 8 脚	地
SQH	第 9 脚	串行数据输出
/RESET	第 10 脚	主复位（低电平有效）
SCK	第 11 脚	数据输入时钟线
RCK	第 12 脚	输出存储器锁存时钟线
/EN	第 13 脚	输出有效（低电平有效）
DATA	第 14 脚	串行数据输入
VCC	第 16 脚	电源

使用方法：

74595 的数据端：

A~H: 八位并行输出端，可以直接控制数码管的 8 个段。

SQH: 级联输出端。将它接下一个 595 的 DATA 端。

DATA: 串行数据输入端,级联的话接上一级的 SQH。

74HC595 的控制端说明：

/RESET(10 脚): 低电平时将移位寄存器的数据清零。通常接到 VCC 防止数据清零。

SCK(11 脚)：上升沿时数据寄存器的数据移位。A->B->C-->D-->...-->H;下降沿移位寄存器数据不变。(脉冲宽度：5V 时，大于几十纳秒就行了。

RCK(12 脚)：上升沿时移位寄存器的数据进入数据存储寄存器，下降沿时存储寄存器数据不变。通常将 RCK 置为低电平，当移位结束后，在 RCK 端产生一个正脉冲(5V 时，大于几十纳秒就行了。我通常都选微秒级)，更新显示数据。

/EN(13 脚): 高电平时禁止输出(高阻态)。如果单片机的引脚不紧张，用一个引脚控制它，可以方便地产生闪烁和熄灭效果。比通过数据端移位控制要省时省力。

74HC595 的主要优点是具有数据存储寄存器，在移位的过程中，输出端的数据可以保持不变。这在串行速度慢的场合很有用处，数码管没有闪烁感。

74HC595 是串入并出带有锁存功能移位寄存器，它的使用方法很简单，在正常使用时 RCK 为低电平， /EN 为低电平。从 DATA 每输入一位数据，串行输入时钟 SCK 上升沿有效一次，直到八位数据输入完毕，输出时钟 RCK 上升沿有效一次，此时，输入的数据就被送到了输出端。

范例程序：

```
//测试工作频率为 40MHz

//main.c
```

```
//程序主函数

#include "config.h"                     //头文件中已包含 ai8051u.h 以及其他头文件

void main()
{
    SYS_Init();                         //系统初始化

    while (1)
    {
        if (f1ms)                       //判断 1ms 标志位
        {
            f1ms = 0;                   //清除 1ms 标志，并处理 1ms 事件

            ScanLed();                  //LED 数码管扫描
        }

        if (f1s)                        //判断 1s 标志位
        {
            f1s = 0;                    //清除 1s 标志，并处理 1s 事件

            RTC_Update();               //更新软件模拟的 RTC 时间
            RTC_Show();                 //在 LED 数码管上显示 RTC 时间
        }
    }
}

void SYS_Init(void)                     //系统初始化函数
{
    EAXFR = 1;                          //使能访问扩展 XFR
    WTST = 0x00;                        //设置最快速度访问程序代码
    CKCON = 0x00;                       //设置最快速度访问内部 XDATA

    P0M0 = 0x00; P0M1 = 0x00;           //初始化 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x00;           //初始化 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;           //初始化 P2 口为准双向口模式
    P3M0 = 0x34; P3M1 = 0x00;           //初始化 P3.2,P3.4,P3.5 为推挽输出，其他口为准双向口模式
    P4M0 = 0x00; P4M1 = 0x00;           //初始化 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;           //初始化 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;           //初始化 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;           //初始化 P7 口为准双向口模式

    Led_Init();                         //LED 数码管初始化
    RTC_Init();                         //模拟 RTC 时钟初始化
    Timer0_Init();                      //定时器 0 初始化,用于产生 1ms 和 500ms 的时间基准

    EA = 1;                             //使能全局中断
}
```

25.8　深大实验 -- 硬件 SPI 控制 74HC595 驱动数码管显示模拟 RTC 时钟

a)　实验介绍

1、熟悉硬件 SPI 控制 74HC595
2、了解使用 74HC595 控制 LED 数码管
3、熟悉如何管理多文件项目

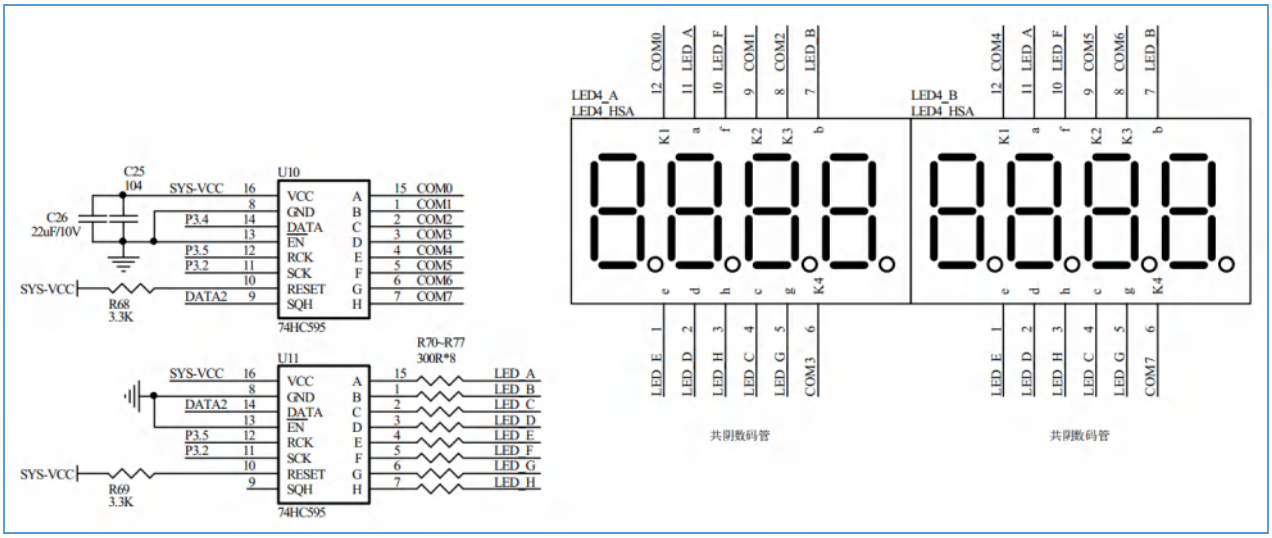
(1)　Keil 环境下多文件项目管理说明

当项目的功能比较复杂时，就需要在 Keil 中建立多文件项目，以方便分工合作、代码复用、模块化管理、增强可读性和可维护性。

- 比较好的建议是将项目功能模块化，不同模块的实现代码放在不同的.c 文件中。
- 一般建议是一个模块对应一个.c 程序文件和一个.h 头文件
- 模块的初始化函数以及相关的数据处理函数都在.c 文件中实现
- 与模块相关的全局变量也必须在.c 文件中进行定义，一定不能在.h 文件中定义变量
- 如果有其他模块需要使用本模块定义的变量或函数，则这些函数和变量都需要在.h 文件中声明。
- 特别提醒：在.h 文件中声明外部变量必须使用 **extern** 关键字，否则就变成变量定义了，这样会出现变量重复定义的错误
- 为防止头文件被多次包含而产生错误或者警告，在头文件中使用类似如下的条件编译组合语句，可避免在同一个.c 文件中对同一个.h 头文件进行多次包含：

```
#ifndef XXXX
#define XXXX
...
#endif
```

b)　原理图



74HC595 引脚说明		
符号	引脚	描述
A~H	第 15 脚，第 1-7 脚	8 位并行数据输出

GND	第 8 脚	地
SQH	第 9 脚	串行数据输出
/RESET	第 10 脚	主复位（低电平有效）
SCK	第 11 脚	数据输入时钟线
RCK	第 12 脚	输出存储器锁存时钟线
/EN	第 13 脚	输出有效（低电平有效）
DATA	第 14 脚	串行数据输入
VCC	第 16 脚	电源

c)实验程序代码

(1)　main.c -- 程序主函数

```
//main.c
//程序主函数

#include "config.h"        //头文件中已包含 ai8051u.h, ai_usb.h 以及其他头文件

void main()
{
    SYS_Init();          //系统初始化

    while (1)
    {
        if (f1ms)              //判断 1ms 标志位
        {
            f1ms = 0;      //清除 1ms 标志，并处理 1ms 事件
            ScanLed();     //LED 数码管扫描
        }

        if (f1s)                    //判断 1s 标志位
        {
            f1s = 0;               //清除 1s 标志，并处理 1s 事件

            RTC_Update();      //更新软件模拟的 RTC 时间
            RTC_Show();        //在 LED 数码管上显示 RTC 时间}
        }
    }
}

void SYS_Init(void)      //系统初始化函数
{
    EAXFR = 1;          //使能访问扩展 XFR
```

```
    WTST = 0x00;      //设置最快速度访问程序代码
    CKCON = 0x00;     //设置最快速度访问内部 XDATA

    P0M0 = 0x00; P0M1 = 0x00;       //初始化 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x00;       //初始化 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;       //初始化 P2 口为准双向口模式
    P3M0 = 0x34; P3M1 = 0x00;       //初始化 P3.2,P3.4,P3.5 为推挽输出，其他口为准双向口模式
    P4M0 = 0x00; P4M1 = 0x00;       //初始化 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;       //初始化 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;       //初始化 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;       //初始化 P7 口为准双向口模式

    Led_Init();           //LED 数码管初始化
    RTC_Init();           //模拟 RTC 时钟初始化
    Timer0_Init();        //定时器 0 初始化,用于产生 1ms 和 1s 的时间基准

    EA = 1;               //总中断允许位打开
}
```

(2)　led.c -- 扫描 LED 数码管

```
//led.c
//使用 SPI 驱动 74HC595 动态扫描 LED 数码管
//由于 LED 数码管是由 8 个 SEG 和 8 个 COM 组成的矩阵 LED
//所以 LED 数码管也需要扫描显示

#include "config.h"       //头文件中已包含 ai8051u.h 以及其他头文件

sbit    P_HC595_SER   = P3^4;       //74HC595 串行数据输入脚
sbit    P_HC595_RCLK  = P3^5;       //74HC595 并行数据输出时钟脚,上升沿有效
sbit    P_HC595_SRCLK = P3^2;       //74HC595 串行数据移入时钟脚,上升沿有效

BYTE    ShowIndex;              //LED 数码管扫描显示索引(8 个数码管轮流扫描显示)
BYTE    LedBuf[8];              //8 个 LED 数码管显示数据缓冲区

BYTE code LED_PATTERN[12] =         //定义 LED 数码管显示数据 0~9 的段码数据
{
    0x3F,   //0   //此表为共阴极数码管的段码数据
    0x06,   //1
    0x5B,   //2
    0x4F,   //3
    0x66,   //4
    0x6D,   //5
    0x7D,   //6
    0x07,   //7
```

```
    0x7F,     //8
    0x6F,     //9
    0x00,     //black
    0x40,     //-
};

void Shift_74HC595(BYTE dat)   //串行移入一个字节的数据到 74HC595
{
    P_HC595_RCLK = 0;         //将 SPI 片选拉低
    SPDAT = dat;              //将数据写入 SPI 数据寄存器
    while(SPIF == 0) ;        //判断传输完成标志，等待传输完成
    P_HC595_RCLK = 1;         //将 SPI 片选拉高
    SPIF = 1;                 //清 SPIF 标志
    WCOL = 1;                 //清 WCOL 标志
}

void Led_Init(void)       //LED 数码管初始化程序
{
    SSIG = 1;             //忽略 SS 引脚功能，使用 MSTR 确定器件是主机还是从机
    SPEN = 1;             //使能 SPI 功能
    DORD = 0;             //先发送/接收数据的高位（ MSB）
    MSTR = 1;             //设置主机模式
    CPOL = 1;             //SCLK 空闲时为高电平，SCLK 的前时钟沿为下降沿，后时钟沿为上升沿
    CPHA = 1;             //数据在 SCLK 前时钟沿驱动，后时钟沿采样
    SPCTL = (SPCTL & ~3) | 2;       //SPI 时钟频率选择, 0: 4T, 1: 8T,   2: 16T,   3: 2T
    SPI_S1 = 1;           //选择 SPI 脚位通道
    SPI_S0 = 1;           //00: P1.4 P1.5 P1.6 P1.7,     01: P2.4 P2.5 P2.6 P2.7,
                          //10: P4.0 P4.1 P4.2 P4.3,     11: P3.5 P3.4 P3.3 P3.2

    P_HC595_SRCLK = 0;        //初始化 74HC595 时钟线和数据线
    P_HC595_SER = 1;
    SPIF = 1;                 //清 SPIF 标志
    WCOL = 1;                 //清 WCOL 标志

    LedBuf[0] = LED_OFF;      //初始化 LED 数码管显示数据缓冲区,上电关闭显示
    LedBuf[1] = LED_OFF;
    LedBuf[2] = LED_OFF;
    LedBuf[3] = LED_OFF;
    LedBuf[4] = LED_OFF;
    LedBuf[5] = LED_OFF;
    LedBuf[6] = LED_OFF;
    LedBuf[7] = LED_OFF;

    ShowIndex = 0;            //初始化扫描显示索引
}
```

```
void ScanLed(void)                  //LED 数码管扫描显示程序,  (需每 1ms 执行一次!!!)
{
    Shift_74HC595(LedBuf[ShowIndex]);       //将当前扫描到的 LED 数码管的段码移入到 74HC595
    Shift_74HC595(~(1 << ShowIndex));       //将当前扫描到的 LED 数码管的 COM 数据移入到 74HC595

    if (++ShowIndex >= 8)           //扫描索引值递增
        ShowIndex = 0;              //扫描完一轮后索引值归零
}

void SetLed (BYTE nIndex,           //设置 LED 显示函数,nIndex:设置目标数码管,0~7 分别对应 8 个数码
             BYTE nNum,             //nNum:设置显示的数据,有效值 0~9, ,-,其他值则关闭显示
             BOOL bShowPoint        //bShowPoint:设置是否显示小数点
            )
{
    if (nIndex < 8)                 //目标数码管的有效值为 0~7
    {
        if (nNum < 12)              //判断显示的值是否为 0~9, ,-
            LedBuf[nIndex] = LED_PATTERN[nNum];
                                    //将相应数据的段码赋值到 LED 数码管显示数据缓冲区
        else
            LedBuf[nIndex] = LED_OFF;       //如果是其他值则关闭显示

        if (bShowPoint)             //判断是否需要显示小数点
            LedBuf[nIndex] |= LED_POINT;  //如果需要显示,则增加小数点的段码数据
    }
}
```

(3)　timer.c -- 定时器初始化以及定时器中断服务程序

```
//timer.c
//定时器初始化以及定时器中断服务程序
//定时器 0:1ms 定时,产生 1ms 和 1s 的标志位

#include "config.h"       //头文件中已包含 ai8051u.h, ai_usb.h 以及其他头文件

bit     f1ms;             //1ms 标志位
bit     f1s;              //1s 标志位
int     cnt1000;          //1000 次计数器,用于产生 1000ms 的时间

void Timer0_Init(void)          //1 毫秒@40.000MHz
{
    AUXR |= 0x80;               //定时器 0 时钟 1T 模式
    TMOD &= 0xF0;               //设置定时器 0 为模式 0
    TL0 = 0xC0;                 //设置定时器 0 初始值
    TH0 = 0x63;                 //设置定时器 0 初始值
```

```
    TF0 = 0;                //清除 TF0 标志
    TR0 = 1;                //定时器 0 开始计时
    ET0 = 1;                //使能定时器 0 中断

    cnt1000 = 0;            //初始化 1000 次计数器
    f1ms = 0;               //初始化 1ms 标志位
    f1s = 0;                //初始化 1s 标志位
}

void Timer0_Isr(void) interrupt 1       //定时器 0 中断服务程序
{
    f1ms = 1;                   //设置 1ms 标志位

    if (cnt1000++ >= 1000)      //判断 1000 次计数是否已满
    {
        cnt1000 = 0;            //清除计数值,开始开始计数
        f1s = 1;                //设置 1000ms 标志位
    }
}
```

(4)　rtc.c -- RTC 显示控制程序

```
//rtc.c
//软件模拟 RTC 程序
//包括 RTC 时间更新控制

#include "config.h"     //头文件中已包含 ai8051u.h 以及其他头文件

BYTE bHour;             //RTC 小时计数值
BYTE bMinute;           //RTC 分钟计数值
BYTE bSecond;           //RTC 秒钟计数值

void RTC_Init()         //RTC 初始化函数
{
    bHour = 12;         //初始化时间为 12:30:00
    bMinute = 30;
    bSecond = 0;
}

void RTC_Update()       //RTC 时间更新函数,  (需每 1s 执行一次!!!)
{
    if (++bSecond >= 60)        //秒钟递增 1,并判断秒钟是否记满 60
    {
        bSecond = 0;            //秒钟归零
        if (++bMinute >= 60)    //分钟递增 1,并判断分钟是否记满 60
```

```
        {
            bMinute = 0;        //分钟归零
            if (++bHour >= 24)      //小时递增 1,并判断小时是否记满 24
            {
                bHour = 0;          //小时归零
            }
        }
    }
}

void RTC_Show()     //RTC 显示控制程序,不同的模式显示方式不一样
{
    SetLed(0, (BYTE)(bHour / 10), 0);           //在数码 0 上显示小时的十位
    SetLed(1, (BYTE)(bHour % 10), 0);           //在数码 1 上显示小时的个位
    SetLed(2, LED_, 0);                         //在数码 2 上显示"-"
    SetLed(3, (BYTE)(bMinute / 10), 0);         //在数码 3 上显示分钟的十位
    SetLed(4, (BYTE)(bMinute % 10), 0);         //在数码 4 上显示分钟的个位
    SetLed(5, LED_, 0);                         //在数码 5 上显示"-"
    SetLed(6, (BYTE)(bSecond / 10), 0);         //在数码 6 上显示秒钟的十位
    SetLed(7, (BYTE)(bSecond % 10), 0);         //在数码 7 上显示秒钟的个位
}
```

(5)　config.h -- 项目配置的头文件

```
#ifndef __CONFIG_H__            //防止头文件被重复包含
#define __CONFIG_H__

#define  HIRC               40000000UL
#define  FOSC               40000000UL
#define  SYSCLK             FOSC
#define  MAIN_Fosc          FOSC

#include <ai8051u.h>            //包含外部头文件
#include <stdio.h>
#include <intrins.h>

#include "def.h"                //包含项目头文件
#include "timer.h"
#include "led.h"
#include "rtc.h"

void SYS_Init(void);            //函数声明

#endif
```

(6) led.h -- 项目配置文件的头文件

```
#ifndef __LED_H__                //防止头文件被重复包含
#define __LED_H__

#define   LED_        0x0B       //定义 LED 数码管中杠的段码位置
#define   LED_OFF     0x00       //定义关闭 LED 数码管的段码数据,
                                 //LED 数码管为共阴极,所以 SEG 线全部输出 0 即可关闭一个 8 段数码管
#define   LED_POINT   0x80       //定义 LED 数码管小数点的段码数据

extern BYTE LedBuf[8];

void Led_Init(void);             //函数声明
void ScanLed(void);              //函数声明
void SetLed(BYTE nIndex, BYTE nNum, BOOL bShowPoint);

#endif
```

(7) timer.h -- 项目配置文件的头文件

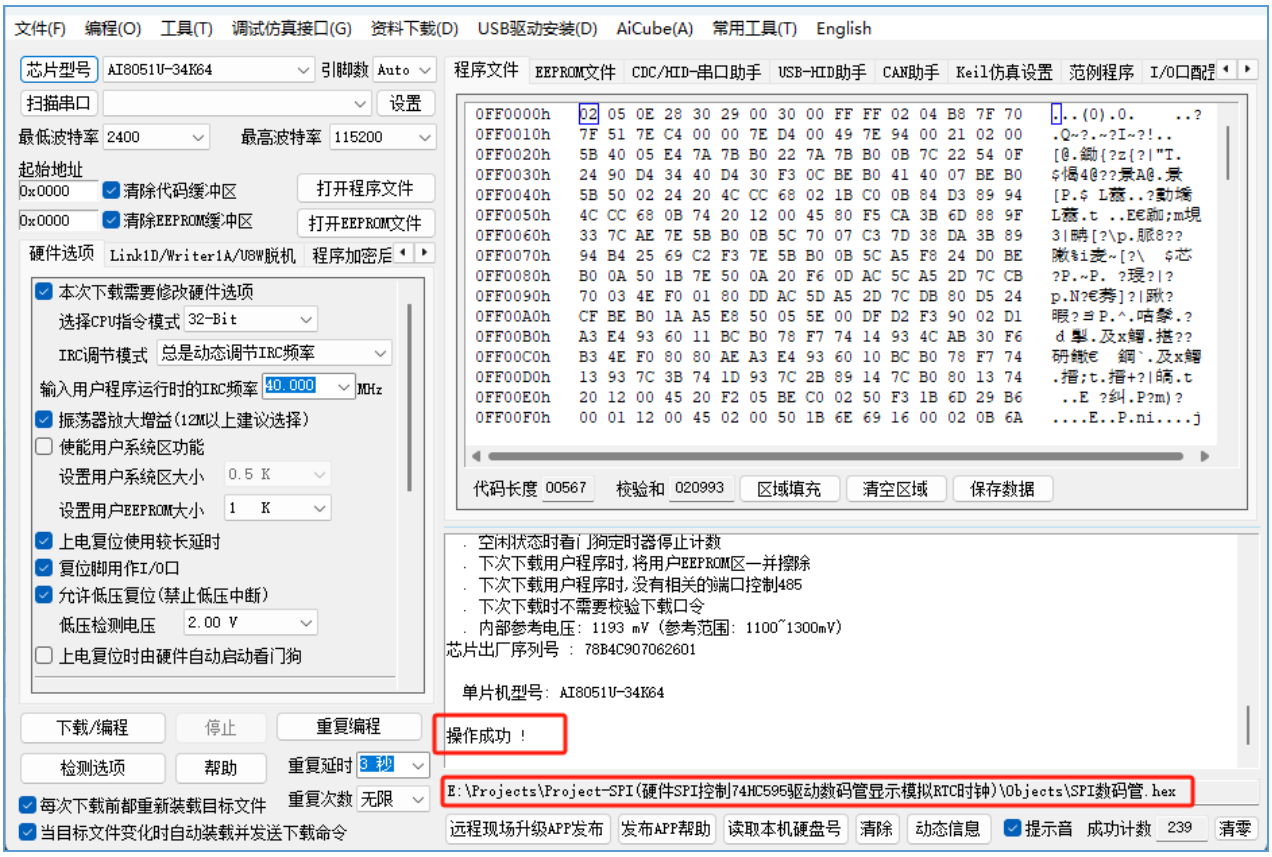
```
#ifndef __TIMER_H__          //防止头文件被重复包含
#define __TIMER_H__

void Timer0_Init(void);      //函数声明

extern bit f1ms;             //外部变量声明
extern bit f1s;              //外部变量声明

#endif
```

(8) rtc.h -- 项目配置文件的头文件

```
#ifndef __RTC_H__            //防止头文件被重复包含
#define __RTC_H__

void RTC_Init();             //函数声明
void RTC_Update();           //函数声明
void RTC_Show();             //函数声明

#endif
```

d)　程序下载

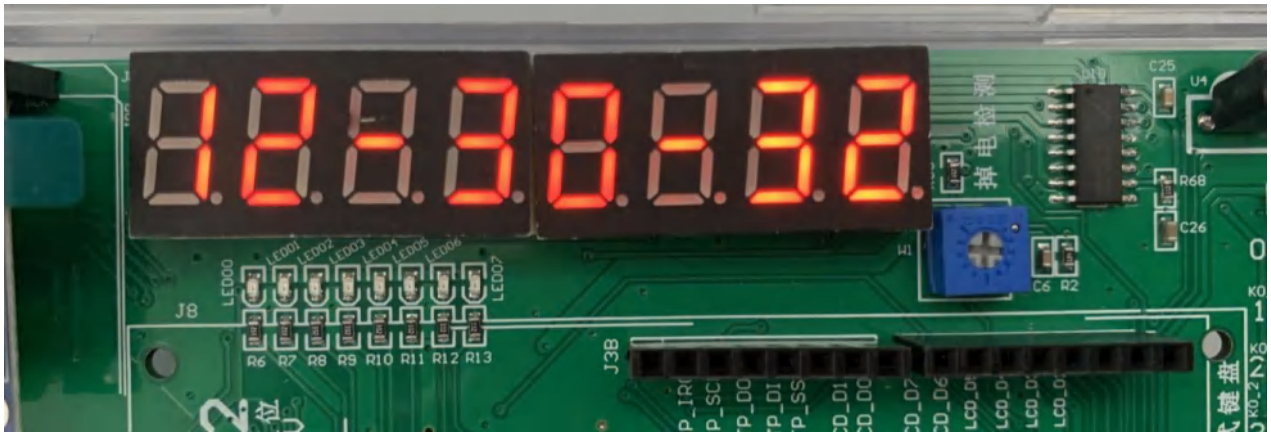
录入代码，保存，编译。连接好 Ai8051U 实验箱，运行 AIapp-ISP 软件系统，按步骤打开并下载程序“SPI 数码管.hex”，如下图：



下载/烧录成功！

e)实验现象

现在，观察 Ai8051U 实验箱，LED 显示 RTC 设置的时间：



25.9 大型综合实战--大学课程设计：硬件 SPI 读写串行 Flash

25.9.1 实验介绍

1、本实验项目主要目的：

- 掌握串口通讯
- 硬件 SPI 读写串行 Flash

2、掌握 Ai8051U 实验箱原理图中的串口通信电路、硬件 SPI 电路

3、继续熟悉 AIapp-ISP 系统软件中串口助手工具

4、继续熟悉如何管理多文件项目

详细实验内容及实验步骤，请查看：

https://www.stcaimcu.com/forum.php?mod=redirect&goto=findpost&ptid=15176&pid=170525

25.10 深大实验--硬件 SPI 驱动 OLED 显示图片、图形、字符

25.10.1 实验介绍

1、本实验项目主要目的：
- 硬件 SPI 驱动 OLED 显示
- 如何实现显示图片、6*8 点阵字符、8*16 点阵字符、16*16 点阵汉字
- 如何画直线、方框、圆

2、掌握 Ai8051U 实验箱原理图中 SPI 驱动 OLED 电路
3、了解图片取模的原理，并熟练使用 AIapp-ISP 系统软件中图片取模工具
4、了解字库生成原理，并熟练使用 AIapp-ISP 系统软件中字库生成工具
5、继续熟悉如何管理多文件项目

a) Aiapp-ISP 系统软件图片取模工具的使用

打开 AIapp-ISP 系统软件：
- 点击菜单栏“工具”→“图片取模工具”，打开“图片取模工具”浮窗
- 点击“打开图片”按钮，选择准备要取模的图片
- 点击“打开”按钮

如下图：

此时，在打开的“图片取模工具”浮窗，我们看到被选择的图片显示在原始图片区和转换后的图片区：

- 扫描模式：选择“垂直扫描，从左到右，从上到下”
- 存储顺序：选择“高位在前”
- 数据格式：选择“C 表格格式”
- 勾选“图片反色”
- 设置：宽度：128；高度：64；输出格式：1 位单色
- 点击“开始转换”

就会生成相应的点阵图片 C 代码，如下图：

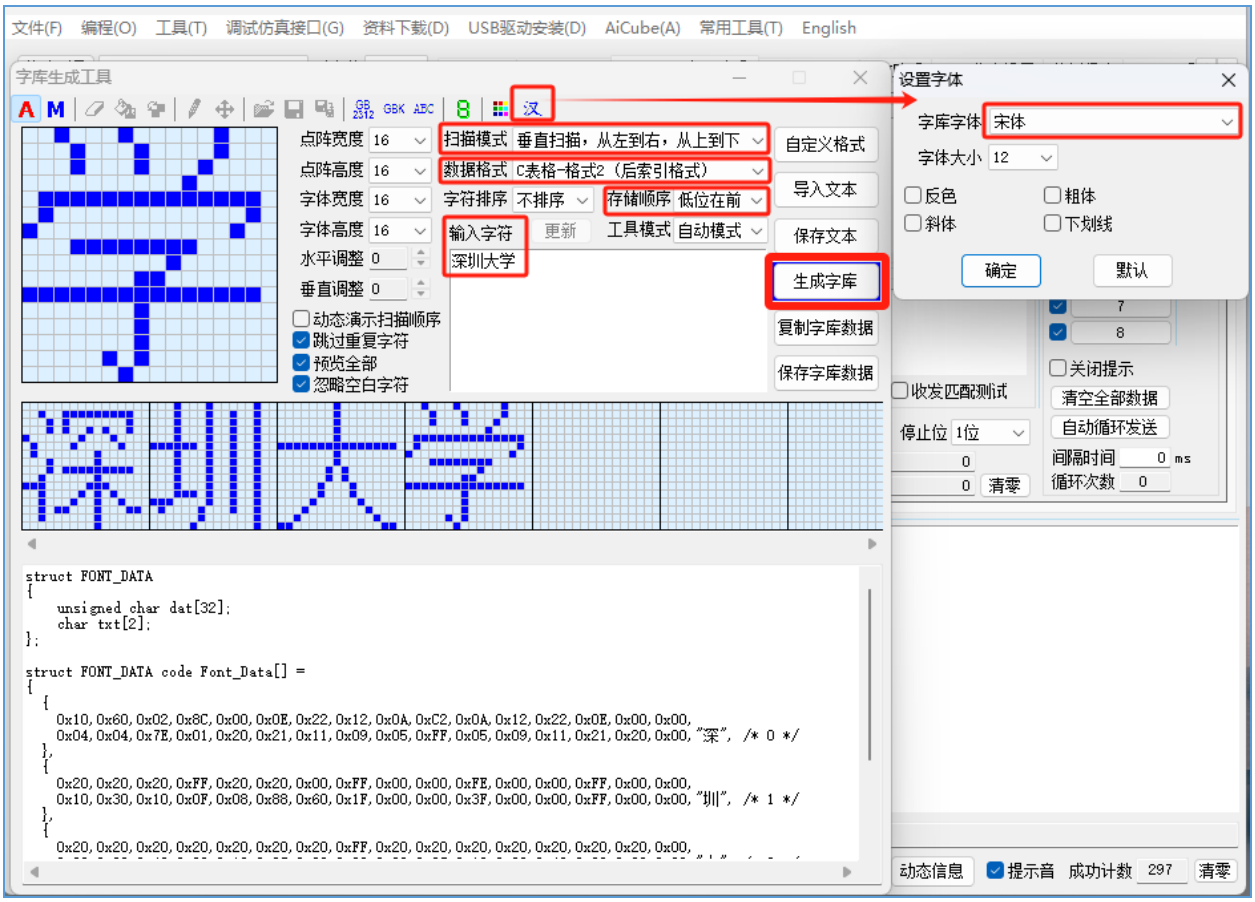
将点阵图片生成的 C 代码数据拷贝出来，本例是将此段代码放在“pic.h”中，形成表格数据，方便程序加载并在 OLED 上显示出来。

b) Aiapp-ISP 系统软件字库生成工具的使用

打开 AIapp-ISP 系统软件，点击菜单栏“工具”→“字库生成工具”，打开“字库生成工具”浮窗：

- 在“输入字符”区输入想要取模的文字，本例输入“深圳大学”
- 点击第一行的工具栏上的“汉”按钮，打开“设置字体”浮窗，字库字体选择“宋体”
- 扫描模式：选择“垂直扫描，从左到右，从上到下”
- 数据格式：选择“C 表格-格式 2（后索引模式）”
- 存储顺序：选择“低位在前”
- 点击“生成字库”

就会生成相应的点阵字库代码，如下图：



将点阵字库生成的 C 代码拷贝出来，本例是将此段代码放在“font.h”中，形成表格数据，方便程序查询并提取字库。

c) 使用 Aiapp-ISP 系统软件图片取模工具生成 ASCII 的点阵字符 C 代码

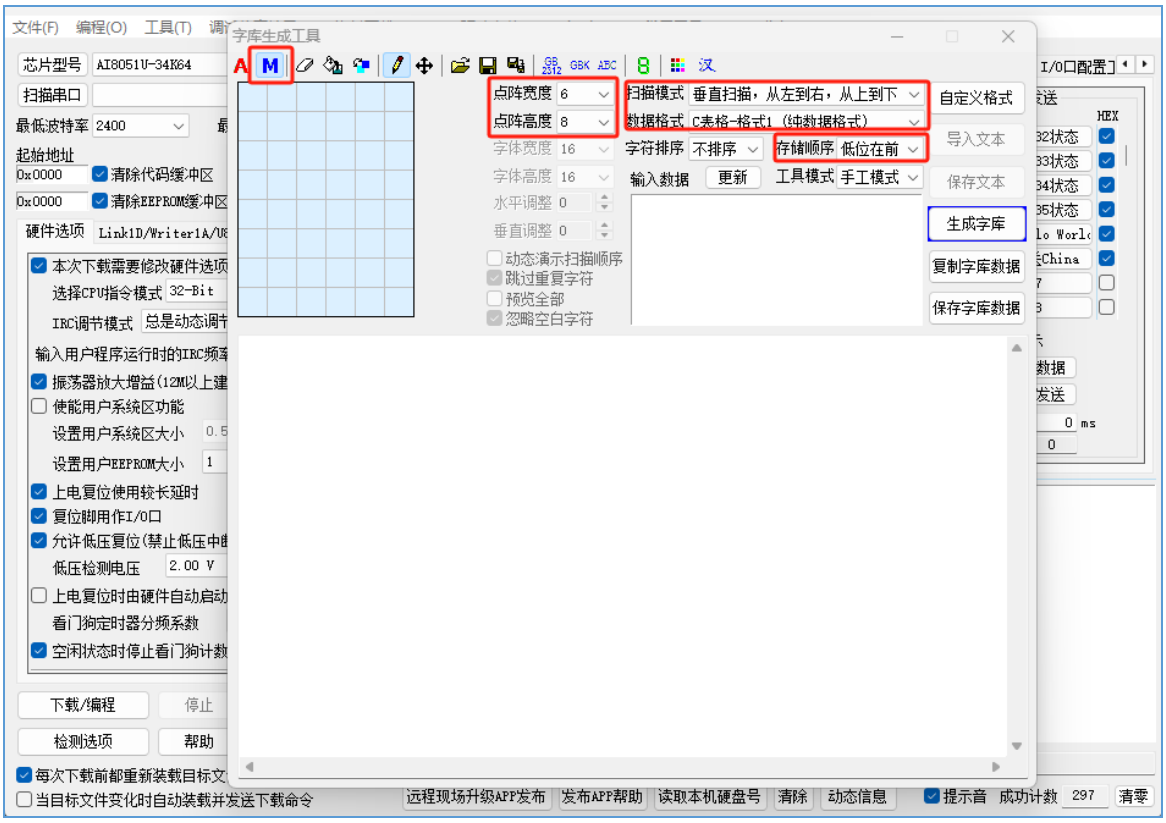
在 AIapp-ISP 系统软件的字库生成工具中，我们除了可以像上述操作步骤生成标准点阵字符 C 代码，还可以自己手动描点生成自己想要的点阵字符 C 代码。

打开 AIapp-ISP 系统软件，点击菜单栏“工具”→“字库生成工具”，打开“字库生成工具”浮窗。

先进行相应的设置：

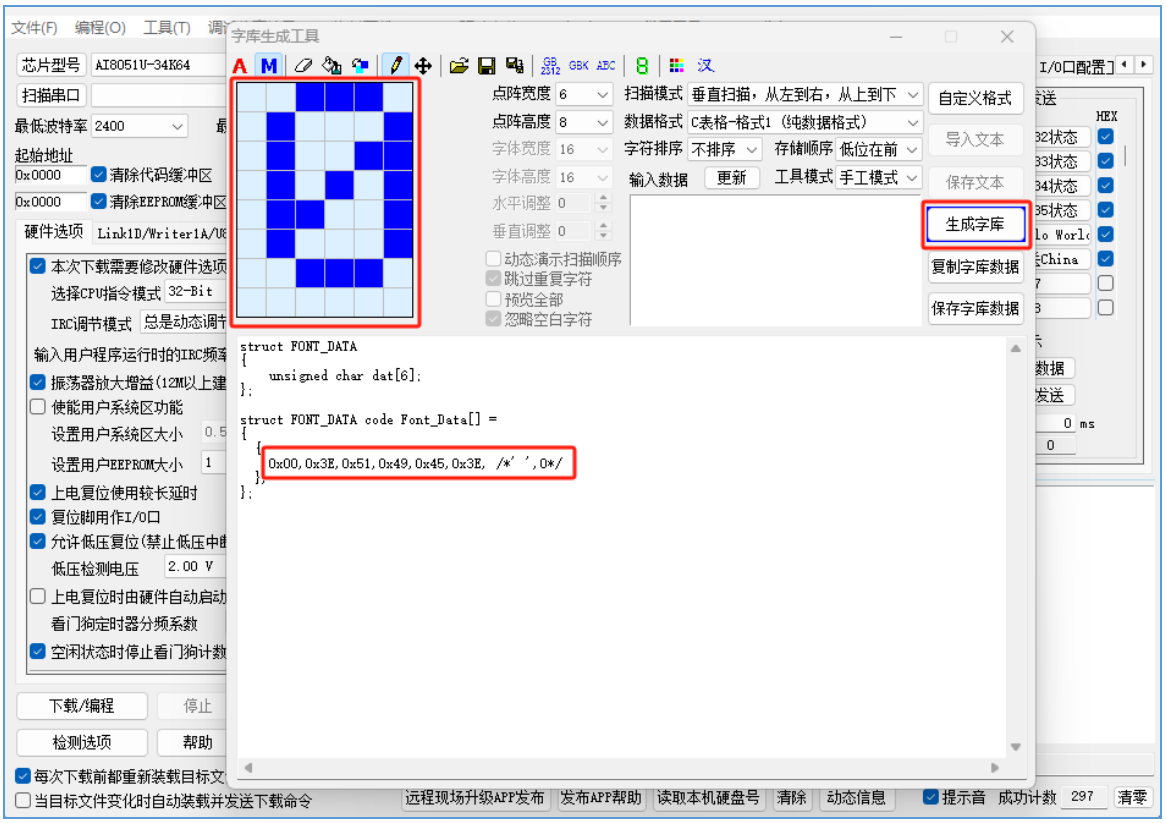
- 点击“字库生成工具”浮窗第一行的“M”按钮，清空字符点阵区域
- 确定点阵宽度：6；点阵高度：8（本例中有 6*8 的点阵 ASCII 字符 C 代码数据和 8*16 的点阵 ASCII 字符代码数据）
- 扫描模式：选择“垂直扫描，从左到右，从上到下”
- 数据格式：选择“C 表格-格式 1（纯数据格式）”
- 存储顺序：选择“低位在前”

如下图：



现在手动用鼠标在窗口左上角点阵描图区域，进行描图。

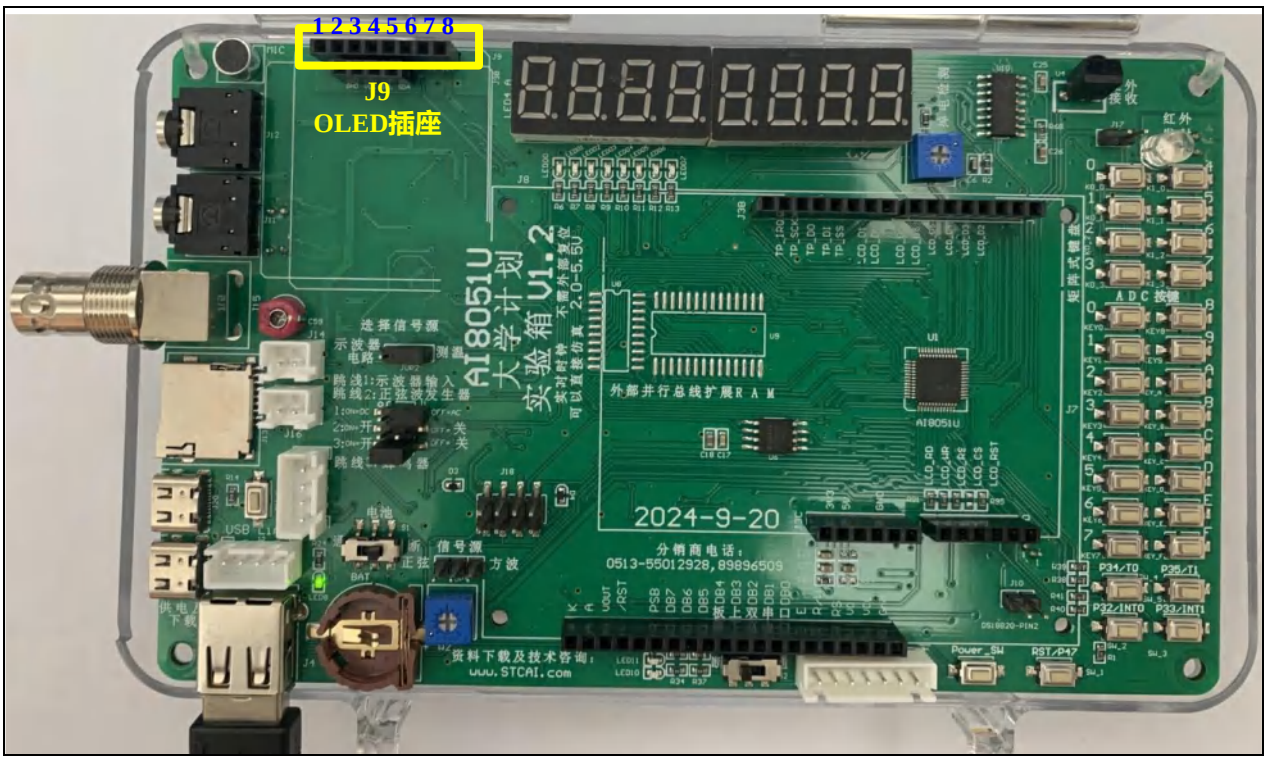
以 ASCII 码的“0”为例，进行如下图中的描绘，再点击“生成字库”按钮，即可生成相应的字库 C 代码数据。以此类推，生成自己想要的全部 ASCII 码的点阵字库 C 代码数据。



将点阵字库生成的 C 代码拷贝出来，本例是将此段代码放在“ascii.h”中，形成表格数据，方便程序查询并提取字库。

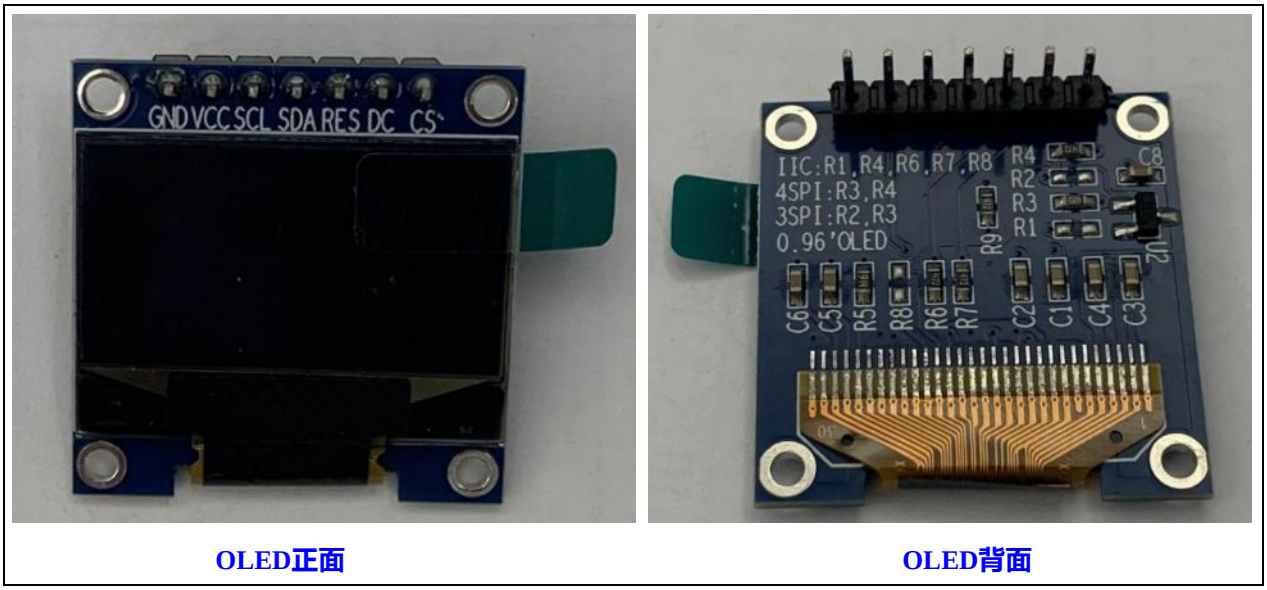
d) 认识 Ai8051U 实验箱和 OLED 屏

Ai8051U 实验箱：



J9：OLED 插座。Ai8051U 实验箱上的 J9 插座可插 7 线或 8 线的 OLED 屏，本实验使用的 OLED 屏是 7 线，第 8 脚浮空。详见 Ai8051U 实验箱中 J9 原理图。

OLED 屏



OLED正面　　OLED背面

e) Keil 环境下多文件项目管理说明

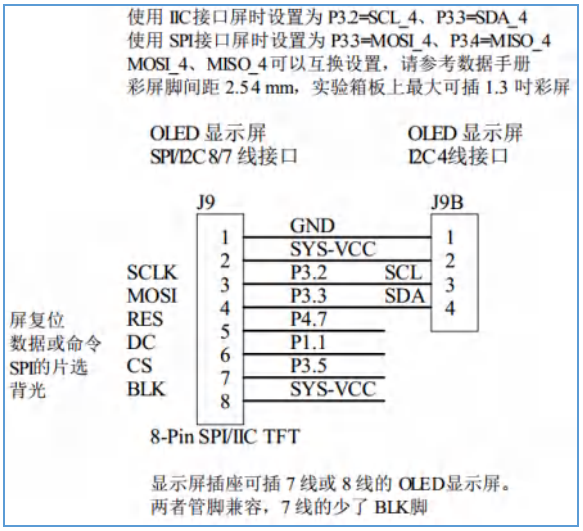
当项目的功能比较复杂时，就需要在 Keil 中建立多文件项目，以**方便分工合作、代码复用、模块化管理、增强可读性和可维护性**。

- 比较好的建议是将项目功能模块化，不同模块的实现代码放在不同的.c 文件中。

- 一般建议是一个模块对应一个.c 程序文件和一个.h 头文件
- 模块的初始化函数以及相关的数据处理函数都在.c 文件中实现
- 与模块相关的全局变量也必须在.c 文件中进行定义，一定不能在.h 文件中定义变量
- 如果有其他模块需要使用本模块定义的变量或函数，则这些函数和变量都需要在.h 文件中声明。
- 特别提醒：在.h 文件中声明外部变量必须使用 **extern** 关键字，否则就变成变量定义了，这样会出现变量重复定义的错误
- 为防止头文件被多次包含而产生错误或者警告，在头文件中使用类似如下的条件编译组合语句，可避免在同一个.c 文件中对同一个.h 头文件进行多次包含：

```
#ifndef XXXX
#define XXXX
...
#endif
```

25.10.2　原理图



详细实验代码及实验步骤，请查看：
https://www.stcaimcu.com/forum.php?mod=redirect&goto=findpost&ptid=15176&pid=171004

26 高速 SPI（HSSPI）

产品线	HSSPI
Ai8051U 系列	●

Ai8051U 系列单片机为 SPI 提供了高速模式（HSSPI）。高速 SPI 是以普通 SPI 为基础，增加了高速模式。

当系统运行在较低工作频率时，高速 SPI 可工作在高达 144M 的频率下。从而可以达到降低内核功耗，提升外设性能的目的

26.1 相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
HSSPI_CFG	高速 SPI 配置寄存器	7EFBF8H	SS_HLD[3:0]				SS_SETUP[3:0]				0011,0011
HSSPI_CFG2	高速 SPI 配置寄存器 2	7EFBF9H	-	IOSW	HSSPIEN	FIFOEN	SS_DACT[3:0]				x000,0011
HSSPI_STA	高速 SPI 状态寄存器	7EFBFAH	-	-	-	-	TXFULL	TXEMPT	RXFULL	RXEMPT	xxxx,0000
HSSPI_PSCR	高速 SPI 时钟分配器	7EFBFBH	DIV[7:0]								0000,0000

26.1.1 高速 SPI 配置寄存器（HSSPI_CFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
HSSPI_CFG	7EFBF8H	SS_HLD[3:0]				SS_SETUP[3:0]			

SS_HLD[3:0]：高速模式时 SS 控制信号的 HOLD 时间

SS_SETUP[3:0]：高速模式时 SS 控制信号的 SETUP 时间

26.1.2 高速 SPI 配置寄存器 2（HSSPI_CFG2）（交换 MISO 和 MOSI）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
HSSPI_CFG2	7EFBF9H	-	IOSW	HSSPIEN	FIFOEN	SS_DACT[3:0]			

IOSW：交换 MOSI 和 MISO 脚位（普通 SPI 模式和高速 SPI 模式通用）

0：不交换，维持上电默认脚位。

1：交换 MOSI 和 MISO 的脚位。

HSSPIEN：高速 SPI 使能位

0：关闭高速模式。

1：使能高速模式。

当 SPI 速度为系统时钟/2（SPCTL.SPR＝3）时，必须设置使能高速模式

SPI+DMA 读取数据时，如果开启了 FIFO，也必须开启高速 SPI 模式

FIFOEN：高速 SPI 的 FIFO 模式使能位

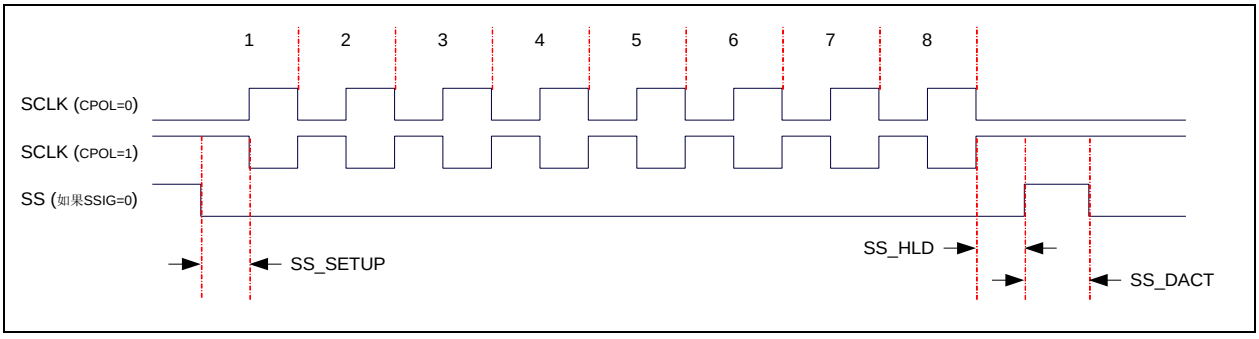
0：关闭 FIFO 模式。

1：使能 FIFO 模式。

SPI 发送和接收的 FIFO 深度均为 4 字节。设置 FIFOEN 可以减少 SPI 数据收发间隔时间，提高 SPI 收发速度。FIFOEN 只有在使用 SPI+DMA 传输时才可以使能，使用普通 SPI 收发时需要关闭 FIFOEN 才能收到 SPIF 标志。

SS_DACT[3:0]：高速模式时 SS 控制信号的 DEACTIVE 时间

注：SS_HLD、SS_SETUP 和 SS_DACT 所设置的值只有在 SPI 主机模式的 DMA 才有意义，只有 SPI 在主机模式时，执行 DMA 数据传输时才需要硬件自动输出 SS 控制信号。当 SPI 速度为系统时钟/2（SPCTL.SPR＝3）时执行 DMA，SS_HLD、SS_SETUP 和 SS_DACT 都必须设置为大于 2 的值。



26.1.3 高速 SPI 状态寄存器（HSSPI_STA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
HSSPI_STA	7EFBFAH	-	-	-	-	TXFULL	TXEMPT	RXFULL	RXEMPT

TXFULL：发送数据 FIFO 满标志（只读）

TXEMPT：发送数据 FIFO 空标志（只读）

RXFULL：接收数据 FIFO 满标志（只读）

RXEMPT：接收数据 FIFO 空标志（只读）

26.1.4 高速 SPI 时钟分频器（HSSPI_PSCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
HSSPI_PSCR	7EFBFBH	DIV[7:0]							

DIV[7:0]：高速 SPI 时钟分频器

高速 SPI 工作时钟＝高速 SPI 输入时钟 / (HSSPI_PSCR[7:0] + 1)

26.2　范例程序

26.2.1　使能 SPI 的高速模式

```
//测试工作频率为 12MHz

#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define    FOSC     12000000UL

#define    HSCK_MCLK        0
#define    HSCK_PLL         1
#define    HSCK_SEL         HSCK_PLL

#define    PLL_96M          0
#define    PLL_144M         1
#define    PLL_SEL          PLL_96M

#define    CKMS             0x80
#define    HSIOCK           0x40
#define    MCK2SEL_MSK      0x0c
#define    MCK2SEL_SEL1     0x00
#define    MCK2SEL_PLL      0x04
#define    MCK2SEL_PLLD2    0x08
#define    MCK2SEL_IRC48    0x0c
#define    MCKSEL_MSK       0x03
#define    MCKSEL_HIRC      0x00
#define    MCKSEL_XOSC      0x01
#define    MCKSEL_X32K      0x02
#define    MCKSEL_IRC32K    0x03

#define    ENCKM            0x80
#define    PCKI_MSK         0x60
#define    PCKI_D1          0x00
#define    PCKI_D2          0x20
#define    PCKI_D4          0x40
#define    PCKI_D8          0x60

void delay()
{
    int    i;

    for    (i=0; i<100; i++);
}

void main()
{
    EAXFR = 1;                                          //允许访问扩展的特殊寄存器，XFR
                                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                      //允许访问扩展的特殊寄存器，XFR
                                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                           //设置取程序代码等待时间，
                                                        //赋值为 0 表示不等待，程序以最快速度运行
```

```
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    //选择 PLL 输出时钟
#if  (PLL_SEL == PLL_96M)
    CLKSEL &= ~CKMS;                            //选择 PLL 的 96M 作为 PLL 的输出时钟
#elif (PLL_SEL == PLL_144M)
    CLKSEL |= CKMS;                             //选择 PLL 的 144M 作为 PLL 的输出时钟
#else
    CLKSEL &= ~CKMS;                            //默认选择 PLL 的 96M 作为 PLL 的输出时钟
#endif

    //选择 PLL 输入时钟分频,保证输入时钟为 12M
    USBCLK &= ~PCKI_MSK;
#if  (FOSC == 12000000UL)
    USBCLK |= PCKI_D1;                          //PLL 输入时钟 1 分频
#elif (FOSC == 24000000UL)
    USBCLK |= PCKI_D2;                          //PLL 输入时钟 2 分频
#elif (FOSC == 48000000UL)
    USBCLK |= PCKI_D4;                          //PLL 输入时钟 4 分频
#elif (FOSC == 96000000UL)
    USBCLK |= PCKI_D8;                          //PLL 输入时钟 8 分频
#else
    USBCLK |= PCKI_D1;                          //默认 PLL 输入时钟 1 分频
#endif

    //启动 PLL
    USBCLK |= ENCKM;                            //使能 PLL 倍频

    delay();                                    //等待 PLL 锁频

    //选择 HSPWM/HSSPI 时钟
#if  (HSCK_SEL == HSCK_MCLK)
    CLKSEL &= ~HSIOCK;                          //HSPWM/HSSPI 选择主时钟为时钟源
#elif (HSCK_SEL == HSCK_PLL)
    CLKSEL |= HSIOCK;                           //HSPWM/HSSPI 选择 PLL 输出时钟为时钟源
#else
    CLKSEL &= ~HSIOCK;                          //默认 HSPWM/HSSPI 选择主时钟为时钟源
#endif

    HSCLKDIV = 0;                               //HSPWM/HSSPI 时钟源不分频

    SPCTL = 0xd0;                               //设置 SPI 为主机模式,速度为 SPI 时钟/4
    HSSPI_CFG2 |= 0x20;                         //使能 SPI 高速模式

    P1M0 = 0x28;
    P1M1 = 0x00;

    while (1)
    {
        SPSTAT = 0xc0;
        SPDAT = 0x5a;
        while (!SPIF);
    }
}
```

27 四线 SPI（QSPI）

产品线	QSPI
Ai8051U 系列	●

Ai8051U 系列单片机集成了 QSPI 控制器。QSPI 是一种专用的通信接口，连接单线、双线或四线的 SPI Flash 存储介质。该接口可以在以下两种模式下工作：

- 间接模式：使用 QSPI 寄存器执行全部操作
- 状态轮询模式：周期性读取外部 Flash 状态寄存器，而且标志位置 1 时会产生中断（如擦除或烧写完成，会产生中断）

27.1 QSPI 功能脚切换

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW4	BFH	-		-		-		QSPI_S[1:0]	

QSPI_S[1:0]：QSPI 功能脚选择位

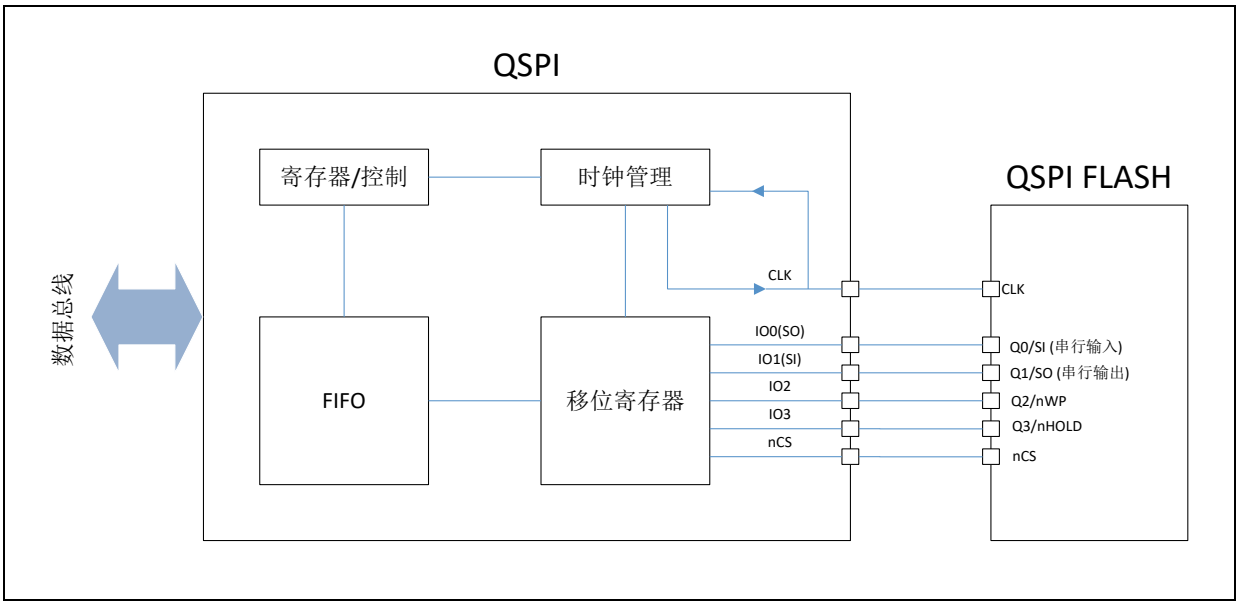
QSPI_S[1:0]	QSPINCS	QSPIIO0	QSPIIO1	QSPIIO2	QSPIIO3	QSPICLK
00	P1.4	P1.5	P1.6	P1.3	P1.2	P1.7
01	P4.0	P4.1	P4.2	P5.2	P5.3	P4.3
10	P4.7	P2.5	P2.6	P4.6	P4.5	P2.7
11	-	-	-	-	-	-

27.2 QSPI 主要特性

- 两种功能模式：间接模式和 状态轮询模式
- 支持 SDR 模式
- 针对间接模式，完全可编程操作码
- 针对间接模式，完全可编程帧格式
- 集成 32 字节的 FIFO，用于发送和接收
- 支持 8 位数据访问
- 在达到 FIFO 阈值、超时、操作完成以及发生访问错误时产生中断

27.3 QSPI 功能说明

27.3.1 QSPI 框图



27.3.2 QSPI 应用注意事项

1、 当使能 QSPI 功能后，QSPI 相关的 6 个 I/O 口均会由 QSPI 模块进行托管，用户程序无法再对这些 I/O 进行输出控制，即使 QSPI 工作在单线模式（只使用 IO0 和 IO1 进行数据传输），用户程序也无法操作 IO2 和 IO3。

2、 QSPI 的 nCS 脚的有效信号是由硬件自动控制的，不能用普通的 I/O 口来代替 nCS 进行控制。

3、 一组硬件 QSPI 口只能用于控制一个 QSPI 设备，不能用类似普通 SPI 的方式采用多个普通 I/O 口来当作 nCS 片选多个设备。但可以使用分时复用的方法将整组 QSPI 切换到不同的 I/O 管脚，来控制多个 QSPI 设备。

QSPI_S[1:0]	QSPINCS	QSPIIO0	QSPIIO1	QSPIIO2	QSPIIO3	QSPICLK
00	P1.4	P1.5	P1.6	P1.3	P1.2	P1.7
01	P4.0	P4.1	P4.2	P5.2	P5.3	P4.3
10	P4.7	P2.5	P2.6	P4.6	P4.5	P2.7
11	-	-	-	-	-	-

4、 Ai8051U 的 QSPI 只有主机模式，没有从机模式。

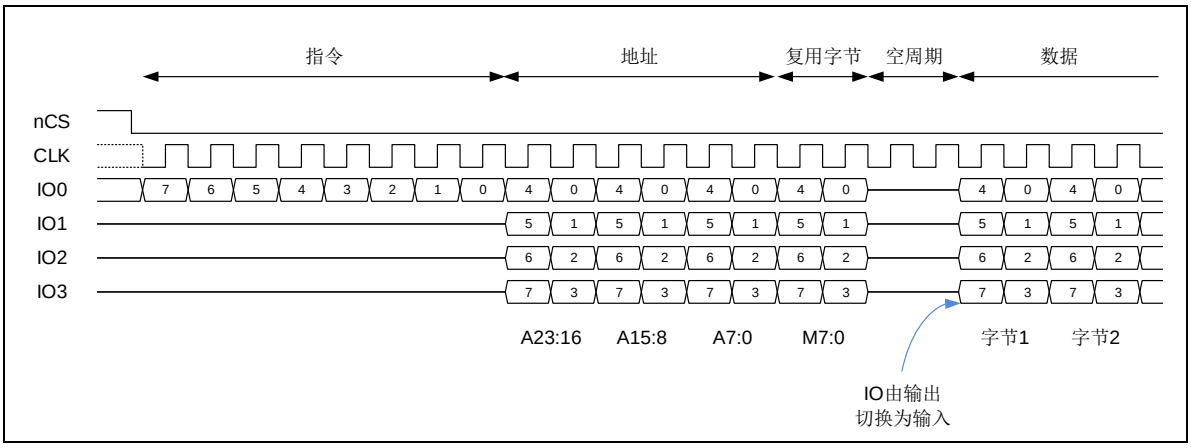
5、 Ai8051U 的 QSPI 的工作模式可配置为兼容普通 SPI 的工作模式 0（空闲是 CLK 为低电平）和兼容普通 SPI 的工作模式 3（空闲是 CLK 为高电平），使用 QSPI_DCR1 寄存器中的 CKMODE 进行配置。

27.3.3 QSPI 命令序列

QSPI 通过命令与 Flash 通信每条命令包括指令、地址、交替字节、空指令和数据这五个阶段。任一阶段均可跳过，但至少要包含指令、地址、交替字节或数据阶段之一。

nCS 在每条指令开始前下降到低电平，在每条指令完成后再次上升为高电平。

（四线模式下的读命令示例）



指令阶段

这一阶段，将在 QSPI_CCR1 寄存器的 INSTRUCTION 字段中配置的一条 8 位指令发送到 Flash，指定待执行操作的类型。

尽管大多数 Flash 从 IO0/SO 信号（单线 SPI 模式）只能以一次 1 位的方式接收指令，但指令阶段可选择一次发送 2 位（在双线 SPI 模式中通过 IO0 / IO1）或一次发送 4 位（在四线 SPI 模式中通过 IO0 / IO1 / IO2 / IO3）。这可通过 QSPI_CCR2 寄存器中的 IMODE[1:0]字段进行配置。若 IMODE = 00，则跳过指令阶段，命令序列从地址阶段（如果存在）开始。

地址阶段

在地址阶段，将 1～4 字节地址发送到 Flash，指示操作地址。待发送的地址字节数在 QSPI_CCR2 寄存器的 ADSIZE[1:0]字段中进行配置。在间接模式和自动轮询模式下，待发送的地址字节在 QSPI_AR1～QSPI_AR4 寄存器的 ADDRESS[31:0]中指定。

地址阶段可一次发送 1 位（在单线 SPI 模式中通过 SO）、2 位（在双线 SPI 模式中通过 IO0 / IO1）或 4 位（在四线 SPI 模式中通过 IO0 / IO1 / IO2 / IO3）。这可通过 QSPI_CCR2 寄存器中的 ADMODE[1:0]字段进行配置。若 ADMODE = 00，则跳过地址阶段，命令序列直接进入下一阶段（如果存在）。

交替字节阶段

在交替字节阶段，将 1 字节发送到 Flash，一般用于控制操作模式。待发送的交替字节数在 QSPI_CCR3 寄存器的 ABSIZE[1:0]字段中进行配置。待发送的字节在 QSPI_ABR 寄存器中指定。

交替字节阶段可一次发送 1 位（在单线 SPI 模式中通过 S0）、2 位（在双线 SPI 模式中通过 IO0 / IO1）或 4 位（在四线 SPI 模式中通过 IO0 / IO1 / IO2 / IO3）。这可通过 QSPI_CCR2 寄存器中的 ABMODE[1:0]字段进行配置。若 ABMODE = 00,则跳过交替字节阶段，命令序列直接进入

下一阶段（如果存在）。

交替字节阶段存在仅需发送单个半字节而不是一个全字节的情况，比如采用双线模式并且仅使用两个周期发送交替字节时。在这种情况下，固件可采用四线模式（ABMODE = 11）并发送一个字节，方法是 ALTERNATE 的 Bit7 和 Bit3 置“1”（I03 保持高电平）且 Bit6 和 Bit2 置“0”（I02 线保持低电平）。此时，半字节的高 2 位存放在 ALTERNATE 的 Bit4 和 Bit3,低 2 位存放在位 1 和 0 中。例如，如果半字节 2（0010）通过 IO0/IO1 发送，则 ALTERNATE 应设置为 0x8A（1000_1010）。

空指令周期阶段

在空指令周期阶段，给定的 1-31 个周期内不发送或不接收任何数据，目的是当采用更高的时钟频率时，给 Flash 留出准备数据阶段的时间。这一阶段中给定的周期数在 QSPI_CCR3 寄存器的 DCYC[4:0]字段中指定。在 SDR 模式下，持续时间被指定为一定个数的全时钟周期。

若 DCYC 为零，则跳过空指令周期阶段，命令序列直接进入数据阶段（如果存在）。

空指令周期阶段的操作模式由 DMODE 确定。

为确保数据信号从输出模式转变为输入模式有足够的“周转”时间，使用双线和四线模式从 Flash 接收数据时，至少需要指定一个空指令周期。

数据阶段

在数据阶段，可从 Flash 接收或向其发送任意数量的字节。

在间接模式和自动轮询模式下，待发送/接收的字节数在 QSPI_DLR 寄存器中指定。

在间接写入模式下，发送到 Flash 的数据必须写入 QSPI_DR 寄存器。在间接读取模式下，通过读取 QSPI_DR 寄存器获得从 Flash 接收的数据。

数据阶段可一次发送/接收 1 位（在单线 SPI 模式中通过 SO）、2 位（在双线 SPI 模式中通过 IO0 / IO1）或 4 位（在四线 SPI 模式中通过 IO0 / IO1 / IO2 / IO3）。这可通过 QSPI_CCR2 寄存器中的 ABMODE[1:0]字段进行配置。

若 DMODE = 00，则跳过数据阶段，命令序列在拉高 nCS 时立即完成。这一配置仅可用于仅间接写入模式。

27.3.4 QSPI 信号接口协议模式

单线 SPI 模式

传统 SPI 模式允许串行发送/接收单独的 1 位。在此模式下，数据通过 SO（IO0）信号发送到 Flash。从 Flash 接收到的数据通过 SI（IO1）送达。

通过将 QSPI_CCRn 寄存器中的 IMODE/ADMODE/ABMODE/DMODE 字段设置为 01，可对不同的命令阶段分别进行配置，以使用此单个位模式。

在每个已配置为单线模式的阶段中：

- IO0（SO）为输出模式
- IO1（SI）为高阻输入模式）

- IO2 为输出模式并强制置“1”（以禁止“写保护”功能）
- IO3 为输出模式并强制置“1”（以禁止“保持”功能）

若 DMODE = 01，这对于空指令阶段也同样如此。

双线 SPI 模式

在双线模式下，通过 IO0 / IO1 信号同时发送/接收两位。

通过将 QSPI_CCRn 寄存器中的 IMODE/ADMODE/ABMODE/DMODE 字段设置为 10,可对不同的命令阶段分别进行配置，以使用双线 SPI 模式。

在每个已配置为双线模式的阶段中：

- IO0/IO1 在数据阶段进行读取操作时处于高阻态（输入），在其他情况下为输出
- IO2 处于输出模式并强制置“1”
- IO3 处于输出模式并强制置“1”

在空指令阶段，若 DMODE = 01，则 IO0/IO1 始终保持高阻态。

四线 SPI 模式

在四线模式下，通过 IO0/IO1/IO2/IO3 信号同时发送/接收四位。

通过将 QSPI_CCRn 寄存器中的 IMODE/ADMODE/ABMODE/DMODE 字段设置为 11,可 对不同的命令阶段分别进行配置，以使用四线 SPI 模式。

在每个已配置为四线模式的阶段中，IO0/IO1/IO2/IO3 在数据阶段进行读取操作时均处于高 阻态（输入），在其他情况下为输出。

在空指令阶段中，若 DMODE = 11,则 IO0/IO1/IO2/IO3 均为高阻态。

I02 和 I03 仅用于 QSPI 模式如果未配置任何阶段使用四线 SPI 模式，即使 QSPI 激活，对应 IO2 和 I03 的引脚也可用于其他功能。

SDR 模式

默认情况下，QSPI 在单倍数据速率（SDR）模式下工作。

在 SDR 模式下，当 QSPI 驱动 IO0/SO、IO1、IO2、IO3 信号时，这些信号仅在 CLK 的下降沿发生转变。

在 SDR 模式下接收数据时，QSPI 假定 Flash 也通过 CLK 的下降沿发送数据。默认情 况下（SSHIFT = 0 时），将使用 CLK 后续的边沿（上升沿）对信号进行采样。

27.3.5 QSPI 间接模式

在间接模式下，通过写入 QSPI 寄存器来触发命令；并通过读写数据寄存器来传输数据，就如同对待其他通信外设那样。

若 FMODE = 00，则 QSPI 处于间接写入模式，字节在数据阶段中发送到 Flash。通过写入数据寄存器 QSPI_DR 的方式提供数据。

若 FMODE = 01，则 QSPI 处于间接读取模式，在数据阶段中从 Flash 接收字节。通过读取

QSPI_DR 来获取数据。

读取/写入的字节数在数据长度寄存器（QSPI_DLR）中指定。如果 QSPI_DLR = OxFFF_FFFF（全为“1”），则数据长度视为未定义，QSPI 将继续传输数据，直到到达（由 FSIZE 定义的）Flash 的结尾。如果不传输任何字节，DMODE 应设置为 00。

如果 QSPI_DLR = 0xFF，并且 FSIZE = 0x1F （最大值指示一个 4GB 的 Flash），在此特殊情况下，传输将无限继续下去，仅在出现终止请求或 QSPI 被禁止后停止。在读取最后一个存储器地址后（地址为 OxFFFF_FFFF），将从地址=0x0000_0000 开始继续读取。

当发送或接收的字节数达到编程设定值时，如果 TCIE = 1,则 TCF 置 1 并产生中断。在数据数量不确定的情况下，将根据 QSPI_CR 中定义的 Flash 大小，在达到外部 SPI 的限制时，TCF 置 1。

触发命令启动

从本质上讲，在固件给出命令所需的最后一点信息时，命令即会启动。根据 QSPI 的配 置，在间接模式下有三种触发命令启动的方式。在出现以下情形时，命令立即启动：

1、 对 INSTRUCTION[7:0] （QSPI_CCR1）执行写入操作，如果没有地址是必需的（当 ADMODE = 00）并且不需要固件提供数据（当 FMODE = 01 或 DMODE = 00）
2、 对 ADDRESS[31:0] （QSPI_AR）执行写入操作，如果地址是必需的（当 ADMODE = 00）并且不需要固件提供数据（当 FMODE = 01 或 DMODE = 00）
3、 对 DATA[7:0]（QSPI_DR）执行写入操作，如果地址是必需的（当 ADMODE != 00） 并且需要固件提供数据（当 FMODE = 00 并且 DMODE != 00）

写入交替字节寄存器（QSPI_ABR）始终不会触发命令启动。如果需要交替字节，必须预先进行编程。

如果命令启动，BUSY 位（QSPI_LSR 的位 5）将自动置 1。

FIFO 和数据管理

在间接模式中，数据将通过 QSPI 内部的一个 32 字节 FIFO。FLEVEL[5:0] （QSPI_SR2）指示 FIFO 目前保存了多少字节。

在间接写入模式下（FMODE = 00）,固件写入 QSPI_DR 时，将在 FIFO 中加入数据。 字写入将在 FIFO 中增加 4 个字节，半字写入增加 2 个字节，而字节写入仅增加 1 个字节。

如果固件在 FIFO 中加入的数据过多（超过 DL[31:0]指示的值），将在写入操作结束（TCF 置 1）时从 FIFO 中清除超出的字节。

对 QSPI_DR 的字节/半字访问必须仅针对该 32 位寄存器的最低有效字节/半字。

FTHRES[3:0]用于定义 FIFO 的阈值如果达到阈值，FTF（FIFO 阈值标志）置 1 在间接读取模式下，从 FIFO 中读取的有效字节数超过阈值时，FTF 置 1。从 Flash 中读取最后一个字节后，如果 FIFO 中依然有数据，则无论 FTHRES 的设置为何，FTF 也都会置 1。在间接写入模式下，当 FIFO 中的空字节数超过阈值时，FTF 置 1。

如果 FTIE = 1,则 FTF 置 1 时产生中断。如果阈值条件不再为“真” （CPU 传输了足够的数据后），则 FTF 由 HW 清零。

在间接模式下，当 FIFO 已满，QSPI 将暂时停止从 Flash 读取字节以避免上溢。

27.3.6 QSPI 状态标志轮询模式

在自动轮询模式下，QSPI 周期性启动命令以读取一定数量的状态字节（最多 4 个）。 可屏蔽接收的字节以隔离一些状态位，从而在所选的位具有定义的值时可产生中断。

对 Flash 的访问最初与在间接读取模式下相同：如果不需要地址（AMODE = 00），则在写入 QSPI_CCR 时即开始访问。否则，如果需要地址，则在写入 QSPI_AR 时开始第一次访问。BUSY 在此时变为高电平，即使在周期性访问期间也保持不变。

在自动轮询模式下，MASK[7:0]（QSPI_PSMAR）的内容用于屏蔽来自 Flash 的数据。

如果 MASK[n] = 0，则屏蔽结果的位 n，从而不考虑该位。如果 MASK[n] = 1 并且位[n]的内 容与 MATCH[n] （QSPI_PSMAR）相同，说明存在位 n 匹配。

如果轮询匹配模式位（PMM， QSPI_CR3[7]）为 0，将激活“AND”匹配模式。这意味着状态匹配标志（SMF）仅在全部未屏蔽位均存在匹配时置 1。

如果 PMM = 1，则激活“OR”匹配模式。这意味着 SMF 在任意未屏蔽位存在匹配时置 1。

如果 SMIE = 1，则在 SMF 置 1 时调用一个中断。

如果自动轮询模式停止（APMS）位置 1，则操作停止并且 BUSY 位在检测到匹配时清零。否则，BUSY 位保持为“1”，在发生中止或禁止 QSPI（EN = 0）前继续进行周期性访问。

数据寄存器（QSPI_DR）包含最新接收的状态字节（FIFO 停用）。数据寄存器的内容不受匹配逻辑所用屏蔽方法的影响。FTF 状态位在新一次状态读取完成后置 1，并且 FTF 在数据读取后清零。

27.3.7 QSPI Flash 配置

设备配置寄存器（QSPI_DCR）可用于指定外部 SPI Flash 的特性。

FSIZE[4:0]字段使用下面的公式定义外部存储器的大小：

Flash 中的字节数=$2^{[FSIZE+1]}$

FSIZE+1 是对 Flash 寻址所需的地址位数。在间接模式下，Flash 容量最高可达 4GB （使用 32 位进行寻址）。

QSPI 连续执行两条命令时，它在两条命令之间将片选信号（nCS）置为高电平默认仅一个 CLK 周期时长。如果 Flash 需要命令之间的时间更长，可使用片选高电平时间（CSHT）字段指定 nCS 必须保持高电平的最少 CLK 周期数（最大为 8）。

时钟模式（CKMODE）位指示命令之间的 CLK 信号逻辑电平（nCS = 1 时）。

27.3.8 QSPI 延迟数据采样

默认情况下，QSPI 在 Flash 驱动信号后过半个 CLK 周期才对 Flash 驱动的数据采样。

27.3.9 QSPI 配置

QSPI 配置分两个阶段

- QSPI IP 配置
- QSPI Flash 配置

QSPI 在配置完毕并使能后，即可在间接模式和状态轮询模式这两种操作模式之一下工作。

QSPI IP 配置

通过 QSPI_CR 配置 QSPI IP 。用户应配置传入数据的时钟预分频器的分频系数以及采样移位设置。

DMA 请求通过 DMAEN 位置 1 使能。若是用于中断，则相关使能位也可在该阶段置 1。

生成 DMA 请求或生成中断的 FIFO 电平在 FTHRES 位中进行编程。

QSPI Flash 配置

与外部目标 Flash 相关的参数通过 QSPI_DCR 寄存器进行配置。用户应在 FSIZE 位中编程 Flash 的大小、在 CSHT 位中编程片选保持高电平的最短时间以及在 MODE 位中编程功能模式（模式 0 或模式 3）。

27.3.10 QSPI 的用法

使用 FMODE[1:0]（QSPI_CCR4）选择操作模式。

间接模式的操作步骤

FMODE 编程为 00 可选择间接写入模式，将数据发送到 Flash。
FMODE 编程为 01 可选择间接读取模式，读取 Flash 中的数据。

QSPI 用于间接模式时，采用以下方式构建帧：

- 在 QSPI_DLR 中指定待读取或写入的字节数
- 在 QSPI_CCR 中指定帧格式、模式和指令代码
- 在 QSPI_ABR 中指定要在地址阶段后立即发送的可选交替字节
- 在 QSPI_AR 中指定目标地址
- 通过 QSPI_DR 从 FIFO 读取数据/向 FIFO 写入数据

在写入控制寄存器（QSPI_CR）时，用户可指定以下设置：

- 使能位（EN）设置为“1”
- 超时计数器使能位（TCEN）
- FIFO 阈值（FTRHES），以指示 FTF 标志在何时置 1
- 中断使能
- 自动轮询模式参数：匹配模式和停止模式（在 FMODE = 11 时有效）
- 时钟预分频器

在写入通信配置寄存器（QSPI_CCRn）时，用户指定以下参数：

- 通过 INSTRUCTION 位指定指令字节
- 通过 IMODE 位指定指令发送方式（1/2/4 线）
- 通过 ADMODE 位指定地址发送方式（无/1/2/4 线）
- 通过 ADSIZE 位指定地址长度（8/16/24/32 位）
- 通过 ABMODE 位指定交替字节发送方式（无/1/2/4 线）
- 通过 ABSIZE 位指定交替字节数（1/2/3/4）
- 通过 DCYC 位指定空指令的周期数
- 通过 DMODE 位指定数据发送/接收方式（无/1/2/4 线）

如果无需为某个命令更新地址寄存器（QSPI_ARn）与数据寄存器（QSPI_DR），则在写入 QSPI_CCR1 时，该命令序列便立即启动。在 ADMODE 和 DMODE 应为 00 时，或在间接读取模式（FMODE = 01）下仅 ADMODE = 00 时，便属于此情况。在需要地址（ADMODE 不为 00），但无需写入数据寄存器（FMODE = 01 或 DMODE = 00）时，通过写入 QSPI_AR 更新地址后，命令序列便立即启动。

在数据传输（FMODE = 00 并且 DMODE ! = 00）中，通过 QSPI_DR 写入 FIFO 触发通道启动。

状态标志轮询模式

将 FMODE 字段（QSPI_CCR4）设置为 10，使能状态标志轮询模式在此模式下，将发送编程的帧并周期性检素数据。

每帧中读取的最大数据量为 4 字节。如果 QSPI_DLR 请求更多的数据，则忽略多余的部分并仅读取 4 个字节。

在 QSPI_PISR 寄存器中指定周期性。

在检索到状态数据后，可在内部进行处理，以达到以下目的：

- 将状态匹配标志位置 1，如果使能，还将产生中断
- 自动停止周期性检索状态字节

接收到的值可通过存储于 QSPI_PSMKR 中的值进行屏蔽，并与存储在 QSPI_PSMAR 中的值进行或运算或与运算。

若是存在匹配，则状态匹配标志置 1，并且在使能了中断的情况下还将产生中断；如果 AMPS 位置 1，则 QSPI 自动停止。

在任何情况下，最新的检索值都在 QSPI_DR 中可用。

27.3.11 指令仅发送一次

一些 Flash （例如 Winbound）能够提供一种模式，指令在该模式中仅通过第一个命令序列进行发送，后续的命令根据地址直接启动。用户通过使用 SIOO 位可利用此功能的优势。

SIOO 对于所有功能模式（间接模式和状态轮询模式）均有效。如果 SIOO 位置 1，仅第一条命令发送指令，接着对 QSPI_CCRn 执行写入操作。后续命令序列都将跳过指令阶段，直到 QSPI_CCR 被写入为止。

IMODE = 00 （无指令）时，SIOO 不起作用。

27.3.12 QSPI 差错管理

在以下情况下可能产生错误：

- 在间接模式或状态标志轮询模式下，如果在 QSPI_AR 中编程了错误的地址（根据 QSPI_DCR 中 FSIZE[4:0]定义的 Flash 大小）：TEF 将置 1，如果使能，还将产生中断。
- 另外，在间接模式下，如果地址加数据的长度超过 Flash 的大小，TEF 将在访问被触发 时置 1。

27.3.13 QSPI 的繁忙位和中止功能

在 QSPI 启动对 Flash 的操作时，QSPI_SR 中的 BUSY 位自动置 1。

在间接模式下，在 QSPI 完成了请求的命令序列并且 FIFO 为空时，BUSY 位复位。

在自动轮询模式下，仅当最后一次周期性访问完成时（因 APMS = 1 时发生匹配，或因中止），BUSY 位才变为低电平。

任何操作都可通过将 QSPI_CR 中的 ABORT 位置 1 来中止。在完成中止时，BUSY 位和 ABORT 位自动复位，FIFO 淆空。

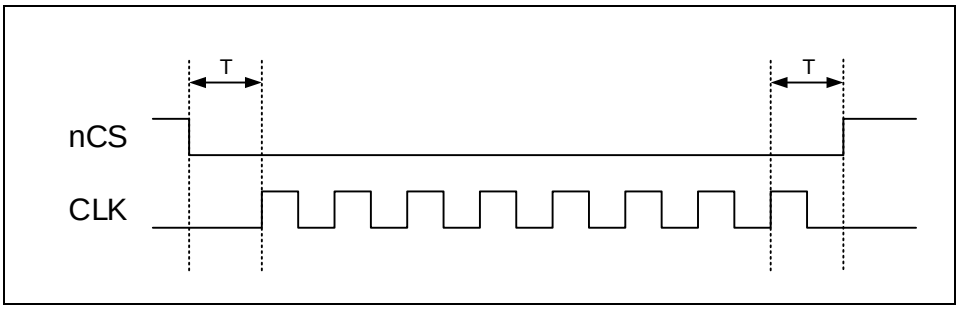
注: 如果中止对状态寄存器的写入操作，有些 Rash 可能发生错误行为。

27.3.14 nCS 行为

默认情况下，nCS 为高电平，取消选择外部 Flash。nCS 在操作开始前下降，在操作完成时立即上升。

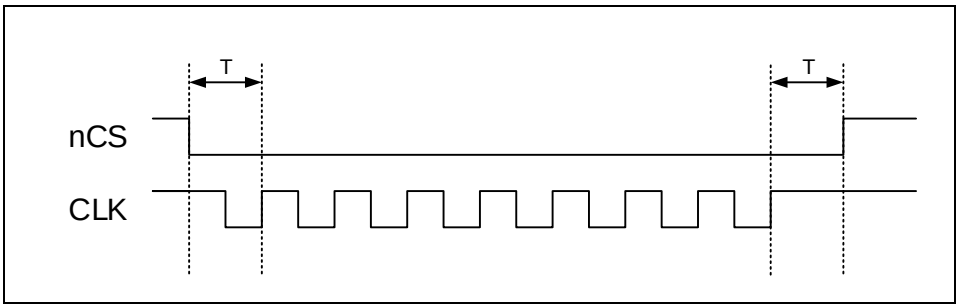
当 CKMODE = 0 （“模式 0”，在未进行任何操作时 CLK 保持低电平）时，nCS 在操作首次升高 CLK 边沿时的一个 CLK 周期前降至低电平，在操作最后一次升高 CLK 边沿时的一个 CLK 周期后升至高电平，如下图所示：

CKMODE = 0 时的 nCS （T = CLK 周期）



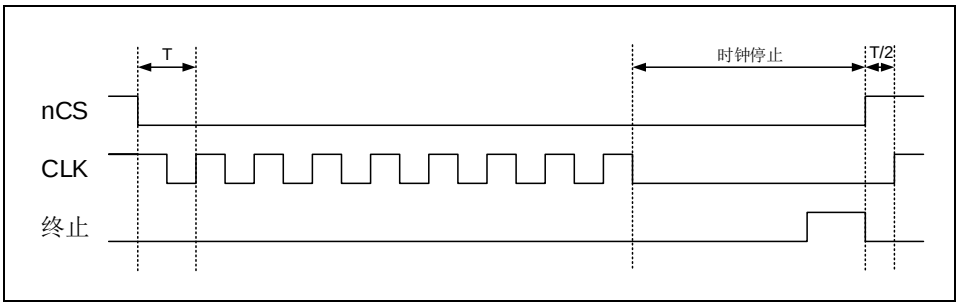
当 CKMODE = 1（“模式 3”，在未进行任何操作时 CLK 升至高电平）且 DDRM = 0（SDR 模式）时，nCS 仍在操作首次升高 CLK 边沿时的一个 CLK 周期前降至低电平，在操作最后一次升高 CLK 边沿时的一个 CLK 周期后升至高电平，如下图所示：

SDR 模式下 CKMODE = 1 时的 nCS （T = CLK 周期）



若 FIFO 在读取操作中保持写满状态或在写入操作中保持为空，则在固件干预 FIFO 前，操作停止并且 CLK 保持低电平。若操作停止时发生中止，则 nCS 在请求中止后即升至高电平，CLK 则在半个周期后升至高电平，如下图所示：

CKMODE=1 且发生中止时的 nCS（T=CLK 周期）



27.4 QSPI 中断

发生如下事件时可生成中断:

- 状态匹配
- FIFO 阈值
- 传输完成

QSPI 中断请求

中断事件	事件标志	使能控制位
状态匹配	SMF	SMIE
FIFO 阈值	FTF	FTIE
传输完成	TCF	TCIE

27.5 QSPI 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
QSPI_CR1	QSPI 控制寄存器 1	7EF900H	-	-	-	-	-	-	ABORT	EN	xxxx,xx00
QSPI_CR2	QSPI 控制寄存器 2	7EF901H	-	-	-	FTHRES[4:0]					xxx0,0000
QSPI_CR3	QSPI 控制寄存器 3	7EF902H	PMM	APMS	-	-	SMIE	FTIE	TCIE	TEIE	00xx,0000
QSPI_CR4	QSPI 控制寄存器 4	7EF903H	PSCR[7:0]								0000,0000
QSPI_DCR1	QSPI 器件配置寄存器 1	7EF904H	-	CSHT[2:0]			QSPI_IP[1:0]		-	CKMODE	x000,00x0
QSPI_DCR2	QSPI 器件配置寄存器 2	7EF905H	-	-	-	FSIZE[4:0]					0000,0000
QSPI_SR1	QSPI 状态寄存器 1	7EF906H	-	-	BUSY	-	SMF	FTF	TCF	TEF	xx0x,0100
QSPI_SR2	QSPI 状态寄存器 2	7EF907H	-	-	FLEVEL[5:0]						xx00,0000
QSPI_FCR	QSPI 标志清零寄存器	7EF908H	-	-	-	-	CSMF	-	CTCF	CTEF	xxxx,0x00
QSPI_DLR1	QSPI 数据长度寄存器 1	7EF910H	DL[7:0]								0000,0000
QSPI_DLR2	QSPI 数据长度寄存器 2	7EF911H	DL[15:8]								0000,0000
QSPI_CCR1	QSPI 通信配置寄存器 1	7EF914H	INSTRUCTION[7:0]								0000,0000
QSPI_CCR2	QSPI 通信配置寄存器 2	7EF915H	ABMODE[1:0]		ADSIZE[1:0]		ADMODE[1:0]		IMODE[1:0]		0000,0000
QSPI_CCR3	QSPI 通信配置寄存器 3	7EF916H	-	DCYC[4:0]					ABSIZE[1:0]		x000,0000
QSPI_CCR4	QSPI 通信配置寄存器 4	7EF917H	-	-	-	SIOO	FMODE[1:0]		DMODE[1:0]		xxx0,0000
QSPI_AR1	QSPI 地址寄存器 1	7EF918H	ADR[7:0]								0000,0000
QSPI_AR2	QSPI 地址寄存器 2	7EF919H	ADR[15:8]								0000,0000
QSPI_AR3	QSPI 地址寄存器 3	7EF91AH	ADR[23:16]								0000,0000
QSPI_AR4	QSPI 地址寄存器 4	7EF91BH	ADR[31:24]								0000,0000
QSPI_ABR	QSPI 交替字节寄存器 1	7EF91CH	ALTERNATED[7:0]								0000,0000
QSPI_DR	QSPI 数据寄存器	7EF920H	DATA[7:0]								0000,0000
QSPI_PSMKR1	QSPI 状态屏蔽寄存器	7EF924H	MASK[7:0]								0000,0000
QSPI_PSMAR1	QSPI 状态匹配寄存器	7EF928H	MATCH[7:0]								0000,0000
QSPI_PIR1	QSPI 轮询间隔寄存器 1	7EF92CH	INTERVAL[7:0]								0000,0000
QSPI_PIR2	QSPI 轮询间隔寄存器 2	7EF92DH	INTERVAL[15:8]								0000,0000

27.5.1 QSPI 控制寄存器 1（QSPI_CR1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
QSPI_CR1	7EF900H	-	-	-	-	-	-	ABORT	EN

ABORT：中止请求。该位中止执行中的命令序列。在中止完成时自动复位。该位可停止当前的传输。在轮询模式或内存映射模式下，该位也用以复位 APM 位或 DM 位。

0：不请求中止

1：请求中止

EN：QSPI 功能使能控制位

0：禁止 QSPI

1：使能 QSPI

27.5.2 QSPI 控制寄存器 2（QSPI_CR2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
QSPI_CR2	7EF901H	-	-	-	FTHRES[4:0]				

FTHRES[4:0]：FIFO 阈值级别（FIFO threshold level）。

定义在间接模式下 FIFO 中将导致 FIFO 阈值标志 FTF 置 1 的字节数阈值。

在间接写入模式下（FMODE = 00）

0：如果 FIFO 中存在 1 个或更多空闲字节可供写入，则 FTF 置 1

1：如果 FIFO 中存在 2 个或更多空闲字节可供写入，则 FTF 置 1

…

31：如果 FIFO 中存在 32 个空闲字节可供写入，则 FTF 置 1

在间接读取模式下（FMODE = 01）

0：如果 FIFO 中存在 1 个或更多有效字节可供读取，则 FTF 置 1

1：如果 FIFO 中存在 2 个或更多有效字节可供读取，则 FTF 置 1

…

31：如果 FIFO 中存在 32 个有效字节可供读取，则 FTF 置 1

27.5.3 QSPI 控制寄存器 3（QSPI_CR3）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
QSPI_CR3	7EF902H	PMM	APMS	-	-	SMIE	FTIE	TCIE	TEIE

PMM：轮询匹配模式（Polling match mode）。

该位指示在自动轮询模式期间用来确定是否匹配的方法。仅可在 BUSY = 0 时修改该位。

0：AND 匹配模式。

如果从 Flash 接收的所有未屏蔽位均与匹配寄存器中的对应位相匹配，则 SMF 置 1。

1：OR 匹配模式。

如果从 Flash 接收的任意一个未屏蔽位与匹配寄存器中的对应位相匹配，则 SMF 置 1。

APMS：自动轮询模式停止（Automatic poll mode stop）

该位确定在匹配后自动轮询是否停止。仅可在 BUSY = 0 时修改该位。

0：仅通过中止或禁用 QSPI 停止自动轮询模式。

1：发生匹配时，自动轮询模式停止。

SMIE：状态匹配中断使能（Status match interrupt enabl）

0：禁止状态匹配中断

1：使能状态匹配中断

FTIE：FIFO 阈值中断使能（FIFO threshold interrupt enable）

0：禁止 FIFO 阈值中断

1：使能 FIFO 阈值中断

TCIE：传输完成中断使能（Transfer complete interrupt enable）

0：禁止传输完成中断

1：使能传输完成中断

TEIE：传输错误中断使能（Transfer error interrupt enable）

0：禁止传输错误中断

1：使能传输错误中断

27.5.4 QSPI 控制寄存器 4（QSPI_CR4）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
QSPI_CR4	7EF903H	PSCR[7:0]							

PSCR[7:0]：时钟预分频器（Clock prescaler）。

对于奇数时钟分频系数，CLK 的占空比并非 50%。时钟信号的高电平持续时间比低电平持续时间多一个周期。仅可在 BUSY = 0 时修改该位。

PSCR	QSPI 时钟频率
0	系统时钟/1
1	系统时钟/2
2	系统时钟/3
3	系统时钟/4
…	…
255	系统时钟/256

27.5.5 QSPI 器件配置寄存器 1（QSPI_DCR1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
QSPI_DCR1	7EF904H	-	CSHT[2:0]			QSPI_IP[1:0]		-	CKMODE

CSHT[2:0]：片选高电平时间（Chip select high time）。仅可在 BUSY = 0 时修改该位。

CSHT[2:0]	nCS 在 Flash 命令之间保持高电平时间
0	1 个周期
1	2 个周期
2	3 个周期
3	4 个周期
…	…
7	8 个周期

QSPI_IP[1:0]：QSPI 中断优先级

CKMODE：SPI 模式 0/模式 3（Mode 0 / mode 3）选择位。该位指示 CLK 在 nCS=1 时的电平。仅可在 BUSY = 0 时修改该位。

0：nCS 为高电平（片选释放）时，CLK 必须保持低电平。这称为模式 0。

1：nCS 为高电平（片选释放）时，CLK 必须保持高电平。这称为模式 3。

27.5.6 QSPI 器件配置寄存器 2（QSPI_DCR2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
QSPI_DCR2	7EF905H	-	-	-	FSIZE[4:0]				

FSIZE[4:0]：配置外部 Flash 存储器的大小。仅可在 BUSY = 0 时修改该位。

Flash 中的字节数 = $2^{[FSIZE+1]}$

FSIZE+1 是对 Flash 寻址所需的地址位数。在间接模式下，Flash 容量最高可达 4GB（使用 32 位进行寻址）。

27.5.7 QSPI 状态寄存器 1（QSPI_SR1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
QSPI_SR1	7EF906H	-	-	BUSY	-	SMF	FTF	TCF	TEF

BUSY：QSPI 忙标志位（只读位）

QSPI 操作进行时，该位置 1。在对 Flash 的操作完成并且 FIFO 为空时，该位自动清零。

SMF：状态匹配标志（Status match flag）

在自动轮询模式下，若未屏蔽的接收数据与匹配寄存器（QSPI_PSMAR）中的对应位相匹配，则该位置 1。向 CSMF 写入 1 可将该位清零。

FTF：FIFO 阈值标志（FIFO threshold flag）

在间接模式下，若达到 FIFO 阈值，或从 Flash 读取完成后，FIFO 中留有数据时，该位置 1。只要阈值条件不再为“真”，该位就自动清零。

在自动轮询模式下，每次读取状态寄存器时，该位即置 1；读取数据寄存器时，该位清零。

TCF：传输完成标志（Transfer complete flag）

在间接模式下，当传输的数据数量达到编程设定值，或在任何模式下传输中止时，该位置 1。向 CTCF 写入 1 时，该位清零。

TEF：传输错误标志（Transfer error flag）

在间接模式下访问无效地址时，该位置 1。向 CTEF 写入 1 可将该位清零。

27.5.8 QSPI 状态寄存器 2（QSPI_SR2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
QSPI_SR2	7EF907H	-	-	FLEVEL[5:0]					

FLEVEL[5:0]：FIFO 级别（FIFO level）

该字段给出 FIFO 中的有效字节数。FIFO 为空时 FLEVEL = 0；写满时 FLEVEL = 32

在自动状态轮询模式下，FLEVEL 为零。

27.5.9 QSPI 标志清零寄存器（QSPI_FCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
QSPI_FCR	7EF908H	-	-	-	-	CSMF	-	CTCF	CTEF

CSMF：清除状态匹配标志（Clear status match flag）。

写入 1 可将 QSPI_SR1 寄存器中的 SMF 标志清零

CTCF：清除传输完成标志（Clear transfer complete flag）。

写入 1 可将 QSPI_SR1 寄存器中的 TCF 标志清零

CTEF：清除传输错误标志（Clear Transfer error flag）。

写入 1 可将 QSPI_SR1 寄存器中的 TEF 标志清零

27.5.10 QSPI 数据长度寄存器（QSPI_DLR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
QSPI_DLR1	7EF910H	DL[7:0]							
QSPI_DLR2	7EF911H	DL[15:8]							

DL[15: 0]：QSPI 传输的数据长度

在间接模式和状态轮询模式下待检索的数据数量（值+1）。对状态轮询模式应使用不大于 3 的值（表示 4 字节）。在间接模式下，所有位置 1 表示未定义长度，QSPI 将继续传输数据直到到达由 FSIZE 定义的存储器末尾。

DL[15: 0]	传输的数据长度
0000H	传输 1 个字节
0001H	传输 2 个字节
0002H	传输 3 个字节
0003H	传输 4 个字节
…	…
FFFEH	传输 65535 个字节
FFFFH	未定义长度 传输所有字节直到到达由 FSIZE 定义的 Flash 的结尾

27.5.11 QSPI 通信配置寄存器 1（QSPI_CCR1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
QSPI_CCR1	7EF914H	INSTRUCTION[7:0]							

INSTRUCTION[7:0]：指令(Instruction)。指定要发送到外部 SPI 设备的指令。该位指示在自动轮询模式期间用来确定是否匹配的方法。仅可在 BUSY = 0 时修改该位。

27.5.12 QSPI 通信配置寄存器 2（QSPI_CCR2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
QSPI_CCR2	7EF915H	ABMODE[1:0]		ADSIZE[1:0]		ADMODE[1:0]		IMODE[1:0]	

ABMODE[1:0]：交替字节模式。仅可在 BUSY = 0 时修改该位。

ABMODE[1:0]	交替字节模式
00	无交替字节
01	单线传输交替字节
10	双线传输交替字节
11	四线传输交替字节

ADSIZE[1:0]：地址长度。仅可在 BUSY = 0 时修改该位。

ABMODE[1:0]	地址长度
00	8 位地址
01	16 位地址
10	24 位地址
11	32 位地址

ADMODE[1:0]：地址模式。仅可在 BUSY = 0 时修改该位。

ADMODE[1:0]	地址模式
00	无地址
01	单线传输地址
10	双线传输地址
11	四线传输地址

IMODE[1:0]：指令模式。仅可在 BUSY = 0 时修改该位。

IMODE[1:0]	指令模式
00	无指令
01	单线传输指令
10	双线传输指令
11	四线传输指令

27.5.13 QSPI 通信配置寄存器 3（QSPI_CCR3）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
QSPI_CCR3	7EF916H	-	DCYC[4:0]					ABSIZE[1:0]	

DCYC[4:0]：空指令周期数（Number of dummy cycles）。该字段定义空指令阶段的持续时间。在 SDR 模式下，它指定 CLK 周期数(0-31)。仅可在 BUSY = 0 时修改该位。

ADSIZE[1:0]：交替字节长度。仅可在 BUSY = 0 时修改该位。

ABMODE[1:0]	交替字节长度
00	8 位交替字节
01	-
10	-
11	-

27.5.14 QSPI 通信配置寄存器 4（QSPI_CCR4）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
QSPI_CCR4	7EF917H	-	-	-	SIOO	FMODE[1:0]		DMODE[1:0]	

SIOO：仅发送指令一次模式。IMODE = 00 时，该位不起作用。仅可在 BUSY = 0 时修改该位。

0：在每个事务中发送指令

1：仅为第一条命令发送指令

FMODE[1:0]：设置 QSPI 操作的功能模式。仅可在 BUSY = 0 时修改该位。

FMODE [1:0]	功能模式
00	间接写入模式
01	间接读取模式
10	自动轮询模式
11	-

DMODE[1:0]：数据模式。该字段还定义空指令阶段的操作模式。仅可在 BUSY = 0 时修改该位。

DMODE[1:0]	数据模式
00	无数据
01	单线传输数据
10	双线传输数据
11	四线传输数据

27.5.15 QSPI 地址寄存器（QSPI_AR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
QSPI_AR1	7EF918H	ADR[7:0]							
QSPI_AR2	7EF919H	ADR[15:8]							
QSPI_AR3	7EF91AH	ADR[23:16]							
QSPI_AR4	7EF91BH	ADR[31:24]							

ADR[31:0]：设置要发送到外部 Flash 的地址。

必须先写 ADR 的 MSB，写完 LSB 后触发操作。

27.5.16 QSPI 交替字节寄存器（QSPI_ABR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
QSPI_ABR	7EF91CH	ALTERNATED[7:0]							

ALTERNATE[7:0]：设置要在地址后立即发送到外部 SPI 设备的可选数据。仅可在 BUSY = 0 时可写入。

27.5.17 QSPI 数据寄存器（QSPI_DR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
QSPI_DR	7EF920H	DATA[7:0]							

DATA[7:0]：设置要与外部 SPI 设备交换的数据。

在间接写入模式下，写入该寄存器的数据在数据阶段发送到 Flash，在此之前则存储于 FIFO，如果 FIFO 太满，将暂停写入，直到 FIFO 具有足够的空间接受要写入的数据才继续。字节写入将在 FIFO 中增加 1 个字节

在间接读取模式下，读取该寄存器可获得（通过 FIFO）已从 Flash 接收的数据。如果 FIFO 所含字节数比读取操作要求的字节数少并且 BUSY=1，将暂停读取，直到足够的数据出现或传输完成（不分先后）才继续。字节读取将移除 FIFO 中的 1 个字节。

在自动轮询模式下，该寄存器包含最后从 Flash 读取的数据（未进行屏蔽）。

27.5.18 QSPI 状态屏蔽寄存器（QSPI_PSMKR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
QSPI_PSMKR1	7EF924H	MASK[7:0]							

MASK[7:0]：状态屏蔽。对在轮询模式下接收的状态字节进行屏蔽。仅可在 BUSY = 0 时可写入。

对于位 n：

0：屏蔽在自动轮询模式下所接收数据的位 n，在匹配逻辑中不考虑其值。

1：不屏蔽在自动轮询模式下所接收数据的位 n，在匹配逻辑中考虑其值。

27.5.19 QSPI 状态匹配寄存器（QSPI_PSMAR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
QSPI_PSMAR1	7EF928H	MATCH[7:0]							

MATCH[7:0]：该值将与屏蔽状态寄存器比较以进行匹配。仅可在 BUSY = 0 时可写入。

27.5.20 QSPI 轮训间隔寄存器（QSPI_PIR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
QSPI_PIR1	7EF92CH	INTERVAL[7:0]							
QSPI_PIR2	7EF92DH	INTERVAL[15:8]							

INTERVAL[15: 0]：轮询间隔（Polling interval）。自动轮询阶段读取操作之间的 CLK 周期数。

仅可在 BUSY = 0 时可写入。

27.6 范例程序

27.6.1 使用 QSPI 直接读写串行 Flash

请参考：

https://www.stcaimcu.com/forum.php?mod=viewthread&tid=10762&extra=

27.6.2 使用 QSPI DMA 读写串行 Flash

请参考：

https://www.stcaimcu.com/forum.php?mod=viewthread&tid=10762&extra=

28 I2C 总线

产品线	I2C
Ai8051U 系列	●

Ai8051U 系列的单片机内部集成了一个 I²C 串行总线控制器。I²C 是一种高速同步通讯总线，通讯使用 SCL（时钟线）和 SDA（数据线）两线进行同步通讯。对于 SCL 和 SDA 的端口分配，Ai8051U 系列的单片机提供了切换模式，可将 SCL 和 SDA 切换到不同的 I/O 口上，以方便用户将一组 I²C 总线当作多组进行分时复用。

与标准 I²C 协议相比较，忽略了如下两种机制：

- 发送起始信号（START）后不进行仲裁
- 时钟信号（SCL）停留在低电平时不进行超时检测

Ai8051U 系列的 I²C 总线提供了两种操作模式：主机模式（SCL 为输出口，发送同步时钟信号）和从机模式（SCL 为输入口，接收同步时钟信号）

28.1 I2C 功能脚切换

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW2	BAH	EAXFR	-	**I2C_S[1:0]**		CMPO_S	S4_S	S3_S	S2_S

I2C_S[1:0]：I²C 功能脚选择位

I2C_S[1:0]	SCL	SDA
00	P2.4	P2.3
01	P1.5	P1.4
10	-	-
11	P3.2	P3.3

28.2 I2C 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
I2CCFG	I^2C 配置寄存器	7EFE80H	ENI2C	MSSL	MSSPEED[5:0]						0000,0000
I2CMSCR	I^2C 主机控制寄存器	7EFE81H	EMSI	-	-	-	MSCMD[3:0]				0xxx,0000
I2CMSST	I^2C 主机状态寄存器	7EFE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO	00xx,xx10
I2CSLCR	I^2C 从机控制寄存器	7EFE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0
I2CSLST	I^2C 从机状态寄存器	7EFE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000
I2CSLADR	I^2C 从机地址寄存器	7EFE85H	SLADR[6:0]							MA	0000,0000
I2CTXD	I^2C 数据发送寄存器	7EFE86H									0000,0000
I2CRXD	I^2C 数据接收寄存器	7EFE87H									0000,0000
I2CMSAUX	I^2C 主机辅助控制寄存器	7EFE88H	-	-	-	-	-	-	-	WDTA	xxxx,xxx0
I2CPSCR	I^2C 主机时钟分频寄存器	7EFE89H	MSSPEED[13:6]								0000,0000

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
I2CTOCR	I2C 从机超时控制寄存器	7EFD84H	ENTO	ENTOI	SCALE	-	-	-	-	-	000x,xxxx
I2CTOSR	I2C 从机超时状态寄存器	7EFD85H	CTOIF	-	-	-	-	-	-	TOIF	0xxx,xxx0
I2CTOTH	I2C 从机超时长度寄存器	7EFD86H	TM[15:8]								0000,0000
I2CTOTL	I2C 从机超时长度寄存器	7EFD87H	TM[7:0]								0000,0000
I2CTOTE	I2C 从机超时长度寄存器	7EFD8DH	TM[23:16]								0000,0000

28.3 I2C 主机模式

28.3.1 I2C 配置寄存器（I2CCFG），总线速度控制 1

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CCFG	7EFE80H	ENI2C	MSSL	MSSPEED[5:0]					

ENI2C：I²C 功能使能控制位

0：禁止 I²C 功能

1：允许 I²C 功能

MSSL：I²C 工作模式选择位

0：从机模式

1：主机模式

MSSPEED[5:0]：I2C 总线速度控制的低 6 位，和 I2CPSCR 中的 MSSPEED[13:6]合并为 14 位分频器

28.3.2 I2C 主机时钟分频寄存器（I2CPSCR），总线速度控制 2

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CPSCR	7EFE889H	MSSPEED[13:6]							

MSSPEED[13:6]和 I2CCFG 中的 MSSPEED[5:0]合并为 14 位 MSSPEED[13:0]分频器

MSSPEED[13:0]：I²C 总线速度（等待时钟数）控制，**I2C 总线速度＝SYSCLK / 2 / (MSSPEED * 2 + 4)**

（I2C 最快速度为 SYSCLK/8）

MSSPEED[13:0]	对应的时钟数
0	4
1	6
2	8
…	…
x	2x+4
…	…
62	128
63	130
…	…
16383	32770

只有当 I²C 模块工作在主机模式时，MSSPEED 参数设置的等待参数才有效。此等待参数主要用于主机模式的以下几个信号：

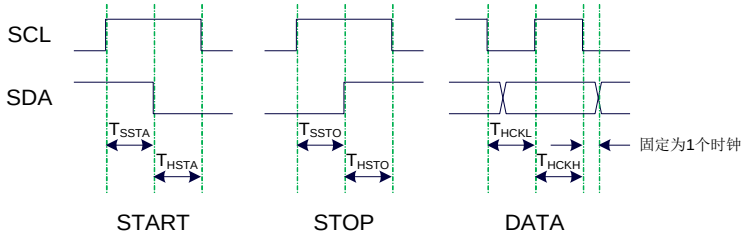
T_{SSTA}：起始信号的建立时间（Setup Time of START）

T_{HSTA}：起始信号的保持时间（Hold Time of START）

T_{SSTO}：停止信号的建立时间（Setup Time of STOP）

T_{HSTO}：停止信号的保持时间（Hold Time of STOP）

T_{HCKL}：时钟信号的低电平保持时间（Hold Time of SCL Low）



例 1：当 MSSPEED＝10 时，T_{SSTA}＝T_{HSTA}＝T_{SSTO}＝T_{HSTO}＝T_{HCKL}＝24/FOSC

例 2：当 24MHz 的工作频率下需要 400K 的 I2C 总线速度时，

MSSPEED＝(24M / 400K / 2 - 4) / 2＝13

28.3.3　I2C 主机控制寄存器（I2CMSCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSCR	7EFE81H	EMSI	-	-	-	MSCMD[3:0]			

EMSI：主机模式中断使能控制位

0：关闭主机模式的中断

1：允许主机模式的中断

MSCMD[3:0]：主机命令

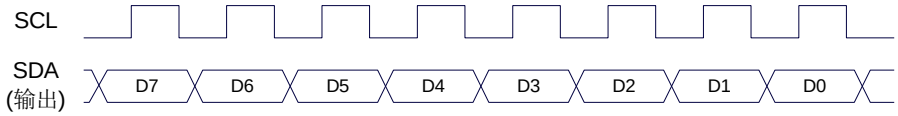
0000：待机，无动作。

0001：起始命令。

发送 START 信号。如果当前 I²C 控制器处于空闲状态，即 MSBUSY（I2CMSST.7）为 0 时，写此命令会使控制器进入忙状态，硬件自动将 MSBUSY 状态位置 1，并开始发送 START 信号；**若当前 I²C 控制器处于忙状态，写此命令可触发发送 START 信号**。发送 START 信号的波形如下图所示：

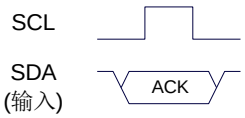
0010：发送数据命令。

写此命令后，I²C 总线控制器会在 SCL 管脚上产生 8 个时钟，并将 I2CTXD 寄存器里面数据按位送到 SDA 管脚上（先发送高位数据）。发送数据的波形如下图所示：



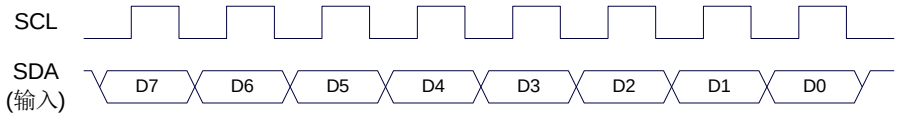
0011：接收 ACK 命令。

写此命令后，I²C 总线控制器会在 SCL 管脚上产生 1 个时钟，并将从 SDA 端口上读取的数据保存到 MSACKI（I2CMSST.1）。接收 ACK 的波形如下图所示：



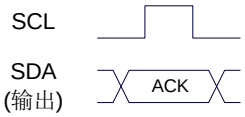
0100：接收数据命令。

写此命令后，I²C 总线控制器会在 SCL 管脚上产生 8 个时钟，并将从 SDA 端口上读取的数据依次左移到 I2CRXD 寄存器（先接收高位数据）。接收数据的波形如下图所示：



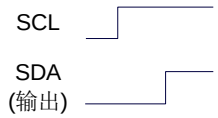
0101：发送 ACK 命令。

写此命令后，I²C 总线控制器会在 SCL 管脚上产生 1 个时钟，并将 MSACKO（I2CMSST.0）中的数据发送到 SDA 端口。发送 ACK 的波形如下图所示：



0110：停止命令。

发送 STOP 信号。写此命令后，I^2C 总线控制器开始发送 STOP 信号。信号发送完成后，硬件自动将 MSBUSY 状态位清零。STOP 信号的波形如下图所示：



0111：保留。

1000：保留。

1001：起始命令+发送数据命令+接收 ACK 命令。

此命令为命令 0001、命令 0010、命令 0011 三个命令的组合，下此命令后控制器会依次执行这三个命令。

1010：发送数据命令+接收 ACK 命令。

此命令为命令 0010、命令 0011 两个命令的组合，下此命令后控制器会依次执行这两个命令。

1011：接收数据命令+发送 ACK(0)命令。

此命令为命令 0100、命令 0101 两个命令的组合，下此命令后控制器会依次执行这两个命令。

注意：此命令所返回的应答信号固定为 ACK（0），不受 MSACKO 位的影响。

1100：接收数据命令+发送 NAK(1)命令。

此命令为命令 0100、命令 0101 两个命令的组合，下此命令后控制器会依次执行这两个命令。

注意：此命令所返回的应答信号固定为 NAK（1），不受 MSACKO 位的影响。

28.3.4 I2C 主机辅助控制寄存器（I2CMSAUX）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSAUX	7EFE88H	-	-	-	-	-	-	-	WDTA

WDTA：主机模式时 I²C 数据自动发送允许位

0：禁止自动发送

1：使能自动发送

若自动发送功能被使能，当 MCU 执行完成对 I2CTXD 数据寄存器的写操作后，I²C 控制器会自动触发“1010”命令，即自动发送数据并接收 ACK 信号。

28.3.5 I2C 主机状态寄存器（I2CMSST）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSST	7EFE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO

MSBUSY：主机模式时 I²C 控制器状态位（只读位）

0：控制器处于空闲状态

1：控制器处于忙碌状态

当 I²C 控制器处于主机模式时，在空闲状态下，发送完成 START 信号后，控制器便进入到忙碌状态，忙碌状态会一直维持到成功发送完成 STOP 信号，之后状态会再次恢复到空闲状态。

MSIF：主机模式的中断请求位（中断标志位）。当处于主机模式的 I²C 控制器执行完成寄存器 I2CMSCR 中 MSCMD 命令后产生中断信号，硬件自动将此位 1，向 CPU 发请求中断，响应中断后 MSIF 位必须用软件清零。

MSACKI：主机模式时，发送“0011”命令到 I2CMSCR 的 MSCMD 位后所接收到的 ACK 数据。（只读位）

MSACKO：主机模式时，准备将要发送出去的 ACK 信号。当发送“0101”命令到 I2CMSCR 的 MSCMD 位后，控制器会自动读取此位的数据当作 ACK 发送到 SDA。

28.4 I2C 从机模式

28.4.1 I2C 从机控制寄存器（I2CSLCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLCR	7EFE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST

ESTAI：从机模式时接收到 START 信号中断允许位

0：禁止从机模式时接收到 START 信号时发生中断

1：使能从机模式时接收到 START 信号时发生中断

ERXI：从机模式时接收到 1 字节数据后中断允许位

0：禁止从机模式时接收到数据后发生中断

1：使能从机模式时接收到 1 字节数据后发生中断

ETXI：从机模式时发送完成 1 字节数据后中断允许位

0：禁止从机模式时发送完成数据后发生中断

1：使能从机模式时发送完成 1 字节数据后发生中断

ESTOI：从机模式时接收到 STOP 信号中断允许位

0：禁止从机模式时接收到 STOP 信号时发生中断

1：使能从机模式时接收到 STOP 信号时发生中断

SLRST：复位从机模式

28.4.2 I2C 从机状态寄存器（I2CSLST）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLST	7EFE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	-	SLACKI	SLACKO

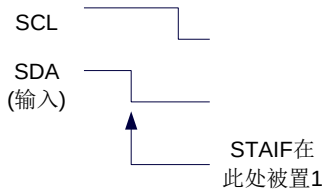
SLBUSY：从机模式时 I²C 控制器状态位（只读位）

0：控制器处于空闲状态

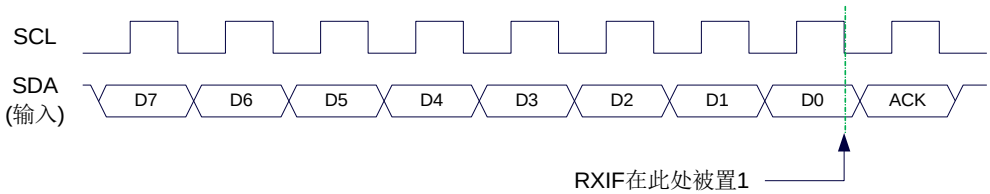
1：控制器处于忙碌状态

当 I²C 控制器处于从机模式时，在空闲状态下，接收到主机发送 START 信号后，控制器会继续检测之后的设备地址数据，若设备地址与当前 I2CSLADR 寄存器中所设置的从机地址像匹配时，控制器便进入到忙碌状态，忙碌状态会一直维持到成功接收到主机发送 STOP 信号，之后状态会再次恢复到空闲状态。

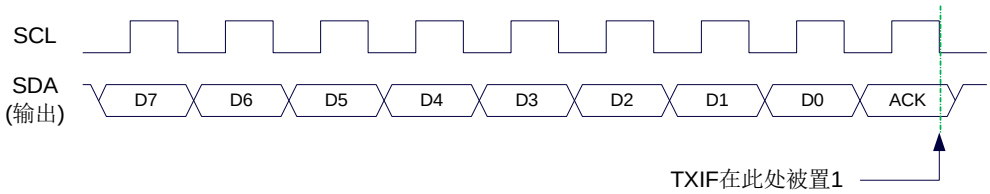
STAIF：从机模式时接收到 START 信号后的中断请求位。从机模式的 I²C 控制器接收到 START 信号后，硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 STAIF 位必须用软件清零。STAIF 被置 1 的时间点如下图所示：



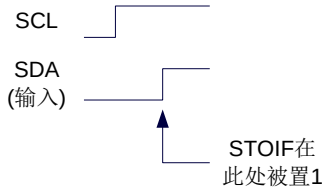
RXIF：从机模式时接收到 1 字节的数据后的中断请求位。从机模式的 I²C 控制器接收到 1 字节的数据后，在第 8 个时钟的下降沿时硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 RXIF 位必须用软件清零。RXIF 被置 1 的时间点如下图所示：



TXIF：从机模式时发送完成 1 字节的数据后的中断请求位。从机模式的 I²C 控制器发送完成 1 字节的数据并成功接收到 1 位 ACK 信号后，在第 9 个时钟的下降沿时硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 TXIF 位必须用软件清零。TXIF 被置 1 的时间点如下图所示：

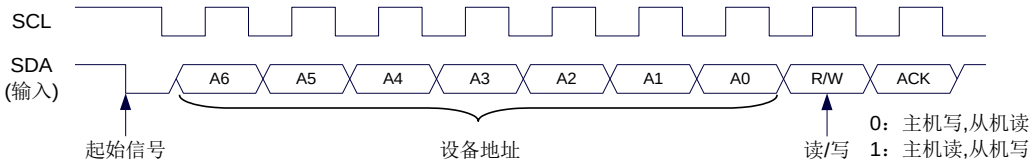


STOIF：从机模式时接收到 STOP 信号后的中断请求位。从机模式的 I²C 控制器接收到 STOP 信号后，硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 STOIF 位必须用软件清零。STOIF 被置 1 的时间点如下图所示：



SLACKI：从机模式时，接收到的 ACK 数据。

SLACKO：从机模式时，准备将要发送出去的 ACK 信号。



28.4.3 I2C 从机地址寄存器（I2CSLADR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLADR	7EFE85H	SLADR[6:0]							MA

SLADR[6:0]：从机设备地址

当 I²C 控制器处于从机模式时，控制器在接收到 START 信号后，会继续检测接下来主机发送出的设备地址数据以及读/写信号。当主机发送出的设备地址与 SLADR[6:0]中所设置的从机设备地址相匹配时，控制器才会向 CPU 发出中断求，请求 CPU 处理 I²C 事件；否则若设备地址不匹配，I²C 控制器继续继续监控，等待下一个起始信号，对下一个设备地址继续匹配。

MA：从机设备地址匹配控制

0：设备地址必须与 SLADR[6:0]继续匹配

1：忽略 SLADR 中的设置，匹配所有的设备地址

28.4.4 I2C 数据寄存器（I2CTXD，I2CRXD）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CTXD	7EFE86H								
I2CRXD	7EFE87H								

I2CTXD 是 I²C 发送数据寄存器，存放将要发送的 I²C 数据

I2CRXD 是 I²C 接收数据寄存器，存放接收完成的 I²C 数据

28.4.5 I2C 从机超时控制寄存器（I2CTOCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CTOCR	7EFD84H	ENTO	ENTOI	SCALE	-	-	-	-	-

ENTO：I2C 从机超时功能控制位

0：禁止 I2C 从机超时功能

1：使能 I2C 从机超时功能（**注：I2C 的主机模式不要使能接收超时功能**）

ENTOI：I2C 从机超时中断控制位

0：禁止 I2C 从机超时中断

1：使能 I2C 从机超时中断

SCALE：I2C 超时计数时钟源选择

0：1us 时钟（1MHz 时钟）。（注意：如需要使用此时钟源，用户必须先正确设置 IAP_TPS 寄存器。例如：若系统时钟为 12MHz，则需要将 IAP_TPS 设置为 12；若系统时钟为 22.1184MHz，则需要将 IAP_TPS 设置为 22；其他频率以此类推。特别注意，此 1us 的时钟并不是精准时间）

1：系统时钟

28.4.6 I2C 从机超时状态寄存器（I2CTOSR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CTOSR	7EFD85H	CTOIF	-	-	-	-	-	-	TOIF

CTOIF：写“1”清除 I2C 超时中断标志位 TOIF。（只写）

TOIF：I2C 超时中断请求标志位。（只读）

当发生 I2C 超时时，TOIF 会被硬件置“1”。如果 ENTOI 为 1 时会产生 I2C 中断，中断入口地址为原 I2C 的中断入口地址。标志位需要软件向 CTOIF 位写“1”清零

28.4.7 I2C 从机超时长度控制寄存器（I2CTOTE/H/L）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CTOTE	7EFD8DH	TM[23:16]							
I2CTOTH	7EFD86H	TM[15:8]							
I2CTOTL	7EFD87H	TM[7:0]							

TM[23:0]：I2C 超时时间控制位。

当 I2C 处于接收空闲状态时，内部超时计数器根据 SCALE 寄存器所选择的时钟源进行计数，当计数时间达到 TM 所设置的超时时间时，便会产生超时中断。当 I2C 接收数据完成时，复位内部超时计数器，重新进行超时计数。

注：

1、如果需要使能接收超时中断功能，则 TM 不可设置为 0，且 I2CTOTL、I2CTOTH、I2CTOTE 寄存器的设置必须先设置 I2CTOTL 和 I2CTOTH，最后设置 I2CTOTE

2、必须使能 I2C 为从机模式才有接收超时功能

3、接收超时功能必须在正确接收到一字节数据后才能触发

4、正在接收过程中，无接收超时功能

28.5　范例程序

28.5.1　I2C 主机模式访问 AT24C256（中断方式）

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

sbit    SDA     =   P1^4;
sbit    SCL     =   P1^5;

bit     busy;

void I2C_Isr() interrupt 24
{
    if (I2CMSST & 0x40)
    {
        I2CMSST &= ~0x40;                               //清中断标志
        busy = 0;
    }
}

void Start()
{
    busy = 1;
    I2CMSCR = 0x81;                                     //发送 START 命令
    while (busy);
}

void SendData(char dat)
{
    I2CTXD = dat;                                       //写数据到数据缓冲区
    busy = 1;
    I2CMSCR = 0x82;                                     //发送 SEND 命令
    while (busy);
}

void RecvACK()
{
    busy = 1;
    I2CMSCR = 0x83;                                     //发送读 ACK 命令
    while (busy);
}

char RecvData()
{
    busy = 1;
    I2CMSCR = 0x84;                                     //发送 RECV 命令
    while (busy);
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;                                     //设置 ACK 信号
```

```
    busy = 1;
    I2CMSCR = 0x85;                         //发送 ACK 命令
    while (busy);
}

void SendNAK()
{
    I2CMSST = 0x01;                         //设置 NAK 信号
    busy = 1;
    I2CMSCR = 0x85;                         //发送 ACK 命令
    while (busy);
}

void Stop()
{
    busy = 1;
    I2CMSCR = 0x86;                         //发送 STOP 命令
    while (busy);
}

void Delay()
{
    int i;

    for (i=0; i<3000; i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
}

void main()
{
    EAXFR = 1;                              //允许访问扩展的特殊寄存器，XFR
                                            //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                          //允许访问扩展的特殊寄存器，XFR
                                            //(8 位模式请使用这句，注释上一句)
    WTST = 0;                               //设置取程序代码等待时间，
                                            //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                              //设置访问片内的 xdata 速度，
                                            //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    I2CCFG = 0xe0;                          //使能 I2C 主机模式
    I2CMSST = 0x00;
```

```
    EA = 1;

    Start();                                    //发送起始命令
    SendData(0xa0);                             //发送设备地址+写命令
    RecvACK();
    SendData(0x00);                             //发送存储地址高字节
    RecvACK();
    SendData(0x00);                             //发送存储地址低字节
    RecvACK();
    SendData(0x12);                             //写测试数据 1
    RecvACK();
    SendData(0x78);                             //写测试数据 2
    RecvACK();
    Stop();                                     //发送停止命令

    Delay();                                    //等待设备写数据

    Start();                                    //发送起始命令
    SendData(0xa0);                             //发送设备地址+写命令
    RecvACK();
    SendData(0x00);                             //发送存储地址高字节
    RecvACK();
    SendData(0x00);                             //发送存储地址低字节
    RecvACK();
    Start();                                    //发送起始命令
    SendData(0xa1);                             //发送设备地址+读命令
    RecvACK();
    P0 = RecvData();                            //读取数据 1
    SendACK();
    P2 = RecvData();                            //读取数据 2
    SendNAK();
    Stop();                                     //发送停止命令

    while (1);
}
```

28.5.2　I2C 主机模式访问 AT24C256（查询方式）

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

sbit        SDA             =     P1^4;
sbit        SCL             =     P1^5;

void Wait()
{
    while (!(I2CMSST & 0x40));
    I2CMSST &= ~0x40;
}

void Start()
{
    I2CMSCR = 0x01;                                     //发送 START 命令
    Wait();
}

void SendData(char dat)
{
    I2CTXD = dat;                                       //写数据到数据缓冲区
    I2CMSCR = 0x02;                                     //发送 SEND 命令
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;                                     //发送读 ACK 命令
    Wait();
}

char RecvData()
{
    I2CMSCR = 0x04;                                     //发送 RECV 命令
    Wait();
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;                                     //设置 ACK 信号
    I2CMSCR = 0x05;                                     //发送 ACK 命令
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;                                     //设置 NAK 信号
    I2CMSCR = 0x05;                                     //发送 ACK 命令
    Wait();
}
```

```
void Stop()
{
    I2CMSCR = 0x06;                         //发送 STOP 命令
    Wait();
}

void Delay()
{
    int i;

    for (i=0; i<3000; i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
}

void main()
{
    EAXFR = 1;                              //允许访问扩展的特殊寄存器，XFR
                                            //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                          //允许访问扩展的特殊寄存器，XFR
                                            //(8 位模式请使用这句，注释上一句)
    WTST = 0;                               //设置取程序代码等待时间，
                                            //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                              //设置访问片内的 xdata 速度，
                                            //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    I2CCFG = 0xe0;                          //使能 I2C 主机模式
    I2CMSST = 0x00;

    Start();                                //发送起始命令
    SendData(0xa0);                         //发送设备地址+写命令
    RecvACK();
    SendData(0x00);                         //发送存储地址高字节
    RecvACK();
    SendData(0x00);                         //发送存储地址低字节
    RecvACK();
    SendData(0x12);                         //写测试数据 1
    RecvACK();
    SendData(0x78);                         //写测试数据 2
    RecvACK();
    Stop();                                 //发送停止命令
```

```
    Delay();                                    //等待设备写数据

    Start();                                    //发送起始命令
    SendData(0xa0);                             //发送设备地址+写命令
    RecvACK();
    SendData(0x00);                             //发送存储地址高字节
    RecvACK();
    SendData(0x00);                             //发送存储地址低字节
    RecvACK();
    Start();                                    //发送起始命令
    SendData(0xa1);                             //发送设备地址+读命令
    RecvACK();
    P0 = RecvData();                            //读取数据 1
    SendACK();
    P2 = RecvData();                            //读取数据 2
    SendNAK();
    Stop();                                     //发送停止命令

    while (1);
}
```

28.5.3　I2C 主机模式访问 PCF8563

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

sbit        SDA             =     P1^4;
sbit        SCL             =     P1^5;

void Wait()
{
    while (!(I2CMSST & 0x40));
    I2CMSST &= ~0x40;
}

void Start()
{
    I2CMSCR = 0x01;                                     //发送 START 命令
    Wait();
}

void SendData(char dat)
{
    I2CTXD = dat;                                       //写数据到数据缓冲区
    I2CMSCR = 0x02;                                     //发送 SEND 命令
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;                                     //发送读 ACK 命令
    Wait();
}

char RecvData()
{
    I2CMSCR = 0x04;                                     //发送 RECV 命令
    Wait();
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;                                     //设置 ACK 信号
    I2CMSCR = 0x05;                                     //发送 ACK 命令
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;                                     //设置 NAK 信号
    I2CMSCR = 0x05;                                     //发送 ACK 命令
    Wait();
}
```

```
void Stop()
{
    I2CMSCR = 0x06;                         //发送 STOP 命令
    Wait();
}

void Delay()
{
    int i;

    for (i=0; i<3000; i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
}

void main()
{
    EAXFR = 1;                              //允许访问扩展的特殊寄存器，XFR
                                            //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                          //允许访问扩展的特殊寄存器，XFR
                                            //(8 位模式请使用这句，注释上一句)
    WTST = 0;                               //设置取程序代码等待时间，
                                            //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                              //设置访问片内的 xdata 速度，
                                            //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    I2CCFG = 0xe0;                          //使能 I2C 主机模式
    I2CMSST = 0x00;

    Start();                                //发送起始命令
    SendData(0xa2);                         //发送设备地址+写命令
    RecvACK();
    SendData(0x02);                         //发送存储地址
    RecvACK();
    SendData(0x00);                         //设置秒值
    RecvACK();
    SendData(0x00);                         //设置分钟值
    RecvACK();
    SendData(0x12);                         //设置小时值
    RecvACK();
    Stop();                                 //发送停止命令
```

```
    while (1)
    {
        Start();                    //发送起始命令
        SendData(0xa2);             //发送设备地址+写命令
        RecvACK();
        SendData(0x02);             //发送存储地址
        RecvACK();
        Start();                    //发送起始命令
        SendData(0xa3);             //发送设备地址+读命令
        RecvACK();
        P0 = RecvData();            //读取秒值
        SendACK();
        P2 = RecvData();            //读取分钟值
        SendACK();
        P3 = RecvData();            //读取小时值
        SendNAK();
        Stop();                     //发送停止命令

        Delay();
    }
}
```

28.5.4　I2C 从机模式（中断方式）

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

sbit        SDA             =     P1^4;
sbit        SCL             =     P1^5;

bit         isda;                                       //设备地址标志
bit         isma;                                       //存储地址标志
unsigned char               addr;
unsigned char edata         buffer[32];

void I2C_Isr() interrupt 24
{
    if (I2CSLST & 0x40)
    {
        I2CSLST &= ~0x40;                               //处理 START 事件
        isda = 1;                                       //若为重复起始信号时必须作此设置
    }
    else if (I2CSLST & 0x20)
    {
        I2CSLST &= ~0x20;                               //处理 RECV 事件
        if (isda)
        {
            isda = 0;                                   //处理 RECV 事件（RECV DEVICE ADDR）
        }
        else if (isma)
        {
            isma = 0;                                   //处理 RECV 事件（RECV MEMORY ADDR）
            addr = I2CRXD;
            I2CTXD = buffer[addr];
        }
        else
        {
            buffer[addr++] = I2CRXD;                    //处理 RECV 事件（RECV DATA）
        }
    }
    else if (I2CSLST & 0x10)
    {
        I2CSLST &= ~0x10;                               //处理 SEND 事件
        if (I2CSLST & 0x02)
        {
            I2CTXD = 0xff;                              //接收到 NAK 则停止读取数据
        }
        else
        {
            I2CTXD = buffer[++addr];                    //接收到 ACK 则继续读取数据
        }
    }
    else if (I2CSLST & 0x08)
    {
        I2CSLST &= ~0x08;                               //处理 STOP 事件
        isda = 1;
```

```
            isma = 1;
        }
}

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                           //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    I2CCFG = 0x81;                      //使能 I2C 从机模式
    I2CSLADR = 0x5a;                    //设置从机设备地址寄存器 I2CSLADR=0101_1010B
                                        //即 I2CSLADR[7:1]=010_1101B,MA=0B。
                                        //由于 MA 为 0,主机发送的设备地址必须与
                                        //I2CSLADR[7:1]相同才能访问此 I2C 从机设备。
                                        //主机若需要写数据则要发送 5AH(0101_1010B)
                                        //主机若需要读数据则要发送 5BH(0101_1011B)
    I2CSLST = 0x00;
    I2CSLCR = 0x78;                     //使能从机模式中断
    EA = 1;

    isda = 1;                           //用户变量初始化
    isma = 1;
    addr = 0;
    I2CTXD = buffer[addr];

    while (1);
}
```

28.5.5　I2C 从机模式（查询方式）

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

sbit        SDA             =     P1^4;
sbit        SCL             =     P1^5;

bit         isda;                                       //设备地址标志
bit         isma;                                       //存储地址标志
unsigned char               addr;
unsigned char edata         buffer[32];

void main()
{
    EAXFR = 1;                                          //允许访问扩展的特殊寄存器，XFR
                                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                      //允许访问扩展的特殊寄存器，XFR
                                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                           //设置取程序代码等待时间，
                                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                          //设置访问片内的 xdata 速度，
                                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    I2CCFG = 0x81;                                      //使能 I2C 从机模式
    I2CSLADR = 0x5a;                                    //设置从机设备地址寄存器 I2CSLADR=0101_1010B
                                                        //即 I2CSLADR[7:1]=010_1101B,MA=0B。
                                                        //由于 MA 为 0,主机发送的设备地址必须与
                                                        //I2CSLADR[7:1]相同才能访问此 I2C 从机设备。
                                                        //主机若需要写数据则要发送 5AH(0101_1010B)
                                                        //主机若需要读数据则要发送 5BH(0101_1011B)
    I2CSLST = 0x00;
    I2CSLCR = 0x00;                                     //禁止从机模式中断

    isda = 1;                                           //用户变量初始化
    isma = 1;
    addr = 0;
    I2CTXD = buffer[addr];

    while (1)
    {
```

```
        if (I2CSLST & 0x40)
        {
            I2CSLST &= ~0x40;                   //处理 START 事件
            isda = 1;                           //若为重复起始信号时必须作此设置
        }
        else if (I2CSLST & 0x20)
        {
            I2CSLST &= ~0x20;                   //处理 RECV 事件
            if (isda)
            {
                isda = 0;                       //处理 RECV 事件（RECV DEVICE ADDR）
            }
            else if (isma)
            {
                isma = 0;                       //处理 RECV 事件（RECV MEMORY ADDR）
                addr = I2CRXD;
                I2CTXD = buffer[addr];
            }
            else
            {
                buffer[addr++] = I2CRXD;        //处理 RECV 事件（RECV DATA）
            }
        }
        else if (I2CSLST & 0x10)
        {
            I2CSLST &= ~0x10;                   //处理 SEND 事件
            if (I2CSLST & 0x02)
            {
                I2CTXD = 0xff;                  //接收到 NAK 则停止读取数据
            }
            else
            {
                I2CTXD = buffer[++addr];        //接收到 ACK 则继续读取数据
            }
        }
        else if (I2CSLST & 0x08)
        {
            I2CSLST &= ~0x08;                   //处理 STOP 事件
            isda = 1;
            isma = 1;
        }
    }
}
```

28.5.6　测试 I2C 从机模式代码的主机代码

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

sbit        SDA             =     P1^4;
sbit        SCL             =     P1^5;

void Wait()
{
    while (!(I2CMSST & 0x40));
    I2CMSST &= ~0x40;
}

void Start()
{
    I2CMSCR = 0x01;                                     //发送 START 命令
    Wait();
}

void SendData(char dat)
{
    I2CTXD = dat;                                       //写数据到数据缓冲区
    I2CMSCR = 0x02;                                     //发送 SEND 命令
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;                                     //发送读 ACK 命令
    Wait();
}

char RecvData()
{
    I2CMSCR = 0x04;                                     //发送 RECV 命令
    Wait();
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;                                     //设置 ACK 信号
    I2CMSCR = 0x05;                                     //发送 ACK 命令
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;                                     //设置 NAK 信号
    I2CMSCR = 0x05;                                     //发送 ACK 命令
    Wait();
}
```

```
void Stop()
{
    I2CMSCR = 0x06;                         //发送 STOP 命令
    Wait();
}

void main()
{   EAXFR = 1;                              //允许访问扩展的特殊寄存器，XFR
                                            //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                          //允许访问扩展的特殊寄存器，XFR
                                            //(8 位模式请使用这句，注释上一句)
    WTST = 0;                               //设置取程序代码等待时间，
                                            //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                              //设置访问片内的 xdata 速度，
                                            //赋值为 0 表示用最快速度访问，不增加额外的等待时间
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    I2CCFG = 0xe0;                          //使能 I2C 主机模式
    I2CMSST = 0x00;

    Start();                                //发送起始命令
    SendData(0x5a);                         //发送设备地址(010_1101B)+写命令(0B)
    RecvACK();
    SendData(0x00);                         //发送存储地址
    RecvACK();
    SendData(0x12);                         //写测试数据 1
    RecvACK();
    SendData(0x78);                         //写测试数据 2
    RecvACK();
    Stop();                                 //发送停止命令

    Start();                                //发送起始命令
    SendData(0x5a);                         //发送设备地址(010_1101B)+写命令(0B)
    RecvACK();
    SendData(0x00);                         //发送存储地址高字节
    RecvACK();
    Start();                                //发送起始命令
    SendData(0x5b);                         //发送设备地址(010_1101B)+读命令(1B)
    RecvACK();
    P0 = RecvData();                        //读取数据 1
    SendACK();
    P2 = RecvData();                        //读取数据 2
    SendNAK();
    Stop();                                 //发送停止命令

    while (1);
}
```

29 16 位高级 PWM 定时器，支持正交编码

产品线	高级 PWM
Ai8051U 系列	●

Ai8051U 系列的单片机内部集成了 8 通道 16 位高级 PWM 定时器，分成两组周期可不同的 PWM，分别命名为 PWMA 和 PWMB，可分别单独设置。第一组 PWM/PWMA 可配置成 4 组互补/对称/死区控制的 PWM 或捕捉外部信号，第二组 PWM/PWMB 可配置成 4 路 PWM 输出或捕捉外部信号。

第一组 PWM/PWMA 的时钟频率可以是系统时钟经过寄存器 PWMA_PSCRH 和 PWMA_PSCRL 进行分频后的时钟，分频值可以是 1~65535 之间的任意值。第二组 PWM/PWMB 的时钟频率可以是系统时钟经过寄存器 PWMB_PSCRH 和 PWMB_PSCRL 进行分频后的时钟，分频值可以是 1~65535 之间的任意值。两组 PWM 的时钟频率可分别独立设置。

第一组 PWM 定时器/PWMA 有 4 个通道（PWM1P/PWM1N、PWM2P/PWM2N、PWM3P/PWM3N、PWM4P/PWM4N），每个通道都可独立实现 PWM 输出（可设置带死区的互补对称 PWM 输出）、捕获和比较功能；第二组 PWM 定时器/PWMB 有 4 个通道（PWM5、PWM6、PWM7、PWM8），每个通道也可独立实现 PWM 输出、捕获和比较功能。两组 PWM 定时器唯一的区别是第一组可输出带死区的互补对称 PWM，而第二组只能输出单端的 PWM，其他功能完全相同。下面关于高级 PWM 定时器的介绍只以第一组为例进行说明。

当使用第一组 PWM 定时器输出 PWM 波形时，可单独使能 PWM1P/PWM2P/PWM3P/PWM4P 输出，也可单独使能 PWM1N/PWM2N/PWM3N/PWM4N 输出。例如：若单独使能了 PWM1P 输出，则 PWM1N 就不能再独立输出，除非 PWM1P 和 PWM1N 组成一组互补对称输出。PWMA 的 4 路输出是可分别独立设置的，例如：可单独使能 PWM1P 和 PWM2N 输出，也可单独使能 PWM2N 和 PWM3N 输出。若需要使用第一组 PWM 定时器进行捕获功能或者测量脉宽时，输入信号只能从每路的正端输入，即只有 PWM1P/PWM2P/PWM3P/PWM4P 才有捕获功能和测量脉宽功能。

两组高级 PWM 定时器对外部信号进行捕获时，可选择上升沿捕获或者下降沿捕获。如果需要同时捕获上升沿和下降沿，则可将输入信号同时接入到两路 PWM，使能其中一路捕获上升沿，另外一路捕获下降沿即可。**更强悍的是，将外部输入信号同时接入到两路 PWM 时，可同时捕获信号的周期值和占空比值**。

三种硬件 PWM 比较：

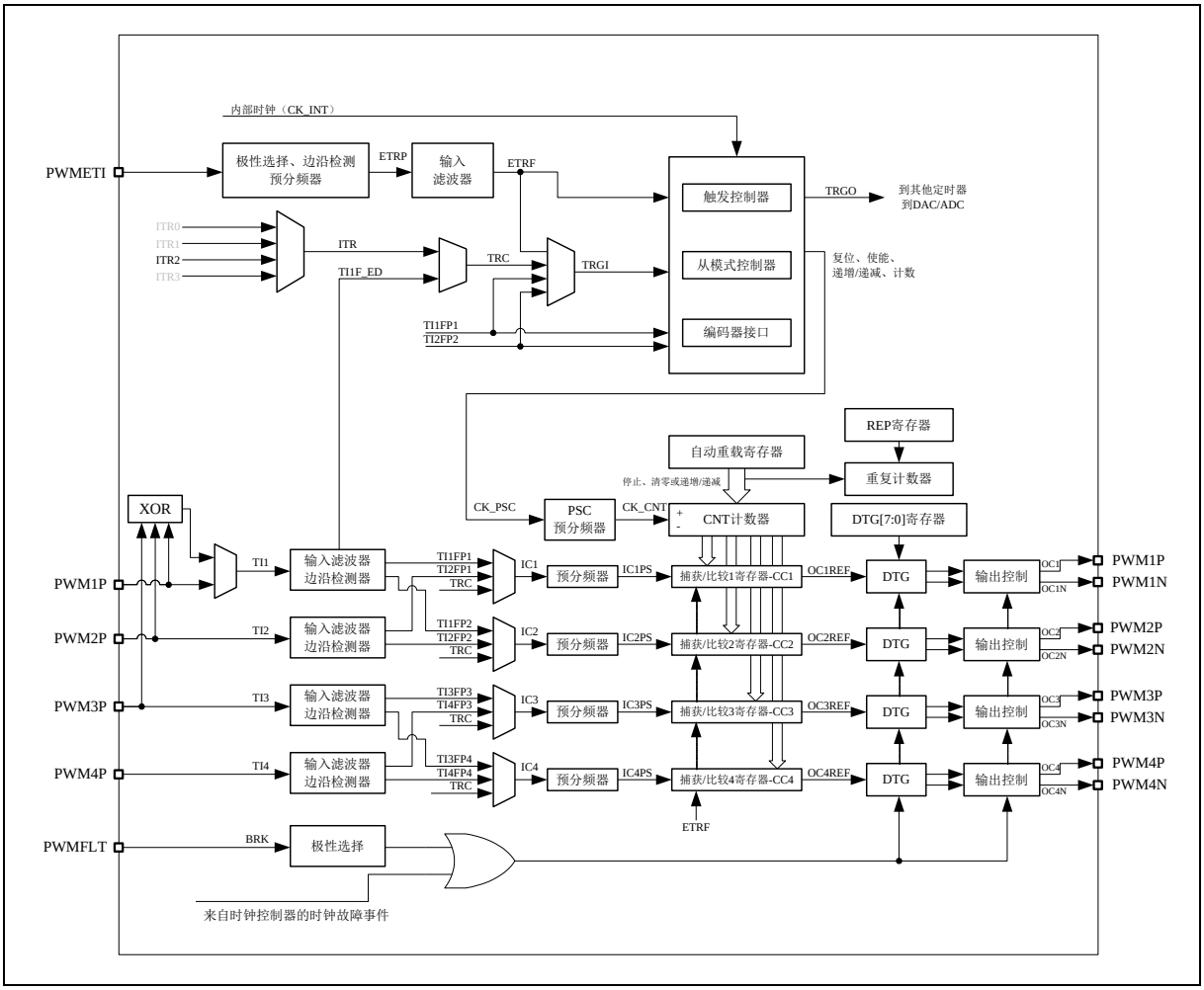
兼容传统 8051 的 PCA/CCP/PWM：可输出 PWM 波形、捕获外部输入信号以及输出高速脉冲。可对外输出 6 位/7 位/8 位/10 位的 PWM 波形，6 位 PWM 波形的频率为 PCA 模块时钟源频率/64；7 位 PWM 波形的频率为 PCA 模块时钟源频率/128；8 位 PWM 波形的频率为 PCA 模块时钟源频率/256；10 位 PWM 波形的频率为 PCA 模块时钟源频率/1024。捕获外部输入信号，可捕获上升沿、下降沿或者同时捕获上升沿和下降沿。

AI32G 系列的 15 位增强型 PWM：只能对外输出 PWM 波形，无输入捕获功能。对外输出 PWM 的频率以及占空比均可任意设置。通过软件干预，可实现多路互补/对称/带死区的 PWM 波形。有外部异常检测功能以及实时触发 ADC 转换功能。

AI32G/AI8H/Ai8051U 系列的 16 位高级 PWM 定时器：是目前功能最强的 PWM，可对外输出任意频率以及任意占空比的 PWM 波形。无需软件干预即可输出互补/对称/带死区的 PWM 波形。能捕获外部输入信号，可捕获上升沿、下降沿或者同时捕获上升沿和下降沿，测量外部波形时，可同时测量波形的周期值和占空比值。有正交编码功能、外部异常检测功能以及实时触发 ADC 转换功能。

下面的说明中，PWMA 代表第一组 PWM 定时器，PWMB 代表第二组 PWM 定时器

29.1 高级 PWM 定时器（PWMA）内部结构框图



TI1：外部时钟输入信号 1（PWM1P 管脚信号或者 PWM1P/PWM2P/PWM3P 相异或后的信号）
TI1F：经过 IC1F 数字滤波后的 TI1 信号
TI1FP：经过 CC1P/CC2P 边沿检测器后的 TI1F 信号
TI1F_ED：TI1F 的边沿信号
TI1FP1：经过 CC1P 边沿检测器后的 TI1F 信号
TI1FP2：经过 CC2P 边沿检测器后的 TI1F 信号
IC1：通过 CC1S 选择的通道 1 的捕获输入信号
OC1REF：输出通道 1 输出的参考波形（中间波形）
OC1：通道 1 的主输出信号（经过 CC1P 极性处理后的 OC1REF 信号）
OC1N：通道 1 的互补输出信号（经过 CC1NP 极性处理后的 OC1REF 信号）
CC1：通道 1 的捕获/比较寄存器

TI2：外部时钟输入信号 2（PWM2P 管脚信号）
TI2F：经过 IC2F 数字滤波后的 TI2 信号
TI2F_ED：TI2F 的边沿信号
TI2FP：经过 CC1P/CC2P 边沿检测器后的 TI2F 信号
TI2FP1：经过 CC1P 边沿检测器后的 TI2F 信号
TI2FP2：经过 CC2P 边沿检测器后的 TI2F 信号
IC2：通过 CC2S 选择的通道 2 的捕获输入信号

OC2REF：输出通道 2 输出的参考波形（中间波形）
OC2：通道 2 的主输出信号（经过 CC2P 极性处理后的 OC2REF 信号）
OC2N：通道 2 的互补输出信号（经过 CC2NP 极性处理后的 OC2REF 信号）
CC2：通道 2 的捕获/比较寄存器

TI3：外部时钟输入信号 3（PWM3P 管脚信号）
TI3F：经过 IC3F 数字滤波后的 TI3 信号
TI3F_ED：TI3F 的边沿信号
TI3FP：经过 CC3P/CC4P 边沿检测器后的 TI3F 信号
TI3FP3：经过 CC3P 边沿检测器后的 TI3F 信号
TI3FP4：经过 CC4P 边沿检测器后的 TI3F 信号
IC3：通过 CC3S 选择的通道 3 的捕获输入信号
OC3REF：输出通道 3 输出的参考波形（中间波形）
OC3：通道 3 的主输出信号（经过 CC3P 极性处理后的 OC3REF 信号）
OC3N：通道 3 的互补输出信号（经过 CC3NP 极性处理后的 OC3REF 信号）
CC3：通道 3 的捕获/比较寄存器

TI4：外部时钟输入信号 4（PWM4P 管脚信号）
TI4F：经过 IC4F 数字滤波后的 TI4 信号
TI4F_ED：TI4F 的边沿信号
TI4FP：经过 CC3P/CC4P 边沿检测器后的 TI4F 信号
TI4FP3：经过 CC3P 边沿检测器后的 TI4F 信号
TI4FP4：经过 CC4P 边沿检测器后的 TI4F 信号
IC4：通过 CC4S 选择的通道 4 的捕获输入信号
OC4REF：输出通道 4 输出的参考波形（中间波形）
OC4：通道 4 的主输出信号（经过 CC4P 极性处理后的 OC4REF 信号）
OC4N：通道 4 的互补输出信号（经过 CC4NP 极性处理后的 OC4REF 信号）
CC4：通道 4 的捕获/比较寄存器

ITR1：内部触发输入信号 1
ITR2：内部触发输入信号 2

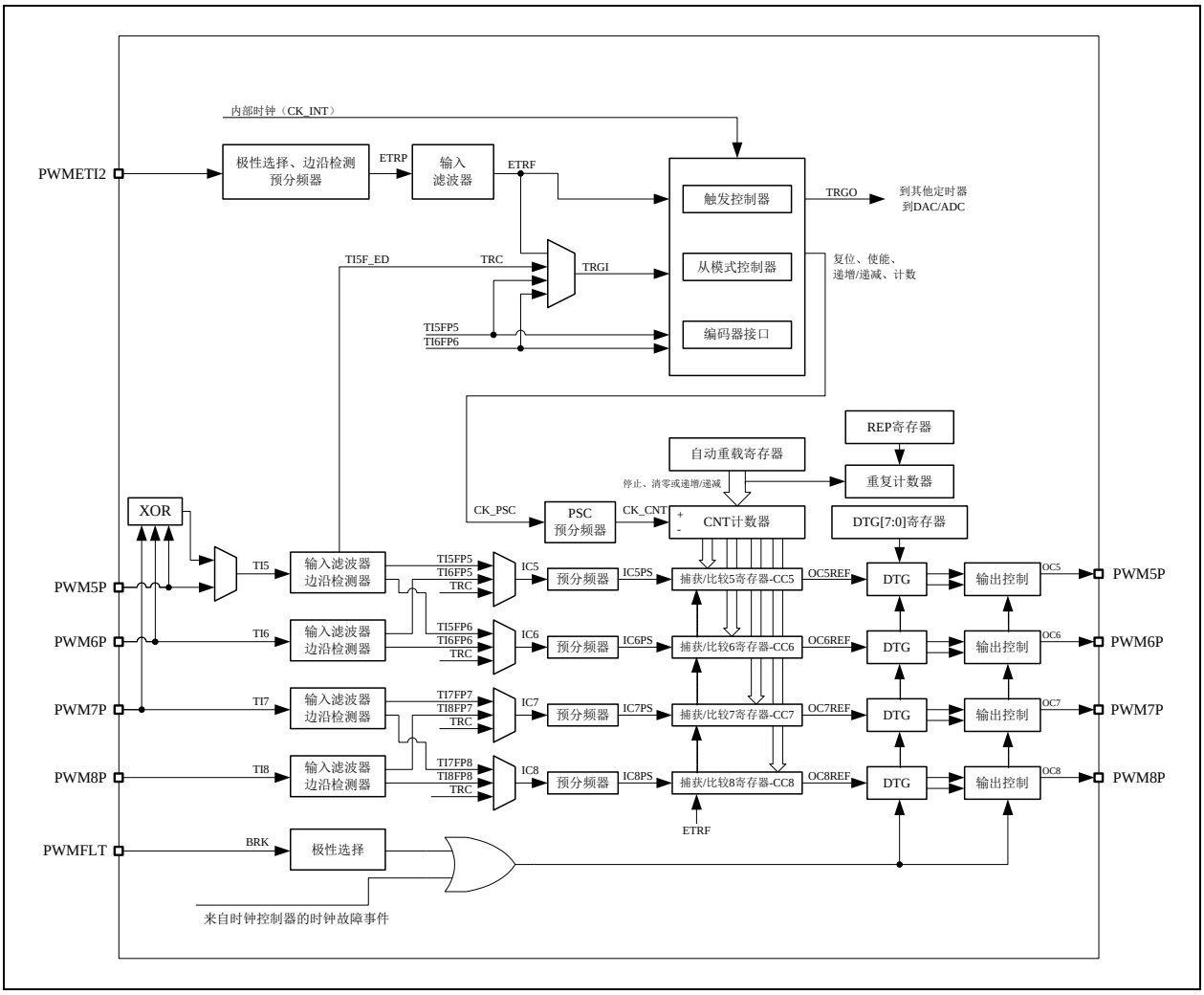
TRC：固定为 TI1_ED
TRGI：经过 TS 多路选择器后的触发输入信号
TRGO：经过 MMS 多路选择器后的触发输出信号

ETR：外部触发输入信号（PWMETI1 管脚信号）
ETRP：经过 ETP 边沿检测器以及 ETPS 分频器后的 ETR 信号
ETRF：经过 ETF 数字滤波后的 ETRP 信号

BRK：刹车输入信号（PWMFLT）

CK_PSC：预分频时钟，PWMA_PSCR 预分频器的输入时钟
CK_CNT：PWMA_PSCR 预分频器的输出时钟，PWM 定时器的时钟

29.2 高级 PWM 定时器（PWMB）内部结构框图



TI5：外部时钟输入信号 5（PWM5 管脚信号或者 PWM5/PWM6/PWM7 相异或后的信号）
TI5F：经过 IC5F 数字滤波后的 TI5 信号
TI5FP：经过 CC5P/CC6P 边沿检测器后的 TI5F 信号
TI5F_ED：TI5F 的边沿信号
TI5FP5：经过 CC5P 边沿检测器后的 TI5F 信号
TI5FP6：经过 CC6P 边沿检测器后的 TI5F 信号
IC5：通过 CC5S 选择的通道 5 的捕获输入信号
OC5REF：输出通道 5 输出的参考波形（中间波形）
OC5：通道 5 的主输出信号（经过 CC5P 极性处理后的 OC5REF 信号）
CC5：通道 5 的捕获/比较寄存器

TI6：外部时钟输入信号 6（PWM6 管脚信号）
TI6F：经过 IC6F 数字滤波后的 TI6 信号
TI6F_ED：TI6F 的边沿信号
TI6FP：经过 CC5P/CC6P 边沿检测器后的 TI6F 信号
TI6FP5：经过 CC5P 边沿检测器后的 TI6F 信号
TI6FP6：经过 CC6P 边沿检测器后的 TI6F 信号
IC6：通过 CC6S 选择的通道 6 的捕获输入信号
OC6REF：输出通道 6 输出的参考波形（中间波形）

OC6：通道 6 的主输出信号（经过 CC6P 极性处理后的 OC6REF 信号）
CC6：通道 6 的捕获/比较寄存器

TI7：外部时钟输入信号 7（PWM7 管脚信号）
TI7F：经过 IC7F 数字滤波后的 TI7 信号
TI7F_ED：TI7F 的边沿信号
TI7FP：经过 CC7P/CC8P 边沿检测器后的 TI7F 信号
TI7FP7：经过 CC7P 边沿检测器后的 TI7F 信号
TI7FP8：经过 CC8P 边沿检测器后的 TI7F 信号
IC7：通过 CC7S 选择的通道 7 的捕获输入信号
OC7REF：输出通道 7 输出的参考波形（中间波形）
OC7：通道 7 的主输出信号（经过 CC7P 极性处理后的 OC7REF 信号）
CC7：通道 7 的捕获/比较寄存器

TI8：外部时钟输入信号 8（PWM8 管脚信号）
TI8F：经过 IC8F 数字滤波后的 TI8 信号
TI8F_ED：TI8F 的边沿信号
TI8FP：经过 CC7P/CC8P 边沿检测器后的 TI8F 信号
TI8FP7：经过 CC7P 边沿检测器后的 TI8F 信号
TI8FP8：经过 CC8P 边沿检测器后的 TI8F 信号
IC8：通过 CC8S 选择的通道 8 的捕获输入信号
OC8REF：输出通道 8 输出的参考波形（中间波形）
OC8：通道 8 的主输出信号（经过 CC8P 极性处理后的 OC8REF 信号）
CC8：通道 8 的捕获/比较寄存器

29.3 简介

PWMB 与 PWMA 唯一的区别是 PWMA 可输出带死区的互补对称 PWM，而 PWMB 则只能输出单端的 PWM，其他功能完全相同。下面关于高级 PWM 的介绍只以 PWMA 为例进行说明。

PWMA 由一个 16 位的自动装载计数器组成，它由一个可编程的预分频器驱动。

PWMA 适用于许多不同的用途：

- 基本的定时
- 测量输入信号的脉冲宽度（输入捕获）
- 产生输出波形（输出比较，PWM 和单脉冲模式）
- 对应与不同事件（捕获，比较，溢出，刹车，触发）的中断
- 与 PWMB 或者外部信号（外部时钟，复位信号，触发和使能信号）同步

PWMA 广泛的适用于各种控制应用中，包括那些需要中间对齐模式 PWM 的应用，该模 式支持互补输出和死区时间控制。

PWMA 的时钟源可以是内部时钟，也可以是外部的信号，可以通过配置寄存器来进行选择。

29.4 主要特性

PWMA 的特性包括：

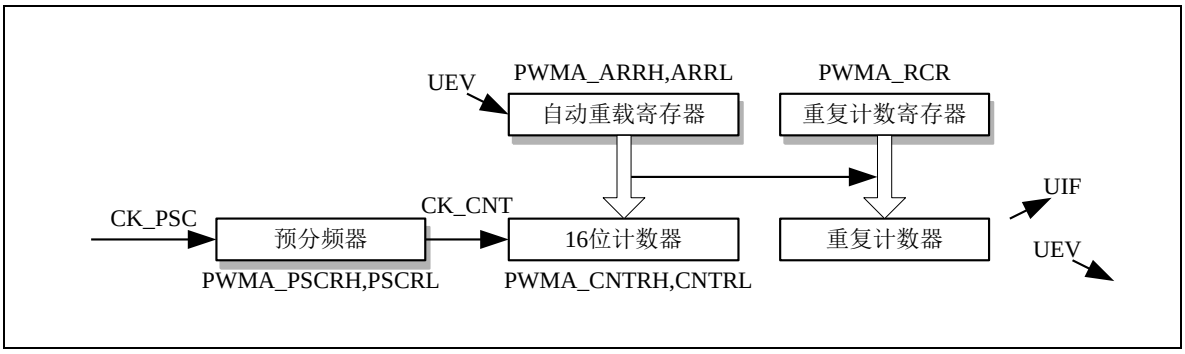
- 16 位向上、向下、向上/下自动装载计数器
- 允许在指定数目的计数器周期之后更新定时器寄存器的重复计数器
- 16 位可编程（可以实时修改）预分频器，计数器时钟频率的分频系数为 1~65535 之间的任意数值
- 同步电路，用于使用外部信号控制定时器以及定时器互联
- 多达 4 个独立通道可以配置成：
  - 输入捕获
  - 输出比较
  - PWM 输出（边缘或中间对齐模式）
  - 六步 PWM 输出
  - 单脉冲模式输出
  - 支持 4 个死区时间可编程的通道上互补输出
- 刹车输入信号（PWMFLT）可以将定时器输出信号置于复位状态或者一个确定状态
- 外部触发输入引脚（PWMETI）
- 产生中断的事件包括：
  - 更新：计数器向上溢出/向下溢出，计数器初始化（通过软件或者内部/外部触发）
  - 触发事件（计数器启动、停止、初始化或者由内部/外部触发计数）
  - 输入捕获
  - 输出比较
  - 刹车信号输入

29.5 时基单元

PWMA 的时基单元包含：

- 16 位向上/向下计数器
- 16 位自动重载寄存器
- 重复计数器
- 预分频器

PWMA 时基单元



16 位计数器、预分频器、自动重载寄存器和重复计数器寄存器都可以通过软件进行读写操作。

自动重载寄存器由预装载寄存器和影子寄存器组成。

可在在两种模式下写自动重载寄存器：

- 自动预装载已使能（PWMA_CR1 寄存器的 ARPE 位为 1）。

在此模式下，写入自动重载寄存器的数据将被保存在预装载寄存器中，并在下一个更新事件（UEV）时传送到影子寄存器。

- 自动预装载已禁止（PWMA_CR1 寄存器的 ARPE 位为 0）。

在此模式下，写入自动重载寄存器的数据将立即写入影子寄存器。

更新事件的产生条件：

- 计数器向上或向下溢出。
- 软件置位了 PWMA_EGR 寄存器的 UG 位。
- 时钟/触发控制器产生了触发事件。

在预装载使能时（ARPE=1），如果发生了更新事件，预装载寄存器中的数值（PWMA_ARR）将写入影子寄存器中，并且 PWMA_PSCR 寄存器中的值将写入预分频器中。

置位 PWMA_CR1 寄存器的 UDIS 位将禁止更新事件（UEV）。

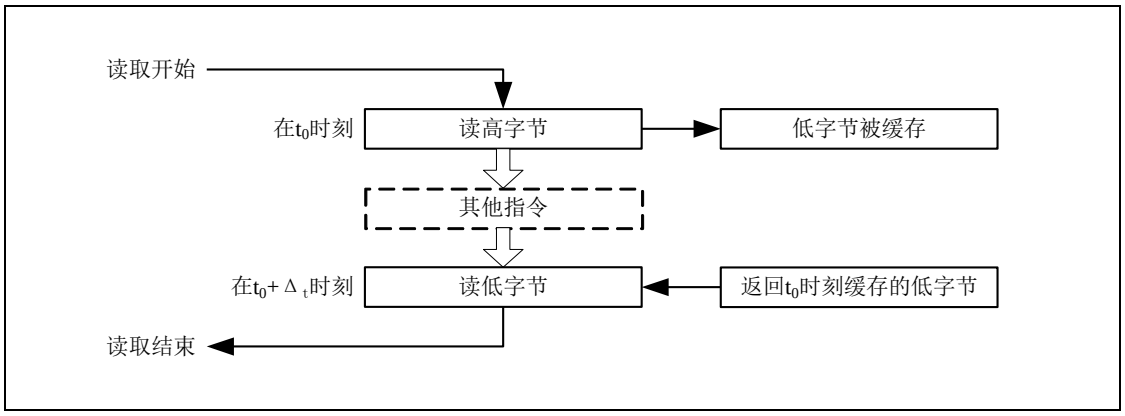
预分频器的输出 CK_CNT 驱动计数器，而 CK_CNT 仅在 IM1_CR1 寄存器的计数器使能位（CEN）被置位时才有效。

注意：实际的计数器在 CEN 位使能的一个时钟周期后才开始计数。

29.5.1 读写 16 位计数器

写计数器的操作没有缓存，在任何时候都可以写 PWMA_CNTRH 和 PWMA_CNTRL 寄存器，因此为避免写入了错误的数值，一般建议不要在计数器运行时写入新的数值。

读计数器的操作带有 8 位的缓存。用户必须先读定时器的高字节，在用户读了高字节后，低字节将被自动缓存，缓存的数据将会一直保持直到 16 位数据的读操作完成。



29.5.2 16 位 PWMA_ARR 寄存器的写操作

预装载寄存器中的值将写入 16 位的 PWMA_ARR 寄存器中，此操作由两条指令完成，每条指令写 入 1 个字节。必须先写高字节，后写低字节。

影子寄存器在写入高字节时被锁定，并保持到低字节写完。

29.5.3 预分频器

预分频器的实现：

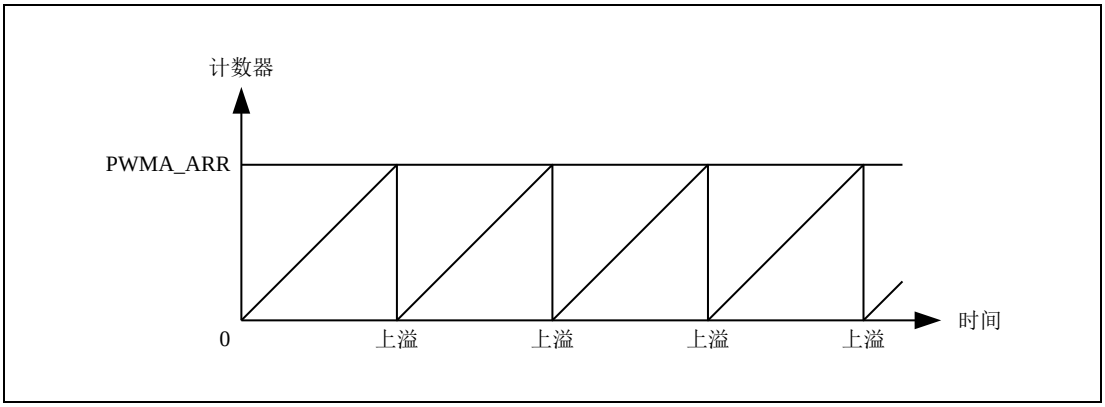
PWMA 的预分频器基于一个由 16 位寄存器（PWMA_PSCR）控制的 16 位计数器。由于这个控制寄存器带有缓冲器，因此它能够在运行时被改变。预分频器可以将计数器的时钟频率按 1 到 65536 之间的任意值分频。预分频器的值由预装载寄存器写入，保存了当前使用值的影子寄存器在低字节写入时被载入。由于需两次单独的写操作来写 16 位寄存器，因此必须保证高字节先写入。新的预分频器的值在下一次更新事件到来时被采用。对 PWMA_PSCR 寄存器的读操作通过预装载寄存器完成。

计数器的频率计算公式：$f_{CK_CNT} = f_{CK_PSC} / (PSCR[15:0] + 1)$

29.5.4　向上计数模式

在向上计数模式中，计数器从 0 计数到用户定义的比较值（PWMA_ARR 寄存器的值），然后重新从 0 开始计数并产生一个计数器溢出事件，此时如果 PWMA_CR1 寄存器的 UDIS 位是 0，将会产生一个更新事件（UEV）。

向上计数模式的计数器



通过软件方式或者通过使用触发控制器置位 PWMA_EGR 寄存器的 UG 位同样也可以产生一个更新事件。

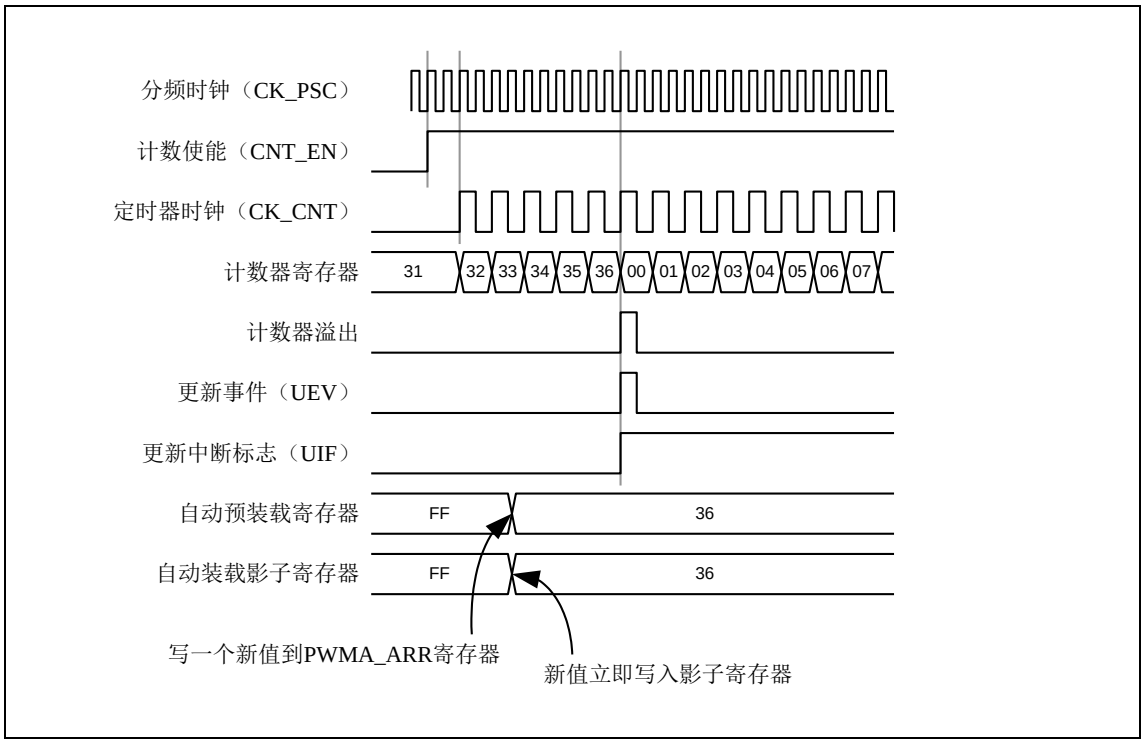
使用软件置位 PWMA_CR1 寄存器的 UDIS 位，可以禁止更新事件，这样可以避免在更新预装载寄存器时更新影子寄存器。在 UDIS 位被清除之前，将不产生更新事件。但是在应该产生更新事件时，计数器仍会被清 0，同时预分频器的计数也被清 0（但预分频器的数值不变）。此外，如果设置了 PWMA_CR1 寄存器中的 URS 位（选择更新请求），设置 UG 位将产生一个更新事件 UEV，但硬件不设置 UIF 标志（即不产生中断请求）。这是为了避免在捕获模式下清除计数器时，同时产生更新和捕获中断。

当发生一个更新事件时，所有的寄存器都被更新，硬件依据 URS 位同时设置更新标志位（PWMA_SR 寄存器的 UIF 位）：

- 自动装载影子寄存器被重新置入预装载寄存器的值（PWMA_ARR）。
- 预分频器的缓存器被置入预装载寄存器的值（PWMA_PSC 寄存器的内容）。

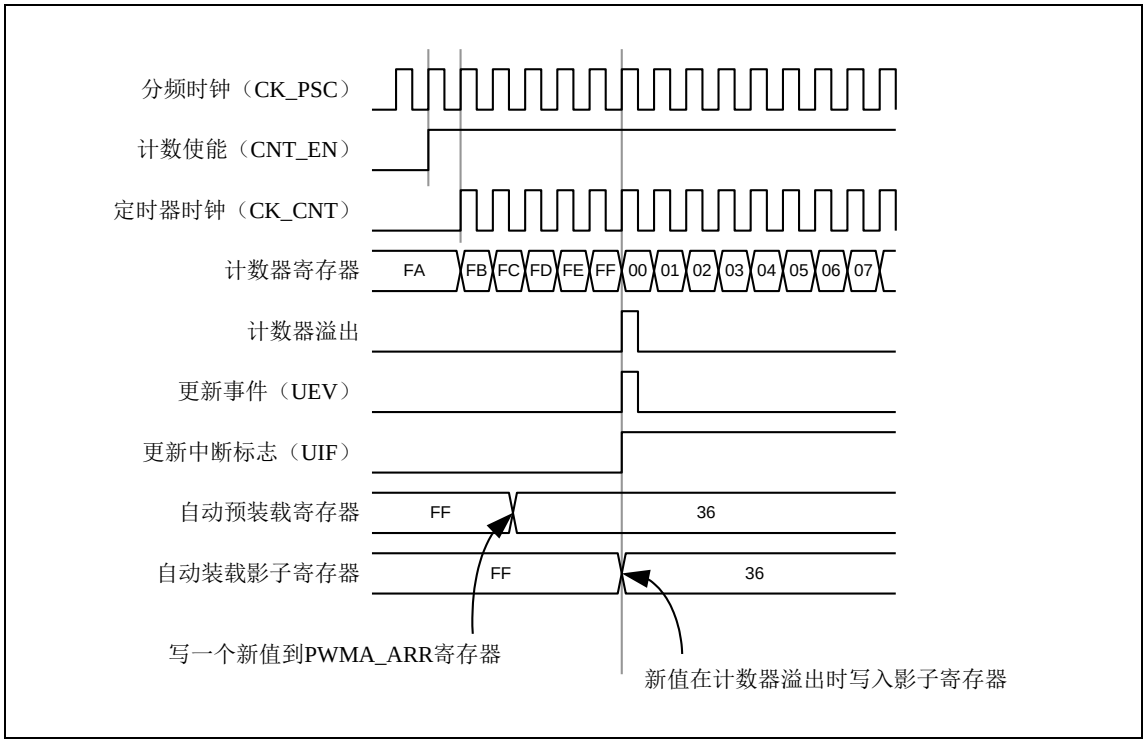
下图给出一些例子，说明当 PWMA_ARR=0x36 时，计数器在不同时钟频率下的动作。图中预分频为 2，因此计数器的时钟（CK_CNT）频率是预分频时钟（CK_PSC）频率的一半。图中禁止了自动装载功能（ARPE=0），所以在计数器达到 0x36 时，计数器溢出，影子寄存器立刻被更新，同时产生一个更新事件。

当 ARPE=0（ARR 不预装载），预分频为 2 时的计数器更新：



下图的预分频为 1，因此 CK_CNT 的频率与 CK_PSC 一致。图中使能了自动重载（ARPE=1），所以在计数器达到 0xFF 产生溢出。0x36 将在溢出时被写入，同时产生一个更新事件。

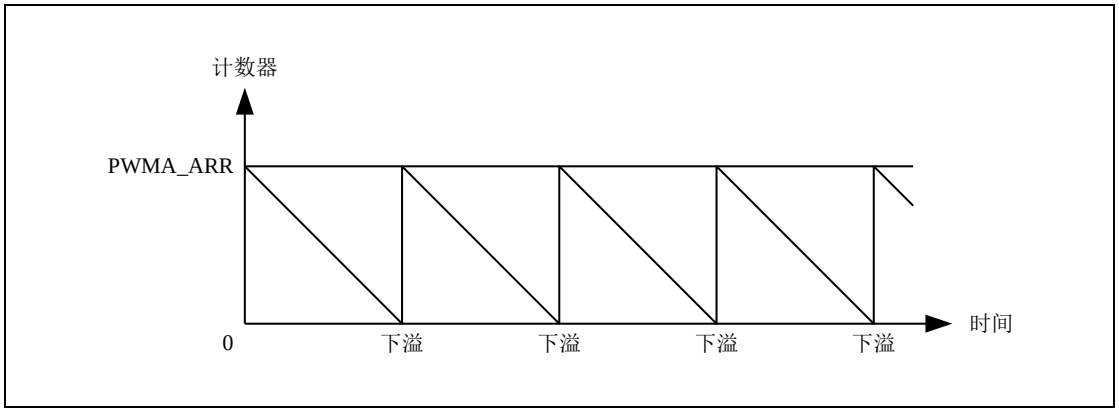
ARPE=1(PWMA_ARR 预装载) 预分频为 1 时的计数器更新：



29.5.5 向下计数模式

在向下模式中，计数器从自动装载的值（PWMA_ARR 寄存器的值）开始向下计数到 0，然后再从自动装载的值重新开始计数，并产生一个计数器向下溢出事件。如果 PWMA_CR1 寄存器的 UDIS 位被清除，还会产生一个更新事件（UEV）。

向下计数模式的计数器



通过软件方式或者通过使用触发控制器置位 PWMA_EGR 寄存器的 UG 位同样也可以产生一个更新事件。

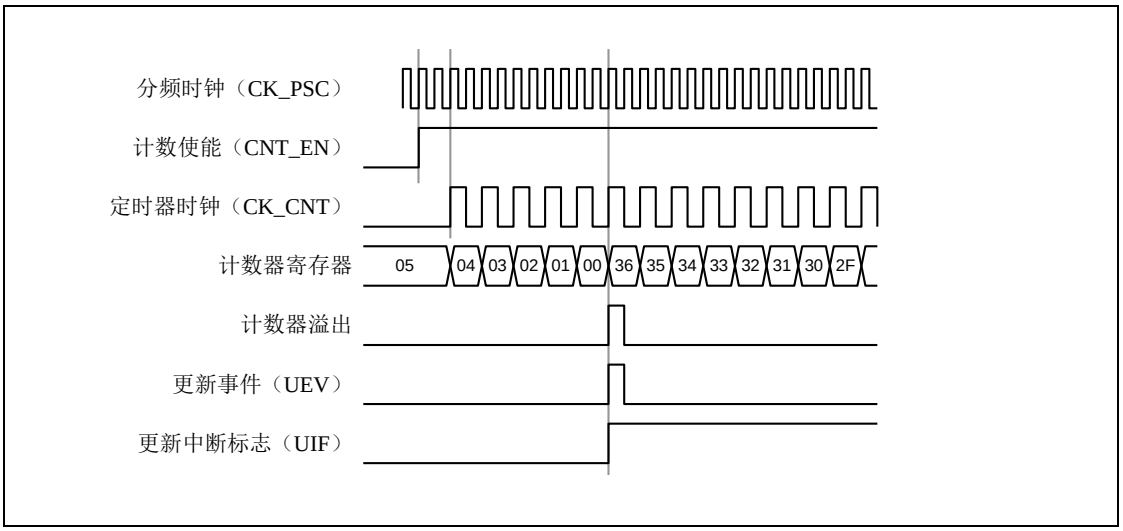
置位 PWMA_CR1 寄存器的 UDIS 位可以禁止 UEV 事件。这样可以避免在更新预装载寄存器时更新影子寄存器。因此 UDIS 位清除之前不会产生更新事件。然而，计数器仍会从当前自动加载值重新开始计数，并且预分频器的计数器重新从 0 开始（但预分频器不能被修改）。此外，如果设置了 PWMA_CR1 寄存器中的 URS 位（选择更新请求），设置 UG 位将产生一个更新事件 UEV 但不设置 UIF 标志（因此不产生中断），这是为了避免在发生捕获事件并清除计数器时，同时产生更新和捕获中断。

当发生更新事件时，所有的寄存器都被更新，硬件依据 URS 位同时设置更新标志位（PWMA_SR 寄存器的 UIF 位）：

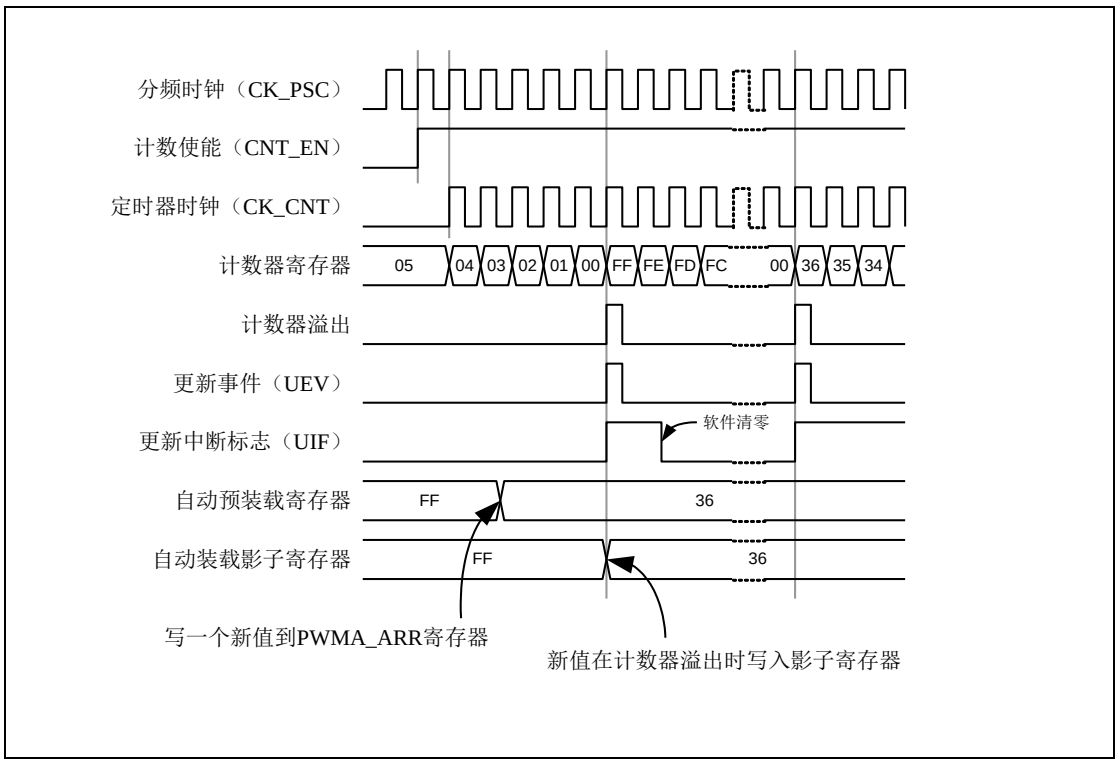
- 自动装载影子寄存器被重新置入预装载寄存器的值（PWMA_ARR）。
- 预分频器的缓存器被置入预装载寄存器的值（PWMA_PSC 寄存器的内容）。

以下是一些当 PWMA_ARR=0x36 时，计数器在不同时钟频率下的图表。下图描述了在向下计数模式下，预装载不使能时新的数值在下个周期时被写入。

ARPE=0（ARR 不预装载），预分频为 2 时的计数器更新：



ARPE=1（ARR 预装载），预分频为 1 时的计数器更新

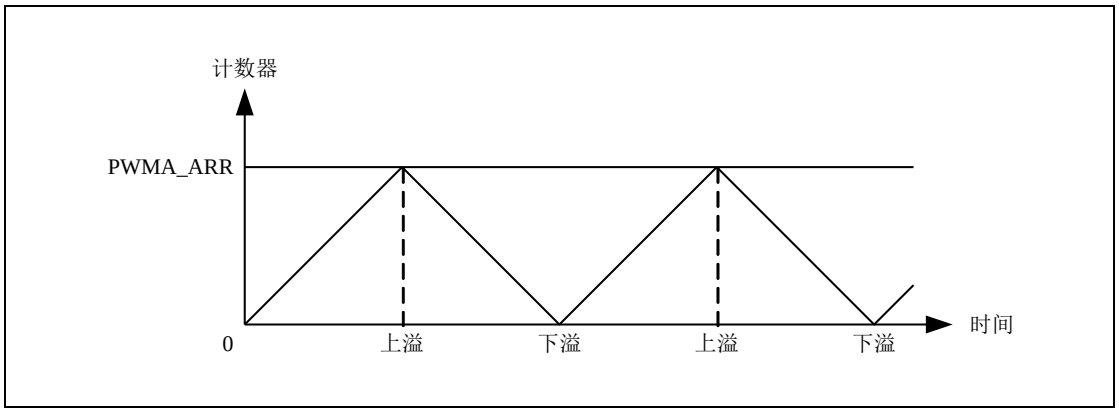


29.5.6 中间对齐模式（向上/向下计数）

在中央对齐模式，计数器从 0 开始计数到 PWMA_ARR 寄存器的值-1，产生一个计数器上溢事件，然后从 PWMA_ARR 寄存器的值向下计数到 1 并且产生一个计数器下溢事件；然后再从 0 开始重新计数。

在此模式下，不能写入 PWMA_CR1 中的 DIR 方向位。它由硬件更新并指示当前的计数方向。

中央对齐模式的计数器



如果定时器带有重复计数器，在重复了指定次数（PWMA_RCR[15:0]的值）的向上和向下溢出之后会产生更新事件（UEV）。否则每一次的向上向下溢出都会产生更新事件。

通过软件方式或者通过使用触发控制器置位 PWMA_EGR 寄存器的 UG 位同样也可以产生一个更新事件。此时，计数器重新从 0 开始计数，预分频器也重新从 0 开始计数。

设置 PWMA_CR1 寄存器中的 UDIS 位可以禁止 UEV 事件。这样可以避免在更新预装载寄存器时更新影子寄存器。因此 UDIS 位被清为 0 之前不会产生更新事件。然而，计数器仍会根据当前自动重加载的值，继续向上或向下计数。如果定时器带有重复计数器，由于重复寄存器没有双重的缓冲，新的重复数值将立刻生效，因此在修改时需要小心。此外，如果设置了 PWMA_CR1 寄存器中的 URS 位（选择更新请求），设置 UG 位将产生一个更新事件 UEV 但不设置 UIF 标志（因此不产生中断），这是为了避免在发生捕获事件并清除计数器时，同时产生更新和捕获中断。

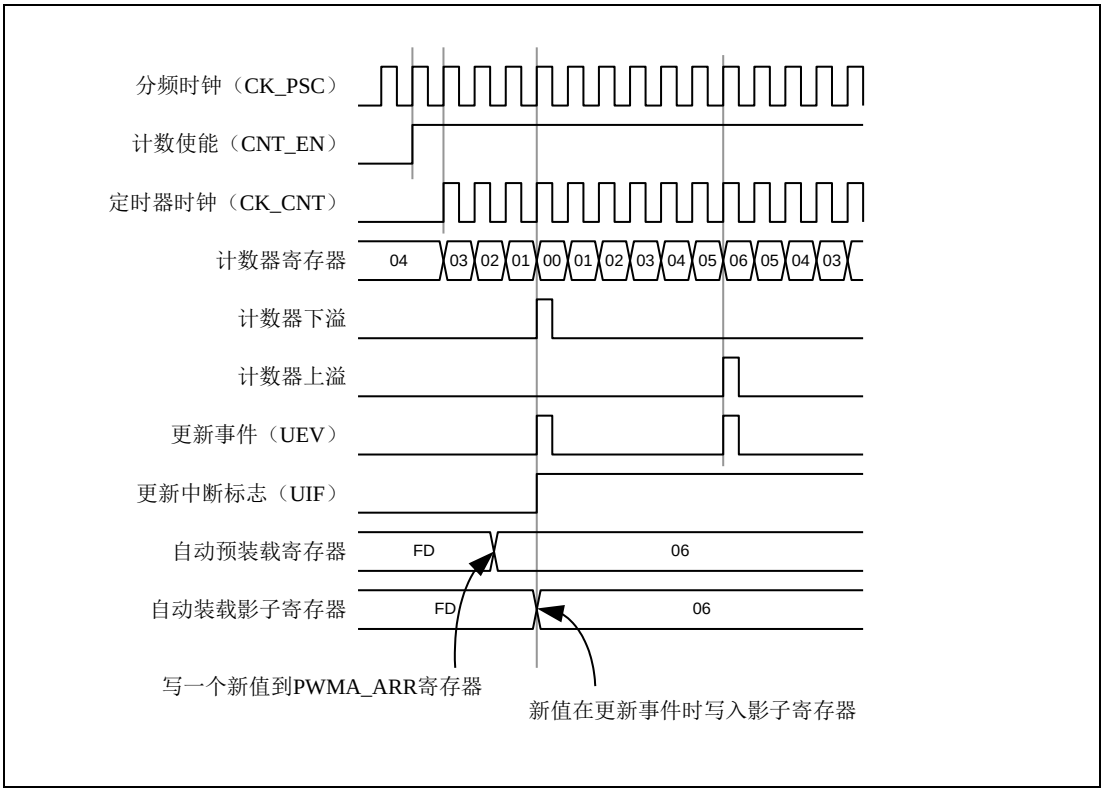
当发生更新事件时，所有的寄存器都被更新，硬件依据 URS 位更新标志位（PWMA_SR 寄存器中的 UIF 位）：

