STC15 系列
单片机原理及应用



扫码去微信小商城

推荐的技术交流论坛：www.STCAIMCU.com
资料更新日期：2025/7/2
(本文档可直接添加备注和标记)

目 录

1 单片机基础概述.....1
1.1 什么是单片机/微控制器.....1
1.1.1 经典单片机 89C52RC/89C58RD+系列组成框图.....1
1.1.2 经典单片机 12C5A60S2 组成框图.....2
1.1.3 STC15 系列内部结构图.....3
1.1.3.1 STC15F2K60S2 内部结构图.....3
1.1.3.2 STC15W4K32S4 内部结构图.....4
1.1.4 Ai8051U 内部结构图.....5
1.1.4.1 Ai8051U 8-Bit 内部结构图，内部总线工作在 8 位宽度.....5
1.1.4.2 Ai8051U 32-Bit 内部结构图，内部总线工作在 32 位宽度.....6
1.1.5 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 下载软件及仿真的介绍.....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 C51 集成开发环境.....19
2.2.2 安装 Keil C251 集成开发环境.....23
2.2.3 Keil 的 C51、C251 和 MDK 可安装在同一台电脑同一个目录.....26
2.2.4 从哪购买 KEIL，获取无限制版 KEIL.....27
2.3 安装 AIapp-ISP 下载/编程/烧录/软件,含强大的辅助开发工具包.....28
2.3.1 安装 AIapp-ISP 下载/编程/烧录/软件，取代 STC-ISP.....28
2.3.2 STC 单片机上电工作过程.....30
2.3.3 ISP 下载流程图（硬件 USB+串口模式）.....31
2.4 添加型号和头文件到 Keil.....32
2.5 在 Keil 中新建一个 8051 项目.....34
2.5.1 准备工作，见前面几小节.....34
2.5.2 Keil 新建一个 8 位 8051 项目.....35
2.5.2.1 新建工程.....35
2.5.2.2 8 位 8051 工程项目的各种基础选项设置。.....39
2.6 在 Keil uVision5 的编辑器中，输入中文出现乱码的解决办法.....41
2.7 嵌入汇编代码在 C 语言中，Keil C51 编译器.....42
2.8 关于 Keil 软件中 0xFD 问题的说明.....44
2.9 C 语言中 printf()函数打印输出数据时，常用的各种输出格式.....45
2.10 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.....46
2.10.1 如何下载 Keil 中断号拓展工具，及安装.....46
2.10.2 BL51/LX51 链接器和扩展中断号工具，REMOVEUNUSED 说明.....49
2.10.3 如不使用扩展中断号工具，则需借用保留中断号进行中转.....50
2.11 单片机程序中头文件的使用方法.....59
2.12 如何对 IAP15F2K61S2 系列及 IAP15W4K61S4 系列仿真.....61
2.12.1 仿真硬件连接示意图.....61

2.12.2 制作仿真芯片及仿真流程.....62
3 STC15 系列选型简介、特性/价格/管脚图/ISP 下载/编程线路图.....65
3.1 STC15F104W 系列.....65
3.1.1 STC15F104W 系列单片机简介（与 STC15F104E 系列有所不兼容）.....65
3.1.2 内部结构图.....68
3.1.3 STC15F104W 系列单片机选型价格一览表.....69
3.1.4 STC15F104W 系列单片机命名规则.....71
3.1.5 管脚图及下载线路图.....73
3.1.6 管脚说明.....75
3.1.7 USB-Link1D 工具自动停电/上电烧录，串口仿真+串口通讯.....76
3.1.8 【一箭双雕之 USB 转双串口】工具进行烧录，串口仿真+串口通讯.....77
3.2 STC15W104 系列.....78
3.2.1 STC15W104 系列简介.....80
3.2.2 内部结构图.....82
3.2.3 STC15W104 系列单片机选型价格一览表.....83
3.2.4 STC15W104 系列单片机命名规则.....84
3.2.5 管脚图及下载线路图，最小系统（SOP8/DFN8/DIP8）.....86
3.2.6 管脚说明.....88
3.2.7 USB-Link1D 工具自动停电/上电烧录，串口仿真+串口通讯.....89
3.2.8 【一箭双雕之 USB 转双串口】工具进行烧录，串口仿真+串口通讯.....89
3.3 STC15W204S 系列.....90
3.3.1 STC15W204S 系列简介.....90
3.3.2 内部结构图.....93
3.3.3 STC15W204S 系列选型及价格一览表.....94
3.3.4 STC15W204S 系列单片机命名规则.....95
3.3.5 管脚图及下载线路图.....97
3.3.5.1 管脚图及下载线路图，最小系统（SOP16/DIP16）.....97
3.3.5.2 管脚图及下载线路图，最小系统（SOP8）.....98
3.3.6 管脚说明.....99
3.3.7 USB-Link1D 工具自动停电/上电烧录，串口仿真+串口通讯.....100
3.3.8 【一箭双雕之 USB 转双串口】工具进行烧录，串口仿真+串口通讯.....101
3.4 STC15W408AS 系列.....102
3.4.1 STC15W408AS 系列简介.....104
3.4.2 内部结构图.....107
3.4.3 STC15W408AS 系列选型一览表.....108
3.4.4 STC15W408AS 系列单片机封装价格一览表.....109
3.4.5 STC15W408AS 系列单片机命名规则.....110
3.4.6 管脚图及下载线路图.....112
3.4.6.1 管脚图及下载线路图，最小系统（LQFP32）.....112
3.4.6.2 管脚图及下载线路图，最小系统（SOP28/TSSOP28/SKDIP28）.....113
3.4.6.3 管脚图及下载线路图，最小系统（QFN28）.....114
3.4.6.4 管脚图及下载线路图，最小系统（SOP20/DIP20/TSSOP20）.....115
3.4.6.5 管脚图及下载线路图，最小系统（SOP16/DIP16）.....116
3.4.7 管脚说明.....118
3.4.8 USB-Link1D 工具自动停电/上电烧录，串口仿真+串口通讯.....121
3.4.9 【一箭双雕之 USB 转双串口】工具进行烧录，串口仿真+串口通讯.....122

3.5 STC15F2K60S2 系列 123
3.5.1 STC15F2K60S2 系列简介 125
3.5.2 内部结构图 128
3.5.3 STC15F2K60S2 系列单片机选型价格一览表 129
3.5.4 STC15F2K60S2 系列单片机封装价格一览表 131
3.5.5 STC15F2K60S2 系列单片机命名规则 132
3.5.6 管脚图及下载线路图 135
3.5.6.1 管脚图及下载线路图，最小系统（LQFP48/QFN48） 135
3.5.6.2 管脚图及下载线路图，最小系统（LQFP44） 136
3.5.6.3 管脚图及下载线路图，最小系统（PDIP40） 137
3.5.6.4 管脚图及下载线路图，最小系统（LQFP/QFN32） 138
3.5.6.5 管脚图及下载线路图，最小系统（SKDIP28/SOP28） 139
3.5.6.6 管脚图及下载线路图，最小系统（TSSOP20） 140
3.5.7 STC15F2K60S2 系列单片机的管脚说明 143
3.5.8 USB-Link1D 工具自动停电/上电烧录，串口仿真+串口通讯 147
3.5.9 【一箭双雕之 USB 转双串口】工具进行烧录，串口仿真+串口通讯 148
3.6 STC15F408AD 系列 149
3.6.1 STC15F408AD 系列单片机简介 149
3.6.2 内部结构图 152
3.6.3 STC15F408AD 系列单片机选型价格一览表 153
3.6.4 STC15F408AD 系列单片机命名规则 154
3.6.5 管脚图及下载线路图 156
3.6.5.1 管脚图及下载线路图，最小系统（LQFP32） 156
3.6.5.2 管脚图及下载线路图，最小系统（SKDIP28/SOP28） 157
3.6.6 STC15F408AD 系列单片机的管脚说明 160
3.6.7 USB-Link1D 工具自动停电/上电烧录，串口仿真+串口通讯 163
3.6.8 【一箭双雕之 USB 转双串口】工具进行烧录，串口仿真+串口通讯 164
3.7 STC15W4K32S4 系列 165
3.7.1 STC15W4K32S4 系列简介 167
3.7.2 内部结构图 171
3.7.3 STC15W4K32S4 系列选型一览表 172
3.7.4 STC15W4K32S4 系列单片机封装价格一览表 173
3.7.5 STC15W4K32S4 系列单片机命名规则 174
3.7.6 管脚图 176
3.7.6.1 管脚图及下载线路图，最小系统（LQFP64/QFN64） 176
3.7.6.2 管脚图及下载线路图，最小系统（LQFP48/QFN48） 177
3.7.6.3 管脚图及下载线路图，最小系统（LQFP44） 178
3.7.6.4 管脚图及下载线路图，最小系统（PDIP40） 179
3.7.6.5 管脚图及下载线路图，最小系统（LQFP32/QFN32） 180
3.7.6.6 管脚图及下载线路图，最小系统（SKDIP28/SOP28） 181
3.7.7 管脚说明 185
3.7.8 USB-Link1D 工具自动停电/上电烧录，串口仿真+串口通讯 190
3.7.9 【一箭双雕之 USB 转双串口】工具进行烧录，串口仿真+串口通讯 191
4 STC15 系列单片机封装尺寸图，USB-TypeC 封装尺寸图/原理图 192
4.1 SOP8 封装尺寸图 192
4.2 DFN8 封装尺寸图（3mm*3mm） 193

4.3 SOP16 封装尺寸图 194
4.4 SOP20 封装尺寸图 195
4.5 TSSOP20 封装尺寸图 196
4.6 QFN20 封装尺寸图（3mm*3mm） 197
4.7 SOP28 封装尺寸图 198
4.8 TSSOP28 封装尺寸图 199
4.9 LQFP32 封装尺寸图（9mm*9mm） 200
4.10 QFN32 封装尺寸图（4mm*4mm） 201
4.11 PDIP40 封装尺寸图 202
4.12 LQFP44/QFP44 封装尺寸图（12mm*12mm） 203
4.13 LQFP48/QFP48 封装尺寸图（9mm*9mm） 204
4.14 QFN48 封装尺寸图（6mm*6mm） 205
4.15 LQFP64 封装尺寸图（12mm*12mm） 206
4.16 QFN64 封装尺寸图（8mm*8mm） 207
4.17 USB-TypeC 封装尺寸图，原理图 208
5 STC15W4K32S4 系列与 STC15F/L2K60S2 系列单片机的区别 209
6 特殊外围设备(CCP/SPI，串口 1/2/3/4)在不同口间进行切换 211
6.1 CCP/PWM/PCA 在多个口之间切换的测试程序(C 和汇编) 213
6.2 PWM2/3/4/5/PWMFLT 在多个口之间切换的测试程序(C 和汇编) 214
6.3 PWM6/PWM7 在多个口之间切换的测试程序(C 和汇编) 215
6.4 SPI 在多个口之间切换的测试程序(C 和汇编) 216
6.5 串口 1 在多个口之间切换的测试程序(C 和汇编) 218
6.6 串口 2 在多个口之间切换的测试程序(C 和汇编) 220
6.7 串口 3 在多个口之间切换的测试程序(C 和汇编) 221
6.8 串口 4 在多个口之间切换的测试程序(C 和汇编) 222
7 每个单片机具有全球唯一身份证号码(ID 号)及其测试程序 224
8 关于 ID 号在大批量生产中的应用方法(较多用户的用法) 229
9 在全球唯-ID 号前添加软复位指令及重要测试参数 230
10 如何识别芯片版本号 232
11 现供货的 STC15 系列中未实现的计划功能 233
11.1 现供货的 STC15F2K60S2 系列 C 版本中未实现的计划功能 233
11.1.1 现供货 STC15F2K60S2 系列 C 版本主时钟输出只可对外输出内部 R/C 时钟 233
11.1.2 现供货的 STC15F2K60S2 系列 C 和 D 版本的串口 1 和串口 2 的接收管脚不能唤醒掉电/停机模式 233
11.2 现供货的 STC15F408AD 系列 C 版本中未实现的计划功能 235
11.2.1 现供货的 STC15F408AD 系列 C 版本的串口 1 接收管脚不能唤醒掉电模式/停机模式 235
12 部分 15 系列单片机的特别注意事项 236
12.1 SPI 的特别注意事项（仅针对以 15F 和 15L 开头的单片机） 236
12.2 进入掉电唤醒模式的特别注意事项（仅针对以 15L 开头的单片机） 236
12.3 STC15W204S 系列 A 版本单片机的比较器下降沿中断不响应 236
12.4 STC15W408S 及 STC15W1K16S 系列 T0CLKO 时钟输出功能的注意事项 237
12.5 STC15W4K32S4 系列 A 版单片机的特别注意事项 237
12.6 STC15W4K32S4 系列 B 版单片机的特别注意事项 238
13 STC15 系列的时钟、复位及省电模式 239
13.1 STC15 系列单片机的时钟 239

13.1.1 STC15 系列单片机的内部可配置时钟 239
13.1.2 主时钟分频和分频寄存器....240
13.1.3 可编程时钟输出（也可作分频器使用）....242
13.1.3.1 与可编程时钟输出有关的特殊功能寄存器....242
13.1.3.2 主时钟输出及测试程序（C 和汇编）....247
13.1.3.3 定时器 0 对系统时钟或外部引脚 T0 的时钟输入进行可编程分频输出及测试程序 250
13.1.3.4 定时器 1 对系统时钟或外部引脚 T1 的时钟输入进行可编程分频输出及测试程序 253
13.1.3.5 定时器 2 对系统时钟或外部引脚 T2 的时钟输入进行可编程分频输出及测试程序 256
13.1.3.6 定时器 3 对系统时钟或外部引脚 T3 的时钟输入进行可编程分频输出及测试程序 258
13.1.3.7 定时器 4 对系统时钟或外部引脚 T4 的时钟输入进行可编程分频输出及测试程序 259
13.2 复位....261
13.2.1 外部 RST 引脚复位....261
13.2.2 软件复位及其测试程序(C 和汇编)....261
13.2.3 掉电复位/上电复位....264
13.2.4 MAX810 专用复位电路复位....264
13.2.5 内部低压检测复位....264
13.2.6 看门狗(WDT)复位....267
13.2.7 程序地址非法复位....271
13.2.8 热启动复位和冷启动复位....272
13.3 STC15 系列单片机的省电模式....273
13.3.1 低速模式及其测试程序(C 和汇编)....275
13.3.2 空闲模式(功耗<1mA)及其测试程序(C 和汇编)....277
13.3.3 掉电模式/停机模式及其测试程序(C 和汇编)....278
13.3.3.1 掉电模式/停机模式被唤醒后程序执行流程说明及测试程序（C 和汇编）....283
13.3.3.2 用掉电唤醒专用定时器唤醒掉电模式/停机模式的测试程序(C 和汇编)....285
13.3.3.3 用外部中断 INT0(上升沿+下降沿)唤醒掉电模式/停机模式测试程序(C 和汇编)287
13.3.3.4 用外部中断 INT1(上升沿+下降沿)唤醒掉电模式/停机模式的测试程序(C 和汇编) 289
13.3.3.5 用外部中断 INT2(下降沿)唤醒掉电模式/停机模式的测试程序(C 和汇编) 291
13.3.3.6 用外部中断 INT3(下降沿)唤醒掉电模式/停机模式的测试程序(C 和汇编) 293
13.3.3.7 用外部中断 INT4(下降沿)唤醒掉电模式/停机模式的测试程序(C 和汇编) 295
13.3.3.8 用 CCP/PCA 扩展的外部中断(下降沿+上升沿)唤醒掉电模式/停机模式的程序.297
13.3.3.9 用串口 1 接收管脚由高到低的变化唤醒掉电模式/停机模式的测试程序(C 和汇编) 301
13.3.3.10 用串口 2 接收管脚由高到低的变化唤醒掉电/停机模式的测试程序(C 和汇编)..305
14 存储器，在哪存储重要参数，全球唯一 ID 号和特殊功能寄存器(SFRS)....309
14.1 程序存储器....309
14.2 数据存储器（SRAM）....310
14.2.1 内部 RAM....310
14.2.2 内部扩展 RAM/XRAM/AUX-RAM 及测试程序....313
14.2.3 使用内部扩展 RAM 的测试程序....315

14.2.4 外部 64K 数据总线- 可外部扩展 64K 字节的数据存储器或外围设备 322
14.2.5 利用并行总线扩展外部 32K SRAM 的应用线路图 325
14.3 特殊功能寄存器(SFRs) 326
14.4 STC15W4K32S4 系列新增特殊功能寄存器(SFRs)表 333
15 STC15 系列单片机的 I/O 口结构 334
15.1 I/O 口各种不同的工作模式及配置介绍 334
15.2 管脚 P1.7/XTAL1 与 P1.6/XTAL2 的特别说明 337
15.3 复位管脚 RST 的特别说明 338
15.4 管脚 RSTOUT_LOW 的特别说明 338
15.5 串行口 1 的中继广播方式 339
15.6 可将 MCU 从掉电模式/停机模式唤醒的外部管脚资源 340
15.7 与 I/O 口有关的特殊功能寄存器及其在程序中的地址声明 341
15.8 STC15 系列单片机 P0/P1/P2/P3/P4/P5 口的测试程序 345
15.9 I/O 口各种不同的工作模式结构框图 351
15.9.1 准双向口（弱上拉）输出配置 351
15.9.2 强推挽输出配置 352
15.9.3 高阻输入（电流既不能流入也不能流出）配置 352
15.9.4 开漏输出配置（若外加上拉电阻，也可读外部状态或输出高电平） 352
15.10 一种典型三极管控制电路 354
15.11 典型发光二极管控制电路 354
15.12 混合电压供电系统 3V/5V 器件 I/O 口互连 354
15.13 I/O 口的外部输入何时低（0.8V 以下）何时高电平（2.2V 以上） 355
15.14 如何让 I/O 口上电复位时为低电平 355
15.15 PWM 输出时 I/O 口的状态 357
15.16 I/O 口行列式按键扫描应用线路图 357
15.17 74HC595 管脚介绍及逻辑表 358
15.18 利用 74HC595 扩展 I/O 口的线路图(串行扩展，3 根线) 359
15.19 利用 74HC595 驱动 8 个数码管(串行扩展,3 根线)的线路图 360
15.20 利用普通 I/O 口控制 74HC595 驱动 8 个数码管的测试程序 360
15.21 I/O 口直接驱动 LED 数码管应用线路图 367
15.22 用 STC MCU 的 I/O 口直接驱动段码 LCD 的原理及扫描程序 368
15.23 A/D 做按键扫描应用线路图 378
15.24 STC15 系列单片机 I/O 口软件模拟 I²C 接口的测试程序 379
15.24.1 STC15 系列单片机 I/O 口软件模拟 I²C 接口的主机模式 379
15.24.2 STC15 系列单片机 I/O 口软件模拟 I²C 接口的从机模式 382
16 指令系统 384
16.1 寻址方式 384
16.1.1 立即寻址 384
16.1.2 直接寻址 384
16.1.3 间接寻址 384
16.1.4 寄存器寻址 384
16.1.5 相对寻址 385
16.1.6 变址寻址 385
16.1.7 位寻址 385
16.2 完整指令集对照表(与传统 8051 对照) 386
16.3 传统 8051 单片机指令定义详解(中文&English) 392

16.3.1 传统 8051 单片机指令定义详解.....392
16.3.2 Instruction Definitions of Traditional 8051 MCU.....425
17 中断系统.....459
17.1 STC15 系列单片机的中断请求源.....460
17.1.1 STC15F104W 系列单片机的中断请求源.....460
17.1.2 STC15F408AD 系列单片机的中断请求源.....460
17.1.3 STC15W204S 系列单片机的中断请求源.....461
17.1.4 STC15W408AS 系列单片机的中断请求源.....461
17.1.5 STC15W404S 系列单片机的中断请求源.....461
17.1.6 STC15W1K16S 系列单片机的中断请求源.....461
17.1.7 STC15F2K60S2 系列单片机的中断请求源.....461
17.1.8 STC15W4K32S4 系列单片机的中断请求源.....462
17.2 中断结构图.....463
17.3 中断向量入口地址/查询次序/优先级/请求标志/允许位表.....466
17.4 在 Keil C 中如何声明中断函数.....467
17.5 中断寄存器.....468
17.6 中断优先级.....481
17.7 中断处理.....482
17.8 中断嵌套.....483
17.9 外部中断.....484
17.10 中断的测试程序(C 和汇编).....485
17.10.1 外部中断 0(INT0)的测试程序.....485
17.10.1.1 外部中断 INT0(上升沿+下降沿)的测试程序(C 和汇编).....485
17.10.1.2 外部中断 INT0(下降沿)的测试程序(C 和汇编).....486
17.10.2 外部中断 1(INT1)的测试程序.....488
17.10.2.1 外部中断 INT1(上升沿+下降沿)的测试程序(C 和汇编).....488
17.10.2.2 外部中断 INT1(下降沿)的测试程序(C 和汇编).....489
17.10.3 外部中断 2(INT2)(下降沿中断)的测试程序(C 和汇编).....490
17.10.4 外部中断 3(INT3)(下降沿中断)的测试程序(C 和汇编).....492
17.10.5 外部中断 4(INT4)(下降沿中断)的测试程序(C 和汇编).....493
17.10.6 T0 扩展为外部下降沿中断的测试程序(C 和汇编).....495
17.10.7 T1 扩展为外部下降沿中断的测试程序(C 和汇编).....496
17.10.8 T2 扩展为外部下降沿中断的测试程序(C 和汇编).....498
17.10.9 用 CCP/PCA 功能扩展外部中断的测试程序(C 和汇编).....500
18 定时器/计数器.....504
18.1 定时器/计数器的相关寄存器.....505
18.2 定时器/计数器 0 工作模式.....512
18.2.1 模式 0(16 位自动重装载模式)及测试程序，建议只学习此模式足矣.....512
18.2.1.1 定时器 0 的 16 位自动重装载模式的测试程序(C 和汇编).....513
18.2.1.2 定时器 0 对系统时钟或外部引脚 T0 的时钟输入进行可编程分频输出的测试程序 515
18.2.1.3 T0 的 16 位自动重装模式(软硬结合)模拟 10 位或 16 位 PWM 输出的程序(C 和汇编) 517
18.2.1.4 T0 的 16 位自动重装载模式扩展为外部下降沿中断的测试程序(C 和汇编).....519
18.2.2 模式 1（16 位不可重装载模式），不建议学习.....521
18.2.3 模式 2（8 位自动重装载模式），不建议学习.....522

18.2.4 模式 3(不可屏蔽中断 16 位自动重装载,实时操作系统用节拍定时器) 524
18.3 定时器/计数器 1 工作模式.....525
18.3.1 模式 0（16 位自动重装载模式）及测试程序，建议只学习此模式足矣 525
18.3.1.1 定时器 1 的 16 位自动重装载模式的测试程序（C 和汇编） 526
18.3.1.2 定时器 1 对系统时钟或外部引脚 T1 的时钟输入进行可编程分频输出的测试程序 528
18.3.1.3 定时器 1 模式 0(16 位自动重载模式)作串口 1 波特率发生器的测试程序(C 和汇编) 529
18.3.1.4 T1 的 16 位自动重装载模式扩展为外部下降沿中断的测试程序(C 和汇编) 534
18.3.2 模式 1（16 位不可重装载模式），不建议学习.....535
18.3.3 模式 2（8 位自动重装载模式），不建议学习.....536
18.3.3.1 定时器 1 模式 2(8 位自动重载模式)作串口 1 波特率发生器的测试程序(C 和汇编) 536
18.3.3.2 T1 的 8 位自动重装载模式扩展为外部下降沿中断的测试程序(C 和汇编) 541
18.4 古老的 Intel 8051 单片机定时器 0/1 应用举例 543
18.5 定时器/计数器 2 及其应用.....546
18.5.1 定时器/计数器 2 的相关特殊功能寄存器.....546
18.5.2 定时器/计数器 2 作定时器及测试程序（C 和汇编） 549
18.5.2.1 定时器 2 的 16 位自动重载模式的测试程序(C 和汇编).....550
18.5.2.2 定时器 2 扩展为外部下降沿中断的的测试程序(C 和汇编).....552
18.5.3 定时器 2 对系统时钟或外部引脚 T2 的时钟输入进行可编程分频输出.....554
18.5.4 定时器/计数器 2 作串行口波特率发生器及测试程序(C 和汇编).....556
18.5.4.1 定时器/计数器 2 作串行口 1 波特率发生器的测试程序(C 和汇编).....557
18.5.4.2 定时器/计数器 2 作串行口 2 波特率发生器的测试程序(C 和汇编).....562
18.6 定时器/计数器 3 及定时器/计数器 4.....567
18.6.1 定时器/计数器 3 和定时器/计数器 4 的相关特殊功能寄存器.....567
18.6.2 定时器/计数器 3 的应用（STC 创新设计，请不要抄袭）.....569
18.6.2.1 定时器/计数器 3 作定时器.....569
18.6.2.2 定时器/计数器 3 对系统时钟或外部引脚 T3 的时钟输入进行可编程时钟分频输出 570
18.6.2.3 定时器/计数器 3 作串行口 3 的波特率发生器.....570
18.6.3 定时器/计数器 4 的应用（STC 创新设计，请不要抄袭）.....571
18.6.3.1 定时器/计数器 4 作定时器.....571
18.6.3.2 定时器/计数器 4 对系统时钟或外部引脚 T4 的时钟输入进行可编程时钟分频输出 572
18.6.3.3 定时器/计数器 4 作串行口 4 的波特率发生器.....572
18.7 如何将定时器 T0/TT1/T2/T3/T4 的速度提高 12 倍 573
18.8 可编程时钟输出（也可作分频器使用） 575
18.8.1 与可编程时钟输出有关的特殊功能寄存器.....576
18.8.2 主时钟输出及其测试程序（C 和汇编） 580
18.8.3 定时器 0 对系统时钟或外部引脚 T0 的时钟输入进行可编程分频输出.....583
18.8.4 定时器 1 对系统时钟或外部引脚 T1 的时钟输入进行可编程分频输出.....586
18.8.5 定时器 2 对系统时钟或外部引脚 T2 的时钟输入进行可编程分频输出.....589
18.8.6 定时器 3 对系统时钟或外部引脚 T3 的时钟输入进行可编程分频输出.....591
18.8.7 定时器 4 对系统时钟或外部引脚 T4 的时钟输入进行可编程分频输出.....592
18.9 掉电唤醒专用定时器及测试程序（C 和汇编） 593

18.10 外部管脚 T0/T1/T2/T3/T4 如何唤醒掉电模式/停机模式 598
19 串行口通信 599
19.1 串行口 1 的相关寄存器 600
19.2 串行口 1 工作模式 606
19.2.1 串行口 1 工作模式 0：同步移位寄存器（建议初学者不学） 606
19.2.2 串行口 1 工作模式 1：8 位 UART，波特率可变 608
19.2.3 串行口 1 工作模式 2：9 位 UART，波特率固定（建议不学习） 611
19.2.4 串行口 1 工作模式 3：9 位 UART，波特率可变 613
19.3 串行口 1 的波特率设置 616
19.4 串行口 1 的测试程序(C 和汇编) 621
19.4.1 定时器 2 作串口 1 波特率发生器的测试程序(C 和汇编) 621
19.4.2 定时器 1 模式 0(16 位自动重装载)作串口 1 波特率发生器程序(C 和汇编) 626
19.4.3 定时器 1 模式 2(8 位自动重装载)作串口 1 波特率发生器程序(建议不学) 630
19.5 串行口 2 的相关寄存器 636
19.6 串行口 2 工作模式 640
19.6.1 串行口 2 的工作模式 0 ---- 8 位 UART，波特率可变 640
19.6.2 串行口 2 的工作模式 1 ---- 9 位 UART，波特率可变 640
19.7 串行口 2 的测试程序(C 和汇编) 641
19.8 串行口 3 的相关寄存器 647
19.9 串行口 3 工作模式 652
19.9.1 串行口 3 的工作模式 0----8 位 UART，波特率可变 652
19.9.2 串行口 3 的工作模式 1----9 位 UART，波特率可变 653
19.10 串行口 4 的相关寄存器 654
19.11 串行口 4 工作模式 658
19.11.1 串行口 4 的工作模式 0 ---- 8 位 UART，波特率可变 658
19.11.2 串行口 4 的工作模式 1 ---- 9 位 UART，波特率可变 659
19.12 双机通信 660
19.13 多机通信 669
19.14 串口 1 作为增强型串口使用时的自动地址识别功能 674
19.14.1 与串口 1 自动地址识别功能相关的特殊功能寄存器 674
19.14.2 串口 1 自动地址识别功能的介绍 676
19.14.3 串口 1 自动地址识别功能的测试程序（C 和汇编） 678
19.15 串行口 1 的中继广播方式 682
19.16 用 T0 软件模拟串行口的测试程序(C 及汇编) 683
19.17 用 T2 结合 INT4 模拟一个半双工串口的测试程序(C 及汇编) 690
19.18 利用两路 CCP/PCA 模拟一个全双工串口的程序(C 及汇编) 700
20 STC15 系列单片机 EEPROM 的应用 711
20.1 IAP 及 EEPROM 新增特殊功能寄存器介绍 711
20.2 STC15 系列单片机 EEPROM 空间大小及地址 715
20.2.1 STC15W4K32S4 系列单片机 EEPROM 空间大小及地址 715
20.2.2 STC15F2K60S2 及 STC15L2K60S2 系列 EEPROM 空间大小及地址 716
20.2.3 STC15W1K08PWM 系列单片机 EEPROM 空间大小及地址 717
20.2.4 STC15W1K16S 系列单片机 EEPROM 空间大小及地址 717
20.2.5 STC15W404S 系列单片机 EEPROM 空间大小及地址 718
20.2.6 STC15W408AS 系列单片机 EEPROM 空间大小及地址 718
20.2.7 STC15W204S 系列单片机 EEPROM 空间大小及地址 719

20.2.8 STC15W104 系列 EEPROM 空间大小及地址.....719
20.2.9 STC15F101W 及 STC15L104W 系列 EEPROM 空间大小及地址.....720
20.2.10 STC15F408AD 及 STC15L408AD 系列 EEPROM 空间大小及地址.....720
20.3 IAP 及 EEPROM 汇编简介.....722
20.4 EEPROM 测试程序(C 和汇编).....725
20.4.1 EEPROM 测试程序(不用串口送出数据)(C 和汇编).....725
20.4.2 EEPROM 测试程序(使用串口送出数据)(C 和汇编).....731
20.5 比较器作外部掉电检测的参考电路.....740
21 STC15 系列单片机的 A/D 转换器.....742
21.1 A/D 转换器的结构.....742
21.2 与 A/D 转换相关的寄存器.....744
21.3 A/D 转换典型应用线路.....748
21.4 A/D 作按键扫描应用线路图.....749
21.5 A/D 转换模块的参考电压源.....750
21.6 A/D 转换的测试程序(C 和汇编).....751
21.6.1 A/D 转换的测试程序(ADC 中断方式).....751
21.6.2 A/D 转换的测试程序(ADC 查询方式).....756
21.7 利用新增的 ADC 第 9 通道测量内部参考电压的测试程序.....761
21.8 利用新增的 ADC 第 9 通道测量外部电压或外部电池电压.....767
21.9 利用外部 TL431 基准测量外部输入电压值的测试程序.....767
21.10 利用 BandGap 电压精确测量外部输入电压值及测试程序.....781
21.11 利用 SPI 接口扩展 12 位 ADC(TLC2543)的应用线路图.....784
22 STC15 系列 CCP/PCA/PWMIDAC 应用.....785
22.1 与 CCP/PWM/PCA 应用有关的特殊功能寄存器.....786
22.2 CCP/PWM/PCA 模块的结构.....793
22.3 CCP/PCA 模块的工作模式.....795
22.3.1 捕获模式.....795
22.3.2 16 位软件定时器模式.....795
22.3.3 高速脉冲输出模式.....796
22.3.4 脉宽调节模式（PWM）.....797
22.3.4.1 8 位脉宽调节模式(PWM).....797
22.3.4.2 7 位脉宽调节模式(PWM)(STC 创新设计).....799
22.3.4.3 6 位脉宽调节模式(PWM)(STC 创新设计).....800
22.4 用 CCP/PCA 功能扩展外部中断的测试程序(C 和汇编).....801
22.5 用 CCP/PCA 功能实现 16 位定时器的测试程序(C 和汇编).....804
22.6 CCP/PCA 输出高速脉冲的测试程序(C 和汇编).....808
22.7 CCP/PCA 输出 PWM(6 位+7 位+8 位)的测试程序(C 和汇编).....811
22.8 用 CCP/PCA 高速脉冲输出功能实现 3 路 9 ~ 16 位 PWM 的程序.....815
22.9 用 CCP/PCA 的 16 位捕获模式测脉冲宽度的程序(C 和汇编).....829
22.10 用 T0 软硬结合模拟 16 路软件 PWM 的程序(C 及汇编).....834
22.11 用 T0 的时钟输出功能实现 8~16 位 PWM 的程序(C 及汇编).....840
22.12 用 T1 的时钟输出功能实现 8~16 位 PWM 的程序(C 及汇编).....848
22.13 用 T2 的时钟输出功能实现 8~16 位 PWM 的程序(C 及汇编).....856
22.14 利用两路 CCP/PCA 模拟一个全双工串口的程序(C 及汇编).....864
22.15 利用 CCP/PCA 模块实现 8~16 位 DAC 的参考线路图.....875
23 STC15W4K32S4 系列新增 6 通道高精度 PWM.....876

23.1 增强型 PWM 波形发生器相关功能寄存器 877
23.2 增强型 PWM 波形发生器的中断控制 888
23.3 利用 PWM 波形发生器控制舞台灯光的示例程序(C 和汇编) 898
23.4 两通道 CCP/PCA/增强型 PWM 912
23.5 用 STC15W4KxxS4 系列单片机输出两路互补 SPWM 913
23.6 用 STC15W4K 系列的 PWM 实现渐变灯的示例程序 925
24 STC15W 系列的比较器 933
24.1 比较器中断方式程序举例(C 及汇编) 936
24.2 比较器查询方式程序举例(C 及汇编) 939
24.3 比较器作外部掉电检测的参考电路 942
24.4 STC15W 系列比较器作 ADC 的程序举例（C 语言） 943
24.5 在比较器负端产生不同的电压由比较器正端进行比较 948
24.6 现供货的 STC15W204S 系列 A 版本比较器下降沿中断不响应 949
25 使用 STC15 系列单片机的 ADC 做电容感应触摸按键 950
26 同步串行外围接口(SPI 接口) 968
26.1 与 SPI 功能模块相关的特殊功能寄存器 969
26.2 SPI 接口的结构 973
26.3 SPI 接口的数据通信 974
26.3.1 SPI 接口的数据通信方式 975
26.3.2 对 SPI 进行配置 977
26.3.3 作为主机/从机时的额外注意事项 978
26.3.4 通过 SS 改变模式 979
26.3.5 写冲突 979
26.3.6 数据模式 980
26.4 适用单主单从系统的 SPI 功能测试程序(C 和汇编) 982
26.4.1 中断方式 982
26.4.2 查询方式 988
26.5 适用互为主从系统的 SPI 功能测试程序(C 和汇编) 993
26.5.1 中断方式 993
26.5.2 查询方式 999
26.6 利用 SPI 控制 74HC595 驱动 8 位数码管及测试程序(C 和汇编) 1004
26.7 利用 SPI 接口扩展 12 位 ADC(TLC2543)的应用线路图 1013
26.8 利用 STC15 系列单片机 SPI 的主模式读写外部串行 Flash 1014
26.8.1 利用 STC15 系列 SPI 的主模式读写外部串行 Flash 的参考电路图 1014
26.8.2 利用 STC15 系列 SPI 的主模式读写外部串行 Flash 的测试程序 1014
26.8.2.1 通过中断方式利用 SPI 的主模式读写外部串行 Flash 的测试程序(C 和汇编) 1014
26.8.2.2 通过查询方式利用 SPI 的主模式读写外部串行 Flash 的测试程序(C 和汇编) 1033
26.9 SPI 的特别注意事项(仅针对以 15F 和 15L 开头的单片机) 1052
27 编译器(汇编器)/ISP 编程器(烧录)/仿真器说明 1053
27.1 编译器/汇编器的说明及头文件 1053
27.2 USB 型联机/脱机下载工具 U8W/U8W-Mini/U8/U8-Mini 1056
27.2.1 如何安装下载工具 U8W/U8W-Mini/U8/U8-Mini 的驱动程序 1059
27.2.2 USB 型联机/脱机下载工具 U8W 的功能介绍(价格为人民币 100 元) 1064
27.2.3 U8W 的在线联机下载使用说明 1065
27.2.3.1 目标芯片直接安装于 U8W 座锁紧上并由 U8W 连接电脑进行在线联机下载的说明 1065

27.2.3.2 目标芯片通过用户系统引线连接 U8W 并由 U8W 连接电脑进行在线联机下载的说明 1066
27.2.4 U8W 的脱机下载使用说明 1068
27.2.4.1 目标芯片直接安装于 U8W 座锁紧上并通过 USB 连接电脑给 U8W 供电进行脱机下载 1068
27.2.4.2 目标芯片由用户系统引线连接 U8W 并通过 USB 连接电脑给 U8W 供电进行脱机下载 1069
27.2.4.3 目标芯片由用户系统引线连接 U8W 并通过用户系统给 U8W 供电进行脱机下载 1072
27.2.4.4 目标芯片由用户系统引线连接 U8W 且 U8W 与用户系统各自独立供电进行脱机下载 1074
27.2.5 USB 型联机/脱机下载工具 U8 的功能介绍（U8 的价格为人民币 100 元） 1076
27.2.6 U8 的在线联机下载使用说明 1077
27.2.6.1 目标芯片直接安装于 U8 的座锁紧上并由 U8 连接电脑进行在线联机下载的说明 1077
27.2.6.2 目标芯片通过用户系统引线连接 U8 并由 U8 连接电脑进行在线联机下载的说明 1078
27.2.7 U8 的脱机下载使用说明 1080
27.2.7.1 目标芯片直接安装于 U8 座锁紧上并通过 USB 连接电脑给 U8 供电进行脱机下载 1080
27.2.7.2 目标芯片由用户系统引线连接 U8 并通过 USB 连接电脑给 U8 供电进行脱机下载 1082
27.2.7.3 目标芯片由用户系统引线连接 U8 并通过用户系统给 U8 供电进行脱机下载 .1083
27.2.7.4 目标芯片由用户系统引线连接 U8 且 U8 与用户系统各自独立供电进行脱机下载 1085
27.2.8 制作/更新 USB 型联机/脱机下载工具 U8W/U8W-Mini/U8/U8-Mini 1088
27.2.8.1 制作 U8W/U8W-Mini/U8/U8-Mini 下载母片（控制母片） 1088
27.2.8.2 手动升级 U8W/U8W-Mini/U8/U8-Mini 1090
27.2.9 USB 型联机/脱机下载板 U8W/U8W-Mini/U8/U8-Mini 的参考电路 1092
27.3 ISP 编程器/烧录器的说明 1095
27.3.1 在系统可编程(ISP)原理使用说明 1095
27.3.2 STC15 系列在系统可编程(ISP)典型应用线路图 1096
27.3.2.1 利用 RS-232 转换器的 ISP 下载典型应用线路图 1096
27.3.2.2 利用 USB 转串口芯片 PL-2303SA 的 ISP 下载编程典型应用线路图 1098
27.3.2.3 利用 USB 转串口芯片 PL-2303HXD/PL-2303HX 的 ISP 下载编程典型应用线路图 1099
27.3.2.4 STC15W4K 系列及 IAP15W4K58S4 单片机的 USB 直接下载编程线路，USB-ISP 1100
27.3.2.5 利用 U8-Mini 进行 ISP 下载的示意图 1102
27.3.2.6 利用 U8 进行 ISP 下载的示意图 1103
27.3.3 所有 STC 系列单片机封装实物图 1104
27.3.4 AIapp-ISP 下载编程工具硬件——AIapp-ISP 下载板 1107
27.3.4.1 STC15 系列 ISP 下载板实物图 1107
27.3.4.2 如何将单片机安装到 AIapp-ISP 下载板上 1108
27.3.4.3 如何使用转换座将贴片封装的单片机安装到 AIapp-ISP 下载板上 1109
27.3.4.4 如何将 AIapp-ISP 下载板连接到电脑 1114

27.3.5 针对 USB-RS232 转换线不兼容问题的几点说明 1116
27.3.6 如何用 AIapp-ISP 下载板给在用户系统上的单片机烧录用户程序 1117
27.3.7 电脑端的 AIapp-ISP 控制软件（Ver6.95G）的界面使用说明 1119
27.3.8 AIapp-ISP 控制软件（Ver6.95G）发布项目程序使用说明 1126
27.3.9 “程序加密后传输”功能说明 1129
27.3.10 "发布项目程序"+"程序加密后传输"结合使用 1133
27.3.11 运行用户程序时收到用户命令后自动启动 ISP 下载（不停电） 1139
27.3.12 用户接口 1141
27.3.13 RS485 控制 1142
27.3.13.1 RS485 控制使用说明 1142
27.3.13.2 RS485 自动控制或 I/O 口控制下载线路图 1144
27.3.14 “可设下次更新程序时需口令”功能使用说明 1145
27.3.15 STC-USB 驱动程序安装说明 1146
27.3.15.1 Windows XP 操作系统下的 STC-USB 驱动程序安装说明 1146
27.3.15.2 Windows7（32 位）操作系统下的 STC-USB 驱动程序安装说明 1149
27.3.15.3 Windows8（32 位）操作系统下的 STC-USB 驱动程序安装说明 1151
27.3.15.4 Windows 8（64 位）操作系统下的 STC-USB 驱动程序安装说明 1155
27.3.15.5 Windows 8.1（64 位）操作系统下的 STC-USB 驱动程序安装说明 1159
27.4 STC 仿真器说明指南（建议用户串口放在 P3.6/P3.7 或 P1.6/P1.7 上） 1164
27.5 如何让传统的 8051 单片机学习板可仿真 1171
27.6 若无仿真器，如何调试/开发用户程序 1173
28 利用主控芯片对从芯片(限 STC15 系列)进行 ISP 下载 1174
附录 A STC15 系列单片机电气特性 1185
附录 B STC15 系列单片机取代传统 8051 注意事项 1186
附录 C 关于回流焊前是否要烘烤 1189
附录 D 如何使用万用表检测芯片 I/O 口好坏 1190
附录 E 大批量生产，如何省去专门的烧录人员，如何无烧录环节 1191
附录 F 内部常规 256 字节 RAM 间接寻址测试程序 1192
附录 G 用串口扩展 I/O 接口 1193
附录 H 一个 I/O 口驱动发光二极管并扫描按键 1195
附录 I STC15 系列对指令系统的提升 1196
附录 J 如何利用 Keil C 软件减少代码长度 1202
附录 K 使用 STC 的 IAP 系列单片机开发自己的 ISP 程序 1203
附录 L 掉电唤醒定时器频率与电压的关系 1216
附录 M 更新记录 1217

1 单片机基础概述

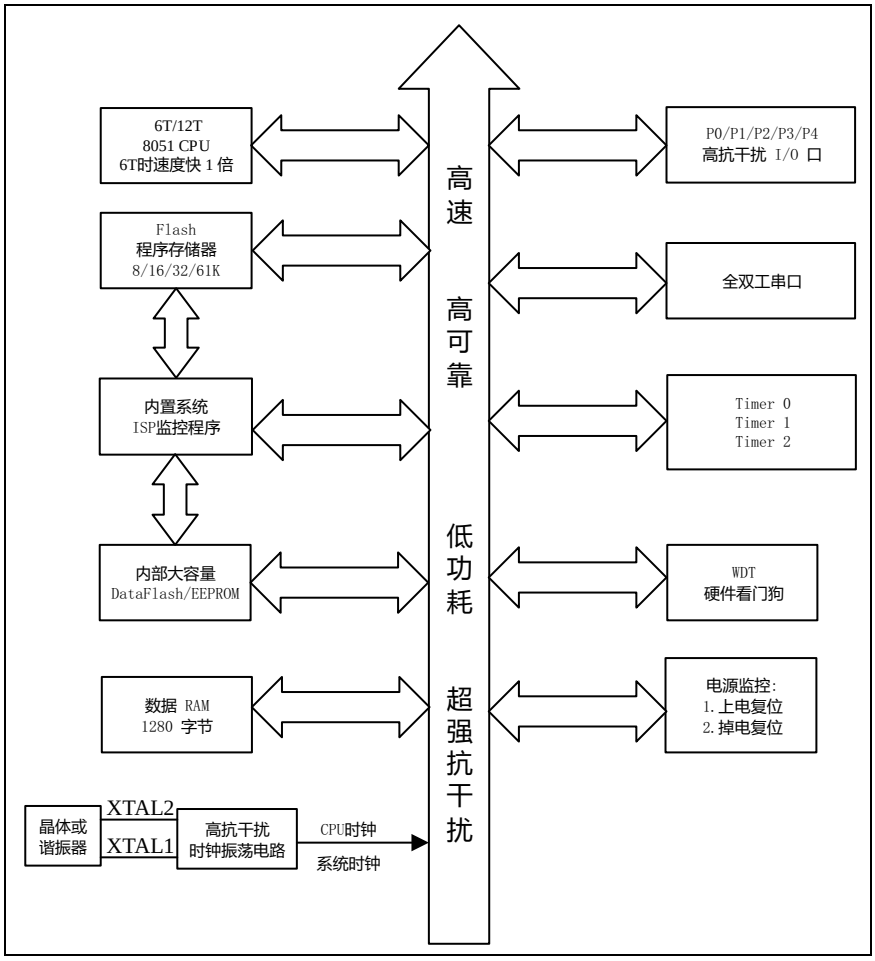
——无微机原理的用户请从本章开始学习

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。

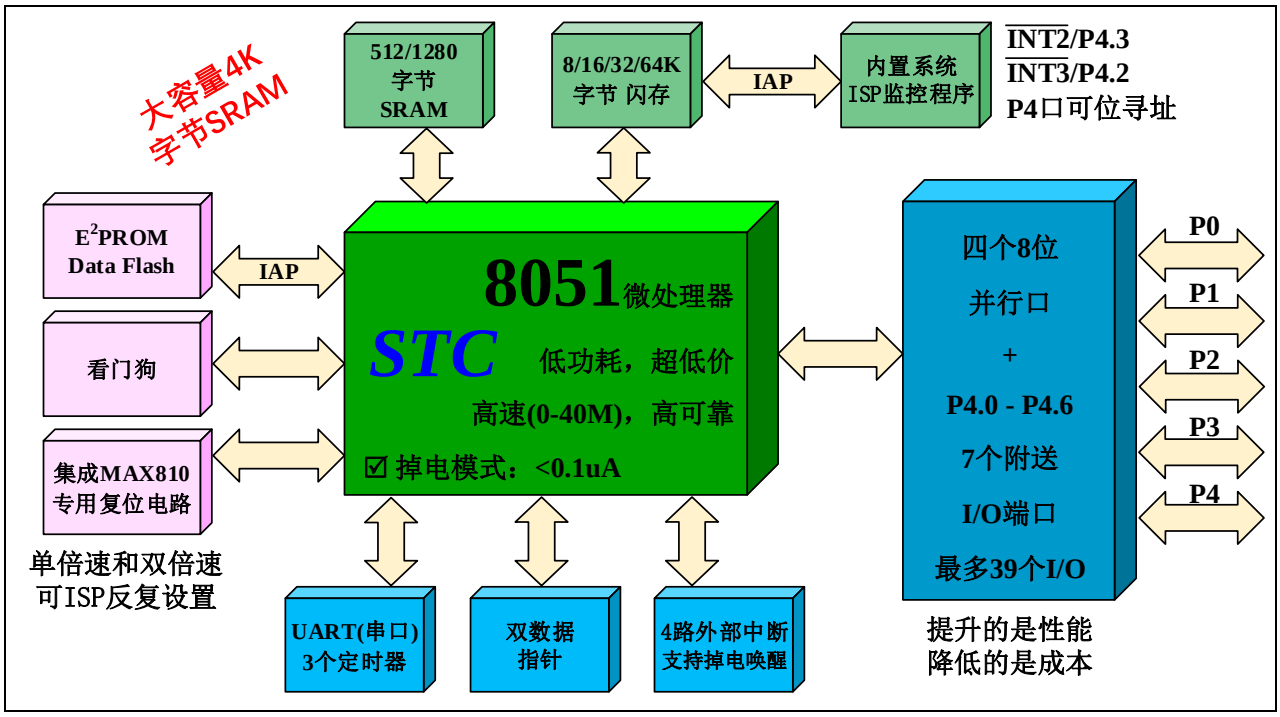
1.1 什么是单片机/微控制器

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。

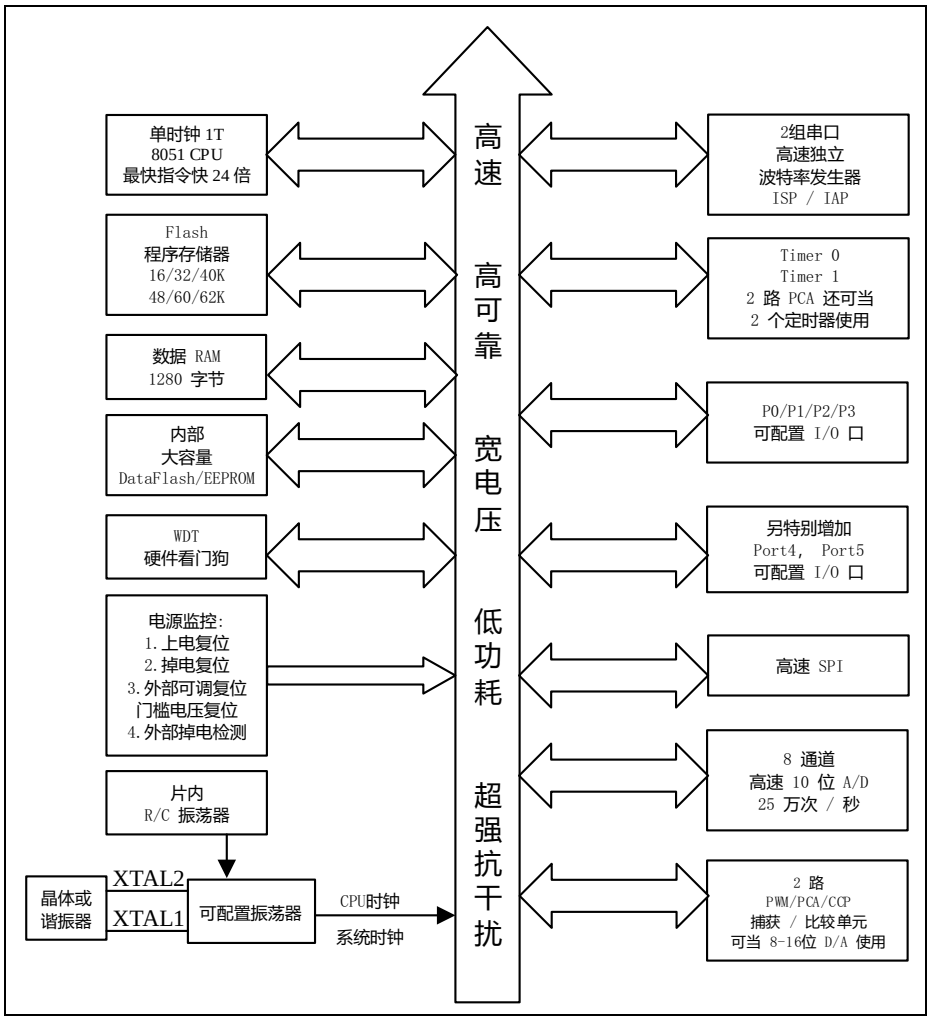
1.1.1 经典单片机 89C52RC/89C58RD+系列组成框图



更通俗的框图：

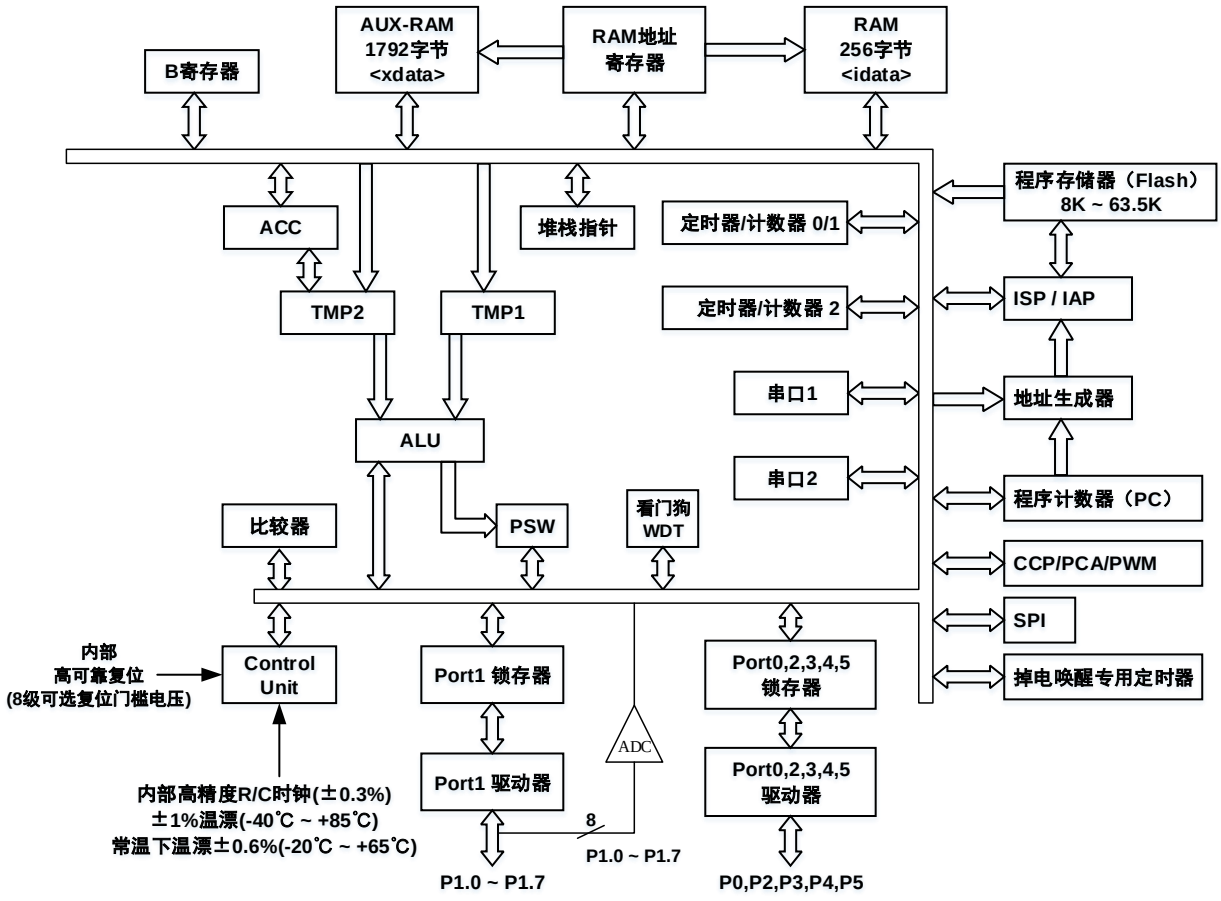


1.1.2 经典单片机 12C5A60S2 组成框图



1.1.3 STC15 系列内部结构图

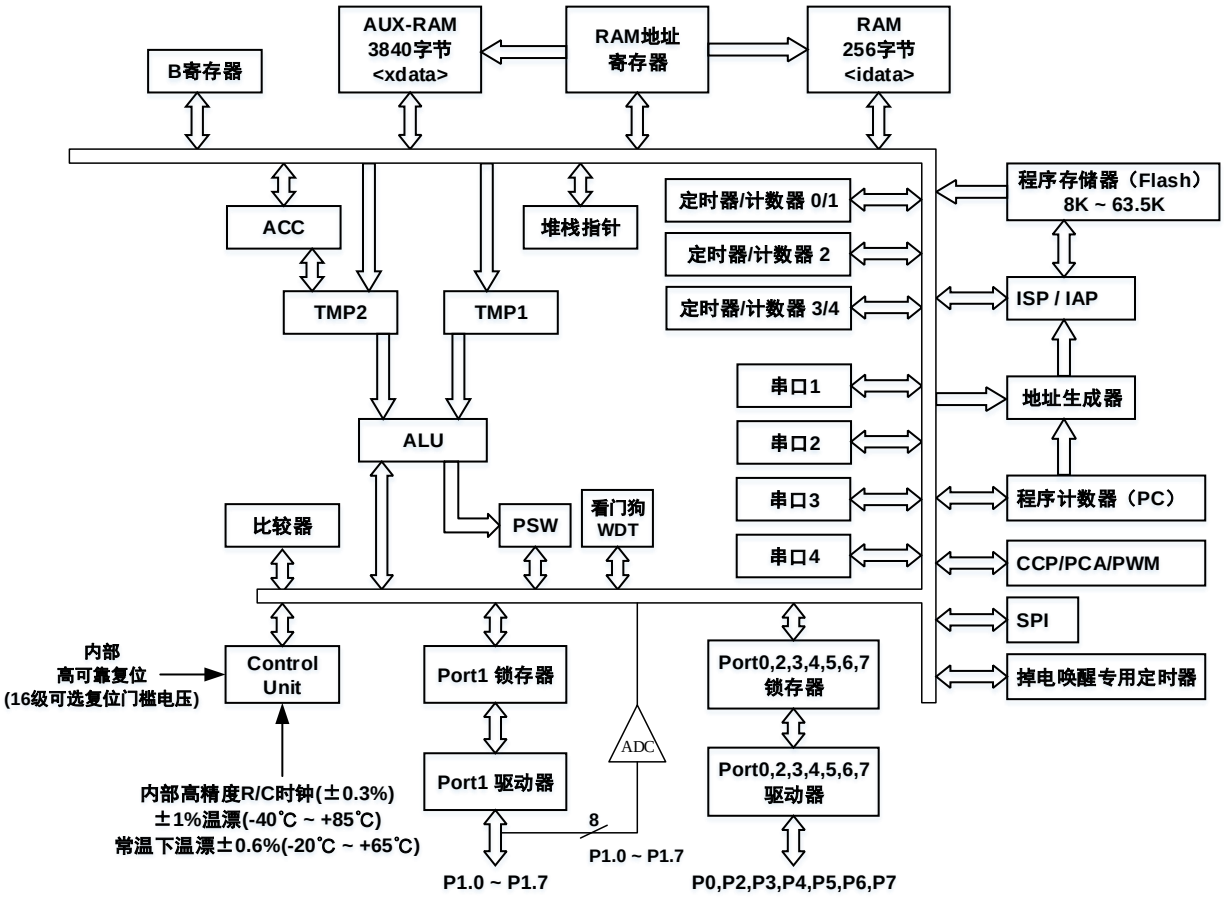
1.1.3.1 STC15F2K60S2 内部结构图



STC15F2K60S2系列内部结构框图

STC15F2K60S2 系列单片机是 STC 生产的单时钟/机器周期（1T）的单片机，是高速/高可靠/低功耗/超强抗干扰的新一代 8051 单片机，采用 STC 第八代加密技术，无法解密，指令代码完全兼容传统 8051，但速度快 8-12 倍。内部集成高精度 R/C 时钟（±0.3%），±1%温漂（-40℃ ~ +85℃），常温下温漂+0.6%（-20℃～+65℃），ISP 编程时 5MHz～35MHz 宽范围可设置，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 8 级复位门槛电压可选）。3 路 CCP/PWM/PCA，8 路高速 10 位 A/D 转换（30 万次/秒），内置 2K 字节大容量 SRAM，2 组超高速步串行通信端口（UART1/UART2，可在 5 组管脚之间进行切换，分时复用可作 5 组串口使用），1 组高速同步串行通信端口 SPI，针对多串行口通信/电机控制干扰场合。

1.1.3.2 STC15W4K32S4 内部结构图



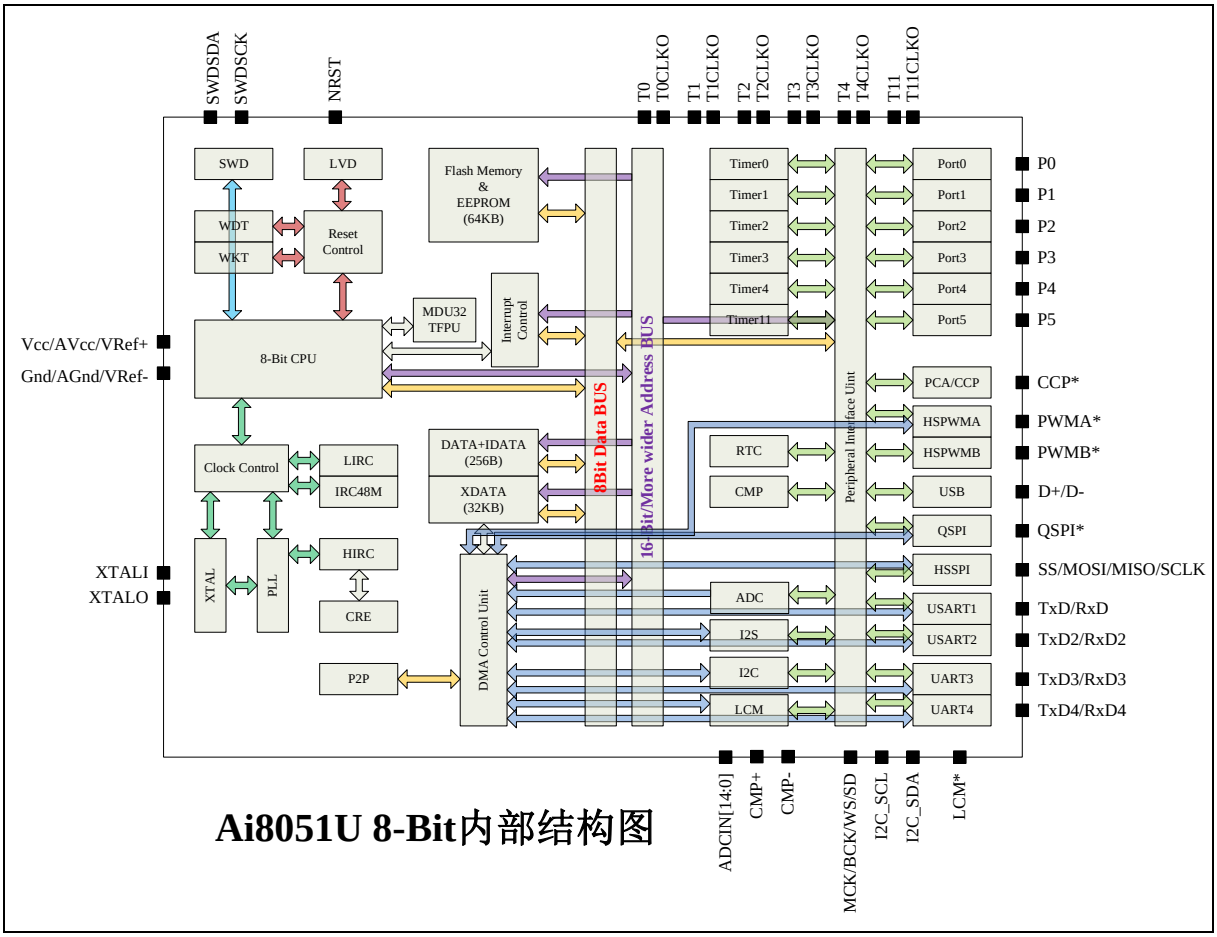
STC15W4K32S4系列内部结构框图

型号	工作频率(MHz)	工作温度(I --工业级)	所有封装价格(RMB ¥) LQFP64S/LQFP64L/QFN64/LQFP48/QFN48/LQFP44/PDIP40/LQFP32/SOP28/SKDIP28									
			SOP28 (26 个 I/O 口)	SKDIP28 (26 个 I/O 口)	LQFP32 (30 个 I/O 口)	PDIP40 (38 个 I/O 口)	LQFP44 (42 个 I/O 口)	LQFP48 (46 个 I/O 口)	QFN48 (46 个 I/O 口)	LQFP64S (62 个 I/O 口)	LQFP64L (62 个 I/O 口)	QFN64 (62 个 I/O 口)
STC15W4K16S4	28	-40℃ ~ +85℃	-	-	-	-	-	-	-	-	-	-
STC15W4K32S4	28	-40℃ ~ +85℃	-	-	-	-	-	-	-	-	-	-
STC15W4K40S4	28	-40℃ ~ +85℃	-	-	-	-	-	-	-	-	-	-
STC15W4K48S4	28	-40℃ ~ +85℃	-	-	-	-	-	-	-	-	-	-
STC15W4K56S4	28	-40℃ ~ +85℃	-	-	-	-	-	-	-	-	-	-
IAP15W4K58S4	28	-40℃ ~ +85℃	-	-	-	-	-	-	-	-	-	-
IAP15W4K61S4	28	-40℃ ~ +85℃	-	-	-	-	-	-	-	-	-	-
IRC15W4K63S4	28	-40℃ ~ +85℃	-	-	-	-	-	-	-	-	-	-

1.1.4 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）。

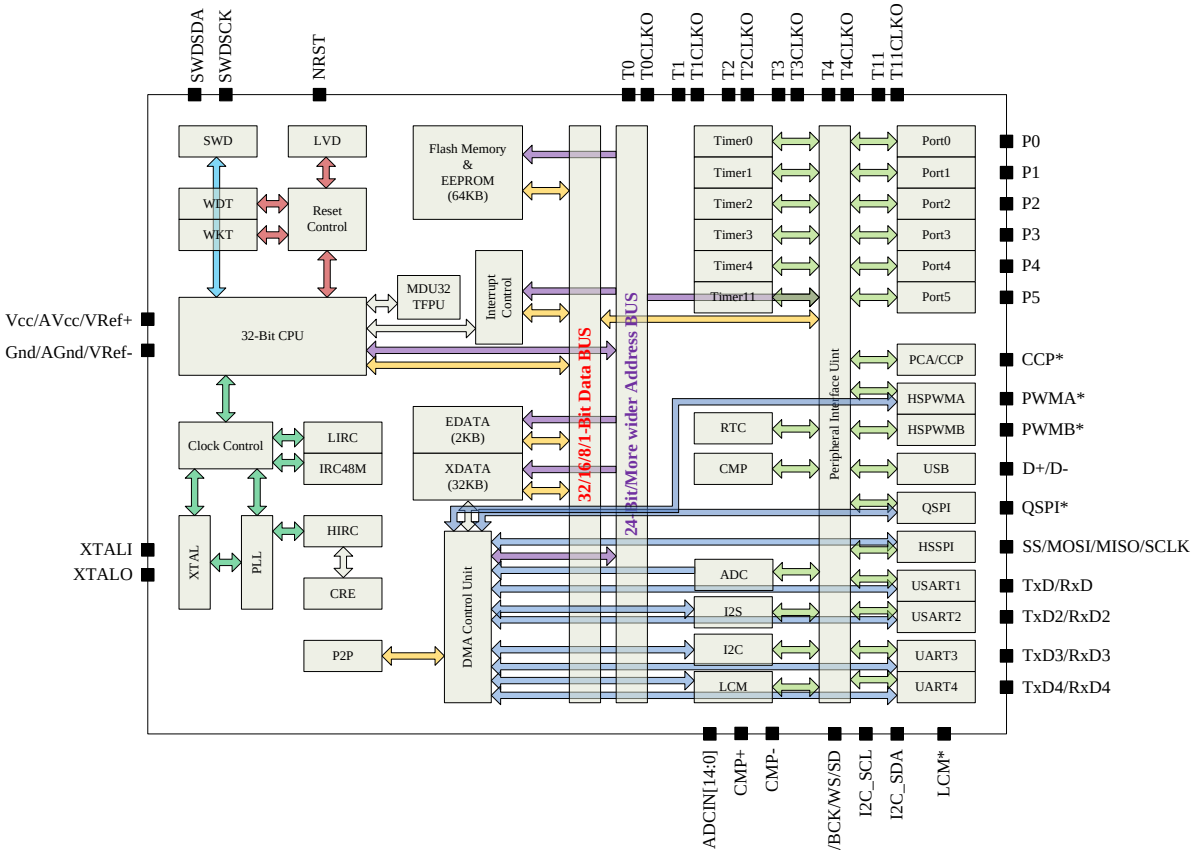
1.1.4.1 Ai8051U 8-Bit 内部结构图，内部总线工作在 8 位宽度



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 I²C 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	支持上升沿中断，下降沿中断以及边沿中断	传统的 PCA/CCP/PWM（可当外部中断并可掉电唤醒）	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	是	¥2.3	¥3.9	¥4.0	

1.1.4.2 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。

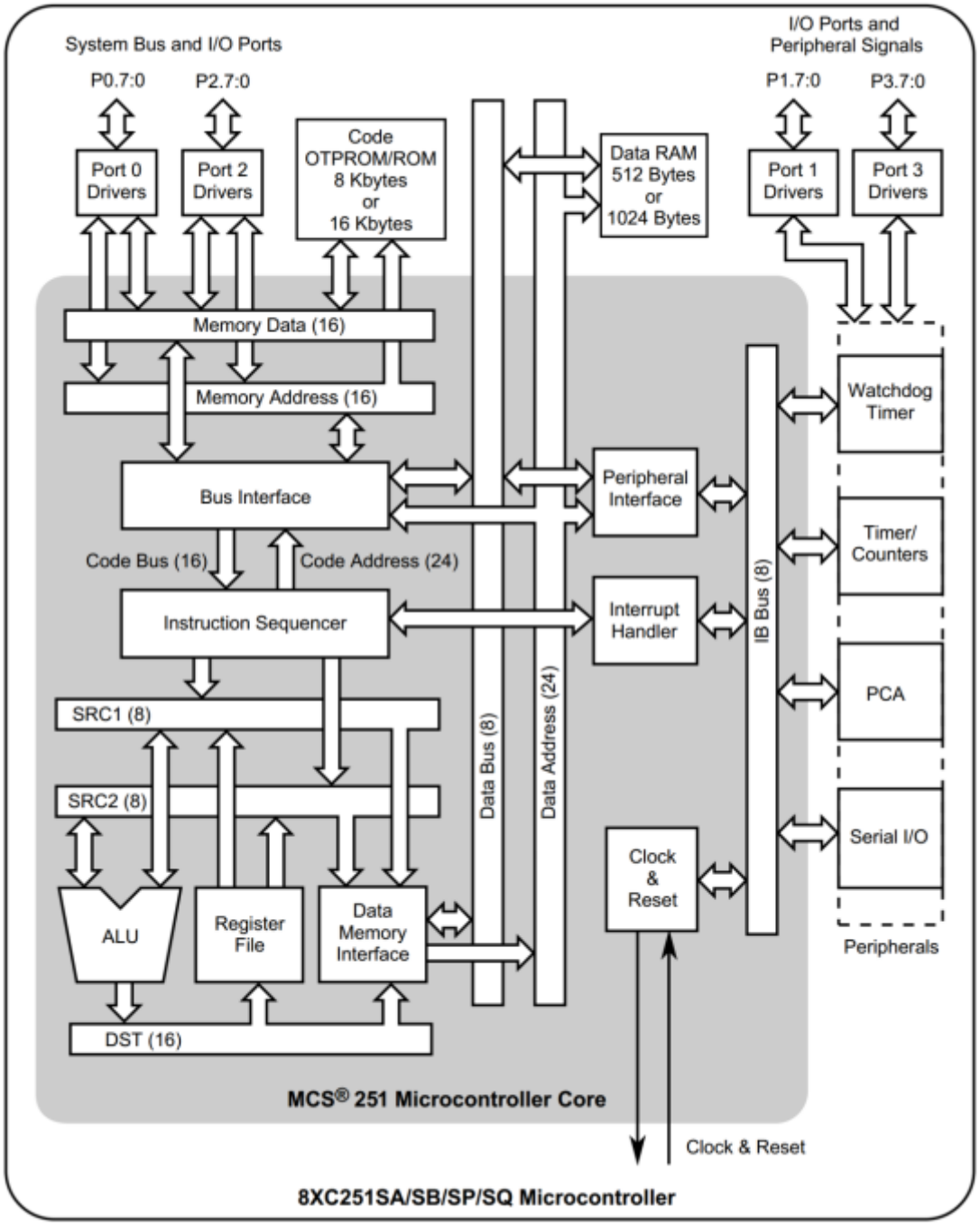
MCU 提供了丰富的数字外设（4 个串口、6 个定时器、2 组针对三相电机控制能够输出互补/对称/带死区控制信号的 16 位高级 PWM 定时器以及 I2C、SPI、QSPI、USB）接口与模拟外设（超高速 12 位 ADC、比较器），可满足广大用户的设计需求。

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

1.1.5 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

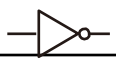
真值表		
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

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)′

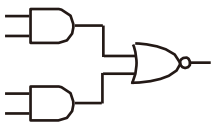
真值表		
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

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 下载软件及仿真的介绍

2.1 下载 Keil 集成开发环境

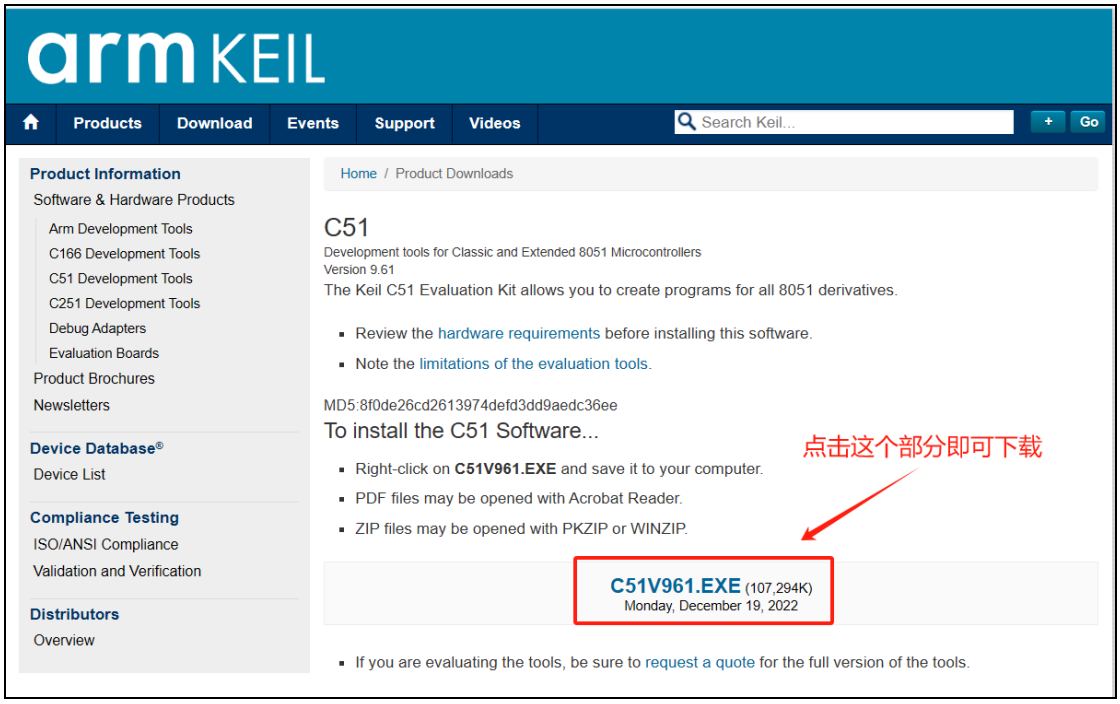
登录 Keil 官网，下载最新版的 C251 和 C51 编译器，下载链接如下：
https://www.keil.com/download/product/
进入下载页面，我们可以看到 C51 和 C251 的下载链接，这里将这两个安装包都下载到桌面上



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。



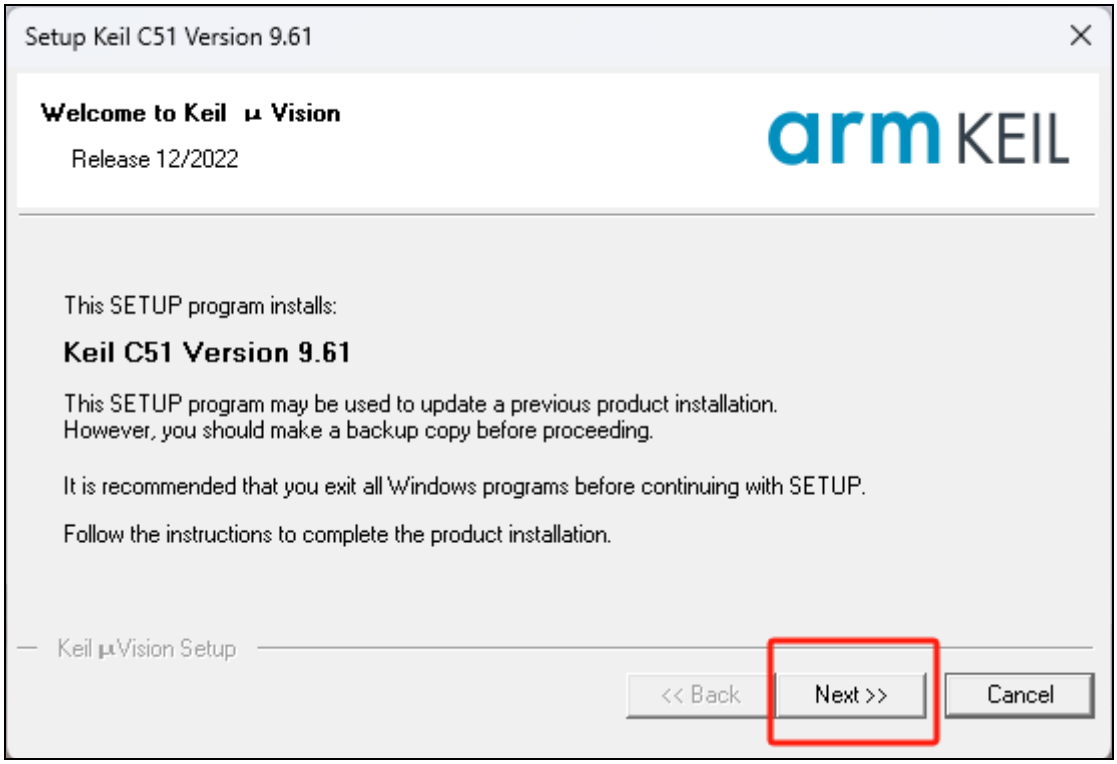
出现下载 C51V961.EXE 的界面/或更高版本的界面，点击粗体蓝色字即可下载 C51 安装包。同样步骤下载 C251 安装包。



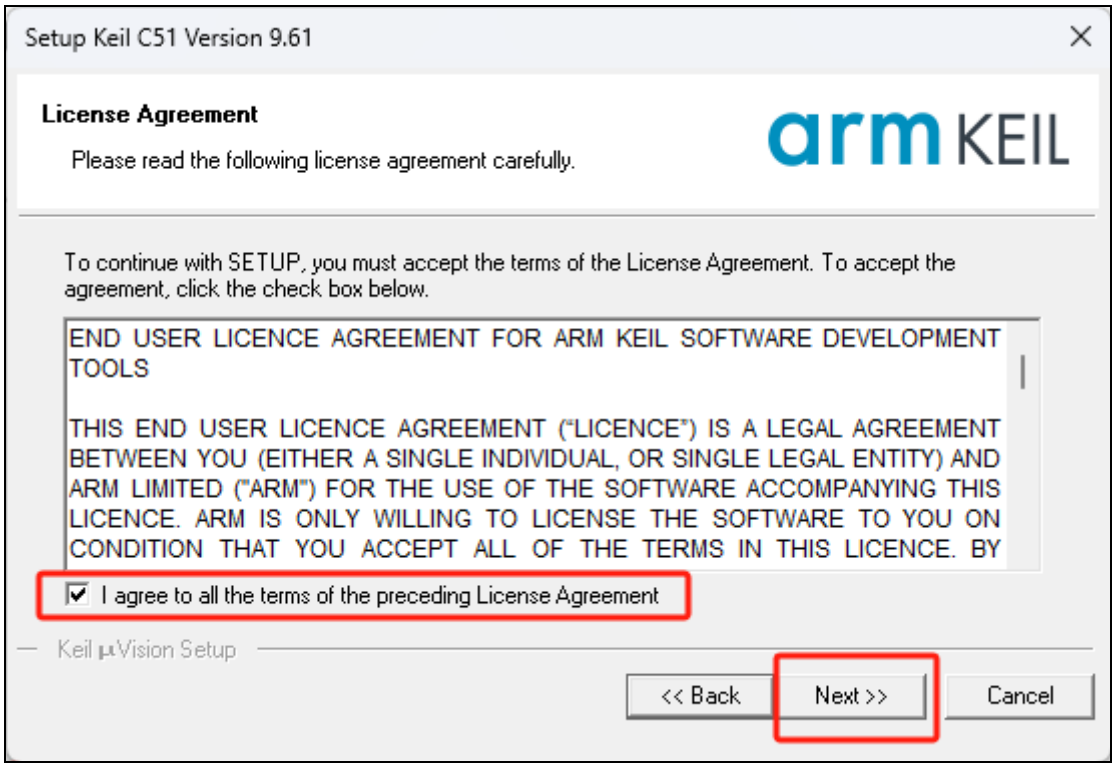
2.2 安装 Keil 集成开发环境

2.2.1 安装 Keil C51 集成开发环境

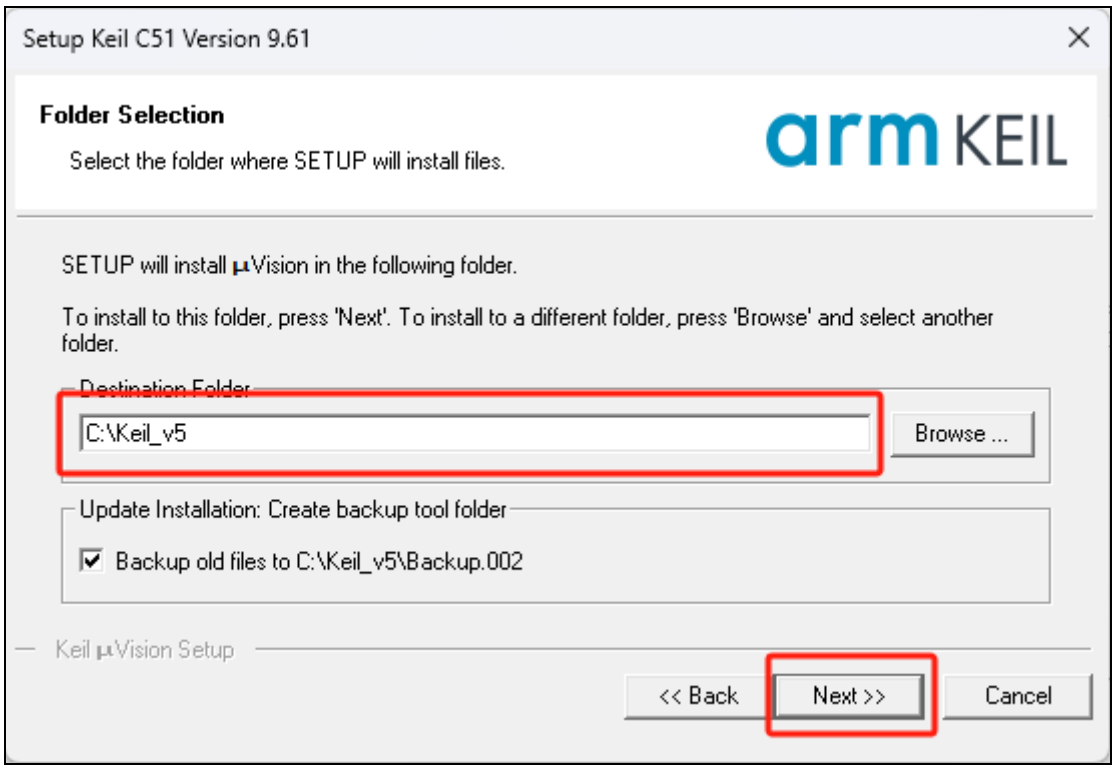
双击下载的安装包开始安装，点击“Next”：



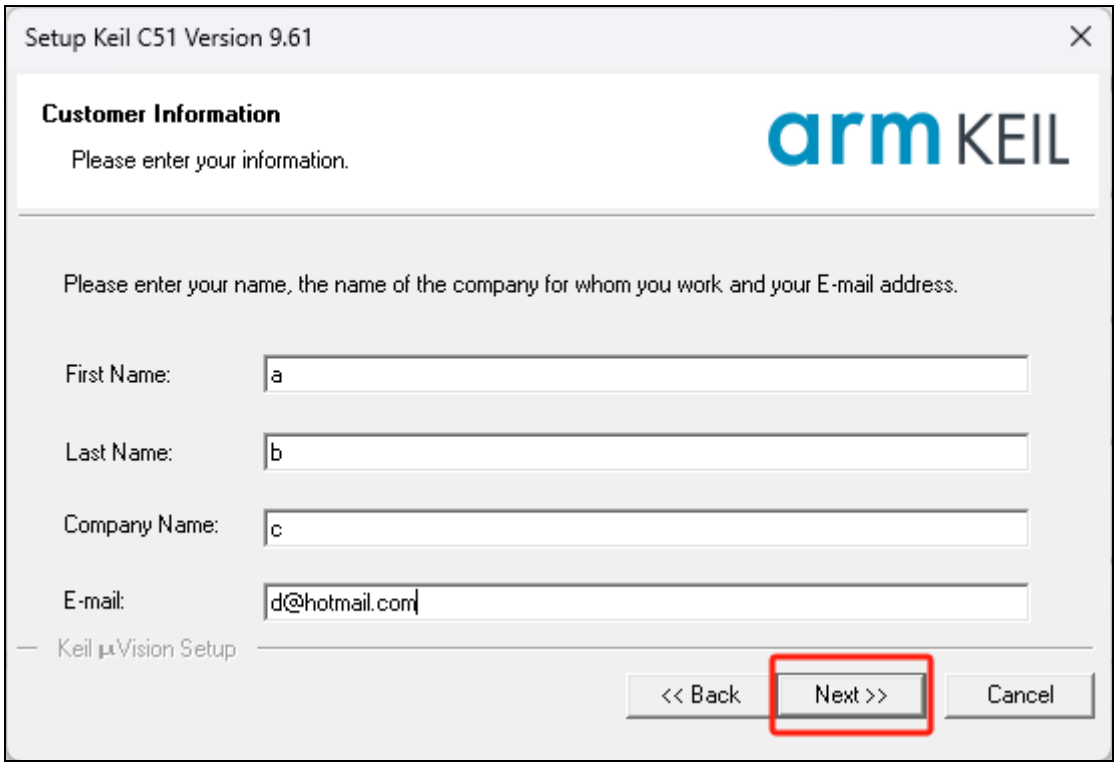
勾选“I agree to all the terms of the preceding License Agreement”,然后点击“Next”：



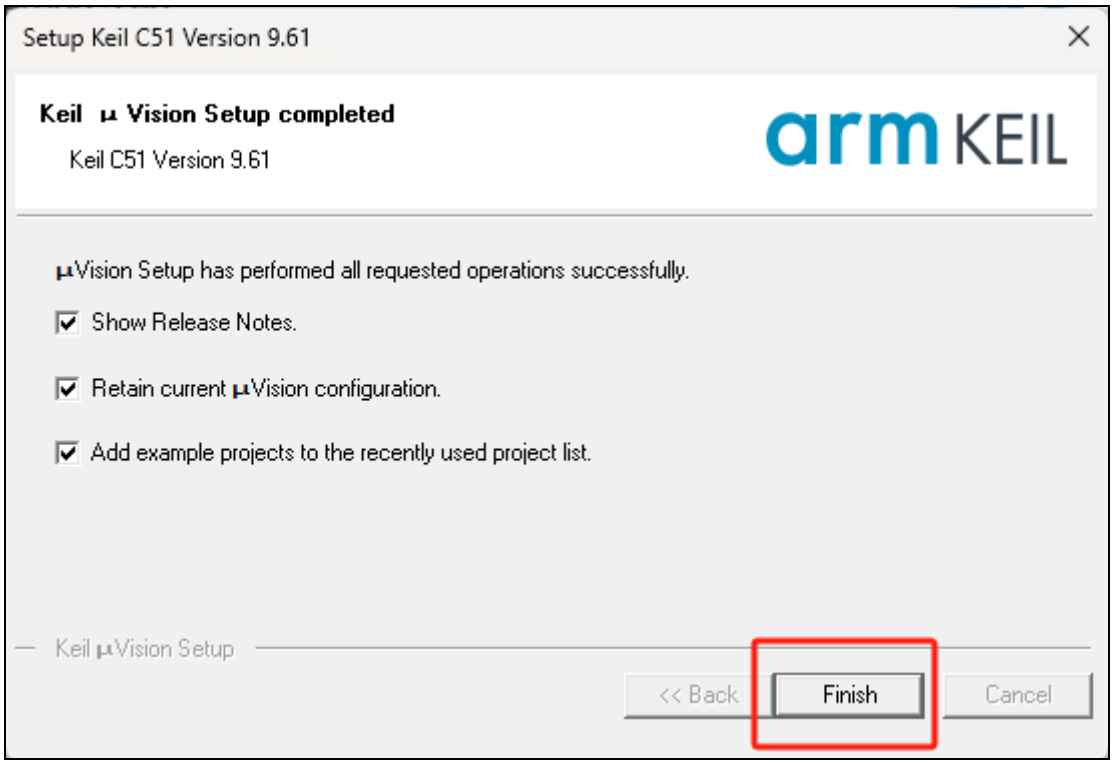
选择安装目录，然后点击“Next”：



填写个人信息，然后点击“Next”：

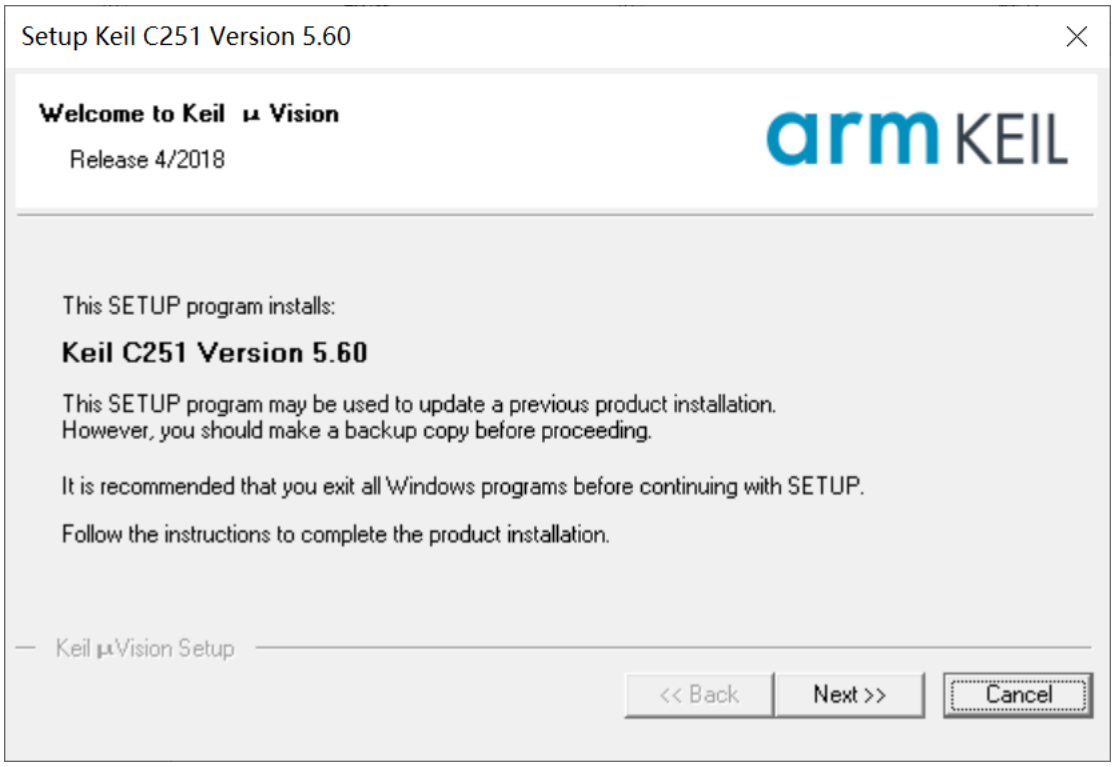


安装完成，点击“Finish”结束。

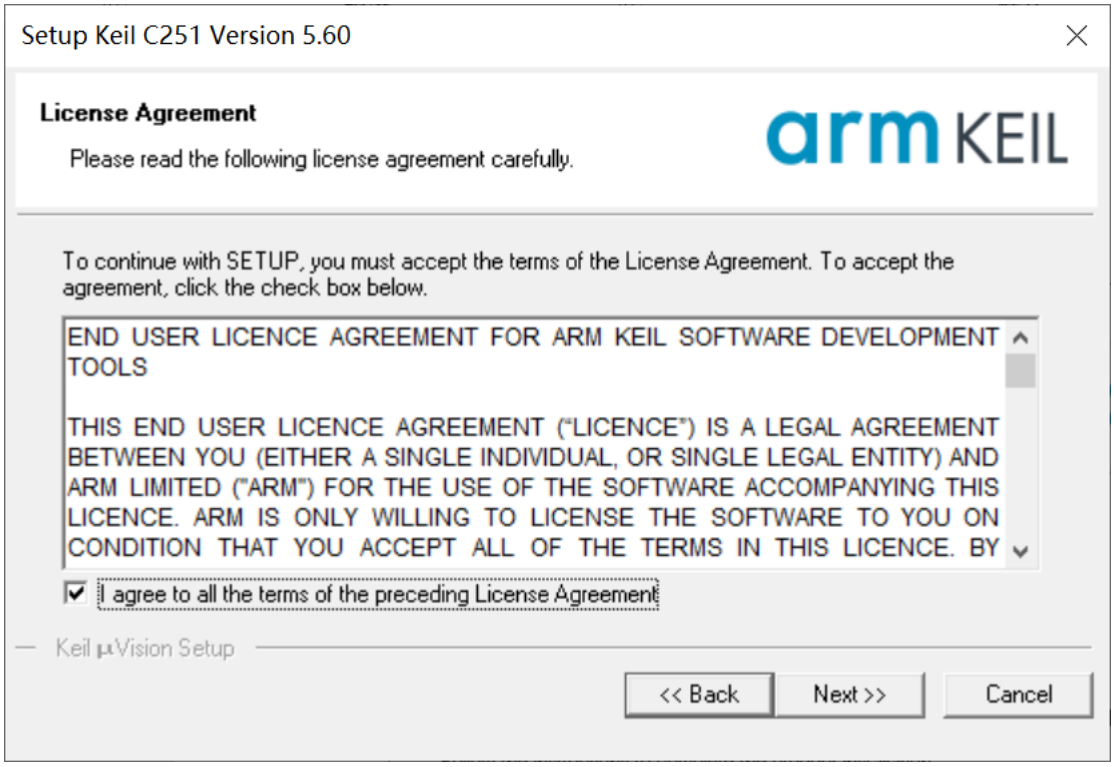


2.2.2 安装 Keil C251 集成开发环境

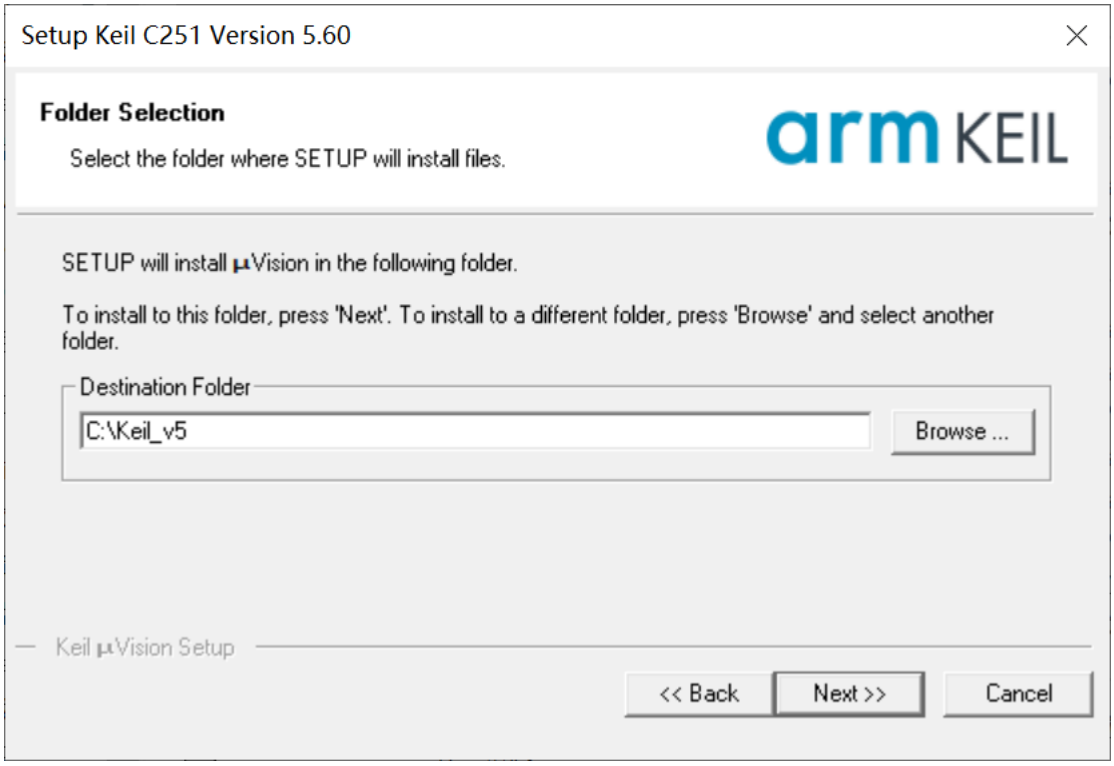
双击下载的安装包开始安装，点击“Next”：



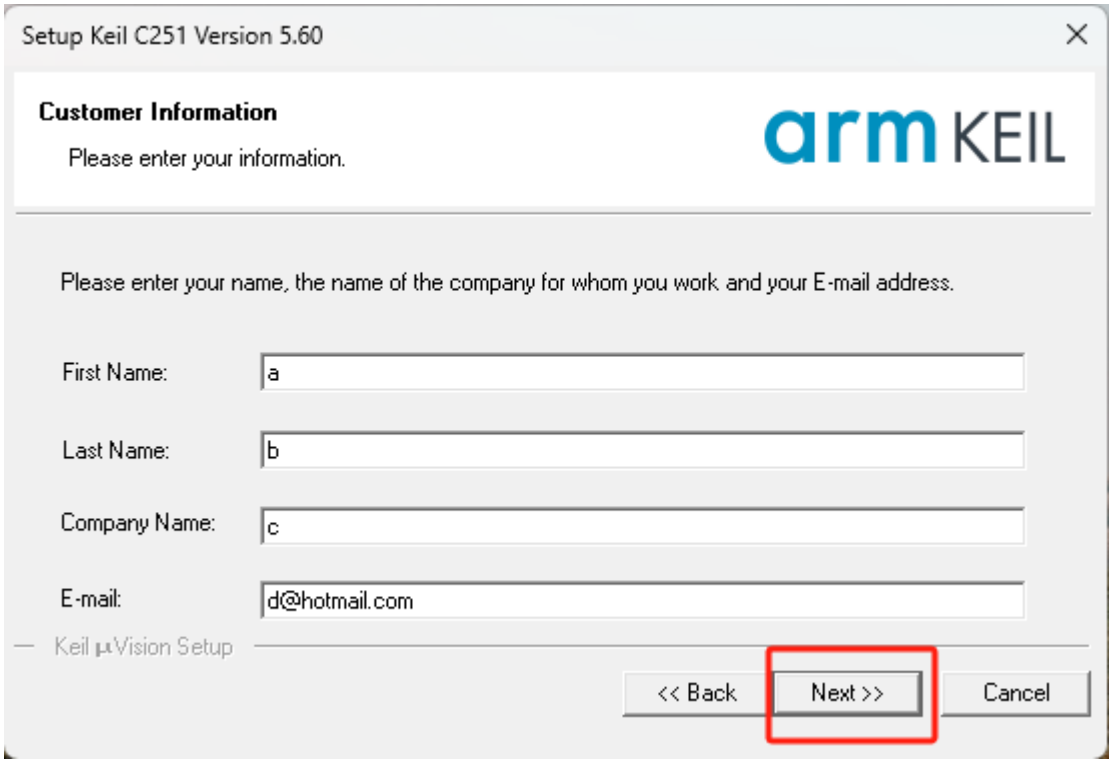
勾选“I agree to all the terms of the preceding License Agreement”,然后点击“Next”：



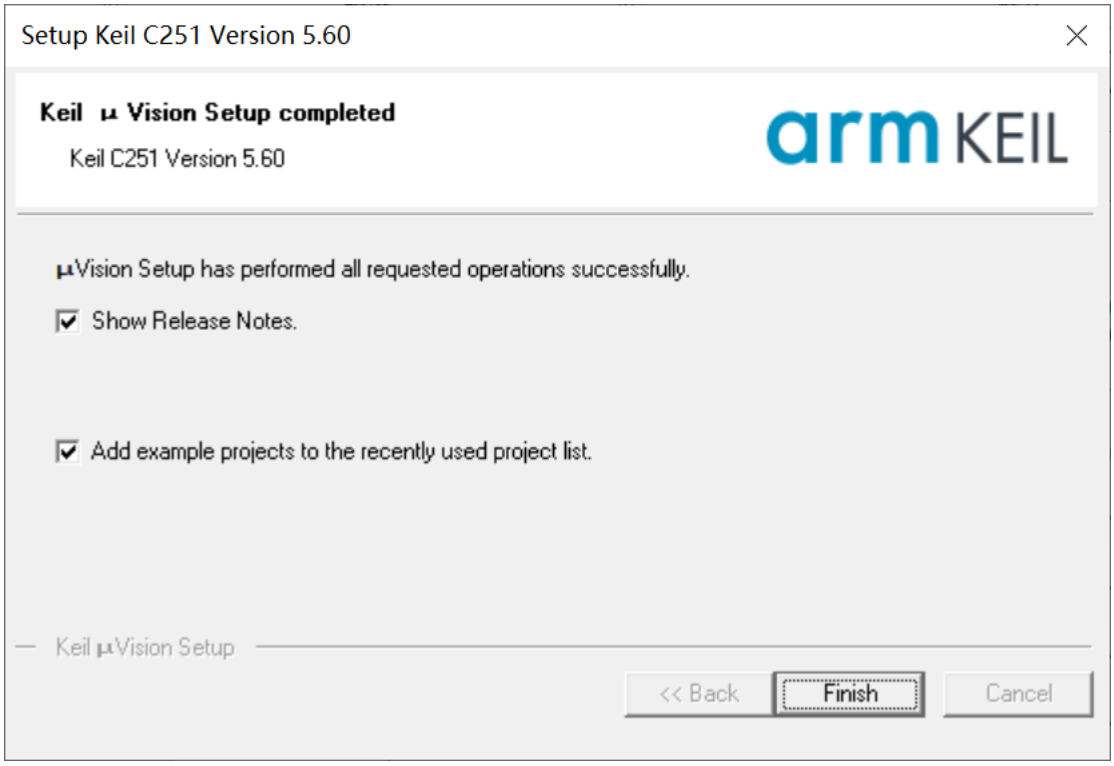
选择安装目录，然后点击“Next”：



填写个人信息，然后点击“Next”：

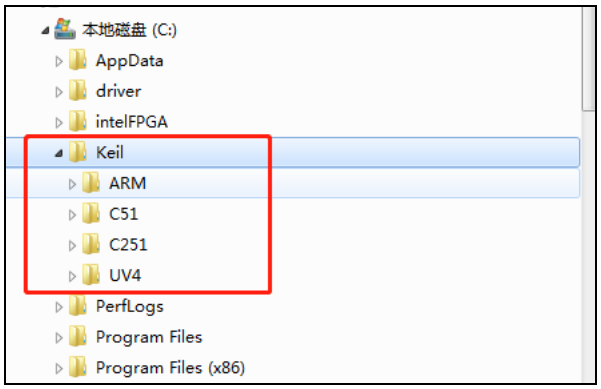


安装完成，点击“Finish”结束。

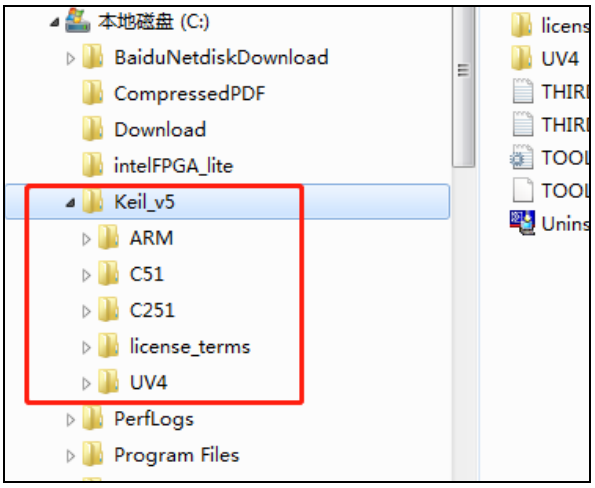


2.2.3 Keil 的 C51、C251 和 MDK 可安装在同一台电脑同一个目录

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/软件,含强大的辅助开发工具包

2.3.1 安装 AIapp-ISP 下载/编程/烧录/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：

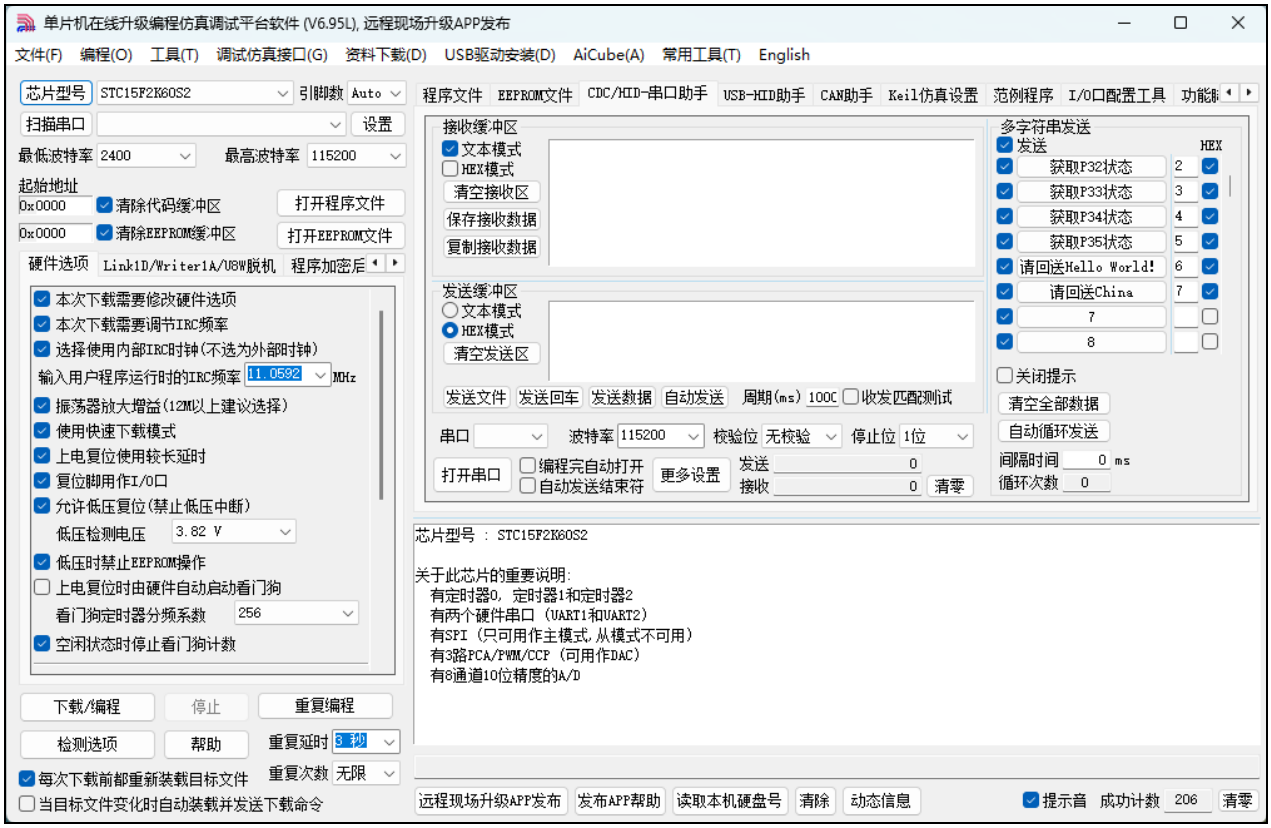


双击后解压缩如下图：



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。

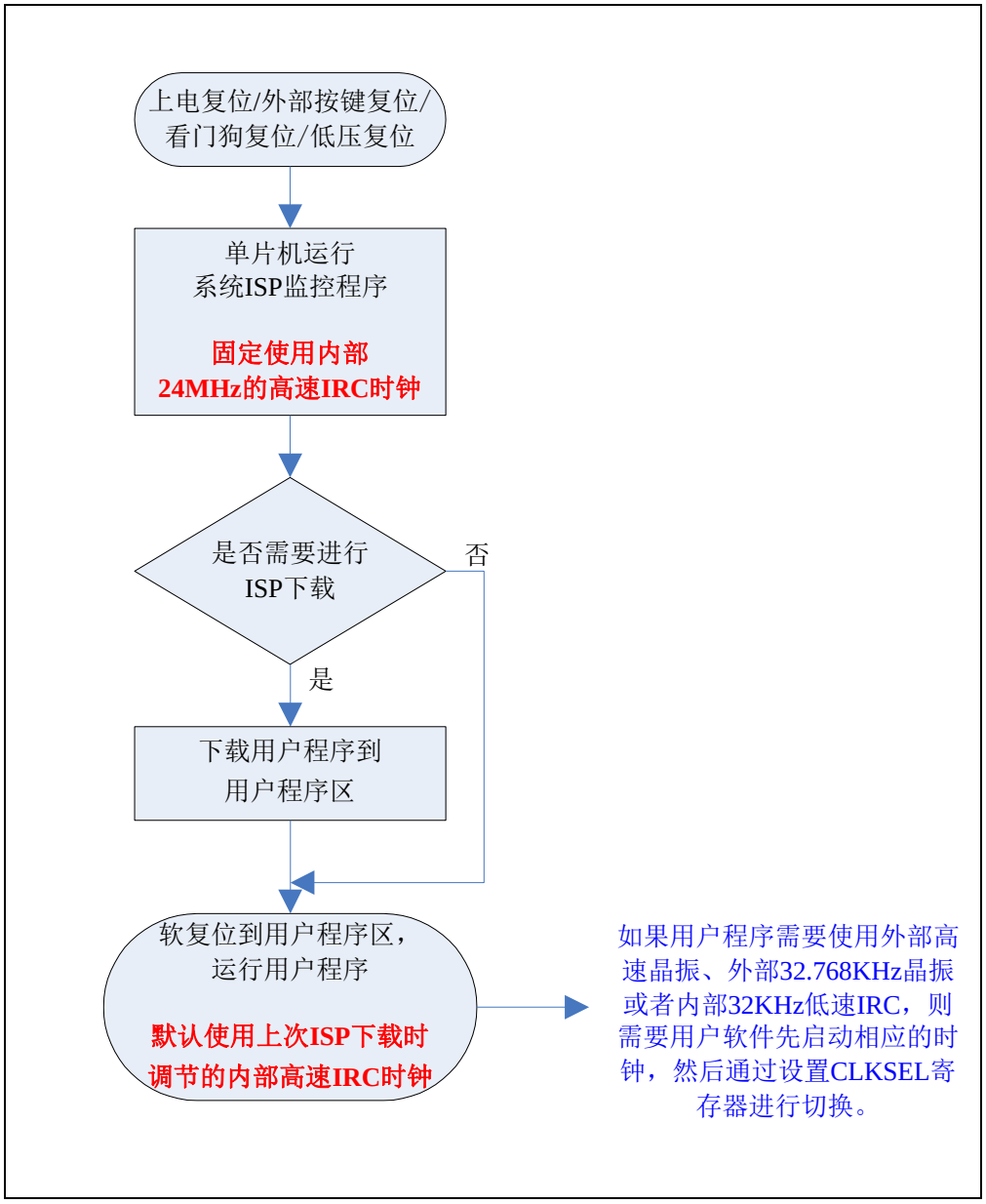
双击图标，即可打开 AIapp-ISP 软件界面，即可直接使用，如下图：


单片机在线升级编程仿真调试平台软件 (V6.95L), 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) AiCube(A) 常用工具(T) English
芯片型号 STC15F2K60S2 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
本次下载需要调节IRC频率
选择使用内部IRC时钟(不选为外部时钟)
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)
使用快速下载模式
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 3.82 V
低压时禁止EEPROM操作
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
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
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
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
获取P32状态 2
获取P33状态 3
获取P34状态 4
获取P35状态 5
请回送Hello World! 6
请回送China 7
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
芯片型号 : STC15F2K60S2
关于此芯片的重要说明:
有定时器0, 定时器1和定时器2
有两个硬件串口 (UART1和UART2)
有SPI (只可用作主模式, 从模式不可用)
有3路PCA/PWM/CCP (可用作DAC)
有8通道10位精度的A/D
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 206 清零

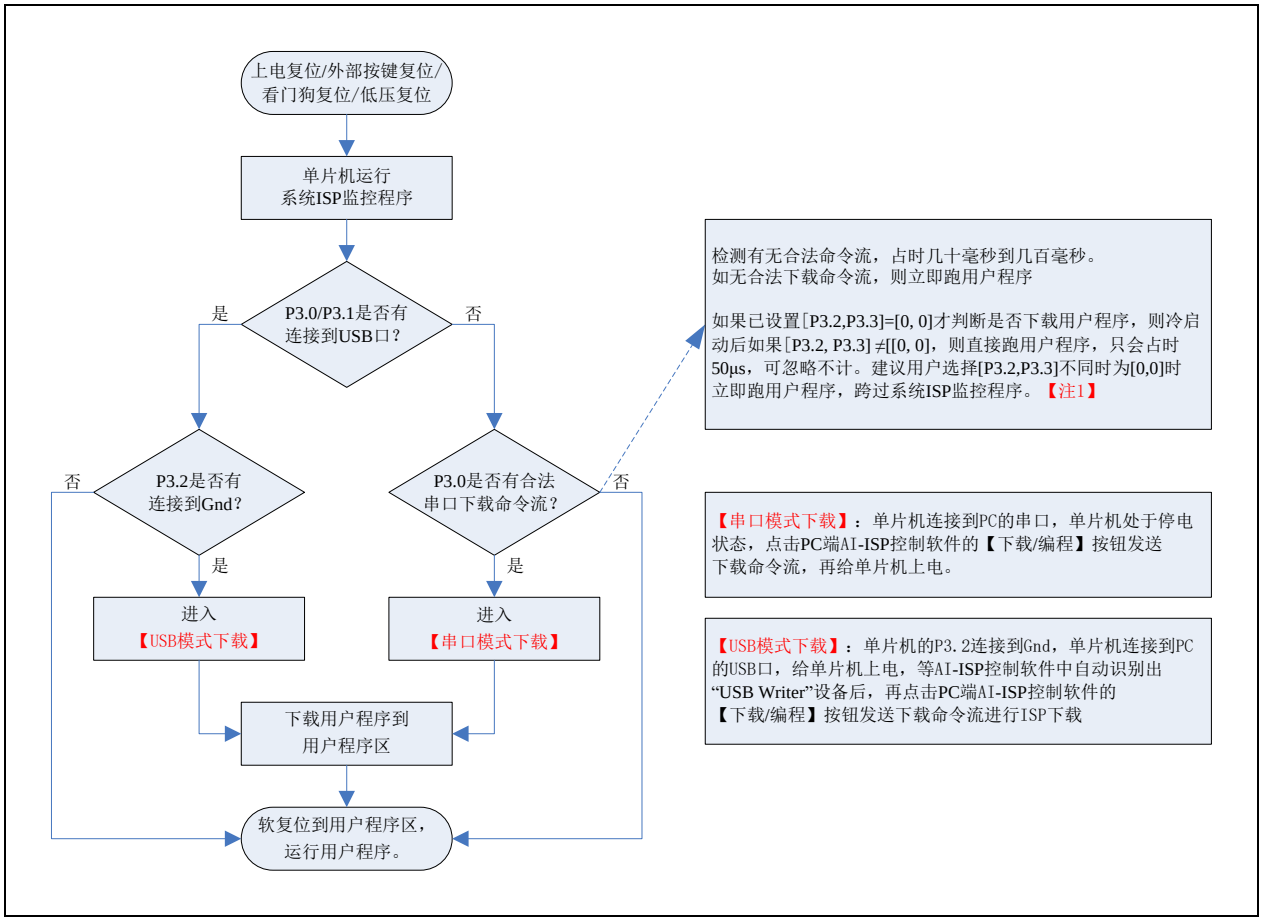
2.3.2 STC 单片机上电工作过程

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：



2.3.3 ISP 下载流程图（硬件 USB+串口模式）

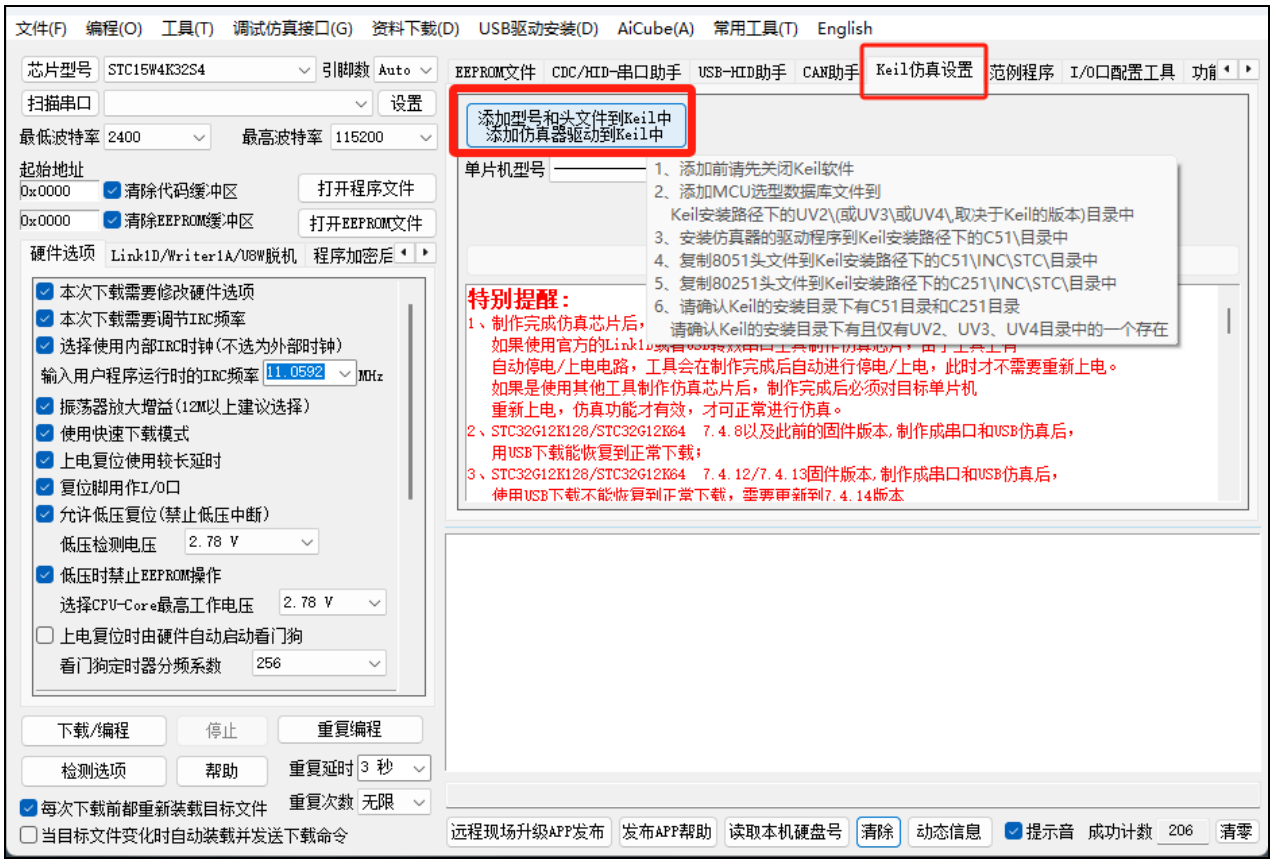


注意：因 [P3.0, P3.1] 作下载/仿真用(下载/仿真接口仅可用 [P3.0, P3.1])，故建议用户将串口 1 放在 P3.6/P3.7 或 P1.6/P1.7 ，若用户不想切换，坚持使用 P3.0/P3.1 工作或作为串口 1 进行通信，则务必在下载程序时，在软件上勾选“下次冷启动时，P3.2/P3.3 为 0/0 时才可以下载程序”。【注 1】

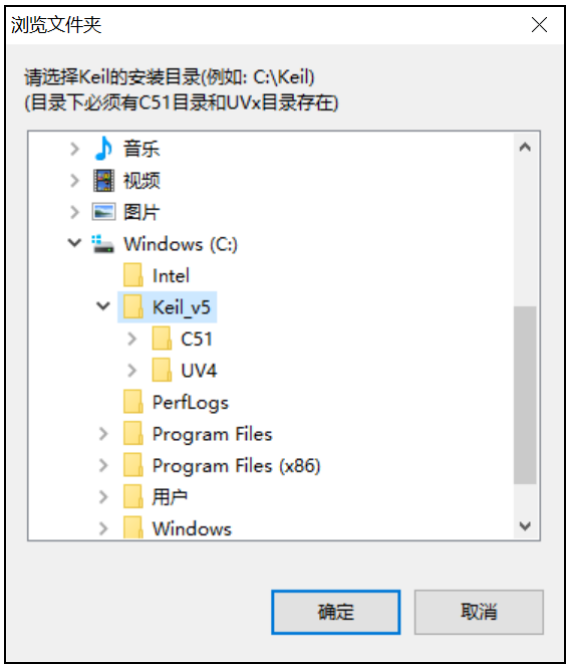
【注 1】：AI15，AI8 系列及以后新出的芯片的烧录保护引脚为 P3.2/P3.3，之前早期芯片的烧录保护引脚为 P1.0/P1.1。

2.4 添加型号和头文件到 Keil

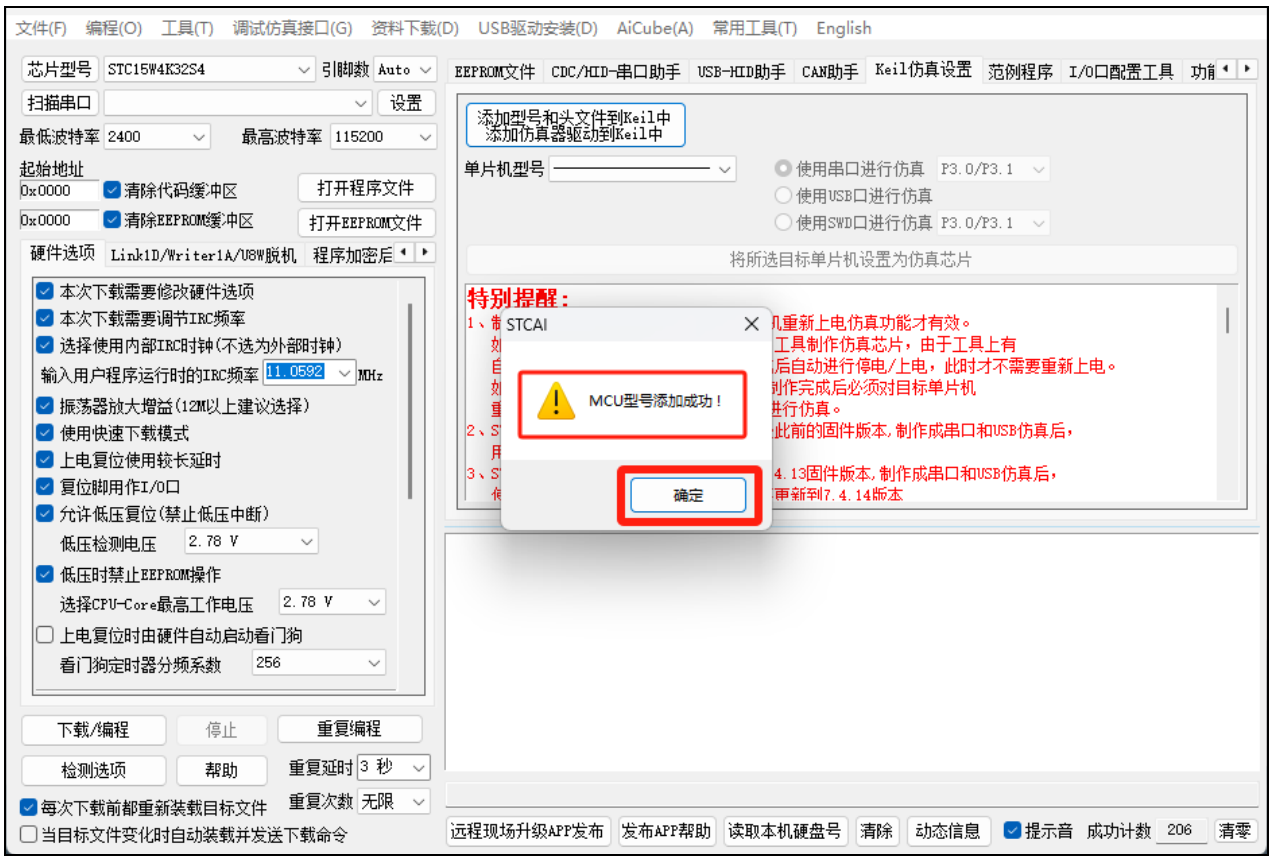
使用 Keil 之前需要先安装 仿真驱动。仿真驱动的安装步骤如下：
首先打开 ISP 下载软件，然后在软件右边功能区的“Keil 仿真设置”页面中点击“添加型号和头文件到 Keil 中 添加仿真器驱动到 Keil 中”按钮：



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：



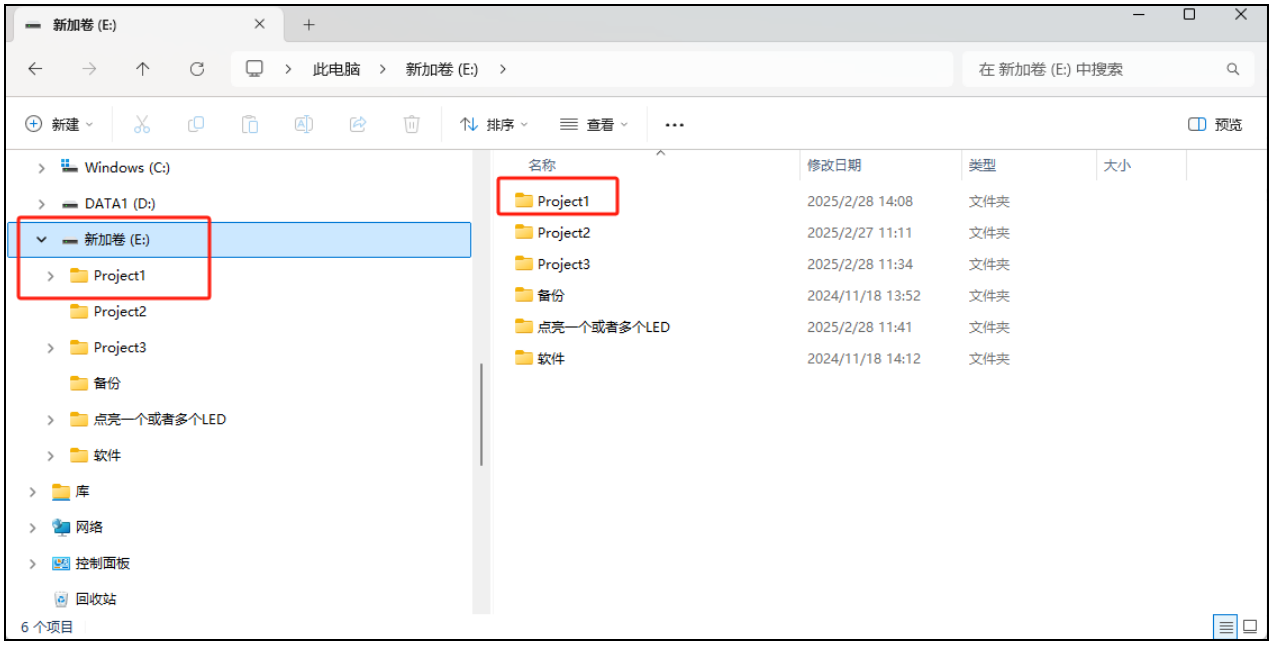
即表示驱动正确安装了

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

2.5 在 Keil 中新建一个 8051 项目

2.5.1 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

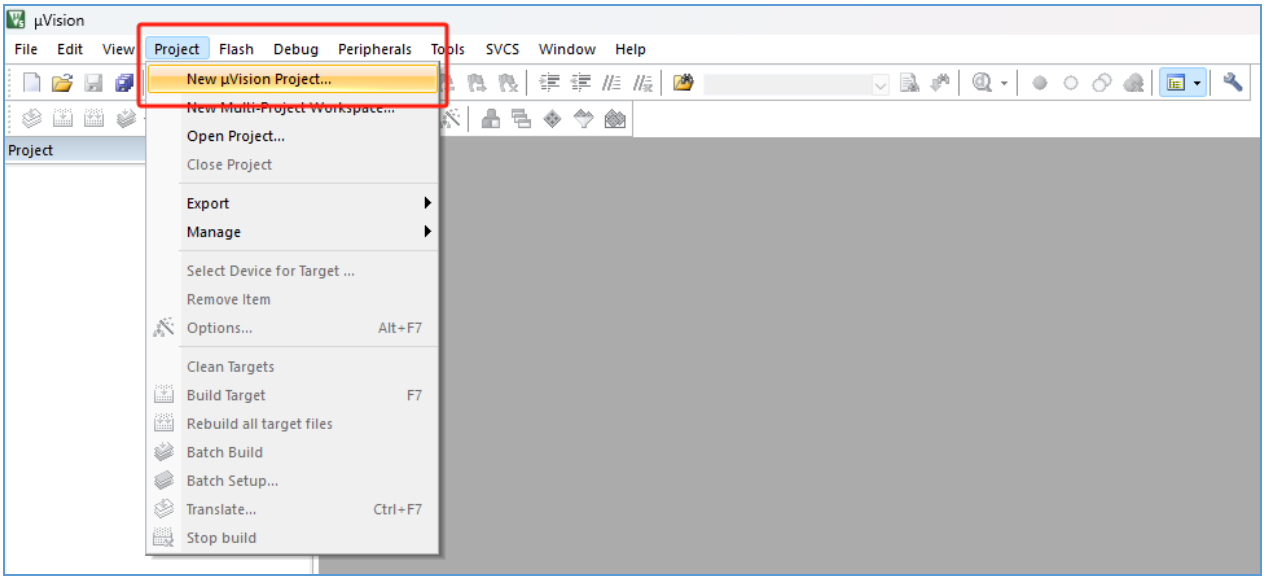


2.5.2 Keil 新建一个 8 位 8051 项目

2.5.2.1 新建工程

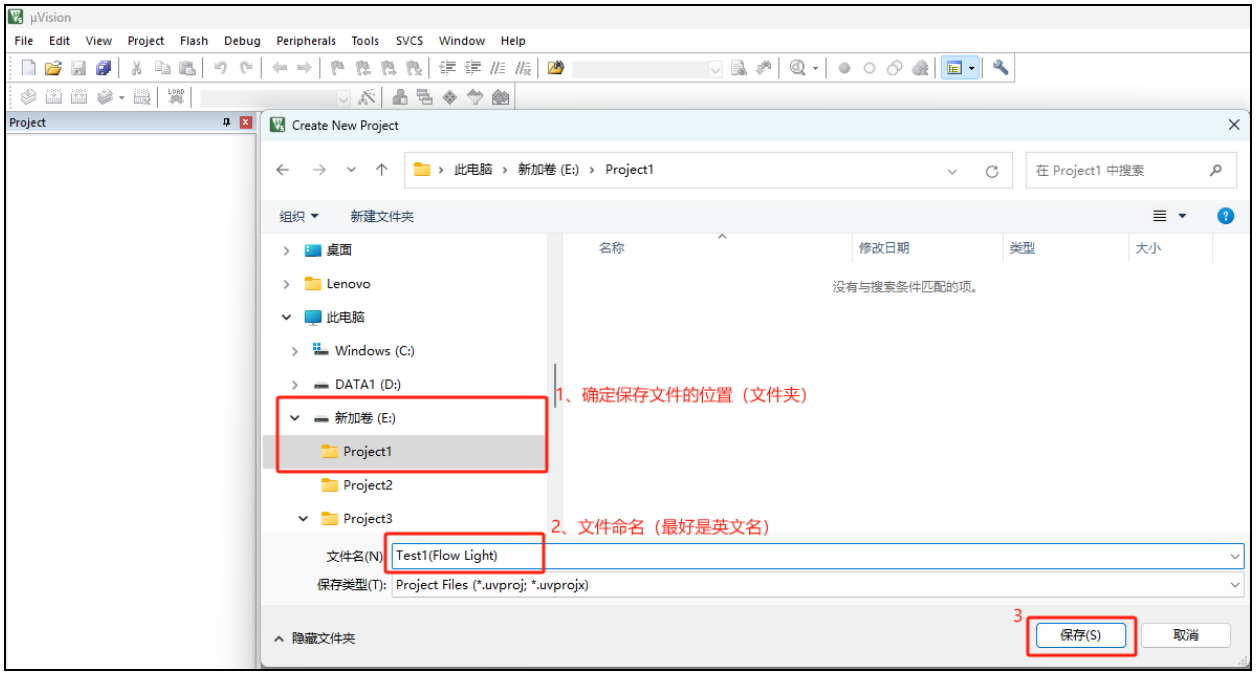
打开 Keil 软件

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



2）将文件存入刚刚建好的文件夹 E:\project1 中

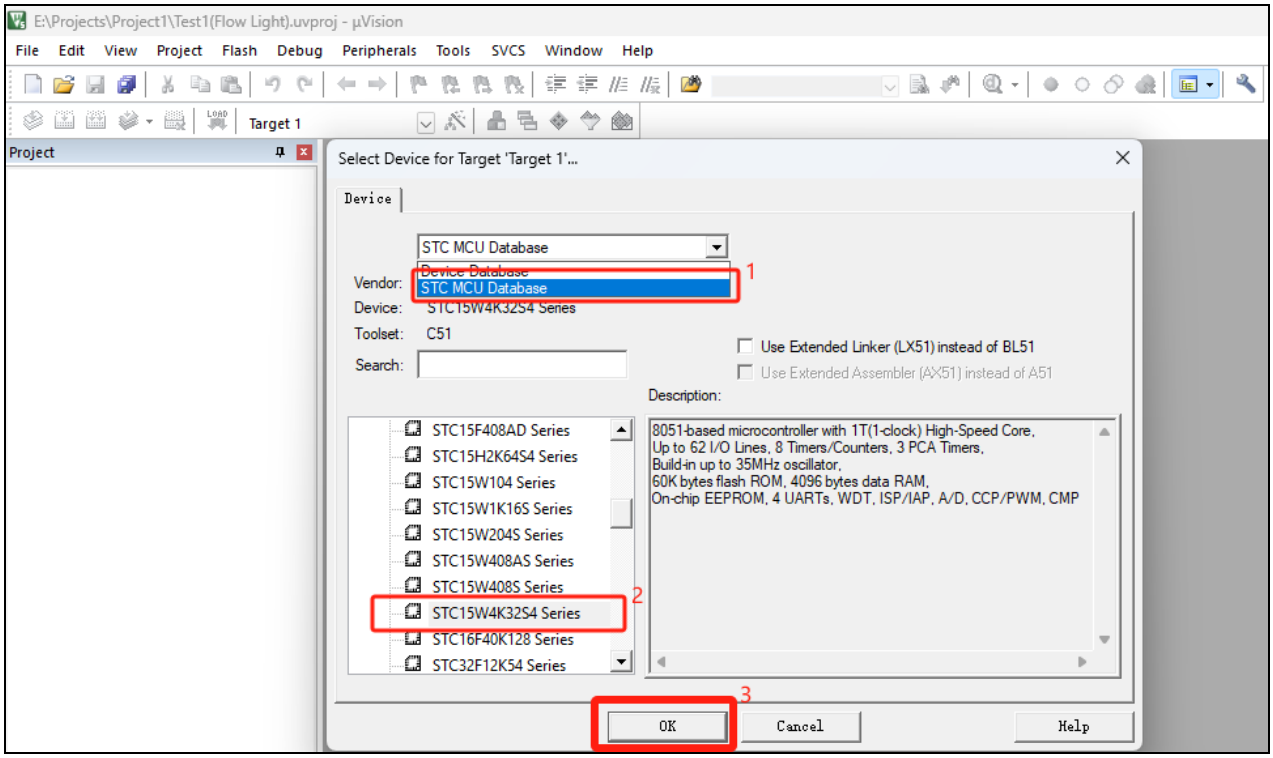
再给项目起个名称（方便查找）（尽量是英文的，如 Test…），点击“保存”。如下图：



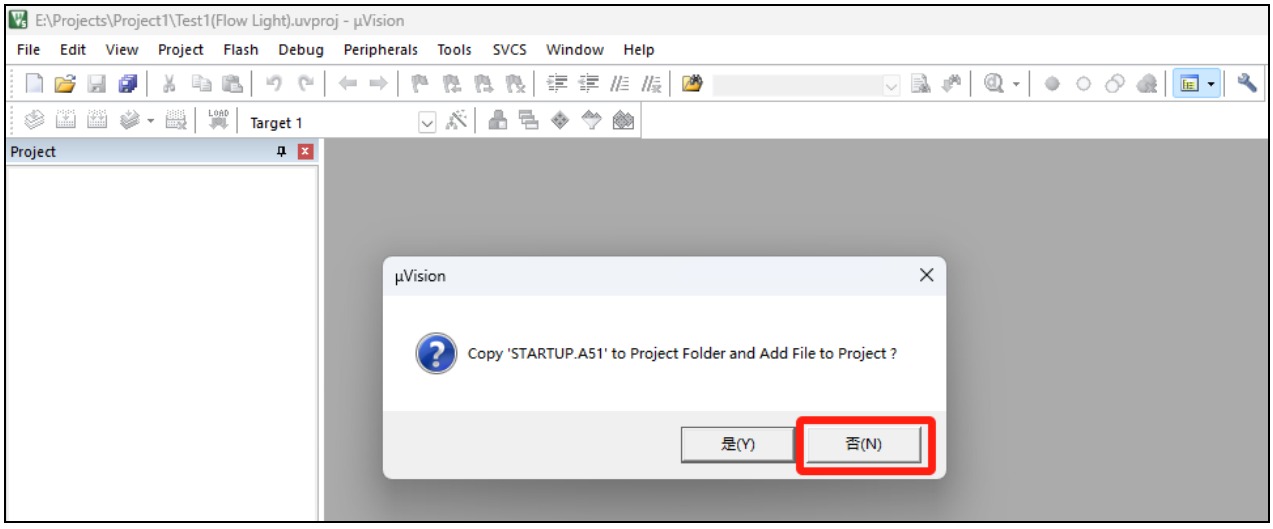
3）选择单片机（以 STC15W4K32S4 系列为例）

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

- 选择 STC MCU Database
- 然后展开 STC 选项，选择“STC15W4K32S4 系列”选项
- 最后点击‘OK’以完成单片机选择。

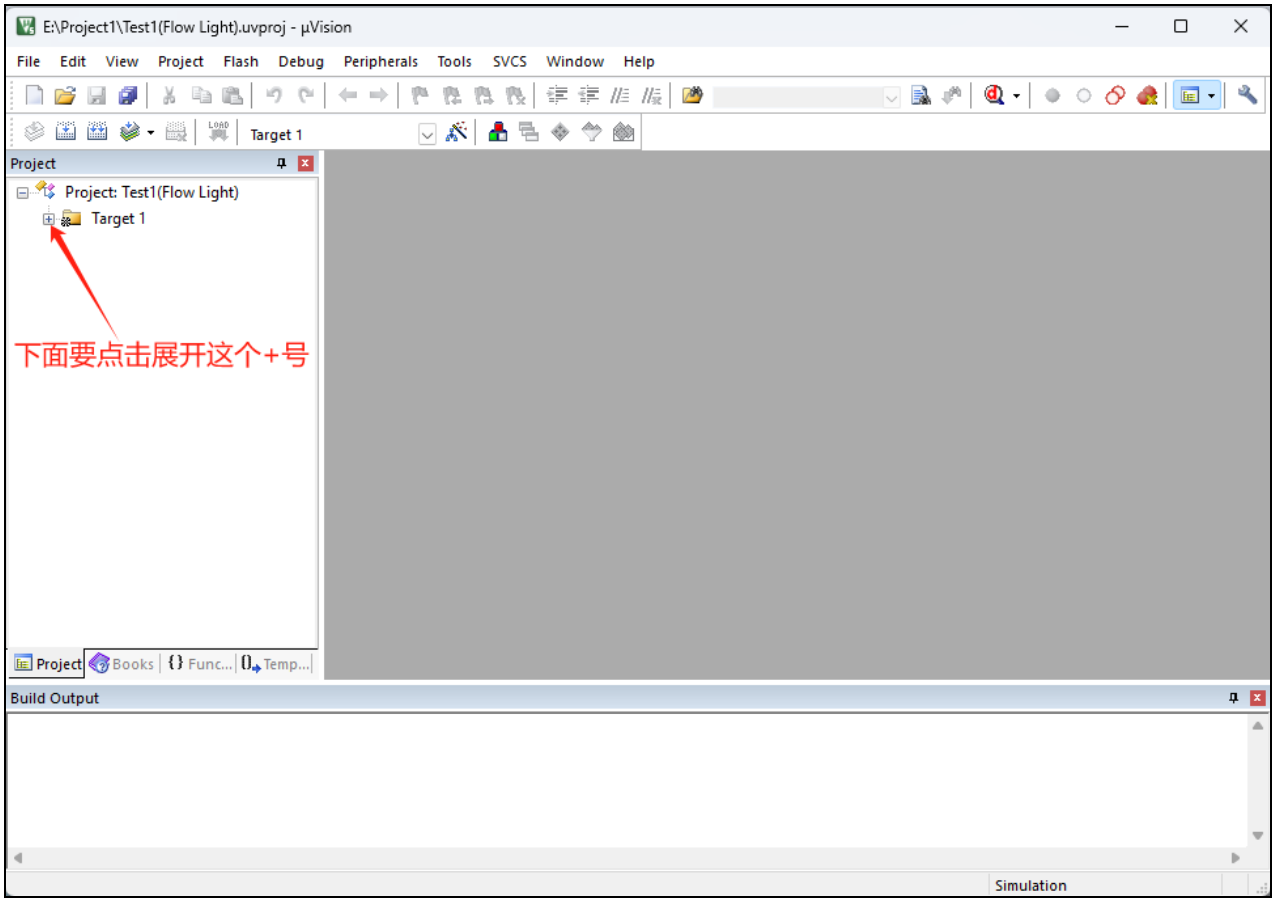


如下图，一般选择“否”（STARTUP.A51：系统与启动代码文件，清理 RAM、设置堆栈等）。

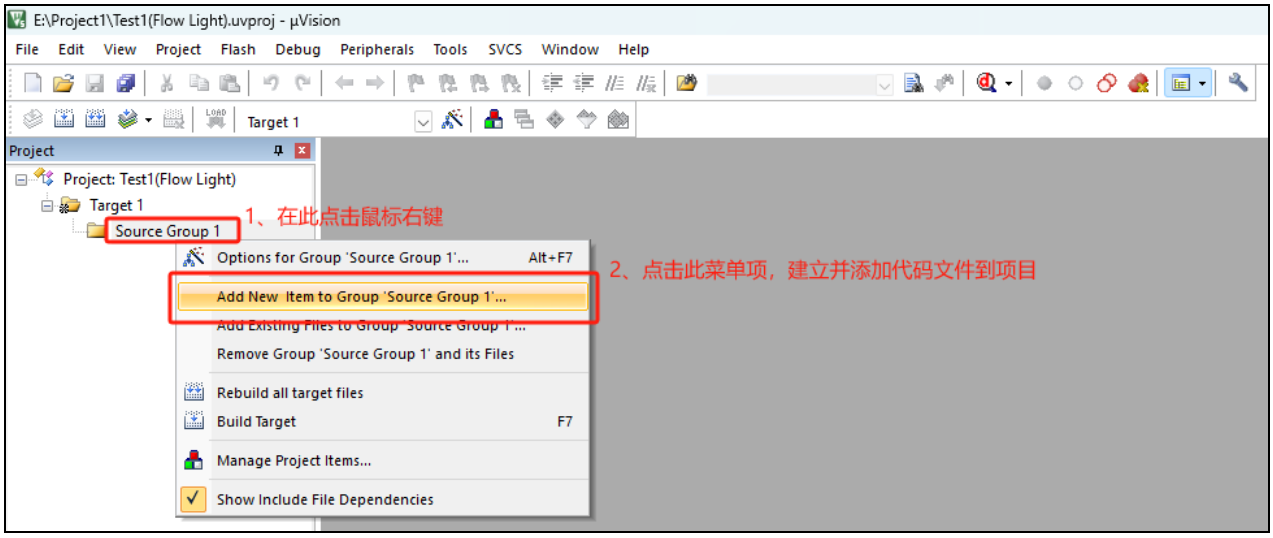


4）给工程项目增加一个主程序文件

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

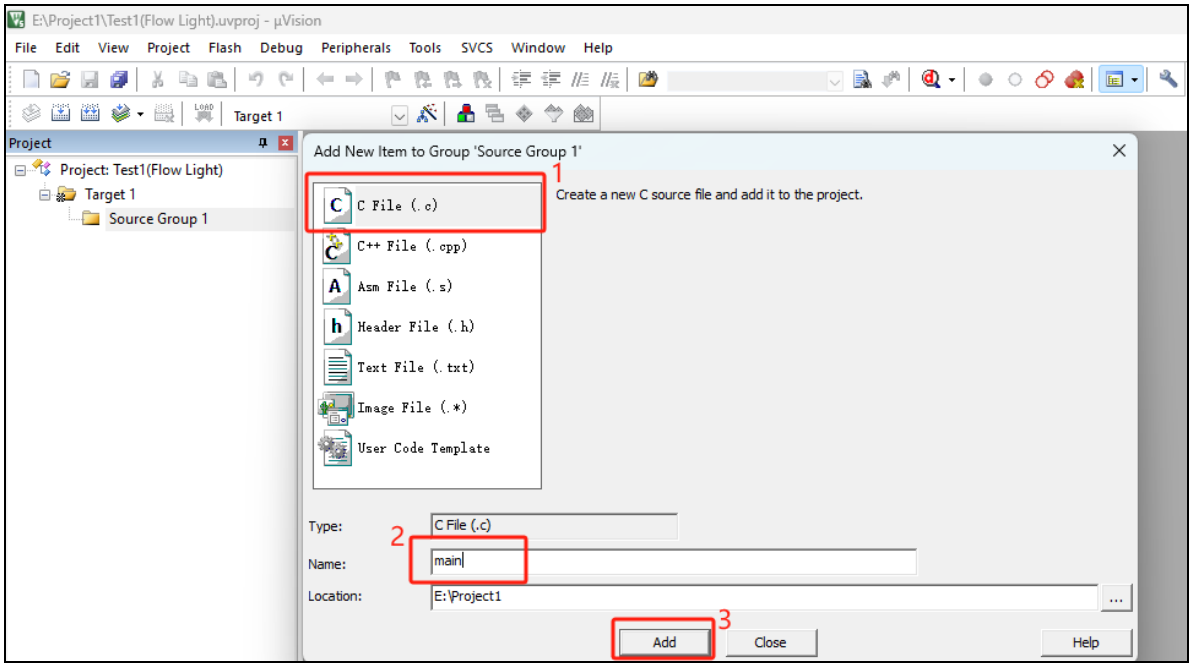


点击 Project 窗口上 Target1 的+，出现“Source Group1”，右键点击“Source Group1”，出现下图。



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：



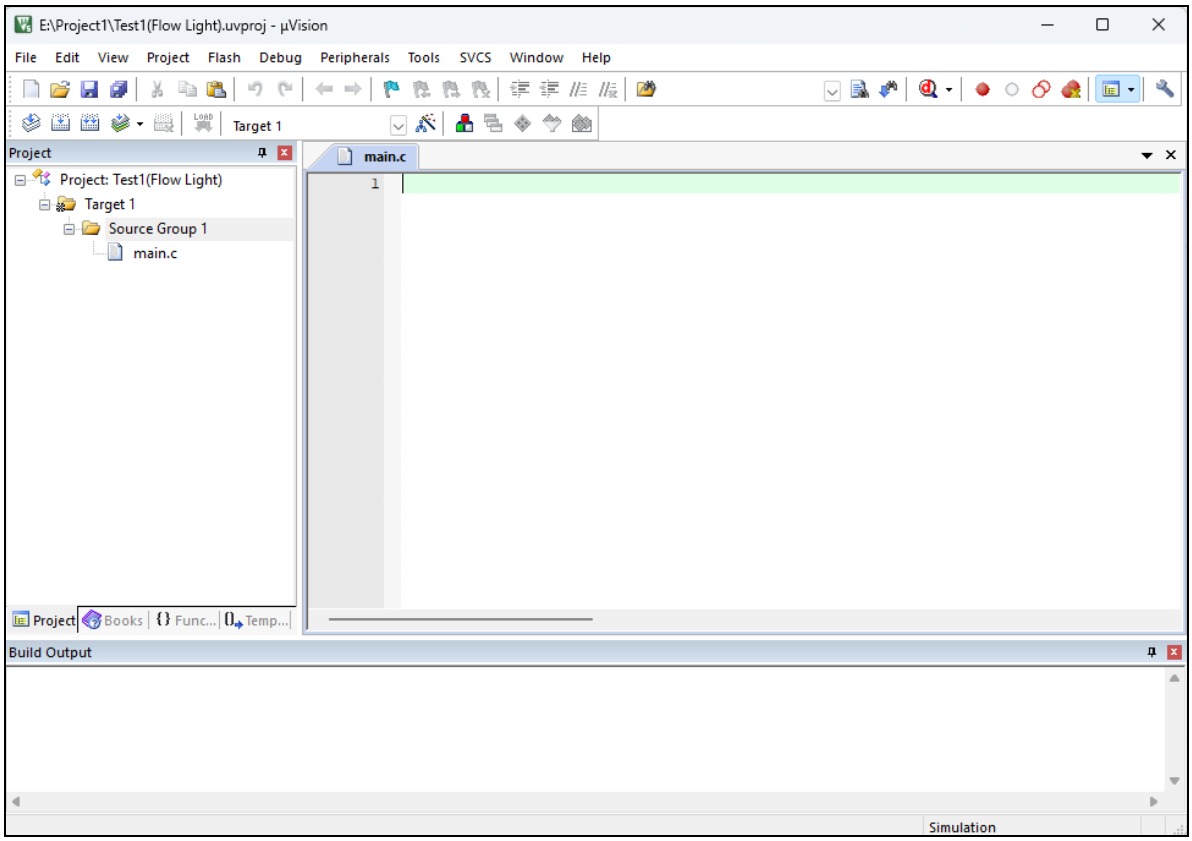
在上图中选择文件类型是：*.c

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。

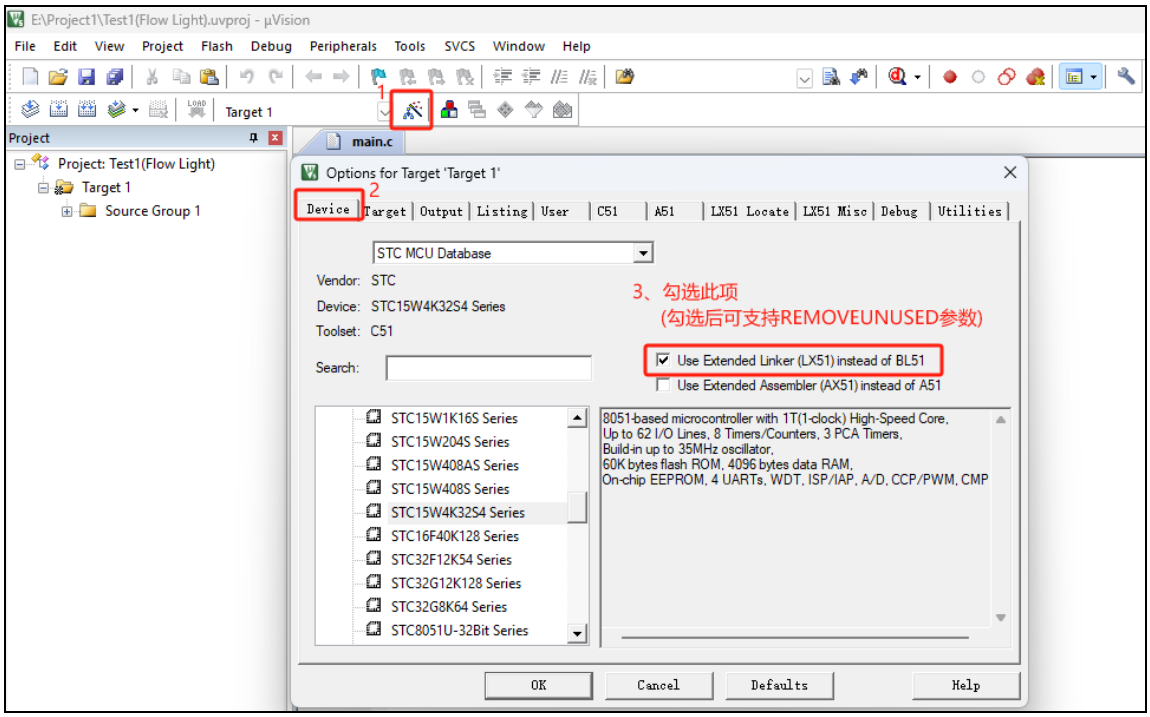


当然，在编写代码前，最好先进行 Project 的各项基础设置。

2.5.2.2 8 位 8051 工程项目的各种基础选项设置。

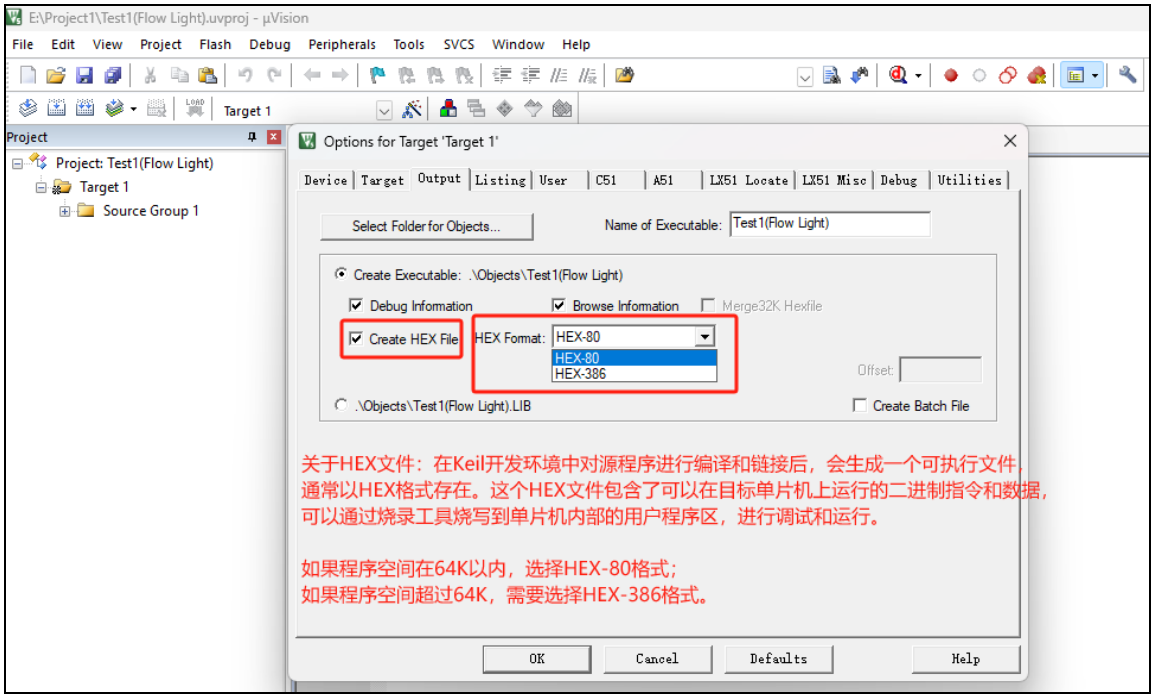
在菜单栏下方找到一个类似魔术棒的工具，就是任务选项的设置。

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



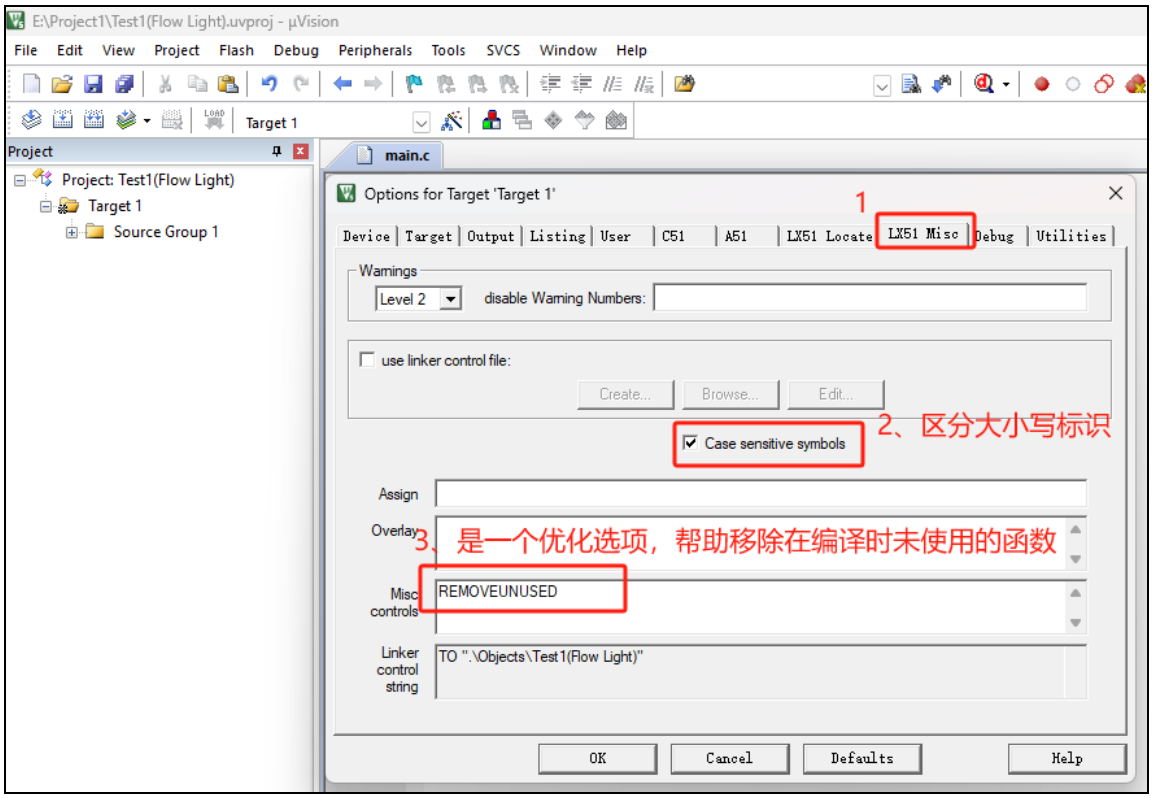
2）Output 选项卡--勾选 Create HEX File，选择 HEX-80

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。

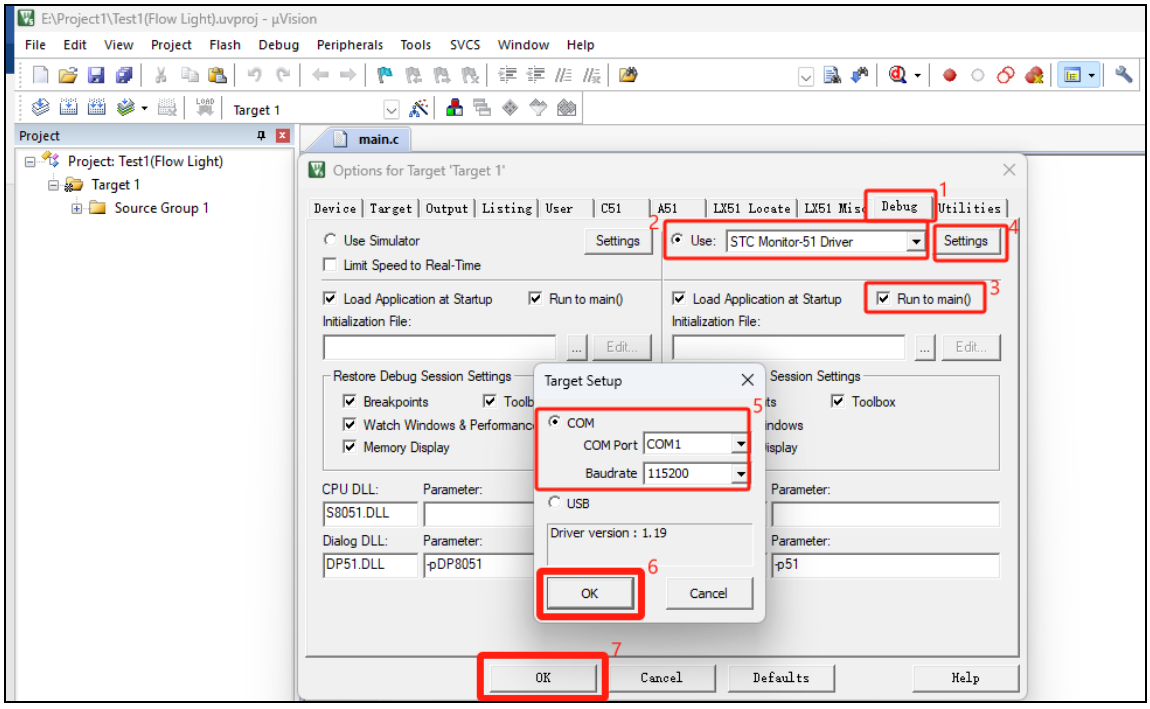


3）LX51 Misc 选项卡--在 Misc controls 输入框输入‘REMOVEUNUSED’

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。



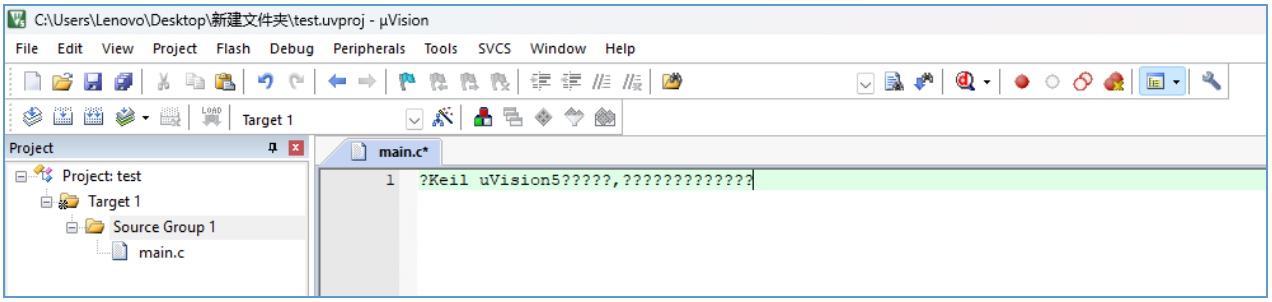
4）Debug 选项卡--仿真设置



至此，工程项目的各种基础选项设置结束。如果不使用仿真，仿真设置可以跳过。可以进入下一节，在程序文件中添加实际的代码。

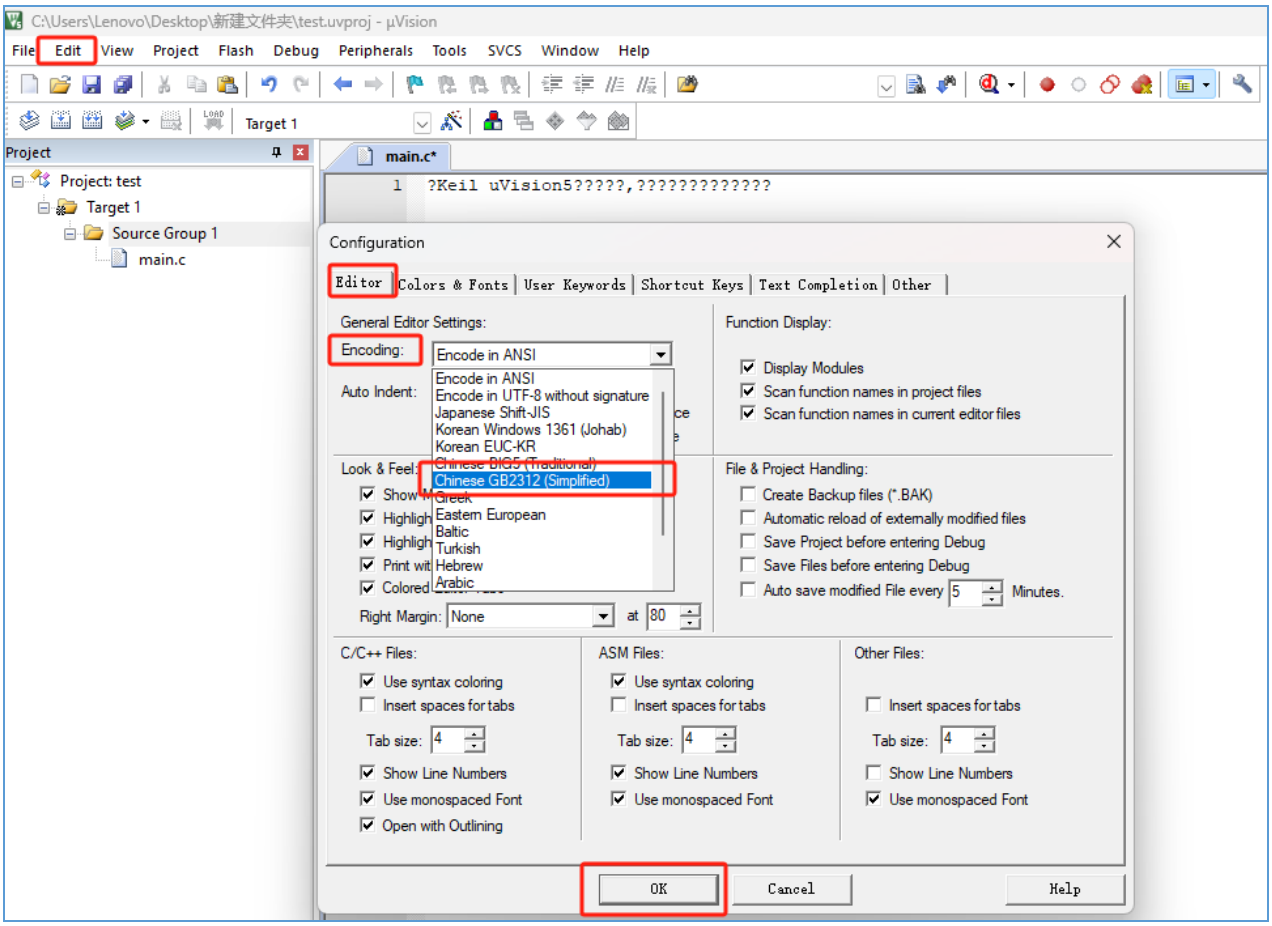
2.6 在 Keil uVision5 的编辑器中，输入中文出现乱码的解决办法

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。



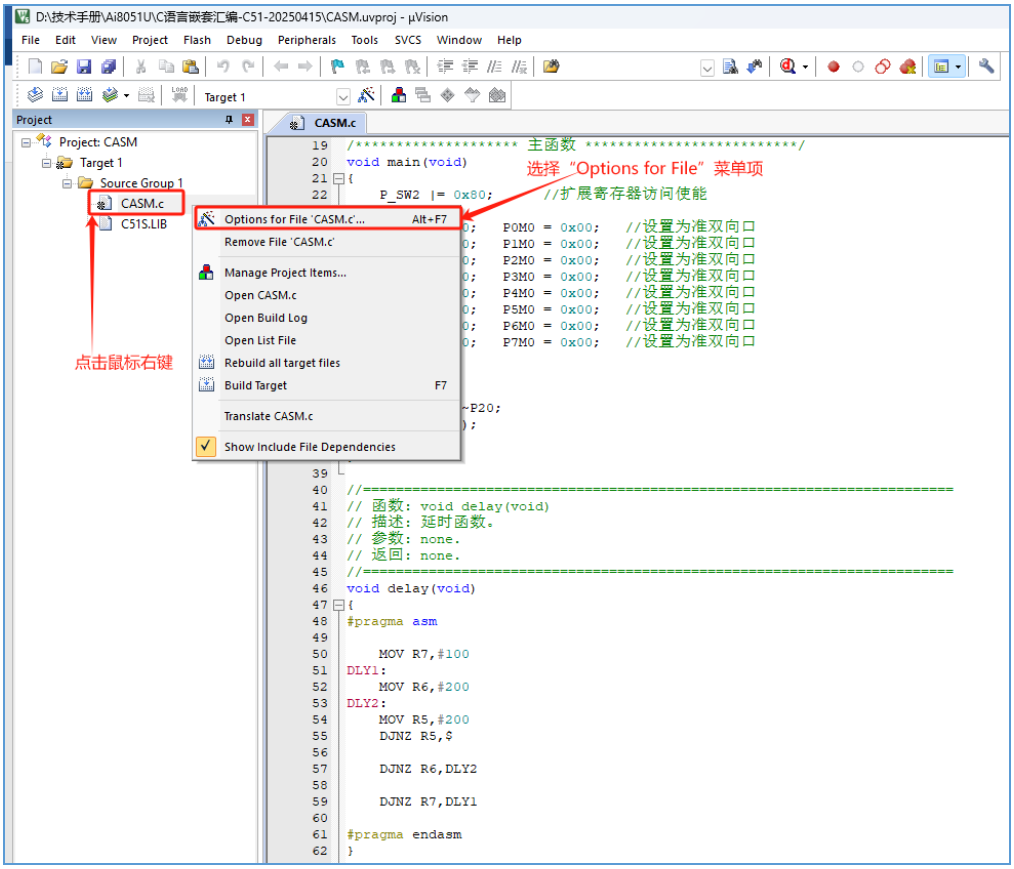
如果设置后编译器的代码中仍是乱码，则需重新复制粘贴即可正确显示。

2.7 嵌入汇编代码在 C 语言中，Keil C51 编译器

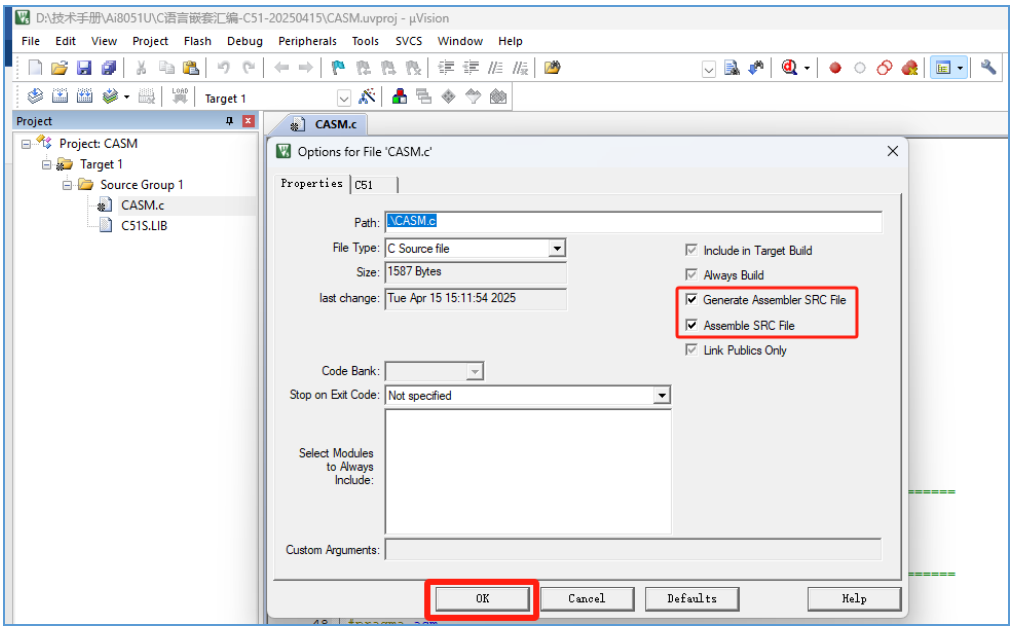
有时为了直接访问硬件资源、优化代码执行速度或精度或实现某些 C 语言本身难以直接表达的算法，需要在 C 语言文件里嵌套汇编代码。

第 1 步：新建一个工程，添加 C 文件，编译通过

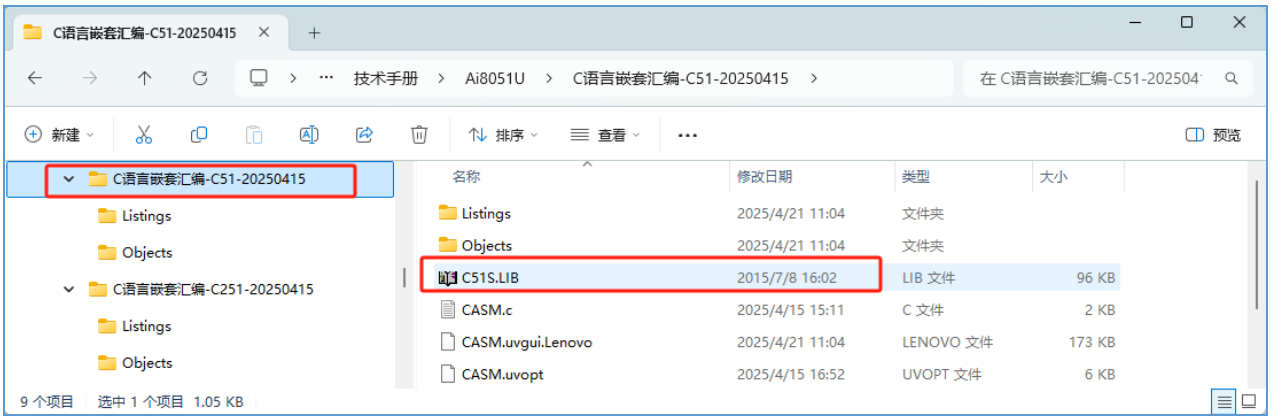
第 2 步：右击 Keil 界面中 Project Workspace 里的 C 文件名，选择‘Option for…’



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：

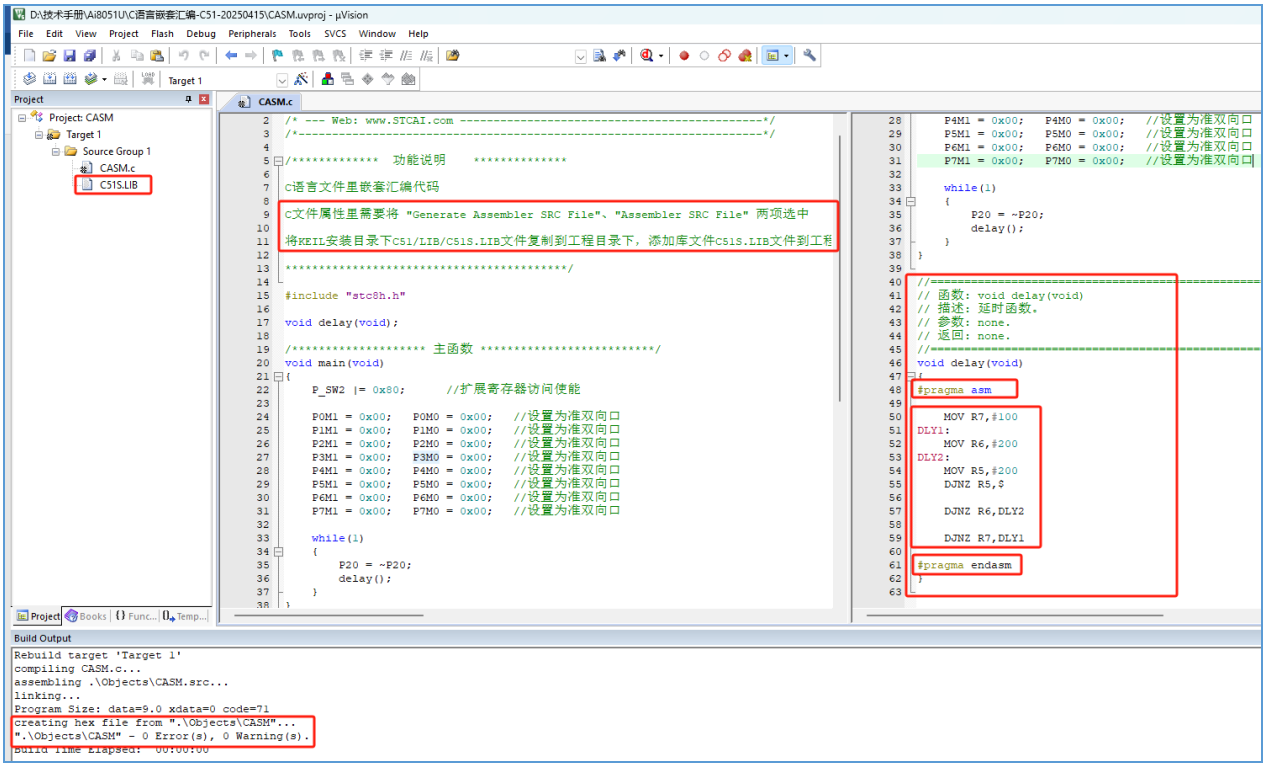
WARNING L1:UNRESOLVED EXTERNAL SYMBOL

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。

2.8 关于 Keil 软件中 0xFD 问题的说明

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：

褒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
瘕颀螨簸酏鲅鳊鼾

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。

2.9 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.dddd*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.

2.10 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。

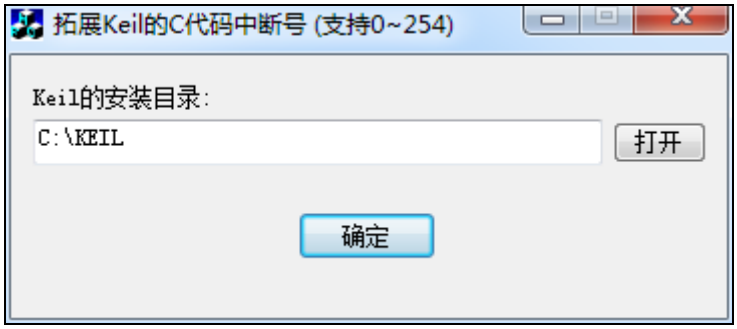
2.10.1 如何下载 Keil 中断号拓展工具，及安装

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：

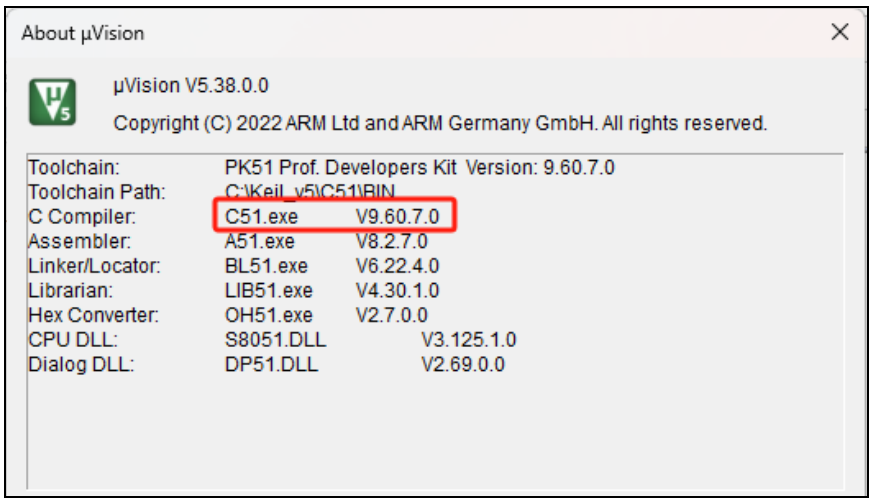
V6.12.0.1
V8.8.0.1
V9.0.0.1
V9.1.0.1
V9.53.0.0
V9.54.0.0
V9.57.0.0
V9.59.0.0
V9.60.0.0

已测试通过的 C251.EXE 版本：

V5.57.0.0
V5.60.0.0

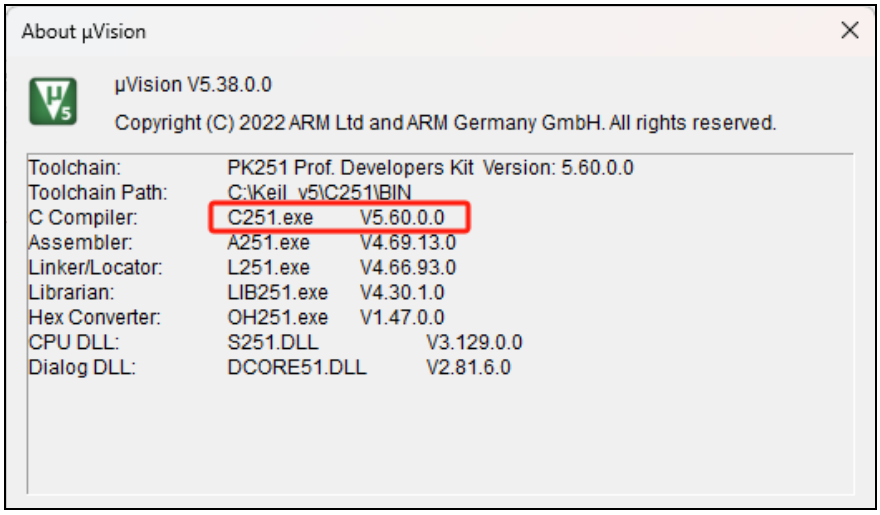
查看 C51.EXE 版本的方法：

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...”