- 预分频器的缓存器被加载为预装载（PWMA_PSC 寄存器）的值。
- 当前的自动加载寄存器被更新为预装载值（PWMA_ARR 寄存器中的内容）。

要注意到如果因为计数器溢出而产生更新，自动重装载寄存器将在计数器重载入之前被更新，因此下一个周期才是预期的值（计数器被装载为新的值）。

以下是一些计数器在不同时钟频率下的操作的例子：

内部时钟分频因子为 1，PWMA_ARR=0x6，ARPE＝1



使用中央对齐模式的提示：

- 启动中央对齐模式时，计数器将按照原有的向上/向下的配置计数。也就是说 PWMA_CR1 寄存器中的 DIR 位将决定计数器是向上还是向下计数。此外，软件不能同时修改 DIR 位和 CMS 位的值。
- 不推荐在中央对齐模式下，计数器正在计数时写计数器的值，这将导致不能预料的后果。具体的说：
  - 向计数器写入了比自动装载值更大的数值时（PWMA_CNT>PWMA_ARR），但计数器的计数方向不发生改变。例如计数器已经向上溢出，但计数器仍然向上计数。
  - 向计数器写入了 0 或者 PWMA_ARR 的值，但更新事件不发生。
- 安全使用中央对齐模式的计数器的方法是在启动计数器之前先用软件（置位 PWMA_EGR 寄存器的 UG 位）产生一个更新事件，并且不在计数器计数时修改计数器的值。

29.5.7　重复计数器

时基单元解释了计数器向上/向下溢出时更新事件（UEV）是如何产生的，然而事实上它只能在重复计数器的值达到 0 的时候产生。这个特性对产生 PWM 信号非常有用。

这意味着在每 N 次计数上溢或下溢时，数据从预装载寄存器传输到影子寄存器（PWMA_ARR 自动重载入寄存器，PWMA_PSC 预装载寄存器，还有在比较模式下的捕获/比较寄存器 PWMA_CCRx），N 是 PWMA_RCR[15:0]重复计数寄存器中的值。

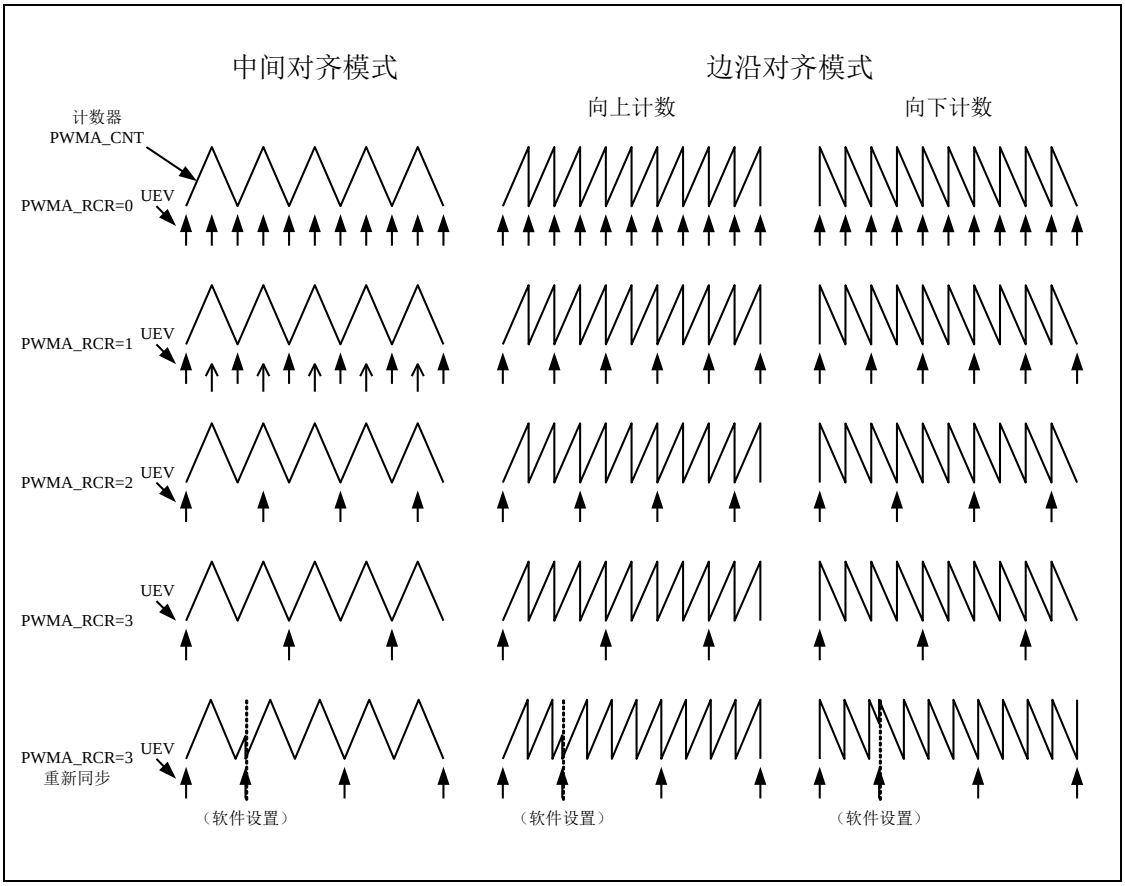
重复计数器在下述任一条件成立时递减：

- 向上计数模式下每次计数器向上溢出时
- 向下计数模式下每次计数器向下溢出时
- 中央对齐模式下每次上溢和每次下溢时。

虽然这样限制了 PWM 的最大循环周期为 128，但它能够在每个 PWM 周期 2 次更新占空比。在中央对齐模式下，因为波形是对称的，如果每个 PWM 周期中仅刷新一次比较寄存器，则最大的分辨率为 2*tCK_PSC。

重复计数器是自动加载的，重复速率由 PWMA_RCR[15:0]寄存器的值定义。当更新事件由软件产生（通过设置 PWMA_EGR 中的 UG 位）或者通过硬件的时钟/触发控制器产生，则无论重复计数器的值是多少，立即发生更新事件，并且 PWMA_RCR[15:0]寄存器中的内容被重载入到重复计数器。

不同模式下更新速率的例子，及 PWMA_RCR[15:0]的寄存器设置



29.6 时钟/触发控制器

时钟/触发控制器允许用户选择计数器的时钟源，输入触发信号和输出信号，

29.6.1 预分频时钟（CK_PSC）

时基单元的预分频时钟（CK_PSC）可以由以下源提供：

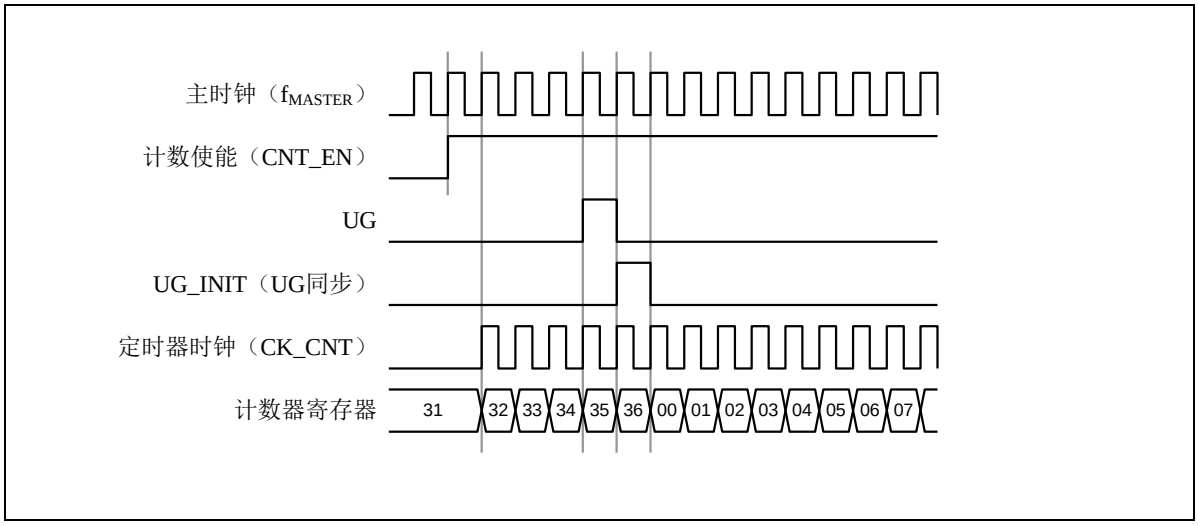
- 内部时钟（fMASTER）
- 外部时钟模式 1：外部时钟输入（TIx）
- 外部时钟模式 2：外部触发输入 ETR
- 内部触发输入（ITRx）：使用一个定时器做为另一个定时器的预分频时钟。

29.6.2 内部时钟源（fMASTER）

如果同时禁止了时钟/触发模式控制器和外部触发输入（PWMA_SMCR 寄存器的 SMS=000，PWMA_ETR 寄存器的 ECE＝0），则 CEN、DIR 和 UG 位是实际上的控制位，并且只能被软件修改（UG 位仍被自动清除）。一旦 CEN 位被写成 1，预分频器的时钟就由内部时钟提供。

下图描述了控制电路和向上计数器在普通模式下，不带预分频器时的操作。

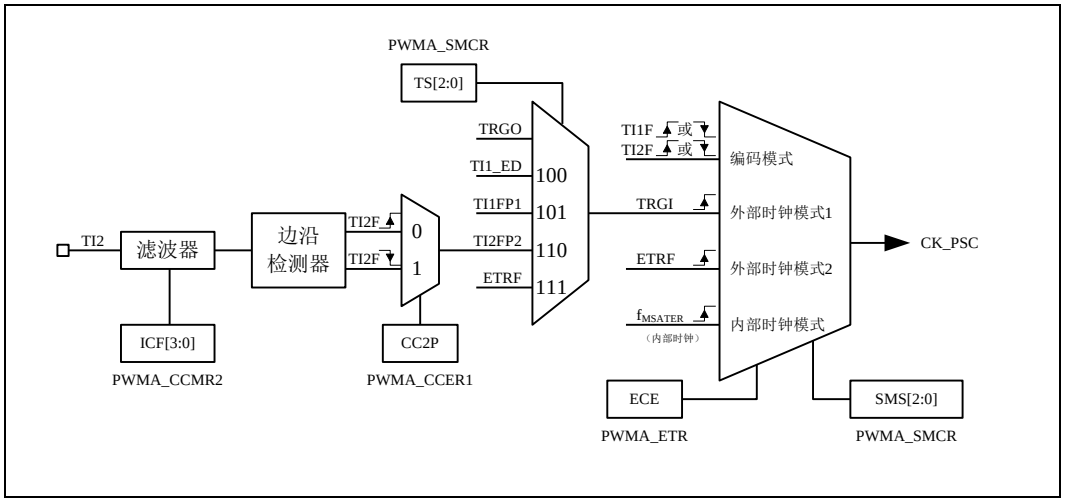
普通模式下的控制电路，f_{MASTER} 分频因子为 1



29.6.3　外部时钟源模式 1

当 PWMA_SMCR 寄存器的 SMS=111 时，此模式被选中。计数器可以在选定输入端的每个上升沿或下降沿计数。

TI2 外部时钟连接例子



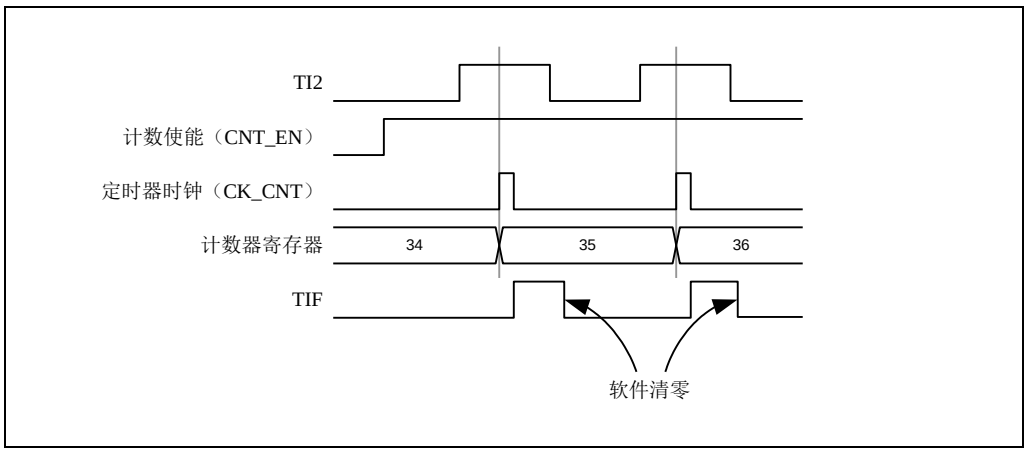
例如，要配置向上计数器在 TI2 输入端的上升沿计数，使用下列步骤：

1. 配置 PWMA_CCMR2 寄存器的 CC2S=01，使用通道 2 检测 TI2 输入
2. 配置 PWMA_CCMR2 寄存器的 IC2F[3:0]位，选择输入滤波器带宽（如果不需要滤波器，保持 IC2F=0000）注：捕获预分频器不用作触发，所以不需要对它进行配置，同样也不需要配置 TI2S 位，他们仅用来选择输入捕获源。
3. 配置 PWMA_CCER1 寄存器的 CC2P=0，选定上升沿极性
4. 配置 PWMA_SMCR 寄存器的 SMS=111，配置计数器使用外部时钟模式 1
5. 配置 PWMA_SMCR 寄存器的 TS=110，选定 TI2 作为输入源
6. 设置 PWMA_CR1 寄存器的 CEN=1，启动计数器

当上升沿出现在 TI2，计数器计数一次，且触发标识位（PWMA_SR1 寄存器的 TIF 位）被置 1，如果使能了中断（在 PWMA_IER 寄存器中配置）则会产生中断请求。

在 TI2 的上升沿和计数器实际时钟之间的延时取决于在 TI2 输入端的重新同步电路。

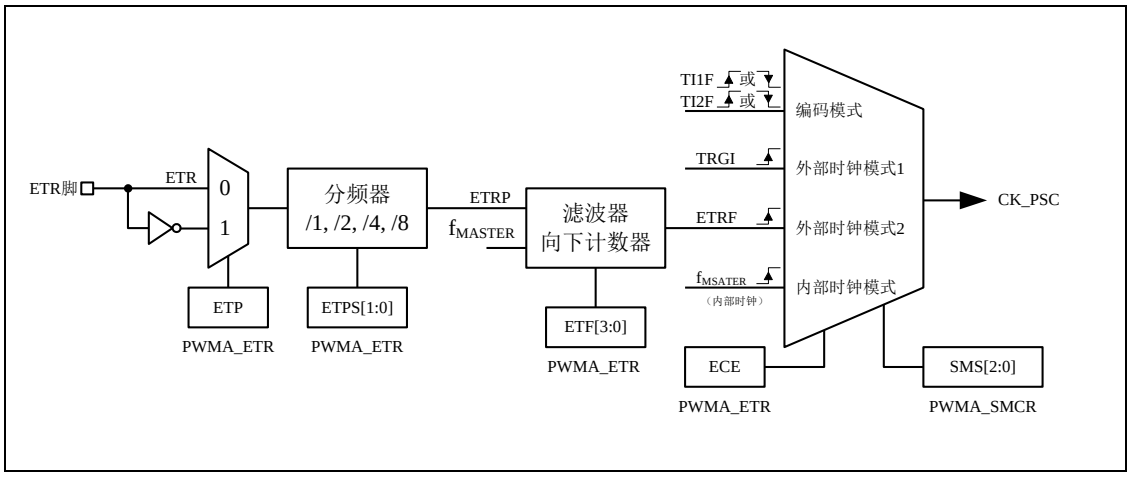
外部时钟模式 1 下的控制电路



29.6.4　外部时钟源模式 2

计数器能够在外部触发输入 ETR 信号的每一个上升沿或下降沿计数。将 PWMA_ETR 寄存器的 ECE 位写 1，即可选定此模式。

外部触发输入的总体框图：

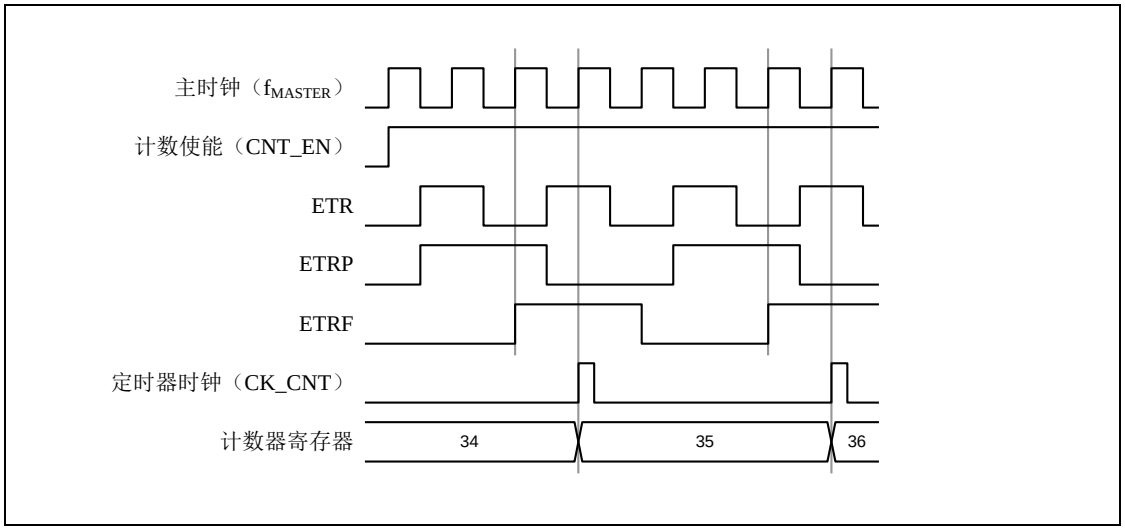


例如，要配置计数器在 ETR 信号的每 2 个上升沿时向上计数一次，需使用下列步骤：

1. 本例中不需要滤波器，配置 PWMA_ETR 寄存器的 ETF[3:0]=0000
2. 设置预分频器，配置 PWMA_ETR 寄存器的 ETPS[1:0]=01
3. 选择 ETR 的上升沿检测，配置 PWMA_ETR 寄存器的 ETP=0
4. 开启外部时钟模式 2，配置 PWMA_ETR 寄存器中的 ECE=1
5. 启动计数器，写 PWMA_CR1 寄存器的 CEN=1

计数器在每 2 个 ETR 上升沿计数一次。

外部时钟模式 2 下的控制电路



29.6.5 触发同步

PWMA 的计数器使用三种模式与外部的触发信号同步：

- 标准触发模式
- 复位触发模式
- 门控触发模式

标准触发模式

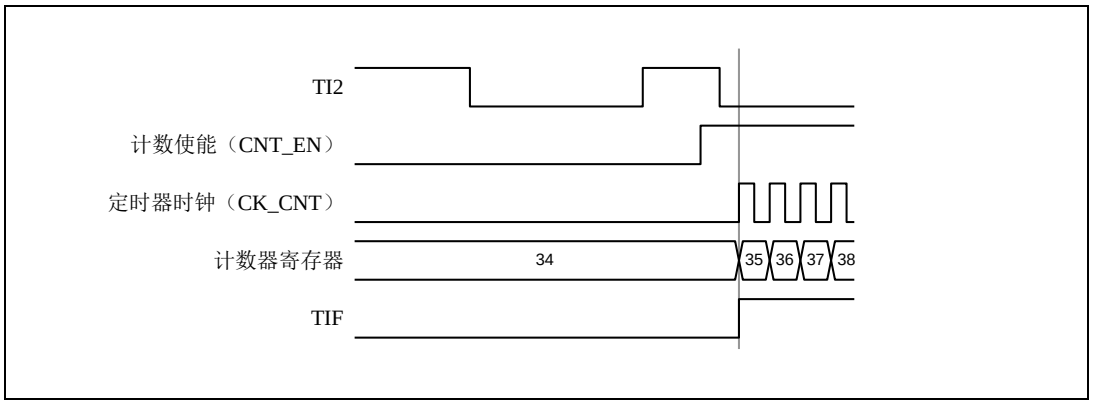
计数器的使能（CEN）依赖于选中的输入端上的事件。

在下面的例子中，计数器在 TI2 输入的上升沿开始向上计数：

1. 配置 PWMA_CCER1 寄存器的 CC2P＝0，选择 TI2 的上升沿做为触发条件。
2. 配置 PWMA_SMCR 寄存器的 SMS=110，选择计数器为触发模式。配置 PWMA_SMCR 寄存器的 TS=110，选择 TI2 作为输入源。

当 TI2 出现一个上升沿时，计数器开始在内部时钟驱动下计数，同时置位 TIF 标志。TI2 上升沿和计数器启动计数之间的延时取决于 TI2 输入端的重同步电路。

标准触发模式的控制电路



复位触发模式

在发生一个触发输入事件时，计数器和它的预分频器能够重新被初始化。同时，如果 PWMA_CR1 寄存器的 URS 位为低，还产生一个更新事件 UEV，然后所有的预装载寄存器（PWMA_ARR，PWMA_CCRx）都会被更新。

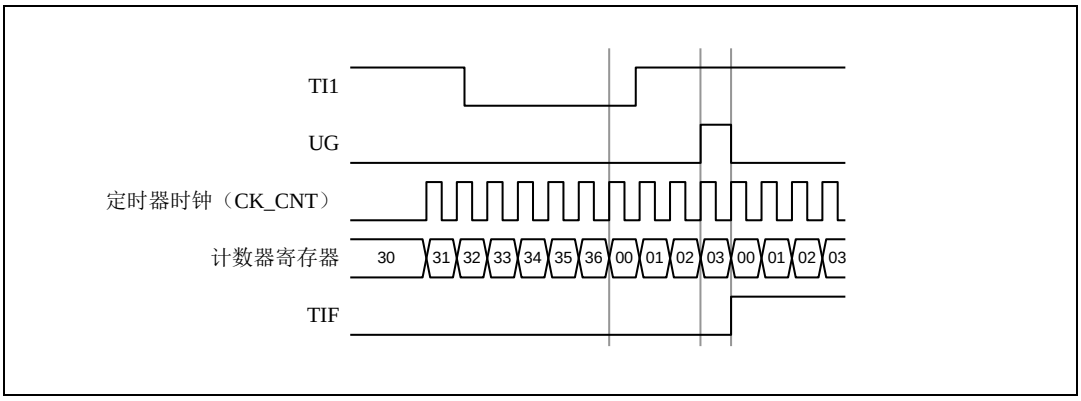
在以下的例子中，TI1 输入端的上升沿导致向上计数器被清零：

1. 配置 PWMA_CCER1 寄存器的 CC1P=0 来选择 TI1 的极性（只检测 TI1 的上升沿）。
2. 配置 PWMA_SMCR 寄存器的 SMS=100，选择定时器为复位触发模式。配置 PWMA_SMCR 寄存器的 TS=101，选择 TI1 作为输入源。
3. 配置 PWMA_CR1 寄存器的 CEN=1，启动计数器。

计数器开始依据内部时钟计数，然后正常计数直到 TI1 出现一个上升沿。此时，计数器被清零然后从 0 重新开始计数。同时，触发标志（PWMA_SR1 寄存器的 TIF 位）被置位，如果使能了中断（PWMA_IER 寄存器的 TIE 位），则产生一个中断请求。

下图显示当自动重装载寄存器 PWMA_ARR=0x36 时的动作。在 TI1 上升沿和计数器的实际复位之间的延时取决于 TI1 输入端的重同步电路。

复位触发模式下的控制电路



门控触发模式

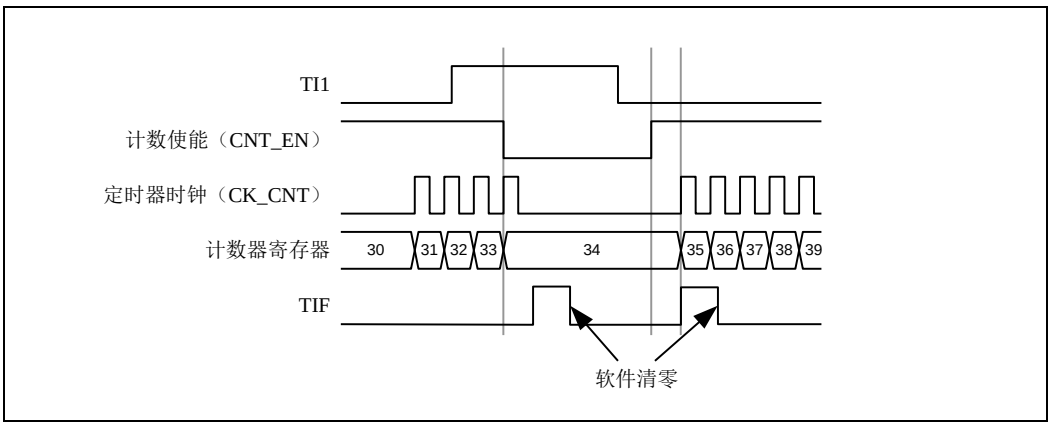
计数器由选中的输入端信号的电平使能。

在如下的例子中，计数器只在 TI1 为低时向上计数：

1. 配置 PWMA_CCER1 寄存器的 CC1P=1 来确定 TI1 的极性（只检测 TI1 上的低电平）。
2. 配置 PWMA_SMCR 寄存器的 SMS=101，选择定时器为门控触发模式，配置 PWMA_SMCR 寄存器中 TS=101，选择 TI1 作为输入源。
3. 配置 PWMA_CR1 寄存器的 CEN=1，启动计数器（在门控模式下，如果 CEN=0，则计数器不能启动，不论触发输入电平如何）。

只要 TI1 为低，计数器开始依据内部时钟计数，一旦 TI1 变高则停止计数。当计数器开始或停止时 TIF 标志位都会被置位。TI1 上升沿和计数器实际停止之间的延时取决于 TI1 输入端的重同步电路。

门控触发模式下的控制电路



外部时钟模式 2 联合触发模式

外部时钟模式 2 可以与另一个输入信号的触发模式一起使用。例如，ETR 信号被用作外部时钟的输入，另一个输入信号可用作触发输入（支持标准触发模式，复位触发模式和门控触发模式）。注意不能通过 PWMA_SMCR 寄存器的 TS 位把 ETR 配置成 TRGI。

在下面的例子中，一旦在 TI1 上出现一个上升沿，计数器即在 ETR 的每一个上升沿向上计数一次：

1. 通过 PWMA_ETR 寄存器配置外部触发输入电路。配置 ETPS=00 禁止预分频，配置 ETP=0 监测 ETR 信号的上升沿，配置 ECE=1 使能外部时钟模式 2。
2. 配置 PWMA_CCER1 寄存器的 CC1P=0 来选择 TI1 的上升沿触发。
3. 配置 PWMA_SMCR 寄存器的 SMS=110 来选择定时器为触发模式。配置 PWMA_SMCR 寄存器的

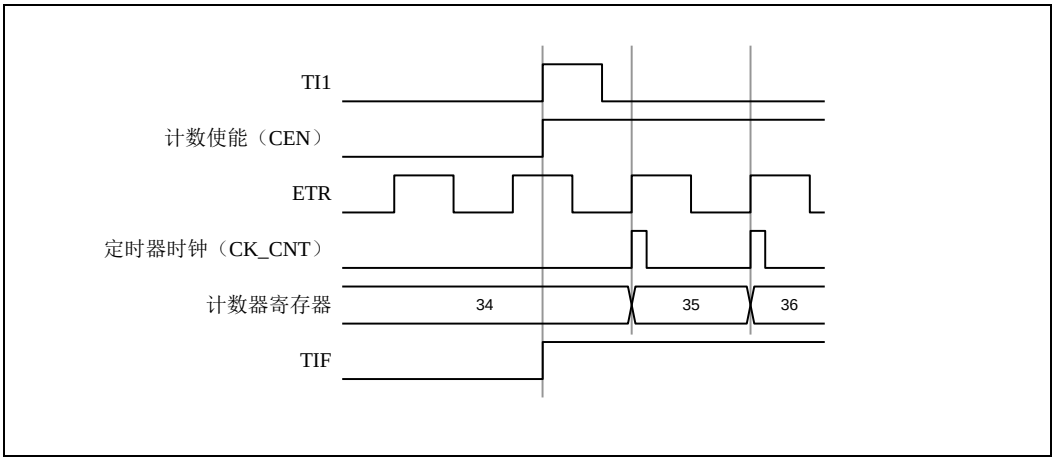
TS=101 来选择 TI1 作为输入源。

当 TI1 上出现一个上升沿时，TIF 标志被设置，计数器开始在 ETR 的上升沿计数。

TI1 信号的上升沿和计数器实际时钟之间的延时取决于 TI1 输入端的重同步电路。

ETR 信号的上升沿和计数器实际时钟之间的延时取决于 ETRP 输入端的重同步电路。

外部时钟模式 2+触发模式下的控制电路



29.6.6　与 PWMB 同步

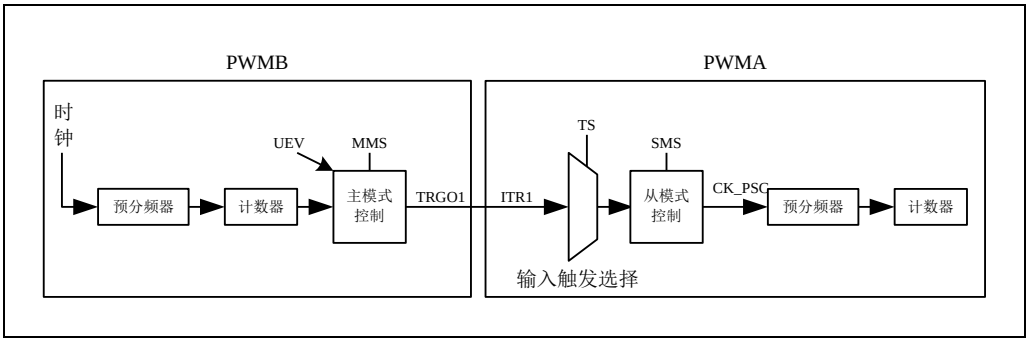
在芯片中，定时器在内部互相联结，用于定时器的同步或链接。当某个定时器配置成主模式时，可以输出触发信号（TRGO）到那些配置为从模式的定时器来完成复位操作、启动操作、停止操作或者作为那些定时器的驱动时钟。

使用 PWMB 的 TRGO 作为 PWMA 的预分频时钟

例如，用户可以配置 PWMB 作为 PWMA 的预分频时钟，需进行如下配置：

1. 配置 PWMB 为主模式，使得在每个更新事件（UEV）时输出周期性的触发信号。配置 PWMB_CR2 寄存器的 MMS=010，使每个更新事件时 TRGO 能输出一个上升沿。
2. PWMB 输出的 TRGO 信号链接到 PWMA。PWMA 需要配置成触发从模式，使用 ITR2 作为输入触发信号。以上操作可以通过配置 PWMA_SMCR 寄存器的 TS=010 实现。
3. 配置 PWMA_SMCR 寄存器的 SMS=111 将时钟/触发控制器设置为外部时钟模式 1。此操作将使 PWMB 输出的周期性触发信号 TRGO 的上升沿驱动 PWMA 的时钟。
4. 最后，置位 PWMB 的 CEN 位（PWMB_CR1 寄存器中），使能两个 PWM。

主/触发从模式的定时器例子



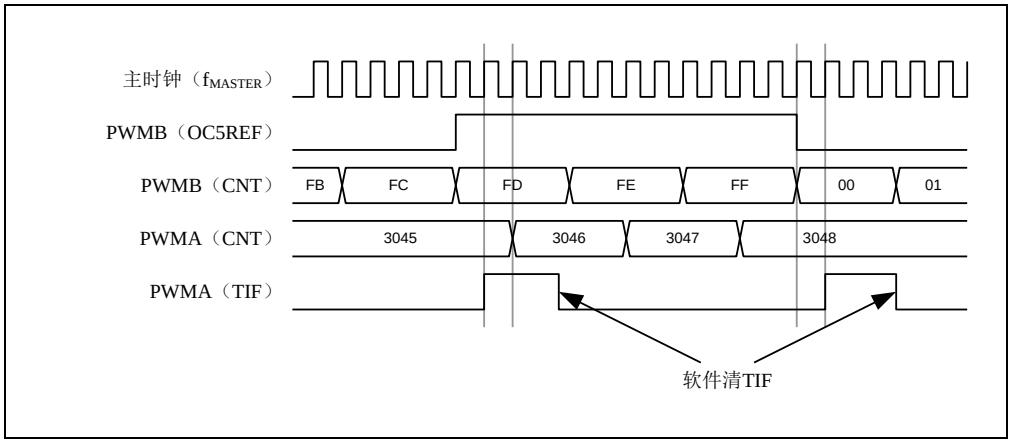
使用 PWMB 使能 PWMA

在本例中，我们用 PWMB 的比较输出使能 PWMA。PWMA 仅在 PWMB 的 OC1REF 信号为高时按照自己的驱动时钟计数。两个 PWM 都使用 4 分频的 f_{MASTER} 为时钟（$f_{CK_CNT} = f_{MASTER}/4$）。

1. 配置 PWMB 为主模式，将比较输出信号（OC5REF）作为触发信号输出。（配置 PWMB_CR2 寄存器的 MMS=100）。
2. 配置 PWMB 的 OC5REF 信号的波形（PWMB_CCMR1 寄存器）。
3. 配置 PWMA 把 PWMB 的输出作为自己的触发输入信号（配置 PWMA_SMCR 寄存器的 TS=010）。
4. 配置 PWMA 为门控触发模式（配置 PWMA_SMCR 寄存器的 SMS=101）。
5. 置位 CEN 位（PWMA_CR1 寄存器），使能 PWMA。
6. 置位 CEN 位（PWMB_CR1 寄存器），使能 PWMB。

注意：两个 PWM 的时钟并不同步，但仅影响 PWMA 的使能信号。

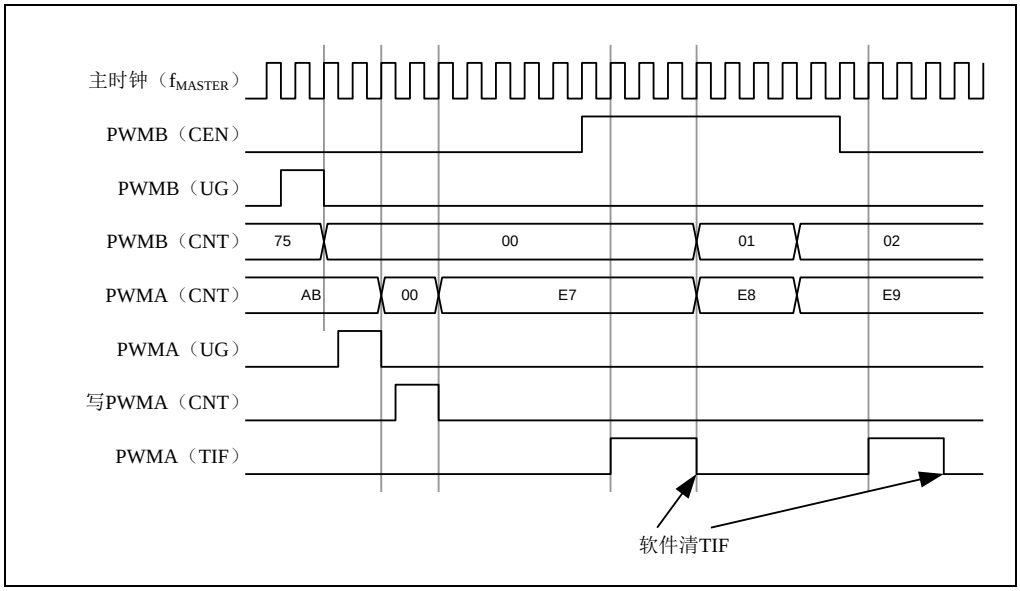
PWMB 的输出门控触发 PWMA



上图中，PWMA 的计数器和预分频器都没有在启动前初始化，所以都是从现有值开始计数的。如果在启动 PWMB 之前复位两个定时器，用户就可以写入期望的数值到 PWMA 的计数器，使之从指定值开始计数。对 PWMA 的复位操作可以通过软件写 PWMA_EGR 寄存器的 UG 位实现。

在下面这个例子中，我们使 PWMB 和 PWMA 同步。PWMB 为主模式并从 0 启动计数。PWMA 为触发从模式，并从 0xE7 启动计数。两个 PWM 采用相同的分频系数。当清除 PWMB_CR1 寄存器的 CEN 位时，PWMB 被禁止，同时 PWMA 停止计数。

1. 配置 PWMB 为主模式，将比较输出信号（OC5REF）作为触发信号输出。（配置 PWMB_CR2 寄存器的 MMS=100）。
2. 配置 PWMB 的 OC5REF 信号的波形（PWMB_CCMR1 寄存器）。
3. 配置 PWMA 把 PWMB 的输出作为自己的触发输入信号（配置 PWMA_SMCR 寄存器的 TS=010）。
4. 配置 PWMA 为门控触发模式（配置 PWMA_SMCR 寄存器的 SMS=101）。
5. 通过对 UG 位（PWMB_EGR 寄存器）写 1，复位 PWMB。
6. 通过对 UG 位（PWMA_EGR 寄存器）写 1，复位 PWMA。
7. 将 0xE7 写入 PWMA 的计数器中（PWMA_CNTRL），初始化 PWMA。
8. 通过对 CEN 位（PWMA_CR1 寄存器）写 1，使能 PWMA。
9. 通过对 CEN 位（PWMB_CR1 寄存器）写 1，启动 PWMB。
10. 通过对 CEN 位（PWMB_CR1 寄存器）写 0，停止 PWMB。



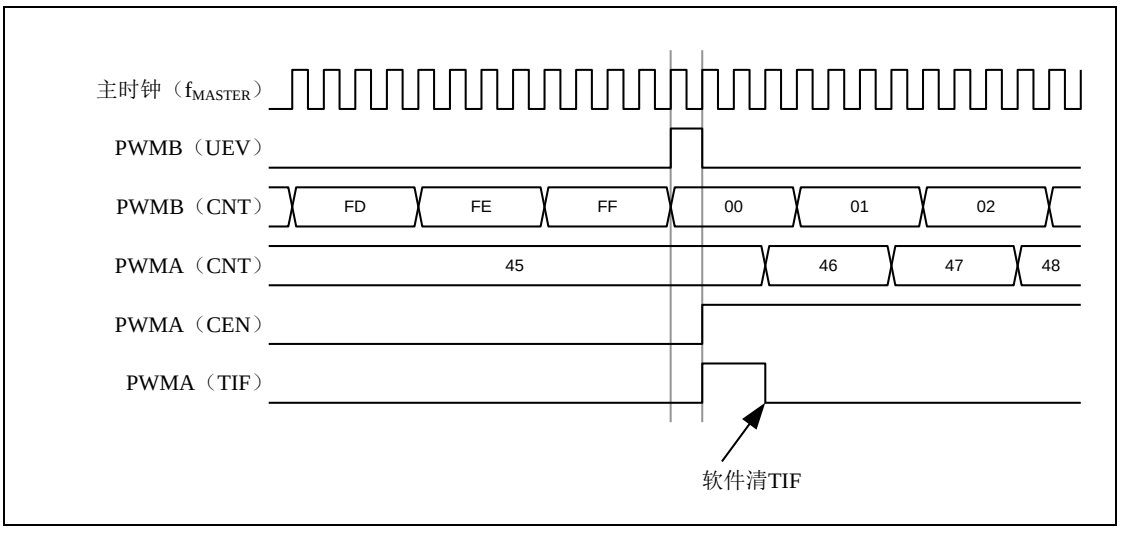
使用 PWMB 启动 PWMA

在本例中，我们用 PWMB 的更新事件来启动 PWMA。

PWMA 在 PWMB 发生更新事件时按照 PWMA 自己的驱动时钟从它的现有值开始计数（可以是非 0 值）。PWMA 在收到触发信号后自动使能 CEN 位，并开始计数，一直持续到用户向 PWMA_CR1 寄存器的 CEN 位写 0。两个 PWM 都使用 4 分频的 f_{MASTER} 作为驱动时钟（$f_{CK_CNT} = f_{MASTER}/4$）。

1. 配置 PWMB 为主模式，输出更新信号（UEV）。（配置 PWMB_CR2 寄存器的 MMS=010）。
2. 配置 PWMB 的周期（PWMB_ARR 寄存器）。
3. 配置 PWMA 用 PWMB 的输出作为输入的触发信号（配置 PWMA_SMCR 寄存器的 TS=010）。
4. 配置 PWMA 为触发模式（配置 PWMA_SMCR 寄存器的 SMS=110）。
5. 置位 CEN 位（PWMB_CR1 寄存器）启动 PWMB。

PWMB 的更新事件（PWMB-UEV）触发 PWMA



如同前面的例子，用户也可以在启动计数器前对它们初始化。

用外部信号同步的触发两个 PWM

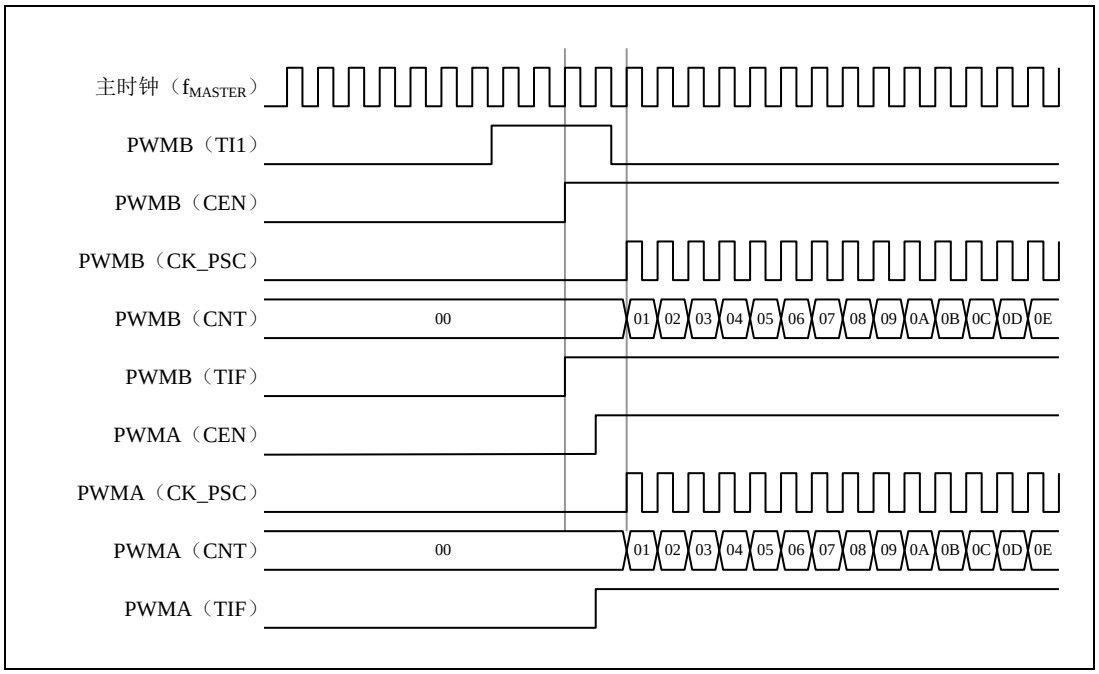
在本例中，使用 TI1 的上升沿使能 PWMB，并同时使能 PWMA。为了保持定时器的对齐，PWMB 需要配置成主/从模式（对于 TI1 信号为从模式，对于 PWMA 为主模式）。

1. 配置 PWMB 为主模式，以输出使能信号作为 PWMA 的触发（配置 PWMB_CR2 寄存器的 MMS=001）。
2. 配置 PWMB 为从模式，把 TI1 信号作为输入的触发信号（配置 PWMB_SMCR 寄存器的 TS=100）。
3. 配置 PWMB 的触发模式（配置 PWMB_SMCR 寄存器的 SMS=110）。
4. 配置 PWMB 为主/从模式（配置 PWMB_SMCR 寄存器的 MSM=1）。
5. 配置 PWMA 以 PWMB 的输出为输入触发信号（配置 PWMA_SMCR 寄存器的 TS=010）。
6. 配置 PWMA 的触发模式（配置 PWMA_SMCR 寄存器的 SMS=110）。

当 TI1 上出现上升沿时，两个定时器同步的开始计数，并且 TIF 位都被置起。

注意：在本例中，两个定时器在启动前都进行了初始化（设置 UG 位），所以它们都从 0 开始计数，但是用户也可以通过修改计数器寄存器（PWMA_CNT）来插入一个偏移量，这样的话，在 PWMB 的 CK_PSC 信号和 CNT_EN 信号间会插入延时。

PWMB 的 TI1 信号触发 PWMB 和 PWMA

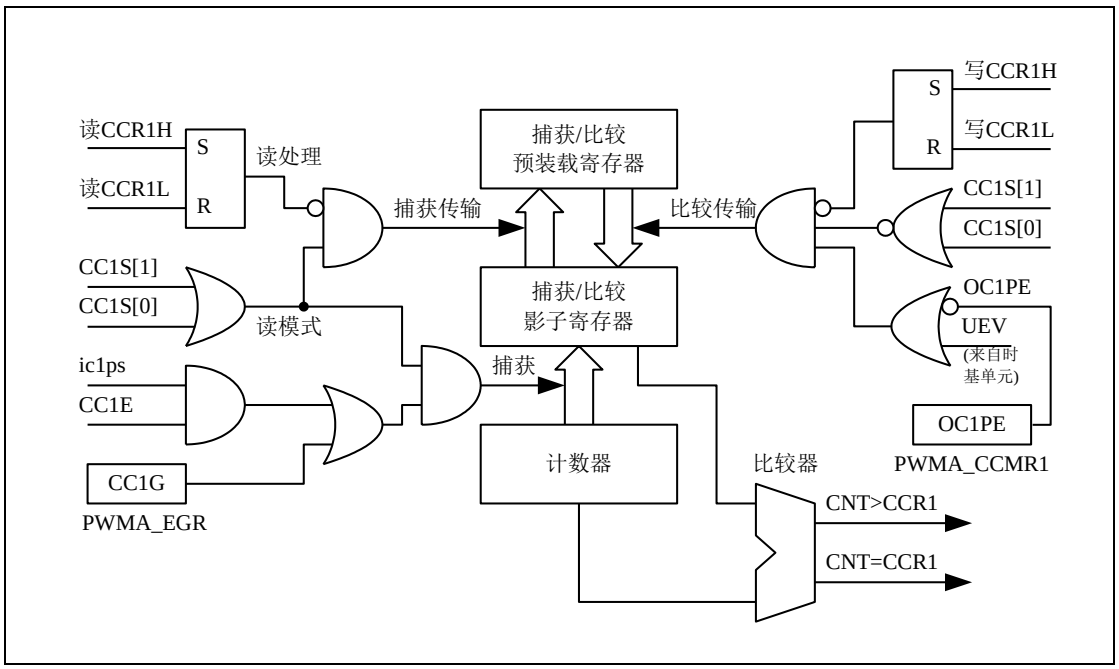


29.7　捕获/比较通道

PWM1P、PWM2P、PWM3P、PWM4P 可以用作输入捕获，PWM1P/PWM1N、PWM2P/PWM2N、PWM3P/PWM3N、PWM4P/PWM4N 可以输出比较，这个功能可以通过配置捕获/比较通道模式寄存器（PWMA_CCMRi）的 CCiS 通道选择位来实现，此处的 i 代表 1~4 的通道数。

每一个捕获/比较通道都是围绕着一个捕获/比较寄存器（包含影子寄存器）来构建的，包括捕获的输入部分（数字滤波、多路复用和预分频器）和输出部分（比较器和输出控制）。

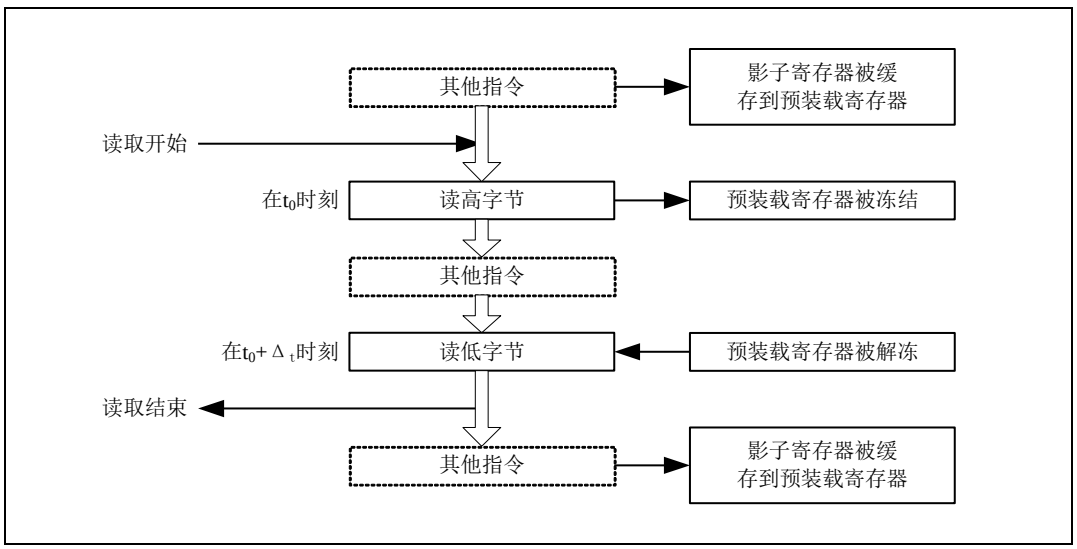
捕获/比较通道 1 的主要电路（其他通道与此类似）



捕获/比较模块由一个预装载寄存器和一个影子寄存器组成。读写过程仅操作预装载寄存器。在捕获模式下，捕获发生在影子寄存器上，然后再复制到预装载寄存器中。在比较模式下，预装载寄存器的内容被复制到影子寄存器中，然后影子寄存器的内容和计数器进行比较。

当通道被配置成输出模式时（PWMA_CCMRi 寄存器的 CCiS=0），可以随时访问 PWMA_CCRi 寄存器。

当通道被配置成输入模式时，对 PWMA_CCRi 寄存器的读操作类似于计数器的读操作。当捕获发生时，计数器的内容被捕获到 PWMA_CCRi 影子寄存器，随后再复制到预装载寄存器中。在读操作进行中，预装载寄存器是被冻结的。



上图描述了 16 位的 CCRi 寄存器的读操作流程，被缓存的数据将保持不变直到读流程结束。

在整个读流程结束后，如果仅仅读了 PWMA_CCRiL 寄存器，返回计数器数值的低位（LS）。

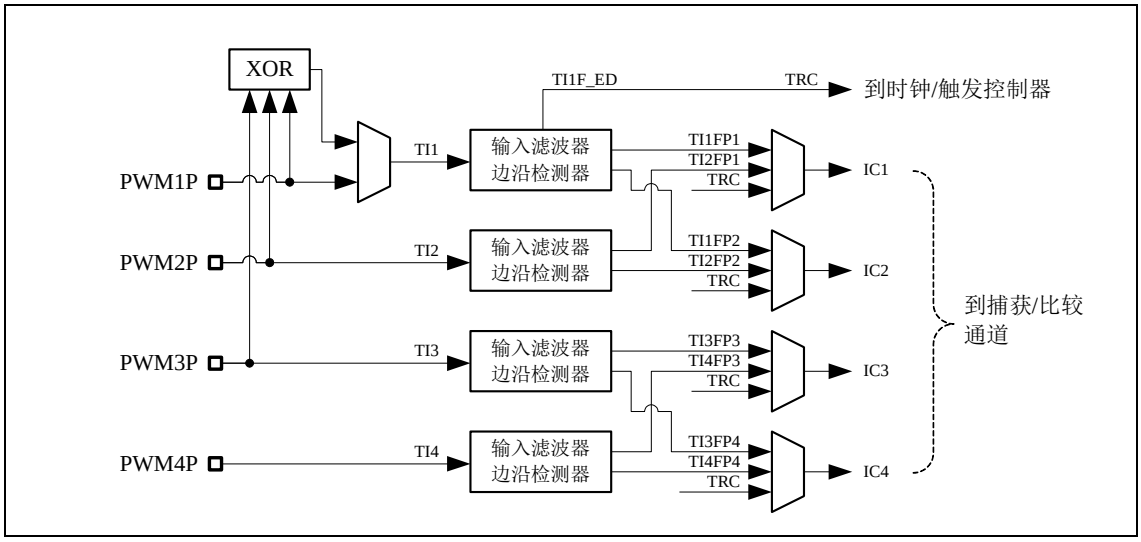
如果在读了低位（LS）数据以后再读高位（MS）数据，将不再返回同样的低位数据。

29.7.1 16 位 PWMA_CCRi 寄存器的写流程

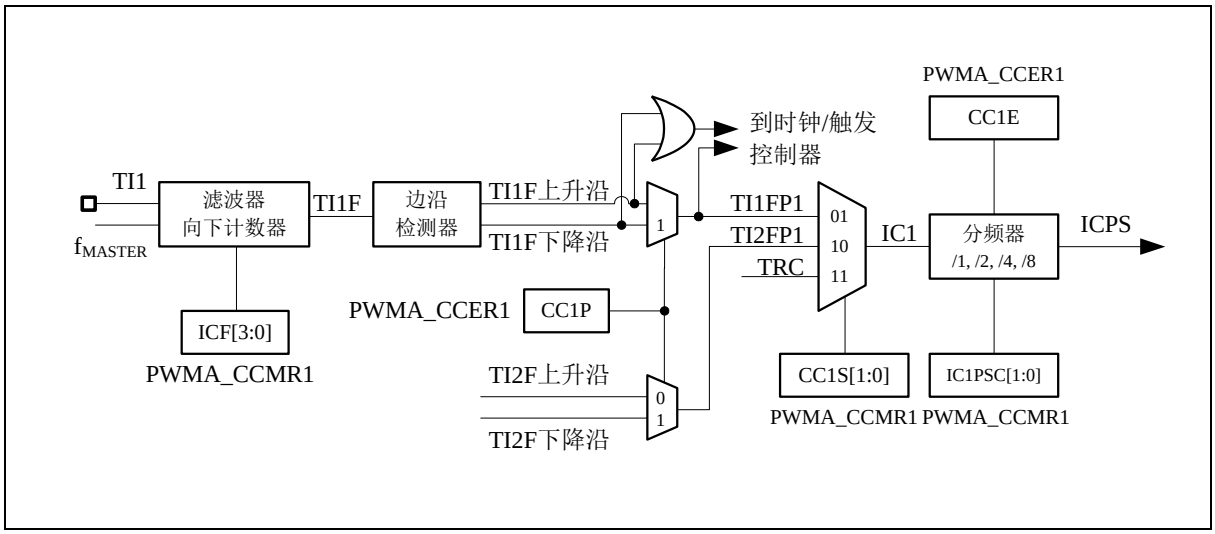
16 位 PWMA_CCRi 寄存器的写操作通过预装载寄存器完成。必需使用两条指令来完成整个流程，一条指令对应一个字节。必需先写高位字节（MS）。在写高位字节（MS）时，影子寄存器的更新被禁止直到低位字节（LS）的写操作完成。

29.7.2　输入模块

输入模块的框图



如图，输入部分对相应的 TIx 输入信号采样，并产生一个滤波后的信号 TIxF。然后，一个带极性选择的边缘监测器产生一个信号（TIxFPx），它可以作为触发模式控制器的输入触发或者作为捕获控制。该信号通过预分频后进入捕获寄存器（ICxPS）。



29.7.3 输入捕获模式

在输入捕获模式下，当检测到 ICi 信号上相应的边沿后，计数器的当前值被锁存到捕获/比较寄存器（PWMA_CCRx）中。当发生捕获事件时，相应的 CCiIF 标志（PWMA_SR 寄存器）被置 1。

如果 PWMA_IER 寄存器的 CCiIE 位被置位，也就是使能了中断，则将产生中断请求。如果发生捕获事件时 CCiIF 标志已经为高，那么重复捕获标志 CCiOF（PWMA_SR2 寄存器）被置 1。写 CCiIF=0 或读取存储在 PWMA_CCRiL 寄存器中的捕获数据都可清除 CCiIF。写 CCiOF=0 可清除 CCiOF。

PWM 输入信号上升沿时捕获

以下例子说明如何在 TI1 输入的上升沿时捕获计数器的值到 PWMA_CCR1 寄存器中，步骤如下：

1. 选择有效输入端：例如 PWMA_CCR1 连接到 TI1 输入，所以写入 PWMA_CCMR1 寄存器中的 CC1S=01，此时通道被配置为输入，并且 PWMA_CCR1 寄存器变为只读。
2. 根据输入信号 TIi 的特点，可通过配置 PWMA_CCMRi 寄存器中的 ICiF 位来设置相应的输入滤波器的滤波时间。假设输入信号在最多 5 个时钟周期的时间内抖动，我们须配置滤波器的带宽长于 5 个时钟周期；因此我们可以连续采样 8 次，以确认在 TI1 上一次真实的边沿变换，即在 TIMi_CCMR1 寄存器中写入 IC1F=0011，此时，只有连续采样到 8 个相同的 TI1 信号，信号才为有效（采样频率为 f_{MASTER}）。
3. 选择 TI1 通道的有效转换边沿，在 PWMA_CCER1 寄存器中写入 CC1P=0（上升沿）。
4. 配置输入预分频器。在本例中，我们希望捕获发生在每一个有效的电平转换时刻，因此预分频器被禁止（写 PWMA_CCMR1 寄存器的 IC1PS=00）。
5. 设置 PWMA_CCER1 寄存器的 CC1E=1，允许捕获计数器的值到捕获寄存器中。
6. 如果需要，通过设置 PWMA_IER 寄存器中的 CC1IE 位允许相关中断请求。

当发生一个输入捕获时：

- 当产生有效的电平转换时，计数器的值被传送到 PWMA_CCR1 寄存器。
- CC1IF 标志被设置。当发生至少 2 个连续的捕获时，而 CC1IF 未曾被清除时，CC1OF 也被置 1。
- 如设置了 CC1IE 位，则会产生一个中断。

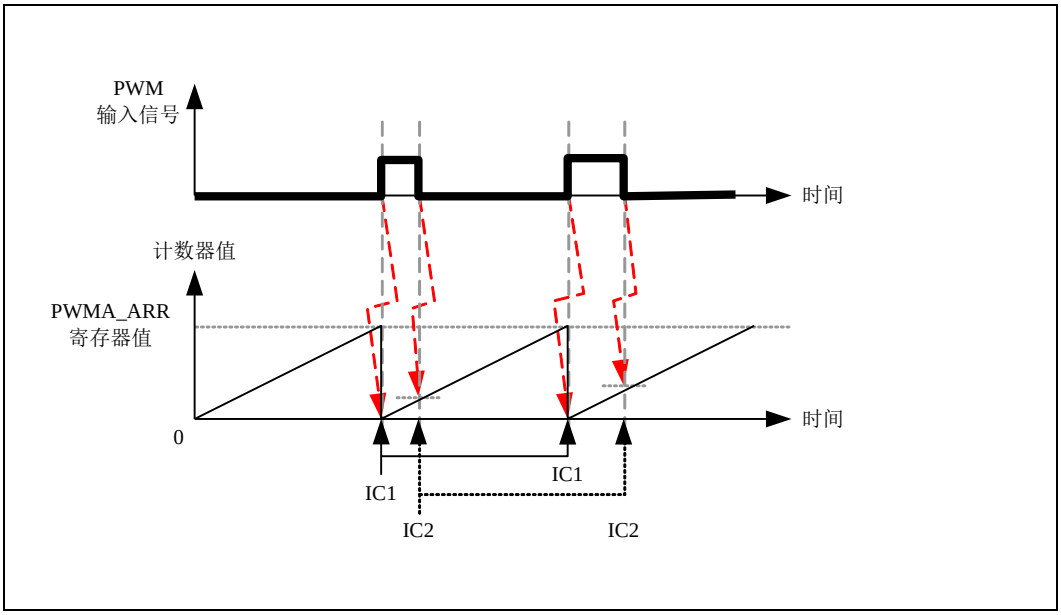
为了处理捕获溢出事件（CC1OF 位），建议在读出重复捕获标志之前读取数据，这是为了避免丢失在读出捕获溢出标志之后和读取数据之前可能产生的重复捕获信息。

注意：设置 PWMA_EGR 寄存器中相应的 CCiG 位，可以通过软件产生输入捕获中断。

PWM 输入信号测量

该模式是输入捕获模式的一个特例，除下列区别外，操作与输入捕获模式相同：

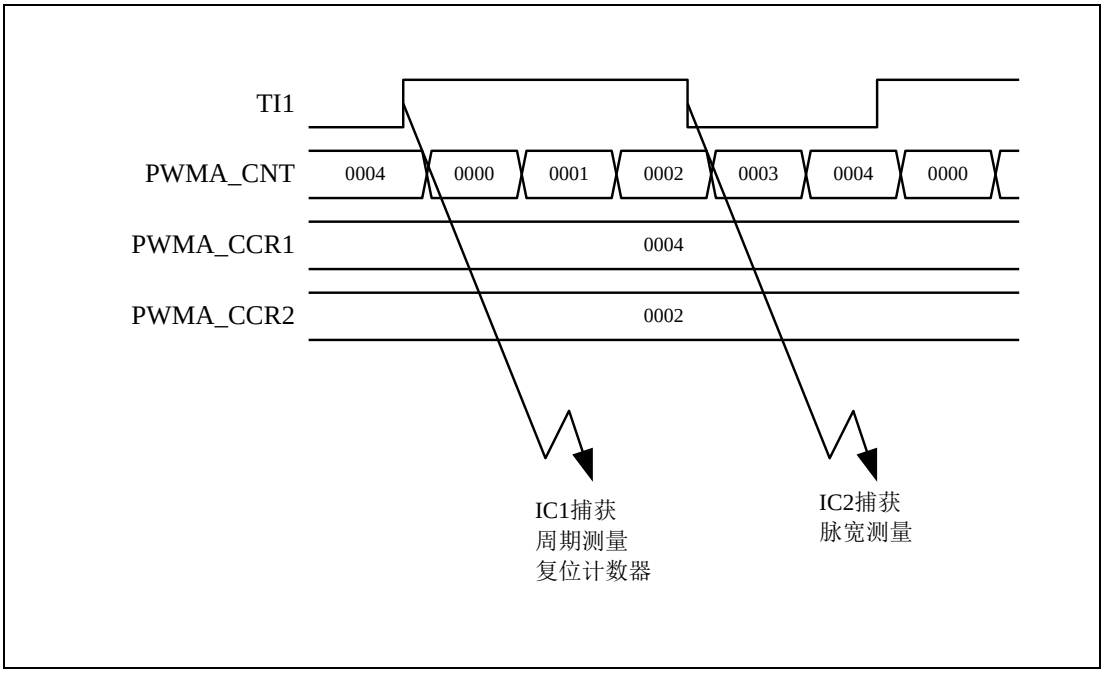
- 两个 ICi 信号被映射至同一个 TIi 输入。
- 这两个 ICi 信号的有效边沿的极性相反。
- 其中一个 TIiFP 信号被作为触发输入信号，而触发模式控制器被配置成复位触发模式。



例如，你可以用以下方式测量 TI1 上输入的 PWM 信号的周期（PWMA_CCR1 寄存器）和占空比（PWMA_CCR2 寄存器）。（具体取决于 f_{MASTER} 的频率和预分频器的值）

1. 选择 PWMA_CCR1 的有效输入：置 PWMA_CCMR1 寄存器的 CC1S=01（选中 TI1）。
2. 选择 TI1FP1 的有效极性（用来捕获数据到 PWMA_CCR1 中和清除计数器）：置 CC1P=0（上升沿有效）。
3. 选择 PWMA_CCR2 的有效输入：置 PWMA_CCMR2 寄存器的 CC2S=10（选中 TI1FP2）。
4. 选择 TI1FP2 的有效极性（捕获数据到 PWMA_CCR2）：置 CC2P=1（下降沿有效）。
5. 选择有效的触发输入信号：置 PWMA_SMCR 寄存器中的 TS=101（选择 TI1FP1）。
6. 配置触发模式控制器为复位触发模式：置 PWMA_SMCR 中的 SMS=100。
7. 使能捕获：置 PWMA_CCER1 寄存器中 CC1E=1，CC2E=1。

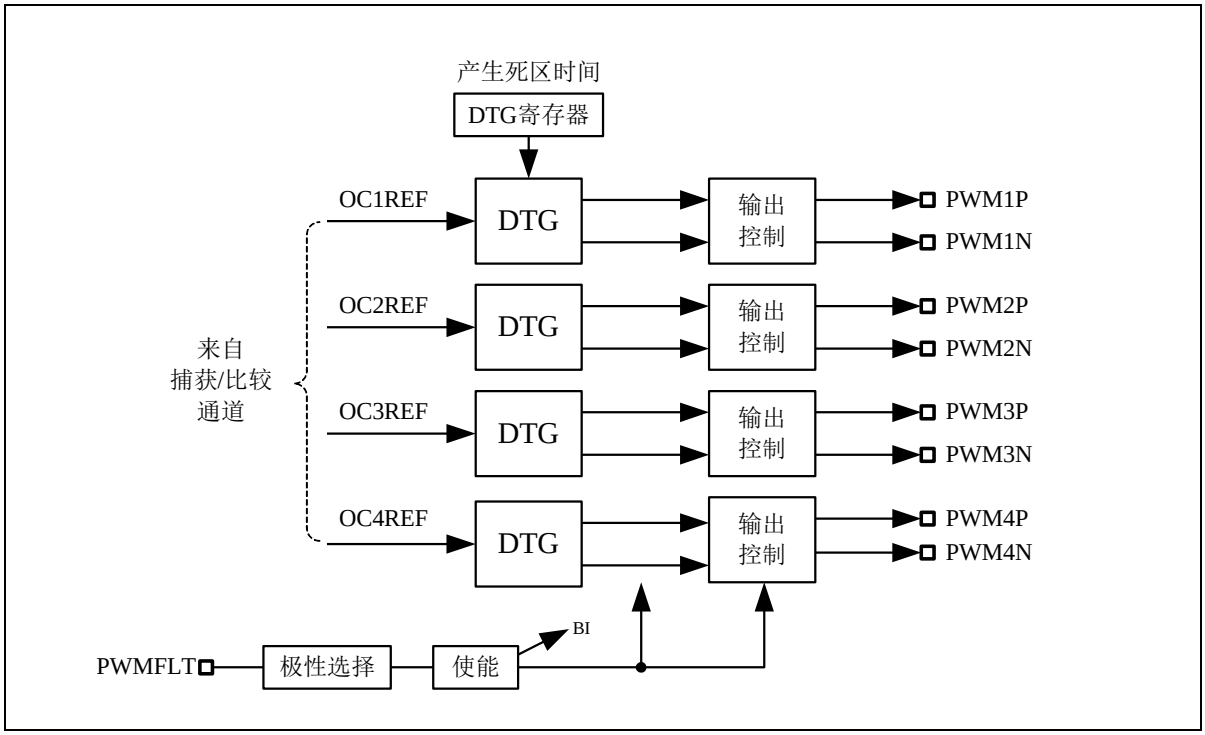
PWM 输入信号测量实例



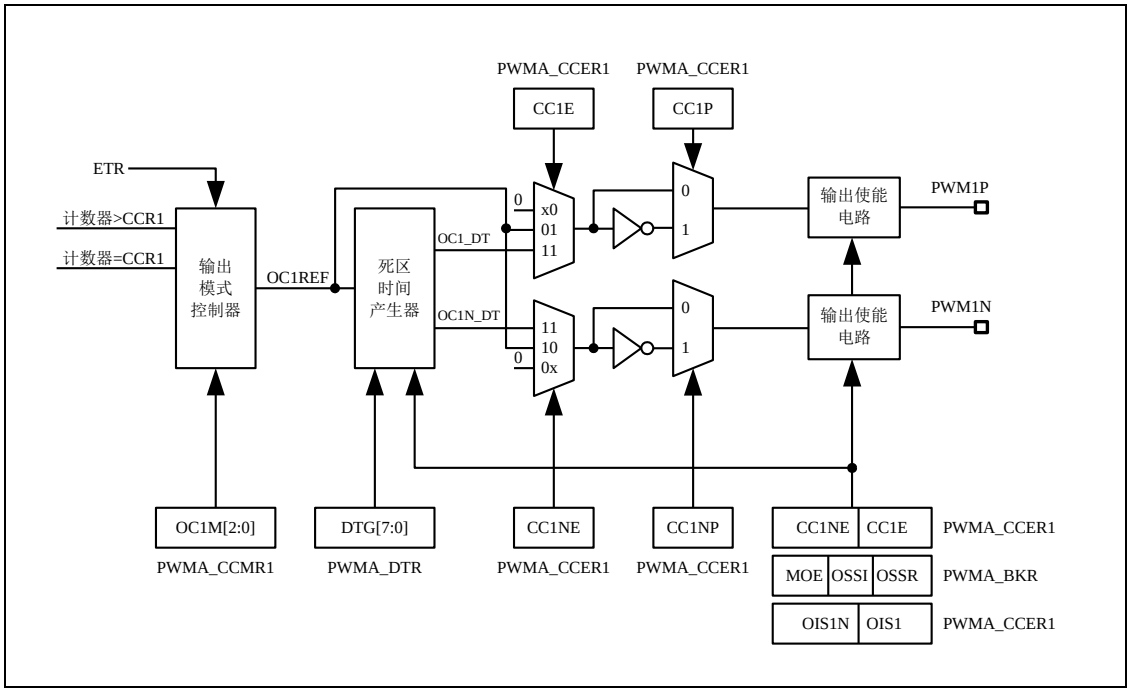
29.7.4　输出模块

输出模块会产生一个用来做参考的中间波形，称为 OCiREF（高有效）。刹车功能和极性的处理都在模块的最后处理。

输出模块框图



通道 1 详细的带互补输出的输出模块框图（其他通道类似）



29.7.5 强制输出模式

在输出模式下，输出比较信号能够直接由软件强制为高或低状态，而不依赖于输出比较寄存器和计数器间的比较结果。

置 PWMA_CCMRi 寄存器的 OCiM=101，可强制 OCiREF 信号为高。
置 PWMA_CCMRi 寄存器的 OCiM=100，可强制 OCiREF 信号为低。
OCi/OCiN 的输出是高还是低则取决于 CCiP/CCiNP 极性标志位。

该模式下，在 PWMA_CCRi 影子寄存器和计数器之间的比较仍然在进行，相应的标志也会被修改，也仍然会产生相应的中断。

29.7.6 输出比较模式

此模式用来控制一个输出波形或者指示一段给定的时间已经达到。

当计数器与捕获/比较寄存器的内容相匹配时，有如下操作：

- 根据不同的输出比较模式，相应的 OCi 输出信号：
  - 保持不变（OCiM=000）
  - 设置为有效电平（OCiM=001）
  - 设置为无效电平（OCiM=010）
  - 翻转（OCiM=011）
- 设置中断状态寄存器中的标志位（PWMA_SR1 寄存器中的 CCiIF 位）。
- 若设置了相应的中断使能位（PWMA_IER 寄存器中的 CCiIE 位），则产生一个中断。

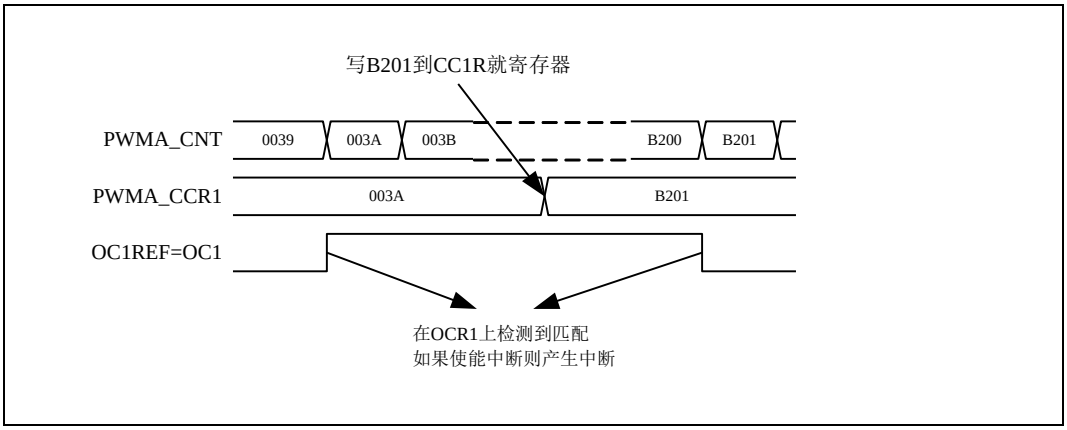
PWMA_CCMRi 寄存器的 OCiM 位用于选择输出比较模式，而 PWMA_CCMRi 寄存器的 CCiP 位用于选择有效和无效的电平极性。PWMA_CCMRi 寄存器的 OCiPE 位用于选择 PWMA_CCRi 寄存器是否需要使用预装载寄存器。在输出比较模式下，更新事件 UEV 对 OCiREF 和 OCi 输出没有影响。时间精度为计数器的一个计数周期。输出比较模式也能用来输出一个单脉冲。

输出比较模式的配置步骤：

1. 选择计数器时钟（内部、外部或者预分频器）。
2. 将相应的数据写入 PWMA_ARR 和 PWMA_CCRi 寄存器中。
3. 如果要产生一个中断请求，设置 CCiIE 位。
4. 选择输出模式步骤：
   1. 设置 OCiM=011，在计数器与 CCRi 匹配时翻转 OCiM 管脚的输出
   2. 设置 OCiPE = 0，禁用预装载寄存器
   3. 设置 CCiP = 0，选择高电平为有效电平
   4. 设置 CCiE = 1，使能输出
   5. 设置 PWMA_CR1 寄存器的 CEN 位来启动计数器

PWMA_CCRi 寄存器能够在任何时候通过软件进行更新以控制输出波形，条件是未使用预装载寄存器（OCiPE=0），否则 PWMA_CCRi 的影子寄存器只能在发生下一次更新事件时被更新。

输出比较模式，翻转 OC1



29.7.7　PWM 模式

脉冲宽度调制（PWM）模式可以产生一个由 PWMA_ARR 寄存器确定频率，由 PWMA_CCRi 寄存器确定占空比的信号。

在 PWMA_CCMRi 寄存器中的 OCiM 位写入 110（PWM 模式 1）或 111（PWM 模式 2），能够独立地设置每个 OCi 输出通道产生一路 PWM。必须设置 PWMA_CCMRi 寄存器的 OCiPE 位使能相应的预装载寄存器，也可以设置 PWMA_CR1 寄存器的 ARPE 位使能自动重装载的预装载寄存器（在向上计数模式或中央对称模式中）。

由于仅当发生一个更新事件的时候，预装载寄存器才能被传送到影子寄存器，因此在计数器开始计数之前，必须通过设置 PWMA_EGR 寄存器的 UG 位来初始化所有的寄存器。

OCi 的极性可以通过软件在 PWMA_CCERi 寄存器中的 CCiP 位设置，它可以设置为高电平有效或低电平有效。OCi 的输出使能通过 PWMA_CCERi 和 PWMA_BKR 寄存器中的 CCiE、MOE、OISi、OSSR 和 OSSI 位的组合来控制。

在 PWM 模式（模式 1 或模式 2）下，PWMA_CNT 和 PWMA_CCRi 始终在进行比较，（依据计数器的计数方向）以确定是否符合 PWMA_CCRi≤PWMA_CNT 或者 PWMA_CNT≤PWMA_CCRi。

根据 PWMA_CR1 寄存器中 CMS 位域的状态，定时器能够产生边沿对齐的 PWM 信号或中央对齐的 PWM 信号。

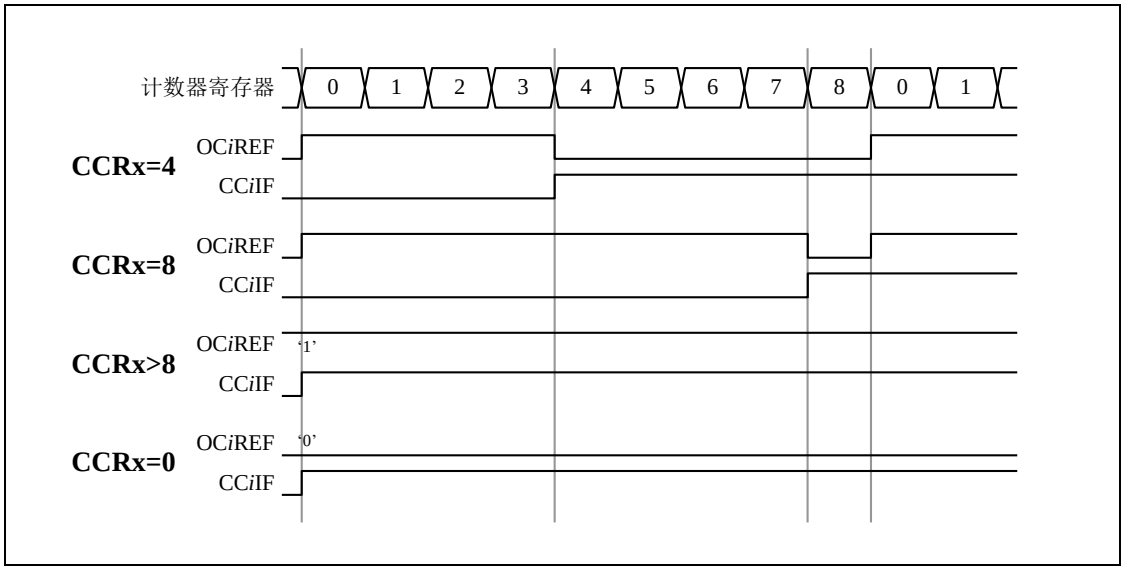
PWM 边沿对齐模式

向上计数配置

当 PWMA_CR1 寄存器中的 DIR 位为 0 时，执行向上计数。

下面是一个 PWM 模式 1 的例子。当 PWMA_CNT<PWMA_CCRi 时，PWM 参考信号 OCiREF 为高，否则为低。如果 PWMA_CCRi 中的比较值大于自动重装载值（PWMA_ARR），则 OCiREF 保持为’1’。如果比较值为 0，则 OCiREF 保持为’0’。

边沿对齐，PWM 模式 1 的波形（ARR=8）



向下计数的配置

当 PWMA_CR1 寄存器的 DIR 位为 1 时，执行向下计数。

在 PWM 模式 1 时，当 PWMA_CNT>PWMA_CCRi 时参考信号 OCiREF 为低，否则为高。如果 PWMA_CCRi 中的比较值大于 PWMA_ARR 中的自动重装载值，则 OCiREF 保持为'1'。该模式下不能产生 0%的 PWM 波形。

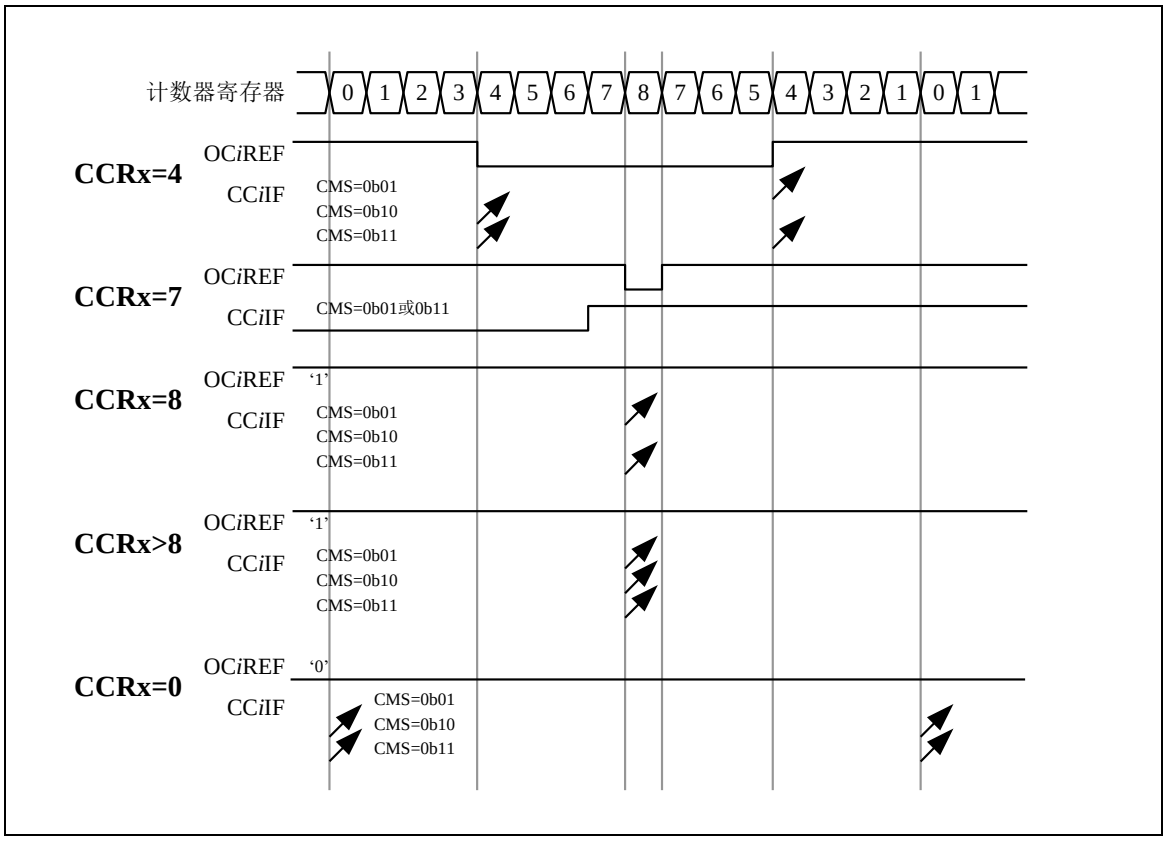
PWM 中央对齐模式

当 PWMA_CR1 寄存器中的 CMS 位不为'00'时为中央对齐模式（所有其他的配置对 OCiREF/OCi 信号都有相同的作用）。

根据不同的 CMS 位的设置，比较标志可以在计数器向上计数，向下计数，或向上和向下计数时被置 1。PWMA_CR1 寄存器中的计数方向位（DIR）由硬件更新，不要用软件修改它。

下面给出了一些中央对齐的 PWM 波形的例子：

- PWMA_ARR=8
- PWM 模式 1
- 标志位在以下三种情况下被置位：
  - 只有在计数器向下计数时（CMS=01）
  - 只有在计数器向上计数时（CMS=10）
  - 在计数器向上和向下计数时（CMS=11）
- 中央对齐的 PWM 波形（ARR=8）



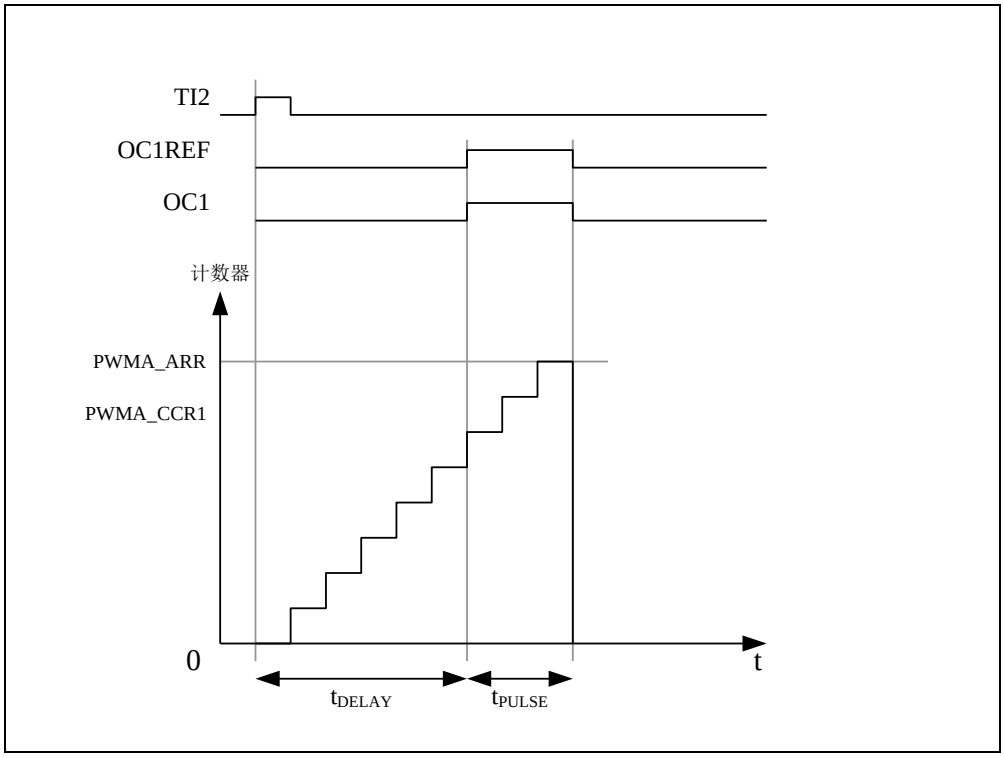
单脉冲模式

单脉冲模式（OPM）是前述众多模式的一个特例。这种模式允许计数器响应一个激励，并在一个程序可控的延时之后产生一个脉宽可控的脉冲。

可以通过时钟/触发控制器启动计数器，在输出比较模式或者 PWM 模式下产生波形。设置 PWMA_CR1 寄存器的 OPM 位将选择单脉冲模式，此时计数器自动地在下一个更新事件 UEV 时停止。仅当比较值与计数器的初始值不同时，才能产生一个脉冲。启动之前（当定时器正在等待触发），必须如下配置：

- 向上计数方式：计数器 CNT < CCRi ≤ ARR，
- 向下计数方式：计数器 CNT > CCRi。

单脉冲模式图例



例如，在从 TI2 输入脚上检测到一个上升沿之后延迟 tDELAY，在 OC1 上产生一个 tPULSE 宽度的正脉冲：（假定 IC2 作为触发 1 通道的触发源）

- 置 PWMA_CCMR2 寄存器的 CC2S=01，把 IC2 映射到 TI2。
- 置 PWMA_CCER1 寄存器的 CC2P=0，使 IC2 能够检测上升沿。
- 置 PWMA_SMCR 寄存器的 TS=110，使 IC2 作为时钟/触发控制器的触发源（TRGI）。
- 置 PWMA_SMCR 寄存器的 SMS=110（触发模式），IC2 被用来启动计数器。OPM 的波形由写入比较寄存器的数值决定（要考虑时钟频率和计数器预分频器）。
- t_{DELAY} 由 PWMA_CCR1 寄存器中的值定义。
- t_{PULSE} 由自动装载值和比较值之间的差值定义（PWMA_ARR – PWMA_CCR1）。
- 假定当发生比较匹配时要产生从 0 到 1 的波形，当计数器达到预装载值时要产生一个从 1 到 0 的波形，首先要置 PWMA_CCMR1 寄存器的 OCiM=111，进入 PWM 模式 2，根据需要有选择的设置 PWMA_CCMR1 寄存器的 OC1PE=1，置位 PWMA_CR1 寄存器中的 ARPE，使能预装载寄存器，然后在 PWMA_CCR1 寄存器中填写比较值，在 PWMA_ARR 寄存器中填写自动装载值，设置 UG 位来产生一个更新事件，然后等待在 TI2 上的一个外部触发事件。

在这个例子中，PWMA_CR1 寄存器中的 DIR 和 CMS 位应该置低。

因为只需要一个脉冲，所以设置 PWMA_CR1 寄存器中的 OPM=1，在下一个更新事件（当计数器从自动装载值翻转到 0）时停止计数。

OCx 快速使能（特殊情况）

在单脉冲模式下，对 TIi 输入脚的边沿检测会设置 CEN 位以启动计数器，然后计数器和比较值间的比较操作产生了单脉冲的输出。但是这些操作需要一定的时钟周期，因此它限制了可得到的最小延时 tDELAY。

如果要以最小延时输出波形，可以设置 PWMA_CCMRi 寄存器中的 OCiFE 位，此时强制 OCiREF（和 OCx）直接响应激励而不再依赖比较的结果，输出的波形与比较匹配时的波形一样。OCiFE 只在通道配

置为 PWMA 和 PWMB 模式时起作用。

互补输出和死区插入

PWMA 能够输出两路互补信号，并且能够管理输出的瞬时关断和接通，这段时间通常被称为死区，用户应该根据连接的输出器件和它们的特性（电平转换的延时、电源开关的延时等）来调整死区时间。

配置 PWMA_CCERi 寄存器中的 CCiP 和 CCiNP 位，可以为每一个输出独立地选择极性（主输出 OCi 或互补输出 OCiN）。

互补信号 OCi 和 OCiN 通过下列控制位的组合进行控制：PWMA_CCERi 寄存器的 CCiE 和 CCiNE 位，PWMA_BKR 寄存器中的 MOE、OISi、OISiN、OSSI 和 OSSR 位。特别的是，在转换到 IDLE 状态时（MOE 下降到 0）死区控制被激活。

同时设置 CCiE 和 CCiNE 位将插入死区，如果存在刹车电路，则还要设置 MOE 位。每一个通道都有一个 8 位的死区发生器。参考信号 OCiREF 可以产生 2 路输出 OCi 和 OCiN。

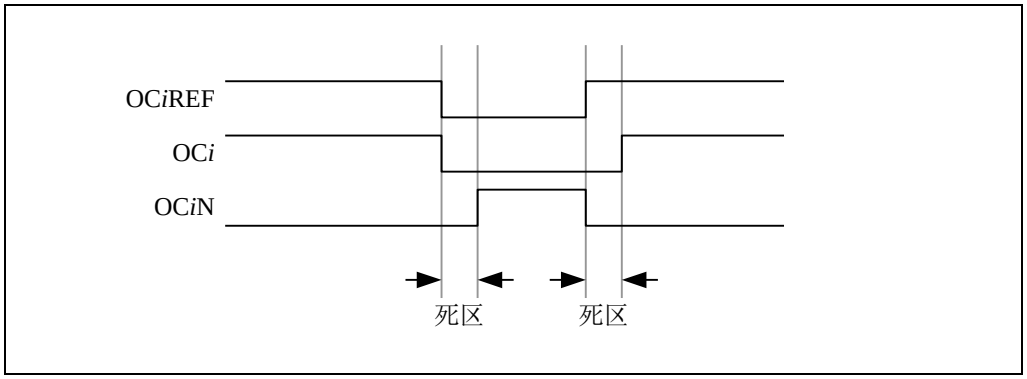
如果 OCi 和 OCiN 为高有效：

- OCi 输出信号与参考信号相同，只是它的上升沿相对于参考信号的上升沿有一个延迟。
- OCiN 输出信号与参考信号相反，只是它的上升沿相对于参考信号的下降沿有一个延迟。

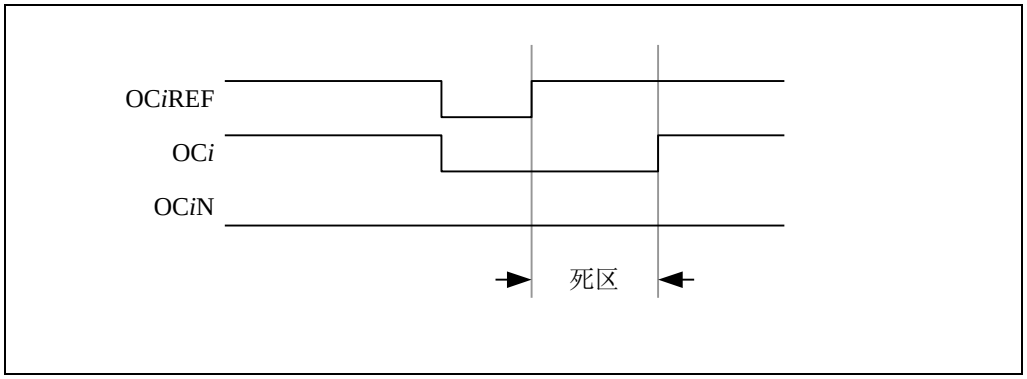
如果延迟大于当前有效的输出宽度（OCi 或者 OCiN），则不会产生相应的脉冲。

下列几张图显示了死区发生器的输出信号和当前参考信号 OCiREF 之间的关系。（假设 CCiP=0、CCiNP=0、MOE=1、CCiE=1 并且 CCiNE=1）

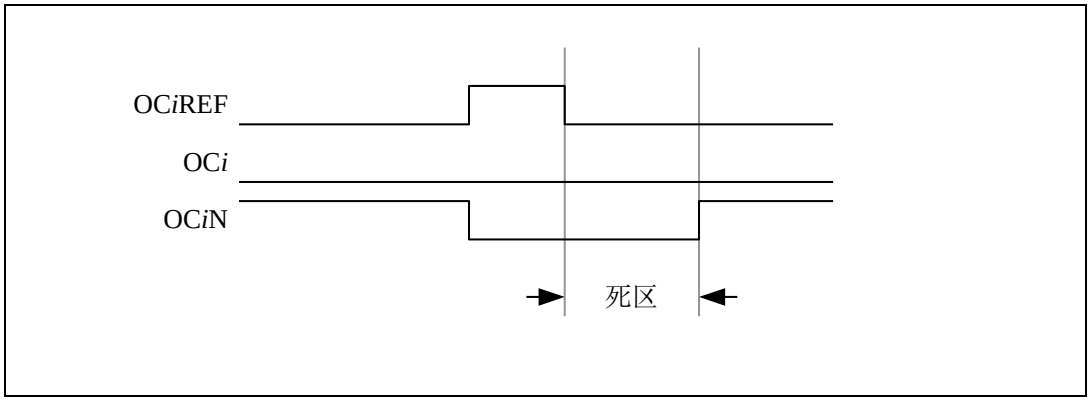
带死区插入的互补输出



死区波形延迟大于负脉冲



死区波形延迟大于正脉冲



每一个通道的死区延时都是相同的，是由 PWMA_DTR 寄存器中的 DTG 位编程配置。

重定向 OCiREF 到 OCi 或 OCiN

在输出模式下（强制输出、输出比较或 PWM 输出），通过配置 PWMA_CCERi 寄存器的 CCiE 和 CCiNE 位，OCiREF 可以被重定向到 OCi 或者 OCiN 的输出。

这个功能可以在互补输出处于无效电平时，在某个输出上送出一个特殊的波形（例如 PWM 或者静态有效电平）。另一个作用是，让两个输出同时处于无效电平，或同时处于有效电平（此时仍然是带死区的互补输出）。

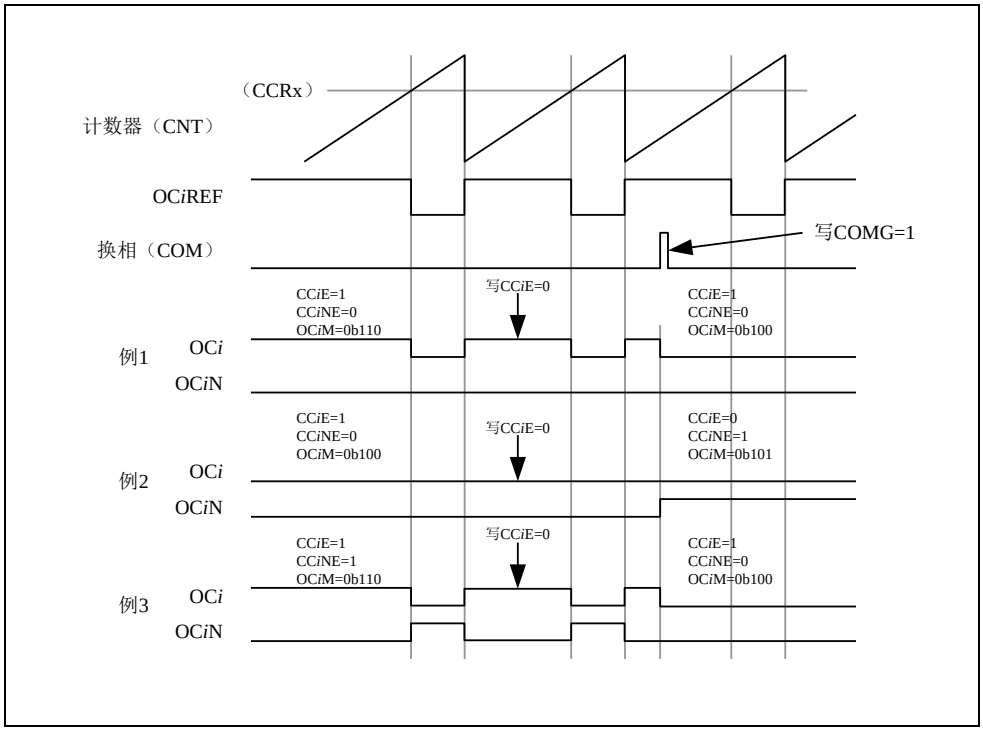
注：当只使能 OCiN（CCiE=0, CCiNE=1）时，它不会反相，而当 OCiREF 变高时立即有效。例如，如果 CCiNP=0，则 OCiN=OCiREF。另一方面，当 OCi 和 OCiN 都被使能时（CCiE=CCiNE=1），当 OCiREF 为高时 OCi 有效；而 OCiN 相反，当 OCiREF 低时 OCiN 变为有效。

针对马达控制的六步 PWM 输出

当在一个通道上需要互补输出时，预装载位有 OCiM、CCiE 和 CCiNE。在发生 COM 换相事件时，这些预装载位被传送到影子寄存器位。这样你就可以预先设置好下一步骤配置，并在同一个时刻同时修更改所有通道的配置。COM 可以通过设置 PWMA_EGR 寄存器的 COMG 位由软件产生，或在 TRGI 上升沿由硬件产生。

下图显示当发生 COM 事件时，三种不同配置下 OCx 和 OCxN 输出。

产生六步 PWM，使用 COM 的例子（OSSR=1）



29.7.8　使用刹车功能（PWMFLT）

刹车功能常用于马达控制中。当使用刹车功能时，依据相应的控制位（PWMA_BKR 寄存器中的 MOE、OSSI 和 OSSR 位），输出使能信号和无效电平都会被修改。

系统复位后，刹车电路被禁止，MOE 位为低。设置 PWMA_BKR 寄存器中的 BKE 位可以使能刹车功能。刹车输入信号的极性可以通过配置同一个寄存器中的 BKP 位选择。BKE 和 BKP 可以被同时修改。

MOE 下降沿相对于时钟模块可以是异步的，因此在实际信号（作用在输出端）和同步控制位（在 PWMA_BKR 寄存器中）之间设置了一个再同步电路。这个再同步电路会在异步信号和同步信号之间产生延迟。特别的，如果当它为低时写 MOE=1，则读出它之前必须先插入一个延时（空指令）才能读到正确的值。这是因为写入的是异步信号而读的是同步信号。

当发生刹车时（在刹车输入端出现选定的电平），有下述动作：

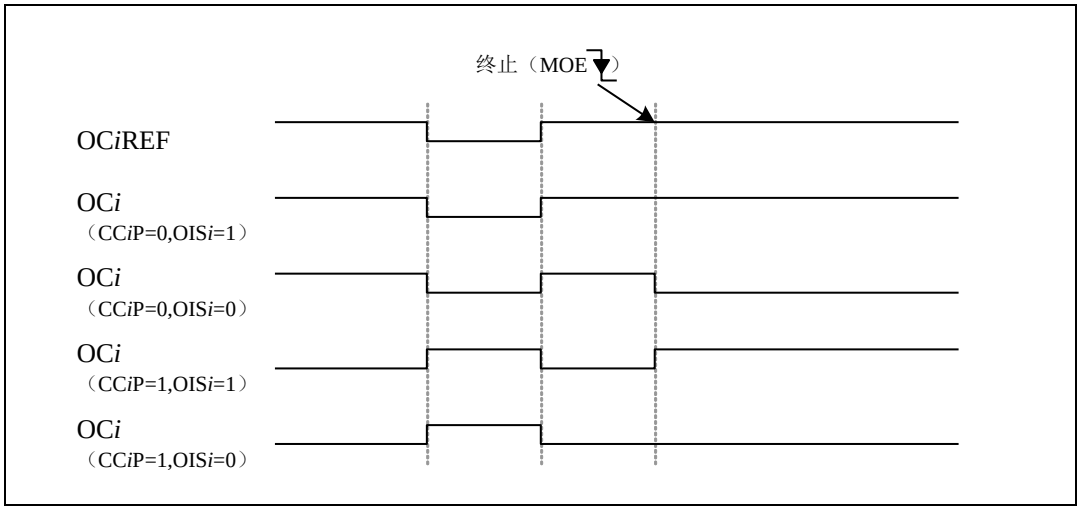
- MOE 位被异步地清除，将输出置于无效状态、空闲状态或者复位状态（由 OSSI 位选择）。这个特性在 MCU 的振荡器关闭时依然有效。
- 一旦 MOE=0，每一个输出通道输出由 PWMA_OISR 寄存器的 OISi 位设定的电平。如果 OSSI=0，则定时器不再控制输出使能信号，否则输出使能信号始终为高。
- 当使用互补输出时：
  - 输出首先被置于复位状态即无效的状态（取决于极性）。这是异步操作，即使定时器没有时钟时，此功能也有效。
  - 如果定时器的时钟依然存在，死区生成器将会重新生效，在死区之后根据 OISi 和 OISiN 位指示的电平驱动输出端口。即使在这种情况下，OCi 和 OCiN 也不能被同时驱动到有效的电平。注：因为重新同步 MOE，死区时间比通常情况下长一些（大约 2 个时钟周期）。
- 如果设置了 PWMA_IER 寄存器的 BIE 位，当刹车状态标志（PWMA_SR1 寄存器中的 BIF 位）为’1’时，则产生一个中断。
- 如果设置了 PWMA_BKR 寄存器中的 AOE 位，在下一个更新事件 UEV 时 MOE 位被自动置位。例如这可以用来进行波形控制，否则，MOE 始终保持低直到被再次置’1’。这个特性可以被用在

安全方面，你可以把刹车输入连到电源驱动的报警输出、热敏传感器或者其他安全器件上。

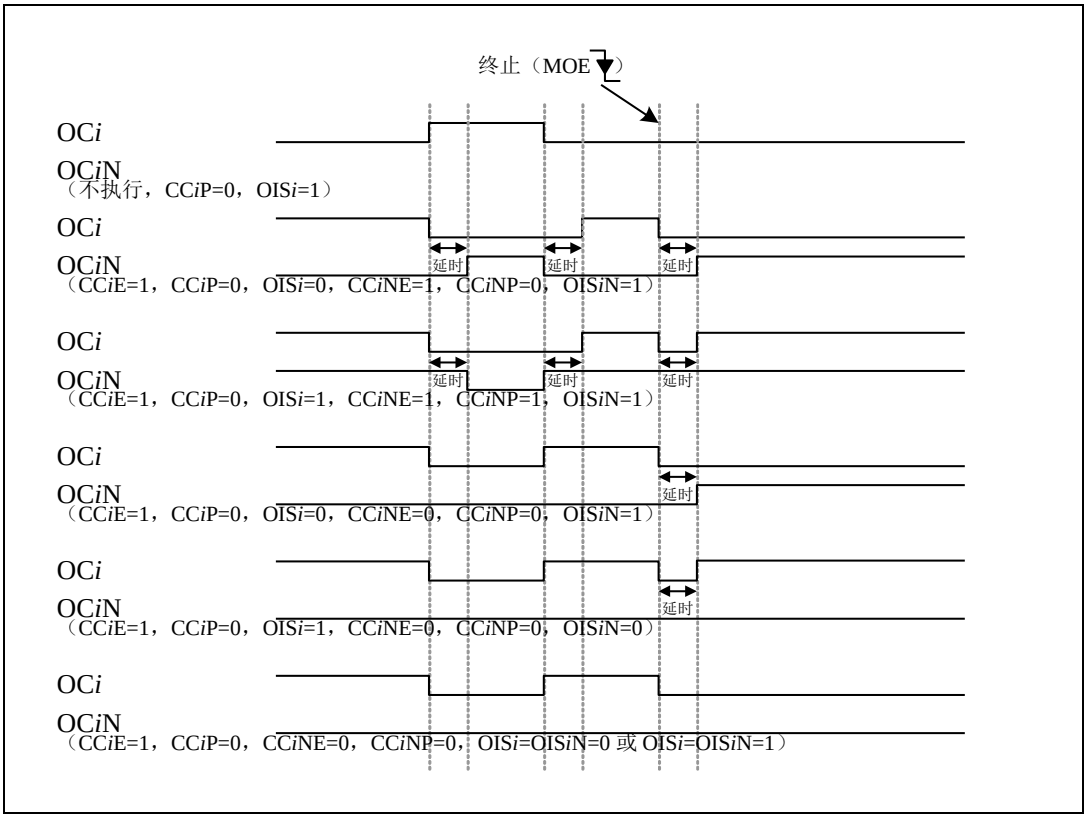
注：刹车输入为电平有效。所以，当刹车输入有效时，不能同时（自动地或者通过软件）设置 MOE。同时，状态标志 BIF 不能被清除。

刹车由 BRK 输入（BKIN）产生，它的有效极性是可编程的，且由 PWMA_BKR 寄存器的 BKE 位开启或禁止。除了刹车输入和输出管理，刹车电路中还实现了写保护以保证应用程序的安全。它允许用户冻结几个配置参数（OCi 极性和被禁止时的状态，OCiM 配置，刹车使能和极性）。用户可以通过 PWMA_BKR 寄存器的 LOCK 位，从三种级别的保护中选择一种。在 MCU 复位后 LOCK 位域只能被修改一次。

刹车响应的输出（不带互补输出的通道）



带互补输出的刹车响应的输出（PWMA 互补输出）



29.7.9 在外部事件发生时清除 OCiREF 信号

对于一个给定的通道，在 ETRF 输入端（设置 PWMA_CCMRi 寄存器中对应的 OCiCE 位为'1'）的高电平能够把 OCiREF 信号拉低，OCiREF 信号将保持为低直到发生下一次的更新事件 UEV。该功能只能用于输出比较模式和 PWM 模式，而不能用于强制模式。

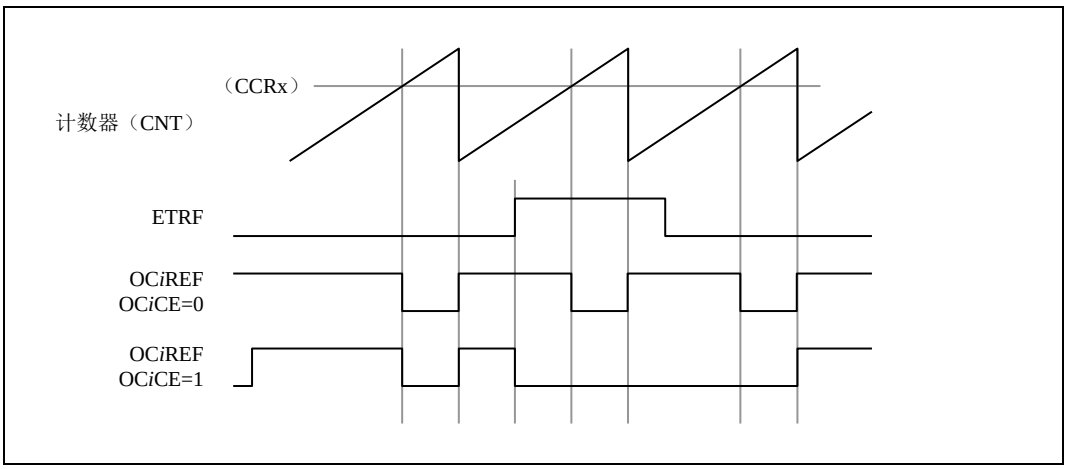
例如，OCiREF 信号可以联到一个比较器的输出，用于控制电流。这时，ETR 必须配置如下：

1. 外部触发预分频器必须处于关闭：PWMA_ETR 寄存器中的 ETPS[1:0]=00。
2. 必须禁止外部时钟模式 2：PWMA_ETR 寄存器中的 ECE=0。
3. 外部触发极性（ETP）和外部触发滤波器（ETF）可以根据需要配置。

下图显示了当 ETRF 输入变为高时，对应不同 OCiCE 的值，OCiREF 信号的动作。

在这个例子中，定时器 PWMA 被置于 PWM 模式。

ETR 清除 PWMA 的 OCiREF



29.7.10 编码器接口模式

编码器接口模式一般用于马达控制。

选择编码器接口模式的方法是：

- 如果计数器只在 TI2 的边沿计数，则置 PWMA_SMCR 寄存器中的 SMS=001；
- 如果只在 TI1 边沿计数，则置 SMS=010；
- 如果计数器同时在 TI1 和 TI2 边沿计数，则置 SMS=011。

通过设置 PWMA_CCER1 寄存器中的 CC1P 和 CC2P 位，可以选择 TI1 和 TI2 极性；如果需要，还可以对输入滤波器编程。

两个输入 TI1 和 TI2 被用来作为增量编码器的接口。假定计数器已经启动（PWMA_CR1 寄存器中的 CEN=1），则计数器在每次 TI1FP1 或 TI2FP2 上产生有效跳变时计数。TI1FP1 和 TI2FP2 是 TI1 和 TI2 在通过输入滤波器和极性控制后的信号。如果没有滤波和极性变换，则 TI1FP1=TI1，TI2FP2=TI2。根据两个输入信号的跳变顺序，产生了计数脉冲和方向信号。依据两个输入信号的跳变顺序，计数器向上或向下计数，同时硬件对 PWMA_CR1 寄存器的 DIR 位进行相应的设置。不管计数器是依靠 TI1 计数、依

靠 TI2 计数或者同时依靠 TI1 和 TI2 计数，在任一输入端（TI1 或者 TI2）的跳变都会重新计算 DIR 位。

编码器接口模式基本上相当于使用了一个带有方向选择的外部时钟。这意味着计数器只在 0 到 PWMA_ARR 寄存器的自动装载值之间连续计数（根据方向，或是 0 到 ARR 计数，或是 ARR 到 0 计数）。所以在开始计数之前必须配置 PWMA_ARR。在这种模式下捕获器、比较器、预分频器、重复计数器、触发输出特性等仍工作如常。编码器模式和外部时钟模式 2 不兼容，因此不能同时操作。

编码器接口模式下，计数器依照增量编码器的速度和方向被自动的修改，因此计数器的内容始终指示着编码器的位置，计数方向与相连的传感器旋转的方向对应。

下表列出了所有可能的组合（假设 TI1 和 TI2 不同时变换）。

计数方向与编码器信号的关系

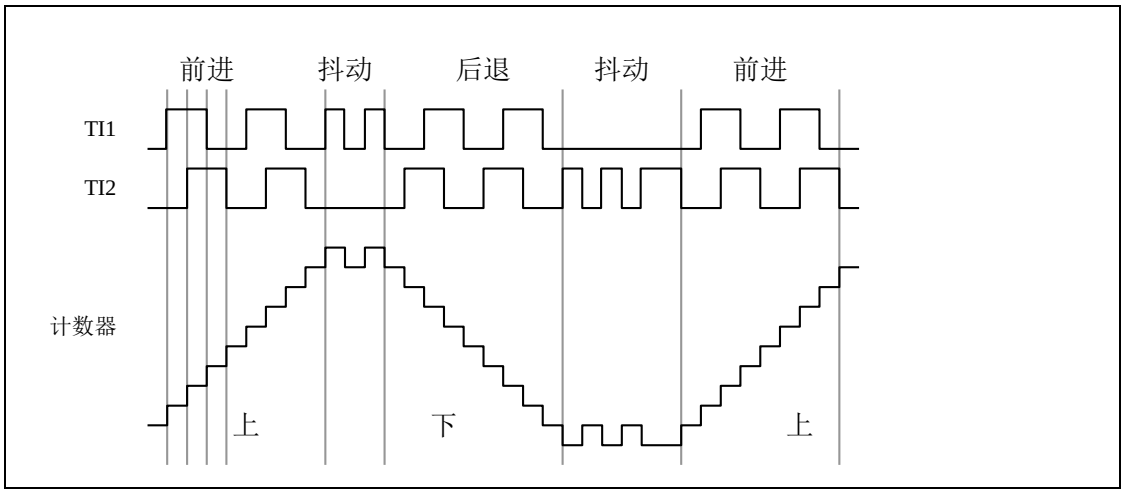
有效边沿	相对信号的电平 （TI1FP1 对应 TI2, TI2FP2 对应 TI1）	TI1FP1 信号		TI2FP2 信号	
		上升	下降	上升	下降
仅在 TI1 计数	高	向下计数	向上计数	不计数	不计数
	低	向上计数	向下计数	不计数	不计数
仅在 TI2 计数	高	不计数	不计数	向上计数	向下计数
	低	不计数	不计数	向下计数	向上计数
在 TI1 和 TI2 上计数	高	向下计数	向上计数	向上计数	向下计数
	低	向上计数	向下计数	向下计数	向上计数

一个外部的增量编码器可以直接与 MCU 连接而不需要外部接口逻辑。但是，一般使用比较器将编码器的差分输出转换成数字信号，这大大增加了抗噪声干扰能力。编码器输出的第三个信号表示机械零点，可以把它连接到一个外部中断输入并触发一个计数器复位。

下面是一个计数器操作的实例，显示了计数信号的产生和方向控制。它还显示了当选择了双边沿时，输入抖动是如何被抑制的；抖动可能会在传感器的位置靠近一个转换点时产生。在这个例子中，我们假定配置如下：

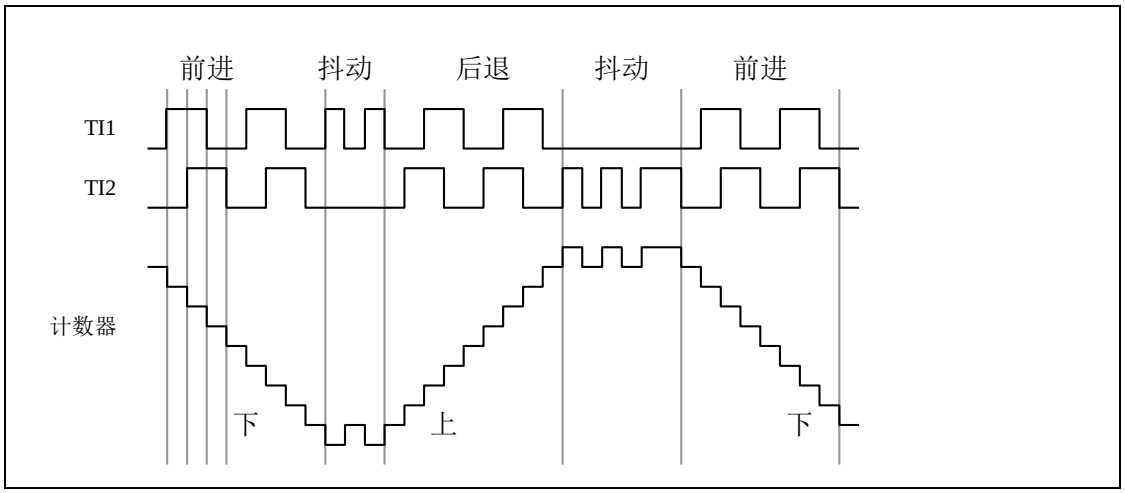
- CC1S=01（PWMA_CCMR1 寄存器，IC1FP1 映射到 TI1）
- CC2S=01（PWMA_CCMR2 寄存器，IC2FP2 映射到 TI2）
- CC1P=0（PWMA_CCER1 寄存器，IC1 不反相，IC1=TI1）
- CC2P=0（PWMA_CCER1 寄存器，IC2 不反相，IC2=TI2）
- SMS=011（PWMA_SMCR 寄存器，所有的输入均在上升沿和下降沿有效）.
- CEN=1（PWMA_CR1 寄存器，计数器使能）

编码器模式下的计数器操作实例



下图为当 IC1 极性反相时计数器的操作实例（CC1P=1，其他配置与上例相同）

IC1 反相的编码器接口模式实例



当定时器配置成编码器接口模式时，提供传感器当前位置的信息。使用另外一个配置在捕获模式下的定时器测量两个编码器事件的间隔，可以获得动态的信息（速度、加速度、减速度）。指示机械零点的编码器输出可被用做此目的。根据两个事件间的间隔，可以按照一定的时间间隔读出计数器。如果可能的话，你可以把计数器的值锁存到第三个输入捕获寄存器（捕获信号必须是周期的并且可以由另一个定时器产生）。

29.8 中断

PWMA/PWMB 各有 8 个中断请求源：

- 刹车中断
- 触发中断
- COM 事件中断
- 输入捕捉/输出比较 4 中断
- 输入捕捉/输出比较 3 中断

- 输入捕捉/输出比较 2 中断
- 输入捕捉/输出比较 1 中断
- 更新事件中断（如：计数器上溢，下溢及初始化）

为了使用中断特性，对每个被使用的中断通道，设置 PWMA_IER/PWMB_IER 寄存器中相应的中断使能位：即 BIE，TIE，COMIE，CCiIE，UIE 位。通过设置 PWMA_EGR/PWMB_EGR 寄存器中的相应位，也可以用软件产生上述各个中断源。

29.9 PWMA/PWMB 寄存器描述

29.9.1 功能脚切换（PWMx_PS）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_PS	7EFEB2H	C4PS[1:0]		C3PS[1:0]		C2PS[1:0]		C1PS[1:0]	
PWMA_PS2	7EFEB8H	-		-		AC6PS[1:0]		AC5PS[1:0]	
PWMB_PS	7EFEB6H	C8PS[1:0]		C7PS[1:0]		C6PS[1:0]		C5PS[1:0]	

C1PS[1:0]：高级 PWM 通道 1 输出脚选择位

C1PS[1:0]	PWM1P	PWM1N
00	P1.0	P1.1
01	P0.0	P0.1
10	P2.0	P2.1
11	-	-

C2PS[1:0]：高级 PWM 通道 2 输出脚选择位

C2PS[1:0]	PWM2P	PWM2N
00	P1.2	P1.3
01	P0.2	P0.3
10	P2.2	P2.3
11	-	-

C3PS[1:0]：高级 PWM 通道 3 输出脚选择位

C3PS[1:0]	PWM3P	PWM3N
00	P1.4	P1.5
01	P0.4	P0.5
10	P2.4	P2.5
11	-	-

C4PS[1:0]：高级 PWM 通道 4 输出脚选择位

C4PS[1:0]	PWM4P	PWM4N
00	P1.6	P1.7
01	P0.6	P0.7
10	P2.6	P2.7
11	-	-

AC5PS[1:0]：高级 PWMA 通道 5 输出脚选择位

AC5PS[1:0]	PWMAPS5
00	P3.3
01	P4.4
10	P5.6
11	-

AC6PS[1:0]：高级 PWMA 通道 6 输出脚选择位

AC6PS[1:0]	PWMAPS6
00	P3.4
01	P4.7
10	P5.7
11	-

C5PS[1:0]：高级 PWM 通道 5 输出脚选择位

C5PS[1:0]	PWM5
00	P0.1
01	P1.1
10	P2.1
11	P5.0

C6PS[1:0]：高级 PWM 通道 6 输出脚选择位

C6PS[1:0]	PWM6
00	P0.3
01	P1.3
10	P2.3
11	P5.1

C7PS[1:0]：高级 PWM 通道 7 输出脚选择位

C7PS[1:0]	PWM7
00	P0.5
01	P1.5
10	P2.5
11	P5.2

C8PS[1:0]：高级 PWM 通道 8 输出脚选择位

C8PS[1:0]	PWM8
00	P0.7
01	P1.7
10	P2.7
11	P5.3

29.9.2 高级 PWM 功能脚选择寄存器（PWMx_ETRPS）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ETRPS	7EFEB0H					BRKAPS[1:0]		ETRAPS[1:0]	
PWMB_ETRPS	7EFEB4H					BRKBPS[1:0]		ETRBPS[1:0]	

ETRAPS[1:0]：高级 PWMA 的外部触发脚 ERI 选择位

ETRAPS [1:0]	PWMETI
00	P3.2
01	P4.1
10	P2.3
11	P1.2

ETRBPS[1:0]：高级 PWMB 的外部触发脚 ERIB 选择位

ETRBPS [1:0]	PWMETI2
00	P3.2
01	P4.1
10	P2.3
11	P1.2

BRKAPS：高级 PWMA 的刹车脚 PWMFLT 选择位

BRKAPS	PWMFLT
00	P3.5
01	比较器的输出
10	P0.6
11	-

BRKBPS：高级 PWMB 的刹车脚 PWMFLT2 选择位

BRKBPS	PWMFLT2
00	P3.5
01	比较器的输出
10	P0.6
11	P0.7

29.9.3 输出使能寄存器（PWMx_ENO）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ENO	7EFEB1H	ENO4N	ENO4P	ENO3N	ENO3P	ENO2N	ENO2P	ENO1N	ENO1P
PWMB_ENO	7EFEB5H	-	ENO8P	-	ENO7P	-	ENO6P	-	ENO5P

ENO8P：PWM8 输出控制位

0：禁止 PWM8 输出

1：使能 PWM8 输出

ENO7P：PWM7 输出控制位

0：禁止 PWM7 输出

1：使能 PWM7 输出

ENO6P：PWM6 输出控制位

0：禁止 PWM6 输出

1：使能 PWM6 输出

ENO5P：PWM5 输出控制位

0：禁止 PWM5 输出

1：使能 PWM5 输出

ENO4N：PWM4N 输出控制位

0：禁止 PWM4N 输出

1：使能 PWM4N 输出

ENO4P：PWM4P 输出控制位

0：禁止 PWM4P 输出

1：使能 PWM4P 输出

ENO3N：PWM3N 输出控制位

0：禁止 PWM3N 输出

1：使能 PWM3N 输出

ENO3P：PWM3P 输出控制位

0：禁止 PWM3P 输出

1：使能 PWM3P 输出

ENO2N：PWM2N 输出控制位

0：禁止 PWM2N 输出

1：使能 PWM2N 输出

ENO2P：PWM2P 输出控制位

0：禁止 PWM2P 输出

1：使能 PWM2P 输出

ENO1N：PWM1N 输出控制位

0：禁止 PWM1N 输出

1：使能 PWM1N 输出

ENO1P：PWM1P 输出控制位

0：禁止 PWM1P 输出

1：使能 PWM1P 输出

29.9.4 输出附加使能寄存器（PWMx_IOAUX）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_IOAUX	7EFEB3H	AUX4N	AUX4P	AUX3N	AUX3P	AUX2N	AUX2P	AUX1N	AUX1P
PWMB_IOAUX	7EFEB7H	-	AUX8P	-	AUX7P	-	AUX6P	-	AUX5P

AUX8P：PWM8 输出附加控制位

0：PWM8 的输出直接由 ENO8P 控制

1：PWM8 的输出由 ENO8P 和 PWMB_BKR 共同控制

AUX7P：PWM7 输出附加控制位

0：PWM7 的输出直接由 ENO7P 控制

1：PWM7 的输出由 ENO7P 和 PWMB_BKR 共同控制

AUX6P：PWM6 输出附加控制位

0：PWM6 的输出直接由 ENO6P 控制

1：PWM6 的输出由 ENO6P 和 PWMB_BKR 共同控制

AUX5P：PWM5 输出附加控制位

0：PWM5 的输出直接由 ENO5P 控制

1：PWM5 的输出由 ENO5P 和 PWMB_BKR 共同控制

AUX4N：PWM4N 输出附加控制位

0：PWM4N 的输出直接由 ENO4N 控制

1：PWM4N 的输出由 ENO4N 和 PWMA_BKR 共同控制

AUX4P：PWM4P 输出附加控制位

0：PWM4P 的输出直接由 ENO4P 控制

1：PWM4P 的输出由 ENO4P 和 PWMA_BKR 共同控制

AUX3N：PWM3N 输出附加控制位

0：PWM3N 的输出直接由 ENO3N 控制

1：PWM3N 的输出由 ENO3N 和 PWMA_BKR 共同控制

AUX3P：PWM3P 输出附加控制位

0：PWM3P 的输出直接由 ENO3P 控制

1：PWM3P 的输出由 ENO3P 和 PWMA_BKR 共同控制

AUX2N：PWM2N 输出附加控制位

0：PWM2N 的输出直接由 ENO2N 控制

1：PWM2N 的输出由 ENO2N 和 PWMA_BKR 共同控制

AUX2P：PWM2P 输出附加控制位

0：PWM2P 的输出直接由 ENO2P 控制

1：PWM2P 的输出由 ENO2P 和 PWMA_BKR 共同控制

AUX1N：PWM1N 输出附加控制位

0：PWM1N 的输出直接由 ENO1N 控制

1：PWM1N 的输出由 ENO1N 和 PWMA_BKR 共同控制

AUX1P：PWM1P 输出附加控制位

0：PWM1P 的输出直接由 ENO1P 控制

1：PWM1P 的输出由 ENO1P 和 PWMA_BKR 共同控制

29.9.5 控制寄存器 1（PWMx_CR1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CR1	7EFEC0H	ARPEA	CMSA[1:0]		DIRA	OPMA	URSA	UDISA	CENA
PWMB_CR1	7EFEE0H	ARPEB	CMSB[1:0]		DIRB	OPMB	URSB	UDISB	CENB

ARPEn：自动预装载允许位（n=A,B）

0：PWMn_ARR 寄存器没有缓冲，它可以被直接写入

1：PWMn_ARR 寄存器由预装载缓冲器缓冲

CMSn[1:0]：选择对齐模式（n= A,B）

CMSn[1:0]	对齐模式	说明
00	边沿对齐模式	计数器依据方向位（DIR）向上或向下计数
01	中央对齐模式1	计数器交替地向上和向下计数。 配置为输出的通道的输出比较中断标志位(CCnIF)只在计数器向下计数时被置1。
10	中央对齐模式2	计数器交替地向上和向下计数。 配置为输出的通道的输出比较中断标志位(CCnIF)只在计数器向上计数时被置1。
11	中央对齐模式3	计数器交替地向上和向下计数。 配置为输出的通道的输出比较中断标志位(CCnIF)在计数器向上和向下计数时均被置1。

注 1：在计数器开启时（CEN=1），不允许从边沿对齐模式转换到中央对齐模式。

注 2：在中央对齐模式下，编码器模式（SMS=001，010，011）必须被禁止。

DIRn：计数器的计数方向（n= A,B）

0：计数器向上计数；

1：计数器向下计数。

注：当计数器配置为中央对齐模式或编码器模式时，该位为只读。

OPMn：单脉冲模式（n= A,B）

0：在发生更新事件时，计数器不停止；

1：在发生下一次更新事件时，清除 CEN 位，计数器停止。

URSn：更新请求源（n= A,B）

0：如果 UDIS 允许产生更新事件，则下述任一事件产生一个更新中断：

- 寄存器被更新（计数器上溢/下溢）
- 软件设置 UG 位
- 时钟/触发控制器产生的更新

1：如果 UDIS 允许产生更新事件，则只有当下列事件发生时才产生更新中断，并 UIF 置 1：

- 寄存器被更新（计数器上溢/下溢）

UDISn：禁止更新（n= A,B）

0：一旦下列事件发生，产生更新（UEV）事件：

- 计数器溢出/下溢
- 产生软件更新事件
- 时钟/触发模式控制器产生的硬件复位 被缓存的寄存器被装入它们的预装载值。

1：不产生更新事件，影子寄存器（ARR、PSC、CCRx）保持它们的值。如果设置了 UG 位或时钟/触发控制器发出了一个硬件复位，则计数器和预分频器被重新初始化。

CENn：允许计数器（n= A,B）

0：禁止计数器；

1：使能计数器。

注：在软件设置了 CEN 位后，外部时钟、门控模式和编码器模式才能工作。然而触发模式可以自动地通过硬件设置 CEN 位。

29.9.6 控制寄存器 2（PWMx_CR2），及实时触发 ADC

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CR2	7EFEC1H	TI1S	MMSA[2:0]			-	COMSA	-	CCPCA
PWMB_CR2	7EFEE1H	TI5S	MMSB[2:0]			-	COMSB	-	CCPCB

TI1S：第一组 PWM/PWMA 的 TI1 选择

0：PWM1P 输入管脚连到 TI1（数字滤波器的输入）；

1：PWM1P、PWM2P 和 PWM3P 管脚经异或后连到第一组 PWM 的 TI1。

TI5S：第二组 PWM/PWMB 的 TI5 选择

0：PWM5 输入管脚连到 TI5（数字滤波器的输入）；

1：PWM5、PWM6 和 PWM7 管脚经异或后连到第二组 PWM 的 TI5。

MMSA[2:0]：主模式选择

MMSA[2:0]	主模式	说明
000	复位	PWMA_EGR寄存器的UG位被用于作为触发输出（TRGO）。如果触发输入（时钟/触发控制器配置为复位模式）产生复位，则TRGO上的信号相对实际的复位会有一个延迟
001	使能	计数器使能信号被用于作为触发输出（TRGO）。其用于启动ADC，以便控制在一段时间内使能ADC。计数器使能信号是通过CEN控制位和门控模式下的触发输入信号的逻辑或产生。除非选择了主/从模式，当计数器使能信号受控于触发输入时，TRGO上会有一个延迟。 注：当需要使用PWM触发ADC转换时，需要先设置ADC_CONTR寄存器中的ADC_POWER、ADC_CHS以及ADC_EPWMT，当PWM产生TRGO内部信号时，系统会自动设置ADC_START来启动AD转换。详细使用请参考范例程序“使用PWM的CEN启动PWMA定时器，实时触发ADC”
010	更新	更新事件被选为触发输出（TRGO）
011	比较脉冲	一旦发生一次捕获或一次比较成功，当CC1IF标志被置1时，触发输出送出一个正脉冲（TRGO）
100	比较	OC1REF信号被用于作为触发输出（TRGO）
101	比较	OC2REF信号被用于作为触发输出（TRGO）
110	比较	OC3REF信号被用于作为触发输出（TRGO）
111	比较	OC4REF信号被用于作为触发输出（TRGO）

MMSB[2:0]：主模式选择

MMSB[2:0]	主模式	说明
000	复位	PWMB_EGR寄存器的UG位被用于作为触发输出（TRGO）。如果触发输入（时钟/触发控制器配置为复位模式）产生复位，则TRGO上的信号相对实际的复位会有一个延迟
001	使能	计数器使能信号被用于作为触发输出（TRGO）。其用于启动多个PWM，以便控制在一段时间内使能从PWM。计数器使能信号是通过CEN控制位和门控模式下的触发输入信号的逻辑或产生。除非选择了主/从模式，当计数器使能信号受控于触发输入时，TRGO上会有一个延迟。
010	更新	更新事件被选为触发输出（TRGO）
011	比较脉冲	一旦发生一次捕获或一次比较成功，当CC5IF标志被置1时，触发输出送出一个正脉冲（TRGO）
100	比较	OC5REF信号被用于作为触发输出（TRGO）
101	比较	OC6REF信号被用于作为触发输出（TRGO）
110	比较	OC7REF信号被用于作为触发输出（TRGO）
111	比较	OC8REF信号被用于作为触发输出（TRGO）

注：只有第一组 PWM（PWMA）的 TRGO 可用于触发启动 ADC

注：只有第二组 PWM（PWMB）的 TRGO 可用于第一组 PWM（PWMA）的 ITR2

COMSn：捕获/比较控制位的更新控制选择（n=A,B）

0：当 CCPCn=1 时，只有在 COMG 位置 1 的时候这些控制位才被更新

1：当 CCPCn=1 时，只有在 COMG 位置 1 或 TRGI 发生上升沿的时候这些控制位才被更新

CCPCn：捕获/比较预装载控制位（n= A,B）

0：CCIE，CCINE，CCiP，CCiNP 和 OCIM 位不是预装载的

1：CCIE，CCINE，CCiP，CCiNP 和 OCIM 位是预装载的；设置该位后，它们只在设置了 COMG 位后被更新。

注：该位只对具有互补输出的通道起作用。

29.9.7 从模式控制寄存器(PWMx_SMCR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_SMCR	7EFEC2H	MSMA	TSA[2:0]			-	SMSA[2:0]		
PWMB_SMCR	7EFEE2H	MSMB	TSB[2:0]			-	SMSB[2:0]		

MSMn：主/从模式（n= A,B）

0：无作用

1：触发输入（TRGI）上的事件被延迟了，以允许 PWMn 与它的从 PWM 间的完美同步（通过 TRGO）

TSA[2:0]：触发源选择

TSA[2:0]	触发源
000	-
001	-
010	内部触发ITR2
011	-
100	TI1的边沿检测器（TI1F_ED）
101	滤波后的定时器输入1（TI1FP1）
110	滤波后的定时器输入2（TI2FP2）
111	外部触发输入（ETRF）

TSB[2:0]：触发源选择

TSB[2:0]	触发源
000	-
001	-
010	-
011	-
100	TI5的边沿检测器（TI5F_ED）
101	滤波后的定时器输入1（TI5FP5）
110	滤波后的定时器输入2（TI6FP6）
111	外部触发输入（ETRF）

注：这些位只能在 SMS=000 时被改变，以避免在改变时产生错误的边沿检测。

SMSA[2:0]：时钟/触发/从模式选择

SMSA[2:0]	功能	说明
000	内部时钟模式	如果CEN=1，则预分频器直接由内部时钟驱动
001	编码器模式1	根据TI1FP1的电平，计数器在TI2FP2的边沿向上/下计数
010	编码器模式2	根据TI2FP2的电平，计数器在TI1FP1的边沿向上/下计数
011	编码器模式3	根据另一个输入的电平，计数器在TI1FP1和TI2FP2的边沿向上/下计数
100	复位模式	在选中的触发输入（TRGI）的上升沿时重新初始化计数器，并且产生一个更新 寄存器的信号
101	门控模式	当触发输入（TRGI）为高时，计数器的时钟开启。一旦触发输入变为低，则计 数器停止（但不复位）。计数器的启动和停止都是受控的
110	触发模式	计数器在触发输入TRGI的上升沿启动（但不复位），只有计数器的启动是受控 的
111	外部时钟模式1	选中的触发输入（TRGI）的上升沿驱动计数器。 注：如果TI1F_ED被选为触发输入（TS=100）时，不要使用门控模式。这是因为TI1F_ED在每次TI1F变化时只是输出一个脉冲，然而门控模式是要检查触发输入的电平

SMSB[2:0]：时钟/触发/从模式选择

SMSB[2:0]	功能	说明
000	内部时钟模式	如果CEN=1，则预分频器直接由内部时钟驱动
001	编码器模式1	根据TI5FP5的电平，计数器在TI6FP6的边沿向上/下计数
010	编码器模式2	根据TI6FP6的电平，计数器在TI5FP5的边沿向上/下计数
011	编码器模式3	根据另一个输入的电平，计数器在TI5FP5和TI6FP6的边沿向上/下计数
100	复位模式	在选中的触发输入（TRGI）的上升沿时重新初始化计数器，并且产生一个更新 寄存器的信号
101	门控模式	当触发输入（TRGI）为高时，计数器的时钟开启。一旦触发输入变为低，则计 数器停止（但不复位）。计数器的启动和停止都是受控的
110	触发模式	计数器在触发输入TRGI的上升沿启动（但不复位），只有计数器的启动是受控 的
111	外部时钟模式1	选中的触发输入（TRGI）的上升沿驱动计数器。 注：如果TI5F_ED被选为触发输入（TS=100）时，不要使用门控模式。这是因为TI5F_ED在每次TI5F变化时只是输出一个脉冲，然而门控模式是要检查触发输入的电平

29.9.8 外部触发寄存器(PWMx_ETR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ETR	7EFEC3H	ETP1	ECEA	ETPSA[1:0]		ETFA[3:0]			
PWMB_ETR	7EFEE3H	ETP2	ECEB	ETPSB[1:0]		ETFB[3:0]			

ETPn：外部触发 ETR 的极性（n= A,B）

0：高电平或上升沿有效

1：低电平或下降沿有效

ECEn：外部时钟使能（n= A,B）

0：禁止外部时钟模式 2

1：使能外部时钟模式 2，计数器的时钟为 ETRF 的有效沿。

注 1：ECE 置 1 的效果与选择把 TRGI 连接到 ETRF 的外部时钟模式 1 相同（PWMn_SMCR 寄存器中，SMS=111，TS=111）。

注 2：外部时钟模式 2 可与下列模式同时使用：触发标准模式；触发复位模式；触发门控模式。但是，此时 TRGI 决不能与 ETRF 相连（PWMn_SMCR 寄存器中，TS 不能为 111）。

注 3：外部时钟模式 1 与外部时钟模式 2 同时使能，外部时钟输入为 ETRF。

ETPSn：外部触发预分频器外部触发信号 EPRP 的频率最大不能超过 fMASTER/4。可用预分频器来降低 ETRP 的频率，当 EPRP 的频率很高时，它非常有用：（n= A,B）

00：预分频器关闭

01：EPRP 的频率/2

02：EPRP 的频率/4

03：EPRP 的频率/8

ETFn[3:0]：外部触发滤波器选择，该位域定义了 ETRP 的采样频率及数字滤波器长度。（n= A,B）

ETFn[3:0]	时钟数	ETF[3:0]	时钟数
0000	1	1000	48
0001	2	1001	64
0010	4	1010	80
0011	8	1011	96
0100	12	1100	128
0101	16	1101	160
0110	24	1110	192
0111	32	1111	256

29.9.9 中断使能寄存器(PWMx_IER)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_IER	7EFEC4H	BIEA	TIEA	COMIEA	CC4IE	CC3IE	CC2IE	CC1IE	UIEA
PWMB_IER	7EFEE4H	BIEB	TIEB	COMIEB	CC8IE	CC7IE	CC6IE	CC5IE	UIEB

BIEn：允许刹车中断（n= A,B）

0：禁止刹车中断；

1：允许刹车中断。

TIE：触发中断使能（n= A,B）

0：禁止触发中断；

1：使能触发中断。

COMIE：允许 COM 中断（n= A,B）

0：禁止 COM 中断；

1：允许 COM 中断。

CCnIE：允许捕获/比较 n 中断（n=1,2,3,4,5,6,7,8）

0：禁止捕获/比较 n 中断；

1：允许捕获/比较 n 中断。

UIEn：允许更新中断（n= A,B）

0：禁止更新中断；

1：允许更新中断。

29.9.10 状态寄存器 1(PWMx_SR1)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_SR1	7EFEC5H	BIFA	TIFA	COMIFA	CC4IF	CC3IF	CC2IF	CC1IF	UIFA
PWMB_SR1	7EFEE5H	BIFB	TIFB	COMIFB	CC8IF	CC7IF	CC6IF	CC5IF	UIFB

BIFn：刹车中断标记。一旦刹车输入有效，由硬件对该位置 1。如果刹车输入无效，则该位可由软件清 0。（n= A,B）

0：无刹车事件产生

1：刹车输入上检测到有效电平

TIFn：触发器中断标记。当发生触发事件时由硬件对该位置 1。由软件清 0。（n= A,B）

0：无触发器事件产生

1：触发中断等待响应

COMIFn：COM 中断标记。一旦产生 COM 事件该位由硬件置 1。由软件清 0。（n= A,B）

0：无 COM 事件产生

1：COM 中断等待响应

CC8IF：捕获/比较8中断标记，参考CC1IF描述

CC7IF：捕获/比较7中断标记，参考CC1IF描述

CC6IF：捕获/比较6中断标记，参考CC1IF描述

CC5IF：捕获/比较5中断标记，参考CC1IF描述

CC4IF：捕获/比较4中断标记，参考CC1IF描述

CC3IF：捕获/比较3中断标记，参考CC1IF描述

CC2IF：捕获/比较2中断标记，参考CC1IF描述

CC1IF：捕获/比较1中断标记。

如果通道CC1配置为输出模式：

当计数器值与比较值匹配时该位由硬件置1，但在中心对称模式下除外。它由软件清0。

0：无匹配发生；

1：PWMA_CNT 的值与 PWMA_CCR1 的值匹配。

注：在中心对称模式下，当计数器值为 0 时，向上计数，当计数器值为 ARR 时，向下计数（它从 0 向上计数到 ARR-1，再由 ARR 向下计数到 1）。因此，对所有的 SMS 位值，这两个值都不置标记。但是，如果 CCR1>ARR，则当 CNT 达到 ARR 值时，CC1IF 置 1。

如果通道CC1配置为输入模式：

当捕获事件发生时该位由硬件置1，它由软件清0或通过读PWMA_CCR1L清0。

0：无输入捕获产生

1：计数器值已被捕获至 PWMA_CCR1

UIFn：更新中断标记 当产生更新事件时该位由硬件置 1。它由软件清 0。（n= A,B）

0：无更新事件产生

1：更新事件等待响应。当寄存器被更新时该位由硬件置 1

- 若 PWMn_CR1 寄存器的 UDIS=0，当计数器上溢或下溢时
- 若 PWMn_CR1 寄存器的 UDIS=0、URS=0，当设置 PWMn_EGR 寄存器的 UG 位软件对计数器 CNT 重新初始化时
- 若 PWMn_CR1 寄存器的 UDIS=0、URS=0，当计数器 CNT 被触发事件重新初始化时

29.9.11 状态寄存器 2(PWMx_SR2)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_SR2	7EFEC6H	-	-	-	CC4OF	CC3OF	CC2OF	CC1OF	-
PWMB_SR2	7EFEE6H	-	-	-	CC8OF	CC7OF	CC6OF	CC5OF	-

CC8OF：捕获/比较8重复捕获标记。参见CC1OF描述。

CC7OF：捕获/比较7重复捕获标记。参见CC1OF描述。

CC6OF：捕获/比较6重复捕获标记。参见CC1OF描述。

CC5OF：捕获/比较5重复捕获标记。参见CC1OF描述。

CC4OF：捕获/比较4重复捕获标记。参见CC1OF描述。

CC3OF：捕获/比较3重复捕获标记。参见CC1OF描述。

CC2OF：捕获/比较2重复捕获标记。参见CC1OF描述。

CC1OF：捕获/比较1重复捕获标记。仅当相应的通道被配置为输入捕获时，该标记可由硬件置1。写0可清除该位。

0：无重复捕获产生；

1：计数器的值被捕获到 PWMA_CCR1 寄存器时，CC1IF 的状态已经为 1。

29.9.12 事件产生寄存器（PWMx_EGR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_EGR	7EFEC7H	BGA	TGA	COMGA	CC4G	CC3G	CC2G	CC1G	UGA
PWMB_EGR	7EFEE7H	BGB	TGB	COMGB	CC8G	CC7G	CC6G	CC5G	UGB

BGn：产生刹车事件。该位由软件置 1，用于产生一个刹车事件，由硬件自动清 0（n= A,B）

0：无动作

1：产生一个刹车事件。此时 MOE=0、BIF=1，若开启对应的中断（BIE=1），则产生相应的中断

TGn：产生触发事件。该位由软件置 1，用于产生一个触发事件，由硬件自动清 0（n= A,B）

0：无动作

1：TIF=1，若开启对应的中断（TIE=1），则产生相应的中断

COMGn：捕获/比较事件，产生控制更新。该位由软件置 1，由硬件自动清 0（n= A,B）

0：无动作

1：CCPC=1，允许更新 CCIE、CCINE、CCiP，CCiNP，OCIM 位。

注：该位只对拥有互补输出的通道有效

CC8G：产生捕获/比较 8 事件。参考 CC1G 描述

CC7G：产生捕获/比较 7 事件。参考 CC1G 描述

CC6G：产生捕获/比较 6 事件。参考 CC1G 描述

CC5G：产生捕获/比较 5 事件。参考 CC1G 描述

CC4G：产生捕获/比较 4 事件。参考 CC1G 描述

CC3G：产生捕获/比较 3 事件。参考 CC1G 描述

CC2G：产生捕获/比较 2 事件。参考 CC1G 描述

CC1G：产生捕获/比较 1 事件。产生捕获/比较 1 事件。该位由软件置 1，用于产生一个捕获/比较事件，由硬件自动清 0。

0：无动作；

1：在通道 CC1 上产生一个捕获/比较事件。

若通道 CC1 配置为输出：设置 CC1IF=1，若开启对应的中断，则产生相应的中断。

若通道 CC1 配置为输入：当前的计数器值被捕获至 PWMA_CCR1 寄存器，设置 CC1IF=1，若开启对应的中断，则产生相应的中断。若 CC1IF 已经为 1，则设置 CC1OF=1。

UGn：产生更新事件 该位由软件置 1，由硬件自动清 0。（n= A,B）

0：无动作；

1：重新初始化计数器，并产生一个更新事件。

注意预分频器的计数器也被清 0（但是预分频系数不变）。若在中心对称模式下或 DIR=0（向上计数）则计数器被清 0；若 DIR=1（向下计数）则计数器取 PWMn_ARR 的值。

29.9.13 捕获/比较模式寄存器 1（PWMx_CCMR1）

通道可用于输入（捕获模式）或输出（比较模式），通道的方向由相应的 CCnS 位定义。该寄存器其它位的作用在输入和输出模式下不同。OCxx 描述了通道在输出模式下的功能，ICxx 描述了通道在输入模式下的功能。因此必须注意，同一个位在输出模式和输入模式下的功能是不同的。

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR1	7EFEC8H	OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]	
PWMB_CCMR1	7EFEE8H	OC5CE	OC5M[2:0]			OC5PE	OC5FE	CC5S[1:0]	

OCnCE：输出比较 n 清零使能。该位用于使能使用 PWMETI 引脚上的外部事件来清通道 n 的输出信号（OCnREF）（n=1,5）

0：OCnREF 不受 ETRF 输入的影响；

1：一旦检测到 ETRF 输入高电平，OCnREF=0。

OCnM[2:0]：输出比较 n 模式。该 3 位定义了输出参考信号 OCnREF 的动作，而 OCnREF 决定了 OCn 的值。OCnREF 是高电平有效，而 OCn 的有效电平取决于 CCnP 位。（n=1,5）

OCnM[2:0]	模式	说明
000	冻结	PWMn_CCR1与PWMn_CNT间的比较对OCnREF不起作用
001	匹配时设置通道n的输出为有效电平	当PWMn_CCR1=PWMn_CNT时，OCnREF输出高
010	匹配时设置通道n的输出为无效电平	当PWMn_CCR1=PWMn_CNT时，OCnREF输出低
011	翻转	当PWMn_CCR1=PWMn_CNT时，翻转OCnREF
100	强制为无效电平	强制OCnREF为低
101	强制为有效电平	强制OCnREF为高
110	PWM模式1	在向上计数时，当PWMn_CNT<PWMn_CCR1时OCnREF输出高，否则OCnREF输出低 在向下计数时，当PWMn_CNT>PWMn_CCR1时OCnREF输出低，否则OCnREF输出高
111	PWM模式2	在向上计数时，当PWMn_CNT<PWMn_CCR1时OCnREF输出低，否则OCnREF输出高 在向下计数时，当PWMn_CNT>PWMn_CCR1时OCnREF输出高，否则OCnREF输出低

注 1：一旦 LOCK 级别设为 3（PWMn_BKR 寄存器中的 LOCK 位）并且 CCnS=00（该通道配置成输出）则该位不能被修改。

注 2：在 PWM 模式 1 或 PWM 模式 2 中，只有当比较结果改变了或在输出比较模式中从冻结模式切换到 PWM 模式时，OCnREF 电平才改变。

注 3：在有互补输出的通道上，这些位是预装载的。如果 PWMn_CR2 寄存器的 CCPC=1，OCM 位只有在 COM 事件发生时，才从预装载位取新值。

OCnPE：输出比较 n 预装载使能（n=1,5）

0：禁止 PWMn_CCR1 寄存器的预装载功能，可随时写入 PWMn_CCR1 寄存器，并且新写入的数值立即起作用。

1：开启 PWMn_CCR1 寄存器的预装载功能，读写操作仅对预装载寄存器操作，PWMn_CCR1 的预装载值在更新事件到来时被加载至当前寄存器中。

注 1：一旦 LOCK 级别设为 3（PWMn_BKR 寄存器中的 LOCK 位）并且 CCnS=00（该通道配置成输出）则该位不能被修改。

注 2：为了操作正确，在 PWM 模式下必须使能预装载功能。但在单脉冲模式下（PWMn_CR1 寄存器的 OPM=1），它不是必须的。

注：OC1PE 必须在通道打开时（PWMA_CCER1 寄存器的 CC1E=1）才是可写的。

注：OC5PE 必须在通道打开时（PWMB_CCER1 寄存器的 CC5E=1）才是可写的。

OCnFE：输出比较 n 快速使能。该位用于加快 CC 输出对触发输入事件的响应。（n=1,5）

0：根据计数器与 CCRn 的值，CCn 正常操作，即使触发器是打开的。当触发器的输入有一个有效沿时，激活 CCn 输出的最小延时为 5 个时钟周期。

1：输入到触发器的有效沿的作用就象发生了一次比较匹配。因此，OC 被设置为比较电平而与比较结果无关。采样触发器的有效沿和 CC1 输出间的延时被缩短为 3 个时钟周期。OCFE 只在通道被配置成 PWMA 或 PWMB 模式时起作用。

注：OC1FE 必须在通道打开时（PWMA_CCER1 寄存器的 CC1E=1）才是可写的。

注：OC5FE 必须在通道打开时（PWMB_CCER1 寄存器的 CC5E=1）才是可写的。

CC1S[1:0]：捕获/比较 1 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC1S[1:0]	方向	输入脚
00	输出	
01	输入	IC1映射在TI1FP1上
10	输入	IC1映射在TI2FP1上
11	输入	IC1映射在TRC上。此模式仅工作在内部触发器输入被选中时（由PWMA_SMCR寄存器的TS位选择）

CC5S[1:0]：捕获/比较 5 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC5S[1:0]	方向	输入脚
00	输出	
01	输入	IC5映射在TI5FP5上
10	输入	IC5映射在TI6FP5上
11	输入	IC5映射在TRC上。此模式仅工作在内部触发器输入被选中时（由PWM5_SMCR寄存器的TS位选择）

注：CC1S 仅在通道关闭时（PWMA_CCER1 寄存器的 CC1E=0）才是可写的。

注：CC5S 仅在通道关闭时（PWMB_CCER1 寄存器的 CC5E=0）才是可写的。

通道配置为捕获输入模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR1	7EFEC8H	IC1F[3:0]				IC1PSC[1:0]		CC1S[1:0]	
PWMB_CCMR1	7EFEE8H	IC5F[3:0]				IC5PSC[1:0]		CC5S[1:0]	

ICnF[3:0]：输入捕获 n 滤波器选择，该位域定义了 TIn 的采样频率及数字滤波器长度。（n=1,5）

ICnF[3:0]	时钟数	ICnF[3:0]	时钟数
0000	1	1000	48
0001	2	1001	64
0010	4	1010	80
0011	8	1011	96
0100	12	1100	128
0101	16	1101	160
0110	24	1110	192
0111	32	1111	256

注：即使对于带互补输出的通道，该位域也是非预装载的，并且不会考虑 CCPC（PWMn_CR2 寄存器）的值

ICnPSC[1:0]：输入/捕获 n 预分频器。这两位定义了 CCn 输入（IC1）的预分频系数。（n=1,5）

00：无预分频器，捕获输入口上检测到的每一个边沿都触发一次捕获

01：每 2 个事件触发一次捕获

10：每 4 个事件触发一次捕获

11：每 8 个事件触发一次捕获

注：IC1PSC 必须在通道打开时（PWMA_CCER1 寄存器的 CC1E=1）才是可写的。

注：IC5PSC 必须在通道打开时（PWMB_CCER1 寄存器的 CC5E=1）才是可写的。

CC1S[1:0]：捕获/比较 1 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC1S[1:0]	方向	输入脚
00	输出	
01	输入	IC1映射在TI1FP1上
10	输入	IC1映射在TI2FP1上
11	输入	IC1映射在TRC上。此模式仅工作在内部触发器输入被选中时（由PWMA_SMCR寄存器的TS位选择）

CC5S[1:0]：捕获/比较 5 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC5S[1:0]	方向	输入脚
00	输出	
01	输入	IC5映射在TI5FP5上
10	输入	IC5映射在TI6FP5上
11	输入	IC5映射在TRC上。此模式仅工作在内部触发器输入被选中时（由PWM5_SMCR寄存器的TS位选择）

注：CC1S 仅在通道关闭时（PWMA_CCER1 寄存器的 CC1E=0）才是可写的。

注：CC5S 仅在通道关闭时（PWMB_CCER1 寄存器的 CC5E=0）才是可写的。

PWMA_CCMR1 寄存器设置示例代码：

```
PWMA_CCER1 = 0x00;                  //必须关闭 CC1E 才能设置 CC1S
PWMA_CCMR1 = 0x01;                  //配置 CC1 为输入模式
PWMA_CCER1 |= 0x01;                 //设置 CC1E 为 1,使能 CC1 通道
PWMA_CCMR1 |= 0x04;                 //设置 IC1PSC, IC1PSC 必须在 CC1E 为 1 时才可写入
```

PWMA_CCMR2/PWMA_CCMR3/PWMA_CCMR4 以及 PWMB 组的
PWMB_CCMR1/PWMB_CCMR2/PWMB_CCMR3/PWMB_CCMR4 的设置方法与上面示例代码类似

29.9.14 捕获/比较模式寄存器 2（PWMx_CCMR2）

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR2	7EFEC9H	OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]	
PWMB_CCMR2	7EFEE9H	OC6CE	OC6M[2:0]			OC6PE	OC6FE	CC6S[1:0]	

OCnCE：输出比较 n 清零使能。该位用于使能使用 PWMETI 引脚上的外部事件来清通道 n 的输出信号（OCnREF）（n=2,6）

0：OCnREF 不受 ETRF 输入的影响；

1：一旦检测到 ETRF 输入高电平，OCnREF=0。

OCnM[2:0]：输出比较 2 模式，参考 OC1M。（n=2,6）

OCnPE：输出比较 2 预装载使能，参考 OP1PE。（n=2,6）

CC2S[1:0]：捕获/比较 2 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC2S[1:0]	方向	输入脚
00	输出	
01	输入	IC2映射在TI2FP2上
10	输入	IC2映射在TI1FP2上
11	输入	IC2映射在TRC上。

CC6S[1:0]：捕获/比较 6 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC6S[1:0]	方向	输入脚
00	输出	
01	输入	IC6映射在TI6FP6上
10	输入	IC6映射在TI5FP6上
11	输入	IC6映射在TRC上。

通道配置为捕获输入模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR2	7EFEC9H	IC2F[3:0]				IC2PSC[1:0]		CC2S[1:0]	
PWMB_CCMR2	7EFEE9H	IC6F[3:0]				IC6PSC[1:0]		CC6S[1:0]	

ICnF[3:0]：输入捕获 n 滤波器选择，参考 IC1F。（n=2,6）

ICnPSC[1:0]：输入/捕获 n 预分频器，参考 IC1PSC。（n=2,6）

CC2S[1:0]：捕获/比较 2 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC2S[1:0]	方向	输入脚
00	输出	
01	输入	IC2映射在TI2FP2上
10	输入	IC2映射在TI1FP2上
11	输入	IC2映射在TRC上。

CC6S[1:0]：捕获/比较 6 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC6S[1:0]	方向	输入脚
00	输出	
01	输入	IC6映射在TI6FP6上
10	输入	IC6映射在TI5FP6上
11	输入	IC6映射在TRC上。

29.9.15 捕获/比较模式寄存器 3（PWMx_CCMR3）

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR3	7EFECAH	OC3CE	OC3M[2:0]			OC3PE	OC3FE	CC3S[1:0]	
PWMB_CCMR3	7EFEEAH	OC7CE	OC7M[2:0]			OC7PE	OC7FE	CC7S[1:0]	

OCnCE：输出比较 n 清零使能。该位用于使能使用 PWMETI 引脚上的外部事件来清通道 n 的输出信号（OCnREF）（n=3,7）

0：OCnREF 不受 ETRF 输入的影响；

1：一旦检测到 ETRF 输入高电平，OCnREF=0。

OCnM[2:0]：输出比较 3 模式，参考 OC1M。（n=3,7）

OCnPE：输出比较 3 预装载使能，参考 OP1PE。（n=3,7）

CC3S[1:0]：捕获/比较 3 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC3S[1:0]	方向	输入脚
00	输出	
01	输入	IC3映射在TI3FP3上
10	输入	IC3映射在TI4FP3上
11	输入	IC3映射在TRC上。

CC7S[1:0]：捕获/比较 7 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC7S[1:0]	方向	输入脚
00	输出	
01	输入	IC7映射在TI7FP7上
10	输入	IC7映射在TI8FP7上
11	输入	IC7映射在TRC上。

通道配置为捕获输入模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR3	7EFECAH	IC3F[3:0]				IC3PSC[1:0]		CC3S[1:0]	
PWMB_CCMR3	7EFEEAH	IC7F[3:0]				IC7PSC[1:0]		CC7S[1:0]	

ICnF[3:0]：输入捕获 n 滤波器选择，参考 IC1F。（n=3,7）

ICnPSC[1:0]：输入/捕获 n 预分频器，参考 IC1PSC。（n=3,7）

CC3S[1:0]：捕获/比较 3 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC3S[1:0]	方向	输入脚
00	输出	
01	输入	IC3映射在TI3FP3上
10	输入	IC3映射在TI4FP3上
11	输入	IC3映射在TRC上。

CC7S[1:0]：捕获/比较 7 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC7S[1:0]	方向	输入脚
00	输出	
01	输入	IC7映射在TI7FP7上
10	输入	IC7映射在TI8FP7上
11	输入	IC7映射在TRC上。

29.9.16 捕获/比较模式寄存器 4（PWMx_CCMR4）

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR4	7EFECBH	OC4CE	OC4M[2:0]			OC4PE	OC4FE	CC4S[1:0]	
PWMB_CCMR4	7EFEEBH	OC8CE	OC8M[2:0]			OC8PE	OC8FE	CC8S[1:0]	

OCnCE：输出比较 n 清零使能。该位用于使能使用 PWMETI 引脚上的外部事件来清通道 n 的输出信号（OCnREF）（n=4,8）

0：OCnREF 不受 ETRF 输入的影响；

1：一旦检测到 ETRF 输入高电平，OCnREF=0。

OCnM[2:0]：输出比较 n 模式，参考 OC1M。（n=4,8）

OCnPE：输出比较 n 预装载使能，参考 OP1PE。（n=4,8）

CC4S[1:0]：捕获/比较 4 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC4S[1:0]	方向	输入脚
00	输出	
01	输入	IC4映射在TI4FP4上
10	输入	IC4映射在TI3FP4上
11	输入	IC4映射在TRC上。

CC8S[1:0]：捕获/比较 8 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC8S[1:0]	方向	输入脚
00	输出	
01	输入	IC8映射在TI8FP8上
10	输入	IC8映射在TI7FP8上
11	输入	IC8映射在TRC上。

通道配置为捕获输入模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR4	7EFECBH	IC4F[3:0]				IC4PSC[1:0]		CC4S[1:0]	
PWMB_CCMR4	7EFEEBH	IC8F[3:0]				IC8PSC[1:0]		CC8S[1:0]	

ICnF[3:0]：输入捕获 n 滤波器选择，参考 IC1F。（n=4,8）

ICnPSC[1:0]：输入/捕获 n 预分频器，参考 IC1PSC。（n=4,8）

CC4S[1:0]：捕获/比较 4 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC4S[1:0]	方向	输入脚
00	输出	
01	输入	IC4映射在TI4FP4上
10	输入	IC4映射在TI3FP4上
11	输入	IC4映射在TRC上。

CC8S[1:0]：捕获/比较 8 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC8S[1:0]	方向	输入脚
00	输出	
01	输入	IC8映射在TI8FP8上
10	输入	IC8映射在TI7FP8上
11	输入	IC8映射在TRC上。

29.9.17 捕获/比较使能寄存器 1（PWMx_CCER1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCER1	7EFECCH	CC2NP	CC2NE	CC2P	CC2E	CC1NP	CC1NE	CC1P	CC1E
PWMB_CCER1	7EFEECH	-	-	CC6P	CC6E	-	-	CC5P	CC5E

CC6P：OC6 输入捕获/比较输出极性。参考 CC1P

CC6E：OC6 输入捕获/比较输出使能。参考 CC1E

CC5P：OC5 输入捕获/比较输出极性。参考 CC1P

CC5E：OC5 输入捕获/比较输出使能。参考 CC1E

CC2NP：OC2N（OC2 通道负极）比较输出极性。参考 CC1NP

CC2NE：OC2N（OC2 通道负极）比较输出使能。参考 CC1NE

CC2P：OC2（OC2 通道正极）输入捕获/比较输出极性。参考 CC1P

CC2E：OC2（OC2 通道正极）输入捕获/比较输出使能。参考 CC1E

CC1NP：OC1N（OC1 通道负极）比较输出极性

0：高电平有效；

1：低电平有效。

注 1：一旦 LOCK 级别（PWMA_BKR 寄存器中的 LOCK 位）设为 3 或 2 且 CC1S=00（通道配置为输出），则该位不能被修改。

注 2：对于有互补输出的通道，该位是预装载的。如果 CCPC=1（PWMA_CR2 寄存器），只有在 COM 事件发生时，CC1NP 位才从预装载位中取新值。

CC1NE：OC1N（OC1 通道负极）比较输出使能

0：关闭比较输出。

1：开启比较输出，其输出电平依赖于 MOE、OSSI、OSSR、OIS1、OIS1N 和 CC1E 位的值。

注：对于有互补输出的通道，该位是预装载的。如果 CCPC=1（PWMA_CR2 寄存器），只有在 COM 事件发生时，CC1NE 位才从预装载位中取新值。

CC1P：OC1（OC1 通道正极）输入捕获/比较输出极性

CC1 通道配置为输出：

0：高电平有效

1：低电平有效

CC1 通道配置为输入或者捕获：

0：捕获发生在 TI1F 或 TI2F 的上升沿；

1：捕获发生在 TI1F 或 TI2F 的下降沿.

注 1：一旦 LOCK 级别（PWMA_BKR 寄存器中的 LOCK 位）设为 3 或 2，则该位不能被修改。

注 2：对于有互补输出的通道，该位是预装载的。如果 CCPC=1（PWMA_CR2 寄存器），只有在 COM 事件发生时，CC1P 位才从预装载位中取新值。

CC1E：OC1（OC1 通道正极）输入捕获/比较输出使能

0：关闭输入捕获/比较输出；

1：开启输入捕获/比较输出。

注：对于有互补输出的通道，该位是预装载的。如果 CCPC=1（PWMA_CR2 寄存器），只有在 COM 事件发生时，CC1E 位才从预装载位中取新值。

带刹车功能的互补输出通道 OCi 和 OCiN 的控制位

控制位					输出状态	
MOE	OSSI	OSSR	CCiE	CCiNE	OCi 输出状态	OCiN 输出状态
1	X	0	0	0	输出禁止	输出禁止
		0	0	1	输出禁止	带极性的 OCiREF
		0	1	0	带极性的 OCiREF	输出禁止
		0	1	1	带极性和死区的 OCiREF	带极性和死区的反向 OCiREF
		1	0	0	输出禁止	输出禁止
		1	0	1	关闭状态 （输出使能且为无效电平） OCi=CCiP	带极性的 OCiREF
		1	1	0	带极性的 OCiREF	关闭状态 （输出使能且为无效电平） OCiN=CCiNP
		1	1	1	带极性和死区的 OCiREF	带极性和死区的反向 OCiREF
0	0	X	X	X	输出禁止	
	1				关闭状态（输出使能且为无效电平）异步地：OCi=CCiP，OCiN=CCiNP；然后，若时钟存在：经过一个死区时间后 OCi=OISi，OCiN=OISiN，假设 OISi 与 OISiN 并不都对应 OCi 和 OCiN 的有效电平。	

注：管脚连接到互补的 OCi 和 OCiN 通道的外部 I/O 管脚的状态，取决于 OCi 和 OCiN 通道状态和 GPIO 寄存器。

29.9.18 捕获/比较使能寄存器 2（PWMx_CCER2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCER2	7EFECDH	CC4NP	CC4NE	CC4P	CC4E	CC3NP	CC3NE	CC3P	CC3E
PWMB_CCER2	7EFEEDH	-	-	CC8P	CC8E	-	-	CC7P	CC7E

CC8P：OC8 输入捕获/比较输出极性。参考 CC1P
CC8E：OC8 输入捕获/比较输出使能。参考 CC1E
CC7P：OC7 输入捕获/比较输出极性。参考 CC1P
CC7E：OC7 输入捕获/比较输出使能。参考 CC1E
CC4NP：OC4N（OC4 通道负极）比较输出极性。参考 CC1NP
CC4NE：OC4N（OC4 通道负极）比较输出使能。参考 CC1NE
CC4P：OC4（OC4 通道正极）输入捕获/比较输出极性。参考 CC1P
CC4E：OC4（OC4 通道正极）输入捕获/比较输出使能。参考 CC1E
CC3NP：OC3N（OC3 通道负极）比较输出极性。参考 CC1NP
CC3NE：OC3N（OC3 通道负极）比较输出使能。参考 CC1NE
CC3P：OC3（OC3 通道正极）输入捕获/比较输出极性。参考 CC1P
CC3E：OC3（OC3 通道正极）输入捕获/比较输出使能。参考 CC1E

29.9.19　计数器高 8 位（PWMx_CNTRH）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CNTRH	7EFECEH	CNT1[15:8]							
PWMB_CNTRH	7EFEEEH	CNT2[15:8]							

CNTn[15:8]：计数器的高 8 位值（n= A,B）

29.9.20　计数器低 8 位（PWMx_CNTRL）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CNTRL	7EFECFH	CNT1[7:0]							
PWMB_CNTRL	7EFEEFH	CNT2[7:0]							

CNTn[7:0]：计数器的低 8 位值（n= A,B）

29.9.21　预分频器高 8 位（PWMx_PSCRH），输出频率计算公式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_PSCRH	7EFED0H	PSC1[15:8]							
PWMB_PSCRH	7EFEF0H	PSC2[15:8]							

PSCn[15:8]：预分频器的高 8 位值。（n= A,B）

预分频器用于对 CK_PSC 进行分频。计数器的时钟频率（fCK_CNT）等于 fCK_PSC/（PSCR[15:0]+1）。PSCR 包含了当更新事件产生时装入当前预分频器寄存器的值（更新事件包括计数器被 TIM_EGR 的 UG 位清 0 或被工作在复位模式的从控制器清 0）。这意味着为了使新的值起作用，必须产生一个更新事件。

PWM 输出频率计算公式

PWMA 和 PWMB 两组 PWM 的输出频率计算公式相同，且每组可设置不同的频率。

对齐模式	PWM输出频率计算公式
边沿对齐	$\text{PWM输出频率} = \dfrac{\text{系统工作频率SYSclk}}{(\text{PWMx_PSCR} + 1) \times (\text{PWMx_ARR} + 1)}$
中间对齐	$\text{PWM输出频率} = \dfrac{\text{系统工作频率SYSclk}}{(\text{PWMx_PSCR} + 1) \times \text{PWMx_ARR} \times 2}$

29.9.22　预分频器低 8 位（PWMx_PSCRL）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_PSCRL	7EFED1H	PSC1[7:0]							
PWMB_PSCRL	7EFEF1H	PSC2[7:0]							

PSCn[7:0]：预分频器的低 8 位值。（n= A,B）

29.9.23 自动重装载寄存器高 8 位（PWMx_ARRH）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ARRH	7EFED2H	ARR1[15:8]							
PWMB_ARRH	7EFEF2H	ARR2[15:8]							

ARRn[15:8]：自动重装载高 8 位值（n= A,B）

ARR 包含了将要装载入实际的自动重装载寄存器的值。当自动重装载的值为 0 时，计数器不工作。

29.9.24 自动重装载寄存器低 8 位（PWMx_ARRL）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ARRL	7EFED3H	ARR1[7:0]							
PWMB_ARRL	7EFEF3H	ARR2[7:0]							

ARRn[7:0]：自动重装载低 8 位值（n= A,B）

29.9.25 重复计数器寄存器（PWMx_RCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_RCR	7EFED4H	REP1[7:0]							
PWMA_RCRH	7EFEB9H	REP1[15:8]							
PWMB_RCR	7EFEF4H	REP2[7:0]							
PWMB_RCRH	7EFEBAH	REP2[15:8]							

REPn[15:0]：重复计数器值（n= A,B）

开启了预装载功能后，这些位允许用户设置比较寄存器的更新速率（即周期性地从预装载寄存器 传输到当前寄存器）；如果允许产生更新中断，则会同时影响产生更新中断的速率。每次向下计数器 REP_CNT 达到 0，会产生一个更新事件并且计数器 REP_CNT 重新从 REP 值开 始计数。由于 REP_CNT 只有在周期更新事件 U_RC 发生时才重载 REP 值，因此对 PWMn_RCR 寄存器写入的新值只在下次周期更新事件发生时才起作用。这意味着在 PWM 模式中，（REP+1）对应着：

- 在边沿对齐模式下，PWM 周期的数目；
- 在中心对称模式下，PWM 半周期的数目。

29.9.26 捕获/比较寄存器 1/5 高 8 位（PWMx_CCR1H）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR1H	7EFED5H	CCR1[15:8]							
PWMB_CCR5H	7EFEF5H	CCR5[15:8]							

CCRn[15:8]：捕获/比较 n 的高 8 位值（n=1,5）

若 CCn 通道配置为输出：CCRn 包含了装入当前比较值（预装载值）。如果在 PWMn_CCMR1 寄存器（OCnPE 位）中未选择预装载功能，写入的数值会立即传输至当前寄存器中。否则只有当更新事件发生时，此预装载值才传输至当前捕获/比较 n 寄存器中。当前比较值同计数器 PWMn_CNT 的值相比较，并在 OCn 端口上产生输出信号。

若 CCn 通道配置为输入：CCRn 包含了上一次输入捕获事件发生时的计数器值（此时该寄存器为只读）。

29.9.27 捕获/比较寄存器 1/5 低 8 位（PWMx_CCR1L）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR1L	7EFED6H	CCR1[7:0]							
PWMB_CCR5L	7EFEF6H	CCR5[7:0]							

CCRn[7:0]：捕获/比较 n 的低 8 位值（n=1,5）

29.9.28 捕获/比较寄存器 2/6 高 8 位（PWMx_CCR2H）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR2H	7EFED7H	CCR2[15:8]							
PWMB_CCR6H	7EFEF7H	CCR6[15:8]							

CCRn[15:8]：捕获/比较 n 的高 8 位值（n=2,6）

29.9.29 捕获/比较寄存器 2/6 低 8 位（PWMx_CCR2L）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR2L	7EFED8H	CCR2[7:0]							
PWMB_CCR6L	7EFEF8H	CCR6[7:0]							

CCRn[7:0]：捕获/比较 n 的低 8 位值（n=2,6）

29.9.30 捕获/比较寄存器 3/7 高 8 位（PWMx_CCR3H）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR3H	7EFED9H	CCR3[15:8]							
PWMB_CCR7H	7EFEF9H	CCR7[15:8]							

CCRn[15:8]：捕获/比较 n 的高 8 位值（n=3,7）

29.9.31 捕获/比较寄存器 3/7 低 8 位（PWMx_CCR3L）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR3L	7EFEDAH	CCR3[7:0]							
PWMB_CCR7L	7EFEFAH	CCR7[7:0]							

CCRn[7:0]：捕获/比较 n 的低 8 位值（n=3,7）

29.9.32 捕获/比较寄存器 4/8 高 8 位（PWMx_CCR4H）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR4H	7EFEDBH	CCR4[15:8]							
PWMB_CCR8H	7EFEFBH	CCR8[15:8]							

CCRn[15:8]：捕获/比较 n 的高 8 位值（n=4,8）

29.9.33 捕获/比较寄存器 4/8 低 8 位（PWMx_CCR4L）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR4L	7EFEDCH	CCR4[7:0]							
PWMB_CCR8L	7EFEFCH	CCR8[7:0]							

CCRn[7:0]：捕获/比较 n 的低 8 位值（n=4,8）

29.9.34 刹车寄存器（PWMx_BKR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_BKR	7EFEDDH	MOEA	AOEA	BKPA	BKEA	OSSRA	OSSIA	LOCKA[1:0]	
PWMB_BKR	7EFEFDH	MOEB	AOEB	BKPB	BKEB	OSSRB	OSSIB	LOCKB[1:0]	

MOEn：主输出使能。一旦刹车输入有效，该位被硬件异步清 0。根据 AOE 位的设置值，该位可以由软件置 1 或被自动置 1。它仅对配置为输出的通道有效。（n= A,B）

0：禁止 OC 和 OCN 输出或强制为空闲状态

1：如果设置了相应的使能位（PWMn_CCERX 寄存器的 CCIE 位），则使能 OC 和 OCN 输出。

注：一旦 LOCK 级别（PWMn_BKR 寄存器中的 LOCK 位）设为 1，则该位不能被修改

AOEn：自动输出使能（n= A,B）

0：MOE 只能被软件置 1；

1：MOE 能被软件置 1 或在下一个更新事件被自动置 1（如果刹车输入无效）。

注：一旦 LOCK 级别（PWMn_BKR 寄存器中的 LOCK 位）设为 1，则该位不能被修改

BKPn：刹车输入极性（n= A,B）

0：刹车输入低电平有效

1：刹车输入高电平有效

注：一旦 LOCK 级别（PWMn_BKR 寄存器中的 LOCK 位）设为 1，则该位不能被修改

BKEn：刹车功能使能（n= A,B）

0：禁止刹车输入（BRK）

1：开启刹车输入（BRK）

注：一旦 LOCK 级别（PWMn_BKR 寄存器中的 LOCK 位）设为 1，则该位不能被修改。

OSSRn：运行模式下“关闭状态”选择。该位在 MOE=1 且通道设为输出时有效（n= A,B）

0：当 PWM 不工作时，禁止 OC/OCN 输出（OC/OCN 使能输出信号=0）；

1：当 PWM 不工作时，一旦 CCiE=1 或 CCiNE=1，首先开启 OC/OCN 并输出无效电平，然后置 OC/OCN 使能输出信号=1。

注：一旦 LOCK 级别（PWMn_BKR 寄存器中的 LOCK 位）设为 2，则该位不能被修改。

OSSIn：空闲模式下“关闭状态”选择。该位在 MOE=0 且通道设为输出时有效。（n= A,B）

0：当 PWM 不工作时，禁止 OC/OCN 输出（OC/OCN 使能输出信号=0）；

1：当 PWM 不工作时，一旦 CCiE=1 或 CCiNE=1，OC/OCN 首先输出其空闲电平，然后 OC/OCN 使能输出信号=1。

注：一旦 LOCK 级别（PWMn_BKR 寄存器中的 LOCK 位）设为 2，则该位不能被修改。

LOCKn[1:0]：锁定设置。该位为防止软件错误而提供的写保护措施（n= A,B）

LOCKn[1:0]	保护级别	保护内容
00	无保护	寄存器无写保护
01	锁定级别1	不能写入PWMn_BKR寄存器的MOE、BKE、BKP、AOE位和PWMn_OISR寄存器的OISI位
10	锁定级别2	不能写入锁定级别1中的各位， 也不能写入CC极性位以及OSSR/OSSI位
11	锁定级别3	不能写入锁定级别2中的各位， 也不能写入CC控制位

注：由于 MOE、BKE、BKP、AOE、OSSR、OSSI 位可被锁定（依赖于 LOCK 位），因此在第一次写 PWMn_BKR 寄存器时必须对它们进行设置。

29.9.35 死区寄存器（PWMx_DTR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_DTR	7EFEDEH	DTGA[7:0]							
PWMB_DTR	7EFEFEH	DTGB[7:0]							

DTGn[7:0]：死区发生器设置。（n= A,B）

这些位定义了插入互补输出之间的死区持续时间。（t_{CK_PSC} 为 PWMn 的时钟脉冲）

DTGn[7:5]	死区时间
000	DTGn[7:0] * t_{CK_PSC}
001	
010	
011	
100	(64 + DTGn[6:0]) * 2 * t_{CK_PSC}
101	
110	(32 + DTGn[5:0]) * 8 * t_{CK_PSC}
111	(32 + DTGn[4:0]) * 16 * t_{CK_PSC}

注：一旦 LOCK 级别（PWMx_BKR 寄存器中的 LOCK 位）设为 1、2 或 3 时，则该位不能被修改。

29.9.36 输出空闲状态寄存器（PWMx_OISR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_OISR	7EFEDFH	OIS4N	OIS4	OIS3N	OIS3	OIS2N	OIS2	OIS1N	OIS1
PWMB_OISR	7EFEFFH	-	OIS8	-	OIS7	-	OIS6	-	OIS5

OIS8：空闲状态时 OC8 输出电平

OIS7：空闲状态时 OC7 输出电平

OIS6：空闲状态时 OC6 输出电平

OIS5：空闲状态时 OC5 输出电平

OIS4N：空闲状态时 OC4N 输出电平

OIS4：空闲状态时 OC4 输出电平

OIS3N：空闲状态时 OC3N 输出电平

OIS3：空闲状态时 OC3 输出电平

OIS2N：空闲状态时 OC2N 输出电平

OIS2：空闲状态时 OC2 输出电平

OIS1N：空闲状态时 OC1N 输出电平

0：当 MOE=0 时，则在一个死区时间后，OC1N=0；

1：当 MOE=0 时，则在一个死区时间后，OC1N=1。

注：一旦 LOCK 级别（PWMx_BKR 寄存器中的 LOCK 位）设为 1、2 或 3 时，则该位不能被修改。

OIS1：空闲状态时 OC1 输出电平

0：当 MOE=0 时，如果 OC1N 使能，则在一个死区后，OC1=0；

1：当 MOE=0 时，如果 OC1N 使能，则在一个死区后，OC1=1。

注：一旦 LOCK 级别（PWMx_BKR 寄存器中的 LOCK 位）设为 1、2 或 3 时，则该位不能被修改。

29.10　范例程序及应用线路图

29.10.1　PWMA+PWMB 实现 8 组定时器

```
//测试工作频率为 12MHz

/************ 功能说明 **************
本例程基于 AI32G12K128 为主控芯片的实验箱 9 进行编写测试.
利用高级 PWMA+PWMB 中断实现 8 组定时器功能.

PWMA 的时钟频率为 SYSclk/2
PWMA 通道 1：定时周期 1ms
PWMA 通道 2：定时周期 2ms
PWMA 通道 3：定时周期 4ms
PWMA 通道 4：定时周期 5ms

PWMB 的时钟频率为 SYSclk/10000
PWMB 通道 1：定时周期 1s
PWMB 通道 2：定时周期 2s
PWMB 通道 3：定时周期 3s
PWMB 通道 4：定时周期 4s

下载时, 选择时钟 24MHZ (用户可自行修改频率).
*****************************************/

#include   “Ai8051U.H”
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define    MAIN_Fosc    24000000L                       //定义主时钟

#define    PSCRA        2
#define    PWMA_T1      MAIN_Fosc/PSCRA/1000            //定时周期 1ms(1KHz)
#define    PWMA_T2      MAIN_Fosc/PSCRA/500             //定时周期 2ms(500Hz)
#define    PWMA_T3      MAIN_Fosc/PSCRA/250             //定时周期 4ms(250Hz)
#define    PWMA_T4      MAIN_Fosc/PSCRA/200             //定时周期 5ms(200Hz)

#define    PSCRB        10000
#define    PWMB_T5      MAIN_Fosc/PSCRB*1               //定时周期 1s
#define    PWMB_T6      MAIN_Fosc/PSCRB*2               //定时周期 2s
#define    PWMB_T7      MAIN_Fosc/PSCRB*3               //定时周期 3s
#define    PWMB_T8      MAIN_Fosc/PSCRB*4               //定时周期 4s

typedef    unsigned char    u8;
typedef    unsigned int     u16;
typedef    unsigned long    u32;

/*****************************************************************************/

void PWMA_Timer();
void PWMB_Timer();

u16 cnt1, cnt2, cnt3, cnt4, cnt5, cnt6, cnt7, cnt8;

/******************** 主函数 **************************/
void main(void)
```

```
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M1 = 0x00;    P0M0 = 0x00;                //设置为准双向口
    P1M1 = 0x00;    P1M0= 0x00;                 //设置为准双向口
    P2M1 = 0x00;    P2M0= 0x00;                 //设置为准双向口
    P3M1 = 0x00;    P3M0= 0x00;                 //设置为准双向口
    P4M1 = 0x00;    P4M0= 0x00;                 //设置为准双向口
    P5M1 = 0x00;    P5M0= 0x00;                 //设置为准双向口
    P6M1 = 0x00;    P6M0= 0x00;                 //设置为准双向口
    P7M1 = 0x00;    P7M0= 0x00;                 //设置为准双向口

    PWMA_Timer();
    PWMB_Timer();

    P40  = 0;                                   //给 LED 供电
    EA = 1;                                     //打开总中断

    while (1);
}

//========================================================================
// 函数: void PWMA_Timer()
// 描述: PWMA 配置函数.
//========================================================================
void PWMA_Timer()
{
    PWMA_PSCRH = (PSCRA-1) >> 8;
    PWMA_PSCRL = (PSCRA-1);                     //设置预分频器
    PWMA_ARRH = 0xff;
    PWMA_ARRL =0xff;
    PWMA_CCER1 = 0x00;
    PWMA_CCMR1 = 0x00;
    PWMA_CCMR2 = 0x00;
    PWMA_CCMR3 = 0x00;
    PWMA_CCMR4 = 0x00;
    cnt1 = PWMA_T1;
    cnt2 = PWMA_T2;
    cnt3 = PWMA_T3;
    cnt4 = PWMA_T4;
    PWMA_CCR1H = cnt1 >> 8;
    PWMA_CCR1L = cnt1;
    PWMA_CCR2H = cnt2 >> 8;
    PWMA_CCR2L = cnt2;
    PWMA_CCR3H = cnt3 >> 8;
    PWMA_CCR3L = cnt3;
    PWMA_CCR4H = cnt4 >> 8;
    PWMA_CCR4L = cnt4;
    PWMA_IER = 0x1e;                            // 使能中断
    PWMA_CR1 |= 0x81;                           //使能 ARR 预装载，开始计时

    cnt1 +=    PWMA_T1;
```

```
    cnt2 +=     PWMA_T2;
    cnt3 +=     PWMA_T3;
    cnt4 +=     PWMA_T4;
}

//========================================================================
// 函数: void PWMB_Timer()
// 描述: PWMB 配置函数.
//========================================================================
void PWMB_Timer()
{
    PWMB_PSCRH = (PSCRB-1) >> 8;
    PWMB_PSCRL = (PSCRB-1);                         //设置预分频器
    PWMB_ARRH = 0xff;
    PWMB_ARRL =0xff;
    PWMB_CCER1 = 0x00;
    PWMB_CCMR1 = 0x00;
    PWMB_CCMR2 = 0x00;
    PWMB_CCMR3 = 0x00;
    PWMB_CCMR4 = 0x00;
    cnt5 = PWMB_T5;
    cnt6 = PWMB_T6;
    cnt7 = PWMB_T7;
    cnt8 = PWMB_T8;
    PWMB_CCR5H = cnt5 >> 8;
    PWMB_CCR5L = cnt5;
    PWMB_CCR6H = cnt6 >> 8;
    PWMB_CCR6L = cnt6;
    PWMB_CCR7H = cnt7 >> 8;
    PWMB_CCR7L = cnt7;
    PWMB_CCR8H = cnt8 >> 8;
    PWMB_CCR8L = cnt8;
    PWMB_IER = 0x1e;                                //  使能中断
    PWMB_CR1 |= 0x81;                               //使能 ARR 预装载，开始计时

    cnt5 +=     PWMB_T5;
    cnt6 +=     PWMB_T6;
    cnt7 +=     PWMB_T7;
    cnt8 +=     PWMB_T8;
}

//========================================================================
// 函数: PWMA_ISR()       interrupt 26
// 描述: PWMA 中断函数.
//========================================================================
void PWMA_ISR()     interrupt 26
{
    u8 sr;

    sr = PWMA_SR1;
    PWMA_SR1 = 0;

    if (sr & 0x02)
    {
        PWMA_CCR1H = cnt1 >> 8;                     //更新比较值
        PWMA_CCR1L = cnt1;
        cnt1 +=     PWMA_T1;                        //计算下一个比较值
        P60  = ~P60;
    }
```

```
    if (sr & 0x04)
    {
        PWMA_CCR2H = cnt2 >> 8;                 //更新比较值
        PWMA_CCR2L = cnt2;
        cnt2 +=  PWMA_T2;                       //计算下一个比较值
        P61  = ~P61;
    }
    if (sr & 0x08)
    {
        PWMA_CCR3H = cnt3 >> 8;                 //更新比较值
        PWMA_CCR3L = cnt3;
        cnt3 +=  PWMA_T3;                       //计算下一个比较值
        P62  = ~P62;
    }
    if (sr & 0x10)
    {
        PWMA_CCR4H = cnt4 >> 8;                 //更新比较值
        PWMA_CCR4L = cnt4;
        cnt4 +=  PWMA_T4;                       //计算下一个比较值
        P63  = ~P63;
    }
}

//========================================================================
// 函数: PWMB_ISR()       interrupt 27
// 描述: PWMB 中断函数.
//========================================================================
void PWMB_ISR()     interrupt 27
{
    u8 sr;

    sr = PWMB_SR1;
    PWMB_SR1 = 0;

    if (sr & 0x02)
    {
        PWMB_CCR5H = cnt5 >> 8;                 //更新比较值
        PWMB_CCR5L = cnt5;
        cnt5 +=  PWMB_T5;                       //计算下一个比较值
        P64  = ~P64;
    }
    if (sr & 0x04)
    {
        PWMB_CCR6H = cnt6 >> 8;                 //更新比较值
        PWMB_CCR6L = cnt6;
        cnt6 +=  PWMB_T6;                       //计算下一个比较值
        P65  = ~P65;
    }
    if (sr & 0x08)
    {
        PWMB_CCR7H = cnt7 >> 8;                 //更新比较值
        PWMB_CCR7L = cnt7;
        cnt7 +=  PWMB_T7;                       //计算下一个比较值
        P66  = ~P66;
    }
    if (sr & 0x10)
    {
        PWMB_CCR8H = cnt8 >> 8;                 //更新比较值
        PWMB_CCR8L = cnt8;
```

```
            cnt8 +=     PWMB_T8;                    //计算下一个比较值
            P67  = ~P67;
        }
}
```

29.10.2 高级 PWM 时钟输出应用（系统时钟分频输出）

```
//测试工作频率为 24MHz

#include "Ai8051U.H"

#define     FOSC                24000000UL

//2 分频输出
#define     PWM_PERIOD          (2-1)           //定义 PWM 周期值
                                                //(频率=FOSC/(PWM_PERIOD+1)=12MHz)
#define     PWM_DUTY            (1)             //定义 PWM 的占空比值 50%

////3 分频输出
//#define   PWM_PERIOD          (3-1)           //定义 PWM 周期值
//                                              //(频率=FOSC/(PWM_PERIOD+1)=8MHz)
//#define   PWM_DUTY            (1)             //定义 PWM 的占空比值 33%

////4 分频输出
//#define   PWM_PERIOD          (4-1)           //定义 PWM 周期值
//                                              //(频率=FOSC/(PWM_PERIOD+1)=6MHz)
//#define   PWM_DUTY            (2)             //定义 PWM 的占空比值 50%

////N 分频输出
//#define   PWM_PERIOD          (N-1)           //定义 PWM 周期值
//                                              //(频率=FOSC/(PWM_PERIOD+1))
//#define   PWM_DUTY            (N/2)           //定义 PWM 的占空比值

void SYS_Init();
void PWM_Init();

void main()
{
    SYS_Init();
    PWM_Init();

    while (1);
}

void SYS_Init()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M1 = 0x00;    P0M0 = 0x00;
    P1M1 = 0x00;    P1M0 = 0x00;
```

```
    P2M1 = 0x00;    P2M0 = 0x00;
    P3M1 = 0x00;    P3M0 = 0x00;
    P4M1 = 0x00;    P4M0 = 0x00;
    P5M1 = 0x00;    P5M0 = 0x00;
    P6M1 = 0x00;    P6M0 = 0x00;
    P7M1 = 0x00;    P7M0 = 0x00;
}

void PWM_Init()
{
    PWMA_CCER1 = 0x00;                          //写 CCMRx 前必须先清零 CCxE 关闭通道
    PWMA_CCMR1 = 0x60;                          //通道模式配置 PWM 模式 1
    PWMA_CCER1 = 0x01;                          //配置通道输出使能和极性
    PWMA_ARRH = (char)(PWM_PERIOD >> 8);        //设置周期时间
    PWMA_ARRL = (char)(PWM_PERIOD);
    PWMA_CCR1H = (char)(PWM_DUTY >> 8);         //设置占空比时间
    PWMA_CCR1L = (char)(PWM_DUTY);
    PWMA_ENO = 0x01;                            //使能 PWM 输出
    PWMA_BKR = 0x80;                            //使能主输出
    PWMA_CR1 = 0x01;                            //开始计时
}
```

29.10.3　输出任意周期和任意占空比的波形

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                           //头文件见下载软件
#include "intrins.h"                            //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void main()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    PWMA_CCER1 = 0x00;                          //写 CCMRx 前必须先清零 CCERx 关闭通道
    PWMA_CCMR1 = 0x60;                          //设置 CC1 为 PWMA 输出模式
```

```
    PWMA_CCER1 = 0x01;                          //使能 CC1 通道
    PWMA_CCR1H = 0x01;                          //设置占空比时间
    PWMA_CCR1L = 0x00;
    PWMA_ARRH = 0x02;                           //设置周期时间
    PWMA_ARRL = 0x00;
    PWMA_ENO = 0x01;                            //使能 PWM1P 端口输出
    PWMA_BKR = 0x80;                            //使能主输出
    PWMA_CR1 = 0x01;                            //开始计时

    while (1);
}
```

29.10.4输出占空比为 100%和 0%的 PWM 波形的方法（以 PWM1P 为例）

29.10.4.1　　方法 1：设置 PWMx_ENO 禁止输出 PWM

```
//测试工作频率为11.0592MHz
#include “Ai8051U.H”                    //头文件见下载软件
#include "intrins.h"                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                           //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    PWMA_ENO &= ~0x01;                  //禁止 PWM1P 端口输出
                                        //此时 PWM1P 端口为GPIO，可通过
                                        //直接操作 I/O 口寄存器强制输出高电平或者低电平

    while (1);
}
```

29.10.4.2　　方法 2：设置 PWMx_CCMRn 寄存器强制输出有效/无效电平

C 语言代码

```
//测试工作频率为11.0592MHz
#include “Ai8051U.H”                    //头文件见下载软件
#include "intrins.h"                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
```

```
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    PWMA_CCER1 = 0x00;
    PWMA_CCMR1 &= ~0x03;                        //CC1 为输出模式
    PWMA_CCMR1 |= 0x40;                         //CC1 强制输出无效电平（占空比 0%）
//  PWMA_CCMR1 |= 0x50;                         //CC1 强制输出有效电平（占空比 100%）
    PWMA_CCER1 |= 0x01;                         //使能 CC1 输出，且设置高电平为有效电平
    PWMA_ENO = 0x01;                            //使能 PWM1P 端口输出
    PWMA_BKR = 0x80;                            //使能主输出
    PWMA_CR1 = 0x01;                            //开始计时

    while (1);
}
```

29.10.5 高级 PWM 输出-频率可调-脉冲计数（软件方式）

```
//测试工作频率为 24MHz
/*************  功能说明  **************
高级 PWM 定时器实现高速 PWM 脉冲输出.
周期/占空比可调, 通过比较/捕获中断进行脉冲个数计数.
通过 P6 口演示输出,每隔 10ms 输出一次 PWM,计数 10 个脉冲后停止输出.
定时器每 1ms 调整 PWM 周期.
下载时,选择时钟 24MHZ (用户可自行修改频率).
*****************************************/

#include “Ai8051U.H”                           //头文件见下载软件
#include "intrins.h"                            //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define    MAIN_Fosc        24000000L

typedef    unsigned char    u8;
typedef    unsigned int     u16;
typedef    unsigned long    u32;

/*****************************用户定义宏 ***********************************/

#define    Timer0_Reload    (65536UL -(MAIN_Fosc / 1000))        //Timer0 中断频率,1000 次/秒

/******************************************************************************/

#define    PWM1_1           0x00                //P:P1.0    N:P1.1
#define    PWM1_2           0x01                //P:P2.0    N:P2.1
```

```
#define     PWM1_3          0x02                //P:P6.0   N:P6.1

#define     PWM2_1          0x00                //P:P1.2   N:P1.3
#define     PWM2_2          0x04                //P:P2.2   N:P2.3
#define     PWM2_3          0x08                //P:P6.2   N:P6.3

#define     PWM3_1          0x00                //P:P1.4   N:P1.5
#define     PWM3_2          0x10                //P:P2.4   N:P2.5
#define     PWM3_3          0x20                //P:P6.4   N:P6.5

#define     PWM4_1          0x00                //P:P1.6   N:P1.7
#define     PWM4_2          0x40                //P:P2.6   N:P2.7
#define     PWM4_3          0x80                //P:P6.6   N:P6.7
#define     PWM4_4          0xC0                //P:P3.4   N:P3.3

#define     ENO1P           0x01
#define     ENO1N           0x02
#define     ENO2P           0x04
#define     ENO2N           0x08
#define     ENO3P           0x10
#define     ENO3N           0x20
#define     ENO4P           0x40
#define     ENO4N           0x80

/*************本地变量声明**************/
bit     B_1ms;                                  //1ms 标志
bit     PWM1_Flag;

u16   Period;
u8 Counter;
u8 msSecond;

void UpdatePwm(void);
void TxPulse(void);

/********************* 主函数 **************************/
void main(void)
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M1 = 0x00;    P0M0 = 0x00;                //设置为准双向口
    P1M1 = 0x00;    P1M0 = 0x00;                //设置为准双向口
    P2M1 = 0x00;    P2M0 = 0x00;                //设置为准双向口
    P3M1 = 0x00;    P3M0 = 0x00;                //设置为准双向口
    P4M1 = 0x00;    P4M0 = 0x00;                //设置为准双向口
    P5M1 = 0x00;    P5M0 = 0x00;                //设置为准双向口
    P6M1 = 0x00;    P6M0 = 0x00;                //设置为准双向口
    P7M1 = 0x00;    P7M0 = 0x00;                //设置为准双向口

    PWM1_Flag = 0;
    Counter = 0;
    Period = 0x1000;
```

```
    //Timer0 初始化
    AUXR = 0x80;                                //Timer0 set as 1T,16 bits timer auto-reload,
    TH0 = (u8)(Timer0_Reload / 256);
    TL0 = (u8)(Timer0_Reload % 256);
    ET0 = 1;                                    //Timer0 interrupt enable
    TR0 = 1;                                    //Tiner0 run

    PWMA_ENO = 0x00;
    PWMA_ENO |= ENO1P;                          //使能输出

    PWMA_PS = 0x00;                             //高级 PWM 通道输出脚选择位
    PWMA_PS |= PWM1_3;                          //选择 PWM1_3 通道

    UpdatePwm();
    PWMA_BKR = 0x80;                            //使能主输出
    PWMA_CR1 |= 0x01;                           //开始计时

    P40 = 0;                                    //给 LED 供电
    EA = 1;                                     //打开总中断

    while (1)
    {
        if(B_1ms)
        {
            B_1ms = 0;
            msSecond++;
            if(msSecond >= 10)
            {
                msSecond = 0;
                TxPulse();                      //10ms 启动一次 PWM 输出
            }
        }
    }
}

/************* 发送脉冲函数 **************/
void TxPulse(void)
{
    PWMA_CCER1 = 0x00;                          //写 CCMRx 前必须先清零 CCxE  关闭通道
    PWMA_CCMR1 = 0x60;                          //设置 PWM1 模式 1 输出
    PWMA_CCER1 = 0x01;                          //使能 CC1E 通道, 高电平有效
    PWMA_SR1 = 0;                               //清标志位
    PWMA_CNTRH = 0;                             //清计数器
    PWMA_CNTRL = 0;                             //清计数器
    PWMA_IER = 0x02;                            //使能捕获/比较 1 中断
}

/*********************       Timer0 1ms 中断函数 **********************/
void timer0(void) interrupt 1
{
    B_1ms = 1;
    if(PWM1_Flag)
    {
        Period++;                               //周期递增
        if(Period >= 0x1000) PWM1_Flag = 0;
    }
    else
    {
```

```
            Period--;                                   //周期递减
            if(Period <= 0x0100) PWM1_Flag = 1;
        }
        UpdatePwm();                                    //设置周期、占空比
}

/******************** PWM 中断函数 ********************/
void PWMA_ISR() interrupt 26
{
        if(PWMA_SR1 & 0X02)
        {
            PWMA_SR1 &=~0X02;                           //清标志位

            Counter++;
            if(Counter >= 10)                           //计数 10 个脉冲后关闭 PWM 计数器
            {
                Counter = 0;
                PWMA_CCER1 = 0x00;                      //写 CCMRx 前必须先清零 CCxE   关闭通道
                PWMA_CCMR1 = 0x40;                      //设置 PWM1 强制为无效电平
                PWMA_CCER1 = 0x01;                      //使能 CC1E 通道, 高电平有效
                PWMA_IER = 0x00;                        //关闭中断
            }
        }
}

//========================================================================
// 函数: UpdatePwm(void)
// 描述: 更新 PWM 周期占空比.
// 参数: none.
// 返回: none.
// 版本: V1.0
//========================================================================
void UpdatePwm(void)
{
        PWMA_ARRH = (u8)(Period >> 8);
        PWMA_ARRL = (u8)Period;
        PWMA_CCR1 = (u8)((Period >> 1) >> 8);           //设置占空比时间: Period/2
        PWMA_CCR1 = (u8)(Period >> 1);
}
```

29.10.6 高级 PWM 输出-频率可调-脉冲计数（硬件方式）

```
//测试工作频率为 24MHz

/************* 功能说明 **************
本例程基于 AI32G 为主控芯片的实验箱进行编写测试。
高级 PWM 定时器实现高速 PWM 脉冲输出.
周期/占空比可调, 通过比较/捕获中断进行脉冲个数计数.
通过 P6 口演示输出,每隔 10ms 输出一次 PWM,计数 10 个脉冲后停止输出.
使用单脉冲模式配合重复计数寄存器，纯硬件控制脉冲个数.
定时器每 1ms 调整 PWM 周期.
下载时,    选择时钟 24MHZ (用户可自行修改频率).
******************************************/

#include “Ai8051U.H”
```

```
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define   MAIN_Fosc         24000000L                   //定义主时钟

typedef   unsigned char   u8;
typedef   unsigned int    u16;
typedef   unsigned long   u32;                          //1:printf 使用 UART1; 2:printf 使用 UART2

/*****************************  用户定义宏  *********************************/

#define   Timer0_Reload  (65536UL -(MAIN_Fosc / 1000))  //Timer   0  中断频率, 1000 次/秒

#define   PWM1_0        0x00                            //P:P1.0   N:P1.1
#define   PWM1_1        0x01                            //P:P2.0   N:P2.1
#define   PWM1_2        0x02                            //P:P6.0   N:P6.1

#define   PWM2_0        0x00                            //P:P1.2   N:P1.3
#define   PWM2_1        0x04                            //P:P2.2   N:P2.3
#define   PWM2_2        0x08                            //P:P6.2   N:P6.3

#define   PWM3_0        0x00                            //P:P1.4   N:P1.5
#define   PWM3_1        0x10                            //P:P2.4   N:P2.5
#define   PWM3_2        0x20                            //P:P6.4   N:P6.5

#define   PWM4_0        0x00                            //P:P1.6   N:P1.7
#define   PWM4_1        0x40                            //P:P2.6   N:P2.7
#define   PWM4_2        0x80                            //P:P6.6   N:P6.7
#define   PWM4_3        0xC0                            //P:P3.4   N:P3.3

#define   ENO1P         0x01
#define   ENO1N         0x02
#define   ENO2P         0x04
#define   ENO2N         0x08
#define   ENO3P         0x10
#define   ENO3N         0x20
#define   ENO4P         0x40
#define   ENO4N         0x80

/*****************************************************************************/

/*************本地变量声明**************/
bit B_1ms;                                              //1ms 标志
bit PWM1_Flag;

u16 Period;
u8 msSecond;

void UpdatePwm(void);
void TxPulse(u8 rep);

/********************  主函数  **************************/
void main(void)
{
    EAXFR = 1;                                          //允许访问扩展的特殊寄存器，XFR
                                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                      //允许访问扩展的特殊寄存器，XFR
                                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                           //设置取程序代码等待时间，
```

```
                                                    //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                      //设置访问片内的 xdata 速度，
                                                    //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M1 = 0x00;    P0M0 = 0x00;                    //设置为准双向口
    P1M1 = 0x00;    P1M0 = 0x00;                    //设置为准双向口
    P2M1 = 0x00;    P2M0 = 0x00;                    //设置为准双向口
    P3M1 = 0x00;    P3M0 = 0x00;                    //设置为准双向口
    P4M1 = 0x00;    P4M0 = 0x00;                    //设置为准双向口
    P5M1 = 0x00;    P5M0 = 0x00;                    //设置为准双向口
    P6M1 = 0x00;    P6M0 = 0x00;                    //设置为准双向口
    P7M1 = 0x00;    P7M0 = 0x00;                    //设置为准双向口

    PWM1_Flag = 0;
    Period = 0x1000;

                                                    //Timer0 初始化
    AUXR = 0x80;                                    //Timer0 set as 1T, 16 bits timer auto-reload,
    TH0 = (u8)(Timer0_Reload / 256);
    TL0 = (u8)(Timer0_Reload % 256);
    ET0 = 1;                                        //Timer0 interrupt enable
    TR0 = 1;                                        //Tiner0 run

    PWMA_ENO = 0x00;
    PWMA_ENO |= ENO1P;                              //使能输出

    PWMA_CCER1 = 0x00;                              //写 CCMRx 前必须先清零 CCxE  关闭通道
    PWMA_CCMR1 = 0x68;                              //设置 PWM1 模式 1 输出
//  PWMA_CCER1 = 0x01;                              //使能 CC1E 通道，高电平有效
    PWMA_CCER1 = 0x03;                              //使能 CC1E 通道，低电平有效

    PWMA_PS = 0x00;                                 //高级 PWM 通道输出脚选择位
    PWMA_PS |= PWM1_2;                              //选择 PWM1_2 通道

    UpdatePwm();
    PWMA_BKR = 0x80;                                //使能主输出
//      PWMA_CR1 |= 0x89;                           //使能 ARR 预装载，单脉冲模式，开始计时

    P40 = 0;                                        //给 LED 供电
    EA = 1;                                         //打开总中断

    while (1)
    {
        if (B_1ms)
        {
            B_1ms = 0;
            msSecond++;
            if (msSecond >= 10)                     //10ms 启动一次 PWM 输出
            {
                msSecond = 0;
                TxPulse(10);                        //输出 10 个脉冲
            }
        }
    }
}

/************ 发送脉冲函数 **************/
void TxPulse(u8 rep)
{
```

```
    if (rep == 0) return;
    rep -= 1;

    PWMA_RCR = rep;                                 //重复计数寄存器，计数 rep 个脉冲后产生更新事件
    PWMA_CR1 |= 0x89;                               //使能 ARR 预装载，单脉冲模式，开始计时
}

/********************** Timer0 1ms 中断函数 ************************/
void timer0(void) interrupt 1
{
    B_1ms = 1;
    if (PWM1_Flag)
    {
        Period++;                                   //周期递增
        if (Period >= 0x1000) PWM1_Flag = 0;
    }
    else
    {
        Period--;                                   //周期递减
        if (Period <= 0x0100) PWM1_Flag = 1;
    }
    UpdatePwm();                                    //设置周期、占空比
}


//========================================================================
// 函数: UpdatePwm(void)
// 描述: 更新 PWM 周期占空比.
// 参数: none.
// 返回: none.
// 版本: V1.0
//========================================================================
void UpdatePwm(void)
{
    PWMA_ARRH = (u8)(Period >> 8);                  //设置周期时间
    PWMA_ARRL = (u8)Period;
    PWMA_CCR1H = (u8)((Period >> 1) >> 8);          //设置占空比时间: Period/2
    PWMA_CCR1L = (u8)((Period >> 1));
}
```

29.10.7　高级 PWM 端口做外部中断（仅下降沿中断或者仅上升沿中断）

特别注意：单个 PWM 口不能实现同时检测上升沿和下降沿信号，如果需要同时检测上升沿和下降沿，可将外部信号同时连接到两路 PWM，使用其中一路检测信号的上升沿，另外一路 PWM 检测信号的下降沿

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                //头文件见下载软件
#include "intrins.h"                                //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void main(void)
{
    EAXFR = 1;                                      //允许访问扩展的特殊寄存器，XFR
                                                    //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                  //允许访问扩展的特殊寄存器，XFR
```

```
                                                    //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                       //设置取程序代码等待时间，
                                                    //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                      //设置访问片内的 xdata 速度，
                                                    //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P1M1 = 0x00;
    P1M0 = 0x00;
    P3M1 = 0x00;
    P3M0 = 0x00;

    PWMA_CCER1 = 0x00;
    PWMA_CCMR1 = 0x01;                              //CC1 为输入模式,且映射到 TI1FP1 上
    PWMA_CCER1 = 0x01;                              //使能 CC1 上的捕获功能
    PWMA_CCER1 |= 0x00;                             //设置捕获极性为 CC1 的上升沿
//  PWMA_CCER1 |= 0x02;                             //设置捕获极性为 CC1 的下降沿
    PWMA_CR1 = 0x01;
    PWMA_IER = 0x02;
    EA = 1;

    while (1);
}

void PWMA_ISR() interrupt 26
{
    if(PWMA_SR1 & 0X02)
    {
        P37 = ~P37;
        PWMA_SR1 &=~0X02;
    }
}
```

29.10.8 高级 PWM 一个端口做外部中断(同时捕获下降沿和上升沿)

原理：使用高级 PWM 内部的两通道的捕获模块 CCx 和 CCx+1 同时捕获外部的同一个管脚，CCx 捕获此管脚的上升沿，CCx+1 捕获此管脚的下降沿，即可同时捕获此管脚的上升沿中断和下降沿中断

范例：使用 PWMA 的第一组捕获模块 CC1 和第二组捕获模块 CC2，同时捕获 PWM1P 管脚（P1.0），其中 CC1 捕获 PWM1P 的上升沿，CC2 捕获 PWM1P 的下降沿。

注意：1、使用的是芯片内部的两路捕获模块同时捕获外部的同一个管脚，所以不需要将外部信号连接到多个 PWM 捕获管脚。

2、只有 CC1+CC2、CC3+CC4、CC5+CC6、CC7+CC8 这 4 种组合才能完成上面的功能。CC1+CC2 组合可以同时仅捕获 PWM1P 管脚，也可以同时仅捕获 PWM2P 管脚；CC3+CC4 组合可以同时仅捕获 PWM3P 管脚，也可以同时仅捕获 PWM4P 管脚；CC5+CC6 组合可以同时仅捕获 PWM5 管脚，也可以同时仅捕获 PWM6 管脚；CC7+CC8 组合可以同时仅捕获 PWM7 管脚，也可以同时仅捕获 PWM8 管脚

C 语言代码

```
//测试工作频率为 11.0592MHz
#include “Ai8051U.H”                                //头文件见下载软件
#include "intrins.h"                                //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void main(void)
```

```
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

                                                //(CC1 捕获 TI1 上升沿,CC2 捕获 TI1 下降沿)
    PWMA_CCER1 = 0x00;
    PWMA_CCMR1 = 0x01;                          //CC1 为输入模式,且映射到 TI1FP1(PWM1P)上
    PWMA_CCMR2 = 0x02;                          //CC2 为输入模式,且映射到 TI1FP2(PWM1P)上
    PWMA_CCER1 = 0x11;                          //使能 CC1/CC2 上的捕获功能
    PWMA_CCER1 |= 0x00;                         //设置捕获极性为 CC1 的上升沿
    PWMA_CCER1 |= 0x20;                         //设置捕获极性为 CC2 的下降沿
    PWMA_CR1 = 0x01;

    PWMA_IER = 0x06;                            //使能 CC1 和 CC2 捕获中断
    EA = 1;

    while (1);
}

void PWMA_ISR() interrupt 26
{
    unsigned int cnt;

    if (PWMA_SR1 & 0x02)
    {
        PWMA_SR1 &= ~0x02;                      //此时为 PWM1P（P1.0）口的上升沿中断
    }
    if (PWMA_SR1 & 0x04)
    {
        PWMA_SR1 &= ~0x04;                      //此时为 PWM1P（P1.0）口的下降沿中断
    }
}
```

29.10.9　输入捕获模式测量脉冲周期(捕获上升沿到上升沿或者下降沿到下降沿)

原理：使用高级 PWM 内部的某一通道的捕获模块 CCx，捕获外部的端口的上升沿或者下降沿，两个上升沿之间或者两个下降沿之间的时间即为脉冲的周期，也就是说，两次捕获计数值的差值即为周期值。

范例：使用 PWMA 的第一组捕获模块 CC1 捕获功能，捕获 PWM1P（P1.0）管脚上的上升沿，在中断中对前后两次的捕获值相减得到周期

注意：只有 PWM1P、PWM2P、PWM3P、PWM4P、PWM5、PWM6、PWM7、PWM8 这些管脚以及相应切换管脚才有捕获功能

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件
#include "intrins.h"                          //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

int         cap;
int         cap_new;
int         cap_old;

void main(void)
{
    EAXFR = 1;                                //允许访问扩展的特殊寄存器，XFR
                                              //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                            //允许访问扩展的特殊寄存器，XFR
                                              //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                 //设置取程序代码等待时间，
                                              //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                //设置访问片内的 xdata 速度，
                                              //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M1 = 0x00;
    P0M0 = 0x00;
    P1M1 = 0x00;
    P1M0 = 0x00;

/*配置成 TRGI 的 pin 需关掉 ENO 对应 bit 并配成 input*/
    PWMA_ENO = 0x00;                          //IO 输出 PWM
    PWMA_PS = 0x00;                           //00:PWM at P1

    PWMA_CCMR1 = 0x01;                        //配置成输入通道
    PWMA_SMCR = 0x56;
    PWMA_CCER1 = 0x01;                        //配置通道输出使能和极性

    PWMA_IER = 0x02;                          //使能中断
    PWMA_CR1 |= 0x01;                         //使能计数器

    EA = 1;
    while (1);
}

/*通道 1 输入，捕获数据通过 PWMA_CCR1H / PWMA_CCR1L 读取 */
void PWMA_ISR() interrupt 26
{
    if(PWMA_SR1 & 0x02)
    {
        cap_old = cap_new;
        cap_new = PWMA_CCR1H;                 //读取 CCR1H
        cap_new = (cap_new << 8) + PWMA_CCR1L;//读取 CCR1L
        cap = cap_new - cap_old;
        PWMA_SR1 &= ~0x02;
    }
}
```

29.10.10 输入捕获模式测量脉冲高电平宽度（捕获上升沿到下降沿）

原理：使用高级 PWM 内部的两通道的捕获模块 CCx 和 CCx+1 同时捕获外部的同一个管脚，CCx 捕获此管脚的上升沿，CCx+1 捕获此管脚的下降沿，然利用 CCx+1 的捕获值减去 CCx 的捕获值，其差值即为脉冲高电平的宽度。

范例：使用 PWMA 的第一组捕获模块 CC1 和第二组捕获模块 CC2，同时捕获 PWM1P 管脚（P1.0），其中 CC1 捕获 PWM1P 的上升沿，CC2 捕获 PWM1P 的下降沿，在中断中使用 CC2 的捕获值减去 CC1 的捕获值，其差值即为脉冲高电平的宽度。

注意：1、使用的是芯片内部的两路捕获模块同时捕获外部的同一个管脚，所以不需要将外部的多个管脚相连接。

2、只有 CC1+CC2、CC3+CC4、CC5+CC6、CC7+CC8 这 4 种组合才能完成上面的功能。CC1+CC2 组合可以同时捕获 PWM1P 管脚，也可以同时捕获 PWM2P 管脚；CC3+CC4 组合可以同时捕获 PWM3P 管脚，也可以同时捕获 PWM4P 管脚；CC5+CC6 组合可以同时捕获 PWM5 管脚，也可以同时捕获 PWM6 管脚；CC7+CC8 组合可以同时捕获 PWM7 管脚，也可以同时捕获 PWM8 管脚

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件
#include "intrins.h"                          //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void main()
{
    EAXFR = 1;                                //允许访问扩展的特殊寄存器，XFR
                                              //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                            //允许访问扩展的特殊寄存器，XFR
                                              //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                 //设置取程序代码等待时间，
                                              //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                //设置访问片内的 xdata 速度，
                                              //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;
                                              //(CC1 捕获 TI1 上升沿,CC2 捕获 TI1 下降沿)
    PWMA_CCER1 = 0x00;
    PWMA_CCMR1 = 0x01;                        //CC1 为输入模式,且映射到 TI1FP1 上
    PWMA_CCMR2 = 0x02;                        //CC2 为输入模式,且映射到 TI1FP2 上
    PWMA_CCER1 = 0x11;                        //使能 CC1/CC2 上的捕获功能
    PWMA_CCER1 |= 0x00;                       //设置捕获极性为 CC1 的上升沿
    PWMA_CCER1 |= 0x20;                       //设置捕获极性为 CC2 的下降沿
    PWMA_CR1 = 0x01;

    PWMA_IER = 0x04;                          //使能 CC2 捕获中断
    EA = 1;

    while (1);
}

void PWMA_ISR() interrupt 26
{
```

```
    unsigned int cnt;
    unsigned int cnt1;
    unsigned int cnt2;

    if (PWMA_SR1 & 0x04)
    {
        PWMA_SR1 &= ~0x04;

        cnt1 = (PWMA_CCR1H << 8) + PWMA_CCR1L;
        cnt2 = (PWMA_CCR2H << 8) + PWMA_CCR2L;
        cnt = cnt2 - cnt1;                                  //差值即为高电平宽度
    }
}
```

29.10.11 输入捕获模式测量脉冲低电平宽度（捕获下降沿到上升沿）

原理：使用高级 PWM 内部的两通道的捕获模块 CCx 和 CCx+1 同时捕获外部的同一个管脚，CCx 捕获此管脚的下降沿，CCx+1 捕获此管脚的上升沿，然利用 CCx+1 的捕获值减去 CCx 的捕获值，其差值即为脉冲低电平的宽度。

范例：使用 PWMA 的第一组捕获模块 CC1 和第二组捕获模块 CC2，同时捕获 PWM1P 管脚（P1.0），其中 CC1 捕获 PWM1P 的下降沿，CC2 捕获 PWM1P 的上升沿，在中断中使用 CC2 的捕获值减去 CC1 的捕获值，其差值即为脉冲低电平的宽度。

注意：1、使用的是芯片内部的两路捕获模块同时捕获外部的同一个管脚，所以不需要将外部的多个管脚相连接。

2、只有 CC1+CC2、CC3+CC4、CC5+CC6、CC7+CC8 这 4 种组合才能完成上面的功能。CC1+CC2 组合可以同时捕获 PWM1P 管脚，也可以同时捕获 PWM2P 管脚；CC3+CC4 组合可以同时捕获 PWM3P 管脚，也可以同时捕获 PWM4P 管脚；CC5+CC6 组合可以同时捕获 PWM5 管脚，也可以同时捕获 PWM6 管脚；CC7+CC8 组合可以同时捕获 PWM7 管脚，也可以同时捕获 PWM8 管脚

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件
#include "intrins.h"                          //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void main()
{
    EAXFR = 1;                                //允许访问扩展的特殊寄存器，XFR
                                              //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                            //允许访问扩展的特殊寄存器，XFR
                                              //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                 //设置取程序代码等待时间，
                                              //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                //设置访问片内的 xdata 速度，
                                              //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;
                                              //(CC1 捕获 TI1 上升沿,CC2 捕获 TI1 下降沿)
    PWMA_CCER1 = 0x00;
    PWMA_CCMR1 = 0x01;                        //CC1 为输入模式,且映射到 TI1FP1 上
    PWMA_CCMR2 = 0x02;                        //CC2 为输入模式,且映射到 TI1FP2 上
    PWMA_CCER1 = 0x11;                        //使能 CC1/CC2 上的捕获功能
    PWMA_CCER1 |= 0x00;                       //设置捕获极性为 CC1 的上升沿
    PWMA_CCER1 |= 0x20;                       //设置捕获极性为 CC2 的下降沿
    PWMA_CR1 = 0x01;

    PWMA_IER = 0x02;                          //使能 CC1 捕获中断
    EA = 1;

    while (1);
}

void PWMA_ISR() interrupt 26
{
```

```
    unsigned int cnt;
    unsigned int cnt1;
    unsigned int cnt2;

    if (PWMA_SR1 & 0x02)
    {
        PWMA_SR1 &= ~0x02;

        cnt1 = (PWMA_CCR1H << 8) + PWMA_CCR1L;
        cnt2 = (PWMA_CCR2H << 8) + PWMA_CCR2L;
        cnt = cnt1 - cnt2;                              //差值即为低电平宽度
    }
}
```

29.10.12 输入捕获模式同时测量脉冲周期和高电平宽度（占空比）

原理：使用高级 PWM 内部的两通道的捕获模块 CCx 和 CCx+1 同时捕获外部的同一个管脚，CCx 捕获此管脚的上升沿，CCx+1 捕获此管脚的下降沿，同时使能此管脚的上升沿信号为复位触发信号，CCx 的捕获值即为周期，CCx+1 的捕获值即为占空比。

范例：使用 PWMA 的第一组捕获模块 CC1 和第二组捕获模块 CC2，同时捕获 PWM1P 管脚（P1.0），其中 CC1 捕获 PWM1P 的上升沿，CC2 捕获 PWM1P 的下降沿，并设置 PWM1P 的上升沿信号为复位触发信号，CC1 的捕获值即为周期，CC2 的捕获值即为占空比。

注意：1、使用的是芯片内部的两路捕获模块同时捕获外部的同一个管脚，所以不需要将外部的多个管脚相连接。

2、只有 CC1+CC2、CC5+CC6 这两种组合才能完成上面的功能。CC1+CC2 组合可以同时捕获 PWM1P 管脚，也可以同时捕获 PWM2P 管脚； CC5+CC6 组合可以同时捕获 PWM5 管脚，也可以同时捕获 PWM6 管脚

3、由于设置了复位触发信号，所以捕获值即为周期值和占空比值，无需再减前一次的捕获值。

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件
#include "intrins.h"                          //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void main()
{
    EAXFR = 1;                                //允许访问扩展的特殊寄存器，XFR
                                              //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                            //允许访问扩展的特殊寄存器，XFR
                                              //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                 //设置取程序代码等待时间，
                                              //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                //设置访问片内的 xdata 速度，
                                              //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

                                              //(CC1 捕获 TI1 上升沿,CC2 捕获 TI1 下降沿)
                                              //CC1 捕获周期宽度,CC2 捕获高电平宽度
    PWMA_CCER1 = 0x00;
    PWMA_CCMR1 = 0x01;                        //CC1 为输入模式,且映射到 TI1FP1 上
    PWMA_CCMR2 = 0x02;                        //CC2 为输入模式,且映射到 TI1FP2 上
    PWMA_CCER1 = 0x11;                        //使能 CC1/CC2 上的捕获功能
    PWMA_CCER1 |= 0x00;                       //设置捕获极性为 CC1 的上升沿
    PWMA_CCER1 |= 0x20;                       //设置捕获极性为 CC2 的下降沿
    PWMA_SMCR = 0x54;                         //TS=TI1FP1,SMS=TI1 上升沿复位模式
    PWMA_CR1 = 0x01;

    PWMA_IER = 0x06;                          //使能 CC1/CC2 捕获中断
    EA = 1;

    while (1);
}
```

```
void PWMA_ISR() interrupt 26
{
    unsigned int cnt;

    if (PWMA_SR1 & 0x02)
    {
        PWMA_SR1 &= ~0x02;

        cnt = (PWMA_CCR1H << 8) + PWMA_CCR1L;        //CC1 捕获周期宽度
    }
    if (PWMA_SR1 & 0x04)
    {
        PWMA_SR1 &= ~0x04;

        cnt = (PWMA_CCR2H << 8) + PWMA_CCR2L;        //CC2 捕获占空比（高电平宽度）
    }
}
```

29.10.13 同时捕获 4 路输入信号的周期和高电平宽度（占空比）

原理：使用高级 PWM 内部的两通道的捕获模块 CCx 和 CCx+1 同时捕获外部的同一个管脚，CCx 捕获此管脚的上升沿，CCx+1 捕获此管脚的下降沿，CCx 的两次捕获值的差值即为周期，CCx+1 的捕获值与 CCx 的前一次捕获值的差值即为占空比。

范例：使用 PWMA 的第一组捕获模块 CC1 和第二组捕获模块 CC2，同时捕获 PWM1P 管脚（P1.0），其中 CC1 捕获 PWM1P 的上升沿（CC1/CC2 均既可以捕获 PWM1P 的上升沿，也可捕获 PWM1P 的下降沿，本例仅演示 CC1 捕获 PWM1P 的上升沿，CC2 捕获 PWM1P 的下降沿。CC3/CC4，CC5/CC6，CC7/CC8 同理），CC2 捕获 PWM1P 的下降沿，CC1 的捕获值减去前一次捕获值即为周期，CC2 的捕获值减去 CC1 的前一次捕获值即为占空比。PWMA 的 CC3 和 CC4 同时捕获 PWM3P（P1.4）、PWMB 的 CC5 和 CC6 同时捕获 PWM5（P2.0）、PWMB 的 CC7 和 CC8 同时捕获 PWM7（P2.2）。另外本例演示使用定时器 0 在 P1.0 上产生波形、定时器 1 在 P1.4 上产生波形、定时器 3 在 P2.0 上产生波形、定时器 4 在 P2.2 上产生波形。捕获值通过串口送到 PC。

注意：1、使用的是芯片内部的两路捕获模块同时捕获外部的同一个管脚，所以不需要将外部的多个管脚相连接。

2、由于没有设置复位触发信号，所以周期值和占空比值均需要作相应的减法运算才能得到。

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                  //头文件见下载软件
#include "intrins.h"                                  //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。
#include "stdio.h"

#define   FOSC        12000000UL
#define   BRT         (65536 - (FOSC / 115200+2) / 4)
                                                      //加 2 操作是为了让 Keil 编译器
                                                      //自动实现四舍五入运算
#define   T10K        (65536 - FOSC / 10000)
#define   T11K        (65536 - FOSC / 11000)
#define   T12K        (65536 - FOSC / 12000)
#define   T13K        (65536 - FOSC / 13000)

unsigned int ccr1;
unsigned int ccr3;
unsigned int ccr5;
unsigned int ccr7;

unsigned int cycle1;
unsigned int duty1;
unsigned int cycle2;
unsigned int duty2;
unsigned int cycle3;
unsigned int duty3;
unsigned int cycle4;
unsigned int duty4;

bit f1, f2, f3, f4;

void main()
{

    EAXFR = 1;                                        //允许访问扩展的特殊寄存器，XFR
                                                      //(32 位模式请使用这句，注释下一句)
```

```
//      P_SW2 |= 0x80;                    //允许访问扩展的特殊寄存器，XFR
                                          //(8 位模式请使用这句，注释上一句)
        WTST = 0;                         //设置取程序代码等待时间，
                                          //赋值为 0 表示不等待，程序以最快速度运行
        CKCON = 0;                        //设置访问片内的 xdata 速度，
                                          //赋值为 0 表示用最快速度访问，不增加额外的等待时间

        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

        AUXR |= 0x80;                     //定时器 0 使用 1T 模式
        AUXR |= 0x40;                     //定时器 1 使用 1T 模式
        TMOD = 0x00;                      //定时器 0/1 使用 16 位自动重载模式
        TL0 = T10K;                       //定时器 0 周期 10K
        TH0 = T10K >> 8;
        TL1 = T11K;                       //定时器 1 周期 11K
        TH1 = T11K >> 8;
        TR0 = 1;                          //定时器 0 开始计数
        TR1 = 1;                          //定时器 1 开始计数
        ET0 = 1;                          //使能定时器 0 中断
        ET1 = 1;                          //使能定时器 1 中断

        T3L = T12K;                       //定时器 3 周期 12K
        T3H = T12K >> 8;
        T4L = T13K;                       //定时器 4 周期 13K
        T4H = T13K >> 8;
        T4T3M = 0xaa;                     //定时器 3/4 使用 1T 模式
        IE2 |= 0x20;                      //使能定时器 3 中断
        IE2 |= 0x40;                      //使能定时器 4 中断

        SCON = 0x52;
        T2L = BRT;
        T2H = BRT >> 8;
        AUXR |= 0x15;

        printf("PWM Test .\n");

        PWMA_CCER1 = 0x00;
        PWMA_CCMR1 = 0x01;                //CC1 为输入模式,且映射到 TI1FP1 上
        PWMA_CCMR2 = 0x02;                //CC2 为输入模式,且映射到 TI1FP2 上
        PWMA_CCER1 = 0x11;                //使能 CC1 上的捕获功能,使能 CC2 上的捕获功能
        PWMA_CCER1 |= 0x00;               //设置捕获极性为 CC1 的上升沿
        PWMA_CCER1 |= 0x20;               //设置捕获极性为 CC2 的下降沿

        PWMA_CCER2 = 0x00;
        PWMA_CCMR3 = 0x01;                //CC3 为输入模式,且映射到 TI3FP3 上
        PWMA_CCMR4 = 0x02;                //CC4 为输入模式,且映射到 TI3FP4 上
        PWMA_CCER2 = 0x11;                //使能 CC3 上的捕获功能,使能 CC4 上的捕获功能
        PWMA_CCER2 |= 0x00;               //设置捕获极性为 CC3 的上升沿
        PWMA_CCER2 |= 0x20;               //设置捕获极性为 CC4 的下降沿
        PWMA_CR1 = 0x01;
```

```
    PWMA_IER = 0x1e;                            //使能 CC1/CC2/CC3/CC4 捕获中断

    PWMB_CCER1 = 0x00;
    PWMB_CCMR1 = 0x01;                          //CC5 为输入模式,且映射到 TI5FP5 上
    PWMB_CCMR2 = 0x02;                          //CC6 为输入模式,且映射到 TI5FP6 上
    PWMB_CCER1 = 0x11;                          //使能 CC5 上的捕获功能,使能 CC6 上的捕获功能
    PWMB_CCER1 |= 0x00;                         //设置捕获极性为 CC5 的上升沿
    PWMB_CCER1 |= 0x20;                         //设置捕获极性为 CC6 的下降沿

    PWMB_CCER2 = 0x00;
    PWMB_CCMR3 = 0x01;                          //CC7 为输入模式,且映射到 TI7FP8 上
    PWMB_CCMR4 = 0x02;                          //CC8 为输入模式,且映射到 TI7FP8 上
    PWMB_CCER2 = 0x11;                          //使能 CC7 上的捕获功能,使能 CC8 上的捕获功能
    PWMB_CCER2 |= 0x00;                         //设置捕获极性为 CC7 的上升沿
    PWMB_CCER2 |= 0x20;                         //设置捕获极性为 CC8 的下降沿
    PWMB_CR1 = 0x01;

    PWMB_IER = 0x1e;                            //使能 CC5/CC6/CC7/CC8 捕获中断

    EA = 1;

    while (1)
    {
        if (f1)
        {
            f1 = 0;
            printf("cycle1 = %04x duty1 = %04x\n", cycle1, duty1);
        }
        if (f2)
        {
            f2 = 0;
            printf("cycle2 = %04x duty2 = %04x\n", cycle2, duty2);
        }
        if (f3)
        {
            f3 = 0;
            printf("cycle3 = %04x duty3 = %04x\n", cycle3, duty3);
        }
        if (f4)
        {
            f4 = 0;
            printf("cycle4 = %04x duty4 = %04x\n", cycle4, duty4);
        }
    }
}

void TMR0_ISR() interrupt TMR0_VECTOR           //产生 CC1 波形到 P1.0 口
{
    static unsigned int cnt = 0;

    cnt++;
    if (cnt == 10)
    {
        P10 = 0;
    }
    else if (cnt == 30)
    {
        P10 = 1;
```

```
            cnt = 0;
        }
}

void TMR1_ISR() interrupt TMR1_VECTOR                   //产生 CC3 波形到 P1.4 口
{
        static unsigned int cnt = 0;

        cnt++;
        if (cnt == 10)
        {
            P14 = 0;
        }
        else if (cnt == 30)
        {
            P14 = 1;
            cnt = 0;
        }
}

void TMR3_ISR() interrupt TMR3_VECTOR                   //产生 CC5 波形到 P2.0 口
{
        static unsigned int cnt = 0;

        cnt++;
        if (cnt == 10)
        {
            P20 = 0;
        }
        else if (cnt == 30)
        {
            P20 = 1;
            cnt = 0;
        }
}

void TMR4_ISR() interrupt TMR4_VECTOR                   //产生 CC7 波形到 P2.2 口
{
        static unsigned int cnt = 0;

        cnt++;
        if (cnt == 10)
        {
            P22 = 0;
        }
        else if (cnt == 30)
        {
            P22 = 1;
            cnt = 0;
        }
}

void PWMA_ISR() interrupt PWMA_VECTOR
{
        unsigned int ccr;

        if (PWMA_SR1 & 0x02)                            //CC1 捕获中断
        {
            PWMA_SR1 &= ~0x02;
```

```
            ccr = (PWMA_CCR1H << 8) + PWMA_CCR1L;       //读取捕获值
            cycle1 = ccr - ccr1;                        //计算周期
            ccr1 = ccr;                                 //保存当前捕获值
            f1 = 1;                                     //波形 1 的周期和占空比捕获完成，触发串口发送
        }
        if (PWMA_SR1 & 0x04)                            //CC2 捕获中断
        {
            PWMA_SR1 &= ~0x04;

            ccr = (PWMA_CCR2H << 8) + PWMA_CCR2L;       //读取捕获值
            duty1 = ccr - ccr1;                         //计算占空比
        }

        if (PWMA_SR1 & 0x08)                            //CC3 捕获中断
        {
            PWMA_SR1 &= ~0x08;

            ccr = (PWMA_CCR3H << 8) + PWMA_CCR3L;       //读取捕获值
            cycle2 = ccr - ccr3;                        //计算周期
            ccr3 = ccr;                                 //保存当前捕获值
            f2 = 1;                                     //波形 2 的周期和占空比捕获完成，触发串口发送
        }
        if (PWMA_SR1 & 0x10)                            //CC4 捕获中断
        {
            PWMA_SR1 &= ~0x10;

            ccr = (PWMA_CCR4H << 8) + PWMA_CCR4L;       //读取捕获值
            duty2 = ccr - ccr3;                         //计算占空比
        }
}

void PWMB_ISR() interrupt PWMB_VECTOR
{
        unsigned int ccr;

        if (PWMB_SR1 & 0x02)                            //CC5 捕获中断
        {
            PWMB_SR1 &= ~0x02;

            ccr = (PWMB_CCR5H << 8) + PWMB_CCR5L;       //读取捕获值
            cycle3 = ccr - ccr5;                        //计算周期
            ccr5 = ccr;                                 //保存当前捕获值
            f3 = 1;                                     //波形 3 的周期和占空比捕获完成，触发串口发送
        }
        if (PWMB_SR1 & 0x04)                            //CC6 捕获中断
        {
            PWMB_SR1 &= ~0x04;

            ccr = (PWMB_CCR6H << 8) + PWMB_CCR6L;       //读取捕获值
            duty3 = ccr - ccr5;                         //计算占空比
        }

        if (PWMB_SR1 & 0x08)                            //CC7 捕获中断
        {
            PWMB_SR1 &= ~0x08;

            ccr = (PWMB_CCR7H << 8) + PWMB_CCR7L;       //读取捕获值
            cycle4 = ccr - ccr7;                        //计算周期
```

```
            ccr7 = ccr;                                 //保存当前捕获值
            f4 = 1;                                     //波形 4 的周期和占空比捕获完成，触发串口发送
        }
        if (PWMB_SR1 & 0x10)                            //CC8 捕获中断
        {
            PWMB_SR1 &= ~0x10;

            ccr = (PWMB_CCR8H << 8) + PWMB_CCR8L;       //读取捕获值
            duty4 = ccr - ccr7;                         //计算占空比
        }
}
```

29.10.14 带死区控制的 PWM 互补输出

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void main(void)
{
    EAXFR = 1;                                          //允许访问扩展的特殊寄存器，XFR
                                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                      //允许访问扩展的特殊寄存器，XFR
                                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                           //设置取程序代码等待时间，
                                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                          //设置访问片内的 xdata 速度，
                                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M1 = 0x00;
    P0M0 = 0xFF;
    P1M1 = 0x00;
    P1M0 = 0xFF;

    PWMA_ENO = 0xFF;                                    //IO 输出 PWM
    PWMA_PS = 0x00;                                     //00:PWM at P1

/*********************************************************
PWMx_duty = [CCRx/(ARR + 1)]*100
*********************************************************/
    PWMA_PSCRH = 0x00;                                  //预分频寄存器
    PWMA_PSCRL = 0x00;
    PWMA_DTR = 0x10;                                    //死区时间配置

    PWMA_CCMR1 = 0x68;                                  //通道模式配置
    PWMA_CCMR2 = 0x68;
    PWMA_CCMR3 = 0x68;
    PWMA_CCMR4 = 0x68;

    PWMA_ARRH = 0x08;                                   //自动重装载寄存器，计数器 overflow 点
    PWMA_ARRL = 0x00;

    PWMA_CCR1H = 0x04;                                  //计数器比较值
    PWMA_CCR1L = 0x00;
```

```
    PWMA_CCR2H = 0x02;
    PWMA_CCR2L = 0x00;
    PWMA_CCR3H = 0x01;
    PWMA_CCR3L = 0x00;
    PWMA_CCR4H = 0x01;
    PWMA_CCR4L = 0x00;

    PWMA_CCER1 = 0x55;                          //配置通道输出使能和极性
    PWMA_CCER2 = 0x55;                          //配置通道输出使能和极性

    PWMA_BKR = 0x80;                            //主输出使能 相当于总开关
    PWMA_IER = 0x02;                            //使能中断
    PWMA_CR1 = 0x01;                            //使能计数器

    EA = 1;
    while (1);
}

void PWMA_ISR() interrupt 26
{
    if(PWMA_SR1 & 0X02)
    {
        P03 = ~P03;
        PWMA_SR1 &=~0X02;
    }
}
```

29.10.15 利用 PWM 实现互补 SPWM

高级 PWM 定时器 PWM1P/PWM1N，PWM2P/PWM2N，PWM3P/PWM3N，PWM4P/PWM4N 每个通道都可独立实现 PWM 输出，或者两两互补对称输出。演示使用 PWM1P，PWM1N 产生互补的 SPWM。主时钟选择 24MHZ，PWM 时钟选择 1T，PWM 周期 2400，死区 12 个时钟(0.5us)，正弦波表用 200 点，输出正弦波频率 = 24000000 / 2400 / 200 = 50 HZ。

本程序仅仅是一个 SPWM 的演示程序，用户可以通过上面的计算方法修改 PWM 周期和正弦波的点数和幅度。本程序输出频率固定，如果需要变频，请用户自己设计变频方案。

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                           //头文件见下载软件
#include "intrins.h"                            //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define    MAIN_Fosc        24000000L           //定义主时钟

typedef    unsigned char    u8;
typedef    unsigned int     u16;
typedef    unsigned long    u32;

/*****************************  用户定义宏  ***********************************/

#define    PWMA_1           0x00                //P:P1.0   N:P1.1
#define    PWMA_2           0x01                //P:P2.0   N:P2.1
#define    PWMA_3           0x02                //P:P6.0   N:P6.1
```

```
#define   PWMB_1          0x00                  //P:P1.2   N:P1.3
#define   PWMB_2          0x04                  //P:P2.2   N:P2.3
#define   PWMB_3          0x08                  //P:P6.2   N:P6.3

#define   PWM3_1          0x00                  //P:P1.4   N:P1.5
#define   PWM3_2          0x10                  //P:P2.4   N:P2.5
#define   PWM3_3          0x20                  //P:P6.4   N:P6.5

#define   PWM4_1          0x00                  //P:P1.6   N:P1.7
#define   PWM4_2          0x40                  //P:P2.6   N:P2.7
#define   PWM4_3          0x80                  //P:P6.6   N:P6.7
#define   PWM4_4          0xC0                  //P:P3.4   N:P3.3

#define   ENO1P           0x01
#define   ENO1N           0x02
#define   ENO2P           0x04
#define   ENO2N           0x08
#define   ENO3P           0x10
#define   ENO3N           0x20
#define   ENO4P           0x40
#define   ENO4N           0x80

/*************  本地变量声明  **************/

unsigned int code T_SinTable[]=
{
    1220, 1256, 1292, 1328, 1364, 1400, 1435, 1471,
    1506, 1541, 1575, 1610, 1643, 1677, 1710, 1742,
    1774, 1805, 1836, 1866, 1896, 1925, 1953, 1981,
    2007, 2033, 2058, 2083, 2106, 2129, 2150, 2171,
    2191, 2210, 2228, 2245, 2261, 2275, 2289, 2302,
    2314, 2324, 2334, 2342, 2350, 2356, 2361, 2365,
    2368, 2369, 2370, 2369, 2368, 2365, 2361, 2356,
    2350, 2342, 2334, 2324, 2314, 2302, 2289, 2275,
    2261, 2245, 2228, 2210, 2191, 2171, 2150, 2129,
    2106, 2083, 2058, 2033, 2007, 1981, 1953, 1925,
    1896, 1866, 1836, 1805, 1774, 1742, 1710, 1677,
    1643, 1610, 1575, 1541, 1506, 1471, 1435, 1400,
    1364, 1328, 1292, 1256, 1220, 1184, 1148, 1112,
    1076, 1040, 1005,  969,  934,  899,  865,  830,
     797,  763,  730,  698,  666,  635,  604,  574,
     544,  515,  487,  459,  433,  407,  382,  357,
     334,  311,  290,  269,  249,  230,  212,  195,
     179,  165,  151,  138,  126,  116,  106,   98,
      90,   84,   79,   75,   72,   71,   70,   71,
      72,   75,   79,   84,   90,   98,  106,  116,
     126,  138,  151,  165,  179,  195,  212,  230,
     249,  269,  290,  311,  334,  357,  382,  407,
     433,  459,  487,  515,  544,  574,  604,  635,
     666,  698,  730,  763,  797,  830,  865,  899,
     934,  969, 1005, 1040, 1076, 1112, 1148, 1184,
};

u16 PWMA_Duty;
u8 PWM_Index;                                   //SPWM 查表索引

/********************  主函数  **************************/
void main(void)
{
```

```
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                           //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M1 = 0;   P0M0 = 0;               //设置为准双向口
    P1M1 = 0;   P1M0 = 0;               //设置为准双向口
    P2M1 = 0;   P2M0 = 0;               //设置为准双向口
    P3M1 = 0;   P3M0 = 0;               //设置为准双向口
    P4M1 = 0;   P4M0 = 0;               //设置为准双向口
    P5M1 = 0;   P5M0 = 0;               //设置为准双向口
    P6M1 = 0;   P6M0 = 0;               //设置为准双向口
    P7M1 = 0;   P7M0 = 0;               //设置为准双向口

    PWMA_Duty = 1220;

    PWMA_CCER1 = 0x00;                  //写 CCMRx 前必须先清零 CCxE 关闭通道
    PWMA_CCER2 = 0x00;
    PWMA_CCMR1 = 0x60;                  //通道模式配置
//  PWMA_CCMR2 = 0x60;
//  PWMA_CCMR3 = 0x60;
//  PWMA_CCMR4 = 0x60;
    PWMA_CCER1 = 0x05;                  //配置通道输出使能和极性
//  PWMA_CCER2 = 0x55;

    PWMA_ARRH = 0x09;                       //设置周期时间
    PWMA_ARRL = 0x60;

    PWMA_CCR1H = (u8)(PWMA_Duty >> 8);  //设置占空比时间
    PWMA_CCR1L = (u8)(PWMA_Duty);

    PWMA_DTR = 0x0C;                    //设置死区时间

    PWMA_ENO = 0x00;
    PWMA_ENO |= ENO1P;                  //使能输出
    PWMA_ENO |= ENO1N;                  //使能输出
//  PWMA_ENO |= ENO2P;                  //使能输出
//  PWMA_ENO |= ENO2N;                  //使能输出
//  PWMA_ENO |= ENO3P;                  //使能输出
//  PWMA_ENO |= ENO3N;                  //使能输出
//  PWMA_ENO |= ENO4P;                  //使能输出
//  PWMA_ENO |= ENO4N;                  //使能输出

    PWMA_PS = 0x00;                     //高级 PWM 通道输出脚选择位
    PWMA_PS |= PWMA_3;                  //选择 PWMA_3 通道
//  PWMA_PS |= PWMB_3;                  //选择 PWMB_3 通道
//  PWMA_PS |= PWM3_3;                  //选择 PWM3_3 通道
//  PWMA_PS |= PWM4_3;                  //选择 PWM4_3 通道

    PWMA_BKR = 0x80;                    //使能主输出
    PWMA_IER = 0x01;                    //使能中断
    PWMA_CR1 |= 0x01;                   //开始计时

    EA = 1;                             //打开总中断
```

```
    while (1)
    {
    }
}

/********************  中断函数  ************************/
void PWMA_ISR() interrupt 26
{
    if (PWMA_SR1 & 0x01)
    {
        PWMA_SR1 &=~0x01;
        PWMA_Duty = T_SinTable[PWM_Index];
        if (++PWM_Index >= 200)
            PWM_Index = 0;

        PWMA_CCR1H = (u8)(PWMA_Duty >> 8);          //设置占空比时间
        PWMA_CCR1L = (u8)(PWMA_Duty);
    }
    PWMA_SR1 = 0;
}
```

29.10.16 产生 3 路相位差 120 度的互补 PWM 波形（网友提供）

```
//测试工作频率为 24MHz

/*************************************************
主要功能: P2.0-P2.5 输出互补的三路相位差 120 度的 PWM
第 1 路 P2.0/P2.1 为 PWM 输出模式，第 2 路 P2.2/P2.3 和第 3 路 P2.4/P2.5 为比较输出模式
程序下载进目标芯片,输出 50hz 的 SPWM,占空比 25%
*************************************************/

#include “Ai8051U.H”

#define   FOSC              24000000UL

#define   PWM_PSC           (240-1)             //定义 PWM 时钟预分频系数
#define   PWM_PERIOD        2000                //定义 PWM 周期值
                                                //(频率=FOSC/(PWM_PSC+1)/PWM_PERIOD=50Hz)
#define   PWM_DUTY          500                 //定义 PWM 的占空比值
                                                //(占空比=PWM_DUTY/PWM_PERIOD*100%=25%)

void SYS_Init();
void PWM_Init();

void main()
{
    SYS_Init();
    PWM_Init();

    EA = 1;                                     //打开总中断

    while (1);
}

void SYS_Init()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M1 = 0x00;   P0M0 = 0x00;
    P1M1 = 0x00;   P1M0 = 0x00;
    P2M1 = 0x00;   P2M0 = 0x00;
    P3M1 = 0x00;   P3M0 = 0x00;
    P4M1 = 0x00;   P4M0 = 0x00;
    P5M1 = 0x00;   P5M0 = 0x00;
    P6M1 = 0x00;   P6M0 = 0x00;
    P7M1 = 0x00;   P7M0 = 0x00;
}

void PWM_Init()
{
    PWMA_PSCRH = (char)(PWM_PSC >> 8);          //配置预分频系数
```

```
    PWMA_PSCRL = (char)(PWM_PSC);

    PWMA_CCER1 = 0x00;                          //写 CCMRx 前必须先清零 CCxE 关闭通道
    PWMA_CCER2 = 0x00;

    PWMA_CCMR1 = 0x60;                          //通道模式配置 PWM 模式 1
    PWMA_CCMR2 = 0x30;                          //通道模式配置输出比较模式
    PWMA_CCMR3 = 0x30;                          //通道模式配置输出比较模式

    PWMA_CCER1 = 0x55;                          //配置通道 1,2,3 输出使能和极性
    PWMA_CCER2 = 0x05;

    PWMA_ARRH = (char)(PWM_PERIOD >> 8);        //设置周期时间
    PWMA_ARRL = (char)(PWM_PERIOD);

    PWMA_ENO = 0x3f;                            //使能 PWM 输出
    PWMA_PS = 0x15;                             //高级 PWM 通道输出脚选择 P2.0-P2.5

    PWMA_CCR1H = (char)(PWM_DUTY >> 8);         //设置占空比时间
    PWMA_CCR1L = (char)(PWM_DUTY);
    PWMA_CCR2H = (char)(PWM_PERIOD/3 >> 8);     //设置 OC2 起始翻转位
    PWMA_CCR2L = (char)(PWM_PERIOD/3);
    PWMA_CCR3H = (char)(PWM_PERIOD/3*2 >> 8);   //设置 OC3 起始翻转位
    PWMA_CCR3L = (char)(PWM_PERIOD/3*2);

    PWMA_IER = 0x0d;                            //使能 OC2/OC3 比较中断,更新中断

    PWMA_BKR = 0x80;                            //使能主输出
    PWMA_CR1 |= 0x01;                           //开始计时
}

void PWMA_ISR()     interrupt 26
{
    if (PWMA_SR1 & 0x01)
    {
        PWMA_CCR2H = (char)(PWM_PERIOD/3 >> 8);     //设置占空比时间
        PWMA_CCR2L = (char)(PWM_PERIOD/3);
        PWMA_CCR3H = (char)(PWM_PERIOD/3*2 >> 8);   //设置占空比时间
        PWMA_CCR3L = (char)(PWM_PERIOD/3*2);
        PWMA_SR1 &= ~0x01;
    }
    else if (PWMA_SR1 &0x04)
    {
        PWMA_CCR2H = (char)((PWM_PERIOD/3+PWM_DUTY) >> 8);      //设置 0C2 结束翻转位
        PWMA_CCR2L = (char)(PWM_PERIOD/3+PWM_DUTY);
        PWMA_SR1 &= ~0x04;
    }
    else if (PWMA_SR1 &0x08)
    {
        PWMA_CCR3H = (char)((PWM_PERIOD/3*2+PWM_DUTY) >> 8);    //设置 0C3 结束翻转位
        PWMA_CCR3L = (char)(PWM_PERIOD/3*2+PWM_DUTY);
        PWMA_SR1 &= ~0x08;
    }
    else
    {
        PWMA_SR1 = 0;
    }
}
```

29.10.17 使用 PWM 的 CEN 启动 PWMA 定时器，实时触发 ADC

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                  //头文件见下载软件
#include "intrins.h"                                  //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void delay()
{
    int i;
    for (i=0; i<100; i++);
}

void main()
{
    EAXFR = 1;                                        //允许访问扩展的特殊寄存器，XFR
                                                      //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                    //允许访问扩展的特殊寄存器，XFR
                                                      //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                         //设置取程序代码等待时间，
                                                      //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                        //设置访问片内的 xdata 速度，
                                                      //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P1M0 = 0x00;
    P1M1 = 0x01;
    P3M0 = 0x00;
    P3M1 = 0x00;

    ADC_CONTR = 0;                                    //选择 P1.0 为 ADC 输入通道
    ADC_POWER = 1;
    ADC_EPWMT = 1;
    delay();                                          //等待 ADC 电源稳定
    EADC = 1;

    PWMA_CR2 = 0x10;                                  //CEN 信号为 TRGO,可用于触发 ADC
    PWMA_ARRH = 0x13;
    PWMA_ARRL = 0x38;
    PWMA_IER = 0x01;
    PWMA_CR1 = 0x01;                                  //设置 CEN 启动 PWMA 定时器，实时触发 ADC
    EA = 1;

    while (1);
}

void ADC_ISR() interrupt 5
{
    ADC_ FLAG = 0;
}

void PWMA_ISR() interrupt 26
{
    if(PWMA_SR1 & 0x01)
    {
        PWMA_SR1 &=~0x01;
```

```
    }
}
```

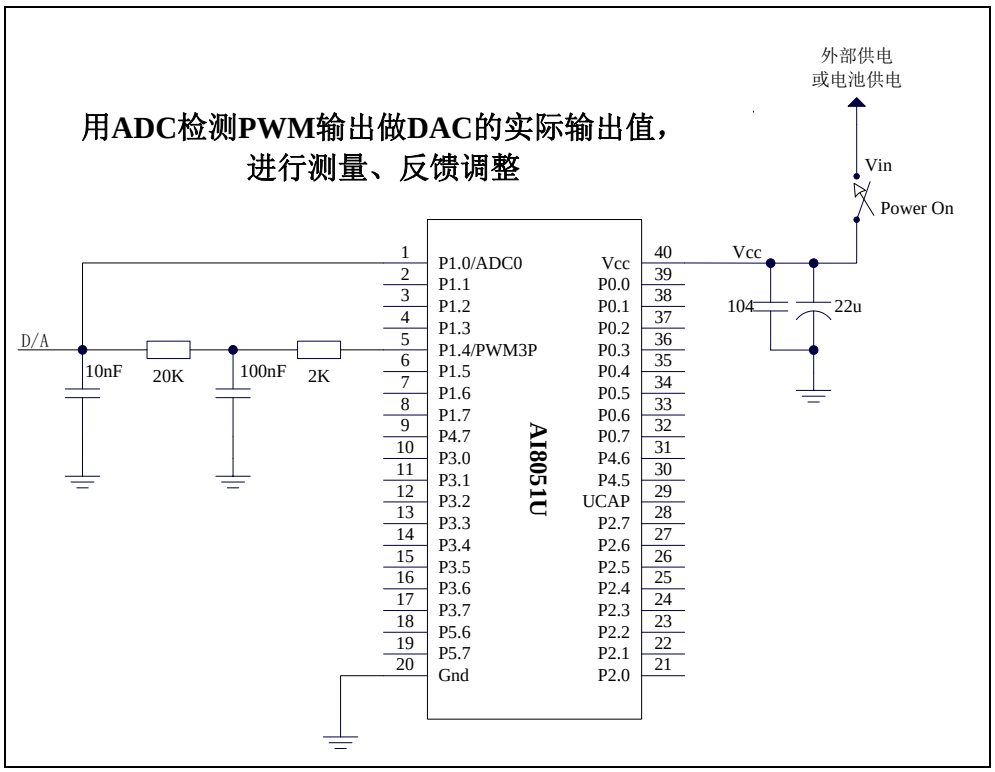
29.10.18 PWM 周期重复触发 ADC

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件
#include "intrins.h"                          //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

void delay()
{
    int i;
    for (i=0; i<100; i++);
}

void main()
{
    EAXFR = 1;                                //允许访问扩展的特殊寄存器，XFR
                                              //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                            //允许访问扩展的特殊寄存器，XFR
                                              //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                 //设置取程序代码等待时间，
                                              //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                //设置访问片内的 xdata 速度，
                                              //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P1M0 = 0x00;
    P1M1 = 0x01;
    P3M0 = 0x00;
    P3M1 = 0x00;

    ADC_CONTR = 0;                            //选择 P1.0 为 ADC 输入通道
    ADC_POWER = 1;
    ADC_EPWMT = 1;
    delay();                                  //等待 ADC 电源稳定
    EADC = 1;

    PWMA_CR2 = 0x20;                          //周期更新事件为 TRGO,用于周期触发 ADC
    PWMA_ARRH = 0x13;
    PWMA_ARRL = 0x38;
    PWMA_IER = 0x01;
    PWMA_CR1 = 0x01;                          //设置 CEN 启动 PWMA 定时器
    EA = 1;

    while (1);
}

void ADC_ISR() interrupt 5
{
    ADC_ FLAG = 0;
}

void PWMA_ISR() interrupt 26
{
```

```
    if(PWMA_SR1 & 0x01)
    {
        PWMA_SR1 &=~0x01;
    }
}
```

29.10.19 利用 PWM 实现 16 位 DAC 的参考线路图

Ai8051U 系列单片机的高级 PWM 定时器可输出 16 位的 PWM 波形，再经过两级低通滤波即可产生 16 位的 DAC 信号，通过调节 PWM 波形的高电平占空比即可实现 DAC 信号的改变。应用线路图如下图所示，输出的 DAC 信号可输入到 MCU 的 ADC 进行反馈测量。



29.10.20 正交编码器模式演示程序

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                          //头文件见下载软件
#include "intrins.h"                          //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

unsigned char cnt_H, cnt_L;

void main(void)
{
    EAXFR = 1;                                //允许访问扩展的特殊寄存器，XFR
                                              //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                            //允许访问扩展的特殊寄存器，XFR
                                              //(8 位模式请使用这句，注释上一句)
```

```
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P1M1 = 0x0f;
    P1M0 = 0x00;

    PWMA_ENO = 0x00;                            //配置成 TRGI 的 pin 需关掉 ENO 对应 bit 并配成 input
    PWMA_PS = 0x00;                             //00:PWM at P1

    PWMA_PSCRH = 0x00;                          //预分频寄存器
    PWMA_PSCRL = 0x00;

    PWMA_CCMR1  = 0x21;                         //通道模式配置为输入，接编码器,滤波器 4 时钟
    PWMA_CCMR2  = 0x21;                         //通道模式配置为输入，接编码器,滤波器 4 时钟

    PWMA_SMCR   = 0x03;                         //编码器模式 3

    PWMA_CCER1  = 0x55;                         //配置通道使能和极性
    PWMA_CCER2  = 0x55;                         //配置通道使能和极性

    PWMA_IER  = 0x02;                           //使能中断

    PWMA_CR1 |= 0x01;                           //使能计数器

    EA = 1;

    while (1);
}

/******************** PWM 中断读编码器计数值*************************/
void PWMA_ISR() interrupt 26
{
    if (PWMA_SR1 & 0X02)
    {
        P03 = ~P03;
        cnt_H = PWMA_CCR1H;
        cnt_L = PWMA_CCR1L;
        PWMA_SR1 &= ~0X02;
    }
}
```

29.10.21　使用高级 PWM 实现编码器演示程序

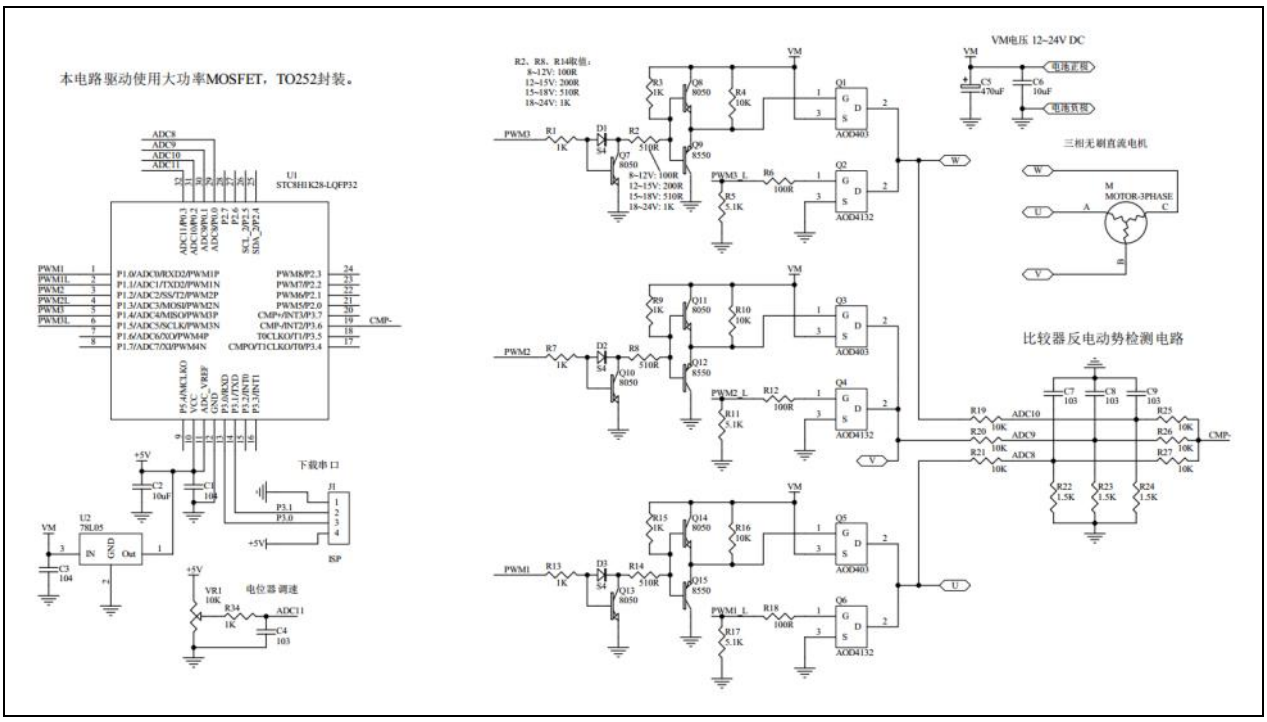
```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                            //头文件见下载软件

#define     MAIN_Fosc     11059200L//定义主时钟

/*************  功能说明 **************

PWMA 模块工作于编码器模式. PWMA 模块只能接一个编码器.
串口 1(RXD-->P3.0    TXD-->P3.1)返回读数结果, 串口设置 115200,8,n,1;
编码器 A 相输入: PWM1P (P1.0)
```

```
编码器 B 相输入: PWM2P (P1.2)

编码器模式,             模式 1:  每个脉冲两个边沿加减 2.
                        模式 2:  每个脉冲两个边沿加减 2.
                        模式 3:  每个脉冲两个边沿加减 4.

*****************************************/

unsigned int    pulse;                          //编码器脉冲
bit             B_Change;                       //编码器计数改变

bit             B_TX1_Busy;                     //  发送忙标志

void            PWMA_config(void);
void            UART1_config(unsigned long brt);    // brt:  通信波特率
void            UART1_TxByte(unsigned char dat);


void main(void)
{
    unsigned int j;

    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间


    P1M1 = 0x00;
    P1M0 = 0x00;

    UART1_config(115200UL);                     // brt:  通信波特率

    EA = 1;

    PWMA_config();
    pulse = 10;

    while (1)
    {
        if(B_Change)
        {
            B_Change = 0;
            j = pulse;
            UART1_TxByte(j/10000+'0');          //转成十进制文本并发送
            UART1_TxByte((j%10000)/1000+'0');
            UART1_TxByte((j%1000)/100+'0');
            UART1_TxByte((j%100)/10+'0');
            UART1_TxByte( j%10+'0');
            UART1_TxByte(0x0d);
            UART1_TxByte(0x0a);
        }
    }
}
```

```
//========================================================================
// 函数: void PWMA_config(void)
// 描述: PPWM 配置函数。
// 参数: noe.
// 返回: none.
// 版本: V1.0
// 备注:
//========================================================================
void PWMA_config(void)
{
    PWMA_PSCRH = 0;                         //预分频 Fck_cnt = Fck_psc/(PSCR[15:0]+1),
    PWMA_PSCRL = 0;                         //边沿对齐 PWM 频率 =SYSclk/((PSCR+1)*(ARR+1)),
                                            //中央对齐频率 =SYSclk/((PSCR+1)*(ARR+1)*2).
    PWMA_ARRH = 0xff;                       //自动重装载寄存器,控制 PWM 周期
    PWMA_ARRL = 0xff;
    PWMA_CNTRH = 0;                         //清零编码器计数器值
    PWMA_CNTRL = 0;
    PWMA_ENO = 0;                           //IO 禁止输出 PWM

    PWMA_CCMR1 = 0x01+(10<<4);              //通道 1 模式配置, 配置成输入通道,
                                            //0~15 对应输入滤波时钟数:
                                            //1 2 4 8 12 16   24 32 48 64 80 96 128 160 192 256
    PWMA_CCMR2 = 0x01+(10<<4);              //通道 2 模式配置, 配置成输入通道,
                                            //0~15 对应输入滤波时钟数:
                                            //1 2 4 8 12 16 24 32 48 64 80 96 128 160 192 256
    PWMA_SMCR = 2;                          //编码器模式,模式 1 或模式 2:每个脉冲两个边沿加减 2.
                                            //模式 3:  每个脉冲四个边沿加减 4.
    PWMA_CCER1 = 0x55;                      //配置通道输入使能和极性, 允许输入,下降沿
    PWMA_PS = 0;                            //IO 选择 P1.0 P1.2
    PWMA_IER = 0x02;                        //使能中断
    PWMA_CR1 = 0x01;                        //使能计数器, 允许自动重装载寄存器缓冲,
                                            //边沿对齐模式,      向上计数,
                                            //bit7=1:写自动重装载寄存器缓冲(本周期不会被打扰),
                                            //    =0:直接写自动重装载寄存器本(周期可能会乱掉)
}

//========================================================================
// 函数: void PWMA_ISR(void) interrupt PWMA_VECTOR
// 描述: PWMA 中断处理程序.
// 参数: None
// 返回: none.
// 版本: V1.0
//========================================================================
void PWMA_ISR(void) interrupt 26
{
    if(PWMA_SR1 & 0x02)                     //编码器中断
    {
        pulse = PWMA_CNTRH;                 //读取当前编码器计数值
        pulse <<= 8;
        pulse |= PWMA_CNTRL;
        B_Change = 1;                       //标志已有捕捉值
    }
    PWMA_SR1 = 0;
}

//========================================================================
// 函数: void                UART1_config(u32 brt)
// 描述: UART1 初始化函数。
// 参数:     brt:            通信波特率.
```

```
//  返回: none.
//  版本: VER1.0
//  备注:
//=========================================================================
void UART1_config(unsigned long brt)                    // brt:  通信波特率
{
    Brt = 65536UL - (MAIN_Fosc / 4) / brt;
    AUXR &= ~0x01;                                      //S1 BRT Use Timer1;
    AUXR |= (1<<6);                                     //Timer1 set as 1T mode
    TMOD &= 0x0f;                                       //Timer1 16bits AutoReload;
    TH1 = (unsigned char)(brt >> 8);
    TL1 = (unsigned char)brt;
    TR1 = 1;                                            //  运行 Timer1
    P_SW1 &= ~0xc0;                                     //串口 1 切换到 P3.0 P3.1
    SCON = (SCON & 0x3f) | (1<<6);                      // 8 位数据, 1 位起始位, 1 位停止位,  无校验
    ES = 1;                                             //允许中断
    REN = 1;                                            //允许接收
}

//=========================================================================
//  函数: void UART1_TxByte(u8 dat)
//  描述:  串口 1 查询发送一个字节函数.
//  参数: dat:  要发送的字节数据.
//  返回: none.
//  版本: VER1.0
//  备注:
//=========================================================================
void UART1_TxByte(unsigned char dat)
{
    B_TX1_Busy = 1;                                     //标志发送忙
    SBUF = dat;                                         //发一个字节
    while(B_TX1_Busy);                                  //等待发送完成
}

//=========================================================================
//  函数: void UART1_int     (void) interrupt UART1_VECTOR
//  描述:  串口 1 中断函数
//  参数: none.
//  返回: none.
//  版本: VER1.0
//  备注:
//=========================================================================
void UART1_int (void) interrupt   4
{
    if(RI)
        RI = 0;

    if(TI)
    {
        TI = 0;
        B_TX1_Busy = 0;
    }
}
```

29.10.22 无 HALL 三相无刷电机驱动演示程序，电位器调速用



详细代码讲解及视频请上官网论坛：三相无刷电机驱动-AI8H-无 HALL，有视频讲解

C 语言代码

```
/************ 功能说明 **************

本程序试验使用 AI8H1K28-LQFP32 来驱动无传感器无刷三相直流电机.

P0.3 接的电位器控制转速, 逆时针旋转电位器电压降低电机减速, 顺时针旋转电位器电压升高电机加速.
关于无感三相无刷直流电机的原理, 用户自行学习了解, 本例不予说明.


*****************************************/

#define MAIN_Fosc       24000000L           //定义主时钟

#include "Ai8051U.H"                        //头文件见下载软件


#define ADC_START  (1<<6)                   /*  自动清 0 */
#define ADC_FLAG   (1<<5)                   /*  软件清 0 */

#define     ADC_SPEED   1                   /* 0~15, ADC 时钟 = SYSclk/2/(n+1) */
#define     RES_FMT     (1<<5)

#define CSSETUP         (0<<7)
#define CSHOLD          (1<<5)
#define SMPDUTY         20

sbit PWM1   = P1^0;
sbit PWM1_L = P1^1;
sbit PWM2   = P1^2;
sbit PWM2_L = P1^3;
```

```
sbit PWM3   = P1^4;
sbit PWM3_L = P1^5;

u8   step;                                  //切换步骤
u8   PWM_Value;                             //  决定 PWM 占空比的值
bit  B_RUN;                                 //运行标志
u8   PWW_Set;                               //目标 PWM 设置
u16  adc11;
bit  B_4ms;                                 //4ms 定时标志

u8   TimeOut;                               //堵转超时
bit  B_start;                               //启动模式
bit  B_Timer3_OverFlow;

u8   TimeIndex;                             //换相时间保存索引
u16  PhaseTimeTmp[8];                       //8 个换相时间, 其 sum/16 就是 30 度电角度
u16  PhaseTime;                             //换相时间计数
u8   XiaoCiCnt;                             //1:需要消磁, 2:正在消磁, 0 已经消磁


/*************************/

void  Delay_n_ms(u8 dly)                    // N ms 延时函数
{
    u16  j;
    do
    {
        j = MAIN_Fosc / 10000;
        while(--j)  ;
    }while(--dly);
}

void delay_us(u8 us)   //N us 延时函数
{
    do
    {
        NOP(20);  //@24MHz
    }
    while(--us);
}

//========================================================================
//  函数: u16    Get_ADC10bitResult(u8 channel))        //channel = 0~15
//========================================================================
u16  Get_ADC10bitResult(u8 channel)         //channel = 0~15
{
    u8 i;
    ADC_RES = 0;
    ADC_RESL = 0;
    ADC_CONTR = 0x80 | ADC_START | channel;
    NOP(5);
//  while((ADC_CONTR & ADC_FLAG) == 0)      ;//等待 ADC 结束
        i = 255;
        while(i != 0)
        {
            i--;
            if((ADC_CONTR & ADC_FLAG) != 0) break;  //等待 ADC 结束
        }
    ADC_CONTR &= ~ADC_FLAG;
```

```
    return      ((u16)ADC_RES * 256 + (u16)ADC_RESL);
}


void  Delay_500ns(void)
{
    NOP(6);
}

void StepMotor(void)                                    // 换相序列函数
{
    switch(step)
    {
    case 0:   // AB   PWM1, PWM2_L=1
              PWMA_ENO = 0x00; PWM1_L=0; PWM3_L=0;
              Delay_500ns();
              PWMA_ENO = 0x01;                          // 打开 A 相的高端 PWM
              PWM2_L = 1;                               // 打开 B 相的低端
              ADC_CONTR = 0x80+10;                      // 选择 P0.2 作为 ADC 输入 即 C 相电压
              CMPCR1 = 0x8c + 0x10;                     //比较器下降沿中断
              break;
    case 1:   // AC   PWM1, PWM3_L=1
              PWMA_ENO = 0x01; PWM1_L=0; PWM2_L=0;      // 打开 A 相的高端 PWM
              Delay_500ns();
              PWM3_L = 1;                               // 打开 C 相的低端
              ADC_CONTR = 0x80+9;                       // 选择 P0.1 作为 ADC 输入 即 B 相电压
              CMPCR1 = 0x8c + 0x20;                     //比较器上升沿中断
              break;
    case 2:   // BC   PWM2, PWM3_L=1
              PWMA_ENO = 0x00; PWM1_L=0; PWM2_L=0;
              Delay_500ns();
              PWMA_ENO = 0x04;                          // 打开 B 相的高端 PWM
              PWM3_L = 1;                               // 打开 C 相的低端
              ADC_CONTR = 0x80+8;                       // 选择 P0.0 作为 ADC 输入 即 A 相电压
              CMPCR1 = 0x8c + 0x10;                     //比较器下降沿中断
              break;
    case 3:   // BA   PWM2, PWM1_L=1
              PWMA_ENO = 0x04; PWM2_L=0; PWM3_L=0;      // 打开 B 相的高端 PWM
              Delay_500ns();
              PWM1_L = 1;                               // 打开 C 相的低端
              ADC_CONTR = 0x80+10;                      // 选择 P0.2 作为 ADC 输入 即 C 相电压
              CMPCR1 = 0x8c + 0x20;                     //比较器上升沿中断
              break;
    case 4:   // CA   PWM3, PWM1_L=1
              PWMA_ENO = 0x00; PWM2_L=0; PWM3_L=0;
              Delay_500ns();
              PWMA_ENO = 0x10;                          // 打开 C 相的高端 PWM
              PWM1_L = 1;                               // 打开 A 相的低端
              adc11 = ((adc11 *7)>>3) + Get_ADC10bitResult(11);
              ADC_CONTR = 0x80+9;                       // 选择 P0.1 作为 ADC 输入 即 B 相电压
              CMPCR1 = 0x8c + 0x10;                     //比较器下降沿中断
              break;
    case 5:   // CB   PWM3, PWM2_L=1
              PWMA_ENO = 0x10; PWM1_L=0; PWM3_L=0;      // 打开 C 相的高端 PWM
              Delay_500ns();
              PWM2_L = 1;                               // 打开 B 相的低端
              ADC_CONTR = 0x80+8;                       // 选择 P0.0 作为 ADC 输入 即 A 相电压
              CMPCR1 = 0x8c + 0x20;                     //比较器上升沿中断
              break;
```

```
        default:
                break;
        }

        if(B_start)         CMPCR1 = 0x8C;                  // 启动时禁止下降沿和上升沿中断
}



void PWMA_config(void)
{
        P_SW2 |= 0x80;          //SFR enable

        PWM1     = 0;
        PWM1_L = 0;
        PWM2     = 0;
        PWM2_L = 0;
        PWM3     = 0;
        PWM3_L = 0;
        P1n_push_pull(0x3f);

        PWMA_PSCRH = 0;                                 //预分频寄存器,分频 Fck_cnt = Fck_psc/(PSCR[15:0}+1),
        PWMA_PSCRL = 3;                                 //边沿对齐 PWM 频率 = SYSclk/((PSCR+1)*(AAR+1)),
                                                        //中央对齐 PWM 频率 = SYSclk/((PSCR+1)*AAR*2).
        PWMA_DTR    = 24;                               // 死区时间配置, n=0~127: DTR= n T,
                                                        //0x80 ~(0x80+n), n=0~63: DTR=(64+n)*2T,
                                                        //0xc0 ~(0xc0+n), n=0~31: DTR=(32+n)*8T,
                                                        //0xE0 ~(0xE0+n), n=0~31: DTR=(32+n)*16T,
        PWMA_ARRH       = 0;                            // 自动重装载寄存器,   控制 PWM 周期
        PWMA_ARRL       = 255;
        PWMA_CCER1    = 0;
        PWMA_CCER2    = 0;
        PWMA_SR1        = 0;
        PWMA_SR2        = 0;
        PWMA_ENO        = 0;
        PWMA_PS         = 0;
        PWMA_IER        = 0;
//      PWMA_ISR_En = 0;

        PWMA_CCMR1    = 0x68;                           // 通道模式配置, PWM 模式 1,  预装载允许
        PWMA_CCR1H      = 0;                                // 比较值,  控制占空比(高电平时钟数)
        PWMA_CCR1L      = 0;
        PWMA_CCER1 |= 0x05;                             // 开启比较输出,  高电平有效
        PWMA_PS       |= 0;                             // 选择 IO, 0:P1.0 P1.1, 1:P2.0 P2.1, 2:P6.0 P6.1,
//      PWMA_IER      |= 0x02;                          // 使能中断

        PWMA_CCMR2    = 0x68;                           // 通道模式配置, PWM 模式 1,  预装载允许
        PWMA_CCR2H      = 0;                                // 比较值,  控制占空比(高电平时钟数)
        PWMA_CCR2L      = 0;
        PWMA_CCER1 |= 0x50;                             // 开启比较输出,  高电平有效
        PWMA_PS       |= (0<<2);                        // 选择 IO, 0:P1.2 P1.3, 1:P2.2 P2.3, 2:P6.2 P6.3,
//      PWMA_IER      |= 0x04;                          // 使能中断

        PWMA_CCMR3    = 0x68;                           // 通道模式配置, PWM 模式 1,  预装载允许
        PWMA_CCR3H      = 0;                                // 比较值,  控制占空比(高电平时钟数)
        PWMA_CCR3L      = 0;
        PWMA_CCER2 |= 0x05;                             // 开启比较输出,  高电平有效
        PWMA_PS       |= (0<<4);                        // 选择 IO, 0:P1.4 P1.5, 1:P2.4 P2.5, 2:P6.4 P6.5,
```

```
//      PWMA_IER    |= 0x08;                    // 使能中断

        PWMA_BKR     = 0x80;                   // 主输出使能 相当于总开关
        PWMA_CR1     = 0x81;                   // 使能计数器, 允许自动重装载寄存器缓冲,
                                               //边沿对齐模式, 向上计数,  bit7=1:写自动重装载
                                               //寄存器缓冲(本周期不会被打扰), =0:直接写自动
                                               //重装载寄存器本(周期可能会乱掉)
        PWMA_EGR     = 0x01;                   //产生一次更新事件, 清除计数器和与分频计数器,
                                               //装载预分频寄存器的值
//      PWMA_ISR_En = PWMA_IER;                  //设置标志允许通道 1~4 中断处理
}

void ADC_config(void)                          //ADC 初始化函数(为了使用 ADC 输入端做比较器信号,
                                               //实际没有启动 ADC 转换)
{
        P1n_pure_input(0xc0);                  //设置为高阻输入
        P0n_pure_input(0x0f);                  //设置为高阻输入
        ADC_CONTR = 0x80 + 6;                  //ADC on + channel
        ADCCFG = RES_FMT + ADC_SPEED;
        P_SW2 |=    0x80;                      //访问 XSFR
        ADCTIM = CSSETUP + CSHOLD + SMPDUTY;
}

void CMP_config(void)                          //比较器初始化程序
{
        CMPCR1 = 0x8C;                         // 1000 1100  打开比较器，P3.6 作为比较器的反相
                                               //输入端，ADC 引脚作为正输入端
        CMPCR2 = 60;                           //60 个时钟滤波     比较结果变化延时周期数, 0~63
        P3n_pure_input(0x40);                  //CMP-(P3.6)设置为高阻.

        P_SW2 |= 0x80;                         //SFR enable
//      CMPEXCFG |= (0<<6);                    //bit7 bit6:  比较器迟滞输入选择: 0: 0mV,
                                               //1: 10mV, 2: 20mV, 3: 30mV
//      CMPEXCFG |= (0<<2);                    //bit2:  输入负极性选择, 0:  选择 P3.6 做输入,
                                               //1:  选择内部 BandGap 电压 BGv 做负输入.
//      CMPEXCFG |=    0;                      //bit1 bit0:  输入正极性选择, 0:  选择 P3.7 做输入,
                                               //1:  选择 P5.0 做输入,   2:  选择 P5.1 做输入,
                                               //3:  选择 ADC 输入(由 ADC_CHS[3:0]所选择的
                                               //ADC 输入端做正输入).
//      CMPEXCFG = (0<<6)+(0<<2)+3;
}

void CMP_ISR(void) interrupt 21                //比较器中断函数, 检测到反电动势过 0 事件
{
        u8    i;
        CMPCR1 &= ~0x40;                       //  需软件清除中断标志位

        if(XiaoCiCnt == 0)                     //消磁后才检测过 0 事件,     XiaoCiCnt=1:需要消磁,
                                               // =2:正在消磁, =0 已经消磁
        {
                T4T3M &= ~(1<<3);              // Timer3 停止运行
                if(B_Timer3_OverFlow)          //切换时间间隔(Timer3)有溢出
                {
                        B_Timer3_OverFlow = 0;
                        PhaseTime = 8000;      //换相时间最大 8ms, 2212 电机 12V 空转最高速 130us
                                               //切换一相(200RPS 12000RPM), 480mA
                }
                else
                {
```

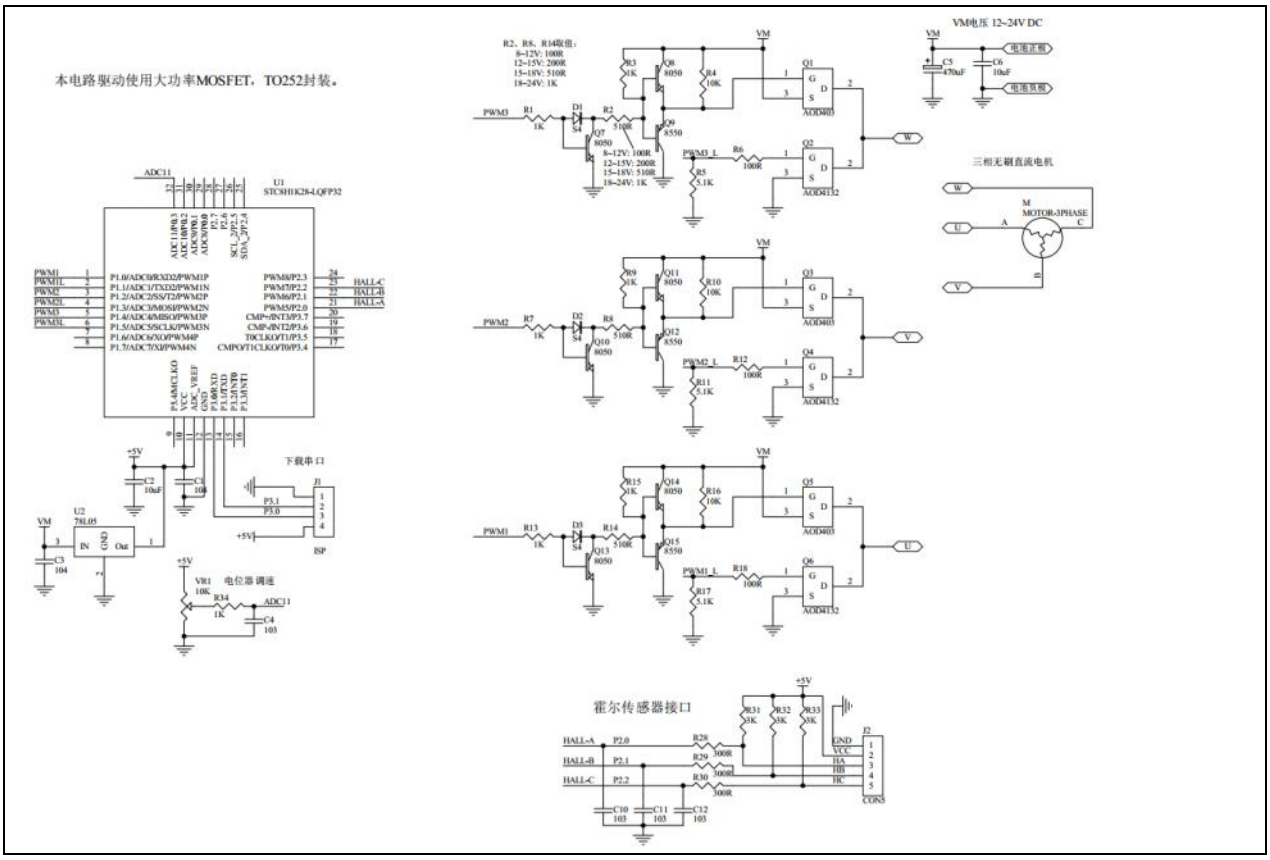
```
            PhaseTime = (((u16)T3H << 8) + T3L) >> 1;       //单位为 1us
            if(PhaseTime >= 8000) PhaseTime = 8000;         //换相时间最大 8ms, 2212 电机 12V 空转最高速 130us
                                                            //切换一相(200RPS 12000RPM), 480mA
        }
        T3H = 0;   T3L = 0;
        T4T3M |=   (1<<3);                                  //Timer3 开始运行

        PhaseTimeTmp[TimeIndex] = PhaseTime;                //保存一次换相时间
        if(++TimeIndex >= 8) TimeIndex = 0;                 //累加 8 次
        for(PhaseTime=0, i=0; i<8; i++) PhaseTime += PhaseTimeTmp[i]; //求 8 次换相时间累加和
        PhaseTime = PhaseTime >> 4;                         //求 8 次换相时间的平均值的一半, 即 30 度电角度
        if((PhaseTime >= 40) && (PhaseTime <= 1000)) TimeOut = 125;   //堵转 500ms 超时
        if( PhaseTime >= 60)  PhaseTime -= 40;              //修正由于滤波电容引起的滞后时间
        else   PhaseTime   = 20;

    //  PhaseTime = 20;                                     //只给 20us, 则无滞后修正, 用于检测滤波电容
                                                            //引起的滞后时间
        T4T3M &= ~(1<<7);                                   //Timer4 停止运行
        PhaseTime   = PhaseTime   << 1;                     //2 个计数 1us
        PhaseTime = 0 - PhaseTime;
        T4H = (u8)(PhaseTime >> 8);                         //装载 30 度角延时
        T4L = (u8)PhaseTime;
        T4T3M |=   (1<<7);                                  //Timer4 开始运行
        XiaoCiCnt = 1;                                      //1:需要消磁, 2:正在消磁, 0 已经消磁
    }
}

void Timer0_config(void)                                    //Timer0 初始化函数
{
    Timer0_16bitAutoReload();                               // T0 工作于 16 位自动重装
    Timer0_12T();
    TH0 = (65536UL-MAIN_Fosc/12 / 250) / 256; //4ms
    TL0 = (65536UL-MAIN_Fosc/12 / 250) % 256;
    TR0 = 1;                                                //  打开定时器 0

    ET0 = 1;                                                //  允许 ET0 中断
}

void Timer0_ISR(void) interrupt 1                           //Timer0 中断函数, 20us
{
    B_4ms = 1;      //4ms 定时标志
}

//========================== timer3 初始化函数 ========================================
void  Timer3_Config(void)
{
    P_SW2 |= 0x80;                                          //SFR enable
    T4T3M &= 0xf0;                                          //停止计数, 定时模式, 12T 模式, 不输出时钟
    T3H = 0;
    T3L = 0;

    IE2   |=   (1<<5);                                      //允许中断
    T4T3M |=   (1<<3);                                      //开始运行
}

//========================== timer4 初始化函数 ========================================
void  Timer4_Config(void)
{
    P_SW2 |= 0x80;                                          //SFR enable
```

```
    T4T3M &= 0x0f;                          //停止计数, 定时模式, 12T 模式, 不输出时钟
    T4H = 0;
    T4L = 0;

    IE2   |=  (1<<6);                       //允许中断
//  T4T3M |=  (1<<7);                       //开始运行
}

//========================== timer3 中断函数 ============================================
void timer3_ISR (void) interrupt TIMER3_VECTOR
{
    B_Timer3_OverFlow = 1;                  //溢出标志
}

//========================== timer4 中断函数 ============================================
void timer4_ISR (void) interrupt TIMER4_VECTOR
{
    T4T3M &= ~(1<<7);                       //Timer4 停止运行
    if(XiaoCiCnt == 1)                      //标记需要消磁. 每次检测到过 0 事件后第一次
                                            //中断为 30 度角延时, 设置消磁延时.
    {
        XiaoCiCnt = 2;                      //1:需要消磁, 2:正在消磁, 0 已经消磁
        if(B_RUN)                           //电机正在运行
        {
            if(++step >= 6)  step = 0;
            StepMotor();
        }
                                            //消磁时间, 换相后线圈(电感)电流减小到 0 的过程中,
                                            //出现反电动势, 电流越大消磁时间越长,
                                            //过 0 检测要在这个时间之后
                                            //100%占空比时施加较重负载, 电机电流上升,
                                            //可以示波器看消磁时间.
                                            //只要在换相后延时几十 us 才检测过零, 就可以了
        T4H = (u8)((65536UL - 40*2) >> 8);  //装载消磁延时
        T4L = (u8)(65536UL - 40*2);
        T4T3M |=  (1<<7);                   //Timer4 开始运行
    }
    else if(XiaoCiCnt == 2)     XiaoCiCnt = 0;      //1:需要消磁, 2:正在消磁, 0 已经消磁
}


#define     D_START_PWM     30
/******************** 强制电机启动函数 ***************************/
void StartMotor(void)
{
    u16 timer,i;
    CMPCR1 = 0x8C;                          // 关比较器中断

    PWM_Value   = D_START_PWM;              // 初始占空比, 根据电机特性设置
    PWMA_CCR1L = PWM_Value;
    PWMA_CCR2L = PWM_Value;
    PWMA_CCR3L = PWM_Value;
    step = 0;   StepMotor();    Delay_n_ms(50);     //Delay_n_ms(250);// 初始位置
    timer = 200;                            //风扇电机启动

    while(1)
    {
        for(i=0; i<timer; i++)  delay_us(100);      //根据电机加速特性, 最高转速等等调整启动加速速度
        timer -= timer /16;
```

```
            if(++step >= 6)   step = 0;
            StepMotor();
            if(timer < 40)    return;
        }
}

/*********************************************/
void main(void)
{
    u8    i;
    u16   j;

    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P2n_standard(0xf8);
    P3n_standard(0xbf);
    P5n_standard(0x10);

    adc11 = 0;

    PWMA_config();
    ADC_config();
    CMP_config();
    Timer0_config();                            // Timer0 初始化函数
    Timer3_Config();                            // Timer3 初始化函数
    Timer4_Config();                            // Timer4 初始化函数
    PWW_Set = 0;
    TimeOut = 0;

    EA   = 1; // 打开总中断


    while (1)
    {
        if(B_4ms)                               // 4ms 时隙
        {
            B_4ms = 0;

            if(TimeOut != 0)
            {
                if(--TimeOut == 0)              //堵转超时
                {
                    B_RUN   = 0;
                    PWM_Value = 0;
                    CMPCR1 = 0x8C;              // 关比较器中断
                    PWMA_ENO   = 0;
                    PWMA_CCR1L = 0; PWMA_CCR2L = 0; PWMA_CCR3L = 0;
                    PWM1_L=0; PWM2_L=0; PWM3_L=0;
                    Delay_n_ms(250);            //堵转时,延时一点时间再启动
                }
            }
```

```
            if(!B_RUN && (PWW_Set >= D_START_PWM)) // 占空比大于设定值, 并且电机未运行, 则启动电机
            {
                B_start = 1;                                //启动模式
                for(i=0; i<8; i++) PhaseTimeTmp[i] = 400;
                StartMotor();                               // 启动电机
                B_start = 0;
                XiaoCiCnt = 0;                              //初始进入时
                CMPCR1 &= ~0x40;                            // 清除中断标志位
                if(step & 1)CMPCR1 = 0xAC;                  //上升沿中断
                else        CMPCR1 = 0x9C;                  //下降沿中断
                B_RUN = 1;
                Delay_n_ms(250);                            //延时一下, 先启动起来
                Delay_n_ms(250);
                TimeOut = 125;                              //启动超时时间 125*4 = 500ms
            }

            if(B_RUN)//正在运行中
            {
                if(PWM_Value < PWW_Set) PWM_Value++;        //油门跟随电位器
                if(PWM_Value > PWW_Set) PWM_Value--;
                if(PWM_Value < (D_START_PWM-10))            // 停转, 停转占空比 比 启动占空比 小 10/256
                {
                    B_RUN = 0;
                    PWM_Value = 0;
                    CMPCR1 = 0x8C;                          // 关比较器中断
                    PWMA_ENO    = 0;
                    PWMA_CCR1L = 0; PWMA_CCR2L = 0; PWMA_CCR3L = 0;
                    PWM1_L=0; PWM2_L=0; PWM3_L=0;
                }
                else
                {
                    PWMA_CCR1L = PWM_Value;
                    PWMA_CCR2L = PWM_Value;
                    PWMA_CCR3L = PWM_Value;
                }
            }
            else
            {
                adc11 = ((adc11 *7)>>3) + Get_ADC10bitResult(11);
            }

            j = adc11;
            if(j != adc11)      j = adc11;
            PWW_Set = (u8)(j >> 5);                         //油门是 8 位的
        }
    }
}
```

29.10.23　带 HALL 三相无刷电机驱动演示程序，电位器调速，捕捉中断换相



详细代码讲解及视频请上官网论坛：三相无刷电机驱动-AI8H-带 HALL

C 语言代码

```
#include “Ai8051U.H”                                    //头文件见下载软件
/************ 功能说明 **************

本程序试验使用 AI8H1K28-LQFP32 来驱动带霍尔传感器的无刷三相直流电机.

PWM 的捕捉中断功能用来检测霍尔信号.
P0.3 接的电位器控制转速，处于中间位置为停止，逆时针旋转电位器电压降低电机为逆时针转，顺时针旋转电位器电压升高电机为顺时针转.
关于带霍尔传感器三相无刷直流电机的原理，用户自行学习了解，本例不予说明.

*****************************************/


#define MAIN_Fosc        24000000L                      //定义主时钟

#define ADC_START  (1<<6)                               /*  自动清 0 */
#define ADC_FLAG   (1<<5)                               /*  软件清 0 */

#define    ADC_SPEED    1
#define    RES_FMT      (1<<5)

#define CSSETUP          (0<<7)
#define SMPDUTY          20
```

```
sbit PWM1   = P1^0;
sbit PWM1_L = P1^1;
sbit PWM2   = P1^2;
sbit PWM2_L = P1^3;
sbit PWM3   = P1^4;
sbit PWM3_L = P1^5;

u8   step;                                          //切换步骤
u8   PWM_Value;                                     //  决定 PWM 占空比的值
bit  B_RUN;                                         //运行标志
u8   PWW_Set;                                       //目标 PWM 设置
u8   YouMen;                                        //油门
bit  B_direct;                                      //转向, 0 顺时针, 1 逆时针
bit  B_4ms;                                         //4ms 定时标志

//========================================================================
//  函数: u16     Get_ADC10bitResult(u8 channel)        //channel = 0~15
//========================================================================
u16  Get_ADC10bitResult(u8 channel)                 //channel = 0~15
{
    u8 i;
    ADC_RES = 0;
    ADC_RESL = 0;
    ADC_CONTR = 0x80 | ADC_START | channel;
    NOP(5);                                         //
    i = 255;                                        //超时限制
    while(i != 0)
    {
        i--;
        if((ADC_CONTR & ADC_FLAG) != 0) break;      //等待 ADC 结束
    }
    ADC_CONTR &= ~ADC_FLAG;
    return      ((u16)ADC_RES * 256 + (u16)ADC_RESL);
}


void  Delay_500ns(void)
{
    NOP(6);
}

void StepMotor(void)                                //  换相序列函数
{
    PWMB_IER  = 0;
    PWMB_CCER1 = 0;
    PWMB_CCER2 = 0;

    step = P2 & 0x07;                               //P2.0-HALL_A P2.1-HALL_B P2.2-HALL_C
    if(!B_direct)                                   //顺时针
    {
        switch(step)
        {
        case 2:  // 010, P2.0-HALL_A 下降沿   PWM3, PWM2_L=1       //顺时针
                 PWMA_ENO = 0x00; PWM1_L=0; PWM3_L=0;
                 Delay_500ns();
                 PWMA_ENO = 0x10;                   //  打开 C 相的高端 PWM
                 PWM2_L = 1;                        //  打开 B 相的低端
                 PWMB_CCER2 = (0x01+0x00);          //P2.2 0x01:允许输入捕获, +0x00:上升沿, +0x02:下降沿
                 PWMB_IER  = 0x08;                  //P2.2  使能中断
```

```
                break;
        case 6:  // 110, P2.2-HALL_C 上升沿   PWM3, PWM1_L=1
                PWMA_ENO = 0x10; PWM2_L=0; PWM3_L=0;        // 打开 C 相的高端 PWM
                Delay_500ns();
                PWM1_L = 1;                                 // 打开 A 相的低端
                PWMB_CCER1 = (0x10+0x20);                   //P2.1 0x10:允许输入捕获, +0x00:上升沿, +0x20: 下降沿
                PWMB_IER   = 0x04;                          //P2.1 使能中断
                break;
        case 4:  // 100, P2.1-HALL_B 下降沿   PWM2, PWM1_L=1
                PWMA_ENO = 0x00; PWM2_L=0; PWM3_L=0;
                Delay_500ns();
                PWMA_ENO = 0x04;                            // 打开 B 相的高端 PWM
                PWM1_L = 1;                                 // 打开 A 相的低端
                PWMB_CCER1 = (0x01+0x00);                   //P2.0 0x01:允许输入捕获, +0x00:上升沿, +0x02: 下降沿
                PWMB_IER   = 0x02;                          //P2.0 使能中断
                break;
        case 5:  // 101, P2.0-HALL_A 上升沿   PWM2, PWM3_L=1
                PWMA_ENO = 0x04; PWM1_L=0; PWM2_L=0;  // 打开 B 相的高端 PWM
                Delay_500ns();
                PWM3_L = 1;                                 // 打开 C 相的低端
                PWMB_CCER2 = (0x01+0x02);                   //P2.2 0x01:允许输入捕获, +0x00:上升沿, +0x02: 下降沿
                PWMB_IER   = 0x08;                          //P2.2 使能中断
                break;
        case 1:  // 001, P2.2-HALL_C 下降沿   PWM1, PWM3_L=1
                PWMA_ENO = 0x00; PWM1_L=0; PWM2_L=0;
                Delay_500ns();
                PWMA_ENO = 0x01;                            // 打开 A 相的高端 PWM
                PWM3_L = 1;                                 // 打开 C 相的低端
                PWMB_CCER1 = (0x10+0x00);                   //P2.1 0x10:允许输入捕获, +0x00:上升沿, +0x20: 下降沿
                PWMB_IER   = 0x04;                          //P2.1 使能中断
                break;
        case 3:  // 011, P2.1-HALL_B 上升沿   PWM1, PWM2_L=1
                PWMA_ENO = 0x01; PWM1_L=0; PWM3_L=0;  // 打开 A 相的高端 PWM
                Delay_500ns();
                PWM2_L = 1;                                 // 打开 B 相的低端
                PWMB_CCER1 = (0x01+0x02);                   //P2.0 0x01:允许输入捕获, +0x00:上升沿, +0x02: 下降沿
                PWMB_IER   = 0x02;                          //P2.0 使能中断
                break;

        default:
                break;
        }
}

else  //逆时针
{
        switch(step)
        {
        case 4:  // 100, P2.0-HALL_A 下降沿   PWM1, PWM2_L=1       //逆时针
                PWMA_ENO = 0x00; PWM1_L=0; PWM3_L=0;
                Delay_500ns();
                PWMA_ENO = 0x01;                            // 打开 A 相的高端 PWM
                PWM2_L = 1;                                 // 打开 B 相的低端
                PWMB_CCER1 = (0x10+0x00);                   //P2.1 0x10:允许输入捕获, +0x00:上升沿, +0x20: 下降沿
                PWMB_IER   = 0x04;                          //P2.1 使能中断
                break;
        case 6:  // 110, P2.1-HALL_B 上升沿   PWM1, PWM3_L=1
                PWMA_ENO = 0x01; PWM1_L=0; PWM2_L=0;  // 打开 A 相的高端 PWM
                Delay_500ns();
```

```
                PWM3_L = 1;                             // 打开 C 相的低端
                PWMB_CCER2 = (0x01+0x02);               //P2.2 0x01:允许输入捕获, +0x00:上升沿, +0x02: 下降沿
                PWMB_IER   = 0x08;                      //P2.2 使能中断
                break;
        case 2:  // 010, P2.2-HALL_C 下降沿   PWM2, PWM3_L=1
                PWMA_ENO = 0x00; PWM1_L=0; PWM2_L=0;
                Delay_500ns();
                PWMA_ENO = 0x04;                        // 打开 B 相的高端 PWM
                PWM3_L = 1;                             // 打开 C 相的低端
                PWMB_CCER1 = (0x01+0x00);               //P2.0 0x01:允许输入捕获, +0x00:上升沿, +0x02: 下降沿
                PWMB_IER   = 0x02;                      //P2.0 使能中断
                break;
        case 3:  // 011, P2.0-HALL_A 上升沿   PWM2, PWM1_L=1
                PWMA_ENO = 0x04; PWM2_L=0; PWM3_L=0;  // 打开 B 相的高端 PWM
                Delay_500ns();
                PWM1_L = 1;                             // 打开 A 相的低端
                PWMB_CCER1 = (0x10+0x20);               //P2.1 0x10:允许输入捕获, +0x00:上升沿, +0x20: 下降沿
                PWMB_IER   = 0x04;                      //P2.1 使能中断
                break;
        case 1:  // 001, P2.1-HALL_B 下降沿   PWM3, PWM1_L=1
                PWMA_ENO = 0x00; PWM2_L=0; PWM3_L=0;
                Delay_500ns();
                PWMA_ENO = 0x10;                        // 打开 C 相的高端 PWM
                PWM1_L = 1;                             // 打开 A 相的低端
                PWMB_CCER2 = (0x01+0x00);               //P2.2 0x01:允许输入捕获, +0x00:上升沿, +0x02: 下降沿
                PWMB_IER   = 0x08;                      //P2.2 使能中断
                break;
        case 5:  // 101, P2.2-HALL_C 上升沿   PWM3, PWM2_L=1
                PWMA_ENO = 0x10; PWM1_L=0; PWM3_L=0;  // 打开 C 相的高端 PWM
                Delay_500ns();
                PWM2_L = 1;                             // 打开 B 相的低端
                PWMB_CCER1 = (0x01+0x02);               //P2.0 0x01:允许输入捕获, +0x00:上升沿, +0x02: 下降沿
                PWMB_IER   = 0x02;                      //P2.0 使能中断
                break;

        default:
                break;
        }
    }
}


void PWMA_config(void)
{
    P_SW2 |= 0x80;          //SFR enable

    PWM1   = 0;
    PWM1_L = 0;
    PWM2   = 0;
    PWM2_L = 0;
    PWM3   = 0;
    PWM3_L = 0;
    P1n_push_pull(0x3f);

    PWMA_PSCRH = 0;                                     //预分频寄存器,分频 Fck_cnt = Fck_psc/(PSCR[15:0}+1),
    PWMA_PSCRL = 3;                                     //边沿对齐 PWM 频率 = SYSclk/((PSCR+1)*(AAR+1)),
                                                        //中央对齐 PWM 频率 = SYSclk/((PSCR+1)*AAR*2).
    PWMA_DTR   = 24;                                    // 死区时间配置, n=0~127: DTR= n T,
                                                        //0x80 ~(0x80+n), n=0~63: DTR=(64+n)*2T,
```

```
                                                //0xc0 ~(0xc0+n), n=0~31: DTR=(32+n)*8T,
                                                //0xE0 ~(0xE0+n), n=0~31: DTR=(32+n)*16T,
    PWMA_ARRH   = 0;                            // 自动重装载寄存器,  控制 PWM 周期
    PWMA_ARRL   = 255;
    PWMA_CCER1  = 0;
    PWMA_CCER2  = 0;
    PWMA_SR1    = 0;
    PWMA_SR2    = 0;
    PWMA_ENO    = 0;
    PWMA_PS     = 0;
    PWMA_IER    = 0;
//  PWMA_ISR_En = 0;

    PWMA_CCMR1  = 0x68;                         // 通道模式配置, PWM 模式 1,  预装载允许
    PWMA_CCR1H  = 0;                                // 比较值,  控制占空比(高电平时钟数)
    PWMA_CCR1L  = 0;
    PWMA_CCER1 |= 0x05;                         // 开启比较输出,  高电平有效
    PWMA_PS    |= 0;                            // 选择 IO, 0:P1.0 P1.1, 1:P2.0 P2.1, 2:P6.0 P6.1,
//  PWMA_IER   |= 0x02;                         // 使能中断

    PWMA_CCMR2  = 0x68;                         // 通道模式配置, PWM 模式 1,  预装载允许
    PWMA_CCR2H  = 0;                                // 比较值,  控制占空比(高电平时钟数)
    PWMA_CCR2L  = 0;
    PWMA_CCER1 |= 0x50;                         // 开启比较输出,  高电平有效
    PWMA_PS    |= (0<<2);                       // 选择 IO, 0:P1.2 P1.3, 1:P2.2 P2.3, 2:P6.2 P6.3,
//  PWMA_IER   |= 0x04;                         // 使能中断

    PWMA_CCMR3  = 0x68;                         // 通道模式配置, PWM 模式 1,  预装载允许
    PWMA_CCR3H  = 0;                                // 比较值,  控制占空比(高电平时钟数)
    PWMA_CCR3L  = 0;
    PWMA_CCER2 |= 0x05;                         // 开启比较输出,  高电平有效
    PWMA_PS    |= (0<<4);                       // 选择 IO, 0: P1.4 P1.5, 1: P2.4 P2.5, 2: P6.4 P6.5,
//  PWMA_IER   |= 0x08;                         // 使能中断

    PWMA_BKR    = 0x80;                         // 主输出使能  相当于总开关
    PWMA_CR1    = 0x81;                         // 使能计数器,  允许自动重装载寄存器缓冲,
                                                //边沿对齐模式,
                                                // 向上计数,   bit7=1:写自动重装载寄存器缓冲
                                                //(本周期不会被打扰),
                                                //=0:直接写自动重装载寄存器本(周期可能会乱掉)
    PWMA_EGR    = 0x01;                         //产生一次更新事件,  清除计数器和与分频计数器,
                                                //装载预分频寄存器的值
//  PWMA_ISR_En = PWMA_IER;                     //设置标志允许通道 1~4 中断处理
}


//========================================================================
// 函数: void PWMB_config(void)
// 描述: PPWM 配置函数。
// 参数: noe.
// 返回: none.
// 版本: V1.0
// 备注:
//========================================================================
void PWMB_config(void)
{
    P_SW2 |= 0x80;                              //SFR enable

    PWMB_PSCR = 11;                             //预分频寄存器,分频 Fck_cnt = Fck_psc/(PSCR[15:0]+1),
```

```
                                                    //边沿对齐 PWM 频率 = SYSclk/((PSCR+1)*(AAR+1)),
                                                    //中央对齐 PWM 频率 = SYSclk/((PSCR+1)*(AAR+1)*2).
    PWMB_DTR    = 0;                                //  死区时间配置, n=0~127: DTR= n T,
                                                    //0x80 ~(0x80+n), n=0~63: DTR=(64+n)*2T,
                                                    //0xc0 ~(0xc0+n), n=0~31: DTR=(32+n)*8T,
                                                    //0xE0 ~(0xE0+n), n=0~31: DTR=(32+n)*16T,
    PWMB_CCER1    = 0;
    PWMB_CCER2    = 0;
    PWMB_CR1      = 0;                              //  使能计数器, 允许自动重装载寄存器缓冲,
                                                    //边沿对齐模式, 向上计数,  bit7=1:写自动重装载
                                                    //寄存器缓冲(本周期不会被打扰), =0:直接写自动
                                                    //重装载寄存器本(周期可能会乱掉)
    PWMB_CR2      = 0;
    PWMB_SR1      = 0;
    PWMB_SR2      = 0;
    PWMB_ENO      = 0;
    PWMB_PS       = 0;
    PWMB_IER      = 0;

    PWMB_CCMR1    = 0x31;                           //  通道 5 模式配置, 配置成输入通道, 8 个时钟滤波
//  PWMB_CCER1 |= (0x01+0x02);                      // 0x01:允许输入捕获, +0x00:上升沿, +0x02: 下降沿
    PWMB_PS       |= 0;                             //  选择 IO, 0:P2.0, 1:P1.7, 2:P0.0, 3:P7.4,
//  PWMB_IER      |= 0x02;                          //  使能中断

    PWMB_CCMR2    = 0x31;                           //  通道 6 模式配置, 配置成输入通道, 8 个时钟滤波
//  PWMB_CCER1 |= (0x10+0x20);                      // 0x10:允许输入捕获, +0x00:上升沿, +0x20: 下降沿
    PWMB_PS       |= (0<<2);                        // 0:选择 P2.1, 1:选择 P5.4, 2:选择 P0.1, 3:选择 P7.5,
//  PWMB_IER      |= 0x04;                          //  使能中断

    PWMB_CCMR3    = 0x31;                           //  通道 7 模式配置, 配置成输入通道, 8 个时钟滤波
//  PWMB_CCER2 |= (0x01+0x02);                      // 0x01:允许输入捕获, +0x00:上升沿, +0x02: 下降沿
    PWMB_PS       |= (0<<4);                        //  选择 IO, 0:P2.2, 1:P3.3, 2:P0.2, 3:P7.6,
//  PWMB_IER      |= 0x08;                          //  使能中断

    PWMB_EGR      = 0x01;                           //产生一次更新事件, 清除计数器和预分频计数器,
                                                    //装载预分频寄存器的值

    PWMB_SMCR     = 0x60;
    PWMB_BKR      = 0x00;                           //主输出使能 相当于总开关
    PWMB_CR1      |= 0x01;                          //  使能计数器, 允许自动重装载寄存器缓冲,
                                                    //边沿对齐模式, 向上计数,  bit7=1:写自动重
                                                    //装载寄存器缓冲(本周期不会被打扰), =0:直接写自
                                                    //动重装载寄存器本(周期可能会乱掉)

//  P2n_standard(0x07);                             //通道 5 6 7 输出 IO 设置为准双向口
}


//========================================================================
//  函数: void PWMB_ISR(void) interrupt PWMB_VECTOR
//  描述: PWMB 中断处理程序. 捕获数据通过 TIM1-> CCRnH / TIM1-> CCRnL 读取
//  参数: None
//  返回: none.
//  版本: V1.0
//========================================================================
void PWMB_ISR(void) interrupt PWMB_VECTOR
{
    PWMB_SR1 = 0;                                   //清除中断标志
    PWMB_SR2 = 0;                                   //清除中断标志
```

```
    if(B_RUN)StepMotor();                       //换相
}


void ADC_config(void)                           //ADC 初始化函数
{
    P1n_pure_input(0xc0);                       //P1.7 P1.6 设置为高阻输入
    P0n_pure_input(0x0f);                       //P0.3~P0.0 设置为高阻输入
    ADC_CONTR = 0x80 + 6;                       //ADC on + channel
    ADCCFG = RES_FMT + ADC_SPEED;
    P_SW2 |=    0x80;                           //访问 XSFR
    ADCTIM = CSSETUP + CSHOLD + SMPDUTY;
}


void Timer0_config(void)                        //Timer0 初始化函数
{
    Timer0_16bitAutoReload();                   // T0 工作于 16 位自动重装
    Timer0_12T();
    TH0 = (65536UL-MAIN_Fosc/12 / 250) / 256;     //4ms
    TL0 = (65536UL-MAIN_Fosc/12 / 250) % 256;
    TR0 = 1;                                    //  打开定时器 0

    ET0 = 1;                                    //  允许 ET0 中断
}

void Timer0_ISR(void) interrupt 1               //Timer0 中断函数
{
    B_4ms = 1;                                  //4ms 定时标志
//  if(B_RUN)StepMotor();                       //换相, 增加定时器里换相可以保证启动成功.
}


/*********************************************/
void main(void)
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间


    P2n_standard(0xf8);
    P3n_standard(0xbf);
    P5n_standard(0x10);


    PWMA_config();
    PWMB_config();
    ADC_config();
    Timer0_config();      //Timer0 初始化函数
    PWW_Set = 0;

    EA   = 1; //  打开总中断
```

```
    while (1)
    {
        if(B_4ms)       // 4ms 时隙
        {
            B_4ms = 0;

            YouMen   = (u8)(Get_ADC10bitResult(11) >> 2);  //油门是 8 位的, P0.3 ADC11-->控制电位器输入
            if(YouMen >= 128) PWW_Set = YouMen - 128, B_direct = 0;  //顺时针
            else   PWW_Set = 127 - YouMen, B_direct = 1;     //逆时针
            PWW_Set *= 2;                                    //PWM 设置值 0~254

            if(!B_RUN && (PWW_Set >= 30))                    // PWM_Set >= 30, 并且电机未运行, 则启动电机
            {
                PWM_Value = 30;                              //启动电机的最低 PWM, 根据具体电机而定
                PWMA_CCR1L = PWM_Value;                      //输出 PWM
                PWMA_CCR2L = PWM_Value;
                PWMA_CCR3L = PWM_Value;
                B_RUN = 1;                                   //标注运行
                StepMotor();                                 //启动换相
            }

            if(B_RUN)//正在运行中
            {
                if(PWM_Value < PWW_Set) PWM_Value++;//油门跟随电位器, 调速柔和
                if(PWM_Value > PWW_Set) PWM_Value--;
                if(PWM_Value < 20)                           // 停转
                {
                    B_RUN = 0;
                    PWMB_IER  = 0;
                    PWMB_CCER1 = 0;
                    PWMB_CCER2 = 0;
                    PWM_Value = 0;
                    PWMA_ENO  = 0;
                    PWMA_CCR1L = 0;
                    PWMA_CCR2L = 0;
                    PWMA_CCR3L = 0;
                    PWM1_L=0;
                    PWM2_L=0;
                    PWM3_L=0;
                }
                else
                {
                    PWMA_CCR1L = PWM_Value;
                    PWMA_CCR2L = PWM_Value;
                    PWMA_CCR3L = PWM_Value;
                }
            }
        }

    }
}
```

30 高级 PWM-硬件移相

产品线	高级 PWM 新增硬件移相功能
Ai8051U 系列	●

30.1 高级 PWM 硬件移相功能脚切换

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_PS2	7EFEB8H	-		-		AC6PS[1:0]		AC5PS[1:0]	

AC5PS[1:0]：高级 PWMA 通道 5 输出脚选择位

AC5PS[1:0]	PWMAPS5
00	P3.3
01	P4.4
10	P5.6
11	-

AC6PS[1:0]：高级 PWMA 通道 6 输出脚选择位

AC6PS[1:0]	PWMAPS6
00	P3.4
01	P4.7
10	P5.7
11	-

30.2 相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
PWMA_ENO2	PWMA 输出使能寄存器 2	7EF930H	-	-	-	-	-	ENO6	-	ENO5	xxxx,x0x0
PWMA_IOAUX2	PWMA 输出附加寄存器 2	7EF931H	-	-	-	-	-	AUX6	-	AUX6	xxxx,x0x0
PWMA_CR3	PWMA 控制寄存器 3	7EF932H	MMS2[3:0]				-	OIS6	-	OIS5	0000,x0x0
PWMA_SR3	PWMA 状态寄存器 3	7EF933H	-	-	-	-	-	-	CC6IF	CC5IF	xxxx,xx00
PWMA_CCER3	PWMA 捕获比较使能寄存器 3	7EF934H	-	-	CC6P	CC6E	-	-	CC5P	CC5E	xx00,xx00
PWMA_CCMR1X	PWMA 捕获比较模式寄存器 1x	7EF938H	-	-	-	-	-	-	-	OC1M[3]	xxx,xxx0
PWMA_CCMR2X	PWMA 捕获比较模式寄存器 2x	7EF939H	-	-	-	-	-	-	-	OC2M[3]	xxx,xxx0
PWMA_CCMR3X	PWMA 捕获比较模式寄存器 3x	7EF93AH	-	-	-	-	-	-	-	OC3M[3]	xxx,xxx0
PWMA_CCMR4X	PWMA 捕获比较模式寄存器 4x	7EF93BH	-	-	-	-	-	-	-	OC4M[3]	0000,0000
PWMA_CCMR5	PWMA 捕获比较模式寄存器 5	7EF93CH	OC5CE	OC5M[2:0]			OC5PE	-	-	-	0000,0xxx
PWMA_CCMR5X	PWMA 捕获比较模式寄存器 5x	7EF93DH	-	-	-	-	-	-	-	OC5M[3]	xxxx,xxx0
PWMA_CCMR6	PWMA 捕获比较模式寄存器 6	7EF93EH	OC6CE	OC6M[2:0]			OC6PE	-	-	-	000x,xxxx
PWMA_CCMR6X	PWMA 捕获比较模式寄存器 6x	7EF93FH	-	-	-	-	-	-	-	OC6M[3]	xxx,xxx0
PWMA_CCR5H	PWMA 捕获比较寄存器 5	7EF940H	CCR5[15:8]								0000,0000
PWMA_CCR5L	PWMA 捕获比较寄存器 5	7EF941H	CCR5[7:0]								0000,0000
PWMA_CCR5X	PWMA 捕获比较寄存器 5	7EF942H	GC5C3	GC5C2	GC5C1	-	-	-	-	-	000x,xxx
PWMA_CCR6H	PWMA 捕获比较寄存器 6	7EF943H	CCR6[15:8]								0000,0000
PWMA_CCR6L	PWMA 捕获比较寄存器 6	7EF944H	CCR6[7:0]								0000,0000

30.2.1 输出使能寄存器 2（PWMA_ENO2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ENO2	7EF930H	-	-	-	-	-	ENO6	-	ENO5

ENO6：PWMAPS6 输出控制位

0：禁止 PWMAPS6 输出

1：使能 PWMAPS6 输出

ENO5：PWMAPS5 输出控制位

0：禁止 PWMAPS5 输出

1：使能 PWMAPS5 输出

30.2.2 输出附加使能寄存器 2（PWMA_IOAUX2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_IOAUX2	7EF931H	-	-	-	-	-	AUX6	-	AUX5

AUX6：PWMAPS6 输出附加控制位

0：PWMAPS6 的输出直接由 ENO6 控制

1：PWMAPS6 的输出由 ENO6 和 PWMA_BKR 共同控制

AUX5：PWMAPS5 输出附加控制位

0：PWMAPS5 的输出直接由 ENO5 控制

1：PWMAPS5 的输出由 ENO5 和 PWMA_BKR 共同控制

30.2.3 控制寄存器 3（PWMA_CR3）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CR3	7EF932H	MMS[3:0]				-	OIS6	-	OIS5

MMS[3:0]：主模式选择 2

MMS[3:0]	模式	说明
0000	复位	PWMA_EGR寄存器的UG位被用于作为触发输出（TRGO2）。如果复位由触发输入生成（从模式控制器配置为复位模式），则TRGO2上的信号相比实际复位会有延迟。
0001	使能	计数器使能信号CNT_EN用作触发输出(TRGO2)。该触发输出可用于同时启动多个定时器，或者控制在一段时间内使能从定时器。计数器使能信号由CEN控制位与门控模式下的触发输入的逻辑或运算组合而成。当计数器使能信号由触发输入控制时，TRGO2上会存在延迟，选择主/从模式时除外（请参见PWMA_SMCR寄存器中MSM位的说明）。
0010	更新	更新事件被选为触发输出（TRGO2）
0011	比较脉冲	CC1IF 标志置1时（即使已为高），只要发生捕获或比较匹配，触发输出(TRGO2)都会发送一个正脉冲。
0100	比较	OC1REF信号用作触发输出（TRGO2）
0101	比较	OC2REF信号用作触发输出（TRGO2）
0110	比较	OC3REF信号用作触发输出（TRGO2）
0111	比较	OC4REF信号用作触发输出（TRGO2）
1000	比较	OC5REF信号用作触发输出（TRGO2）
1001	比较	OC6REF信号用作触发输出（TRGO2）
1010	比较脉冲	OC4REF 上升沿或下降沿时，TRGO2 上生成脉冲
1011	比较脉冲	OC6REF 上升沿或下降沿时，TRGO2 上生成脉冲
1100	比较脉冲	OC4REF 或 OC6REF 上升沿时，TRGO2 上生成脉冲
1101	比较脉冲	OC4REF 上升沿或 OC6REF 下降沿时，TRGO2 上生成脉冲
1110	比较脉冲	OC5REF 或 OC6REF 上升沿时，TRGO2 上生成脉冲
1111	比较脉冲	OC5REF 上升沿或 OC6REF 下降沿时，TRGO2 上生成脉冲

注：必须先使能从定时器或 ADC 的时钟，才能从主定时器接收事件；并且从主定时器接收触发信号时，不得实时更改从定时器或 ADC 的时钟。

OIS6：空闲状态时 OC6 输出电平（请参见 OIS1 位）

OIS5：空闲状态时 OC5 输出电平（请参见 OIS1 位）

30.2.4 状态寄存器 3(PWMA_SR3)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_SR1	7EF933H	-	-	-	-	-	-	CC6IF	CC5IF

CC6IF：比较6中断标记，参考CC1IF描述（注意：通道6只能配置为输出）

CC5IF：比较5中断标记，参考CC1IF描述（注意：通道5只能配置为输出）

30.2.5 捕获/比较使能寄存器 3（PWMA_CCER3）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCER3	7EF934H	-	-	CC6P	CC6E	-	-	CC5P	CC5E

CC6P：OC6 输入捕获/比较输出极性。参考 CC1P

CC6E：OC6 输入捕获/比较输出使能。参考 CC1E

CC5P：OC5 输入捕获/比较输出极性。参考 CC1P

CC5E：OC5 输入捕获/比较输出使能。参考 CC1E

30.2.6 捕获/比较模式扩展寄存器 1（PWMA_CCMR1X）

通道可用于捕获输入模式或比较输出模式，通道的方向由相应的 CCnS 位定义。该寄存器其它位的作用在输入和输出模式下不同。OCxx 描述了通道在输出模式下的功能，ICxx 描述了通道在输入模式下的功能。因此必须注意，同一个位在输出模式和输入模式下的功能是不同的。

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR1X	7EF938H	-	-	-	-	-	-	-	OC1M[3]

OC1M[3]：和 PWMA_CCMR1 中的 OC1M[2:0]合并为 4 位对输出模式进行扩展

OCnM[3:0]	模式	说明
0000	冻结	PWMn_CCR1与PWMn_CNT间的比较对OCnREF不起作用
0001	匹配时设置通道n的输出为有效电平	当PWMn_CCR1=PWMn_CNT时，OCnREF输出高
0010	匹配时设置通道n的输出为无效电平	当PWMn_CCR1=PWMn_CNT时，OCnREF输出低
0011	翻转	当PWMn_CCR1=PWMn_CNT时，翻转OCnREF
0100	强制为无效电平	强制OCnREF为低
0101	强制为有效电平	强制OCnREF为高
0110	PWM模式1	在向上计数时，当PWMn_CNT<PWMn_CCR1时OCnREF输出高，否则OCnREF输出低 在向下计数时，当PWMn_CNT>PWMn_CCR1时OCnREF输出低，否则OCnREF输出高
0111	PWM模式2	在向上计数时，当PWMn_CNT<PWMn_CCR1时OCnREF输出低，否则OCnREF输出高 在向下计数时，当PWMn_CNT>PWMn_CCR1时OCnREF输出高，否则OCnREF输出低
1000	可再触发OPM模式1	在递增计数模式下，通道为有效状态，直至（在TRGI信号上）检测到触发事件。然后，在PWM模式1下进行比较，通道会在下一次更新时再次变为有效状态。在递减计数模式下，通道为无效状态，直至（在TRGI 信号上）检测到触发事件。然后，在PWM模式1下进行比较，通道会在下一次更新时再次变为无效状态。
1001	可再触发OPM模式2	在递增计数模式下，通道为无效状态，直至（在TRGI信号上）检测到触发事件。然后，在PWM模式2下进行比较，通道会在下一次更新时再次变为无效状态。在递减计数模式下，通道为有效状态，直至（在TRGI信号上）检测到触发事件。然后，在PWM模式2

		下进行比较，通道会在下一次更新时再次变为有效状态。
1010	保留	
1011	保留	
1100	组合PWM模式1	OC1REF与在PWM模式1下的行为相同。OC1REFC是OC1REF 和OC2REF的逻辑或运算结果。
1101	组合PWM模式2	OC1REF与在PWM模式2下的行为相同。OC1REFC是OC1REF 和OC2REF的逻辑与运算结果。
1110	不对称PWM模式1	OC1REF与在PWM模式1下的行为相同。计数器递增计数时，OC1REFC输出OC1REF；计数器递减计数时，OC1REFC 输出OC2REF。
1111	不对称PWM模式2	OC1REF与在PWM模式2下的行为相同。计数器递增计数时，OC1REFC输出OC1REF；计数器递减计数时，OC1REFC 输出OC2REF。

注 1：一旦 LOCK 级别设为 3（PWMn_BKR 寄存器中的 LOCK 位）并且 CCnS=00（该通道配置成输出）则该位不能被修改。

注 2：在 PWM 模式 1 或 PWM 模式 2 中，只有当比较结果改变了或在输出比较模式中从冻结模式切换到 PWM 模式时，OCnREF 电平才改变。

注 3：在有互补输出的通道上，这些位是预装载的。如果 PWMn_CR2 寄存器的 CCPC=1，OCM 位只有在 COM 事件发生时，才从预装载位取新值。

30.2.7 捕获/比较模式扩展寄存器 2（PWMA_CCMR2X）

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR2X	7EF939H	-	-	-	-	-	-	-	OC2M[3]

OC2M[3]：和 PWMA_CCMR2 中的 OC2M[2:0]合并为 4 位对输出模式进行扩展，详情参考 OC1M。

30.2.8 捕获/比较模式扩展寄存器 3（PWMA_CCMR3X）

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR3X	7EF93AH	-	-	-	-	-	-	-	OC3M[3]

OC3M[3]：和 PWMA_CCMR3 中的 OC3M[2:0]合并为 4 位对输出模式进行扩展，详情参考 OC1M。

30.2.9 捕获/比较模式扩展寄存器 4（PWMA_CCMR4X）

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR4X	7EF93BH	-	-	-	-	-	-	-	OC4M[3]

OC4M[3]：和 PWMA_CCMR4 中的 OC4M[2:0]合并为 4 位对输出模式进行扩展，详情参考 OC1M。

30.2.10 捕获/比较模式寄存器 5（PWMx_CCMR5）

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR5	7EF93CH	OC5CE	OC5M[2:0]			OC5PE	-	-	-

OC5CE：输出比较 5 清零使能。该位用于使能使用 PWMETI 引脚上的外部事件来清通道 5 的输出信号（OC5REF）

0：OC5REF 不受 ETRF 输入的影响；

1：一旦检测到 ETRF 输入高电平，OC5REF=0。

OC5M[2:0]：输出比较 5 模式，参考 OC1M。

OC5PE：输出比较 5 预装载使能，参考 OP5PE。

30.2.11 捕获/比较模式扩展寄存器 5（PWMA_CCMR5X）

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR5X	7EF93DH	-	-	-	-	-	-	-	OC5M[3]

OC5M[3]：和 PWMA_CCMR5 中的 OC5M[2:0]合并为 4 位对输出模式进行扩展，详情参考 OC1M。

30.2.12 捕获/比较模式寄存器 6（PWMx_CCMR6）

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR6	7EF93EH	OC6CE	OC6M[2:0]			OC6PE	-	-	-

OC6CE：输出比较 6 清零使能。该位用于使能使用 PWMETI 引脚上的外部事件来清通道 6 的输出信号（OC6REF）

0：OC6REF 不受 ETRF 输入的影响；

1：一旦检测到 ETRF 输入高电平，OC6REF=0。

OC6M[2:0]：输出比较 6 模式，参考 OC1M。

OC6PE：输出比较 6 预装载使能，参考 OP6PE。

30.2.13 捕获/比较模式扩展寄存器 6（PWMA_CCMR6X）

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR6X	7EF93FH	-	-	-	-	-	-	-	OC6M[3]

OC6M[3]：和 PWMA_CCMR6 中的 OC6M[2:0]合并为 4 位对输出模式进行扩展，详情参考 OC1M。

30.2.14 捕获/比较寄存器 5 高 8 位（PWMA_CCR5H）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR5H	7EF940H	CCR5[15:8]							

CCR5[15:8]：捕获/比较 5 的高 8 位值

CCR5 包含了装入当前比较值（预装载值）。如果在 PWM5_CCMR1 寄存器（OC5PE 位）中未选择预装载功能，写入的数值会立即传输至当前寄存器中。否则只有当更新事件发生时，此预装载值才传输至当前捕获/比较 5 寄存器中。当前比较值同计数器 PWM5_CNT 的值相比较，并在 OC5 端口上产生输出信号。

30.2.15 捕获/比较寄存器 5 低 8 位（PWMA_CCR5L）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR5L	7EF941H	CCR5[7:0]							

CCR5[7:0]：捕获/比较 5 的低 8 位值

30.2.16 捕获/比较扩展寄存器 5（PWMA_CCR5X）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR5X	7EF942H	GC5C3	GC5C2	GC5C1	-	-	-	-	-

GC5C3：通道 5 和通道 3 组（Group Channel 5 and Channel 3）。通道 3 输出上失真：

0：OC5REF 对 OC3REFC 无影响；

1：OC3REFC 是 OC3REFC 和 OC5REF 的逻辑与运算结果。该位可以立即生效，也可预装载并在更新事件后执行（如果在 PWMA_CCMR3 中选择了预装载功能）。

注：也可在组合 PWM 信号上应用此失真。

GC5C2：通道 5 和通道 2 组（Group Channel 5 and Channel 2）。通道 2 输出上失真：

0：OC5REF 对 OC2REFC 无影响；

1：OC2REFC 是 OC2REFC 和 OC5REF 的逻辑与运算结果。该位可以立即生效，也可预装载并在更新事件后执行（如果在 PWMA_CCMR2 中选择了预装载功能）。

注：也可在组合 PWM 信号上应用此失真。

GC5C1：通道 5 和通道 1 组（Group Channel 5 and Channel 1）。通道 1 输出上失真：

0：OC5REF 对 OC1REFC 无影响；

1：OC1REFC 是 OC1REFC 和 OC5REF 的逻辑与运算结果。该位可以立即生效，也可预装载并在更新事件后执行（如果在 PWMA_CCMR1 中选择了预装载功能）。

注：也可在组合 PWM 信号上应用此失真。

30.2.17 捕获/比较寄存器 6 高 8 位（PWMA_CCR6H）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR6H	7EF943H	CCR6[15:8]							

CCR6[15:8]：捕获/比较 6 的高 8 位值

CCR6 包含了装入当前比较值（预装载值）。如果在 PWM6_CCMR1 寄存器（OC6PE 位）中未选择预装载功能，写入的数值会立即传输至当前寄存器中。否则只有当更新事件发生时，此预装载值才传输至当前捕获/比较 6 寄存器中。当前比较值同计数器 PWM6_CNT 的值相比较，并在 OC6 端口上产生输出信号。

30.2.18 捕获/比较寄存器 6 低 8 位（PWMA_CCR6L）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR6L	7EF944H	CCR6[7:0]							

CCR6[7:0]：捕获/比较 6 的低 8 位值

30.3 移相 PWM 输出模式

30.3.1 不对称 PWM 模式

在不对称模式下，生成的两个中心对齐 PWM 信号间允许存在可编程相移。频率由 PWMA_ARR 寄存器的值确定，而占空比和相移则由一对 PWMA_CCRx 寄存器确定。两个寄存器分别控制递增计数和递减计数期间的 PWM，这样每半个 PWM 周期便会调节一次 PWM：

- OC1REFC（或 OC2REFC）由 PWMA_CCR1 和 PWMA_CCR2 控制
- OC3REFC（或 OC4REFC）由 PWMA_CCR3 和 PWMA_CCR4 控制

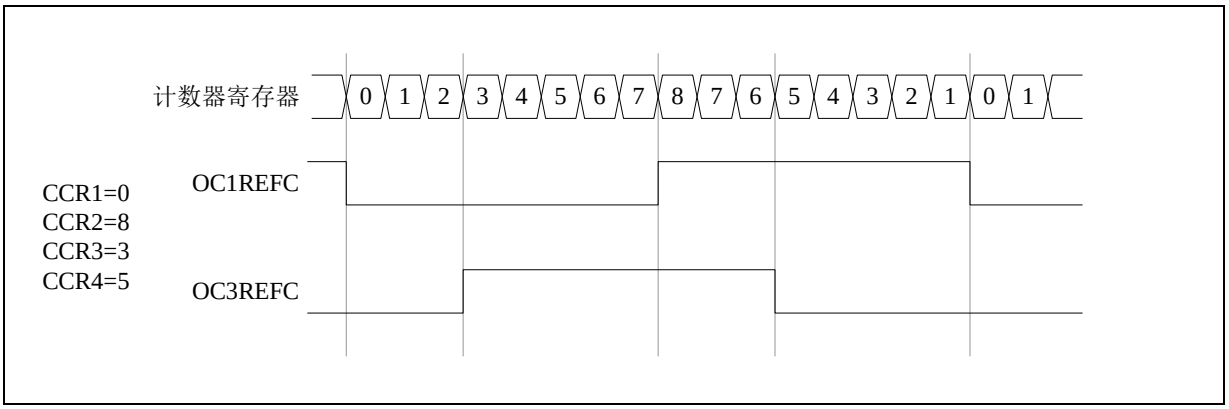
两个通道可以独立选择不对称 PWM 模式（每对 CCR 寄存器控制一个 OCx 输出），只需向 PWMA_CCMRx 寄存器的 OCxM 位写入“1110”（不对称 PWM 模式 1）或“1111”（不对称 PWM 模式 2）。

给定通道用作不对称 PWM 通道时，也可使用其互补通道。例如，如果通道 1 上产生 OC1REFC 信号（不对称 PWM 模式 1），则由于不对称 PWM 模式 1 的原因，通道 2 上可输出 OC2REF 信号或 OC2REFC 信号。

（注：出于兼容性原因，OCxM[3:0]位域分为两部分，最高有效位与最低有效的 3 位不相邻。）

下图显示了不对称 PWM 模式下可以产生的信号示例（通道 1 到通道 4 在不对称 PWM 模式 1 下配置）。与死区发生器配合使用时，这可控制相移全桥直流到直流转换器。

（产生 2 个 50%占空比的相移 PWM 信号）



30.3.2　组合 PWM 模式

在组合 PWM 模式下，生成的两个边沿或中心对齐 PWM 信号的各个脉冲间允许存在可编程延时和相移。频率由 PWMA_ARR 寄存器的值确定，而占空比和延时则由两个 PWMA_CCRx 寄存器确定。产生的信号 OCxREFC 由两个参考 PWM 的逻辑或运算或者逻辑与运算组合组成。

- OC1REFC（或 OC2REFC）由 PWMA_CCR1 和 PWMA_CCR2 控制
- OC3REFC（或 OC4REFC）由 PWMA_CCR3 和 PWMA_CCR4 控制

两个通道可以独立选择组合 PWM 模式（每对 CCR 寄存器控制一个 OCx 输出），只需向 PWMA_CCMRx 寄存器的 OCxM 位写入“1100”（组合 PWM 模式 1）或“1101”（组合 PWM 模式 2）。

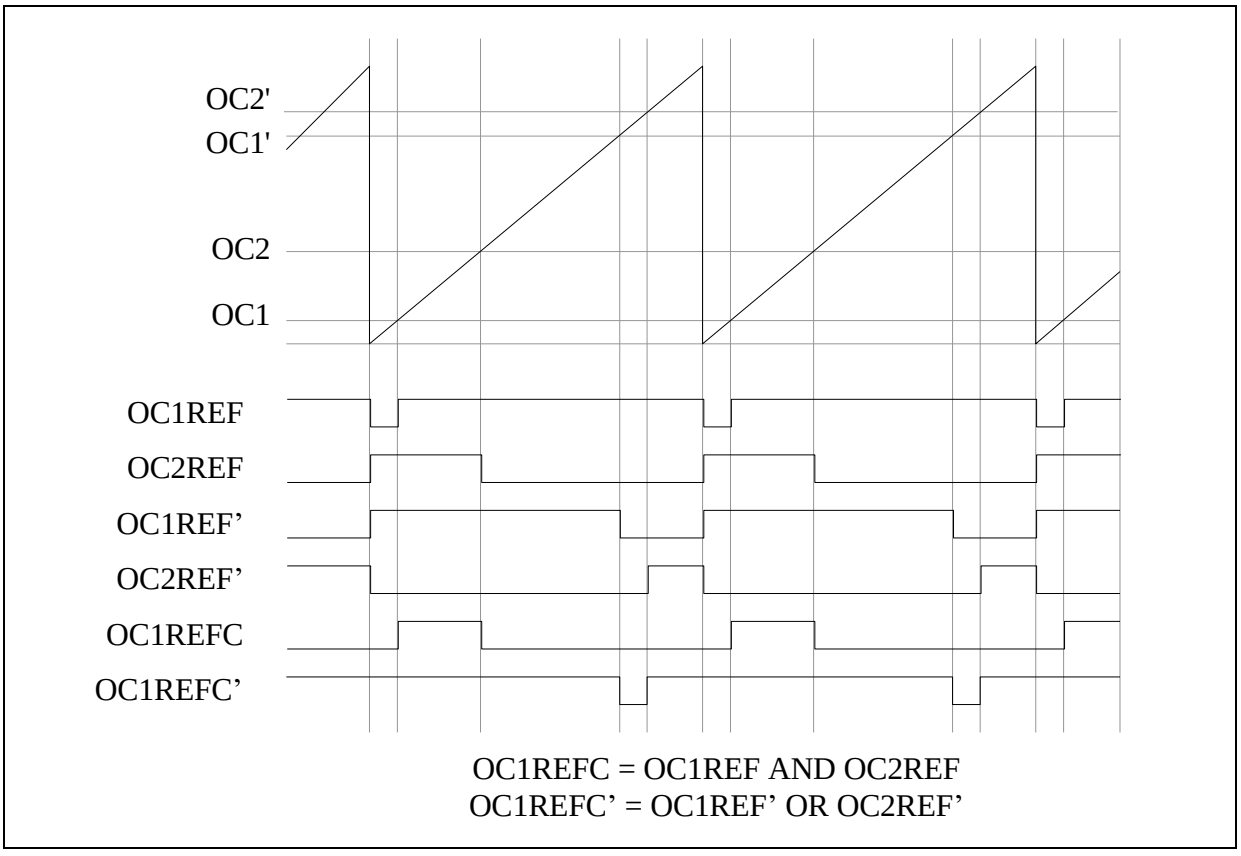
当给定通道用作组合 PWM 通道时，其互补通道必须在相反的 PWM 模式下配置。例如，一个通道在组合 PWM 模式 1 下配置，另一个通道在组合 PWM 模式 2 下配置。

（注：出于兼容性原因，OCxM[3:0]位域分为两部分，最高有效位与最低有效的 3 位不相邻。）

下图显示了不对称 PWM 模式下可以产生的信号示例，通过以下配置可获得这些信号：

- 通道 1 在组合 PWM 模式 2 下配置。
- 通道 2 在 PWM 模式 1 下配置。
- 通道 3 在组合 PWM 模式 2 下配置。
- 通道 4 在 PWM 模式 1 下配置。

（通道 1 和通道 3 上的组合 PWM 模式）



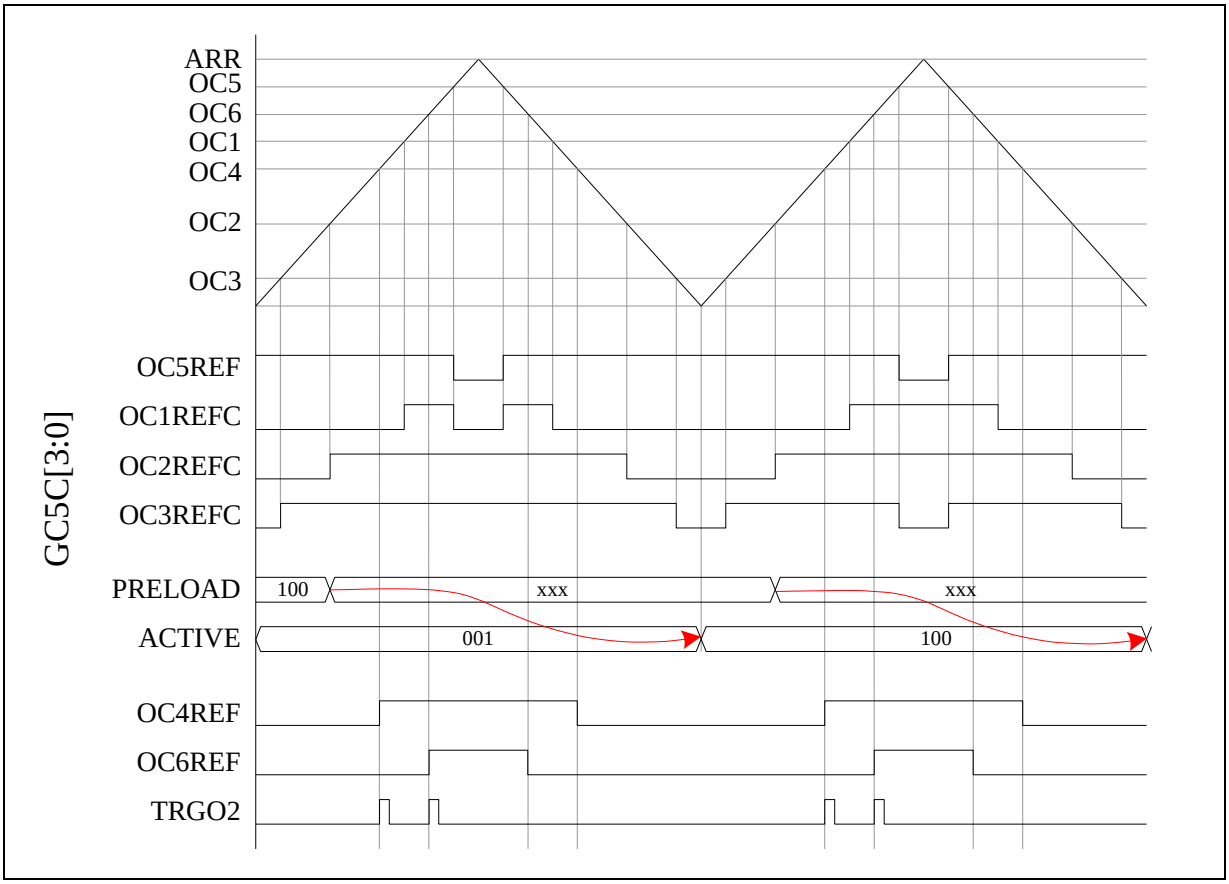
30.3.3　组合三相 PWM 模式

在组合三相 PWM 模式下，产生的一至三个中心对齐 PWM 信号与一个可编程信号间允许在脉冲中间进行逻辑与运算。OC5REF 信号用于定义产生的组合信号。凭借 PWMA_CCR5 中的 3 位 GC5C[3:1]，可以选择 OC5REF 与哪个参考信号组合。产生的信号 OCxREFC 由两个参考 PWM 的逻辑与运算组合组成。

- 如果 GC5C1 置 1，则 OC1REFC 由 PWMA_CCR1 和 PWMA_CCR5 控制
- 如果 GC5C2 置 1，则 OC2REFC 由 PWMA_CCR2 和 PWMA_CCR5 控制
- 如果 GC5C3 置 1，则 OC3REFC 由 PWMA_CCR3 和 PWMA_CCR5 控制

通道 1 到通道 3 可独立选择组合三相 PWM 模式，只需将 3 位 GC5C[3:1]中的至少一位置 1。

（三相组合 PWM 信号，每个周期多个触发脉冲）



（TRGO2 波形说明了如何根据给定的三相 PWM 信号同步 ADC）

30.4 PWM 硬件移相范例程序

请参考论坛如下的帖子

https://www.stcaimcu.com/forum.php?mod=viewthread&tid=4707&page=3#pid50750

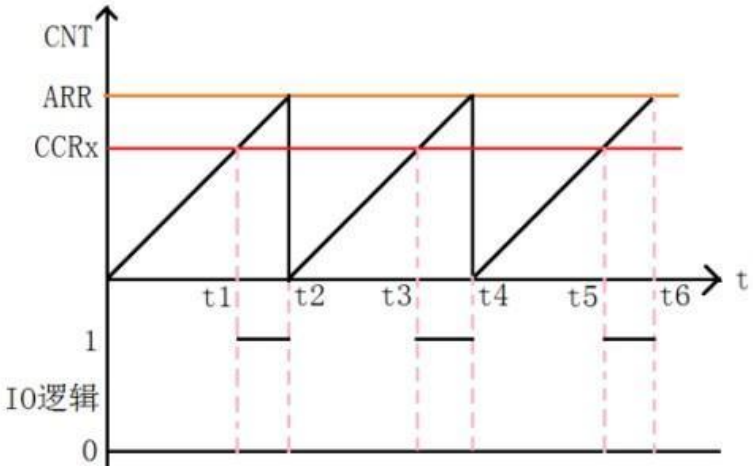
30.5 利用不对称 PWM 实现高速正交编码信号输出（热心网友冲哥提供）

前言：

众所周知，早早的就出了带硬件移相的 PWM，但是好多人还不理解这个硬件移相 PWM 有什么作用，这篇帖子带大家仔细研究下这个硬件移相 PWM 的一部分——不对称 PWM。

一、普通 PWM 模式

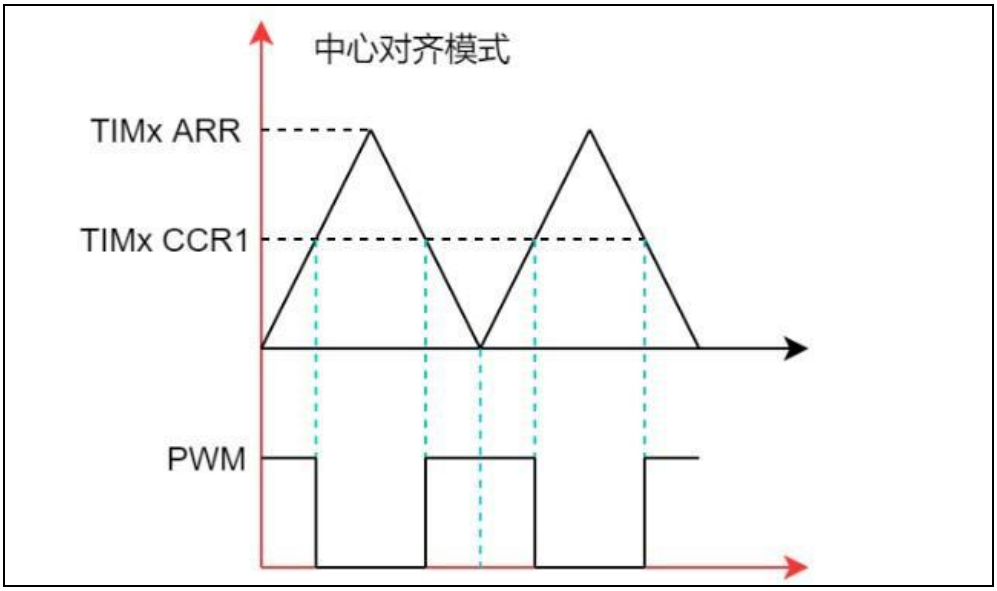
常见的 PWM 模式如下图所示，通过一个 ARR 寄存器设定最大计数值（向上计数模式），CNT 表示当前的计数，当计数超过 ARR 的时候被清 0，当 CNT 数值小于当前通道的 CCRx 的时候输出低电平，反之输出高电平，这个是常规的 PWM 模式 2（不开启极性反相的时候！）。



再来看下手册这部分，可以看到 PWM 模式 1 其实就是 PWM 模式 2 的反过来的波形

110	PWM模式1	在向上计数时，当PWMn_CNT<PWMn_CCR1时 OCnREF输出高，否则OCnREF输出低 在向下计数时，当PWMn_CNT>PWMn_CCR1时 OCnREF输出低，否则OCnREF输出高
111	PWM模式2	在向上计数时，当PWMn_CNT<PWMn_CCR1时 OCnREF输出低，否则OCnREF输出高 在向下计数时，当PWMn_CNT>PWMn_CCR1时 OCnREF输出高，否则OCnREF输出低

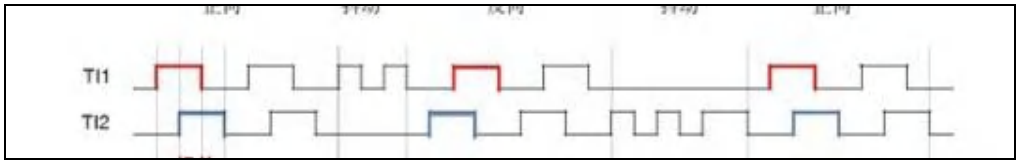
当然上面的都是边沿计数，还有一种中心对齐的计数



但是这种模式就是只能生成中心对称的波形，如果单纯的只想做几路不同占空比输出的波形，那确实没一点问题。可以思考下，像上面的几种模式，总的规划出来就分为四大类：

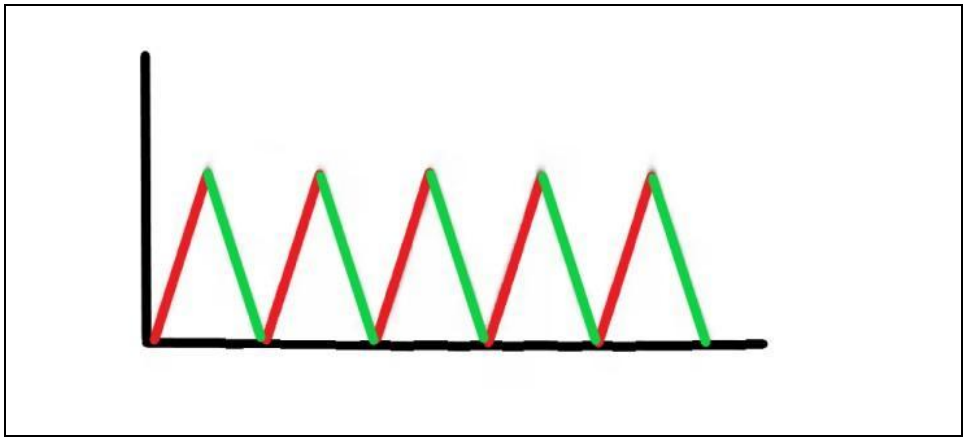
1.前面高电平后面低电平
2.前面低电平后面高电平
3.两边低电平中间高电平
4.两边高电平中间低电平

上面 1 和 2 是同一种计数方式，3 和 4 是一种计数方式，同一时间只能选择一种计数方式。但是这时候，我们如果想要生成一个如下图的正交编码信号的 PWM 波形该如何实现的？

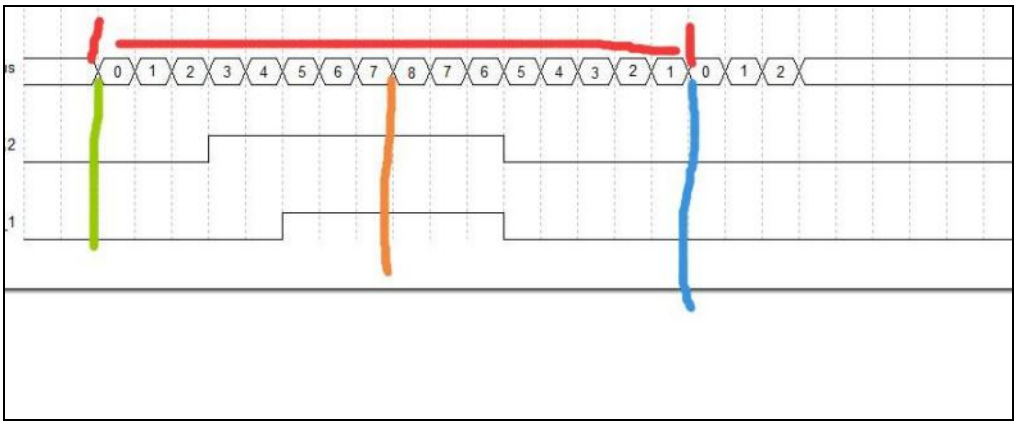


二、不对称互补 PWM

按照我个人的理解，不对称 PWM 的本质就是起始和结束电平不受约束，按照手册的说法他必须使用中心对齐计数模式，开始电平是可以在上升计数段里任意定义，结束电平是在下降计数段里任意定义的。在下面的图里，一个红色线和一个绿色线为一个中心计数模式的一个周期（从 0 计数到最大值 arr 再到 0 的过程），开始电平可以在红线里的任意位置，结束电平可以是绿线的任意位置，这样是不是就很好分配了。

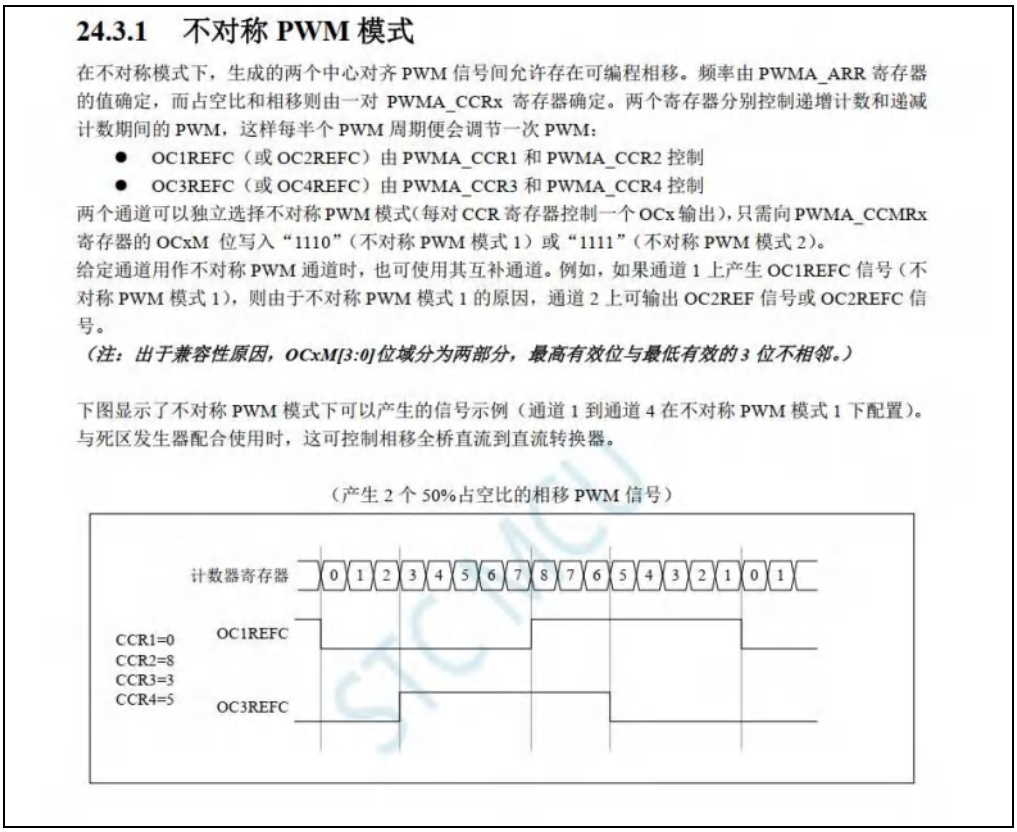


再画个简单的示意图：



假设红色的是整个计数周期，将他对半的分为绿-橙区域和橙-蓝区域，开始的电平可以在这个绿-橙区域任意位置，结束电平也可以定义在这个橙-蓝区域的任意位置，此时你想要这个波形横移多少位置不就是分分钟的了。

当然在开始之前我们还需要了解一些基本知识：



24.3.1　不对称 PWM 模式

在不对称模式下，生成的两个中心对齐 PWM 信号间允许存在可编程相移。频率由 PWMA_ARR 寄存器的值确定，而占空比和相移则由一对 PWMA_CCRx 寄存器确定。两个寄存器分别控制递增计数和递减计数期间的 PWM，这样每半个 PWM 周期便会调节一次 PWM：

- OC1REFC（或 OC2REFC）由 PWMA_CCR1 和 PWMA_CCR2 控制
- OC3REFC（或 OC4REFC）由 PWMA_CCR3 和 PWMA_CCR4 控制

两个通道可以独立选择不对称 PWM 模式(每对 CCR 寄存器控制一个 OCx 输出)，只需向 PWMA_CCMRx 寄存器的 OCxM 位写入“1110”（不对称 PWM 模式 1）或“1111”（不对称 PWM 模式 2）。

给定通道用作不对称 PWM 通道时，也可使用其互补通道。例如，如果通道 1 上产生 OC1REFC 信号（不对称 PWM 模式 1），则由于不对称 PWM 模式 1 的原因，通道 2 上可输出 OC2REF 信号或 OC2REFC 信号。

（注：出于兼容性原因，OCxM[3:0]位域分为两部分，最高有效位与最低有效的 3 位不相邻。）

下图显示了不对称 PWM 模式下可以产生的信号示例（通道 1 到通道 4 在不对称 PWM 模式 1 下配置）。与死区发生器配合使用时，这可控制相移全桥直流到直流转换器。

（产生 2 个 50%占空比的相移 PWM 信号）

手册上关于这个模式的描述就这么一页，我给他归纳总结了一下：

1.需要在中心对齐计数模式下使用

2.必须手动设置为不对称模式才能使用这个输出（**切记这个寄存器的位置不连续，要重点注意**）

3.两个通道才能组合成为一个完整的不对称输出通道，且通道 1（又名 OC1REFC 或 OC2REFC）由 CCR1 和 CCR2 控制，通道 2（又名 OC3REFC 或 OC4REFC）由 CCR3 和 CCR4 控制。

4.1P 端口设置为不对称输出时起始电平时间点为 CCR1,结束时间点为 CCR2;　2P 端口设置为不对称输出时起始电平时间点为 CCR2,结束时间点为 CCR1;

5.别的配置都可以参考普通 PWM 的配置

6.一个通道会占用两个端口的 CCR 寄存器，其中一个作为不对称输出的时候，另一个还可以作为普通的 pwm 模式输出。

三、寄存器描述

1.设置为中心对齐计数

STC8H 系列技术手册　官方网站: www.STCAI.com　车规 MCU 设计公司　技术支持: 19864885985　选型顾问: 13922805190

23.9.4　控制寄存器 1（PWMx_CR1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CR1	FEC0H	ARPEA	CMSA[1:0]		DIRA	OPMA	URSA	UDISA	CENA
PWMB_CR1	FEE0H	ARPEB	CMSB[1:0]		DIRB	OPMB	URSB	UDISB	CENB

ARPEn：自动预装载允许位（n=A,B）

0：PWMn_ARR 寄存器没有缓冲，它可以被直接写入

1：PWMn_ARR 寄存器由预装载缓冲器缓冲

CMSn[1:0]：选择对齐模式（n= A,B）

CMSn[1:0]	对齐模式	说明
00	边沿对齐模式	计数器依据方向位（DIR）向上或向下计数
01	中央对齐模式1	计数器交替地向上和向下计数。 配置为输出的通道的输出比较中断标志位(CCnIF)只在计数器向下计数时被置1。
10	中央对齐模式2	计数器交替地向上和向下计数。 配置为输出的通道的输出比较中断标志位(CCnIF)只在计数器向上计数时被置1。
11	中央对齐模式3	计数器交替地向上和向下计数。 配置为输出的通道的输出比较中断标志位(CCnIF)在计数器向上和向下计数时均被置1。

注 1：在计数器开启时（CEN=1），不允许从边沿对齐模式转换到中央对齐模式。

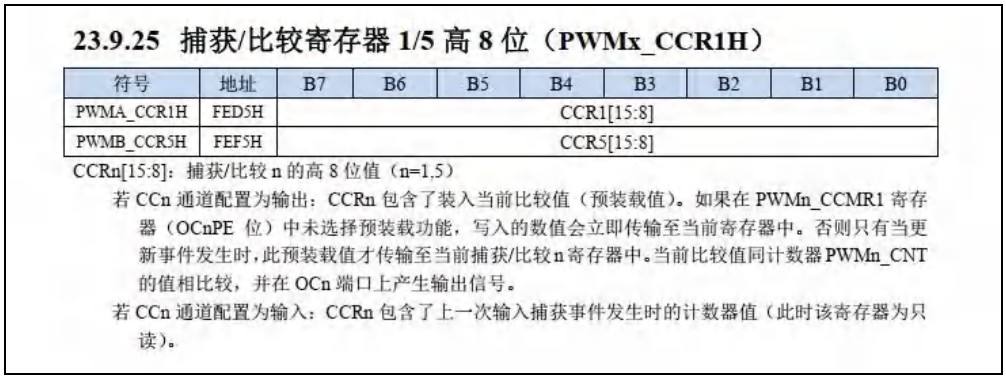
2.手动使能不对称模式

		和OC2REF的逻辑与运算结果。
1110	不对称PWM模式1	OC1REF与在PWM模式1下的行为相同。计数器递增计数时，OC1REFC输出OC1REF；计数器递减计数时，OC1REFC输出OC2REF。
1111	不对称PWM模式2	OC1REF与在PWM模式2下的行为相同。计数器递增计数时，OC1REFC输出OC1REF；计数器递减计数时，OC1REFC输出OC2REF。

注 1：一旦 LOCK 级别设为 3（PWMn_BKR 寄存器中的 LOCK 位）并且 CCnS=00（该通道配置成

这里尤其要注意 OCxM[bit2:0]在 CCMx 寄存器，但是最高位 OCxM[bit3]在 CCMxX 寄存器（小写的 x 表示哪个端口）；

3.通道比较数值设置



23.9.25　捕获/比较寄存器 1/5 高 8 位（PWMx_CCR1H）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR1H	FED5H	CCR1[15:8]							
PWMB_CCR5H	FEF5H	CCR5[15:8]							

CCRn[15:8]：捕获/比较 n 的高 8 位值（n=1,5）

若 CCn 通道配置为输出：CCRn 包含了装入当前比较值（预装载值）。如果在 PWMn_CCMR1 寄存器（OCnPE 位）中未选择预装载功能，写入的数值会立即传输至当前寄存器中。否则只有当更新事件发生时，此预装载值才传输至当前捕获/比较 n 寄存器中。当前比较值同计数器 PWMn_CNT 的值相比较，并在 OCn 端口上产生输出信号。

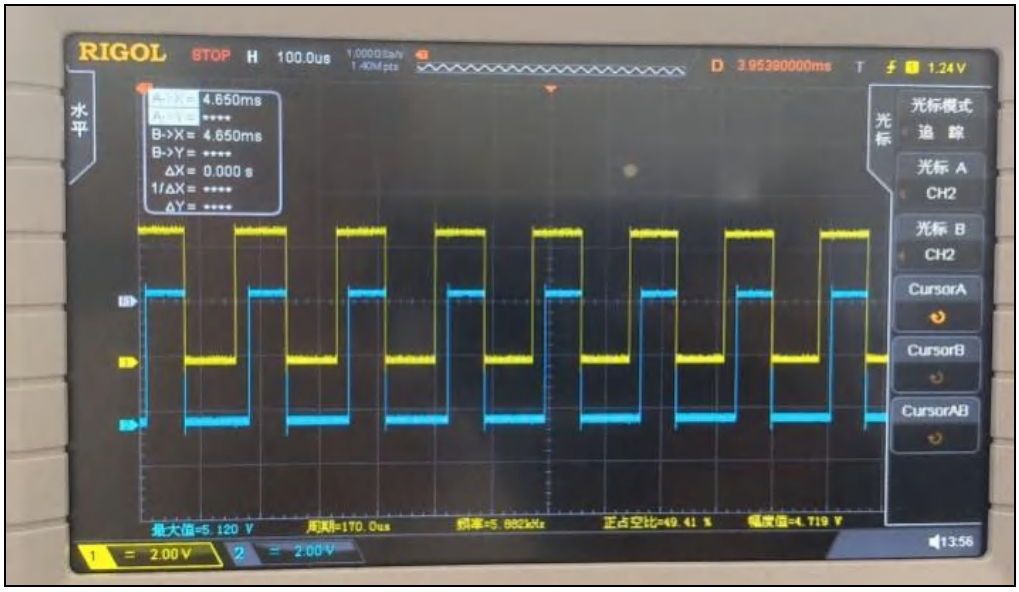
若 CCn 通道配置为输入：CCRn 包含了上一次输入捕获事件发生时的计数器值（此时该寄存器为只读）。

四、代码实战

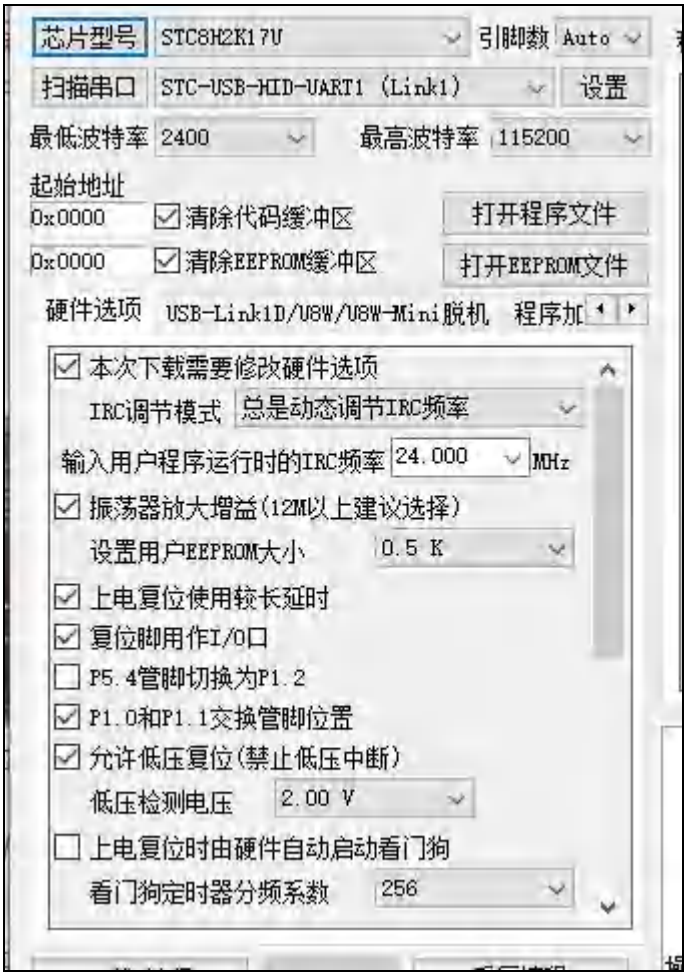
首先我们可以看下来验证一下第二章里的第四点 4.1P 端口设置为不对称输出时起始电平时间点为 CCR1, 结束时间点为 CCR2;　2P 端口设置为不对称输出时起始电平时间点为 CCR2,结束时间点为 CCR1）是什么意思：

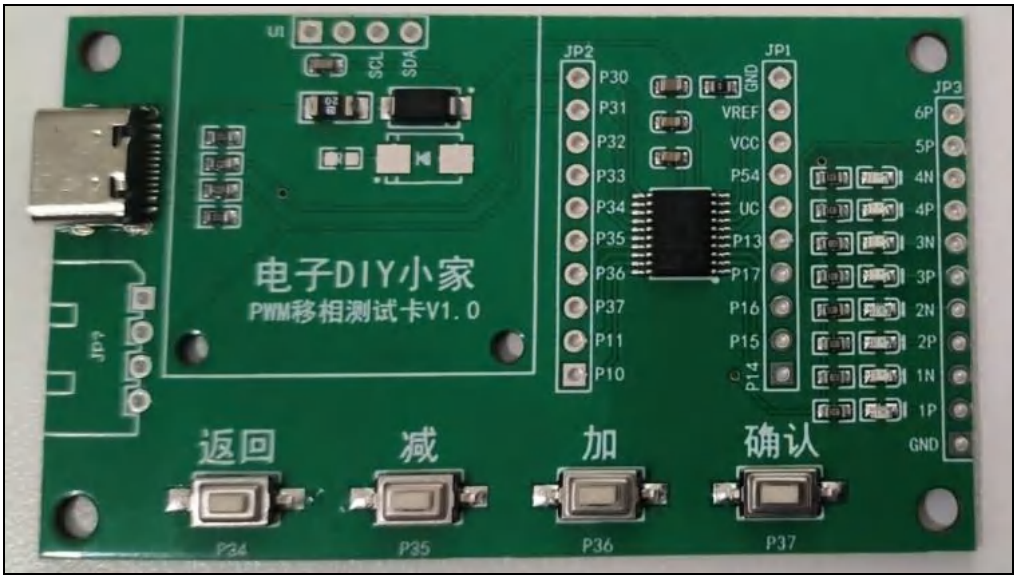
1）首先配置 1P 端口为不对称 PWM 模式 2 输出，2P 端口 PWM 模式 2 输出

```
PWM1_Duty = 3;
PWM2_Duty = 5;
PWMA_PSCRH = 0;
PWMA_PSCRL = 255;
PWMA_CCER1 = 0x00; //写 CCMRx 前必须先清零 CCxE 关闭通道
PWMA_CCER2 = 0x00;
PWMA_CCMR1X = 0x01;//端口 1:不对称 PWM2 模式
PWMA_CCMR1 = 0x70;
PWMA_CCMR2X = 0x00;//端口 2:PWM2 模式
PWMA_CCMR2 = 0x70;
PWMA_CCER1 = 0x55; //配置通道输出使能和极性
PWMA_CCER2 = 0x55;
PWMA_ARRH = (u8)(PWM_PERIOD >> 8); //设置周期时间
PWMA_ARRL = (u8)PWM_PERIOD;
PWMA_ENO = 0x00;
PWMA_ENO |= ENO1P; //使能输出
PWMA_ENO |= ENO2P; //使能输出
PWMA_PS = 0x00;　//高级 PWM 通道输出脚选择位
PWMA_PS |= PWM1_0; //选择 PWM1_0 通道
PWMA_PS |= PWM2_0; //选择 PWM2_0 通道
```



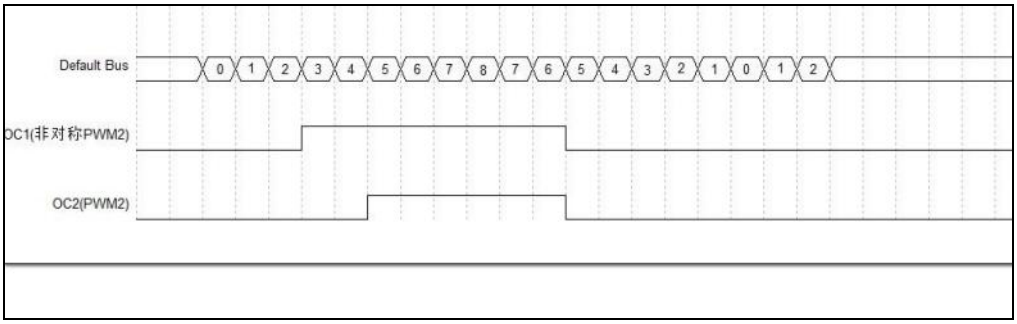
效果图如上，黄色线条为 1P 端口，即不对称 PWM2 模式。蓝色线条为 2P 端口，即 PWM2 模式。下载选项和使用的硬件如下图所示





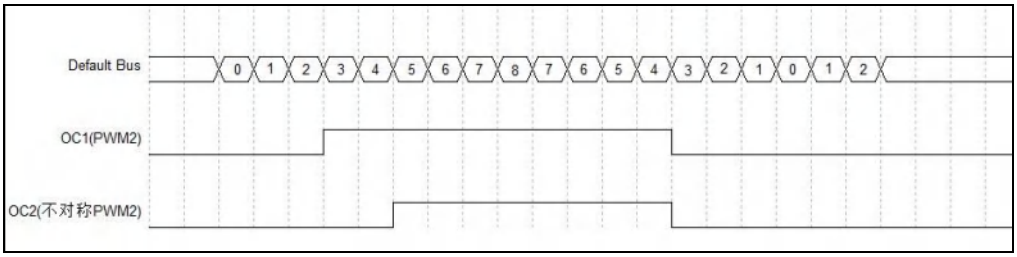
1.首先 PWM 频率 = 时钟/(分频+1)/ARR/2 = 24M / 256/8/2 ≈ 5.86Khz，和示波器测的频率基本一致

2.按照我们的分析，实际上端口 1 应该在上升计数为 3 的时候变成高电平，下降计数为 5 的时候变为低电平，占空比为 50%。端口 2 应该在上升计数为 5 的时候变成高电平，下降计数为 5 的时候变为低电平，例如下图所示，可以看到最终测量到的结果和这个一致，可见这个分析是正确的。

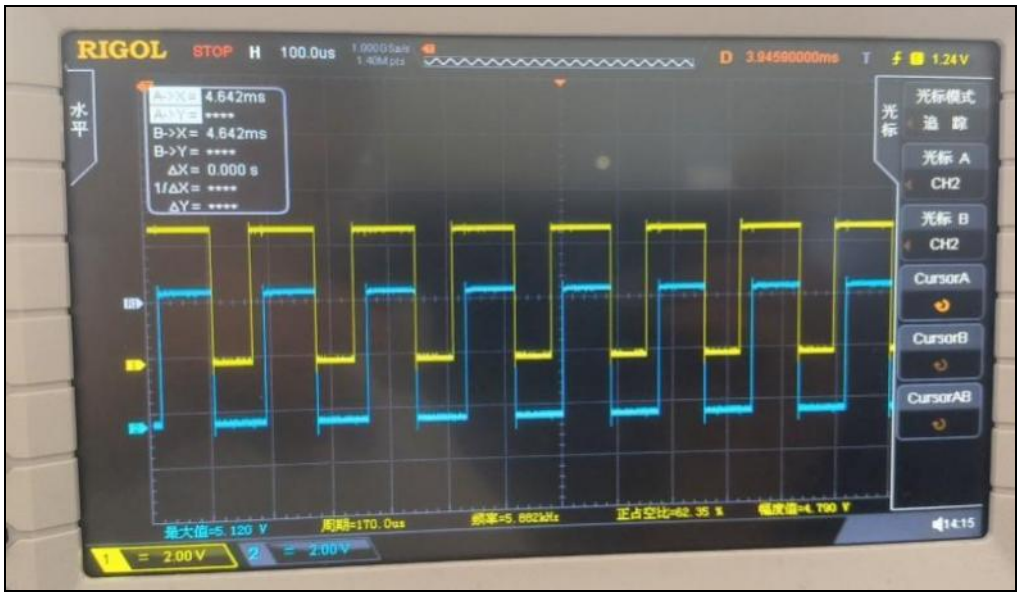


2）首先配置 1P 端口为不对称 PWM 模式 2 输出，2P 端口 PWM 模式 2 输出

```
PWM1_Duty = 3;
PWM2_Duty = 5;
PWMA_PSCRH = 0;
PWMA_PSCRL = 255;
PWMA_CCER1 = 0x00; //写 CCMRx 前必须先清零 CCxE 关闭通道
PWMA_CCER2 = 0x00;
PWMA_CCMR1X = 0x00;//端口 1:称 PWM2 模式
PWMA_CCMR1 = 0x70;
PWMA_CCMR2X = 0x01;//端口 2:不对 PWM2 模式
PWMA_CCMR2 = 0x70;
PWMA_CCER1 = 0x55; //配置通道输出使能和极性
PWMA_CCER2 = 0x55;
PWMA_ARRH = (u8)(PWM_PERIOD >> 8); //设置周期时间
PWMA_ARRL = (u8)PWM_PERIOD;
PWMA_ENO = 0x00;
PWMA_ENO |= ENO1P; //使能输出
PWMA_ENO |= ENO2P; //使能输出
PWMA_PS = 0x00;  //高级 PWM 通道输出脚选择位
PWMA_PS |= PWM1_0; //选择 PWM1_0 通道
PWMA_PS |= PWM2_0; //选择 PWM2_0 通道
```



我们下载进去实测一下：



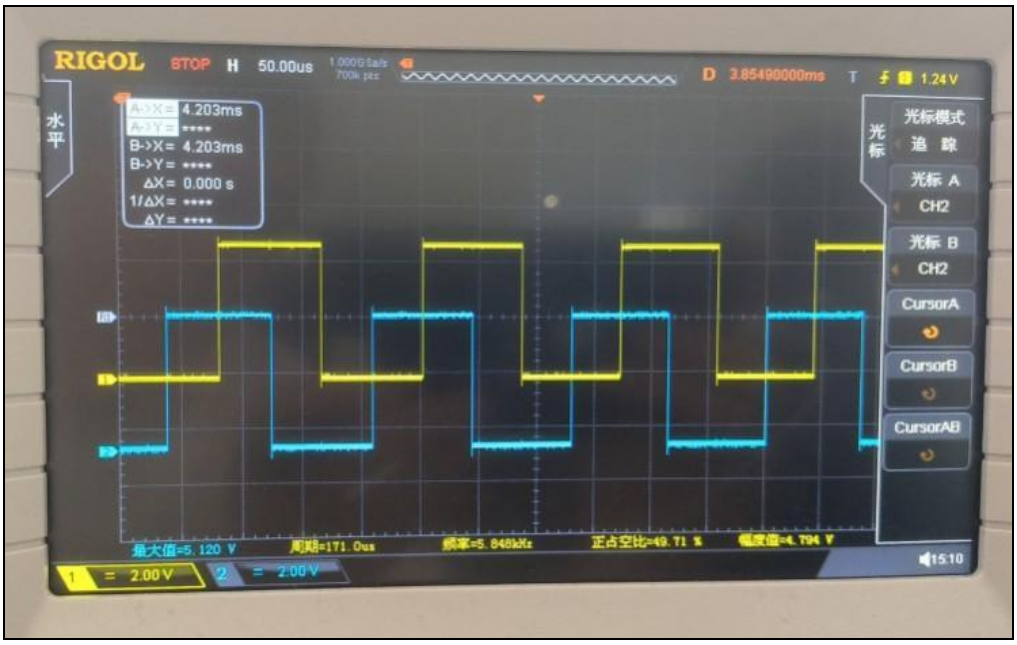
可以看到这个波形就是我们预期的波形

3）最后我们来实现一下正交编码信号的波形。
通道 1 使用 PWM2 不对称模式，通道 3 使用不对称 PWM2 模式

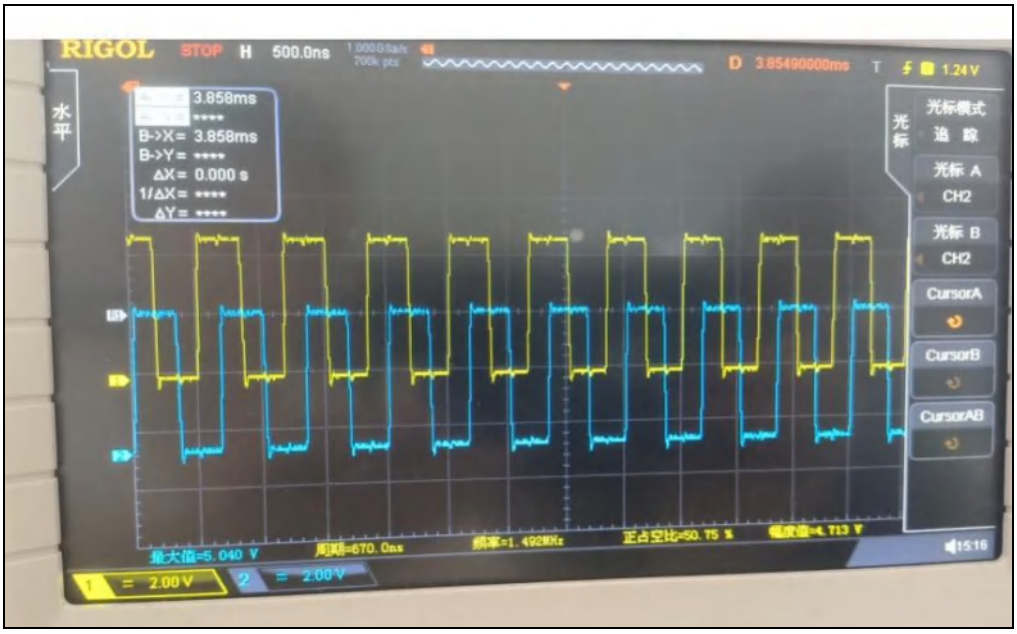
```
PWM1_Duty = 0;
PWM2_Duty = 8;
PWM3_Duty = 4;
PWM4_Duty = 4;
PWMA_PSCRH = 0;
PWMA_PSCRL = 255;
PWMA_CCER1 = 0x00; //写 CCMRx 前必须先清零 CCxE 关闭通道
PWMA_CCER2 = 0x00;
PWMA_CCMR1X = 0x01;//通道 1:PWM2 不对称模式
PWMA_CCMR1 = 0x70;
PWMA_CCMR2X = 0x00;//通道 2:PWM2 模式,极性反转
PWMA_CCMR2 = 0x70;
PWMA_CCMR3X = 0x01;//通道 3:不对称 PWM2 模式
PWMA_CCMR3 = 0x70;
PWMA_CCMR4X = 0x00;//通道 4:PWM2 模式
PWMA_CCMR4 = 0x70;
PWMA_CCER1 = 0x55; //配置通道输出使能和极性
PWMA_CCER2 = 0x55;
PWMA_ARRH = (u8)(PWM_PERIOD >> 8); //设置周期时间
PWMA_ARRL = (u8)PWM_PERIOD;
PWMA_ENO = 0x00;
PWMA_ENO |= ENO1P; //使能输出
PWMA_ENO |= ENO3P; //使能输出
PWMA_PS = 0x00;  //高级 PWM 通道输出脚选择位
PWMA_PS |= PWM1_0; //选择 PWM1_0 通道
```

```
PWMA_PS |= PWM2_0; //选择 PWM2_0 通道
PWMA_PS |= PWM3_0; //选择 PWM3_0 通道
PWMA_PS |= PWM4_0; //选择 PWM4_0 通道
```