2.10.2 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

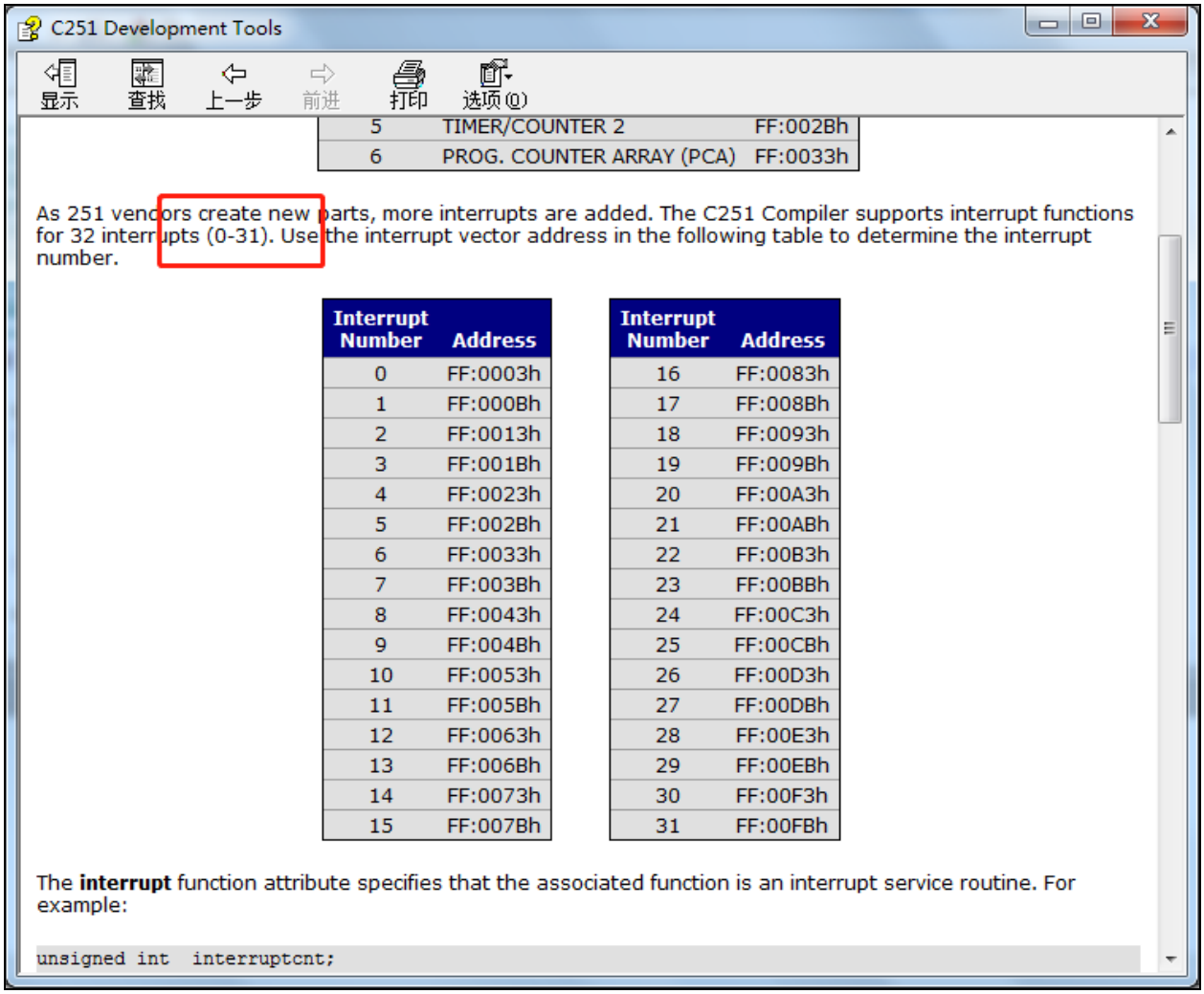
所以：

===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

2.10.3 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。

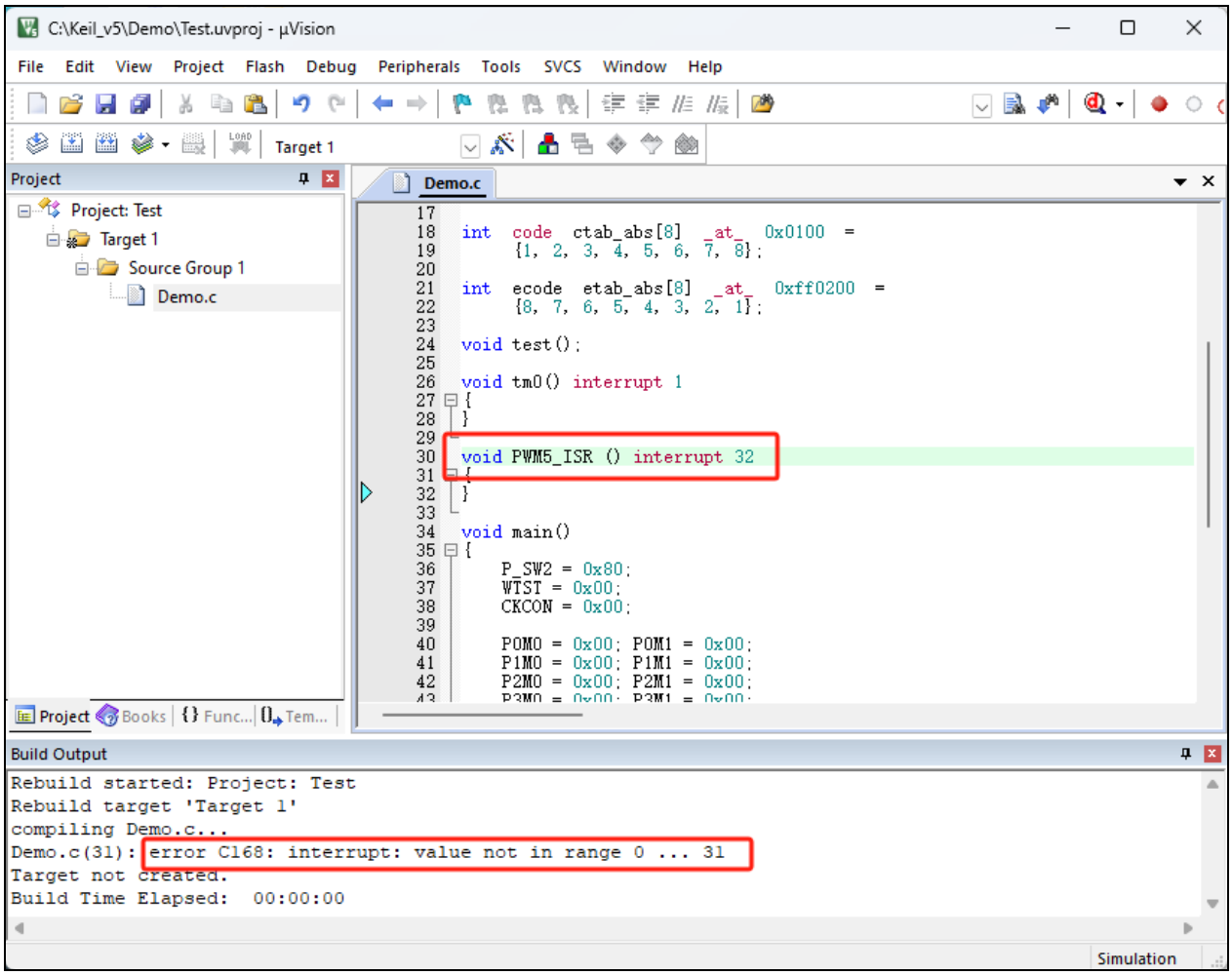


下表是目前所有系列的中断列表：

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

中断号	中断向量	中断类型
15	007B H	系统内部中断
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：



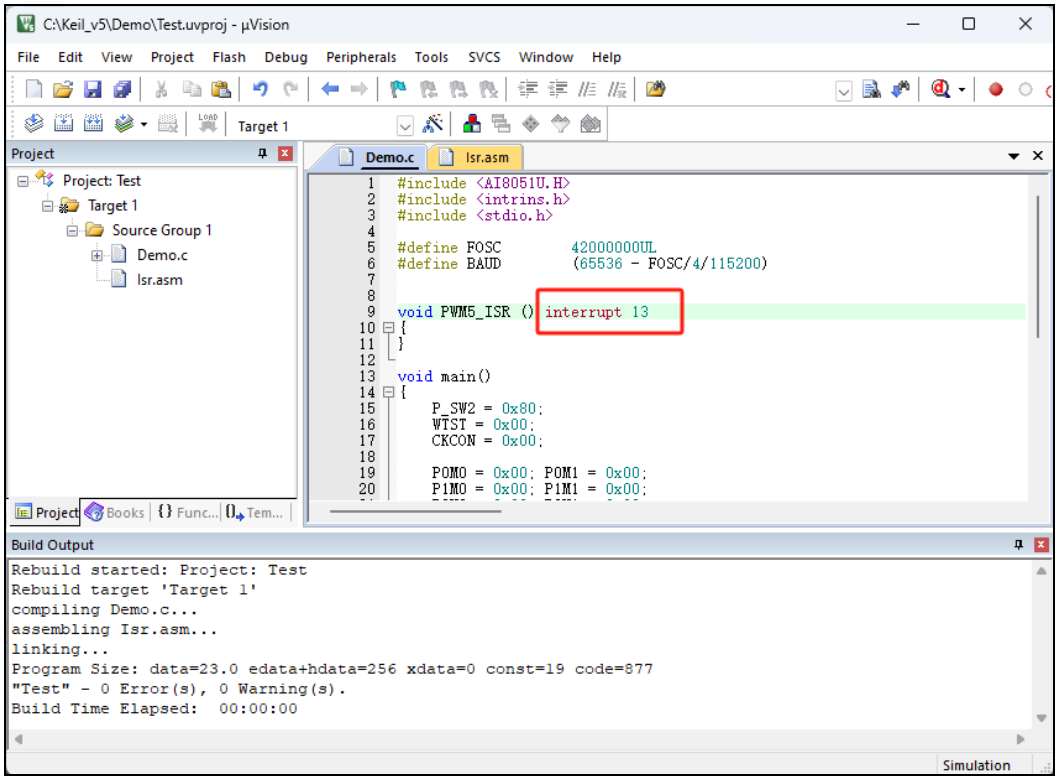
处理这种错误有如下三种方法：（均需要借助于汇编代码，优先推荐使用方法 1）

方法 1：借用 13 号中断向量

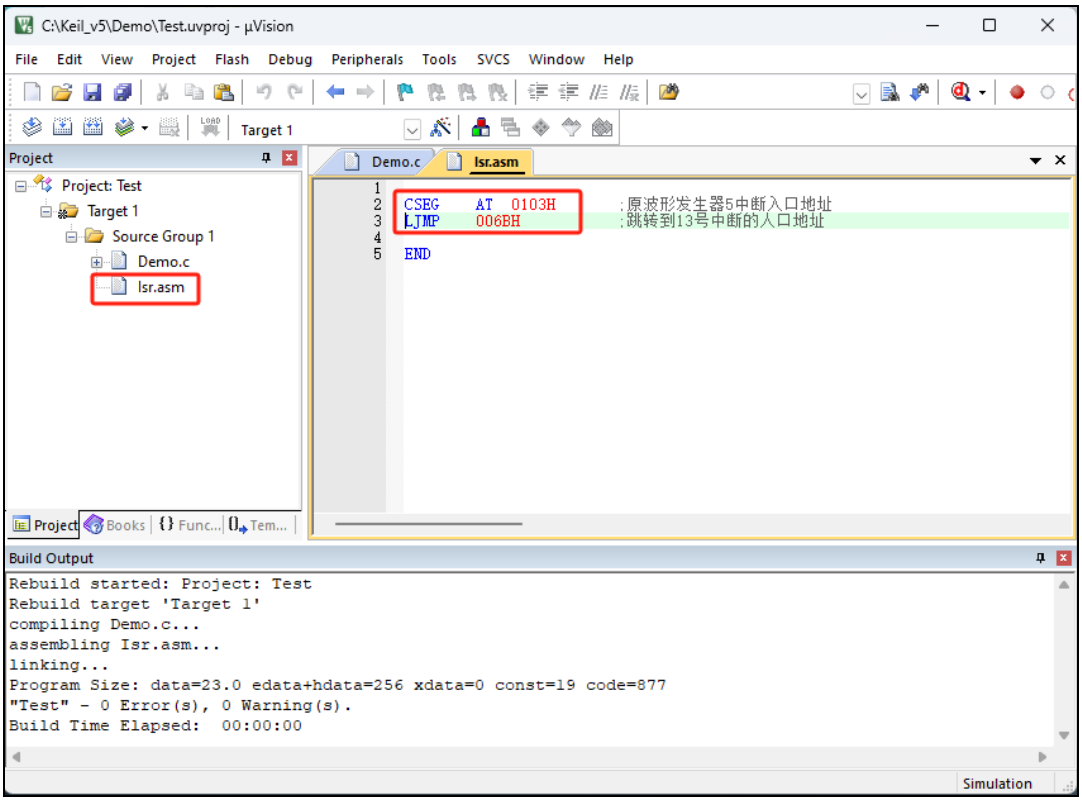
0~31 号中断中，第 13 号是保留中断号，我们可以借用此中断号

操作步骤如下：

1、将我们报错的中断号改为“13”，如下图：

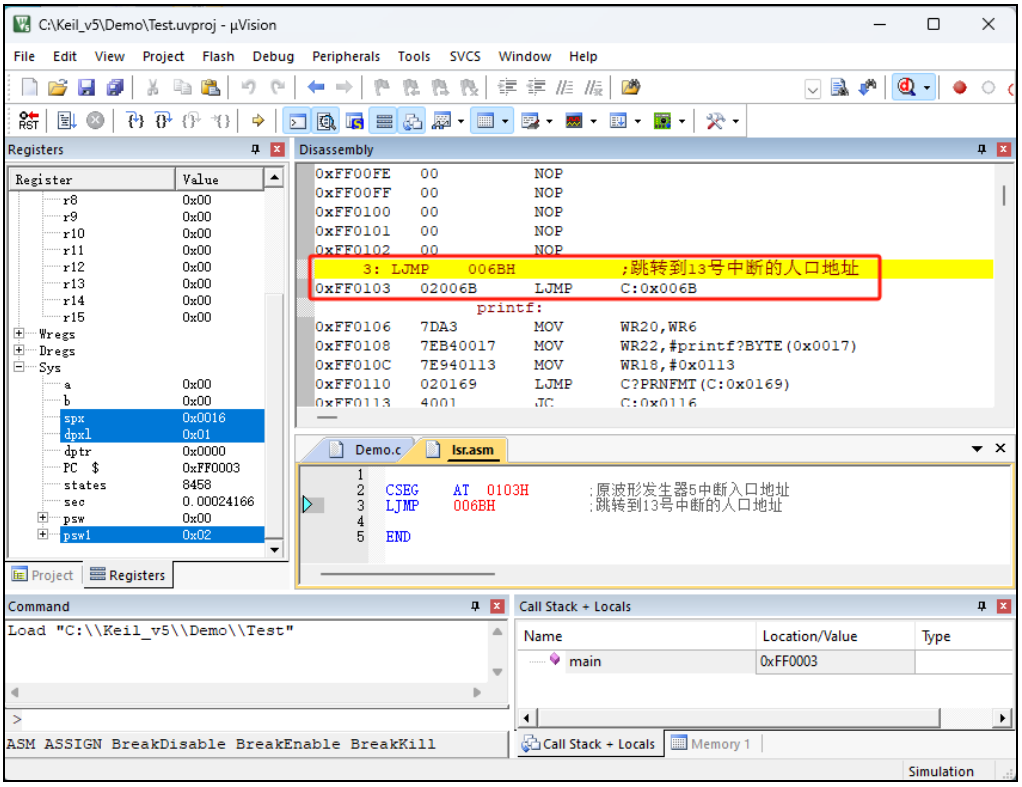


2、新建一个汇编语言文件，比如“isr.asm”，加入到项目，并在地址“0103H”的地方添加一条“LJMP 006BH”，如下图：

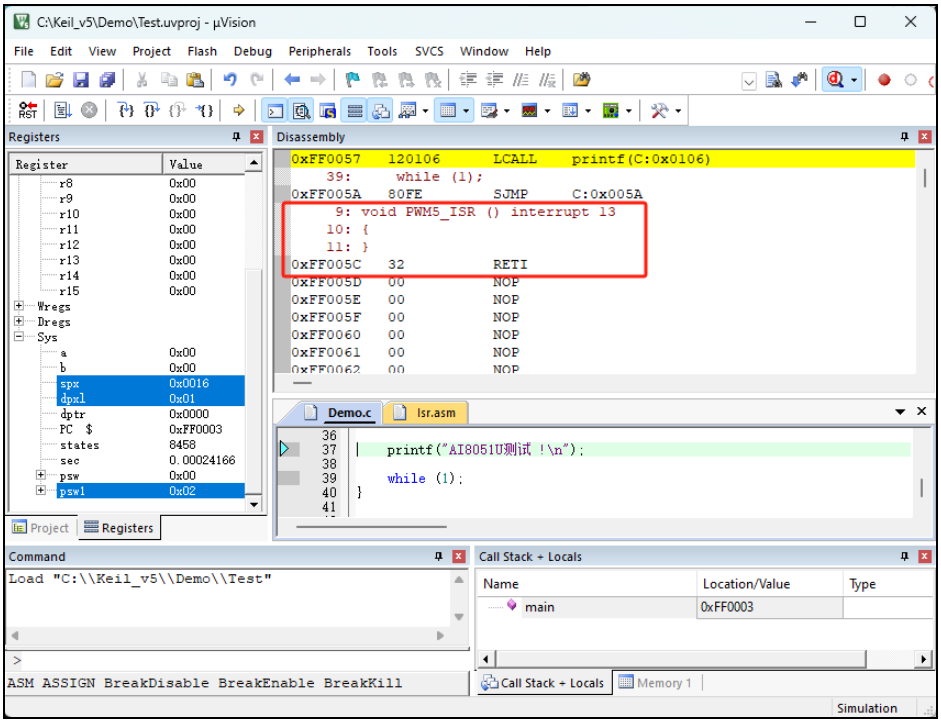


3、编译即可通过。

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：



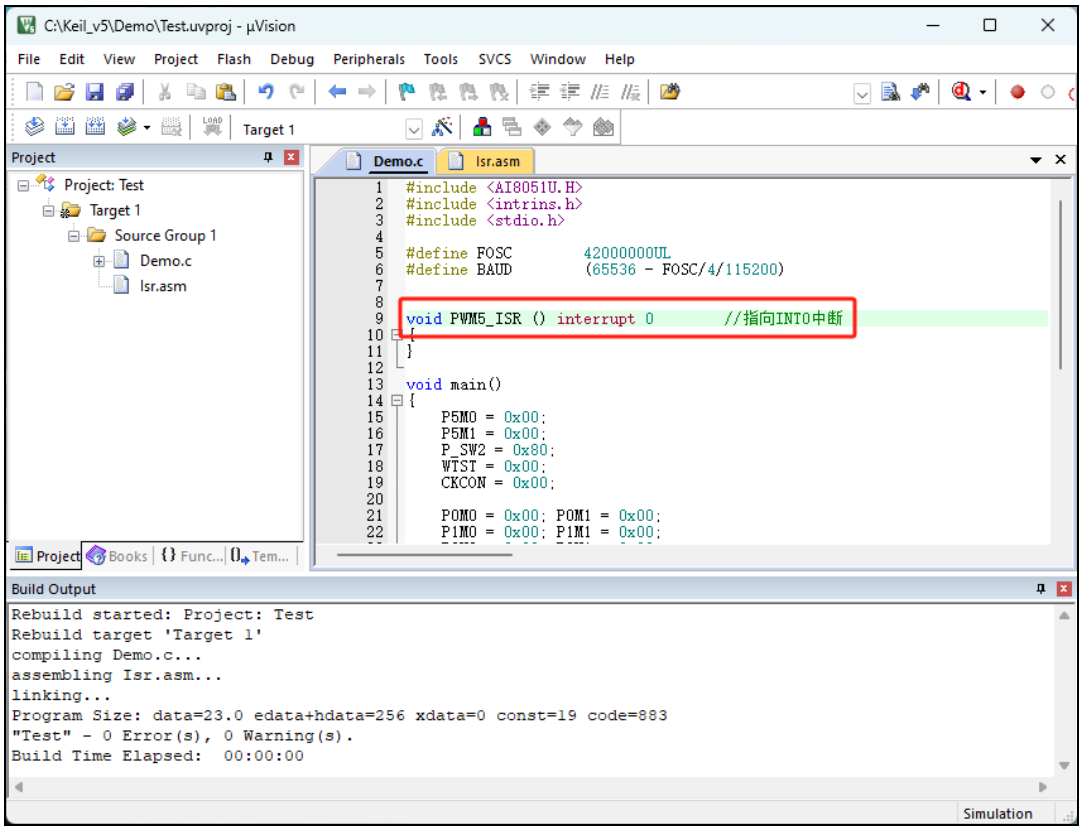
当发生 PWM5 中断时，硬件会自动跳转到 0103H 地址执行“LJMP 006BH”，然后在 006BH 处再执行“LJMP PWM5_ISR”即可跳转到真正的中断服务程序，如下图：

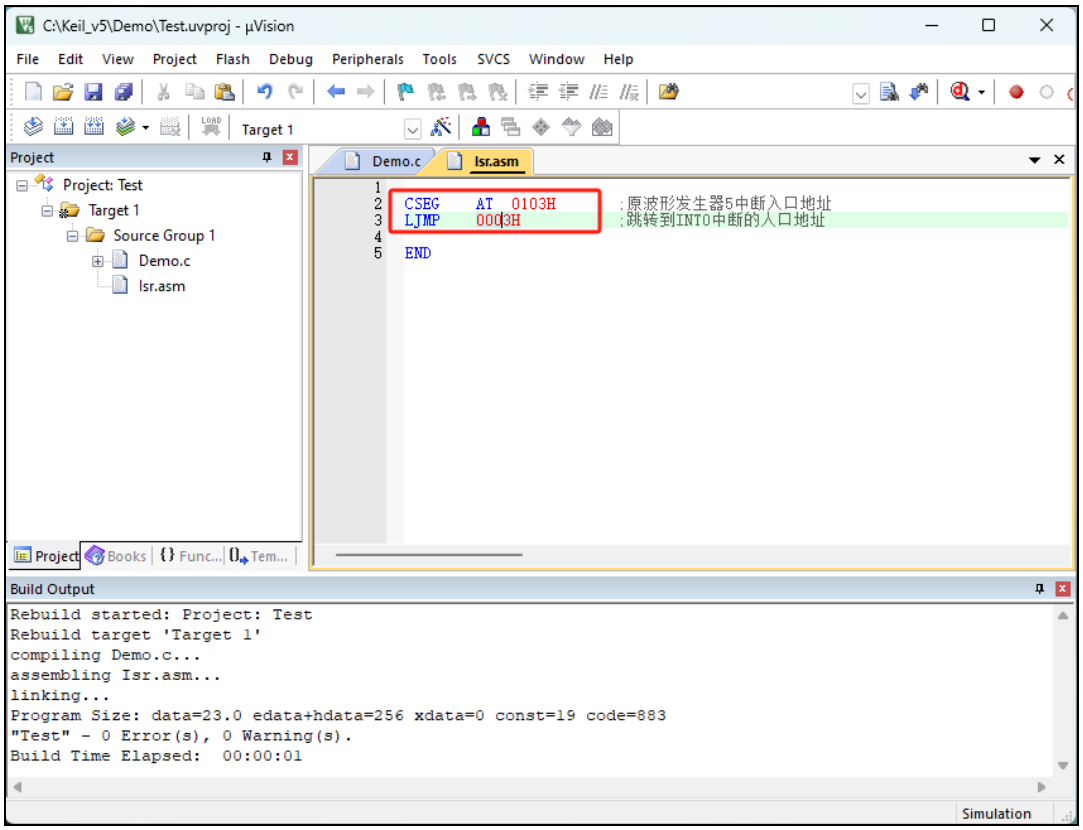


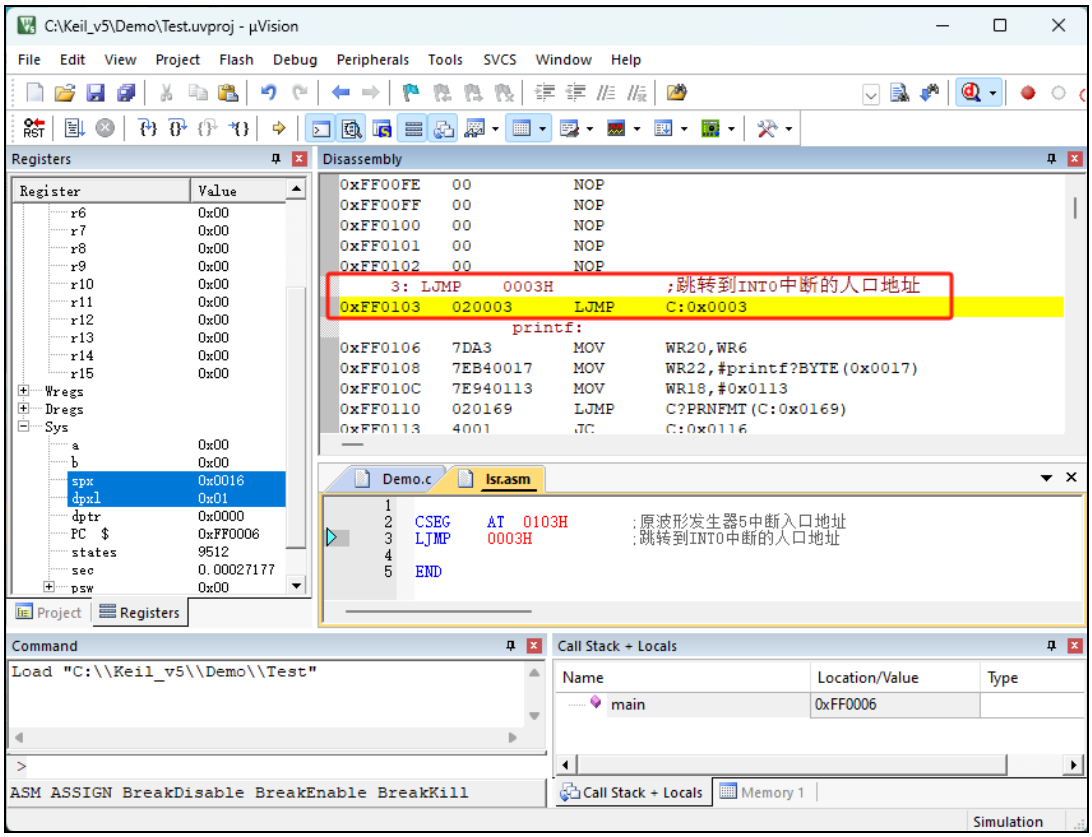
中断服务程序执行完成后，再通过 RETI 指令返回。整个中断响应过程只是多执行了一条 LJMP 语句而已。

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：





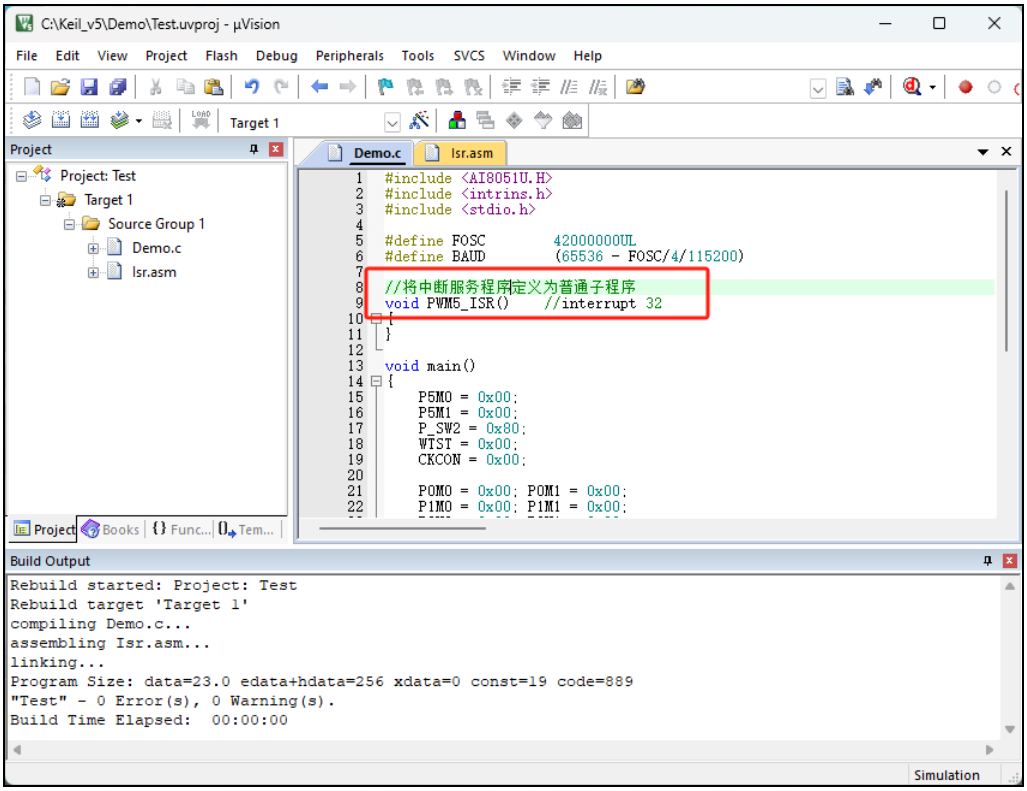


执行效果与方法 1 相同，此方法适用于需要重映射多个中断号大于 31 的情况。

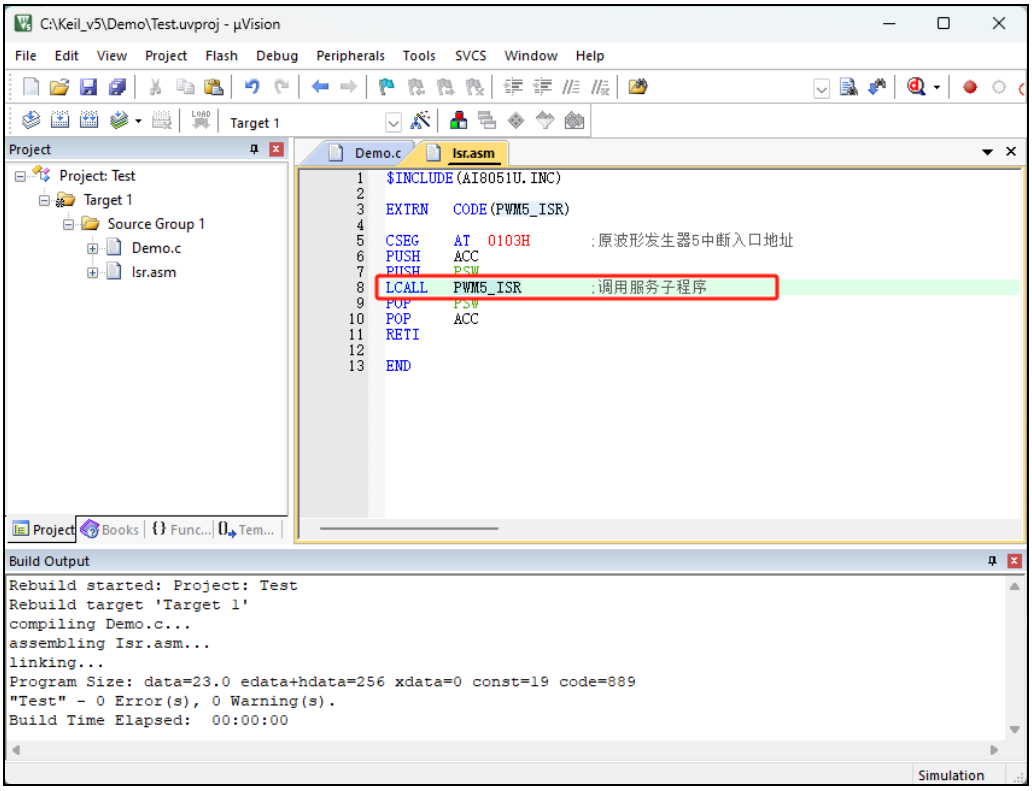
方法 3：将中断服务程序定义成子程序，然后在汇编代码中的中断入口地址中使用 LCALL 指令执行服务程序

操作步骤如下：

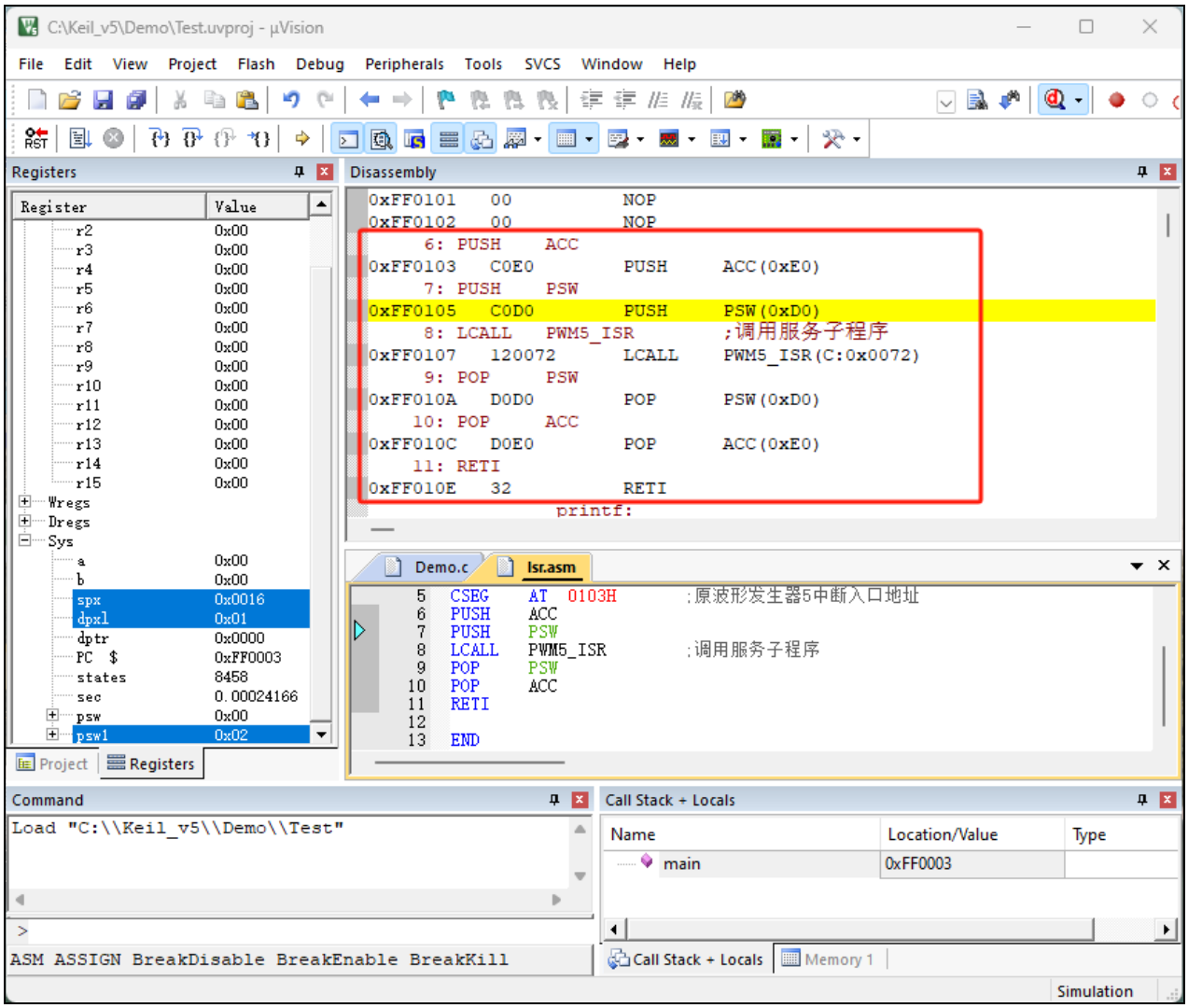
1、首先将中断服务程序去掉“interrupt”属性，定义成普通子程序



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。

2.11 单片机程序中头文件的使用方法

c 语言中 include 用法

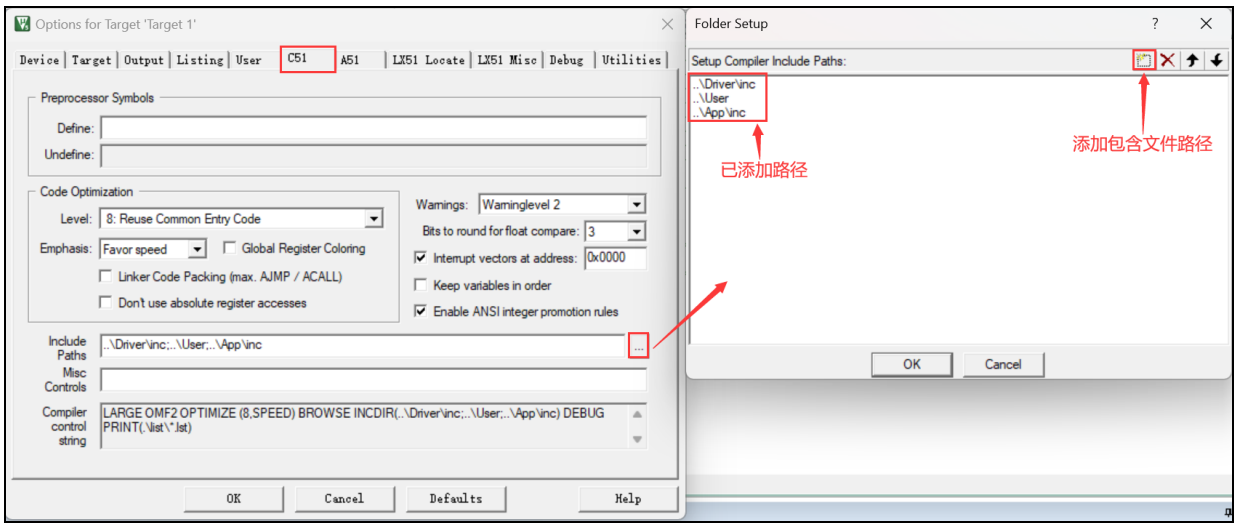
#include 命令是预处理命令的一种，预处理命令可以将别的源代码内容插入到所指定的位置。
有两种方式可以指定插入头文件：

#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。

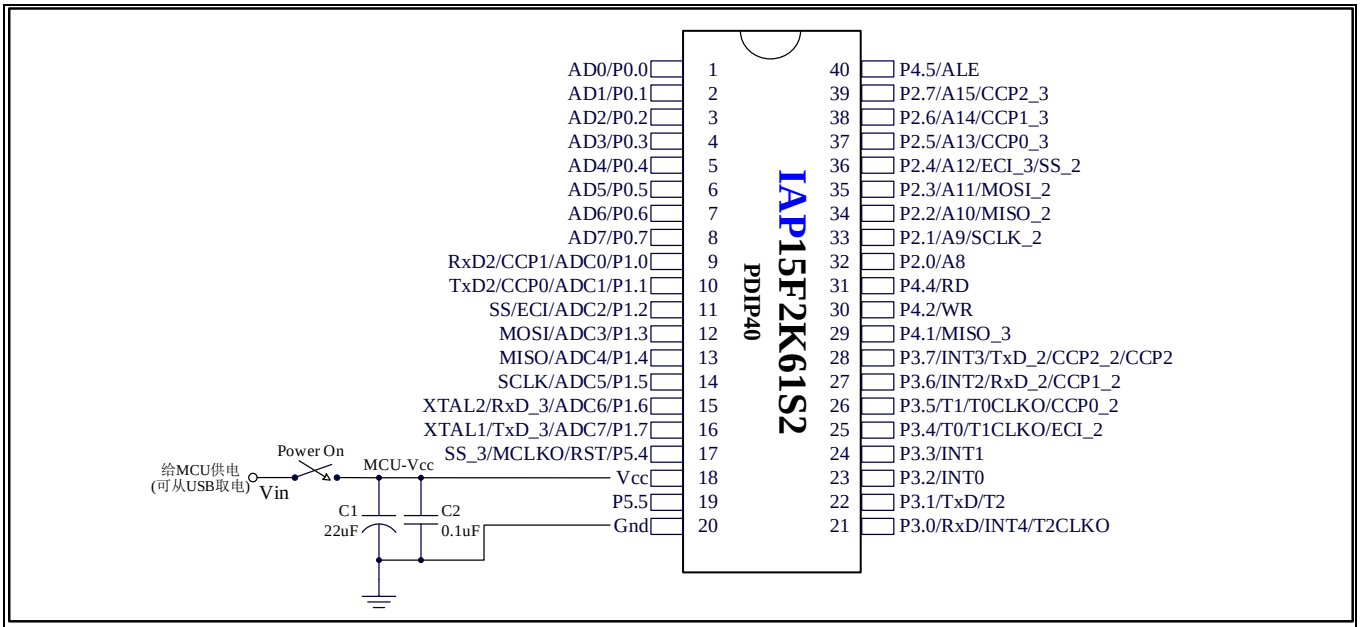
汇编语言中 include 用法与 c 语言类似，将"#"换成"$"，用小括号()包含文件：

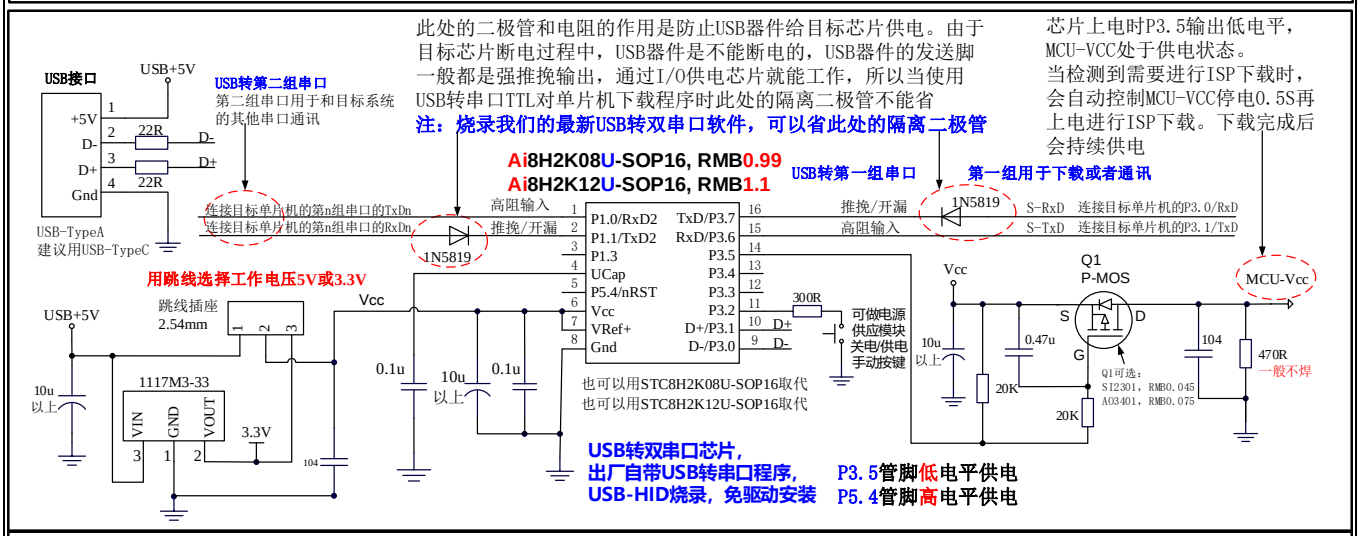
```
$include (../../comm/AI8H.INC)
```

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。

2.12 如何对 IAP15F2K61S2 系列及 IAP15W4K61S4 系列仿真

2.12.1 仿真硬件连接示意图





【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】

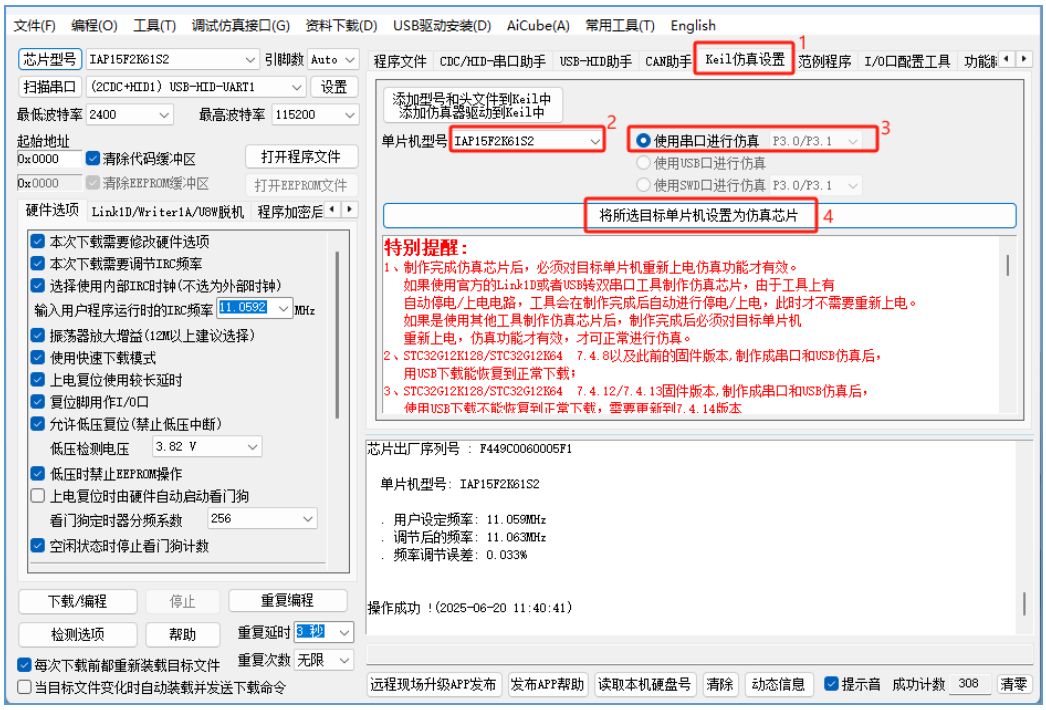
1、点击 电脑端 ISP 软件的【下载/编程】按钮

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，P5.4/RST变成复位脚。

2.12.2 制作仿真芯片及仿真流程

1、制作仿真芯片



按照上面的步骤，将目标芯片制作成仿真芯片。

1）打开 AIapp-ISP 系统软件，选择“Keil 仿真设置”选项卡

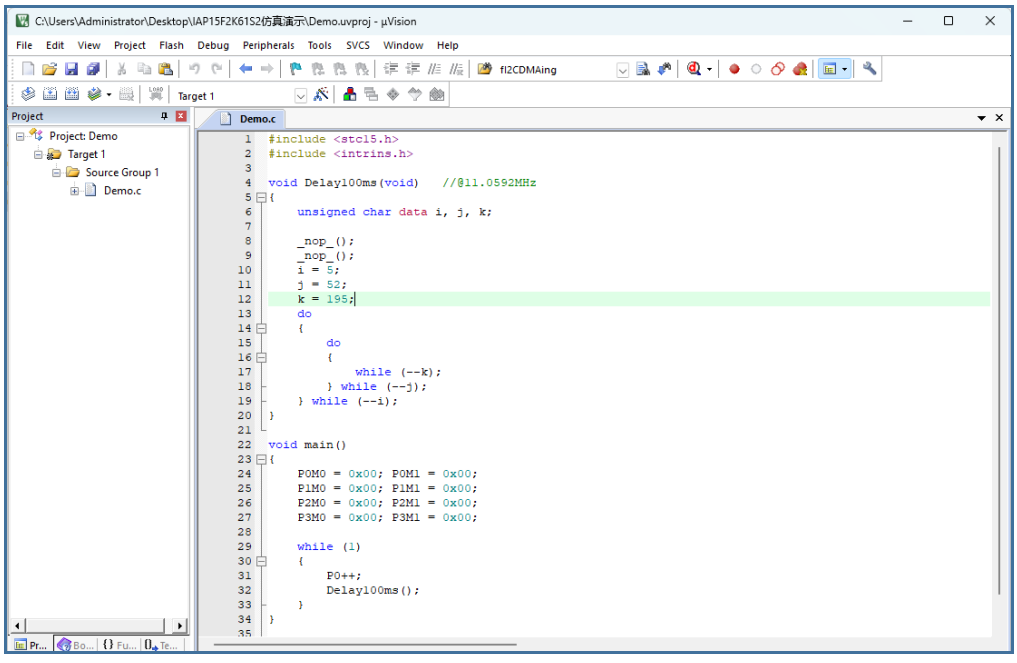
2）选择单片机型号，本例选择：IAP15F2K61S2

3）选择“使用串口进行仿真”

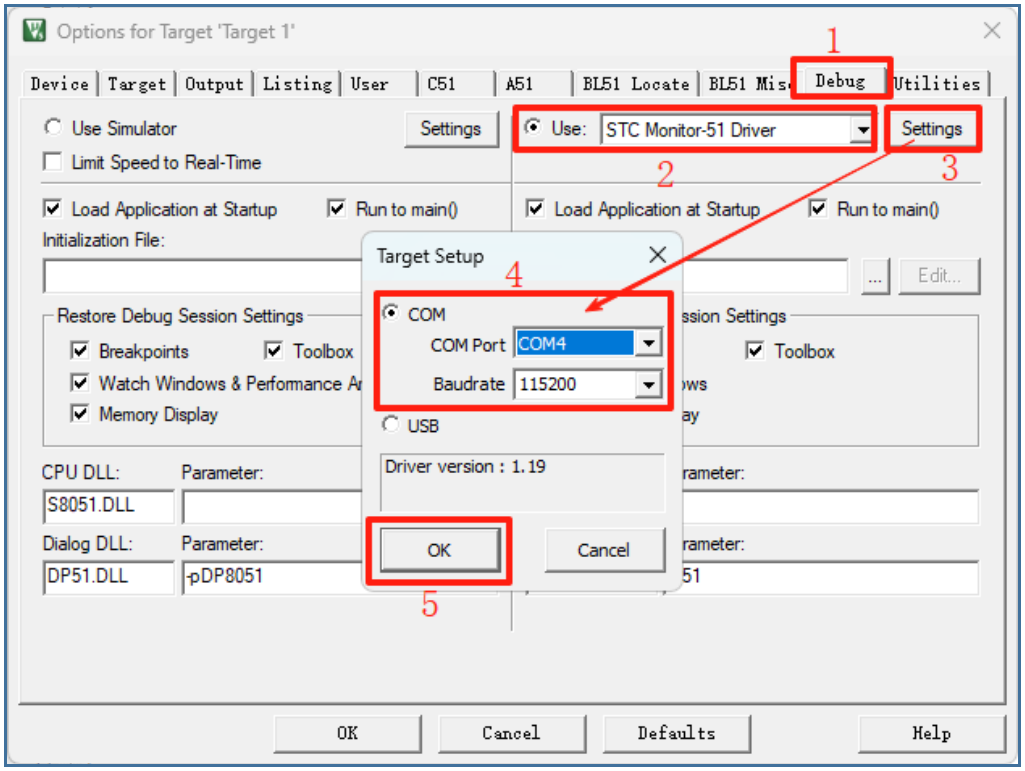
4）点击“将所选目标单片机设置为仿真芯片”按钮

如果使用的下载工具不是 STC 的 USB 转双串口工具或者 Link1D 工具，则制作完成后必须将目标芯片重新断电再上电，仿真功能才能生效

2、新建 Keil 项目，并将代码文件加入到项目中



3、Keil 仿真设置



按照上图的顺序，对 Keil 仿真进行设置

1）打开 Keil 项目选项窗口中的“Debug”页面

2）选择仿真驱动“STC Monitor-51 Driver”

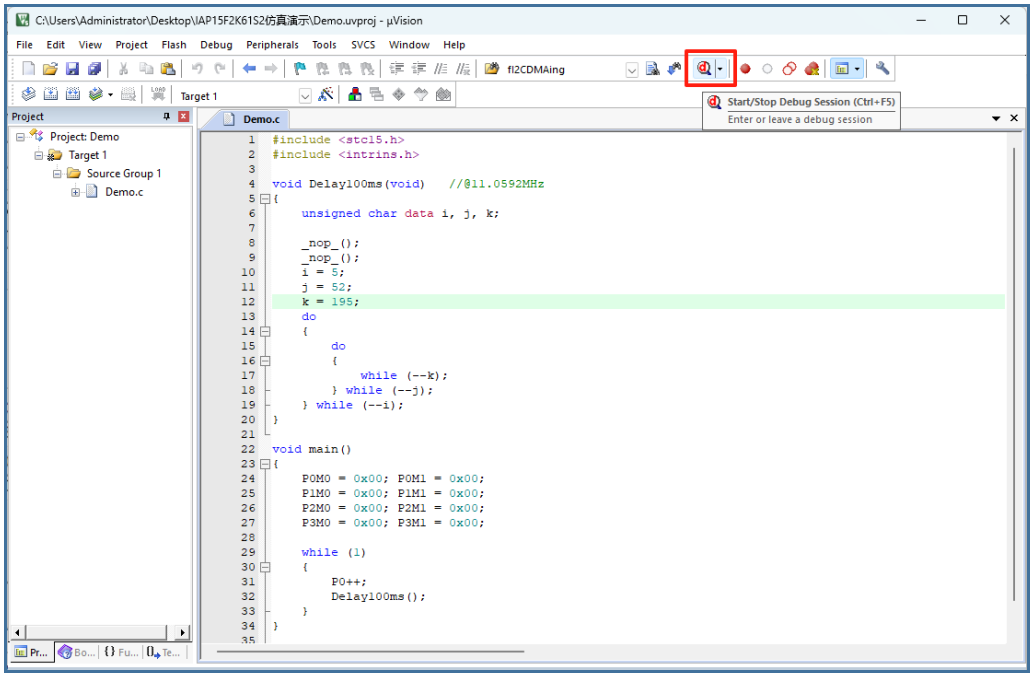
3）点击“Settings”按钮进入仿真接口设置

4）选择正确的串口号，并设置波特率为 115200

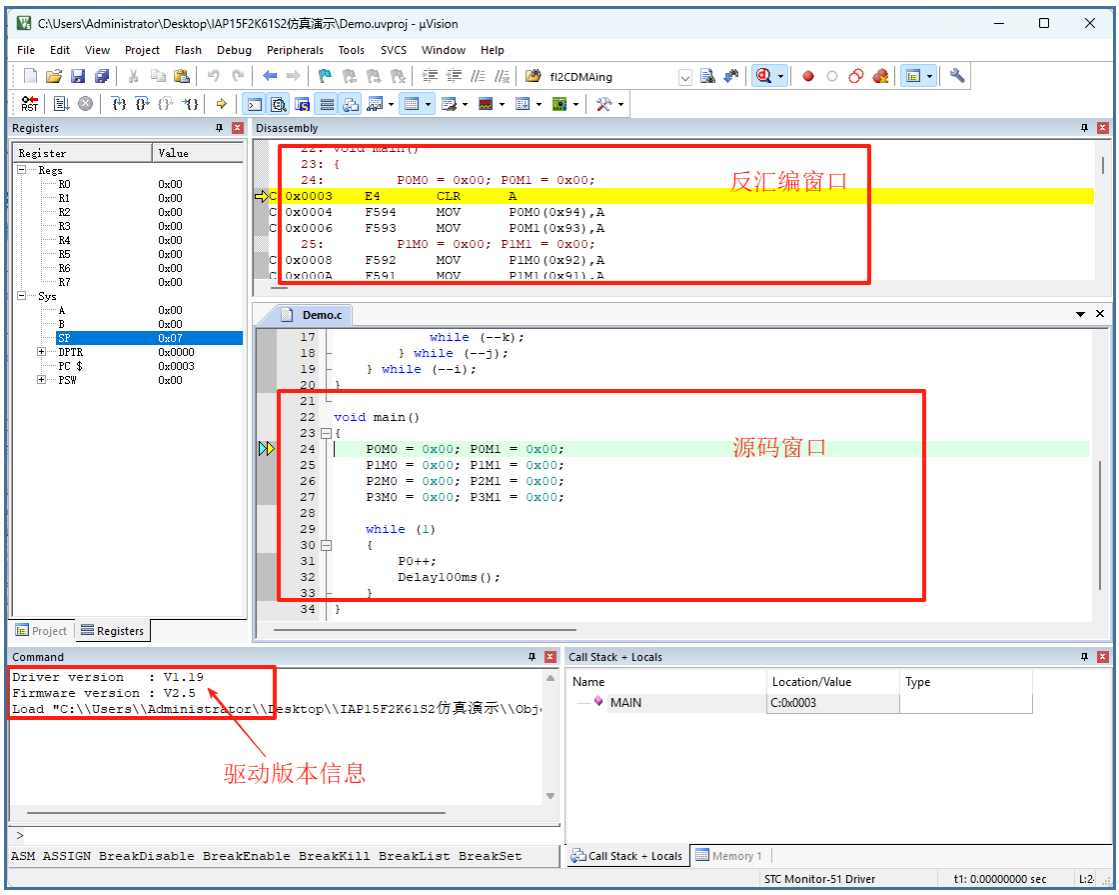
5）点击“OK”按钮即可完成设置

4、开始仿真

点击 Keil 界面中的“Start/Stop Debug”按钮，进入仿真状态



如下图，进入仿真状态后即可进行常规的单步、设置断点、全速运行等仿真操作了



3 STC15 系列选型简介、特性/价格/管脚图/ISP 下载/编程线路图

3.1 STC15F104W 系列

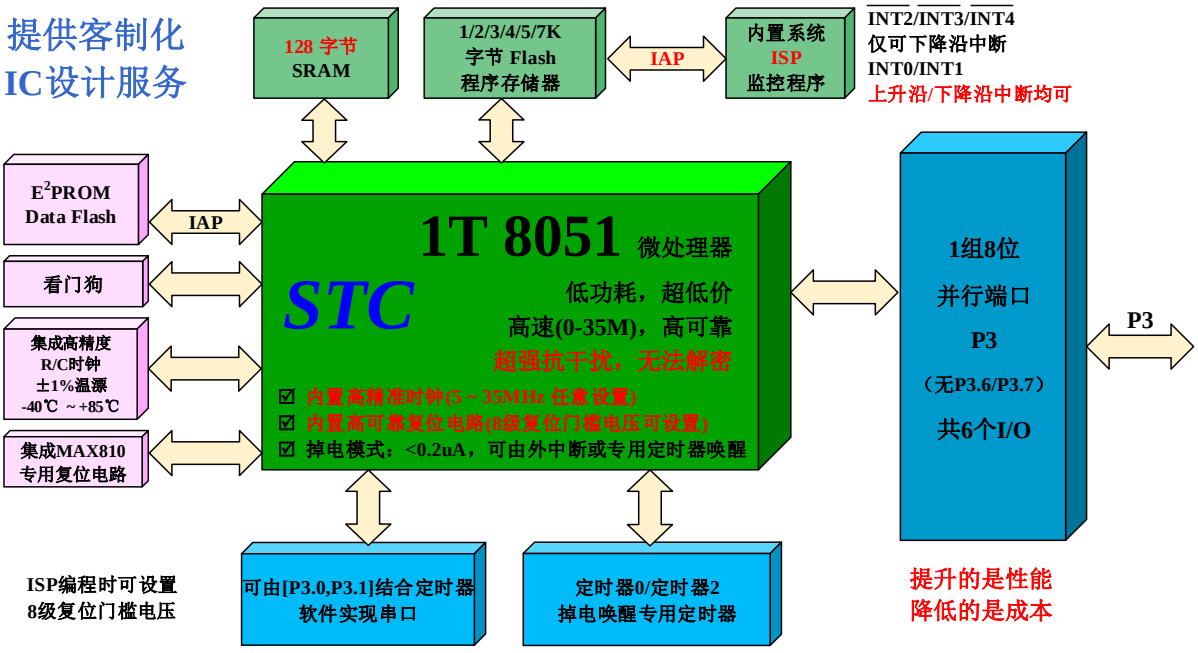
----STC15F104W 系列建议用 STC15W104 系列取代

3.1.1 STC15F104W 系列单片机简介（与 STC15F104E 系列有所不兼容）

STC15F104W 系列单片机是 STC 生产的单时钟/机器周期（1T）的单片机，是高速/高可靠/低功耗/超强抗干扰的新一代 8051 单片机，采用 STC 第八代加密技术，无法解密，指令代码完全兼容传统 8051，但速度快 8--12 倍。内部集成高精度 R/C 时钟（±0.3%），±1%温漂（-40℃～+85℃），常温下温漂±0.6%（-20℃～+65℃），ISP 编程时 5MHz～35MHz 宽范围可设置，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 8 级复位门槛电压可选）。

在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可

现 STC15 系列单片机采用 STC-Y5 超高速 CPU 内核，在相同的时钟频率下，速度又比 STC 早期的 1T 系列单片机（如 STC12 系列/STC11 系列/STC10 系列）的速度快 20%。



1. 增强型 8051 CPU，1T，单时钟/机器周期，速度比普通 8051 快 8 - 12 倍
2. 工作电压：

   STC15F104W 系列工作电压：5.5V - 3.8V（5V 单片机）

   STC15L104W 系列工作电压：3.6V - 2.4V（3V 单片机）
3. 1K/2K/3K/4K/5K/7K 字节片内 Flash 程序存储器，可擦写次数 10 万次以上
4. 片内 128 字节的 SRAM
5. 有片内 EEPROM 功能，擦写次数 10 万次以上
6. ISP/IAP，在系统可编程/在应用可编程，无需编程器/仿真器

7. 内部高可靠复位，ISP 编程时 8 级复位门槛电压可选，可彻底省掉外部复位电路
8. 工作频率范围：5MHz ~ 35MHz，相当于普通 8051 的 60MHz ~ 420MHz
9. 内部高精度 R/C 时钟（±0.3%），±1%温漂（-40℃ ~ +85℃），常温下温漂±0.6%（-20℃ ~ +65℃），ISP 编程时内部时钟从 5MHz ~ 35MHz 可设（5.5296MHz/11.0592MHz/22.1184MHz/33.1776MHz）
10. 不需外部晶振和外部复位，还可对外输出时钟和低电平复位信号
11. 串口功能可由[P3.0/INT4，P3.1]结合定时器实现
12. 支持程序加密后传输，防拦截
13. 支持 RS485 下载
14. 低功耗设计：低速模式，空闲模式，掉电模式/停机模式
15. 可将掉电模式/停机模式唤醒的定时器：有内部低功耗掉电唤醒专用定时器。
16. 可将掉电模式/停机模式唤醒的资源有：
    - INT0/P3.2，INT1/P3.3（INT0/INT1 上升沿下降沿中断均可），INT2/P3.4，INT3/P3.5，INT4/P3.0（INT2/INT3/INT4 仅可下降沿中断）；
    - 管脚 T0/T2（下降沿，不产生中断，前提是在进入掉电模式/停机模式前相应的定时器中断已经被允许）；
    - 内部低功耗掉电唤醒专用定时器。
17. 共 2 个定时器/计数器----T0（兼容普通 8051 的定时器）/T2，并均可实现可编程时钟输出，另外管脚 MCLKO 可将内部主时钟对外分频输出（÷1 或÷2 或÷4）
18. 可编程时钟输出功能（对内部系统时钟或对外部管脚的时钟输入进行时钟分频输出）：
    - 由于 STC15 系列 5V 单片机 I/O 口的对外输出速度最快不超过 13.5MHz，所以 5V 单片机的对外可编程时钟输出速度最快也不超过 13.5MHz.；
    - 而 3.3V 单片机 I/O 口的对外输出速度最快不超过 8MHz，故 3.3V 单片机的对外可编程时钟输出速度最快也不超过 8MHz.

    1) T0 在 P3.5/T0CLKO 进行可编程输出时钟（对内部系统时钟或对外部管脚 T0/P3.4 的时钟输入进行可编程时钟分频输出）；
    2) T2 在 P3.0/T2CLKO 进行可编程输出时钟（对内部系统时钟或对外部管脚 T2/P3.1 的时钟输入进行可编程时钟分频输出）；

    以上 2 个定时器/计数器均可 1 ~ 65536 级分频输出。

    3) 主时钟在 P3.4/MCLKO 对外输出时钟，并可如下分频 MCLK/1，MCLK/2，MCLK/4

    STC15F104W 系列单片机不支持外接外部晶体，其主时钟对外输出管脚 P3.4/MCLKO 只可以对外输出内部 R/C 时钟。MCLK 是指主时钟频率，MCLKO 是指主时钟输出。

    STC15 系列 8-pin 单片机（如 STC15F104W 系列）在 MCLKO/P3.4 口对外输出时钟，STC15 系列 16-pin 及其以上单片机均在 MCLKO/P5.4 口对外输出时钟，且 STC15W 系列 20-pin 及其以上单片机除可在 MCLKO/P5.4 口对外输出时钟外，还可在 MCLKO_2/P1.6 口对外输出时钟。
19. 硬件看门狗（WDT）
20. 先进的指令集结构，兼容普通 8051 指令集，有硬件乘法/除法指令
21. 共 6 个通用 I/O 口，复位后为：准双向口上拉（普通 8051 传统 I/O 口）

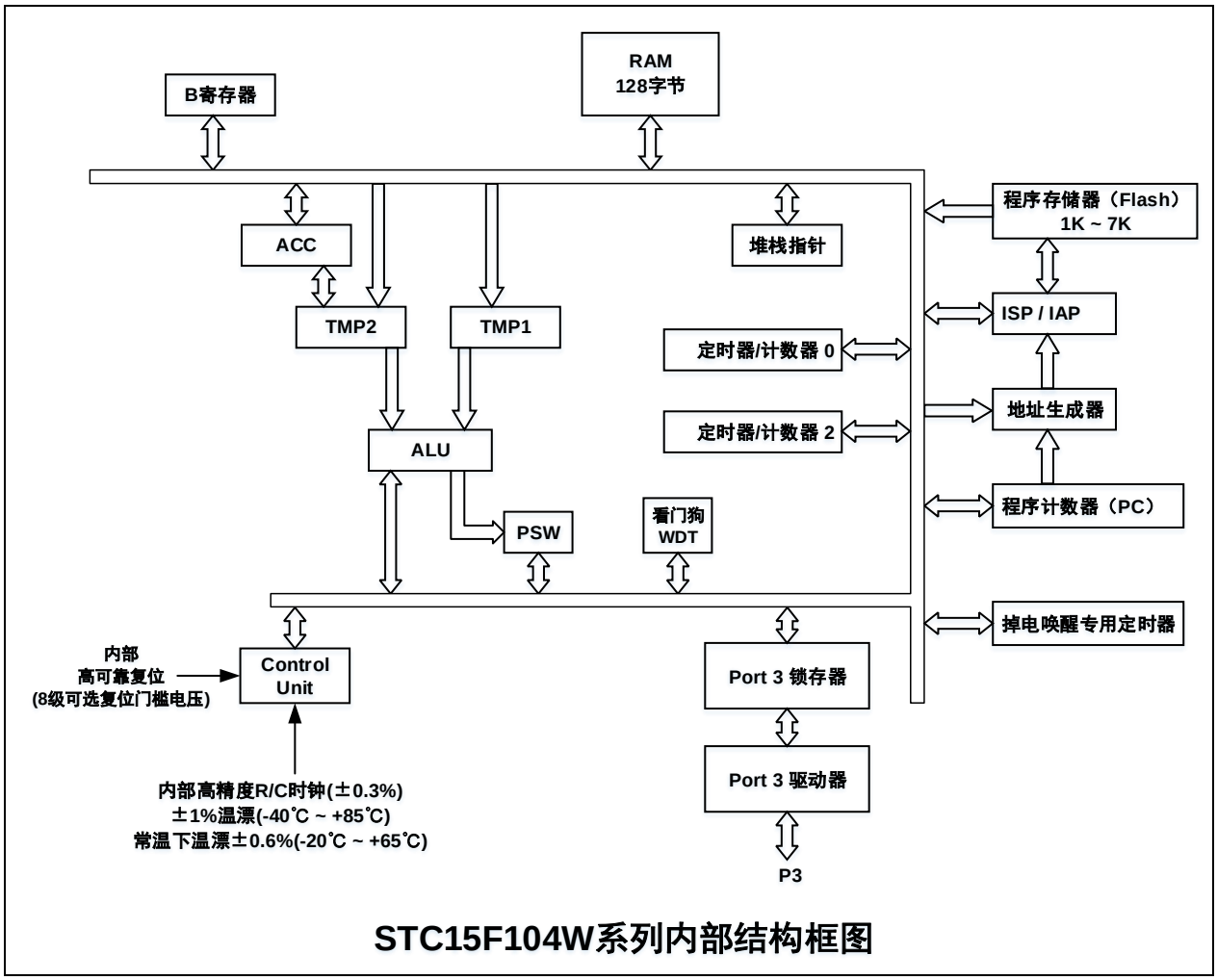
    可设置成四种模式：准双向口/弱上拉，强推挽/强上拉，仅为输入/高阻，开漏
    每个 I/O 口驱动能力均可达到 20mA，但整个芯片电流最大不要超过 90mA

如果 I/O 口不够用，可外接 74HC595（参考价 0.15 元）来扩展 I/O 口，并可多芯片级联扩展几十个 I/O 口。

22. 封装：SOP-8，DIP-8，DFN-8（不推荐）
23. 全部 175℃八小时高温烘烤，高品质制造保证
24. 开发环境：在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可

3.1.2 内部结构图

STC15F104W 系列单片机的内部结构框图如下图所示。STC15F104W 系列单片机中包含中央处理器（CPU）、程序存储器（Flash）、数据存储器（SRAM）、定时器/计数器、掉电唤醒专用定时器、I/O 口、看门狗、片内高精度 R/C 时钟及高可靠复位等模块。



STC15F104W系列内部结构框图

3.1.3 STC15F104W 系列单片机选型价格一览表

型号	工作电压(V)	Flash 程序存储器(字节 byte)	SRAM 字节	串行口并可掉电唤醒	SPI	定时器计数器 T0/T2 外部管脚也能掉电唤醒	CCP PCA PWM 并可掉电唤醒	掉电唤醒专用定时器	标准外部中断支持掉电唤醒	A/D8 路(3 路 PWM 可当 3 路 D/A 使用)	DPTR	EEPROM	内部低压检测中断并可掉电唤醒	看门狗	内部高可靠复位(可选复位门电压)	内部高精准时钟	可对外输出时钟及复位	程序加密后传输(防拦截)	可设下次更新程序需口令	支持 RS485 下载	8-Pin 封装 SOP8 / DIP8 /DFN8 (6 个 I/O 口) 价格(RMB ¥)		
																					SOP8	DIP8	DFN8
STC15F104W 系列单片机选型价格一览表																							
STC15F100W	5.5-3.8	0.5K	128	-	-	2	-	有	5	-	1	-	有	有	8 级	有	是	有	是	是	¥1.20	¥1.40	¥1.60
STC15F101W	5.5-3.8	1K	128	-	-	2	-	有	5	-	1	4K	有	有	8 级	有	是	有	是	是	¥1.20	¥1.40	¥1.60
STC15F102W	5.5-3.8	2K	128	-	-	2	-	有	5	-	1	3K	有	有	8 级	有	是	有	是	是	¥1.20	¥1.40	¥1.60
STC15F103W	5.5-3.8	3K	128	-	-	2	-	有	5	-	1	2K	有	有	8 级	有	是	有	是	是	-	-	-
STC15F104W	5.5-3.8	4K	128	-	-	2	-	有	5	-	1	1K	有	有	8 级	有	是	有	是	是	¥1.20	¥1.40	¥1.60
IAP15F105W	5.5-3.8	5K	128	-	-	2	-	有	5	-	1	IAP	有	有	8 级	有	是	有	是	是	¥1.40	¥1.60	-
																					用户可将用户程序区的程序 FLASH 当 EEPROM 使用		
IRC15F107W 默认使用内部 24MHz 时钟	5.5-3.8	7K	128	-	-	2	-	有	5	-	1	IAP	有	有	固定	有	是	无	否	否	¥1.40	¥1.6	
																					用户可将用户程序区的程序 FLASH 当 EEPROM 使用		
STC15L104W 系列单片机选型价格一览表， 此系列建议用 STC15W104 系列取代（STC15W104 系列大批量现货供应中）																							
STC15L100W	3.6-2.4	0.5K	128	-	-	2	-	有	5	-	1	-	有	有	8 级	有	是	有	是	是	¥1.20	¥1.40	¥1.60
STC15L101W	3.6-2.4	1K	128	-	-	2	-	有	5	-	1	4K	有	有	8 级	有	是	有	是	是	¥1.20	¥1.40	¥1.60
STC15L102W	3.6-2.4	2K	128	-	-	2	-	有	5	-	1	3K	有	有	8 级	有	是	有	是	是	¥1.20	¥1.40	¥1.60
STC15L104W	3.6-2.4	4K	128	-	-	2	-	有	5	-	1	1K	有	有	8 级	有	是	有	是	是	¥1.20	¥1.40	¥1.60
IAP15L105W	3.6-2.4	5K	128	-	-	2	-	有	5	-	1	IAP	有	有	8 级	有	是	有	是	是	¥1.20	¥1.40	-
																					用户可将用户程序区的程序 FLASH 当 EEPROM 使用		

STC15F104W 系列单片机只有定时器 0 和定时器 2，无定时器 1

提供客制化 IC 服务

建议用户选用 S0P8 封装，但 DIP8 封装以及新生产 DFN8 封装仍正常供货。

STC15L104W 系列单片机建议用 STC15W104 系列取代

STC15F/L104W 系列只有固件版本为 Ver7.2（成功烧录程序时在 AIapp-ISP 软件界面右下角查询）及其以上的单片机才支持“可设下次更新程序需口令”功能

程序加密后传输：程序拥有者产品出厂时将源程序和加密钥匙一起烧录 MCU 中，以后需要升级软件时，就可将程序加密后再用“发布项目程序”功能，生成一个用户自己界面的只有一个升级按钮的简单易用的升级软件，给最终使用者自己升级，而拦截不到您的原始程序。

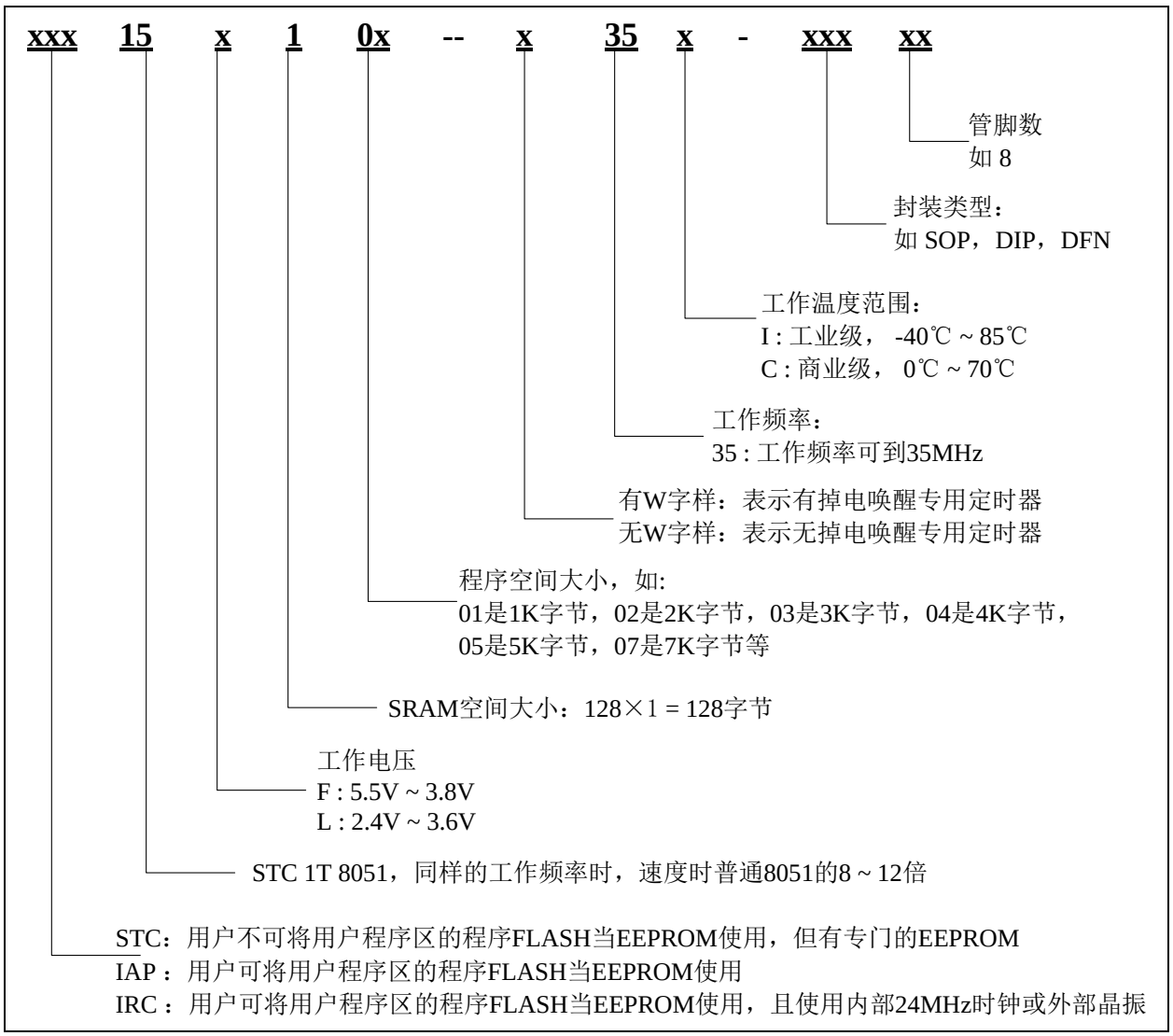
因为程序区的最后 7 个字节单元被强制性的放入全球唯一 ID 号的内容，所以用户实际可以使用的程序空间大小要比选型表中的大小少 7 个字节。

特别声明：以 15L 开头的芯片如需进入“掉电模式”，进入“掉电模式”前必须启动掉电唤醒定时器<3uA>，不超过 1 秒要唤醒一次，以 15F 和 15W 开头的芯片则不需要。

【总结】:

- STC15F/L104W 系列单片机（含 IRC15F107W 型号单片机）有：
  - ✓ 两个 16 位重装载定时器/计数器（这两个定时器/计数器分别是：定时器/计数器 0 和定时器/计数器 2）；
  - ✓ 有 5 个外部中断 INT0/INT1/INT2/INT3/INT4；
  - ✓ 有掉电唤醒专用定时器；
  - ✓ 有 1 个数据指针 DPTR。
  - ✓ 表中“-”表示该型号的单片机无相应的功能。
  - ✓ STC15F/L104W 系列单片机（含 IRC15F107W 型号单片机）无串行口、无 SPI、无 A/D 转换、无 CCP/PWM/PCA、无外部数据总线等功能。

3.1.4 STC15F104W 系列单片机命名规则



如何识别芯片版本号：如需知道芯片版本号，请查阅芯片表面印刷字中最下面一行的最后一个字母（如 D），该字母代表芯片版本号（如 D 版）

命名举例:

1) STC15F101W-35I-SOP8 表示：

用户不可以将用户程序区的程序 FLASH 当 EEPROM 使用，但有专门的 EEPROM。该单片机为 1T 8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 5.5V ~ 3.8V，SRAM 空间大小为 128 字节，程序空间大小为 1K，有掉电唤醒专用定时器，工作频率可到 35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 SOP 贴片封装，管脚数为 8。

2) STC15L101W-35I-SOP8 表示：

用户不可以将用户程序区的程序 FLASH 当 EEPROM 使用，但有专门的 EEPROM。该单片机为 1T 8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 2.4V ~ 3.6V，SRAM 空间大小为 128 字节，程序空间大小为 1K，有掉电唤醒专用定时器，工作频率可到 35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 SOP 贴片封装，管脚数为 8。

3) STC15F104W-35I-SOP8 表示：

用户不可以将用户程序区的程序 FLASH 当 EEPROM 使用，但有专门的 EEPROM。该单片机为 1T 8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 5.5V ~ 3.8V，SRAM 空间大小为 128 字节，程序空间大小为 4K，有掉电唤醒专用定时器，工作频率可到 35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 SOP 贴片封装，管脚数为 8。

4) STC15L104W-35I-SOP8 表示：

用户不可以将用户程序区的程序 FLASH 当 EEPROM 使用，但有专门的 EEPROM。该单片机为 1T 8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 2.4V ~ 3.6V，SRAM 空间大小为 128 字节，程序空间大小为 4K，有掉电唤醒专用定时器，工作频率可到 35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 SOP 贴片封装，管脚数为 8。

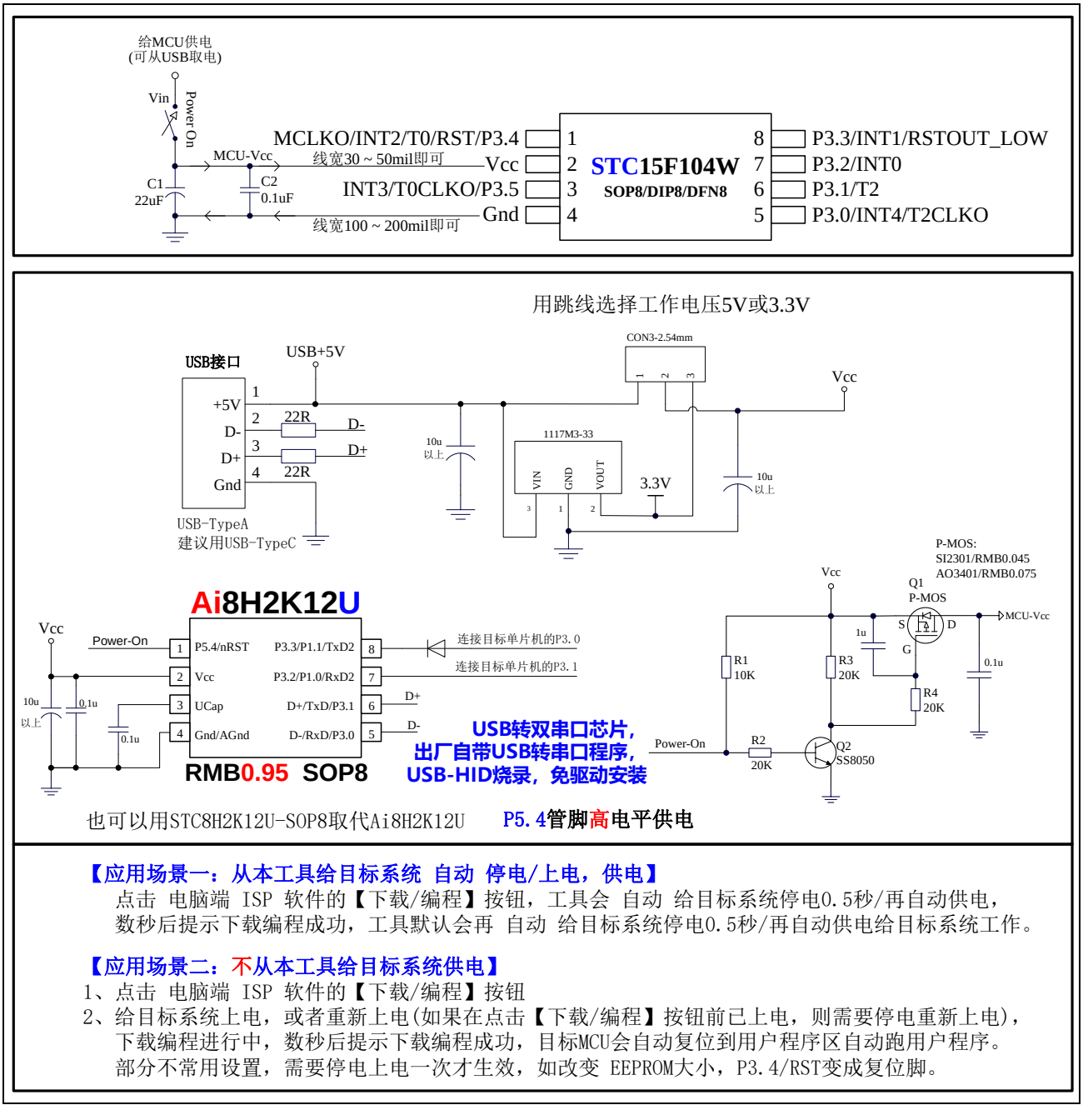
5) IAP15F105W-35I-SOP8 表示：

用户可以将用户程序区的程序 FLASH 当 EEPROM 使用。该单片机为 IT 8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 5.5V ~ 3.8V，SRAM 空间大小为 128 字节，程序空间大小为 5K，有掉电唤醒专用定时器，工作频率可到 35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 SOP 贴片封装，管脚数为 8。

6) IAP15L105W-35I-SOP8 表示：

用户可以将用户程序区的程序 FLASH 当 EEPROM 使用。该单片机为 1T 8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 2.4V ~ 3.6V，SRAM 空间大小为 128 字节，程序空间大小为 5K，有掉电唤醒专用定时器，工作频率可到 35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 SOP 贴片封装，管脚数为 8。

3.1.5 管脚图及下载线路图



【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，P3.4/RST变成复位脚。

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	Tx2_Rx2	CLKS2	CLKS1	CLKS0	00x0, x000

MCKO_S1	MCKO_S0	主时钟可外分频输出控制位 (主时钟可对外输出内部 R/C 时钟，也可对外输出外部输入的时钟或外部晶体振荡产生的时钟)
0	0	主时钟不对外输出时钟
0	1	主时钟对外输出时钟，但时钟频率不被分频，输出时钟频率 = MCLK / 1
1	0	主时钟对外输出时钟，但时钟频率被 2 分频，输出时钟频率 = MCLK / 2
1	1	主时钟对外输出时钟，但时钟频率被 4 分频，输出时钟频率 = MCLK / 4

STC15F104W 系列单片机不支持外接外部晶体，其主时钟对外输出管脚 P3.4/MCLKO 只可以对外分频输出内部 R/C 时钟，MCLK 是指主时钟频率。

STC15 系列 8-pin 单片机（如 STC15F104W 系列）在 MCLKO/P3.4 口对外输出时钟，STC15 系列 16-pin 及其以上单片机（如 STC15W4K32S4 系列）均在 MCLKO/P5.4 口对外输出时钟。

Tx_Rx：P3.1 口的对外输出实时反映 P3.0 口的外部输入状态的选择位

0：P3.1 口的对外输出不反映 P3.0 口的外部输入状态

1：将 P3.0 管脚输入的电平状态实时输出在 P3.1 外部管脚上，即 P3.1 口的对外输出实时反映 P3.0 口的外部输入状态。当 P3.0 外部输入为 1 时，P3.1 口的对外输出就为 1；当 P3.0 外部输入为 0 时，P3.1 口的对外输出也就为 0。

CLKS2	CLKS1	CLKS0	系统时钟选择控制位 (系统时钟是指对主时钟进行分频后供给 CPU、定时器的实际工作时钟)
0	0	0	主时钟频率/1,不分频
0	0	1	主时钟频率/2
0	1	0	主时钟频率/4
0	1	1	主时钟频率/8
1	0	0	主时钟频率/16
1	0	1	主时钟频率/32
1	1	0	主时钟频率/64
1	1	1	主时钟频率/128

3.1.6 管脚说明

管脚	管脚编号 (封装 SOP8/DIP8/DFN8)	说明	
P3.0	5	P3.0	标准 I/O 口 PORT3[0]
		INT4	外部中断 4，只能下降沿中断 INT4 支持掉电唤醒
		T2CLKO	T2 的时钟输出 可通过设置 INT_CLKO[2]位/T2CLKO 将该管脚配置为 T2CLKO
P3.1	6	P3.1	标准 I/O 口 PORT3[1]
		T2	定时器/计数器 2 的外部输入
P3.2	7	P3.2	标准 I/O 口 PORT3[2]
		INT0	外部中断 0，既可上升沿中断也可下降沿中断。 如果 IT0(TCON.0)被置为 1，INT0 管脚仅为下降沿中断。 如果 IT0(TCON.0)被清 0，INT0 管脚既支持上升沿中断也支持下降沿中断。 INT0 支持掉电唤醒。
P3.3	8	P3.3	标准 I/O 口 PORT3[3]
		INT1	外部中断 1，既可上升沿中断也可下降沿中断。 如果 IT1(TCON.2)被置为 1，INT1 管脚仅为下降沿中断。 如果 IT1(TCON.2)被清 0，INT1 管脚既支持上升沿中断也支持下降沿中断。 INT1 支持掉电唤醒。
		RSTOUT_LOW	上电后，输出低电平，在复位期间也是输出低电平，用户可用软件将其设置为高电平或低电平，如果要读外部状态，可将该口先置高后再读。
P3.4	1	P3.4	标准 I/O 口 PORT3[4]
		RST	复位脚，高电平复位
		T0	定时器/计数器 0 的外部输入
		INT2	外部中断 2，只能下降沿中断。 INT2 支持掉电唤醒
		MCLKO	主时钟输出 输出的频率可为 MCLK/1，MCLK/2，MCLK/4 (MCLK 为主时钟频率)。 此系列的主时钟外部输出管脚 P3.4/MCLKO 只可以对外输出内部 R/C 时钟。
P3.5	3	P3.5	标准 I/O 口 PORT3[5]
		T0CLKO	定时器/计数器 0 的时钟输出 可通过设置 INT_CLKO[0]位/T0CLKO 将该管脚配置为 T0CLKO，也可对 T0 脚的外部时钟输入进行分频输出
		INT3	外部中断 3，只能下降沿中断。 INT3 支持掉电唤醒
Vcc	2	电源正极	
Gnd	4	电源负极，接地	

3.1.7 USB-Link1D 工具自动停电/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】

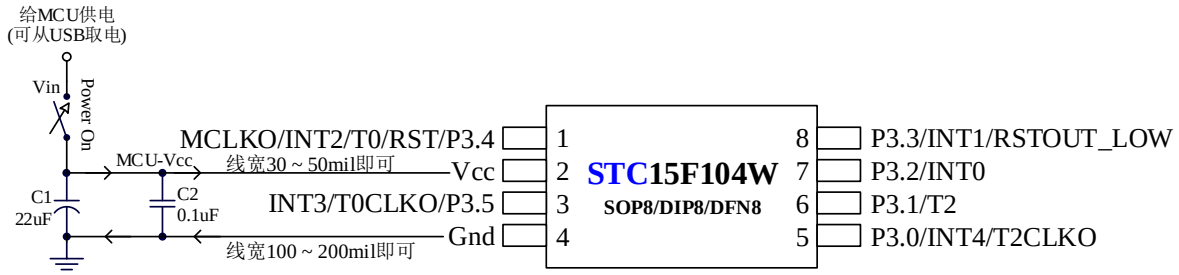
点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【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，P3.4/nRST变成复位脚



3.1.8 【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】

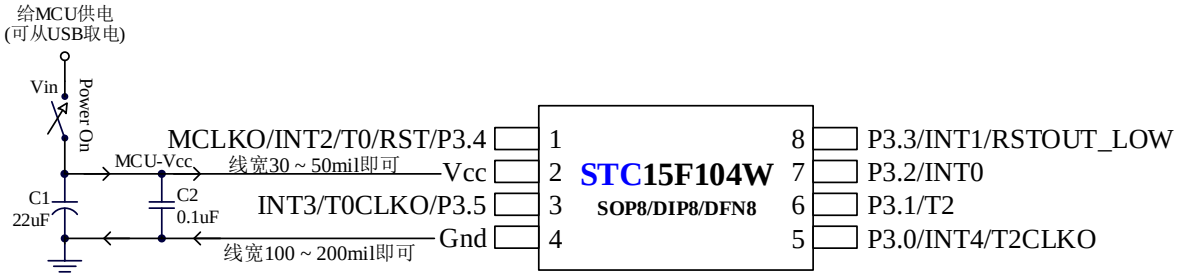
点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【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，P3.4/nRST变成复位脚



3.2 STC15W104 系列

STC15W104 系列新一代 1T 8051 单片机，宽电压，高可靠，超低价

不需外部晶振的单片机，不需外部复位的单片机
全球第一款真正意义上的单片机 ISP/IAP 技术全球领导

STC15W104 系列主要性能：

- ✓ 高速：1 个时钟/机器周期，增强型 8051 内核，速度比普通 8051 快 6 ~ 12 倍
- ✓ 速度也比 STC 早期的 1T 系列单片机（如 STC12/11/10 系列）的速度快 20%
- ✓ 宽电压：5.5 ~ 2.5V
- ✓ 低功耗设计：低速模式，空闲模式，掉电模式（可由外部中断或内部掉电唤醒定时器唤醒）
- ✓ 不需外部复位的单片机，ISP 编程时 16 级复位门槛电压可选，内置高可靠复位电路
- ✓ 不需外部晶振的单片机，ISP 编程时内部时钟从 5MHZ ~ 35MHz 可设（相当于 8051：60 ~ 420MHz）内部高精度 R/C 时钟（±0.3%），+1%温漂（-40℃ ~ +85℃），常温下温漂+0.6%（-20℃ ~ +65℃）
- ✓ 支持掉电唤醒的资源有：INT0/INT1（上升沿/下降沿中断均可），IINT2/INT3/INT4（下降沿中断）；T0/T2 管脚；内部掉电唤醒专用定时器
- ✓ 1K/2K/3K/4K/5K/7K 字节片内 Flash 程序存储器，擦写次数 10 万次以上
- ✓ 128 字节片内 RAM 数据存储器
- ✓ 片内 EEPROM 功能，擦写次数 10 万次以上
- ✓ ISP/IAP，在系统可编程/在应用可编程，无需编程器/仿真器
- ✓ 2 个 16 位可重装载定时器 T0/T2，并可实现时钟输出功能，另外管脚 MCLKO 可将内部主时钟对外分频输出（÷1 或÷2 或÷4）
- ✓ 可编程时钟输出功能（对内部系统时钟或外部管脚的时钟输入进行时钟分频输出）；
  - ➢ T0 在 P3.5 输出时钟；
  - ➢ T2 在 P3.0 输出时钟；
  - ➢ 内部主时钟在 P3.4/MCLKO 对外输出时钟（STC15 系列 8-pin 以上单片机的主时钟在 P5.4/MCLKO 对外输出时钟）
- ✓ 硬件看门狗（WDT）
- ✓ 串口功能可由[P3.0/INT4，P3.1]结合定时器实现
- ✓ 先进的指令集结构，兼容普通 8051 指令集，有硬件乘法/除法指令
- ✓ 通用 I/O 口（8 个），复位后为：准双向口/弱上拉（8051 传统 I/O 口）

  可设置四种模式：准双向口/弱上拉，强推挽/强上拉，仅为输入/高阻，开漏

  每个 I/O 口驱动能力均可达到 20mA，但整个芯片最大不要超过 90mA
- ✓ 如果 I/O 口不够用可以用 3 根普通 I/O 端口外接 74HC595（¥0.15 元）来扩展 I/O 口，并可多芯片级联扩展几十个 I/O 口，还可用 AD 作按键扫描来节省 I/O 口

选择 STC15W104 系列单片机理由：

★ 不需外部晶振和外部复位，还可对外输出时钟和低电平复位信号

★ 不需外部晶振的单片机，内部集成高精度 R/C 时钟（±0.3%），±1%温漂（-40℃ ～ +85℃），常温下温漂±0.6%（-20℃～+65℃）

★ 不需外部复位的单片机，内部集成高可靠复位电路，ISP 编程时 16 级复位门槛电压可选，当然也可以继续用外部复位电路

★ 无法解密，STC 第九代加密技术，现悬赏 20 万元人民币请专家帮忙查找加密有无漏洞

★ 超强抗干扰：

- 高抗静电（ESD 保护）整机轻松过 2 万伏静电测试
- 轻松过 4kV 快速脉冲干扰（EFT 测试）
- 宽电压，不怕电源抖动
- 宽温度范围，-40℃～+85℃

★ 大幅降低 EMI，内部可配置时钟，1 个时钟/机器周期，可用低频时钟

----出口欧美的有力保证

★ 超低功耗：

- 掉电模式：外部中断唤醒功耗<0.2uA
- 空闲模式：典型功耗<1mA，
- 正常工作模式：4mA -6mA
- 掉电模式可由外部中断唤醒，适用于电池供电系统，如水表、气表、便携设备等

★ 在系统可仿真，在系统可编程，无需专用编程器，无需专用仿真器，可远程升级

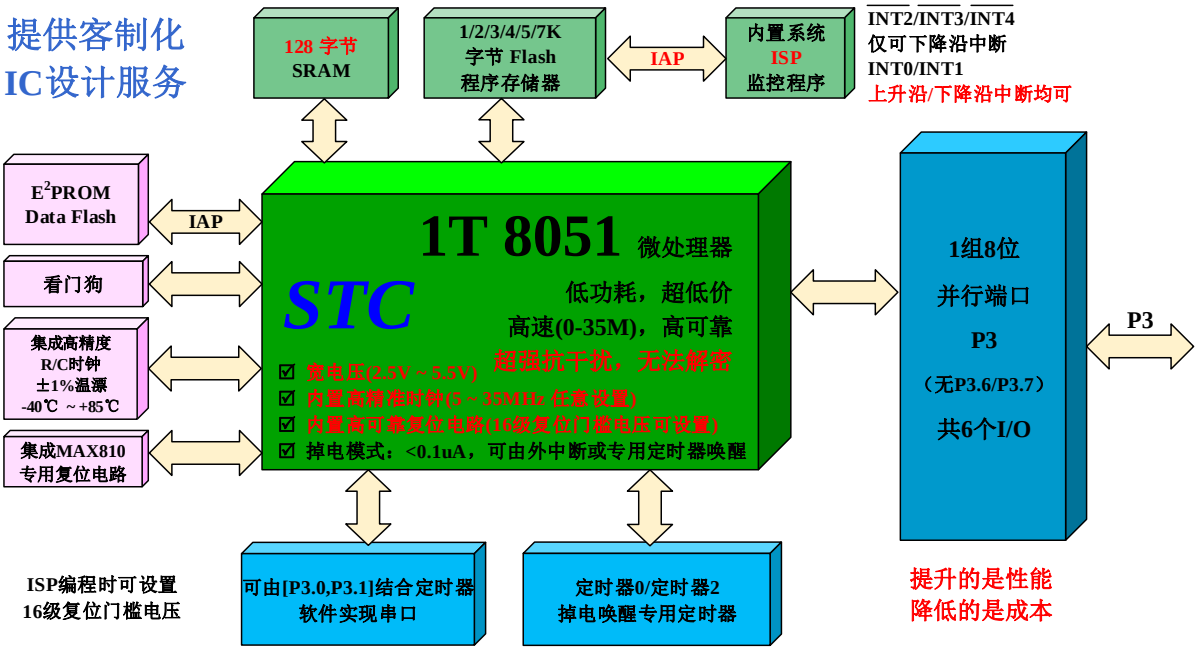
★ 可送 USB 型联机/脱机下载烧录工具 STC-U8（人民币 100 元），1 万片/人/天，有自动烧录机接口

3.2.1 STC15W104 系列简介

STC15W104 系列单片机是 STC 生产的单时钟/机器周期（1T）的单片机，是高速/高可靠/宽电压/低功耗/超强抗干扰的新一代 8051 单片机，采用 STC 第九代加密技术，无法解密，指令代码完全兼容传统 8051，但速度快 8-12 倍。内部集成高精度 R/C 时钟（±0.3%），±1%温漂（-40℃～+85℃），常温下温漂±0.6%（-20℃～+65℃）。ISP 编程时 5MHz～35MHz 宽范围可设置，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 16 级复位门槛电压可选）。

在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可

现 STC15 系列单片机采用 STC-Y5 超高速 CPU 内核，在相同的时钟频率下，速度又比 STC 早期的 1T 系列单片机（如 STC12 系列/STC11 系列/STC10 系列）的速度快 20%。



1. 增强型 8051 CPU，1T，单时钟/机器周期，速度比普通 8051 快 8-12 倍
2. 工作电压：2.5V -- 5.5V
3. 1K/2K/3K/4K/5K/7K 字节片内 Flash 程序存储器，可擦写次数 10 万次以上
4. 片内 128 字节的 SRAM
5. 有片内 EEPROM 功能，擦写次数 10 万次以上
6. ISP/IAP，在系统可编程/在应用可编程，无需编程器/仿真器
7. 内部高可靠复位，ISP 编程时 16 级复位门槛电压可选，可彻底省掉外部复位电路
8. 工作频率范围：5MHz～35MHz，相当于普通 8051 的 60MHz～420MHz
9. 内部高精度 R/C 时钟（±0.3%），±1%温漂（-40℃～+85℃），常温下温漂±0.6%（-20℃～+65℃），ISP 编程时内部时钟从 5MHz～35MHz 可设（5.5296MHz/11.0592MHz/22.1184MHz/33.1776MHz）
10. 不需外部晶振和外部复位，还可对外输出时钟和低电平复位信号
11. 串口功能可由[P3.0/INT4，P3.1]结合定时器实现
12. 支持程序加密后传输，防拦截

13. 支持 RS485 下载
14. 低功耗设计：低速模式，空闲模式，掉电模式/停机模式.
15. 可将掉电模式/停机模式唤醒的定时器：有内部低功耗掉电唤醒专用定时器。
16. 可将掉电模式/停机模式唤醒的资源有：
    - INT0/P3.2，INT1/P3.3 （INT0/INT1 上升沿下降沿中断均可），INT2/P3.4，INT3/P3.5，INT4/P3.0（INT2/INT3/INT4 仅可下降沿中断）；
    - 管脚 T0/T2（下降沿，不产生中断，前提是在进入掉电模式/停机模式前相应的定时器中断已经被允许）；
    - 内部低功耗掉电唤醒专用定时器。
17. 共 2 个定时器/计数器--T0（兼容普通 8051 的定时器）/T2，并均可实现可编程时钟输出，另外管脚 MCLKO 可将内部主时钟对外分频输出（÷1 或÷2 或÷4）
18. 可编程时钟输出功能（对内部系统时钟或对外部管脚的时钟输入进行时钟分频输出）：
    - 由于 STC15 系列 5V 单片机 I/O 口的对外输出速度最快不超过 13.5MHz，所以 5V 单片机的对外可编程时钟输出速度最快也不超过 13.5MHz；
    - 而 3.3V 单片机 I/O 口的对外输出速度最快不超过 8MHz，故 3.3V 单片机的对外可编程时钟输出速度最快也不超过 8MHz

    1) T0 在 P3.5/T0CLKO 进行可编程输出时钟（对内部系统时钟或对外部管脚 T0/P3.4 的时钟输入进行可编程时钟分频输出）；
    2) T2 在 P3.0/T2CLKO 进行可编程输出时钟（对内部系统时钟或对外部管脚 T2/P3.1 的时钟输入进行可编程时钟分频输出）；

    以上 2 个定时器/计数器均可 1 ~ 65536 级分频输出。

    3) 主时钟在 P3.4/MCLKO 对外输出时钟，并可如下分频 MCLK/1，MCLK/2，MCLK/4

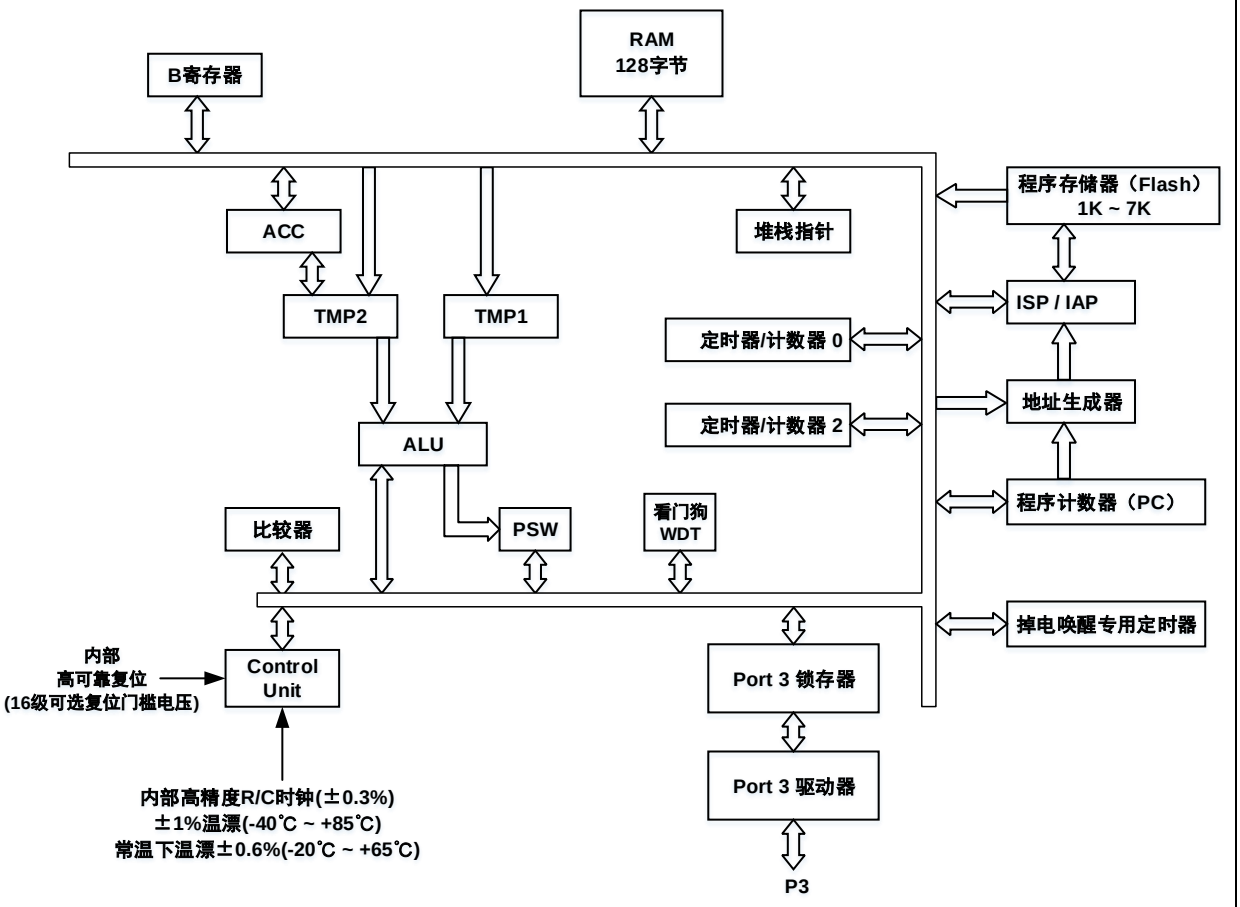
    STC15W104 系列单片机不支持外接外部晶体，其主时钟对外输出管脚 P3.4/MCLKO 只可以对外输出内部 R/C 时钟。MCLK 是指主时钟频率，MCLKO 是指主时钟输出。

    STC15 系列 8-pin 单片机（如 STC15W104 系列）在 MCLKO/P3.4 口对外输出时钟，STC15 系列 16-pin 及其以上单片机均在 MCLKO/P5.4 口对外输出时钟，且 STC15W 系列 20-pin 及其以上单片机除可在 MCLKO/P5.4 口对外输出时钟外，还可在 MCLKO_2/P1.6 口对外输出时钟。
19. 硬件看门狗（WDT）
20. 先进的指令集结构，兼容普通 8051 指令集，有硬件乘法/除法指令
21. 共 6 个通用 I/O 口，复位后为：准双向口上拉（普通 8051 传统 I/O 口）

    可设置成四种模式：准双向口/弱上拉，强推挽/强上拉，仅为输入/高阻，开漏
    每个 I/O 口驱动能力均可达到 20mA，但整个芯片电流最大不要超过 90mA
    如果 I/O 口不够用，可外接 74HC595（参考价 0.15 元）来扩展 I/O 口，并可多芯片级联扩展几十个 I/O 口。
22. 封装：SOP-8，DIP-8，DFN-8（不推荐）.
23. 全部 175℃八小时高温烘烤，高品质制造保证
24. 开发环境：在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可

3.2.2 内部结构图

STC15W104 系列单片机的内部结构框图如下图所示。STC15W104 系列单片机中包含中央处理器(CPU)、程序存储器(Flash)、数据存储器(SRAM)、定时器/计数器、掉电唤醒专用定时器、I/O 口、看门狗、片内高精度 R/C 时钟及高可靠复位等模块。



STC15W104系列内部结构框图

3.2.3 STC15W104 系列单片机选型价格一览表

型号	工作电压(V)	Flash 程序存储器(byte)	SRAM 字节	EEPROM	串行口	SPI	定时器计数器 T0/T2 外部管脚也能掉电唤醒	CCP PCA PWM D/A	掉电唤醒定时器	标准外部中断支持掉电唤醒	A/D 8 路	看门狗	内部低压检测中断	内部复位(可选复位门槛电压)	内部高精准时钟	可对外输出时钟及复位	程序加密后传输(防拦截)	可设下次更新程序需口令	支持 RS485 下载	封装 8-Pin SOP-8 / DIP-8 / DFN-8 (6 个 I/O 口) 价格(RMB ¥)		
																				SOP-8	DIP-8	DFN-8
STC15W104 系列单片机选型价格一览表，此系列大批量现货供应中 串行口功能可由[P3.0/INT4, P3.1]结合定时器实现																						
STC15W100	5.5-2.5	512	128	无	无	无	2	无	有	5 个	无	有	有	16 级	有	是	有	是	是	¥1.60	¥1.80	¥1.90
STC15W101	5.5-2.5	1K	128	4K	无	无	2	无	有	5 个	无	有	有	16 级	有	是	有	是	是	¥1.60	¥1.80	¥1.90
STC15W102	5.5-2.5	2K	128	3K	无	无	2	无	有	5 个	无	有	有	16 级	有	是	有	是	是	¥1.60	¥1.80	¥1.90
STC15W104	5.5-2.5	4K	128	1K	无	无	2	无	有	5 个	无	有	有	16 级	有	是	有	是	是	¥1.60	¥1.80	¥1.90
IAP15W105	5.5-2.5	5K	128	IAP	无	无	2	无	有	5 个	无	有	有	16 级	有	是	有	是	是	¥1.60	¥1.80	-
																				用户可将用户程序区的程序 FLASH 当 EEPROM 使用		
IRC15W107 默认使用内部 24MHz 时钟	5.5-2.5	7K	128	IAP	无	无	2	无	有	5 个	无	有	有	固定	有	是	有	否	否	¥1.60	¥1.80	-
																				用户可将用户程序区的程序 FLASH 当 EEPROM 使用		
STC15F104W 系列单片机选型价格一览表，该系列低压型号(工作电压 2.4V-3.6V) 建议用 STC15W104 系列代替串行口功能可由[P3.0/INT4, P3.1]结合定时器实现 此系列大批量现货供应中																						
STC15F100W	5.5-3.8	512	128	无	无	无	2	无	有	5 个	无	有	有	8 级	有	是	有		是	¥1.60	¥1.80	¥1.90
STC15F101W	5.5-3.8	1K	128	4K	无	无	2	无	有	5 个	无	有	有	8 级	有	是	有		是	¥1.60	¥1.80	¥1.90
STC15F104W	5.5-3.8	4K	128	1K	无	无	2	无	有	5 个	无	有	有	8 级	有	是	有		是	¥1.60	¥1.80	¥1.90
IRC15F107W 默认使用内部 24MHz 时钟	5.5-3.8	7K	128	IAP	无	无	2	无	有	5 个	无	有	有	固定	有	是	有		否	¥1.60	¥1.80	-
																				用户可将用户程序区的程序 FLASH 当 EEPROM 使用		

STC15W104 系列单片机只有定时器 0 和定时器 2，无定时器 1

提供客制化 IC 服务

建议用户选用 SOP8 封装，但 DIP8 封装以及新生产 DFN8 封装仍正常供货。

我们直销，所以低价，以上单价为 10K/M 起定量，量小每片需加 0.1 元，以上价格运费由客户承担，零售 10 片起，如对价格不满，可来电要求降价。

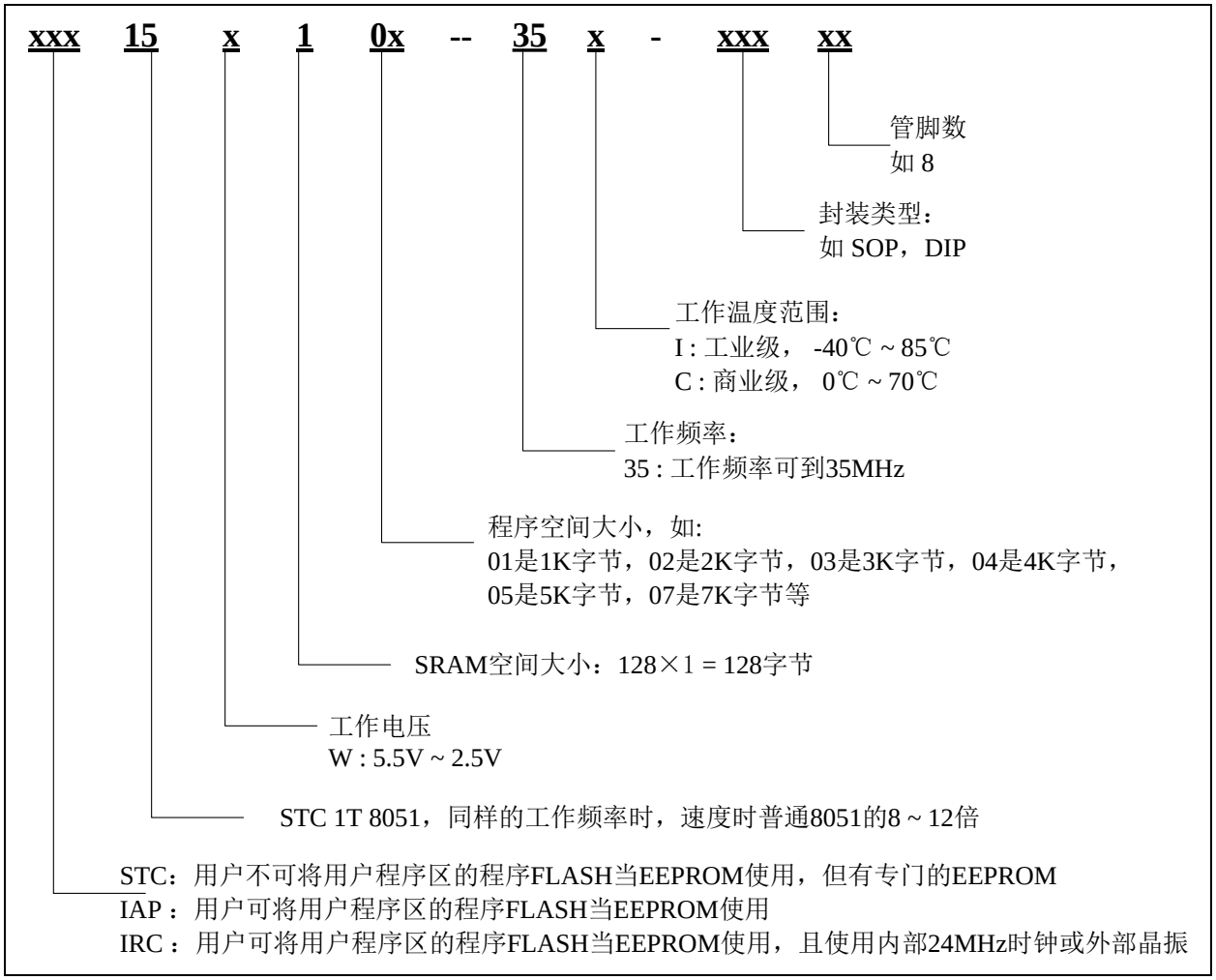
程序加密后传输：程序拥有者产品出厂时将源程序和加密钥匙一起烧录 MCU 中，以后需要升级软件时，就可将程序加密后再用“发布项目程序“功能，生成一个用户自己界面的只有一个升级按钮的简单易用的升级软件，给最终使用者自己升级，而拦截不到您的原始程序。

因为程序区的最后 7 个字节单元被强制性的放入全球唯一 1D 号的内容，所以用户实际可以使用的程序空间大小要比选型表中的大小少 7 个字节。

【总结】：STC15W104 系列单片机(含 IRC15W107 型号单片机)有：

- ✓ 两个 16 位重装载定时器/计数器(这两个定时器/计数器分别是：定时器/计数器 0 和定时器/计数器 2)
- ✓ 有 5 个外部中断 INT0/INT1/INT2/INT3/INT4
- ✓ 有掉电唤醒专用定时器
- ✓ 有 1 个数据指针 DPTR。
- ✓ 表中“-”表示该型号的单片机无相应的功能。
- ✓ STC15F/L101W 系列单片机(含 IRC15F107W 型号单片机)无串行口、无比较器、无 SPI、无 A/D 转换、无 CCP/PWM/PCA、无外部数据总线等功能。

3.2.4 STC15W104 系列单片机命名规则



如何识别芯片版本号：如需知道芯片版本号，请查阅芯片表面印刷字中最下面一行的最后一个字母(如 A)，该字母代表芯片版本号(如 A 版)

命名举例:

1) STC15W101-35I-SOP8 表示：

用户不可以将用户程序区的程序 FLASH 当 EEPROM 使用，但有专门的 EEPROM。该单片机为 1T 8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 5.5V ~ 2.5V，SRAM 空间大小为 128 字节，程序空间大小为 1K，有掉电唤醒专用定时器，工作频率可到 35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 SOP 贴片封装，管脚数为 8。

2) STC15W104-35I-SOP8 表示：

用户不可以将用户程序区的程序 FLASH 当 EEPROM 使用，但有专门的 EEPROM。该单片机为 IT 8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 5.5V ~ 2.5V，SRAM 空间大小为 128 字节，程序空间大小为 4K，有掉电唤醒专用定时器，工作频率可到 35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 SOP 贴片封装，管脚数为 8。

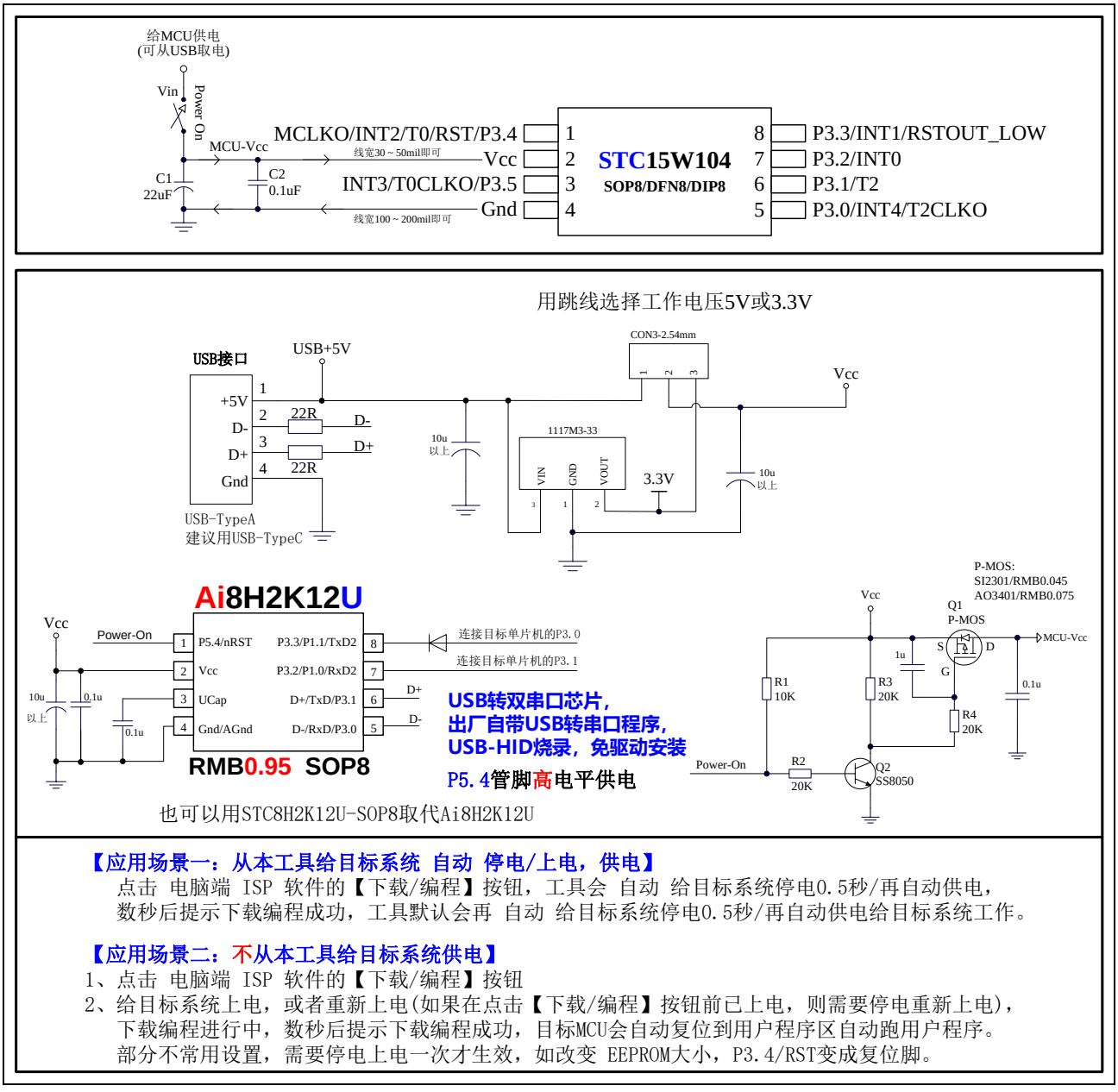
3) IAP15W105-35I-SOP8 表示：

用户可以将用户程序区的程序 FLASH 当 EEPROM 使用。该单片机为 IT 8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 5.5V ~ 2.5V，SRAM 空间大小为 128 字节，程序空间大小为 5K，有掉电唤醒专用定时器，工作频率可到 35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 SOP 贴片封装，管脚数为 8。

4) IRC15W107-35I-DIP8 表示：

用户可将用户程序区的程序 FLASH 当 EEPROM 使用，且默认使用内部 24MHz 时钟，该单片机为 IT 8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 5.5V ~ 2.5V，SRAM 空间大小为 128 字节，程序空间大小为 7K，有掉电唤醒专用定时器，工作频率可到 35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 DIP 贴片封装，管脚数为 8。

3.2.5 管脚图及下载线路图，最小系统（SOP8/DFN8/DIP8）



【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，P3.4/RST变成复位脚。

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	Tx2_Rx2	CLKS2	CLKS1	CLKS0	00x0, x000

MCKO_S1	MCKO_S0	主时钟可外分频输出控制位 (主时钟可对外输出内部 R/C 时钟，也可对外输出外部输入的时钟或外部晶体振荡产生的时钟)
0	0	主时钟不对外输出时钟
0	1	主时钟对外输出时钟，但时钟频率不被分频，输出时钟频率 = MCLK / 1
1	0	主时钟对外输出时钟，但时钟频率被 2 分频，输出时钟频率 = MCLK / 2
1	1	主时钟对外输出时钟，但时钟频率被 4 分频，输出时钟频率 = MCLK / 4

STC15W104 系列单片机不支持外接外部晶体，其主时钟对外输出管脚 P3.4/MCLKO 只可以对外分频输出内部 R/C 时钟，MCLK 是指主时钟频率。

STC15 系列 8-pin 单片机（如 STC15W104 系列）在 MCLKO/P3.4 口对外输出时钟，STC15 系列 16-pin 及其以上单片机（如 STC15W4K32S4 系列）均在 MCLKO/P5.4 口对外输出时钟。

Tx_Rx：P3.1 口的对外输出实时反映 P3.0 口的外部输入状态的选择位

0：P3.1 口的对外输出不反映 P3.0 口的外部输入状态

1：将 P3.0 管脚输入的电平状态实时输出在 P3.1 外部管脚上，即 P3.1 口的对外输出实时反映 P3.0 口的外部输入状态。当 P3.0 外部输入为 1 时，P3.1 口的对外输出就为 1；当 P3.0 外部输入为 0 时，P3.1 口的对外输出也就为 0。

CLKS2	CLKS1	CLKS0	系统时钟选择控制位 (系统时钟是指对主时钟进行分频后供给 CPU、定时器的实际工作时钟)
0	0	0	主时钟频率/1,不分频
0	0	1	主时钟频率/2
0	1	0	主时钟频率/4
0	1	1	主时钟频率/8
1	0	0	主时钟频率/16
1	0	1	主时钟频率/32
1	1	0	主时钟频率/64
1	1	1	主时钟频率/128

3.2.6 管脚说明

管脚	管脚编号 (封装 SOP8/DIP8)	说明	
P3.0	5	P3.0	标准 I/O 口 PORT3[0]
		INT4	外部中断 4，只能下降沿中断 INT4 支持掉电唤醒
		T2CLKO	T2 的时钟输出 可通过设置 INT_CLKO[2]位/T2CLKO 将该管脚配置为 T2CLKO
P3.1	6	P3.1	标准 I/O 口 PORT3[1]
		T2	定时器/计数器 2 的外部输入
P3.2	7	P3.2	标准 I/O 口 PORT3[2]
		INT0	外部中断 0，既可上升沿中断也可下降沿中断。 如果 IT0(TCON.0)被置为 1，INT0 管脚仅为下降沿中断。 如果 IT0(TCON.0)被清 0，INT0 管脚既支持上升沿中断也支持下降沿中断。 INT0 支持掉电唤醒。
P3.3	8	P3.3	标准 I/O 口 PORT3[3]
		INT1	外部中断 1，既可上升沿中断也可下降沿中断。 如果 IT1(TCON.2)被置为 1，INT1 管脚仅为下降沿中断。 如果 IT1(TCON.2)被清 0，INT1 管脚既支持上升沿中断也支持下降沿中断。 INT1 支持掉电唤醒。
		RSTOUT_LOW	上电后，输出低电平，在复位期间也是输出低电平，用户可用软件将其设置为高电平或低电平，如果要读外部状态，可将该口先置高后再读。
P3.4	1	P3.4	标准 I/O 口 PORT3[4]
		RST	复位脚，高电平复位
		T0	定时器/计数器 0 的外部输入
		INT2	外部中断 2，只能下降沿中断。 INT2 支持掉电唤醒
		MCLKO	主时钟输出 输出的频率可为 MCLK/1，MCLK/2，MCLK/4 (MCLK 为主时钟频率)。 此系列的主时钟外部输出管脚 P3.4/MCLKO 只可以对外输出内部 R/C 时钟。
P3.5	3	P3.5	标准 I/O 口 PORT3[5]
		T0CLKO	定时器/计数器 0 的时钟输出 可通过设置 INT_CLKO[0]位/T0CLKO 将该管脚配置为 T0CLKO，也可对 T0 脚的外部时钟输入进行分频输出
		INT3	外部中断 3，只能下降沿中断。 INT3 支持掉电唤醒
Vcc	2	电源正极	
Gnd	4	电源负极，接地	

3.2.7 USB-Link1D 工具自动停电/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

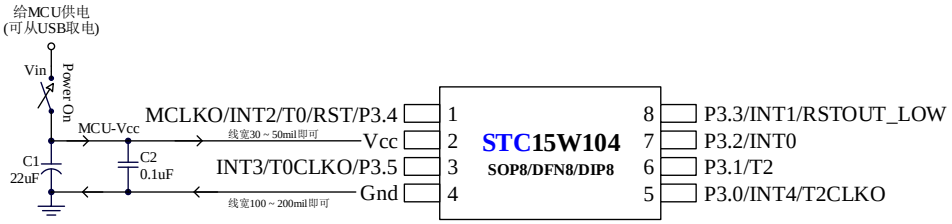
【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

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，P3.4/nRST变成复位脚



3.2.8 【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

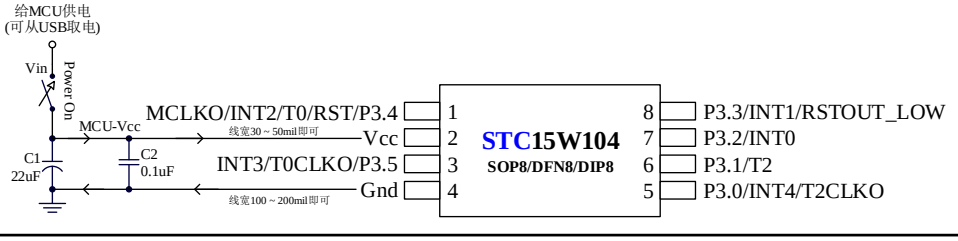
【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

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，P3.4/nRST变成复位脚



3.3 STC15W204S 系列

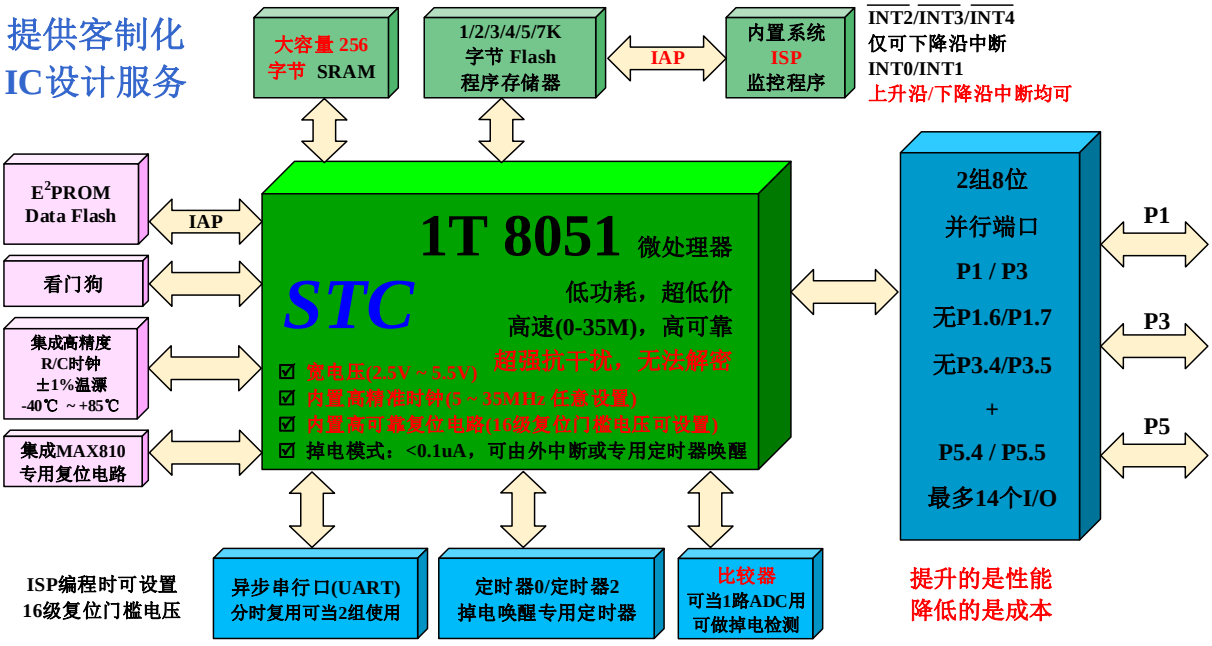
STC15W204S 系列 1T 8051 单片机，宽电压，超高速串行口，比较器
不需外部晶振的单片机，不需外部复位的单片机

STC15W204S 系列是目前全球抗干扰最强的 Flash 型单片机

3.3.1 STC15W204S 系列简介

STC15W204S 系列单片机是 STC 生产的单时钟/机器周期（1T）的单片机，是宽电压/高速/高可靠/低功耗/超强抗干扰的新一代 8051 单片机，采用 STC 第九代加密技术，无法解密，指令代码完全兼容传统 8051，但速度快 8--12 倍。内部集成高精度 R/C 时钟（±0.3%），±1%温漂（-40℃～+85℃），常温下温漂+0.6%（-20℃～+65℃），ISP 编程时 5MHz～35MHz 宽范围可设置，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 16 级复位门槛电压可选）。1 组高速异步串行通信口（UART），可在 2 组管脚之间进行切换，分时复用可作 2 组串口使用，针对串行口通信/电机控制干扰场合。内置比较器，功能更强大。

在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可现 STC15 系列单片机采用 STC-Y5 超高速 CPU 内核，在相同的时钟频率下，速度又比 STC 早期的 1T 系列单片机（如 STC12 系列/STC11 系列/STC10 系列）的速度快 20%。



1 增强型 8051 CPU，1T，单时钟/机器周期，速度比普通 8051 快 8--12 倍

2 工作电压：2.5V--5.5V

3 1K/2K/3K/4K/5K/7.5K 字节片内 Flash 程序存储器，擦写次数 10 万次以上

4 片内集成 256 字节的 SRAM

5 有片内 EEPROM 功能，擦写次数 10 万次以上

6 ISP/IAP，在系统可编程/在应用可编程，无需编程器/仿真器

7 内部高可靠复位，ISP 编程时 16 级复位门槛电压可选，可彻底省掉外部复位电路

8 工作频率范围：5MHz ~ 35MHz，相当于普通 8051 的 60MHz ~ 420MHz

9 内部高精度 R/C 时钟（±0.3%），±1%温漂（-40℃ ~ +85℃），常温下温漂+0.6%（-20℃ ~ +65℃），ISP 编程时内部时钟从 5MHz ~ 35MHz 可设（5.5296MHz/11.0592MHz/22.1184MHz/33.1776MHz）

10 不需外部晶振和外部复位，还可对外输出时钟和低电平复位信号

11 一组高速异步串行通信端口，可在 2 组管脚之间进行切换，分时复用可当 2 组串口使用：

串行口（RxD/P3.0，TxD/P3.1）可以切换到（RxD_2/P3.6，TxD_2/P3.7）

注意：建议用户将串口放在[P3.6/RxD_2，P3.7/TxD_2]上（[P3.0，P3.1]作下载/仿真用）；若用户未将串口切换到[P3.6/RxD_2，P3.7/TxD_2]，而是用[P3.0/RxD，P3.I/TxD]作串口，则务必在 ISP 编程时在 AIapp-ISP 软件的硬件选项中勾选“下次冷启动时，P3.2/P3.3 为 0/0 时才可以下载程序”

12 支持程序加密后传输，防拦截

13 支持 RS485 下载

14 低功耗设计：低速模式，空闲模式，掉电模式/停机模式。

15 可将掉电模式/停机模式唤醒的定时器：有内部低功耗掉电唤醒专用定时器

16 可将掉电模式/停机模式唤醒的资源有：

- INT0/P3.2，INT1/P3.3（INT0/INT1 上升沿下降沿中断均可），INT2/P3.6，INT3/P3.7，INT4/P3.0（INT2/INT3/INT4 仅可下降沿中断）；
- 管脚 RxD（可在 RxD/P3.0 和 RxD_2/P3.6 之间切换）；
- 管脚 T0/T2（下降沿，不产生中断，前提是在进入掉电模式/停机模式前相应的定时器中断已经被允许）；
- 内部低功耗掉电唤醒专用定时器。

17 共 2 个定时器/计数器，分别是 16 位可重装载的定时器/计数器 0（即 T0）和定时器/计数器 2（即 T2），并都可实现可编程时钟输出，另外管脚 MCLKO 可将内部主时钟对外分频输出（÷1 或÷2 或÷4）。

18 可编程时钟输出功能（对内部系统时钟或对外部管脚的时钟输入进行时钟分频输出）：

- 由于 STC15 系列 5V 单片机 I/O 口的对外输出速度最快不超过 13.5MHz，所以 5V 单片机的对外可编程时钟输出速度最快也不超过 13.5MHz；
- 而 3.3V 单片机 I/O 口的对外输出速度最快不超过 8MHz，故 3.3V 单片机的对外可编程时钟输出速度最快也不超过 8MHz

1) T0 在 P3.5/T0CLKO 进行可编程输出时钟（对内部系统时钟或对外部管脚 T0/P3.4 的时钟输入进行可编程时钟分频输出）；
2) T2 在 P3.0/T2CLKO 进行可编程输出时钟（对内部系统时钟或对外部管脚 T2/P3.1 的时钟输入进行可编程时钟分频输出）；

以上 2 个定时器/计数器均可 1 ~ 65536 级分频输出。

3) 主时钟在 P5.4/MCLKO 对外输出时钟，并可如下分频 MCLK/1，MCLK/2，MCLK/4

STC15W204S 系列单片机不支持外接外部晶体，其主时钟对外输出管脚 P5.4/MCLKO 只可以对外输出内部 R/C 时钟。MCLK 是指主时钟频率，MCLKO 是指主时钟输出。

STC15 系列 8-pin 单片机（如 STC15F104W 系列）在 MCLKO/P3.4 口对外输出时钟，STC15 系列 16-pin 及其以上单片机均在 MCLKO/P5.4 口对外输出时钟，且 STC15W 系列 20-pin 及其以上单片机除可在 MCLKO/P5.4 口对外输出时钟外，还可在 MCLKO_2/P1.6 口对外输出时钟。

19 比较器，可当 1 路 ADC 使用，并可作掉电检测，支持外部管脚 CMP+与外部管脚 CMP-进行比较，可产生中断，并可在管脚 CMPO 上产生输出（可设置极性），也支持外部管脚 CMP+与内部参考电压进行比较
若[P5.5/CMP+，P5.4/CMP-]被用作比较器正极（CMP+）/负极（CMP-），则[P5.5/CMP+，P5.4/CMP-]要被设置为高阻输入

20 硬件看门狗（WDT）

21 先进的指令集结构，兼容普通 8051 指令集，有硬件乘法/除法指令

22 通用 I/O 口（14/6 个），复位后为：准双向口/上拉（普通 8051 传统 I/O 口），
- 可设置成四种模式：准双向口上拉，强推挽上拉，仅为输入/高阻，开漏
- 每个 I/O 口驱动能力均可达到 20mA，但整个芯片电流最大不要超过 90mA.
- 如果 I/O 口不够用，可外接 74HC595（参考价 0.15 元）来扩展 I/O 口，并可多芯片级联扩展几十个 I/O 口。

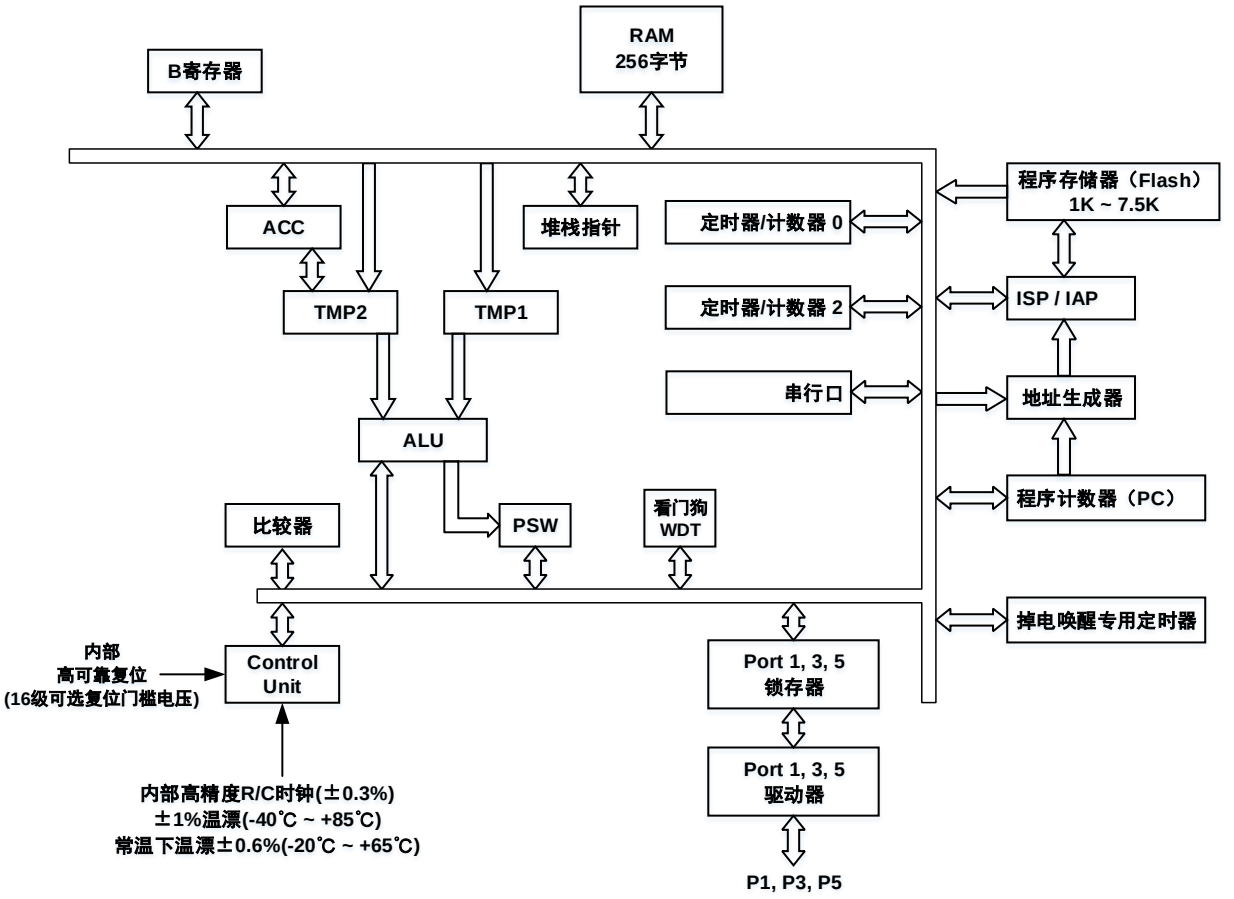
23 封装：SOP8，SOP16（6mm×9.9mm），DIP16

24 全部 175℃八小时高温烘烤，高品质制造保证

25 开发环境：在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可

3.3.2 内部结构图

STC15W204S 系列单片机的内部结构框图如下图所示。STC15W204S 系列单片机中包含中央处理器(CPU)、程序存储器(Flash)、数据存储器(SRAM)、定时器/计数器、掉电唤醒专用定时器、I/O 口、1 组高速异步串行通信端口、比较器、看门狗、片内高精度 R/C 时钟及高可靠复位等模块。



STC15W204S系列内部结构框图

3.3.3 STC15W204S 系列选型及价格一览表

型号	工作电压（V）	Flash 程序存储器 (byte)	SRAM 字节	串行口并可掉电唤醒	SPI	普通定时器计数器 T0/T2 外部脚能掉电唤醒	CCP PCA PWM 并可掉电唤醒	掉电唤醒专用定时器	标准外部中断支持掉电唤醒	A/D 8 路(3 路 PWM 可当 3 路 D/A 使用)	比较器(可当 1 路 A/D 使用,可作外部掉电检测)	DPTR	EEPROM	内部低压检测中断并可掉电唤醒	看门狗	内部高可靠复位 (可选复位门槛电压)	内部高精准时钟	可对外输出时钟及复位	程序加密后传输 (防拦截)	可设下次更新程序需口令	支持 RS485 下载	所有封装 SOP8SOP16/DIP16 价格 (RMB ¥)		
																						SOP8 (6 个 I/O 口)	SOP16 (14 个 I/O 口)	DIP16 (14 个 I/O 口)
STC15W204S 系列单片机选型价格一览表																								
STC15W201S	5.5-2.5	1K	256	1	-	2	-	有	5	-	有	1	4K	有	有	16 级	有	是	有	是	是	¥1.90	¥1.90	¥2.10
STC15W202S	5.5-2.5	2K	256	1	-	2	-	有	5	-	有	1	3K	有	有	16 级	有	是	有	是	是	¥1.90	¥1.90	¥2.10
STC15W203S	5.5-2.5	3K	256	1	-	2	-	有	5	-	有	1	2K	有	有	16 级	有	是	有	是	是	-	-	-
STC15W204S	5.5-2.5	4K	256	1	-	2	-	有	5	-	有	1	1K	有	有	16 级	有	是	有	是	是	¥1.90	¥1.90	¥2.10
IAP15W205S	5.5-2.5	5K	256	1	-	2	-	有	5	-	有	1	IAP	有	有	16 级	有	是	有	是	是	-	¥1.90	¥2.10
																						用户可将用户程序区的程序 FLASH 当 EEPROM 使用		
IRC15W207S 默认使用内部 24MHz 时钟	5.5-2.5	7.5K	256	1	-	2	-	有	5	-	有	1	IAP	有	有	固定	有	是	无	否	否	-	¥1.90	¥2.10
																						用户可将用户程序区的程序 FLASH 当 EEPROM 使用		

我们直销，所以低价，以上单价为 10K/M 起定量，量小每片需加 0.1 元，以上价格运费由客户承担，零售 10 片起，如对价格不满，可来电要求降价

STC15W204S 系列单片机只有定时器 0 和定时器 2，无定时器 1

提供客制化 IC 服务

如果要用 16-pin 单片机，建议用户选用 SOP16 封装。

若[P5.5/CMP+，P5.4/CMP-]被用作比较器正极（CMP+）/负极（CMP-），则[P5.5/CMP+，P5.4/CMP-]要被设置为高阻输入

程序加密后传输：程序拥有者产品出厂时将源程序和加密钥匙一起烧录 MCU 中，以后需要升级软件时，就可将程序加密后再用“发布项目程序“功能，生成一个用户自己界面的只有一个升级按钮的简单易用的升级软件，给最终使用者自己升级，而拦截不到您的原始程序。

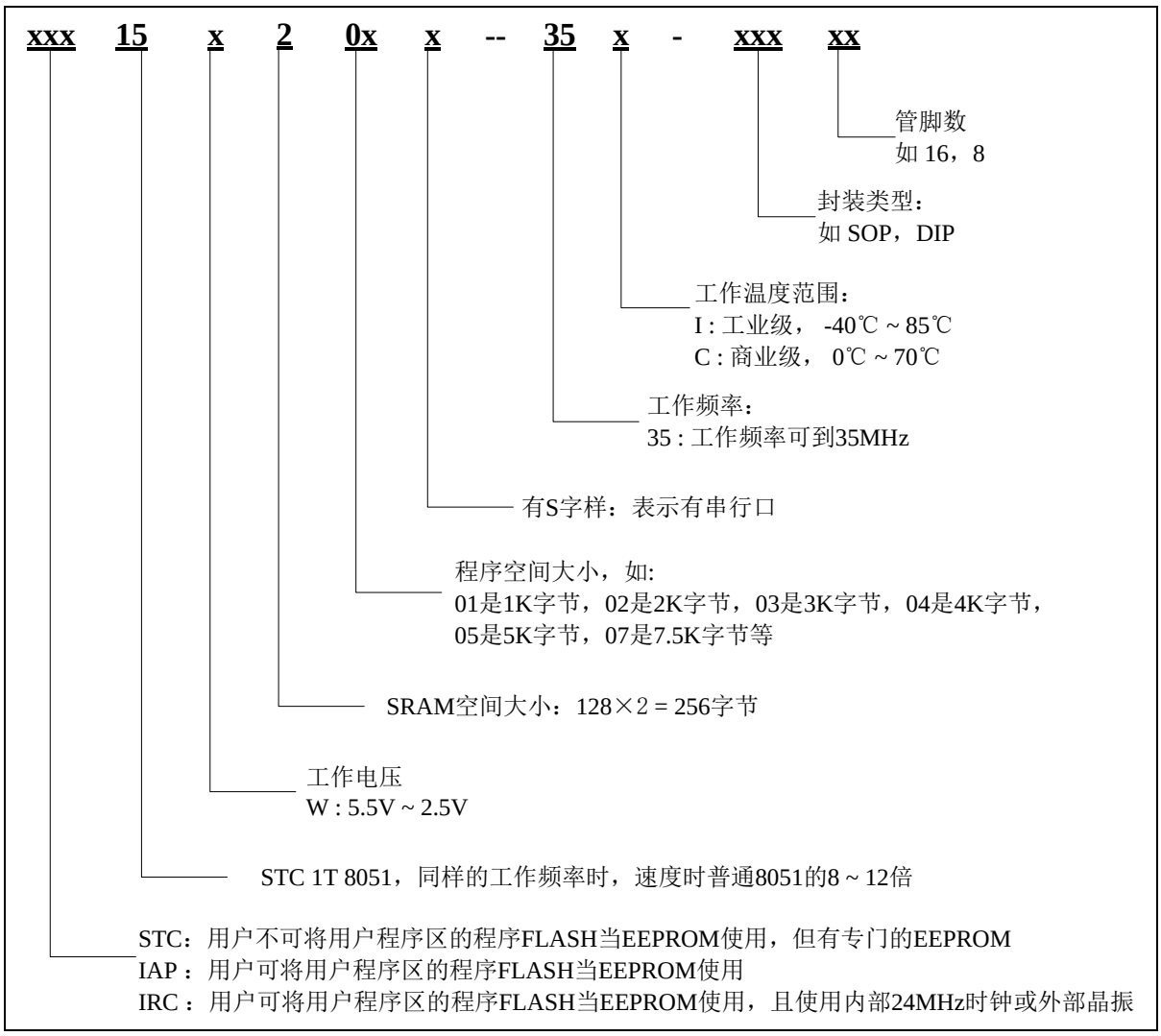
上表中以 IRC15W207S 型号的单片机默认使用内部 24MHz 时钟，其内部复位门槛电压固定，同时不支持“程序加密后传输”功能，其 P5.4 不可当复位管脚 RST 使用，且 P3.2/P303 与下载无关。

【总结】：STC15W204S 系列单片机有：

- ✓ 2 普通定时器/计数器（这 2 个普通定时器/计数器是指：T0 和 T2）；
- ✓ 掉电唤醒专用定时器；
- ✓ 5 个支持掉电唤醒的外部中断 INT0/INT1/INT2/INT3/INT4；
- ✓ 1 组高速异步串行通信端口；1 个比较器；1 个数据指针 DPTR 等功能。
- ✓ 表中“-”表示该型号的单片机无相应的功能。STC15W204S 系列单片机无 SPI、无 A/D 转换、无 CCP/PWM/PCA、无外部数据总线等功能。

因为程序区的最后 7 个字节单元被强制性的放入全球唯一 1D 号的内容，所以用户实际可以使用的程序空间大小要比选型表中的大小少 7 个字节。

3.3.4 STC15W204S 系列单片机命名规则



如何识别芯片版本号：如需知道芯片版本号，请查阅芯片表面印刷字中最下面一行的最后一个字母(如 A)，该字母代表芯片版本号(如 A 版)

命名举例：

1) STC15W201S-35I-SOP16 表示：
用户不可将用户程序区的程序 FLASH 当 EEPROM 使用，但有专门的 EEPROM。该单片机为 1T 8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 5.5V ~ 2.5V，SRAM 空间大小为 256 字节，程序空间大小为 1K，有一组串行口，工作频率可到 35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 SOP 贴片封装，管脚数为 16。

2) STC15W201S-35I-DIP16 表示：
用户不可将用户程序区的程序 FLASH 当 EEPROM 使用，但有专门的 EEPROM。该单片机为 1T 8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 5.5V ~ 2.5V，SRAM 空间大小为 256 字节，程序空间大小为 1K，有一组串行口，工作频率可到 35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 DIP 封装，管脚数为 16。

3) IAP15W205S-35I-SOP16 表示：

用户可将用户程序区的程序 FLASH 当 EEPROM 使用。该单片机为 1T 8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 5.5V ~ 2.5V，SRAM 空间大小为 256 字节，程序空间大小为 5K，有一组串行口，工作频率可到 35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 SOP 贴片封装，管脚数为 16。

4) IAP15W205S-35I-DIP16 表示：

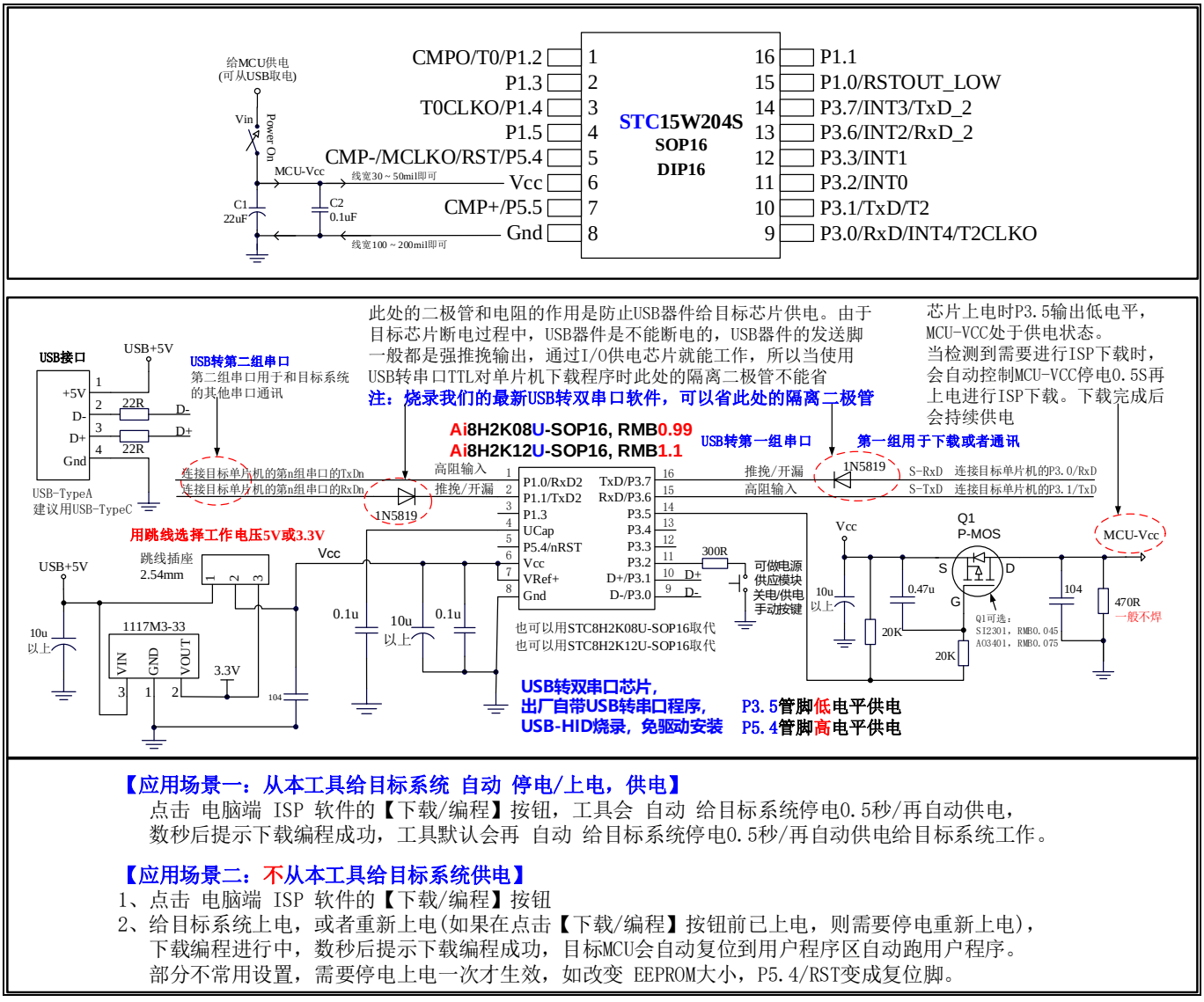
用户可将用户程序区的程序 FLASH 当 EEPROM 使用。该单片机为 1T 8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 5.5V ~ 2.5V，SRAM 空间大小为 256 字节，程序空间大小为 5K，有一组串行口，工作频率可到 35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 DIP 封装，管脚数为 16。

5) IRC15W207S-35I-DIP16 表示：

用户可将用户程序区的程序 FLASH 当 EEPROM 使用，且默认使用内部 24MHz 时钟。该单片机为 1T 8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 5.5V ~ 2.5V，SRAM 空间大小为 256 字节，程序空间大小为 7.5K，有一组串行口，工作频率可到 35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 DIP 封装，管脚数为 16。

3.3.5 管脚图及下载线路图

3.3.5.1 管脚图及下载线路图，最小系统（SOP16/DIP16）



【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。

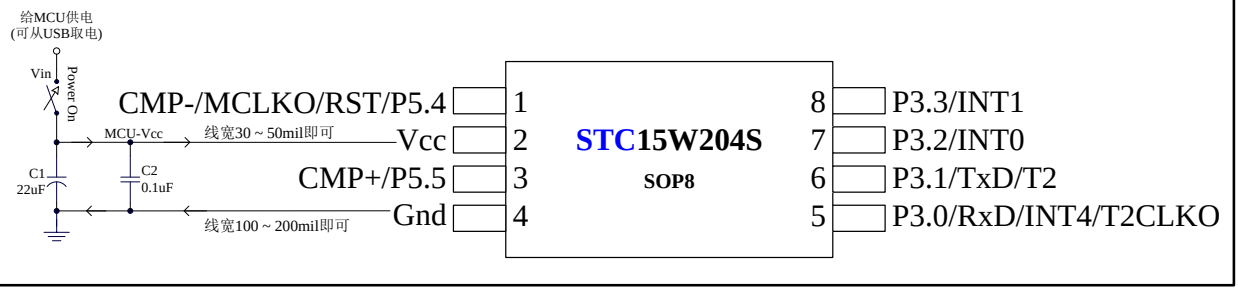
【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮

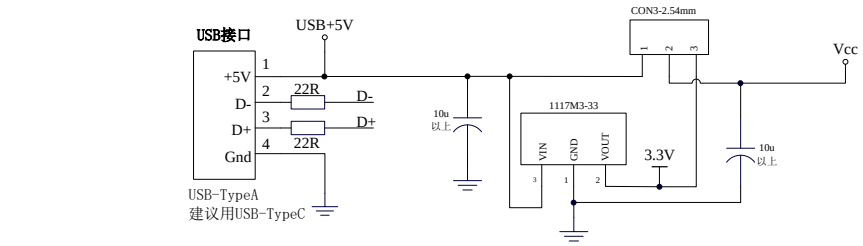
2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，下载编程进行中，数秒后提示下载编程成功，目标MCU会自动复位到用户程序区自动跑用户程序。

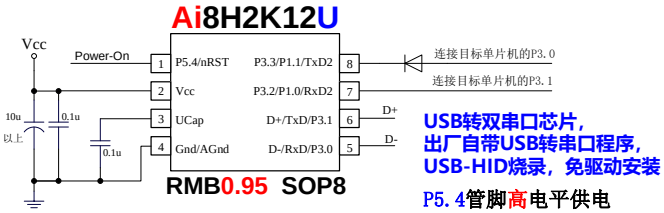
部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P5.4/RST变成复位脚。

3.3.5.2 管脚图及下载线路图，最小系统（SOP8）



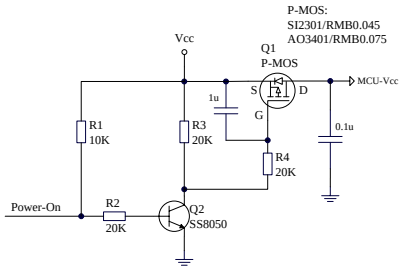
用跳线选择工作电压5V或3.3V





USB转双串口芯片，
出厂自带USB转串口程序，
USB-HID烧录，免驱动安装

P5.4管脚高电平供电



也可以用STC8H2K12U-SOP8取代Ai8H2K12U

【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，P5.4/RST变成复位脚。

3.3.6 管脚说明

管脚	封装		说明	
	SOP8	SOP16/ DIP16		
P1.0		15	P1.0	标准 I/O 口 PORT1[0]
			RSTOUT_LOW	上电后，输出低电平，在复位期间也是输出低电平，用户可用软件将其设置为高电平或低电平，如果要读外部状态，可将该口先置高后再读
P1.1		16	标准 I/O 口 PORT1[1]	
P1.2		1	P1.2	标准 I/O 口 PORT1[2]
			T0	定时器/计数器 0 的外部输入
			CMPO	比较器的比较结果输出管脚
P1.3		2	标准 I/O 口 PORT1[3]	
P1.4		3	P1.4	标准 I/O 口 PORT1[4]
			T0CLKO	定时器/计数器 0 的时钟输出可通过设置 INT_CLKO[0]位/T0CLKO 将该管脚配置为 T0CLKO，也可对 T0 脚的外部时钟输入进行分频输出
P1.5		4	标准 I/O 口 PORT1[5]	
P3.0	5	9	P3.0	标准 I/O 口 PORT3[0]
			RxD	串口数据接收端
			INT4	外部中断 4，只能下降沿中断，INT4 支持掉电唤醒
			T2CLKO	T2 的时钟输出可通过设置 INT_CLKO[2]位/T2CLKO 将该管脚配置为 T2CLKO
P3.1	6	10	P3.1	标准 I/O 口 PORT3[1]
			TxD	串口数据发送端
			T2	定时器/计数器 2 的外部输入
P3.2	7	11	P3.2	标准 I/O 口 PORT3[2]
			INT0	外部中断 0，既可上升沿中断也可下降沿中断。 如果 IT0(TCON.0)被置为 1，INT0 管脚仅为下降沿中断。 如果 IT0(TCON.0)被清 0，INT0 管脚既支持上升沿中断也支持下降沿中断。 INT0 支持掉电唤醒。
P3.3	8	12	P3.3	标准 I/O 口 PORT3[3]
			INT1	外部中断 1，既可上升沿中断也可下降沿中断。 如果 IT1(TCON.2)被置为 1，INT1 管脚仅为下降沿中断。 如果 IT1(TCON.2)被清 0，INT1 管脚既支持上升沿中断也支持下降沿中断。 INT1 支持掉电唤醒。
P3.6		13	P3.6	标准 I/O 口 PORT3[6]
			INT2	外部中断 2，只能下降沿中断，INT2 支持掉电唤醒
			RxD_2	串口数据接收端
P3.7		14	P3.7	标准 I/O 口 PORT3[7]
			INT3	外部中断 3，只能下降沿中断，INT3 支持掉电唤醒
			TxD_2	串口数据发送端
P5.4	1	5	P5.4	标准 I/O 口 PORT5[4]
			RST	复位脚(高电平复位)
			MCLKO	主时钟输出；输出的频率可为 MCLK/1，MCLK/2，MCLK/4（MCLK 是指主时钟频率）。 此系列的主时钟外部输出管脚 P5.4/MCLKO 只可以外部输出内部 R/C 时钟，MCLK 指主时钟频率。
			CMP-	比较器负极输入端 （若该口被用作比较器负极，则该口需被设置为高阻输入）
P5.5	3	7	P5.5	标准 I/O 口 PORT5[5]
			CMP+	比较器正极输入端 （若该口被用作比较器正极，则该口需被设置为高阻输入）
Vcc	2	6	电源正极	
Gnd	4	8	电源负极，接地	

3.3.7 USB-Link1D 工具自动停电/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】

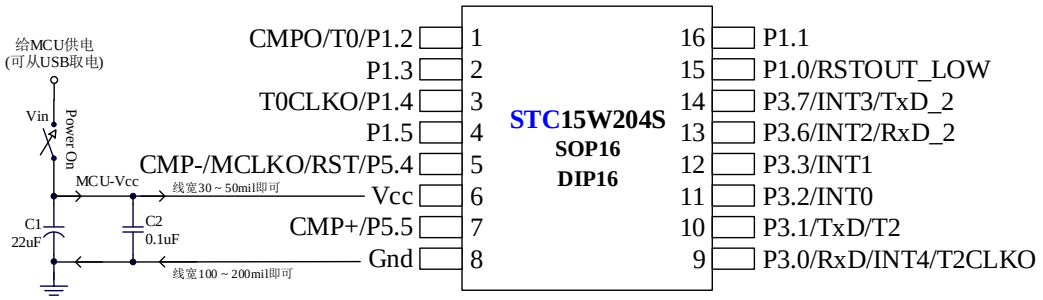
点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【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，P5.4/nRST变成复位脚



3.3.8 【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】

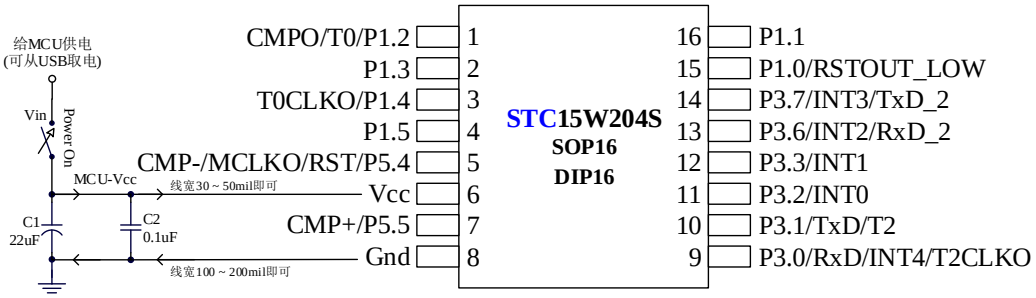
点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【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，P5.4/nRST变成复位脚



3.4 STC15W408AS 系列

STC15W408AS 系列 1T8051 单片机，超高速串口，高速 A/D，比较器，宽电压

不需外部晶振的单片机，不需外部复位的单片机

STC15W408AS 系列主要性能：

- ✓ 512 字节片内 RAM 数据存储器
- ✓ 高速：1 个时钟/机器周期，增强型 8051 内核，速度比传统 8051 快 7 ～ 12 倍
- ✓ 速度也比 STC 早期的 1T 系列单片机（如 STC12/11/10 系列）的速度快 20%
- ✓ 宽电压：2.4V ～ 5.5V
- ✓ 低功耗设计：低速模式，空闲模式，掉电模式（可由外部中断或内部掉电唤醒定时器唤醒）
- ✓ 不需外部复位的单片机，ISP 编程时 16 级复位门槛电压可选，内置高可靠复位电路
- ✓ 不需外部晶振的单片机，ISP 编程时内部时钟从 5MHz~35MHz 可设（相当于普通 8051：60 ～ 420MHz）

  内部高精度 R/C 时钟（±0.3%），±1%温漂（-40℃ ～ +85℃），常温下温漂±0.6%（-20℃ ～ +65℃）
- ✓ 支持掉电唤醒的资源有：INT0/INT1（上升沿/下降沿中断均可），INT2/INT3/INT4（下降沿中断）；CCP0/CCP1/CCP2/RxD/T0/T2 管脚；内部掉电唤醒专用定时器
- ✓ 4K/8K/13K/15.5K 字节片内 Flash 程序存储器，擦写次数 10 万次以上
- ✓ 大容量片内 EEPROM 功能，擦写次数 10 万次以上
- ✓ ISP/IAP，在系统可编程/在应用可编程，无需编程器/仿真器
- ✓ 高速 ADC，8 通道 10 位，速度可达 30 万次/秒，3 路 PWM 还可当 3 路 D/A 使用
- ✓ 比较器，可当 1 路 ADC 使用，并可作掉电检测，支持外部管脚 CMP+与外部管脚 CMP-进行比较，可产生中断，并可在管脚 CMPO 上产生输出（可设置极性），也支持外部管脚 CMP+与内部参考电压进行比较。
- ✓ **3** 通道捕获/比较单元（**CCP/PCA/PWM**）
  ----也可用来再实现 3 路 D/A 或 3 个定时器或 3 个外部中断（支持上升沿/下降沿中断）
- ✓ 利用 CCP/PCA 高速脉冲输出功能可实现 3 路 9 ~ 16 位 PWM（每通道占用系统时间小于 0.6%）
- ✓ 利用定时器 T0 的时钟输出功能可实现高精度的 8 ~ 16 位 PWM（占用系统时间小于 0.4%）
- ✓ 5 个定时器，2 个 16 位可重装载定时器/计数器（T0/T2，其中 T0 兼容普通 8051 的定时器/计数器），并都可实现可编程时钟输出，另外管脚 SysClkO 可将系统时钟对外分频输出（÷1 或÷2 或÷4），3 路 CCP/PCA 可再实现 3 个定时器
- ✓ 可编程时钟输出功能（对内部系统时钟或外部管脚的时钟输入进行时钟分频输出）：
  ① T0 在 P3.5 输出时钟；
  ② T2 在 P3.0 输出时钟，以上 2 个定时器/计数器输出时钟均可 1 ~ 65536 级分频输出；

③ 系统时钟在 P5.4/SysClkO 对外输出时钟（STC15 系列 8-pin 单片机的主时钟在 P3.4/MCLKO 对外输出时钟）

- ✓ 硬件看门狗（WDT）
- ✓ 超高速异步串行通信端口/UART，分时切换可当 3 组串口使用
- ✓ SPI 高速同步串行通信接口
- ✓ 先进的指令集结构，兼容普通 8051 指令集，有硬件乘法/除法指令
- ✓ 通用 I/O 口（26/18/14 个），复位后为：准双向口/弱上拉（8051 传统 I/O 口）。可设置四种模式：准双向口/弱上拉，强推挽/强上拉，仅为输入/高阻，开漏。每个 I/O 口驱动能力均可达到 20mA，但整个芯片最大不要超过 120mA

选择【STC15W408AS】系列单片机理由：

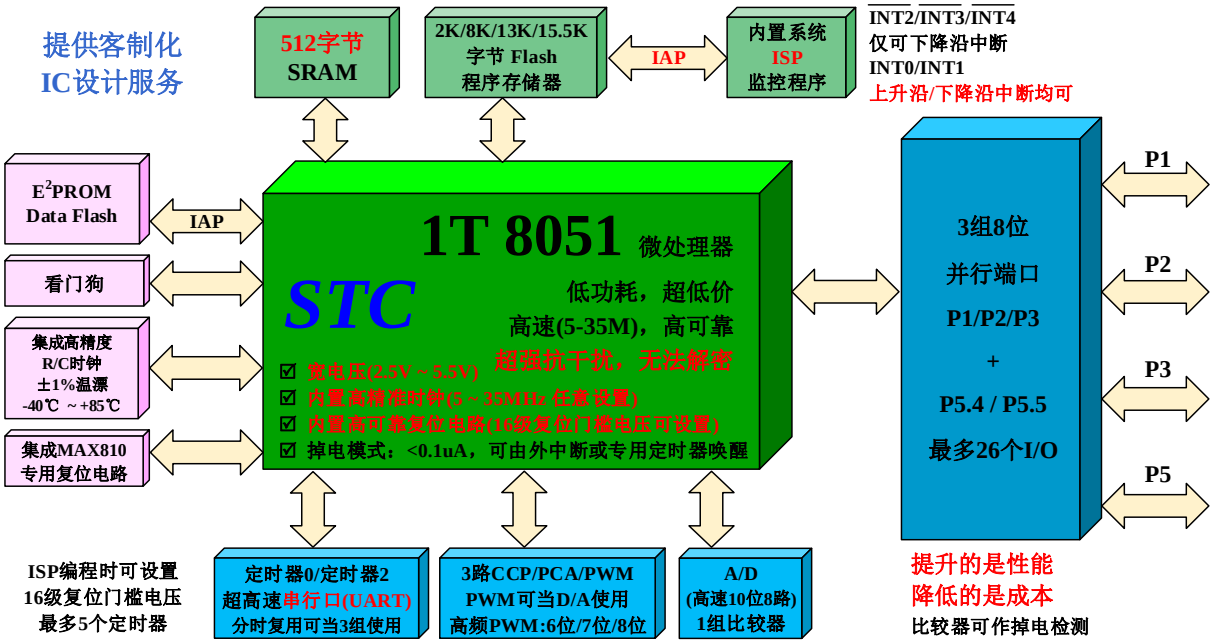
- ★ 不需外部晶振和外部复位，还可对外输出时钟和低电平复位信号
- ★ 无法解密，STC 第九代加密技术，现悬赏 20 万元人民币请专家帮忙查找加有无漏洞
- ★ 超强抗干扰：
  - ➢ 高抗静电（ESD 保护）整机轻松过 2 万伏静电测试
  - ➢ 轻松过 4kV 快速脉冲干扰（EFT 测试）
  - ➢ 宽电压，不怕电源抖动
  - ➢ 宽温度范围，-40℃ ~ +85℃
- ★ 大幅降低 EMI，内部可配置时钟，1 个时钟/机器周期，可用低频时钟
  ----出口欧美的有力保证
- ★ 超低功耗：
  - ➢ 掉电模式：外部中断唤醒功耗**<0.1uA**
  - ➢ 空闲模式：典型功耗<1mA
  - ➢ 正常工作模式：4mA ~ 6mA
  - ➢ 掉电模式可由外部中断或内部掉电唤醒专用定时器唤醒，适用于电池供电系统，如水表、气表便携式设备等
- ★ 利用 CCP/PCA 高速脉冲输出功能可实现 3 路 9 ~ 16 位 PWM（每通道占用系统时间小于 0.6%）
- ★ 利用定时器 T0 的时钟输出功能可实现高精度的 8 ~ 16 位 PWM（占用系统时间小于 0.4%）
- ★ 在系统可仿真，在系统可编程，无需专用编程器，无需专用仿真器，可远程升级
- ★ 可送 USB 型联机/脱机下载烧录工具 STC-U8W（人民币 100 元），1 万片/人/天，有自动烧录机接口

3.4.1 STC15W408AS 系列简介

STC15W408AS 系列单片机是 STC 生产的单时钟/机器周期（1T）的单片机，是宽电压/高速/高可靠/低功耗/超强抗干扰的新一代 8051 单片机，采用 STC 第九代加密技术，无法解密，指令代码完全兼容传统 8051，但速度快 8-12 倍。内部集成高精度 R/C 时钟（±0.3%），±1%温漂（-40℃～+85℃），常温下温漂±0.6%（-20℃～+65℃），ISP 编程时 5MHz~35MHz 宽范围可设置，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 16 级复位门槛电压可选）。3 路 CCP/PWM/PCA，8 路高速 10 位 A/D 转换（30 万次/秒），1 组超高速异步串行通信口 UART，可在 3 组管脚之间进行切换，分时复用可作 3 组串口使用），1 组高速同步串行通信端口 SPI，针对串行口通信/电机控制干扰场合。内置比较器，功能更强大。

在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可。

速度又比 STC 早现 STC15 系列单片机采用 STC-Y5 超高速 CPU 内核，在相同的时钟频率下，期的 1T 系列单片机（如 STC12 系列/STC11 系列/STC10 系列）的速度快 20%。



1 增强型 8051 CPU，1T，单时钟/机器周期，速度比普通 8051 快 8--12 倍

2 工作电压：5.5V--2.5V

3 1K/2K/4K/8K/13K/15.5K 字节片内 Flash 程序存储器，擦写次数 10 万次以上

4 片内集成 512 字节的 SRAM，包括常规的 256 字节 RAM <idata>和内部扩展的 256 字节 XRAM <xdata>

5 有片内 EEPROM 功能，擦写次数 10 万次以上

6 ISP/IAP，在系统可编程/在应用可编程，无需编程器/仿真器

7 共 8 通道 10 位高速 ADC，速度可达 30 万次/秒，3 路 PWM 还可当 3 路 D/A 使用

8 共 3 通道捕获/比较单元（CCP/PWM/PCA）
----也可用来再实现 3 个定时器或 3 个外部中断（支持上升沿/下降沿中断）或 3 路 D/A

9 利用 CCP/PCA 高速脉冲输出功能可实现 3 路 9 ~ 16 位 PWM（每通道占用系统时间小于 0.6%

10 利用定时器 T0 的时钟输出功能可实现高精度的 8 ~ 16 位 PWM（占用系统时间小于 0.4%）

11 内部高可靠复位，ISP 编程时 16 级复位门槛电压可选，可彻底省掉外部复位电路

12 工作频率范围：5MHz ~ 35MHz，相当于普通 8051 的 60MHz ~ 420MHz

13 内部高精度 R/C 时钟（±0.3%），±1%温漂（-40℃ ~ +85℃），常温下温漂+0.6%（-20℃ ~ +65℃）ISP 编程时内部时钟从 5MHz ~ 35MHz 可设（5.5296MHz/11.0592MHz/22.1184MHz/33.1776MHz）

14 不需外部晶振和外部复位，还可对外输出时钟和低电平复位信号

15 一组高速异步串行通信端口，可在 3 组管脚之间进行切换，分时复用可当 3 组串口使用：

串行口（RxD/P3.0，TxD/P3.1）可以切换到（RxD_2/P3.6，TxD_2/P3.7），
还可以切换到（RxD_3/P1.6，TxD_3/P1.7）

注意：建议用户将串口放在[P3.6/RxD_2，P3.7/TxD_2]或[P1.6/RxD_3，P1.7/TxD_3]上（[P3.0，P3.1]做下载/仿真用）；若用户未将串口切换到[P3.6/RxD_2，P3.7/TxD_2]或[P1.6/RxD_3，P1.7/TxD_3]，而是用[P3.0/RxD，P3.1/TxD]作串口，则务必在 ISP 编程时在 AIapp-ISP 软件的硬件选项中勾选“下次冷启动时，P3.2/P3.3 为 0/0 时才可以下载程序”

16 一组高速同步串行通信端口 SPI

17 支持程序加密后传输，防拦截

18 支持 RS485 下载

19 低功耗设计：低速模式，空闲模式，掉电模式/停机模式

20 可将掉电模式/停机模式唤醒的定时器：有内部低功耗掉电唤醒专用定时器。

21 可将掉电模式/停机模式唤醒的资源有：

- INT0/P3.2，INT1/P3.3（INT0/INT1 上升沿下降沿中断均可），INT2/P3.6，INT3/P3.7，INT4/P3.0（INT2/INT3/INT4 仅可下降沿中断）；
- 管脚 CCP0/CCP1/CCP2；
- 管脚 RxD（可在 RxD/P3.0、RxD_2/P3.6 和 RxD_3/P1.6 之间切换）；
- 管脚 T0/T2（下降沿，不产生中断，前提是在进入掉电模式/停机模式前相应的定时器中断已经被允许）；
- 内部低功耗掉电唤醒专用定时器。

22 共 5 个定时器/计数器，2 个 16 位可重装载定时器/计数器（T0/T2，其中 T0 兼容普通 8051 的定时器/计数器），并均可独立实现对外可编程时钟输出（2 通道），另外管脚 SysClkO 可将系统时钟对外分频输出（÷1 或÷2 或÷4），3 路 CCP/PWM/PCA 还可再实现 3 个定时器

23 可编程时钟输出功能（对内部系统时钟或对外部管脚的时钟输入进行时钟分频输出）：

- 由于 STC15 系列 5V 单片机 I/O 口的对外输出速度最快不超过 13.5MHz，所以 5V 单片机的对外可编程时钟输出速度最快也不超过 13.5MHz
- 而 3.3V 单片机 I/O 口的对外输出速度最快不超过 8MHz，故 3.3V 单片机的对外可编程时钟输出速度最快也不超过 8MHz

1) T0 在 P3.5/T0CLKO 进行可编程输出时钟（对内部系统时钟或对外部管脚 T0/P3.4 的时钟输入进行可编程时钟分频输出）；

2) T2 在 P3.0/T2CLKO 进行可编程输出时钟(对内部系统时钟或对外部管脚 T2/P3.1 的时钟输入进行可编程时钟分频输出)；
以上 2 个定时器/计数器均可 1 ~ 65536 级分频输出。

3) 系统时钟在 P5.4/SysClkO 或 P1.6/XTAL2/SysClkO_2 对外输出时钟，并可如下分频 SysClk/1，SysClk/2，SysClk/4。

系统时钟是指对主时钟进行分频后供给 CPU、定时器的实际工作时钟；主时钟可以是内部 R/C 时钟，也可以是外部输入的时钟或外部晶体振荡产生的时钟；SysClk 是指系统时钟频率，SysClkO 是指系统时钟输出。

STC15 系列中除 STC15W408AS 系列、STC15W4K32S4 系列、STC15W1K08PWM 系列及 STC15W1K20S-LQFP64 单片机是将系统时钟对外分频输出外，其他系列单片机均是将主时钟对外分频输出。

24 比较器，可当 1 路 ADC 使用，并可做掉电检测，支持外部管脚 CMP+与外部管脚 CMP-进行比较，可产生中断，并可在管脚 CMPO 上产生输出（可设置极性），也支持外部管脚 CMP+与内部参考电压进行比较

若[P5.5/CMP+，P5.4/CMP-]被用作比较器正极（CMP+）/负极（CMP-），则[P5.5/CMP+，P5.4/CMP-]要被设置为高阻输入
除 P5.5 可用作比较器正极（CMP+）外，8 路 ADC 口也可用作比较器正极（CMP+）。

25 硬件看门狗（WDT）

26 先进的指令集结构，兼容普通 8051 指令集，有硬件乘法/除法指令

27 通用 I/O 口（26/18/14 个），复位后为：准双向口/弱上拉（普通 8051 传统 I/O 口）

可设置成四种模式：准双向口/弱上拉，强推挽/强上拉，仅为输入/高阻，开漏
每个 I/O 口驱动能力均可达到 20mA，但整个芯片电流最大不要超过 90mA。
如果 I/O 口不够用，可外接 74HC595（参考价 0.15 元）来扩展 I/O 口，并可多芯片级联扩展几十个 IO 口

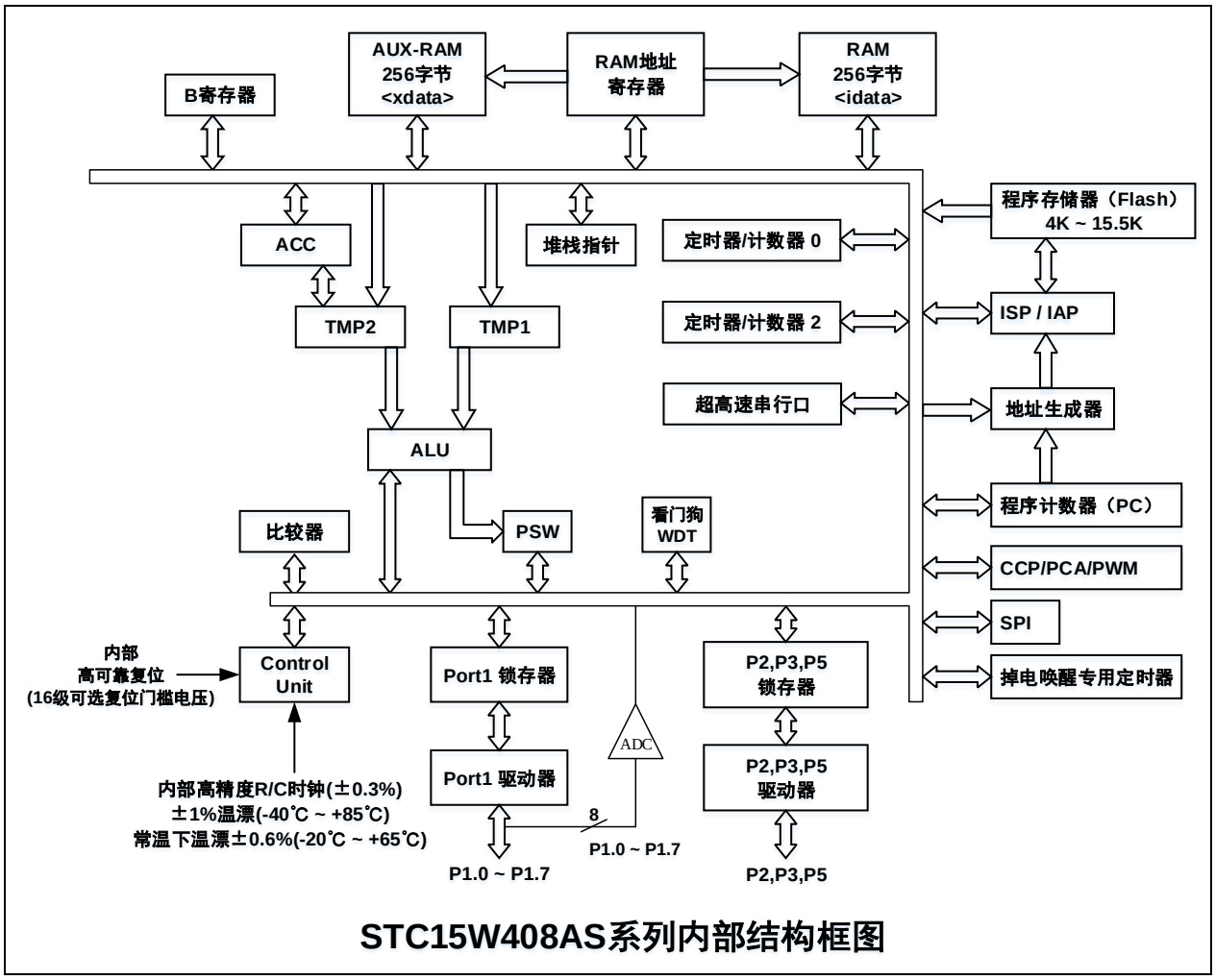
28 封装：SOP28，TSSOP28（6.4mm×9.7mm），QFN28（5mm×5mm），SKDIP28，SOP20，TSSOP20（6.5mm×6.5mm），DIP20，SOP16，DIP16

29 全部 175℃八小时高温烘烤，高品质制造保证

30 开发环境：在 Kei C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可

3.4.2 内部结构图

STC15W408AS 系列单片机的内部结构框图如下图所示。STC15W408AS 系列单片机中包含中央处理器（CPU）、程序存储器（Flash）、数据存储器（SRAM）、定时器/计数器、掉电唤醒专用定时器、I/O 口、高速 A/D 转换（30 万次/秒）、比较器、看门狗、高速异步串行通信端口 UART、CCP/PWM/PCA、高速同步串行端口 SPI，片内高精度 R/C 时钟及高可靠复位等模块。STC15W408AS 系列单片机几乎包含了数据采集和控制中所需的所有单元模块，可称得上是一个片上系统（SysTem Chip 或 SysTem on Chip，简写为 STC，这是 STC 名称的由来）。



STC15W408AS系列内部结构框图

3.4.3 STC15W408AS 系列选型一览表

型号	工作电压(V)	Flash程序存储器(byte)	SRAM字节	串行口并可掉电唤醒	SPI	普通定时器T0/T2外部管脚也能掉电唤醒	CCP PCA PWM并可掉电唤醒	掉电唤醒专用定时器	标准外部中断支持掉电唤醒	A/D 8路(3路PWM可当3路D/A使用)	比较器(可当1路ADC使用,可作掉电检测)	DPTR	EEPROM	内部低压检测中断并可掉电唤醒	看门狗	内部高可靠复位(可选复位门槛电压)	内部高精准时钟	可对外输出时钟及复位	程序加密后传输(防拦)截	可设下次更新程序需口令	支持RS485下载	所有封装 SOP28/SKDIP28/TSSOP28/QFN28 SOP20/DIP20/TSSOP20 SOP16/DIP16 部分封装价格(RMB ¥)								
																						SOP16	DIP16	SOP20	DIP20	TSSOP20	SOP28	SKDIP28	TSSOP28	QFN28
STC15W408AS 系列单片机选型价格一览表，大批量现货供应中 特别提示：3 路 CCP/PCA/PWM 还可当 3 路定时器使用																														
STC15W401AS	2.5-5.5	1K	512	1	有	2	3-ch	有	5	10-bit	有	1	5K	有	有	16 级	有	是	有	是	是	-	-	-	-	-	-	-	-	-
STC15W402AS	2.5-5.5	2K	512	1	有	2	3-ch	有	5	10-bit	有	1	5K	有	有	16 级	有	是	有	是	是	-	-	-	-	-	-	-	-	-
STC15W404AS	2.5-5.5	4K	512	1	有	2	3-ch	有	5	10-bit	有	1	9K	有	有	16 级	有	是	有	是	是	-	-	-	-	-	-	-	-	-
STC15W408AS	2.5-5.5	8K	512	1	有	2	3-ch	有	5	10-bit	有	1	5K	有	有	16 级	有	是	有	是	是	¥2.90	¥3.20	¥3.20	¥3.30	¥2.90	¥3.30	¥3.40	¥3.30	¥3.30
IAP15W413AS	2.5-5.5	13K	512	1	有	2	3-ch	有	5	10-bit	有	1	IAP	有	有	16 级	有	是	有	是	是	¥2.90	¥3.50	¥3.20	¥3.30	¥2.90	¥3.30	¥3.40	¥3.30	¥3.60
																						用户可将用户程序区的程序 Flash 当 EEPROM 使用								
IRC15W415AS 默认使用外部晶振如无外部晶振则使用内部 24MHz 时钟	2.5-5.5	15.5K	512	1	有	2	3-ch	有	5	10-bit	有	1	IAP	有	有	固定	有	是	有	否	否	¥2.90	¥3.50	¥3.20	¥3.30	¥2.90	¥3.30	¥3.40	¥3.30	¥3.60
																						用户可将用户程序区的程序 Flash 当 EEPROM 使用								

STC5W408AS 系列单片机只有定时器 0 和定时器 2，无定时器 1。若定时器不够用，3 路 CCP/PCA/PWM 可作 3 路定时器使用。

我们直销，所以低价，以上单价为 10K/M 起定量，量小每片需加 0.1 元，以上价格运费由客户承担，零售 10 片起，如对价格不满，可来电要求降价

程序加密后传输：程序拥有者产品出厂时将源程序和加密钥匙一起烧录 MCU 中，以后需要升级软件时，就可将程序加密后再用“发布项目程序“功能，生成一个用户自己界面的只有一个升级按钮的简单易用的升级软件，给最终使用者自己升级，而拦截不到您的原始程序。

若[P5.5/CMP+，P5.4/CMP-]被用作比较器正极（CMP+）/负极（CMP-），则[P5.5/CMP+，P5.4/CMP-]要被设置为高阻输入

STC15W408AS 系列单片机除 P5.5 可用作比较器正极（CMP+）外，8 路 ADC 口也可用作比较器正极（CMP+）

一秒钟能运行 1000 万条指令的 STC 8051 也能做四轴飞行器。

简单的四轴飞行器可采用一片 STC15W4K32S4 来完成，真正商用高端无人航拍四轴飞行器流行做法是 4 个无刷电机各用一片 STC15W404AS 控制（用到它 3 路 PWM+3 通道比较器/8 路 ADC 口也可设为比较器的正极），中央飞控系统用一片 STC15W4K48S4。

IRC15W415AS 型号单片机的内部复位门槛电压固定，P5.4 不可当复位管脚 RST 使用，[XTAL2/P1.6，XTAL1/P1.7]不可当 I/O 口使用，P3.2/P3.3 与下载无关，且不支持“程序加密后传输”功能。

如果要用 28-pin 单片机，建议用户选用 S0P28 封装；
如果要用 20-pin 单片机，建议用户选用 S0P20 封装；
如果要用 16-pin 单片机，建议用户选用 S0P16 封装。提供客制化 IC 服务。

因为程序区的最后 7 个字节单元被强制性的放入全球唯一 1D 号的内容，所以用户实际可以使用的程序空间大小要比选型表中的大小少 7 个字节。

【总结】: STC15W408AS 系列单片机（含 IRC15W415AS）有：

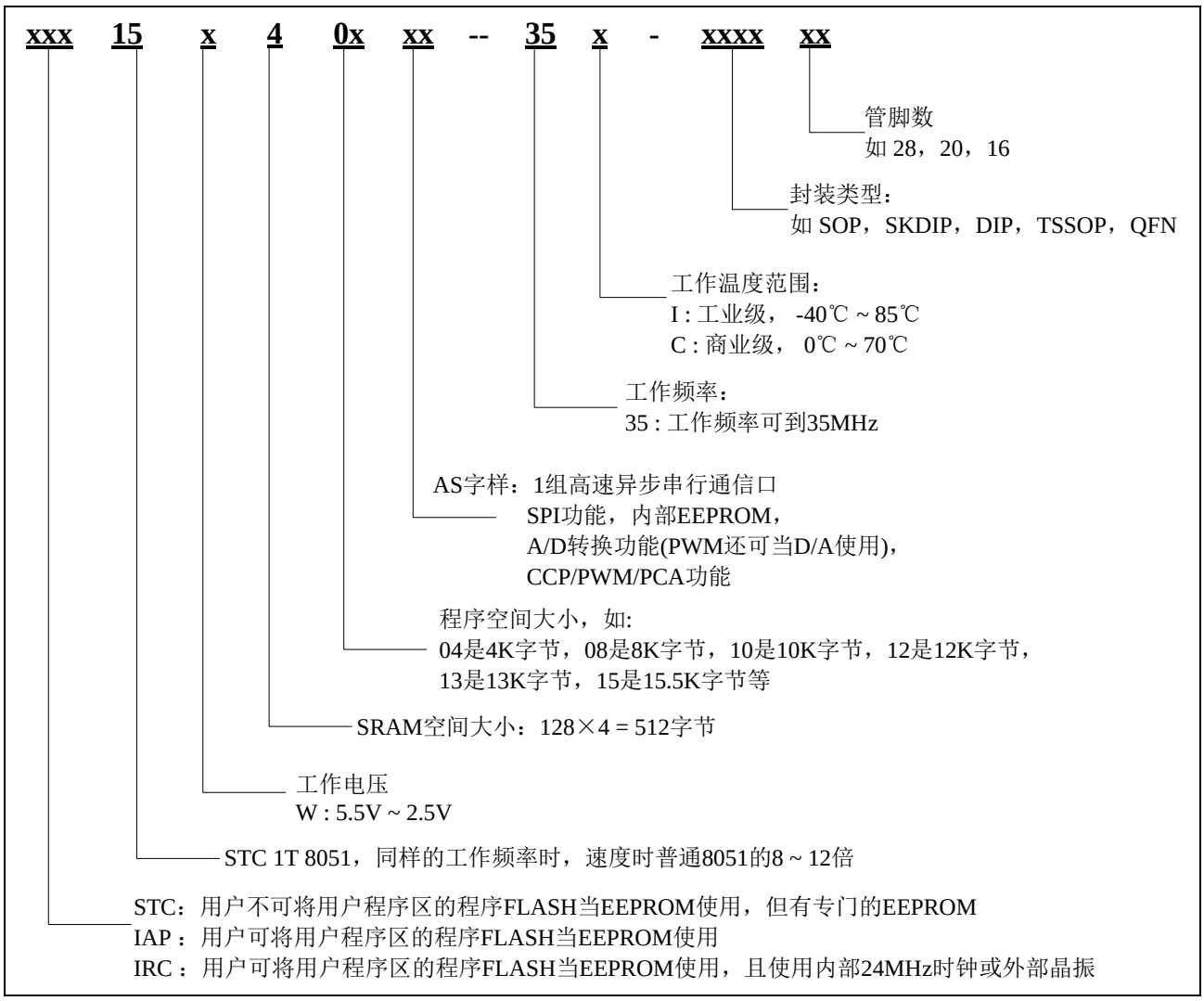
- ✓2 普通定时器/计数器（T0 和 T2），3 路 CCP/PWM/PCA（可再实现 3 个定时器使用）；
- ✓掉电唤醒专用定时器；
- ✓5 个支持掉电唤醒的外部中断 INT0/INT1/INT2/INT3/NT4；
- ✓1 组高速异步串行通信端口；
- ✓1 组高速同步串行通信端口 SPI；
- ✓8 路高速 10 位 A/D 转换器；
- ✓1 个比较器；
- ✓1 个数据指针 DPTR 等功能。
- ✓STC15W408AS 系列单片机没有外部数据总线。

3.4.4 STC15W408AS 系列单片机封装价格一览表

型号	工作电压（V）	工作频率 (MHz)	工作温度 (I--工业级)	所有封装 SOP28/ TSSOP28/ SKDIP28/ QFN28/ SOP20/ TSSOP20/ DIP20/ SOP16/ DIP16								
				SOP28 (26 个 I/O 口)	TSSOP28 (26 个 I/O 口)	SKDIP28 (26 个 I/O 口)	QFN28 (26 个 I/O 口	SOP20 (18 个 I/O 口)	TSSOP20 (18 个 I/O 口)	DIP20 (18 个 I/O 口)	SOP16 (14 个 I/O 口)	DIP16 (14 个 I/O 口)
STC15W408AS 系列单片机封装价格一览表												
STC15W401AS	5.5-2.5	35	-40℃ ~ +85℃	-	-	-	-	-	-	-	-	-
STC15W402AS	5.5-2.5	35	-40℃ ~ +85℃	-	-	-	-	-	-	-	-	-
STC15W404AS	5.5-2.5	35	-40℃ ~ +85℃	-	-	-	-	-	-	-	-	-
STC15W408AS	5.5-2.5	35	-40℃ ~ +85℃	¥3.30	¥3.30	¥3.40	¥3.30	¥3.20	¥2.90	¥3.30	¥2.90	¥3.20
IAP15W413AS	5.5-2.5	35	-40℃ ~ +85℃	¥3.30	¥3.30	¥3.40	¥3.60	¥3.20	¥2.90	¥3.30	¥2.90	¥3.50
IRC15W415AS	5.5-2.5	35	-40℃ ~ +85℃	¥3.30	¥3.30	¥3.40	¥3.60	¥3.20	¥2.90	¥3.30	¥2.90	¥3.50

我们直销，所以低价，以上单价为 10K 起订，量小每片需加 0.1 元，以上价格运费由客户承担，零售 10 片起，如对价格不满，可来电要求降价。

3.4.5 STC15W408AS 系列单片机命名规则



命名举例：

1) STC15W404AS-35I-SOP16 表示：

用户不可以将用户程序区的程序 FLASH 当 EEPROM 使用，但有专门的 EEPROM。该单片机为 1T 8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 5.5V ~ 2.5V，SRAM 空间大小为 512 字节，程序空间大小为 4K，有 1 组高速异步串行通信端口 UART 及 SPI、内部 EEPROM、A/D 转换、CCP/PCA/PWM 功能，工作频率可到 35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 SOP 贴片封装，管脚数为 16。

2) STC15W408AS-35I-SOP20 表示：

用户不可以将用户程序区的程序 FLASH 当 EEPROM 使用，但有专门的 EEPROM。该单片机为 1T 8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 5.5V ~ 2.5V，SRAM 空间大小为 512 字节，程序空间大小为 8K，有 1 组高速异步串行通信端口 UART 及 SPI、内部 EEPROM、A/D 转换、CCP/PCA/PWM 功能，工作频率可到 35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 SOP 贴片封装，管脚数为 20。

3) STC15W412AS-35I-SOP28 表示：

用户不可以将用户程序区的程序 FLASH 当 EEPROM 使用，但有专门的 EEPROM，该单片机为 1T 8051 单片机，同样工作频率时，速度是普通 8051 的 8～12 倍，其工作电压为 5.5V～2.5V，SRAM 空间大小为 512 字节，程序空间大小为 12K，有 1 组高速异步串行通信端口 UART 及 SPI、内部 EEPROM、A/D 转换、CCP/PCA/PWM 功能，工作频率可到 35MHz，为工业级芯片，工作温度范围为-40℃～85℃，封装类型为 SOP 贴片封装，管脚数为 28。

4) IAP15W413AS-35I-SOP28 表示：

用户可以将用户程序区的程序 FLASH 当 EEPROM 使用，该单片机为 IT8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 5.5V ~ 2.5V，SRAM 空间大小为 512 字节，程序空间大小为 13K，有 1 组高速异步串行通信端口 UART 及 SPI、内部 EEPROM、A/D 转换、CCP/PCA/PWM 功能，工作频率可到 35MHz，为工业级芯片，工作温度范围为-40℃～85℃，封装类型为 SOP 贴片封装，管脚数为 28。