最后下载到单片机看下最终的波形效果。



之所以这里要用硬件，就是频率可以很快，且不需要进中断即可实现。（代码里为了方便测量将 PSC 分频寄存器配置了 255，实际上这个改为 0，即不分频速度分分钟能到 Mhz 级别，效果如下）



31 高速高级 PWM（HSPWM），可以使用 PLL 高速时钟作为时钟源

产品线	高速高级 PWM
Ai8051U 系列	●

Ai8051U 系列单片机为高级 PWMA 和高级 PWMB 提供了高速模式（HSPWMA 和 HSPWMB）。高速高级 PWM 是以高级 PWMA 和高级 PWMB 为基础，增加了高速模式。

当系统运行在较低工作频率时，高速高级 PWM 可工作在高达 144M 的频率下。从而可以达到降低内核功耗，提升外设性能的目的

31.1 相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
HSPWMA_CFG	高速 PWMA 配置寄存器	7EFBF0H	-	-	-	EXTN	-	INTEN	ASYNCEN	1	xxx0,0001
HSPWMA_ADR	高速 PWMA 地址寄存器	7EFBF1H	RW/BUSY	ADDR[6:0]							0000,0000
HSPWMA_DAT	高速 PWMA 数据寄存器	7EFBF2H	DATA[7:0]								0000,0000
HSPWMB_CFG	高速 PWMB 配置寄存器	7EFBF4H	-	-	-	EXTN	-	INTEN	ASYNCEN	1	xxx0,0001
HSPWMB_ADR	高速 PWMB 地址寄存器	7EFBF5H	RW/BUSY	ADDR[6:0]							0000,0000
HSPWMB_DAT	高速 PWMB 数据寄存器	7EFBF6H	DATA[7:0]								0000,0000

31.1.1 HSPWM 配置寄存器（HSPWMn_CFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
HSPWMA_CFG	7EFBF0H	-	-	-	EXTN	-	INTEN	ASYNCEN	1
HSPWMB_CFG	7EFBF4H	-	-	-	EXTN	-	INTEN	ASYNCEN	1

ASYNCEN：异步控制模式使能位

0：关闭异步控制。

1：使能异步控制模式。

注：当关闭异步控制时，高级 PWMA/高级 PWMB 为传统模式，此时高级 PWM 会自动选择系统工作频率，PWM 工作频率与系统工作频率相同；若需要时 PWM 工作在高速模式，则需要使能异步控制模式，此时 PWM 时钟可选择主时钟（MCLK）或者 PLL 输出时钟

INTEN：异步模式中断使能位

0：关闭异步模式下的 PWM 中断。

1：使能异步模式下的 PWM 中断。

注：异步模式下，若需要响应高级 PWMA/高级 PWMB 的中断，则必须使能 INTEN 位

EXTN：间接寻址的基地址选择

0：间接寻址的基地址，BASE[15:7] = 0x1fd，间接寻址范围为 FE80H～FEFFH

1：间接寻址的基地址，BASE[15:7] = 0x1f2，间接寻址范围为 F900H～F97FH

31.1.2 HSPWM 地址寄存器（HSPWMn_AD）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
HSPWMA_ADR	7EFBF1H	RW/BUSY	ADDR[6:0]						
HSPWMB_ADR	7EFBF5H	RW/BUSY	ADDR[6:0]						

ADDR[6:0]：高级 PWMA/PWMB 的特殊功能寄存器地址低 7 位

RW/BUSY：读写控制位、状态位

写 0：异步方式写 PWMA/PWMB 的特殊功能寄存器。

写 1：异步方式读 PWMA/PWMB 的特殊功能寄存器。

读 0：异步读写已经完成

读 1：异步读写正在进行，处于忙状态

31.1.3 HSPWM 数据寄存器（HSPWMn_DAT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
HSPWMA_DAT	7EFBF2H	DATA[7:0]							
HSPWMB_DAT	7EFBF6H	DATA[7:0]							

DATA[7:0]：高级 PWMA/PWMB 的特殊功能寄存器数据

写：写数据到高级 PWMA/PWMB 的特殊功能寄存器。

读：从高级 PWMA/PWMB 的特殊功能寄存器读取数据。

异步读取 PWMA 的特殊功能寄存器步骤：（PWMB 类似）

1、读取 HSPWMA_ADR，等待 BUSY 为 0，确定前一个异步读写已完成

2、将 PWMA 的特殊功能寄存器的低 7 位写入 HSPWMA_ADR，同时置“1”HSPWMA_ADR.7

3、读取 HSPWMA_ADR，等待 BUSY 为 0

4、读取 HSPWMA_DAT

异步写 PWMA 的特殊功能寄存器步骤：（PWMB 类似）

1、读取 HSPWMA_ADR，等待 BUSY 为 0，确定前一个异步读写已完成

2、将需要写入 PWMA 的特殊功能寄存器的数据写入 HSPWMA_DAT

3、将 PWMA 的特殊功能寄存器的低 7 位写入 HSPWMA_ADR，同时清“0”HSPWMA_ADR.7

4、读取 HSPWMA_ADR，等待 BUSY 为 0。（可跳过此步继续执行其他代码，以提高系统效率）

特别注意：特殊功能寄存器 PWMA_PS 和 PWMB_PS 属于端口控制寄存器，不属于 PWMA 和 PWMB 寄存器组，所以无论是否启动 PWM 的异步控制模式，PWMA_PS 和 PWMB_PS 寄存器都只能使用普通同步模式进行读写

31.2 高速高级 PWM + DMA（组合使用）特别注意事项

Ai8051U 系列单片机的高级 PWMA 工作在普通模式和高速模式时，均支持 DMA 操作。

下表是 PWMA 进行 DMA 操作时的相关特殊功能寄存器。

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
PWMA_DER	PWMA_DMA 事件使能寄存器	7EF948H	-	TDE	COMDE	CC4DE	CC3DE	CC2DE	CC1DE	UDE	0000,0000
PWMA_DBA	PWMA_DMA 基址寄存器	7EF949H	-	-	-	DBA[4:0]					xxx0,0000
PWMA_DBL	PWMA_DMA 基址长度寄存器	7EF94AH	-	-	-	DBL[4:0]					xxx0,0000
PWMA_DMACR	PWMA_DMA 控制寄存器	7EF94BH	-	-	-	DSKIP	DDIR	DMAEN	SIZE[1:0]		xxx0,0000

当 PWMA 工作在普通模式（非高速模式）时:

PWMA 的所有相关寄存器均使用直接寻址方式读写，DMA 操作时，上表中的 PWMA 的 DMA 相关寄存器也使用直接寻址方式读写。

当 PWMA 工作在高速模式时:

PWMA 的所有相关寄存器均使用间接寻址方式读写，**DMA 操作时，上表中的 PWMA 的 DMA 相关寄存器也必须使用间接寻址的方式进行读写（这里需要特别注意）**。

31.3 范例程序

31.3.1 使能高级 PWM 的高速模式（异步模式）

```
//测试工作频率为 12MHz

#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#define    FOSC    12000000UL

#define    HSCK_MCLK       0
#define    HSCK_PLL        1
#define    HSCK_SEL        HSCK_PLL

#define    PLL_96M         0
#define    PLL_144M        1
#define    PLL_SEL         PLL_144M

#define    CKMS            0x80
#define    HSIOCK          0x40
#define    MCK2SEL_MSK     0x0c
#define    MCK2SEL_SEL1    0x00
#define    MCK2SEL_PLL     0x04
#define    MCK2SEL_PLLD2   0x08
#define    MCK2SEL_IRC48   0x0c
#define    MCKSEL_MSK      0x03
#define    MCKSEL_HIRC     0x00
#define    MCKSEL_XOSC     0x01
#define    MCKSEL_X32K     0x02
#define    MCKSEL_IRC32K   0x03

#define    ENCKM           0x80
#define    PCKI_MSK        0x60
#define    PCKI_D1         0x00
#define    PCKI_D2         0x20
#define    PCKI_D4         0x40
#define    PCKI_D8         0x60

void delay()
{
    int    i;

    for    (i=0; i<100; i++);
}

char ReadPWMA(char addr)
{
    char dat;

    while (HSPWMA_ADR & 0x80);                          //等待前一个异步读写完成
    HSPWMA_ADR = addr | 0x80;                           //设置间接访问地址,只需要设置原 XFR 地址的低 7 位
                                                        //HSPWMA_ADR 寄存器的最高位写 1,表示读数据
    while (HSPWMA_ADR & 0x80);                          //等待当前异步读取完成
```

```
    dat    = HSPWMA_DAT;                    //读取异步数据

    return dat;
}

void WritePWMA(char addr, char dat)
{
    while (HSPWMA_ADR & 0x80);              //等待前一个异步读写完成
    HSPWMA_DAT = dat;                       //准备需要写入的数据
    HSPWMA_ADR = addr & 0x7f;               //设置间接访问地址,只需要设置原 XFR 地址的低 7 位
                                            //HSPWMA_ADR 寄存器的最高位写 0,表示写数据
}

void main()
{
    EAXFR = 1;                              //允许访问扩展的特殊寄存器，XFR
                                            //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                          //允许访问扩展的特殊寄存器，XFR
                                            //(8 位模式请使用这句，注释上一句)
    WTST = 0;                               //设置取程序代码等待时间，
                                            //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                              //设置访问片内的 xdata 速度，
                                            //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    //选择 PLL 输出时钟
#if  (PLL_SEL == PLL_96M)
    CLKSEL &= ~CKMS;                        //选择 PLL 的 96M 作为 PLL 的输出时钟
#elif (PLL_SEL == PLL_144M)
    CLKSEL |= CKMS;                         //选择 PLL 的 144M 作为 PLL 的输出时钟
#else
    CLKSEL &= ~CKMS;                        //默认选择 PLL 的 96M 作为 PLL 的输出时钟
#endif

    //选择 PLL 输入时钟分频,保证输入时钟为 12M
    USBCLK &= ~PCKI_MSK;
#if  (FOSC == 12000000UL)
    USBCLK |= PCKI_D1;                      //PLL 输入时钟 1 分频
#elif (FOSC == 24000000UL)
    USBCLK |= PCKI_D2;                      //PLL 输入时钟 2 分频
#elif (FOSC == 48000000UL)
    USBCLK |= PCKI_D4;                      //PLL 输入时钟 4 分频
#elif (FOSC == 96000000UL)
    USBCLK |= PCKI_D8;                      //PLL 输入时钟 8 分频
#else
    USBCLK |= PCKI_D1;                      //默认 PLL 输入时钟 1 分频
#endif

    //启动 PLL
    USBCLK |= ENCKM;                        //使能 PLL 倍频

    delay();                                //等待 PLL 锁频

    //选择 HSPWM/HSSPI 时钟
#if  (HSCK_SEL == HSCK_MCLK)
    CLKSEL &= ~HSIOCK;                      //HSPWM/HSSPI 选择主时钟为时钟源
#elif (HSCK_SEL == HSCK_PLL)
    CLKSEL |= HSIOCK;                       //HSPWM/HSSPI 选择 PLL 输出时钟为时钟源
#else
    CLKSEL &= ~HSIOCK;                      //默认 HSPWM/HSSPI 选择主时钟为时钟源
```

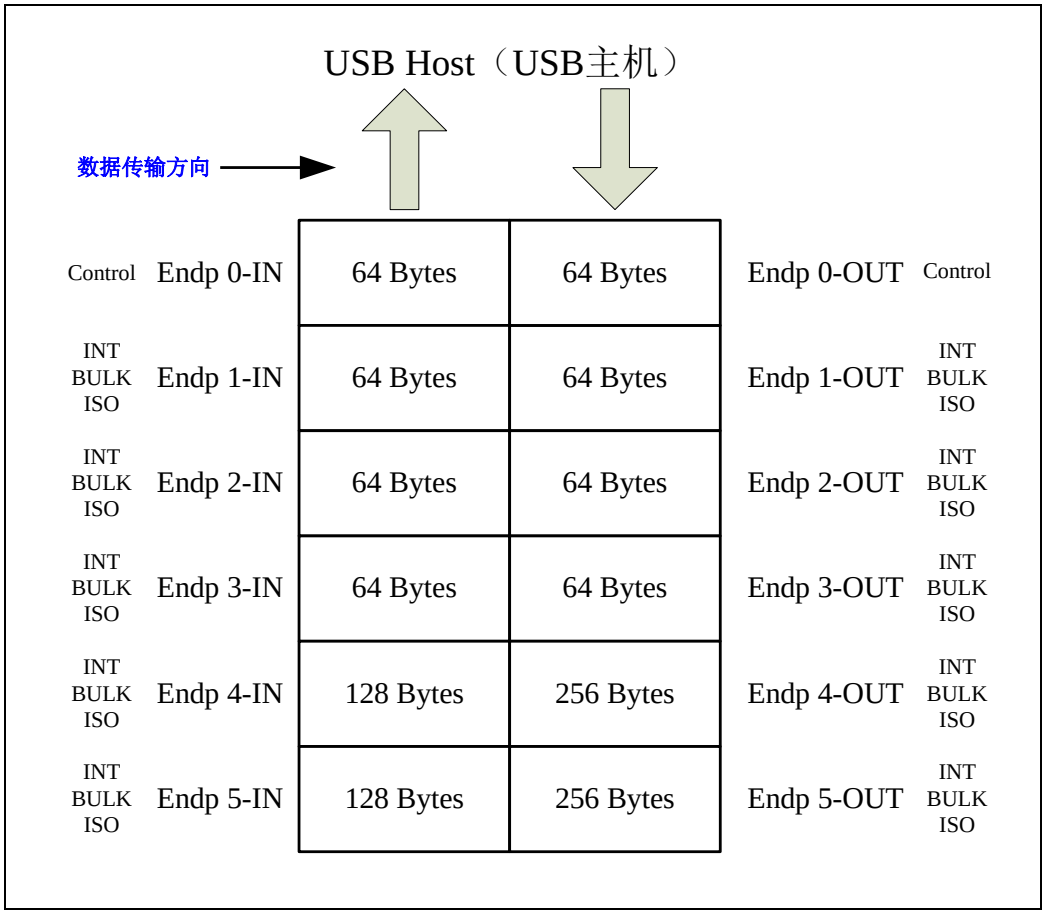
```
#endif

    HSCLKDIV = 0;                                   //HSPWM/HSSPI 时钟源不分频

    HSPWMA_CFG = 0x03;                              //使能 PWMA 相关寄存器异步访问功能

    //通过异步方式设置 PWMA 的相关寄存器
    WritePWMA((char)&PWMA_CCER1, 0x00);
    WritePWMA((char)&PWMA_CCMR1, 0x00);             //CC1 为输出模式
    WritePWMA((char)&PWMA_CCMR1, 0x60);             //OC1REF 输出 PWM1(CNT<CCR 时输出有效电平 1)
    WritePWMA((char)&PWMA_CCER1, 0x05);             //使能 CC1/CC1N 上的输出功能
    WritePWMA((char)&PWMA_ENO, 0x03);               //使能 PWM 信号输出到端口
    WritePWMA((char)&PWMA_BKR, 0x80);               //使能主输出
    WritePWMA((char)&PWMA_CCR1H, 200 >> 8);         //设置输出 PWM 的占空比
    WritePWMA((char)&PWMA_CCR1L, 200);
    WritePWMA((char)&PWMA_ARRH, 1000 >> 8);         //设置输出 PWM 的周期
    WritePWMA((char)&PWMA_ARRL, 1000);
    WritePWMA((char)&PWMA_DTR, 10);                 //设置互补对称输出 PWM 的死区
    WritePWMA((char)&PWMA_CR1, 0x01);               //开始 PWM 计数

    P2M0 = 0;
    P2M1 = 0;
    P3M0 = 0;
    P3M1 = 0;

    P2 = ReadPWMA((char)&PWMA_ARRH);                //异步方式读取寄存器
    P3 = ReadPWMA((char)&PWMA_ARRL);

    while (1);
}
```

32　USB 2.0-FS 通用串行总线

产品线	USB
Ai8051U 系列	●

Ai8051U 系列单片机内部集成 USB2.0/USB1.1 兼容全速 USB，6 个双向端点，支持 4 种端点传输模式（控制传输、中断传输、批量传输和同步传输），每个端点拥有 64 字节的缓冲区。

USB 模块共有 1280 字节的 FIFO，结构如下：



32.1 USB 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
USBCLK	USB 时钟控制寄存器	DCH	ENCKM	PCKI[1:0]		CRE	TST_USB	TST_PHY	PHYTST[1:0]		0010,0000
USBCON	USB 控制寄存器	F4H	ENUSB	ENUSBRST	PS2M	PUEN	PDEN	DFREC	DP	DM	0000,0000
USBADR	USB 地址寄存器	ECH	BUSY	AUTORD	UADR[5:0]						0000,0000
USBDAT	USB 数据寄存器	FCH									0000,0000

32.1.1 USB 控制寄存器（USBCON）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
USBCON	F4H	ENUSB	ENUSBRST	PS2M	PUEN	PDEN	DFREC	DP	DM

ENUSB：USB 功能与 USB 时钟控制位

0：关闭 USB 功能与 USB 时钟

1：使能 USB 功能与 USB 时钟

ENUSBRST：USB 复位设置控制位

0：关闭 USB 复位设置

1：使能 USB 复位

PS2M：PS2 mode 功能控制位

0：关闭 PS2 mode 功能

1：使能 PS2 mode 功能

PUEN：DP/DM 端口上 1.5K 上拉电阻控制位

0：禁止上拉电阻

1：使能上拉电阻

PDEN：DP/DM 端口上 500K 下拉电阻控制位

0：禁止下拉电阻

1：使能下拉电阻

DFREC：差分接收状态位（只读）

0：当前 DP/DM 的差分状态为“0”

1：当前 DP/DM 的差分状态为“1”

DP：D+端口状态（PS2 为 0 时只读，PS2 为 1 时可读写）

0：当前 D+为逻辑 0 电平

1：当前 D+为逻辑 1 电平

DM：D-端口状态（PS2 为 0 时只读，PS2 为 1 时可读写）

0：当前 D-为逻辑 0 电平

1：当前 D-为逻辑 1 电平

32.1.2 USB 时钟控制寄存器（USBCLK）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
USBCLK	DCH	ENCKM	PCKI[1:0]		CRE	TST_USB	TST_PHY	PHYTST[1:0]	

ENCKM：PLL 倍频控制

0：禁止 PLL 倍频

1：使能 PLL 倍频

PCKI[1:0]：PLL 时钟选择

PCKI[1:0]	PLL 时钟源
00	/1
01	/2
10	/4
11	/8

CRE：时钟追频控制位

0：禁止时钟追频

1：使能时钟追频

TST_USB: USB 测试模式

0：禁止 USB 测试模式

1：使能 USB 测试模式

TST_PHY: PHY 测试模式

0：禁止 PHY 测试模式

1：使能 PHY 测试模式

PHYTST[1:0]：USB PHY 测试

PHYTST[1:0]	方式	DP	DM
00	方式 0：正常	x	x
01	方式 1：强制“1”	1	0
10	方式 2：强制“0”	0	1
11	方式 3：强制单端“0”	0	0

32.1.3 USB 间址地址寄存器（USBADR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
USBADR	FCH	BUSY	AUTORD	UADR[5:0]					

BUSY：USB 寄存器读忙标志位

写 0：无意义

写 1：启动 USB 间接寄存器的读操作，地址由 USBADR 设定

读 0：USBDAT 寄存器中的数据有效

读 1：USBDAT 寄存器中的数据无效，USB 正在读取间接寄存器

~~AUTORD：USB 寄存器自动读标志，用于 USB 的 FIFO 的块读取~~

~~写 0：每次读取间接 USB 寄存器都必须先写 BUSY 标志位~~

~~写 1：当软件读取 USBDAT 时，下一个 USB 间接寄存器的读取将自动启动（USBADR 不变）~~

UADR[5:0]：USB 间接寄存器的地址

32.1.4 USB 间址数据寄存器（USBDAT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
USBDAT	ECH	UDAT[7:0]							

UDAT[7:0]：用于间接读写 USB 寄存器

32.2 USB 控制器寄存器（SIE）

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
30H	UTRKCTL	UTRKSTS						
28H								
20H	FIFO0	FIFO1	FIFO2	FIFO3	FIFO4	FIFO5		
18H								
10H	INMAXP	CSR0 INCSR1	INCSR2	OUTMAXP	OUTCSR1	OUTCSR2	COUNT0 OUTCOUNT1	OUTCOUNT2
08H		INTROUT1E		INTRUSBE	FRAME1	FRAME2	INDEX	
00H	FADDR	POWER	INTRIN1	-	INTROUT1	-	INTRUSB	INTRIN1E

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
FADDR	USB 功能地址寄存器	00H	UPDATE	UADDR[6:0]							0000,0000
POWER	USB 电源管理寄存器	01H	ISOUD	-	-	-	USBRST	USBRSU	USBSUS	ENSUS	0xxx,0000
INTRIN1	USB 端点 IN 中断标志位	02H	-	-	EP5INIF	EP4INIF	EP3INIF	EP2INIF	EP1INIF	EP0IF	xx00,0000
INTROUT1	USB 端点 OUT 中断标志位	04H	-	-	EP5OUTIF	EP4OUTIF	EP3OUTIF	EP2OUTIF	EP1OUTIF	-	xx00,000x
INTRUSB	USB 电源中断标志位	06H	-	-	-	-	SOFIF	RSTIF	RSUIF	SUSIF	xxxx,0000
INTRIN1E	USB 端点 IN 中断允许位	07H	-	-	EP5INIE	EP4INIE	EP3INIE	EP2INIE	EP1INIE	EP0IE	xx11,1111
INTROUT1E	USB 端点 OUT 中断允许位	09H	-	-	EP5OUTIE	EP4OUTIE	EP3OUTIE	EP2OUTIE	EP1OUTIE	-	xx11,111x
INTRUSBE	USB 电源中断允许位	0BH	-	-	-	-	SOFIE	RSTIE	RSUIE	SUSIE	xxxx,0110
FRAME1	USB 数据帧号低字节	0CH	FRAME[7:0]								0000,0000
FRAME2	USB 数据帧号高字节	0DH	-	-	-	-	-	FRAME[10:8]			xxxx,x000
INDEX	USB 端点索引寄存器	0EH	-	-	-	-	-	INDEX[2:0]			xxxx,x000
INMAXP	IN 端点的最大数据包大小	10H	INMAXP[7:0]								0000,0000
CSR0	端点 0 控制状态寄存器	11H	SSUEND	SOPRDY	SDSTL	SUEND	DATEND	STSTL	IPRDY	OPRDY	0000,0000
INCSR1	IN 端点控制状态寄存器 1	11H	-	CLRDT	STSTL	SDSTL	FLUSH	UNDRUN	FIFONE	IPRDY	x000,0000
INCSR2	IN 端点控制状态寄存器 2	12H	AUTOSET	ISO	MODE	ENDMA	FCDT	-	-	-	0010,0xxx
OUTMAXP	OUT 端点的最大数据包大小	13H	OUTMAXP[7:0]								0000,0000
OUTCSR1	OUT 端点控制状态寄存器 1	14H	CLRDT	STSTL	SDSTL	FLUSH	DATERR	OVRRUN	FIFOFUL	OPRDY	0000,0000
OUTCSR2	OUT 端点控制状态寄存器 2	15H	AUTOCLR	ISO	ENDMA	DMAMD	-	-	-	-	0000,xxxx
COUNT0	端点 0 的 OUT 长度	16H	-	OUTCNT0[6:0]							x000,0000
OUTCOUNT1	USB 端点 OUT 长度低字节	16H	OUTCNT[7:0]								0000,0000
OUTCOUNT2	USB 端点 OUT 长度高字节	17H	-	-	-	-	-	OUTCNT[10:8]			xxxx,x000
FIFO0	端点 0 的 FIFO 访问寄存器	20H	FIFO0[7:0]								0000,0000
FIFO1	端点 1 的 FIFO 访问寄存器	21H	FIFO1[7:0]								0000,0000
FIFO2	端点 2 的 FIFO 访问寄存器	22H	FIFO2[7:0]								0000,0000
FIFO3	端点 3 的 FIFO 访问寄存器	23H	FIFO3[7:0]								0000,0000
FIFO4	端点 4 的 FIFO 访问寄存器	24H	FIFO4[7:0]								0000,0000
FIFO5	端点 5 的 FIFO 访问寄存器	25H	FIFO5[7:0]								0000,0000
UTRKCTL	USB 跟踪控制寄存器	30H	FTM1	FTM0	INTV[1:0]		ENST5	RES[2:0]			1011,1011
UTRKSTS	USB 跟踪状态寄存器	31H	INTVCNT[3:0]				STS[1:0]		TST_UTRK	UTRK_RDY	1111,00x0

32.2.1 USB 功能地址寄存器（FADDR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
FADDR	00H	UPDATE	UADDR[6:0]						

UPDATE：更新 USB 功能地址

0：最后的 UADDR 地址已生效

1：最后的 UADDR 地址还未生效

UADDR[6:0]：保存 USB 的 7 位功能地址

32.2.2 USB 电源控制寄存器（POWER）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
POWER	01H	ISOUD	-	-	-	USBRST	USBRSU	USBSUS	ENSUS

ISOUD（ISO Update）：ISO 更新

0：当软件向 IPRDY 写“1”时，USB 在收到下一个 IN 令牌后发送数据包

1：当软件向 IPRDY 写“1”时，USB 在收到 SOF 令牌后发送数据包，如果 SOF 令牌之前收到 IN 令牌，则 USB 发送长度为 0 的数据包

USBRST（USB Reset）：USB 复位控制位

向此位写“1”，可强制产生异步 USB 复位。读取此位可以获得当前总线上的复位状态信息

0：总线上没有检测到复位信号

1：总线上检测到了复位信号

USBRSU（USB Resume）：USB 恢复控制位

以软件方式在总线上强制产生恢复信号，以便将 USB 设备从挂起方式进行远程唤醒。当 USB 处于挂起模式（USBSUS=1）时，向此位写“1”，将强制在 USB 总线上产生恢复信号，软件应在 10-15ms 后向此位写“0”，以结束恢复信号。软件向 USBRSU 写入“0”后将产生 USB 恢复中断，此时硬件会自动将 USBSUS 清“0”

USBSUS（USB Suspend）：USB 挂起控制位

当 USB 进入挂起方式时，此位被硬件置“1”。当以软件方式在总线上强制产生恢复信号后或者在总线上检测到恢复信号时且在读取了 INTRUSB 寄存器后，硬件自动将此位清“0”。

ENSUS（Enable Suspend Detection）：使能 USB 挂起方式检测

0：禁止挂起检测，USB 将忽略总线上的挂起信号

1：使能挂起检测，当检测到总线上的挂起信号，USB 将进入挂起方式

32.2.3 USB 端点 IN 中断标志位（INTRIN1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INTRIN1	02H	-	-	EP5INIF	EP4INIF	EP3INIF	EP2INIF	EP1INIF	EP0IF

EP5INIF：端点 5 的 IN 中断标志位

0：端点 5 的 IN 中断无效

1：端点 5 的 IN 中断有效

EP4INIF：端点 4 的 IN 中断标志位

0：端点 4 的 IN 中断无效

1：端点 4 的 IN 中断有效

EP3INIF：端点 3 的 IN 中断标志位

0：端点 3 的 IN 中断无效

1：端点 3 的 IN 中断有效

EP2INIF：端点 2 的 IN 中断标志位

0：端点 2 的 IN 中断无效

1：端点 2 的 IN 中断有效

EP1INIF：端点 1 的 IN 中断标志位

0：端点 1 的 IN 中断无效

1：端点 1 的 IN 中断有效

EP0IF：端点 0 的 IN/OUT 中断标志位

0：端点 0 的 IN/OUT 中断无效

1：端点 0 的 IN/OUT 中断有效

在软件读取 INTRIN1 寄存器后，硬件将自动清除 INTRIN1 中的所有的中断标志

32.2.4 USB 端点 OUT 中断标志位（INTROUT1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INTROUT1	04H	-	-	EP5OUTIF	EP4OUTIF	EP3OUTIF	EP2OUTIF	EP1OUTIF	-

EP5OUTIF：端点 5 的 OUT 中断标志位

0：端点 5 的 OUT 中断无效

1：端点 5 的 OUT 中断有效

EP4OUTIF：端点 4 的 OUT 中断标志位

0：端点 4 的 OUT 中断无效

1：端点 4 的 OUT 中断有效

EP3OUTIF：端点 3 的 OUT 中断标志位

0：端点 3 的 OUT 中断无效

1：端点 3 的 OUT 中断有效

EP2OUTIF：端点 2 的 OUT 中断标志位

0：端点 2 的 OUT 中断无效

1：端点 2 的 OUT 中断有效

EP1OUTIF：端点 1 的 OUT 中断标志位

0：端点 1 的 OUT 中断无效

1：端点 1 的 OUT 中断有效

在软件读取 INTROUT1 寄存器后，硬件将自动清除 INTROUT1 中的所有的中断标志

32.2.5 USB 电源中断标志（INTRUSB）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INTRUSB	06H	-	-	-	-	SOFIF	RSTIF	RSUIF	SUSIF

SOFIF：USB 帧起始信号中断标志

0：USB 帧起始信号中断无效

1：USB 帧起始信号中断有效

RSTIF：USB 复位信号中断标志

0：USB 复位信号中断无效

1：USB 复位信号中断有效

RSUIF：USB 恢复信号中断标志

0：USB 恢复信号中断无效

1：USB 恢复信号中断有效

SUSIF：USB 挂起信号中断标志

0：USB 挂起信号中断无效

1：USB 挂起信号中断有效

在软件读取 INTRUSB 寄存器后，硬件将自动清除 INTRUSB 中的所有的中断标志

32.2.6 USB 端点 IN 中断允许寄存器（INTRIN1E）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INTRIN1E	07H	-	-	EP5INIE	EP4INIE	EP3INIE	EP2INIE	EP1INIE	EP0IE

EP5INIE：端点 5 的 IN 中断控制位

0：禁止端点 5 的 IN 中断

1：允许端点 5 的 IN 中断

EP4INIE：端点 4 的 IN 中断控制位

0：禁止端点 4 的 IN 中断

1：允许端点 4 的 IN 中断

EP3INIE：端点 3 的 IN 中断控制位

0：禁止端点 3 的 IN 中断

1：允许端点 3 的 IN 中断

EP2INIE：端点 2 的 IN 中断控制位

0：禁止端点 2 的 IN 中断

1：允许端点 2 的 IN 中断

EP1INIE：端点 1 的 IN 中断控制位

0：禁止端点 1 的 IN 中断

1：允许端点 1 的 IN 中断

EP0IE：端点 0 的 IN/OUT 中断控制位

0：禁止端点 0 的 IN/OUT 中断

1：允许端点 0 的 IN/OUT 中断

32.2.7 USB 端点 OUT 中断允许寄存器（INTROUT1E）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INTROUT1E	09H	-	-	EP5OUTIE	EP4OUTIE	EP3OUTIE	EP2OUTIE	EP1OUTIE	-

EP5OUTIE：端点 5 的 OUT 中断控制位

0：禁止端点 5 的 OUT 中断

1：允许端点 5 的 OUT 中断

EP4OUTIE：端点 4 的 OUT 中断控制位

0：禁止端点 4 的 OUT 中断

1：允许端点 4 的 OUT 中断

EP3OUTIE：端点 3 的 OUT 中断控制位

0：禁止端点 3 的 OUT 中断

1：允许端点 3 的 OUT 中断

EP2OUTIE：端点 2 的 OUT 中断控制位

0：禁止端点 2 的 OUT 中断

1：允许端点 2 的 OUT 中断

EP1OUTIE：端点 1 的 OUT 中断控制位

0：禁止端点 1 的 OUT 中断

1：允许端点 1 的 OUT 中断

32.2.8 USB 电源中断允许寄存器（INTRUSBE）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INTRUSBE	0BH	-	-	-	-	SOFIE	RSTIE	RSUIE	SUSIE

SOFIE：USB 帧起始信号中断控制位

0：禁止 USB 帧起始信号中断

1：允许 USB 帧起始信号中断

RSTIE：USB 复位信号中断控制位

0：禁止 USB 复位信号中断

1：允许 USB 复位信号中断

RSUIE：USB 恢复信号中断控制位

0：禁止 USB 恢复信号中断

1：允许 USB 恢复信号中断

SUSIE：USB 挂起信号中断控制位

0：禁止 USB 挂起信号中断

1：允许 USB 挂起信号中断

32.2.9 USB 数据帧号寄存器（FRAMEn）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
FRAME1	0CH	FRAME[7:0]							
FRAME2	0DH	-	-	-	-	-	FRAME[10:8]		

FRAME[10:0]：用于保存最后接收到的数据帧的 11 位帧号

32.2.10 USB 端点索引寄存器（INDEX）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INDEX	0EH	-	-	-	-	-	INDEX[2:0]		

INDEX[2:0]：选择 USB 端点

INDEX[2:0]	目标端点
000	端点 0
001	端点 1
010	端点 2
011	端点 3
100	端点 4
101	端点 5

32.2.11 IN 端点的最大数据包大小（INMAXP）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INMAXP	10H	INMAXP[7:0]							

INMAXP[7:0]：设置 USB 的 IN 端点最大数据包的大小（**注意：数据包是以 8 字节为单位。例如若需要将端点的数据包设置为 64 字节，则需要将此寄存器设置为 8**）

当需要获取/设置此信息时，首先必须使用 INDEX 来选择目标端点 0~5

32.2.12 USB 端点 0 控制状态寄存器（CSR0）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CSR0	11H	SSUEND	SOPRDY	SDSTL	SUEND	DATEND	STSTL	IPRDY	OPRDY

SSUEND（Serviced Setup End）

SETUP 结束事件处理完成标志。当处理完成 SETUP 结束事件（SUEND）后，软件需要设置 SSUEND 标志位，硬件检测到 SSUEND 被写入“1”时自动将 SUEND 位清“0”。

SOPRDY（Serviced OPRDY）

OPRDY 事件处理完成标志。当处理完成从端点 0 接收到的数据包后，软件需要设置 SOPRDY 标志位，硬件检测到 SOPRDY 被写入“1”时自动将 OPRDY 位清“0”。

SDSTL（Send Stall）

当接收到错误的条件或者不支持的请求时，可以向此位写“1”来结束当前的数据传输。当 STALL 信号被发送后，硬件自动将此位清“0”。

SUEND（Setup End）

SETUP 安装包结束标志。当一次控制传输在软件向 DATAEND 位写“1”之前结束时，硬件将此只读位置“1”。当软件向 SSUEND 写‘1’后，硬件将该位清“0”。

DATEND（Data End）

数据结束。软件应在下列情况下将此位写“1”：

1、当发送最后一个数据包后，固件向 IPRDY 写“1”时；

2、当发送一个零长度数据包后，固件向 IPRDY 写“1”时；

3、当接收完最后一个数据包后，固件向 SOPRDY 写“1”时；

该位将被硬件自动清“0”

STSTL（Sent Stall）

STALL 信号发送完成标志。发送完成 STALL 信号后，硬件将该位置“1”。该位必须用软件清“0”。

IPRDY（IN Packet Ready）

IN 数据包准备完成标志。软件应在将一个要发送的数据包装入到端点 0 的 FIFO 后，将该位置“1”。在发生下列条件之一时，硬件将该位清“0”：

1、数据包已发送时；

2、数据包被一个 SETUP 包覆盖时；

3、数据包被一个 OUT 包覆盖时；

OPRDY（OUT Packet Ready）

OUT 数据包准备完成标志。当收到一个 OUT 数据包时，硬件将该只读位置“1”，并产生中断。该位只在软件向 SOPRDY 位写“1”时才被清“0”。

当需要获取/设置此信息时，首先必须使用 INDEX 来选择目标端点 0

32.2.13 IN 端点控制状态寄存器 1（INCSR1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INCSR1	11H	-	CLRDT	STSTL	SDSTL	FLUSH	UNDRUN	FIFONE	IPRDY

CLRDT（Clear Data Toggle）：复位 IN 数据切换位。

当 IN 端点由于被重新配置或者被 STALL 而需要将数据切换位复位到“0”时，软件需要向此数据位写“1”。

STSTL（Sent Stall）：STALL 信号发送完成标志。

当 STALL 信号被发送完成后，硬件会将该位置“1”（此时 FIFO 被清空，IPRDY 位被清“0”）。该标志必须用软件清“0”。

SDSTL（Send Stall）：STALL 信号发送请求位。

软件应向该位写“1”以产生 STALL 信号作为对一个 IN 令牌的应答。软件应向该位写“0”以结束 STALL 信号。该位对 ISO 方式没有影响。

FLUSH（FIFO Flush）：清除 IN 端点的 FIFO 的下一个数据包。

向该位写“1”将从 IN 端点 FIFO 中清除待发送的下一个数据包。FIFO 指针被复位，IPRDY 位被清“0”。如果 FIFO 中包含多个数据包，软件必须对每个数据包向 FLUSH 写“1”。当 FIFO 清空完成后，硬件将 FLUSH 位清“0”。

UNDRUN（Data Underrun）：数据不足。

该位的功能取决于 IN 端点的方式：

ISO 方式：在 IPRDY 为“0”且收到一个 IN 令牌后发送了一个零长度数据包时，该位被置“1”。

中断/批量方式：当使用 NAK 作为对一个 IN 令牌的应答时，该位被置“1”。

该位必须用软件清“0”。

FIFONE（FIFO Not Empty）：IN 端点的 FIFO 非空标志

0：IN 端点的 FIFO 为空

1：IN 端点的 FIFO 包含有一个或者多个数据包

IPRDY（IN Packet Ready）：IN 数据包准备完成标志。

软件应在将一个要发送的数据包装入到端点的 FIFO 后，将该位置“1”。在发生下列条件之一时，硬件将该位清“0”：

1、数据包已发送时；

2、自动设置被使能（AUTOSET =‘1’）且端点 IN 的 FIFO 数据包达到 INMAXP 所设置的值；

3、如果端点处于同步方式且 ISOUD 为“1”，在收到下一个 SOF 之前 IPRDY 的读出值总是为 0。

需要获取/设置此信息时，首先必须使用 INDEX 来选择目标端点 1~5

32.2.14 IN 端点控制状态寄存器 2（INCSR2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INCSR2	12H	AUTOSET	ISO	MODE	ENDMA	FCDT	-	-	-

AUTOSET：自动设置 IPRDY 标志控制位。

0：禁止自动设置 IPRDY 标志

1：使能自动设置 IPRDY（必须向 IN FIFO 中装载的数据达到 INMAXP 所设置的值，否则 IPRDY 标志必须手动设置）

ISO：同步传输使能。

0：端点被配置为批量/中断传输方式

1：端点被配置为同步传输方式

MODE：端点方向选择位。

0：选择端点方向为 OUT

1：选择端点方向为 IN

~~ENDMA：IN 端点的 DMA 控制~~

~~0：禁止 IN 端点的 DMA 请求~~

~~1：使能 IN 端点的 DMA 请求~~

FCDT：强制 DATA0/DATA1 数据切换设置。

0：端点数据只在发送完一个数据包后且收到 ACK 时切换。

1：端点数据在每发送完一个数据包后被强制切换，不管是否收到 ACK。

需要获取/设置此信息时，首先必须使用 INDEX 来选择目标端点 1~5

32.2.15 OUT 端点的最大数据包大小（OUTMAXP）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
OUTMAXP	13H	OUTMAXP[7:0]							

OUTMAXP[7:0]：设置 USB 的 OUT 端点最大数据包的大小（**注意：数据包是以 8 字节为单位。例如若需要将端点的数据包设置为 64 字节，则需要将此寄存器设置为 8**）

需要获取/设置此信息时，首先必须使用 INDEX 来选择目标端点 1~5

32.2.16 OUT 端点控制状态寄存器 1（OUTCSR1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
OUTCSR1	14H	CLRDT	STSTL	SDSTL	FLUSH	DATERR	OVRRUN	FIFOFUL	OPRDY

CLRDT（Clear Data Toggle）：复位 OUT 数据切换位。

当 OUT 端点由于被重新配置或者被 STALL 而需要将数据切换位复位到“0”时，软件需要向此数据位写“1”。

STSTL（Sent Stall）：STALL 信号发送完成标志。

当 STALL 信号被发送完成后，硬件会将该位置“1”。该标志必须用软件清“0”。

SDSTL（Send Stall）：STALL 信号发送请求位。

软件应向该位写“1”以产生 STALL 信号作为对一个 OUT 令牌的应答。软件应向该位写“0”以结束 STALL 信号。该位对 ISO 方式没有影响。

FLUSH（FIFO Flush）：清除 OUT 端点的 FIFO 的下一个数据包。

向该位写“1”将从 OUT 端点 FIFO 中清除下一个数据包。FIFO 指针被复位，OPRDY 位被清“0”。如果 FIFO 中包含多个数据包，软件必须对每个数据包向 FLUSH 写“1”。当 FIFO 清空完成后，硬件将 FLUSH 位清“0”。

DATAERR（Data Error）：数据错误。

在 ISO 方式，如果接收到的数据包有 CRC 或位填充错误，该位被硬件置‘1’。当软件清除 OPRDY 时，该位被清‘0’。该位只在 ISO 方式有效

OVRRUN（Data Overrun）：数据溢出。

当一个输入数据包不能被装入到 OUT 端点 FIFO 时，该位被硬件置“1”。该位只在 ISO 方式有效。该位必须用软件清“0”。

0：无数据溢出

1：自该标志最后一次被清除以来，因 FIFO 已满导致数据包丢失

FIFOFUL（FIFO Full）：OUT 端点的 FIFO 数据满标志。

0：OUT 端点的 FIFO 未满

1：OUT 端点的 FIFO 已满

OPRDY（OUT Packet Ready）：OUT 数据包接收完成标志。

当有数据包可用时硬件将该位置“1”。软件应在将每个数据包从 OUT 端点 FIFO 卸载后将该位清‘0’。

需要获取/设置此信息时，首先必须使用 INDEX 来选择目标端点 1~5

32.2.17 OUT 端点控制状态寄存器 2（OUTCSR2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
OUTCSR2	15H	AUTOCLR	ISO	ENDMA	DMAMD	-	-	-	-

AUTOCLR：自动清除 OPRDY 标志控制位。

0：禁止自动清除 OPRDY 标志

1：使能自动清除 OPRDY（必须从 OUT FIFO 中下载的数据达到 OUTMAXP 所设置的值，否则 OPRDY 标志必须手动清除）

ISO：同步传输使能。

0：端点被配置为批量/中断传输方式

1：端点被配置为同步传输方式

ENDMA：OUT 端点的 DMA 控制

0：禁止 OUT 端点的 DMA 请求

1：使能 OUT 端点的 DMA 请求

DMAMD：设置 OUT 端点的 DMA 模式

需要获取/设置此信息时，首先必须使用 INDEX 来选择目标端点 1~5

32.2.18 USB 端点 0 的 OUT 长度（COUNT0）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
COUNT0	16H	-	OUTCNT0[6:0]						

OUTCNT0[6:0]：端点 0 的 OUT 字节长度

COUNT0 专用于保存端点 0 最后接收到的 OUT 数据包的数据长度（由于端点 0 数据包最长只能为 64 字节，所以只需要 7 位）。此长度值只在端点 0 的 OPRDY 位为“1”时才有效。

需要获取此长度信息时，首先必须使用 INDEX 来选择目标端点 0

32.2.19 USB 端点的 OUT 长度（OUTCOUNTn）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
OUTCOUNT1	16H	OUTCNT[7:0]							
OUTCOUNT2	17H	-	-	-	-	-	OUTCNT[10:8]		

OUTCNT[10:0]：端点的 OUT 字节长度

OUTCOUNT1 和 OUTCOUNT2 联合组成一个 11 位的数，保存最后 OUT 数据包的数据长度，适用于端点 1~5。此长度值只在端点 1~5 的 OPRDY 位为“1”时才有效。

需要获取此长度信息时，首先必须使用 INDEX 来选择目标端点 1~5

32.2.20 USB 端点的 FIFO 数据访问寄存器（FIFOn）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
FIFO0	20H	FIFO0[7:0]							
FIFO1	21H	FIFO1[7:0]							
FIFO2	22H	FIFO2[7:0]							
FIFO3	23H	FIFO3[7:0]							
FIFO4	24H	FIFO4[7:0]							
FIFO5	25H	FIFO5[7:0]							

FIFOn[7:0]：USB 各个端点的 IN/OUT 数据间接访问寄存器

32.2.21 USB 跟踪控制寄存器（UTRKCTL）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
UTRKCTL	30H	FTM1	FTM0	INTV[1:0]		ENST5	RES[2:0]		

FTM1：HFOSC 微调控制位

0：硬件使用粗调和微调为调整频率

1：硬件仅使用微调位调整 HFOSC

FTM0：FTM1 为 0 时有效

0：UTRK 使用全部 128 级微调

1：UTRK 禁止使用最大及最小 12 级之微调

INTV[1:0]：选择 UTRK 更新周期

INTV[1:0]	周期
00	2ms
01	4ms
10	8ms
11	16ms

ENST5：使能加 5 阶和减 5 阶

0：校准上下限为 10%

1：校准上下限为 20%

RES[2:0]：UTRK 自动调节分辨率设置

RES[2:0]	分辨率	调节值
000	8	0.067%
001	12	0.100%
010	16	0.133%
011	**24**	**0.200%**
100	28	0.233%
101	32	0.267%
110	48	0.4%
111	64	0.5%

32.2.22 USB 跟踪状态寄存器（UTRKSTS）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
UTRKSTS	31H	INTVCNT[3:0]				STS[1:0]		TST_UTRK	UTRK_RDY

INTVCNT[3:0]：USB 跟踪时的内部硬件计数器（软件只读）

STS[1:0]：UTRK 状态

STS[1:0]	状态
00	**INC 5**
01	DEC 5
10	INC 1
11	DEC 1

TST_UTRK：UTRK 测试模式控制

0：禁止 UTRK 测试模式

1：使能 UTRK 测试模式

UTRK_RDY：UTRK 校准完成状态位

32.3 USB 产品开发注意事项

每个 USB 产品都必须有自己唯一的 VID&PID 组合，才能被电脑正确识别。若两个不同的 USB 产品对应的 VID&PID 组合相同，则可能出现电脑对 USB 产品的识别出现异常，从而无法正常使用 USB 产品。为避免出现这种情况，VID 和 PID 均需通过正规途径进行统一规划和分配。

目前已通过 USB-IF 组织取得了专用 USB 设备的 VID 编号 13503（十六进制：34BF）。客户使用 USB 芯片开发自己的 USB 产品时，若您已通过其它途径获取了您自己的 VID，则相应的 PID 可自行规划。若您的 USB 产品需要使用官方 VID，则产品的 PID 务必请您向我司申请。

32.4 如何软复位到系统区启动【USB 直接下载，不管 P3.2】

如果当前用户程序正在运行 USB,
请先【在用户区先关闭 USB】
再软复位到系统区，就是等待 USB 下载

```
USBCON = 0x00;          //清除USB设置
USBCLK = 0x00;          //停止USB时钟
IRC48MCR = 0x00;        //停止48M IRC时钟

sleep_ms(10);           //等待USB总线复位

IAP_CONTR = 0x60;       //触发软件复位
```

32.5 范例程序

32.5.1 HID 人机接口设备范例

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

typedef     unsigned char       BYTE;
typedef     unsigned int        WORD;
typedef     unsigned long       DWORD;

#define     FADDR       0
#define     POWER       1
#define     INTRIN1     2
#define     EP5INIF     0x20
#define     EP4INIF     0x10
#define     EP3INIF     0x08
#define     EP2INIF     0x04
#define     EP1INIF     0x02
#define     EP0IF       0x01
#define     INTROUT1    4
#define     EP5OUTIF    0x20
#define     EP4OUTIF    0x10
#define     EP3OUTIF    0x08
#define     EP2OUTIF    0x04
#define     EP1OUTIF    0x02
#define     INTRUSB     6
#define     SOFIF       0x08
#define     RSTIF       0x04
#define     RSUIF       0x02
#define     SUSIF       0x01
#define     INTRIN1E    7
#define     EP5INIE     0x20
#define     EP4INIE     0x10
#define     EP3INIE     0x08
#define     EP2INIE     0x04
#define     EP1INIE     0x02
#define     EP0IE       0x01
#define     INTROUT1E   9
#define     EP5OUTIE    0x20
#define     EP4OUTIE    0x10
#define     EP3OUTIE    0x08
#define     EP2OUTIE    0x04
#define     EP1OUTIE    0x02
#define     INTRUSBE    11
#define     SOFIE       0x08
#define     RSTIE       0x04
#define     RSUIE       0x02
#define     SUSIE       0x01
#define     FRAME1      12
#define     FRAME2      13
#define     INDEX       14
#define     INMAXP      16
#define     CSR0        17
```

```
#define  SSUEND          0x80
#define  SOPRDY          0x40
#define  SDSTL           0x20
#define  SUEND           0x10
#define  DATEND          0x08
#define  STSTL           0x04
#define  IPRDY           0x02
#define  OPRDY           0x01
#define  INCSR1          17
#define  INCLRDT         0x40
#define  INSTSTL         0x20
#define  INSDSTL         0x10
#define  INFLUSH         0x08
#define  INUNDRUN        0x04
#define  INFIFONE        0x02
#define  INIPRDY         0x01
#define  INCSR2          18
#define  INAUTOSET       0x80
#define  INISO           0x40
#define  INMODEIN        0x20
#define  INMODEOUT       0x00
#define  INENDMA         0x10
#define  INFCDT          0x08
#define  OUTMAXP         19
#define  OUTCSR1         20
#define  OUTCLRDT        0x80
#define  OUTSTSTL        0x40
#define  OUTSDSTL        0x20
#define  OUTFLUSH        0x10
#define  OUTDATERR       0x08
#define  OUTOVRRUN       0x04
#define  OUTFIFOFUL      0x02
#define  OUTOPRDY        0x01
#define  OUTCSR2         21
#define  OUTAUTOCLR      0x80
#define  OUTISO          0x40
#define  OUTENDMA        0x20
#define  OUTDMAMD        0x10
#define  COUNT0          22
#define  OUTCOUNT1       22
#define  OUTCOUNT2       23
#define  FIFO0           32
#define  FIFO1           33
#define  FIFO2           34
#define  FIFO3           35
#define  FIFO4           36
#define  FIFO5           37
#define  UTRKCTL         48
#define  UTRKSTS         49

#define  EPIDLE          0
#define  EPSTATUS        1
#define  EPDATAIN        2
#define  EPDATAOUT       3
#define  EPSTALL         -1

#define  GET_STATUS              0x00
#define  CLEAR_FEATURE           0x01
#define  SET_FEATURE             0x03
```

```
#define     SET_ADDRESS             0x05
#define     GET_DESCRIPTOR          0x06
#define     SET_DESCRIPTOR          0x07
#define     GET_CONFIG              0x08
#define     SET_CONFIG              0x09
#define     GET_INTERFACE           0x0A
#define     SET_INTERFACE           0x0B
#define     SYNCH_FRAME             0x0C

#define     GET_REPORT              0x01
#define     GET_IDLE                0x02
#define     GET_PROTOCOL            0x03
#define     SET_REPORT              0x09
#define     SET_IDLE                0x0A
#define     SET_PROTOCOL            0x0B

#define     DESC_DEVICE             0x01
#define     DESC_CONFIG             0x02
#define     DESC_STRING             0x03
#define     DESC_HIDREPORT          0x22

#define     STANDARD_REQUEST        0x00
#define     CLASS_REQUEST           0x20
#define     VENDOR_REQUEST          0x40
#define     REQUEST_MASK            0x60

typedef struct
{
    BYTE    bmRequestType;
    BYTE    bRequest;
    BYTE    wValueL;
    BYTE    wValueH;
    BYTE    wIndexL;
    BYTE    wIndexH;
    BYTE    wLengthL;
    BYTE    wLengthH;
}SETUP;

typedef struct
{
    BYTE    bStage;
    WORD    wResidue;
    BYTE    *pData;
}EP0STAGE;

void UsbInit();
BYTE ReadReg(BYTE addr);
void WriteReg(BYTE addr, BYTE dat);
BYTE ReadFifo(BYTE fifo, BYTE *pdat);
void WriteFifo(BYTE fifo, BYTE *pdat, BYTE cnt);

char code DEVICEDESC[18];
char code CONFIGDESC[41];
char code HIDREPORTDESC[27];
char code LANGIDDESC[4];
char code MANUFACTDESC[8];
char code PRODUCTDESC[30];

SETUP Setup;
```

```
EP0STAGE Ep0Stage;
BYTE xdata HidFreature[64];
BYTE xdata HidInput[64];
BYTE xdata HidOutput[64];

void main()
{
    EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                           //设置取程序代码等待时间，
                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                          //设置访问片内的 xdata 速度，
                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    UsbInit();

    EA = 1;

    while (1);
}

BYTE ReadReg(BYTE addr)
{
    BYTE dat;

    while (USBADR & 0x80);
    USBADR = addr | 0x80;
    while (USBADR & 0x80);
    dat = USBDAT;

    return dat;
}

void WriteReg(BYTE addr, BYTE dat)
{
    while (USBADR & 0x80);
    USBADR = addr & 0x7f;
    USBDAT = dat;
}

BYTE ReadFifo(BYTE fifo, BYTE *pdat)
{
    BYTE cnt;
    BYTE ret;
```

```
    ret = cnt = ReadReg(COUNT0);
    while (cnt--)
    {
        *pdat++ = ReadReg(fifo);
    }

    return ret;
}

void WriteFifo(BYTE fifo, BYTE *pdat, BYTE cnt)
{
    while (cnt--)
    {
        WriteReg(fifo, *pdat++);
    }
}

void DelayXns(WORD delayTime)
{
    while( delayTime-- );
}

void UsbInit()
{
    P3M0 = 0x00;
    P3M1 = 0x03;

    P_SW2 |= 0x80;
    PLLCR = (1<<7)|(0<<5)|(1<<3)|(1<<1);          // enable PLL
    DelayXns(100);
    CLKSEL = 0x02;                                //选择系统时钟源为内部 pll 输出
    CLKDIV = 0;
    P_SW2 &= ~0x80;
    USBCLK = 0x90;
    USBCON = 0x90;

    WriteReg(FADDR, 0x00);
    WriteReg(POWER, 0x08);
    WriteReg(INTRIN1E, 0x3f);
    WriteReg(INTROUT1E, 0x3f);
    WriteReg(INTRUSBE, 0x00);
    WriteReg(POWER, 0x01);

    Ep0Stage.bStage = EPIDLE;
}

void usb_isr() interrupt 25
{
    BYTE intrusb;
    BYTE intrin;
    BYTE introut;
    BYTE csr;
    BYTE cnt;
    WORD len;

    intrusb = ReadReg(INTRUSB);
    intrin = ReadReg(INTRIN1);
    introut = ReadReg(INTROUT1);
```

```
if (intrusb  & RSTIF)
{
    WriteReg(INDEX, 1);
    WriteReg(INCSR1, INCLRDT);
    WriteReg(INDEX, 1);
    WriteReg(OUTCSR1, OUTCLRDT);
    Ep0Stage.bStage = EPIDLE;
}

if (intrin & EP0IF)
{
    WriteReg(INDEX, 0);
    csr = ReadReg(CSR0);
    if (csr & STSTL)
    {
        WriteReg(CSR0, csr & ~STSTL);
        Ep0Stage.bStage = EPIDLE;
    }
    if (csr & SUEND)
    {
        WriteReg(CSR0, csr | SSUEND);
    }

    switch (Ep0Stage.bStage)
    {
    case EPIDLE:
        if (csr & OPRDY)
        {
            Ep0Stage.bStage = EPSTATUS;
            ReadFifo(FIFO0, (BYTE *)&Setup);
            ((BYTE *)&Ep0Stage.wResidue)[0] = Setup.wLengthH;
            ((BYTE *)&Ep0Stage.wResidue)[1]= Setup.wLengthL;
            switch (Setup.bmRequestType & REQUEST_MASK)
            {
            case STANDARD_REQUEST:
                switch (Setup.bRequest)
                {
                case SET_ADDRESS:
                    WriteReg(FADDR, Setup.wValueL);
                    break;
                case SET_CONFIG:
                    WriteReg(INDEX, 1);
                    WriteReg(INCSR2, INMODEIN);
                    WriteReg(INMAXP, 8);
                    WriteReg(INDEX, 1);
                    WriteReg(INCSR2, INMODEOUT);
                    WriteReg(OUTMAXP, 8);
                    WriteReg(INDEX, 0);
                    break;
                case GET_DESCRIPTOR:
                    Ep0Stage.bStage = EPDATAIN;
                    switch (Setup.wValueH)
                    {
                    case DESC_DEVICE:
                        Ep0Stage.pData = DEVICEDESC;
                        len = sizeof(DEVICEDESC);
                        break;
                    case DESC_CONFIG:
```

```
                Ep0Stage.pData = CONFIGDESC;
                len = sizeof(CONFIGDESC);
                break;
            case DESC_STRING:
                switch (Setup.wValueL)
                {
                case 0:
                    Ep0Stage.pData = LANGIDDESC;
                    len = sizeof(LANGIDDESC);
                    break;
                case 1:
                    Ep0Stage.pData = MANUFACTDESC;
                    len = sizeof(MANUFACTDESC);
                    break;
                case 2:
                    Ep0Stage.pData = PRODUCTDESC;
                    len = sizeof(PRODUCTDESC);
                    break;
                default:
                    Ep0Stage.bStage = EPSTALL;
                    break;
                }
                break;
            case DESC_HIDREPORT:
                Ep0Stage.pData = HIDREPORTDESC;
                len = sizeof(HIDREPORTDESC);
                break;
            default:
                Ep0Stage.bStage = EPSTALL;
                break;
            }
            if (len < Ep0Stage.wResidue)
            {
                Ep0Stage.wResidue = len;
            }
            break;
        default:
            Ep0Stage.bStage = EPSTALL;
            break;
        }
        break;
    case CLASS_REQUEST:
        switch (Setup.bRequest)
        {
        case GET_REPORT:
            Ep0Stage.pData = HidFreature;
            Ep0Stage.bStage = EPDATAIN;
            break;
        case SET_REPORT:
            Ep0Stage.pData = HidFreature;
            Ep0Stage.bStage = EPDATAOUT;
            break;
        case SET_IDLE:
            break;
        case GET_IDLE:
        case GET_PROTOCOL:
        case SET_PROTOCOL:
        default:
            Ep0Stage.bStage = EPSTALL;
```

```
                        break;
                    }
                    break;
                default:
                    Ep0Stage.bStage = EPSTALL;
                    break;
                }

                switch (Ep0Stage.bStage)
                {
                case EPDATAIN:
                    WriteReg(CSR0, SOPRDY);
                    goto L_Ep0SendData;
                    break;
                case EPDATAOUT:
                    WriteReg(CSR0, SOPRDY);
                    break;
                case EPSTATUS:
                    WriteReg(CSR0, SOPRDY | DATEND);
                    Ep0Stage.bStage = EPIDLE;
                    break;
                case EPSTALL:
                    WriteReg(CSR0, SOPRDY | SDSTL);
                    Ep0Stage.bStage = EPIDLE;
                    break;
                }
            }
            break;
        case EPDATAIN:
            if (!(csr & IPRDY))
            {
L_Ep0SendData:
                cnt = Ep0Stage.wResidue > 64 ? 64 : Ep0Stage.wResidue;
                WriteFifo(FIFO0, Ep0Stage.pData, cnt);
                Ep0Stage.wResidue -= cnt;
                Ep0Stage.pData += cnt;
                if (Ep0Stage.wResidue == 0)
                {
                    WriteReg(CSR0, IPRDY | DATEND);
                    Ep0Stage.bStage = EPIDLE;
                }
                else
                {
                    WriteReg(CSR0, IPRDY);
                }
            }
            break;
        case EPDATAOUT:
            if (csr & OPRDY)
            {
                cnt = ReadFifo(FIFO0, Ep0Stage.pData);
                Ep0Stage.wResidue -= cnt;
                Ep0Stage.pData += cnt;
                if (Ep0Stage.wResidue == 0)
                {
                    WriteReg(CSR0, SOPRDY | DATEND);
                    Ep0Stage.bStage = EPIDLE;
                }
                else
```

```
                {
                    WriteReg(CSR0, SOPRDY);
                }
            }
            break;
        }
    }

    if (intrin & EP1INIF)
    {
        WriteReg(INDEX, 1);
        csr = ReadReg(INCSR1);
        if (csr & INSTSTL)
        {
            WriteReg(INCSR1, INCLRDT);
        }
        if (csr & INUNDRUN)
        {
            WriteReg(INCSR1, 0);
        }
    }

    if (introut & EP1OUTIF)
    {
        WriteReg(INDEX, 1);
        csr = ReadReg(OUTCSR1);
        if (csr & OUTSTSTL)
        {
            WriteReg(OUTCSR1, OUTCLRDT);
        }
        if (csr & OUTOPRDY)
        {
            ReadFifo(FIFO1, HidOutput);
            WriteReg(OUTCSR1, 0);

            WriteReg(INDEX, 1);
            WriteFifo(FIFO1, HidOutput, 64);
            WriteReg(INCSR1, INIPRDY);
        }
    }
}

char code DEVICEDESC[18] =
{
    0x12,                                   //bLength(18);
    0x01,                                   //bDescriptorType(Device);
    0x00,0x02,                              //bcdUSB(2.00);
    0x00,                                   //bDeviceClass(0);
    0x00,                                   //bDeviceSubClass0);
    0x00,                                   //bDeviceProtocol(0);
    0x40,                                   //bMaxPacketSize0(64);
    0xbf,0x34,                              //idVendor(34bf);
    0x80,0x43,                              //idProduct(4380);
    0x00,0x01,                              //bcdDevice(1.00);
    0x01,                                   //iManufacturer(1);
    0x02,                                   //iProduct(2);
    0x00,                                   //iSerialNumber(0);
    0x01,                                   //bNumConfigurations(1);
};
```

```
char code CONFIGDESC[41] =
{
    0x09,                                   //bLength(9);
    0x02,                                   //bDescriptorType(Configuration);
    0x29,0x00,                              //wTotalLength(41);
    0x01,                                   //bNumInterfaces(1);
    0x01,                                   //bConfigurationValue(1);
    0x00,                                   //iConfiguration(0);
    0x80,                                   //bmAttributes(BUSPower);
    0x32,                                   //MaxPower(100mA);

    0x09,                                   //bLength(9);
    0x04,                                   //bDescriptorType(Interface);
    0x00,                                   //bInterfaceNumber(0);
    0x00,                                   //bAlternateSetting(0);
    0x02,                                   //bNumEndpoints(2);
    0x03,                                   //bInterfaceClass(HID);
    0x00,                                   //bInterfaceSubClass(0);
    0x00,                                   //bInterfaceProtocol(0);
    0x00,                                   //iInterface(0);

    0x09,                                   //bLength(9);
    0x21,                                   //bDescriptorType(HID);
    0x01,0x01,                              //bcdHID(1.01);
    0x00,                                   //bCountryCode(0);
    0x01,                                   //bNumDescriptors(1);
    0x22,                                   //bDescriptorType(HID Report);
    0x1b,0x00,                              //wDescriptorLength(27);

    0x07,                                   //bLength(7);
    0x05,                                   //bDescriptorType(Endpoint);
    0x81,                                   //bEndpointAddress(EndPoint1 as      IN);
    0x03,                                   //bmAttributes(Interrupt);
    0x40,0x00,                              //wMaxPacketSize(64);
    0x01,                                   //bInterval(10ms);

    0x07,                                   //bLength(7);
    0x05,                                   //bDescriptorType(Endpoint);
    0x01,                                   //bEndpointAddress(EndPoint1 as      OUT);
    0x03,                                   //bmAttributes(Interrupt);
    0x40,0x00,                              //wMaxPacketSize(64);
    0x01,                                   //bInterval(10ms);
};

char code HIDREPORTDESC[27] =
{
    0x05,0x0c,                              //USAGE_PAGE(Consumer);
    0x09,0x01,                              //USAGE(Consumer Control);
    0xa1,0x01,                              //COLLECTION(Application);
    0x15,0x00,                              //    LOGICAL_MINIMUM(0);
    0x25,0xff,                              //    LOGICAL_MAXIMUM(255);
    0x75,0x08,                              //    REPORT_SIZE(8);
    0x95,0x40,                              //    REPORT_COUNT(64);
    0x09,0x01,                              //    USAGE(Consumer Control);
    0xb1,0x02,                              //    FEATURE(Data,Variable);
    0x09,0x01,                              //    USAGE(Consumer Control);
    0x81,0x02,                              //    INPUT(Data,Variable);
    0x09,0x01,                              //    USAGE(Consumer Control);
```

```
        0x91,0x02,                                  //      OUTPUT(Data,Variable);
        0xc0,                                       //END_COLLECTION;
};

char code LANGIDDESC[4] =
{
        0x04,0x03,
        0x09,0x04,
};

char code MANUFACTDESC[8] =
{
        0x08,0x03,
        'S',0,
        'T',0,
        'C',0,
};

char code PRODUCTDESC[30] =
{
        0x1e,0x03,
        'S',0,
        'T',0,
        'C',0,
        ' ',0,
        'U',0,
        'S',0,
        'B',0,
        ' ',0,
        'D',0,
        'e',0,
        'v',0,
        'i',0,
        'c',0,
        'e',0,
};
```

32.5.2 HID(Human Interface Device)协议范例

将代码下载到实验箱后，可使用最新的 ISP 下载软件中的 HID 助手检测测试
详细代码请参考官网上的“AI32G 实验箱演示程序”包中的“69－HID(Human Interface Device)协议范例”

32.5.3 CDC(Communication Device Class)协议范例

WIN10 以下的操作系统需要安装 sys 目录中的驱动程序，WIN10 和 WIN11 免安装驱动
将代码下载到实验箱后，在 PC 端可识别为 USB 转串口的设备
使用实验箱上的 DB9 接口即可与其它串口进行通讯
串口的数据位只支持 8 位，停止位只支持 1 位
校验位可支持：无校验、奇校验、偶校验、1 校验和 0 校验
波特率最高可支持 460800，且支持自定义波特率
详细代码请参考官网上的“AI32G 实验箱演示程序”包中的“70－CDC(Communication Device Class)协议范例”

CDC 协议是通讯设备类的通用协议。官网上的 CDC 范例模拟的是 CDC 协议中的通讯接口类（02）和数据接口类（0a），在 PC 上的用户界面就是串口，设备挂载在 windows 的 “usbser.sys” 通用驱动程序上。使用普通串口助手，选择其中的“COMx”端口即可与 CDC 设备进行数据传输。

如果 CDC 设备当作 Bridge（USB 转串口桥接器），则上位机驱动传下来的波特率、校验位、停止位等参数均需要进行处理，然后将上位机传下来的数据再通过正确的波特率、校验位、停止位等格式将串口数据下传到第三方器件。

如果 CDC 设备只是作为普通的 USB 设备，直接 PC 进行数据传输，则可完全不用处理波特率、校验位、停止位等参数，此时 CDC 设备与 PC 之间的数据传输完全不会受到串口波特率的影响。实际测试的数据传输平均比特率可达 2~4MBPS。

32.5.4 基于 HID 协议的 USB 键盘范例

将代码下载到实验箱后即可实现 USB 键盘的基本功能
跑马灯中的 LED17 为 NumLock 灯、LED16 为 CapsLock 灯、LED15 为 ScrollLock 灯
矩阵按键中的 KEY0~KEY7 分别为键盘中的 1~8
详细代码请参考官网上的“AI32G 实验箱演示程序”包中的“71－基于 HID 协议的 USB 键盘范例”

32.5.5 基于 HID 协议的 USB 鼠标范例

将代码下载到实验箱后即可实现 USB 鼠标的基本功能
矩阵按键中的 KEY0 为鼠标左键，KEY1 为鼠标中键，KEY2 为鼠标右键
矩阵按键中的 KEY4 为左移，KEY5 为右移，KEY6 为上移，KEY7 为下移
详细代码请参考官网上的“AI32G 实验箱演示程序”包中的“72－基于 HID 协议的 USB 鼠标范例”

32.5.6 基于 HID 协议的 USB 鼠标+键盘二合一范例

请前往下面的官网地址下载完整参考范例
https://www.stcaimcu.com/forum.php?mod=viewthread&tid=572&extra=

32.5.7 基于 WINUSB 协议的范例

WIN10 以下的操作系统需要安装 sys 目录中的驱动程序，WIN10 和 WIN11 免安装驱动
可使用 exe 目录下的"AI_WINUSB.exe"进行测试
详细代码请参考官网上的“AI32G 实验箱演示程序”包中的“73－基于 WINUSB 协议的范例”

32.5.8 MSC(Mass Storage Class)协议范例

将代码下载到实验箱后，在 PC 端可识别为一个 512K 的 U 盘
U 盘存储器为实验箱上的外挂 512K 的串行 FLASH
在没有外挂 FLASH 的实验箱上，也可以使用 AI32G12K128 内部的 EEPOM 当存储器
只需要在 config.h 文件中将存储器类型改为 MEMTYPE_INT
然后在 ISP 下载时，设置 EEPROM 大小为 64K，即可实现一个 64K 容量的 U 盘
详细代码请参考官网上的“AI32G 实验箱演示程序”包中的“74－MSC(Mass Storage Class)协议范例”

33 TFT 彩屏接口接口（8/16 位 i8080/M6800 接口）

产品线	TFT 彩屏接口
Ai8051U 系列	●

Ai8051U 系列的部分单片机内部集成了一个 TFT 彩屏接口控制器，可用于驱动目前流行的液晶显示屏模块。可驱动 i8080 接口和 M6800 接口彩屏，支持 8 位和 16 位数据宽度

33.1 TFT 彩屏接口功能脚切换

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFCFG	7EFE50H	LCMIFIE	-	LCMIFIP[1:0]		LCMIFDPS[1:0]		D16_D8	M68_I80
LCMIFCFG2	7EFE51H		LCMIFCPS[1:0]		SETUPT[2:0]			HOLDT[1:0]	

LCMIFCPS[1:0]：TFT 彩屏接口控制脚选择位

LCMIFCPS [1:0]	RS	i8080 的读信号 RD M6800 的使能信号 E	i8080 的写信号 WR M6800 的读写信号 RW
00	P4.5	P4.4	P4.2
01	P4.5	P3.7	P3.6
10	P4.0	P4.4	P4.2
11	P4.0	P3.7	P3.6

LCMIFDPS[1:0]：8 位数据位 LCM 接口数据脚选择位

LCMIFDPS [1:0]	D16_D8	数据字节 DAT[7:0]
00	0	P2[7:0]
01	0	-
10	0	P2[7:0]
11	0	-

LCMIFDPS[1:0]：16 位数据位 TFT 彩屏接口数据脚选择位

LCMIFDPS [1:0]	D16_D8	数据高字节 DAT[15:8]	数据低字节 DAT[7:0]
00	1	P2[7:0]	P0[7:0]
01	1	-	-
10	1	P2[7:0]	{P0[7:4],P4[7],P4[6],P4[3],P4[1]}
11	1	-	-

33.2 TFT 彩屏相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
LCMIFCFG	TFT 彩屏接口配置寄存器	7EFE50H	LCMIFIE	-	LCMIFIP[1:0]		LCMIFDPS[1:0]		D16_D8	M68_I80	0x00,0000
LCMIFCFG2	TFT 彩屏接口配置寄存器 2	7EFE51H	-	LCMIFCPS[1:0]		SETUPT[2:0]			HOLDT[1:0]		x000,0000
LCMIFCR	TFT 彩屏接口控制寄存器	7EFE52H	ENLCMIF	-	-	ST_ENDIAN	-	CMD[2:0]			0xx0,x000
LCMIFSTA	TFT 彩屏接口状态寄存器	7EFE53H	-	-	-	-	-	-	-	LCMIFIF	xxxx,xxx0
LCMIFDATL	TFT 彩屏接口低字节数据	7EFE54H	LCMIFDAT[7:0]								0000,0000
LCMIFDATH	TFT 彩屏接口高字节数据	7EFE55H	LCMIFDAT[15:8]								0000,0000
LCMIFPSCR	TFT 彩屏接口时钟预分频	7EFE56H	LCMIFPSCR[7:0]								0000,0000

33.2.1 TFT 彩屏接口配置寄存器（LCMIFCFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFCFG	7EFE50H	LCMIFIE	-	LCMIFIP[1:0]		LCMIFDPS[1:0]		D16_D8	M68_I80

LCMIFIE：TFT 彩屏接口中断使能控制位

0：禁止 TFT 彩屏接口中断

1：允许 TFT 彩屏接口中断

LCMIFIP[1:0]：TFT 彩屏接口中断优先级控制位

LCMIFIP[1:0]	中断优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

LCMIFDPS[1:0]：TFT 彩屏接口数据脚选择位

LCMIFDPS [1:0]	D16_D8	数据高字节 DAT[15:8]	数据低字节 DAT[7:0]
00	0	N/A	P2[7:0]
01	0	-	-
10	0	N/A	P2[7:0]
11	0	-	-
00	1	P2[7:0]	P0[7:0]
01	1	-	-
10	1	P2[7:0]	{P0[7:4],P4[7],P4[6],P4[3],P4[1]}
11	1	-	-

D16_D8：TFT 彩屏接口数据宽度控制位

0：8 位数据宽度

1：16 位数据宽度

M68_I80：TFT 彩屏接口模式选择位

0：i8080 模式

1：M6800 模式

33.2.2 TFT 彩屏接口配置寄存器 2（LCMIFCFG2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFCFG2	7EFE51H	-	LCMIFCPS[1:0]		SETUPT[2:0]			HOLDT[1:0]	

LCMIFCPS[1:0]：TFT 彩屏接口控制脚选择位

LCMIFCPS [1:0]	RS	i8080 的读信号 RD M6800 的使能信号 E	i8080 的写信号 WR M6800 的读写信号 RW
00	P4.5	P4.4	P4.2
01	P4.5	P3.7	P3.6
10	P4.0	P4.4	P4.2
11	P4.0	P3.7	P3.6

SETUPT[2:0]：TFT 彩屏接口通讯的数据建立时间控制位（详见后续章节的时序图）

HOLDT[1:0]：TFT 彩屏接口通讯的数据保持时间控制位（详见后续章节的时序图）

33.2.3 TFT 彩屏接口控制寄存器（LCMIFCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFCR	7EFE52H	ENLCMIF	-	-	ST_ENDIAN	-	CMD[2:0]		

ENLCMIF：TFT 彩屏接口使能控制位

0：禁止 TFT 彩屏接口功能

1：允许 TFT 彩屏接口功能

ST_ENDIAN：TFT_DMA 从 TFT 彩屏接口读取的 16 位命令或数据存储到 XRAM 的大小端顺序控制

0：大端模式（Big-Endian），即低地址存放高字节数据，高地址存放低字节数据

1：小端模式（Little-Endian），即低地址存放低字节数据，高地址存放高字节数据

注：此位仅在 DMA 对 TFT 进行读取操作时才有效，且必须是读取 16 位命令或数据（8 位数据不存在大小端的问题）

CMD[2:0]：TFT 彩屏接口触发命令

CMD[2:0]	触发命令
100	写命令
101	写数据
110	读命令/状态
111	读数据

33.2.4 TFT 彩屏接口状态寄存器（LCMIFSTA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFSTA	7EFE53H	-	-	-	-	-	-	-	LCMIFIF

LCMIFIF：TFT 彩屏接口中断请求标志，需软件清 0

33.2.5 TFT 彩屏接口数据寄存器（LCMIFDATL，LCMIFDATH）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFDATL	7EFE54H	LCMIFDAT[7:0]							
LCMIFDATH	7EFE55H	LCMIFDAT[15:8]							

LCMIFDAT：TFT 彩屏接口数据寄存器。

当数据宽度为 8 位数据时，只有 LCMDATL 数据有效；
当数据宽度为 16 位数据时，由 LCMDATL 和 LCMDATH 共同组合成 16 位数据

33.2.6 TFT 彩屏接口时钟预分频（LCMIFPSCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFPSCR	7EFE56H	LCMIFPSCR[7:0]							

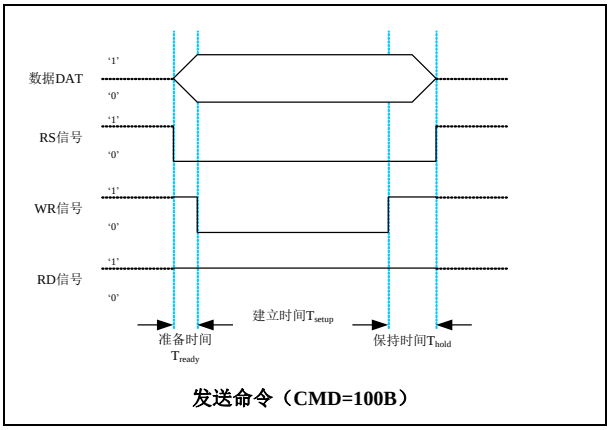
LCMIFPSCR[7:0]：LCMIF 预分频器。预分频器用于对系统时钟进行预分频，分频后的时钟提供给 LCMIF 模块。

LCMIF 时钟频率＝SYSclk /（LCMIFPSCR[7:0]+1）。

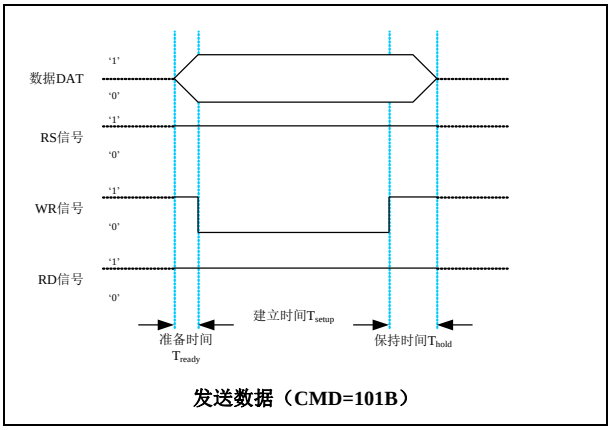
33.3　i8080/M6800 模式 TFT 彩屏接口时序图

注：　T_{ready} ＝1 个系统时钟
　　T_{setup} ＝（SETUPT +1）个系统时钟
　　T_{hold} ＝（HOLDT+1）个系统时钟

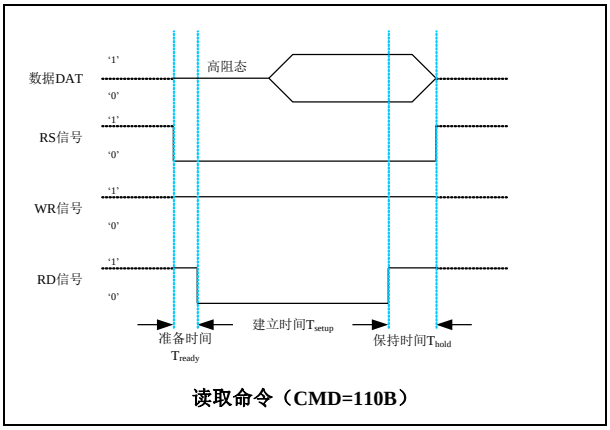
33.3.1　i8080 模式



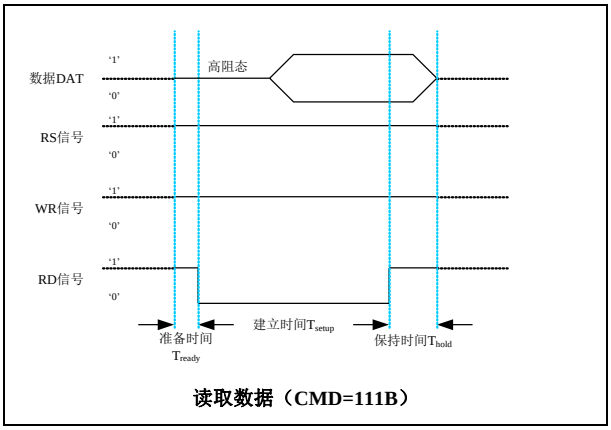
发送命令（CMD=100B）



发送数据（CMD=101B）

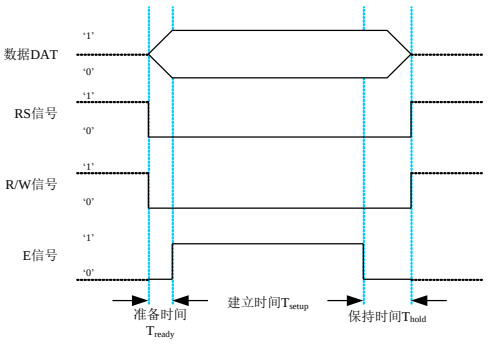


读取命令（CMD=110B）

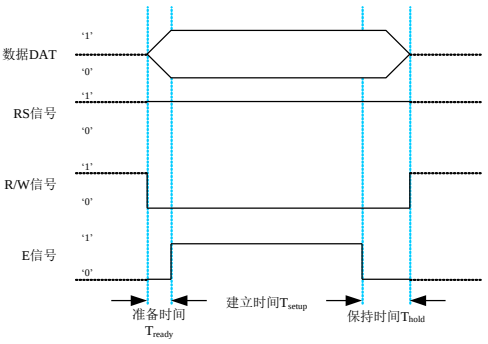


读取数据（CMD=111B）

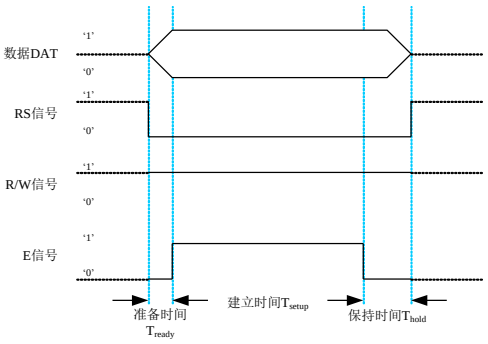
33.3.2 M6800 模式



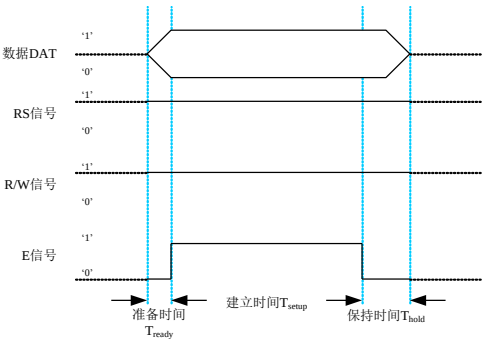
发送命令（CMD=100B）



发送数据（CMD=101B）



读取命令（CMD=110B）



读取数据（CMD=111B）

34 DMA，支持解放 CPU 的外设到外设传输（P2P）

产品线	DMA
Ai8051U 系列	●

Ai8051U 系列单片机支持批量数据存储功能，即传统的 DMA。

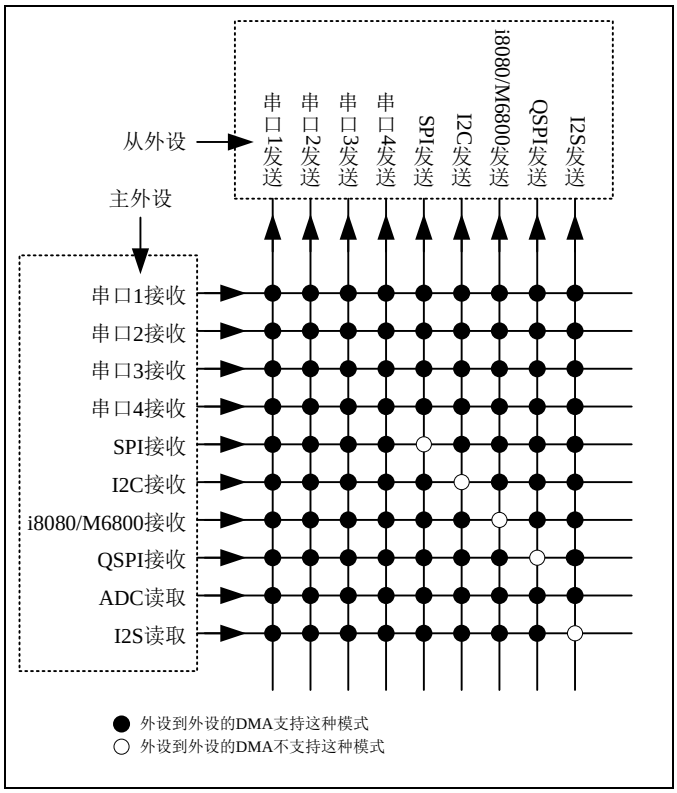
支持如下几种 DMA 操作：

- M2M_DMA：XRAM 存储器到 XRAM 存储器的数据读写
- ADC_DMA：自动扫描使能的 ADC 通道并将转换的 ADC 数据自动存储到 XRAM 中
- SPI_DMA：自动将 XRAM 中的数据和 SPI 外设之间进行数据交换
- UR1T_DMA：自动将 XRAM 中的数据通过串口 1 发送出去
- UR1R_DMA：自动将串口 1 接收到的数据存储到 XRAM 中
- UR2T_DMA：自动将 XRAM 中的数据通过串口 2 发送出去
- UR2R_DMA：自动将串口 2 接收到的数据存储到 XRAM 中
- UR3T_DMA：自动将 XRAM 中的数据通过串口 3 发送出去
- UR3R_DMA：自动将串口 3 接收到的数据存储到 XRAM 中
- UR4T_DMA：自动将 XRAM 中的数据通过串口 4 发送出去
- UR4R_DMA：自动将串口 4 接收到的数据存储到 XRAM 中
- TFT 彩屏 DMA：自动将 XRAM 中的数据和 TFT 彩屏设备之间进行数据交换
- I2CT_DMA：自动将 XRAM 中的数据通过 I2C 接口发送出去
- I2CR_DMA：自动将 I2C 接收到的数据存储到 XRAM 中
- I2ST_DMA：自动将 XRAM 中的数据通过 I2S 发送出去
- I2SR_DMA：自动将 I2S 接收到的数据存储到 XRAM 中
- QSPI_DMA：自动将 XRAM 中的数据和 QSPI 外设之间进行数据交换
- PWMAT_DMA：自动将 XRAM 中的数据通过 PWMA 接口发送出去
- PWMAR_DMA：自动将 PWMA 接收到的数据存储到 XRAM 中
- P2P：自动将源外设接收到的数据透传到目标外设（支持两组 P2P）

每次 DMA 数据传输最大数据量为 65535 字节,即将 AMT 设置为 0xFFFE；当将 AMT 设置为 0xFFFF 时，则 DMA 为**无限循环模式**。（存储器到存储器的 DMA 没有无限循环模式）

每种 DMA 对 XRAM 的读写操作都可设置 4 级访问优先级，硬件自动进行 XRAM 总线的访问仲裁，不会影响 CPU 的 XRAM 的访问。相同优先级下，不同 DMA 对 XRAM 的访问顺序如下：M2M_DMA，ADC_DMA，SPI_DMA，UR1R_DMA，UR1T_DMA，UR2R_DMA，UR2T_DMA，UR3R_DMA，UR3T_DMA，UR4R_DMA，UR4T_DMA，TFT 彩屏 DMA，I2CR_DMA，I2CT_DMA，I2SR_DMA，I2ST_DMA，QSPI_DMA，PWMAR_DMA，PWMAT_DMA

34.1　外设到外设（P2P）DMA 矩阵



P2P 使用注意事项：

目前测试发现 Ai8051U 的 QSPI 到其他外设的 P2P 方式 DMA 会影响 XRAM 的数据，在使用 QSPI 到其他外设的 P2P 功能时，**必须将 QSPI 存储接收数据的 XRAM 起始地址设置到 32K　XRAM 的后面部分。注意：由于内部的 XRAM 只有 32K，有效的地址范围为 0000H~7FFFH，当地址超出 32K 时，地址会重新回到 0，所以设置的 DMA 的 XRAM 起始地址+DMA 的传输数据长度的总和不能大于 32K。**

【例如】当使用 QSPI 到 i8080 彩屏接口的 P2P 功能时，可将{DMA_QSPI_RXAH,　DMA_QSPI_RXAL}中的地址设置到 **20K** 开始的地方，地址为 **5000H**。由于地址设置在 20K 起始的地方，所以每次 P2P 的最大传输数据量不能大于 **32K-20K=12K** 字节。

另外，寄存器{DMA_QSPI_RXAH,　DMA_QSPI_RXAL}中设置的是每次 DMA 读取 XRAM 的起始地址，在 DMA 过程中{DMA_QSPI_RXAH,　DMA_QSPI_RXAL}的值不会改变，所以只需要在初始化时设置好{DMA_QSPI_RXAH,　DMA_QSPI_RXAL}中的初始值就可以了，不需要每次 DMA 时都设置。

34.2 存储器与存储器之间的数据读写（M2M_DMA）

34.2.1 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
DMA_M2M_CFG	M2M_DMA 配置寄存器	7EFA00H	M2MIE	-	TXACO	RXACO	M2MIP[1:0]		M2MPTY[1:0]		0x00,0000
DMA_M2M_CR	M2M_DMA 控制寄存器	7EFA01H	ENM2M	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_M2M_STA	M2M_DMA 状态寄存器	7EFA02H	-	-	-	-	-	-	-	M2MIF	xxxx,xxx0
DMA_M2M_AMT	M2M_DMA 传输总字节数	7EFA03H	AMT[7:0]								0000,0000
DMA_M2M_DONE	M2M_DMA 传输完成字节数	7EFA04H	DONE[7:0]								0000,0000
DMA_M2M_TXAH	M2M_DMA 发送高地址	7EFA05H	ADR[15:0]								0000,0000
DMA_M2M_TXAL	M2M_DMA 发送低地址	7EFA06H	ADR[7:0]								0000,0000
DMA_M2M_RXAH	M2M_DMA 接收高地址	7EFA07H	ADR[15:0]								0000,0000
DMA_M2M_RXAL	M2M_DMA 接收低地址	7EFA08H	ADR[7:0]								0000,0000
DMA_M2M_AMTH	M2M_DMA 传输总字节数	7EFA80H	AMT[15:8]								0000,0000
DMA_M2M_DONEH	M2M_DMA 传输完成字节数	7EFA81H	DONE[15:8]								0000,0000

34.2.2 M2M_DMA 配置寄存器（DMA_M2M_CFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_CFG	7EFA00H	M2MIE	-	TXACO	RXACO	M2MIP[1:0]		M2MPTY[1:0]	

M2MIE：M2M_DMA 中断使能控制位

0：禁止 M2M_DMA 中断

1：允许 M2M_DMA 中断

TXACO：M2M_DMA 源地址（读取地址）改变方向

0：数据读取完成后地址自动递增

1：数据读取完成后地址自动递减

RXACO：M2M_DMA 目标地址（写入地址）改变方向

0：数据写入完成后地址自动递增

1：数据写入完成后地址自动递减

M2MIP[1:0]：M2M_DMA 中断优先级控制位

M2MIP[1:0]	中断优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

M2MPTY[1:0]：M2M_DMA 数据总线访问优先级控制位

M2MPTY [1:0]	总线优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

34.2.3 M2M_DMA 控制寄存器（DMA_M2M_CR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_CR	7EFA01H	ENM2M	TRIG	-	-	-	-	-	-

ENM2M：M2M_DMA 功能使能控制位

0：禁止 M2M_DMA 功能

1：允许 M2M_DMA 功能

TRIG：M2M_DMA 数据读写触发控制位

0：写 0 无效

1：写 1 开始 M2M_DMA 操作，

34.2.4 M2M_DMA 状态寄存器（DMA_M2M_STA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_STA	7EFA02H	-	-	-	-	-	-	-	M2MIF

M2MIF：M2M_DMA 中断请求标志位，当 M2M_DMA 操作完成后，硬件自动将 M2MIF 置 1，若使能 M2M_DMA 中断则进入中断服务程序。标志位需软件清零

34.2.5 M2M_DMA 传输总字节寄存器（DMA_M2M_AMT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_AMT	7EFA03H	AMT[7:0]							
DMA_M2M_AMTH	7EFA80H	AMT[15:8]							

AMT[15:0]：设置需要进行数据读写的字节数。

注：实际读写的字节数为（AMT+1），即当 AMT 设置为 0 时，读写 1 字节，当 AMT 设置 255 时，读写 256 字节。特别注意 AMT 不能设置为 FFFFH。

34.2.6 M2M_DMA 传输完成字节寄存器（DMA_M2M_DONE）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_DONE	7EFA04H	DONE[7:0]							
DMA_M2M_DONEH	7EFA81H	DONE[15:8]							

DONE[15:0]：当前已经读写完成的字节数。

34.2.7 M2M_DMA 发送地址寄存器（DMA_M2M_TXAx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_TXAH	7EFA05H	ADDR[15:8]							
DMA_M2M_TXAL	7EFA06H	ADDR[7:0]							

DMA_M2M_TXA：设置进行数据读写时的源地址。执行 M2M_DMA 操作时会从这个地址开始读数据。

34.2.8 M2M_DMA 接收地址寄存器（DMA_M2M_RXAx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_RXAH	7EFA07H	ADDR[15:8]							
DMA_M2M_RXAL	7EFA08H	ADDR[7:0]							

DMA_M2M_RXA：设置进行数据读写时的目标地址。执行 M2M_DMA 操作时会从这个地址开始写入数据。

34.3 ADC 数据自动存储（ADC_DMA）

34.3.1 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
DMA_ADC_CFG	ADC_DMA 配置寄存器	7EFA10H	ADCIE	-	-	-	ADCMIP[1:0]		ADCPTY[1:0]		0xxx,0000
DMA_ADC_CR	ADC_DMA 控制寄存器	7EFA11H	ENADC	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_ADC_STA	ADC_DMA 状态寄存器	7EFA12H	-	-	-	-	-	-	-	ADCIF	xxxx,xxx0
DMA_ADC_AMT	ADC_DMA 传输总字节数	7EFA13H	AMT[7:0]								0000,0000
DMA_ADC_DONE	ADC_DMA 传输完成字节数	7EFA14H	DONE[7:0]								0000,0000
DMA_ADC_RXAH	ADC_DMA 接收高地址	7EFA17H	ADR[15:0]								0000,0000
DMA_ADC_RXAL	ADC_DMA 接收低地址	7EFA18H	ADR[7:0]								0000,0000
DMA_ADC_CFG2	ADC_DMA 配置寄存器 2	7EFA19H	-	-	-	-	CVTIMESEL[3:0]				xxxx,0000
DMA_ADC_CHSW0	ADC_DMA 通道使能	7EFA1AH	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0	0000,0001
DMA_ADC_CHSW1	ADC_DMA 通道使能	7EFA1BH	CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8	1000,0000
DMA_ADC_ITVH	ADC_DMA 时间间隔寄存器	7EFA1EH	ITV[15:8]								0000,0000
DMA_ADC_ITVL	ADC_DMA 时间间隔寄存器	7EFA1FH	ITV[7:0]								0000,0000
DMA_ADC_AMTH	ADC_DMA 传输总字节数	7EFA82H	AMT[15:8]								0000,0000
DMA_ADC_DONEH	ADC_DMA 传输完成字节数	7EFA83H	DONE[15:8]								0000,0000

34.3.2 ADC_DMA 配置寄存器（DMA_ADC_CFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_CFG	7EFA10H	ADCIE	-			ADCIP[1:0]		ADCPTY[1:0]	

ADCIE：ADC_DMA 中断使能控制位

0：禁止 ADC_DMA 中断

1：允许 ADC_DMA 中断

ADCIP[1:0]：ADC_DMA 中断优先级控制位

ADCIP[1:0]	中断优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

ADCPTY[1:0]：ADC_DMA 数据总线访问优先级控制位

ADCPTY [1:0]	总线优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

34.3.3 ADC_DMA 控制寄存器（DMA_ADC_CR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_CR	7EFA11H	ENADC	TRIG	-	-	-	-	-	-

ENADC：ADC_DMA 功能使能控制位

0：禁止 ADC_DMA 功能

1：允许 ADC_DMA 功能

TRIG：ADC_DMA 操作触发控制位

0：写 0 无效

1：写 1 开始 ADC_DMA 操作，

34.3.4 ADC_DMA 状态寄存器（DMA_ADC_STA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_STA	7EFA12H	-	-	-	-	-	-	-	ADCIF

ADCIF：ADC_DMA 中断请求标志位，当 ADC_DMA 完成扫描所有使能的 ADC 通道后后，硬件自动将 ADCIF 置 1，若使能 ADC_DMA 中断则进入中断服务程序。标志位需软件清零

34.3.5 ADC_DMA 循环扫描总次数寄存器（DMA_ADC_AMT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_AMT	7EFA13H	AMT[7:0]							
DMA_ADC_AMTH	7EFA82H	AMT[15:8]							

AMT[15:0]：设置需要循环扫描 ADC 通道的总次数。

注：将 DMA_ADC_CHSW0 和 DMA_ADC_CHSW1 所使能的 ADC 通道全部扫描一遍为一次。实际扫描的次数为（AMT+1），即当 AMT 设置为 0 时，扫描 1 次，当 AMT 设置 255 时，扫描 256 次。特别注意 AMT 设置为 FFFFH 时表示无限循环扫描模式。

34.3.6 ADC_DMA 完成扫描次数寄存器（DMA_ADC_DONE）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_DONE	7EFA14H	DONE[7:0]							
DMA_ADC_DONEH	7EFA83H	DONE[15:8]							

DONE[15:0]：当前已经完成 ADC 扫描的次数。

34.3.7 ADC_DMA 接收地址寄存器（DMA_ADC_RXAx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_RXAH	7EFA17H	ADDR[15:8]							
DMA_ADC_RXAL	7EFA18H	ADDR[7:0]							

DMA_ADC_RXA：设置进行 ADC_DMA 操作时 ADC 转换数据的存储地址。

34.3.8 ADC_DMA 配置寄存器 2（DMA_ADC_CFG2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_CFG2	7EFA19H	-	-	-	-	CVTIMESEL[3:0]			

CVTIMESEL[3:0]：设置进行 ADC_DMA 操作时，对每个 ADC 通道进行 ADC 转换的次数

CVTIMESEL[3:0]	转换次数	CVTIMESEL[3:0]	转换次数
0xxx	1 次		
1000	2 次	1100	32 次
1001	4 次	1101	64 次
1010	8 次	1110	128 次
1011	16 次	1111	256 次

34.3.9 ADC_DMA 通道使能寄存器（DMA_ADC_CHSWx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_CHSW0	7EFA1AH	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
DMA_ADC_CHSW1	7EFA1BH	CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8

CHn：设置 ADC_DMA 操作时，自动扫描的 ADC 通道。通道扫描总是从编号小的通道开始。

34.3.10 ADC_DMA 转换周期（DMA_ADC_ITVx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_ITVH	7EFA1EH	ITV[15:8]							
DMA_ADC_ITVL	7EFA1FH	ITV[7:0]							

DMA_ADC_ITV[15:0]：设置 ADC 转换的间隔时间（基于系统时钟）。

注：DMA_ADC_ITV 设置的是两次 ADC 转换之间的间隔时间。

34.3.11 ADC_DMA 的数据存储格式

注：ADC 转换速度和转换结果的对齐方式均由 ADC 相关寄存器进行设置

XRAM[DMA_ADC_RXA+0] = 使能的第 1 通道的第 1 次 ADC 转换结果的高字节;
XRAM[DMA_ADC_RXA+1] = 使能的第 1 通道的第 1 次 ADC 转换结果的低字节;
XRAM[DMA_ADC_RXA+2] = 使能的第 1 通道的第 2 次 ADC 转换结果的高字节;
XRAM[DMA_ADC_RXA+3] = 使能的第 1 通道的第 2 次 ADC 转换结果的低字节;
...
XRAM[DMA_ADC_RXA+2n-2] = 使能的第 1 通道的第 n 次 ADC 转换结果的高字节;
XRAM[DMA_ADC_RXA+2n-1] = 使能的第 1 通道的第 n 次 ADC 转换结果的低字节;
XRAM[DMA_ADC_RXA+2n] = 第 1 通道的 ADC 通道号;
XRAM[DMA_ADC_RXA+2n+1] = 第 1 通道 n 次 ADC 转换结果取完平均值之后的余数;
XRAM[DMA_ADC_RXA+2n+2] = 第 1 通道 n 次 ADC 转换结果平均值的高字节;
XRAM[DMA_ADC_RXA+2n+3] = 第 1 通道 n 次 ADC 转换结果平均值的低字节;

XRAM[DMA_ADC_RXA+(2n+4)+0] = 使能的第 2 通道的第 1 次 ADC 转换结果的高字节;
XRAM[DMA_ADC_RXA+(2n+4)+1] = 使能的第 2 通道的第 1 次 ADC 转换结果的低字节;
XRAM[DMA_ADC_RXA+(2n+4)+2] = 使能的第 2 通道的第 2 次 ADC 转换结果的高字节;
XRAM[DMA_ADC_RXA+(2n+4)+3] = 使能的第 2 通道的第 2 次 ADC 转换结果的低字节;
...
XRAM[DMA_ADC_RXA+(2n+4)+2n-2] = 使能的第 2 通道的第 n 次 ADC 转换结果的高字节;
XRAM[DMA_ADC_RXA+(2n+4)+2n-1] = 使能的第 2 通道的第 n 次 ADC 转换结果的低字节;
XRAM[DMA_ADC_RXA+(2n+4)+2n] = 第 2 通道的 ADC 通道号;
XRAM[DMA_ADC_RXA+(2n+4)+2n+1] = 第 2 通道的 n 次 ADC 转换结果取完平均值之后的余数;
XRAM[DMA_ADC_RXA+(2n+4)+2n+2] =第 2 通道的 n 次 ADC 转换结果平均值的高字节;
XRAM[DMA_ADC_RXA+(2n+4)+2n+3] =第 2 通道的 n 次 ADC 转换结果平均值的低字节;
...
XRAM[DMA_ADC_RXA+(m-1)(2n+4)+0] = 使能的第 m 通道的第 1 次 ADC 转换结果的高字节;
XRAM[DMA_ADC_RXA+(m-1)(2n+4)+1] = 使能的第 m 通道的第 1 次 ADC 转换结果的低字节;
XRAM[DMA_ADC_RXA+(m-1)(2n+4)+2] = 使能的第 m 通道的第 2 次 ADC 转换结果的高字节;
XRAM[DMA_ADC_RXA+(m-1)(2n+4)+3] = 使能的第 m 通道的第 2 次 ADC 转换结果的低字节;
...
XRAM[DMA_ADC_RXA+(m-1)(2n+4)+2n-2] = 使能的第 m 通道的第 n 次 ADC 转换结果的高字节;
XRAM[DMA_ADC_RXA+(m-1)(2n+4)+2n-1] = 使能的第 m 通道的第 n 次 ADC 转换结果的低字节;
XRAM[DMA_ADC_RXA+(m-1)(2n+4)+2n] = 第 m 通道的 ADC 通道号;
XRAM[DMA_ADC_RXA+(m-1)(2n+4)+2n+1] = 第 m 通道的 n 次 ADC 转换结果取完平均值之后的余数;
XRAM[DMA_ADC_RXA+(m-1)(2n+4)+2n+2] =第 m 通道的 n 次 ADC 转换结果平均值的高字节;
XRAM[DMA_ADC_RXA+(m-1)(2n+4)+2n+3] =第 m 通道的 n 次 ADC 转换结果平均值的低字节;

表格形式如下：

ADC 通道	偏移地址	数据
第 1 通道	0	使能的第 1 通道的第 1 次 ADC 转换结果的高字节
	1	使能的第 1 通道的第 1 次 ADC 转换结果的低字节
	2	使能的第 1 通道的第 2 次 ADC 转换结果的高字节
	3	使能的第 1 通道的第 2 次 ADC 转换结果的低字节
	...	...
	2n-2	使能的第 1 通道的第 n 次 ADC 转换结果的高字节
	2n-1	使能的第 1 通道的第 n 次 ADC 转换结果的低字节
	2n	第 1 通道的 ADC 通道号
	2n+1	第 1 通道 n 次 ADC 转换结果取完平均值之后的余数
	2n+2	第 1 通道 n 次 ADC 转换结果平均值的高字节
	2n+3	第 1 通道 n 次 ADC 转换结果平均值的低字节
第 2 通道	(2n+4) + 0	使能的第 2 通道的第 1 次 ADC 转换结果的高字节
	(2n+4) + 1	使能的第 2 通道的第 1 次 ADC 转换结果的低字节
	(2n+4) + 2	使能的第 2 通道的第 2 次 ADC 转换结果的高字节
	(2n+4) + 3	使能的第 2 通道的第 2 次 ADC 转换结果的低字节
	...	...
	(2n+4) + 2n-2	使能的第 2 通道的第 n 次 ADC 转换结果的高字节
	(2n+4) + 2n-1	使能的第 2 通道的第 n 次 ADC 转换结果的低字节
	(2n+4) + 2n	第 2 通道的 ADC 通道号
	(2n+4) + 2n+1	第 2 通道 n 次 ADC 转换结果取完平均值之后的余数
	(2n+4) + 2n+2	第 2 通道 n 次 ADC 转换结果平均值的高字节
	(2n+4) + 2n+3	第 2 通道 n 次 ADC 转换结果平均值的低字节
	...	...
第 m 通道	(m-1)(2n+4) + 0	使能的第 m 通道的第 1 次 ADC 转换结果的高字节
	(m-1)(2n+4) + 1	使能的第 m 通道的第 1 次 ADC 转换结果的低字节
	(m-1)(2n+4) + 2	使能的第 m 通道的第 2 次 ADC 转换结果的高字节
	(m-1)(2n+4) + 3	使能的第 m 通道的第 2 次 ADC 转换结果的低字节
	...	...
	(m-1)(2n+4) + 2n-2	使能的第 m 通道的第 n 次 ADC 转换结果的高字节
	(m-1)(2n+4) + 2n-1	使能的第 m 通道的第 n 次 ADC 转换结果的低字节
	(m-1)(2n+4) + 2n	第 m 通道的 ADC 通道号
	(m-1)(2n+4) + 2n+1	第 m 通道 n 次 ADC 转换结果取完平均值之后的余数
	(m-1)(2n+4) + 2n+2	第 m 通道 n 次 ADC 转换结果平均值的高字节
	(m-1)(2n+4) + 2n+3	第 m 通道 n 次 ADC 转换结果平均值的低字节

34.4 SPI 与存储器之间的数据交换（SPI_DMA）

34.4.1 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
DMA_SPI_CFG	SPI_DMA 配置寄存器	7EFA20H	SPIIE	ACT_TX	ACT_RX	-	SPIIP[1:0]		SPIPTY[1:0]		000x,0000
DMA_SPI_CR	SPI_DMA 控制寄存器	7EFA21H	ENSPI	TRIG_M	TRIG_S	-	-	-	-	CLRFIFO	000x,xxx0
DMA_SPI_STA	SPI_DMA 状态寄存器	7EFA22H	-	-	-	-	-	TXOVW	RXLOSS	SPIIF	xxxx,x000
DMA_SPI_AMT	SPI_DMA 传输总字节数	7EFA23H	AMT[7:0]								0000,0000
DMA_SPI_DONE	SPI_DMA 传输完成字节数	7EFA24H	DONE[7:0]								0000,0000
DMA_SPI_TXAH	SPI_DMA 发送高地址	7EFA25H	ADR[15:0]								0000,0000
DMA_SPI_TXAL	SPI_DMA 发送低地址	7EFA26H	ADR[7:0]								0000,0000
DMA_SPI_RXAH	SPI_DMA 接收高地址	7EFA27H	ADR[15:0]								0000,0000
DMA_SPI_RXAL	SPI_DMA 接收低地址	7EFA28H	ADR[7:0]								0000,0000
DMA_SPI_CFG2	SPI_DMA 配置寄存器 2	7EFA29H	-	-	-	-	-	WRPSS	SSS[1:0]		xxxx,x000
DMA_SPI_ITVH	SPI_DMA 时间间隔寄存器	7EFA2EH	ITV[15:8]								0000,0000
DMA_SPI_ITVL	SPI_DMA 时间间隔寄存器	7EFA2FH	ITV[7:0]								0000,0000
DMA_SPI_AMTH	SPI_DMA 传输总字节数	7EFA84H	AMT[15:8]								0000,0000
DMA_SPI_DONEH	SPI_DMA 传输完成字节数	7EFA85H	DONE[15:8]								0000,0000

34.4.2 SPI_DMA 配置寄存器（DMA_SPI_CFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_CFG	7EFA20H	SPIIE	ACT_TX	ACT_RX	-	SPIIP[1:0]		SPIPTY[1:0]	

SPIIE：SPI_DMA 中断使能控制位

0：禁止 SPI_DMA 中断

1：允许 SPI_DMA 中断

ACT_TX：SPI_DMA 发送数据控制位

0：禁止 SPI_DMA 发送数据。主机模式时，SPI 只发送时钟到 SCLK 端口，但不从 XRAM 读取数据，也不向 MOSI 端口上发送数据；从机模式时，SPI 不从 XRAM 读取数据，也不向 MISO 端口上发送数据

1：允许 SPI_DMA 发送数据。主机模式时，SPI 发送时钟到 SCLK 端口，同时从 XRAM 读取数据，并将数据发送到 MOSI 端口；从机模式时，SPI 从 XRAM 读取数据，并将数据发送到 MISO 端口

ACT_RX：SPI_DMA 接收数据控制位

0：禁止 SPI_DMA 接收数据。主机模式时，SPI 只发送时钟到 SCLK 端口，但不从 MISO 端口读取数据，也不向 XRAM 写数据；从机模式时，SPI 不从 MOSI 端口读取数据，也不向 XRAM 写数据。

1：允许 SPI_DMA 接收数据。主机模式时，SPI 发送时钟到 SCLK 端口，同时从 MISO 端口读取数据，并将数据写入 XRAM；从机模式时，SPI 从 MOSI 端口读取数据，并写入 XRAM。

SPIIP[1:0]：SPI_DMA 中断优先级控制位

SPIIP[1:0]	中断优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

SPIPTY[1:0]：SPI_DMA 数据总线访问优先级控制位

SPIPTY [1:0]	总线优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

34.4.3 SPI_DMA 控制寄存器（DMA_SPI_CR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_CR	7EFA21H	ENSPI	TRIG_M	TRIG_S	-	-	-	-	CLRFIFO

ENSPI：SPI_DMA 功能使能控制位

0：禁止 SPI_DMA 功能

1：允许 SPI_DMA 功能

TRIG_M：SPI_DMA 主机模式触发控制位

0：写 0 无效

1：写 1 开始 SPI_DMA 主机模式操作，

TRIG_S：SPI_DMA 从机模式触发控制位

0：写 0 无效

1：写 1 开始 SPI_DMA 从机模式操作，

CLRFIFO：清除 SPI_DMA 接收 FIFO 控制位

0：写 0 无效

1：开始 SPI_DMA 操作前，先清空 SPI_DMA 内置的 FIFO

34.4.4 SPI_DMA 状态寄存器（DMA_SPI_STA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_STA	7EFA22H	-	-	-	-	-	TXOVW	RXLOSS	SPIIF

SPIIF：SPI_DMA 中断请求标志位，当 SPI_DMA 数据交换完成后，硬件自动将 SPIIF 置 1，若使能 SPI_DMA 中断则进入中断服务程序。标志位需软件清零

RXLOSS：SPI_DMA 接收数据丢弃标志位。SPI_DMA 操作过程中，当 XRAM 总线过于繁忙，来不及清空 SPI_DMA 的接收 FIFO 导致 SPI_DMA 接收的数据自动丢弃时，硬件硬件自动将 RXLOSS 置 1。标志位需软件清零

TXOVW：SPI_DMA 数据覆盖标志位。SPI_DMA 正在数据传输过程中，主机模式的 SPI 写 SPDAT 寄存器再次触发 SPI 数据传输时，会导致数据传输失败，此时硬件硬件自动将 TXOVW 置 1。标志位需软件清零

34.4.5 SPI_DMA 传输总字节寄存器（DMA_SPI_AMT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_AMT	7EFA23H	AMT[7:0]							
DMA_SPI_AMTH	7EFA84H	AMT[15:8]							

AMT[15:0]：设置需要进行数据读写的字节数。

注：实际读写的字节数为（AMT+1），即当 AMT 设置为 0 时，读写 1 字节，当 AMT 设置 255 时，读写 256 字节。特别注意 AMT 设置为 FFFFH 时表示无限循环模式。

34.4.6 SPI_DMA 传输完成字节寄存器（DMA_SPI_DONE）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_DONE	7EFA24H	DONE[7:0]							
DMA_SPI_DONEH	7EFA85H	DONE[15:8]							

DONE[15:0]：当前已经传输完成的字节数。

34.4.7 SPI_DMA 发送地址寄存器（DMA_SPI_TXAx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_TXAH	7EFA25H	ADDR[15:8]							
DMA_SPI_TXAL	7EFA26H	ADDR[7:0]							

DMA_SPI_TXA：设置进行数据传输时的源地址。执行 SPI_DMA 操作时会从这个地址开始读数据。

34.4.8 SPI_DMA 接收地址寄存器（DMA_SPI_RXAx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_RXAH	7EFA27H	ADDR[15:8]							
DMA_SPI_RXAL	7EFA28H	ADDR[7:0]							

DMA_SPI_RXA：设置进行数据传输时的目标地址。执行 SPI_DMA 操作时会从这个地址开始写入数据。

34.4.9 SPI_DMA 配置寄存器 2（DMA_SPI_CFG2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_CFG2	7EFA29H	-	-	-	-	-	WRPSS	SSS[1:0]	

WRPSS：SPI_DMA 过程中使能 SS 脚控制位

0：SPI_DMA 传输过程中，不自动控制 SS 脚

1：SPI_DMA 传输过程中，自动拉低 SS 脚，传输完成后，自动恢复原始状态

SSS[1:0]：SPI_DMA 过程中，自动控制 SS 选择位

SSS[1:0]	SS 脚
00	P1.4
01	P2.4
10	P4.0
11	P3.5

34.4.10 SPI_DMA 传输周期（DMA_SPI_ITVx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_ITVH	7EFA2EH	ITV[15:8]							
DMA_SPI_ITVL	7EFA2FH	ITV[7:0]							

DMA_SPI_ITV[15:0]：设置 SPI 传输的间隔时间（基于系统时钟）。

34.5 串口 1 与存储器之间的数据交换（UR1T_DMA，UR1R_DMA）

34.5.1 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
DMA_UR1T_CFG	UR1T_DMA 配置寄存器	7EFA30H	UR1TIE	-	-	-	UR1TIP[1:0]		UR1TPTY[1:0]		0xxx,0000
DMA_UR1T_CR	UR1T_DMA 控制寄存器	7EFA31H	ENUR1T	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_UR1T_STA	UR1T_DMA 状态寄存器	7EFA32H	-	-	-	-	-	TXOVW	-	UR1TIF	xxxx,x0x0
DMA_UR1T_AMT	UR1T_DMA 传输总字节数	7EFA33H	AMT[7:0]								0000,0000
DMA_UR1T_DONE	UR1T_DMA 传输完成字节数	7EFA34H	DONE[7:0]								0000,0000
DMA_UR1T_TXAH	UR1T_DMA 发送高地址	7EFA35H	ADR[15:0]								0000,0000
DMA_UR1T_TXAL	UR1T_DMA 发送低地址	7EFA36H	ADR[7:0]								0000,0000
DMA_UR1R_CFG	UR1R_DMA 配置寄存器	7EFA38H	UR1RIE	-	-	-	UR1RIP[1:0]		UR1RPTY[1:0]		0xxx,0000
DMA_UR1R_CR	UR1R_DMA 控制寄存器	7EFA39H	ENUR1R	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0
DMA_UR1R_STA	UR1R_DMA 状态寄存器	7EFA3AH	-	-	-	-	-	-	RXLOSS	UR1RIF	xxxx,xx00
DMA_UR1R_AMT	UR1R_DMA 传输总字节数	7EFA3BH	AMT[7:0]								0000,0000
DMA_UR1R_DONE	UR1R_DMA 传输完成字节数	7EFA3CH	DONE[7:0]								0000,0000
DMA_UR1R_RXAH	UR1R_DMA 接收高地址	7EFA3DH	ADR[15:0]								0000,0000
DMA_UR1R_RXAL	UR1R_DMA 接收低地址	7EFA3EH	ADR[7:0]								0000,0000
DMA_UR1T_AMTH	UR1T_DMA 传输总字节数	7EFA88H	AMT[15:8]								0000,0000
DMA_UR1T_DONEH	UR1T_DMA 传输完成字节数	7EFA89H	DONE[15:8]								0000,0000
DMA_UR1R_AMTH	UR1R_DMA 传输总字节数	7EFA8AH	AMT[15:8]								0000,0000
DMA_UR1R_DONEH	UR1R_DMA 传输完成字节数	7EFA8BH	DONE[15:8]								0000,0000
DMA_UR1_ITVH	UR1_DMA 时间间隔寄存器	7EFAC8H	ITV[15:8]								0000,0000
DMA_UR1_ITVL	UR1_DMA 时间间隔寄存器	7EFAC9H	ITV[7:0]								0000,0000

34.5.2 UR1T_DMA 配置寄存器（DMA_UR1T_CFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_CFG	7EFA30H	UR1TIE	-	-	-	UR1TIP[1:0]		UR1TPTY[1:0]	

UR1TIE：UR1T_DMA 中断使能控制位

0：禁止 UR1T_DMA 中断

1：允许 UR1T_DMA 中断

UR1TIP[1:0]：UR1T_DMA 中断优先级控制位

UR1TIP[1:0]	中断优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

UR1TPTY[1:0]：UR1T_DMA 数据总线访问优先级控制位

UR1TPTY [1:0]	总线优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

34.5.3 UR1T_DMA 控制寄存器（DMA_UR1T_CR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_CR	7EFA31H	ENUR1T	TRIG	-	-	-	-	-	-

ENUR1T：UR1T_DMA 功能使能控制位

0：禁止 UR1T_DMA 功能

1：允许 UR1T_DMA 功能

TRIG：UR1T_DMA 发送触发控制位

0：写 0 无效

1：写 1 开始 UR1T_DMA 自动发送数据

34.5.4 UR1T_DMA 状态寄存器（DMA_UR1T_STA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_STA	7EFA32H	-	-	-	-	-	TXOVW	-	UR1TIF

UR1TIF：UR1T_DMA 中断请求标志位，当 UR1T_DMA 数据发送完成后，硬件自动将 UR1TIF 置 1，若使能 UR1T_DMA 中断则进入中断服务程序。标志位需软件清零

TXOVW：UR1T_DMA 数据覆盖标志位。UR1T_DMA 正在数据传输过程中，串口写 SBUF 寄存器再次触发串口发送数据时，会导致数据传输失败，此时硬件硬件自动将 TXOVW 置 1。标志位需软件清零

34.5.5 UR1T_DMA 传输总字节寄存器（DMA_UR1T_AMT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_AMT	7EFA33H	AMT[7:0]							
DMA_UR1T_AMTH	7EFA88H	AMT[15:8]							

AMT[15:0]：设置需要进行数据读写的字节数。

注：实际读写的字节数为（AMT+1），即当 AMT 设置为 0 时，读写 1 字节，当 AMT 设置 255 时，读写 256 字节。特别注意 AMT 设置为 FFFFH 时表示无限循环模式。

34.5.6 UR1T_DMA 传输完成字节寄存器（DMA_UR1T_DONE）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_DONE	7EFA34H	DONE[7:0]							
DMA_UR1T_DONEH	7EFA89H	DONE[15:8]							

DONE[15:0]：当前已经发送完成的字节数。

34.5.7 UR1T_DMA 发送地址寄存器（DMA_UR1T_TXAx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_TXAH	7EFA35H	ADDR[15:8]							
DMA_UR1T_TXAL	7EFA36H	ADDR[7:0]							

DMA_UR1T_TXA：设置自动发送数据的源地址。执行 UR1T_DMA 操作时会从这个地址开始读数据。

34.5.8 UR1_DMA 传输周期（DMA_UR1_ITVx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1_ITVH	7EFAC8H	ITV[15:8]							
DMA_UR1_ITVL	7EFAC9H	ITV[7:0]							

DMA_UR1_ITV[15:0]：设置 UART1 发送和接收的间隔时间（基于系统时钟）。

34.5.9 UR1R_DMA 配置寄存器（DMA_UR1R_CFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1R_CFG	7EFA38H	UR1RIE	-	-	-	UR1RIP[1:0]		UR1RPTY[1:0]	

UR1RIE：UR1R_DMA 中断使能控制位

0：禁止 UR1R_DMA 中断

1：允许 UR1R_DMA 中断

UR1RIP[1:0]：UR1R_DMA 中断优先级控制位

UR1RIP[1:0]	中断优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

UR1RPTY[1:0]：UR1R_DMA 数据总线访问优先级控制位

UR1RPTY [1:0]	总线优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

34.5.10 UR1R_DMA 控制寄存器（DMA_UR1R_CR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1R_CR	7EFA39H	ENUR1R	-	TRIG	-	-	-	-	CLRFIFO

ENUR1R：UR1R_DMA 功能使能控制位

0：禁止 UR1R_DMA 功能

1：允许 UR1R_DMA 功能

TRIG：UR1R_DMA 接收触发控制位

0：写 0 无效

1：写 1 开始 UR1R_DMA 自动接收数据

CLRFIFO：清除 UR1R_DMA 接收 FIFO 控制位

0：写 0 无效

1：开始 UR1R_DMA 操作前，先清空 UR1R_DMA 内置的 FIFO

34.5.11 UR1R_DMA 状态寄存器（DMA_UR1R_STA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1R_STA	7EFA3AH	-	-	-	-	-	-	RXLOSS	UR1RIF

UR1RIF：UR1R_DMA 中断请求标志位，当 UR1R_DMA 接收数据完成后，硬件自动将 UR1RIF 置 1，若使能 UR1R_DMA 中断则进入中断服务程序。标志位需软件清零

RXLOSS：UR1R_DMA 接收数据丢弃标志位。UR1R_DMA 操作过程中，当 XRAM 总线过于繁忙，来不及清空 UR1R_DMA 的接收 FIFO 导致 UR1R_DMA 接收的数据自动丢弃时，硬件硬件自动将 RXLOSS 置 1。标志位需软件清零

34.5.12 UR1R_DMA 传输总字节寄存器（DMA_UR1R_AMT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1R_AMT	7EFA3BH	AMT[7:0]							
DMA_UR1R_AMTH	7EFA8AH	AMT[15:8]							

AMT[15:0]：设置需要进行数据读写的字节数。

注：实际读写的字节数为（AMT+1），即当 AMT 设置为 0 时，读写 1 字节，当 AMT 设置 255 时，读写 256 字节。特别注意 AMT 设置为 FFFFH 时表示无限循环模式。

34.5.13 UR1R_DMA 传输完成字节寄存器（DMA_UR1R_DONE）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1R_DONE	7EFA3CH	DONE[7:0]							
DMA_UR1R_DONEH	7EFA8BH	DONE[15:8]							

DONE[15:0]：当前已经接收完成的字节数。

34.5.14 UR1R_DMA 接收地址寄存器（DMA_UR1R_RXAx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1R_RXAH	7EFA3DH	ADDR[15:8]							
DMA_UR1R_RXAL	7EFA3EH	ADDR[7:0]							

DMA_UR1R_RXA：设置自动接收数据的目标地址。执行 UR1R_DMA 操作时会从这个地址开始写数据。

34.6 串口 2 与存储器之间的数据交换（UR2T_DMA，UR2R_DMA）

34.6.1 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
DMA_UR2T_CFG	UR2T_DMA 配置寄存器	7EFA40H	UR2TIE	-	-	-	UR2TIP[1:0]		UR2TPTY[1:0]		0xxx,0000
DMA_UR2T_CR	UR2T_DMA 控制寄存器	7EFA41H	ENUR2T	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_UR2T_STA	UR2T_DMA 状态寄存器	7EFA42H	-	-	-	-	-	TXOVW	-	UR2TIF	xxxx,x0x0
DMA_UR2T_AMT	UR2T_DMA 传输总字节数	7EFA43H	AMT[7:0]								0000,0000
DMA_UR2T_DONE	UR2T_DMA 传输完成字节数	7EFA44H	DONE[7:0]								0000,0000
DMA_UR2T_TXAH	UR2T_DMA 发送高地址	7EFA45H	ADR[15:0]								0000,0000
DMA_UR2T_TXAL	UR2T_DMA 发送低地址	7EFA46H	ADR[7:0]								0000,0000
DMA_UR2R_CFG	UR2R_DMA 配置寄存器	7EFA48H	UR2RIE	-	-	-	UR2RIP[1:0]		UR2RPTY[1:0]		0xxx,0000
DMA_UR2R_CR	UR2R_DMA 控制寄存器	7EFA49H	ENUR2R	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0
DMA_UR2R_STA	UR2R_DMA 状态寄存器	7EFA4AH	-	-	-	-	-	-	RXLOSS	UR2RIF	xxxx,xx00
DMA_UR2R_AMT	UR2R_DMA 传输总字节数	7EFA4BH	AMT[7:0]								0000,0000
DMA_UR2R_DONE	UR2R_DMA 传输完成字节数	7EFA4CH	DONE[7:0]								0000,0000
DMA_UR2R_RXAH	UR2R_DMA 接收高地址	7EFA4DH	ADR[15:0]								0000,0000
DMA_UR2R_RXAL	UR2R_DMA 接收低地址	7EFA4EH	ADR[7:0]								0000,0000
DMA_UR2T_AMTH	UR2T_DMA 传输总字节数	7EFA8CH	AMT[15:8]								0000,0000
DMA_UR2T_DONEH	UR2T_DMA 传输完成字节数	7EFA8DH	DONE[15:8]								0000,0000
DMA_UR2R_AMTH	UR2R_DMA 传输总字节数	7EFA8EH	AMT[15:8]								0000,0000
DMA_UR2R_DONEH	UR2R_DMA 传输完成字节数	7EFA8FH	DONE[15:8]								0000,0000
DMA_UR2_ITVH	UR2_DMA 时间间隔寄存器	7EFACAH	ITV[15:8]								0000,0000
DMA_UR2_ITVL	UR2_DMA 时间间隔寄存器	7EFACBH	ITV[7:0]								0000,0000

34.6.2 UR2T_DMA 配置寄存器（DMA_UR2T_CFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_CFG	7EFA40H	UR2TIE	-	-	-	UR2TIP[1:0]		UR2TPTY[1:0]	

UR2TIE：UR2T_DMA 中断使能控制位

0：禁止 UR2T_DMA 中断

1：允许 UR2T_DMA 中断

UR2TIP[1:0]：UR2T_DMA 中断优先级控制位

UR2TIP[1:0]	中断优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

UR2TPTY[1:0]：UR2T_DMA 数据总线访问优先级控制位

UR2TPTY [1:0]	总线优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

34.6.3 UR2T_DMA 控制寄存器（DMA_UR2T_CR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_CR	7EFA41H	ENUR2T	TRIG	-	-	-	-	-	-

ENUR2T：UR2T_DMA 功能使能控制位

0：禁止 UR2T_DMA 功能

1：允许 UR2T_DMA 功能

TRIG：UR2T_DMA 发送触发控制位

0：写 0 无效

1：写 1 开始 UR2T_DMA 自动发送数据

34.6.4 UR2T_DMA 状态寄存器（DMA_UR2T_STA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_STA	7EFA42H	-	-	-	-	-	TXOVW	-	UR2TIF

UR2TIF：UR2T_DMA 中断请求标志位，当 UR2T_DMA 数据发送完成后，硬件自动将 UR2TIF 置 1，若使能 UR2T_DMA 中断则进入中断服务程序。标志位需软件清零

TXOVW：UR2T_DMA 数据覆盖标志位。UR2T_DMA 正在数据传输过程中，串口写 S2BUF 寄存器再次触发串口发送数据时，会导致数据传输失败，此时硬件硬件自动将 TXOVW 置 1。标志位需软件清零

34.6.5 UR2T_DMA 传输总字节寄存器（DMA_UR2T_AMT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_AMT	7EFA43H	AMT[7:0]							
DMA_UR2T_AMTH	7EFA8CH	AMT[15:8]							

AMT[15:0]：设置需要进行数据读写的字节数。

注：实际读写的字节数为（AMT+1），即当 AMT 设置为 0 时，读写 1 字节，当 AMT 设置 255 时，读写 256 字节。特别注意 AMT 设置为 FFFFH 时表示无限循环模式。

34.6.6 UR2T_DMA 传输完成字节寄存器（DMA_UR2T_DONE）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_DONE	7EFA44H	DONE[7:0]							
DMA_UR2T_DONEH	7EFA8DH	DONE[15:8]							

DONE[15:0]：当前已经发送完成的字节数。

34.6.7 UR2T_DMA 发送地址寄存器（DMA_UR2T_TXAx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_TXAH	7EFA45H	ADDR[15:8]							
DMA_UR2T_TXAL	7EFA46H	ADDR[7:0]							

DMA_UR2T_TXA：设置自动发送数据的源地址。执行 UR2T_DMA 操作时会从这个地址开始读数据。

34.6.8 UR2_DMA 传输周期（DMA_UR2_ITVx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2_ITVH	7EFACAH	ITV[15:8]							
DMA_UR2_ITVL	7EFACBH	ITV[7:0]							

DMA_UR2_ITV[15:0]：设置 UART2 发送和接收的间隔时间（基于系统时钟）。

34.6.9 UR2R_DMA 配置寄存器（DMA_UR2R_CFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2R_CFG	7EFA48H	UR2RIE	-	-	-	UR2RIP[1:0]		UR2RPTY[1:0]	

UR2RIE：UR2R_DMA 中断使能控制位

0：禁止 UR2R_DMA 中断

1：允许 UR2R_DMA 中断

UR2RIP[1:0]：UR2R_DMA 中断优先级控制位

UR2RIP[1:0]	中断优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

UR2RPTY[1:0]：UR2R_DMA 数据总线访问优先级控制位

UR2RPTY [1:0]	总线优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

34.6.10 UR2R_DMA 控制寄存器（DMA_UR2R_CR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2R_CR	7EFA49H	ENUR2R	-	TRIG	-	-	-	-	CLRFIFO

ENUR2R：UR2R_DMA 功能使能控制位

0：禁止 UR2R_DMA 功能

1：允许 UR2R_DMA 功能

TRIG：UR2R_DMA 接收触发控制位

0：写 0 无效

1：写 1 开始 UR2R_DMA 自动接收数据

CLRFIFO：清除 UR2R_DMA 接收 FIFO 控制位

0：写 0 无效

1：开始 UR2R_DMA 操作前，先清空 UR2R_DMA 内置的 FIFO

34.6.11 UR2R_DMA 状态寄存器（DMA_UR2R_STA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2R_STA	7EFA4AH	-	-	-	-	-	-	RXLOSS	UR2RIF

UR2RIF：UR2R_DMA 中断请求标志位，当 UR2R_DMA 接收数据完成后，硬件自动将 UR2RIF 置 1，若使能 UR2R_DMA 中断则进入中断服务程序。标志位需软件清零

RXLOSS：UR2R_DMA 接收数据丢弃标志位。UR2R_DMA 操作过程中，当 XRAM 总线过于繁忙，来不及清空 UR2R_DMA 的接收 FIFO 导致 UR2R_DMA 接收的数据自动丢弃时，硬件硬件自动将 RXLOSS 置 1。标志位需软件清零

34.6.12 UR2R_DMA 传输总字节寄存器（DMA_UR2R_AMT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2R_AMT	7EFA4BH	AMT[7:0]							
DMA_UR2R_AMTH	7EFA8EH	AMT[15:8]							

AMT[15:0]：设置需要进行数据读写的字节数。

注：实际读写的字节数为（AMT+1），即当 AMT 设置为 0 时，读写 1 字节，当 AMT 设置 255 时，读写 256 字节。特别注意 AMT 设置为 FFFFH 时表示无限循环模式。

34.6.13 UR2R_DMA 传输完成字节寄存器（DMA_UR2R_DONE）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2R_DONE	7EFA4CH	DONE[7:0]							
DMA_UR2R_DONEH	7EFA8FH	DONE[15:8]							

DONE[15:0]：当前已经接收完成的字节数。

34.6.14 UR2R_DMA 接收地址寄存器（DMA_UR2R_RXAx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2R_RXAH	7EFA4DH	ADDR[15:8]							
DMA_UR2R_RXAL	7EFA4EH	ADDR[7:0]							

DMA_UR2R_RXA：设置自动接收数据的目标地址。执行 UR2R_DMA 操作时会从这个地址开始写数据。

34.7 串口 3 与存储器之间的数据交换（UR3T_DMA，UR3R_DMA）

34.7.1 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
DMA_UR3T_CFG	UR3T_DMA 配置寄存器	7EFA50H	UR3TIE	-	-	-	UR3TIP[1:0]		UR3TPTY[1:0]		0xxx,0000
DMA_UR3T_CR	UR3T_DMA 控制寄存器	7EFA51H	ENUR3T	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_UR3T_STA	UR3T_DMA 状态寄存器	7EFA52H	-	-	-	-	-	TXOVW	-	UR3TIF	xxxx,x0x0
DMA_UR3T_AMT	UR3T_DMA 传输总字节数	7EFA53H	AMT[7:0]								0000,0000
DMA_UR3T_DONE	UR3T_DMA 传输完成字节数	7EFA54H	DONE[7:0]								0000,0000
DMA_UR3T_TXAH	UR3T_DMA 发送高地址	7EFA55H	ADR[15:0]								0000,0000
DMA_UR3T_TXAL	UR3T_DMA 发送低地址	7EFA56H	ADR[7:0]								0000,0000
DMA_UR3R_CFG	UR3R_DMA 配置寄存器	7EFA58H	UR3RIE	-	-	-	UR3RIP[1:0]		UR3RPTY[1:0]		0xxx,0000
DMA_UR3R_CR	UR3R_DMA 控制寄存器	7EFA59H	ENUR3R	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0
DMA_UR3R_STA	UR3R_DMA 状态寄存器	7EFA5AH	-	-	-	-	-	-	RXLOSS	UR3RIF	xxxx,xx00
DMA_UR3R_AMT	UR3R_DMA 传输总字节数	7EFA5BH	AMT[7:0]								0000,0000
DMA_UR3R_DONE	UR3R_DMA 传输完成字节数	7EFA5CH	DONE[7:0]								0000,0000
DMA_UR3R_RXAH	UR3R_DMA 接收高地址	7EFA5DH	ADR[15:0]								0000,0000
DMA_UR3R_RXAL	UR3R_DMA 接收低地址	7EFA5EH	ADR[7:0]								0000,0000
DMA_UR3T_AMTH	UR3T_DMA 传输总字节数	7EFA90H	AMT[15:8]								0000,0000
DMA_UR3T_DONEH	UR3T_DMA 传输完成字节数	7EFA91H	DONE[15:8]								0000,0000
DMA_UR3R_AMTH	UR3R_DMA 传输总字节数	7EFA92H	AMT[15:8]								0000,0000
DMA_UR3R_DONEH	UR3R_DMA 传输完成字节数	7EFA93H	DONE[15:8]								0000,0000
DMA_UR3_ITVH	UR3_DMA 时间间隔寄存器	7EFACCH	ITV[15:8]								0000,0000
DMA_UR3_ITVL	UR3_DMA 时间间隔寄存器	7EFACDH	ITV[7:0]								0000,0000

34.7.2 UR3T_DMA 配置寄存器（DMA_UR3T_CFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3T_CFG	7EFA50H	UR3TIE	-	-	-	UR3TIP[1:0]		UR3TPTY[1:0]	

UR3TIE：UR3T_DMA 中断使能控制位

0：禁止 UR3T_DMA 中断

1：允许 UR3T_DMA 中断

UR3TIP[1:0]：UR3T_DMA 中断优先级控制位

UR3TIP[1:0]	中断优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

UR3TPTY[1:0]：UR3T_DMA 数据总线访问优先级控制位

UR3TPTY [1:0]	总线优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

34.7.3 UR3T_DMA 控制寄存器（DMA_UR3T_CR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3T_CR	7EFA51H	ENUR3T	TRIG	-	-	-	-	-	-

ENUR3T：UR3T_DMA 功能使能控制位

0：禁止 UR3T_DMA 功能

1：允许 UR3T_DMA 功能

TRIG：UR3T_DMA 发送触发控制位

0：写 0 无效

1：写 1 开始 UR3T_DMA 自动发送数据

34.7.4 UR3T_DMA 状态寄存器（DMA_UR3T_STA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3T_STA	7EFA52H	-	-	-	-	-	TXOVW	-	UR3TIF

UR3TIF：UR3T_DMA 中断请求标志位，当 UR3T_DMA 数据发送完成后，硬件自动将 UR3TIF 置 1，若使能 UR3T_DMA 中断则进入中断服务程序。标志位需软件清零

TXOVW：UR3T_DMA 数据覆盖标志位。UR3T_DMA 正在数据传输过程中，串口写 S3BUF 寄存器再次触发串口发送数据时，会导致数据传输失败，此时硬件硬件自动将 TXOVW 置 1。标志位需软件清零

34.7.5 UR3T_DMA 传输总字节寄存器（DMA_UR3T_AMT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3T_AMT	7EFA53H	AMT[7:0]							
DMA_UR3T_AMTH	7EFA90H	AMT[15:8]							

AMT[15:0]：设置需要进行数据读写的字节数。

注：实际读写的字节数为（AMT+1），即当 AMT 设置为 0 时，读写 1 字节，当 AMT 设置 255 时，读写 256 字节。特别注意 AMT 设置为 FFFFH 时表示无限循环模式。

34.7.6 UR3T_DMA 传输完成字节寄存器（DMA_UR3T_DONE）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3T_DONE	7EFA54H	DONE[7:0]							
DMA_UR3T_DONEH	7EFA91H	DONE[15:8]							

DONE[15:0]：当前已经发送完成的字节数。

34.7.7 UR3T_DMA 发送地址寄存器（DMA_UR3T_TXAx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3T_TXAH	7EFA55H	ADDR[15:8]							
DMA_UR3T_TXAL	7EFA56H	ADDR[7:0]							

DMA_UR3T_TXA：设置自动发送数据的源地址。执行 UR3T_DMA 操作时会从这个地址开始读数据。

34.7.8 UR3_DMA 传输周期（DMA_UR3_ITVx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3_ITVH	7EFACCH	ITV[15:8]							
DMA_UR3_ITVL	7EFACDH	ITV[7:0]							

DMA_UR3_ITV[15:0]：设置 UART3 发送和接收的间隔时间（基于系统时钟）。

34.7.9 UR3R_DMA 配置寄存器（DMA_UR3R_CFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3R_CFG	7EFA58H	UR3RIE	-	-	-	UR3RIP[1:0]		UR3RPTY[1:0]	

UR3RIE：UR3R_DMA 中断使能控制位

0：禁止 UR3R_DMA 中断

1：允许 UR3R_DMA 中断

UR3RIP[1:0]：UR3R_DMA 中断优先级控制位

UR3RIP[1:0]	中断优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

UR3RPTY[1:0]：UR3R_DMA 数据总线访问优先级控制位

UR3RPTY [1:0]	总线优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

34.7.10 UR3R_DMA 控制寄存器（DMA_UR3R_CR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3R_CR	7EFA59H	ENUR3R	-	TRIG	-	-	-	-	CLRFIFO

ENUR3R：UR3R_DMA 功能使能控制位

0：禁止 UR3R_DMA 功能

1：允许 UR3R_DMA 功能

TRIG：UR3R_DMA 接收触发控制位

0：写 0 无效

1：写 1 开始 UR3R_DMA 自动接收数据

CLRFIFO：清除 UR3R_DMA 接收 FIFO 控制位

0：写 0 无效

1：开始 UR3R_DMA 操作前，先清空 UR3R_DMA 内置的 FIFO

34.7.11 UR3R_DMA 状态寄存器（DMA_UR3R_STA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3R_STA	7EFA5AH	-	-	-	-	-	-	RXLOSS	UR3RIF

UR3RIF：UR3R_DMA 中断请求标志位，当 UR3R_DMA 接收数据完成后，硬件自动将 UR3RIF 置 1，若使能 UR3R_DMA 中断则进入中断服务程序。标志位需软件清零

RXLOSS：UR3R_DMA 接收数据丢弃标志位。UR3R_DMA 操作过程中，当 XRAM 总线过于繁忙，来不及清空 UR3R_DMA 的接收 FIFO 导致 UR3R_DMA 接收的数据自动丢弃时，硬件硬件自动将 RXLOSS 置 1。标志位需软件清零

34.7.12 UR3R_DMA 传输总字节寄存器（DMA_UR3R_AMT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3R_AMT	7EFA5BH	AMT[7:0]							
DMA_UR3R_AMTH	7EFA92H	AMT[15:8]							

AMT[15:0]：设置需要进行数据读写的字节数。

注：实际读写的字节数为（AMT+1），即当 AMT 设置为 0 时，读写 1 字节，当 AMT 设置 255 时，读写 256 字节。特别注意 AMT 设置为 FFFFH 时表示无限循环模式。

34.7.13 UR3R_DMA 传输完成字节寄存器（DMA_UR3R_DONE）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3R_DONE	7EFA5CH	DONE[7:0]							
DMA_UR3R_DONEH	7EFA93H	DONE[15:8]							

DONE[15:0]：当前已经接收完成的字节数。

34.7.14 UR3R_DMA 接收地址寄存器（DMA_UR3R_RXAx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3R_RXAH	7EFA5DH	ADDR[15:8]							
DMA_UR3R_RXAL	7EFA5EH	ADDR[7:0]							

DMA_UR3R_RXA：设置自动接收数据的目标地址。执行 UR3R_DMA 操作时会从这个地址开始写数据。

34.8 串口 4 与存储器之间的数据交换（UR4T_DMA，UR4R_DMA）

34.8.1 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
DMA_UR4T_CFG	UR4T_DMA 配置寄存器	7EFA60H	UR4TIE	-	-	-	UR4TIP[1:0]		UR4TPTY[1:0]		0xxx,0000
DMA_UR4T_CR	UR4T_DMA 控制寄存器	7EFA61H	ENUR4T	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_UR4T_STA	UR4T_DMA 状态寄存器	7EFA62H	-	-	-	-	-	TXOVW	-	UR4TIF	xxxx,x0x0
DMA_UR4T_AMT	UR4T_DMA 传输总字节数	7EFA63H	AMT[7:0]								0000,0000
DMA_UR4T_DONE	UR4T_DMA 传输完成字节数	7EFA64H	DONE[7:0]								0000,0000
DMA_UR4T_TXAH	UR4T_DMA 发送高地址	7EFA65H	ADR[15:0]								0000,0000
DMA_UR4T_TXAL	UR4T_DMA 发送低地址	7EFA66H	ADR[7:0]								0000,0000
DMA_UR4R_CFG	UR4R_DMA 配置寄存器	7EFA68H	UR4RIE	-	-	-	UR4RIP[1:0]		UR4RPTY[1:0]		0xxx,0000
DMA_UR4R_CR	UR4R_DMA 控制寄存器	7EFA69H	ENUR4R	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0
DMA_UR4R_STA	UR4R_DMA 状态寄存器	7EFA6AH	-	-	-	-	-	-	RXLOSS	UR4RIF	xxxx,xx00
DMA_UR4R_AMT	UR4R_DMA 传输总字节数	7EFA6BH	AMT[7:0]								0000,0000
DMA_UR4R_DONE	UR4R_DMA 传输完成字节数	7EFA6CH	DONE[7:0]								0000,0000
DMA_UR4R_RXAH	UR4R_DMA 接收高地址	7EFA6DH	ADR[15:0]								0000,0000
DMA_UR4R_RXAL	UR4R_DMA 接收低地址	7EFA6EH	ADR[7:0]								0000,0000
DMA_UR4T_AMTH	UR4T_DMA 传输总字节数	7EFA94H	AMT[15:8]								0000,0000
DMA_UR4T_DONEH	UR4T_DMA 传输完成字节数	7EFA95H	DONE[15:8]								0000,0000
DMA_UR4R_AMTH	UR4R_DMA 传输总字节数	7EFA96H	AMT[15:8]								0000,0000
DMA_UR4R_DONEH	UR4R_DMA 传输完成字节数	7EFA97H	DONE[15:8]								0000,0000
DMA_UR4_ITVH	UR4_DMA 时间间隔寄存器	7EFACEH	ITV[15:8]								0000,0000
DMA_UR4_ITVL	UR4_DMA 时间间隔寄存器	7EFACFH	ITV[7:0]								0000,0000

34.8.2 UR4T_DMA 配置寄存器（DMA_UR4T_CFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4T_CFG	7EFA60H	UR4TIE	-	-	-	UR4TIP[1:0]		UR4TPTY[1:0]	

UR4TIE：UR4T_DMA 中断使能控制位

0：禁止 UR4T_DMA 中断

1：允许 UR4T_DMA 中断

UR4TIP[1:0]：UR4T_DMA 中断优先级控制位

UR4TIP[1:0]	中断优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

UR4TPTY[1:0]：UR4T_DMA 数据总线访问优先级控制位

UR4TPTY [1:0]	总线优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

34.8.3 UR4T_DMA 控制寄存器（DMA_UR4T_CR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4T_CR	7EFA61H	ENUR4T	TRIG	-	-	-	-	-	-

ENUR4T：UR4T_DMA 功能使能控制位

0：禁止 UR4T_DMA 功能

1：允许 UR4T_DMA 功能

TRIG：UR4T_DMA 发送触发控制位

0：写 0 无效

1：写 1 开始 UR4T_DMA 自动发送数据

34.8.4 UR4T_DMA 状态寄存器（DMA_UR4T_STA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4T_STA	7EFA62H	-	-	-	-	-	TXOVW	-	UR4TIF

UR4TIF：UR4T_DMA 中断请求标志位，当 UR4T_DMA 数据发送完成后，硬件自动将 UR4TIF 置 1，若使能 UR4T_DMA 中断则进入中断服务程序。标志位需软件清零

TXOVW：UR4T_DMA 数据覆盖标志位。UR4T_DMA 正在数据传输过程中，串口写 S4BUF 寄存器再次触发串口发送数据时，会导致数据传输失败，此时硬件硬件自动将 TXOVW 置 1。标志位需软件清零

34.8.5 UR4T_DMA 传输总字节寄存器（DMA_UR4T_AMT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4T_AMT	7EFA63H	AMT[7:0]							
DMA_UR4T_AMTH	7EFA94H	AMT[15:8]							

AMT[15:0]：设置需要进行数据读写的字节数。

注：实际读写的字节数为（AMT+1），即当 AMT 设置为 0 时，读写 1 字节，当 AMT 设置 255 时，读写 256 字节。特别注意 AMT 设置为 FFFFH 时表示无限循环模式。

34.8.6 UR4T_DMA 传输完成字节寄存器（DMA_UR4T_DONE）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4T_DONE	7EFA64H	DONE[7:0]							
DMA_UR4T_DONEH	7EFA95H	DONE[15:8]							

DONE[15:0]：当前已经发送完成的字节数。

34.8.7 UR4T_DMA 发送地址寄存器（DMA_UR4T_TXAx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4T_TXAH	7EFA65H	ADDR[15:8]							
DMA_UR4T_TXAL	7EFA66H	ADDR[7:0]							

DMA_UR4T_TXA：设置自动发送数据的源地址。执行 UR4T_DMA 操作时会从这个地址开始读数据。

34.8.8 UR4_DMA 传输周期（DMA_UR4_ITVx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4_ITVH	7EFACEH	ITV[15:8]							
DMA_UR4_ITVL	7EFACFH	ITV[7:0]							

DMA_UR4_ITV[15:0]：设置 UART4 发送和接收的间隔时间（基于系统时钟）。

34.8.9 UR4R_DMA 配置寄存器（DMA_UR4R_CFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4R_CFG	7EFA68H	UR4RIE	-	-	-	UR4RIP[1:0]		UR4RPTY[1:0]	

UR4RIE：UR4R_DMA 中断使能控制位

0：禁止 UR4R_DMA 中断

1：允许 UR4R_DMA 中断

UR4RIP[1:0]：UR4R_DMA 中断优先级控制位

UR4RIP[1:0]	中断优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

UR4RPTY[1:0]：UR4R_DMA 数据总线访问优先级控制位

UR4RPTY [1:0]	总线优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

34.8.10 UR4R_DMA 控制寄存器（DMA_UR4R_CR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4R_CR	7EFA69H	ENUR4R	-	TRIG	-	-	-	-	CLRFIFO

ENUR4R：UR4R_DMA 功能使能控制位

0：禁止 UR4R_DMA 功能

1：允许 UR4R_DMA 功能

TRIG：UR4R_DMA 接收触发控制位

0：写 0 无效

1：写 1 开始 UR4R_DMA 自动接收数据

CLRFIFO：清除 UR4R_DMA 接收 FIFO 控制位

0：写 0 无效

1：开始 UR4R_DMA 操作前，先清空 UR4R_DMA 内置的 FIFO

34.8.11 UR4R_DMA 状态寄存器（DMA_UR4R_STA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4R_STA	7EFA6AH	-	-	-	-	-	-	RXLOSS	UR4RIF

UR4RIF：UR4R_DMA 中断请求标志位，当 UR4R_DMA 接收数据完成后，硬件自动将 UR4RIF 置 1，若使能 UR4R_DMA 中断则进入中断服务程序。标志位需软件清零

RXLOSS：UR4R_DMA 接收数据丢弃标志位。UR4R_DMA 操作过程中，当 XRAM 总线过于繁忙，来不及清空 UR4R_DMA 的接收 FIFO 导致 UR4R_DMA 接收的数据自动丢弃时，硬件硬件自动将 RXLOSS 置 1。标志位需软件清零

34.8.12 UR4R_DMA 传输总字节寄存器（DMA_UR4R_AMT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4R_AMT	7EFA6BH	AMT[7:0]							
DMA_UR4R_AMTH	7EFA96H	AMT[15:8]							

AMT[15:0]：设置需要进行数据读写的字节数。

注：实际读写的字节数为（AMT+1），即当 AMT 设置为 0 时，读写 1 字节，当 AMT 设置 255 时，读写 256 字节。特别注意 AMT 设置为 FFFFH 时表示无限循环模式。

34.8.13 UR4R_DMA 传输完成字节寄存器（DMA_UR4R_DONE）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4R_DONE	7EFA6CH	DONE[7:0]							
DMA_UR4R_DONEH	7EFA97H	DONE[15:8]							

DONE[15:0]：当前已经接收完成的字节数。

34.8.14 UR4R_DMA 接收地址寄存器（DMA_UR4R_RXAx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4R_RXAH	7EFA6DH	ADDR[15:8]							
DMA_UR4R_RXAL	7EFA6EH	ADDR[7:0]							

DMA_UR4R_RXA：设置自动接收数据的目标地址。执行 UR4R_DMA 操作时会从这个地址开始写数据。

34.9 TFT 彩屏与存储器之间的数据读写（LCM_DMA）

34.9.1 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
DMA_LCM_CFG	TFT 彩屏 DMA 配置寄存器	7EFA70H	LCMIE	-	-	-	LCMIP[1:0]		LCMPTY[1:0]		0xxx,0000
DMA_LCM_CR	TFT 彩屏 DMA 控制寄存器	7EFA71H	ENLCM	TRIGWC	TRIGWD	TRIGRC	TRIGRD	-	-	-	0000,0xxx
DMA_LCM_STA	TFT 彩屏 DMA 状态寄存器	7EFA72H	-	-	-	-	-	-	TXOVW	LCMIF	xxxx,xx00
DMA_LCM_AMT	TFT 彩屏 DMA 传输总字节数	7EFA73H	AMT[7:0]								0000,0000
DMA_LCM_DONE	TFT 彩屏 DMA 传输完成字节数	7EFA74H	DONE[7:0]								0000,0000
DMA_LCM_TXAH	TFT 彩屏 DMA 发送高地址	7EFA75H	ADR[15:0]								0000,0000
DMA_LCM_TXAL	TFT 彩屏 DMA 发送低地址	7EFA76H	ADR[7:0]								0000,0000
DMA_LCM_RXAH	TFT 彩屏 DMA 接收高地址	7EFA77H	ADR[15:0]								0000,0000
DMA_LCM_RXAL	TFT 彩屏 DMA 接收低地址	7EFA78H	ADR[7:0]								0000,0000
DMA_LCM_ITVH	TFT 彩屏 DMA 时间间隔寄存器	7EFA7EH	ITV[15:8]								0000,0000
DMA_LCM_ITVL	TFT 彩屏 DMA 时间间隔寄存器	7EFA7FH	ITV[7:0]								0000,0000
DMA_LCM_AMTH	TFT 彩屏 DMA 传输总字节数	7EFA86H	AMT[15:8]								0000,0000
DMA_LCM_DONEH	TFT 彩屏 DMA 传输完成字节数	7EFA87H	DONE[15:8]								0000,0000

34.9.2 TFT 彩屏 DMA 配置寄存器（DMA_LCM_CFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_CFG	7EFA70H	LCMIE	-	-	-	LCMIP[1:0]		LCMPTY[1:0]	

LCMIE：TFT 彩屏 DMA 中断使能控制位

0：禁止 TFT 彩屏 DMA 中断

1：允许 TFT 彩屏 DMA 中断

LCMIP[1:0]：TFT 彩屏 DMA 中断优先级控制位

LCMIP[1:0]	中断优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

LCMPTY[1:0]：TFT 彩屏 DMA 数据总线访问优先级控制位

LCMPTY [1:0]	总线优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

34.9.3 TFT 彩屏 DMA 控制寄存器（DMA_LCM_CR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_CR	7EFA71H	ENLCM	TRIGWC	TRIGWD	TRIGRC	TRIGRD	-	-	CLRFIFO

ENLCM：TFT 彩屏 DMA 功能使能控制位

0：禁止 TFT 彩屏 DMA 功能

1：允许 TFT 彩屏 DMA 功能

TRIGWC：TFT 彩屏 DMA 发送命令模式触发控制位

0：写 0 无效

1：写 1 开始 TFT 彩屏 DMA 发送命令模式操作

TRIGWD：TFT 彩屏 DMA 发送数据模式触发控制位

0：写 0 无效

1：写 1 开始 TFT 彩屏 DMA 发送数据模式操作

TRIGRC：TFT 彩屏 DMA 读取命令模式触发控制位

0：写 0 无效

1：写 1 开始 TFT 彩屏 DMA 读取命令模式操作

TRIGRD：TFT 彩屏 DMA 读取数据模式触发控制位

0：写 0 无效

1：写 1 开始 TFT 彩屏 DMA 读取数据模式操作

CLRFIFO：清除 TFT 彩屏 DMA 接收 FIFO 控制位

0：写 0 无效

1：开始 TFT 彩屏 DMA 操作前，先清空内置的 FIFO

34.9.4 TFT 彩屏 DMA 状态寄存器（DMA_LCM_STA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_STA	7EFA72H	-	-	-	-	-	-	TXOVW	LCMIF

LCMIF：TFT 彩屏 DMA 中断请求标志位，当 TFT 彩屏 DMA 数据交换完成后，硬件自动将 LCMIF 置 1，若使能 TFT 彩屏 DMA 中断则进入中断服务程序。标志位需软件清零

TXOVW：TFT 彩屏 DMA 数据覆盖标志位。TFT 彩屏 DMA 正在数据传输过程中，LCMIF 写 LCMIFDATL 和 LCMIFDATH 寄存器时，会导致数据传输失败，此时硬件硬件自动将 TXOVW 置 1。标志位需软件清零

34.9.5 TFT 彩屏 DMA 传输总字节寄存器（DMA_LCM_AMT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_AMT	7EFA73H	AMT[7:0]							
DMA_LCM_AMTH	7EFA86H	AMT[15:8]							

AMT[15:0]：设置需要进行数据读写的字节数。

注：实际读写的字节数为（AMT+1），即当 AMT 设置为 0 时，读写 1 字节，当 AMT 设置 255 时，读写 256 字节。特别注意 AMT 设置为 FFFFH 时表示无限循环模式。

34.9.6 TFT 彩屏 DMA 传输完成字节寄存器（DMA_LCM_DONE）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_DONE	7EFA74H	DONE[7:0]							
DMA_LCM_DONEH	7EFA87H	DONE[15:8]							

DONE[15:0]：当前已经传输完成的字节数。

34.9.7 TFT 彩屏 DMA 发送地址寄存器（DMA_LCM_TXAx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_TXAH	7EFA75H	ADDR[15:8]							
DMA_LCM_TXAL	7EFA76H	ADDR[7:0]							

DMA_LCM_TXA：设置进行数据传输时的源地址。执行 TFT 彩屏 DMA 操作时会从这个地址开始读数据。

34.9.8 TFT 彩屏 DMA 接收地址寄存器（DMA_LCM_RXAx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_RXAH	7EFA77H	ADDR[15:8]							
DMA_LCM_RXAL	7EFA78H	ADDR[7:0]							

DMA_LCM_RXA：设置进行数据传输时的目标地址。执行 TFT 彩屏 DMA 操作时会从这个地址开始写入数据。

34.9.9 TFT 彩屏 DMA 传输周期（DMA_LCM_ITVx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_ITVH	7EFA7EH	ITV[15:8]							
DMA_LCM_ITVL	7EFA7FH	ITV[7:0]							

DMA_LCM_ITV[15:0]：设置数据传输时周期（基于系统时钟经过 LCMIFPSCR 分频器分频后的 LCMIF 时钟）。

34.10 I2C 与存储器之间的数据交换（I2CT_DMA，I2CR_DMA）

34.10.1 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
DMA_I2CT_CFG	I2CT_DMA 配置寄存器	7EFA98H	I2CTIE	-	-	-	I2CTIP[1:0]		I2CTPTY[1:0]		0xxx,0000
DMA_I2CT_CR	I2CT_DMA 控制寄存器	7EFA99H	ENI2CT	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_I2CT_STA	I2CT_DMA 状态寄存器	7EFA9AH	-	-	-	-	-	TXOVW	-	I2CTIF	xxxx,x0x0
DMA_I2CT_AMT	I2CT_DMA 传输总字节数	7EFA9BH	AMT[7:0]								0000,0000
DMA_I2CT_DONE	I2CT_DMA 传输完成字节数	7EFA9CH	DONE[7:0]								0000,0000
DMA_I2CT_TXAH	I2CT_DMA 发送高地址	7EFA9DH	ADR[15:0]								0000,0000
DMA_I2CT_TXAL	I2CT_DMA 发送低地址	7EFA9EH	ADR[7:0]								0000,0000
DMA_I2CR_CFG	I2CR_DMA 配置寄存器	7EFAA0H	I2CRIE	-	-	-	I2CRIP[1:0]		I2CRPTY[1:0]		0xxx,0000
DMA_I2CR_CR	I2CR_DMA 控制寄存器	7EFAA1H	ENI2CR	TRIG	-	-	-	-	-	CLRFIFO	00xx,xxx0
DMA_I2CR_STA	I2CR_DMA 状态寄存器	7EFAA2H	-	-	-	-	-	-	RXLOSS	I2CRIF	xxxx,xx00
DMA_I2CR_AMT	I2CR_DMA 传输总字节数	7EFAA3H	AMT[7:0]								0000,0000
DMA_I2CR_DONE	I2CR_DMA 传输完成字节数	7EFAA4H	DONE[7:0]								0000,0000
DMA_I2CR_RXAH	I2CR_DMA 接收高地址	7EFAA5H	ADR[15:0]								0000,0000
DMA_I2CR_RXAL	I2CR_DMA 接收低地址	7EFAA6H	ADR[7:0]								0000,0000
DMA_I2CT_AMTH	I2CT_DMA 传输总字节数	7EFAA8H	AMT[15:8]								0000,0000
DMA_I2CT_DONEH	I2CT_DMA 传输完成字节数	7EFAA9H	DONE[15:8]								0000,0000
DMA_I2CR_AMTH	I2CR_DMA 传输总字节数	7EFAAAH	AMT[15:8]								0000,0000
DMA_I2CR_DONEH	I2CR_DMA 传输完成字节数	7EFAABH	DONE[15:8]								0000,0000
DMA_I2C_CR	I2C_DMA 控制寄存器	7EFAADH	RDSEL	-	-	-	-	ACKERR	INTEN	BMMEN	0xxx,x000
DMA_I2C_ST1	I2C_DMA 状态寄存器	7EFAAEH	COUNT[7:0]								0000,0000
DMA_I2C_ST2	I2C_DMA 状态寄存器	7EFAAFH	COUNT[15:8]								0000,0000
DMA_I2C_ITVH	I2C_DMA 时间间隔寄存器	7EFAC4H	ITV[15:8]								0000,0000
DMA_I2C_ITVL	I2C_DMA 时间间隔寄存器	7EFAC5H	ITV[7:0]								0000,0000

34.10.2 I2CT_DMA 配置寄存器（DMA_I2CT_CFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2CT_CFG	7EFA98H	I2CTIE	-	-	-	I2CTIP[1:0]		I2CTPTY[1:0]	

I2CTIE：I2CT_DMA 中断使能控制位

0：禁止 I2CT_DMA 中断

1：允许 I2CT_DMA 中断

I2CTIP[1:0]：I2CT_DMA 中断优先级控制位

I2CTIP[1:0]	中断优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

I2CTPTY[1:0]：I2CT_DMA 数据总线访问优先级控制位

I2CTPTY [1:0]	总线优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

34.10.3 I2CT_DMA 控制寄存器（DMA_I2CT_CR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2CT_CR	7EFA99H	ENI2CT	TRIG	-	-	-	-	-	-

ENI2CT：I2CT_DMA 功能使能控制位

0：禁止 I2CT_DMA 功能

1：允许 I2CT_DMA 功能

TRIG：I2CT_DMA 发送触发控制位

0：写 0 无效

1：写 1 开始 I2CT_DMA 自动发送数据

34.10.4 I2CT_DMA 状态寄存器（DMA_I2CT_STA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2CT_STA	7EFA9AH	-	-	-	-	-	TXOVW	-	I2CTIF

I2CTIF：I2CT_DMA 中断请求标志位，当 I2CT_DMA 数据发送完成后，硬件自动将 I2CTIF 置 1，若使能 I2CT_DMA 中断则进入中断服务程序。标志位需软件清零

TXOVW：I2CT_DMA 数据覆盖标志位。I2CT_DMA 正在数据传输过程中，写 I2C 数据寄存器 I2CTXD 时，会导致数据传输失败，此时硬件硬件自动将 TXOVW 置 1。标志位需软件清零

34.10.5 I2CT_DMA 传输总字节寄存器（DMA_I2CT_AMT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2CT_AMT	7EFA9BH	AMT[7:0]							
DMA_I2CT_AMTH	7EFAA8H	AMT[15:8]							

AMT[15:0]：设置需要进行数据读写的字节数。

注：实际读写的字节数为（AMT+1），即当 AMT 设置为 0 时，读写 1 字节，当 AMT 设置 255 时，读写 256 字节。特别注意 AMT 设置为 FFFFH 时表示无限循环模式。

34.10.6 I2CT_DMA 传输完成字节寄存器（DMA_I2CT_DONE）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2CT_DONE	7EFA9CH	DONE[7:0]							
DMA_I2CT_DONEH	7EFAA9H	DONE[15:8]							

DONE[15:0]：当前已经发送完成的字节数。

34.10.7 I2CT_DMA 发送地址寄存器（DMA_I2CT_TXAx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2CT_TXAH	7EFA9DH	ADDR[15:8]							
DMA_I2CT_TXAL	7EFA9EH	ADDR[7:0]							

DMA_I2CT_TXA：设置自动发送数据的源地址。执行 I2CT_DMA 操作时会从这个地址开始读数据。

34.10.8 I2C_DMA 传输周期（DMA_I2C_ITVx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2C_ITVH	7EFAC4H	ITV[15:8]							
DMA_I2C_ITVL	7EFAC5H	ITV[7:0]							

DMA_I2C_ITV[15:0]：设置 I2C 发送和接收的间隔时间（基于系统时钟）。

34.10.9 I2CR_DMA 配置寄存器（DMA_I2CR_CFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2CR_CFG	7EFAA0H	I2CRIE	-	-	-	I2CRIP[1:0]		I2CRPTY[1:0]	

I2CRIE：I2CR_DMA 中断使能控制位

0：禁止 I2CR_DMA 中断

1：允许 I2CR_DMA 中断

I2CRIP[1:0]：I2CR_DMA 中断优先级控制位

I2CRIP[1:0]	中断优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

I2CRPTY[1:0]：I2CR_DMA 数据总线访问优先级控制位

I2CRPTY [1:0]	总线优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

34.10.10 I2CR_DMA 控制寄存器（DMA_I2CR_CR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2CR_CR	7EFAA1H	ENI2CR	TRIG	-	-	-	-	-	CLRFIFO

ENI2CR：I2CR_DMA 功能使能控制位

0：禁止 I2CR_DMA 功能

1：允许 I2CR_DMA 功能

TRIG：I2CR_DMA 接收触发控制位

0：写 0 无效

1：写 1 开始 I2CR_DMA 自动接收数据

CLRFIFO：清除 I2CR_DMA 接收 FIFO 控制位

0：写 0 无效

1：开始 I2CR_DMA 操作前，先清空 I2CR_DMA 内置的 FIFO

34.10.11 I2CR_DMA 状态寄存器（DMA_I2CR_STA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2CR_STA	7EFAA2H	-	-	-	-	-	-	RXLOSS	I2CRIF

I2CRIF：I2CR_DMA 中断请求标志位，当 I2CR_DMA 接收数据完成后，硬件自动将 I2CRIF 置 1，若使能 I2CR_DMA 中断则进入中断服务程序。标志位需软件清零

RXLOSS：I2CR_DMA 接收数据丢弃标志位。I2CR_DMA 操作过程中，当 XRAM 总线过于繁忙，来不及清空 I2CR_DMA 的接收 FIFO 导致 I2CR_DMA 接收的数据自动丢弃时，硬件硬件自动将 RXLOSS 置 1。标志位需软件清零

34.10.12 I2CR_DMA 传输总字节寄存器（DMA_I2CR_AMT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2CR_AMT	7EFAA3H	AMT[7:0]							
DMA_I2CR_AMTH	7EFAAAH	AMT[15:8]							

AMT[15:0]：设置需要进行数据读写的字节数。

注：实际读写的字节数为（AMT+1），即当 AMT 设置为 0 时，读写 1 字节，当 AMT 设置 255 时，读写 256 字节。特别注意 AMT 设置为 FFFFH 时表示无限循环模式。

34.10.13 I2CR_DMA 传输完成字节寄存器（DMA_I2CR_DONE）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2CR_DONE	7EFAA4H	DONE[7:0]							
DMA_I2CR_DONEH	7EFAABH	DONE[15:8]							

DONE[15:0]：当前已经接收完成的字节数。

34.10.14 I2CR_DMA 接收地址寄存器（DMA_I2CR_RXAx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2CR_RXAH	7EFAA5H	ADDR[15:8]							
DMA_I2CR_RXAL	7EFAA6H	ADDR[7:0]							

DMA_I2CR_RXA：设置自动接收数据的目标地址。执行 I2CR_DMA 操作时会从这个地址开始写数据。

34.10.15 I2C_DMA 控制寄存器（DMA_I2C_CR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2C_CR	7EFAADH	RDSEL	-	-	-	-	ACKERR	ERRIE	BMMEN

RDSEL：I2C_DMA_ST 寄存器读取功能选择

ACKERR：ACK 错误

0：发送数据后收到的应答是 ACK

1：发送数据后收到的应答是 NAK（需软件清零）

ERRIE：ACKERR 中断使能控制位

0：禁止 ACKERR 中断

1：允许 ACKERR 中断（ACKERR 中断入口地址为 I2C 中断入口）

BMMEN：I2C 的 DMA 功能使能位

0：禁止 I2C_DMA 功能

1：允许 I2C_DMA 功能

34.10.16 I2C_DMA 状态寄存器（DMA_I2C_ST）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2C_ST1	7EFAAEH	COUNT[7:0]							
DMA_I2C_ST2	7EFAAFH	COUNT[15:8]							

COUNT：I2C_DMA 传输字节控制

写寄存器：设置 I2C_DMA 传输字节数

读寄存器：RDSEL＝0 时，COUNT 为需要传输的字节数

RDSEL＝1 时，COUNT 为已经传输完成的字节数

34.11 I2S 与存储器之间的数据交换（I2ST_DMA，I2SR_DMA）

34.11.1 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
DMA_I2ST_CFG	I2ST_DMA 配置寄存器	7EFAB0H	I2STIE	-	-	-	I2STIP[1:0]		I2STPTY[1:0]		0xxx,0000
DMA_I2ST_CR	I2ST_DMA 控制寄存器	7EFAB1H	ENI2ST	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_I2ST_STA	I2ST_DMA 状态寄存器	7EFAB2H	-	-	-	-	-	TXOVW	-	I2STIF	xxxx,x0x0
DMA_I2ST_AMT	I2ST_DMA 传输总字节数	7EFAB3H	AMT[7:0]								0000,0000
DMA_I2ST_DONE	I2ST_DMA 传输完成字节数	7EFAB4H	DONE[7:0]								0000,0000
DMA_I2ST_TXAH	I2ST_DMA 发送高地址	7EFAB5H	ADR[15:0]								0000,0000
DMA_I2ST_TXAL	I2ST_DMA 发送低地址	7EFAB6H	ADR[7:0]								0000,0000
DMA_I2SR_CFG	I2SR_DMA 配置寄存器	7EFAB8H	I2SRIE	-	-	-	I2SRIP[1:0]		I2SRPTY[1:0]		0xxx,0000
DMA_I2SR_CR	I2SR_DMA 控制寄存器	7EFAB9H	ENI2SR	TRIG	-	-	-	-	-	CLRFIFO	00xx,xxx0
DMA_I2SR_STA	I2SR_DMA 状态寄存器	7EFABAH	-	-	-	-	-	-	RXLOSS	I2SRIF	xxxx,xx00
DMA_I2SR_AMT	I2SR_DMA 传输总字节数	7EFABBH	AMT[7:0]								0000,0000
DMA_I2SR_DONE	I2SR_DMA 传输完成字节数	7EFABCH	DONE[7:0]								0000,0000
DMA_I2SR_RXAH	I2SR_DMA 接收高地址	7EFABDH	ADR[15:0]								0000,0000
DMA_I2SR_RXAL	I2SR_DMA 接收低地址	7EFABEH	ADR[7:0]								0000,0000
DMA_I2ST_AMTH	I2ST_DMA 传输总字节数	7EFAC0H	AMT[15:8]								0000,0000
DMA_I2ST_DONEH	I2ST_DMA 传输完成字节数	7EFAC1H	DONE[15:8]								0000,0000
DMA_I2SR_AMTH	I2SR_DMA 传输总字节数	7EFAC2H	AMT[15:8]								0000,0000
DMA_I2SR_DONEH	I2SR_DMA 传输完成字节数	7EFAC3H	DONE[15:8]								0000,0000
DMA_I2S_ITVH	I2S_DMA 时间间隔寄存器	7EFAC6H	ITV[15:8]								0000,0000
DMA_I2S_ITVL	I2S_DMA 时间间隔寄存器	7EFAC7H	ITV[7:0]								0000,0000

34.11.2 I2ST_DMA 配置寄存器（DMA_I2ST_CFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2ST_CFG	7EFAB0H	I2STIE	-	-	-	I2STIP[1:0]		I2STPTY[1:0]	

I2STIE：I2ST_DMA 中断使能控制位

0：禁止 I2ST_DMA 中断

1：允许 I2ST_DMA 中断

I2STIP[1:0]：I2ST_DMA 中断优先级控制位

I2STIP[1:0]	中断优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

I2STPTY[1:0]：I2ST_DMA 数据总线访问优先级控制位

I2STPTY [1:0]	总线优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

34.11.3 I2ST_DMA 控制寄存器（DMA_I2ST_CR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2ST_CR	7EFAB1H	ENI2ST	TRIG	-	-	-	-	-	-

ENI2ST：I2ST_DMA 功能使能控制位

0：禁止 I2ST_DMA 功能

1：允许 I2ST_DMA 功能

TRIG：I2ST_DMA 发送触发控制位

0：写 0 无效

1：写 1 开始 I2ST_DMA 自动发送数据

34.11.4 I2ST_DMA 状态寄存器（DMA_I2ST_STA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2ST_STA	7EFAB2H	-	-	-	-	-	TXOVW	-	I2STIF

I2STIF：I2ST_DMA 中断请求标志位，当 I2ST_DMA 数据发送完成后，硬件自动将 I2STIF 置 1，若使能 I2ST_DMA 中断则进入中断服务程序。标志位需软件清零

TXOVW：I2ST_DMA 数据覆盖标志位。I2ST_DMA 正在数据传输过程中，写 I2S 数据寄存器 I2S_DRH 和 I2S_DRL 时，会导致数据传输失败，此时硬件硬件自动将 TXOVW 置 1。标志位需软件清零

34.11.5 I2ST_DMA 传输总字节寄存器（DMA_I2ST_AMT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2ST_AMT	7EFAB3H	AMT[7:0]							
DMA_I2ST_AMTH	7EFAC0H	AMT[15:8]							

AMT[15:0]：设置需要进行数据读写的字节数。

注：实际读写的字节数为（AMT+1），即当 AMT 设置为 0 时，读写 1 字节，当 AMT 设置 255 时，读写 256 字节。特别注意 AMT 设置为 FFFFH 时表示无限循环模式。

34.11.6 I2ST_DMA 传输完成字节寄存器（DMA_I2ST_DONE）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2ST_DONE	7EFAB4H	DONE[7:0]							
DMA_I2ST_DONEH	7EFAC1H	DONE[15:8]							

DONE[15:0]：当前已经发送完成的字节数。

34.11.7 I2ST_DMA 发送地址寄存器（DMA_I2ST_TXAx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2ST_TXAH	7EFAB5H	ADDR[15:8]							
DMA_I2ST_TXAL	7EFAB6H	ADDR[7:0]							

DMA_I2ST_TXA：设置自动发送数据的源地址。执行 I2ST_DMA 操作时会从这个地址开始读数据。

34.11.8 I2SR_DMA 配置寄存器（DMA_I2SR_CFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2SR_CFG	7EFAB8H	I2SRIE	-	-	-	I2SRIP[1:0]		I2SRPTY[1:0]	

I2SRIE：I2SR_DMA 中断使能控制位

0：禁止 I2SR_DMA 中断

1：允许 I2SR_DMA 中断

I2SRIP[1:0]：I2SR_DMA 中断优先级控制位

I2SRIP[1:0]	中断优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

I2SRPTY[1:0]：I2SR_DMA 数据总线访问优先级控制位

I2SRPTY [1:0]	总线优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

34.11.9 I2S_DMA 传输周期（DMA_I2S_ITVx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2S_ITVH	7EFAC6H	ITV[15:8]							
DMA_I2S_ITVL	7EFAC7H	ITV[7:0]							

DMA_I2S_ITV[15:0]：设置 I2S 发送和接收的间隔时间（基于系统时钟）。

34.11.10 I2SR_DMA 控制寄存器（DMA_I2SR_CR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2SR_CR	7EFAB9H	ENI2SR	TRIG	-	-	-	-	-	CLRFIFO

ENI2SR：I2SR_DMA 功能使能控制位

0：禁止 I2SR_DMA 功能

1：允许 I2SR_DMA 功能

TRIG：I2SR_DMA 接收触发控制位

0：写 0 无效

1：写 1 开始 I2SR_DMA 自动接收数据

CLRFIFO：清除 I2SR_DMA 接收 FIFO 控制位

0：写 0 无效

1：开始 I2SR_DMA 操作前，先清空 I2SR_DMA 内置的 FIFO

34.11.11 I2SR_DMA 状态寄存器（DMA_I2SR_STA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2SR_STA	7EFABAH	-	-	-	-	-	-	RXLOSS	I2SRIF

I2SRIF：I2SR_DMA 中断请求标志位，当 I2SR_DMA 接收数据完成后，硬件自动将 I2SRIF 置 1，若使能 I2SR_DMA 中断则进入中断服务程序。标志位需软件清零

RXLOSS：I2SR_DMA 接收数据丢弃标志位。I2SR_DMA 操作过程中，当 XRAM 总线过于繁忙，来不及清空 I2SR_DMA 的接收 FIFO 导致 I2SR_DMA 接收的数据自动丢弃时，硬件硬件自动将 RXLOSS 置 1。标志位需软件清零

34.11.12 I2SR_DMA 传输总字节寄存器（DMA_I2SR_AMT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2SR_AMT	7EFABBH	AMT[7:0]							
DMA_I2SR_AMTH	7EFAC2H	AMT[15:8]							

AMT[15:0]：设置需要进行数据读写的字节数。

注：实际读写的字节数为（AMT+1），即当 AMT 设置为 0 时，读写 1 字节，当 AMT 设置 255 时，读写 256 字节。特别注意 AMT 设置为 FFFFH 时表示无限循环模式。

34.11.13 I2SR_DMA 传输完成字节寄存器（DMA_I2SR_DONE）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2SR_DONE	7EFABCH	DONE[7:0]							
DMA_I2SR_DONEH	7EFAC3H	DONE[15:8]							

DONE[15:0]：当前已经接收完成的字节数。

34.11.14 I2SR_DMA 接收地址寄存器（DMA_I2SR_RXAx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2SR_RXAH	7EFABDH	ADDR[15:8]							
DMA_I2SR_RXAL	7EFABEH	ADDR[7:0]							

DMA_I2SR_RXA：设置自动接收数据的目标地址。执行 I2SR_DMA 操作时会从这个地址开始写数据。

34.12 QSPI 与存储器之间的数据交换（QSPI_DMA）

34.12.1 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
DMA_QSPI_CFG	QSPI_DMA 配置寄存器	7EFAD0H	QSPIIE	ACT_WR	ACT_RD	-	QSPIIP[1:0]		QSPIPTY[1:0]		000x,0000
DMA_QSPI_CR	QSPI_DMA 控制寄存器	7EFAD1H	ENQSPI	TRIG_W	TRIG_R	-	-	-	CLRWFIFO	CLRRFIFO	000x,xx00
DMA_QSPI_STA	QSPI_DMA 状态寄存器	7EFAD2H	-	-	-	-	-	-	-	QSPIIF	xxxx,xxx0
DMA_QSPI_AMT	QSPI_DMA 传输总字节数	7EFAD3H	AMT[7:0]								0000,0000
DMA_QSPI_DONE	QSPI_DMA 传输完成字节数	7EFAD4H	DONE[7:0]								0000,0000
DMA_QSPI_TXAH	QSPI_DMA 发送高地址	7EFAD5H	ADR[15:0]								0000,0000
DMA_QSPI_TXAL	QSPI_DMA 发送低地址	7EFAD6H	ADR[7:0]								0000,0000
DMA_QSPI_RXAH	QSPI_DMA 接收高地址	7EFAD7H	ADR[15:0]								0000,0000
DMA_QSPI_RXAL	QSPI_DMA 接收低地址	7EFAD8H	ADR[7:0]								0000,0000
DMA_QSPI_AMTH	QSPI_DMA 传输总字节数	7EFADBH	AMT[15:8]								0000,0000
DMA_QSPI_DONEH	QSPI_DMA 传输完成字节数	7EFADCH	DONE[15:8]								0000,0000
DMA_QSPI_ITVH	QSPI_DMA 时间间隔寄存器	7EFADEH	ITV[15:8]								0000,0000
DMA_QSPI_ITVL	QSPI_DMA 时间间隔寄存器	7EFADFH	ITV[7:0]								0000,0000
DMA_P2P_CR1	P2P DMA 控制寄存器 1	7EFAF0H	SRC1[3:0]				DEST1[3:0]				0000,0000
DMA_P2P_CR2	P2P DMA 控制寄存器 2	7EFAF1H	SRC2[3:0]				DEST2[3:0]				0000,0000
DMA_ARB_CFG	DMA 总裁配置寄存器	7EFAF8H	WTRREN	-	-	-	STASEL[3:0]-				0xxx,0000
DMA_ARB_STA	DMA 总裁状态寄存器	7EFAF9H	STA[7:0]								0000,0000

34.12.2 QSPI_DMA 配置寄存器（DMA_QSPI_CFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_QSPI_CFG	7EFAD0H	QSPIIE	ACT_WR	ACT_RD	-	QSPIIP[1:0]		QSPIPTY[1:0]	

QSPIIE：QSPI_DMA 中断使能控制位

0：禁止 QSPI_DMA 中断

1：允许 QSPI_DMA 中断

ACT_WR：QSPI_DMA 写数据控制位

0：禁止 QSPI_DMA 写数据。

1：允许 QSPI_DMA 写数据。

ACT_RD：QSPI_DMA 读数据控制位

0：禁止 QSPI_DMA 读数据。

1：允许 QSPI_DMA 读数据。

QSPIIP[1:0]：QSPI_DMA 中断优先级控制位

QSPIIP[1:0]	中断优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

QSPIPTY[1:0]：QSPI_DMA 数据总线访问优先级控制位

QSPIPTY [1:0]	总线优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

34.12.3 QSPI_DMA 控制寄存器（DMA_QSPI_CR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_QSPI_CR	7EFA21H	ENQSPI	TRIG_W	TRIG_R	-	-	-	CLRWFIFO	CLRRFIFO

ENQSPI：QSPI_DMA 功能使能控制位

0：禁止 QSPI_DMA 功能

1：允许 QSPI_DMA 功能

TRIG_W：QSPI_DMA 触发写控制位

0：写 0 无效

1：写 1 开始 QSPI_DMA 写操作，

TRIG_R：QSPI_DMA 触发读控制位

0：写 0 无效

1：写 1 开始 QSPI_DMA 读操作，

CLRWFIFO：清除 QSPI_DMA 写 FIFO 控制位

0：写 0 无效

1：开始 QSPI_DMA 操作前，先清空 QSPI_DMA 内置的 FIFO

CLRRFIFO：清除 QSPI_DMA 读 FIFO 控制位

0：写 0 无效

1：开始 QSPI_DMA 操作前，先清空 QSPI_DMA 内置的 FIFO

34.12.4 QSPI_DMA 状态寄存器（DMA_QSPI_STA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_QSPI_STA	7EFA22H	-	-	-	-	-	-	-	QSPIIF

QSPIIF：QSPI_DMA 中断请求标志位，当 QSPI_DMA 数据交换完成后，硬件自动将 QSPIIF 置 1，若使能 QSPI_DMA 中断则进入中断服务程序。标志位需软件清零

34.12.5 QSPI_DMA 传输总字节寄存器（DMA_QSPI_AMT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_QSPI_AMT	7EFA23H	AMT[7:0]							
DMA_QSPI_AMTH	7EFA84H	AMT[15:8]							

AMT[15:0]：设置需要进行数据读写的字节数。

注：实际读写的字节数为（AMT+1），即当 AMT 设置为 0 时，读写 1 字节，当 AMT 设置 255 时，读写 256 字节。特别注意 AMT 设置为 FFFFH 时表示无限循环模式。

34.12.6 QSPI_DMA 传输完成字节寄存器（DMA_QSPI_DONE）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_QSPI_DONE	7EFA24H	DONE[7:0]							
DMA_QSPI_DONEH	7EFA85H	DONE[15:8]							

DONE[15:0]：当前已经传输完成的字节数。

34.12.7 QSPI_DMA 发送地址寄存器（DMA_QSPI_TXAx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_QSPI_TXAH	7EFA25H	ADDR[15:8]							
DMA_QSPI_TXAL	7EFA26H	ADDR[7:0]							

DMA_QSPI_TXA：设置进行数据传输时的源地址。执行 QSPI_DMA 操作时会从这个地址开始读数据。

34.12.8 QSPI_DMA 接收地址寄存器（DMA_QSPI_RXAx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_QSPI_RXAH	7EFA27H	ADDR[15:8]							
DMA_QSPI_RXAL	7EFA28H	ADDR[7:0]							

DMA_QSPI_RXA：设置进行数据传输时的目标地址。执行 QSPI_DMA 操作时会从这个地址开始写入数据。

34.12.9 QSPI_DMA 传输周期（DMA_QSPI_ITVx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_QSPI_ITVH	7EFA2EH	ITV[15:8]							
DMA_QSPI_ITVL	7EFA2FH	ITV[7:0]							

DMA_QSPI_ITV[15:0]：设置 QSPI 传输的间隔时间（基于系统时钟）。

34.13 PWMA 与存储器之间的数据交换（PWMAT_DMA，PWMAR_DMA）

34.13.1 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
PWMA_DER	PWMA_DMA 事件使能寄存器	7EF948H	-	TDE	COMDE	CC4DE	CC3DE	CC2DE	CC1DE	UDE	0000,0000
PWMA_DBA	PWMA_DMA 基址寄存器	7EF949H	-	-	-	DBA[4:0]					xxx0,0000
PWMA_DBL	PWMA_DMA 基址长度寄存器	7EF94AH	-	-	-	DBL[4:0]					xxx0,0000
PWMA_DMACR	PWMA_DMA 控制寄存器	7EF94BH	-	-	-	DSKIP	DDIR	DMAEN	SIZE[1:0]		xxx0,0000
DMA_PWMAT_CFG	PWMAT_DMA 配置寄存器	7EF980H	PWMATIE	-	-	-	PWMATMIP[1:0]		PWMATBAP[1:0]		0xxx,0000
DMA_PWMAT_CR	PWMAT_DMA 控制寄存器	7EF981H	ENPWMAT	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_PWMAT_STA	PWMAT_DMA 状态寄存器	7EF982H	RDOV	-	-	-	-	TXOVW	-	PWMATIF	0xxx,x0x0
DMA_PWMAT_AMTH	PWMAT_DMA 传输总字节数	7EF984H	AMT[15:8]								0000,0000
DMA_PWMAT_AMT	PWMAT_DMA 传输总字节数	7EF985H	AMT[7:0]								0000,0000
DMA_PWMAT_DONEH	PWMAT_DMA 传输完成字节数	7EF986H	DONE[15:8]								0000,0000
DMA_PWMAT_DONE	PWMAT_DMA 传输完成字节数	7EF987H	DONE [7:0]								0000,0000
DMA_PWMAT_TXAH	PWMAT_DMA 发送高地址	7EF988H	ADR[15:8]								0000,0000
DMA_PWMAT_TXAL	PWMAT_DMA 发送低地址	7EF989H	ADR[7:0]								0000,0000
DMA_PWMA_ITVH	PWMAT_DMA 时间间隔寄存器	7EF98EH	ITV[15:8]								0000,0000
DMA_PWMA_ITVL	PWMAT _DMA 时间间隔寄存器	7EF98FH	ITV[7:0]								0000,0000
DMA_PWMAR_CFG	PWMAR_DMA 配置寄存器	7EF990H	PWMARIE	-	-	-	PWMARMIP[1:0]		PWMARBAP[1:0]		0xxx,0000
DMA_PWMAR_CR	PWMAR_DMA 控制寄存器	7EF991H	ENPWMAR	TRIG	-	-	-	-	-	CLRFIFO	00xx,xxx0
DMA_PWMAR_STA	PWMAR_DMA 状态寄存器	7EF992H	WROV	-	-	-	-	-	RXLOSS	PWMARIF	0xxx,xx00
DMA_PWMAR_AMTH	PWMAR_DMA 传输总字节数	7EF994H	AMT[15:8]								0000,0000
DMA_PWMAR_AMT	PWMAR_DMA 传输总字节数	7EF995H	AMT[7:0]								0000,0000
DMA_PWMAR_DONEH	PWMAR_DMA 传输完成字节数	7EF996H	DONE[15:8]								0000,0000
DMA_PWMAR_DONE	PWMAR_DMA 传输完成字节数	7EF997H	DONE [7:0]								0000,0000
DMA_PWMAR_RXAH	PWMAR_DMA 接收高地址	7EF998H	ADR[15:8]								0000,0000
DMA_PWMAR_RXAL	PWMAR_DMA 接收低地址	7EF999H	ADR[7:0]								0000,0000

34.13.2 PWMAT_DMA 配置寄存器（DMA_PWMAT_CFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_PWMAT_CFG	7EF980H	PWMATIE	-	-	-	PWMATIP[1:0]		PWMATPTY[1:0]	

PWMATIE：PWMAT_DMA 中断使能控制位

0：禁止 PWMAT_DMA 中断

1：允许 PWMAT_DMA 中断

PWMATIP[1:0]：PWMAT_DMA 中断优先级控制位

PWMATIP[1:0]	中断优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

PWMATPTY[1:0]：PWMAT_DMA 数据总线访问优先级控制位

PWMATPTY [1:0]	总线优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

34.13.3 PWMAT_DMA 控制寄存器（DMA_PWMAT_CR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_PWMAT_CR	7EF981H	ENPWMAT	TRIG	-	-	-	-	-	-

ENPWMAT：PWMAT_DMA 功能使能控制位

0：禁止 PWMAT_DMA 功能

1：允许 PWMAT_DMA 功能

TRIG：PWMAT_DMA 发送触发控制位

0：写 0 无效

1：写 1 开始 PWMAT_DMA 自动发送数据

34.13.4 PWMAT_DMA 状态寄存器（DMA_PWMAT_STA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_PWMAT_STA	7EF982H	RDOV	-	-	-	-	TXOVW	-	PWMATIF

PWMATIF：PWMAT_DMA 中断请求标志位，当 PWMAT_DMA 数据发送完成后，硬件自动将 PWMATIF 置 1，若使能 PWMAT_DMA 中断则进入中断服务程序。标志位需软件清零

TXOVW：PWMAT_DMA 数据覆盖标志位。PWMAT_DMA 正在数据传输过程中，写 PWMA 数据寄存器 PWMATXD 时，会导致数据传输失败，此时硬件硬件自动将 TXOVW 置 1。标志位需软件清零

RDOV：当源 XRAM 地址超过当前 XRAM 所在地址的 64K 时，RDOV 被置 1。（只读位）

34.13.5 PWMAT_DMA 传输总字节寄存器（DMA_PWMAT_AMT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_PWMAT_AMTH	7EF984H	AMT[15:8]							
DMA_PWMAT_AMT	7EF985H	AMT[7:0]							

AMT[15:0]：设置需要进行数据读写的字节数。

注：实际读写的字节数为（AMT+1），即当 AMT 设置为 0 时，读写 1 字节，当 AMT 设置 255 时，读写 256 字节。特别注意 AMT 设置为 FFFFH 时表示无限循环模式。

34.13.6 PWMAT_DMA 传输完成字节寄存器（DMA_PWMAT_DONE）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_PWMAT_DONEH	7EF9869H	DONE[15:8]							
DMA_PWMAT_DONE	7EF987H	DONE[7:0]							

DONE[15:0]：当前已经发送完成的字节数。

34.13.7 PWMAT_DMA 发送地址寄存器（DMA_PWMAT_TXAx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_PWMAT_TXAH	7EF988H	ADDR[15:8]							
DMA_PWMAT_TXAL	7EF989H	ADDR[7:0]							

DMA_PWMAT_TXA：设置自动发送数据的源地址。执行 PWMAT_DMA 操作时会从这个地址开始读数据。

34.13.8 PWMA_DMA 传输周期（DMA_PWMA_ITVx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_PWMA_ITVH	7EF98EH	ITV[15:8]							
DMA_PWMA_ITVL	7EF98FH	ITV[7:0]							

DMA_PWMA_ITV[15:0]：设置 PWMA 发送和接收的间隔时间（基于系统时钟）。

34.13.9 PWMAR_DMA 配置寄存器（DMA_PWMAR_CFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_PWMAR_CFG	7EF990H	PWMARIE	-	-	-	PWMARIP[1:0]		PWMARPTY[1:0]	

PWMARIE：PWMAR_DMA 中断使能控制位

0：禁止 PWMAR_DMA 中断

1：允许 PWMAR_DMA 中断

PWMARIP[1:0]：PWMAR_DMA 中断优先级控制位

PWMARIP[1:0]	中断优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

PWMARPTY[1:0]：PWMAR_DMA 数据总线访问优先级控制位

PWMARPTY [1:0]	总线优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

34.13.10 PWMAR_DMA 控制寄存器（DMA_PWMAR_CR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_PWMAR_CR	7EF991H	ENPWMAR	TRIG	-	-	-	-	-	CLRFIFO

ENPWMAR：PWMAR_DMA 功能使能控制位

0：禁止 PWMAR_DMA 功能

1：允许 PWMAR_DMA 功能

TRIG：PWMAR_DMA 接收触发控制位

0：写 0 无效

1：写 1 开始 PWMAR_DMA 自动接收数据

CLRFIFO：清除 PWMAR_DMA 接收 FIFO 控制位

0：写 0 无效

1：开始 PWMAR_DMA 操作前，先清空 PWMAR_DMA 内置的 FIFO

34.13.11 PWMAR_DMA 状态寄存器（DMA_PWMAR_STA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_PWMAR_STA	7EF992H	WROV	-	-	-	-	-	RXLOSS	PWMARIF

PWMARIF：PWMAR_DMA 中断请求标志位，当 PWMAR_DMA 接收数据完成后，硬件自动将 PWMARIF 置 1，若使能 PWMAR_DMA 中断则进入中断服务程序。标志位需软件清零

RXLOSS：PWMAR_DMA 接收数据丢弃标志位。PWMAR_DMA 操作过程中，当 XRAM 总线过于繁忙，来不及清空 PWMAR_DMA 的接收 FIFO 导致 PWMAR_DMA 接收的数据自动丢弃时，硬件硬件自动将 RXLOSS 置 1。标志位需软件清零

WROV：当目标 XRAM 地址超过当前 XRAM 所在地址的 64K 时，WROV 被置 1。（只读位）

34.13.12 PWMAR_DMA 传输总字节寄存器（DMA_PWMAR_AMT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_PWMAR_AMTH	7EF994H	AMT[15:8]							
DMA_PWMAR_AMT	7EF995H	AMT[7:0]							

AMT[15:0]：设置需要进行数据读写的字节数。

注：实际读写的字节数为（AMT+1），即当 AMT 设置为 0 时，读写 1 字节，当 AMT 设置 255 时，读写 256 字节。特别注意 AMT 设置为 FFFFH 时表示无限循环模式。

34.13.13 PWMAR_DMA 传输完成字节寄存器（DMA_PWMAR_DONE）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_PWMAR_DONEH	7EF996H	DONE[15:8]							
DMA_PWMAR_DONE	7EF997H	DONE[7:0]							

DONE[15:0]：当前已经接收完成的字节数。

34.13.14 PWMAR_DMA 接收地址寄存器（DMA_PWMAR_RXAx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_PWMAR_RXAH	7EF998H	ADDR[15:8]							
DMA_PWMAR_RXAL	7EF999H	ADDR[7:0]							

DMA_PWMAR_RXA：设置自动接收数据的目标地址。执行 PWMAR_DMA 操作时会从这个地址开始写数据。

34.13.15 PWMA_DMA 事件使能寄存器（PWMA_DER）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_DER	7EF948H	-	TDE	COMDE	CC4DE	CC3DE	CC2DE	CC1DE	UDE

TDE：使能 PWMA 触发中断为 PWMA_DMA 触发事件

COMDE：使能 PWMA 的 COM 中断为 PWMA_DMA 触发事件

CC4DE：使能 PWMA 的捕获/比较 4 中断为 PWMA_DMA 触发事件

CC3DE：使能 PWMA 的捕获/比较 3 中断为 PWMA_DMA 触发事件

CC2DE：使能 PWMA 的捕获/比较 2 中断为 PWMA_DMA 触发事件

CC1DE：使能 PWMA 的捕获/比较 1 中断为 PWMA_DMA 触发事件

UDE：使能 PWMA 的更新中断为 PWMA_DMA 触发事件

34.13.16 PWMA_DMA 基址寄存器（PWMA_DBA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_DBA	7EF949H	-	-	-	DBA[4:0]				

DBA[4:0]：设置 PWMA_DMA 的寄存器基址。

PWMA_DMA 基地址映射表

基地址	Byte3	Byte2	Byte1	Byte0
0x00	-	-	-	PWMA_CR1
0x01	-	PWMA_CR3	PWMA_OISR	PWMA_CR2
0x02	-	-	PWMA_ETR	PWMA_SMCR
0x03	-	-	PWMA_DER	PWMA_IER
0x04	-	PWMA_SR3	PWMA_SR2	PWMA_SR1
0x05	-	-	-	PWMA_EGR
0x06	PWMA_CCMR2X	PWMA_CCMR1X	PWMA_CCMR2	PWMA_CCMR1
0x07	PWMA_CCMR4X	PWMA_CCMR3X	PWMA_CCMR4	PWMA_CCMR3
0x08	-	PWMA_CCER3	PWMA_CCER2	PWMA_CCER1
0x09	-	-	PWMA_CNTH	PWMA_CNTL
0x0A	-	-	PWMA_PSCRH	PWMA_PSCRL
0x0B	-	-	PWMA_ARRH	PWMA_ARRL
0x0C	-	-	-	PWMA_RCR
0x0D	-	-	PWMA_CCR1H	PWMA_CCR1L
0x0E	-	-	PWMA_CCR2H	PWMA_CCR2L
0x0F	-	-	PWMA_CCR3H	PWMA_CCR3L
0x10	-	-	PWMA_CCR4H	PWMA_CCR4L
0x11	-	-	PWMA_BKR	PWMA_DTR
0x12	-	-	PWMA_DBL	PWMA_DBA
0x13	-	-	-	-
0x14	-	-	-	-
0x15	PWMA_CCMR6X	PWMA_CCMR5X	PWMA_CCMR6	PWMA_CCMR5
0x16	PWMA_CCR5X	-	PWMA_CCR5H	PWMA_CCR5L
0x17	-	-	PWMA_CCR6H	PWMA_CCR6L
		-	-	-

34.13.17 PWMA_DMA 突发传输次数寄存器（PWMA_DBL）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_DBL	7EF94AH	-	-	-	DBL[4:0]				

DBL[4:0]：设置 PWMA_DMA 的突发传输（Burst transmission）次数

DBL[4:0]	长度
00H	1 次
01H	2 次
02H	3 次
…	…
1FH	32 次

34.13.18 PWMA_DMA 控制寄存器（PWMA_DMACR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_DMACR	7EF948H	-	-	-	DSKIP	DDIR	DMAEN	SIZE[1:0]	

DSKIP：设置是否跳过保留的字节

0：不跳过

1：跳过

DDIR：PWMA_DMA 方向

0：XRAM 到 PWMA（输出）

1：PWMA 到 XRAM（输入）

DMAEN：PWMA_DMA 使能位

0：禁止 PWMA_DMA

1：使能 PWMA_DAM

SIZE[1:0]：PWMA DMA 读写字节数

00：读写 1 字节（Byte0）

01：读写 2 字节（Byte1，Byte0）

10：读写 3 字节（Byte2，Byte1，Byte0）

11：读写 4 字节（Byte3，Byte2，Byte1，Byte0）

例如：PWMA_DBA=0x06，PWMA_DBL=1，PWMA_DMACR.SIZE=3，DMA 缓冲区为字节数据 Data。

则当第 1 次 PWMA 的 DMA 更新请求时，

Data[0] 被传送到 PWMA_CCMR2X，

Data[1] 被传送到 PWMA_CCMR1X，

Data[2] 被传送到 PWMA_CCMR2，

Data[3] 被传送到 PWMA_CCMR1，

Data[4] 被传送到 PWMA_CCMR4X，

Data[5] 被传送到 PWMA_CCMR3X，

Data[6] 被传送到 PWMA_CCMR4，

Data[7] 被传送到 PWMA_CCMR3；

当第 2 次 PWMA 的 DMA 更新请求时，

Data[8] 被传送到 PWMA_CCMR2X，

Data[9] 被传送到 PWMA_CCMR1X，
Data[10] 被传送到 PWMA_CCMR2，
Data[11] 被传送到 PWMA_CCMR1，
Data[12] 被传送到 PWMA_CCMR4X，
Data[13] 被传送到 PWMA_CCMR3X，
Data[14] 被传送到 PWMA_CCMR4，
Data[15] 被传送到 PWMA_CCMR3

34.14 外设与外设之间的数据交换（P2P_DMA）

P2P 使用注意事项：

目前测试发现 Ai8051U 的 QSPI 到其他外设的 P2P 方式 DMA 会影响 XRAM 的数据，在使用 QSPI 到其他外设的 P2P 功能时，**必须将 QSPI 存储接收数据的 XRAM 起始地址设置到 32K XRAM 的后面部分。注意：由于内部的 XRAM 只有 32K，有效的地址范围为 0000H~7FFFH，当地址超出 32K 时，地址会重新回到 0，所以设置的 DMA 的 XRAM 起始地址+DMA 的传输数据长度的总和不能大于 32K。**

【例如】当使用 QSPI 到 i8080 彩屏接口的 P2P 功能时，可将{DMA_QSPI_RXAH, DMA_QSPI_RXAL}中的地址设置到 **20K** 开始的地方，地址为 **5000H**。由于地址设置在 20K 起始的地方，所以每次 P2P 的最大传输数据量不能大于 **32K-20K=12K** 字节。

另外，寄存器{DMA_QSPI_RXAH, DMA_QSPI_RXAL}中设置的是每次 DMA 读取 XRAM 的起始地址，在 DMA 过程中{DMA_QSPI_RXAH, DMA_QSPI_RXAL}的值不会改变，所以只需要在初始化时设置好{DMA_QSPI_RXAH, DMA_QSPI_RXAL}中的初始值就可以了，不需要每次 DMA 时都设置。

34.14.1 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
DMA_P2P_CR1	P2P DMA 控制寄存器 1	7EFAF0H	SRC1[3:0]				DEST1[3:0]				0000,0000
DMA_P2P_CR2	P2P DMA 控制寄存器 2	7EFAF1H	SRC2[3:0]				DEST2[3:0]				0000,0000

34.14.2 P2P_DMA 配置寄存器 1（DMA_P2P_CR1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_P2P_CR1	7EFAF0H	SRC1[3:0]				DEST1[3:0]			

SRC1[3:0]：选择第一组 P2P_DMA 中的源外设（接收外设）

SRC1[3:0]	源外设
0	无外设
1	串口 1（接收）
2	串口 2（接收）
3	串口 3（接收）
4	串口 4（接收）
5	SPI（接收）
6	I2C（接收）
7	LCM（接收）
8	QSPI（接收）
9	ADC（接收）
10	I2S（接收）
11~15	保留

DEST1[3:0]：选择第一组 P2P_DMA 中的目标外设（发送外设）

DEST1[3:0]	目标外设
0	无外设
1	串口 1（发送）
2	串口 2（发送）
3	串口 3（发送）
4	串口 4（发送）
5	SPI（发送）
6	I2C（发送）
7	LCM（发送）
8	QSPI（发送）
9	ADC（发送）
10	I2S（发送）
11~15	保留

34.14.3 P2P_DMA 配置寄存器 2（DMA_P2P_CR2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_P2P_CR2	7EFAF1H	SRC2[3:0]				DEST23:0]			

SRC2[3:0]：选择第二组 P2P_DMA 中的源外设（接收外设）

SRC2[3:0]	源外设
0	无外设
1	串口 1（接收）
2	串口 2（接收）
3	串口 3（接收）
4	串口 4（接收）
5	SPI（接收）
6	I2C（接收）
7	LCM（接收）
8	QSPI（接收）
9	ADC（接收）
10	I2S（接收）
11~15	保留

DEST2[3:0]：选择第二组 P2P_DMA 中的目标外设（发送外设）

DEST2[3:0]	目标外设
0	无外设
1	串口 1（发送）
2	串口 2（发送）
3	串口 3（发送）
4	串口 4（发送）
5	SPI（发送）
6	I2C（发送）
7	LCM（发送）
8	QSPI（发送）
9	ADC（发送）
10	I2S（发送）
11~15	保留

34.15　范例程序

34.15.1　串口 1 中断模式与电脑收发测试 - DMA 接收超时中断

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                    //头文件见下载软件

/**************  功能说明  **************
串口 1 全双工中断方式收发通讯程序。通过 PC 向 MCU 发送数据, MCU 将收到的数据自动存入 DMA 空间。当一次性接收的内容存满设置的 DMA 空间后，通过串口 1 的 DMA 自动发送功能把存储空间的数据输出。利用串口接收中断进行超时判断，超时没有收到新的数据，表示一串数据已经接收完毕，将已接收的内容输出，并清除 DMA 空间。用定时器做波特率发生器，建议使用 1T 模式(除非低波特率用 12T)，并选择可被波特率整除的时钟频率，以提高精度。
下载时, 选择时钟 22.1184MHz (用户可自行修改频率).
*****************************************/

#include "stdio.h"

#define   MAIN_Fosc       22118400L          //定义主时钟（精确计算 115200 波特率）
#define   Baudrate1       115200L
#define   Timer0_Reload   (65536UL -(MAIN_Fosc / 1000))

#define   DMA_AMT_LEN     255                //设置传输总字节数(0~255)：DMA_AMT_LEN+1

bit       B_1ms;                             //1ms 标志
bit       DMATxFlag;
bit       DMARxFlag;
bit       BusyFlag;
u8        Rx_cnt;
u8        RX1_TimeOut;

u8        xdata DMABuffer[256];

void UART1_config(u8 brt);
void DMA_Config(void);

void UartPutc(unsigned char dat)
{
    BusyFlag = 1;
    SBUF = dat;
    while(BusyFlag);
}

char putchar(char c)
{
    UartPutc(c);
    return c;
}

void main(void)
{
    u16 i;
```

```
CKCON = 0x00;                                   //设置外部数据总线速度为最快
WTST = 0x00;                                    //设置程序代码等待参数,
                                                //赋值为 0 可将 CPU 执行程序的速度设置为最快

P0M1 = 0x00;    P0M0 = 0x00;                    //设置为准双向口
P1M1 = 0x00;    P1M0 = 0x00;                    //设置为准双向口
P2M1 = 0x00;    P2M0 = 0x00;                    //设置为准双向口
P3M1 = 0x00;    P3M0 = 0x00;                    //设置为准双向口
P4M1 = 0x00;    P4M0 = 0x00;                    //设置为准双向口
P5M1 = 0x00;    P5M0 = 0x00;                    //设置为准双向口
P6M1 = 0x00;    P6M0 = 0x00;                    //设置为准双向口
P7M1 = 0x00;    P7M0 = 0x00;                    //设置为准双向口

for(i=0; i<256; i++)
{
    DMABuffer[i] = i;
}

AUXR = 0x80;                                    //Timer0 set as 1T, 16 bits timer auto-reload,
TH0 = (u8)(Timer0_Reload / 256);
TL0 = (u8)(Timer0_Reload % 256);
ET0 = 1;                                        //Timer0 interrupt enable
TR0 = 1;                                        //Tiner0 run

UART1_config(1);                                //使用 Timer1 做波特率.
DMA_Config();
EA = 1;                     //允许总中断

printf("UART1 DMA Timeout Programme!\r\n");     //UART1 发送一个字符串
DMATxFlag = 0;
DMARxFlag = 0;

while (1)
{
    if((DMATxFlag) && (DMARxFlag))              //判断发送完成标志与接收完成标志
    {
        Rx_cnt = 0;
        RX1_TimeOut = 0;
        printf("\r\nUART1 DMA FULL!\r\n");      //UART1 发送一个字符串
        DMATxFlag = 0;
        DMA_UR1T_CR = 0xc0;                     //bit7 1:使能 UART1_DMA,
                                                //bit6 1:开始 UART1_DMA 自动发送

        DMARxFlag = 0;
        DMA_UR1R_CR = 0xa1;                     //bit7 1:使能 UART1_DMA,
                                                //bit5 1:开始 UART1_DMA 自动接收,
                                                //bit0 1:清除 FIFO
    }

    if(B_1ms)                                   //1ms 到
    {
        B_1ms = 0;
        if(RX1_TimeOut > 0)                     //超时计数
        {
            if(--RX1_TimeOut == 0)
            {
                DMA_UR1R_CR = 0x00;             //关闭 UART1_DMA
                printf("\r\nUART1 Timeout!\r\n");//UART1 发送一个字符串
```

```
                    for(i=0;i<Rx_cnt;i++) UartPutc(DMABuffer[i]);
                    printf("\r\n");

                    Rx_cnt = 0;
                    DMA_UR1R_CR = 0xa1;             //bit7 1:使能 UART1_DMA,
                                                    //bit5 1:开始 UART1_DMA 自动接收,
                                                    //bit0 1:清除 FIFO
                }
            }
        }
    }
}

void DMA_Config(void)
{
    EAXFR = 1;                                      //允许访问扩展的特殊寄存器，XFR
                                                    //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                  //允许访问扩展的特殊寄存器，XFR
                                                    //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                       //设置取程序代码等待时间，
                                                    //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                      //设置访问片内的 xdata 速度，
                                                    //赋值为 0 表示用最快速度访问，不增加额外的等待时间
    DMA_UR1T_CFG = 0x80;                            //bit7 1:Enable Interrupt
    DMA_UR1T_STA = 0x00;
    DMA_UR1T_AMTH = (u8)(DMA_AMT_LEN >> 8);
    DMA_UR1T_AMT = (u8)(DMA_AMT_LEN);               //设置传输总字节数：n+1
    DMA_UR1T_TXAH = (u8)(((u16)DMABuffer) >> 8);
    DMA_UR1T_TXAL = (u8)((u16)DMABuffer);
    DMA_UR1T_CR = 0xc0;                             //bit7 1:使能 UART1_DMA,
                                                    //bit6 1:开始 UART1_DMA 自动发送

    DMA_UR1R_CFG = 0x80;                            //bit7 1:Enable Interrupt
    DMA_UR1R_STA = 0x00;
    DMA_UR1R_AMTH = (u8)(DMA_AMT_LEN >> 8);
    DMA_UR1R_AMT = (u8)(DMA_AMT_LEN);               //设置传输总字节数：n+1
    DMA_UR1R_RXAH = (u8)(((u16)DMABuffer) >> 8);
    DMA_UR1R_RXAL = (u8)((u16)DMABuffer);
    DMA_UR1R_CR = 0xa1;                             //bit7 1:使能 UART1_DMA,
                                                    //bit5 1:开始 UART1_DMA 自动接收,
                                                    //bit0 1:清除 FIFO
}

void SetTimer2Baudraye(u16 dat)
{
    AUXR &= ~(1<<4);                                //Timer stop
    AUXR &= ~(1<<3);                                //Timer2 set As Timer
    AUXR |=  (1<<2);                                //Timer2 set as 1T mode
    T2H = dat / 256;
    T2L = dat % 256;
    IE2  &= ~(1<<2);                                //禁止中断
    AUXR |=  (1<<4);                                //Timer run enable
}

void UART1_config(u8 brt)                           //选择波特率:
                                                    //2: 使用 Timer2 做波特率,
                                                    //其它值: 使用 Timer1 做波特率.
{
    /********** 波特率使用定时器 2 **********/
```

```
    if(brt == 2)
    {
        AUXR |= 0x01;                                   //S1 BRT Use Timer2;
        SetTimer2Baudraye(65536UL - (MAIN_Fosc / 4) / Baudrate1);
    }

    /********** 波特率使用定时器 1 *****************/
    else
    {
        TR1 = 0;
        AUXR &= ~0x01;                                  //S1 BRT Use Timer1;
        AUXR |=  (1<<6);                                //Timer1 set as 1T mode
        TMOD &= ~(1<<6);                                //Timer1 set As Timer
        TMOD &= ~0x30;                                  //Timer1_16bitAutoReload;
        TH1 = (u8)((65536UL - (MAIN_Fosc / 4) / Baudrate1) / 256);
        TL1 = (u8)((65536UL - (MAIN_Fosc / 4) / Baudrate1) % 256);
        ET1 = 0;                                        //禁止中断
        INTCLKO &= ~0x02;                               //不输出时钟
        TR1  = 1;
    }
    /*************************************************/

    SCON = (SCON & 0x3f) | 0x40;                        //UART1 模式:
                                                        //0x00:  同步移位输出,
                                                        //0x40: 8 位数据,可变波特率,
                                                        //0x80: 9 位数据,固定波特率,
                                                        //0xc0: 9 位数据,可变波特率
//  PS = 1;                                             //高优先级中断
    ES = 1;                                             //允许中断
    REN = 1;                                            //允许接收
    P_SW1 &= 0x3f;
    P_SW1 |= 0x00;                                      //UART1 switch to:
                                                        //0x00: P3.0 P3.1,
                                                        //0x40: P3.6 P3.7,
                                                        //0x80: P1.6 P1.7,
                                                        //0xC0: P4.3 P4.4

    RX1_TimeOut = 0;
}

void UART1_int (void) interrupt 4
{
    if(RI)
    {
        RI = 0;
        Rx_cnt++;
        if(Rx_cnt >= DMA_AMT_LEN) Rx_cnt = 0;
        RX1_TimeOut = 5;                                //如果 5ms 没收到新的数据，判定一串数据接收完毕
    }

    if(TI)
    {
        TI = 0;
        BusyFlag = 0;
    }
}

void timer0 (void) interrupt 1
{
```

```
    B_1ms = 1;                                  //1ms 标志
}

void UART1_DMA_Interrupt(void) interrupt 13
{
    if (DMA_UR1T_STA & 0x01)                    //发送完成
    {
        DMA_UR1T_STA &= ~0x01;
        DMATxFlag = 1;
    }
    if (DMA_UR1T_STA & 0x04)                    //数据覆盖
    {
        DMA_UR1T_STA &= ~0x04;
    }

    if (DMA_UR1R_STA & 0x01)                    //接收完成
    {
        DMA_UR1R_STA &= ~0x01;
        DMARxFlag = 1;
    }
    if (DMA_UR1R_STA & 0x02)                    //数据丢弃
    {
        DMA_UR1R_STA &= ~0x02;
    }
}
```

//文件：ISR.ASM
//中断号大于 31 的中断，需要进行中断入口地址重映射处理

```
        CSEG    AT      012BH           ;P0INT_VECTOR
        JMP             P0INT_ISR
        CSEG    AT      0133H           ;P1INT_VECTOR
        JMP             P1INT_ISR
        CSEG    AT      013BH           ;P2INT_VECTOR
        JMP             P2INT_ISR
        CSEG    AT      0143H           ;P3INT_VECTOR
        JMP             P3INT_ISR
        CSEG    AT      014BH           ;P4INT_VECTOR
        JMP             P4INT_ISR
        CSEG    AT      0153H           ;P5INT_VECTOR
        JMP             P5INT_ISR
        CSEG    AT      015BH           ;P6INT_VECTOR
        JMP             P6INT_ISR
        CSEG    AT      0163H           ;P7INT_VECTOR
        JMP             P7INT_ISR
        CSEG    AT      016BH           ;P8INT_VECTOR
        JMP             P8INT_ISR
        CSEG    AT      0173H           ;P9INT_VECTOR
        JMP             P9INT_ISR
        CSEG    AT      017BH           ;M2MDMA_VECTOR
        JMP             M2MDMA_ISR
        CSEG    AT      0183H           ;ADCDMA_VECTOR
        JMP             ADCDMA_ISR
        CSEG    AT      018BH           ;SPIDMA_VECTOR
        JMP             SPIDMA_ISR
        CSEG    AT      0193H           ;U1TXDMA_VECTOR
        JMP             U1TXDMA_ISR
        CSEG    AT      019BH           ;U1RXDMA_VECTOR
        JMP             U1RXDMA_ISR
```

```
            CSEG    AT      01A3H               ;U2TXDMA_VECTOR
            JMP             U2TXDMA_ISR
            CSEG    AT      01ABH               ;U2RXDMA_VECTOR
            JMP             U2RXDMA_ISR
            CSEG    AT      01B3H               ;U3TXDMA_VECTOR
            JMP             U3TXDMA_ISR
            CSEG    AT      01BBH               ;U3RXDMA_VECTOR
            JMP             U3RXDMA_ISR
            CSEG    AT      01C3H               ;U4TXDMA_VECTOR
            JMP             U4TXDMA_ISR
            CSEG    AT      01CBH               ;U4RXDMA_VECTOR
            JMP             U4RXDMA_ISR
            CSEG    AT      01D3H               ;LCMDMA_VECTOR
            JMP             LCMDMA_ISR
            CSEG    AT      01DBH               ;LCMIF_VECTOR
            JMP             LCMIF_ISR

P0INT_ISR:
P1INT_ISR:
P2INT_ISR:
P3INT_ISR:
P4INT_ISR:
P5INT_ISR:
P6INT_ISR:
P7INT_ISR:
P8INT_ISR:
P9INT_ISR:
M2MDMA_ISR:
ADCDMA_ISR:
SPIDMA_ISR:
U1TXDMA_ISR:
U1RXDMA_ISR:
U2TXDMA_ISR:
U2RXDMA_ISR:
U3TXDMA_ISR:
U3RXDMA_ISR:
U4TXDMA_ISR:
U4RXDMA_ISR:
LCMDMA_ISR:
LCMIF_ISR:

            JMP             006BH

            END
```

34.15.2　串口 1 中断模式与电脑收发测试 - DMA 数据校验

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                  //头文件见下载软件
```

/**********　功能说明　　**********

串口 1 全双工中断方式收发通讯程序。通过 PC 向 MCU 发送数据, MCU 将收到的数据自动存入 DMA 空间。数据包的最后两个字节作为校验位，例程以 crc16_ccitt 算法进行校验。当 DMA 空间存满设置大小

的内容后，对有效数据进行校验计算，然后与最后两位校验位进行对比。通过串口 1 的 DMA 自动发送功能把存储空间的数据输出。用定时器做波特率发生器，建议使用 1T 模式(除非低波特率用 12T)，并选择可被波特率整除的时钟频率，以提高精度。
下载时，选择时钟 22.1184MHz (用户可自行修改频率)。
***/

```
#include "stdio.h"
#include "crc16.h"

#define    MAIN_Fosc        22118400L          //定义主时钟（精确计算 115200 波特率）
#define    Baudrate1        115200L

#define    DMA_AMT_LEN      255                //设置传输总字节数(0~255)：DMA_AMT_LEN+1

bit        DMATxFlag;
bit        DMARxFlag;

u8         xdata DMABuffer[256];

void UART1_config(u8 brt);
void DMA_Config(void);

void UartPutc(unsigned char dat)
{
    SBUF = dat;
    while(TI == 0);
    TI = 0;
}

char putchar(char c)
{
    UartPutc(c);
    return c;
}
```

/*****CRC 计算函数********/

```
u16 crc16_ccitt(u8 *pbuf, u16 len)
{
    unsigned short code crc16_ccitt_table[256] =
    {
        0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50A5, 0x60C6, 0x70E7,
        0x8108, 0x9129, 0xA14A, 0xB16B, 0xC18C, 0xD1AD, 0xE1CE, 0xF1EF,
        0x1231, 0x0210, 0x3273, 0x2252, 0x52B5, 0x4294, 0x72F7, 0x62D6,
        0x9339, 0x8318, 0xB37B, 0xA35A, 0xD3BD, 0xC39C, 0xF3FF, 0xE3DE,
        0x2462, 0x3443, 0x0420, 0x1401, 0x64E6, 0x74C7, 0x44A4, 0x5485,
        0xA56A, 0xB54B, 0x8528, 0x9509, 0xE5EE, 0xF5CF, 0xC5AC, 0xD58D,
        0x3653, 0x2672, 0x1611, 0x0630, 0x76D7, 0x66F6, 0x5695, 0x46B4,
        0xB75B, 0xA77A, 0x9719, 0x8738, 0xF7DF, 0xE7FE, 0xD79D, 0xC7BC,
        0x48C4, 0x58E5, 0x6886, 0x78A7, 0x0840, 0x1861, 0x2802, 0x3823,
        0xC9CC, 0xD9ED, 0xE98E, 0xF9AF, 0x8948, 0x9969, 0xA90A, 0xB92B,
        0x5AF5, 0x4AD4, 0x7AB7, 0x6A96, 0x1A71, 0x0A50, 0x3A33, 0x2A12,
        0xDBFD, 0xCBDC, 0xFBBF, 0xEB9E, 0x9B79, 0x8B58, 0xBB3B, 0xAB1A,
        0x6CA6, 0x7C87, 0x4CE4, 0x5CC5, 0x2C22, 0x3C03, 0x0C60, 0x1C41,
        0xEDAE, 0xFD8F, 0xCDEC, 0xDDCD, 0xAD2A, 0xBD0B, 0x8D68, 0x9D49,
        0x7E97, 0x6EB6, 0x5ED5, 0x4EF4, 0x3E13, 0x2E32, 0x1E51, 0x0E70,
        0xFF9F, 0xEFBE, 0xDFDD, 0xCFFC, 0xBF1B, 0xAF3A, 0x9F59, 0x8F78,
```

```
        0x9188,  0x81A9,  0xB1CA,  0xA1EB,  0xD10C,  0xC12D,  0xF14E,  0xE16F,
        0x1080,  0x00A1,  0x30C2,  0x20E3,  0x5004,  0x4025,  0x7046,  0x6067,
        0x83B9,  0x9398,  0xA3FB,  0xB3DA,  0xC33D,  0xD31C,  0xE37F,  0xF35E,
        0x02B1,  0x1290,  0x22F3,  0x32D2,  0x4235,  0x5214,  0x6277,  0x7256,
        0xB5EA,  0xA5CB,  0x95A8,  0x8589,  0xF56E,  0xE54F,  0xD52C,  0xC50D,
        0x34E2,  0x24C3,  0x14A0,  0x0481,  0x7466,  0x6447,  0x5424,  0x4405,
        0xA7DB,  0xB7FA,  0x8799,  0x97B8,  0xE75F,  0xF77E,  0xC71D,  0xD73C,
        0x26D3,  0x36F2,  0x0691,  0x16B0,  0x6657,  0x7676,  0x4615,  0x5634,
        0xD94C,  0xC96D,  0xF90E,  0xE92F,  0x99C8,  0x89E9,  0xB98A,  0xA9AB,
        0x5844,  0x4865,  0x7806,  0x6827,  0x18C0,  0x08E1,  0x3882,  0x28A3,
        0xCB7D,  0xDB5C,  0xEB3F,  0xFB1E,  0x8BF9,  0x9BD8,  0xABBB,  0xBB9A,
        0x4A75,  0x5A54,  0x6A37,  0x7A16,  0x0AF1,  0x1AD0,  0x2AB3,  0x3A92,
        0xFD2E,  0xED0F,  0xDD6C,  0xCD4D,  0xBDAA,  0xAD8B,  0x9DE8,  0x8DC9,
        0x7C26,  0x6C07,  0x5C64,  0x4C45,  0x3CA2,  0x2C83,  0x1CE0,  0x0CC1,
        0xEF1F,  0xFF3E,  0xCF5D,  0xDF7C,  0xAF9B,  0xBFBA,  0x8FD9,  0x9FF8,
        0x6E17,  0x7E36,  0x4E55,  0x5E74,  0x2E93,  0x3EB2,  0x0ED1,  0x1EF0
    };

    u16 crc16 = 0x0000;
    u16 crc_h8, crc_l8;

    while( len-- ) {
        crc_h8 = (crc16 >> 8);
        crc_l8 = (crc16 << 8);
        crc16 = crc_l8 ^ crc16_ccitt_table[crc_h8 ^ *pbuf];
        pbuf++;
    }

    return crc16;
}

void main(void)
{
    u16 i;
    u16 CheckSum;

    CKCON = 0x00;                                   //设置外部数据总线速度为最快
    WTST = 0x00;                                    //设置程序代码等待参数,
                                                    //赋值为 0 可将 CPU 执行程序的速度设置为最快

    P0M1 = 0x00;    P0M0 = 0x00;                    //设置为准双向口
    P1M1 = 0x00;    P1M0 = 0x00;                    //设置为准双向口
    P2M1 = 0x00;    P2M0 = 0x00;                    //设置为准双向口
    P3M1 = 0x00;    P3M0 = 0x00;                    //设置为准双向口
    P4M1 = 0x00;    P4M0 = 0x00;                    //设置为准双向口
    P5M1 = 0x00;    P5M0 = 0x00;                    //设置为准双向口
    P6M1 = 0x00;    P6M0 = 0x00;                    //设置为准双向口
    P7M1 = 0x00;    P7M0 = 0x00;                    //设置为准双向口

    for(i=0; i<256; i++)
    {
        DMABuffer[i] = i;
    }

    EAXFR = 1;                                      //允许访问扩展的特殊寄存器，XFR
                                                    //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                  //允许访问扩展的特殊寄存器，XFR
                                                    //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                       //设置取程序代码等待时间，
```

```
                                                    //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                      //设置访问片内的 xdata 速度，
                                                    //赋值为 0 表示用最快速度访问，不增加额外的等待时间
    DMA_UR1T_STA = 0x00;
    UART1_config(1);
    printf("UART1 DMA CRC Programme!\r\n");

    DMA_Config();
    EA = 1;                                         //允许总中断

    DMATxFlag = 0;
    DMARxFlag = 0;

    while (1)
    {
        if((DMATxFlag) && (DMARxFlag))
        {
            CheckSum = crc16_ccitt(DMABuffer,DMA_AMT_LEN-1);
            if(((u8)CheckSum == DMABuffer[DMA_AMT_LEN-1]) &&
            ((u8)(CheckSum>>8) == DMABuffer[DMA_AMT_LEN]))
            {
                printf("\r\nOK! CheckSum = %04x\r\n",CheckSum);
            }
            else
            {
                printf("\r\nERROR! CheckSum = %04x\r\n",CheckSum);
            }
            DMATxFlag = 0;
            DMA_UR1T_CR = 0xc0;                     //bit7 1:使能 UART1_DMA,
                                                    //bit6 1:开始 UART1_DMA 自动发送
            DMARxFlag = 0;
            DMA_UR1R_CR = 0xa1;                     //bit7 1:使能 UART1_DMA,
                                                    //bit5 1:开始 UART1_DMA 自动接收,
                                                    //bit0 1:清除 FIFO
        }
    }
}

void DMA_Config(void)
{
    EAXFR = 1;                                      //允许访问扩展的特殊寄存器，XFR
                                                    //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                  //允许访问扩展的特殊寄存器，XFR
                                                    //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                       //设置取程序代码等待时间，
                                                    //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                      //设置访问片内的 xdata 速度，
                                                    //赋值为 0 表示用最快速度访问，不增加额外的等待时间
    DMA_UR1T_CFG = 0x80;                            //bit7 1:Enable Interrupt
    DMA_UR1T_STA = 0x00;
    DMA_UR1T_AMTH = (u8)(DMA_AMT_LEN >> 8);
    DMA_UR1T_AMT = (u8)(DMA_AMT_LEN);               //设置传输总字节数：n+1
    DMA_UR1T_TXAH = (u8)(((u16)DMABuffer) >> 8);
    DMA_UR1T_TXAL = (u8)((u16)DMABuffer);
    DMA_UR1T_CR = 0xc0;                             //bit7 1:使能 UART1_DMA,
                                                    //bit6 1:开始 UART1_DMA 自动发送

    DMA_UR1R_CFG = 0x80;                            //bit7 1:Enable Interrupt
    DMA_UR1R_STA = 0x00;
```

```
    DMA_UR1R_AMTH = (u8)(DMA_AMT_LEN >> 8);
    DMA_UR1R_AMT = (u8)(DMA_AMT_LEN);               //设置传输总字节数：n+1
    DMA_UR1R_RXAH = (u8)(((u16)DMABuffer) >> 8);
    DMA_UR1R_RXAL = (u8)((u16)DMABuffer);
    DMA_UR1R_CR = 0xa1;                             //bit7 1:使能 UART1_DMA,
                                                    //bit5 1:开始 UART1_DMA 自动接收, bit0 1:清除 FIFO
}

void SetTimer2Baudraye(u16 dat)                     //选择波特率:
                                                    //2: 使用 Timer2 做波特率,
                                                    //其它值: 使用 Timer1 做波特率.
{
    AUXR &= ~(1<<4);                                //Timer stop
    AUXR &= ~(1<<3);                                //Timer2 set As Timer
    AUXR |=  (1<<2);                                //Timer2 set as 1T mode
    T2H = dat / 256;
    T2L = dat % 256;
    IE2  &= ~(1<<2);                                //禁止中断
    AUXR |=  (1<<4);                                //Timer run enable
}

void UART1_config(u8 brt)                           //选择波特率:
                                                    //2: 使用 Timer2 做波特率
                                                    //其它值: 使用 Timer1 做波特率.
{
    /********** 波特率使用定时器 2 *****************/
    if(brt == 2)
    {
        AUXR |= 0x01;                               //S1 BRT Use Timer2;
        SetTimer2Baudraye(65536UL - (MAIN_Fosc / 4) / Baudrate1);
    }

    /********** 波特率使用定时器 1 *****************/
    else
    {
        TR1 = 0;
        AUXR &= ~0x01;                              //S1 BRT Use Timer1;
        AUXR |=  (1<<6);                            //Timer1 set as 1T mode
        TMOD &= ~(1<<6);                            //Timer1 set As Timer
        TMOD &= ~0x30;                              //Timer1_16bitAutoReload;
        TH1 = (u8)((65536UL - (MAIN_Fosc / 4) / Baudrate1) / 256);
        TL1 = (u8)((65536UL - (MAIN_Fosc / 4) / Baudrate1) % 256);
        ET1 = 0;                                    //禁止中断
        INTCLKO &= ~0x02;                           //不输出时钟
        TR1  = 1;
    }
    /*************************************************/

    SCON = (SCON & 0x3f) | 0x40;                    //UART1 模式,
                                                    //0x00: 同步移位输出,
                                                    //0x40: 8 位数据,可变波特率,
                                                    //0x80: 9 位数据,固定波特率,
                                                    //0xc0: 9 位数据,可变波特率
//  PS  = 1;                                        //高优先级中断
//  ES  = 1;                                        //允许中断
    REN = 1;                                        //允许接收
    P_SW1 &= 0x3f;
    P_SW1 |= 0x00;
}
```

```
void UART1_DMA_Interrupt(void) interrupt 13
{
    if (DMA_UR1T_STA & 0x01)                    //发送完成
    {
        DMA_UR1T_STA &= ~0x01;
        DMATxFlag = 1;
    }
    if (DMA_UR1T_STA & 0x04)                    //数据覆盖
    {
        DMA_UR1T_STA &= ~0x04;
    }

    if (DMA_UR1R_STA & 0x01)                    //接收完成
    {
        DMA_UR1R_STA &= ~0x01;
        DMARxFlag = 1;
    }
    if (DMA_UR1R_STA & 0x02)                    //数据丢弃
    {
        DMA_UR1R_STA &= ~0x02;
    }
}
```

//文件：ISR.ASM

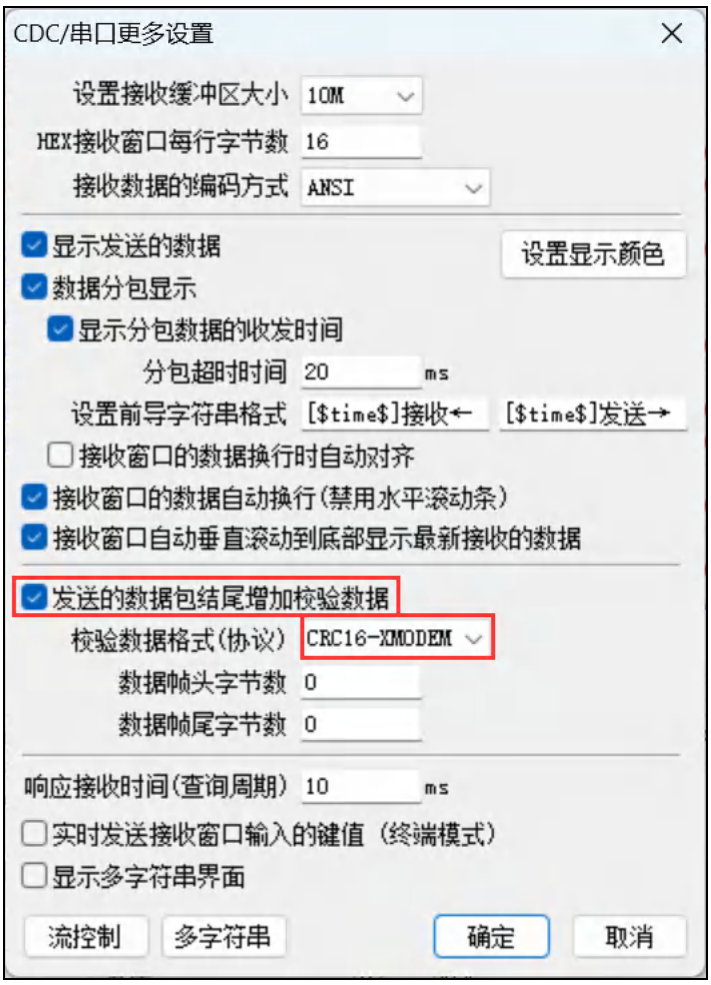
//中断号大于 31 的中断，需要进行中断入口地址重映射处理

```
            CSEG    AT      012BH               ;P0INT_VECTOR
            JMP             P0INT_ISR
            CSEG    AT      0133H               ;P1INT_VECTOR
            JMP             P1INT_ISR
            CSEG    AT      013BH               ;P2INT_VECTOR
            JMP             P2INT_ISR
            CSEG    AT      0143H               ;P3INT_VECTOR
            JMP             P3INT_ISR
            CSEG    AT      014BH               ;P4INT_VECTOR
            JMP             P4INT_ISR
            CSEG    AT      0153H               ;P5INT_VECTOR
            JMP             P5INT_ISR
            CSEG    AT      015BH               ;P6INT_VECTOR
            JMP             P6INT_ISR
            CSEG    AT      0163H               ;P7INT_VECTOR
            JMP             P7INT_ISR
            CSEG    AT      016BH               ;P8INT_VECTOR
            JMP             P8INT_ISR
            CSEG    AT      0173H               ;P9INT_VECTOR
            JMP             P9INT_ISR
            CSEG    AT      017BH               ;M2MDMA_VECTOR
            JMP             M2MDMA_ISR
            CSEG    AT      0183H               ;ADCDMA_VECTOR
            JMP             ADCDMA_ISR
            CSEG    AT      018BH               ;SPIDMA_VECTOR
            JMP             SPIDMA_ISR
            CSEG    AT      0193H               ;U1TXDMA_VECTOR
            JMP             U1TXDMA_ISR
            CSEG    AT      019BH               ;U1RXDMA_VECTOR
            JMP             U1RXDMA_ISR
            CSEG    AT      01A3H               ;U2TXDMA_VECTOR
            JMP             U2TXDMA_ISR
```

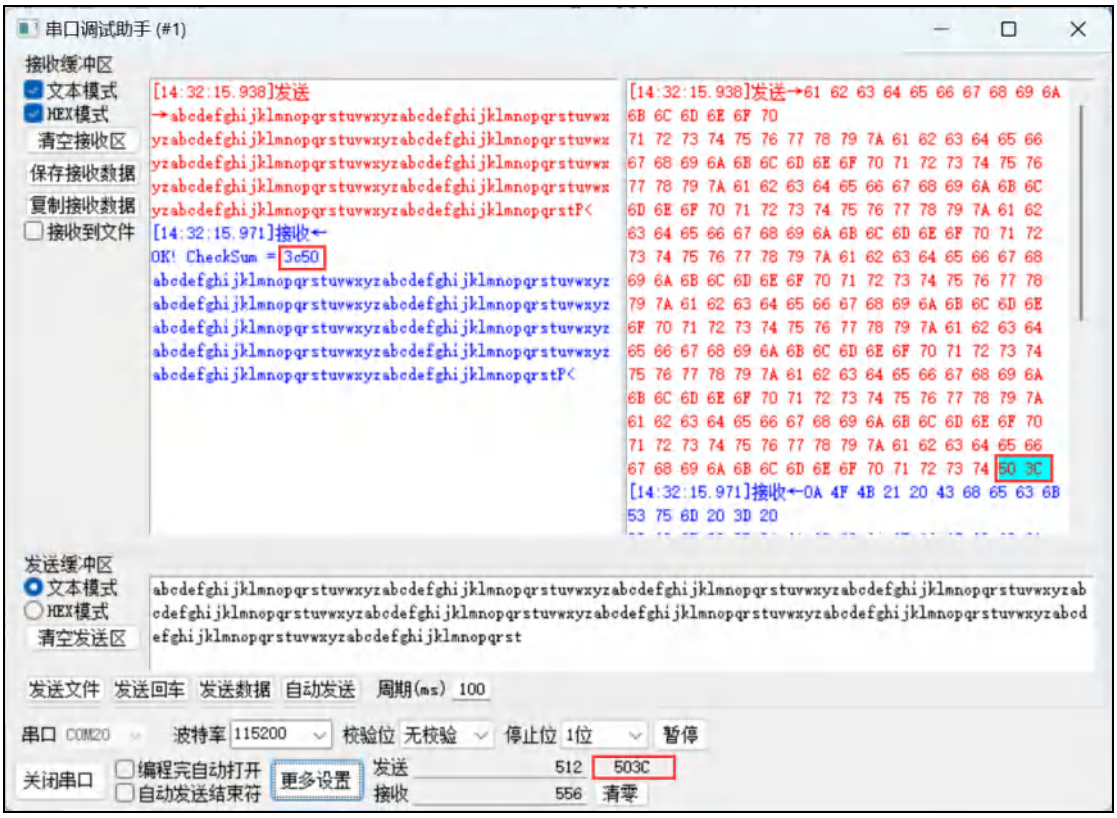
```
            CSEG    AT      01ABH           ;U2RXDMA_VECTOR
            JMP             U2RXDMA_ISR
            CSEG    AT      01B3H           ;U3TXDMA_VECTOR
            JMP             U3TXDMA_ISR
            CSEG    AT      01BBH           ;U3RXDMA_VECTOR
            JMP             U3RXDMA_ISR
            CSEG    AT      01C3H           ;U4TXDMA_VECTOR
            JMP             U4TXDMA_ISR
            CSEG    AT      01CBH           ;U4RXDMA_VECTOR
            JMP             U4RXDMA_ISR
            CSEG    AT      01D3H           ;LCMDMA_VECTOR
            JMP             LCMDMA_ISR
            CSEG    AT      01DBH           ;LCMIF_VECTOR
            JMP             LCMIF_ISR

P0INT_ISR:
P1INT_ISR:
P2INT_ISR:
P3INT_ISR:
P4INT_ISR:
P5INT_ISR:
P6INT_ISR:
P7INT_ISR:
P8INT_ISR:
P9INT_ISR:
M2MDMA_ISR:
ADCDMA_ISR:
SPIDMA_ISR:
U1TXDMA_ISR:
U1RXDMA_ISR:
U2TXDMA_ISR:
U2RXDMA_ISR:
U3TXDMA_ISR:
U3RXDMA_ISR:
U4TXDMA_ISR:
U4RXDMA_ISR:
LCMDMA_ISR:
LCMIF_ISR:

            JMP             006BH

            END
```

代码测试方法

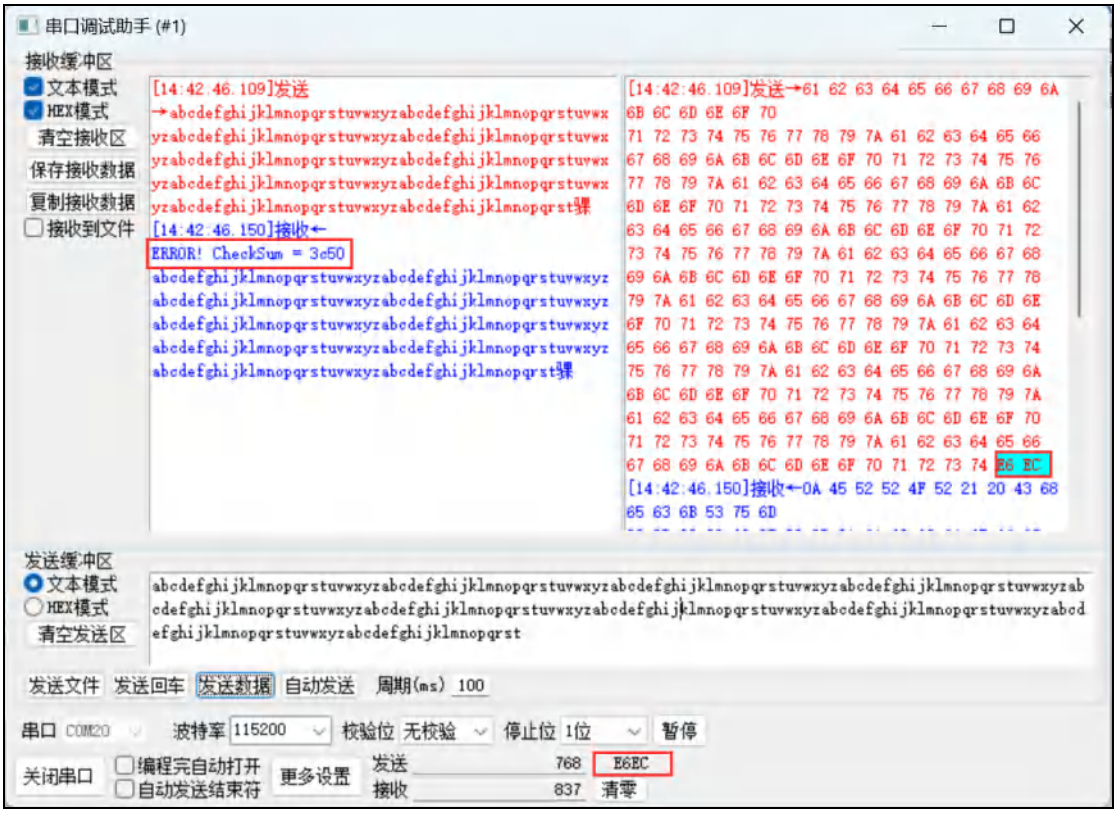
使用 ISP 软件的串口助手进行功能测试，在串口助手“更多设置”里面开启发送的数据包结尾增加校验数据功能，并选择校验数据格式（协议）为：CRC16-XMODEM



根据预定义的 DMA 数据包长度（例如：256 字节），通过串口工具发送一包数据（254 字节），串口助手自动在最后加上 2 个字节的 CRC16 校验码：



MCU 收到整包数据（256 字节）之后对前面 254 字节数据进行 CRC16 校验，得出来的校验码与最后两个字节进行比较，如果数值相等，打印“OK!”以及计算出来的校验码，然后输出 DMA 空间收取的内容。



如果校验码数值不相等，打印“ERROR!”以及计算出来的校验码。

34.15.3　使用 SPI_DMA+TFT 彩屏 DMA 双缓冲对 TFT 刷屏

```
//测试工作频率为35MHz

/**************  功能说明  ***************
使用 SPI 的 DMA 方式对外挂的串行 FLASH 进行读取数据，并将数据存储在 XDATA 的缓冲区中，然后使用 LCM 的 DMA 方式将该缓冲区的数据写入到 TFT 彩屏。
整个过程采样双缓冲 Ping-Pang 模式：
1、SPI_DMA 从 FLASH 读取数据到缓冲区 1
2、上一步的 SPI_DMA 完成后，启动 TFT 彩屏 DMA 将缓冲区 1 的数据送彩屏，同时 SPI_DMA 从 FLASH 读取数据到缓冲区 2
3、上一步的 SPI_DMA 完成后，启动 TFT 彩屏 DMA 将缓冲区 2 的数据送彩屏，同时 SPI_DMA 从 FLASH 读取数据到缓冲区
4、重复步骤 2 和步骤 3
本测试代码在实验箱 9.4B 上测试通过。使用 DMA 中断加双缓冲可极大提高 CPU 效率，实测数据如下：
```

DMA 缓冲区大小	中断周期	中断代码执行时间	DMA 中断处理程序的 CPU 占有率
3K	7.5ms	1.6us	0.021%
2K	5.0ms	1.6us	0.032%

1K	2.49ms	1.6us	0.064%
512	1.25ms	1.6us	0.128%
256	610us	1.6us	0.262%
128	310us	1.6us	0.516%

```
*****************************************/

#include “Ai8051U.H”                                  //头文件见下载软件
#include "stdio.h"

#define   FOSC              35000000UL              //系统工作频率
#define   BAUD              (65536 - (FOSC/115200+2)/4)
                                                    //加 2 操作是为了让 Keil 编译器
                                                    //自动实现四舍五入运算
#define   T2S               (65536 - FOSC*2/12/128)

#define   SCREENCX          320                     //TFT 彩屏的宽度(横向像素)
#define   SCREENCY          240                     //TFT 彩屏的高度(纵向像素)

#define   IMG1_ADDR         0x00000000              //第 1 幅图片在 FLASH 中的起始地址
#define   IMG2_ADDR         0x00025800              //第 2 幅图片在 FLASH 中的起始地址
#define   IMG3_ADDR         0x0004b000              //第 3 幅图片在 FLASH 中的起始地址

#define   HIBYTE(w)         (BYTE)(((WORD)(w)) >> 8)
#define   LOBYTE(w)         (BYTE)(w)

#define   RGB565(r, g, b)   ((((r) & 0x1f) << 11) |(((g) & 0x3f) << 5) | ((b) & 0x1f))

#define   WHITE             RGB565(31, 63, 31)      //白色
#define   BLACK             RGB565(0, 0, 0)         //黑色
#define   GRAY              RGB565(16, 32, 16)      //灰色
#define   RED               RGB565(31, 0, 0)        //红色
#define   GREEN             RGB565(0, 63, 0)        //绿色
#define   BLUE              RGB565(0, 0, 31)        //蓝色
#define   CYAN              RGB565(0, 63, 31)       //青色
#define   MAGENTA           RGB565(31, 0, 31)       //紫色
#define   YELLOW            RGB565(31, 63, 0)       //黄色

#define   DMA_BUFSIZE       (3*1024)                //DMA 缓冲区大小
                                                    //320*240 的彩屏一屏的图片数据为 150KB
                                                    //DMA 缓冲区设置为 3K 比较好

typedef   bit               BOOL;
typedef   unsigned char     BYTE;
typedef   unsigned int      WORD;
typedef   unsigned long     DWORD;

sbit SPI_SS         =   P2^2;
sbit SPI_MOSI       =   P2^3;
sbit SPI_MISO       =   P2^4;
sbit SPI_SCLK       =   P2^5;

sbit LCM_CS         =   P3^4;
sbit LCM_RST        =   P4^3;
sbit LCM_RS         =   P4^5;
sbit LCM_RD         =   P4^4;
sbit LCM_WR         =   P4^2;
#define LCM_DB          P6

#define   LCM_WRDB(d)       LCM_WR = 0;\
```

```
                                LCM_DB = (d);\
                                _nop_();\
                                LCM_WR = 1

void delay_ms(WORD n);
void delay_us(WORD n);
void sys_init();
void lcm_init();
void lcm_write_cmd(BYTE  cmd);
void lcm_write_dat(BYTE   dat);
void lcm_write_dat2(WORD dat);
void lcm_set_window(int x0, int x1, int  y0, int y1);
void lcm_clear_screen();
void lcm_show_next_image();

BYTE xdata buffer1[DMA_BUFSIZE];                    //定义缓冲区
BYTE xdata buffer2[DMA_BUFSIZE];                    //注意:如果需要使用 DMA 发送数据,
                                                    //则缓冲区必须定义在 xdata 区域内
BYTE image;
BYTE stage;

BOOL f2s;                                           //2 秒标志位

void main()
{
    sys_init();                                     //系统初始化
    lcm_init();

    f2s   = 1;
    while (1)
    {
        if (f2s)
        {
            f2s   = 0;
            lcm_show_next_image();                  //每 2 秒自动显示下一幅图片
        }
    }
}

void tm0_isr() interrupt 1
{
    f2s   = 1;                                      //设置 2 秒标志
}

void common_isr(void) interrupt  13
{
    if (DMA_LCM_STA & 0x01)
    {
        DMA_LCM_STA = 0;
        if (stage >= 2L*SCREENCY*SCREENCX/DMA_BUFSIZE)
        {
            LCM_CS = 1;
        }
    }

    if (DMA_SPI_STA & 0x01)
    {
        DMA_SPI_STA = 0;
```

```
            if (!(stage & 1))
            {
                DMA_LCM_TXAL = (BYTE)&buffer1;          //设置 DMA 缓冲区起始地址
                DMA_LCM_TXAH = (WORD)&buffer1 >> 8;
                DMA_LCM_CR = 0xa1;                      //启动 DMA 开始发送数据
            }
            else
            {
                DMA_LCM_TXAL = (BYTE)&buffer2;          //设置 DMA 缓冲区起始地址
                DMA_LCM_TXAH = (WORD)&buffer2 >> 8;
                DMA_LCM_CR = 0xa1;                      //启动 DMA 开始发送数据
            }

            if (stage < 2L*SCREENCY*SCREENCX/DMA_BUFSIZE)
            {
                if (!(stage & 1))
                {
                    DMA_SPI_RXAL = (BYTE)&buffer2;      //设置 DMA 缓冲区起始地址
                    DMA_SPI_RXAH = (WORD)&buffer2 >> 8;
                    DMA_SPI_CR = 0xc1;                  //启动 DMA 开始接收数据
                }
                else
                {
                    DMA_SPI_RXAL = (BYTE)&buffer1;      //设置 DMA 缓冲区起始地址
                    DMA_SPI_RXAH = (WORD)&buffer1 >> 8;
                    DMA_SPI_CR = 0xc1;                  //启动 DMA 开始接收数据
                }
                stage++;
            }
            else
            {
                SPI_SS = 1;
            }
        }
}

void delay_ms(WORD n)
{
    while (n--)
        delay_us(1000);
}

void delay_us(WORD n)
{
    while (n--)                                         //每个循环 24 个时钟
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();

        _nop_();
        _nop_();
```

```
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
        }
}

void sys_init()
{
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00; P0M1 = 0x00;
    P1M0 = 0x00; P1M1 = 0x00;
    P2M0 = 0x00; P2M1 = 0x00;
    P3M0 = 0x10; P3M1 = 0x00;
    P4M0 = 0x3c; P4M1 = 0x00;
    P5M0 = 0x00; P5M1 = 0x00;
    P6M0 = 0xff; P6M1 = 0x00;
    P7M0 = 0x00; P7M1 = 0x00;

    TM0PS = 127;                                //设置定时器 0 时钟预分频系统为 128(127+1)
    TMOD = 0x00;
    T0x12 = 0;
    TL0 = T2S;                                  //设置定时器 0 的定时周期为 2 秒
    TH0 = T2S >> 8;
    TR0 = 1;
    ET0 = 1;
    EA = 1;

    SPI_S0 = 1;                                 //P2.2(SS_2)/P2.3(MOSI_2)/P2.4(MISO_2)/P2.5(SCLK_2)
    SPI_S1 = 0;
    SPI_SS = 1;
    SPCTL = 0xd0;                               //初始化 SPI 模块(主模式,CPHA=CPOL=0)
    SPIF = 1;

    f3s = 0;
    image = 0;
}

void lcm_init()
{
    static BYTE code INIT[]     =               //TFT 彩屏初始化命令
    {
        4,    0xcf,     0x00, 0xc1, 0x30,
        5,    0xed,     0x64, 0x03, 0x12, 0x81,
        4,    0xe8,     0x85, 0x10, 0x7a,
        6,    0xcb,     0x39, 0x2c, 0x00, 0x34, 0x02,
        2,    0xf7,     0x20,
        3,    0xea,     0x00, 0x00,
        2,    0xc0,     0x1b,
```

```
        2,    0xc1,    0x01,
        3,    0xc5,    0x30, 0x30,
        2,    0xc7,    0xb7,
        2,    0x36,    0x28,
        2,    0x3a,    0x55,
        3,    0xb1,    0x00, 0x1a,
        3,    0xb6,    0x0a, 0xa2,
        2,    0xf2,    0x00,
        2,    0x26,    0x01,
        16,   0xe0,    0x0f, 0x2a, 0x28, 0x08, 0x0e, 0x08, 0x54,
              0xa9,    0x43, 0x0a, 0x0f, 0x00, 0x00, 0x00, 0x00,
        16,   0xe1,    0x00, 0x15, 0x17, 0x07, 0x11, 0x06, 0x2b,
              0x56,    0x3c, 0x05, 0x10, 0x0f, 0x3f, 0x3f, 0x0f,
        5,    0x2b,    0x00, 0x00, HIBYTE(SCREENCY-1), LOBYTE(SCREENCY-1),
        5,    0x2a,    0x00, 0x00, HIBYTE(SCREENCX-1), LOBYTE(SCREENCX-1),
        1,    0x11,
        0
    };
    BYTE i, j;

    LCM_DB = 0xff;
    LCM_RS = 1;
    LCM_CS = 1;
    LCM_WR = 1;
    LCM_RD = 1;
    LCM_RST = 0;                                    //复位 TFT 彩屏
    delay_ms(50);
    LCM_RST = 1;
    delay_ms(50);

    i = 0;
    while (INIT[i])
    {
        j = INIT[i++] - 1;
        lcm_write_cmd(INIT[i++]);
        while (j--)
        {
            lcm_write_dat(INIT[i++]);
        }
    }

    lcm_clear_screen();
    lcm_write_cmd(0x29);                            //打开显示
}

void lcm_write_cmd(BYTE cmd)                        //写命令到彩屏
{
    LCM_RS = 0;
    LCM_CS = 0;
    LCM_WRDB(cmd);
    LCM_CS = 1;
}

void lcm_write_dat(BYTE dat)                        //写 8 位数据到彩屏
{
    LCM_RS = 1;
    LCM_CS = 0;
    LCM_WRDB(dat);
    LCM_CS = 1;
```

```
}

void lcm_write_dat2(WORD dat)                          //写 16 位数据到彩屏
{
    LCM_RS = 1;
    LCM_CS = 0;
    LCM_WRDB(dat >> 8);
    LCM_WRDB(dat);
    LCM_CS = 1;
}

BYTE spi_shift(BYTE dat)                               //使用 SPI 读写 FLASH 数据
{
    SPIF = 1;
    SPDAT = dat;
    while (!SPIF);

    return SPDAT;
}

void lcm_set_window(int x0, int x1, int y0, int y1)    //在 TFT 彩屏中定义窗口大小
{
    lcm_write_cmd(0x2a);
    lcm_write_dat2(x0);
    lcm_write_dat2(x1);
    lcm_write_cmd(0x2b);
    lcm_write_dat2(y0);
    lcm_write_dat2(y1);
}

void lcm_clear_screen()                                //清屏(使用白色填充全屏)
{
    int i, j;

    lcm_set_window(0, SCREENCX - 1, 0, SCREENCY - 1);

    lcm_write_cmd(0x2c);                               //设置写显示数据命令
    LCM_RS = 1;
    LCM_CS = 0;
    for (i=SCREENCY; i; i--)
    {
        for (j=SCREENCX; j; j--)
        {
            LCM_WRDB(WHITE >> 8);
            LCM_WRDB(WHITE);
        }
    }
    LCM_CS = 1;
}

void lcm_show_next_image()
{
    DWORD addr;

    switch (image++)                                   //获取图片在 Flash 中的起始地址
    {
    default:    image = 1;
    case 0:     addr = IMG1_ADDR; break;
    case 1:     addr = IMG2_ADDR; break;
```

```
    case 2:     addr = IMG3_ADDR; break;
    }

    SPI_SS = 0;

    spi_shift(0x03);                                    //发送读取 FLASH 数据命令
    spi_shift((BYTE)(addr >> 16));                      //设置目标地址
    spi_shift((BYTE)(addr >> 8));
    spi_shift((BYTE)(addr));

    lcm_set_window(0, SCREENCX - 1, 0, SCREENCY - 1);

    lcm_write_cmd(0x2c);                                //设置写显示数据命令
    LCM_RS = 1;
    LCM_CS = 0;

    LCMIFCFG = 0x04;                                    //设置 LCM 接口为 8 位数据位,i8080 接口,数据口为 P6
    LCMIFCFG2 = 0x09;                                   //配置 LCM 时序
    LCMIFSTA = 0x00;                                    //清除 LCM 状态
    LCMIFCR = 0x80;                                     //使能 LCM 接口
    DMA_LCM_CFG = 0x80;                                 //使能 LCM 发送 DMA 功能,使能 DMA 中断
    DMA_LCM_STA = 0x00;                                 //清除 DMA 状态
    DMA_LCM_AMT = (DMA_BUFSIZE - 1);                    //设置 DMA 传输数据长度
    DMA_LCM_AMTH = (DMA_BUFSIZE - 1) >> 8;
    DMA_SPI_CFG = 0xa0;                                 //使能 SPI 接收 DMA 功能,使能 DMA 中断
    DMA_SPI_STA = 0x00;                                 //清除 DMA 状态
    DMA_SPI_AMT = (DMA_BUFSIZE - 1);                    //设置 DMA 传输数据长度
    DMA_SPI_AMTH = (DMA_BUFSIZE - 1) >> 8;
    DMA_SPI_CFG2 = 0x00;                                //DMA 传输过程中不控制 SS 脚
    DMA_SPI_RXAL = (BYTE)&buffer1;                      //设置 DMA 缓冲区起始地址
    DMA_SPI_RXAH = (WORD)&buffer1 >> 8;
    DMA_SPI_CR = 0xc1;                                  //启动 DMA 开始接收数据

    stage = 0;
}
```

//文件：ISR.ASM

//中断号大于 31 的中断，需要进行中断入口地址重映射处理

```
            CSEG    AT      018BH                   ;SPIDMA_VECTOR
            JMP             SPIDMA_ISR
            CSEG    AT      01D3H                   ;LCMDMA_VECTOR
            JMP             LCMDMA_ISR

SPIDMA_ISR:
LCMDMA_ISR:
            JMP             006BH

            END
```

34.15.4　实战硬件 SPI+DMA 驱动 WS2812 彩灯

使用 SPI-MOSI 输出直接驱动 WS2812 三基色彩灯, DMA 传输，不占用 CPU 时间。
本例驱动 32 个灯，接成环状，每个灯 3 个字节，分别对应绿、红、蓝，MSB 先发。
800KHz 码率，数据 0(1/4 占空比): H=0.3125us，L=0.9375us；数据 1(3/4 占空比): H=0.9375us，L=0.3125us，RESET>=50us。
高电平时间要精确控制在要求的范围内，低电平时间不需要精确控制，大于要求的最小值并小于 RES 的 50us 即可。

WS2812S 的标准时序如下：
TH+TL = 1.25us±150ns, RES>50us
T0H = 0.25us±150ns = 0.10us - 0.40us
T0L = 1.00us±150ns = 0.85us - 1.15us
T1H = 1.00us±150ns = 0.85us - 1.15us
T1L = 0.25us±150ns = 0.10us - 0.40us
两个位数据之间的间隔要小于 RES 的 50us。

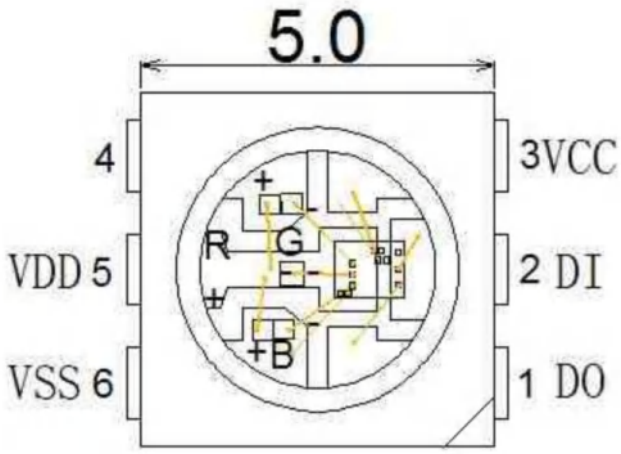
SPI 方案：
本例使用 SPI 模式的 P1.5-MOSI 输出驱动信号直接驱动 WS2812。
由于使用 SPI 主机模式，则 P1.6、P1.7 被 SPI 占用，不能做他用，但可以将 P1.7-SCLK 设置为高阻作为输入使用。
MISO-P1.6 设置为高阻，并允许下拉电阻，这样 SPI 发送完成后 MOSI 是低电平。
DMA 发送时间：@25.6MHz，DMA 发送一个字节耗时 67T(2.6172us)，一个灯要发送 12 个字节耗时 31.5us，32K 字节 xdata 最多一次驱动 2730 个灯。

用 SPI 传输，速度 3.0~3.5MHz，以 3.2MHz 为最佳，MSB 先发，每个字节高 4 位和低 4 位分别对应一个位数据，1000 为数据 0，1110 为数据 1。

SPI 数据位	D7 D6 D5 D4	D3 D2 D1 D0
SPI 数据	1 0 0 0	1 1 1 0
	WS2812 数据 0	WS2812 数据 1

SPI 数据高半字节对应的 WS2812 数据 0-->0x80，数据 1-->0xe0，
SPI 数据低半字节对应的 WS2812 数据 0-->0x08，数据 1-->0x0e，
主频 25.6MHz，SPI 分频 8 = 3.2MHz 最佳。

WS2812 介绍：



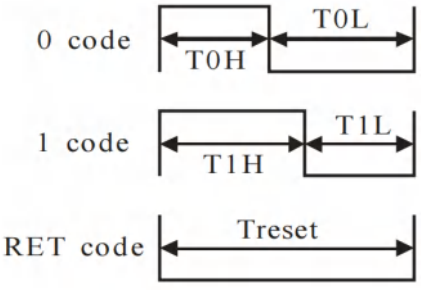
脚位描述：

编号	名称	功能描述
1	DOOUT	控制数据信号输出
2	DIN	控制数据信号输入
3	VCC	控制电路电源
4	NC	空脚
5	VDD	LED 电源
6	VSS	地线

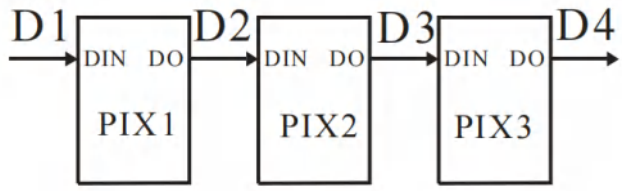
数据传输：

T0H	0 码，高电平时间	0.35us	±150ns
T1H	1 码，高电平时间	0.7us	±150ns
T0L	0 码，低电平时间	0.8us	±150ns
T1L	1 码，低电平时间	0.6us	±150ns
RES	低电平时间	50us 以上	

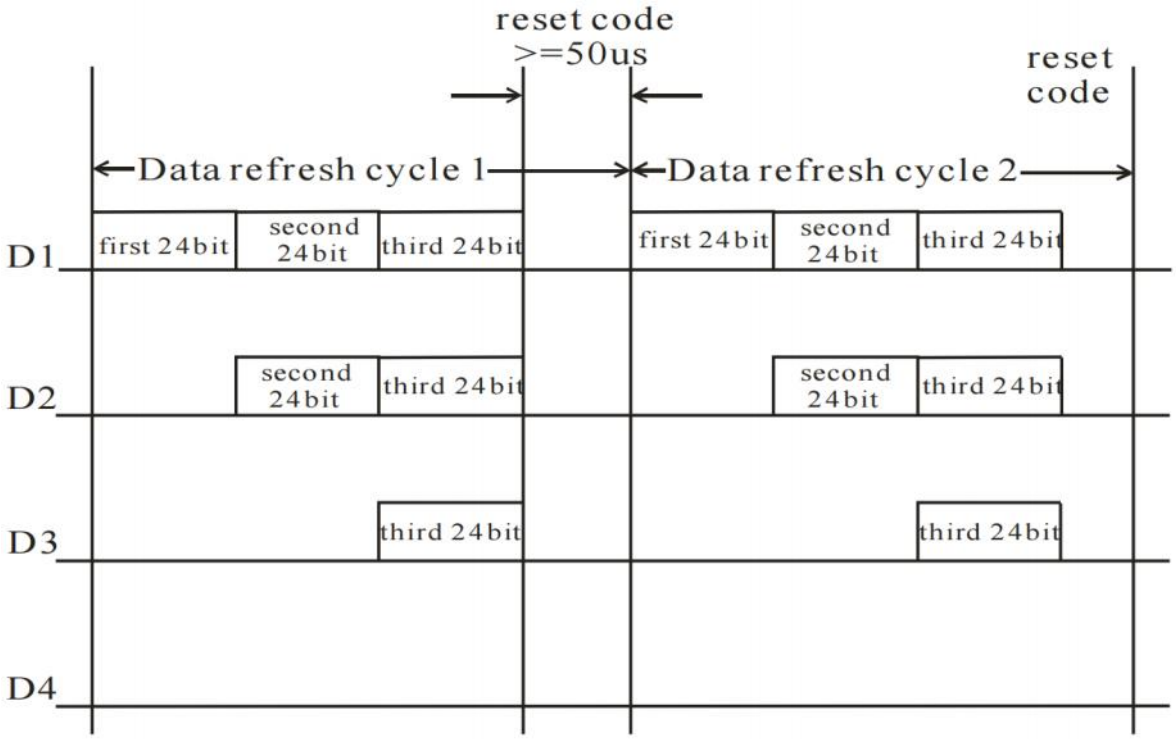
序列图：



连接方式：



数据传输方式：



注：D1 的数据由 MCU 发送，D2、D3、D4 通过像素内部重塑放大后进行传输。

像素点在上电复位以后，DIN 端接受从控制器传输过来的数据，首先送过来的 24bit 数据被第一个像素点提取后，送到像素点内部的数据锁存器，剩余的数据经过内部整形处理电路整形放大后通过 DO 端口转发输出给下一个级联的像素点，每经过一个像素点的传输，信号减少 24bit。像素点采用自动整形转发技术，使得该像素点的级联个数不受信号传送的限制，仅受限信号传输速度要求。

24 位数据的组成：

G7	G6	G5	G4	G3	G2	G1	G0	R7	R6	R5	R4	R3	R2	R1	R0	B7	B6	B5	B4	B3	B2	B1	B0

注：按照 GRB 发送数据的顺序，先发送高位。

范例程序：

```
//测试工作频率为 25.6MHz

//main.c
//程序主函数

#include "config.h"                    //头文件中已包含 ai8051u.h 以及其他头文件

void main()
{
    SYS_Init();

    while (1)
    {
        if (f50ms)                     //判断 50ms 标志位
        {
            f50ms = 0;                 //清除 50ms 标志，并处理 50ms 事件
```

```
            DMA_Update();                   //更新 DMA 缓冲区数据并发送给 WS2812
        }
    }
}

void SYS_Init(void)                         //系统初始化函数
{
    EAXFR = 1;                              //使能访问扩展 XFR
    WTST = 0x00;                            //设置最快速度访问程序代码
    CKCON = 0x00;                           //设置最快速度访问内部 XDATA

    P0M0 = 0x00; P0M1 = 0x00;               //初始化 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x40;               //初始化 P1 口为准双向口模式，设置 P1.6 高阻输入
    P2M0 = 0x00; P2M1 = 0x00;               //初始化 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x00;               //初始化 P3 口为准双向口模式
    P4M0 = 0x00; P4M1 = 0x00;               //初始化 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;               //初始化 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;               //初始化 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;               //初始化 P7 口为准双向口模式

    P1PU |= 0xb0;                           //使能 P1.4,P1.5,P1.7 内部上拉
    P1PD |= 0x40;                           //使能 P1.6 内部下拉，这样 SPI 发送完成后 MOSI 是低电平
    P1SR &= ~0xa0;                          //设置 P1.5,P1.7 快速模式

    SPI_Init();                             //SPI 接口初始化
    Timer0_Init();                          //定时器 0 初始化,用于产生 50ms 的时间基准

    EA = 1;                                 //使能全局中断
}
```

35 I2S 音频总线，可以使用 PLL 高速时钟作为时钟源

产品线	I2S
Ai8051U 系列	●

Ai8051U 系列单片机内部集成 I2S 音频总线功能单元。

35.1 I2S 功能脚切换

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW3	BBH	I2S_S		S2SPI_S[1:0]		S1SPI_S[1:0]		CAN2_S[1:0]	

I2S_S[1:0]：I2S 功能脚选择位

I2S_S[1:0]	I2SMCLK	I2SBCK	I2SLRCK	I2SData
	主时钟	位时钟	左右声道时钟	音频数据
00	P3.3	P3.2	P3.5	P3.4
01	P1.6	P1.7	P1.4	P1.5
10	P2.2	P2.3	P2.0	P2.1
11	P1.6	P4.3	P4.0	P4.1

I2SMCLK， I2S 主时钟
I2SBCK， I2S 位时钟
I2SLRCK， I2S 左右声道时钟
I2SData， I2S 音频数据

35.2 I2S 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
I2SCR	I2S 控制寄存器	7EFD98H	TXEIE	RXNEIE	ERRIE	FRF	-	-	TXDMAEN	RXDMAEN	0000,xx00
I2SSR	I2S 状态寄存器	7EFD99H	-	FRE	BUY	OVR	UDR	CHSID	TXE	RXNE	x000,0000
I2SDRH	I2S 数据寄存器高字节	7EFD9AH	DR[15:8]								0000,0000
I2SDRL	I2S 数据寄存器低字节	7EFD9BH	DR[7:0]								0000,0000
I2SPRH	I2S 分频寄存器高字节	7EFD9CH	-	-	-	-	-	-	MCKOE	ODD	xxxx,xx00
I2SPRL	I2S 分频寄存器低字节	7EFD9DH	DIV[7:0]								0000,0000
I2SCFGH	I2S 配置寄存器高字节	7EFD9EH	-	-	-	-	-	I2SE	I2SCFG[1:0]		xxxx,x000
I2SCFGL	I2S 配置寄存器低字节	7EFD9FH	PCMSYNC	-	STD[1:0]		CKPOL	DATLEN[1:0]		CHLEN	0x00,0000
I2SMD	I2S 从模式控制寄存器	7EFDA0H	MODE[7:0]								0000,0000
I2SMCKDIV	I2S 主时钟分配器	7EFDA1H	DIV[7:0]								0000,0000

35.2.1 I2S 控制寄存器（I2SCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2SCR	7EFD98H	TXEIE	RXNEIE	ERRIE	FRF	-	-	TXDMAEN	RXDMAEN

TXEIE：输发送缓冲区空中断允许位。

0：禁止发缓冲区空中断

1：允许发送缓冲区空中断

RXNEIE：接收缓冲区非空中断允许位。

0：禁止接收缓冲区非空中断

1：允许接收缓冲区非空中断

ERRIE：错误中断允许位。

0：禁止错误中断

1：允许错误中断

FRF：帧格式

0：Motorola 格式

1：TI 格式

TXDMAEN：发送DMA控制

0：禁止发送 DMA

1：使能发送 DMA

RXDMAEN：接收DMA控制

0：禁止接收 DMA

1：使能接收 DMA

35.2.2 I2S 状态寄存器（I2SSR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2SSR	7EFD99H	-	FRE	BUY	OVR	UDR	CHSID	TXE	RXNE

FRE：帧格式错误

BUY：I2S忙状态。（正在通讯中或者发送缓冲区非空）

OVR：发生上溢

UDR：发生下溢

CHSID：通道选择标志

0：左通道正在发送或接收数据

1：右通道正在发送或接收数据

TXE：发送缓冲区空标志位

RXNE：接收缓冲区非空标志位

35.2.3 I2S 数据寄存器（I2SDRH、I2SDRL）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2SDRH	7EFD9AH	DR[15:8]							
I2SDRL	7EFD9BH	DR[7:0]							

DR[15:0]：接收或发送的数据。写数据寄存器时应先写高字节I2SDRH，后写I2SDRL。读数据寄存器时应先读高字节I2SDRH，后读I2SDRL。

35.2.4 I2S 分频寄存器高字节（I2SPRH）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2SPRH	7EFD9CH	-	-	-	-	-	-	MCKOE	ODD

MCKOE：I2S主时钟输出控制

0：禁止 I2S 主时钟输出

1：使能 I2S 主时钟输出

ODD：预分频器的奇数因子控制

0：实际分频值＝DIV*2

1：实际分频值＝DIV*2+1

35.2.5 I2S 分频寄存器低字节（I2SPRL）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2SPRL	7EFD9DH	DIV[7:0]							

DIV[7:0]：I2S的时钟预分频器。DIV不可设置为0或者1。

I2S 比特率＝每个通道的比特数×通道数×音频采样频率(FS)

对于 16 位音频、左声道和右声道，I2S 比特率计算如下：I2S 比特率＝16×2×FS

对于 32 位宽度的数据包长度，I2S 比特率＝32×2×FS。

当 I2S 主时钟输出使能时（MCKOE 设置为 1）：

当信道帧宽度为 16 位时，音频采样频率 FS＝I2S 时钟÷[(16×2)×(2×DIV+ODD)×8)]

当信道帧宽度为 32 位时，音频采样频率 FS＝I2S 时钟÷[(32×2)×(2×DIV+ODD)×4)]

当 I2S 主时钟输出被禁用时（MCKOE 设置为 0）：

当信道帧宽度为 16 位时，音频采样频率 FS＝I2S 时钟÷[(16×2)×(2×DIV+ODD)]

当信道帧宽度为 32 位时，音频采样频率 FS＝I2S 时钟÷[(32×2)×(2×DIV+ODD)]

上述 I2S 时钟为系统时钟或 PLL 时钟

35.2.6 I2S 配置寄存器高字节（I2SCFGH）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2SCFGH	7EFD9EH	-	-	-	-	-	I2SE	I2SCFG[1:0]	

I2SE：I2S模块控制

0：禁止 I2S 功能

1：使能 I2S 功能

I2SCFG[1:0]：I2S模式配置

00：从机发送模式

01：从机接收模式

10：主机发送模式

11：主机接收模式

35.2.7 I2S 分频寄存器低字节（I2SCFGL）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2SCFGL	7EFD9FH	PCMSYNC	-	STD[1:0]		CKPOL	DATLEN[1:0]		CHLEN

PCMSYNC：PCM帧同步

0：短帧同步

1：长帧同步

STD[1:0]：I2S标准选择

00：I2S 飞利浦标准

01：MSB 左对齐标准

10：LSB 右对齐标准

11：PCM 标准

CKPOL：稳态时钟极性

0：I2S 时钟稳定状态为低电平

1：I2S 时钟稳定状态为高电平

DATLEN[1:0]：数据长度

00：16 位

01：24 位

10：32 位

11：保留

CHLEN：通道长度（每个音频通道的位数）

0：16 位

1：32 位

35.2.8 I2S 从模式控制寄存器（I2SMD）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2SMD	7EFDA0H	MODE[7:0]							

内部测试用，若用户需要使用 I2S 的从机模式，则需要将此寄存器设置为 FFH

35.2.9 I2S 主时钟分频寄存器（I2SMCKDIV）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2SMCKDIV	7EFDA1H								

I2SMCKDIV：I2S 主时钟分频系数。

I2SMCKDIV	I2S 主时钟频率
0	高速 I/O 时钟源/1
1	高速 I/O 时钟源/1
2	高速 I/O 时钟源/2
3	高速 I/O 时钟源/3
…	…
255	高速 I/O 时钟源/255

35.3　范例程序

35.3.1　输出 2 路三角波

```
//测试工作频率为 45.1584MHz

#include “Ai8051U.H”                                    //头文件见下载软件
#include "intrins.h"                                    //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

//#define   FOSC        24576000UL                      //定义主时钟
//#define   SampleRate  8000                            //定义采样率

//#define   FOSC        32768000UL                      //定义主时钟
//#define   SampleRate  32000                           //定义采样率

//#define   FOSC        33868800UL                      //定义主时钟
#define     FOSC        45158400UL                      //定义主时钟
#define     SampleRate  44100                           //定义采样率

#define     MCKOE       1                               //I2S 主时钟输出控制
                                                        //0:禁止 I2S 主时钟输出
                                                        //1:允许 I2S 主时钟输出

#define     I2SEN       0x04                            //I2S 模块使能
                                                        //0x00:禁止
                                                        //0x04:允许

#define     I2S_MODE    2                               //I2S 模式
                                                        //0:从机发送模式
                                                        //1:从机接收模式
                                                        //2:主机发送模式
                                                        //3:主机接收模式

#define     PCMSYNC     0                               //PCM 帧同步
                                                        //0:短帧同步
                                                        //1:长帧同步

#define     STD_MODE    1                               //I2S 标准选择
                                                        //0: I2S 飞利浦标准
                                                        //1: MSB 左对齐标准
                                                        //2:LSB 右对齐标准
                                                        //3:PCM 标准

#define     CKPOL       0                               //I2S 稳态时钟极性
                                                        //0:时钟稳定状态为低电平
                                                        //1:时钟稳定状态为高电平

#define     DATLEN      0                               //数据长度
                                                        //0:16 位
                                                        //1:24 位
                                                        //2:32 位
                                                        //3:保留

#define     CHLEN       0                               //通道长度(每个音频通道的位数)
                                                        //0:16 位
                                                        //1: 32 位
```

```
#if (MCKOE == 1)                                        //允许主时钟输出
    #define     I2SDIV     FOSC/(16*2*8*SampleRate)     //对于双声道允许主时钟输出 16bit 256fs
                                                        //则 2*DIV + ODD = I2S 时钟 / 256fs
                                                        //必要需要 DIV >= 2, 即分频系数 >=4.
#endif
#if (MCKOE == 0)                                        //禁止主时钟输出
    #define     I2SDIV     FOSC/(16*2*SampleRate)       //对于双声道禁止主时钟输出 16bit
                                                        //则 2*DIV + ODD = I2S 时钟 / 32fs
                                                        //必要需要 DIV >= 2, 即分频系数 >=4.
#endif

Typedef unsigned char u8;

u8 dac_index;                                           //输出计数索引
bit B_rise;

void main(void)
{
    EAXFR = 1;                                          //允许访问扩展的特殊寄存器，XFR
                                                        //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                      //允许访问扩展的特殊寄存器，XFR
                                                        //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                           //设置取程序代码等待时间，
                                                        //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                          //设置访问片内的 xdata 速度，
                                                        //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00; P0M1 = 0x00;
    P1M0 = 0x00; P1M1 = 0x00;
    P2M0 = 0x00; P2M1 = 0x00;
    P3M0 = 0x00; P3M1 = 0x00;
    P4M0 = 0x00; P4M1 = 0x00;
    P5M0 = 0x00; P5M1 = 0x00;

    I2SMD = 0xff;                                       //内部保留字节,需设置为 FFH
    I2SCR = 0x80;                                       //使能发送缓冲区空中断(0x80)
    I2SPRH = (MCKOE << 1) + (I2SDIV & 1);               //设置 I2S 主时钟输出(I2SMCK), 设置 ODD
    I2SPRL = I2SDIV/2;                                  //设置 I2S 时钟分频
    I2SCFGH = I2S_MODE;                                 //设置 I2S 模式为主机发送模式
    I2SCFGL = (PCMSYNC << 7) + (STD_MODE << 4) + (CKPOL    << 3) + (DATLEN << 1) + CHLEN;
    P_SW3 = (P_SW3 & 0x3f) | (0<<6);                    //I2S 端口切换
                                                        //0: P3.2(BCLK) P3.4(SD) P5.4(MCLK) P3.5(WS)
                                                        //1: P1.5(BCLK) P1.3(SD) P1.6(MCLK) P5.4(WS)
                                                        //2: P2.5(BCLK) P2.3(SD) P5.4(MCLK) P2.2(WS)
                                                        //3: P4.3(BCLK) P4.0(SD) P1.6(MCLK) P5.4(WS)
    I2SCFGH |= I2SEN;                                   //使能 I2S 模块

    EA = 1;

    dac_index = 0;
    B_rise = 1;

    while (1);
}

void I2S_ISR(void) interrupt 62                         //输出 2 个三角波
{
    u8 i;
```

```
    if (I2SSR & 0x02)                       //发送缓冲区空
    {
        if (I2SSR & 0x04)                   //右声道
        {
            i = dac_index + 0x80;           //单极性转成双极性(无符号转有符号)
            I2SDRH = i;                     //发送下一帧音频数据
            I2SDRL = 0;
            if(B_rise)
            {
                if(++dac_index == 255)
                    B_rise = 0;
            }
            else
            {
                if(--dac_index == 0)
                    B_rise = 1;
            }
        }
        else                                //左声道
        {
            i = dac_index + 0x80;           //单极性转成双极性(无符号转有符号)
            I2SDRH = i;                     //发送下一帧音频数据
            I2SDRL = 0;
        }
    }
}
```

35.3.2　输出 2 路正弦波

```
//测试工作频率为 45.1584MHz

#include “Ai8051U.H”                                  //头文件见下载软件
#include "intrins.h"                                  //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

//#define   FOSC        24576000UL                    //定义主时钟
//#define   SampleRate  8000                          //定义采样率

//#define   FOSC        32768000UL                    //定义主时钟
//#define   SampleRate  32000                         //定义采样率

//#define   FOSC        33868800UL                    //定义主时钟
#define   FOSC        45158400UL                    //定义主时钟
#define   SampleRate  44100                         //定义采样率

#define   MCKOE       1                             //I2S 主时钟输出控制
                                                    //0:禁止 I2S 主时钟输出
                                                    //1:允许 I2S 主时钟输出

#define   I2SEN       0x04                          //I2S 模块使能
                                                    //0x00:禁止
                                                    //0x04:允许

#define   I2S_MODE    2                             //I2S 模式
                                                    //0:从机发送模式
                                                    //1:从机接收模式
                                                    //2:主机发送模式
                                                    //3:主机接收模式

#define   PCMSYNC     0                             //PCM 帧同步
                                                    //0:短帧同步
                                                    //1:长帧同步

#define   STD_MODE    1                             //I2S 标准选择
                                                    //0: I2S 飞利浦标准
                                                    //1: MSB 左对齐标准
                                                    //2:LSB 右对齐标准
                                                    //3:PCM 标准

#define   CKPOL       0                             //I2S 稳态时钟极性
                                                    //0:时钟稳定状态为低电平
                                                    //1:时钟稳定状态为高电平

#define   DATLEN      0                             //数据长度
                                                    //0:16 位
                                                    //1:24 位
                                                    //2:32 位
                                                    //3:保留

#define   CHLEN       0                             //通道长度(每个音频通道的位数)
                                                    //0:16 位
                                                    //1: 32 位

#if (MCKOE == 1)                                    //允许主时钟输出
```

```
    #define     I2SDIV     FOSC/(16*2*8*SampleRate)          //对于双声道允许主时钟输出 16bit 256fs
                                                             //则 2*DIV + ODD = I2S 时钟 / 256fs
                                                             //必要需要 DIV >= 2, 即分频系数 >=4.
#endif
#if (MCKOE == 0)                                             //禁止主时钟输出
    #define     I2SDIV     FOSC/(16*2*SampleRate)            //对于双声道禁止主时钟输出 16bit
                                                             //则 2*DIV + ODD = I2S 时钟 / 32fs
                                                             //必要需要 DIV >= 2, 即分频系数 >=4.
#endif

typedef unsigned char u8;
typedef unsigned int u16;

u16 code T_SINE_L[32];
u16 code T_SINE_R[32];

u8 dac_index;                                                //输出计数索引

void main(void)
{
    EAXFR = 1;                                               //允许访问扩展的特殊寄存器，XFR
                                                             //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                           //允许访问扩展的特殊寄存器，XFR
                                                             //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                                //设置取程序代码等待时间，
                                                             //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                               //设置访问片内的 xdata 速度，
                                                             //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M0 = 0x00; P0M1 = 0x00;
    P1M0 = 0x00; P1M1 = 0x00;
    P2M0 = 0x00; P2M1 = 0x00;
    P3M0 = 0x00; P3M1 = 0x00;
    P4M0 = 0x00; P4M1 = 0x00;
    P5M0 = 0x00; P5M1 = 0x00;

    I2SMD = 0xff;                                            //内部保留字节,需设置为 FFH
    I2SCR = 0x80;                                            //使能发送缓冲区空中断(0x80)
    I2SPRH = (MCKOE << 1) + (I2SDIV & 1);                    //设置 I2S 主时钟输出(I2SMCK), 设置 ODD
    I2SPRL = I2SDIV/2;                                       //设置 I2S 时钟分频
    I2SCFGH = I2S_MODE;                                      //设置 I2S 模式为主机发送模式
    I2SCFGL = (PCMSYNC << 7) + (STD_MODE << 4) + (CKPOL << 3) + (DATLEN << 1) + CHLEN;
    P_SW3 = (P_SW3 & 0x3f) | (0<<6);                         //I2S 端口切换
                                                             //0: P3.2(BCLK) P3.4(SD) P5.4(MCLK) P3.5(WS)
                                                             //1: P1.5(BCLK) P1.3(SD) P1.6(MCLK) P5.4(WS)
                                                             //2: P2.5(BCLK) P2.3(SD) P5.4(MCLK) P2.2(WS)
                                                             //3: P4.3(BCLK) P4.0(SD) P1.6(MCLK) P5.4(WS)
    I2SCFGH |= I2SEN;                                        //使能 I2S 模块

    dac_index = 0;
    EA = 1;

    while (1);
}

void I2S_ISR(void) interrupt 62                              //输出 2 个正弦波
{
    u16 j;
```

```
    if (I2SSR & 0x02)                           //发送缓冲区空
    {
        if (I2SSR & 0x04)                       //右声道
        {
            j = T_SINE_R[dac_index] + 32768;    //单极性转成双极性(无符号转有符号)
            I2SDRH = (u8)(j /256);              //发送下一帧音频数据
            I2SDRL = (u8)(j %256) & 0xfe;
            dac_index++;
            dac_index &= 31;
        }
        Else                                    //左声道
        {
            j = T_SINE_L[dac_index] + 32768;    //单极性转成双极性(无符号转有符号)
            I2SDRH = (u8)(j /256);              //发送下一帧音频数据
            I2SDRL = (u8)(j %256) & 0xfe;
        }
    }
}

u16 code T_SINE_L[]=                            //4 点一个正弦波
{
    32768, 64768, 32768, 768,
    32768, 64768, 32768, 768,
    32768, 64768, 32768, 768,
    32768, 64768, 32768, 768,
    32768, 64768, 32768, 768,
    32768, 64768, 32768, 768,
    32768, 64768, 32768, 768,
    32768, 64768, 32768, 768,
};

u16 code T_SINE_R[]=                            //16 点一个正弦波
{
    32768, 45014, 55395, 62332, 64768, 62332, 55395, 45014,
    32768, 20522, 10141, 3204,  768,     3204,    10141, 20522,
    32768, 45014, 55395, 62332, 64768, 62332, 55395, 45014,
    32768, 20522, 10141, 3204,  768,     3204,    10141, 20522,
};
```

36 32 位硬件乘除单元（MDU32）

产品线	MDU32
Ai8051U 系列	●

乘法和除法单元（称为 MDU32）提供快速的 32 位算术运算。MDU32 支持无符号和补码有符号整数操作数。MDU32 由专用的直接内存访问控制模块（称为 DMA）。所有 MDU32 算术操作都是通过向 DMA 控件写入 DMA 指令来启动的寄存器 DMAIR。MDU32 模块执行的所有算术运算的操作数和结果位于寄存器 R0-R7。

注意：

1、DMA 模块执行算术运算所需的执行时间，包括：

- 操作数从 DR0-DR4 寄存器加载到 MDU32 模块
- MDU32 算术运算
- 从 MDU32 模块到 R0-R7 寄存器的结果存储

2、处理器执行 C 编译算术函数所需的执行时间，包括：

- DMA 指令写入 DMAIR 寄存器
- 操作数从 DR0-DR4 寄存器加载到 MDU32 模块
- MDU32 算术运算
- 从 MDU32 模块到 R0-R7 寄存器的结果存储
- 从函数返回（RET 指令）

3、MDU32 执行乘除法运算时，单片机会自动切换到 IDLE 模式，即 CPU 停止时钟指令，其它外设仍继续工作。运算完成后，单片机自动切换到正常工作模式

36.1 相关的特殊功能寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
DMAIR	DMA 指令寄存器	EDH	DMAIR[7:0]								0000,0000

注：向 DMAIR 寄存器写入指令码，只能使用立即数寻址方式的指令“MOV DMAIR,#N”，使用其它指令会无法正常触发计算。

36.2 运算执行时间表

MDU 运算	DMA 指令码		执行时钟数
	HEX	DEC	
32 位乘法	0x02	2	3
32 位无符号除法	0x04	4	19
32 位有符号除法	0x06	6	21

36.3 MDU32 算术运算

36.3.1 32 位乘法

32 位乘法运算是对两个无符号或有符号的补码整数参数执行的。第一个参数位于 R4-R7 寄存器中，第二个参数位于 R0-R3 寄存器中。运算结果存储到 R4-R7 寄存器。

结果				=	被乘数				×	乘数			
MSB			LSB		MSB			LSB		MSB			LSB
R4	R5	R6	R7	=	R4	R5	R6	R7	×	R0	R1	R2	R3

DMA 指令码：　0x02

执行时间：　3clk

36.3.2 32 位无符号除法

对两个无符号整数参数执行 32 位无符号除法运算。第一个参数“被除数”是位于 R4-R7 寄存器中，第二个参数“除数”位于 R0-R3 寄存器中。结果存储到 R4-R7 寄存器。余数在 R0-R3 中返回。除以零返回 0xFFFFFFFF。

结果				=	被除数				÷	除数			
MSB			LSB		MSB			LSB		MSB			LSB
R4	R5	R6	R7	=	R4	R5	R6	R7	÷	R0	R1	R2	R3

余数				=	被除数				%	除数			
MSB			LSB		MSB			LSB		MSB			LSB
R0	R1	R2	R3	=	R4	R5	R6	R7	%	R0	R1	R2	R3

DMA 指令码：　0x04

执行时间：　19clk

36.3.3 32 位有符号除法

对两个有符号的补码参数执行 32 位有符号除法运算。第一个参数“被除数”位于 R4-R7 寄存器中，第二个参数“除数”位于 R0-R3 寄存器中。结果存储到 R4-R7 寄存器。余数在 R0-R3 中返回。除以零返回 0xFFFFFFFF。

结果				=	被除数				÷	除数			
MSB			LSB		MSB			LSB		MSB			LSB
R4	R5	R6	R7	=	R4	R5	R6	R7	÷	R0	R1	R2	R3

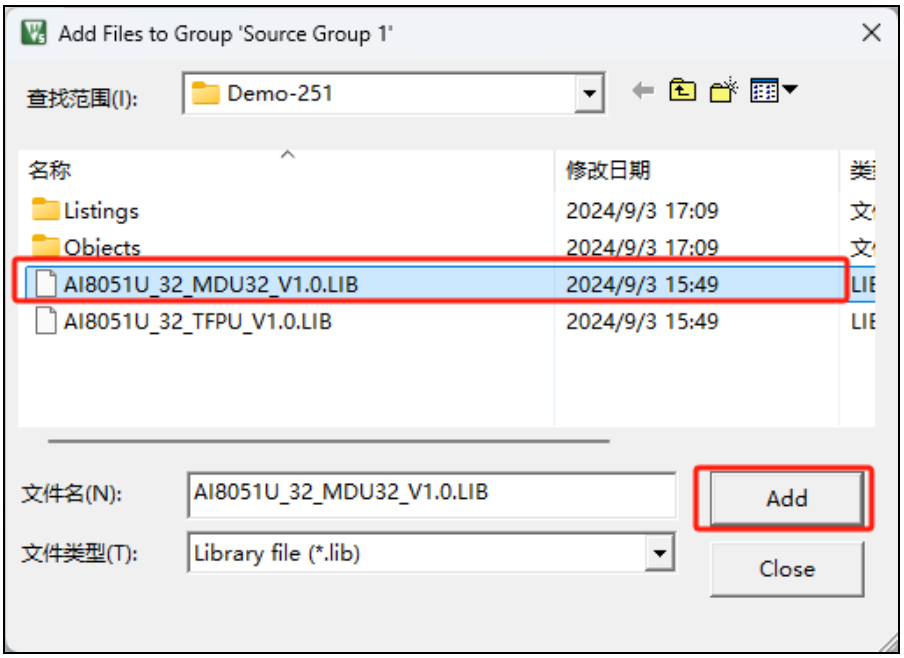
余数				=	被除数				%	除数			
MSB			LSB		MSB			LSB		MSB			LSB
R0	R1	R2	R3	=	R4	R5	R6	R7	%	R0	R1	R2	R3

DMA 指令码：　0x06

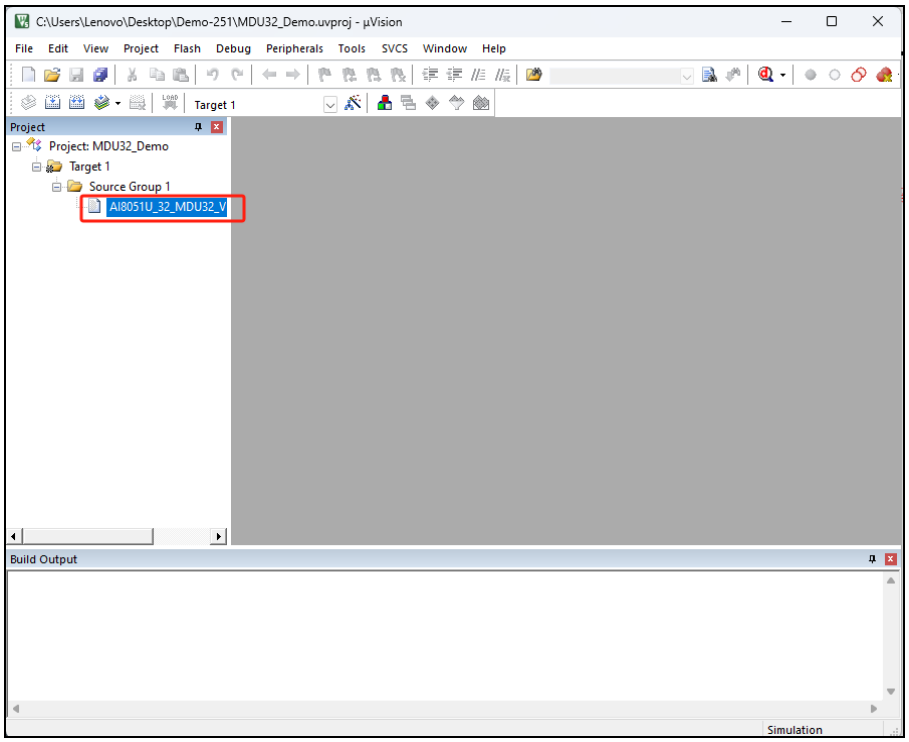
执行时间：　21clk

36.4　范例程序

当要使用 32 位硬件乘除单元时，只要在 keil 项目中加入库文件“Ai8051U_32_MDU32_V1.0.LIB”即可。



添加库文件到项目：



```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                  //头文件见下载软件
#include "intrins.h"                                  //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

volatile unsigned long int near uint1, uint2, xuint;\
volatile long int sint1, sint2, xsint;

void main(void)
{
    EAXFR = 1;                                        //允许访问扩展的特殊寄存器，XFR
                                                      //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                    //允许访问扩展的特殊寄存器，XFR
                                                      //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                         //设置取程序代码等待时间，
                                                      //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                        //设置访问片内的 xdata 速度，
                                                      //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M1 = 0;   P0M0 = 0;                             //设置为准双向口
    P1M1 = 0;   P1M0 = 0;                             //设置为准双向口
    P2M1 = 0;   P2M0 = 0;                             //设置为准双向口
    P3M1 = 0;   P3M0 = 0;                             //设置为准双向口
    P4M1 = 0;   P4M0 = 0;                             //设置为准双向口
    P5M1 = 0;   P5M0 = 0;                             //设置为准双向口
    P6M1 = 0;   P6M0 = 0;                             //设置为准双向口
    P7M1 = 0;   P7M0 = 0;                             //设置为准双向口

    P10 = 0;
    sint1 = 0x31030F05;
    sint2 = 0x00401350;
    xsint = sint1 * sint2;

    uint1 =   5;
    uint2 =   50;
    xuint = uint1 * uint2;

    uint1 = 528745;
    uint2 = 654689;
    xuint = uint1 / uint2;

    sint1 = 2000000000;
    sint2 = 2134135177;
    xsint = sint1 / sint2;

    sint1 = -2000000000;
    sint2 = -2134135177;
    xsint = sint1 / sint2;

    sint1 = -2000000000;
    sint2 = 2134135177;
    xsint = sint1 / sint2;
    P10 = 1;

    while(1);
}
```

37 TFPU（三角函数+单精度浮点运算器），可以使用 PLL 高速时钟作为时钟源

产品线	TFPU
Ai8051U 系列	●

37.1 TFPU 浮点运算器简介

单精度浮点运算器（TFPU）提供了快速的单精度浮点算术运算。TFPU 支持单精度浮点数的加、减、乘、除、开方、比较和三角函数（正弦、余弦、正切和反正切）。同时支持整数类型和单精度浮点数之间的转换。输入的浮点数字格式符合 IEEE-754 标准。TFPU 由专用直接内存访问 DMA 控制。所有算术运算都是通过将运算指令写入称为 DMAIR 控制寄存器来启动的。TFPU 模块执行的所有算术运算的操作数（或其指针）和结果（或其指针）位于当前组的寄存器 R0-R7 中。

37.2 相关的特殊功能寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
DMAIR	DMA 指令寄存器	EDH	DMAIR[7:0]								0000,0000

注：向 DMAIR 寄存器写入指令码，只能使用立即数寻址方式的指令“MOV DMAIR,#N”，使用其它指令会无法正常触发计算。

37.3 运算执行时间表

TFPU 运算	DMA 指令码		执行时钟数
	HEX	DEC	
浮点数加法	0x1C	28	31 ~ 40
浮点数减法	0x1D	29	31 ~ 40
浮点数乘法	0x1E	30	26 ~ 34
浮点数除法	0x1F	31	58 ~ 67
浮点数开方	0x20	32	50 ~ 54
浮点数比较	0x21	33	18
浮点数检测	0x22	34	15
正弦函数	0x2D	45	32 ~ 270
余弦函数	0x2E	46	32 ~ 270
正切函数	0x2F	47	58 ~ 258
反正切函数	0x30	48	62 ~ 175
浮点数转 8 位整数	0x23	35	19 ~ 30
浮点数转 16 位整数	0x24	36	19 ~ 30
浮点数转 32 位整数	0x25	37	23 ~ 39
8 位整数转浮点数	0x27	39	23 ~ 33
16 位整数转浮点数	0x28	40	23 ~ 33
32 位整数转浮点数	0x29	41	24 ~ 33
初始化协处理器	0x31	49	2
清除异常	0x32	50	4
读状态寄存器	0x33	51	4
写状态寄存器	0x34	52	4
读控制寄存器	0x35	53	4
写控制寄存器	0x36	54	4
TFPU 选择系统时钟	**0x3E**	**62**	-
TFPU 选择 PLL 高速时钟	**0x3F**	**63**	-

注：当使用 0x3E 和 0x3F 命令选择 TFPU 的时钟源后，再向 DMAIR 寄存器中写入其他运算指令不会改变 TFPU 的时钟源

37.4 TFPU 基本算术运算

本小节描述了 TFPU 模块使用 DMA 控制器可以执行的所有算术运算。所有操作数必须位于数据内存中。操作结果也存储在由 PSW（0xD0）位选择的 R0-R7 当前组的数据存储器空间中。

37.4.1 浮点数加法（+）

对两个浮点数进行加法运算。加数 BR 位于 R0~R3 寄存器中，被加数 AR 位于 R4~R7 寄存器中，计算结果和保存到 R4~R7 寄存器

和				=	被加数 AR				+	加数 BR			
MSB			LSB		MSB			LSB		MSB			LSB
R4	R5	R6	R7	=	R4	R5	R6	R7	+	R0	R1	R2	R3

指令码　0x1C(28)

执行时间（时钟数）　31 ~ 40

37.4.2 浮点数减法（－）

对两个浮点数进行减法运算。减数 BR 位于 R0~R3 寄存器中，被减数 AR 位于 R4~R7 寄存器中，计算结果差保存到 R4~R7 寄存器

差				=	被减数 AR				－	减数 BR			
MSB			LSB		MSB			LSB		MSB			LSB
R4	R5	R6	R7	=	R4	R5	R6	R7	－	R0	R1	R2	R3

指令码　0x1D(29)

执行时间（时钟数）　31 ~ 40

37.4.3 浮点数乘法（×）

对两个浮点数进行乘运算。乘数 BR 位于 R0~R3 寄存器中，被乘数 AR 位于 R4~R7 寄存器中，计算结果积保存到 R4~R7 寄存器

积				=	被乘数 AR				×	乘数 BR			
MSB			LSB		MSB			LSB		MSB			LSB
R4	R5	R6	R7	=	R4	R5	R6	R7	×	R0	R1	R2	R3

指令码　0x1E(30)

执行时间（时钟数）　26 ~ 34、

37.4.4 浮点数除法（÷）

对两个浮点数进行除运算。除数 BR 位于 R0~R3 寄存器中，被除数 AR 位于 R4~R7 寄存器中，计算结果商保存到 R4~R7 寄存器

商				=	被除数 AR				÷	除数 BR			
MSB			LSB		MSB			LSB		MSB			LSB
R4	R5	R6	R7	=	R4	R5	R6	R7	÷	R0	R1	R2	R3

指令码 0x1F(31)

执行时间（时钟数） 58 ~ 67

37.4.5 浮点数开方/平方根（sqrt）

对 1 个浮点数进行开方运算。被开方数 AR 位于 R4~R7 寄存器中，计算结果平方根保存到 R4~R7 寄存器

平方根				= sqrt	浮点数 AR			
MSB			LSB		MSB			LSB
R4	R5	R6	R7	= sqrt	R4	R5	R6	R7

指令码 0x20(32)

执行时间（时钟数） 50 - 54

37.4.6 浮点数比较（comp）

对两个浮点数进行算术比较运算。比较数 BR 位于 R0~R3 寄存器中，被比较数 AR 位于 R4~R7 寄存器中，比较结果保存到 R7 寄存器

R7.3	R7.2	R7.1	R7.0	比较结果	结果描述
0	0	0	0	0x0001	AR > BR
1	0	0	0	0x0000	AR = BR
0	0	0	1	0xFFFF	AR < BR
1	1	0	1	unchanged	Unordered

比较结果				=	被比较数 AR				?	比较数 BR			
			LSB		MSB			LSB		MSB			LSB
—	—	—	R7	=	R0	R1	R2	R3	?	R4	R5	R6	R7

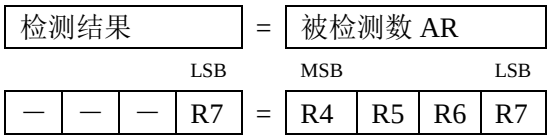
指令码 0x21(33)

执行时间（时钟数） 18

37.4.7 浮点数检测（check）

对 1 个浮点数进行检测。被检测数 AR 位于 R4~R7 寄存器中，检测结果保存到 R7 寄存器

R7[7:4]	R7.3	R7.2	R7.1	R7.0	结果描述
0000	0	0	0	0	正非浮点数（+NaN）
0000	0	0	1	1	负非浮点数（-NaN）
0000	0	1	0	0	正规范浮点数
0000	0	1	1	0	负规范浮点数
0000	0	1	0	1	正无穷（+INF）
0000	0	1	1	1	负无穷（-INF）
0000	1	0	0	0	正零
0000	1	0	1	0	负零
0000	1	1	0	0	正非规范浮点数
0000	1	1	1	0	负非规范浮点数



指令码 0x22(34)

执行时间（时钟数） 15

37.5 TFPU 三角函数

注：所有三角函数的角度参数类型均为弧度。弧度与角度的转换公式：

$$角度 = \frac{180^\circ}{\pi} \times 弧度 \qquad 弧度 = \frac{\pi}{180^\circ} \times 角度$$

37.5.1 正弦函数（sin）

求一个单精度弧度浮点数的正弦值。弧度数 AR 位于 R4~R7 寄存器中，计算结果保存到 R4~R7 中

结果				= sin	弧度 AR			
MSB			LSB		MSB			LSB
R4	R5	R6	R7	= sin	R4	R5	R6	R7

指令码　0x2D(45)

执行时间（时钟数）　32 - 270 clk

37.5.2 余弦函数（cos）

求一个单精度弧度浮点数的余弦值。弧度数 AR 位于 R4~R7 寄存器中，计算结果保存到 R4~R7 中

结果				= cos	弧度 AR			
MSB			LSB		MSB			LSB
R4	R5	R6	R7	= cos	R4	R5	R6	R7

指令码　0x2E(46)

执行时间（时钟数）　32 - 270

37.5.3 正切函数（tan）

求一个单精度弧度浮点数的正切值。弧度数 AR 位于 R4~R7 寄存器中，计算结果保存到 R4~R7 中

结果				= tan	弧度 AR			
MSB			LSB		MSB			LSB
R4	R5	R6	R7	= tan	R4	R5	R6	R7

指令码　0x2F(47)

执行时间（时钟数）　58 - 258

37.5.4 反正切函数（arctan）

求一个单精度弧度浮点数的反正切值。弧度数 AR 位于 R4~R7 寄存器中，计算结果保存到 R4~R7 中

result				= arctan	弧度 AR			
MSB			LSB		MSB			LSB
R4	R5	R6	R7	= arctan	R4	R5	R6	R7

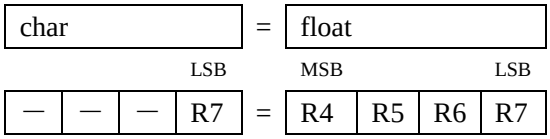
指令码　0x30(48)

执行时间（时钟数）　62 - 175

37.6 TFPU 数据转换操作

37.6.1 浮点数转 8 位整数（float → char）

将浮点数转换为 8 位整数（字符型 char）。浮点数位于 R4~R7 寄存器中，8 位整数保存到 R7 中

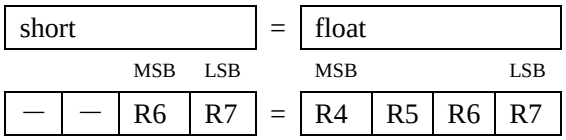


指令码　0x23(35)

执行时间（时钟数）　19 - 30

37.6.2 浮点数转 16 位整数（float → short）

将浮点数转换为 16 位整数（短整型 short）。浮点数位于 R4~R7 寄存器中，16 位整数保存到 R6~R7 中

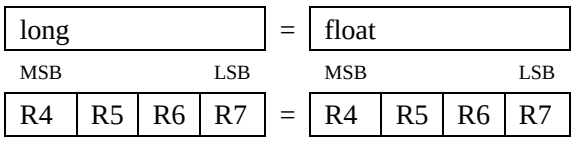


指令码　0x24(36)

执行时间（时钟数）　19 - 30

37.6.3 浮点数转 32 位整数（float → long）

将浮点数转换为 32 位整数（长整型 long）。浮点数 AR 位于 R4~R7 寄存器中，32 位整数保存到 R4~R7 中

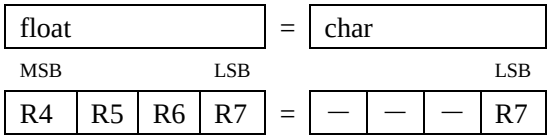


指令码　0x25(37)

执行时间（时钟数）　23 - 39

37.6.4 8 位整数转浮点数（char → float）

将 8 位整数（字符型 char）转换为浮点数。8 位整数位于 R7 寄存器中，浮点数保存到 R4~R7 中



指令码　0x27(39)

执行时间（时钟数）　23 - 33

37.6.5 16 位整数转浮点数（short → float）

将 16 位整数（短整型 short）转换为浮点数。16 位整数位于 R6~R7 寄存器中，浮点数保存到 R4~R7 中

float				=	short			
MSB			LSB				MSB	LSB
R4	R5	R6	R7	=	—	—	R6	R7

指令码 0x28(40)

执行时间（时钟数） 23 - 33

37.6.6 32 位整数转浮点数（long → float）

将 32 位整数（长整型 long）转换为浮点数。32 位整数位于 R4~R7 寄存器中，浮点数保存到 R4~R7 中

float				=	long			
MSB			LSB		MSB			LSB
R4	R5	R6	R7	=	R4	R5	R6	R7

指令码 0x29(41)

执行时间（时钟数） 24 - 33

37.7 TFPU 协处理器控制操作

37.7.1 初始化协处理器

初始化 TFPU 协处理器，初始化完成后会产生一个异常状态，需要软件清除。

指令码 0x31(49)

执行时间（时钟数） 2

37.7.2 清除异常

清除所有的异常状态

指令码 0x32(50)

执行时间（时钟数） 4

37.7.3 读状态寄存器

读取协处理器的状态寄存器。结果保存到 R7 中

指令码 0x33(51)

执行时间（时钟数） 4

37.7.4 写状态寄存器

写协处理器的状态寄存器。待写入的值位于 R7 寄存器中，完成后新的状态寄存器值保存到 R7 中

指令码 0x34(52)

执行时间（时钟数） 4

37.7.5 读控制寄存器

读取协处理器的控制寄存器。结果保存到 R7 中

指令码 0x35(53)

执行时间（时钟数） 4

37.7.6 写控制寄存器

写协处理器的控制寄存器。待写入的值位于 R7 寄存器中，完成后新的控制寄存器值保存到 R7 中

指令码 0x36(54)

执行时间（时钟数） 4

37.8 如何选择 TFPU 的时钟源

Ai8051U 系列的 TFPU 可选择和 CPU 时钟同频，也可选择和 CPU 时钟异步且频率更高的 PLL 时钟源作为 TFPU 的运算时钟

37.8.1 选择系统时钟（和 CPU 时钟同步）作为 TFPU 时钟源

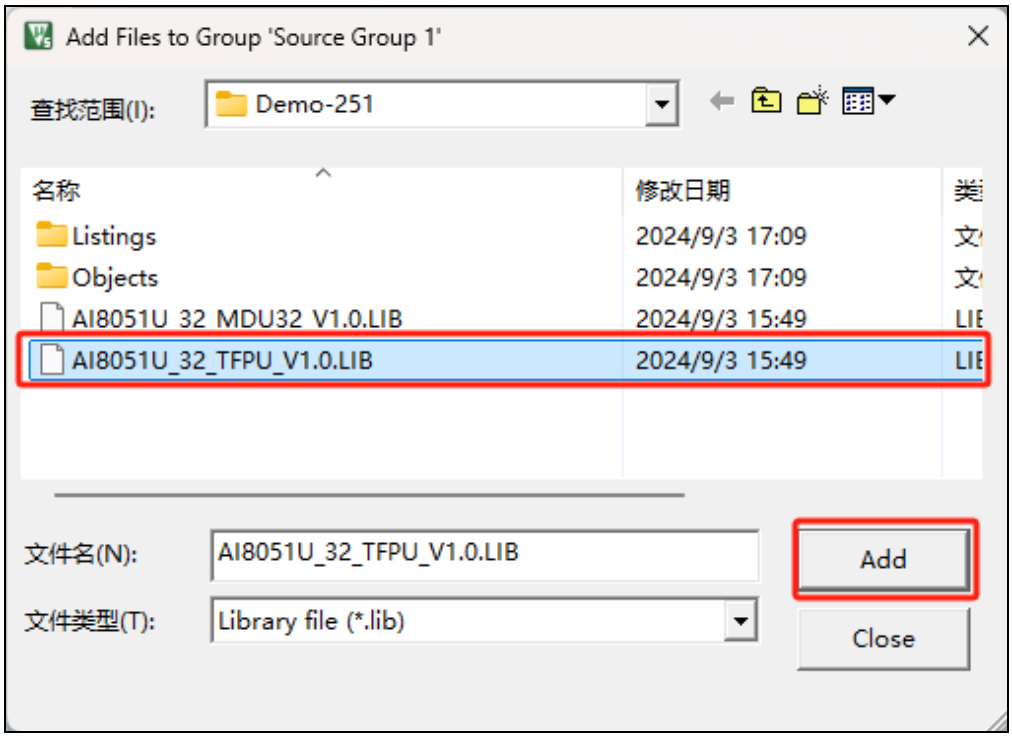
```
DMAIR = 0x3E;                    //此语句可放在系统初始化的地方
```

37.8.2 选择 PLL 时钟（和 CPU 时钟异步）作为 TFPU 时钟源

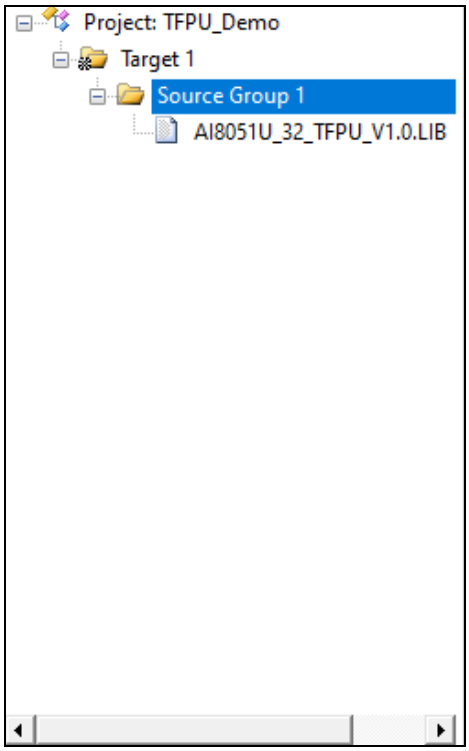
```
DMAIR = 0x3F;                    //此语句可放在系统初始化的地方
```

37.9　范例程序

当要使用硬件浮点时，只要在 keil 项目中加入
库文件“Ai8051U_32_TFPU_V1.0.LIB”



添加库文件到项目：



```
//测试工作频率为 52MHz

#include “Ai8051U.H”                                  //头文件见下载软件
#include "intrins.h"                                  //使用_nop_()函数所必须要包含的头文件,
//否则延时函数中调用的_nop_()函数没有被头文件引用过来，会导致编译器找不到这个而函数而报错。

#include <math.h>

float data cfl1=3.9;
float data cfl2=5.1;
float data cfl3;

void main(void)
{
    EAXFR = 1;                                        //允许访问扩展的特殊寄存器，XFR
                                                      //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                                    //允许访问扩展的特殊寄存器，XFR
                                                      //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                         //设置取程序代码等待时间，
                                                      //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                        //设置访问片内的 xdata 速度，
                                                      //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    P0M1 = 0;   P0M0 = 0;                             //设置为准双向口
    P1M1 = 0;   P1M0 = 0;                             //设置为准双向口
    P2M1 = 0;   P2M0 = 0;                             //设置为准双向口
    P3M1 = 0;   P3M0 = 0;                             //设置为准双向口
    P4M1 = 0;   P4M0 = 0;                             //设置为准双向口
    P5M1 = 0;   P5M0 = 0;                             //设置为准双向口
    P6M1 = 0;   P6M0 = 0;                             //设置为准双向口
    P7M1 = 0;   P7M0 = 0;                             //设置为准双向口

    P10 = 0;
    cfl3 = cfl1*cfl2;
    cfl3 = cfl1/cfl2-cfl3;
    cfl3 = cfl1*cfl2+cfl3;
    cfl3 = cfl1/cfl2*sin(cfl3);
    cfl3 = cfl1/cfl2*cos(cfl3);
    cfl3 = cfl1/cfl2*tan(cfl3);
    cfl3 = cfl1/cfl2*sqrt(cfl3);
    cfl3 = cfl1/cfl2*atan(cfl3);
    P10 = 1;