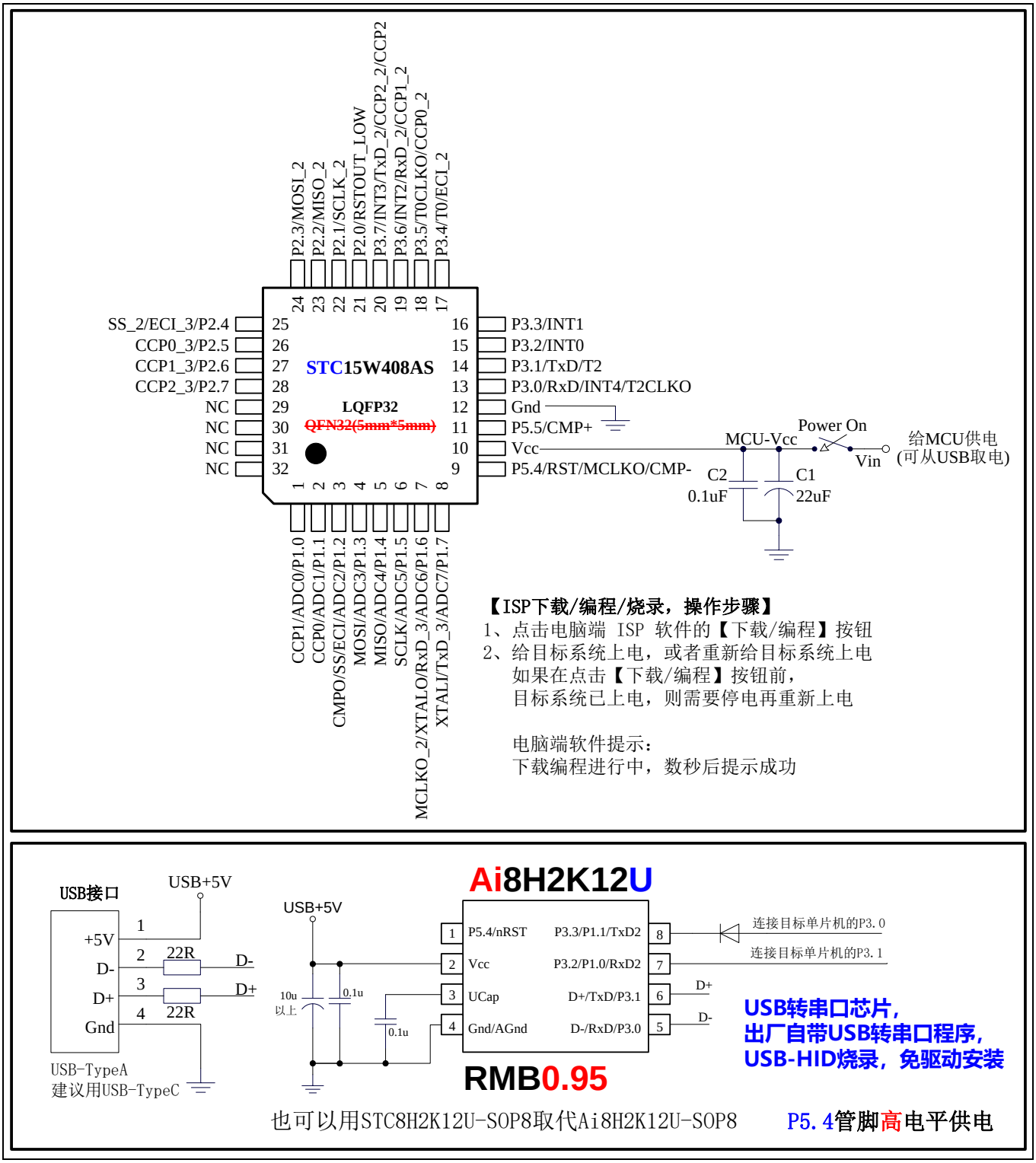
5) IAP15W413AS-35I-SKDIP28 表示：

用户可以将用户程序区的程序 FLASH 当 EEPROM 使用，该单片机为 IT8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 5.5V ~ 2.5V，SRAM 空间大小为 512 字节，程序空间大小为 13K，有 1 组高速异步串行通信端口 UART 及 SPI、内部 EEPROM、A/D 转换、CCP/PCA/PWM 功能，工作频率可到 35MHz，为工业级芯片，工作温度范围为-40℃～85℃，封装类型为 SKDIP 封装，管脚数为 28。

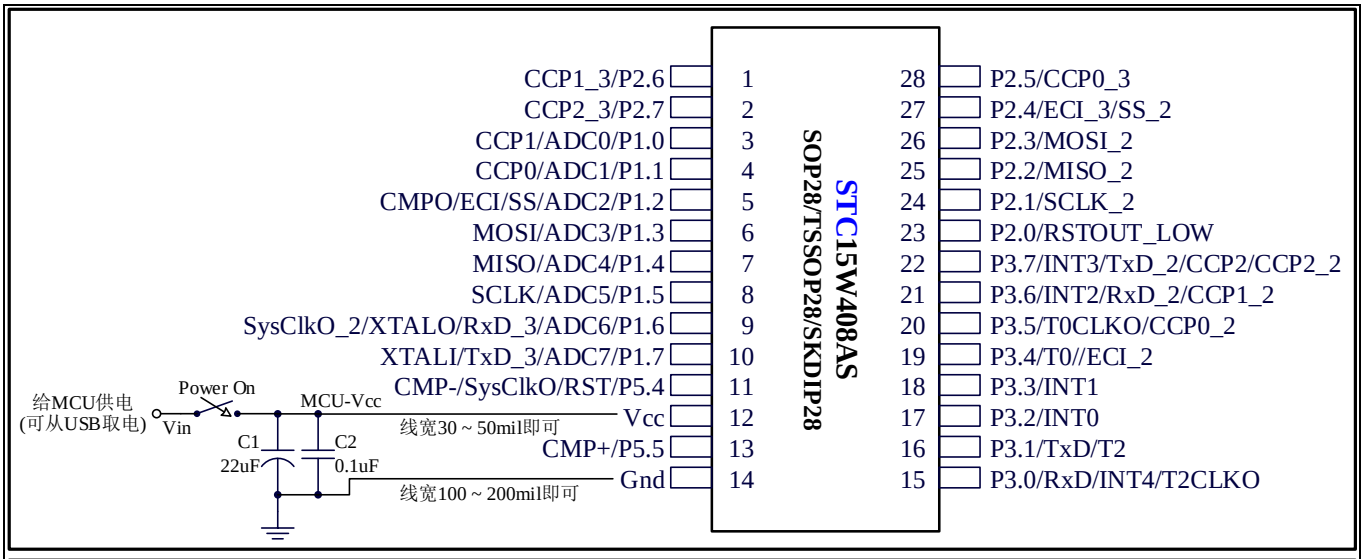
如何识别芯片版本号：如需知道芯片版本号，请查阅芯片表面印刷字中最下面一行的最后一个字母（如 A），该字母代表芯片版本号（如 A 版）

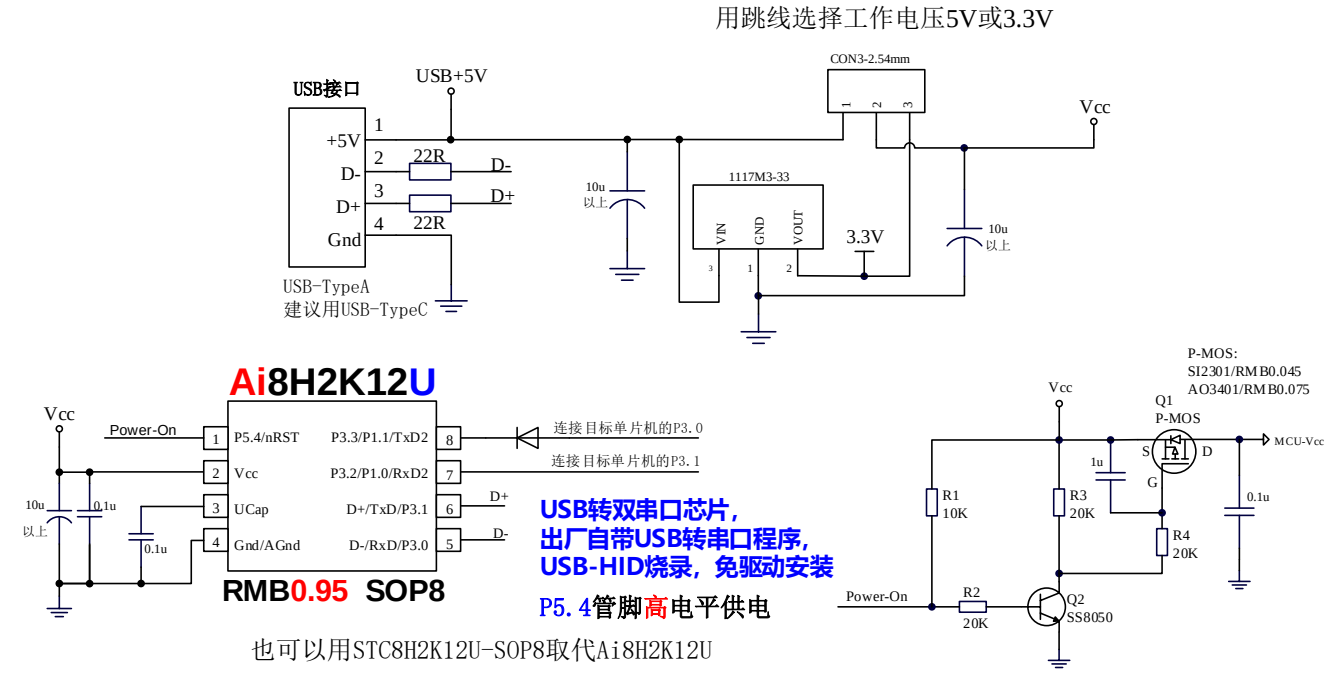
3.4.6 管脚图及下载线路图

3.4.6.1 管脚图及下载线路图，最小系统（LQFP32）



3.4.6.2 管脚图及下载线路图，最小系统（SOP28/TSSOP28/SKDIP28）





【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。

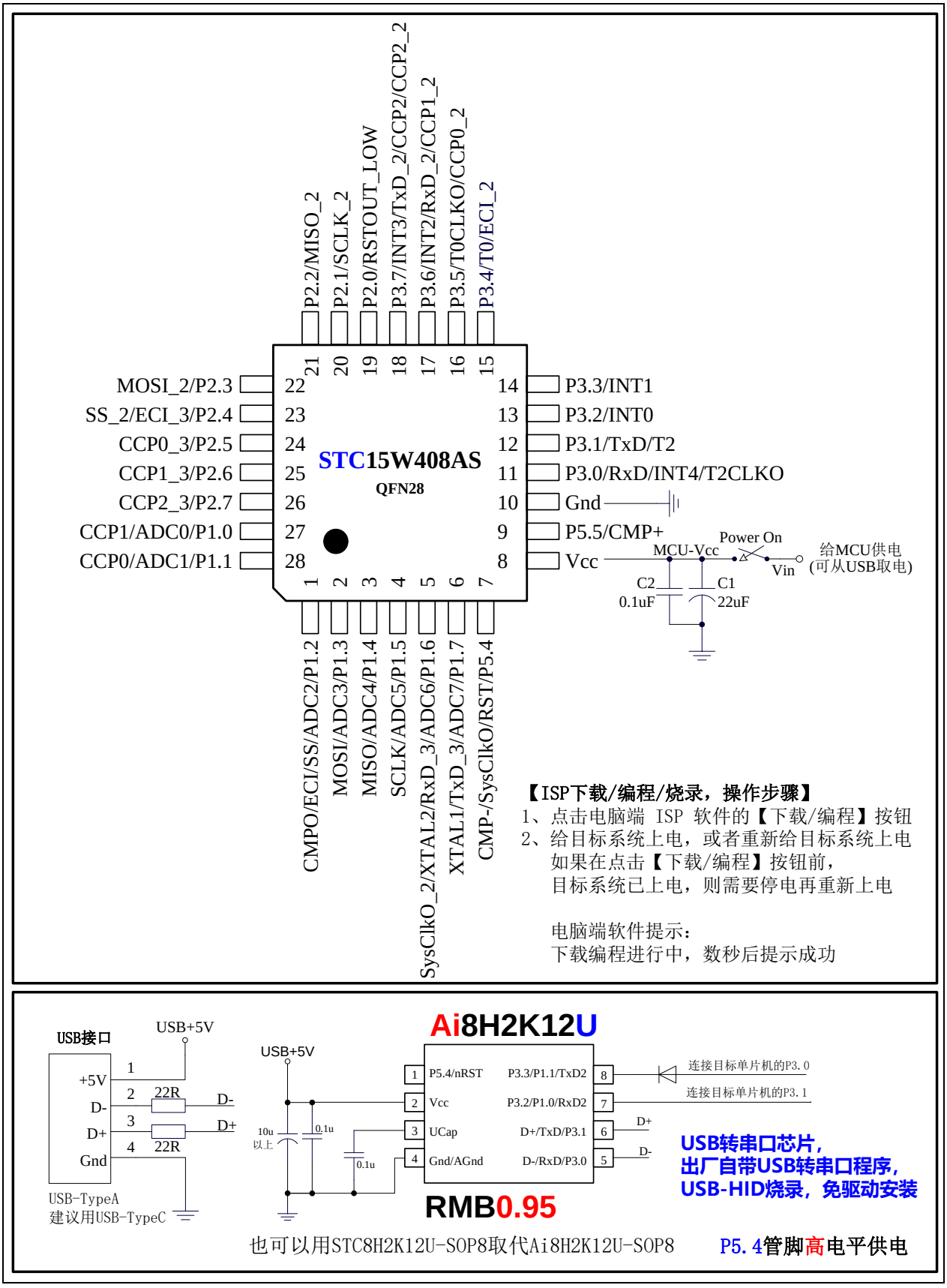
【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮

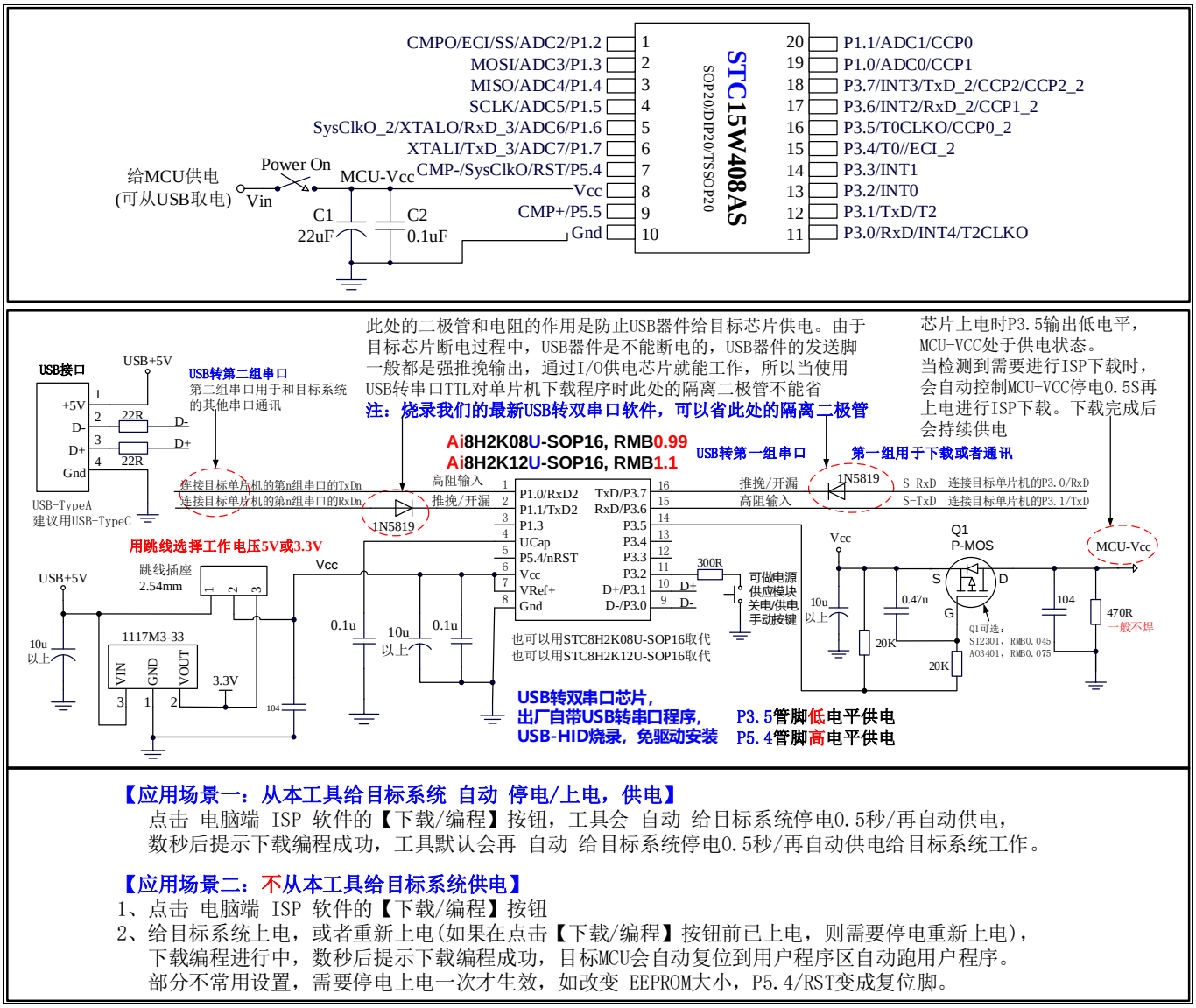
2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，下载编程进行中，数秒后提示下载编程成功，目标MCU会自动复位到用户程序区自动跑用户程序。

部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P5.4/RST变成复位脚。

3.4.6.3 管脚图及下载线路图，最小系统（QFN28）



3.4.6.4 管脚图及下载线路图，最小系统（SOP20/DIP20/TSSOP20）



【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。

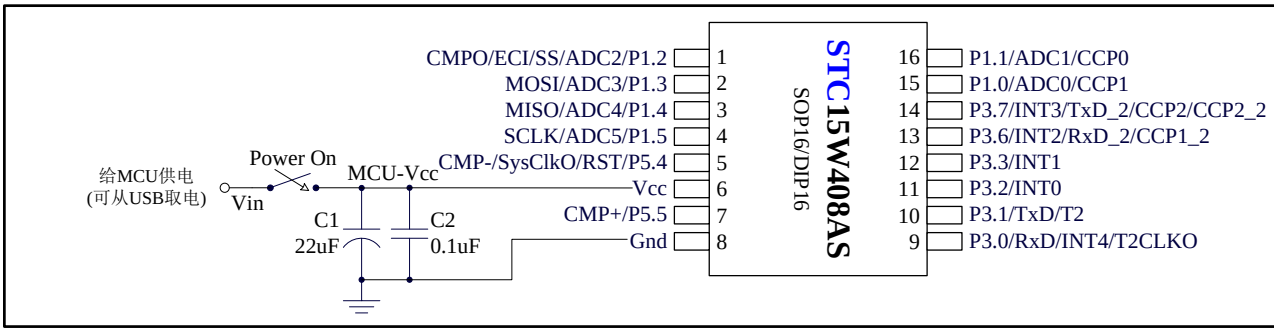
【应用场景二：不从本工具给目标系统供电】

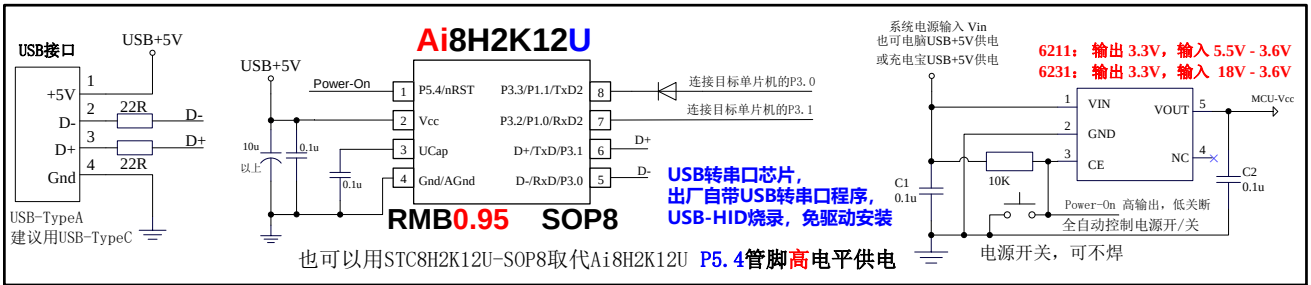
1、点击 电脑端 ISP 软件的【下载/编程】按钮

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，P5.4/RST变成复位脚。

3.4.6.5 管脚图及下载线路图，最小系统（SOP16/DIP16）





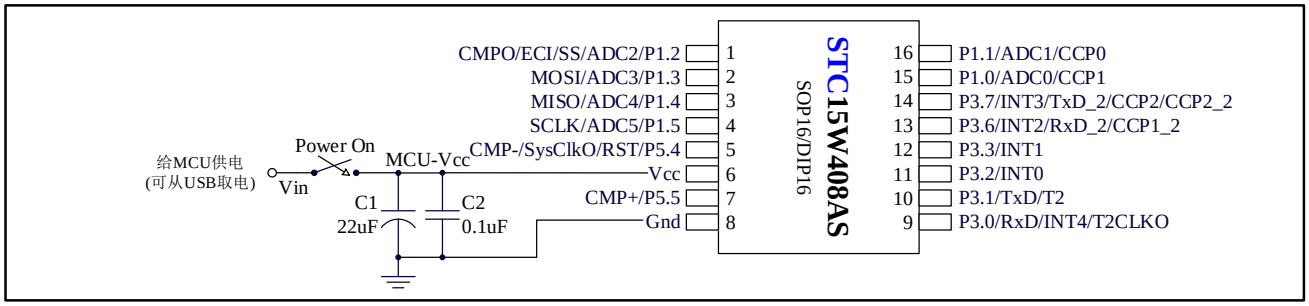
也可以用STC8H2K12U-SOP8取代Ai8H2K12U P5.4管脚高电平供电

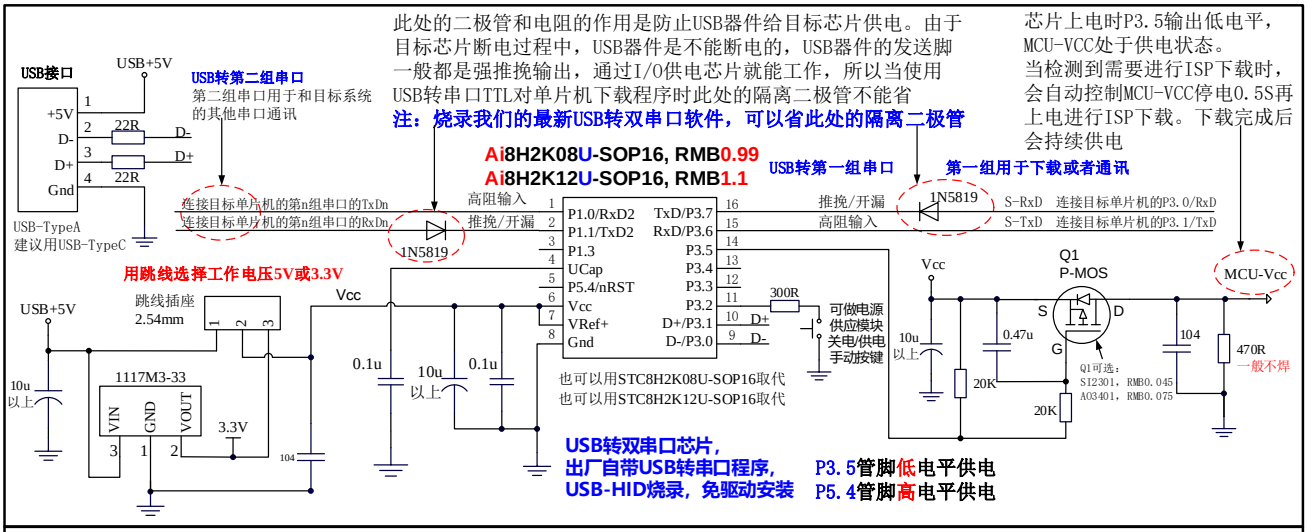
【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

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





【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，P5.4/RST变成复位脚。

3.4.7 管脚说明

管脚	管脚编号				说明	
	SOP28 TSSOP28 SKDIP28	QFN28	SOP20 DIP20 TSSOP20	SOP16 DIP16		
P1.0	3	27	19	15	P1.0	标准 I/O 口 PORT1[0]
					ADC0	ADC 输入通道-0
					CCP1	外部信号捕获（频率测量或当外部中断使用）、 高速脉冲输出及脉宽调制输出通道-1
P1.1	4	28	20	16	P1.1	标准 I/O 口 PORT1[1]
					ADC1	ADC 输入通道-1
					CCP0	外部信号捕获（频率测量或当外部中断使用）、 高速脉冲输出及脉宽调制输出通道-0
P1.2	5	1	1	1	P1.2	标准 I/O 口 PORT1[2]
					ADC2	ADC 输入通道-2
					SS	SPI 同步串行接口的从机选择信号
					ECI	CCP / PCA 计数器的外部脉冲输入脚
					CMPO	比较器的比较结果输出管脚
P1.3	6	2	2	2	P1.3	标准 I/O 口 PORT1[3]
					ADC3	ADC 输入通道-3
					MOSI	SPI 同步串行接口的主出从入（主器件的输出和从器件的输入）
P1.4	7	3	3	3	P1.4	标准 I/O 口 PORT1[4]
					ADC4	ADC 输入通道-4
					MISO	SPI 同步串行接口的主入从出（主器件的输入和从器件的输出）
P1.5	8	4	4	4	P1.5	标准 I/O 口 PORT1[5]
					ADC5	ADC 输入通道-5
					SCLK	SPI 同步串行接口的时钟信号
P1.6	9	5	5		P1.6	标准 I/O 口 PORT1[6]
					ADC6	ADC 输入通道-6
					RxD_3	串口数据接收端
					XTAL2	内部时钟电路反相放大器的输出端，接外部晶振的其中一端。 当直接使用外部时钟源时，此引脚可浮空，此时 XTAL2 实际将 XTAL1 输入的时钟进行输出。
					SysClkO_2	系统时钟输出（输出的频率可为 SysClk/1，SysClk/2，SysClk/4） 系统时钟是指对主时钟进行分频后供给 CPU、定时器的实际工作时钟；主时钟可以是内部 R/C 时钟，也可以是外部输入的时钟或外部晶体振荡产生的时钟；SysClk 是指系统时钟频率。
P1.7	10	6	6		P1.7	标准 I/O 口 PORT1[7]
					ADC7	ADC 输入通道-7
					TxD_3	串口数据发送端
					XTAL1	内部时钟电路反相放大器输入端，接外部晶振的其中一端。 当直接使用外部时钟源时，此引脚是外部时钟源的输入端。
P2.0	23	19			P2.0	标准 I/O 口 PORT2[0]
					RSTOUT_LOW	上电后，输出低电平，在复位期间也是输出低电平。 用户可用软件将其设置为高电平或低电平，如果要读外部状态，可将该口先置高后再读
P2.1	24	20			P2.1	标准 I/O 口 PORT2[1]
					SCLK_2	SPI 同步串行接口的时钟信号

管脚	管脚编号				说明	
	SOP28 TSSOP28 SKDIP28	QFN28	SOP20 DIP20 TSSOP20	SOP16 DIP16		
P2.2	25	21			P2.2	标准 I/O 口 PORT2[2]
					MISO_2	SPI 同步串行接口的主入从出（主器件的输入和从器件的输出）
P2.3	26	22			P2.3	标准 I/O 口 PORT2[3]
					MOSI_2	SPI 同步串行接口的主出从入（主器件的输出和从器件的输入）
P2.4	27	23			P2.4	标准 I/O 口 PORT2[4]
					ECI_3	CCP / PCA 计数器的外部脉冲输入脚
					SS_2	SPI 同步串行接口的从机选择信号
P2.5	28	24			P2.5	标准 I/O 口 PORT2[5]
					CCP0_3	外部信号捕获（频率测量或当外部中断使用）、 高速脉冲输出及脉宽调制输出通道-0
P2.6	1	25			P2.6	标准 I/O 口 PORT2[6]
					CCP1_3	外部信号捕获（频率测量或当外部中断使用）、 高速脉冲输出及脉宽调制输出通道-1
P2.7	2	26			P2.7	标准 I/O 口 PORT2[7]
					CCP2_3	外部信号捕获（频率测量或当外部中断使用）、 高速脉冲输出及脉宽调制输出通道-2
P3.0	15	11	11	9	P3.0	标准 I/O 口 PORT3[0]
					RxD	串口数据接收端
					INT4	外部中断 4，只能下降沿中断， INT4 支持掉电唤醒
					T2CLKO	T2 的时钟输出 可通过设置 INT_CLKO[2]位/T2CLKO 将该管脚 配置为 T2CLKO
P3.1	16	12	12	10	P3.1	标准 I/O 口 PORT3[1]
					TxD	串口数据发送端
					T2	定时器/计数器 2 的外部输入
P3.2	17	13	13	11	P3.2	标准 I/O 口 PORT3[2]
					INT0	外部中断 0，既可上升沿中断也可下降沿中断 如果 IT0（TCON.0）被置为 1，INT0 管脚仅为下降沿中断。 如果 IT0（TCON.0）被清 0，INT0 管脚既支持上升沿中断也支持下降沿中断。 INT0 支持掉电唤醒。
P3.3	18	14	14	12	P3.3	标准 I/O 口 PORT3[3]
					INT1	外部中断 1，既可上升沿中断也可下降沿中断 如果 IT1（TCON.2）被置为 1，INT1 管脚仅为下降沿中断。 如果 IT1（TCON.2）被清 0，INT1 管脚既支持上升沿中断也支持下降沿中断。 INT1 支持掉电唤醒。
P3.4	19	15	15		P3.4	标准 I/O 口 PORT3[4]
					T0	定时器/计数器 0 的外部输入
					ECI_2	CCP/PCA 计数器的外部脉冲输入脚
P3.5	20	16	16		P3.5	标准 I/O 口 PORT3[5]
					T0CLKO	定时器/计数器 0 的时钟输出 可通过设置 INT_CLKO[0]位/T0CLKO 将该管脚配置为 T0CLKO，也可对 T0 脚的外部时钟输入进行分频输出
					CCP0_2	外部信号捕获（频率测量或当外部中断使用）、 高速脉冲输出及脉宽调制输出通道-0

管脚	管脚编号				说明	
	SOP28 TSSOP28 SKDIP28	QFN28	SOP20 DIP20 TSSOP20	SOP16 DIP16		
P3.6	21	17	17	13	P3.6	标准 I/O 口 PORT33[6]
					INT2	外部中断 2，只能下降沿中断 INT2 支持掉电唤醒
					RxD_2	串口数据接收端
					CCP1_2	外部信号捕获（频率测量或当外部中断使用）、 高速脉冲输出及脉宽调制输出通道-1
P3.7	22	18	18	14	P3.7	标准 I/O 口 PORT3[7]
					INT3	外部中断 3，只能下降沿中断 INT3 支持掉电唤醒
					TxD_2	串口数据发送端
					CCP2	外部信号捕获（频率测量或当外部中断使用）、 高速脉冲输出及脉宽调制输出通道-2
					CCP2_2	外部信号捕获（频率测量或当外部中断使用）、 高速脉冲输出及脉宽调制输出通道-2
P5.4	11	7	7	5	P5.4	标准 I/O 口 PORT5[4]
					RST	复位脚（高电平复位）
					SysClkO	系统时钟输出（输出的频率可为 SysClk/1，SysClk/2，SysClk/4） 系统时钟是指对主时钟进行分频后供给 CPU、定时器的实际工作时钟；主时钟可以是内部 R/C 时钟，也可以是外部输入的时钟或外部晶体振荡产生的时钟；SysClk 是指系统时钟频率。
					CMP-	比较器负极输入端 （若该口被用作比较器负极，则该口需被设置为高阻输入）
P5.5	13	9	9	7	P5.5	标准 I/O 口 PORT5[5]
					CMP+	比较器正极输入端 （若该口被用作比较器正极，则该口需被设置为高阻输入）
Vcc	12	8	8	6	电源正极	
Gnd	14	10	10	8	电源负极，接地	

3.4.8 USB-Link1D 工具自动停电/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】

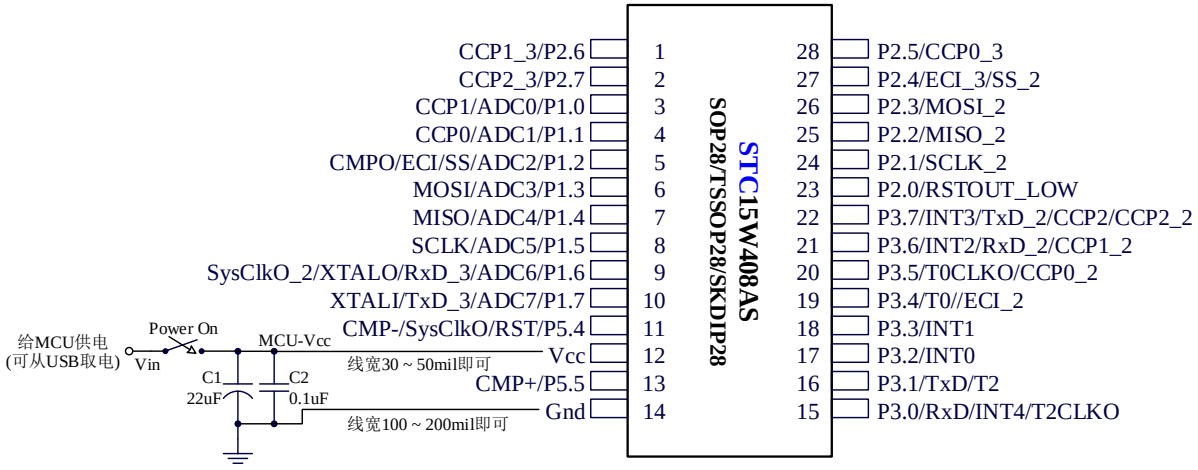
点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【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，P5.4/nRST变成复位脚



3.4.9 【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】

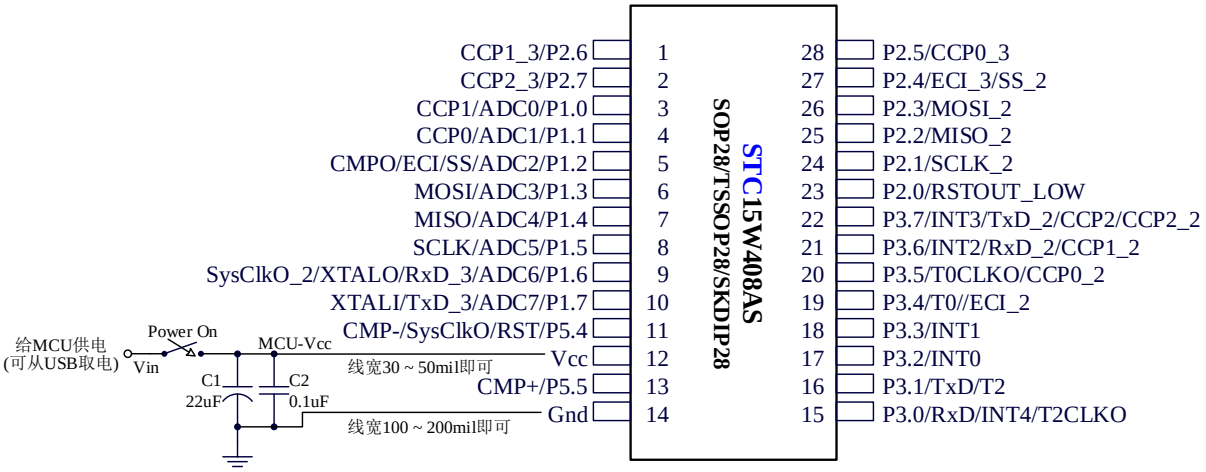
点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【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，P5.4/nRST变成复位脚



3.5 STC15F2K60S2 系列

STC15F2K60S2 系列 1T 8051 单片机，2K 字节 SRAM，超高速双串口，高速 A/D

不需外部晶振的单片机，不需外部复位的单片机

STC15F2K60S2 系列主要性能：

- ✓ 大容量 2048 字节片内 RAM 数据存储器
- ✓ 高速：1 个时钟/机器周期，增强型 8051 内核，速度比传统 8051 快 7 ~ 12 倍
  速度也比 STC 早期的 1T 系列单片机（如 STC12/11/10 系列）的速度快 20%
- ✓ 宽电压：5.5 ~ 4.2V，2.4 ~ 3.6V（STC15L2K60S2 系列）
- ✓ 低功耗设计：低速模式，空闲模式，掉电模式（可由外部中断或内部掉电唤醒定时器唤醒）
- ✓ 不需外部复位的单片机，ISP 编程时 8 级复位门槛电压可选，内置高可靠复位电路
- ✓ 不需外部晶振的单片机，ISP 编程时内部时钟从 5MHz ~ 28MHz 可设（相当于普通 8051：60 ~ 336MHz）
  内部高精度 R/C 时钟（±0.3%），+1%温漂（-40℃ ~ +85℃），常温下温漂±0.6%（-20℃ ~ +65℃）
- ✓ 支持掉电唤醒的资源有：INT0/INT1（上升沿/下降沿中断均可），INT2/INT3/INT4（下降沿中断）；
  CCP0/CCP1/CCP2/T0/T1/T2 管脚；内部掉电唤醒专用定时器
- ✓ 8/16/24/32/40/48/56/60/61/63.5K 字节片内 Flash 程序存储器，擦写次数 10 万次以上
- ✓ 大容量片内 EEPROM 功能，擦写次数 10 万次以上
- ✓ ISP/IAP，在系统可编程/在应用可编程，无需编程器/仿真器
- ✓ 高速 ADC，8 通道 10 位，速度可达 30 万次/秒，3 路 PWM 还可当 3 路 D/A 使用
- ✓ 3 通道捕获/比较单元（CCP/PCAPWM）
  ----也可用来再实现 3 路 D/A 或 3 个定时器或 3 个外部中断（支持上升沿/下降沿中断）
- ✓ 利用 CCP/PCA 高速脉冲输出功能可实现 3 路 9-16 位 PWM（每通道占用系统时间小于 0.6%）
- ✓ 利用定时器 T0、T1 或 T2 的时钟输出功能可实现高精度的 8-16 位 PWM（占用系统时间小于 0.4%）
- ✓ 6 个定时器，2 个 16 位可重装载定时器 T0 和 T1 兼容普通 8051 的定时器，新增了一个 16 位的定时器 T2，并都可实现可编程时钟输出，另外管脚 MCLKO 可将内部主时钟对外分频输出（÷1 或÷2 或÷4），3 路 CCP/PCA 可再实现 3 个定时器
- ✓ 可编程时钟输出功能（对内部系统时钟或外部管脚的时钟输入进行时钟分频输出）：
  - ➢ T0 在 P3.5 输出时钟；
  - ➢ T1 在 P3.4 输出时钟；
  - ➢ T2 在 P3.0 输出时钟。
    以上 3 个定时器/计数器输出时钟均可 1 ~ 65536 级分频输出；
  - ➢ 内部主时钟在 P5.4/MCLKO 对外输出时钟（STC15 系列 8-pin 单片机的主时钟在 P3.4/MCLKO 对外输出时钟）
- ✓ 超高速双串口/UART，两个完全独立的高速异步串行通信端口，分时切换可当 5 组串口使用
- ✓ SPI 高速同步串行通信接口
- ✓ 硬件看门狗（WDT）

- ✓ 先进的指令集结构，兼容普通 8051 指令集，有硬件乘法/除法指令
- ✓ 通用 I/O 口（42/38/30/26/18 个），复位后为：准双向口/上拉（8051 传统 I/O 口）
  可设置四种模式：准双向口/弱上拉，强推挽/强上拉，仅为输入/高阻，开漏
  每个 I/O 口驱动能力均可达到 20mA，但整个芯片最大不要超过 120mA
- ✓ 如果 I/O 口不够用可以用 3 根普通 I/O 端口外接 74HC595（¥0.15 元）来扩展 I/O 口，并可多芯片级联扩展几十个 I/O 口，还可用 A/D 作按键扫描来节省 I/O 口

选择 STC15F2K60S2 系列单片机理由：

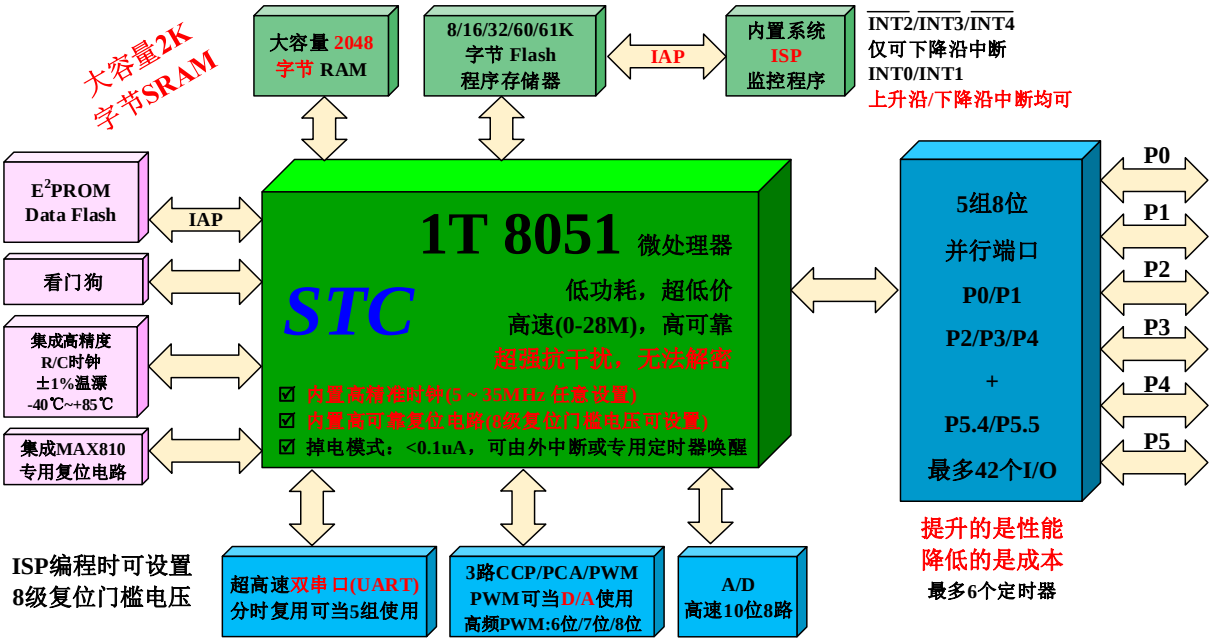
- ★ 不需外部晶振和外部复位，还可对外输出时钟和低电平复位信号
- ★ 片内大容量 2048 字节 SRAM
- ★ 无法解密，STC 第八代加密技术，现悬赏 10 万元人民币请专家帮忙查找加密有无漏洞
- ★ 超强抗干扰
  - ➢ 高抗静电（ESD 保护）整机轻松过 2 万伏静电测试
  - ➢ 轻松过 4kV 快速脉冲干扰（EFT 测试）
  - ➢ 宽电压，不怕电源抖动
  - ➢ 宽温度范围，-40℃ ~ +85℃
- ★ 大幅降低 EMI，内部可配置时钟，1 个时钟/机器周期，可用低频时钟
  ----出口欧美的有力保证
- ★ 超低功耗：
  - ➢ 掉电模式：外部中断唤醒功耗<0.1uA
  - ➢ 空闲模式：典型功耗<1mA
  - ➢ 正常工作模式：4mA ~ 6mA
  - ➢ 掉电模式可由外部中断或内部掉电唤醒专用定时器唤醒，适用于电池供电系统，如水表、气表等
- ★ 在系统可仿真，在系统可编程，无需专用编程器，无需专用仿真，可远程升级
- ★ 可送 USB 型联机/脱机下载烧录工具 STC-U8W（人民币 100 元），1 万片/人/天，有自动烧录机接口

3.5.1 STC15F2K60S2 系列简介

STC15F2K60S2 系列单片机是 STC 生产的单时钟/机器周期（1T）的单片机，是高速/高可靠/低功耗/超强抗干扰的新一代 8051 单片机，采用 STC 第八代加密技术，无法解密，指令代码完全兼容传统 8051，但速度快 8-12 倍。内部集成高精度 R/C 时钟（±0.3%），±1%温漂（-40℃ ~ +85℃），常温下温漂+0.6%（-20℃ ~ +65℃），ISP 编程时 5MHz ~ 35MHz 宽范围可设置，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 8 级复位门槛电压可选）。3 路 CCP/PWM/PCA，8 路高速 10 位 A/D 转换（30 万次/秒），内置 2K 字节大容量 SRAM，2 组超高速步串行通信端口（UART1/UART2，可在 5 组管脚之间进行切换，分时复用可作 5 组串口使用），1 组高速同步串行通信端口 SPI，针对多串行口通信/电机控制干扰场合。

在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可

现 STC15 系列单片机采用 STC-Y5 超高速 CPU 内核，在相同的时钟频率下，速度又比 STC 早期的 1T 系列单片机（如 STC12 系列/STC11 系列/STC10 系列）的速度快 20%。



1. 增强型 8051 CPU，1T，单时钟/机器周期，速度比普通 8051 快 8-12 倍
2. 工作电压：STC15F2K60S2 系列工作电压：5.5V-4.5V（5V 单片机）
   STC15L2K60S2 系列工作电压：3.6V-2.4V（3V 单片机）
3. 8K/16K/24K/32K/40K/48K/56K/60K/61K/63.5K 字节片内 Flash 程序存储器，可擦写次数 10 万次以上
4. 片内大容量 2048 字节的 SRAM，包括常规的 256 字节 RAM <idata>和内部扩展的 1792 字节 XRAM <xdata>
5. 大容量片内 EEPROM，擦写次数 10 万次以上
6. ISP/IAP，在系统可编程/在应用可编程，无需编程器，无需仿真器
7. 共 8 通道 10 位高速 ADC，速度可达 30 万次/秒，3 路 PWM 还可当 3 路 D/A 使用
8. 共 3 通道捕获/比较单元（CCP/PWM/PCA）
   ----也可用来再实现 3 个定时器或 3 个外部中断（支持上升沿/下降沿中断）或 3 路 D/A
9. 利用 CCP/PCA 高速脉冲输出功能可实现 3 路 9 ~ 16 位 PWM（每通道占用系统时间小于 0.6%）

10. 利用定时器 T0、T1 或 T2 的时钟输出功能可实现高精度的 8 ~ 16 位 PWM（占用系统时间小于 0.4%）
11. 内部高可靠复位，ISP 编程时 8 级复位门槛电压可选，可彻底省掉外部复位电路
12. 工作频率范围：0MHz ~ 28MHz，相当于普通 8051 的 0MHz ~ 336MHz
13. 内部高精度 R/C 时钟（±0.3%），±1%温漂（-40℃ ~ +85℃），常温下温漂±0.6%（-20℃ ~ +65℃），ISP 编程时内部时钟从 5MHZ ~ 28MHz 可设（5.5296MHz / 11.0592MHz / 22.1184MHz）
14. 不需外部晶振和外部复位，还可对外输出时钟和低电平复位信号
15. 两组超高速异步串行通信端口（可同时使用），可在 5 组管脚之间进行切换，分时复用可当 5 组串口使用：

    串口 1（RxD/P3.0，TxD/P3.1）可以切换到（RxD_2/P3.6，TxD_2/P3.7），
    还可以切换到（RxD_3/P1.6，TxD_3/P1.7）；
    串口 2（RxD2/P1.0，TxD2/P1.1）可以切换到（RxD2_2/P4.6，TxD2_2/P4.7）

    注意：建议用户将串口 1 放在 P3.6/P3.7 或 P1.6/P1.7（P3.0/P3.1 作下载/仿真用）；若用户不想切换，坚持使用 P3.0/P3.1 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”。
16. 一组高速异步串行通信端口 SPI
17. 支持程序加密后传输，防拦截
18. 支持 RS485 下载
19. 低功耗设计：低速模式，空闲模式，掉电模式/停机模式
20. 可将掉电模式/停机模式唤醒的定时器：有内部低功耗掉电唤醒专用定时器。
21. 可将掉电模式/停机模式唤醒的资源有：
    - INT0/P3.2，INT1/P3.3（INT0/INT1 上升沿下降沿中断均可），INT2/P3.6，INT3/P3.7，INT4/P3.0（INT2/INT3/INT4 仅可下降沿中断）；
    - 管脚 CCP0/CCP1/CCP2；
    - 管脚 T0/T1/T2（下降沿，不产生中断，前提是在进入掉电模式/停机模式前相应的定时器中断已经被允许）；
    - 内部低功耗掉电唤醒专用定时器。
22. 共 6 个定时器/计数器，3 个 16 位可重装载定时器/计数器（T0/T1/T2，其中 T0/T1 兼容普通 8051 的定时器/计数器），并均可独立实现对外可编程时钟输出（3 通道），另外管脚 MCLKO 可将内部主时钟对外分频输出（÷1 或÷2 或÷4），3 路 CCP/PWM/PCA 还可再实现 3 个定时器。
23. 定时器/计数器 T2，也可实现 1 个 16 位重装载定时器/计数器，T2 也可产生时钟输出 T2CLKO。
24. 可编程时钟输出功能（对内部系统时钟或对外部管脚的时钟输入进行时钟分频输出）：
    - 由于 STC15 系列 5V 单片机 I/O 口的对外输出速度最快不超过 13.5MHz，所以 5V 单片机的对外可编程时钟输出速度最快也不超过 13.5MHz.；
    - 而 3.3V 单片机 I/O 口的对外输出速度最快不超过 8MHz，故 3.3V 单片机的对外可编程时钟输出速度最快也不超过 8MHZ.

    1) T0 在 P3.5/T0CLKO 进行可编程输出时钟（对内部系统时钟或对外部管脚 T0/P3.4 的时钟输入进行可编程时钟分频输出）；
    2) T1 在 P3.4/T1CLKO 进行可编程输出时钟（对内部系统时钟或对外部管脚 T1/P3.5 的时钟输入进行可编程时钟分频输出）；
    3) T2 在 P3.0/T2CLKO 进行可编程输出时钟（对内部系统时钟或对外部管脚 T2/P3.1 的时钟输入进

行可编程时钟分频输出）；

以上 3 个定时器/计数器均可 1 ~ 65536 级分频输出。

4) 主时钟在 P5.4/MCLKO 对外输出时钟，并可如下分频 MCLK/1，MCLK/2，MCLK/4

现供货的 STC15F2K60S2 系列 C 版本单片机主时钟对外输出管脚 P5.4/MCLKO 只可以对外输出内部 R/C 时钟，但是其他可外接外部晶体的 STC15 系列单片机主时钟对外输出管脚 P5.4/MCLKO 既可以对外输出内部 R/C 时钟，也可对外输出外部输入的时钟或外部晶体振荡产生的时钟。STC15F2K60S2 系列下一升级版本--STC15W2K60S2 系列单片机将同其他系列单片机一样，其主时钟对外输出管脚 P5.4/MCLKO 既可以对外输出内部 R/C 时钟，也可对外输出外部输入的时钟或外部晶体振荡产生的时钟。

上述 MCLK 是指主时钟频率，MCLKO 是指主时钟输出。

STC15 系列 8-pin 单片机（如 STC15F104W 系列）在 MCLKO/P3.4 口对外输出时钟，STC15 系列 16-pin 及其以上单片机（如 STC15W4K32S4 系列和 STC15F2K60S2 系列等）均在 MCLKO/P5.4 口对外输出时钟，且 STC15W 系列 20-pin 及其以上单片机除可在 MCLKO/P5.4 口对外输出时钟外，还可在 MCLKO_2/P1.6 口对外输出时钟。

25. 硬件看门狗（WDT）

26. 先进的指令集结构，兼容普通 8051 指令集，有硬件乘法/除法指令

27. 通用 I/O 口（42/38/30/26 个），复位后为：准双向口/弱上拉（普通 8051 传统 I/O 口）

    可设置成四种模式：准双向口上拉，强推挽上拉，仅为输入/高阻，开漏
    每个 I/O 口驱动能力均可达到 20mA，但 40-pin 及 40-pin 以上单片机的整个芯片电流最大不要超过 120mA，16-pin 及以上/32-pin 及以下单片机的整个芯片电流最大不要超过 90mA.
    如果 I/O 口不够用，可外接 74HC595（参考价 0.15 元）来扩展 I/O 口，并可多芯片级联扩展几十个 I/O 口

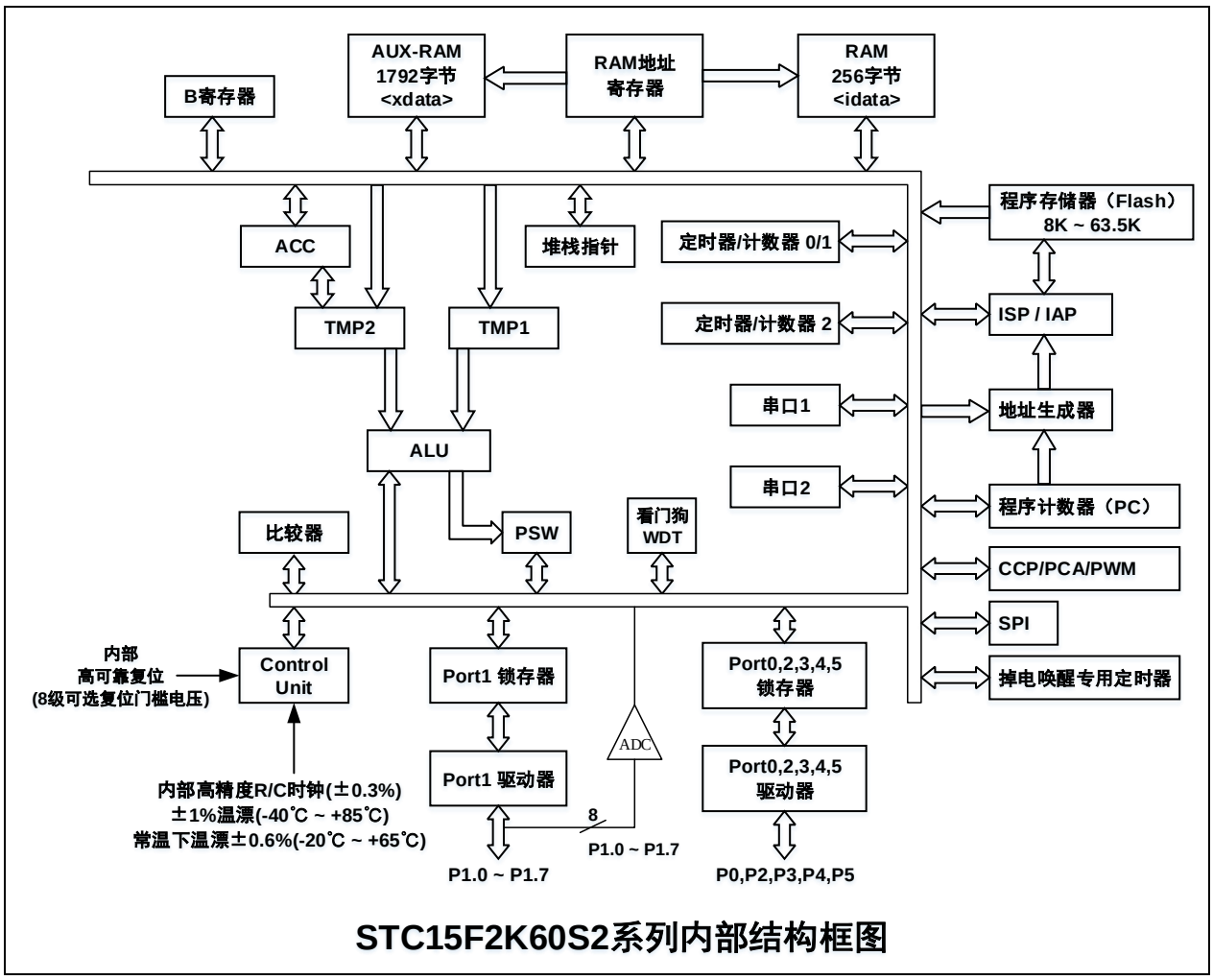
28. 封装：LQFP44（12mm×12mm），LQFP32（9mm×9mm），QFN32（5mm×5mm）
    TSSOP20（6.5mm×6.5mm），SOP28，SKDIP28，PDIP40，

29. 全部 175℃八小时高温烘烤，高品质制造保证

30. 开发环境：在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可

3.5.2 内部结构图

STC15F2K60S2 系列单片机的内部结构框图如下图所示。STC15F2K60S2 系列单片机中包含中央处理器（CPU）、程序存储器（Flash）、数据存储器（SRAM）、定时器、I/O 口、高速 A/D 转换、看门狗、UART 超高速异步串行通信口 1/串行通信口 2，CCP/PWM/PCA，1 组高速同步串行端口 SPI，片内高精度 R/C 时钟及高可靠复位等模块。STC15F2K60S2 系列单片机几乎包含了数据采集和控制中所需的所有单元模块，可称得上是一个片上系统（SysTem Chip 或 SysTem onChip，简写为 STC，这是 STC 名称的由来）。



STC15F2K60S2系列内部结构框图

3.5.3 STC15F2K60S2 系列单片机选型价格一览表

型号	工作电压（V）	Flash 程序存储器 (byte)	大容量 SRAM 字节	串口	SPI 仅有主机模式	普通定时器 计数器 T0-T2 外部管脚也能掉电唤醒	CCP PCA PWM 可当外部中断并可掉电唤醒	掉电唤醒专用定时器	标准外部中断支持掉电唤醒	A/D 8 路 (3 路 PWM 可当 3 路 D/A 使用)	DPTR	EEPROM	内部低压检测中断并可掉电唤醒	看门狗	内部可靠复位 (可选复位门槛电压)	内部高精准时钟	可对外输出时钟及复位	程序加密后传输（防拦截）	可设下次更新程序需口令	支持 RS485 下载	所有封装 LQFP44/PDIP40 LQFP32/QFN32 SOP28/SKDIP28 TSSOP20 (现此系列未生产 PLCC44 和 SOP32 封装)						
																					部分封装价格(RMB ¥)						
																					TSSOP 20	SOP 28	SKDIP 28	LQFP 32	QFN 32	PDIP 40	LQFP 44
大批量 现货供应中	STC15F2K60S2 系列单片机选型价格一览表，另有 STC15L 系列(工作电压 2.4V-3.6V) 特别提醒：3 路 CCP/PCA/PWM 还可当 3 路定时器使用																									
STC15F2K08S2	5.5-4.5	8K	2K	2	有	3	3-ch	有	5	10 位	2	53K	有	有	8 级	有	是	有	是	是	-	¥4.40	¥4.60	¥4.30	-	¥5.20	¥4.50
STC15F2K16S2	5.5-4.5	16K	2K	2	有	3	3-ch	有	5	10 位	2	45K	有	有	8 级	有	是	有	是	是	-	¥4.40	¥4.60	¥4.30	-	¥5.20	¥4.50
STC15F2K32S2	5.5-4.5	32K	2K	2	有	3	3-ch	有	5	10 位	2	29K	有	有	8 级	有	是	有	是	是	-	¥4.40	¥4.60	¥4.30	-	¥5.20	¥4.50
STC15F2K40S2	5.5-4.5	40K	2K	2	有	3	3-ch	有	5	10 位	2	21K	有	有	8 级	有	是	有	是	是	-	¥4.40	¥4.60	¥4.30	-	¥5.20	¥4.50
STC15F2K48S2	5.5-4.5	48K	2K	2	有	3	3-ch	有	5	10 位	2	13K	有	有	8 级	有	是	有	是	是	-	¥4.40	¥4.60	¥4.30	-	¥5.20	¥4.50
STC15F2K56S2	5.5-4.5	56K	2K	2	有	3	3-ch	有	5	10 位	2	5K	有	有	8 级	有	是	有	是	是	-	¥4.40	¥4.60	¥4.30	-	¥5.20	¥4.50
STC15F2K60S2	5.5-4.5	60K	2K	2	有	3	3-ch	有	5	10 位	2	1K	有	有	8 级	有	是	有	是	是	-	¥4.40	¥4.60	¥4.30	-	¥5.20	¥4.50
IAP15F2K61S2 本身就是仿真器	5.5-4.5	61K	2K	2	有	3	3-ch	有	5	10 位	2	IAP	有	有	8 级	有	是	有	是	是	-	¥4.40	¥4.60	¥4.30	¥4.50	¥5.20	¥4.50
																					用户可将用户程序区的程序 Flash 当 EEPROM 使用						
IRC15F2K63S2 默认使用外部晶振如无外部晶振则使用内部 24MHz 时钟	5.5-4.5	63.5K	2K	2	有	3	3-ch	有	5	10 位	2	IAP	有	有	固定	有	是	无	否	否	-	-	-	-	-	¥5.20	¥4.50
																					用户可将用户程序区的程序 Flash 当 EEPROM 使用						
STC15F2K32S	5.5-4.5	32K	2K	1	有	3	-	有	5	-	-	29K	有	有	8 级	有	是	有	是	是	-	-	-	-	-	¥5.20	¥4.50
STC15F2K60S	5.5-4.5	60K	2K	1	有	3	-	有	5	-	-	1K	有	有	8 级	有	是	有	是	是	-	-	-	-	-	¥5.20	¥4.50
IAP15F2K61S	5.5-4.5	61K	2K	1	有	3	-	有	5	-	2	IAP	有	有	8 级	有	是	有	是	是	-	-	-	-	-	¥5.20	¥4.50
																					用户可将用户程序区的程序 Flash 当 EEPROM 使用						
STC15F2K24AS	5.5-4.5	24K	2K	1	有	3	3-ch	有	5	10 位	-	37K	有	有	8 级	有	是	有	是	是	-	-	-	-	-	-	¥4.50
STC15F2K48AS	5.5-4.5	48K	2K	1	有	3	3-ch	有	5	10 位	-	13K	有	有	8 级	有	是	有	是	是	-	-	-	-	-	-	¥4.50
大批量 现货供应中	STC15F104W 系列单片机选型价格一览表， 另有 STC15L 系列(工作电压 2.4V-3.6V)串行口功能可由[P3.0/INT4, P3.1]结合定时器实现																				SOP8	DIP8	DFN8	-	-	-	-
STC15F100W	5.5-3.8	0.5K	128	-	-	-	-	有	5	-	1	-	有	有	8 级	有	是	有	是	是	¥1.20	¥1.40	¥1.60	-	-	-	-
STC15F101W	5.5-3.8	1K	128	-	-	-	-	有	5	-	1	4K	有	有	8 级	有	是	有	是	是	¥1.20	¥1.40	¥1.60	-	-	-	-
STC15F102W	5.5-3.8	2K	128	-	-	-	-	有	5	-	1	3K	有	有	8 级	有	是	有	是	是	¥1.20	¥1.40	¥1.60	-	-	-	-
STC15F104W	5.5-3.8	4K	128	-	-	-	-	有	5	-	1	1K	有	有	8 级	有	是	有	是	是	¥1.20	¥1.40	¥1.60	-	-	-	-
IRC15F107W 默认使用内部 24MHz 时钟	5.5-3.8	7K	128	-	-	-	-	有	5	-	1	IAP	有	有	固定	有	是	有	否	否	¥1.40	¥1.60	-	-	-	-	-
																					用户可将用户程序区的程序 Flash 当 EEPROM 使用						

我们直销，所以低价，以上单价为 10K/M 起定量，量小每片需加 0.1 元，以上价格运费由客户承担，零售 10 片起，如对价格不满，可来电要求降价。

程序加密后传输：程序拥有者产品出厂时将源程序和加密钥匙一起烧录 MCU 中，以后需要升级软件时，就可将程序加密后再用"发布项目程序"功能，生成一个用户自己界面的只有一个升级按钮的简单易用的升级软件，给最终使用者自己升级，而拦截不到您的原始程序。

STC15F/L2K60S2 系列只有固件版本为 Ver7.2（成功烧录程序时在 AIapp-ISP 软件界面右下角查询该固件

版本号）及其以上的单片机才支持"可设下次更新程序需口令"功能。

注意：因为程序区的最后 7 个字节单元被强制性的放入全球唯一 1D 号的内容，所以用户实际可以使用的程序空间大小要比选型表中的大小少 7 个字节。

IRC15F2K63S2 和 IRC15L2K63S2 型号单片机的内部复位门槛电压固定，P5.4 不可当复位管脚 RST 使用，[XTAL2/P1.6，XTAL1/P1.7]不可当 I/O 口使用，P3.2/P3.3 与下载无关，且不支持"程序加密后传输"功能。

特别声明：以 15F 和 15L 开头且有 SPI 功能的芯片，只支持"SPI 主机模式"，不支持"SPI 从机模式"；以 15W 开头且有 SPI 功能的芯片，SPI 主/从机模式均支持。

特别声明：以 15L 开头的 C 版本芯片如需进入"掉电模式"，进入"掉电模式"前必须启动掉电唤醒定时器<3uA>，不超过 1 秒要唤醒一次，以 15F 和 15W 开头的芯片以及最新的以 15L 开头的 D 版本芯片则不需要。

【总结】:

- STC15F/L2K60S2 系列单片机（含 IRC15F2K63S2 型号单片机）有：
  - ✓ 3 个 16 位可重装载普通定时器/计数器，这 3 个普通定时器/计数器分别是定时器/计数器 0、定时器/计数器 1 和定时器/计数器 2；
  - ✓ 3 路 CCP/PWM/PCA（可再实现 3 个定时器或 3 个 D/A 转换器）；
  - ✓ 掉电唤醒专用定时器；
  - ✓ 5 个外部中断 INT0/INT1/INT2/INT3/INT4；
  - ✓ 2 组高速异步串行口（可同时使用）；
  - ✓ 1 组高速同步串行通信端口 SPI；
  - ✓ 8 路高速 10 位 A/D 转换器；
  - ✓ 2 个数据指针 DPTR；
  - ✓ 外部数据总线等功能。
- IAP15F/L2K61S 型号单片机有：
  - ✓ 3 个 16 位可重装载普通定时器/计数器，这 3 个普通定时器/计数器分别是定时器/计数器 0、定时器/计数器 1 和定时器/计数器 2；
  - ✓ 掉电唤醒专用定时器；
  - ✓ 5 个外部中断 INT0/INT1/INT2/INT3/INT4；
  - ✓ 1 组高速异步串行口；
  - ✓ 1 组高速同步串行通信端口 SPI；
  - ✓ 2 个数据指针 DPTR；
  - ✓ 外部数据总线等功能。
- STC15F2K24AS 型号单片机有：
  - ✓ 3 个 16 位可重装载普通定时器/计数器，这 3 个普通定时器/计数器分别是定时器/计数器 0、定时器/计数器 1 和定时器/计数器 2；
  - ✓ 3 路 CCP/PWMPCA（可再实现 3 个定时器或 3 个 D/A 转换器）；
  - ✓ 掉电唤醒专用定时器；
  - ✓ 5 个外部中断 INT0/INT1/INT2/INT3/INT4；
  - ✓ 1 组高速异步串行口；
  - ✓ 1 组高速同步串行通信端口 SPI；
  - ✓ 8 路高速 10 位 A/D 转换器；
  - ✓ 2 个数据指针 DPTR；
  - ✓ 外部数据总线等功能。

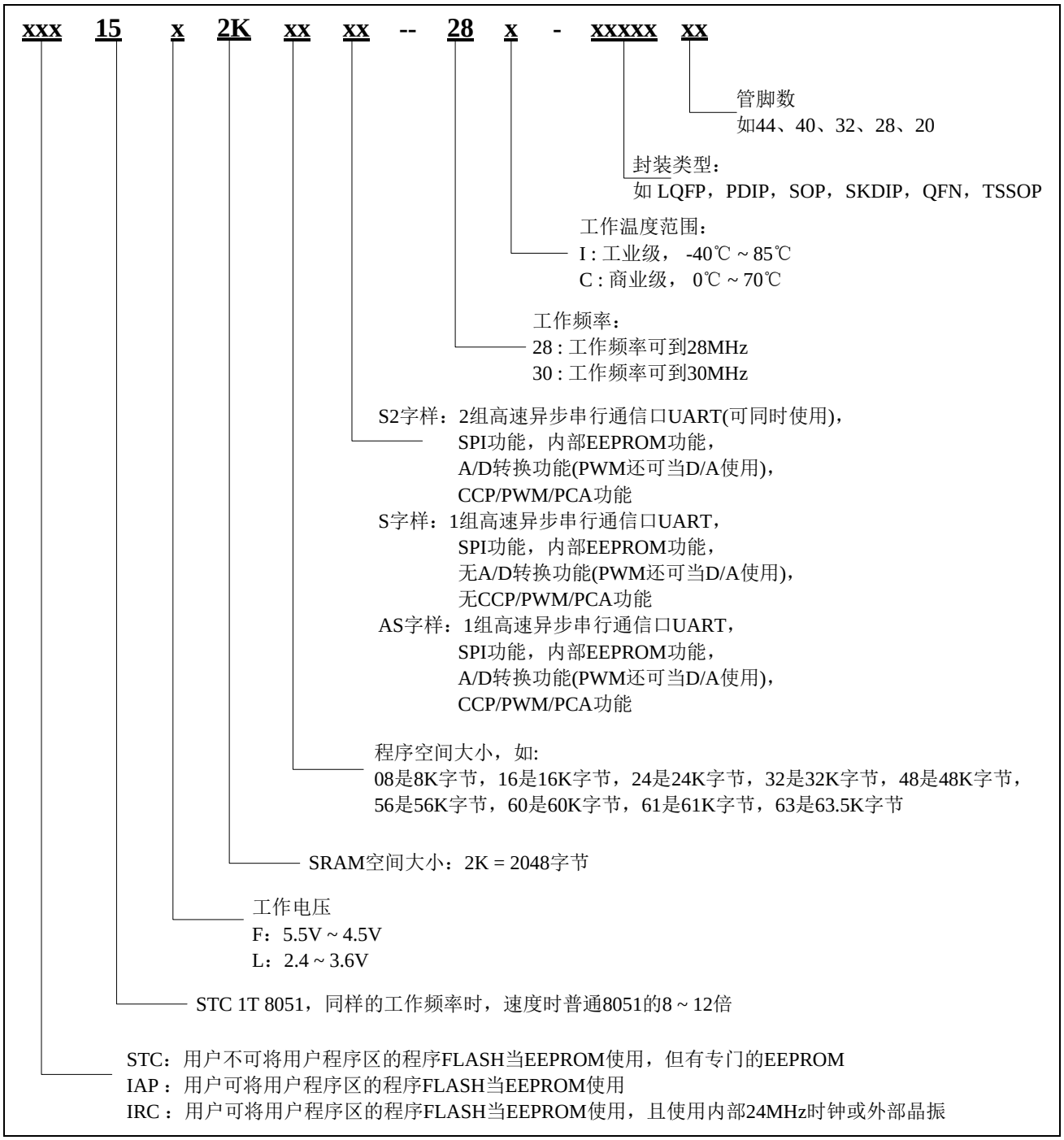
3.5.4 STC15F2K60S2 系列单片机封装价格一览表

型号	工作电压（V）	工作频率(MHz)	工作温度 (I -- 工业级)	所有封装价格(RMB ¥) LQFP44 / PDIP40 / LQFP32 / QFN32 / SOP28 / SKDIP28 / TSSOP20						
				TSSOP20 (18 个 I/O)	SOP28 (26 个 I/O)	SKDIP28 (26 个 I/O)	LQFP32 (30 个 I/O)	QFN32 (30 个 I/O)	PDIP40 (38 个 I/O)	LQFP44 (42 个 I/O)
STC15F2K60S2 系列单片机封装价格一览表										
STC15F2K08S2	5.5-4.5	28	-40℃ ~ +85℃	-	¥4.40	¥4.60	¥4.30	-	¥5.20	¥4.50
STC15F2K16S2	5.5-4.5	28	-40℃ ~ +85℃	-	¥4.40	¥4.60	¥4.30	-	¥5.20	¥4.50
STC15F2K32S2	5.5-4.5	28	-40℃ ~ +85℃	-	¥4.40	¥4.60	¥4.30	-	¥5.20	¥4.50
STC15F2K40S2	5.5-4.5	28	-40℃ ~ +85℃	-	¥4.40	¥4.60	¥4.30	-	¥5.20	¥4.50
STC15F2K48S2	5.5-4.5	28	-40℃ ~ +85℃	-	¥4.40	¥4.60	¥4.30	-	¥5.20	¥4.50
STC15F2K56S2	5.5-4.5	28	-40℃ ~ +85℃	-	¥4.40	¥4.60	¥4.30	-	¥5.20	¥4.50
STC15F2K60S2	5.5-4.5	28	-40℃ ~ +85℃	-	¥4.40	¥4.60	¥4.30	-	¥5.20	¥4.50
IAP15F2K61S2 本身就是仿真器	5.5-4.5	28	-40℃ ~ +85℃	¥4.40	¥4.40	¥4.60	¥4.30	¥4.50	¥5.20	¥4.50
IRC15F2K63S2 默认使用外部晶振 如无外部晶振则使用内部 24MHz 时钟	5.5-4.5	28	-40℃ ~ +85℃	-	-	-	-	-	¥5.20	¥4.50
STC15F2K32S	5.5-4.5	28	-40℃ ~ +85℃	-	-	-	-	-	¥5.20	¥4.50
STC15F2K60S	5.5-4.5	28	-40℃ ~ +85℃	-	-	-	-	-	¥5.20	¥4.50
IAP15F2K61S	5.5-4.5	28	-40℃ ~ +85℃	-	-	-	-	-	¥5.20	¥4.50
STC15F2K24AS	5.5-4.5	28	-40℃ ~ +85℃	-	-	-	-	-	-	¥4.50
STC15F2K48AS	5.5-4.5	28	-40℃ ~ +85℃	-	-	-	-	-	-	¥4.50
STC15L2K60S2 系列单片机封装价格一览表										
STC15L2K08S2	2.4-3.6	28	-40℃ ~ +85℃	-	¥4.40	¥4.60	¥4.30	-	¥5.20	¥4.50
STC15L2K16S2	2.4-3.6	28	-40℃ ~ +85℃	-	¥4.40	¥4.60	¥4.30	-	¥5.20	¥4.50
STC15L2K24S2	2.4-3.6	28	-40℃ ~ +85℃	-	¥4.40	¥4.60	¥4.30	-	¥5.20	¥4.50
STC15L2K32S2	2.4-3.6	28	-40℃ ~ +85℃	-	¥4.40	¥4.60	¥4.30	-	¥5.20	¥4.50
STC15L2K40S2	2.4-3.6	28	-40℃ ~ +85℃	-	¥4.40	¥4.60	¥4.30	-	¥5.20	¥4.50
STC15L2K48S2	2.4-3.6	28	-40℃ ~ +85℃	-	¥4.40	¥4.60	¥4.30	-	¥5.20	¥4.50
STC15L2K56S2	2.4-3.6	28	-40℃ ~ +85℃	-	¥4.40	¥4.60	¥4.30	-	¥5.20	¥4.50
STC15L2K60S2	2.4-3.6	30	-40℃ ~ +85℃	¥4.40	¥4.40	¥4.60	¥4.30	¥4.50	¥5.20	¥4.50
IAP15L2K61S2 本身就是仿真器	2.4-3.6	28	-40℃ ~ +85℃	-	-	-	-	-	¥5.20	¥4.50
STC15L2K32S	2.4-3.6	28	-40℃ ~ +85℃	-	-	-	-	-	¥5.20	¥4.50
STC15L2K60S	2.4-3.6	28	-40℃ ~ +85℃	-	-	-	-	-	¥5.20	¥4.50
IAP15L2K61S	2.4-3.6	28	-40℃ ~ +85℃	-	-	-	-	-	¥5.20	¥4.50
STC15L2K24AS	2.4-3.6	28	-40℃ ~ +85℃	-	-	-	-	-	-	¥4.50
STC15L2K48AS	2.4-3.6	28	-40℃ ~ +85℃	-	-	-	-	-	-	¥4.50

STC15F2K24AS-LQFP44 及 STC15F2K48AS-LQFP44 由于是无利润产品不零售，100K 起定，小量建议用 STC15W1K16S 或 STC15W408AS 系列取代。

- ✓ 如果用户要用 40-pin 及以上的单片机，建议选用 LQFP44 的封装，但 PDIP40 封装仍正常供货；
- ✓ 如果用户要用 32-pin 单片机，建议用户选用 LQFP32 封装；
- ✓ 如果用户要用 28-pin 单片机，建议用户选用 S0P28 封装。

3.5.5 STC15F2K60S2 系列单片机命名规则



如何识别芯片版本号：如需知道芯片版本号，请查阅芯片表面印刷字中最下面一行的最后一个字母（如 D），该字母代表芯片版本号（如 D 版）

命名举例:

1) STC15F2K60S2-28I-LQFP44 表示：

用户不可以将用户程序区的程序 FLASH 当 EEPROM 使用，但有专门的 EEPROM。该单片机为 1T 8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 5.5V ~ 4.5V，SRAM

空间大小为 2K（2048）字节，程序空间大小为 60K，有两组高速异步串行通信端口 UART 及 1 组 SPI、内部 EEPROM、A/D 转换、CCP/PCA/PWM 功能，工作频率可到 28MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 LQFP 贴片封装，管脚数为 44。

2) STC15L2K60S2-30I-LQFP44 表示：

用户不可以将用户程序区的程序 FLASH 当 EEPROM 使用，但有专门的 EEPROM。该单片机为 1T 8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 2.4V ~ 3.6V，SRAM 空间大小为 2K（2048）字节，程序空间大小为 60K，有两组高速异步串行通信端口 UART 及 1 组 SPI、内部 EEPROM、A/D 转换、CCP/PCA/PWM 功能，工作频率可到 30MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 LQFP 贴片封装，管脚数为 44。

3) IAP15F2K61S2-28I-LQFP44 表示：

用户可以将用户程序区的程序 FLASH 当 EEPROM 使用。该单片机为 1T 8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 5.5V ~ 4.5V，SRAM 空间大小为 2K（2048）字节，程序空间大小为 61K，有两组高速异步串行通信端口 UART 及 1 组 SPI、内部 EEPROM、A/D 转换、CCP/PCAPWM 功能，工作频率可到 28MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 LQFP 贴片封装，管脚数为 44。

4) IAP15L2K61S2-28I-LQFP44 表示：

用户可以将用户程序区的程序 FLASH 当 EEPROM 使用。该单片机为 1T8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 2.4V ~ 3.6V，SRAM 空间大小为 2K（2048）字节，程序空间大小为 61K，有两组高速异步串行通信端口 UART 及 1 组 SP1、内部 EEPROM、A/D 转换、CCP/PCA/PWM 功能，工作频率可到 28MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 LQFP 贴片封装，管脚数为 44。

5) IAP15F2K61S-28I-LQFP44 表示：

用户可以将用户程序区的程序 FLASH 当 EEPROM 使用。该单片机为 1T8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 5.5V ~ 4.5V，SRAM 空间大小为 2K（2048）字节，程序空间大小为 61K，有 1 组高速异步串行通信端口 UART 及 1 组 SPI、内部 EEPROM、工作频率可到 28MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 LQFP 贴片封装，管脚数为 44。

6) IAP15L2K61S-28I-LQFP44 表示：

用户可以将用户程序区的程序 FLASH 当 EEPROM 使用。该单片机为 IT 8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 2.4V ~ 3.6V，SRAM 空间大小为 2K（2048）字节，程序空间大小为 61K，有 1 组高速异步串行通信端口 UART 及 1 组 SPI、内部 EEPROM、工作频率可到 28MHZ，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 LQFP 贴片封装，管脚数为 44。

7) IRC15F2K63S2-28I-LQFP44 表示：

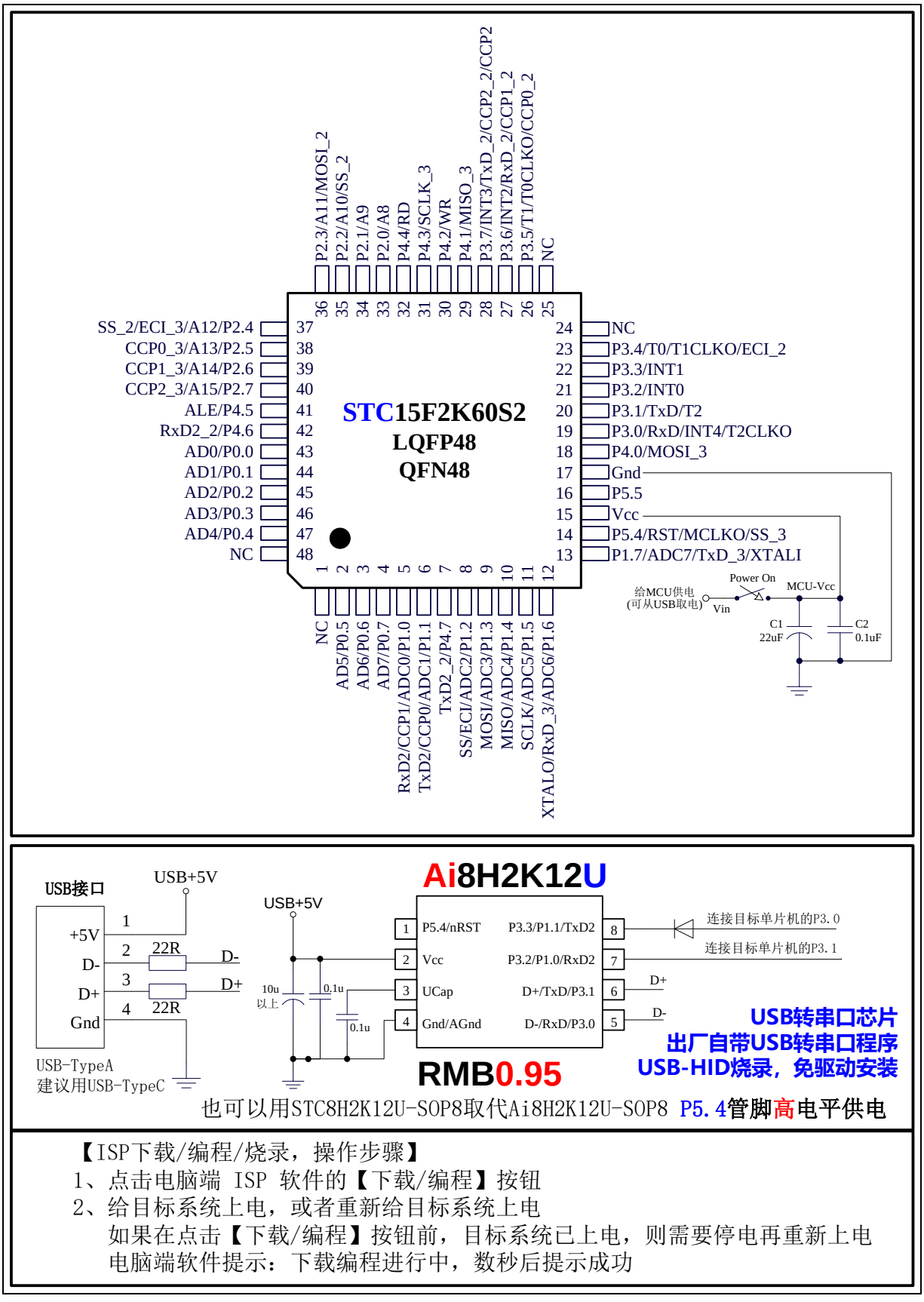
用户可以将用户程序区的程序 FLASH 当 EEPROM 使用，且使用内部 24MHz 时钟或外部晶振。该单片机为 1T 8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 5.5V ~ 4.5V，SRAM 空间大小为 2K（2048）字节，程序空间大小为 63.5K，有两组高速异步串行通信端口 UART 及 1 组 SPI、内部 EEPROM、A/D 转换、CCP/PCA/PWM 功能，工作频率可到 28MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 LQFP 贴片封装，管脚数为 44。

8) STC15F2K24AS-28I-LQFP44 表示：

用户不可以将用户程序区的程序 FLASH 当 EEPROM 使用，但有专门的 EEPROM。该单片机为 1T 8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 5.5V ~ 4.5V，SRAM 空间大小为 2K（2048）字节，程序空间大小为 24K，有 1 组高速异步串行通信端口 UART 及 1 组 SPI、内部 EEPROM、A/D 转换、CCP/PCA/PWM 功能，工作频率可到 28MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 LQFP 贴片封装，管脚数为 44。

3.5.6 管脚图及下载线路图

3.5.6.1 管脚图及下载线路图，最小系统（LQFP48/QFN48）



也可以用STC8H2K12U-SOP8取代Ai8H2K12U-SOP8 P5.4管脚高电平供电

【ISP下载/编程/烧录，操作步骤】

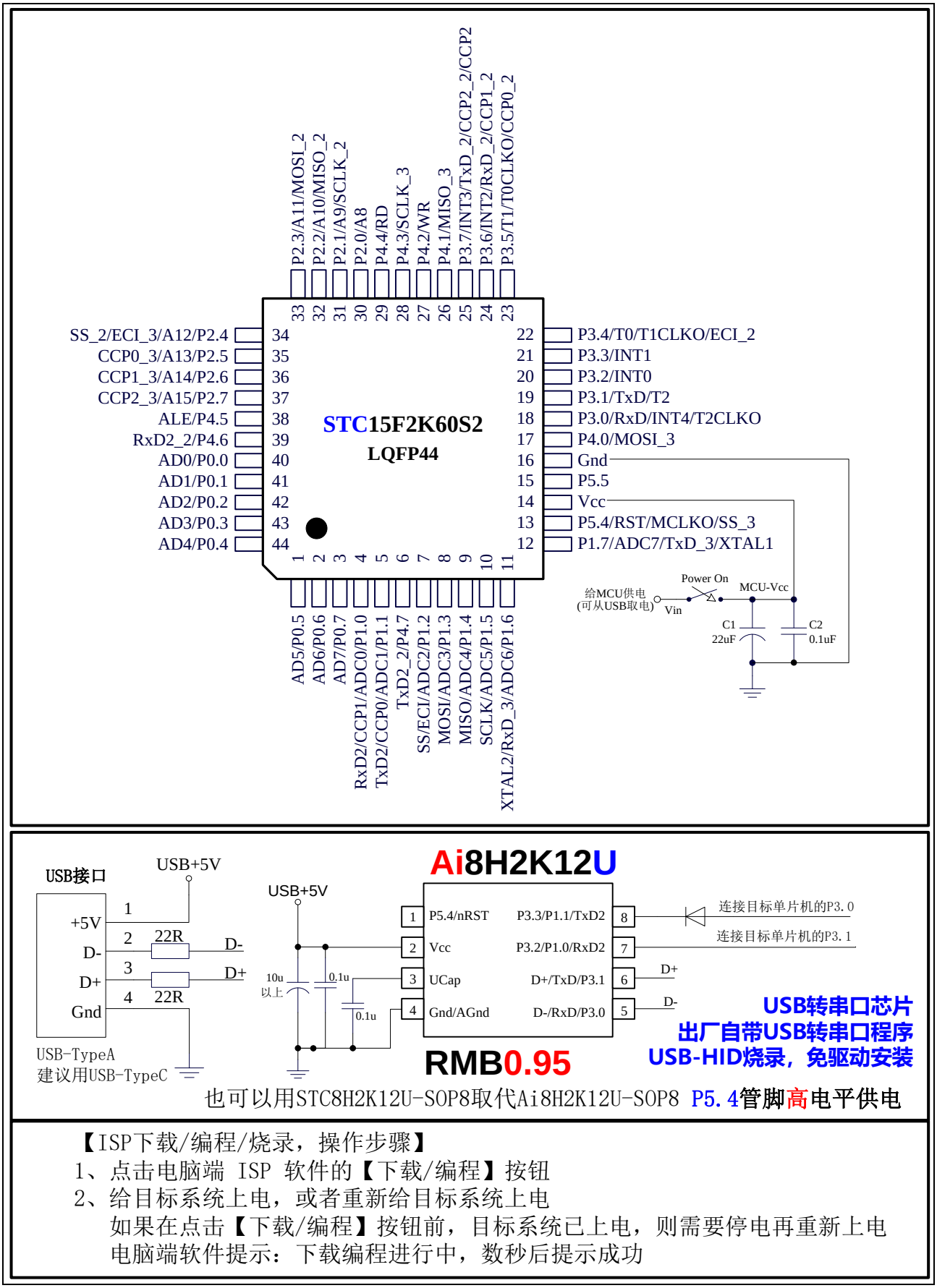
1、点击电脑端 ISP 软件的【下载/编程】按钮

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

3.5.6.2 管脚图及下载线路图，最小系统（LQFP44）



也可以用STC8H2K12U-SOP8取代Ai8H2K12U-SOP8 P5.4管脚高电平供电

【ISP下载/编程/烧录，操作步骤】

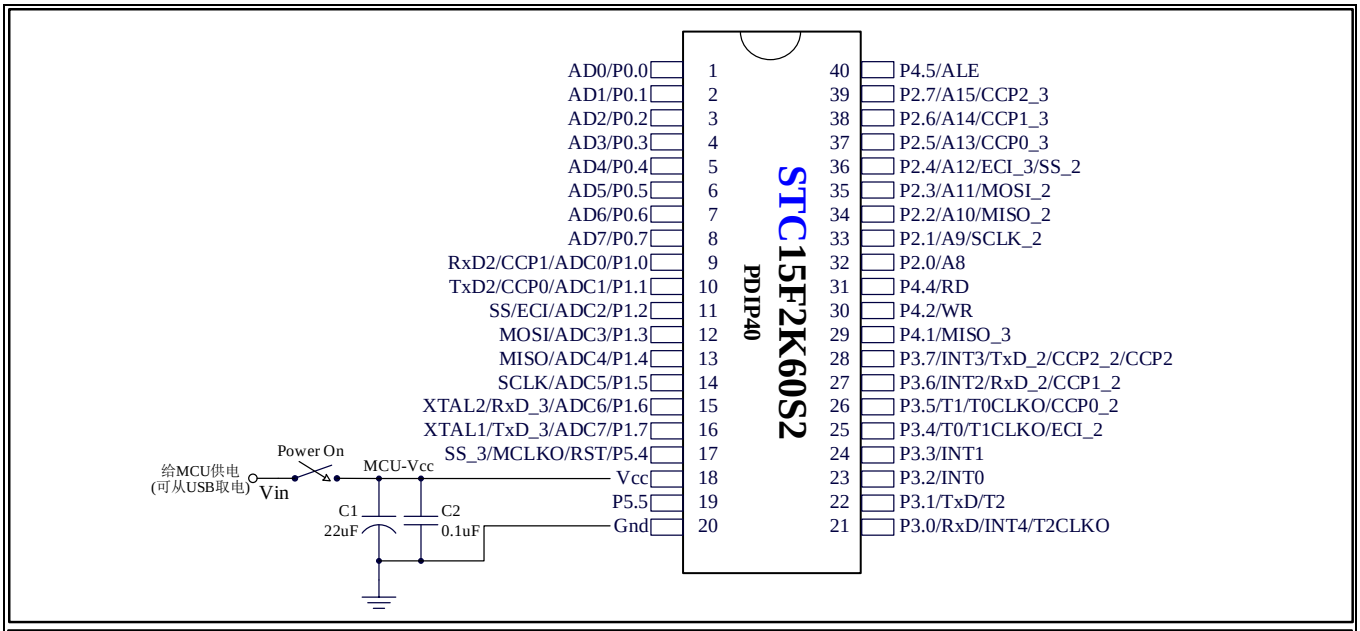
1、点击电脑端 ISP 软件的【下载/编程】按钮

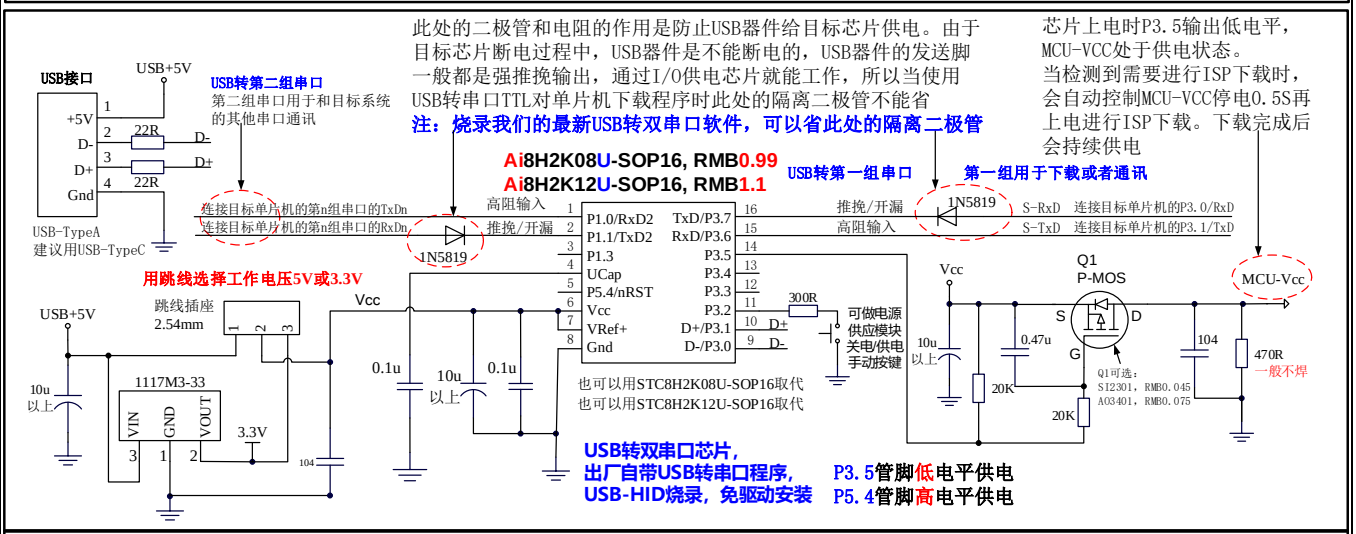
2、给目标系统上电，或者重新给目标系统上电

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

3.5.6.3 管脚图及下载线路图，最小系统（PDIP40）





【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。

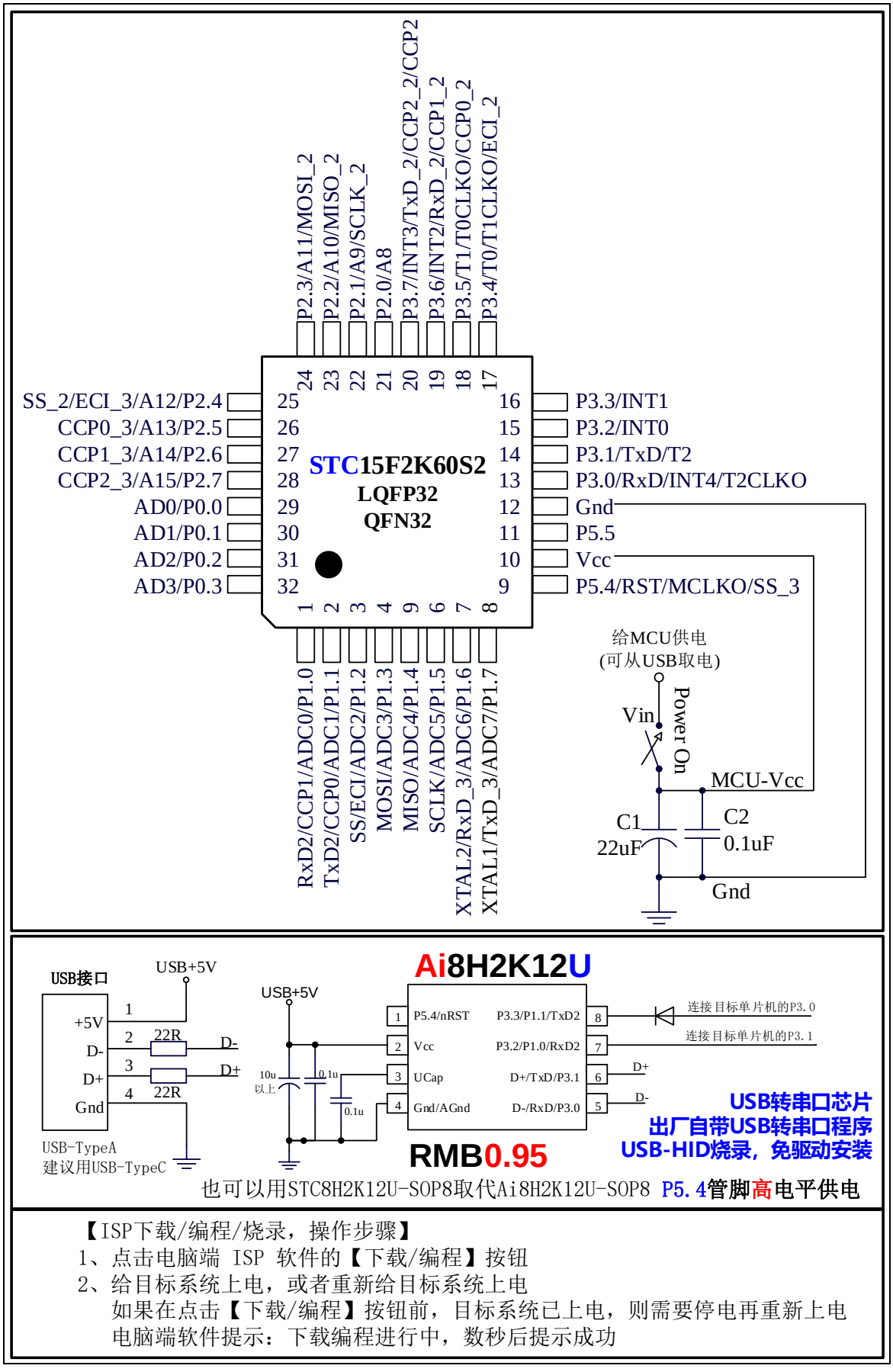
【应用场景二：不从本工具给目标系统供电】

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，P5.4/RST变成复位脚。

3.5.6.4 管脚图及下载线路图，最小系统（LQFP/QFN32）



【ISP下载/编程/烧录，操作步骤】

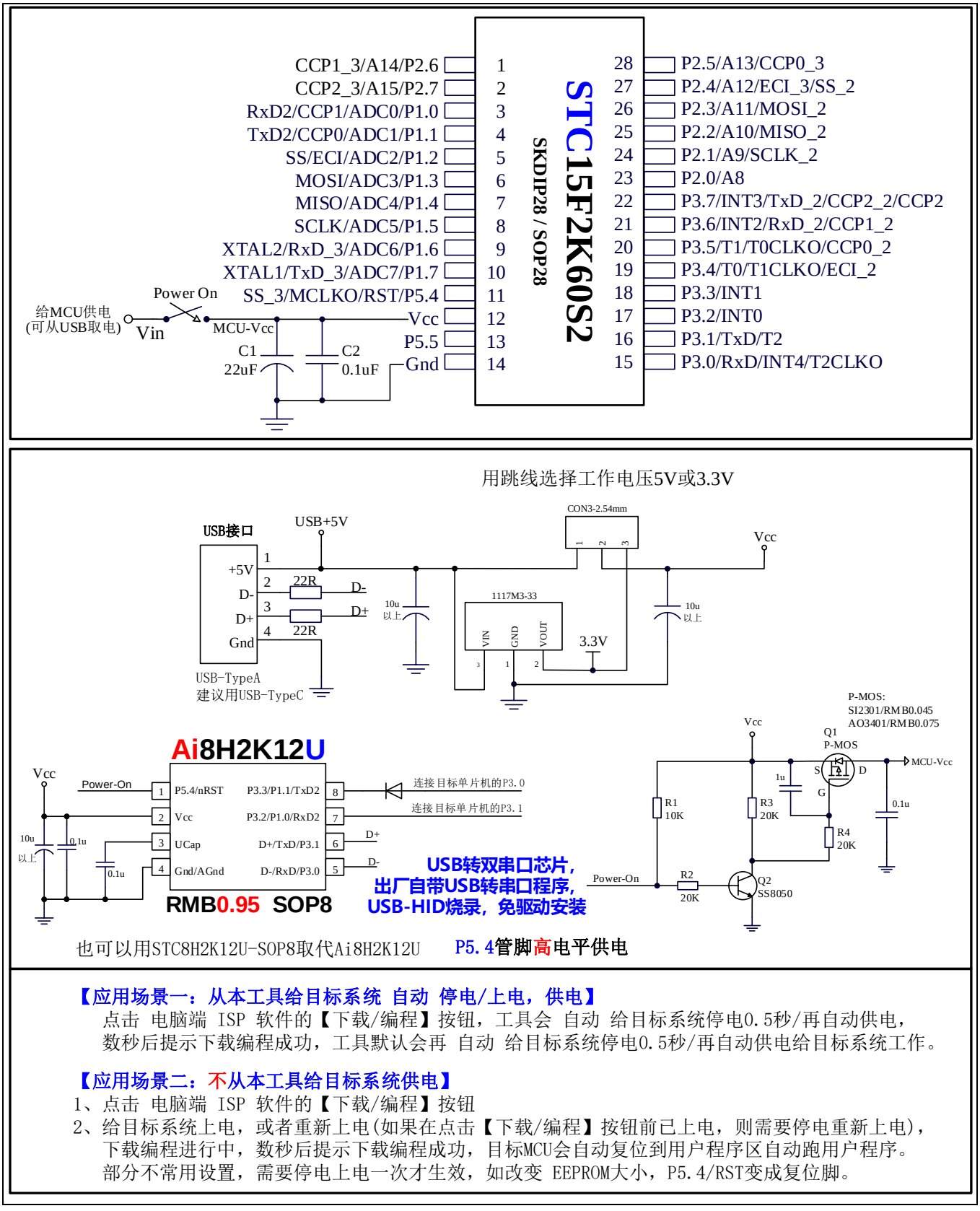
1、点击电脑端 ISP 软件的【下载/编程】按钮

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

3.5.6.5 管脚图及下载线路图，最小系统（SKDIP28/SOP28）



【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。

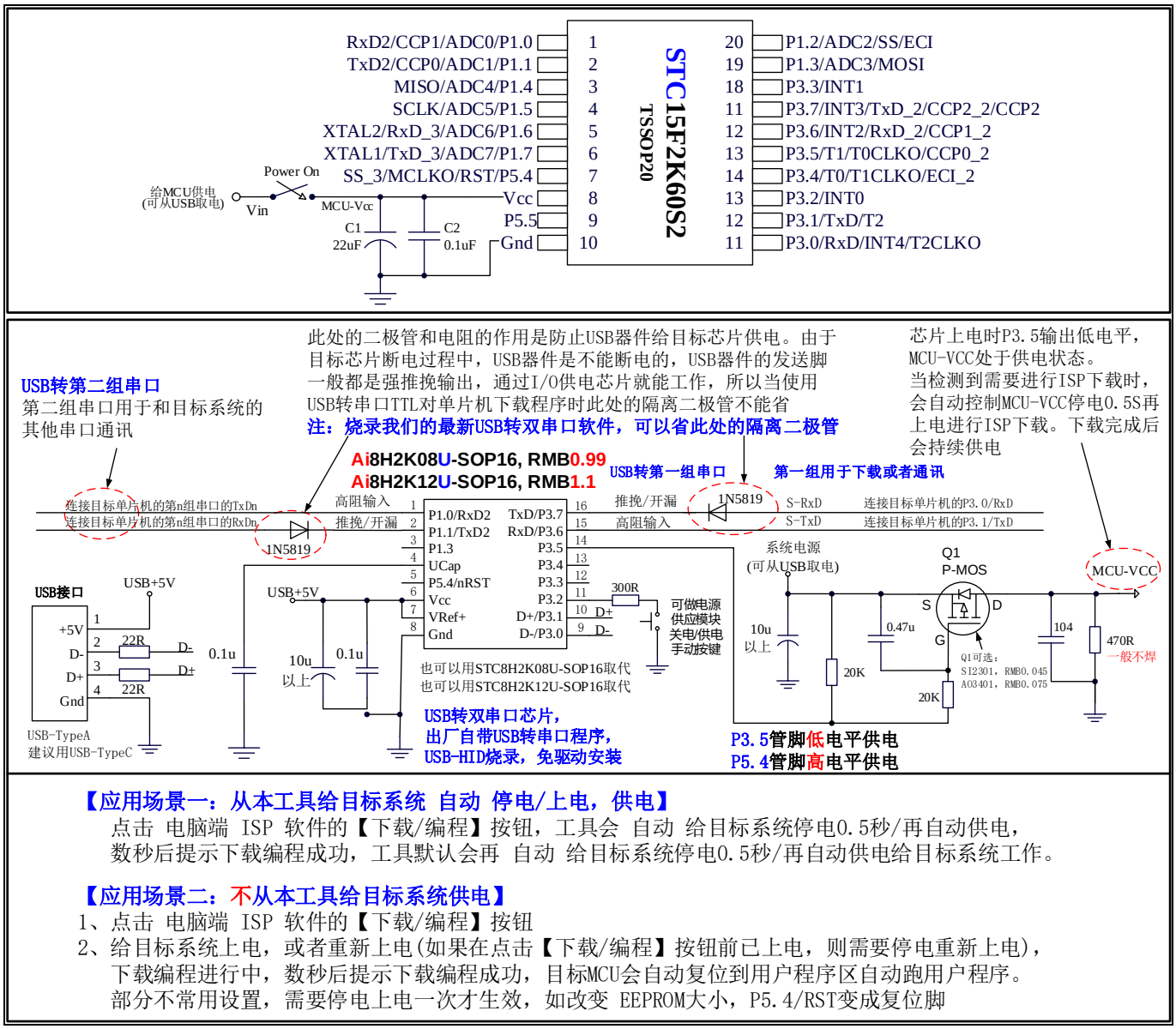
【应用场景二：不从本工具给目标系统供电】

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，P5.4/RST变成复位脚。

3.5.6.6 管脚图及下载线路图，最小系统（TSSOP20）



【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，P5.4/RST变成复位脚

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000,0000
P_SW2	BAH	Peripheral function switch	S4_S	S3_S	S2_S	xxxx,xxx0					
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	Tx2_Rx2	CLKS2	CLKS1	CLKS0	0000,x000

串口 1/S1 可在 3 个地方切换，由 S1_S0 及 S1_S1 控制位来选择		
S1_S1	S1_S0	串口 1/S1 可在 P1/P3 之间来回切换
0	0	串口 1/S1 在[P3.0/RxD，P3.1/TxD]
0	1	串口 1/S1 在[P3.6/RxD_2，P3.7/TxD_2]
1	0	串口 1/S1 在[P1.6/RxD_3/XTAL2，P1.7/TxD_3/XTAL1] 串口 1 在 P1 口时要使用内部时钟
1	1	无效

串口 1 建议放在[P3.6/RxD_2，P3.7/TxD_2]或[P1.6/RxD_3/XTAL2，P1.7/TxD_3/XTAL1]上。

建议用户在程序中将[S1_S1，S1_S0]的值设置为[0,1]或[1,0]，进而将串口 1 放在[P3.6/RxD_2，P3.7/TxD_2]或[P1.6/RxD_3/XTAL2，P1.7/TxD_3/XTAL1]上

CCP 可在 3 个地方切换，由 CCP_S1 / CCP_S0 两个控制位来选择		
CCP_S1	CCP_S10	CCP 可在 P1/P2/P3 之间来回切换
0	0	CCP 在[P1.2/ECI，P1.1/CCP 0，P1.0/CCP1，P3.7/CCP2]
0	1	CCP 在[P3.4/ECI_2，P3.5/CCP0_2，P3.6/CCP1_2，P3.7/CCP2_2]
1	0	CCP 在[P2.4/ECI_3，P2.5/CCP0_3，P2.6/CCP1_3，P2.7/CCP2_3]
1	1	无效

SPI 可在 3 个地方切换，由 SPI_S1 / SPI_S0 两个控制位来选择		
SPI_S1	SPI_S0	SPI 可在 P1/P2/P4 之间来回切换
0	0	SPI 在[P1.2/SS，P1.3/MOSI，P1.4/MISO，P1.5/SCLK]
0	1	SPI 在[P2.4/SS_2，P2.3/MOSI_2，P2.2/MISO_2，P2.1/SCLK_2]
1	0	SPI 在[P5.4/SS_3，P4.0/MOSI_3，P4.1/MISO_3，P4.3/SCLK_3]
1	1	无效

串口 2/S2 可在 2 个地方切换，由 S2_S 控制位来选择	
S2_S	S2 可在 P1/P4 之间来回切换
0	串口 2/S2 在[P1.0/RxD2，P1.1/TxD2]
1	串口 2/S2 在[P4.6/RxD2_2，P4.7/TxD2_2]

DPS：DPTR registers select bit. DPTR 寄存器选择位 DPTRO 被选择 DPTR1 被选择

0：DPTR0 is selected

1：DPTRl is selected

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	Tx2_Rx2	CLKS2	CLKS1	CLKS0	0000,x000

MCKO_S1	MCKO_S0	主时钟对外分频输出控制位 (主时钟可对外输出内部 R/C 时钟，也可对外输出外部输入的时钟或外部晶体振荡产生的时钟)
0	0	主时钟不对外输出时钟
0	1	主时钟对外输出时钟，但时钟频率不被分频，输出时钟频率 = MCLK / 1
1	0	主时钟对外输出时钟，但时钟频率被 2 分频，输出时钟频率 = MCLK / 2
1	1	主时钟对外输出时钟，但时钟频率被 4 分频，输出时钟频率 = MCLK / 4

现供货的 STC15F2K60S2 系列 C 版本单片机的主时钟对外输出管脚 P5.4/MCLKO 只可以对外分频输出内部 R/C 时钟，STC15F2K60S2 系列下一升级版本----STC15W2K60S2 系列单片机主时钟对外输出管脚 P5.4/MCLKO 既可以对外分频输出内部 R/C 时钟，也可对外分频输出外部输入的时钟或外部晶体振荡产生的时钟。

上述 MCLK 是指主时钟频率。STC15F2K60S2 系列单片机在 MCLKO/P5.4 口对外输出时钟。

STC15 系列 8-pin 单片机（如 STC15F104W 系列）在 MCLKO/P3.4 口对外输出时钟，STC15 系列 16-pin 及其以上单片机（如 STC15W4K32S4 系列、STC15F2K60S2 系列等）均在 MCLKO/P5.4 口对外输出时钟。

若用户要对外输出 13.56MHz 时钟，则建议选择主时钟输出 27.12MHz（27.12÷2=13.56）

ADRJ：ADC 转换结果调整

0：ADC_RES[7：0]存放高 8 位 ADC 结果，ADC_RESL[1：0]存放低 2 位 ADC 结果

1：ADC_RES[1：0]存放高 2 位 ADC 结果，ADC_RESL[7：0]存放低 8 位 ADC 结果

Tx_Rx：串口 1 的中继广播方式设置

0：串口 1 为正常工作方式

1：串口 1 为中继广播方式，即将 RxD 端口输入的电平状态实时输出在 TxD 外部管脚上，TxD 外部管脚可以对 RxD 管脚的输入信号进行实时整形放大输出，TxD 管脚的对外输出实时反映 RxD 端口输入的电平状态。

串口 1 的 RxD 管脚和 TxD 管脚可以在 3 组不同管脚之间进行切换：[RxD/P3.0，TxD/P3.1]
[RxD_2/P3.6，TxD_2/P3.7]；
[RxD_3/P1.6，TxD_3/P1.7]

Tx2_Rx2：串口 2 的中继广播方式设置，功能暂时保留

串口 2 的 RxD2 管脚和 TxD2 管脚可以在 2 组不同管脚之间进行切换：[RxD2/P1.0，TxD2/P1.1]；
[RxD2_2/P4.6TxD2_2/P4.7]

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	Tx2_Rx2	CLKS2	CLKS1	CLKS0	0000, x000

CLKS2	CLKS1	CLKS0	系统时钟选择控制位 (系统时钟是指对主时钟进行分频后供给 CPU、串行口、SPI、定时器、CCP/PWM/PCA、A/D 转换的实际工作时钟)
0	0	0	主时钟频率/1,不分频
0	0	1	主时钟频率/2
0	1	0	主时钟频率/4
0	1	1	主时钟频率/8
1	0	0	主时钟频率/16
1	0	1	主时钟频率/32
1	1	0	主时钟频率/64
1	1	1	主时钟频率/128

现供货的 STC15F2K60S2 系列 C 版本单片机的主时钟对外输出管脚 P5.4/MCLKIO 只可以对外分频输出内部 R/C 时钟，STC15F2K60S2 系列下一升级版本--STC15W2K60S2 系列单片机主时钟对外输出管脚 P5.4/MCLKO 既可以对外分频输出内部 R/C 时钟，也可对外分频输出外部输入的时钟或外部晶体振荡产生的时钟。

3.5.7 STC15F2K60S2 系列单片机的管脚说明

管脚	管脚编号							说明	
	LQFP44	PLCC44	PDIP40	SOP32	LQFP32 QFN32	SOP28 SKDIP28	TSSOP20		
P0.0	40	2	1	1	29	-		P0.0/AD0	P0：P0 口既可作为输入/输出口，也可作为地址/数据复用总线使用。 当 P0 口作为输入/输出口时，P0 可以由软件配置成准双向口/弱上拉、推挽输出/强上拉、高阻输入(电流既不能流入也不能流出)及开漏输出等 4 种工作类型之一，上电复位后为准双向口/弱上拉模式。 当 P0 作为地址/数据复用总线使用时，是低 8 位地址线[A0~A7]及数据线的[D0~D7]。
P0.1	41	3	2	2	30	-		P0.1/AD1	
P0.2	42	4	3	3	31	-		P0.2/AD2	
P0.3	43	5	4	4	32	-		P0.3/AD3	
P0.4	44	6	5	-	-	-		P0.4/AD4	
P0.5	1	7	6	-	-	-		P0.5/AD5	
P0.6	2	8	7	-	-	-		P0.6/AD5	
P0.7	3	9	8	-	-	-		P0.7/AD7	
P1.0	4	10	9	5	1	3	1	P1.0	标准 I/O 口 PORT1[0]
								ADC0	ADC 输入通道-0
								CCP1	外部信号捕获(频率测量或当外部中断使用)、 高速脉冲输出及脉宽调制输出通道-1
								RxD2	串口 2 数据接收端
P1.1	5	11	10	6	2	4	2	P1.1	标准 I/O 口 PORT1[1]
								ADC1	ADC 输入通道-1
								CCP0	外部信号捕获(频率测量或当外部中断使用)、 高速脉冲输出及脉宽调制输出通道-0
								TxD2	串口 2 数据发送端
P1.2	7	13	11	7	3	5	20	P1.2	标准 I/O 口 PORT1[2]
								ADC2	ADC 输入通道-2
								SS	SPI 同步串行接口的从机选择信号
								ECI	CCP / PCA 计数器的外部脉冲输入脚
P1.3	8	14	12	8	4	6	19	P1.3	标准 I/O 口 PORT1[3]
								ADC3	ADC 输入通道-3
								MOSI	SPI 同步串行接口的主出从入 (主器件的输出和从器件的输入)
P1.4	9	15	13	9	5	7	3	P1.4	标准 I/O 口 PORT1[4]
								ADC4	ADC 输入通道-4
								MISO	SPI 同步串行接口的主入从出 (主器件的输入和从器件的输出)
P1.5	10	16	14	10	6	8	4	P1.5	标准 I/O 口 PORT1[5]
								ADC5	ADC 输入通道-5
								SCLK	SPI 同步串行接口的时钟信号
P1.6	11	17	15	11	7	9	5	P1.6	标准 I/O 口 PORT1[6]
								ADC6	ADC 输入通道-6
								RxD_3	串口 1 数据接收端
								XTAL2	内部时钟电路反相放大器的输出端，接外部晶振的其中一端。当直接使用外部时钟源时，此引脚可浮空，此时 XTAL2 实际将 XTAL1 输入的时钟进行输出。

管脚	管脚编号							说明	
	LQFP44	PLCC44	PDIP40	SOP32	LQFP32 QFN32	SOP28 SKDIP28	TSSOP20		
P1.7	12	18	16	12	8	10	6	P1.7	标准 I/O 口 PORT1[7]
								ADC7	ADC 输入通道-7
								TxD_3	串口 1 数据发送端
								XTAL1	内部时钟电路反相放大器输入端，接外部晶振的其中一端。 当直接使用外部时钟源时，此引脚是外部时钟源的输入端。
P2.0	30	36	32	25	21	23		P2.0	标准 I/O 口 PORT2[0]
								A8	地址总线第 8 位 — A8
								RSTOUT_LOW	上电后，输出低电平，在复位期间也是输出低电平，用户可用软件将其设置为高电平或低电平，如果要读外部状态，可将该口先置高后再读
P2.1	31	37	33	26	22	24		P2.1	标准 I/O 口 PORT2[1]
								A9	地址总线第 9 位 — A9
								SCLK_2	SPI 同步串行接口的时钟信号
P2.2	32	38	34	27	23	25		P2.2	标准 I/O 口 PORT2[2]
								A10	地址总线第 10 位 — A10
								MISO_2	SPI 同步串行接口的主入从出 (主器件的输入和从器件的输出)
P2.3	33	39	35	28	24	26		P2.3	标准 I/O 口 PORT2[3]
								A11	地址总线第 11 位 — A11
								MOSI_2	SPI 同步串行接口的主出从入 (主器件的输出和从器件的输入)
P2.4	34	40	36	29	25	27		P2.4	标准 I/O 口 PORT2[4]
								A12	地址总线第 12 位 — A12
								ECI_3	CCP / PCA 计数器的外部脉冲输入脚
								SS_2	SPI 同步串行接口的从机选择信号
P2.5	35	41	37	30	26	28		P2.5	标准 I/O 口 PORT2[5]
								A13	地址总线第 13 位 — A13
								CCP0_3	外部信号捕获(频率测量或当外部中断使用）、 高速脉冲输出及脉宽调制输出通道-0
P2.6	36	42	38	31	27	1		P2.6	标准 I/O 口 PORT2[6]
								A14	地址总线第 14 位 — A14
								CCP1_3	外部信号捕获(频率测量或当外部中断使用）、 高速脉冲输出及脉宽调制输出通道-1
P2.7	37	43	39	32	28	2		P2.7	标准 I/O 口 PORT2[7]
								A15	地址总线第 15 位 — A15
								CCP2_3	外部信号捕获(频率测量或当外部中断使用）、 高速脉冲输出及脉宽调制输出通道-2
P3.0	18	24	21	17	13	15	11	P3.0	标准 I/O 口 PORT3[0]
								RxD	串口 1 数据接收端
								INT4	外部中断 4，只能下降沿中断，INT4 支持掉电唤醒
								T2CLKO	T2 的时钟输出 可通过设置 INT_CLKO[2]位/T2CLKO 将该管脚配置为 T2CLKO

管脚	管脚编号							说明	
	LQFP44	PLCC44	PDIP40	SOP32	LQFP32 QFN32	SOP28 SKDIP28	TSSOP20		
P3.1	19	25	22	18	14	16	12	P3.1	标准 I/O 口 PORT3[1]
								TxD	串口 1 数据发送端
								T2	定时器/计数器 2 的外部输入
P3.2	20	26	23	19	15	17	13	P3.2	标准 I/O 口 PORT3[2]
								INT0	外部中断 0，既可上升沿中断也可下降沿中断. 如果 IT0(TCON.0)被置为 1，INT0 管脚仅为下降沿中断。 如果 IT0(TCON.0)被清 0，INT0 管脚既支持上升沿中断也支持下降沿中断。 INT0 支持掉电唤醒。
P3.3	21	27	24	20	16	18	18	P3.3	标准 I/O 口 PORT3[3]
								INT1	外部中断 1，既可上升沿中断也可下降沿中断. 如果 IT1(TCON.2)被置为 1，INT1 管脚仅为下降沿中断。 如果 IT1(TCON.2)被清 0，INT1 管脚既支持上升沿中断也支持下降沿中断。 INT1 支持掉电唤醒。
P3.4	22	28	25	21	17	19	14	P3.4	标准 I/O 口 PORT3[4]
								T0	定时器/计数器 0 的外部输入
								T1CLKO	定时器/计数器 1 的时钟输出 可通过设置 INT_CLKO[1]位/T1CLKO 将该管脚配置为 T1CLKO，也可对 T1 脚的外部时钟输入进行分频输出
								ECI_2	CCP / PCA 计数器的外部脉冲输入脚
P3.5	23	29	26	22	18	20	15	P3.5	标准 I/O 口 PORT3[5]
								T1	定时器/计数器 1 的外部输入
								T0CLKO	定时器/计数器 0 的时钟输出 可通过设置 INT_CLKO[0]位/T0CLKO 将该管脚配置为 T0CLKO，也可对 T0 脚的外部时钟输入进行分频输出
								CCP0_2	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-0
P3.6	24	30	27	23	19	21	16	P3.6	标准 I/O 口 PORT3[6]
								INT2	外部中断 2，只能下降沿中断。支持掉电唤醒
								RxD_2	串口 1 数据接收端
								CCP1_2	外部信号捕获(频率测量或当外部中断使用）、 高速脉冲输出及脉宽调制输出通道-1
P3.7	25	31	28	24	20	22	17	P3.7	标准 I/O 口 PORT3[7]
								INT3	外部中断 3，只能下降沿中断。支持掉电唤醒
								TxD_2	串口 1 数据发送端
								CCP2	外部信号捕获(频率测量或当外部中断使用）、 高速脉冲输出及脉宽调制输出通道-2
								CCP2_2	外部信号捕获(频率测量或当外部中断使用）、 高速脉冲输出及脉宽调制输出通道-2
P4.0	17	23	-	-	-	-		P4.0	标准 I/O 口 PORT4[0]
								MISO_3	SPI 同步串行接口的主入从出 (主器件的输入和从器件的输出)

管脚	管脚编号							说明	
	LQFP44	PLCC44	PDIP40	SOP32	LQFP32 QFN32	SOP28 SKDIP28	TSSOP20		
P4.1	26	32	29	-	-	-		P4.1	标准 I/O 口 PORT4[1]
								MOSI_3	SPI 同步串行接口的主出从入 (主器件的输出和从器件的输入)
P4.2	27	33	30	-	-	-		P4.2	标准 I/O 口 PORT4[2]
								WR	外部数据存储器写脉冲
P4.3	28	34	-	-	-	-		P4.3	标准 I/O 口 PORT4[3]
								SCLK_3	SPI 同步串行接口的时钟信号
P4.4	29	35	31	-	-	-		P4.4	标准 I/O 口 PORT4[4]
								RD	外部数据存储器读脉冲
P4.5	38	44	40	-	-	-		P4.5	标准 I/O 口 PORT4[5]
								ALE	地址锁存允许
P4.6	39	1	-	-	-	-		P4.6	标准 I/O 口 PORT4[6]
								RxD2_2	串口 2 数据接收端
P4.7	6	12	-	-	-	-		P4.7	标准 I/O 口 PORT4[7]
								TxD2_2	串口 2 数据发送端
P5.4	13	19	17	13	9	11	7	P5.4	标准 I/O 口 PORT5[4]
								RST	复位脚(高电平复位)
								MCLKO	主时钟输出:输出的频率可为 MCLK/1，MCLK/2，MCLK/4（MCLK 是指主时钟频率）。 现供货的 STC15F2K60S2 系列 C 版本单片机的主时钟外部输出管脚 P5.4/MCLKO 只可以对外分频输出内部 R/C 时钟，STC15F2K60S2 系列下一升级版本----STC15W2K60S2 系列单片机主时钟外部输出管脚 P5.4/MCLKO 既可以外部分频输出内部 R/C 时钟，也可外部分频输出外部输入的时钟或外部晶体振荡产生的时钟。
								SS_3	SPI 同步串行接口的从机选择信号
P5.5	15	21	19	15	11	13	9	标准 I/O 口 PORT5[5]	
Vcc	14	20	18	14	10	12	8	电源正极	
Gnd	16	22	20	16	12	14	10	电源负极，接地	

3.5.8 USB-Link1D 工具自动停电/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】

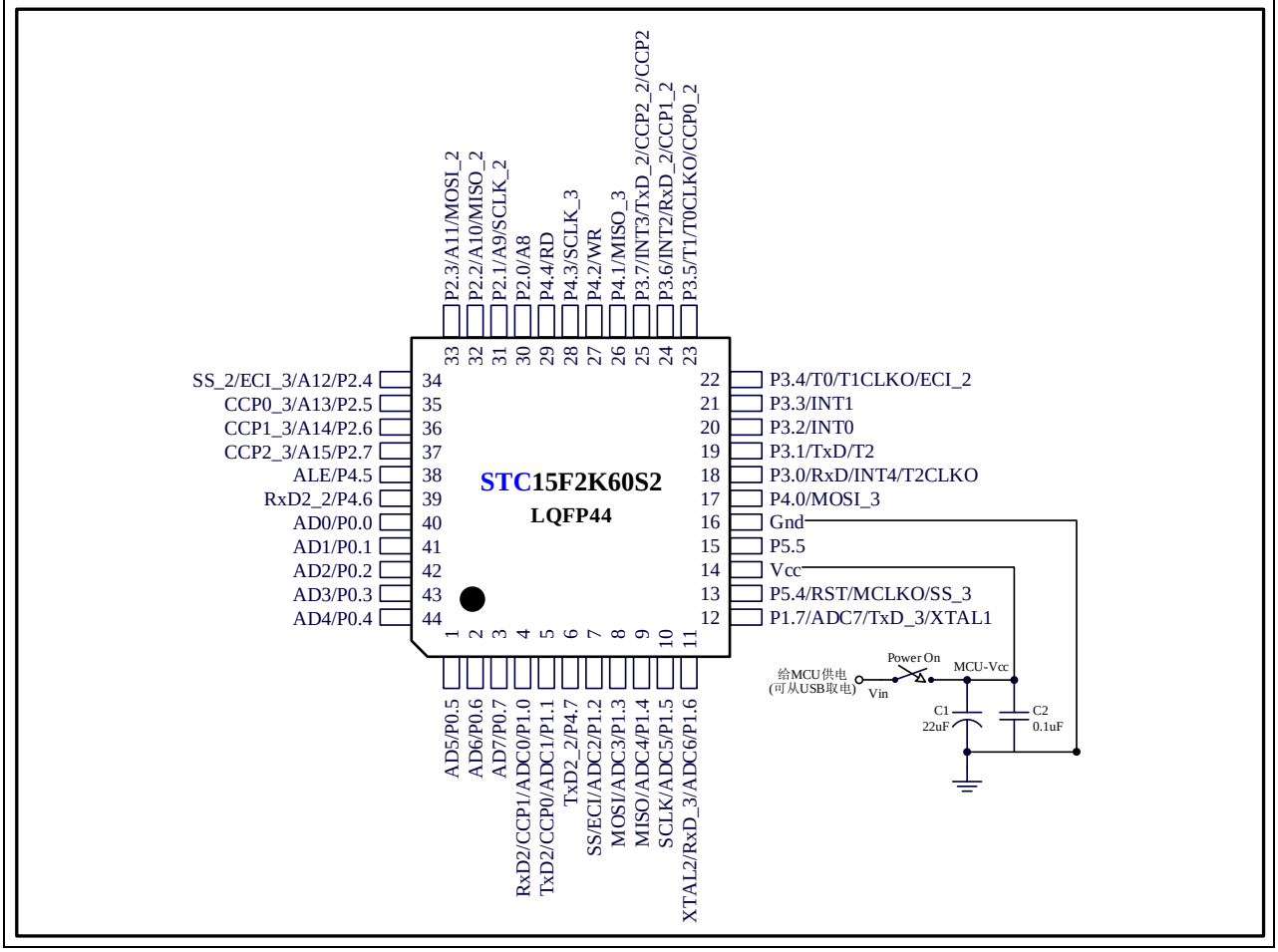
点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【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，P5.4/RST变成复位脚



3.5.9 【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

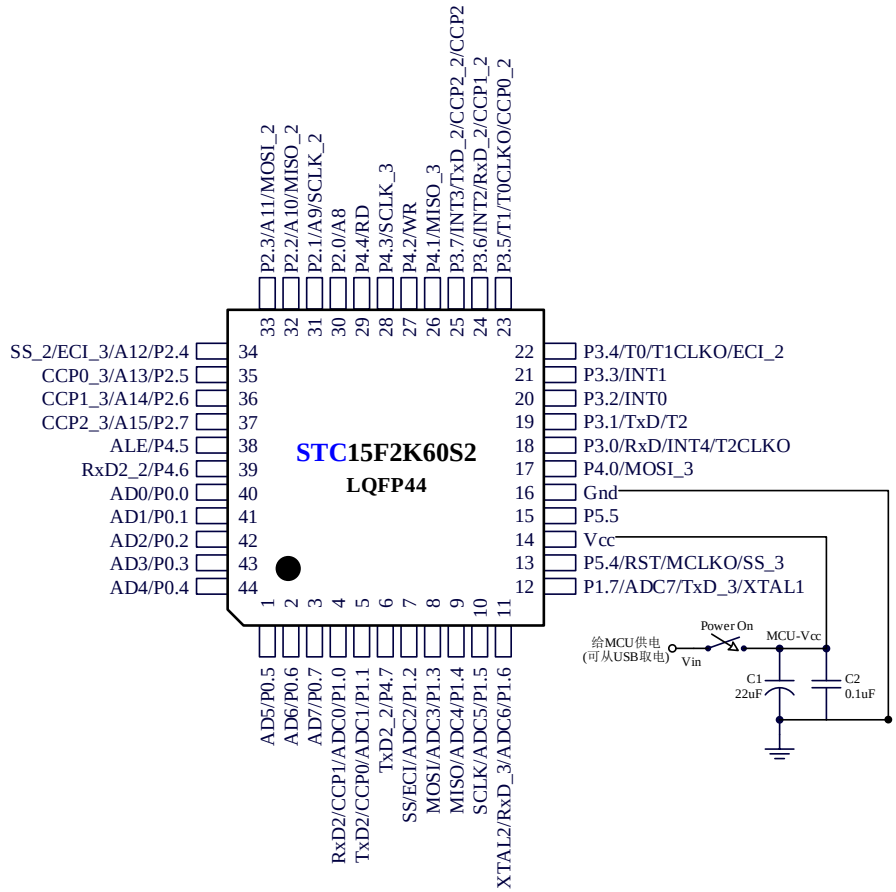
【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

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，P5.4/RST变成复位脚



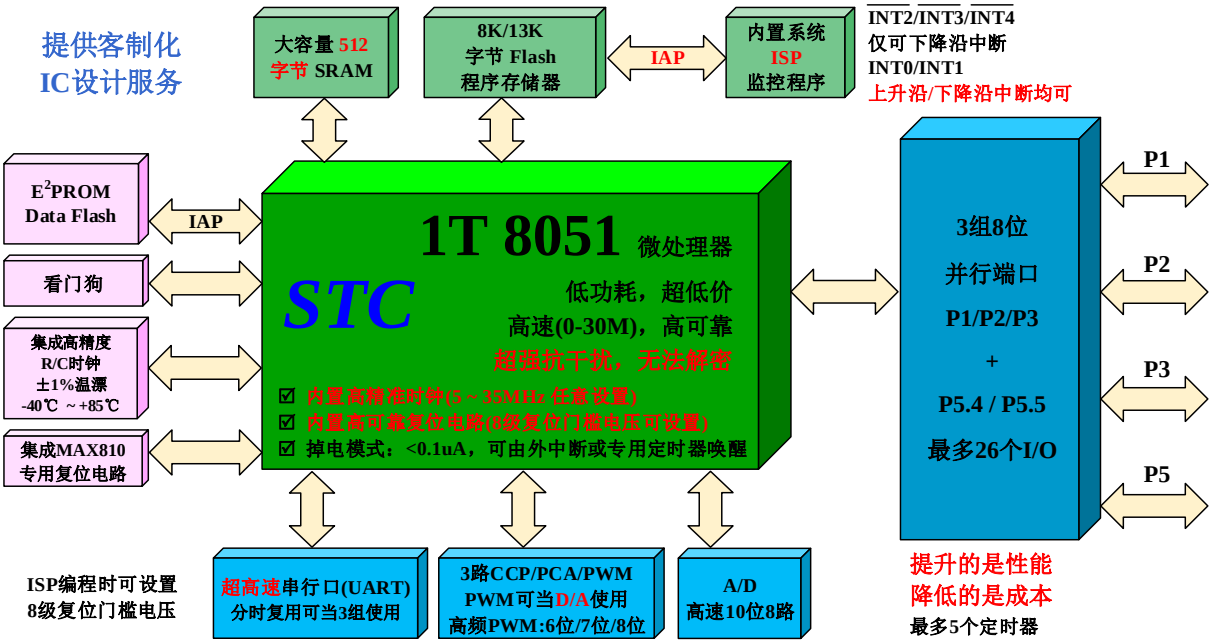
3.6 STC15F408AD 系列

3.6.1 STC15F408AD 系列单片机简介

STC15F408AD 系列单片机是 STC 生产的单时钟/机器周期（1T）的单片机，是高速/高可靠/低功耗/超强抗干扰的新一代 8051 单片机，采用 STC 第八代加密技术，无法解密，指令代码完全兼容传统 8051，但速度快 8-12 倍。内部集成高精度 R/C 时钟（±0.3%），+1%温漂（-40℃ ~ +85℃），常温下温漂+0.6%（-20℃ ~ +65℃），ISP 编程时 5MHz ~ 35MHz 宽范围可设置，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 8 级复位门槛电压可选）。3 路 CCP/PWM/PCA，8 路高速 10 位 A/D 转换（30 万次/秒），1 组高速异步串行通信口（UART，可在 3 组管脚之间进行切换，分时复用可作 3 组串口使用），1 组高速同步串行通信端口 SPI，针对串行口通信/电机控制干扰场合。

在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可

现 STC15 系列单片机采用 STC-Y5 超高速 CPU 内核，在相同的时钟频率下，速度又比 STC 早期的 1T 系列单片机（如 STC12 系列/STC11 系列/STC10 系列）的速度快 20%。



1. 增强型 8051 CPU，1T，单时钟/机器周期，速度比普通 8051 快 8-12 倍
2. 工作电压：STC15F408AD 系列工作电压：5.5V-4.5V（5V 单片机）
   STC15L408AD 系列工作电压：3.6V-2.4V（3V 单片机）
3. 8K/13K 字节片内 Flash 程序存储器，擦写次数 10 万次以上
4. 片内集成 512 字节的 SRAM，包括常规的 256 字节 RAM <idata>和内部扩展的 256 字节 XRAM <xdata>
5. 有片内 EEPROM 功能，擦写次数 10 万次以上
6. ISP/IAP，在系统可编程/在应用可编程，无需编程器/仿真器
7. 共 8 通道 10 位高速 ADC，速度可达 30 万次/秒，3 路 PWM 还可当 3 路 D/A 使用

8. 共 3 通道捕获/比较单元（CCP/PWM/PCA）
   ----也可用来再实现 3 个定时器或 3 个外部中断（支持上升沿/下隆沿中断）或 3 路 D/A
9. 利用 CCP/PCA 高速脉冲输出功能可实现 3 路 9 ~ 16 位 PWM（每通道占用系统时间小于 0.6%）
10. 利用定时器 T0 的时钟输出功能可实现高精度的 8 ~ 16 位 PWM（占用系统时间小于 0.4%）
11. 工作频率范围：5MHz ~ 28MHz，相当于普通 8051 的 60MHz ~ 336MHz
12. 内部高可靠复位，ISP 编程时 8 级复位门槛电压可选，可彻底省掉外部复位电路
13. 内部高精度 R/C 时钟（±0.3%），±1%温漂（-40℃ ~ +85℃），常温下温漂+0.6%（-20℃ ~ +65℃），ISP 编程时内部时钟从 5MHz ~ 35MHz 可设（5.5296MHz/11.0592MHz/22.1184MHz/33.1776MHz）
14. 不需外部晶振和外部复位，还可对外输出时钟和低电平复位信号
15. 一组高速异步串行通信端口 UART，可在 3 组管脚之间切换，分时复用可当 3 组串口使用：
    串行口（RxD/P3.0，TxD/P3.1）可以切换到（RxD_2/P3.6，TxD_2/P3.7），
    还可以切换到（RxD_3/P1.6，TxD_3/P1.7）
    注意：建议用户将串口放在 [P3.6/RxD_2，P3.7/TxD_2]或[P1.6/RxD_3，P1.7/TxD_3]上（[P3.0，P3.1 作下载/仿真用）；若用户未将串口切换到 [P3.6/RxD_2，P3.7/TxD_2]或 [P1.6/RxD_3，P1.7/TxD_3]，而是用[P3.0/RxD，P3.1/TxD]作串口，则务必在 ISP 编程时在 AIapp-ISP 软件的硬件选项中勾选“下次冷启动时，P3.2/P3.3 为 0/0 时才可以下载程序”
16. 一组高速同步串行通信端口 SPI
17. 支持程序加密后传输，防拦截
18. 支持 RS485 下载
19. 低功耗设计：低速模式，空闲模式，掉电模式/停机模式。
20. 可将掉电模式/停机模式唤醒的定时器：有内部低功耗掉电唤醒专用定时器。
21. 可将掉电模式/停机模式唤醒的资源有：
    - INT0/P3.2，INT1/P3.3（INT0/INT1 上升沿及下降沿中断均可），INT2/P3.6，INT3/P3.7，INT4/P3.0（INT2/INT3/INT4 仅可下降沿中断）；
    - 管脚 CCP0/CCP1/CCP2；
    - 管脚 T0/T2（下降沿，不产生中断，前提是在进入掉电模式/停机模式前相应的定时器中断已经被允许）；
    - 内部低功耗掉电唤醒专用定时器。
22. 共 5 个定时器/计数器，2 个 16 位可重装载并可实现可编程时钟输出的定时器/计数器，分别是定时器/计数器 0（即 T0）和定时器/计数器 2（即 T2），3 路 CCP/PWM/PCA 还可再实现 3 个定时器。
23. 可编程时钟输出功能（对内部系统时钟或对外部管脚的时钟输入进行时钟分频输出）：
    - 由于 STC15 系列 5V 单片机 I/O 口的对外输出速度最快不超过 13.5MHz，所以 5V 单片机的对外可编程时钟输出速度最快也不超过 13.5MHz.；
    - 而 3.3V 单片机 I/O 口的对外输出速度最快不超过 8MHz，故 3.3V 单片机的对外可编程时钟输出速度最快也不超过 8MHz.

    1) T0 在 P3.5/T0CLKO 进行可编程输出时钟（对内部系统时钟或对外部管脚 T0/P3.4 的时钟输入进行可编程时钟分频输出）；
    2) T2 在 P3.0/T2CLKO 进行可编程输出时钟（对内部系统时钟或对外部管脚 T2/P3.1 的时钟输入进行可编程时钟分频输出）；

    以上 2 个定时器/计数器均可 1 ~ 65536 级分频输出。

3) 主时钟在 P5.4/MCLKO 对外输出时钟，并可如下分频 MCLK/1，MCLK/2，MCLK/4

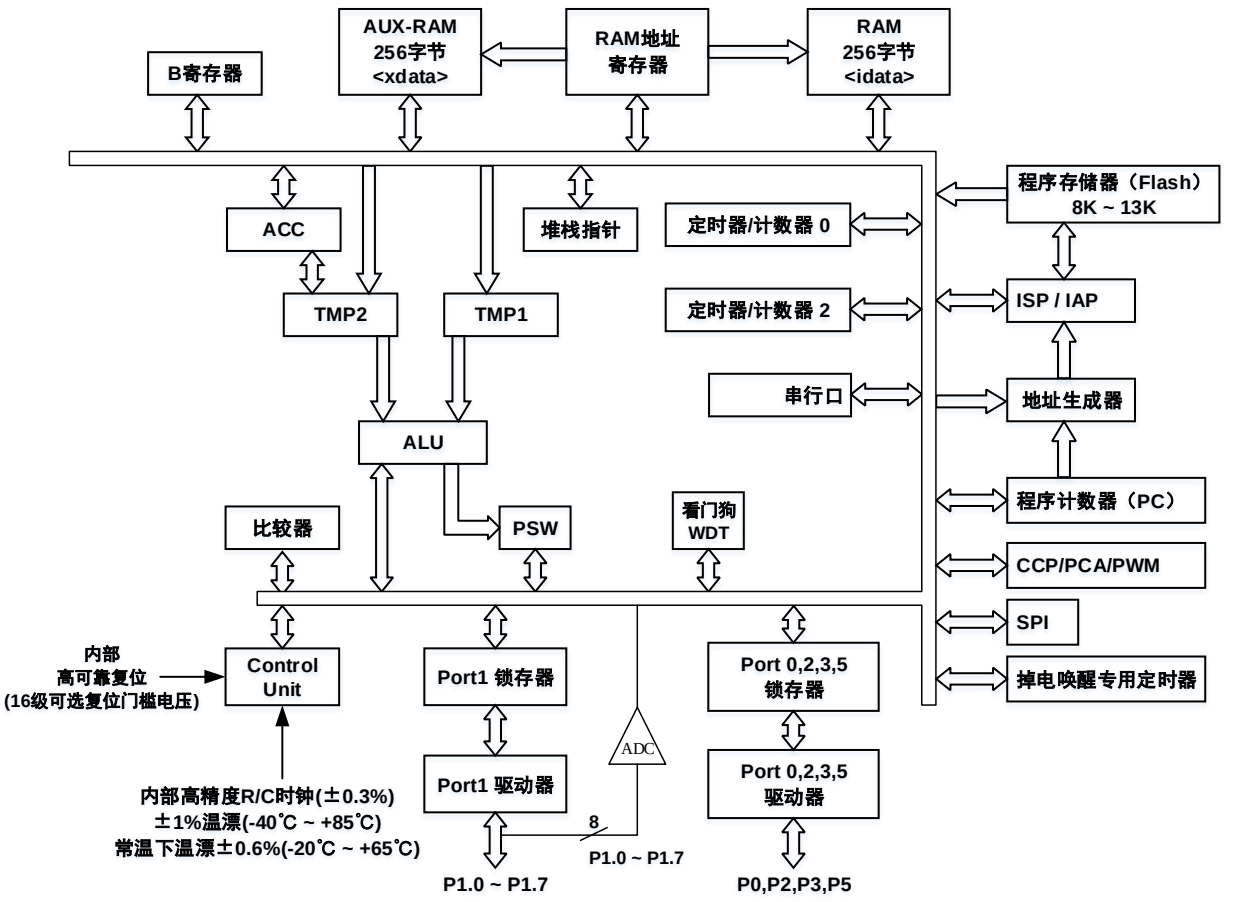
主时钟对外输出管脚 P5.4/MCLKO 既可对外输出内部 R/C 时钟，也可对外输出外部输）的时钟或外部晶体振荡产生的时钟。MCLK 是指主时钟频率，MCLKO 是指主时钟输出。

STC15 系列 8-pin 单片机（如 STC15F104W 系列）在 MCLKO/P3.4 口对外输出时钟，STC15 系列 16-pin 及其以上单片机均在 MCLKO/P5.4 口对外输出时钟，且 STC15W 系列 20-pin 及其以上单片机除可在 MCLKO/P5.4 口对外输出时钟外，还可在 MCLKO_2/P1.6 口对外输出时钟。

24. 硬件看门狗（WDT）
25. 先进的指令集结构，兼容普通 8051 指令集，有硬件乘法/除法指令
26. 通用 I/O 口（30/26 个），复位后为：准双向口/弱上拉（普通 8051 传统 I/O 口）。可设置成四种模式：准双向口/弱上拉，强推挽/强上拉，仅为输入/高阻，开漏
    每个 I/O 口驱动能力均可达到 20mA，但整个芯片电流最大不要超过 90mA.
    如果 I/O 口不够用，可外接 74HC595（参考价 0.15 元）来扩展 I/O 口，并可多芯片级联扩展几十个 I/O 口
27. 封装：LQFP32（9mmx9mm），SOP28，SKDIP28（★此系列的 28pin 单片机建议用 STC15W408AS 系列的相应封装的单片机取代）
28. 全部 175℃八小时高温烘烤，高品质制造保证
29. 开发环境：在 Keil C 开发环境中，选择 Inte1 8052 编译，头文件包含<reg51.h>即可

3.6.2 内部结构图

STC15F408AD 系列单片机的内部结构框图如下图所示。STC15F408AD 系列单片机中包含中央处理器（CPU）、程序存储器（Flash）、数据存储器（SRAM）、定时器/计数器、掉电唤醒专用定时器、I/O 口、高速 A/D 转换（30 万次/秒）、看门狗、高速异步串行通信端口 UART、CCP/PWM/PCA、高速同步串行端口 SPI，片内高精度 R/C 时钟及高可靠复位等模块。STC15F408AD 系列单片机几乎包含了数据采集和控制中所需要的所有单元模块，可称得上是一个片上系统（SysTem Chip 或 SysTem on Chip，简写为 STC，这是 STC 名称的由来）。



STC15F408AD系列内部结构框图

3.6.3 STC15F408AD 系列单片机选型价格一览表

型号	工作电压(V)	Flash 程序存储器(byte)	大容量 SRAM 字节	串行口	SPI 仅有主机模式	普通定时器计数器 T0/T2 外部管脚也能掉电唤醒	CCP PCA PWM 并可掉电唤醒	掉电唤醒专用定时器	标准外部中断支持掉电唤醒	A/D 8 路 (3 路 PWM 可当 3 路 D/A 使用)	DPTR	EEPROM	内部低压检测中断并可掉电唤醒	看门狗	内部高可靠复位(可选复位门槛电压)	内部高精准时钟	可对外输出时钟及复位	程序加密后传输(防拦截)	可设下次更新程序需口令	支持 RS485 下载	所有封装 SOP28 / SKDIP28 / LQFP32 封装价格 (RMB ¥)		
																					LQFP32 (30 个 I/O 口)	SOP28 (26 个 I/O 口)	SKDIP28 (26 个 I/O 口)
																					此系列的 28pin 单片机建议用 STC15W408AS 系列的相应封装的单片机取代		
STC15F408AD 系列单片机选型价格一览表特别提示：3 路 CCP/PCA/PWM 还可当 3 路定时器使用																							
STC15F408AD	5.5-4.5	8K	512	1	有	2	3-ch	有	5	10 位	1	5K	有	有	8 级	有	是	有	是	是	-	¥4.40	¥4.60
IAP15F413AD	5.5-4.5	13K	512	1	有	2	3-ch	有	5	10 位	1	IAP	有	有	8 级	有	是	有	是	是	-	¥4.40	¥4.60
																					用户可将用户程序区的程序 FLASH 当 EEPROM 使用		
STC15L408AD 系列单片机选型价格一览表																							
STC15L408AD	2.4-3.6	8K	512	1	有	2	3-ch	有	5	10 位	1	5K	有	有	8 级	有	是	有	是	是	-	¥4.40	¥4.60
IAP15L413AD	2.4-3.6	13K	512	1	有	2	3-ch	有	5	10 位	1	IAP	有	有	8 级	有	是	有	是	是	-	¥4.40	¥4.60
																					用户可将用户程序区的程序 FLASH 当 EEPROM 使用		

STC15F408AD 系列单片机只有定时器 0 和定时器 2，无定时器 1。

提供客制化 IC 服务

如果要用 28-pin 单片机，建议用户选用 SOP28 封装；

如果要用 32-pin 单片机，建议用户选用 LQFP32 封装。

我们直销，所以低价以上单价为 10K 起订量小每片需加 0.1 元，以上价格运费由客户承担，零售 10 片起如对价格不满，可来电要求降

程序加密后传输：程序拥有者产品出厂时将源程序和加密钥匙一起烧录 MCU 中，以后需要升级软件时，就可将程序加密后再用“发布项目程序”功能，生成一个用户自己界面的只有一个升级按钮的简单易用的升级软件，给最终使用者自己升级，而拦截不到您的原始程序。

特别声明：以 15F 和 15L 开头且有 SPI 功能的芯片，只支持“SPI 主机模式”，不支持“SPI 从机模式”；以 15W 开头且有 SPI 功能的芯片，SPI 主/从机模式均支持。

特别声明：以 15L 开头的芯片如需进入“掉电模式”，进入“掉电模式”前必须启动掉电唤醒定时器<3uA>，不超过 1 秒要唤醒一次，以 15F 和 15W 开头的芯片则不需要。

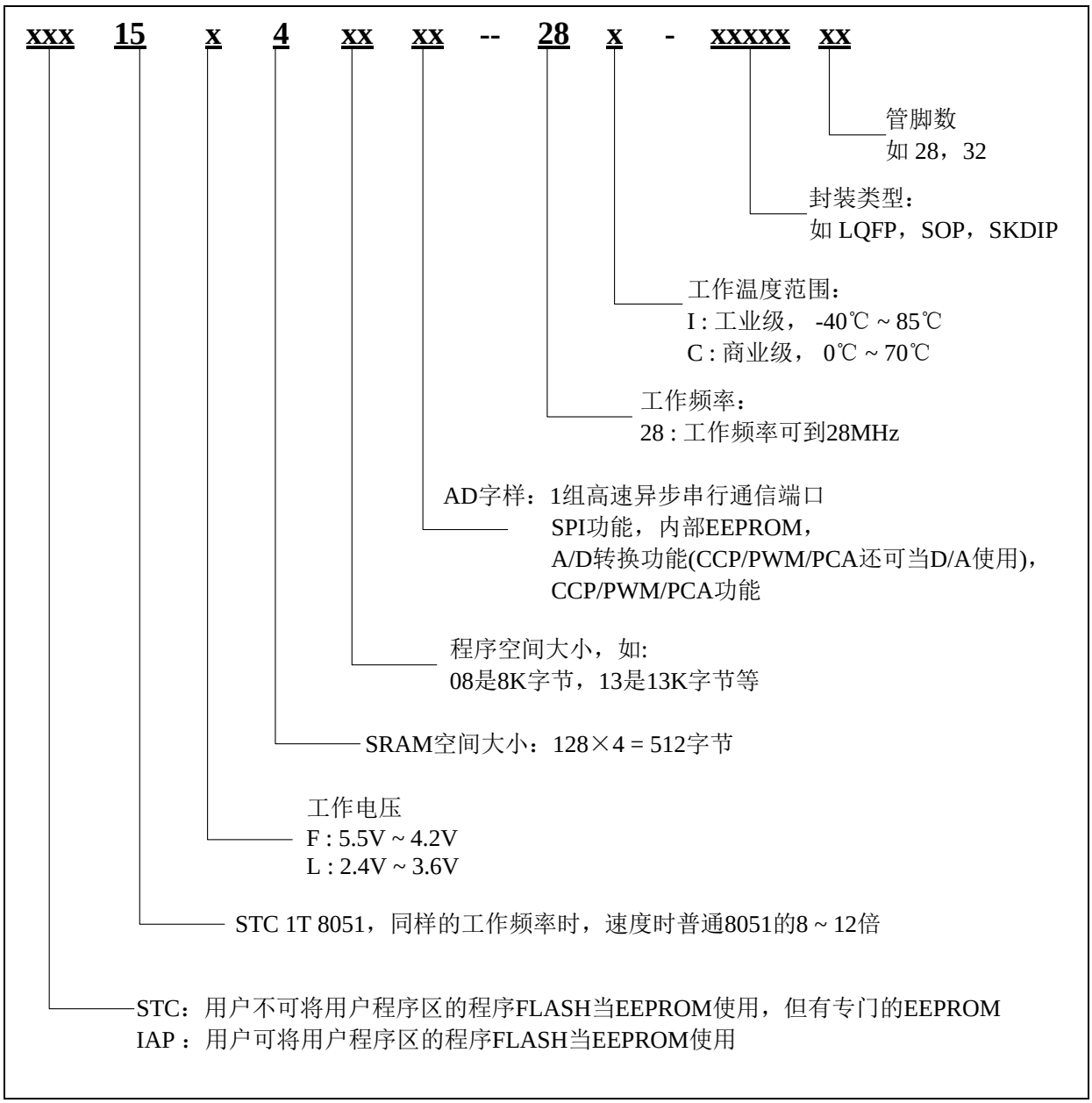
【总结】:

- STC15F408AD 系列单片机有：
  - ✓ 2 普通定时器/计数器（这 2 个普通定时器/计数器是指：T0 和 T2）；
  - ✓ 3 路 CCP/PWM/PCA（可再实现 3 个定时器使用）；
  - ✓ 掉电唤醒专用定时器；
  - ✓ 5 个支持掉电唤醒的外部中断 INT0/INT1/INT2/INT3/INT4；
  - ✓ 1 组高速异步串行通信端口；
  - ✓ 1 组高速同步串行通信端口 SPI；

- ✓ 8 路高速 10 位 A/D 转换器；
- ✓ 1 个数据指针 DPTR 等功能；
- ✓ STC15F408AD 系列单片机没有外部数据总线。

因为程序区的最后 7 个字节单元被强制性的放入全球唯一 1D 号的内容，所以用户实际可以使用的程序空间大小要比选型表中的大小少 7 个字节。

3.6.4 STC15F408AD 系列单片机命名规则



如何识别芯片版本号：如需知道芯片版本号，请查阅芯片表面印刷字中最下面一行的最后一个字母（如 D），该字母代表芯片版本号（如 D 版）

命名举例:

1) STC15F408AD-28I-SOP28 表示：

用户不可将用户程序区的程序 FLASH 当 EEPROM 使用，但有专门的 EEPROM。该单片机为 1T 8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 5.5V ~ 4.2V，SRAM 空间大小为 512 字节，程序空间大小为 8K，有 1 组高速异步串行通信端口 UART 及 SPI、内部 EEPROM、A/D 转换、CCP/PCA/PWM 功能，工作频率可到 28MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 SOP 贴片封装，管脚数为 28。

2) STC15L408AD-28I-SOP28 表示：

用户不可将用户程序区的程序 FLASH 当 EEPROM 使用，但有专门的 EEPROM。该单片机为 1T 8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 2.4V ~ 3.6V，SRAM 空间大小为 512 字节，程序空间大小为 8K，有 1 组高速异步串行通信端口 UART 及 SPI、内部 EEPROM、A/D 转换、CCP/PCA/PWM 功能，工作频率可到 28MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 SOP 贴片封装，管脚数为 28。

3) IAP15F413AD-28I-SOP28 表示：

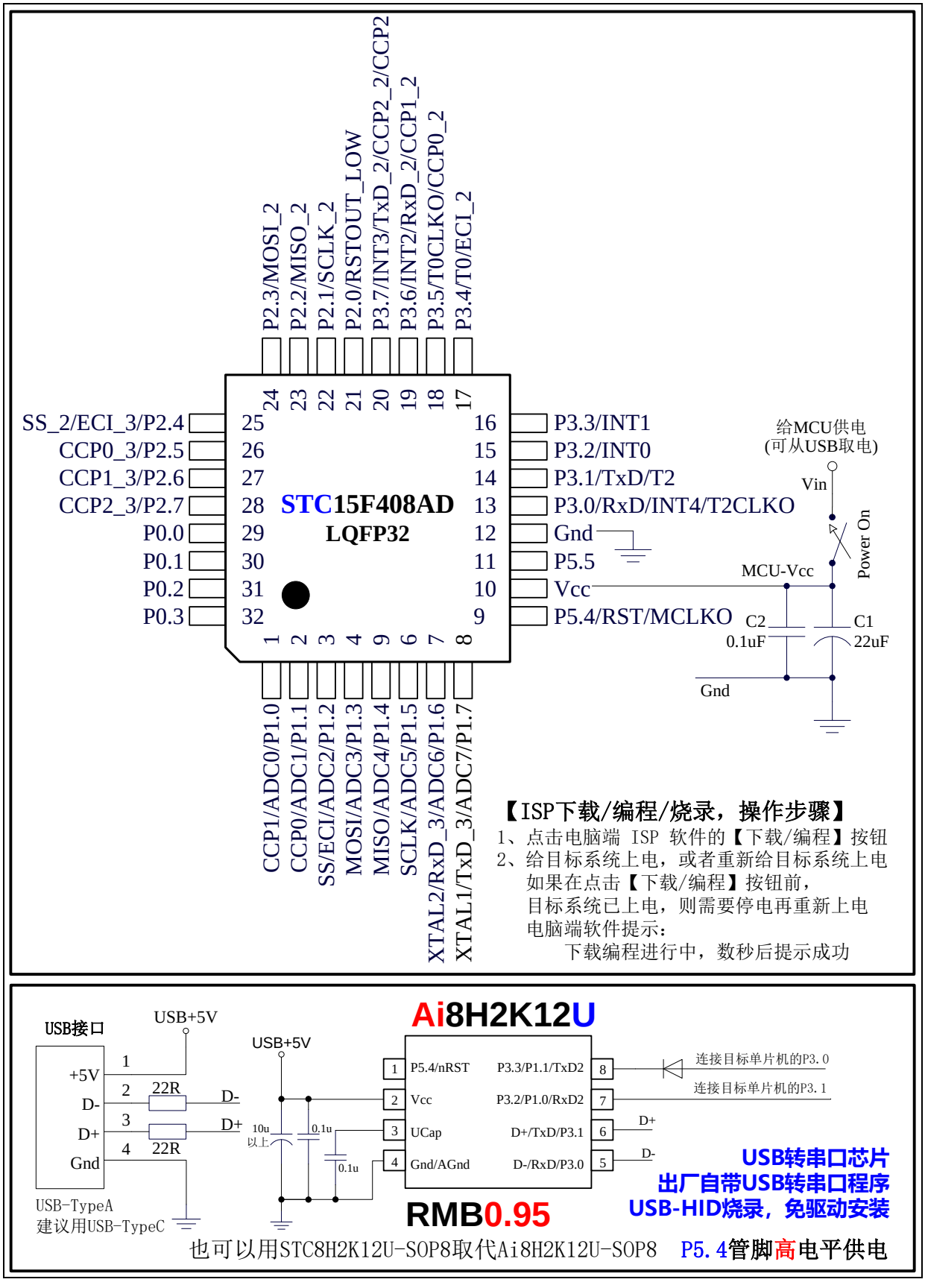
用户可将用户程序区的程序 FLASH 当 EEPROM 使用。该单片机为 IT 8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 5.5V ~ 4.2V，SRAM 空间大小为 512 字节，程序空间大小为 13K，有 1 组高速异步串行通信端口 UART 及 SPI、内部 EEPROM、A/D 转换、CCP/PCAPWM 功能，工作频率可到 28MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 SOP 贴片封装，管脚数为 28。

4) IAP15L413AD-28I-SOP28 表示：

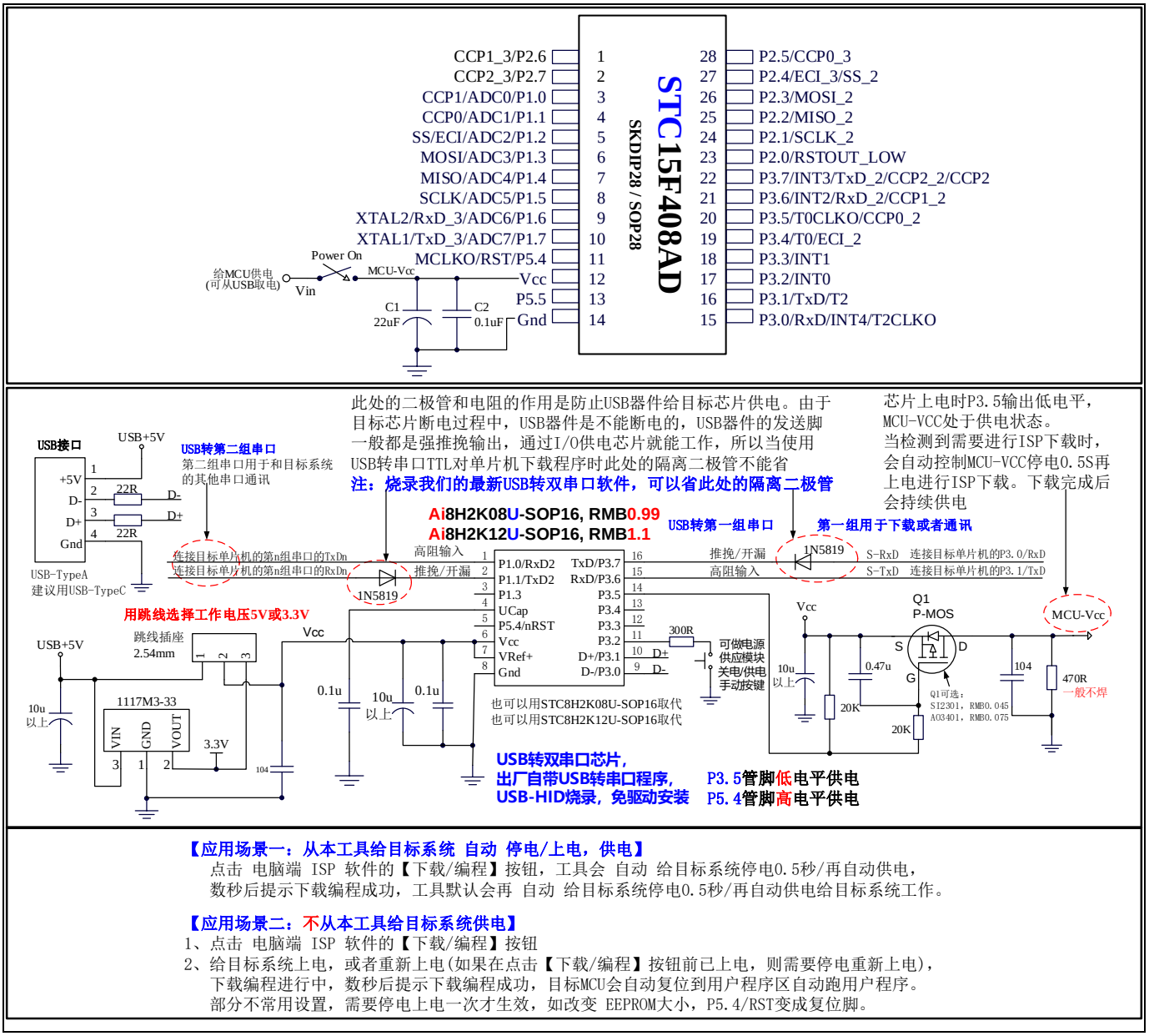
用户可将用户程序区的程序 FLASH 当 EEPROM 使用。该单片机为 1T8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 2.4V ~ 3.6V，SRAM 空间大小为 512 字节，程序空间大小为 13K，有 1 组高速异步串行通信端口 UART 及 SPI、内部 EEPROM、A/D 转换、CCP/PCA/PWM 功能，工作频率可到 28MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 SOP 贴片封装，管脚数为 28。

3.6.5 管脚图及下载线路图

3.6.5.1 管脚图及下载线路图，最小系统（LQFP32）



3.6.5.2 管脚图及下载线路图，最小系统（SKDIP28/SOP28）



【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，P5.4/RST变成复位脚。

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000, x00x
CLK_DIV （PCON2）	97H	时钟分频 寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	Tx2_Rx2	CLKS2	CLKS1	CLKS0	0000, x000

串口 1/S1 可在 3 个地方切换，由 S1_S0 及 S1_S1 控制位来选择		
S1_S1	S1_S0	串口 1/ S 1 可在 P1/P3 之间来回切换
0	0	串口 1/ S 1 在[P3.0/RxD，P3.1/TxD]
0	1	串口 1/ S 1 在[P3.6/RxD_2，P3.7/TxD_2]
1	0	串口 1/S1 在[P1.6/RxD_3/XTAL2，P1.7/TxD_3/XTAL1] 串口 1 在 P1 口时要使用内部时钟
1	1	无效

串口 1 建议放在[P3.6/RxD_2，P3.7/TxD_2]或[P1.6/RxD_3/XTAL2，P1.7/TxD_3/XTAL1]上。

建议用户在程序中将[S1_S1，S1_S0]的值设置为[0，1]或[1，0]，）进而将串口 1 放在[P3.6/RxD_2，P3.7/TxD_2]或[P1.6/RxD_3/XTAL2，P1.7/TxD_3/XTAL1]上

CCP 可在 3 个地方切换，由 CCP_S1 / CCP_S0 两个控制位来选择		
CCP_S1	CCP_S0	CCP 可在 P1/P2/P3 之间来回切换
0	0	CCP 在[P1.2/ECI，P1.1/CCP0，P1.0/CCP1，P3.7/CCP2]
0	1	CCP 在[P3.4/ECI_2，P3.5/CCP0_2，P3.6/CCP1_2，P3.7/CCP2_2]
1	0	CCP 在[P2.4/ECI_3，P2.5/CCP0_3，P2.6/CCP1_3，P2.7/CCP2_3]
1	1	无效

SPI 可在 2 个地方切换，由 SPI_S0 两个控制位来选择		
SPI_S1	SPI_S0	SPI 可在 P1/P2 之间来回切换
0	0	SPI 在[P1.2/SS，P1.3/MOSI，P1.4/MISO，P1.5/SCLK]
0	1	SPI 在[P2.4/SS_2，P2.3/MOSI_2，P2.2/MISO_2，P2.1/SCLK_2]
1	0	SPI 在[P5.4/SS_3，P4.0/MOSI_3，P4.1/MISO_3，P4.3/SCLK_3]
1	1	无效

MCKO_S1	MCKO_S0	主时钟对外分频输出控制位 （主时钟可对外输出内部 R/C 时钟，也可对外输出外部输入的时钟或外部晶体振荡产生的时钟）
0	0	主时钟不对外输出时钟
0	1	主时钟对外输出时钟，但时钟频率不被分频，输出时钟频率 = MCLK / 1
1	0	主时钟对外输出时钟，但时钟频率被 2 分频，输出时钟频率 = MCLK / 2
1	1	主时钟对外输出时钟，但时钟频率被 4 分频，输出时钟频率 = MCLK / 4

主时钟对外输出管脚 P5.4/MCLKO 既可对外输出内部 R/C 时钟，也可对外输出外部输入的时钟或外部晶体振荡产生的时钟，MCLK 是指主时钟频率。

STC15F408AD 系列单片机在 MCLKO/P5.4 口对外输出时钟。

STC15 系列 8-pin 单片机（如 STC15F104W 系列）在 MCLKO/P3.4 口对外输出时钟，STC15 系列 16-pin 及其以上单片机（如 STC15W4K32S4 系列）均在 MCLKO/P5.4 口对外输出时钟。

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000, x00x
CLK_DIV （PCON2）	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	Tx2_Rx2	CLKS2	CLKS1	CLKS0	0000, x000

ADRJ：ADC 转换结果调整

0：ADC_RES[7：0]存放高 8 位 ADC 结果，ADC_RESL[1：0]存放低 2 位 ADC 结果

1：ADC_RES[1：0]存放高 2 位 ADC 结果，ADC_RESL[7：0]存放低 8 位 ADC 结果

Tx_Rx：串口 1 的中继广播方式设置

0：串口 1 为正常工作方式

1：串口 1 为中继广播方式，即将 RxD 端口输入的电平状态实时输出在 TxD 外部管脚上，TxD 外部管脚可以对 RxD 管脚的输入信号进行实时整形放大输出，TxD 管脚的对外输出实时反映 RxD 端口输入的电平状态。

串口 1 的 RxD 管脚和 TxD 管脚可以在 3 组不同管脚之间进行切换：[RxD/P3.0，TxD/P3.1]；
[RxD_2/P3.6，TxD_2/P3.7]；

[RxD_3/P1.6，TxD_3/P1.7]

CLKS2	CLKS1	CLKS0	系统时钟选择控制位 （系统时钟是指对主时钟进行分频后供给 CPU、串行口、SPI、定时器、CCP/PWM/PCA、A/D 转换的实际工作时钟）
0	0	0	主时钟频率/1，不分频
0	0	1	主时钟频率/2
0	1	0	主时钟频率/4
0	1	1	主时钟频率/8
1	0	0	主时钟频率/16
1	0	1	主时钟频率/32
1	1	0	主时钟频率/64
1	1	1	主时钟频率/128

主时钟对外输出管脚 P5.4/CLKO 既可对外输出内部 R/C 时钟，也可对外输出外部输入的时钟或外部晶体振荡产生的时钟。

3.6.6 STC15F408AD 系列单片机的管脚说明

管脚	管脚编号		说明	
	SOP28/ SKDIP28	LQFP32		
P0.0		29	标准 I/O 口 PORT0[0]	
P0.1		30	标准 I/O 口 PORT0[1]	
P0.2		31	标准 I/O 口 PORT0[2]	
P0.3		32	标准 I/O 口 PORT0[3]	
P1.0	3	1	P1.0	标准 I/O 口 PORT1[0]
			ADC0	ADC 输入通道-0
			CCP1	外部信号捕获(频率测量或当外部中断使用)、高速脉冲输出及脉宽调制输出通道-1
P1.1	4	2	P1.1	标准 I/O 口 PORT1[1]
			ADC1	ADC 输入通道-1
			CCP0	外部信号捕获(频率测量或当外部中断使用)、高速脉冲输出及脉宽调制输出通道-0
P1.2	5	3	P1.2	标准 I/O 口 PORT1[2]
			ADC2	ADC 输入通道-2
			SS	SPI 同步串行接口的从机选择信号
			ECI	CCP / PCA 计数器的外部脉冲输入脚
P1.3	6	4	P1.3	标准 I/O 口 PORT1[3]
			ADC3	ADC 输入通道-3
			MOSI	SPI 同步串行接口的主出从入(主器件的输出和从器件的输入)
P1.4	7	5	P1.4	标准 I/O 口 PORT1[4]
			ADC4	ADC 输入通道-4
			MISO	SPI 同步串行接口的主入从出(主器件的输入和从器件的输出)
P1.5	8	6	P1.5	标准 I/O 口 PORT1[5]
			ADC5	ADC 输入通道-5
			SCLK	SPI 同步串行接口的时钟信号
P1.6	9	7	P1.6	标准 I/O 口 PORT1[6]
			ADC6	ADC 输入通道-6
			RxD_3	串口数据接收端
			XTAL2	内部时钟电路反相放大器的输出端，接外部晶振的其中一端。 当直接使用外部时钟源时，此引脚可浮空，此时 XTAL2 实际将 XTAL1 输入的时钟进行输出。
P1.7	10	8	P1.7	标准 I/O 口 PORT1[7]
			ADC7	ADC 输入通道-7
			TxD_3	串口数据发送端
			XTAL1	内部时钟电路反相放大器输入端，接外部晶振的其中一端。 当直接使用外部时钟源时，此引脚是外部时钟源的输入端。
P2.0	23	21	P2.0	标准 I/O 口 PORT2[0]
			RSTOUT_LOW	上电后，输出低电平，在复位期间也是输出低电平，用户可用软件将其设置为高电平或低电平，如果要读外部状态，可将该口先置高后再读
P2.1	24	22	P2.1	标准 I/O 口 PORT2[1]
			SCLK_2	SPI 同步串行接口的时钟信号

管脚	管脚编号		说明	
	SOP28/ SKDIP28	LQFP32		
P2.2	25	23	P2.2	标准 I/O 口 PORT2[2]
			MISO_2	SPI 同步串行接口的主入从出(主器件的输入和从器件的输出)
P2.3	26	24	P2.3	标准 I/O 口 PORT2[3]
			MOSI_2	SPI 同步串行接口的主出从入(主器件的输出和从器件的输入)
P2.4	27	25	P2.4	标准 I/O 口 PORT2[4]
			ECI_3	CCP / PCA 计数器的外部脉冲输入脚
			SS_2	SPI 同步串行接口的从机选择信号
P2.5	28	26	P2.5	标准 I/O 口 PORT2[5]
			CCP0_3	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-0
P2.6	1	27	P2.6	标准 I/O 口 PORT2[6]
			CCP1_3	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-1
P2.7	2	28	P2.7	标准 I/O 口 PORT2[7]
			CCP2_3	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-2
P3.0	15	13	P3.0	标准 I/O 口 PORT3[0]
			RxD	串口 1 数据接收端
			INT4	外部中断 4，只能下降沿中断，INT4 支持掉电唤醒
			T2CLKO	T2 的时钟输出 可通过设置 INT_CLKO[2]位/T2CLKO 将该管脚配置为 T2CLKO
P3.1	16	14	P3.1	标准 I/O 口 PORT3[1]
			TxD	串口数据发送端
			T2	定时器/计数器 2 的外部输入
P3.2	17	15	P3.2	标准 I/O 口 PORT3[2]
			INT0	外部中断 0，既可上升沿中断也可下降沿中断. 如果 IT0(TCON.0)被置为 1，INT0 管脚仅为下降沿中断。 如果 IT0(TCON.0)被清 0，INT0 管脚既支持上升沿中断也支持下降沿中断。 INT0 支持掉电唤醒。
P3.3	18	16	P3.3	标准 I/O 口 PORT3[3]
			INT1	外部中断 1，既可上升沿中断也可下降沿中断. 如果 IT1(TCON.2)被置为 1，INT1 管脚仅为下降沿中断。 如果 IT1(TCON.2)被清 0，INT1 管脚既支持上升沿中断也支持下降沿中断。 INT1 支持掉电唤醒。
P3.4	19	17	P3.4	标准 I/O 口 PORT3[4]
			T0	定时器/计数器 0 的外部输入
			ECI_2	CCP/PCA 计数器的外部脉冲输入脚
P3.5	20	18	P3.5	标准 I/O 口 PORT3[5]
			T0CLKO	定时器/计数器 0 的时钟输出 可通过设置 INT_CLKO[0]位/T0CLKO 将该管脚配置为 T0CLKO，也可对 T0 脚的外部时钟输入进行分频输出
			CCP0_2	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-0
P3.6	21	19	P3.6	标准 I/O 口 PORT3[6]
			INT2	外部中断 2，只能下降沿中断 INT2 支持掉电唤醒
			RxD_2	串口数据接收端
			CCP1_2	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-1

管脚	管脚编号		说明	
	SOP28/ SKDIP28	LQFP32		
P3.7	22	20	P3.7	标准 I/O 口 PORT3[7]
			INT3	外部中断 3，只能下降沿中断 INT3 支持掉电唤醒
			TxD_2	串口数据发送端
			CCP2	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-2
			CCP2_2	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-2
P5.4	11	9	P5.4	标准 I/O 口 PORT5[4]
			RST	复位脚(高电平复位)
			MCLKO	主时钟输出：输出的频率可为 MCLK/1，MCLK/2，MCLK/4(MCLK 是指主时钟频率)。 主时钟外部输出管脚 P5.4/MCLKO 既可外部输出内部 R/C 时钟，也可外部输出外部输入的时钟或外部晶体振荡产生的时钟，MCLK 指主时钟频率。
P5.5	13	11	标准 I/O 口 PORT5[5]	
Vcc	12	10	电源正极	
Gnd	14	12	电源负极，接地	

3.6.7 USB-Link1D 工具自动停电/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】

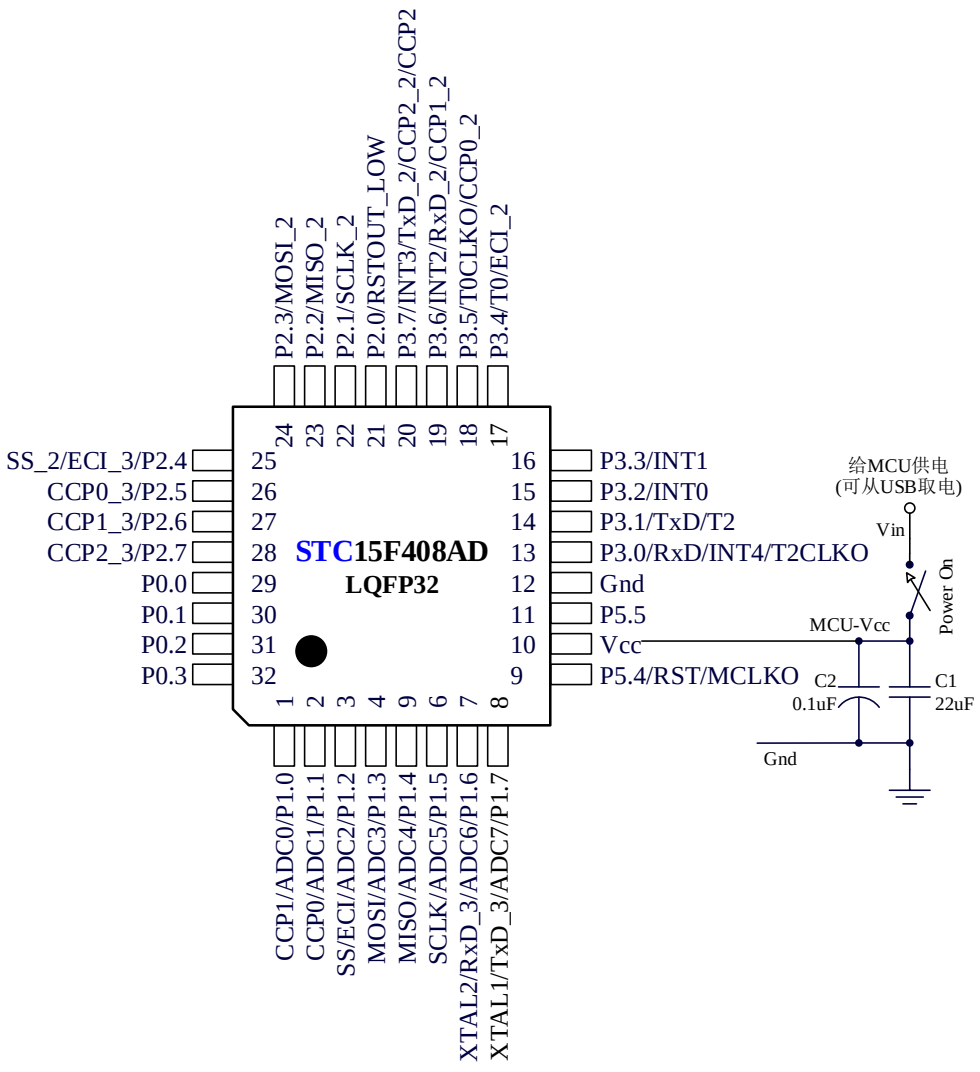
点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【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，P5.4/nRST变成复位脚



3.6.8 【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】

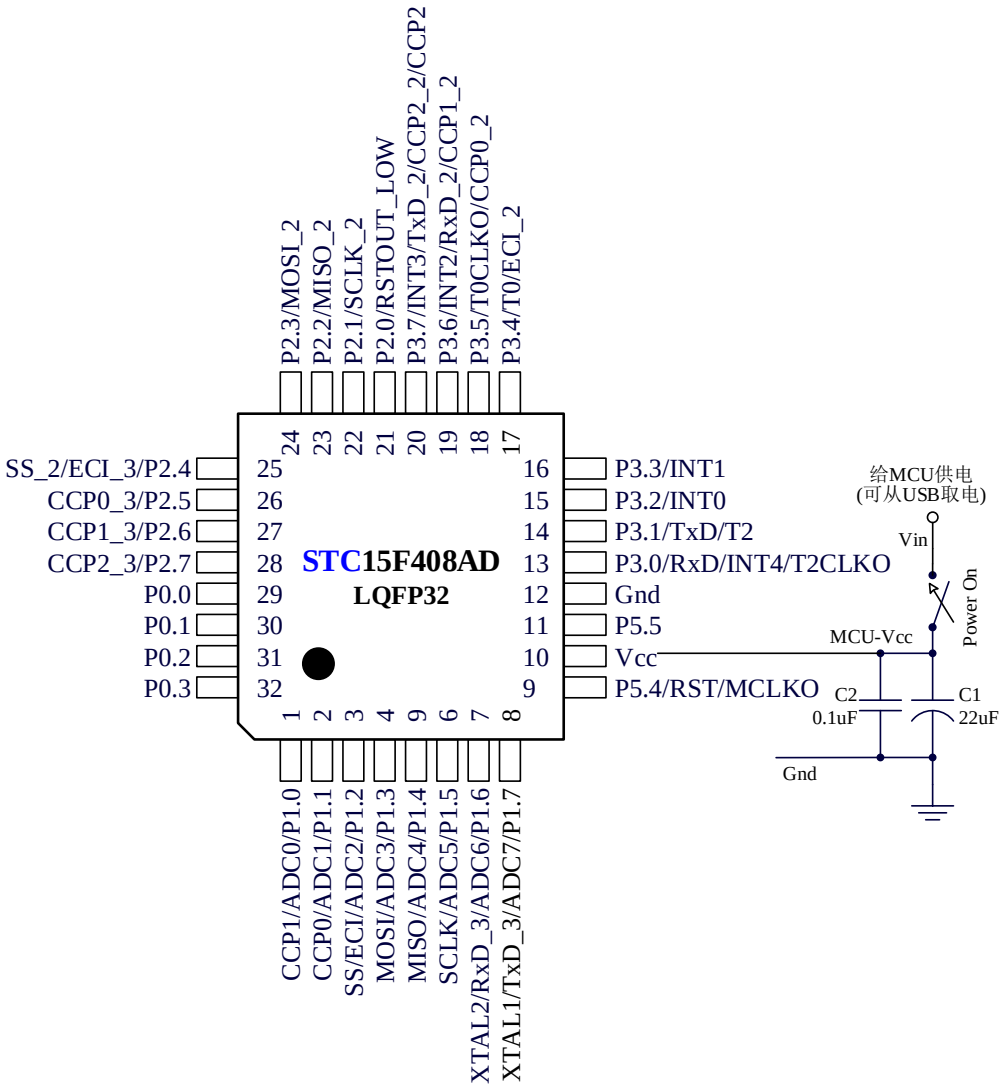
点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【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



3.7 STC15W4K32S4 系列

STC15W4K32S4 系列 1T 8051 单片机，4K 字节 SRAM，超高速四串口，6 路 15 位 PWM
不需外部晶振的单片机，不需外部复位的单片机

STC15W4K32S4 系列主要性能

- ✓ 大容量 4096 字节片内 RAM 数据存储器
- ✓ 高速：1 个时钟/机器周期，增强型 8051 内核，速度比传统 8051 快 7 ～ 12 倍
- ✓ 速度也比 STC 早期的 1T 系列单片机（如 STC12/11/10 系列）的速度快 20%
- ✓ 宽电压：2.5V ～ 5.5V
- ✓ 低功耗设计：低速模式，空闲模式，掉电模式（可由外部中断或内部掉电唤醒定时器唤醒）
- ✓ 不需外部复位的单片机，ISP 编程时 16 级复位门槛电压可选，内置高可靠复位电路
- ✓ 不需外部晶振的单片机，ISP 编程时内部时钟从 5MHz ~ 30MHz 可设（相当于普通 8051：60 ～ 360MHz）
- ✓ 内部高精度 R/C 时钟（±0.3%），±1%温漂（-40℃～+85℃），常温下温漂±0.6%（-20℃～+65℃）
- ✓ 支持掉电唤醒的资源有：INT0/INT1（上升沿/下降沿中断均可），INT2/INT3/INT4（下降沿中断）；CCP0/CCP1/RxD/RxD2/RxD3/RxD4/T0/T1/T2/T3/T4 管脚；内部掉电唤醒专用定时器
- ✓ 16/32/40/48/56/58K/61K/63.5K 字节片内 Flash 程序存储器，擦写次数 10 万次以上
- ✓ 大容量片内 EEPROM 功能，擦写次数 10 万次以上
- ✓ ISP/IAP，在系统可编程/在应用可编程，无需编程器/仿真器
- ✓ 高速 ADC，8 通道 10 位，速度可达 30 万次/秒，8 路 PWM 还可当 8 路 D/A 使用
- ✓ 比较器，可当 1 路 ADC 使用，并可作掉电检测，支持外部管脚 CMP+与外部管脚 CMP-进行比较，可产生中断，并可在管脚 CMPO 上产生输出（可设置极性），也支持外部管脚 CMP+与内部参考电压进行比较
- ✓ 6 通道 15 位专门的高精度 PWM（带死区控制） + 2 通道 CCP（利用它的高速脉冲输出功能可实现 2 路 11 ~ 16 位 PWM）
  ----可用来再实现 8 路 D/A，或 2 个 16 位定时器，或 2 个外部中断（支持上升沿/下降沿中断）
- ✓ 共 7 个定时器/计数器，5 个 16 位可重装载定时器/计数器（T0/T1/T2/T3/T4，其中 T0 和 T1 兼容普通 8051 的定时器/计数器），并均可实现时钟输出，另外管脚 SysClkO 可将系统时钟对外分频输出（÷1 或÷2 或÷4 或÷16），2 路 CCP 可再实现 2 个定时器
- ✓ 可编程时钟输出功能（对内部系统时钟或外部管脚的时钟输入进行时钟分频输出）：
  - ① T0 在 P3.5 输出时钟；
  - ② T1 在 P3.4 输出时钟；
  - ③ T2 在 P3.0 输出时钟；
  - ④ T3 在 P0.4 输出时钟；
  - ⑤ T4 在 P0.6 输出时钟，以上 5 个定时器/计数器输出时钟均可 1 ~ 65536 级分频输出；
  - ⑥ 系统时钟在 P5.4/SysClkO 对外输出时钟（STC15 系列 8-pin 单片机的主时钟在 P3.4/MCLKO

对外输出时钟）

- ✓ 超高速四串口/UART，四个完全独立的高速异步串行通信端口，分时切换可当 9 组串口使用
- ✓ SPI 高速同步串行通信接口
- ✓ 硬件看门狗（WDT）
- ✓ 先进的指令集结构，兼容普通 8051 指令集，有硬件乘法/除法指令
- ✓ 通用 I/O 口（62/46/42/38/30/26 个），复位后为：准双向口/弱上拉（8051 传统 I/O 口）。可设置四种模式：准双向口/弱上拉，强推挽/强上拉，仅为输入/高阻，开漏。每个 I/O 口驱动能力均可达到 20mA，但整个芯片最大不要超过 120mA。

选择【STC15W4K32S4】系列单片机理由：

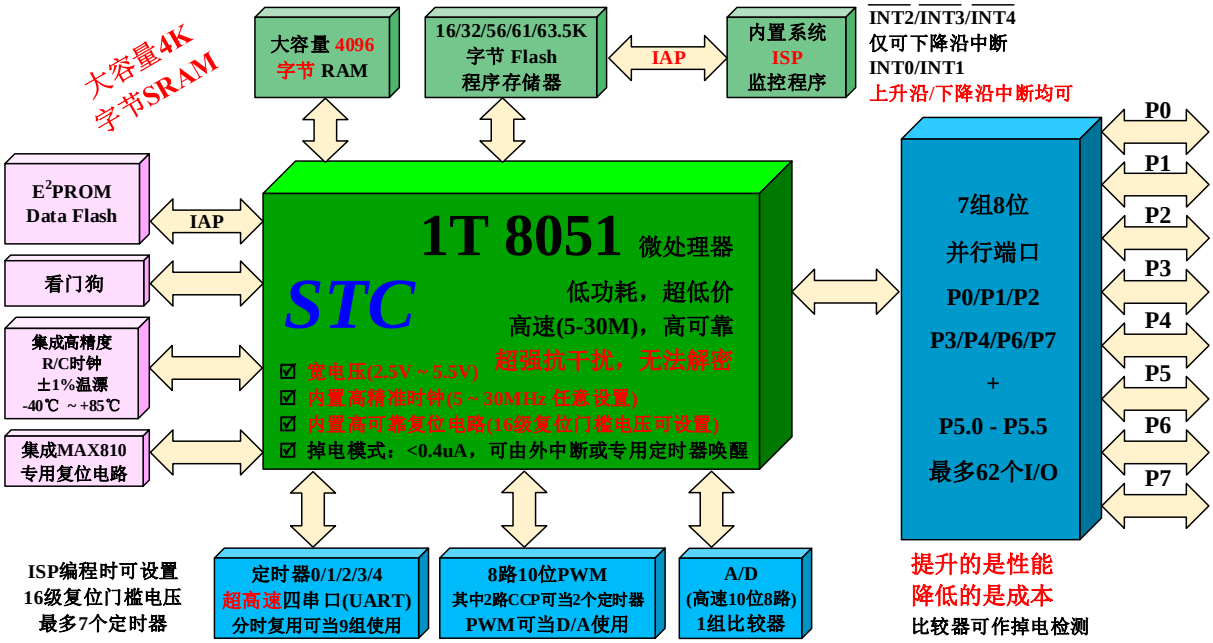
1. 不需外部晶振和外部复位，还可对外输出时钟和低电平复位信号
2. 片内大容量 4096 字节 SRAM
3. 6 通道 15 位专门的高精度 PWM（带死区控制） + 2 通道 CCP （利用它的高速脉冲输出功能可实现 2 路 11~16 位 PWM）
   ----可用来再实现 8 路 D/A，或 2 个 16 位定时器，或 2 个外部中断（支持上升沿/下降沿中断）
4. 无法解密，STC 第九代加密技术，现悬赏 20 万元人民币请专家帮忙查找加密有无漏洞
5. 超强抗干扰：
   - ➢ 高抗静电（ESD 保护）整机轻松过 2 万伏静电测试
   - ➢ 轻松过 4kV 快速脉冲干扰（EFT 测试）
   - ➢ 宽电压，不怕电源抖动
   - ➢ 宽温度范围，-40℃ ~ +85℃
6. 大幅降低 EMI，内部可配置时钟，1 个时钟/机器周期，可用低频时钟
   ----出口欧美的有力保证
7. 超低功耗：
   - ➢ 掉电模式：外部中断唤醒功耗 <0.4uA
   - ➢ 空闲模式：典型功耗 <1mA
   - ➢ 正常工作模式：4mA ~ 6mA
   - ➢ 掉电模式可由外部中断或内部掉电唤醒专用定时器唤醒，适用电池供电系统，如水表、气表等
8. 在系统可仿真，在系统可编程，无需专用编程器，无需专用仿真器，可远程升级
9. 可送 USB 型联机/脱机下载烧录工具 STC-U8W（人民币 100 元），1 万片/人/天，有自动烧录机接口

3.7.1 STC15W4K32S4 系列简介

STC15W4K32S4 系列单片机是 STC 生产的单时钟/机器周期（1T）的单片机，是宽电压/高速/高可靠/低功耗/超强抗干扰的新一代 8051 单片机，采用 STC 第九代加密技术，无法解密，指令代码完全兼容传统 8051，但速度快 8-12 倍。内部集成高精度 R/C 时钟（±0.3%），±1%温漂（-40℃～+85℃），常温下温漂±0.6%（-20℃～+65℃），ISP 编程时 5MHz～30MHz 宽范围可设置，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 16 级复位门槛电压可选）。8 路 10 位 PWM，8 路高速 10 位 A/D 转换（30 万次/秒），内置 4K 字节大容量 SRAM，4 组独立的高速异步串行通信端口（UART/UART2/UART3/UART4），1 组高速同步串行通信端口 SRI，针对多串行口通信/电机控制干扰场合。内置比较器，功能更强大。

在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可。

现 STC15 系列单片机采用 STC-Y5 超高速 CPU 内核，在相同的时钟频率下，速度又比 STC 早期的 1T 系列单片机（如 STC12 系列/STC11 系列/STC10 系列）的速度快 20%。



1 增强型 8051 CPU，1T，单时钟/机器周期，速度比普通 8051 快 8-12 倍

2 工作电压：2.5V -- 5.5V

3 16K/32K/40K/48K/56K/58K/61K/63.5K 字节片内 Flash 程序存储器，擦写次数 10 万次以上

4 片内大容量 4096 字节的 SRAM，包括常规的 256 字节 RAM <idata>和内部扩展的 3840 字节 XRAM <xdata>

5 大容量片内 EEPROM，擦写次数 10 万次以上

6 ISP/IAP，在系统可编程/在应用可编程，无需编程器/仿真器

7 共 8 通道 10 位高速 ADC，速度可达 30 万次/秒，8 路 PWM 还可当 8 路 D/A 使用

8 6 通道 15 位专门的高精度 PWM（带死区控制）+2 通道 CCP（利用它的高速脉冲输出功能可实现 11~16 位 PWM

--可用来再实现 8 路 D/A，或 2 个 16 位定时器，或 2 个外部中断（支持上升沿/下隆沿中断）

9 与 STC15W4K32S4 系列单片机的 6 路增强型 PWM 相关的 12 个端口[P3.7/PWM2，P2.1/PWM3，P2.2/PWM4，P2.3/PWM5，P1.6/PWM6，P1.7/PWM7，P2.7/PWM2_2，P4.5/PWM3_2，P4.4/PWM4_2，P4.2/PWM5_2，P0.7/PWM6_2，P0.6/PWM7_2]上电复位前要进行初始化。因为这些端口上电复位后默认为高阻输入（既不向外输出电流也不向内输出电流），若要使其能对外能输出，要用软件将其改设为强推挽输出或准双向口上拉，因此上电前用户须在程序中将这些端口设置为其他模式（如准双向口或强推挽模式）；注意这些端口进入掉电模式时不能为高阻输入，否则需外部加上拉电阻。

10 内部高可靠复位，ISP 编程时 16 级复位门槛电压可选，可彻底省掉外部复位电路

11 工作频率范围：5MHz~30MHz，相当于普通 8051 的 60MHz~360MHz

12 内部高精度 R/C 时钟（±0.3%），±1%温漂（-40℃ ~ +85℃），常温下温漂±0.6%（-20℃ ~ +65℃），ISP 编程时内部时钟从 5MHZ~30MHz 可设（5.5296MHz / 6MHz / 11.0592MHz / 12MHz / 18.432MHz / 20MHz / 22.1184MHz / 24MHz / 27MHz / 30MHz）

13 不需外部晶振和外部复位，还可对外输出时钟和低电平复位信号

14 四组完全独立的高速异步串行通信端口，分时切换可当 9 组串口使用

- 串口 1（RxD/P3.0，TxD/P3.1）可以切换到（RxD_2/P3.6，TxD_2/P3.7），还可以切换到（RxD_3/P1.6，TxD_3/P1.7）；
- 串口 2（RxD2/P1.0，TxD2/P1.1）可以切换到（RxD2_2/P4.6，TxD2_2/P4.7）
- 串口 3（RxD3/P0.0，TxD3/P0.1）可以切换到（RxD3_2/P5.0，TxD3_2/P5.1）
- 串口 4（RxD4/P0.2，TxD4/P0.3）可以切换到（RxD4_2/P5.2，TxD4_2/P5.3）

注意：建议用户将串口 1 放在 P3.6/P3.7 或 P1.6/ P1.7（P3.0/P3.1 作下载/仿真用）；若用户不想切换，坚持使用 P3.0/P3.1 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”。

15 一组高速同步串行通信端口 SPI

16 支持程序加密后传输，防拦截

17 支持 RS485 下载

18 低功耗设计：低速模式，空闲模式，掉电模式/停机模式

19 可将掉电模式/停机模式唤醒的定时器：有内部低功耗掉电唤醒专用定时器

20 可将掉电模式/停机模式唤醒的资源有：

- INT0/P3.2，INT1/P3.3 （INT0/INT1 上升沿下降沿中断均可），INT2/P3.6，INT3/P3.7，INT4/P3.0（INT2/INT3/INT4 仅可下降沿中断）；
- 管脚 CCP0/CCP1；
- 外部管脚 RxD/RxD2/RxD3/RxD4（下降沿，不产生中断，前提是在进入掉电模式/停机模式前相应的串行口中断已经被允许）；
- 外部管脚 T0/T1/T2/T3/T4（下降沿，不产生中断，前提是在进入掉电模式/停机模式前相应的定时器中断已经被允许）；
- 内部低功耗掉电唤醒专用定时器。

21 共 7 个定时器，5 个 16 位可重装载定时器/计数器（T0/T1/T2/T3/T4，其中 T0/T1 兼容普通 8051 的定时器/计数器），并均可独立实现对外可编程时钟输出（5 通道），另外管脚 SysClkO 可将系统时钟

对外分频输出（÷1 或÷2 或÷4 或÷16），2 路 CCP 还可再实现 2 个定时器

22 定时器/计数器 2，也可实现 1 个 16 位重装载定时器/计数器，定时器/计数器 2 也可产生时钟输出 T2CLKO

23 新增可 16 位重装载定时器 T3/T4，也可产生可编程时钟输出 T3CLKO/T4CLKO

24 可编程时钟输出功能（对内部系统时钟或对外部管脚的时钟输入进行时钟分频输出）：

- 由于 STC15 系列 5V 单片机 I/O 口的对外输出速度最快不超过 13.5MHz，所以 5V 单片机的对外可编程时钟输出速度最快也不超过 13.5MHz；
- 而 3.3V 单片机 I/O 口的对外输出速度最快不超过 8MHz，故 3.3V 单片机的对外可编程时钟输出速度最快也不超过 8MHz。

1) T0 在 P3.5/T0CLKO 进行可编程输出时钟（对内部系统时钟或对外部管脚 T0/P3.4 的时钟输入进行可编程时钟分频输出）；
2) T1 在 P3.4/T1CLKO 进行可编程输出时钟（对内部系统时钟或对外部管脚 T1/P3.5 的时钟输入进行可编程时钟分频输出）；
3) T2 在 P3.0/T2CLKO 进行可编程输出时钟（对内部系统时钟或对外部管脚 T2/P3.1 的时钟输入进行可编程时钟分频输出）；
4) T3 在 P0.4/T3CLKO 进行可编程输出时钟（对内部系统时钟或对外部管脚 T3/P0.5 的时钟输入进行可编程时钟分频输出）；
5) T4 在 P0.6/T4CLKO 进行可编程输出时钟（对内部系统时钟或对外部管脚 T4/P0.7 的时钟输入进行可编程时钟分频输出）；
   以上 5 个定时器/计数器均可 1 ~ 65536 级分频输出。
6) 系统时钟在 P5.4/SysClkO 或 P1.6/XTAL2/SysClkO_2 对外输出时钟，并可如下分频 SysClk/1，SysClk/2，SysClk/4，SysClk/16。

系统时钟是指对主时钟进行分频后供给 CPU、定时器、串行口、SPI、CCP/PWM/PCA、A/D 转换的实际工作时钟；主时钟可以是内部 R/C 时钟，也可以是外部输入的时钟或外部晶体振荡产生的时钟；SysClk 是指系统时钟频率，SysClkO 是指系统时钟输出。

STC15 系列中除 STC15W4K32S4 系列、STC15W408AS 系列、STC15W1K08PWM 系列及 STC15W1K20S-LQFP64 单片机是将系统时钟对外分频输出外，其他系列单片机均是将主时钟对外分频输出。

25 比较器，可当 1 路 ADC 使用，可作掉电检测，支持外部管脚 CMP+与外部管脚 CMP-进行比较，可产生中断，并可在管脚 CMPO 上产生输出（可设置极性），也支持外部管脚 CMP+与内部参考电压进行比较。

若[P5.5/CMP+，P5.4/CMP-]被用作比较器正极（CMP+）/负极（CMP-），则[P5.5/CMP+，P5.4/CMP-]要被设置为高阻输入。

注意：STC15W4K32S4 系列单片机的 8 路 ADC 口不可用作比较器正极（CMP+）。

26 硬件看门狗（WDT）

27 先进的指令集结构，兼容普通 8051 指令集，有硬件乘法/除法指令

28 通用 I/O 口（62/46/42/38/30/26 个），复位后为：准双向口上拉（普通 8051 传统 I/O 口）。可设置成四种模式：准双向口上拉，强推挽上拉，仅为输入/高阻，开漏每个 I/O 口驱动能力均可达到 20mA。但 40-pin 及 40-pin 以上单片机的整个芯片电流最大不要超过 120mA，16-pin 及以上/32-pin 及以下单

片机的整个芯片电流最大不要超过 90mA。如果 I/O 口不够用，可外接 74HC595（参考价 0.15 元）来扩展 I/O 口，并可多芯片级联扩展几十个 I/O 口。

29 封装：LQFP64L（16mm×16mm），LQFP64S（12mm×12mm），QFN64（9mm×9mm），

LQFP48（9mm×9mm），QFN48（7mm×7mm），LQFP44（12mm×12mm），

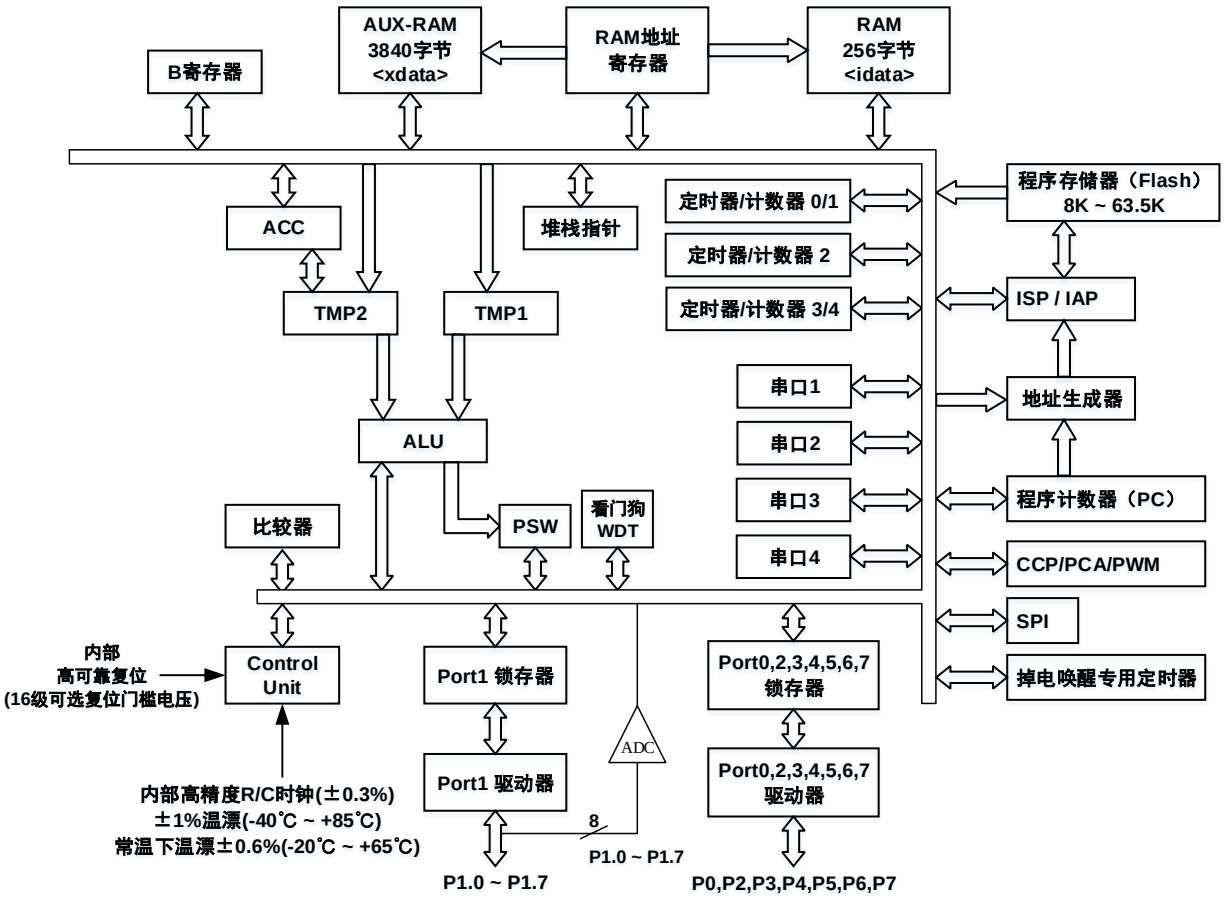
LQFP32（9mm×9mm），SOP28，SKDIP28，PDIP40

30 全部 175℃八小时高温烘烤，高品质制造保证

31 开发环境：在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可。

3.7.2 内部结构图

STC15W4K32S4 系列单片机的内部结构框图如下图所示。STC15W4K32S4 系列单片机中包含中央处理器(CPU)、程序存储器(Flash)、数据存储器(SRAM)、定时器/计数器、掉电唤醒专用定时器、I/O 口、高速 A/D 转换、比较器、看门狗、UART 高速异步串行通信口 1、串行口 2、串行口 3、串行口 4、CCP/PWM/PCA、高速同步串行通信端口 SPI，片内高精度 R/C 时钟及高可靠复位等模块。STC15W4K32S4 系列单片机几乎包含了数据采集和控制中所需要的所有单元模块，可称得上是一个真正的片上系统(SysTem Chip 或 SysTem on Chip，简写为 STC，这是 STC 名称的由来)。



STC15W4K32S4系列内部结构框图

3.7.3 STC15W4K32S4 系列选型一览表

型号	工作电压（V）	Flash 程序存储器 (byte)	大容量 SRAM 字节	串行口并可掉电唤醒	SPI	普通定时器计数器 T0-T4 外部管脚也能掉电唤醒	8 路 PWM		掉电唤醒专用定时器	标准外部中断支持掉电唤醒	A/D8 路(8 路 PWM 可当 8 路 D/A 使用)	比较器(可当 1 路 ADC 使用，可做外部掉电检测)	DPTR	EEPROM	内部低压检测中断并可掉电唤醒	看门狗	内部复位（可选复位门槛电压）	内部高精准时钟	可对外输出时钟及复位	程序加密后传输(防拦)截	可设下次更新程序需口令	支持 RS485 下载	支持 USB 直接下载	所有封装 LQFP64S/LQFP64L/QFN64/ LQFP48/QFN48 LQFP44/PDIP40 LQFP32/SOP28/SKDIP28			
							6 路 15 位专门的 PWM(带死区控制)	2 路 CCP10 位 PWM 可当外部中断并可掉电唤醒																部分封装价格(RMB ¥)			
																								PDIP40 (38 个 I/O 口)	LQFP44 (42 个 I/O 口)	LQFP48 (46 个 I/O 口)	LQFP64S (62 个 I/O 口)
STC15W4K32S4 系列单片机选型价格一览表，已开始供货 特别提醒：8 路 PWM 可当 8 路 D/A 使用,2 路 CCP 可当 2 个定时器或 2 个外部中断使用																											
STC15W4K16S4	2.5-5.5	16K	4K	4	有	5	6-ch	2-ch	有	有	10 位	√	2	42K	有	有	16 级	有	是	有	是	是	是	¥6.70	¥6.00	¥6.00	¥6.50
STC15W4K32S4	2.5-5.5	32K	4K	4	有	5	6-ch	2-ch	有	有	10 位	√	2	26K	有	有	16 级	有	是	有	是	是	是	¥6.70	¥6.00	¥6.00	¥6.50
STC15W4K40S4	2.5-5.5	40K	4K	4	有	5	6-ch	2-ch	有	有	10 位	√	2	18K	有	有	16 级	有	是	有	是	是	是	¥6.70	¥6.00	¥6.00	¥6.50
STC15W4K48S4	2.5-5.5	48K	4K	4	有	5	6-ch	2-ch	有	有	10 位	√	2	10K	有	有	16 级	有	是	有	是	是	是	¥6.70	¥6.00	¥6.00	¥6.50
STC15W4K56S4	2.5-5.5	56K	4K	4	有	5	6-ch	2-ch	有	有	10 位	√	2	2K	有	有	16 级	有	是	有	是	是	是	¥6.70	¥6.00	¥6.00	¥6.50
IAP15W4K58S4 本身就是仿真器	2.5-5.5	58K	4K	4	有	5	6-ch	2-ch	有	有	10 位	√	2	IAP	有	有	16 级	有	是	有	是	是	是	¥6.70	¥6.00	¥6.00	¥6.50
																								用户可将用户程序区的程序 Flash 当 EEPROM 使用			
IAP15W4K61S4 本身就是仿真器	2.5-5.5	61K	4K	4	有	5	6-ch	2-ch	有	有	10 位	√	2	IAP	有	有	16 级	有	是	有	是	是	否	¥6.70	¥6.00	¥6.00	¥6.50
																								用户可将用户程序区的程序 Flash 当 EEPROM 使用			
IRC15W4K63S4 默认使用外部晶振如无外部晶振则使用内部 24MHz 时钟	2.5-5.5	63.5K	4K	4	有	5	6-ch	2-ch	有	有	10 位	√	2	IAP	有	有	16 级	有	是	有	否	否	否	¥6.70	¥6.00	¥6.00	¥6.50
																								用户可将用户程序区的程序 Flash 当 EEPROM 使用			
STC15W1K08PWM 系列单片机选型价格一览表,2014 年 10 月开始供货 特别提醒：8 路 PWM 可当 8 路 D/A 使用,2 路 CCP 可当 2 个定时器或 2 个外部中断使用																								SOP28	LQFP32		
STC15W1K08PWM	2.5-5.5	8K	1K	1	有	3	6-ch	2-ch	有	有	10 位	√	1	19K	有	有	16 级	有	是	有	是	是	是	¥6.20	¥6.00		
STC15W1K16PWM	2.5-5.5	16K	1K	1	有	3	6-ch	2-ch	有	有	10 位	√	1	11K	有	有	16 级	有	是	有	是	是	是	¥6.20	¥6.00		

我们直销，所以低价，以上单价为 **10K/M** 起定量，量小每片需加 **0.1** 元，以上价格运费由客户承担，零售 **10** 片起，如对价格不满，可来电要求降价。

如果用户要用 40-pin 及以上的单片机，建议选用 LQFP44 的封装，但 PDIP40 封装仍正常供货；
如果用户要用 32-pin 单片机，建议用户选用 LQFP32 封装；
如果用户要用 28-pin 单片机，建议用户选用 S0P28 封装。

程序加密后传输：程序拥有者产品出厂时将源程序和加密钥匙一起烧录 MCU 中，以后需要升级软件时，就可将程序加密后再用“发布项目程序”功能，生成一个用户自己界面的只有一个升级按钮的简单易用的升级软件，给最终使用者自己升级，而拦截不到您的原始程序。

若[P5.5/CMP+，P5.4/CMP-]被用作比较器正极(CMP+)/负极(CMP-)，则[P5.5/CMP+，P5.4/CMP-]要被设置为高阻输入

一秒钟能运行 1000 万条指令的 STC 8051 也能做四轴飞行器，简单的四轴飞行器可采用一片 STC15W4K32S4 来完成，正真商用高端无人航拍四轴飞行器流行做法是 4 个无刷电机各用一片 STC15W404AS 控制(用到它 3 路 PW+3 通道比较器/8 路 ADC 口也可设为比较器的正极)，中央飞控系统用一片 STC15W4K48S4。

注意：STC15W4K32S4 系列单片机的 8 路 ADC 口不可用作比较器正极(CMP+)

与 STC15W4K32S4 系列单片机的 6 路增强型 PWM 相关的端口上电后默认为高阻输入，上电前用户须在程序中将这些端口设置为其他模式(如准双向口或强推挽模式)；注意这些端口进入掉电模式时不能为高阻输入，否则需外部加上拉电阻。

因为程序区的最后 7 个字节单元被强制性的放入全球唯一 1D 号的内容，所以用户实际可以使用的程序空间大小要比选型表中的大小少 7 个字节。

上表中 IRC15W4K63S4 型号的单片机内部复位门槛电压固定，同时不支持“程序加密后传输”功能，其 P5.4 不可当复位管脚 RST 使用，且 P3.2/P3.3 与下载无关。

IAP15W4K58S4 型号的单片机不能设置“P3.2/P3.3 同时为 0/0 才能下载程序”，管脚 P5.4 不可设置为复位脚 RST 使用。

【总结】：STC15W4K32S4 系列单片机(含 IRC15W4K63S4)有：

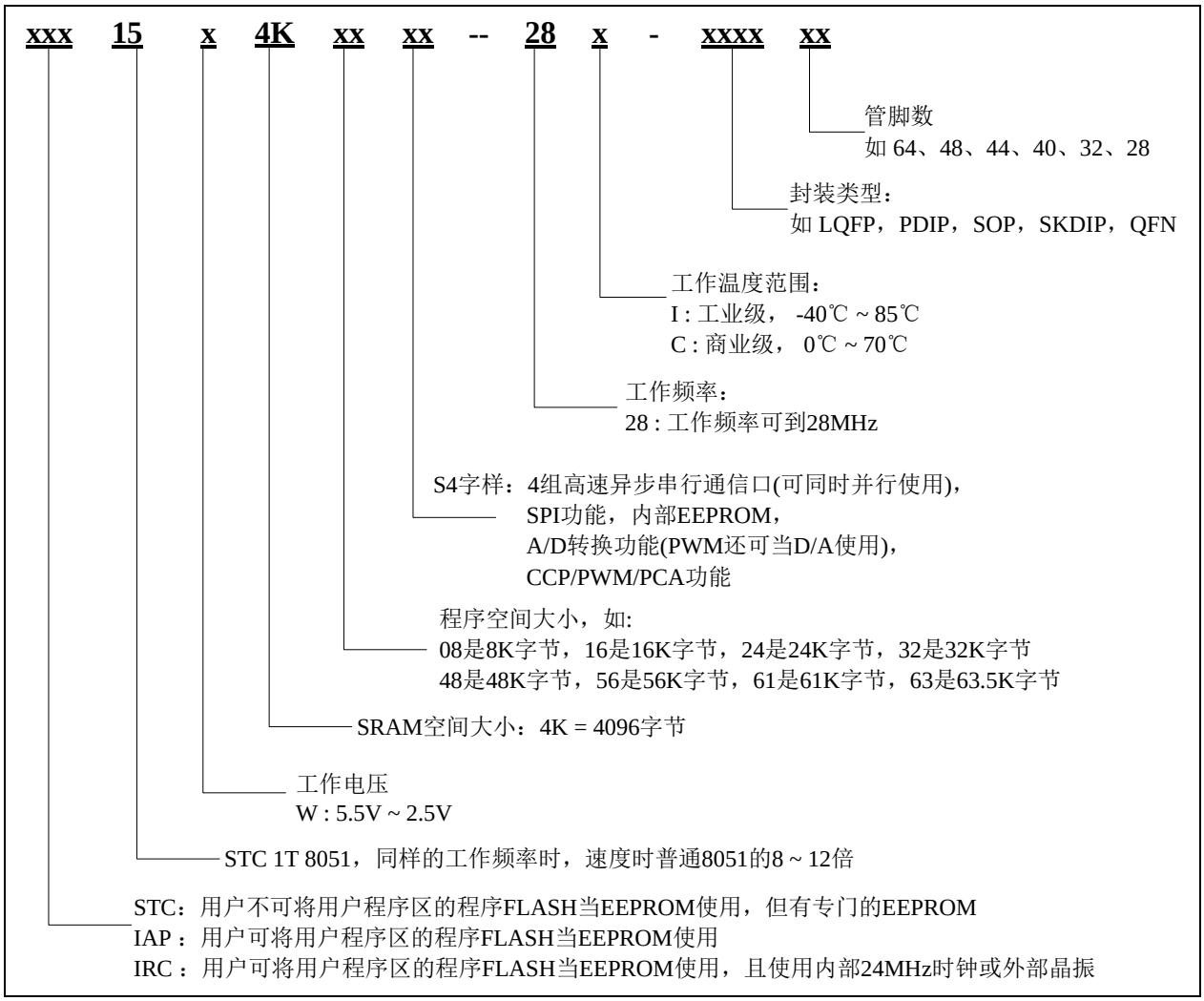
- ✓ 5 个 16 位可重装载普通定时器/计数器，这 5 个普通定时器/计数器分别是定时器/计数器 0、定时器/计数器 1、定时器/计数器 2、定时器/计数器 3 和定时器/计数器 4；
- ✓ 8 路 10 位 PWM(可再实现 8 个 D/A 转换器或 2 个定时器或 2 个外部中断)；
- ✓ 掉电唤醒专用定时器；
- ✓ 5 个外部中断 INT0/INT1/INT2/INT3/INT4；
- ✓ 4 组高速异步串行通信端口(可同时使用)；
- ✓ 1 组高速同步串行通信端口 SPI；
- ✓ 8 路高速 10 位 A/D 转换器；
- ✓ 1 个比较器；
- ✓ 2 个数据指针 DPTR；
- ✓ 外部数据总线等功能。

3.7.4 STC15W4K32S4 系列单片机封装价格一览表

型号	工作频率(MHz)	工作温度(I --工业级)	所有封装价格(RMB ¥) LQFP64S/LQFP64L/QFN64/LQFP48/QFN48/LQFP44/PDIP40/LQFP32/SOP28/SKDIP28									
			SOP28 (26 个 I/O 口)	SKDIP28 (26 个 I/O 口)	LQFP32 (30 个 I/O 口)	PDIP40 (38 个 I/O 口)	LQFP44 (42 个 I/O 口)	LQFP48 (46 个 I/O 口)	QFN48 (46 个 I/O 口)	LQFP64S (62 个 I/O 口)	LQFP64L (62 个 I/O 口)	QFN64 (62 个 I/O 口)
STC15W4K16S4	28	-40℃ ~ +85℃	¥6.20	¥6.40	¥6.00	¥6.70	¥6.00	¥6.00	¥6.00	¥6.50	-	¥6.50
STC15W4K32S4	28	-40℃ ~ +85℃	¥6.20	¥6.40	¥6.00	¥6.70	¥6.00	¥6.00	¥6.00	¥6.50	-	¥6.50
STC15W4K40S4	28	-40℃ ~ +85℃	¥5.40	¥6.40	¥6.00	¥6.70	¥6.00	¥6.00	¥6.00	¥6.50	-	¥6.50
STC15W4K48S4	28	-40℃ ~ +85℃	¥5.40	¥6.40	¥6.00	¥6.70	¥6.00	¥6.00	¥6.00	¥6.50	-	¥6.50
STC15W4K56S4	28	-40℃ ~ +85℃	¥5.40	¥6.40	¥6.00	¥6.70	¥6.00	¥6.00	¥6.00	¥6.50	-	¥6.50
IAP15W4K58S4	28	-40℃ ~ +85℃	¥5.40	¥6.40	¥6.00	¥6.70	¥6.00	¥6.00	¥6.00	¥6.50	-	¥6.50
IAP15W4K61S4	28	-40℃ ~ +85℃	¥5.40	¥6.40	¥6.00	¥6.70	¥6.00	¥6.00	¥6.00	¥6.50	-	¥6.50
IRC15W4K63S4	28	-40℃ ~ +85℃	¥5.40	¥6.40	¥6.00	¥6.70	¥6.00	¥6.00	¥6.00	¥6.50	-	¥6.50

我们直销，所以低价，以上单价为 10K 起订，量小每片需加 0.1 元，以上价格运费由客户承担，零售 10 片起，如对价格不满，可来电要求降价。

3.7.5 STC15W4K32S4 系列单片机命名规则



命名举例:

1) STC15W4K32S4-28I-SOP28 表示：

用户不可以将用户程序区的程序 FLASH 当 EEPROM 使用，但有专门的 EEPROM。该单片机为 IT 8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 5.5V ~ 2.5V，SRAM 空间大小为 4K（4096）字节，程序空间大小为 32K，有四组高速异步串行通信端口 UART 及 SPI、内部 EEPROM、A/D 转换、CCP/PCA/PWM 功能，工作频率可到 28MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 SOP 贴片封装，管脚数为 28。

2) STC15W4K40S4-28I-SKDIP28 表示：

用户不可以将用户程序区的程序 FLASH 当 EEPROM 使用，但有专门的 EEPROM。该单片机为 1T 8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 5.5V ~ 2.5V，SRAM 空间大小为 4K（4096）字节，程序空间大小为 40K，有四组高速异步串行通信端口 UART 及 SPI、内部 EEPROM、A/D 转换、CCP/PCA/PWM 功能，工作频率可到 28MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 SKDIP 封装，管脚数为 28。

3) STC15W4K48S4-28I-LQFP32 表示：

用户不可以将用户程序区的程序 FLASH 当 EEPROM 使用，但有专门的 EEPROM。该单片机为 IT

8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 5.5V ~ 2.5V，SRAM 空间大小为 4K（4096）字节，程序空间大小为 48K，有四组高速异步串行通信端口 UART 及 SPI、内部 EEPROM、A/D 转换、CCP/PCA/PWM 功能，工作频率可到 28MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 LQFP 贴片封装，管脚数为 32。

4) IAP15W4K61S4-28I-PDIP40 表示：

用户可以将用户程序区的程序 FLASH 当 EEPROM 使用。该单片机为 1T 8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 5.5V ~ 2.5V，SRAM 空间大小为 4K（4096）字节，程序空间大小为 61K，有四组高速异步串行通信端口 UART 及 SPI、内部 EEPROM、A/D 转换、CCP/PCA/PWM 功能，工作频率可到 28MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 PDIP 贴片封装，管脚数为 40。

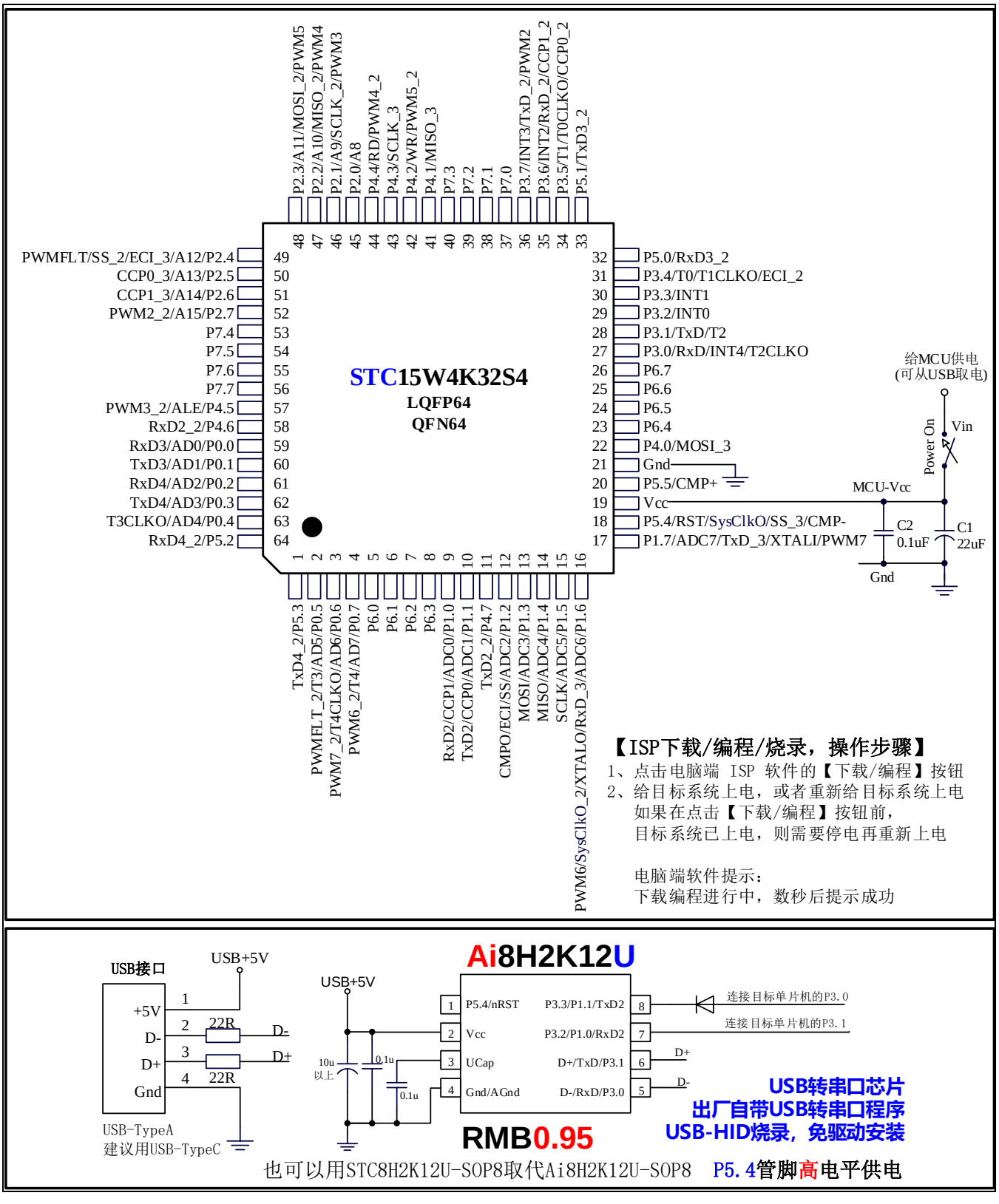
5) IRC15W4K63S4-28I-LQFP44 表示：

用户可以将用户程序区的程序 FLASH 当 EEPROM 使用，且优先使用外部晶振，当外部没有晶振时自动切换到内部 24MHz 时钟。该单片机为 IT8051 单片机，同样工作频率时，速度是普通 8051 的 8 ~ 12 倍，其工作电压为 5.5V ~ 2.5V，SRAM 空间大小为 4K（4096）字节，程序空间大小为 63K，有四组高速异步串行通信端口 UART 及 SPI、内部 EEPROM、A/D 转换、CCP/PCA/PWM 功能，工作频率可到 28MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为 LQFP 贴片封装，管脚数为 44。

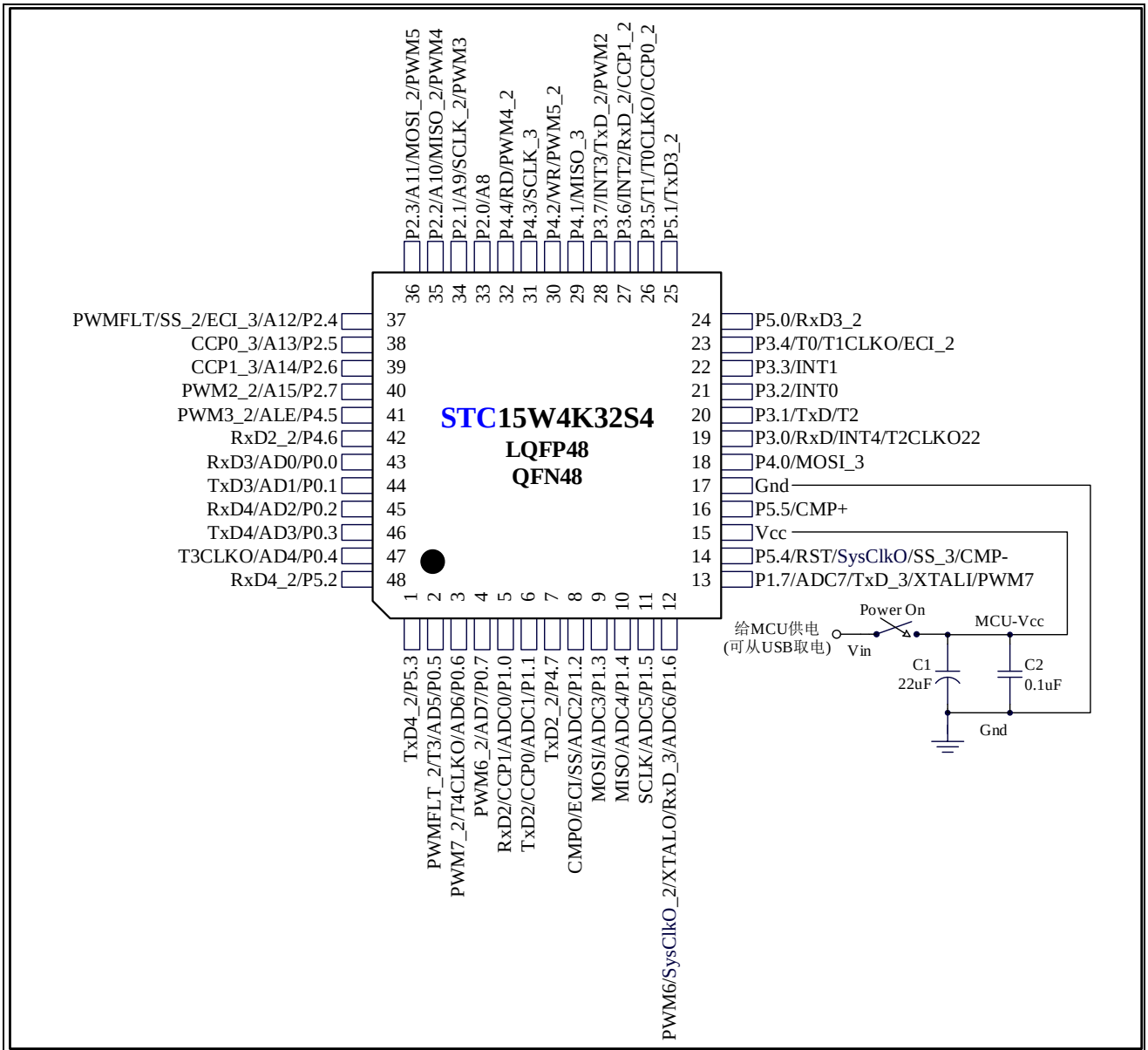
如何识别芯片版本号：如需知道芯片版本号，请查阅芯片表面印刷字中最下面一行的最后一个字母（如 B），该字母代表芯片版本号（如 B 版）

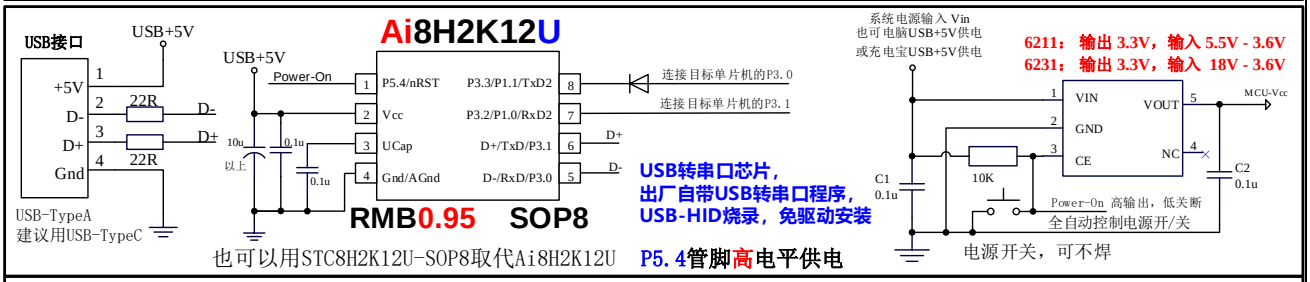
3.7.6 管脚图

3.7.6.1 管脚图及下载线路图，最小系统（LQFP64/QFN64）



3.7.6.2 管脚图及下载线路图，最小系统（LQFP48/QFN48）





【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。

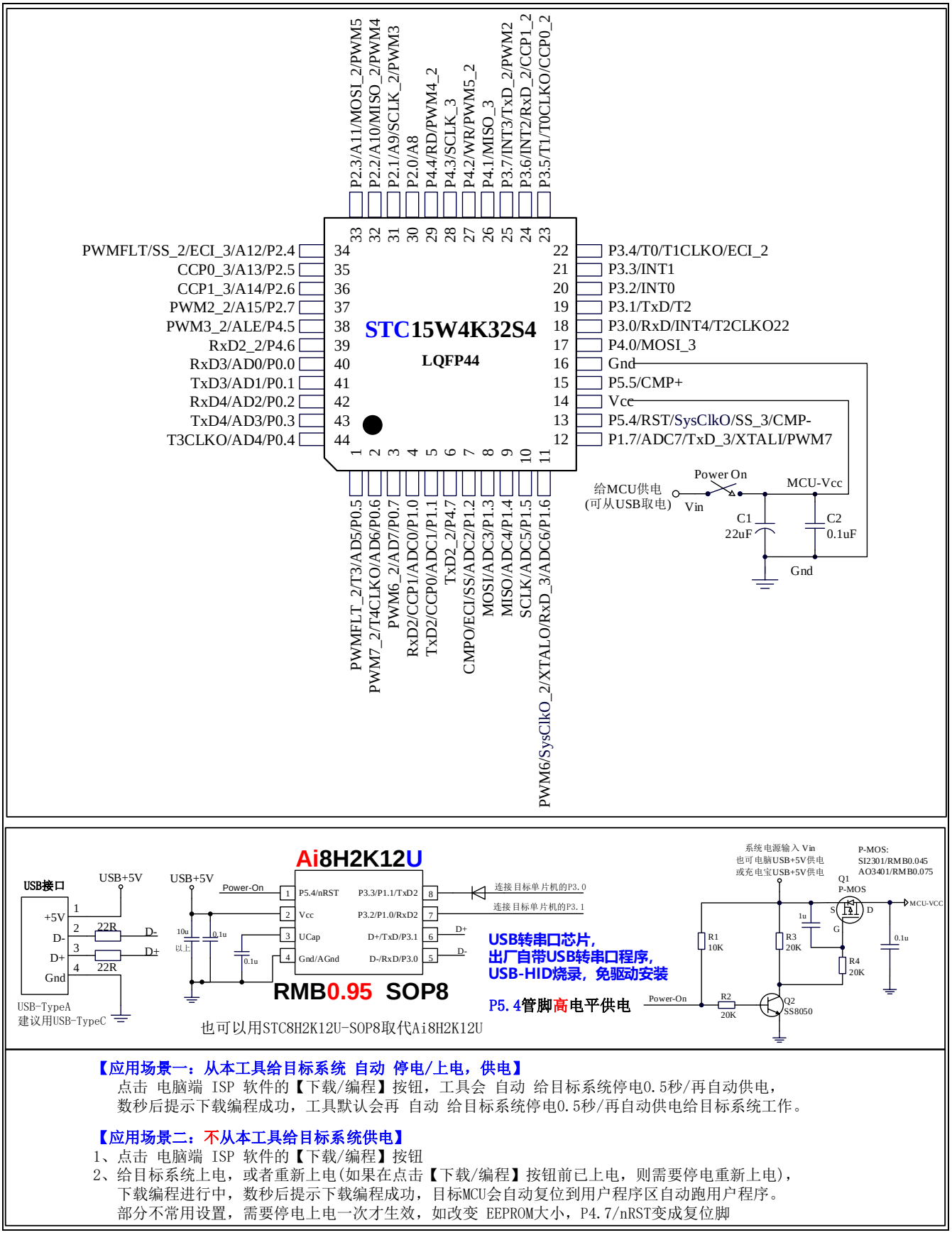
【应用场景二：不从本工具给目标系统供电】

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，P5.4/RST变成复位脚

3.7.6.3 管脚图及下载线路图，最小系统（LQFP44）



也可以用STC8H2K12U-SOP8取代Ai8H2K12U

【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。

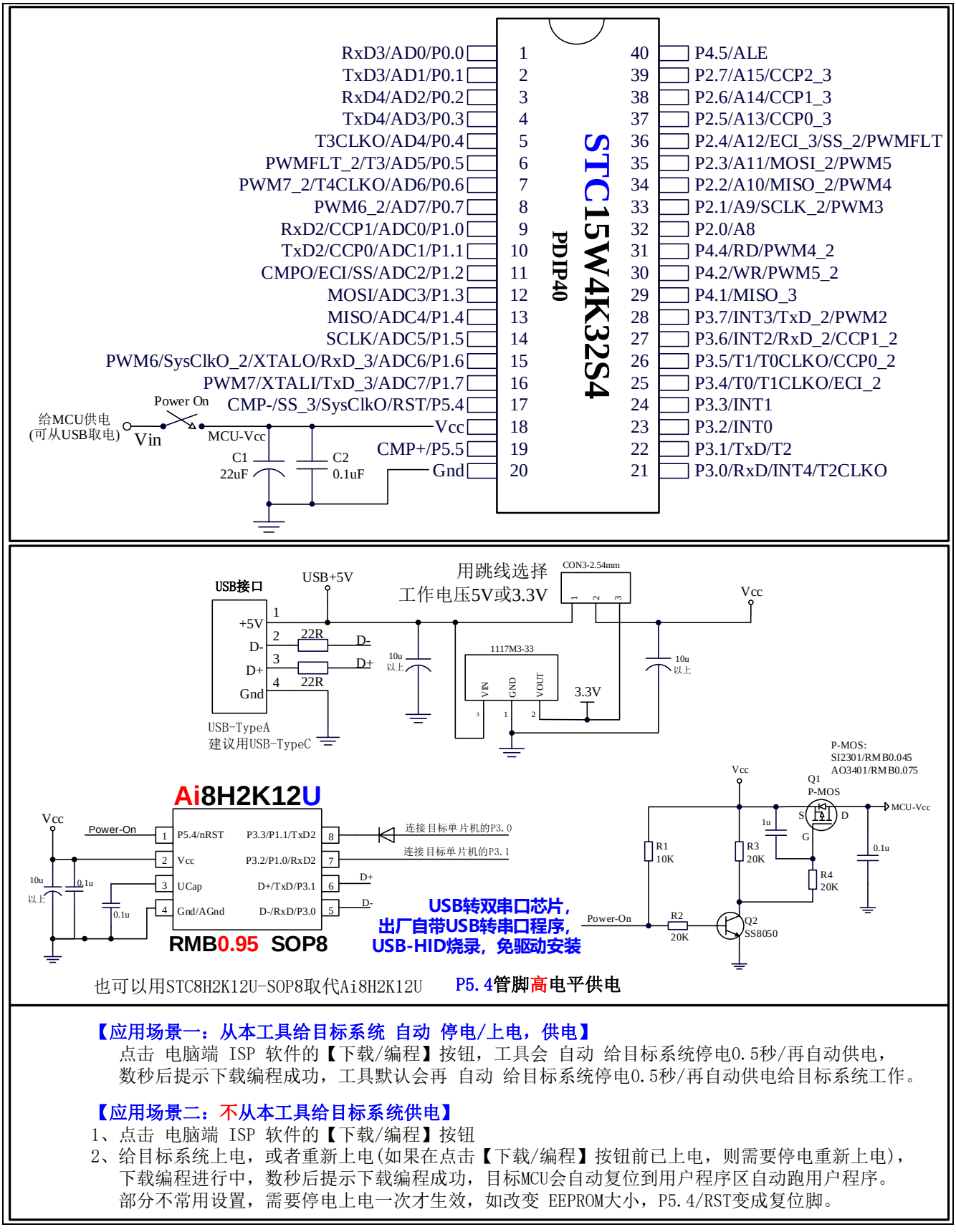
【应用场景二：不从本工具给目标系统供电】

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

3.7.6.4 管脚图及下载线路图，最小系统（PDIP40）



【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。

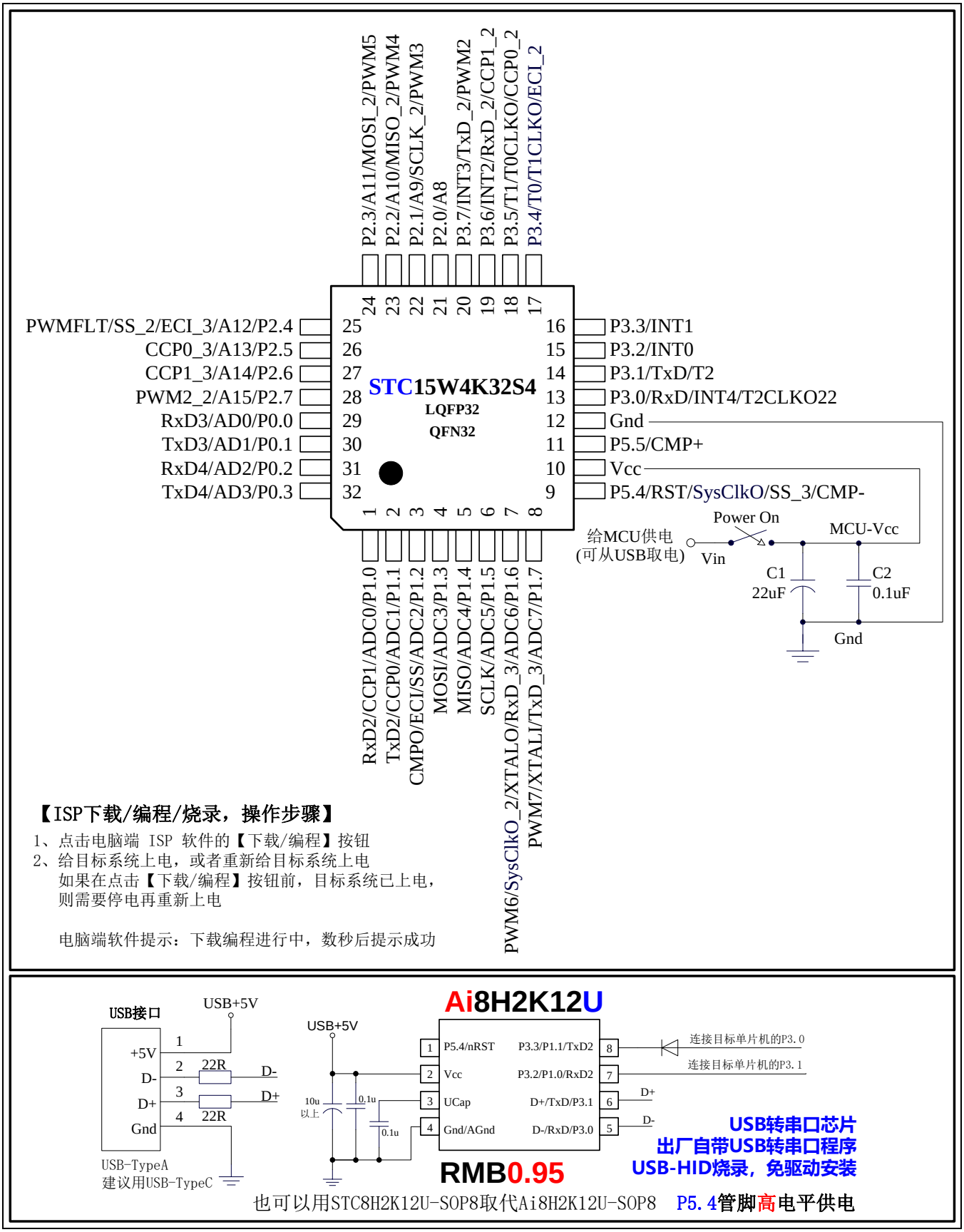
【应用场景二：不从本工具给目标系统供电】

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，P5.4/RST变成复位脚。

3.7.6.5 管脚图及下载线路图，最小系统（LQFP32/QFN32）



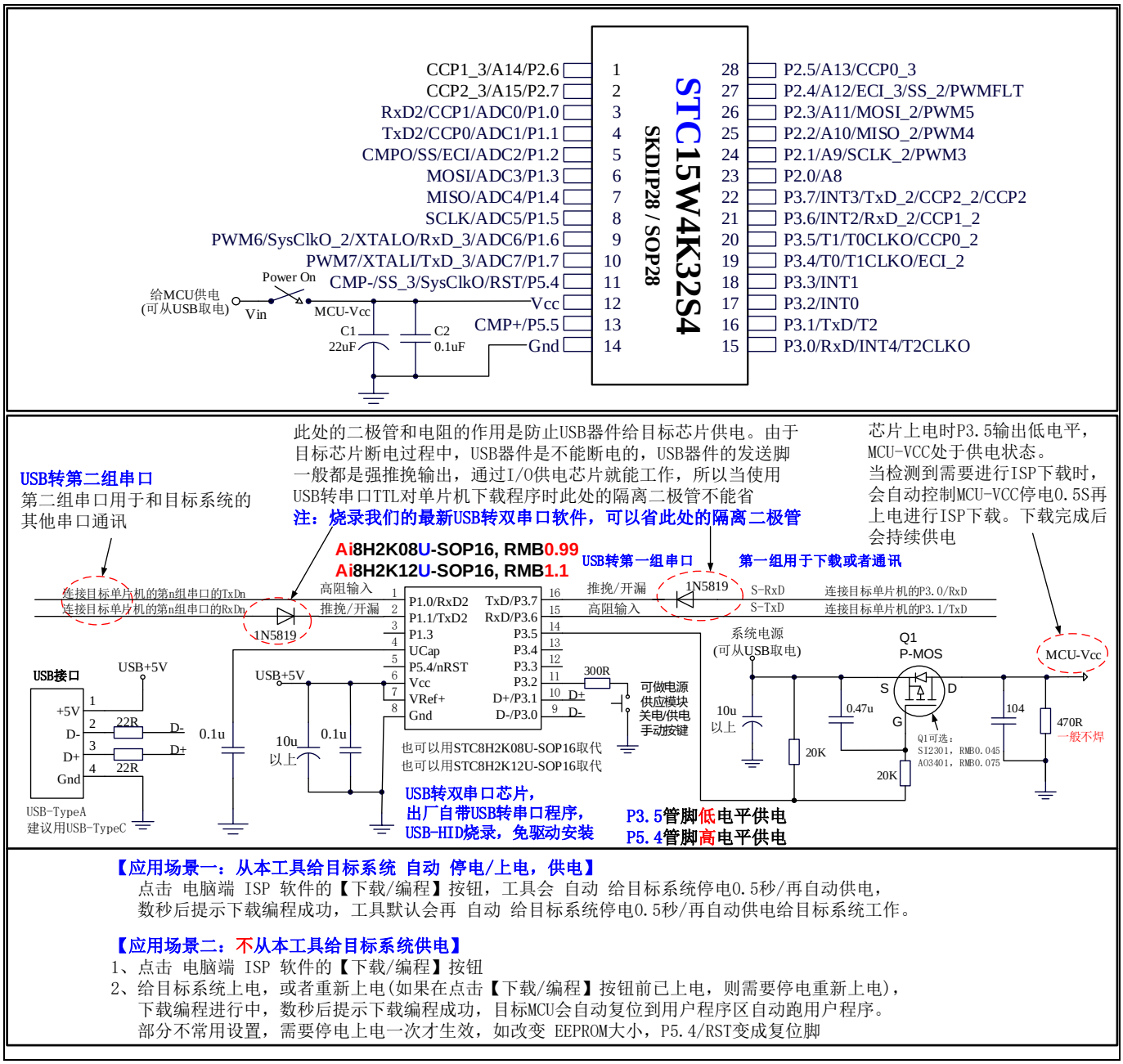
【ISP下载/编程/烧录，操作步骤】

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

也可以用STC8H2K12U-SOP8取代Ai8H2K12U-SOP8 P5.4管脚高电平供电

3.7.6.6 管脚图及下载线路图，最小系统（SKDIP28/SOP28）



【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，P5.4/RST变成复位脚

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000 0000
P_SW2	BAH	Peripheral function switch			PWM67_S	PWM2345_S		S4_S	S3_S	S2_S	xxxx x000
CLK_DIV (PCON2)	97H	时钟分频寄存器	SysCKO_S1	SysCKO_S0	ADRJ	Tx_Rx	SysClkO_2	CLKS2	CLKS1	CLKS0	0000 0000
INT_CLKO (AUXR2)	8FH	外并时钟输出并中断允许	-	EX4	EX3	EX2	SysCKO_S2	T2CLKO	T1CLKO	T0CLKO	x000 000

串口 1/S1 可在 3 个地方切换，由 S1_S0 及 S1_S1 控制位来选择		
S1_S1	S1_S0	串口 1/S1 可在 P1/P3 之间来回切换
0	0	串口 1/S1 在[P3.0/RxD，P3.1/TxD]
0	1	串口 1/S1 在[P3.6/RxD_2，P3.7/TxD_2]
1	0	串口 1/S1 在[P1.6/RxD_3/XTAL2，P1.7/TxD_3/XTAL1] 串口 1 在 P1 口时要使用内部时钟
1	1	无效

串口 1 建议放在[P3.6/RxD_2，P3.7/TxD_2]或[P1.6/RxD_3/XTAL2，P1.7/TxD_3/XTAL1]上。

建议用户在程序中将[S1_S1，S1_S0]的值设置为[0,1]或[1,0]，进而将串口 1 放在[P3.6/RxD_2.P3.7/TxD_2]或[P1.6/RxD_3/XTAL2，P1.7/TxD_3/XTAL1]上

CCP 可在 3 个地方切换，由 CCP_S1 / CCP_S0 两个控制位来选择		
CCP_S1	CCP_S0	CCP 可在 P1/P2/P3 之间来回切换
0	0	CCP 在[P1.2/ECI，P1.1/CCP0，P1.0/CCP1]
0	1	CCP 在[P3.4/ECI_2，P3.5/CCP0_2，P3.6/CCP1_2]
1	0	CCP 在[P2.4/ECI_3，P2.5/CCP0_3，P2.6/CCP1_3]
1	1	无效

PWM2/PWM3/PWM4/PWM5/PWMFLT 可在 2 个地方切换，由 PWM2345_S 控制位来选择	
PWM2345_S	切换 PWM2/PWM3/PWM4/PWM5/PWMFLT 管脚
0	PWM2/PWM3/PWM4/PWM5/PWMFLT 在[P3.7/PWM2，P2.1/PWM3，P2.2/PWM4，P2.3/PWM5，P2.4/PWMFLT]
1	PWM2/PWM3/PWM4/PWM5/PWMFLT 在[P2.7/PWM2_2，P4.5/PWM3_2，P4.4/PWM4_2，P4.2/PWM5_2，P0.5/PWMFLT_2]

PWM6/PWM7 可在 2 个地方切换，由 PWM67_S 控制位来选择	
PWM67_S	切换 PWM6/PWM7 管脚
0	PWM6/PWM7 在[P1.6/PWM6，P1.7/PWM7]
1	PWM6/PWM7 在[P0.7/PWM6_2，P0.6/PWM7_2]

与 STC15W4K32S4 系列单片机的 6 路增强型 PWM 相关的端口上电后默认为高阻输入，上电前用户须在程序中将这些端口设置为其他模式（如准双向口或强推挽模式）：注意这些端口进入掉电模式时不能为高阻输入，否则需外部加上拉电阻。

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000 0000
P_SW2	BAH	Peripheral function switch			PWM67_S	PWM2345_S		S4_S	S3_S	S2_S	xxxx x000
CLK_DIV (PCON2)	97H	时钟分频寄存器	SysCKO_S1	SysCKO_S0	ADRJ	Tx_Rx	SysClkO_2	CLKS2	CLKS1	CLKS0	0000 0000
INT_CLKO (AUXR2)	8FH	外部中断允许并时钟输出	-	EX4	EX3	EX2	SysCKO_S2	T2CLKO	T1CLKO	T0CLKO	x000 0000

SPI 可在 3 个地方切换，由 SPI_S1 / SPI_S0 两个控制位来选择		
SPI_S1	SPI_S0	SPI 可在 P1/P2/P4 之间来回切换
0	0	SPI 在[P1.2/SS，P1.3/MOSI，P1.4/MISO，P1.5/SCLK]
0	1	SPI 在[P2.4/SS_2，P2.3/MOSI_2，P2.2/MISO_2，P2.1/SCLK_2]
1	0	SPI 在[P5.4/SS_3，P4.0/MOSI_3，P4.1/MISO_3，P4.3/SCLK_3]
1	1	无效

DPS：DPTR registers select bit. DPTR 寄存器选择位

0：DPTR0 is selected DPTR0 被选择

1：DPTRl is selected DPTR1 被选择

串口 2/S2 可在 2 个地方切换，由 S2_S 控制位来选择	
S2_S	S2 可在 P1/P4 之间来回切换
0	串口 2/S2 在[P1.0/RxD2，P1.1/TxD2]
1	串口 2/S2 在[P4.6/RxD2_2，P4.7/TxD2_2]

串口 3/S3 可在 2 个地方切换，由 S3_S 控制位来选择	
S3_S	S3 可在 P0/P5 之间来回切换
0	串口 3/S3 在[P0.0/RxD3，P0.1/TxD3]
1	串口 3/S3 在[P5.0/RxD3_2，P5.1/TxD3_2]

串口 4/S4 可在 2 个地方切换，由 S4_S 控制位来选择	
S4_S	S4 可在 P0/P5 之间来回切换
0	串口 4/S4 在[P0.2/RxD4，P0.3/TxD4]
1	串口 4/S4 在[P5.2/RxD4_2，P5.3/TxD4_2]

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
CLK_DIV (PCON2)	97H	时钟分频寄存器	SysCKO_S1	SysCKO_S0	ADRJ	Tx_Rx	SysClkO_2	CLKS2	CLKS1	CLKS0	0000 0000
INT_CLKO (AUXR2)	8FH	外部中断允许并时钟输出	-	EX4	EX3	EX2	SysCKO_S2	T2CLKO	T1CLKO	T0CLKO	x000 0000

SysCKO_S2	SysCKO_S1	SysCKO_S0	系统时钟时钟对外分频输出控制位 (系统时钟是指对主时钟进行分频后供给 CPU、串行口、SPI、定时器、CCP/PWM/PCA、A/D 转换的实际工作时钟)
0	0	0	系统时钟不对外输出时钟
0	0	1	系统时钟对外输出时钟，但时钟频率不被分频，输出时钟频率 = SysClk/ 1
0	1	0	系统时钟对外输出时钟，但时钟频率被 2 分频， 输出时钟频率 = SysClk / 2
0	1	1	系统时钟对外输出时钟，但时钟频率被 4 分频， 输出时钟频率 = SysClk / 4
1	0	0	系统时钟对外输出时钟，但时钟频率被 16 分频， 输出时钟频率 = SysClk / 16

主时钟可以是内部 R/C 时钟，也可以是外部输入的时钟或外部晶体振荡产生的时钟；SysCk 是指系统时钟频率。

STC15 系列中除 STC15W4K32S4 系列、STC15W408AS 系列、STC15W1K08PWM 系列及 STC15W1K20S-LQFP64 单片机是将系统时钟对外分频输出外，其他系列单片机均是将主时钟对外分频输出。

若用户要对外输出 13.56MHz 时钟，则建议选择主时钟输出 27.12MHz（27.12÷2=13.56）

STC15W4K32S4 系列单片机通过 CLK_DIV.3/SysClk_2 位来选择是在 SysClkO/P5.4 口对外输出时钟，还是在 SysClkO_2/P1.6 口对外输出时钟。

SysClkO_2：系统时钟对外输出位置的选择位

0：在 SysClkO/P5.4 日对外输出时钟；

1：在 SysClkO_2/XTAL2/P1.6 口对外输出时钟。

系统时钟是指对主时钟进行分频后供给 CPU、串行口、SPI、定时器、CCP/PWMIPCA、A/D 转换的实际工作时钟;主时钟可以是内部 R/C 时钟，也可以是外部输入的时钟或外部晶体振荡产生的时钟。

ADRJ：ADC 转换结果调整

0：ADC_RES[7:0]存放高 8 位 ADC 结果，ADC_RESL[1:0]存放低 2 位 ADC 结果

1：ADC_RES[1:0]存放高 2 位 ADC 结果，ADC_RESL[7:0]存放低 8 位 ADC 结果

Tx_Rx：串口 1 的中继广播方式设置

0：串口 1 为正常工作方式

1：串口 1 为中继广播方式，即将 RxD 端口输入的电平状态实时输出在 TxD 外部管脚上，TxD 外部管脚可以对 RxD 管脚的输入信号进行实时整形放大输出，TxD 管脚的对外输出实时反

映 RxD 端口输入的电平状态。

串口 1 的 RxD 管脚和 TxD 管脚可以在 3 组不同管脚之间进行切换：[RxD/P3.0，TxD/P3.11]；
[RxD_2/P3.6，TxD_2/P3.7]；
[RxD_3/P1.6，TxD_3/P1.7].

CLKS2	CLKS1	CLKS0	系统时钟选择控制位 (系统时钟是指对主时钟进行分频后供给 CPU、串行口、SPI、定时器、CCP/PWM/PCA、A/D 转换的实际工作时钟)
0	0	0	主时钟频率/1，不分频
0	0	1	主时钟频率/2
0	1	0	主时钟频率/4
0	1	1	主时钟频率/8
1	0	0	主时钟频率/16
1	0	1	主时钟频率/32
1	1	0	主时钟频率/64
1	1	1	主时钟频率/128

3.7.7 管脚说明

管脚	管脚编号							说明	
	LQFP64	LQFP48	LQFP44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28		
P0.0	59	43	40	1	1	29	-	P0.0	标准 I/O 口 PORT0[0]
								AD0	地址/数据总线
								RxD3	串口 3 数据接收端
P0.1	60	44	41	2	2	30	-	P0.1	标准 I/O 口 PORT0[1]
								AD1	地址/数据总线
								TxD3	串口 3 数据发送端
P0.2	61	45	42	3	3	31	-	P0.2	标准 I/O 口 PORT0[2]
								AD2	地址/数据总线
								RxD4	串口 4 数据接收端
P0.3	62	46	43	4	4	32	-	P0.3	标准 I/O 口 PORT0[3]
								AD3	地址/数据总线
								TxD4	串口 4 数据发送端
P0.4	63	47	44	5	-	-	-	P0.4	标准 I/O 口 PORT0[4]
								AD4	地址/数据总线
								T3CLKO	定时器/计数器 3 的时钟输出 可通过设置 T4T3M[0]位/T3CLKO 将该管脚配置为 T3CLKO
P0.5	2	2	1	6	-	-	-	P0.5	标准 I/O 口 PORT0[5]
								AD5	地址/数据总线
								T3	定时器/计数器 3 的外部输入
								PWMFLT_2	PWM 异常停机控制管脚。
P0.6	3	3	2	7	-	-	-	P0.6	标准 I/O 口 PORT0[6]
								AD6	地址/数据总线
								T4CLKO	定时器/计数器 4 的时钟输出 可通过设置 T4T3M[4]位/T4CLKO 将该管脚配置为 T4CLKO
								PWM7_2	脉宽调制输出通道-7 该端口上电后默认为高阻输入，上电前用户须在程序中将该端口设置为其他模式（如准双向口或强推挽模式）；该端口进入掉电模式时不能为高阻输入，否则需外加上拉电阻。
P0.7	4	4	3	8	-	-	-	P0.7	标准 I/O 口 PORT0[7]
								AD7	地址/数据总线
								T4	定时器/计数器 4 的外部输入
								PWM6_2	脉宽调制输出通道-6 该端口上电后默认为高阻输入，上电前用户须在程序中将该端口设置为其他模式（如准双向口或强推挽模式）；该端口进入掉电模式时不能为高阻输入，否则需外部加上拉电阻。
P1.0	9	5	4	9	5	1	3	P1.0	标准 I/O 口 PORT1[0]
								ADC0	ADC 输入通道-0
								CCP1	外部信号捕获（频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-1
								RxD2	串口 2 数据接收端
P1.1	10	6	5	10	6	2	4	P1.1	标准 I/O 口 PORT1[1]
								ADC1	ADC 输入通道-1
								CCP0	外部信号捕获（频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-0
								TxD2	串口 2 数据发送端

管脚	管脚编号							说明	
	LQFP64	LQFP48	LQFP44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28		
P1.2	12	8	7	11	7	3	5	P1.2	标准 I/O 口 PORT1[2]
								ADC2	ADC 输入通道-2
								SS	SPI 同步串行接口的从机选择信号
								ECI	CCP / PCA 计数器的外部脉冲输入脚
								CMPO	比较器的比较结果输出管脚
P1.3	13	9	8	12	8	4	6	P1.3	标准 I/O 口 PORT1[3]
								ADC3	ADC 输入通道-3
								MOSI	SPI 同步串行接口的主出从入 （主器件的输出和从器件的输入）
P1.4	14	10	9	13	9	5	7	P1.4	标准 I/O 口 PORT1[4]
								ADC4	ADC 输入通道-4
								MISO	SPI 同步串行接口的主入从出 （主器件的输入和从器件的输出）
P1.5	15	11	10	14	10	6	8	P1.5	标准 I/O 口 PORT1[5]
								ADC5	ADC 输入通道-5
								SCLK	SPI 同步串行接口的时钟信号
P1.6	16	12	11	15	11	7	9	P1.6	标准 I/O 口 PORT1[6]
								ADC6	ADC 输入通道-6
								RxD_3	串口 1 数据接收端
								SysClkO_2	系统时钟输出（输出的频率可为 SysClk/1，SysClk/2，SysClk/4，SysClk/16）。 系统时钟是指对主时钟进行分频后供给 CPU、定时器、串行口、SPI 的实际工作时钟；主时钟可以是内部 R/C 时钟，也可以是外部输入的时钟或外部晶体振荡产生的时钟；SysClk 是指系统时钟频率。
								XTAL2	内部时钟电路反相放大器的输出端，接外部晶振的其中一端。当直接使用外部时钟源时，此引脚可浮空，此时 XTAL2 实际将 XTAL1 输入的时钟进行输出。
								PWM6	脉宽调制输出通道-6 该端口上电后默认为高阻输入，上电前用户须在程序中将该端口设置为其他模式（如准双向口或强推挽模式）；该端口进入掉电模式时不能为高阻输入，否则需外部加上拉电阻。
P1.7	17	13	12	16	12	8	10	P1.7	标准 I/O 口 PORT1[7]
								ADC7	ADC 输入通道-7
								TxD_3	串口 1 数据发送端
								XTAL1	内部时钟电路反相放大器输入端，接外部晶振的其中一端。当直接使用外部时钟源时，此引脚是外部时钟源的输入端。
								PWM7	脉宽调制输出通道-7 该端口上电后默认为高阻输入，上电前用户须在程序中将该端口设置为其他模式（如准双向口或强推挽模式）；该端口进入掉电模式时不能为高阻输入，否则需外部加上拉电阻。
P2.0	45	33	30	32	25	21	23	P2.0	标准 I/O 口 PORT2[0]
								A8	地址总线第 8 位 — A8
								RSTOUT_LOW	上电后，输出低电平，在复位期间也是输出低电平，用户可用软件将其设置为高电平或低电平，如果要读外部状态，可将该口先置高后再读

管脚	管脚编号							说明	
	LQFP64	LQFP48	LQFP44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28		
P2.1	46	34	31	33	26	22	24	P2.1	标准 I/O 口 PORT2[1]
								A9	地址总线第 9 位 — A9
								SCLK_2	SPI 同步串行接口的时钟信号
								PWM3	脉宽调制输出通道-3 该端口上电后默认为高阻输入，上电前用户须在程序中将该端口设置为其他模式（如准双向口或强推挽模式）；该端口进入掉电模式时不能为高阻输入，否则需外部加上拉电阻。
P2.2	47	35	32	34	27	23	25	P2.2	标准 I/O 口 PORT2[2]
								A10	地址总线第 10 位 — A10
								MISO_2	SPI 同步串行接口的主入从出 （主器件的输入和从器件的输出）
								PWM4	脉宽调制输出通道-4 该端口上电后默认为高阻输入，上电前用户须在程序中将该端口设置为其他模式（如准双向口或强推挽模式）；该端口进入掉电模式时不能为高阻输入，否则需外部加上拉电阻。
P2.3	48	36	33	35	28	24	26	P2.3	标准 I/O 口 PORT2[3]
								A11	地址总线第 11 位 — A11
								MOSI_2	SPI 同步串行接口的主出从入 （主器件的输出和从器件的输入）
								PWM5	脉宽调制输出通道-5 该端口上电后默认为高阻输入，上电前用户须在程序中将该端口设置为其他模式（如准双向口或强推挽模式）；该端口进入掉电模式时不能为高阻输入，否则需外部加上拉电阻。
P2.4	49	37	34	36	29	25	27	P2.4	标准 I/O 口 PORT2[4]
								A12	地址总线第 12 位 — A12
								ECI_3	CCP / PCA 计数器的外部脉冲输入脚
								SS_2	SPI 同步串行接口的从机选择信号
								PWMFLT	PWM 异常停机控制管脚
P2.5	50	38	35	37	30	26	28	P2.5	标准 I/O 口 PORT2[5]
								A13	地址总线第 13 位 — A13
								CCP0_3	外部信号捕获（频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-0
P2.6	51	39	36	38	31	27	1	P2.6	标准 I/O 口 PORT2[6]
								A14	地址总线第 14 位 — A14
								CCP1_3	外部信号捕获（频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-1
P2.7	52	40	37	39	32	28	2	P2.7	标准 I/O 口 PORT2[7]
								A15	地址总线第 15 位 — A15
								PWM2_2	脉宽调制输出通道-2 该端口上电后默认为高阻输入，上电前用户须在程序中将该端口设置为其他模式（如准双向口或强推挽模式）；该端口进入掉电模式时不能为高阻输入，否则需外部加上拉电阻。
P3.0	27	19	18	21	17	13	15	P3.0	标准 I/O 口 PORT3[0]
								RxD	串口 1 数据接收端
								INT4	外部中断 4，只能下降沿中断，INT4 支持掉电唤醒
								T2CLKO	T2 的时钟输出 可通过设置 INT_CLKO[2]位/T2CLKO 将该管脚配置为 T2CLKO

管脚	管脚编号							说明	
	LQFP64	LQFP48	LQFP44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28		
P3.1	28	20	19	22	18	14	16	P3.1	标准 I/O 口 PORT3[1]
								TxD	串口 1 数据发送端
								T2	定时器/计数器 2 的外部输入
P3.2	29	21	20	23	19	15	17	P3.2	标准 I/O 口 PORT3[2]
								INT0	外部中断 0，既可上升沿中断也可下降沿中断. 如果 IT0（TCON.0）被置为 1，INT0 管脚仅为下降沿中断。 如果 IT0（TCON.0）被清 0，INT0 管脚既支持上升沿中断也支持下降沿中断。 INT0 支持掉电唤醒。
P3.3	30	22	21	24	20	16	18	P3.3	标准 I/O 口 PORT3[3]
								INT1	外部中断 1，既可上升沿中断也可下降沿中断. 如果 IT1（TCON.2）被置为 1，INT1 管脚仅为下降沿中断。 如果 IT1（TCON.2）被清 0，INT1 管脚既支持上升沿中断也支持下降沿中断。 INT1 支持掉电唤醒。
P3.4	31	23	22	25	21	17	19	P3.4	标准 I/O 口 PORT3[4]
								T0	定时器/计数器 0 的外部输入
								T1CLKO	定时器/计数器 1 的时钟输出 可通过设置 INT_CLKO[1]位/T1CLKO 将该管脚配置为 T1CLKO，也可对 T1 脚的外部时钟输入进行分频输出
								ECI_2	CCP/PCA 计数器的外部脉冲输入脚
P3.5	34	26	23	26	22	18	20	P3.5	标准 I/O 口 PORT3[5]
								T1	定时器/计数器 1 的外部输入
								T0CLKO	定时器/计数器 0 的时钟输出 可通过设置 INT_CLKO[0]位/T0CLKO 将该管脚配置为 T0CLKO，也可对 T0 脚的外部时钟输入进行分频输出
								CCP0_2	外部信号捕获（频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-0
P3.6	35	27	24	27	23	19	21	P3.6	标准 I/O 口 PORT3[6]
								INT2	外部中断 2，只能下降沿中断 INT2 支持掉电唤醒
								RxD_2	串口 1 数据接收端
								CCP1_2	外部信号捕获（频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-1
P3.7	36	28	25	28	24	20	22	P3.7	标准 I/O 口 PORT3[7]
								INT3	外部中断 3，只能下降沿中断 INT3 支持掉电唤醒
								TxD_2	串口 1 数据发送端
								PWM2	脉宽调制输出通道-2 该端口上电后默认为高阻输入，上电前用户须在程序中将该端口设置为其他模式（如准双向口或强推挽模式）；该端口进入掉电模式时不能为高阻输入，否则需外部加上拉电阻。
P4.0	22	18	17	-	-	-	-	P4.0	标准 I/O 口 PORT4[0]
								MISO_3	SPI 同步串行接口的主入从出 （主器件的输入和从器件的输出）
P4.1	41	29	26	29	-	-	-	P4.1	标准 I/O 口 PORT4[1]
								MOSI_3	SPI 同步串行接口的主出从入 （主器件的输出和从器件的输入）
P4.2	42	30	27	30	-	-	-	P4.2	标准 I/O 口 PORT4[2]
								WR	外部数据存储器写脉冲
								PWM5_2	脉宽调制输出通道-5。 该端口上电后默认为高阻输入，上电前用户须在程序中将该端口设置为其他模式（如准双向口或强推挽模式）；该端口进入掉电模式时不能为高阻输入，否则需外部加上拉电阻。

管脚	管脚编号							说明	
	LQFP64	LQFP48	LQFP44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28		
P4.3	43	31	28	-	-	-	-	P4.3	标准 I/O 口 PORT4[3]
								SCLK_3	SPI 同步串行接口的时钟信号
P4.4	44	32	29	31	-	-	-	P4.4	标准 I/O 口 PORT4[4]
								RD	外部数据存储器读脉冲
								PWM4_2	脉宽调制输出通道-4。 该端口上电后默认为高阻输入，上电前用户须在程序中将该端口设置为其他模式（如准双向口或强推挽模式）；该端口进入掉电模式时不能为高阻输入，否则需外部加上拉电阻。
P4.5	57	41	38	40	-	-	-	P4.5	标准 I/O 口 PORT4[5]
								ALE	地址锁存允许
								PWM3_2	脉宽调制输出通道-3 该端口上电后默认为高阻输入，上电前用户须在程序中将该端口设置为其他模式（如准双向口或强推挽模式）；该端口进入掉电模式时不能为高阻输入，否则需外部加上拉电阻。
P4.6	58	42	39	-	-	-	-	P4.6	标准 I/O 口 PORT4[6]
								RxD2_2	串口 2 数据接收端
P4.7	11	7	6	-	-	-	-	P4.7	标准 I/O 口 PORT4[7]
								TxD2_2	串口 2 数据发送端
P5.0	32	24	-	-	-	-	-	P5.0	标准 I/O 口 PORT5[0]
								RxD3_2	串口 3 数据接收端
P5.1	33	25	-	-	-	-	-	P5.1	标准 I/O 口 PORT5[1]
								TxD3_2	串口 3 数据发送端
P5.2	64	48	-	-	-	-	-	P5.2	标准 I/O 口 PORT5[2]
								RxD4_2	串口 4 数据接收端
P5.3	1	1	-	-	-	-	-	P5.3	标准 I/O 口 PORT5[3]
								TxD4_2	串口 4 数据发送端
P5.4	18	14	13	17	13	9	11	P5.4	标准 I/O 口 PORT5[4]
								RST	复位脚（高电平复位）
								SysClkO	系统时钟输出（输出的频率可为 SysClk/1，SysClk/2，SysClk/4，SysClk/16）。 系统时钟是指对主时钟进行分频后供给 CPU、定时器、串行口、SPI 的实际工作时钟；主时钟可以是内部 R/C 时钟，也可以是外部输入的时钟或外部晶体振荡产生的时钟；SysClk 是指系统时钟频率。
								SS_3	SPI 同步串行接口的从机选择信号
								CMP-	比较器负极输入端 （若该口被用作比较器负极，则该口需被设置为高阻输入）
P5.5	20	16	15	19	15	11	13	P5.5	标准 I/O 口 PORT5[5]
								CMP+	比较器正极输入端 （若该口被用作比较器正极，则该口需被设置为高阻输入）
P6.0	5							标准 I/O 口 PORT6[0]	
P6.1	6							标准 I/O 口 PORT6[1]	
P6.2	7							标准 I/O 口 PORT6[2]	
P6.3	8							标准 I/O 口 PORT6[3]	
P6.4	23							标准 I/O 口 PORT6[4]	
P6.5	24							标准 I/O 口 PORT6[5]	
P6.6	25							标准 I/O 口 PORT6[6]	
P6.7	26							标准 I/O 口 PORT6[7]	
P7.0	37							标准 I/O 口 PORT7[0]	

管脚	管脚编号							说明
	LQFP64	LQFP48	LQFP44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28	
P7.1	38							标准 I/O 口 PORT7[1]
P7.2	39							标准 I/O 口 PORT7[2]
P7.3	40							标准 I/O 口 PORT7[3]
P7.4	53							标准 I/O 口 PORT7[4]
P7.5	54							标准 I/O 口 PORT7[5]
P7.6	55							标准 I/O 口 PORT7[6]
P7.7	56							标准 I/O 口 PORT7[7]
Vcc	19	15	14	18	14	10	12	电源正极
Gnd	21	17	16	20	16	12	14	电源负极，接地

3.7.8 USB-Link1D 工具自动停电/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】

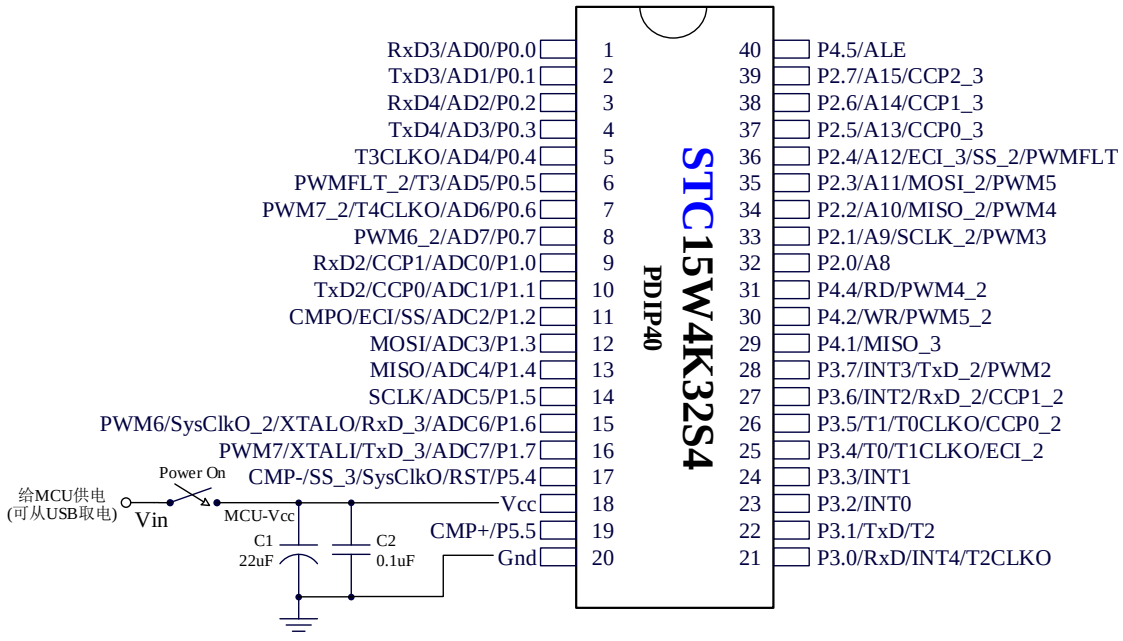
点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【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



3.7.9 【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】

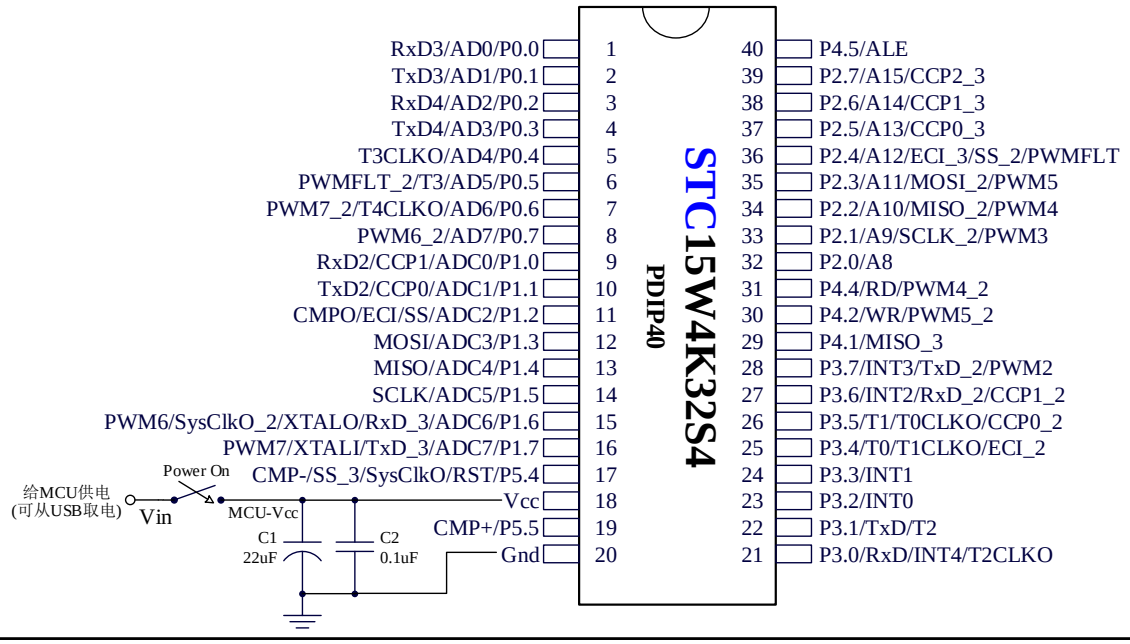
点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【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

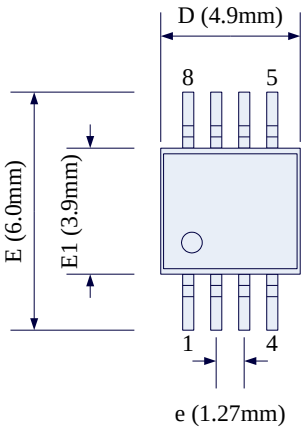
2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，下载编程进行中，数秒后提示下载编程成功，目标MCU会自动复位到用户程序区自动跑用户程序。

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

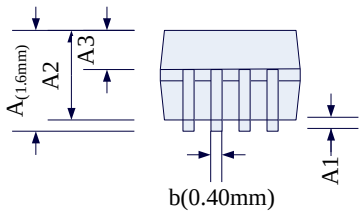


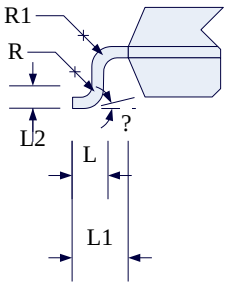
4 STC15 系列单片机封装尺寸图，USB-TypeC 封装尺寸图/原理图

4.1 SOP8 封装尺寸图

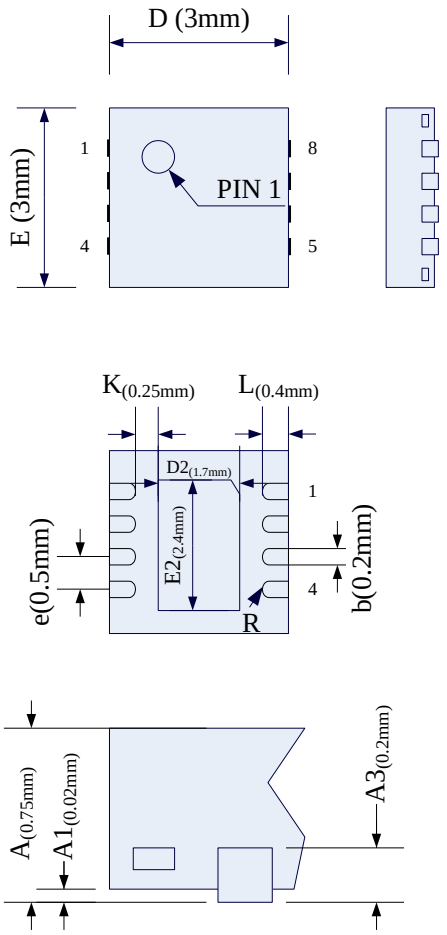


一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	1.35	1.60	1.75
A1	0.10	0.15	0.25
A2	1.25	1.45	1.65
A3	0.55	0.65	0.75
b	0.35	0.40	0.45
D	4.80	4.90	5.00
E	5.80	6.00	6.20
E1	3.80	3.90	4.00
e	1.27BSC		
L	0.45	0.60	0.80
L1	1.04REF		
L2	0.25BSC		
R1	0.07	-	-
R2	0.07	-	-





4.2 DFN8 封装尺寸图（3mm*3mm）

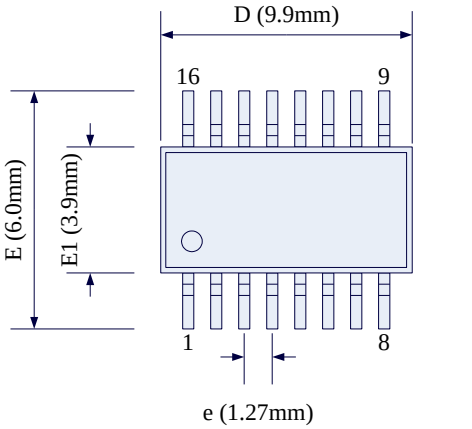


一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	0.70	0.75	0.80
A1	0	0.02	0.05
A2	0.50	0.55	0.60
A3	-	0.203REF	-
b	0.175	0.20	0.225
D	2.89	3.00	3.11
E	2.89	3.00	3.11
D2	1.65	1.70	1.75
E2	2.35	2.40	2.45
e	0.45	0.50	0.55
L	0.381	0.40	0.419
K	0.25REF		
R	0.125	-	-

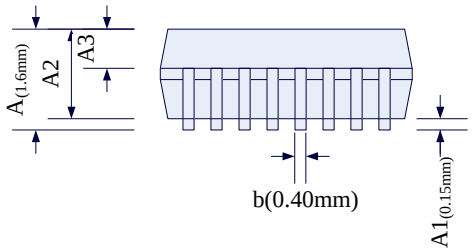
现有 DFN8 封装芯片的背面金属片（衬底），在芯片内部并未接地，在用户的 PCB 板上可以接地，也可以不接地，不会对芯片性能造成影响

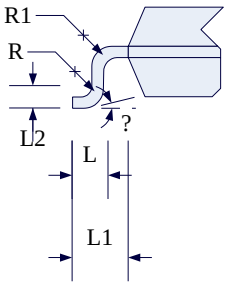
特别说明：造 PCB 的封装库时，焊盘可以向外延伸 0.1～0.2mm，但绝对不能向内延伸，必须确保 K 不能小于 0.25mm，否则会造成管脚短路

4.3 SOP16 封装尺寸图

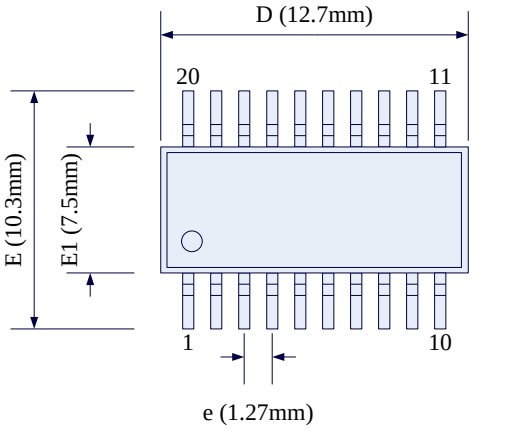


一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	1.35	1.60	1.75
A1	0.10	0.15	0.25
A2	1.25	1.45	1.65
A3	0.55	0.65	0.75
b	0.35	0.40	0.45
D	9.80	9.90	10.00
E	5.80	6.00	6.20
E1	3.80	3.90	4.00
e		1.27BSC	
L	0.45	0.60	0.80
L1		1.04REF	
L2		0.25BSC	
R1	0.07	-	-
R2	0.07	-	-

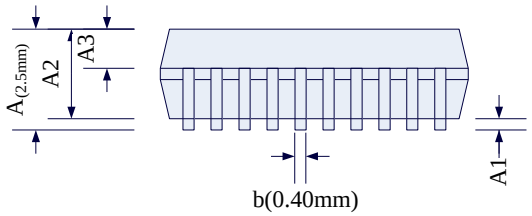


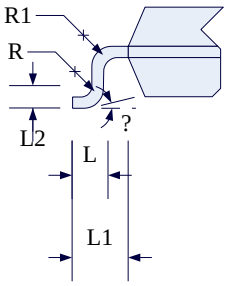


4.4 SOP20 封装尺寸图

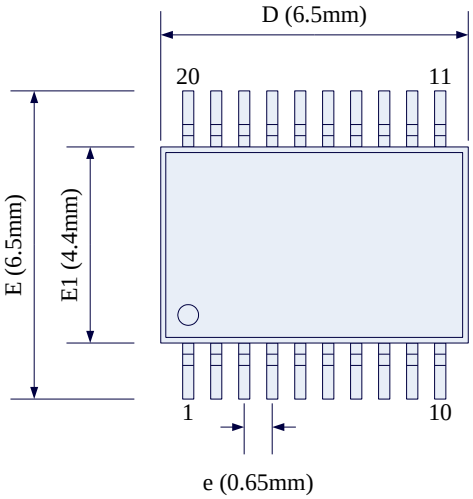


一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	2.465	2.515	2.565
A1	0.10	0.15	0.20
A2	2.10	2.30	2.50
A3	0.55	0.65	0.75
b	0.356	0.406	0.456
D	12.50	12.70	12.90
E	10.20	10.30	10.40
E1	7.45	7.50	7.55
e		1.27BSC	
L	0.80	0.86	0.90
L1		1.40REF	
L2		0.27BSC	
R1	-	0.30	-
R2	-	0.20	-

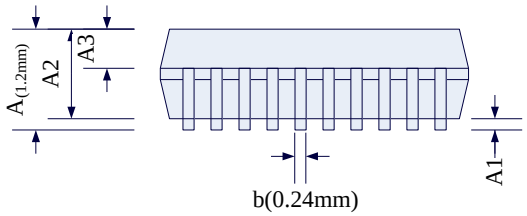


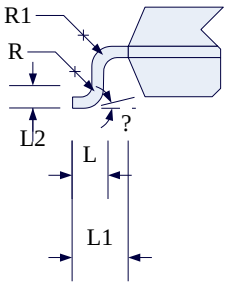


4.5 TSSOP20 封装尺寸图

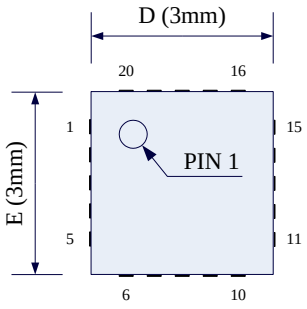


一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	-	-	1.20
A1	0.05	-	0.15
A2	0.90	1.00	1.05
A3	0.34	0.44	0.54
b	0.20	0.24	0.28
D	6.40	6.50	6.60
E	6.20	6.50	6.60
E1	4.30	4.40	4.50
e		0.65BSC	
L	0.45	0.60	0.75
L1		1.00REF	
L2		0.25BSC	
R1	0.09	-	-
R2	0.09	-	-

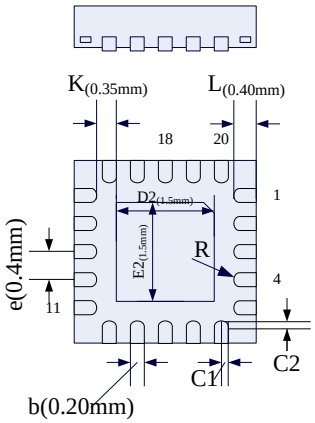


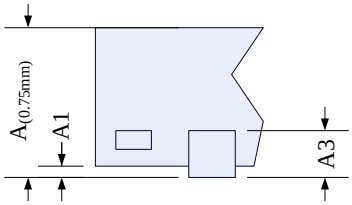


4.6 QFN20 封装尺寸图（3mm*3mm）



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	-

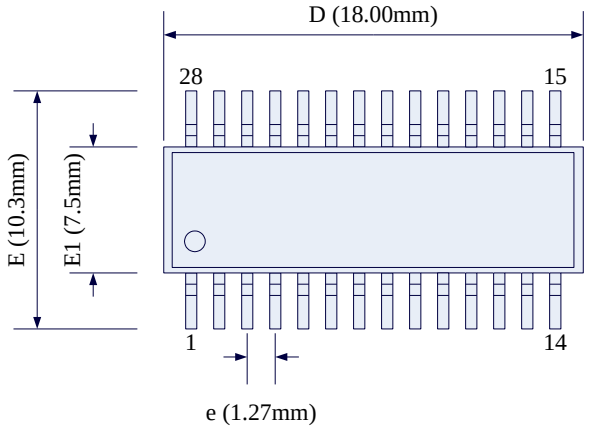




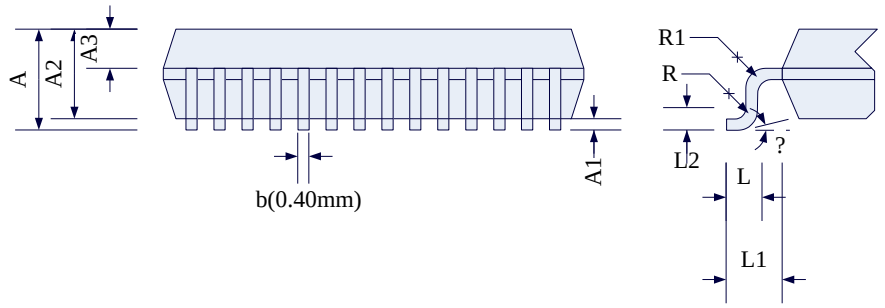
现有 QFN20 封装芯片的背面金属片（衬底），在芯片内部并未接地，在用户的 PCB 板上可以接地，也可以不接地，不会对芯片性能造成影响

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

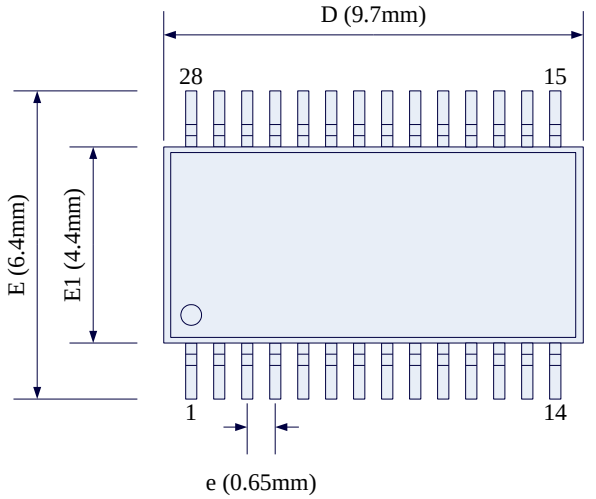
4.7 SOP28 封装尺寸图



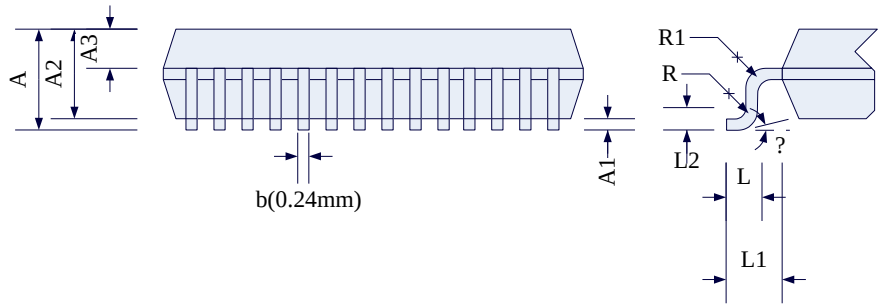
一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	-	-	2.65
A1	0.03	0.08	0.12
A2	2.15	2.20	2.25
A3	0.97	1.02	1.07
b	0.39	-	0.47
D	17.90	18.00	18.10
E	10.10	10.30	10.50
E1	7.40	7.50	7.60
e		1.27BSC	
L	0.70	-	1.00
L1		1.40REF	
L2		0.27BSC	
R1	-	0.30	-
R2	-	0.20	-



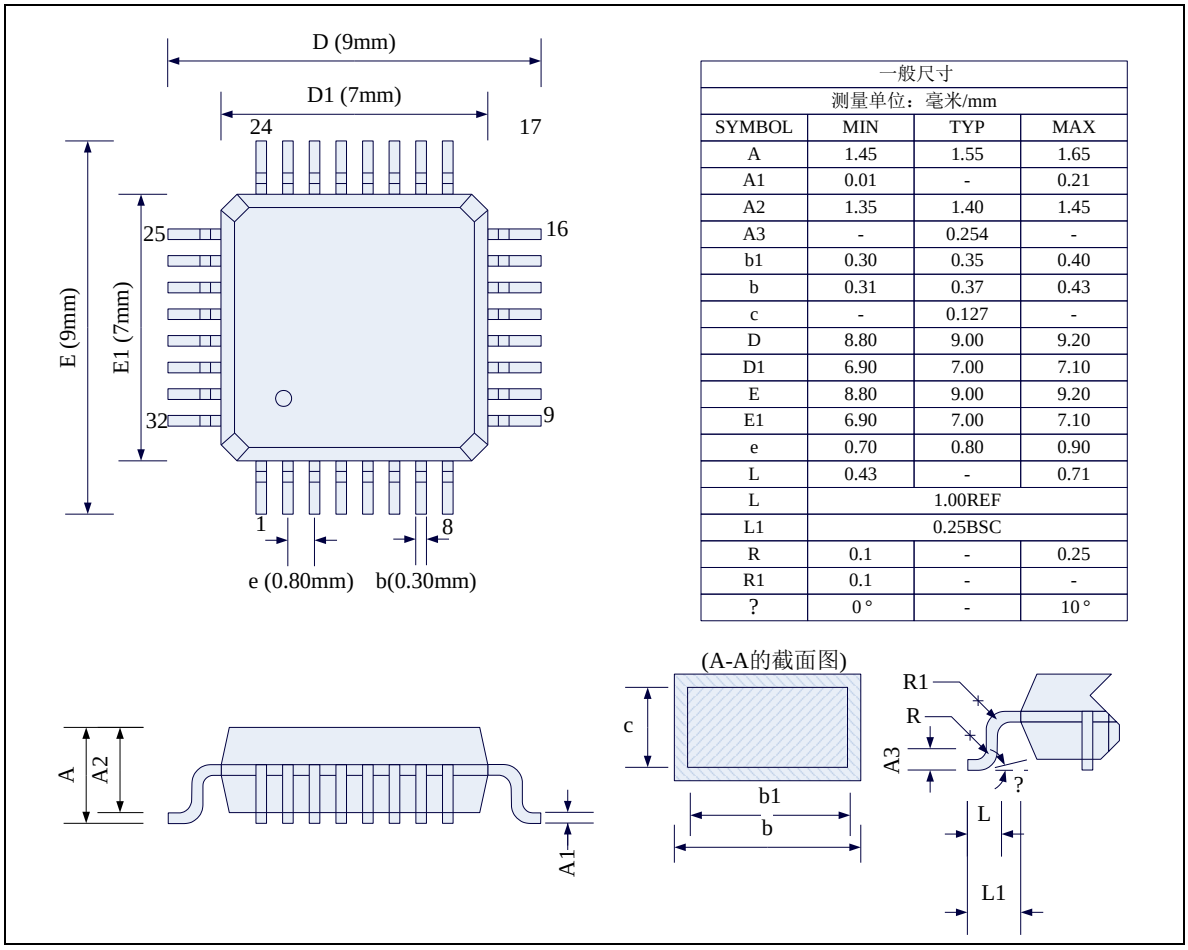
4.8 TSSOP28 封装尺寸图



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	-	-

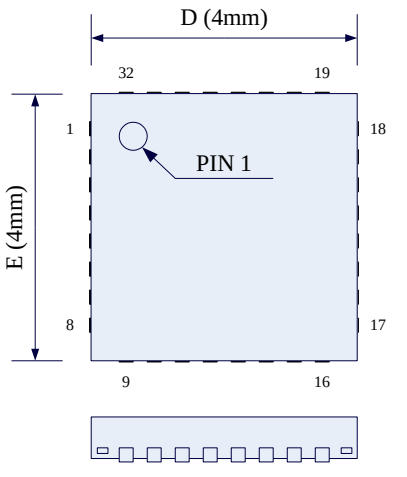


4.9 LQFP32 封装尺寸图（9mm*9mm）

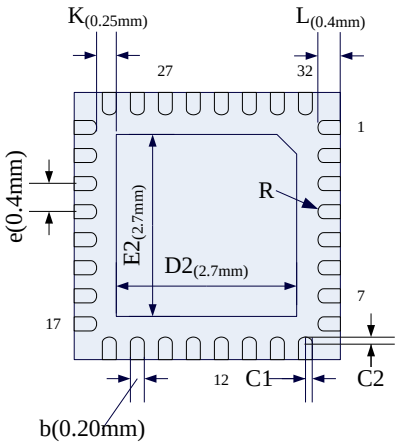


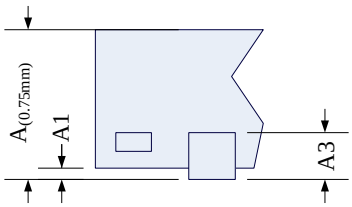
一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	1.45	1.55	1.65
A1	0.01	-	0.21
A2	1.35	1.40	1.45
A3	-	0.254	-
b1	0.30	0.35	0.40
b	0.31	0.37	0.43
c	-	0.127	-
D	8.80	9.00	9.20
D1	6.90	7.00	7.10
E	8.80	9.00	9.20
E1	6.90	7.00	7.10
e	0.70	0.80	0.90
L	0.43	-	0.71
L	1.00REF		
L1	0.25BSC		
R	0.1	-	0.25
R1	0.1	-	-
?	0°	-	10°

4.10 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	-

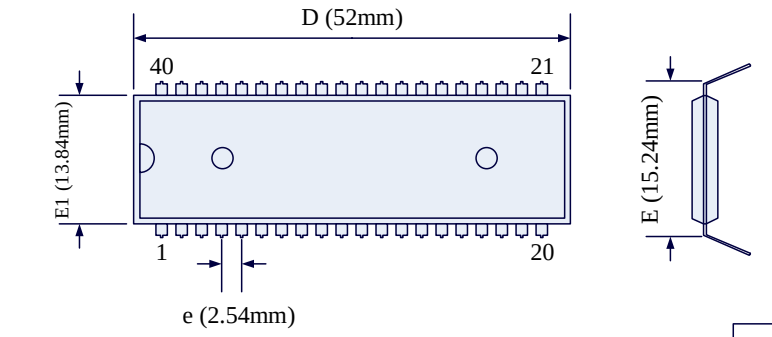


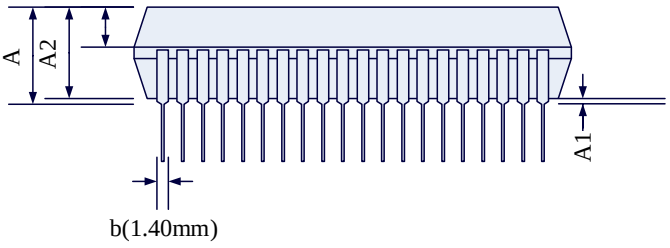


现有 QFN32 封装芯片的背面金属片（衬底），在芯片内部并未接地，在用户的 PCB 板上可以接地，也可以不接地，不会对芯片性能造成影响

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

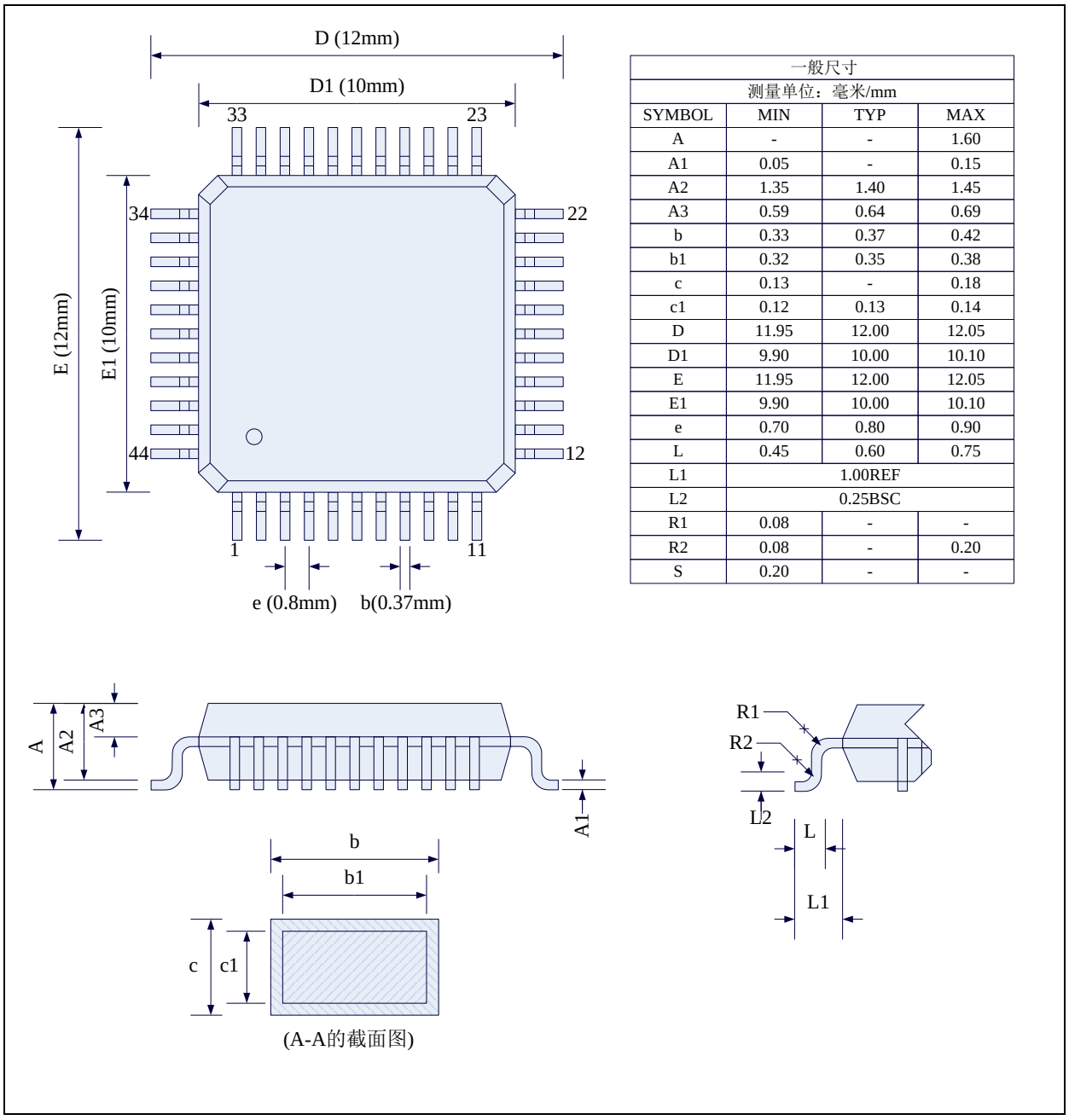
4.11 PDIP40 封装尺寸图





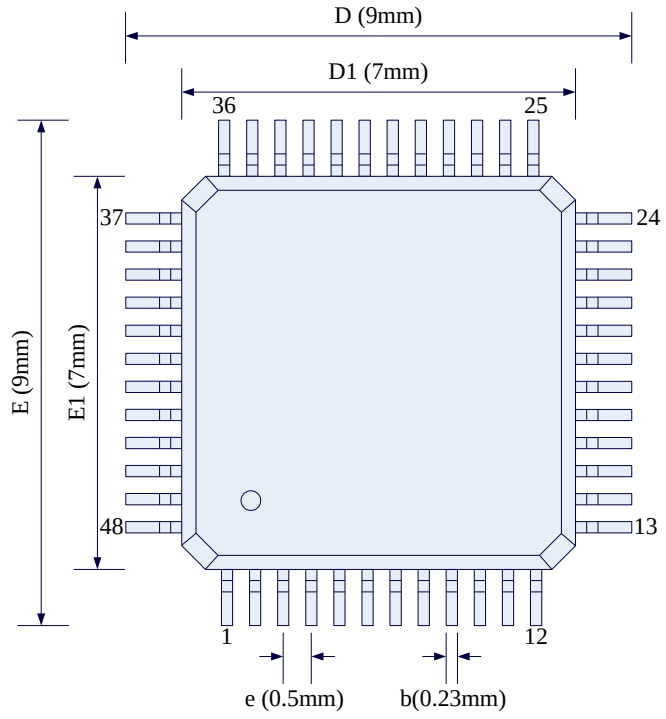
一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	-	-	4.83
A1	0.38	-	0.51
A2	3.81	3.94	4.06
b	1.14	-	1.70
D	51.5	52.00	52.5
E	-	15.24	-
E1	13.72	13.84	13.97
L	3.05	3.30	3.56

4.12 LQFP44/QFP44 封装尺寸图（12mm*12mm）

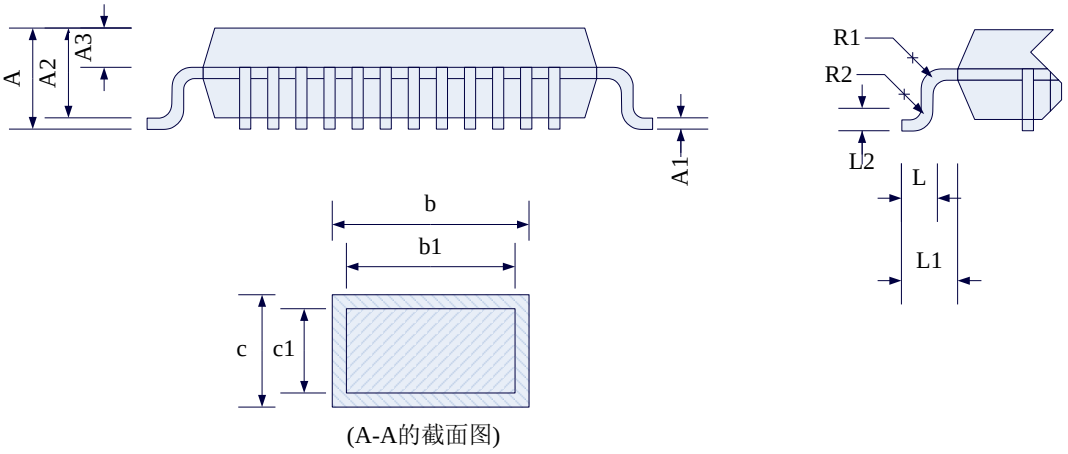


一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	-	-	1.60
A1	0.05	-	0.15
A2	1.35	1.40	1.45
A3	0.59	0.64	0.69
b	0.33	0.37	0.42
b1	0.32	0.35	0.38
c	0.13	-	0.18
c1	0.12	0.13	0.14
D	11.95	12.00	12.05
D1	9.90	10.00	10.10
E	11.95	12.00	12.05
E1	9.90	10.00	10.10
e	0.70	0.80	0.90
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25BSC		
R1	0.08	-	-
R2	0.08	-	0.20
S	0.20	-	-

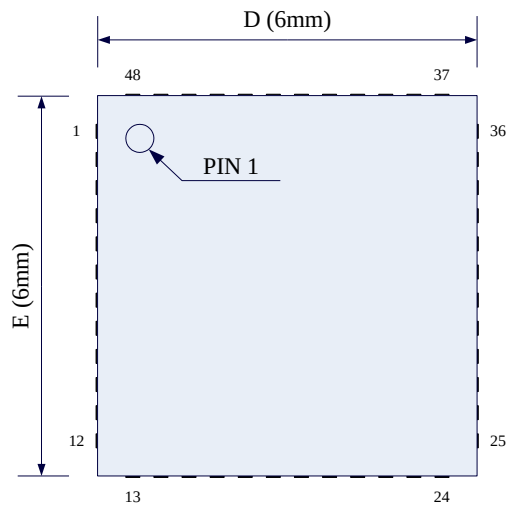
4.13 LQFP48/QFP48 封装尺寸图（9mm*9mm）



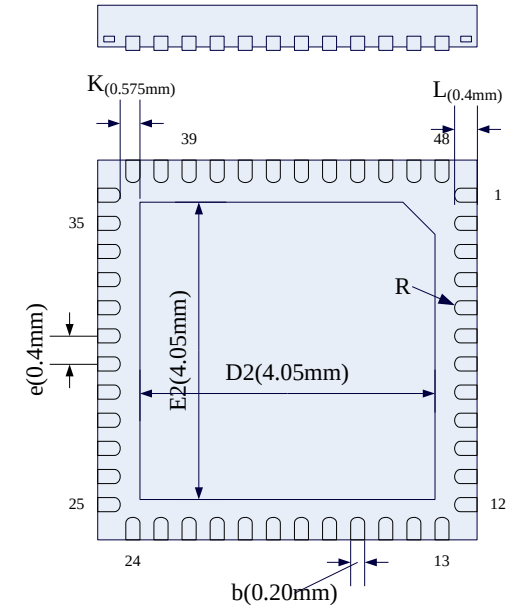
一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	-	-	1.60
A1	0.05	-	0.15
A2	1.35	1.40	1.45
A3	0.59	0.64	0.69
b	0.18	0.23	0.27
b1	0.17	0.20	0.23
c	0.13	-	0.18
c1	0.12	0.127	0.134
D	8.80	9.00	9.20
D1	6.90	7.00	7.10
E	8.80	9.00	9.20
E1	6.90	7.00	7.10
e	0.45	0.50	0.55
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25BSC		
R1	0.08	-	-
R2	0.08	-	0.20
S	0.20	-	-

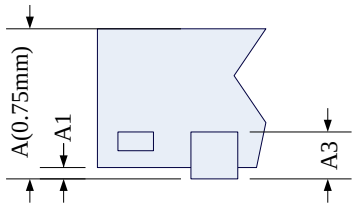


4.14 QFN48 封装尺寸图（6mm*6mm）



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	-	-

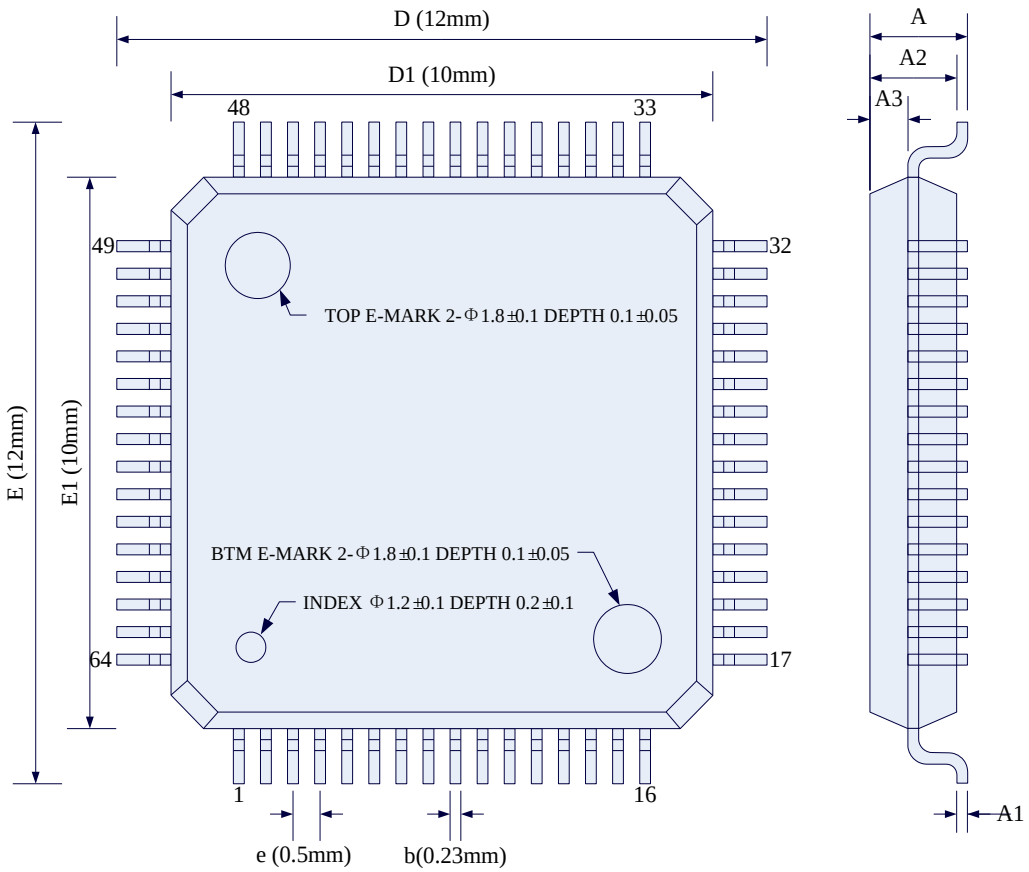




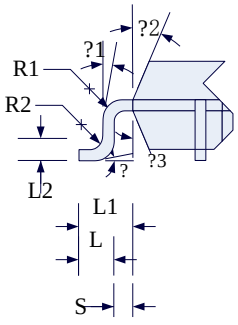
现 Ai8051U 的 QFN48 封装芯片的背面金属片（衬底），接了内部 ADC 的模拟地线，在用户的 PCB 板上可以接模拟地，也可以不接地，不会对芯片性能造成影响。

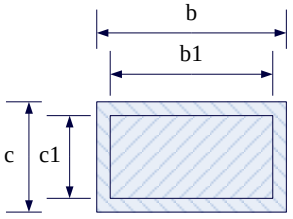
特别说明：造 PCB 的封装库时，焊盘可以向外延伸 0.1～0.2mm，但绝对不能向内延伸，必须确保 K 不能小于 0.2mm，否则会造成管脚短路

4.15 LQFP64 封装尺寸图（12mm*12mm）



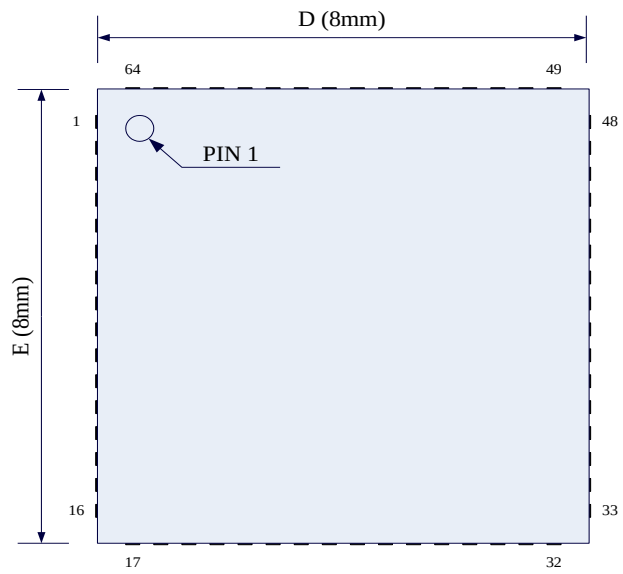
一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	-	-	1.60
A1	0.05	-	0.15
A2	1.35	1.40	1.45
A3	0.59	0.64	0.69
b	0.18	0.23	0.27
b1	0.17	0.20	0.23
c	0.13	-	0.18
c1	0.12	0.127	0.134
D	11.80	12.00	12.20
D1	9.90	10.00	10.10
E	11.80	12.00	12.20
E1	9.90	10.00	10.10
e	0.50BSC		
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25BSC		
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°



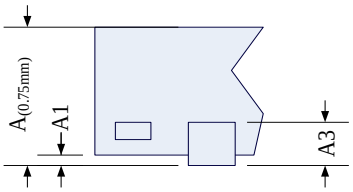


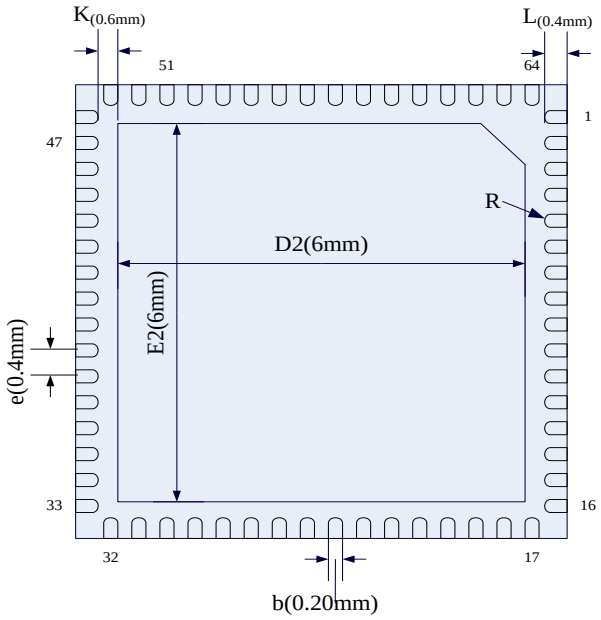
(A-A的截面图)

4.16 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	-	-

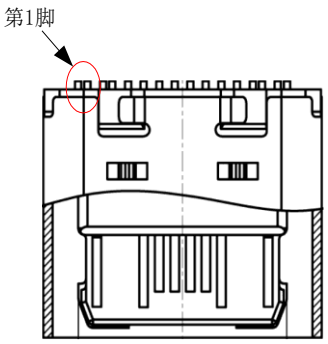




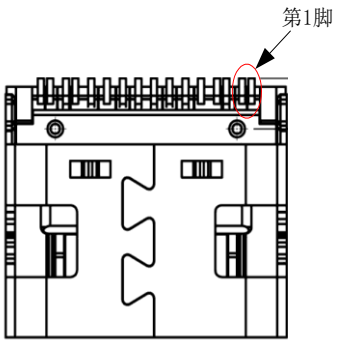
现有 QFN64 封装芯片的背面金属片（衬底），在芯片内部并未接地，在用户的 PCB 板上可以接地，也可以不接地，不会对芯片性能造成影响

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

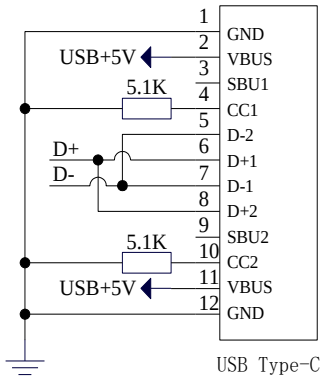
4.17 USB-TypeC 封装尺寸图，原理图



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	

5 STC15W4K32S4 系列与 STC15F/L2K60S2 系列单片机的区别

1、工作电压：

- STC15W4K32S4 系列为宽电压单片机，工作电压为 2.5V--5.5V；
- STC15F/L2K60S2 系列单片机分 5V 和 3V 单片机，其中 5V 单片机（STC15F2K60S2）的电压为 5.5V--4.5V，3V 单片机（STC15L2K60S2）的电压为 3.6V--2.4V。

2、SRAM：

- STC15W4K32S4 系列单片机具有 4K 的 SRAM；
- STC15F2K60S2 系列具有 2K 的 SRAM。

3 串行口：

- STC15W4K32S4 系列单片机具有 4 个串行口（串行口 1/串行口 2/串行口 3/串行口 4，分时复用可当 9 组串口使用）；
- STC15F/L2K60S2 系列单片机具有 2 个串行口（串行口 1/串行口 2，分时复用可当 5 组串口使用）。

4、CCP/PCAIPWM：

- STC15W4K32S4 系列单片机具有 6 通道 15 位专门的高精度 PWM（带死区控制）和 2 通道 CCP（利用它的高速脉冲输出功能可实现 11 ~ 16 位 PWM）；
- STC15F/L2K60S2 系列单片机具有 3 通道捕获/比较单元（CCP/PWM/PCA）。

5、SPI 时钟速度：

STC15W 系列与 STC15F/系列具有不同的 SPI 时钟频率，其中：

- STC15W 系列单片机的 SPI 时钟频率的选择

SPR1	SPR0	时钟（ SCLK ）
0	0	CPU_CLK/4
0	1	CPU_CLK/8
1	0	CPU_CLK/16
1	1	CPU_CLK/32

- STC15F/L 系列单片机的 SPI 时钟频率选择

SPR1	SPR0	时钟（ SCLK ）
0	0	CPU_CLK/4
0	1	CPU_CLK/16
1	0	CPU_CLK/64
1	1	CPU_CLK/128

表中，CPU_CLK 是 CPU 时钟，SPR1 和 SPR0 为 SPI 控制寄存器的 B1 和 B0。

SPCTL ： SPI 控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
SPCTL	CEH	name	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR1	SPR0

6、定时器/计数器：

- STC15W4K32S4 系列单片机具有 5 个 16 位可重装载定时器/计数器（T0/T1/T2/T3/T4），另外 2 通道 CCP 可再实现 2 个定时器；

- STC15F/L2K60S2 系列单片机具有 3 个 16 位可重装载定时器/计数器（T0/T1/T2），另外 3 通道 CCP 可再实现 3 个定时器。

7、比较器：

- STC15W4K32S4 系列单片机具有比较器功能，该比较器可当 1 路 ADC 使用，可作掉电检测支持外部管脚 CMP+与外部管脚 CMP-进行比较，可产生中断，并可在管脚 CMPO 上产生输出（可设置极性），也支持外部管脚 CMP+与内部参考电压进行比较；
- STC15F/L2K60S2 系列单片机不具有比较器功能。

8、管脚：

- STC15W4K32S4 系列单片机新增 12 个与 6 通道 15 位专门的高精度 PWM 相关的 I/O 口（[P3.7/PWM2，P2.1/PWM3，P2.2/PWM4，P2.3/PWM5，P1.6/PWM6，P1.7/PWM7，P2.7/PWM2_2，P4.5/PWM3_2，P4.4/PWM4_2，P4.2/PWM5_2，P0.7/PWM6_2，P0.6/PWM7_2]），该 12 个 I/O 口上电复位后是高阻输入（既不向外输出电流也不向内输出电流），若要使其能对外能输出，要用软件将其改设为强推挽输出或准双向口/弱上拉；
- STC15F/L2K60S2 系列单片机没有这些 I/O 口。

9、支持 USB 直接下载：

- STC15W4K32S4 系列单片机中以 STC15W4K 开头的单片机和 IAP15W4K58S4 单片机支持 USB 直接下载线路；
- STC15F/L2K60S2 系列单片机不支持 USB 直接下载线路

10、时钟分频输出：

- STC15W4K32S4 系列单片机是在 P5.4/SysClkO 或 P1.6/XTAL2/SysClkO_2 对外分频输出系统时钟，并可如下分频 SysClk/1，SysClk/2，SysClk/4，SysClk/16。SysClk 是指系统时钟频率，SysClkO 是指系统时钟输出。

  系统时钟是指对主时钟进行分频后供给 CPU、定时器、串行口、SPI、CCP/PWM/PCA、A/D 转换的实际工作时钟：STC15W4K32S4 系列单片机的主时钟可以是内部 R/C 时钟，也可以是外部输入的时钟或外部晶体振荡产生的时钟。
- STC15F/L2K60S2 系列单片机是在在 P5.4/MCLKO 对外分频输出主时钟，并可如下分频 MCLK/1，MCLK/2，MCLK/4。MCLK 是指主时钟频率，MCLKO 是指主时钟输出。
  现供货的 STC15F2K60S2 系列 C 版单片机的主时钟只可以是内部 R/C 时钟。

6 特殊外围设备(CCP/SPI，串口 1/2/3/4)在不同口间进行切换

CCP：是英文单词的缩写 Capture（捕获），Compare（比较），PWM（脉宽调制）

STC15W4K60S4 的特殊外围设备 CCP/PWM、SPI、串口 1、串口 2、串口 3、串口 4 可以在多个口之间进行任意切换。

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000 0000
P_SW2	BAH	Peripheral function switch			PWM67_S	PWM2345_S		S4_S	S3_S	S2_S	xxxx x000

CCP 可在 3 个地方切换，由 CCP_S1 / CCP_S0 两个控制位来选择		
CCP_S1	CCP_S0	CCP 可在 P1/P2/P3 之间来回切换
0	0	CCP 在[P1.2/ECI，P1.1/CCP0，P1.0/CCP1]
0	1	CCP 在[P3.4/ECI_2，P3.5/CCP0_2，P3.6/CCP1_2]
1	0	CCP 在[P2.4/ECI_3，P2.5/CCP0_3，P2.6/CCP1_3]
1	1	无效

PWM2/PWM3/PWM4/PWM5/PWMFLT 可在 2 个地方切换，由 PWM2345_S 控制位来选择	
PWM2345_S	切换 PWM2/PWM3/PWM4/PWM5/PWMFLT 管脚
0	PWM2/PWM3/PWM4/PWM5/PWMFLT 在 [P3.7/PWM2， P2.1/PWM3， P2.2/PWM4， P2.3/PWM5，P2.4/PWMFLT]
1	PWM2/PWM3/PWM4/PWM5/PWMFLT 在[P2.7/PWM2_2，P4.5/PWM3_2，P4.4/PWM4_2，P4.2/PWM5_2，P0.5/PWMFLT_2]

PWM6/PWM7 可在 2 个地方切换，由 PWM67_S 控制位来选择	
PWM67_S	切换 PWM6/PWM7 管脚
0	PWM6/PWM7 在[P1.6/PWM6，P1.7/PWM7]
1	PWM6/PWM7 在[P0.7/PWM6_2，P0.6/PWM7_2]

SPI 可在 3 个地方切换，由 SPI_S1 / SPI_S0 两个控制位来选择		
SPI_S1	SPI_S0	SPI 可在 P1/P2/P4 之间来回切换
0	0	SPI 在[P1.2/SS，P1.3/MOSI，P1.4/MISO，P1.5/SCLK]
0	1	SPI 在[P2.4/SS_2，P2.3/MOSI_2，P2.2/MISO_2，P2.1/SCLK_2]
1	0	SPI 在[P5.4/SS_3，P4.0/MOSI_3，P4.1/MISO_3，P4.3/SCLK_3]
1	1	无效

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000 0000
P_SW2	BAH	Peripheral function switch			PWM67_S	PWM2345_S		S4_S	S3_S	S2_S	xxxx x000

串口 1/S1 可在 3 个地方切换，由 S1_S0 及 S1_S1 控制位来选择		
S1_S1	S1_S0	串口 1/S1 可在 P1/P3 之间来回切换
0	0	串口 1/S1 在[P3.0/RxD， P3.1/TxD]
0	1	串口 1/S1 在[P3.6/RxD_2， P3.7/TxD_2]
1	0	串口 1/S1 在[P1.6/RxD_3/XTAL2， P1.7/TxD_3/XTAL1] 串口 1 在 P1 口时要使用内部时钟
1	1	无效

串口 2/S2 可在 2 个地方切换，由 S2_S 控制位来选择	
S2_S	S2 可在 P1/P4 之间来回切换
0	串口 2/S2 在[P1.0/RxD2， P1.1/TxD2]
1	串口 2/S2 在[P4.6/RxD2_2， P4.7/TxD2_2]

串口 3/S3 可在 2 个地方切换，由 S3_S 控制位来选择	
S3_S	S3 可在 P0/P5 之间来回切换
0	串口 3/S3 在[P0.0/RxD3，P0.1/TxD3]
1	串口 3/S3 在[P5.0/RxD3_2，P5.1/TxD3_2]

串口 4/S4 可在 2 个地方切换，由 S4_S 控制位来选择	
S4_S	S4 可在 P0/P5 之间来回切换
0	串口 4/S4 在[P0.2/RxD4，P0.3/TxD4]
1	串口 4/S4 在[P5.2/RxD4_2，P5.3/TxD4_2]

DPS：DPTR registers select bit. DPTR 寄存器选择位

0：DPTR0 is selected DPTR0 被选择

1：DPTRl is selected DPTR1 被选择

6.1 CCP/PWM/PCA 在多个口之间切换的测试程序(C 和汇编)

CCP：是下列英文单词的缩写 Capture(捕获)，Compare(比较)，PWM(脉宽调制)

1.C 程序:

```
/*--------------------STC15F2K60S2 系列 CCP 在多个口之同切换举例----------------------*/
/*在 Kei C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可---------- */
//假定测试芯片的工作频率为 18.432MHz

#include "reg51.h"
#define     FOSC        18432000L
//-----------------------------------------
sfr         P_SW1 = 0xA2;                   //外设功能切换寄存器 1
#define     CCP_S0      0x10                //P_SW1.4
#define     CCP_S1      0x20                //P_SW1.5
//-----------------------------------------
void main()
{
    ACC = P_SW1;
    ACC &= ~(CCP_S0 | CCP_S1);              //CCP_S0=0 CCP_S1=0
    P_SW1 = ACC;                            //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1)

//  ACC = P_SW1;
//  ACC &= ~(CCP_S0 | CCP_S1);              //CCP_S0=1 CCP_S1=0
//  ACC |= CCP_S0;                          //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2)
//  P_SW1 = ACC;
//
//  ACC = P_SW1;
//  ACC &= ~(CCP_S0 | CCP_S1);              //CCP_S0=0 CCP_S1=1
//  ACC |= CCP_S1;                          //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3)
//  P_SW1 = ACC;

    while (1);                              //程序终止
}
```

2.汇编程序：

```
/*-------------------------------------------------------------------------------------------------*/
/*----STC15F2K60S2 系列 CCP 在多个口之间切换举例---------------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可---------*/
/*-------------------------------------------------------------------------------------------------*/
//假定测试芯片的工作频率为 18.432MHz
#define     FOSC    18432000L
;-----------------------------------------
P_SW1       EQU     0A2H                    ;外设功能切换寄存器 1
```

```
CCP_S0          EQU     10H                     ;P_SW1.4
CCP_S1          EQU     20H                     ;P_SW1.5
;-------------------------------------
        ORG     0000H
        LJMP    START                           ;复位入口
;-------------------------------------
        ORG     0100H
START:
        MOV     SP, #3FH
        MOV     A, P_SW1
        ANL     A, #0CFH                        ;CCP_S0=0 CCP_S1=0
        MOV     P_SW1, A                        ;(P1.2/ECI, P1.1/CCP0, P1.0/CCP1)

;       MOV     A, P_SW1
;       ANL     A, #0CFH                        ;CCP_S0=1 CCP_S1=0
;       ORL     A, #CCP_S0                      ;(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2)
;       MOV     P_SW1, A

;       MOV     A, P_SW1
;       ANL     A, #0CFH                        ;CCP_S0=0 CCP_S1=1
;       ORL     A, #CCP_S1                      ;(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3)
;       MOV     P_SW1, A
        SJMP    $                               ;程序终止

        END
```

6.2 PWM2/3/4/5/PWMFLT 在多个口之间切换的测试程序(C 和汇编)

1.C 程序：

```
/*----------------------PWM(脉宽调制)-----------------------------------------------------*/
/*----STC15W4K60S4 系列 PWM2/3/4/5/PWMFLT 在多个口之间切换举例----*/
/*在 Kei C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------*/
//假定测试芯片的工作频率为 18.432MHz

#include "reg51.h"
#define   FOSC          18432000L
//-----------------------------------------
sfr       P_SW2 = 0xBA;                         //外设功能切换寄存器 2
#define   PWM2345_S  0x10                       //P_SW2.4
//-----------------------------------------
void main()
{
```

```
    P_SW2 &= ~PWM2345_S;                    //PWM2345_S=0 ( P3.7/PWM2, P2.1/PWM3，
                                            //P2.2/PWM4, P2.3/PWM5, P2.4/PWMFLT )
//  P_SW2 |= PWM2345_S;                     //PWM2345_S=1 (P2.7/PWM2_2, P4.5/PWM3_2,
                                            //P4.4/PWM4_2, P4.2/PWM5_2, P0.5/PWMFLT_2)

    while (1);                              //程序终止
}
```

2.汇编程序：

```
/*-------------------------PWM(脉宽调制)------------------------------------------------------*/
/*----STC15W4K60S4 系列 PWM2/3/4/5/PWMFLT 在多个口之间切换举例---------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可--------*/
/*-------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz

#define     FOSC    18432000L
;-----------------------------------------
P_SW2       EQU     0BAH                ;外设功能切换寄存器 2
PWM2345_S   EQU     10H                 ;P_SW2.4
;-----------------------------------------
    ORG     0000H
    LJMP    START                       ;复位入口
;-----------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    ANL     P_SW2, #NOT PWM2345_S       ;PWM2345_S=0 ( P3.7/PWM2, P2.1/PWM3,
                                        ;P2.2/PWM4, P2.3/PWM5, P2.4/PWMFLT )
;   ORL     P_SW2, #PWM2345_S           ;PWM2345_S=1 (P2.7/PWM2_2, P4.5/PWM3_2,
                                        ;P4.4/PWM4_2, P4.2/PWM5_2, P0.5/PWMFLT_2)
    SJMP    $                           ;程序终止

    END
```

6.3 PWM6/PWM7 在多个口之间切换的测试程序(C 和汇编)

1.C 程序：

```
/*-----------------------PWM(脉宽调制)------------------------------------------------------*/
/*----STC15W4K60S4 系列 PWM6/PWM7 在多个口之间切换举例-------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------*/
//假定测试芯片的工作频率为 18.432MHz
```

```
#define     FOSC        18432000L
//---------------------------------------
sfr         P_SW2 = 0xBA;               //外设功能切换寄存器 2
#define     PWM67_S     0x20            //P_SW2.5
//---------------------------------------
void main()
{
    P_SW2 &= ~PWM67_S;                  //PWM67_S=0 ( P1.6/PWM6, P1.7/PWM7 )
//  P_SW2 |= PWM67_S;                   //PWM67_S=1 ( P0.7/PWM6_2, P0.6/PWM7_2 )

    while (1);                          //程序终止
}
```

2.汇编程序：

```
/*-----------------------PWM(脉宽调制)------------------------------------------------------------*/
/*----STC15W4K60S4 系列 PWM6/PWM7 在多个口之间切换举例--------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可--------*/
/*-------------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz

#define     FOSC        18432000L
;---------------------------------------
P_SW2       EQU         0BAH            ;外设功能切换寄存器 2
PWM67_S     EQU         20H             ;P_SW2.5
;---------------------------------------
    ORG     0000H
    LJMP    START                       ;复位入口
;---------------------------------------
    ORG     0100H
START:
    MOV     SP,  #3FH
    ANL     P_SW2, #NOT PWM67_S         ;PWM67_S=0 ( P1.6/PWM6, P1.7/PWM7 )
;   ORL     P_SW2, #PWM67_S             ;PWM67_S=1 ( P0.7/PWM6_2, P0.6/PWM7_2 )
    SJMP    $                           ;程序终止

    END
```

6.4 SPI 在多个口之间切换的测试程序(C 和汇编)

1.C 程序：

```
/*-------------------------------------------------------------------------------------------------*/
/*----STC15F2K60S2 系列 SPI 在多个口之间切换举例---------------------------------*/
```

```
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------*/
//假定测试芯片的工作频率为 18.432MHz
//-----------------------------------------

#include "reg51.h"
#define     FOSC        18432000L
//-----------------------------------------
sfr         P_SW1 = 0xA2;               //外设功能切换寄存器 1
#define     SPI_S0      0x04            //P_SW1.2
#define     SPI_S1      0x08            //P_SW1.3
//-----------------------------------------
void main()
{
    ACC = P_SW1;
    ACC &= ~(SPI_S0 | SPI_S1);          //SPI_S0=0 SPI_S1=0
    P_SW1 = ACC;                        //(P1.2/SS, P1.3/MOSI, P1.4/MISO, P1.5/SCLK)
//  ACC = P_SW1;

//  ACC &= ~(SPI_S0 | SPI_S1);          //SPI_S0=1 SPI_S1=0
//  ACC |= SPI_S0;                      //(P2.4/SS_2, P2.3/MOSI_2, P2.2/MISO_2, P2.1/SCLK_2)
//  P_SW1 = ACC;
//
//  ACC = P_SW1;
//  ACC &= ~(SPI_S0 | SPI_S1);          //SPI_S0=0 SPI_S1=1
//  ACC |= SPI_S1;                      //(P5.4/SS_3, P4.0/MOSI_3, P4.1/MISO_3, P4.3/SCLK_3)
//  P_SW1 = ACC;

    while (1);                          //程序终止
}
```

2.汇编程序：

```
/*-------------------------------------------------------------------------------------------*/
/*----STC15F2K60S2 系列 SPI 在多个口之间切换举例---------------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz

#define     FOSC        18432000L
;-----------------------------------------
P_SW1       EQU         0A2H            ;外设功能切换寄存器 1
SPI_S0      EQU         04H             ;P_SW1.2
SPI_S1      EQU         08H             ;P_SW1.3
;-----------------------------------------
    ORG     0000H
```

```
    LJMP    START                       ;复位入口
;----------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    MOV     A, P_SW1
    ANL     A, #0F3H                    ;SPI_S0=0 SPI_S1=0
    MOV     P_SW1, A                    ;(P1.2/SS, P1.3/MOSI, P1.4/MISO, P1.5/SCLK)

;   MOV     A, P_SW1
;   ANL     A, #0F3H                    ;SPI_S0=1 SPI_S1=0
;   ORL     A, #SPI_S0                  ;(P2.4/SS_2, P2.3/MOSI_2, P2.2/MISO_2, P2.1/SCLK_2)
;   MOV     P_SW1, A

;   MOV     A, P_SW1
;   ANL     A, #0F3H                    ;SPI_S0=0 SPI_S1=1
;   ORL     A, #SPI_S1                  ;(P5.4/SS_3, P4.0/MOSI_3, P4.1/MISO_3, P4.3/SCLK_3)
;   MOV     P_SW1, A

    SJMP    $                           ;程序终止

    END
```

6.5 串口 1 在多个口之间切换的测试程序(C 和汇编)

1.C 程序：

```
/*-------------------------------------------------------------------------------------------------*/
/*--- STC15F2K60S2 系列 串行口 1 在多个口之间切换举例-----------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052 编译,头文件包含<reg51.h>即可---------*/
/*-------------------------------------------------------------------------------------------------*/
//假定测试芯片的工作频率为 18.432MHz

#include "reg51.h"
#define     FOSC        18432000L
//-----------------------------------------
sfr         P_SW1 = 0xA2;               //外设功能切换寄存器 1
#define     S1_S0       0x40            //P_SW1.6
#define     S1_S1       0x80            //P_SW1.7
//-----------------------------------------
void main()
{
    ACC = P_SW1;
    ACC &= ~(S1_S0 | S1_S1);            //S1_S0=0 S1_S1=0
```

```
    P_SW1 = ACC;                                //(P3.0/RxD, P3.1/TxD)

//  ACC = P_SW1;
//  ACC &= ~(S1_S0 | S1_S1);                    //S1_S0=1 S1_S1=0
//  ACC |= S1_S0;                               //(P3.6/RxD_2, P3.7/TxD_2)
//  P_SW1 = ACC;
//
//  ACC = P_SW1;
//  ACC &= ~(S1_S0 | S1_S1);                    //S1_S0=0 S1_S1=1
//  ACC |= S1_S1;                               //(P1.6/RxD_3, P1.7/TxD_3)
//  P_SW1 = ACC;

    while (1);                                  //程序终止
}
```

2.汇编程序：

```
/*-------------------------------------------------------------------------------------------------*/
/*----STC15F2K60S2 系列串行口 1 在多个口之间切换举例---------------------------*/
/*----在 Keil C 开发环境中选择  Intel 8052 编译,头文件包含<reg51.h>即可--------*/
/*-------------------------------------------------------------------------------------------------*
;假定测试芯片的工作频率为 18.432MHz

#define     FOSC        18432000L
;-----------------------------------------
P_SW1       EQU         0A2H                ;外设功能切换寄存器 1
S1_S0       EQU         40H                 ;P_SW1.6
S1_S1       EQU         80H                 ;P_SW1.7
;-----------------------------------------
    ORG     0000H
    LJMP    START                           ;复位入口
;-----------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    MOV     A, P_SW1

    ANL     A, #03FH                        ;S1_S0=0 S1_S1=0
    MOV     P_SW1, A                        ;(P3.0/RxD, P3.1/TxD)

;   MOV     A, P_SW1
;   ANL     A, #03FH                        ;S1_S0=1 S1_S1=0
;   ORL     A, #S1_S0                       ;(P3.6/RxD_2, P3.7/TxD_2)
;   MOV     P_SW1, A

;   MOV     A, P_SW1
```

```
;       ANL     A, #03FH                        ;S1_S0=0 S1_S1=1
;       ORL     A, #S1_S1                       ;(P1.6/RxD_3, P1.7/TxD_3)
;       MOV     P_SW1, A
        SJMP    $                               ;程序终止

        END
```

6.6 串口 2 在多个口之间切换的测试程序(C 和汇编)

1.C 程序：

```
/*-------------------------------------------------------------------------------------------------*/
/*----STC15F2K60S2 系列串行口 2 在多个口之间切换举例--------------------------*/
/*----在 Keil C 开发环境中选择  Intel 8052 编译,头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------------
//假定测试芯片的工作频率为 18.432MHz

#include "reg51.h"
#define         FOSC            18432000L
//-----------------------------------------
sfr             P_SW2 = 0xBA;                   //外设功能切换寄存器 2
#define         S2_S            0x01            //P_SW2.0
//-----------------------------------------
void main()
{
        P_SW2 &= ~S2_S;                         //S2_S0=0 (P1.0/RxD2, P1.1/TxD2)
//      P_SW2 |= S2_S;                          //S2_S0=1 (P4.6/RxD2_2, P4.7/TxD2_2)

        while (1);                              //程序终止

}
```

2.汇编程序：

```
/*-------------------------------------------------------------------------------------------------*/
/*---- STC15F2K60S2 系列串行口 2 在多个口之间切换举例--------------------------*/
/*----  在 Keil C 开发环境中选择  Intel 8052 编译,头文件包含<reg51.h>即可--------*/
/*-------------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz

#define         FOSC            18432000L
;---------------------------------------
P_SW2           EQU             0BAH            ;外设功能切换寄存器 2
S2_S            EQU             01H             ;P_SW2.0
```

```
;----------------------------------------
    ORG     0000H
    LJMP    START                           ;复位入口
;----------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    ANL     P_SW2, #NOT S2_S                ;S2_S0=0 (P1.0/RxD2, P1.1/TxD2)
;   ORL     P_SW2, #S2_S                    ;S2_S0=1 (P4.6/RxD2_2, P4.7/TxD2_2)
    SJMP    $                               ;程序终止

    END
```

6.7 串口 3 在多个口之间切换的测试程序(C 和汇编)

1.C 程序：

```
/*-------------------------------------------------------------------------------------------------*/
/*---- STC15F2K60S2 系列串口 3 在多个口之间切换举例---------------------------*/
/*----  在 Keil C 开发环境中选择  Intel 8052 编译,头文件包含<reg51.h>即可----*/
/*-------------------------------------------------------------------------------------------------*/
//假定测试芯片的工作频率为 18.432MHz

#include "reg51.h"
#define     FOSC        18432000L
//-----------------------------------------
sfr         P_SW2 = 0xBA;               //外设功能切换寄存器 2
#define     S3_S        0x02            //P_SW2.1
//-----------------------------------------
void main()
{
    P_SW2 &= ~S3_S;                     //S3_S0=0 (P0.0/RxD3, P0.1/TxD3)
//  P_SW2 |= S3_S;                      //S3_S0=1 (P5.0/RxD3_2, P5.1/TxD3_2)

    while (1);                          //程序终止

}
```

2.汇编程序：

```
/*-------------------------------------------------------------------------------------------------*/
/*---- STC15F2K60S2 系列串口 3 在多个口之间切换举例---------------------------*/
/*----  在 Keil C 开发环境中选择 Intel 8052 编译,头文件包含<reg51.h>即可----*/
/*-------------------------------------------------------------------------------------------------*
```

```
;假定测试芯片的工作频率为 18.432MHz

#define     FOSC        18432000L
;-------------------------------------
P_SW2       EQU         0BAH            ;外设功能切换寄存器 2
S3_S        EQU         02H             ;P_SW2.1

;-------------------------------------
    ORG     0000H
    LJMP    START                       ;复位入口
;-------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    ANL     P_SW2, #NOT S3_S            ;S3_S0=0 (P0.0/RxD3, P0.1/TxD3)
;   ORL     P_SW2, #S3_S                ;S3_S0=1 (P5.0/RxD3_2, P5.1/TxD3_2)
    SJMP    $                           ;程序终止

    END
```

6.8 串口 4 在多个口之间切换的测试程序(C 和汇编)

1.C 程序：

```
/*-------------------------------------------------------------------------------------------*/
/*----STC15W4K60S4 系列串口 4 在多个口之间切换举例-------------------------*/
/*----在 Keil C 开发环境中选择  Intel 8052 编译,头文件包含<reg51.h>即可----*/
/*-------------------------------------------------------------------------------------------
//假定测试芯片的工作频率为 18.432MHz

#include "reg51.h"
#define     FOSC        18432000L
//-------------------------------------
sfr         P_SW2 = 0xBA;               //外设功能切换寄存器 2
#define     S4_S        0x04            //P_SW2.2
//-------------------------------------
void main()
{
    P_SW2 &= ~S4_S;                     //S4_S0=0 (P0.2/RxD4, P0.3/TxD4)
//  P_SW2 |= S4_S;                      //S4_S0=1 (P5.2/RxD4_2, P5.3/TxD4_2)

    while (1);                          //程序终止
}
```

2.汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/*----STC15W4K60S4 系列串口 4 在多个口之间切换举例-------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译,头文件包含<reg51.h>即可----*/
/*---------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz

#define     FOSC        18432000L
;-----------------------------------------
P_SW2       EQU         0BAH            ;外设功能切换寄存器 2
S4_S0       EQU         04H             ;P_SW2.2

;-----------------------------------------
    ORG     0000H
    LJMP    START                       ;复位入口
;-----------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    ANL     P_SW2, #NOT S4_S            ;S4_S0=0 (P0.2/RxD4, P0.3/TxD4)
;   ORL     P_SW2, #S4_S                ;S4_S0=1 (P5.2/RxD4_2, P5.3/TxD4_2)

    SJMP    $                           ;程序终止

    END
```

7 每个单片机具有全球唯一身份证号码(ID 号)及其测试程序

STC 最新一代 STC15 系列单片机出厂时都具有全球唯一身份证号码（ID 号）。最新 STC15 系列单片机的程序存储器的最后 7 个字节单元的值是全球唯一 D 号，用户不可修改，但 IAP15 系列单片机的整个程序区是开放的，可以修改。建议利用全球唯一 ID 号加密时，使用 STC15 系列单片机，并将 EEPROM 功能使用上，从 EEPROM 起始地址 0000H 开始使用，可以有效杜绝对全球唯一 ID 号的攻击。

除程序存储器的最后 7 个字节单元的内容是全球唯一 ID 号外，单片机内部 RAM 的 F1H ~ F7H 单元（对于 STC15F104W 系列及 STC15W104SW 系列单片机是内部 RAM 的 71H--77H 单元）的内容也为全球唯一 ID 号。用户可以在单片机上电后读取内部 RAM 单元 F1H--F7H（对于 STC15F104W 系列及 STC15W104SW 系列单片机是内部 RAM 单元 71H--77H）连续 7 个单元的值来获取此单片机的唯一身份证号码（ID 号）,使用“ MOV @Ri”指令来读取。如果用户需要用全球唯一 ID 号进行用户自己的软件加密，建议用户在程序的多个地方有技巧地判断自己的用户程序有无被非法修改，提高解密的难度，防止解密者修改程序，绕过对全球唯一 1D 号的判断。

使用程序区的最后 7 个字节的全球唯一 ID 号比使用内部 RAM 单元 F1H--F7H（或内部 RAM 单元 71H--77H）的全球唯一 ID 号更难被攻击。建议用户使用程序区最后 7 个字节的全球唯一 ID 号，而不要使用内部 RAM 单元 F1H--F7H（或内部 RAM 单元 71H--77H）的全球唯一 ID 号。

//从 RAM 区和程序区获取全球唯一身份证号码(ID 号)的程序举例

1.C 程序

```
/*-------------------------------------------------------------------------------------------------*/
/*---- STC15F2K60S2 系列 获取全球唯一身份证号码（ID 号）举例----------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可----------*/
/*-------------------------------------------------------------------------------------------------*/
//假定测试芯片的工作频率为 18.432MHz

#include "reg51.h"
typedef     unsigned char   BYTE;
typedef     unsigned int    WORD;
#define     URMD            0           //0:使用定时器 2 作为波特率发生器
                                        //1:使用定时器 1 的模式 0(16 位自动重载模式)作为波特率发生器
                                        //2:使用定时器 1 的模式 2(8 位自动重载模式)作为波特率发生器
sfr         T2H = 0xd6;                 //定时器 2 高 8 位
sfr         T2L = 0xd7;                 //定时器 2 低 8 位
sfr         AUXR = 0x8e;                //辅助寄存器
#define     ID_ADDR_RAM     0xf1        //ID 号的存放在 RAM 区的地址为 0F1H
                                        //ID 号的存放在程序区的地址为程序空间的最后 7 字节
//#define   ID_ADDR_ROM     0x03f9      //1K 程序空间的 MCU(如 STC15F201EA, STC15F101EA)
//#define   ID_ADDR_ROM     0x07f9      //2K 程序空间的 MCU(如 STC15F402AD,
                                        //STC15F202EA, STC15F102EA)
//#define   ID_ADDR_ROM     0x0bf9      //3K 程序空间的 MCU(如 STC15F203EA, STC15F103EA)
//#define   ID_ADDR_ROM     0x0ff9      //4K 程序空间的 MCU(如 STC15F404AD, STC15F204EA,
                                        //STC15F104EA)
```

```
//#define     ID_ADDR_ROM   0x13f9      //5K 程序空间的 MCU(如 STC15F206EA, STC15F106EA)
//#define     ID_ADDR_ROM   0x1ff9      //8K 程序空间的 MCU(如 STC15F2K08S2, STC15F1K08AD,
                                        //STC15F408AD)
//#define     ID_ADDR_ROM   0x27f9      //10K 程序空间的 MCU(如 STC15F410AD)
//#define     ID_ADDR_ROM   0x2ff9      //12K 程序空间的 MCU(如 STC15F408AD)
//#define     ID_ADDR_ROM   0x3ff9      //16K 程序空间的 MCU(如 STC15F2K16S2, //STC15F1K16AD)
//#define     ID_ADDR_ROM   0x4ff9      //20K 程序空间的 MCU(如 STC15F2K20S2)
//#define     ID_ADDR_ROM   0x5ff9      //24K 程序空间的 MCU
//#define     ID_ADDR_ROM   0x6ff9      //28K 程序空间的 MCU
//#define     ID_ADDR_ROM   0x7ff9      //32K 程序空间的 MCU(如 STC15F2K32S2)
//#define     ID_ADDR_ROM   0x9ff9      //40K 程序空间的 MCU(如 STC15F2K40S2)
//#define     ID_ADDR_ROM   0xbff9      //48K 程序空间的 MCU(如 STC15F2K48S2)
//#define     ID_ADDR_ROM   0xcff9      //52K 程序空间的 MCU(如 STC15F2K52S2)
//#define     ID_ADDR_ROM   0xdff9      //56K 程序空间的 MCU(如 STC15F2K56S2)

#define     ID_ADDR_ROM   0xeff9      //60K 程序空间的 MCU(如 STC15W4K60S4)

//----------------------------------------
void InitUart();
void SendUart(BYTE dat);
//----------------------------------------
void main()
{
    BYTE    idata   *iptr;
    BYTE    code    *cptr;
    BYTE    i;
    InitUart();                     //串口初始化
    iptr = ID_ADDR_RAM;             //从 RAM 区读取 ID 号
    for (i=0; i<7; i++)             //读 7 个字节
    {
        SendUart(*iptr++);          //发送 ID 到串口
    }
    cptr = ID_ADDR_ROM;             //从程序区读取 ID 号
    for (i=0; i<7; i++)             //读 7 个字节
    {
        SendUart(*cptr++);          //发送 ID 到串口
    }
    while (1);                      //程序终止
}
/*---------------------------
串口初始化
---------------------------*/
void InitUart()
{
    SCON = 0x5a;                    //设置串口为 8 位可变波特率
#if   URMD == 0
```

```
    T2L = 0xd8;                         //设置波特率重装值
    T2H = 0xff;                         //115200 bps(65536-18432000/4/115200)
    AUXR = 0x14;                        //T2 为 1T 模式, 并启动定时器 2
    AUXR |= 0x01;                       //选择定时器 2 为串口 1 的波特率发生器
#elif URMD == 1
    AUXR = 0x40;                        //定时器 1 为 1T 模式
    TMOD = 0x00;                        //定时器 1 为模式 0(16 位自动重载)
    TL1 = 0xd8;                         //设置波特率重装值
    TH1 = 0xff;                         //115200 bps(65536-18432000/4/115200)
    TR1 = 1;                            //定时器 1 开始启动
#else
    TMOD = 0x20;                        //设置定时器 1 为 8 位自动重装载模式
    AUXR = 0x40;                        //定时器 1 为 1T 模式
    TH1 = TL1 = 0xfb;                   //115200 bps(256 - 18432000/32/115200)
    TR1 = 1;
#endif
}
/*---------------------------
发送串口数据
---------------------------*/
void SendUart(BYTE dat)
{
    while (!TI);                        //等待前面的数据发送完成
    TI = 0;                             //清除发送完成标志
    SBUF = dat;                         //发送串口数据
}
```

2.汇编程序

```
/*-------------------------------------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 获取全球唯一身份证号码(ID 号)举例----------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz

#define     URMD        0       ;0:使用定时器 2 作为波特率发生器
                                ;1:使用定时器 1 的模式 0(16 位自动重载模式)作为波特率发生器
                                ;2:使用定时器 1 的模式 2(8 位自动重载模式)作为波特率发生器

T2H         DATA        0D6H        ;定时器 2 高 8 位
T2L         DATA        0D7H        ;定时器 2 低 8 位
AUXR        DATA        08EH        ;辅助寄存器
;----------------------------------------
#define     ID_ADDR_RAM     0xf1    ;ID 号的存放在 RAM 区的地址为 0F1H
                                    ;ID 号的存放在程序区的地址为程序空间的最后 7 字节
;#define    ID_ADDR_ROM     0x03f9  ;1K 程序空间的 MCU(如 STC15F201EA, STC15F101EA)
```

```
;#define     ID_ADDR_ROM     0x07f9  ;2K 程序空间的 MCU(如  STC15F402AD, STC15F202EA,
                                    ; STC15F102EA)
;#define     ID_ADDR_ROM     0x0bf9  ;3K 程序空间的 MCU 如 STC15F203EA, STC15F103EA)
;#define     ID_ADDR_ROM     0x0ff9  ;4K 程序空间的 MCU(如 STC15F404AD, STC15F204EA,
                                    ;STC15F104EA)
;#define     ID_ADDR_ROM     0x13f9  ;5K 程序空间的 MCU(如 STC15F206EA, STC15F106EA)
;#define     ID_ADDR_ROM     0x1ff9  ;8K 程序空间的 MCU(如 STC15F2K08S2,
                                    ;STC15F1K08AD, STC15F408AD)
;#define     ID_ADDR_ROM     0x27f9  ;10K 程序空间的 MCU(如  STC15F410AD)
;#define     ID_ADDR_ROM     0x2ff9  ;12K 程序空间的 MCU(如 STC15F408AD)
;#define     ID_ADDR_ROM     0x3ff9  ;16K 程序空间的 MCU(如 STC15F2K16S2)
;#define     ID_ADDR_ROM     0x4ff9  ;20K 程序空间的 MCU(如 STC15F2K20S2)
;#define     ID_ADDR_ROM     0x5ff9  ;24K 程序空间的 MCU
;#define     ID_ADDR_ROM     0x6ff9  ;28K 程序空间的 MCU
;#define     ID_ADDR_ROM     0x7ff9  ;32K 程序空间的 MCU(如 STC15F2K32S2)
;#define     ID_ADDR_ROM     0x9ff9  ;40K 程序空间的 MCU(如 STC15F2K40S2)
;#define     ID_ADDR_ROM     0xbff9  ;48K 程序空间的 MCU(如 STC15F2K48S2)
;#define     ID_ADDR_ROM     0xcff9  ;52K 程序空间的 MCU(如 STC15F2K52S2)
;#define     ID_ADDR_ROM     0xdff9  ;56K 程序空间的 MCU(如 STC15F2K56S2)

#define     ID_ADDR_ROM     0xeff9  ;60K 程序空间的 MCU(如 STC15W4K60S4)
;---------------------------------------
    ORG     0000H
    LJMP    START                   ;复位入口
;---------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    LCALL   INIT_UART               ;串口初始化
    MOV     R0, #ID_ADDR_RAM        ;从 RAM 区读取 ID 号
    MOV     R1, #7                  ;读 7 个字节

NEXT1:
    MOV     A, @R0
    LCALL   SEND_UART               ;发送 ID 到串口
    INC     R0
    DJNZ    R1, NEXT1
    MOV     DPTR, #ID_ADDR_ROM      ;从程序区读取 ID 号
    MOV     R1, #7                  ;读 7 个字节

NEXT2:
    CLR     A
    MOVC    A, @A+DPTR
    LCALL   SEND_UART               ;发送 ID 到串口
    INC     DPTR
    DJNZ    R1, NEXT2
```

```
    SJMP    $                       ;程序终止
/*---------------------------
串口初始化
----------------------------*/
INIT_UART:
    MOV     SCON, #5AH              ;设置串口为 8 位可变波特率

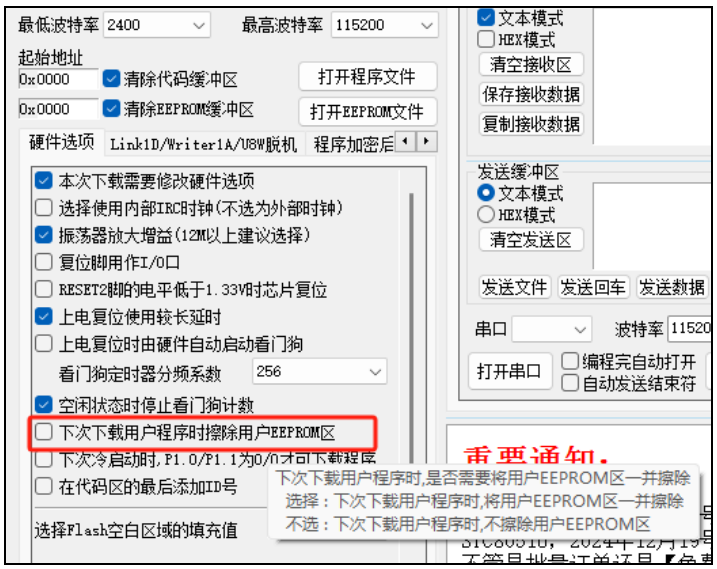
#if URMD == 0
    MOV     T2L, #0D8H              ;设置波特率重装值(65536-18432000/4/115200)
    MOV     T2H, #0FFH
    MOV     AUXR, #14H              ;T2 为 1T 模式, 并启动定时器 2
    ORL     AUXR, #01H              ;选择定时器 2 为串口 1 的波特率发生器
#elif URMD == 1
    MOV     AUXR, #40H              ;定时器 1 为 1T 模式
    MOV     TMOD, #00H              ;定时器 1 为模式 0(16 位自动重载)
    MOV     TL1, #0D8H              ;设置波特率重装值(65536-18432000/4/115200)
    MOV     TH1, #0FFH
    SETB    TR1                     ;定时器 1 开始运行
#else
    MOV     TMOD, #20H              ;设置定时器 1 为 8 位自动重装载模式
    MOV     AUXR, #40H              ;定时器 1 为 1T 模式
    MOV     TL1, #0FBH              ;115200 bps(256 - 18432000/32/115200)
    MOV     TH1, #0FBH
    SETB    TR1
#endif

    RET
/*---------------------------
发送串口数据
入口参数: ACC
出口参数: 无
----------------------------*/
SEND_UART:
    JNB     TI, $                   ;等待前面的数据发送完成
    CLR     TI                      ;清除发送完成标志
    MOV     SBUF, A                 ;发送串口数据
    RET

    END
```

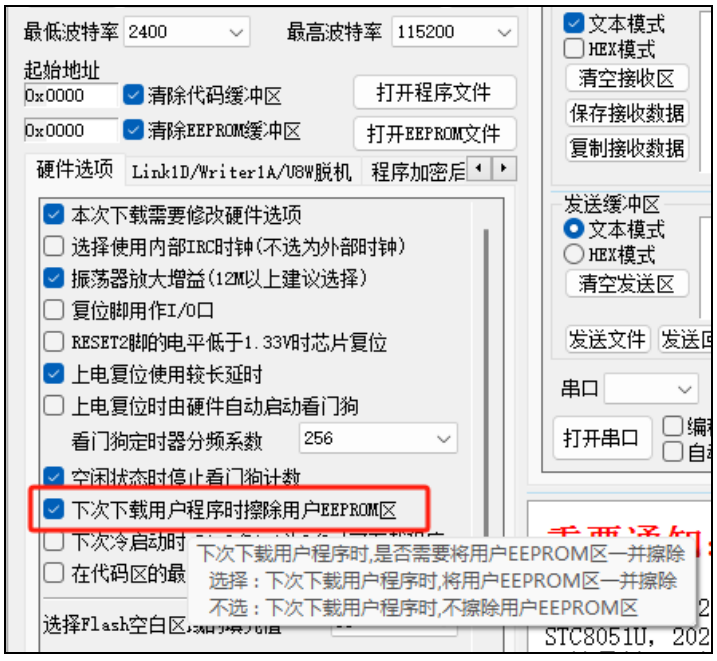
8 关于 ID 号在大批量生产中的应用方法(较多用户的用法)

（1）先烧一个程序进去(选择下次下载用户程序时不擦除用户 EEPROM 区);



(2)读程序区的 ID 号（STC15 系列是程序区的最后 7 个字节），经用户自己的复杂的加密算法对程序区的 ID 号加密运算后生成一个新的数--用户自加密 ID 号，写入 STC15 系列用户 EEPROM 区的 EEPROM;

(3)再烧一个最终出厂的程序进去（选择下次下载用户程序时将用户 EEPROM 区一并擦除），如下图所示：

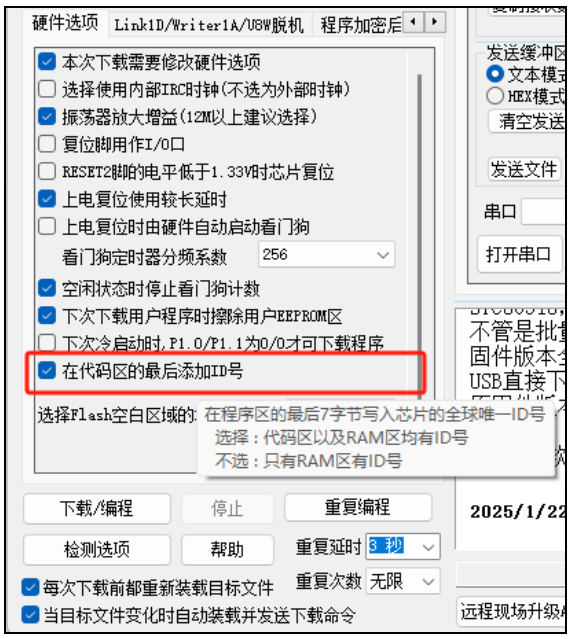


(4)在用户程序区的多处读程序区的 ID 号和用户自加密 ID 号比较（经用户自己的复杂的解密算法解密后），如不对应，则 6 个月后随机异常，或 200 次开机后随机异常最终出厂的程序不含加密算法

(5)另外，在程序区的多个地方判断用户自己的程序是否被修改，如被修改，则 6 个月后随机异常，或 200 次开机后随机异常，将不用的用户程序区用所谓的有效程序全部填满

9 在全球唯-ID 号前添加软复位指令及重要测试参数

用户可以在 AIapp-ISP 下载编程工具中设置在全球唯一身份证号码（ID 号）前增加 15 字节的软复位指令（包含重要测试参数），具体设置方法见下图。这 15 字节的软复位指令可以在程序意外跑飞时帮助 CPU 自动复位。



该 15 字节的复位指令中包含一些重要的测试参数。

当全球唯一 ID 号在程序存储器（ROM）的最后 7 个字节单元时，这些测试参数在 ROM 区的信息如下：

（1）内部 BandGap 电压值（毫伏，高字节在前）（2 字节，程序空间最后第 8 字节和第 9 字节）

例如：STC15F104W 4K 程序空间 地址为 0FF7H--0FF8H

STC15F2K60S2 60K 程序空间 地址为 EFF7H--EFF8H

（2）32K 掉电唤醒定时器频率（高字节在前）（2 字节，程序空间最后第 10 字节和第 11 字节）

例如：STC15F104W 4K 程序空间 地址为 0FF5H--0FF6H

STC15F2K60S2 60K 程序空间 地址为 EFF5H--EFF6H

（3）12MHz 内部 IRC 设定参数值（3 字节，程序空间最后第 12 字节、第 13 字节和第 14 字节）

例如：STC15F104W 4K 程序空间 地址为 0FF2H--0FF4H

STC15F2K60S2 60K 程序空间 地址为 EFF2H--EFF4H

（4）24MHz 内部 IRC 设定参数值（3 字节，程序空间最后第 15 字节、第 16 字节和第 17 字节）

例如：STC15F104W 4K 程序空间 地址为 0FEEH--0FF1H

STC15F2K60S2 60K 程序空间 地址为 EFEEH--EFFIH

当全球唯一 ID 号在内部 RAM 单元时，这些测试参数在 RAM 区的信息如下：

（1）24MHz 内部 IRC 设定参数值 1（1 字节）

例如：STC15F104W 128 字节 RAM 地址为 07FH--07FH

STC15F2K60S2 256 以上字节 RAM 地址为 0FFH--0FFH

（2）12MHz 内部 IRC 设定参数值 1（1 字节）

例如：	STC15F104W	128 字节 RAM	地址为 07EH--07EH
	STC15F2K60S2	256 以上字节 RAM	地址为 0FEH--0FEH

（3）24MHz 内部 IRC 设定参数值 2（1 字节）

例如：	STC15F104W	128 字节 RAM	地址为 07DH--07DH
	STC15F2K60S2	256 以上字节 RAM	地址为 0FDH--0FDH

（4）12MHz 内部 IRC 设定参数值 2（1 字节）

例如：	STC15F104W	128 字节 RAM	地址为 07CH--07CH
	STC15F2K60S2	256 以上字节 RAM	地址为 0FCH--0FCH

（5）24MHz 内部 IRC 设定参数值 3（1 字节）

例如：	STC15F104W	128 字节 RAM	地址为 07BH--07BH
	STC15F2K60S2	256 以上字节 RAM	地址为 0FBH--0FBH

（6）12MHz 内部 IRC 设定参数值 3（1 字节）

例如：	STC15F104W	128 字节 RAM	地址为 07AH--07AH
	STC15F2K60S2	256 以上字节 RAM	地址为 0FAH--0FAH

（7）32K 掉电唤醒定时器频率（高字节在前）（2 字节）

例如：	STC15F104W	128 字节 RAM	地址为 078H--079H
	STC15F2K60S2	256 以上字节 RAM	地址为 0F8H--0F9H

（8）内部 BandGap 电压值（毫伏，高字节在前）（2 字节）

例如：	STC15F104W	128 字节 RAM	地址为 06FH--070H
	STC15F2K60S2	256 以上字节 RAM	地址为 0EFH--0F0H

10 如何识别芯片版本号

如需知道芯片版本号，请查阅芯片表面印刷字中最下面一行的最后一个字母，该字母代表芯片版本号。若查询到芯片表面最下面一行的最后一个字母为 C，则该芯片版本为 C 版，如下图所示：



11 现供货的 STC15 系列中未实现的计划功能

计划功能是指芯片原规格说明书中计划设计的功能。这些功能在单片机指南中已作介绍，但是由于某些原因，在芯片定型生产时，现供货的 STC15F2K60S2 系列 C 版本、STC15F408AD 系列 C 版本、STC15W204S 系列 A 版本及 STC15W4K32S4 系列 A 版本的部分计划功能实际未设计进去，在此特别指出这些系列中未实现的计划功能，敬请广大客户留意!

11.1 现供货的 STC15F2K60S2 系列 C 版本中未实现的计划功能

11.1.1 现供货 STC15F2K60S2 系列 C 版本主时钟输出只可对外输出内部 R/C 时钟

----将在 STC15W2K60S2 系列中修正

STC15F2K60S2 系列单片机的主时钟在 P5.4/MCLKO 口对外输出时钟，并可如下分频 MCLK/1，MCLK/2，MCLK/4 (MCLK 为主时钟频率)。现供货的 STC15F2K60S2 系列 C 版本单片机的主时钟在 P5.4/MCLKO 口只可以对外输出内部 R/C 时钟，暂时不可以对外输出外部输入的时钟或外部晶体振荡产生的时钟，特此说明，敬请广大客户留意！用户可通过在 P1.6/XTAL2 脚串一电阻将外部输入的时钟或外部晶体振荡产生的时钟进行对外输出。

STC15F2K60S2 及 STC15L2K60S2 系列下一升级版本----STC15W2K60S2 系列单片机将会设计实现该计划功能，其主时钟将既可以对外输出内部 R/C 时钟，也可对外输出外部输入的时钟或外部晶体振荡产生的时钟。同时，STC15W2K60S2 系列单片机还将增加比较器的功能。

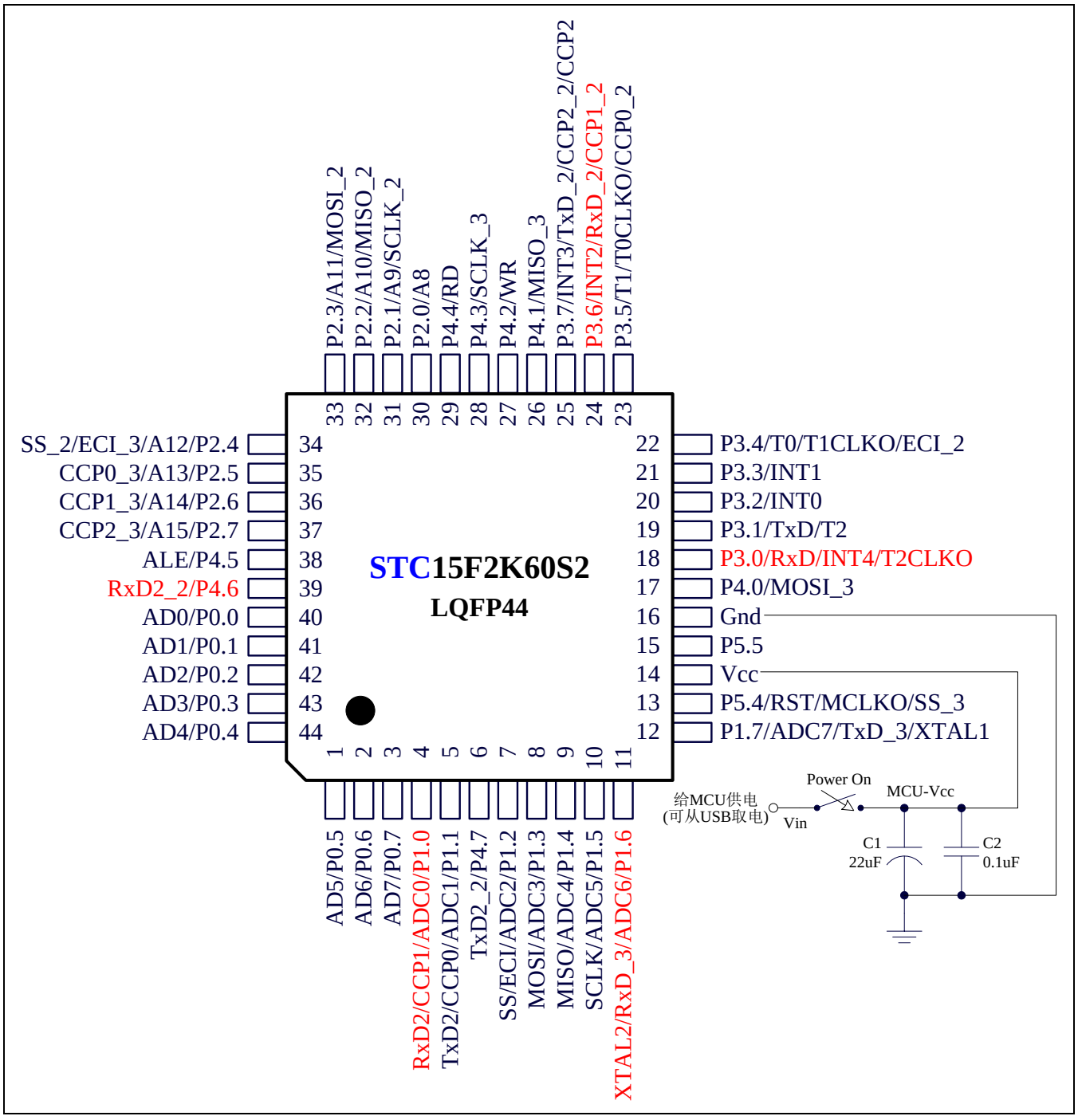
11.1.2 现供货的 STC15F2K60S2 系列 C 和 D 版本的串口 1 和串口 2 的接收管脚不能唤醒掉电/停机模式

----将在 STC15W2K60S2 系列中修正

现供货的 STC15F2K60S2 系列 C 版本和 D 版本单片机的串口 1 接收管脚(RxD/P3.0、RxD_2/P3.6、RxD_3/P1.6)和串口 2 的接收管脚(RxD2/P1.0、RxD2_2/P4.6)暂时不能将掉电模式/停机模式唤醒，特此说明，敬请广大客户留意！但是，当串口 1 接收管脚切换至管脚 P3.0/RxD/INT4 或 P3.6/RxD_2/INT2 时，其与 INT4(外部中断 4)或 INT2(外部中断 2)占用同一管脚资源，若相应的 INT4(外部中断 4)或 INT2(外部中断 2)被允许，则该管脚资源可通过 INT4 或 INT2 来唤醒掉电模式/停机模式。

STC15F2K60S2 及 STC15L2K60S2 系列下一升级版本----STC15W2K60S2 系列单片机将会设计实现该计划功能，STC15W2K60S2 系列单片机的串口 1 接收管脚(RxD/P3.0、RxD_2/P3.6、RxD_3/P1.6)和串口 2 的接收管脚(RxD2/P1.0、RxD2_2/P4.6)将均可用于唤醒掉电模式/停机模式。同时，STC15W2K60S2 系列单片机还将增加比较器的功能。

注意：现供货的 STC15F2K60S2 系列 C 版本和 D 版本的串口 2 如切换到[P4.7/TxD2_2，P4.6/RxD2_2]时，P4.7 要加 3.3K 上拉电阻，且须工作在弱上拉/准双向口模式


STC15F2K60S2
LQFP44
P2.3/A11/MOSI_2
P2.2/A10/MISO_2
P2.1/A9/SCLK_2
P2.0/A8
P4.4/RD
P4.3/SCLK_3
P4.2/WR
P4.1/MISO_3
P3.7/INT3/TxD_2/CCP2_2/CCP2
P3.6/INT2/RxD_2/CCP1_2
P3.5/T1/T0CLKO/CCP0_2
SS_2/ECI_3/A12/P2.4
CCP0_3/A13/P2.5
CCP1_3/A14/P2.6
CCP2_3/A15/P2.7
ALE/P4.5
RxD2_2/P4.6
AD0/P0.0
AD1/P0.1
AD2/P0.2
AD3/P0.3
AD4/P0.4
P3.4/T0/T1CLKO/ECI_2
P3.3/INT1
P3.2/INT0
P3.1/TxD/T2
P3.0/RxD/INT4/T2CLKO
P4.0/MOSI_3
Gnd
P5.5
Vcc
P5.4/RST/MCLKO/SS_3
P1.7/ADC7/TxD_3/XTAL1
AD5/P0.5
AD6/P0.6
AD7/P0.7
RxD2/CCP1/ADC0/P1.0
TxD2/CCP0/ADC1/P1.1
TxD2_2/P4.7
SS/ECI/ADC2/P1.2
MOSI/ADC3/P1.3
MISO/ADC4/P1.4
SCLK/ADC5/P1.5
XTAL2/RxD_3/ADC6/P1.6
Power On
MCU-Vcc
给MCU供电
(可从USB取电)
Vin
C1
22uF
C2
0.1uF

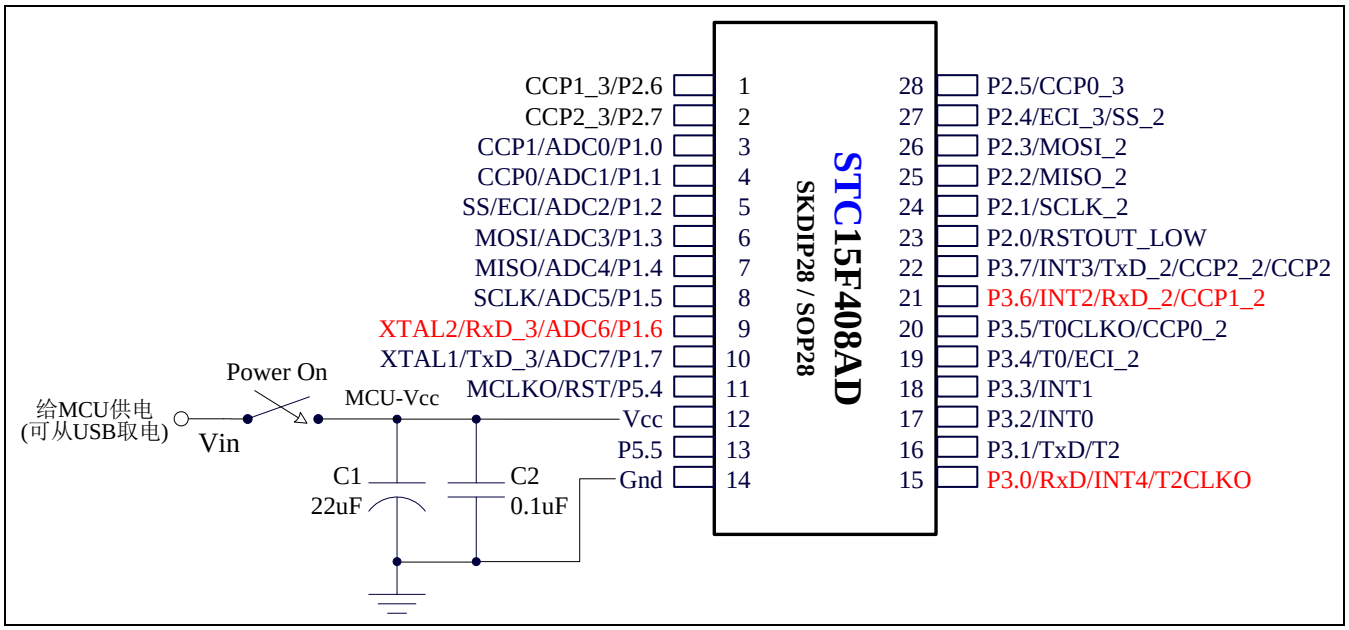
11.2 现供货的 STC15F408AD 系列 C 版本中未实现的计划功能

11.2.1 现供货的 STC15F408AD 系列 C 版本的串口 1 接收管脚不能唤醒掉电模式/停机模式

----将在 STC15W408AS 系列中修正

STC15F408AD 系列芯片已经开始供货了。现供货的 STC15F408AD 系列 C 版本单片机的串口 1 接收管脚(RxD/P3.0，RxD_2/P3.6，RxD_3/P1.6)暂时不能将掉电模式/停机模式唤醒，特此说明，敬请广大客户留意！但是，RxD 管脚与 INT4(外部中断 4)占用同一管脚资源，若 INT4(外部中断 4)被允许，则该管脚资源可通过 INT4 来唤醒掉电模式/停机模式：同样地，RxD_2 管脚与 INT2(外部中断 2)占用同一管脚资源，若 INT2(外部中断 2)被允许，则 RxD_2 管脚资源也可通过 INT2 来唤醒掉电模式/停机模式

STC15F408AD 及 STC15L408AD 系列下一升级版本----STC15W408AS 系列单片机将会设计实现该计划功能，STC15W408AS 系列单片机的 RxD 管脚将可用于唤醒掉电模式/停机模式。



12 部分 15 系列单片机的特别注意事项

12.1 SPI 的特别注意事项（仅针对以 15F 和 15L 开头的单片机）

----只支持 SPI 主机模式，不支持 SPI 从机模式

STC 单片机中以 15F 和 15L 开头且有 SPI 功能的单片机（如 STC15F2K60S2 型号及 STC15L408AD 单片机）的 SPI 从机模式暂不能使用，但它们的 SPI 主机模式可正常使用。因此，建议用户不要使用以 15F 和 15L 开头且有 SPI 功能的单片机的 SPI 从机模式。

注意，以 15W 开头的单片机不存在上述问题，以 15W 开头且有 SPI 功能的单片机既支持 SPI 主机模式，也支持 SPI 从机模式。如，STC15W408S、STC15W1K16S 等型号单片机既支持 SPI 主机模式，也支持 SPI 从机模式。

12.2 进入掉电唤醒模式的特别注意事项（仅针对以 15L 开头的单片机）

----以 15L 开头的单片机进入掉电模式前必须启动掉电唤醒定时器

STC 单片机中以 15L 开头的 C 版本及 C 版本以下单片机（如 STC15L2K60S2 型号单片机）如需进入“掉电模式”，则它进入“掉电模式”前必须启动掉电唤醒定时器（功耗为 3uA），且其掉电唤醒定时器不超过 3 秒（约 2 秒）要唤醒一次。而以 15F 和 15W 开头的单片机以及以 15L 开头的 D 版本单片机则不需要。如，STC15F2K60S2、STC15W408S 等型号单片机在进入“掉电模式”前不需要启动掉电唤醒定时器。

12.3 STC15W204S 系列 A 版本单片机的比较器下降沿中断不响应

----将在 STC15W204S 系列 B 版本中修正

STC15W204S 系列的 A 版本单片机正大批量现货供应中，STC15W204S 系列包括 STC15W201S 型号、STC15W202S 型号、STC15W203S 型号、STC15W204S 型号、IAP15W205S 型号及 IRC15W207S 型号单片机。当仅允许该系列单片机的比较器下降沿中断时，该比较器的下降沿中断暂不能使用。

但是，STC15W204S 系列 A 版本单片机的比较器下降沿中断不是绝对不能使用，用户可以通过以下两种办法解决这一问题：

一、STC15W204S 系列 A 版本的比较器上升沿中断是可正常使用的，且当用户将其比较器上升沿中断和下降沿中断都允许后，该比较器上升沿中断和下降沿中断都可以正常使用。由于比较器比较结果标志位 CMPRES（CMPCR1.0）是正确的，因此在比较器中断服务程序中查询比较器比较结果标志位 CMPRES（CMPCR1.0）的值可判断单片机进入的是比较器上升沿中断还是比较器下降沿中断，如果 CMPRES/CMPCR1.0=1，即 CMP+的电平高于 CMP-的电平（或内部 BandGap 参考电压的电平），则表示单片机进入的是比较器上升沿中断；反之，如果 CMPRES（CMPCR1.0）=0，即 CMP+的电平低于 CMP-的电平（或内部 BandGap 参考电压的电平），则表示单片机进入的是比较器下降沿中断，此时比较器下降沿中断是可正常使用的，这是解决办法之一。

二、STC15W204S 系列 A 版本的比较器比较结果标志位（CMPRES）是正确的，因此用户还可用软

件查询方式解决该这一问题。

对于上述问题，我们将在 STC15W204S 系列的下一版本，即 STC15W204S 系列 B 版本中修正。

12.4 STC15W408S 及 STC15W1K16S 系列 T0CLKO 时钟输出功能的注意事项

----若要使用 T0CLKO 时钟输出功能，须将 P3.5 口设置为强推挽输出

STC15W404S 系列单片机和 STC15W1K16S 系列单片机均已开始供货，但 STC15W404S 系列单片机和 STC15W1K16S 系列单片机若要使用其 T0CLKO 时钟输出功能，必须将 P3.5/T0CLKO 口设置为强推挽输出模式。只有将 P3.5 口设置成了强推挽输出模式，STC15W404S 系列单片机和 STC15W1K16S 系列单片机的 T0CLKO 时钟输出功能才有效。

12.5 STC15W4K32S4 系列 A 版单片机的特别注意事项

1、P1.0 和 P1.4 被误设计为强推挽输出，建议上电复位后用软件将其改设为弱上拉/准双向口或需要的模式，另外 P1.0 和 P1.4 对外时最好串 100 欧电阻

2、[T3/P0.5，T4/P0.7]在掉电模式时不要作掉电唤醒

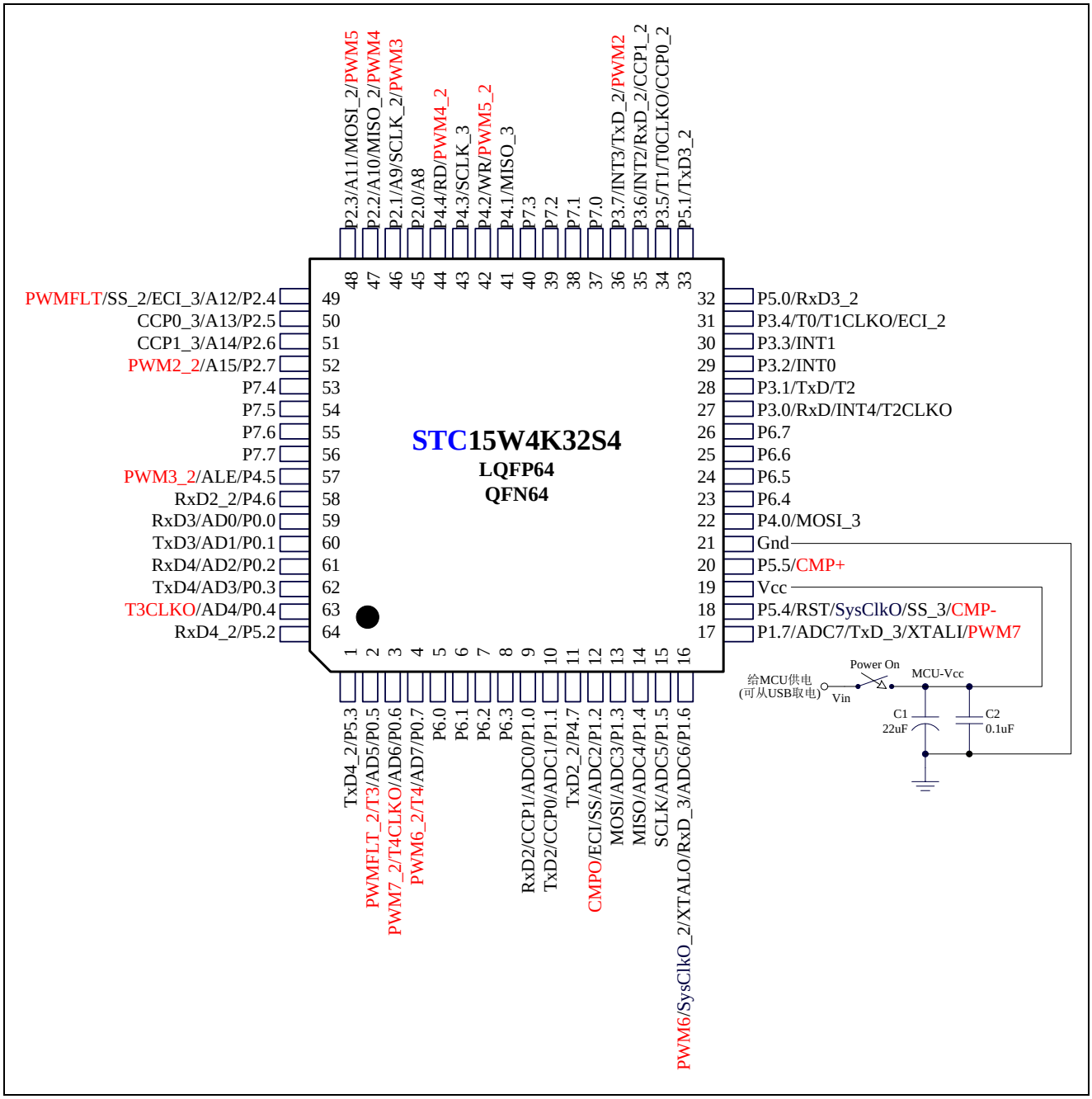
3、与 PWM2 到 PWM7 相关的 12 个口[P3.7/PWM2，P2.1/PWM3，P2.2/PWM4，P2.3/PWM5，P1.6/PWM6，P1.7/PWM7， P2.7/PWM2_2， P4.5/PWM3_2， P4.4/PWM4_2， P4.2/PWM5_2， P0.7/PWM6_2，P0.6/PWM7_2]，上电复位后是高阻输入（既不向外输出电流也不向内输出电流），若要使其能对外能输出，要用软件将其改设为强推挽输出或准双向口上拉

4、掉电模式时漏电流<0.4uA

5、如将串口 2 切换到[P4.7/TxD，P4.6/RxD]时，P4.7 要加 3.3K 上拉电阻，且须工作在弱上拉/准双向口模式

6、比较器不支持单独的下降沿中断，可将下降沿/上升沿中断均打开，进中断后在比较器中断服务程序中查询比较器比较结果标志位 CMPRES（CMPCR1.0）的值来判断单片机进入的是比较器上升沿中断还是比较器下降沿中断，如果 CMPRES/CMPCR1.0=1，即 CMP+的电平高于 CMP-的电平（或内部 BandGap 参考电压的电平），则表示单片机进入的是比较器上升沿中断；反之，如果 CMPRES（CMPCR1.0）=0，即 CMP+的电平低于 CMP-的电平（或内部 BandGap 参考电压的电平），则表示单片机进入的是比较器下降沿中断，此时比较器下降沿中断是可正常使用的

7、如果用户要将单片机设置成使用内部时钟，则最好不要外接外部晶振；但是如果用户既想将单片机设置成使用内部时钟，又想外挂外部晶振，则上电复位的额外延时<180ms>不能设



12.6 STC15W4K32S4 系列 B 版单片机的特别注意事项

1、与 PWM2 到 PWM7 相关的 12 个口[P3.7/PWM2, P2.1/PWM3, P2.2/PWM4, P2.3/PWM5, P1.6/PWM6, P1.7/PWM7, P2.7/PWM2_2, P4.5/PWM3_2, P4.4/PWM4_2, P4.2/PWM5_2, P0.7/PWM6_2, P0.6/PWM7_2]，上电复位后是高阻输入（既不向外输出电流也不向内输出电流），若要使其能对外能输出，要用软件将其改设为强推挽输出或准双向口/上拉；这些端口进入掉电模式时不能为高阻输入，否则需外部加上拉电阻到 Vcc 或下拉电阻到地。

2、8 路 ADC 口不可作比较器正极（CMP+），但 STC15W408AS 系列的 8 路 ADC 口可以用作比较器正极（CMP+）

13 STC15 系列的时钟、复位及省电模式

13.1 STC15 系列单片机的时钟

STC15F2K60S2 系列、STC15W4K32S4 系列、STC15W408AS 系列和 STC15F408AD 系列单片机有两个时钟源：内部高精度 R/C 时钟和外部时钟（外部输入的时钟或外部晶体振荡产生的时钟）。而 STC15F104W 系列、STC15W204S 系列、STC15W404S 系列和 STC15W1K16S 系列无外部时钟只有内部高精度 R/C 时钟。内部高精度 R/C 时钟（±0.3%），±1%温漂（-40℃～+85℃），常温下温漂±0.6%（-20℃～+65℃）

STC15 系列单片机的的时钟源见下表所示。

时钟源类型 单片机型号	内部高精度 R/C 时钟(±0.3%)， ±1%温漂(-40℃～+85℃)， 常温下温漂±0.6%(-20℃～+65℃)	外部时钟 (外部输入的时钟 或外部晶体振荡产生的时钟)
STC15F104W 系列	✓	
STC15F408AD 系列	✓	✓
STC15W204S 系列	✓	
STC15W408AS 系列	✓	✓
STC15W404S 系列	✓	
STC15W1K16S 系列	✓	
STC15F2K60S2 系列	✓	✓
STC15W4K32S4 系列	✓	✓

上表中✓表示对应的系列有相应的时钟源。

13.1.1 STC15 系列单片机的内部可配置时钟



13.1.2 主时钟分频和分频寄存器

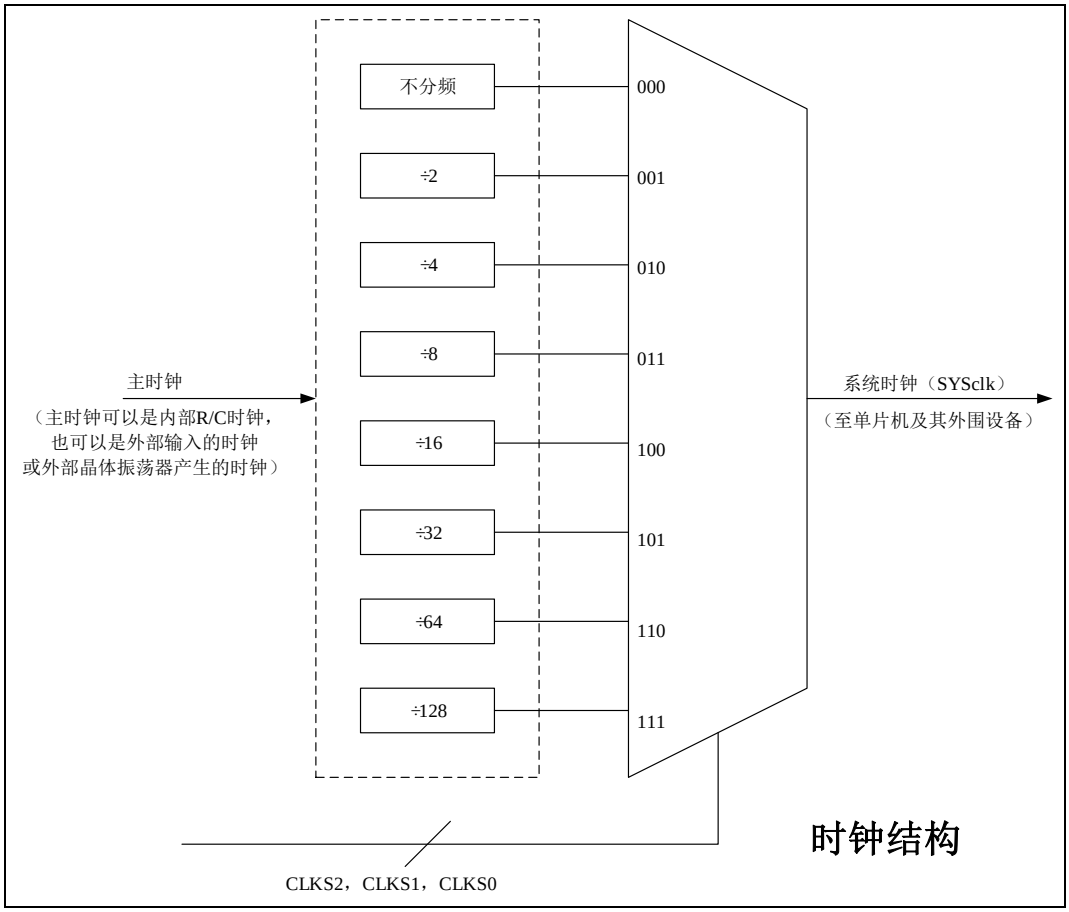
如果希望降低系统功耗，可对时钟进行分频。利用时钟分频控制寄存器 CLK_DIV(PCON2)可进行时钟分频，从而使单片机在较低频率下工作。

时钟分频寄存器 CLK_DIV(PCON2)各位的定义如下:

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CLK_DIV (PCON2)	97H	name	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0

CLKS2	CLKS1	CLKS0	系统时钟选择控制位 (系统时钟是指对主时钟进行分频后供给 CPU、串行口、SPI、定时器、CCP/PWM/PCA、A/D 转换的实际工作时钟)
0	0	0	主时钟频率/1，不分频
0	0	1	主时钟频率/2
0	1	0	主时钟频率/4
0	1	1	主时钟频率/8
1	0	0	主时钟频率/16
1	0	1	主时钟频率/32
1	1	0	主时钟频率/64
1	1	1	主时钟频率/128

主时钟可以是内部 R/C 时钟，也可以是外部输入的时钟或外部晶体振荡产生的时钟。



时钟分频寄存器 CLK_DIV (PCON2)其他各位的说明如下：

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CLK_DIV (PCON2)	97H	name	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0

MCKO_S1	MCKO_S0	主时钟对外分频输出控制位 (主时钟既可是内部 R/C 时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟)
0	0	主时钟不对外输出时钟
0	1	主时钟对外输出时钟，但时钟频率不被分频，输出时钟频率 = MCLK / 1
1	0	主时钟对外输出时钟，但时钟频率被 2 分频，输出时钟频率 = MCLK / 2
1	1	主时钟对外输出时钟，但时钟频率被 4 分频，输出时钟频率 = MCLK / 4

主时钟既可以是内部 R/C 时钟，也可以是外部输入的时钟或外部晶体振荡产生的时钟。但对于无外部时钟源的单片机（STC15F104W 系列、STC15W204S 系列、STC15W404S 系列、STC15W1K16S 系列）以及现供货的 STC15F2K60S2 系列 C 版单片机，其主时钟只能是内部 R/C 时钟。

主时钟可在管脚 MCLKO 或 MCLKO_2 对外输出。其中，STC15 系列 8-pin 单片机（如 STC15F104W 系列）在 MCLKO/P3.4 口对外输出时钟；STC15F2K60S2 系列、STC15W204S 系列及 STC15F408AD 系列单片机在 MCLKO/P5.4 口对外输出时钟；而 STC15W404S 系列及 STC15W1K16S 系列单片机除可在 MCLKO/P5.4 口对外输出时钟外，还可在 MCLKO_2/P1.6 口对外输出时钟。

【特意注意】: STC15W4K32S4 系列、STC15W1K08PWM 系列及 STC15W1K20S-LQFP64 单片机是将系统时钟在管脚 SysClkO/P5.4 或 SysClkO_2/XTAL2/P1.6 对外分频输出，并可如下分频 SysClk/1，SysClk/2，SysClk/4，SysClk/16。
STC15W408AS 系列单片机也是将系统时钟在管脚 SysClkO/P5.4 或 SysClkO_2/ XTAL2/ P1.6 对外分频输出，但只可如下分频 SysClk/1，SysClk/2，SysClk/4。

系统时钟是指对主时钟进行分频后供给 CPU、定时器、串行口、SPI、CCP/PWM/PCA、A/D 转换的实际工作时钟。STC15W4K32S4 系列、STC15W408AS 系列、STC15W1K08PWM 系列及 STC15W1K20S-LQFP64 单片机的主时钟既可以是内部 R/C 时钟，也可以是外部输入的时钟或外部晶体振荡产生的时钟。
MCLK 是指主时钟频率，MCLKO 是指系统时钟输出。SysClk 是指系统时钟频率，SysClkO 是指系统时钟输出。

STC15W404S 系列及 STC15W1K16S 系列单片机通过 CLK_DIV3/MCLKO_2 位来选择是在 MCLKO/P5.4 口对外输出主时钟，还是在 MCLKO_2/P1.6 口对外输出主时钟。

MCLKO_2：主时钟对外输出位置的选择位
0：在 MCLKO/P5.4 口对外输出主时钟
1：在 MCLKO_2/P1.6 口对外输出主时钟
STC15W404S 系列及 STC15W1K16S 系列单片机的主时钟只能是内部 R/C 时钟。

STC15W4K32S4 系列、STC15W408AS 系列、STC15W1K08PWM 系列及 STC15W1K20S-LQFP64 单片机通过 CLK_DIV.3/SysClkO_2 位来选择是在 SysClkO/P5.4 口对外输出系统时钟，还是在 SysClkO_2/P1.6 口对外输出系统时钟。

SysClkO_2：系统时钟对外输出位置的选择位
0：在 SysClkO/P5.4 口对外输出系统时钟；
1：在 SysClkO_2/P1.6 口对外输出系统时钟

13.1.3 可编程时钟输出（也可作分频器使用）

STC15 系列单片机最多有六路可编程时钟输出（如 STC15W4K32S4 系列），如下表所示。对于 STC15 系列 5V 单片机，由于 I/O 口的对外输出速度最快不超过 13.5MHz，所以对外可编程时钟输出速度最快也不超过 13.5MHz；对于 3.3V 单片机，由于 I/O 口的对外输出速度最快不超过 8MHz，所以对外可编程时钟输出速度最快也不超过 8MHz。

STC15 全系列的可编程时钟输出的类型如下表所示。

可编程时钟输出 / 单片机型号	主时钟输出 (MCLKO/P5.4)	系统时钟输出 (SysClkO/P5.4 或 SysClkO_2/P1.6)	定时器/计数器 0 时钟输出 (T0CLKO/P3.5)	定时器/计数器 1 时钟输出 (T1CLKO/P3.4)	定时器/计数器 2 时钟输出 (T2CLKO/P3.0)	定时器/计数器 3 时钟输出 (T3CLKO/P0.4)	定时器/计数器 4 时钟输出 (T4CLKO/P0.6)
STC15F104W 系列	该系列主时钟输出在 MCLKO/P3.4		√		√		
STC15F408AD 系列	√		√		√		
STC15W204S 系列	√		√		√		
STC15W408AS 系列		√	√		√		
STC15W404S 系列	√ (该系列主时钟输出还可在 MCLKO_2/P1.6)		√	√	√		
STC15W1K16S 系列	√ (该系列主时钟输出还可在 MCLKO_2/ XTAL2/P1.6)		√	√	√		
STC15F2K60S2 系列	√		√	√	√		
STC15W4K32S4 系列		√	√	√	√	√	√
STC15W1K08PWM 系列		√	√	√	√		
STC15W1K20S-LQFP64		√	√	√	√		

上表中√表示对应的系列有相应的可编程时钟输出。

【特别注意】：对于 STC15W1K16S 系列和 STC15W408S 单片机，若要使用 T0CLKO 时钟输出功能，必须将 P3.5 口设置为强推挽输出模式。

13.1.3.1 与可编程时钟输出有关的特殊功能寄存器

符号	描述	地址	位地址及其符号 MSB							LSB	复位值
AUXR	辅助寄存器	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000 0001B
INT_CLKO AUXR2	External Interrupt enable and Clock output register	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000 x000B
CLK_DIV (PCON2)	时钟分配器	97H	MCKO_S1	MCKO_S1	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000 0000B
T4T3M	T4 和 T3 的控制寄存器	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO	0000 0000B

特殊功能寄存器 INT_CLKO/AUXR/CLK_DIV/T4T3M 的 C 语言声明：