    while(1);
}
```

38 DPU32（DSP 指令，32 位及 64 位整数运算器）

注：DPU32 中的所有 32 位的乘法，结果一律为 64 位。（例如：64 位的乘加运算是两个 32 位整数相乘得到 64 位的积，然后和 64 位整数相加最后得到 64 位的结果）

产品线	DPU32
Ai8052U 系列	●

38.1 相关的特殊功能寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
DPUST	DPU32 状态寄存器	86H	Z	DBZ	SC[5:0]						0000,0000
DPUOP	DPU32 指令寄存器	EDH	OPCODE[7:0]								0000,0000
DPUCFG	DPU32 配置寄存器	-	-	-	-	-	-	CDRS[2:0]			xxxx,x101

DPU32 状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DPUST	86H	Z	DBZ	SC[5:0]					

Z：零状态。

0：计算结果不为 0

1：计算结果为 0

DBZ：除数为0状态

0：除数不为 0

1：除数为 0

SC[5:0]：规格化时移动的位数。

运算过程中如果产生了进位标志和溢出标志，会直接修改到PSW中的CY和OV

DPU32 指令寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DPUOP	EDH	OP[7:0]							

DPU32的运算需要软件向寄存器DPUOP写入相应的指令码进行触发

DPU32 配置寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DPUCFG	-	-	-	-	-	-	CDRS[2:0]		

CDRS[2:0]：ECX和EDX寄存器组选择（详情参见：DPU32操作寄存器说明）

DPUCFG寄存器的修改方法：先将需要修改的值写入ACC寄存器中，然后DPUOP触发寄存器中写入0x80，触发硬件自动将ACC的值写入DPUCFG寄存器中。

38.2 DPU32 操作寄存器说明

PSW[5:4] / EAX/EBX		00	01	10	11	80251
EBX	BX2	DATA[0]	DATA[8]	DATA[10]	DATA[18]	DR0
		DATA[1]	DATA[9]	DATA[11]	DATA[19]	
	BX	DATA[2]	DATA[A]	DATA[12]	DATA[1A]	
		DATA[3]	DATA[B]	DATA[13]	DATA[1B]	
EAX	AX2	DATA[4]	DATA[C]	DATA[14]	DATA[1C]	DR4
		DATA[5]	DATA[D]	DATA[15]	DATA[1D]	
	AX	DATA[6]	DATA[E]	DATA[16]	DATA[1E]	
		DATA[7]	DATA[F]	DATA[17]	DATA[1F]	

CDRS[2:0] / ECX/RDX		000	001	010	011	101	110
EDX	DX2	DATA[0]	DATA[8]	DATA[10]	DATA[18]	DR16	DR24
		DATA[1]	DATA[9]	DATA[11]	DATA[19]		
	DX	DATA[2]	DATA[A]	DATA[12]	DATA[1A]		
		DATA[3]	DATA[B]	DATA[13]	DATA[1B]		
ECX	CX2	DATA[4]	DATA[C]	DATA[14]	DATA[1C]	DR20	DR28
		DATA[5]	DATA[D]	DATA[15]	DATA[1D]		
	CX	DATA[6]	DATA[E]	DATA[16]	DATA[1E]		
		DATA[7]	DATA[F]	DATA[17]	DATA[1F]		

38.3 运算执行时间表

DPU32 运算	指令码		执行时钟数	标志位			
	HEX	DEC		CY	OV	Z	DBZ
FILL_DPUCFG	0x80	128	1	–	–	–	–
BIN2BCD	0x81	129	36	–	–	–	–
BCD2BIN	0x82	130	13	–	–	–	–
NRM_BX	0x83	131	6	–	–	–	–
NRM_EBX	0x84	132	7	–	–	–	–
NRM_EABX	0x85	133	9	–	–	–	–
SWAP_EAB	0x86	134	3	–	–	–	–
SWAP_ECD	0x87	135	3	–	–	–	–
SWAP_EAC	0x88	136	3	–	–	–	–
SWAP_EBD	0x89	137	3	–	–	–	–
ADDC_EABX	0x90	144	3	✓	✓	✓	–
ADDC_ABX	0x91	145	3	✓	✓	✓	–
ADD_EABX	0x92	146	3	✓	✓	✓	–
ADD_ABX	0x93	147	3	✓	✓	✓	–
SUBB_EABX	0x94	148	3	✓	✓	✓	–
SUBB_ABX	0x95	149	3	✓	✓	✓	–
SUB_EABX	0x96	150	3	✓	✓	✓	–
SUB_ABX	0x97	151	3	✓	✓	✓	–
CMP_EABX	0x98	152	3	✓	✓	✓	–
CMP_ABX	0x99	153	3	✓	✓	✓	–
MULU_EABX	0x9A	154	8	–	–	–	–
MULS_EABX	0x9B	155	10	–	–	–	–
MULU_ABX	0x9C	156	3	–	–	–	–
MULS_ABX	0x9D	157	4	–	–	–	–
MULX_KEIL16	0x9E	158	3	–	–	–	–
DIVU_EABX	0x9F	159	19	–	–	–	✓
DIVS_EABX	0xA0	160	21	–	–	–	✓
DIVU_ABX	0xA1	161	11	–	–	–	✓
DIVU_KEIL16	0xA2	162	10	–	–	–	✓
DIVS_ABX	0xA3	163	13	–	–	–	✓
DIVS_KEIL16	0xA4	164	12	–	–	–	✓
DIVU_EABXD	0xA5	165	36	–	–	–	✓
DIVU_EAXB	0xA6	166	19	–	–	–	✓
SET0_EAX	0xB0	176	2	–	–	–	–
SET1_EAX	0xB1	177	2	–	–	–	–
SET0_EBX	0xB2	178	2	–	–	–	–
SET1_EBX	0xB3	179	2	–	–	–	–
SET0_ECX	0xB4	180	2	–	–	–	–
SET1_ECX	0xB5	181	2	–	–	–	–

DPU32 运算	指令码		执行时钟数	标志位			
	HEX	DEC		CY	OV	Z	DBZ
SET0_EDX	0xB6	182	2	–	–	–	–
SET1_EDX	0xB7	183	2	–	–	–	–
NEGS_EAX	0xB8	184	2	–	–	–	–
NEGS_EBX	0xB9	185	2	–	–	–	–
NEGS_AX	0xBA	186	2	–	–	–	–
NEGS_BX	0xBB	187	2	–	–	–	–
INC1_EAX	0xC0	192	3	✓	–	✓	–
INC1_EBX	0xC1	193	3	✓	–	✓	–
INC4_EAX	0xC2	194	3	✓	–	✓	–
INC4_EBX	0xC3	195	3	✓	–	✓	–
INC1_AX	0xC4	196	3	✓	–	✓	–
INC1_BX	0xC5	197	3	✓	–	✓	–
INC2_AX	0xC6	198	3	✓	–	✓	–
INC2_BX	0xC7	199	3	✓	–	✓	–
DEC1_EAX	0xC8	200	3	✓	–	✓	–
DEC1_EBX	0xC9	201	3	✓	–	✓	–
DEC4_EAX	0xCA	202	3	✓	–	✓	–
DEC4_EBX	0xCB	203	3	✓	–	✓	–
DEC1_AX	0xCC	204	3	✓	–	✓	–
DEC1_BX	0xCD	205	3	✓	–	✓	–
DEC2_AX	0xCE	206	3	✓	–	✓	–
DEC2_BX	0xCF	207	3	✓	–	✓	–
BAND_EABX	0xD0	208	3	–	–	✓	–
BAND_ABX	0xD1	209	3	–	–	✓	–
BOR_EABX	0xD2	210	3	–	–	✓	–
BOR_ABX	0xD3	211	3	–	–	✓	–
BXOR_EABX	0xD4	212	3	–	–	✓	–
BXOR_ABX	0xD5	213	3	–	–	✓	–
BCPL_EAX	0xD6	214	2	–	–	✓	–
BCPL_EBX	0xD7	215	2	–	–	✓	–
SHL_EAX	0xE0	224	7	✓	–	–	–
SHL_EBX	0xE1	225	7	✓	–	–	–
SHL_AX	0xE2	226	6	✓	–	–	–
SHL_BX	0xE3	227	6	✓	–	–	–
SHRU_EAX	0xE4	228	7	✓	–	–	–
SHRU_EBX	0xE5	229	7	✓	–	–	–
SHRU_AX	0xE6	230	6	✓	–	–	–
SHRU_BX	0xE7	231	6	✓	–	–	–
SHRS_EAX	0xE8	232	7	✓	–	–	–
SHRS_EBX	0xE9	233	7	✓	–	–	–
SHRS_AX	0xEA	234	6	✓	–	–	–

DPU32 运算	指令码		执行时钟数	标志位			
	HEX	DEC		CY	OV	Z	DBZ
SHRS_BX	0xEB	235	6	✓	–	–	–
ROR_EAX	0xEC	236	7	–	–	–	–
ROR_EBX	0xED	237	7	–	–	–	–
ROR_AX	0xEE	238	6	–	–	–	–
ROR_BX	0xEF	239	6	–	–	–	–
MMD32_EABX	0xF0	240	43	–	–	–	✓
MMD16_ABX	0xF1	241	21	–	–	–	✓
LTC32_EAX	0xF2	242	44	✓	✓	–	✓
LTC16_AX	0xF3	243	22	✓	✓	–	✓
MA32_EDX	0xF4	244	5	✓	✓	–	–
MA64_ECDX	0xF5	245	12	✓	✓	–	–

38.4 DPU32 指令说明

配置 DPU（FILL_DPUCFG）

FILL_DPUCFG

指令操作: DPUCFG[7:0] = ACC[7:0]

指令说明:

影响标志位:

CY	OV	Z	DBZ
-	-	-	-

[指令码] 80H

时钟数: 1

BIN 转 BCD（BIN2BCD）

BIN2BCD

指令操作: { EAX, EBX } <= BCD(EBX)

指令说明: 将 EBX 中的 BIN 码转换为 BCD 码后写入{ EAX, EBX }

影响标志位:

CY	OV	Z	DBZ
-	-	-	-

[指令码] 81H

时钟数: 36

BCD 转 BIN（BCD2BIN）

BCD2BIN

指令操作: EAX <= BIN({EAX[6:0], EBX})

指令说明: 将{EAX[6:0], EBX}中的 BCD 码转换为 BIN 码后写入 EAX

影响标志位:

CY	OV	Z	DBZ
-	-	-	-

[指令码] 82H

时钟数: 13

16 位 AX 规格化（NRM_BX）

NRM_BX

指令操作: BX=NRM(BX); SC[5:0] <= Shifted#

指令说明: 规格化过程中移动的位数存放在 SC[5:0]中

影响标志位:

CY	OV	Z	DBZ
–	–	-	-

[指令码] 83H

时钟数: 6

32 位 EBX 规格化（NRM_EBX）

NRM_EBX

指令操作: EBX=NRM(EBX); SC[5:0] <= Shifted#

指令说明: 规格化过程中移动的位数存放在 SC[5:0]中

影响标志位:

CY	OV	Z	DBZ
–	–	-	-

[指令码] 84H

时钟数: 7

64 位｛EAX, EBX｝规格化（NRM_EABX）

NRM_EABX

指令操作: {EAX,EBX}=NRM({EAX,EBX}); SC[5:0] <= Shifted#

指令说明: 规格化过程中移动的位数存放在 SC[5:0]中

影响标志位:

CY	OV	Z	DBZ
–	–	-	-

[指令码] 85H

时钟数: 8

EAX 与 EBX 内容互换（SWAP_EAB）

SWAP_EAB

指令操作: (EAX) ↔(EBX)

指令说明:

影响标志位:

CY	OV	Z	DBZ
-	-	-	-

[指令码] 86H

时钟数: 3

ECX 与 EDX 内容互换（SWAP_ECD）

SWAP_ECD

指令操作: (ECX) ↔(EDX)

指令说明:

影响标志位:

CY	OV	Z	DBZ
-	-	-	-

[指令码] 87H

时钟数: 3

EAX 与 ECX 内容互换（SWAP_EAC）

SWAP_EAC

指令操作: (EAX) ↔(ECX)

指令说明:

影响标志位:

CY	OV	Z	DBZ
-	-	-	-

[指令码] 88H

时钟数: 3

EBX 与 EDX 内容互换（SWAP_EBD）

SWAP_EBD

指令操作: (EBX) ↔(EDX)

指令说明:

影响标志位:

CY	OV	Z	DBZ
-	-	-	-

[指令码] 89H

时钟数: 3

32 位带进位加法（ADDC_EABX）

ADDC_EABX

指令操作: EAX=EAX + EBX+CY

指令说明:

影响标志位:

CY	OV	Z	DBZ
✓	✓	✓	-

[指令码] 90H

时钟数: 3

16 位带进位加法（ADDC_ABX）

ADDC_ABX

指令操作: AX= AX+ BX + CY

指令说明:

影响标志位:

CY	OV	Z	DBZ
✓	✓	✓	-

[指令码] 91H

时钟数: 3

32 位加法（ADD_EABX）

ADD_EABX

指令操作: EAX=EAX + EBX

指令说明:

影响标志位:

CY	OV	Z	DBZ
✓	✓	✓	-

[指令码] 92H

时钟数: 3

16 位加法（ADD_ABX）

ADD_ABX

指令操作: AX=AX + BX

指令说明:

影响标志位:

CY	OV	Z	DBZ
✓	✓	✓	-

[指令码] 93H

时钟数: 3

32 位带借位减法（SUBB_EABX）

SUBB_EABX

指令操作: EAX=EAX – EBX - CY

指令说明:

影响标志位:

CY	OV	Z	DBZ
✓	✓	✓	-

[指令码] 94H

时钟数: 3

16 位带借位减法（SUBB_ABX）

SUBB_ABX

指令操作: AX=AX – BX - CY

指令说明:

影响标志位:

CY	OV	Z	DBZ
✓	✓	✓	-

[指令码] 95H

时钟数: 3

32 位减法（SUB_EABX）

SUB_EABX

指令操作: EAX=EAX – EBX

指令说明:

影响标志位:

CY	OV	Z	DBZ
✓	✓	✓	-

[指令码] 96H

时钟数: 3

16 位减法（SUB_ABX）

SUB_ABX

指令操作: AX=AX – BX

指令说明:

影响标志位:

CY	OV	Z	DBZ
✓	✓	✓	-

[指令码] 97H

时钟数: 3

32 位比较（CMP_EABX）

CMP_EABX

指令操作: EAX – EBX

指令说明:

影响标志位:

CY	OV	Z	DBZ
✓	✓	✓	-

[指令码] 98H

时钟数: 3

16 位比较（CMP_ABX）

CMP_ABX

指令操作: AX – BX

指令说明:

影响标志位:

CY	OV	Z	DBZ
✓	✓	✓	-

[指令码] 99H

时钟数: 3

32 位无符号数乘法（MULU_EABX）

MULU_EABX

指令操作: { EDX, EAX } = EAX * EBX

指令说明: 结果 64 位，高 32 位在 EDX，低 32 位在 EAX

影响标志位:

CY	OV	Z	DBZ
-	-	-	-

[指令码] 9AH

时钟数: 8

32 位有符号数乘法（MULS_EABX）

MULS_EABX

指令操作: { EDX, EAX } = EAX * EBX

指令说明: 结果 64 位，高 32 位在 EDX，低 32 位在 EAX

影响标志位:

CY	OV	Z	DBZ
-	-	-	-

[指令码] 9BH

时钟数: 10

16 位无符号数乘法（MULU_ABX）

MULU_ABX

指令操作: EAX = AX * BX

指令说明: 结果 32 位，乘积低 16 位在 AX,高 16 位在 AX2,合并为 EAX

影响标志位:

CY	OV	Z	DBZ
-	-	-	-

[指令码] 9CH

时钟数: 3

16 位有符号数乘法（MULS_ABX）

MULS_ABX

指令操作: EAX = AX * BX

指令说明: 结果 32 位，乘积低 16 位在 AX,高 16 位在 AX2,合并为 EAX

影响标志位:

CY	OV	Z	DBZ
-	-	-	-

[指令码] 9DH

时钟数: 4

兼容 KEIL 编译器的 16 位无符号数乘法（MULX_KEIL16）

MULX_KEIL16

指令操作: AX=AX2 * AX

指令说明: 结果 16 位在 AX

影响标志位:

CY	OV	Z	DBZ
-	-	-	-

[指令码] 9EH

时钟数: 3

32 位无符号除法（DIVU_EABX）

DIVU_EABX

指令操作: EAX=EAX / EBX ;

EBX=EAX % EBX

指令说明: 结果 32 位商在 EAX，32 位余数在 EBX

影响标志位:

CY	OV	Z	DBZ
-	-	-	✓

[指令码] 9FH

时钟数: 19

32 位有符号除法（DIVS_EABX）

DIVS_EABX

指令操作: EAX=EAX / EBX ;

EBX=EAX % EBX

指令说明: 结果 32 位商在 EAX，32 位余数在 EBX

影响标志位:

CY	OV	Z	DBZ
-	-	-	✓

[指令码] A0H

时钟数: 21

16 位无符号除法（DIVU_ABX）

DIVU_ABX

指令操作: AX=AX / BX ;

BX=AX % BX

指令说明: 结果 16 位商在 AX，16 位余数在 BX

影响标志位:

CY	OV	Z	DBZ
-	-	-	✓

[指令码] A1H

时钟数: 11

兼容 KEIL 编译器的 16 位无符号除法（DIVU_KEIL16）

DIVU_KEIL16

指令操作: AX=AX / AX2 ;

AX2=AX % AX2

指令说明: 结果 16 位商在 AX，16 位余数在 AX2

影响标志位:

CY	OV	Z	DBZ
-	-	-	✓

[指令码] A2H

时钟数: 10

16 位有符号除法（DIVS_ABX）

DIVS_ABX

指令操作: AX=AX / BX ;

BX=AX % BX

指令说明: 结果 16 位商在 AX，16 位余数在 BX

影响标志位:

CY	OV	Z	DBZ
-	-	-	✓

[指令码] A3H

时钟数: 13

兼容 KEIL 编译器的 16 位有符号除法（DIVS_KEIL16）

DIVS_KEIL16

指令操作: AX=AX / AX2 ;

AX2=AX % AX2

指令说明: 结果 16 位商在 AX，16 位余数在 AX2

影响标志位:

CY	OV	Z	DBZ
-	-	-	✓

[指令码] A4H

时钟数: 12

64 位无符号除法（DIVU_EABXD）（64 位除以 32 位）

DIVU_EABXD

指令操作: {EAX, EBX} = ({EAX, EBX} / EDX)

EDX = ({EAX, EBX} % EDX)

指令说明: 结果 64 位商在{EAX, EBX}，32 位余数在 EDX

影响标志位:

CY	OV	Z	DBZ
-	-	-	✓

[指令码] A5H

时钟数: 36

32 位无符号除法（DIVU_EBXA）（32 位除以 16 位）

DIVU_EAXB

指令操作: EAX = (EAX / BX)

BX = (EAX % BX)

指令说明: 结果 32 位商在 EAX，16 位余数在 BX

影响标志位:

CY	OV	Z	DBZ
-	-	-	✓

[指令码] A6H

时钟数: 19

清零 32 位 EAX（SET0_EAX）

SET0_EAX

指令操作: EAX=0

指令说明:

影响标志位:

CY	OV	Z	DBZ
-	-	-	-

[指令码] B0H

时钟数: 2

置位 32 位 EAX（SET1_EAX）

SET1_EAX

指令操作: EAX=0xFFFFFFFF

指令说明:

影响标志位:

CY	OV	Z	DBZ
-	-	-	-

[指令码] B1H

时钟数: 2

清零 32 位 EBX（SET0_EBX）

SET0_EBX

指令操作: EBX=0

指令说明:

影响标志位:

CY	OV	Z	DBZ
-	-	-	-

[指令码] B2H

时钟数: 2

置位 32 位 EBX（SET1_EBX）

SET1_EBX

指令操作: EBX=0xFFFFFFFF

指令说明:

影响标志位:

CY	OV	Z	DBZ
-	-	-	-

[指令码] B3H

时钟数: 2

清零 32 位 ECX（SET0_ECX）

SET0_ECX

指令操作: ECX=0

指令说明:

影响标志位:

CY	OV	Z	DBZ
-	-	-	-

[指令码] B4H

时钟数: 2

置位 32 位 ECX（SET1_ECX）

SET1_ECX

指令操作: ECX=0xFFFFFFFF

指令说明:

影响标志位:

CY	OV	Z	DBZ
-	-	-	-

[指令码] B5H

时钟数: 2

清零 32 位 EDX（SET0_EDX）

SET0_EDX

指令操作: EDX=0

指令说明:

影响标志位:

CY	OV	Z	DBZ
-	-	-	-

[指令码] B6H

时钟数: 2

置位 32 位 EDX（SET1_EDX）

SET1_EDX

指令操作: EDX=0xFFFFFFFF

指令说明:

影响标志位:

CY	OV	Z	DBZ
-	-	-	-

[指令码] B7H

时钟数: 2

32 位 EAX 取负数（NEGS_EAX）

NEGS_EAX

指令操作: EAX=-EAX

指令说明:

影响标志位:

CY	OV	Z	DBZ
-	-	-	-

[指令码] B8H

时钟数: 2

32 位 EBX 取负数（NEGS_EBX）

NEGS_EBX

指令操作: EBX=-EBX

指令说明:

影响标志位:

CY	OV	Z	DBZ
-	-	-	-

[指令码] B9H

时钟数: 2

16 位 AX 取负数（NEGS_AX）

NEGS_AX

指令操作: AX=-AX

指令说明:

影响标志位:

CY	OV	Z	DBZ
-	-	-	-

[指令码] BAH

时钟数: 2

16 位 BX 取负数（NEGS_BX）

NEGS_BX

指令操作: BX=-BX

指令说明:

影响标志位:

CY	OV	Z	DBZ
-	-	-	-

[指令码] BBH

时钟数: 2

32 位 EAX 加 1 运算（INC1_EAX）

INC1_EAX

指令操作: EAX=EAX+1

指令说明:

影响标志位:

CY	OV	Z	DBZ
✓	–	✓	-

[指令码] C0H

时钟数: 3

32 位 EBX 加 1 运算（INC1_EBX）

INC1_EBX

指令操作: EBX=EBX+1

指令说明:

影响标志位:

CY	OV	Z	DBZ
✓	–	✓	-

[指令码] C1H

时钟数: 3

32 位 EAX 加 4 运算（INC4_EAX）

INC4_EAX

指令操作: EAX=EAX+4

指令说明:

影响标志位:

CY	OV	Z	DBZ
✓	–	✓	-

[指令码] C2H

时钟数: 3

32 位 EBX 加 4 运算（INC4_EBX）

INC4_EBX

指令操作: EBX=EBX+4

指令说明:

影响标志位:

CY	OV	Z	DBZ
✓	–	✓	-

[指令码] C3H

时钟数: 3

16 位 AX 加 1 运算（INC1_AX）

INC1_AX

指令操作: AX=AX+1

指令说明:

影响标志位:

CY	OV	Z	DBZ
✓	–	✓	-

[指令码] C4H

时钟数: 3

16 位 BX 加 1 运算（INC1_BX）

INC1_BX

指令操作: BX=BX+1

指令说明:

影响标志位:

CY	OV	Z	DBZ
✓	–	✓	-

[指令码] C5H

时钟数: 3

16 位 AX 加 2 运算（INC2_AX）

INC2_AX

指令操作: AX=AX+2

指令说明:

影响标志位:

CY	OV	Z	DBZ
✓	–	✓	-

[指令码] C6H

时钟数: 3

16 位 BX 加 2 运算（INC2_BX）

INC2_BX

指令操作: BX=BX+2

指令说明:

影响标志位:

CY	OV	Z	DBZ
✓	–	✓	-

[指令码] C7H

时钟数: 3

32 位 EAX 减 1 运算（DEC1_EAX）

DEC1_EAX

指令操作: EAX=EAX-1

指令说明:

影响标志位:

CY	OV	Z	DBZ
✓	–	✓	-

[指令码] C8H

时钟数: 3

32 位 EBX 减 1 运算（DEC1_EBX）

DEC1_EBX

指令操作: EBX=EBX-1

指令说明:

影响标志位:

CY	OV	Z	DBZ
✓	–	✓	-

[指令码] C9H

时钟数: 3

32 位 EAX 减 4 运算（DEC4_EAX）

DEC4_EAX

指令操作: EAX=EAX-4

指令说明:

影响标志位:

CY	OV	Z	DBZ
✓	–	✓	-

[指令码] CAH

时钟数: 3

32 位 EBX 减 4 运算（DEC4_EBX）

DEC4_EBX

指令操作: EBX=EBX-4

指令说明:

影响标志位:

CY	OV	Z	DBZ
✓	–	✓	-

[指令码] CBH

时钟数: 3

16 位 AX 减 1 运算（DEC1_AX）

DEC1_AX

指令操作: AX=AX-1

指令说明:

影响标志位:

CY	OV	Z	DBZ
✓	–	✓	-

[指令码] CCH

时钟数: 3

16 位 BX 减 1 运算（DEC1_BX）

DEC1_BX

指令操作: BX=BX-1

指令说明:

影响标志位:

CY	OV	Z	DBZ
✓	–	✓	-

[指令码] CDH

时钟数: 3

16 位 AX 减 2 运算（DEC2_AX）

DEC2_AX

指令操作: AX=AX-2

指令说明:

影响标志位:

CY	OV	Z	DBZ
✓	–	✓	-

[指令码] CEH

时钟数: 3

16 位 BX 减 2 运算（DEC2_BX）

DEC2_BX

指令操作: BX=BX-2

指令说明:

影响标志位:

CY	OV	Z	DBZ
✓	–	✓	-

[指令码] CFH

时钟数: 3

32 位与运算（BAND_EABX）

BAND_EABX

指令操作: EBX=EBX & EAX

指令说明:

影响标志位:

CY	OV	Z	DBZ
–	–	✓	-

[指令码] D0H

时钟数: 3

16 位与运算（BAND_ABX）

BAND_ABX

指令操作: BX=BX & AX

指令说明:

影响标志位:

CY	OV	Z	DBZ
–	–	✓	-

[指令码] D1H

时钟数: 3

32 位或运算（BOR_EABX）

BOR_EABX

指令操作: EBX=EBX | EAX

指令说明:

影响标志位:

CY	OV	Z	DBZ
–	–	✓	-

[指令码] D2H

时钟数: 3

16 位或运算（BOR_ABX）

BOR_ABX

指令操作: BX=BX | AX

指令说明:

影响标志位:

CY	OV	Z	DBZ
–	–	✓	-

[指令码] D3H

时钟数: 3

32 位异或运算（BXOR_EABX）

BXOR_EABX

指令操作: EBX=EBX ^ EAX

指令说明:

影响标志位:

CY	OV	Z	DBZ
–	–	✓	-

[指令码] D4H

时钟数: 3

16 位异或运算（BXOR_ABX）

BXOR_ABX

指令操作: BX=BX ^ AX

指令说明:

影响标志位:

CY	OV	Z	DBZ
–	–	✓	-

[指令码] D5H

时钟数: 3

32 位取反运算（BCPL_EAX）

BCPL_EAX

指令操作: EAX=~EAX

指令说明:

影响标志位:

CY	OV	Z	DBZ
–	–	✓	-

[指令码] D6H

时钟数: 2

32 位取反运算（BCPL_EBX）

BCPL_EBX

指令操作: EBX=~EBX

指令说明:

影响标志位:

CY	OV	Z	DBZ
–	–	✓	-

[指令码] D7H

时钟数: 2

32 位 EAX 逻辑左移 n 位（SHL_EAX）

SHL_EAX

指令操作: {CY, EAX}=EAX << ACC[4:0]

指令说明: 需要移动位数存放在 ACC[4:0]中,低位填充 0

影响标志位:

CY	OV	Z	DBZ
✓	–	–	-

[指令码] E0H

时钟数: 7

32 位 EBX 逻辑左移 n 位（SHL_EBX）

SHL_EBX

指令操作: (CY, EBX)=EBX << ACC[4:0]

指令说明: 需要移动位数存放在 ACC[4:0]中,低位填充 0

影响标志位:

CY	OV	Z	DBZ
✓	–	–	-

[指令码] E1H

时钟数: 7

16 位 AX 逻辑左移 n 位（SHL_AX）

SHL_AX

指令操作: {CY, AX}=AX << ACC[3:0]

指令说明: 需要移动位数存放在 ACC[3:0]中,低位填充 0

影响标志位:

CY	OV	Z	DBZ
✓	–	–	-

[指令码] E2H

时钟数: 6

16 位 BX 逻辑左移 n 位（SHL_BX）

SHL_BX

指令操作: {CY, BX}=BX << ACC[3:0]

指令说明: 需要移动位数存放在 ACC[3:0]中,低位填充 0

影响标志位:

CY	OV	Z	DBZ
✓	–	–	-

[指令码] E3H

时钟数: 6

32 位 EAX 逻辑右移 n 位（SHRU_EAX）

SHRU_EAX

指令操作: {EAX, CY}=EAX >> ACC[4:0]

指令说明: 无符号整数右移，需要移动位数存放在 ACC[4:0]中,高位填充 0

影响标志位:

CY	OV	Z	DBZ
✓	–	–	-

[指令码] E4H

时钟数: 7

32 位 EBX 逻辑右移 n 位（SHRU_EBX）

SHRU_EBX

指令操作: {EBX, CY}=EBX >> ACC[4:0]

指令说明: 无符号整数右移，需要移动位数存放在 ACC[4:0]中,高位填充 0

影响标志位:

CY	OV	Z	DBZ
✓	–	–	-

[指令码] E5H

时钟数: 7

16 位 AX 逻辑右移 n 位（SHRU_AX）

SHRU_AX

指令操作: {AX, CY}=AX >> ACC[3:0]

指令说明: 无符号整数右移，需要移动位数存放在 ACC[3:0]中,高位填充 0

影响标志位:

CY	OV	Z	DBZ
✓	–	–	-

[指令码] E6H

时钟数: 6

16 位 BX 逻辑右移 n 位（SHRU_BX）

SHRU_BX

指令操作: {BX, CY}=BX >> ACC[3:0]

指令说明: 无符号整数右移，需要移动位数存放在 ACC[3:0]中,高位填充 0

影响标志位:

CY	OV	Z	DBZ
✓	–	–	-

[指令码] E7H

时钟数: 6

32 位 EAX 算术右移 n 位（SHRS_EAX）

SHRS_EAX

指令操作: {EAX, CY}=EAX >> ACC[4:0]

指令说明: 有符号整数右移，需要移动位数存放在 ACC[4:0]中,高位填充符号位位

影响标志位:

CY	OV	Z	DBZ
✓	–	–	-

[指令码] E8H

时钟数: 7

32 位 EBX 算术右移 n 位（SHRS_EBX）

SHRS_EBX

指令操作: {EBX, CY}=EBX >> ACC[4:0]

指令说明: 有符号整数右移，需要移动位数存放在 ACC[4:0]中,高位填充符号位位

影响标志位:

CY	OV	Z	DBZ
✓	–	–	-

[指令码] E9H

时钟数: 7

16 位 AX 算术右移 n 位（SHRS_AX）

SHRS_AX

指令操作: {AX, CY}=AX >> ACC[3:0]

指令说明: 有符号整数右移，需要移动位数存放在 ACC[3:0]中,高位填充符号位位

影响标志位:

CY	OV	Z	DBZ
✓	–	–	-

[指令码] EAH

时钟数: 6

16 位 BX 算术右移 n 位（SHRS_BX）

SHRS_BX

指令操作: {BX, CY}=BX >> ACC[3:0]

指令说明: 有符号整数右移，需要移动位数存放在 ACC[3:0]中,高位填充符号位位

影响标志位:

CY	OV	Z	DBZ
✓	–	–	-

[指令码] EBH

时钟数: 6

32 位 EAX 循环右移 n 位（ROR_EAX）

ROR_EAX

指令操作: EAX=EAX >> ACC[4:0]

指令说明: 无符号整数循环右移，需要移动位数存放在 ACC[4:0]中

影响标志位:

CY	OV	Z	DBZ
–	–	–	-

[指令码] ECH

时钟数: 7

32 位 EBX 循环右移 n 位（ROR_EBX）

ROR_EBX

指令操作: EBX=EBX >> ACC[4:0]

指令说明: 无符号整数循环右移，需要移动位数存放在 ACC[4:0]中

影响标志位:

CY	OV	Z	DBZ
–	–	–	-

[指令码] EDH

时钟数: 7

16 位 AX 循环右移 n 位（ROR_AX）

ROR_AX

指令操作: AX=AX >> ACC[3:0]

指令说明: 无符号整数循环右移，需要移动位数存放在 ACC[3:0]中

影响标志位:

CY	OV	Z	DBZ
–	–	–	-

[指令码] EEH

时钟数: 6

16 位 BX 循环右移 n 位（ROR_BX）

ROR_BX

指令操作: BX=BX >> ACC[3:0]

指令说明: 无符号整数循环右移，需要移动位数存放在 ACC[3:0]中

影响标志位:

CY	OV	Z	DBZ
–	–	–	-

[指令码] EFH

时钟数: 6

32 位乘除运算（MMD32_EABX）

MMD32_EABX

指令操作: {ECX,EAX}=EAX*EBX/EDX

指令说明: EAX、EBX、EDX 均为有符号数

影响标志位:

CY	OV	Z	DBZ
–	–	–	✓

[指令码] F0H

时钟数: 43

16 位乘除运算（MMD16_ABX）

MMD16_ABX

指令操作: EAX=AX*BX/DX

指令说明: AX、BX、DX 均为有符号数

影响标志位:

CY	OV	Z	DBZ
–	–	–	✓

[指令码] F1H

时钟数: 21

32 位线性标定（LTC32_EAX）

LTC32_EAX

指令操作: {ECX, EAX}=(EAX-EBX)*ECX/EDX

指令说明:

影响标志位:

CY	OV	Z	DBZ
✓	✓	–	✓

[指令码] F2H

时钟数: 44

16 位线性标定（LTC16_AX）

LTC16_AX

指令操作: EAX=(AX-BX)*CX/DX

指令说明:

影响标志位:

CY	OV	Z	DBZ
✓	✓	–	✓

[指令码] F3H

时钟数: 22

32 位乘加运算（MA32_EDX）

MA32_EDX

指令操作: EDX=EDX+AX*BX

指令说明:

影响标志位:

CY	OV	Z	DBZ
✓	✓	–	–

[指令码] F4H

时钟数: 5

64 位乘加运算（MA64_EDX）

MA64_ECDX

指令操作: { ECX, EDX }={ECX, EDX}+EAX*EBX

指令说明:

影响标志位:

CY	OV	Z	DBZ
✓	✓	–	–

[指令码] F5H

时钟数: 12

附录A　指令集

A.1　指令集简介

A.1.1　BINARY 模式和 SOURCE 模式

Binary 模式和 Source 模式是指令集提供操作码的两种方式。根据用户程序，Binary 模式或者 Source 模式可能会生成更高效的代码。Binary 模式是指 MCU51 的标准操作码。Source 模式是指 MCU251 特定的操作码集，它通过额外的操作和寻址模式扩展指令集。特殊助记符 0xA5 用于区分每种模式下的具体指令。所有未使用的操作码均已正确解码并作为 NOP 执行。

A.1.2　指令集标记

指令有五种不同的寻址模式：立即数寻址、直接寻址、寄存器寻址、间接寻址寻址和相对寻址。在立即寻址模式中，操作数包含在操作码中。对于直接寻址，8 位地址或 16 位地址是操作码的一部分；对于寄存器寻址，在操作码中选择一个寄存器用于操作。在间接寻址模式中，在操作码中选择一个寄存器来指向操作使用的地址。相对寻址模式用于跳转指令。下表提供了 Ai8051U 微控制器内核指令集的周期数。一个周期数等于一个系统时钟。表 1 和表 2 包含指令集表中使用的助记符说明。表 3 到表 7 标示了每条指令执行所需的十六进制代码、字节数和系统时钟数。

Rn N rrr	当前工作字节寄存器R0、R1、…、R7 字节寄存器索引0~7 n的二进制表示
Rm Rmd Rms m,md,ms ssss SSSS	字节寄存器R0、R1、…、R15 目标寄存器 源数寄存器 字节寄存器索引：m、md、ms = 0、1、…、15 m或者md的二进制表示 ms的二进制表示
WR WRjd WRjs @WRj @WRj+dis j, jd, js tttt TTTT	字寄存器WR0、WR2、…、WR30 目标寄存器 源数寄存器 由字寄存器寻址的间接内存位置（0x0000~0xFFFF） 由字寄存器寻址＋0到64KB偏移值的间接内存位置（0x0000~0xFFFF） 字寄存器索引： j、jd、js = 0、2、…、30 j或者jd的二进制表示 js的二进制表示
DRk DRkd DRks	双字寄存器DR0、DR4、…、DR28、DR56、DR60 目标寄存器 源数寄存器

@DRk @DRk+dis k, kd, ks uuuu UUUU	由双字寄存器寻址的间接内存位置（0x000000~0xFFFFFF） 由双字寄存器寻址＋0到64KB偏移值的间接内存位置（0x000000~0xFFFFFF） 双字寄存器索引： k、kd、ks ＝0、4、…、28、56、60 k或者kd的二进制表示 ks的二进制表示
dir8	128个内部内存位置，各种特殊功能寄存器
dir16	16位内存地址（0x000000~0x00FFFF）
@Ri	由寄存器R0或者R1寻址的间接内存位置（0x00~0xFF）
#data	8位立即数包含在指令中
#data16	16位立即数包含在指令中的第2和第3字节
#0data16	32位立即数；高位字填充零，低位字包含在指令中的第2和第3字节
#1data16	32位立即数；高位字填充1，低位字包含在指令中的第2和第3字节
#short vv	等于1、2或者4的常数，包含在指令中 #short 的二进制表示
bit	内存位置（0x20~0x7F）或者任何已定义的SFR中的直接寻址位
bit51	存储器或者SFR中的直接寻址位（位号 = 0x00~0xFF）。位0x00~0x7F是内部内存里字节位置为0x20~0x2F的128个位。 位0x80~0xFF是16个SFR中的128个位，其地址以0或者8结尾：0x80、0x88、0x90、…、0xF0、0xF8
A	累加器

表 1、数据寻址模式注释

addr24	24位目标地址可位于16MB地址空间的任意位置。它用于ECALL和EJMP指令。
addr16	LCALL和LJMP的目标地址可以是64KB程序存储器地址空间内的任何位置。
addr11	ACALL和AJMP的目标地址将与下一条指令的第一个字节位于相同的2 KB程序存储器页面内。
rel	SJMP 和所有条件跳转都包含一个8位偏移字节。范围是相对于下一条指令的第一个字节偏移+127到-128字节。

表 2、程序寻址模式注释

-	该指令不修改标志位。
√	该指令根据需要置位或者清零标志位。
1	该指令置位标志位。
0	该指令清零标志位。

表 3、标志位说明注释

A.1.3 指令表（功能排序）

指令执行总时间取决于 WTST（0xe9）的值。下面每个表标示了 WTST=0 时的时钟数。计算每条指令的总时钟数的通用公式是：

指令时钟数 = 时钟数 + nrPRGACS*WTST （nrPRGACS＝0 或 1）

如果指令访问 XDM 存储器，则应将 CKCON[2:0]值乘以 nrXDMACS（1 或者 2）来计算正确的周期数。

指令时钟数 = 时钟数 + nrPRGACS*WTST + nrXDMACS*CKCON

专用于二进制模式的指令标记为红色。这些指令在源代码模式执行时需要在操作码之前加上 0xA5 前缀（ESC）。专用于源代码模式的指令标记为蓝色。当以二进制模式执行时，这些指令在操作码之前需要加上 0xA5 前缀（ESC）。无论 CPU 模式如何，所有其它指令始终可用。

算术运算

助记符	描述	编码	字节	时钟数
ADD A,Rn	将寄存器加到累加器	0x28-0x2F	1	2
ADD A,dir8	将直接字节加到累加器	0x25	2	1
ADD A,@Ri	将间接内存加到累加器	0x26-0x27	1	2
ADD A,#data	将立即数加到累加器	0x24	2	1
ADD Rm,Rm	将字节寄存器加到字节寄存器	0x2C	2	1
ADD WRj,WRj	将字寄存器加到字寄存器	0x2D	2	1
ADD reg,op2(3)	将操作数加到 Rm、WRj 或者 DRk	0x2E	注1	注2
ADD DRk,DRk	将双字寄存器加到双字寄存器	0x2F	2	1
ADDC A,Rn	将寄存器加到带有进位标志位的累加器	0x38-0x3F	1	2
ADDC A,dir8	将直接字节加到带有进位标志的累加器 A	0x35	2	1
ADDC A,@Ri	将间接内存加到带有进位标志的累加器 A	0x36-0x37	1	2
ADDC A,#data	将立即数加到带有进位标志的累加器 A	0x34	2	1
SUBB A,Rn	从 A 借位再减去寄存器	0x98-0x9F	1	2
SUBB A,dir8	从 A 借位再减去直接字节	0x95	2	1
SUBB A,@Ri	从 A 借位再减去间接内存	0x96-0x97	1	2
SUBB A,#data	从 A 借位再减去立即数	0x94	2	1
SUB Rm,Rm	从字节寄存器中减去字节寄存器	0x9C	2	1
SUB WRj,WRj	从字寄存器中减去字寄存器	0x9D	2	1
SUB reg,op2(3)	从 Rm、WRj 或者 DRk 中减去操作数	0x9E	注1	注2
SUB DRk,DRk	从双字寄存器中减去双字寄存器	0x9F	2	1
CMP Rm,Rm	比较两个字节寄存器	0xBC	2	1
CMP WRj,WRj	比较两个字寄存器	0xBD	2	1
CMP reg,op2(3)	将 Rm、WRj 或者 DRk 与操作数进行比较	0xBE	注1	注2
CMP DRk,DRk	比较两个双字寄存器	0xBF	2	1
INC A	递增累加器	0x04	1	1

助记符	描述	编码	字节	时钟数
INC Rn	递增寄存器	0x08-0x0F	1	2
INC dir8	递增直接字节	0x05	2	1
INC @Ri	递增间接内存	0x06-0x07	1	2
INC reg,#short(3)	递增 Rm、WRj 或者 DRk	0x0B	2	注2
DEC A	递减累加器	0x14	1	1
DEC Rn	递减寄存器	0x18-0x1F	1	2
DEC dir8	递减直接字节	0x15	1	1
DEC @Ri	递减间接内存	0x16-0x17	2	2
DEC reg,#short(3)	递减 Rm、WRj 或者 DRk	0x1B	2	注2
INC DPTR	递增数据指针	0xA3	1	1
MUL A,B	将 A 乘以 B	0xA4	1	1
MUL Rm,Rm	字节寄存器相乘	0xAC	2	1
MUL WRj,WRj	字寄存器相乘	0xAD	2	1
DIV A,B	将 A 除以 B	0x84	1	6
DIV Rm,Rm	字节寄存器相除	0x8C	1	6
DIV WRj,WRj	字寄存器相除	0x8D	1	10
DA A	十进制调整累加器	0xD4	1	3

注 1：指令所需的字节数取决于由紧接着的字节确定的寻址模式。请参考指令集详解。

注 2：指令所需的周期取决于由紧接着的字节确定的寻址模式。请参考指令集详解。

注 3：操作数和寻址模式取决于紧接着的字节。指令集详解中描述了所有选项。

逻辑运算

助记符	描述	编码	字节	时钟数
ANL A,Rn	寄存器逻辑与累加器	0x58-0x5F	1	2
ANL A,dir8	直接字节逻辑与累加器	0x55	2	1
ANL A,@Ri	间接内存逻辑与累加器	0x56-0x57	1	2
ANL A,#data	立即数逻辑与累加器	0x54	2	1
ANL dir8,A	直接字节逻辑与累加器	0x52	2	1
ANL dir8,#data	直接数据逻辑与直接字节	0x53	3	1
ANL Rm,Rm	两个字节寄存器逻辑与	0x5C	2	1
ANL WRj,WRj	两个字寄存器逻辑与	0x5D	2	1
ANL reg,op2(3)	操作数逻辑与 Rm、WRj 或者 DRk	0x5E	注1	注2
ORL A,Rn	寄存器逻辑或累加器	0x48-0x4F	1	2
ORL A,dir8	直接字节逻辑或累加器	0x45	2	1
ORL A,@Ri	间接内存逻辑或累加器	0x46-0x47	1	2
ORL A,#data	立即数逻辑或累加器	0x44	2	1
ORL dir8,A	累加器直接字节	0x42	2	1
ORL dir8,#data	立即数逻辑或直接字节	0x43	3	1
ORL Rm,Rm	两个字节寄存器逻辑或	0x4C	2	1

助记符	描述	编码	字节	时钟数
ORL WRj,WRj	两个字寄存器逻辑或	0x4D	2	1
ORL reg,op2(3)	操作数逻辑或 Rm、WRj 或者 DRk	0x4E	注1	注2
XRL A,Rn	寄存器异或累加器	0x68-0x6F	1	2
XRL A,dir8	直接字节异或累加器	0x65	2	1
XRL A,@Ri	间接内存异或累加器	0x66-0x67	1	2
XRL A,#data	立即数异或累加器	0x64	2	1
XRL dir8,A	直接字节异或累加器	0x62	2	1
XRL dir8,#data	立即数到异或直接字节	0x63	3	1
XRL Rm,Rm	两个字节寄存器异或	0x6C	2	1
XRL WRj,WRj	两个字寄存器异或	0x6D	2	1
XRL reg,op2(3)	操作数异或 Rm、WRj 或者 DRk	0x6E	注1	注2
CLR A	累加器清零	0xE4	1	1
CPL A	累加器取反	0xF4	1	1
RL A	累加器向左循环移动	0x23	1	1
RLC A	累加器带进位向左循环移动	0x33	1	1
RR A	累加器向右循环移动	0x03	1	1
RRC A	累加器带进位向右循环移动	0x13	1	1
SRA reg(3)	通过 MSB 右移 Rm 或者 WRj	0x0E	2	1
SRL reg(3)	右移 Rm 或者 WRj	0x1E	2	1
SLL reg(3)	左移 Rm 或者 WRj	0x3E	2	1
SWAP A	累加器内的交换半字节	0xC4	1	1

注 1：指令所需的字节数取决于由紧接着的字节确定的寻址模式。请参考指令集详解。

注 2：指令所需的周期取决于由紧接着的字节确定的寻址模式。请参考指令集详解。

注 3：操作数和寻址模式取决于紧接着的字节。指令集详解中描述了所有选项。

布尔操作

助记符	描述	编码	字节	时钟数
CLR C	进位标志位清零	0xC3	1	1
CLR bit	直接位清零	0xC2	2	1
SETB C	进位标志位置位	0xD3	1	1
SETB bit	直接位置位	0xD2	2	1
CPL C	进位标志位取反	0xB3	1	1
CPL bit	直接位取反	0xB2	2	1
ANL C,bit	直接位逻辑与进位标志位	0x82	2	1
ANL C,/bit	直接位的非逻辑与进位标志位	0xB0	2	1
ORL C,bit	直接位逻辑或进位标志位	0x72	2	1
ORL C,/bit	直接位的非逻辑或进位标志位	0xA0	2	1
MOV C,bit	直接位搬运到进位标志位	0xA2	2	1
MOV bit,C	进位标志位搬运到直接位	0x92	2	1

Bit instr(1)	位指令集（MCU251 特有）	0xA9	3	1

数据传输

助记符	描述	编码	字节	时钟数
MOV A,Rn	将寄存器搬运到累加器	0xE8-0xEF	1	2
MOV A,dir8	将直接字节搬运到累加器	0xE5	2	1
MOV A,@Ri	将间接内存搬运到累加器	0xE6-0xE7	1	2
MOV A,#data	将立即数搬运到累加器	0x74	2	1
MOV Rn,A	将累加器搬运到寄存器	0xF8-0xFF	1	2
MOV Rn,dir8	将直接字节搬运到寄存器	0xA8-0xAF	2	1
MOV Rn,#data	将立即数搬运到寄存器	0x78-0x7F	2	1
MOV dir8,A	将累加器搬运到直接字节	0xF5	2	1
MOV dir8,Rn	将寄存器搬运到直接字节	0x88-0x8F	2	1
MOV dir8,dir8	将直接字节搬运到直接字节	0x85	3	1
MOV dir8,@Ri	将间接内存搬运到直接字节	0x86-0x87	2	2
MOV dir8,#data	将立即数搬运到直接字节	0x75	3	1
MOV @Ri,A	将累加器搬运到间接内存	0xF6-0xF7	1	2
MOV @Ri,dir8	将直接字节搬运到间接内存	0xA6-0xA7	2	2
MOV @Ri,#data	将立即数搬运到间接内存	0x76-0x77	2	2
MOV Rm,Rm	将字节寄存器搬运到字节寄存器	0x7C	2	1
MOV WRj,WRj	将字寄存器搬运到字寄存器	0x7D	2	1
MOV reg,op2(3)	将操作数搬运到 Rm、WRj 或者 DRk	0x7E	注1	注2
MOV DRk,DRk	将双字寄存器搬运到双字寄存器	0x7F	2	1
MOV WRj,@DRk	将间接（24 位）内存搬运到 WRj	0x0B	3	1
MOV @DRk,WRj	将 WRj 搬运到间接（24 位）内存	0x1B	3	1
MOV Rm, @WRj+dis	将 16 位偏移的间接（16 位）内存搬运到 Rm	0x09	4	1
MOV @WRj+dis,Rm	将 Rm 搬运到 16 位偏移的间接（16 位）内存	0x19	4	1
MOV Rm, @DRk+dis	将 16 位偏移的间接（24 位）内存搬运到 Rm	0x29	4	1
MOV @DRk+dis, Rm	将 Rm 搬运到 16 位偏移的间接（24 位）内存	0x39	4	1
MOV WRj,@WRj+dis	将 16 位偏移的间接（16 位）内存搬运到 WRj	0x49	4	1
MOV @WRj+dis,WRj	将 WRj 搬运到 16 位偏移的间接（16 位）内存	0x59	4	1
MOV WRj,@DRk+dis	将 16 位偏移的间接（24 位）内存搬运到 WRj	0x69	4	1
MOV @DRk+dis,WRj	将 WRj 搬运到 16 位偏移的间接（24 位）内存	0x79	4	1
MOV op1,reg(3)	将 Rm、WRj 或者 DRk 搬运到操作数	0x7A	注1	注2
MOVH DRk,#data16(4)	将 16 位立即数搬运到双字寄存器的高位字	0x7A	4	1
MOVZ WRj,Rm	将字节寄存器搬运到零扩展的字寄存器	0x0A	2	1
MOVS WRj,Rm	将字节寄存器搬运到带符号扩展的字寄存器	0x1A	2	1
MOV DPTR,#data16	将 16 位常数加载到活动的 DPTR	0x90	3	1
MOVC A,@A+DPTR	将代码字节搬运到 DPTR 偏移的累加器	0x93	1	4

助记符	描述	编码	字节	时钟数
MOVC A,@A+PC	将代码字节搬运到 PC 偏移的累加器	0x83	1	3
MOVX A,@Ri	将外部存储器（8 位地址）搬运到 A	0xE2-0xE3	1	3
MOVX A,@DPTR	将外部存储器（16 位地址）搬运到 A	0xE0	1	3
MOVX @Ri,A	将 A 搬运到外部存储器（8 位地址）	0xF2-0xF3	1	2
MOVX @DPTR,A	将 A 搬运到外部存储器（16 位地址）	0xF0	1	2
PUSH dir8	将直接字节压入 IDM 堆栈	0xC0	2	1
POP dir8	从 IDM 堆栈中弹出直接字节	0xD0	2	1
PUSH op1(3)	将操作数压入 IDM 堆栈	0xCA	注1	注2
POP op1(3)	从 IDM 堆栈弹出操作数	0xDA	注1	注2
XCH A,Rn	寄存器跟累加器交换	0xC8-0xCF	1	2
XCH A,dir8	直接字节跟累加器交换	0xC5	2	1
XCH A,@Ri	间接内存跟累加器交换	0xC6-0xC7	1	2
XCHD A,@Ri	间接内存的低位半字节跟 A 交换	0xD6-0xD7	1	2

注 1：指令所需的字节数取决于由紧接着的字节确定的寻址模式。请参考指令集详解。

注 2：指令所需的周期取决于由紧接着的字节确定的寻址模式。请参考指令集详解。

注 3：操作数和寻址模式取决于紧接着的字节。指令集详解中描述了所有选项。

注 4：MOVH 指令的操作码首字节与 MOV op1,reg 组相同。指令通过第二个字节的值来区分。

程序跳转

助记符	描述	编码	字节	时钟数
ACALL addr11	绝对子程序调用	0x11-0xF1	2	3
LCALL addr16	长子程序直接调用	0x12	3	3
ECALL addr24	扩展子程序直接调用	0x9A	4	3
ECALL @DRk	扩展子程序间接调用	0x99	2	3
LCALL @WRj	长子程序间接调用	0x99	2	3
RET	子程序返回	0x22	1	3
ERET	扩展子程序返回	0xAA	1	3
RETI	中断返回	0x32	1	3
AJMP addr11	绝对跳转	0x01-0xE1	2	3
LJMP addr16	直接长跳转	0x02	3	3
EJMP addr24	直接扩展跳转	0x8A	4	3
LJMP @WRj	间接长跳转	0x89	2	3
EJMP @DRk	间接扩展跳转	0x89	2	3
SJMP rel	短跳转（相对地址）	0x80	2	3
JMP @A+DPTR	DPTR 偏移的间接跳转	0x73	1	3
JZ rel	如果累加器为零则跳转	0x60	2	1/3
JNZ rel	如果累加器不为零则跳转	0x70	2	1/3
JC rel	如果进位标志位置位则跳转	0x40	2	1/3
JNC rel	如果进位标志位未置位则跳转	0x50	2	1/3

助记符	描述	编码	字节	时钟数
JB bit,rel	如果直接位置位则跳转	0x20	3	1/3
JNB bit,rel	如果直接位未置位则跳转	0x30	3	1/3
JBC bit,rel	如果直接位置位则跳转并清零位	0x10	3	1/3
JSLE rel	如果小于或者等于则跳转（有符号）	0x08	2	1/3
JSG rel	如果大于则跳转（有符号）	0x18	2	1/3
JLE rel	如果小于或者等于则跳转	0x28	2	1/3
JG rel	如果大于则跳转	0x38	2	1/3
JSL rel	如果小于则跳转（有符号）	0x48	2	1/3
JSGE rel	如果大于或者等于则跳转（有符号）	0x58	2	1/3
JE rel	如果相等则跳转	0x68	2	1/3
JNE rel	如果不相等则跳转	0x78	2	1/3
CJNE A,dir8,rel	将直接字节与 A 比较，如果不相等则跳转	0xB5	3	2/3
CJNE A,#data,rel	将立即数与 A 比较，如果不相等则跳转	0xB4	3	1/3
CJNE Rn,#data,rel	将立即数与寄存器比较。如果不相等则跳转	0xB8-0xBF	3	3/4
CJNE @Ri,#data,rel	将立即数与间接内存比较。如果不相等则跳转	0xB6-0xB7	3	3/4
DJNZ Rn,rel	寄存器递减，如果不为零则跳转	0xD8-0xDF	2	3/4
DJNZ dir8,rel	直接字节递减，如果不为零则跳转	0xD5	3	2/3
NOP	无操作	0x00	1	1

注意：未带条件的跳转在 1 个时钟周期内执行。

A.1.4　指令表（机器码排序）

BINARY 模式

操作码	助记符	操作码	助记符	操作码	助记符	操作码	助记符
00 H	NOP	40 H	JC rel	80 H	SJMP rel	C0 H	PUSH direct
01 H	AJMP addr11	41 H	AJMP addr11	81 H	AJMP addr11	C1 H	AJMP addr11
02 H	LJMP addr16	42 H	ORL direct,A	82 H	ANL C,bit	C2 H	CLR bit
03 H	RR A	43 H	ORL direct,#data	83 H	MOVC A,@A+PC	C3 H	CLR C
04 H	INC A	44 H	ORL A,#data	84 H	DIV AB	C4 H	SWAP A
05 H	INC direct	45 H	ORL A,direct	85 H	MOV direct,direct	C5 H	XCH A, direct
06 H	INC @R0	46 H	ORL A,@R0	86 H	MOV direct,@R0	C6 H	XCH A,@R0
07 H	INC @R1	47 H	ORL A,@R1	87 H	MOV direct,@R1	C7 H	XCH A,@R1
08 H	INC R0	48 H	ORL A,R0	88 H	MOV direct,R0	C8 H	XCH A,R0
09 H	INC R1	49 H	ORL A,R1	89 H	MOV direct,R1	C9 H	XCH A,R1
0A H	INC R2	4A H	ORL A,R2	8A H	MOV direct,R2	CA H	XCH A,R2
0B H	INC R3	4B H	ORL A,R3	8B H	MOV direct,R3	CB H	XCH A,R3
0C H	INC R4	4C H	ORL A,R4	8C H	MOV direct,R4	CC H	XCH A,R4
0D H	INC R5	4D H	ORL A,R5	8D H	MOV direct,R5	CD H	XCH A,R5
0E H	INC R6	4E H	ORL A,R6	8E H	MOV direct,R6	CE H	XCH A,R6
0F H	INC R7	4F H	ORL A,R7	8F H	MOV direct,R7	CF H	XCH A,R7
10 H	JBC bit,rel	50 H	JNC rel	90 H	MOV DPTR,#data16	D0 H	POP direct
11 H	ACALL addr11	51 H	ACALL addr11	91 H	ACALL addr11	D1 H	ACALL addr11
12 H	LCALL addr16	52 H	ANL direct,A	92 H	MOV bit,C	D2 H	SETB bit
13 H	RRC A	53 H	ANL direct,#data	93 H	MOVC A,@A+DPTR	D3 H	SETB C
14 H	DEC A	54 H	ANL A,#data	94 H	SUBB A,#data	D4 H	DA A
15 H	DEC direct	55 H	ANL A,direct	95 H	SUBB A,direct	D5 H	DJNZ direct, rel
16 H	DEC @R0	56 H	ANL A,@R0	96 H	SUBB A,@R0	D6 H	XCHD A,@R0
17 H	DEC @R1	57 H	ANL A,@R1	97 H	SUBB A,@R1	D7 H	XCHD A,@R1
18 H	DEC R0	58 H	ANL A,R0	98 H	SUBB A,R0	D8 H	DJNZ R0,rel
19 H	DEC R1	59 H	ANL A,R1	99 H	SUBB A,R1	D9 H	DJNZ R1,rel
1A H	DEC R2	5A H	ANL A,R2	9A H	SUBB A,R2	DA H	DJNZ R2,rel
1B H	DEC R3	5B H	ANL A,R3	9B H	SUBB A,R3	DB H	DJNZ R3,rel
1C H	DEC R4	5C H	ANL A,R4	9C H	SUBB A,R4	DC H	DJNZ R4,rel
1D H	DEC R5	5D H	ANL A,R5	9D H	SUBB A,R5	DD H	DJNZ R5,rel
1E H	DEC R6	5E H	ANL A,R6	9E H	SUBB A,R6	DE H	DJNZ R6,rel
1F H	DEC R7	5F H	ANL A,R7	9F H	SUBB A,R7	DF H	DJNZ R7,rel
20 H	JB bit,rel	60 H	JZ rel	A0 H	ORL C,bit	E0 H	MOVX A,@DPTR
21 H	AJMP addr11	61 H	AJMP addr11	A1 H	AJMP addr11	E1 H	AJMP addr11

操作码	助记符	操作码	助记符	操作码	助记符	操作码	助记符
22 H	RET	62 H	XRL direct,A	A2 H	MOV C,bit	E2 H	MOVX A,@R0
23 H	RL A	63 H	XRL direct,#data	A3 H	INC DPTR	E3 H	MOVX A,@R1
24 H	ADD A,#data	64 H	XRL A,#data	A4 H	MUL AB	E4 H	CLR A
25 H	ADD A,direct	65 H	XRL A,direct	A5 H	ESC	E5 H	MOV A, direct
26 H	ADD A,@R0	66 H	XRL A,@R0	A6 H	MOV @R0,direct	E6 H	MOV A,@R0
27 H	ADD A,@R1	67 H	XRL A,@R1	A7 H	MOV @R1,direct	E7 H	MOV A,@R1
28 H	ADD A,R0	68 H	XRL A,R0	A8 H	MOV R0,direct	E8 H	MOV A,R0
29 H	ADD A,R1	69 H	XRL A,R1	A9 H	MOV R1,direct	E9 H	MOV A,R1
2A H	ADD A,R2	6A H	XRL A,R2	AA H	MOV R2,direct	EA H	MOV A,R2
2B H	ADD A,R3	6B H	XRL A,R3	AB H	MOV R3,direct	EB H	MOV A,R3
2C H	ADD A,R4	6C H	XRL A,R4	AC H	MOV R4,direct	EC H	MOV A,R4
2D H	ADD A,R5	6D H	XRL A,R5	AD H	MOV R5,direct	ED H	MOV A,R5
2E H	ADD A,R6	6E H	XRL A,R6	AE H	MOV R6,direct	EE H	MOV A,R6
2F H	ADD A,R7	6F H	XRL A,R7	AF H	MOV R7,direct	EF H	MOV A,R7
30 H	JNB bit,rel	70 H	JNZ rel	B0 H	ANL C,bit	F0 H	MOVX @DPTR,A
31 H	ACALL addr11	71 H	ACALL addr11	B1 H	ACALL addr11	F1 H	ACALL addr11
32 H	RETI	72 H	ORL C,direct	B2 H	CPL bit	F2 H	MOVX @R0,A
33 H	RLC A	73 H	JMP @A+DPTR	B3 H	CPL C	F3 H	MOVX @R1,A
34 H	ADDC A,#data	74 H	MOV A,#data	B4 H	CJNE A,#data,rel	F4 H	CPL A
35 H	ADDC A,direct	75 H	MOV direct,#data	B5 H	CJNE A,direct,rel	F5 H	MOV direct, A
36 H	ADDC A,@R0	76 H	MOV @R0,#data	B6 H	CJNE @R0,#data,rel	F6 H	MOV @R0,A
37 H	ADDC A,@R1	77 H	MOV @R1,#data	B7 H	CJNE @R1,#data,rel	F7 H	MOV @R1,A
38 H	ADDC A,R0	78 H	MOV R0,#data	B8 H	CJNE R0,#data,rel	F8 H	MOV R0,A
39 H	ADDC A,R1	79 H	MOV R1,#data	B9 H	CJNE R1,#data,rel	F9 H	MOV R1,A
3A H	ADDC A,R2	7A H	MOV R2,#data	BA H	CJNE R2,#data,rel	FA H	MOV R2,A
3B H	ADDC A,R3	7B H	MOV R3,#data	BB H	CJNE R3,#data,rel	FB H	MOV R3,A
3C H	ADDC A,R4	7C H	MOV R4,#data	BC H	CJNE R4,#data,rel	FC H	MOV R4,A
3D H	ADDC A,R5	7D H	MOV R5,#data	BD H	CJNE R5,#data,rel	FD H	MOV R5,A
3E H	ADDC A,R6	7E H	MOV R6,#data	BE H	CJNE R6,#data,rel	FE H	MOV R6,A
3F H	ADDC A,R7	7F H	MOV R7,#data	BF H	CJNE R7,#data,rel	FF H	MOV R7,A

SOURCE 模式

操作码	助记符	操作码	助记符	操作码	助记符	操作码	助记符
00 H	NOP	40 H	JC rel	80 H	SJMP rel	C0 H	PUSH direct
01 H	AJMP addr11	41 H	AJMP addr11	81 H	AJMP addr11	C1 H	AJMP addr11
02 H	LJMP addr16	42 H	ORL direct,A	82 H	ANL C,bit	C2 H	CLR bit
03 H	RR A	43 H	ORL direct,#data	83 H	MOVC A,@A+PC	C3 H	CLR C
04 H	INC A	44 H	ORL A,#data	84 H	DIV AB	C4 H	SWAP A
05 H	INC direct	45 H	ORL A,direct	85 H	MOV direct,direct	C5 H	XCH A, direct
06 H	-	46 H	-	86 H	-	C6 H	-
07 H	-	47 H	-	87 H	-	C7 H	-
08 H	JSLE rel	48 H	JSL rel	88 H	-	C8 H	-
09 H	MOV Rm,@WRj+dis	49 H	MOV WRj,@WRj+dis	89 H	LJMP @WRj EJMP @DRk	C9 H	-
0A H	MOVZ WRj,Rm	4A H	-	8A H	EJMP addr24	CA H	PUSH op1
0B H	INC R,#short MOV WRj,@DRk	4B H	-	8B H	-	CB H	-
0C H	-	4C H	ORL Rm,Rm	8C H	DIV Rm,Rm	CC H	-
0D H	-	4D H	ORL WRj,WRj	8D H	DIV WRj,WRj	CD H	-
0E H	SRA reg	4E H	ORL reg,op2	8E H	-	CE H	-
0F H	-	4F H	-	8F H	-	CF H	-
10 H	JBC bit,rel	50 H	JNC rel	90 H	MOV DPTR,#data16	D0 H	POP direct
11 H	ACALL addr11	51 H	ACALL addr11	91 H	ACALL addr11	D1 H	ACALL addr11
12 H	LCALL addr16	52 H	ANL direct,A	92 H	MOV bit,C	D2 H	SETB bit
13 H	RRC A	53 H	ANL direct,#data	93 H	MOVC A,@A+DPTR	D3 H	SETB C
14 H	DEC A	54 H	ANL A,#data	94 H	SUBB A,#data	D4 H	DA A
15 H	DEC direct	55 H	ANL A,direct	95 H	SUBB A,direct	D5 H	DJNZ direct, rel
16 H	-	56 H	-	96 H	-	D6 H	-
17 H	-	57 H	-	97 H	-	D7 H	-
18 H	JSG rel	58 H	JSGE rel	98 H	-	D8 H	-
19 H	MOV @WRj+dis,Rm	59 H	MOV @WRj+dis,WRj	99 H	LCALL @WRj ECALL @DRk	D9 H	-
1A H	MOVS WRj,Rm	5A H	-	9A H	ECALL addr24	DA H	POP op1
1B H	DEC R,#short MOV @DRk, WRj	5B H	-	9B H	-	DB H	-
1C H	-	5C H	ANL Rm,Rm	9C H	SUB Rm,Rm	DC H	-
1D H	-	5D H	ANL WRj,WRj	9D H	SUB WRj,WRj	DD H	-
1E H	SRL reg	5E H	ANL reg,op2	9E H	SUB reg,op2	DE H	-
1F H	-	5F H	-	9F H	SUB DRk,DRk	DF H	-
20 H	JB bit,rel	60 H	JZ rel	A0 H	ORL C,bit	E0 H	MOVX A,@DPTR

操作码	助记符	操作码	助记符	操作码	助记符	操作码	助记符
21 H	AJMP addr11	61 H	AJMP addr11	A1 H	AJMP addr11	E1 H	AJMP addr11
22 H	RET	62 H	XRL direct,A	A2 H	MOV C,bit	E2 H	MOVX A,@R0
23 H	RL A	63 H	XRL direct,#data	A3 H	INC DPTR	E3 H	MOVX A,@R1
24 H	ADD A,#data	64 H	XRL A,#data	A4 H	MUL AB	E4 H	CLR A
25 H	ADD A,direct	65 H	XRL A,direct	A5 H	ESC	E5 H	MOV A, direct
26 H	-	66 H	-	A6 H	-	E6 H	-
27 H	-	67 H	-	A7 H	-	E7 H	-
28 H	JLE rel	68 H	JE rel	A8 H	-	E8 H	-
29 H	MOV Rm,@DRk+dis	69 H	MOV WRj,@DRk+dis	A9 H	Bit instructions	E9 H	-
2A H	-	6A H	-	AA H	ERET	EA H	-
2B H	-	6B H	-	AB H	-	EB H	-
2C H	ADD Rm,Rm	6C H	XRL Rm,Rm	AC H	MUL Rm,Rm	EC H	-
2D H	ADD WRj,WRj	6D H	XRL WRj,WRj	AD H	MUL WRj,WRj	ED H	-
2E H	ADD reg,op2	6E H	XRL reg,op2	AE H	-	EE H	-
2F H	ADD DRk,DRk	6F H	-	AF H	-	EF H	-
30 H	JNB bit,rel	70 H	JNZ rel	B0 H	ANL C,bit	F0 H	MOVX @DPTR,A
31 H	ACALL addr11	71 H	ACALL addr11	B1 H	ACALL addr11	F1 H	ACALL addr11
32 H	RETI	72 H	ORL C,direct	B2 H	CPL bit	F2 H	MOVX @R0,A
33 H	RLC A	73 H	JMP @A+DPTR	B3 H	CPL C	F3 H	MOVX @R1,A
34 H	ADDC A,#data	74 H	MOV A,#data	B4 H	CJNE A,#data,rel	F4 H	CPL A
35 H	ADDC A,direct	75 H	MOV direct,#data	B5 H	CJNE A,direct,rel	F5 H	MOV direct, A
36 H	-	76 H	-	B6 H	-	F6 H	-
37 H	-	77 H	-	B7 H	-	F7 H	-
38 H	JG rel	78 H	JNE rel	B8 H	-	F8 H	-
39 H	MOV @DRk+dis,Rm	79 H	MOV @DRk+dis,WRj	B9 H	TRAP	F9 H	-
3A H	-	7A H	MOVH DRk,#data16 MOV op1,reg	BA H	-	FA H	-
3B H	-	7B H	-	BB H	-	FB H	-
3C H	-	7C H	MOV Rm,Rm	BC H	CMP Rm,Rm	FC H	-
3D H	-	7D H	MOV WRj,WRj	BD H	CMP WRj,WRj	FD H	-
3E H	SLL reg	7E H	MOV reg,op2	BE H	CMP reg,op2	FE H	-
3F H	-	7F H	MOV DRk,DRk	BF H	CMP DRk,DRk	FF H	-

A.2　指令详解

ACALL <addr11>

功能: 绝对调用

说明: ACALL 无条件调用位于指定地址的子程序。该指令将 PC 递增两次以获得下一条指令的地址，然后将 16 位结果压入堆栈（低字节在前）并增加堆栈指针两次。目标地址是由 PC 递增后的 5 个高位、操作码第 7 到第 5 位以及指令的第二个字节依次串联得到的。因此调用的子程序必须位于 ACALL 之后指令的第一个字节相同的 2K 程序存储器块内开始。不影响标志位。

影响标志位:

CY	AC	OV	N	Z
—	—	—	—	—

Binary 模式	**Source 模式**
指令长度: 2	2
时钟数: 3	3
Hex: 指令码	指令码

[指令码] 11H, 31H, 51H, 71H, 91H, B1H, D1H, F1H

A10	A9	A8	1	0	0	0	1	A7	A6	A5	A4	A3	A2	A1	A0

指令操作: ACALL

(PC) ← (PC) + 2

(SP) ← (SP) + 1

((SP)) ← (PC.7:0)

(SP) ← (SP) + 1

((SP)) ← (PC.15:8)

(PC.10:0) ← 页地址

ADD <dest>,<src>

功能: 源数加到目标操作数。

说明: 将源操作数加到目标操作数，可以是寄存器或者累加器，将计算结果留在寄存器或者累加器中。如果第 7 位（CY）有进位，则 CY 标志位置位。

如果加了字节变量，并且如果第 3 位有进位（AC），则 AC 标志位置位。对于无符号整数的加法，CY 标志位指示发生了溢出。如果第 6 位进位但是第 7 位没有进位，或者第 7 位有进位但是第 6 位没有进位，则 OV 标志位置位。有符号整数相加时，OV 标志位指示两个正操作数之和出现了负数，或者两个负操作数之和出现了正数。本描述中的第 6 位和第 7 位指的是操作数的最高有效字节（8、16 或者 32 位）。结果会影响 N 和 Z 标志位。源操作数允许的寻址模式是寄存器、直接、寄存器间接和立即数寻址。

ADD A,Rn

指令操作: (PC) ←(PC) + 1

(A) ←(A) + (Rn)

影响标志位:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[指令码] 28H – 2FH

0	0	1	0	1	r	r	r

Binary 模式	**Source 模式**
指令长度: 1	2

时钟数:	1	2
Hex:	指令码	[A5]指令码

ADD A,Direct

指令操作:	(PC)	←(PC) + 2
	(A)	←(A) + (direct)

影响标志位:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[指令码] 25H

0	0	1	0	0	1	0	1	直接地址

	Binary 模式	**Source 模式**
指令长度:	2	2
时钟数:	1	1
Hex:	指令码	指令码

ADD A,@Ri

指令操作:	(PC)	←(PC) + 1
	(A)	←(A) + ((Ri))

影响标志位:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[指令码] 26H, 27H

0	0	1	0	0	1	1	i	

	Binary 模式	**Source 模式**
指令长度:	1	2
时钟数:	1	2
Hex:	指令码	[A5]指令码

ADD A,#DATA

指令操作:	(PC)	←(PC) + 2
	(A)	←(A) + #data

影响标志位:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[指令码] 24H

0	0	1	0	0	1	0	0	立即数

	Binary 模式	**Source 模式**
指令长度:	2	2
时钟数:	1	1
Hex:	指令码	指令码

ADD Rmd,Rms

指令操作:	(PC)	←(PC) + 2
	(Rmd)	←(Rms) + (Rmd)

影响标志位:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[指令码] 2CH

0	0	1	0	1	1	0	0	s	s	s	s	S	S	S	S

	Binary 模式	Source 模式
指令长度:	3	2
时钟数:	2	1
Hex:	[A5]指令码	指令码

ADD WRjd,WRjs

指令操作: (PC) ←(PC) + 2

(WRjd) ←(WRjs) + (WRjd)

影响标志位:

CY	AC	OV	N	Z
✓	-	✓	✓	✓

[指令码] 2DH

0	0	1	0	1	1	0	1	t	t	t	t	T	T	T	T

	Binary 模式	Source 模式
指令长度:	3	2
时钟数:	2	1
Hex:	[A5]指令码	指令码

ADD DRkd,DRks

指令操作: (PC) ←(PC) + 2

(DRkd) ←(DRks) + (DRkd)

影响标志位:

CY	AC	OV	N	Z
✓	-	✓	✓	✓

[指令码] 2FH

0	0	1	0	1	1	1	1	u	u	u	u	U	U	U	U

	Binary 模式	Source 模式
指令长度:	3	2
时钟数:	2	1
Hex:	[A5]指令码	指令码

ADD Rm,#DATA

指令操作: (PC) ←(PC) + 3

(Rm) ←(Rm) + #data

影响标志位:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[指令码] 2E(s)0H

0	0	1	0	1	1	1	0	s	s	s	s	0	0	0	0	立即数

	Binary 模式	Source 模式
指令长度:	4	3
时钟数:	2	1
Hex:	[A5]指令码	指令码

ADD WRj,#DATA16

指令操作: (PC) ←(PC) + 4

(WRj) ←(WRj) + #data16

影响标志位:

CY	AC	OV	N	Z
✓	-	✓	✓	✓

[指令码] 2E(t)4H

0 0 1 0	1 1 1 0	t t t t	0 1 0 0	立即数高字节	立即数低字节

	Binary 模式	**Source 模式**
指令长度:	5	4
时钟数:	2	1
Hex:	[A5]指令码	指令码

ADD DRk,#0DATA16

指令操作: (PC) ←(PC) + 4

(DRk) ←(DRk) + #data16

影响标志位:

CY	AC	OV	N	Z
✓	-	✓	✓	✓

[指令码] 2E(u)8H

0 0 1 0	1 1 1 0	u u u u	1 0 0 0	立即数高字节	立即数低字节

	Binary 模式	**Source 模式**
指令长度:	5	4
时钟数:	2	1
Hex:	[A5]指令码	指令码

ADD Rm,DIR8

指令操作: (PC) ←(PC) + 3

(Rm) ←(Rm) + (dir8)

影响标志位:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[指令码] 2E(s)1H

0 0 1 0	1 1 1 0	s s s s	0 0 0 1	直接地址

	Binary 模式	**Source 模式**
指令长度:	4	3
时钟数:	2	1
Hex:	[A5]指令码	指令码

ADD WRj,DIR8

指令操作: (PC) ←(PC) + 3

(WRj) ←(WRj) + (dir8)

影响标志位:

CY	AC	OV	N	Z
✓	-	✓	✓	✓

[指令码] 2E(t)5H

0 0 1 0	1 1 1 0	t t t t	0 1 0 1	直接地址

	Binary 模式	**Source 模式**
指令长度:	4	3
时钟数:	2 (3 for SFR)	1 (2 for SFR)
Hex:	[A5]指令码	指令码

ADD Rm,DIR16

指令操作: (PC) ←(PC) + 4

(Rm) ←(Rm) + (dir16)

影响标志位:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[指令码] 2E(s)3H

0	0	1	0	1	1	1	0	s	s	s	s	0	0	1	1	地址高字节	地址低字节

Binary 模式　**Source 模式**

指令长度: 5　4

时钟数: 2　1

Hex: [A5]指令码　指令码

ADD WRj,DIR16

指令操作: (PC)　←(PC) + 4

(WRj)　←(WRj) + (dir16)

影响标志位:

CY	AC	OV	N	Z
✓	-	✓	✓	✓

[指令码] 2E(t)7H

0	0	1	0	1	1	1	0	t	t	t	t	0	1	1	1	地址高字节	地址低字节

Binary 模式　**Source 模式**

指令长度: 5　4

时钟数: 2　1

Hex: [A5]指令码　指令码

ADD Rm,@WRj

指令操作: (PC)　←(PC) + 3

(Rm)　←(Rm) + ((WRj))

影响标志位:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[指令码] 2E(t)9(s)0H

0	0	1	0	1	1	1	0	t	t	t	t	1	0	0	1	s	s	s	s	0	0	0	0

Binary 模式　**Source 模式**

指令长度: 4　3

时钟数: 1　1

Hex: [A5]指令码　指令码

ADD Rm,@DRk

指令操作: (PC)　←(PC) + 3

(Rm)　←(Rm) + ((DRk))

影响标志位:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[指令码] 2E(u)B(s)0H

0	0	1	0	1	1	1	0	u	u	u	u	1	0	1	1	s	s	s	s	0	0	0	0

Binary 模式　**Source 模式**

指令长度: 4　3

时钟数: 2　1

Hex: [A5]指令码　指令码

ADDC A,<src-byte>

功能: 将 A 和源操作数相加，如果 CY 已置位，则加一（1），并将结果放入 A。

说明: ADDC 同时将指定的字节变量、进位标志位和累加器内容相加，将结果留在累加器中。如果第 7 位或者第 3 位有进位，则分别将进位和辅助进位标志位置位，否则清零。添加无符号整数时，进位标志位指示发生溢出。如果第 6 位有进位但是第 7 位没有进位，或者第 7 位有进位但是第 6 位没有进位，则 OV 置位；否则 OV 清零。有符号整数相加时，OV 表示两个正操作数之和出现了负数，或者两个负操作数之和出现了正数。N 和 Z 标志位也会根据结果受到影响。源操作数允许四种寻址模式：寄存器、直接、寄存器间接或者立即数寻址。

影响标志位:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

ADDC A,Rn

指令操作: (PC) ←(PC) + 1

(A) ←(A) + (C) + (Rn)

[指令码] 38H – 3FH

0	0	1	1	1	r	r	r

Binary 模式 **Source 模式**

指令长度: 1 2

时钟数: 1 2

Hex: 指令码 [A5]指令码

ADDC A,DIRECT

指令操作: (PC) ←(PC) + 2

(A) ←(A) + (C) + (direct)

[指令码] 35H

0	0	1	1	0	1	0	1	直接地址

Binary 模式 **Source 模式**

指令长度: 2 2

时钟数: 1 1

Hex: 指令码 指令码

ADDC A,@Ri

指令操作: (PC) ←(PC) + 1

(A) ←(A) + (C) + ((Ri))

[指令码] 36H, 37H

0	0	1	1	0	1	1	i

Binary 模式 **Source 模式**

指令长度: 1 2

时钟数: 1 2

Hex: 指令码 [A5]指令码

ADDC A,#DATA

指令操作: (PC) ←(PC) + 2

(A) ←(A) + (C) + #data

[指令码] 34H

0	0	1	1	0	1	0	0	立即数

	Binary 模式	Source 模式
指令长度:	2	2
时钟数:	1	1
Hex:	指令码	指令码

AJMP addr11

功能: 绝对跳转

说明: AJMP 将程序执行转移到指定的地址，该地址是在运行时通过串联 PC 的高 5 位（在 PC 递增两次后）、操作码第 7 到第 5 位和指令的第二个字节而形成的。因此，目标必须与 AJMP 之后指令的第一个字节位于相同的 2K 程序存储器块内。

影响标志位:

CY	AC	OV	N	Z
—	—	—	—	—

指令操作:	(PC)	←(PC) + 2
	(PC10-0)	←页地址

[指令码] 01H, 21H, 41H, 61H, 81H, A1H, C1H, E1H

a10	a9	a8	0	0	0	0	1	a7	a6	a5	a4	a3	a2	a1	a0

	Binary 模式	Source 模式
指令长度:	2	2
时钟数:	3	3
Hex:	指令码	指令码

ANL <dest>,<src>

功能: 字节操作数的逻辑与

说明: 在指定变量之间执行按位逻辑与运算并将结果存储在目标变量中。这两个操作数允许 10 种寻址模式组合。当目标是寄存器或者累加器时，源数可以使用寄存器、直接、寄存器间接或者立即数寻址；当目标是直接地址时，源数可以是累加器或者立即数。N 和 Z 标志位会根据结果受到影响。

注意：当该指令用于修改输出端口时，用作原始端口数据的值将从输出数据锁存器中读取，而不是输入引脚。

ANL A,Rn

指令操作:	(PC)	←(PC) + 1
	(A)	←(A) and (Rn)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 58H – 5FH

0	1	0	1	1	r	r	r

	Binary 模式	Source 模式
指令长度:	1	2
时钟数:	1	2
Hex:	指令码	[A5]指令码

ANL A,Direct

指令操作:	(PC)	←(PC) + 2
	(A)	←(A) and (direct)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 55H

0	1	0	1	0	1	0	1	直接地址

Binary 模式　**Source 模式**

指令长度: 2　2

时钟数: 1　1

Hex: 指令码　指令码

ANL A,@Ri

指令操作: (PC)　←(PC) + 1

(A)　←(A) and ((Ri))

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 56H, 57H

0	1	0	1	0	1	1	i

Binary 模式　**Source 模式**

指令长度: 1　2

时钟数: 1　2

Hex: 指令码　[A5]指令码

ANL A,#DATA

指令操作: (PC)　←(PC) + 2

(A)　←(A) and #data

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 54H

0	1	0	1	0	1	0	0	立即数

Binary 模式　**Source 模式**

指令长度: 2　2

时钟数: 1　1

Hex: 指令码　指令码

ANL Direct,A

指令操作: (PC)　←(PC) + 2

(direct)　←(direct) and (A)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 52H

0	1	0	1	0	0	1	0	直接地址

Binary 模式　**Source 模式**

指令长度: 2　2

时钟数: 1　1

Hex: 指令码　指令码

ANL Direct,#DATA

指令操作: (PC) ←(PC) + 3

(direct) ←(direct) and #data

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 53H

0	1	0	1	0	0	1	1	直接地址	立即数

Binary 模式 **Source 模式**

指令长度: 3 3

时钟数: 1 1

Hex: 指令码 指令码

ANL Rmd,Rms

指令操作: (PC) ←(PC) + 2

(Rmd) ←(Rms) and (Rmd)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 5CH

0	1	0	1	1	1	0	0	s	s	s	s	S	S	S	S

Binary 模式 **Source 模式**

指令长度: 3 2

时钟数: 2 1

Hex: [A5]指令码 指令码

ANL WRjd,WRjs

指令操作: (PC) ←(PC) + 2

(WRjd) ←(WRjs) and (WRjd)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 5DH

0	1	0	1	1	1	0	1	t	t	t	t	T	T	T	T

Binary 模式 **Source 模式**

指令长度: 3 2

时钟数: 2 1

Hex: [A5]指令码 指令码

ANL Rm,#DATA

指令操作: (PC) ←(PC) + 3

(Rm) ←(Rm) and #data

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 5E(s)0H

0	1	0	1	1	1	1	0	s	s	s	s	0	0	0	0	立即数

Binary 模式 **Source 模式**

指令长度: 4 3

时钟数: 2 1

Hex: [A5]指令码 指令码

ANL WRj,#DATA16

指令操作: (PC) ←(PC) + 4

(WRj) ←(WRj) and #data16

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 5E(t)4H

0 1 0 1 1 1 1 0	t t t t	0 1 0 0	立即数高字节	立即数低字节	

Binary 模式 **Source 模式**

指令长度: 5 4

时钟数: 2 1

Hex: [A5]指令码 指令码

ANL Rm,DIR8

指令操作: (PC) ←(PC) + 3

(Rm) ←(Rm) and (dir8)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 5E(s)1H

0 1 0 1 1 1 1 0	s s s s	0 0 0 1	直接地址	

Binary 模式 **Source 模式**

指令长度: 4 3

时钟数: 2 1

Hex: [A5]指令码 指令码

ANL WRj,DIR8

指令操作: (PC) ←(PC) + 3

(WRj) ←(WRj) and (dir8)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 5E(t)5H

0 1 0 1 1 1 1 0	t t t t	0 1 0 1	直接地址	

Binary 模式 **Source 模式**

指令长度: 4 3

时钟数: 2 (3 for SFR) 1 (2 for SFR)

Hex: [A5]指令码 指令码

ANL Rm,DIR16

指令操作: (PC) ←(PC) + 4

(Rm) ←(Rm) and (dir16)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 5E(s)3H

0 1 0 1 1 1 1 0	s s s s	0 0 1 1	地址高字节	地址低字节	

Binary 模式 **Source 模式**

指令长度: 5 4

时钟数: 2　1

Hex: [A5]指令码　指令码

ANL WRj,DIR16

指令操作: (PC)　←(PC) + 4

(WRj)　←(WRj) and (dir16)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 5E(t)7H

0 1 0 1	1 1 1 0	t t t t	0 1 1 1	地址高字节	地址低字节

Binary 模式　**Source 模式**

指令长度: 5　4

时钟数: 2　1

Hex: [A5]指令码　指令码

ANL Rm,@WRj

指令操作: (PC)　←(PC) + 3

(Rm)　←(Rm) and ((WRj))

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 5E(t)9(s)0H

0 1 0 1	1 1 1 0	t t t t	1 0 0 1	s s s s	0 0 0 0

Binary 模式　**Source 模式**

指令长度: 4　3

时钟数: 2　1

Hex: [A5]指令码　指令码

ANL Rm,@DRk

指令操作: (PC)　←(PC) + 3

(Rm)　←(Rm) and ((DRk))

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 5E(u)B(s)0H

0 1 0 1	1 1 1 0	u u u u	1 0 1 1	s s s s	0 0 0 0

Binary 模式　**Source 模式**

指令长度: 4　3

时钟数: 2　1

Hex: [A5]指令码　指令码

ANL C,<src–bit>

功能: 位操作数的逻辑与

说明: 如果源位的布尔值为逻辑 0，则进位标志位清零；否则进位标志位保持当前状态。汇编语言中操作数前的斜杠（“/”）表示将寻址位的逻辑补码用作源数的值，但源位本身不受影响。不影响其它标志位。只允许直接位寻址作为源操作数。

ANL C,bit51

指令操作: (PC) ←(PC) + 2

(A) ←(C) and (bit51)

影响标志位:

CY	AC	OV	N	Z
✓	-	-	-	-

[指令码] 82H

1 0 0 0	0 0 1 0	位地址

Binary 模式 **Source 模式**

指令长度: 2 2

时钟数: 1 1

Hex: 指令码 指令码

ANL C,/bit51

指令操作: (PC) ←(PC) + 2

(B) ←(C) and /(bit51)

影响标志位:

CY	AC	OV	N	Z
✓	-	-	-	-

[指令码] B0H

1 0 1 1	0 0 0 0	位地址

Binary 模式 **Source 模式**

指令长度: 2 2

时钟数: 1 1

Hex: 指令码 指令码

ANL C,bit

指令操作: (PC) ←(PC) + 3

(A) ←(C) and (bit)

影响标志位:

CY	AC	OV	N	Z
✓	-	-	-	-

[指令码] A98(y)H

1 0 1 0	1 0 0 1	1 0 0 0	0 y y y	直接地址

Binary 模式 **Source 模式**

指令长度: 4 3

时钟数: 2 1

Hex: [A5]指令码 指令码

ANL C,/bit

指令操作: (PC) ←(PC) + 3

(A) ←(C) and /(bit)

影响标志位:

CY	AC	OV	N	Z
✓	-	-	-	-

[指令码] A9F(y)H

1 0 1 0	1 0 0 1	1 1 1 1	0 y y y	直接地址

Binary 模式 **Source 模式**

指令长度: 4 3

时钟数: 2 1

Hex: [A5]指令码 指令码

CJNE <dest-byte>,<src-byte>,rel

功能: 比较，如果不相等就跳转。

说明: CJNE 比较前两个操作数的大小，如果它们的值不相等则跳转。将 PC 加上指令最后一个字节中的有符号相对偏移来计算跳转目标，之后递增 PC 到下一条指令的开头。如果无符号整数 <dest-byte> 的值小于无符号整数 <src-byte> 的值，则进位标志位置位；否则进位清零。两个操作数都不受影响。前两个操作数允许四种寻址模式组合：累加器可以与任何直接寻址的字节或者立即数比较，任何间接内存位置或者工作寄存器都可以与立即数比较。影响 C、N 和 Z 标志位。

CJNE A,Direct,rel

指令操作:

	(PC)	←(PC) + 3
if	(A)	≠ (direct)
then	(PC)	←(PC) + 相对偏移
if	(A)	< (direct)
then	(C)	←1
else	(A)	←0

影响标志位:

CY	AC	OV	N	Z
✓	-	-	✓	✓

[指令码] B5H

1	0	1	1	0	1	0	1	直接地址	相对地址

Binary 模式 **Source 模式**

指令长度: 3 3

时钟数: 2/3 2/3

Hex: 指令码 指令码

CJNE A,#DATA,rel

指令操作:

	(PC)	←(PC) + 3
if	(A)	≠ data
then	(PC)	←(PC) + 相对偏移
if	(A)	< data
then	(C)	←1
else	(C)	←0

影响标志位:

CY	AC	OV	N	Z
✓	-	-	✓	✓

[指令码] B4H

1	0	1	1	0	1	0	0	立即数	相对地址

Binary 模式 **Source 模式**

指令长度: 3 3

时钟数: 1/3 1/3

Hex: 指令码 指令码

CJNE Rn,#DATA,rel

指令操作:

	(PC)	←(PC) + 3
if	(Rn)	≠ data
then	(PC)	←(PC) + 相对偏移
if	(Rn)	< data
then	(C)	←1
else	(C)	←0

影响标志位:

CY	AC	OV	N	Z

✓	-	-	✓	✓

[指令码] B8H - BFH

1	0	1	1	1	r	r	r	立即数		相对地址	

Binary 模式　**Source 模式**

指令长度: 3　4

时钟数: 2/3　3/4

Hex: 指令码　[A5]指令码

CJNE @Ri,#DATA,rel

指令操作:

(PC) ←(PC) + 3
if ((Ri)) ≠ data
then (PC) ←(PC) + 相对偏移
if ((Ri)) < data
then (C) ←1
else (C) ←0

影响标志位:

CY	AC	OV	N	Z
✓	-	-	✓	✓

[指令码] B6H, B7H

1	0	1	1	0	1	1	i	立即数		相对地址	

Binary 模式　**Source 模式**

指令长度: 3　4

时钟数: 2/3　3/4

Hex: 指令码　[A5]指令码

CLR A

功能: 累计器清零

说明: 累加器被清零（所有的位置位为零）。影响 N 和 Z 标志位。

指令操作: (PC) ←(PC) + 1
(A) ←0

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] E4H

1	1	1	0	0	1	0	0

Binary 模式　**Source 模式**

指令长度: 1　1

时钟数: 1　1

Hex: 指令码　指令码

CLR bit51

功能: 位清零

说明: 指定的位被清零（重置为零）。不影响其它标志位。

指令操作: (PC) ←(PC) + 2
bit ←0

影响标志位:

CY	AC	OV	N	Z
-	-	-	–	–

[指令码] C2H

1	1	0	0	0	0	1	0	位地址

	Binary 模式	Source 模式
指令长度:	2	2
时钟数:	1	1
Hex:	指令码	指令码

CLR C

功能: 进位清零

说明: 进位标志位被清零（重置为零）。不影响其它标志位。

指令操作: (PC) ←(PC) + 1

(C) ←0

影响标志位:

CY	AC	OV	N	Z
✓	-	-	–	–

[指令码] C3H

1	1	0	0	0	0	1	1

	Binary 模式	Source 模式
指令长度:	1	1
时钟数:	1	1
Hex:	指令码	指令码

CLR bit

功能: 位清零

说明: 指定的位被清零（重置为零）。不影响其它标志位。

指令操作: (PC) ←(PC) + 3

(bit) ←0

影响标志位:

CY	AC	OV	N	Z
-	-	-	–	–

[指令码] A9C(y)H

1	1	0	0	1	0	0	1	1	1	0	0	0	y	y	y	直接地址

	Binary 模式	Source 模式
指令长度:	4	3
时钟数:	2	1
Hex:	[A5]指令码	指令码

CMP <dest>,<src>

功能: 比较

说明: 从目标操作数中减去源操作数。结果不存储在目标操作数中。如果第 7 位需要借位，则 CY（借位）标志位置位；否则清零。当减去有符号整数时，OV 标志位指示正数减去负数出现了负结果，或者指示负数减去正数出现了正结果。本描述中的第 7 位是指操作数的最高有效字节（8、16 或者 32 位）。AC 仅受字节操作数影响。N 和 Z 标志位会根据结果受到影响。源操作数允许四种寻址模式：寄存器、直接、立即数和间接寻址。

CMP Rmd,Rms

指令操作: (PC) ←(PC) + 2

(Rmd) -(Rms)

影响标志位:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[指令码] BCH

1	0	1	1	1	1	0	0	s	s	s	s	S	S	S	S

Binary 模式　**Source 模式**

指令长度: 3　2

时钟数: 2　1

Hex: [A5]指令码　指令码

CMP WRjd,WRjs

指令操作: (PC)　←(PC) + 2

(WRjd)　-(WRjs)

影响标志位:

CY	AC	OV	N	Z
✓	–	✓	✓	✓

[指令码] BDH

1	0	1	1	1	1	0	1	t	t	t	t	T	T	T	T

Binary 模式　**Source 模式**

指令长度: 3　2

时钟数: 2　1

Hex: [A5]指令码　指令码

CMP DRkd,DRks

指令操作: (PC)　←(PC) + 2

(DRkd)　-(DRks)

影响标志位:

CY	AC	OV	N	Z
✓	–	✓	✓	✓

[指令码] BFH

1	0	1	1	1	1	1	1	u	u	u	u	U	U	U	U

Binary 模式　**Source 模式**

指令长度: 3　2

时钟数: 2　1

Hex: [A5]指令码　指令码

CMP Rm,#DATA

指令操作: (PC)　←(PC) + 3

(Rm)　-#data

影响标志位:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[指令码] BE(s)0H

1	0	1	1	1	1	1	0	s	s	s	s	0	0	0	0	立即数

Binary 模式　**Source 模式**

指令长度: 4　3

时钟数: 2　1

Hex: [A5]指令码　指令码

CMP WRj,#DATA16

指令操作: (PC)　←(PC) + 4

(WRj)　-#data16

影响标志位:

CY	AC	OV	N	Z
✓	–	✓	✓	✓

[指令码] BE(t)4H

1	0	1	1	1	1	1	0	t	t	t	t	0	1	0	0	立即数高字节	立即数低字节

Binary 模式　**Source 模式**

指令长度: 5　4

时钟数: 2　1

Hex: [A5]指令码　指令码

CMP DRk,#0DATA16

指令操作: (PC)　←(PC) + 4

(DRk)　-#0data16

影响标志位:

CY	AC	OV	N	Z
✓	–	✓	✓	✓

[指令码] BE(u)8H

1	0	1	1	1	1	1	0	u	u	u	u	1	0	0	0	立即数高字节	立即数低字节

Binary 模式　**Source 模式**

指令长度: 5　4

时钟数: 2　1

Hex: [A5]指令码　指令码

CMP DRk,#1DATA16

指令操作: (PC)　←(PC) + 4

(DRk)　-#1data16

影响标志位:

CY	AC	OV	N	Z
✓	–	✓	✓	✓

[指令码] BE(u)CH

1	0	1	1	1	1	1	0	u	u	u	u	1	1	0	0	立即数高字节	立即数低字节

Binary 模式　**Source 模式**

指令长度: 5　4

时钟数: 2　1

Hex: [A5]指令码　指令码

CMP Rm,Dir8

指令操作: (PC)　←(PC) + 3

(Rm)　-(dir8)

影响标志位:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[指令码] BE(s)1H

1	0	1	1	1	1	1	0	s	s	s	s	0	0	0	1	直接地址

Binary 模式　**Source 模式**

指令长度: 4　3

时钟数: 2　1

Hex: [A5]指令码　指令码

CMP WRj,Dir8

指令操作: (PC) ←(PC) + 3

(WRj) -(dir8)

影响标志位:

CY	AC	OV	N	Z
✓	–	✓	✓	✓

[指令码] BE(t)5H

1 0 1 1	1 1 1 0	t t t t	0 1 0 1	直接地址

Binary 模式 **Source 模式**

指令长度: 4 3

时钟数: 2(3 for SFR) 1(2 for SFR)

Hex: [A5]指令码 指令码

CMP Rm,Dir16

指令操作: (PC) ←(PC) + 4

(Rm) -(dir16)

影响标志位:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[指令码] BE(s)3H

1 0 1 1	1 1 1 0	s s s s	0 0 1 1	地址高字节	地址低字节

Binary 模式 **Source 模式**

指令长度: 5 4

时钟数: 2 1

Hex: [A5]指令码 指令码

CMP WRj,Dir16

指令操作: (PC) ←(PC) + 4

(WRj) -(dir16)

影响标志位:

CY	AC	OV	N	Z
✓	–	✓	✓	✓

[指令码] BE(t)7H

1 0 1 1	1 1 1 0	t t t t	0 1 1 1	地址高字节	地址低字节

Binary 模式 **Source 模式**

指令长度: 4 3

时钟数: 2 1

Hex: [A5]指令码 指令码

CMP Rm,@WRj

指令操作: (PC) ←(PC) + 3

(Rm) -((WRj))

影响标志位:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[指令码] BE(t)9(s)0H

1 0 1 1	1 1 1 0	t t t t	1 0 0 1	s s s s	0 0 0 0

BE(WRj)9(Rm)0

Binary 模式 **Source 模式**

指令长度: 4 3

时钟数: 2 1

Hex: [A5]指令码　指令码

CMP Rm,@DRk

指令操作: (PC) ←(PC) + 3

(Rm) -((DRk))

影响标志位:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[指令码] BE(u)B(s)0H

1 0 1 1	1 1 1 0	u u u u	1 0 1 1	s s s s	0 0 0 0

BE (DRk) B (Rm) 0

Binary 模式 **Source 模式**

指令长度: 4　3

时钟数: 2　1

Hex: [A5]指令码　指令码

CPL A

功能: 累加器取反

说明: 累加器的每一位都逻辑取反（逐个取反）。先前为 1 的位将改变为 0，反之亦然。只影响 N 和 Z 标志位。

指令操作: (PC) ←(PC) + 1

(A) ←/(A)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] F4H

1 1 1 1	0 1 0 0

Binary 模式 **Source 模式**

指令长度: 1　1

时钟数: 1　1

Hex: 指令码　指令码

CPL Bit51

功能: 位取反

说明: 对指定的位变量取反。之前是 1 的位被改变为零，反之亦然。不影响其它标志位。

注意：当该指令用于修改输出引脚时，用作原始数据的值将从输出数据锁存器中读取，而不是输入引脚。

指令操作: (PC) ←(PC) + 2

(bit51) ←/(bit51)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] B2H

1 0 1 1	0 0 1 0	位地址

Binary 模式 **Source 模式**

指令长度: 2　2

时钟数: 1　1

Hex: 指令码　指令码

CPL C

功能: 进位取反

说明: 对进位标志位取反。之前是 1 的位被改变为零，反之亦然。

指令操作:
(PC) ←(PC) + 1
(C) ←/(C)

影响标志位:

CY	AC	OV	N	Z
✓	-	-	-	-

[指令码] B3H

1	0	1	1	0	0	1	1

Binary 模式 **Source 模式**

指令长度: 1 1

时钟数: 1 1

Hex: 指令码 指令码

CPL Bit

功能: 位取反

说明: 对指定的位取反。

指令操作:
(PC) ←(PC) + 3
(bit) ←/(bit)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] A9B(y)H

1	0	1	0	1	0	0	1	1	0	1	1	0	y	y	y	直接地址

Binary 模式 **Source 模式**

指令长度: 4 3

时钟数: 2 1

Hex: [A5]指令码 指令码

DA A

功能: 十进制加法后调整累加器

说明: DA A 调整累加器中的 8 位值，该值由之前两个变量（每个变量为压缩 BCD 格式）相加生成，生成两个 4 位数字。任何 ADD 或者 ADDC 指令都可能用来执行加法。如果累加器第 3 到 0 位大于 9（xxxx1010-xxxx1111），或者如果 AC 标志位为 1，则将累加器加 6 使得低位半字节生成正确的 BCD 数字。如果低四位域的进位通过所有高位传送，则此内部加法将置位进位标志位，否则却不会清零进位标志位。如果现在置位了进位标志位，或者如果现在四个高位超过九（1010xxxx-1111xxxx），这些高位将加六，在高半字节中生成正确的 BCD 数字。同样，如果高位进位，将会置位进位标志位，但不会清零进位。因此，进位标志位指示原始的两个 BCD 变量的和是否大于 100，从而允许多种准确的十进制加法。OV 不受影响。所有这些都发生在一个指令周期内。本质上，该指令通过将累加器加 00H、06H、60H 或者 66H 来执行十进制转换，具体取决于初始累加器和 PSW 条件。影响 C、N 和 Z 标志位。

注意：DA A 不能简单地将累加器中的十六进制数转换为 BCD 表示法，DA A 也不适用于十进制减法。

指令操作:

(PC)←(PC) + 3
if [[(A3-0) > 9] ^ [(AC) = 1]]
then (A3-0) ← (A3-0) + 6
next if [[(A7-4) > 9] ^ [(C) = 1]]

then　(A7-4) ← (A7-4) + 6

影响标志位:

CY	AC	OV	N	Z
✓	-	-	✓	✓

[指令码] B4H

1	0	1	1	0	1	0	0

Binary 模式　**Source 模式**

指令长度: 1　1

时钟数: 3　3

Hex: 指令码　指令码

DEC byte

功能: 递减

说明: 指定的变量减 1。00H 的原始值将下溢到 0FFH。只影响 N 和 Z 标志位。允许使用以下寻址模式：累加器、寄存器、直接或者寄存器间接寻址。

注意：当该指令用于修改输出端口时，用作原始端口数据的值将从输出数据锁存器中读取，而不是输入引脚。

DEC A

指令操作: (PC)　←(PC) + 1

(A)　←(A) - 1

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 14H

0	0	0	1	0	1	0	0

Binary 模式　**Source 模式**

指令长度: 1　1

时钟数: 1　1

Hex: 指令码　指令码

DEC Rn

指令操作: (PC)　←(PC) + 1

(Rn)　←(Rn) - 1

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 18H - 1FH

0	0	0	1	1	r	r	r

Binary 模式　**Source 模式**

指令长度: 1　2

时钟数: 1　2

Hex: 指令码　[A5]指令码

DEC Direct

指令操作: (PC)　←(PC) + 2

(direct)　←(direct) - 1

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 15H

0 0 0 1	0 1 0 1	直接地址

Binary 模式　**Source 模式**

指令长度: 2　2

时钟数: 1　1

Hex: 指令码　指令码

DEC @Ri

指令操作: (PC)　←(PC) + 1

((Ri))　←((Ri)) - 1

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 16H, 17H

0 0 0 1	0 1 1 i

Binary 模式　**Source 模式**

指令长度: 1　2

时钟数: 1　2

Hex: 指令码　[A5]指令码

DEC <dest>,<src>

功能: 目标值递减

说明: 将目标操作数所指定的变量递减 1、2 或者 4。原始值 00H 将下溢至 0FFH。影响 N 和 Z 标志位。

递减的值编码如下：

vv	值
00	1
01	2
10	4

DEC Rm,#short

指令操作: (PC)　←(PC) + 2

(Rm)　←(Rm) - #short

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 1B(s)(00v)H

0 0 0 1	1 0 1 1	s s s s	0 0 v v

Binary 模式　**Source 模式**

指令长度: 3　2

时钟数: 2　1

Hex: [A5]指令码　指令码

DEC WRj,#short

指令操作: (PC)　←(PC) + 2

(WRj)　←(WRj) - #short

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 1B(t)(01v)H

0 0 0 1	1 0 1 1	t t t t	0 1 v v

Binary 模式　**Source 模式**

指令长度: 3　2

时钟数: 2　1

Hex: [A5]指令码　指令码

DEC DRk,#short

指令操作: (PC)　←(PC) + 2

(DRk)　←(DRk) - #short

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 1B(u)(11v)H

0 0 0 1	1 0 1 1	u u u u	1 1 v v

Binary 模式　**Source 模式**

指令长度: 3　2

时钟数: 2　1

Hex: [A5]指令码　指令码

DIV AB

说明: DIV AB 将累加器中的无符号八位整数除以寄存器 B 中的无符号八位整数。累加器保存商的整数部分，寄存器 B 保存整数余数。进位和 OV 标志位将被清零。N 和 Z 标志位也会根据结果受到影响。

例外：如果 B 原来放入 00H，则返回到累加器和 B 寄存器的值为未定义的。此外，溢出标志位置位。进位标志位在任何情况下都会被清零。其它标志位未定义。

指令操作: (PC)　←(PC) + 1

(A15-8)　← (A) / (B) –结果的位 15..8

(A7-0)　← (A) / (B) –结果的位 7..0

影响标志位:

CY	AC	OV	N	Z
0	-	✓*	✓*	✓*

*) –如果 B 原来放入 00H，则 OV=1，N 和 Z 未定义

[指令码] 84H

1 0 0 0	0 1 0 0

Binary 模式　**Source 模式**

指令长度: 1　1

时钟数: 6　6

Hex: 指令码　指令码

DIV <dest>,<src>

说明: 在寄存器寻址模式下将寄存器中的无符号整数除以无符号整数操作数，并清零 CY 和 OV 标志位。对于字节操作数（<dest>, <src> = Rmd,Rms），结果为 16 位。8 位的商存储在 Rmd 所在字的高字节中，8 位的余数存储在 Rmd 所在字的低字节中。例如：寄存器 1 放入 251 （0FBH），寄存器 5 放入 18（12H）。执行 DIV R1 指令后，R5 寄存器 1 放入 13（0DH 或者 00001101B）；寄存器 0 放入 17（11H 或者 00010001B），因为 251= (13×18) + 17； 同时 CY 和 OV 位被清零。CY 标志位被清零。如果商的最高有效位已置位，则 N 标志位置位。如果商为零，则 Z 标志位置位。

例外：如果 <src> 放入 00H，则两个操作数中返回的值都是未定义的；CY 标志位被清零， OV

标志位置位，其余标志位未定义。

DIV Rmd,Rms

指令操作:

(PC)	←(PC) + 2	
(Rmd)	余数(Rmd) / (Rms)	if <dest> md = 0,2,4,..,14
(Rmd+1)	商(Rmd) / (Rms)	
(Rmd-1)	余数(Rmd) / (Rms)	if <dest> md = 1,3,5,..,15
(Rmd)	商(Rmd) / (Rms)	

影响标志位:

CY	AC	OV	N	Z
0	-	✓*	✓*	✓*

*) –如果源数原来放入 0，则 OV=1，N 和 Z 未定义

[指令码] 8CH

1	0	0	0	1	1	0	0	s	s	s	s	S	S	S	S

Binary 模式　**Source 模式**

指令长度: 3　2

时钟数: 7　6

Hex: [A5]指令码　指令码

DIV WRjd,WRjs

指令操作:

(PC)	←(PC) + 2	
(WRjd)	余数(WRjd) / (WRjs)	if <dest> jd = 0,4,8,..,28
(WRjd+1)	商(WRjd) / (WRjs)	
(WRjd-1)	余数(WRjd) / (WRjs)	if <dest> jd = 2,6,10,..,30
(WRjd)	商(WRjd) / (WRjs)	

对于字操作数（<dest>,<src> = WRjd,WRjs），16 位的商在 WR(jd+2)中，16 位的余数在 WRjd 中。例如，对于目标寄存器 WR4，假设商为 1122H，余数为 3344H。然后，将结果存储在这些寄存器文件位置：(4)->0x33、(5)->0x44、(6)->0x11、(7)->0x22。

影响标志位:

CY	AC	OV	N	Z
0	-	✓*	✓*	✓*

*) –如果源数原来放入 0，则 OV=1，N 和 Z 未定义

[指令码] 8DH

1	0	0	0	1	1	0	1	t	t	t	t	T	T	T	T

Binary 模式　**Source 模式**

指令长度: 3　2

时钟数: 11　10

Hex: [A5]指令码　指令码

DJNZ <byte>,<rel–addr>

功能: 递减，如果不为零则跳转

说明: DJNZ 将指定位置的值递减 1，如果结果值不为零，则跳转到第二个操作数指定的地址。 00H 的原始值将下溢至 0FFH。只影响 N 和 Z 标志位。将 PC 加上指令最后一个字节中的有符号相对偏移来计算跳转目标，之后将 PC 递增到下一条指令的第一个字节。递减的位置可以是寄存器或者直接寻址的字节。

注意：当该指令用于修改输出端口时，用作原始端口数据的值将从输出数据锁存器中读取，而不是输入引脚。

DJNZ Rn,rel

指令操作:

(PC)←(PC) + 2
(Rn)←(Rn) - 1
if (Rn) ≠ 0
then (PC)←(PC) + rel

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] D8H - DFH

1 1 0 1	1 r r r	相对地址

Binary 模式	**Source 模式**
2	3
2/3	3/4
指令码	[A5]指令码

指令长度:
时钟数:
Hex:

DJNZ Direct,rel

指令操作:

(PC)←(PC) + 3
(direct)←(direct) - 1
if (direct) ≠ 0
then (PC)←(PC) + rel

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] D5H

1 1 0 1	0 1 0 1	直接地址	相对地址

Binary 模式	**Source 模式**
3	4
2/3	2/3
指令码	[A5]指令码

指令长度:
时钟数:
Hex:

ECALL <dest>

功能: 扩展调用

说明: 调用位于指定地址的子程序。该指令将程序计数器加 4 以生成下一条指令的地址，然后将 24 位结果压入堆栈（高字节在前），堆栈指针增加 3。然后将 PC 的 8 位高字和 16 位低字分别装载到 ECALL 指令的第二、第三和第四字节。程序继续执行该地址处的指令。因此，子程序可以在整个 16MB 存储空间中的任何位置开始。不影响标志位。

ECALL addr24

指令操作:

(PC) ←(PC) + 4
(SP) ←(SP) + 1
((SP)) ←(PC.23:16)
(SP) ←(SP) + 1
((SP)) ←(PC.15:8)
(SP) ←(SP) + 1
((SP)) ←(PC.7:0)
(PC) ←(addr.23:0)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 9AH

1	0	0	1	1	0	1	0	addr 23-16	addr 15-8	addr 7-0

Binary 模式　**Source 模式**

指令长度: 5　4

时钟数: 4　3

Hex: [A5]指令码　指令码

ECALL @DRk

指令操作:
(PC) ←(PC) + 4
(SP) ←(SP) + 1
((SP)) ←(PC.23:16)
(SP) ←(SP) + 1
((SP)) ←(PC.15:8)
(SP) ←(SP) + 1
((SP)) ←(PC.7:0)
(PC) ←((DRk))

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 99(u)8H

1	0	0	1	1	0	0	1	u	u	u	u	1	0	0	0

Binary 模式　**Source 模式**

指令长度: 3　2

时钟数: 4　3

Hex: [A5]指令码　指令码

EJMP <dest>

功能: 扩展跳转

说明: 通过把指令的第二、第三和第四个字节加载到 PC 的 8 位高位字和 16 位低位字，进行无条件跳转到指定地址。因此，目标可以在整个 16MB 内存空间中的任何位置。不影响标志位。

EJMP addr24

指令操作: (PC) ←(addr.23:0)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 8AH

1	0	0	0	1	0	1	0	addr 23-16	addr 15-8	addr 7-0

Binary 模式　**Source 模式**

指令长度: 5　4

时钟数: 4　3

Hex: [A5]指令码　指令码

EJMP @DRk

指令操作: (PC) ←((DRk))

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 89(u)8H

1 0 0 0	1 0 0 1	u u u u	1 0 0 0

Binary 模式　**Source 模式**

指令长度: 3　2

时钟数: 4　3

Hex: [A5]指令码　指令码

ESC

功能: 切换到相反的模式

说明: 以相反的模式继续执行紧接着的指令。除了 PC 之外，没有寄存器或者标志位受到影响。

指令操作: (PC)　←(PC) + 1

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] A5H

1 0 1 0	0 1 0 1

Binary 模式　**Source 模式**

指令长度: 1　1

时钟数: 1　1

Hex: 指令码　指令码

ERET

功能: 扩展返回

说明: 依次从堆栈中弹出 PC 的第 3 字节、第 2 字节、第 1 字节和第 0 字节，并将堆栈指针减 3。程序在结果地址处继续执行，通常是马上紧接着 ECALL 之后的指令。不影响标志位。

指令操作: (PC.7:0)　¨((SP))

(SP)　¨(SP) - 1

(PC.15:8)　¨((SP))

(SP)　¨(SP) - 1

(PC.23:16)　←((SP))

(SP)　←(SP) - 1

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] AAH

1 0 1 0	1 0 1 0

Binary 模式　**Source 模式**

指令长度: 2　1

时钟数: 4　3

Hex: [A5]指令码　指令码

INC byte

功能: 递增

说明: INC 将指定的变量加 1。原来值 0FFh 将溢出到 00h。只影响 N 和 Z 标志位。允许三种寻址模式：寄存器、直接或者寄存器间接寻址。

注意：当该指令用于修改输出端口时，用作原始端口数据的值将从输出数据锁存器中读取，而不是输入引脚。

INC A

指令操作: (PC) ←(PC) + 1

(A) ←(A) + 1

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 04H

0	0	0	0	0	1	0	0

Binary 模式 **Source 模式**

指令长度: 1 1

时钟数: 1 1

Hex: 指令码 指令码

INC Rn

指令操作: (PC) ←(PC) + 1

(Rn) ←(Rn) + 1

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 08H - 0FH

0	0	0	0	1	r	r	r

Binary 模式 **Source 模式**

指令长度: 1 2

时钟数: 1 2

Hex: 指令码 [A5]指令码

INC Direct

指令操作: (PC) ←(PC) + 1

(direct) ←(direct) + 1

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 05H

0	0	0	0	0	1	0	1	直接地址

Binary 模式 **Source 模式**

指令长度: 2 2

时钟数: 1 1

Hex: 指令码 指令码

INC @Ri

指令操作: (PC) ←(PC) + 1

((Ri)) ←((Ri)) + 1

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 06H, 07H

0	0	0	0	0	1	1	i

Binary 模式 **Source 模式**

指令长度: 1 2

时钟数: 1 2

Hex: 指令码 [A5]指令码

INC <dest>,<src>

功能: 递增

说明: 将指定变量递增 1、2 或者 4。原来值 0FFH 溢出到 00H。只影响 N 和 Z 标志位。递增值的编码如下：

vv	值
00	1
01	2
10	4

INC Rm,#short

指令操作: (PC) ←(PC) + 2

(Rm) ←(Rm) + #short

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 0B(s)(00v)H

0	0	0	0	1	0	1	1	s	s	s	s	0	0	v	v

Binary 模式 **Source 模式**

指令长度: 3 2

时钟数: 2 1

Hex: [A5]指令码 指令码

INC WRj,#short

指令操作: (PC) ←(PC) + 2

(WRj) ←(WRj) + #short

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 0B(t)(01v)H

0	0	0	0	1	0	1	1	t	t	t	t	0	1	v	v

Binary 模式 **Source 模式**

指令长度: 3 2

时钟数: 2 1

Hex: [A5]指令码 指令码

INC DRk,#short

指令操作: (PC) ←(PC) + 2

(DRk) ←(DRk) + #short

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 0B(u)(11v)H

0	0	0	0	1	0	1	1	u	u	u	u	1	1	v	v

Binary 模式 **Source 模式**

指令长度: 3 2

时钟数: 2 1

Hex: [A5]指令码 指令码

INC DPTR

功能: 递增数据指针

说明: 将 16 位的数据指针加 1。执行 16 位递增（模 2^16）； 数据指针的低字节（DPL）从 0FFH 溢出到 00H 时会向高位字节（DPH）递增。高字节（DPH）溢出不会增加扩展数据指针的高字（DPX = DR56）。只影响 N 和 Z 标志位。

指令操作: (PC) ←(PC) + 1

(DPTR) ←(DPTR) + 1

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] A3H

1	0	1	0	0	0	1	1

Binary 模式 **Source 模式**

指令长度: 1 1

时钟数: 1 1

Hex: 指令码 指令码

JB

功能: 如果置位则跳转

说明: 如果指定的位为 1，则跳转到指定的地址；否则继续下一条指令。将 PC 加上指令第三个字节中的有符号相对偏移来计算跳转目标，之后将 PC 递增到下一条指令的第一个字节。不修改被检测的位。不影响标志位。

JB Bit51,rel

指令操作: (PC)←(PC) + 3

if (bit51) = 1

then (PC)←(PC) + rel

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 20H

0	0	1	0	0	0	0	0	位地址	相对地址

Binary 模式 **Source 模式**

指令长度: 3 3

时钟数: 1/3 1/3

Hex: 指令码 指令码

JB Bit,rel

指令操作: (PC)←(PC) + 3

if (bit) = 1

then (PC)←(PC) + rel

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] A92(y)H

1	0	1	0	1	0	0	1	0	0	1	0	0	y	y	y	直接地址	相对地址

Binary 模式 **Source 模式**

指令长度: 5 4

时钟数: 2/5 1/3

Hex: [A5]指令码　指令码

JBC

功能: 如果置位则跳转并清零位

说明: 如果指定的位为 1，则跳转到指定的地址；否则继续下一条指令。任何一种情况，都清零指定位。将 PC 加上指令第三个字节中的有符号相对偏移来计算跳转目标，之后将 PC 递增到下一条指令的第一个字节。不影响标志位。

注意：当该指令用于测试输出引脚时，用作原始数据的值将从输出数据锁存器中读取，而不是输入引脚。

JBC Bit51,rel

指令操作:

(PC)←(PC) + 3

if (bit51) = 1

then (PC)←(PC) + rel

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 10H

0 0 0 1	0 0 0 0	位地址	相对地址

Binary 模式　**Source 模式**

指令长度: 3　3

时钟数: 1/3　1/3

Hex: 指令码　指令码

JBC Bit,rel

指令操作:

(PC)←(PC) + 3

if (bit) = 1

then (PC)←(PC) + rel

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] A91(y)H

1 0 1 0	1 0 0 1	0 0 0 1	0 y y y	直接地址	相对地址

Binary 模式　**Source 模式**

指令长度: 5　4

时钟数: 2/5　1/3

Hex: [A5]指令码　指令码

JC

功能: 如果置位了进位则跳转

说明: 如果进位标志位被置位，则跳转到指定的地址；否则继续下一条指令。将 PC 加上指令第二个字节中的有符号相对偏移来计算跳转目标，之后将 PC 递增两次。不修改被检测的位。不影响标志位。

指令操作:

(PC)←(PC) + 2

if (C) = 1

then (PC)←(PC) + rel

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 40H

0 1 0 0	0 0 0 0	相对地址

Binary 模式　**Source 模式**

指令长度: 2　2

时钟数: 1/3　1/3

Hex: 指令码　指令码

JE

功能: 如果相等则跳转

说明: 如果 Z 标志位已置位，则跳转到指定的地址；否则继续下一条指令。将 PC 加上指令第二个字节中的有符号相对偏移来计算跳转目标，之后将 PC 递增两次。

指令操作:

(PC)←(PC) + 2

if (Z) = 1

then (PC)←(PC) + rel

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 68H

0 1 1 0	1 0 0 0	相对地址

Binary 模式　**Source 模式**

指令长度: 3　2

时钟数: 2/5　1/3

Hex: [A5]指令码　指令码

JG

功能: 如果大于则跳转

说明: 如果 Z 标志位和 CY 标志位都清零，则跳转到指定的地址；否则继续下一条指令。将 PC 加上指令第二个字节中的有符号相对偏移来计算跳转目标，之后将 PC 递增两次。

指令操作:

(PC)←(PC) + 2

if (Z=0 and C=0) = 1

then (PC)←(PC) + rel

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 38H

0 0 1 1	1 0 0 0	相对地址

Binary 模式　**Source 模式**

指令长度: 3　2

时钟数: 2/5　1/3

Hex: [A5]指令码　指令码

JLE

功能: 如果小于等于则跳转

说明: 如果 Z 标志位和 CY 标志位都已置位，则跳转到指定的地址；否则继续下一条指令。将 PC 加上指令第二个字节中的有符号相对偏移来计算跳转目标，之后将 PC 递增两次。

指令操作:

(PC)←(PC) + 2

if (Z=1 and C=1) = 1

then (PC)←(PC) + rel

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 28H

0 0 1 0	1 0 0 0	相对地址

Binary 模式　**Source 模式**

指令长度: 3　2

时钟数: 2/5　1/3

Hex: [A5]指令码　指令码

JMP @A+DPTR

功能: 间接跳转

说明: 把累加器的 8 位无符号内容和 16 位数据指针相加，并将结果和加载到程序计数器。这将是后续指令提取的地址。执行 16 位数加法（模 2^16）：低 8 位的进位传送到高位。累加器和数据指针都没有改变。不影响标志位。

指令操作: (PC)　←(A) + (DPTR)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 73H

0 1 1 1	0 0 1 1

Binary 模式　**Source 模式**

指令长度: 1　1

时钟数: 4　4

Hex: 指令码　指令码

JNB

功能: 如果位未置位则跳转

说明: 如果指定的位为零，则跳转到指定的地址；否则继续下一条指令。将 PC 加上指令第三个字节中的有符号相对偏移来计算跳转目标，之后将 PC 递增到下一条指令的第一个字节。不修改被检测的位。不影响标志位。

JNB Bit51,rel

指令操作:　(PC)←(PC) + 3

if　(bit51) = 0

then　(PC)←(PC) + rel

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 30H

0 0 1 1	0 0 0 0	位地址	相对地址

Binary 模式　**Source 模式**

指令长度: 3　3

时钟数: 1/3　1/3

Hex: 指令码　指令码

JNB Bit,rel

指令操作:　(PC)←(PC) + 3

if　(bit) = 0

then　(PC)←(PC) + rel

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] A93(y)H

1 0 1 0	1 0 0 1	0 0 1 1	0 y y y	直接地址	相对地址

Binary 模式　**Source 模式**

指令长度: 5　4

时钟数: 2/5　1/3

Hex: [A5]指令码　指令码

JNC

功能: 如果进位未置位则跳转

说明: 如果进位标志位为零，则跳转到指定的地址；否则继续下一条指令。将 PC 加上指令第二个字节中的有符号相对偏移来计算跳转目标，之后将 PC 递增到下一条指令。

指令操作:

(PC)←(PC) + 2

if　(C) = 0

then　(PC)←(PC) + rel

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 50H

0 1 0 1	0 0 0 0	相对地址

Binary 模式　**Source 模式**

指令长度: 2　2

时钟数: 1/3　1/3

Hex: 指令码　指令码

JNE

功能: 如果不相等则跳转

说明: 如果 Z 标志位清零，则跳转到指定的地址；否则继续下一条指令。将 PC 加上指令第二个字节中的有符号相对偏移来计算跳转目标，之后将 PC 递增两次。

指令操作:

(PC)←(PC) + 2

if　(Z) = 0

then　(PC)←(PC) + rel

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 78H

0 1 1 1	1 0 0 0	相对地址

Binary 模式　**Source 模式**

指令长度: 3　2

时钟数: 2/5　1/3

Hex: [A5]指令码　指令码

JNZ

功能: 如果累加器不为零则跳转

说明: 如果累加器的任何一位为 1，则跳转到指定的地址；否则继续下一条指令。将 PC 加上指令第二个字节中的有符号相对偏移来计算跳转目标，之后将 PC 递增两次。

指令操作:

(PC)←(PC) + 2
if (A) ≠ 0
then (PC)←(PC) + rel

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 70H

0	1	1	1	0	0	0	0	相对地址

Binary 模式　**Source 模式**

指令长度: 2　2

时钟数: 1/3　1/3

Hex: 指令码　指令码

JSG

功能: 如果大于则跳转（有符号）

说明: 如果 Z 标志位清零并且 N 标志位和 OV 标志位具有相同的值，则跳转到指定的地址；否则继续下一条指令。将 PC 加上指令第二个字节中的有符号相对偏移来计算跳转目标，之后将 PC 递增两次。

指令操作:

(PC)←(PC) + 2
if (Z=0 and N=OV)
then (PC)←(PC) + rel

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 18H

0	0	0	1	1	0	0	0	相对地址

Binary 模式　**Source 模式**

指令长度: 3　2

时钟数: 2/5　1/3

Hex: [A5]指令码　指令码

JSGE

功能: 大于等于跳转（有符号）

说明: 如果 N 标志位和 OV 标志位具有相同的值，则跳转到指定的地址；否则继续下一条指令。将 PC 加上指令第二个字节中的有符号相对偏移来计算跳转目标，之后将 PC 递增两次。

指令操作:

(PC)←(PC) + 2
if (N=OV)
then (PC)←(PC) + rel

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 58H

0	1	0	1	1	0	0	0	相对地址

Binary 模式　**Source 模式**

指令长度: 3　2

时钟数: 2/5　1/3

Hex: [A5]指令码　指令码

JSL

功能: 如果小于则跳转（有符号）

说明: 如果 N 标志位和 OV 标志位具有不同的值，则跳转到指定的地址；否则继续下一条指令。将 PC 加上指令第二个字节中的有符号相对偏移来计算跳转目标，之后将 PC 递增两次。

指令操作:

(PC)←(PC) + 2
if (N≠OV)
then (PC)←(PC) + rel

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 48H

0 1 0 0	1 0 0 0	相对地址

Binary 模式 **Source 模式**

指令长度: 3 2

时钟数: 2/5 1/3

Hex: [A5]指令码 指令码

JSLE

功能: 如果小于等于则跳转（有符号）

说明: 如果 Z 标志位已置位或者如果 N 标志位和 OV 标志位具有不同的值，则跳转到指定的地址；否则继续下一条指令。将 PC 加上指令第二个字节中的有符号相对偏移来计算跳转目标，之后将 PC 递增两次。

指令操作:

(PC)←(PC) + 2
if (Z=1 or (N≠OV))
then (PC)←(PC) + rel

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 08H

0 0 0 0	1 0 0 0	相对地址

Binary 模式 **Source 模式**

指令长度: 3 2

时钟数: 2/5 1/3

Hex: [A5]指令码 指令码

JZ

功能: 如果累加器为零则跳转

说明: 如果累加器的所有位都为零，则跳转到指定的地址；否则继续下一条指令。将 PC 加上指令第二个字节中的有符号相对偏移来计算跳转目标，之后将 PC 递增两次。不修改累加器。 不影响标志位。

指令操作:

(PC)←(PC) + 2
if (A) = 0
then (PC)←(PC) + rel

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 60H

0 1 1 0	0 0 0 0	相对地址

Binary 模式 **Source 模式**

指令长度: 2 2

时钟数: 1/3　1/3

Hex: 指令码　指令码

LCALL

功能: 长调用

说明: 调用位于指定地址的子程序。该指令将程序计数器加 3 以生成下一条指令的地址，然后将 16 位结果压入堆栈（先是低字节），堆栈指针加 2。然后将 PC 的高位和低位字节分别装载到 LCALL 指令的第二和第三字节。程序继续执行该地址处的指令。因此，子程序可以在整个 64KB 程序存储器地址空间中的任何位置开始。不影响标志位。

LCALL addr16

指令操作:

(PC) ←(PC) + 3
(SP) ←(SP) + 1
((SP)) ←(PC.7:0)
(SP) ←(SP) + 1
((SP)) ←(PC.15:8)
(PC) ←(addr.15:0)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 12H

0 0 0 1	0 0 1 0	地址高字节	地址低字节

Binary 模式　**Source 模式**

指令长度: 3　3

时钟数: 3　3

Hex: 指令码　指令码

LCALL @WRj

指令操作:

(PC) ←(PC) + 3
(SP) ←(SP) + 1
((SP)) ←(PC.7:0)
(SP) ←(SP) + 1
((SP)) ←(PC.15:8)
(PC) ←((WRj))

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 99(t)4H

1 0 0 1	1 0 0 1	t t t t	0 1 0 0

Binary 模式　**Source 模式**

指令长度: 3　2

时钟数: 4　3

Hex: [A5]指令码　指令码

LJMP

功能: 长跳转

说明: LJMP 通过把指令的第二和第三个字节分别加载到 PC 高位和低位字节，进行无条件跳转到指定地址。因此，目标可以在整个 64KB 程序存储器地址空间中的任何位置。不影响标志位。

LJMP addr16

指令操作: (PC) ←(addr.15:0)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 02H

0 0 0 0	0 0 1 0	地址高字节	地址低字节

Binary 模式 **Source 模式**

指令长度: 3 3

时钟数: 3 3

Hex: 指令码 指令码

LJMP @WRj

指令操作: (PC) ←((WRj))

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 89(t)4H

1 0 0 0	1 0 0 1	t t t t	0 1 0 0

Binary 模式 **Source 模式**

指令长度: 3 2

时钟数: 4 3

Hex: [A5]指令码 指令码

MOV

功能: 搬运变量

说明: 第二个操作数指定的变量被复制到第一个操作数指定的位置。源数字节不受影响。没有其它寄存器或者标志位受到影响。这是迄今为止最灵活的操作。允许源数和目标寻址模式的 24 种组合。

MOV A,Rn

指令操作: (PC) ←(PC) + 1

(A) ←(Rn)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] E8H - EFH

1 1 1 0	1 r r r

Binary 模式 **Source 模式**

指令长度: 1 2

时钟数: 1 2

Hex: 指令码 [A5]指令码

MOV A,Direct

指令操作: (PC) ←(PC) + 2

(A) ←(direct)

注：MOV A, ACC 是有效指令。

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] E5H

1 1 1 0	0 1 0 1	直接地址

Binary 模式 **Source 模式**

指令长度: 2 2

时钟数: 1 1

Hex: 指令码 指令码

MOV A,@Ri

指令操作: (PC) ←(PC) + 1

(A) ←((Ri))

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] E6H, E7H

1 1 1 0	0 1 1 i

Binary 模式 **Source 模式**

指令长度: 1 2

时钟数: 1 2

Hex: 指令码 [A5]指令码

MOV A,#DATA

指令操作: (PC) ←(PC) + 2

(A) ←#data

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 74H

0 1 1 1	0 1 0 0	立即数

Binary 模式 **Source 模式**

指令长度: 2 2

时钟数: 1 1

Hex: 指令码 指令码

MOV Rn,A

指令操作: (PC) ←(PC) + 1

(Rn) ←(A)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] F8H - FFH

1 1 1 1	1 r r r

Binary 模式 **Source 模式**

指令长度: 1 2

时钟数: 1 2

Hex: 指令码 [A5]指令码

MOV Rn,Direct

指令操作: (PC) ←(PC) + 2

(Rn) ←(direct)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] A8H - AFH

1 0 1 0	1 r r r	直接地址

Binary 模式 **Source 模式**

指令长度: 2 3

时钟数: 1 1

Hex: 指令码 [A5]指令码

MOV Rn,#DATA

指令操作: (PC) ←(PC) + 2

(Rn) ←#data

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 78H - 7FH

0 1 1 1	1 r r r	立即数

Binary 模式 **Source 模式**

指令长度: 2 3

时钟数: 1 1

Hex: 指令码 [A5]指令码

MOV Direct,A

指令操作: (PC) ←(PC) + 2

(direct) ←(A)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] F5H

1 1 1 1	0 1 0 1	直接地址

Binary 模式 **Source 模式**

指令长度: 2 2

时钟数: 1 1

Hex: 指令码 指令码

MOV Direct,Rn

指令操作: (PC) ←(PC) + 2

(direct) ←(Rn)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 88H - 8FH

1 0 0 0	1 r r r	直接地址

Binary 模式 **Source 模式**

指令长度: 2 3

时钟数: 1 1

Hex: 指令码 [A5]指令码

MOV Direct,Direct

指令操作: (PC) ←(PC) + 3

(direct) ←(direct)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 85H

1 0 0 0	0 1 0 1	源直接地址	目标直接地址

Binary 模式 **Source 模式**

指令长度: 3 3

时钟数: 1 1

Hex: 指令码 指令码

MOV Direct,@Ri

指令操作: (PC) ←(PC) + 2

(direct) ←((Ri))

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 86H, 87H

1 0 0 0	0 1 1 i	直接地址

Binary 模式 **Source 模式**

指令长度: 2 3

时钟数: 1 2

Hex: 指令码 [A5]指令码

MOV Direct,#DATA

指令操作: (PC) ←(PC) + 2

(direct) ←#data

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 75H

0 1 1 1	0 1 0 1	直接地址	立即数

Binary 模式 **Source 模式**

指令长度: 3 3

时钟数: 1 1

Hex: 指令码 指令码

MOV @Ri,A

指令操作: (PC) ←(PC) + 1

((Ri)) ←(A)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] F6H, F7H

1 1 1 1	0 1 1 i

Binary 模式 **Source 模式**

指令长度: 1 2

时钟数: 1 2

Hex: 指令码 [A5]指令码

MOV @Ri,Direct

指令操作: (PC) ←(PC) + 2

((Ri)) ←(direct)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] A6H, A7H

1 0 1 0	0 1 1 i	直接地址

Binary 模式 **Source 模式**

指令长度: 2 3

时钟数: 1 2

Hex: 指令码 [A5]指令码

MOV @Ri,#DATA

指令操作: (PC) ←(PC) + 2

((Ri)) ←#data

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 76H, 77H

0 1 1 1	0 1 1 i	立即数

Binary 模式 **Source 模式**

指令长度: 2 3

时钟数: 1 2

Hex: 指令码 [A5]指令码

MOV Rmd,Rms

指令操作: (PC) ←(PC) + 2

(Rmd) ←(Rms)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 7CH

0 1 1 1	1 1 0 0	s s s s	S S S S

Binary 模式 **Source 模式**

指令长度: 3 2

时钟数: 2 1

Hex: [A5]指令码 指令码

MOV WRjd,WRjs

指令操作: (PC) ←(PC) + 2

(WRjd) ←(WRjs)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 7DH

0 1 1 1	1 1 0 1	t t t t	T T T T

Binary 模式 **Source 模式**

指令长度: 3 2

时钟数: 2　1

Hex: [A5]指令码　指令码

MOV DRkd,DRks

指令操作: (PC)　←(PC) + 2

(DRkd)　←(DRks)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 7FH

0 1 1 1	1 1 1 1	u u u u	U U U U

Binary 模式　**Source 模式**

指令长度: 3　2

时钟数: 2　1

Hex: [A5]指令码　指令码

MOV Rm,#DATA

指令操作: (PC)　←(PC) + 3

(Rm)　←#data

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 7E(s)0H

0 1 1 1	1 1 1 0	s s s s	0 0 0 0	立即数

Binary 模式　**Source 模式**

指令长度: 4　3

时钟数: 2　1

Hex: [A5]指令码　指令码

MOV WRj,#DATA16

指令操作: (PC)　←(PC) + 4

(WRj)　←#data16

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 7E(t)4H

0 1 1 1	1 1 1 0	t t t t	0 1 0 0	立即数高字节	立即数低字节

Binary 模式　**Source 模式**

指令长度: 5　4

时钟数: 2　1

Hex: [A5]指令码　指令码

MOV DRk,#0DATA16

指令操作: (PC)　←(PC) + 4

(DRk)　←#0data16

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 7E(u)8H

0 1 1 1	1 1 1 0	u u u u	1 0 0 0	立即数高字节	立即数低字节

	Binary 模式	Source 模式
指令长度:	5	4
时钟数:	2	1
Hex:	[A5]指令码	指令码

MOV DRk,#1DATA16

指令操作: (PC) ←(PC) + 4

(DRk) ←#1data16

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 7E(u)CH

0	1	1	1	1	1	1	0	u	u	u	u	1	1	0	0	立即数高字节		立即数低字节	

	Binary 模式	Source 模式
指令长度:	5	4
时钟数:	2	1
Hex:	[A5]指令码	指令码

MOV Rm,Dir8

指令操作: (PC) ←(PC) + 3

(Rm) ←(dir8)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 7E(s)1H

0	1	1	1	1	1	1	0	s	s	s	s	0	0	0	1	直接地址			

	Binary 模式	Source 模式
指令长度:	4	3
时钟数:	2	1
Hex:	[A5]指令码	指令码

MOV WRj,Dir8

指令操作: (PC) ←(PC) + 3

(WRj) ←(dir8)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 7E(t)5H

0	1	1	1	1	1	1	0	t	t	t	t	0	1	0	1	直接地址			

	Binary 模式	Source 模式
指令长度:	4	3
时钟数:	2（3 for SFR）	1（2 for SFR）
Hex:	[A5]指令码	指令码

MOV DRk,Dir8

指令操作: (PC) ←(PC) + 3

(DRk) ←(dir8)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 7E(u)DH

0 1 1 1	1 1 1 0	u u u u	1 1 0 1	直接地址

Binary 模式　**Source 模式**

指令长度: 4　3

时钟数: 2(5 for SFR)　1(4 for SFR)

Hex: [A5]指令码　指令码

MOV Rm,Dir16

指令操作: (PC)　←(PC) + 4

(Rm)　←(dir16)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 7E(s)3H

0 1 1 1	1 1 1 0	s s s s	0 0 1 1	地址高字节	地址低字节

Binary 模式　**Source 模式**

指令长度: 5　4

时钟数: 2　1

Hex: [A5]指令码　指令码

MOV WRj,Dir16

指令操作: (PC)　←(PC) + 4

(WRj)　←(dir16)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 7E(t)7H

0 1 1 1	1 1 1 0	t t t t	0 1 1 1	地址高字节	地址低字节

Binary 模式　**Source 模式**

指令长度: 5　4

时钟数: 2　1

Hex: [A5]指令码　指令码

MOV DRk,Dir16

指令操作: (PC)　←(PC) + 4

(DRk)　←(dir16)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 7E(u)FH

0 1 1 1	1 1 1 0	u u u u	1 1 1 1	地址高字节	地址低字节

Binary 模式　**Source 模式**

指令长度: 5　4

时钟数: 2　1

Hex: [A5]指令码　指令码

MOV Rm,@WRj

指令操作: (PC)　←(PC) + 3

(Rm)　←((WRj))

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 7E(t)9(s)0H

0	1	1	1	1	1	1	0	t	t	t	t	1	0	0	1	s	s	s	s	0	0	0	0

Binary 模式　**Source 模式**

指令长度: 4　3

时钟数: 2　1

Hex: [A5]指令码　指令码

MOV Rm,@DRk

指令操作: (PC)　←(PC) + 3

(Rm)　←((DRk))

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 7E(u)B(s)0H

0	1	1	1	1	1	1	0	u	u	u	u	1	0	1	1	s	s	s	s	0	0	0	0

Binary 模式　**Source 模式**

指令长度: 4　3

时钟数: 2　1

Hex: [A5]指令码　指令码

MOV WRjd,@WRjs

指令操作: (PC)　←(PC) + 3

(WRjd)　←((WRjs))

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 0B(u)8(s)0H

0	0	0	0	1	0	1	1	u	u	u	u	1	0	0	0	s	s	s	s	0	0	0	0

Binary 模式　**Source 模式**

指令长度: 4　3

时钟数: 2　1

Hex: [A5]指令码　指令码

MOV WRj,@DRk

指令操作: (PC)　←(PC) + 3

(WRj)　←((DRk))

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 0B(u)A(t)0H

0	0	0	0	1	0	1	1	u	u	u	u	1	0	1	0	t	t	t	t	0	0	0	0

Binary 模式　**Source 模式**

指令长度: 4　3

时钟数: 2　1

Hex: [A5]指令码　指令码

MOV Dir8,Rm

指令操作: (PC) ←(PC) + 3

(dir8) ←(Rm)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 7A(s)1H

0	1	1	1	1	0	1	0	s	s	s	s	0	0	0	1	直接地址

Binary 模式 **Source 模式**

指令长度: 4 3

时钟数: 2 1

Hex: [A5]指令码 指令码

MOV Dir8,WRj

指令操作: (PC) ←(PC) + 3

(dir8) ←(WRj)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 7A(t)5H

0	1	1	1	1	0	1	0	t	t	t	t	0	1	0	1	直接地址

Binary 模式 **Source 模式**

指令长度: 4 3

时钟数: 2 1

Hex: [A5]指令码 指令码

MOV Dir8,DRk

指令操作: (PC) ←(PC) + 3

(dir8) ←(DRk)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 7A(u)DH

0	1	1	1	1	0	1	0	u	u	u	u	1	1	0	1	直接地址

Binary 模式 **Source 模式**

指令长度: 4 3

时钟数: 2 1

Hex: [A5]指令码 指令码

MOV Dir16,Rm

指令操作: (PC) ←(PC) + 4

(dir16) ←(Rm)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 7A(s)3H

0	1	1	1	1	0	1	0	s	s	s	s	0	0	1	1	地址高字节	地址低字节

Binary 模式 **Source 模式**

指令长度: 5 4

时钟数: 2 1

Hex: [A5]指令码 指令码

MOV Dir16,WRj

指令操作: (PC) ←(PC) + 4

(dir16) ←(WRj)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 7A(t)7H

0 1 1 1	1 0 1 0	t t t t	0 1 1 1	地址高字节	地址低字节

Binary 模式 **Source 模式**

指令长度: 5 4

时钟数: 2 1

Hex: [A5]指令码 指令码

MOV Dir16,DRk

指令操作: (PC) ←(PC) + 4

(dir16) ←(DRk)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 7A(u)FH

0 1 1 1	1 0 1 0	u u u u	1 1 1 1	地址高字节	地址低字节

Binary 模式 **Source 模式**

指令长度: 5 4

时钟数: 2 1

Hex: [A5]指令码 指令码

MOV @WRj,Rm

指令操作: (PC) ←(PC) + 3

((WRj)) ←(Rm)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 7A(t)9(s)0H

0 1 1 1	1 0 1 0	t t t t	1 0 0 1	s s s s	0 0 0 0

Binary 模式 **Source 模式**

指令长度: 4 3

时钟数: 2 1

Hex: [A5]指令码 指令码

MOV @DRk,Rm

指令操作: (PC) ←(PC) + 3

((DRk)) ←(Rm)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 7A(u)B(s)0H

0 1 1 1	1 0 1 0	u u u u	1 0 1 1	s s s s	0 0 0 0

Binary 模式 **Source 模式**

指令长度: 4 3

时钟数: 2　1

Hex: [A5]指令码　指令码

MOV @WRjd,WRjs

指令操作: (PC)　←(PC) + 3

((WRjd))　←(WRjs)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 1B(t)8(T)0H

0 0 0 1	1 0 1 1	t t t t	1 0 0 0	T T T T	0 0 0 0

Binary 模式　**Source 模式**

指令长度: 4　3

时钟数: 2　1

Hex: [A5]指令码　指令码

MOV @DRk,WRj

指令操作: (PC)　←(PC) + 3

((DRk))　←(WRj)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 1B(u)A(t)0H

0 0 0 1	1 0 1 1	u u u u	1 0 1 0	t t t t	0 0 0 0

Binary 模式　**Source 模式**

指令长度: 4　3

时钟数: 2　1

Hex: [A5]指令码　指令码

MOV Rm,@WRj+dis

指令操作: (PC)　←(PC) + 4

(Rm)　←((WRj)+dis)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 09H

0 0 0 0	1 0 0 1	s s s s	t t t t	偏移高字节	偏移低字节

Binary 模式　**Source 模式**

指令长度: 5　4

时钟数: 2　1

Hex: [A5]指令码　指令码

MOV WRjd,@WRjs+dis

指令操作: (PC)　←(PC) + 4

(WRjd)　←((WRjs)+dis)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 49H

0 1 0 0	1 0 0 1	t t t t	T T T T	偏移高字节	偏移低字节

	Binary 模式	Source 模式
指令长度:	5	4
时钟数:	2	1
Hex:	[A5]指令码	指令码

MOV Rm,@DRk+dis

指令操作: (PC) ←(PC) + 4

(Rm) ←((DRk)+dis)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 29H

0	0	1	0	1	0	0	1	s	s	s	s	u	u	u	u	偏移高字节	偏移低字节

	Binary 模式	Source 模式
指令长度:	5	4
时钟数:	2	1
Hex:	[A5]指令码	指令码

MOV WRj,@DRk+dis

指令操作: (PC) ←(PC) + 4

(WRj) ←((DRk)+dis)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 69H

0	1	1	0	1	0	0	1	t	t	t	t	u	u	u	u	偏移高字节	偏移低字节

	Binary 模式	Source 模式
指令长度:	5	4
时钟数:	2	1
Hex:	[A5]指令码	指令码

MOV @WRj+dis,Rm

指令操作: (PC) ←(PC) + 4

((WRj)+dis) ←(Rm)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 19H

0	0	0	1	1	0	0	1	s	s	s	s	t	t	t	t	偏移高字节	偏移低字节

	Binary 模式	Source 模式
指令长度:	5	4
时钟数:	2	1
Hex:	[A5]指令码	指令码

MOV @WRjd+dis,WRjs

指令操作: (PC) ←(PC) + 4

((WRjd)+dis) ←(WRjs)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 59H

0 1 0 1	1 0 0 1	T T T T	t t t t	偏移高字节	偏移低字节

	Binary 模式	**Source 模式**
指令长度:	5	4
时钟数:	2	1
Hex:	[A5]指令码	指令码

MOV @DRk+dis,Rm

指令操作: (PC) ←(PC) + 4

((DRk)+dis) ←(Rm)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 39H

0 0 1 1	1 0 0 1	s s s s	u u u u	偏移高字节	偏移低字节

	Binary 模式	**Source 模式**
指令长度:	5	4
时钟数:	2	1
Hex:	[A5]指令码	指令码

MOV @DRk+dis,WRj

指令操作: (PC) ←(PC) + 4

((DRk)+dis) ←(WRj)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 79H

0 1 1 1	1 0 0 1	t t t t	u u u u	偏移高字节	偏移低字节

	Binary 模式	**Source 模式**
指令长度:	5	4
时钟数:	2	1
Hex:	[A5]指令码	指令码

MOV <dest–bit>,<src–bit>

功能: 搬运位数据

说明: 第二个操作数（可直接寻址的位）指定的布尔变量被复制到进位标志位中。没有其它寄存器或者标志位受到影响。

MOV C,Bit51

指令操作: (PC) ←(PC) + 2

(C) ←(bit51)

影响标志位:

CY	AC	OV	N	Z
✓	-	-	-	-

[指令码] A2H

1 0 1 0	0 0 1 0	位地址

	Binary 模式	**Source 模式**
指令长度:	2	2
时钟数:	1	1

Hex: 指令码　指令码

MOV C,Bit

指令操作: (PC)　←(PC) + 3

(C)　←(bit)

影响标志位:

CY	AC	OV	N	Z
✓	-	-	-	-

[指令码] A9A(y)H

1 0 1 0	1 0 0 1	1 0 1 0	y y y y	直接地址

Binary 模式　**Source 模式**

指令长度: 4　3

时钟数: 2　1

Hex: [A5]指令码　指令码

MOV Bit51,C

指令操作: (PC)　←(PC) + 2

(bit51)　←(C)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 92H

1 0 0 1	0 0 1 0	位地址

Binary 模式　**Source 模式**

指令长度: 2　2

时钟数: 1　1

Hex: 指令码　指令码

MOV Bit,C

指令操作: (PC)　←(PC) + 3

(bit)　←(C)

影响标志位:

CY	AC	OV	N	Z
✓	-	-	-	-

[指令码] A99(y)H

1 0 1 0	1 0 0 1	1 0 0 1	y y y y	直接地址

Binary 模式　**Source 模式**

指令长度: 4　3

时钟数: 2　1

Hex: [A5]指令码　指令码

MOV DPTR,#DATA16

功能: 数据指针加载 16 位常数

说明: 数据指针加载指定的 16 位常数。16 位常数加载到指令的第二个和第三个字节中。第二个字节（DPH）是高位字节，而第三个字节（DPL）是低位字节。不影响标志位。

指令操作: (PC)　←(PC) + 3

DPH　←立即数 15..8

DPL　←立即数 7..0

影响标志位:

CY	AC	OV	N	Z

-	-	-	-	-

[指令码] 90H

1 0 0 1	0 0 0 0	立即数高字节	立即数低字节

Binary 模式 **Source 模式**

指令长度: 3 3

时钟数: 1 1

Hex: 指令码 指令码

MOVC

功能: 搬运代码字节

说明: MOVC 指令将代码字节或者程序存储器中的常数加载到累加器中。取出的字节地址是原始的无符号 8 位累加器内容以及 16 位基址寄存器的内容之和，该基址寄存器可以是数据指针也可以是 PC。在后一种情况下，PC 会递增到下一条指令的地址再加上累加器；否则，基址寄存器不会改变。执行 16 位加法，因此低 8 位的进位可以传送到高位。不影响标志位。

MOVC A,@A+DPTR

指令操作: (PC) ←(PC) + 1

(A) ←((A) + (DPTR))

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 93H

1 0 0 1	0 0 1 1

Binary 模式 **Source 模式**

指令长度: 1 1

时钟数: 4 4

Hex: 指令码 指令码

MOVC A,@A+PC

指令操作: (PC) ←(PC) + 1

(A) ←((A) + (PC))

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 83H

1 0 0 0	0 0 1 1

Binary 模式 **Source 模式**

指令长度: 1 1

时钟数: 3 3

Hex: 指令码 指令码

MOVH DRk,#DATA16

功能: 将 16 位立即数搬运到 dword（双字）寄存器的高位字。

说明: 将 16 位立即数搬运到双字（32 位）寄存器的高位字。双字寄存器的低位字不变。

指令操作: (PC) ←(PC) + 4

(DRk).31-16 ←#data16

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 7A(u)CH

0 1 1 1	1 0 1 0	u u u u	1 1 0 0	立即数高字节	立即数低字节

Binary 模式　**Source 模式**

指令长度: 5　4

时钟数: 2　1

Hex: [A5]指令码　指令码

MOVS WRj,Rm

功能: 将 8 位寄存器搬运到有符号扩展的 16 位寄存器

说明: 将 8 位寄存器的内容搬运到 16 位寄存器的低字节。16 位寄存器的高字节用符号扩展填充，符号是从 8 位源数寄存器的最高有效位获得的。

指令操作:
(PC) ←(PC) + 2
(WRj).7-0 ←(Rm)
(WRj).15-8 ←符号扩展

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 1AH

0 0 0 1	1 0 1 0	t t t t	s s s s

Binary 模式　**Source 模式**

指令长度: 3　2

时钟数: 2　1

Hex: [A5]指令码　指令码

MOVX

功能: 搬运外部的

说明: MOVX 指令在累加器和外部数据存储器的一个字节之间传输数据，因此 MOV 后附加 X。 有两种类型的指令，不同之处在于它们向外部数据内存提供 8 位还是 16 位间接地址。在第一种类型中，当前寄存器组 R0 或者 R1 中的内容提供了一个 8 位地址，在第二种 MOVX 指令的类型中，数据指针生成了一个 16 位地址。

MOVX A,@Ri

指令操作:
(PC) ←(PC) + 1
(A) ←((Ri))

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] E2H, E3H

1 1 1 0	0 0 1 i

Binary 模式　**Source 模式**

指令长度: 1　1

时钟数: 3*　3*

Hex: 指令码　指令码

MOVX A,@DPTR

指令操作:
(PC) ←(PC) + 1
(A) ←((DPTR))

影响标志位:

CY	AC	OV	N	Z

	-	-	-	-	-

[指令码] E0H

1	1	1	0	0	0	0	0

Binary 模式 **Source 模式**

指令长度: 1 1

时钟数: 3* 3*

Hex: 指令码 指令码

MOVX @Ri,A

指令操作: (PC) ←(PC) + 1

((Ri)) ←(A)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] F2H, F3H

1	1	1	1	0	0	1	i

Binary 模式 **Source 模式**

指令长度: 1 1

时钟数: 2* 2*

Hex: 指令码 指令码

MOVX @DPTR,A

指令操作: (PC) ←(PC) + 1

((DPTR)) ←(A)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] F0H

1	1	1	1	0	0	0	0

Binary 模式 **Source 模式**

指令长度: 1 1

时钟数: 2* 2*

Hex: 指令码 指令码

MOVZ WRj,Rm

功能: 将 8 位寄存器搬运到 16 位寄存器并扩展零

说明: 将 8 位寄存器的内容搬运到 16 位寄存器的低字节。16 位寄存器的高字节用零填充。

指令操作: (PC) ←(PC) + 2

(WRj).7-0 ←(Rm)

(WRj).15-8 ←0

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 0AH

0	0	0	0	1	0	1	0	t	t	t	t	s	s	s	s

Binary 模式 **Source 模式**

指令长度: 3 2

时钟数: 2 1

Hex: [A5]指令码 指令码

MUL

功能: 相乘

说明: 将源数寄存器中的无符号整数与目标寄存器中的无符号整数相乘。只允许寄存器寻址。对于 8 位操作数，结果为 16 位。结果的最高有效字节存储在目标寄存器所在字的低字节中。最低有效字节存储在紧接着的字节寄存器中。如果乘积大于 255(0FFH)，则 OV 标志位置位；否则将被清零。对于 16 位操作数，结果为 32 位。最高有效字存储在目标寄存器所在的双字的低位字中。最低有效字存储在紧接着的字寄存器中。在此操作中，如果乘积大于 0FFFFH，则 OV 标志位置位，否则清零。CY 标志位总是被清零。当置位结果的最高有效字节时 N 标志位置位。当结果为零时 Z 标志位置位。

MUL AB

指令操作:

(PC) ←(PC) + 1

(A) ←(A) × (B) -结果位 7..0

(B) ←(A) × (B) -结果位 15..8

影响标志位:

CY	AC	OV	N	Z
0	-	✓	✓	✓

[指令码] A4H

1	0	1	0	0	1	0	0

Binary 模式 **Source 模式**

指令长度: 1 1

时钟数: 1 1

Hex: 指令码 指令码

MUL Rmd,Rms

指令操作: (PC)←(PC) + 2

if <dest>md = 0,2,4..,14

Rmd←Rmd × Rms 的高字节

Rmd+1←Rmd × Rms 的低字节

if <dest>md = 1,3,5..,15

Rmd-1←Rmd × Rms 的高字节

Rmd←Rmd × Rms 的低字节

影响标志位:

CY	AC	OV	N	Z
0	-	✓	✓	✓

[指令码] ACH

1	0	1	0	1	1	0	0	s	s	s	s	S	S	S	S

Binary 模式 **Source 模式**

指令长度: 3 2

时钟数: 2 1

Hex: [A5]指令码 指令码

MUL WRjd,WRjs

指令操作: (PC)←(PC) + 2

if <dest>jd = 0,4,8..,28

WRjd←WRjd × WRjs 的高字节

WRjd+2←WRjd × WRjs 的低字节

if　<dest>jd = 2,6,10..,30

WRjd-2←WRjd × WRjs 的高字节

WRjd←WRjd × WRjs 的低字节

影响标志位:

CY	AC	OV	N	Z
0	-	✓	✓	✓

[指令码] ADH

1 0 1 0	1 1 0 1	t t t T	T T T T

Binary 模式　**Source 模式**

指令长度: 3　2

时钟数: 2　1

Hex: [A5]指令码　指令码

NOP

功能: 无操作

说明: 继续执行接下来的指令。除了 PC 之外，不影响任何寄存器或者标志位。

指令操作: (PC)　←(PC) + 1

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 00H

0 0 0 0	0 0 0 0

Binary 模式　**Source 模式**

指令长度: 1　1

时钟数: 1　1

Hex: 指令码　指令码

ORL

功能: 变量的逻辑或

说明: 在指定变量之间执行按位逻辑或运算，将结果存储在目标操作数中。目标操作数可以是寄存器、累加器或者直接地址。这两个操作数允许 12 种寻址模式组合。当目标是累加器时，源数可以是寄存器、直接、寄存器间接或者立即寻址；当目标是直接地址时，源数可以是累加器或者立即数。当目标是寄存器时，源数可以是寄存器、立即、直接和间接寻址。只影响 N 和 Z 标志位。注意：当该指令用于修改输出端口时，用作原始端口数据的值将从输出数据锁存器中读取，而不是输入引脚。

ORL A,Rn

指令操作: (PC)　←(PC) + 1

(A)　←(A) or (Rn)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 48H - 4FH

0 1 0 0	1 r r r

Binary 模式　**Source 模式**

指令长度: 1　2

时钟数: 1　2

Hex: 指令码　[A5]指令码

ORL A,Dir

指令操作: (PC) ←(PC) + 2

(A) ←(A) or (direct)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 45H

0 1 0 0	0 1 0 1	直接地址

Binary 模式 **Source 模式**

指令长度: 2 2

时钟数: 1 1

Hex: 指令码 指令码

ORL A,@Ri

指令操作: (PC) ←(PC) + 1

(A) ←(A) or ((Ri))

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 46H, 47H

0 1 0 0	0 1 1 i

Binary 模式 **Source 模式**

指令长度: 1 2

时钟数: 1 2

Hex: 指令码 [A5]指令码

ORL A,#DATA

指令操作: (PC) ←(PC) + 1

(A) ←(A) or #data

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 44H

0 1 0 0	0 1 0 0	立即数

Binary 模式 **Source 模式**

指令长度: 2 2

时钟数: 1 1

Hex: 指令码 指令码

ORL Dir,A

指令操作: (PC) ←(PC) + 1

(direct) ←(direct) or (A)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 42H

0 1 0 0	0 0 1 0	直接地址

Binary 模式 **Source 模式**

指令长度: 2 2

时钟数: 1 1

Hex: 指令码　指令码

ORL Dir,#DATA

指令操作: (PC) ←(PC) + 1

(direct) ←(direct) or #data

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 43H

0 1 0 0	0 0 1 1	直接地址	立即数

Binary 模式 **Source 模式**

指令长度: 3　3

时钟数: 1　1

Hex: 指令码　指令码

ORL Rmd,Rms

指令操作: (PC) ←(PC) + 2

(Rmd) ←(Rms) or (Rmd)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 4CH

0 1 0 0	1 1 0 0	s s s s	S S S S

Binary 模式 **Source 模式**

指令长度: 3　2

时钟数: 2　1

Hex: [A5]指令码　指令码

ORL WRjd,WRjs

指令操作: (PC) ←(PC) + 2

(WRjd) ←(WRjs) or (WRjd)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 4DH

0 1 0 0	1 1 0 1	t t t t	S S S

Binary 模式 **Source 模式**

指令长度: 3　2

时钟数: 2　1

Hex: [A5]指令码　指令码

ORL Rm,#DATA

指令操作: (PC) ←(PC) + 3

(Rm) ←(Rm) or #data

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 4E(s)0H

0 1 0 0	1 1 1 0	s s s s	0 0 0 0	立即数

Binary 模式 **Source 模式**

指令长度:	4	3
时钟数:	2	1
Hex:	[A5]指令码	指令码

ORL WRj,#DATA16

指令操作:	(PC)	←(PC) + 4
	(WRj)	←(WRj) or #data16

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 4E(t)4H

0	1	0	0	1	1	1	0	t	t	t	t	0	1	0	0	立即数高字节	立即数低字节

Binary 模式　**Source 模式**

指令长度:	5	4
时钟数:	2	1
Hex:	[A5]指令码	指令码

ORL Rm,Dir8

指令操作:	(PC)	←(PC) + 3
	(Rm)	←(Rm) or (dir8)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 4E(s)1H

0	1	0	0	1	1	1	0	s	s	s	s	0	0	0	1	直接地址	

Binary 模式　**Source 模式**

指令长度:	4	3
时钟数:	2	1
Hex:	[A5]指令码	指令码

ORL WRj,Dir8

指令操作:	(PC)	←(PC) + 3
	(WRj)	←(WRj) or (dir8)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 4E(t)5H

0	1	0	0	1	1	1	0	t	t	t	t	0	1	0	1	直接地址	

Binary 模式　**Source 模式**

指令长度:	4	3
时钟数:	2(3 for SFR)	1(2 for SFR)
Hex:	[A5]指令码	指令码

ORL Rm,Dir16

指令操作:	(PC)	←(PC) + 4
	(Rm)	←(Rm) or (dir16)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 4E(s)3H

0 1 0 0	1 1 1 0	s s s s	0 0 1 1	地址高字节	地址低字节

Binary 模式　**Source 模式**

指令长度: 5　4

时钟数: 2　1

Hex: [A5]指令码　指令码

ORL WRj,Dir16

指令操作: (PC)　←(PC) + 4

(WRj)　←(WRj) or (dir16)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 4E(t)7H

0 1 0 0	1 1 1 0	t t t t	0 1 1 1	地址高字节	地址低字节

Binary 模式　**Source 模式**

指令长度: 5　4

时钟数: 2　1

Hex: [A5]指令码　指令码

ORL Rm,@WRj

指令操作: (PC)　←(PC) + 3

(Rm)　←(Rm) or ((WRj))

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 4E(t)9(s)0H

0 1 0 0	1 1 1 0	t t t t	1 0 0 1	s s s s	0 0 0 0

Binary 模式　**Source 模式**

指令长度: 4　3

时钟数: 2　1

Hex: [A5]指令码　指令码

ORL Rm,@DRk

指令操作: (PC)　←(PC) + 3

(Rm)　←(Rm) or ((DRk))

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 4E(u)B(s)0H

0 1 0 0	1 1 1 0	u u u U	1 0 1 1	s s s s	0 0 0 0

Binary 模式　**Source 模式**

指令长度: 4　3

时钟数: 2　1

Hex: [A5]指令码　指令码

ORL CY,<src–bit>

功能: 位变量的逻辑或

说明: 如果布尔值为逻辑 1，则进位标志位置位；否则进位保持在当前状态。汇编语言中操作数前的斜杠（“/”）表示将寻址位的逻辑补码用作源数的值，但源位本身不受影响。不影响其它标志位。

ORL C,Bit51

指令操作: (PC) ←(PC) + 2

(C) ←(C) or (bit51)

影响标志位:

CY	AC	OV	N	Z
✓	-	-	-	-

[指令码] 72H

0 1 1 1	0 0 1 0	位地址

Binary 模式 **Source 模式**

指令长度: 2 2

时钟数: 1 1

Hex: 指令码 指令码

ORL C,/Bit51

指令操作: (PC) ←(PC) + 2

(C) ←(C) or /(bit51)

影响标志位:

CY	AC	OV	N	Z
✓	-	-	-	-

[指令码] A0H

1 0 1 0	0 0 0 0	位地址

Binary 模式 **Source 模式**

指令长度: 2 2

时钟数: 1 1

Hex: 指令码 指令码

ORL C,Bit

指令操作: (PC) ←(PC) + 2

(C) ←(C) or (bit)

影响标志位:

CY	AC	OV	N	Z
✓	-	-	-	-

[指令码] A97(y)H

1 0 1 0	1 0 0 1	0 1 1 1	0 y y y	直接地址

Binary 模式 **Source 模式**

指令长度: 4 3

时钟数: 2 1

Hex: [A5]指令码 指令码

ORL C,/Bit

指令操作: (PC) ←(PC) + 2

(C) ←(C) or /(bit)

影响标志位:

CY	AC	OV	N	Z
✓	-	-	-	-

[指令码] A9E(y)H

1 0 1 0	1 0 0 1	1 1 1 0	0 y y y	直接地址

Binary 模式 **Source 模式**

指令长度: 4 3

时钟数: 2　1

Hex: [A5]指令码　指令码

POP

功能: 弹出堆栈

说明: 读取由堆栈指针寻址的片上内存位置的内容，然后将堆栈指针减 1。在之前内存位置读取的值被传输到新寻址的位置，该值可以是 8 位或者 16 位。不影响标志位。

POP Dir8

指令操作: (PC) ←(PC) + 2

(dir8) ←((SP))

(SP) ←(SP) - 1

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] D0H

1 1 0 1	0 0 0 0	直接地址

Binary 模式 **Source 模式**

指令长度: 2　2

时钟数: 1　1

Hex: 指令码　指令码

POP Rm

指令操作: (PC) ←(PC) + 2

(Rm) ←((SP))

(SP) ←(SP) - 1

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] DA(s)8H

1 1 0 1	1 0 1 0	s s s s	1 0 0 0

Binary 模式 **Source 模式**

指令长度: 3　2

时钟数: 2　1

Hex: [A5]指令码　指令码

POP WRj

指令操作: (PC) ←(PC) + 2

(WRj) ←((SP))

(SP) ←(SP) - 2

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] DA(t)9H

1 1 0 1	1 0 1 0	t t t t	1 0 0 1

Binary 模式 **Source 模式**

指令长度: 3　2

时钟数: 2　1

Hex: [A5]指令码　指令码

POP DRk

指令操作: (PC) ←(PC) + 2
(DRk) ←((SP))
(SP) ←(SP) - 3

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] DA(u)BH

1	1	0	1	1	0	1	0	u	u	u	u	1	0	1	1

Binary 模式 **Source 模式**

指令长度: 3 2
时钟数: 2 1
Hex: [A5]指令码 指令码

PUSH

功能: 压入堆栈
说明: 将堆栈指针加 1。然后将指定变量的内容复制到堆栈指针寻址的片上内存位置。不影响标志位。

PUSH Dir8

指令操作: (PC) ←(PC) + 2
(SP) ←(SP) + 1
((SP)) ←(dir8)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] C0H

1	1	0	0	0	0	0	0	直接地址

Binary 模式 **Source 模式**

指令长度: 2 2
时钟数: 1 1
Hex: 指令码 指令码

PUSH #DATA

指令操作: (PC) ←(PC) + 2
(SP) ←(SP) + 1
((SP)) ←#data

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] CA02H

1	1	0	0	1	0	1	0	0	0	0	0	0	0	1	0	立即数

Binary 模式 **Source 模式**

指令长度: 4 3
时钟数: 2 1
Hex: [A5]指令码 指令码

PUSH #DATA16

指令操作: (PC) ←(PC) + 2

(SP) ←(SP) + 1
((SP)) ←MSB #data
(SP) ←(SP) + 1
((SP)) ←LSB #data

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] CA06H

1 1 0 0	1 0 1 0	0 0 0 0	0 1 1 0	立即数高字节	立即数低字节

Binary 模式 **Source 模式**

指令长度: 5 4

时钟数: 2 1

Hex: [A5]指令码 指令码

PUSH Rm

指令操作: (PC) ←(PC) + 2
(SP) ←(SP) + 1
((SP)) ←(Rm)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] CA(s)8H

1 1 0 0	1 0 1 0	s s s s	1 0 0 0

Binary 模式 **Source 模式**

指令长度: 3 2

时钟数: 2 1

Hex: [A5]指令码 指令码

PUSH WRj

指令操作: (PC) ←(PC) + 2
(SP) ←(SP) + 1
((SP)) ←(WRj)
(SP) ←(SP) + 1

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] CA(t)9H

1 1 0 0	1 0 1 0	t t t t	1 0 0 1

Binary 模式 **Source 模式**

指令长度: 3 2

时钟数: 2 1

Hex: [A5]指令码 指令码

PUSH DRk

指令操作: (PC) ←(PC) + 2
(SP) ←(SP) + 1
((SP)) ←(DRk)
(SP) ←(SP) + 3

影响标志位:

CY	AC	OV	N	Z

-	-	-	-	-

[指令码] CA(u)BH

1	1	0	0	1	0	1	0	u	u	u	u	1	0	1	1

Binary 模式　**Source 模式**

指令长度: 3　2

时钟数: 2　1

Hex: [A5]指令码　指令码

RET

功能: 从子程序返回

说明: RET 依次从堆栈中弹出 PC 的高位和低位字节，堆栈指针减 2。程序在结果地址处继续执行，通常是紧跟在 ACALL 或者 LCALL 之后的指令。不影响标志位。

指令操作:
(PC15-8) ←((SP))
(SP) ←(SP) - 1
(PC7-0) ←((SP))
(SP) ←(SP) - 1

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 22H

0	0	1	0	0	0	1	0

Binary 模式　**Source 模式**

指令长度: 1　1

时钟数: 3　3

Hex: 指令码　指令码

RETI

功能: 从中断返回

说明: 该指令从堆栈中弹出两个或者四个字节，具体取决于 CONFIG1 寄存器中的 INTR 位。如果 INTR = 0，RETI 将 PC 的高字节和低字节依次从堆栈中弹出，并用作 FF:区域的 16 位返回地址。堆栈指针减 2。不影响其它寄存器，PSW 和 PSW1 都不会自动恢复到中断前的状态。如果 INTR = 1，RETI 从堆栈中弹出四个字节：PSW1 和 PC 的三个字节。PC 的三个字节是返回地址，它可以是 16MB 内存空间中的任何位置。堆栈指针减四。PSW1 恢复到中断前状态，但 PSW 没有恢复到中断前状态。不影响其它寄存器。对于 INTR 的任意值，硬件都会恢复中断逻辑以接收跟刚处理的中断具有相同优先级的其它中断。程序在返回地址继续执行，该地址通常是检测到中断请求之后的指令。如果有相同或者较低优先级的中断在等待时执行了 RETI 指令，则在处理等待中断之前先执行该条指令。

指令操作:

INTR=1:		INTR=0:	
(PC15-8)	←((SP))	(PC15-8)	←((SP))
(SP)	←(SP) - 1	(SP)	←(SP) - 1
(PC7-0)	←((SP))	(PC7-0)	←((SP))
(SP)	←(SP) - 1	(SP)	←(SP) - 1
(PC23-16)	←((SP))		
(SP)	←(SP) - 1		
PSW1	←((SP))		
(SP)	←(SP) - 1		

影响标志位:

CY	AC	OV	N	Z

-	-	-	-	-

[指令码] 32H

0	0	1	1	0	0	1	0

Binary 模式 **Source 模式**

指令长度: 1 1

时钟数: 3 3

Hex: 指令码 指令码

RL

功能: 累加器循环左移

说明: 累加器中的八位向左循环移动一位。第 7 位循环到第 0 位的位置。只影响 N 和 Z 标志位。

指令操作: (PC) ←(PC) + 1

(An + 1) ←(An) n = 0-6

(A0) ←(A7)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 23H

0	0	1	0	0	0	1	1

Binary 模式 **Source 模式**

指令长度: 1 1

时钟数: 1 1

Hex: 指令码 指令码

RLC

功能: 带进位标志位累加器循环左移

说明: 累加器中的八位和进位标志位一起向左循环移动一位。第 7 位移入进位标志位；进位标志位的之前状态移入第 0 位的位置。N 和 Z 标志位也会受到影响。

指令操作: (PC) ←(PC) + 1

(An + 1) ←(An) n = 0-6

(A0) ←(C)

(C) ←(A7)

影响标志位:

CY	AC	OV	N	Z
✓	-	-	✓	✓

[指令码] 33H

0	0	1	1	0	0	1	1

Binary 模式 **Source 模式**

指令长度: 1 1

时钟数: 1 1

Hex: 指令码 指令码

RR

功能: 累加器循环右移

说明: 累加器中的八位向右循环移动一位。第 0 位循环到第 7 位的位置。只影响 N 和 Z 标志位。

指令操作: (PC) ←(PC) + 1

(An) ←(An + 1) n = 0-6

(A7) ←(A0)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 03H

0 0 0 0	0 0 1 1

Binary 模式　**Source 模式**

指令长度: 1　1

时钟数: 1　1

Hex: 指令码　指令码

RRC

功能: 带进位标志位累加器循环右移

说明: 累加器中的八位和进位标志位一起向右循环移动一位。第 0 位移入进位标志位；进位标志位的之前状态移入第 7 位的位置。N 和 Z 标志位也会受到影响。

指令操作:
(PC)　←(PC) + 1
(An)　←(An + 1) n = 0-6
(A7)　←(C)
(C)　←(A0)

影响标志位:

CY	AC	OV	N	Z
✓	-	-	✓	✓

[指令码] 13H

0 0 0 1	0 0 1 1

Binary 模式　**Source 模式**

指令长度: 1　1

时钟数: 1　1

Hex: 指令码　指令码

SETB

功能: 置位

说明: SETB 将指定位置位为 1。SETB 可以对进位标志位或者任何可直接寻址的位进行操作。不影响其它标志位。

SETB C

指令操作:
(PC)　←(PC) + 1
(C)　←1

影响标志位:

CY	AC	OV	N	Z
✓	-	-	-	-

[指令码] D3H

1 1 0 1	0 0 1 1

Binary 模式　**Source 模式**

指令长度: 1　1

时钟数: 1　1

Hex: 指令码　指令码

SETB Bit51

指令操作:
(PC)　←(PC) + 2
(bit51)　←1

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] D2H

1 1 0 1	0 0 1 0	位地址

Binary 模式　**Source 模式**

指令长度: 2　2

时钟数: 1　1

Hex: 指令码　指令码

SETB Bit

指令操作: (PC)　←(PC) + 3

(bit)　←1

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] A9D(y)H

1 0 1 0	1 0 0 1	1 1 0 1	0 y y y	直接地址

Binary 模式　**Source 模式**

指令长度: 4　3

时钟数: 2　1

Hex: [A5]指令码　指令码

SJMP

功能: 短跳转

说明: 程序控制无条件地跳转到指定的地址。将 PC 加上指令第二个字节中的有符号相对偏移来计算跳转目标，然后 PC 递增两次。因此，允许的目标的范围是从该指令之前的 128 个字节到它之后的 127 个字节之间。不影响标志位。

指令操作: (PC)　←(PC) + 2

(PC)　←(PC) + rel

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] 80H

1 0 0 0	0 0 0 0	相对地址

Binary 模式　**Source 模式**

指令长度: 2　2

时钟数: 3　3

Hex: 指令码　指令码

SLL

功能: 逻辑左移 1 位

说明: 将指定变量左移 1 位，将最低有效位替换为零。移出的位最高有效位（MSB）存储在 CY 位中。N 和 Z 标志位也受到影响。

SLL Rm

指令操作: (PC)　←(PC) + 2

(Rm).a + 1　←(Rm).a

(Rm).0　←0

CY ←(Rm).7

影响标志位:

CY	AC	OV	N	Z
✓	-	-	✓	✓

[指令码] 3E(s)0H

0 0 1 1	1 1 1 0	s s s s	0 0 0 0

Binary 模式 **Source 模式**

指令长度: 3 2

时钟数: 2 1

Hex: [A5]指令码 指令码

SLL WRj

指令操作: (PC) ←(PC) + 2

(WRj).b + 1 ←(WRj).b

(WRj).0 ←0

CY ←(WRj).15

影响标志位:

CY	AC	OV	N	Z
✓	-	-	✓	✓

[指令码] 3E(t)4H

0 0 1 1	1 1 1 0	t t t t	0 1 0 0

Binary 模式 **Source 模式**

指令长度: 3 2

时钟数: 2 1

Hex: [A5]指令码 指令码

SRA

功能: 算术右移 1 位（有符号）

说明: 将指定的变量进行算术右移 1 位。最高有效位不变。移出的位（LSB）存储在 CY 位中。N 和 Z 标志位也受到影响。

SRA Rm

指令操作: (PC) ←(PC) + 2

(Rm).7 ←(Rm).7

(Rm).a ←(Rm).a+1

CY ←(Rm).0

影响标志位:

CY	AC	OV	N	Z
✓	-	-	✓	✓

[指令码] 0E(s)0H

0 0 0 0	1 1 1 0	s s s s	0 0 0 0

Binary 模式 **Source 模式**

指令长度: 3 2

时钟数: 2 1

Hex: [A5]指令码 指令码

SRA WRj

指令操作: (PC) ←(PC) + 2

(WRj).15 ←(WRj).15

(WRj).b　←(WRj).b + 1

CY　←(WRj).0

影响标志位:

CY	AC	OV	N	Z
✓	-	-	✓	✓

[指令码] 0E(t)4H

0	0	0	0	1	1	1	0	t	t	t	t	0	1	0	0

Binary 模式　**Source 模式**

指令长度: 3　2

时钟数: 2　1

Hex: [A5]指令码　指令码

SRL

功能: 逻辑右移 1 位

说明: SRL 将指定变量右移 1 位，将最高有效位替换为零。移出的位（LSB）存储在 CY 位中。N 和 Z 标志位也受到影响。

SRL Rm

指令操作:

(PC)　←(PC) + 2

(Rm).7　←(Rm).0

(Rm).a　←(Rm).a+1

CY　←(Rm).0

影响标志位:

CY	AC	OV	N	Z
✓	-	-	✓	✓

[指令码] 1E(s)0H

0	0	0	1	1	1	1	0	s	s	s	s	0	0	0	0

Binary 模式　**Source 模式**

指令长度: 3　2

时钟数: 2　1

Hex: [A5]指令码　指令码

SRL WRj

指令操作:

(PC)　←(PC) + 2

(WRj).15　←0

(WRj).b　←(WRj).b + 1

CY　←(WRj).0

影响标志位:

CY	AC	OV	N	Z
✓	-	-	✓	✓

[指令码] 1E(t)4H

0	0	0	1	1	1	1	0	t	t	t	t	0	1	0	0

Binary 模式　**Source 模式**

指令长度: 3　2

时钟数: 2　1

Hex: [A5]指令码　指令码

SUB

功能: 相减

说明: 从目标操作数中减去指定的变量，将结果留在目标操作数中。如果第 7 位需要借位，则 SUB 置位 CY 标志位（借位），否则清零 CY 位。当有符号整数相减时，OV 标志位表示正数减负数时出现了负数，或者负数减正数时出现了正结果。本说明中的第 7 位是指操作数的最高有效字节（8、16 或者 32 位）。源操作数允许四种寻址模式：立即、间接、寄存器和直接寻址。除了 AC，所有标志位都受到影响，它不影响字和双字减法。

SUB Rmd,Rms

指令操作: (PC) ←(PC) + 2

(Rmd) ←(Rmd) - (Rms)

影响标志位:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[指令码] 9CH

1 0 0 1	1 1 0 0	s s s s	S S S S

Binary 模式 **Source 模式**

指令长度: 3 2

时钟数: 2 1

Hex: [A5]指令码 指令码

SUB WRjd,WRjs

指令操作: (PC) ←(PC) + 2

(WRjd) ←(WRjd) - (WRjs)

影响标志位:

CY	AC	OV	N	Z
✓	-	✓	✓	✓

[指令码] 9DH

1 0 0 1	1 1 0 1	t t t t	T T T T

Binary 模式 **Source 模式**

指令长度: 3 2

时钟数: 2 1

Hex: [A5]指令码 指令码

SUB DRkd,DRks

指令操作: (PC) ←(PC) + 2

(DRkd) ←(DRkd) - (DRks)

影响标志位:

CY	AC	OV	N	Z
✓	-	✓	✓	✓

[指令码] 9FH

1 0 0 1	1 1 1 1	u u u u	U U U U

Binary 模式 **Source 模式**

指令长度: 3 2

时钟数: 2 1

Hex: [A5]指令码 指令码

SUB Rm,#DATA

指令操作: (PC) ←(PC) + 3

(Rm) ←(Rm) - #data

影响标志位:

CY	AC	OV	N	Z

✓	✓	✓	✓	✓

[指令码] 9E(s)0H

1	0	0	1	1	1	1	0	s	s	s	s	0	0	0	0	立即数

Binary 模式　**Source 模式**

指令长度: 4　3

时钟数: 2　1

Hex: [A5]指令码　指令码

SUB WRj,#DATA16

指令操作: (PC)　←(PC) + 4

(WRj)　←(WRj) - #data16

影响标志位:

CY	AC	OV	N	Z
✓	-	✓	✓	✓

[指令码] 9E(t)4H

1	0	0	1	1	1	1	0	t	t	t	t	0	1	0	0	立即数高字节	立即数低字节

Binary 模式　**Source 模式**

指令长度: 5　4

时钟数: 2　1

Hex: [A5]指令码　指令码

SUB DRk,#0DATA16

指令操作: (PC)　←(PC) + 4

(DRk)　←(DRk) - #data16

影响标志位:

CY	AC	OV	N	Z
✓	-	✓	✓	✓

[指令码] 9E(u)8H

1	0	0	1	1	1	1	0	u	u	u	u	1	0	0	0	立即数高字节	立即数低字节

Binary 模式　**Source 模式**

指令长度: 5　4

时钟数: 2　1

Hex: [A5]指令码　指令码

SUB Rm,Dir8

指令操作: (PC)　←(PC) + 3

(Rm)　←(Rm) - (dir8)

影响标志位:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[指令码] 9E(s)1H

1	0	0	1	1	1	1	0	s	s	s	s	0	0	0	1	直接地址

Binary 模式　**Source 模式**

指令长度: 4　3

时钟数: 2　1

Hex: [A5]指令码　指令码

SUB WRj,Dir8

指令操作: (PC)　←(PC) + 3

(WRj)　←(WRj) - (dir8)

影响标志位:

CY	AC	OV	N	Z
✓	-	✓	✓	✓

[指令码] 9E(t)5H

1	0	0	1	1	1	1	0	t	t	t	t	0	1	0	1	直接地址

Binary 模式　**Source 模式**

指令长度: 4　3

时钟数: 2(3 for SFR)　1(2 for SFR)

Hex: [A5]指令码　指令码

SUB Rm,Dir16

指令操作: (PC)　←(PC) + 4

(Rm)　←(Rm) - (dir16)

影响标志位:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[指令码] 9E(s)3H

1	0	0	1	1	1	1	0	s	s	s	s	0	0	1	1	地址高字节	地址低字节

Binary 模式　**Source 模式**

指令长度: 5　4

时钟数: 2　1

Hex: [A5]指令码　指令码

SUB WRj,Dir16

指令操作: (PC)　←(PC) + 4

(WRj)　←(WRj) - (dir16)

影响标志位:

CY	AC	OV	N	Z
✓	-	✓	✓	✓

[指令码] 9E(t)7H

1	0	0	1	1	1	1	0	t	t	t	t	0	1	1	1	地址高字节	地址低字节

Binary 模式　**Source 模式**

指令长度: 5　4

时钟数: 2　1

Hex: [A5]指令码　指令码

SUB Rm,@WRj

指令操作: (PC)　←(PC) + 3

(Rm)　←(Rm) - ((WRj))

影响标志位:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[指令码] 9E(t)9(s)0H

1	0	0	1	1	1	1	0	t	t	t	t	1	0	0	1	s	s	s	s	0	0	0	0

Binary 模式　**Source 模式**

指令长度: 4　3

时钟数: 2　1

Hex: [A5]指令码　指令码

SUB Rm,@DRk

指令操作: (PC) ←(PC) + 3

(Rm) ←(Rm) - ((DRk))

影响标志位:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[指令码] 9E(u)B(s)0H

1	0	0	1	1	1	1	0	u	u	u	U	1	0	1	1	s	s	s	s	0	0	0	0

Binary 模式 **Source 模式**

指令长度: 4 3

时钟数: 2 1

Hex: [A5]指令码 指令码

SUBB A,<src-byte>

功能: 借位减法

说明: SUBB 从累加器中一起减去指定的变量和进位标志位，将结果留在累加器中。如果第 7 位需要借位，则 SUBB 置位进位（借位）标志位，否则清零 C。（如果在执行 SUBB 指令之前 C 已置位，这表示在多种准确的减法中的上一步需要借位，因此从累加器中减去进位以及源操作数）。如果第 3 位需要借位，则 AC 置位，否则清零。如果需要借位到第 6 位而不是第 7 位，或者需要借位到第 7 位而不是第 6 位，则 OV 置位。OV 标志位表示正数减负数时出现了负数，或者负数减正数时出现了正数。源操作数允许四种寻址模式：寄存器、直接、寄存器间接或者立即数寻址。所有标志位都会受到影响。

SUBB A,Rn

指令操作: (PC) ←(PC) + 1

(A) ←(A) - (C) - (Rn)

影响标志位:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[指令码] 98H - 9FH

1	0	0	1	1	r	r	r

Binary 模式 **Source 模式**

指令长度: 1 2

时钟数: 1 2

Hex: 指令码 [A5]指令码

SUBB A,Direct

指令操作: (PC) ←(PC) + 2

(A) ←(A) - (C) - (direct)

影响标志位:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[指令码] 95H

1	0	0	1	0	1	0	1	直接地址

Binary 模式 **Source 模式**

指令长度: 2 2

时钟数: 1 1

Hex: 指令码 指令码

SUBB A,@Ri

指令操作: (PC) ←(PC) + 1

(A) ←(A) - (C) - ((Ri))

影响标志位:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[指令码] 96H, 97H

1 0 0 1	0 1 1 i
Binary 模式	**Source 模式**

指令长度: 1 2

时钟数: 1 2

Hex: 指令码 [A5]指令码

SUBB A,#DATA

指令操作: (PC) ←(PC) + 2

(A) ←(A) - (C) - #data

影响标志位:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[指令码] 94H

1 0 0 1	0 1 0 0	立即数
Binary 模式	**Source 模式**	

指令长度: 2 2

时钟数: 1 1

Hex: 指令码 指令码

SWAP A

功能: 累加器交换半字节

说明: SWAP A 交换累加器的低位和高位半字节（四位域）（第 3 到第 0 位和第 7 到第 4 位）。该操作也可以认为是一个四位循环指令。 只影响 N 和 Z 标志位。

指令操作: (PC) ←(PC) + 1

(A3-0) ←(A7-4)

(A7-4) ←(A3-0)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] C4H

1 1 0 0	0 1 0 0
Binary 模式	**Source 模式**

指令长度: 1 1

时钟数: 1 1

Hex: 指令码 指令码

TRAP

功能: 作为 NOP 执行

指令操作: (PC) ←(PC) + 1

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] B9H

1 0 1 1	1 0 0 1
Binary 模式	**Source 模式**

指令长度: 2　1

时钟数: 2　1

Hex: [A5]指令码　指令码

XCH A,<byte>

功能: 累加器交换字节变量

说明: XCH 将指定变量的内容加载到累加器，同时将之前累加器内容写入指定变量。源数、目标操作数可以使用寄存器、直接或者寄存器间接寻址。不影响标志位。

XCH A,Rn

指令操作: (PC)　←(PC) + 1

(A)　↔(Rn)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] C8H - CFH

1 1 0 0	1 r r r
Binary 模式	**Source 模式**

指令长度: 1　2

时钟数: 1　2

Hex: 指令码　[A5]指令码

XCH A,Direct

指令操作: (PC)　←(PC) + 2

(A)　↔(direct)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] C5H

1 1 0 0	0 1 0 1	直接地址
Binary 模式	**Source 模式**	

指令长度: 2　2

时钟数: 1　1

Hex: 指令码　指令码

XCH A,@Ri

指令操作: (PC)　←(PC) + 1

(A)　↔((Ri))

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] C6H, C7H

1 1 0 0	0 1 1 i
Binary 模式	**Source 模式**

指令长度: 1　2

时钟数: 1　2

Hex: 指令码　[A5]指令码

XCHD A,@Ri

功能: 半字节交换数字

说明: XCHD 将累加器的低半字节（第 3 到第 0 位，通常表示十六进制或者 BCD 数字）与指定寄存器间接寻址的内部内存位置的半字节交换。每个寄存器的高位半字节（第 7 到第 4 位）不受影响。不影响标志位。

指令操作: (PC) ←(PC) + 1

(A3-0) ↔((Ri)3-0)

影响标志位:

CY	AC	OV	N	Z
-	-	-	-	-

[指令码] D6H, D7H

1 1 0 1	0 1 1 i

Binary 模式 **Source 模式**

指令长度: 1 2

时钟数: 1 2

Hex: 指令码 [A5]指令码

XRL

功能: 变量的逻辑异或

说明: 在指定变量之间执行按位逻辑异或运算，将结果存储在目标中。目标操作数可以是累加器、寄存器或者直接地址。 这两个操作数允许 12 种寻址模式组合。当目标是累加器或者寄存器时，源数可以是寄存器、直接、寄存器间接或者立即数寻址；当目标是直接地址时，源数可以是累加器或者立即数。只影响 N 和 Z 标志位。

注意：当该指令用于修改输出端口时，用作原始端口数据的值将从输出数据锁存器中读取，而不是输入引脚。

XRL A,Rn

指令操作: (PC) ←(PC) + 1

(A) ←(A) xor (Rn)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 68H - 6FH

0 1 1 0	1 r r r

Binary 模式 **Source 模式**

指令长度: 1 2

时钟数: 1 2

Hex: 指令码 [A5]指令码

XRL A,Direct

指令操作: (PC) ←(PC) + 2

(A) ←(A) xor (direct)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 65H

0 1 1 0	0 1 0 1	直接地址

Binary 模式 **Source 模式**

指令长度:	2	3
时钟数:	1	1
Hex:	指令码	指令码

XRL A,@Ri

指令操作: (PC) ←(PC) + 1

(A) ←(A) xor ((Ri))

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 66H, 67H

0	1	1	0	0	1	1	i

Binary 模式　**Source 模式**

指令长度:	1	2
时钟数:	1	2
Hex:	指令码	[A5]指令码

XRL A,#DATA

指令操作: (PC) ←(PC) + 2

(A) ←(A) xor #data

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 64H

0	1	1	0	0	1	0	1	立即数

Binary 模式　**Source 模式**

指令长度:	2	2
时钟数:	1	1
Hex:	指令码	指令码

XRL Direct,A

指令操作: (PC) ←(PC) + 2

(direct) ←(direct) xor (A)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 62H

0	1	1	0	0	0	1	0	直接地址

Binary 模式　**Source 模式**

指令长度:	2	2
时钟数:	1	1
Hex:	指令码	指令码

XRL Direct,#DATA

指令操作: (PC) ←(PC) + 3

(direct) ←(direct) xor #data

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 63H

0 1 1 0	0 0 1 1	直接地址	立即数

Binary 模式　**Source 模式**

指令长度: 3　3

时钟数: 1　1

Hex: 指令码　指令码

XRL Rmd,Rms

指令操作: (PC)　←(PC) + 2

(Rmd)　←(Rms) xor (Rmd)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 6CH

0 1 1 0	1 1 0 0	s s s s	S S S S

Binary 模式　**Source 模式**

指令长度: 3　2

时钟数: 2　1

Hex: [A5]指令码　指令码

XRL WRjd,WRjs

指令操作: (PC)　←(PC) + 2

(WRjd)　←(WRjs) xor (WRjd)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 6DH

0 1 1 0	1 1 0 1	t t t t	T T T T

Binary 模式　**Source 模式**

指令长度: 3　2

时钟数: 2　1

Hex: [A5]指令码　指令码

XRL Rm,#DATA

指令操作: (PC)　←(PC) + 3

(Rm)　←(Rm) xor #data

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 6E(s)0H

0 1 1 0	1 1 1 0	s s s s	0 0 0 0	立即数

Binary 模式　**Source 模式**

指令长度: 4　3

时钟数: 2　1

Hex: [A5]指令码　指令码

XRL WRj,#DATA16

指令操作: (PC)　←(PC) + 4

(WRj)　←(WRj) xor #data16

影响标志位:

CY	AC	OV	N	Z

-	-	-	✓	✓	

[指令码] 6E(t)4H

0	1	1	0	1	1	0	1	t	t	t	t	0	1	0	0	立即数高字节	立即数低字节

Binary 模式　**Source 模式**

指令长度: 5　4

时钟数: 2　1

Hex: [A5]指令码　指令码

XRL Rm,Dir8

指令操作: (PC)　←(PC) + 3

(Rm)　←(Rm) xor (dir8)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 6E(s)1H

0	1	1	0	1	1	1	0	s	s	s	s	0	0	0	1	直接地址

Binary 模式　**Source 模式**

指令长度: 4　3

时钟数: 2　1

Hex: [A5]指令码　指令码

XRL WRj,Dir8

指令操作: (PC)　←(PC) + 3

(WRj)　←(WRj) xor (dir8)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 6E(t)5H

0	1	1	0	1	1	0	1	t	t	t	t	0	1	0	1	直接地址

Binary 模式　**Source 模式**

指令长度: 4　3

时钟数: 2(3 for SFR)　1(2 for SFR)

Hex: [A5]指令码　指令码

XRL Rm,Dir16

指令操作: (PC)　←(PC) + 4

(Rm)　←(Rm) xor (dir16)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 6E(s)3H

0	1	1	0	1	1	1	0	s	s	s	s	0	0	1	1	地址高字节	地址低字节

Binary 模式　**Source 模式**

指令长度: 5　4

时钟数: 2　1

Hex: [A5]指令码　指令码

XRL WRj,Dir16

指令操作: (PC)　←(PC) + 4

(WRj)　←(WRj) xor (dir16)

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 6E(t)7H

0 1 1 0	1 1 0 1	t t t t	0 1 1 1	地址高字节	地址低字节

Binary 模式　**Source 模式**

指令长度: 5　4

时钟数: 2　1

Hex: [A5]指令码　指令码

XRL Rm,@WRj

指令操作: (PC)　←(PC) + 3

(Rm)　←(Rm) xor ((WRj))

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 6E(t)9(s)0H

0 1 1 0	1 1 1 0	t t t t	1 0 0 1	s s s s	0 0 0 0

Binary 模式　**Source 模式**

指令长度: 4　3

时钟数: 2　1

Hex: [A5]指令码　指令码

XRL Rm,@DRk

指令操作: (PC)　←(PC) + 3

(Rm)　←(Rm) xor ((DRk))

影响标志位:

CY	AC	OV	N	Z
-	-	-	✓	✓

[指令码] 6E(u)B(s)0H

0 1 1 0	1 1 1 0	u u u u	1 0 1 1	s s s s	0 0 0 0

Binary 模式　**Source 模式**

指令长度: 4　3

时钟数: 2　1

Hex: [A5]指令码　指令码

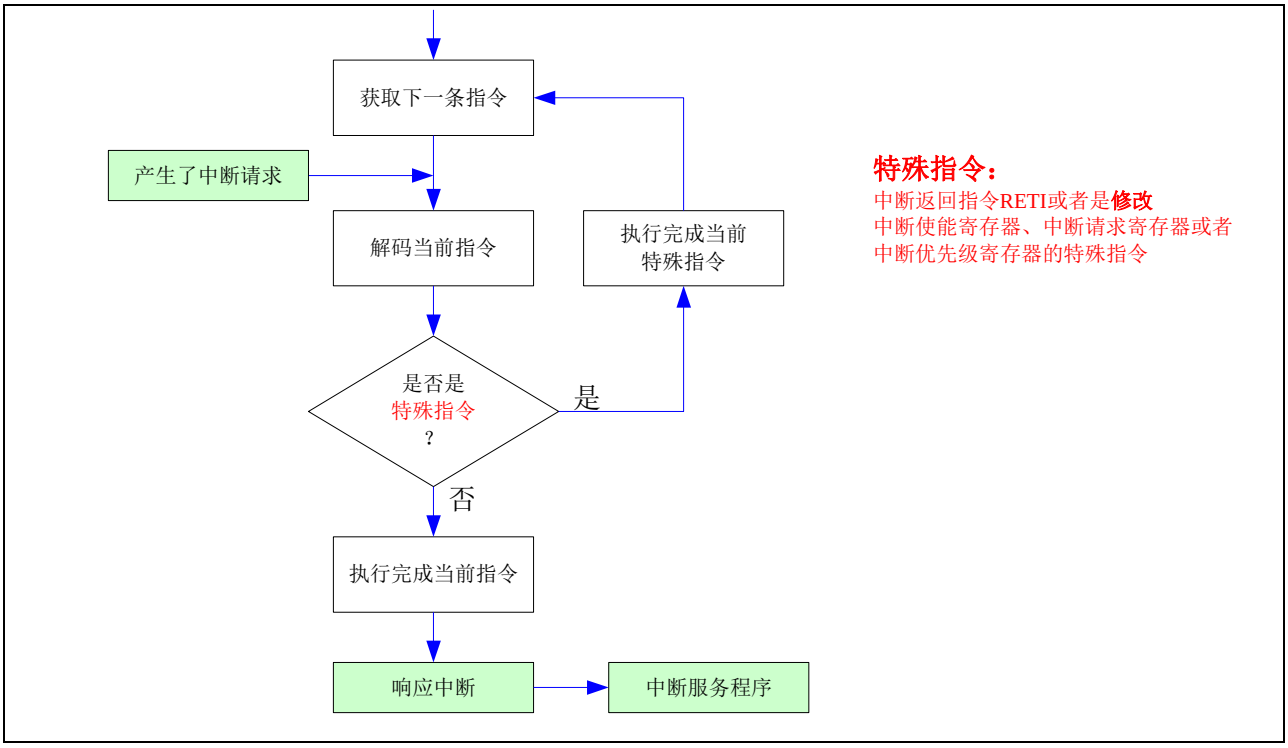
A.3　　多级流水线内核的中断响应

增强型 8051（例如：AI32G/AI8H 系列）和 32 位 8051（例如：Ai8051U 系列）的 MCU 内核为多级流水线设计，在中断响应方面的设计和传统的 8051（例如：STC89C52 系列）略有差异。

对于传统的 8051（例如：STC89C52 系列）：

如果当前正在执行的指令是中断返回指令 RETI 或者是访问中断使能寄存器、中断请求寄存器或者中断优先级寄存器的特殊指令时，CPU 等待当前的这条特殊的指令执行完，再执行一条指令才能响应中断请求；

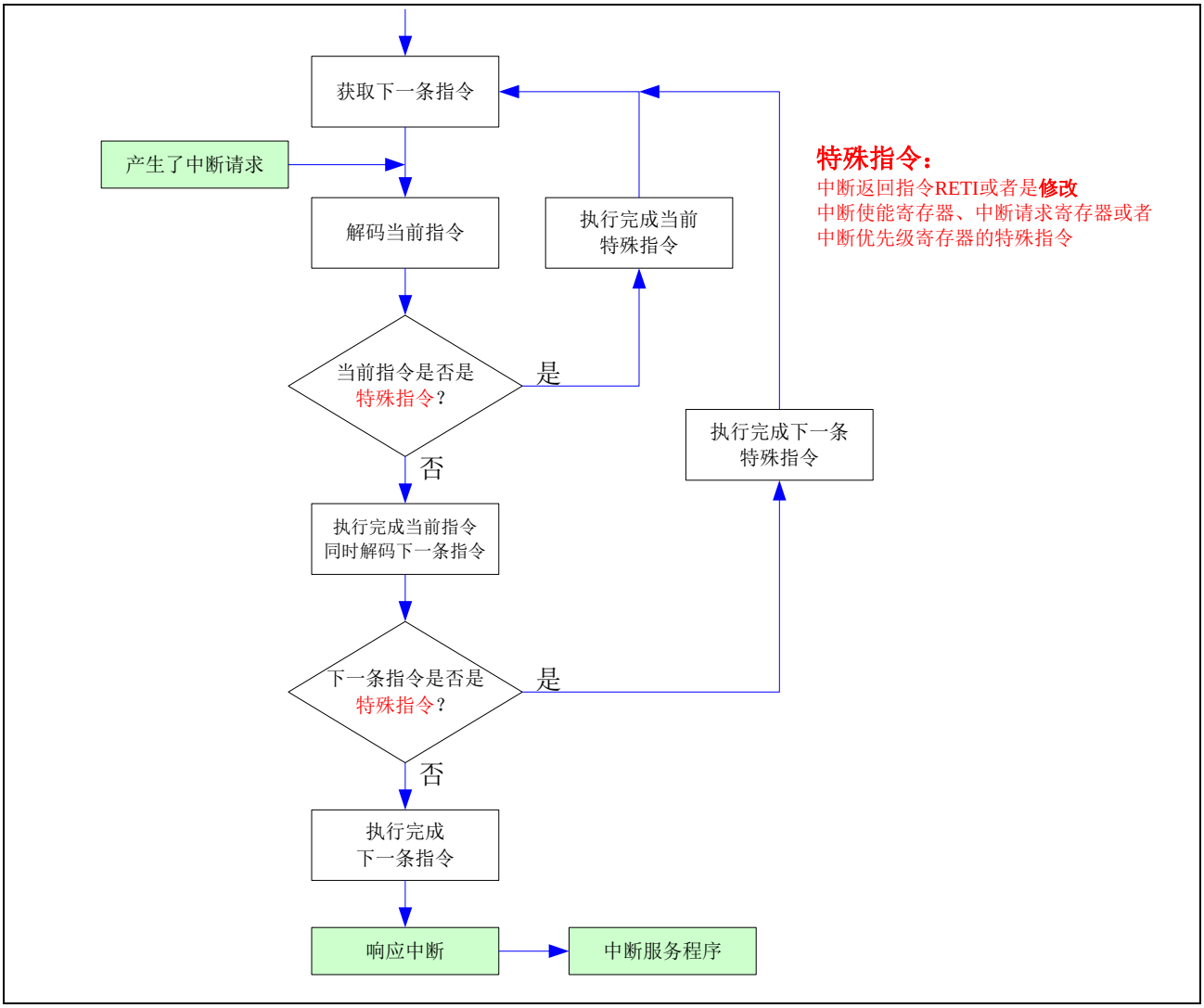
如果当前正在执行的指令不是上面所指的特殊指令，则等当前指令执行完成后就立即响应中断请求；



对于增强型 8051 单片机（例如：AI32G/AI8H 系列），由于是多级流水线设计，响应中断上会比传统的 8051（例如：STC89C52 系列）再多执行一条语句：

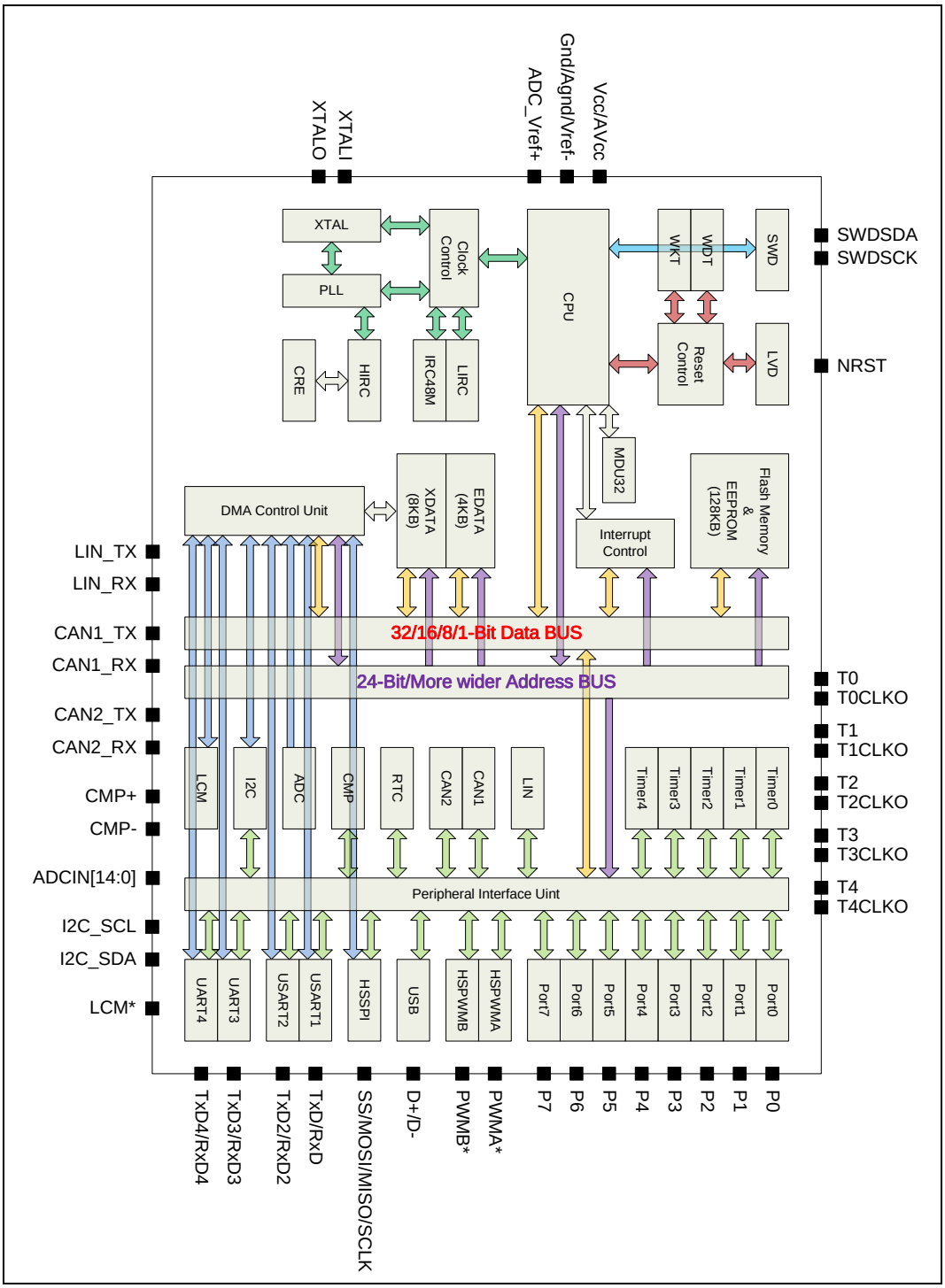
如果当前正在执行的指令是中断返回指令 RETI 或者是访问中断使能寄存器、中断请求寄存器或者中断优先级寄存器的特殊指令时，CPU 等待当前的这条特殊的指令执行完，同时解码下一条指令，直到下一条指令不是特殊指令，则等下一条指令执行完成才能响应中断请求；

如果当前正在执行的指令不是上面所指的特殊指令，则等当前指令执行完成后，同时会解码下一条指令，如果下一条也不是特殊指令，则会等下一条指令执行完成后再立即响应中断请求；

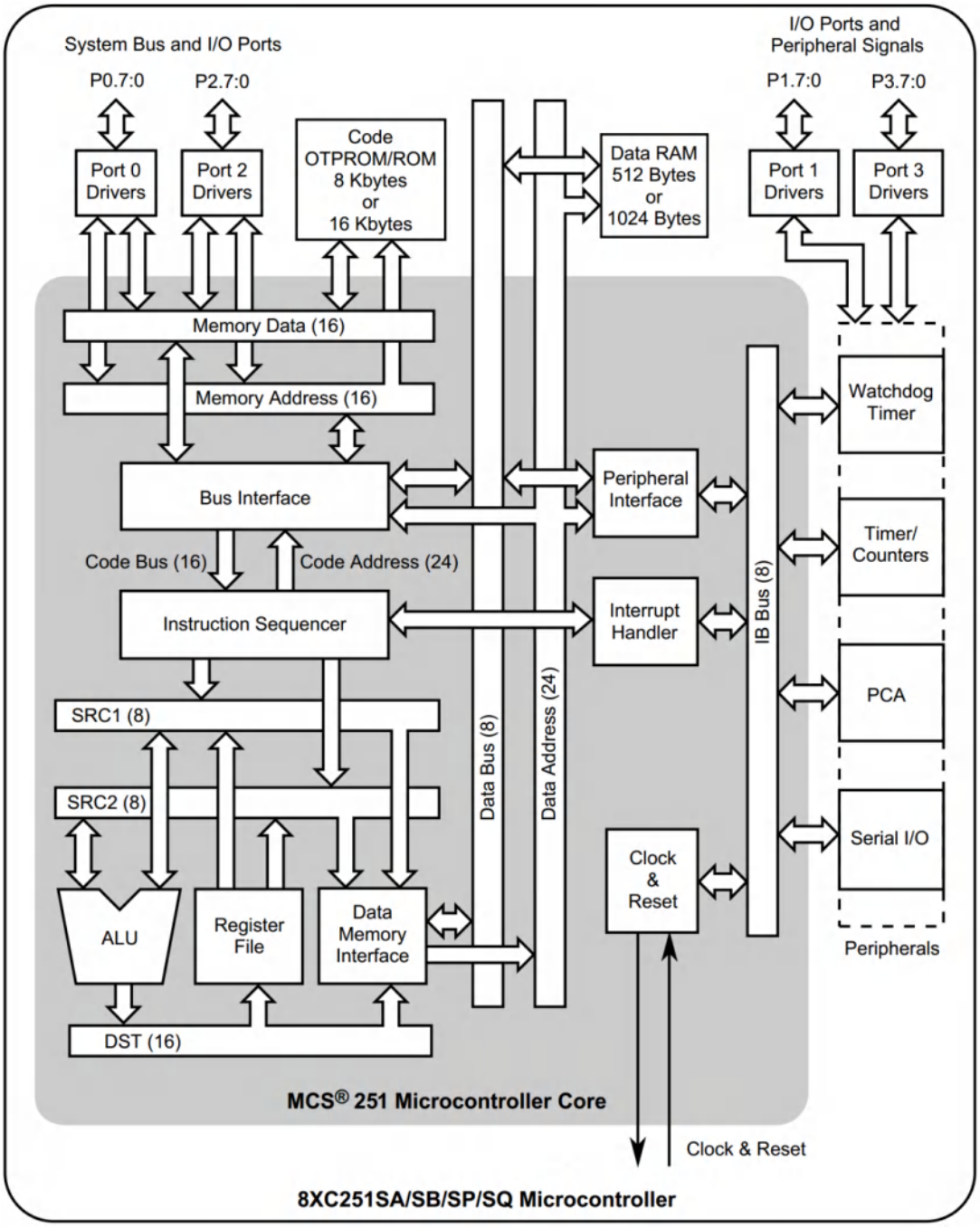

获取下一条指令
产生了中断请求
解码当前指令
执行完成当前
特殊指令
当前指令是否是
特殊指令?
是
否
执行完成下一条
特殊指令
执行完成当前指令
同时解码下一条指令
下一条指令是否是
特殊指令?
是
否
执行完成
下一条指令
响应中断
中断服务程序
特殊指令：
中断返回指令RETI或者是修改
中断使能寄存器、中断请求寄存器或者
中断优先级寄存器的特殊指令

附录B　　Ai8051U 系列和 Intel-80251 内部结构图对比

Ai8051U 系列内部的数据总线宽度是 32 位（兼容 16/8/1 位）。内部结构图如下：



Intel 的 80251 芯片内部数据总线宽度是 8 位。内部结构图如下：



Ai8051U 系列芯片内部则是采样 32 位宽度的数据总线，从而确保 32 位的 edata 数据可以一个 CPU 时钟就可完成数据读写。而 Intel 的 80251，由于制造工艺和成本的原因，芯片内部的数据总线采样的 8 位宽度的设计，同样 32 位的 edata 数据的读写，Intel 的 80251 则需要读写 4 次，再进行数据合并。

附录C　应用注意事项

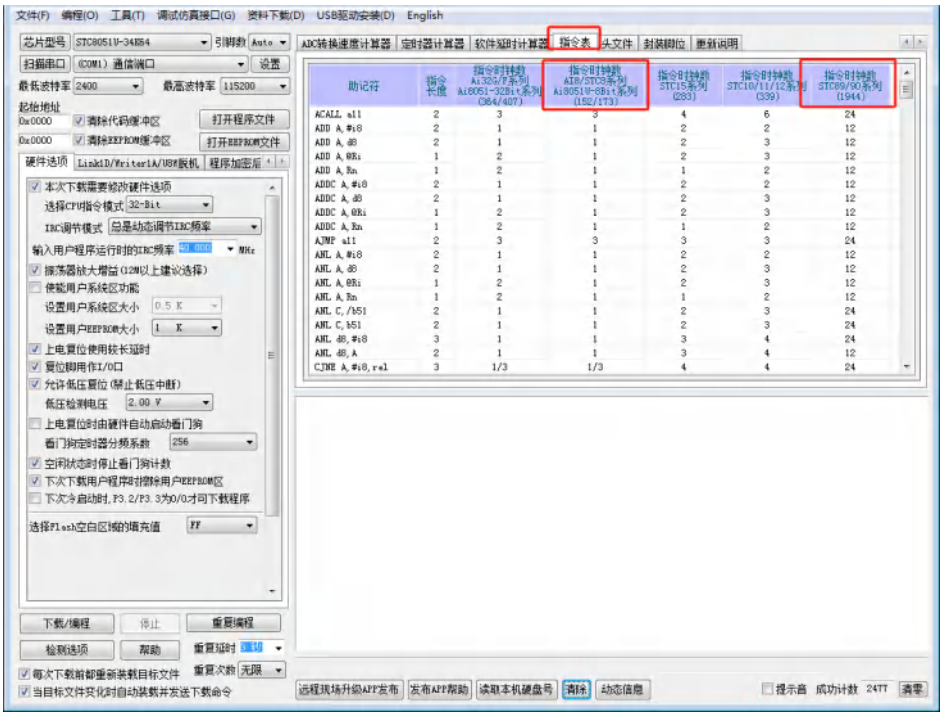
C.1　关于 Ai8051U 系列的注意事项

1. 之前的包含有 USB 功能的芯片中，IE2 寄存器中的 bit7（EUSB）位为只写寄存器，不可读取，读取时始终为 0。Ai8051U 系列芯片 IE2 寄存器中的 bit7（EUSB）位已修改为可读可写寄存器。
2. Ai8051U 系列单片机的比较器输入脚正端为：P4.6、P5.0、P5.1 和 ADC 输入口，负端为：P4.4 和内部 VREF，之前版本的手册有误。
3. SPI 发送和接收的 FIFO 深度均为 4 字节。设置 FIFOEN 可以减少 SPI 数据收发间隔时间，提高 SPI 收发速度。FIFOEN 只有在使用 SPI+DMA 传输时才可以使能，使用普通 SPI 收发时需要关闭 FIFOEN 才能收到 SPIF 标志。
4. 特别提示：如果使能 Ai8051U 系列的 PCA 中断时，PCA 的中断优先级固定为最高优先级，软件不能设置 PPCA（IP.7）和 PPCAH（IPH.7）来调整 PCA 的中断优先级，否则会导致无法产生 PCA 中断。
5. Ai8051U 系列单片机仿真建议使用 USB 直接通讯模式或者串口直接通讯模式进行仿真。硬件 SWD 仿真有如下问题：单步 LCALL 指令和 ACALL 指令这两条指令时会出现异常。所以暂不建议使用 SWD 硬件仿真，建议使用 USB 直接仿真或者串口直接仿真。
6. RTC 模块的星期功能暂不可用。
7. 高级 PWM 输出的波形从有占空比（占空比非 0%）到无占空比（占空比为 0%）时会有一个 1T 宽度的毛刺。
8. QSPI 当使用 DMA 模式操作完成 QSPI 外设后，如果需要使用寄存器方式再操作 QSPI，需要先清除一下 QSPI 的状态寄存器 QSPI_SR1 中的 TCF 和 TCEF 位。
9. **P2P 使用注意事项:** 目前测试发现 Ai8051U 的 QSPI 到其他外设的 P2P 方式 DMA 会影响 XRAM 的数据，在使用 QSPI 到其他外设的 P2P 功能时，**必须将 QSPI 存储接收数据的 XRAM 起始地址设置到 32K　XRAM 的后面部分。注意：由于内部的 XRAM 只有 32K，有效的地址范围为 0000H~7FFFH，当地址超出 32K 时，地址会重新回到 0，所以设置的 DMA 的 XRAM 起始地址+DMA 的传输数据长度的总和不能大于 32K。**
10. 寄存器{DMA_QSPI_RXAH,　DMA_QSPI_RXAL}中设置的是每次 DMA 读取 XRAM 的起始地址，在 DMA 过程中{DMA_QSPI_RXAH,　DMA_QSPI_RXAL}的值不会改变，所以只需要在初始化时设置好{DMA_QSPI_RXAH,　DMA_QSPI_RXAL}中的初始值就可以了，不需要每次 DMA 时都设置。

附录D　Ai8051U 系列单片机取代 STC89 系列的注意事项

■ 单片机指令

Ai8051U-8Bit 系列的指令码与 STC89 系列是完全一致的，所以 STC89 系列的代码移植到 Ai8051U 上，运行依然正确，但 Ai8051U-8Bit 系列的指令速度比 STC89 系列要快，Ai8051U-8Bit 系列的大部分指令执行都只需要一个 CPU 时钟。如果用户代码中有指令延时的代码，则需要进行调整。有个每条指令的对比可参考 STC 下载软件的指令表，如下图：



■ I/O 口

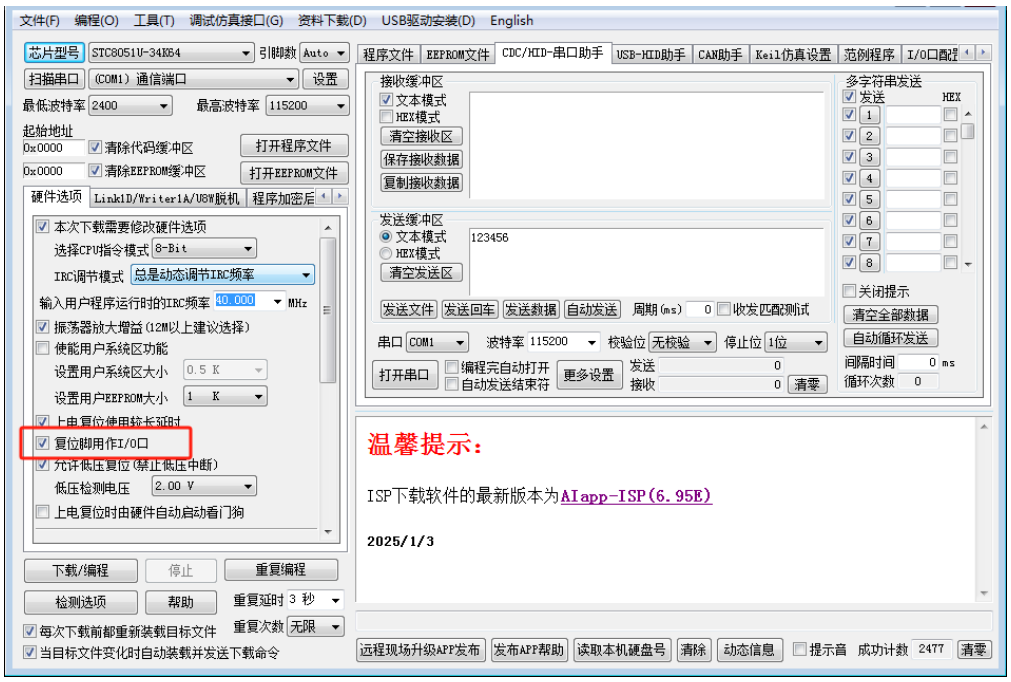
Ai8051U-8Bit 系列单片机上电后，I/O 的模式与 STC89 系列不一样。STC89 系列单片机所有 I/O 口上电后都是 8051 的准双向口模式，而 Ai8051U-8Bit 系列单片机的 I/O 中，除了 ISP 下载脚 P3.0/P3.1 为准双向口模式外，其余的所有 I/O 口在上电后都是高阻输入模式。传统的 8051 和 STC89 系列单片机上电后即为准双向口模式并输出高电平，经常有客户的系统中使用 I/O 驱动马达或者 LED 灯，因此会出现单片机上电的瞬间马达会动一下或者 LED 会闪一下。Ai8051U-8Bit 系列的 I/O 上电后为高阻输入模式，就可避免马达和 LED 的这种误动作。

由于 Ai8051U-8Bit 系列单片机的 I/O 中，除了 ISP 下载脚 P3.0/P3.1 为准双向口模式外，其余的所有 I/O 口在上电后都是高阻输入模式，所以当用户需要 Ai8051U-8Bit 系列的 I/O 口向外输出信号前，必须先使用 PxM0 和 PxM1 两个寄存器对 I/O 的工作模式进行设置。

■ 复位脚

Ai8051U-8Bit 系列的复位脚一般情况下是当作普通 I/O 口 P4.7 使用的，当用户在 ISP 下载时关

闭了复位脚当作 I/O 口使用时，P4.7 口则为单片机的复位脚（RESET 脚）。而 STC89 系列的复位脚则固定为复位脚。复位脚配置方式如下图：



关于复位脚的复位电平，Ai8051U-8Bit 系列和 STC89 系列也不一样。STC89 系列复位脚为高电平时单片机处于复位状态，低电平时单片机解除复位状态。而 Ai8051U-8Bit 系列与 STC89 系列的复位电平是向反的，即对于 Ai8051U-8Bit 系列，复位脚为低电平时，单片机处于复位状态，高电平时单片机解除复位状态。

所以当用户使能 P4.7 口的复位脚功能是需要注意复位电平的问题。

■ EEPROM

Ai8051U-8Bit 系列的 EEPROM 操作寄存器与 STC89 系列的 EEPROM 操作寄存器地址不一样。EEPROM 擦除和编程操作的等待时间设置也不一样。STC89 系列的 EEPROM 擦除和编程的等待时间用寄存器 IAP_CONTR 的 Bit2-Bit0 设置，设置的只是一个大概的频率范围值，Ai8051U-8Bit 系列新增了一个寄存器 IAP_TPS（SFR 地址：0F5H），专用于设置 EEPROM 擦除和编程的等待时间，且用户不需要去计算，只需要根据当前 CPU 的工作频率，直接填入 IAP_TPS 即可，硬件会自动计算等待时间。（比如：当前 CPU 的工作频率为 24MHz，则只需要向 IAP_TPS 填入 24 即可）

■ 定时器

Ai8051U-8Bit 系列的定时器 0 的模式 0 为 16 位自动重载模式，STC89 系列的定时器 0 的模式 0 为 13 位模式；

Ai8051U-8Bit 系列的定时器 0 的模式 3 为 16 位自动重载模式且为不可屏蔽中断，STC89 系列的定时器 0 的模式 3 为双 8 位模式；

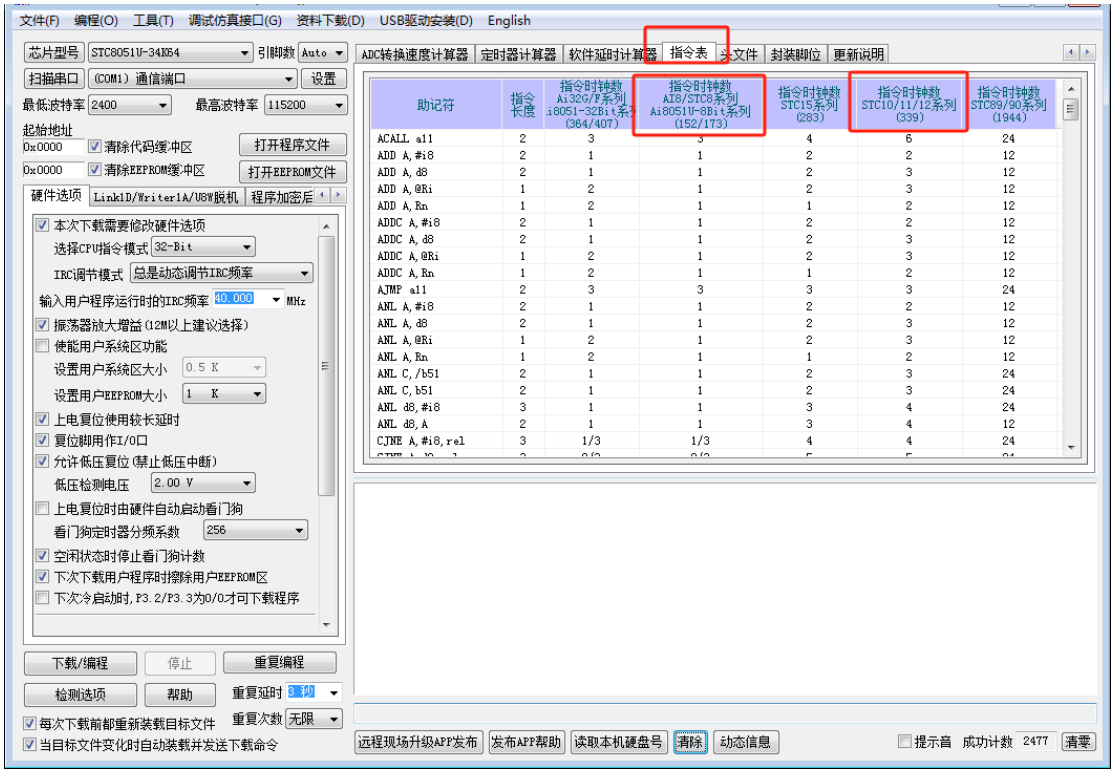
Ai8051U-8Bit 系列的定时器 1 的模式 0 为 16 位自动重载模式，STC89 系列的定时器 1 的模式 0 为 13 位模式；

Ai8051U-8Bit 系列的定时器 2 与 STC89 系列的定时器 2 不兼容

附录E　Ai8051U 系列单片机取代 STC12C5A60S2 系列的注意事项

■ 单片机指令

Ai8051U-8Bit 系列的指令码与 STC12C5A60S2 系列是完全一致的，所以 STC12C5A60S2 系列的代码移植到 Ai8051U 上，运行依然正确，但 Ai8051U-8Bit 系列的指令速度比 STC12C5A60S2 系列要快，Ai8051U-8Bit 系列的大部分指令执行都只需要一个 CPU 时钟。如果用户代码中有指令延时的代码，则需要进行调整。有个每条指令的对比可参考 STC 下载软件的指令表，如下图：



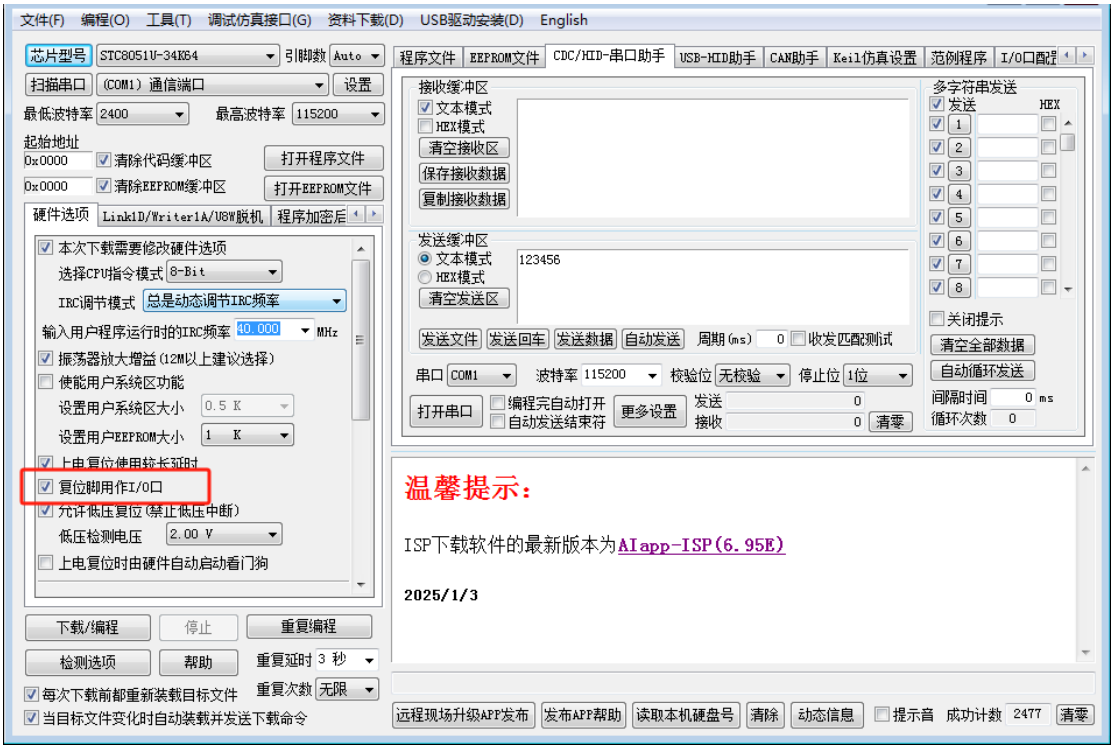
■ I/O 口

Ai8051U-8Bit 系列单片机上电后，I/O 的模式与 STC12C5A60S2 系列不一样。STC12C5A60S2 系列单片机所有 I/O 口上电后都是 8051 的准双向口模式，而 Ai8051U-8Bit 系列单片机的 I/O 中，除了 ISP 下载脚 P3.0/P3.1 为准双向口模式外，其余的所有 I/O 口在上电后都是高阻输入模式。传统的 8051 和 STC12C5A60S2 系列单片机上电后即为准双向口模式并输出高电平，经常有客户的系统中使用 I/O 驱动马达或者 LED 灯，因此会出现单片机上电的瞬间马达会动一下或者 LED 会闪一下。Ai8051U-8Bit 系列的 I/O 上电后为高阻输入模式，就可避免马达和 LED 的这种误动作。

由于 Ai8051U-8Bit 系列单片机的 I/O 中，除了 ISP 下载脚 P3.0/P3.1 为准双向口模式外，其余的所有 I/O 口在上电后都是高阻输入模式，所以当用户需要 Ai8051U-8Bit 系列的 I/O 口向外输出信号前，必须先使用 PxM0 和 PxM1 两个寄存器对 I/O 的工作模式进行设置。

■ 复位脚

Ai8051U-8Bit 系列的复位脚一般情况下是当作普通 I/O 口 P4.7 使用的，当用户在 ISP 下载时关闭了复位脚当作 I/O 口使用时，P4.7 口则为单片机的复位脚（RESET 脚）。而 STC12C5A60S2 系列的复位脚则固定为复位脚。复位脚配置方式如下图：



关于复位脚的复位电平，Ai8051U-8Bit 系列和 STC12C5A60S2 系列也不一样。STC12C5A60S2 系列复位脚为高电平时单片机处于复位状态，低电平时单片机解除复位状态。而 Ai8051U-8Bit 系列与 STC89 系列的复位电平是向反的，即对于 Ai8051U-8Bit 系列，复位脚为低电平时，单片机处于复位状态，高电平时单片机解除复位状态。

所以当用户使能 P4.7 口的复位脚功能是需要注意复位电平的问题。

■ ADC

Ai8051U-8Bit 系列和 STC15 系列的 ADC_CONTR 寄存器定义、ADC_RES、ADC_RESL3 个寄存器地址相同的。但 STC8H 系列另外新增加了两个寄存器：ADCCFG 和 ADCTIM。

STC15 系列开始 ADC 转换位 ADC_START 位于寄存器 ADC_CONTR 的 BIT3，而 STC8H 系列的位于 ADC_CONTR 的 BIT6

STC15 系列 ADC 转换完成标志位 ADC_FLAG 位于寄存器 ADC_CONTR 的 BIT4，而 STC8H 系列的位于 ADC_CONTR 的 BIT5

STC15 系列 ADC 速度控制为 ADC_SPEED 位于寄存器 ADC_CONTR 的 BIT6-BIT5，而 STC8H 系列的位于 ADCCFG 的 BIT3-BIT0

STC15 系列 ADC 转换结果的对齐控制位 ADRJ 位于寄存器 CLK_DIV 的 BIT5，而 STC8H 系列的对齐控制位 RESFMT 位于 ADCCFG 的 BIT5

STC8H 系列新增了更为精准的 ADC 转换时序控制机制，通过寄存器 ADCTIM 进行设置

■ EEPROM

Ai8051U-8Bit 系列的 EEPROM 操作寄存器与 STC12C5A60S2 系列的 EEPROM 操作寄存器地址不一样。EEPROM 擦除和编程操作的等待时间设置也不一样。STC12C5A60S2 系列的 EEPROM 擦除和编程的等待时间用寄存器 IAP_CONTR 的 Bit2-Bit0 设置，设置的只是一个大概的频率范围值，Ai8051U-8Bit 系列新增了一个寄存器 IAP_TPS（SFR 地址：0F5H），专用于设置 EEPROM 擦除和编程的等待时间，且用户不需要去计算，只需要根据当前 CPU 的工作频率，直接填入 IAP_TPS 即可，硬件会自动计算等待时间。（比如：当前 CPU 的工作频率为 24MHz，则只需要向 IAP_TPS 填入 24 即可）

■ 定时器

Ai8051U-8Bit 系列的定时器 0 的模式 0 为 16 位自动重载模式，STC12C5A60S2 系列的定时器 0 的模式 0 为 13 位模式；

Ai8051U-8Bit 系列的定时器 0 的模式 3 为 16 位自动重载模式且为不可屏蔽中断，STC12C5A60S2 系列的定时器 0 的模式 3 为双 8 位模式；

Ai8051U-8Bit 系列的定时器 1 的模式 0 为 16 位自动重载模式，STC12C5A60S2 系列的定时器 1 的模式 0 为 13 位模式；

■ 串口 2

STC12C5A60S2 系列的串口 2 有专用波特率发生器，Ai8051U-8Bit 系列的无专用波特率发生器，Ai8051U-8Bit 系列的串口 2 可以使用定时器 2 当作波特率发生器。Ai8051U-8Bit 系列的串口 2 波特率计算公式与 STC12C5A60S2 系列的串口 2 计算公式不一样。

■ SPI

Ai8051U-8Bit 系列的 SPI 速度与 STC12C5A60S2 系列的 SPI 的速度不兼容。Ai8051U-8Bit 系列的 SPI 速度分别为系统时钟/4/8/16/2，STC12C5A60S2 系列的 SPI 速度分别为系统时钟/4/16/64/128。

附录F　7.4.12 固件版本说明

固件版本为 7.4.12 及更高版本的芯片启动时有如下注意事项：

1、芯片上电会将 P3.2 口的 4K 上拉电阻打开，直至芯片复位到用户程序。

2、芯片上电等待时间与旧版本相比有变化

- 如果上电时 P3.2 口为高电平，则会等待 1ms 后，进入之前版本的串口下载流程；
- 如果上电时 P3.2 口为低电平，且检测到 P3.1 口为低电平，则立即进入 USB 下载流程；
- 如果上电时 P3.2 口为低电平，且 P3.1 口维持高电平，若检测到 P3.0 有 10 次高低电平变化，则进入之前版本的串口下载流程；
- 如果上电时 P3.2 口为低电平，且 P3.0 和 P3.1 都一直维持高电平无变化，则最长会等待约 0.5s 的 USB 检测时间，超时后进入之前版本的串口下载流程。

相关的单片机系列如下：Ai8051U 系列、STC32G12K128 系列、STC8H8K64U 系列、Ai8H8K64U 系列、STC8H2K12U 系列、Ai8H2K12U 系列、STC8H2K32U 系列、Ai8H2K32U 系列。

附录G 电气特性

绝对最大额定值

参数	最小值	最大值	单位	说明
存储温度	-55	+155	℃	
工作温度	-40	+85	℃	使用内部高速 IRC 和外部晶振
	-40	+125	℃	当温度高于 85℃时请使用外部晶振， 且工作频率建议控制在 24M 以下
工作电压	1.9	5.5	V	当工作温度低于-40℃时，工作电压不得低于 3.0V
VDD 对地电压	-0.3	+5.5	V	
I/O 口对地电压	-0.3	VDD+0.3	V	

直流特性（VSS=0V，VDD=5.0V，测试温度=25℃）

标号	参数	范围				测试环境
		最小值	典型值	最大值	单位	
I_{PD}	主时钟停振/省电模式电流	-	0.6	-	uA	
I_{WKT}	掉电唤醒定时器	-	4.4	-	uA	
I_{LVD}	低压检测模块	-	430	-	uA	
I_{IDL}	空闲模式电流（6MHz）	-	0.99	-	mA	
	空闲模式电流（11.0592MHz）	-	1.14	-	mA	
	空闲模式电流（24MHz）	-	1.37	-	mA	
	空闲模式电流（内部 32KHz）	-	0.57	-	mA	
I_{NOR}	正常模式电流（6MHz）	-	1.65	-	mA	
	正常模式电流（11.0592MHz）	-	2.29	-	mA	
	正常模式电流（24MHz）	-	3.49	-	mA	
	正常模式电流（内部 32KHz）	-	0.57	-	mA	
I_{CC}	普通工作模式电流	-	4	20	mA	
V_{IL1}	输入低电平	-	-	1.32	V	打开施密特触发
		-	-	1.48	V	关闭施密特触发
V_{IH1}	输入高电平（普通 I/O）	1.60	-	-	V	打开施密特触发
		1.54	-	-	V	关闭施密特触发
I_{OL1}	输出低电平的灌电流	-	20	-	mA	端口电压 0.45V
I_{OH1}	输出高电平电流（双向模式）	-	200	-	uA	
I_{OH2}	输出高电平电流（推挽模式）	-	20	-	mA	端口电压 2.4V
I_{IL}	逻辑 0 输入电流	-	-	50	uA	端口电压 0V
I_{TL}	逻辑 1 到 0 的转移电流	100	270	600	uA	端口电压 2.0V
R_{PU}	I/O 口上拉电阻	4.1	4.2	4.4	KΩ	

I/O 速度	I/O 大电流驱动，I/O 快速转换		36		MHz	PxDR=0, PxSR=0
	I/O 小电流驱动，I/O 快速转换		32		MHz	PxDR=1, PxSR=0
	I/O 大电流驱动，I/O 慢速转换		26		MHz	PxDR=0, PxSR=1
	I/O 小电流驱动，I/O 慢速转换		22		MHz	PxDR=1, PxSR=1
比较器	最快速度		10		MHz	关闭所有模拟和数字滤波
	模拟滤波时间		0.1		us	
	数字滤波时间		0		系统时钟	LCDTY＝0
			n+2			LCDTY=n (n=1~63)
I_{PD2}	使能比较器时主时钟停振/省电模式功耗	-	460	-	uA	
I_{PD3}	使能 LVD 时主时钟停振/省电模式功耗	-	520	-	uA	

直流特性（VSS=0V，VDD=3.3V，测试温度=25℃）

标号	参数	范围				测试环境
		最小值	典型值	最大值	单位	
I_{PD}	主时钟停振/省电模式电流	-	0.4	-	uA	
I_{WKT}	掉电唤醒定时器	-	1.5	-	uA	
I_{LVD}	低压检测模块	-	364	-	uA	
I_{IDL}	空闲模式电流（6MHz）	-	0.89	-	mA	
	空闲模式电流（11.0592MHz）	-	1.05	-	mA	
	空闲模式电流（24MHz）	-	1.28	-	mA	
	空闲模式电流（内部 32KHz）	-	0.47	-	mA	
I_{NOR}	正常模式电流（6MHz）	-	1.55	-	mA	
	正常模式电流（11.0592MHz）	-	2.19	-	mA	
	正常模式电流（24MHz）	-	3.38	-	mA	
	正常模式电流（内部 32KHz）	-	0.47	-	mA	
I_{CC}	普通工作模式电流	-	4	20	mA	
V_{IL1}	输入低电平	-	-	0.99	V	打开施密特触发
		-	-	1.07	V	关闭施密特触发
V_{IH1}	输入高电平（普通 I/O）	1.18	-	-	V	打开施密特触发
		1.09	-	-	V	关闭施密特触发
I_{OL1}	输出低电平的灌电流	-	20	-	mA	端口电压 0.45V
I_{OH1}	输出高电平电流（双向模式）	-	110	-	uA	
I_{OH2}	输出高电平电流（推挽模式）	-	20	-	mA	端口电压 2.4V
I_{IL}	逻辑 0 输入电流	-	-	50	uA	端口电压 0V
I_{TL}	逻辑 1 到 0 的转移电流	100	270	600	uA	端口电压 2.0V
R_{PU}	I/O 口上拉电阻	5.8	5.9	6.0	KΩ	

I/O 速度	I/O 大电流驱动，I/O 快速转换		25		MHz	PxDR=0, PxSR=0
	I/O 小电流驱动，I/O 快速转换		22		MHz	PxDR=1, PxSR=0
	I/O 大电流驱动，I/O 慢速转换		16		MHz	PxDR=0, PxSR=1
	I/O 小电流驱动，I/O 慢速转换		12		MHz	PxDR=1, PxSR=1
比较器	最快速度		10		MHz	关闭所有模拟和数字滤波
	模拟滤波时间		0.1		us	
	数字滤波时间		0		系统时钟	LCDTY＝0
			n+2			LCDTY=n (n=1~63)
I_{PD2}	使能比较器时主时钟停振/省电模式功耗	-	400	-	uA	
I_{PD3}	使能 LVD 时主时钟停振/省电模式功耗	-	470	-	uA	

内部 IRC 温漂特性（参考温度 25℃）

温度	范围		
	最小值	典型值	最大值
-20℃~65℃		-0.88%~+1.05%	
-40℃~85℃		-1.38%~+1.42%	
-40℃~125℃		-3%~+3%	

低压复位门槛电压（测试温度 25℃）

级别	电压		
	最小值	典型值	最大值
POR		1.9V	
LVR0		2.0V	
LVR1		2.4V	
LVR2		2.7V	
LVR3		3.0V	

附录H　STCAI Reliability Report V1.2

Date : 2025/02/21

可靠性参考数据，安全参数
由同生产制造工艺，已通过车规 AEC-Q100 Grade1 的 STC32G12K128-24A 类比获得

- Under Temperature 55°C

MCU 型号（55°C）	FIT（1E9 hr/ea）	MTTF（device hr）	PFH(1/h, DFR=20%)
STC8H1K28-36I	~160	6,244,566.8	6.4E−9
AI8H1K28-36I	~160	6,244,566.8	6.4E−9
STC8H2K32U-45I	~160	6,244,566.8	6.4E−9
AI8H2K32U-45I	~160	6,244,566.8	6.4E−9

参数	数值 (1/h)（55°C）
λ（失效率）	1.60E−7
DFR（缺陷失效率）	20%
λs（安全失效率）	1.28E−7
λd（危险失效率）	3.20E−8
λdd（可检测危险失效率）	2.56E−8
λdu（不可检测危险失效率）	6.4E−9
SFF（安全失效率比）	96%
DC（诊断覆盖率）	80%
HFT（硬件故障公差)	0 (单 MCU，无冗余)

Note:

- The report is based on confidence limit 95%
- MTTF 6,244,566.8 device hr 相当于 712.85 年
- 随着温度上升 MTTF 的值会下降

- Under Temperature 85°C

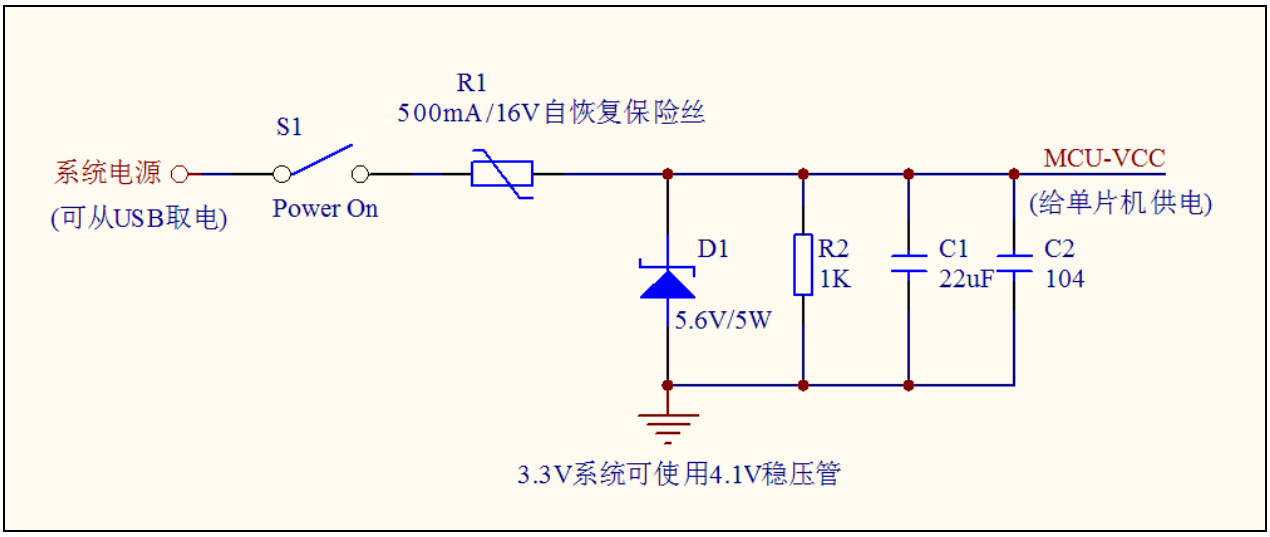
MCU 型号（85°C）	FIT（1E9 hr/ea）	MTTF（device hr）	PFH(1/h, DFR=20%)
STC8H1K28-36I	~1276	783,575.4	5.1E−8
AI8H1K28-36I	~1276	783,575.4	5.1E−8
STC8H2K32U-45I	~1276	783,575.4	5.1E−8
AI8H2K32U-45I	~1276	783,575.4	5.1E−8

参数	数值 (1/h)（85°C）
λ（失效率）	1.28E−6
DFR（缺陷失效率）	20%
λs（安全失效率）	1.02E−6
λd（危险失效率）	2.55E−7
λdd（可检测危险失效率）	2.04E−7
λdu（不可检测危险失效率）	5.1E−8
SFF（安全失效率比）	96%
DC（诊断覆盖率）	80%
HFT（硬件故障公差）	0 (单 MCU，无冗余)

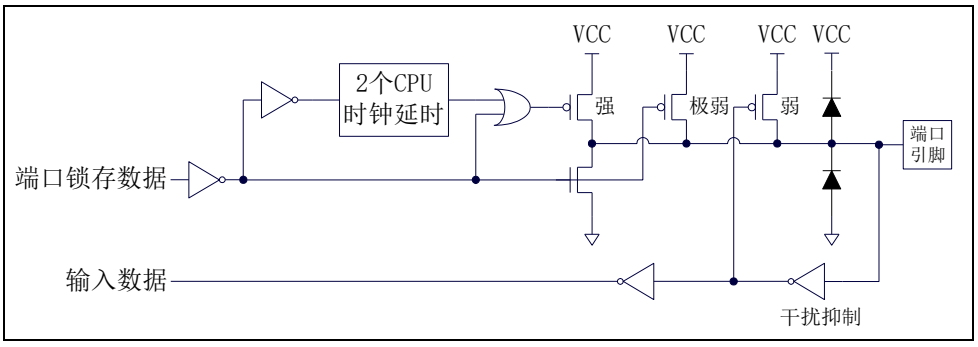
Note:

- The report is based on confidence limit 95%
- MTTF 783.575.4 device hr 相当于 89.45 年
- 随着温度上升 MTTF 的值会下降

附录I　单片机电源系统最简易自我保护电路



I/O 的内部结构图



附录J　关于回流焊前是否要烘烤

根据国际湿气敏感性等级 3（MSL3）规范的要求，贴片元器件在拆开真空包装后，168 小时内，7 天内，必须回流焊贴片完成，如未完成，必须再次高温烘烤。

LQFP/QFN/DFN 托盘能耐 100 度以上的高温，拆开真空包装后 7 天内必须回流焊贴片完成，否则回流焊前必须重新烘烤：110~125℃，4~8 个小时都可以。

SOP/TSSOP 塑料管耐不了 100 度以上的高温，拆开真空包装后 7 天内必须回流焊贴片完成，否则回流焊前去除耐不了 100 度以上高温的塑料管，放到金属托盘中，重新烘烤：110~125℃，4~8 个小时都可以，再人工放回塑料管。所以尽量选择 LQFP/QFN/DFN 封装形式，方便重新烘烤。

STCAI 贴片 MCU 最小真空包装数量：

真空托盘包装，是最高等级包装，可保存 N 年，回流焊前方便重新烘烤。拆开真空包装后，7 天内必须贴片完成！托盘耐高温 150 度，包装成本高。

LQFP64：　最小真空包装，1600pcs/包；拆开最小真空包装后，160pcs/盘
LQFP48：　最小真空包装，2500pcs/包；拆开最小真空包装后，250pcs/盘
LQFP44：　最小真空包装，1600pcs/包；拆开最小真空包装后，160pcs/盘
LQFP32：　最小真空包装，2500pcs/包；拆开最小真空包装后，250pcs/盘

QFN64/8*8：最小真空包装，2600pcs/包；拆开最小真空包装后，260pcs/盘
QFN48/6*6：最小真空包装，4900pcs/包；拆开最小真空包装后，490pcs/盘
QFN32/4*4：最小真空包装，4900pcs/包；拆开最小真空包装后，490pcs/盘
QFN20/3*3：最小真空包装，4900pcs/包；拆开最小真空包装后，490pcs/盘
DFN8/3*3：　最小真空包装，4900pcs/包；拆开最小真空包装后，490pcs/盘

管装真空包装，塑料管是普通材料，不耐高温，包装成本低，回流焊前不方便重新烘烤！拆开真空包装后，7 天内必须贴片完成！

SOP28：　最小真空包装，2080pcs/包；拆开最小真空包装后，26pcs/管
SOP20：　最小真空包装，2880pcs/包；拆开最小真空包装后，36pcs/管
TSSOP28：最小真空包装，7500pcs/包；拆开最小真空包装后，50pcs/管
TSSOP20：最小真空包装，14400pcs/包；拆开最小真空包装后，72pcs/管
SOP16：　最小真空包装，5000pcs/包；拆开最小真空包装后，50pcs/管
SOP8：　最小真空包装，10000pcs/包；拆开最小真空包装后，100pcs/管

请按最小包装的整数倍买，否则零头部分，回流焊前需要重新烘烤。

【举例 1】：TSSOP20，你买了 1000pcs，你以为是整数，其实是零头，因为最小真空包装是 14400pcs/包，你拿到的并不是一个最小真空包装，而是已拆开真空包装不知道多少天的零头部分，必须重新烘烤。
【举例 2】：TSSOP20，你买了 10000pcs，你以为是整数，其实是零头，因为最小真空包装是 14400pcs/包，你拿到的并不是一个最小真空包装，而是已拆开真空包装不知道多少天的零头部分，必须重新烘烤。
【举例 3】：LQFP64，你买了 2000pcs，则拿到的是 1 个最小真空包装/1600 + 400 个零头
===400 个零头你回流焊前需重新烘烤；
===LQFP64：最小真空包装，1600pcs/包；拆开最小真空包装后，160pcs/盘

编带真空包装，编带是普通材料，不耐高温，包装成本低，回流焊前不方便重新烘烤！拆开真空包装后，7 天内必须贴片完成！据说回流焊前，可以 55℃烘烤 6 小时，再贴片 / 回流焊。

TSSOP20，　编带：最小真空包装，3000pcs/卷；
SOP16，　编带：最小真空包装，4000pcs/卷。
SOP8，　编带：最小真空包装，4000pcs/卷。

附录K　如何使用万用表检测芯片 I/O 口好坏

根据国际湿气敏感性等级 3（MSL3）规范的要求，贴片元器件在拆开真空包装后，168 小时内，7 天内，必须回流焊贴片完成，如未完成，必须再次高温烘烤。如果没有高温烘烤的流程，直接进行回流焊，则可能由于芯片内外受热不均导致芯片内部金属线被拉断，最终出现的现象是芯片 I/O 口损坏。

STC 的单片机在芯片设计时，每个 I/O 口都有两个分别接到 VCC 和 GND 的保护二极管，用万用表的二极管检测档可以进行测量。可使用此方法简单判断 I/O 管脚的好坏情况。使用万用表测量方法如下（注：这里使用的是数字万用表）

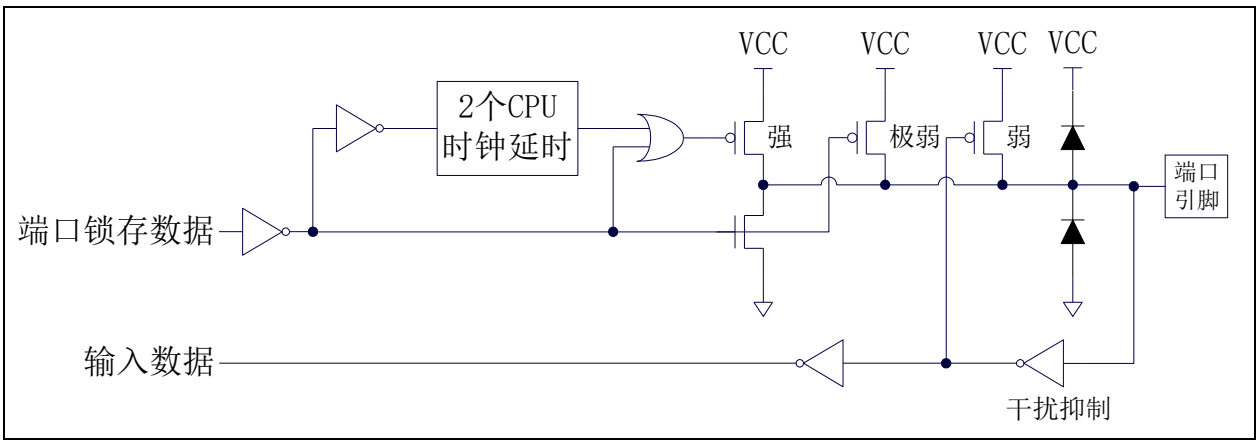
首先将万用表调到二极管检测档，被测芯片不要供电，将万用表的红表笔连接到被测芯片的 **GND 管脚**，黑表笔依次测量每个 I/O 口，如果万用表显示的参数为 0.45～0.75V 左右，则表示芯片的内部 I/O 到 GND 的保护二极管正常，即打线也是完好的，若显示的参数为开路/断开的状态，则表示芯片内部的打线已断路。

上面是检测芯片内部的打线情况的方法。

另外，如果用户板上，单片机的管脚没有加保护电路，一旦出现过流或者过压都可能导致 I/O 烧坏。为了检测管脚是否被烧坏，除了使用上面的方法检测 I/O 口到 GND 的保护二极管外，还需要检测 I/O 口到 VCC 的保护二极管。使用万用表检测 I/O 口到 VCC 的保护二极管的方法如下：

首先将万用表调到二极管检测档，被测芯片不要供电，将万用表的黑表笔连接到被测芯片的 **VCC 管脚**，红表笔依次测量每个 I/O 口，如果万用表显示的参数为 0.45～0.75V 左右，则表示芯片的内部 I/O 到 VCC 的保护二极管正常，若显示的参数为开路/断开的状态，则表示芯片此端口已被损坏。

I/O 的内部结构图



附录L　大批量生产，如何省去专门的烧录人员，如何无烧录环节

大批量生产，你在将由 MCU 作为主控芯片的控制板组装到设备里面之前在你将 MCU 贴片到你的控制板完成之后，你必须测试你的控制板的好坏。不要说 100%，直通无问题，那是抬杠，不是搞生产，只要生产，就会虚焊，短路，部分元件贴错，部分元件采购错。

所以在贴片回来后，组装到外壳里面之前，你必须要测试，你的含有 MCU 控制板的好坏，好的去组装，坏的去维修抢救。

测试，大批量生产，必须有测试架/下面接上我们的脱机烧录工具 U8W/U8W-Mini/USB-Link1D，还要接上其他控制部分

通过 USER-VCC、P3.0、P3.1、GND 连接，要工人每次都开电源
通过 S-VCC、P3.0、P3.1、GND 连接，不要你开电源，脱机工具给你自动供电

外面帮你做一个测试架的成本 500 元以下，就是有机玻璃，夹具，顶针。
1 个测试你控制板是否正常的工人管理 2 - 3 个 测试架

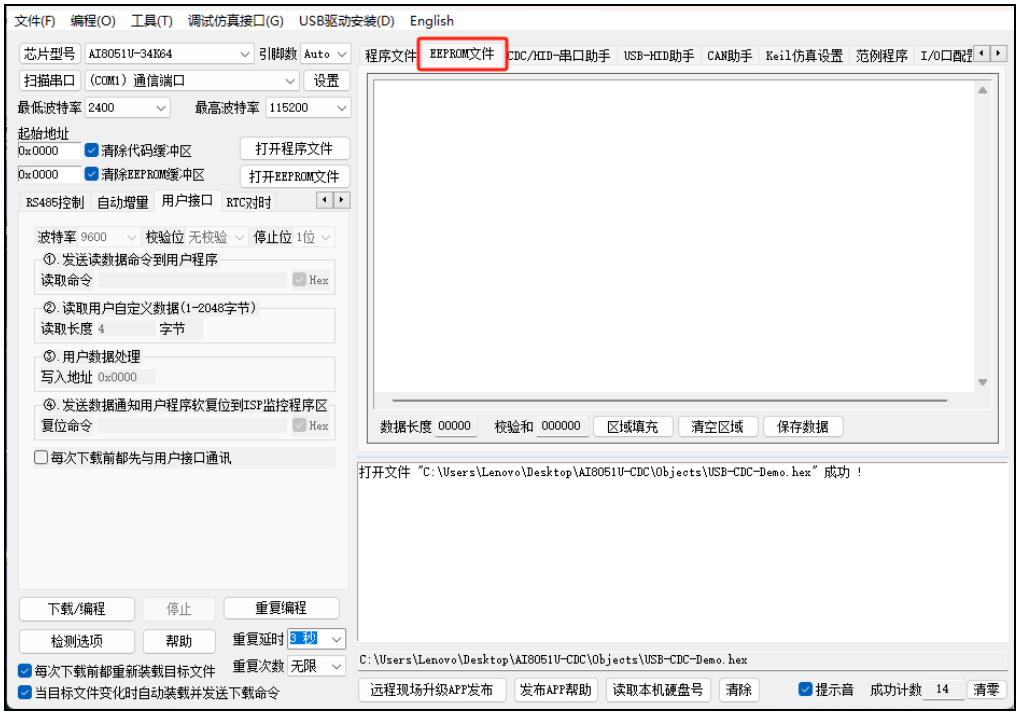
操作流程：

1、 将你的 MCU 控制板卡到测试架 1 上
2、 将你的 MCU 控制板卡到测试架 2 上，测试架 1 上的程序已烧录完成/感觉不到烧录时间
3、 测试 测试架 1 上的主控板功能是否正常，正常放到正常区，不正常，放到不正常区
4、 给测试架 1 卡上新的未测试的无程序的控制板
5、 测试 测试架 2 上的未测试控制板/程序不知何时早就不知不觉的烧好了，换新的未测试未烧录的控制板
6、 循环步骤 3~步骤 5

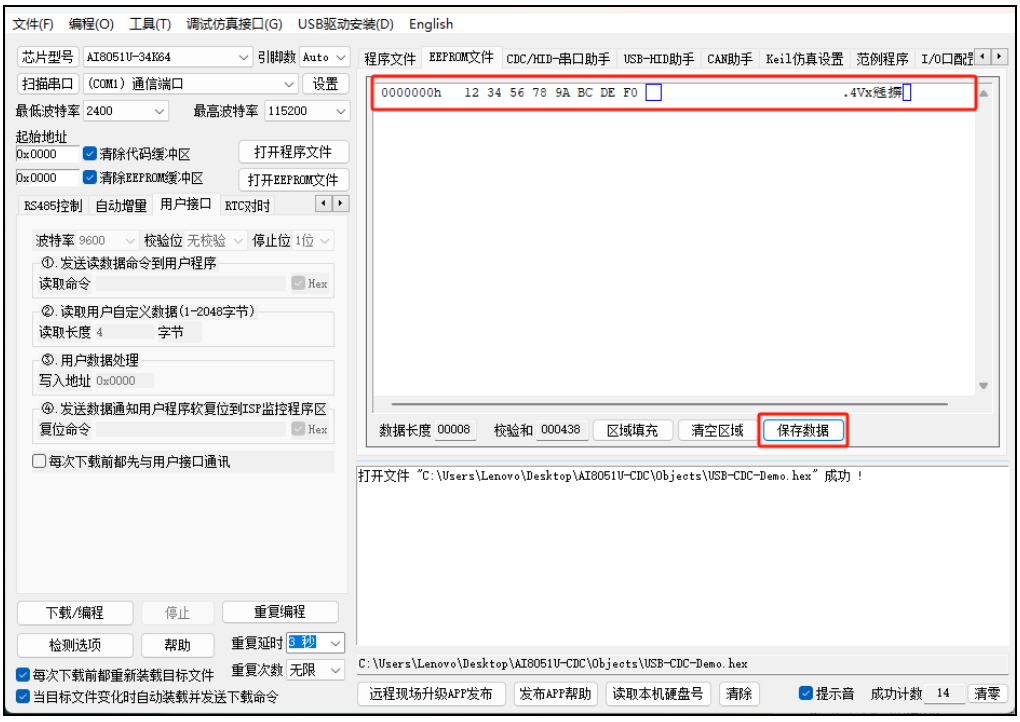
========不需要安排烧录人员

附录M　如何使用 ISP 下载软件制作和编辑 EEPROM 文件

打开任意版本 ISP 下载软件，选择“EEPROM”页面，单击数据窗口，如下所示



当出现黑色长方形的光标时，即可手动输入 16 进制数据，包括数字 0~9、字母 A~F（大小写通用）数据输入完成后，点击“保存数据”按钮即可保存 EEPROM 数据



附录N　开源汇编语言调用 USB-CDC 库文件实现 USB-CDC 虚拟串口通信

示例代码链接地址 ：

AI8H_cdc_demo_asm.zip

AI32G_cdc_demo_asm.zip

附录O　可以自己去生产的【USB 转双串口】资料

【USB转双串口】量产PCB/SCH开源，芯片出厂自带USB程序

Ai8H2K12U-45MHz-SOP8，USB转单串口，RMB0.95

Ai8H2K12U-45MHz-SOP16，USB转双串口，RMB1.1

USB插头支持：USB-TypeA、USB-TypeC

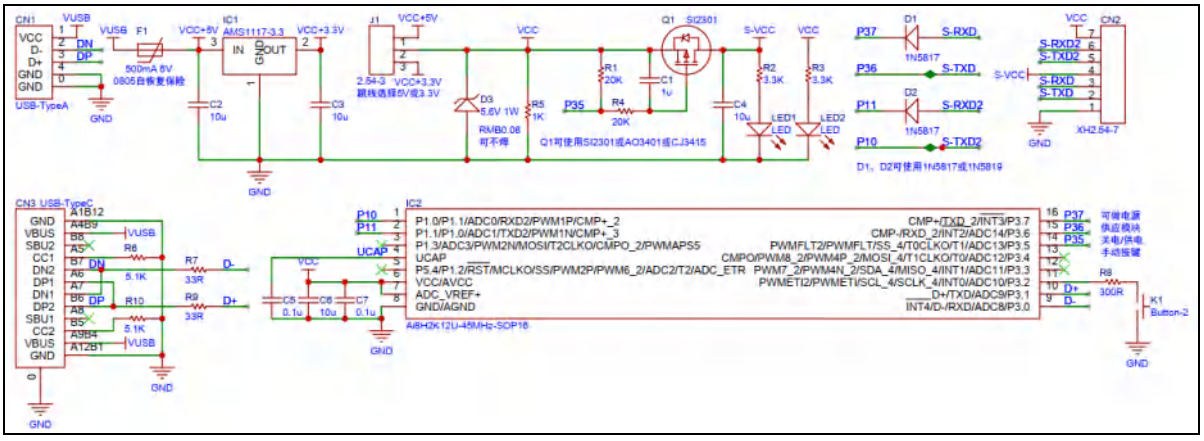
支持任意波特率，最高到10Mbps，程序早已稳定，免驱动安装

全自动停电/上电，ISP下载编程烧录器

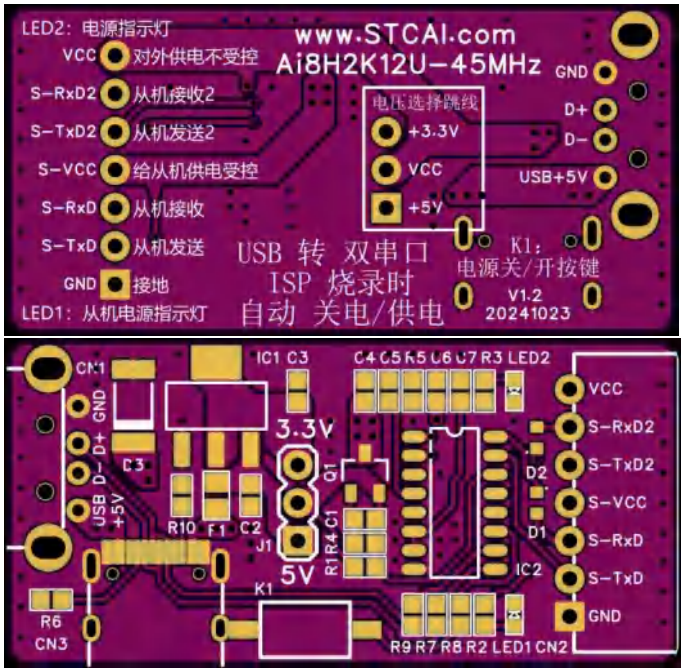
有手动电源开/关按键，可做电源供电模块

立创开源SCH/PCB，V1.2-20241023：

原理图：



制板量产的PCB链接见下面链接：



附录P　　串口中断收发－MODBUS 协议

C 语言代码

```
//测试工作频率为 11.0592MHz

#include “Ai8051U.H”                                    //头文件见下载软件

#define MAIN_Fosc        11059200L      //定义主时钟

/*************        功能说明        **************

请先别修改程序，直接下载"08-串口 1 中断收发-C 语言-MODBUS 协议"里的"UART1.hex"测试，主频选择 11.0592MHZ.
测试正常后再修改移植.