```
sfr     INT_CLKO    = 0x8F;     //新增加的特殊功能寄存器 INTCLKO 的地址声明
sfr     AUXR        = 0x8E;     //特殊功能寄存器 AUXR 的地址声明
sfr     CLK_DIV     = 0x97;     //特殊功能寄存器 CLK DIV 的地址声明
sfr     T4T3M       = 0xD1;     //新增加的特殊功能寄存器 T4T3M 的地址声明
```

特殊功能寄存器 INT CLKO/AUXR/CLK DIV/T4T3M 的汇编语言声明：

```
INT CLKO    EOU     8FH     ;新增加的特殊功能寄存器 INTCLKO 的地址声明
AUXR        EQU     8EH     ;特殊功能寄存器 AUXR 的地址声明
CLK_DIV     EQU     97H     ;特殊功能寄存器 CLK_DIV 的地址声明
T4T3M       EQU     D1H     ;新增加的特殊功能寄存器 T4T3M 的地址声明
```

1.CLK_DIV(PCON2)：时钟分频寄存器(不可位寻址)

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CLK_DIV (PCON2)	97H	name	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0

MCKO_S1	MCKO_S0	主时钟对外分频输出控制位 (主时钟对外输出管脚 MCLKO 或 MCLKO_2 既可对外输出内部 R/C 时钟，也可对外输出外部输入的时钟或外部晶体振荡产生的时钟)
0	0	主时钟不对外输出时钟
0	1	主时钟对外输出时钟，但时钟频率不被分频，输出时钟频率 = MCLK / 1
1	0	主时钟对外输出时钟，但时钟频率被 2 分频，输出时钟频率 = MCLK / 2
1	1	主时钟对外输出时钟，但时钟频率被 4 分频，输出时钟频率 = MCLK / 4

主时钟既可以是内部 R/C 时钟，也可以是外部输入的时钟或外部晶体振荡产生的时钟。但对于无外部时钟源的单片机（STC15F104W 系列、STC15W204S 系列、STC15W404S 系列、STC15W1K16S 系列）以及现供货的 STC15F2K60S2 系列 C 版单片机，其主时钟只能是内部 R/C 时钟。

主时钟可在管脚 MCLKO 或 MCLKO_2 对外输出。其中，STC15 系列 8-pin 单片机（如 STC15F104W 系列）在 MCLKO/P3.4 口对外输出时钟；STC15F2K60S2 系列、STC15W204S 系列及 STC15F408AD 系列单片机在 MCLKO/P5.4 口对外输出时钟；而 STC15W404S 系列及 STC15W1K16S 系列单片机除可在 MCLKO/P5.4 口对外输出时钟外，还可在 MCLKO_2/P1.6 口对外输出时钟。

【特意注意】：STC15W4K32S4 系列、STC15W1K08PWM 系列及 STC15W1K20S-LQFP64 单片机是将系统时钟在管脚 SysClkO/P5.4 或 SysClkO_2/XTAL2/P1.6 对外分频输出，并可如下分频 SysClk/l，SysClk/2，SysClk/4，SysClk/16。
STC15W408AS 系列单片机也是将系统时钟在管脚 SysClkO/P5.4 或 SysClkO_2/XTAL2/P1.6 对外分频输出，但只可如下分频 SysClk/1，SysClk/2，SysClk/4。

系统时钟是指对主时钟进行分频后供给 CPU、定时器、串行口、SPI、CCP/PWM/PCA、A/D 转换的实际工作时钟。STC15W4K32S4 系列、STC15W408AS 系列、STC15W1K08PWM 系列及 STC15W1K20S-LQFP64 单片机的主时钟既可以是内部 R/C 时钟，也可以是外部输入的时钟或外部晶体振荡产生的时钟。

MCLK 是指主时钟频率，MCLKO 是指系统时钟输出。SysClk 是指系统时钟频率，SysClkO 是指系统时钟输出。

STC15W404S 系列及 STC15W1K16S 系列单片机通过 CLK_DIV.3/MCLKO_2 位来选择是在 MCLKO/P5.4 口对外输出主时钟，还是在 MCLKO_2/P1.6 口对外输出主时钟。

MCLKO_2：主时钟对外输出位置的选择位

0：在 MCLKO/P5.4 日对外输出主时钟；

1：在 MCLKO_2/P1.6 口对外输出主时钟

STC15W404S 系列及 STC15W1K16S 系列单片机的主时钟只能是内部 R/C 时钟。

STC15W4K32S4 系列、STC15W408AS 系列、STC15W1K08PWM 系列及 STC15W1K20S-LQFP64 单片机通过 CLK_DIV.3/SysClkO_2 位来选择是在 SysClkO/P5.4 口对外输出系统时钟，还是在 SysClkO_2/P1.6 口对外输出系统时钟。

SysClkO_2：系统时钟对外输出位置的选择位

0：在 SysClkO/P5.4 口对外输出系统时钟；

1：在 SysClkO_2/P1.6 口对外输出系统时钟

系统时钟是指对主时钟进行分频后供给 CPU、定时器、串行口、SPI、CCP/PWM/PCA、A/D 转换的实际工作时钟。STC15W4K32S4 系列、STC15W408AS 系列、STC15W1K08PWM 系列及 STC15W1K20S-LQFP64 单片机的主时钟既可以是内部 R/C 时钟，也可以是外部输入的时钟或外部晶体振荡产生的时钟。

CLKS2	CLKS1	CLKS0	系统时钟选择控制位 (系统时钟是指对主时钟进行分频后供给 CPU、串行口、SPI、定时器、CCP/PWM/PCA、A/D 转换的实际工作时钟)
0	0	0	主时钟频率/1,不分频
0	0	1	主时钟频率/2
0	1	0	主时钟频率/4
0	1	1	主时钟频率/8
1	0	0	主时钟频率/16
1	0	1	主时钟频率/32
1	1	0	主时钟频率/64
1	1	1	主时钟频率/128

主时钟既可是内部 R/C 时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。

2. INT_CLKO（AUXR2）：External Interrupt Enable and Clock Output register

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
INT_CLKO AUXR2	8FH	name	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO

B0 - T0CLKO ：是否允许将 P3.5/T1 脚配置为定时器 0（T0）的时钟输出 T0CLKO

1，将 P3.5/T1 管脚配置为定时器 0 的时钟输出 T0CLKO，输出时钟频率=T0 溢出率/2

若定时器/计数器 T0 工作在定时器模式 0（16 位自动重装载模式）时，

- 如果 C/T=0，定时器/计数器 T0 是对内部系统时钟计数，则：
  - ✓ T0 工作在 1T 模式（AUXR.7/T0x12=1）时的输出频率=（SYSclk）/（65536-[RL_TH0，RL_TL0]）/2
  - ✓ T0 工作在 12T 模式（AUXR.7/T0x12=0）时的输出频率=（SYSclk）/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

若定时器/计数器 T0 工作在定时器模式 2（8 位自动重装模式），

- 如果 C/T=0，定时器/计数器 T0 是对内部系统时钟计数，则：
  - ✓ T0 工作在 1T 模式（AUXR.7/T0x12=1）时的输出频率=（SYSclk）/（256-TH0）/2
  - ✓ T0 工作在 12T 模式（AUXR.7/T0x12=0）时的输出频率=（SYSclk）/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

0，不允许 P3.5/T1 管脚被配置为定时器 0 的时钟输出

B1 - T1CLKO：是否允许将 P3.4/T0 脚配置为定时器 1（T1）的时钟输出 T1CLKO

1，将 P3.4/T0 管脚配置为定时器 1 的时钟输出 T1CLKO，输出时钟频率=T1 溢出率/2

若定时器/计数器 T1 工作在定时器模式 0（16 位自动重装载模式），

- 如果 C/T=0，定时器/计数器 T1 是对内部系统时钟计数，则：
  - ✓ T1 工作在 1T 模式（AUXR.6/T1x12=1）时的输出频率=（SYSclk）/（65536-[RL_TH1，RL_TL1]）/2
  - ✓ T1 工作在 12T 模式（AUXR.6/T1x12=0）时的输出频率=（SYSclk）/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

若定时器/计数器 T1 工作在模式 2（8 位自动重装模式），

- 如果 C/T=0，定时器/计数器 T1 是对内部系统时钟计数，则：
  - ✓ T1 工作在 1T 模式（AUXR.6/T1x12=1）时的输出频率=（SYSclk）/（256-TH1）/2
  - ✓ T1 工作在 12T 模式（AUXR.6/T1x12=0）时的输出频率=（SYSclk）/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

0，不允许 P3.4/T0 管脚被配置为定时器 1 的时钟输出

B2 - T2CLKO：是否允许将 P3.0 脚配置为定时器 2（T2）的时钟输出 T2CLKO

1，允许将 P3.0 脚配置为定时器 2 的时钟输出 T2CLKO，输出时钟频率=T2 溢出率/2

- 如果 T2_C/T=0，定时器/计数器 T2 是对内部系统时钟计数，则：
  - ✓ T2 工作在 1T 模式（AUXR.2/T2x12=1）时的输出频率=（SYSclk）/（65536-[RL_TH2，RL_TL2]）/2
  - ✓ T2 工作在 12T 模式（AUXR.2/T2x12=0）时的输出频率=（SYSclk）/12/（65536-[RL_TH2，RL_TL2]）/2
- 如果 T2_C/T=1，定时器/计数器 T2 是对外部脉冲输入（P3.1/T2）计数，则：
  - ✓ 输出时钟频率=（T2_Pin_CLK）/（65536-[RL_TH2，RL_TL2]）/2

0，不允许将 P3.0 脚配置为定时器 2（T2）的时钟输出 T2CLKO

B4-EX2：允许外部中断 2（INT2）

B5-EX3：允许外部中断 3（INT3）

B6-EX4：允许外部中断 4（INT4）

3、辅助特殊功能寄存器：AUXR（地址：0x8E）

AUXR：Auxiliary register（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	name	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

B7-T0x12：定时器 0 速度控制位

0：定时器 0 速度是传统 8051 单片机定时器的速度，即 12 分频；

1：定时器 0 速度是传统 8051 单片机定时器速度的 12 倍，即不分频。

B6-T1x12：定时器 1 速度控制位

0：定时器 1 速度是传统 8051 单片机定时器的速度，即 12 分频；

1：定时器 1 速度是传统 8051 单片机定时器速度的 12 倍，即不分频。

如果串口 1 用 T1 作为波特率发生器，则由 T1x12 位决定串口 1 是 12T 还是 1T。

B5-UART_M0x6：串口 1 模式 0 的通信速度设置位

0：串口 1 模式 0 的速度是传统 8051 单片机串口的速度，即 12 分频；

1：串口 1 模式 0 的速度是传统 8051 单片机串口速度的 6 倍，即 2 分频。

B4 - T2R：定时器 2 运行控制位

0：不允许定时器 2 运行；

1：允许定时器 2 运行。

B3-T2_C/T：控制定时器 2 用作定时器或计数器

0：用作定时器（对内部系统时钟进行计数）；

1：用作计数器（对引脚 T2/P3.1 的外部脉冲进行计数）

B2-T2x12：定时器 2 速度控制位

0：定时器 2 是传统 8051 单片机的速度，12 分频；

1：定时器 2 的速度是传统 8051 单片机速度的 12 倍，不分频

如果串口 1 或串口 2 用 T2 作为波特率发生器，则由 T2x12 决定串口 1 或串口 2 是 12T 还是 IT。

B1-EXTRAM：内部/外部 RAM 存取控制位

0：允许使用逻辑上在片外、物理上在片内的扩展 RAM；

1：禁止使用逻辑上在片外、物理上在片内的扩展 RAM。

B0-S1ST2：串口 1（UART1）选择定时器 2 作波特率发生器的控制位

0：选择定时器 1 作为串口 1（UART1）的波特率发生器；

1：选择定时器 2 作为串口 1（UART1）的波特率发生器，此时定时器 1 得到释放，可以作为独立定时器使用。

4、定时器 T4 和 T3 的控制寄存器：T4T3M（地址：0xD1）

T4T3M（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	name	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO

B7-T4R：定时器 4 运行控制位

0：不允许定时器 4 运行；

1：允许定时器 4 运行。

B6-T4_C/T：控制定时器 4 用作定时器或计数器

0：用作定时器（对内部系统时钟进行计数）；

1：用作计数器（对引脚 T4/P0.7 的外部脉冲进行计数）

B5-T4x12：定时器 4 速度控制位

0：定时器 4 速度是 8051 单片机定时器的速度，即 12 分频；

1：定时器 4 速度是 8051 单片机定时器速度的 12 倍，即不分频。

B4-T4CLKO：是否允许将 P0.6 脚配置为定时器 4（T4）的时钟输出 T4CLKO

1：允许将 P0.6 脚配置为定时器 4 的时钟输出 T4CLKO，输出时钟频率=T4 溢出率/2

- 如果 T4_C/T=0，定时器/计数器 T4 是对内部系统时钟计数，则：
  - ✓ T4 工作在 1T 模式（T4T3M.5/T4x12=1）时的输出频率=（SYSclk）/（65536-[RL_TH4，RL_TL4]）/2
  - ✓ T4 工作在 12T 模式（T4T3M.5/T4x12=0）时的输出频率=（SYSclk）/12/（65536-[RL_TH4，RL_TL4]）/2
- 如果 T4_C/T=1，定时器/计数器 T4 是对外部脉冲输入（P0.7/T4）计数，则：
  - ✓ 输出时钟频率=（T4_Pin_CLK）/（65536-[RL_TH4，RL_TL4]）/2

0：不允许将 P0.6 脚配置为定时器 4（T4）的时钟输出 T4CLKO

B3-T3R：定时器 3 运行控制位

0：不允许定时器 3 运行；

1：允许定时器 3 运行。

B2-T3_C/T：控制定时器 3 用作定时器或计数器

0：用作定时器（对内部系统时钟进行计数）；

1：用作计数器（对引脚 T3/P0.5 的外部脉冲进行计数）

B1-T3x12：定时器 3 速度控制位；

0：定时器 3 速度是 8051 单片机定时器的速度，即 12 分频；

1：定时器 3 速度是 8051 单片机定时器速度的 12 倍，即不分频。

B0-T3CLKO：是否允许将 P0.4 脚配置为定时器 3（T3）的时钟输出 T3CLKO

1：允许将 P0.4 脚配置为定时器 3 的时钟输出 T3CLKO，输出时钟频率=T3 溢出率/2

- 如果 T3_C/T=0，定时器/计数器 T3 是对内部系统时钟计数，则：
  - ✓ T3 工作在 1T 模式（T4T3M.1/T3x12=1）时的输出频率=（SYSclk）/（65536-[RL_TH3，RL_TL3]）/2
  - ✓ T3 工作在 12T 模式（T4T3M.1/T3x12=0）时的输出频率=（SYSclk）/12/（65536-[RL_TH3，RL_TL3]）/2
- 如果 T3_C/T=1，定时器/计数器 T3 是对外部脉冲输入（P0.5/T3）计数，则：
  - ✓ 输出时钟频率=（T3_Pin_CLK）/（65536-[RL_TH3，RL_TL3]）/2

0：不允许将 P0.4 脚配置为定时器 3（T3）的时钟输出 T3CLKO

13.1.3.2 主时钟输出及测试程序（C 和汇编）

主时钟可以是内部高精度 R/C 时钟，也可以是外部输入的时钟或外部晶体振荡产生的时钟。由于 STC15 系列 5V 单片机 I/O 口的对外输出速度最快不超过 13.5MHz，所以 5V 单片机的对外可编程时钟输出速度最快也不超过 13.5MHz，如果频率过高，需进行分频输出；而 3.3V 单片机 I/O 口的对外输出速度最快不超过 8MHz，故 3.3V 单片机的对外可编程时钟输出速度最快也不超过 8MHz，如果频率过高，需进行分频输出。

主时钟对外输出控制寄存器：CLK_DIV（不可位寻址）与 INT_CLKO（不可位寻址）

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000 0000

如何利用 MCLKO/P5.4 或 MCLKO_2/XTAL2/P1.6 管脚输出时钟

MCLKO/P5.4 或 MCLKO_2/XTAL2/P1.6 的时钟输出控制由 CLK_DIV 寄存器的 MCKO_S1 和 MCKO_S0 位控制。通过设置 MCKO_S1(CLK_DIV.7)和 MCKO_S0(CLK_DIV.6)可将 MCLKO/P5.4 管脚配置为主时钟输出同时还可以设置该主时钟的输出频率。

特殊功能寄存器：CLK_DIV(地址：97H)

MCKO_S1	MCKO_S0	主时钟对外分频输出控制位 (主时钟对外输出管脚 MCLKO 或 MCLKO_2 既可对外输出内部 R/C 时钟，也可对外输出外部输入的时钟或外部晶体振荡产生的时钟)
0	0	主时钟不对外输出时钟
0	1	主时钟对外输出时钟，但时钟频率不被分频，输出时钟频率 = MCLK / 1
1	0	主时钟对外输出时钟，但时钟频率被 2 分频，输出时钟频率 = MCLK / 2
1	1	主时钟对外输出时钟，但时钟频率被 4 分频，输出时钟频率 = MCLK / 4

主时钟既可以是内部 R/C 时钟，也可以是外部输入的时钟或外部晶体振荡产生的时钟。但对于无外部时钟源的单片机（STC15F104W 系列、STC15W204S 系列、STC15W404S 系列、STC15W1K16S 系列）以及现供货的 STC15F2K60S2 系列 C 版单片机，其主时钟只能是内部 R/C 时钟。

主时钟可在管脚 MCLKO 或 MCLKO_2 对外输出。其中，STC15 系列 8-pin 单片机（如 STC15F104W 系列）在 MCLKO/P3.4 口对外输出时钟；STC15F2K60S2 系列、STC15W204S 系列及 STC15F408AD 系列单片机在 MCLKO/P5.4 口对外输出时钟；而 STC15W404S 系列及 STC15W1K16S 系列单片机除可在 MCLKO/P5.4 口对外输出时钟外，还可在 MCLKO_2/P1.6 口对外输出时钟。

【特意注意】: STC15W4K32S4 系列、STC15W1K08PWM 系列及 STC15W1K20S-LQFP64 单片机是将系统时钟在管脚 SysClkO/P5.4 或 SysClkO_2/XTAL2/P1.6 对外分频输出，并可如下分频 SysClk/1，SysClk/2，SysClk/4，SysClk/16。
STC15W408AS 系列单片机也是将系统时钟在管脚 SysClkO/P5.4 或 SysClkO_2/XTAL2/P1.6 对外分频输出，但只可如下分频 SysClk/1，SysClk/2，SysClk/4。

系统时钟是指对主时钟进行分频后供给 CPU、定时器、串行口、SPI、CCP/PWMIPCA、AD 转换的实际工作时钟。STC15W4K32S4 系列、STC15W408AS 系列、STC15W1K08PWM 系列及 STC15W1K20S-LQFP64 单片机的主时钟既可以是内部 R/C 时钟，也可以是外部输入的时钟或外部晶体振荡产生的时钟。

MCLK 是指主时钟频率，MCLKO 是指系统时钟输出。SysClk 是指系统时钟频率，SysClkO 是指系统时钟输出。

STC15W404S 系列及 STC15W1K16S 系列单片机通过 CLK_DIV.3/MCLKO_2 位来选择是在 MCLKO/P5.4 口对外输出主时钟，还是在 MCLKO_2/P1.6 口对外输出主时钟。

MCLKO_2：主时钟对外输出位置的选择位
0：在 MCLKO/P5.4 口对外输出主时钟；
1：在 MCLKO 2/P1.6 日对外输出主时钟

STC15W404S 系列及 STC15W1K16S 系列单片机的主时钟只能是内部 R/C 时钟。

STC15W4K32S4 系列、STC15W408AS 系列、STC15W1K08PWM 系列及 STC15W1K20S-LQFP64 单片机通过 CLK_DIV.3/SysClkO_2 位来选择是在 SysClkO/P5.4 口对外输出系统时钟，还是在 SysClkO_2/P1.6 口对外输出系统时钟。

SysClkO_2：系统时钟对外输出位置的选择位
0：在 SysClkO/P5.4 口对外输出系统时钟；
1：在 SysClkO_2/P1.6 口对外输出系统时钟

系统时钟是指对主时钟进行分频后供给 CPU、定时器、串行口、SPI、CCP/PWM/PCA、A/D 转换的实际工作时钟。STC15W4K32S4 系列、STC15W408AS 系列、STC15W1K08PWM 系列及 STC15W1K20S-

LQFP64 单片机的主时钟既可以是内部 R/C 时钟，也可以是外部输入的时钟或外部晶体振荡产生的时钟。

由于 STC15 系列 5V 单片机 I/O 口的对外输出速度最快不超过 13.5MHz，所以 5V 单片机的对外可编程时钟输出速度最快也不超过 13.5MHz，如果频率过高，需进行分频输出。

而 3.3V 单片机 I/O 口的对外输出速度最快不超过 8MHz，故 3.3V 单片机的对外可编程时钟输出速度最快也不超过 8MHz，如果频率过高，需进行分频输出。

下面是主时钟输出的示例程序:

1.C 程序:

```
/*----演示 STC15F2K60S2 系列单片机的主时钟输出-----------------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
//假定测试芯片的工作频率为 18.432MHz

#include "reg51.h"
type        defunsigned char   BYTE;
type        defunsigned int    WORD;
#define     FOSC               18432000L
sfr         CLK_DIV            0x97;          //时钟分频寄存器
void main()
{
    CLK_DIV=0x40;                             //0100,0000 P5.4 输出频率为 SYSclk
//  CLK_DIV=0x80;                             //1000,0000 P5.4 输出频率为 SYSclk/2
//  CLK_DIV=0xC0;                             //1100,0000 P5.4 输出频率为 SYSclk/4

    while (1);                                //程序终止
```

2.汇编程序：

```
/*-------------------------------------------------------------------------------------------*/
/* ---  演示 STC15F2K60S2 系列单片机的主时钟输出  -----------------------------------*/
/*----  在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz

CLK_DIV     DATA    097H        ;IRC 时钟输出控制寄存器
;---------------------------------------
;interrupt vector table
    ORG     0000H
    LJMP    START               ;复位入口

;---------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH            ;initial SP
```

```
        MOV     CLK_DIV, #40H           ;0100,0000 P5.4 输出频率为 SYSclk
;       MOV     CLK_DIV, #80H           ;1000,0000 P5.4 输出频率为 SYSclk/2
;       MOV     CLK_DIV, #C0H           ;1100,0000 P5.4 输出频率为 SYSclk/4
        SJMP    $


;---------------------------------------
END
```

13.1.3.3 定时器 0 对系统时钟或外部引脚 T0 的时钟输入进行可编程分频输出及测试程序

如何利用 T0CLKO/P3.5 管脚输出时钟

T0CLKO/P3.5 管脚是否输出时钟由 INT_CLKO(AUXR2)寄存器的 T0CLKO 位控制

AUXR2.0-T0CLKO：1，允许时钟输出
0，禁止时钟输出

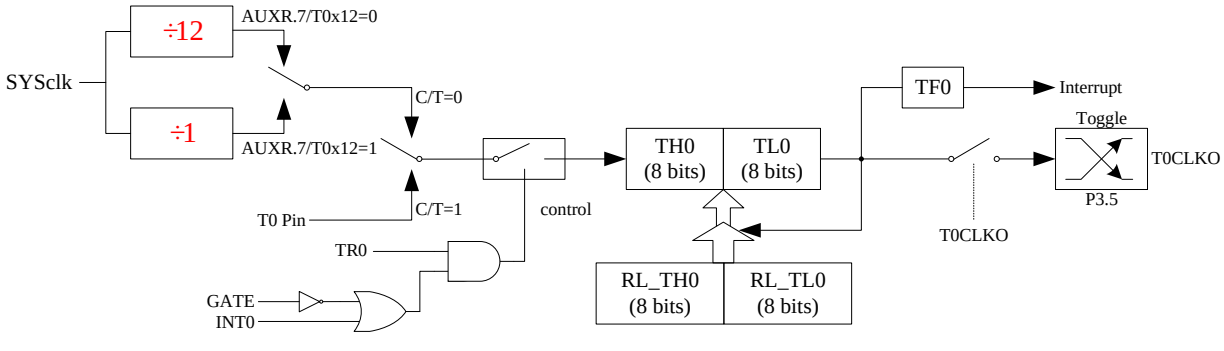
T0CLKO 的输出时钟频率由定时器 0 控制，相应的定时器 0 需要工作在定时器的模式 0(16 位自动重装载模式)或模式 2(8 位自动重装载模式)，不要允许相应的定时器中断，免得 CPU 反复进中断，当然在特殊情况下也可允许相应的定时器中断。

新增加的特殊功能寄存器：INT_CLKO(AUXR2)(地址：0x8F)

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=T0 溢出率/2
若定时器/计数器 T0 工作在定时器模式 0(16 位自动重装载模式)时，(如下图所示)

- 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)/(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)/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

RL_TH0 为 TH0 的重装载寄存器，RL_TL0 为 TL0 的重装载寄存器。



定时器/计数器0的模式0：16位自动重装

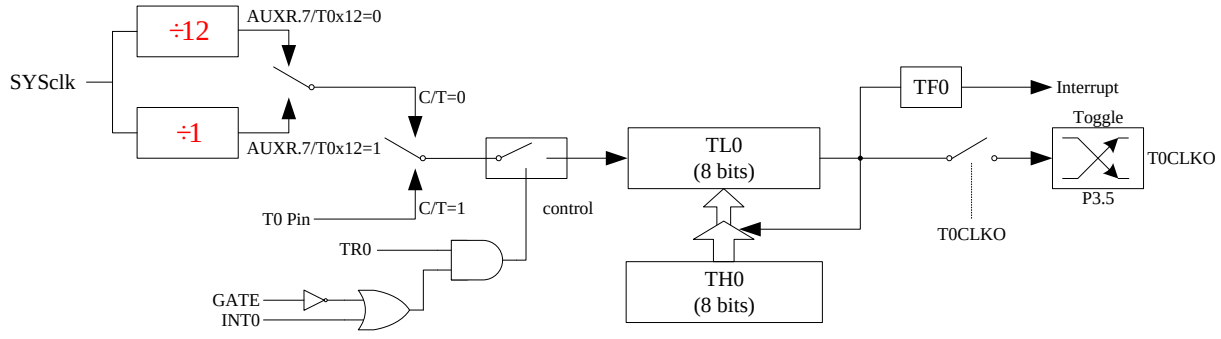
当 T0CLKO/INT_CLKO.0=1 且定时器/计数器 T0 工作在定时器模式 2(8 位自动重装模式)时，(如下图所示)

- 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)/(256-TH0)/2
- ✓ T0 工作在 12T 模式(AUXR.7/T0x12=0)时的输出时钟频率=(SYSclk)/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



定时器/计数器0的模式2：8位自动重装

【特别注意】：对于 STC15W1K16S 系列和 STC15W408S 单片机，若要使用 T0CLKO 时钟输出功能，必须将 P3.5 口设置为强推挽输出模式。

下面是定时器 0 对内部系统时钟或外部引脚 T0/P3.4 的时钟输入进行可编程时钟分频输出的程序举例(C 和汇编)：

1.C 程序:

```
/*------演示 STC15F2K60S2 系列单片机定时器 0 的可编程时钟分频输出----------*/
/*------在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可---------*/
//假定测试芯片的工作频率为 18.432MHz

#include "reg51.h"
typedef     unsigned char     BYTE;
typedef     unsigned int      WORD;
#define     FOSC              18432000L
//-----------------------------------------------
sfr         AUXR = 0x8e;                          //辅助特殊功能寄存器
sfr         INT_CLKO = 0x8f;                      //唤醒和时钟输出功能寄存器
sbit        T0CLKO = P3^5;                        //定时器 0 的时钟输出脚
#define     F38_4KHz    (65536-FOSC/2/38400)      //1T 模式
//#define     F38_4KHz    (65536-FOSC/2/12/38400)   //12T 模式
//-----------------------------------------------
void main()
{
    AUXR |= 0x80;               //定时器 0 为 1T 模式
//  AUXR &= ~0x80;              //定时器 0 为 12T 模式
    TMOD = 0x00;                //设置定时器为模式 0(16 位自动重装载)
    TMOD &= ~0x04;              //C/T0=0，对内部时钟进行时钟输出
//  TMOD |= 0x04;               //C/T0=1，对 T0 引脚的外部时钟进行时钟输出
```

```
    TL0 = F38_4KHz;             //初始化计时值
    TH0 = F38_4KHz >> 8;
    TR0 = 1;
    INT_CLKO = 0x01;            //使能定时器 0 的时钟输出功能
    while (1);                  //程序终止
}
```

2.汇编程序:

```
/*----演示 STC15F2K60S2 系列单片机定时器 0 的可编程时钟分频输出----------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可----------*/
;假定测试芯片的工作频率为 18.432MHz

AUXR        DATA    08EH        ;辅助特殊功能寄存器
INT_CLKO    DATA    08FH        ;唤醒和时钟输出功能寄存器

T0CLKO      BIT     P3.5        ;定时器 0 的时钟输出脚

F38_4KHz    EQU     0FF10H      ;38.4KHz(1T 模式下, 65536-18432000/2/38400)
;F38_4KHz   EQU     0FFECH      ;38.4KHz(12T 模式下, (65536-18432000/2/12/38400))
;---------------------------------------------
    ORG     0000H
    LJMP    START               ;复位入口
;---------------------------------------
    ORG     0100H

START:
    MOV     SP,   #3FH

    ORL     AUXR, #80H          ;定时器 0 为 1T 模式
;   ANL     AUXR, #7FH          ;定时器 0 为 12T 模式

    MOV     TMOD, #00H          ;设置定时器为模式 0(16 位自动重装载)

    ANL     TMOD, #0FBH         ;C/T0=0, 对内部时钟进行时钟输出
;   ORL     TMOD, #04H          ;C/T0=1, 对 T0 引脚的外部时钟进行时钟输出

    MOV     TL0, #LOW F38_4KHz  ;初始化计时值
    MOV     TH0, #HIGH F38_4KHz
    SETB    TR0
    MOV     INT_CLKO, #01H      ;使能定时器 0 的时钟输出功能
    SJMP    $                   ;程序终止
;---------------------------------------------
END
```

13.1.3.4 定时器 1 对系统时钟或外部引脚 T1 的时钟输入进行可编程分频输出及测试程序

如何利用 T1CLKO/P3.4 管脚输出时钟

T1CLKO/P3.4 管脚是否输出时钟由 INT_CLKO(AUXR2)寄存器的 T1CLKO 位控制

AUXR2.1-T1CLKO： 1，允许时钟输出

0，禁止时钟输出

T1CLKO 的输出时钟频率由定时器 1 控制，相应的定时器 1 需要工作在定时器的模式 0(16 位自动重装载模式)或模式 2(8 位自动重装载模式)，不要允许相应的定时器中断，免得 CPU 反复进中断，当然在特殊情况下也可允许相应的定时器中断。

新增加的特殊功能寄存器：INT_CLKO(AUXR2)(地址：0x8F)

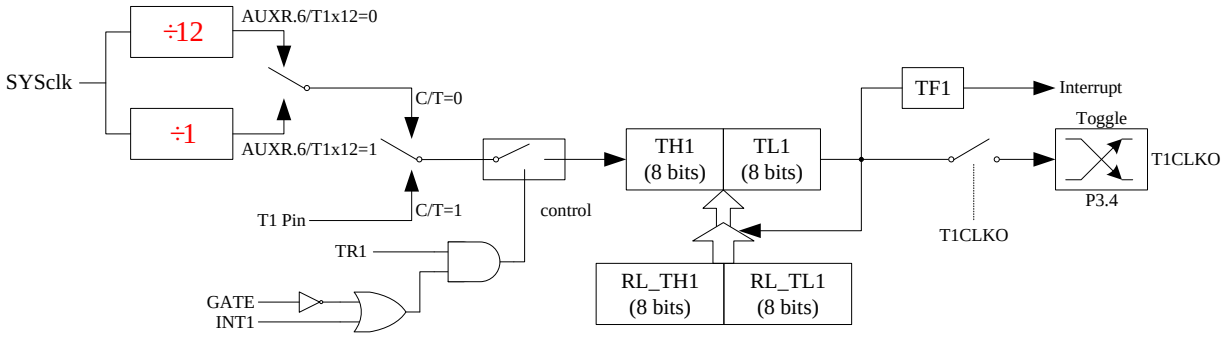
当 T1CLKO/INT_CLKO.1=1 时，P3.4/T0 管脚配置为定时器 1 的时钟输出 T1CLKO。

输出时钟频率=T1 溢出率/2

若定时器/计数器 T1 工作在定时器模式 0(16 位自动重装载模式)时，(如下图所示)

- 如果 C/T=0，定时器/计数器 T1 是对内部系统时钟计数，则：
  - ✓ T1 工作在 1T 模式(AUXR.6/T1x12=1)时的输出频率=(SYSclk)/(65536-[RL_TH1, RL_TL1])/2
  - ✓ T1 工作在 12T 模式(AUXR.6/T1x12=0)时的输出频率=(SYSclk)/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

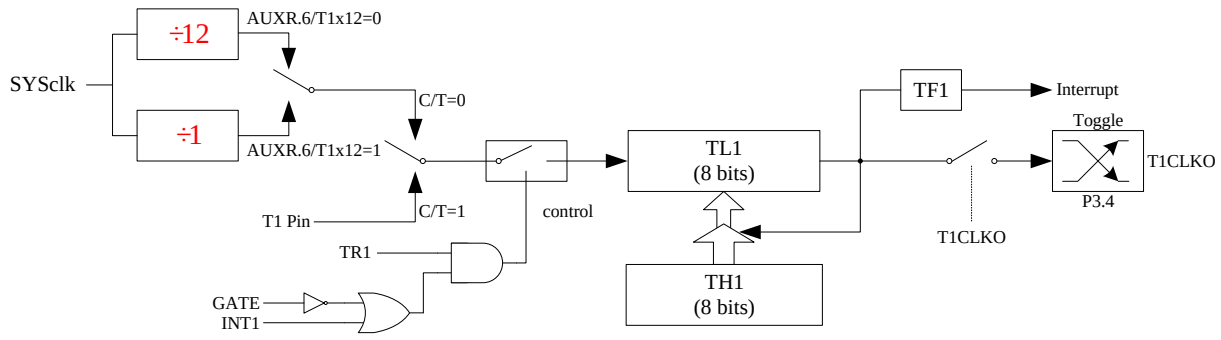
RL_TH1 为 TH1 的重装载寄存器，RL_TL1 为 TL1 的重装载寄存器。



定时器/计数器1的模式0：16位自动重装

当 T1CLKO/INT_CLKO.1=1 且定时器/计数器 T1 工作在定时器模式 2(8 位自动重装模式)时，(如下图所示)

- 如果 C/T=0，定时器/计数器 T1 是对内部系统时钟计数，则：
  - ✓ T1 工作在 1T 模式(AUXR.6/T1x12=1)时的输出频率=(SYSclk)/(256-TH1)/2
  - ✓ T1 工作在 12T 模式(AUXR.6/T1x12=0)时的输出频率=(SYSclk)/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



定时器/计数器1的模式2：8位自动重装

下面是定时器 1 对内部系统时钟或外部引脚 T1/P3.5 的时钟输入进行可编程时钟分频输出的程序举例(C 和汇编):

1.C 程序:

```
/*---演示 STC 15  系列单片机定时器 1 的可编程时钟分频输出----------------------*/
/*---在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
//假定测试芯片的工作频率为 18.432MHz

#include "reg51.h"
typedef     unsigned char     BYTE;
typedef     unsigned int      WORD;

#define     FOSC              18432000L
//-----------------------------------------------
sfr         AUXR = 0x8e;                          //辅助特殊功能寄存器
sfr         INT_CLKO = 0x8f;                      //唤醒和时钟输出功能寄存器

sbit        T1CLKO = P3^4;                        //定时器 1 的时钟输出脚

#define     F38_4KHz          (65536-FOSC/2/38400)      //1T 模式
//#define     F38_4KHz          (65536-FOSC/2/12/38400)   //12T 模式
//-----------------------------------------------
void main()
{
    AUXR |= 0x40;               //定时器 1 为 1T 模式
//  AUXR &= ~0x40;              //定时器 1 为 12T 模式

    TMOD = 0x00;                //设置定时器为模式 0(16 位自动重装载)

    TMOD &= ~0x40;              //C/T1=0，对内部时钟进行时钟输出
//  TMOD |= 0x40;               //C/T1=1，对 T1 引脚的外部时钟进行时钟输出

    TL1 = F38_4KHz;             //初始化计时值
```

```
    TH1 = F38_4KHz >> 8;
    TR1 = 1;
    INT_CLKO = 0x02;          //使能定时器 1 的时钟输出功能

    while (1);                //程序终止
}
```

2.汇编程序:

```
/*---演示 STC15  系列单片机定时器 1 的可编程时钟分频输出-------------------------*/
/*---在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可……*/
;假定测试芯片的工作频率为 18.432MHz

AUXR        DATA    08EH        ;辅助特殊功能寄存器
INT_CLKO    DATA    08FH        ;唤醒和时钟输出功能寄存器

T1CLKO      BIT     P3.4        ;定时器 1 的时钟输出脚

F38_4KHz    EQU     0FF10H      ;38.4KHz(1T 模式下, 65536-18432000/2/38400)
;F38_4KHz   EQU     0FFECH      ;38.4KHz(12T 模式下, (65536-18432000/2/12/38400))

    ORG     0000H
    LJMP    START               ;复位入口
;---------------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    ORL     AUXR, #40H          ;定时器 1 为 1T 模式

;   ANL     AUXR, #0BFH         ;定时器 1 为 12T 模式
    MOV     TMOD, #00H          ;设置定时器为模式 0(16 位自动重装载)

    ANL     TMOD, #0BFH         ;C/T1=0, 对内部时钟进行时钟输出
;   ORL     TMOD, #40H          ;C/T1=1, 对 T1 引脚的外部时钟进行时钟输出

    MOV     TL1, #LOW F38_4KHz  ;初始化计时值
    MOV     TH1, #HIGH F38_4KHz
    SETB    TR1
    MOV     INT_CLKO, #02H      ;使能定时器 1 的时钟输出功能

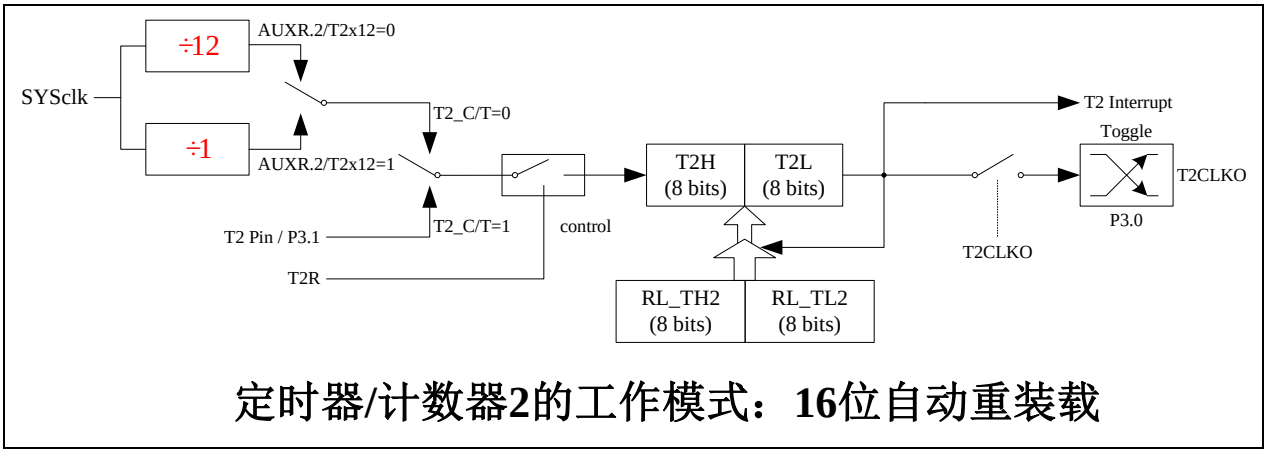
    SJMP    $                   ;程序终止
;---------------------------------------------

END
```

13.1.3.5 定时器 2 对系统时钟或外部引脚 T2 的时钟输入进行可编程分频输出及测试程序

T2 可以当定时器用，也可以当串口的波特率发生器和可编程时钟输出。

定时器 2 的原理框图如下:



如何利用 T2CLKO/P3.0 管脚输出时钟

AUXR2.2-T2CLKO：是否允许将 P3.0 脚配置为定时器 2(T2)的时钟输出 T2CLKO

1：允许将 P3.0 脚配置为定时器 2(T2)的时钟输出 T2CLKO

0：不允许将 P3.0 脚配置为定时器 2(T2)的时钟输出 T2CLKO

当 T2CLKO/INT_CLKO.2=1 时，P3.0 管脚配置为定时器 2 的时钟输出 T2CLKO。

输出时钟频率 = T2 溢出率/2

- 如果 T2_C/T=0，定时器/计数器 T2 对内部系统时钟计数，则：
  - ✓ T2 工作在 1T 模式(AUXR.2/T2x12=1)时的输出时钟频率=(SYSclk)/(65536-[RL_TH2, RL_TL2])/2
  - ✓ T2 工作在 12T 模式(AUXR.2/T2x12=0)时的输出时钟频率=(SYSclk)/12/(65536-[RL_TH2, RL_TL2])/2
- 如果 T2_C/T=1，定时器/计数器 T2 是对外部脉冲输入(P3.1/T2)计数，则：
  - ✓ 输出时钟频率=(T2_Pin_CLK)/(65536-[RL_TH2, RL_TL2])/2

RL_TH2 为 T2H 的重装载寄存器，RL_TL2 为 T2L 的重装载寄存器。

用户在程序中如何具体设置 T2CLKO/P3.0 管脚输出时钟

1.对定时器 2 寄存器 T2H/T2L 送 16 位重装载值，[T2H, T2L] = #reload_data

2.对 AUXR 寄存器中的 T2R 位置 1，让定时器 2 运行

3.对 AUXR2/INT_CLKO 寄存器中的 T2CLKO 位置 1，让定时器 2 的溢出在 P3.0 口输出时钟。

注意：当定时器/计数器 2 用作可编程时钟输出时，不要允许相应的定时器中断，免得 CPU 反复进中断，在特殊情况下也可允许定时器/计数器 2 中断。

下面是定时器 2 对内部系统时钟或外部引脚 T2/P3.1 的时钟输入进行可编程时钟分频输出的程序举例(C 和汇编)

1.C 程序：

```
/*------STC15F2K60S2 系列定时器 2 的可编程时钟分频输出举例----------------------*/
/*------在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可--------*/
//假定测试芯片的工作频率为 18.432MHz
```

```
#include "reg51.h"
typedef     unsigned char     BYTE;
typedef     unsigned int      WORD;

#define     FOSC              18432000L
//-----------------------------------------------
sfr         AUXR = 0x8e;                          //辅助特殊功能寄存器
sfr         INT_CLKO = 0x8f;                      //唤醒和时钟输出功能寄存器
sfr         T2H = 0xD6;                           //定时器 2 高 8 位
sfr         T2L = 0xD7;                           //定时器 2 低 8 位

sbit        T2CLKO = P3^0;                        //定时器 2 的时钟输出脚

#define     F38_4KHz          (65536-FOSC/2/38400)      //1T 模式
//#define   F38_4KHz          (65536-FOSC/2/12/38400)   //12T 模式
//-----------------------------------------------
void main()
{
    AUXR |= 0x04;             //定时器 2 为 1T 模式
//  AUXR &= ~0x04;            //定时器 2 为 12T 模式

    AUXR &= ~0x08;            //T2_C/T=0, 对内部时钟进行时钟输出
//  AUXR |= 0x08;             //T2_C/T=1, 对 T2(P3.1)引脚的外部时钟进行时钟输出

    T2L = F38_4KHz;           //初始化计时值
    T2H = F38_4KHz >> 8;

    AUXR |= 0x10;             //定时器 2 开始计时
    INT_CLKO = 0x04;          //使能定时器 2 的时钟输出功能

    while (1);                //程序终止
}
```

2.汇编程序:

```
/*----STC15F2K60S2 系列定时器 2 可编程时钟分输出举例------------------------------*/
/*----在 Keil C 开发环境中选择  Intel 8052 编译，头文件包含<reg51.h>即可--------*/
;假定测试芯片的工作频率为 18.432MHz

AUXR        DATA    08EH        ;辅助特殊功能寄存器
INT_CLKO    DATA    08FH        ;唤醒和时钟输出功能寄存器
T2H         DATA    0D6H        ;定时器 2 高 8 位
T2L         DATA    0D7H        ;定时器 2 低 8 位

T2CLKO      BIT     P3.0        ;定时器 2 的时钟输出脚
```

```
F38_4KHz        EQU     0FF10H          ;38.4KHz(1T 模式下, 65536-18432000/2/38400)
;F38_4KHz       EQU     0FFECH          ;38.4KHz(12T 模式下, (65536-18432000/2/12/38400))
;---------------------------------------------
        ORG     0000H
        LJMP    START                   ;复位入口
;---------------------------------------
        ORG     0100H
START:
        MOV     SP, #3FH

        ORL     AUXR, #04H              ;定时器 2 为 1T 模式
;       ANL     AUXR, #0FBH             ;定时器 2 为 12T 模式

        ANL     AUXR, #0F7H             ;T2_C/T=0, 对内部时钟进行时钟输出
;       ORL     AUXR, #08H              ;T2_C/T=1, 对 T2(P3.1)引脚的外部时钟进行时钟输出

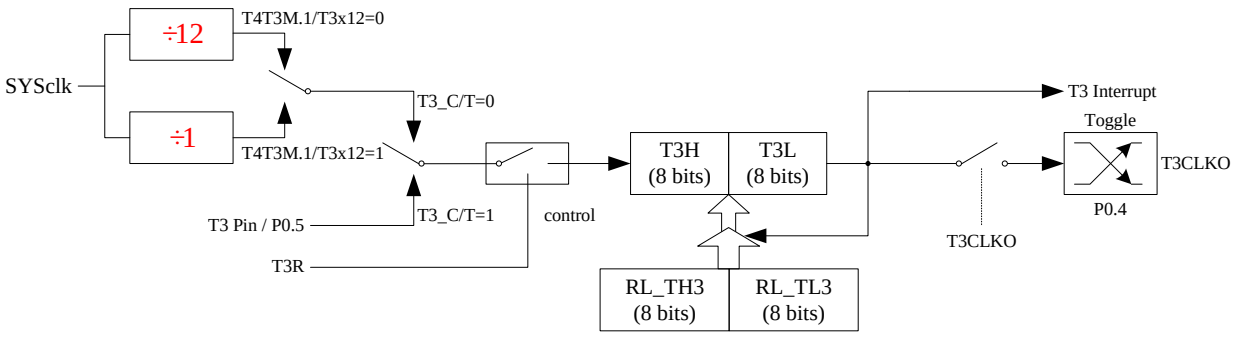
        MOV     T2L, #LOW F38_4KHz      ;初始化计时值
        MOV     T2H, #HIGH F38_4KHz
        ORL     AUXR, #10H              ;定时器 2 开始计时
        MOV     INT_CLKO, #04H          ;使能定时器 2 的时钟输出功能

        SJMP    $                       ;程序终止
;---------------------------------------------
END
```

13.1.3.6 定时器 3 对系统时钟或外部引脚 T3 的时钟输入进行可编程分频输出及测试程序

T3 可以当定时器用，也可以当串口 3 的波特率发生器和可编程时钟输出。

定时器 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

如何利用 T3CLKO/P0.4 管脚输出时钟

T4T3M.0-T3CLKO：是否允许将 P0.4 脚配置为定时器 3(T3)的时钟输出 T3CLKO

1：允许将 P0.4 脚配置为定时器 3(T3)的时钟输出 T3CLKO

0：不允许将 P0.4 脚配置为定时器 3(T3)的时钟输出 T3CLKO

当 T3CLKO/T4T3M.0=1 时，P0.4 管脚配置为定时器 3 的时钟输出 T3CLKO。
输出时钟频率 = T3 溢出率/2

- 如果 T3_C/T=0，定时器/计数器 T3 对内部系统时钟计数，则：
  - ✓ T3 工作在 1T 模式(T4T3M.1/T3x12=1)时的输出时钟频率=(SYSclk)/(65536-[RL_TH3, RL_TL3])/2
  - ✓ T3 工作在 12T 模式(T4T3M.1/T3x12=0)时的输出时钟频率=(SYSclk)/12/(65536-[RL_TH3, RL_TL3])/2
- 如果 T3_C/T=1，定时器/计数器 T3 是对外部脉冲输入(P0.5/T3)计数，则：
  - ✓ 输出时钟频率=(T3_Pin_CLK)/(65536-[RL_TH3, RL_TL3])/2

RL_TH3 为 T3H 的重装载寄存器，RL_TL3 为 T3L 的重装载寄存器。

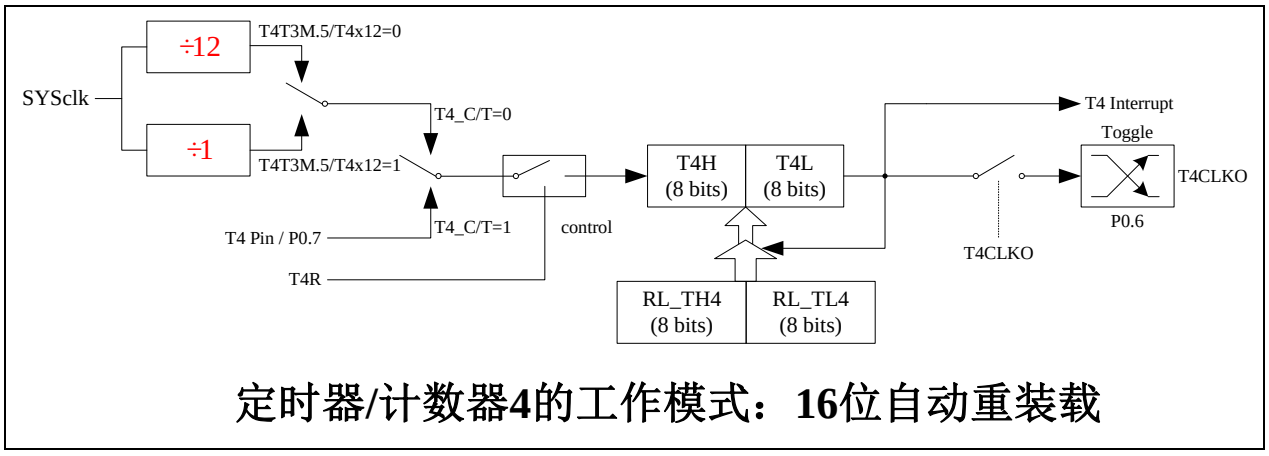
用户在程序中如何具体设置 T3CLKO/P0.4 管脚输出时钟
1.对定时器 3 寄存器 T3H/T3L 送 16 位重装载值，[T3H, T3L] = #reload_data
2.对 T4T3M 寄存器中的 T3R 位置 1，让定时器 3 运行
3.对 T4T3M 寄存器中的 T3CLKO 位置 1，让定时器 3 的溢出在 P0.4 口输出时钟。

注意：当定时器/计数器 3 用作可编程时钟输出时，不要允许相应的定时器中断，免得 CPU 反复进中断，在特殊情况下也可允许定时器/计数器 3 中断。

13.1.3.7 定时器 4 对系统时钟或外部引脚 T4 的时钟输入进行可编程分频输出及测试程序

T4 可以当定时器用，也可以当串口 4 的波特率发生器和可编程时钟输出。

定时器 4 的原理框图如下:



如何利用 T4CLKO/P0.6 管脚输出时钟

T4T3M.4-T4CLKO：是否允许将 P0.6 脚配置为定时器 4(T4)的时钟输出 T4CLKO
1：允许将 P0.6 脚配置为定时器 4(T4)的时钟输出 T4CLKO,
0：不允许将 P0.6 脚配置为定时器 4(T4)的时钟输出 T4CLKO

当 T4CLKO/T4T3M.4=1 时，P0.6 管脚配置为定时器 4 的时钟输出 T4CLKO。
输出时钟频率 = T4 溢出率/2

- 如果 T4_C/T=0，定时器/计数器 T4 对内部系统时钟计数，则：
  - ✓ T4 工作在 1T 模式(T4T3M.5/T4x12=1)时的输出时钟频率=(SYSclk)/(65536-[RL_TH4, RL_TL4])/2
  - ✓ T4 工作在 12T 模式(T4T3M.5/T4x12=0)时的输出时钟频率=(SYSclk)/12/(65536-[RL_TH4, RL_TL4])/2

- 如果 T4_C/T=1，定时器/计数器 T4 是对外部脉冲输入(P0.7/T4)计数，则：
  - ✓ 输出时钟频率=(T4_Pin_CLK)/(65536-[RL_TH4, RL_TL4])/2

RL_TH4 为 T4H 的重装载寄存器，RL_TL4 为 T4L 的重装载寄存器。

用户在程序中如何具体设置 T4CLKO/P0.6 管脚输出时钟

1.对定时器 4 寄存器 T4H/T4L 送 16 位重装载值，[T4H, T4L] = #reload_data

2.对 T4T3M 寄存器中的 T4R 位置 1，让定时器 4 运行

3.对 T4T3M 寄存器中的 T4CLKO 位置 1，让定时器 4 的溢出在 P0.6 口输出时钟。

注意：当定时器/计数器 4 用作可编程时钟输出时，不要允许相应的定时器中断，免得 CPU 反复进中断，在特殊情况下也可允许定时器/计数器 4 中断。

13.2 复位

STC15 系列单片机有 7 种复位方式：

1. 外部 RST 引脚复位
2. 软件复位
3. 掉电复位/上电复位(并可选择增加额外的复位延时 180mS，也叫 MAX810 专用复位电路，其实就是在上电复位后增加一个 180ms 复位延时)
4. 内部低压检测复位
5. MAX810 专用复位电路复位
6. 看门狗复位
7. 程序地址非法复位

13.2.1 外部 RST 引脚复位

STC15F104W 系列单片机的复位管脚在 RST/P3.4 口，其他 STC15 系列单片机的复位管脚均在 RST/P5.4 口。下面以 P5.4/RST 为例介绍外部 RST 引脚的复位。

外部 RST 引脚复位就是从外部向 RST 引脚施加一定宽度的复位脉冲，从而实现单片机的复位。P5.4/RST 管脚出厂时被配置为 I/O 口，要将其配置为复位管脚，可在 ISP 烧录程序时设置。如果 P5.4/RST 管脚已在 ISP 烧录程序时被设置为复位脚，那 P5.4/RST 就是芯片复位的输入脚。将 RST 复位管脚拉高并维持至少 24 个时钟加 20us 后，单片机会进入复位状态。将 RST 复位管脚拉回低电平后，单片机结束复位状态并将特殊功能寄存器 IAP_CONTR 中的 SWBS/IAP_CONTR.6 位置 1，同时从系统 ISP 监控程序区启动。外部 RST 引脚复位是热启动复位中的硬复位。

13.2.2 软件复位及其测试程序(C 和汇编)

用户应用程序在运行过程当中，有时会有特殊需求，需要实现单片机系统软复位(热启动复位中的软复位之一)，传统的 8051 单片机由于硬件上未支持此功能，用户必须用软件模拟实现，实现起来较麻烦。现 STC 新推出的增强型 8051 根据客户要求增加了 IAP_CONTR 特殊功能寄存器，实现了此功能。用户只需简单的控制 IAP_CONTR 特殊功能寄存器的其中两位 SWBS / SWRST 就可以实现系统复位了。

IAP_CONTR：ISP/IAP 控制寄存器

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CONTR	C7H	name	IAPEN	SWBS	SWRST	CMD_FAIL	-	WT2	WT1	WT0

IAPEN：ISP/IAP 功能允许位。0：禁止 IAP 读/写/擦除 Data Flash/EEPROM
1：允许 IAP 读/写/擦除 Data Flash/EEPROM

SWBS：软件选择复位后从用户应用程序区启动(送 0)，还是从系统 ISP 监控程序区启动(送 1)。要与 SWRST 直接配合才可以实现

SWRST：0：不操作；1：软件控制产生复位，单片机自动复位。

CMD_FAIL：如果 IAP 地址(由 IAP 地址寄存器 IAP_ADDRH 和 IAP_ADDRL 的值决定)指向了非法地址或无效地址，且送了 ISP/IAP 命令，并对 IAP_TRIG 送 5AH/A5h 触发失败，则 CMD_FAIL 为 1，需由软件清零。

```
;从用户应用程序区(AP 区)软件复位并切换到用户应用程序区(AP 区)开始执行程序
MOV     IAP_CONTR, #00100000B       ;SWBS=0(选择 AP 区)，SWRST=1(软复位)

;从系统 ISP 监控程序区软件复位并切换到用户应用程序区(AP 区)开始执行程序
MOV     IAP_CONTR, #00100000B       ;SWBS=0(选择 AP 区)，SWRST =1(软复位)

;从用户应用程序区(AP 区)软件复位并切换到系统 ISP 监控程序区开始执行程序
MOV     IAP_CONTR, #01100000B       ;SWBS=1(选择 ISP 区)，SWRST =1(软复位)

;从系统 ISP 监控程序区软件复位并切换到系统 ISP 监控程序区开始执行程序
MOV     IAP_CONTR, #01100000B       ;SWBS=1(选择 ISP 区)，SWRST =1(软复位)
```

本复位是整个系统复位，所有的特殊功能寄存器都会复位到初始值，I/O 口也会初始化

1.C 程序:

```
/*---STC15F2K60S2 系列软件复位举例------------------------------------------------*/
/*---在 Keil C 开发环境中选择  Intel 8052 编译，头文件包含<reg5l.h>即可.------*/
//假定测试芯片的工作频率为 18.432MHz
#include "reg5l.h"
#include "intrins.h"
//-----------------------------------------------------------------------------------------
sfr     IAP_CONTR = 0x7;        //IAP 控制寄存器
sbit    P10 = P1^0;
void delay()                    //软件延时
{
    int i;
    for (i=0; i<10000; i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
}

void main()
{
    P10 = !P10;                 //上电 P1.0 闪烁一次,便于观察
    delay();
    P10 = !P10;
    delay();

    IAP_CONTR = 0x20;           //软件复位,系统重新从用户代码区开始运行程序
//  IAP_CONTR = 0x60;           //软件复位,系统重新从 ISP 代码区开始运行程序

    while (1);
}
```

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 软件复位举例-----------------------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz

IAP_CONTR       DATA    0C7H        ;IAP 控制寄存器
;-------------------------------------------
    ORG         0000H
    LJMP        START               ;复位入口
;-------------------------------------
    ORG         0100H
START:
    MOV         SP, #3FH

    CPL         P1.0                ;上电 P1.0 闪烁一次,便于观察
    LCALL       DELAY
    CPL         P1.0
    LCALL       DELAY

    MOV         IAP_CONTR, #20H     ;软件复位,系统重新从用户代码区开始运行程序
;   MOV         IAP_CONTR, #60H     ;软件复位,系统重新从 ISP 代码区开始运行程序
    JMP         $
;-------------------------------------------
DELAY:
    MOV         R0, #0              ;软件延时
    MOV         R1, #0

WAIT:
    DJNZ        R0, WAIT
    DJNZ        R1, WAIT
    RET
;-------------------------------------------
END
```

13.2.3 掉电复位/上电复位

当电源电压 VCC 低于掉电复位/上电复位检测门槛电压时，所有的逻辑电路都会复位。当内部 VCC 上升至上电复位检测门槛电压以上后，延迟 32768 个时钟，掉电复位/上电复位结束。复位状态结束后，单片机将特殊功能寄存器 IAP_CONTR 中的 SWBS/IAP_CONTR.6 位置 1，同时从系统 ISP 监控程序区启动。掉电复位/上电复位是冷启动复位之一。

对于 5V 单片机，它的掉电复位/上电复位检测门槛电压为 3.2V；对于 3.3V 单片机，它的掉电复位/上电复位检测门槛电压为 1.8V。

13.2.4 MAX810 专用复位电路复位

STC15 系列单片机内部集成了 MAX810 专用复位电路。若 MAX810 专用复位电路在 AIapp-ISP 编程器中被允许，则以后掉电复位/上电复位后将产生约 180ms 复位延时，复位才被解除。复位解除后单片机将特殊功能寄存器 IAP_CONTR 中的 SWBS/IAP_CONTR.6 位置 1，同时从系统 ISP 监控程序区启动。MAX810 专用复位电路复位是冷启动复位之一。

13.2.5 内部低压检测复位

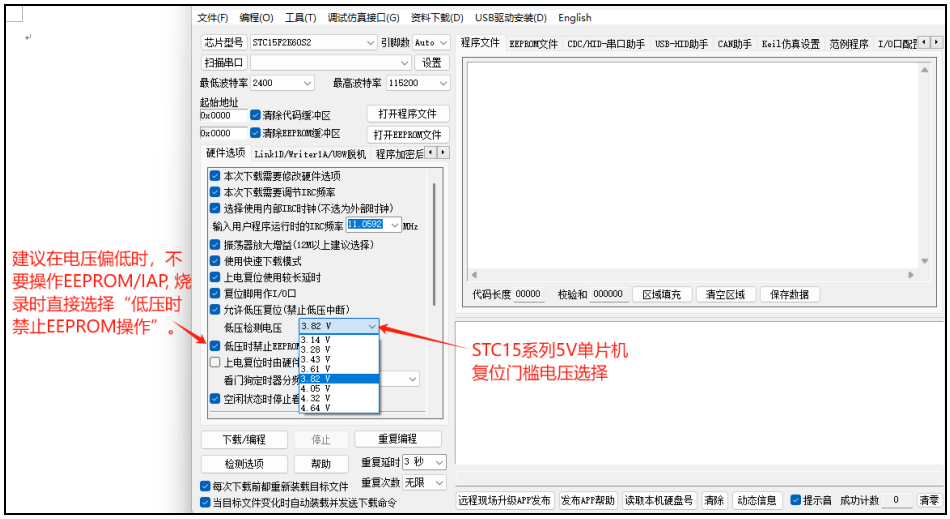
除了上电复位检测门槛电压外，STC15 单片机还有一组更可靠的内部低压检测门槛电压。当电源电压 Vcc 低于内部低压检测(LVD)门槛电压时，可产生复位(前提是在 AIapp-ISP 编程/烧录用户程序时，允许低压检测复位/禁止低压中断，即将低压检测门槛电压设置为复位门槛电压)。低压检测复位结束后，不影响特殊功能寄存器 IAP_CONTR 中的 SWBS/IAP_CONTR.6 位的值，单片机根据复位前 SWBS/IAP_CONTR.6 的值选择是从用户应用程序区启动，还是从系统 ISP 监控程序区启动。如果复位前 SWBS/IAP_CONTR.6 的值为 0，则单片机从用户应用程序区启动。反之，如果复位前 SWBS/IAP_CONTR.6 的值为 1，则单片机从系统 ISP 监控程序区启动。内部低压检测复位是热启动复位中的硬复位之一。

STC15 单片机内置了 8 级可选内部低压检测门槛电压。下表列出了不同温度下 STC15 系列 5V 单片机和 3.3V 单片机所有的低压检测门槛电压。

5V 单片机的低压检测门槛电压：

-40 ℃	25 ℃	85 ℃
4.74	4.64	4.60
4.41	4.32	4.27
4.14	4.05	4.00
3.90	3.82	3.77
3.69	3.61	3.56
3.51	3.43	3.38
3.36	3.28	3.23
3.21	3.14	3.09

如果用户所使用的是 STC15 系列 5V 单片机，那么用户可以根据单片机的实际工频率在 AIapp-ISP 编程器中选择上表中所列出的低压检测门槛电压作为复位门槛电压。如：常温下工作频率是 20MHz 以上时，可以选择 4.32V 电压作为复位门槛电压；常温下工作频率是 12MHz 以下时，可以选择 3.82V 电压作为复位门槛电压。

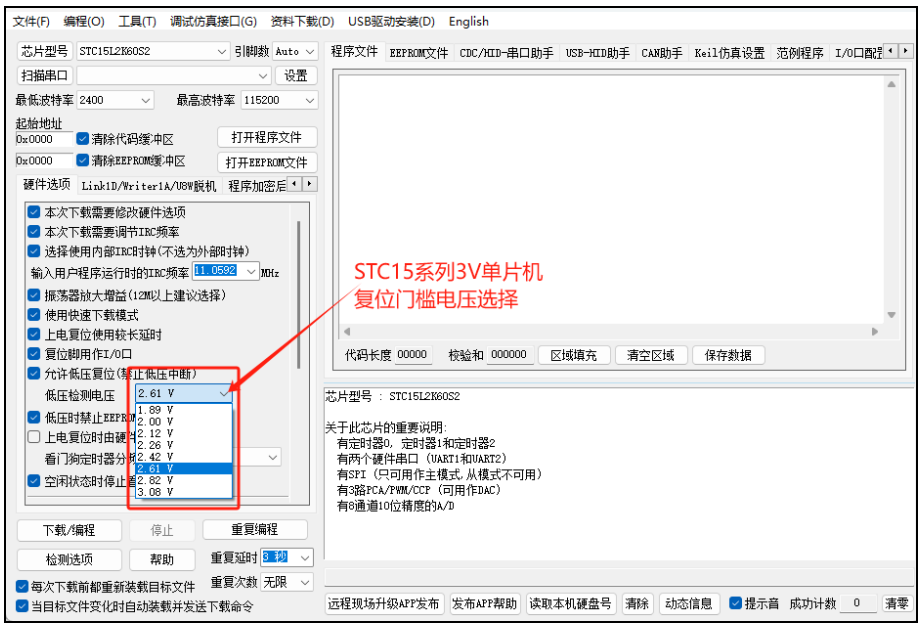


3.3V 单片机的低压检测门槛电压：

-40 0C	25 0C	85 0C
3.11	3.08	3.09
2.85	2.82	2.83
2.63	2.61	2.61
2.44	2.42	2.43
2.29	2.26	2.26
2.14	2.12	2.12
2.01	2.00	2.00
1.90	1.89	1.89

如果用户所使用的是 STC15 系列 3.3V 单片机，那么用户可以根据单片机的实际工作频率在 AIapp-ISP 编程器中选择上表中所列出的低压检测门槛电压作为复位门槛电压。如：常温下工作频率是 20MHz 以上时，可以选择 2.82V 电压作为内部低压检测复位门槛电压；常温下工作频率是 12MHz 以下时，可以选择 2.42V 电压作为复位门槛电压。

建议在电压偏低时，不要操作 EEPROM/IAP，烧录时直接选择“低压禁止 EEPROM 操作”

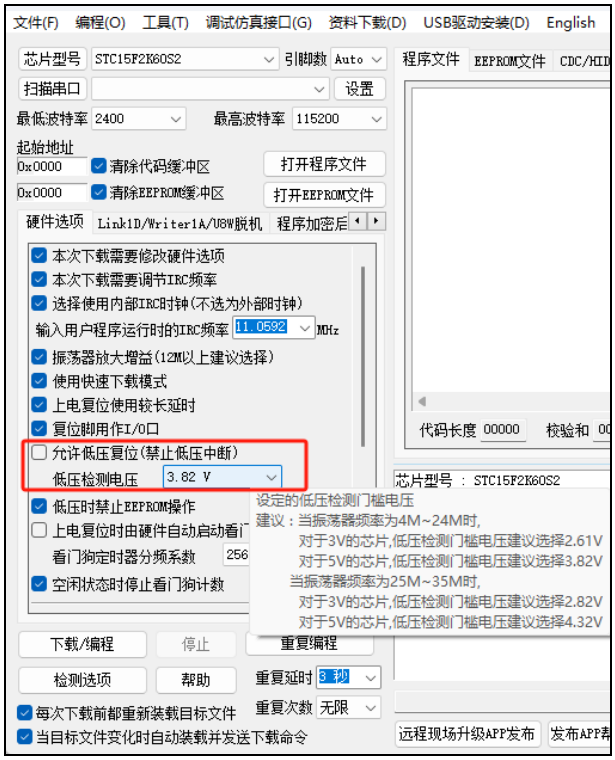


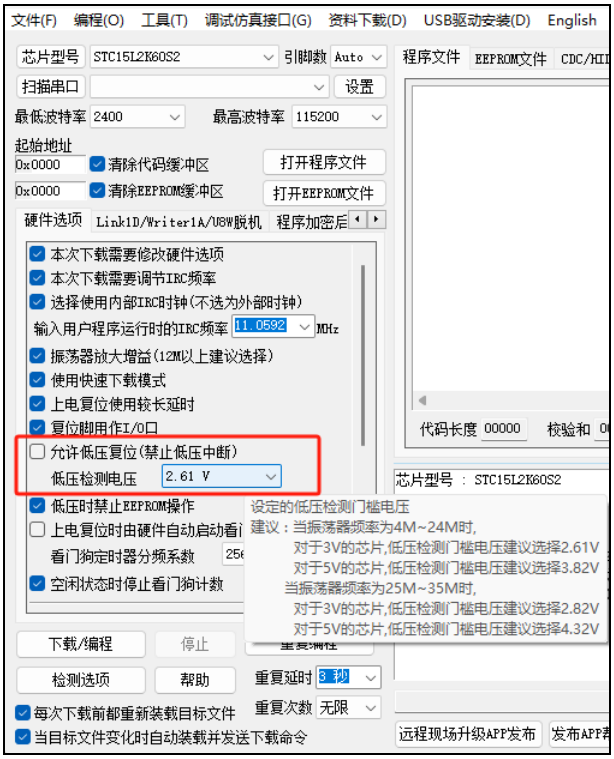
如果在 AIapp-ISP 编程/烧录用户应用程序时，不将低压检测设置为低压检测复位，则在用户程序中用户可将低压检测设置为低压检测中断。当电源电压 VCC 低于内部低压检测(LVD)门槛电压时，低压检测中断请求标志位(LVDF/PCON.5)就会被硬件置位。如果 ELVD/IE.6(低压检测中断允许位)被设置为 1，低压检测中断请求标志位就能产生一个低压检测中断。

在正常工作和空闲工作状态时，如果内部工作电压 Vcc 低于低压检测门槛电压，低压中断请求标志位(LVDF/PCON.5)自动置 1，与低压检测中断是否被允许无关。即在内部工作电压 Vcc 低于低压检测门槛电压时，不管有没有允许低压检测中断，LVDF/PCON.5 都自动为 1。该位要用软件清 0，清 0 后，如内部工作电压 Vcc 低于低压检测门槛电压，该位又被自动设置为 1。

在进入掉电工作状态前，如果低压检测电路未被允许可产生中断，则在进入掉电模式后，该低压检测电路不工作以降低功耗。如果被允许可产生低压检测中断(相应的中断允许位是 ELVD/E.6，中断请求标志位是 LVDF/PCON.5)，则在进入掉电模式后，该低压检测电路继续工作，在内部工作电压 Vcc 低于低压检测门槛电压后，产生低压检测中断，可将 MCU 从掉电状态唤醒。

建议在电压偏低时，不要操作 EEPROM/AP，烧录时直接选择“低压禁止 EEPROM 操作”





与低压检测相关的一些寄存器:

PCON：电源控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
PCON	87H	name	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL

LVDF： 低压检测标志位，同时也是低压检测中断请求标志位。

在正常工作和空闲工作状态时，如果内部工作电压 Vcc 低于低压检测门槛电压，该位自动置 1，与低压检测中断是否被允许无关。即在内部工作电压 Vcc 低于低压检测门槛电压时，不管有没有允许低压检测中断，该位都自动为 1。该位要用软件清 0，清 0 后，如内部工作电压 Vcc 继续低于低压检测门槛电压，该位又被自动设置为 1。

在进入掉电工作状态前，如果低压检测电路未被允许可产生中断，则在进入掉电模式后，该低压检测电路不工作以降低功耗。如果被允许可产生低压检测中断，则在进入掉电模式后，该低压检测电路继续工作，在内部工作电压 Vcc 低于低压检测门槛电压后，产生低压检测中断，可将 MCU 从掉电状态唤醒。

PD： 掉电模式控制位

IDL： 空闲模式控制位

GF1，GF0：两个通用工作标志位，用户可以任意使用。

IE：中断允许寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	name	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

EA： 中断允许总控制位

EA=0，屏蔽所有的中断请求

EA=1，开放中断，但每个中断源还有自己的独立允许控制位。

ELVD： 低压检测中断允许位

ELVD=0，禁止低压检测中断

ELVD=1，允许低压检测中断

IP：中断优先级控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IP	B8H	name	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0

PLVD： 低压检测中断优先级控制位

PLVD=0，低压检测中断位低优先级

PLVD=1，低压检测中断为高优先级

13.2.6 看门狗(WDT)复位

在工业控制/汽车电子/航空航天等需要高可靠性的系统中，为了防止“系统在异常情况下，受到干扰，MCU/CPU 程序跑飞，导致系统长时间异常工作”，通常是引进看门狗，如果 MCU/CPU 不在规定的时间内按要求访问看门狗，就认为 MCU/CPU 处于异常状态，看门狗就会强迫 MCU/CPU 复位，使系统重新从头开始按规律执行用户程序。

看门狗复位是热启动复位中的软复位之一。STC15 系列单片机内部也引进了此看门狗功能，使单片机系统可靠性设计变得更加方便/简洁。

看门狗复位状态结束后，不影响特殊功能寄存器 IAP_CONTR 中 SWBS/IAP_CONTR.6 位的值。

对于 STC15F/L101W 系列、STC15F/L2K60S2 系列、STC15F/L408AD 系列及 STC15W408AS 系列单片机，它们根据复位前 SWBS/IAP_CONTR.6 的值，选择是从用户应用程序区启动，还是从系统 ISP 监控程序区启动。

- 如果看门狗复位前它们的 SWBS/IAP_CONTR.6 的值为 0，则看门口复位状态结束后上述系列单片机将从用户应用程序区启动。
- 如果看门狗复位前它们的 SWBS/IAP_CONTR.6 的值为 1，则看门口复位状态结束后上述系列单片机将从系统 ISP 监控程序区启动。

对于 STC15W204S 系列、STC15W1K16S 系列及 STC15W404S 系列单片机，它们的看门狗复位状态结束后始终从系统 ISP 监控程序区启动，与复位前 SWBS/IAP_CONTR.6 的值无关。

对于看门狗复位功能，我们增加如下特殊功能寄存器 WDT_CONTR：

WDT_CONTR：看门狗(Watch-Dog-Timer)控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
WDT_CONTR	0C1H	name	WDT_FLAG	-	EN_WDT	CLR_WDT	IDLE_WDT	PS2	PS1	PS0

Symbol 符号　　Function 功能

WDT_FLAG：　When WDT overflows, this bit is set. It can be cleared by software.
看门狗溢出标志位，当溢出时，该位由硬件置 1，可用软件将其清 0。

EN_WDT：　Enable WDT bit. When set, WDT is started
看门狗允许位，当设置为“1”时，看门狗启动。

CLR_WDT：　WDT clear bit. If set, WDT will recount. Hardware will automatically clear this bit.
看门狗清“0”位,当设为“1”时,看门狗将重新计数。硬件将自动清“0”此位。

IDLE_WDT：　When set, WDT is enabled in IDLE mode. When clear, WDT is disabled in IDLE
看门狗“IDLE”模式位，当设置为“1”时，看门狗定时器在“空闲模式”计数当清“0”该位时，看门狗定时器在“空闲模式”时不计数

PS2,PS1,PS0：　Pre-scale value of Watchdog timer is shown as the bellowed table:
看门狗定时器预分频值，如下表所示

The WDT period is determined by the following equation 看门狗溢出时间计算：
看门狗溢出时间 = (12×Pre-scale×32768) / Oscillator frequency

其中，设时钟为 20MHz：
则：看门狗溢出时间 = (12Pre-scale×32768) / 12000000 = Pre-scale×393216 / 20000000

PS2	PS1	PS0	Pre-scale 预分频	WDT overflow Time @20MHz
0	0	0	2	39.3 ms
0	0	1	4	78.6 ms
0	1	0	8	157.3 ms
0	1	1	16	314.6 ms
1	0	0	32	629.1 ms
1	0	1	64	1.25 s
1	1	0	128	2.5 s
1	1	1	256	5 s

又设时钟为 12MHz：
看门狗溢出时间 = (12×Pre-scale×32768) / 12000000 = Pre-scale×393216 / 12000000

PS2	PS1	PS0	Pre-scale 预分频	WDT overflow Time @12MHz
0	0	0	2	65.5 ms
0	0	1	4	131.0 ms

0	1	0	8	262.1 ms
0	1	1	16	524.2 ms
1	0	0	32	1.0485 s
1	0	1	64	2.0971 s
1	1	0	128	4.1943 s
1	1	1	256	8.3886 s

再设时钟为 11.0592MHz：

看门狗溢出时间= (12×Pre-scale×32768) / 11059200 = Pre-scale×393216 / 11059200

PS2	PS1	PS0	Pre-scale	WDT overflow Time @11.0592MHz
0	0	0	2	71.1 ms
0	0	1	4	142.2 ms
0	1	0	8	284.4 ms
0	1	1	16	568.8 ms
1	0	0	32	1.1377 s
1	0	1	64	2.2755 s
1	1	0	128	4.5511 s
1	1	1	256	9.1022 s



看门狗测试程序，在 STC 的下载板上可以直接测试

/*---- 演示 STC15 系列单片机看门狗及其溢出时间计算公式--------*/

/*---- 在 Keil C 开发环境中，选择 Intel 8052 编译即可----------*/

;本演示程序在 STC15 系列 ISP 的下载编程工具上测试通过，相关的工作状态在 P1 口上显示

```
;看门狗及其溢出时间 = (12 * Pre_scale *32768)/Oscillator frequency

WDT_CONTR       EQU     0C1H            ;看门狗地址
WDT_TIME_LED    EQU     P1.5            ;用 P1.5 控制看门狗溢出时间指示灯,
                                        ;看门狗溢出时间可由该指示灯亮的时间长度或熄灭的时间长度表示
WDT_FLAG_LED    EQU     P1.7
                                        ;用 P1.7 控制看门狗溢出复位指示灯，如点亮表示为看门狗溢出复位

Last_WDT_Time_LED_Status        EQU     00H     ;位变量，存储看门狗溢出时间指示灯的上一次状态位
                                        ;WDT 复位时间(所用的 0scillator frequency = 18.432MHz):
;Pre_scale_Word  EQU     00111100B       ;清 0，启动看门狗，预分频数=32，0.68s
Pre_scale_Word  EQU     00111101B       ;清 0，启动看门狗，预分频数=64，1.36s
;Pre_scale_Word  EQU     00111110B       ;清 0，启动看门狗，预分频数=128，2.72s
Pre_scale_Word  EQU     00111111B       ;清 0，启动看门狗，预分频数=256，5.44s

    ORG     0000H
    AJMP    START

    ORG     0100H

START:
    MOV     A, WDT CONTR                ;检测是否为看门狗复位
    ANL     A，#10000000B
    JNZ     WDT_Reset
                                        ;WDT_CONTR.7=1，看门狗复位，跳转到看门狗复位程序
                                        ;WDT CONTR.7=0，上电复位，冷启动，RAM 单元内容为随机值
    SETB    Last_WDT_Time_LED_Status    ;上电复位，
                                        ;初始化看门狗溢出时间指示灯的状态位=1
    CLR     WDT_TIME_LED                ;上电复位，点亮看门狗溢出时间指示灯
    MOV     WDT_CONTR, #Pre scale Word  ;启动看门狗

WAIT1:
    SJMP    WAIT1                       ;循环执行本语句(停机)，等待看门狗溢出复位
                                        ;WDT CONTR.7=1,看门狗复位，热启动，RAM 单元内容不变，为复位前的值

WDT_Reset:                              ;看门狗复位，热启动
    CLR     WDT_FLAG_LED                ;是看门狗复位，点亮看门狗溢出复位指示灯
    JB      Last_WDT_Time_LED_Status, Power_Off_WDT_TIME_LED
                                        ;为 1 熄灭相应的灯,为 0 亮相应灯
                                        ;根据看门狗溢出时间指示灯的上一次状态位设置 WDT_TIME_LED 灯，
                                        ;若上次亮本次就熄灭，若上次熄灭本次就亮
    CLR     WDT_TIME_LED                ;上次熄灭本次点亮看门狗溢出时间指示灯
    CPL     Last_WDT_Time_LED_Status    ;将看门狗溢出时间指示灯的上一次状态位取反

WAIT2:
    SJMP    WAIT2                       ;循环执行本语句(停机)，等待看门狗溢出复位
```

```
Power_Off_WDT_TIME_LED:
    SETB        WDT_TIME_LED                ;上次亮本次就熄灭看门狗溢出时间指示灯
    CPL         Last_WDT_Time_LED_Status    ;将看门狗溢出时间指示灯的上一次状态位取反

WAIT3:
    SJMP        WAIT3                       ;循环执行本语句(停机)，等待看门狗溢出复位

END
```

13.2.7 程序地址非法复位

如果程序指针 PC 指向的地址超过了有效程序空间的大小，就会引起程序地址非法复位。程序地址非法复位状态结束后，不影响特殊功能寄存器 IA_PCONTR 中 SWBS/IAP_CONTR.6 位的值，单片机将根据复位前 SWBS/IAP_CONTR.6 的值选择是从用户应用程序区启动，还是从系统 ISP 监控程序区启动。如果复位前 SWBS/IAP_CONTR.6 的值为 0，则单片机从用户应用程序区启动。反之，如果复位前 SWBS/IAP_CONTR.6 的值为 1，则单片机从系统 ISP 监控程序区启动。程序地址非法复位是热启动复位中的软复位之一。

13.2.8 热启动复位和冷启动复位

		复位源		现象	复位后 SWBS/IAP_CONTR.6 的值
热启动复位	软复位	软件复位 (通过控制 IAP_CONTR.6 特殊功能寄存器的其中两位 SWBS/SWRST 实现复位)	通过对 IAP_CONTR.6 寄存器送入 20H 产生的软复位	会使系统从用户应用程序区 0000H 处开始执行用户程序	0
			通过对 IAP_CONTR.6 寄存器送入 60H 产生的软复位	会使系统从系统 ISP 监控程序区开始执行程序，检测不到合法的 ISP 下载命令流后，或检测到合法的 ISP 下载命令流并下载完用户程序后，均会软复位到用户应用程序区执行用户程序	1
		看门狗复位 (由 MCU/CPU 不在规定的时间内按要求访问看门狗所引起的复位) (不影响 SWBS/IAP_CONTR.6 的值)	复位前 SWBS/IAP_CONTR.6 的值为 0	会使系统从用户应用程序区 0000H 处开始执行用户程序	0
			复位前 SWBS/IAP_CONTR.6 的值为 1	会使系统从系统 ISP 监控程序区开始执行程序，检测不到合法的 ISP 下载命令流后，或检测到合法的 ISP 下载命令流并下载完用户程序后，均会软复位到用户应用程序区执行用户程序	1
		程序地址非法复位 (由程序指针 PC 指向的地址超过有效程序空间的大小所引起的复位) (不影响 SWBS/IAP_CONTR.6 的值)	复位前 SWBS/IAP_CONTR.6 的值为 0	会使系统从用户应用程序区 0000H 处开始执行用户程序	0
			复位前 SWBS/IAP_CONTR.6 的值为 1	会使系统从系统 ISP 监控程序区开始执行程序，检测不到合法的 ISP 下载命令流后，或检测到合法的 ISP 下载命令流并下载完用户程序后，均会软复位到用户应用程序区执行用户程序	1
	硬复位	内部低压检测复位 (当用户在 ISP 编程时允许低压检测复位并且电源电压 Vcc 在上电复位门槛电压以上时，由电源电压 Vcc 低于内部低压检测门槛电压所产生的复位) (不影响 SWBS/IAP_CONTR.6 的值)	复位前 SWBS/IAP_CONTR.6 的值为 0	会使系统从用户应用程序区 0000H 处开始执行用户程序	0
			复位前 SWBS/IAP_CONTR.6 的值为 1	会使系统从系统 ISP 监控程序区开始执行程序，检测不到合法的 ISP 下载命令流后，或检测到合法的 ISP 下载命令流并下载完用户程序后，均会软复位到用户应用程序区执行用户程序	1
		外部 RST 引脚复位 (通过从外部向 RST 引脚施加一定宽度的复位脉冲所产生的复位)		会将特殊功能寄存器 IAP_CONTR 中的 SWBS/IAP_CONTR.6 位置 1，同时会使系统从系统 ISP 监控程序区开始执行程序，检测不到合法的 ISP 下载命令流后，或检测到合法的 ISP 下载命令流并下载完用户程序后，均会软复位到用户应用程序区执行用户程序	1
冷启动复位		冷启动复位即掉电复位/上电复位 系统停电后再上电引起的复位 当电源电压 Vcc 低于掉电复位检测门槛电压时，就会引起系统复位，此复位称为掉电复位。如果电源电压 Vcc 再次上升至上电复位检测门槛电压(与掉电复位检测门槛电压相等)以上时，系统仍处于复位状态，此时称为上电复位。上电复位延时 32768 个时钟后，复位状态才会结束。如果用户在 ISP 编程时选择了 180ms 的长复位延时，则上电复位后将产生约 180ms 复位延时，复位状态才结束。 (对于 5V 单片机，目前掉电复位/上电复位检测门槛电压约为 3.2V; 对于 3.3V 单片机，目前掉电复位/上电复位检测门槛电压约为 1.8V)		会将特殊功能寄存器 IAPCONTR 中的 SWBS/IAP_CONTR.6 位置 1,同时会使系统从系统 ISP 监控程序区开始执行程序，检测不到合法的 ISP 下载命令流后，或检测到合法的 ISP 下载命令流并下载完用户程序后，均会软复位到用户应用程序区执行用户程序	1

IAP_CONTR：ISP/IAP 控制寄存器

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CONTR	C7H	name	IAPEN	SWBS	SWRST	CMD_FAIL	-	WT2	WT1	WT0

SWBS： 软件选择复位后从用户应用程序区启动(送 0)，还是从系统 ISP 监控程序区启动(送 1)。
要与 SWRST 直接配合才可以实现

SWRST：0：不操作；1：软件控制产生复位，单片机自动复位。

13.3 STC15 系列单片机的省电模式

STC15 系列单片机可以运行 3 种省电模式以降低功耗，它们分别是：低速模式，空闲模式和掉电模式。正常工作模式下，STC15 系列单片机的典型功耗是 2.7mA ~ 7mA，而掉电模式下的典型功耗是<0.1uA，空闲模式下的典型功耗是 1.8mA.

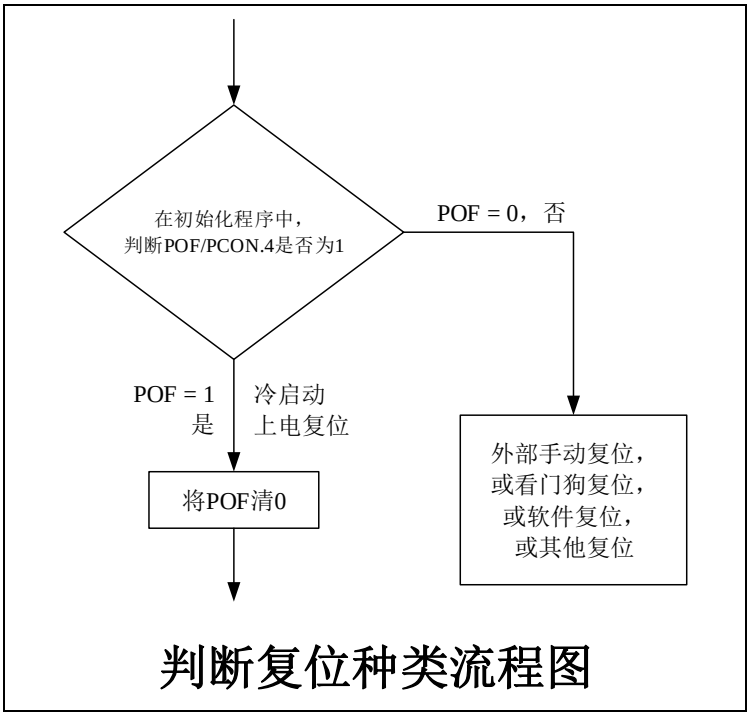
低速模式由时钟分频器 CLK_DIV（PCON2）控制，而空闲模式和掉电模式的进入由电源控制寄存器 PCON 的相应位控制。PCON 寄存器定义如下：

PCON(Power Control Register)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
PCON	87H	name	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL

LVDF： 低压检测标志位，同时也是低压检测中断请求标志位。
在正常工作和空闲工作状态时，如果内部工作电压 Vcc 低于低压检测门槛电压，该位自动置 1，与低压检测中断是否被允许无关。即在内部工作电压 Vcc 低于低压检测门槛电压时，不管有没有允许低压检测中断，该位都自动为 1。该位要用软件清 0，清 0 后，如内部工作电压 Vcc 继续低于低压检测门槛电压，该位又被自动设置为 1。
在进入掉电工作状态前，如果低压检测电路未被允许可产生中断，则在进入掉电模式后，该低压检测电路不工作以降低功耗。如果被允许可产生低压检测中断，则在进入掉电模式后，该低压检测电路继续工作，在内部工作电压 Vcc 低于低压检测门槛电压后，产生低压检测中断，可将 MCU 从掉电状态唤醒。

POF： 上电复位标志位，单片机停电后，上电复位标志位为 1，可由软件清 0。
实际应用:要判断是上电复位(冷启动)，还是外部复位脚输入复位信号产生的复位，还是内部看门狗复位，还是软件复位或者其他复位，可通过如下方法来判断:



判断复位种类流程图

PD： 将其置 1 时，进入 Power Down 模式，可由外部中断上升沿触发或下降沿触发唤醒，进入掉电模式时，内部时钟停振，由于无时钟，所以 CPU、定时器等功能部件停止工作，只有外部中断继续工作。可将 CPU 从掉电模式唤醒的外部管脚有：INT0/P3.2，INT1/P3.3，INT2/P3.6，INT3/P3.7，

INT4/P3.0；管脚 CCP0/CCP1/CCP2；管脚 RxD/RxD2/RxD3/RxD4；管脚 T0/T1/T2/T3/T4；有些单片机还具有内部低功耗掉电唤醒专用定时器。掉电模式也叫停机模式，此时功耗<0.1uA。

IDL：将其置 1，进入 IDLE 模式(空闲)，除系统不给 CPU 供时钟，CPU 不执行指令外，其余功能部件仍可继续工作，可由外部中断、定时器中断、低压检测中断及 A/D 转换中断中的任何一个中断唤醒。

GF1, GF0： 两个通用工作标志位，用户可以任意使用。

SMOD, SMOD0： 与电源控制无关，与串口有关，在此不作介绍。

13.3.1 低速模式及其测试程序(C 和汇编)

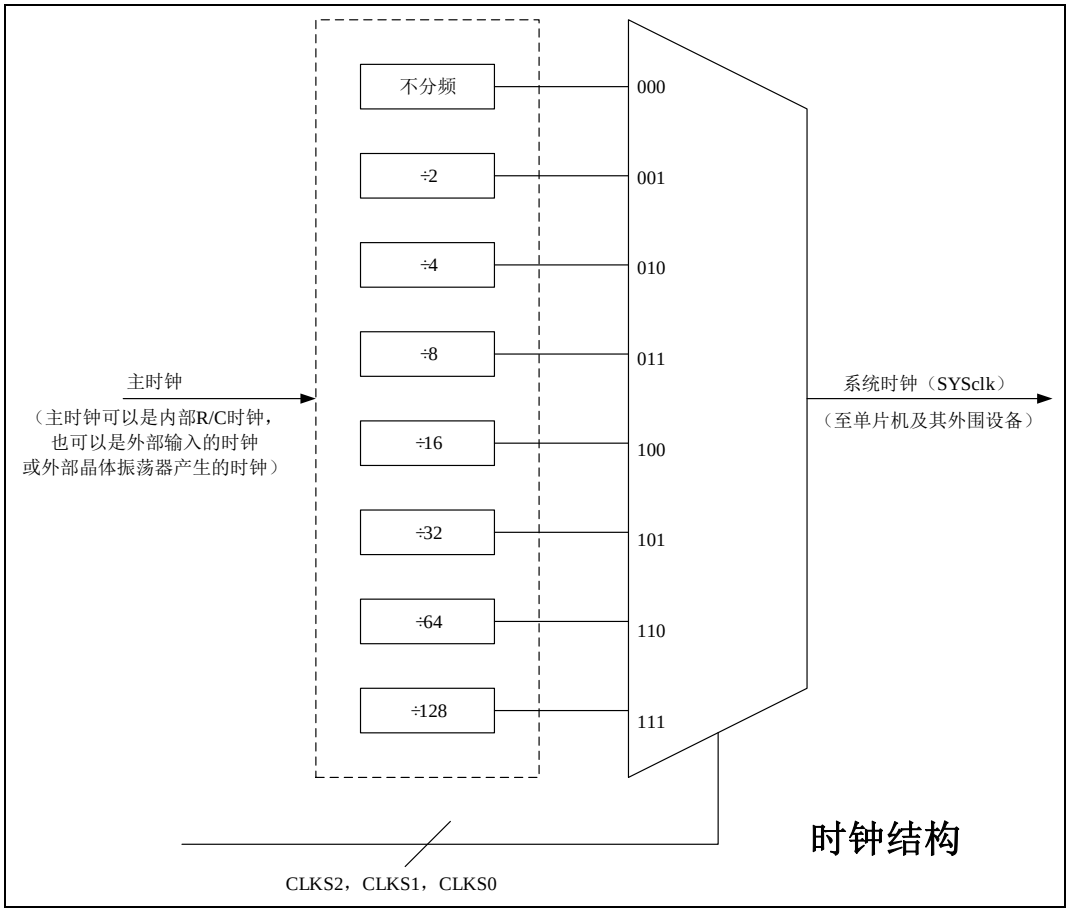
时钟分频器可以对内部时钟进行分频，从而降低工作时钟频率，降低功耗，降低 EMI。

时钟分频寄存器 CLK_DIV（PCON2）各位的定义如下：

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CLK-DIV (PCON2)	97H	name	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0

CLKS2	CLKS1	CLKS0	系统时钟选择控制位 (系统时钟是指对主时钟进行分频后供给 CPU、串行口、SPI、定时器、CCP/PWM/PCA、A/D 转换的实际工作时钟)
0	0	0	主时钟频率/1,不分频
0	0	1	主时钟频率/2
0	1	0	主时钟频率/4
0	1	1	主时钟频率/8
1	0	0	主时钟频率/16
1	0	1	主时钟频率/32
1	1	0	主时钟频率/64
1	1	1	主时钟频率/128

主时钟对外输出管脚 P5.4/MCLKO 或 P1.6/XTAL2/MCLKO_2 既可对外输出内部 R/C 时钟，也可对外输出外部输入的时钟或外部晶体振荡产生的时钟。



1.C 程序

```
/*---- STC15F2K60S2 系列 低速模式举例------------------------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
//假定测试芯片的工作频率为 18.432MHz
sfr   CLK_DIV = 0x97;            //时钟分频寄存器
//-----------------------------------------------
void main()
{
      CLK_DIV = 0x00;            //系统时钟为主时钟
//    CLK_DIV = 0x01;            //系统时钟为主时钟/2
//    CLK_DIV = 0x02;            //系统时钟为主时钟/4
//    CLK_DIV = 0x03;            //系统时钟为主时钟/8
//    CLK_DIV = 0x04;            //系统时钟为主时钟/16
//    CLK_DIV = 0x05;            //系统时钟为主时钟/32
//    CLK_DIV = 0x06;            //系统时钟为主时钟/64
//    CLK_DIV = 0x07;            //系统时钟为主时钟/128
      while (1);                 //程序终止
}
```

2. 汇编程序：

```
/*---- STC15F2K60S2 系列 低速模式举例------------------------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-----*/
;假定测试芯片的工作频率为 18.432MHz
CLK_DIV         DATA 097H           ;时钟分频寄存器
;-----------------------------------------
      ORG       0000H
      LJMP      START               ;复位入口
;-----------------------------------------
      ORG       0100H
START:
      MOV       SP,  #3FH
      MOV       CLK_DIV, #0         ;系统时钟为主时钟
;     MOV       CLK_DIV, #1         ;系统时钟为主时钟/2
;     MOV       CLK_DIV, #2         ;系统时钟为主时钟/4
;     MOV       CLK_DIV, #3         ;系统时钟为主时钟/8
;     MOV       CLK_DIV, #4         ;系统时钟为主时钟/16
;     MOV       CLK_DIV, #5         ;系统时钟为主时钟/32
;     MOV       CLK_DIV, #6         ;系统时钟为主时钟/64
;     MOV       CLK_DIV, #7         ;系统时钟为主时钟/128
      SJMP      $                   ;程序终止

      END
```

13.3.2 空闲模式(功耗<1mA)及其测试程序(C 和汇编)

将 IDL/PCON.0 置为 1，单片机将进入 IDLE（空闲）模式。在空闲模式下，仅 CPU 无时钟停止工作，但是外部中断、内部低压检测电路、定时器、A/D 转换等仍正常运行。而看门狗在空闲模式下是否工作取决于其自身有一个“IDLE”模式位：IDLE_WDT（WDT_CONTR.3）。

- 当 IDLE_WDT 位被设置为“1”时，看门狗定时器在“空闲模式”计数，即正常工作。
- 当 IDLE_WDT 位被清“0”时，看门狗定时器在“空闲模式”时不计数，即停止工作。

在空闲模式下，RAM、堆栈指针（SP）、程序计数器（PC）、程序状态字（PSW）、累加器（A）等寄存器都保持原有数据。I/O 口保持着空闲模式被激活前那一刻的逻辑状态。空闲模式下单片机的所有外围设备都能正常运行（除 CPU 无时钟不工作外）。当任何一个中断产生时，它们都可以将单片机唤醒，单片机被唤醒后，CPU 将继续执行进入空闲模式语句的下一条指令。

有两种方式可以退出空闲模式。任何一个中断的产生都会引起 IDL/PCON.0 被硬件清除，从而退出空闲模式。另一个退出空闲模式的方法是：外部 RST 引脚复位，将复位脚拉高，产生复位。这种拉高复位引脚来产生复位的信号源需要被保持 24 个时钟加上 20us，才能产生复位，再将 RST 引脚拉低，结束复位，单片机从系统 ISP 监控程序区开始启动。

1.C 程序:

```
/*---- STC15F2K60S2 系列空闲模式举例------------------------------------------------------*/
/*----  在 Keil C  开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------------*/
//特别注意：在将进入空闲模式时一定要加入 1-4 条_nop_()语句（空语句），即一定要在设置 MCU 进入
//空闲模式的语句后加 1-4 条_nop_()语句（空语句），如本程序中所示。
//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"
#include "intrins.h"
//-----------------------------------------
void main()
{
    while (1)
    {
        PCON |= 0x01;           //将 IDL(PCON.0)置 1,MCU 将进入空闲模式
        _nop_();                //此时 CPU 无时钟，不执行指令
        _nop_();                //内部中断信号和外部复位信号可以终止空闲模式
        _nop_();
        _nop_();
    }
}
```

2. 汇编程序：

```
/*-------------------------------------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列空闲模式举例------------------------------------------------------*/
```

```
/*---- 在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------------*/
;特别注意：在将进入空闲模式时一定要加入 1-4 条_nop_()语句（空语句），即一定要在设置 MCU 进入
;空闲模式的语句后加 1-4 条_nop_()语句（空语句），如本程序中所示。
;假定测试芯片的工作频率为 18.432MHz
;-----------------------------------------------
    ORG         0000H
    LJMP        START                   ;复位入口
;-----------------------------------------
    ORG         0100H
START:
    MOV         SP, #3FH

LOOP:
    MOV         PCON, #01H              ;将 IDL(PCON.0)置 1，MCU 将进入空闲模式
    NOP                                 ;此时 CPU 无时钟,不执行指令
    NOP                                 ;内部中断信号和外部复位信号可以终止空闲模式
    NOP
    NOP

    JMP         LOOP
;-------------------------------------------
END
```

13.3.3 掉电模式/停机模式及其测试程序(C 和汇编)

将 PD/PCON.1 置为 1，单片机将进入 Power Down（掉电）模式，掉电模式也叫停机模式。进入掉电模式/停机模式后，单片机所使用的时钟（内部系统时钟或外部晶体/时钟）停振，由于无时钟源，CPU、看门狗、定时器、串行口、A/D 转换等功能模块停止工作，外部中断（INT0/INT1/INT2/INT3/INT4）、CCP 继续工作。如果低压检测电路被允许可产生中断，则低压检测电路也可继续工作，否则将停止工作。进入掉电模式/停机模式后，所有 I/O 口、SFRs（特殊功能寄存器）维持进入掉电模式/停机模式前那一刻的状态不变。如果掉电唤醒专用定时器在进入掉电模式之前被打开（即在进入掉电模式/停机模式之前 WKTEN/WKTCH.7=1），则进入掉电模式/停机模式后，掉电唤醒专用定时器将开始工作。

进入掉电模式/停机模式后，STC15W4K32S4 系列单片机中可将掉电模式/停机模式唤醒的管脚资源有：

- INT0/P3.2，INT1/P3.3（INT0/INT1 上升沿下降沿中断均可），INT2/P3.6，INT3/P3.7，INT4/P3.0（INT2/INT3/INT4 仅可下降沿中断）；
- 管脚 CCP0/CCP1/CCP2；
- 管脚 RxD/RxD2/RxD3/RxD4；
- 管脚 T0/T1/T2/T3/T4（下降沿即外部管脚 T0/T1/T2/T3/T4 由高到低的变化，前提是在进入掉电模式/停机模式前相应的定时器中断已经被允许）；
- 低压检测中断（前提是低压检测中断被允许即 ELVD/IE.6 被置 1，且在 AIapp-ISP 编程/烧录用户应用程序时不选择“允许低压复位/禁止低压中断”）；
- 内部低功耗掉电唤醒专用定时器。

STC15 系列单片机的内部低功耗掉电唤醒专用定时器由特殊功能寄存器 WKTCH 和 WKTCL 进行管理和控制。

WKTCL（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
WKTCL	AAH	name									1111 1111B

WKTCH(不可位寻址)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
WKTCH	ABH	name	WKTEN								0111 1111B

内部掉电唤醒定时器是一个 15 位定时器，{WKTCH[6:0], WKTCL[7:0]}构成最长 15 位计数值(32768 个)定时从 0 开始计数。

WKTEN： 内部停机唤醒定时器的使能控制位。
WKTEN=1，允许内部停机唤醒定时器
WKTEN=0，禁止内部停机唤醒定时器

STC15 系列有内部低功耗掉电唤醒专用定时器的单片机除增加了特殊功能寄存器 WKTCL 和 WKTCH，还设计了了 2 个隐藏的特殊功能寄存器 WKTCL_CNT 和 WKTCH_CNT 来控制内部掉电唤醒专用定时器。

WKTCL_CNT 与 WKTCL 共用同一个地址，WKTCH_CNT 与 WKTCH 共用同一个地址，WKTCL_CNT 和 WKTCH_CNT 是隐藏的，对用户不可见。

WKTCL_CNT 和 WKTCH_CNT 实际上是作计数器使用，而 WKTCL 和 WKTCH 实际上作比较器使用。

当用户对 WKTCL 和 WKTCH 写入内容时，该内容只写入寄存器 WKTCL 和 WKTCH 中，而不会写入 WKTCL_CNT 和 WKTCH_CNT 中。

当用户读寄存器 WKTCL 和 WKTCH 中的内容时，实际上读的是寄存器 WKTCL_CNT 和 WKTCH_CNT 中的内容，而不是 WKTCL 和 WKTCH 中的内容。

特殊功能寄存器 WKTCL_CNT 和 WKTCH_CNT 的格式如下所示：

WKTCL_CNT

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
WKTCL_CNT	AAH	name									1111 1111B

WKTCH_CNT

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
WKTCH_CNT	ABH	name	-								x111 1111B

如果 STC15 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/WKTCH.7 位置‘1’，就可以打开内部掉电唤醒专用定时器），当 MCU 进入掉电模式/停机模式后，掉电唤醒专用定时器开始工作，MCU 可由该掉电唤醒专用定时器唤醒。

掉电唤醒专用定时器将 MCU 从掉电模式/停机模式唤醒的执行过程是：

- 一旦 MCU 进入掉电模式/停机模式，内部掉电唤醒专用定时器[WKTCH_CNT，WKTCL_CNT]就从 7FFFH 开始计数，直到计数到与{WKTCH[6:0]，WKTCL[7:0]}寄存器所设定的计数值相等

后就让系统时钟开始振荡；

- 如果主时钟使用的是内部系统时钟（由用户在 ISP 烧录程序时自行设置），MCU 在等待 64 个时钟后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态，就将时钟供给 CPU、定时器、看门狗、A/D 转换等功能模块工作；
- 如果主时钟使用的是外部晶体或时钟（由用户在 ISP 烧录程序时自行设置），MCU 在等待 1024 个时钟后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态，就将时钟供给 CPU、定时器、看门狗、A/D 转换等功能模块工作；
- CPU 获得时钟后，程序从上次设置单片机进入掉电模式语句的下一条语句开始往下执行。

掉电唤醒之后，WKTCH_CNT 和 WKTCL_CNT 的内容保持不变，因此可以通过读[WKTCH，WKTCL]的内容（实际上是读[WKTCH_CNT，WKTCL_CNT]的内容）读出单片机在停机模式/掉电模式所等待的时间。

这里请注意：用户在设置寄存器{WKTCH[6:0]，WKTCL[7:0]}的计数值时，要按照所需要的计数次数，在计数次数的基础上减 1 所得的数值才是{WKTCH，WKTCL}的计数值。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 32768 次，则应对{WKTCH[6:0]，WKTCL[7:0]}写入 7FFFH（32767）。

内部掉电唤醒定时器有自己的内部时钟，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。内部掉电唤醒定时器的时钟频率约为 32768Hz，当然误差较大。

- 对于 16-pin 及其以上的单片机，用户可以通过读 RAM 区 F8 单元和 F9 单元的内容来获取内部掉电唤醒专用定时器常温下的时钟频率。
- 对于 8-pin 单片机即 STC15F104W 系列，用户可以通过读 RAM 区 78 单元和 79 单元的内容来获取内部掉电唤醒专用定时器常温下的时钟频率。

下面以 16-pin 及其以上的单片机为例，介绍如何计算内部掉电唤醒专用定时器的计数时间。

假设我们用[WIRC_H，WIRC_L]来表示从 RAM 区 F8 单元和 F9 单元获取到的内部掉电唤醒专用定时器常温下的时钟频率，则内部掉电唤醒专用定时器计数时间按下式计算:

$$\text{内部掉电唤醒专用定时器计数时间} = \frac{10^6\text{us}}{[\text{WIRC_H},\text{WIRC_L}]} \times 16 \times \text{计数次数}$$

例如：假设读到 RAM 区 F8 单元的内容为 80H，F9 单元的内容为 00H，即内部掉电唤醒定时器的时钟频率[WIRC_H，WIRC_L]为 32768Hz，则内部掉电唤醒专用定时器最短计数时间（即计数一次的时间）为：

$$= \frac{10^6\text{us}}{32768} \times 16 \times 1 \approx 488.28\text{us}$$

内部掉电唤醒专用定时器最长计数时间约为 488.28us×32768 = 16s

设定{WKTCH[6:0]，WKTCL[7:0]}寄存器的值等于 9（即计数 10 次）且内部掉电唤醒定时器的时钟频率[WIRC_H，WIRC_L]为 32768Hz，则从系统掉电到启动系统振荡器，所需要等待的时间为 488.28us×10 = 4882.8us

设定{WKTCH[6:0]，WKTCL[7:0]}寄存器的值等于 32767（即最大计数值=32768=2^{15}）且内部掉电唤醒定时器的时钟频率[WIRC_H，WIRC_L]为 32768Hz，则从系统掉电到启动系统振荡器，所需要等待的时间为 488.28us×32768 = 16s

下面给出了在读到 RAM 区 F8 单元的内容为 80H，F9 单元的内容为 00H，即内部掉电唤醒定时器

的时钟频率[WIRC_H，WIRC_L]为 32768Hz 情况下，内部掉电唤醒专用定时器的计数时间：

{WKTCH[6:0]，WKTCL[7:0]} = 0，	488.28us×1	= 488.28us
{WKTCH[6:0]，WKTCL[7:0]} = 9，	488.28us×10	= 4.8828ms
{WKTCH[6:0]，WKTCL[7:0]} = 99，	488.28us×100	= 48.828ms
{WKTCH[6:0]，WKTCL[7:0]} = 999，	488.28us×1000	= 488.28ms
{WKTCH[6:0]，WKTCL[7:0]} = 4095，	488.28us×4096	= 2.0s
{WKTCH[6:0]，WKTCL[7:0]} = 32767，	488.28u8×32768	= 16s

为了降低功耗，未制作掉电唤醒定时器的抗误差和抗温漂的电路，因此，掉电唤醒定时器制造误差较大，压漂（电压抖动）较大。

除掉电唤醒专用定时器外，还可将掉电模式/停机模式唤醒的中断有：INT0/P3.2，INT1/P3.3（INT0/INT1 上升沿下降沿中断均可），INT2/P3.6，INT3/P3.7，INT4/P3.0（INT2/INT3/INT4 仅可下降沿中断）；管脚 CCP（CCP 可以在 CCP0/P1.1，CCP1/P1.0，CCP2/CCP2_2/P3.7，CCP0_2/P3.5，CCP1_2/P3.6，CCP2/CCP2_2/P3.7，CCP0_3/P2.5，CCP1_3/P2.6，CCP2_3/P2.7 之间切换）。

如果掉电模式/停机模式是由外部中断 INT0（上升沿+下降沿中断）、INT1（上升沿+下降沿中断）、INT2（仅可下降沿中断）、INT3（仅可下降沿中断）、INT4（仅可下降沿中断）或 CCP 管脚唤醒，则掉电唤醒之后 CPU 首先执行设置单片机进入掉电模式的语句的下一条语句（建议在设置单片机进入掉电模式的语句后多加几个 NOP 空指令），然后执行相应的中断服务程序。

另外，在串行中断被允许后，串行口 1、串行口 2、串行口 3 和串行口 4 的接收管脚 RxD（可以在 RxD/P3.0，RxD_2/P3.6，RxD_3/P2.6 之间切换）、RxD2（可以在 RxD2/P1.0，RxD2_2/P4.6 之间切换）、RxD3（可以在 RxD3/P0.0，RxD3_2/P5.0 之间切换）和 RxD4（可以在 RxD4/P0.2，RxD4_2/P5.2 之间切换），如发生由高到低的变化时（起始位接收）也可以将 MCU 从掉电模式/停机模式唤醒。

当 MCU 由 RxD 或 RxD2 或 RxD3 或 RxD4 唤醒时：

- 如果主时钟使用的是内部系统时钟（由用户在 ISP 烧录程序时自行设置），MCU 在等待 64 个时钟后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态，就将时钟供给 CPU 工作；
- 如果主时钟使用的是外部晶体或时钟（由用户在 ISP 烧录程序时自行设置），MCU 在等待 1024 个时钟后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态，就将时钟供给 CPU 工作；

CPU 获得时钟后，程序从上次设置单片机进入掉电模式语句的下一条语句开始往下执行。

如果定时器 T0/T1/T2/T3/T4 的中断在进入掉电模式/停机模式前被允许了，即进入掉电模式/停机模式前 ET0/ET1/ET2/ET3/ET4 及 EA 已经被设置为 1，则进入掉电模式/停机模式后，定时器 T0/T1/T2/T3/T4 的外部管脚（T0/P3.4，T1/P3.5，T2/P3.1，T3/P0.5，T4/P0.7）如发生由高到低的变化可以将 MCU 从掉电模式/停机模式唤醒。当 MCU 由定时器 T0/T1/T2/T3/T4 的外部管脚由高到低的变化唤醒时，

- 如果主时钟使用的是内部系统时钟（由用户在 ISP 烧录程序时自行设置），MCU 在等待 64 个时钟后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态，就将时钟供给 CPU 工作；
- 如果主时钟使用的是外部晶体或时钟（由用户在 ISP 烧录程序时自行设置），MCU 在等待 1024 个时钟后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态就将时钟供给 CPU 工作；

CPU 获得时钟后，程序从上次设置单片机进入掉电模式语句的下一条语句开始往下执行，不进入相应定时器的中断程序。

还有外部 RST 引脚复位也可将 MCU 从掉电模式/停机模式中唤醒，复位唤醒后的 MCU 将从系统 ISP

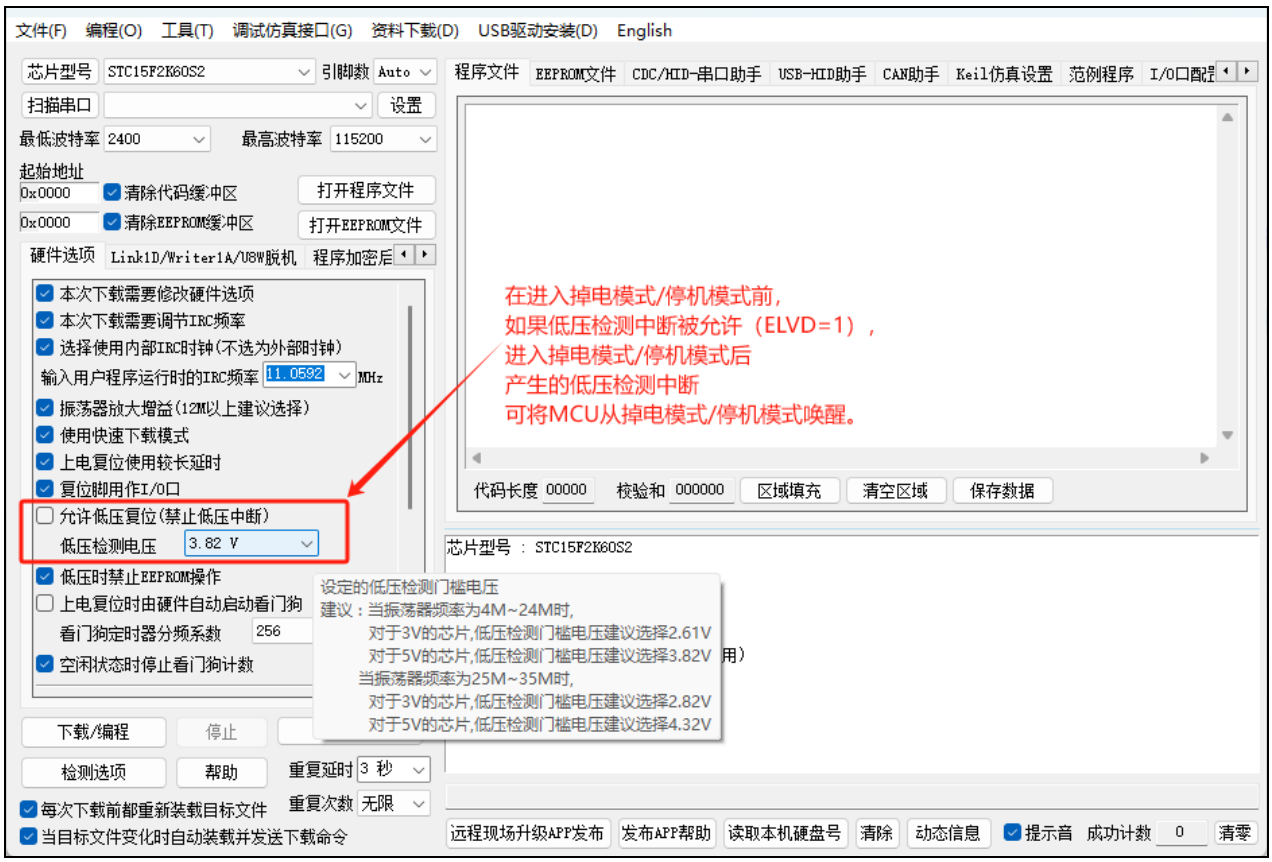
监控程序区开始启动。

【特别声明】: 以 15L 开头的芯片如需进入“掉电模式”，进入“掉电模式”前必须启动掉电唤醒定时器<3uA>，不超过 1 秒要唤醒一次，以 15F 和 15W 开头的芯片以及新供货的 STC15L2K60S2 系列 D 版本芯片则不需要。

如果在 AIapp-ISP 编程/烧录用户应用程序时，不将低压检测设置为低压检测复位，则在用户程序中用户可将低压检测设置为低压检测中断。当电源电压 VCC 低于内部低压检测（LVD）门槛电压时，低压检测中断请求标志位（LVDF/PCON.5）就会被硬件置位。如果 ELVD/IE.6（低压检测中断允许位）被设置为 1，低压检测中断请求标志位就能产生一个低压检测中断。

在进入掉电模式/停机模式前，如果低压检测电路未被允许可产生中断，则在进入掉电模式/停机模式后，该低压检测电路不工作以降低功耗。如果被允许可产生低压检测中断（相应的中断允许位是 ELVD/IE.6，中断请求标志位是 LVDF/PCON.5），则在进入掉电模式/停机模式后，该低压检测电路继续工作，在内部工作电压 Vcc 低于低压检测门槛电压后，产生低压检测中断，可将 MCU 从掉电模式/停机模式唤醒。

建议在电压偏低时，不要操作 EEPROM/IAP，烧录时直接选择“低压禁止 EEPROM 操作”



注意：

现供货的 STC15F2K60S2 系列 C 版本单片机的 RxD 管脚和 RxD2 管脚暂时不能将掉电模式/停机模式唤醒！STC15F2K60S2 及 STC15L2K60S2 系列下一升级版本----STC15W2K60S2 系列单片机将会设计实现该计划功能，STC15W2K60S2 系列单片机的 RxD 管脚和 RxD2 管脚将均可用于唤醒掉电模式/停机模式。同时，STC15W2K60S2 系列单片机还将增加比较器的功能。

现供货的 STC15F408AD 系列 C 版本单片机的 RxD 管脚暂时也不能将掉电模式/停机模式唤醒！

STC15F408AD 及 STC15L408AD 系列下一升级版本----STC15W408AS 系列单片机将会设计实现该计划功能，STC15W408AS 系列单片机的 RxD 管脚将可用于唤醒掉电模式/停机模式。

现供货的 STC15W4K32S4 系列 A 版本单片机掉电模式<0.4uA，[T3/P0.5，T4/P0.7]在掉电模式时不要作掉电唤醒，与 PWM2 到 PWM7 相关的 12 个口[P3.7/PWM2，P2.1/PWM3，P2.2/PWM4，P2.3/PWM5，P1.6/PWM6，P1.7/PWM7，P2.7/PWM2_2，P4.5/PWM3_2，P4.4/PWM4_2，P4.2/PWM5_2，P0.7/PWM6_2，P0.6/PWM7_2]，上电复位后是高阻输入，在进入掉电模式前要软件将其改设为强推挽输出或准双向口上拉模式。

13.3.3.1 掉电模式/停机模式被唤醒后程序执行流程说明及测试程序（C 和汇编）

当 STC15W4K32S4 系列单片机内置掉电唤醒专用定时器被允许（WKTEN=1），掉电唤醒专用定时器将 MCU 从掉电模式/停机模式唤醒的执行过程是：

一旦 MCU 进入掉电模式/停机模式，内部掉电唤醒专用定时器[WKTCH_CNT，WKTCL_CNT]就从 7FFFH 开始计数，直到计数到与{WKTCH[6:0]，WKTCL[7:0]}寄存器所设定的计数值相等后就让系统时钟开始振荡；

- 如果主时钟使用的是内部系统时钟（由用户在 ISP 烧录程序时自行设置），MCU 在等待 64 个时钟后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态，就将时钟供给 CPU 工作；
- 如果主时钟使用的是外部晶体或时钟（由用户在 ISP 烧录程序时自行设置），MCU 在等待 1024 个时钟后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态，就将时钟供给 CPU 工作；

CPU 获得时钟后，程序从上次设置单片机进入掉电模式语句的下一条语句开始往下执行。

掉电模式/停机模式由中断 INT0/P3.2，INT1/P3.3（INT0/INT1 上升沿下降沿中断均可），INT2/P3.6，INT3/P3.7，INT4/P3.0（INT2/INT3/INT4 仅可下降沿中断）；管脚 CCP（可以在 CCP0/P1.1，CCP1/P1.0，CCP2/P3.7，CCP0_2/P3.5，CCP1_2/P3.6，CCP2_2/P3.7，CCP0_3/P2.5，CCP1_3P2.6，CCP2_3/P2.7 之间切换）唤醒之后程序的执行流程为：CPU 首先执行从上次设置单片机进入掉电模式语句的下一条语句（建议在设置单片机进入掉电模式的语句后多加几个 NOP 空指令）然后执行相应的中断服务程序。

掉电模式/停机模式由串行口 1、串行口 2、串行口 3 和串行口 4 的接收管脚 RxD（可以在 RxD/P3.0，RxD_2/P3.6，RxD_3/P2.6 之间切换）、RxD2（可以在 RxD2/P1.0，RxD2_2/P4.6 之间切换）、RxD3（可以在 RxD3/P0.0，RxD3_2/P5.0 之间切换）和 RxD4（可以在 RxD4/P0.2，RxD4_2/P5.2 之间切换）的下降沿（不产生中断）唤醒后的程序执行流程：

当 MCU 由 RxD 的下降沿或 RxD2 的下降沿或 RxD3 的下降沿或 RxD4 的下降沿唤醒后，

- 如果主时钟使用的是内部系统时钟，MCU 在等待 64 个时钟（由用户在 ISP 烧录程序时自行设置）后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态，才将时钟供给 CPU 工作；
- 如果主时钟使用的是外部晶体或时钟，MCU 在等待 1024 个时钟（由用户在 ISP 烧录程序时自行设置）后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态，才将时钟供给 CPU 工作；

CPU 获得时钟后，程序从上次设置单片机进入掉电模式语句的下一条语句开始往下执行。

掉电模式/停机模式由定时器 T0/T1/T2/T3/T4 的外部管脚的下降沿（不产生中断）唤醒后的程序执行流程：

当 MCU 由定时器 T0/T1/T2/T3/T4 的外部管脚的下降沿唤醒后，

- 如果主时钟使用的是内部系统时钟，MCU 在等待 64 个时钟（由用户在 ISP 烧录程序时自行设置）后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态，才将时钟供给 CPU 工作；
- 如果主时钟使用的是外部晶体或时钟，MCU 在等待 1024 个时钟（由用户在 ISP 烧录程序时自行设置）后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态，才将时钟供给 CPU 工作；

CPU 获得时钟后，程序从上次设置单片机进入掉电模式语句的下一条语句开始往下执行，不进入相应定时器的中断程序。

1. C 程序：

```
/*-------------------------------------------------------------------------------------------------*/
/*----STC15W4K60S4 系列掉电模式中指令执行流程说明----------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可------*/
/*-------------------------------------------------------------------------------------------------*/

//特别注意：在将进入掉电模式时一定要加入 2--4 条_nop_()语句(空语句)，即一定要在设置 MCU 进入
//掉电模式的语句后加 2--4 条_nop_()语句(空语句)，如本程序中所示。

//假定测试芯片的工作频率为 18.432MHz

#include "reg51.h"
#include "intrins.h"
//-----------------------------------------------
void main()
{
    while (1)
    {
        PCON |= 0x02;                   //将 STOP(PCON.1)置 1，MCU 将进入掉电模式
        _nop_();
        //当有有效的掉电唤醒源产生时，若使用的是内部振荡器，则立即启动内部振荡器，
        //在 64 个时钟周期后，将时钟提供给 MCU，作为系统时钟；若使用的是外部振荡器
        //则立即启动外部振荡器，在 1024 个时钟周期后，将时钟提供给 MCU，作为系统时钟。
        //在时钟信号到达 CPU 后，若掉电唤醒源是内部 32K 掉电唤醒定时器、RxD 和 RxD2 时
        //CPU 直接从此语句开始向下执行程序代码，而不产生中断；若掉电唤醒源是 INT0、
        //INT1、INT2、INT3、INT4、CCP0、CCP1、CCP2 时，则 CPU 首先执行此语句，
        //然后执行中断服务程序。

        _nop_();
        _nop_();
        _nop_();
    }
}
```

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列掉电模式中指令执行流程说明----------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

;特别注意：在将进入掉电模式时一定要加入 2--4 条 NOP 语句(空语句)，即一定要在设置 MCU 进入
;掉电模式的语句后加 2--4 条 NOP 语句(空语句)，如本程序中所示。

;假定测试芯片的工作频率为 18.432MHz
;-----------------------------------------
    ORG     0000H
    LJMP    START           ;复位入口
;-----------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH

LOOP:
    MOV     PCON, #02H      ;将 STOP(PCON.1)置 1，MCU 将进入掉电模式

    NOP     ;当有有效的掉电唤醒源产生时，若使用的是内部振荡器，则立即启动内部振荡器，
            ;在 64 个时钟周期后，将时钟提供给 MCU，作为系统时钟；若使用的是外部振荡器
            ;则立即启动外部振荡器，在 1024 个时钟周期后,将时钟提供给 MCU，作为系统时钟。
            ;在时钟信号到达 CPU 后，若掉电唤醒源是内部 32K 掉电唤醒定时器、RxD 和 RxD2
            ;时，CPU 直接从此语句开始向下执行程序代码，而不产生中断；若掉电唤醒源是 INT0、
            ;INT1、INT2、INT3、INT4、CCP0、CCP1、CCP2 时，则 CPU 首先执行此语句，
            ;然后执行中断服务程序。
    NOP
    NOP
    NOP
    JMP     LOOP
;-----------------------------------------
    END
```

13.3.3.2 用掉电唤醒专用定时器唤醒掉电模式/停机模式的测试程序(C 和汇编)

----以 15L 开头的单片机进入掉电模式/停机模式前必须启动掉电唤醒专用定时器

【特别声明】：以 15L 开头的芯片如需进入“掉电模式”，进入“掉电模式”前必须启动掉电唤醒定时器<3uA>，不超过 1 秒要唤醒一次，以 15F 和 15W 开头的芯片以及新供货的 STC15L2K60S2 系列 D 版本芯片则不需要

/*利用内部专用掉电唤醒定时器来唤醒掉电模式的示例程序(C 程序)

1.C 程序:

```
/*---------------------------------------------------------------------------------------------*/
/*------演示 STC15W4K60S4 系列掉电唤醒定时器举例---------------------------------*/
/*------在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可--------*/
/*---------------------------------------------------------------------------------------------*/

//特别注意：在将进入掉电模式时一定要加入 2--4 条_nop_()语句(空语句)，即一定要在设置 MCU 进入
//掉电模式的语句后加 2--4 条_nop_()语句(空语句)，如本程序中所示。
//假定测试芯片的工作频率为 18.432MHz

#include "reg51.h"
#include "intrins.h"
//-----------------------------------------
sfr     WKTCL = 0xaa;           //掉电唤醒定时器计时低字节
sfr     WKTCH = 0xab;           //掉电唤醒定时器计时高字节
sbit    P10 = P1^0;
//-----------------------------------------
void main()
{
    WKTCL = 49;                 //设置唤醒周期为 488us×(49+1) = 24.4ms
    WKTCH = 0x80;               //使能掉电唤醒定时器
    while (1)
    {
        PCON = 0x02;            //进入掉电模式
        _nop_();                //掉电模式被唤醒后，直接从此语句开始向下执行，
                                //不进入中断服务程序
        _nop_();
        P10 = !P10;             //掉电唤醒后，取反测试口
    }
}
```

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* ---  演示 STC15W4K60S4 系列掉电唤醒定时器举例---------------------------------*/
/*------在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可--------*/
/*---------------------------------------------------------------------------------------------*/

;特别注意：在将进入掉电模式时一定要加入 2--4 条 NOP 语句(空语句)，即一定要在设置 MCU 进入
;掉电模式的语句后加 2--4 条 NOP 语句(空语句)，如本程序中所示。
;假定测试芯片的工作频率为 18.432MHz

WKTCL       DATA    0AAH        ;掉电唤醒定时器计时低字节
WKTCH       DATA    0ABH        ;掉电唤醒定时器计时高字节
;---------------------------------------
```

```
    ORG     0000H
    LJMP    START               ;复位入口
;---------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    MOV     WKTCL, #49          ;设置唤醒周期为 488us×(49+1) = 24.4ms
    MOV     WKTCH,#80H          ;使能掉电唤醒定时器

LOOP:
    MOV     PCON, #02H          ;进入掉电模式
    NOP                         ;掉电模式被唤醒后，直接从此语句开始向下执行，
                                ;不进入中断服务程序
    NOP
    CPL     P1.0                ;掉电唤醒后，取反测试口
    JMP     LOOP
    SJMP    $
;---------------------------------------
    END
```

13.3.3.3 用外部中断 INT0(上升沿+下降沿)唤醒掉电模式/停机模式测试程序(C 和汇编)

1.C 程序:

```
/*-------------------------------------------------------------------------------------------------*/
/*------STC15W4K60S4 系列 INT0 唤醒掉电模式举例----------------------------------*/
/*------在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------------*/

//特别注意：在将进入掉电模式时一定要加入 2--4 条_nop_()语句(空语句)，即一定要在设置 MCU 进入
//掉电模式的语句后加 2--4 条_nop_()语句(空语句)，如本程序中所示。
//假定测试芯片的工作频率为 18.432MHz

#include "reg51.h"
#include "intrins.h"
//-----------------------------------------------
bit     FLAG;                   //1:上升沿中断  0:下降沿中断
sbit    P10 = P1^0;
//---------------------------------------
//中断服务程序
void exint0() interrupt 0       //INT0 中断入口
{
    P10 = !P10;                 //将测试口取反
    FLAG = INT0;                //保存 INT0 口的状态, INT0=0(下降沿); INT0=1(上升沿)
}
```

```
//-----------------------------------------------
void main()
{
    IT0 = 0;                    //设置 INT0 的中断类型为上升沿和下降沿,上升沿和下降沿均可唤醒
//  IT0 = 1;                    //设置 INT0 的中断类型为仅下降沿,下降沿唤醒
    EX0 = 1;                    //使能 INT0 中断
    EA = 1;
    while (1)
    {
        PCON = 0x02;            //MCU 进入掉电模式
        _nop_();                //掉电模式被唤醒后,首先执行此语句,然后再进入中断服务程序
        _nop_();
    }
}
```

2. 汇编程序：

```
/*-------------------------------------------------------------------------------------------------*/
/*------STC15W4K60S4 系列 INT0 唤醒掉电模式举例-----------------------------------*/
/*------在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------------*/

;特别注意：在将进入掉电模式时一定要加入 2--4 条 NOP 语句(空语句)，即一定要在设置 MCU 进入
;掉电模式的语句后加 2--4 条 NOP 语句(空语句)，如本程序中所示。
;假定测试芯片的工作频率为 18.432MHz

FLAG        BIT     20H.0   ;1:上升沿中断；0:下降沿中断
;---------------------------------------
    ORG     0000H
    LJMP    START           ;复位入口
    ORG     0003H           ;INT0 中断入口
    LJMP    EXINT0
;---------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    CLR     IT0             ;设置 INT0 的中断类型为上升沿和下降沿,上升沿和下降沿均可唤醒
;   SETB    IT0             ;设置 INT0 的中断类型为仅下降沿，下降沿唤醒
    SETB    EX0             ;使能 INT0 中断
    SETB    EA

LOOP:
    MOV     PCON, #02H      ;MCU 进入掉电模式
    NOP                     ;掉电模式被唤醒后,首先执行此语句,然后再进入中断服务程序
    NOP
```

```
    SJMP    LOOP
;---------------------------------------
;中断服务程序
EXINT0:
    CPL     P1.0            ;将测试口取反
    PUSH    PSW
    MOV     C, INT0         ;读取 INT0 口的状态
    MOV     FLAG, C         ;保存, INT0=0(下降沿); INT0=1(上升沿)
    POP     PSW
    RETI
;---------------------------------------
    END
```

13.3.3.4 用外部中断 INT1(上升沿+下降沿)唤醒掉电模式/停机模式的测试程序(C 和汇编)

1.C 程序:

```
/*-------------------------------------------------------------------------------------------------*/
/*------STC15W4K60S4 系列 INT1 唤醒掉电模式举例-----------------------------------*/
/*------在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------------*/

//特别注意：在将进入掉电模式时一定要加入 2--4 条_nop_()语句(空语句)，即一定要在设置 MCU 进入
//掉电模式的语句后加 2--4 条_nop_()语句(空语句)，如本程序中所示。
//假定测试芯片的工作频率为 18.432MHz

#include "reg51.h"
#include "intrins.h"
//-----------------------------------------------
bit     FLAG;               //1:上升沿中断  0:下降沿中断
sbit    P10 = P1^0;
//---------------------------------------
//中断服务程序
void exint1() interrupt 2       //INT1 中断入口
{
    P10 = !P10;                 //将测试口取反
    FLAG = INT1;                //保存 INT1 口的状态, INT1=0(下降沿); INT1=1(上升沿)
}
//-----------------------------------------------
void main()
{
    IT1 = 0;                    //设置 INT1 的中断类型为上升沿和下降沿,上升沿和下降沿均可唤醒
//  IT1 = 1;                    //设置 INT1 的中断类型为仅下降沿，下降沿唤醒
    EX1 = 1;                    //使能 INT1 中断
    EA = 1;
```

```
    while (1)
    {
        PCON = 0x02;        //MCU 进入掉电模式
        _nop_();            //掉电模式被唤醒后,首先执行此语句,然后再进入中断服务程序
        _nop_();
    }
}
```

2. 汇编程序：

```
/*-------------------------------------------------------------------------------------------------*/
/*------STC15W4K60S4 系列 INT1 唤醒掉电模式举例-----------------------------------*/
/*------在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------------*/

;特别注意：在将进入掉电模式时一定要加入 2--4 条 NOP 语句(空语句)，即一定要在设置 MCU 进入
;掉电模式的语句后加 2--4 条 NOP 语句(空语句)，如本程序中所示。
;假定测试芯片的工作频率为 18.432MHz

FLAG        BIT     20H.0   ;1:上升沿中断  0:下降沿中断
;---------------------------------------
    ORG     0000H
    LJMP    START           ;复位入口

    ORG     0013H           ;INT1 中断入口
    LJMP    EXINT1
;---------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    CLR     IT1             ;设置 INT1 的中断类型为上升沿和下降沿,上升沿和下降沿均可唤醒
;   SETB    IT1             ;设置 INT1 的中断类型为仅下降沿,下降沿唤醒
    SETB    EX1             ;使能 INT1 中断
    SETB    EA

LOOP:
    MOV     PCON, #02H      ;MCU 进入掉电模式
    NOP                     ;掉电模式被唤醒后,首先执行此语句,然后再进入中断服务程序
    NOP
    SJMP    LOOP
;---------------------------------------
EXINT1:
    CPL     P1.0            ;取反测试口
    PUSH    PSW
    MOV     C, INT1         ;读取 INT1 口的状态
    MOV     FLAG, C         ;保存, INT1=0(下降沿); INT0=1(上升沿)
```

```
    POP     PSW
    RETI
;----------------------------------------
    END
```

13.3.3.5 用外部中断 INT2(下降沿)唤醒掉电模式/停机模式的测试程序(C 和汇编)

1.C 程序:

```
/*-------------------------------------------------------------------------------------------------*/
/*------STC15W4K60S4 系列 INT2 下降沿唤醒掉电模式举例---------------------------*/
/*------在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------------*/

//特别注意：在将进入掉电模式时一定要加入 2--4 条_nop_()语句(空语句)，即一定要在设置 MCU 进入
//掉电模式的语句后加 2--4 条_nop_()语句(空语句)，如本程序中所示。
//假定测试芯片的工作频率为 18.432MHz

#include "reg51.h"
#include "intrins.h"
//-----------------------------------------------
sfr     INT_CLKO = 0x8F;        //外部中断与时钟输出控制寄存器
sbit    INT2 = P3^6;            //INT2 引脚定义
sbit    P10 = P1^0;
//-----------------------------------------
//中断服务程序
void exint2() interrupt 10
{
    P10 = !P10;                 //将测试口取反
//  INT_CLKO &= 0xEF;           //若需要手动清除中断标志,可先关闭中断,
                                //此时系统会自动清除内部的中断标志
//  INT_CLKO |= 0x10;           //然后再开中断即可
}
//-----------------------------------------------
void main()
{
    INT_CLKO |= 0x10;           //(EX2 = 1)使能 INT2 下降沿中断
    EA = 1;
    while (1)
    {
        PCON = 0x02;            //MCU 进入掉电模式
        _nop_();                //掉电模式被唤醒后,首先执行此语句,然后再进入中断服务程序
        _nop_();
    }
}
```

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/*------STC15W4K60S4 系列 INT2 下降沿唤醒掉电模式举例----------------------------------*/
/*------在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

;特别注意：在将进入掉电模式时一定要加入 2--4 条 NOP 语句(空语句)，即一定要在设置 MCU 进入
;掉电模式的语句后加 2--4 条 NOP 语句(空语句)，如本程序中所示。
;假定测试芯片的工作频率为 18.432MHz

INT_CLKO    DATA    08FH        ;外部中断与时钟输出控制寄存器
INT2        BIT     P3.6        ;INT2 引脚定义
;---------------------------------------
    ORG     0000H
    LJMP    START               ;复位入口
    ORG     0053H               ;INT2 中断入口

    LJMP    EXINT2
    ;---------------------------------------

    ORG     0100H

START:
    MOV     SP, #3FH
    ORL     INT_CLKO, #10H      ;(EX2 = 1)使能 INT2 下降沿中断
    SETB    EA

LOOP:
    MOV     PCON, #02H          ;MCU 进入掉电模式
    NOP                         ;掉电模式被唤醒后,首先执行此语句,然后再进入中断服务程序
    NOP
    SJMP    LOOP
;---------------------------------------
;中断服务程序
EXINT2:
    CPL     P1.0                ;将测试口取反
;   ANL     INT_CLKO, #0EFH     ;若需要手动清除中断标志,可先关闭中断,
                                ;此时系统会自动清除内部的中断标志
;   ORL     INT_CLKO, #10H      ;然后再开中断即可
    RETI
;---------------------------------------
    END
```

13.3.3.6 用外部中断 INT3(下降沿)唤醒掉电模式/停机模式的测试程序(C 和汇编)

1.C 程序:

```
/*---------------------------------------------------------------------------------------------*/
/*------STC15W4K60S4 系列 INT3 下降沿唤醒掉电模式举例---------------------------*/
/*------在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//特别注意：在将进入掉电模式时一定要加入 2--4 条_nop_()语句(空语句)，即一定要在设置 MCU 进入
//掉电模式的语句后加 2--4 条_nop_()语句(空语句)，如本程序中所示。
//假定测试芯片的工作频率为 18.432MHz

#include "reg51.h"
#include "intrins.h"
//-----------------------------------------------
sfr         INT_CLKO = 0x8F;      //外部中断与时钟输出控制寄存器

sbit        INT3 = P3^7;          //INT3 引脚定义

sbit        P10 = P1^0;
//-----------------------------------------
//中断服务程序
void exint3() interrupt 11
{
    P10 = !P10;                   //将测试口取反
//  INT_CLKO &= 0xDF;             //若需要手动清除中断标志,可先关闭中断,
                                  //此时系统会自动清除内部的中断标志
//  INT_CLKO |= 0x20;             //然后再开中断即可
}
//-----------------------------------------------
void main()
{
    INT_CLKO |= 0x20;             //(EX3 = 1)使能 INT3 下降沿中断
    EA = 1;
    while (1)
    {
        PCON = 0x02;              //MCU 进入掉电模式
        _nop_();                  //掉电模式被唤醒后,首先执行此语句,然后再进入中断服务程序
        _nop_();
    }
}
```

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/*------STC15W4K60S4 系列 INT3 下降沿唤醒掉电模式举例----------------------------------*/
/*------在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

;特别注意：在将进入掉电模式时一定要加入 2--4 条 NOP 语句(空语句)，即一定要在设置 MCU 进入
;掉电模式的语句后加 2--4 条 NOP 语句(空语句)，如本程序中所示。
;假定测试芯片的工作频率为 18.432MHz

INT_CLKO    DATA    08FH        ;外部中断与时钟输出控制寄存器
INT3        BIT     P3.7        ;INT3 引脚定义
;---------------------------------------
    ORG     0000H
    LJMP    START               ;复位入口

    ORG     005BH               ;INT3 中断入口
    LJMP    EXINT3
;---------------------------------------
    ORG     0100H

START:
    MOV     SP, #3FH
    ORL     INT_CLKO, #20H      ;(EX3 = 1)使能 INT3 下降沿中断
    SETB    EA

LOOP:
    MOV     PCON, #02H          ;MCU 进入掉电模式
    NOP                         ;掉电模式被唤醒后,首先执行此语句,然后再进入中断服务程序
    NOP
    SJMP    LOOP
;---------------------------------------
;中断服务程序
EXINT3:
    CPL     P1.0                ;将测试口取反
;   ANL     INT_CLKO, #0DFH     ;若需要手动清除中断标志,可先关闭中断,
                                ;此时系统会自动清除内部的中断标志
;   ORL     INT_CLKO, #20H      ;然后再开中断即可
    RETI
;---------------------------------------
    END
```

13.3.3.7 用外部中断 INT4(下降沿)唤醒掉电模式/停机模式的测试程序(C 和汇编)

1.C 程序:

```
/*-------------------------------------------------------------------------------------------------*/
/*------STC15W4K60S4 系列 INT4 下降沿唤醒掉电模式举例---------------------------*/
/*------在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------------*/

//特别注意：在将进入掉电模式时一定要加入 2--4 条_nop_()语句(空语句)，即一定要在设置 MCU 进入
//掉电模式的语句后加 2--4 条_nop_()语句(空语句)，如本程序中所示。
//假定测试芯片的工作频率为 18.432MHz

#include "reg51.h"
#include "intrins.h"
//-----------------------------------------------
sfr     INT_CLKO = 0x8F;    //外部中断与时钟输出控制寄存器

sbit    INT4 = P3^0;        //INT4 引脚定义

sbit    P10 = P1^0;
//-----------------------------------------

//中断服务程序
void exint4() interrupt 16
{
    P10 = !P10;             //将测试口取反
//  INT_CLKO &= 0xBF;       //若需要手动清除中断标志,可先关闭中断,
                            //此时系统会自动清除内部的中断标志
//  INT_CLKO |= 0x40;       //然后再开中断即可
}
//-----------------------------------------------

void main()
{
    INT_CLKO |= 0x40;       //(EX4 = 1)使能 INT4 下降沿中断
    EA = 1;
    while (1)
    {
        PCON = 0x02;        //MCU 进入掉电模式
        _nop_();            //掉电模式被唤醒后,首先执行此语句,然后再进入中断服务程序
        _nop_();
    }
}
```

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/*------STC15W4K60S4 系列 INT3 下降沿唤醒掉电模式举例----------------------------------*/
/*------在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

;特别注意：在将进入掉电模式时一定要加入 2--4 条 NOP 语句(空语句)，即一定要在设置 MCU 进入
;掉电模式的语句后加 2--4 条 NOP 语句(空语句)，如本程序中所示。
;假定测试芯片的工作频率为 18.432MHz

INT_CLKO    DATA    08FH        ;外部中断与时钟输出控制寄存器
INT4        BIT     P3.0        ;INT4 引脚定义
;---------------------------------------
    ORG     0000H
    LJMP    START               ;复位入口

    ORG     0083H               ;INT4 中断入口

    LJMP    EXINT4
;---------------------------------------
    ORG     0100H

START:
    MOV     SP, #3FH
    ORL     INT_CLKO, #40H      ;(EX4 = 1)使能 INT4 下降沿中断
    SETB    EA

LOOP:
    MOV     PCON, #02H          ;MCU 进入掉电模式
    NOP                         ;掉电模式被唤醒后,首先执行此语句,然后再进入中断服务程序
    NOP
    SJMP    LOOP
;---------------------------------------
;中断服务程序
EXINT4:
    CPL     P1.0                ;将测试口取反
;   ANL     INT_CLKO, #0BFH     ;若需要手动清除中断标志,可先关闭中断,
                                ;此时系统会自动清除内部的中断标志
;   ORL     INT_CLKO, #40H      ;然后再开中断即可
    RETI
;---------------------------------------
    END
```

13.3.3.8 用 CCP/PCA 扩展的外部中断(下降沿+上升沿)唤醒掉电模式/停机模式的程序

1.C 程序:

```
/*---------------------------------------------------------------------------------------------------*/
/*------STC15W4K60S4 系列 PCA 扩展为外部中断唤醒掉电模式举例----------------*/
/*------在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------------*/

//特别注意：在将进入掉电模式时一定要加入 2--4 条_nop_()语句(空语句)，即一定要在设置 MCU 进入
//掉电模式的语句后加 2--4 条_nop_()语句(空语句)，如本程序中所示。
//假定测试芯片的工作频率为 18.432MHz

//本测试程序以 PCA 模块 0 为例进行说明,PCA 的模块 1 和模块 2 与模块 0 的实用方法相同
#include "reg51.h"
#include "intrins.h"

#define     FOSC            18432000L

typedef     unsigned char   BYTE;
typedef     unsigned int    WORD;
typedef     unsigned long   DWORD;

sfr         P_SW1 = 0xA2;           //外设功能切换寄存器 1

#define     CCP_S0 0x10             //P_SW1.4
#define     CCP_S1 0x20             //P_SW1.5

sfr         CCON = 0xD8;            //PCA 控制寄存器

sbit        CCF0 = CCON^0;          //PCA 模块 0 中断标志
sbit        CCF1 = CCON^1;          //PCA 模块 1 中断标志
sbit        CR = CCON^6;            //PCA 定时器运行控制位
sbit        CF = CCON^7;            //PCA 定时器溢出标志

sfr         CMOD = 0xD9;            //PCA 模式寄存器
sfr         CL = 0xE9;              //PCA 定时器低字节
sfr         CH = 0xF9;              //PCA 定时器高字节
sfr         CCAPM0 = 0xDA;          //PCA 模块 0 模式寄存器
sfr         CCAP0L = 0xEA;          //PCA 模块 0 捕获寄存器 LOW
sfr         CCAP0H = 0xFA;          //PCA 模块 0 捕获寄存器 HIGH
sfr         CCAPM1= 0xDB;           //PCA 模块 1 模式寄存器
sfr         CCAP1L = 0xEB;          //PCA 模块 1 捕获寄存器 LOW
sfr         CCAP1H = 0xFB;          //PCA 模块 1 捕获寄存器 HIGH
sfr         CCAPM2= 0xDC;           //PCA 模块 2 模式寄存器
sfr         CCAP2L = 0xEC;          //PCA 模块 2 捕获寄存器 LOW
```

```
sfr         CCAP2H = 0xFC;          //PCA 模块 2 捕获寄存器 HIGH
sfr         PCA_PWM0 = 0xf2;        //PCA 模块 0 的 PWM 寄存器
sfr         PCA_PWM1 = 0xf3;        //PCA 模块 1 的 PWM 寄存器
sfr         PCA_PWM2 = 0xf4;        //PCA 模块 2 的 PWM 寄存器

sbit        P10 = P1^0;

void main()
{
    ACC = P_SW1;
    ACC &= ~(CCP_S0 | CCP_S1);      //CCP_S0=0, CCP_S1=0
    P_SW1 = ACC;                    //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)

//  ACC = P_SW1;
//  ACC &= ~(CCP_S0 | CCP_S1);      //CCP_S0=1 CCP_S1=0
//  ACC |= CCP_S0;                  //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
//  P_SW1 = ACC;

//  ACC = P_SW1;
//  ACC &= ~(CCP_S0 | CCP_S1);      //CCP_S0=0 CCP_S1=1
//  ACC |= CCP_S1;                  //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
//  P_SW1 = ACC;
    CCON = 0;                       //初始化 PCA 控制寄存器
                                    //PCA 定时器停止
                                    //清除 CF 标志
                                    //清除模块中断标志
    CL = 0;                         //复位 PCA 寄存器
    CH = 0;
    CCAP0L = 0;
    CCAP0H = 0;
    CMOD = 0x08;                    //设置 PCA 时钟源为系统时钟
    CCAPM0= 0x21;                   //PCA 模块 0 为 16 位捕获模式(上升沿捕获,可测从高电平开始
                                    //的整个周期),且产生捕获中断,此时 CCP0 上的上升沿中断
                                    //可唤醒掉电模式
//  CCAPM0 = 0x11;                  //PCA 模块 0 为 16 位捕获模式(下降沿捕获,可测从低电平开始
                                    //的整个周期),且产生捕获中断,此时 CCP0 上的下降沿中断
                                    //可唤醒掉电模式
//  CCAPM0 = 0x31;                  //PCA 模块 0 为 16 位捕获模式(上升沿/下降沿捕获,可测高电平
                                    //或者低电平宽度),且产生捕获中断,此时 CCP0 上的上升沿和
                                    //下降沿中断可唤醒掉电模式
    CR = 1;                         //PCA 定时器开始工作
    EA = 1;
    while (1)
    {
        PCON = 0x02;                //MCU 进入掉电模式
        _nop_();                    //掉电模式被唤醒后,首先执行此语句,然后再进入中断服务程序
```

```
            _nop_();
        }
}
void PCA_isr() interrupt 7 using 1
{
    if (CCF0)                           //判断是否为捕获中断
    {
        CCF0 = 0;
        P10 = !P10;                     //将测试口取反
    }
}
```

2. 汇编程序：

```
/*-------------------------------------------------------------------------------------------------*/
/*------STC15W4K60S4 系列 PCA 扩展为外部中断唤醒掉电模式举例----------------*/
/*------在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------------*/

;特别注意：在将进入掉电模式时一定要加入 2--4 条 NOP 语句(空语句)，即一定要在设置 MCU 进入
;掉电模式的语句后加 2--4 条 NOP 语句(空语句)，如本程序中所示。
;假定测试芯片的工作频率为 18.432MHz

;本测试程序以 PCA 模块 0 为例进行说明,PCA 的模块 1 和模块 2 与模块 0 的实用方法相同

P_SW1       EQU     0A2H            ;外设功能切换寄存器 1

CCP_S0      EQU     10H             ;P_SW1.4
CCP_S1      EQU     20H             ;P_SW1.5
CCON        EQU     0D8H            ;PCA 控制寄存器
CCF0        BIT     CCON.0          ;PCA 模块 0 中断标志
CCF1        BIT     CCON.1          ;PCA 模块 1 中断标志
CR          BIT     CCON.6          ;PCA 定时器运行控制位
CF          BIT     CCON.7          ;PCA 定时器溢出标志
CMOD        EQU     0D9H            ;PCA 模式寄存器
CL          EQU     0E9H            ;PCA 定时器低字节
CH          EQU     0F9H            ;PCA 定时器高字节

CCAPM0      EQU     0DAH            ;PCA 模块 0 模式寄存器
CCAP0L      EQU     0EAH            ;PCA 模块 0 捕获寄存器  LOW
CCAP0H      EQU     0FAH            ;PCA 模块 0 捕获寄存器  HIGH

CCAPM1      EQU     0DBH            ;PCA 模块 1 模式寄存器
CCAP1L      EQU     0EBH            ;PCA 模块 1 捕获寄存器  LOW
CCAP1H      EQU     0FBH            ;PCA 模块 1 捕获寄存器  HIGH
```

```
CCAPM2      EQU     0DCH            ;PCA 模块 2 模式寄存器
CCAP2L      EQU     0ECH            ;PCA 模块 2 捕获寄存器 LOW
CCAP2H      EQU     0FCH            ;PCA 模块 2 捕获寄存器 HIGH

PCA_PWM0    EQU     0F2H            ;PCA 模块 0 的 PWM 寄存器
PCA_PWM1    EQU     0F3H            ;PCA 模块 1 的 PWM 寄存器
PCA_PWM2    EQU     0F4H            ;PCA 模块 2 的 PWM 寄存器
;---------------------------------------
    ORG     0000H
    LJMP    START

    ORG     003BH

PCA_ISR:
    PUSH    PSW
    PUSH    ACC

CKECK_CCF0:
    JNB     CCF0, PCA_ISR_EXIT      ;判断是否为捕获中断
    CLR     CCF0
    CPL     P1.0                    ;将测试口取反

PCA_ISR_EXIT:
    POP     ACC
    POP     PSW
    RETI
;---------------------------------------
    ORG     0100H

START:
    MOV     SP, #5FH
    MOV     A, P_SW1

    ANL     A, #0CFH                ;CCP_S0=0 CCP_S1=0
    MOV     P_SW1, A                ;(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)

;   MOV     A, P_SW1
;   ANL     A, #0CFH                ;CCP_S0=1 CCP_S1=0
;   ORL     A, #CCP_S0              ;(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
;   MOV     P_SW1, A

;   MOV     A, P_SW1
;   ANL     A, #0CFH                ;CCP_S0=0 CCP_S1=1
;   ORL     A, #CCP_S1              ;(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
;   MOV     P_SW1, A
```

```
    MOV     CCON, #0            ;初始化 PCA 控制寄存器
                                ;PCA 定时器停止
                                ;清除 CF 标志
                                ;清除模块中断标志
    CLR     A
    MOV     CL, A               ;复位 PCA 计时器
    MOV     CH, A
    MOV     CCAP0L, A
    MOV     CCAP0H, A
    MOV     CMOD, #08H          ;设置 PCA 时钟源为系统时钟
    MOV     CCAPM0, #21H        ;PCA 模块 0 为 16 位捕获模式(上升沿捕获,可测从高电平开始
                                ;的整个周期),且产生捕获中断,此时 CCP0 上的上升沿中断
                                ;可唤醒掉电模式
;   MOV     CCAPM0, #11H        ;PCA 模块 0 为 16 位捕获模式(下降沿捕获,可测从低电平开始
                                ;的整个周期),且产生捕获中断,次时 CCP0 上的下降沿中断
                                ;可唤醒掉电模式
;   MOV     CCAPM0, #31H        ;PCA 模块 0 为 16 位捕获模式(上升沿/下降沿捕获,可测高电平
                                ;或者低电平宽度),且产生捕获中断,此时 CCP0 上的上升沿和
                                ;下降沿中断均可唤醒掉电模式

    SETB    CR                  ;PCA 定时器开始工作
    SETB    EA

LOOP:
    MOV     PCON, #02H          ;MCU 进入掉电模式
    NOP                         ;掉电模式被唤醒后,首先执行此语句,然后再进入中断服务程序
    NOP
    SJMP    LOOP
;---------------------------------------
    END
```

13.3.3.9 用串口 1 接收管脚由高到低的变化唤醒掉电模式/停机模式的测试程序(C 和汇编)

——现供货的 STC15F2K60S2 系列 C 版本的串口 1 接收管脚不能唤醒掉电模式/停机模式

1.C 程序:

```
/*-------------------------------------------------------------------------------------------------*/
/*------STC15F2K60S2 系列 RxD 串行中断 1 唤醒掉电模式（停电模式）举例------*/
/*------在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------------*/

//特别注意：在将进入掉电模式时一定要加入 2--4 条_nop_()语句(空语句)，即一定要在设置 MCU 进入
//掉电模式的语句后加 2--4 条_nop_()语句(空语句)，如本程序中所示。
//假定测试芯片的工作频率为 18.432MHz
```

```
#include "reg51.h"
#include "intrins.h"
//-----------------------------------------------
sfr         AUXR = 0x8e;                //辅助寄存器
sfr         T2H = 0xd6;                 //定时器 2 高 8 位
sfr         T2L = 0xd7;                 //定时器 2 低 8 位
sfr         P_SW1 = 0xA2;               //外设功能切换寄存器 1

#define   S1_S0         0x40            //P_SW1.6
#define   S1_S1         0x80            //P_SW1.7

sbit        P10 = P1^0;

//-----------------------------------------------
void main()
{
    ACC = P_SW1;
    ACC &= ~(S1_S0 | S1_S1);            //S1_S0=0 S1_S1=0
    P_SW1 = ACC;                        //(P3.0/RxD, P3.1/TxD)

//  ACC = P_SW1;
//  ACC &= ~(S1_S0 | S1_S1);            //S1_S0=1 S1_S1=0
//  ACC |= S1_S0;                       //(P3.6/RxD_2, P3.7/TxD_2)
//  P_SW1 = ACC;

//  ACC = P_SW1;
//  ACC &= ~(S1_S0 | S1_S1);            //S1_S0=0 S1_S1=1
//  ACC |= S1_S1;                       //(P1.6/RxD_3, P1.7/TxD_3)
//  P_SW1 = ACC;

    SCON = 0x50;                        //8 位可变波特率
    T2L = (65536 - (FOSC/4/BAUD));      //设置波特率重装值
    T2H = (65536 - (FOSC/4/BAUD))>>8;
    AUXR = 0x14;                        //T2 为 1T 模式, 并启动定时器 2
    AUXR |= 0x01;                       //选择定时器 2 为串口 1 的波特率发生器

    ES = 1;
    EA = 1;

    while (1)
    {
        PCON = 0x02;                    //MCU 进入掉电模式
        _nop_();                        //掉电模式被唤醒后,直接从此语句开始向下执行,
                                        //不进入中断服务程序
        _nop_();
```

```
            P10 = !P10;                 //将测试口取反
        }
}
/*----------------------------
UART 中断服务程序
-----------------------------*/
void Uart() interrupt 4 using 1
{
    if (RI)
    {
        RI = 0;                         //清除 RI 位
        P0 = SBUF;                      //P0 显示串口数据
    }
    if (TI)
    {
        TI = 0;                         //清除 TI 位
    }
}
```

2. 汇编程序：

```
/*-------------------------------------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 RxD 串行中断 1 唤醒掉电模式(停电模式)举例----------*/
/*------在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------------*/
;特别注意：在将进入掉电模式时一定要加入 2--4 条 NOP 语句(空语句)，即一定要在设置 MCU 进入
;掉电模式的语句后加 2--4 条 NOP 语句(空语句)，如本程序中所示。
;假定测试芯片的工作频率为 18.432MHz
;---------------------------------------
AUXR        EQU     08EH        ;辅助寄存器
T2H         DATA    0D6H        ;定时器 2 高 8 位
T2L         DATA    0D7H        ;定时器 2 低 8 位
P_SW1       EQU     0A2H        ;外设功能切换寄存器 1

S1_S0       EQU     40H         ;P_SW1.6
S1_S1       EQU     80H         ;P_SW1.7
;---------------------------------------
    ORG     0000H
    LJMP    START               ;复位入口
    ORG     0023H               ;中断入口

    LJMP    UART_ISR
;---------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
```

```
        MOV     A, P_SW1
        ANL     A, #03FH            ;S1_S0=0 S1_S1=0
        MOV     P_SW1, A            ;(P3.0/RxD, P3.1/TxD)
;       MOV     A, P_SW1
;       ANL     A, #03FH            ;S1_S0=1 S1_S1=0
;       ORL     A, #S1_S0           ;(P3.6/RxD_2, P3.7/TxD_2)
;       MOV     P_SW1, A
;       MOV     A, P_SW1
;       ANL     A, #03FH            ;S1_S0=0 S1_S1=1
;       ORL     A, #S1_S1           ;(P1.6/RxD_3, P1.7/TxD_3)
;       MOV     P_SW1, A
        MOV     SCON, #50H          ;8 位可变波特率
        MOV     T2L, #0D8H          ;设置波特率重装值(65536-18432000/4/115200)
        MOV     T2H, #0FFH
        MOV     AUXR, #14H          ;T2 为 1T 模式, 并启动定时器 2
        ORL     AUXR, #01H          ;选择定时器 2 为串口 1 的波特率发生器
        SETB    ES                  ;打开串口中断
        SETB    A
LOOP:
        MOV     PCON, #02H          ;MCU 进入掉电模式
        NOP                         ;掉电模式被唤醒后,直接从此语句开始向下执行,
                                    ;不进入中断服务程序
        NOP
        CPL     P1.0                ;掉电唤醒后,取反测试口
        SJMP    LOOP
;/*---------------------------
;UART 中断服务程序
;---------------------------*/
UART_ISR:
        PUSH    ACC
        PUSH    PSW
        JNB     RI, CHECKTI         ;检测 RI 位
        CLR     RI                  ;清除 RI 位
        MOV     P0, SBUF            ;P0 显示串口数据

CHECKTI:
        JNB     TI, ISR_EXIT        ;检测 TI 位
        CLR     TI                  ;清除 TI 位

ISR_EXIT:
        POP     PSW
        POP     ACC
        RETI
;----------------------------------------
END
```

13.3.3.10 用串口 2 接收管脚由高到低的变化唤醒掉电/停机模式的测试程序(C 和汇编)

1. C 程序：

```
/*-------------------------------------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 RxD2 串行中断 2 唤醒掉电模式举例----------------------*/
/*------在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------------*/

//特别注意：在将进入掉电模式时一定要加入 2--4 条_nop_()语句(空语句)，即一定要在设置 MCU 进入
//掉电模式的语句后加 2--4 条_nop_()语句(空语句)，如本程序中所示。
//假定测试芯片的工作频率为 18.432MHz

#include "reg51.h"
#include "intrins.h"

#define     FOSC        18432000L           //系统频率
#define     BAUD        115200              //串口波特率
#define     TM          (65536 - (FOSC/4/BAUD))
//-----------------------------------------------
sfr         AUXR = 0x8e;                    //辅助寄存器
sfr         S2CON = 0x9a;                   //UART2  控制寄存器
sfr         S2BUF = 0x9b;                   //UART2  数据寄存器
sfr         T2H = 0xd6;                     //定时器 2 高 8 位
sfr         T2L = 0xd7;                     //定时器 2 低 8 位
sfr         IE2 = 0xaf;                     //中断控制寄存器 2

#define     S2RI        0x01                //S2CON.0
#define     S2TI        0x02                //S2CON.1
#define     S2RB8       0x04                //S2CON.2
#define     S2TB8       0x08                //S2CON.3

sfr         P_SW2 = 0xBA;                   //外设功能切换寄存器 2

#define     S2_S        0x01                //P_SW2.0

sbit        P20 = P2^0;
//-----------------------------------------------
void main()
{
    P_SW2 &= ~S2_S;             //S2_S=0 (P1.0/RxD2, P1.1/TxD2)
//  P_SW2 |= S2_S;              //S2_S=1 (P4.6/RxD2_2, P4.7/TxD2_2)
    S2CON = 0x50;               //8 位可变波特率
    T2L = TM;                   //设置波特率重装值
    T2H = TM>>8;
    AUXR = 0x14;                //T2 为 1T 模式, 并启动定时器 2
```

```
    IE2  = 0x01;                    //使能串口 2 中断
    EA = 1;
    while (1)
    {
        PCON =  0x02;               //MCU 进入掉电模式
        _nop_();                    //掉电模式被唤醒后,直接从此语句开始向下执行,
                                    //不进入中断服务程序
        _nop_();
        P20 = !P20;                 //将测试口取反
    }
}
/*----------------------------
UART2  中断服务程序
-----------------------------*/
void Uart2() interrupt 8 using 1
{
    if (S2CON & S2RI)
    {
        S2CON &= ~S2RI;             //清除 S2RI 位
        P0 = S2BUF;                 //P0 显示串口数据
    }
    if (S2CON & S2TI)
    {
        S2CON &= ~S2TI;             //清除 S2TI 位
    }
}
```

2. 汇编程序：

```
/*------------------------------------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 RxD2 串行中断 2 唤醒掉电模式举例-------------------------*/
/*------在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*------------------------------------------------------------------------------------------------*/

;特别注意：在将进入掉电模式时一定要加入 2--4 条 NOP 语句(空语句)，即一定要在设置 MCU 进入
;掉电模式的语句后加 2--4 条 NOP 语句(空语句)，如本程序中所示。
;假定测试芯片的工作频率为 18.432MHz

AUXR        EQU     08EH            ;辅助寄存器
S2CON       EQU     09AH            ;UART2  控制寄存器
S2BUF       EQU     09BH            ;UART2  数据寄存器
T2H         DATA    0D6H            ;定时器 2 高 8 位
T2L         DATA    0D7H            ;定时器 2 低 8 位
IE2         EQU     0AFH            ;中断控制寄存器 2

P_SW2       EQU     0BAH            ;外设功能切换寄存器 2
```

```
S2_S        EQU     01H             ;P_SW2.0

S2RI        EQU     01H             ;S2CON.0
S2TI        EQU     02H             ;S2CON.1
S2RB8       EQU     04H             ;S2CON.2
S2TB8       EQU     08H             ;S2CON.3
;-----------------------------------------
    ORG     0000H
    LJMP    START                   ;复位入口

    ORG     0043H                   ;中断入口
    LJMP    UART2_ISR
;-----------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    ANL     P_SW2, #NOT S2_S        ;S2_S=0 (P1.0/RxD2, P1.1/TxD2)

;   ORL     P_SW2, #S2_S            ;S2_S=1 (P4.6/RxD2_2, P4.7/TxD2_2)

    MOV     S2CON, #50H             ;8 位可变波特率
    MOV     T2L, #0D8H              ;设置波特率重装值(65536-18432000/4/115200)
    MOV     T2H, #0FFH
    MOV     AUXR, #14H              ;T2 为 1T 模式, 并启动定时器 2

    ORL     IE2, #01H               ;使能串口 2 中断
    SETB    EA

LOOP:
    MOV     PCON, #02H              ;MCU 进入掉电模式
    NOP                             ;掉电模式被唤醒后,直接从此语句开始向下执行,
                                    ;不进入中断服务程序
    NOP
    CPL     P1.0                    ;掉电唤醒后,取反测试口
    SJMP    LOOP
;/*---------------------------
;UART2  中断服务程序
;---------------------------*/
UART2_ISR:
    PUSH    ACC
    PUSH    PSW
    MOV     A, S2CON                ;读取 UART2 控制寄存器
    JNB     ACC.0, CHECKTI          ;检测 S2RI 位
    ANL     S2CON, #NOT S2RI        ;清除 S2RI 位
    MOV     P0, S2BUF               ;P0 显示串口数据
```

```
CHECKTI:
    MOV     A, S2CON                ;读取 UART2 控制寄存器
    JNB     ACC.1, ISR_EXIT         ;检测 S2TI 位
    ANL     S2CON, #NOT S2TI        ;清除 S2TI 位

ISR_EXIT:
    POP     PSW
    POP     ACC
    RETI
;----------------------------------------
END
```

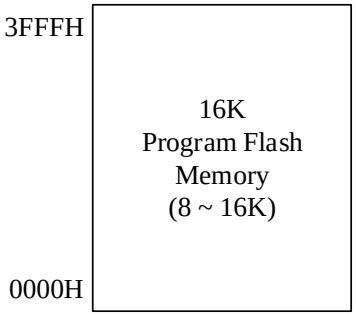
14 存储器，在哪存储重要参数，全球唯一 ID 号和特殊功能寄存器(SFRS)

STC15 系列单片机的程序存储器和数据存储器是各自独立编址的。STC15 系列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，因为没有访问外部程序存储器的总线。

STC15 系列单片机内部集成了大容量的数据存储器，如 STC15W4K32S4 系列单片机内部有 4096 字节的数据存储器、STC15F2K60S2 系列单片机内部有 2048 字节的数据存储器等。STC15W4K32S4 系列单片机内部的 4096 字节数据存储器在物理和逻辑上都分为两个地址空间：内部 RAM（256 字节）和内部扩展 RAM（3840 字节）。其中内部 RAM 的高 128 字节的数据存储器与特殊功能寄存器（SFRs）貌似地址重叠，实际使用时通过不同的寻址方式加以区分。另外，STC15 系列 40-pin 及其以上的单片机还可以访问在片外扩展的 64KB 外部数据存储器。

14.1 程序存储器

程序存储器用于存放用户程序、数据和表格等信息。以 STC15W4K32S4 系列单片机为例，STC15W4K32S4 系列单片内部集成了 8K ~ 61K 字节的 Flash 程序存储器。STC15W4K32S4 系列各种型号单片机的程序 Flash 存储器的地址如下表所示。



STC15W4K16S4单片机程序存储器

Type	Program Memory
STC15W4K08S4	0000H ~ 1FFFH (8K)
STC15W4K16S4	0000H ~ 3FFFH (16K)
STC15W4K24S4	0000H ~ 5FFFH (24K)
STC15W4K32S4	0000H ~ 7FFFH (32K)
STC15W4K40S4	0000H ~ 9FFFH (40K)
STC15W4K48S4	0000H ~ 0BFFFH (48K)
STC15W4K56S4	0000H ~ 0DFFFH (56K)
STC15W4K60S4	0000H ~ 0EFFFH (60K)
IAP15W4K61S4	0000H ~ 0F3FFH (61K)

单片机复位后，程序计数器（PC）的内容为 0000H，从 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 的中断服务程序的入口地址是 0003H，
- 定时器/计数器 0 中断服务程序的入口地址是 000BH，
- 外部中断 1 的中断服务程序的入口地址是 0013H，

- 定时器/计数器 1 的中断服务程序的入口地址是 001BH 等。

更多的中断服务程序的入口地址（中断向量）见单独的中断章节。

由于相邻中断入口地址的间隔区间（8 个字节）有限，一般情况下无法保存完整的中断服务程序，因此，一般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。

程序 Flash 存储器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。

14.2 数据存储器（SRAM）

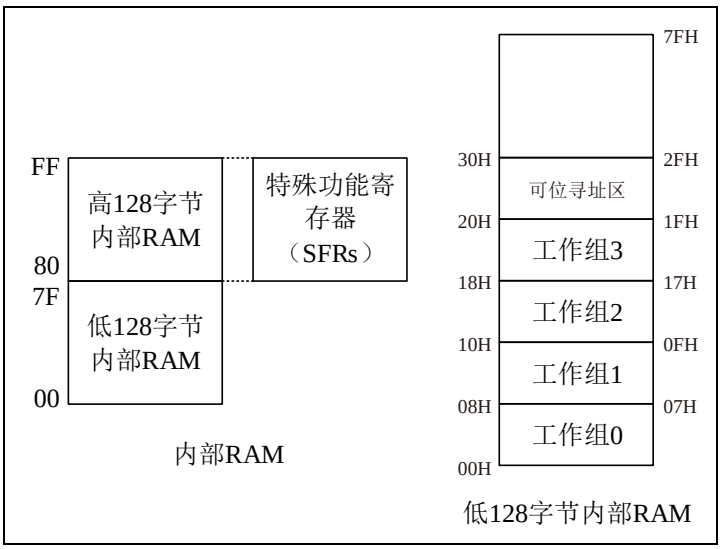
下表总结了 STC15 系列单片机内部数据存储器（SRAMD）的空间大小以及是否可以扩展片外数据存储器：

数据存储器 / 单片机型号	内部集成的总 SRAM 大小 (字节/Byte)	内部扩展 RAM 空间大小 (字节/Byte)	是否可以片外扩展 64KB 的外部数据存储器
STC15W4K32S4 系列	4K (256 <idata> + 3840 <xdata>)	3840	可以
STC15F2K60S2 系列	2K (256 <idata> + 1792 <xdata>)	1792	可以
STC15W1K16S 系列	1K (256 <idata> +768 <xdata>)	768	可以
STC15W404S 系列	512 (256 <idata> +256 <xdata>)	256	可以
STC15W408AS 系列	512 (256 <idata> +256 <xdata>)	256	不可以
STC15W204S 系列	256 <idata>	无内部扩展 RAM	不可以
STC15F408AD 系列	512 (256 <idata> +256 <xdata>)	256	不可以
STC15F104W 系列	128 <idata>	无内部扩展 RAM	不可以

STC15 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。以 STC15W4K32S4 系列单片机为例，STC15W4K32S4 系列单片机内部集成了 4096 字节 RAM 内部数据存储器，其在物理和逻辑上都分为两个地址空间：内部 RAM（256 字节）和内部扩展 RAM（3840 字节）。此外，STC15 系列 40-pin 及其以上的单片机还可以访问在片外扩展的 64KB 外部数据存储器。

14.2.1 内部 RAM

内部 RAM 共 256 字节，可分为 3 个部分：**低 128 字节 RAM（与统 8051 兼容）、高 128 字节 RAM（Intel 在 8052 中扩展了高 128 字节 RAM）**及特殊功能寄存器区。低 128 字节的数据存储器既可直接寻址也可间接寻址。高 128 字节 RAM 与特殊功能寄存器区貌似共用相同的地址范围，都使用 80H ~ FFH，地址空间虽然貌似重叠，但物理上是独立的，使用时通过不同的寻址方式加以区分。高 128 字节 RAM 只能间接寻址，特殊功能寄存器区只可直接寻址。内部 RAM 的结构如下图所示，地址范围是 00H ~ FFH。



低 128 字节 RAM 也称通用 RAM 区。通用 RAM 区又可分为工作寄存器组区，可位寻址区，用户 RAM 区和堆栈区。

工作寄存器组区地址从 00H ~ 1FH 共 32B（字节）单元，分为 4 组（每一组称为一个寄存器组），每组包含 8 个 8 位的工作寄存器，编号均为 R0 ~ R7，但属于不同的物理空间。通过使用工作寄存器组，可以提高运算速度。R0 ~ R7 是常用的寄存器，提供 4 组是因为 1 组往往不够用。

程序状态字 PSW 寄存器中的 RS1 和 RS0 组合决定当前使用的工作寄存器组。见下面 PSW 寄存器的介绍。

可位寻址区的地址从 20H ~ 2FH 共 16 个字节单元。20H ~ 2FH 单元既可向普通 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。内部 RAM 中的 30H ~ FFH 单元是用户 RAM 和堆栈区。一个 8 位的堆栈指针（SP），用于指向堆栈区。单片机复位后，堆栈指针 SP 为 07H，指向了工作寄存器组 0 中的 R7，因此，用户初始化程序都应对 SP 设置初值，一般设置在 80H 以后的单元为宜。

PSW ：程序状态字寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
PSW	D0H	name	CY	AC	F0	RS1	RS0	OV	-	P

CY： 标志位。进行加法运算时，当最高位即 B7 位有进位，或执行减法运算最高位有借位时 CY 为 1；反之为 0

AC： 进位辅助位。进行加法运算时，当 B3 位有进位，或执行减法运算 B3 有借位时，AC 为 1；反之为 0。设置辅助进位标志 AC 的目的是为了便于 BCD 码加法、减法运算的调整。

F0： 用户标志位

RS1、RS0：工作寄存器组的选择位。如下表

RS1	RS0	当前使用的工作寄存器组(R0 ~ R7)
0	0	0 组(00H ~ 07H)
0	1	1 组(08H ~ 0FH)
1	0	2 组(10H ~ 17H)

1	1	3 组(18H ~ 1FH)

OV： 溢出标志位

B1： 保留位

P： 奇偶标志位。该标志位始终体现累加器 ACC 中 1 的个数的奇偶性。如果累加器 ACC 中 1 的个数为奇数，则 P 置 1；当累加器 ACC 中的个数为偶数(包括 0 个)时，P 位为 0。

堆栈指针(SP)：

堆栈指针是一个 8 位专用寄存器。它指示出堆栈顶部在内部 RAM 块中的位置。系统复位后，SP 初始化位 07H，使得堆栈事实上由 08H 单元开始，考虑 08H ~ 1FH 单元分别属于工作寄存器组 1 ~ 3，若在程序设计中用到这些区，则最好把 SP 值改变为 80H 或更大的值为宜。STC15 系列单片机的堆栈是向上生长的，即将数据压入堆栈后，SP 内容增大。

14.2.2 内部扩展 RAM/XRAM/AUX-RAM 及测试程序

STC15W4K32S4 系列单片机片内除了集成 256 字节的内部 RAM 外，还集成了 3840 字节的扩展 RAM，地址范围是 0000H ~ 0EFFH.访问内部扩展 RAM 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(数据总线和高八位地址总线)、P2 口(低八位地址总线)、WR/P4.2、RD/P4.4 和 ALE/P4.5。在汇编语言中，内部扩展 RAM 通过 MOVX 指令访问，即使用"MOVX @ DPTR"或者"MOVX @ Ri"指令访问。在 C 语言中，可使用 xdata 声明存储类型即可，如"unsigned char xdata i=0;"。

单片机内部扩展 RAM 是否可以访问受辅助寄存器 AUXR(地址为 8EH)中的 EXTRAM 位控制。

STC15W4K32S4 系列单片机 8051 单片机 扩展 RAM 管理及禁止 ALE 输出 特殊功能寄存器

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR	8EH	Auxiliary Register	T0x12	T1x12	UAR_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000,0001

EXTRAM： Internal/External RAM access 内部/外部 RAM 存取

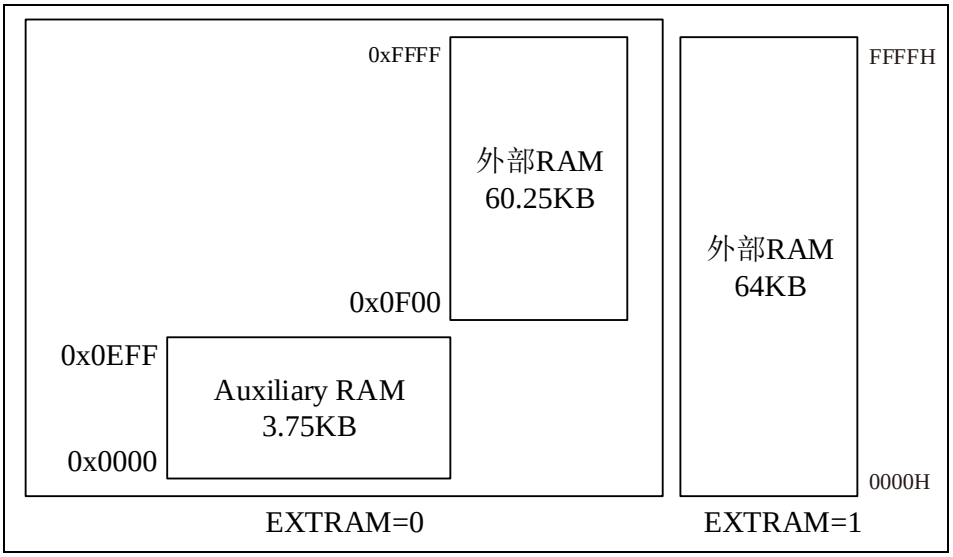
0：内部扩展的 EXT_RAM 可以存取

STC15W4K32S4 系列单片机

在 00H 到 EFFH 单元(3840 字节)，使用 MOVX @ DPTR 指令访问，超过 F00H(含 F00H 单元)的地址空间总是访问外部数据存储器，MOVX @ Ri 只能访问 00H 到 FFH 单元

1：External data memory access. 外部数据存储器存取

禁止访问内部扩展 RAM，此时 MOVX @DPTR/MOVX@Ri 的使用同普通 8052 单片机



应用示例供参考(汇编)

访问内部扩展的 EXTRAM

```
;新增特殊功能寄存器声明(汇编方式)
AUXR        DATA    8EH         ;或者用 AUXR    EQU     8EH 定义
MOV         AUXR, #00000000B    ;EXTRAM 位清为“0”，其实上电复位时此位就为“0”
;MOVXA      @DPTR / MOVX        @DPTR, A 指令可访问内部扩展的 EXTRAM
                                ;STC15W4K32S4 系列为(00H - EFFH, 共 3840 字节)

;MOVX       A, @Ri / MOVX       A, @Ri 指令可直接访问内部扩展的 EXTRAM
                                ;使用此指令 只能访问内部扩展的 EXTRAM(00H - FFH,共 256 字节)
```

```
;写芯片内部扩展的 EXTRAM
    MOV     DPTR, #address
    MOV     A, #value
    MOVX    @DPTR, A
;读芯片内部扩展的 EXTRAM
    MOV     DPTR, #address
    MOVX    A, @DPTR
```

STC15W4K32S4 系列

;如果 #address < F00H，则在 EXTRAM 位为"0"时，访问物理上在内部，逻辑上在外部的此 EXTRAM

;如果 #address >= F00H，则总是访问物理上外部扩展的 RAM 或 I/O 空间(F00H -- FFFFH)

禁止访问内部扩展的 EXTRAM，以防冲突

```
    MOV     AUXR, #00000010B        ; EXTRAM 控制位设置为"1"，禁止访问 EXTRAM，以防冲突
```

有些用户系统因为外部扩展了 I/O 或者用片选去选多个 RAM 区，有时与此内部扩展的 EXTRAM 逻辑地址上有冲突，将此位设置为"1"，禁止访问此内部扩展的 EXTRAM 就可以了。

大实话：

其实不用设置 AUXR 寄存器即可直接用 MOVX @DPTR 指令访问此内部扩展的 EXTRAM，超过此 RAM 空间，将访问片外单元。如果系统外扩了 SRAM，而实际使用的空间小于 3840 字节，则可直接将此 SRAM 省去，比如省去 STC62WV256，IS62C256，UT6264 等。

应用示例供参考(C 语言):

```
/*访问内部扩展的 EXTRAM */
/*STC15W4K32S4 系列单片机为(00H - EFFH，共 3840 字节扩展的 EXTRAM)*/
/*新增特殊功能寄存器声明(C 语言方式)*/
sfr     AUXR=0x8e           /*如果不需设置 AUXR 就不用声明 AUXR*/
AUXR = 0x00                 /*0000,0000 EXTRAM 位清 0，其实上电复位时此位就为 0*/
unsigned char xdata sum, loop_counter, test_array[128];
/*将变量声明成 xdata  即可直接访问此内部扩展的 EXTRAM*/

/*  写芯片内部扩展的 EXTRAM */
    sum = 0;
    loop_counter = 128;
    test_array[0] = 5;

/*  读芯片内部扩展的 EXTRAM */
    sum = test_array[0];
```

如果 #address < F00H，则在 EXTRAM 位为"0"时，访问物理上在内部，逻辑上在外部的此 EXTRAM

如果 #address >= F00H，则总是访问物理上外部扩展的 RAM 或 I/O 空间（F00H - FFFFH）

禁止访问内部扩展的 EXTRAM.，以防冲突

```
    AUXR = 0x02     /*0000,0010,EXTRAM 位设为"1",禁止访问 EXTRAM,以防冲突*/
```

有些用户系统因为外部扩展了 I/O 或者用片选去选多个 RAM 区，有时与此内部扩展的 EXTRAM 逻辑上有冲突，将此位设置为"1"，禁止访问此内部扩展的 EXTRAM 就可以了。

14.2.3 使用内部扩展 RAM 的测试程序

STC15 系列单片机内部扩展 RAM 演示程序，本程序应用 2048 的内部扩展 RAM

```
;/* ---  演示 STC15 系列单片机 MCU 内部扩展 RAM 演示程序---------------------- */
;/* ---  本演示程序在 STC15 系列 ISP 的下载编程工具上测试通过  ------------------- */

#include <reg52.h>
#include <intrins.h>                    /* use _nop_() function */
sfr     AUXR = 0x8e;

sbit    ERROR_LED = P1^5;
sbit    OK_LED = P1^7;

void main()
{
    unsigned int array_point = 0;
    /*测试数组 Test_array_one[1024],Test_array_two[1024]*/
    unsigned char xdata Test_array_one[1024] =
    {
        0x00,   0x01,   0x02,   0x03,   0x04,   0x05,   0x06,   0x07,
        0x08,   0x09,   0x0c,   0x0a,   0x0b,   0x0d,   0x0e,   0x0f,
        0x10,   0x11,   0x12,   0x13,   0x14,   0x15,   0x16,   0x17,
        0x18,   0x19,   0xla,   0x1b,   0x1c,   0xld,   0xle,   0xlf,
        0x20,   0x21,   0x22,   0x23,   0x24,   0x25,   0x26,   0x27,
        0x28,   0x29,   0x2a,   0x2b,   0x2c,   0x2d,   0x2e,   0x2f,
        0x30,   0x31,   0x32,   0x33,   0x34,   0x35,   0x36,   0x37,
        0x38,   0x39,   0x3a,   0x3b,   0x3c,   0x3d,   0x3e,   0x3f,
        0x40,   0x41,   0x42,   0x43,   0x44,   0x45,   0x46,   0x47,
        0x48,   0x49,   0x4a,   0x4b，  0x4c,   0x4d,   0x4e,   0x4f,
        0x50,   0x51,   0x52,   0x53,   0x54,   0x55,   0x56,   0x57,
        0x58,   0x59,   0x5a,   0x5b,   0x5c,   0x5d,   0x5e,   0x5f,
        0x60,   0x61,   0x62,   0x63,   0x64,   0x65,   0x66,   0x67,
        0x68,   0x69,   0x6a,   0x6b,   0x6c,   0x6d,   0x6e,   0x6f,
        0x70,   0x71,   0x72,   0x73,   0x74,   0x75,   0x76,   0x77,
        0x78,   0x79,   0x7a,   0x7b,   0x7c,   0x7d,   0x7e,   0x7f,
        0x80,   0x81,   0x82,   0x83,   0x84,   0x85,   0x86,   0x87,
        0x88,   0x89,   0x8a,   0x8b,   0x8c,   0x8d,   0x8e,   0x8f,
        0x90,   0x91,   0x92,   0x93,   0x94,   0x95,   0x96,   0x97,
        0x98,   0x99,   0x9a,   0x9b,   0x9c,   0x9d,   0x9e,   0x9f,
        0xa0,   0xal,   0xa2,   0xa3,   0xa4,   0xa5,   0xa6,   0xa7,
        0xa8,   0xa9,   0xaa,   0xab,   0xac,   0xad,   0xae,   0xaf,
        0xb0,   0xb1,   0xb2,   0xb3,   0xb4,   0xb5,   0xb6,   0xb7,
        0xb8,   0xb9,   0xba,   0xbb,   0xbc，  0xbd,   0xbe,   0xbf,
        0xc0,   0xc1,   0xc2,   0xc3,   0xe4,   0xc5,   0xc6,   0xe7,
        0xc8,   0xc9,   0xca,   0xcb,   0xcc,   0xcd,   0xce,   0xef,
```

```
0xd0,  0xd1,  0xd2,  0xd3,  0xd4,  0xd5,  0xd6,  0xd7,
0xd8,  0xd9,  0xda,  0xdb,  0xdc,  0xdd,  0xde,  0xdf,
0xe0,  0xe1,  0xe2,  0xe3,  0xe4,  0xe5,  0xe6,  0xe7,
0xe8,  0xe9,  0xea,  0xeb,  0xec,  0xed,  0xee,  0xef,
0xf0,  0xf1,  0xf2,  0xf3,  0xf4,  0xf5,  0xf6,  0xf7,
0xf8,  0xf9,  0xfa,  0xfb,  0xfc,  0xfd,  0xfe,  0xff,

0xff,  0xfe,  0xfd,  0xfc,  0xfb,  0xfa,  0xf9,  0xf8,
0xf7,  0xf6,  0xf5,  0xf4,  0xf3,  0xf2,  0xf1,  0xf0,
0xef,  0xee,  0xed,  0xec,  0xeb,  0xea,  0xe9,  0xe8,
0xe7,  0xe6,  0xe5,  0xe4,  0xe3,  0xe2,  0xel,  0xe0,
0xdf,  0xde,  0xdd,  0xdc,  0xdb,  0xda,  0xd9,  0xd8
0xd7,  0xd6,  0xd5,  0xd4,  0xd3,  0xd2,  0xd1,  0xd0
0xef,  0xce,  0xcd,  0xcc,  0xcb,  0xca,  0xc9,  0xc8,
0xe7,  0xc6,  0xc5，0xc4,  0xc3,  0xc2,  0xc1,  0xe0,
0xbf,  0xbe,  0xbd,  0xbc,  0xbb,  0xba,  0xb9,  0xb8,
0xb7,  0xb6,  0xb5,  0xb4,  0xb3,  0xb2,  0xb1,  0xb0,
0xaf,  0xae,  0xad,  0xac,  0xab,  0xaa,  0xa9,  0xa8,
0xa7,  0xa6,  0xa5,  0xa4,  0xa3,  0xa2,  0xa1,  0xa0,
0x9f,  0x9e,  0x9d,  0x9c,  0x9b,  0x9a,  0x99,  0x98,
0x97,  0x96,  0x95,  0x94,  0x93,  0x92,  0x91,  0x90,
0x8f,  0x8e,  0x8d,  0x8c,  0x8b,  0x8a,  0x89,  0x88,
0x87,  0x86,  0x85,  0x84,  0x83,  0x82,  0x81,  0x80,
0x7f,  0x7e,  0x7d,  0x7c,  0x7b,  0x7a,  0x79,  0x78,
0x77,  0x76,  0x75,  0x74,  0x73,  0x72,  0x71,  0x70,
0x6f,  0x6e,  0x6d,  0x6c,  0x6b,  0x6a,  0x69,  0x68,
0x67,  0x66,  0x65,  0x64,  0x63,  0x62,  0x61,  0x60,
0x5f,  0x5e,  0x5d,  0x5c,  0x5b,  0x5a,  0x59,  0x58,
0x57,  0x56,  0x55,  0x54,  0x53,  0x52,  0x51,  0x50,
0x4f,  0x4e,  0x4d,  0x4c,  0x4b,  0x4a,  0x49,  0x48,
0x47,  0x46,  0x45,  0x44,  0x43,  0x42,  0x41,  0x40,
0x3f,  0x3e,  0x3d,  0x3c,  0x3b,  0x3a,  0x39,  0x38,
0x37,  0x35,  0x36,  0x34,  0x33,  0x32,  0x31,  0x30,
0x2f,  0x2d,  0x2e,  0x2c,  0x2b,  0x2a,  0x29,  0x28,
0x27,  0x26,  0x25,  0x24,  0x23,  0x22,  0x20,  0x21,
0x1f,  0x1e,  0x1d,  0x1c,  0x1b,  0x1a,  0x19,  0x18,
0x17,  0x16,  0x15,  0x14,  0x13,  0x12,  0x11,  0x10,
0x0f,  0x0e,  0x0d,  0x0c,  0x0b,  0x0a,  0x09,  0x08,
0x05,  0x07,  0x06,  0x04,  0x03,  0x02,  0x01,  0x00,

0x00,  0x01,  0x02,  0x03,  0x04,  0x05,  0x06,  0x07,
0x08,  0x09,  0x0c,  0x0a,  0x0b,  0x0d,  0x0e,  0x0f,
0x10,  0x11,  0x12,  0x13,  0x14,  0x15,  0x16,  0x17,
0x18,  0x19,  0xla,  0x1b,  0x1c,  0xld,  0xle,  0xlf,
0x20,  0x21,  0x22,  0x23,  0x24,  0x25,  0x26,  0x27,
0x28,  0x29,  0x2a,  0x2b,  0x2c,  0x2d,  0x2e,  0x2f,
```

```
0x30,   0x31,   0x32,   0x33,    0x34,   0x35,   0x36,   0x37,
0x38,   0x39,   0x3a,   0x3b,   0x3c,   0x3d,   0x3e,   0x3f,
0x40,   0x41,   0x42,   0x43,   0x44,   0x45,   0x46,   0x47,
0x48,   0x49,   0x4a,   0x4b，  0x4c,   0x4d,   0x4e,   0x4f,
0x50,   0x51,   0x52,   0x53,   0x54,   0x55,   0x56,   0x57,
0x58,   0x59,   0x5a,   0x5b,   0x5c,   0x5d,   0x5e,   0x5f,
0x60,   0x61,   0x62,   0x63,   0x64,   0x65,   0x66,   0x67,
0x68,   0x69,   0x6a,   0x6b,   0x6c,   0x6d,   0x6e,   0x6f,
0x70,   0x71,   0x72,   0x73,   0x74,   0x75,   0x76,   0x77,
0x78,   0x79,   0x7a,   0x7b,   0x7c,   0x7d,   0x7e,   0x7f,
0x80,   0x81,   0x82,   0x83,   0x84,   0x85,   0x86,   0x87,
0x88,   0x89,   0x8a,   0x8b,   0x8c,   0x8d,   0x8e,   0x8f,
0x90,   0x91,   0x92,   0x93,   0x94,   0x95,   0x96,   0x97,
0x98,   0x99,   0x9a,   0x9b,   0x9c,   0x9d,   0x9e,   0x9f,
0xa0,   0xal,   0xa2,   0xa3,   0xa4,   0xa5,   0xa6,   0xa7,
0xa8,   0xa9,   0xaa,   0xab,   0xac,   0xad,   0xae,   0xaf,
0xb0,   0xb1,   0xb2,   0xb3,   0xb4,   0xb5,   0xb6,   0xb7,
0xb8,   0xb9,   0xba,   0xbb,   0xbc，  0xbd,   0xbe,   0xbf,
0xc0,   0xc1,   0xc2,   0xc3,   0xe4,   0xc5,   0xc6,   0xe7,
0xc8,   0xc9,   0xca,   0xcb,   0xcc,   0xcd,   0xce,   0xef,
0xd0,   0xd1,   0xd2,   0xd3,   0xd4,   0xd5,   0xd6,   0xd7,
0xd8,   0xd9,   0xda,   0xdb,   0xdc,   0xdd,   0xde,   0xdf,
0xe0,   0xe1,   0xe2,   0xe3,   0xe4,   0xe5,   0xe6,   0xe7,
0xe8,   0xe9,   0xea,   0xeb,   0xec,   0xed,   0xee,   0xef,
0xf0,   0xf1,   0xf2,   0xf3,   0xf4,   0xf5,   0xf6,   0xf7,
0xf8,   0xf9,   0xfa,   0xfb,   0xfc,   0xfd,   0xfe,   0xff,

0xff,   0xfe,   0xfd,   0xfc,   0xfb,   0xfa,   0xf9,   0xf8,
0xf7,   0xf6,   0xf5,   0xf4,   0xf3,   0xf2,   0xf1,   0xf0,
0xef,   0xee,   0xed,   0xec,   0xeb,   0xea,   0xe9,   0xe8,
0xe7,   0xe6,   0xe5,   0xe4,   0xe3,   0xe2,   0xel,   0xe0,
0xdf,   0xde,   0xdd,   0xdc,   0xdb,   0xda,   0xd9,   0xd8,
0xd7,   0xd6,   0xd5,   0xd4,   0xd3,   0xd2,   0xd1,   0xd0,
0xef,   0xce,   0xcd,   0xcc,   0xcb,   0xca,   0xc9,   0xc8,
0xe7,   0xc6,   0xc5，  0xc4,   0xc3,   0xc2,   0xc1,   0xe0,
0xbf,   0xbe,   0xbd,   0xbc,   0xbb,   0xba,   0xb9,   0xb8,
0xb7,   0xb6,   0xb5,   0xb4,   0xb3,   0xb2,   0xb1,   0xb0,
0xaf,   0xae,   0xad,   0xac,   0xab,   0xaa,   0xa9,   0xa8,
0xa7,   0xa6,   0xa5,   0xa4,   0xa3,   0xa2,   0xa1,   0xa0,
0x9f,   0x9e,   0x9d,   0x9c,   0x9b,   0x9a,   0x99,   0x98,
0x97,   0x96,   0x95,   0x94,   0x93,   0x92,   0x91,   0x90,
0x8f,   0x8e,   0x8d,   0x8c,   0x8b,   0x8a,   0x89,   0x88,
0x87,   0x86,   0x85,   0x84,   0x83,   0x82,   0x81,   0x80,
0x7f,   0x7e,   0x7d,   0x7c,   0x7b,   0x7a,   0x79,   0x78,
0x77,   0x76,   0x75,   0x74,   0x73,   0x72,   0x71,   0x70,
0x6f,   0x6e,   0x6d,   0x6c,   0x6b,   0x6a,   0x69,   0x68,
```

```
    0x67,   0x66,   0x65,   0x64,   0x63,   0x62,   0x61,   0x60,
    0x5f,   0x5e,   0x5d,   0x5c,   0x5b,   0x5a,   0x59,   0x58,
    0x57,   0x56,   0x55,   0x54,   0x53,   0x52,   0x51,   0x50,
    0x4f,   0x4e,   0x4d,   0x4c,   0x4b,   0x4a,   0x49,   0x48,
    0x47,   0x46,   0x45,   0x44,   0x43,   0x42,   0x41,   0x40,
    0x3f,   0x3e,   0x3d,   0x3c,   0x3b,   0x3a,   0x39,   0x38,
    0x37,   0x35,   0x36,   0x34,   0x33,   0x32,   0x31,   0x30,
    0x2f,   0x2d,   0x2e,   0x2c,   0x2b,   0x2a,   0x29,   0x28,
    0x27,   0x26,   0x25,   0x24,   0x23,   0x22,   0x20,   0x21,
    0x1f,   0x1e,   0x1d,   0x1c,   0x1b,   0x1a,   0x19,   0x18,
    0x17,   0x16,   0x15,   0x14,   0x13,   0x12,   0x11,   0x10,
    0x0f,   0x0e,   0x0d,   0x0c,   0x0b,   0x0a,   0x09,   0x08,
    0x05,   0x07,   0x06,   0x04,   0x03,   0x02,   0x01,   0x00,
};
unsigned char xdata Test_array_two[1024] =
{
    0x00,   0x01,   0x02,   0x03,   0x04,   0x05,   0x06,   0x07,
    0x08,   0x09,   0x0c,   0x0a,   0x0b,   0x0d,   0x0e,   0x0f,
    0x10,   0x11,   0x12,   0x13,   0x14,   0x15,   0x16,   0x17,
    0x18,   0x19,   0xla,   0x1b,   0x1c,   0xld,   0xle,   0xlf,
    0x20,   0x21,   0x22,   0x23,   0x24,   0x25,   0x26,   0x27,
    0x28,   0x29,   0x2a,   0x2b,   0x2c,   0x2d,   0x2e,   0x2f,
    0x30,   0x31,   0x32,   0x33,   0x34,   0x35,   0x36,   0x37,
    0x38,   0x39,   0x3a,   0x3b,   0x3c,   0x3d,   0x3e,   0x3f,
    0x40,   0x41,   0x42,   0x43,   0x44,   0x45,   0x46,   0x47,
    0x48,   0x49,   0x4a,   0x4b，  0x4c,   0x4d,   0x4e,   0x4f,
    0x50,   0x51,   0x52,   0x53,   0x54,   0x55,   0x56,   0x57,
    0x58,   0x59,   0x5a,   0x5b,   0x5c,   0x5d,   0x5e,   0x5f,
    0x60,   0x61,   0x62,   0x63,   0x64,   0x65,   0x66,   0x67,
    0x68,   0x69,   0x6a,   0x6b,   0x6c,   0x6d,   0x6e,   0x6f,
    0x70,   0x71,   0x72,   0x73,   0x74,   0x75,   0x76,   0x77,
    0x78,   0x79,   0x7a,   0x7b,   0x7c,   0x7d,   0x7e,   0x7f,
    0x80,   0x81,   0x82,   0x83,   0x84,   0x85,   0x86,   0x87,
    0x88,   0x89,   0x8a,   0x8b,   0x8c,   0x8d,   0x8e,   0x8f,
    0x90,   0x91,   0x92,   0x93,   0x94,   0x95,   0x96,   0x97,
    0x98,   0x99,   0x9a,   0x9b,   0x9c,   0x9d,   0x9e,   0x9f,
    0xa0,   0xal,   0xa2,   0xa3,   0xa4,   0xa5,   0xa6,   0xa7,
    0xa8,   0xa9,   0xaa,   0xab,   0xac,   0xad,   0xae,   0xaf,
    0xb0,   0xb1,   0xb2,   0xb3,   0xb4,   0xb5,   0xb6,   0xb7,
    0xb8,   0xb9,   0xba,   0xbb,   0xbc，  0xbd,   0xbe,   0xbf,
    0xc0,   0xc1,   0xc2,   0xc3,   0xe4,   0xc5,   0xc6,   0xe7,
    0xc8,   0xc9,   0xca,   0xcb,   0xcc,   0xcd,   0xce,   0xef,
    0xd0,   0xd1,   0xd2,   0xd3,   0xd4,   0xd5,   0xd6,   0xd7,
    0xd8,   0xd9,   0xda,   0xdb,   0xdc,   0xdd,   0xde,   0xdf,
    0xe0,   0xe1,   0xe2,   0xe3,   0xe4,   0xe5,   0xe6,   0xe7,
    0xe8,   0xe9,   0xea,   0xeb,   0xec,   0xed,   0xee,   0xef,
```

```
0xf0,   0xf1,   0xf2,   0xf3,   0xf4,   0xf5,   0xf6,   0xf7,
0xf8,   0xf9,   0xfa,   0xfb,   0xfc,   0xfd,   0xfe,   0xff,

0xff,   0xfe,   0xfd,   0xfc,   0xfb,   0xfa,   0xf9,   0xf8,
0xf7,   0xf6,   0xf5,   0xf4,   0xf3,   0xf2,   0xf1,   0xf0,
0xef,   0xee,   0xed,   0xec,   0xeb,   0xea,   0xe9,   0xe8,
0xe7,   0xe6,   0xe5,   0xe4,   0xe3,   0xe2,   0xel,   0xe0,
0xdf,   0xde,   0xdd,   0xdc,   0xdb,   0xda,   0xd9,   0xd8
0xd7,   0xd6,   0xd5,   0xd4,   0xd3,   0xd2,   0xd1,   0xd0
0xef,   0xce,   0xcd,   0xcc,   0xcb,   0xca,   0xc9,   0xc8,
0xe7,   0xc6,   0xc5，  0xc4,   0xc3,   0xc2,   0xc1,   0xe0,
0xbf,   0xbe,   0xbd,   0xbc,   0xbb,   0xba,   0xb9,   0xb8,
0xb7,   0xb6,   0xb5,   0xb4,   0xb3,   0xb2,   0xb1,   0xb0,
0xaf,   0xae,   0xad,   0xac,   0xab,   0xaa,   0xa9,   0xa8,
0xa7,   0xa6,   0xa5,   0xa4,   0xa3,   0xa2,   0xa1,   0xa0,
0x9f,   0x9e,   0x9d,   0x9c,   0x9b,   0x9a,   0x99,   0x98,
0x97,   0x96,   0x95,   0x94,   0x93,   0x92,   0x91,   0x90,
0x8f,   0x8e,   0x8d,   0x8c,   0x8b,   0x8a,   0x89,   0x88,
0x87,   0x86,   0x85,   0x84,   0x83,   0x82,   0x81,   0x80,
0x7f,   0x7e,   0x7d,   0x7c,   0x7b,   0x7a,   0x79,   0x78,
0x77,   0x76,   0x75,   0x74,   0x73,   0x72,   0x71,   0x70,
0x6f,   0x6e,   0x6d,   0x6c,   0x6b,   0x6a,   0x69,   0x68,
0x67,   0x66,   0x65,   0x64,   0x63,   0x62,   0x61,   0x60,
0x5f,   0x5e,   0x5d,   0x5c,   0x5b,   0x5a,   0x59,   0x58,
0x57,   0x56,   0x55,   0x54,   0x53,   0x52,   0x51,   0x50,
0x4f,   0x4e,   0x4d,   0x4c,   0x4b,   0x4a,   0x49,   0x48,
0x47,   0x46,   0x45,   0x44,   0x43,   0x42,   0x41,   0x40,
0x3f,   0x3e,   0x3d,   0x3c,   0x3b,   0x3a,   0x39,   0x38,
0x37,   0x35,   0x36,   0x34,   0x33,   0x32,   0x31,   0x30,
0x2f,   0x2d,   0x2e,   0x2c,   0x2b,   0x2a,   0x29,   0x28,
0x27,   0x26,   0x25,   0x24,   0x23,   0x22,   0x20,   0x21,
0x1f,   0x1e,   0x1d,   0x1c,   0x1b,   0x1a,   0x19,   0x18,
0x17,   0x16,   0x15,   0x14,   0x13,   0x12,   0x11,   0x10,
0x0f,   0x0e,   0x0d,   0x0c,   0x0b,   0x0a,   0x09,   0x08,
0x05,   0x07,   0x06,   0x04,   0x03,   0x02,   0x01,   0x00,

0x00,   0x01,   0x02,   0x03,   0x04,   0x05,   0x06,   0x07,
0x08,   0x09,   0x0c,   0x0a,   0x0b,   0x0d,   0x0e,   0x0f,
0x10,   0x11,   0x12,   0x13,   0x14,   0x15,   0x16,   0x17,
0x18,   0x19,   0xla,   0x1b,   0x1c,   0xld,   0xle,   0xlf,
0x20,   0x21,   0x22,   0x23,   0x24,   0x25,   0x26,   0x27,
0x28,   0x29,   0x2a,   0x2b,   0x2c,   0x2d,   0x2e,   0x2f,
0x30,   0x31,   0x32,   0x33,    0x34,  0x35,   0x36,   0x37,
0x38,   0x39,   0x3a,   0x3b,   0x3c,   0x3d,   0x3e,   0x3f,
0x40,   0x41,   0x42,   0x43,   0x44,   0x45,   0x46,   0x47,
0x48,   0x49,   0x4a,   0x4b，  0x4c,   0x4d,   0x4e,   0x4f,
```

```
0x50,  0x51,  0x52,  0x53,  0x54,  0x55,  0x56,  0x57,
0x58,  0x59,  0x5a,  0x5b,  0x5c,  0x5d,  0x5e,  0x5f,
0x60,  0x61,  0x62,  0x63,  0x64,  0x65,  0x66,  0x67,
0x68,  0x69,  0x6a,  0x6b,  0x6c,  0x6d,  0x6e,  0x6f,
0x70,  0x71,  0x72,  0x73,  0x74,  0x75,  0x76,  0x77,
0x78,  0x79,  0x7a,  0x7b,  0x7c,  0x7d,  0x7e,  0x7f,
0x80,  0x81,  0x82,  0x83,  0x84,  0x85,  0x86,  0x87,
0x88,  0x89,  0x8a,  0x8b,  0x8c,  0x8d,  0x8e,  0x8f,
0x90,  0x91,  0x92,  0x93,  0x94,  0x95,  0x96,  0x97,
0x98,  0x99,  0x9a,  0x9b,  0x9c,  0x9d,  0x9e,  0x9f,
0xa0,  0xal,  0xa2,  0xa3,  0xa4,  0xa5,  0xa6,  0xa7,
0xa8,  0xa9,  0xaa,  0xab,  0xac,  0xad,  0xae,  0xaf,
0xb0,  0xb1,  0xb2,  0xb3,  0xb4,  0xb5,  0xb6,  0xb7,
0xb8,  0xb9,  0xba,  0xbb,  0xbc，0xbd,  0xbe,  0xbf,
0xc0,  0xc1,  0xc2,  0xc3,  0xe4,  0xc5,  0xc6,  0xe7,
0xc8,  0xc9,  0xca,  0xcb,  0xcc,  0xcd,  0xce,  0xef,
0xd0,  0xd1,  0xd2,  0xd3,  0xd4,  0xd5,  0xd6,  0xd7,
0xd8,  0xd9,  0xda,  0xdb,  0xdc,  0xdd,  0xde,  0xdf,
0xe0,  0xe1,  0xe2,  0xe3,  0xe4,  0xe5,  0xe6,  0xe7,
0xe8,  0xe9,  0xea,  0xeb,  0xec,  0xed,  0xee,  0xef,
0xf0,  0xf1,  0xf2,  0xf3,  0xf4,  0xf5,  0xf6,  0xf7,
0xf8,  0xf9,  0xfa,  0xfb,  0xfc,  0xfd,  0xfe,  0xff,

0xff,  0xfe,  0xfd,  0xfc,  0xfb,  0xfa,  0xf9,  0xf8,
0xf7,  0xf6,  0xf5,  0xf4,  0xf3,  0xf2,  0xf1,  0xf0,
0xef,  0xee,  0xed,  0xec,  0xeb,  0xea,  0xe9,  0xe8,
0xe7,  0xe6,  0xe5,  0xe4,  0xe3,  0xe2,  0xel,  0xe0,
0xdf,  0xde,  0xdd,  0xdc,  0xdb,  0xda,  0xd9,  0xd8,
0xd7,  0xd6,  0xd5,  0xd4,  0xd3,  0xd2,  0xd1,  0xd0,
0xef,  0xce,  0xcd,  0xcc,  0xcb,  0xca,  0xc9,  0xc8,
0xe7,  0xc6,  0xc5，0xc4,  0xc3,  0xc2,  0xc1,  0xe0,
0xbf,  0xbe,  0xbd,  0xbc,  0xbb,  0xba,  0xb9,  0xb8,
0xb7,  0xb6,  0xb5,  0xb4,  0xb3,  0xb2,  0xb1,  0xb0,
0xaf,  0xae,  0xad,  0xac,  0xab,  0xaa,  0xa9,  0xa8,
0xa7,  0xa6,  0xa5,  0xa4,  0xa3,  0xa2,  0xa1,  0xa0,
0x9f,  0x9e,  0x9d,  0x9c,  0x9b,  0x9a,  0x99,  0x98,
0x97,  0x96,  0x95,  0x94,  0x93,  0x92,  0x91,  0x90,
0x8f,  0x8e,  0x8d,  0x8c,  0x8b,  0x8a,  0x89,  0x88,
0x87,  0x86,  0x85,  0x84,  0x83,  0x82,  0x81,  0x80,
0x7f,  0x7e,  0x7d,  0x7c,  0x7b,  0x7a,  0x79,  0x78,
0x77,  0x76,  0x75,  0x74,  0x73,  0x72,  0x71,  0x70,
0x6f,  0x6e,  0x6d,  0x6c,  0x6b,  0x6a,  0x69,  0x68,
0x67,  0x66,  0x65,  0x64,  0x63,  0x62,  0x61,  0x60,
0x5f,  0x5e,  0x5d,  0x5c,  0x5b,  0x5a,  0x59,  0x58,
0x57,  0x56,  0x55,  0x54,  0x53,  0x52,  0x51,  0x50,
0x4f,  0x4e,  0x4d,  0x4c,  0x4b,  0x4a,  0x49,  0x48,
```

```
        0x47,   0x46,   0x45,   0x44,   0x43,   0x42,   0x41,   0x40,
        0x3f,   0x3e,   0x3d,   0x3c,   0x3b,   0x3a,   0x39,   0x38,
        0x37,   0x35,   0x36,   0x34,   0x33,   0x32,   0x31,   0x30,
        0x2f,   0x2d,   0x2e,   0x2c,   0x2b,   0x2a,   0x29,   0x28,
        0x27,   0x26,   0x25,   0x24,   0x23,   0x22,   0x20,   0x21,
        0x1f,   0x1e,   0x1d,   0x1c,   0x1b,   0x1a,   0x19,   0x18,
        0x17,   0x16,   0x15,   0x14,   0x13,   0x12,   0x11,   0x10,
        0x0f,   0x0e,   0x0d,   0x0c,   0x0b,   0x0a,   0x09,   0x08,
        0x05,   0x07,   0x06,   0x04,   0x03,   0x02,   0x01,   0x00,
    };
    ERROR_LED = 1;
    OK_LED = 1;
    for(array_point=0; array_point<1024; array_point++)
    {
        if(Test_array_one[array_point]!=Test_array_two[array_point])
        {
            ERROR_LED = 0;
            OK_LED = 1;
            break;
        }
        else
        {
            OK_LED = 0;
            ERROR_LED = 1;
        }
    }
    while(1);
}
```

14.2.4 外部 64K 数据总线- 可外部扩展 64K 字节的数据存储器或外围设备

STC15 系列 40-pin 及其以上的单片机具有扩展 64KB 外部数据存储器和 I/O 口的能力。访问外部数据存储器期间，WR 或 RD 信号要有效。STC15 系列单片机新增了一个控制外部 64K 字节数据总线速度的特殊功能寄存器----BUS_SPEED，该寄存器的格式如下。

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
BUS_SPEED	A1H	Bus-Speed Control	-	-	-	-	-	-	EXRTS[1:0]		xxxx,xx10

EXRTS (Extend Ram Timing Selector)

0 0 ： Setup / Hold / Read and Write Duty ← 1 clock cycle; EXRAC ← 1
0 1 ： Setup / Hold / Read and Write Duty ← 2 clock cycle; EXRAC ← 2
1 0 ： Setup / Hold / Read and Write Duty ← 4 clock cycle; EXRAC ← 4
1 1 ： Setup / Hold / Read and Write Duty ← 8 clock cycle; EXRAC ← 8

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR	8EH	Auxiliary Register	T0x12	T1x12	UAR_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000,0001

EXTRAM：Internal/External RAM access 内部/外部 RAM 存取

0：内部扩展的 EXT_RAM 可以存取
STC15W4K32S4 系列单片机
在 00H 到 EFFH 单元(3840 字节)，使用 MOVX @DPTR 指令访问，超过 F00H(含 F00H 单元)的地址空间总是访问外部数据存储器，MOVX @Ri 只能访问 00H 到 FFH 单元

1：External data memory access.外部数据存储器存取
禁止访问内部扩展 RAM，此时 MOVX @DPTR / MOVX @Ri 的使用同普通 8052 单片机

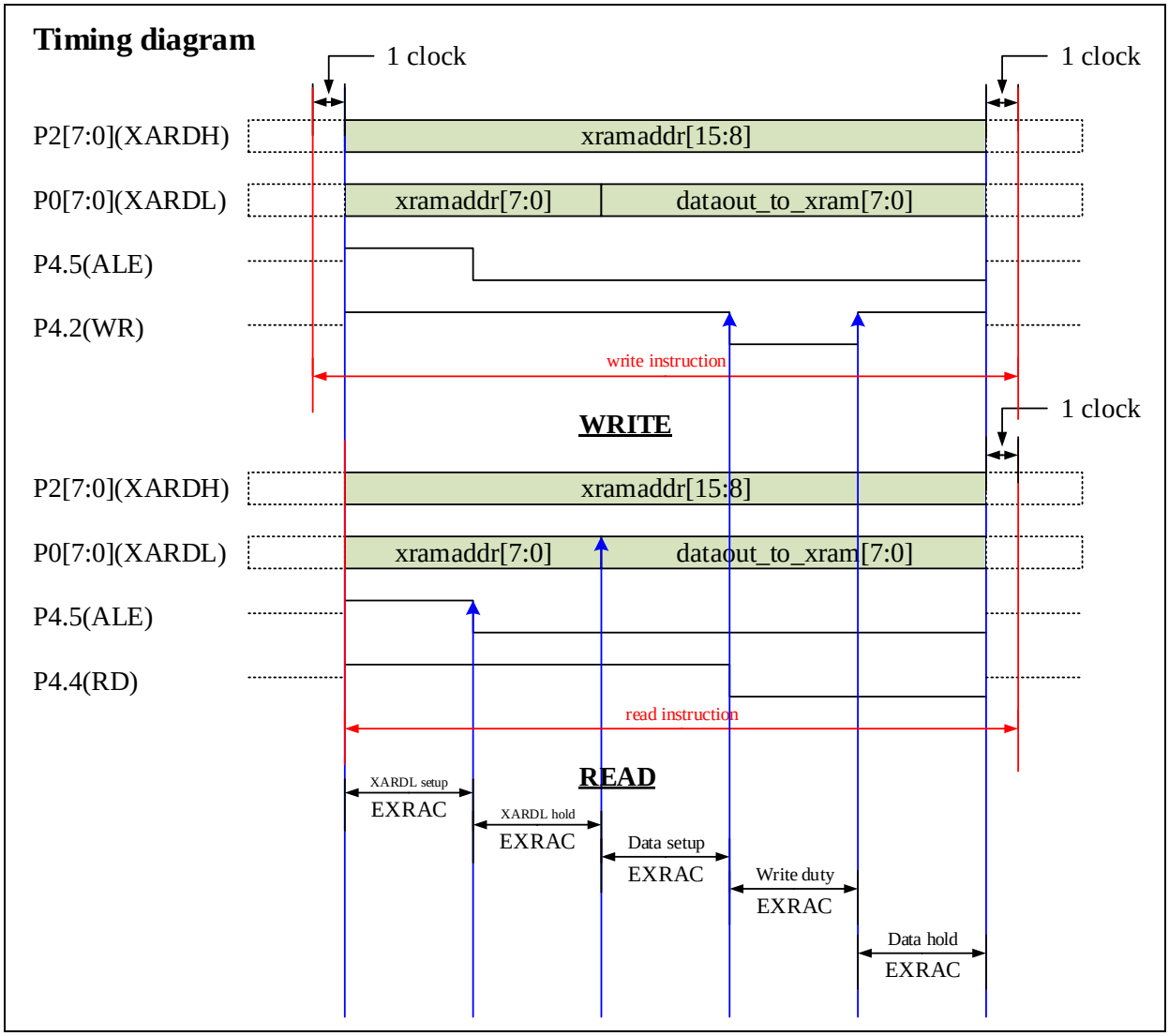
助记符	功能说明	所需时钟	条件 (以 STC15W4K32S4 系列为例， 即假定片内扩展 RAM 为 3840 字节)
MOVX @DPTR, A	将累加器 A 的内容送入逻辑上在片外、物理上在片内的扩展 RAM(16 位地址)中，写操作	3	DPTR 的内容为 0000H ~ 0EFFH 即(3840 字节 = [4096-256])
MOVX A, @DPTR	将逻辑上在片外、物理上在片内的扩展 RAM(16 位地址)的内容送入累加器 A 中，读操作	2	DPTR 的内容为 0000H ~ 0EFFH 即(3840 字节 = [4096-256])
MOVX @Ri, A	将累加器 A 的内容送入逻辑上在片外、物理上在片内的扩展 RAM(8 位地址)中，写操作	4	EXTRAM=0
MOVX A, @Ri	将逻辑上在片外、物理上在片内的扩展 RAM(8 位地址)的内容送入累加器 A 中，读操作	3	EXTRAM=0
MOVX @Ri, A	将累加器 A 的内容送入逻辑上在片外、物理上也在片外的扩展 RAM(8 位地址)中，写操作	8	EXRTS[1:0] = [0,0], EXTRAM=1
MOVX A, @Ri	将逻辑上在片外、物理上也在片外的扩展 RAM(8 位地址)的内容送入累加器 A 中，读操作	7	EXRTS[1:0] = [0,0], EXTRAM=1
MOVX @Ri, A	将累加器 A 的内容送入逻辑上在片外、物理上也在片外的扩展 RAM(8 位地址)中，写操作	13	EXRTS[1:0] = [0,1], EXTRAM=1

助记符	功能说明	所需时钟	条件 (以 STC15W4K32S4 系列为例， 即假定片内扩展 RAM 为 3840 字节)
MOVX A, @Ri	将逻辑上在片外、物理上也在片外的扩展 RAM(8 位地址)的内容送入累加器 A 中，读操作	12	EXRTS[1:0] = [0,1], EXTRAM=1
MOVX @Ri, A	将累加器 A 的内容送入逻辑上在片外、物理上也在片外的扩展 RAM(8 位地址)中，写操作	23	EXRTS[1:0] = [1,0], EXTRAM=1
MOVX A, @Ri	将逻辑上在片外、物理上也在片外的扩展 RAM(8 位地址)的内容送入累加器 A 中，读操作	22	EXRTS[1:0] = [1,0], EXTRAM=1
MOVX @Ri, A	将累加器 A 的内容送入逻辑上在片外、物理上也在片外的扩展 RAM(8 位地址)中，写操作	43	EXRTS[1:0] = [1,1], EXTRAM=1
MOVX A, @Ri	将逻辑上在片外、物理上也在片外的扩展 RAM(8 位地址)的内容送入累加器 A 中，读操作	42	EXRTS[1:0] = [1,1], EXTRAM=1
注意：对于传统 8051 单片机，Ri 只能取 R0 和 R1，STC15 系列单片机与传统 8051 单片机一样，Ri 也只能为 R0 或 R1，即上表中 Ri 中的 i=0,1。			
MOVX @DPTR, A	将累加器 A 的内容送入逻辑上在片外、物理上也在片外的扩展 RAM(16 位地址)中，写操作	7	EXRTS[1:0] = [0,0], DPTR>=3840 即(4096-256) or EXTRAM=1
MOVX A, @DPTR	将逻辑上在片外、物理上也在片外的扩展 RAM(16 位地址)的内容送入累加器 A 中，读操作	6	EXRTS[1:0] = [0,0], DPTR>=3840 即(4096-256) or EXTRAM=1
MOVX @DPTR, A	将累加器 A 的内容送入逻辑上在片外、物理上也在片外的扩展 RAM(16 位地址)中，写操作	12	EXRTS[1:0] = [0,1], DPTR>=3840 即(4096-256) or EXTRAM=1
MOVX A, @DPTR	将逻辑上在片外、物理上也在片外的扩展 RAM(16 位地址)的内容送入累加器 A 中，读操作	11	EXRTS[1:0] = [0,1], DPTR>=3840 即(4096-256) or EXTRAM=1
MOVX @DPTR, A	将累加器 A 的内容送入逻辑上在片外、物理上也在片外的扩展 RAM(16 位地址)中，写操作	22	EXRTS[1:0] = [1,0], DPTR>=3840 即(4096-256) or EXTRAM=1
MOVX A, @DPTR	将逻辑上在片外、物理上也在片外的扩展 RAM(16 位地址)的内容送入累加器 A 中，读操作	21	EXRTS[1:0] = [1,0], DPTR>=3840 即(4096-256) or EXTRAM=1
MOVX @DPTR, A	将累加器 A 的内容送入逻辑上在片外、物理上也在片外的扩展 RAM(16 位地址)中，写操作	42	EXRTS[1:0] = [1,1], DPTR>=3840 即(4096-256) or EXTRAM=1
MOVX A, @DPTR	将逻辑上在片外、物理上也在片外的扩展 RAM(16 位地址)的内容送入累加器 A 中，读操作	41	EXRTS[1:0] = [1,1], DPTR>=3840 即(4096-256) or EXTRAM=1

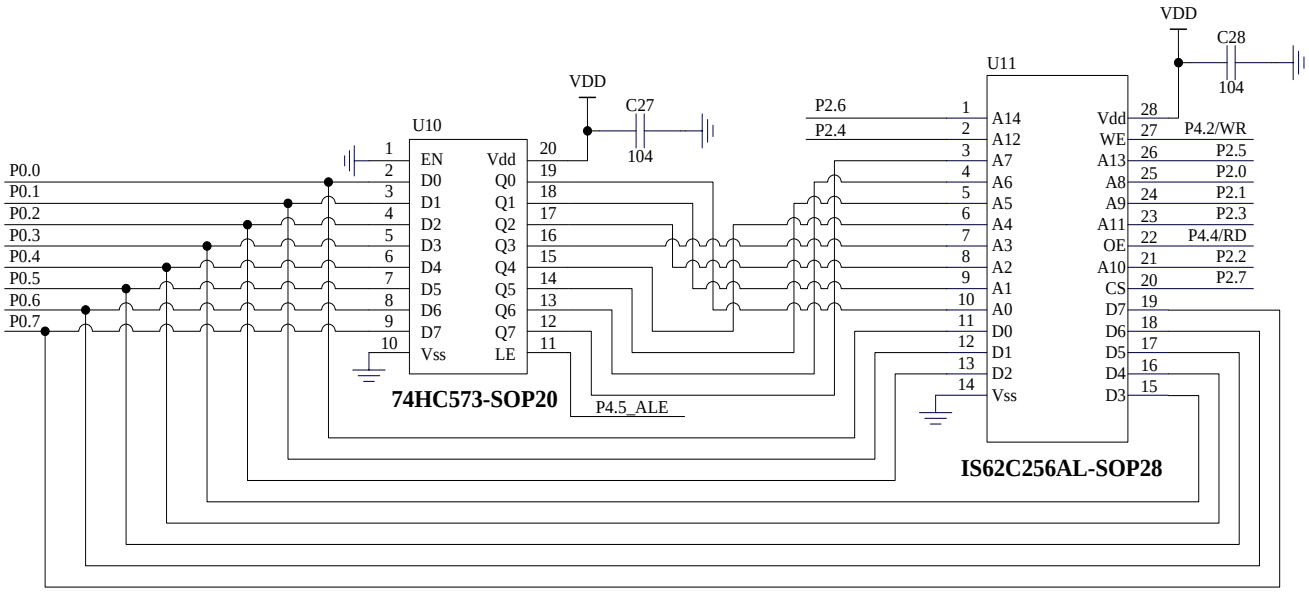
访问片外扩展 RAM 指令所需时钟计算公式：

MOVX @R0/R1		MOVX @DPTR	
write(写操作):	5×N+3	write(写操作):	5×N+2
read(读操作):	5×N+2	read(读操作):	5×N+1

当 EXRTS[1:0] = [0,0]时，上式中 N=1;
当 EXRTS[1:0] = [0,1]时，上式中 N=2;
当 EXRTS[1:0] = [1,0]时，上式中 N=4;
当 EXRTS[1:0] = [1,1]时，上式中 N=8.
可见，对于 STC15 系列单片机，访问片外扩展 RAM 指令的速度是可调的


Timing diagram
1 clock
1 clock
P2[7:0](XARDH)
xramaddr[15:8]
P0[7:0](XARDL)
xramaddr[7:0]
dataout_to_xram[7:0]
P4.5(ALE)
P4.2(WR)
write instruction
WRITE
1 clock
P2[7:0](XARDH)
xramaddr[15:8]
P0[7:0](XARDL)
xramaddr[7:0]
dataout_to_xram[7:0]
P4.5(ALE)
P4.4(RD)
read instruction
READ
XARDL setup
EXRAC
XARDL hold
EXRAC
Data setup
EXRAC
Write duty
EXRAC
Data hold
EXRAC

14.2.5 利用并行总线扩展外部 32K SRAM 的应用线路图



注意：由于历史原因，IS62C256AL-SOP28封装比STC的SOP28封装要宽

14.3 特殊功能寄存器(SFRs)

特殊功能寄存器(SFR)是用来对片内各功能模块进行管理、控制、监视的控制寄存器和状态寄存器，是一个特殊功能的 RAM 区。STC15 系列单片机内的特殊功能寄存器（SFR）与高 128 字节 RAM 共用相同的地址范围，都使用 80H ~ FFH，特殊功能寄存器（SFR）必须用直接寻址指令访问。

STC15 系列单片机的特殊功能寄存器名称及地址映象如下表所示

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
0F8H	P7 1111,1111	CH 0000,0000	CCAP0H 0000,0000	CCAP1H 0000,0000	CCAP2H 0000,0000				0FFH
0F0H	B 0000,0000	PWMCFG 0000,0000	PCA_PWM0 00xx,xx00	PCA_PWM1 00xx,xx00	PCA_PWM2 00xx,xx00	PWMCR 0000,0000	PWMIF x000,0000	PWMFDCR xx00,0000	0F7H
0E8H	P6 1111,1111	CL 0000,0000	CCAP0L 0000,0000	CCAP1L 0000,0000	CCAP2L 0000,0000				0EFH
0E0H	ACC 0000,0000	P7M1 0000,0000	P7M0 0000,0000						0E7H
0D8H	CCON 00xx,0000	CMOD 0xxx,x000	CCAPM0 x000,0000	CCAPM1 x000,0000	CCAPM2 x000,0000				0DFH
0D0H	PSW 0000,00x0	T4T3M 0000,0000	T4H RL_TH4 0000,0000	T4L RL_TL4 0000,0000	T3H RL_TH3 0000,0000	T3L RL_TL3 0000,0000	T2H RL_TH2 0000,0000	T2L RL_TL2 0000,0000	0D7H
0C8H	P5 xxxx,1111	P5M1 xxxx,0000	P5M0 xxxx,0000	P6M1 0000,0000	P6M0 0000,0000	SPSTAT 00xx,xxxx	SPCTL 0000,0100	SPDAT 0000,0000	0CFH
0C0H	P4 1111,1111	WDT_CONTR 0x00,0000	IAP_DATA 1111,1111	IAP_ADDRH 0000,0000	IAP_ADDRL 0000,0000	IAP_CMD xxxx,xx00	IAP_TRIG xxxx,xxxx	IAP_CONTR 0000,0000	0C7H
0B8H	IP x0x0,0000	SADEN	P_SW2 xxxx,x000		ADC_CONTR 0000,0000	ADC_RES 0000,0000	ADC_RESL 0000,0000		0BFH
0B0H	P3 1111,1111	P3M1 0000,0000	P3M0 0000,0000	P4M1 0000,0000	P4M0 0000,0000	IP2 xxx0,0000	IP2H xxxx,xx00	IPH 0000,0000	0B7H
0A8H	IE 0000,0000	SADDR	WKTCL WKTCL_CNT 0111,1111	WKTCH WKTCH_CNT 0111,1111	S3CON 0000,0000	S3BUF xxxx,xxxx		IE2 x000,0000	0AFH
0A0H	P2 1111,1111	BUS_SPEED xxxx,xx10	AUXR1 P_SW1 0100,0000	Don't use	Don't use	Don't use		Don't use	0A7H
098H	SCON 0000,0000	SBUF xxxx,xxxx	S2CON 0100,0000	S2BUF xxxx,xxxx	Don't use	P1ASF 0000,0000	Don't use	Don't use	09FH
090H	P1 1111,1111	P1M1 0000,0000	P1M0 0000,0000	P0M1 0000,0000	P0M0 0000,0000	P2M1 0000,0000	P2M0 0000,0000	CLK_DIV PCON2	097H
088H	TCON 0000,0000	TMOD 0000,0000	TL0 RL_TL0 0000,0000	TL1 RL_TL1 0000,0000	TH0 RL_TH0 0000,0000	TH1 RL_TH1 0000,0000	AUXR 0000,0001	INT_CLKO AUXR2 0000,0000	08FH
080H	P0 1111,1111	SP 0000,0111	DPL 0000,0000	DPH 0000,0000	S4CON 0000,0000	S4BUF xxxx,xxxx	0011,0000	PCON	087H
	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	

可位寻址 不可位寻址

注意：寄存器地址能够被 8 整除的才可以进行位操作，不能够被 8 整除的不可以进行位操作

符号		描述	地址	位地址及符号								复位值
				MSB							LSB	
P0		Port 0	80H	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0	1111 1111B
SP		堆栈指针	81H									0000 0111B
DPTR	DPL	数据指针(低)	82H									0000 0000B
	DPH	数据指针(高)	83H									0000 0000B
S4CON		串口 4 控制寄存器	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000 0000B
S4BUF		串口 4 数据缓冲器	85H									xxxx xxxxB
PCON		电源控制寄存器	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011 0000B
TCON		定时器控制寄存器	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000 0000B
TMOD		定时器工作方式寄存器	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	0000 0000B
TL0		定时器 0 低 8 位寄存器	8AH									0000 0000B
TL1		定时器 1 低 8 位寄存器	8BH									0000 0000B
TH0		定时器 0 高 8 位寄存器	8CH									0000 0000B
TH1		定时器 1 高 8 位寄存器	8DH									0000 0000B
AUXR		辅助寄存器	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000 0001B
INT_CLKO AUXR2		外部中断允许和时钟输出寄存器	8FH	-	EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	T0CLKO	x000 0000B
P1		Port 1	90H	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	1111 1111B
P1M1		P1 口模式配置寄存器 1	91H									0000 0000B
P1M0		P1 口模式配置寄存器 0	92H									0000 0000B
P0M1		P0 口模式配置寄存器 1	93H									0000 0000B
P0M0		P0 口模式配置寄存器 0	94H									0000 0000B
P2M1		P2 口模式配置寄存器 1	95H									0000 0000B
P2M0		P2 口模式配置寄存器 0	96H									0000 0000B
CLK_DIV PCON2		时钟分频寄存器	97H	MCKO_S1	MCKO_S1	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000 0000B
SCON		串口 1 控制寄存器	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000 0000B
SBUF		串口 1 数据缓冲器	99H									xxxx xxxxB
S2CON		串口 2 控制寄存器	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0100 0000B
S2BUF		串口 2 数据缓冲器	9BH									xxxx xxxxB
P1ASF		P1 Analog Function Configure register	9DH	P17ASF	P16ASF	P15ASF	P14ASF	P13ASF	P12ASF	P11ASF	P10ASF	0000 0000B
P2		Port 2	A0H	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0	1111 1111B
BUS_SPEED		Bus-Speed Control	A1H	-	-	-	-	-	-	EXRTS[1:0]		xxxx xx10B
AUXR1 P_SW1		辅助寄存器 1	A2H	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000 0000B
IE		中断允许寄存器	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000 0000B
SADDR		从机地址控制寄存器	A9H									0000 0000B
WKTCL WKTCL_CNT		掉电唤醒专用定时器控制寄存器低 8 位	AAH									1111 1111B
WKTCH WKTCH_CNT		掉电唤醒专用定时器控制寄存器高 8 位	ABH	WKTEN								0111 1111B
S3CON		串口 3 控制寄存器	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000
S3BUF		串口 3 数据缓冲器	ADH									xxxx,xxxx
IE2		中断允许寄存器	AFH		ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000 0000B
P3		Port 3	B0H	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0	1111 1111B

符号	描述	地址	位地址及符号								复位值
			MSB							LSB	
P3M1	P3 口模式配置寄存器 1	B1H									0000 0000B
P3M0	P3 口模式配置寄存器 0	B2H									0000 0000B
P4M1	P4 口模式配置寄存器 1	B3H									0000 0000B
P4M0	P4 口模式配置寄存器 0	B4H									0000 0000B
IP2	第二中断优先级低字节寄存器	B5H	-	-	-	PX4	PPWMFD	PPWM	PSPI	PS2	xxx0 0000B
IP	中断优先级寄存器	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	0000 0000B
SADEN	从机地址掩模寄存器	B9H									0000 0000B
P_SW2	外围设备功能切换控制寄存器	BAH	-	-	-	-	-	S4_S	S3_S	S2_S	xxxx x000B
ADC_CONTR	A/D 转换控制寄存器	BCH	ADC_POWER	SPEED1	SPEED0	ADC_FLAG	ADC_START	CHS2	CHS1	CHS0	0000 0000B
ADC_RES	A/D 转换结果高 8 位寄存器	BDH									0000 0000B
ADC_RESL	A/D 转换结果低 2 位寄存器	BEH									0000 0000B
P4	Port 4	C0H	P4.7	P4.6	P4.5	P4.4	P4.3	P4.2	P4.1	P4.0	1111 1111B
WDT_CONTR	看门狗控制寄存器	C1H	WDT_FLAG	-	EN_WDT	CLR_WDT	IDLE_WDT	PS2	PS1	PS0	0x00 0000B
IAP_DATA	ISP/IAP 数据寄存器	C2H									1111 1111B
IAP_ADDRH	ISP/IAP 高 8 位地址寄存器	C3H									0000 0000B
IAP_ADDRL	ISP/IAP 低 8 位地址寄存器	C4H									0000 0000B
IAP_CMD	ISP/IAP 命令寄存器	C5H	-	-	-	-	-	-	MS1	MS0	xxxx xx00B
IAP_TRIG	ISP/IAP 命令触发寄存器	C6H									xxxx xxxxB
IAP_CONTR	ISP/IAP 控制寄存器	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	WT2	WT1	WT0	0000 x000B
P5	Port 5	C8H	-	-	P5.5	P5.4	P5.3	P5.2	P5.1	P5.0	xx11 1111B
P5M1	P5 口模式配置寄存器 1	C9H									xxx0 0000B
P5M0	P5 口模式配置寄存器 0	CAH									xxx0 0000B
P6M1	P6 口模式配置寄存器 1	CBH									
P6M0	P6 口模式配置寄存器 0	CCH									
SPSTAT	SPI 状态寄存器	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx xxxxB
SPCTL	SPI 控制寄存器	CEH	SSIG	SPEN	DORD	MSTR	CPOL	CAPHA	SPR1	SPR0	0000 0100B
SPDAT	SPI 数据寄存器	CFH									0000 0000B
PSW	程序状态字寄存器	D0H	CY	AC	F0	RS1	RS0	OV	-	P	0000 00x0B
T4T3M	T4 和 T3 的控制寄存器	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO	0000 0000B
T4H	定时器 4 高 8 位寄存器	D2H									0000 0000B
T4L	定时器 4 低 8 位寄存器	D3H									0000 0000B
T3H	定时器 3 高 8 位寄存器	D4H									0000 0000B
T3L	定时器 3 低 8 位寄存器	D5H									0000 0000B
T2H	定时器 2 高 8 位寄存器	D6H									0000 0000B
T2L	定时器 2 低 8 位寄存器	D7H									0000 0000B
CCON	PCA 控制寄存器	D8H	CF	CR	-	-	-	CCF2	CCF1	CCF0	00xx 0000B

符号	描述	地址	位地址及符号								复位值
			MSB							LSB	
CMOD	PCA 模式寄存器	D9H	CIDL	-	-	-	-	CPS1	CPS0	ECF	0xxx x000B
CCAPM0	PCA Module 0 Mode Register	DAH	-	ECOM0	CAPP0	CAPN0	MAT0	TOG0	PWM0	ECCF0	x000 0000B
CCAPM1	PCA Module 1 Mode Register	DBH	-	ECOM1	CAPP1	CAPN1	MAT1	TOG1	PWM1	ECCF1	x000 0000B
CCAPM2	PCA Module 2 Mode Register	DCH	-	ECOM2	CAPP2	CAPN2	MAT2	TOG2	PWM2	ECCF2	x000 0000B
ACC	累加器	E0H									0000 0000B
P7M1	P7 口模式配置寄存器 1	E1H									0000 0000B
P7M0	P7 口模式配置寄存器 0	E2H									0000 0000B
P6	Port 6	E8H									1111 1111B
CL	PCA Base Timer Low	E9H									0000 0000B
CCAP0L	PCA Module-0 Capture Register Low	EAH									0000 0000B
CCAP1L	PCA Module-1 Capture Register Low	EBH									0000 0000B
CCAP2L	PCA Module-2 Capture Register Low	ECH									0000 0000B
B	B 寄存器	F0H									0000 0000B
PCA_PWM0	PCA PWM Mode Auxiliary Register 0	F2H	EBS0_1	EBS0_0	-	-	-	-	EPC0H	EPC0L	xxxx xx00B
PCA_PWM1	PCA PWM Mode Auxiliary Register 1	F3H	EBS1_1	EBS1_0	-	-	-	-	EPC1H	EPC1L	xxxx xx00B
PCA_PWM2	PCA PWM Mode Auxiliary Register 2	F4H	EBS2_1	EBS2_0	-	-	-	-	EPC2H	EPC2L	xxxx xx00B
P7	Port 7	F8H									1111 1111B
CH	PCA Base Timer High	F9H									0000 0000B
CCAP0H	PCA Module-0 Capture Register High	FAH									0000 0000B
CCAP1H	PCA Module-1 Capture Register High	FBH									0000 0000B
CCAP2H	PCA Module-2 Capture Register High	FCH									0000 0000B

扩展的特殊功能寄存器

符号	描述	地址	位址及符号								初始值
			B7	B6	B5	B4	B3	B2	B1	B0	
P_SW2	端口配置寄存器	BAH	EAXSFR	0	0	0	-	S4_S	S3_S	S2_S	0000,0000
PWMCFG	PWM 配置	F1H	-	CBTADC	C7INI	C6INI	C5INI	C4INI	C3INI	C2INI	0000,0000
PWMCR	PWM 控制	F5H	ENPWM	ECBI	ENC7O	ENC6O	ENC5O	ENC4O	ENC3O	ENC2O	0000,0000
PWMIF	PWM 中断标志	F6H	-	CBIF	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	x000,0000
PWMFDCR	PWM 外部异常控制	F7H	-	-	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	xx00,0000
PWMCH	PWM 计数器高位	FFF0H	-	PWMCH[14:8]							x000,0000
PWMCL	PWM 计数器低位	FFF1H	PWMCL[7:0]								0000,0000
PWMCKS	PWM 时钟选择	FFF2H	-	-	-	SELT2	PS[3:0]				xxx0,0000
PWM2T1H	PWM2T1 计数高位	FF00H	-	PWM2T1H[14:8]							x000,0000
PWM2T1L	PWM2T1 计数低位	FF01H	PWM2T1L[7:0]								0000,0000
PWM2T2H	PWM2T2 计数高位	FF02H	-	PWM2T2H[14:8]							x000,0000
PWM2T2L	PWM2T2 计数低位	FF03H	PWM2T2L[7:0]								0000,0000
PWM2CR	PWM2 控制	FF04H	-	-	-	-	PWM2_PS	EPWM2I	EC2T2SI	EC2T1SI	xxxx,0000
PWM3T1H	PWM3T1 计数高位	FF10H	-	PWM3T1H[14:8]							x000,0000

符号	描述	地址	位址及符号								初始值
			B7	B6	B5	B4	B3	B2	B1	B0	
PWM3T1L	PWM3T1 计数低位	FF11H	PWM3T1L[7:0]								0000,0000
PWM3T2H	PWM3T2 计数高位	FF12H	-	PWM3T2H[14:8]							x000,0000
PWM3T2L	PWM3T2 计数低位	FF13H	PWM3T2L[7:0]								0000,0000
PWM3CR	PWM3 控制	FF14H	-	-	-	-	PWM3_PS	EPWM3I	EC3T2SI	EC3T1SI	xxxx,0000
PWM4T1H	PWM4T1 计数高位	FF20H	-	PWM4T1H[14:8]							x000,0000
PWM4T1L	PWM4T1 计数低位	FF21H	PWM4T1L[7:0]								0000,0000
PWM4T2H	PWM4T2 计数高位	FF22H	-	PWM4T2H[14:8]							x000,0000
PWM4T2L	PWM4T2 计数低位	FF23H	PWM4T2L[7:0]								0000,0000
PWM4CR	PWM4 控制	FF24H	-	-	-	-	PWM4_PS	EPWM4I	EC4T2SI	EC4T1SI	xxxx,0000
PWM5T1H	PWM5T1 计数高位	FF30H	-	PWM5T1H[14:8]							x000,0000
PWM5T1L	PWM5T1 计数低位	FF31H	PWM5T1L[7:0]								0000,0000
PWM5T2H	PWM5T2 计数高位	FF32H	-	PWM5T2H[14:8]							x000,0000
PWM5T2L	PWM5T2 计数低位	FF33H	PWM5T2L[7:0]								0000,0000
PWM5CR	PWM5 控制	FF34H	-	-	-	-	PWM5_PS	EPWM5I	EC5T2SI	EC5T1SI	xxxx,0000
PWM6T1H	PWM6T1 计数高位	FF40H	-	PWM6T1H[14:8]							x000,0000
PWM6T1L	PWM6T1 计数低位	FF41H	PWM6T1L[7:0]								0000,0000
PWM6T2H	PWM6T2 计数高位	FF42H	-	PWM6T2H[14:8]							x000,0000
PWM6T2L	PWM6T2 计数低位	FF43H	PWM6T2L[7:0]								0000,0000
PWM6CR	PWM6 控制	FF44H	-	-	-	-	PWM6_PS	EPWM6I	EC6T2SI	EC6T1SI	xxxx,0000
PWM7T1H	PWM7T1 计数高位	FF50H	-	PWM7T1H[14:8]							x000,0000
PWM7T1L	PWM7T1 计数低位	FF51H	PWM7T1L[7:0]								0000,0000
PWM7T2H	PWM7T2 计数高位	FF52H	-	PWM7T2H[14:8]							x000,0000
PWM7T2L	PWM7T2 计数低位	FF53H	PWM7T2L[7:0]								0000,0000
PWM7CR	PWM7 控制	FF54H	-	-	-	-	PWM7_PS	EPWM7I	EC7T2SI	EC7T1SI	xxxx,0000

下面简单的介绍一下普通 8051 单片机常用的一些寄存器：

1.程序计数器(PC)

程序计数器 PC 在物理上是独立的，不属于 SFR 之列。PC 字长 16 位，是专门用来控制指令执行顺序的寄存器。单片机上电或复位后，PC = 0000H，强制单片机从程序的零单元开始执行程序。

2.累加器(ACC)

累加器 ACC 是 8051 单片机内部最常用的寄存器，也可写作 A。常用于存放参加算术或逻辑运算的操作数及运算结果。

3.B 寄存器

B 寄存器在乘法和除法运算中须与累加器 A 配合使用。MUL AB 指令把累加器 A 和寄存器 B 中的 8 位无符号数相乘，所得的 16 位乘积的低字节存放在 A 中，高字节存放在 B 中。DIV AB 指令用 B 除以 A，整数商存放在 A 中，余数存放在 B 中。寄存器 B 还可以用作通用暂存寄存器。

4.程序状态字(PSW)寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
PSW	D0H	name	CY	AC	F0	RS1	RS0	OV	-	P

CY： 标志位。进行加法运算时，当最高位即 B7 位有进位，或执行减法运算最高位有借位时，CY 为 1；反之为 0。

AC： 进位辅助位。进行加法运算时，当 B3 位有进位，或执行减法运算 B3 有借位时，AC 为 1；反之为 0。设置辅助进位标志 AC 的目的是为了便于 BCD 码加法、减法运算的调整。

FO： 用户标志位。

RS1、RS0：工作寄存器组的选择位。如下表

RS1	RS0	当前使用的工作寄存器组(R0 ~ R7)
0	0	0 组(00H ~ 07H)
0	1	1 组(08H ~ 0FH)
1	0	2 组(10H ~ 17H)
1	1	3 组(18H ~ 1FH)

OV： 溢出标志位

B1： 保留位

P： 奇偶标志位。该标志位始终体现累加器 ACC 中 1 的个数的奇偶性。如果累加器 ACC 中 1 的个数为奇数，则 P 置 1；当累加器 ACC 中的个数为偶数(包括 0 个)时，P 位为 0

5.堆栈指针(SP)

堆栈指针是一个 8 位专用寄存器。它指示出堆栈顶部在内部 RAM 块中的位置。系统复位后，SP 初始化位 07H，使得堆栈事实上由 08H 单元开始，考虑 08H ~ 1FH 单元分别属于工作寄存器组 1 ~ 3，若在程序设计中用到这些区，则最好把 SP 值改变为 80H 或更大的值为宜。STC15 系列单片机的堆栈是向上生长的，即将数据压入堆栈后，SP 内容增大。

6.数据指针(DPTR)

数据指针(DPTR)是一个 16 位专用寄存器，由 DPL(低 8 位)和 DPH(高 8 位)组成,地址是 82H(DPL,低字节)和 83H(DPH,高字节)。DPTR 是传统 8051 机中唯一可以直接进行 16 位操作的寄存器也可分别对 DPL 河 DPH 按字节进行操作。

如果用户所使用的 STC15 系列单片机无外部数据总线，那么该单片机只设计了一个 16 位的数据指针。相反，如果用户所使用的 STC15 系列单片机有外部数据总线，那么该单片机设计了两个 16 位的数据指针 DPRT0 和 DPTR1，这两个数据指针共用同一个地址空间，可通过设置 DPS/P_SW1.0 来选择具体被使用的数据指针。

当用户所使用的 STC15 系列单片机有外部数据总线，即该单片机设计了两个 16 位的数据指针 DPRT0 和 DPTR1 时，这两个数据指针可通过软件设置 DPS/P_SW1.0 来选择具体哪个数据指针被使用，见下面寄存器 AUXR1 的说明。

STC15 系列 8051 单片机双数据指针特殊功能寄存器

Mnemonic	Address	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary Register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000,0000

DPS： DPTR registers select bit. DPTR 寄存器选择位

0：DPTR0 is selected DPTR0 被选择

1：DPTRl is selected DPTR1 被选择

此系列单片机有两个 16-bit 数据指针，DPTR0，DPTR1。当 DPS 选择位为 0 时，选择 DPTR0，当 DPS 选择位为 1 时，选择 DPTR1。

AUXR1 特殊功能寄存器，位于 A2H 单元，其中的位不可用布尔指令快速访问。但由于 DPS 位位于 bit0，故对 AUXR1 寄存器用 INC 指令，DPS 位便会反转，由 0 变成 1 或由 1 变成 0，即可实现双数据指

针的快速切换。

应用示例供参考:

```
;新增特殊功能寄存器定义
    P_SW1       DATA        0A2H
    MOV         P_SW1, #0       ;此时 DPS 为 0，DPTR0 有效

    MOV         DPTR, #1FFH     ;置 DPTR0 为 1FFH
    MOV         A, #55H
    MOVX        @DPTR, A        ;将 1FFH 单元置为 55H

    MOV         DPTR, #2FFH     ;置 DPTR0 为 2FFH
    MOV         A, #0AAH
    MOVX        @DPTR, A        ;将 2FFH 单元置为 0AAH

    INC         P_SW1           ;此时 DPS 为 1, DPTR1 有效
    MOV         DPTR, #1FFH     ;置 DPTR1 为 1FFH
    MOVX        A, @DPTR        ;读 DPTR1 数据指针指向的 1FFH 单元的内容,累加器 A 变为 55H.

    INC         P_SW1           ;此时 DPS 为 0, DPTR0 有效
    MOVX        A, @DPTR        ;读 DPTR0 数据指针指向的 2FFH 单元的内容,累加器 A 变为 0AAH.

    INC         P_SW1           ;此时 DPS 为 1, DPTR1 有效
    MOVX        A, @DPTR        ;读 DPTR1 数据指针指向的 1FFH 单元的内容,累加器 A 变为 55H.

    INC         P_SW1           ;此时 DPS 为 0, DPTR0 有效
    MOVX        A, @DPTR        ;读 DPTR0 数据指针指向的 2FFH 单元的内容,累加器 A 变为 0AAH.
```

当用户所使用的 STC 系列单片机无外部数据总线，即该单片机只设计了一个 16 位的数据指针时，DPS/P_SW1.0 位无效。

Mnemonic	Address	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary Register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000,000x

14.4 STC15W4K32S4 系列新增特殊功能寄存器(SFRs)表

STC15W4K32S4 系列单片机的特殊功能寄存器名称及地址映象如下表所示

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
0F8H	P7 1111,1111	CH 0000,0000	CCAP0H 0000,0000	CCAP1H 0000,0000					0FFH
0F0H	B 0000,0000	PWMCFG 0000,0000	PCA_PWM0 0000,0000	PCA_PWM1 0000,0000		PWMCR 0000,0000	PWMIF 0000,0000	PWMFDCR 0000,0000	0F7H
0E8H	P6 1111,1111	CL 0000,0000	CCAP0L 0000,0000	CCAP1L 0000,0000					0EFH
0E0H	ACC 0000,0000	P7M1 0000,0000	P7M0 0000,0000				CMPCR1 0000,0000	CMPCR2 0000,1001	0E7H
0D8H	CCON 0000,0000	CMOD 0000,0000	CCAPM0 0000,0000	CCAPM1 0000,0000					0DFH
0D0H	PSW 0000,0100	T4T3M 0000,0000	T4H RL_TH4 0000,0000	T4L RL_TL4 0000,0000	T3H RL_TH3 0000,0000	T3L RL_TL3 0000,0000	T2H RL_TH2 0000,0000	T2L RL_TL2 0000,0000	0D7H
0C8H	P5 xx11,1111	P5M1 xx00,0000	P5M0 xx00,0000	P6M1 0000,0000	P6M0 0000,0000	SPSTAT 00xx,xxxx	SPCTL 0000,1100	SPDAT 1111,1111	0CFH
0C0H	P4 1111,1111	WDT_CONTR 0000,0000	IAP_DATA 0000,0000	IAP_ADDRH 0000,0000	IAP_ADDRL 0000,0000	IAP_CMD xxxx,xx00	IAP_TRIG xxxx,xxxx	IAP_CONTR 0000,0000	0C7H
0B8H	IP 0000,0000	SADEN 0000,0000	P_SW2 0000,0000		ADC_CONTR 0000,0000	ADC_RES 0000,0010	ADC_RESL 0000,0000		0BFH
0B0H	P3 1111,1111	P3M1 1000,0000	P3M0 0000,0000	P4M1 0011,0100	P4M0 0000,0000	IP2 0000,0000	IP2H 0000,0000	IPH 0000,0000	0B7H
0A8H	IE 0000,0000	SADDR	WKTCL WKTCL_CNT 1111,1111	WKTCH WKTCH_CNT 0111,1111	S3CON 1000000	S3BUF xxxx,xxxx		IE2 x000,0000	0AFH
0A0H	P2 1111,1111	BUS_SPEED 0000,0010	AUXR1 P_SW1 0000,0000	Don't use	Don't use	Don't use		Don't use	0A7H
098H	SCON 0000,0000	SBUF 0000,0000	S2CON 0100,0000	S2BUF xxxx,xxxx	Don't use	P1ASF 0000,0000	Don't use	Don't use	09FH
090H	P1 1111,1111	P1M1 1100,0000	P1M0 0001,0001	P0M1 1100,0000	P0M0 0000,0000	P2M1 1000,1110	P2M0 0000,0000	CLK_DIV PCON2 0000,0000	097H
088H	TCON 0000,0000	TMOD 0000,0000	TL0 RL_TL0 0000,0000	TL1 RL_TL1 0000,0000	TH0 RL_TH0 0000,0000	TH1 RL_TH1 0000,0000	AUXR 0000,0001	INT_CLKO AUXR2 0000,0000	08FH
080H	P0 1111,1111	SP 0000,1010	DPL 0010,0011	DPH 0000,0000	S4CON 0100,0000	S4BUF xxxx,xxxx		PCON 0011,0000	087H
	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	

可位寻址　　不可位寻址

注意：寄存器地址能够被 8 整除的才可以进行位操作，不能够被 8 整除的不可以进行位操作。

15 STC15 系列单片机的 I/O 口结构

15.1 I/O 口各种不同的工作模式及配置介绍

STC15 系列单片机最多有 62 个 I/O 口（如 64-pin 单片机）：P0.0 ~ P0.7，P1.0 ~ P1.7，P2.0 ~ P2.7，P3.0 ~ P3.7，P4.0 ~ P4.7，P5.0 ~ P5.5，P6.0 ~ P6.7，P7.0 ~ P7.7。其所有 I/O 口均可由软件配置成 4 种工作类型之一，如下表所示。4 种类型分别为：准双向口/弱上拉（标准 8051 输出模式）、推挽输出/强上拉、高阻输入（电流既不能流入也不能流出）或开漏输出功能。每个口由 2 个控制寄存器中的相应位控制每个引脚工作类型。STC15 系列单片机的 I/O 口上电复位后为准双向口/弱上拉（传统 8051 的 I/O 口）模式。每个 I/O 口驱动能力均可达到 20mA，但 40-pin 及 40-pin 以上单片机的整个芯片最大不要超过 120mA，20-pin 以上及 32-pin 以下（包括 32-pin）单片机的整个芯片最大不要超过 90mA。

I/O 口工作类型设定

P0 口设定<P0.7，P0.6，P0.5，P0.4，P0.3，P0.2，P0.1，P0.0 口>（P0 口地址：80H）

P0M1[7：0] 寄存器 P0M1 地址为 93H	P0M0[7：0] 寄存器 P0M0 地址为 94H	I/O 口模式
0	0	准双向口（传统 8051 I/O 口模式，弱上拉）， 灌电流可达 20mA，拉电流为 270μA， 由于制造误差，实际为 270uA～150uA
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（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。

```
举例：  MOV     P0M1, #10100000B
        MOV     P0M0, #11000000B
;P0.7 为开漏，P0.6 为强推挽输出，P0.5 为高阻输入，P0.4/P0.3/P0.2/P0.1/P0.0 为准双向口/弱上拉。
```

P1 口设定<P1.7，P1.6，P1.5，P1.4，P1.3，P1.2，P1.1，P1.0 口>（P1 口地址：90H）

P1M1[7：0] 寄存器 P1M1 地址为 91H	P1M0[7：0] 寄存器 P1M0 地址为 92H	I/O 口模式 （P1.x 如做 A/D 使用，需先将其设置成开漏或高阻输入）
0	0	准双向口（传统 8051 I/O 口模式，弱上拉）， 灌电流可达 20mA，拉电流为 270μA， 由于制造误差，实际为 270uA～150uA
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（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。

```
举例：  MOV     P1M1, #10100000B
        MOV     P1M0, #11000000B
;P1.7 为开漏，P1.6 为强推挽输出，P1.5 为高阻输入，P1.4/P1.3/P1.2/P1.1/P1.0 为准双向口/弱上拉。
```

P2 口设定<P2.7，P2.6，P2.5，P2.4，P2.3，P2.2，P2.1，P2.0>（P2 口地址：A0H）

P2M1[7：0] 寄存器 P2M1 地址为 95H	P2M0[7：0] 寄存器 P2M0 地址为 96H	I/O 口模式
0	0	准双向口（传统 8051 I/O 口模式，弱上拉）， 灌电流可达 20mA，拉电流为 270μA， 由于制造误差，实际为 270uA～150uA
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（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。

```
举例：  MOV   P2M1, #10100000B
        MOV   P2M0, #11000000B
```

;P2.7 为开漏，P2.6 为强推挽输出，P2.5 为高阻输入，P2.4/P2.3/P2.2/P2.1/P2.0 为准双向口/弱上拉。

P3 口设定<P3.7，P3.6，P3.5，P3.4，P3.3，P3.2，P3.1，P3.0 口>（P3 口地址：B0H）

P3M1[7：0] 寄存器 P3M1 地址为 B1H	P3M0[7：0] 寄存器 P3M0 地址为 B2H	I/O 口模式
0	0	准双向口（传统 8051 I/O 口模式，弱上拉）， 灌电流可达 20mA，拉电流为 270μA， 由于制造误差，实际为 270uA～150uA
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（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。

```
举例：  MOV   P3M1, #10100000B
        MOV   P3M0, #11000000B
```

;P3.7 为开漏，P3.6 为强推挽输出，P3.5 为高阻输入，P3.4/P3.3/P3.2/P3.1/P3.0 为准双向口/弱上拉。

P4 口设定<P4.7，P4.6，P4.5，P4.4，P4.3，P4.2，P4.1，P4.0>（P4 口地址：C0H）

P4M1[7：0] 寄存器 P4M1 地址为 B3H	P4M0[7：0] 寄存器 P4M0 地址为 B4H	I/O 口模式
0	0	准双向口（传统 8051 I/O 口模式，弱上拉）， 灌电流可达 20mA，拉电流为 270μA， 由于制造误差，实际为 270uA～150uA
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（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。

```
举例：  MOV   P4M1, #10100000B
        MOV   P4M0, #11000000B
```

;P4.7 为开漏，P4.6 为强推换输出，P4.5 为高阻输入，P4.4/P4.3/P4.2/P4.1/P4.0 为准双向口/弱上拉。

P5 口设定<X，X，P5.5，P5.4，P5.3，P5.2，P5.1，P5.0 口>（P5 口地址：C8H）

P5M1[7：0] 寄存器 P5M1 地址为 C9H	P5M0[7：0] 寄存器 P5M0 地址为 CAH	I/O 口模式
0	0	准双向口（传统 8051 I/O 口模式，弱上拉）， 灌电流可达 20mA，拉电流为 270μA， 由于制造误差，实际为 270uA～150uA
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（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。

```
举例：  MOV     P5M1, #00101000B
        MOV     P5M0, #00110000B
```

;P5.5 为开漏，P5.4 为强推挽输出，P5.3 为高阻输入，P5.2/P5.1/P5.0 为准双向口/弱上拉。

P6 口设定<P6.7，P6.6，P6.5，P6.4，P6.3，P6.2，P6.1，P6.0>（P6 口地址：E8H）

P6M1[7：0] 寄存器 P6M1 地址为 CBH	P6M0[7：0] 寄存器 P6M0 地址为 CCH	I/O 口模式
0	0	准双向口（传统 8051 I/O 口模式，弱上拉）， 灌电流可达 20mA，拉电流为 270μA， 由于制造误差，实际为 270uA～150uA
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（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。

```
举例：  MOV     P6M1, #10100000B
        MOV     P6M0, #11000000B
```

;P6.7 为开漏，P6.6 为强推挽输出，P6.5 为高阻输入，P6.4/P6.3/P6.2/P6.1/P6.0 为准双向口/弱上拉。

P7 口设定<P7.7，P7.6，P7.5，P7.4，P7.3，P7.2，P7.1，P7.0>（P7 口地址：F8H）

P7M1[7：0] 寄存器 P7M1 地址为 E1H	P7M0[7：0] 寄存器 P7M0 地址为 E2H	I/O 口模式
0	0	准双向口（传统 8051 I/O 口模式，弱上拉）， 灌电流可达 20mA，拉电流为 270μA， 由于制造误差，实际为 270uA～150uA
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（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。

```
举例：  MOV     P7M1, #10100000B
        MOV     P7M0, #11000000B
```

;P7.7 为开漏，P7.6 为强推挽输出，P7.5 为高阻输入，P7.4/P7.3/P7.2/P7.1/P7.0 为准双向口/弱上拉。

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 MCU-VCC 流入的电流建议不要超过 90mA，从 MCU-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**。

对于现供货的 STC15W4K32S4 系列 A 版本单片机的 I/O 口，请注意：

1、P1.0 和 P1.4 被误设为强推挽输出，可上电复位后可用软件将其改设为弱上拉/准双向口或需要的模式，另外硬件对外时最好串 100Ω 电阻

2、与 PWM2 到 PWM7 相关的 12 个口[P3.7/PWM2，P2.1/PWM3，P2.2/PWM4，P2.3/PWM5，P1.6/PWM6，P1.7/PWM7， P2.7/PWM2_2， P4.5/PWM3_2， P4.4/PWM4_2， P4.2/PWM5_2， P0.7/PWM6_2，P0.6/PWM7_2]，上电复位后是高阻输入，要对外能输出，要软件将其改设为强推挽输出或准双向口/弱上拉

3、如串口 2 切换到[P4.7/TxD，P4.6/RxD]时，P4.7 要加 3.3K 上拉电阻，且需工作在弱上拉/准双向口模式

15.2 管脚 P1.7/XTAL1 与 P1.6/XTAL2 的特别说明

STC15 系列单片机的所有 I/O 口上电复位后均为准双向口/弱上拉模式。但是由于 P1.7 和 P1.6 口还可以分别作外部晶体或时钟电路的引脚 XTAL1 和 XTAL2，所以 P1.7/XTAL1 和 P1.6/XTAL2 上电复位后的模式不一定就是准双向口/弱上拉模式。当 P1.7 和 P1.6 口作为外部晶体或时钟电路的引脚 XTAL1 和 XTAL2 使用时，P1.7/XTAL1 和 P1.6/XTAL2 上电复位后的模式是高阻输入。

每次上电复位时，单片机对 P1.7/XTAL1 和 P1.6/XTAL2 的工作模式按如下步骤进行设置：

1.首先，单片机短时间(几十个时钟)内会将 P1.7/XTAL1 和 P1.6/XTAL2 设置成高阻输入；

2.然后，单片机会自动判断上一次用户 ISP 烧录程序时是将 P1.7/XTAL1 和 P1.6/XTAL2 设置成普通 I/O 口还是 XTAL1/XTAL2；

3.如果上一次用户 ISP 烧录程序时是将 P1.7/XTAL1 和 P1.6/XTAL2 设置成普通 I/O 口，则单片机会将 P1.7/XTAL1 和 P1.6/XTAL2 上电复位后的模式设置成准双向口/弱上拉；

4.如果上一次用户 ISP 编程时是将 P1.7/XTAL1 和 P1.6/XTAL2 设置成 XTAL1/XTAL2，则单片机会将 P1.7/XTAL1 和 P1.6/XTAL2 上电复位后的模式设置成高阻输入。

15.3 复位管脚 RST 的特别说明

STC15 系列 8-pin 单片机(如 STC15F104W 系列)的复位管脚在 RST/P3.4 口，STC15 系列 16-pin 及其以上的单片机(如 STC15W4K32S4 系列、STC15F2K60S2 系列等)的复位管脚在 RST/P5.4 口。下面以 RST/P5.4 为例，介绍复位管脚 RST。

P5.4/RST(或 P3.4/RST)既可作普通 I/O 使用，还可作复位管脚，用户可以在 ISP 烧录程序时设置 P5.4/RST 的功能。当用户 ISP 烧录程序时将 P5.4/RST 设置成普通 I/O 口用时，其上电后为准双向口/弱上拉模式。

每次上电时，单片机会自动判断上一次用户 ISP 烧录程序时，是将 P5.4/RST 设置成普通 I/O 口还是复位脚。

- 如果上一次用户在 ISP 编程时是将 P5.4/RST 设置成普通 I/O 口，则单片机会将 P5.4/RST 上电后的模式设置成准双向口/弱上拉。
- 如果上一次用户 ISP 烧录程序时是将 P5.4/RST 设置成复位脚，则上电后，P5.4/RST 仍为复位脚。

15.4 管脚 RSTOUT_LOW 的特别说明

STC15 系列有的单片机（如 STC15W4K32S4 及 STC15F2K60S2 系列）的 RSTOUT_LOW 脚在 P2.0 口（P2.0/RSTOUT_LOW），有的单片机（如 STC15W104SW 系列）的 RSTOUT_LOW 脚在 P1.0 口（P1.0/RSTOUT_LOW），有的单片机（如 STC15F104W 系列）的 RSTOUT_LOW 脚在 P3.3 口（P3.3/RSTOUT_LOW）。下面以 P2.0/RSTOUT_LOW 管脚为例，介绍管脚 P2.0/RSTOUT_LOW 的一些特别注意事项。

P2.0/RSTOUT_LOW 管脚在单片机上电复位后输出可以为低电平，也可以为高电平。当单片机的工作电压 Vcc 高于上电复位门槛电压（POR）时，用户可以在 ISP 烧录程序时，设置该管脚上电复位后输出的是低电平还是高电平。

当单片机的工作电压 Vcc 低于上电复位门槛电压（POR，3V 单片机在 1.8V 附近，5V 单片机在 3.2V 附近）时，P2.0/RSTOUT_LOW 管脚输出低电平。当单片机的工作电压 Vcc 高于上电复位门槛电压（POR，3V 单片机在 1.8V 附近，5V 单片机在 3.2V 附近）时，单片机首先读取用户在 ISP 烧录程序时的设置，如用户将 P2.0/RSTOUT_LOW 管脚设置为上电复位后输出高电平，则 P2.0/RSTOUT_LOW 管脚上电复位后输出高电平；如用户将 P2.0/RSTOUT_LOW 管脚设置为上电复位后输出低电平，则 P2.0/RSTOUT_LOW 管脚上电复位后输出低电平。

15.5 串行口 1 的中继广播方式

串行口 1 可在 3 组管脚间进行切换：[RxD/P3.0, TxD/P3.1];
[RxD_2/P3.6, TxD_2/P3.7];
[RxD_3/P1.6, TxD_3/P1.7].

Mnemonic	Address	Name	7	6	5	4	3	2	1	0	Reset Value
CLK_DIV(PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000,x000

Tx_Rx：串口 1 的中继广播方式设置

0：串口 1 为正常工作方式

1：串口 1 为中继广播方式，即将 RxD 端口输入的电平状态实时输出在 TxD 外部管脚上，TxD 外部管脚可以对 RxD 管脚的输入信号进行实时整形放大输出，TxD 管脚的对外输出实时反映 RxD 端口输入的电平状态。

串口 1 的 RxD 管脚和 TxD 管脚可以在 3 组不同管脚之间进行切换：[RxD/P3.0, TxD/P3.1];
[RxD_2/P3.6, TxD_2/P3.7];
[RxD_3/P1.6, TxD_3/P1.7].

串行口 1 的中继广播方式除可以在用户程序中设置 Tx_Rx/CLK_DIV.4 来选择外，还可以在 AIapp-ISP 下载编程软件中设置。

当单片机的工作电压低于上电复位门槛电压（POR，3V 单片机在 1.8V 附近，5V 单片机在 3.2V 附近）时，Tx_Rx 默认为 0，即串行口 1 默认为正常工作方式。

当单片机的工作电压高于上电复位门槛电压（POR，3V 单片机在 1.8V 附近，5V 单片机在 3.2V 附近）时，单片机首先读取用户在 AIapp-ISP 下载编程软件中的设置。

- 如果用户允许了“单片机 TxD 管脚的对外输出实时反映 RxD 端口输入的电平状态”即中继广播方式，则上电复位后 P3.7/TxD_2 管脚的对外输出可以实时反映 P3.6/RxD_2 端口输入的电平状态；
- 如果用户未选择“单片机 TxD 管脚的对外输出实时反映 RxD 端口输入的电平状态”，则上电复位后串口 1 为正常工作方式，即 P3.7/TxD_2 管脚的对外输出不实时反映 P3.6/RxD_2 端口输入的电平状态。

串行口 1 的位置和中继广播方式除可以在 AIapp-ISP 下载编程软件中设置外，还可以在用户的用户程序中用设置。在 AIapp-ISP 下载编程软件中的设置是在单片机上电复位后就可以执行的，如果用户在用户程序中的设置与 AIapp-ISP 下载编程软件中的设置不一致时，当执行到相应的用户程序时就会覆盖原来 AIapp-ISP 下载编程软件中的设置。

15.6 可将 MCU 从掉电模式/停机模式唤醒的外部管脚资源

可将 MCU 从掉电模式/停机模式唤醒的外部管脚资源有：

- INT0/P3.2，INT1/P3.3（INT0/INT1 上升沿下降沿中断均可），INT2/P3.6，INT3/P3.7，INT4/P3.0（INT2/INT3/INT4 仅可下降沿中断）；
- 管脚 CCP0/CCP1/CCP2；
- 管脚 RxD/RxD2/RxD3/RxD4（下降沿，不产生中断）；
- 管脚 T0/T1/T2/T3/T4（下降沿即外部管脚由高到低的变化，前提是在进入掉电模式/停机模式前相应的定时器中断已经被允许）；
- 内部低功耗掉电唤醒专用定时器。

INT0/P3.2 和 INT1/P3.3 的上升沿/下降沿中断均可唤醒掉电模式/停机模式。而 INT2/P3.6，INT3/P3.7，INT4/P3.0 仅下降沿中断才可将 MCU 从掉电模式/停机模式唤醒。

管脚 CCP 可以在[CCP0/P1.1，CCP1/P1.0，CCP2/CCP2_2/P3.7]，[CCP0_2/P3.5，CCP1_2/P3.6，CCP2/CCP2_2/P3.7]，[CCP0_3/P2.5，CCP1_3/P2.6，CCP2_3/P2.7]之间进行切换。管脚 CCP 同样可以将 MCU 从掉电模式/停机模式中唤醒。

如果掉电模式/停机模式是由外部中断 INT0（上升沿+下降沿中断）、INT1（上升沿+下降沿中断）、INT2（仅可下降沿中断）、INT3（仅可下降沿中断）、INT4（仅可下降沿中断）或管脚 CCP 唤醒，则掉电唤醒之后，CPU 首先执行设置单片机进入掉电模式的语句的下一条语句（建议在设置单片机进入掉电模式的语句后多加几个 NOP 空指令），然后执行相应的中断服务程序。

如果在进入掉电模式/停机模式前串行中断被允许，则进入掉电模式/停机模式后，串行口的接收管脚由高到低的变化可以唤醒掉电模式/停机模式。

- 串行口 1 的接收管脚 RxD 可以从 RxD/P3.0 切换到 RxD_2/P3.6，还可以切换到 RxD_3/P2.6 之间切换，上电复位后 RxD 默认在 RxD_2/P3.6 管脚上。
- 串行口 2 的接收管脚 RxD2 可以从 RxD2/P1.0 切换到 RxD2_2/P4.6，上电复位后 RxD2 默认在 RxD2/P1.0 管脚上。
- 串行口 3 的接收管脚 RxD3 可以从 RxD3/P0.0 切换到 RxD3_2/P5.0，上电复位后 RxD3 默认在 RxD3/P0.0 管脚上。
- 串行口 4 的接收管脚 RxD4 可以从 RxD4/P0.2 切换到 RxD4_2/P5.2，上电复位后 RxD4 默认在 RxD4/P0.2 管脚上。

如果在进入掉电模式/停机模式前定时器中断被允许，即进入掉电模式/停机模式前 ET0/ET1/ET2/ET3/ET4 及 EA 已经被设置为 1，则进入掉电模式/停机模式后，定时器的外部管脚（T0/P3.4，T1/P3.5，T2/P3.1，T3/P0.5，T4/P0.7）由高到低的变化也可以将 MCU 从掉电模式/停机模式唤醒。

当 MCU 由 RxD 或 RxD2 或 RxD3 或 RxD4 管脚的下降沿（由高到低的变化）唤醒或由定时器 T0/T1/T2/T3/T4 的外部管脚的下降沿（由高到低的变化）唤醒时，如果主时钟使用的是内部系统时钟（由用户在 ISP 烧录程序时自行设置），MCU 在等待 64 个时钟后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态，就将时钟供给 CPU 工作；如果主时钟使用的是外部晶体或时钟（由用户在 ISP 烧录程序时自行设置），MCU 在等待 1024 个时钟后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态，就将时钟供给 CPU 工作；CPU 获得时钟后，程序从上次设置单片机进入掉电模式语句的下一条语句开始往下执行。

15.7 与 I/O 口有关的特殊功能寄存器及其在程序中的地址声明

下面将与 I/O 口相关的寄存器及其地址列于此处，以方便用户查询

P0 register(可位寻址)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P0	80H	name	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0

P0M1 register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P0M1	93H	name	P0M1.7	P0M1.6	P0M1.5	P0M1.4	P0M1.3	P0M1.2	P0M1.1	P0M1.0

P0M0 register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P0M0	94H	name	P0M0.7	P0M0.6	P0M0.5	P0M0.4	P0M0.3	P0M0.2	P0M0.1	P0M0.0

P1 register (可位寻址)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P1	90H	name	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0

P1M1 register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P1M1	91H	name	P1M1.7	P1M1.6	P1M1.5	P1M1.4	P1M1.3	P1M1.2	P1M1.1	P1M1.0

P1M0 register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P1M0	92H	name	P1M0.7	P1M0.6	P1M0.5	P1M0.4	P1M0.3	P1M0.2	P1M0.1	P1M0.0

P2 register (可位寻址)

SFR name	Address	bit	B1	B6	B5	B4	B3	B2	B1	B0
P2	A0H	name	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0

P2M1 register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P2M1	95H	name	P2M1.7	P2M1.6	P2M1.5	P2M1.4	P2M1.3	P2M1.2	P2M1.1	P2M1.0

P2M0 register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P2M0	96H	name	P2M0.7	P2M0.6	P2M0.5	P2M0.4	P2M0.3	P2M0.2	P2M0.1	P2M0.0

P3 register (可位寻址)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P3	B0H	name	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0

P3M1 register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P3M1	B1H	name	P3M1.7	P3M1.6	P3M1.5	P3M1.4	P3M1.3	P3M1.2	P3M1.1	P3M1.0

P3M0 register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P3M0	B2H	name	P3M0.7	P3M0.6	P3M0.5	P3M0.4	P3M0.3	P3M0.2	P3M0.1	P3M0.0

P4 register (可位寻址)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P4	C0H	name	P4.7	P4.6	P4.5	P4.4	P4.3	P4.2	P4.1	P4.0

P4M1 register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P4M1	B3H	name	P4M1.7	P4M1.6	P4M1.5	P4M1.4	P4M1.3	P4M1.2	P4M1.1	P4M1.0

P4M0 register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P4M0	B4H	name	P4M0.7	P4M0.6	P4M0.5	P4M0.4	P4M0.3	P4M0.2	P4M0.1	P4M0.0

P5 register (可位寻址)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P5	C8H	name	-	-	P5.5	P5.4	P5.3	P5.2	P5.1	P5.0

P5M1 register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P5M1	C9H	name	-	-	P5M1.5	P5M1.4	P5M1.3	P5M1.2	P5M1.1	P5M1.0

P5M0 register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P5M0	CAH	name	-	-	P5M0.5	P5M0.4	P5M0.3	P5M0.2	P5M0.1	P5M0.0

P6 register (可位寻址)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P6	E8H	name	P6.7	P6.6	P6.5	P6.4	P6.3	P6.2	P6.1	P6.0

P6M1 register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P6M1	CBH	name	P6M1.7	P6M1.6	P6M1.5	P6M1.4	P6M1.3	P6M1.2	P6M1.1	P6M1.0

P6M0 register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P6M0	CCH	name	P6M0.7	P6M0.6	P6M0.5	P6M0.4	P6M0.3	P6M0.2	P6M0.1	P6M0.0

P7 register (可位寻址**)**

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P7	F8H	name	P7.7	P7.6	P7.5	P7.4	P7.3	P7.2	P7.1	P7.0

P7M1 register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P7M1	E1H	name	P7M1.7	P7M1.6	P7M1.5	P7M1.4	P7M1.3	P7M1.2	P7M1.1	P7M1.0

P7M0 register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P7M0	E2H	name	P7M0.7	P7M0.6	P7M0.5	P7M0.4	P7M0.3	P7M0.2	P7M0.1	P7M0.0

汇编语言：

```
    P7      EQU     0F8H            ; or P7     DATA    0F8H
    P7M1    EQU     0E1H            ; or P7M1   DATA    0E1H
    P7M0    EQU     0E2H
;以上为 P7 口新增功能寄存器的地址声明

    P6      EQU     0E8H            ; or P6     DATA    0E8H
    P6M1    EQU     0CBH            ; or P6M1   DATA    0CBH
    P6M0    EQU     0CCH
;以上为 P6 口新增功能寄存器的地址声明

    P5      EQU     0C8H            ; or P5     DATA    0C8H
    P5M1    EQU     0C9H            ; or P5M1   DATA    0C9H
    P5M0    EQU     0CAH
;以上为 P5 口新增功能寄存器的地址声明

    P4      EQU     0C0H            ; or P4     DATA    0C0H
    P4M1    EQU     0B3H            ; or P4M1   DATA    0B3H
    P4M0    EQU     0B4H
;以上为 P4 口新增功能寄存器的地址声明

    P3M1    EQU     0B1H            ; or P3M1   DATA    0B1H
    P3M0    EQU     0B2H
;以上为 P3 口新增功能寄存器的地址声明

    P2M1    EQU     095H
    P2M0    EQU     096H
;以上为 P2 口新增功能寄存器的地址声明

    P1M1    EQU     091H
    P1M0    EQU     092H
;以上为 P1 口新增功能寄存器的地址声明
```

```
P0M1    EQU     093H
P0M0    EQU     094H
```

;以上为 **P0** 口新增功能寄存器的地址声明

C 语言：

```
sfr     P7 = 0xf8;
sfr     P7M1 = 0xe1;
sfr     P7M0 = 0xe2;
```

/*以上为 **P7** 新增功能寄存器的 C 语言地址声明*/

```
sfr     P6 = 0xe8;
sfr     P6M1 = 0xcb;
sfr     P6M0 = 0xcc;
```

/*以上为 **P6** 新增功能寄存器的 C 语言地址声明*/

```
sfr     P5 = 0xc8;
sfr     P5M1 = 0xc9;
sfr     P5M0 = 0xca;
```

/*以上为 **P5** 新增功能寄存器的 C 语言地址声明*/

```
sfr     P4 = 0xc0;
sfr     P4M1 = 0xb3;
sfr     P4M0 = 0xb4;
```

/*以上为 **P4** 新增功能寄存器的 C 语言地址声明*/

```
sfr     P3M1 = 0xb1;
sfr     P3M0 = 0xb2;
```

/*以上为 **P3** 新增功能寄存器的 C 语言地址声明*/

```
sfr     P2M1 = 0x95;
sfr     P2M0 = 0x96;
```

/*以上为 **P2** 新增功能寄存器的 C 语言地址声明*/

```
sfr     P1M1 = 0x91;
sfr     P1M0 = 0x92;
```

/*以上为 **P1** 新增功能寄存器的 C 语言地址声明*/

```
sfr     P0M1 = 0x93;
sfr     P0M0 = 0x94;
```

/*以上为 **P0** 新增功能寄存器的 C 语言地址声明*/

15.8 STC15 系列单片机 P0/P1/P2/P3/P4/P5 口的测试程序

```
/*.-------------------------------------------------------------------------------------------*/
/*---STC15W4K60S4 系列 P0/P1/P2/P3/P4/P5 举例-----------------------------------*/
/*---在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可----*/
/*.-------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"
#include "intrins.h"
sfr     P5 = 0xC8;      //6 bit Port5   P5.7 P5.6 P5.5 P5.4 P5.3 P5.2 P5.1 P5.0     xx11,1111
sfr     P5M0 = 0xCA;    //                                                          0000,0000
sfr     P5M1 = 0xC9;    //                                                          0000,0000
                        //              7   6   5   4   3   2   1   0               Reset Value

sfr     P4 = 0xC0;      //8 bitPort4    P4.7 P4.6 P4.5 P4.4 P4.3 P4.2 P4.1 P4.0     1111,1111
sfr     P4M0 = 0xB4;    //                                                          0000,0000
sfr     P4M1 = 0xB3;    //                                                          0000,0000

sbit    P10 = P1^0;
sbit    P11 = P1^1;
sbit    P12 = P1^2;
sbit    P13 = P1^3;
sbit    P14 = P1^4;
sbit    P15 = P1^5;
sbit    P16 = P1^6;
sbit    P17 = P1^7;

sbit    P30 = P3^0;
sbit    P31 = P3^1;
sbit    P32 = P3^2;
sbit    P33 = P3^3;
sbit    P34 = P3^4;
sbit    P35 = P3^5;
sbit    P36 = P3^6;
sbit    P37 = P3^7;

sbit    P20 = P2^0;
sbit    P21 = P2^1;
sbit    P22 = P2^2;
sbit    P23 = P2^3;
sbit    P24 = P2^4;
sbit    P25 = P2^5;
sbit    P26 = P2^6;
sbit    P27 = P2^7;
```

```
sbit	P00 = P0^0;
sbit	P01 = P0^1;
sbit	P02 = P0^2;
sbit	P03 = P0^3;
sbit	P04 = P0^4;
sbit	P05 = P0^5;
sbit	P06 = P0^6;
sbit	P07 = P0^7;

sbit	P40 = P4^0;
sbit	P41 = P4^1;
sbit	P42 = P4^2;
sbit	P43 = P4^3;
sbit	P44 = P4^4;
sbit	P45 = P4^5;
sbit	P46 = P4^6;
sbit	P47 = P4^7;

sbit	P50 = P5^0;
sbit	P51 = P5^1;
sbit	P52 = P5^2;
sbit	P53 = P5^3;
sbit	P54 = P5^4;
sbit	P55 = P5^5;

void delay(void);
void main(void)
{
	P10 = 0;
	delay();
	P11 = 0;
	delay();
	P12 = 0;
	delay();
	P13 = 0;
	delay();
	P14 = 0;
	delay();
	P15 = 0;
	delay();
	P16 = 0;
	delay();
	P17 = 0;
	delay();

	P1 = 0xff;
```

```
P30 = 0;
delay();
P31 = 0;
delay();
P32 = 0;
delay();
P33 = 0;
delay();
P34 = 0;
delay();
P35 = 0;
delay();
P36 = 0;
delay();
P37 = 0;
delay();

P3 = 0xff;

P20 = 0;
delay();
P21 = 0;
delay();
P22 = 0;
delay();
P23 = 0;
delay();
P24 = 0;
delay();
P25 = 0;
delay();
P26 = 0;
delay();
P27 = 0;
delay();

P2 = 0xff;

P07 = 0;
delay();
P06 = 0;
delay();
P05 = 0;
delay();
P04 = 0;
```

```
    delay();
    P03 = 0;
    delay();
    P02 = 0;
    delay();
    P01 = 0;
    delay();
    P00 = 0;
    delay();

    P0 = 0xff;

    P40 = 0;
    delay();
    P41 = 0;
    delay();
    P42 = 0;
    delay();
    P43 = 0;
    delay();
    P44 = 0;
    delay();
    P45 = 0;
    delay();
    P46 = 0;
    delay();
    P47 = 0;
    delay();

    P4 = 0xff;

    P50 = 0;
    delay();
    P51 = 0;
    delay();
    P52 = 0;
    delay();
    P53 = 0;
    delay();
    P54 = 0;
    delay();
    P55 = 0;
    delay();

    P5 = 0xff;
    while(1)
```

```
    {
        P1 = 0x00;
        delay();
        P1 = 0xff;

        P3 = 0x00;
        delay();
        P3 = 0xff;

        P2 = 0x00;
        delay();
        P2 = 0xff;

        P0 = 0x00;
        delay();
        P0 = 0xff;

        P4 = 0x00;
        delay();
        P4 = 0xff;

        P5 = 0x00;
        delay();
        P5 = 0xff;
    }
}

void delay(void)
{
    unsigned int i = 0;
    for(i=60000; i>0; i--)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
```

```
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();

            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
        }
}
```

15.9 I/O 口各种不同的工作模式结构框图

15.9.1 准双向口（弱上拉）输出配置

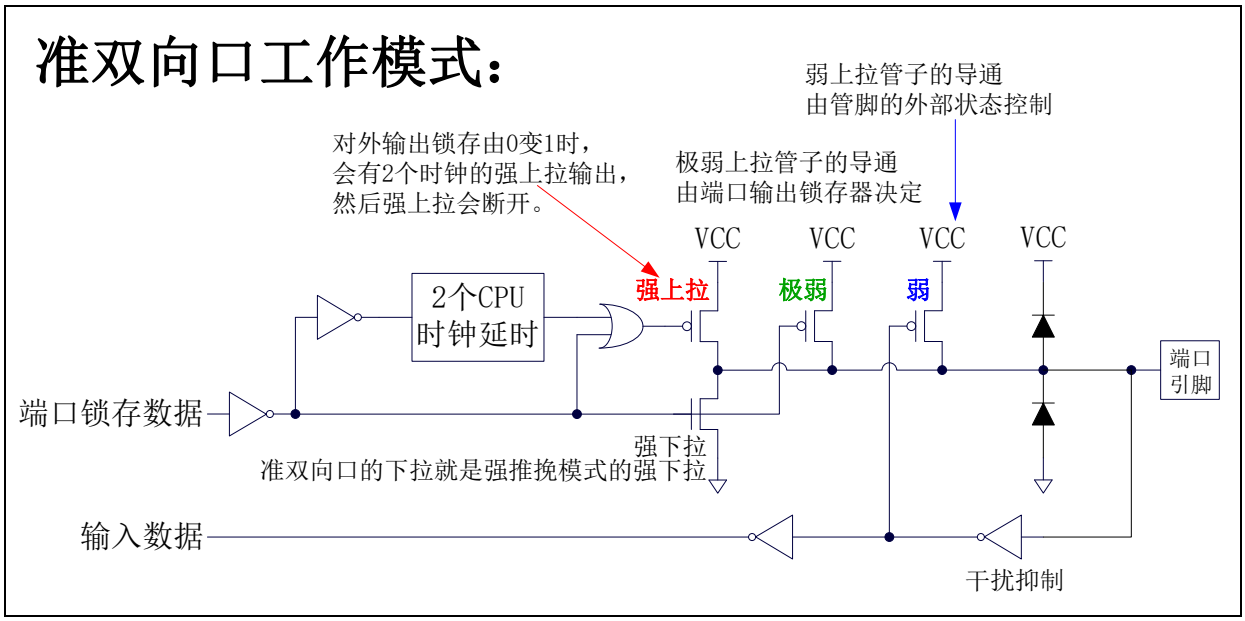
准双向口（弱上拉）输出类型可用作输出和输入功能而不需重新配置端口输出状态。这是因为当端口输出为 1 时驱动能力很弱，允许外部装置将其拉低。当引脚输出为低时，它的驱动能力很强，可吸收相当大的电流。准双向口有 3 个上拉晶体管适应不同的需要。

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）输出如下图所示。



STC 1T 系列单片机为 3.3V 器件，如果用户在引脚加上 5V 电压，将会有电流从引脚流向 Vcc，这样导致额外的功率消耗。因此，建议不要在准双向口（弱上拉）模式中向 3.3V 单片机引脚施加 5V 电压，如使用的话，要加限流电阻，或用二极管做输入隔离，或用三极管做输出隔离。

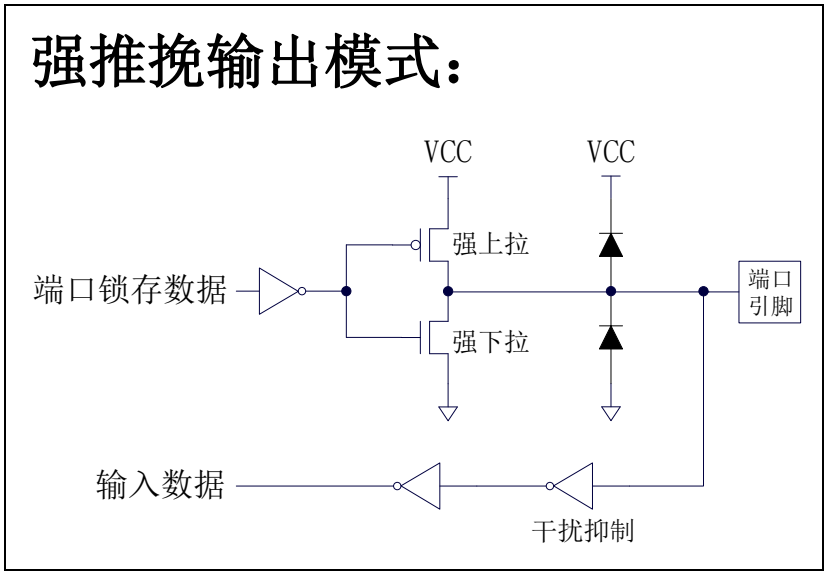
准双向口（弱上拉）带有一个施密特触发输入以及一个干扰抑制电路。

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。

15.9.2 强推挽输出配置

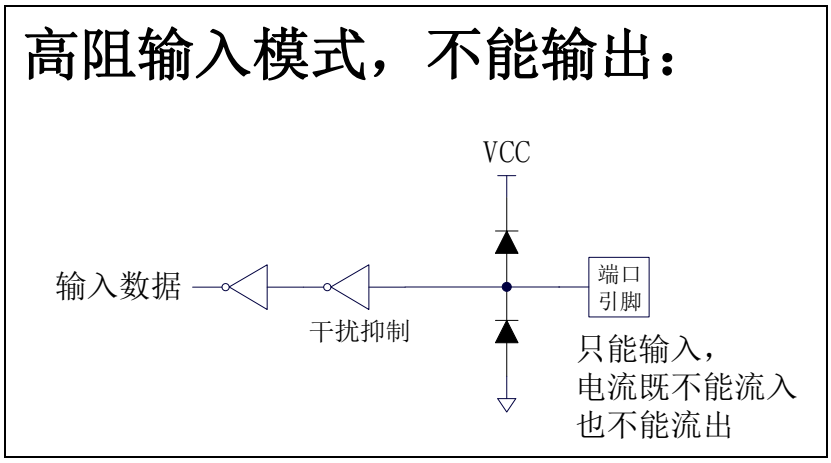
强推挽输出配置的下拉结构与开漏输出以及准双向口的下拉结构相同，但当锁存器为 1 时提供持续的强上拉。推挽模式一般用于需要更大驱动电流的情况。

强推挽引脚配置如下图所示。



15.9.3 高阻输入（电流既不能流入也不能流出）配置

高阻输入口配置如下图所示。



输入口带有一个施密特触发输入以及一个干扰抑制电路。

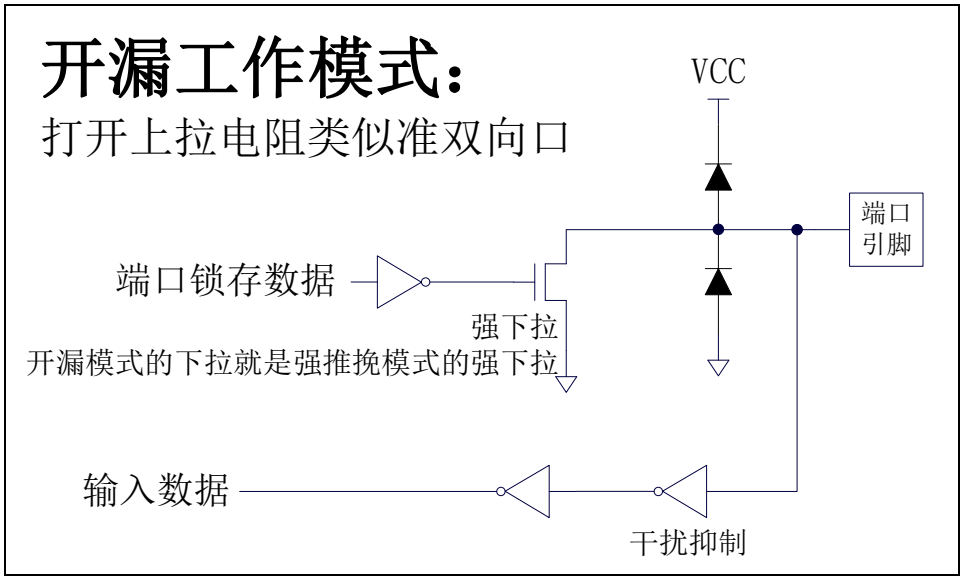
15.9.4 开漏输出配置（若外加上拉电阻，也可读外部状态或输出高电平）

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，开漏输出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。输出端口配置如下图所示。

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。

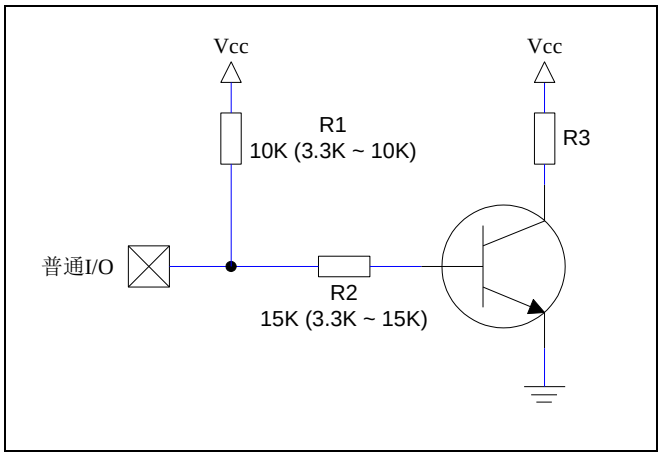


关于 I/O 口应用注意事项：

少数用户反映 I/O 口有损坏现象，后发现是

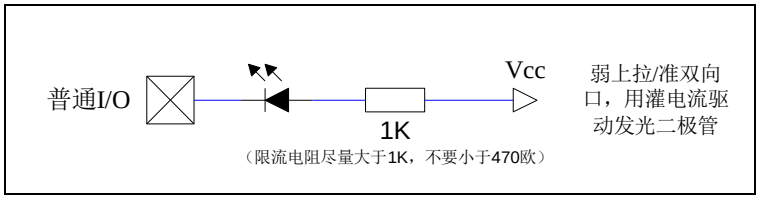
- 有些是 I/O 口由低变高读外部状态时，读不对，实际没有损坏，软件处理一下即可。
  因为 1T 的 8051 单片机速度太快了，软件执行由低变高指令后立即读外部状态，此时由于实际输出还没有变高，就有可能读不对，正确的方法是在软件设置由低变高后加 1 到 2 个空操作指令延时，再读就对了。
- 有些实际没有损坏，加上拉电阻就 OK 了
- 有些是外围接的是 NPN 三极管，没有加上拉电阻，其实基极串多大电阻，I/O 口就应该上拉多大的电阻，或者将该 I/O 口设置为强推挽输出.
- 有些确实是损坏了，原因：
  - ✓ 发现有些是驱动 LED 发光二极管没有加限流电阻，建议加 1K 以上的限流电阻，至少也要加 470 欧姆以上
  - ✓ 发现有些是做行列矩阵按键扫描电路时，实际工作时没有加限流电阻，实际工作时可能出现 2 个 I/O 口均输出为低，并且在按键按下时，短接在一起。我们知道一个 CMOS 电路的 2 个输出脚不应该直接短接在一起，按键扫描电路中，此时一个口为了读另外一个口的状态，必须先置高才能读另外一个口的状态。而 8051 单片机的弱上拉口在由 0 变为 1 时，会有 2 个时钟的强推挽高输出电流，输出到另外一个输出为低的 I/O 口，就有可能造成 I/O 口损坏。建议在两侧各加 300 欧姆限流电阻，或者在软件处理上，不要出现按键两端的 I/O 口同时为低。

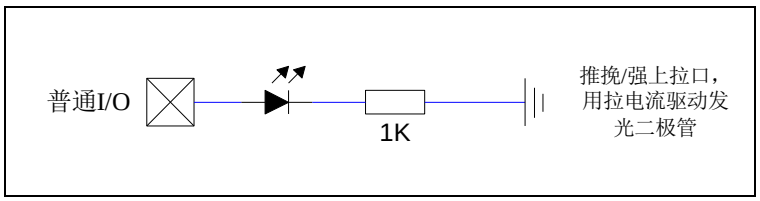
15.10 一种典型三极管控制电路



如果用弱上拉控制，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。

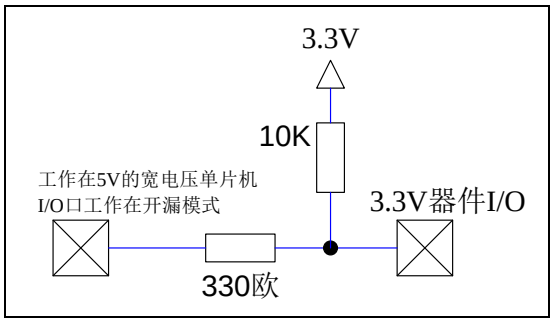
15.11 典型发光二极管控制电路



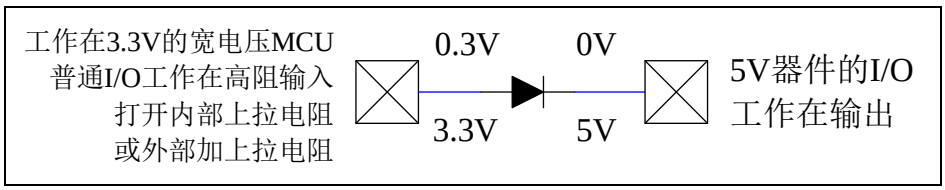


15.12 混合电压供电系统 3V/5V 器件 I/O 口互连

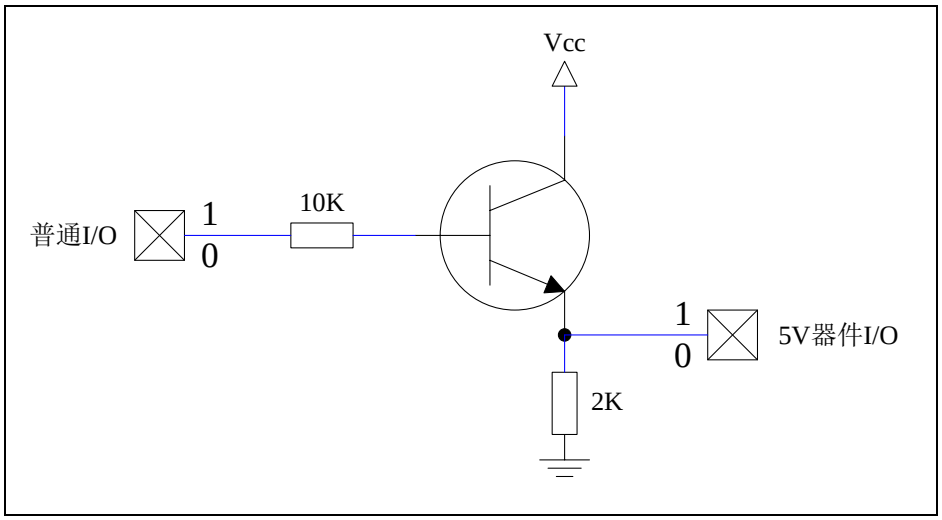
STC 1T 系列 5V 单片机连接 3.3V 器件时，为防止 3.3V 器件承受不了 5V，可将相应的 5V 单片机 I/O 口先串一个 330Ω 的限流电阻到 3.3V 器件 I/O 口，程序初始化时将 5V 单片机的 I/O 口设置成开漏配置，断开内部上拉电阻，相应的 3.3V 器件 I/O 口外部加 10K 上拉电阻到 3.3V 器件的 Vcc，这样高电平是 3.3V，低电平是 0V，输入输出一切正常。



STC 1T 系列 3V 单片机连接 5V 器件时，为防止 3V 单片机承受不了 5V，如果相应的 I/O 口是输入，可在该 I/O 口上串接一个隔离二极管，隔离高压部分。外部信号电压高于单片机工作电压时截止，I/O 口因内部上拉到高电平，所以读 I/O 口状态是高电平;外部信号电压为低时导通，I/O 口被钳位在 0.7V，小于 0.8V 时单片机读 I/O 口状态是低电平。



STC 1T 系列 3V 单片机连接 5V 器件时，为防止 3V 单片机承受不了 5V，如果相应的 I/O 口是输出，可用一个 NPN 三极管隔离，电路如下：



15.13 I/O 口的外部输入何时低（0.8V 以下）何时高电平（2.2V 以上）

当 I/O 口的外部输入电平在 0.8V 以下时，则单片机认为该 I/O 口的外部输入为低电平；

当 I/O 口的外部输入电平在 2.2V 以上时，则单片机认为该 I/O 口的外部输入为高电平。

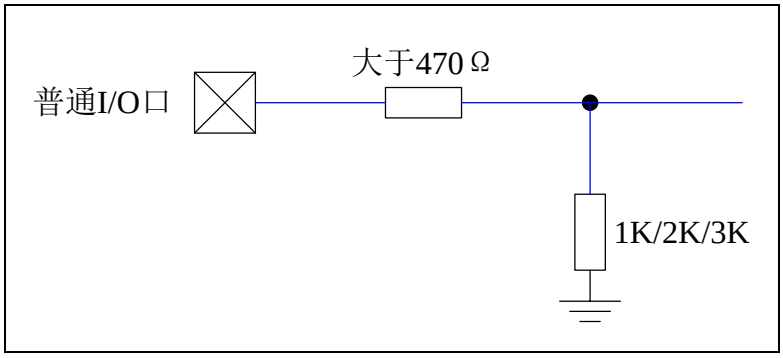
实际制造时按 I/O 口的外部输入电平在 1.2V 以下时为低电平，在 1.8V 以上时为高电平。但由于存在制造误差，1.2V 以下单片机不一定认为 I/O 口的外部输入为低电平，1.8V 以上单片机也不一定就认为 I/O 口的外部输入为高电平。但我们保证 0.8V 以下可以为低电平，2.2V 以上可以为高电平，外部输入电平在 0.8V ~ 2.2V 之间不保证单片机能固定地识别 I/O 口的外部输入为低电平还是为高电平。

15.14 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 STC15 系列单片机由于既有弱上拉输出又有强推挽输出，就可以很轻松的解决此问题。

现可在 STC15 系列单片机 I/O 口上加一个下拉电阻（1K/2K/3K），这样上电复位时，虽然单片机内部 I/O 口是弱上拉（准双向口）/高电平输出，但由于内部上拉能力有限，而外部下拉电阻又较小，无法将其拉高，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。



【特别提示】：STC15 系列单片机的 RSTOUT_LOW 管脚可以在 ISP 烧录程序时设置上电复位后输出低电平还是高电平，其他管脚上电复位后均输出高电平。

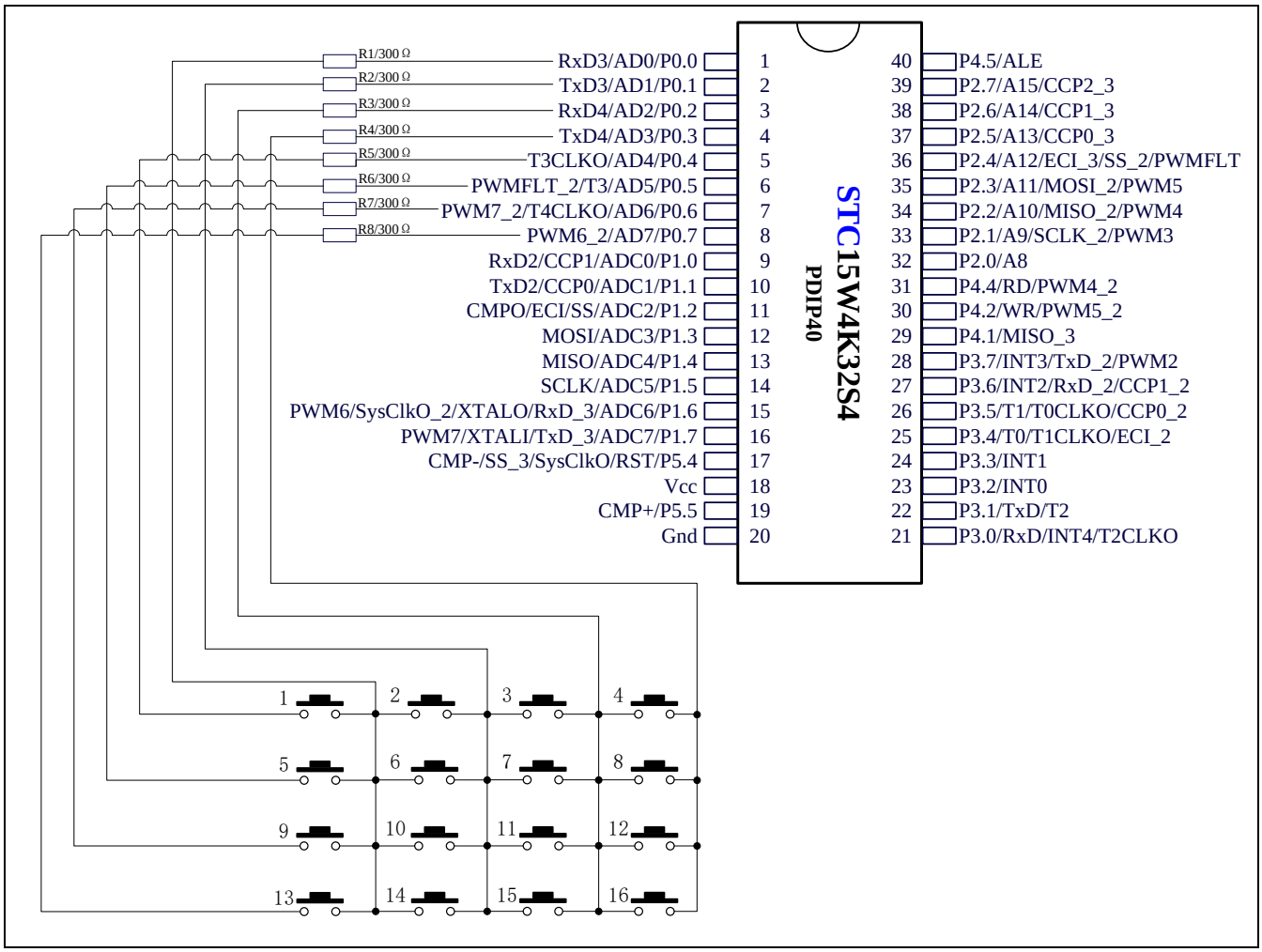
STC15 系列有的单片机（如 STC15W4K60S4 及 STC15F2K60S2 系列）的 RSTOUT_LOW 脚在 P2.0 口（P2.0/RSTOUT_LOW），有的单片机（如 STC15W204S 系列）的 RSTOUT_LOW 脚在 P1.0 口（P1.0/RSTOUT_LOW），有的单片机（如 STC15F104W 系列）的 RSTOUT_LOW 脚在 P3.3 口（P3.3/RSTOUT_LOW）。

15.15 PWM 输出时 I/O 口的状态

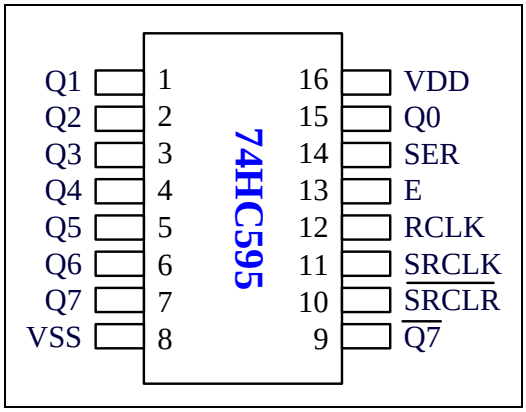
当 I/O 口作为 PWM 输出用时，不改变口的输出状态，要软件设置，建议用户将作 PWM 用的 I/O 口状态设置成强推挽输出，与早期的 STC 1T 系列单片机（如 STC12 系列）不同。下表为 STC12 系列单片机的 I/O 口作 PWM 用时，该口的状态：

PWM 之前口的状态	PWM 时口的状态
弱上拉/准双向口	强推挽输出/强上拉输出，要加输出限流电阻 10K ~ 1K
强推挽输出	强推挽输出/强上拉输出，要加输出限流电阻 10K ~ 1K
仅为输入/高阻	PWM 无效
开漏	开漏

15.16 I/O 口行列式按键扫描应用线路图



15.17 74HC595 管脚介绍及逻辑表



74HC595 管脚介绍		
管脚名称	管脚编号	管脚功能
Q0 ~ Q7	15, 1~7	并行数据输出
Q7	9	串行数据输出
SRCLR	10	清除端(低电平有效)
SRCLK	11	移位寄存器的时钟输入
RCLK	12	三态输出锁存器的时钟输入
E	13	输出允许控制
SER	14	串行数据输入
VDD	16	电源正极
VSS	8	电源接地端

单片机系统中，常采用 74HC595 作为 LED 的静态显示接口。74HC595 的管脚如上图所示。

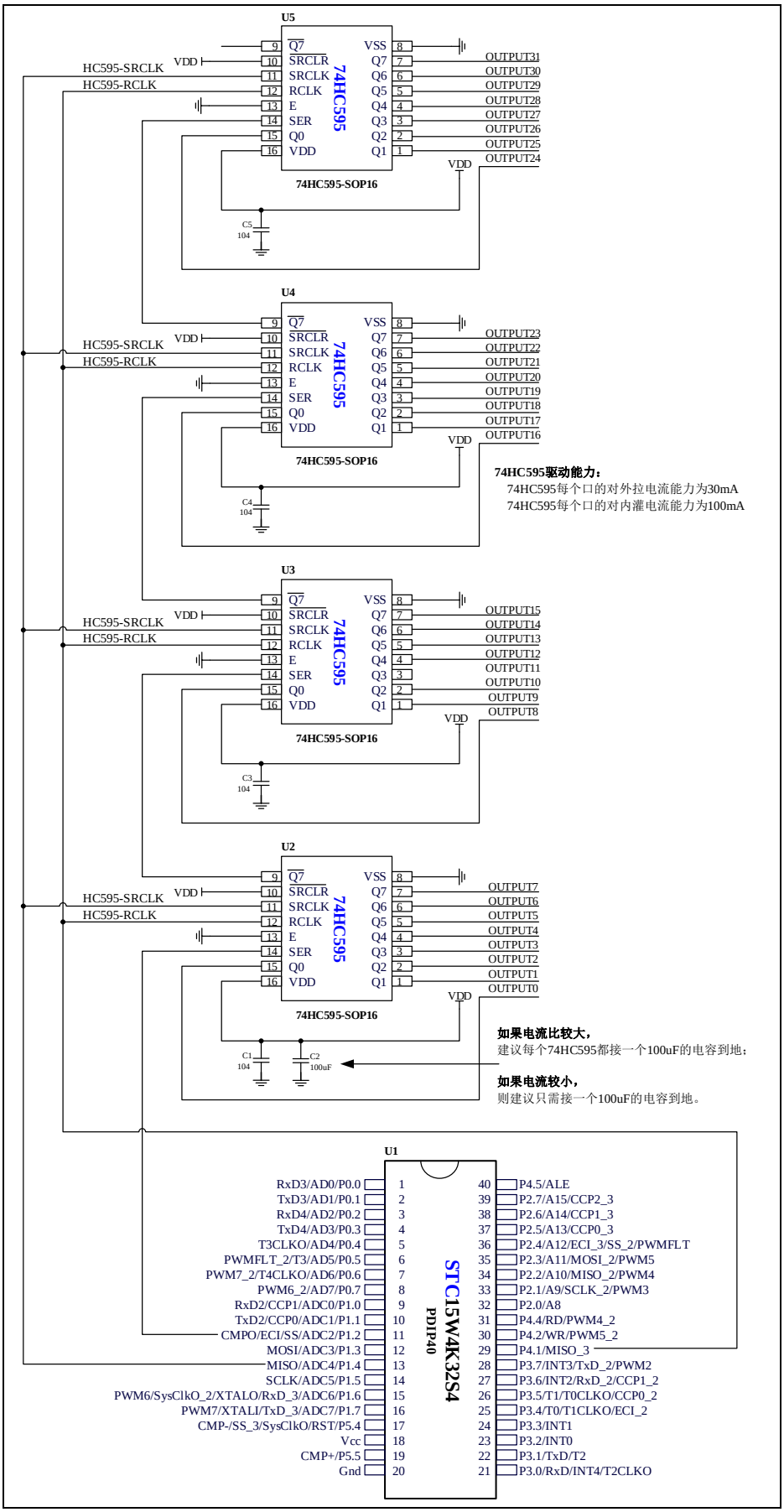
该芯片内含 8 位串入、串/并出移位寄存器和 8 位三态输出锁存器。寄存器和锁存器分别有各自的时钟输入（SRCLK 和 RCLK），都是上升沿有效。当 SRCLK 从低到高电平跳变时，串行输入数据（SER）移入寄存器；当 RCLK 从低到高电平跳变时，寄存器的数据置入锁存器。清除端（SRCLR）的低电平只对寄存器复位（Q7 为低电平），而对锁存器无影响。当输出允许控制（E）为高电平时，并行输出（Q0 ~ Q7）为高阻态，而串行输出（Q7）不受影响。

74HC595 最多需要 5 根控制线，即 SER、SRCLK、RCLK、SRCLR 和 E。其中 SRCLR 可以直接接到高电平，用软件来实现寄存器清零；如果不需要软件改变亮度，E 可以直接接到低电平，而用硬件来改变亮度。把其余三根线和单片机的 I/O 口相接，即可实现对 LED 的控制。

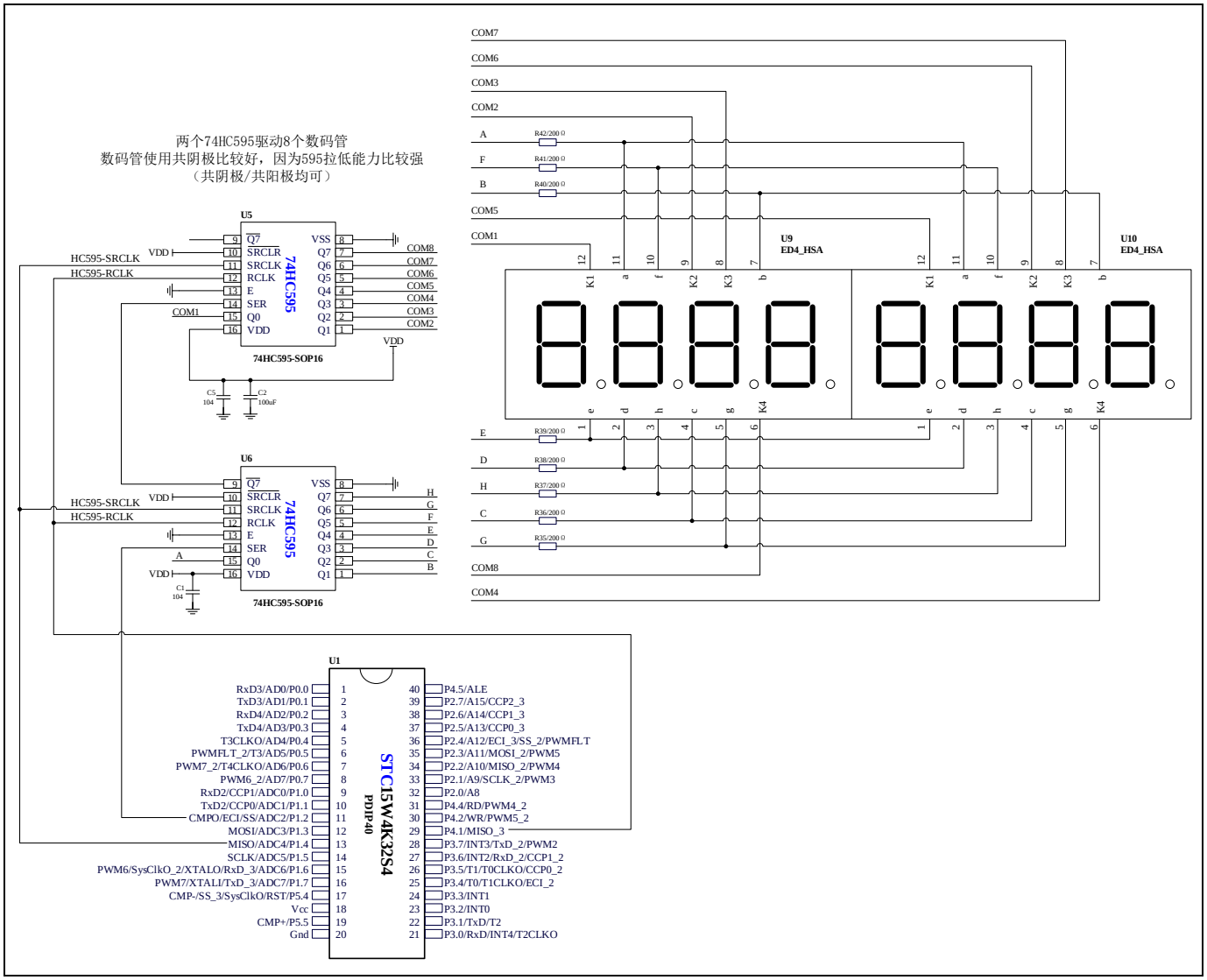
数据从 SDI 口送入 74HC595，在每个 SRCLK 的上升沿，SER 口上的数据移入寄存器，在 SRCLK 的第 9 个上升沿，数据开始从 Q7 移出。如果把第一个 74HC595 的 Q7 和第二个 74HC595 的 SER 相接，数据即移入第二个 74HC595 中，照此一个一个接下去，可任意多个。数据全部传送完后，给 RCLK 一个上升沿，寄存器中的数据即置入锁存器中。此时如果 E 为低电平，数据即从并口 Q0 ~ Q7 输出，把 Q0 ~ Q7 与 LED 的 8 段相接，LED 就可以显示了。想要软件改变 LED 亮度，只需改变 E 的占空比就行了。

74HC595 逻辑表					
输入管脚					输出管脚
SER	SRCLK	SRCLR	RCLK	E	
X	X	X	X	H	Q0 ~ Q7 输出高阻
X	X	X	X	L	Q0 ~ Q7 输出有效值
X	X	L	X	X	移位寄存器清零
L	上升沿	H	X	X	移位寄存器存储 L
H	上升沿	H	X	X	移位寄存器存储 H
X	下降沿	H	X	X	移位寄存器状态保持
X	X	X	上升沿	X	输出存储器锁存移位寄存器中的状态值
X	X	X	下降沿	X	输出存储器状态保持

15.18 利用 74HC595 扩展 I/O 口的线路图(串行扩展，3 根线)



15.19 利用 74HC595 驱动 8 个数码管(串行扩展,3 根线)的线路图



15.20 利用普通 I/O 口控制 74HC595 驱动 8 个数码管的测试程序

1.C 程序

```
/*---------------------------------------------------------------------*/
/*----STC 1T Series MCU Programme Demo----------------------*/
/*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*********************
用 STC 的 MCU 的普通 I/O 方式控制 74HC595 驱动 8 位数码管。
用户可以修改宏来选择时钟频率
用户可以在显示函数里修改成共阴或共阳，推荐尽量使用共阴数码管
显示效果为：8 个数码管循环显示 0,1,2..A,B..F 消隐
*****************************************************/
#include "reg52.h"
```

```
/*************************用户定义宏************************/
#define         MAIN_Fosc       11059200UL      //定义主时钟
//#define       MAIN_Fosc       22118400UL      //定义时钟
/***********************************************************/

/*************下面的宏自动生成，用户可不用修改*************/
#define         Timer0_Reload   (MAIN_Fosc/12000)
/***********************************************************/

/*******************本地常量声明**************************/
unsigned        char        code t_display[] = {
//              0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F   消隐
                0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x77,0x7C,0x39,0x5E,0x79,0x71,0x00
                                        };                                      //段码
unsigned        char        code T_COM[] = {0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};   //位码
/*****************  本地变量声明  ***************/
//sbit          P_HC595_SER = P3^2;             //pin 14 SER data input
//sbit          P_HC595_RCLK = P3^4;            //pin 12 RCLK store (latch) clock
//sbit          P_HC595_SRCLK = P3^3;           //pin 11 SRCLK Shift data clock

sbit            P_HC595_SER = P1^3;             //pin 14 SER data input
sbit            P_HC595_RCLK = P4^1;            //pin 12 RCLK store (latch) clock
sbit            P_HC595_SRCLK = P1^5;           //pin 11 SRCLK Shift data clock

unsigned        char        LED8[8];            //显示缓冲
unsigned        char        display_index;      //显示位索引
bit             B_1ms;                          //1ms 标志
/***********************************************/
void main(void)
{
    unsigned char i, k;
    unsigned int j;

    TMOD = 0x01;                                //Timer 0 config as 16bit timer, 12T
    TH0 = (65536 - Timer0_Reload) / 256;
    TL0 = (65536 - Timer0_Reload) % 256;
    ET0 = 1;
    TR0 = 1;
    EA = 1;

    for(i=0; i<8; i++) LED8[i] = 0x10;          //上电消隐
    j = 0;
    k = 0;
//  for(i=0; i<8; i++) LED8[i] = i;
```

```
    while(1)
    {
        while(!B_1ms);                          //等待 1ms 到
        B_1ms = 0;
        if(++j >= 500)                          //500ms 到
        {
            j = 0;
            for(i=0; i<8; i++) LED8[i] = k;     //刷新显示
            if(++k > 0x10) k = 0;               //8 个数码管循环显示 0,1,2...,A,B..F,消隐.
        }
    }
}
/*********************************************/
void Send_595(unsigned char dat)                //发送一个字节
{
    unsigned char i;
    for(i=0; i<8; i++)
    {
        if(dat & 0x80)      P_HC595_SER = 1;
        else                P_HC595_SER = 0;
        P_HC595_SRCLK = 1;
        P_HC595_SRCLK = 0;
        dat = dat << 1;
    }
}
/*********************************************/
void DisplayScan(void)                          //显示扫描函数
{
//  Send_595(~T_COM[display_index]);            //共阴  输出位码
//  Send_595(t_display[LED8[display_index]]);   //共阴  输出段码
    Send_595(T_COM[display_index]);             //共阳  输出位码
    Send_595(~t_display[LED8[display_index]]);  //共阳  输出段码
    P_HC595_RCLK = 1;
    P_HC595_RCLK = 0;                           //锁存输出数据
    if(++display_index >= 8) display_index = 0; //8 位结束回 0
}
/*********************************************/
void timer0 (void) interrupt 1 //Timer0 1ms 中断函数
{
    TH0 = (65536 - Timer0_Reload) / 256;        //重装定时值
    TL0 = (65536 - Timer0_Reload) % 256;

    DisplayScan();                              //1ms 扫描显示一位
    B_1ms = 1;                                  //1ms 标志
}
```

2. 汇编程序

```
;/*---------------------------------------------------------------------------*/
;/* --- STC 1T Series MCU Programme Demo -------------------------*/
;/*  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 ***************
;用 STC 的 MCU 的任意 I/O 方式控制 74HC595 驱动 8 位数码管
;用户可以在显示函数里修改成共阴或共阳.推荐尽量使用共阴数码管.
;显示效果为：8 个数码管循环显示 0,1,2,...,A,B,…,F,消隐.
;*******************************************/
;定义 Timer0 1ms 重装值

D_Timer0_Reload     EQU     (0 - 921)           ;1ms for 11.0592MHz
//D_Timer0_Reload   EQU     (0 - 1832)          ;1ms for 22.1184MHz

;/************* 本地变量声明 **************/
;P_HC595_SER        BIT     P3.2                ;pin 14 SER data input
;P_HC595_RCLK       BIT     P3.4                ;pin 12 RCLK store (latch) clock
;P_HC595_SRCLK      BIT     P3.3                ;pin 11 SRCLK Shift data clock

P_HC595_SER         BIT     P1.3                ;pin 14 SER data input
P_HC595_RCLK        BIT     P4.1                ;pin 12 RCLK store (latch) clock
P_HC595_SRCLK       BIT     P1.5                ;pin 11 SRCLK Shift data clock

LED8                EQU     030H
display_index       DATA    038H
FLAG0               DATA    20H
B_1ms               BIT     FLAG0.0
;*******************************************************************
    ORG     00H                             ;reset
    LJMP    F_MAIN_FUNC
    ORG     03H                             ;INT0 interrupt
;   LJMP    F_INT0_interrupt
    RETI

    ORG     0BH                             ;Timer0 interrupt
    LJMP    F_Timer0_interrupt
    RETI

    ORG     13H                             ;INT1 interrupt
;   LJMP    F_INT1_interrupt

    ORG     1BH                             ;Timer1 interrupt
;   LJMP    F_Timer1_interrupt
    RETI
;*******************************************************************
```

```
F_MAIN_FUNC:
    MOV     SP, #50H
    MOV     TMOD, #01H                  ;Timer 0 config as 16bit timer, 12T
    MOV     TH0, #HIGH D_Timer0_Reload  ;1ms
    MOV     TL0, #LOW D_Timer0_Reload
    SETB    ET0
    SETB    TR0
    SETB    EA
    MOV     R0, #LED8

L_InitLoop1:
    MOV     @R0, #10H                   ;上电消隐
    INC     R0
    MOV     A, R0
    CJNE    A, #(LED8+8), L_InitLoop1
    MOV     R2, #HIGH 500               ;500ms
    MOV     R3, #LOW 500
    MOV     R4, #0

L_MainLoop:
    JNB     B_1ms, $                    ;等待 1ms 到
    CLR     B_1ms
    MOV     A, R3
    CLR     C
    SUBB    A, #1
    MOV     R3, A
    MOV     A, R2
    SUBB    A, #0
    MOV     R2, A
    ORL     A, R3
    JNZ     L_MainLoop
    MOV     R2, #HIGH 500               ;500ms
    MOV     R3, #LOW 500
    MOV     R0, #LED8                   ;刷新显示

L_OptionLoop1:
    MOV     A, R4
    MOV     @R0, A
    INC     R0
    MOV     A, R0
    CJNE    A, #(LED8+8), L_OptionLoop1
    INC     R4
    MOV     A, R4
    CJNE    A, #11H, L_MainLoop         ;8 个数码管循环显示 0,1,2,...,A,B,…,F,消隐.
    MOV     R4, #0
    SJMP    L_MainLoop
```

```
;/*********************************************/
t_display:
; 0 1 2 3 4 5 6 7 8 9 A B C D E F  消隐
DB 03FH,006H,05BH,04FH,066H,06DH,07DH,007H,07FH,06FH,077H,07CH,039H,05EH,079H,071H,000H
;段码
T_COM:
        DB      01H,02H,04H,08H,10H,20H,40H,80H      ;位码
;/*********************************************/
F_Send_595:                                          ;发送一个字节
        MOV     R0, #8

L_Send595_Loop:
        RLC     A
        MOV     P_HC595_SER,C
        SETB    P_HC595_SRCLK
        CLR     P_HC595_SRCLK
        DJNZ    R0, L_Send595_Loop
        RET
;/*********************************************/
F_DisplayScan:                                       ;显示扫描函数
        MOV     DPTR, #T_COM
        MOV     A, display_index
        MOVC    A, @A+DPTR
;       CPL     A                                    ;共阴  共阳时注释掉本句
        LCALL   F_Send_595                           ;输出位码

        MOV     DPTR, #t_display
        MOV     A, #LED8
        ADD     A, display_index
        MOV     R0, A
        MOV     A, @R0
        MOVC    A, @A+DPTR
        CPL     A                                    ;共阳  共阴时注释掉本句
        LCALL   F_Send_595                           ;输出段码
        SETB    P_HC595_RCLK
        CLR     P_HC595_RCLK                         ;锁存输出数据
        INC     display_index
        MOV     A, display_index
        CJNE    A, #8, L_QuitDisplayScan
        MOV     display_index, #0                    ;8 位结束回 0

L_QuitDisplayScan:
        RET
;*********************************************************************
F_Timer0_interrupt:                                  ;Timer0 1ms 中断函数
        PUSH    PSW                                  ;现场保护
```

```
        PUSH    ACC
        MOV     A, R0
        PUSH    ACC
        PUSH    DPH
        PUSH    DPL

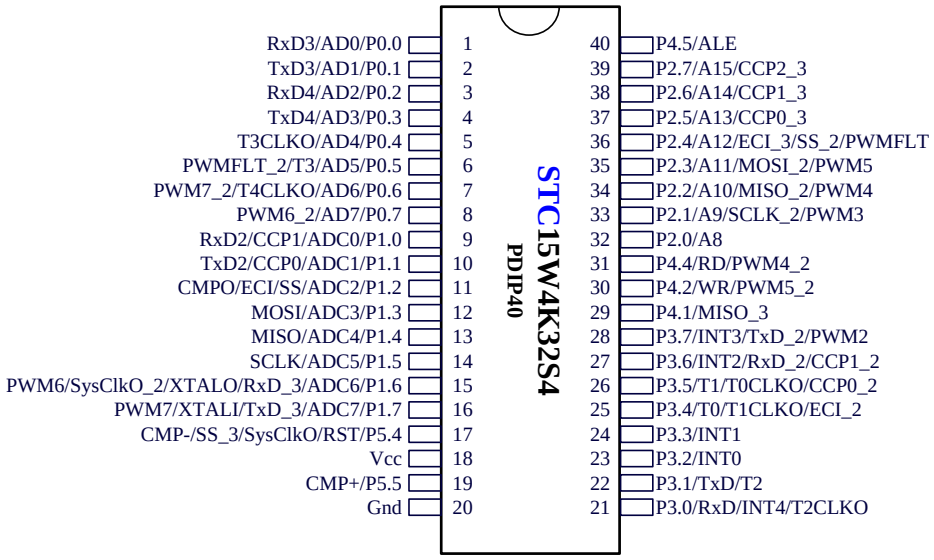
        MOV     TH0, #HIGH D_Timer0_Reload      ;1ms  重装定时值
        MOV     TL0, #LOW D_Timer0_Reload

        LCALL   F_DisplayScan                   ;1ms 扫描显示一位
        SETB    B_1ms                           ;1ms 标志

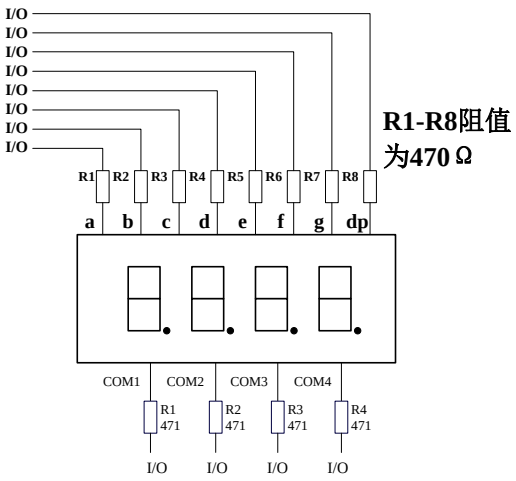
L_QuitT0Interrupt:
        POP     DPL                             ;现场恢复
        POP     DPH
        POP     ACC
        MOV     R0, A
        POP     ACC
        POP     PSW
        RETI

        END
```

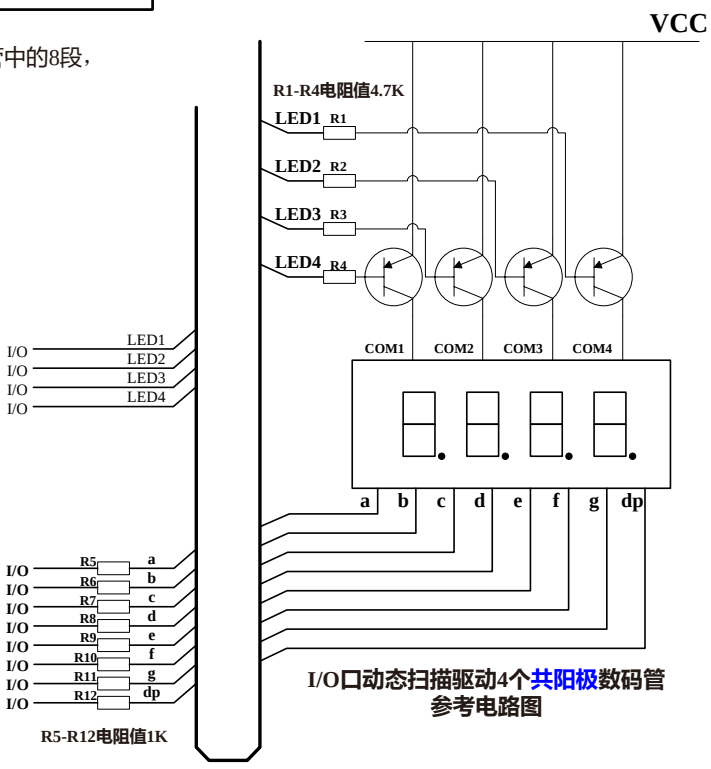
15.21 I/O 口直接驱动 LED 数码管应用线路图



I/O口动态扫描驱动数码管时，可以一次点亮一个数码管中的8段，但为降低功耗，建议可以一次只点亮其中的4段或者2段



I/O口动态扫描驱动4个共阴极数码管参考电路图



I/O口动态扫描驱动4个共阳极数码管参考电路图

15.22 用 STC MCU 的 I/O 口直接驱动段码 LCD 的原理及扫描程序

当产品需要段码 LCD 显示时，如果使用不带 LCD 驱动器的 MCU，则需要外接 LCD 驱动 IC，这会增加成本和 PCB 面积。事实上，很多小项目，比如大量的小家电，需要显示的段码不多，常见的是 4 个 8 带小数点或时钟的冒号“:”，这样如果使用 I/O 口直接扫描显示，则会减小 PCB 面积，降低成本。因此提出使用 STC MCU I/O 口直接驱动段码 LCD 的方案，段码 LCD 驱动简单原理如下图 1。但是，本方案不合适驱动太多的段(占用 I/O 太多)，也不合适非常低功耗的场合。

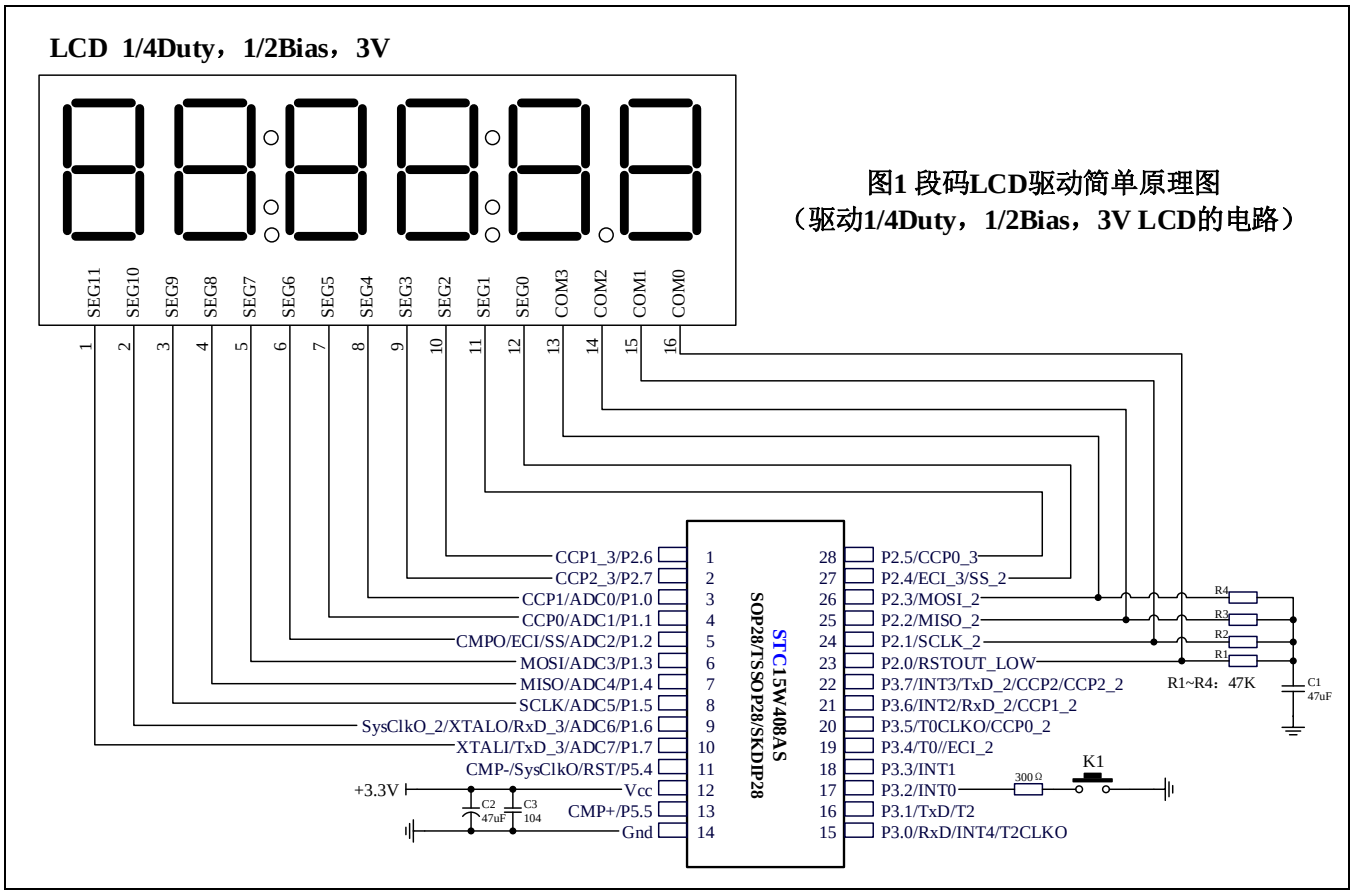


图1 段码LCD驱动简单原理图
（驱动1/4Duty，1/2Bias，3V LCD的电路）

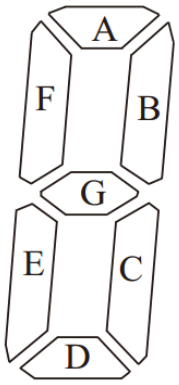


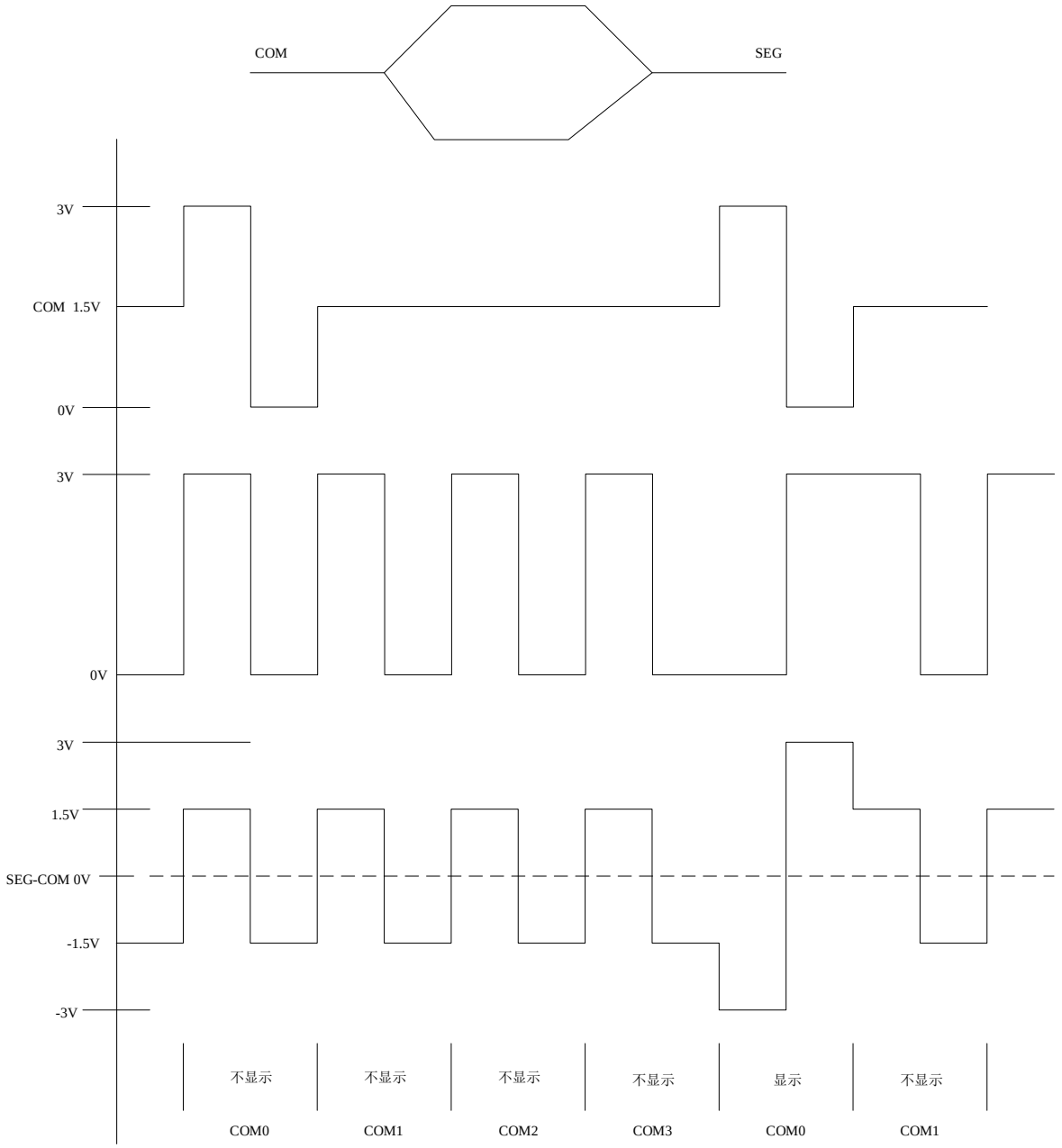
图2 段码名称图

LCD 是一种特殊的液态晶体，在电场的作用下晶体的排列方向会发生扭转，因而改变其透光性，从而可以看到显示内容。LCD 有一个扭转阀值，当 LCD 两端电压高于此阀值时，显示内容，低于此阀值时，不显示。通常 LCD 有 3 个参数：工作电压、DUTY（对应 COM 数）和 BIAS（即偏压，对应阀值），

比如 4.5V、1/4 DUTY、1/3 BIAS，表示 LCD 显示电压为 4.5V，4 个 COM，阀值大约是 1.5V，当加在某段 LCD 两端电压大于 1.5V 时（一般加 4.5V）显示，而加 1.5V 时不显示。但是 LCD 对于驱动电压的反应不是很明显的，比如加 2V 时，可能会微弱显示，这就是通常说的“鬼影”。所以要保证驱动显示时，要大于阀值电压比较多，而不显示时，要用比阀值小比较多的电压。

注意：LCD 的两端不能加直流电压，否则时间稍长就会损坏，所以要保证加在 LCD 两端的驱动电压的平均电压为 0。LCD 使用时分割扫描法，任何时候一个 COM 扫描有效，另外的 COM 处于无效状态。

图3 1/4Duty 1/2Bias LCD扫描原理图



驱动 1/4Duty 1/2BIAS 3V 的方案电路见图 1，LCD 扫描原理见图 3，MCU 为 3V 工作，用双向口做 COM，PUSH-PULL 或 STANDARD 输出口接 SEG，并且每个 COM 都接一个 47K 电阻到一个电容，RC

滤波后得到一个中点电压。在轮到某个 COM 扫描时，设置成 PUSH-PULL 输出，如果与本 COM 连接的 SEG 不显示，则 SEG 输出与 COM 同相，如果显示，则反相。扫完后，这个 COM 的 I/O 就设置成高阻，这样这个 COM 就通过 47K 电阻连接到 1/2VDD 电压，而 SEG 继续输出方波，这样加在 LCD 上的电压，显示时是+-VDD，不显示时是+-1/2VDD，保证了 LCD 两端平均直流电压为 0。

驱动 1/4Duty 1/3BIAS 3V 的方案电路见图 4，LCD 扫描原理见图 5，MCU 为 5V 工作，SEG 线通过电阻分压输出 1.5V、3.5V，COM 线通过电阻分压输出 0.5V、2.5V（高阻时）、4.5V。在轮到某个 COM 扫描时，设置成 PUSH-PULL 输出，如果与本 COM 连接的 SEG 不显示，则 SEG 输出与 COM 同相，如果显示，则反相。扫描完后，这个 COM 的 I/O 就设置成高阻，这样这个 COM 就通过 47K 电阻连接到 2.5V 电压，而 SEG 继续输出方波，这样加在 LCD 上的电压，显示时是+-3.0V，不显示时是+-1.0V，完全满足 LCD 的扫描要求。

当需要睡眠省电时，把所有 COM 和 SEG 驱动 I/O 全部输出低电平，LCD 驱动部分不会增加额外电流。

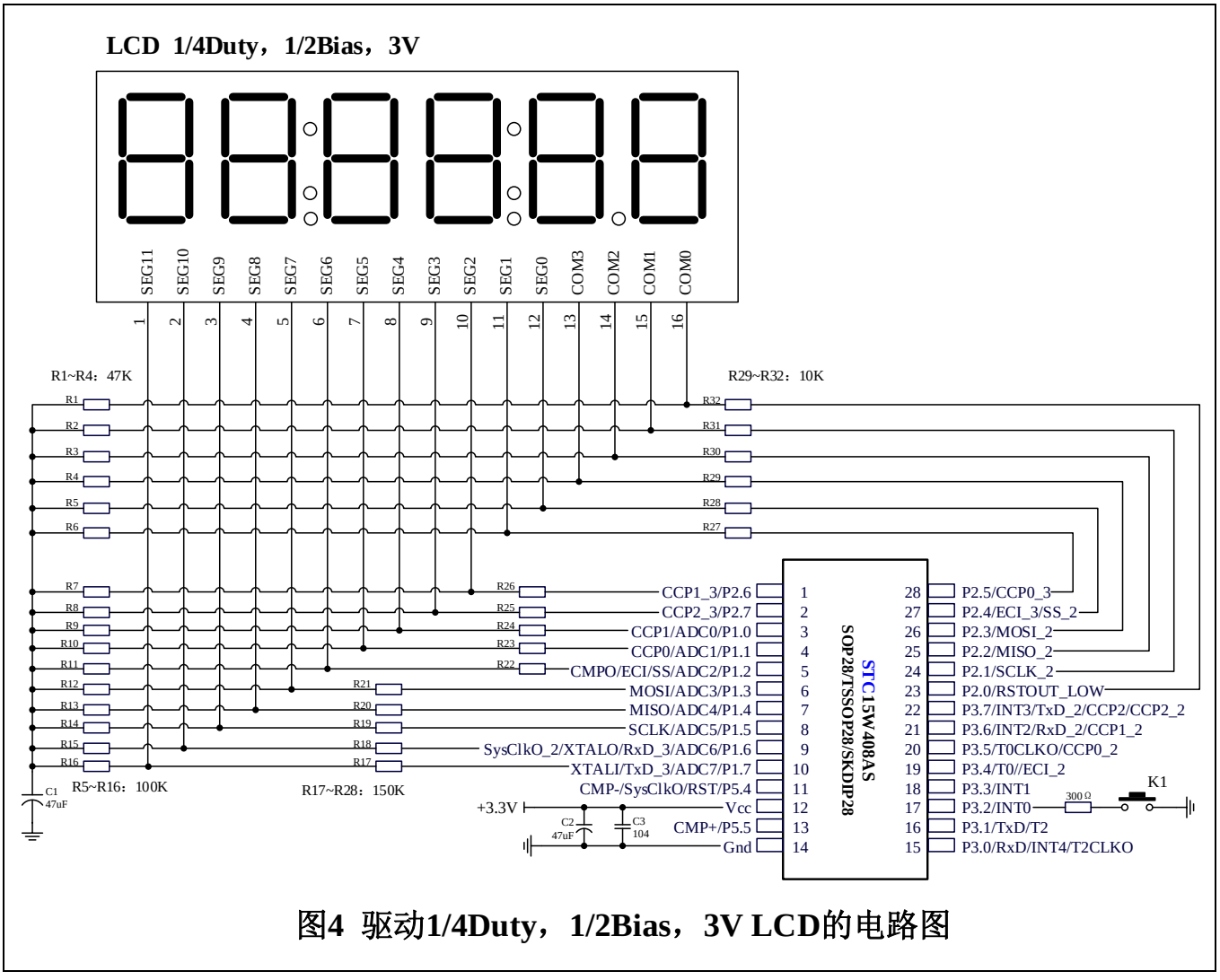


图4 驱动1/4Duty，1/2Bias，3V LCD的电路图

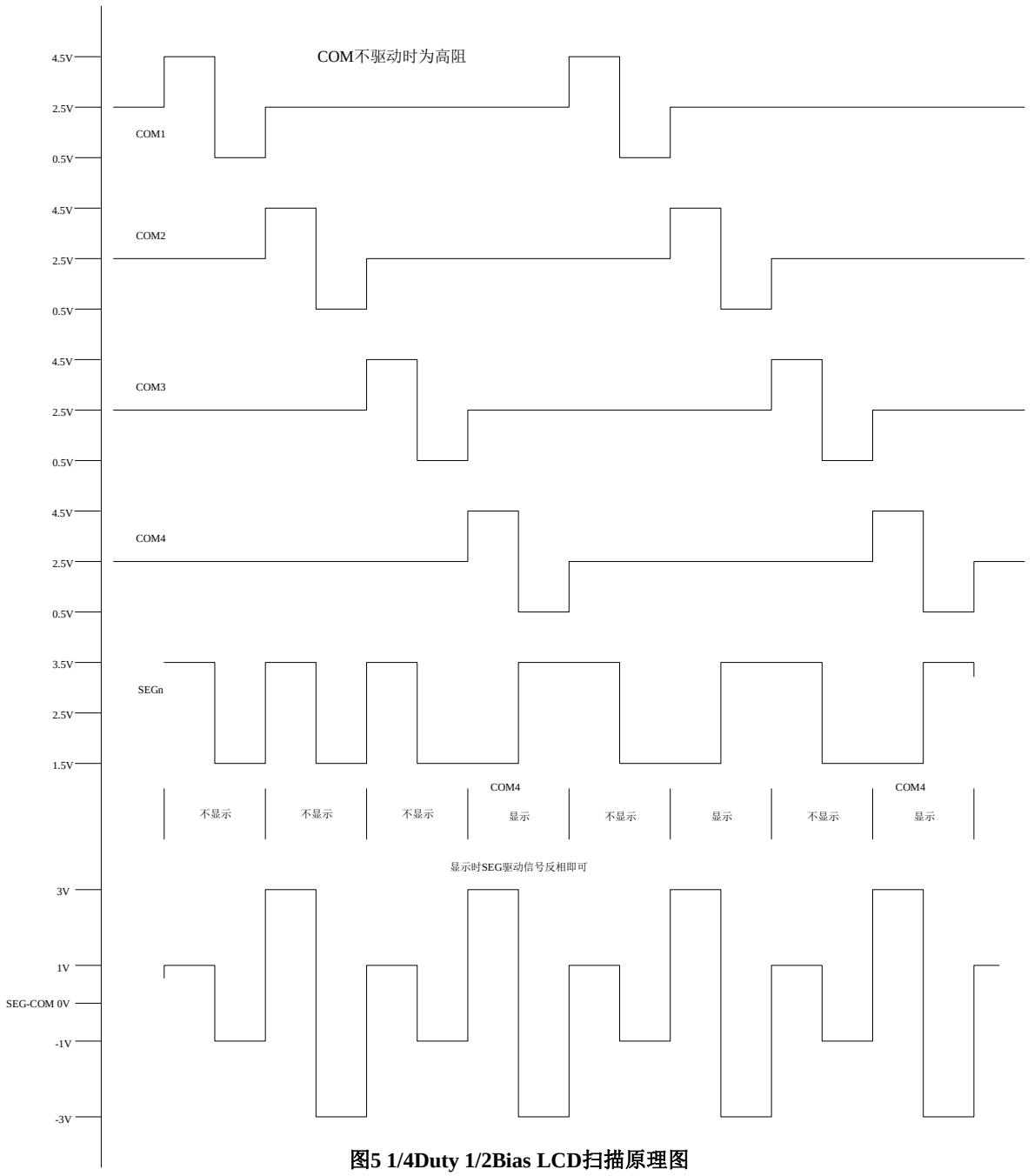


图5 1/4Duty 1/2Bias LCD扫描原理图

为使用方便，显示内容放在一个显存中，其中的各个位与 LCD 的段一一对应，见图 6。

图 6：LCD 真值表和显存影射表

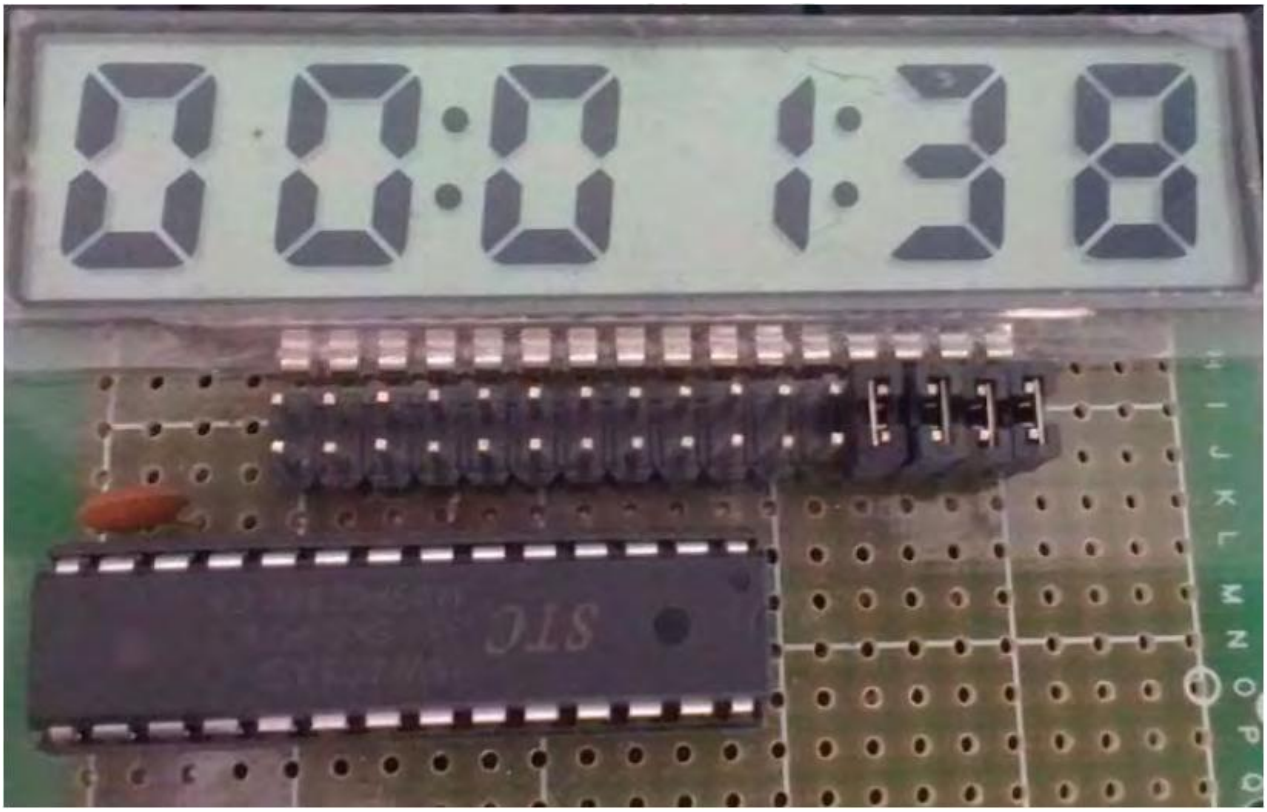
LCD 真值表：

MCU PIN	P17	P16	P15	P14	P13	P12	P11	P10	P27	P26	P25	P24	P23	P22	P21	P20
LCD PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
LCD PIN name	SEG11	SEG10	SEG9	SEG8	SEG7	EG6	SEG5	SEG4	SEG3	SEG2	SEG1	SEG0	COM3	COM2	COM1	COM0
	--	1D	2:	2D	2.	3D	4:	4D	4.	5D	5.	6D	COM3			
	1E	1C	2E	2C	3E	3C	4E	4C	5E	5C	6E	6C		COM2		
	1G	1B	2G	2B	3G	3B	4G	4B	5G	5B	6G	6B			COM1	
	1F	1A	2F	2A	3F	3A	4F	4A	5F	5A	6F	6A				COM0

显存影射表:

	B7	B6	B5	B4	B3	B2	B1	B0
buff[0]:	--	1D	2:	2D	2.	3D	4:	4D
buff[1]:	1E	1C	2E	2C	3E	3C	4E	4C
buff[2]:	1G	1B	2G	2B	3G	3B	4G	4B
buff[3]:	1F	1A	2F	2A	3F	3A	4F	4A
buff[4]:	4.	5D	5.	6D	--	--	--	--
buff[5]:	5E	5C	6E	6C	--	--	--	--
buff[6]:	5G	5B	6G	6B	--	--	--	--
buff[7]:	5F	5A	6F	6A	--	--	--	--

图 7：驱动效果照片



本 LCD 扫描程序仅需要两个函数:

1、LCD 段码扫描函数　　void　　LCD_scan(void)
程序隔一定的时间调用这个函数，就会将 LCD 显示缓冲的内容显示到 LCD 上，全部扫描一次需要 8 个调用周期，调用间隔一般是 1 ~ 2ms，假如使用 1ms，则扫描周期就是 8ms，刷新率就是 125Hz。

2、LCD 段码显示缓冲装载函数　　void　　LCD_load(u8 n,u8 dat)
本函数用来将显示的数字或字符放在 LCD 显示缓冲中，比如 LCD_load(1,6)，就是要在第一个数字位置显示数字 6，支持显示 0 ~ 9，A ~ F，其它字符用户可以自己添加。
另外，用宏来显示、熄灭或闪烁冒号或小数点。

```
/****************** LCD 段码扫描函数 ****************************
u8      code T_COM[4] = {0x08,0x04,0x02,0x01};

void    LCD_scan(void)                          //5us @22.1184MHZ
{
    u8  j;
```

```
    j = scan_index >> 1;                    //COMx
    P2n_pure_input(0x0f);                   //全部 COM 输出高阻, COM 为中点电压

    if(scan_index & 1)                      //反相扫描
    {
        P1 = ~LCD_buff[j];                  //送 SEG 驱动码
        P2 = ~(LCD_buff[j|4] & 0xf0);       //送 SEG 驱动码和 COM 驱动码
    }
    else                                    //正相扫描
    {
        P1 = LCD_buff[j];                   //送 SEG 驱动码
        P2 = LCD_buff[j|4] & 0xf0;          //送 SEG 驱动码和 COM 驱动码
    }

    P2n_push_pull(T_COM[j]);                //某个 COM 设置为推挽输出
    if(++scan_index >= 8)   scan_index = 0; //扫描完成，重复扫描
}

/****************** LCD 段码显示缓冲装载函数 ****************************/
/******************  对第 1~6 数字装载显示函数 ****************************/
u8  code T_LCD_mask[4] = {~0xc0, ~0x30, ~0x0c, ~0x03};
u8  code T_LCD_mask4[4] = {~0x40, ~0x10, ~0x04, ~0x01};

void LCD_load(u8 n,u8 dat)              //n 为第几个数字，为 1~6，dat 为要显示的数字 10us@22.1184MHz
{
    u8  i, k;
    u8  *p;
    if((n == 0) || (n > 6))     return;
    i = t_display[dat];
    if(n <= 4)//1~4
    {
        n--;
        p = LCD_buff;
    }
    else
    {
        n = n - 5;
        p = &LCD_buff[4];
    }

    k = 0;
    if(i & 0x08)    k |= 0x40;              //D
    *p = (*p & T_LCD_mask4[n]) | (k>>2*n);
    p++;

    k = 0;
```

```
    if(i & 0x04)    k |= 0x40;                  //C
    if(i & 0x10)    k |= 0x80;                  //E
    *p = (*p & T_LCD_mask[n]) | (k>>2*n);
    p++;

    k = 0;
    if(i & 0x02)    k |= 0x40;                  //B
    if(i & 0x40)    k |= 0x80;                  //G
    *p = (*p & T_LCD_mask[n]) | (k>>2*n);
    p++;

    k = 0;
    if(i & 0x01)    k |= 0x40;                  //A
    if(i & 0x20)    k |= 0x80;                  //F
    *p = (*p & T_LCD_mask[n]) | (k>>2*n);
    }
```

详细的程序如下，用户也可从 STC 的官网 www.STCAI.com 下载

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC 1T Series MCU Demo Programme -----------------------------------------*/
/*---------------------------------------------------------------------------------------------*/
#include <intrins.h>
#include "config.h"
#include "timer.h"
#include "LCD_IO16.h"
/************* 功能说明 **************
用 STC115 系列测试 I/O 直接驱动段码 LCD(6 个 8 字 LCD, 1/4 Dutys, 1/3 bias)
上电后显示一个时间(时分秒).
P3.2 对地接一个开关，用来进入睡眠或唤醒.
*****************************************/
/************* 本地变量声明 **************/
u8      cnt_500ms;
u8      second, minute, hour;
bit     B_Second;                               //秒信号
/****************** 本地函数声明 ********************/
/************* 外部函数和变量声明 *****************/
extern  bit B_2ms;
/****************** 定时器配置 ************************/
void Timer_config(void)
{
    TIM_InitTypeDef   TIM_InitStructure;        //结构定义
    TIM_InitStructure.TIM_Mode = TIM_16BitAutoReload;
//指定工作模式, TIM_16BitAutoReload,TIM_16Bit,TIM_8BitAutoReload,TIM_16BitAutoReloadNoMask
    TIM_InitStructure.TIM_Polity = PolityLow;       //指定中断优先级, PolityHigh, PolityLow
    TIM_InitStructure.TIM_Interrupt = ENABLE;       //中断是否允许, ENABLE 或 DISABLE
```

```
    TIM_InitStructure.TIM_ClkSource = TIM_CLOCK_1T;
                                //指定时钟源, TIM_CLOCK_1T,TIM_CLOCK_12T,TIM_CLOCK_Ext

    TIM_InitStructure.TIM_ClkOut = DISABLE;         //是否输出高速脉冲, ENABLE 或 DISABLE

    TIM_InitStructure.TIM_Value = 65536 - (MAIN_Fosc / 500);        //初值, 节拍为 500HZ
    TIM_InitStructure.TIM_Run = ENABLE;             //是否初始化后启动定时器, ENABLE 或 DISABLE

    Timer_Inilize(Timer0,&TIM_InitStructure);       //初始化 Timer0, Timer0,Timer1,Timer2
    }
/********************  显示时间  *************************/
void LoadRTC(void)
{
    LCD_load(1,hour/10);
    LCD_load(2,hour%10);
    LCD_load(3,minute/10);
    LCD_load(4,minute%10);
    LCD_load(5,second/10);
    LCD_load(6,second%10);
}
//========================================================================
// 函数: void delay_ms(unsigned char 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.
// 备注:
//========================================================================
void delay_ms(u8 ms)
{
    unsigned int i;
    do
    {
        i = MAIN_Fosc / 13000;
        while(--i) ;                            //14T per loop
    }while(--ms);
}

/********************  主函数*************************/
void main(void)
{
    Init_LCD_Buffer();
    Timer_config();
    EA = 1;
    LCD_SET_2M;                                 //显示时分间隔:
    LCD_SET_4M;                                 //显示分秒间隔:
    LoadRTC();                                  //显示时间
```

```
while (1)
{
    PCON |= 0x01;                       //为了省电, 进入空闲模式(电流大约 2.5 ~ 3.0mA @5V),
                                        //由 Timer0 2ms 唤醒退出
    _nop_();
    _nop_();
    _nop_();
    if(B_2ms)                           //2ms 节拍到
    {
        B_2ms = 0;
        if(++cnt_500ms >= 250)          //500ms 到
        {
            cnt_500ms = 0;
        //  LCD_FLASH_2M;               //闪烁时分间隔:
        //  LCD_FLASH_4M;               //闪烁分秒间隔:
            B_Second = ~B_Second;
            if(B_Second)
            {
                if(++second >= 60)      //1 分钟到
                {
                    second = 0;
                    if(++minute >= 60)  //1 小时到
                    {
                        minute = 0;
                        if(++hour >= 24) hour = 0;      //24 小时到
                    }
                }
                LoadRTC();              //显示时间
            }
        }
        if(!P32)                        //键按下，准备睡眠
        {
            LCD_CLR_2M;                 //显示时分间隔:
            LCD_CLR_4M;                 //显示分秒间隔:
            LCD_load(1,DIS_BLACK);
            LCD_load(2,DIS_BLACK);
            LCD_load(3,0);
            LCD_load(4,0x0F);
            LCD_load(5,0x0F);
            LCD_load(6,DIS_BLACK);

            while(!P32) delay_ms(10);   //等待释放按键
            delay_ms(50);
            while(!P32) delay_ms(10);   //再次等待释放按键
            TR0 = 0;                    //关闭定时器
            IE0 = 0;                    //外中断 0 标志位
```

```
                EX0 = 1;                        //INT0 Enable
                IT0 = 1;                        //INT0  下降沿中断

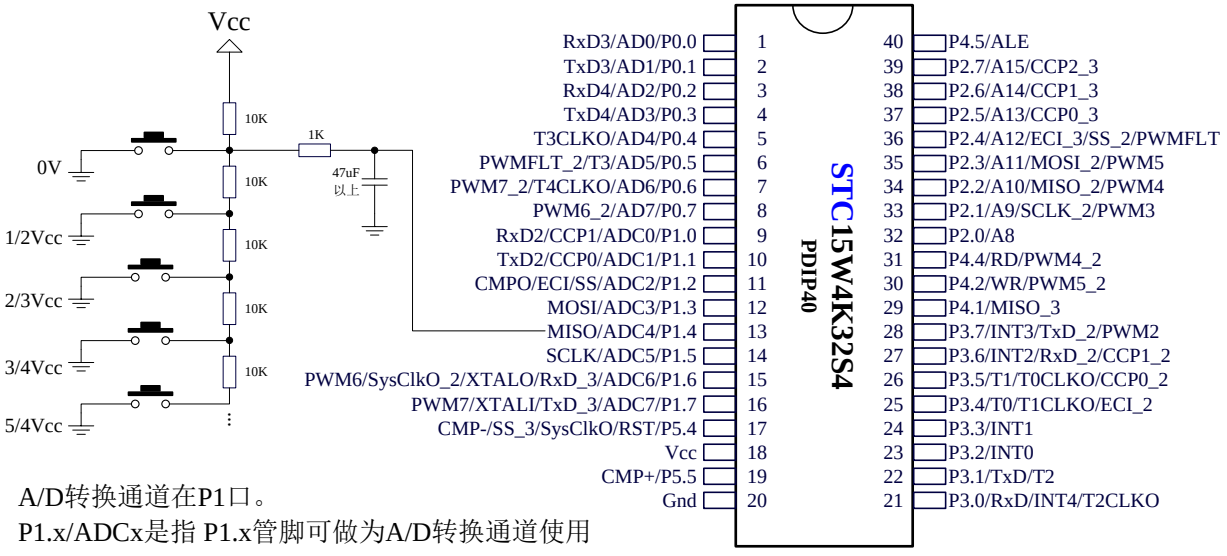
                P1n_push_pull(0xff);            //com 和 seg 全部输出 0
                P2n_push_pull(0xff);            //com 和 seg 全部输出 0
                P1 = 0;
                P2 = 0;

                PCON |= 0x02;                   //Sleep
                _nop_();
                _nop_();
                _nop_();

                LCD_SET_2M;                     //显示时分间隔:
                LCD_SET_4M;                     //显示分秒间隔:
                LoadRTC();                      //显示时间
                TR0 = 1;                        //打开定时器
                while(!P32)    delay_ms(10);    //等待释放按键
                delay_ms(50);
                while(!P32)    delay_ms(10);    //再次等待释放按键
            }
        }
    }
}

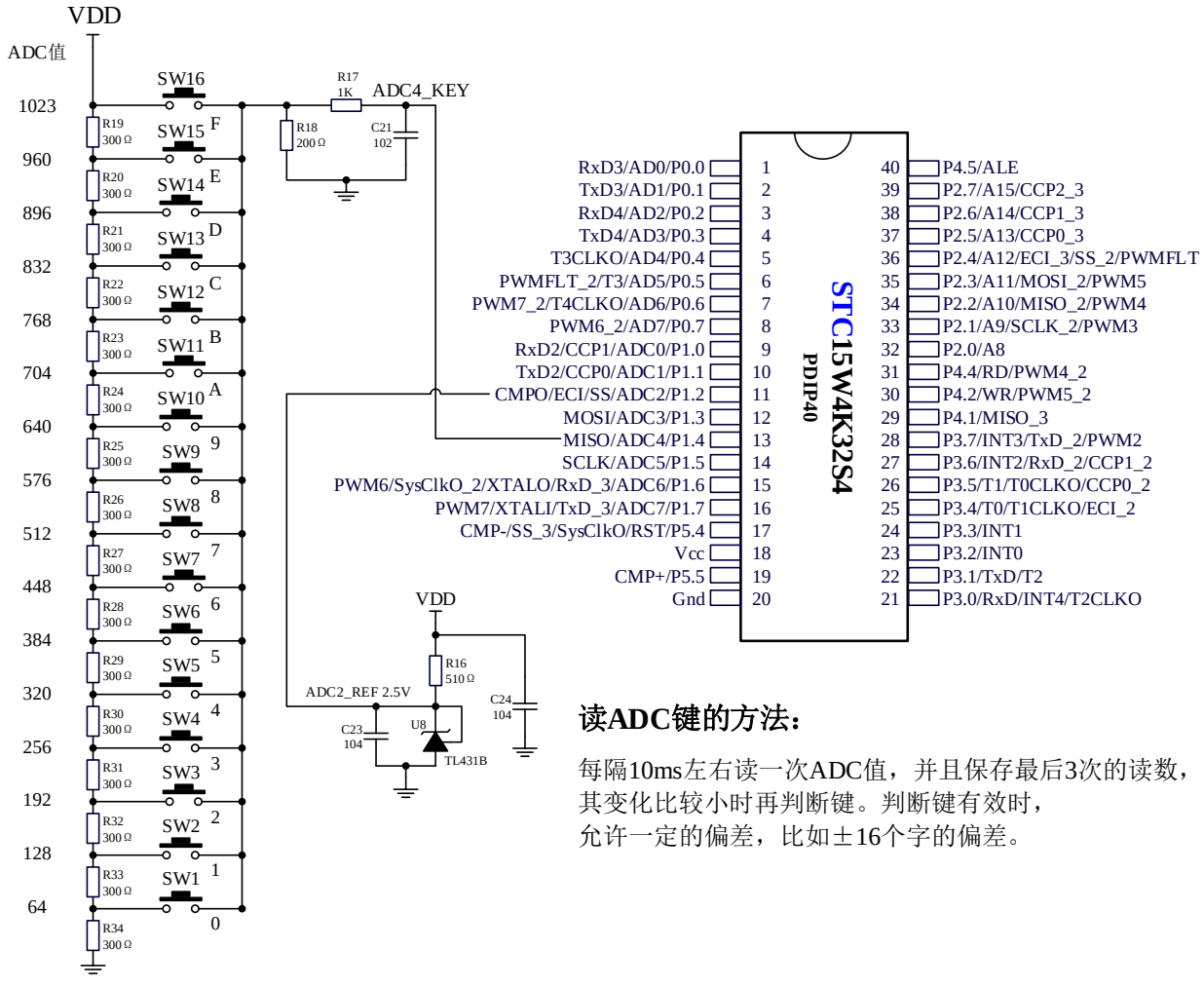
/********************** INT0 中断函数  *************************/
void INT0_int (void) interrupt INT0_VECTOR
{
    EX0 = 0;                                    //INT0 Disable
    IE0 = 0;                                    //外中断 0 标志位
}
```