串口 1 按 MODBUS-RTU 协议通信. 本例为从机程序，主机一般是电脑端.

本例程只支持多寄存器读和多寄存器写，寄存器长度为 64 个，别的命令用户可以根据需要按 MODBUS-RTU 协议自行添加.

本例子数据使用大端模式(与 C51 一致), CRC16 使用小端模式(与 PC 一致).

默认参数:
串口 1 设置均为 1 位起始位, 8 位数据位, 1 位停止位，无校验.
串口 1(P3.0 P3.1): 9600bps.

定时器 0 用于超时计时. 串口每收到一个字节都会重置超时计数，当串口空闲超过 35bit 时间时(9600bps 对应 3.6ms)则接收完成.
用户修改波特率时注意要修改这个超时时间.

本例程只是一个应用例子，科普 MODBUS-RTU 协议并不在本例子职责范围，用户可以上网搜索相关协议文本参考.
本例定义了 64 个寄存器，访问地址为 0x1000~0x103f.
命令例子:
写入 4 个寄存器(8 个字节):
10 10 1000 0004 08 1234 5678 90AB CDEF 4930
返回:
10 10 10 00 00 04 4B C6
读出 4 个寄存器:
10 03 1000 0004 4388
返回:
10 03 08 12 34 56 78 90 AB CD EF 3D D5

命令错误返回信息(自定义):
0x90: 功能码错误. 收到了不支持的功能码.
0x91: 命令长度错误.
0x92: 写入或读出寄存器个数或字节数错误.
0x93: 寄存器地址错误.

注意: 收到广播地址 0x00 时要处理信息，但不返回应答.

*****************************************/

typedef     unsigned char    u8;
typedef     unsigned int     u16;
typedef     unsigned long    u32;
```

```
/*************       本地常量声明    **************/
#define    RX1_Length    128        /*  接收缓冲长度      */
#define    TX1_Length    128        /*  发送缓冲长度      */


/*************       本地变量声明    **************/
u8    xdataRX1_Buffer[RX1_Length];      //接收缓冲
u8    xdataTX1_Buffer[TX1_Length];      //发送缓冲

u8    RX1_cnt;              //接收字节计数.
u8    TX1_cnt;              //发送字节计数
u8    TX1_number;           //要发送的字节数
u8    RX1_TimeOut;          //接收超时计时器

bit   B_RX1_OK;             //  接收数据标志
bit   B_TX1_Busy;           //  发送忙标志


/************* 本地函数声明    **************/
void  UART1_config(u32 brt, u8 timer, u8 io);     // brt:通信波特率,      timer=2:  波特率使用定时器  2,  其它值:  使用
Timer1 做波特率. io=0:  串口 1 切换到 P3.0 P3.1,    =1:  切换到 P3.6 P3.7, =2:  切换到 P1.6 P1.7,     =3:    切  换  到    P4.3
      P4.4.
u8          Timer0_Config(u8 t,   u32   reload);    //t=0: reload 值是主时钟周期数, t=1: reload 值是时间(单位 us), 返回 0 正
确,    返回 1 装载值过大错误.
u16         MODBUS_CRC16(u8 *p,    u8 n);
u8          MODBUS_RTU(void);



#define    SL_ADDR       0x10 /*  本从机站号地址  */
#define    REG_ADDRESS       0x1000     /*  寄存器首地址          */
#define    REG_LENGTH 64           /*  寄存器长度      */
u16        xdata modbus_reg[REG_LENGTH];   /*  寄存器地址  */

//========================================================================
//  函数: void main(void)
//  描述:  主函数
//  参数: none.
//  返回: none.
//  版本: VER1.0
//  备注:
//========================================================================
void main(void)
{
      u8    i;
      u16   crc;
```

```
    EAXFR = 1;                                  //允许访问扩展的特殊寄存器，XFR
                                                //(32 位模式请使用这句，注释下一句)
//  P_SW2 |= 0x80;                              //允许访问扩展的特殊寄存器，XFR
                                                //(8 位模式请使用这句，注释上一句)
    WTST = 0;                                   //设置取程序代码等待时间，
                                                //赋值为 0 表示不等待，程序以最快速度运行
    CKCON = 0;                                  //设置访问片内的 xdata 速度，
                                                //赋值为 0 表示用最快速度访问，不增加额外的等待时间

    Timer0_Config(0, MAIN_Fosc / 10000);    //t=0: reload 值是主时钟周期数, (中断频率, 20000 次/秒)
    UART1_config(9600UL, 1, 0);   // brt:通信波特率,     timer=2: 波特率使用定时器 2, 其它值: 使用 Timer1 做波特率. io=0: 串口 1 切换到 P3.0 P3.1,  =1: 切换到 P3.6 P3.7, =2: 切换到 P1.6 P1.7,  =3: 切换到 P4.3    P4.4.

    EA = 1;

    while (1)
    {
        if(B_RX1_OK  && !B_TX1_Busy)   //收到数据,     进行 MODBUS-RTU 协议解析
        {
            if(MODBUS_CRC16(RX1_Buffer,    RX1_cnt) ==    0)   //首先判断 CRC16 是否正确, 不正确则忽略, 不处理也不返回信息
            {
                if((RX1_Buffer[0] == 0x00) || (RX1_Buffer[0] ==  SL_ADDR))    //然后判断站号地址是否正确,  或者是否广播地址(不返回信息)
                {
                    if(RX1_cnt > 2)  RX1_cnt  -= 2; //去掉 CRC16 校验字节
                    i =  MODBUS_RTU();    //MODBUS-RTU 协议解析
                    if(i !=    0)   //错误处理
                    {
                        TX1_Buffer[0] =SL_ADDR;     //站号地址
                        TX1_Buffer[1] =i;           //错误代码
                        crc  = MODBUS_CRC16(TX1_Buffer, 2);
                        TX1_Buffer[2] =(u8)(crc>>8);   //CRC 是小端模式
                        TX1_Buffer[3] =(u8)crc;
                        B_TX1_Busy = 1;             //标志发送忙
                        TX1_cnt    = 0;             //发送字节计数
                        TX1_number = 4;             //要发送的字节数
                        TI = 1;                     //启动发送
                    }
                }
            }
            RX1_cnt  = 0;
            B_RX1_OK = 0;
        }
    }
}
```

```
/*****************************    MODBUS_CRC (shift) ************** past      test 06-11-27 *********
      计算 CRC，调用方式MODBUS_CRC16(&CRC,8);     &CRC 为首地址，8 为字节数
      CRC-16 for MODBUS
      CRC16=X16+X15+X2+1
      TEST: ---> ABCDEFGHIJ  CRC16=0x0BEE       1627T
*/
//===========================================================================
// 函数: u16      MODBUS_CRC16(u8 *p,    u8 n)
// 描述:  计算 CRC16 函数.
// 参数: *p:  要计算的数据指针.
//              n:  要计算的字节数.
// 返回: CRC16 值.
// 版本: V1.0
//===========================================================================
u16   MODBUS_CRC16(u8 *p,    u8 n)
{
      u8    i;
      u16   crc16;

      crc16 =     0xffff;      //预置 16 位 CRC 寄存器为 0xffff（即全为 1）
      do
      {
            crc16 ^= (u16)*p;            //把 8 位数据与 16 位 CRC 寄存器的低位相异或，把结果放于 CRC 寄存器
            for(i=0; i<8; i++)           //8 位数据
            {
                  if(crc16 & 1)     crc16 =     (crc16 >> 1) ^ 0xA001;      //如果最低位为 0，把 CRC 寄存器的内容右移一位
(朝低位)，用 0 填补最高位，
                                                                              //再异或多项式 0xA001
                  else   crc16 >>= 1;                                         //如果最低位为 0，把 CRC 寄存器的内容右移一位
(朝低位)，用 0 填补最高位
            }
            p++;
      }while(--n != 0);
      return      (crc16);
}

/********************* modbus 协议 ************************/
/*****************************************************************************
  写多寄存器
数据:        地址        功能码       寄存地址  寄存器个数   写入字节数       写入数据     CRC16
偏移:         0           1        2 3          4 5          6                 7~             最后 2 字节
字节:       1 byte       1 byte        2 byte   2 byte           1byte       2*n   byte     2 byte
             addr         0x10          xxxx    xxxx             xx              xx....xx     xxxx

   返回
数据:        地址        功能码       寄存地址  寄存器个数    CRC16
```

```
偏移:        0          1       2 3          4 5          6 7
字节:     1 byte     1 byte      2 byte   2 byte          2 byte
          addr       0x10        xxxx     xxxx            xxxx


 读多寄存器
数据:站号(地址)      功能码     寄存地址  寄存器个数   CRC16
偏移:        0     1      2 3          4 5          6 7
字节:     1 byte     1 byte      2 byte   2 byte          2 byte
          addr       0x03        xxxx     xxxx            xxxx

  返回
数据:站号(地址)      功能码     读出字节数   读出数据  CRC16
偏移:        0     1      2                3~          最后 2 字节
字节:     1 byte     1 byte      1byte     2*n byte 2 byte
          addr       0x03        xx          xx....xx    xxxx

  返回错误代码
数据:站号(地址)      错误码     CRC16
偏移:        0     1    最后 2 字节
字节:     1 byte     1 byte      2  byte
          addr       0x03        xxxx
*****************************************************************************/
u8   MODBUS_RTU(void)
{
    u8   i,j,k;
    u16  reg_addr;  //寄存器地址
    u8   reg_len;   //写入寄存器个数
    u16  crc;

    if(RX1_Buffer[1] ==  0x10)//写多寄存器
    {
        if(RX1_cnt < 9)         return 0x91;            //命令长度错误
        if((RX1_Buffer[4] != 0)     || ((RX1_Buffer[5] *2) != RX1_Buffer[6]))   return 0x92;     //写入寄存器个数与字节数错误
        if((RX1_Buffer[5]==0) || (RX1_Buffer[5]     > REG_LENGTH))   return 0x92;     //写入寄存器个数错误

        reg_addr = ((u16)RX1_Buffer[2] << 8) + RX1_Buffer[3];      //寄存器地址
        reg_len     = RX1_Buffer[5];      //写入寄存器个数
        if((reg_addr+(u16)RX1_Buffer[5]) > (REG_ADDRESS+REG_LENGTH))      return 0x93;      //寄存器地址错误
        if(reg_addr< REG_ADDRESS)           return 0x93;      //寄存器地址错误
        if((reg_len*2+7) !=     RX1_cnt)   return 0x91;      //命令长度错误

        j =    reg_addr - REG_ADDRESS;      //寄存器数据下标
        for(k=7, i=0; i<reg_len; i++,j++)
        {
```

```
                modbus_reg[j] = ((u16)RX1_Buffer[k]  << 8) +     RX1_Buffer[k+1];     //写入数据,       大端模式
                k += 2;
            }

            if(RX1_Buffer[0] !=   0)     //非广播地址则应答
            {
                for(i=0; i<6; i++)       TX1_Buffer[i] = RX1_Buffer[i];  //要返回的应答
                crc    = MODBUS_CRC16(TX1_Buffer, 6);
                TX1_Buffer[6] =(u8)(crc>>8);    //CRC 是小端模式
                TX1_Buffer[7] =(u8)crc;
                B_TX1_Busy = 1;             //标志发送忙
                TX1_cnt        = 0;         //发送字节计数
                TX1_number = 8;             //要发送的字节数
                TI = 1;                     //启动发送
            }
        }
        else if(RX1_Buffer[1] == 0x03)   //读多寄存器
        {
            if(RX1_Buffer[0] !=   0)     //非广播地址则应答
            {
                if(RX1_cnt != 6)        return 0x91;            //命令长度错误
                if(RX1_Buffer[4] !=   0)     return 0x92;       //读出寄存器个数错误
                if((RX1_Buffer[5]==0) || (RX1_Buffer[5]     > REG_LENGTH))    return 0x92;     //读出寄存器个数错误

                reg_addr = ((u16)RX1_Buffer[2] << 8) + RX1_Buffer[3];       //寄存器地址
                reg_len     = RX1_Buffer[5];      //读出寄存器个数
                if((reg_addr+(u16)RX1_Buffer[5]) > (REG_ADDRESS+REG_LENGTH))      return 0x93;      //寄存器地址
错误
                if(reg_addr< REG_ADDRESS)          return 0x93;     //寄存器地址错误

                j =    reg_addr - REG_ADDRESS;      //寄存器数据下标
                TX1_Buffer[0] =SL_ADDR;      //站号地址
                TX1_Buffer[1] =0x03;      //读功能码
                TX1_Buffer[2] =reg_len*2; //返回字节数

                for(k=3, i=0; i<reg_len; i++,j++)
                {
                    TX1_Buffer[k++]      = (u8)(modbus_reg[j] >>     8);    //数据为大端模式
                    TX1_Buffer[k++]      = (u8)modbus_reg[j];
                }
                crc    = MODBUS_CRC16(TX1_Buffer, k);
                TX1_Buffer[k++]      = (u8)(crc>>8);  //CRC 是小端模式
                TX1_Buffer[k++]      = (u8)crc;
                B_TX1_Busy = 1;             //标志发送忙
                TX1_cnt        = 0;         //发送字节计数
                TX1_number = k;             //要发送的字节数
```

```
            TI = 1;                     //启动发送
        }
    }
    else   return 0x90;     //功能码错误

    return 0;   //解析正确
}


//==========================================================================
// 函数:u8 Timer0_Config(u8 t,   u32   reload)
// 描述: timer0 初始化函数.
// 参数:       t:  重装值类型, 0 表示重装的是系统时钟数,  其余值表示重装的是时间(us).
//             reload:  重装值.
// 返回: 0: 初始化正确,      1:  重装值过大,  初始化错误.
// 版本: V1.0
//==========================================================================
u8    Timer0_Config(u8 t,   u32   reload)     //t=0: reload 值是主时钟周期数, t=1: reload 值是时间(单位 us)
{
    TR0  = 0;  //停止计数

    if(t !=       0)     reload = (u32)(((float)MAIN_Fosc * (float)reload)/1000000UL);     //重装的是时间(us),  计算所需要的系统时钟数.
    if(reload >= (65536UL *     12))   return 1;   //值过大,  返回错误
    if(reload < 65536UL)  AUXR |=   0x80;       //1T mode
    else
    {
        AUXR &= ~0x80;      //12T mode
        reload = reload   / 12;
    }
    reload = 65536UL - reload;
    TH0  = (u8)(reload >> 8);
    TL0  = (u8)(reload);

    ET0  = 1;  //允许中断
    TMOD &= 0xf0;
    TMOD |=  0;     //工作模式,       0: 16 位自动重装, 1:  16 位定时/计数, 2: 8 位自动重装, 3: 16 位自动重装,     不可屏蔽中断
    TR0  = 1;                   //开始运行
    return 0;
}

//==========================================================================
// 函数: void timer0_ISR (void)    interrupt TIMER0_VECTOR
// 描述:    timer0 中断函数.
// 参数: none.
```

```
// 返回: none.
// 版本: V1.0
//========================================================================
void timer0_ISR (void) interrupt 1
{
    if(RX1_TimeOut != 0)
    {
        if(--RX1_TimeOut == 0)    //超时
        {
            if(RX1_cnt != 0)//接收有数据
            {
                B_RX1_OK = 1;//标志已收到数据块
            }
        }
    }
}


//========================================================================
// 函数: SetTimer2Baudraye(u16 dat)
// 描述:  设置 Timer2 做波特率发生器。
// 参数: dat: Timer2 的重装值.
// 返回: none.
// 版本: VER1.0
// 备注:
//========================================================================
void  SetTimer2Baudraye(u16 dat)      // 选择波特率, 2: 使用 Timer2 做波特率, 其它值: 使用 Timer1 做波特率.
{
    AUXR &= ~(1<<4);   //Timer     stop
    AUXR &= ~(1<<3);   //Timer2 set As   Timer
    AUXR |=    (1<<2);   //Timer2 set as   1T mode
    T2H  = (u8)(dat >> 8);
    T2L  = (u8)dat;
    IE2    &= ~(1<<2);   //禁止中断
    AUXR |=    (1<<4);   //Timer     run   enable
}


//========================================================================
// 函数: void     UART1_config(u32 brt, u8 timer, u8 io)
// 描述: UART1 初始化函数。
// 参数:     brt: 通信波特率.
//              timer:     波特率使用的定时器,timer=2: 波特率使用定时器 2,  其它值:    使用 Timer1 做波特率.
//                  io:    串口 1 切换到的 IO,   io=0:       串口 1 切换到 P3.0    P3.1,   =1: 切换到 P3.6 P3.7,    =2:   切 换 到
P1.6 P1.7,   =3: 切换到 P4.3 P4.4.
// 返回: none.
```

```
// 版本: VER1.0
// 备注:
//=========================================================================
void  UART1_config(u32 brt, u8 timer, u8 io)      // brt:通信波特率,      timer=2: 波特率使用定时器 2, 其它值: 使用 Timer1 做波特率. io=0: 串口 1 切换到 P3.0 P3.1,   =1: 切换到 P3.6 P3.7, =2: 切换到 P1.6 P1.7,    =3:   切 换 到   P4.3 P4.4.
{
    brt    = 65536UL -     (MAIN_Fosc / 4)      / brt;
    if(timer == 2)    //波特率使用定时器 2
    {
        AUXR |=   0x01;       //S1 BRT Use Timer2;
        SetTimer2Baudraye((u16)brt);
    }

    else        //波特率使用定时器 1
    {
        TR1  = 0;
        AUXR &= ~0x01;            //S1 BRT Use Timer1;
        AUXR |=    (1<<6);   //Timer1 set as   1T mode
        TMOD &=~(1<<6);   //Timer1 set As   Timer
        TMOD &=~0x30;             //Timer1_16bitAutoReload;
        TH1  = (u8)(brt >> 8);
        TL1  = (u8)brt;
        ET1  = 0;                 // 禁止 Timer1 中断
        TR1   = 1;                // 运行 Timer1
    }
    P_SW1 &= ~0xc0;    //默认切换到 P3.0 P3.1
    if(io == 1)
    {
        P_SW1   |=   0x40;   //切换到 P3.6 P3.7
        P3M1 &=  ~0xc0;
        P3M0 &=  ~0xc0;
    }
    else if(io == 2)
    {
        P_SW1 |= 0x80; //切换到 P1.6 P1.7
        P1M1 &=  ~0xc0;
        P1M0 &=  ~0xc0;
    }
    else if(io == 3)
    {
        P_SW1 |= 0xc0; //切换到 P4.3 P4.4
        P4M1 &=  ~0x18;
        P4M0 &=  ~0x18;
    }
    else
```

```
    {
        P3M1 &= ~0x03;
        P3M0 &= ~0x03;
    }

    SCON = (SCON & 0x3f) | (1<<6);      // 8 位数据,      1 位起始位, 1 位停止位, 无校验
//  PS   = 1;  //高优先级中断
    ES   = 1;  //允许中断
    REN = 1;  //允许接收
}


//========================================================================
// 函数: void UART1_ISR   (void) interrupt UART1_VECTOR
// 描述: 串口 1 中断函数
// 参数: none.
// 返回: none.
// 版本: VER1.0
// 备注:
//========================================================================
void UART1_ISR (void) interrupt 4
{
    if(RI)
    {
        RI = 0;
        if(!B_RX1_OK) //接收缓冲空闲
        {
            if(RX1_cnt >= RX1_Length)      RX1_cnt = 0;
            RX1_Buffer[RX1_cnt++] = SBUF;
            RX1_TimeOut   = 36;//接收超时计时器, 35 个位时间
        }
    }

    if(TI)
    {
        TI = 0;
        if(TX1_number != 0)  //有数据要发
        {
            SBUF = TX1_Buffer[TX1_cnt++];
            TX1_number--;
        }
        else  B_TX1_Busy = 0;
    }
}
```

附录Q　如何通过读取寄存器获取芯片版本

Ai8051U 系列的芯片硬件改版一般会有比较大的功能修改，通常可以通过看芯片上面的丝印最下面一行的最后一个字母可以识别芯片的版本（注意：芯片版本和芯片内部的 ISP 固件版本不同）。有时为了程序代码能够兼容不同版本的芯片，希望能通过软件方式获取芯片的版本。下面的代码演示了如果通过寄存器（DID）来获取芯片版本的方法

```
#include <stdio.h>
#include <8051U.h>

#define    FOSC    11059200UL
#define    BAUD    (65536 – (FOSC/115200+2)/4)

sfr    DID  =    0xa7;

int    main()
{
       char id;

       EAXFR = 1;                          //允许访问扩展的特殊寄存器，XFR
                                           //(32 位模式请使用这句，注释下一句)
//     P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR
                                           //(8 位模式请使用这句，注释上一句)
       WTST = 0;                           //设置取程序代码等待时间，
                                           //赋值为 0 表示不等待，程序以最快速度运行
       CKCON = 0;                          //设置访问片内的 xdata 速度，
                                           //赋值为 0 表示用最快速度访问，不增加额外的等待时间

       IAP_ADDRL =  0x02;                  //设置地址
       id = DID;                           //读取芯片版本

       SCON = 0x52;
       AUXR = 0x40;
       TMOD = 0x00;
       TH1 = BAUD >> 8;
       TL1 = BAUD;
       TR1 = 1;

       printf("芯片版本：%c 版本\n", 'A' + id);

       while (1);
}
```

附录R　单总线（1-Wire）

单片机内部集成了一个单总线控制器。单总线控制器支持对 WS2812、DS18B20、DHT11 等器件的控制，并支持通用红外发送接收、串口发送接收等模式。单总线支持直接寄存器读写数据，也支持 DMA 方式读写数据。

控制器内部有集成 16 字节的发送 FIFO 和 16 字节的接收 FIFO，当使能寄存器直接读写数据时，可软件使能或关闭 FIFO 功能。

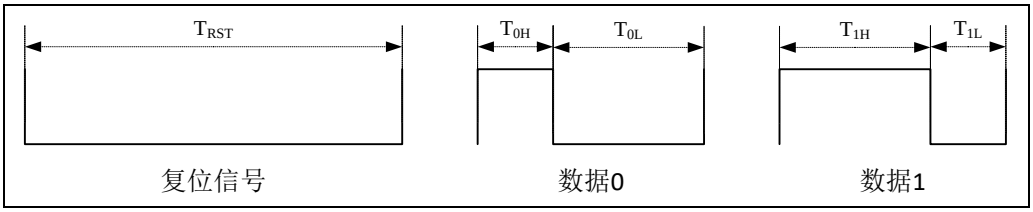
单总线目前可支持的工作模式：WS2812 模式、DS18B20 模式、DHT11 模式、红外模式、UARTX 模式和 UART 模式。

R.1　单总线工作模式

R.1.1　WS2812 模式

WS2812 模式可用于驱动 WS2812 灯带，以及数据格式与 WS2812 相类似的 TM3130、TM1809、DSHOT 等协议，均可以使用此模式进行数据发送。WS2812 模式为主机只写模式，只能主动向外发送数据，从机模式和读取操作对此模式无效。

WS2812 有 3 种信号：复位信号、数据 0 信号、数据 1 信号。低电平维持一定的时间宽度，即为复位信号，用不同的高低电平宽度表示数据 0 和数据 1。如下图：



特别的，如果对数据两个数据之间的时间间隔有要求时，建议开启 FIFO 模式发送数据或者使用 DMA 发送数据，这样可保证两个数据之间的时间间隔等距。

（使用 FIFO 模式发送 FIFO 缓冲区中的数据，两个数据之间的间隔为 1 个系统时钟；使用 DMA 发送 XDATA 缓冲区中的数据，在没有数据总线仲裁的情况下，两个数据之间的间隔为 7 个系统时钟。）

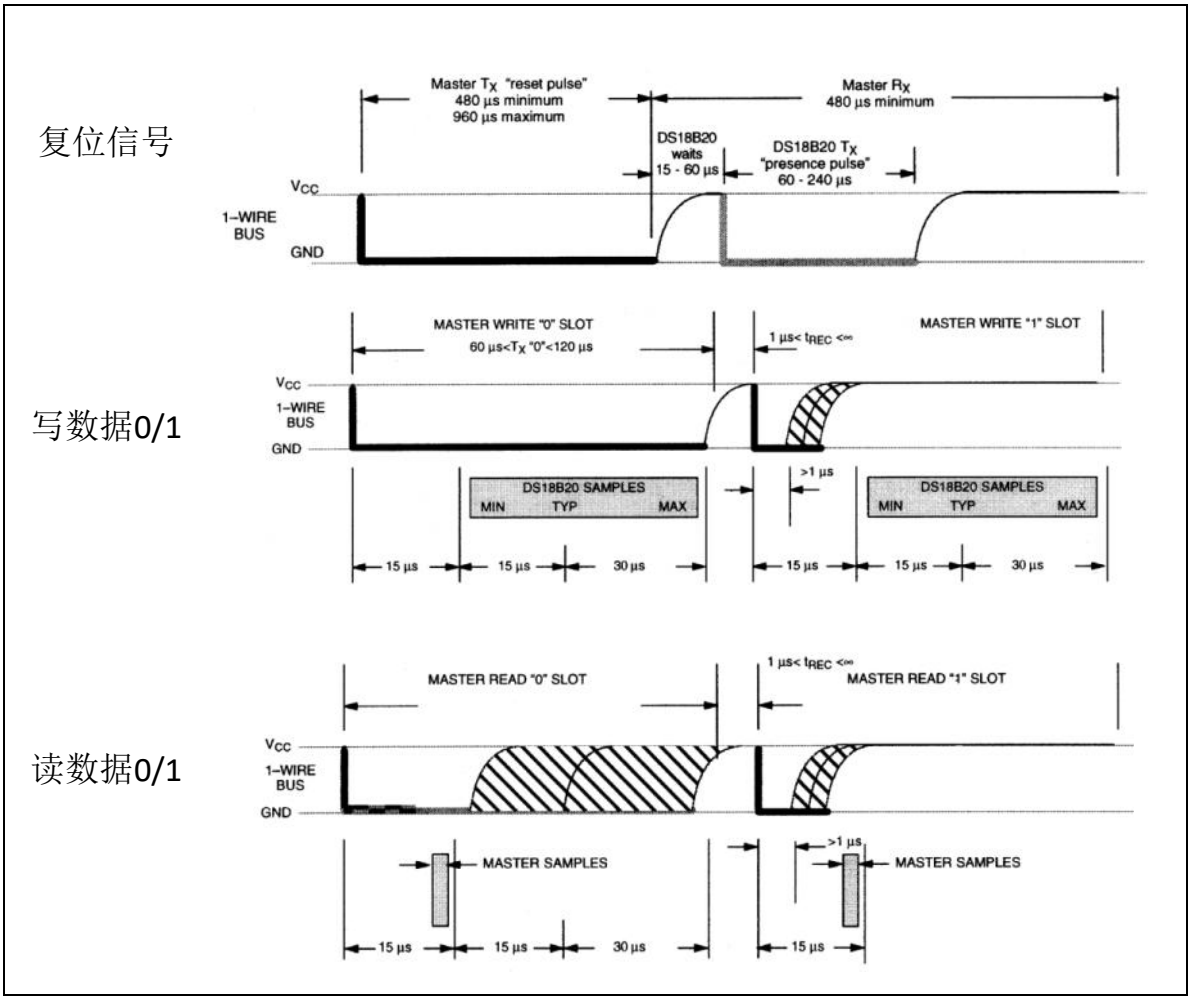
R.1.2　DS18B20 模式

DS18B20 模式专用于对 DS18B20 温度传感器进行数据读写。DS18B20 有 3 中操作：总线复位、写数据和读数据。

总线复位信号是单总线控制器发送一定时长的低电平信号，触发 DS18B20 传感器的初始化动作，在控制器拉高数据总线后，DS18B20 会等待一定的时间后发出设备存在脉冲（拉低总线），表示设备存在且工作正常，等待 DS18B20 自动释放总线（拉高总线）时，总线复位信号完成。

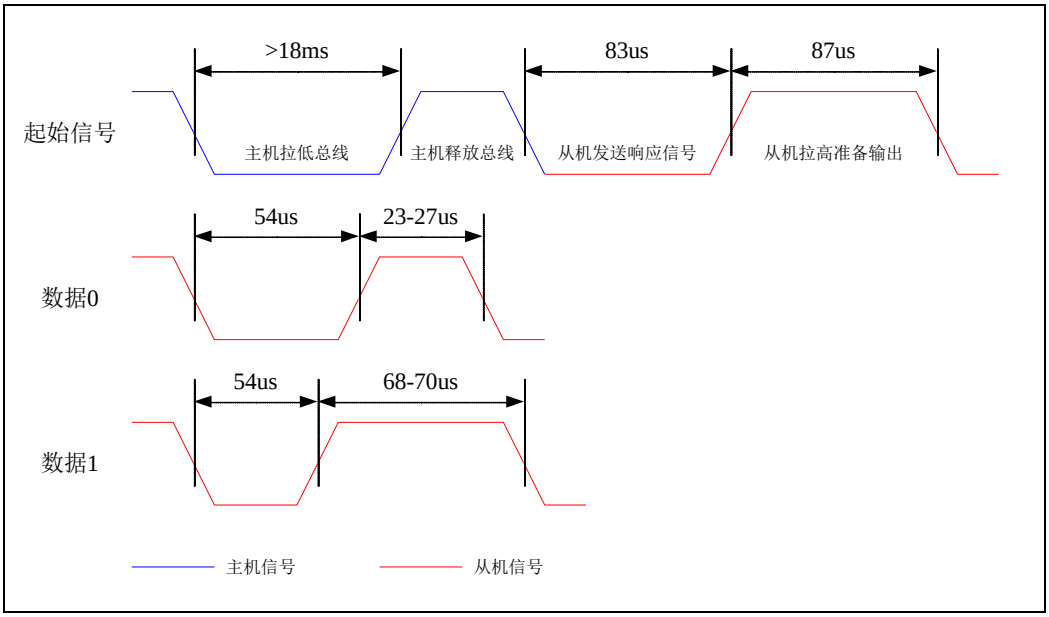
写数据是控制器发送一串命令序列，触发 DS18B20 完成指定的工作，比如发送 CCH、44H 命令可触发 DS18B20 完成一次温度转换。

读数据是从 DS18B20 传感器中读取温度数据以及其他的附加信息。DS18B20 的每个 bit 的数据读取均需要控制器发送起始信号，DS18B20 随后被动的将相应的数据位送到总线上。



R.1.3　DHT11 模式

DHT11 模式专用于对 DHT11 温度湿度传感器进行数据读取。DHT11 工作模式相对较简单，接收到主机发送的复位信号（开始信号）后，DHT11 会发送一个位的存在脉冲（相应信号），接下来会连续发送 5 字节的温湿度数据以及校验字节。



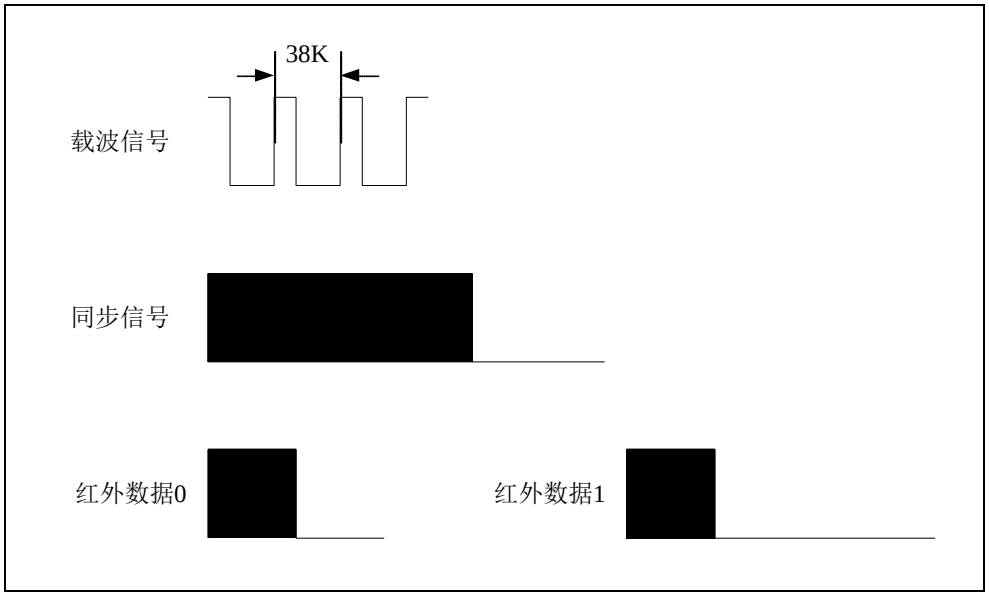
R.1.4　红外模式

红外模式可用于兼容 NEC 红外格式数据的发送和接收。

主机模式为红外发送，从机模式为红外接收。

主机模式可配置发送红外数据时是否发送载波信号。如需要发送红外载波，载波的频率和占空比也均可软件配置。主机支持两种发送两种信号：同步信号和红外 8 位数据。

从机模式接收的红外数据必须是经过红外接收头对载波信号进行滤波后的信号，接收信号格式：有载波信号时为低电平，无载波信号时为高电平。

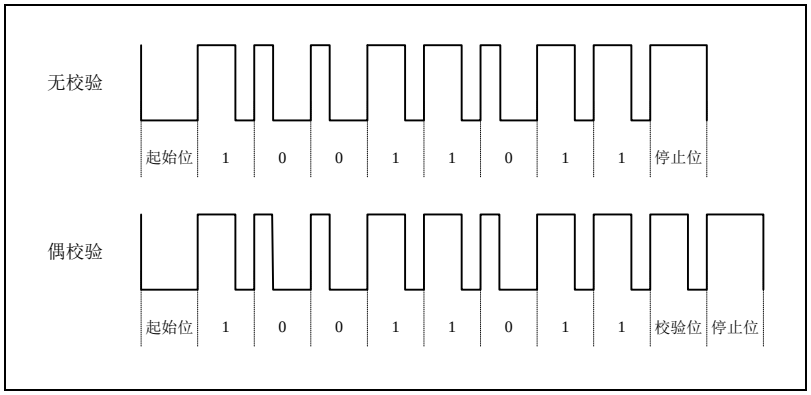


R.1.5　UARTX 模式

UARTX 模式与传统的异步串口的数据格式相同，但 UARTX 没有固定的波特率，数据位采用不同的高低电平表示，从而可以避免传统的异步串口通讯时由于波特率误差而导致的数据出错。

主机模式为数据发送模式。硬件自动产生校验位，校验位支持 5 种格式：无校验、奇校验、偶校验、1 校验、0 校验。数据位固定为 8bit。

从机模式为数据接收模式。自动对校验位进行校验，如接收的校验位错误，则会产生相应的标志位。

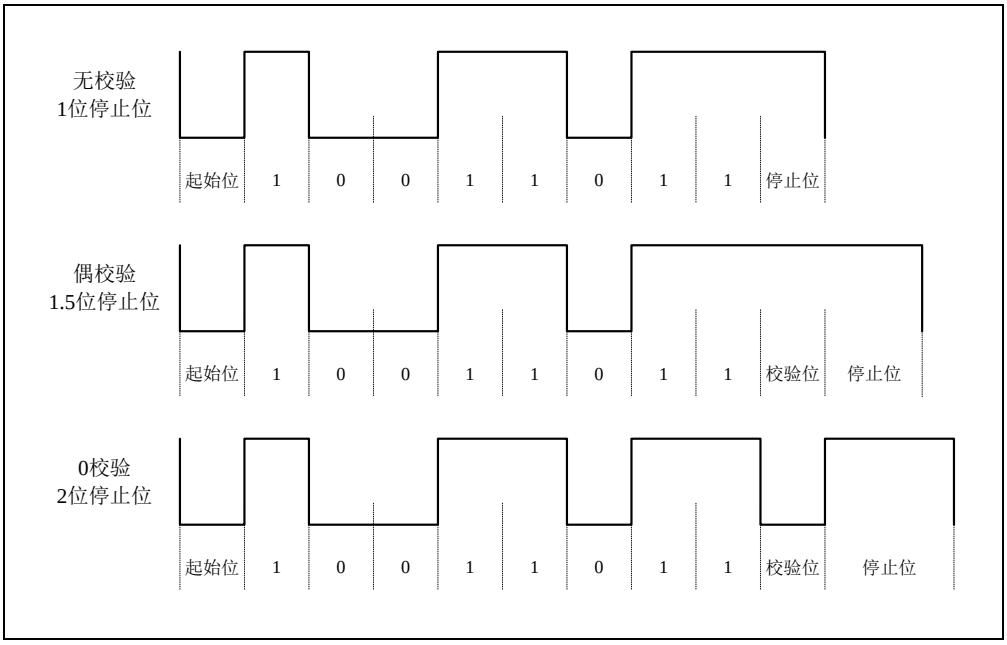


R.1.6　UART 模式

UART 模式与传统的异步串口完全兼容。

主机模式为串口数据发送模式，发送波特率最快可为系统时钟。硬件自动产生校验位，校验位支持 5 种格式：无校验、奇校验、偶校验、1 校验、0 校验。停止位支持 3 中格式：1 位、1.5 位、2 位。数据位固定为 8bit。

从机模式为串口数据接收模式，接收波特率最快为系统时钟/2，自动对校验位和停止位进行校验，如接收的校验位错误或者停止位为低电平，则会产生相应的标志位。



R.2 单总线相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
OWCR	单总线控制寄存器	F730H	ENOW	DORD	INVO	RSTBUS	READ	SLAVE	OWPS[1:0]		0000,0000
OWIR	单总线中断寄存器	F731H	RFFIE	TFEIE	TOIE	RXIE	TXIE	RSTIE	OWIP[1:0]		0000,0000
OWSR	单总线状态寄存器	F732H	RFFIF	TFEIF	TOIF	RXIF	TXIF	RSTIF	PRES	BUSY	0000,0000
OWDR	单总线数据寄存器	F733H	DATA[7:0]								0000,0000
OWPSCR	单总线时钟预分频寄存器	F734H	SCALE[7:0]								0000,0000
OWMR	单总线模式寄存器	F735H	保留				-	MODE[2:0]			xxxx,x000
OWFCSR	单总线 FIFO 控制状态寄存器	F736H	ENFIFO	TFULL **(R)** TFLUSH **(W)**	REMPTY **(R)** RFLUSH **(W)**	RCOUNT[4:0] **(R)** THRESHOLD[4:0] **(W)**					0010,0000
OWCFG1	单总线配置寄存器 1	F738H	CFG1[7:0]								0000,0000
OWCFG2	单总线配置寄存器 2	F739H	CFG2[7:0]								0000,0000
OWCFG3	单总线配置寄存器 3	F73AH	CFG3[7:0]								0000,0000
OWCFG4	单总线配置寄存器 4	F73BH	CFG4[7:0]								0000,0000
OWCFG5	单总线配置寄存器 5	F73CH	CFG5[7:0]								0000,0000
OWCFG6	单总线配置寄存器 6	F73DH	CFG6[7:0]								0000,0000
OWCFG7	单总线配置寄存器 7	F73EH	CFG7[7:0]								0000,0000
OWCFG8	单总线配置寄存器 8	F73FH	CFG8[7:0]								0000,0000

R.3 寄存器说明

R.3.1 单总线控制寄存器（OWCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
OWCR	F730H	ENOW	DORD	INVO	RSTBUS	READ	SLAVE	OWPS[1:0]	

ENOW：单总线功能使能控制位。

0：禁止单总线功能。

1：允许单总线功能。

DORD：单总线数据发送/接收顺序。（设备为忙状态时，此位不能改写）

0：先发送/接收高位数据（MSB）。

1：先发送/接收低位数据（LSB）。

INVO：输出电平反转控制。（设备为忙状态时，此位不能改写）

0：数据直接输出。

1：数据反转后再输出。

RSTBUS：复位单总线。（只写）

0：写 0 无效。

1：触发单总线发送复位信号。

不同的工作模式，复位信号会不一样，对于没有复位信号的模式，写此位无效。

READ：触发读取单总线数据。（只写）

0：写 0 无效。

1：触发单总线读取数据。

对于 DS18B20 和 DHT11 这两种模式，写此位可触发数据读取操作。当使能 FIFO 且设置了 FIFO 的接收阈值，则只需触发一次读取，单总线会自动连续读取，直到读取的数据达到设定的阈值；

WS2812 设备为只写设备，写此位无效；

红外模式、UARTX 和 UART 模式的接收为硬件自动接收数据，写此位也无效。

SLAVE：设置单总线工作模式。（设备为忙状态时，此位不能改写）

0：主机模式。

1：从机模式。

对于 WS2812、DS18B20 和 DHT11 这 3 种模式，固定为主机模式；

红外模式设置主机模式时，可实现对外发送红外数据；设置从机模式时可实现接收外部的红外数据。

UARTX 和 UART 模式，主机模式为对外发送串口数据；从机模式为接收外部的数据。

OWPS[1:0]：单总线数据口选择。

OWPS[1:0]	单总线数据口
00	P3.5
01	P0.5
10	P2.7
11	P4.0

R.3.2 单总线中断寄存器（OWIR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
OWIR	F731H	RFFIE	TFEIE	TOIE	RXIE	TXIE	RSTIE	OWIP[1:0]	

RFFIE：接收 FIFO 缓冲区满（接收数据的字节数达到接收阈值）中断控制位。

0：关闭 RFF 中断功能。

1：使能 RFF 中断功能。

TFEIE：发送 FIFO 缓冲区空中断控制位。

0：关闭 TFE 中断功能。

1：使能 TFE 中断功能。

TOIE：超时中断控制位。

0：关闭 TO 中断功能。

1：使能 TO 中断功能。

RXIE：字节接收完成中断控制位。

0：关闭 RX 中断功能。

1：使能 RX 中断功能。

TXIE：字节发送完成中断控制位。

0：关闭 TX 中断功能。

1：使能 TX 中断功能。

RSTIE：主机模式时，总线复位信号发送完成中断控制位。

0：关闭 RST 中断功能。

1：使能 RST 中断功能。

OWIP[1:0]：单总线中断优先级控制。

OWIP[1:0]	单总线数据口
00	最低优先级
01	较低优先级
10	较高优先级
11	最高优先级

R.3.3 单总线状态寄存器（OWSR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
OWSR	F732H	RFFIF	TFEIF	TOIF	RXIF	TXIF	RSTIF	PRES	BUSY

RFFIF：接收 FIFO 缓冲区满（接收数量达到接收阈值）中断标志。需软件写 0 清除标志。

TFEIF：发送 FIFO 缓冲区空中断标志。需软件写 0 清除标志。

TOIF：超时中断标志。需软件写 0 清除标志。

RXIF：字节接收完成中断标志。需软件写 0 清除标志。

TXIF：字节发送完成中断标志。需软件写 0 清除标志。

RSTIF：总线复位完成中断标志。需软件写 0 清除标志。

PRES：从设备存在标志。（只读）

DS18B20 模式和 DHT11 模式时，当发送完成复位信号/起始信号后，如果接收到设置存在信号/相应信号，此标志位为 1，否则为 0。

BUSY：单总线忙标志。（只读）

单总线正在进行数据发送或数据接收时为忙状态。

R.3.4 单总线数据寄存器（OWDR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
OWDR	F733H	DATA[7:0]							

DATA[7:0]：发送/接收的数据寄存器。

当使能 FIFO 后，写操作会将寄存器的数据 PUSH 到发送 FIFO 中，读操作则会从接收 FIFO 中 POP 数据到寄存器。

注意：PUSH 数据前需要判断发送 FIFO 的满标志，非满状态才可执行 PUSH 操作，满状态时执行 PUSH，数据会自动丢失；POP 数据前也需要判断接收 FIFO 的空标志，只有为非空时 POP 的数据才有效，否则 POP 的数据为未知数据。

R.3.5 单总线时钟预分频寄存器（OWPSCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
OWPSCR	F734H	SCALE[7:0]							

SCALE[7:0]：时钟预分频寄存器。系统时钟经过 SCALE 分配后的时钟为单总线控制器时钟。

单总线控制器工作频率 = 系统时钟 SYSclk / (OWPSCR[7:0] + 1)。

（设备为忙状态时，此位不能改写）

R.3.6 单总线模式寄存器（OWMR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
OWMR	F735H	保留				-	MODE[2:0]		

MODE[2:0]：单总线工作模式。（设备为忙状态时，此位不能改写）

MODE[2:0]	工作模式
000	WS2812 模式
001	DS18B20 模式
010	DHT11 模式
011	IR 红外模式
100	UARTX 模式
101	UART 模式
110	保留
111	保留

R.3.7 单总线 FIFO 控制/状态寄存器（OWFCSR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
OWTCSR	F736H	ENFIFO	TFULL (R) TFLUSH (W)	REMPTY (R) RFLUSH (W)	RCOUNT[4:0] (R) THRESHOLD[4:0] (W)				

ENFIFO：内部 FIFO 控制位。

0：关闭 FIFO 功能。

1：使能 FIFO 功能。内部发送 FIFO 和接收 FIFO 是两个独立的缓冲区，大小均为 16 字节。

TFULL：发送 FIFO 缓冲区满标志。（只读）

TFLUSH：写“1”清空发送 FIFO，写“0”无效。（只写）

REMPTY：接收 FIFO 缓冲区空标志。（只读）

RFLUSH：写“1”清空接收 FIFO，写“0”无效。（只写）

COUNT[4:0]：接收 FIFO 中的数据个数。（只读）

THRESHOLD[4:0]：设置接收数据个数的阈值。（只写）

R.3.8 单总线配置寄存器（OWCFGn）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
OWCFG1	F738H	CFG1[7:0]							
OWCFG2	F739H	CFG2[7:0]							
OWCFG3	F73AH	CFG3[7:0]							
OWCFG4	F73BH	CFG4[7:0]							
OWCFG5	F73CH	CFG5[7:0]							
OWCFG6	F73DH	CFG6[7:0]							
OWCFG7	F73EH	CFG7[7:0]							
OWCFG8	F73FH	CFG8[7:0]							

CFGn[7:0]：配置不同工作模式的参数。配置此参数前，必须先设置工作模式。

（设备为忙状态时，这些寄存器不能改写）

WS2812 工作模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
OWCFG1	F738H	TRST[15:8]							
OWCFG2	F739H	TRST[7:0]							
OWCFG3	F73AH	DAT0H[7:0]							
OWCFG4	F73BH	DAT0L[7:0]							
OWCFG5	F73CH	DAT1H[7:0]							
OWCFG6	F73DH	DAT1L[7:0]							
OWCFG7	F73EH	-							
OWCFG8	F73FH	-							

TRST[15:0]：复位信号宽度（基于单数据总线时钟，以下相同）。

DAT0H[7:0]：数据 0 的高电平信号宽度。

DAT0L[7:0]：数据 0 的低电平信号宽度。

DAT1H[7:0]：数据 1 的高电平信号宽度。

DAT1L[7:0]：数据 1 的低电平信号宽度。

DS18B20 工作模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
OWCFG1	F738H	TRST[15:8]							
OWCFG2	F739H	TRST[7:0]							
OWCFG3	F73AH	TSLOT[7:0]							
OWCFG4	F73BH	TREC[7:0]							
OWCFG5	F73CH	TSTART[7:0]							
OWCFG6	F73DH	TRDV[7:0]							
OWCFG7	F73EH	TPDHIGH[7:0]							
OWCFG8	F73FH	TPDLOW[7:0]							

TRST[15:0]：复位信号的高/低电平宽度（基于单数据总线时钟，以下相同）。
TSLOT[7:0]：时间隙宽度（读写 1bit 是时间）。
TREC[7:0]：恢复时间宽度。
TSTART[7:0]：读写起始信号宽度。
TRDV[7:0]：读数据有效宽度。
TPDHIGH[7:0]：存在检测高电平宽度。
TPDLOW[7:0]：存在检测低电平宽度。

DHT11 工作模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
OWCFG1	F738H	TBE[15:8]							
OWCFG2	F739H	TBE[7:0]							
OWCFG3	F73AH	TGO[7:0]							
OWCFG4	F73BH	THRE[7:0]							
OWCFG5	F73CH	TTMO[7:0]							
OWCFG6	F73DH	-							
OWCFG7	F73EH	-							
OWCFG8	F73FH	-							

TBE[15:0]：主机起始信号拉低时间宽度。（基于单数据总线时钟，以下相同）
TGO[7:0]：主机释放总线时间宽度。
THRE[7:0]：读取数据阈值宽度（建议设置为信号 0 和信号 1 高电平的中间值）。
TTMO[7:0]：读取数据超时宽度。

NEC 红外工作模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
OWCFG1	F738H	TCH[3:0]				TCL[3:0]			
OWCFG2	F739H	TBASE[7:0]							
OWCFG3	F73AH	TSYNCH[7:0]							
OWCFG4	F73BH	TSYNCL[7:0]							
OWCFG5	F73CH	DATXH[3:0]				DAT0L[3:0]			
OWCFG6	F73DH	CCTL				DAT1L[3:0]			
OWCFG7	F73EH	RTHRE[7:0]							
OWCFG8	F73FH	RTMO[7:0]							

TCH[3:0]：红外载波高电平宽度。（基于单数据总线时钟）
TCL[3:0]：红外载波低电平宽度。

TBASE[7:0]：数据基本宽度。（基于单数据总线时钟）
TSYNCH[7:0]：红外发送同步头的载波基本宽度个数。（基于数据基本宽度）
TSYNCL[7:0]：红外发送同步头的空闲基本宽度个数。（基于数据基本宽度）
DATXH[3:0]：发送数据 0/1 的载波基本宽度个数。（基于数据基本宽度）
DAT0L[3:0]：发送数据 0 的空闲基本宽度个数。（基于数据基本宽度）
DAT1L[3:0]：发送数据 1 的空闲基本宽度个数。（基于数据基本宽度）
RTHRE[7:0]：红外接收数据阈值宽度。（基于数据基本宽度）
RTMO[7:0]：红外接收数据超时宽度。（基于数据基本宽度）
CCTL：红外载波发送控制
0：输出载波。
1：关闭载波。直接输出高电平。

UARTX 工作模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
OWCFG1	F738H	-				PBEF	PBE	PBT[1:0]	
OWCFG2	F739H	DAT0H[7:0]							
OWCFG3	F73AH	DAT0L[7:0]							
OWCFG4	F73BH	DAT1H[7:0]							
OWCFG5	F73CH	DAT1L[7:0]							
OWCFG6	F73DH	TSTART[7:0]							
OWCFG7	F73EH	TSTOP[7:0]							
OWCFG8	F73FH	TTMO[7:0]							

PBEF：接收校验位错误标志。
PBE：校验位控制。
0：无校验位。
1：使能校验位。
PBT[1:0]：校验位格式。PBE 为 1 时设置有效。
00：奇校验。
01：偶校验。
10：“1”校验。
11：“0”校验。
DAT0H[7:0]：数据 0 的高电平信号宽度。（基于单数据总线时钟）
DAT0L[7:0]：数据 0 的低电平信号宽度。（基于单数据总线时钟）
DAT1H[7:0]：数据 1 的高电平信号宽度。（基于单数据总线时钟）
DAT1L[7:0]：数据 1 的低电平信号宽度。（基于单数据总线时钟）
TSTART[7:0]：起始信号宽度。（基于单数据总线时钟）
TSTOP[7:0]：停止信号宽度。（基于单数据总线时钟）
RTMO[7:0]：接收数据超时宽度。（基于单数据总线时钟）

UART 工作模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
OWCFG1	F738H	SBEF	-	SBT[1:0]		PBEF	PBE	PBT[1:0]	
OWCFG2	F739H	TDAT[7:0]							
OWCFG3	F73AH	-							
OWCFG4	F73BH	-							
OWCFG5	F73CH	-							
OWCFG6	F73DH	-							
OWCFG7	F73EH	-							
OWCFG8	F73FH	-							

SBEF：接收停止位错误标志。

SBT[1:0]：停止位格式。

00：1 位停止位。

01：1.5 位停止位。

10：2 位停止位。

11：-。

PBEF：接收校验位错误标志。

PBE：校验位控制。

0：无校验位。

1：使能校验位。

PBT[1:0]：校验位格式。PBE 为 1 时设置有效。

00：奇校验。

01：偶校验。

10：“1”校验。

11：“0”校验。

TDAT[7:0]：起始位/数据位/校验位/停止位宽度。（基于单数据总线时钟）

R.4 单总线 DMA 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
DMA_OWT_CFG	OW-DMA 发送配置寄存器	F640H	OWTIE	-	-	-	OWTIP[1:0]		OWTBAP[1:0]		0xxx,0000
DMA_OWT_CR	OW-DMA 发送控制寄存器	F641H	ENOWT	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_OWT_STA	OW-DMA 发送状态寄存器	F642H	-	-	-	-	-	TXOVW	-	OWTIF	xxxx,x0x0
DMA_OWT_AMTH	OW-DMA 发送传输总字节数高位	F644H									0000,0000
DMA_OWT_AMT	OW-DMA 发送传输总字节数低位	F645H									0000,0000
DMA_OWT_DONEH	OW-DMA 发送传输完成字节数高位	F646H									0000,0000
DMA_OWT_DONE	OW-DMA 发送传输完成字节数低位	F647H									0000,0000
DMA_OWT_TXAH	OW-DMA 发送地址高位	FA48H									0000,0000
DMA_OWT_TXAL	OW-DMA 发送地址低位	FA49H									0000,0000

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
DMA_OWR_CFG	OW-DMA 接收配置寄存器	F650H	OWRIE	-	-	-	OWRIP[1:0]		OWRBAP[1:0]		0xxx,0000
DMA_OWR_CR	OW-DMA 接收控制寄存器	F651H	ENOWR	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0
DMA_OWR_STA	OW-DMA 接收状态寄存器	F652H	-	-	-	-	-	-	RXLOSS	OWRIF	xxxx,xx00
DMA_OWR_AMTH	OW-DMA 接收传输总字节数高位	F654H									0000,0000
DMA_OWR_AMT	OW-DMA 接收传输总字节数低位	F655H									0000,0000
DMA_OWR_DONEH	OW-DMA 接收传输完成字节数高位	F656H									0000,0000
DMA_OWR_DONE	OW-DMA 接收传输完成字节数低位	F657H									0000,0000
DMA_OWR_RXAH	OW-DMA 接收接收地址高位	FA58H									0000,0000
DMA_OWR_RXAL	OW-DMA 接收接收低地址低位	FA59H									0000,0000

R.5 单总线 DMA 寄存器说明

R.5.1 OWT_DMA 配置寄存器（DMA_OWT_CFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_OWT_CFG	F640H	OWTIE	-	-	-	OWTIP[1:0]		OWTBAP[1:0]	

OWTIE：OWT_DMA 中断使能控制位

0：禁止 OWT_DMA 中断

1：允许 OWT_DMA 中断

OWTIP[1:0]：OWT_DMA 中断优先级控制位

OWTIP[1:0]	中断优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

OWTBAP[1:0]：OWT_DMA 数据总线访问优先级控制位

OWTBAP [1:0]	总线优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

R.5.2 OWT_DMA 控制寄存器（DMA_OWT_CR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_OWT_CR	F641H	ENOWT	TRIG	-	-	-	-	-	-

ENOWT：OWT_DMA 功能使能控制位

0：禁止 OWT_DMA 功能

1：允许 OWT_DMA 功能

TRIG：OWT_DMA 发送触发控制位

0：写 0 无效

1：写 1 开始 OWT_DMA 自动发送数据

R.5.3 OWT_DMA 状态寄存器（DMA_OWT_STA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_OWT_STA	F642H	-	-	-	-	-	TXOVW	-	OWTIF

OWTIF：OWT_DMA 中断请求标志位，当 OWT_DMA 数据发送完成后，硬件自动将 OWTIF 置 1，若使能 OWT_DMA 中断则进入中断服务程序。标志位需软件清零

TXOVW：OWT_DMA 数据覆盖标志位。OWT_DMA 正在数据传输过程中，写 OWDR 寄存器再次触发发送数据时，会导致数据传输失败，此时硬件硬件自动将 TXOVW 置 1。标志位需软件清零

R.5.4 OWT_DMA 传输总字节寄存器（DMA_OWT_AMT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_OWT_AMTH	F644H	AMT[15:8]							
DMA_OWT_AMT	F645H	AMT[7:0]							

DMA_OWT_AMT[15:0]：设置需要自动发送数据的字节数。

R.5.5 OWT_DMA 传输完成字节寄存器（DMA_OWT_DONE）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_OWT_DONEH	F646H	DONE[15:8]							
DMA_OWT_DONE	F647H	DONE[7:0]							

DMA_OWT_DONE[15:0]：当前已经发送完成的字节数。

R.5.6 OWT_DMA 发送地址寄存器（DMA_OWT_TXAx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_OWT_TXAH	F648H	ADDR[15:8]							
DMA_OWT_TXAL	F649H	ADDR[7:0]							

DMA_OWT_TXA：设置自动发送数据的源地址。执行 OWT_DMA 操作时会从这个地址开始读数据。

R.5.7 OWR_DMA 配置寄存器（DMA_OWR_CFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_OWR_CFG	F650H	OWRIE	-	-	-	OWRIP[1:0]		OWRBAP[1:0]	

OWRIE：OWR_DMA 中断使能控制位

0：禁止 OWR_DMA 中断

1：允许 OWR_DMA 中断

OWRIP[1:0]：OWR_DMA 中断优先级控制位

OWRIP[1:0]	中断优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

OWRBAP[1:0]：OWR_DMA 数据总线访问优先级控制位

OWRBAP [1:0]	总线优先级
00	最低级（0）
01	较低级（1）
10	较高级（2）
11	最高级（3）

R.5.8 OWR_DMA 控制寄存器（DMA_OWR_CR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_OWR_CR	F651H	ENOWR	-	TRIG	-	-	-	-	CLRFIFO

ENOWR：OWR_DMA 功能使能控制位

0：禁止 OWR_DMA 功能

1：允许 OWR_DMA 功能

TRIG：OWR_DMA 接收触发控制位

0：写 0 无效

1：写 1 开始 OWR_DMA 自动接收数据

CLRFIFO：清除 OWR_DMA 接收 FIFO 控制位

0：写 0 无效

1：开始 OWR_DMA 操作前，先复位 OWR_DMA 内置的 FIFO 指针

R.5.9 OWR_DMA 状态寄存器（DMA_OWR_STA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_OWR_STA	F652H	-	-	-	-	-	-	RXLOSS	OWRIF

OWRIF：OWR_DMA 中断请求标志位，当 OWR_DMA 接收数据完成后，硬件自动将 OWRIF 置 1，若使能 OWR_DMA 中断则进入中断服务程序。标志位需软件清零

RXLOSS：OWR_DMA 接收数据丢弃标志位。OWR_DMA 操作过程中，当 XRAM 总线过于繁忙，FIFO 中的数据来不及写入 XRAM 导致 OWR_DMA 新接收的数据自动丢弃时，硬件硬件自动将 RXLOSS 置 1。标志位需软件清零

R.5.10 OWR_DMA 传输总字节寄存器（DMA_OWR_AMT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_OWR_AMTH	F654H	AMT[15:8]							
DMA_OWR_AMT	F655H	AMT[7:0]							

DMA_OWR_AMT[15:0]：设置需要自动接收数据的字节数。

R.5.11 OWR_DMA 传输完成字节寄存器（DMA_OWR_DONE）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_OWR_DONE	F656H	DONE[15:8]							
DMA_OWR_DONE	F657H	DONE[7:0]							

DMA_OWR_DONE[15:0]：当前已经接收完成的字节数。

R.5.12 OWR_DMA 接收地址寄存器（DMA_OWR_RXAx）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_OWR_RXAH	FA6DH	ADDR[15:8]							
DMA_OWR_RXAL	FA6EH	ADDR[7:0]							

DMA_OWR_RXA：设置自动接收数据的目标地址。执行 OWR_DMA 操作时会从这个地址开始写数据。

附录S USB 原理及应用线上培训教程（何老师）

https://www.stcaimcu.com/forum.php?mod=viewthread&tid=4526&extra=page%3D1

在 Ai8051U 系列 32 位单片机集成了 USB2.0 模块，进一步扩展了 32 位 8051 单片机的应用范围。本节主要介绍了 USB 原理以及基于 USB 的 HID 和 CDC 应用实现。

本章主要内容包括 USB 协议概述、USB 2.0 程序设计实现、人机交互设备原理、人机交互设备程序设计、通信设备类原理、通信设备类程序设计，以及 USB 寄存器。

S.1 USB 协议概述

本节对 USB2.0 协议进行了简要概述，以帮助读者从整体上对 USB2.0 协议有一个初步了解。本节内容主要包括 USB 通信基础、USB 通信的时域构成、USB 通信模型、USB 标准请求、USB 通信过程，以及 USB 描述符。

S.1.1 USB 通信基础

USB 通信基础主要涉及到 USB 系统组成、USB 架构、USB 物理接口、USB 编码方式、USB 总线状态、USB 速度和 USB 电源。

1. USB 系统组成

USB 系统分为 USB 主机和 USB 设备。

USB 主机：提供 USB 接口和接口管理功能的硬件、软件，固件的复合体。PC 机或 OTG 设备，系统中只能有一个主机，并且与设备进行的通信是从主机的角度进行的。例如，主机从设备接收数据为 IN 类型，主机发送数据给设备为 OUT 类型。USB 主机的硬件主要包括 USB 主控制器和 USB 根集线器。

USB 设备：1.集线器 HUB：扩展主机接口，设备可以通过其接入主机

2.功能设备，如 U 盘，USB 摄像头，HID 键盘鼠标等。

物理连接：即 USB 电缆，USB 使用差分信号传输数据，USB 全速/高速模式电缆必须外层屏蔽铜质传输线，且差分数据线双绞。一条 USB 的传输线分别由地线、电源线、D+、D-四条线构成，其中 D+、D-是差分输入线，使用的电压为 3.3V，而电源线与地线可向设备提供 5V 电压，最大电流 500mA。

2. USB 架构

（1）USB 主控制器

USB 主控制器是 USB 主机上的控件，USB 主控制器是具有软件驱动器层的硬件芯片组，用于执行以下任务：检测 USB 设备的插入和拔出、管理主机和设备间的数据流、提供并管理所连接设备的电源、监视总线上的活动。主机可以有一个或多个主控制器。通过使用外部 USB 集线器，每个控制器最多可以连接 127 个设备。具体为以下几种主控制器。

OHCI（open host controller interface）是支持 USB1.1 的标准，但它不仅仅是针对 USB，还支持其他的一些接口，如 Apple 的火线（firewire，EEE 1394）接口。与 UHCI 相比，OHCI 的硬件复杂，硬件做的事情更多，所以实现对应软件驱动的任务，相对较简单。主要用于非 X86 的 USB，如扩展卡、嵌入式开发板的 USB 主控。

UHCI（universal host controller interface），是 Intel 主导的对 USB1.0、1.1 的接口标准，与 OHCI 不兼容。UHCI 的软件驱动的任务重，需要做得比较复杂，但可以使用较便宜、较简单的硬件 USB

控制器。Intel 和 VIA 使用 UHCI，而其余的硬件提供商使用 OHCI。

EHCI（enhanced host controller interface），是 Intel 主导的 USB2.0 的接口标准。EHCI 仅提供 USB2.0 的高速功能，而依靠 UHCI 或 OHCI 来提供对全速（full-speed）或低速（low-speed）设备的支持。

xHCI（eXtensible host controller interface），USB3.0 的接口标准，它在速度、节能、虚拟化等方面都比前面 3 种有了较大的提高。xHCI 支持所有种类速度的 USB 设备（USB3.0 Super-Speed，USB 2.0 Low-，Full-，and High-speed，USB 1.1 Low-and Full-speed）。xHCI 的目的是为了替换前面 3 种 USB 主控制器（UHCI/OHCI/EHCI）。

（2）USB 拓扑结构

USB 系统包括一台主机（一般是一台个人计算机（PC））和多个通过分层星形拓扑连接的外围设备。该拓扑也可以包括集线器，从而能够提供更多与 USB 系统的连接点。主机本身包含两个组件，即主控制器和根集线器。

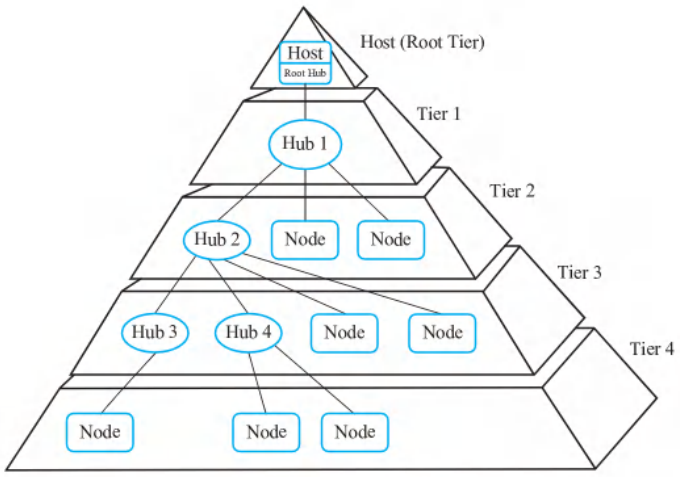


图 15.1　USB 拓扑结构

每个 USB 系统只允许有一个 HOST（主机）。允许的最大层数为 7 层（包含主机）；每层的电缆最大长度为 5 米，电缆总长度为 30 米；每层最大允许接 5 个 DEVICE（设备）；

（3）USB 协议分层

端点（endpoint）：USB 通信的基本单元，设备端点是 USB 设备中一个独特的可寻址部分，它作为主机和设备间通信流的信息源或库。

总线接口层：提供了物理连接、电气信号和数据包连接。该层由设备硬件处理，并通过设备的外部接口完成。

接口层（interface）：描述 USB 设备的具体功能，例如一个 USB 设备既有键盘的功能又有存储功能，该设备就有两个接口。

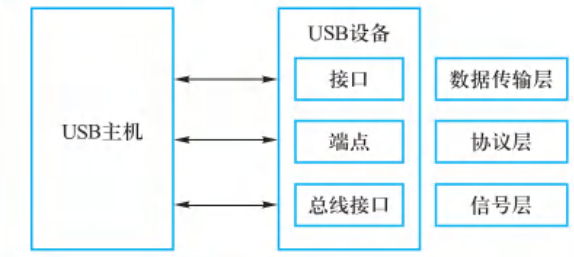


图 15.2　USB 协议分层

3. USB 物理接口

（1）USB 线缆

一个 USB 线缆包含一个绝缘套保护的多个组件。该绝缘套下面是一个包含了一个带有铜面的外部扩展板。外部扩展板内包含多个连线：一个铜排流线、一个 Vbus 线（红色）和一个接地线（黑色）。由铝制成的内部扩展板包含一对用双绞线制成的数据线，如图 15.3 所示。有一个 D+线（绿色）和一个 D-线（白色）。VBUS 线为所有相连设备提供了恒定的 4.40V、5.25V 电源。当 USB 为设备提供 5.25V 电源时，数据线（D+和 D-在 3.3V 电压下工作）

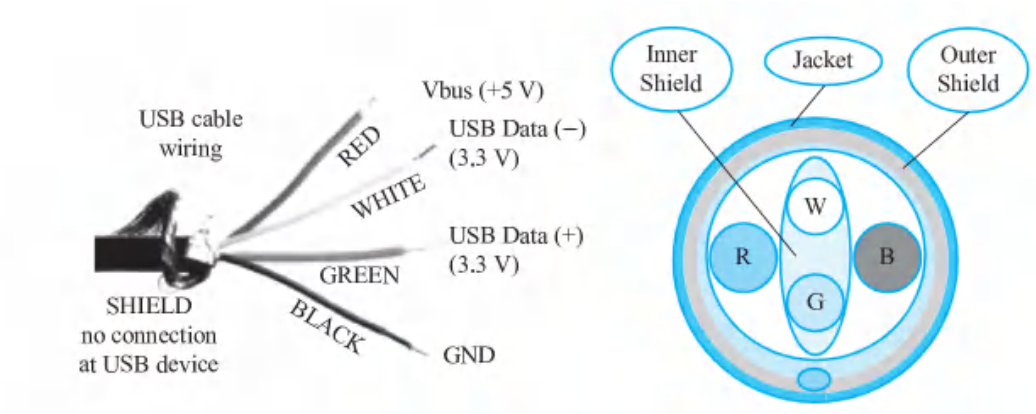
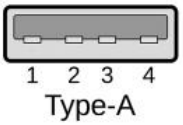


图 15.3　USB 线缆内部

（2）USB 连接器

由于本次程序设计仅涉及 TYPE-A 与 Micro-B 连接器，因此本文仅介绍这两种连接器，其余连接器的详细资料读者可查看 USB 中文网或 USB 官网进行了解。

表 15.1　USB 连接器

类型	引脚图	连接器图
TYPE-A		
Micro-B		

引脚定义及颜色如表 15.2、15.3 所示。

表 15.2 TYPE-A 连接器引脚定义

引脚	名称	线缆颜色	描述
1	VBUS	红色或者/橙色	+5V 供电
2	D-	白色或者/金色	差分数据-
3	D+	绿色/绿色	差分数据+
4	GND	黑色/蓝色	地

表 15.3 Micro-B 连接器引脚定义

引脚	名称	线缆颜色	描述
1	VBUS	红色	+5 供电
2	D-	白色	差分数据-
3	D+	绿色	差分数据+
4	ID	N/A	区分另一端接口类型 A 接口（主机）：接地 B 接口（设备）：不连接
5	GND	黑色	地

4. USB 编码方式

USB 采用不归零反转差分（non-return to zero indicates，NRZI）编码方式，在该编码方案中如果电压电平不变，则表示逻辑 1；如果电压电平变化，则表示逻辑 0，为保证定时信息的准确，需要在每 6 个 1 插入一个逻辑 0 保证同步。图 15.4 为 NRZI 码与待发送数据波形：

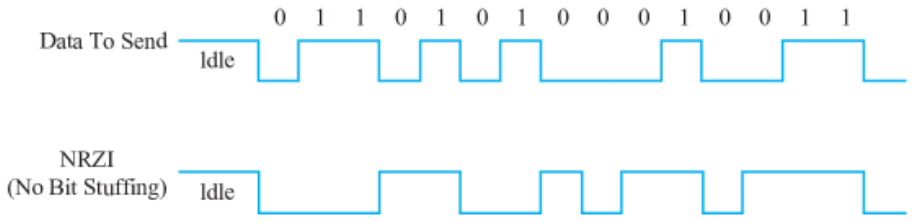


图 15.4 NRZI 码与待发送数据波形

通过在 6 个连续的逻辑 1 后面插入一个逻辑 0 可以实现位填充。USB 硬件上的接收器会自动检测额外位，并忽略它。使用差分 D+和 D-信号是为了抑制共模噪声。

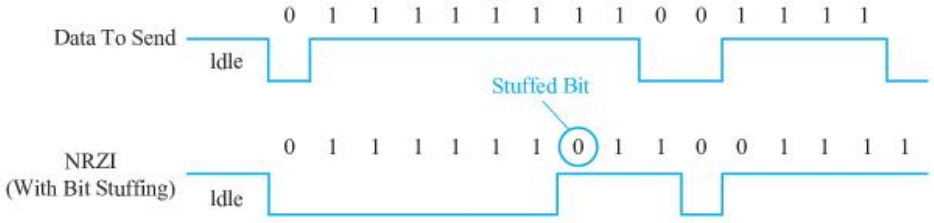


图 15.5 6 个连续的 1 的 NRZI 编码的逻辑电平

5. USB 总线状态

D+线和 D-线上的高低电平的组合表示了 USB 总线的不同状态，其状态可描述为表 15.4

表 15.4 USB 总线状态与指示

总线状态	指示
差分 1	D+为高电平，D-为低电平
差分 0	D+为低电平，D-为高电平
单端 0（SE0）	D+和 D-为低电平
单端 1（SE1）	D+和 D-为高电平

总线状态	指示
J 状态： 低速 全速 高速	 差分 0 差分 1 差分 1
K 状态： 低速 全速 高速	 差分 1 差分 0 差分 0
恢复状态：	K 状态
包起始（SOF）	USB 数据总线从 idle 状态切换到 K 状态。
包末尾（EOP）	SEO 持续两个基本时间单位，以及 J 状态持续一个时间单位。

表 15.4 不同总线状态的详细内容见 USB 协议官方文档。读者若对 USB 线缆上的信号感兴趣可以使用示波器自行测量。

6. USB 速度

USB 规范已经为 USB 系统定义了以下四种速度模式：低速（Low-Speed）、全速（Full-Speed）、高速（Hi-Speed）和超高速（SuperSpeed）。

低速、全速和高速设备的速率分别为 1.5Mb/s、12Mb/s 和 480Mb/s。但是，这些指的是总线速率，并不是数据速率。实际的数据速率受总线加载速度、传输类型、开销、操作系统等因素的影响。数据传输则受以下内容的限制：

低速设备：如键盘、鼠标和游戏等外设。总线速率：1.5Mb/s。最大的有效数据速率：800 B/S。

全速设备：如手机、音频设备和压缩视频口总线速率：12Mb/s。最大的有效数据速率：1.2 MB/s。

高速设备：如视频、影像和存储设备口总线速率：480Mb/s。最大的有效数据速率：53 MB/s。

本文仅介绍 USB 的全速模式以及 USB 低速模式的连接图。USB 全速模式是 USB 设备在 D+线上有一个 1.5K 的上拉电阻。USB 低速模式是 USB 设备在 D-线上有一个 1.5K 的上拉电阻。

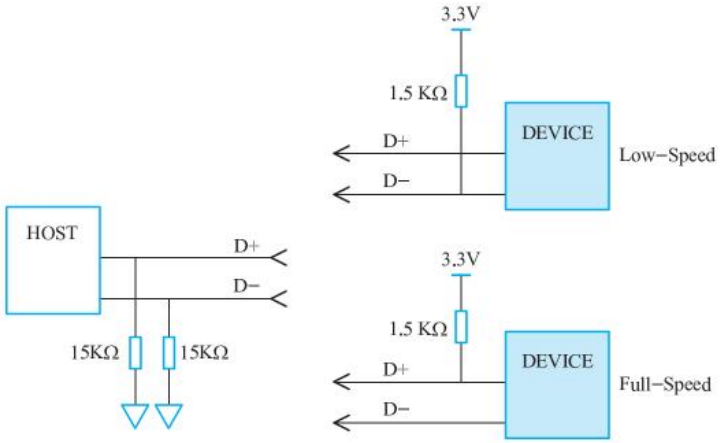


图 15.6　USB 的全速模式以及 USB 低速模式的连接图

USB 进行枚举时需要使用上拉电阻。否则，USB 会认为总线上没有连接设备。部分设备要求在 D+/D-信号线上使用一个外部上拉电阻。但是单片机已经带有所需的内部上拉电阻因此不需要外部上拉电阻。

7. USB 电源

USB 供电方式可以是：总线供电、自供电或者二者相结合的方式。依据程序设计此处仅给出总线供电方式的说明。

总线供电的设备共有以下两种：高功耗和低功耗设备。低功耗设备最多 100mA 的电流，高功耗设备最多 500mA 的电流。电流超过 500mA 的设备要自供电。

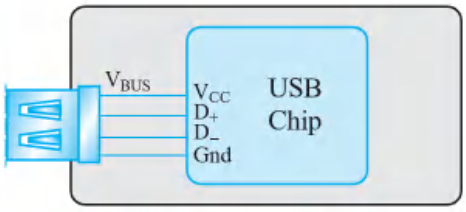


图 15.7　USB 总线供电设备

S.1.2　通信的时域构成

从时间角度来看，USB 通信由一系列帧构成。每一帧都有一个帧开始（SOF），随后是一个或多个数据操作。每一个数据操作都由一系列数据包构成。一个数据包由一个同步信号开始，结尾是一个数据包结束（EOP）信号，一个数据操作至少有一个令牌数据包。具体的数据操作可能有一个或多个数据数据包；一些数据操作可能会有一个握手数据包，也可能无握手数据包。

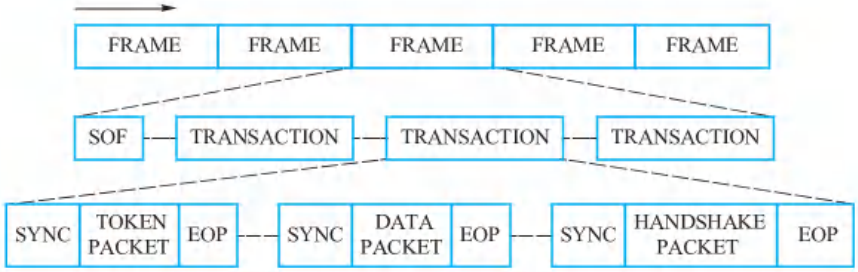


图 15.8　从时间角度观察 USB 通信

数据操作是对数据包进行交换的操作，该操作使用了三种不同的数据包：一个令牌数据包、一个数据数据包（可选）和一个握手数据包。

这些数据操作都在各个帧内进行，始终不会超过帧（除了高速同步传输以外）或终止其他数据操作。图 15.9 显示了一次数据操作的框图。

每个数据包可能带有不同的信息块。所带有的信息会因数据包类型的不同而异。数据包的结构如图 15.10 所示。图 15.10 可作为数据包的模板，具体的数据包构成需要依据具体的数据包类型。

- PID：数据包 ID，共 8 位，其分为 4 个类型位和 4 个错误检测位。这些位将数据传输定文为 IN/OUT/SETUP/SOF。
- ADDR：可选的设备地址，共 7 位，最多可支持 127 个设备。
- EP：可选的端点地址，其 4 位，最多支持 16 个端点。USB 规范支持多达 32 个端点。虽然 4 位地址最多仅支持 16 个端点，但我们具有一个 IN PID 和一个 OUT PID，它们各自使用了端点地址 1 到 16，因此共有 32 个端点。注意，它表示端点的地址，而不是端点的编号。
- PAYLOAD DATA：可选的加载数据，0~1023B。
- CRC：可选，5 或 16 位。

图 15.9　数据操作框图

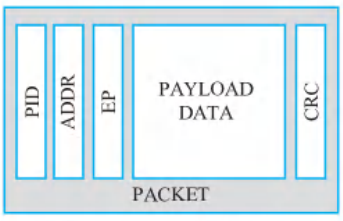


图 15.10　USB 数据包内容

1．数据包类型

数据包类型共有四种，数据包类型由 PID 决定

令牌（token）数据包：开始数据操作，指定与传输有关的设备，始终由主机发送。

数据（data）数据包：传输加载数据、由主机或设备发送。

握手（handshake）数据包：确认已接收到无错误的数据、由接收方发送。

特殊数据包：支持多种不同的速度、由主机传输给集线器设备。

如上所述，数据包中的任意信息（除了 PID 之外）均是可选的。令牌、数据和握手数据包具有不同的信息组合。令牌数据包、数据数据包和握手数据包部分对各数据包所带有的信息进行了介绍。

（1）令牌数据包：令牌数据包始终由主机发送，用于定义总线上的数据传输。令牌数据包的类型取决于所执行的传输类型。主机向设备发送 IN 令牌数据包，用于从设备读取数据。主机向设备发送 OUT 令牌数据包，用于将数据从主机传输给设备。主机向设备发送 SETUP 令牌数据包，用于将主机的请求传输给设备（有关该数据包的具体通信流程详见 USB 标准请求以及 USB 通信过程这节）。SOF 令牌数据包用于确定帧的起始位置。IN、OUT 和 SETUP 令牌数据包都有一个 7 位设备地址（ADDR）、4 位端点 ID（EP）和 5 位 CRC。图 15.11 显示了这四个令牌数据包的结构图。

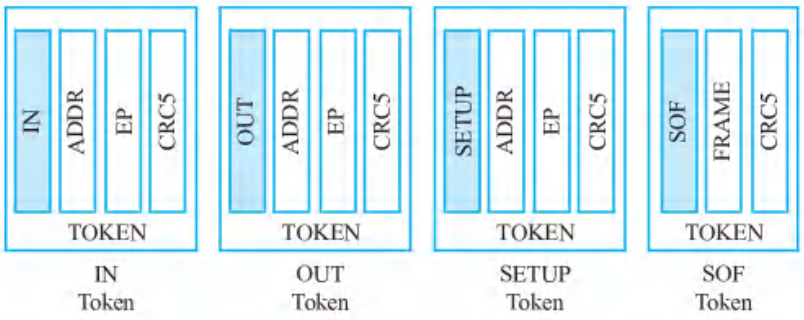


图 15.11　USB 令牌数据包类型

SOF 令牌数据包帮助设备确定帧的起始位置，从而与主机进行同步。该数据包还有助于防止设备进入挂起模式（经过 3ms 后，如果设备未收到 SOF 设备将会进入挂起模式）。SOF 数据包适用于全速和高速设备，并且每隔 1ms 发送一次，如图 15.12 所示。该数据包具有一个 8 位的 SOF PID、11 位的帧计数值（FRAME），达到最大值时进行反转，还有一个 5 位的 CRC。CRC 是该数据包使用的唯一一个错误检测方法。传输 SOF 数据包时，不会使用握手数据包，高速通信使用了更小的时间单位，即微帧。对于高速设备，SOF 每经过 125us 发送一次，而帧计数值则每经过 1ms 递增“1”。

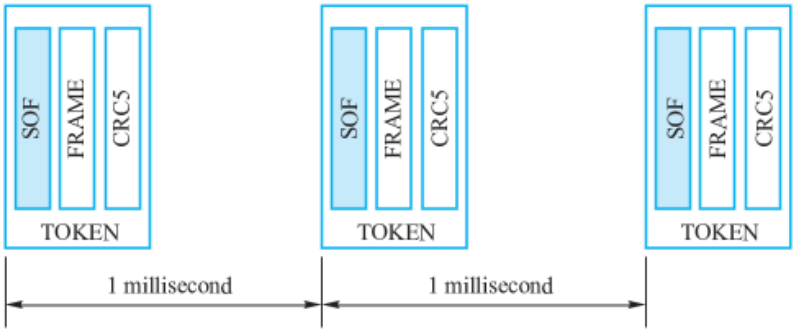


图 15.12　全速设备的 USB SOF 传输

（2）数据数据包：加载数据的大小会因传输类型的不同而异，数据大小的范围为 0-1024B。在每一个数据数据包成功传输后，数据包 ID 在 DATA0 和 DATA1 之间切换，数据包由一个 16 位 CRC 结束。数据数据包结构如图 15.13 所示。

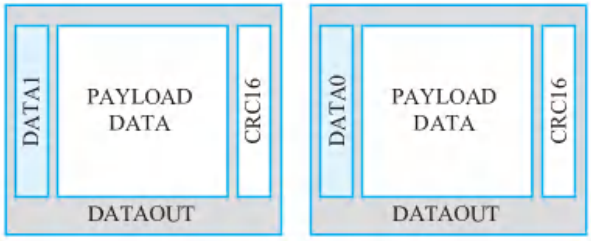


图 15.13　USB 数据数据包

在每一个数据数据包成功传输后，主机和设备将对数据包进行切换。例如，上一个数据包 ID 为 DATA0，下一个数据包 ID 则为 DATA1。数据包 ID 切换的优点在于它可作为附加的错误检测方法。如果接收到的数据包 ID 同预期的不一样，则设备可判断传输中发生了错误，并能进行适当的处理。数据包 ID 切换的示例：ACK 在发送后，若未能正常接收，发送方将数据从 1，更新为 0，但接收方则没有进行相应的更新，而仍然保持为 1，在下一个数据步骤中，主机和设备将不再同步。图 15.14 显示了一个 USB 传输中的数据切换示例。在本文所有的图中，白色框表示来自主机的数据包，黑色框则表示来自设备的数据包。

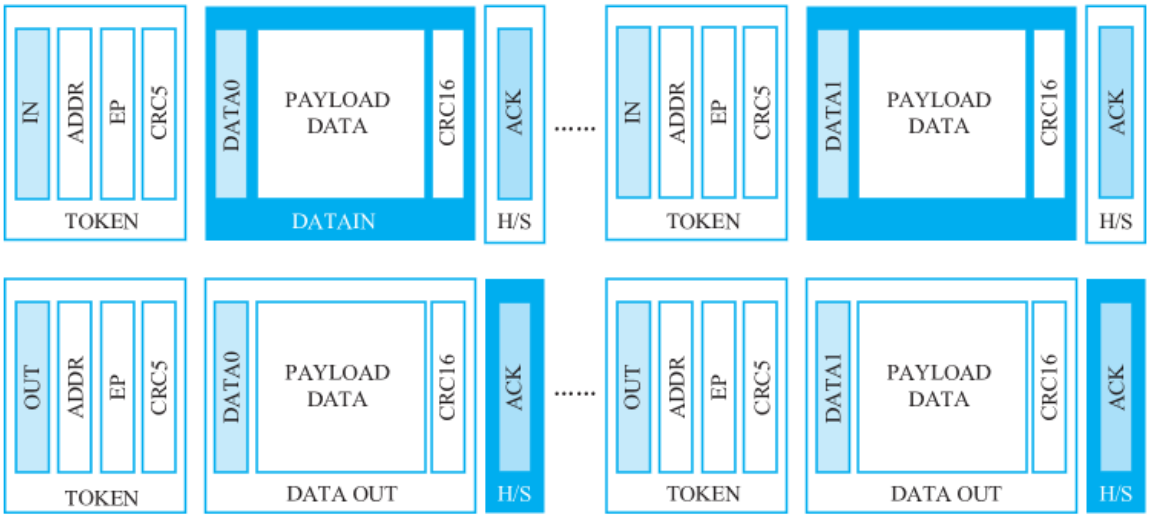


图 15.14　数据切换示例

（3）据手数据包：握手数据包指示数据操作的结束。每个握手数据包都带有一个 8 位数据包 ID，并由传输中的接收方发送，如图 15.15 所示。不同 USB 速度都有不同的握手数据包响应选项。所支持的类型由 USB 速度决定：

- ACK：确认数据操作成功完成。（低速/全速/高速）
- NAK：否定确认。（低速/全速/高速）
- STALL：设备发送错误指示。（低速/全速/高速）
- NYET：表示设备当前未能接收其他数据数据包（仅高速）

（4）特殊数据包：USB 规范定义了四种特殊数据包，如图 15.16 所示。

- PRE：主机向集线器发送的数据包，用于指示下一个数据包是低速的
- SPLIT：发送在令牌数据包之前，用于指示一个分割数据操作。（仅高速）
- ERR：由集线器返回的数据包，用于报告分割数据操作中发生了错误。（仅高速）
- PING：接收到 NYET 握手数据包后，检查批量传输 OUT 或控制写人的状态。（仅高速）

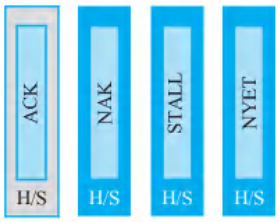


图 15.15　握手数据包的指示

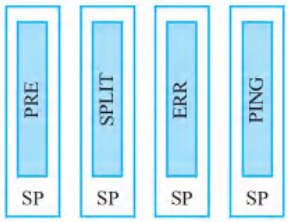


图 15.16　特殊数据包的指示

2．数据传输类型

USB 数据传输是指主机和设备之间的数据传输方式。一共有三种不同的数据传输类型，它们经常使用不同名称来代表相同的概念。这三种不同的数据传输类型具体如下。

（1）IN/读取/上行数据传输

IN、读取和上行是专用术语，表示从设备到主机的数据传输方式。主机通过向设备发送一个 IN 令牌数据包，将启动此类数据传输。设备将发送一个或多个数据包，主机则发送一个握手数据包来作出响应。在图 15.17 中，白框显示的是从主机发送的数据包，黑框显示的是从设备发送的数据包。

图 15.17　IN/读取/上行框图

在图 15.18 中，设备发送了 NAK 作为响应，说明主机发送请求时，它还没准备好发送数据。主机持续发出请求，如果设备已经准备好，它将发送一个数据包来响应主机。然后，主机将发送一个 ACK 握手数据包来确认接收到设备发送的数据。

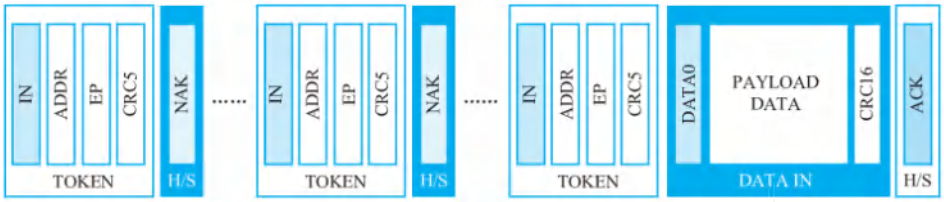


图 15.18　IN 数据传输示例

（2）OUT/写入/下行数据传输

OUT、写入和下行是专用术语，指的是从主机到设备的数据传输方式。在这种数据传输类型中，主机将发送相应的令牌数据包（包括 OUT 或 SETUP），然后发送一个或多个数据包。接收设备将发送相应的握手数据包，以结束数据传输。在图 15.19 中，白框显示的是从主机发送的数据包，黑框显示的是从设备发送的数据包。

图 15.19　OUT/写入/下行框图

在图 15.20 中，主机将发送 OUT 令牌数据包和 DATA0 数据包，如果设备未准备好，主机则会接收到设备所发送的 NAK 包。然后，主机会重新发送数据。注意，设备拒绝接收来自主机的数据，不会改变数据包 ID 的状态。如果主机再次尝试发送数据，设备若准备好，设备将发送一个 ACK 信号来响应主机，表示 OUT 数据传输已经成功。

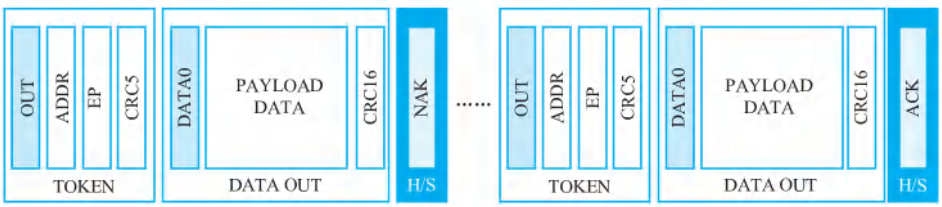


图 15.20　OUT 数据传输示例

S.1.3　USB 通信模型

本节介绍的是 USB 通信模型中的逻辑通信模型，该模型中的许多概念将会在后续的程序设计中用到，希望读者能仔细阅读。

1. USB 管道

USB 设备的通信通过管道实现。这些管道是主控制器到可寻址缓冲区（称为端点）间的连接路径。一个端点会保存收到来自主机的数据并保存将要发送给主机的数据。一个 USB 设备能够具有多个端点，并且每个端点都有相应的管道，如图 15.21 所示

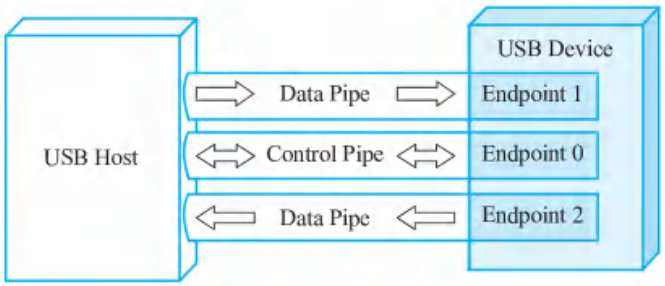


图 15.21　USB 管道与端点模型

USB 系统中的管道共有两种，分别为控制管道和数据管道。USB 规范中定义了四种不同的数据传输类型。使用哪个管道由数据传输类型决定。

控制传输：用于将指令发送到设备上，进行查询并且配置设备。该传输使用了控制管道。

中断传输：用于发送少量的突发性数据，并且保证传输延迟最小。该传输使用了数据管道。

批量传输：利用了全部可用的 USB 带宽来传输大量数据，但传输速度或延迟得不到保证该传输使用了数据管道。

同步传输：数据传输采用了得到保证的传输速率。随着传输延迟和总线带宽的保证，传输时间也得到保证。同步传输没有错误纠正功能，因此在重新发送有误的数据包过程中，不能停止传输。该传输使用了数据管道。

每个设备都有一个控制管道，用于控制发送给设备或从设备接收信息的状况。设备可以有任意多个数据管道，设备可以通过中断、批量或同步传输类型进行数据传输。控制管道是 USB 系统中唯一一个双向管道，所有的数据管道均是单向的。

每个端点均可通过设备地址（由主机分配）和端点编号（由设备分配）进行访问。主机会通过向设备发送令牌数据包来获取设备的地址和端点编号。

USB 设备首次与主机相连时，将启动 USB 枚举过程。枚举是设备和主机间进行的信息交换过程，包含用于识别设备的信息。此外，枚举过程还分配设备地址、读取描述符（提供有关设备信息的数据结构），并分配和加载设备驱动程序。整个过程需要数秒时间。更多有关信息，请参考 USB 枚举和配置一节。完成该过程后，设备可以向主机传输数据，例如：我们接入 U 盘，主机完成枚举过程后，我们就可以正常使用 U 盘与主机进行数据互传。

2. USB 端点

根据 USB 规范，设备端点是 USB 设备中一个独特的可寻址部分，它作为主机和设备间通信流的信息源或库。简而言之 USB 端点中有一个叫做端点缓冲区的重要组成部分，USB 主机端通过发送 USB 数据到 USB 设备端的缓冲区实现数据的收到功能。而 USB 设备通过向缓冲区写入数据实现 USB 设备向主机发送数据的功能。

USB 规范定义了四种端点，并根据类型以及所支持的设备速度限制了数据包的尺寸。根据设计要求，开发者使用端点描述符用以指出端点类型以及数据包最大尺寸。四种端点和各自的特性如下：

控制端点：这些端点支持控制传输（即所有设备支持的传输）。控制传输通过总线发送和接收设备的信息。它的优点是可以保证传输准确。它能够立即检测到错误的发生，并重新发送数据。控制传输在低速和全速设备上使用 10%的保留带宽（在高速设备上为 20%）并提供 USB 系统级控制。

中断端点：这些端点支持中断传输。这种传输非常适合需要使用高度可靠的方式来传输少量数据的设备。它通常用于 HID 设计。这种传输的名称可引起误会。实际通信中，使用的不是并中断，而是轮询。进行该传输时，主机将在规定的时间间隔内检查数据。通过及时检测错误并重新传输数据，该传输可确保数据操作的准确性。在低速和全速设备上，中断传输使用带宽的 90%，而在高速设备上，所用的带宽为 80%。同步端点与其共享该带宽。

中断端点的数据包最大尺寸与设备的速度相关。高速设备支持最大为 1024 字节的数据包。全速设备支持最大为 64 字节的数据包。低速设备支持最大为 8 字节的数据包。

批量端点：这些端点支持批量传输，即是在高度可变的时间内传输大量数据并且可用任意大带宽空间的传输。它们是 USB 设备的最通用传输类型。因为用于批量传输的带宽并不是固定的，该传输的传送时间也是可变的。传送时间取决于总线上的可用带宽，由于该因素，便不能预测实际的传送时间。通过及时检测错误并重新传输数据，该传输可确保数据操作的准确性。批量传输非常适合对时间没有严格要求的大量数据传输。

批量端点的数据包最大尺寸与设备速度相关。高速设备支持最大为 512 字节的数据包。全速设备支持最大为 64 字节的数据包。低速设备不支持批量传输。

同步端点：这些端点支持同步传输，即具有预定带宽的连续性实时传输。由于同步传输没有错误恢复机制和握手数据包，它们需要支持容忍错误的数据流。错误由 CRC 字段检测，但不会被修改。因此，同步传输可保证传输速度，但以数据的准确性作为代价。流式音乐或视频即是使用同步端点的应用示例，因为我们的耳朵和眼睛通常忽略偶尔被错过的数据。在低速和全速设备上，同步传输使用带宽的 90%（在高速设备上，所用的带宽为 80%），中断传输与其共享该带宽。

高速设备支持最大为 1024 字节的数据包。全速设备支持最大为 1023 字节的数据包。低速设备不支持同步传输。有关同步传输，请注意一些重点内容。为了保证数据传输，通常需要使用三个缓冲区，一个正在传输数据，一个已加载数据和一个正在进行加载数据。

表 15.5 端点传输类型特性

传输类型	控制	中断	批量	同步
适用场合	设备初始化和管理	鼠标和键盘	打印机和批量存储	流式音频和视频
支持低速	有	有	无	无
修改错误	有	有	有	无
保证传输速度	无	无	无	有
使用固定带宽	有（10%）	有（90%）[1]	无	有（90%）[1]
减少延迟时间	无	有	无	有
传输的最大尺寸	64 字节	64 字节	64 字节	1023 字节（FS） 1024 字节（HS）
传输的最高速度	832 KB/s	1.216MB/s	1.216 MB/s	1.023 MB/s

[1]同步和中断端点的共享带宽。

USB 枚举和配置一节介绍了设备向默认地址做出响应的步骤。枚举过程中，该事件在主机读取端点描述符等其他描述符信息之前发生。在枚举过程中，需要使用一套专用的端点用于与设备进行通信，这些专用的端点（统称为控制端点（端点 0））被定义为端点 0 IN 和端点 0 OUT。虽然端点 0 IN 和端点 0 OUT 是两个不同的端点，但对开发者来说，它们的构建和运行方式是一样的。每一个 USB 设备都需要支持端点 0。因此，该端点不需要使用独立的描述符。

除了端点 0 外，特定设备所支持的端点数量将由各自的设计要求决定。简单的设计（如鼠标）可能仅要一个 IN 端点。复杂的设计可能需要多个数据端点。USB 规范对高速和全速设备的端点数量进行了限制，即每个方向最多使用 16 个端点（16 个 IN、16 个 OUT，总共为 32 个），其中不包含控制端点 0 IN 和 0 OUT 在内。低速设备仅能使用两个端点。USB 类设备可对端点数量设定更严格的限制。例如，低速人机界面设备（HID）设计的端点可能不超过两个，通常有一个 IN 端点和一个 OUT 端点。数据端点本身具有双向特性，只有对它们进行配置后才支持单向传输（具有单向特性）。（例如，端点 1 可作为 IN 或 OUT 端点使用，设备的描述符将正式使其成为一个 IN 端点。）

各端点使用循环冗余校验（CRC）来检测传输中发生的错误。USB 硬件会进行 CRC 检验。如果两者匹配，那么接收方将发出一个 ACK。如果两者匹配失败，便不会发出任何握手数据包。在这种情况下，发送方将重新发送数据。

3. USB 传输方式

在 USB 管道和 USB 端点一节我们可以知道：端点的类型和 USB 数据传输的方式是相对应的。本节将详细讲述 USB 的三种传输方式。

1）控制传输

控制传输是一种特殊的传输方式。当 USB 设备初次连接主机时，用控制传输传送控制命令等对设备进行配置。同时设备接入主机时，需要通过控制传输去获取 USB 设备的描述符以及对设备进行识别，在设备的枚举过程中都是使用控制传输进行数据交换。在后面程序设计一节中将会对控制传输的具体实现做出详细的解析。

控制传输主要应用于 USB 设备的枚举过程，其他应用场合，读者可自行了解，本文不作过多介绍。

控制传输最大包长度不超过控制端点的大小。

控制传输由三个阶段构成，分别是建立阶段（或称之为设置阶段）、数据阶段和状态阶段。程序设计中控制传输部分设备对请求处理函数将主要按这三个阶段进行编写。

（1）建立阶段

该过程具体描述为：

①主机发送令牌包：SETUP

②主机发送数据包：DATA0

③设备返回握手包：ACK 或不应答

注意：设备不能返回 NAK 或 STALL，即设备必须接收建立阶段的数据。设备只能使用 ACK 来应答（或者由于出错不应答）

（2）数据阶段

数据阶段是控制传输中可选的，需根据实际情况而定。数据阶段的通信是单向的，只能是主机发送数据给设备（OUT），或者设备发送数据给主机（IN）。如果通信方向发生了变化，则认为进入了状态阶段。数据阶段的第一个数据包必须为 DATA1，然后每次正确传输一个数据包后就在 DATA0 和 DATA1 之间交替。

（3）状态阶段

状态阶段的传输方向和数据阶段相反，即数据阶段是通信，则状态阶段必然是下行通信。且该阶段

只能使用 DATA1 数据包。

控制传输之所以如此复杂，乃是为了数据传输的完整性，确保数据传输的正确无误。USB 设备在枚举过程使用的乃是控制传输，且控制传输只能在端点 0 进行。

2）中断传输

中断传输一般用于小批量的和非连续的数据传输，通俗的来说就是用于数据量小的数据不连续的但实时性高的场合的一种传输方式，主要应用于 HID 设备中的 USB 鼠标和 USB 键盘等。

在中断端点中有介绍：USB 中断传输和我们传统意义上的中断不一样。它不是由设备主动地发起一个中断请求，主机响应，而是主机将在规定的时间间隔内检查数据。所以 USB 的中断传输的实际意义是实时轮询操作,即 USB 的中断传输是主机在一定的时间不断地主动轮询设备检查其是否有数据需要传输。

中断传输有 3 个重要参数需要在端点描述符中进行配置：即传输类型、每次传输的最大数据包大小、轮循时间间隔。

中断端点需指定一个范围在 1-255ms 内的轮询周期。主机保留足够的带宽以确保在指定频率上直接向中断端点发出 IN 或 OUT 事务。对于中断传输，如果传输的数据长度大于端点支持的最大包长度，这时一个中断传输内会有多个事务。在传输数据时，如果最后一个事务的数据长度小于端点支持的最大包长度，则认为数据传输完成。

对于 Ai8051U 的端点每个端点的大小为 64 字节，故端点支持的最大包长为 64 字节，若 Ai8051U 仅模拟 HID 键盘设备或 HID 鼠标设备，中断端点每次仅要传输 1 字节数据即可。

中断传输事务

USB 中断数据流传输包括 IN 传输和 OUT 传输，分别对应于数据的读和写，其也分为 3 个阶段，分别为令牌阶段、数据段和握手段。

中断传输和批量传输的结构基本一致，只是中断传输没有 ping 和 nyet 两种包。

当主机准备接收 USB 设备的中断端点的数据时，其发送 IN 令牌包，USB 设备响应并返回 DATAx 数据包，NAK 或 STALL 握手包。

当主机向 USB 设备的中断端点发送数据时，其发送 OUT 令牌包和 DATAx 数据包，而 USB 设备将向主机返回 ACK、NAK 和 STALL 握手包。如图 15.22 为一个中断传输事务在不同阶段数据包的构成。

令牌段（Taken）

- 主机发出令牌包，寻址设备。

数据段（Dala）

- 设备如果接收令牌包出错，无响应；
- 设备端点不存在，设备回复 STALL 包；
- 设备端点数据未准备好，设备回复 NAK 包；
- 设备端点数据准备好，设备回复数据包。

握手段（handshake）

- 主机如果接收数据包出错，无响应；
- 主机如果接收数据包正确，设备回复 ACK 包。

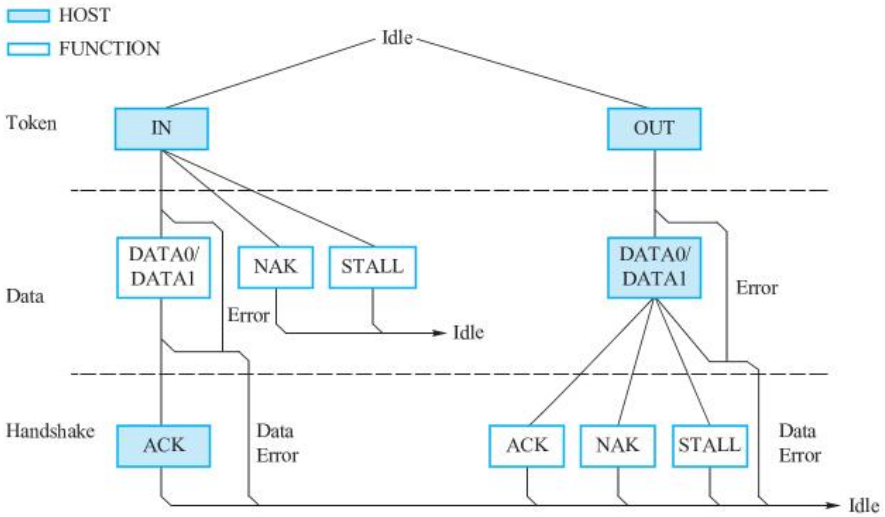


图 15.22　中断传输事务在不同阶段数据包的构成

3）批量传输

批量传输一般用于批量的和非实时的数据传输，通俗的来说就是用于数据量大但对时间要求又不高的场合的一种传输方式，类似用于 USB 打印机和 USB 扫描仪等等。

批量传输使用批量传输事务，一次批量传输事务分为三个阶段：令牌包阶段、数据包阶段握手包阶段。

批量传输分为批量读和批量写，批量读使用批量输入事务，批量写使用批量输出事务。注意：不论输入还是输出都是以主机为参考的。

对于批量传输，如果启动批量传输，如果 USB 总线中有多余的总线带宽，批量传输会立即执行，但当带宽比较紧张时，批量传输会把带宽让给其他传输类型。所以批量传输的优先级相对其它传输优先级比较低。

批量传输数据包

只有全速和高速设备可以使用批量传输，低速模式不支持批量传输。

高速模式中，传输的数据包大小固定为 512 个字节；

全速模式中，传输的数据包大小可在 8、16、32、64 字节中选择：

当为超高速设备时数据包最大长度为 1024 字节，批量传输端点应在其端点描述符中设置最大的数据包负载大小为 1024 字节。它还指定端点可以接受或者发送到超高速总线的突发大小。对于批量端点允许的突发大小应在 1 至 16 范围。

批量功能端点必须传输数据字段小于或等于 1024 字节的数据负载。如果批量传输有比之更多的数据，在突发事务交易的所有数据的有效大小必须为 1024 字节长度，除了突发的最后一个数据有效载荷，它可能包含未使用的数据空间。

如果传输的数据量大于端点所支持的最大数据包长度，USB 主控制器会把该数据按最大数据包长度分为多个批量数据包进行传输，最后一个批量传输长度可以小于或等于最大包长度。

批量传输流程：

批量传输数据流传输包括 IN 传输和 OUT 传输，分别对应于数据的读和写，其也分为 3 个阶段，分别为为令牌阶段、数据段和握手段。

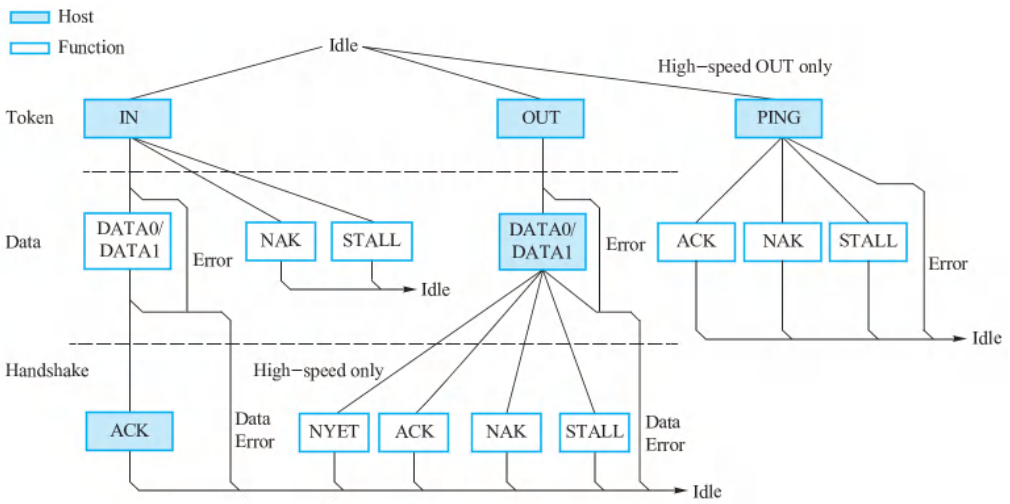


图 15.23　批量传输不同阶段数据包构成

对于批量传输的最后一个 IN 事务：

如果 USB 主机收到的数据长度小于端点支持的最大包长度，那么 USB 主机认为数据已经接收完成。

如果 USB 主机收到的数据长度等于端点支持的最大包长度，需要额外的 0 数据的包告诉 USB 主机数据已经接收完成。

批量输出流程 OUT

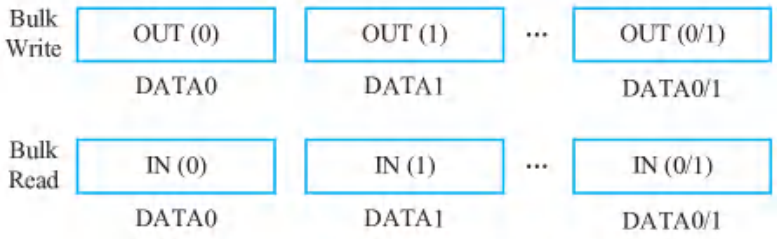


图 15.24　批量传输数据传输简要框图

令牌阶段

- 主机发送 BULK 令牌包，令牌包中包含设备地址、端点号和数据方向。

数据段

- 设备如果接收令牌包出错，无响应，让主机等待超时；
- 设备端点不存在，设备回复 STALL；
- 设备端点数据未准备好，设备回复 NAK；
- 设备端点数据准备好，设备回复数据包。

握手端设备

- 数据包正确，并有足够的空间保存数据：设备返回 ACK 握手包或 NYET 握手包（只有高速模式才有 NYET 握手包，它表示本次数据接收成功，但是没有能力接收下一次传输）。
- 数据包正确，但是没有足够的空间保存数据：设备返回 NAK 握手包。主机收到 NAK，延时一段时间后，再重新进行批量输出事务。
- 数据包正确，但端点处于挂起状态：设备返回一个 STALL 握手包。
- 数据包错误：设备不返回任何握手包，让主机等待超时。
- CRC 错误或位填充错误：设备不返回任何握手包，让主机等待超时，

S.1.4 USB 标准请求

USB 标准请求是一个大小为 8 个字节数据结构，USB 标准请求用于：一是对设备进行枚举，二是对设备的状态进行更改。USB 的标准请求的数据传输方式都是控制传输方式，所以使用的端点是设备的默认端点 0。

后续的程序设计环节我们将只从设备的角度来对主机发送的 USB 标准请求进行处理，为了了解 USB 设备如何响应 USB 主机发送的 USB 标准请求，需要了解请求的数据结构，以便知道 USB 主机发送的是哪一种 USB 标准请求，需要了解每一个标准请求设备需要作何响应。

为了便于理解我们将 USB 标准请求又称作 SETUP 数据结构，在后续的程序设计中，设备对 USB 主机的标准请求的响应将会单独使用一个.c 文件进行编写。后续 SETUP 数据结构将频繁使用希，望读者看到此概念知道说的是 USB 标准请求。

注意：除了 USB 标准请求外，USB 还有特定类的请求，在后续的程序设计中会介绍其中两种特定类请求，分别为 HD 特定类请求、BOT 特定类请求。

1. SETUP 数据结构

SETUP 数据结构由 5 个字段构成，其分别为：

- 1 字节的 bmRequestType
- 1 字节的 bReqest
- 2 字节的 wValue
- 2 字节的 wInde
- 2 字节的 wLength

以上所有字段合起来共 8 字节。SETUP 数据结构可以描述为下表 15.6

表 15.6 SETUP 数据结构

偏移量	字段	大小/字节	取值	含义
0	bmRequestType	1		请求特性 D7：数据传输方向 0=从主机到设备 1=从设备到主机 D6-D5：请求的类型 0=标准类型 1=类类型 2=厂商类型 3=保留 D4-0：请求的接收者 0=设备 1=接口 2=端点 3-其他 其余：保留
1	bRequest	1	数值	请求代码
2	wValue	2	数值	该域的意义由具体请求决定

偏移量	字段	大小/字节	取值	含义
4	wIndex	2	索引或偏移量	该域的意义由具体请求决定
6	wLength	2	字节数	数据过程所需要传输的字节数

2. USB 标准请求分类

请求的类型由 SETUP 数据结构的 bRequest 字段指定，bRequest 字段值指定的请求类型和大致功能可以由下表 15.7 进行描述：

表 15.7 标准请求及其功能

请求	bRequest 字段值	功能
ClearFeature	1	清除设备、接口的某种特征（或性能）
GetConfigration	8	获取指定设备当前的配置值
GetDescriptor	6	获取设备的某种标准描述符
GetInterface	10	获取设备接口当前工作的选择设置值
GetStatus	0	获取设备、接口或端点的某种状态
SetAddress	5	为设备设置唯一的地址
SetConfiguration	9	激活设备的某个配置
SetDescriptor	7	主机会更新或创立描述符
SetFeature	3	主机启用一个在设备、接口或端点上的特征
SetInterface	11	主机激活设备的某个接口的设置值
SynchFrame	12	在实时传输中，用于同步某个帧开始传输序列

3. USB 标准请求的功能

（1）ClearFeature

ClearFeature 请求用于清除或禁用 USB 设备，接口或端点的某些特性，该请求无数据阶段其每个字段具体如表 15.8。

表 15.8

bmRequest’T’ype	bRequest	wValue	wIndex	wLength	数据过程
0x00 0x01 0x02	CLEAR_FEATURE	特性选择	0 接口号 端点号	0	无

- bmRequestType 值为 00，表示从主机到设备，设备接收
- bmRequestType 值为 01，表示从主机到设备，接口接收
- bmRequestType 值为 02，表示从主机到设备，端点接收

表 15.9 为特性选择字段的详细内容。

表 15.9

wValue	接收者	特性名	功能
0	端点	ENDPOINT_HALT	挂起端点
1	USB 设备	DEVICE REMOVE_WAKEUP	远程唤醒设备
2	USB 设备	TEST_MODE	用于 USB 测试

（2）GetConfiguration

GetConfiguration 用于主机读取 USB 设备当前的配置值，在 GetConfiguration 的数据阶段，USB 设备将向主机返回一个字节的配置值。

表 15.10

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
0x80	GET_CONFIGURATION	0	0	1	配置值

wLength 字段值指明 USB 设备返回的配置值的大小

如果设备已经配置好，设备接收该请求会发送则发送 PACKET1 数据包，如果未被配置，则发送 PACKET0 数据包：

（3）GetDescriptor

GetDescriptor 用于 USB 主机读取设备的描述符，在请求数据阶段，USB 设备将向主机返回指定的描述符。GetDescriptor 请求是 USB 通信中最常使用的请求，有关此请求的内容十分重要，可以说 STEUP 阶段的主要数据负载就在此请求的响应中。

表 15.11

bmRequestType	bReques	wValue	wIndex	wLength	数据过程
0x80	GET_DESCRIPTOR	描述符类型和索引	0 或者语言 ID	描述符长度	描述符

其中 wValue 字段的值指明需要发送的描述符类型，其具体指明的设备类型如表 15.12。

表 15.12

wValue	值	描述符类型
DESC_DEVICE	0x0100	设备描述符
DESC_CONFIGURATION	0x0200	配置描述符
DESC_STRING	0x0300 0x0301 0x0302	LANGIDDESC MANUFACTDESC MANUFACTDESC
DESC_HIDREPORT	0x2200	报告描述符

注意：

1.wValue 的第一字节（字符）表示同一中描述类型（比如字符串描述符）中具体的某个描述符（如厂商或者产品字符），第二字节表示描述类型的编号。

2.对于全速和低速模式，获取描述符的标准请求只有三种：获取设备、配置、字符串的描述符，另外的接口和端点描述符是跟随配置描述符一并返回的，不能单独请求返回。

（4）GetInterface

GetInterface 请求用于 USB 主机读取指定接口的设置值，即获取接口描述符中 bAlter-nateSetting 字段中的值。在 GetInterface 请求的数据阶段，USB 设备向 USB 主机返回 1 个字节的可替换设置值。在程序设计中若主机发出该请求，我们仅需使用 PACKET0 对请求进行回复

表 15.13

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
0x80	GET_INTERFACE	0	0	1	备用接口号

（5）GetStatus

CelStatus 请求主要用于 USB 主机读取 USB 设备，接口或端点的状态。USB 设备返回 2 字节的设备状态。其每个字段具体的值如表 15.14 所示。

表 15.14

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
0x80 0x81 0x82	GET_STATUS	0	0 接口号 端点号	2	设备、接口或者端点状态

其中，wLength=2 表示 USB 设备返回的数据为 2 字节。

bmRequestType 共三个值，其代表的功能如表 15.15 所示

表 15.15

bmRequestType	值	功能
DEVICE_RECIPIENT	0x80	获取设备状态
INTERFACE_RECIPIENT	0x81	获取接口状态
ENDPOINT_RECIPIENT	0x82	获取端点状态

1）若为 DEVICE_RECIPIENT，设备将返回，如表 15.16 所示。

表 15.16

偏移	D15-D2	D1	D0
含义	保留为 0	远程唤醒	自供电

2）若为 INTERFACE_RECIPIENT，设备将返回数据 0。

3）若为 ENDPOINT_RECIPIENT，设备将返回，如表 15.17 所示。

表 15.17

偏移	D15-D1	D0
含义	保留为 0	端点是否已停止（1 停止，0 未停止）

（6）SetAddress

SetAddress 用于枚举（enumeration）阶段为设备分配一个唯一的地址，地址在 wValue 字段中且最大值为 127。该请求特别的地方在于，直到状态阶段完成，设备才完成地址设置。其他所有请求必须在

状态阶段之前完成。该请求同样无数据阶段，如表 15.18 所示。

表 15.18

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
0x00	SET_ADDRESS	设备地址	0	0	无

wValue 为主机给设备分配的唯一地址，设备若有 USB 控制寄存器，如 Ai8051U 有一个寄存器 FADDR，该寄存器将会存放主机为设备分配的唯一地址。

（7）SetConfiguration

SetConfiguralion 和 SetAddress 请求很类似，区别是 wValue 字段的意义：SetAddress 中,wValue 的第一字节（低字节）表示设备的地址；SetConfiguration 则为配置的值。该值与配置描述符的配置编号一致时，表示选中该配置，通常为 1，因为大多数 USB 设备只有一种设置；若为 0，则设备进人地址设置状态，如表 15.19 所示。

表 15.19

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
0x00	SET_CONFIGURATION	配置值（0x0001）	0	0	无

一般的设备只有一个配置，当有多个配置时，会让用户选择。一个设备只能工作在一个配置状态下。

（8）SetDescriptor（*）

用于修改选中设备需要修改的描述符，或者增加新的描述符，在 SetDescriptor 请求的数据阶段，主机将向 USB 设备发送指定的描述符类型，如表 15.20 所示。

表 15.20

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
0x00	07	类型和索引	00 或语言 ID	0	无

（9）SetFeature

SetFeature 请求用于设置或使能 USB 设备、接口或端点的特性值，如表 15.21 所示。

表 15.21

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
0x00 0x01 0x02	SET_FEATURE	特性选择	0 接口号 端点号	0	无

本次设计仅设置 USB 设备对应的端点产生 stall 信号进行回应。

（10）SetInterface

SetInterface 请求用于 USB 主机为设备指定的接口选择一个合适的替换值，如表 15.22 所示。

表 15.22

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
0x01	SET_INTERFACE	可替换的接口值	接口号	1	无

SetInterface 请求只在 USB 处于配置状态时有效。

当 USB 设备的一个接口存在 1 个或多个可替换设置时，SetInterface 请求使得主机可以为其选择所

需要的可替换值。

（11）SynchFrame（*）

SynchFrame 用于设置并报告端点的同步帧号，用于同步传输，只适用于同步端点。在 SynchFrame 请求的数据阶段，USB 设备将向 USB 主机返回 2 个字节的帧号数据，如表 15.23 所示。

表 15.23

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
0x82	SYNCH_FRAME	0	端点号	2	帧号

SynchFrame 请求只在 USB 设备处于配置状态时有效，

S.1.5　USB 通信过程

1. USB 枚举和配置

USB 设备首次连接时会进行该过程，USB 物理连接后最初的通信过程。开发者一般将枚举视为 USB 设备中的单一程序。实际上，枚举是三个阶段的其中一个：动态检测，枚举、配置。动态检测是指识别 USB 端口状态的变化。在某个设备被插入时，根据该设备的速度，相应线将被上拉。主机/集线器通过电压变换来检测总线端口状态的变化。枚举阶段紧接着设备检测阶段。枚举指的是给新插入设备分配唯一地址的过程。配置阶段是指通过交换设备请求来确定设备性能的过程。主机用来了解设备信息的请求是标准请求。所有 USB 设备都要支持这类请求。

以下为枚举的全部过程。

1）动态检测

第一步：设备连接到 USB 端口上，并被主机检测。此时，设备可从总线吸收 100mA 的电流，并处于被供电状态。

第二步：集线器通过监控端口的电压来检测设备。集线器的 D+线和 D-线上带有下拉电阻。如上所述，根据设备的速度，D+或 D-线上会带有上拉电阻。通过监控这些线上的电压变换，集线器检测设备是否得到连接。

2）枚举

第三步：主机使用中断端点获得集线器状态（包括端口状态的变化），从而了解新连接的设备。主机从集线器获得设备检测情况后，它会向集线器发送一个请求，以便询问在 GET_PORT_STATUS 请求有效时所发生状态变化的详细信息。

第四步：主机收集该信息后，它通过“USB 速度”一节中所介绍的方法来检测设备的速度。最初，通过确定上拉电阻位于 D+线还是 D-线，集线器可以检测设备速度是全速还是低速。通过另一个 GET_PORT_STATUS 请求，该信息被报告给主机。

第五步：主机向集线器发送 SET_PORT_FEATURE 请求，要求它复位新连接的设备。通过将 D+和 D-线下拉至 GND（0V），使设备进入复位状态。D+和 D-线处于低电平状态的持续时间为 2.5us，因此发生复位条件。集线器在 10ms 内维持复位状态。

第六步：复位期间发生一系列 J 状态和 K 状态，这样是为了确定设备是否支持高速传输。如果设备支持高速，它会发出一个单一的 K 状态。高速集线器检测该 K 状态并用 J 和 K 顺序（组成“KJKJKJ”格式）来回应。设备检测到该格式后，它会移除 D+线上的上拉电阻。低速设备和全速设备则会忽略这一步。

第七步：通过发送 GET_PORT_STATUS 请求，主机检查设备是否仍处于复位状态。如果设备仍处于复位状态，则主机会继续发送请求，直到它获取设备信息退出复位状态为止。设备退出复位状态后，它便

进人默认状态，如 USB 电源一节所述。现在，设备可以回应主机的请求，具体是对其默认地址 00h 进行控制传输。所有 USB 设备的起始地址均等于该默认地址。每次只能有一个 USB 设备使用该地址。因此，同时将多个 USB 设备连接到同一个端口时，它们会轮流进行枚举，而不是同时枚举。

第八步：主机开始了解有关设备的更详细的信息。首先，它要知道默认管道（端点 0）的最大数据包大小。主机首先向设备发送 GET_DESCRIPTOR 请求。设备发给主机相应应用笔记 USB 描述符一节所介绍的描述符。在设备描述符中，第八个字节（bMaxPacketSize0）包含了有关端点 0（EP0）最大数据包尺寸的信息。Windows 主机要求 64 字节，但仅在收到 8 字节设备描述符后它才转换到控制传输的状态阶段，并要求集线器复位设备。USB 规范要求，如果设备的默认地址为 00h，当它得到请求时，设备至少要返回 8 字节设备描述符。要求 64 字节是为了防止设备发生不确定行为。此外，仅在收到 8 字节后才进行复位的操作是早期 USB 设备遗留的特性。在早期 USB 设备中，当发送第二个请求来询问设备描述符时，某些设备没有正确回应。为了解决该问题，在第一个设备描述符请求后需要进行一次复位。被传输的 8 字节包含 bMaxPacketSize0 的足够信息。

第九步：主机通过 SET_ADDRESS 请求为设备分配地址。在使用新分配地址前，设备使用默认地址 00h 完成所请求的状态阶段。在该阶段后进行的所有通信均会使用新地址。如果断开与设备的连接、端口被复位或者 PC 重启，该地址可能被更改。现在，设备处于地址状态。

3）配置

第十步：设备退出复位状态后，主机会发送 GET_DESCRIPTOR 命令，以便使用新分配地址读取设备的描述符。不过，此次所有描述符均被读取。主机通过该信息了解设备及其性能。该信息包含外设接口数量、电源连接方法以及所需要的最大电源。主机先请求设备描述符，而这一次它将收到全部描述符，而不仅是描述符的一部分。然后，主机将发送另一个 GET_DESCRIPTOR 命令，以便请求设备发送配置描述符。该请求的结果不仅是主机获取了设备的配置描述符，并且还获取了与配置描述符相关联的所有描述符，如接口描述符和端点描述符。Windows PC 首先只请求设备发送配置描述符（9 个字节），然后它会发送第二个 GET_DESCRIPTOR 请求，以请求设备发送配置描述符以及与配置描述符相关联的所有播述符（如接口和端点描述符）

第十一步：为了让主机 PC（此情况是 Windows PC）成功使用设备，主机必须加载设备驱动程序。主机会搜索一个用于管理它与设备通信的驱动程序。Windows 使用它的.inf 文件寻找与设备产品 ID 和供应商 ID 匹配的驱动程序。也可以选择性寻找与设备发布版本号匹配的程序。如果 Windows 未能找到匹配的驱动程序，它会寻找与设备的类别、子类以及协议相匹配的程序。如果设备先前已经进行了枚举，Windows 会使用设备所注册的信息来寻找合适的驱动程序。确定好驱动程序后，主机可能请求设备的特定描述符或者请求设备重新发送描述符。

第十二步：收到所有描述符后，主机使用 SET_CONFIGURATION 请求进行特殊的设备配置。大部分设备只有唯一种配置。对于支持多项配置的设备，用户或驱动程序可选择合适的配置。

第十三步：此时设备将处于配置状态。设备将按照描述符所定义的功能进行操作。所定义的最大电源是从 Vbus 吸取的。

现在就可以在应用中使用设备。（在 HID 键盘或鼠标中对应的是中断传输，在 CDC 中对应的也是中断传输）

S.1.6 USB 描述符

本节仅对 USB 描述符的数据结构进行说明，数据结构中详细字段的值所代表的含义还请读者阅读 USB 协议文档中描述符部分的内容，本次实验的描述符文件通过打开工程目录下的 usb_desc.c 文件进行查阅。

1. 设备描述符：

设备描述符是 USB 设备的第一个描述符，每个 USB 设备都得具有设备描述符，且只能拥有一个。设备描述符中包含了 USB 规范、设备配置编号、设备支持的协议等信息。设备描述符的数据结构可以描述如表 15.24 所示。

表 15.24 设备描述符表数据结构

偏移	字段	大小（字节）	说明
0	bLength	1	设备描述符的大小，共 18 个字节，该字段固定为 0x12
1	bDescriptorType	1	描述符类型=设备（01h）
2	bcdUSB	1	USB 规范版本（BCD）
4	bDeviceClass	1	设备类别
5	bDeviceSubClass	1	设备子类别
6	bDeviceProtocol	1	设备协议
7	bMaxPacketSize0	1	端点 0 的最大数据包大小
8	idVendor	1	供应商 ID（VID，由 USB-IF 分配）
10	idProduct	2	产品 ID（PID，由制造商分配）
12	bcdDevice	2	设备释放编号（BCD）
14	iManufacturer	1	制造商字符串索引
15	iProduct	1	产品字符串索引
16	iSerial Number	1	序列号字符串索引
17	bNumConfigurations	1	受支持的配置数量

bLength 是设备描述符的总长度，以字节为单位。

bedUSB 则显小了设备支持的 USB 版本，通常是最新版本。这是一个二进制代码形式的十进制数据，采用 OxAABC 的形式，其中 A 是主版本号，B 是次版本号，C 是子次版本号。例如，USB2.0 设备拥有 OX0200 值，USB1.1 设备拥有 0x0110 值。通常，主机将使用 bcdUSB 以确定需要加载的 USB 驱动器

bDeviceClass、bDeviceSubCass 和 bDeviceProtocol 均由操作系统使用，以便在枚举过程中识别 USB 设备的驱动器。将代码填充设备措述符中的这些字段内可以防止各种不同的接口独立运行，如一个复合设备。大部分 USB 设备都在接口描述符中定义了它的类别，并将这些字段保持为 00h。

bMaxPacketSize 会报告由端点 0 支持的数据包的最大字节数量。根据设备，数据包的大小可以为 8 个字节、16 个字节、32 个字节和 64 个字节

iManufacturer、iProduct 和 iSerialNumber 都是字符串描述符索引。字符串描述符包括有关制造商、产品和序列号等信息。如果存在字符串描述符，这些变量应该指向其索引位置。如果无字符串，那么应该将零值填充到各个字段内。

bNumConfgurations 定义了设备可支持的配置总数。多个配置使设备能够根据特定条件按照特定条件进行不同的配置，如由总线供电或自供电。更多有关该问题的信息将在后面详细介绍。

2. 配置描述符：

接口数量、设备供电方法。注意：在实际获取配置描述符时，设备会将配置描述符、接口描述符、端点描述符、HID 描述符打包在一起发送。配置描述符的数据结构如表 15.25 所示。

表 15.25 配置描述符表数据结构

偏移	字段	大小（字节）	说明
0	bLength	1	该描述符的长度=9 个字节
1	bDescriptorType	1	描述符类型=配置（02h）
2	wTotalLength	2	总长度包括接口和端点描述符在内
4	bNumInterfaces	1	本配置中接口的数量
5	bConfigurationValue	1	SET_CONFIGURATION 请求所使用的配置值，用于选择该配置
6	Configuration	1	描述该配置的字符串索引
7	bmAttributes	1	位 7：预留（设置为 1）； 位 6：自供电；位 5：远程唤醒
8	bMaxPower	1	本配置所需的最大功耗（单位为 2mA）

- wTotalLength 是本配置的整个层次的长度。该值报告了本配置的字节总数以及一个配置所需的接口和端点描述符。
- bNumInterfaces 则定义了在该指定配置中接口总数。最小为 1 个接口。
- bConfigurationValue 将定义某一值作为参数，SET CONFIGURATION 请求会使用该参数来选择该配置。
- bmAttributes 定义了 USB 设备的参数。如果设备由总线供电，那么位 6 将被设置为 0，如果设备自供电，那么位 6 将被设置为 1。如果 USB 设备支持远程唤醒，则位 5 将被设置为 1。如果不支持远程唤醒，则位 5 将被设置为 0。
- bMaxPower 定义了设备全速运行时通过总线消耗的最大功耗，以 2mA 为单位。如果拔出自供电设备的外部电源，那么它的功耗不会超过该字段中所显示的值。

3. 接口描述符：

一个接口描述符介绍了包含在配置中的特定接口，此处结合程序设计进行简要介绍，在 HID 鼠标设备中，用于标识接口类别的 bIntefaceClass 字段值为 3，用于标识接口子类别的 bInterfaceSubClass 字段值为 1，用于识别 HID 鼠标设备的 bIntefaceProtocol 字段的值为 2。接口描述符的数据结构如表 15.26 所示。

表 15.26 接口描述符表数据结构

偏移	字段	大小（字节）	说明
0	bLength	1	该描述符的长度=9 个字节
1	bDescriptorType	1	描述符类型=接口（04h）
2	bInterfaceNumber	1	该接口基于零的索引
3	bAlternateSetting	1	备用设置值
4	bNumEndpoints	1	该接口所使用的端点数量（不包含 EP0）
5	bInterfaceClass	1	接口类别
6	bInterfaceSubclass	1	接口子类别

偏移	字段	大小（字节）	说明
7	bInterfaceProtocol	1	接口协议
8	iInterface	1	该接口字符串描述符索引

4. 端点描述符：

由于主机通过端点与地址的组合与设备通信，因此该描述符存放了主机所需要的设备端点的信息。端点述符的数据结构如表 15.27 所示。

表 15.27 端点描述符数据结构

偏移	字段	大小（字节）	说明
0	bLength	1	该描述符长度=7 个字节
1	bDescriptorType	1	描述符类型=端点（05h）
2	bEndpointAddress	1	位 3……0：端点数量 位 6……4：预留，复位为零 位 7：端点的方向。控制端点可以忽略该位。（0=OUT 端点，1=IN 端点）
3	bmAttributes	1	位 1……0：传输类型（00=控制，01-同步，10=批量，11=中断） 如果该端点不是同步端点，那么位 5 到位 2 将被预留，必须将这些位设置为零。如果该端点是同步的,这些位将按如下内容定义:位 3…2：同步类型（00=无同步，01=异步：10=自适应）位 5……4：用途类型（00=数据端点，01=反馈端点，10=隐式反馈数据端点数值，11 表示保留）
4	wMaxPacketSize	2	该端点的数据包最大尺寸
6	bInterval	1	中断端点的轮询间隔，单位为 ms（对于同步端点：间隔为 1mg；控制或批量端点可能忽略该字段）

5. 字符串描述符：

该描述符为主机提供了设备名称、生产厂家、序列号或不同接口等信息。字符串描述符的数据结构如表 15.28 所示。

表 15.28 字符串描还符数据结构

偏移	字段	大小（字节）	说明
0	bLength	1	该描述符的长度=7 个字节
1	bDescriptorType	1	描述符类型=STRING（03h）
2…n	bStringlangID	变化	Unicode 编码字符串或 LANGID 代码

6. 使用多个 USB 描述符

每个 USB 设备只有一个设备描述符。但是，一个设备可以有多种配置、接口、端点和字符串描述符。设备执行枚举时，终端阶段中有一步是读取设备描述符，并选择需要使能的设备配置类型。每一次操作只能使能一种配置。例如，某个设计中存在两种配置：一种适用于自供电的设备，另一种适用于由总线供电的设备。这时，用于自供电设备的 USB 的总体性能会与使用于总线供电设备的不一样。拥有多种配置和多种配置描述符可允许设备选择性实现该功能。

同时一个设备可以有多种接口，因此，它也会有多种接口描述符。具有多种接口的 USB 设备（能够执行不同功能）被称为复合设备。USB 头戴式音频耳机便是一个复合设备示例。这种音频耳机包括一个带有两个接口的 USB 设备。其中，一个接口用于音频传输，另一个接口可用于音量调整。可以同时使能多个接口。图 15.25 显示的是单个 USB 设备中如何分配两种接口。

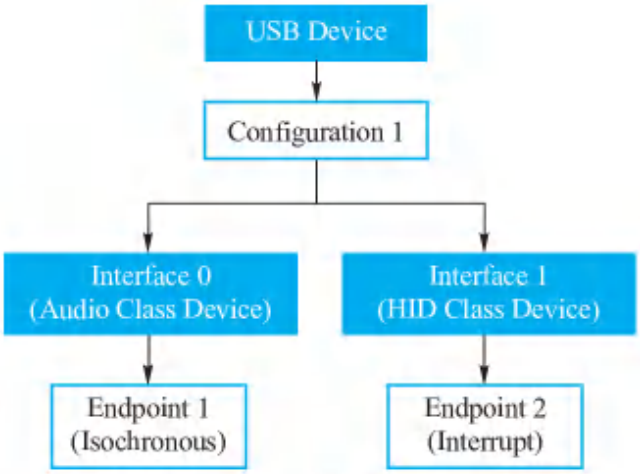


图 15.25　多个接口的设置框图

最终，每个接口可以有多种配置。这些配置类型被称为备用设置。可以使用这些备用设置来更改设备的端点配置，从而保留带宽的不同能力。例如，在某种备用设置中，可以将设备的端点配置为批量传输（没有保证的总线带宽），在另一种备用设置中，可以将设备的端点配置为同步传输（有保证的总线带宽）。图 15.26 详细演示了该示例。

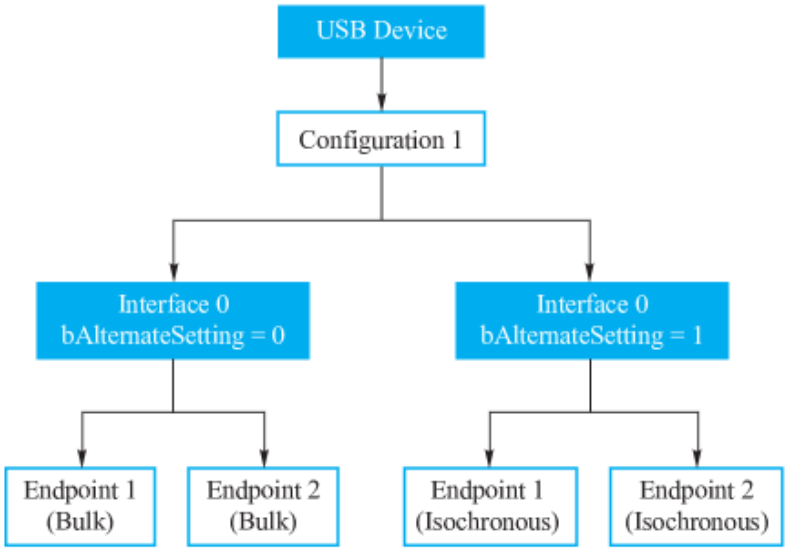


图 15.26　多接口设置框图

图 15.27 显示的是各种配置选项的总体框图，有助于根据不同的配置选项来创建准确的可配置 USB 设备。

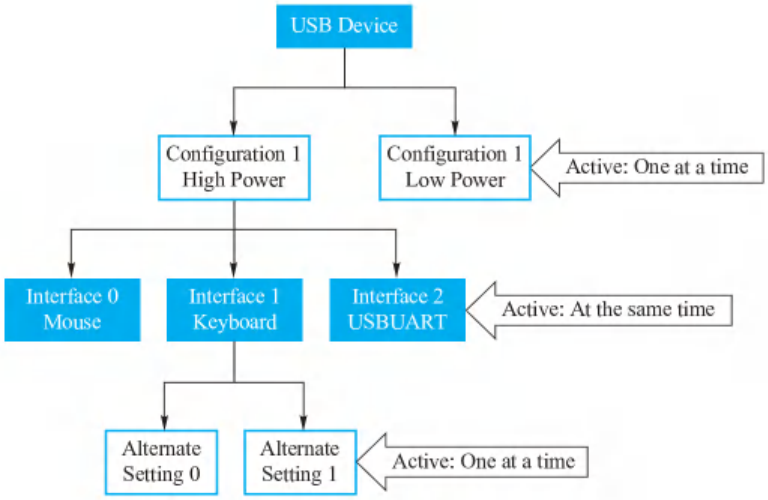


图 15.27　配置框图

S.2　USB2.0 程序设计实现

由于 USB 接口支持的设备众多，一个 USB 设备的程序会涉及多种协议的描述，因此程序设计需要分层编写。这样做的好处是，对于诸如 USB2.0 这样设备都需要使用的协议的源文件仅需较小的修改即可使用，并且可以依据不同的设备将其他不同的协议规范的源文件添加进入工程以满足需要。

本文档介绍如何编程实现 USB2.0 规范的两层，一是 USB 层，二是 USB 标准请求层，本次程序设计主干框如图 15.28 所示。

图 15.28　程序的主干结构

15.2.1 的第 1 小节介绍如何控制 USB 控制寄存器，对应的函数为 usb_read_reg()和 usb_write_reg()。

15.2.1 的第 2 小节介绍如何通过 FIFO 寄存器来实现主机和设备之间的数据收发，对应的函数为 usb_read_fifo()和 usb_write_fifo()。

15.2.1 的第 3 小节介绍 USB 控制传输类型的实现方法，对应的函数为 usb_ctrl_in()和 usb_ctrl_out()。

15.2.2 的第 1 小节介绍的 USB 初始化的重要工作是打开 USB 有关的中断，对应的函数为 usb_init()。

15.2.2 的第 2 小节介绍讲述如果触发了中断，设备如何依据不同的中断类型进行相应的操作，1.5 小结对应的函数为 usb_isr()，对应的中断号为 25。

15.2.2 的第 3 小节介绍 USB 通信中最开始的 SETUP 过程如何实现，对应的函数为 usb_ setup()

15.2.2 的第 4 和第 5 小节分别介绍 USB 标准请求和特定类请求的实现方法，对应的函数为 usb_req_std()和 usb_req_class()。

控制传输的主要流程为：主机向设备不断发送 SETUP 数据包，设备接收 SETUP 数据包解析并依据解析的内容进行响应，响应内容可以是设备软件状态的设置，或者对主机回复特定的信息。该过程结束后，若设备为 HID 设备，则可能进入中断传输阶段，该阶段为设备的主要工作状态。

S.2.1　USB 的寄存器操作和数据传输

1. USB 控制寄存器操作方式

1）. 读取 USB 控制寄存器方式：

先判断 USBADR 寄存器的最高位是否为 1，启动 USB 控制寄存器的读操作，地址由 USBADR 设定。此时将需要操作的 USB 控制寄存器的地址，通过按位或上 0x80 的方式传给 USBADR 寄存器，然后返回 USBDAT 寄存器的内容即可，USBDAT 的内容即为需要读取的 USB 控制寄存器的内容，其具体的写法代码如代码清单 15-1 所示。

代码清单 15-1　读取 USB 控制器的 C 代码描述

```
/**********************************读取 USB 控制寄存器的内容*******************************/
```

```
BYTE usb_read_reg (BYTE addr)
{
    BYTE dat;
    while (USBADR & 0x80);                          //按位与，直到读取忙时跳出循环
    USBADR = addr | 0x80;                           //将传人的 USB 控制寄存器的地址赋给
USBADR 寄存器
    while (USBADR & 0x80);                          //按位与，直到读取忙时跳出循环
    dat = USBDAT;                                           //读取 USB 控制寄存器内容
    return dat;
}
```

2）.写 USB 控制寄存器方式：

先判断 USBADR 寄存器的最高位是否为 1，若为 1 表示 USBDAT 寄存器中的数据无效，USB 正在读取间接寄存器。此时将需要操作的 USB 控制寄存器的地址，通过按位与上 0x7f 的方式赋值给 USBADR 寄存器，然后将传入的 USB 控制寄存器的值赋给 USBDAT 寄存器，即完成对 USB 控制寄存器的写操作，其具体的写法代码如代码清单 15-2 所示。

代码清单 15-2　写 USB 控制寄存器的 C 语言代码

```
/*********************************写 USB 控制寄存器*********************************/
void usb_write_reg(BYTE addr, BYTE dat)
{
    while (USBADR & 0x80);              //按位与，直到读取忙时跳出循环
    USBADR = addr & 0x7f;               //最高位清零 USBDAT 寄存器中的数据有效
    USBDAT = dat;                                   //将传入的数值赋给 USBDAT 寄存器
}
```

2.FIFO 的操作方式

FIFO 寄存器是 USB 数据传输时的核心寄存器。在 USB 通信中，主机向设备发送的数据，是通过 FIFO 寄存器以字节为单位读入的；而通过设备向主机发送的数据，同样也是通过 FIFO 寄存器以字节为单位发送的。

1）.读取 FIFO 寄存器方式(接收主机的数据：传输类型 OUT)：

通过读取 COUNT0 寄存器以获取端点 0 最后接收到的 OUT 数据包的长度，依据数据包的长度以字节为单位读入传入的指针所指向的地址间，即完成一次数据的接收。其具体的写法代码如代码清单 15-3 所示。

代码清单 15-3　读取 FIFO 的 C 语言代码

```
/*********************************读取 FIFO*********************************/
BYTE usb_read_fifo(BYTE fifo, BYTE *pdat)
{
    BYTE cnt;
    BYTE ret;
```

```
    ret = cnt = usb_read_reg(COUNT0);                    //获取端点 0 最后接收到的 OUT 数据包
    长度
    while (cnt--)
    {
        *pdat++ = usb_read_reg(fifo);
    }
    return ret;                                                    //返回端点 0 最后接收
到的数据包长度(单位:字节)
}
```

2）.写 FFO 寄存器方式(向主机发送数据：传输类型 IN)：

直接依据设备需要发送的数据大小，以字节为单位写入 FIFO 中，即完成一次数据的发送。其具体的写法代码如代码清单 15-4 所示。

代码清单 15-4　写 FIFO 的 C 语言代码

```
/*************************************写 FIFO*************************************/
void usb_write_fifo(BYTE fifo, BYTE *pdat, BYTE cnt)
{
    while(cnt--)
    {
        usb_write_reg(fifo, *pdat++);
    }
}
```

3.USB 控制传输类型实现

首先，控制传输仅使用端点 0 进行数据的传输，其依据传输方向分为 IN 类型和 OUT 类型。与前面所讲的 IN 类型与 OUT 类型相区别的是，其必须通过端点 0 传输数据，且该传输仅在标准请求以及特定类请求中使用。

1.控制 IN 类型：设备向主机发送数据

首先将 INDEX 寄存器指向端点 0，然后读取 CSR0 存器的内容，依据 CSR 寄存器的 IPRDY 是否为 1，来判定是否已经将 IN 数据包装入了 FIFO，若已经完成装入，则直接返回，否则控制需要向主机发送的数据以字节为单位装入 FIFO 中进行发送，由于端点 0 大小 EP0_SIZE 有限，如果发送的数据大小超过端点 0 大小，则需要分批次发送，其具体的写法代码如代码清单 15-5 所示。

代码清单 15-5　USB 控制传输 IN 类型的 C 语言代码

```
/*********************************USB 控制传输 IN 类型*********************************/
void usb_ctrl_in()
{
    BYTE csr;                          //创建 8 位 csr 变量
    BYTE cnt;                          //创建 8 位 cnt 变量
    usb_write_reg(INDEX,0);            //INDEX 寄存器指向端点 0
    csr = usb_read_reg(CSR0);          //csr 变量读取 CSR0 存器的内容
    if (csr & IPRDY) return;           //如果 IN 数据包已经装入 FIFO，则直接返回

    cnt = EpOstate.wSize > EP0_SIZE ? EP0_SIZE : EpOstate.wSize;
```

```
    usb_write_fifo(FIFO0,Ep0State.pData, cnt);
                                    //将结构体 Ep0State 中的 pDATA 数据写入 FIFO
    Ep0State.wSize -= cnt;          //wsize 减去 cnt 后的值
    Ep0State.pData += cnt;          //pdata 加上 cnt 后的值
    If (Ep0State.wSize == 0)        //如果数据包大小为 0
    {
        usb_write_reg(CSR0,IPRDY | IDATEND);
                                    //置 1IPRDY 和 DATEND 表示发送了一个 0 长的数据包(发送完毕)
        Ep0State.bState = EPSTATE_IDLE;
                                    //端点 0 的状态指向 IDLE 状态
    }
    else
    {
        usb_write_reg(CSR0,IPRDY);
         //如果数据包大小不为 0 则置 1IPRDY，表示要发送的数据包装入到端点 0 的 FIFO(未发完还要接着发)
     }
}
```

2.控制 OUT 类型：主机向设备发送数据，设备接收数据

首先将 INDEX 寄存器指向端点 0，然后读取 CSR0 寄存器的内容，依据 CSR 寄存器的 OPRDY 是否为 0 来判定 OUT 数据包是否已经收到，若收到则返回。否则调用读取 FIFO 寄存器的函数对数据进行读取，同理由于端点 0 大小 EP0_SIZE 有限，如果读取的数据大小超过端点 0 大小，则需要分批次读取，其具体的写法代码如代码清单 15-6 所示。

代码清单 15-6　USB 控制传输 OUT 类型的 C 语言代码

```
/*************************************USB 控制传输 OUT 类型*********************************/
void usb_ctrl_out()
{
    BYTE csr;                                   //创建 8 位 csr 变量
    BYTE cnt;                                   //创建 8 位 cnt 变量
    usb_write_reg(INDEX,0);                     //INDEX 寄存器指向端点 0
    csr = usb_read_reg(CSR0);                   //csr 变量读取 CSR0 存器的内容
    if (!(csr & OPRDY)) return;                 //如果 Out 数据包已经收到，则退出
    cnt = usb_read_fifo(FIFO0, Ep0State.pData); //cnt=FIFO 数据包的大小
    Ep0State.wSize -= cnt;                      //wsize 减去 cnt 后的值
    Ep0State.pData += cnt;                      //pdata 加上 cnt 后指向的地址
    If (Ep0State.wSize == 0)                    //如果全部读取完毕
     {
         usb_write_reg(CSR0, SOPRDY | DATEND);  //清零 OPRDY，表示接收完最后一个数据包
         Ep0State.bState = EPSTATE_IDLE;        //端点 0 的状态指向 IDLE 状态
     }
    else
     {
        usb_write_reg(CSR0, SOPRDY)
        //如果数据包大小不为 0 则清零 OPRDY，表示完成一个 OUT 数据包的接收(未读完还需接着读)
```

```
    }
}
```

S.2.2　　USB SETUP 阶段实现

1.USB 初始化的工作

USB 初始化最重要的工作是初始化 USB 设备状态，包括 USB 设备的硬件状态，还有软件状态，以下为 USB 初始化的内容：

1.硬件状态设置(寄存器)

1)FADDR 寄存器：使最后的 UADDR 地址生效，更新 USB 的功能地址

2)POWER 寄存器：强制产生异步 USB 复位，使能挂起检测

3)INTRIN1E：打开所有 IN 端点中断

4)INTROUT1E 寄存器：打开所有 OUT 端点中断

5)INTRUSBE 寄存器：允许 USB 复位信号中断，允许 USB 恢复信号中断，允许 USB 挂起信号中断

6)POWER 寄存器：使能挂起检测，当检测到总线上的挂起信号，USB 将进入挂起方式

2.软件状态设置

1)设备状态：默认

2)端点 0 状态：IDLE

3)IN 端点状态：0

3)OUT 端点状态：0

具体写法如代码清单 15-7 所示。

代码清单 15-7　USB 初始化 C 语言代码

```
/*****************************************USB 初始化*************************************/
void usb_init()
{
    USBCLK = 0X00;
    USBCON = 0x90;
    usb_write_reg(FADDR, 0x00);
    //使最后的 UADDR 地址生效，更新 USB 的功能地址，FADDR 低 7 位为保存 USB 的七位功能地址
    usb_write_reg(POWER, 0x09);                 //强制产生异步 USB 复位，使能挂起检测
    usb_write_reg(INTRIN1E, 0x3f);              //打开所有 IN 端点中断
    usb_write_reg(INTROUT1E, 0x3f);             //打开所有 OUT 端点中断
    usb_write_reg(INTRUSBE, 0x07);
    //允许 USB 复位信号中断，允许 USB 恢复信号中断，允许 USB 挂起信号中断
    usb_write_reg(POWER, 0x01);
    //使能挂起检测，当检测到总线上的挂起信号，USB 将进入挂起方式
    DeviceState = DEVSTATE_DEFAULT;             //设置设备起始状态默认
    EpOState.bState = EPSTATE_IDLE;             //设置端点 0 状态为 IDLE 状态
    InEpstate = 0x00;                           //设置 IN 端点为初始状态
    OutEpState = 0x00;                          //设置 OUT 端点为初始状态
```

```
    EUSB = 1;
}
```

2. USB 中断处理内容

会触发进入中断处理函数的中断：参照 USB 初始化一节中 USB 有关的中断已打开的那些中断。

中断处理的内容：

1)有关电源的中断

- 如果中断来自恢复信号，则跳转至 USB 恢复函数
- 如果中断来自复位信号，则跳转至 USB 复位函数
- 如果中断来自挂起信号，则跳转至 USB 挂起函数

2)有关 USB 端点的中断

(1)如果中断来自端点 0，即中断来自控制端点，则进人 USB 的 SETUP 阶段，SETUP 阶段是 USB 重要的通信阶段，其具体写法详见 USB SETUP 阶段的实现一节。

(2)如果中断来自其他端点，则进入对应的 USB 端点函数，USB 端点函数就是 USB 在 SETUP 阶段后的主要工作的内容，可以理解为 USB 设备的一般的使用功能就在对应的端点函数中(重点)，例如：键盘在工作时使用 IN1 端点向主机发送键盘码值的报告数据，而主机仅需将对应的报告数据翻译成对应的动作即可。

具体写法如代码清单 15-8 所示。

代码清单 15-8　USB 中断句柄/中断服务程序的 C 语言代码

```
/*************************************USB 中断句柄/中断服务程序*********************************/
void usb_isr() interrupt USB_VECTOR                 //USB 中断向量
{
    BYTE intrusb;                                   //usb 中断
    BYTE intrin;                                    //in 中断
    BYTE introut;                                   //out 中断
    BYTE adrTemp;                                   //临时保存地址变量
    adrTemp = USBADR;
    //USBADR 现场保存，避免主循环里写完 USBADR 后产生中断，在中断里修改了 USBADR 内容
    intrusb = usb_read_reg(INTRUSB);                //usb 中断=USB 电源中断标志
    intrin = usb_read_reg(INTRIN1);                 //USB 端点 IN 中断标志位
    introut = usb_read_reg(INTROUT1);               //USB 端点 OUT 中断标志位

    if (intrusb & RSUIF) usb_resume();              //若恢复信号中断有效，则 USB 恢复(空的不用看)
    if (intrusb & RSTIF) usb_reset();               //若复位信号中断有效，则 USB 复位

    if (intrin & EP0IF) usb_setup();                //若端点 0 的 IN/OUT 中断有效，USB 进人 setup 阶段

#ifdef EN_EP1IN
    If (intrin & EP1INIF) usb_in_ep1();            //若端点 1 的 IN 中断有效，则进入对应函数
#endif
```

```
                                        //后续不同 IN 端点的写法与端点 1IN 相同
#ifdef EN_EP1OUT
    if (introut & EP1OUTIF) usb_out_ep1();      //若端点 1 的 OUT 中断有效，则进入对应的函数
#endif

                                        //后续不同 OUT 端点的写法与端点 1OUT 相同
    If (intrusb & SUSIF) usb_suspend();
    USBADR = adrTemp;                   //USBADR 现场恢复
}
```

3.USB SETUP 阶段的实现

SETUP 阶段最重要的任务：一是依据主机向设备发送的不同请求，从而对设备进行对应的硬件状态设置，以及软件状态设置；二是主机向设备发送请求后，可能需要设备回复某些信息，这些信息以数据包的形式从设备发送给主机。

例如：一个完整的设备描述符信息会在 SETUP 阶段中，主机向设备发送 GET_DESCRIPTOR 请求后，由设备将设备描述符信息打包发送给主机。

SETUP 阶段的具体内容为：

1.先将端点定位到控制端点 0(因为在 USB 协议中，控制端点 0 就是服务于 SETUP 阶段中的，其执行着控制传输的任务)，之后我们读取 USB 端点 0 控制状态寄存器 CSR0 中的内容，依据 CSR0 中的 SDSTL 位来设置端点零的状态，SDSTL 的状态表示我们是否已经完成 STALI 信号的发送。依据 USB 协议内容：如果 STALL 信号发送完成，则我们可以设置端点 0 的状态为 EPSTATE_IDLE 状态，此状态表示我们可以接收来自主机的请求并进行请求处理。如果 SETUP 阶段结束，我们需要清零 CSR0 寄存器的 SSUEND 位。

2.请求分为三大类：

1）、USB 标准请求，对应的函数为 usb_req_std()

2）、特定类请求，对应的函数为 usb_req_class()

3）、厂商请求，对应的函数为 usb_req_vendor()。

其中厂商请求在此次实验中不会涉及。故如果主机发送该请求，我们默认将 USB 设备在 SETUP 阶段挂起。如果请求不是三大类中的任意一种，则我们默认将设备挂起。USB 标准请求以及特定类请求参照 USB 标准请求一节，以及类请求一节。

3.在 SETUP 阶段还可能存在单独的数据过程，该过程可能是 CTRL_IN 或者 CTRL_OUT 过程。

SETUP 具体的写法可以参照如下代码清单 15-9 所示。

代码清单 15-9　USB 的 SETUP 阶段的 C 语言代码

```
/*************************************USB 的 SETUP 阶段*********************************/
void usb_setup()
{
    BYTE csr;                           //csr 寄存器变量
    usb_write_reg(INDEX,0);             //选择端点 0
    csr = usb_read_reg(CSR0);           //读取 USB 端点 0 控制状态寄存器
    if (csr & STSTL)                    //如果 STALL 信号发送完成
    {
        usb_write_reg(CSR0, csr & SDSTL);       //未接收到错误条件，SDSTL 清零
```

```
        EpOState.bState = EPSTATE_IDLE;          //端点 0 设置 IDLE 状态
    }
    if (csr & SUEND)                             //如果 SETUP 阶段结束
    {
        usb_write_reg(CSR0, csr & ~SSUEND);      //清零 SSUEND
    }
    switch (EpOState.bState)                     //依据端点 0 状态进入状态机
    {
    case EPSTATE_IDLE:                           //IDLE 状态
        if (csr & OPRDY)                         //如果收到一个 OUT 数据包
        {
            usb_read_fifo(FIFO0, (BYTE * ) & Setup);     //从 FIFO0 中读取 Setup 信息
            Setup.wLength = reverse2(Setup.wLength);     //wlength 字节高 8 位和低 8 位交换
            switch (Setup.bmRequestType & REQUEST_MASK)
                                                 //依据 setup 中请求数据的类型进入状态机
        {
        case STANDARD_REQUEST:                   //标准请求(usb 请求核心)
            usb_req_std();
            break;
        case CLASS_REQUEST:                      //类请请求(hid 请求核心)
            usb_req_class();
            break;
        case VENDOR_REQUEST:                     //厂商请求
            usb_req_vendor();                    //实验不涉及厂商请求故不作内容编写
            break;
        default:                                 //默认
        usb_setup_stall();                       //建立阶段挂起
        return;
        }
    }
    break;
    case EPSTATE_DATAIN:                         //数据输入状态
        usb_ctrl_in();
        break;
    case EPSTATE_DATAOUT:                       //数据输出状态
        usb_ctrl_out();
        break;
    }
}
```

4.USB 标准请求

本文此处开始讲解 USB 标准请求层，该层对应 usb_req_std.c 源文件。此节强烈建议配合提供的例程进行阅读，这样不仅有利于理解协议内容，更可以理解如何将协议内容转化为计算机编程语言进行实现。

1)USB 标准设备请求数据结构

在协议中 USB 标准设备请求数据结构被称为 SETUP 数据结构，这也是为什么在编程时将其定义为一个 SETUP 结构体的原因。在程序设计中，可以知道 USB 标准的数据结构需要从 USB 控制寄存器中的 FIFO 中读取出来，USB 标准设备请求数据结构十分重要，我们可以通过读取到的 USB 标准设备请求数据结构进行解析，从而得到不同的信息，我们可以根据这些信息对不同的 USB 设备请求进行响应，表 15.29 给出类 USB 标准设备请求的数据结构。

表 15.29　SETUP 数据结构

偏移量	字段	大小 / 字节	取值	含义
0	bmRequestType	1		请求特性 D7：数据传输方向 0＝从主机到设备 1＝从设备到主机 D6-D5：请求的类型 0＝标准类型 1＝类类型 2＝厂商类型 3＝保留 D4-0：请求的接收者 0＝设备 1＝接口 2＝端点 3＝其他 其余：保留
1	bRequest	1	数值	请求代码
2	wValue	2	数值	该域的意义由具体请求决定
4	wIndex	2	索引或偏移量	该域的意义由具体请求决定
6	wLength	2	字节数	数据过程所需要传输的字节数

在程序设计时，我们可以通过定义如下 SETUP 结构体来定义一个 USB 标准设备请求数据结构，如代码清单 15-10 所示。

代码清单 15-10　USB 标准设备请求的数据结构

```
/*************************** USB 标准设备请求的数据结构***************************/
typedef struct                  //USB 标准设备请求的数据结构
{
    BYTE bmRequestType;         //请求类型(子级)
    BYTE bRequest;              //请求类型(父级)
    BYTE wValueL;               //值低 8 位
    BYTE wValueH;               //值高 8 位
    BYTE wIndexL;               //指针低 8 位
```

```
    BYTE wIndexH;                   //指针高 8 位
    WORD wLength;                   //长度
}SETUP;                             //(获取方式：从 FIFO 中读取)
```

2)USB 标准请求

在以上 SETUP 结构体中，USB 标准请求信息包含在 bRequest 字段中，通常来讲，所有 USB 设备都需要能够支持 USB 标准请求，对于标准请求的分类，不同标准请求对应的 bRequest 字段的值，还有不同标准请求功能,如表 15.30 所示。

表 15.30　标准请求及其功能

请求	bRequest 字段值	功能
ClearFeature	1	清除设备，接口的某种特征（或性能）
GetConfigration	8	获取指定设备当前的配置值
GetDeseriptor	6	获取设备的某种标准描述符
GetInterface	10	获取设备接口当前工作的选择设置值
GetStatus	0	获取设备、接口或端点的某种状态
SetAddress	5	为设备设置唯一的地址
SetConfiguration	9	激活设备的某个配置
SetDeseriptor	7	主机会更新或创立描述符
SetFeature	3	主机启用一个在设备、接口或端点上的特征
SetInterface	11	主机激活设备的某个接口的设置值
SynchFrame	12	在实时传输中，用于同步某个帧开始传输序列

对于标准请求，根据 bRequest 字段值，我们可以使用状态机来进入不同的请求，其写法具体如代码清单 15-11 所示。

代码清单 15-11　标准请求的定义

```
/*****************************USB 标准请求的定义*******************************/
#define GET_STATUS          0x00
#define CLEAR_FEATURE       0x01
#define SET_FEATURE         0x03
#define SET_ADDRESS         0x05
#define GET_DESCRIPTOR      0x06
#define SET_DESCRIPTOR      0x07
#define GET_COFIGURATION    0x08
#define GET_COFIGURATION    0x09
#define GET_INTERFACE       0x0A
#define SET_INTERFACE       0x0B
#define SYNCH_FRAME         0x0C
void usb_req_std()
{
    switch(Setup.bRequest)              //依据 SETUP 请求类型进入状态机
```