15.23 A/D 做按键扫描应用线路图



A/D转换通道在P1口。
P1.x/ADCx是指 P1.x管脚可做为A/D转换通道使用

A/D转换在P1口，P1.0 - P1.7共8路



读ADC键的方法：

每隔10ms左右读一次ADC值，并且保存最后3次的读数，
其变化比较小时再判断键。判断键有效时，
允许一定的偏差，比如±16个字的偏差。

15.24 STC15 系列单片机 I/O 口软件模拟 I²C 接口的测试程序

15.24.1 STC15 系列单片机 I/O 口软件模拟 I²C 接口的主机模式

```
;/*------------------------------------------------------------------------------*/
;/* --- STC 1T Series MCU Simulate I2C Master Demo ----------------*/
;/*------------------------------------------------------------------------------*/
SCL     BIT     P1.0
SDA     BIT     P1.1
;-----------------------------
        ORG     0000H
        MOV     TMOD, #20H              ;初始化串口为(9600, n, 8 , 1)
        MOV     SCON, #5AH
        MOV     A, #-5                  ;-18432000/12/32/9600
        MOV     TH1, A
        MOV     TL1, A
        SETB    TR1
START:
        CALL    UART_RXDATA             ;接收下一个串口数据
        MOV     R0, A                   ;临时保存到 R0
                                        ;读取 I2C 设备 IDATA 80H 的数据
        CALL    I2C_START               ;开始读取
        MOV     A, #01H
        CALL    I2C_TXBYTE              ;发送地址数据+读信号
        CALL    I2C_RXACK               ;接收 ACK
        CALL    I2C_RXBYTE              ;接收数据
        SETB    C
        CALL    I2C_TXACK               ;发送 NAK
        CALL    I2C_STOP                ;读取完成
        CALL    UART_TXDATA             ;将读到的数据发送到串口
                                        ;将 R0 的数据写入 I2C 设备 IDATA 80H
        CALL    I2C_START               ;开始写
        MOV     A, #00H
        CALL    I2C_TXBYTE              ;发送地址数据+写信号
        CALL    I2C_RXACK               ;接收 ACK
        MOV     A, R0
        CALL    I2C_TXBYTE              ;写数据
        CALL    I2C_RXACK               ;接收 ACK
        CALL    I2C_STOP                ;写完成
        JMP     START
;-----------------------------
;等待串口数据
;-----------------------------
UART_RXDATA:
```

```
    JNB     RI, $                   ;等待接收完成标志
    CLR     RI                      ;清除标志
    MOV     A, SBUF                 ;保存数据
    RET
;----------------------------
;发送串口数据
;----------------------------
UART_TXDATA:
    JNB     TI, $                   ;等待上一个数据发送完成
    CLR     TI                      ;清除标志
    MOV     SBUF, A                 ;发送数据
    RET
;----------------------------
;发送 I2C 起始信号
;----------------------------
I2C_START:
    CLR     SDA                     ;数据线下降沿
    CALL    I2C_DELAY               ;延时
    CLR     SCL                     ;时钟→低
    CALL    I2C_DELAY               ;延时
    RET
;----------------------------
;发送 I2C 停止信号
;----------------------------
I2C_STOP:
    CLR     SDA
    SETB    SCL                     ;时钟→高
    CALL    I2C_DELAY               ;延时
    SETB    SDA                     ;数据线上升沿
    CALL    I2C_DELAY               ;延时
    RET
;----------------------------
;发送 ACK/NAK 信号
;----------------------------
I2C_TXACK:
    MOV     SDA, C                  ;送 ACK 数据
    SETB    SCL                     ;时钟→高
    CALL    I2C_DELAY               ;延时
    CLR     SCL                     ;时钟→低
    CALL    I2C_DELAY               ;延时
    SETB    SDA                     ;发送完成
    RET
;----------------------------
;接收 ACK/NAK 信号
;----------------------------
I2C_RXACK:
```

```
    SETB    SDA             ;准备读数据
    SETB    SCL             ;时钟→高
    CALL    I2C_DELAY       ;延时
    MOV     C, SDA          ;读取 ACK 信号
    CLR     SCL             ;时钟→低
    CALL    I2C_DELAY       ;延时
    RET
;-----------------------------
;接收一字节数据
;-----------------------------
I2C_TXBYTE:
    MOV     R7, #8          ;8 位计数
TXNEXT:
    RLC     A               ;移出数据位
    MOV     SDA, C          ;数据送数据口
    SETB    SCL             ;时钟→高
    CALL    I2C_DELAY       ;延时
    CLR     SCL             ;时钟→低
    CALL    I2C_DELAY       ;延时
    DJNZ    R7, TXNEXT      ;送下一位
    RET
;-----------------------------
;发送一字节数据
;-----------------------------
I2C_RXBYTE:
    MOV     R7, #8          ;8 位计数
RXNEXT:
    SETB    SCL             ;时钟→高
    CALL    I2C_DELAY       ;延时
    MOV     C, SDA
    RLC     A
    CLR     SCL             ;时钟→低
    CALL    I2C_DELAY       ;延时
    DJNZ    R7, RXNEXT      ;收下一位
    RET
;-----------------------------
I2C_DELAY:                  ;6
    PUSH    0               ;4
    MOV     R0, #1          ;2 6(200K) 1(400K) [18'432'000/400'000=46]
    DJNZ    R0, $           ;4
    POP     0               ;3
    RET                     ;4
;-----------------------------
END
```

15.24.2 STC15 系列单片机 I/O 口软件模拟 I²C 接口的从机模式

```
;/*-------------------------------------------------------------------------*/
;/* --- STC 1T Series MCU Simulate I2C Slave Demo ----------------*/
;/*-------------------------------------------------------------------------*/
    SCL     BIT   P1.0
    SDA     BIT   P1.1
;----------------------------
    ORG     0
RESET:
    SETB    SCL
    SETB    SDA
    CALL    I2C_WAITSTART           ;等待起始信号
    CALL    I2C_RXBYTE              ;接收地址数据
    CLR     C
    CALL    I2C_TXACK               ;回应 ACK
    SETB    C                       ;读/写  IDATA[80H - FFH]
    RRC     A                       ;读/写位→C
    MOV     R0, A                   ;地址送入 R0
    JC      READDATA                ;C=1(读) C=0(写)

WRITEDATA:
    CALL    I2C_RXBYTE              ;接收数据
    MOV     @R0, A                  ;写入 IDATA
    INC     R0                          ;地址+1
    CLR     C
    CALL    I2C_TXACK               ;回应 ACK
    CALL    I2C_WAITSTOP            ;等待停止信号
    JMP     RESET

READDATA:
    MOV     A, @R0
    INC     R0
    CALL    I2C_TXBYTE              ;发送 IDATA 数据
    CALL    I2C_RXACK               ;接收 ACK
    CALL    I2C_WAITSTOP            ;等待停止信号
    JMP     RESET

;----------------------------等待起始信号----------------------------
I2C_WAITSTART:
    JNB     SCL, $                  ;等待时钟→高
    JB      SDA, $                  ;等待数据线下降沿
    JB      SCL, $                  ;等待时钟→低
    RET
;----------------------------等待结束信号----------------------------
```

```
I2C_WAITSTOP:
    JNB     SCL, $                  ;等待时钟→高
    JNB     SDA, $                  ;等待数据线上升沿
    RET

;----------------------------发送 ACK/NAK 信号----------------------------
I2C_TXACK:
    MOV     SDA, C                  ;送 ACK 数据
    JNB     SCL, $                  ;等待时钟→高
    JB      SCL, $                  ;等待时钟→低
    SETB    SDA                     ;发送完成
    RET

;----------------------------接收 ACK/NAK 信号----------------------------
I2C_RXACK:
    SETB    SDA                     ;准备读数据
    JNB     SCL, $                  ;等待时钟→高
    MOV     C, SDA                  ;读取 ACK 信号
    JB      SCL, $                  ;等待时钟→低
    RET

;--------------------------接收一字节数据--------------------------
I2C_RXBYTE:
    MOV     R7, #8                  ;8 位计数
RXNEXT:
    JNB     SCL, $                  ;等待时钟→高
    MOV     C, SDA                  ;读取数据口
    RLC     A                       ;保存数据
    JB      SCL, $                  ;等待时钟→低
    DJNZ    R7, RXNEXT              ;收下一位
    RET
;----------------------------发送一字节数据----------------------------
I2C_TXBYTE:
    MOV     R7, #8                  ;8 位计数
TXNEXT:
    RLC     A                       ;移出数据位
    MOV     SDA, C                  ;数据送数据口
    JNB     SCL, $                  ;等待时钟→高
    JB      SCL, $                  ;等待时钟→低
    DJNZ    R7, TXNEXT              ;送下一位
    RET
;----------------------------
END
```

16 指令系统

16.1 寻址方式

寻址方式是每一种计算机的指令集中不可缺少的部分。寻址方式规定了数据的来源和目的地。对不同的程序指令，来源和目的地的规定也会不同。在 STC 单片机中的寻址方式可概括为:

- 立即寻址
- 直接寻址
- 间接寻址
- 寄存器寻址
- 相对寻址
- 变址寻址
- 位寻址

16.1.1 立即寻址

立即寻址也称立即数，它是在指令操作数中直接给出参加运算的操作数，其指令格式如下：

MOV A, #70H

这条指令的功能是将立即数 70H 传送到累加器 A 中

16.1.2 直接寻址

在直接寻址方式中，指令操作数域给出的是参加运算操作数地址。直接寻址方式只能用来表示特殊功能寄存器、内部数据寄存器和位地址空间。其中特殊功能寄存器和位地址空间只能用直接寻址方式访问。

如：ANL 70H, #48H

表示 70H 单元中的数与立即数 48H 相“与”，结果存放在 70H 单元中。其中 70H 为直接地址，表示内部数据存储器 RAM 中的一个单元。

16.1.3 间接寻址

间接寻址采用 R0 或 R1 前添加“@”符号来表示。例如，假设 R1 中的数据是 40H，内部数据存储器 40H 单元所包含的数据为 55H，那么如下指令：

MOV A, @RI

把数据 55H 传送到累加器。

16.1.4 寄存器寻址

寄存器寻址是对选定的工作寄存器 R7 ~ R0、累加器 A、通用寄存器 B、地址寄存器和进位 C 中的数进行操作。其中寄存器 R7~R0 由指令码的低 3 位表示，ACC、B、DPTR 及进位位 C 隐含在指令码中。因此，寄存器寻址也包含一种隐含寻址方式。

寄存器工作区的选择由程序状态字寄存器 PSW 中的 RS1、RS0 来决定。指令操作数指定的寄存器均指当前工作区中的寄存器。

如：INC R0 ;(R0)+1 → R0

16.1.5 相对寻址

相对寻址是将程序计数器 PC 中的当前值与指令第二字节给出的数相加，其结果作为转移指令的转移地址。转移地址也称为转移目的地址，PC 中的当前值称为基地址，指令第二字节给出的数称为偏移量。由于目的地址是相对于 PC 中的基地址而言，所以这种寻址方式称为相对寻址。偏移量为带符号的数，所能表示的范围为+127 ~ -128。这种寻址方式主要用于转移指令。

如：JC 80H ;C=1 跳转

表示若进位位 C 为 0，则程序计数器 PC 中的内容不改变，即不转移。若进位位 C 为 1，则以 PC 中的当前值为基地址，加上偏移量 80H 后所得到的结果作为该转移指令的目的地址。

16.1.6 变址寻址

在变址寻址方式中，指令操作数指定一个存放变址基值的变址寄存器。变址寻址时，偏移量与变址基值相加，其结果作为操作数的地址。变址寄存器有程序计数器 PC 和地址寄存器 DPTR。

如：MOVC A, @A+DPTR

表示累加器 A 为偏移量寄存器，其内容与地址寄存器 DPTR 中的内容相加，其结果作为操作数的地址，取出该单元中的数送入累加器 A。

16.1.7 位寻址

位寻址是指对一些内部数据存储器 RAM 和特殊功能寄存器进行位操作时的寻址。在进行位操作时，借助于进位位 C 作为位操作累加器，指令操作数直接给出该位的地址，然后根据操作码的性质对该位进行位操作。位地址与字节直接寻址中的字节地址形式完全一样，主要由操作码加以区分，使用时应注意。

如：MOV C, 20H ;片内位单元位操作型指令

16.2 完整指令集对照表(与传统 8051 对照)

----共 111 条指令，每条指令的详细执行时间
----与普通 8051 指令代码完全兼容，但执行的时间效率大幅提升
----其中 INC DPTR 和 MUL AB 指令的执行速度大幅提升 24 倍
----共有 22 条指令，一个时钟就可以执行完成，平均速度快 8 ~ 12 倍

如果按功能分类，STC15 系列单片机指令系统可分为:

1.算术操作类指令;
2.逻辑操作类指令;
3.数据传送类指令;
4.布尔变量操作类指令:
5.控制转移类指令。

按功能分类的指令系统表如下表所示。

指令执行速度效率提升总结(新 15 系列);

指令系统共包括 111 条指令，其中：

执行速度快 24 倍的	共 2 条
执行速度快 12 倍的	共 28 条
执行速度快 8 倍的	共 19 条
执行速度快 6 倍的	共 40 条
执行速度快 4.8 倍的	共 8 条
执行速度快 4 倍的	共 14 条

根据对指令的使用频率分析统计，STC15 系列 1T 的 8051 单片机比普通的 8051 单片机在同样的工作频率下运行速度提升了 8 ~ 12 倍。

指令执行时钟数统计(供参考)(新 15 系列):

指令系统共包括 111 条指令，其中:

1 个时钟就可执行完成的指令	共 22 条
2 个时钟就可执行完成的指令	共 37 条
3 个时钟就可执行完成的指令	共 31 条
4 个时钟就可执行完成的指令	共 12 条
5 个时钟就可执行完成的指令	共 8 条
6 个时钟就可执行完成的指令	共 1 条

STC15 系列将 111 条指令全部执行完一遍所需的时钟为 283 个时钟，而传统 8051 单片机将 111 条指令全部执行一遍要 1944 个时钟。可见与传统 8051 相比较，STC 新 15 系列的指令执行速度大幅提升，平均速度快 8 ~ 12 倍。

现 STC15 系列单片机采用 STC-Y5 超高速 CPU 内核，在相同的时钟频率下，速度又比 STC 早期的 1T 系列单片机(如 STC12 系列/STC11 系列/STC10 系列)的速度快 20%。

算术操作类指令

助记符	功能说明	字节数	传统 8051 单片机所需时钟	STC15 系列单片机所需时钟 (采用 STC-Y5 超高速 1T 8051 内核)	效率提升
ADD A, Rn	寄存器内容加到累加器	1	12	1	12 倍
ADD A, direct	直接地址单元中的数据加到累加器	2	12	2	6 倍
ADD A, @Ri	间接 RAM 中的数据加到累加器	1	12	2	6 倍
ADD A, #data	立即数加到累加器	2	12	2	6 倍
ADDC A, Rn	寄存器带进位加到累加器	1	12	1	12 倍
ADDC A, direct	直接地址单元的内容带进位加到累加器	2	12	2	6 倍
ADDC A, @Ri	间接 RAM 内容带进位加到累加器	1	12	2	6 倍
ADDC A, #data	立即数带进位加到累加器	2	12	2	6 倍
SUBB A, Rn	累加器带借位减寄存器内容	1	12	1	6 倍
SUBB A, direct	累加器带借位减直接地址单元的内容	2	12	2	6 倍
SUBB A, @Ri	累加器带借位减间接 RAM 中的内容	1	12	2	6 倍
SUBB A, #data	累加器带借位减立即数	2	12	2	6 倍
INC A	累加器加 1	1	12	1	12 倍
INC Rn	寄存器加 1	1	12	2	6 倍
INC direct	直接地址单元加 1	2	12	3	4 倍
INC @Ri	间接 RAM 单元加 1	1	12	3	4 倍
DEC A	累加器减 1	1	12	1	12 倍
DEC Rn	寄存器减 1	1	12	2	6 倍
DEC direct	直接地址单元减 1	2	12	3	4 倍
DEC @Ri	间接 RAM 单元减 1	1	12	3	4 倍
INC DPTR	地址寄存器 DPTR 加 1	1	24	1	24 倍
MUL AB	A 乘以 B	1	48	2	24 倍
DIV AB	A 除以 B	1	48	6	8 倍
DA A	累加器十进制调整	1	12	3	4 倍

逻辑操作类指令

助记符	功能说明	字节数	传统 8051 单片机所需时钟	STC15 系列单片机所需时钟(采用 STC-Y5 超高速 1T 8051 内核)	效率提升
ANL A, Rn	累加器与寄存器相“与”	1	12	1	12 倍
ANL A, direct	累加器与直接地址单元相 “与”	2	12	2	6 倍
ANL A, @Ri	累加器与间接 RAM 单元相“与”	1	12	2	6 倍
ANL A, #data	累加器与立即数相 “与”	2	12	2	6 倍
ANL direct, A	直接地址单元与累加器相“与”	2	12	3	4 倍
ANL direct, #data	直接地址单元与立即数相“与”	3	24	3	8 倍
ORL A, Rn	累加器与寄存器相“或”	1	12	1	12 倍
ORL A, direct	累加器与直接地址单元相 “或”	2	12	2	6 倍
ORL A, @Ri	累加器与间接 RAM 单元相“或”	1	12	2	6 倍
ORL A, # data	累加器与立即数相 “或”	2	12	2	6 倍
ORL direct, A	直接地址单元与累加器相“或”	2	12	3	4 倍
ORL direct, #data	直接地址单元与立即数相“或”	3	24	3	8 倍
XRL A, Rn	累加器与寄存器相“异或”	1	12	1	12 倍
XRL A, direct	累加器与直接地址单元相 “异或”	2	12	2	6 倍
XRL A, @Ri	累加器与间接 RAM 单元相“异或”	1	12	2	6 倍
XRL A, # data	累加器与立即数相 “异或”	2	12	2	6 倍
XRL direct, A	直接地址单元与累加器相“异或”	2	12	3	4 倍
XRL direct, #data	直接地址单元与立即数相“异或”	3	24	3	8 倍
CLR A	累加器清“ 0”	1	12	1	12 倍
CPL A	累加器求反	1	12	1	12 倍
RL A	累加器循环左移	1	12	1	12 倍
RLC A	累加器带进位位循环左移	1	12	1	12 倍
RR A	累加器循环右移	1	12	1	12 倍
RRC A	累加器带进位位循环右移	1	12	1	12 倍
SWAP A	累加器内高低半字节交换	1	12	1	12 倍

数据传送类指令

助记符	功能说明	字节数	传统 8051 单片机所需时钟	STC15 系列单片机所需时钟 (采用 STC-Y5 超高速 1T 8051 内核)	效率提升
MOV A, Rn	寄存器内容送入累加器	1	12	1	12 倍
MOV A, direct	直接地址单元中的数据送入累加器	2	12	2	6 倍
MOV A, @Ri	间接 RAM 中的数据送入累加器	1	12	2	6 倍
MOV A, #data	立即数送入累加器	2	12	2	6 倍
MOV Rn, A	累加器内容送入寄存器	1	12	1	12 倍
MOV Rn, direct	直接地址单元中的数据送入寄存器	2	24	3	8 倍
MOV Rn, #data	立即数送入寄存器	2	12	2	6 倍
MOV direct, A	累加器内容送入直接地址单元	2	12	2	6 倍
MOV direct, Rn	寄存器内容送入直接地址单元	2	24	2	12 倍
MOV direct, direct	直接地址单元中的数据送入另一个直接地址单元	3	24	3	8 倍
MOV direct, @Ri	间接 RAM 中的数据送入直接地址单元	2	24	3	8 倍
MOV direct, #data	立即数送入直接地址单元	3	24	3	8 倍
MOV @Ri, A	累加器内容送间接 RAM 单元	1	12	2	6 倍
MOV @Ri, direct	直接地址单元数据送入间接 RAM 单元	2	24	3	8 倍
MOV @Ri, #data	立即数送入间接 RAM 单元	2	12	2	6 倍
MOV DPTR,#data16	16 位立即数送入数据指针	3	24	3	8 倍
MOVC A, @A+DPTR	以 DPTR 为基地址变址寻址单元中的数据送入累加器	1	24	5	4.8 倍
MOVC A, @A+PC	以 PC 为基地址变址寻址单元中的数据送入累加器	1	24	4	6 倍
MOVX A, @Ri	将逻辑上在片外、物理上在片内的扩展 RAM(8 位地址)的内容送入累加器 A 中，读操作	1	24	3	8 倍
MOVX @Ri, A	将累加器 A 的内容送入逻辑上在片外、物理上在片内的扩展 RAM(8 位地址)中，写操作	1	24	4	8 倍
MOVX A, @DPTR	将逻辑上在片外、物理上在片内的扩展 RAM(16 位地址)的内容送入累加器 A 中，读操作	1	24	2	12 倍
MOVX @DPTR, A	将累加器 A 的内容送入逻辑上在片外、物理上在片内的扩展 RAM(16 位地址)中，写操作	1	24	3	8 倍
MOVX A, @Ri	将逻辑上在片外、物理上也在片外的扩展 RAM(8 位地址)的内容送入累加器 A 中，读操作	1	24	5×N+2 N 的取值见下列说明	*Note1
MOVX @Ri, A	将累加器 A 的内容送入逻辑上在片外、物理上也在片外的扩展 RAM(8 位地址)中，写操作	1	24	5×N+3 N 的取值见下列说明	*Note1
MOVX A, @DPTR	将逻辑上在片外、物理上也在片外的扩展 RAM(16 位地址)的内容送入累加器 A 中，读操作	1	24	5×N+1 N 的取值见下列说明	*Note1
MOVX @DPTR, A	将累加器 A 的内容送入逻辑上在片外、物理上也在片外的扩展 RAM(16 位地址)中，写操作	1	24	5×N+2 N 的取值见下列说明	*Note1
PUSH direct	直接地址单元中的数据压入堆栈	2	24	3	8 倍
POP direct	栈底数据弹出送入直接地址单元	2	24	2	12 倍
XCH A, Rn	寄存器与累加器交换	1	12	2	6 倍
XCH A,direct	直接地址单元与累加器交换	2	12	3	4 倍
XCH A, @Ri	间接 RAM 与累加器交换	1	12	3	4 倍
XCHD A, @Ri	间接 RAM 的低半字节与累加器交换	1	12	3	4 倍

当 EXRTS[1:0] = [0,0]时，表中 N=1

当 EXRTS[1:0] = [0,1]时，表中 N=2

当 EXRTS[1:0] = [1,0]时，表中 N=4

当 EXRTS[1:0] = [1,1]时，表中 N=8.　　EXRTS[1：0]为寄存器 BUS_SPEED 中的 B1，B0 位

布尔变量操作类指令

助记符	功能说明	字节数	传统 8051 单片机所需时钟	STC15 系列单片机所需时钟 (采用 STC-Y5 超高速 1T 8051 内核)	效率提升
CLR C	清零进位位	1	12	1	12 倍
CLR bit	清 0 直接地址位	2	12	3	4 倍
SETB C	置 1 进位位	1	12	1	12 倍
SETB bit	置 1 直接地址位	2	12	3	4 倍
CPL C	进位位求反	1	12	1	12 倍
CPL bit	直接地址位求反	2	12	3	4 倍
ANL C, bit	进位位和直接地址位相“与”	2	24	2	12 倍
ANL C, /bit	进位位和直接地址位的反码相“与”	2	24	2	12 倍
ORL C, bit	进位位和直接地址位相	2	24	2	12 倍
ORL C, /bit	进位位和直接地址位的反码	2	24	2	12 倍
MOV C, bit	直接地址位送入进位位	2	12	2	12 倍
MOV bit, C	进位位送入直接地址位	2	24	3	8 倍
JC rel	进位位为 1 则转移	2	24	3	8 倍
JNC rel	进位位为 0 则转移	2	24	3	8 倍
JB bit, rel	直接地址位为 1 则转移	3	24	5	4.8 倍
JNB bit, rel	直接地址位为 0 则转移	3	24	5	4.8 倍
JBC bit, rel	直接地址位为 1 则转移，该位清 0	3	24	5	4.8 倍

控制转移类指令

助记符	功能说明	字节数	传统 8051 单片机所需时钟	STC15 系列单片机所需时钟 (采用 STC-Y5 超高速 1T 8051 内核)	效率提升
ACALL addr11	绝对（短）调用子程序	2	24	4	6 倍
LCALL addr16	长调用子程序	3	24	4	6 倍
RET	子程序返回	1	24	4	6 倍
RETI	中断返回	1	24	4	6 倍
AJMP addr11	绝对（短）转移	2	24	3	8 倍
LJMP addr16	长转移	3	24	4	6 倍
SJMP re1	相对转移	2	24	3	8 倍
JMP @A+DPTR	相对于 DPTR 的间接转移	1	24	5	4.8 倍
JZ re1	累加器为零转移	2	24	4	6 倍
JNZ re1	累加器非零转移	2	24	4	6 倍
CJNE A, direct, re1	累加器与直接地址单元比较，不相等则转移	3	24	5	4.8 倍
CJNE A, #data, re1	累加器与立即数比较，不相等则转移	3	24	4	6 倍
CJNE Rn, #data, re1	寄存器与立即数比较，不相等则转移	3	24	4	6 倍
CJNE @Ri, #data, re1	间接 RAM 单元与立即数比较，不相等则转移	3	24	5	4.8 倍
DJNZ Rn, re1	寄存器减 1，非零转移	2	24	4	6 倍
DJNZ direct, re1	直接地址单元减 1，非零转移	3	24	5	4.8 倍
NOP	空操作	1	12	1	12 倍

16.3 传统 8051 单片机指令定义详解(中文&English)

16.3.1 传统 8051 单片机指令定义详解

ACALL addr11

功能： 绝对调用

说明： ACALL 指令实现无条件调用位于 addr11 参数所表示地址的子例程。在执行该指令时，首先将 PC 的值增加 2，即使得 PC 指向 ACALL 的下一条指令，然后把 16 位 PC 的低 8 位和高 8 位依次压入栈，同时把栈指针两次加 1。然后，把当前 PC 值的高 5 位、ACALL 指令第 1 字节的 7~5 位和第 2 字节组合起来，得到一个 16 位目的地址，该地址即为即将调用的子例程的入口地址。要求该子例程的起始地址必须与紧随 ACALL 之后的指令处于同 1 个 2KB 的程序存储页中。ACALL 指令在执行时不会改变各个标志位。

举例： SP 的初始值为 07H，标号 SUBRTN 位于程序存储器的 0345H 地址处，如果执行位于地址 0123H 处的指令：

ACALL SUBRTN

那么 SP 变为 09H，内部 RAM 地址 08H 和 09H 单元的内容分别为 25H 和 01H，PC 值变为 0345H。

指令长度(字节)： 2

执行周期： 3

二进制编码：

A10	A9	A8	1	0	0	0	1		A7	A6	A5	A4	A3	A2	A1	A0

注意：a10 a9 a8 是 11 位目标地址 addr11 的 A10~A8 位，a7 a6 a5 a4 a3 a2 a1 a0 是 addr11 的 A7~A0 位。

操作： ACALL

(PC)←(PC)+ 2

(SP)←(SP) + 1

((SP)) ←(PC_{7-0})

(SP)←(SP) + 1

((SP))←(PC_{15-8})

(PC_{10-0})←页码地址

ADD A, <src-byte>

功能： 加法

说明： ADD 指令可用于完成把 src-byte 所表示的源操作数和累加器 *A* 的当前值相加。并将结果置于累加器 *A* 中。根据运算结果，若第 7 位有进位则置进位标志为 1，否则清零；若第 3 位有进位则置辅助进位标志为 1，否则清零。如果是无符号整数相加则进位置位，显示当前运算结果发生溢出。如果第 6 位有进位生成而第 7 位没有，或第 7 位有进位生成而第 6 位没有，则置 OV 为 1，否则 OV 被清零。在进行有符号整数的相加运算的时候，OV 置位表示两个正整数之和为一负数，或是两个负整数之和为一正数。

本类指令的源操作数可接受 4 种寻址方式：寄存器寻址、直接寻址、寄存器间接寻址和立即寻址。

举例： 假设累加器 *A* 中的数据为 0C3H(11000011B)，R0 的值为 0AAH(10101010B)。执行如下指令：

ADD A, R0

累加器 *A* 中的结果为 6DH(01101101B)，辅助进位标志 AC 被清零，进位标志 C 和溢出标志 OV 被置 1。

ADD A, Rn

指令长度(字节)： 1

执行周期： 1

二进制编码：

0	0	1	0	1	r	r	r

操作： ADD

(A)← (A) + (Rn)

ADD A, direct

指令长度(字节)： 2

执行周期： 1

二进制编码：

0	0	1	0	0	1	0	1		direct address

操作： ADD

(A)← (A) + (direct)

ADD A, @Ri

指令长度(字节)： 1

执行周期： 1

二进制编码：

0	0	1	0	0	1	1	i

操作： ADD

(A)← (A) + ((Ri))

ADD A, #data

指令长度(字节)： 2

执行周期： 1

二进制编码：

0	0	1	0	0	1	0	0		immediate data

操作： ADD

(A)← (A) + #data

ADDC A, <src-byte>

功能： 带进位的加法

说明： 执行 ADDC 指令时，把 src-byte 所代表的源操作数连同进位标志一起加到累加器 *A* 上，并将结果置于累加器 *A* 中。根据运算结果，若在第 7 位有进位生成，则将进位标志置 1，否则清零；若在第 3 位有进位生成，则置辅助进位标志为 1，否则清零。如果是无符号数整数相加，进位的置位显示当前运算结果发生溢出。

如果第 6 位有进位生成而第 7 位没有，或第 7 位有进位生成而第 6 位没有，则将 OV 置 1，否则将 OV 清零。在进行有符号整数相加运算的时候，OV 置位，表示两个正整数之和为一负数，或是两个负整数之和为一正数。

本类指令的源操作数允许 4 种寻址方式：寄存器寻址、直接寻址、寄存器间接寻址和立即寻址。

举例： 假设累加器 *A* 中的数据为 0C3H(11000011B)，*R*0 的值为 0AAH(10101010B)，进位标志为 1，执行如下指令：

ADDC A,R0

累加器 *A* 中的结果为 6EH(01101110B)，辅助进位标志 AC 被清零，进位标志 C 和溢出标志 OV 被置 1。

ADDC A, Rn

指令长度(字节)： 1

执行周期： 1

二进制编码：

0	0	1	1	1	r	r	r

操作： ADDC

(A)← (A) + (C) + (Rn)

ADDC A, direct

指令长度(字节)： 2

执行周期： 1

二进制编码：

0	0	1	1	0	1	0	1		direct address

操作： ADDC

(A)← (A) + (C) + (direct)

ADDC A, @Ri

指令长度(字节)： 1

执行周期： 1

二进制编码：

0	0	1	1	0	1	1	i

操作： ADDC

(A)← (A) + (C) + ((Ri))

ADDC A, #data

指令长度(字节)： 2

执行周期： 1

二进制编码：

0	0	1	1	0	1	0	0		immediate data

操作： ADDC

(A)← (A) + (C) + #data

AJMP addr11

功能： 绝对跳转

说明： AJMP 指令用于将程序转到相应的目的地址去执行，该地址在程序执行过程之中产生，由 PC 值（两次递增之后）的高 5 位、操作码的 7~5 位和指令的第 2 字节连接形成。要求跳转的目的地址和 AJMP 指令的后一条指令的第 1 字节位于同一 2KB 的程序存储页内。

举例： 假设标号 JMPADR 位于程序存储器的 0123H，指令：

AJMP JMPADR

位于 0345H，执行完该指令后 PC 值变为 0123H。

指令长度(字节)： 2

执行周期： 3

二进制编码：

A10	A9	A8	0	0	0	0	1		A7	A6	A5	A4	A3	A2	A1	A0

注意：目的地址的 A10－A8＝a10~a8，A7－A0＝a7~a0。

操作： AJMP

(PC)←(PC)+ 2

($PC_{10\text{-}0}$)←page address

ANL <dest-byte> , <src-byte>

功能： 对字节变量进行逻辑与运算

说明： ANL 指令将由<dest-byte>和<src-byte>所指定的两个字节变量逐位进行逻辑与运算，并将运算结果存放在<dest-byte>所指定的目的操作数中。该指令的执行不会影响标志位。

两个操作数组合起来允许 6 种寻址模式。当目的操作数为累加器时，源操作数允许寄存器寻址、直接寻址、寄存器间接寻址和立即寻址。当目的操作数是直接地址时，源操作数可以是累加器

或立即数。

注意：当该指令用于修改输出端口时，读入的原始数据来自于输出数据的锁存器而非输入引脚。

举例： 如果累加器的内容为 0C3H(11000011B)，寄存器 0 的内容为 55H(01010101B)，那么指令:

ANL A,R0

执行结果是累加器的内容变为 41H(01000001H)。

当目的操作数是可直接寻址的数据时，ANL 指令可用来把任何 RAM 单元或者硬件寄存器中的某些位清零。屏蔽字节将决定哪些位将被清零。屏蔽字节可能是常数，也可能是累加器在计算过程中产生。如下指令:

ANL Pl, #01110011B

将端口 1 的位 7、位 3 和位 2 清零。

ANL A, Rn

指令长度(字节)： 1

执行周期： 1

二进制编码：

0	1	0	1	1	r	r	r

操作： ANL

(A)← (A) ∧ (Rn)

ANL A, direct

指令长度(字节)： 2

执行周期： 1

二进制编码：

0	1	0	1	0	1	0	1		direct address

操作： ANL

(A)← (A) ∧ (direct)

ANL A, @Ri

指令长度(字节)： 1

执行周期： 1

二进制编码：

0	1	0	1	0	1	1	i

操作： ANL

(A)← (A) ∧ ((Ri))

ANL A, #data

指令长度(字节)： 2

执行周期： 1

二进制编码：

0	1	0	1	0	1	0	0		immediate data

操作： ANL

(A)← (A) ∧ #data

ANL direct, A

指令长度(字节)： 2

执行周期： 1

二进制编码：

0	1	0	1	0	0	1	0		direct address

操作： ANL

(direct)←(direct) ∧ (A)

ANL direct, #data

指令长度(字节)： 3

执行周期： 1

二进制编码：

0	1	0	1	0	0	1	1		direct address		immediate data

操作： ANL

(direct)←(direct) ∧ #data

ANL C, <src-bit>

功能： 对位变量进行逻辑与运算

说明： 如果 src-bit 表示的布尔变量为逻辑 0，清零进位标志位；否则，保持进位标志的当前状态不变。在汇编语言程序中，操作数前面的“/”符号表示在计算时需要先对被寻址位取反，然后才作为源操作数，但源操作数本身不会改变。该指令在执行时不会影响其他各个标志位。

源操作数只能采取直接寻址方式。

举例： 下面的指令序列当且仅当 P1.0＝1、ACC.7＝1 和 OV＝0 时，将进位标志 C 置 1:

```
MOV C, P1.0          ; LOAD CARRY WITH INPUT PIN STATE
ANL C, ACC.7         ; AND CARRY WITH ACCUM. BIT.7
ANL C, /OV           ;AND WITH INVERSE OF OVERFLOW FLAG
```

ANL C, bit

指令长度(字节)： 2

执行周期： 1

二进制编码：

1	0	0	0	0	0	1	0		bit address

操作： ANL

(C) ←(C) ∧ (bit)

ANL C, /bit

指令长度(字节)： 2

执行周期： 1

二进制编码：

1	0	1	1	0	0	0	0		bit address

操作： ANL

(C)← (C) ∧($\overline{bit}$)

CJNE <dest-byte>, <src-byte>, rel

功能： 若两个操作数不相等则转移

说明： CJNE 首先比较两个操作数的大小，如果二者不等则程序转移。目标地址由位于 CJNE 指令最后 1 个字节的有符号偏移量和 PC 的当前值（紧邻 CJNE 的下一条指令的地址）相加而成。如果目标操作数作为一个无符号整数，其值小于源操作数对应的无符号整数，那么将进位标志置 1，否则将进位标志清零。但操作数本身不会受到影响。

<dest-byte>和<src-byte>组合起来，允许 4 种寻址模式。累加器 *A* 可以与任何可直接寻址的数据或立即数进行比较，任何间接寻址的 RAM 单元或当前工作寄存器都可以和立即常数进行比较。

举例： 设累加器 *A* 中值为 34H，*R7* 包含的数据为 56H。如下指令序列:

```
        CJNE R7,#60H, NOT_EQ
;       . . .     . . . . . .      ; R7 = 60H.
NOT_EQ: JC        REQ_LOW          ; IF R7 < 60H.
;       . . .     . . . . . .      ; R7 > 60H.
```

的第 1 条指令将进位标志置 1，程序跳转到标号 NOT_EQ 处。接下去，通过测试进位标志，可以确定 *R7* 是大于 60H 还是小于 60H。

假设端口 1 的数据也是 34H，那么如下指令：

WAIT: CJNE A,P1,WAIT

清除进位标志并继续往下执行，因为此时累加器的值也为 34H，即和 P1 口的数据相等。（如果 *P*l 端口的数据是其他的值，那么程序在此不停地循环，直到 *P*1 端口的数据变成 34H 为止。）

CJNE A, direct, rel

指令长度(字节)： 3

执行周期： 2 或 3

二进制编码：

1	0	1	1	0	1	0	1		direct address		rel. address

操作：

(PC) ←(PC) + 3

IF (A) < > *(direct)*

THEN

(PC) ←(PC) +*relative offset*

IF (A) < *(direct)*

THEN

(C) ← 1

ELSE

(C) ← 0

CJNE A, #data, rel

指令长度(字节)： 3

执行周期： 1 或 3

二进制编码：

1	0	1	1	0	1	0	0		immediate data		rel. address

操作：

(PC) ←(PC) + 3

IF (A) < > *(data)*

THEN

(PC) ←(PC) +*relative offset*

IF (A) < *(data)*

THEN

(C) ← 1

ELSE

(C) ← 0

CJNE Rn, #data, rel

指令长度(字节)： 3

执行周期： 2 或 3

二进制编码：

1	0	1	1	1	r	r	r		immediate data		rel. address

操作：

(PC) ←(PC) + 3

IF (Rn) < > *(data)*

THEN

(PC) ←(PC) +*relative offset*

IF (Rn) < *(data)*

THEN

(C) ← 1

ELSE

(C) ← 0

CJNE @Ri, #data, rel

指令长度(字节)： 3

执行周期： 2 或 3

二进制编码：

1	0	1	1	0	1	1	i		immediate data		rel. address

操作： (PC) ←(PC) + 3

IF (Ri) < > *(data)*

THEN

(PC) ←(PC) +*relative offset*

IF (Ri) < *(data)*

THEN

(C) ← 1

ELSE

(C) ← 0

CLR A

功能： 清除累加器

说明： 该指令用于将累加器 *A* 的所有位清零，不影响标志位。

举例： 假设累加器 *A* 的内容为 5CH(01011100B)，那么指令：

CLR A

执行后，累加器的值变为 00H(00000000B)。

指令长度(字节)： 1

执行周期： 1

二进制编码：

1	1	1	0	0	1	0	0

操作： CLR

(A)←0

CLR bit

功能： 清零指定的位

说明： 将 bit 所代表的位清零，没有标志位会受到影响。CLR 可用于进位标志 C 或者所有可直接寻址的位。

举例： 假设端口 1 的数据为 5DH(01011101B)，那么指令：

CLR P1.2

执行后，Pl 端口被设置为 59H(01011001B)。

CLR C

指令长度(字节)： 1

执行周期： 1

二进制编码：

1	1	0	0	0	0	1	1

操作： CLR

(C)←0

CLR bit

指令长度(字节)： 2

执行周期： 1

二进制编码：

1	1	0	0	0	0	1	0		bit address

操作： CLR

(bit)←0

CPL A

功能： 累加器 *A* 求反

说明： 将累加器 *A* 的每一位都取反，即原来为 1 的位变为 0，原来为 0 的位变为 1。该指令不影响标志位。

举例： 设累加器 *A* 的内容为 5CH(01011100B)，那么指令：

CPL A

执行后，累加器的内容变成 0A3H（10100011B）。

指令长度(字节)： 1

执行周期： 1

二进制编码：

1	1	1	1	0	1	0	0

操作： CPL

(A)← ($\bar{A}$)

CPL bit

功能： 将 bit 所表示的位求反

说明： 将 bit 变量所代表的位取反，即原来位为 1 的变为 0，原来为 0 的变为 1。没有标志位会受到影响。CPL 可用于进位标志 C 或者所有可直接寻址的位。

注意：如果该指令被用来修改输出端口的状态，那么 bit 所代表的数据是端口锁存器中的数据，而不是从引脚上输入的当前状态。

举例： 设 *P*1 端口的数据为 5BH(01011011B)，那么指令：

CPL P1.1

CPL P1.2

执行完后，*P*l 端口被设置为 5DH(01011101B)。

CPL C

指令长度(字节)： 1

执行周期： 1

二进制编码：

1	0	1	1	0	0	1	1

操作： CPL

(C)← ($\bar{C}$)

CPL bit

指令长度(字节)： 2

执行周期： 1

二进制编码：

1	0	1	1	0	0	1	0		bit address

操作： CPL

(bit)← ($\overline{bit}$)

DA A

功能： 在加法运算之后，对累加器 *A* 进行十进制调整

说明： DA 指令对累加器 *A* 中存放的由此前的加法运算产生的 8 位数据进行调整（ADD 或 ADDC 指令可以用来实现两个压缩 BCD 码的加法），生成两个 4 位的数字。

如果累加器的低 4 位（位 3~位 0）大于 9 (xxxx1010~xxxx 1111），或者加法运算后，辅助

进位标志 AC 为 1，那么 DA 指令将把 6 加到累加器上，以在低 4 位生成正确的 BCD 数字。若加 6 后，低 4 位向上有进位，且高 4 位都为 1，进位则会一直向前传递，以致最后进位标志被置 1；但在其他情况下进位标志并不会被清零，进位标志会保持原来的值。

如果进位标志为 1，或者高 4 位的值超过 9 (1010xxxx~1111xxxx)，那么 DA 指令将把 6 加到高 4 位，在高 4 位生成正确的 BCD 数字，但不清除标志位。若高 4 位有进位输出，则置进位标志为 1，否则，不改变进位标志。进位标志的状态指明了原来的两个 BCD 数据之和是否大于 99，因而 DA 指令使得 CPU 可以精确地进行十进制的加法运算。注意，OV 标志不会受影响。

DA 指令的以上操作在一个指令周期内完成。实际上，根据累加器 *A* 和机器状态字 PSW 中的不同内容，DA 把 00H、06H、60H、66H 加到累加器 *A* 上，从而实现十进制转换。

注意：如果前面没有进行加法运算，不能直接用 DA 指令把累加器 *A* 中的十六进制数据转换为 BCD 数，此外，如果先前执行的是减法运算，DA 指令也不会有所预期的效果。

举例： 如果累加器中的内容为 56H（01010110B），表示十进制数 56 的 BCD 码，寄存器 3 的内容为 67H（01100111B），表示十进制数 67 的 BCD 码。进位标志为 1，则指令：

```
ADDC A,R3
DA   A
```

先执行标准的补码二进制加法，累加器 *A* 的值变为 0BEH，进位标志和辅助进位标志被清零。

接着，DA 执行十进制调整，将累加器 *A* 的内容变为 24H（00100100B），表示十进制数 24 的 BCD 码，也就是 56、67 及进位标志之和的后两位数字。DA 指令会把进位标志置位 1，这表示在进行十进制加法时，发生了溢出。56、67 以及 1 的和为 124。

把 BCD 格式的变量加上 0lH 或 99H，可以实现加 1 或者减 1。假设累加器的初始值为 30H（表示十进制数 30），指令序列：

```
ADD A, #99H
DA   A
```

将把进位 C 置为 1，累加器 *A* 的数据变为 29H，因为 30＋99＝129。加法和的低位数据可以看作减法运算的结果，即 30－1＝29。

指令长度（字节）： 1

执行周期： 3

二进制编码：

1	1	0	1	0	1	0	0

操作： DA

-contents of Accumulator are BCD

IF [[(A_{3-0}) > 9] V [(AC) = 1]]

THEN(A_{3-0}) ←(A_{3-0}) + 6

AND

IF [[(A_{7-4}) > 9] V [(C) = 1]]

THEN (A_{7-4})←(A_{7-4}) + 6

DEC byte

功能： 把 BYTE 所代表的操作数减 1

说明： BYTE 所代表的变量被减去 l。如果原来的值为 00H，那么减去 1 后，变成 0FFH。没有标志位会受到影响。该指令支持 4 种操作数寻址方式：累加器寻址、寄存器寻址、直接寻址和寄存器间接寻址。

注意：当 DEC 指令用于修改输出端口的状态时，BYTE 所代表的数据是从端口输出数据锁

存器中获取的，而不是从引脚上读取的输入状态。

举例： 假设寄存器 0 的内容为 7FH（01111111B），内部 RAM 的 7EH 和 7FH 单元的内容分别为 00H 和 40H。则指令：

```
DEC @R0
DEC R0
DEC @R0
```

执行后，寄存器 0 的内容变成 7EH，内部 RAM 的 7EH 和 7FH 单元的内容分别变为 0FFH 和 3FH。

DEC A

指令长度(字节)： 1

执行周期： 1

二进制编码：

0	0	0	1	0	1	0	0

操作： DEC

(A)← (A) −1

DEC Rn

指令长度(字节)： 1

执行周期： 1

二进制编码：

0	0	0	1	1	r	r	r

操作： DEC

(Rn)←(Rn) - 1

DEC direct

指令长度(字节)： 2

执行周期： 1

二进制编码：

0	0	0	1	0	1	0	1		Direct address

操作： DEC

(direct)←(direct) - 1

DEC @Ri

指令长度(字节)： 1

执行周期： 1

二进制编码：

0	0	0	1	0	1	1	i

操作： DEC

((Ri))←((Ri)) - 1

DIV AB

功能： 除法

说明： DIV 指令把累加器 *A* 中的 8 位无符号整数除以寄存器 *B* 中的 8 位无符号整数，并将商置于累加器 *A* 中，余数置于寄存器 *B* 中。进位标志 C 和溢出标志 OV 被清零。

例外：如果寄存器 *B* 的初始值为 00H（即除数为 0），那么执行 DIV 指令后，累加器 *A* 和寄存器 *B* 中的值是不确定的，且溢出标志 OV 将被置位。但在任何情况下，进位标志 C 都会被清零。

举例： 假设累加器的值为 251 (0FBH 或 11111011B)，寄存器 *B* 的值为 18 (12H 或 00010010B)。则指令：

```
DIV AB
```

执行后，累加器的值变成 13 (0DH 或 00001101B)，寄存器 *B* 的值变成 17 (11H 或 00010001B)，

正好符合 251＝13×18＋17。进位和溢出标志都被清零。

指令长度(字节)： 1

执行周期： 6

二进制编码：

1	0	0	0	0	1	0	0

操作： DIV

$(A)_{15-8}(B)_{7-0} \leftarrow (A)/(B)$

DJNZ <byte>, <rel-addr>

功能： 减 1，若非 0 则跳转

说明： DJNZ 指令首先将第 1 个操作数所代表的变量减 1，如果结果不为 0，则转移到第 2 个操作数所指定的地址处去执行。如果第 1 个操作数的值为 00H，则减 1 后变为 0FFH。该指令不影响标志位。跳转目标地址的计算：首先将 PC 值加 2（即指向下一条指令的首字节），然后将第 2 操作数表示的有符号的相对偏移量加到 PC 上去即可。

byte 所代表的操作数可采用寄存器寻址或直接寻址。

注意：如果该指令被用来修改输出引脚上的状态，那么 byte 所代表的数据是从端口输出数据锁存器中获取的，而不是直接读取引脚。

举例： 假设内部 RAM 的 40H、50H 和 60H 单元分别存放着 01H、70H 和 15H，则指令：

```
DJNZ 40H, LABEL_1
DJNZ 50H, LABEL_2
DJNZ 60H, LABEL_3
```

执行之后，程序将跳转到标号 LABEL2 处执行，且相应的 3 个 RAM 单元的内容变成 00H、6FH 和 15H。之所以第 1 个跳转没被执行，是因为减 1 后其结果为 0，不满足跳转条件。

使用 DJNZ 指令可以方便地在程序中实现指定次数的循环，此外用一条指令就可以在程序中实现中等长度的时间延迟（2~512 个机器周期）。指令序列：

```
        MOV R2,#8
TOOOLE: CPL P1.7
        DJNZ R2, TOOGLE
```

将使得 P1.7 的电平翻转 8 次，从而在 P1.7 产生 4 个脉冲，每个脉冲将持续 3 个机器周期，其中 2 个为 DJNZ 指令的执行时间，1 个为 CPL 指令的执行时间。

DJNZ Rn, rel

指令长度(字节)： 2

执行周期： 2 或 3

二进制编码：

1	1	0	1	1	r	r	r		rel. address

操作： DJNZ

(PC) ←(PC) + 2

(Rn) ←(Rn) – 1

IF (Rn) > 0 or (Rn) < 0

THEN

(PC) ←(PC)+ rel

DJNZ direct, rel

指令长度(字节)： 3

执行周期： 2 或 3

二进制编码：

1	1	0	1	0	1	0	1		direct address		rel. address

操作： DJNZ

(PC) ←(PC) + 2
(direct) ←(direct) – 1
IF (direct) > 0 or (direct) < 0
THEN
(PC) ←(PC)+ rel

INC <byte>

功能： 加 1

说明： INC 指令将<byte>所代表的数据加 1。如果原来的值为 FFH，则加 1 后变为 00H，该指令不影响标志位。支持 3 种寻址模式：寄存器寻址、直接寻址、寄存器间接寻址。

注意：如果该指令被用来修改输出引脚上的状态，那么 byte 所代表的数据是从端口输出数据锁存器中获取的，而不是直接读的引脚。

举例： 假设寄存器 0 的内容为 7EH(01111110B)，内部 RAM 的 7E 单元和 7F 单元分别存放着 0FFH 和 40H，则指令序列：

```
INC @R0
INC R0
INC @R0
```

执行完毕后，寄存器 0 的内容变为 7FH，而内部 RAM 的 7EH 和 7FH 单元的内容分别变成 00H 和 41H。

INC A

指令长度(字节)： 1
执行周期： 1
二进制编码：

0	0	0	0	0	1	0	0

操作： INC
(A) ←(A)+1

INC Rn

指令长度(字节)： 1
执行周期： 1
二进制编码：

0	0	0	0	1	r	r	r

操作： INC
(Rn) ←(Rn)+1

INC direct

指令长度(字节)： 2
执行周期： 1
二进制编码：

0	0	0	0	0	1	0	1		direct address

操作： INC
(direct) ←(direct)+1

INC @Ri

指令长度(字节)： 1
执行周期： 1
二进制编码：

0	0	0	0	0	1	1	i

操作： INC
((Ri))←((Ri)) + 1

INC DPTR

功能： 数据指针加 1

说明： 该指令实现将 DPTR 加 1 功能。需要注意的是，这是 16 位的递增指令，低位字节 DPL 从 FFH 增加 1 之后变为 00H，同时进位到高位字节 DPH。该操作不影响标志位。

该指令是唯一 1 条 16 位寄存器递增指令。

举例： 假设寄存器 DPH 和 DPL 的内容分别为 12H 和 0FEH，则指令序列：

IINC DPTR

INC DPTR

INC DPTR

执行完毕后，DPH 和 DPL 变成 13H 和 01H。

指令长度(字节)： 1

执行周期： 1

二进制编码：

1	0	1	0	0	0	1	1

操作： INC

(DPTR) ← (DPTR) + 1

JB bit, rel

功能： 若位数据为 1 则跳转

说明： 如果 bit 代表的位数据为 1，则跳转到 rel 所指定的地址处去执行；否则，继续执行下一条指令。跳转的目标地址按照如下方式计算：先增加 PC 的值，使其指向下一条指令的首字节地址，然后把 rel 所代表的有符号的相对偏移量（指令的第 3 个字节）加到 PC 上去，新的 PC 值即为目标地址。该指令只是测试相应的位数据，但不会改变其数值，而且该操作不会影响标志位。

举例： 假设端口 1 的输入数据为 11001010B，累加器的值为 56H（01010110B）。则指令：

JB P1.2, LABEL1

JB ACC.2, LABEL2

将导致程序转到标号 LABEL2 处去执行。

指令长度(字节)： 3

执行周期： 1 或 3

二进制编码：

0	0	1	0	0	0	0	0		bit address		rel. address

操作： JB

(PC) ← (PC) + 3

IF (bit) = 1

THEN

(PC) ← (PC)+ rel

JBC bit, rel

功能： 若位数据为 1 则跳转并将其清零

说明： 如果 bit 代表的位数据为 1，则将其清零并跳转到 rel 所指定的地址处去执行。如果 bit 代表的位数据为 0，则继续执行下一条指令。跳转的目标地址按照如下方式计算：先增加 PC 的值，使其指向下一条指令的首字节地址，然后把 rel 所代表的有符号的相对偏移量（指令的第 3 个字节）加到 PC 上去，新的 PC 值即为目标地址，而且该操作不会影响标志位。

注意：如果该指令被用来修改输出引脚上的状态，那么 byte 所代表的数据是从端口输出数据锁存器中获取的，而不是直接读取引脚。

举例： 假设累加器的内容为 56H(01010110B)，则指令序列：

```
JBC ACC.3, LABEL1
JBC ACC.2, LABEL2
```

将导致程序转到标号 LABEL2 处去执行，且累加器的内容变为 52H（01010010B）。

指令长度(字节)： 3

执行周期： 1 或 3

二进制编码：

0	0	0	1	0	0	0	0		bit address		rel. address

操作：

```
JB
(PC) ← (PC) + 3
IF (bit) = 1
    THEN
        (bit) ← 0
        (PC) ← (PC)+ rel
```

JC rel

功能： 若进位标志为 1，则跳转

说明： 如果进位标志为 1，则程序跳转到 rel 所代表的地址处去执行；否则，继续执行下面的指令。跳转的目标地址按照如下方式计算：先增加 PC 的值，使其指向紧接 JC 指令的下一条指令的首地址，然后把 rel 所代表的有符号的相对偏移量（指令的第 2 个字节）加到 PC 上去，新的 PC 值即为目标地址。该操作不会影响标志位。

举例： 假设进位标志此时为 0，则指令序列：

```
JC LABEL1
CPL C
JC LABEL2
```

执行完毕后，进位标志变成 1，并导致程序跳转到标号 LABEL2 处去执行。

指令长度(字节)： 2

执行周期： 1 或 3

二进制编码：

0	1	0	0	0	0	0	0		rel. address

操作：

```
JC
(PC) ← (PC) + 2
IF (C) = 1
    THEN
        (PC) ← (PC)+ rel
```

JMP @A+DPTR

功能： 间接跳转

说明： 把累加器 *A* 中的 8 位无符号数据和 16 位的数据指针的值相加，其和作为下一条将要执行的指令的地址，传送给程序计数器 PC。执行 16 位的加法时，低字节 DPL 的进位会传到高字节 DPH。累加器 *A* 和数据指针 DPTR 的内容都不会发生变化。不影响任何标志位。

举例： 假设累加器 *A* 中的值是偶数（从 0 到 6）。下面的指令序列将使得程序跳转到位于跳转表 JMP_TBL 的 4 条 AJMP 指令中的某一条去执行：

```
        MOV DPTR, #JMP_TBL
        JMP @A+DPTR
JMP-TBL: AJMP LABEL0
        AJMP LABEL1
        AJMP LABEL2
        AJMP LABEL3
```

如果开始执行上述指令序列时，累加器 *A* 中的值为 04H，那么程序最终会跳转到标号 LABEL2 处去执行。

注意：AJMP 是一个 2 字节指令，因而在跳转表中，各个跳转指令的入口地址依次相差 2 个字节。

指令长度(字节)： 1

执行周期： 4

二进制编码：

0	1	1	1	0	0	1	1

操作： JMP

(PC) ← (A) + (DPTR)

JNB bit, rel

功能： 如果 bit 所代表的位不为 1 则跳转

说明： 如果 bit 所表示的位为 0，则转移到 rel 所代表的地址去执行；否则，继续执行下一条指令。跳转的目标地址如此计算：先增加 PC 的值，使其指向下一条指令的首字节地址，然后把 rel 所代表的有符号的相对偏移量（指令的第 3 个字节）加到 PC 上去，新的 PC 值即为目标地址。该指令只是测试相应的位数据，但不会改变其数值，而且该操作不会影响标志位。

举例： 假设端口 1 的输入数据为 11001010B，累加器的值为 56H（01010110B）。则指令序列：

```
JNB P1.3, LABEL1
JNB ACC.3, LABEL2
```

执行后将导致程序转到标号 LABEL2 处去执行。

指令长度(字节)： 3

执行周期： 1 或 3

二进制编码：

0	0	1	1	0	0	0	0		bit address		rel. address

操作： JNB

(PC) ←(PC) + 3

IF (bit) = 0

  THEN (PC) ← (PC) + rel

JNC rel

功能： 若进位标志非 1 则跳转

说明： 如果进位标志为 0，则程序跳转到 rel 所代表的地址处去执行；否则，继续执行下面的指令。跳转的目标地址按照如下方式计算：先增加 PC 的值加 2，使其指向紧接 JNC 指令的下一条指令的地址，然后把 rel 所代表的有符号的相对偏移量（指令的第 2 个字节）加到 PC 上去，新的 PC 值即为目标地址。该操作不会影响标志位。

举例： 假设进位标志此时为 1，则指令序列：

```
JNC LABEL1
CPL C
```

JNC LABEL2

执行完毕后，进位标志变成 0，并导致程序跳转到标号 LABEL2 处去执行。

指令长度(字节)： 2

执行周期： 1 或 3

二进制编码：

0	1	0	1	0	0	0	0		rel. address

操作： JNC

(PC) ←(PC) + 2

IF (C) = 0

THEN (PC) ← (PC) + rel

JNZ rel

功能： 如果累加器的内容非 0 则跳转

说明： 如果累加器 A 的任何一位为 1，那么程序跳转到 rel 所代表的地址处去执行，如果各个位都为 0，继续执行下一条指令。跳转的目标地址按照如下方式计算：先把 PC 的值增加 2，然后把 rel 所代表的有符号的相对偏移量（指令的第 2 个字节）加到 PC 上去，新的 PC 值即为目标地址。操作过程中累加器的值不会发生变化，不会影响标志位。

举例： 设累加器的初始值为 00H，则指令序列：

JNZ LABEL1

INC A

JNZ LAEEL2

执行完毕后，累加器的内容变成 01H，且程序将跳转到标号 LABEL2 处去执行。

指令长度(字节)： 2

执行周期： 1 或 3

二进制编码：

0	1	1	1	0	0	0	0		rel. address

操作： JNZ

(PC) ←(PC) + 2

IF (A) ≠ 0

THEN (PC) ← (PC) + rel

JZ rel

功能： 若累加器的内容为 0 则跳转

说明： 如果累加器 A 的任何一位为 0，那么程序跳转到 rel 所代表的地址处去执行，如果各个位都为 0，继续执行下一条指令。跳转的目标地址按照如下方式计算：先把 PC 的值增加 2，然后把 rel 所代表的有符号的相对偏移量（指令的第 2 个字节）加到 PC 上去，新的 PC 值即为目标地址。操作过程中累加器的值不会发生变化，不会影响标志位。

举例： 设累加器的初始值为 01H，则指令序列：

JZ LABEL1

DEC A

JZ LAEEL2

执行完毕后，累加器的内容变成 00H，且程序将跳转到标号 LABEL2 处去执行。

指令长度(字节)： 2

执行周期： 1 或 3

二进制编码：

0	1	1	0	0	0	0	0		rel. address

操作： JZ

(PC) ←(PC) + 2

IF (A) = 0

THEN (PC) ← (PC) + rel

LCALL addr16

功能： 长调用

说明： LCALL 用于调用 addr16 所指地址处的子例程。首先将 PC 的值增加 3，使得 PC 指向紧随 LCALL 的下一条指令的地址，然后把 16 位 PC 的低 8 位和高 8 位依次压入栈（低位字节在先），同时把栈指针加 2。然后再把 LCALL 指令的第 2 字节和第 3 字节的数据分别装入 PC 的高位字节 DPH 和低位字节 DPL，程序从新的 PC 所对应的地址处开始执行。因而子例程可以位于 64KB 程序存储空间的任何地址处。该操作不影响标志位。

举例： 栈指针的初始值为 07H，标号 SUBRTN 被分配的程序存储器地址为 1234H。则执行如下位于地址 0123H 的指令后：

LCALL SUBRTN

栈指针变成 09H，内部 RAM 的 08H 和 09H 单元的内容分别为 26H 和 01H，且 PC 的当前值为 1234H。

指令长度(字节)： 3

执行周期： 3

二进制编码：

0	0	0	1	0	0	1	0		addr15-addr8		addr7-addr0

操作： LCALL

(PC) ←(PC) + 3

(SP) ←(SP) + 1

((SP)) ←(PC_{7-0})

(SP) ←(SP) + 1

((SP)) ←(PC_{15-8})

(PC) ← $addr_{15-0}$

LJMP addr16

功能： 长跳转

说明： LJMP 使得 CPU 无条件跳转到 addr16 所指的地址处执行程序。把该指令的第 2 字节和第 3 字节分别装入程序计数器 PC 的高位字节 DPH 和低位字节 DPL。程序从新 PC 值对应的地址处开始执行。该 16 位目标地址可位于 64KB 程序存储空间的任何地址处。该操作不影响标志位。

举例： 假设标号 JMPADR 被分配的程序存储器地址为 1234H。则位于地址 1234H 的指令：

LJMP JMPADR

执行完毕后，PC 的当前值变为 1234H。

指令长度(字节)： 3

执行周期： 3

二进制编码：

0	0	0	0	0	0	1	0		addr15-addr8		addr7-addr0

操作： LJMP

(PC) ← $addr_{15-0}$

MOV <dest-byte> , <src-byte>

功能： 传送字节变量

说明： 将第 2 操作数代表字节变量的内容复制到第 1 操作数所代表的存储单元中去。该指令不会改变源操作数，也不会影响其他寄存器和标志位。

MOV 指令是迄今为止使用最灵活的指令，源操作数和目的操作数组合起来，寻址方式可达 15 种。

举例： 假设内部 RAM 的 30H 单元的内容为 40H，而 40H 单元的内容为 10H。端口 1 的数据为 11001010B（0CAH）。则指令序列：

```
MOV R0, #30H        ; R0< = 30H
MOV A, @R0          ; A < = 40H
MOV R1, A           ; R1 < = 40H
MOV B, @Rl          ; B < = 10H
MOV @Rl, Pl         ; RAM (40H) < = 0CAH
MOV P2, P1          ; P2 #0CAH
```

执行完毕后，寄存器 0 的内容为 30H，累加器和寄存器 1 的内容都为 40H，寄存器 B 的内容为 10H，RAM 中 40H 单元和 P2 口的内容均为 0CAH。

MOV A,Rn

指令长度（字节）： 1

执行周期： 1

二进制编码：

1	1	1	0	1	r	r	r

操作： MOV

(A) ←(Rn)

***MOV A,direct**

指令长度（字节）： 2

执行周期： 1

二进制编码：

1	1	1	0	0	1	0	1		direct address

操作： MOV

(A)← (direct)

注意：**MOV A, ACC** 是无效指令。

MOV A,@Ri

指令长度（字节）： 1

执行周期： 1

二进制编码：

1	1	1	0	0	1	1	i

操作： MOV

(A) ←((Ri))

MOV A,#data

指令长度（字节）： 2

执行周期： 1

二进制编码：

0	1	1	1	0	1	0	0		immediate data

操作： MOV

(A)←#data

MOV Rn, A

指令长度（字节）： 1

执行周期： 1

二进制编码：

1	1	1	1	1	r	r	r

操作： MOV

(Rn) ←(A)

MOV Rn,direct

指令长度(字节)： 2

执行周期： 1

二进制编码：

1	0	1	0	1	r	r	r		direct address

操作： MOV

(Rn)← (direct)

MOV Rn,#data

指令长度(字节)： 2

执行周期： 1

二进制编码：

0	1	1	1	1	r	r	r		immediate data

操作： MOV

(Rn)←#data

MOV direct, A

指令长度(字节)： 2

执行周期： 1

二进制编码：

1	1	1	1	0	1	0	1		direct address

操作： MOV

(direct)← (A)

MOV direct, Rn

指令长度(字节)： 2

执行周期： 2

二进制编码：

1	0	0	0	1	r	r	r		direct address

操作： MOV

(direct)← (Rn)

MOV direct, direct

指令长度(字节)： 3

执行周期： 1

二进制编码：

1	0	0	0	0	1	0	1		dir.addr. (src)		dir.addr. (dest)

操作： MOV

(direct)← (direct)

MOV direct, @Ri

指令长度(字节)： 2

执行周期： 1

二进制编码：

1	0	0	0	0	1	1	i		direct address

操作： MOV

(direct)←((Ri))

MOV direct, #data

指令长度(字节)： 3

执行周期： 1

二进制编码：

0	1	1	1	0	1	0	1		direct address		immediate data

操作： MOV

(direct)←#data

MOV @Ri, A

指令长度(字节)： 1

执行周期： 1

二进制编码：

1	1	1	1	0	1	1	i

操作： MOV

((Ri))←(A)

MOV @Ri, direct

指令长度(字节)： 2

执行周期： 1

二进制编码：

1	0	1	0	0	1	1	i		direct address

操作： MOV

((Ri))←(direct)

MOV @Ri, #data

指令长度(字节)： 2

执行周期： 1

二进制编码：

0	1	1	1	0	1	1	i		immediate data

操作： MOV

((Ri))←#data

MOV <dest-bit> , <src-bit>

功能： 传送位变量

说明： 将<src-bit>代表的布尔变量复制到<dest-bit>所指定的数据单元中去，两个操作数必须有一个是进位标志，而另外一个是可直接寻址的位。本指令不影响其他寄存器和标志位。

举例： 假设进位标志 C 的初值为 1，端口 P3 中的数据是 11000101B，端口 1 的数据被设置为 35H(00110101B)。则指令序列：

MOV P1.3, C

MOV C, P3.3

MOV P1.2, C

执行后，进位标志被清零，端口 1 的数据变为 39H（00111001B）。

MOV C, bit

指令长度(字节)： 2

执行周期： 1

二进制编码：

1	0	1	0	0	0	1	0		bit address

操作： MOV

(C)← (bit)

MOV bit, C

指令长度(字节)： 2

执行周期： 1

二进制编码：

1	0	0	1	0	0	1	0		bit address

操作： MOV

(bit)← (C)

MOV DPTR, #data 16

功能： 将 16 位的常数存放到数据指针

说明： 该指令将 16 位常数传递给数据指针 DPTR。16 位的常数包含在指令的第 2 字节和第 3 字节中。其中 DPH 中存放的是#data16 的高字节，而 DPL 中存放的是#data16 的低字节。不影响标志位。

该指令是唯一一条能一次性移动 16 位数据的指令。

举例： 指令：

MOV DPTR, #1234H

将立即数 1234H 装入数据指针寄存器中。DPH 的值为 12H，DPL 的值为 34H。

指令长度(字节)： 3

执行周期： 1

二进制编码：

1	0	0	1	0	0	0	0		immediate data15-8		immediate data7-0

操作： MOV

(DPTR) ← #data$_{15-0}$

DPH DPL ← #data$_{15-8}$ #data$_{7-0}$

MOVC A , @A+ <base-reg>

功能： 把程序存储器中的代码字节数据（常数数据）转送至累加器 *A*

说明： MOVC 指令将程序存储器中的代码字节或常数字节传送到累加器 *A*。被传送的数据字节的地址是由累加器中的无符号 8 位数据和 16 位基址寄存器（DPTR 或 PC）的数值相加产生的。如果以 PC 为基址寄存器，则在累加器内容加到 PC 之前，PC 需要先增加到指向紧邻 MOVC 之后的语句的地址；如果是以 DPTR 为基址寄存器，则没有此问题。在执行 16 位的加法时，低 8 位产生的进位会传递给高 8 位。本指令不影响标志位。

举例： 假设累加器 *A* 的值处于 0~4 之间，如下子例程将累加器 *A* 中的值转换为用 DB 伪指令（定义字节）定义的 4 个值之一：

```
REL-PC: INC A
        MOVC A, @A+PC
        RET
        DB 66H
        DB 77H
        DB 88H
        DB 99H
```

如果在调用该子例程之前累加器的值为 01H，执行完该子例程后，累加器的值变为 77H。

MOVC 指令之前的 INC A 指令是为了在查表时越过 RET 而设置的。如果 MOVC 和表格之间被多个代码字节所隔开，那么为了正确地读取表格，必须将相应的字节数预先加到累加器 *A* 上。

MOVC A,@A+DPTR

指令长度(字节)： 1

执行周期： 4

二进制编码：

1	0	0	1	0	0	1	1

操作： MOVC

(A) ←((A)+(DPTR))

MOVC A,@A+PC

指令长度(字节)： 1

执行周期： 3

二进制编码：

1	0	0	0	0	0	1	1

操作： MOVC

(PC) ←(PC)+1

(A) ←((A)+(PC))

MOVX <dest-byte> , <src-byte>

功能： 外部传送

说明： MOVX 指令用于在累加器和外部数据存储器之间传递数据。因此在传送指令 MOV 后附加了 X。MOVX 又分为两种类型，它们之间的区别在于访问外部数据 RAM 的间接地址是 8 位的还是 16 位的。

对于第 1 种类型，当前工作寄存器组的 R0 和 R1 提供 8 位地址到复用端口 P0。对于外部 I/O 扩展译码或者较小的 RAM 阵列，8 位的地址已经够用。若要访问较大的 RAM 阵列，可在端口引脚上输出高位的地址信号。此时可在 MOVX 指令之前添加输出指令，对这些端口引脚施加控制。

对于第 2 种类型，通过数据指针 DPTR 产生 16 位的地址。当 P2 端口的输出缓冲器发送 DPH 的内容时，P2 的特殊功能寄存器保持原来的数据。在访问规模较大的数据阵列时，这种方式更为有效和快捷，因为不需要额外指令来配置输出端口。

在某些情况下，可以混合使用两种类型的 MOVX 指令。在访问大容量的 RAM 空间时，既可以用数据指针 DP 在 P2 端口上输出地址的高位字节，也可以先用某条指令，把地址的高位字节从 P2 端口上输出，再使用通过 R0 或 Rl 间址寻址的 MOVX 指令。

举例： 假设有一个分时复用地址/数据线的外部 RAM 存储器，容量为 256B（如：Intel 的 8155 RAM / I/O / TIMER），该存储器被连接到 8051 的端口 P0 上，端口 P3 被用于提供外部 RAM 所需的控制信号。端口 Pl 和 P2 用作通用输入/输出端口。R0 和 Rl 中的数据分别为 12H 和 34H，外部 RAM 的 34H 单元存储的数据为 56H，则下面的指令序列：

MOVX A, @R1

MOVX @R0, A

将数据 56H 复制到累加器 A 以及外部 RAM 的 12H 单元中。

MOVX A,@Ri

指令长度（字节）： 1

执行周期： 3 或 1

二进制编码：

1	1	1	0	0	0	1	i

操作： MOVX

(A) ← ((Ri))

MOVX A,@DPTR

指令长度（字节）： 1

执行周期： 1 或 2

二进制编码：

1	1	1	0	0	0	0	0

操作： MOVX

(A) ←((DPTR))

MOVX @Ri, A

指令长度（字节）： 1

执行周期： 3 或 1

二进制编码：

1	1	1	1	0	0	1	i

操作： MOVX

((Ri))←(A)

MOVX @DPTR, A

指令长度(字节)： 1

执行周期： 2 或 1

二进制编码：

1	1	1	1	0	0	0	0

操作： MOVX

(DPTR)←(A)

MUL AB

功能： 乘法

说明： 该指令可用于实现累加器和寄存器 B 中的无符号 8 位整数的乘法。所产生的 16 位乘积的低 8 位存放在累加器中，而高 8 位存放在寄存器 B 中。若乘积大于 255(0FFH)，则置位溢出标志；否则清零标志位。在执行该指令时，进位标志总是被清零。

举例： 假设累加器 A 的初始值为 80(50H)，寄存器 B 的初始值为 160 (0A0H)，则指令：

MUL AB

求得乘积 12800 (3200H)，所以寄存器 B 的值变成 32H (00110010B)，累加器被清零，溢出标志被置位，进位标志被清零。

指令长度(字节)： 1

执行周期： 2

二进制编码：

1	0	1	0	0	1	0	0

操作： $(A)_{7\text{-}0} \leftarrow (A)\times(B)$

$(B)_{15\text{-}8}$

NOP

功能： 空操作

说明： 执行本指令后，将继续执行随后的指令。除了 PC 外，其他寄存器和标志位都不会有变化。

举例： 假设期望在端口 P2 的第 7 号引脚上输出一个长时间的低电平脉冲，该脉冲持续 5 个机器周期（精确）。若是仅使用 SETB 和 CLR 指令序列，生成的脉冲只能持续 1 个机器周期。因而需要设法增加 4 个额外的机器周期。可以按照如下方式来实现所要求的功能（假设中断没有被启用）：

```
CLR P2.7
NOP
NOP
NOP
NOP
SETB P2.7
```

指令长度(字节)： 1

执行周期： 1

二进制编码：

0	0	0	0	0	0	0	0

操作： NOP

(PC)←(PC)+1

ORL <dest-byte> , <src-byte>

功能： 两个字节变量的逻辑或运算

说明： ORL 指令将由<dest-byte>和<src_byte>所指定的两个字节变量进行逐位逻辑或运算，结果存放在<dest-byte>所代表的数据单元中。该操作不影响标志位。

两个操作数组合起来，支持 6 种寻址方式。当目的操作数是累加器 *A* 时，源操作数可以采用寄存器寻址、直接寻址、寄存器间接寻址或者立即寻址。当目的操作数采用直接寻址方式时，源操作数可以是累加器或立即数。

注意：如果该指令被用来修改输出引脚上的状态，那么<dest-byte>所代表的数据是从端口输出数据锁存器中获取的数据，而不是从引脚上读取的数据。

举例： 假设累加器 *A* 中数据为 0C3H (11000011B)，寄存器 *R0* 中的数据为 55H(01010101) ，则指令：

ORL A, R0

执行后，累加器的内容变成 0D7H(11010111B)。当目的操作数是直接寻址数据字节时，ORL 指令可用来把任何 RAM 单元或者硬件寄存器中的各个位设置为 1。究竟哪些位会被置 1 由屏蔽字节决定，屏蔽字节既可以是包含在指令中的常数，也可以是累加器 *A* 在运行过程中实时计算出的数值。执行指令：

ORL P1, #00110010B

之后，把 1 口的第 5、4、1 位置 1。

ORL A, Rn

指令长度（字节）： 1

执行周期： 1

二进制编码：

0	1	0	0	1	r	r	r

操作： ORL

(A) ←(A)∨(Rn)

ORL A, direct

指令长度（字节）： 2

执行周期： 1

二进制编码：

0	1	0	0	0	1	0	1		direct address

操作： ORL

(A) ←(A)∨(direct)

ORL A, @Ri

指令长度（字节）： 1

执行周期： 1

二进制编码：

0	1	0	0	0	1	1	i

操作： ORL

(A) ←(A)∨((Ri))

ORL A, #data

指令长度（字节）： 2

执行周期： 1

二进制编码：

0	1	0	0	0	1	0	0		immediate data

操作： ORL

(A) ←(A)∨#data

ORL direct, A

指令长度（字节）： 2

执行周期： 1

二进制编码：	0	1	0	0	0	0	1	0		direct address

操作： ORL

(direct)← (direct)∨(A)

ORL direct, #data

指令长度(字节)： 3

执行周期： 1

二进制编码：	0	1	0	0	0	0	1	1		direct address		immediate data

操作： ORL

(direct)← (direct)∨#data

ORL C, <src-bit>

功能： 位变量的逻辑或运算

说明： 如果<src-bit>所表示的位变量为 1，则置位进位标志；否则，保持进位标志的当前状态不变。在汇编语言中，位于源操作数之前的“/”表示将源操作数取反后使用，但源操作数本身不发生变化。在执行本指令时，不影响其他标志位。

举例： 当执行如下指令序列时，当且仅当 P1.0＝1 或 ACC.7＝1 或 OV＝0 时，置位进位标志 C：

```
MOV C, P1.0         ;LOAD CARRY WITH INPUT PIN P10
ORL C, ACC.7        ;OR CARRY WITH THE ACC.BIT 7
ORL C, /OV          ;OR CARRY WITH THE INVERSE OF OV
```

ORL C, bit

指令长度(字节)： 2

执行周期： 1 或 4

二进制编码：	0	1	1	1	0	0	1	0		bit address

操作： ORL

(C) ←(C)∨(bit)

ORL C, /bit

指令长度(字节)： 2

执行周期： 1

二进制编码：	1	0	1	0	0	0	0	0		bit address

操作： ORL

(C) ←(C)∨($\overline{bit}$)

POP direct

功能： 出栈

说明： 读取栈指针所指定的内部 RAM 单元的内容，栈指针减 1。然后，将读到的内容传送到由 direct 所指示的存储单元（直接寻址方式）中去。该操作不影响标志位。

举例： 设栈指针的初值为 32H，内部 RAM 的 30H~32H 单元的数据分别为 20H、23H 和 01H。则执行指令：

```
POP DPH
POP DPL
```

之后，栈指针的值变成 30H，数据指针变为 0123H。此时指令：

```
POP SP
```

将把栈指针变为 20H。

注意：在这种特殊情况下，在写入出栈数据（20H）之前，栈指针先减小到 2FH，然后再随着 20H 的写入，变成 20H。

指令长度(字节)： 2

执行周期： 1

二进制编码：

1	1	0	1	0	0	0	0		direct address

操作： POP

(direct)←((SP))

(SP) ← (SP) - 1

PUSH direct

功能： 压栈

说明： 栈指针首先加 1，然后将 direct 所表示的变量内容复制到由栈指针指定的内部 RAM 存储单元中去。该操作不影响标志位。

举例： 设在进入中断服务程序时栈指针的值为 09H，数据指针 DPTR 的值为 0123H。则执行如下指令序列：

PUSH DPL

PUSH DPH

之后，栈指针变为 0BH，并把数据 23H 和 01H 分别存入内部 RAM 的 0AH 和 0BH 存储单元之中。

指令长度(字节)： 2

执行周期： 1

二进制编码：

1	1	0	0	0	0	0	0		direct address

操作： PUSH

(SP) ←(SP) + 1

((SP)) ← (direct)

RET

功能： 从子例程返回

说明： 执行 RET 指令时，首先将 PC 值的高位字节和低位字节从栈中弹出，栈指针减 2。然后，程序从形成的 PC 值所对应的地址处开始执行，一般情况下，该指令和 ACALL 或 LCALL 配合使用。改指令的执行不影响标志位。

举例： 设栈指针的初值为 0BH，内部 RAM 的 0AH 和 0BH 存储单元中的数据分别为 23H 和 01H。则指令：

RET

执行后，栈指针变为 09H。程序将从 0123H 地址处继续执行。

指令长度(字节)： 1

执行周期： 3

二进制编码：

0	0	1	0	0	0	1	0

操作： RET

(PC_{15-8}) ←((SP))

(SP) ←(SP) -1

(PC_{7-0}) ←((SP))

(SP) ←(SP) -1

RETI

功能： 中断返回

说明： 执行该指令时，首先从栈中弹出 PC 值的高位和低位字节，然后恢复中断启用，准备接受同优先级的其他中断，栈指针减 2。其他寄存器不受影响。但程序状态字 PSW 不会自动恢复到中断前的状态。程序将继续从新产生的 PC 值所对应的地址处开始执行，一般情况下是此次中断入口的下一条指令。在执行 RETI 指令时，如果有一个优先级较低的或同优先级的其他中断在等待处理，那么在处理这些等待中的中断之前需要执行 1 条指令。

举例： 设栈指针的初值为 0BH，结束在地址 0123H 处的指令执行结束期间产生中断，内部 RAM 的 0AH 和 0BH 单元的内容分别为 23H 和 01H。则指令：

RETI

执行完毕后，栈指针变成 09H，中断返回后程序继续从 0123H 地址开始执行。

指令长度(字节)： 1

执行周期： 3

二进制编码：

0	0	1	1	0	0	1	0

操作： RETI

$(PC_{15-8}) \leftarrow ((SP))$

$(SP) \leftarrow (SP) - 1$

$(PC_{7-0}) \leftarrow ((SP))$

$(SP) \leftarrow (SP) - 1$

RL A

功能： 将累加器 *A* 中的数据位循环左移

说明： 将累加器中的 8 位数据均左移 1 位，其中位 7 移动到位 0。该指令的执行不影响标志位。

举例： 设累加器的内容为 0C5H（11000101B），则指令：

RL A

执行后，累加器的内容变成 8BH（10001011B），且标志位不受影响。

指令长度(字节)： 1

执行周期： 1

二进制编码：

0	0	1	0	0	0	1	1

操作： RL

$(An+1) \leftarrow (An)$ n = 0-6

$(A0) \leftarrow (A7)$

RLC A

功能： 带进位循环左移

说明： 累加器的 8 位数据和进位标志一起循环左移 1 位。其中位 7 移入进位标志，进位标志的初始状态值移到位 0。该指令不影响其他标志位。

举例： 假设累加器 *A* 的值为 0C5H(11000101B)，则指令：

RLC A

执行后，将把累加器 *A* 的数据变为 8BH(10001011B)，进位标志被置位。

指令长度(字节)： 1

执行周期： 1

二进制编码：

0	0	1	1	0	0	1	1

操作： RLC

(An+1) ←(An) n = 0-6

(A0) ←(C)

(C) ←(A7)

RR A

功能： 将累加器的数据位循环右移

说明： 将累加器的 8 个数据位均右移 1 位，位 0 将被移到位 7，即循环右移，该指令不影响标志位。

举例： 设累加器的内容为 0C5H（11000101B），则指令：

RR A

执行后累加器的内容变成 0E2H（11100010B），标志位不受影响。

指令长度(字节)： 1

执行周期： 1

二进制编码：

0	0	0	0	0	0	1	1

操作： RR

(An) ←(An+1) n = 0 - 6

(A7) ←(A0)

RRC A

功能： 带进位循环右移

说明： 累加器的 8 位数据和进位标志一起循环右移 1 位。其中位 0 移入进位标志，进位标志的初始状态值移到位 7。该指令不影响其他标志位。

举例： 假设累加器的值为 0C5H(11000101B)，进位标志为 0，则指令：

RRC A

执行后，将把累加器的数据变为 62H(01100010B)，进位标志被置位。

指令长度(字节)： 1

执行周期： 1

二进制编码：

0	0	0	1	0	0	1	1

操作： RRC

(An+1) ←(An) n = 0 - 6

(A7) ←(C)

(C) ←(A0)

SETB <bit>

功能： 置位

说明： SETB 指令可将相应的位置 1，其操作对象可以是进位标志或其他可直接寻址的位。该指令不影响其他标志位。

举例： 设进位标志被清零，端口 1 的输出状态为 34H(00110100B)，则指令：

SETB C

SETB P1.0

执行后，进位标志变为 1，端口 1 的输出状态变成 35H(00110101B)。

SETB C

指令长度(字节)： 1

执行周期： 1

二进制编码：

1	1	0	1	0	0	1	1

操作： SETB

(C) ← 1

SETB bit

指令长度(字节)： 2

执行周期： 1

二进制编码：

1	1	0	1	0	0	1	0		bit address

操作： SETB

(bit) ← 1

SJMP rel

功能： 短跳转

说明： 程序无条件跳转到 rel 所示的地址去执行。目标地址按如下方法计算：首先 PC 值加 2，然后将指令第 2 字节（即 rel）所表示的有符号偏移量加到 PC 上，得到的新 PC 值即短跳转的目标地址。所以，跳转的范围是当前指令（即 SJMP）地址的前 128 字节和后 127 字节。

举例： 设标号 RELADR 对应的指令地址位于程序存储器的 0123H 地址，则指令：

SJMP RELADR

汇编后位于 0100H。当执行完该指令后，PC 值变成 0123H。

注意：在上例中，紧接 SJMP 的下一条指令的地址是 0102H，因此，跳转的偏移量为 0123H−0102H＝21H。另外，如果 SJMP 的偏移量是 0FEH，那么构成只有 1 条指令的无限循环。

指令长度(字节)： 2

执行周期： 3

二进制编码：

1	0	0	0	0	0	0	0		rel. address

操作： SJMP

(PC) ←(PC)+2

(PC) ← (PC)+rel

SUBB A, <src-byte>

功能： 带借位的减法

说明： SUBB 指令从累加器中减去<src-byte>所代表的字节变量的数值及进位标志，减法运算的结果置于累加器中。如果执行减法时第 7 位需要借位，SUBB 将会置位进位标志（表示借位）；否则，清零进位标志。（如果在执行 SUBB 指令前，进位标志 C 已经被置位，这意味着在前面进行多精度的减法运算时，产生了借位。因而在执行本条指令时，必须把进位连同源操作数一起从累加器中减去。）如果在进行减法运算的时候，第 3 位处向上有借位，那么辅助进位标志 AC 会被置位；如果第 6 位有借位；而第 7 位没有，或是第 7 位有借位，而第 6 位没有， 则溢出标志 OV 被置位。

当进行有符号整数减法运算时，若 OV 置位，则表示在正数减负数的过程中产生了负数；

或者，在负数减正数的过程中产生了正数。

源操作数支持的寻址方式：寄存器寻址、直接寻址、寄存器间接寻址和立即数寻址。

举例： 设累加器中的数据为 0C9H(11001001B)。寄存器 *R2* 的值为 54H(01010100B)，进位标志 C 被置位。则如下指令：

SUBB A, R2

执行后，累加器的数据变为 74H(01110100B)，进位标志 C 和辅助进位标志 AC 被清零，溢出标志 C 被置位。

注意：0C9H 减去 54H 应该是 75H，但在上面的计算中，由于在 SUBB 指令执行前，进位标志 C 已经被置位，因而最终结果还需要减去进位标志，得到 74H。因此，如果在进行单精度或者多精度减法运算前，进位标志 C 的状态未知，那么应改采用 CLR C 指令把进位标志 C 清零。

SUBB A, Rn

指令长度(字节)： 1

执行周期： 1

二进制编码：

1	0	0	1	1	r	r	r

操作： SUBB

(A) ← (A) - (C) - (Rn)

SUBB A, direct

指令长度(字节)： 2

执行周期： 1

二进制编码：

1	0	0	1	0	1	0	1		direct address

操作： SUBB

(A) ← (A) - (C) - (direct)

SUBB A, @Ri

指令长度(字节)： 1

执行周期： 1

二进制编码：

1	0	0	1	0	1	1	i

操作： SUBB

(A) ← (A) - (C) - ((Ri))

SUBB A, #data

指令长度(字节)： 2

执行周期： 1

二进制编码：

1	0	0	1	0	1	0	0		immediate data

操作： SUBB

(A) ← (A) - (C) - #data

SWAP A

功能： 交换累加器的高低半字节

说明： SWAP 指令把累加器的低 4 位（位 3~位 0）和高 4 位（位 7~位 4）数据进行交换。实际上 SWAP 指令也可视为 4 位的循环指令。该指令不影响标志位。

举例： 设累加器的内容为 0C5H（11000101B），则指令：

SWAP A

执行后，累加器的内容变成 5CH（01011100B）。

指令长度(字节)： 1

执行周期： 1

二进制编码：

1	1	0	0	0	1	0	0

操作： SWAP

$(A_{3-0}) \leftrightarrow (A_{7-4})$

XCH A, <byte>

功能： 交换累加器和字节变量的内容

说明： XCH 指令将<byte>所指定的字节变量的内容装载到累加器，同时将累加器的旧内容写入<byte>所指定的字节变量。指令中的源操作数和目的操作数允许的寻址方式：寄存器寻址、直接寻址和寄存器间接寻址。

举例： 设 R0 的内容为地址 20H，累加器的值为 3FH (00111111B)。内部 RAM 的 20H 单元的内容为 75H (01110101B)。则指令：

XCH A, @R0

执行后，内部 RAM 的 20H 单元的数据变为 3FH (00111111B)，累加器的内容变为 75H(01110101B)。

XCH A, Rn

指令长度(字节)： 1

执行周期： 1

二进制编码：

1	1	0	0	1	r	r	r

操作： XCH

(A) ↔ (Rn)

XCH A, direct

指令长度(字节)： 2

执行周期： 1

二进制编码：

1	1	0	0	0	1	0	1		direct address

操作： XCH

(A) ↔ (direct)

XCH A, @Ri

指令长度(字节)： 1

执行周期： 1

二进制编码：

1	1	0	0	0	1	1	i

操作： XCH

(A) ↔ ((Ri))

XCHD A, @Ri

功能： 交换累加器和@Ri 对应单元中的数据的低 4 位

说明： XCHD 指令将累加器内容的低半字节（位 0~3，一般是十六进制数或 BCD 码）和间接寻址的内部 RAM 单元的数据进行交换，各自的高半字（位 7~4）节不受影响。另外，该指令不影响标志位。

举例： 设 R0 保存了地址 20H，累加器的内容为 36H (00110110B)。内部 RAM 的 20H 单元存储的数据为 75H (011110101B)。则指令：

XCHD A, @R0

执行后，内部 RAM 20H 单元的内容变成 76H (01110110B)，累加器的内容变为 35H(00110101B)。

指令长度(字节)： 1

执行周期： 1

二进制编码：

1	1	0	1	0	1	1	i

操作： XCHD

$(A_{3-0}) \leftrightarrow (Ri_{3-0})$

XRL <dest-byte>, <src-byte>

功能： 字节变量的逻辑异或

说明： XRL 指令将<dest-byte>和<src-byte>所代表的字节变量逐位进行逻辑异或运算，结果保存在<dest-byte>所代表的字节变量里。该指令不影响标志位。

两个操作数组合起来共支持 6 种寻址方式：当目的操作数为累加器时，源操作数可以采用寄存器寻址、直接寻址、寄存器间接寻址和立即数寻址；当目的操作数是可直接寻址的数据时，源操作数可以是累加器或者立即数。

注意：如果该指令被用来修改输出引脚上的状态，那么 dest-byte 所代表的数据就是从端口输出数据锁存器中获取的数据，而不是从引脚上读取的数据。

举例： 如果累加器和寄存器 0 的内容分别为 0C3H (11000011B)和 0AAH(10101010B)，则指令：

XRL A, R0

执行后，累加器的内容变成 69H (01101001B)。

当目的操作数是可直接寻址字节数据时，该指令可把任何 RAM 单元或者寄存器中的各个位取反。具体哪些位会被取反，在运行过程当中确定。指令：

XRL P1, #00110001B

执行后，P1 口的位 5、4、0 被取反。

XRL A, Rn

指令长度(字节)： 1

执行周期： 1

二进制编码：

0	1	1	0	1	r	r	r

操作： XRL

(A) ←(A) ∀ (Rn)

XRL A, direct

指令长度(字节)： 2

执行周期： 1

二进制编码：

0	1	1	0	0	1	0	1		direct address

操作： XRL

(A) ←(A) ∀ (direct)

XRL A, @Ri

指令长度(字节)： 1

执行周期： 1

二进制编码：

0	1	1	0	0	1	1	i

操作： XRL

(A) ←(A) ∀ ((Ri))

XRL A, #data

指令长度(字节)： 2

执行周期： 1

二进制编码：

0	1	1	0	0	1	0	0		immediate data

操作： XRL

(A) ←(A) ∀ #data

XRL direct, A

指令长度(字节)： 2

执行周期： 1

二进制编码：

0	1	1	0	0	0	1	0		direct address

操作： XRL

(direct) ←(direct) ∀ (A)

XRL direct, #data

指令长度(字节)： 3

执行周期： 1

二进制编码：

0	1	1	0	0	0	1	1		direct address		immediate data

操作： XRL

(direct)← (direct) ∀ #data

16.3.2 Instruction Definitions of Traditional 8051 MCU

ACALL addr11

Function: Absolute Call

Description: ACALL unconditionally calls a subroutine located at the indicated address. The instruction increments the PC twice to obtain the address of the following instruction, then pushes the 16-bit result onto the stack (low-order byte first) and increments the Stack Pointer twice.
The destination address is obtained by successively concatenating the five high-order bits of the incremented PC opcode bits 7-5, and the second byte of the instruction. The subroutine called must therefore start within the same 2K block of the program memory as the first byte of the instruction following ACALL. No flags are affected.

Example: Initially SP equals 07H. The label "SUBRTN" is at program memory location 0345H. After executing the instruction,
ACALL SUBRTN
at location 0123H, SP will contain 09H, internal RAM locations 08H and 09H will contain 25H and 01H, respectively, and the PC will contain 0345H.

Bytes: 2

Cycles: 3

Encoding:

A10	A9	A8	1	0	0	0	1		A7	A6	A5	A4	A3	A2	A1	A0

Operation: ACALL
(PC)←(PC)+ 2
(SP)←(SP) + 1
((SP)) ←(PC_{7-0})
(SP)←(SP) + 1
((SP))←(PC_{15-8})
(PC_{10-0})←page address

ADD A, <src-byte>

Function: Add

Description: ADD adds the byte variable indicated to the Accumulator, leaving the result in the Accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.
OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.
Four source operand addressing modes are allowed: register, direct register-indirect, or immediate.

Example: The Accumulator holds 0C3H(11000011B) and register 0 holds 0AAH (10101010B). The instruction,
ADD A, R0
will leave 6DH (01101101B) in the Accumulator with the AC flag cleared and both the carry flag and OV set to 1.

ADD A, Rn

Bytes: 1

Cycles: 1

Encoding:

0	0	1	0	1	r	r	r

Operation: ADD

(A)← (A) + (Rn)

ADD A, direct

Bytes: 2

Cycles: 1

Encoding:

0	0	1	0	0	1	0	1		direct address

Operation: ADD

(A)← (A) + (direct)

ADD A, @Ri

Bytes: 1

Cycles: 1

Encoding:

0	0	1	0	0	1	1	i

Operation: ADD

(A)← (A) + ((Ri))

ADD A, #data

Bytes: 2

Cycles: 1

Encoding:

0	0	1	0	0	1	0	0		immediate data

Operation: ADD

(A)← (A) + #data

ADDC A, <src-byte>

Function: Add with Carry

Description: ADDC simultaneously adds the byte variable indicated, the Carry flag and the Accumulator, leaving the result in the Accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not out of bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

Example: The Accumulator holds 0C3H(11000011B) and register 0 holds 0AAH (10101010B) with the Carry. The instruction,

ADDC A,R0

will leave 6EH (01101110B) in the Accumulator with the AC flag cleared and both the carry flag and OV set to 1.

ADDC A, Rn

Bytes: 1

Cycles: 1

Encoding:

0	0	1	1	1	r	r	r

Operation: ADDC

(A)← (A) + (C) + (Rn)

ADDC A, direct

Bytes: 2

Cycles: 1

Encoding:	0	0	1	1	0	1	0	1		direct address

Operation: ADDC

(A)← (A) + (C) + (direct)

ADDC A, @Ri

Bytes: 1

Cycles: 1

Encoding:	0	0	1	1	0	1	1	i

Operation: ADDC

(A)← (A) + (C) + ((Ri))

ADDC A, #data

Bytes: 2

Cycles: 1

Encoding:	0	0	1	1	0	1	0	0		immediate data

Operation: ADDC

(A)← (A) + (C) + #data

AJMP addr11

Function: Absolute Jump

Description: AJMP transfers program execution to the indicated address, which is formed at run-time by concatenating the high-order five bits of the PC (after incrementing the PC twice), opcode bits 7-5, and the second byte of the instruction. The destination must therefore be within the same 2K block of program memory as the first byte of the instruction following AJMP.

Example: The label “JMPADR” is at program memory location 0123H. The instruction,

AJMP JMPADR

is at location 0345H and will load the PC with 0123H.

Bytes: 2

Cycles: 3

Encoding:	A10	A9	A8	0	0	0	0	1		A7	A6	A5	A4	A3	A2	A1	A0

Operation: AJMP

(PC)←(PC)+ 2

(PC_{10-0})←page address

ANL <dest-byte> , <src-byte>

Function: Logical-AND for byte variables

Description: ANL performs the bitwise logical-AND operation between the variables indicated and stores the results in the destination variable. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch not the input pins.

Example: If the Accumulator holds 0C3H(11000011B) and register 0 holds 55H (01010101B) then the instruction,

ANL A,R0

will leave 41H (01000001B) in the Accumulator.

When the destination is a directly addressed byte, this instruction will clear combinations of bits in any RAM location or hardware register. The mask byte determining the pattern of bits to be cleared would either be a constant contained in the instruction or a value computed in the Accumulator at run-time. The instruction,

ANL Pl, #01110011B

will clear bits 7, 3, and 2 of output port 1.

ANL A, Rn

Bytes: 1

Cycles: 1

Encoding:

0	1	0	1	1	r	r	r

Operation: ANL

(A)← (A) ∧ (Rn)

ANL A, direct

Bytes: 2

Cycles: 1

Encoding:

0	1	0	1	0	1	0	1		direct address

Operation: ANL

(A)← (A) ∧ (direct)

ANL A, @Ri

Bytes: 1

Cycles: 1

Encoding:

0	1	0	1	0	1	1	i

Operation: ANL

(A)← (A) ∧ ((Ri))

ANL A, #data

Bytes: 2

Cycles: 1

Encoding:

0	1	0	1	0	1	0	0		immediate data

Operation: ANL

(A)← (A) ∧ #data

ANL direct, A

Bytes: 2

Cycles: 1

Encoding:

0	1	0	1	0	0	1	0		direct address

Operation: ANL

(direct)←(direct) ∧ (A)

ANL direct, #data

Bytes: 3

Cycles: 1

Encoding:

0	1	0	1	0	0	1	1		direct address		immediate data

Operation: ANL

(direct)←(direct) ∧ #data

ANL C, <src-bit>

Function: Logical-AND for bit variables

Description: If the Boolean value of the source bit is a logical 0 then clear the carry flag; otherwise leave the carry flag in its current state. A slash (" / ") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, *but the source bit itself is not affected*. No other flags are affected.

Only direct addressing is allowed for the source operand.

Example: Set the carry flag if, and only if, P1.0 = 1, ACC. 7 = 1, and OV = 0:

```
MOV C, P1.0          ; LOAD CARRY WITH INPUT PIN STATE
ANL C, ACC.7         ; AND CARRY WITH ACCUM. BIT.7
ANL C, /OV           ;AND WITH INVERSE OF OVERFLOW FLAG
```

ANL C, bit

Bytes: 2

Cycles: 1

Encoding:

1	0	0	0	0	0	1	0		bit address

Operation: ANL

(C) ←(C) ∧ (bit)

ANL C, /bit

Bytes: 2

Cycles: 1

Encoding:

1	0	1	1	0	0	0	0		bit address

Operation: ANL

(C)← (C) ∧ ($\overline{bit}$)

CJNE <dest-byte>, <src-byte>, rel

Function: Compare and Jump if Not Equal

Description: CJNE compares the magnitudes of the first two operands, and branches if their values are not equal. The branch destination is computed by adding the signed relative-displacement in the last instruction byte to the PC, after incrementing the PC to the start of the next instruction. The carry flag is set if the unsigned integer value of <dest-byte> is less than the unsigned integer value of <src-byte>; otherwise, the carry is cleared. Neither operand is affected.

The first two operands allow four addressing mode combinations: the Accumulator may be compared with any directly addressed byte or immediate data, and any indirect RAM location or working register can be compared with an immediate constant.

Example: The Accumulator contains 34H. Register 7 contains 56H. The first instruction in the sequence,

```
        CJNE R7,#60H, NOT_EQ
;       . . .       . . . . . .     ; R7 = 60H.
NOT_EQ: JC          REQ_LOW         ; IF R7 < 60H.
;       . . .       . . . . . .     ; R7 > 60H.
```

sets the carry flag and branches to the instruction at label NOT-EQ. By testing the carry flag, this instruction determines whether R7 is greater or less than 60H.

If the data being presented to Port 1 is also 34H, then the instruction,

WAIT: CJNE A,P1,WAIT

clears the carry flag and continues with the next instruction in sequence, since the Accumulator does equal the data read from P1. (If some other value was being input on Pl, the program will loop at this point until the P1 data changes to 34H.)

CJNE A, direct, rel

Bytes: 3

Cycles: 2 or 3

Encoding:

1	0	1	1	0	1	0	1		direct address		rel. address

Operation:

(PC) ←(PC) + 3

IF (A) < > *(direct)*

THEN

(PC) ←(PC) +*relative offset*

IF (A) < *(direct)*

THEN

(C) ← 1

ELSE

(C) ← 0

CJNE A, #data, rel

Bytes: 3

Cycles: 1 or 3

Encoding:

1	0	1	1	0	1	0	0		immediate data		rel. address

Operation:

(PC) ←(PC) + 3

IF (A) < > *(data)*

THEN

(PC) ←(PC) +*relative offset*

IF (A) < *(data)*

THEN

(C) ← 1

ELSE

(C) ← 0

CJNE Rn, #data, rel

Bytes: 3

Cycles: 2 or 3

Encoding:

1	0	1	1	1	r	r	r		immediate data		rel. address

Operation:

(PC) ←(PC) + 3

IF (Rn) < > *(data)*

THEN

(PC) ←(PC) +*relative offset*

IF (Rn) < *(data)*

THEN

(C) ← 1

ELSE

(C) ← 0

CJNE @Ri, #data, rel

Bytes: 3

Cycles: 2 or 3

Encoding:

1	0	1	1	0	1	1	i		immediate data		rel. address

Operation: (PC) ←(PC) + 3

IF (Ri) < > *(data)*

THEN

(PC) ←(PC) +*relative offset*

IF (Ri) < *(data)*

THEN

(C) ← 1

ELSE

(C) ← 0

CLR A

Function: Clear Accumulator

Description: The Accumulator is cleared (all bits set on zero). No flags are affected.

Example: The Accumulator contains 5CH (01011100B). The instruction,

CLR A

will leave the Accumulator set to 00H (00000000B).

Bytes: 1

Cycles: 1

Encoding:

1	1	1	0	0	1	0	0

Operation: CLR

(A)←0

CLR bit

Function: Clear bit

Description: The indicated bit is cleared (reset to zero). No other flags are affected. CLR can operate on the carry flag or any directly addressable bit.

Example: Port 1 has previously been written with 5DH (01011101B). The instruction,

CLR P1.2

will leave the port set to 59H (01011001B).

CLR C

Bytes: 1

Cycles: 1

Encoding:

1	1	0	0	0	0	1	1

Operation: CLR

(C)←0

CLR bit

Bytes: 2

Cycles: 1

Encoding:

1	1	0	0	0	0	1	0		bit address

Operation: CLR
(bit)←0

CPL A

Function: Complement Accumulator
Description: Each bit of the Accumulator is logically complemented (one's complement). Bits which previously contained a one are changed to a zero and vice-versa. No flags are affected.
Example: The Accumulator contains 5CH(01011100B). The instruction,
CPL A
will leave the Accumulator set to 0A3H (10100011B).
Bytes: 1
Cycles: 1

Encoding:	1	1	1	1	0	1	0	0

Operation: CPL
(A)← ($\bar{A}$)

CPL bit

Function: Complement bit
Description: The bit variable specified is complemented. A bit which had been a one is changed to zero and vice-versa. No other flags are affected. CLR can operate on the carry or any directly addressable bit.
Note: When this instruction is used to modify an output pin, the value used as the original data will be read from the output data latch, not the input pin.
Example: Port 1 has previously been written with 5BH(01011011B). The instruction,
CPL P1.1
CPL P1.2
will leave the port set to 5DH(01011101B).

CPL C

Bytes: 1
Cycles: 1

Encoding:	1	0	1	1	0	0	1	1

Operation: CPL
(C)← ($\bar{C}$)

CPL bit

Bytes: 2
Cycles: 1

Encoding:	1	0	1	1	0	0	1	0		bit address

Operation: CPL
(bit)← ($\overline{bit}$)

DA A

Function: Decimal-adjust Accumulator for Addition

Description: DA A adjusts the eight-bit value in the Accumulator resulting from the earlier addition of two variables (each in packed-BCD format), producing two four-bit digits. Any ADD or ADDC instruction may have been used to perform the addition.

If Accumulator bits 3-0 are greater than nine (xxxx1010-xxxx1111), or if the AC flag is one, six is added to the Accumulator producing the proper BCD digit in the low-order nibble. This internal addition would set the carry flag if a carry-out of the low-order four-bit field propagated through all high-order bits, but it would not clear the carry flag otherwise.

If the carry flag is now set or if the four high-order bits now exceed nine(1010xxxx- 1111xxxx), these high-order bits are incremented by six, producing the proper BCD digit in the high-order nibble. Again, this would set the carry flag if there was a carry-out of the high-order bits, but wouldn't clear the carry. The carry flag thus indicates if the sum of the original two BCD variables is greater than 100, allowing multiple precision decimal addition. OV is not affected.

All of this occurs during the one instruction cycle. Essentially, this instruction performs the decimal conversion by adding 00H, 06H, 60H, or 66H to the Accumulator, depending on initial Accumulator and PSW conditions.

Note: DA A cannot simply convert a hexadecimal number in the Accumulator to BCD notation, nor does DA A apply to decimal subtraction.

Example: The Accumulator holds the value 56H(01010110B) representing the packed BCD digits of the decimal number 56. Register 3 contains the value 67H (01100111B) representing the packed BCD digits of the decimal number 67. The carry flag is set. The instruction sequence.

```
ADDC A,R3
DA   A
```

will first perform a standard twos-complement binary addition, resulting in the value 0BEH (10111110) in the Accumulator. The carry and auxiliary carry flags will be cleared.

The Decimal Adjust instruction will then alter the Accumulator to the value 24H (00100100B), indicating the packed BCD digits of the decimal number 24, the low-order two digits of the decimal sum of 56,67, and the carry-in. The carry flag will be set by the Decimal Adjust instruction, indicating that a decimal overflow occurred. The true sum 56, 67, and 1 is 124.

BCD variables can be incremented or decremented by adding 01H or 99H. If the Accumulator initially holds 30H (representing the digits of 30 decimal), then the instruction sequence,

```
ADD A, #99H
DA   A
```

will leave the carry set and 29H in the Accumulator, since 30+99=129. The low-order byte of the sum can be interpreted to mean 30 – 1 = 29.

Bytes: 1

Cycles: 3

Encoding:

1	1	0	1	0	1	0	0

Operation: DA

-contents of Accumulator are BCD

IF [[(A_{3-0}) > 9] V [(AC) = 1]]

THEN(A_{3-0}) ←(A_{3-0}) + 6

AND

IF [[(A_{7-4}) > 9] V [(C) = 1]]

THEN (A_{7-4})←(A_{7-4}) + 6

DEC byte

Function: Decrement

Description: The variable indicated is decremented by 1. An original value of 00H will underflow to 0FFH. No flags are affected. Four operand addressing modes are allowed: accumulator, register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

Example: Register 0 contains 7FH (01111111B). Internal RAM locations 7EH and 7FH contain 00H and 40H, respectively. The instruction sequence,

DEC @R0

DEC R0

DEC @R0

will leave register 0 set to 7EH and internal RAM locations 7EH and 7FH set to 0FFH and 3FH.

DEC A

Bytes: 1

Cycles: 1

Encoding:

0	0	0	1	0	1	0	0

Operation: DEC

(A)← (A) −1

DEC Rn

Bytes: 1

Cycles: 1

Encoding:

0	0	0	1	1	r	r	r

Operation: DEC

(Rn)←(Rn) - 1

DEC direct

Bytes: 2

Cycles: 1

Encoding:

0	0	0	1	0	1	0	1		Direct address

Operation: DEC

(direct)←(direct) - 1

DEC @Ri

Bytes: 1

Cycles: 1

Encoding:

0	0	0	1	0	1	1	i

Operation: DEC

((Ri))←((Ri)) - 1

DIV AB

Function: Divide

Description: DIV AB divides the unsigned eight-bit integer in the Accumulator by the unsigned eight-bit integer in register B. The Accumulator receives the integer part of the quotient; register B receives the integer remainder. The carry and OV flags will be cleared.

Exception: if B had originally contained 00H, the values returned in the Accumulator and B-register will be undefined and the overflow flag will be set. The carry flag is cleared in any case.

Example: The Accumulator contains 251(OFBH or 11111011B) and B contains 18(12H or 00010010B). The instruction,

DIV AB

will leave 13 in the Accumulator (0DH or 00001101B) and the value 17 (11H or 00010001B) in B, since 251 = (13×18) + 17. Carry and OV will both be cleared.

Bytes: 1

Cycles: 6

Encoding:

1	0	0	0	0	1	0	0

Operation: DIV

$(A)_{15\text{-}8}(B)_{7\text{-}0} \leftarrow (A)/(B)$

DJNZ <byte>, <rel-addr>

Function: Decrement and Jump if Not Zero

Description: DJNZ decrements the location indicated by 1, and branches to the address indicated by the second operand if the resulting value is not zero. An original value of 00H will underflow to 0FFH. No flags are affected. The branch destination would be computed by adding the signed relative-displacement value in the last instruction byte to the PC, after incrementing the PC to the first byte of the following instruction.

The location decremented may be a register or directly addressed byte.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

Example: Internal RAM locations 40H, 50H, and 60H contain the values 01H, 70H, and 15H, respectively. The instruction sequence,

DJNZ 40H, LABEL_1

DJNZ 50H, LABEL_2

DJNZ 60H, LABEL_3

will cause a jump to the instruction at label LABEL_2 with the values 00H, 6FH, and 15H in the three RAM locations. The first jump was not taken because the result was zero.

This instruction provides a simple way of executing a program loop a given number of times, or for adding a moderate time delay (from 2 to 512 machine cycles) with a single instruction. The instruction sequence,

```
        MOV R2,#8
TOOOLE: CPL P1.7
```

DJNZ R2, TOOGLE

will toggle P1.7 eight times, causing four output pulses to appear at bit 7 of output Port 1. Each pulse will last three machine cycles; two for DJNZ and one to alter the pin.

DJNZ Rn, rel

Bytes: 2

Cycles: 2 or 3

Encoding:

1	1	0	1	1	r	r	r		rel. address

Operation: DJNZ

(PC) ←(PC) + 2

(Rn) ←(Rn) – 1

IF (Rn) > 0 or (Rn) < 0

THEN

(PC) ←(PC)+ rel

DJNZ direct, rel

Bytes: 3

Cycles: 2 or 3

Encoding:

1	1	0	1	0	1	0	1		direct address		rel. address

Operation: DJNZ

(PC) ←(PC) + 2

(direct) ←(direct) – 1

IF (direct) > 0 or (direct) < 0

THEN

(PC) ←(PC)+ rel

INC <byte>

Function: Increment

Description: INC increments the indicated variable by 1. An original value of 0FFH will overflow to 00H.No flags are affected. Three addressing modes are allowed: register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

Example: Register 0 contains 7EH (01111110B). Internal RAM locations 7EH and 7FH contain 0FFH and 40H, respectively. The instruction sequence,

INC @R0

INC R0

INC @R0

will leave register 0 set to 7FH and internal RAM locations 7EH and 7FH holding (respectively) 00H and 41H.

INC A

Bytes: 1

Cycles: 1

Encoding:

0	0	0	0	0	1	0	0

Operation: INC

(A) ←(A)+1

INC Rn

Bytes: 1

Cycles: 1

Encoding:

0	0	0	0	1	r	r	r

Operation: INC

(Rn) ←(Rn)+1

INC direct

Bytes: 2

Cycles: 1

Encoding:

0	0	0	0	0	1	0	1		direct address

Operation: INC

(direct) ←(direct)+1

INC @Ri

Bytes: 1

Cycles: 1

Encoding:

0	0	0	0	0	1	1	i

Operation: INC

((Ri))←((Ri)) + 1

INC DPTR

Function: Increment Data Pointer

Description: Increment the 16-bit data pointer by 1. A 16-bit increment (modulo 2_{16}) is performed; an overflow of the low-order byte of the data pointer (DPL) from 0FFH to 00H will increment the high-order-byte (DPH). No flags are affected.

This is the only 16-bit register which can be incremented.

Example: Register DPH and DPL contains 12H and 0FEH, respectively. The instruction sequence,

IINC DPTR

INC DPTR

INC DPTR

will change DPH and DPL to 13H and 01H.

Bytes: 1

Cycles: 1

Encoding:

1	0	1	0	0	0	1	1

Operation: INC

(DPTR) ← (DPTR) + 1

JB bit, rel

Function: Jump if Bit set

Description: If the indicated bit is a one, jump to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction.

The bit tested is not modified. No flags are affected

Example: The data present at input port 1 is 11001010B. The Accumulator holds 56 (01010110B). The instruction sequence,

JB P1.2, LABEL1

JB ACC.2, LABEL2

will cause program execution to branch to the instruction at label LABEL2.

Bytes: 3

Cycles: 1 or 3

Encoding:

0	0	1	0	0	0	0	0		bit address		rel. address

Operation: JB

(PC) ← (PC) + 3

IF (bit) = 1

THEN

(PC) ← (PC)+ rel

JBC bit, rel

Function: Jump if Bit is set and Clear bit

Description: If the indicated bit is one, branch to the address indicated; otherwise proceed with the next instruction. *The bit will not be cleared if it is already a zero.* The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. No flags are affected.

Note: When this instruction is used to test an output pin, the value used as the original data will be read from the output data latch, not the input pin.

Example: The Accumulator holds 56H (01010110B). The instruction sequence,

JBC ACC.3, LABEL1

JBC ACC.2, LABEL2

will cause program execution to continue at the instruction identified by the label LABEL2, with the Accumulator modified to 52H (01010010B).

Bytes: 3

Cycles: 1 or 3

Encoding:

0	0	0	1	0	0	0	0		bit address		rel. address

Operation: JB

(PC) ← (PC) + 3

IF (bit) = 1

THEN

(bit) ← 0

(PC) ← (PC)+ rel

JC rel

Function: Jump if Carry is set

Description: If the carry flag is set, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. No flags are affected.

Example: The carry flag is cleared. The instruction sequence,

JC LABEL1

CPL C

JC LABEL2

will set the carry and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 1 or 3

Encoding:

0	1	0	0	0	0	0	0		rel. address

Operation: JC

(PC) ← (PC) + 2

IF (C) = 1

THEN

(PC) ← (PC)+ rel

JMP @A+DPTR

Function: Jump indirect

Description: Add the eight-bit unsigned contents of the Accumulator with the sixteen-bit data pointer, and load the resulting sum to the program counter. This will be the address for subsequent instruction fetches. Sixteen-bit addition is performed (modulo 2^{16}): a carry-out from the low-order eight bits propagates through the higher-order bits. Neither the Accumulator nor the Data Pointer is altered. No flags are affected.

Example: An even number from 0 to 6 is in the Accumulator. The following sequence of instructions will branch to one of four AJMP instructions in a jump table starting at JMP_TBL:

```
        MOV DPTR, #JMP_TBL
        JMP @A+DPTR
JMP-TBL: AJMP LABEL0
        AJMP LABEL1
        AJMP LABEL2
        AJMP LABEL3
```

If the Accumulator equals 04H when starting this sequence, execution will jump to label LABEL2. Remember that AJMP is a two-byte instruction, so the jump instructions start at every other address.

Bytes: 1

Cycles: 4

Encoding:

0	1	1	1	0	0	1	1

Operation: JMP

(PC) ← (A) + (DPTR)

JNB bit, rel

Function: Jump if Bit is not set

Description: If the indicated bit is a zero, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified.* No flags are affected.

Example: The data present at input port 1 is 11001010B. The Accumulator holds 56H (01010110B). The

instruction sequence,

JNB P1.3, LABEL1

JNB ACC.3, LABEL2

will cause program execution to continue at the instruction at label LABEL2.

Bytes: 3

Cycles: 1 or 3

Encoding:

0	0	1	1	0	0	0	0		bit address		rel. address

Operation: JNB

(PC) ←(PC) + 3

IF (bit) = 0

THEN (PC) ← (PC) + rel

JNC rel

Function: Jump if Carry not set

Description: If the carry flag is a zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice to point to the next instruction. The carry flag is not modified.

Example: The carry flag is set. The instruction sequence,

JNC LABEL1

CPL C

JNC LABEL2

will clear the carry and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 1 or 3

Encoding:

0	1	0	1	0	0	0	0		rel. address

Operation: JNC

(PC) ←(PC) + 2

IF (C) = 0

THEN (PC) ← (PC) + rel

JNZ rel

Function: Jump if Accumulator Not Zero

Description: If any bit of the Accumulator is a one, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.

Example: The Accumulator originally holds 00H. The instruction sequence,

JNZ LABEL1

INC A

JNZ LAEEL2

will set the Accumulator to 01H and continue at label LABEL2.

Bytes: 2

Cycles: 1 or 3

Encoding:

0	1	1	1	0	0	0	0		rel. address

Operation: JNZ

(PC) ←(PC) + 2

IF (A) ≠ 0

THEN (PC) ← (PC) + rel

JZ rel

Function: Jump if Accumulator Zero

Description: If all bits of the Accumulator are zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.

Example: The Accumulator originally contains 01H. The instruction sequence,

JZ LABEL1

DEC A

JZ LAEEL2

will change the Accumulator to 00H and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 1 or 3

Encoding:

0	1	1	0	0	0	0	0		rel. address

Operation: JZ

(PC) ←(PC) + 2

IF (A) = 0

THEN (PC) ← (PC) + rel

LCALL addr16

Function: Long call

Description: LCALL calls a subroutine located at the indicated address. The instruction adds three to the program counter to generate the address of the next instruction and then pushes the 16-bit result onto the stack (low byte first), incrementing the Stack Pointer by two. The high-order and low-order bytes of the PC are then loaded, respectively, with the second and third bytes of the LCALL instruction. Program execution continues with the instruction at this address.
The subroutine may therefore begin anywhere in the full 64K-byte program memory address space. No flags are affected.

Example: Initially the Stack Pointer equals 07H. The label “SUBRTN” is assigned to program memory location 1234H. After executing the instruction,

LCALL SUBRTN

at location 0123H, the Stack Pointer will contain 09H, internal RAM locations 08H and 09H will contain 26H and 01H, and the PC will contain 1234H.

Bytes: 3

Cycles: 3

Encoding:	0	0	0	1	0	0	1	0		addr15-addr8		addr7-addr0

Operation: LCALL

(PC) ←(PC) + 3

(SP) ←(SP) + 1

((SP)) ←(PC_{7-0})

(SP) ←(SP) + 1

((SP)) ←(PC_{15-8})

(PC) ← $addr_{15-0}$

LJMP addr16

Function: Long Jump

Description: LJMP causes an unconditional branch to the indicated address, by loading the high-order and low-order bytes of the PC (respectively) with the second and third instruction bytes. The destination may therefore be anywhere in the full 64K program memory address space. No flags are affected.

Example: The label “JMPADR” is assigned to the instruction at program memory location 1234H. The instruction,

LJMP JMPADR

at location 0123H will load the program counter with 1234H.

Bytes: 3

Cycles: 3

Encoding:	0	0	0	0	0	0	1	0		addr15-addr8		addr7-addr0

Operation: LJMP

(PC) ← $addr_{15-0}$

MOV <dest-byte> , <src-byte>

Function: Move byte variable

Description: The byte variable indicated by the second operand is copied into the location specified by the first operand. The source byte is not affected. No other register or flag is affected.

This is by far the most flexible operation. Fifteen combinations of source and destination addressing modes are allowed.

Example: Internal RAM location 30H holds 40H. The value of RAM location 40H is 10H. The data present at input port 1 is 11001010B (0CAH).

MOV R0, #30H ; R0< = 30H

MOV A, @R0 ; A < = 40H

MOV R1, A ; R1 < = 40H

MOV B, @Rl ; B < = 10H

MOV @Rl, Pl ; RAM (40H) < = 0CAH

MOV P2, P1 ; P2 #0CAH

leaves the value 30H in register 0,40H in both the Accumulator and register 1,10H in register B, and 0CAH(11001010B) both in RAM location 40H and output on port 2.

MOV A,Rn

Bytes: 1

Cycles: 1

Encoding:

1	1	1	0	1	r	r	r

Operation: MOV

(A) ←(Rn)

***MOV A,direct**

Bytes: 2

Cycles: 1

Encoding:

1	1	1	0	0	1	0	1		direct address

Operation: MOV

(A)← (direct)

***MOV A, ACC is not a valid instruction.**

MOV A,@Ri

Bytes: 1

Cycles: 1

Encoding:

1	1	1	0	0	1	1	i

Operation: MOV

(A) ←((Ri))

MOV A,#data

Bytes: 2

Cycles: 1

Encoding:

0	1	1	1	0	1	0	0		immediate data

Operation: MOV

(A)←#data

MOV Rn, A

Bytes: 1

Cycles: 1

Encoding:

1	1	1	1	1	r	r	r

Operation: MOV

(Rn) ←(A)

MOV Rn,direct

Bytes: 2

Cycles: 1

Encoding:

1	0	1	0	1	r	r	r		direct address

Operation: MOV

(Rn)← (direct)

MOV Rn,#data

Bytes: 2

Cycles: 1

Encoding:

0	1	1	1	1	r	r	r		immediate data

Operation: MOV

(Rn)←#data

MOV direct, A

Bytes: 2

Cycles: 1

Encoding:

1	1	1	1	0	1	0	1		direct address

Operation: MOV

(direct)← (A)

MOV direct, Rn

Bytes: 2

Cycles: 1

Encoding:

1	0	0	0	1	r	r	r		direct address

Operation: MOV

(direct)← (Rn)

MOV direct, direct

Bytes: 3

Cycles: 1

Encoding:

1	0	0	0	0	1	0	1		dir.addr. (src)		dir.addr. (dest)

Operation: MOV

(direct)← (direct)

MOV direct, @Ri

Bytes: 2

Cycles: 1

Encoding:

1	0	0	0	0	1	1	i		direct address

Operation: MOV

(direct)←((Ri))

MOV direct, #data

Bytes: 3

Cycles: 1

Encoding:

0	1	1	1	0	1	0	1		direct address		immediate data

Operation: MOV

(direct)←#data

MOV @Ri, A

Bytes: 1

Cycles: 1

Encoding:

1	1	1	1	0	1	1	i

Operation: MOV

((Ri))←(A)

MOV @Ri, direct

Bytes: 2

Cycles: 1

Encoding:

1	0	1	0	0	1	1	i		direct address

Operation: MOV

((Ri))←(direct)

MOV @Ri, #data

Bytes: 2

Cycles: 1

Encoding:

0	1	1	1	0	1	1	i		immediate data

Operation: MOV

((Ri))←#data

MOV <dest-bit> , <src-bit>

Function: Move bit data

Description: The Boolean variable indicated by the second operand is copied into the location specified by the first operand. One of the operands must be the carry flag; the other may be any directly addressable bit. No other register or flag is affected.

Example: The carry flag is originally set. The data present at input Port 3 is 11000101B. The data previously written to output Port 1 is 35H (00110101B).

MOV P1.3, C

MOV C, P3.3

MOV P1.2, C

will leave the carry cleared and change Port 1 to 39H (00111001B).

MOV C, bit

Bytes: 2

Cycles: 1

Encoding:

1	0	1	0	0	0	1	0		bit address

Operation: MOV

(C)← (bit)

MOV bit, C

Bytes: 2

Cycles: 1

Encoding:

1	0	0	1	0	0	1	0		bit address

Operation: MOV

(bit)← (C)

MOV DPTR , #data 16

Function: Load Data Pointer with a 16-bit constant

Description: The Data Pointer is loaded with the 16-bit constant indicated. The 16-bit constant is loaded into the second and third bytes of the instruction. The second byte (DPH) is the high-order byte, while the third byte (DPL) holds the low-order byte. No flags are affected.

This is the only instruction which moves 16 bits of data at once.

Example: The instruction,

MOV DPTR, #1234H

will load the value 1234H into the Data Pointer: DPH will hold 12H and DPL will hold 34H.

Bytes: 3

Cycles: 1

Encoding:

1	0	0	1	0	0	0	0		immediate data15-8		immediate data7-0

Operation: MOV

(DPTR) ← #data$_{15\text{-}0}$

DPH DPL ← #data$_{15\text{-}8}$ #data$_{7\text{-}0}$

MOVC A , @A+ <base-reg>

Function: Move Code byte

Description: The MOVC instructions load the Accumulator with a code byte, or constant from program memory. The address of the byte fetched is the sum of the original unsigned eight-bit. Accumulator contents and the contents of a sixteen-bit base register, which may be either the Data Pointer or the PC. In the latter case, the PC is incremented to the address of the following instruction before being added with the Accumulator; otherwise the base register is not altered. Sixteen-bit addition is performed so a carry-out from the low-order eight bits may propagate through higher-order bits. No flags are affected.

Example: A value between 0 and 3 is in the Accumulator. The following instructions will translate the value in the Accumulator to one of four values defined by the DB (define byte) directive.

```
REL-PC: INC A
        MOVC A, @A+PC
        RET
        DB 66H
        DB 77H
        DB 88H
        DB 99H
```

If the subroutine is called with the Accumulator equal to 01H, it will return with 77H in the Accumulator. The INC A before the MOVC instruction is needed to “get around” the RET instruction above the table. If several bytes of code separated the MOVC from the table, the corresponding number would be added to the Accumulator instead.

MOVC A,@A+DPTR

Bytes: 1

Cycles: 4

Encoding:

1	0	0	1	0	0	1	1

Operation: MOVC

(A) ←((A)+(DPTR))

MOVC A,@A+PC

Bytes: 1

Cycles: 3

Encoding:

1	0	0	0	0	0	1	1

Operation: MOVC

(PC) ←(PC)+1

(A) ←((A)+(PC))

MOVX <dest-byte> , <src-byte>

Function: Move External

Description: The MOVX instructions transfer data between the Accumulator and a byte of external data memory, hence the “X” appended to MOV. There are two types of instructions, differing in whether they provide an eight-bit or sixteen-bit indirect address to the external data RAM.

In the first type, the contents of R0 or R1 in the current register bank provide an eight-bit address multiplexed with data on P0. Eight bits are sufficient for external I/O expansion decoding or for a relatively small RAM array. For somewhat larger arrays, any output port pins can be used to

output higher-order address bits. These pins would be controlled by an output instruction preceding the MOVX.

In the second type of MOVX instruction, the Data Pointer generates a sixteen-bit address. P2 outputs the high-order eight address bits (the contents of DPH) while P0 multiplexes the low-order eight bits (DPL) with data. The P2 Special Function Register retains its previous contents while the P2 output buffers are emitting the contents of DPH. This form is faster and more efficient when accessing very large data arrays (up to 64K bytes), since no additional instructions are needed to set up the output ports.

It is possible in some situations to mix the two MOVX types. A large RAM array with its high-order address lines driven by P2 can be addressed via the Data Pointer, or with code to output high-order address bits to P2 followed by a MOVX instruction using R0 or R1.

Example: An external 256 byte RAM using multiplexed address/data lines (e.g., an Intel 8155 RAM/ I/O/Timer) is connected to the 8051 Port 0. Port 3 provides control lines for the external RAM. Ports 1 and 2 are used for normal I/O. Registers 0 and 1 contain 12H and 34H.

Location 34H of the external RAM holds the value 56H. The instruction sequence,

MOVX A, @R1

MOVX @R0, A

copies the value 56H into both the Accumulator and external RAM location 12H.

MOVX A,@Ri

Bytes: 1

Cycles: 3 or 1

Encoding:

1	1	1	0	0	0	1	i

Operation: MOVX

(A) ← ((Ri))

MOVX A,@DPTR

Bytes: 1

Cycles: 2 or 1

Encoding:

1	1	1	0	0	0	0	0

Operation: MOVX

(A) ←((DPTR))

MOVX @Ri, A

Bytes: 1

Cycles: 3 or 1

Encoding:

1	1	1	1	0	0	1	i

Operation: MOVX

((Ri))←(A)

MOVX @DPTR, A

Bytes: 1

Cycles: 2 or 1

Encoding:

1	1	1	1	0	0	0	0

Operation: MOVX

(DPTR)←(A)

MUL AB

Function: Multiply

Description: MUL AB multiplies the unsigned eight-bit integers in the Accumulator and register B. The low-order byte of the sixteen-bit product is left in the Accumulator, and the high-order byte in B. If the product is greater than 255 (0FFH) the overflow flag is set; otherwise it is cleared.
The carry flag is always cleared.

Example: Originally the Accumulator holds the value 80 (50H). Register B holds the value 160 (0A0H). The instruction,
MUL AB
will give the product 12,800 (3200H), so B is changed to 32H (00110010B) and the Accumulator is cleared. The overflow flag is set, carry is cleared.

Bytes: 1

Cycles: 2

Encoding:

1	0	1	0	0	1	0	0

Operation: $(A)_{7\text{-}0} \leftarrow (A) \times (B)$
$(B)_{15\text{-}8}$

NOP

Function: No Operation

Description: Execution continues at the following instruction. Other than the PC, no registers or flags are affected.

Example: It is desired to produce a low-going output pulse on bit 7 of Port 2 lasting exactly 5 cycles. A simple SETB/CLR sequence would generate a one-cycle pulse, so four additional cycles must be inserted. This may be done (assuming no interrupts are enabled) with the instruction sequence.
CLR P2.7
NOP
NOP
NOP
NOP
SETB P2.7

Bytes: 1

Cycles: 1

Encoding:

0	0	0	0	0	0	0	0

Operation: NOP
(PC)←(PC)+1

ORL <dest-byte> , <src-byte>

Function: Logical-OR for byte variables

Description: ORL performs the bitwise logical-OR operation between the indicated variables, storing the results in the destination byte. No flags are affected.
The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.
Note: When this instruction is used to modify an output port, the value used as the original port

data will be read from the output data latch, not the input pins.

Example: If the Accumulator holds 0C3H (11000011B) and R0 holds 55H (01010101B) then the instruction,

ORL A, R0

will leave the Accumulator holding the value 0D7H (11010111B).

When the destination is a directly addressed byte, the instruction can set combinations of bits in any RAM location or hardware register. The pattern of bits to be set is determined by a mask byte, which may be either a constant data value in the instruction or a variable computed in the Accumulator at run-time. The instruction,

ORL P1, #00110010B

will set bits 5,4, and 1of output Port 1.

ORL A, Rn

Bytes: 1

Cycles: 1

Encoding:

0	1	0	0	1	r	r	r

Operation: ORL

(A) ←(A)∨(Rn)

ORL A, direct

Bytes: 2

Cycles: 1

Encoding:

0	1	0	0	0	1	0	1		direct address

Operation: ORL

(A) ←(A)∨(direct)

ORL A, @Ri

Bytes: 1

Cycles: 1

Encoding:

0	1	0	0	0	1	1	i

Operation: ORL

(A) ←(A)∨((Ri))

ORL A, #data

Bytes: 2

Cycles: 1

Encoding:

0	1	0	0	0	1	0	0		immediate data

Operation: ORL

(A) ←(A)∨#data

ORL direct, A

Bytes: 2

Cycles: 1

Encoding:

0	1	0	0	0	0	1	0		direct address

Operation: ORL

(direct)← (direct)∨(A)

ORL direct, #data

Bytes: 3

Cycles: 1

Encoding:

0	1	0	0	0	0	1	1		direct address		immediate data

Operation: ORL

(direct)← (direct)∨#data

ORL C, <src-bit>

Function: Logical-OR for bit variables

Description: Set the carry flag if the Boolean value is a logical 1; leave the carry in its current state otherwise. A slash (" / ") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected.

Example: Set the carry flag if and only if P1.0 = 1, ACC. 7 = 1, or OV = 0:

```
MOV C, P1.0         ;LOAD CARRY WITH INPUT PIN P10
ORL C, ACC.7        ;OR CARRY WITH THE ACC.BIT 7
ORL C, /OV          ;OR CARRY WITH THE INVERSE OF OV
```

ORL C, bit

Bytes: 2

Cycles: 1 or 4

Encoding:

0	1	1	1	0	0	1	0		bit address

Operation: ORL

(C) ←(C)∨(bit)

ORL C, /bit

Bytes: 2

Cycles: 1

Encoding:

1	0	1	0	0	0	0	0		bit address

Operation: ORL

(C) ←(C)∨($\overline{bit}$)

POP direct

Function: Pop from stack

Description: The contents of the internal RAM location addressed by the Stack Pointer is read, and the Stack Pointer is decremented by one. The value read is then transferred to the directly addressed byte indicated. No flags are affected.

Example: The Stack Pointer originally contains the value 32H, and internal RAM locations 30H through 32H contain the values 20H, 23H, and 01H, respectively. The instruction sequence,

```
POP DPH
POP DPL
```

will leave the Stack Pointer equal to the value 30H and the Data Pointer set to 0123H. At this point the instruction,

```
POP SP
```

will leave the Stack Pointer set to 20H. Note that in this special case the Stack Pointer was decremented to 2FH before being loaded with the value popped (20H).

Bytes: 2

Cycles: 1

Encoding:

1	1	0	1	0	0	0	0		direct address

Operation: POP
(direct)←((SP))
(SP) ← (SP) - 1

PUSH direct

Function: Push onto stack
Description: The Stack Pointer is incremented by one. The contents of the indicated variable is then copied into the internal RAM location addressed by the Stack Pointer. Otherwise no flags are affected.
Example: On entering an interrupt routine the Stack Pointer contains 09H. The Data Pointer holds the value 0123H. The instruction sequence,
PUSH DPL
PUSH DPH
will leave the Stack Pointer set to 0BH and store 23H and 01H in internal RAM locations 0AH and 0BH, respectively.
Bytes: 2
Cycles: 1
Encoding:

1	1	0	0	0	0	0	0		direct address

Operation: PUSH
(SP) ←(SP) + 1
((SP)) ← (direct)

RET

Function: Return from subroutine
Description: RET pops the high-and low-order bytes of the PC successively from the stack, decrementing the Stack Pointer by two. Program execution continues at the resulting address, generally the instruction immediately following an ACALL or LCALL. No flags are affected.
Example: The Stack Pointer originally contains the value 0BH. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction,
RET
will leave the Stack Pointer equal to the value 09H. Program execution will continue at location 0123H.
Bytes: 1
Cycles: 3
Encoding:

0	0	1	0	0	0	1	0

Operation: RET
(PC_{15-8}) ←((SP))
(SP) ←(SP) -1
(PC_{7-0}) ←((SP))
(SP) ←(SP) -1

RETI

Function: Return from interrupt

Description: RETI pops the high- and low-order bytes of the PC successively from the stack, and restores the interrupt logic to accept additional interrupts at the same priority level as the one just processed. The Stack Pointer is left decremented by two. No other registers are affected; the PSW is not automatically restored to its pre-interrupt status. Program execution continues at the resulting address, which is generally the instruction immediately after the point at which the interrupt request was detected. If a lower- or same-level interrupt had been pending when the RETI instruction is executed, that one instruction will be executed before the pending interrupt is processed.

Example: The Stack Pointer originally contains the value 0BH. An interrupt was detected during the instruction ending at location 0122H. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction,

RETI

will leave the Stack Pointer equal to 09H and return program execution to location 0123H.

Bytes: 1

Cycles: 3

Encoding:

0	0	1	1	0	0	1	0

Operation: RETI

$(PC_{15-8}) \leftarrow ((SP))$

$(SP) \leftarrow (SP) - 1$

$(PC_{7-0}) \leftarrow ((SP))$

$(SP) \leftarrow (SP) - 1$

RL A

Function: Rotate Accumulator Left

Description: The eight bits in the Accumulator are rotated one bit to the left. Bit 7 is rotated into the bit 0 position. No flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B). The instruction,

RL A

leaves the Accumulator holding the value 8BH (10001011B) with the carry unaffected.

Bytes: 1

Cycles: 1

Encoding:

0	0	1	0	0	0	1	1

Operation: RL

$(An+1) \leftarrow (An)$ n = 0-6

$(A0) \leftarrow (A7)$

RLC A

Function: Rotate Accumulator Left through the Carry flag

Description: The eight bits in the Accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag; the original state of the carry flag moves into the bit 0 position. No other flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B), and the carry is zero. The instruction,

RLC A

leaves the Accumulator holding the value 8BH (10001011B) with the carry set.

Bytes: 1

Cycles: 1

Encoding:

0	0	1	1	0	0	1	1

Operation: RLC

(An+1) ←(An) n = 0-6

(A0) ←(C)

(C) ←(A7)

RR A

Function: Rotate Accumulator Right

Description: The eight bits in the Accumulator are rotated one bit to the right. Bit 0 is rotated into the bit 7 position. No flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B). The instruction,

RR A

leaves the Accumulator holding the value 0E2H (11100010B) with the carry unaffected.

Bytes: 1

Cycles: 1

Encoding:

0	0	0	0	0	0	1	1

Operation: RR

(An) ←(An+1) n = 0 - 6

(A7) ←(A0)

RRC A

Function: Rotate Accumulator Right through the Carry flag

Description: The eight bits in the Accumulator and the carry flag are together rotated one bit to the right. Bit 0 moves into the carry flag; the original value of the carry flag moves into the bit 7 position. No other flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B), and the carry is zero. The instruction,

RRC A

leaves the Accumulator holding the value 62H (01100010B) with the carry set.

Bytes: 1

Cycles: 1

Encoding:

0	0	0	1	0	0	1	1

Operation: RRC

(An+1) ←(An) n = 0 - 6

(A7) ←(C)

(C) ←(A0)

SETB <bit>

Function: Set bit

Description: SETB sets the indicated bit to one. SETB can operate on the carry flag or any directly

addressable bit. No other flags are affected.

Example: The carry flag is cleared. Output Port 1 has been written with the value 34H (00110100B).
The instructions,
SETB C
SETB P1.0
will leave the carry flag set to 1 and change the data output on Port 1 to 35H (00110101B).

SETB C

Bytes: 1

Cycles: 1

Encoding:

1	1	0	1	0	0	1	1

Operation: SETB
(C) ← 1

SETB bit

Bytes: 2

Cycles: 1

Encoding:

1	1	0	1	0	0	1	0		bit address

Operation: SETB
(bit) ← 1

SJMP rel

Function: Short Jump

Description: Program control branches unconditionally to the address indicated. The branch destination is computed by adding the signed displacement in the second instruction byte to the PC, after incrementing the PC twice. Therefore, the range of destinations allowed is from 128bytes preceding this instruction to 127 bytes following it.

Example: The label “RELADR” is assigned to an instruction at program memory location 0123H. The instruction,
SJMP RELADR
will assemble into location 0100H. After the instruction is executed, the PC will contain the value 0123H.
(*Note:* Under the above conditions the instruction following SJMP will be at 102H.Therefore, the displacement byte of the instruction will be the relative offset (0123H - 0102H) = 21H. Put another way, an SJMP with a displacement of 0FEH would be an one-instruction infinite loop).

Bytes: 2

Cycles: 3

Encoding:

1	0	0	0	0	0	0	0		rel. address

Operation: SJMP
(PC) ←(PC)+2
(PC) ← (PC)+rel

SUBB A, <src-byte>

Function: Subtract with borrow

Description: SUBB subtracts the indicated variable and the carry flag together from the Accumulator, leaving

the result in the Accumulator. SUBB sets the carry (borrow)flag if a borrow is needed for bit 7, and clears C otherwise.(If C was set before executing a SUBB instruction, this indicates that a borrow was needed for the previous step in a multiple precision subtraction, so the carry is subtracted from the Accumulator along with the source operand).AC is set if a borrow is needed for bit 3, and cleared otherwise. OV is set if a borrow is needed into bit 6, but not into bit 7, or into bit 7, but not bit 6.

When subtracting signed integers OV indicates a negative number produced when a negative value is subtracted from a positive value, or a positive result when a positive number is subtracted from a negative number.

The source operand allows four addressing modes: register, direct, register-indirect, or immediate.

Example: The Accumulator holds 0C9H (11001001B), register 2 holds 54H (01010100B), and the carry flag is set. The instruction,

SUBB A, R2

will leave the value 74H (01110100B) in the accumulator, with the carry flag and AC cleared but OV set.

Notice that 0C9H minus 54H is 75H. The difference between this and the above result is due to the carry (borrow) flag being set before the operation. If the state of the carry is not known before starting a single or multiple-precision subtraction, it should be explicitly cleared by a CLR C instruction.

SUBB A, Rn

Bytes: 1

Cycles: 1

Encoding:

1	0	0	1	1	r	r	r

Operation: SUBB

(A) ← (A) - (C) - (Rn)

SUBB A, direct

Bytes: 2

Cycles: 1

Encoding:

1	0	0	1	0	1	0	1		direct address

Operation: SUBB

(A) ← (A) - (C) - (direct)

SUBB A, @Ri

Bytes: 1

Cycles: 1

Encoding:

1	0	0	1	0	1	1	i

Operation: SUBB

(A) ← (A) - (C) - ((Ri))

SUBB A, #data

Bytes: 2

Cycles: 1

Encoding:

1	0	0	1	0	1	0	0		immediate data

Operation: SUBB

(A) ← (A) - (C) - #data

SWAP A

Function: Swap nibbles within the Accumulator

Description: SWAP A interchanges the low- and high-order nibbles (four-bit fields) of the Accumulator (bits 3-0 and bits 7-4). The operation can also be thought of as a four-bit rotate instruction.

No flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B). The instruction,

SWAP A

leaves the Accumulator holding the value 5CH (01011100B).

Bytes: 1

Cycles: 1

Encoding:

1	1	0	0	0	1	0	0

Operation: SWAP

$(A_{3-0}) \leftrightarrow (A_{7-4})$

XCH A, <byte>

Function: Exchange Accumulator with byte variable

Description: XCH loads the Accumulator with the contents of the indicated variable, at the same time writing the original Accumulator contents to the indicated variable. The source/destination operand can use register, direct, or register-indirect addressing.

Example: R0 contains the address 20H. The Accumulator holds the value 3FH (00111111B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

XCH A, @R0

will leave RAM location 20H holding the values 3FH (00111111B) and 75H (01110101B) in the accumulator.

XCH A, Rn

Bytes: 1

Cycles: 1

Encoding:

1	1	0	0	1	r	r	r

Operation: XCH

(A) ↔ (Rn)

XCH A, direct

Bytes: 2

Cycles: 1

Encoding:

1	1	0	0	0	1	0	1		direct address

Operation: XCH

(A) ↔ (direct)

XCH A, @Ri

Bytes: 1

Cycles: 1

Encoding:

1	1	0	0	0	1	1	i

Operation: XCH

(A) ↔ ((Ri))

XCHD A, @Ri

Function: Exchange Digit

Description: XCHD exchanges the low-order nibble of the Accumulator (bits 3-0), generally representing a hexadecimal or BCD digit, with that of the internal RAM location indirectly addressed by the specified register. The high-order nibbles (bits 7-4) of each register are not affected. No flags are affected.

Example: R0 contains the address 20H. The Accumulator holds the value 36H (00110110B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

XCHD A, @R0

will leave RAM location 20H holding the value 76H (01110110B) and 35H (00110101B) in the accumulator.

Bytes: 1

Cycles: 1

Encoding:

1	1	0	1	0	1	1	i

Operation: XCHD

(A_{3-0}) ↔ (Ri_{3-0})

XRL <dest-byte>, <src-byte>

Function: Logical Exclusive-OR for byte variables

Description: XRL performs the bitwise logical Exclusive-OR operation between the indicated variables, storing the results in the destination. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

(*Note*: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.)

Example: If the Accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) then the instruction,

XRL A, R0

will leave the Accumulator holding the value 69H (01101001B).

When the destination is a directly addressed byte, this instruction can complement combination of bits in any RAM location or hardware register. The pattern of bits to be complemented is then determined by a mask byte, either a constant contained in the instruction or a variable computed in the Accumulator at run-time. The instruction,

XRL P1, #00110001B

will complement bits 5,4 and 0 of output Port 1.

XRL A, Rn

Bytes: 1

Cycles: 1

Encoding:

0	1	1	0	1	r	r	r

Operation: XRL

(A) ←(A) ∀ (Rn)

XRL A, direct

Bytes: 2

Cycles: 1

Encoding:

0	1	1	0	0	1	0	1		direct address

Operation: XRL

(A) ←(A) ∀ (direct)

XRL A, @Ri

Bytes: 1

Cycles: 1

Encoding:

0	1	1	0	0	1	1	i

Operation: XRL

(A) ←(A) ∀ ((Ri))

XRL A, #data

Bytes: 2

Cycles: 1

Encoding:

0	1	1	0	0	1	0	0		immediate data

Operation: XRL

(A) ←(A) ∀ #data

XRL direct, A

Bytes: 2

Cycles: 1

Encoding:

0	1	1	0	0	0	1	0		direct address

Operation: XRL

(direct) ←(direct) ∀ (A)

XRL direct, #data

Bytes: 3

Cycles: 1

Encoding:

0	1	1	0	0	0	1	1		direct address		immediate data

Operation: XRL

(direct)← (direct) ∀ #data

17 中断系统

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。

STC15W4K32S4 系列单片机提供了 21 个中断请求源，它们分别是：

外部中断 0（INT0）、外部中断 1（INT1）、外部中断 2（INT2）、外部中断 3（INT3）、外部中断 4（INT4）；定时器 0 中断、定时器 1 中断、定时器 2 中断、定时器 3 中断、定时器 4 中断；串口 1 中断、串口 2 中断、串口 3 中断、串口 4 中断；A/D 转换中断、低压检测（LVD）中断、CCP/PWM/PCA 中断、SPI 中断、比较器中断、PWM 中断及 PWM 异常检测中断。

除外部中断 2（INT2）、外部中断 3（INT3）、外部中断 4（INT4）、定时器 2 中断、定时器 3 中断、定时器 4 中断、串口 3 中断、串口 4 中断及比较器中断固定是最低优先级中断外，其它的中断都具有 2 个中断优先级，可实现 2 级中断服务程序嵌套。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。

17.1 STC15 系列单片机的中断请求源

STC15 全系列的中断请求源的类型如下表所示（下表中√表示对应的系列有相应的中断源）

单片机型号 / 中断源类型	STC15F104W 系列	STC15F408AD 系列	STC15W204S 系列	STC15W408AS 系列	STC15W404S 系列	STC15W1K16S 系列	STC15F2K60S2 系列	STC15W4K32S4 系列
外部中断 0 (INT0)	√	√	√	√	√	√	√	√
定时器 0 中断	√	√	√	√	√	√	√	√
外部中断 1 (INT1)	√	√	√	√	√	√	√	√
定时器 1 中断					√	√	√	√
串口 1 中断		√	√	√	√	√	√	√
A/D 转换中断		√		√			√	√
低压检测(LVD)中断	√	√	√	√	√	√	√	√
CCP/PWM/PCA 中断		√		√			√	√
串口 2 中断							√	√
SPI 中断		√		√	√	√	√	√
外部中断 2 (INT2)	√	√	√	√	√	√	√	√
外部中断 3 (INT3)	√	√	√	√	√	√	√	√
定时器 2 中断	√	√	√	√	√	√	√	√
外部中断 4 (INT4)	√	√	√	√	√	√	√	√
串口 3 中断								√
串口 4 中断								√
定时器 3 中断								√
定时器 4 中断								√
比较器中断			√	√	√	√		√
PWM 中断								√
PWM 异常检测中断								√

17.1.1 STC15F104W 系列单片机的中断请求源

STC15F104W 系列单片机提供了 **8** 个中断请求源，它们分别是：外部中断 0（INT0）、定时器 0 中断、外部中断 1（INT1）、低压检测（LVD）中断、外部中断 2（INT2）、外部中断 3（INT3）、定时器 2 中断以及外部中断 4（INT4）。

除外部中断 2（INT2）、外部中断 3（INT3）、定时器 2 中断及外部中断 4（INT4）固定是最低优先级中断外，其它的中断都具有两个中断优先级。

17.1.2 STC15F408AD 系列单片机的中断请求源

STC15F408AD 系列单片机提供了 **12** 个中断请求源，它们分别是：外部中断 0（INT0）、定时器 0 中断、外部中断 1（INT1）、串口中断、A/D 转换中断、低压检测（LVD）中断、CCP/PWM/PCA 中断、SPI 中断、外部中断 2（INT2）、外部中断 3（INT3），定时器 2 中断以及外部中断 4（INT4）。

除外部中断 2（INT2）、外部中断 3（INT3）、定时器 T2 中断及外部中断 4（INT4）固定是最低优先

级中断外，其它的中断都具有两个中断优先级。

17.1.3 STC15W204S 系列单片机的中断请求源

STC15W204S 系列单片机提供了 **10** 个中断请求源，它们分别是：外部中断 0（INT0）、定时器 0 中断、外部中断 1（INT1）、串口中断、低压检测（LVD）中断、外部中断 2（INT2）、外部中断 3（INT3）、定时器 2 中断、外部中断 4（INT4）以及比较器中断。

除外部中断 2（INT2）、外部中断 3（INT3）、定时器 2 中断、外部中断 4（INT4）及比较器中断固定是最低优先级中断外，其它的中断都具有两个中断优先级。

17.1.4 STC15W408AS 系列单片机的中断请求源

STC15W408AS 系列单片机提供了 **13** 个中断请求源，它们分别是：外部中断 0（INT0）、定时器 0 中断、外部中断 1（INT1）、串口中断、A/D 转换中断、低压检测（LVD）中断、CCP/PWM/PCA 中断、SPI 中断、外部中断 2（INT2）、外部中断 3（INT3）、定时器 2 中断、外部中断 4（INT4）及比较器中断。

除外部中断 2（INT2）、外部中断 3（INT3）、定时器 2 中断、外部中断 4（INT4）及比较器中断固定是最低优先级中断外，其它的中断都具有 2 个中断优先级。

17.1.5 STC15W404S 系列单片机的中断请求源

STC15W404S 系列单片机提供了 **12** 个中断请求源，它们分别是：外部中断 0（INT0）、定时器 0 中断、外部中断 1（INT1）、定时器 1 中断、串口中断、低压检测（LVD）中断、SPI 中断、外部中断 2（INT2）、外部中断 3（INT3）、定时器 2 中断、外部中断 4（INT4）及比较器中断。

除外部中断 2（INT2）、外部中断 3（INT3）、定时器 2 中断、外部中断 4（INT4）及比较器中断固定是最低优先级中断外，其它的中断都具有两个中断优先级。

17.1.6 STC15W1K16S 系列单片机的中断请求源

STC15W1K16S 系列单片机提供了 **12** 个中断请求源，它们分别是：外部中断 0（INT0）、定时器 0 中断、外部中断 1（INT1）、定时器 1 中断、串口中断、低压检测（LVD）中断、SPI 中断、外部中断 2（INT2）、外部中断 3（INT3），定时器 2 中断、外部中断 4（INT4）及比较器中断。

除外部中断 2（INT2）、外部中断 3（INT3）、定时器 2 中断、外部中断 4（INT4）及比较器中断固定是最低优先级中断外，其它的中断都具有 2 个中断优先级，可实现 2 级中断服务程序嵌套。

17.1.7 STC15F2K60S2 系列单片机的中断请求源

STC15F2K60S2 系列单片机提供了 **14** 个中断请求源，它们分别是：外部中断 0（INT0）、定时器 0 中断、外部中断 1（INT1）、定时器 1 中断、串口 1 中断、A/D 转换中断、低压检测（LVD）中断、CCP/PWM/PCA 中断、串口 2 中断、SPI 中断、外部中断 2（INT2）、外部中断 3（INT3），定时器 2 中断以及外部中断 4（INT4）。

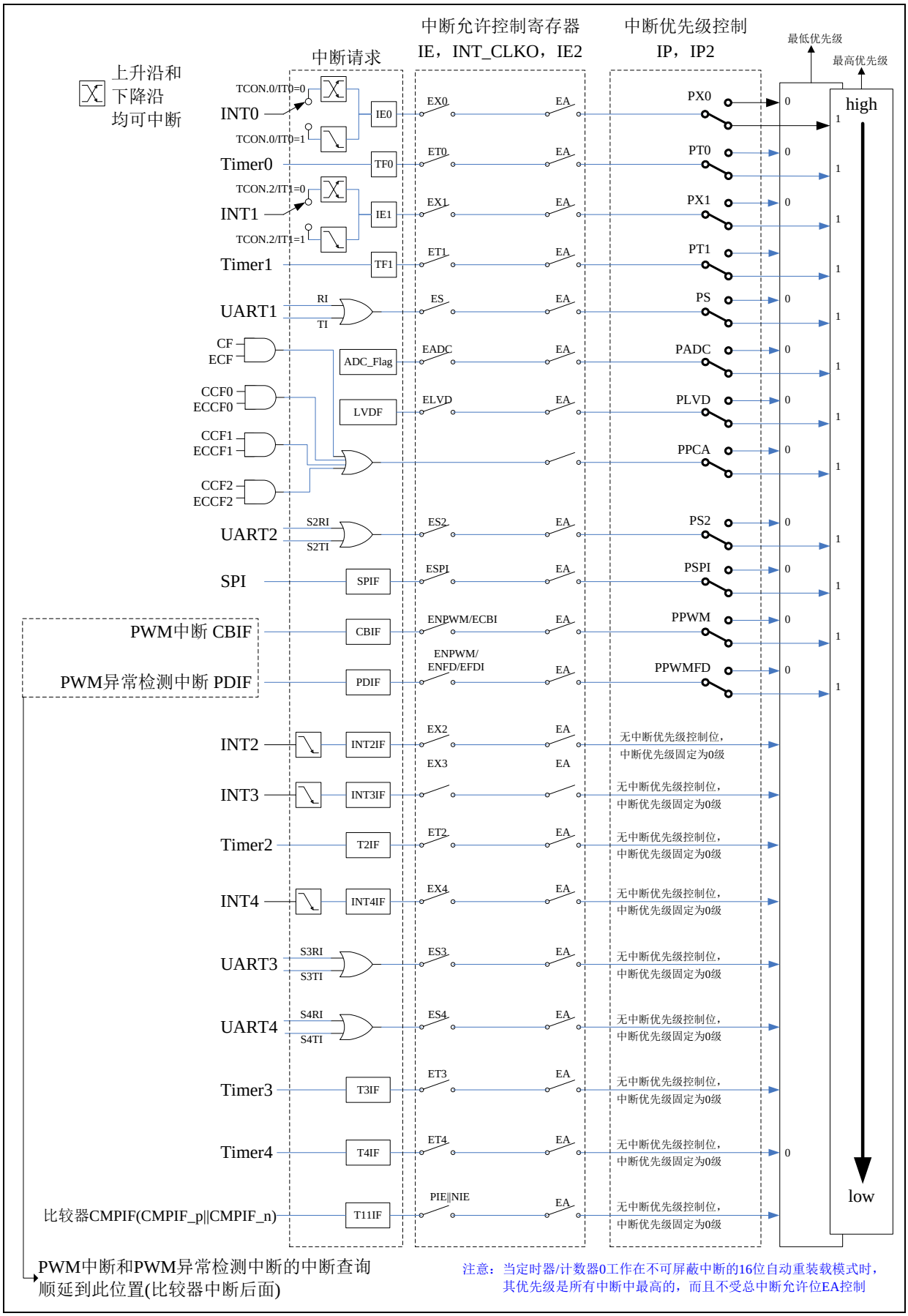
除外部中断 2（INT2）、外部中断 3（INT3）、定时器 2 中断及外部中断 4（INT4）固定是最低优先级中断外，其它的中断都具有 2 个中断优先级，可实现 2 级中断服务程序嵌套。

17.1.8 STC15W4K32S4 系列单片机的中断请求源

STC15W4K32S4 系列单片机提供了 **21** 个中断请求源，它们分别是：外部中断 0（INT0）、定时器 0 中断、外部中断 1（INT1）、定时器 1 中断、串口 1 中断、A/D 转换中断、低压检测（LVD）中断、CCP/PWM/PCA 中断、串口 2 中断、SPI 中断、外部中断 2（INT2）、外部中断 3（INT3）、定时器 2 中断、外部中断 4（INT4）、串口 3 中断、串口 4 中断、定时器 3 中断、定时器 4 中断、比较器中断、PWM 中断及 PWM 异常检测中断。

除外部中断 2（INT2）、外部中断 3（INT3）、定时器 2 中断、串口 3 中断、串口 4 中断、定时器 3 中断、定时器 4 中断及比较器中断固定是最低优先级中断外，其它的中断都具有 2 个中断优先级。

17.2 中断结构图


中断请求
中断允许控制寄存器
IE，INT_CLKO，IE2
中断优先级控制
IP，IP2
最低优先级
最高优先级
high
low
上升沿和
下降沿
均可中断
INT0
TCON.0/IT0=0
TCON.0/IT0=1
IE0
EX0
EA
PX0
Timer0
TF0
ET0
PT0
INT1
TCON.2/IT1=0
TCON.2/IT1=1
IE1
EX1
PX1
Timer1
TF1
ET1
PT1
UART1
RI
TI
ES
PS
CF
ECF
ADC_Flag
EADC
PADC
CCF0
ECCF0
LVDF
ELVD
PLVD
CCF1
ECCF1
PPCA
CCF2
ECCF2
UART2
S2RI
S2TI
ES2
PS2
SPI
SPIF
ESPI
PSPI
PWM中断 CBIF
CBIF
ENPWM/ECBI
PPWM
PWM异常检测中断 PDIF
PDIF
ENPWM/
ENFD/EFDI
PPWMFD
INT2
INT2IF
EX2
无中断优先级控制位，
中断优先级固定为0级
INT3
INT3IF
EX3
Timer2
T2IF
ET2
INT4
INT4IF
EX4
UART3
S3RI
S3TI
ES3
UART4
S4RI
S4TI
ES4
Timer3
T3IF
ET3
Timer4
T4IF
ET4
比较器CMPIF(CMPIF_p||CMPIF_n)
T11IF
PIE||NIE
PWM中断和PWM异常检测中断的中断查询
顺延到此位置(比较器中断后面)
注意：当定时器/计数器0工作在不可屏蔽中断的16位自动重装载模式时，
其优先级是所有中断中最高的，而且不受总中断允许位EA控制

外部中断 0（INT0）和外部中断 1（INT1）既可上升沿触发，又可下降沿触发。请求两个外部中断的标志位是位于寄存器 TCON 中的 IE0/TCON.1 和 IE1/TCON.3。当外部中断服务程序被响应后，中断标志位 IE0 和 IE1 会自动被清 0。TCON 寄存器中的 IT0/TCON.0 和 IT1/TCON.2 决定了外部中断 0 和 1 是上升沿触发还是下降沿触发。

- 如果 ITx = 0（x = 0,1），那么系统在 INTx（x = 0,1）脚探测到上升沿或下降沿后均可产生外部中断。
- 如果 ITx = 1（x = 0,1），那么系统在 INTx（x = 0,1）脚探测下降沿后才可产生外部中断。

外部中断 0（INT0）和外部中断 1（INT1）还可以用于将单片机从掉电模式唤醒。

定时器 0 和 1 的中断请求标志位是 TF0 和 TF1。当定时器寄存器 THx/TLx（x = 0,1）溢出时，溢出标志位 TFx（x = 0,1）会被置位，如果定时器 0/1 的中断被打开，则定时器中断发生。当单片机转去执行该定时器中断时，定时器的溢出标志位 TFx（x = 0,1）会被硬件清除。

外部中断 2（INT2）、外部中断 3（INT3）及外部中断 4（INT4）都只能下降沿触发。外部中断 2 ~ 4 的中断请求标志位被隐藏起来了，对用户不可见。当相应的中断服务程序被响应后或 EXn = 0（n = 2, 3, 4），这些中断请求标志位会立即自动地被清 0。外部中断 2（INT2）、外部中断 3（INT3）及外部中断 4（INT4）也可以用于将单片机从掉电模式唤醒。

定时器 2 的中断请求标志位被隐藏起来了，对用户不可见。当相应的中断服务程序被响应后或 ET2=0，该中断请求标志位会立即自动地被清 0。

定时器 3 和定时器 4 的中断请求标志位同样被隐藏起来了，对用户不可见。当相应的中断服务程序被响应后或 ET3=0 / ET4=0，该中断请求标志位会立即自动地被清 0。

当串行口 1 发送或接收完成时，相应的中断请求标志位 TI 或 RI 就会被置位，如果串口 1 中断被打开，向 CPU 请求中断，单片机转去执行该串口 1 中断。中断响应后，TI 或 RI 需由软件清零。

当串行口 2 发送或接收完成时，相应的中断请求标志位 S2TI 或 S2RI 就会被置位，如果串口 2 中断被打开，向 CPU 请求中断，则单片机转去执行该串口 2 中断。中断响应后，S2TI 或 S2RI 需由软件清零。

当串行口 3 发送或接收完成时，相应的中断请求标志位 S3TI 或 S3RI 就会被置位，如果串口 3 中断被打开，向 CPU 请求中断，则单片机转去执行该串口 3 中断。中断响应后，S3TI 或 S3RI 需由软件清零。

当串行口 4 发送或接收完成时，相应的中断请求标志位 S4TI 或 S4RI 就会被置位，如果串口 4 中断被打开，向 CPU 请求中断，则单片机转去执行该串口 4 中断。中断响应后，S4TI 或 S4RI 需由软件清零。

A/D 转换的中断是由 ADC_FLAG / ADC_CONTR.4 请求产生的。该位需用软件清除。

低压检测（LVD）中断是由 LVDF / PCON.5 请求产生的。该位也需用软件清除。

当同步串行口 SPI 传输完成时，SPIF / SPCTL.7 被置位，如果 SPI 中断被打开，则向 CPU 请求中断，单片机转去执行该 SPI 中断。中断响应完成后，SPIF 需通过软件向其写入“1”清零。

比较器中断标志位 CMPIF=（CMPIF_p ‖ CMPIF_n），其中 CMPIF_p 是内建的标志比较器上升沿中断的寄存器，CMPIF_n 是内建的标志比较器下降沿中断的寄存器。当 CPU 去读取 CMPIF 的数值时会读到（CMPIF_p ‖ CMPIF_n）；当 CPU 对 CMPIF 写“0”后 CMPIF_p 及 CMPIF_n 会被自动设置为“0”。

- 因此，当比较器的比较结果由 LOW 变成 HIGH 时，那么内建的标志比较器上升沿中断的寄存器 CMPIF_p 会被设置成 1，即比较器中断标志位 CMPIF 也会被设置成 1。如果比较器上升沿中断已被允许，即 PIE（CMPCR1.5）已被设置成 1，则向 CPU 请求中断，单片机转去执行该比较器上升中断；
- 同理，当比较器的比较结果由 HIGH 变成 LOW 时，那么内建的标志比较器下降沿中断的寄存

器 CMPIF_n 会被设置成 1，即比较器中断标志位 CMPIF 也会被设置成 1，如果比较器下降沿中断已被允许，即 NIE（CMPCR1.4）已被设置成 1，则向 CPU 请求中断，单片机转去执行该比较器下降中断。

中断响应完成后，比较器中断标志位 CMPIF 不会自动被消零，用户需通过软件向其写入“0”清零它。

各个中断触发行为总结如下表所示:

中断触发表

中断源	触发行为
INT0 (外部中断 0)	(IT0 = 1)：下降沿；(IT0 = 0)：上升沿和下降沿均可
Timer 0	定时器 0 溢出
INT1 (外部中断 1)	(IT1 = 1)：下降沿；(IT1 = 0)：上升沿和下降沿均可
Timer1	定时器 1 溢出
UART1	串口 1 发送或接收完成
ADC	A/D 转换完成
LVD	电源电压下降到低于 LVD 检测电压
UART2	串口 2 发送或接收完成
SPI	SPI 数据传输完成
INT2 (外部中断 2)	下降沿
INT3 (外部中断 3)	下降沿
Timer2	定时器 2 溢出
INT4 (外部中断 4)	下降沿
UART3	串口 3 发送或接收完成
UART4	串口 4 发送或接收完成
Timer3	定时器 3 溢出
Timer4	定时器 4 溢出
Comparator (比较器)	比较器比较结果由 LOW 变成 HIGH 或 HIGH 变成 LOW

17.3 中断向量入口地址/查询次序/优先级/请求标志/允许位表

中断向量入口地址/查询次序/优先级/请求标志位/允许位

中断源	中断向量地址	相同优先级内的查询次序	中断优先级设置	优先级 0（最低）	优先级 1（最高）	中断请求标志位		中断允许控制位
INT0(外部中断 0)	0003H	0 (highest)	PX0	0	1	IE0		EX0/EA
Timer 0	000BH	1	PT0	0	1	TF0		ET0/EA
INT1(外部中断 1)	0013H	2	PX1	0	1	IE1		EX1/EA
Timer1	001BH	3	PT1	0	1	TF1		ET1/EA
S1(UART1)	0023B	4	PS	0	1	RI+TI		ES/EA
ADC	002BH	5	PADC	0	1	ADC_FLAG		EADC/EA
LVD	0033H	6	PLVD	0	1	LVDF		ELVD/EA
CCP/PCA/PWM	003BH	7	PPCA	0	1	CF + CCF0 +CCF1 + CCF2		(ECF+ECCF0+ECCF1+ECCF2)/EA
S2(UART2)	0043H	8	PS2	0	1	S2RI+S2TI		ES2/EA
SPI	004BH	9	PSPI	0	1	SPIF		ESPI/EA
INT2(外部中断 2)	0053H	10	0	0				EX2/EA
INT3(外部中断 3)	005BH	11	0	0				EX3/EA
Timer 2	0063H	12	0	0				ET2/EA
System Reserved	0073H	14						
System Reserved	007BH	15						
INT4(外部中断 4)	0083H	16	0	0				EX4/EA
S3(UART3)	008BH	17	0	0		S3RI+S3TI		ES3/EA
S4(UART4)	0093H	18	0	0		S4RI+S4TI		ES4/EA
Timer 3	009BH	19	0	0				ET3/EA
Timer 4	00A3H	20	0	0				ET4/EA
Comparator(比较器)	00ABH	21	0	0		CMPIF	CMPIF_p	PIE/EA(比较器上升沿中断允许位)
							CMPIF_n	NIE/EA(比较器下降沿中断允许位)
PWM	00B3H	22	PPWM	0	1	CBIF		ENPWM/ECBI/EA
						C2IF		ENPWM / EPWM2I /EC2T2SI || EC2T1SI / EA
						C3IF		ENPWM / EPWM3I /EC3T2SI || EC3T1SI / EA
						C4IF		ENPWM / EPWM4I /EC4T2SI || EC4T1SI / EA
						C5IF		ENPWM / EPWM5I /EC5T2SI || EC5T1SI / EA
						C6IF		ENPWM / EPWM6I /EC6T2SI || EC6T1SI / EA
						C7IF		ENPWM / EPWM7I /EC7T2SI || EC7T1SI / EA
PWM 异常检测	00BBH	23(lowest)	PPWMFD	0	1	FDIF		ENPWM/ENFD/EFDI / EA

17.4 在 Keil C 中如何声明中断函数

如果使用 C 语言编程，中断查询次序号就是中断号，例如:

```
void    Int0_Routine(void)          interrupt 0;
void    Timer0_Rountine(void)       interrupt 1;
void    Int1_Routine(void)          interrupt 2;
void    Timer1_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    Timer2_Routine(void)        interrupt 12;
void    Int4_Routine(void)          interrupt 16;
void    S3_Routine(void)            interrupt 17;
void    S4_Routine(void)            interrupt 18;
void    Timer3_Routine(void)        interrupt 19;
void    Timer4_Routine(void)        interrupt 20;
void    Comparator_Routine(void)    interrupt 21;
void    PWM_Routine(void)           interrupt 22;
void    PWMFD_Routine(void)         interrupt 23;
```

17.5 中断寄存器

符号	描述	地址	位地址及符号								复位值
			MSB							LSB	
IE	Interrupt Enable	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000 0000B
IE2	Interrupt Enable 2	AFH	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000 0000B
INT_CLKO AUXR2	外部中断允许和时钟输出寄存器	8FH	-	EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	T0CLKO	x000 0000B
IP	Interrupt Priority Low	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	0000 0000B
IP2	2rd Interrupt Priority Low register	B5H	-	-	-	PX4	PPWMFD	PPWM	PSPI	PS2	xx00 0000B
TCON	Timer Control register	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000 0000B
SCON	Serial Control	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000 0000B
S2CON	Serial 2/ UART2 Control	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0100 0000B
S3CON	串口 3 控制寄存器	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000 0000B
S4CON	串口 4 控制寄存器	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000 0000B
T4T3M	T4 和 T3 的控制寄存器	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO	0000 0000B
PCON	Power Control register	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011 0000B
ADC_CONTR	ADC control register	BCH	ADC_POWER	SPEED1	SPEED0	ADC_FLAG	ADC_START	CHS2	CHS1	CHS0	0000 0000B
SPSTAT	SPI Status register	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx xxxxB
CCON	PCA Control Register	D8H	CF	CR	-	-	-	CCF2	CCF1	CCF0	00xx x000B
CMOD	PCA Mode Register	D9H	CIDL	-	-	-	-	CPS1	CPS0	ECF	0xxx 0000B
CCAPM0	PCA Module 0 Mode Register	DAH	-	ECOM0	CAPP0	CAPN0	MAT0	TOG0	PWM0	ECCF0	x000 0000B
CCAPM1	PCA Module 1 Mode Register	DBH	-	ECOM1	CAPP1	CAPN1	MAT1	TOG1	PWM1	ECCF1	x000 0000B
CCAPM2	PCA Module 2 Mode Register	DCH	-	ECOM2	CAPP2	CAPN2	MAT2	TOG2	PWM2	ECCF2	x000 0000B
AUXR	辅助寄存器	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000 0001B
CMPCR1	比较器控制寄存器 1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES	0000 0000B

STC15W4K32S4 系列新增 6 通道带死区控制的 PWM 波形发生器的中断相关特殊功能寄存器

符号	描述	地址	位址及符号								初始值
			B7	B6	B5	B4	B3	B2	B1	B0	
IP2	中断优先级控制	B5H	-	-	-	PX4	PPWMFD	PPWM	PSPI	PS2	xxx0,0000
PWMCR	PWM 控制	F5H	ENPWM	ECBI	ENC7O	ENC6O	ENC5O	ENC4O	ENC3O	ENC2O	0000,0000
PWMIF	PWM 中断标志	F6H	-	CBIF	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	x000,0000
PWMFDCR	PWM 外部异常控制	F7H	-	-	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	xx00,0000
PWM2CR	PWM2 控制	FF04H	-	-	-	-	PWM2_PS	EPWM2I	EC2T2SI	EC2T1SI	xxxx,0000
PWM3CR	PWM3 控制	FF14H	-	-	-	-	PWM3_PS	EPWM3I	EC3T2SI	EC3T1SI	xxxx,0000
PWM4CR	PWM4 控制	FF24H	-	-	-	-	PWM4_PS	EPWM4I	EC4T2SI	EC4T1SI	xxxx,0000
PWM5CR	PWM5 控制	FF34H	-	-	-	-	PWM5_PS	EPWM5I	EC5T2SI	EC5T1SI	xxxx,0000
PWM6CR	PWM6 控制	FF44H	-	-	-	-	PWM6_PS	EPWM6I	EC6T2SI	EC6T1SI	xxxx,0000
PWM7CR	PWM7 控制	FF54H	-	-	-	-	PWM7_PS	EPWM7I	EC7T2SI	EC7T1SI	xxxx,0000

上表中列出了与 STC15F2K60S2 系列单片机中断相关的所有寄存器，下面逐一地对这些寄存器进行介绍。

1.中断允许寄存器 IE、IE2 和 INT_CLKO

STC15F2K60S2 系列单片机 CPU 对中断源的开放或屏蔽，每一个中断源是否被允许中断，是由内部的中断允许寄存器 IE（IE 为特殊功能寄存器，它的字节地址为 A8H）控制的，其格式如下:

IE：中断允许寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	name	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

EA： CPU 的总中断允许控制位
EA=1，CPU 开放中断；
EA=0，CPU 屏蔽所有的中断申请。
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。

ELVD： 低压检测中断允许位
ELVD=1，允许低压检测中断；
ELVD=0，禁止低压检测中断。

EADC： A/D 转换中断允许位
EADC=1，允许 A/D 转换中断；
EADC=0，禁止 A/D 转换中断。

ES： 串行口 1 中断允许位
ES=1，允许串行口 1 中断；
ES=0，禁止串行口 1 中断。

ET1： 定时/计数器 T1 的溢出中断允许位
ET1=1，允许 T1 中断；
ET1=0，禁止 T1 中断。

EX1： 外部中断 1 中断允许位
EX1=1，允许外部中断 1 中断；
EX1=0，禁止外部中断 1 中断。

ET0： T0 的溢出中断允许位
ET0=1 允许 T0 中断；
ET0=0 禁止 T0 中断。

EX0： 外部中断 0 中断允许位
EX0=1 允许中断；
EX0=0 禁止中断。

IE2：中断允许寄存器（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE2	AFH	name	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2

ET4： 定时器 4 的中断允许位。
ET4=1，允许定时器 4 产生中断；
ET4=0，禁止定时器 4 产生中断

ET3： 定时器 3 的中断允许位。
ET3=1，允许定时器 3 产生中断；
ET3=0，禁止定时器 3 产生中断

ES4： 串行口 4 中断允许位。

ES4=1，允许串行口 4 中断；
ES4=0，禁止串行口 4 中断

ES3： 串行口 3 中断允许位。
ES3=1，允许串行口 3 中断；
ES3=0，禁止串行口 3 中断

ET2： 定时器 2 的中断允许位。
ET2=1，允许定时器 2 产生中断；
ET2=0，禁止定时器 2 产生中断

ESPI： SPI 中断允许位。
ESPI=1，允许 SPI 中断；
ESPI=0，禁止 SPI 中断

ES2： 串行口 2 中断允许位。
ES2=1，允许串行口 2 中断；
ES2=0，禁止串行口 2 中断

INT CLKO（AUXR2）是 STC15 系列单片机新增寄存器，地址是 8FH，INT CLKO（AUXR2）格式如下：

INT CLKO（AUXR2）：外部中断允许和时钟输出寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
INT_CLKO（AUXR2）	8FH	name	-	EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	T0CLKO

EX4： 外部中断 4（INT4）中断允许位，外部中断 4（INT4）只能下降沿触发。
EX4=1 允许中断；
EX4=0 禁止中断。

EX3： 外部中断 3（INT3）中断允许位，外部中断 3（INT3）也只能下降沿触发。
EX3=1 允许中断；
EX3=0 禁止中断。

EX2： 外部中断 2（INT2）中断允许位，外部中断 2（INT2）同样只能下降沿触发。
EX2=1 允许中断；
EX2=0 禁止中断。

MCKO_S2，T2CLKO，T1CLKO，T0CLKO 与中断无关，在此不作介绍。

STC15 系列单片机复位以后，IE、IE2 和 INT_CLKO（AUXR2）被清 0，由用户程序置“1”或清“0”IE、IE2 和 INT_CLKO（AUXR2）的相应位，实现允许或禁止各中断源的中断申请，若使某一个中断源允许中断必须同时使 CPU 开放中断。

- 更新 IE 的内容可由位操作指令来实现（SETB BIT; CLR BIT），也可用字节操作指令实现（即 MOV IE, #DATA; ANL IE, #DATA; ORL IE, #DATA; MOV IE, A 等）。
- 更新 IE2 和 INT_CLKO（不可位寻址）的内容只可用字节操作指令（即 MOV IE2, #DATA 或 MOV INT_CLKO, #DATA）来解决。

2.中断优先级控制寄存器 IP、IP2

传统 8051 单片机具有两个中断优先级，即高优先级和低优先级，可以实现两级中断嵌套。STC15 系列单片机通过设置特殊功能寄存器（IP 和 IP2）中的相应位，可将部分中断设有 2 个中断优先级，除

外部中断 2（INT2）、外部中断 3（INT3）、定时器 2、定时器 3、定时器 4、串行口 3 及串行口 4 外，所有中断请求源可编程为 2 个优先级中断。一个正在执行的低优先级中断能被高优先级中断所中断，但不能被另一个低优先级中断所中断，一直执行到结束，遇到返回指令 RETI，返回主程序后再执行一条指令才能响应新的中断申请。以上所述可归纳为下面两条基本规则：

1.低优先级中断可被高优先级中断所中断，反之不能。
2.任何一种中断（不管是高级还是低级），一旦得到响应，不会再被它的同级中断所中断

STC15 系列单片机的片内各优先级控制寄存器的格式如下：

IP：中断优先级控制寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IP	B8H	name	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0

PPCA： PCA 中断优先级控制位。
当 PPCA=0 时，PCA 中断为最低优先级中断（优先级 0）
当 PPCA=1 时，PCA 中断为最高优先级中断（优先级 1）

PLVD： 低压检测中断优先级控制位。
当 PLVD=0 时，低压检测中断为最低优先级中断（优先级 0）
当 PLVD=1 时，低压检测中断为最高优先级中断（优先级 1）

PADC： A/D 转换中断优先级控制位。
当 PADC=0 时，A/D 转换中断为最低优先级中断（优先级 0）
当 PADC=1 时，A/D 转换中断为最高优先级中断（优先级 1）

PS： 串口 1 中断优先级控制位。
当 PS-0 时，串口 1 中断为最低优先级中断（优先级 0）
当 PS=1 时，串口 1 中断为最高优先级中断（优先级 1）

PT1： 定时器 1 中断优先级控制位。
当 PT1=0 时，定时器 1 中断为最低优先级中断（优先级 0）
当 PT1=1 时，定时器 1 中断为最高优先级中断（优先级 1）

PX1： 外部中断 1 优先级控制位。
当 PX1=0 时，外部中断 1 为最低优先级中断（优先级 0）
当 PX1=1 时，外部中断 1 为最高优先级中断（优先级 1）

PT0： 定时器 0 中断优先级控制位。
当 PT0=0 时，定时器 0 中断为最低优先级中断（优先级 0）
当 PT0=1 时，定时器 0 中断为最高优先级中断（优先级 1）

PX0： 外部中断 0 优先级控制位。
当 PX0=0 时，外部中断 0 为最低优先级中断（优先级 0）
当 PX0=1 时，外部中断 0 为最低优先级中断（优先级 1）

IP2：中断优先级控制寄存器 （不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IP2	B5H	name	-	-	-	PX4	PPWMFD	PPWM	PSPI	PS2

PX4： 外部中断 4（INT4）优先级控制位。
当 PX4=0 时，外部中断 4（INT4）为最低优先级中断（优先级 0）

当 PX4=1 时，外部中断 4（INT4）为最高优先级中断（优先级 1）

PPWMFD：PWM 异常检测中断优先级控制位。

当 PPWMFD=0 时，PWM 异常检测中断为最低优先级中断（优先级 0）

当 PPWMFD=1 时，PWM 异常检测中断为最高优先级中断（优先级 1）

PPWM：PWM 中断优先级控制位。

当 PPWM=0 时，PWM 中断为最低优先级中断（优先级 0）

当 PPWM=1 时，PWM 中断为最高优先级中断（优先级 1）

PSPI： SPI 中断优先级控制位。

当 PSPI=0 时，SPI 中断为最低优先级中断（优先级 0）

当 PSPI=1 时，SPI 中断为最高优先级中断（优先级 1）

PS2： 串口 2 中断优先级控制位。

当 PS2-0 时，串口 2 中断为最低优先级中断（优先级 0）

当 PS2=1 时，串口 2 中断为最高优先级中断（优先级 1）

中断优先级控制寄存器 IP 和 IP2 的各位都由可用户程序置“1”和清“0”。但 IP 寄存器可位操作，所以可用位操作指令或字节操作指令更新 IP 的内容。而 IP2 寄存器的内容只能用字节操作指令来更新。STC15 系列单片机复位后 IP 和 IP2 均为 00H，各个中断源均为低优先级中断。

3.定时器/计数器控制寄存器 TCON

TCON 为定时器/计数器 T0、T1 的控制寄存器，同时也锁存 T0、T1 溢出中断源和外部请求中断源等，TCON 格式如下：

TCON：定时器/计数器中断控制寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
TCON	88H	name	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，由硬件置“1”TF1，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 1 的运行控制位。

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 0 的运行控制位。

IE1： 外部中断 1（INT1/P3.3）中断请求标志。

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 1 中断源类型选择位。

IT1=0，INT1/P3.3 引脚上的上升沿或下降沿信号均可触发外部中断 1。

IT1=1，外部中断 1 为下降沿触发方式。

IE0： 外部中断 0（INT0/P3.2）中断请求标志。

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。

IT0： 外部中断 0 中断源类型选择位。

IT0=0，INT0/P3.2 引脚上的上升沿或下降沿均可触发外部中断 0。

IT0=1，外部中断 0 为下降沿触发方式。

4.串行口 1 控制寄存器 SCON

SCON 为串行口控制寄存器，SCON 格式如下：

SCON：串行口控制寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
SCON	98H	name	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI

RI： 串行口 1 接收中断标志。
若串行口 1 允许接收且以方式 0 工作，则每当接收到第 8 位数据时置 1；
若以方式 1、2、3 工作且 SM2=0 时，则每当接收到停止位的中间时置 1；
当串行口以方式 2 或方式 3 工作且 SM2=1 时，则仅当接收到的第 9 位数据 RB8 为 1 后，同时还要接收到停止位的中间时置 1。
RI 为 1 表示串行口 1 正向 CPU 申请中断（接收中断），RI 必须由用户的中断服务程序清零。

TI： 串行口 1 发送中断标志。
串行口 1 以方式 0 发送时，每当发送完 8 位数据，由硬件置 1；
若以方式 1、方式 2 或方式 3 发送时，在发送停止位的开始时置 1。
TI=1 表示串行口 1 正在向 CPI 申请中断（发送中断）。
值得注意的是，CPU 响应发送中断请求，转向执行中断服务程序时并不将 TI 清零，TI 必须由用户在中断服务程序中清零。

SCON 寄存器的其他位与中断无关，在此不作介绍。

5.串行口 2 控制寄存器 S2CON

S2CON 为串行口 2 控制寄存器，S2CON 格式如下：

S2CON：串行口 2 控制寄存器（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
S2CON	9AH	name	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI

S2RI： 串行口 2 接收中断标志。
若串行口 2 允许接收且以方式 0 工作，则每当接收到第 8 位数据时置 1；
若以方式 1、2、3 工作且 S2SM2=0 时，则每当接收到停止位的中间时置 1；
当串行口 2 以方式 2 或方式 3 工作且 S2SM2=1 时，则仅当接收到的第 9 位数据 S2RB8 为 1 后，同时还要接收到停止位的中间时置 1。
S2RI 为 1 表示串行口 2 正向 CPU 申请中断(接收中断)，S2RI 必须由用户的中断服务程序清零。

S2TI： 串行口 2 发送中断标志。
串行口 2 以方式 0 发送时，每当发送完 8 位数据，由硬件置 1；
若以方式 1、方式 2 或方式 3 发送时，在发送停止位的开始时置 1。
S2TI=1 表示串行口 2 正在向 CPU 申请中断(发送中断)。
值得注意的是，CPU 响应发送中断请求，转向执行中断服务程序时并不将 S2TI 清零，S2TI 必须由用户在中断服务程序中清零。

S2CON 寄存器的其他位与中断无关，在此不作介绍。

6.串行口 3 控制寄存器 S3CON

S3CON 为串行口 3 控制寄存器，S3CON 格式如下：

S3CON：串行口 3 控制寄存器（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
S3CON	ACH	name	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI

S3RI： 串行口 3 接收中断标志。
若串行口 3 允许接收且以方式 0 工作，则每当接收到第 8 位数据时置 1；
若以方式 1、2、3 工作且 S3SM2=0 时，则每当接收到停止位的中间时置 1；
当串行口 3 以方式 2 或方式 3 工作且 S3SM2=1 时，则仅当接收到的第 9 位数据 S3RB8 为 1 后，同时还要接收到停止位的中间时置 1。
S3RI 为 1 表示串行口 3 正向 CPU 申请中断(接收中断)，S3RI 必须由用户的中断服务程序清零。

S3TI： 串行口 3 发送中断标志。
串行口 3 以方式 0 发送时，每当发送完 8 位数据，由硬件置 1；
若以方式 1、方式 2 或方式 3 发送时，在发送停止位的开始时置 1。
S3TI=1 表示串行口 3 正在向 CPU 申请中断（发送中断）。
值得注意的是，CPU 响应发送中断请求，转向执行中断服务程序时并不将 S3TI 清零，S3TI 必须由用户在中断服务程序中清零。

S3CON 寄存器的其他位与中断无关，在此不作介绍。

7.串行口 4 控制寄存器 S4CON

S4CON 为串行口 4 控制寄存器，S4CON 格式如下：

S4CON：串行口 4 控制寄存器（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
S4CON	84H	name	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI

S4RI： 串行口 4 接收中断标志。
若串行口 4 允许接收且以方式 0 工作，则每当接收到第 8 位数据时置 1；
若以方式 1、2、3 工作且 S4SM2=0 时，则每当接收到停止位的中间时置 1；
当串行口 4 以方式 2 或方式 3 工作且 S4SM2=1 时，则仅当接收到的第 9 位数据 S4RB8 为 1 后，同时还要接收到停止位的中间时置 1。
S4RI 为 1 表示串行口 4 正向 CPU 申请中断(接收中断)，S4RI 必须由用户的中断服务程序清零。

S4TI： 串行口 4 发送中断标志。
串行口 4 以方式 0 发送时，每当发送完 8 位数据，由硬件置 1；
若以方式 1、方式 2 或方式 3 发送时，在发送停止位的开始时置 1。
S4TI=1 表示串行口 4 正在向 CPU 申请中断（发送中断）。
值得注意的是，CPU 响应发送中断请求，转向执行中断服务程序时并不将 S4TI 清零，S4TI 必须由用户在中断服务程序中清零。

S4CON 寄存器的其他位与中断无关，在此不作介绍。

8. 低压检测中断相关寄存器：电源控制寄存器 PCON

PCON 为电源控制寄存器，PCON 格式如下：

PCON：电源控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
PCON	87H	name	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL

LVDF： 低压检测标志位，同时也是低压检测中断请求标志位。
在正常工作和空闲工作状态时，如果内部工作电压 Vcc 低于低压检测门槛电压，该位自动置 1，与低压检测中断是否被允许无关。即在内部工作电压 Vcc 低于低压检测门槛电压时，不管有没有允许低压检测中断，该位都自动为 1。该位要用软件清 0，清 0 后，如内部工作电压 Vcc 继续低于低压检测门槛电压，该位又被自动设置为 1。

在进入掉电工作状态前，如果低压检测电路未被允许可产生中断，则在进入掉电模式后，该低压检测电路不工作以降低功耗。如果被允许可产生低压检测中断，则在进入掉电模式后，该低压检测电路继续工作，在内部工作电压 Vcc 低于低压检测门槛电压后，产生低压检测中断，可将 MCU 从掉电状态唤醒。

电源控制寄存器 PCON 中的其他位与低压检测中断无关，在此不作介绍。

在中断允许寄存器 IE 中，低压检测中断相应的允许位是 ELVD/IE.6

IE：中断允许寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	name	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

EA： CPU 的总中断允许控制位

EA=1，CPU 开放中断；

EA=0，CPU 屏蔽所有的中断申请。

EA 的作用是使中断允许形成两级控制。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。

ELVD： 低压检测中断允许位

ELVD=1，允许低压检测中断；

ELVD=0，禁止低压检测中断。

9.A/D 转换控制寄存器 ADC_CONTR

ADC_CONTR 为 A/D 转换控制寄存器，ADC_CONTR 格式如下：

ADC_CONTR：A/D 转换控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
ADC_CONTR	BCH	name	ADC_POWER	SPEED1	SPEED0	ADC_FLAG	ADC_START	CHS2	CHS1	CHS0

ADC_POWER： ADC 电源控制位。

当 ADC_POWER=0 时，关闭 ADC 电源；

当 ADC_PWOER=1 时，打开 ADC 电源。

ADC_FLAG： ADC 转换结束标志位，可用于请求 A/D 转换的中断。

当 A/D 转换完成后，ADC_FLAG=1，要用软件清 0。不管是 A/D 转换完成后由该位申请产生中断，还是由软件查询该标志位 A/D 转换是否结束，当 A/D 转换完成后，ADC_FLAG=1，一定要软件清 0。

ADC_START： ADC 转换启动控制位。

设置为“1”时，开始转换，转换结束后为 0。

A/D 转换控制寄存器 ADC——CONTR 中的其他位与中断无关，在此不作介绍。

在中断允许寄存器 IE 中，A/D 转换器的中断允许位是 EADC / IE.5

IE：中断允许寄存器 （可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	name	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