```
    {
    case GET_STATUS;
        usb_get_status();               //获取状态
        break;
    case CLEAR_FEATURE;                 //清除特征
        usb_clear_feature();
        break;
    case SET_FEATURE:                   //设置特征
        usb_set_feature();
        break;
    case SET_ADDRESS:                   //设置地址
        usb_set_address();
        break;
    case GET_DESCRIPTOR:                //获取描述符
        usb_get_descriptor();
        break;
    case SET_DESCRIPTOR;                //设置描述符
        usb_set_descriptor();
        break;
    case GET_COFIGURATION:              //获取配置值
        usb_get_configuration();
        break;
    case SET_CONFIGURATION:             //激活某个配置
        usb_set_configuration();
        break;
    case GET_INTERFACE:                 //获取接口
        usb_get_interface();
        break;
    case SET_INTERFACE:                 //设置接口
        usb_set_interface();
        break;
    case SYNCH_FRAME                    //帧同步
        usb_synch_frame();
        break;
    default:
        usb_setup_stall();             //建立阶段挂起
        return;
    }
}
```

表 15.31 将列出所有标准 USB 请求的结构和需要传输的数据

表 15.31　标准 USB 请求内容

bRequest 字段	wValue 字段	wIndex 字段	wLength 字段	数据过程
CLEAR_FEATURE	特性选择	0 接口号	0	没有

bRequest 字段	wValue 字段	wIndex 字段	wLength 字段	数据过程
		端点号		
GET_CONFIGURATION	0	0	1	配置值
GET_DESCRIPTOR	描述符类型和索引	0 或者语言 ID	描述符长度	描述符
GET_INTERFACE	0	接口号	1	备用接口号
GET_STATUS	0	0 接口号 端点号	2	设备、接口或者端点状态
SET_ADDRESS	设备地址	0	0	没有
SET_CONFIGURATION	配置值	0	0	没有
SET_DESCRIPTOR	描述符类型和索引	0 或者语言 ID	描述符的长度	描述符
SET_FEATURE	特性选择	0 接口号 端点号	0	没有
SET_INTERFACE	备用接口号	接口号	0	没有
SYNCH_FRAME	0	端点号	2	帧号

3)标准请求函数的参考写法

为了标准起见，本书的标准请求函数分为三个步骤：1.特殊情况处理；2.依据状态发包；3 结束状态，即进行数据操作(如发送数据，或者接收数据，或者不进行数据操作)。此部分的代码请参考工程 HID 协议范例文件 usb_req_std.c 内容。

(1)GET_STATUS

依据以表 15.32，可以对该请求的特殊情况进行设计

表 15.32 GET_STATUS 内容

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
0x80 0x81 0x82	GET_STATUS	0	0 接口号 端点号	2	设备、接口或者端点状态

第 1 步：特殊情况的处理，可以加入一判断语句，将不满足上表 GET_STATUS 的所有 USB 标准设备请求数据结构进行统一处理，其意义在于当所设计的程序错误进入该请求时，程序可以通过挂起的方式以终止该请求的处理。

第 2 步：依据 USB 协议，需要选择不同的数据包发送给主机，此处仅依据本次模拟 HID 实验，对其中出现的几个动作进行程序的编写。如表 15.33，根据 bmRequestType 进行不同的动作：

表 15.33 bmRequestType 表

bmRequestType	值	功能
DEVICE_RECIPIENT	0x80	获取设备状态

INTERFACE_RECIPIENT	0x81	获取接口状态
ENDPOINT_RECIPIENT	0x82	获取端点状态

如果 bmReguestType 不包含以上状态，默认状态都为挂起状态。

第 3 步：进入标准请求结束状态(进行数据过程)。例如，bmRequestType 为 DEVICE_RECIPIENT，这时在步骤 2 中软件已经依据 USB 协议内容将 PACKET0 数据包装入了数据缓冲区，然后软件再通过调用 usb_setup_in()，完成数据包的发送。

注意：usb_setup_in()的内容是：先将控制端点 0 设置为 DATAIN 状态，控制寄存器 CSR0 的 SPORDY 写 1 以清零 OPRDY，随后调用 usb_ctrl_in()进行数据发送，即使用控制 IN 类型向主机发送数据包。

(2)CLEAR_FEATURE

如表 15.34 所示，可以对该请求的特殊情况进行设计

表 15.34 CLEAR_FEATURE 内容

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
0x00 0x01 0x02	CLEAR_FEATURE	特性选择	0 接口号 端点号	0	无

第 1 步：特殊情况的处理，与 GET_STATUS 请求的设计思路一致，依旧是特殊情况处理。如果 USB 标准设备请求数据结构不满足表 15.34 所有条件，则挂起。

第 2 步：由于本次实验仅涉及到依据 wIndex 指向的端点号寄存器操作，故仅对端点号的情况进行处理；如果端点定义成 IN 端点，需要将 INDEX 寄存器根据端点号指向对应的端点，如端点号为 1，则 INDEX 寄存器需要指向端点 1，之后我们操作 INCSR1 寄存器的 CLRDT 位置 1(意思为数据切换位复位到 0)，最后将端点索引器 INDEX 指向端点 0，以便后续的控制操作。同理，端点定义成 OUT 端点，则需要操作的是 OUTCSR1 的 CLRDT 位置 1，使得数据切换位复位到 0。如果未设置端点的 IN 状态或 OUT 状态，则默认挂起。

第 3 步：进入请求结束状态：端点 0 设置成空闲状态，寄存器 CSR0 的 SPORDY 位以及 DATEND 位置位，表示处理完成从端点 0 接收到的数据包，且数据结束。

(3)SET_FEATURE

依据表 15.35，可以对该请求的特殊情况进行设计

表 15.35 SET FEATURE 内容

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
0x00 0x01 0x02	SET_FEATURE	特性选择	0 接口号 端点号	0	无

第 1 步：特殊情况的处理，与 GET_STATUS 请求的设计思路一致，依旧是特殊情况处理。如果 USB 标准设备请求数据结构不满足表 15.35 所有条件，则挂起。

第 2 步：由于本次实验仅涉及到依据 wIndex 指向的端点号寄存器操作，故仅对端点号的情况进行处理；如果端点定义成 IN 端点，需要将 INDEX 寄存器根据端点号指向对应的端点,如端点号为 1，则 INDEX 寄存器需要指向端点 1,之后我们操作 INCSR1 寄存器的 SDSTL 位置 1(意思为产生 STALL 信号作为对一个 IN 令牌的应答)，最后将端点索引器 INDEX 指向端点 0，以便后续的控制操作。同理，端点定义成 OUT 端点，则需要操作的是 OUTCSR1 的 SDSTL 位置 1，产生 STALL 信号作为对一个 OUT 令牌的应答。如果未

设置端点的 IN 状态或 OUT 状态，则默认挂起。

第 3 步：进入请求结束状态：端点 0 设置成空闲状态，存器 CSR0 的 SPORDY 位以及 DATEND 位置位，表示处理完成从端点 0 接收到的数据包，且数据结束。

(4)SET_ADDRESS

依据以表 15.36，可以对该请求的特殊情况进行设计

表 15.36 SET_ADDRESS 内容

bmReguestType	bRequest	wValue	wIndex	wLength	数据过程
0x00	SET_ADDRESS	设备地址	0	0	无

第 1 步：特殊情况的处理，与 GET_STATUS 请求的设计思路一致，依旧是特殊情况处理。如果 USB 标准设备请求数据结构不满足表 15.36 所有条件，则挂起。

第 2 步：依据 USB 标准设备请求数据结构 wValue 字段确定主机给设备分配的唯一地址，之后将主机分配给设备的唯一地址写入 FADDR 寄存器中，表示设备已接收并设置了地址；最后依据地址是否为零设置设备的状态，如果地址为零，设备为默认状态；地址不为零，设备状态为 SET_ADDRESS。

第 3 步：进入请求结束状态：端点 0 设置成空闲状态，寄存器 CSR0 的 SPORDY 位以及 DATEND 位置位，表示处理完成从端点 0 接收到的数据包，且数据结束。

(5)GET_DESCRIPTOR

GET_DESCRIPTOR 请求是 USB 通信中最常使用的请求，有关此请求的内容十分重要，可以说 STEUP 阶段的主要数据负载就在此请求的响应中，首先，按照惯例，依据表 15.37，可以对该请求的特殊情况进行设计。

表 15.37 GET_DESCRIPTOR 内容

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
0x80	GET_DESCRIPTOR	描述符类型和索引	0 或者语言 ID	描述符长度	描述符

第 1 步：特殊情况的处理，与 GET_STATUS 请求的设计思路一致，依旧是特殊情况处理。

如果 USB 标准设备请求数据结构不满足表 15.37 所有条件，则挂起。

第 2 步：依据 wValue 的值判定需要发送的描述符类型，表 15.38 将列出本次实验中 wValue 所确定的所有描述符类型。

表 15.38 wValue 所确定的所有描述符类型

wValue	值	描述符类型
DESC_DEVICE	0x0100	设备描述符
DESC_CONFIGURATION	0x0200	配置描述符
DESC_STRING	0x0300 0x0301 0x0302	LANGIDDESC MANUFACTDESC PRODUCTDESC
DESC_HIDREPORT	0x2200	报告描述符

若 wValue 不为以上情况，则默认为挂起操作。

注意：

1.wValue 的第一字节(字符)表示同一种描述类型(比如字符串描述符)中具体的某个描述符(如厂商

或者产品字符)，第二字节表示描述类型的编号。

2.对于全速和低速模式，获取描述符的标准请求只有三种：获取设备、配置、字符串的描述符，另外的接口和端点描述符是跟随配置描述符一并返回的，不能单独请求返回。

3.描述符的数据结构参照 1.3 节中，描述符

第 3 步：进入标准请求结束状态(进行数据过程)，例如：在步骤 2 中软件已经依据 USB 协议内容将 DEVICEDESC 设备描述符数据包装入了数据缓冲区，然后软件再通过调用 usb_setup_in()，完成数据包的发送。

注意：usb_setup_in()的内容是：先将控制端点 0 设置为 DATAIN 状态，控制寄存器 CSR0 的 SPORDY 写 1 以清零 OPRDY，随后调用 usb_ctrl_in()进行数据发送，即使用控制 IN 类型向主机发送数据包。

(6)SET_DESCRIPTOR

此次实验不涉及该标准请求，故不对内容进行编写，若进入该函数，则调用 USB 挂起函数以替代。

(7)GET_CONFIGURATION

依据表 15.39，可以对该请求的特殊情况进行设计

表 15.39　GET_CONFIGURATION 内容

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
0x80	GET_CONFIGURATION	0	0	1	配置值

第 1 步：特殊情况的处理，与 GET_STATUS 请求的设计思路一致，依旧是特殊情况处理。如果 USB 标准设备请求数据结构不满足表 15.39 所有条件，则挂起。

第 2 步：依据设备的状态选择需要发送的数据包，如果设备已配置，则发送 PACKET1 数据包，如果未被配置则发送 PACKET0 数据包。

第 3 步：进入标准请求结束状态(进行数据过程)，例如：在步骤 2 中软件已经依据 USB 协议内容将 PACKET0 数据包装入了数据缓冲区，然后软件再通过调用 usb_setup_in()，完成数据包的发送。

注意：usb_setup_in()的内容是：先将控制端点 0 设置为 DATAIN 状态，控制寄存器 CSR0 的 SPORDY 写 1 以清零 OPRDY，随后调用 usb_ctrl_in()进行数据发送，即使用控制 IN 类型向主机发送数据包。

(8)SET_CONFIGURATION

SET_CONFIGURATION 和 SET_ADDRESS 请求很类似，区别是 wValue 的意义：SET_ADDRESS 中，wValue 的第一字节(低字节)表示设备的地址，SET_CONFIGURATION 则为配置的值。该值与配置描述符的配置编号一致时，表示选中该配置，通常为 1，因为大多数 USB 设备只有一种设置；若为 0，则设备进入地址设置状态。

依据表 15.40，可以对该请求的特殊情况进行设计。

表 15.40　SET_CONFIGURATION 内容

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
0x00	SET_CONFIGURATION	配置值（0x0001）	0	0	无

第 1 步：特殊情况的处理，与 GET_STATUS 请求的设计思路一致，依旧是特殊情况处理。如果 USB 标准设备请求数据结构不满足表 15.40 所有条件，则挂起。

第 2 步：依据 wValue 的低字节确定配置值，在操作寄存器配置设备状态前先设置设备状态为配置状态，并将 IN 端点状态以及 OUT 端点状态设为 0 状态，之后我们依据需要使用的端点配置相应的寄存器，例如：我们需要使用端点 1 作为 IN 端点，则先配置 INDEX 寄存器定位到 1，然后配置 INCSR2 寄存器的 MODEIN 位为 1，从而设置端点的方向，最后配置 INMAXP 寄存器来设置 USB 的 IN 端点 1 最大数据包的大

小。如果 wValue 值不为 0x0001，则设备回到配置地址的状态。

第 3 步：进入请求结束状态：端点 0 设置成空闲状态，寄存器 CSR0 的 SPORDY 位以及 DATEND 位置位，表示处理完成从端点 0 接收到的数据包，且数据结束。

(9)GET_INTERFACE

依据表 15.41，可以对该请求的特殊情况进行设计。

表 15，41 GET_INTERFACE 内容

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
0x80	GET_INTERFACE	0	接口号	1	备用接口号

第 1 步：特殊情况的处理，与 GET_STATUS 请求的设计思路一致，依旧是特殊情况处理。如果 USB 标准设备请求数据结构不满足表 15.41 所有条件，则挂起。

第 2 步：选择发送 PACKET0 数据包

第 3 步：进入标准请求结束状态(进行数据过程)，例如：在步骤 2 中软件已经依据 USB 协议内容将 PACKET0 数据包装入了数据缓冲区，然后软件再通过调用 usb_setup_in()，完成数据包的发送。

注意：usb_setup_in()的内容是：先将控制端点 0 设置为 DATAIN 状态，控制寄存器 CSR0 的 SPORDY 写 1 以清零 OPRDY，随后调用 usb_ctrl_in()进行数据发送，即使用控制 IN 类型向主机发送数据包。

(10)SET_INTERFACE

依据表 15.42，可以对该请求的特殊情况进行设计。

表 15.42 SET_INTERFACE 内容

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
0x01	SET_INTERFACE	备用接口号	接口号	0	无

第 1 步：特殊情况的处理，与 GET_STATUS 请求的设计思路一致，依旧是特殊情况处理。如果 USB 标准设备请求数据结构不满足表 15.42 所有条件，则挂起。

第 2 步：无操作。

第 3 步：进入请求结束状态：端点 0 设置成空闲状态，寄存器 CSR0 的 SPORDY 位以及 DATEND 位置位，表示处理完成从端点 0 接收到的数据包，且数据结束。

(11)SYNCH_FRAME

此次实验不涉及该标准请求，故不对内容进行编写，若进入该函数，则调用 USB 挂起函数以替代。

S.3 人机交互设备原理

S.3.1 HID 类设备简介

1. 设备范围

HID(human interface device)类主要由人类用来控制计算机系统操作的设备组成。HID 类设备的典型示例包括：

(1)键盘和指针设备：标准鼠标设备、跟踪球和操纵杆。

(2)面板控制：旋钮、开关、按钮和滑块。

(3)电话、VCR 遥控器、游戏或模拟设备等设备上的控件：数据手套、油门、方向盘和方向舵踏板。

(4)某些 HID 类设备可能不需要人工交互，但可以提供 HID 类似的格式数据的设备：条形码阅读器、温度计或电压表。

许多典型的 HID 类设备包括指示器、专用显示器、音频反馈以及力或触觉反馈。因此，HID 类定义包括对针对最终用户的各种类型的输出的支持。

2. HID 规范的意义

HID 规范的主要意义是：

(1)尽最节省设备的数据空间。

(2)允许软件应用程序跳过未知信息。

(3)可扩展且鲁棒。

(4)支持嵌套和集合。

(5)自我描述以允许通用软件应用程序。

S.3.2 主机请求

1. 标准请求

HID 类使用 USB 标准请求 GetDescriptor,如 USB 规范中所述。当发出 GetDescriptor (configuration)请求时，它返回每个接口的配置描述符，所有接口描述符、所有端点描述符和 HID 描述符。它不应返回字符串描述符、HID 报告描述符或其他可选的 HID 类描述符。HID 描述符应处于 HID 接口的接口和端点描述符之间。顺序应为：

(1)配置描述符

(2)接口描述符(指定 HID 类)

(3)HID 描述符(与上述接口关联)

(4)端点描述符(用于 HID 中断端点)

(5)可选端点描述符(用于 HID 中断输出端点)

注：GetDescriptor 可用于检索标准，类别和供应商具体描述符，取决于描述符类型字段的设置。

表 15.43 定义了 Descriptor Type(Get_Descriptor 请求中 wValue 的高字节)。

表 15.43 Descriptor Type 说明

部分	说明
Deseriptor Type	指定 Deseriptor Type 的特征位 7 保留(应始终 0) 6..5 类型 0=标准 1=类 2=供应商 3=保留 4..0 描述符 请参阅标准类或供应商描述符类型表。

表 15.44 定义了类描述符的有效类型。

表 15.44 类描述符的有效类型

值	类描述符类型
0x21	HID
0x22	报告
0x23	物理描述符
0x24-0x2F	保留

(1)GetDescriptor 请求

GetDescriptor 请求返回设备的描述符，如表 15.45 所示。

表 15.45 GetDescriptor 数据结构的构成

字段	USB 标准描述符	HID 类描述符
bmRequestType	100 xxxxx	10000001
bRequest	GET_DESCRIPTOR(0x06)	GET_DESCRIPTOR(0x06)
wValue	描述符类型一节描述符索引	描述符类型一节描述符索引
wIndex	0 或者语言 ID	接口号
wLength	描述符长度	描述符长度
Data	描述符	描述符

注意：

- 于标准 USB 描述符，bmRequestType 的位 0~4 表示请求的描述符是否与设备、接口、端点或其他相关联。
- wValue 字段中高字节指定描述符类型，低字节指定描述符索引。
- 低字节是用于指定物理描述符集的描述符索引，对于其他 HID 类描述符，低字节将重置为零。

1)如果正在请求 HID 类描述符,则 wIndex 字段指示 HID 接口的编号。如果请求标准描述符,则 wIndex

字段指定字符串描述符的语言 ID，并为其他标准描述符重置为零。

2)请求物理描述符集 0 返回一个特殊的描述符，标识描述符集的数量及其大小。

3)物理索引等于 1 的 GetDescriptor 请求将请求第一个物理描述符集。设备可能对其项目有其他用途。可以通过在递增描述符索引的同时发出后续 GetDescriptor 请求来枚举这些请求。设备将向索引大于 HID 描述符中定义的最后一个数字的请求返回最后一个描述符集。

(2)SetDescriptor 请求

Set_Descriptor 请求允许主机更改设备中的描述符。对该请求的支持是可选的，如表 15.46 所示。

表 15.46　Set_Descriptor 数据结构的构成

字段	USB 标准描述符	HID 类描述符
bmRequestType	100xxxxx	10000001
bRequest	SET_DESCRIPTOR(0x07)	SET_DESCRIPTOR(0x07)
wValue	描述符类型一节描述符索引	描述符类型一节描述符索引
wIndex	0 或者语言 ID	接口号
wLength	描述符长度	描述符长度
Data	描述符	描述符

2.类特定请求

此部分请求特定于 HID 类设备，请求允许主机查询设备的功能和状态，并设置输出和功能项的状态。这些事务通过控制管道完成，因此遵循 USB 规范中定义的控制管道中标准请求格式。表 15.47 为特定类请求数据结构的一般格式。

表 15.47　特定类请求数据结构

字段	偏移(字节)	说明
bmRequestType	0/1	指定请求特征的位。有效值为 10100001 或 00100001 仅基于以下描述: 7　数据传输方向 0=从主机到设备 1=从设备到主机 6..5 类型 1=类 4..0 接收对象 1=接口
bRequest	1/1	详细的请求
wValue	2/2	指定字长字段的数值表达式(根据请求变化而变化)
wIndex	4/2	指定字长字段的索引或偏移量(根据请求变化而变化)
wLength	6/2	指定在数据阶段传输的字节数

表 15.48 定义了 bRequest 的有效值。

表 15.48 bRequest 有效值表

请求	bRequest 值	功能
GET_REPORT	0x01	请求允许主机通过控制管道接收报告
SET_REPORT	0x09	请求允许主机向设备发送报告，以设置输入、输出或功能控件的状态
GET_IDLE	0x02	请求读取特定输入报告的当前空闲率
SET_IDLE	0x0A	请求屏蔽中断输入管道上的特定报告，直到新事件发生或经过指定的时间间隔
GET_PROTOCOL	0x03	请求读取当前已激活的协议(例如引导协议或报告协议)
SET_PROTOCOL	0x0B	切换引导协议或报告协议

注意：以上请求所有设备需要强制支持，以上请求仅启动引导类设备需要。

1)Get_Report 请求

Get_Report 请求允许主机通过控制管道接收报告。

表 15.49 Get_Report 请求内容

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
0xA1	GET_REPORT	报告类型和报告 ID	接口号	报告大小	报告

wValue 字段在高字节中指定报告类型，在低字节中指定报告 ID。如果不使用报告 ID，需要将报告 ID 设置为 0(零)。报告类型指定如见表 15.50。

表 15.50 wValue 字段值表

值	报告类型
01	输入
02	输出
03	特征
04-FF	保留

此请求在初始化绝对项和确定特征项的状态时很有用。此请求不用于定期轮询设备状态。

中断 IN 管道应该用于重复的输入报告。输入报告回复与来自中断管道的报告具有相同的格式。

可以选择将中断输出管道用于低延迟输出报告。如果未声明中断输出端点，则通过中断输出管道的输出报告具有与通过控制管道发送的输出报告相同的格式。

2)Set_Report 请求

Set_Report 请求允许主机向设备发送报告，以设置输入、输出或控件功能，如表 15.51 所示。

表 15.51 Set_Report 请求内容

bmRequestType	bRequest	wValue	wIndex	wLenght	数据
0x21	SET_REPORT	报告类型和报告 ID	接口号	报告大小	报告

Set_Report 请求的请求字段的含义与 Get_Report 请求相同，只是数据方向相反，报告数据是从主

机发送到设备的。

设备可能会选择忽略输入的 Set_Report 请求，因为它没有意义。或者，这些报告可用于重置控件的原点(即，当前位置应报告为零)。发送报告的效果还取决于收件者控件是绝对的还是相对的。

3)Get_Idle 请求

Get_Idle 请求读取特定输入报告的当前空闲率，如表 15.52 所示。

表 15.52 Get_Idle 请求内容

bmRequestType	bRequest	wValue	wIndex	wLength	数据
0x81	GET_IDLE	0 和报告 ID	接口号	1	空闲速率

4)Set_Idle 请求

Set_Idle 请求使中断输入管道上的特定报告被屏蔽，直到发生新事件或经过指定的时间量，如表 15.53 所示。

表 15.53 Set_Idle 请求内容

bmRequestType	bRequest	wValue	wIndex	wLength	数据
0x21	SET_IDLE	持续和报告 ID	接口号	0	无

备注：该请求用于限制端点中断的报告频率。具体来说，该请求会导致端点对端点中的中断进行任何轮询，而其当前报告保持不变。在没有更改的情况下，轮询将在给定的基于时间的持续时间内继续进行 NAK。该请求包含以下部分，如表 15.54 所示。

表 15.54 Set_Idle 请求特性

部分	说明
持续	当 wValue 的高字节为 0(零)时，持续时间是不确定的。端点将永远禁止报告，仅在报告数据中检测到更改时才报告。 当 wValue 的高字节不为零时，则使用固定的持续时间。持续时间将与高字节的值线性相关，LSB 的权重为 4 毫秒。这提供了从 0.004 到 1.020 秒的值范围，分辨率为 4 毫秒。如果持续时间小于设备轮询率，则以轮询率生成报告。 如果给定的持续时间过去了，报告数据没有变化，那么端点将生成单个报告，并且报告抑制将使用先前的持续时间重新开始。
报告 ID	如果 wValue 的低字节为零，则空闲率适用于设备生成的所有输入报告。当 wValue 的低字节为非零时，则空闲率仅适用于低字节的值指定的 Report ID。
精度	此持续时间应具有+/-(10%+2 毫秒)的精度
延迟	如果在当前执行周期结束前至少 4 毫秒收到新请求，则新请求将被执行，就好像它是在上次报告之后立即发出的一样。如果在当前周期结束后的 4 毫秒内收到新请求，则新请求将在报告之后才生效。 如果当前期间已超过新规定的持续时间，则将立即生成报告。

如果端点的中断为多个报告提供服务，Set_Idle 请求可用于仅影响为指定报告 ID 生成重复报告的速率。例如，具有两个输入报告的设备可以为报告 1 指定 20ms 的空闲率，为报告 2 指定 500ms 的空闲率。

推荐的默认空闲速率(设备初始化时的速率)对于键盘是 500ms(在第一次重复速率之前的延迟)，对于操纵杆和鼠标是无穷大。

5)Get_Protocol 请求

Get_Protocol 请求读取当前处于活动状态的协议(引导协议或报告协议)，如表 15.55 所示。

表 15.55　Get_Protocol 请求内容

bmRequestType	bRequest	wValue	wIndex	wLength	数据
0x81	GET_PROTOCOL	0	接口号	1	0=boot 协议 1=报告协议

Boot 子类中的设备支持此请求。wValue 字段指示应该使用哪个协议。

6)Set_Protocol 请求

Set_Protocol 在引导协议和报告协议之间切换(反之亦然)，如表 15.56 所示。

表 15.56　Set_Protacal 请求内容

bmRequestType	bRequest	wValue	wIndex	wLength	数据
0x21	SET_PROTOCOL	0=Boot 协议 1=报告协议	接口号	0	无

启动子类中的设备支持此请求。wValue 字段指示应该使用哪个协议。

初始化时，所有设备默认报告协议。然而，主机不应该对设备的状态做出假设，并且应该在初始化设备时设置所需的协议。

S.3.3　描述符

有关 USB 设备的信息存储在其 ROM 的段中。这些段称为描述符。描述符分为 HID 标准描述符和类特定描述符，而 HID 类设备的描述符结构可由图 15.29 描述。

1. 标准描述符

HID 类设备类使用以下标准 USB 描述符。

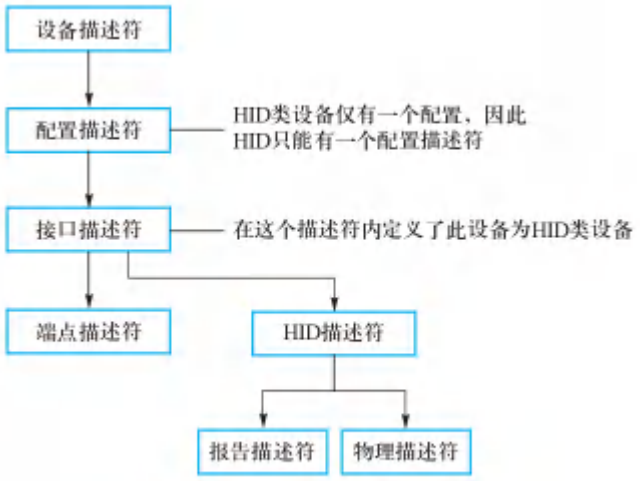


图 15.29　HID 类设备的描述符结构

1)设备描述符

注意：设备描述符中的 bDeviceClass 和 bDeviceSubClass 字段不应用于将设备标识为属于 HID 类。标识设备是 HID 类是使用接口描述符中的 bInterfaceClass 和 bInterfaceSubClass 字段。

2)配置描述符

3)接口描述符

- bInterfaceClass 字段始终为 3。
- bInterfaceSubClass 字段若支持引导协议为 1，否则为 0
  - bInterfaceProtocol 字段有在 bInterfaceSubClass 字段声明设备支持启动接口时才有意义,否则为 0。bInterfaceProtocol 字段值若为 0，则表示无设备，bInterfaceProtocol 字段值若为 2 表示该设备为鼠标设备，bInterfaceProtocol 字段值若为 1 表示该设备为键盘设备。其余值作保留

4)端点描述符

5)字符串描述符

有关标准描述符数据结构其他内容请参照 USB 标准描述符一节

2.类特定描述符

每个设备类包含一类或多类特定的描述符。这些描述符区别于标准 USB 描述符。HID 类设备使用以下特定类描述符：

- HID 描述符(HID)
- 报告描述符(Report)
- 物理描述符(Physical)

1)HID 描述符

HID 描述符标识设备的从属描述符的长度和类型。

表 15.57 HID 描述符字段描述

字段	偏移位/大小(字节)	描述
bLength	0/1	表示 HID 描述符总大小的数值表示。
bDescriptorType	1/1	指定 HID 类型的常量名称描述符。
bcdHID(版本)	2/2	标识 HID 类规范版本的数值表示。
bCountryCode	4/1	识别本地化硬件的国家代码的数值表示。
bNumDescriptors	5/1	指定类描述符数量的数值表示总是至少一个，即报告描述符。
bDescriptorType	6/1	标识类描述符类型的常量名称。
wDescriptorLength	7/2	表示报告描述符总大小的数值表式。
[bDescriptorType]...	9/1	指定可选描述符类型的常量名称。
[wDescriptorLength]...	10/2	表示可选描述符总大小的数值表式。

注意：

- 如果指定了可选描述符，则相应的长度项目也必须是指定的。
- 多个可选描述符和相关长度可以分别指定为偏移量(3*n)+6 和(3*n)+7。

- 值 bNumDescriptors 标识附加类特定的数量存在的描述符。此数字必须至少为一(1)，因为报告描述符始终存在。HID 描述符的其余部分具有每个附加类描述符的长度和类型。
- 值 bCountryCode 标识硬件的国家/地区从属。大多数硬件未本地化，因此该值为零(0)。但是，键盘可以使用该字段来指示键帽的语言。设备不需要在此字段中放置零以外的值，但某些操作环境可能需要此信息。表 15.58 指定了有效的国家代码。

表 15.58　有效国家代码

码值(十进制)	国家	码值(十进制)	国家
00	Not Supported	18	Netherlands/Dutch
01	Arabic	19	Norwegian
02	Belgian	20	Persian(Farsi)
03	Canadian-Bilingual	21	Poland
04	Canadian-French	22	Portuguese
05	Czech Republic	23	Russia
06	Danish	24	Slovakia
07	Finnish	25	Spanish
08	French	26	Swedish
09	German	27	Swiss/French
10	Greek	28	Swiss/German
11	Hebrew	29	Switzerland
12	Hungary	30	Taiwan
13	International(ISO)	31	Turkish-Q
14	Italian	32	UK
15	Japan(Katakana)	33	US
16	Korean	34	Yugoslavia
17	Latin American	35	Turkish-F

2)报告描述符

报告描述符是 HID 设备最重要的一个描述符，报告描述符重要的作用是描述在中断传输中的报告的格式。对于初学者，仅仅需了解报告描述符的构成即可。要注意，报告描述符和报告是两个概念。

报告描述符与其他描述符不同，因为它不仅仅是一个值表。报告描述符的长度和内容因设备报告所需的一个或多个报告所需的数据字段数而异。报告描述符由提供有关设备的信息的项目(item)组成。项目的第一部分包含三个字段：项目类型(item type)、项目标签(item tag)和项目大小(item size)。这些字段共同标识项目提供的信息类型。

有三种项目类型：“主项目(main)”、“全局项目(global)”和“本地项目(local)”。主项目标签有五个，分别如下：

① 输入项标签(input item tag)：指来自设备上一个或多个类似控件的数据。例如，可变数据(例如读取单个轴或一组杠杆的位置)或阵列数据(例如一个或多个按钮或开关)。

② 输出项标签(output item tag)：指的是设备上一个或多个类似控件的数据。例如，设置单个轴或一组杠杆的位置(可变数据)。或者，它可以将数据表示到一个或多个 LED(阵列数据)。

③ 特征项标签(feature item tag)：描述不用于最终用户使用的设备输入和输出。例如，软件功能或控制面板切换。

④ 集合项标签(collection item tag)：输入、输出和功能项的有意义的分组。例如，鼠标、键盘、操纵杆和指针。

⑤ 结束集合项目标签(end collection tag)：用于指定项目集合结束的终止项目。

报告描述符提供了设备中每个控件所提供的数据的描述。每个主项目标签(输入 Input、输出 Output 或特征 Feature)确定由特定控件返回的数据的大小，并确定数据是绝对的还是相对的，以及其他相关的信息。前面的局部和全局项目定义了最小和最大的数据值，以此类推。一个报告描述符是一个设备的所有项目的完整集合。仅仅通过查看报告描述符，应用程序就可以知道如何处理传入的数据，以及数据可以用来做什么。

控件的一个或多个数据字段由一个主项定义，并由前面的全局和本地项进一步描述。本地项只描述下一个主项所定义的数据字段。全局项是该描述符中所有后续数据字段的默认属性。

报告描述符的构成：一个报告描述符可能包含几个主项。一个报告描述符必须包含以下各项来描述控件的数据(所有其他项都是可选的)：

- Input(Output or Feature)//输入或输出标志
- Usage
- Usage Page
- Logical Minimum
- Logical Maximum
- Report Size
- Report Count

以下是 3 键鼠标的项目的编码示例。在这种情况下，主项目前面是全局项目，例如使用(Usage)、报告数量(Report Count)或报告大小(Report Size)(每行都是一个新项目)。

代码清单 15-12　报告描述符示例

```
Usage Page(Generic Desktop),       ;Use the Generic Desktop Usage Page
Usage(Mouse),
    Collection (Application),      ;鼠标集合起始
    Usage(Pointer),
    Collection(Physical),          ;指针集合起始
        Usage Page(Buttons),
        Usage Minimum(1),
        Usage Maximum(3),
        Logical Minimum(0),
        Logical Maximum(1),        ;Fields return values from 0 to 1
        Report Count(3),
```

```
        Report Size(1),           ;Create three 1 bit fields (button 1,2,& 3)
        Input(Data, Variable, Absolute),
                                  ;Add fields to the input report.
        Report Count (1),
        Report Size(5),           ;Create 5 bit constant field
        Input(Constant),          ;Add field to the input report
        Usage Page(Generic Desktop),
        Usage(x),
        Usage(Y),
        Logical Minimum(-127),
        Logical Maximum(127),     ;Fields return values from -127 to 127
        Report Size (8),
        Report Count(2),          ;Create two 8 bit fields(X & Y position)
        Input(Data, Variable, Relative),
                                  ;Add fields to the input report
    End collection,               ;指针集合结束
End Collection                    ;鼠标集合结束
```

(1)项目的类型和标签

所有项目都包含一个 1 字节的前缀，表示项目的基本类型。HID 类定义了项目的两种基本格式：

- 短项目：总长度 1-5 个字节；用于最常见的项目。一个短项通常包含 1 或 0 个字节的可选数据。
- 长项目：长度为 3-258 字节；用于需要较大数据的项目零件的结构。

注意：本文仅对短项目进行说明，长项目与短项目的概念与主项目、全局项目、本地项目的概念是不同的。

(2)短项目

程序设计中使用的项目均为短项，故读者需要仔细阅读短项的内容。短项目格式将项目大小、类型和标签由第一个字节描述。第一个字节后面可能跟着 0，1、2 或 4 个可选数据字节，具体取决于数据的大小。

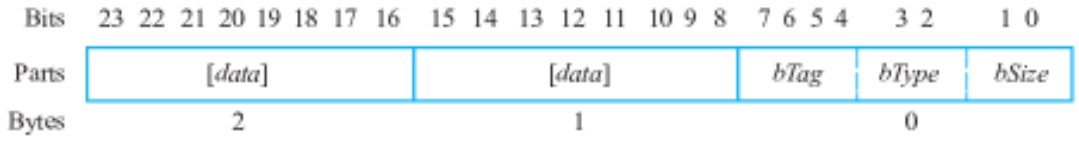


图 15.30　短项目位图

其中具体的字段的定义如表 15.59 所示。

表 15.59　字段含义

字段	描述
bSize	指明数据的大小 0=0 bytes，1=1 byte，2=2 bytes，3=4 bytes
bType	指定项目类型

字段	描述
	0=主项目，1=全局项目，2=本地项目，3=保留
bTag	指定该项目的功能
[data]	可选的数据

注意

- 短项目标签没有明确的 bSize 值与之关联。相反，项目数据部分的值决定了项目的大小。也就是说，如果项目数据可以用一个字节表示，那么数据部分可以指定为 1 个字节。
- 如果预计有一个长的数据项，如果它的高位都是 0，此数据仍可以被缩写。例如，一个 32 位的数据，其中字节 1、2、3 都是 0，则该数据可以缩写为一个字节。
- 短项目分为三类：主项目、全局项目和本地项目。项目类型(bType)指定项目类别。

(3)长项目

与短项格式一样，长项格式将项大小，类型和标签由第一个字节描述。长项格式使用一个特殊的项目标签值来指示它是一个长项。项目数据最多可包含 255 个字节的数据，如图 15.31 所示。

图 15.31　长项目位图

其中具体的字段的定义如表 15.60 所示。

表 15.60　字段含义

字段	描述
bSize	指定项目的总大小，其中大小为 10(2 字节)；表示项目类型为长项目
bType	指定项目类型的数字表达式，其中 3=保留
bTag	指定该项目的功能，只能是 1111
[bDataSize]	长项目数据大小
[bLongItemTag]	长项目标签
[data]	可选的数据

重要提示：本文中没有长项目标签相关的定义。这些标签被保留给将来使用。标签 xF0-xFF 由供应商定义。

(4)主项目

主项目用于定义或分组报告描述符中的某些类型的数据字段，如表 15.61 所示。主项目有两种类型：数据类型和非数据类型。数据类型用于在报告中创建字段，包括输入项目、输出项目和特征项目。随后接着的字段为非数据类型，该类型不在报告中创建字段。

所有主项目默认数据值为 0。

表 15.61 主项目有效值表

主项的标签	一字节前缀 (nn 表示大小)	有效值	
Input(输入项)	1000 00 nn	Bit 0 Bit 1 Bit 2 Bit 3 Bit 4 Bit 5 Bit 6 Bit 7 Bit 8 Bit 32-9	{数据(0)|常数(1)} {数组(0)|变量(1)} {绝对(0)|相对(1)} {无回转(0)|回转(1)} {线性(0)|非线性(1)} {首选状态(0)|非首选状态(1)} {无空状态(0)|空状态(1)} 保留(0){字段(0)|缓冲作用字节(1)} 保留(0)
Output(输出项)	1001 00 nn	Bit 0 Bit 1 Bit 2 Bit 3 Bit 4 Bit 5 Bit 6 Bit 7 Bit 8 Bit 32-9	{数据(0)|常数(1)} {数组(0)|变量(1)} {绝对(0)|相对(1)} {无回转(0)|回转(1)} {线性(0)|非线性(1)} {首选状态(0)|非首选状态(1)} {无空状态(0)|空状态(1)} {非易失(0)|易失的(1)} {字段(0)|缓冲作用字节(1)} 保留(0)
Feature(特征项)	1011 00 nn	Bit 0 Bit 1 Bit 2 Bit 3 Bit 4 Bit 5 Bit 6 Bit 7 Bit 8 Bit 32-9	{数据(0)|常数(1)} {数组(0)|变量(1)} {绝对(0)|相对(1)} {无回转(0)|回转(1)} {线性(0)|非线性(1)} {首选状态(0)|非首选状态(1)} {无空状态(0)|空状态(1)} {非易失的(0)|易失的(1)} {字段(0)|缓冲作用字节(1)} 保留(0)
Collection(集合项)	1010 00 nn	0x00 0x01 0x02 0x03	Physical(group of axes) Application(mouse, keyboard) Logical(interrelated data) Report

主项的标签	一字节前缀 (nn 表示大小)	有效值	
		0x04 0x05 0x06 0x07-0x7F 0x80-0xFF	Named Array Usage Switch Usage Modifier Reserved Vendor-defined
End Collection	1100 00 nn	无	关闭项目集合
Reserved	1101 00 nn 到 1111 00 nn	无	为以后项目作保留

(5)输入项、输出项以及特征项

输入、输出和特征项用于在报告中创建数据字段。

输入项描述了有关由一个或多个物理控件提供的数据的信息。应用程序可以使用此信息来解释设备提供的数据。在单个项目中定义的所有数据字段共享相同的数据格式。

输出项用于定义报告中的输出数据字段。除了描述发送到设备的数据(例如键盘 LED 状态)外，该项目类似于输入项目。

特征项描述可以发送到设备的设备配置信息。

表 15.62 输入项、输出项以及特征项的详细描述

位	Part	值	描述
0	数据|常数	0|1	指示项目是数据还是常量值 数据：该项目正在定义包含可修改设备数据的报告字段 常量:该项目是报告中的静态只读字段,不能由主机修改(写人)
1	数组|变量	0|1	指示项目是否在报告中创建变量或数组字段。 变量：每个字段表示来自物理控件的数据。为每个字段保留的位数由前面的报告大小/报告计数项确定。例如，一组八个开/关开关可以由变量输入项声明的 1 个字节报告，其中每个位代表一个开关，开(1)或关(0)(报告大小=1，报告计数=8)。或者，可变输入项可以添加 1 个报告字节，用于表示四个三位按钮的状态，其中每个按钮的状态由两位表示(报告大小=2，报告计数=4)。或者来自变量输入项的1个字节可以表示操纵杆的x位置(报告大小=8，报告计数=1)。 数组：提供了另一种方法来描述从一组按钮返回的数据。数组更有效，但不如变量灵活。数组不为组中的每个按钮返回一个位，而是在每个字段中返回一个与按下的按钮相对应的索引(如键盘扫描代码)。数组字段中的超出范围的值被视为未断言的控件。数组中同时按下的按钮或键需要在多个字段中报告。因此，数组输入项(报告计数)中的字段数决定了可以同时报告的最大数量。使用具有 3 个 8 位字段的数组(报告大小=8，报告计数=3)，键盘最多可以同时报告三个键。逻辑最小值指定数组返回的最低索引值，

位	Part	值	描述
			逻辑最大值指定最大值。可以通过检查逻辑最小值和逻辑最大值之间的差异来推断数组中的元素数量(元素数量=逻辑最大值-逻辑最小值+1)
2	绝对|相对	0|1	指示数据是绝对的(基于固定的原点)还是相对的(指示上次报告中值的变化) 鼠标设备通常提供相对数据 平板电脑通常提供绝对数据
3	无回转|回转	0|1	指示数据在达到极高值或极低值时是否“翻转”。例如，可以自由旋转 360 度的表盘可能会输出 0 到 10 的值。如果指示回转，在递增方向经过 10 位置后报告的下一个值将是 0
4	线性|非线性	0|1	指示来自设备的原始数据是否经过某种方式处理，不再代表测量的数据与报告的数据之间的线性关系，加速度曲线和操纵杆死区就是这类数据的例子。灵敏度设置会影响单元项目，但数据仍然是线性的
5	首选状态|非首选状态	0|1	指示控件是否具有当用户未与控件进行物理交互时它将返回到的首选状态。按钮(与切换按钮相反)和自定义操纵杆就是示例
6	无空状态|空状态	0|1	指示控件是否处于未发送有意义数据的状态。空状态的一种可能用途是用于需要用户与控件进行物理交互以报告有用数据的控件。例如，一些操纵杆具有多向开关(帽子开关)。当帽子开关没有被按下时，它处于空状态。当处于空状态时，控件将报告指定逻辑最小值和逻辑最大值之外的值(最大负值，例如 8 位值的-128)
7	非易失的|易失的 Reserved	0|10	指示功能或输出控件的值是否应由主机更改。易失性输出可以在有或没有主机交互的情况下发生变化。为避免同步问题，易失性控制应尽可能是相对的。如果 volatile 输出是绝对的，则在发出 Set Report(Output)时，请求将您不想更改的任何控件的值设置为指定的逻辑最小值和逻辑最大值之外的值(最大负值，例如-128 为 8 位值)。设备会忽略对控件的无效输出 数据第 7 位没有定义输入项且其用作保留
8	字段|缓冲字节	0|1	指示控件发出固定大小的字节流。数据字段的内容由应用程序确定。缓冲区的内容不会被解释为单个数字量。由缓冲字节定义的报告数据必须在 8 位边界上对齐。来自条形码阅读器的数据就是一个例子
9-31	保留	0	保留

附注：

- 如果输入项是一个数组，则只有“数据/常数”、“变量/数组”还有“绝对/相对”属性适用。
- 可以通过检查“报告大小(Report Size)”和“报告计数(Report Count)”值来确定报告中的数据字段大小。例如，报告大小为 8，且报告计数为 3，则报告具有 3 个 8 位数据字段。
- 数组项返回的值是一个索引，因此建议：

1)当数组中没有控件被断言时，数组字段返回 0 值。

2)逻辑最小值(Logical Minimum)值等于 1。

3)逻辑最大值(Logical Maximum)等于数组中元素的数量。

- 输入项定义输入报告，可通过控制管道用 Get_Report(Input)请求访问。
- 输入报告也通过中断输入管道以轮询方式发送。
- 输出项的数据/常数、变量/数组，绝对/相对、非线性、包和空状态数据与输入项的数据相同。
- 输出项使输出报告可以通过 Set_Report(output)命令通过控制管道访问。
- 输出类型的报告可以选择通过一个中断输出管道发送。虽然功能相似，但输出项和特征项在以下方面有所不同：

1)特征项定义了设备的配置选项,通常由控制面板应用程序设置。因为它们会影响设备的行为(例如，按键重复速率、复位原点等)，特征项通常对软件应用程序不可见。相反，输出项代表设备对用户的输出(例如，LED、音频、触觉反馈等等)。软件应用可能会设置设备的输出项。

2)特征项目可以是其他项目的属性。例如，一个原点复位特性可能适用于一个或多个位置输入项目。与输出项一样，特征项可通过控制管道的 Get_Report 和 Set_Report 请求访问。

(6)集合项和集合结束项

集合项标识两个或多个数据(输入(Input)项、输出(Output)项或特征(Feature)项)之间的关系。例如，鼠标可以描述为两到四个数据(x 坐标、y 坐标、按键 1、按键 2)的集合。当使用集合项(Collection item)打开一个数据集合时，必须使用集合结束项(End Collection item)关闭一个集合。表 15.63 详细描述了集合项第一字节代表的含义。

表 15.63 集合项详细

集合的类型	值	描述
物理(physical)	0X00	物理集合用于代表在一个几何点收集的数据点的一组数据项目。这对于可能需要将测量或传感的数据集与一个点联系起来的传感设备很有用。它并不表示一组数据值来自一个设备，如键盘。在报告多个传感器位置的设备的情况下，物理集合被用来显示哪些数据来自于每个独立的传感器
应用(Application)	0X01	一组应用程序可能熟悉的主要项目。它还可用于识别在单个设备中服务于不同目的项目组。常见的例子是键盘或鼠标。带有集成定点设备的键盘可以定义为两个不同的应用程序集合。数据报告通常(但不一定)与应用程序集合相关联(每个应用程序至少有一个报告 ID)
逻辑(Logical)	0X02	当一组数据项形成一个复合数据结构时，就会使用逻辑集合。这方面的一个例子是数据缓冲区和数据的字节计数之间的关联。该集合建立了计数和缓冲区之间的联系
报告(report)	0X03	定义一个包含报告中所有字段的逻辑集合。此集合中将包含一个唯一的报告 ID。应用程序可以很容易地确定设备是否支持某个功能。请注意，可以为报告集合声明任意有效的报告 ID 值
命名数组(named array)	0X04	命名数组是一个逻辑集合，包含一个选择器用法的数组。对于给定的功能，类似设备使用的选择器集可能会有所不同。在记录硬件寄存器时，字段的命名是常见的做法。要确定设备是

集合的类型	值	描述
		否支持特定功能(如状态),应用程序可能必须查询几个已知的状态选择器用法，然后才能确定设备是否支持状态。命名数组用法允许命名包含选择器的数组字段，因此应用程序只需查询状态用法即可确定设备支持状态信息
用法切换(Usage Switch)	0X05	Usage Switch 是一个逻辑集合，可以修改它所包含的用法的含义。此集合类型向应用程序指示在此集合中找到的用法必须是特殊情况。例如，不是在 LED 页面上为每个可能的功能声明用法，而是可以将指示符用法应用于 Usage Switch 集合，并且该集合中定义的标准用法现在可以标识为功能的指示符，而不是功能本身。请注意，此集合类型不用于标记 Ordinal 集合，而是使用逻辑集合类型
用法修饰(Usage Modifier)	0X06	修改附加到包含集合的用法的含义。用法通常定义控件的单一操作模式。用法修饰允许扩展控件的操作模式。例如，LED 通常是打开或关闭的。对于特定状态，设备可能需要一种通用方法来闪烁或选择标准 LED 的颜色。将 LED 使用情况附加到 Usage Modifier 集合将向应用程序指示该使用情况支持新的操作模式
Reserved	0X07-0X7F 0X80-0XFF	保留 厂商定义

附注：

- 集合项和集合结束项之间的所有主项都包含在该集合中。一个集合可能包含其他集合中。
- 集合项不生成数据。但是，使用项标签必须与任何集合(例如鼠标或油门设备)相关联。集合项可以嵌套，且它们始终是可选的，但顶级应用程序集合除外。
- 如果遇到未知的供应商定义的集合类型，则应用程序必须忽略该集合中声明的所有主要项目。请注意，在该集合中声明的全局项目将影响状态表。
- 如果有未知用法附加到已知集合中，则应忽略该集合的内容。请注意，在该集合中声明的全局项目将依然影响状态表。
- 字符串和物理索引以及分隔符可能与集合相关联。

(7)全局项

全局项描述而不是定义来自控件的数据。一个新的主项目承担着项目状态表的特征。全局项目可以改变状态表。因此，除非被另一个全局项覆盖，否则全局项标记适用于所有随后定义的项。表 15.64 一字节前缀指的是项目的第 0 字节。

表 15.64 全局项

全局项标签 (global item tag)	一字节前缀 (nn 表示大小值)	描述
Usage page	0000 01nn	指定当前使用页面的无符号整数。由于使用是 32 位值，因此可以使用页面项通过设置后续使用的高 16 位来节省报告描述符中的空间。定义为 16 位或更少的任何使用都被解释为使用 ID

全局项标签 (global item tag)	一字节前缀 (nn 表示大小值)	描述
		并与使用页面连接形成一个 32 位的用法
Logical Minimum	0001 01nn	以逻辑单位表示的范围值。这是变量或数组项将报告的最小值。例如，报告 x 位置值从第 0 到 128 范围变化的鼠标，将具有 0 的逻辑最小值和 128 的逻辑最大值
Logical Maximum	0010 01nn	以逻辑单位表示的范围值。这是变量或数组项将报告的最大值
Physical Minimum	0011 01nn	变量项的物理范围的最小值。这表示应用了单位的逻辑最小值
Physical Maximum	0100 01nn	变量项的物理范围的最大值
Unit Exponent	0101 01nn	以 10 为底的单位指数值
Unit	0110 01nn	单位值
Report Size	0111 01nn	无符号整数，以位为单位指定报告字段的大小。这允许解析器构建一个项目映射供报告处理程序使用
Report ID	1000 01nn	指定报告 ID 的无符号值。如果在报告描述符的任何地方使用了报告 ID 标签，则设备的所有数据报告都在一个单字节的 ID 字段之前。在第一个报告 ID 标签之后，但在第二个报告 ID 标签之前的所有项目，都包括在一个以 1 字节 ID 为前缀的报告中。在第二个报告 ID 标签之后，但在第三个报告 ID 标签之前的，所有项目都包括在第二个报告中，以第二个 ID 为前缀，依此类推 这个报告 ID 值表示添加到一个特定报告的前缀。例如，一个报告描述符可以定义一个 3 字节的报告，报告 ID 为 01。这个设备将产生一个 4 字节的数据报告，其中第一个字节是 01。该设备还可以生成其他报告，每个报告都有一个独特的 ID。这允许主机区分不同类型的报告通过管道中的一个中断传送。并允许设备区分通过单个中断输出管道传送的不同类型的报告。报告 ID 0 保留，不能被使用
Report Count	1001 01nn	无符号整数，指定项目的数据字段数；确定该特定项目的报告中包含多少字段(以及因此向报告中添加多少位)
Push	1010 01nn	将全局项目状态表的副本放在堆栈上

全局项标签 (global item tag)	一字节前缀 (nn 表示大小值)	描述
Pop	1011 01nn	将项目状态表替换为堆栈中的栈顶
Reserved	1100 01nn to 1111 01nn	保留

附注：

- 虽然逻辑最小值和逻辑最大值(范围)限制了设备返回的值，但物理最小值和物理最大值通过允许报告值偏移和缩放来赋予这些边界意义。例如，温度计的逻辑范围可能为 0 和 999，但物理范围为 32 和 212。分辨率可以通过以下算法确定。

代码清单 15-13 算法

```
if ((Physical Maximum UNDEFINED)
|| (Physical Minimum == UNDEFINED)
|| ((Physical Maximum == 0) && (Physical Minimum == 0)))
{
  Physical Maximum = Logical Maximum;
  Physical Minimum = Logical Minimum;
}

If(Unit Exponent == UNDEFINED)
  Unit Exponent = 0;

Resolution = (Logical Maximum - Logical Minimum) / (Physical Maximum-Physcal Minimum)
* (10 Unit Exponent))
```

当线性解析报告描述符时，单位指数、物理最小值和物理最大值的全局状态值被认为处于“未定义”状态，直到它们被声明。

例如：一个 400-dpi 的鼠标可能有表 15.65 中所示的项目

表 15.65 鼠标项目

项目	值
Logical Minimum	-127
Logical Maximum	127
Physical Minimum	-3175
Physical Maximum	3175
Unit Exponent	-4
Unit	Inches

因此其计算分辨率的公式为：

Resolution=(127-(-127))/((3175-(-3175))*10-4)=400 计数值/英尺

单位项目的限定值如表 15.66 所示。

表 15.66 单位项目的限定值

半字节	单位制度	0x0	0x1	0x2	0x3	0x4
	Exponent	0	1	2	3	4
0	System	None	SI Linear	SI Rotation	English Linear	English Rotation
1	Length	None	Centimeter	Radians	Inch	Degrees
2	Mass	None	Gram	Cram	Slug	Slug
3	Time	None	Seconds	Seconds	Seconds	Seconds
4	Temperature	None	Kelvin	Kelvin	Fahrenheit	Fahrenheit
5	Current	None	Ampere	Ampere	Ampere	Ampere
6	Luminous intensity	None	Candela	Candela	Candela	Candela
7	Reserved	None	None	None	None	None

注意：对于系统部分，代码 0x5 至 0xE 是保留；代码 0xF 是供应商定义的

- 如果逻辑最小值和逻辑最大值都被定义为正值(0 或更大)，那么报告字段可以被假定为无符号值。否则，所有整数值都是以 2 的补码格式表示的有符号值。
- 在报告描述符中声明物理最小值和物理最大值之前，HID 分析器会假定它们分别等于逻辑最小值和逻辑最大值。将它们声明为可以应用于(输入项、输出项或特征项)主项后，它们将继续影响所有后续的主项。如果物理最小值和物理最大值的范围都等于 0，那么它们将恢复到它们的默认释义。
- 之前的表格中没有显示的代码和指数，如表 15.67 所示。

表 15.67 指数项

码值	指数
0x5	5
0x6	6
0x7	7
0x8	-8
0x9	-7
0xA	-6
0xB	-5
0xC	-4
0xD	-3

0xE	-2
0xF	-1

- 大多数复杂的单位可以从长度、质量、时间、温度、电流和发光强度的基本单位中得出。例如，能量(焦耳)可以表示为：

```
joule = [mass(grams)][length(centimeters)2][time(seconds)-2]
```

单位指数将是 7，因为一个焦耳是由公斤(1 公斤等于 103 克)和米组成，如表 15.68 所示。

表 15.68　示例

Nibble	Part	Value
3	Time	-2
2	Mass	1
1	Length	2
0	System	1

- 一些常见单位的部件如表 15.69 所示。

表 15.69　常见单位的部件

单位	半字节						
	5(i)	4(l)	3(l)	2(m)	1(l)	0(sys)	Code
Distance(cm)	0	0	0	0	1	1	x0011
Mass(g)	0	0	0	1	0	1	x0101
Time(s)	0	0	1	0	0	1	x1001
Velocity(cm/s)	0	0	-1	0	1	1	xF011
Momentum	0	0	-1	1	1	1	xF111
Acceleration	0	0	-2	0	1	1	xE011
Force	0	0	-2	1	1	1	xE111
Energy	0	0	-2	1	2	1	xE121
Angular Acceleration	0	0	-2	0	1	2	xE012
Voltage	-1	0	-3	1	2	1	x00F0D121

- 在数组的情况下，报告计数决定了可能包含在报告中的最大控件数量，因此也决定了可能同时按下的键或按钮的数量以及每个元素的大小。例如，一个支持最多同时按三个键的数组，每个字段是 1 个字节，看起来像这样：

```
…
Report size (8),
Report Count (3),
```

```
…
```

- 在变量项目的情况下，报告计数指定了报告中包括多少个控件。例如，八个按钮可以是这样的：

```
…
Report size (1),
Report Count (8),
…
```

- 如果使用报告 ID，则必须在报告描述符中的第一个输入、输出或功能主项声明之前声明一个报告 ID。
- 在报告描述符中可以多次遇到相同的报告 ID 值。随后声明的输入、输出或功能主要项目将在相应的 ID/类型(输入、输出或功能)报告中找到。

(8)本地项

本地项目标签定义控件的特征。这些项目不会延续到下一个主项目。如果一个主项定义了一个以上的控件，那么它前面可能有几个类似的本地项标签。例如，一个输入项可能有几个与之相关的使用标签。表 15.70 所示一字节前缀指的是项目的第 0 字节。

表 15.70　本地项

标签位	一字节前缀 (nn 表示大小值)	描述
Usage	0000 10 nn	项目使用情况的使用索引；表示项目或集合的建议用法。在一个项目表示多个控件的情况下，Usage 标签可能会建议数组中每个变量或元素的用法
Usage Minimum	0001 10 nn	定义与数组或位图关联的起始用法
Usage Maximum	0010 10 nn	定义与数组或位图关联的结束用法。
Designator Index	0011 10 nm	确定用于控件的主体部分。索引指向物理描述符中的指示符
Designator Minimum	0100 10 nn	定义与数组或位图关联的起始指示符的索引
Designator Maximum	0101 10 nn	定义与数组或位图关联的结束指示符的索引
String Index	0111 10 nn	字符串描述符的字符串索引;允许字符串与特定项或控件相关联
String Minimum	1000 10 nn	指定将一组连续字符串分配给数组或位图中的控件时的第一个字符串索引
String Maximum	1001 10 nn	指定将一组连续字符串分配给数组或位图中的控件时的最后一个字符串索引
Delimiter(定界符)	1010 10 nn	定义一组局部项的开始或结束(1=打开集(open set)，0=关闭集(close set))

标签位	一字节前缀 (nn 表示大小值)	描述
Reserved	1010 10 nn to 1111 10 nn	保留

附注：

- 虽然本地项目不会延续到下一个主要项目，但它们可能适用于单个项目中的多个控件。例如，如果定义五个控件的输入项前面有三个用法标记，则三个用法将按顺序分配给前三个控件，第三个用法也将分配给第四个和第五个控件。如果项目没有控件(报告计数=0)，则本地项目标签适用于主项目(通常是集合项目)。
- 要为单个主项中的每个控件分配唯一的用法，只需按顺序指定每个 Usage 标签(或使用 Usage Minimum 或 Usage Maximum)
- 所有本地项都是无符号整数。

  注意：正确使用用法很重要。虽然存在非常具体的用途(起落架、自行车车轮等)，但这些用途旨在识别具有非常具体应用的设备。带有通用按钮的操纵杆绝不应将特定于应用程序的用途分配给任何按钮。相反，它应该分配一个通用用法，例如“按钮”。但是，健身车或飞行模拟器的驾驶舱可能希望狭义地定义其每个数据源的功能。
- 同样重要的是要记住，使用项(Usage items)传达有关数据预期用途的信息，并且可能与实际测量的内容不对应。例如，操纵杆将具有与其轴数据关联的 X 和 Y(Y Usage)使用情况(而不是使用情况 Rx 和 Ry。)
- 因为按钮位图和数组可以用一个项目表示多个按钮或开关，所以将多个用途分配给一个主项目可能很有用。Usage Minimum 指定要与数组或位图中的第一个未关联控件关联的使用情况。使用最大值指定与项目元素关联的使用值范围的结束。以下示例明了如何将其用于 105 键键盘，如表 15.71 所示。

表 15.71 键盘报表描述符

Tag(标签)	Result(结果)
Report Count(1)	一个字段将添加到报告中
Report Size(8)	新增字段的大小为 1 字节(8 位)
Logical Minimum(0)	将 0 定义为可能的最低返回值
Logical Maximum(101)	将 101 定义为可能的最高返回值，并将范围设置为 0 到 101。将 101 定义为可能的最高返回值，并将范围设置为 0 到 101
Usage page(0X07)	选择键盘使用页面。(select keyboard usage page)
Usage Minimum(0X00)	分配 101 键用法第一个键
Usage Maximum(0X65)	分配 101 键用法最后一个键
Input:(Data, Array, Absolute)	在中断报告中创建并添加 1 字节数组

- 如果将使用最小值(Usage Minimum)声明为扩展使用，则关联的使用最大值(Usage Maximum)也必须声明为扩展使用。
- Usage、Usage Minimum 或 Usage Maximum 项目的解释根据项目的 bSize 字段而有所不同。如果

bSize 字段=3，则该项目被解释为 32 位无符号值，其中高 16 位定义使用页面(Usage page)，低 16 位定义使用 ID(Usage ID)。定义使用页面和使用 ID 的 32 位使用项通常称为“扩展”使用。

如果 bSize 字段=1 或 2，则 Usage 被解释为在当前定义的 Usage Page 上选择 Usage ID 的无符号值。当解析器遇到一个主要项目时，它会将最后声明的 Usage Page 与一个 Usage 连接起来，形成一个完整的使用值。扩展使用可用于覆盖当前定义的单个使用的使用页面。

- 通过简单地将它们与分隔符项(delimiter item)括起来，可以将两个或多个替代用法与控件相关联。分隔符允许为控件定义别名，以便应用程序可以通过多种方式访问它，形成分隔集的用法按优先顺序组织，其中声明的第一个用法是控件的最首选用法。

  HID 解析器必须处理分隔符(delimiters)，但是，对它们定义的替代用法的支持是可选的。系统软件可能无法访问除定义的第一个(最首选)用法之外的其他用法。

定义适用于应用程序集合(Application collection)或数组项(Array item)的用法时，不能使用分隔符。

(9)填充

报告可以通过声明适当大小的主项和不声明主项的用法来填充到字节对齐的字段。

3)物理描述符

该特定描述符不在之后的程序设计中涉及，故在此不作讲解，读者如对此感兴趣请阅读 HID 官方协议。

S.3.4　报告

设备通过向主机发送报告，可以将诸如鼠标移动、点击等信息返回给主机。报告通过中断输入管道从设备发送到主机。报告也可以请求(轮询)报告并通过控制管道发送，或通过可选的中断输出管道发送。主机需依据设备提供的报告描述符对设备上传的报告进行识别。在此重复提醒：报告与报告描述符是两个概念

如果在报告描述符中使用了报告 ID 标签，则所有报告都包含一个单字节 ID 前缀。如果未使用报告 ID 标签，则所有值都将在单个报告中返回，并且前缀 ID 不包含在该报告中。

1. 标准项报告格式

报告格式由 8 位(8-bit)报告标识符和属于该报告的数据组成，如图 15.32 所示。

图 15.32　位图

报告 ID

报告 ID 字段的长度为 8 位。如果报告描述符中没有使用报告 ID 标签，则只有一份报告，并且报告 ID 字段被省略。

报告数据

数据字段是报告可变长度字段。

2. 数组项的报告格式

数组中的每个按钮都报告一个分配的编号，称为数组索引。这可以通过查找数组元素“用法”页和

“用法”转换为键码。当任何按钮在打开和关闭之间转换时，数组中当前关闭的按钮的整个索引列表将传输到主机。

由于每个数组字段中只能报告一个数组元素，因此修饰键应报告为位图数据(一组 1 位变量字段)。例如，CTRL，SHIFT，ALT 和 GUI 键等键构成了标准键盘报告中的 8 位修饰符字节。尽管这些用法代码在用法表中定义为 E0-E7，但用法不会作为数组数据发送。修饰符字节定义如表 15.72 所示。

表 15.72　修饰按键字节位结构

偏移	位长	说明
0	1 bit	左 ctrl
1	1 bit	左 shift
2	1 bit	左 alt
3	1 bit	左 GUI(如 windows 键)
4	1 bit	右 ctrl
5	1 bit	右 shift
6	1 bit	右 alt
7	I bit	右 GUI(如 windows 键)

下面的示例演示由键入 ALT+CTRL+DEL 的用户生成的报告，如表 15.73 所示。

表 15.73　修饰按键字节位结构

按键变化	修饰字节	数组字节
左 alt 按下	0000 0100	00
右 ctrl 按下	0001 0100	00
del 按下	0001 0100	4C
del 弹起	0001 0100	00
右 ctrl 弹起	0000 0100	00
左 alt 弹起	0000 0000	00

如果此设备有多个报告，则每个报告前面都会有其唯一的报告 ID，如图 15.33 所示。

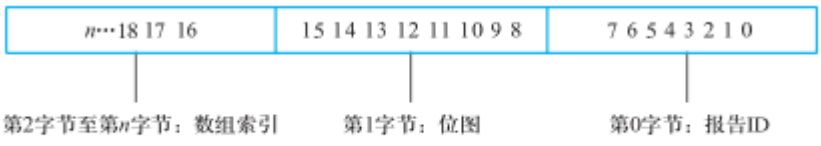


图 15.33

如果一组键或按钮不能互斥，则它们必须表示为位图或多个数组。例如，101 键键盘上的功能键有时用作修饰键,例如F1、A。在这种情况下,应在一个数组项中报告至少两个数组字段,即Report Count(2)。

3. 报告约束

以下约束适用于报告和报告处理程序：

- 报告中的项目字段不能超过 4 个字节。例如，一个 32 位项目必须从字节边界开始以满足此条件。
- 一次 USB 传输中只允许一份报告。
- 一份报告可能跨越一个或多个 USB 事务。例如，具有 10 字节报告的应用程序将跨越低速设备中的至少两个 USB 事务。
- 除了最长的超过端点的 wMaxPacketSize 的报告之外的所有报告都必须以一个短数据包终止。最长的报告不需要短数据包终止。
- 每个顶级集合必须是一个应用程序集合，并且报告不得跨越多个顶级集合。
- 如果顶级集合中有多个报告，则所有报告(最长的报告除外)都必须以短数据包结束。
- 报告始终是字节对齐的。如果需要，报告用位(0)填充，直到到达下一个字节边界。

4. 报告实例

代码清单 15-14 中的内容为某个设备的报告描述符

代码清单 15-14　报告描述符示例

```
    Usage Page (Generic Desktop),
    Usage (Mouse),
    Collection (Application),
        Usage(Pointe),
        Collection (Physical),
        Report ID(0A),          ;Make changes to report 0A Usage(x), usage (Y),
        Logical Minimum(-127), ;Report data values range from -127
        Logical Maximum(127),  ;to 127
        Report Size(8), Report Count(2),
        Input(Data, variable, Relative),
                                ;Add 2 bytes of position data (X&Y) to report 0A
        Logical Minimum (0),    ;Report data values range from -127
        Logical Maximum (1),    ;to 127
        Report count (3), Report Size (1),
        Usage Page (Button Page),
        Usage Minimum(1),
        Usage Maximum(3),
        Input (Data, Variable, Absolute),
                                ;Add 2 bits (Button1,2 & 3) to report 0A
        Report Size(5),
        Input (Constant),       ;Add 5 bits padding to byte align the report 0A
    End Collection,
End Collection
```

上述设备生成的报告结构如图 15.34 所示。

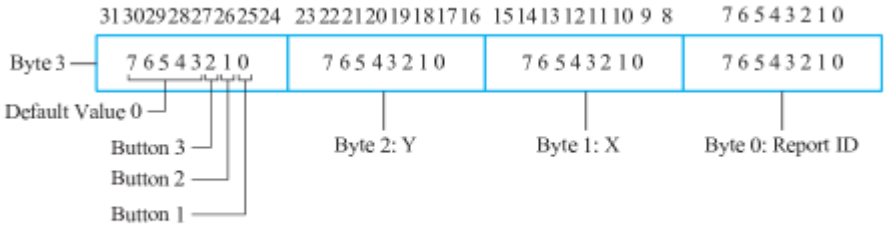


图 15.34　示例报告描述符设备生成的报告结构

表 15.74 使用带有集成鼠标功能的键盘来演示如何为只有一个接口的设备使用两个报告。

表 15.74　生成的报告

条目	用法	报告 ID
Collection(Application)	键盘	
Report ID(00)		
Input(变量，绝对)	修饰键	00
Output(变量，绝对)	LEDs	00
Input(数组，绝对)	主键	00
End Collection		
Collection(Application)	鼠标	
Report ID(01)		
Collection(Physical)	指针	
Input(变量，相对)	X,Y 坐标	01
Input(变量，绝对)	Button	01
End Collection		
End Collection		

注意：仅存在输入、输出和特征项(非集合项)报告中的数据。此示例演示了多个报告，但是此接口不能用于引导设备(对键盘和鼠标设备要使用单独的接口)。

S.3.5　HID 协议小结

USB HID 设备主要基于两种协议：报告协议(report protocol)和引导协议(boot protocol)。报告是从设备发送到主机的数据结构，也可以从主机发送到设备。当设备向主机发送报告时，通常包含状态变化信息，例如按键，鼠标移动等。当主机向设备发送报告时，通常包含用于配置设备的命令(set_report 请求)，例如在键盘上设置 LED。这个协议依赖于标准的 USB 框架。USB HID 设备使用中断传输进行通信，因为设备并不是一直在传输数据，但当设备传输数据时，主机需要软件有快速的响应。中断传输时传输的数据量通常很小。报告通常有两种类型，具体取决于协议类型。HID 报告协议基于“项目(item)”的概念，其数据结构在报告描述符中定义。引导协议要简单得多，并且遵循鼠标和键盘的标准结构。依据本次实验内容，此处将仅讨论引导协议。

1. 主机检测 HID 设备

HID 设备在其设备描述符中的类(bInterfaceClass)/子类(bInterfaceSubclass)值都为零，其接口描述符中的类(bInterfaceClass)与子类(bInterfaceSubclass)的值才有效。注意：不能手动请求接口描

述符，并且必须与配置和端点描述符一起获取。接口描述符中标识 HID 设备的类(bInterfaceClass)值为 3。接口描述符中的子类(bInterfaceSubClass)值可以是 1，表示设备支持引导协议，也可以是 0，表示设备仅支持报告协议。接口描述符中的协议(bInterfaceProtocol)字段也确定它是鼠标还是键盘。具体来说，1 表示 HID 设备是键盘，2 表示 HID 设备是鼠标。

2. “GET_REPORT”请求

该软件可以使用控制端点和常规 SETUP 数据包从 USB 设备请求报告。SETUP 数据包的请求类型将包含 0xA1，“GET_REPORT”的请求代码为 1，SETUP 数据包的“wValue”字段将包含 0x0100，以请求 ID 为 0 的输入数据包，并且该字长将会是数据阶段所期望接收的字长。SETUP 数据包应发送到设备端点 0(控制端点)。对于键盘，中断传输阶段通常为 8 个字节，而对于鼠标，中断传输阶段具有标准格式的前 3 个字节，而其余的可能由设备特定的功能使用。建议仅以这种方式接收报告以测试设备初始化是否成功完成等，并且不推荐使用“GET_REPORT”请求来轮询 HID 设备以进行更改，因为 SETUP 和 STATUS stage 浪费了太多时间。相反，软件应使用中断传输，使用中断 IN 端点轮询 HID。

3. 中断端点

通常建议 HID 使用中断传输向软件报告，并且软件通常应避免上述“GET_REPORT”请求。驱动程序软件应请求 HID 设备的配置描述符。HID 设备必须支持至少一种配置。然后软件应扫描端点描述符，搜索指示“中断输入”类型的描述符，该端点是使用中断传输将设备数据发送到主机的端点。软件应保存端点的 4 位 ID，以及端点的 8 位间隔。间隔值以毫秒(ms)为单位对时间进行编码，软件应在该时间空间内轮询一次报告包。例如，如果间隔值为 8，则软件应每 8 毫秒向设备请求一次报告。如果软件过早请求报告，例如在 6ms 之后，设备可能会发送与之前相同的数据包，或者它可能不发送任何内容，而是返回 NAK。如果软件在时间跨度之后请求报告，例如 9 毫秒，则设备将发送新数据包。软件使用此描述的方法不断轮询 USB HID 设备。

4. USB 键盘

USB 键盘使用报告与软件通信，就像其他 HID 设备一样。通过接口描述符中的类字段(bInterfaceClass)3 和协议字段(bInterfaceProtocol)来检测 USB 键盘。

1)报告格式

该报告必须由软件每隔固定时间使用一次中断传输请求，并且此间隔应在 USB 键盘的端点 IN 触发的中断中定义。USB 键盘报告的大小可能高达 8 个字节，但这些字节不一定都被使用，如果只使用前三个或四个字节来实现正确的键码传输是可行的。此处描述键盘的完整报告机制。表 15.75 定义的报告数据结构仅适用于引导协议。

表 15.75　USB 键盘报告结构

偏移	大小	说明
0	1 字节	修饰按键状态
1	1 字节	保留
2	1 字节	普通按键按下#1
3	1 字节	普通按键按下#2
4	1 字节	普通按键按下#3
5	1 字节	普通按键按下#4
6	1 字节	普通按键按下#5

偏移	大小	说明
7	1 字节	普通按键按下#6

修饰按键状态：该字节是一个位域，其中每个位对应于一个特定的修饰键。当某位置 1 时，则表示对应修饰键被按下。该字节的位结构见表 15.76。

表 15.76 修饰按键字节位结构

位	位长	说明
0	1 bit	左 ctrl
1	1 bit	左 shift
2	1 bit	左 alt
3	1 bit	左 GUI(如 windows 键)
4	1 bit	右 ctal
5	1 bit	右 shift
6	1 bit	右 alt
7	1 bit	右 GUI(如 windows 键)

按键字段：一个键盘报告数据结构最多可表示 6 个普通按键按下状态。按键值是无符号的 8 位二进制编码，该编码为扫描码(HID 码)，不是 ASCII 码。

2)按键机制

USB 键盘在按下或释放键时发送中断。当用户按下一个键时，中断会在其中一个按键字段中带有一个扫描码值。当一个键被释放时，对应的按键字段在下一个数据包中返回零。为了更清楚地说明这一点并说明为什么有多个普通按键扫描码字段，让我们看以下示例。假设用户按下“A”键，即扫描码 0x04。返回的中断数据包为：

```
00 00 04 00  00 00 00 00
```

注意修饰键为零，因为用户没有按下修饰按键。按照 USB HID 规范的建议，保留字段也为零。第一个普通按键字段包含 0x04，对应于“A”键。现在，让我们假设用户松开“A”键。发送的数据包如下所示：

```
00 00 00 00  00 00 00 00
```

现在，假设用户按下“A”键，然后按下“B”键(扫描码 0x05)而不松开“A”键，发送的数据包如下：

```
00 00 04 05  00 00 00 00
```

注意一个中断数据包如何能够同时传输两个按键。现在让我们假设用户按下“C”键(扫描码 0x06)而不放开“A”或“B”键，发送的数据包如下：

```
00 00 04 05  06 00 00 00
```

显然，USB 键盘按照首先按下的顺序返回扫描码。因此，如果第一个普通按键字段为零，则没有按键被按下。如果它不为零，主机软件可以检查下一个字段，以查看是否也按下了另一个普通按键。

修饰键的概念十分简单，我们假设用户使用“X”键(扫描码 0x1B)按下左 shift 键。发送的数据包

如下：

```
02 00 1B 00  00 00 00 00
```

注意，修饰符字段的第 1 位(值 0x02)已设置，表示正在按下左 shift 键。

还有一个“幻象状态”，可以将其视为溢出。一个 USB 键盘数据包在一次传输中最多可以指示 6 个按键，但是让我们想象一下有人一次按下了 6 个以上的按键。键盘将进入幻象状态，其中所有报告的键都是无效的扫描码 0x01。但是，仍会报告修改键。例如：有 8 个键(或任何大于 6 的随机数)被按下，同时右 shift 键也被按下。发送的数据包如下所示：

```
20 00 01 01  01 01 01 01
```

修饰键字段数据正常，但普通按键字段都返回了幻像状态。除了幻象条件外，还有其他特殊的扫描码：0x00 表示没有扫描码并且没有按键被按下，0x01 表示我们刚刚解释的幻象状态,0x02 表示键盘的自检失败，0x03 表示发生了未定义的错误。从 0x04 开始，扫描码有效并对应于“真实”按键。

3)LED 灯的设置

LED 灯也是在软件中处理的，根据硬件，NumLock、CapsLock 和 ScrollLock 是发送正常扫描码的正常键。当按下这些键之一时，驱动程序负责操作 LED 灯。

为了设置 LED 灯，驱动程序使用标准 USB 设置事务向设备发送一个 SET_REPORT 请求,其中包含一个字节的数据阶段。设置包的请求类型应为 0x21，SET_REPORT 的请求代码为 0x09。SETUP 数据包的 value 字段在低字节中包含报告 ID，其值为 0。高字节包含报告类型，其值为 0x02 表示输出报告，或从软件发送到硬件的报告。wIndexL 字段应包含 USB 键盘的接口编号，即接口描述符中的编号，表明该设备是 USB 键盘。如果硬件支持中断 OUT 端点，只需将 1 字节数据阶段传输到中断 OUT 端点，而无需额外的 SETUP 阶段开销。如果硬件支持中断 OUT 端点，则应尽可能避免使用控制端点，因为轮询中断 OUT 端点更快。

主机向 OUT 端点发送的设置 LED 灯的 1 个字节数据结构如表 15.77 所示。

表 15.77　1 个字节数据的数据结构

位	位长	说明
0	1	Num Lock 指示灯
1	1	Caps Lock 指示灯
2	1	Scroll Lock 指示灯
3	1	Compose
4	1	Kana
5	3	保留

5.USB 鼠标

USB 鼠标与任何其他 HID 设备一样，使用报告与软件通信，报告通过中断端点发送，或者可以通过“GET_REPORT”请求手动请求。USB 鼠标在接口描述符中的协议字段(bInterfaceProtocol)为 2。

1)报告格式

主机必须每隔固定时间使用一次中断传输请求此报告。USB 鼠标设备中，报告仅有 3 个字节，其依次从低到高进行排列，表 15.78 给出了支持引导协议的 USB 鼠标的报告数据结构。

表 15.78　USB 鼠标的报告数据结构

偏移	大小/字节	描述
0	1	按键状态
1	1	X 方向上的位移(相对值)
2	1	Y 方向上的位移(相对值)

按键状态：该字段的字节位结构如表 15.79 所示。

表 15.79 按键状态字节位结构

偏移位(第几个比特位)	比特长度/bit	描述
0	1	当其值为 1 的时候表示左键按下
1	1	当其值为 1 的时候表示右键按下
2	1	当其值为 1 的时候表示中键按下

对于第二个和第三个字节则表示 X 和 Y 方向上的位移，该表示方法遵循右手坐标系，右移为正，下移为正。其 256 个数据的分配为：值小于 127 为正，大于 127 视为负值(-1 到-127 的补码表示)。

S.4　人机交互设备程序设计

以下内容依据 STC 官方提供的源码以及 HID 协议规范对程序设计进行讲解，本次程序设计将实现 HID 键盘设备。

S.4.1　设计分层

此次程序设计由于涉及不同的协议规范，因此需要对设计进行分层，以便日后扩展和修改。此次程序设计依据不同协议规范分为以下几层，由于厂商没有定义特定的请求，故厂商请求层部分将略去。

(1)USB 层

(2)USB 标准请求层

(3)HID 特定类请求层

(4)定时器层

1.USB 层

该层对应程序设计中的 usb.c 文件，该层存放 USB 初始化函数以及 USB 中断处理函数，USB 中断处理函数中提供了 USB 所有相关中断的处理函数的入口，其中代码相同部分的分析详见 USB2.0 程序设计实现。

下面讲解 USB 层的不同处。

1)USB HID 键盘的 IN 中断原理

在中断传输前，USB 设备已完成了 SETUP 阶段的内容。在中断传输中，主机会定期轮询一次设备的报告，从而触发设备端点 1IN 的中断。如果触发的是端点 1IN 的中断，则程序会跳转至 usb_in_ep1()函数中，而 usb_in_ep1()就是端点 1IN 的中断处理函数。该中断用于传输键盘码值。

端点 1IN 中断数的具体内容为:

①硬件状态设置

将 INDEX 寄存器定位到端点 1；读取 INCSR1 寄存器的内容，依据 INCSR1 寄存器的 INSTSTL 位决定是否复位数据切换位到“0”，如果 INSTSTL 位为 1 表示 STALL 信号发送完成，可以复位数据切换位到“0”；如果对主机的轮询回复的是 NACK 数据包，则清零 INCSR1 寄存器的所有位。

②)软件状态设置

设置 UsbInBusy 为 0(IN 端点空闲，可以传输报告数据)

具体写法如代码清单 15-15 所示。

代码清单 15-15　USB 端点 1IN 中断处理函数的 C 语言代码

```
/*************************USB 端点 1IN 中断处理函数*************************/
void usb_in_ep1()
{
    BYTE csr;
    usb_write_reg(INDEX, 1);
    csr = usb_read_reg(INCSR1);
    if (csr & INSTSTL)
    {
```

```
        usb_write_reg(INCSR1,INCLRDT);
    }
    if (csr & INUNDRUN)
    {
        usb_write_reg(INCSR1,0);
    }
    UsbInBusy = 0;
}
```

2)USB HID 键盘的 OUT 中断原理

与键盘的 IN 中断原理一致，如果触发的是端点 1OUT 的中断，则程序会跳转至 usb_out_ep1()函数中，而 usb_out_ep1()就是端点 1OUT 的中断处理函数。不同的是此中断用于设置键盘状态，例如：当按下 Capslk 按键后，主机会发一个 OUT 包，当 OUT 包接收完毕后，端点 1OUT 中断函数会调用 usb_class_out()函数对开发板上对应的 LED 灯进行设置。

具体写法如代码清单 15-16 所示。

代码清单 15-16　USB 端点 1OUT 中断处理函数的 C 语言代码

```
/************************USB 端点 1OUT 中断处理函数************************/
void usb_out_ep1()
{
    BYTE csr;

    usb_write_reg(INDEX,1);
    esr-usb_read_reg(OUTCSR1);
    if (csr & OUTSTSTL)
    {
        usb_write_reg(OUTCSR1,OUTCLRDT);
    }
    if (csr & OUTOPRDY)
    {
        usb_class_out();            //调用 class_out()函数设置灯的状态
    }
}
```

2.USB 标准请求层

该层对应程序设计中的 usb_req_std.c 文件，该层存放设备对 USB 主机 11 个标准请求的响应函数，具体详见第 3 章。

不同点，若设备接收主机的 Get Descriptor 请求，设备除了需要依据 wVlaue 状态字发送标准描述符外，还需要发送 HID 类设备特有的描述符。

3. HID 特定类请求层

该层对应程序设计中的 usb_req_class.c 文件，该层除了存放 HID 特定类请求处理函数外，还存放了发送键盘报告的函数 usb_class_in()，读取主机报告函数 usb_class_out()，以及键盘码值读取函数 scan_key()。

1)类请求函数的参考写法

如表 15.80 类特定请求依据 USB 标准设备请求的数据结构的 bRequest 值来进入具体的请求函数。

表 15.80 类特定请求

请求	bRequest 值	功能
GET_REPORT	0x01	请求允许主机通过控制管道接收报告
SET_REPORT	0x09	请求允许主机向设备发送报告，以设置输入、输出或功能控件的状态
GET_IDLE	0x02	请求读取特定输入报告的当前空闲率
SET_IDLE	0x0A	请求屏蔽中断输入管道上的特定报告，直到新事件发生或经过指定的时间间隔
GET_PROTOCOL	0x03	请求读取当前已激活的协议(例如引导协议或报告协议)
SET_PROTOCOL	0x0B	切换引导协议或报告协议

为了标准起见，本书的类请求函数同样分为三个步骤：1.特殊情况处理；2.依据状态发包；3.结束状态，即进行数据操作(如发送数据，或者接收数据，或者不进行数据操作)。此部分的代码请参考工程 HID 协议范例文件 usb_req_class.c 内容。

① GET_REPORT

表 15.81 GET_REPORT 请求内容

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
0xA1	GET_REPORT	报告类型和报告 ID	接口号	报告的长度	报告

第 1 步：特殊情况的处理，与 GET_STATUS 请求的设计思路一致，依旧是特殊情况处理。如果 USB 标准设备请求数据结构不满足表 15.81 所有条件，则挂起。

第 2 步：UsbBuffer 的数据装入端点 0 的数据区，并设置大小为 UsbBuffer 的大小为 wLength 大小。UsbBuffer 的数据用于后续 HID 键盘设备的状态设置。

第 3 步：进入标准请求结束状态(进行数据过程)，例如：这时在步骤 2 中软件已经依据 USB 协议内容将 UsbBuffer 数据包装入了数据缓冲区，然后软件再通过调用 usb_setup_in()，完成数据包的发送。

注意：usb_setup_in()的内容是：先将控制端点 0 设置为 DATAIN 状态，控制寄存器 CSR0 的 SPORDY 写 1 以清零 OPRDY，随后调用 usb_ctrl_in()进行数据发送，即使用控制 IN 类型向主机发送数据包。

② SET_REPORT(见表 15.82)

表 15.82 SET_REPORT 请求内容

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
0x21	SET_REPORT	报告类型和报告 ID	接口号	报告的长度	报告

第 1 步:特殊情况的处理，与 GET_STATUS 请求的设计思路一致，依旧是特殊情况处理。

如果 USB 标准设备请求数据结构不满足表 15.82 所有条件，则挂起。

第 2 步:将 UsbBufer 数据装人端点 0 的数据区，并设置大小为 UsbBuffer 的大小为 wLengh 大小。

第 3 步:进入标准请求结束状态(进行数据过程)，端点 0 状态设置为 OUT 状态，最后将 CSRO 寄存器的 SOPRRDY 置 1，表示处理完成从端点 0 接收到的数据包。

③ GET_IDLE(见表 15.83)

表 15.83　GET_IDLE 请求内容

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
0xA1	GET_IDLE	0 和报告 ID	接口号	1	bHidIdle

第 1 步：特殊情况的处理，与 CET_STATUS 请求的设计思路一致，依旧是特殊情况处理。如果 USB 标准设备请求数据结构不满足表 15.83 所有条件，则挂起。

第 2 步：将数据为 bHidIdle 所指向的数据区域装人端点 0 的数据缓冲区，并设置大小为数据缓冲区的大小为 1 字节大小。UsbBuffer 的数据用于后续 HID 键盘设备的状态设置。

第 3 步：进入标准请求结束状态(进行数据过程)，例如：这时在步骤 2 中软件已经依据 USB 协议内容将 bHidIdle 所指向的数据装入了端点 0 的数据缓冲区，然后软件再通过调用 usb_setup_in()，完成数据包的发送。

注意:usb_setup_in()的内容是：先将控制端点 0 设置为 DATAIN 状态，控制寄存器 CSRO 的 SPORDY 写 1 以清零 OPRDY，随后调用 usb_ctrl_in()进行数据发送，即使用控制 IN 类型向主机发送数据包。

④ SET_IDLE(见表 15.84)

表 15.84 SET_IDLE 请求内容

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
0x21	SET_IDLE	持续时间和报告 ID	接口号	0	无

第 1 步：特殊情况的处理，与 GET_STATUS 请求的设计思路一致，依旧是特殊情况处理。如果 USB 标准设备请求数据结构不满足表 15.84 所有条件，则挂起。

第 2 步：bHidIdle 的状态设置为 wValue 的值。

第 3 步:进入请求结束状态:端点 0 设置成空闲状态，寄存器 CSRO 的 SPORDY 位以及 DATEND 位置位，表示处理完成从端点 0 接收到的数据包，且数据结束。

⑤ GET_PROTOCOL

此次实验不涉及该请求，故不对内容进行编写，若进入该函数，则调用 USB 挂起函数以替代。

⑥ SET_PROTOCOL

此次实验不涉及该请求，故不对内容进行编写，若进入该函数，则调用 USB 挂起函数以替代。

2)发送键盘报告的函数

先判断 IN 端点是否空闲且矩阵按键是否已完成读入，若已完成则进行译码；但需要注意的是键盘上传的报告数据并不是 ASCII 码，而是键盘的扫描码即 HID 码，因此我们需要将矩阵按键译码为 HID 码，这样主机才正确识别到设备上传的报告值，此处我们仅使用报告的第三个字节即 key[2]来上传键盘 HID 码。

发送阶段：先设置 UsbInBusy 状态字为 1(IN 端点忙碌，正在传输报告数据)。我们需要操作 INDEX 寄存器指向端点 1，后我们将待发送的报告数据以字节为单位发送出去即可。

3)读取主机报告函数

该函数使用 UsbBuffer 缓冲区从 FIFO0 中提取出主机向设备发送的控制数据，然后使用该数据对实验箱上的 LED 灯进行设置。

4)键盘码值读取函数

此处详细见矩阵按键程序设计部分，此处我们只需设计一个可以识别矩阵按键的函数即可，在我们调用过该函数后可以通过 bKeyCode 变量读出按键值。

4. 定时器层

该层对应于程序设计中的 timer.c 文件，定时器层中存放了定时器初始化函数，定时器中断处理函数。

定时器层的作用是提供一个固定的时间间隔，Ai8051U 使用该固定时间间隔调用 scan_key()函数对实验箱上的按键进行扫描，并使用 bKeyCode 变量读出按键值。

S.4.2 描述符数据结构

1.设备描述符

设备描述符 bDeviceClass 字段，bDeviceSubClass 以及 bDeviceProtocol 字段值均固定为 00H，其余字段值依据 USB 规范以及厂商规范进行定义。

2.配置描述符

依据 USB 规范，USB 主机发送 Get Descriptor 请求获取设备的描述符数据结构，其中配置描述符、接口描述符、HID 描述符、端点描述符是合并为一个数据结构发送给主机的。

1)配置描述符

配置描述符依据 USB 规范定义方式进行定义。由于仅实现键盘一个功能故 bNumInterfaces 为 1。

2)接口描述符

依据 HID 协议，接口描述符的 bInterfaceClass 字段值定义为 03H(HID)，bInterfaceSubClass 字段值定义为 01H(Boot)，bInterfaceProtocol 字段值定义 01H(Keyboard)。

3)HID 描述符

各字段值请参照 HID 描述符一节进行定义。

4)端点描述符

由于端点 1 既要使用 IN 端点又要使用 OUT 端点，因此需要用两个数据结构分别定义端点 1。端点 1 的数据传输模式依据协议需要定义为中断类型的端点。因此端点 1IN 的 bEndpointAddress 字段值定义为 81H，端点 1OUT 的 bEndpointAddress 字段值定义为 01H，两个端点描述符结构中 bmAttributes 字段值都为 03H，其余字段依据 USB 规范定义方式进行定义。

3. HID 报告描述符

HID 报告描述符中所使用的项均为短项目。此部分建议读者配合 HID 报告描述一节对其内容进行理解：

例如 USAGE_PAGE(Generic Desktop)项，该项属于短项目中的全局项，通过全局项的一字节前缀的表格可以知道，该项的第 0 个字节为 0000 01nn(b)其中 nn 为后面跟随的数据字节的数量，故 nn 为 01，该项的第 1 个字节则是确定 USAGE_PACE 项的类型，该部分需要对照 USB 官方文档 HID usage table1 进行查看。

4.厂商描述符

此处依据厂商标准对该描述符进行定义。

S.4.3 例程使用说明

下载程序至实验箱，在主机上打开文件编辑器，定位好光标，分别按下实验箱上的矩阵按键，光标输出内容；按下主机的大写锁定键，发现实验箱上对应的指示灯亮起。

拓展：请依据 HID 协议内容，使用实验箱模拟一个 HID 鼠标设备。

S.5 通信设备类原理

本章依据程序设计代码，仅以抽象控制为例对 CDC 协议规范进行部分讲解，完整的 CDC 协议参阅文档 usbcdc11.pdf(universal serial bus class definitions for communication devices)。

S.5.1 USB 通信类设备简介

通信类设备(communication device class)定义了三个类别：通信设备类、通信接口类和数据接口类。通信设备类是一个设备级别的定义，被主机用来正确识别一个可能呈现几种不同类型接口的通信设备。通信接口类定义了一个通用的机制，可以用来启用通用串行总线(USB)上的所有类型的通信服务。数据接口类定义了一种通用机制，当数据不符合其他类别的要求时，可以在 USB 上启用批量或同步传输。

CDC 设备目前主要用于以下场景：

- 电信设备：模拟调制解调器、ISDN 终端适配器、数字电话和模拟电话
- 网络设备：ADSL 调制解调器、电缆调制解调器、10BASE-T 以太网适配器/集线器，以及“以太网”交叉电缆。

1. 通信类设备功能特点

本节介绍了通信设备类、通信接口类和数据接口类的功能特点。

(1)设备组织。

- 端点需求。
- 从端点构筑接口。
- 从各种接口构建配置，其中一些是由其他类规范定义的。
- 识别配置内构成功能单元的接口组，并为每个联合分配一个主接口。

(2)设备操作

尽管 CDC 规范同时定义了通信接口类和数据接口类，但它们是两个不同的类。所有通信设备应具有一个使用通信类管理设备的接口，并可选择通过使用通信设备类代码将自己指定为通信设备。此外，设备还有一些用于实际数据传输的其他接口。当数据与任何其他类型的类(如音频)的结构或使用模式不匹配时，数据接口类确定了数据传输接口。

1)设备组织

一个通信设备有三个基本职责：

- 设备管理
- 操作管理
- 数据传输

设备应使用通信类接口来进行设备管理，其也可选择通信类接口用于呼叫管理。数据流是根据正在传输的数据的 USB 类来定义的。如果没有合适的 USB 类，那么设计者可以使用本 CDC 规范中定义的数据类来模拟数据流。

设备管理指的是控制和配置设备运行状态的请求和通知，以及通知主机在设备上发生的事件。

呼叫管理指的是一个负责设置和解除呼叫的过程。这个过程也控制呼叫的操作参数。术语“呼叫”，因此也是“呼叫管理”，描述的是指比负责物理连接的进程更高级别的呼叫控制进程。

数据传输除了使用通信类接口外，还使用接口来完成。这些接口可以使用任意定义的 USB 类，也可以是供应商特定的类。

2)通信设备管理

通信设备的设备管理有两个层次。最基本的设备管理形式是在端点 0 上进行的控制传输。设备管理还需要在更高的层次上进行，这是对通信设备的特殊要求。一个例子是配置特定国家的细节，以正确配置电话服务。

为了允许在通信设备层面进行设备管理，应在构成设备功能单元的所有接口之间建立一个联合。功能描述符用于定义构成设备内功能单元的接口组。随着多通道设备的日益普及，一类新设备可能需要暴露多个设备管理接口，以便在通信设备层面进行设备管理。这将允许对多通道的单独控制，例如 ISDN 设备。在这种情况下，联合将在提供呼叫控制的通信类接口和它目前管理的各种接口之间。

2. 设备操作

通信类设备以另一类定义的形式向主机呈现数据，如音频，数据或 HID 接口。为了允许适当的类驱动程序管理该数据，主机将提供一个或多个接口，如为该类指定的那样。所需的接口可能会根据通信会话期间由用户或网络发起的事件而改变。例如，从仅数据呼叫到数据和语音呼叫的转换。

为了使主机能够正确处理使用多个接口创建单个功能的情况，设备可以选择使用通信设备类代码在设备级别标识自己。这允许主机在需要时加载任何特殊驱动程序以将多个接口正确配置为主机中的单个功能。

注意：在设备未选择使用通信设备类代码在设备级别标识自身的情况下，设备应采用 USB 通用类功能机制，将设备上的多个接口与主机中的单个驱动程序相关联。

设备的静态特性，例如物理连接，是根据 USB 设备、接口和端点描述符来描述的。在物理接口上搬移的数据本质上是动态的，导致接口的特性随着数据需求的变化而变化。这些动态变化是根据通过通信类接口在设备和主机之间传输的消息来定义的。设备可以使用标准或专有机制来通知其主机软件何时可以使用接口以及数据格式。主机软件也可以使用相同的机制来检索有关接口数据格式的信息，并在有多个可用的接口时选择一种数据格式。

3. 接口定义

USBCDC 规范中描述了两类接口：通信类接口和数据类接口。通信类接口是一个管理接口，所有通信类设备都需要它。数据类接口可用于传输结构和用法未由类定义的数据，例如音频数据。可以使用关联的通信类接口来识别通过该接口传输的数据格式。

1)通信类接口

通信类接口用于设备管理，也可以选择呼叫管理。设备管理包括管理设备的运行状态的请求，设备响应和事件通知。呼叫管理包括设置和关闭呼叫的请求，以及对其操作参数的管理。

通信类定义了一个通信类接口，由一个管理元素和可选的通知元素组成。管理元素配置和控制设备，由端点 0 组成。通知元素将事件传送到主机，在大多数情况下，由一个中断端点组成。

通知元素通过一个中断或批量端点传递信息，使用标准化的格式。消息的格式是一个标准化的 8 字节的标头，后面是一个可变长度的数据域。标头确定了通知的种类，以及与通知相关的接口；标头还确定了消息中数据域的长度。

通信类接口应通过提供一个管理元素来提供设备管理，该接口可通过提供一个通知元素来提供主机通知。对于一个完整的通信类接口来说，管理元素是必要的。管理元素也符合 USB 规范中对设备的要求。呼叫管理在通信类接口中提供，也可选择在数据接口上进行多路复用。下面的配置描述了设备如何在使用和不使用通信类接口的情况下提供呼叫管理：

- 设备不在通信类接口上提供任何呼叫管理，只由一个管理元素(端点 0)组成。在这种情况下，通信类接口被最小化，只通过一个管理元素(端点 0)提供设备管理。其中多通道控制模型和 CAPI 控制模型使用了该配置。

- 设备不提供呼叫管理的内部实现，只接受来自主机的最小呼叫管理命令集。在这种情况下，管理元素和通知元素都代表通信类接口。直接线路控制模型使用了该配置。
- 设备通过数据类接口提供呼叫管理的内部实现，而不是通信类接口。在这种情况下，通信类接口也被最小化，只通过一个管理元素(端点 0)提供设备管理。这种配置最接近于抽象控制模式，其中命令和数据通过数据类接口进行复用。从数据模式激活命令模式是通过 Heatherington Escape 序列或 TIES 方法完成的。关于抽象控制模式的更多信息，请参见第 2 节，“抽象控制模型”。
- 设备提供呼叫管理的内部实现，主机通过通信类接口访问。在这种情况下，通信类接口同时执行呼叫和设备管理，并由一个管理元素(端点 0)和一个通知元素(通常是一个中断端点)组成。管理元素将传输呼叫管理和设备管理命令。通知元素将传输从设备到主机的异步事件信息，如可用响应的通知，然后提示主机通过管理元素检索响应。这与抽象控制模型相对应。关于抽象控制模型的更多信息，见第 2 节，“抽象控制模型”。

2)数据类接口

数据类将数据接口定义为类类型为数据类的接口。通信设备上的已定义的 USB 数据传输并不依靠数据类接口。相反，数据接口用于传输和/或接收未由其他类定义的数据。这些数据可能是：

- 来自通信线路的某种形式的原始数据。
- 传统调制解调器数据。
- 使用专有的数据格式。

此时，主机软件和设备需要通过其他接口(例如通信类接口)相互通信以确定要使用的适当格式。随着更复杂的通信类设备被定义，可能有必要定义一种方法来描述数据类接口中使用的协议。数据类接口的屈性如下：

- 接口描述符使用数据类码值作为其类别类型。这是唯一要使用数据类码值的地方。
- 数据形式是字节流。数据类接口一般不定义数据流的格式，除非使用协议数据包装器。
- 如果接口包含同步的端点，在这些端点上，数据被认为是同步的。

同步管道用于满足以下条件的数据：

- 恒定比特率。
- 需要低延时的实时通信。

通常，同步端点可用于将来自网络的原始信息(采样的或直接的)发送到主机以进行进一步处理和解释。例如，便宜的 ISDN TA 可以使用同步管道从网络中传输原始采样位。在这种情况下，主机系统将负责构成 ISDN 连接的不同网络协议。这种类型的接口只能在音频类接口不提供必要的定义或控制的情况下使用。

当主机激活接口或设备请求激活接口时，通过通信类接口的管理元素上的消息传递来指定要使用的媒体的类型和格式。管道的带宽由端点描述符定义，可以通过选择适当带宽的备用接口来更故。

3)协议数据包装器

为了支持设备中的嵌入式高层协议，主机和设备之间的数据和命令必须保留其顺序。这就保证了一个设计在实时操作系统中运行的协议栈，可以被分成两部分在不同的设备中运行。因此，一个协议的命令和数据必须使用一个包装器复用到同一个接口上；这个包装器还具有向协议栈的任意一层发送数据的功能。每个协议都规定了如何定义协议特定的命令和数据字段，以穿过其上层接口边缘。

在建立数据的协议时，主机和设备就包装器的功能达成一致。主机软件和设备需要通过其他接口(如通信类接口)相互通信以确定协议。如果没有建立协议，则不使用包装器。如果建立的协议可以使用包装器，那么使用包装器是可选的；如果协议要求使用包装器，那么就必须使用包装器。

为了支持通信设备上不同类型的协议栈，数据包装器头定义了两种一般形式，如表 15.85 所示。两

种形式的结构都是一样的，唯一的区别是源协议 ID 的使用。如果不需要或不知道源协议，那么偏移 3，bSrcProtocol 被设置为 00h。

数据包装器头的第二种形式允许在需要源协议和目标协议的情况下，为结构更合理的协议栈提供源协议和目标协议。

除了数据是字节数据的一般要求外，这两种数据包装器形式对数据格式都没有限制。在任何情况下，如果源协议是不需要的或未知的，就会使用 00h 的源协议 ID(bSrcProtocol)。

注意：不建议在同步管道上使用协议数据封装器，因为同步管道的不可靠性质可能导致数据丢失。

表 15.85　数据类协议封装布局

偏移	域	大小	值	描述
0		2	数字	包装器的大小(以字节为单位)
2	bDstProtocol	1	协议	目标协议 ID
3	bSrcProtocol	1	协议	源协议 ID
4	BData0	1	数字	第一个数据字节
……	……	……	……	
N+3	BDataN-1	1	数字	第 N 个数据字节

4. 端点需求

以下部分描述了通信类或数据类接口中端点的要求。

1)通信类端点需求

通信类接口需要一个端点，即管理元素。其还可以添加一个额外的端点，即通知元素。对于所有 USB 标准类和 USB 通信类的请求，管理元素使用默认端点(端点 0，EP0)。通知元素通常使用中断端点(端点 x，EPx。x 为具体使用的端点号，不能为 0)。

2)数据类端点需求

属于数据类接口的端点，其类型仅限于同步端点或批量端点，并且端点应成对存在，即：一个端点号对应存在一个输入端点和一个输出端点。

5. 设备模型

特定的 USB 通信设备配置是由前几节所述的接口和其他类规范所描述的接口构成的。所有的通信设备都由一个通信类接口加上 0 个或更多的其他数据传输接口组成，遵守一些其他的 USB 类要求或作为供应商的特定接口来实现。例如，以下描述符适合于一个通信设备：

- 设备描述符包含通信设备类的类代码，若设备描述符包含一个 00h 的类代码，这表明主机应该查看接口以确定如何使用该设备。
- 一个具有通信类代码的接口描述符，它包含一个管理元素和可选的一个通知元素。
- 零个或多个其他接口，有各种类型的类代码，如音频、数据等。

设备模型被分为几个类别，依据程序设计部分的内容，本文将仅介绍 POTS 模型的抽象模型。随着 CDC 规范的发展，其他模型将被陆续添加进来。模型描述了一种设备类型和构成它的接口。控制模型描述了正在使用的通信类接口的类型，并为该接口分配了一个 SubClass 代码。一个控制模型可以用于几个设备模型中，其中设备控制和呼叫管理的方法是相似的。

S.5.2　USB 通信类设备抽象模型介绍

1. USB POTS 模型

在 POTS 线路上使用的 USB 电话设备有几种类型的接口可以提交给主机。这些不同接口的安排和使用取决于 POTS 电话设备的类型和用于建立设备的基本模型。

各种型号的电话设备之间的区别可以根据设备在将模拟信号呈现给主机之前对其进行的处理量来划分。为了帮助说明如何将不同类型的接口放在一起建立一个 USB POTS 电话设备，在下面将介绍抽象模型。

注意：大部分情况下，数据类接口可能不会被用来向主机提供数据。在 USB 设备的构造具有最小的智能时，需要一些模拟类特定的接口控制代码。

2. 抽象控制模型

通过抽象控制模型，USB 设备可以理解标准 V.25ter(AT)命令。该设备包含一个数据泵和微控制器，用于处理 AT 命令和继电器控制。该设备同时使用数据类接口和通信类接口。有关这两种接口的使用说明，请参照图 15.35。设备有时也可以使用其他类接口，如使用音频类接口来实现扬声器中的音频功能。

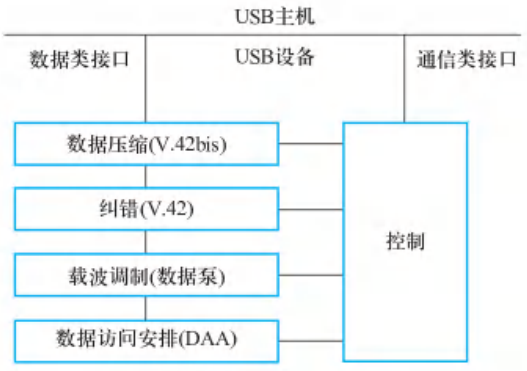


图 15.35　抽象控制模型

抽象控制模型类型的通信类接口将至少包含两个管道;一个用于实现管理元素，另一个用于实现通知元素。此外，该设备可以使用两个管道来实现传输未指定数据的通道，通常是通过数据类接口。

对于 POTS 线路控制，抽象控制模型应支持嵌入在数据流中的 V.25ter 命令或沿着通信类接口发送的 V.25ter 命令。当 V.25ter 命令在数据流中多路复用时，Heatherington 转义序列或 TIES 方法将定义唯一支持的转义序列。

纠错和数据压缩可以在主机上实现，而不必在设备上实现。这种类型的设备不同于直接线路控制模型，因为来自 USB 设备的数据通过本地类定义的接口而不是供应商特定的数据泵接口呈现给主机。此外，V.25ter 命令用于控制 POTS 线路接口。V.80 定义了主机可以控制 DCE 数据流来完成这一任务的一种方法，但也有一些专有的方法。

3. 抽象控制模型串行模拟

抽象控制模型可以弥合传统调制解调器设备和 USB 设备之间的差距。为了支持某些类型的遗留应用程序，需要解决两个问题。第一个是支持特定的传统控制信号和状态变量，这些信号和状态变量直接由各种载波调制标准处理。由于这些依赖性，它们对于开发模拟调制解调器非常重要，模拟调制解调器向主机提供抽象控制模型类型的通信类接口。为了支持这些要求，需创建额外的请求(见表 15.86)和通知(见表 15.87)

表 15.86　请求抽象控制模型

请求	值	描述	要求/选择
SEND_ENCAPSULATED_COMMAND	00H	以支持的控制协议的格式发出命令	必需
GET_ENCAPSULATED_RESPONSE	01H	以支持的控制协议的格式请求响应	必需
SET_COMM_FEATURE	02H	控制特定通信功能的设置	可选

请求	值	描述	要求/选择
GET_COMM_FEATURE	03H	返回通信功能的当前设置	可选
CLEAR_COMM_FEATURE	04H	清除特定通信功能的设置	可选
SET_LINE_CODING	20H	配置 DTE(数据终端设备)速率、停止位、奇偶校验和字符位数	可选+
GET_LINE_CODING	21H	请求当前 DTE 速率、停止位、奇偶校验和字符位数	可选+
SET_CONTROL_LINE_STATE	22H	RS-232 信号，用于告诉 DCE 设备 DTE 设备现在存在	可选
SEND_BREAK	23H	发送用于指定 RS-232 样式中断的特殊载波调制	可选

表 15.87 通知抽象控制横型

通知	代码	描述	要求/选择
NETWORK_CONNECTION	00H	通知主机网络连接状态	可选
RESPONSE_AVAILABLE	01H	通知主机发出 GET_ENCAPSULATED_RESPONSE 请求	必需
SERIAL_STATE	20H	返回载波检测、DSR、中断和振铃信号的当前状态	可选

弥补传统调制解调器设计和抽象控制模型之间差距的第二个重要项目是在数据类接口上复用呼叫控制(AT 命令)的方法。传统的调制解调器设计受限于只支持一个通道的“AT”命令和实际数据。为了允许这种类型的功能，设备必须有一种方法来向主机指定这种限制。

当描述这种类型的设备时，通信类接口仍将指定一个抽象控制模型，但呼叫控制实际上将通过数据类接口发生。为了描述这个特定的特征，呼叫管理功能描述符将置位 bmCapabilities 的 D1 位。

对于同时支持两种模式的设备，即通过通信类接口调用控制和通过数据类接口调用控制，并且需要在两者之间进行切换，则使用 GetCommFeature 请求进行模式切换。

注意：这些请求特定于通信类的模拟调制解调器，强烈建议支持 SET_LINE_CODING 和 SET_LINE_CODING 请求。

具有抽象控制模型的通信类子类代码的通信类接口唯一有效的类特定请求代码在如表 15.87 中列出。其他在上表中未列出的类特定请求，如 SET_HOOK_STATE，不适合抽象控制模型，如果发送到此类接口，将生成 STALL 条件。例如，挂断线路将通过 SEND_ENCAPSULATED_COMMAND 发送“ATH”来完成，而不是使用 SET_HOOK_STATE。

注意：这些请求特定于通信类的模拟调制解调器，强烈建议支持 NETWORK_CONNECTION 和 SERIAL_STATE 请求。

唯一类特定的通知代码，对于具有抽象控制模型的通信类子类代码的通信类接口有效，其他类特定的通知未在上表中列出，例如 RING_DETECT，不适合抽象控制模型，不应由此类设备发送。

S.5.3 USB 通信类设备类特定码

本节列出了通信设备类、通信接口类和数据接口类的代码，包括子类和协议。这些值用于 USB 规范中定义的标准设备描述符的 bDeviceClass、bInterfaceClass、bInterfaceSubClass 和 bInterfaceProtocol 字段。

1. 通信设备类代码 bDeviceClass

表 15.88 定义了通信设备类代码：

表 15.88 通信设备类别代码

码值	类
02H	通信设备类

2. 通信接口类代码 bInterfaceClass

表 15.89 定义了通信类代码：

表 15.89 通信接口类代码

码值	类
02H	通信接口类

3. 通信接口类子类代码 bInterfaceSubClass

表 15.90 定义了通信接口类的子类代码：

表 15.90 通信接口类子类代码

码值	类
00H	保留
01H	直线控制模型
02H	抽象控制模型
03H	电话控制模型
04H	多通道控制模型
05H	CAPI 控制模型
06H	以太网网络控制模型
07H	ATM 网络控制模型
08H-7FH	保留(将来使用)
80H-FEH	保留(供应商特定)

数据泵模型未在通信类子类代码中列出，因为该类型的设备将使用直接线路控制模型进行 POTS 线路控制，并且使用供应商特定接口。

4. 通信接口类控制协议代码 bInterfaceProtocol

USB 主机使用通信控制协议来控制设备或网络上的通信功能。CDC 规范定义了某些标准控制协议的代码值。它还为其他标准或供应商特定的控制协议保留代码。如果通信类控制模型不需要特定的协议，则应使用 00h 的值。

表 15.91 通信接口类控制协议代码

协议码值	参考文档	描述
00H	USB 规范	不需要类特定协议
01H	V.25ter	通用 AT 命令(也称为“Hayes"兼容”)
02H-FEH		保留(将来使用)
FFH	USB 规范	供应商特定

5. 数据接口类代码 bInterfaceClass

表 15.92 定义了数据接口类代码。

表 15.92 数据接口类代码

代码	类
0AH	数据接口

6. 数据接口类子类代码 bInterfaceSubClass

此时此字段未用于数据类接口，其值应为 00h。

7. 数据接口类协议代码 bInterfaceProtocol

表 15.93 定义了数据接口类的协议代码。

表 15.93 数据接口类协议代码

协议码值	参考文档	描述
00H	USB 规范	不需要类特定协议
01H-2FH	无	保留(将来使用)
30H	1.430	ISDN BRI 的物理接口协议
31H	ISO/IEC 3309-1993	高密度液晶
32H	无	透明的
33H-4FH	无	保留(将来使用)
50H	Q.921M	Q.921 数据链路协议的管理协议
51H	Q.921	Q.931 的数据链路协议
52H	Q921TM	Q.921 数据链路协议的 TEI 多路复用器
53h-8Fh	无	保留(将来使用)

协议码值	参考文档	描述
90H	V.42 之二	数据压缩程序
91H	Q.931/欧洲-ISDN	Euro-ISDN 协议控制
92H	V.120	V.24 速率适应 ISDN
93H	CAPI2.0	CAPI 命令
94H-FCH	无	保留(将来使用)
FDH	无	基于主机的驱动程序。 注意：此协议代码只能用于主机和设备之间的消息中，以识别协议栈的主机驱动程序部分
FEH	CDC 规范	使用通信类接口上的协议单元功能描述符来描述协议
FFH	USB 规范	供应商特定

在某些类型的 USB 通信设备中，不需要在数据类接口描述符中指定任何协议。在这些情况下，应使用 00h 的值。

S.5.4 USB 通信类设备描述符

1. 标准 USB 描述符定义

本节定义了通信设备类，通信接口类和数据接口类的标准 USB 描述符的要求，

1)设备描述符

通信设备的功能在接口层，通信设备类代码的定义是个例外。设备代码仅用于识别设备是一个通信设备，因此，多个接口可能被用来构成 USB 功能。这对主机配置驱动程序以正确列举设备是很重要的。所有通信设备将至少有一个通信类接口，作为设备的主接口。表 15.94 定义了正确建立设备描述符和附带的接口描述符的值。

表 15.94 通信设备类描述符要求

偏移	字段	大小	值	描述
4	bDeviceClass	1	02H	通信设备类别代码
5	bDeviceSubClass	1	00H	通信设备子类代码，此时未使用
6	bDeviceProtocol	1	00H	通信设备协议代码，此时未使用

2)配置描述符

通信设备类使用 USB 规范中定义的标准配置描述符。

3)接口描述符

通信接口类使用 USB 规范中定义的标准接口描述符。表 15.95 中定义的字段应按规定使用。通信接口类描述符的其余字段的使用保持不变。

表 15.95 通信类接口描述符要求

偏移	字段	大小	值	描述
5	bInterfaceClass	1	类	通信接口类代码
6	bInterfaceSubClass	1	子类	通信接口类子类代码
7	bInterfaceProtocol	1	协议	通信接口类协议代码，如前一字段中指定的，适用于子类

数据接口类也使用 USB 规范中定义的标准接口描述符。表 15.96 中定义的字段应按规定使用。数据接口类描述符的其余字段的使用保持不变

表 15.96 数据类接口描述符要求

偏移	字段	大小	值	描述
5	bInterfaceClass	1	0AH	数据接口类代码
6	bInterfaceSubClass	1	00H	数据类子类代码
7	bInterfaceProtocol	1	协议	应用于子类的数据类协议代码，如前一字段中指定的

4)端点描述符

USB 规范中定义的标准端点描述符。

2.类特定描述符

本节描述通信接口类和数据接口类的类特定描述符。特定于类的描述符仅存在于接口级别。每个特定于类的描述符都定义为接口的所有功能描述符的串联。设备为接口返回的第一个功能描述符应该是一个头功能描述符。

1)类特定设备描述符

这个描述符包含适用于整个通信设备的信息。通信设备类目前在设备级别上不使用任何特定于类的描述符信息。

2)类特定配置描述符

通信设备类当前不在配置级别使用任何特定于类的描述符信息。

3)功能描述符

功能描述符描述了接口描述符中类特定信息的内容。功能描述符都以一个通用的头描述符开头，它允许主机软件轻松解析类特定描述符的内容，如表 15.97 所示。每个类特定的描述符都包含一个或多个功能描述符。尽管通信类目前定义了类特定的描述符信息，但数据类没有。

表 15.97 功能描述符通用格式

偏移	字段	大小	值	描述
0	bFunctionLength	1	数字	此描述符的大小
1 个	bDescriptorType	1	常数	CS_INTERFACE
2 个	bDescriptorSubtype	1	常数	功能描述符的标识符(ID)。有关支持值的列表

偏移	字段	大小	值	描述
3 个	(function specific data0)	1	杂项	第一个函数特定数据字节。这些字段将根据所表示的功能描述符而有所不同
……	……	……	……	……
N+2	(functional specific data N-1)	1	杂项	第 N 个功能专用数据字节。这些字段将根据所代表的功能描述符的不同而变化

bDescriptorType 值与音频设备规范的 USB 设备类定义中定义的值相同如表 15.98、15.99 所示。它们是通过使用 USB 规范中定义的 DEVICE、CONFIGURATION、STRING、INTERFACE 和 ENDPOINT 常量派生的，并通过设置通用类规范中定义的类特定位来生成相应的类特定常量。

表 15.98 bDescriptorType 字段的类型值

描述符类型	值
CS_INTERFACE	24H
CS_ENDPOINT	25H

表 15.99 功能描述符中的 bDescriptor 子类型

描述符子类型	通信 IF 描述符	数据 IF 描述符	功能说明
00H	是	是	头部功能描述符，它标志着接口的功能描述符串联集的开始
01H	是	不	呼叫管理功能描述符
02H	是	不	抽象控制管理功能描述符
03H	是	不	直线管理功能描述符
04H	是	不	电话铃声功能描述符
05H	是	不	电话呼叫和线路状态报告功能功能描述符
06H	是	不	联合功能描述符
07H	是	不	国家选择功能描述符
08H	是	不	电话操作模式功能描述符
09H	是	不	USB 终端功能描述符
0AH	是	不	网络通道终端描述符
0BH	是	不	协议单元功能描述符
0CH	是	不	扩展单元功能描述符
0DH	是	不	多渠道管理功能描述符
0EH	是	不	CAPI 控制管理功能描述符

描述符子类型	通信 IF 描述符	数据 IF 描述符	功能说明
0FH	是	不	以太网网络功能描述符
10H	是	不	ATM 网络功能描述符
11H-FFH	不适用	不适用	保留(将来使用)

(1)标头功能描述符类

特定描述符应以表 15.100 中定义的标头开头。bcdCDC 字段标识通信设备规范的 USB 类定义的版本，该接口及其描述符符合该版本。

表 15.100 类特定描述符标头格式

偏移	字段	大小	值	描述
0	bFunctionLength	1	数字	此描述符的大小(以字节为单位)
1	bDescriptorType	1	常数	CS_INTERFACE 描述符类型
2	bDescriptorSubtype	1	常数	标头功能描述符子类型
3	bcdCDC	2	数字	通信设备规范的 USB 类定义二进制编码的十进制版本号

(2)呼叫管理功能描述符

呼叫管理功能描述符描述了对通信类接口的呼叫处理。它只能出现在接口描述符的特定类别部分。

表 15.101 呼叫管理功能描述符

偏移	字段	大小	值	描述
0	bFunctionLength	1	数字	此描述符的大小，以字节为单位
1	bDescriptorType	1	常数	CS_INTERFACE
2	bDescriptorSubtype	1	常数	呼叫管理功能描述符子类型
3	bmCapabilities	1	位图	此配置支持的功能： D7..D2：保留(重置为零) D1：0-设备仅发送/接收呼叫管理信息通过通信类接口。 1-设备可以通过数据类接口发送/接收呼叫管理信息。 D0：0-设备不处理呼叫管理本身。 1-设备处理呼叫管理本身。前面的位组合在一起，标识使用哪种呼叫管理方案。如果位 D0 重置为 0，则忽略位 D1 的值。在这种情况下，为了将来的兼容性，位 D1 被重置为零
4	bDataInterface	1	数字	可选地用于呼叫管理的数据类接口的接口号

*本配置中接口的零基索引。(bInterfaceNum)

(3)抽象控制管理功能描述符

抽象控制管理功能描述符描述了通信类接口支持的命令，如第 6.3.6 节中定义的，带有抽象控制模型的子类代码。它只能出现在接口描述符的类特定部分中。

表 15.102 抽象控制管理功能描述符

偏移	字段	大小	值	描述
0	bFunctionLength	1	数字	此描述符的大小，以字节为单位
1 个	bDescriptorType	1	常数	CS_INTERFACE
2 个	bDescriptorSubType	1	常数	呼叫管理功能描述符子类型
3 个	bmCapabilities	1	位图	此配置支持的功能(位值为零表示不支持该请求) D7..D4：保留(重置为零) D3：1-设备支持通知网络连接 D2：1-设备支持请求 Send_Break D1：1-设备支持 Set_Line_Coding、Set_Control_Line_State、Get_Line_Coding 的请求组合和通知串行状态。 D0：1-设备支持请求 Set_Comm_Feature、Clear_Comm_Feature 和 Get_Comm_Feature 的组合。 前面的位组合起来标识哪些请求/通知由具有抽象控制模型的子类代码的通信类接口支持

3. 样本类特定的功能描述符

表 15.103 给出了一个简单抽象控制模型设备的通信类功能描述符的示例。该描述符特定于通信类。

表 15.103 示例通信类特定接口描述符 *

偏移	字段	大小	值	描述
0	bFunctionLength	1	05H	此功能描述符的大小，以字节为单位
1	bDescriptorType	1	24H	CS_INTERFACE
2	bDescriptorSubType	1	00H	标头
3	bcdCDC	2	0110H	通信设备规范的 USB 类定义二进制编码的十进制版本号
5	bFunctionLength	1	04H	此功能描述符的大小，以字节为单位
6	bDescriptorType	1	24H	CS_INTERFACE
7	bDescriptorSubType	1	02H	抽象控制管理功能描述符子类型
8	bmCapabilities	1	0FH	该字段包含值 0Fh，因为设备支持抽象控制模型接

偏移	字段	大小	值	描述
				口的所有相应命令
9	bFunctionLength	1	05H	此功能描述符的大小，以字节为单位
10	bDescriptorType	1	24H	CS_INTERFACE
11	bDescriptorSubType	1	06H	联合描述符功能描述符子类型
12	bMasterInterface	1	00H	控制(communication class)接口的接口号
13	bSlaveInterface0	1	01H	从站(数据类)接口的接口号
14	bFunctionLength	1	05H	此功能描述符的大小，以字节为单位
15	bDescriptorType	1	24H	CS_INTERFACE
16	bDescriptorSubType	1	01H	呼叫管理功能描述符子类型
17	bmCapabilities	1	03H	表示设备自己处理呼叫管理(位 D0 已设置)，除了使用 SEND_ENCAPSULATED_COMMAND(位 D1)发送的命令外，还将处理通过数据接口多路复用的命令已设置)
18	bDataInterface	1	01H	表示多路复用命令是通过数据接口 01h 处理的(与 UNION 功能描述符中使用的值相同)

S.5.5 USB 通信类设备主机请求以及设备通知

通信接口类支持 USB 规范中定义的标准请求。此外，通信接口类有一些特定的类请求和通知。这些是用于设备和呼叫管理的。

1. 管理元素请求

通信接口类支持以下特定于类的请求。本节描述特定于通信接口类的请求。这些请求通过管理元素发送，并且可以应用于由通信类接口代码定义的不同设备视图。本书仅讲述抽象模型涉及的主机请求，如表 15.104、15.105 所示。

表 15.104 类特定请求

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SEND_ENCAPSULATED_COMMAND	Zero	Interface	Amount of data , in bytes , associated with this recipient.	Control protocol-based command
10100001B	GET_ENCAPSULATED_RESPONSE	Zero	Interface	Amount of data , in bytes , associated with this recipient.	Protocol dependent data response
00100001B	SET_COMM_FEATURE	Feature Selector	Interface	Length of State Data	State
10100001B	GET_COMM_FEATURE	Feature	Interface	Length of State Data	State

bmRequestType	bRequest	wValue	wIndex	wLength	Data
		Selector			
00100001B	CLEAR_COMM_FEATURE	Feature Selector	Interface	Zero	None
00100001B	SET_LINE_CODING	Zero	Interface	Size of properties	Line Coding Structure
10100001B	GET_LINE_CODING	Zero	Interface	Size of properties	Line Coding Structure
00100001B	SET_CONTROL_LINE_ STATE	Control Signal Bitmap	Interface	Zero	
00100001B	SEND BREAK	Duration of Break		Interface	Zero

表 15.105 类特定请求代码

请求	值
SET_COMM_FEATURE	02h
GET_COMM_FEATURE	03h
CLEAR_COMM_FEATURE	04h
RESERVED(future use)	05h-0Fh
SET_AUX_LINE_STATE	10h
SET_HOOK_STATE	11h
PULSE_SETUP	12h
SEND_PULSE	13h
SET_PULSE_TIME	14h
RING_AUX_JACK	15h
RESERVED(future use)	16h-1Fh
SET_LINE_CODING	20h
GET_LINE_CODING	21h
SET_CONTROL_LINE_STATE	22h
SEND_BREAK	23h
RESERVED(future use)	24h-2Fh
SET_RINGER_PARMS	30h

请求	值
GET_RINGER_PARMS	31h
SET_OPERATION_PARMS	32h
GET_OPERATION_PARMS	33h
SET_LINE_PARMS	34h
GET_LINE_PARMS	35h
DIAL_DIGITS	36h
SET_UNIT_PARAMETER	37h
GET_UNIT_PARAMETER	38h
CLEAR_UNIT_PARAMETER	39h
GET_PROFILE	3Ah
RESERVED(future use)	3Bh-3Fh
SET_ETHERNET_MULTICAST_FILTERS	40h
SET_ETHERNET_POWER_MANAGEMENT_PATTERN FILTER	41h
GET_ETHERNET_POWER_MANAGEMENT_PATTERN FILTER	42h
SET_ETHERNET_PACKET_FILTER	43h
GET_ETHERNET_STATISTIC	44h
RESERVED(future use)	45h-4Fh
SET_COMM_FEATURE	02h
GET_COMM_FEATURE	03h
SET_ATM_DATA_FORMAT	50h
GET_ATM_DEVICE_STATISTICS	51h
SET_ATM_DEFAULT_VC	52h
GET_ATM_VC_STATISTICS	53h
RESERVED(future use)	54h-FFh

1)发送封装命令 SendEncapsulatedCommand

该请求用于以通信类接口支持的控制协议的格式发出命令，如表 15.106 所示。

表 15.106 Send_Encapsulated_Command 内容

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SEND_ENCAPSULATED_COMMAND	0	接口号	与该接收者相关的数据量，以字节为单位	基于控制协议的命令

2)获取封装响应 GetEncapsulatedResponse

该请求用于请求以通信类接口支持的控制协议格式的响应，如表 15.107 所示

表 15.107 Get_Encapsulated_response 内容

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_ENCAPSULATED_RESPONSE	0	接口号	与该接收者相关的数据量，以字节为单位	与协议有关的数据

3)设置通讯功能 SetCommFeature

此请求控制特定目标的特定通信功能的设置，如表 15.108 所示

表 15.108 Set_Comm_Feature 内容

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_COMM_FEATURE	特征选择器	接口号	状态数据的长度	状态

有关每个目标的已定义功能选择器列表的更多信息，请参阅第 65.1.4 节，“GetCommFea-ture

4)获取通讯功能 GetCommFeature

此请求返回所选通信功能的当前设置，见表 15.109、15.110

表 15.109 GetCommFeature 内容

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_COMM_FEATURE	特征选择器	接口号	状态数据的长度	状态

表 15.110 通信功能选择器代码

特征选择器	码值	目标	数据长度	描述
保留	00H	无	无	留作将来使用
ABSTRACT_STATE	01H	接口号	2 个	描述此抽象模型通信设备的多路复用状态和空闲状态的两个字节数据。该选择器仅对抽象控制模型有效
COUNTRY_SETTING	02H	接口号	2 个	ISO 3166 中定义的十六进制格式的国家代码，国家选择功能描述符的偏移量 3 中指定的发布日期。此选择器仅对提供国家选择功能描述符的设备有效，提供的值应在国家选择功能描述符中显示为受支持的国家

对于 ABSTRACT_STATE 选择器，见表 15.111，定义了以下两个字节的数据。

表 15.111 为 ABSTRACT_STATE 选择器返回的特征状态

位位置	描述
D15..D2	保留(重置为零)

D1	数据复用状态 1：在数据类上启用呼叫管理命令的多路复用 0：禁用复用
D0	空闲设置 1：该接口中的所有端点均不接受来自主机的数据或向主机提供数据。这允许主机呼叫管理软件将呼叫管理元素与其他媒体流接口和端点同步，特别是那些与不同主机实体关联的接口和端点(例如配置为 USB 音频类的语音流设备)。 0：此接口中的端点将继续接受/提供数据

5)清除通讯功能 ClearCommFeature

此请求控制特定目标的特定通信功能的设置，将所选功能设置为其默认状态。特征选择器的有效性取决于请求的目标类型，如表 15.112 所示。

表 15.112 Clear_Comm_Feature 内容

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	Clear_Comm_Feature	特征选择器	接口号	0	状态

有关每个目标的已定义功能选择器列表的更多信息，请参阅第 6.2.4 节，“Get_Comm_Feature”。

6)设置线路编码 SetLineCoding

此请求允许主机指定典型的异步行字符格式属性，某些应用程序可能需要这些属性，如表 15.113 所示。此请求适用于异步字节流数据类接口和端点；它还适用于从主机到设备以及从设备到主机的数据传输。

表 15.113 Set_Line_Coding 内容

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	Set_Line_Coding	0	接口号	结构体大小	线路编码结构体

有关有效属性的定义，请参阅“GetLineCoding”。

7)获取线路编码 GetLineCoding

该请求允许主机找出当前配置的线路编码，如表 15.114 所示，

表 15.114 Get_Line_Coding 内容

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	Get_Line_Coding	0	接口号	结构尺寸	线路编码结构

表 15.115 定义了线路编码结构体属性

表 15.115 定义了线路编码结构体属性：

偏移	字段	大小	值	描述
0	dwDTERate	4	数字	数据终端速率，以比特每秒为单位。

4	bCharFormat	1	数字	停止位 0-1 个停止位 1-1.5 停止位 2-2 个停止位
5	bParityType	1	数字	校验类型 0-None 1-Odd 2-Even 3-Mark 4-Space
6	bDataBits	1	数字	数据位(5、6、7、8 或 16)

8)设置控制线状态 SetControlLineState

此请求生成 RS-232/V.24 样式的控制信号，如表 15.116-1、15.116-2 所示。

表 15.116-1　Set_Control_Line_State 内容

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_CONTROL_LINE_STATE	控制信号位图	接口号	0	无

表 15.116-2　Set_Control_Line_State 的控制信号位图值

位位置	描述
D15..D2	保留(重置为零)
D1	半双工调制解调器的载波控制。该信号对应于 V.24 信号 105 和 RS-232 实时信号。 0-停用运营商 1-激活载体 在全双工模式下运行时，设备会忽略该位的值
D0	向 DCE 指示 DTE 是否存在。该信号对应于 V.24 信号 108/2 和 RS-232 信号 DTR 0-不存在 1-存在

9)发送中断 SendBreak

此请求发送生成 RS-232 样式中断的特殊载波调制，如表 15.117 所示

表 15.117　Send_Break 内容

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SEND_BREAK	break 持续	接口号	0	无

wValue 字段包含中断信号的时间长度(以毫秒为单位)。如果 wValue 包含值 FFFFh，则设备将发送中断，直到收到另一个 wValue 为 0000h 的 SendBreak 请求。

把没有任何替换为无

2. 通知元素通知

本节定义设备用于通知主机接口或端点事件的通信接口类通知，如表 15.118、15.119 所示。

表 15.118 类特定通知

bmRequestType	bNotificalion	wValue	wIndex	wLength	Data
10100001B	NETWORK_CONNECTION	0-断开 1-连接的	界面	0	无
10100001B	RESPONSE_AVAILABLE	0	界面	0	无
10100001B	SERIAL_STATE	0	界面	2	UART 状态位图

表 15.119 类特定通知代码

Notification	值
NETWORK_CONNECTIN	00h
RESPONSE_AVAILABLE	01h
RESERVED(future use)	02h-07h
AUX_JACK_HOOK_STATE	08h
RING_DETECT	09h
RESERVED(future use)	0Ah-1Fh
SERIAL_STATE	20h
RESERVED(future use)	21h-27h
CALL_STATE_CHANGE	28h
LINE_STATE_CHANGE	29h
CONNECTION_SPEED_CHANGE	2Ah
RESERVED (future use)	2Bh-FFh

1)网络连接 NetworkConnection

此通知允许设备通知主机网络连接状态，如表 15.120 所示，

表 15.120 Network_Connection 内容

bmRequestType	bNotificalion	wValue	wIndex	wLength	Data
10100001B	NETWORK_CONNECTION	0-断开 1-连接的	接口号	0	无

2)应可用 ResponseAvailable

此通知允许设备通知主机响应可用。可以使用后续的 GeEncapsulated_Response 请求检索此响应，如表 15.121 所示。

表 15.121 Response_Available 内容

bmRequestType	bNotificalion	wValue	wIndex	wLength	Data
10100001B	响应可用	0	接口号	0	无

3)串行状态 SerialState

此通知发送 UART 状态的异步通知，如表 15.122 所示

表 15.122 Serialstate 内容

bmRequestType	bNotificalion	wValue	wIndex	wLength	Data
10100001B	串行状态	0	接口号	2	UART 状态位图

数据字段是一个位图值，包含载波检测、传输载波、中断、振铃信号和设备溢出错误的当前状态。这些信号通常位于 UART 上，用于报告通信状态。如果状态的相应位设置为 1，则该状态被视为已启用。

串行状态就像一个真正的中断状态寄存器一样使用。发送通知后，设备将重置并重新评估不同的信号。对于载波检测或传输载波等一致信号，这意味着在状态发生变化之前不会生成另一个通知。对于不规则信号，如中断、传入振铃信号或超限错误状态，这会将它们的值重置为零，并且在它们的状态发生变化之前不会再次发送通知，如表 15.123 所示。

表 15.123 UART 状态位图值

位	字段	描述
D15..D7		保留(将来使用)
D6	bOverRun	由于设备溢出，接收到的数据已被丢弃
D5	bParity	发生奇偶校验错误
D4	bFraming	发生帧错误
D3	bRingSignal	设备振铃信号检测状态
D2	bBreak	设备中断检测机制的状态
D1	bTxCarrier	传输载体的状态。此信号对应于 V，24 信号 106 和 RS-232 信号 DSR
D0	bRxCarrier	设备接收载波检测机制的状态。此信号对应于 V.24 信号 109 和 RS-232 信号 DCD

S.6 通信设备类程序设计

本文档依据 STC 官方提供的源码对程序设计进行讲解，本文档仅涉及 CDC 设备中 POTS 模型中的抽象模型，其他模型的源码需要读者自行了解。本文档将使用 USB CDC 协议相关的知识，实现一个 USB 设备和 UART 设备的通信桥。

S.6.1 CDC 程序设计原理(缺少 LINECODING 数据结构的说明)

1. 设计分层

此次程序设计由于涉及不同的协议规范，因此需要对设计进行分层，以便日后扩展和修改。此次程序设计依据不同协议规范分为以下几层，由于厂商没有定义特定的请求，故厂商请求层部分将略去。

(1)USB 层

(2)USB 标准请求层

(3)CDC 协议子类请求层

(4)串口层

1)USB 层

该层对应程序设计中的 usb.c 文件，该层存放 USB 初始化函数以及 USB 中断处理函数，USB 中断处理函数中提供了 USB 所有相关中断的处理函数的入口，其中代码相同部分的分析详见 USB2.0 程序设计。

不同处，EPOUT1 的中断响应函数需要对主机发出的 OUT 进行响应：需要进行将主机下行的数据包存入 256 字节的 RxBuffer 缓冲区中。

2)USB 标准请求层

该层对应程序设计中的 usb_req_std.c 文件，该层存放设备对 USB 主机 11 个标准请求的响应函数，具体详见第 3 章。

不同处，在控制传输阶段，若主机发送 OUT 包，需要设备对串口进行设置，串口设置函数处于 CDC 协议子类请求层。

3)CDC 协议子类请求层

该层对应程序设计中的 usb_req_class.c 文件，为了标准起见，本书 CDC 协议子类请求函数同样分为三个步骤:1.特殊情况处理；2.依据状态发包；3.结束状态，即进行数据操作(如发送数据，或者接收数据，或者不进行数据操作)。

(1)usb_set_line_coding，格式见表 15.124。

表 15.124

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_LINE_CODING	0	接口号	结构体大小	线路编码结构体

第 1 步:特殊情况处理,若请求的 bmRequestType 字段不满足 usb_set_line_coding 请求的要求则挂起。

第 2 步:端点 0 数据区指向线路编码结构体的数据区，端点 0 的大小定义为请求数据结构的大小。

第 3 步:进入请求结束状态,端点 0 设置成 DATAOUT 状态表示设备接收来自主机的数据,寄存器 CSR0 的 SPORDY 位置位，表示处理完成从端点 0 接收到的数据包。

(2)usb_get_line_coding，格式见表 15.125。

表 15.125

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_LINE_CODING	0	接口号	结构体大小	线路编码结构体

第 1 步:特殊情况处理,若请求的 bmRequestType 字段不满足 usb_get_line_coding 请求的要求则挂起。

第 2 步:端点 0 数据区指向线路编码结构体的数据区，端点 0 的大小定义为请求数据结构的大小。

第 3 步:请求结束状态，软件调用 usb_setup_in()，完成数据包的发送。

注意:usb_setup_in()的内容是:先将控制端点 0 设置为 DATAIN 状态，控制寄存器 CSR0 的 SPORDY 写 1 以清零 OPRDY，随后调用 usb_ctrl_in()进行数据发送，即使用控制 IN 类型向主机发送数据包。

(3)usb_set_ctrl_line_state，格式见表 15.126。

表 15.126

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_CONTROL_LINE_STATE	控制信号位图	接口号	0	无

第 1 步:特殊情况处理,若请求的 bmRequestType 字段不满足 usb_set_ctrl_line_state 请求的要求则挂起。

第 2 步:无该步骤。

第 3 步:进人请求结束状态，端点 0 设置成空闲状态，寄存器 CSR0 的 SPORDY 位以及 DATEND 位置位，表示处理完成从端点 0 接收到的数据包，且数据结束。

4)串口层

该层对应程序设计中的 uart.c 文件，串口层中存放了串口初始化函数、串口中断处理函数，串口设置奇偶校验函数、串口设置波特率函数，还有串口轮询函数。由于轮询函数是本次程序设计中 Ai8051U 最主要的工作负载，因此本节最为重要的是讲述轮询数的写法。

(1)串口设置

在前面 1.1.2 节中说过在控制传输阶段，若主机发送 OUT 包，需要设备对串口进行设置，尽管 USB 串口设置函数(sb_uart_settings)处于协议子类请求层，但是 USB 串口设置函数调用了串口设置奇偶校验函数、串口设置波特率函数。

其原理是，若在控制传输阶段，若主机发送 OUT 包，且请求类型为 SET_LINE_CODING，则需要依据接收到的线路编码结构体的奇偶校验字段以及波特率字段，对 Ai8051U 上串口的奇偶校验以及波特率进行设置。

以上设置的好处是，通过次程序设计创建的工程不需要依据串口设备的波特率和奇偶校验进行额外的设置。

(2)串口轮询

DATAIN:若串口设备通过发送数据给 Ai8051U 触发了 Ai8051U 的串口中断，Ai8051U 会将串口发送的数据缓存进缓存区 TxBuffer 中，此时指向 TxBuffer 数据的头指针与尾指针不相等。

若在串口轮询时发现 TxBuffer 数据的头指针与尾指针不相等,则启动 Ai8051U 发送数据给 USB 主机的通信过程，具体步骤如下:

第一步：关闭 USB 中断;

第二步：设置端点为忙碌状态，将端点定位至端点 1;

第三步：将 TxBuffer 中的数据以字节为单位写人 FIFO1 寄存器中，若写人的字节数超过端点的大小

或者数据的头指针与尾指针相等，则退出循环。

最后设置 INCSR1 寄存器的 INIPRDY 位(硬件要求)再打开 USB 中断。

注意:由于以上过程为轮询执行，故当 TxBuffer 中存储的数据大小超过端点大小时，在下一次轮询时 Ai8051U 会继续发送 TxBuffer 缓冲存储区的数据，直至 TxBuffer 中的数据为空。

TX：在 1.1.1 节 USB 层中若 USB 主机通过下行数据给端点 1，从而触发了 EP1OUT 的中断，则 Ai8051U 会将 USB 主机下行的数据缓存至 RxBuffer 中等待轮询发送，与 TxBuffer 一样，RxBuffer 也定义了数据的头指针与尾指针。

若在串口轮询时发现 RxBuffer 数据的头指针与尾指针不相等,则启动 Ai8051U 发送数据给串口设备的通信过程，具体为:

第一步：设置串口忙

第二步：依据线路编码结构体中的校验类型字段设置 STC2 串口的校验类型。

第三步：以字节为单位将 RxBuffer 中的数据写人 S2BUF 寄存器中，并等待发送完成。

缓存区即将溢出处理：RxBuffer 和 TxBuffer 缓冲区的大小是有限的，其固定为 256 字节。如果下一次缓冲区中接收的数据要超过缓冲区的大小，则缓冲区应停止接收数据，并设置相应的状态字，如：UsboutBusy，表示若接收下一次数据，则缓冲区数据将溢出，拒绝接收下一次数据。

2.描述符数据结构

描述符数据结构存储于工程目录下 usb_desc.c 文件中，数据结构依据《通用串行总线通讯类设备定义》中的数据进行定义。读者可配合本节讲解对 usb_desc.c 文件进行阅读。

(1)设备描述符

通信类设备描述符 bDeviceClass 字段值固定为 02H，bDeviceSubClass 以及 bDeviceProtocol 字段值固定为 00H，其余字段值依据 USB 规范进行定义。

(2)配置描述符

依据 USB 规范，USB 主机发送 Get Descriptor 请求获取设备的描述符数据结构，其中配置描述符、接口描述符、类特定描述符、端点描述符是合并为一个数据结构发送给主机的。

① 配置描述符

配置描述符依据 USB 规范定义方式进行定义。需要注意的是本次实验中接口定义了两个，因此配置描述符中 bNumInterfaces 字段值定义为 0x02H，后面的接口描述符也将有两个，其接口号分别为 0 还有 1。

②)接口描述符

接口描述符 0

该接口描述符用于定义设备的接口类型为通信类模型，接口子类为抽象控制模型，协议类型为 V.25ter(通用 AT 命令),故 bInterfaceClass 字段值定义为 02H,bInterfaceSubClass 字段值定义为 02H,bInterfaceProtocol 字段值定义 01H。其余字段依据 USB 规范定义方式进行定义。由于该描述符定义了抽象模型，故在该描述符后应当接续类特定描述符。该接口使用了定义为中断端点的端点 2。

接口描述符 1

该接口描述符由于定义设备的接口类型为数据类接口，该接口使用了定义为批量的端点 1

③类特定描述符

各字段值请参照通用串行总线通信类设备(CDC)文档。

④)端点描述符

端点 2

由于端点 2 仅被定义为中断 IN 端点，故 bEndpointAddress 字段值定义为 82H，bmAttributes 字段值定义为 03H，其余字段依据 USB 规范定义方式进行定义。

端点 1

由于端点 1 既要使用 IN 端点又要使用 OUT 端点，因此需要用两个数据结构分别定义端点 1。端点 1 的数据传输模式依据协议需要定义为批量类型的端点。因此端点 1IN 的 bEndpointAddress 字段值定义为 81H，端点 1OUT 的 bEndpointAddress 字段值定义为 01H，两个端点描述符结构中 bmAttributes 字段值都为 02H，其余字段依据 USB 规范定义方式进行定义。

(3)厂商描述符

此处依据厂商标准对该描述符进行定义。

3. 通信数据流图

图 15.36 为本次程序设计的设备通信数据流简化图。

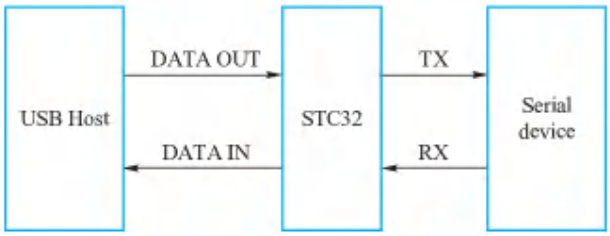


图 15.36　设备通信数据流图

更为详细的描述为:

USB 主机发送数据到串口设备:USB 主机触发 Ai8051U 的 USB 中断,Ai8051U 使用 EP1OUT 端点的 FIFO1 寄存器负责接收来自 USB 主机的数据，并将接收到的数据放入 RxBuffer 缓冲区内。在 RxBuffer 缓冲区内的数据需要等待 Ai8051U 的轮询才能通过 S2BUF 寄存器被发送给串口设备。

串口设备发送数据给 USB 主机:串口设备发送数据触发 Ai8051U 的串口中断,Ai8051U 使用 TxBuffer 缓存来自 S2BUF 的数据,TxBuffer 缓冲区内的数据需要等待 Ai8051U 的轮询才能通过 EP1IN 端点的 FIFO1 寄存器被发送给 USB 主机。

注意:在 Ai8051U 的 RX 中断触发时，Ai8051U 是从串口设备接收数据，但是其数据存入的缓冲区是 TxBuffer 缓冲区。

为了更详细地说明以上数据过程，上述通信模型可以描述为如图 15.37 所示。

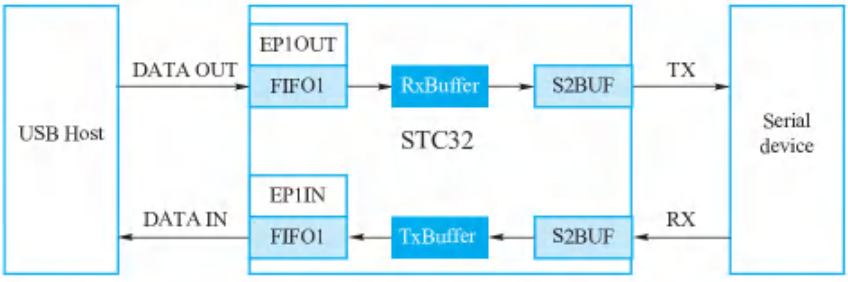


图 15.37　设备通信模型

图 15.37 中，黑框的 TxBuffer 与 RxBuffer 为软件定义地 256 字节数据缓冲区，灰框为 Ai8051U 本身有的寄存器，白框为 USB 端点，箭头指向为数据流流向。注意 S2BUF 与 FIFO1 都是单字节的，从两个寄存器中读取数据至软件定义的数据的缓冲区是以入队的方式读取的，而从软件定义的数据缓冲区中发送数据给这两个寄存器是以出队的方式进行的。

4. 例程使用说明

Ai8051U 实验箱上的 DB9 接口连接串口设备，Ai8051U 实验箱上的 USB 接口连接 USB 主机，如图 15.38 所示，为了便捷起见我们使用 PC 兼做 USB 主机以及串口设备，需要注意的是，使用该例程时不能将串口与 USB 连接于同一 USB Hub，这样会造成设备冲突导致系统蓝屏。

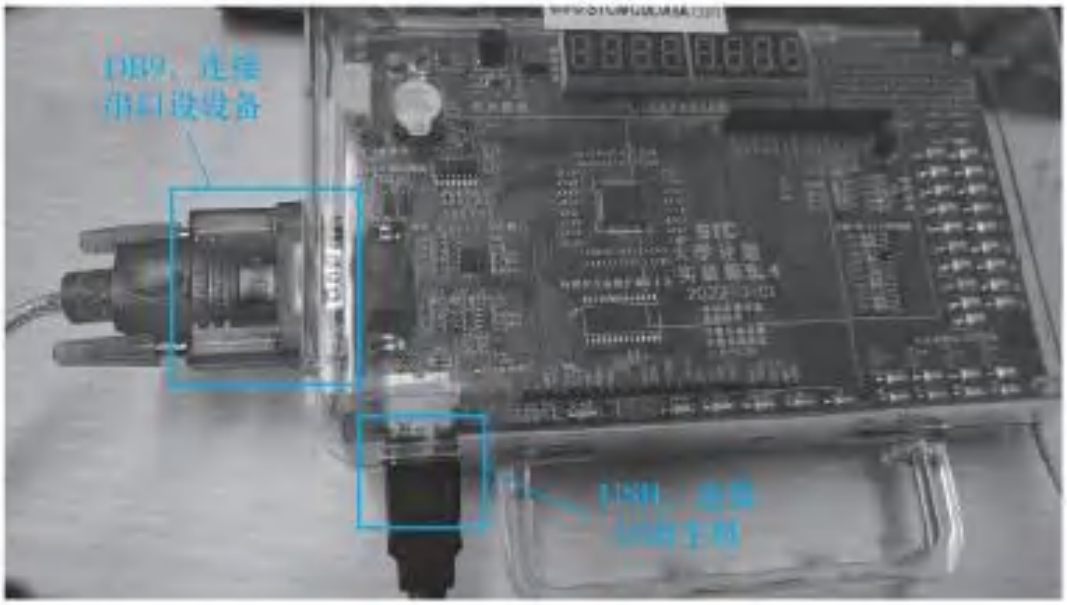


图 15.38　实验箱连接图

将连接好设备，右键开始打开 PC 的设备管理器，我们可以读取到连接 Ai8051U 实验箱的串口号，如图 15.38 所示，串口号为 17。

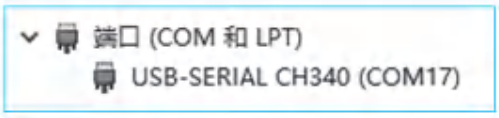


图 15.39　串口号

打开 XCOM 软件或者其他类型的串口管理软件，将串口号定位至设备管理器所读取的串口号，设置波特率为 115200，停止位为 1，数据位为 8，无奇偶校验，打开串口。如图 15.40 所示。

打开 STC-ISP 烧录程序，此时设备管理器会新识别一个串口设备，如图 15.41 所示，将 ISP 切换至 USB-CDC 串口，将串口定位至 PC 新识别的那个串口号，其余串口参数按之前的设置情况进行设置。打开串口，如图 15.42 所示。

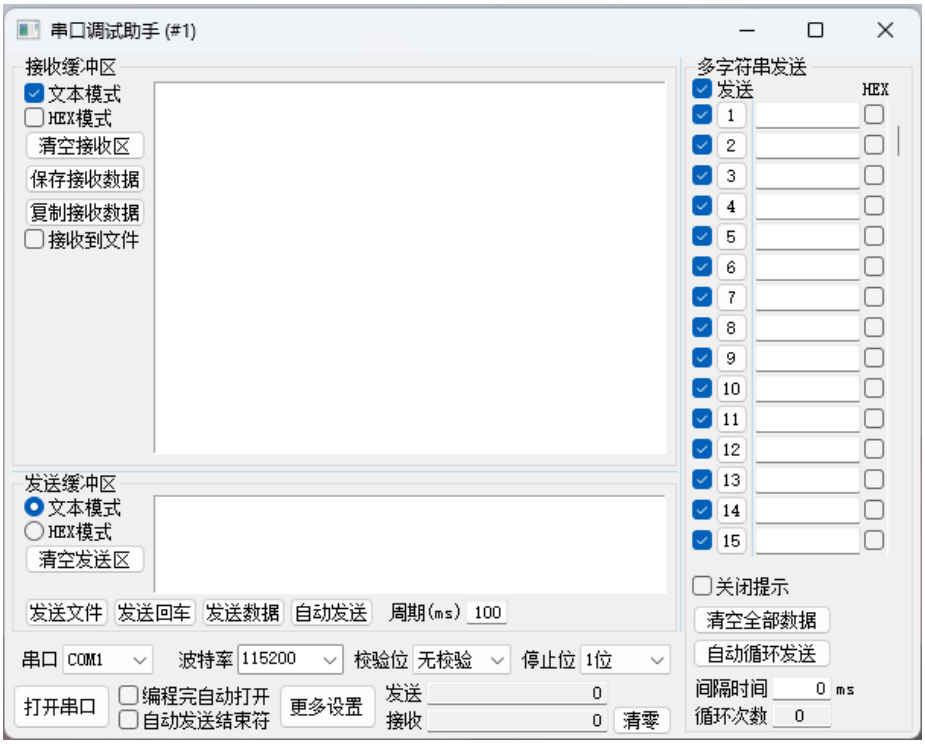


图 15.40　XCOM 设置界面

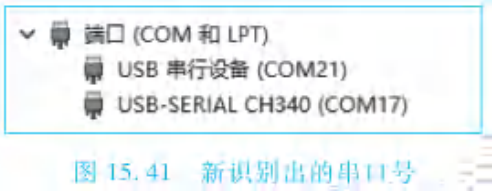


图 15.41　新识别出的串口号

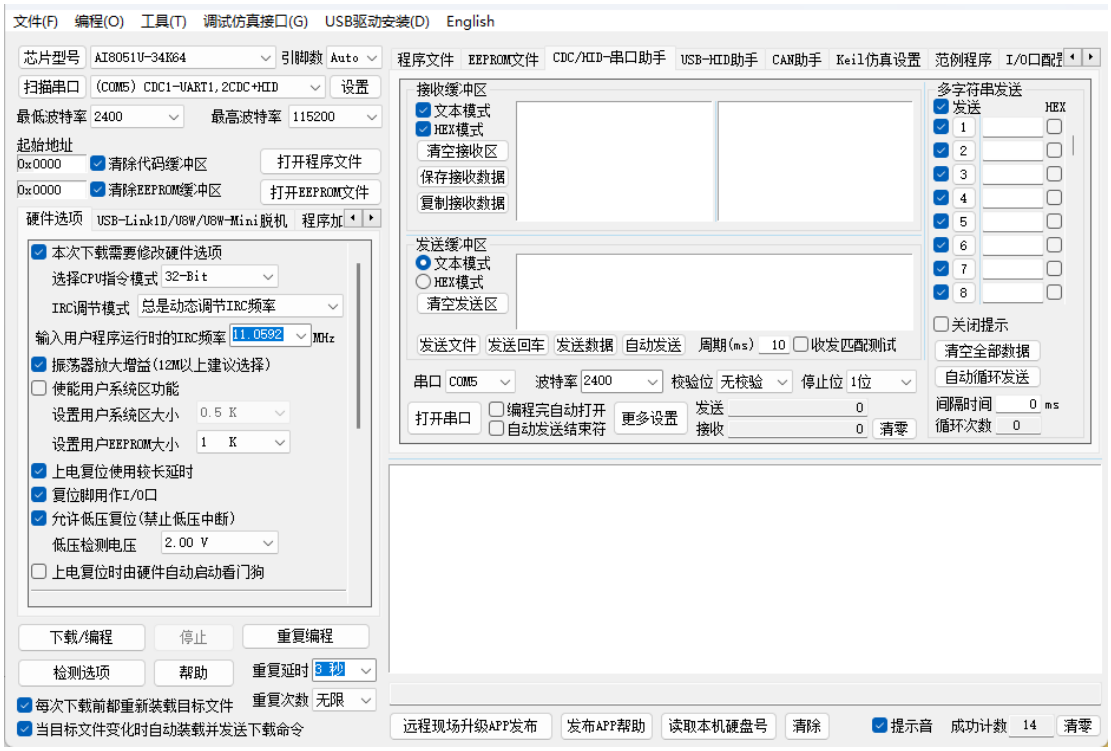


图 15.42　AIapp-ISP USB-CDC/串口助手界面

在 STC-ISP 的 USB-CDC/串口助手界面中的发送缓冲区输入数据，单击发送数据。之后切换回 XCOM 软件界面，发现正确接收数据，如图 15.43 所示。

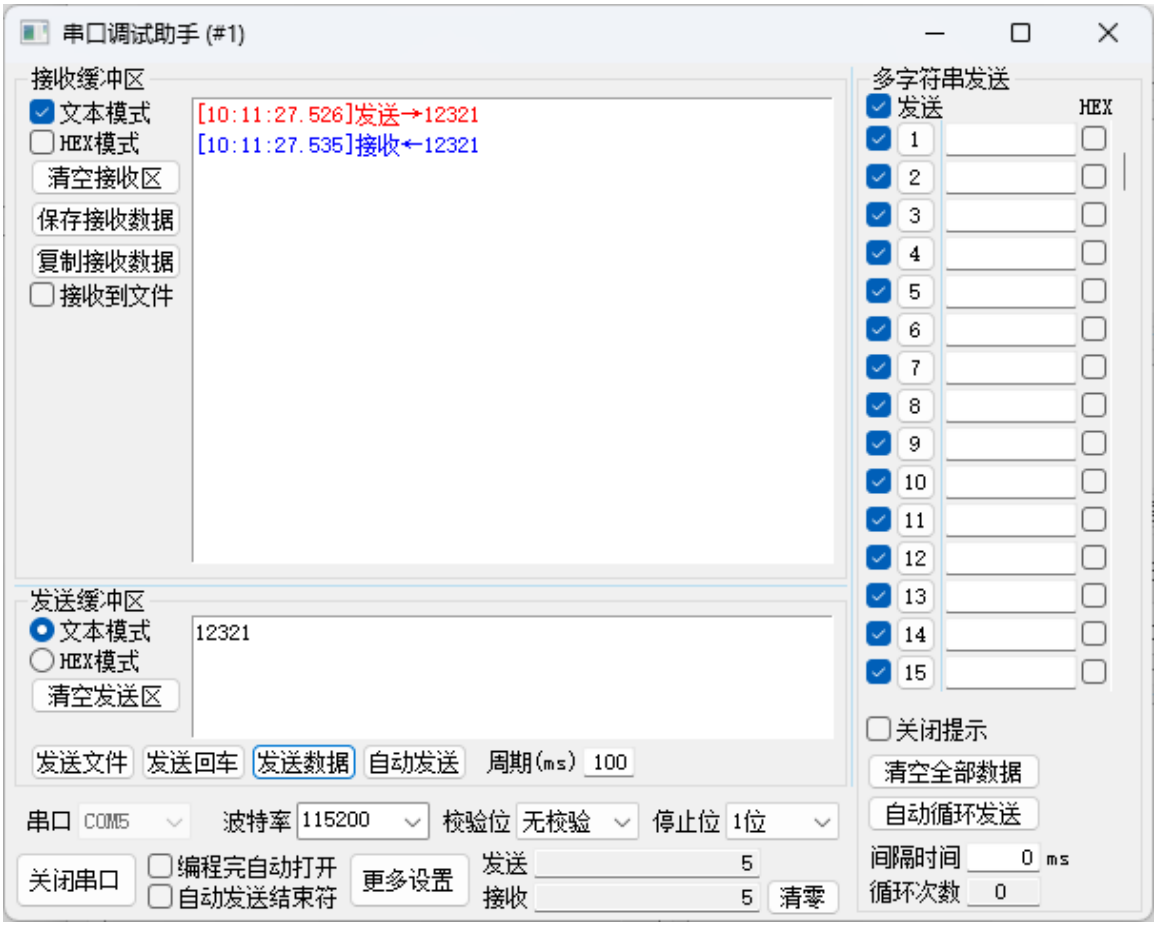


图 15.43　串口设备接收到的数据

在 XCOM 中输入数据并单击发送，且换回 STC-ISP，发现正确接收数据，如图 15.44 所示

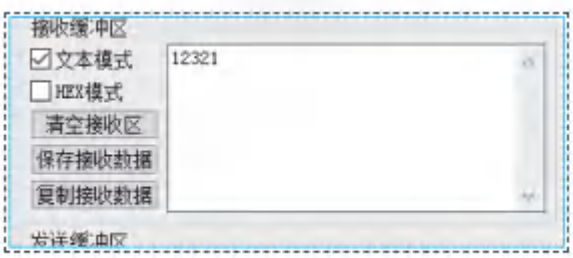


图 15.44　USB 主机接收到的数据

附录T　关于“深大--定时器 0/1，24 位定时器实验”的定时器解释

【背景】在验证“深大--定时器 0/1，24 位定时器实验（定时器中断点灯并通过 USB-CDC 端口输出信息）”这个实验效果时，有同学对在 AIapp-ISP 系统软件的 USB-CDC 窗口显示的“Timer0！”和“Timer1！”的顺序提出疑问：为什么在 2s 的时候，总是显示“Timer0！”在前而“Timer1！”在后？他也试着调换源程序中“Timer0_Init();”和“Timer1_Init();”的顺序，但并不改变在 USB-CDC 窗口显示的显示顺序。

对此，做出如下解释：

源代码中

```
void Timer0_Init(void)          //2 秒@40.000MHz
{
    TM0PS = 0x65;               //设置定时器时钟预分频
    AUXR &= 0x7F;               //定时器时钟 12T 模式
    TMOD &= 0xF0;               //设置定时器模式
    TL0 = 0xB1;                 //设置定时初始值
    TH0 = 0x00;                 //设置定时初始值
    TF0 = 0;                    //清除 TF0 标志
    TR0 = 1;                    //定时器 0 开始计时
    ET0 = 1;                    //使能定时器 0 中断
}
```

$$\text{定时器周期 公式} = \frac{65536 - [TH0, TL0]}{SYSclk/(TM0PS+1)} \times 12$$

上面的代码是使用 AIapp-ISP 软件自动生成的 40MHz 主频定时 2 秒的初始化程序，由于存在误差，我们根据定时器周期计算公式，结合上面的代码反推一下实际的定时时长，

$$\text{定时器周期} = \frac{65536 - [0x00, 0xB1]}{40MHz/(0x65+1)} \times 12 \quad = 1.9999854\ (\text{秒})$$

实际的定时时长　= (0x10000 - 0x00B1) * 12 * (0x65 + 1) / 40000000
= (65536 - 177) * 12 * (101 + 1) / 40000000
= 1.9999854 (秒)

```
void Timer1_Init(void)          //500 毫秒@40.000MHz
{
    TM1PS = 0x19;               //设置定时器时钟预分频
    AUXR &= 0xBF;               //定时器时钟 12T 模式
    TMOD &= 0x0F;               //设置定时器模式
    TL1 = 0x99;                 //设置定时初始值
```

```
    TH1 = 0x05;                 //设置定时初始值
    TF1 = 0;                    //清除 TF1 标志
    TR1 = 1;                    //定时器 1 开始计时
    ET1 = 1;                    //使能定时器 1 中断
}
```

上面的代码是使用 AIapp-ISP 软件自动生成的 40MHz 主频定时 0.5 秒的初始化程序，由于存在误差，我们使用上面的代码反推一下实际的定时时长

$$\text{定时器周期} = \frac{65536 - [0x05, 0x99]}{40MHz/(0x19+1)} \times 12 \quad = 0.5000034\ (\text{秒})$$

实际的定时时长 = (0x10000 - 0x0599) * 12 * (0x19 + 1) / 40000000
= (65536 - 1433) * 12 * (25 + 1) / 40000000
= 0.5000034 (秒)

【结论一】：

使用 40M 的主频产生 2 秒和 500ms 都存在误差，定时器 0 定时 2 秒，实际的定时长度约为 1.9999854 秒；定时器 1 定时 500ms，实际的定时长度约为 0.5000034 秒，（累积 4 次后，时间为 2.0000136）。定时器 0 定时 2 秒的时间比定时器 1 定时 0.5 秒*4 的时间短，所以 CDC 串口显示的打印数据，一定是 Timer0 在前，Timer1 在后。

现在修改工作频率为 24MHz，再来观察：

24M 定时器 0 测试代码 1

```
void Timer0_Init(void)          //2 秒@24.000MHz
{
    TM0PS = 0x3D;               //设置定时器时钟预分频
    AUXR &= 0x7F;               //定时器时钟 12T 模式
    TMOD &= 0xF0;               //设置定时器模式
    TL0 = 0xFC;                 //设置定时初始值
    TH0 = 0x03;                 //设置定时初始值
    TF0 = 0;                    //清除 TF0 标志
    TR0 = 1;                    //定时器 0 开始计时
    ET0 = 1;                    //使能定时器 0 中断
}
```

实际的定时时长 = (0x10000 - 0x03FC) * 12 * (0x3D + 1) / 24000000
= (65536 - 1020) * 12 * (61 + 1) / 24000000
= 1.999996 (秒)

24M 定时器 0 测试代码 2（在代码中，人为修改 TL0 = 0xFB）

```
void Timer0_Init(void)          //2 秒@24.000MHz
{
    TM0PS = 0x3D;               //设置定时器时钟预分频
    AUXR &= 0x7F;               //定时器时钟 12T 模式
    TMOD &= 0xF0;               //设置定时器模式
```

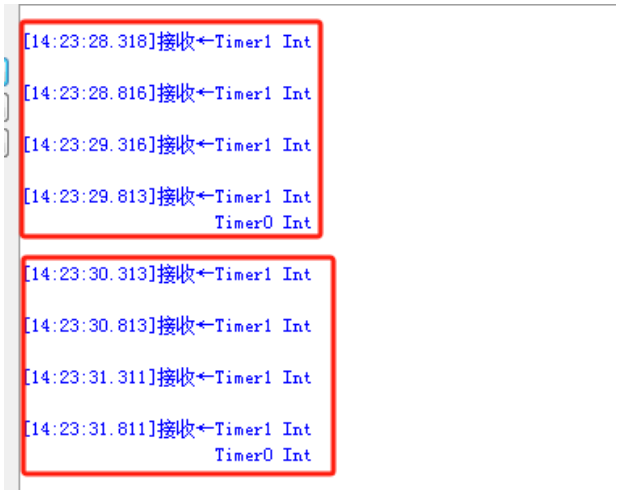
```
    TL0 = 0xFB;                 //设置定时初始值
    TH0 = 0x03;                 //设置定时初始值
    TF0 = 0;                    //清除 TF0 标志
    TR0 = 1;                    //定时器 0 开始计时
    ET0 = 1;                    //使能定时器 0 中断
}
实际的定时时长  = (0x10000 - 0x03FB) * 12 * (0x3D + 1) / 24000000
                = (65536 - 1019) * 12 * (61 + 1) / 24000000
                = 2.000027 (秒)
```

```
void Timer1_Init(void)          //500 毫秒@24.000MHz
{
    TM1PS = 0x0F;               //设置定时器时钟预分频
    AUXR &= 0xBF;               //定时器时钟 12T 模式
    TMOD &= 0x0F;               //设置定时器模式
    TL1 = 0xDC;                 //设置定时初始值
    TH1 = 0x0B;                 //设置定时初始值
    TF1 = 0;                    //清除 TF1 标志
    TR1 = 1;                    //定时器 1 开始计时
    ET1 = 1;                    //使能定时器 1 中断
}
实际的定时时长  = (0x10000 - 0x0BDC) * 12 * (0x0F + 1) / 24000000
                = (65536 - 3036) * 12 * (15 + 1) / 24000000
                = 0.5 (秒)
```

【结论二】：

1、使用 24M 的主频产生 2 秒存在误差，产生的 500ms 无误差。定时器 0 定时 2 秒，实际的定时长度约为 1.999996 秒；定时器 1 定时 500ms，实际的定时长度为 0.5 秒，（累积 4 次后，时间为 2 秒整）。定时器 0 定时 2 秒的时间比定时器 1 定时 0.5 秒*4 的时间短，所以 CDC 串口显示的打印数据，还是 Timer0 在前，Timer1 在后。

2、如果使用上面的“24M 定时器 0 测试代码 2”人为的修改定时器 0 初始化代码，将“TL0 = 0xFC;”修改为“TL0 = 0xFB;”，使得定时器 0 的定时时间变为 2.000027 秒。定时器 0 定时 2 秒的时间比定时器 1 定时 0.5 秒*4 的时间长，CDC 串口显示的打印数据就会变成 Timer1 在前，Timer0 在后。

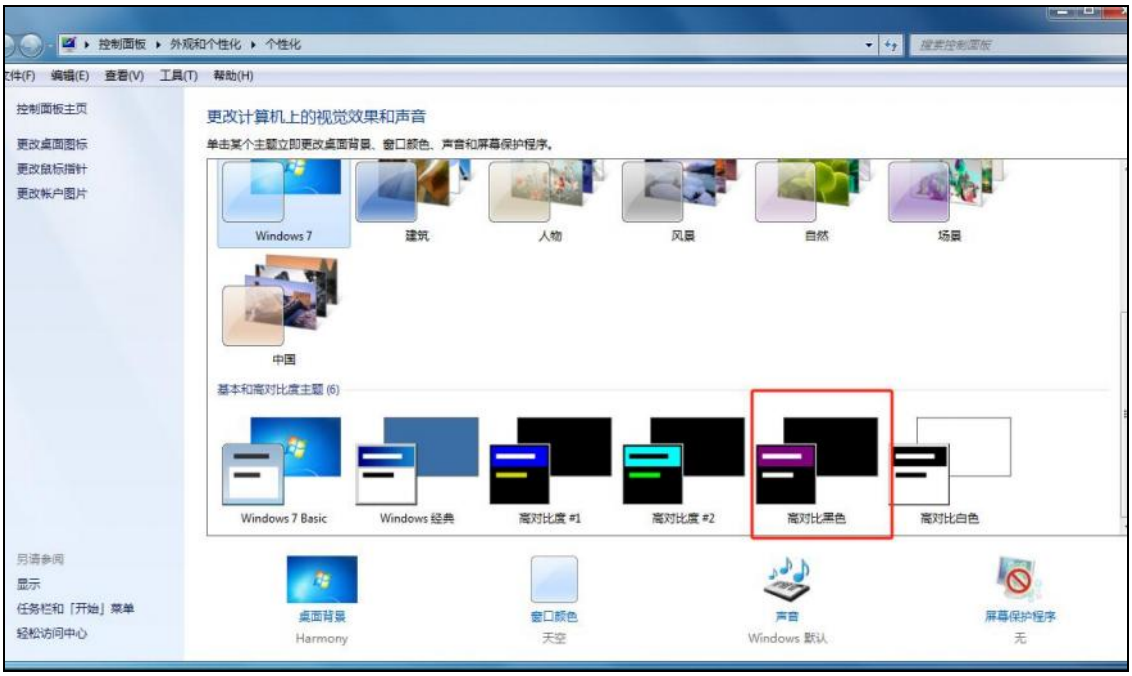


附录U　设置屏幕显示为对比鲜明的黑底白字

U.1　Win7 操作系统：

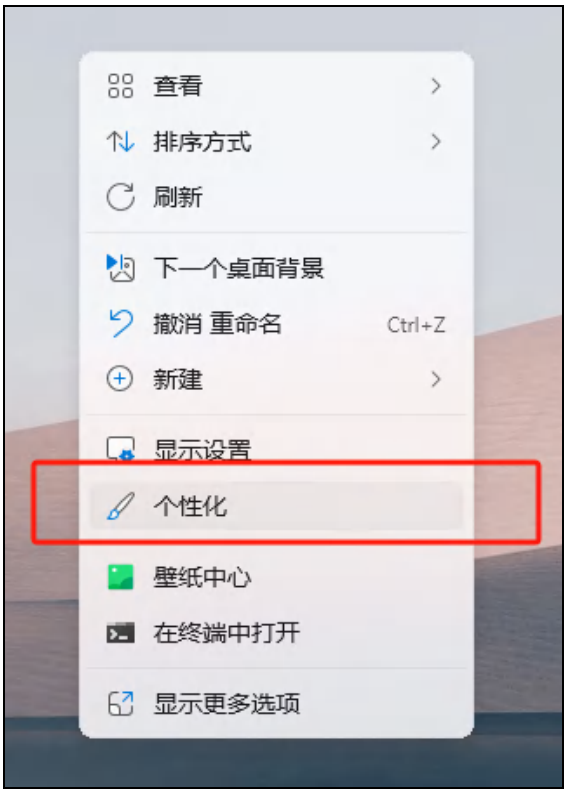
【步骤一】在屏幕空白处单击鼠标右键，出现下图，选择“个性化”：

【步骤二】点击“个性化”，出现下图。在“基本和高对比度主题(6)”下，点击“高对比黑色”，即可完成想要的黑底白字效果。如果想恢复默认设置，则点击第一个图标“Windows 7 Basic”,即可。



U.2　Win11 操作系统：

【步骤一】在屏幕空白处单击鼠标右键，出现下图，点击“个性化”：



【步骤二】点击“个性化”，出现下图，点击“背景”：

【步骤三】在“背景”标签下，点击“对比度主题”

【步骤四】在“对比度主题”标签中，按下图标注的顺序设置，即可完成想要的黑底白字效果。如果想恢复默认设置，则在本步骤的“对比度主题”下拉菜单下选择“无”,然后点击“应用”，就可恢复默认设置。

附录V　Windows 系统页面字体放大功能

Win7 系统只能页面局部放大，Win11 系统可以全屏放大。下面就 Win11 系统介绍使用放大镜功能。

使用快捷键：同时按下键盘上的【“Windows 徽标键”+加号键（+）】可以打开 Windows 系统自带的放大镜窗口，如图所示。可以用鼠标点击调整放大倍数。若要关闭此功能，直接关闭放大镜窗口，也可同时按下【“Windows 徽标键”+“Esc”】即可关闭放大镜功能。

附录W 更新记录

- **2025/07/14**

1. **重大利好：Ai8051U-LQFP48 特价￥1.90**
2. 增加章节：定时器 3/4 功能脚切换

- **2025/07/09**

1. 增加章节：范例程序--使用 RTC 实时时钟中断唤醒省电模式
2. 增加：在各管脚图中增加“USB 直接下载”描述
3. 更新：电源控制寄存器（PCON）的注和特别注意事项
4. 更新：省电模式相关的范例程序

- **2025/07/08**

1. 更新：Ai8051U 系列单片机 P2P（外设到外设 DMA）的使用注意事项

- **2025/07/04**

1. 增加 Ai8051U 系列单片机 P2P（外设到外设 DMA）的使用注意事项

- **2025/06/30**

1. 增加章节：用【USB 转串口的下载器】下载，不要勾选【下次使用 HID 接口进行 ISP 下载】
2. 增加章节：USB-TypeC 封装尺寸图，原理图
3. 增加章节：EAXFR 允许访问扩展的 XFR，取指令不减速，访问片内部 XRAM 不减速
4. 更新：I/O 章节的 I/O 注意事项
5. 更新：部分章节标题

- **2025/06/27**

1. 增加章节：如果使用 3.3V-LDO 供电，建议用有电源关断功能的 6211 / 6231
2. 增加章节：单片机命名规则
3. 更新：各管脚图中 P4.7-nRST 复位脚的说明

- **2025/06/24**

1. 增加章节：BL51/LX51 链接器和扩展中断号工具，REMOVEUNUSED 说明
2. 增加章节：大学实验--串口 1_2_3_4 + 定时器 0_1_3_4 + 外部中断 INT0_1_2_3 (中断设置标志位，主循环查询)，演示实验
3. 增加：I/O 中断掉电唤醒/省电模式唤醒说明

4. 增加：I/O 口数字通道使能寄存器（PxIE）对照表
5. 修正：指令详解中个别笔误

● 2025/06/19

1. 增加章节：大学实验--串口 1_2_3_4(中断设置标志位，主循环查询)，演示实验
2. 更新 EEPROM 触发寄存器（IAP_TRIG）的说明并更新相关范例程序

● 2025/06/12

1. 增加章节：深大实验--硬件 SPI 驱动 OLED 显示图片、图形、字符
2. 更新 IRC 稳定等待时钟数控制寄存器（IRCDB）的说明
3. 更新省电模式相关的范例程序

● 2025/06/10

1. 增加章节：大型综合实战--大学课程设计：硬件 SPI 读写串行 Flash

● 2025/06/05

1. 更新：硬件 USB 下载线路图中增加关于 P3.2 的注意事项

● 2025/05/30

1. 更新：部分章节标题

● 2025/05/29

1. 增加章节：变量的声明要求在可执行语句之前，C89 标准，KEIL C 语言
2. 增加章节：内部高速 IRC 的 4 个频段的中心频率，供参考
3. 增加章节：深大--定时器 0/1，INT0/INT1，中断组合应用实验，演示程序
4. 更新： 中断向量表及中断优先级描述

● 2025/05/28

1. 增加章节：深大实验-串口通讯(USB-CDC 发命令控制硬件 UART1 与 UART2 多机通讯)

● 2025/05/27

1. 增加章节：深大实验-串口通讯(串口通讯，串口绘图，ADC 检测按键，ADC 热敏电阻测温)
2. 增加章节：深大实验-串口通讯(硬件 UART1 与 UART2 收发通讯-奇偶校验)
3. 更新： 《深大--中断实验 1》和《深大--中断实验 2》实验代码中增加供主循环查询的中断标志位，在主循环中处理中断事件任务，不堵塞其他中断

● 2025/05/23

1. 增加章节：深大实验-串口通讯(硬件 UART1 与电脑收发通讯)
2. 增加章节：深大实验-串口通讯(硬件 UART1 与 UART2 数据透传)
3. 更新：　　RTC 寄存器（YEAR、MONTH、DAY、HOUR、MIN、SEC、SSEC）名称修改为（RTCYEAR、RTCMONTH、RTCDAY、RTCHOUR、RTCMIN、RTCSEC、RTCSSEC）

● 2025/05/22

1. 增加章节：深大实验-串口通讯(硬件 UART1 与 UART2 收发通讯)
2. 更新：日志文件使用说明

● 2025/05/21

1. 增加章节：大型综合实战--深大课程设计：硬件 UART1 发送按键码，接收端蜂鸣器根据收到的键码发声

● 2025/05/15

1. 增加章节：实战硬件 SPI 控制 74HC595 驱动数码管显示模拟 RTC 时钟
2. 增加章节：实战硬件 SPI+DMA 驱动 WS2812 彩灯
3. 增加章节：深大实验 -- 硬件 SPI 控制 74HC595 驱动数码管显示模拟 RTC 时钟

● 2025/05/12

1. 增加章节：深大实验 -- ADC 中断方式检测 NTC 热敏电阻测温+数码管显示

● 2025/05/09

1. 增加章节：大型综合实战--深大课程设计：时钟+数码管+ADC 按键扫描
2. 更新本文封面

● 2025/05/07

1. 更新：Ai8052U 系列管脚图

● 2025/04/30

1. 增加章节：大型综合实战--深大课程设计：时钟+数码管+矩阵按键+蜂鸣器发声 -- 流程图
2. 更新：使用【AiCube】工具创建 -- printf_usb("Hello World !\r\n")，第一个完整的 C 语言程序
3. 更新：使用【AiCube】工具创建 -- 查询方式，查询到电脑命令后，printf_usb("Hello World !\r\n")或其他
4. 更新：使用【AiCube】工具创建 -- 中断方式，单片机 USB 中断接收服务函数收到命令后，printf_usb("Hello World !\r\n")或其他”

● 2025/04/27

1. 增加章节：ADC 内部结构图（ADC 框图）

2025/04/25

1. 调整了部分目录标题，更容易理解

2025/04/23

1. 增加章节：深大--定时器 0/1，24 位定时器实验(实战：时钟+数码管+矩阵按键扫描+蜂鸣器发声)

2025/04/22

1. 增加章节：嵌入汇编代码在 C 语言中，Keil C51 编译器
2. 更新：深圳大学 I/O 口实验 5 的注释

2025/04/15

5. 增加章节：Keil C251 编译器中，C 语言文件里如何嵌套汇编代码
6. 增加章节：使用【AiCube】项目助手创建“printf_usb("Hello World !\r\n")，第一个完整的 C 语言程序”
7. 增加章节：使用【AiCube】项目助手创建“查询方式，查询到电脑命令后，printf_usb("Hello World !\r\n")或其他”
8. 增加章节：使用【AiCube】项目助手创建“中断方式，单片机 USB 中断接收服务函数收到命令后，printf_usb("Hello World !\r\n")或其他”

2025/04/14

1. 增加章节：“点灯实验 4”的实验程序汇编语言代码
2. 更新章节：USB 不停电下载，ISP 软件如何设置

2025/04/11

1. 增加章节：“深圳大学 I/O 口实验 1”的实验程序汇编语言代码
2. 增加章节：“深大--中断实验 1”的实验程序汇编语言代码
3. 更新：USB 库函数的详细说明
4. 更新：ai_usb.h 头文件及 printf_usb()库函数在各实验中的注释

2025/04/09

1. 增加章节：USB 不停电下载，ISP 软件如何设置

2025/04/08

1. 增加章节：附录--关于“深大--定时器 0/1，24 位定时器实验”的定时器解释
2. 更新：补充完善“实验一：printf_usb("Hello World !\r\n")，第一个完整的 C 语言程序”
3. 更新：补充完善“实验二：查询方式，查询到电脑命令后，printf_usb("Hello World !\r\n")或其他”
4. 更新 DMA 章节范例程序

2025/04/07

1. 增加章节：C 语言中 printf()函数打印输出数据时，常用的各种输出格式
2. 增加章节：深大-使用【AiCube】助手实现 RTC 实验(测试 RTC，使用 USB-CDC 打印年/月/日/时/分/秒)
3. 更　　新：实验一、二、三及其他部分章节的标题
4. 更新章节：补充完善“实验二：查询方式，查询到电脑命令后，printf_usb("Hello World !\r\n")或其他”实验步骤

2025/04/03

1. 增加章节：printf()实验一：建立第一个完整的单片机 C 语言项目文件，还是“Hello World！”，从 printf()到 printf_usb()
2. 增加章节：printf()实验二：查询方式，单片机主动查询到电脑命令后，发送“Hello World！”或其他
3. 增加章节：printf()实验三：中断方式，USB 中断接收服务函数收到电脑命令后，发送“Hello World！”或其他
4. 增加章节：深圳大学 I/O 实验--I/O 模式读取按键状态并点灯（【AiCube】助手创建实验 4A）
5. 增加章节：附录--7.4.12 固件版本说明

2025/04/02

1. 增加章节：深圳大学 I/O 实验--I/O 模式读取按键状态并点灯（【AiCube】助手创建实验 1A）
2. 增加章节：深圳大学 I/O 实验--I/O 模式读取按键状态并点灯（【AiCube】助手创建实验 2A）
3. 增加章节：深圳大学 I/O 实验--I/O 模式读取按键状态并点灯（【AiCube】助手创建实验 3A）

2025/04/01

1. 增加章节：在 Keil uVision5 的编辑器中，输入中文出现乱码的解决办法
2. 增加章节：使用【AiCube】项目创建助手工具创建通过 USB-CDC 方式控制 Ai8051U 进行 LED 点灯
3. 增加章节：使用【AiCube】项目创建助手工具创建使用 Ai8051U 内置 USB 仿真流水灯
4. 删除章节：关于使用 USB 库时，设置 REMOVEUNUSED 引起的单片机重启问题解决方案

2025/03/31

1. 增加章节：深大--使用【AiCube】助手实现定时器 0/1，24 位定时器实验（定时器中断点灯并通过 USB-CDC 端口输出信息）
2. 更新：定时器的内部结构图

2025/03/28

1. 增加章节：使用【AiCube】项目创建助手工具创建并实现点亮 LED 灯
2. 增加章节：使用【AiCube】项目创建助手工具创建并实现 LED 灯闪烁

2025/03/26

1. 增加章节：深大--RTC 实验(拓展学习之使用 RTC 功能设定闹钟)
2. 完善细节：深大--定时器 0，24 位定时器实验（定时器中断点灯并通过 USB-CDC 端口输出信息）

2025/03/25

1. 增加章节：深大--定时器 0，24 位定时器实验（定时器中断点灯并通过 USB-CDC 端口输出信息）
2. 更新：使用 USB 库的范例程序，去掉等待 USB 配置完成语句。
3. 更新：定时器的内部结构图
4. 更新：范例程序中 usb_init( )函数增加如下使用说明：

   USB 型单片机从设备，如需要主动向电脑发送数据，
   在执行 USB_SendData( )函数和 printf_usb( )函数时，
   这两个函数已增加了判断电脑是否已正确识别到 USB 从设备的程序。
   如果电脑要主动发送数据给 USB 从设备，电脑自己会主动判断与 USB 从设备是否已正确连接。

2025/03/24

1. 增加：“AI_USB.H”头文件的说明
2. 更新：管脚图中 USART1 和 USART2 的名称
3. 更新：USB 库函数的使用说明

2025/03/21

1. 调整部分目录结构
2. 增加章节：关于“下次使用 HID 接口进行 ISP 下载”的选项说明
3. 更新 USB 库函数使用范例程序，增加关于 USB 库函数“USER_DEVICEDESC，USER_PRODUCTDESC，USER_STCISPCMD”这 3 个变量定义的情况说明

2025/03/20

1. 增加章节：管脚说明
2. 完善细节：深圳大学传统外部中断实验：INT0，INT1，INT2，INT3（实验 1、2）

2025/03/19

1. 增加章节：STCAI 的 USB 库函数使用说明
2. 增加章节：深圳大学传统外部中断实验：INT0，INT1，INT2，INT3（实验 2）
3. 更新：传统外部中断端口说明
4. 更新：“REMOVEUNUSED”关键字的说明

2025/03/18

1. 增加章节：深圳大学传统外部中断实验：INT0，INT1，INT2，INT3（实验 1）

2. 更新：经典单片机 89C52RC/89C58RD+系列组成框图
3. 更新：深圳大学 I/O 实验--I/O 模式读取按键状态并点灯（实验 5）的程序代码

2025/03/13

1. 增加章节：关于使用 USB 库时，设置 REMOVEUNUSED 引起的单片机重启问题解决方案

2025/03/12

1. 增加章节：堆栈不用用户设置@Ai8051U-32Bit
2. 完善细节：深圳大学 I/O 实验--I/O 模式读取按键状态并点灯（实验 3、4、5）

2025/03/11

1. 增加章节：附录--Ai8051U 系列单片机取代 STC89 系列的注意事项
2. 增加章节：附录--Ai8051U 系列单片机取代 STC12C5A60S2 系列的注意事项
3. 更新：I/O 的结构图
4. 完善细节：深圳大学 I/O 实验--I/O 模式读取按键状态并点灯（实验 1--5）

2025/03/10

1. 增加章节：认识 Keil 软件的 Build Toolbar（Build 工具栏）
2. 增加章节：实验 5--I/O 口行列按键扫描，蜂鸣器发声大型实战
3. 完善细节：深圳大学 I/O 实验--I/O 模式读取按键状态并点灯（实验 1--4）

2025/03/07

1. 增加章节：深圳大学 I/O 实验--I/O 模式读取按键状态并点灯（实验 1--4）
2. 更新：Ai8052U 系列管脚图
3. 更新：“实验四--使用 Ai8051U 内置 USB 直接仿真流水灯”部分图片
4. 调整部分目录的结构

2025/03/06

1. 增加章节：实验四--使用 Ai8051U 内置 USB 直接仿真流水灯
2. 更新：高级 PWM 相关范例程序

2025/03/05

1. 在第 1 章增加了：什么是单片机/微控制器，Ai8051U 是 32 位机的说明
2. 增加：【USB 下载方法二】备注

2025/03/04

1. 调整部分目录的结构

- **2025/03/03**

1. 增加章节：实验三--通过 USB-CDC 方式控制 Ai8051U 进行 LED 点灯
2. 增加：下载 Keil 中断号拓展工具，及安装
3. 增加：“关于回流焊前是否要烘烤”中“STCAI 贴片 MCU 最小真空包装”内容
4. 调整部分章节的位置

- **2025/02/28**

1. 增加章节：在 Keil 中新建一个 8 位 8051 项目
2. 增加章节：实验二--让 LED 灯闪烁起来
3. 修正文档中错别字、乱码、笔误等

- **2025/02/27**

1. 增加附录：STCAI Reliability Report V1.2（可靠性参考数据，安全参数）
2. 增加：实验的视频链接
3. 更新：“Ai8052U 的管脚图”还在更新中…
4. 删除：原附录 B：逻辑代数的基础

- **2025/02/26**

1. 更新章节：在 Keil 中新建项目
2. 更新章节：实验一--点亮 LED 灯
3. 调整部分章节位置

- **2025/02/25**

1. 增加章节：在 Keil 中新建项目
2. 增加章节：实验--点亮 LED 灯
3. 增加部分程序代码的注释

- **2025/02/14**

1. 更新：“高级 PWM 一个端口做外部中断（同时捕获下降沿和上升沿）”的“注意”
2. 更新：“同时捕获 4 路输入信号的周期和高电平宽度（占空比）”的“范例”
3. 更新：利用 PWM 实现 16 位 DAC 的参考线路图

- **2025/02/12**

1. 增加：高级 PWM 端口做外部中断（同时捕获下降沿和上升沿）的范例程序

- **2025/02/10**

1. 更新：应用注意事项

- **2025/02/08**

1. 增加：Keil 小技巧----在多文件项目的源文件中声明并使用外部变量和外部函数

- **2025/01/24**

1. 增加：使用 I/O 和 R-2R 电阻分压实现 DAC 的经典线路图
2. 修正文档中错别字、乱码、笔误等

- **2025/01/14**

1. 修正文档中错别字、乱码、笔误等，统一一个别管脚的标注名称

- **2024/12/30**

1. 增加：PDIP 封装尺寸图
2. 增加 RTC 说明（Ai8051U 无星期功能）
3. 增加主时钟停振/省电模式前需要对 IAP_TPS 寄存器进行初始化的说明

- **2024/12/13**

1. 更新：DPU32（DSP 指令，32 位及 64 位整数运算器）

- **2024/12/12**

1. 更新：内部高速 IRC 时钟恢复振荡到稳定需要等待的时钟数控制寄存器（IRCDB）

- **2024/12/5**

1. 修正 PWMA_ETRPS、PWMB_ETRPS 寄存器的说明

- **2024/11/29**

1. 更新 Ai8052U 两组独立 DAC、4 组运放、4 组比较器结构图

- **2024/11/28**

1. 更新 USB-CDC 不停电自动下载字符串命令的说明

- **2024/11/22**

1. 更新 DMA 章节中各个小节的相关寄存器说明
2. 增加：高速高级 PWM + DMA（组合使用）特别注意事项

- **2024/11/20**

1. 更新 ISP 下载相关硬件选项的说明

● 2024/11/19

1. 更新 USB-Link1D 各种豪华配线的使用场景说明

● 2024/11/15

2. 增加 USB-Link1D 各种豪华配线的使用场景说明

● 2024/11/14

1. 增加复位标志寄存器 SWRSTF 的说明

● 2024/11/13

1. 更新：附录：单总线（1-Wire）
2. 更新 ADC 参考线路图及利用 ADC15 通道的相关注意事项

● 2024/11/04

1. 增加：DPU32（DSP 指令，32 位及 64 位整数运算器）
2. 增加：附录：单总线（1-Wire）
3. 更新 USB-CDC 不停电自动下载字符串命令的说明

● 2024/11/01

1. 更新 Ai8052U 章节中的运放、比较器、ADC 的内部结构图

● 2024/10/30

1. 更新 Ai8052U 章节中的运放、比较器、ADC 的内部结构图
2. 增加 TFT 彩屏 DMA 存储读取的 16 位数据大小端控制寄存器的说明

● 2024/10/29

1. 增加：Ai8052U 两组独立 DAC、4 组运放、4 组比较器结构及应用

● 2024/10/28

1. 增加：利用 74HC595 驱动 8 个数码管(串行扩展,3 根线)，节省 I/O
2. 增加：利用 ADC 做按键扫描节省 I/O 口
3. 更新：在 Keil 中添加 Ai8051U 型号及头文件的说明
4. 增加：附录：Ai8051U 库文件使用

2024/10/25

1. 增加：附录：可以自己去生产的【USB 转双串口】资料

2024/10/23

1. 增加：Ai8052U 系列的选型介绍
2. 增加：Ai8052U 系列的功能脚切换图表

2024/10/21

3. 增加：USB-Link1D 支持 脱机下载，如何免烧录环节
4. 增加：传统芯片下载线路对比图

2024/10/17

1. 增加：USB 转双串口芯片进行烧录/仿真+串口通讯，5V/3.3V 跳线选择
2. 增加：USB 转串口芯片进行全自动烧录/仿真/通信，5V/3.3V 跳线选择
3. 增加：USB-Link1D 支持 脱机下载 说明
4. 增加：USB-Writer1A 编程器/烧录器・支持・插在・锁紧座上・烧录
5. 增加：USB-Writer1A 支持 自动烧录机，通信协议和接口
6. 更新了部分管脚图说明

2024/10/15

1. 增加唤醒省电模式中断源的说明
2. 增加 Keil 中添加 Ai8051U 系列头文件的说明
3. 增加了 3.3V 的 USB 转双串口芯片进行烧录、串口仿真、串口通讯原理图
4. 更新了部分管脚图说明

2024/10/14

1. 增加了 RMB0.95 的 USB 转单串口芯片 Ai8H2K12U 全自动停电/上电 3.3V 下载线路图
2. 增加了 RMB0.95 的 USB 转单串口芯片 Ai8H2K12U 手动停电/上电 3.3V 下载线路图
3. 增加了 RMB0.95 的 USB 转单串口芯片 Ai8H2K12U 手动停电/上电 5V/3.3V 下载线路图
4. 更新 PCA 中断说明
5. 附录中的重要说明中增加 USB 中断允许位的读写说明
6. 增加使用 1117M-33 芯片和 ME6231 芯片进行 3.3V 的 ISP 下载参考电路
7. 更新：ISP 下载流程图（硬件/软件模拟 USB+串口模式）

2024/10/11

1. 增加：省电模式，I/O 口如何设置才省电
2. 更新管脚图

- **2024/10/10**

1. 更新管脚图中功能脚切换说明
2. I/O 章节中增加 3V/5V 器件互连参考线路图

- **2024/9/19**

1. 增加高速 SPI 配置寄存器说明
2. 更新存储器结构框图

- **2024/9/14**

1. 增加附录：USB 原理及应用—O.3、人机交互设备原理

- **2024/9/13**

1. 在附录 K 应用注意事项 K.1 中增加了第 3 点
2. 增加附录：USB 原理及应用—O.5、通信设备类原理
3. 增加附录：USB 原理及应用—O.6、通信设备类程序设计

- **2024/9/12**

1. 增加第 2.2 节：AI8052U-144I-LQFP100/64/48/44
2. 增加附录：USB 原理及应用—O.2、USB2.0 程序设计实现
3. 增加附录：USB 原理及应用—O.4、人机交互设备程序设计

- **2024/9/10**

1. 增加第 12 章“关于 I/O 的注意事项”--“6、内部上拉电阻和外部上拉电阻的部分区别”
2. 增加附录：USB 原理及应用—O.1、USB 协议概述

- **2024/9/5**

1. 更新 USB-Writer1A 的资料及图片

- **2024/9/4**

1. 增加：附录--设置屏幕显示为对比鲜明的黑底白字
2. 增加：附录--Windows 系统页面字体放大功能

- **2024/9/3**

1. 更新部分图片
2. 更新第 33.16.2 节中“代码测试方法”

- **2024/9/2**

1. 更新部分图片

- **2024/8/29**

1. 更新管脚图
2. 增加自动停电上电参考电路

- **2024/8/21**

1. 更新 LOGO 图标
2. 增加软件复位寄存器 IAP_CONTR 的说明
3. 更正 Ai8051U 系列的管脚数
4. 更新 Ai8051U 的管脚图
5. 更新文档中部分寄存器的描述

- **2024/7/18**

1. 增加 TFPU 时钟选择的说明
2. 更正比较器的输入管脚切换

- **2024/7/12**

1. 增加下载需口令的高级功能说明
2. I/O 章节增加将 I/O 设置传统 8051 端口模式的范例程序
3. 增加扩展 CHIPID（CHIPIDX）的内容描述

- **2024/7/10**

1. 增加 P2P（外设到外设）DMA 的矩阵图
2. 增加 I2S 主时钟分频寄存器说明
3. 重新整理各个章节的名称
4. 更新管脚图及部分参考线路图
5. 更正 QFN48 和 QFN64 的封装图

- **2024/7/8**

1. 完成 Ai8051U 系列单片机技术参考手册初版文档

本系列产品标准销售合同

一．产品质量标准：货物为全新正品。符合 ROHS 质量标准。

二．供方责任：如是供方质量问题，经双方确认后，需方退回芯片，有一换一，质保一年。

三．需方责任：

A、验收：在快递送货到时，需方确认数量无误，无芯片散落，无管脚变形，无其他品质异常情况后再签收。如有异常需方不能签收，由快递公司承担责任。一经需方签收，需方就是认可供方已按要求完成该订单，不再有其他连带责任。

B、保管及贴片加工：根据国际湿敏度 3（MSL3）规范的要求，贴片元器件在拆开真空包装后，168 小时内，7 天内，必须回流焊贴片完成。LQFP/QFN/DFN 托盘能耐 100 度以上的高温，拆开真空包装后 7 天内必须回流焊贴片完成，如未完成，回流焊前必须重新烘烤：110~125℃，4~8 个小时都可以。SOP/TSSOP 塑料管耐不了 100 度以上的高温，拆开真空包装后 7 天内必须回流焊贴片完成，否则回流焊前先去除耐不了 100 度以上高温的塑料管，放到金属托盘中，重新烘烤：110~125℃，4~8 个小时都可以

由于经常有客户退回来的货物中含有来历不明产品，且贴片原器件拆开真空包装后，需要在 168 小时/7 天内完成回流焊贴片工序。

我司无产能对退回器件再进行重新详细检测，再进行重新烘烤，无能力对客户退回的所谓未拆封芯片进行评估，为保证全体客户的利益，产品一经出库，概不退换，以确保品质，确保所有客户的安全。

四．解决纠纷方式：对本合同不详尽之处或产生争议，双方协商解决。协商不成在供方所在地申请仲裁。

五．其他条款：合同一式两份。自双方签署起生效。供方若因外力因素而导致无法交货，供方应及时通知需方，并重新协商本合同相关事宜，需方免除供方应承担的义务。本合同未能列入条款可在合同附件详细列入。

六．本合同双方代表签字且款到后方可生效。

备注：如特殊情况，买方买的型号要更换成其他型号，供方也同意的：

1，开机 13 小时高温烘烤，　1000 元一次

2，开机测试 RMB500 一次，　+0.2 元/片

产　品　授　权　书

致：江苏国芯科技有限公司

深圳国芯人工智能有限公司设计的集成电路产品的知识产权归深圳国芯人工智能有限公司所有。现授权江苏国芯科技有限公司可从事我司产品在中国的推广和销售工作。

授权单位：　深圳国芯人工智能有限公司

授权至有效期：　2030 年 1 月 1 日前

附录X　　自主产权，生产可控

自主产权，生产可控

深圳国芯人工智能有限公司是中华人民共和国大陆独资企业，按中国法律法规独立运营的企业，注册地址在深圳市前海深港合作区前湾一路 1 号 A 栋 201 室。

本手册所描述的器件是在中国境内自主研发，具备独立自主知识产权。

产品核心研发在中国境内，具备芯片设计、封装设计、结构设计、可靠性设计、器件仿真、工艺模拟等全部设计能力；产品核心研发团队人员及带头人全部为我国境内人员组成，其中研发团队带头人研发从业年限十年以上，具备长期、稳定的后续支持能力，具有在我国境内申请的专利证书及软件著作权等。

晶圆制造：本器件设计完成后的晶圆制造加工，在中华人民共和国大陆境内的晶圆厂加工制造完成，受中华人民共和国法律法规管理监管和控制，完全可控。

封装制造：本器件设计完成后的封装制造，在中华人民共和国大陆境内的封装厂加工完成，受中华人民共和国法律法规管理监管和控制，完全可控。

测试：本器件设计完成后的测试，在中华人民共和国大陆境内测试完成，受中华人民共和国法律法规管理监管和控制，完全可控。

本器件全部关键工艺均在我国自有生产线上完成，可以长期供货，无被断供的困扰。

特此说明。

深圳国芯人工智能有限公司