EA： CPU 的总中断允许控制位。

EA=1，CPU 开放中断；

EA=0，CPU 屏蔽所有的中断申请。

EA 的作用是使中断允许形成两级控制。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。

EADC： A/D 转换中断允许位。
EADC=1，允许 A/D 转换中断；
EADC=0，禁止 A/D 转换中断。

10.比较器控制寄存器 1：CMPCR1

比较器控制寄存器 1 的格式如下：

CMPCR1：比较器控制寄存器 1

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR1	E6H	name	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES

CMPEN： 比较器模块使能位
CMPEN=1，使能比较器模块；
CMPEN=0，禁用比较器模块，比较器的电源关闭。

CMPIF： 比较器中断标志位（Interrupt Flag）
在 CMPEN 为 1 的情况下：

- 当比较器的比较结果由 LOW 变成 HIGH 时，若是 PIE 被设置成 1，那么内建的某一个叫做 CMPIF_p 的寄存器会被设置成 1；
- 当比较器的比较结果由 HIGH 变成 LOW 时，若是 NIE 被设置成 1，那么内建的某一个叫做 CMPIF_n 的寄存器会被设置成 1；

当 CPU 去读取 CMPIF 的数值时，会读到（CMPIF_P || CMPIF_n）；
当 CPU 对 CMPIF 写 0 后，CMPIF_p 以及 CMPIF_n 都会被清除为 0。
而中断产生的条件是[（EA == 1）&&（（（PIE == 1）&&（CMPIF_p == 1）） || （（NIE == 1）&&（CMPIF_n == 1））]
CPU 接受中断后，并不会自动清除此 CMPIF 标志，用户必须用软件写”0”去清除它。

PIE： 比较器上升沿中断使能位（Pos-edge Interrupt Enabling）
PIE=1，使能比较器由 LOW 变 HIGH 的事件设定 CMPIF_p 产生中断；
PIE=0，禁用比较器由 LOW 变 HIGH 的事件设定 CMPIF_p 产生中断

NIE： 比较器下降沿中断使能位（Neg-edge Interrupt Enabling）
NIE=1，使能比较器由 HIGH 变 LOW 的事件设定 CMPIF_n 产生中断；
NIE=0，禁用比较器由 HIGH 变 LOW 的事件设定 CMPIF_n 产生中断。

比较器控制寄存器 1--CMPCR1 的其他位与中断无关，在此不作介绍。

11. PWM 控制寄存器：PWMCR

PWM 控制寄存器的格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWMCR	F5H	name	ENPWM	ECBI	ENC70	ENC60	ENC50	ENC40	ENC30	ENC20	0000, 0000B

ECBI： PWM 计数器归零中断使能位
0：关闭 PWM 计数器归零中断（CBIF 依然会被硬件置位）
1：使能 PWM 计数器归零中断

12. PWM 中断标志寄存器：PWMIF

PWM 中断标志寄存器的格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWMIF	F6H	name	-	CBIF	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	x000, 0000B

CBIF： PWM 计数器归零中断标志位
当 PWM 计数器归零时，硬件自动将此位置 1。当 ECBI == 1 时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

C7IF： 第 7 通道的 PWM 中断标志位
可设置在翻转点 1 和翻转点 2 触发 C7IF（详见 EC7T1SI 和 EC7T2SI）。当 PWM 发生翻转时，硬件自动将此位置 1。当 EPWM7I == 1 时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

C6IF： 第 6 通道的 PWM 中断标志位
可设置在翻转点 1 和翻转点 2 触发 C6IF（详见 EC6T1SI 和 EC6T2SI）。当 PWM 发生翻转时，硬件自动将此位置 1。当 EPWM6I == 1 时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

C5IF： 第 5 通道的 PWM 中断标志位
可设置在翻转点 1 和翻转点 2 触发 C5IF（详见 EC5T1SI 和 EC5T2SI）。当 PWM 发生翻转时，硬件自动将此位置 1。当 EPWM5I == 1 时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

C4IF： 第 4 通道的 PWM 中断标志位
可设置在翻转点 1 和翻转点 2 触发 C4IF（详见 EC4T1SI 和 EC4T2SI）。当 PWM 发生翻转时，硬件自动将此位置 1。当 EPWM4I == 1 时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

C3IF： 第 3 通道的 PWM 中断标志位
可设置在翻转点 1 和翻转点 2 触发 C3IF（详见 EC3T1SI 和 EC3T2SI）。当 PWM 发生翻转时，硬件自动将此位置 1。当 EPWM3I == 1 时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

C2IF： 第 2 通道的 PWM 中断标志位
可设置在翻转点 1 和翻转点 2 触发 C2IF（详见 EC2T1SI 和 EC2T2SI）。当 PWM 发生翻转时，硬件自动将此位置 1。当 EPWM2I == 1 时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

13. PWM 外部异常控制寄存器：PWMFDCR

PWM 外部异常控制寄存器的格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWMFDCR	F7H	name	-	-	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	xx00, 0000B

EFDI： PWM 异常检测中断使能位
0：关闭 PWM 异常检测中断（FDIF 依然会被硬件置位）
1：使能 PWM 异常检测中断

FDIF： PWM 异常检测中断标志位
当发生 PWM 异常（比较器正极 P5.5/CMP+的电平比比较器负极 P5.4/CMP-的电平高或比较器正极 P5.5/CMP+的电平比内部参考电压源 1.28V 高或者 P2.4 的电平为高）时，硬件自动将此位置 1。当 EFDI == 1 时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

14. PWM2 的控制寄存器：PWM2CR

PWM2 的控制寄存器的格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM2CR	FF04H（XSFR）	name	-	-	-	-	PWM2_PS	EPWM2I	EC2T2SI	EC2T1SI	xxxx, 0000B

EPWM2I：PWM2 中断使能控制位

0：关闭 PWM2 中断

1：使能 PWM2 中断，当 C2IF 被硬件置 1 时，程序将跳转到相应中断入口执行中断服务程序。

EC2T2SI：PWM2 的 T2 匹配发生波形翻转时的中断控制位

0：关闭 T2 翻转时中断

1：使能 T2 翻转时中断，当 PWM 波形发生器内部计数值与 T2 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C2IF 置 1，此时若 EPWM2I == 1，则程序将跳转到相应中断入口执行中断服务程序。

EC2T1SI：PWM2 的 T1 匹配发生波形翻转时的中断控制位

0：关闭 T1 翻转时中断

1：使能 T1 翻转时中断，当 PWM 波形发生器内部计数值与 T1 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C2IF 置 1，此时若 EPWM2I == 1，则程序将跳转到相应中断入口执行中断服务程序。

15. PWM3 的控制寄存器：PWM3CR

PWM3 的控制寄存器的格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM3CR	FF14H（XSFR）	name	-	-	-	-	PWM3_PS	EPWM3I	EC3T2SI	EC3T1SI	xxxx, 0000B

EPWM3I：PWM3 中断使能控制位

0：关闭 PWM3 中断

1：使能 PWM3 中断，当 C3IF 被硬件置 1 时，程序将跳转到相应中断入口执行中断服务程序。

EC3T2SI：PWM3 的 T2 匹配发生波形翻转时的中断控制位

0：关闭 T2 翻转时中断

1：使能 T2 翻转时中断，当 PWM 波形发生器内部计数值与 T2 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C3IF 置 1，此时若 EPWM3I == 1，则程序将跳转到相应中断入口执行中断服务程序。

EC3T1SI：PWM3 的 T1 匹配发生波形翻转时的中断控制位

0：关闭 T1 翻转时中断

1：使能 T1 翻转时中断，当 PWM 波形发生器内部计数值与 T1 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C3IF 置 1，此时若 EPWM3I == 1，则程序将跳转到相应中断入口执行中断服务程序。

16.PWM4 的控制寄存器：PWM4CR

PWM4 的控制寄存器的格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM4CR	FF24H（XSFR）	name	-	-	-	-	PWM4_PS	EPWM4I	EC4T2SI	EC4T1SI	xxxx, 0000B

EPWM4I：PWM4 中断使能控制位

0：关闭 PWM4 中断

1：使能 PWM4 中断，当 C4IF 被硬件置 1 时，程序将跳转到相应中断入口执行中断服务程序。

EC4T2SI：PWM4 的 T2 匹配发生波形翻转时的中断控制位

0：关闭 T2 翻转时中断

1：使能 T2 翻转时中断，当 PWM 波形发生器内部计数值与 T2 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C4IF 置 1，此时若 EPWM4I == 1，则程序将跳转到相应中断入口执行中断服务程序。

EC4T1SI：PWM4 的 T1 匹配发生波形翻转时的中断控制位

0：关闭 T1 翻转时中断

1：使能 T1 翻转时中断，当 PWM 波形发生器内部计数值与 T1 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C4IF 置 1，此时若 EPWM4I == 1，则程序将跳转到相应中断入口执行中断服务程序。

17.PWM5 的控制寄存器：PWMSCR

PWM5 的控制寄存器的格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM5CR	FF34H（XSFR）	name	-	-	-	-	PWM5_PS	EPWM5I	EC5T2SI	EC5T1SI	xxxx，0000B

EPWM5I：PWM5 中断使能控制位

0：关闭 PWM5 中断

1：使能 PWM5 中断，当 C5IF 被硬件置 1 时，程序将跳转到相应中断入口执行中断服务程序。

EC5T2SI：PWM5 的 T2 匹配发生波形翻转时的中断控制位

0：关闭 T2 翻转时中断

1：使能 T2 翻转时中断，当 PWM 波形发生器内部计数值与 T2 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C5IF 置 1，此时若 EPWM5I == 1，则程序将跳转到相应中断入口执行中断服务程序。

EC5T1SI：PWM5 的 T1 匹配发生波形翻转时的中断控制位

0：关闭 T1 翻转时中断

1：使能 T1 翻转时中断，当 PWM 波形发生器内部计数值与 T1 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C5IF 置 1，此时若 EPWM5I == 1，则程序将跳转到相应中断入口执行中断服务程序。

18.PWM6 的控制寄存器：PWM6CR

PWM6 的控制寄存器的格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM6CR	FF44H（XSFR）	name	-	-	-	-	PWM6_PS	EPWM6I	EC6T2SI	EC6T1SI	xxxx, 0000B

EPWM6I：PWM6 中断使能控制位

0：关闭 PWM6 中断

1：使能 PWM6 中断，当 C6IF 被硬件置 1 时，程序将跳转到相应中断入口执行中断服务程序。

EC6T2SI：PWM6 的 T2 匹配发生波形翻转时的中断控制位

0：关闭 T2 翻转时中断

1：使能 T2 翻转时中断，当 PWM 波形发生器内部计数值与 T2 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C6IF 置 1，此时若 EPWM6I == 1，则程序将跳转到相应中断入口执行中断服务程序。

EC6T1SI：PWM6 的 T1 匹配发生波形翻转时的中断控制位

0：关闭 T1 翻转时中断

1：使能 T1 翻转时中断，当 PWM 波形发生器内部计数值与 T1 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C6IF 置 1，此时若 EPWM6I == 1，则程序将跳转到相应中断入口执行中断服务程序。

19.PWM7 的控制寄存器：PWM7CR

PWM7 的控制寄存器的格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM7CR	FF54H（XSFR）	name	-	-	-	-	PWM7_PS	EPWM7I	EC7T2SI	EC7T1SI	xxxx，0000B

EPWM7I：PWM7 中断使能控制位

0：关闭 PWM7 中断

1：使能 PWM7 中断，当 C7IF 被硬件置 1 时，程序将跳转到相应中断入口执行中断服务程序。

EC7T2SI：PWM7 的 T2 匹配发生波形翻转时的中断控制位

0：关闭 T2 翻转时中断

1：使能 T2 翻转时中断，当 PWM 波形发生器内部计数值与 T2 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C7IF 置 1，此时若 EPWM7I == 1，则程序将跳转到相应中断入口执行中断服务程序。

EC7T1SI：PWM7 的 T1 匹配发生波形翻转时的中断控制位

0：关闭 T1 翻转时中断

1：使能 T1 翻转时中断，当 PWM 波形发生器内部计数值与 T1 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C7IF 置 1，此时若 EPWM7I == 1，则程序将跳转到相应中断入口执行中断服务程序。

17.6 中断优先级

除外部中断 2（INT2）、外部中断 3（INT3）、定时器 T2 中断、外部中断 4（INT4）、串口 3 中断、串口 4 中断、定时器 T3 中断、定时器 T4 中断及比较器中断外，STC15W4K32S4 系列单片机的所有的中断都具有 2 个中断优先级。一个正在执行的低优先级中断能被高优先级中断所中断，但不能被另一个低优先级中断所中断，一直执行到结束，遇到返回指令 RETI，返回主程序后再执行一条指令才能响应新的中断申请。以上所述可归纳为下面两条基本规则:

1.低优先级中断可被高优先级中断所中断，反之不能。

2.任何一种中断（不管是高级还是低级），一旦得到响应，不能被它的同级中断所中断。

当同时收到几个同一优先级的中断要求时，哪一个要求得到服务，取决于内部的查询次序。这相当于在每个优先级内，还同时存在另一个辅助优先级结构，STC15W4K32S4 系列单片机各中断优先查询次序如下:

	中断源	查询次序
0	INT0	（highest）
1	Timer 0	
2	INT1	
3	Timer 1	
4	UART1	
5	ADC interrupt	
6	LVD	
7	PCA	
8	UART2	
9	SPI	
10	INT2	
11	INT3	
12	Timer 2	
13		
14		
15		
16	INT4	
17	UART3	
18	UART4	
19	Timer 3	
20	Timer 4	
21	Comparator	
22	PWM	
23	PWMFD	（lowest）

【注意】: 当定时器/计数器 0 工作在不可屏蔽中断的 16 位自动重装载模式时，如果此时定时器/计数器 0 中断被允许了（只需置位 ET0 即可，不需置位 EA，工作在不可屏蔽中断的 16 位自动重装载模式下的定时器/计数器 0 中断与 EA 无关），则该中断优先级是所有中断中最高的，任何一个中断都不能打断它，而且该中断被打开后它不仅不受 EA 控制，也不再受 ET0 控制了。

如果使用 C 语言编程，中断查询次序号就是中断号，例如：

```
void    Int0_Routine（void）        interrupt 0;
void    Timer0_Rountine（void）     interrupt 1;
void    Int1_Routine（void）        interrupt 2;
void    Timer1_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    Timer2_Routine（void）      interrupt 12;
void    Int4_Routine（void）        interrupt 16;
void    S3_Routine（void）          interrupt 17;
void    S4_Routine（void）          interrupt 18;
void    Timer3_Routine（void）      interrupt 19;
void    Timer4_Routine（void）      interrupt 20;
void    Comparator_Routine（void）  interrupt 21;
void    PWM_Routine（void）         interrupt 22;
void    PWMFD_Routine（void）       interrupt 23;
```

17.7 中断处理

当某中断产生而且被 CPU 响应，主程序被中断，接下来将执行如下操作:

1.当前正被执行的指令全部执行完毕；
2.PC 值被压入栈；
3.现场保护；
4.阻止同级别其他中断；
5.将中断向量地址装载到程序计数器 PC；
6.执行相应的中断服务程序。

中断服务程序 ISR 完成和该中断相应的一些操作。中断服务程序 ISR 以 RETI（中断返回）指令结束，将 PC 值从栈中取回，并恢复原来的中断设置，之后从主程序的断点处继续执行。

当某中断被响应时，被装载到程序计数器 PC 中的数值称为中断向量，是该中断源相对应的中断服务程序的起始地址。各中断源服务程序的入口地址（即中断向量）为：

中断源	中断向量
External Interrupt 0	0003H
Timer 0	000BH
External Interrupt 1	0013H
Timer 1	001BH
S1（UART1）	0023H
ADC interrupt	002BH
LVD	0033H
PCA	003BH
S2（UART2）	0043H
SPI	004BH

中断源	中断向量
External Interrupt 2	0053H
External Interrupt 3	005BH
Timer 2	0063H
/	006BH
/	0073H
/	007BH
External Interrupt 4	0083H
S3（UART3）	008BH
S4（UART4）	0093H
Timer 3	009BH
Timer 4	00A3H
Comparator	00ABH
PWM	00B3H
PWMFD	00BBH

当“转去执行中断”时，引起外部中断 INT0/INT1/INT2/INT3/INT4 请求标志位和定时器/计数器 0、定时器/计数器 1 的中断请求标志位将被硬件自动清零，其它中断的中断请求标志位需软件清“0”。由于中断向量入口地址位于程序存储器的开始部分，所以主程序的第 1 条指令通常为跳转指令，越过中断向量区（LJMP START）。

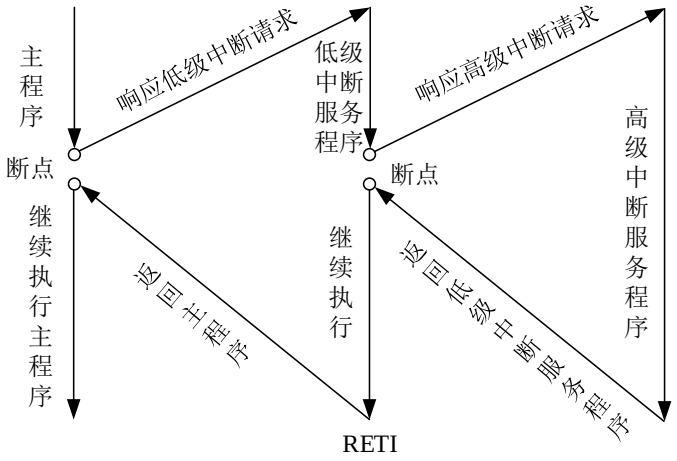
【注意】：不能用 RET 指令代替 RETI 指令

RET 指令虽然也能控制 PC 返回到原来中断的地方，但 RET 指令没有清零中断优先级状态触发器的功能，中断控制系统会认为中断仍在进行，其后果是与此同级或低级的中断请求将不被响应。

若用户在中断服务程序中进行了入栈操作，则在 RETI 指令执行前应进行相应的出栈操作，即在中断服务程序中 PUSH 指令与 POP 指令必须成对使用，否则不能正确返回断点。

17.8 中断嵌套

中断优先级高的中断请求可以中断 CPU 正在处理的优先级更低的中断服务程序，待完成了中断优先权高的中断服务程序后，再继续被打断的更低的中断服务程序。这就是中断嵌套。下图描述了主程序和中断服务程序运行示意图。



17.9 外部中断

外部中断 0（INT0）和外部中断 1（INT1）触发有两种触发方式，上升沿或下降沿均可触发方式和仅下降沿触发方式。

TCON 寄存器中的 IT0/TCON.0 和 IT1/TCON.2 决定了外部中断 0 和 1 是上升沿和下降沿均可触发还是仅下降沿触发。如果 ITx=0（x=0,1），那么系统在 INTx（x=0,1）脚探测到上升沿或下降沿后均可产生外部中断。如果 ITx=1（x=0,1），那么系统在 INTx（x=0,1）脚探测下降沿后才可产生外部中断。外部中断 0（INT0）和外部中断 1（INT1）还可以用于将单片机从掉电模式唤醒。

外部中断 2（INT2）、外部中断 3（INT3）及外部中断 4（INT4）都只能下降沿触发。外部中断 2~4 的中断请求标志位被隐藏起来了，对用户不可见，故也无需用户清“0”。当相应的中断服务程序被响应后或中断允许位 EXn（n=2,3,4）被清零后，这些中断请求标志位会立即自动地被清 0。这些中断请求标志位也可以通过软件禁止相应的中断允许控制位将其清“0”（特殊应用）。外部中断 2（INT2）、外部中断 3（INT3）及外部中断 4（INT4）也可以用于将单片机从掉电模式唤醒。

由于系统每个时钟对外部中断引脚采样 1 次，所以为了确保被检测到，输入信号应该至少维持 2 个时钟。如果外部中断是仅下降沿触发，要求必须在相应的引脚维持高电平至少 1 个时钟，而且低电平也要持续至少一个时钟，才能确保该下降沿被 CPU 检测到。同样，如果外部中断是上升沿、下降沿均可触发，则要求必须在相应的引脚维持低电平或高电平至少 1 个时钟，而且高电平或低电平也要持续至少一个时钟，这样才能确保 CPU 能够检测到该上升沿或下降沿。

STC15 系列单片机的 3 路 CCP/PCA/PWM 还再可实现 3 个外部中断（支持上升沿/下降沿中断）

17.10 中断的测试程序(C 和汇编)

17.10.1 外部中断 0(INT0)的测试程序

17.10.1.1 外部中断 INT0(上升沿+下降沿)的测试程序(C 和汇编)

1.C 程序:

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 INT0 中断举例-----------------------------------------------------*/
/*----  在 Keil C  开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------------*/
//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"
#include "intrins.h"
//-----------------------------------------------
bit    FLAG;                    //1:上升沿中断  0:下降沿中断
sbit   P10 = P1^0;
//-----------------------------------------
//外部中断服务程序
void exint0() interrupt 0       //INT0 中断入口
{
       P10 = !P10;              //将测试口取反
       FLAG = INT0;             //保存 INT0 口的状态, INT0=0(下降沿); INT0=1(上升沿)
}
//-----------------------------------------------
void main()
{
       INT0 = 1;
       IT0 = 0;                 //设置 INT0 的中断类型  (1:仅下降沿  0:上升沿和下降沿)
       EX0 = 1;                 //使能 INT0 中断
       EA = 1;
       while (1);
}
```

2.汇编程序:

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 INT0 中断举例-----------------------------------------------------*/
/*----  在 Keil C  开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可------*/
/*---------------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz
/*---------------------------------------------------------------------------------------------------*/
FLAG      BIT   20H.0           ;1:上升沿中断  0:下降沿中断
```

```
;-----------------------------------------
        ORG     0000H
        LJMP    START           ;复位入口

        ORG     0003H           ;INT0 中断入口
        LJMP    EXINT0
;-----------------------------------------
        ORG     0100H
START:
        MOV     SP, #3FH

        CLR     IT0             ;设置 INT0 的中断类型  (1:仅下降沿  0:上升沿和下降沿)
        SETB    EX0             ;使能 INT0 中断
        SETB    EA
        SJMP    $
;-----------------------------------------
;外部中断服务程序
EXINT0:
        CPL     P1.0            ;将测试口取反
        PUSH    PSW
        MOV     C, INT0         ;读取 INT0 口的状态
        MOV     FLAG, C         ;保存, INT0=0(下降沿); INT0=1(上升沿)
        POP     PSW
        RETI
;-----------------------------------------
END
```

17.10.1.2 外部中断 INT0(下降沿)的测试程序(C 和汇编)

1.C 程序：

```
/*-------------------------------------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 INT0 中断举例-------------------------------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可------*/
/*-------------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz
/*-------------------------------------------------------------------------------------------------*/

#include "reg51.h"
#include "intrins.h"
;-----------------------------------------------
sbit        P10 = P1^0;
;-----------------------------------------
;外部中断服务程序
void exint0() interrupt 0           ;INT0 中断入口
```

```
{
    P10 = !P10;                 ;将测试口取反
}
;-----------------------------------------
void main()
{
    INT0 = 1;
    IT0 = 1;                    ;设置 INT0 的中断类型  (1:仅下降沿  0:上升沿和下降沿)
    EX0 = 1;                    ;使能 INT0 中断
    EA = 1;
    while (1);
}
```

2.汇编程序：

```
/*-------------------------------------------------------------------------------------------------*/
/* --- STC15F2K60S2  系列  INT0 中断举例-----------------------------------------------------*/
/*----  在 Keil C  开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可------*/
/*-------------------------------------------------------------------------------------------------*/
;/假定测试芯片的工作频率为 18.432MHz
/*-------------------------------------------------------------------------------------------------*/
    ORG     0000H
    LJMP    START           ;复位入口
    ORG     0003H           ;INT0 中断入口

    LJMP    EXINT0
;---------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    SETB    IT0             ;设置 INT0 的中断类型  (1:仅下降沿  0:上升沿和下降沿)

    SETB    EX0             ;使能 INT0 中断
    SETB    EA
    SJMP    $
;---------------------------------------
;外部中断服务程序
EXINT0:
    CPL     P1.0            ;将测试口取反
    RETI
;---------------------------------------
END
```

17.10.2 外部中断 1(INT1)的测试程序

17.10.2.1 外部中断 INT1(上升沿+下降沿)的测试程序(C 和汇编)

1.C 程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 INT1 中断举例-----------------------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可------*/
/*---------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz
/*---------------------------------------------------------------------------------------------*/
#include "reg51.h"
#include "intrins.h"
;-----------------------------------------------
bit     FLAG;               ;1:上升沿中断 0: 下降沿中断
sbit    P10 = P1^0;
;-----------------------------------------
;外部中断服务程序
void exint1() interrupt 2       ;INT1 中断入口
{
    P10 = !P10;             ;将测试口取反
    FLAG = INT1;            ;保存 INT1 口的状态, INT1=0(下降沿); INT1=1(上升沿)
}
;-----------------------------------------------
void main()
{
    INT1 = 1;
    IT1 = 0;                ;设置 INT1 的中断类型 (1:仅下降沿 0:上升沿和下降沿)
    EX1 = 1;                ;使能 INT1 中断
    EA = 1;
    while (1);
}
```

2.汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 INT1 中断举例-----------------------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可------*/
/*---------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz
/*---------------------------------------------------------------------------------------------*/
FLAG    BIT 20H.0           ;1:上升沿中断 0: 下降沿中断
;---------------------------------------
    ORG     0000H
```

```
    LJMP    START           ;复位入口
    ORG     0013H           ;INT1 中断入口

    LJMP    EXINT1
;----------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    CLR     IT1             ;设置 INT1 的中断类型  (1:仅下降沿  0:上升沿和下降沿)

    SETB    EX1             ;使能 INT1 中断
    SETB    EA
    SJMP    $
;----------------------------------------
;外部中断服务程序
EXINT1:
    CPL     P1.0            ;将测试口取反
    PUSH    PSW
    MOV     C, INT1         ;读取 INT1 口的状态
    MOV     FLAG, C         ;保存, INT1=0(下降沿); INT0=1(上升沿)
    POP     PSW
    RETI
;----------------------------------------
END
```

17.10.2.2 外部中断 INT1(下降沿)的测试程序(C 和汇编)

1.C 程序：

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC15F2K60S2  系列  INT1 中断举例-----------------------------------------------*/
/*----  在 Keil C  开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可------*/
/*---------------------------------------------------------------------------------------------------*/
//假定测试芯片的工作频率为 18.432MHz
/*---------------------------------------------------------------------------------------------------*/
#include "reg51.h"
#include "intrins.h"
//-----------------------------------------------
sbit        P10 = P1^0;
//-----------------------------------------
//外部中断服务程序
void exint1() interrupt 2           //INT1 中断入口
{
    P10 = !P10;                     //将测试口取反
}
```

```
//-----------------------------------------------
void main()
{
    INT1 = 1;
    IT1 = 1;                    //设置 INT1 的中断类型  (1:仅下降沿  0:上升沿和下降沿)
    EX1 = 1;                    //使能 INT1 中断
    EA = 1;
    while (1);
}
```

2.汇编程序：

```
/*------------------------------------------------------------------------------------------------*/
/* --- STC15F2K60S2  系列  INT1 中断举例-----------------------------------------------*/
/*----  在 Keil C  开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可------*/
/*------------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz
/*------------------------------------------------------------------------------------------------*/
    ORG     0000H
    LJMP    START           ;复位入口
    ORG     0013H           ;INT1 中断入口

    LJMP    EXINT1
;---------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    SETB    IT1             ;设置 INT1 的中断类型  (1:仅下降沿  0:上升沿和下降沿)

    SETB    EX1             ;使能 INT1 中断
    SETB    EA
    SJMP    $
;---------------------------------------
;外部中断服务程序
EXINT1:
    CPL     P1.0            ;将测试口取反
    RETI
;---------------------------------------
END
```

17.10.3　外部中断 2(INT2)(下降沿中断)的测试程序(C 和汇编)

1.C 程序：

```
/*------------------------------------------------------------------------------------------------*/
```

```
/* --- 演示 STC 15 系列单片机外 c 中断 2 (INT2) (下降沿) ----------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可------*/
/*-------------------------------------------------------------------------------------*/
//假定测试芯片的工作频率为 18.432MHz
/*-------------------------------------------------------------------------------------*/
#include "reg51.h"
#include "intrins.h"
//---------------------------------------------
sfr     INT_CLKO = 0x8f;        //外部中断与时钟输出控制寄存器
sbit    P10 = P1^0;
//---------------------------------------------
//外部中断服务程序
void exint2() interrupt 10              //INT2 中断入口
{
    P10     =   !P10;               //将测试口取反
//  INT_CLKO &= 0xEF;               //若需要手动清除中断标志,可先关闭中断,
                                    //此系统会自动清除内部的中断标志
//  INT_CLKO |= 0x10;               //然后再开中断即可
}

void main()
{
    INT_CLKO |= 0x10;               //(EX2 = 1)使能 INT2 中断
    EA = 1;
    while (1);
}
```

2.汇编程序：

```
/*-------------------------------------------------------------------------------------*/
/* --- 演示 STC 15  系列单片机外部中断 2 (INT2) (下降沿) ----------------------------*/
/*---- 在 Keil C  开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可------*/
/*-------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz
/*-------------------------------------------------------------------------------------*/
INT_CLKO    DATA    08FH        ;外部中断与时钟输出控制寄存器
;---------------------------------------
    ORG     0000H
    LJMP    START               ;复位入口
    ORG     0053H               ;INT2 中断入口

    LJMP    EXINT2
;---------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
```

```
    ORL     INT_CLKO, #10H      ;(EX2 = 1)使能 INT2 中断
    SETB    EA
    SJMP    $
;---------------------------------------
;外部中断服务程序
EXINT2:
    CPL     P1.0                ;将测试口取反
;   ANL     INT_CLKO, #0EFH     ;若需要手动清除中断标志,可先关闭中断,
                                ;此系统会自动清除内部的中断标志
;   ORL     INT_CLKO, #10H      ;然后再开中断即可
    RETI
;---------------------------------------
END
```

17.10.4 外部中断 3(INT3)(下降沿中断)的测试程序(C 和汇编)

1.C 程序：

```
/*-------------------------------------------------------------------------------------------------*/
/* --- 演示 STC 15 系列单片机外部中断 3 (INT3) (下降沿) ------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------------*/
//假定测试芯片的工作频率为 18.432MHz
/*-------------------------------------------------------------------------------------------------*/
#include "reg51.h"
#include "intrins.h"
//---------------------------------------------
sfr     INT_CLKO = 0x8f;        //外部中断与时钟输出控制寄存器
sbit    P10 = P1^0;
//---------------------------------------------
//外部中断服务程序
void exint3() interrupt 11          //INT3 中断入口
{
    P10 = !P10;                     //将测试口取反
//  INT_CLKO &= 0xDF;               //若需要手动清除中断标志,可先关闭中断,
                                    //此系统会自动清除内部的中断标志
//  INT_CLKO |= 0x20;               //然后再开中断即可
}

void main()
{
    INT_CLKO |= 0x20;               //(EX3 = 1)使能 INT3 中断
    EA = 1;
    while (1);
}
```

2.汇编程序：

```
/*-------------------------------------------------------------------------------------------------*/
/*----演示 STC 15 系列单片机外部中断 3 (INT3) (下降沿) -------------------------------*/
/*----  在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可------*/
/*-------------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz
/*-------------------------------------------------------------------------------------------------*/
INT_CLKO    DATA    08FH            ;外部中断与时钟输出控制寄存器
;----------------------------------------

        ORG     0000H
        LJMP    START               ;复位入口
        ORG     005BH               ;INT3 中断入口

        LJMP    EXINT3
;----------------------------------------
        ORG     0100H
START:
        MOV     SP, #3FH
        ORL     INT_CLKO, #20H      ;(EX3 = 1)使能 INT3 中断
        SETB    EA
        SJMP    $
;----------------------------------------
;外部中断服务程序
EXINT3:
        CPL     P1.0                ;将测试口取反
;       ANL     INT_CLKO, #0DFH     ;若需要手动清除中断标志,可先关闭中断,
                                    ;此系统会自动清除内部的中断标志
;       ORL     INT_CLKO, #20H      ;然后再开中断即可
        RETI
;----------------------------------------
END
```

17.10.5 外部中断 4(INT4)(下降沿中断)的测试程序(C 和汇编)

1. C 程序：

```
/*-------------------------------------------------------------------------------------------------*/
/* ---  演示 STC 15  系列单片机外部中断 4 (INT4) (下降沿) ----------------------------*/
/*----  在 Keil C  开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可--------*/
/*-------------------------------------------------------------------------------------------------*/
//假定测试芯片的工作频率为 18.432MHz
/*-------------------------------------------------------------------------------------------------*/
```

```
#include "reg51.h"
#include "intrins.h"
//-----------------------------------------------
sfr         INT_CLKO = 0x8f;            //外部中断与时钟输出控制寄存器
sbit        P10 = P1^0;
//-----------------------------------------------
//外部中断服务程序
void exint4() interrupt 16              //INT4 中断入口
{
    P10 = !P10;                         //将测试口取反
//  INT_CLKO    &=  0xBF;               //若需要手动清除中断标志,可先关闭中断,
                                        //此系统会自动清除内部的中断标志
//  INT_CLKO |= 0x40;                   //然后再开中断即可
}

void main()
{
    INT_CLKO |= 0x40;                   //(EX4 = 1)使能 INT4 中断
    EA = 1;
    while (1);
}
```

2.汇编程序：

```
/*-------------------------------------------------------------------------------------------------*/
/* ---  演示 STC 15  系列单片机外部中断 4 (INT4) (下降沿) ----------------------------*/
/*----  在 Keil C  开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可------*/
/*-------------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz
/*-------------------------------------------------------------------------------------------------*/
INT_CLKO    DATA    08FH            ;外部中断与时钟输出控制寄存器
;---------------------------------------

    ORG     0000H
    LJMP    START                   ;复位入口
    ORG     0083H                   ;INT4 中断入口

    LJMP    EXINT4
;---------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    ORL     INT_CLKO, #40H          ;(EX4 = 1)使能 INT4 中断
    SETB    EA
    SJMP    $
;---------------------------------------
```

```
;外部中断服务程序
EXINT4:
    CPL     P1.0                ;将测试口取反
;   ANL     INT_CLKO, #0BFH     ;若需要手动清除中断标志,可先关闭中断,
                                ;此系统会自动清除内部的中断标志
;   ORL     INT_CLKO, #40H      ;然后再开中断即可
    RETI
;----------------------------------------
END
```

17.10.6 T0 扩展为外部下降沿中断的测试程序(C 和汇编)

——利用 T0 的外部计数方式

1. C 程序：

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 T0 扩展为外部下降沿中断举例----------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可------*/
/*---------------------------------------------------------------------------------------------------*/
//假定测试芯片的工作频率为 18.432MHz
/*---------------------------------------------------------------------------------------------------*/
#include "reg51.h"
#include "intrins.h"
//-----------------------------------------------
sfr     AUXR = 0x8e;                //辅助寄存器
sbit    P10 = P1^0;
//-----------------------------------------------
//定时器 0 中断服务程序
void t0int() interrupt 1            //中断入口
{
    P10 = !P10;                     //将测试口取反
}

void main()
{
    AUXR = 0x80;                    //定时器 0 为 1T 模式
    TMOD = 0x04;                    //设置定时器 0 为 16 位自动重装载外部记数模式
    TH0 = TL0 = 0xff;               //设置定时器 0 初始值
    TR0 = 1;                        //定时器 0 开始工作
    ET0 = 1;                        //开定时器 0 中断
    EA = 1;
    while (1);
}
```

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 T0 扩展为外部下降沿中断举例----------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz
/*---------------------------------------------------------------------------------------------*/
AUXR        DATA    08EH        ;辅助寄存器
;-----------------------------------
    ORG     0000H
    LJMP    START               ;复位入口
    ORG     000BH               ;中断入口

    LJMP    T0INT
;-----------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    MOV     AUXR, #80H          ;定时器 0 为 1T 模式
    MOV     TMOD, #04H          ;设置定时器 0 为 16 位自动重装载外部记数模式
    MOV     A, #0FFH            ;设置定时器 0 初始值
    MOV     TL0, A
    MOV     TH0, A
    SETB    TR0                 ;定时器 0 开始工作
    SETB    ET0                 ;开定时器 0 中断
    SETB    EA
    SJMP    $
;-------------------------------------
;定时器 0 中断服务程序
T0INT:
    CPL     P1.0                ;将测试口取反
    RETI
;-------------------------------------
END
```

17.10.7 T1 扩展为外部下降沿中断的测试程序(C 和汇编)

——利用 T1 的外部计数方式

1. C 程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 T1 扩展为外部下降沿中断举例----------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/
```

```
//假定测试芯片的工作频率为 18.432MHz
/*---------------------------------------------------------------------------------------------------*/
#include "reg51.h"
#include "intrins.h"
//-----------------------------------------------
sfr         AUXR = 0x8e;                //辅助寄存器
sbit        P10 = P1^0;
//-----------------------------------------------
//定时器 1 中断服务程序
void t1int() interrupt 3                //中断入口
{
    P10 = !P10;                         //将测试口取反
}

void main()
{
    AUXR = 0x40;                        //定时器 1 为 1T 模式
    TMOD = 0x40;                        //设置定时器 10 为 16 位自动重装载外部记数模式
    TH1 = TL1 = 0xff;                   //设置定时器 1 初始值
    TR1 = 1;                            //定时器 1 开始工作
    ET1 = 1;                            //开定时器 1 中断
    EA = 1;
    while (1);
}
```

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 T1 扩展为外部下降沿中断举例----------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz
/*---------------------------------------------------------------------------------------------------*/
AUXR        DATA    08EH            ;辅助寄存器
;-----------------------------------------
    ORG     0000H
    LJMP    START                   ;复位入口
    ORG     001BH                   ;中断入口

    LJMP    T1INT
;-----------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    MOV     AUXR, #40H              ;定时器 1 为 1T 模式
    MOV     TMOD, #40H              ;设置定时器 10 为 16 位自动重装载外部记数模式
```

```
    MOV     A, #0FFH            ;设置定时器 1 初始值
    MOV     TL1, A
    MOV     TH1, A
    SETB    TR1                 ;定时器 1 开始工作
    SETB    ET1                 ;开定时器 1 中断
    SETB    EA
    SJMP    $
;---------------------------------------
;定时器 1 中断服务程序
T1INT:
    CPL     P1.0                ;将测试口取反
    RETI
;---------------------------------------
END
```

17.10.8 T2 扩展为外部下降沿中断的测试程序(C 和汇编)

——利用 T2 的外部计数方式

1. C 程序：

```
/*-------------------------------------------------------------------------------------------------*/
/* --- STC15F2K60S2  系列  T2 扩展为外部下降沿中断举例-----------------------------*/
/*----  在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------------*/
//假定测试芯片的工作频率为 18.432MHz
/*-------------------------------------------------------------------------------------------------*/
#include "reg51.h"
#include "intrins.h"
//-----------------------------------------------
sfr         IE2 = 0xaf;             //中断使能寄存器 2
sfr         AUXR = 0x8e;            //辅助寄存器
sfr         T2H = 0xD6;             //定时器 2 高 8 位
sfr         T2L = 0xD7;             //定时器 2 低 8 位
sbit        P10 = P1^0;
//-----------------------------------------------
//定时器 2 中断服务程序
void t2int() interrupt 12           //中断入口
{
    P10 = !P10;                     //将测试口取反
//  IE2 &= ~0x04;                   //若需要手动清除中断标志,可先关闭中断,
                                    //此时系统会自动清除内部的中断标志
//  IE2 |= 0x04;                    //然后再开中断即可
}

void main()
```

```
{
    AUXR |= 0x04;               //定时器 2 为 1T 模式
    AUXR |= 0x08;               //T2_C/T=1, T2(P3.1)引脚为时钟源
    T2H = T2L = 0xff;           //初始化计时值
    AUXR |= 0x10;               //定时器 2 开始计时
    IE2 |= 0x04;                //开定时器 2 中断
    EA = 1;
    while (1);
}
```

2. 汇编程序：

```
/*-------------------------------------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 T2 扩展为外部下降沿中断举例----------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz
/*-------------------------------------------------------------------------------------------------*/
IE2         DATA    0AFH        ;中断使能寄存器 2
AUXR        DATA    08EH        ;辅助寄存器
T2H         DATA    0D6H        ;定时器 2 高 8 位
T2L         DATA    0D7H        ;定时器 2 低 8 位
;---------------------------------------
    ORG     0000H
    LJMP    START               ;复位入口
    ORG     0063H               ;中断入口

    LJMP    T2INT
;---------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    ORL     AUXR, #04H          ;定时器 2 为 1T 模式

    ORL     AUXR, #08H          ;T2_C/T=1, T2(P3.1)引脚为时钟源
    MOV     A, #0FFH            ;初始化计时值

    MOV     T2L, A
    MOV     T2H, A
    ORL     AUXR, #10H          ;定时器 2 开始计时
    ORL     IE2, #04H           ;开定时器 2 中断
    SETB    EA
    SJMP    $

;---------------------------------------
;定时器 2 中断服务程序
```

```
T2INT:
    CPL     P1.0                ;将测试口取反
;   ANL     IE2, #0FBH          ;若需要手动清除中断标志,可先关闭中断,
                                ;此时系统会自动清除内部的中断标志
;   ORL     IE2, #04H           ;然后再开中断即可
    RETI
;----------------------------------------
END
```

17.10.9 用 CCP/PCA 功能扩展外部中断的测试程序(C 和汇编)

1. C 程序：

```
/*---------------------------------------------------------------------------------------------------*/
/* --- 演示 STC 1T 系列单片机 用 PCA 功能扩展外部中断 ----------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可--------*/
/*---------------------------------------------------------------------------------------------------*/
//假定测试芯片的工作频率为 18.432MHz
/*---------------------------------------------------------------------------------------------------*/
#include "reg51.h"
#include "intrins.h"
#define     FOSC            18432000L
typedef     unsigned char   BYTE;
typedef     unsigned int    WORD;
typedef     unsigned long   DWORD;

sfr         P_SW1 = 0xA2;               //外设功能切换寄存器 1
#define     CCP_S0          0x10        //P_SW1.4
#define     CCP_S1          0x20        //P_SW1.5

sfr         CCON = 0xD8;                //PCA 控制寄存器
sbit        CCF0 = CCON^0;              //PCA 模块 0 中断标志
sbit        CCF1 = CCON^1;              //PCA 模块 1 中断标志
sbit        CR = CCON^6;                //PCA 定时器运行控制位
sbit        CF = CCON^7;                //PCA 定时器溢出标志
sfr         CMOD = 0xD9;                //PCA 模式寄存器
sfr         CL = 0xE9;                  //PCA 定时器低字节
sfr         CH = 0xF9;                  //PCA 定时器高字节
sfr         CCAPM0 = 0xDA;              //PCA 模块 0 模式寄存器
sfr         CCAP0L = 0xEA;              //PCA 模块 0 捕获寄存器 LOW
sfr         CCAP0H = 0xFA;              //PCA 模块 0 捕获寄存器 HIGH
sfr         CCAPM1 = 0xDB;              //PCA 模块 1 模式寄存器
sfr         CCAP1L = 0xEB;              //PCA 模块 1 捕获寄存器 LOW
sfr         CCAP1H = 0xFB;              //PCA 模块 1 捕获寄存器 HIGH
sfr         CCAPM2 = 0xDC;              //PCA 模块 2 模式寄存器
```

```
sfr         CCAP2L = 0xEC;          //PCA 模块 2 捕获寄存器 LOW
sfr         CCAP2H = 0xFC;          //PCA 模块 2 捕获寄存器 HIGH
sfr         PCAPWM0 = 0xf2;         //PCA 模块 0 的 PWM 寄存器
sfr         PCAPWM1 = 0xf3;         //PCA 模块 1 的 PWM 寄存器
sfr         PCA_ PWM2 = 0xf4;       //PCA 模块 2 的 PWM 寄存器
sbit        PCA_LED = P1^0;         //PCA 测试 LED

void PCA_isr() interrupt 7 using 1
{
    CCF0 = 0;                       //清中断标志
    PCA_LED = !PCA_LED;             //测试 LED 取反
}

void main()
{
    ACC = P_SW1;
    ACC &= ~(CCP_S0 | CCP_S1);      //CCP_S0=0 CCP_S1=0
    P_SW1 = ACC;                    //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)

//  ACC = P_SW1;
//  ACC &= ~(CCP_S0 | CCP_S1);      //CCP_S0=1 CCP_S1=0
//  ACC |= CCP_S0;                  //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
//  P_SW1 = ACC;
//
//  ACC = P_SW1;
//  ACC &= ~(CCP_S0 | CCP_S1);      //CCP_S0=0 CCP_S1=1
//  ACC |= CCP_S1;                  //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
//  P_SW1 = ACC;

    CCON = 0;                       //初始化 PCA 控制寄存器
                                    //PCA 定时器停止
                                    //清除 CF 标志
                                    //清除模块中断标志
    CL = 0;                         //复位 PCA 寄存器
    CH = 0;
    CMOD = 0x00;                    //设置 PCA 时钟源
                                    //禁止 PCA 定时器溢出中断
    CCAPM0 = 0x11;                  //PCA 模块 0 为下降沿触发
//  CCAPM0 = 0x21;                  //PCA 模块 0 为上升沿沿触发
//  CCAPM0 = 0x31;                  //PCA 模块 0 为上升沿/下降沿触发

    CR = 1;                         //PCA 定时器开始工作
    EA = 1;
    while (1);
}
```

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- 演示 STC 1T 系列单片机 用 PCA 功能扩展外部中断 ----------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可------*/
/*---------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz
/*---------------------------------------------------------------------------------------------*/
;本测试程序以 PCA 模块 0 为例进行说明，PCA 的模块 1 和模块 2 与模块 0 的实用方法相同
P_SW1       EQU   0A2H        ;外设功能切换寄存器 1
CCP_S0      EQU   10H         ;P_SW1.4

CCP_S1      EQU   20H         ;P_SW1.5
CCON        EQU   0D8H        ;PCA 控制寄存器
CCF0        BIT   CCON.0      ;PCA 模块 0 中断标志
CCF1        BIT   CCON.1      ;PCA 模块 1 中断标志
CR          BIT   CCON.6      ;PCA 定时器运行控制位
CF          BIT   CCON.7      ;PCA 定时器溢出标志
CMOD        EQU   0D9H        ;PCA 模式寄存器
CL          EQU   0E9H        ;PCA 定时器低字节
CH          EQU   0F9H        ;PCA 定时器高字节
CCAPM0      EQU   0DAH        ;PCA 模块 0 模式寄存器
CCAP0L      EQU   0EAH        ;PCA 模块 0 捕获寄存器 LOW
CCAP0H      EQU   0FAH        ;PCA 模块 0 捕获寄存器 HIGH
CCAPM1      EQU   0DBH        ;PCA 模块 1 模式寄存器
CCAP1L      EQU   0EBH        ;PCA 模块 1 捕获寄存器 LOW
CCAP1H      EQU   0FBH        ;PCA 模块 1 捕获寄存器 HIGH
CCAPM2      EQU   0DCH        ;PCA 模块 2 模式寄存器
CCAP2L      EQU   0ECH        ;PCA 模块 2 捕获寄存器 LOW
CCAP2H      EQU   0FCH        ;PCA 模块 2 捕获寄存器 HIGH
PCA_PWM0    EQU   0F2H        ;PCA 模块 0 的 PWM 寄存器
PCA_PWM1    EQU   0F3H        ;PCA 模块 1 的 PWM 寄存器
PCA_PWM2    EQU   0F4H        ;PCA 模块 2 的 PWM 寄存器

PCA_LED     BIT   P1.0        ;PCA 测试 LED
;---------------------------------------

    ORG     0000H
    LJMP    START
    ORG     003BH
PCA_ISR:
    PUSH    PSW
    PUSH    ACC
CKECK_CCF0:
    JNB     CCF0, PCA_ISR_EXIT      ;判断是否为捕获中断
    CLR     CCF0                    ;清中断标志
```

```
    CPL     PCA_LED             ;测试 LED 取反
PCA_ISR_EXIT:
    POP     ACC
    POP     PSW
    RETI
;----------------------------------------
    ORG     0100H

START:
    MOV     SP, #5FH
    MOV     A, P_SW1

    ANL     A, #0CFH            ;CCP_S0=0 CCP_S1=0
    MOV     P_SW1, A            ;(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)
;   MOV     A, P_SW1
;   ANL     A, #0CFH            ;CCP_S0=1 CCP_S1=0
;   ORL     A, #CCP_S0          ;(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
;   MOV     P_SW1, A

;   MOV     A, P_SW1
;   ANL     A, #0CFH            ;CCP_S0=0 CCP_S1=1
;   ORL     A, #CCP_S1          ;(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
;   MOV     P_SW1, A
    MOV     CCON, #0            ;初始化 PCA 控制寄存器
                                ;PCA 定时器停止
                                ;清除 CF 标志
                                ;清除模块中断标志
    CLR     A
    MOV     CL, A               ;复位 PCA 寄存器
    MOV     CH, A
    MOV     CMOD, #00H          ;设置 PCA 时钟源
                                ;禁止 PCA 定时器溢出中断
    MOV     CCAPM0, #11H        ;PCA 模块 0 捕获 CCP0(P1.3)的下降沿
;   MOV     CCAPM0, #21H        ;PCA 模块 0 捕获 CCP0(P1.3)的上升沿
;   MOV     CCAPM0, #31H        ;PCA 模块 0 捕获 CCP0(P1.3)的上升沿/下降沿
;------------------------------
    SETB    CR                  ;PCA 定时器开始工作
    SETB    EA
    SJMP    $
;----------------------------------------
    END
```

18 定时器/计数器

STC15W4K32S4 系列单片机内部设置了 5 个 16 位定时器/计数器：16 位定时器/计数器 T0 和 T1、T2、T3 以及 T4。5 个 16 位定时器 T0、T1、T2、T3 和 T4 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。

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(T0 为 P3.4，T1 为 P3.5，T2 为 P3.1，T3 为 P0.7，T4 为 P0.5)，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 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。

STC15 全系列的定时器/计数器的类型如下表所示。

定时器 计数器 单片机型号	定时器/计数器 0	定时器/计数器 1	定时器/计数器 2	定时器/计数器 3	定时器/计数器 4
STC15F104W 系列	√		√		
STC15F408AD 系列	√		√		
STC15W204S 系列	√		√		
STC15W408AS 系列	√		√		
STC15W404S 系列	√	√	√		
STC15W1K16S 系列	√	√	√		
STC15F2K60S2 系列	√	√	√		
STC15W4K32S4 系列	√	√	√	√	√

上表中√表示对应的系列有相应的定时器/计数器。

18.1 定时器/计数器的相关寄存器

符号	描述	地址	位地址及其符号								复位值
			MSB							LSB	
TCON	Timer Control	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000 0000B
TMOD	Timer Mode	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	0000 0000B
TL0	Timer Low 0	8AH									0000 0000B
TL1	Timer Low 1	8BH									0000 0000B
TH0	Timer High 0	8CH									0000 0000B
TH1	Timer High 1	8DH									0000 0000B
IE	中断允许寄存器	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000 0000B
IP	中断优先级寄存器	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	0000 0000B
T2H	定时器 2 高 8 位寄存器	D6H									0000 0000B
T2L	定时器 2 低 8 位寄存器	D7H									0000 0000B
AUXR	辅助寄存器	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000 0001B
INT_CLKO AUXR2	外部中断允许和时钟输出寄存器	8FH	-	EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	T0CLKO	x000 0000B
T4T3M	T4 和 T3 的控制寄存器	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO	0000 0000B
T4H	定时器 4 高 8 位寄存器	D2H									0000 0000B
T4L	定时器 4 低 8 位寄存器	D3H									0000 0000B
T3H	定时器 3 高 8 位寄存器	D4H									0000 0000B
T3L	定时器 3 低 8 位寄存器	D5H									0000 0000B
IE2	Interrupt Enableregister	AFH	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000 0000B

1、定时器/计数器 0/1 控制寄存器 TCON

TCON 为定时器/计数器 T0、T1 的控制寄存器，同时也锁存 T0、T1 溢出中断源和外部请求中断源等，TCON 格式如下：

TCON：定时器/计数器中断控制寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
TCON	88H	name	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，由硬件置“1”TF1，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）

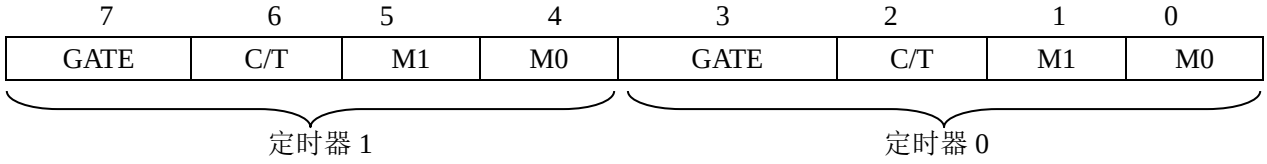
IT0： 外部中断源 0 触发控制位。
IT0=0，上升沿或下降沿均可触发外部中断 0。
IT0=1，外部中断 0 程控为下降沿触发方式。

2、定时器/计数器工作模式寄存器 TMOD

定时和计数功能由特殊功能寄存器 TMOD 的控制位 C/T 进行选择，TMOD 寄存器的各位信息如下表所列。可以看出，2 个定时/计数器有 4 种操作模式，通过 TMOD 的 M1 和 M0 选择。2 个定时/计数器的模式 0、1 和 2 都相同，模式 3 不同，各式下的功能如下所述。

寄存器 TMOD 各位的功能描述

TMOD 地址：89H 复位值：00H
不可位寻址



位	符号		功能
TMOD.7/	GATE		TMOD.7 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。
TMOD.3/	GATE		TMOD.3 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。
TMOD.6/	C/T		TMOD.6 控制定时器 1 用作定时器或计数器，清零则用作定时器（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）
TMOD.2/	C/T		TMOD.2 控制定时器 0 用作定时器或计数器，清零则用作定时器（对内部系统时钟进行计数），置 1 用作计数器（对引脚 T0/P3.4 的外部脉冲进行计数）
TMOD.5/ TMOD.4	M1	M0	定时器/计数器 1 模式选择
	0	0	16 位自动重装定时器。当溢出时将 RL_TH1 和 RL_TL1 存放的值自动重装入 TH1 和 TL1 中。
	0	1	16 位不可重装载模式，TL1、TH1 全用
	1	0	8 位自动重装载定时器，当溢出时将 TH1 存放的值自动重装入 TL1
	1	1	定时器/计数器 1 此时无效（停止计数）。
TMOD.1/ TMOD.0	M1	M0	定时器/计数器 0 模式选择
	0	0	16 位自动重装定时器。当溢出时将 RL_TH0 和 RL_TL0 存放的值自动重装入 TH0 和 TL0 中。
	0	1	16 位不可重装载模式，TL0、TH0 全用
	1	0	8 位自动重装载定时器，当溢出时将 TH0 存放的值自动重装入 TL0
	1	1	不可屏蔽中断的 16 位自动重装定时器

3、辅助寄存器 AUXR

STC15 系列单片机是 1T 的 8051 单片机，为兼容传统 8051，定时器 0、定时器 1，和定时器 2 复位后是传统 8051 的速度，即 12 分频，这是为了兼容传统 8051。但也可不进行 12 分频，通过设置新增加的特殊功能寄存器 AUXR，将 T0，T1，T2 设置为 1T。普通 111 条机器指令执行速度是固定的，快 4 到 24 倍，无法改变。

AUXR 格式如下：

AUXR：辅助寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	name	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

T0x12： 定时器 0 速度控制位
0，定时器 0 是传统 8051 速度，12 分频；
1，定时器 0 的速度是传统 8051 的 12 倍，不分频

T1x12： 定时器 1 速度控制位
0，定时器 1 是传统 8051 速度，12 分频；
1，定时器 1 的速度是传统 8051 的 12 倍，不分频
如果 UART1/串口 1 用 T1 作为波特率发生器，则由 T1x12 决定 UART1/串口 1 是 12T 还是 1T

UART_M0x6：串口 1 模式 0 的通信速度设置位。
0，串口 1 模式 0 的速度是传统 8051 单片机串口的速度，12 分频；
1，串口 1 模式 0 的速度是传统 8051 单片机串口速度的 6 倍，2 分频

T2R： 定时器 2 允许控制位
0，不允许定时器 2 运行；
1，允许定时器 2 运行

T2_C/T：控制定时器 2 用作定时器或计数器。
0，用作定时器（对内部系统时钟进行计数）；
1，用作计数器（对引脚 T2/P3.1 的外部脉冲进行计数）

T2x12： 定时器 2 速度控制位
0，定时器 2 是传统 8051 速度，12 分频；
1，定时器 2 的速度是传统 8051 的 12 倍，不分频
如果串口 1 或串口 2 用 T2 作为波特率发生器，则由 T2x12 决定串口 1 或串口 2 是 12T 还是 1T

EXTRAM：内部/外部 RAM 存取控制位
0，允许使用逻辑上在片外、物理上在片内的扩展 RAM；
1，禁止使用逻辑上在片外、物理上在片内的扩展 RAM

S1ST2： 串口 1（UART1）选择定时器 2 作波特率发生器的控制位
0，选择定时器 1 作为串口 1（UART1）的波特率发生器；
1，选择定时器 2 作为串口 1（UART1）的波特率发生器，此时定时器 1 得到释放，可以作为独立定时器使用

4、T0，T1 和 T2 的时钟输出寄存器和外部中断允许 INT_CLKO（AUXR2）

T0CLKO/P3.5、T1CLKO/P3.4 和 T2CLKO/P3.0 的时钟输出控制由 INT_CLKO（AUXR2）寄存器的

T0CLKO 位、T1CLKO 位和 T2CLKO 位控制。

- T0CLKO 的输出时钟频率由定时器 0 控制
- T1CLKO 的输出时钟频率由定时器 1 控制，相应的定时器需要工作在定时器的模式 0（16 位自动重装载模式）或模式 2（8 位自动重装载模式），不要允许相应的定时器中断，免得 CPU 反复进中断。
- T2CLKO 的输出时钟频率由定时器 2 控制，同样不要允许相应的定时器中断，免得 CPU 反复进中断。

定时器 2 的工作模式固定为模式 0（16 位自动重装载模式），在此模式下定时器 2 可用作可编程时钟输出。

INT_CLKO（AUXR2）格式如下：

INT_CLKO（AUXR2）：外部中断允许和时钟输出寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
INT_CLKO AUXR2	8FH	name	-	EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	T0CLKO

T0CLKO：是否允许将 P3.5/T1 脚配置为定时器 0（T0）的时钟输出 T0CLKO

1，将 P3.5/T1 管脚配置为定时器 0 的时钟输出 T0CLKO，输出时钟频率=T0 溢出率/2

若定时器/计数器 T0 工作在定时器模式 0（16 位自动重装载模式）时，

- 如果 C/T=0，定时器/计数器 T0 是对内部系统时钟计数，则：
  - ✓ T0 工作在 1T 模式（AUXR.7/T0x12=1）时的输出频率=（SYSclk）/（65536-[RL_TH0, RL_TL0]）/2
  - ✓ T0 工作在 12T 模式（AUXR.7/T0x12=0）时的输出频率=（SYSclk）/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

若定时器/计数器 T0 工作在定时器模式 2（8 位自动重装模式），

- 如果 C/T=0，定时器/计数器 T1 是对内部系统时钟计数，则：
  - ✓ T0 工作在 1T 模式（AUXR.7/T0x12=1）时的输出频率=（SYSclk）/（256-TH0）/2
  - ✓ T0 工作在 12T 模式（AUXR.7/T0x12=0）时的输出频率=（SYSclk）/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

0，不允许 P3.5/T1 管脚被配置为定时器 0 的时钟输出

T1CLKO：是否允许将 P3.4/T0 脚配置为定时器 1（T1）的时钟输出 T1CLKO

1，将 P3.4/T0 管脚配置为定时器 1 的时钟输出 T1CLKO，输出时钟频率=T1 溢出率/2

若定时器/计数器 T1 工作在定时器模式 0（16 位自动重装载模式），

- 如果 C/T=0，定时器/计数器 T1 是对内部系统时钟计数，则：
  - ✓ T1 工作在 IT 模式（AUXR.6/T1x12=1）时的输出频率=（SYSclk）/（65536-[RL_TH1, RL_TL1]）/2
  - ✓ T1 工作在 12T 模式（AUXR.6/T1x12=0）时的输出频率=（SYSclk）/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

若定时器/计数器 T1 工作在模式 2（8 位自动重装模式），

- 如果 C/T=0，定时器/计数器 T1 是对内部系统时钟计数，则：
  - ✓ T1 工作在 1T 模式（AUXR.6/T1x12=1）时的输出频率=（SYSclk）/（256-TH1）/2
  - ✓ T1 工作在 12T 模式（AUXR.6/T1x12=0）时的输出频率=（SYSclk）/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

0，不允许 P3.4/T0 管脚被配置为定时器 1 的时钟输出

T2CLKO：是否允许将 P3.0 脚配置为定时器 2（T2）的时钟输出 T2CLKO

1，允许将 P3.0 脚配置为定时器 2 的时钟输出 T2CLKO，输出时钟频率=T2 溢出率/2

- 如果 T2_C/T=0，定时器/计数器 T2 是对内部系统时钟计数，则：
  - ✓ T2 工作在 1T 模式（AUXR.2/T2x12=1）时的输出频率=（SYSclk）/（65536-[RL_TH2, RL_TL2]）/2
  - ✓ T2 工作在 12T 模式（AUXR.2/T2x12=0）时的输出频率=（SYSclk）/12/（65536-[RL_TH2, RL_TL2]）/2
- 如果 T2_C/T=1，定时器/计数器 T2 是对外部脉冲输入（P3.1/T2）计数，则：
  - ✓ 输出时钟频率=（T2_Pin_CLK）/（65536-[RL_TH2, RL_TL2]）/2

0，不允许将 P3.0 脚配置为定时器 2（T2）的时钟输出 T2CLKO

EX4： 外部中断 4（INT4）中断允许位。外部中断 4（INT4）只能下降沿触发。
EX4=1 允许中断；
EX4=0 禁止中断。

EX3： 外部中断 3（INT3）中断允许位。外部中断 3（INT3）也只能下降沿触发。
EX3=1 允许中断；
EX3=0 禁止中断。

EX2： 外部中断 2（INT2）中断允许位。外部中断 2（INT2）同样只能下降沿触发。
EX2=1 允许中断；
EX2=0 禁止中断。

5、定时器 T0 和 T1 的中断控制寄存器：IE 和 IP

IE：中断允许寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	name	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

EA： CPU 的总中断允许控制位。
EA=1，CPU 开放中断；
EA=0，CPU 屏蔽所有的中断申请。
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。

ET1： 定时/计数器 T1 的溢出中断允许位。
ET1=1，允许 T1 中断；
ET1=0，禁止 T1 中断。

ET0： T0 的溢出中断允许位。
ET0=1，允许 T0 中断
ET0=0，禁止 T0 中断。

IP：中断优先级控制寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IP	B8H	name	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0

PT1： 定时器 1 中断优先级控制位。
当 PT1=0 时，定时器 1 中断为最低优先级中断（优先级 0）
当 PT1=1 时，定时器 1 中断为最高优先级中断（优先级 1）

PT0： 定时器 0 中断优先级控制位。

当 PT0=0 时，定时器 0 中断为最低优先级中断（优先级 0）
当 PT0=1 时，定时器 0 中断为最高优先级中断（优先级 1）

【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。

6、定时器 T4 和 T3 的控制寄存器：T4T3M（地址：0xD1）

T4T3M（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	name	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO

T4R： 定时器 4 运行控制位。
0：不允许定时器 4 运行；
1：允许定时器 4 运行。

T4_C/T：控制定时器 4 用作定时器或计数器。
0：用作定时器（对内部系统时钟进行计数）；
1：用作计数器（对引脚 T4/P0.7 的外部脉冲进行计数）

T4x12： 定时器 4 速度控制位。
0：定时器 4 速度是 8051 单片机定时器的速度，即 12 分频；
1：定时器 4 速度是 8051 单片机定时器速度的 12 倍，即不分频。

T4CLKO：是否允许将 P0.6 脚配置为定时器 4（T4）的时钟输出 T4CLKO
1：允许将 P0.6 脚配置为定时器 4 的时钟输出 T4CLKO，输出时钟频率=T4 溢出率/2
- 如果 T4_C/T=0，定时器/计数器 T4 是对内部系统时钟计数，则：
  - ✓ T4 工作在 1T 模式（T4T3M.5/T4x12=1）时的输出频率=（SYSclk）/（65536 - [RL_TH4, RL_TL4]）/2
  - ✓ T4 工作在 12T 模式（T4T3M.5/T4x12=0）时的输出频率=（SYSclk）/12/（65536 - [RL_TH4, RL_TL4]）/2
- 如果 T4_C/T=1，定时器/计数器 T4 是对外部脉冲输入（P0.7/T4）计数，则：
  - ✓ 输出时钟频率=（T4_Pin_CLK）/（65536 - [RL_TH4, RL_TL4]）/2

0：不允许将 P0.6 脚配置为定时器 4（T4）的时钟输出 T4CLKO

T3R： 定时器 3 运行控制位，
0：不允许定时器 3 运行；
1：允许定时器 3 运行。

T3_C/T：控制定时器 3 用作定时器或计数器。
0：用作定时器（对内部系统时钟进行计数）；
1：用作计数器（对引脚 T3/P0.5 的外部脉冲进行计数）

T3x12： 定时器 3 速度控制位。
0：定时器 3 速度是 8051 单片机定时器的速度，即 12 分频；
1：定时器 3 速度是 8051 单片机定时器速度的 12 倍，即不分频。

T3CLKO：是否允许将 P0.4 脚配置为定时器 3（T3）的时钟输出 T3CLKO

1：允许将 P0.4 脚配置为定时器 3 的时钟输出 T3CLKO，输出时钟频率=T3 溢出率/2

- 如果 T3_C/T=0，定时器/计数器 T3 是对内部系统时钟计数，则：
  - ✓ T3 工作在 IT 模式（T4T3M.1/T3x12=1）时的输出频率=（SYSclk）/（65536 - [RL_TH3, RL_TL3]）/2
  - ✓ T3 工作在 12T 模式（T4T3M.1/T3x12=0）时的输出频率=（SYSclk）/12/（65536 -[RL_TH3, RL_TL3]）/2
- 如果 T3_C/T=1，定时器/计数器 T3 是对外部脉冲输入（P0.5/T3）计数，则：
  - ✓ 输出时钟频率=（T3_Pin_CLK）/（65536 - [RL_TH3, RL_TL3]）/2

0：不允许将 P0.4 脚配置为定时器 3（T3）的时钟输出 T3CLKO

7、定时器 T2、T3 和 T4 的中断控制寄存器：IE2

IE2：中断允许寄存器（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE2	AFH	name	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2

ET4： 定时器 4 的中断允许位。
1，允许定时器 4 产生中断；
0，禁止定时器 4 产生中断。

ET3： 定时器 3 的中断允许位。
1，允许定时器 3 产生中断；
0，禁止定时器 3 产生中断。

ES4： 串行口 4 中断允许位。
1，允许串行口 4 中断；
0，禁止串行口 4 中断。

ES3： 串行口 3 中断允许位。
1，允许串行口 3 中断；
0，禁止串行口 3 中断。

ET2： 定时器 2 的中断允许位。
1，允许定时器 2 产生中断；
0，禁止定时器 2 产生中断。

ESPI： SPI 中断允许位。
1，允许 SPI 中断；
0，禁止 SPI 中断。

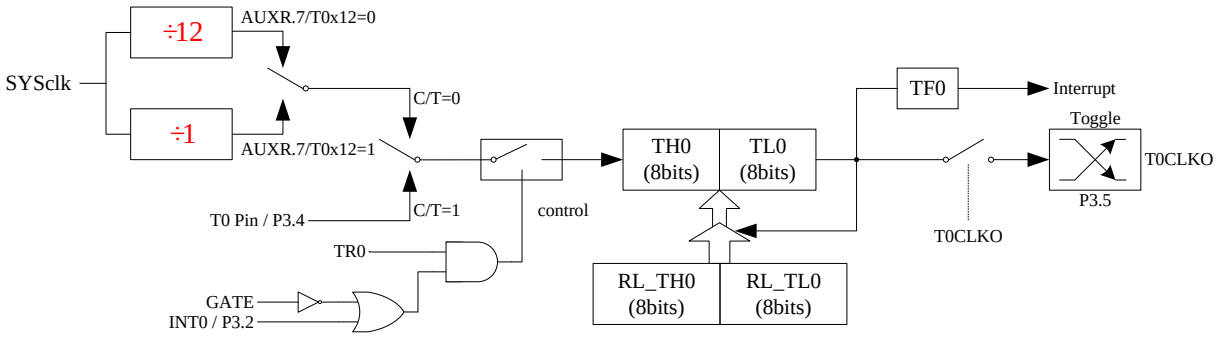
ES2： 串行口 2 中断允许位。
1，允许串行口 2 中断；
0，禁止串行口 2 中断。

18.2 定时器/计数器 0 工作模式

通过对寄存器 TMOD 中的 M1（TMOD.1）、M0（TMOD.0）的设置，定时器/计数器 0 有 4 种不同的工作模式

18.2.1 模式 0(16 位自动重装载模式)及测试程序，建议只学习此模式足矣

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

当 GATE=0（TMOD.3）时，如 TR0=1，则定时器计数。

当 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。

STC15 系列单片机的定时器有两种计数速率：

- 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 2 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，[TL0, TH0]的溢出不仅置位 TF0，而且会自动将[RL_TL0, RL_TH0]的内容重新装入[TL0, TH0]。

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=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）/（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）/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

18.2.1.1 定时器 0 的 16 位自动重装载模式的测试程序(C 和汇编)

1.C 程序:

```
/*-----  演示 STC15 系列单片机定时器 0 的 16 位自动重装载模式----------------*/
/*-----  在 Kei C 开发环境中选择 Intel 8052 编译，头文件包含<reg5l.h>即可----*/
//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"
typedef     unsigned char     BYTE;
typedef     unsigned int      WORD;
//-----------------------------------------------
#define     FOSC              18432000L
#define     T1MS              (65536-FOSC/1000)       //1T 模式，18.432MHz
//#define   T1MS              (65536-FOSC/12/1000)    //12T 模式，18.432MHz

sfr         AUXR = 0x8e;                              //Auxiliary register
sbit        P10 = P1^0;
//-----------------------------------------------
/* Timer0 interrupt routine */
void tm0_isr() interrupt 1 using 1
{
    P10 = ! P10;                //将测试口取反
}
//-----------------------------------------------
void main()
{
    AUXR |= 0x80;               //定时器 0 为 1T 模式
//  AUXR &= 0x7f;               //定时器 0 为 12T 模式
    TMOD = 0x00;                //设置定时器为模式 0(16 位自动重装载)
    TL0 = T1MS;                 //初始化计时值
    TH0 = T1MS >> 8;
    TR0 = 1;                    //定时器 0 开始计时
    ET0 = 1;                    //使能定时器 0 中断
    EA = 1;
    while (1);
}
```

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/*----- 演示 STC15 系列单片机定时器 0 的 16 位自动重装载模式--------------------*/
/*----- 在 Kei C 开发环境中选择 Intel 8052 编译，头文件包含<reg5l.h>即可-----*/
/*---------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz
AUXR        DATA    08EH            ;辅助特殊功能寄存器
;-----------------------------------------
T1MS        EQU     0B800H          ;1T 模式的 1ms 定时值(65536-18432000/1000)
;T1MS       EQU     0FA00H          ;12T 模式的 1ms 定时值(65536-18432000/1000/12)
;-----------------------------------------
    ORG     0000H
    LJMP    START                   ;复位入口
    ORG     000BH                   ;中断入口

    LJMP    T0INT
;-----------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    ORL     AUXR, #80H              ;定时器 0 为 1T 模式
;   ANL     AUXR, #7FH              ;定时器 0 为 12T 模式

    MOV     TMOD, #00H              ;设置定时器为模式 0(16 位自动重装载)

    MOV     TL0, #LOW T1MS          ;初始化计时值
    MOV     TH0, #HIGH T1MS
    SETB    TR0
    SETB    ET0                     ;使能定时器 0 中断
    SETB    EA
    SJMP    $                       ;程序终止
;-----------------------------------
;中断服务程序
T0INT:
    CPL     P1.0                    ;将测试口取反
    RETI
;-----------------------------------
    END
```

18.2.1.2 定时器 0 对系统时钟或外部引脚 T0 的时钟输入进行可编程分频输出的测试程序

----定时器 0 工作在 16 位自动重装载模式

下面是定时器 0 工作在 16 位重装模式时，对内部系统时钟或外部引脚 T0/P3.4 的时钟输入进行可编程时钟分频输出的程序举例（C 和汇编）：

1.C 程序:

```
/*----演示 STC15F2K60S2 系列单片机定时器 0 的可编程时钟分频输出--------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg5l.h>即可----*/

//假定测试芯片的工作频率为 18.432MHz

#include "reg51.h"
typedef   unsigned char       BYTE;
typedef   unsigned int        WORD;
#define   FOSC                18432000L
//-----------------------------------------------
sfr             AUXR = 0x8e;                            //辅助特殊功能寄存器
sfr             INT_CLKO = 0x8f;                        //唤醒和时钟输出功能寄存器
sbit            T0CLKO = P3^5;                          //定时器 0 的时钟输出脚
#define         F38_4KHz    (65536-FOSC/2/38400)        //1T 模式
//#define       F38_4KHz    (65536-FOSC/2/12/38400)     //12T 模式

//-----------------------------------------------
void main()
{
    AUXR |= 0x80;               //定时器 0 为 1T 模式
//  AUXR &= ~0x80;              //定时器 0 为 12T 模式

    TMOD = 0x00;                //设置定时器为模式 0(16 位自动重装载)
    TMOD &= ~0x04;              //C/T0=0，对内部时钟进行时钟输出
//  TMOD |= 0x04;               //C/T0=1，对 T0 引脚的外部时钟进行时钟输出

    TL0 = F38_4KHz;             //初始化计时值
    TH0 = F38_4KHz >> 8;
    TR0 = 1;
    INT_CLKO = 0x01;            //使能定时器 0 的时钟输出功能
    while (1);                  //程序终止
}
```

2. 汇编程序：

```
/*-------------------------------------------------------------------------------------------*/
/*----演示 STC15F2K60S2 系列单片机定时器 0 的可编程时钟分频输出--------*/
```

```
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg5l.h>即可----*/
/*-------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz
AUXR        DATA    08EH            ;辅助特殊功能寄存器
INT_CLKO    DATA    08FH            ;唤醒和时钟输出功能寄存器

T0CLKO      BIT     P3.5            ;定时器 0 的时钟输出脚

F38_4KHz    EQU     0FF10H          ;38.4KHz(1T 模式下, 65536-18432000/2/38400)
;F38_4KHz   EQU     0FFECH          ;38.4KHz(12T 模式下, (65536-18432000/2/12/38400)

;-----------------------------------------------
    ORG     0000H
    LJMP    START                   ;复位入口
;---------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    ORL     AUXR, #80H              ;定时器 0 为 1T 模式
;   ANL     AUXR, #7FH              ;定时器 0 为 12T 模式

    MOV     TMOD, #00H              ;设置定时器为模式 0(16 位自动重装载)

    ANL     TMOD, #0FBH             ;C/T0=0, 对内部时钟进行时钟输出
;   ORL     TMOD, #04H              ;C/T0=1, 对 T0 引脚的外部时钟进行时钟输出

    MOV     TL0, #LOW F38_4KHz      ;初始化计时值
    MOV     TH0, #HIGH F38_4KHz
    SETB    TR0
    MOV     INT_CLKO, #01H          ;使能定时器 0 的时钟输出功能

    SJMP    $                       ;程序终止

;-----------------------------------------------
END
```

18.2.1.3 T0 的 16 位自动重装模式(软硬结合)模拟 10 位或 16 位 PWM 输出的程序(C 和汇编)

1.C 程序:

```
/*----STC15F2K60S2 系列定时器软件模拟 PWM 举例----------------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"
//#define       PWM6BIT     64                  //6-bit PWM  周期数
#define         PWM8BIT     256                 //8-bit PWM  周期数
//#define       PWM10BIT    1024                //10-bit PWM  周期数
//#define       PWM16BIT    65536               //16-bit PWM  周期数

#define         HIGHDUTY    64                  //高电平周期数(占空比 64/256=25%)
#define         LOWDUTY     (PWM8BIT-HIGHDUTY)    //低电平周期数

sfr             AUXR = 0x8e;                    //辅助寄存器
sfr             INT_CLKO = 0x8f;                //时钟输出控制寄存器
sbit            T0CLKO = P3^5;                  //定时器 0 的时钟输出口
bit             flag;

//定时器 0 中断服务程序
void tm0() interrupt 1
{
    flag = !flag;                               //反转 PWM 的输出标志
    if (flag)
    {
        TL0 = (65536-HIGHDUTY);                 //准备高电平的重载值
        TH0 = (65536-HIGHDUTY) >> 8;
    }
    else
    {
        TL0 = (65536-LOWDUTY);                  //准备低电平的重载值
        TH0 = (65536-LOWDUTY) >> 8;
    }
}

void main()
{
    AUXR = 0x80;                                //定时器 0 为 1T 模式
    INT_CLKO = 0x01;                            //使能定时器 0 的时钟输出功能
    TMOD &= 0xf0;                               //设置定时器 0 为模式 0(16 位自动重装载)
    TL0 = (65536-LOWDUTY);                      //初始化定时器初值和重装值
    TH0 = (65536-LOWDUTY) >> 8;
    T0CLKO = 1;                                 //初始化时钟输出脚(软 PWM 口)
```

```
    flag = 0;                                   //初始化标志位
    TR0 = 1;                                    //定时器 0 开始计时
    ET0 = 1;                                    //使能定时器 0 中断
    EA = 1;
    while (1);
}
```

2. 汇编程序：

```
/*-------------------------------------------------------------------------------------------------*/
/*----STC15F2K60S2 系列定时器软件模拟 PWM 举例----------------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------------*/
//假定测试芯片的工作频率为 18.432MHz
;PWM6BIT    EQU     64          ;6-bit PWM 周期数
PWM8BIT     EQU     256         ;8-bit PWM 周期数
;PWM10BIT   EQU     1024        ;10-bit PWM 周期数
;PWM16BIT   EQU     65536       ;16-bit PWM 周期数

HIGHDUTY    EQU     64          ;高电平周期数(占空比 64/256=25%)
LOWDUTY     EQU     (PWM8BIT-HIGHDUTY)  ;低电平周期数

AUXR        DATA    08EH        ;辅助寄存器
INT_CLKO    DATA    08FH        ;时钟输出控制寄存器
T0CLKO      BIT     P3.5        ;定时器 0 的时钟输出口
FLAG        BIT     20H.0
;-----------------------------------------
    ORG     0000H
    LJMP    START
    ORG     000BH
    LJMP    TM0_ISR
;-----------------------------------------
START:
    MOV     AUXR, #80H                  ;定时器 0 为 1T 模式
    MOV     INT_CLKO, #01H              ;使能定时器 0 的时钟输出功能
    ANL     TMOD, #0F0H                 ;设置定时器 0 为模式 0(16 位自动重装载)

    MOV     TL0, #LOW (65536-LOWDUTY)   ;初始化定时器初值和重装值
    MOV     TH0, #HIGH (65536-LOWDUTY)

    SETB    T0CLKO                      ;初始化时钟输出脚(软 PWM 口)
    CLR     FLAG                        ;初始化标志位
    SETB    TR0                         ;定时器 0 开始计时
    SETB    ET0                         ;使能定时器 0 中断
    SETB    EA
    SJMP    $
```

```
;-----------------------------------------------
;定时器 0 中断服务程序
TM0_ISR:
    CPL     FLAG                                ;反转 PWM 的输出标志
    JNB     FLAG, READYLOW
READYHIGH:
    MOV     TL0, #LOW (65536-HIGHDUTY)          ;准备高电平的重载值
    MOV     TH0, #HIGH (65536-HIGHDUTY)
    JMP     TM0ISR_EXIT
READYLOW:
    MOV     TL0, #LOW (65536-LOWDUTY)           ;准备低电平的重载值
    MOV     TH0, #HIGH (65536-LOWDUTY)
TM0ISR_EXIT:
    RETI
;-----------------------------------------------
END
```

18.2.1.4 T0 的 16 位自动重装载模式扩展为外部下降沿中断的测试程序(C 和汇编)

----利用 T0 的外部计数方式

;定时器 0 中断(下降沿中断)的测试程序，定时器/计数器 0 工作在计数模式中的 16 位自动重装载模式，定时器/计数器 0 工作载外部计数模式。

1.C 程序:

```
/*----STC15F2K60S2 系列 T0 扩展为外部下降沿中断举例-------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可---*/
//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"
#include "intrins.h"
//-----------------------------------------------
sfr     AUXR = 0x8e;                    //辅助寄存器
sbit    P10 = P1^0;
//-----------------------------------------------
//外不中断服务程序
void t0int() interrupt 1                //中断入口
{
    P10 = !P10;                         //将测试口取反
}

void main()
{
    AUXR = 0x80;                        //定时器 0 为 1T 模式
    TMOD = 0x04;                        //设置定时器 0 工作在 16 位自动重装载模式，同时为外部记数模式
    TH0 = 0xff;
```

```
    TL0 = 0xff;                     //设置定时器 0 初始值
    TR0 = 1;                        //定时器 0 开始工作
    ET0 = 1;                        //开定时器 0 中断
    EA = 1;
    while (1);
}
```

2. 汇编程序：

```
/*-------------------------------------------------------------------------------------------*/
/*----STC15F2K60S2 系列 T0 扩展为外部下降沿中断举例-------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可---*/
/*-------------------------------------------------------------------------------------------*/
//假定测试芯片的工作频率为 18.432MHz
AUXR        DATA    08EH            ;辅助寄存器
;---------------------------------------

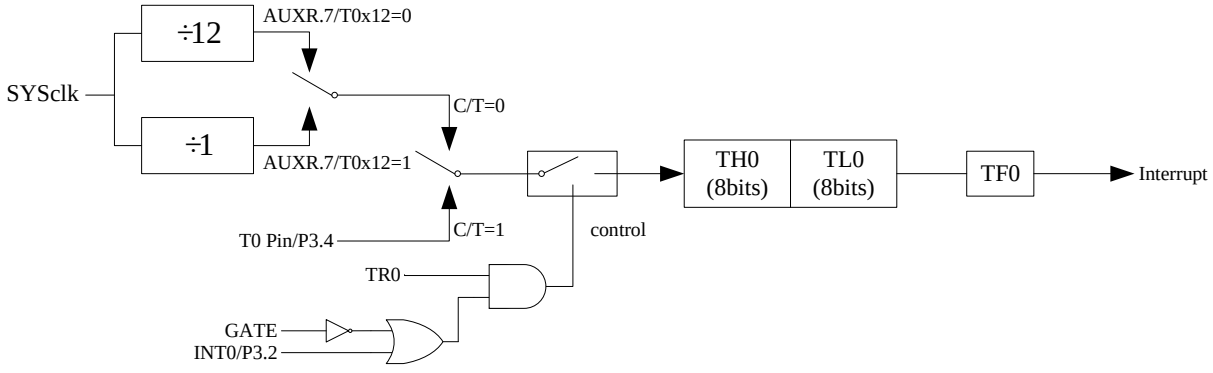
    ORG     0000H
    LJMP    START                   ;复位入口
    ORG     000BH                   ;中断入口

    LJMP    T0INT
;---------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    MOV     AUXR, #80H              ;定时器 0 为 1T 模式
    MOV     TMOD,   #04H            ;设置定时器 0 工作在 16 位自动重装载模式，
                                    ;同时为外部记数模式
    MOV     A,    #0FFH             ;设置定时器 0 初始值

    MOV     TL0, A
    MOV     TH0, A
    SETB    TR0                     ;定时器 0 开始工作
    SETB    ET0                     ;开定时器 0 中断
    SETB    EA
    SJMP    $
;---------------------------------------
;外部中断服务程序
T0INT:
    CPL     P1.0                    ;将测试口取反
    RETI
;---------------------------------------
END
```

18.2.2 模式 1（16 位不可重装载模式），不建议学习

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。

当 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。

STC15 系列单片机的定时器有两种计数速率：

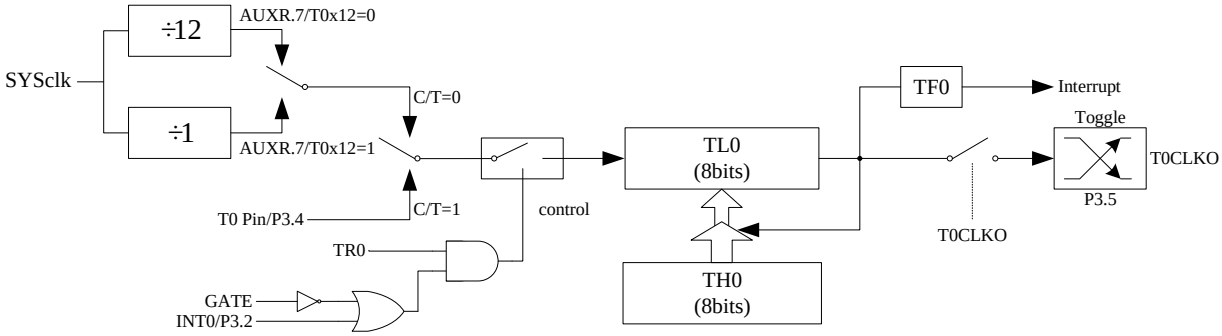
- 一种是 12T 模式，每 12 个时钟加 1，与传统 8051 单片机相同；
- 另外一种是 1T 模式，每个时钟加 1，速度是传统 8051 单片机的 12 倍。

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。

18.2.3 模式 2（8 位自动重装载模式），不建议学习

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

TL0 的溢出不仅置位 TF0，而且将 TH0 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=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）/（256-TH0）/2
  - ✓ T0 工作在 12T 模式（AUXR.7/T0x12=0）时的输出时钟频率=（SYSclk）/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

;定时器 0 中断（下降沿中断）的测试程序，定时器/计数器 0 工作在计数模式中的 8 位自动重装模式

1. C 程序：

```
/*-------------------------------------------------------------------------------------------*/
/*----STC15F2K60S2 系列 T0 扩展为外部下降沿中断举例-----------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------*/
//假定测试芯片的工作频率为 18.432MHz

#include "reg51.h"
#include "intrins.h"
//-----------------------------------------------
sfr     AUXR = 0x8e;                //辅助寄存器
sbit    P10 = P1^0;
//-----------------------------------------------
//外部中断服务程序
void t0int() interrupt 1            //中断入口
{
```

```
    P10 = !P10;                 //将测试口取反
}

void main()
{
    AUXR = 0x80;                //定时器 0 为 1T 模式
    TMOD = 0x06;                //设置定时器 0 为外部记数模式
    TH0 = TL0 = 0xff;           //设置定时器 0 初始值
    TR0 = 1;                    //定时器 0 开始工作
    ET0 = 1;                    //开定时器 0 中断
    EA = 1;
    while (1);
}
```

2. 汇编程序：

```
/*-------------------------------------------------------------------------------------------------*/
/*----STC15F2K60S2 系列 T0 扩展为外部下降沿中断举例----------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz
AUXR        DATA    08EH        ;辅助寄存器
;-----------------------------------------
    ORG     0000H
    LJMP    START               ;复位入口
    ORG     000BH               ;中断入口

    LJMP    T0INT
;-----------------------------------------
    ORG     0100H

START:
    MOV     SP, #3FH
    MOV     AUXR, #80H          ;定时器 0 为 1T 模式
    MOV     TMOD, #06H          ;设置定时器 0 为外部记数模式
    MOV     A, #0FFH            ;设置定时器 0 初始值
    MOV     TL0, A
    MOV     TH0, A
    SETB    TR0                 ;定时器 0 开始工作
    SETB    ET0                 ;开定时器 0 中断
    SETB    EA
    SJMP    $
;-----------------------------------------
;外部中断服务程序
T0INT:
    CPL P1.0                    ;将测试口取反
```

```
    RETI
;---------------------------------------
END
```

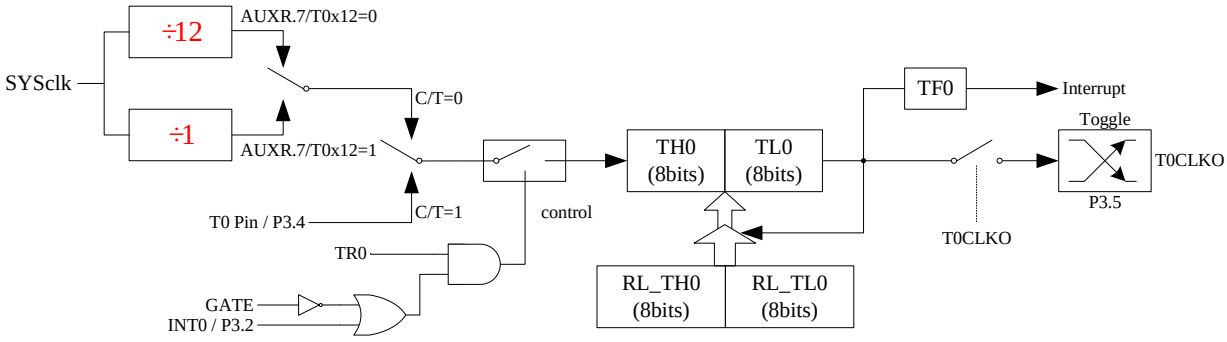
18.2.4 模式 3(不可屏蔽中断 16 位自动重装载,实时操作系统用节拍定时器)

对定时器/计数器 1，在模式 3 时，定时器 1 停止计数，效果与将 TR1 设置为 0 相同。

对定时器/计数器 0，其工作模式 3 与工作模式 0 是一样的（下图是定时器模式 3 的原理图，与模式 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

那么当定时器/计数器 0 工作在模式 3 时，如何打开定时器/计数器 0 的中断呢?

下面的语句可以令定时器/计数器 0 工作在模式 3（不可屏蔽中断的 16 位自动重装在模式）并打开定时器/计数器 0 的中断（此时该中断是最高优先级，任何中断都不能屏蔽它）。

C 语言设置：

```
    TMOD = 0x11;          //设置定时器为模式 3（不可屏蔽中断的 16 位自动重装载）
    TR0 = 1;              //定时器 0 开始计时
//  EA = 1;               //定时器 0 工作在模式 3（不可屏蔽中断的 16 位自动重装载/模式）时,
                          //不需要使能总中断允许位 EA
    ET0 = 1;              //使能定时器 0 工作在模式 3（不可屏蔽中断的 16 位自动重装/载模式）时的中断
```

汇编语言设置：

```
    MOV   TMOD, #11H      ;设置定时器为模式 3（不可屏蔽中断的 16 位自动重装载）
    SETB  TR0             ;定时器 0 开始计时
//  SETB  EA              ;定时器 0 工作在模式 3（不可屏蔽中断的 16 位自动重装载/模式）时,
                          ;不需要使能总中断允许位 EA
```

```
    SETB    ET0                 ;使能定时器 0 工作在模式 3（不可屏蔽中断的 16 位自动重装/载模式）时的中断
```

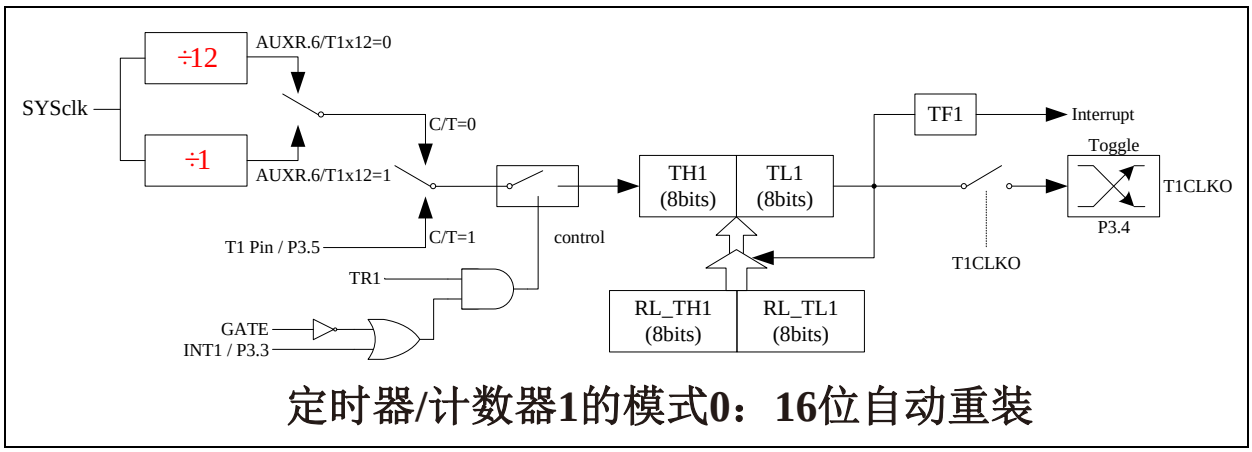
【注意】: 当定时器/计数器 0 工作在模式 3（不可屏蔽中断的 16 位自动重装载模式）时，不需要允许 EA/IE.7（总中断使能位），只需允许 ET0/IE.1（定时器/计数器 0 中断允许位），就能打开定时器/计数器 0 的中断，此模式下的定时器/计数器 0 中断与总中断使能位 EA 无关。一旦此模式下的定时器/计数器 0 中断被打开后，该定时器/计数器 0 中断优先级就是最高的，它不能被其它任何中断所打断（不管是比定时器/计数器 0 中断优先级低的中断还是比其优先级高的中断，都不能打断此时的定时器/计数器 0 中断），而且该中断打开后既不受 EA/IE.7 控制也不再受 ET0 控制了，清零 EA 或 ET0 都不能关闭此中断。

18.3 定时器/计数器 1 工作模式

通过对寄存器 TMOD 中的 M1（TMOD.5）、M0（TMOD.4）的设置，定时器/计数器 1 有 3 种不同的工作模式。

18.3.1 模式 0（16 位自动重装载模式）及测试程序，建议只学习此模式足矣

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

当 GATE=0（TMOD.7）时，如 TR1=1，则定时器计数。

当 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。

STC15 系列单片机的定时器有两种计数速率：

- 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 2 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 中。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 0（TMOD[5:4]/[M1, M0]=00B）时，[TL1, TH1]的溢出不仅置位 TF1，而且会自动将[RL_TL1, RL_TH1]的内容重新装入[TL1, TH1]。

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=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）/（65536 - [RL_TH1, RL_TL1]）/ 2
  - ✓ T1 工作在 12T 模式（AUXR.6/T1x12=0）时的输出时钟率=（SYSclk / 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

18.3.1.1 定时器 1 的 16 位自动重装载模式的测试程序（C 和汇编）

1.C 程序:

```
/*----演示 STC15 系列单片机定时器 1 的 16 位自动重装载模式----------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可------*/
//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"
typedef unsigned char BYTE;
typedef unsigned int WORD;
//-----------------------------------------
#define     FOSC    18432000L
#define     T1MS    (65536-FOSC/1000)       //1T 模式，18.432MHz
//#define     T1MS    (65536-FOSC/12/1000)    //12T 模式，18.432MHz
sfr         AUXR = 0x8e;                    //Auxiliary register
sbit        P10 = P1^0;
//-----------------------------------------
/* Timer1 interrupt routine */
void tm1_isr() interrupt 3 using 1
{
    P10 = ! P10;                            //将测试口取反
}
//-----------------------------------------
void main()
{
    AUXR |= 0x40;                           //定时器 1 为 1T 模式
    // AUXR &= 0xdf;                        //定时器 1 为 12T 模式
    TMOD = 0x00;                            //设置定时器为模式 0(16 位自动重装载)
```

```
    TL1 = T1MS;                         //初始化计时值
    TH1 = T1MS >> 8;
    TR1 = 1;                            //定时器 1 开始计时
    ET1 = 1;                            //使能定时器 1 中断
    EA = 1;
    while (1);
}
```

2. 汇编程序：

```
/*-------------------------------------------------------------------------------------------------*/
/*----演示 STC15 系列单片机定时器 1 的 16 位自动重装载模式----------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可------*/
/*-------------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz

AUXR        DATA    08EH            ;辅助特殊功能寄存器
;-----------------------------------------
T1MS        EQU     0B800H          ;1T 模式的 1ms 定时值(65536-18432000/1000)
;T1MS       EQU     0FA00H          ;12T 模式的 1ms 定时值(65536-18432000/1000/12)
;-----------------------------------------
    ORG     0000H
    LJMP    START                   ;复位入口
    ORG     001BH                   ;中断入口
    LJMP    T1INT
;-----------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    ORL     AUXR, #40H              ;定时器 1 为 1T 模式
;   ANL     AUXR, #0DFH             ;定时器 1 为 12T 模式
    MOV     TMOD, #00H              ;设置定时器为模式 0(16 位自动重装载)
    MOV     TL1, #LOW T1MS          ;初始化计时值
    MOV     TH1, #HIGH T1MS
    SETB    TR1
    SETB    ET1                     ;使能定时器 1 中断
    SETB    EA
    SJMP    $                       ;程序终止
;---------------------------------------
;中断服务程序
T1INT:
    CPL P1.0                        ;将测试口取反
    RETI
;---------------------------------------
END
```

18.3.1.2 定时器 1 对系统时钟或外部引脚 T1 的时钟输入进行可编程分频输出的测试程序

----定时器 1 工作在 16 位自动重装载模式

下面是定时器 1 工作在 16 位重装模式时对内部系统时钟或外部引脚 T1/P3.5 的时钟输入进行可编程时钟分频输出的程序举例（C 和汇编）：

1.C 程序:

```
/*----演示 STC15 系列单片机定时器 1 的可编程时钟分频输出----------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可.----*/
//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"
typedef     unsigned char     BYTE;
typedef     unsigned int      WORD;
#define     FOSC              18432000L
//-----------------------------------------------
sfr         AUXR = 0x8e;                          //辅助特殊功能寄存器
sfr         INT_CLKO = 0x8f;                      //唤醒和时钟输出功能寄存器
sbit        T1CLKO = P3^4;                        //定时器 1 的时钟输出脚
#define     F38_4KHz (65536-FOSC/2/38400)         //1T 模式
//#define   F38_4KHz (65536-FOSC/2/12/38400)      //12T 模式
//-----------------------------------------------
void main()
{
    AUXR |= 0x40;                                 //定时器 1 为 1T 模式
//  AUXR &= ~0x40;                                //定时器 1 为 1 为 T 模式
    TMOD = 0x00;                                  //设置定时器为模式 1(16 位自动重装载)

    TMOD &= ~0x40;                                //C/T1=0, 对内部时钟进行时钟输出
    // TMOD |= 0x40;                              //C/T1=1, 对 T1 引脚的外部时钟进行时钟输出
    TL1 = F38_4KHz;                               //初始化计时值
    TH1 = F38_4KHz >> 8;
    TR1 = 1;
    INT_CLKO = 0x02;                              //使能定时器 1 的时钟输出功能
    while (1);                                    //程序终止
}
```

2. 汇编程序:

```
/*-------------------------------------------------------------------------------------------------*/
/*----演示 STC15 系列单片机定时器 1 的可编程时钟分频输出----------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可.----*/
/*-------------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz

AUXR        DATA    08EH                      ;辅助特殊功能寄存器
```

```
INT_CLKO    DATA    08FH                ;唤醒和时钟输出功能寄存器
T1CLKO      BIT     P3.4                ;定时器 1 的时钟输出脚

F38_4KHz    EQU     0FF10H              ;38.4KHz(1T 模式下, 65536-18432000/2/38400)
;F38_4KHz   EQU     0FFECH              ;38.4KHz(12T 模式下, (65536-18432000/2/12/38400)

    ORG     0000H
    LJMP    START                       ;复位入口
;-----------------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    ORL     AUXR, #40H                  ;定时器 1 为 1T 模式
;   ANL     AUXR, #0BFH                 ;定时器 1 为 12T 模式
    MOV     TMOD, #00H                  ;设置定时器为模式 0(16 位自动重装载)
    ANL     TMOD, #0BFH                 ;C/T1=0, 对内部时钟进行时钟输出
;   ORL     TMOD, #40H                  ;C/T1=1, 对 T1 引脚的部时钟进行时钟输出
    MOV     TL1, #LOW F38_4KHz          ;初始化计时值
    MOV     TH1, #HIGH F38_4KHz
    SETB    TR1
    MOV     INT_CLKO, #02H              ;使能定时器 1 的时钟输出功能
    SJMP    $                           ;程序终止

;-----------------------------------------------
    END
```

18.3.1.3 定时器 1 模式 0(16 位自动重载模式)作串口 1 波特率发生器的测试程序(C 和汇编)

1.C 程序:

```
/*----STC15F2K60S2 系列定时器 1 用作串口 1 的波特率发生器举例----------------*/
/*----在 Keil C 开发环境中选择  Intel 8052 编译，头文件包含<reg51.h>即可-----*/
//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"
#include "intrins.h"
typedef     unsigned char   BYTE;
typedef     unsigned int    WORD;

#define     FOSC            18432000L       //系统频率
#define     BAUD            115200          //串口波特率

#define     NONE_PARITY     0               //无校验
#define     ODD_PARITY      1               //奇校验
#define     EVEN_PARITY     2               //偶校验
#define     MARK_PARITY     3               //标记校验
```

```
#define     SPACE_PARITY    4               //空白校验
#define     PARITYBIT       EVEN_PARITY     //定义校验位

sfr         AUXR = 0x8e;                    //辅助寄存器
sbit        P22 = P2^2;
bit         busy;

void SendData(BYTE dat);
void SendString(char *s);

void main()
{
#if (PARITYBIT == NONE_PARITY)
    SCON = 0x50;                            //8 位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
    SCON = 0xda;                            //9 位可变波特率,校验位初始为 1
#elif (PARITYBIT == SPACE_PARITY)
    SCON = 0xd2;                            //9 位可变波特率,校验位初始为 0
#endif
    AUXR = 0x40;                            //定时器 1 为 1T 模式
    TMOD = 0x00;                            //定时器 1 为模式 0(16 位自动重载)
    TL1 = (65536 - (FOSC/32/BAUD));         //设置波特率重装值
    TH1 = (65536 - (FOSC/32/BAUD))>>8;
    TR1 = 1;                                //定时器 1 开始启动
    ES = 1;                                 //使能串口中断
    EA = 1;
    SendString("STC15F2K60S2\r\nUart Test !\r\n");
    while(1);
}

/*----------------------------UART 中断服务程序-----------------------------*/
void Uart() interrupt 4 using 1
{
    if (RI)
    {
        RI = 0;                             //清除 RI 位
        P0 = SBUF;                          //P0 显示串口数据
        P22 = RB8;                          //P2.2 显示校验位
    }
    if (TI)
    {
        TI = 0;                             //清除 TI 位
        busy = 0;                           //清忙标志
    }
}
```

```
/*----------------------------发送串口数据----------------------------*/
void SendData(BYTE dat)
{
    while (busy);                   //等待前面的数据发送完成
    ACC = dat;                      //获取校验位 P (PSW.0)
    if (P)                          //根据 P 来设置校验位
    {
#if (PARITYBIT == ODD_PARITY)
    TB8 = 0;                        //设置校验位为 0
#elif (PARITYBIT == EVEN_PARITY)
    TB8 = 1;                        //设置校验位为 1
#endif
    }
    else
    {
#if (PARITYBIT == ODD_PARITY)
    TB8 = 1;                        //设置校验位为 1
#elif (PARITYBIT == EVEN_PARITY)
    TB8 = 0;                        //设置校验位为 0
#endif
    }
    busy = 1;
    SBUF = ACC;                     //写数据到 UART 数据寄存器
}

/*----------------------------发送字符串----------------------------*/
void SendString(char *s)
{
    while (*s)                      //检测字符串结束标志
    {
        SendData(*s++);             //发送当前字符
    }
}
```

2. 汇编程序：

```
/*-------------------------------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 定时器 1 用作串口 1 的波特率发生器举例------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-----*/
/*-------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz

#define     NONE_PARITY     0       ;无校验
#define     ODD_PARITY      1       ;奇校验
#define     EVEN_PARITY     2       ;偶校验
#define     MARK_PARITY     3       ;标记校验
```

```
#define     SPACE_PARITY    4               ;空白校验
#define     PARITYBIT       EVEN_PARITY     ;定义校验位
;-----------------------------------------
AUXR        EQU             08EH            ;辅助寄存器
BUSY        BIT             20H.0           ;忙标志位
;-----------------------------------------
    ORG     0000H
    LJMP    START
    ORG     0023H
    LJMP    UART_ISR
;-----------------------------------------
    ORG     0100H
START:
    CLR     BUSY
    CLR     EA
    MOV     SP, #3FH
#if (PARITYBIT == NONE_PARITY)
    MOV     SCON, #50H                  ;8 位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
    MOV     SCON, #0DAH                 ;9 位可变波特率,校验位初始为 1
#elif (PARITYBIT == SPACE_PARITY)
    MOV     SCON, #0D2H                 ;9 位可变波特率,校验位初始为 0
#endif
;------------------------------
    MOV     AUXR, #40H                  ;定时器 1 为 1T 模式
    MOV     TMOD, #00H                  ;定时器 1 为模式 0(16 位自动重载)
    MOV     TL1, #0FBH                  ;设置波特率重装值(65536-18432000/32/115200)
    MOV     TH1, #0FFH
    SETB    TR1                         ;定时器 1 开始运行
    SETB    ES                          ;使能串口中断
    SETB    EA
    MOV     DPTR, #TESTSTR              ;发送测试字符串
    LCALL   SENDSTRING
    SJMP    $
;-----------------------------------------
TESTSTR:
    DB      "STC15F2K60S2 Uart1 Test !",0DH,0AH,0
;---------------------------UART 中断服务程序----------------------------
UART_ISR:
    PUSH    ACC
    PUSH    PSW
    JNB     RI, CHECKTI                 ;检测 RI 位
    CLR     RI                          ;清除 RI 位
    MOV     P0, SBUF                    ;P0 显示串口数据
    MOV     C, RB8
    MOV     P2.2, C                     ;P2.2 显示校验位
```

```
CHECKTI:
    JNB     TI, ISR_EXIT                ;检测 TI 位
    CLR     TI                          ;清除 TI 位
    CLR     BUSY                        ;清忙标志
ISR_EXIT:
    POP     PSW
    POP     ACC
    RETI
;----------------------------发送串口数据----------------------------
SENDDATA:
    JB      BUSY, $                     ;等待前面的数据发送完成
    MOV     ACC, A                      ;获取校验位 P (PSW.0)
    JNB     P, EVEN1INACC               ;根据 P 来设置校验位
ODD1INACC:
#if (PARITYBIT == ODD_PARITY)
    CLR     TB8                         ;设置校验位为 0
#elif (PARITYBIT == EVEN_PARITY)
    SETB    TB8                         ;设置校验位为 1
#endif
    SJMP    PARITYBITOK
EVEN1INACC:
#if (PARITYBIT == ODD_PARITY)
    SETB    TB8                         ;设置校验位为 1
#elif (PARITYBIT == EVEN_PARITY)
    CLR     TB8                         ;设置校验位为 0
#endif
PARITYBITOK:                            ;校验位设置完成
    SETB    BUSY
    MOV     SBUF, A                     ;写数据到 UART 数据寄存器
    RET
;----------------------------发送字符串----------------------------
SENDSTRING:
    CLR     A
    MOVC    A, @A+DPTR                  ;读取字符
    JZ      STRINGEND                   ;检测字符串结束标志
    INC     DPTR                        ;字符串地址+1
    LCALL   SENDDATA                    ;发送当前字符
    SJMP    SENDSTRING
STRINGEND:
    RET
;----------------------------------------
END
```

18.3.1.4 T1 的 16 位自动重装载模式扩展为外部下降沿中断的测试程序(C 和汇编)

----利用 T1 的外部计数方式

;定时器 1 中断(下降沿中断)的测试程序，定时器/计数器 1 工作在计数模式中的 16 位自动重装载模式

1.C 程序:

```
/*----STC15F2K60S2 系列 T1 扩展为外部下降沿中断举例--------------------------*/
/*----在 Keil C 开发环境中选择  Intel 8052 编译，头文件包含<reg51.h>即可----*/
//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"
#include "intrins.h"
//-----------------------------------------------
sfr         AUXR = 0x8e;                //辅助寄存器
sbit        P10 = P1^0;
//-----------------------------------------------
//外部中断服务程序
void t1int() interrupt 3                //中断入口
{
    P10 = !P10;                         //将测试口取反
}

void main()
{
    AUXR = 0x40;                        //定时器 1 为 1T 模式
    TMOD = 0x40;                        //设置定时器 1 为外部记数模式，工作在 16 位自动重装载模式
    TH1 = TL1 = 0xff;                   //设置定时器 1 初始值
    TR1 = 1;                            //定时器 1 开始工作
    ET1 = 1;                            //开定时器 1 中断
    EA = 1;
    while (1);
}
```

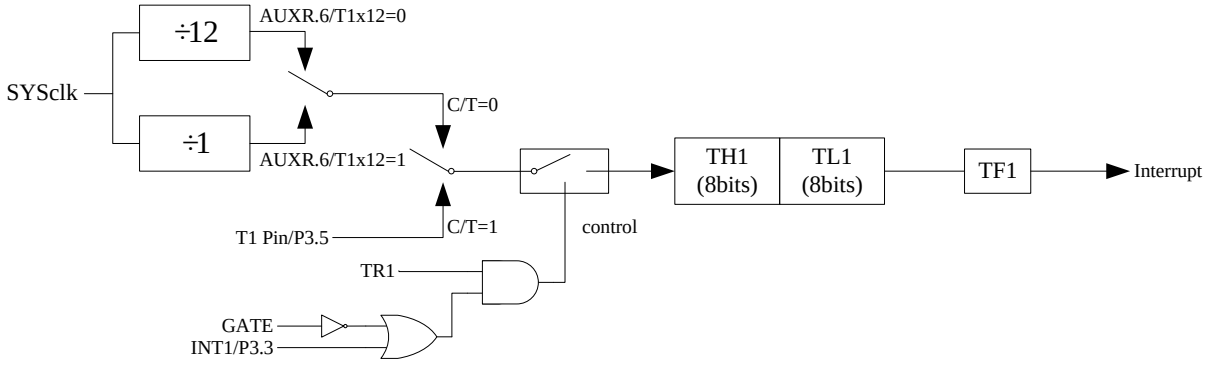
2. 汇编程序:

```
/*-------------------------------------------------------------------------------------------------*/
/*----STC15F2K60S2 系列 T1 扩展为外部下降沿中断举例--------------------------*/
/*----在 Keil C 开发环境中选择  Intel 8052 编译，头文件包含<reg51.h>即可----*/
/*-------------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz
AUXR        DATA    08EH            ;辅助寄存器
;---------------------------------------
    ORG     0000H
    LJMP    START                   ;复位入口
    ORG     001BH                   ;中断入口
    LJMP    T1INT
```

```
;---------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    MOV     AUXR, #40H          ;定时器 1 为 1T 模式
    MOV     TMOD, #40H          ;设置定时器 1 为外部记数模式，工作在 16 位重装载模式
    MOV     A, #0FFH            ;设置定时器 1 初始值
    MOV     TL1, A
    MOV     TH1, A
    SETB    TR1                 ;定时器 1 开始工作
    SETB    ET1                 ;开定时器 1 中断
    SETB    EA
    SJMP    $
;---------------------------------------
;外部中断服务程序
T1INT:
    CPL     P1.0                ;将测试口取反
    RETI
;---------------------------------------
END
```

18.3.2 模式 1（16 位不可重装载模式），不建议学习

此模式下定时器/计数器 1 作为 16 位定时器，如下图所示。



定时器/计数器1的模式1：16位不可重装载

此模式下，定时器 1 配置为 16 位不可重装在模式，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。

当 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。

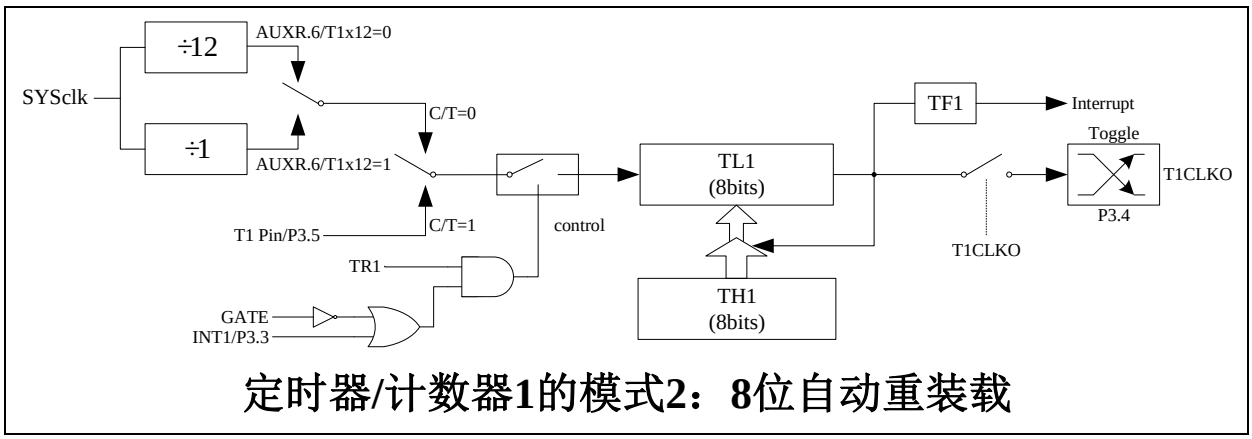
STC15 系列单片机的定时器有两种计数速率：

- 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。

18.3.3 模式 2（8 位自动重装载模式），不建议学习

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。



TL1 的溢出不仅置位 TF1，而且将 TH1 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=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)/(256-TH1)/2
  - ✓ T1 工作在 12T 模式(AUXR.6/T1x12=0)时的输出时钟频率=(SYSclk)/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

18.3.3.1 定时器 1 模式 2(8 位自动重载模式)作串口 1 波特率发生器的测试程序(C 和汇编)

1. C 程序：

```
/*-------------------------------------------------------------------------------------------*/
/*----STC15F2K60S2 系列定时器 1 用作串口 1 的波特率发生器举例-------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可----*/
/*-------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"
#include "intrins.h"
```

```
typedef     unsigned char   BYTE;
typedef     unsigned int    WORD;

#define     FOSC            18432000L       //系统频率
#define     BAUD            115200          //串口波特率
#define     NONE_PARITY     0               //无校验
#define     ODD_PARITY      1               //奇校验
#define     EVEN_PARITY     2               //偶校验
#define     MARK_PARITY     3               //标记校验
#define     SPACE_PARITY    4               //空白校验
#define     PARITYBIT       EVEN_PARITY     //定义校验位

sfr         AUXR = 0x8e;                    //辅助寄存器
sbit        P22 = P2^2;
bit         busy;

void SendData(BYTE dat);
void SendString(char *s);
void main()
{
#if (PARITYBIT == NONE_PARITY)
    SCON = 0x50;                            //8 位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
    SCON = 0xda;                            //9 位可变波特率,校验位初始为 1
#elif (PARITYBIT == SPACE_PARITY)
    SCON = 0xd2;                            //9 位可变波特率,校验位初始为 0
#endif

    AUXR = 0x40;                            //定时器 1 为 1T 模式
    TMOD = 0x20;                            //定时器 1 为模式 2(8 位自动重载)
    TL1 = (256 - (FOSC/32/BAUD));           //设置波特率重装值
    TH1 = (256 - (FOSC/32/BAUD));
    TR1 = 1;                                //定时器 1 开始工作
    ES = 1;                                 //使能串口中断
    EA = 1;

    SendString("STC15F2K60S2\r\nUart Test !\r\n");

    while(1);
}
/*----------------------------UART 中断服务程序-----------------------------*/
void Uart() interrupt 4 using 1
{
    if (RI)
    {
        RI = 0;                             //清除 RI 位
```

```
            P0 = SBUF;                          //P0 显示串口数据
            P22 = RB8;                          //P2.2 显示校验位
        }
        if (TI)
        {
            TI = 0;                             //清除 TI 位
            busy = 0;                           //清忙标志
        }
}
/*----------------------------发送串口数据----------------------------*/
void SendData(BYTE dat)
{
        while (busy);                           //等待前面的数据发送完成
        ACC = dat;                              //获取校验位 P (PSW.0)

        if (P)                                  //根据 P 来设置校验位
        {
#if (PARITYBIT == ODD_PARITY)
        TB8 = 0;                                //设置校验位为 0
#elif (PARITYBIT == EVEN_PARITY)
        TB8 = 1;                                //设置校验位为 1
#endif
        }
        else
        {
#if (PARITYBIT == ODD_PARITY)
        TB8 = 1;                                //设置校验位为 1
#elif (PARITYBIT == EVEN_PARITY)
        TB8 = 0;                                //设置校验位为 0
#endif
        }

        busy = 1;
        SBUF = ACC;                             //写数据到 UART 数据寄存器
}

/*----------------------------发送字符串----------------------------*/
void SendString(char *s)
{
        while (*s)                              //检测字符串结束标志
        {
            SendData(*s++);                     //发送当前字符
        }
}
```

2. 汇编程序：

```
/*------------------------------------------------------------------------------------------------- */
/*----STC15F2K60S2 系列定时器 1 用作串口 1 的波特率发生器举例-------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可----*/
/*-------------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz

#define     NONE_PARITY     0           ;无校验
#define     ODD_PARITY      1           ;奇校验
#define     EVEN_PARITY     2           ;偶校验
#define     MARK_PARITY     3           ;标记校验
#define     SPACE_PARITY    4           ;空白校验
#define     PARITYBIT       EVEN_PARITY ;定义校验位
;---------------------------------------
AUXR    EQU     08EH                    ;辅助寄存器
BUSY    BIT     20H.0                   ;忙标志位

;---------------------------------------
    ORG     0000H
    LJMP    START
    ORG     0023H
    LJMP    UART_ISR
;---------------------------------------
    ORG     0100H

START:
    CLR     BUSY
    CLR     EA
    MOV     SP, #3FH

#if (PARITYBIT == NONE_PARITY)
    MOV     SCON, #50H                  ;8 位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
    MOV     SCON, #0DAH                 ;9 位可变波特率,校验位初始为 1
#elif (PARITYBIT == SPACE_PARITY)
    MOV     SCON, #0D2H                 ;9 位可变波特率,校验位初始为 0
#endif
;-------------------------------
    MOV     AUXR, #40H                  ;定时器 1 为 1T 模式
    MOV     TMOD, #20H                  ;定时器 1 为模式 2(8 位自动重载)
    MOV     TL1, #0FBH                  ;设置波特率重装值(256-18432000/32/115200)
    MOV     TH1, #0FBH
    SETB    TR1                         ;定时器 1 开始运行
    SETB    ES                          ;使能串口中断
    SETB    EA
```

```
    MOV     DPTR, #TESTSTR                  ;发送测试字符串

    LCALL   SENDSTRING
    SJMP    $
;----------------------------------------
TESTSTR:
    DB      "STC15F2K60S2 Uart1 Test !",0DH,0AH,0
;/*---------------------------UART 中断服务程序---------------------------*/
UART_ISR:
    PUSH    ACC
    PUSH    PSW
    JNB     RI, CHECKTI                 ;检测 RI 位
    CLR     RI                          ;清除 RI 位
    MOV     P0, SBUF                    ;P0 显示串口数据
    MOV     C, RB8
    MOV     P2.2, C                     ;P2.2 显示校验位

CHECKTI:
    JNB     TI, ISR_EXIT                ;检测 TI 位
    CLR     TI                          ;清除 TI 位
    CLR     BUSY                        ;清忙标志

ISR_EXIT:
    POP     PSW
    POP     ACC
    RETI
;/*---------------------------发送串口数据---------------------------*/
SENDDATA:
    JB      BUSY, $                     ;等待前面的数据发送完成
    MOV     ACC, A                      ;获取校验位 P (PSW.0)
    JNB     P, EVEN1INACC               ;根据 P 来设置校验位

ODD1INACC:
#if (PARITYBIT == ODD_PARITY)
    CLR     TB8                         ;设置校验位为 0
#elif (PARITYBIT == EVEN_PARITY)
    SETB    TB8                         ;设置校验位为 1
#endif

    SJMP    PARITYBITOK

EVEN1INACC:
#if (PARITYBIT == ODD_PARITY)
    SETB    TB8                         ;设置校验位为 1
#elif (PARITYBIT == EVEN_PARITY)
    CLR     TB8                         ;设置校验位为 0
```

```
#endif

PARITYBITOK:                            ;校验位设置完成
    SETB    BUSY
    MOV     SBUF, A                     ;写数据到 UART 数据寄存器
    RET
;/*----------------------------发送字符串----------------------------*/
SENDSTRING:
    CLR     A
    MOVC    A, @A+DPTR                  ;读取字符
    JZ      STRINGEND                   ;检测字符串结束标志
    INC     DPTR                        ;字符串地址+1
    LCALL   SENDDATA                    ;发送当前字符
    SJMP    SENDSTRING

STRINGEND:
    RET
;----------------------------------------
END
```

18.3.3.2 T1 的 8 位自动重装载模式扩展为外部下降沿中断的测试程序(C 和汇编)

;定时器 1 中断(下降沿中断)的测试程序，定时器/计数器 1 工作在计数模式中的 8 位自动重装载模式

1. C 程序：

```
/*-------------------------------------------------------------------------------------------------*/
/*----STC15F2K60S2 系列 T1 扩展为外部下降沿中断举例----------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------------*/
//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"
#include "intrins.h"
//-----------------------------------------------
sfr         AUXR = 0x8e;                //辅助寄存器
sbit        P10 = P1^0;
//-----------------------------------------------
//外部中断服务程序
void t1int() interrupt 3                //中断入口
{
    P10 = !P10;                         //将测试口取反
}

void main()
{
    AUXR = 0x40;                        //定时器 1 为 1T 模式
```

```
    TMOD = 0x60;                //设置定时器 1 为外部记数模式，工作在 8 位自动重装载模式
    TH1 = TL1 = 0xff;           //设置定时器 1 初时值
    TR1 = 1;                    //定时器 1 开始工作
    ET1 = 1;                    //开定时器 1 中断
    EA = 1;
    while (1);
}
```

2. 汇编程序：

```
/*-------------------------------------------------------------------------------------------------*/
/*----STC15F2K60S2 系列 T1 扩展为外部下降沿中断举例-----------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz
AUXR        DATA    08EH        ;辅助寄存器
;---------------------------------------
    ORG     0000H
    LJMP    START               ;复位入口
    ORG     001BH               ;中断入口
    LJMP    T1INT
;---------------------------------------
    ORG     0100H

START:
    MOV     SP, #3FH
    MOV     AUXR, #40H          ;定时器 1 为 1T 模式
    MOV     TMOD, #60H          ;设置定时器 1 为外部记数模式
    MOV     A, #0FFH            ;设置定时器 1 初时值
    MOV     TL1, A
    MOV     TH1, A
    SETB    TR1                 ;定时器 1 开始工作
    SETB    ET1                 ;开定时器 1 中断
    SETB    EA
    SJMP    $
;---------------------------------------
;外部中断服务程序
T1INT:
    CPL     P1.0                ;将测试口取反
    RETI
;---------------------------------------
END
```

18.4 古老的 Intel 8051 单片机定时器 0/1 应用举例

【例 1】定时/计数器应用编程，设某应用系统，选择定时/计数器 1 定时模式，定时时间 Tc = 10ms，主频频率为 12MHz，每 10ms 向主机请求处理。选定工作方式 1。计算得计数初值：低 8 位初值为 F0H，高 8 位初值为 D8H。

(1)初始化程序

所谓初始化，一般在主程序中根据应用要求对定时/计数器进行功能选择及参数设定等预置程序，本例初始化程序如下:

```
START：
    …
    MOV     SP,  #60H           ;设置堆栈区域
    MOV     TMOD, #10H          ;选择 T1、定时模式，工作方式 1
    MOV     TH1, #0D8H          ;设置高字节计数初值
    MOV     TL1, #0F0H          ;设置低字节计数初值
    SETB    EA                  ;开中断
    SETB    ET1
    …                           ;其他初始化程序
    SETB    TR1                 ;启动 T1 开始计时
    …                           ;继续主程序
```

（2）中断服务程序

```
INTT1：
    PUSH    A
    PUSH    DPL                 ;现场保护
    PUSH    DPH
    …
    MOV     TL1, #0F0H          ;重新置初值
    MOV     TH1, #0D8H
    …                           ;中断处理主体程序
    POP     DPH                 ;现场恢复
    POP     DPL
    POP     A
    RETI                        ;返回
```

这里展示了中断服务子程序的基本格式。STC15 系列单片机的中断属于天量中断，每一个矢量中断源只留有 8 个字节单元，一般是不够用的，常需用转移指令转到真正的中断服务子程序区去执行。

【例 2】利用定时/计数器 0 或定时/计数器 1 的 Tx 端口改造成外部中断源输入端口的应用设计。在某些应用系统中常会出现原有的两个外部中断源 INT0 和 INT1 不够用，而定时/计数器有多余，则可将 Tx 用于增加的外部中断源。现选择定时/计数器 1 为对外部事件计数模式工作方式 2(自动再装入)，设置计数初值为 FFH，则 T1 端口输入一个负跳变脉冲，计数器即回 0 溢出，置位对应的中断请求标志位 TF1 为 1，向主机请求中断处理，从而达到了增加一个外部中断源的目的。应用定时/计数器 1(T1)的中断天量转入中断服务程序处理。其程序示例如下:

(1)主程序段:

```
    ORG     0000H
    AJMP    START           ;转主程序
    ORG     001BH
    LJMP    INTER           ;转 T1 中断服务程序
    …
    ORG     0100            ;主程序入口
START:
    …
    MOV     SP, #60H        ;设置堆栈区
    MOV     TMOD, #60H      ;设置定时/计数器 1，计数方式 2
    MOV     TL1, #0FFH      ;设置计数常数
    MOV     TH1,#0FFH
    SETB    EA              ;开中断
    SETB    ET1             ;开定时/计数器 1 中断
    SETB    TR1             ;启动定时/计数器 1 计数
```

(2)中断服务程序(具体处理程序略)

```
    ORG     1000H
INTER:
    PUSH    A
    PUSH    DPL             ;现场入栈保护
    PUSH    DPH
    …                       ;中断处理主体程序
    POP     DPH             ;现场出栈复原
    POP     DPL
    POP     A
    RETI                    ;返回
```

这是中断服务程序的基本格式。

【例 3】某应用系统需通过 P1.0 和 P1.1 分别输出周期为 200rs 和 400μs 的方波。为此，系统选用定时器/计数器 0(T0)，定时方式 3，主频为 6MHz，TP=2μs，经计算得定时常数为 9CH 和 38H.

本例程序段编制如下:

(1)初始化程序段

```
PLT0:
    MOVT    MOD, #03H       ;设置 T0 定时方式 3
    MOV     TL0, #9CH       ;设置 TL0 初值
    MOV     TH0, #38H       ;设置 TH0 初值
    SETB    EA              ;开中断
    SETB    ET0
    SETB    ET1
    SETB    TR0             ;启动
    SETB    TR1
```

(2)中断服务程序段

1)

```
INT0P:
```

```
    MOV     TL0, #9CH       ;重新设置初值中
    CPL     P1.0            ;对 P1.0 输出信号取反
    RETI                    ;返回
2)
INT1P:
    …
    MOV     TH0, #38H       ;重新设置初值
    CPL     P1.1            ;对 P1.1 输出信号取反
    RETI                    ;返回
```

在实际应用中应注意的问题如下。

(1)定时/计数器的实时性

定时/计数器启动计数后，当计满回 0 溢出向主机请求中断处理，由内部硬件自动进行。但从回 0 溢出请求中断到主机响应中断并作出处理存在时间延迟，且这种延时随中断请求时的现场环境的不同而不同，一般需延时 3 个机器周期以上，这就给实时处理带来误差。大多数应用场合可忽略不计，但对某些要求实时性苛刻的场合，应采用补偿措施。

这种由中断响应引起的时间延时，对定时/计数器工作于方式 0 或 1 而言有两种含义：

- 一是由于中断响应延时而引起的实时处理的误差；
- 二是如需多次且连续不间断地定时/计数，由于中断响应延时，则在中断服务程序中再置计数初值时已延误了若干个计数值而引起误差，特别是用于定时就更明显。

例如选用定时方式 1 设置系统时钟，由于上述原因就会产生实时误差。这种场合应采用动态补偿办法以减少系统始终误差。所谓动态补偿，即在中断服务程序中对 THx、TLx 重新置计数初值时，应将 THx、TLx 从回 0 溢出又重新从 0 开始继续计数的值读出，并补偿到原计数初值中去进行重新设置。可考虑如下补偿方法:

```
    …
    CLR     EA              ;止中断
    MOV     A, TLx          ;读 TLx 中已计数值
    ADD     A, #LOW         ;LOW 为原低字节计数初值
    MOV     TLx, A          ;设置低字节计数初值
    MOV     A, #HIGH        ;原高字节计数初值送 A
    ADDC    A, THx          ;高字节计数初值补偿
    MOV     THx, A          ;置高字节计数初值
    SETB    EA              ;开中断
    …
```

(2)动态读取运行中的计数值

在动态读取运行中的定时/计数器的计数值时，如果不加注意，就可能出错。这是因为不可能在同一时刻同时读取 THx 和 TLx 中的计数值。比如，先读 TLx 后读 THx，因为定时/计数器处于运行状态，在读 TLx 时尚未产生向 THx 进位，而在读 THx 前已产生进位，这时读得的 THx 就不对了；同样，先读 THx 后读 TLx 也可能出错。

一种可避免读错的方法是：先读 THx，后读 TLx，将两次读得的 THx 进行比较；若两次读得的值相等，则可确定读的值是正确的，否则重复上述过程，重复读得的值一般不会再错。此法的软件编程如下：

```
RDTM:
```

```
MOV     A, THx          ;读取 THx 存 A 中
MOV     R0, TLx         ;读取 TLx 存 R0 中重
CJNE    A, THx, RDTM    ;比较两次 THx 值,若相等,则读得的值正确,程序往下执行,否则重读
MOV     R1, A           ;将 THx 存于 R1 中
…
```

18.5 定时器/计数器 2 及其应用

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/计数器用，也可以当可编程时钟输出和串口的波特率发生器。

下面首先介绍与定时器/计数器 2 相关的寄存器:

18.5.1 定时器/计数器 2 的相关特殊功能寄存器

与定时器/计数器 2 有关的特殊功能寄存器:

符号	描述	地址	位地址及其符号								复位值
			MSB							LSB	
T2H	定时器 2 高 8 位寄存器	D6H									0000 0000B
T2L	定时器 2 低 8 位寄存器	D7H									0000 0000B
AUXR	辅助寄存器	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000 0001B
INT_CLKO AUXR2	外部中断允许和时钟输出寄存器	8FH	-	EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	T0CLKO	x000 0000B
IE2	Interrupt Enableregister	AFH	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000 0000B

1、定时器 2 的控制寄存器：辅助寄存器 AUXR

STC15 系列单片机是 1T 的 8051 单片机，为兼容传统 8051，定时器 0、定时器 1，和定时器 2 复位后是传统 8051 的速度，即 12 分频，这是为了兼容传统 8051。但也可不进行 12 分频，通过设置新增加的特殊功能寄存器 AUXR，将 T0，T1，T2 设置为 1T。普通 111 条机器指令执行速度是固定的，快 3 到 24 倍，无法改变。

AUXR 格式如下:

AUXR：辅助寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	name	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

T2R： 定时器 2 允许控制位
0，不允许定时器 2 运行；
1，允许定时器 2 运行

T2_C/T：控制定时器 2 用作定时器或计数器。
0，用作定时器（对内部系统时钟进行计数）；
1，用作计数器（对引脚 T2/P3.1 的外部脉冲进行计数）

T2x12： 定时器 2 速度控制位
0，定时器 2 是传统 8051 速度，12 分频;
1，定时器 2 的速度是传统 8051 的 12 倍，不分频
如果串口 1 或串口 2 用 T2 作为波特率发生器，则由 T2x12 决定串口 1 或串口 2 是 12T 还是 1T

T0x12： 定时器 0 速度控制位

0，定时器 0 是传统 8051 速度，12 分频；

1，定时器 0 的速度是传统 8051 的 12 倍，不分频

T1x12： 定时器 1 速度控制位

0，定时器 1 是传统 8051 速度，12 分频；

1，定时器 1 的速度是传统 8051 的 12 倍，不分频

如果 UART1/串口 1 用 T1 作为波特率发生器，则由 T1x12 决定 UART1/串口是 12T 还是 1T

UART_M0x6：串口模式 0 的通信速度设置位。

0，串口 1 模式 0 的速度是传统 8051 单片机串口的速度，12 分频；

1，串口 1 模式 0 的速度是传统 8051 单片机串口速度的 6 倍，2 分频

EXTRAM：内部/外部 RAM 存取控制位

0，允许使用逻辑上在片外、物理上在片内的扩展 RAM；

1，禁止使用逻辑上在片外、物理上在片内的扩展 RAM

S1ST2： 串口 1（UART1）选择定时器 2 作波特率发生器的控制位

0，选择定时器 1 作为串口 1（UART1）的波特率发生器；

1，选择定时器 2 作为串口 1（UART1）的波特率发生器，此时定时器 1 得到释放，可以作为独立定时器使用

2、T2 的时钟输出允许控制位 T2CLKO

T2CLKO/P3.0 的时钟输出控制由 INT_CLKO（AUXR2）寄存器中的 T2CLKO 位控制。T2CLKO 的输出时钟频率由定时器 2 控制，不要允许相应的定时器中断，免得 CPU 反复进中断。定时器 2 的工作模式固定为模式 0（16 位自动重装载模式），在此模式下定时器 2 可用作时钟输出。

INT_CLKO（AUXR2）格式如下：

INT_CLKO（AUXR2）：外部中断允许和时钟输出寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
INT_CLKO AUXR2	8FH	name	-	EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	T0CLKO

T2CLKO：是否允许将 P3.0 脚配置为定时器 2（T2）的时钟输出 T2CLKO

1，允许将 P3.0 脚配置为定时器 2 的时钟输出 T2CLKO，输出时钟频率=T2 溢出率/2

- 如果 T2_C/T=0，定时器/计数器 T2 是对内部系统时钟计数，则：
  - T2 工作在 1T 模式（AUXR.2/T2x12=1）时的输出频率=（SYSclk）/（65536 - [RL_TH2, RL_TL2]）/2
  - T2 工作在 12T 模式（AUXR.2/T2x12=0）时的输出频率=（SYSclk）/ 12 /（65536 - [RL_TH2, RL_TL2]）/2
- 如果 T2_C/T=1，定时器/计数器 T2 是对外部脉冲输入（P3.1/T2）计数，则：
  - 输出时钟频率=（T2_Pin_CLK）/（65536 - [RL_TH2, RL_TL2]）/2

0，不允许将 P3.0 脚配置为定时器 2（T2）的时钟输出 T2CLKO

T0CLKO：是否允许将 P3.5/T1 脚配置为定时器 0（T0）的时钟输出 T0CLKO

1，将 P3.5/T1 管脚配置为定时器 0 的时钟输出 T0CLKO，输出时钟频率=T0 溢出率/2

若定时器/计数器 T0 工作在定时器模式 0（16 位自动重装载模式）时，

- 如果 C/T=0，定时器/计数器 T0 是对内部系统时钟计数，则：
  - T0 工作在 1T 模式（AUXR.7/T0x12=1）时的输出频率=（SYSclk）/（65536 - [RL_TH0, RL_TL0]）/2
  - T0 工作在 12T 模式（AUXR.7/T0x12=0）时的输出频率=（SYSclk）/ 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

若定时器/计数器 T0 工作在定时器模式 2（8 位自动重装模式），

- 如果 C/T=0，定时器/计数器 T1 是对内部系统时钟计数，则：
  - T0 工作在 1T 模式（AUXR.7/T0x12=1）时的输出频率=（SYSclk）/（256-TH0）/2
  - T0 工作在 12T 模式（AUXR.7/T0x12=0）时的输出频率=（SYSclk）/ 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

0，不允许 P3.5/T1 管脚被配置为定时器 0 的时钟输出

T1CLKO：是否允许将 P3.4/T0 脚配置为定时器 1（T1）的时钟输出 T1CLKO

1，将 P3.4/T0 管脚配置为定时器 1 的时钟输出 T1CLKO，输出时钟频率=T1 溢出率/2

若定时器/计数器 T1 工作在定时器模式 0（16 位自动重装载模式），

- 如果 C/T=0，定时器/计数器 T1 是对内部系统时钟计数，则：
  - T1 工作在 1T 模式（AUXR.6/T1x12=1）时的输出频率=（SYSclk）/（65536 - [RL_TH1, RL_TL1]）/2
  - T1 工作在 12T 模式（AUXR.6/T1x12=0）时的输出频率=（SYSclk）/ 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

若定时器/计数器 T1 工作在模式 2（8 位自动重装模式），

- 如果 C/T=0，定时器/计数器 T1 是对内部系统时钟计数，则：
  - T1 工作在 1T 模式（AUXR.6/T1x12=1）时的输出频率=（SYSclk）/（256 - TH1）/2
  - T1 工作在 12T 模式（AUXR.6/T1x12=0）时的输出频率=（SYSclk）/ 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

0，不允许 P3.4/T0 管脚被配置为定时器 1 的时钟输出

EX4： 外部中断 4（INT4）中断允许位。外部中断 4（INT4）只能下降沿触发。
EX4=1 允许中断；
EX4=0 禁止中断。

EX3： 外部中断 3（INT3）中断允许位。外部中断 3（INT3）也只能下降沿触发。
EX3=1 允许中断；
EX3=0 禁止中断。

EX2： 外部中断 2（INT2）中断允许位。外部中断 2（INT2）同样只能下降沿触发。
EX2=1 允许中断；
EX2=0 禁止中断。

3、T2 的中断允许控制位 ET2

IE2：中断允许寄存器（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE2	AFH	name	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2

ET4： 定时器 4 的中断允许位。
1，允许定时器 4 产生中断；
0，禁止定时器 4 产生中断。

ET3： 定时器 3 的中断允许位。
1，允许定时器 3 产生中断；
0，禁止定时器 3 产生中断。

ES4： 串行口 4 中断允许位。
1，允许串行口 4 中断；
0，禁止串行口 4 中断

ES3： 串行口 3 中断允许位。
1，允许串行口 3 中断；
0，禁止串行口 3 中断。

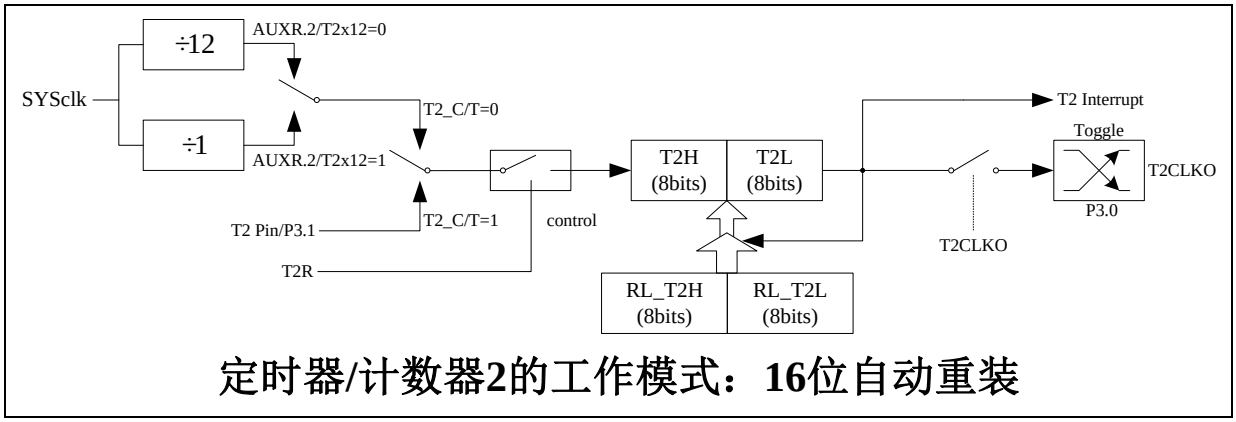
ET2： 定时器 2 的中断允许位。
1，允许定时器 2 产生中断；
0，禁止定时器 2 产生中断。

ESPI： SPI 中断允许位。
1，允许 SPI 中断；
0，禁止 SPI 中断。

ES2： 串行口 2 中断允许位。
1，允许串行口 2 中断；
0，禁止串行口 2 中断。

18.5.2 定时器/计数器 2 作定时器及测试程序（C 和汇编）

定时器/计数器 2 的原理框图如下:



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入 P3.1/T2，即 T2 工作在计数方式。

STC15 系列单片机的定时器 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 2 个隐藏的寄存器 RL_TH2 和 RL_TL2。RL_TH2 与 T2H 共有同一个地址，RL_TL2 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_TL2，对 T2H 写入的内容也会同时写入 RL_TH2。
- 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_TL2 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_TH2。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_TH2 和 RL_TL2 的内容。

这样可以巧妙地实现 16 位重装载定时器。[T2L, T2H]的溢出不仅置位被隐藏的中断请求标志位（定时器 2 的中断请求标志位对用户不可见），使 CPU 转去执行定时器 2 的中断程序，而且会自动将[RL_TL2, RL_TH2]的内容重新装入[T2L, T2H]。

18.5.2.1 定时器 2 的 16 位自动重载模式的测试程序(C 和汇编)

1.C 程序:

```
/*----STC15F2K60S2 系列定时器 2 的 16 位自动重装载模式举例-----------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可----*/
//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"
typedef     unsigned char     BYTE;
typedef     unsigned int      WORD;
//-----------------------------------------------
/* define     constants */
#define     FOSC              18432000L
#define     T38_4KHz          (256-18432000/12/38400/2)      //38.4KHz

/*define     SFR */
sfr         IE2 = 0xAF;                 //(IE2.2)timer2 interrupt control bit
sfr         AUXR = 0x8E;
sfr         T2H = 0xD6;
sfr         T2L = 0xD7;
sbit        TEST_PIN = P0^0;            //test pin
//-----------------------------------------------
/* Timer2 interrupt routine */
void t2_isr() interrupt 12 using 1
{
    TEST_PIN = !TEST_PIN;
}
//-----------------------------------------------

/* main program */
void main()
{
    T2L = T38_4KHz;                     //set timer2 reload value
    T2H = T38_4KH >> 8;
```

```
    AUXR |= 0x10;                   //timer2 start run
    IE2 |= 0x04;                    //enable timer2 interrupt
    EA = 1;                         //open global interrupt switch
    while (1);                      //loop
}
```

2. 汇编程序：

```
/*-------------------------------------------------------------------------------------------------*/
/*----STC15F2K60S2 系列定时器 2 的 16 位自动重装载模式举例-----------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可----*/
/*-------------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz
IE2         DATA    0AFH            ;中断使能寄存器 2
AUXR        DATA    08EH            ;辅助寄存器
T2H         DATA    0D6H            ;定时器 2 高 8 位
T2L         DATA    0D7H            ;定时器 2 低 8 位
F38_4KHz    EQU     0FF10H          ;38.4KHz(1T 模式下, 65536-18432000/2/38400)
;----------------------------------------
    ORG     0000H
    LJMP    START                   ;复位入口
    ORG     0063H                   ;中断入口

    LJMP    T2INT
;----------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    ORL     AUXR, #04H              ;定时器 2 为 1T 模式
    MOV     T2L, #LOW F38_4KHz      ;初始化计时值
    MOV     T2H, #HIGH F38_4KHz
    ORL     AUXR, #10H              ;定时器 2 开始计时
    ORL     IE2, #04H               ;开定时器 2 中断
    SETB    EA
    SJMP    $
;----------------------------------------
;外部中断服务程序
T2INT:
    CPL     P1.0                    ;将测试口取反
;   ANL     IE2, #0FBH              ;若需要手动清除中断标志,可先关闭中断,
                                    ;此时系统会自动清除内部的中断标志
;   ORL     IE2, #04H               ;然后再开中断即可
    RETI
;----------------------------------------
END
```

18.5.2.2 定时器 2 扩展为外部下降沿中断的的测试程序(C 和汇编)

;定时器 2 中断(下降沿中断)的测试程序，定时器/计数器 2 工作在计数模式中的 16 位自动重装载模式

1.C 程序:

```
/*----STC15F2K60S2 系列 T2 扩展为外部下降沿中断举例--------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可------*/
//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"
#include "intrins.h"
//-----------------------------------------------
sfr     IE2 = 0xaf;             //中断使能寄存器 2
sfr     AUXR = 0x8e;            //辅助寄存器
sfr     T2H = 0xD6;             //定时器 2 高 8 位
sfr     T2L = 0xD7;             //定时器 2 低 8 位
sbit    P10 = P1^0;
//-----------------------------------------------
//中断服务程序
void t2int() interrupt 12               //中断入口
{
    P10 = !P10;                         //将测试口取反
//  IE2 &= ~0x04;                       //若需要手动清除中断标志,可先关闭中断,
                                        //此时系统会自动清除内部的中断标志
//  IE2 |= 0x04;                        //然后再开中断即可
}

void main()
{
    AUXR |= 0x04;               //定时器 2 为 1T 模式
    AUXR |= 0x08;               //T2_C/T=1, T2(P3.1)引脚为时钟源
    T2H = T2L = 0xff;           //初始化计时值
    AUXR |= 0x10;               //定时器 2 开始计时
    IE2 |= 0x04;                //开定时器 2 中断
    EA = 1;
    while (1);
}
```

2. 汇编程序：

```
/*-------------------------------------------------------------------------------------------*/
/*----STC15F2K60S2 系列 T2 扩展为外部下降沿中断举例--------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可------*/
/*-------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz
IE2             DATA    0AFH        ;中断使能寄存器 2
```

```
AUXR        DATA    08EH        ;辅助寄存器
T2H         DATA    0D6H        ;定时器 2 高 8 位
T2L         DATA    0D7H        ;定时器 2 低 8 位
;-------------------------------------
    ORG     0000H
    LJMP    START               ;复位入口
    ORG     0063H               ;中断入口

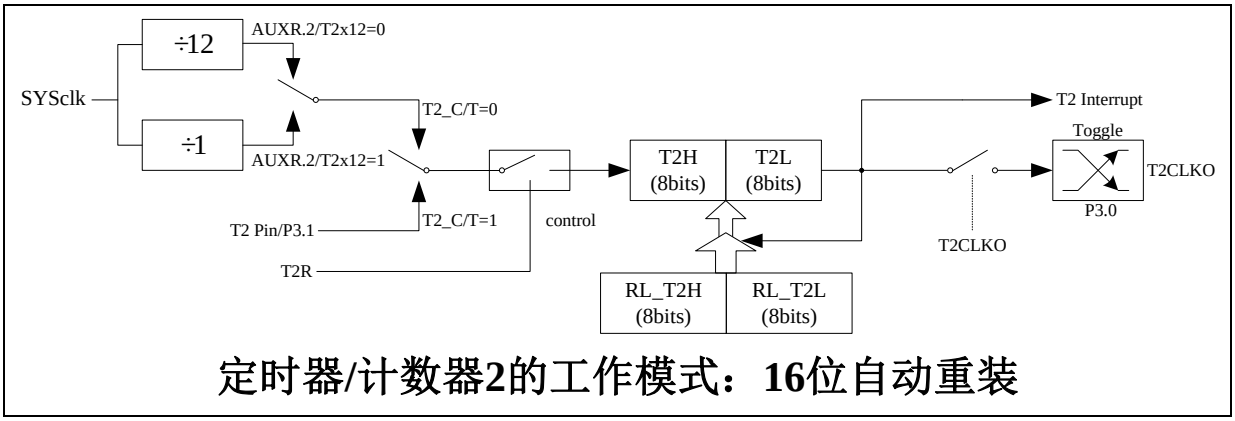
    LJMP    T2INT
;-------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    ORL     AUXR, #04H          ;定时器 2 为 1T 模式
    ORL     AUXR, #08H          ;T2_C/T=1, T2(P3.1)引脚为时钟源
    MOV     A, #0FFH            ;初始化计时值

    MOV     T2L, A
    MOV     T2H, A
    ORL     AUXR, #10H          ;定时器 2 开始计时
    ORL     IE2, #04H           ;开定时器 2 中断
    SETB    EA
    SJMP    $

;-------------------------------------
;外部中断服务程序
T2INT:
    CPL     P1.0                ;将测试口取反
;   ANL     IE2, #0FBH          ;若需要手动清除中断标志,可先关闭中断,
                                ;此时系统会自动清除内部的中断标志
;   ORL     IE2, #04H           ;然后再开中断即可
    RETI
;-------------------------------------
END
```

18.5.3 定时器 2 对系统时钟或外部引脚 T2 的时钟输入进行可编程分频输出

定时器/计数器 2 的原理框图如下:



定时器/计数器 2 除可当定时器/计数器使用外，还可作可编程时钟输出。当定时器/计数器 2 用作可编程时钟输出时，不要允许相应的定时器中断，免得 CPU 反复进中断。

当 T2CLKO/INT_CLKO.2=1 时，P3.0 管脚配置为定时器 2 的时钟输出 T2CLKO。

输出时钟频率=T2 溢出率/2

- 如果 T2_C/T=0，定时器/计数器 T2 对内部系统时钟计数，则：
  - ✓ T2 工作在 1T 模式(AUXR.2/T2x12=1)时的输出时钟频率=(SYSclk) / (65536 - [RL_TH2, RL_TL2]) / 2
  - ✓ T2 工作在 12T 模式(AUXR.2/T2x12=0)时的输出时钟频率=(SYSclk) / 12 / (65536 - [RL_TH2, RL_TL2]) / 2
- 如果 T2_C/T=1，定时器/计数器 T2 是对外部脉冲输入(P3.1/T2)计数，则：
  - ✓ 输出时钟频率=(T2_Pin_CLK) / (65536 - [RL_TH2, RL_TL2]) / 2

上面所有的式子中 RL_TH2 是 T2H 的重装载寄存器，RL_TL2 是 T2L 的重装载寄存器。

下面是定时器 2 对内部系统时钟或外部引脚 T2/P3.1 的时钟输入进行可编程时钟分频输出的程序举例(C 和汇编):

1.C 程序:

```
/*----STC15F2K60S2 系列定时器 2 的可编程时钟分频输出举例------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可---*/
//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"
typedef     unsigned char     BYTE;
typedef     unsigned int      WORD;
#define     FOSC              18432000L
//-----------------------------------------------
sfr         AUXR = 0x8e;                          //辅助特殊功能寄存器
sfr         INT_CLKO = 0x8f;                      //唤醒和时钟输出功能寄存器
sfr         T2H = 0xD6;                           //定时器 2 高 8 位
sfr         T2L = 0xD7;                           //定时器 2 低 8 位
sbit        T2CLKO = P3^0;                        //定时器 2 的时钟输出脚
#define     F38_4KHz          (65536-FOSC/2/38400)    //1T 模式
```

```
//#define       F38_4KHz          (65536-FOSC/2/12/38400)    //12T 模式
//---------------------------------------------
void main()
{
    AUXR |= 0x04;                   //定时器 2 为 1T 模式
//  AUXR &= ~0x04;                  //定时器 2 为 12T 模式
    AUXR &= ~0x08;                  //T2_C/T=0, 对内部时钟进行时钟输出
//  AUXR |= 0x08;                   //T2_C/T=1, 对 T2(P3.1)引脚的外部时钟进行时钟输出
    T2L = F38_4KHz;                 //初始化计时值
    T2H = F38_4KHz >> 8;
    AUXR |= 0x10;                   //定时器 2 开始计时
    INT_CLKO = 0x04;                //使能定时器 2 的时钟输出功能
    while (1);                      //程序终止
}
```

2. 汇编程序：

```
/*-------------------------------------------------------------------------------------------*/
/*----STC15F2K60S2 系列定时器 2 的可编程时钟分频输出举例-----------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可---*/
/*-------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz
AUXR            DATA 08EH           ;辅助特殊功能寄存器
INT_CLKO        DATA 08FH           ;唤醒和时钟输出功能寄存器
T2H             DATA 0D6H           ;定时器为高 8 位
T2L             DATA 0D7H           ;定时器为低 8 位

T2CLKO          BIT     P3.0        ;定时器 2 的时钟输出脚

F38_4KHz        EQU     0FF10H      ;38.4KHz(1T 模式下, 65536-18432000/2/38400)
;F38_4KHz       EQU     0FFECH      ;38.4KHz(12T 模式下, (65536-18432000/2/12/38400)

;---------------------------------------------
    ORG     0000H
    LJMP    START                   ;复位入口
;-----------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    ORL     AUXR, #04H              ;定时器 2 为 1T 模式

;   ANL     AUXR, #0FBH             ;定时器 2 为 12T 模式
    ANL     AUXR, #0F7H             ;T2_C/T=0, 对内部时钟进行时钟输出
;   ORL     AUXR, #08H              ;T2_C/T=1, 对 T2(P3.1)引脚的外部时钟进行时钟输出
    MOV     T2L, #LOW F38_4KHz      ;初始化计时值
    MOV     T2H, #HIGH F38_4KHz
```

```
    ORL     AUXR, #10H          ;定时器 2 开始计时
    MOV     INT_CLKO, #04H      ;使能定时器 2 的时钟输出功能
    SJMP    $                   ;程序终止
;----------------------------------------------
END
```

18.5.4 定时器/计数器 2 作串行口波特率发生器及测试程序(C 和汇编)

定时器/计数器 2 除可当定时器/计数器和可编程时钟输出使用外，还可作串行口波特率发生器。串行口 1 优先选择定时器 2 作为其波特率发生器，串行口 2 只能选择定时器 2 作为其波特率发生器，串行口 3/串口 4 默认选择定时器 2 作为其波特率发生器。

串行口 1 如果工作在模式 1（8 位 UART，波特率可变）和模式 3（9 位 UART，波特率可变）时，其可变的波特率可以由定时器 T2 产生。此时：

串行口 1 的波特率=（定时器 T2 的溢出率）/4，注意：此时波特率也与 SMOD 无关。

- 当 T2 工作在 1T 模式（AUXR.2/T2x12=1）时，定时器 T2 的溢出率=SYSclk/（65536-[RL_TH2, RL_TL2]）；

即此时，串行口 1 的波特率=SYSclk /（65536-[RL_TH2, RL_TL2]）/4

- 当 T2 工作在 12T 模式（AUXR.2/T2x12=0）时，定时器 2 的溢出率=SYSclk /12/（65536-[RL_TH2, RL_TL2]）；

即此时，串行口 1 的波特率=SYSclk / 12 /（65536-[RL_TH2, RL_TL2]）/4

串行口 2 的工作模式只有两种：模式 0（8 位 UART，波特率可变）和模式 1（9 位 UART，波特率可变）。串行口 2 只能选择定时器 T2 作其波特率发生器。串行口 2 的波特率按如下公式计算：

串行口 2 的波特率=（定时器 T2 的溢出率）/4

- 当 T2 工作在 1T 模式（AUXR.2/T2x12=1）时，定时器 2 的溢出率=SYSclk/（65536-[RL_TH2, RL_TL2]）；

即此时，串行口 2 的波特率=SYSclk /（65536-[RL_TH2, RL_TL2]）/4

- 当 T2 工作在 12T 模式（AUXR.2/T2x12=0）时，定时器 2 的溢出率=SYSclk /12/（65536-[RL_TH2, RL_TL2]）；

即此时，串行口 2 的波特率=SYSclk / 12 /（65536-[RL_TH2, RL_TL2]）/4

串行口 3 的工作模式只有两种：模式 0（8 位 UART，波特率可变）和模式 1（9 位 UART，波特率可变）。串行口 3 可以选择定时器 T3 作为其波特率发生器，也可以选择定时器 T2 作其波特率发生器。当选择定时器 2 作为其波特率发生器时，串行口 3 的波特率按如下公式计算：

串行口 3 的波特率=（定时器 T2 的溢出率）/4

- 当 T2 工作在 1T 模式（AUXR.2/T2x12=1）时，定时器 2 的溢出率=SYSclk/（65536-[RL_TH2, RL_TL2]）；

即此时，串行口 3 的波特率=SYSclk /（65536-[RL_TH2, RL_TL2]）/4

- 当 T2 工作在 12T 模式（AUXR.2/T2x12=0）时，定时器 2 的溢出率=SYSclk /12/（65536-[RL_TH2, RL_TL2]）；

即此时，串行口 3 的波特率=SYSclk / 12 /（65536-[RL_TH2, RL_TL2]）/4

串行口 4 的工作模式只有两种：模式 0（8 位 UART，波特率可变）和模式 1（9 位 UART，波特率可变）。串行口 4 可以选择定时器 T4 作为其波特率发生器，也可以选择定时器 T2 作其波特率发生器。当选择定时器 2 作为其波特率发生器时，串行口 4 的波特率按如下公式计算：

串行口 4 的波特率=（定时器 T2 的溢出率）/4

- 当 T2 工作在 1T 模式（AUXR.2/T2x12=1）时，定时器 2 的溢出率=SYSclk/（65536-[RL_TH2, RL_TL2]）；

即此时，串行口 4 的波特率=SYSclk /（65536-[RL_TH2, RL_TL2]）/4

- 当 T2 工作在 12T 模式（AUXR.2/T2x12=0）时，定时器 2 的溢出率=SYSclk /12/（65536-[RL_TH2, RL_TL2]）；

即此时，串行口 4 的波特率=SYSclk / 12 /（65536-RL_TH2.RL_TL2]）/4

上面所有的式子中 RL_TH2 是 T2H 的重装载寄存器，RL_TL2 是 T2L 的重装载寄存器。

18.5.4.1 定时器/计数器 2 作串行口 1 波特率发生器的测试程序(C 和汇编)

1.C 程序:

```
/*----STC15F2K60S2 系列定时器 2 用作串口 1 的波特率发生器举例-------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可----*/
//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"
#include "intrins.h"
typedef     unsigned char     BYTE;
typedef     unsigned int      WORD;

#define     FOSC              18432000L          //系统频率
#define     BAUD              115200             //串口波特率
#define     NONE_PARITY       0                  //无校验
#define     ODD_PARITY        1                  //奇校验
#define     EVEN_PARITY       2                  //偶校验
#define     MARK_PARITY       3                  //标记校验
#define     SPACE_PARITY      4                  //空白校验
#define     PARITYBIT         EVEN_PARITY        //定义校验位

sfr         AUXR = 0x8e;                         //辅助寄存器
sfr         T2H = 0xd6;                          //定时器 2 高 8 位
sfr         T2L = 0xd7;                          //定时器 2 低 8 位
sbit        P22 = P2^2;
bit         busy;

void SendData(BYTE dat);
void SendString(char *s);
void main()
{
#if (PARITYBIT == NONE_PARITY)
    SCON = 0x50;                                 //8 位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
    SCON = 0xda;                                 //9 位可变波特率,校验位初始为 1
#elif (PARITYBIT == SPACE_PARITY)
    SCON = 0xd2;                                 //9 位可变波特率,校验位初始为 0
#endif

    T2L = (65536 - (FOSC/4/BAUD));               //设置波特率重装值
    T2H = (65536 - (FOSC/4/BAUD))>>8;
    AUXR = 0x14;                                 //T2 为 1T 模式, 并启动定时器 2
    AUXR |= 0x01;                                //选择定时器 2 为串口 1 的波特率发生器
    ES = 1;                                      //使能串口 1 中断
    EA = 1;
```

```
    SendString("STC15F2K60S2\r\nUart Test !\r\n");
    while(1);
}

/*----------------------------UART 中断服务程序----------------------------*/
void Uart() interrupt 4 using 1
{
    if (RI)
    {
        RI = 0;                         //清除 RI 位
        P0 = SBUF;                      //P0 显示串口数据
        P22 = RB8;                      //P2.2 显示校验位
    }
    if (TI)
    {
        TI = 0;                         //清除 TI 位
        busy = 0;                       //清忙标志
    }
}

/*----------------------------发送串口数据----------------------------*/
void SendData(BYTE dat)
{
    while (busy);                       //等待前面的数据发送完成
    ACC = dat;                          //获取校验位 P (PSW.0)
    if (P)                              //根据 P 来设置校验位
    {
#if (PARITYBIT == ODD_PARITY)
    TB8 = 0;                            //设置校验位为 0
#elif (PARITYBIT == EVEN_PARITY)
    TB8 = 1;                            //设置校验位为 1
#endif
    }
    else
    {
#if (PARITYBIT == ODD_PARITY)
    TB8 = 1;                            //设置校验位为 1
#elif (PARITYBIT == EVEN_PARITY)
    TB8 = 0;                            //设置校验位为 0
#endif
    }
    busy = 1;
    SBUF = ACC;                         //写数据到 UART 数据寄存器
}

/*----------------------------发送字符串----------------------------*/
```

```
void SendString(char *s)
{
    while (*s)                                  //检测字符串结束标志
    {
        SendData(*s++);                         //发送当前字符
    }
}
```

2. 汇编程序：

```
/*-------------------------------------------------------------------------------------------------------- */
/*----STC15F2K60S2 系列定时器 2 用作串口 1 的波特率发生器举例-------------*/
/*----在 Keil C 开发环境中选择  Intel 8052 编译，头文件包含<reg51.h>即可----*/
/*-------------------------------------------------------------------------------------------------------- */
;假定测试芯片的工作频率为 18.432MHz

#define     NONE_PARITY     0               ;无校验
#define     ODD_PARITY      1               ;奇校验
#define     EVEN_PARITY     2               ;偶校验
#define     MARK_PARITY     3               ;标记校验
#define     SPACE_PARITY    4               ;空白校验
#define     PARITYBIT       EVEN_PARITY     ;定义校验位
;---------------------------------------
AUXR        EQU         08EH                ;辅助寄存器
T2H         DATA        0D6H                ;定时器 2 高 8 位
T2L         DATA        0D7H                ;定时器 2 低 8 位
;---------------------------------------
BUSY        BIT         20H.0               ;忙标志位
;---------------------------------------
    ORG     0000H
    LJMP    START
    ORG     0023H
    LJMP    UART_ISR
;---------------------------------------
    ORG     0100H

START:
    CLR     BUSY
    CLR     EA
    MOV     SP, #3FH

#if (PARITYBIT == NONE_PARITY)
    MOV     SCON, #50H                  ;8 位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
    MOV     SCON, #0DAH                 ;9 位可变波特率,校验位初始为 1
#elif (PARITYBIT == SPACE_PARITY)
```

```
        MOV     SCON, #0D2H             ;9 位可变波特率,校验位初始为 0
    #endif
;-------------------------------
    MOV     T2L, #0D8H                  ;设置波特率重装值(65536-18432000/4/115200)
    MOV     T2H, #0FFH
    MOV     AUXR, #14H                  ;T2 为 1T 模式, 并启动定时器 2
    ORL     AUXR, #01H                  ;选择定时器 2 为串口 1 的波特率发生器
    SETB    ES                          ;使能串口中断
    SETB    EA

    MOV     DPTR, #TESTSTR              ;发送测试字符串
    LCALL   SENDSTRING

    SJMP    $
;----------------------------------------
TESTSTR:
    DB      "STC15F2K60S2 Uart1 Test !",0DH,0AH,0
;/*----------------------------UART 中断服务程序----------------------------*/
UART_ISR:
    PUSH    ACC
    PUSH    PSW
    JNB     RI, CHECKTI                 ;检测 RI 位
    CLR     RI                          ;清除 RI 位
    MOV     P0, SBUF                    ;P0 显示串口数据
    MOV     C, RB8
    MOV     P2.2, C                     ;P2.2 显示校验位

CHECKTI:
    JNB     TI, ISR_EXIT                ;检测 TI 位
    CLR     TI                          ;清除 TI 位
    CLR     BUSY                        ;清忙标志

ISR_EXIT:
    POP     PSW
    POP     ACC
    RETI
;/*---------------------------发送串口数据---------------------------*/
SENDDATA:
    JB      BUSY, $                     ;等待前面的数据发送完成
    MOV     ACC, A                      ;获取校验位 P (PSW.0)
    JNB     P, EVEN1INACC               ;根据 P 来设置校验位

ODD1INACC:
#if (PARITYBIT == ODD_PARITY)
    CLR     TB8                         ;设置校验位为 0
#elif (PARITYBIT == EVEN_PARITY)
```

```
    SETB    TB8                     ;设置校验位为 1
#endif
    SJMP    PARITYBITOK

EVEN1INACC:
#if (PARITYBIT == ODD_PARITY)
    SETB    TB8                     ;设置校验位为 1
#elif (PARITYBIT == EVEN_PARITY)
    CLR     TB8                     ;设置校验位为 0
#endif

PARITYBITOK:                        ;校验位设置完成
    SETB    BUSY
    MOV     SBUF, A                 ;写数据到 UART 数据寄存器
    RET
;/*----------------------------发送字符串----------------------------*/
SENDSTRING:
    CLR     A
    MOVC    A, @A+DPTR              ;读取字符
    JZ      STRINGEND               ;检测字符串结束标志
    INC     DPTR                    ;字符串地址+1
    LCALL   SENDDATA                ;发送当前字符
    SJMP    SENDSTRING

STRINGEND:
    RET
;--------------------------------------
END
```

18.5.4.2 定时器/计数器 2 作串行口 2 波特率发生器的测试程序(C 和汇编)

1.C 程序:

```
/*----STC15F2K60S2 系列定时器 2 用作串口 2 的波特率发生器举例------------*/
/*----在 Keil C 开发环境中选择  Intel 8052 编译，头文件包含<reg51.h>即可---*/
//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"
#include "intrins.h"
typedef     unsigned char     BYTE;
typedef     unsigned int      WORD;

#define     FOSC              18432000L          //系统频率
#define     BAUD              115200             //串口波特率
#define     TM                (65536 - (FOSC/4/BAUD))

#define     NONE_PARITY       0                  //无校验
#define     ODD_PARITY        1                  //奇校验
#define     EVEN_PARITY       2                  //偶校验
#define     MARK_PARITY       3                  //标记校验
#define     SPACE_PARITY      4                  //空白校验
#define     PARITYBIT         EVEN_PARITY        //定义校验位

sfr         AUXR = 0x8e;                         //辅助寄存器
sfr         S2CON = 0x9a;                        //UART2  控制寄存器
sfr         S2BUF = 0x9b;                        //UART2  数据寄存器
sfr         T2H = 0xd6;                          //定时器 2 高 8 位
sfr         T2L = 0xd7;                          //定时器 2 低 8 位
sfr         IE2 = 0xaf;                          //中断控制寄存器 2

#define     S2RI              0x01               //S2CON.0
#define     S2TI              0x02               //S2CON.1
#define     S2RB8             0x04               //S2CON.2
#define     S2TB8             0x08               //S2CON.3

bit         busy;

void SendData(BYTE dat);
void SendString(char *s);
void main()
{
#if (PARITYBIT == NONE_PARITY)
    S2CON = 0x50;                                //8 位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
    S2CON = 0xda;                                //9 位可变波特率,校验位初始为 1
```

```
#elif (PARITYBIT == SPACE_PARITY)
    S2CON = 0xd2;                       //9 位可变波特率,校验位初始为 0
#endif

    T2L = TM;                           //设置波特率重装值
    T2H = TM>>8;

    AUXR = 0x14;                        //T2 为 1T 模式, 并启动定时器 2
    IE2 = 0x01;                         //使能串口 2 中断
    EA = 1;
    SendString("STC15F2K60S2\r\nUart2 Test !\r\n");
    while(1);
}

/*----------------------------UART2  中断服务程序----------------------------*/
void Uart2() interrupt 8 using 1
{
    if (S2CON & S2RI)
    {
        S2CON &= ~S2RI;                 //清除 S2RI 位
        P0 = S2BUF;                     //P0 显示串口数据
        P2 = (S2CON & S2RB8);           //P2.2 显示校验位
    }
    if (S2CON & S2TI)
    {
        S2CON &= ~S2TI;                 //清除 S2TI 位
        busy = 0;                       //清忙标志
    }
}

/*----------------------------发送串口数据----------------------------*/
void SendData(BYTE dat)
{
    while (busy);                       //等待前面的数据发送完成
    ACC = dat;                          //获取校验位 P (PSW.0)
    if (P)                              //根据 P 来设置校验位
    {
#if (PARITYBIT == ODD_PARITY)
    S2CON &= ~S2TB8;                    //设置校验位为 0
#elif (PARITYBIT == EVEN_PARITY)
    S2CON |= S2TB8;                     //设置校验位为 1
#endif
    }
    else
    {
#if (PARITYBIT == ODD_PARITY)
```

```
    S2CON |= S2TB8;                 //设置校验位为 1
#elif (PARITYBIT == EVEN_PARITY)
    S2CON &= ~S2TB8;                //设置校验位为 0
#endif
    }
    busy = 1;
    S2BUF = ACC;                    //写数据到 UART2 数据寄存器
}

/*----------------------------发送字符串----------------------------*/
void SendString(char *s)
{
    while (*s)                      //检测字符串结束标志
    {
        SendData(*s++);             //发送当前字符
    }
}
```

2. 汇编程序：

```
/*-------------------------------------------------------------------------------------------------*/
/*----STC15F2K60S2 系列定时器 2 用作串口 2 的波特率发生器举例------------*/
/*----在 Keil C 开发环境中选择  Intel 8052 编译，头文件包含<reg51.h>即可---*/
/*-------------------------------------------------------------------------------------------------*/
;假定测试芯片的工作频率为 18.432MHz

#define     NONE_PARITY     0           ;无校验
#define     ODD_PARITY      1           ;奇校验
#define     EVEN_PARITY     2           ;偶校验
#define     MARK_PARITY     3           ;标记校验
#define     SPACE_PARITY    4           ;空白校验
#define     PARITYBIT       EVEN_PARITY ;定义校验位
;---------------------------------------
AUXR        EQU             08EH        ;辅助寄存器

S2CON       EQU             09AH        ;UART2  控制寄存器
S2BUF       EQU             09BH        ;UART2  数据寄存器

T2H         DATA            0D6H        ;定时器 2 高 8 位
T2L         DATA            0D7H        ;定时器 2 低 8 位

IE2         EQU             0AFH        ;中断控制寄存器 2

S2RI        EQU             01H         ;S2CON.0
S2TI        EQU             02H         ;S2CON.1
S2RB8       EQU             04H         ;S2CON.2
```

```
S2TB8       EQU         08H             ;S2CON.3
;---------- ------------------------------
BUSY        BIT         20H.0           ;忙标志位
;----------------------------------------
    ORG     0000H
    LJMP    START
    ORG     0043H
    LJMP    UART2_ISR
;----------------------------------------
    ORG     0100H
START:
    CLR     BUSY
    CLR     EA
    MOV     SP, #3FH
#if (PARITYBIT == NONE_PARITY)
    MOV     S2CON, #50H                 ;8 位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
    MOV     S2CON, #0DAH                ;9 位可变波特率,校验位初始为 1
#elif (PARITYBIT == SPACE_PARITY)
    MOV     S2CON, #0D2H                ;9 位可变波特率,校验位初始为 0
#endif
;------------------------------
    MOV     T2L, #0D8H                  ;设置波特率重装值(65536-18432000/4/115200)
    MOV     T2H, #0FFH
    MOV     AUXR, #14H                  ;T2 为 1T 模式, 并启动定时器 2
    ORL     IE2, #01H                   ;使能串口 2 中断
    SETB    EA
    MOV     DPTR, #TESTSTR              ;发送测试字符串
    LCALL   SENDSTRING
    SJMP    $
;----------------------------------------
TESTSTR:
    DB      "STC15F2K60S2 Uart2 Test !",0DH,0AH,0
;/*--------------------------UART2  中断服务程序--------------------------*/
UART2_ISR:
    PUSH    ACC
    PUSH    PSW
    MOV     A, S2CON                    ;读取 UART2 控制寄存器
    JNB     ACC.0, CHECKTI              ;检测 S2RI 位
    ANL     S2CON, #NOT S2RI            ;清除 S2RI 位
    MOV     P0, S2BUF                   ;P0 显示串口数据
    ANL     A, #S2RB8 ;
    MOV     P2, A                       ;P2.2 显示校验位

CHECKTI:
    MOV     A, S2CON                    ;读取 UART2 控制寄存器
```

```
    JNB     ACC.1, ISR_EXIT             ;检测 S2TI 位
    ANL     S2CON, #NOT S2TI            ;清除 S2TI 位
    CLR     BUSY                        ;清忙标志
ISR_EXIT:
    POP     PSW
    POP     ACC
    RETI
;/*----------------------------发送串口数据----------------------------*/
SENDDATA:
    JB      BUSY, $                     ;等待前面的数据发送完成
    MOV     ACC, A                      ;获取校验位 P (PSW.0)
    JNB     P, EVEN1INACC               ;根据 P 来设置校验位

ODD1INACC:
#if (PARITYBIT == ODD_PARITY)
    ANL     S2CON, #NOT S2TB8           ;设置校验位为 0
#elif (PARITYBIT == EVEN_PARITY)
    ORL     S2CON, #S2TB8               ;设置校验位为 1
#endif
    SJMP    PARITYBITOK

EVEN1INACC:
#if (PARITYBIT == ODD_PARITY)
    ORL     S2CON, #S2TB8               ;设置校验位为 1
#elif (PARITYBIT == EVEN_PARITY)
    ANL     S2CON, #NOT S2TB8           ;设置校验位为 0
#endif

PARITYBITOK:                            ;校验位设置完成
    SETB    BUSY
    MOV     S2BUF, A                    ;写数据到 UART2 数据寄存器
    RET
;/*----------------------------发送字符串----------------------------*/
SENDSTRING:
    CLR     A
    MOVC    A, @A+DPTR                  ;读取字符
    JZ      STRINGEND                   ;检测字符串结束标志
    INC     DPTR                        ;字符串地址+1
    LCALL   SENDDATA                    ;发送当前字符
    SJMP    SENDSTRING
STRINGEND:
    RET
;---------------------------------------
END
```

18.6 定时器/计数器 3 及定时器/计数器 4

STC15W4K60S4 还新增了两个 16 位定时/计数器：T3 和 T4。T3、T4 和 T2 一样，它们的工作模式固定为 16 位自动重装载模式。T3 和 T4 既可以当定时器/计数器用，也可以当可编程时钟输出和串口的波特率发生器。

下面首先介绍与定时器/计数器 T3 和 T4 相关的寄存器:

18.6.1 定时器/计数器 3 和定时器/计数器 4 的相关特殊功能寄存器

与定时器/计数器 3 和定时器/计数器 4 有关的特殊功能寄存器:

符号	描述	地址	位地址及其符号								复位值
			MSB							LSB	
T4T3M	T4 和 T3 的控制寄存器	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO	0000 0000B
T4H	定时器 4 高 8 位寄存器	D2H									0000 0000B
T4L	定时器 4 低 8 位寄存器	D3H									0000 0000B
T3H	定时器 3 高 8 位寄存器	D4H									0000 0000B
T3L	定时器 3 低 8 位寄存器	D5H									0000 0000B
IE2	Interrupt Enable register	AFH	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000 0000B

1、定时器 T4 和 T3 的控制寄存器：T4T3M（地址： 0xD1）

T4T3M（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	name	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO

B7_T4R：定时器 4 运行控制位。

0：不允许定时器 4 运行；

1：允许定时器 4 运行。

B6-T4_C/T：控制定时器 4 用作定时器或计数器。

0：用作定时器（对内部系统时钟进行计数）；

1：用作计数器（对引脚 T4/P0.7 的外部脉冲进行计数）

B5-T4x12：定时器 4 速度控制位。

0：定时器 4 速度是 8051 单片机定时器的速度，即 12 分频；

1：定时器 4 速度是 8051 单片机定时器速度的 12 倍，即不分频。

B4-T4CLKO：是否允许将 P0.6 脚配置为定时器 4（T4）的时钟输出 T4CLKO

1：允许将 P0.6 脚配置为定时器 4 的时钟输出 T4CLKO，输出时钟频率=T4 溢出率/2

- 如果 T4_C/T=0，定时器/计数器 T4 是对内部系统时钟计数，则：
  - ✓ T4 工作在 1T 模式（T4T3M.5/T4x12=1）时的输出频率=（SYSclk）/（65536-[RL_TH4, RL_TL4]）/2
  - ✓ T4 工作在 12T 模式（T4T3M.5/T4x12=0）时的输出频率=（SYSclk）/12/（65536-[RL_TH4, RL_TL4]）/2
- 如果 T4_C/T=1，定时器/计数器 T4 是对外部脉冲输入（P0.7/T4）计数，则：
  - ✓ 输出时钟频率=（T4_Pin_CLK）/（65536-[RL_TH4, RL_TL4]）/2

0：不允许将 P0.6 脚配置为定时器 4（T4）的时钟输出 T4CLKO

B3-T3R：定时器 3 运行控制位。

0：不允许定时器 3 运行;

1：允许定时器 3 运行。

B2-T3_C/T：控制定时器 3 用作定时器或计数器。

0：用作定时器（对内部系统时钟进行计数）；

1：用作计数器（对引脚 T3/P0.5 的外部脉冲进行计数）

B1-T3x12：定时器 3 速度控制位。

0：定时器 3 速度是 8051 单片机定时器的速度，即 12 分频；

1：定时器 3 速度是 8051 单片机定时器速度的 12 倍，即不分频。

B0-T3CLKO：是否允许将 P0.4 脚配置为定时器 3（T3）的时钟输出 T3CLKO

1：允许将 P0.4 脚配置为定时器 3 的时钟输出 T3CLKO，输出时钟频率=T3 溢出率/2

- 如果 T3_C/T=0，定时器/计数器 T3 是对内部系统时钟计数，则：
  - ✓ T3 工作在 1T 模式（T4T3M.1/T3x12=1）时的输出频率=（SYSclk）/（65536-[RL_TH3, RL_TL3]）/2
  - ✓ T3 工作在 12T 模式（T4T3M.1/T3x12=0）时的输出频率=（SYSclk）/12/（65536-[RL_TH3, RL_TL3]）/2
- 如果 T3_C/T=1，定时器/计数器 T3 是对外部脉冲输入（P0.5/T3）计数，则：
  - ✓ 输出时钟频率=（T3_Pin_CLK）/（65536-[RL_TH3, RL_TL3]）/2

0：不允许将 P0.4 脚配置为定时器 3（T3）的时钟输出 T3CLKO

2、定时器 T3 和 T4 的中断控制寄存器：IE2

IE2：中断允许寄存器（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE2	AFH	name	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2

ET4： 定时器 4 的中断允许位。

1：允许定时器 4 产生中断;

0：禁止定时器 4 产生中断。

ET3： 定时器 3 的中断允许位。

1：允许定时器 3 产生中断;

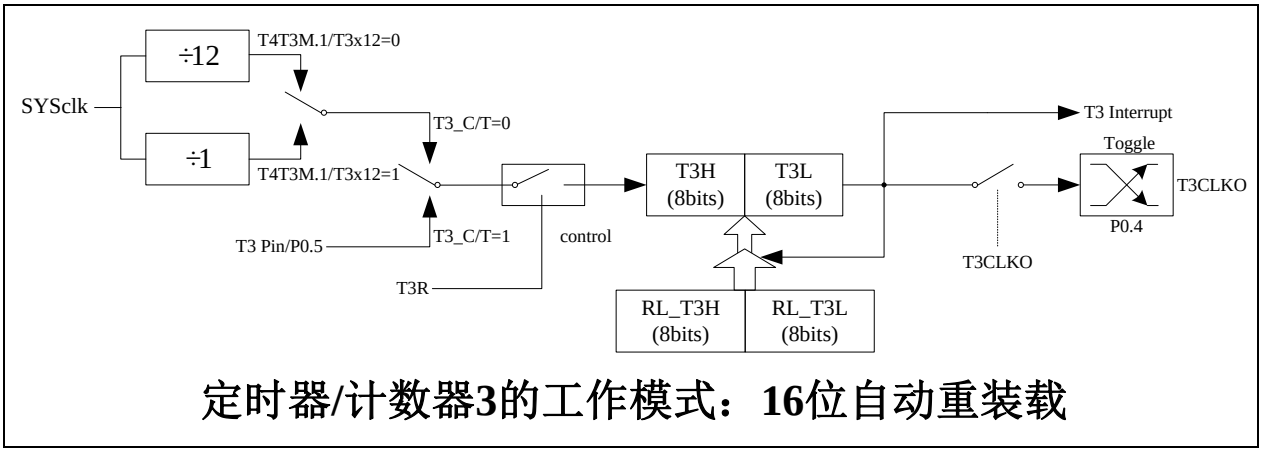
0：禁止定时器 3 产生中断。

18.6.2 定时器/计数器 3 的应用（STC 创新设计，请不要抄袭）

定时器/计数器 3 既可以当定时器/计数器用，也可以当可编程时钟输出和串口的波特率发生器。

18.6.2.1 定时器/计数器 3 作定时器

定时器/计数器 3 的原理框图如下：



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入 P0.5/T3，即 T3 工作在计数方式。

STC15W4K32S4 系列单片机的定时器 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。

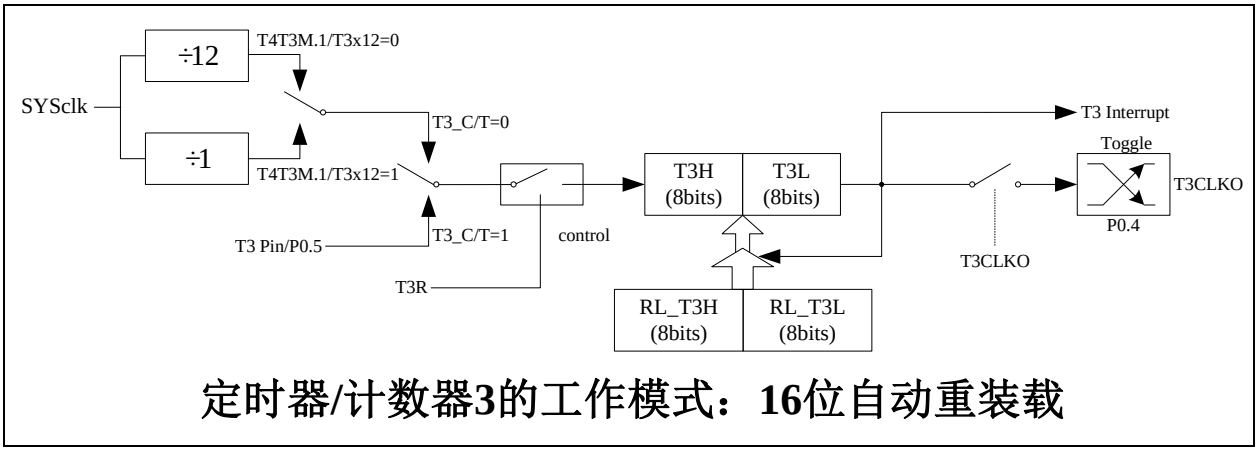
定时器 3 有 2 个隐藏的寄存器 RL_TH3 和 RL_TL3。RL_TH3 与 T3H 共有同一个地址，RL_TL3 与 T3L 共有同一个地址。

- 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_TL3，对 T3H 写入的内容也会同时写入 RL_TH3。
- 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_TL3 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_TH3。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_TH3 和 RL_TL3 的内容。

这样可以巧妙地实现 16 位重装载定时器。[T3L, T3H]的溢出不仅置位被隐藏的中断请求标志位（定时器 3 的中断请求标志位对用户不可见），使 CPU 转去执行定时器 3 中断的程序，而且会自动将[RL_TL3, RL_TH3]的内容重新装入[T3L, T3H]。

18.6.2.2 定时器/计数器 3 对系统时钟或外部引脚 T3 的时钟输入进行可编程时钟分频输出

定时器/计数器 3 的原理框图如下：



定时器/计数器 3 除可当定时器/计数器使用外，还可作可编程时钟输出。当定时器/计数器 3 用作可编程时钟输出时，不要允许相应的定时器中断,免得 CPU 反复进中断。

当 T3CLKO/T4T3M.0=1 时，P0.4 管脚配置为定时器 3 的时钟输出 T3CLKO。

输出时钟频率=T3 溢出率/2

- 如果 T3_C/T=0，定时器/计数器 T3 对内部系统时钟计数，则：
  - ✓ T3 工作在 1T 模式（T4T3M.1/T3x12=1）时的输出时钟频率=（SYSclk）/（65536-[RL_TH3, RL_TL3]）/2
  - ✓ T3 工作在 12T 模式（T4T3M.1/T3x12=0）时的输出时钟频率=（SYSclk）/12/（65536-[RL_TH3, RL_TL3]）/2
- 如果 T3_C/T=1，定时器/计数器 T3 是对外部脉冲输入（P0.5/T3）计数，则：
  - ✓ 输出时钟频率=（T3_Pin_CLK）/（65536-[RL_TH3, RL_TL3]）/2

上面所有的式子中 RL_TH3 是 T3H 的重装载寄存器，RL_TL3 是 T3L 的重装载寄存器。

18.6.2.3 定时器/计数器 3 作串行口 3 的波特率发生器

定时器/计数器 3 除可当定时器/计数器和可编程时钟输出使用外，还可作串行口 3 波特率发生器。串行口 3 默认选择定时器 2 作为其波特率发生器，但通过设置 S3ST3/S3CON.6，串行口 3 也可以选择定时器 3 作为其波特率发生器。

S3CON：串行口 3 控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
S3CON	ACH	name	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI

S3ST3： 串口 3（UART3）选择定时器 3 作波特率发生器的控制位
0，选择定时器 2 作为串口 3（UART3）的波特率发生器；
1，选择定时器 3 作为串口 3（UART3）的波特率发生器

串行口 3 的工作模式只有两种：模式 0（8 位 UART，波特率可变）和模式 1（9 位 UART，波特率可变）。当串行口 3 被设置为选择定时器 3 作为其波特率发生器时，串行口 3 的波特率按如下公式计算：

串行口 3 的波特率=（定时器 T3 的溢出率）/4

- 当 T3 工作在 1T 模式（T4T3M.1/T3x12=1）时，定时器 3 的溢出率=SYSclk/（65536-[RL_TH3, RL_TL3]）；

即此时，串行口 3 的波特率=SYSclk/（65536-[RL_TH3, RL_TL3]）/4

- 当 T3 工作在 12T 模式（T4T3M.1/T3x12=0）时，定时器 3 的溢出率=SYSclk/12/（65536-[RL_TH3, RL_TL3]）；

即此时，串行口 3 的波特率=SYSclk/12/（65536-[RL_TH3, RL_TL3]）/4

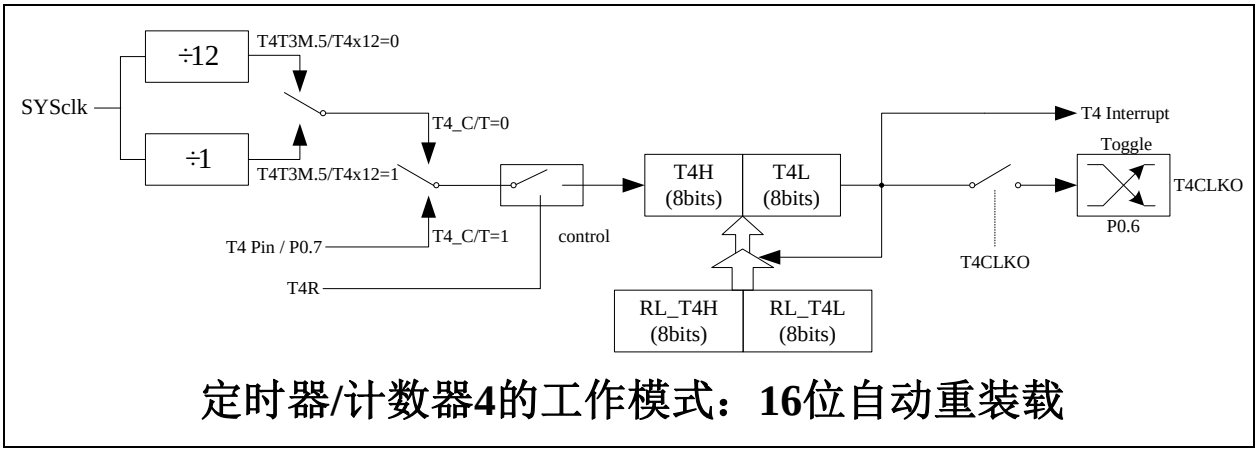
上面所有的式子中 RL_TH3 是 T3H 的重装载寄存器，RL_TL3 是 T3L 的重装载寄存器。

18.6.3 定时器/计数器 4 的应用（STC 创新设计，请不要抄袭）

定时器/计数器 4 既可以当定时器/计数器用，也可以当可编程时钟输出和串口的波特率发生器。

18.6.3.1 定时器/计数器 4 作定时器

定时器/计数器 4 的原理框图如下：



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入 P0.7/T4，即 T4 工作在计数方式。

STC15W4K32S4 系列单片机的定时器 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 2 个隐藏的寄存器 RL_TH4 和 RL_TL4。RL_TH4 与 T4H 共有同一个地址，RL_TL4 与 T4L 共有同一个地址。

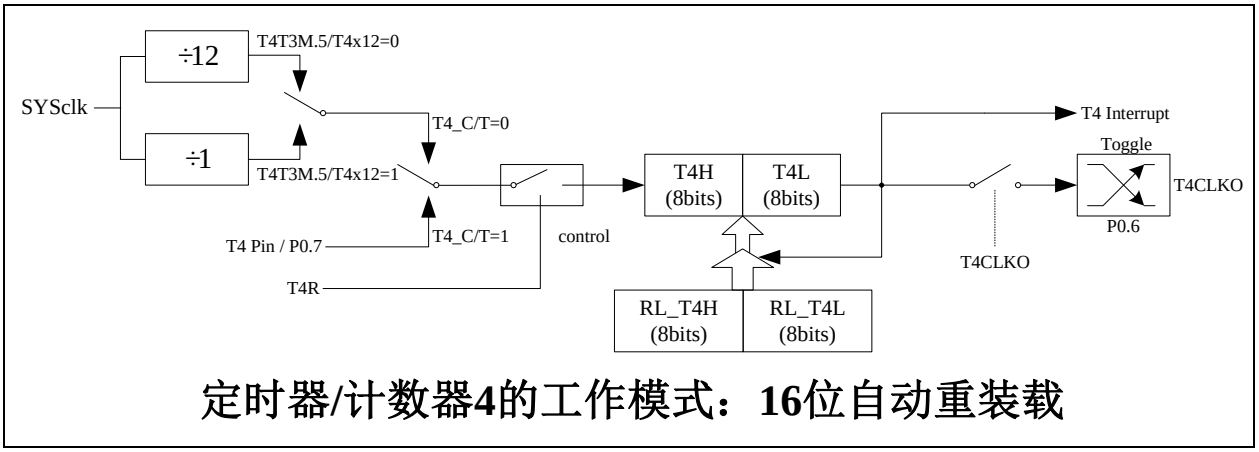
- 当 T4R=0 即定时器/计数器 4 被禁止工作时，对 T4L 写入的内容会同时写入 RL_TL4，对 T4H 写入的内容也会同时写入 RL_TH4。
- 当 T4R=1 即定时器/计数器 4 被允许工作时，对 T4L 写入内容，实际上不是写入当前寄存器 T4L 中,而是写入隐藏的寄存器 RL_TL4 中；对 T4H 写入内容，实际上也不是写入当前寄存器 T4H 中，而是写入隐藏的寄存器 RL_TH4。当读 TH 和 T4L 的内容时，所读的内容就是 T4H 和 T4L

的内容，而不是 RL_TH4 和 RL_TL4 的内容。

这样可以巧妙地实现 16 位重装载定时器。[T4L, T4H]的溢出不仅置位被隐藏的中断请求标志位（定时器 4 的中断请求标志位对用户不可见），使 CPU 转去执行定时器 4 中断的程序，而且会自动将[RL_TL4, RL_TH4]的内容重新装入[T4L, T4H]。

18.6.3.2 定时器/计数器 4 对系统时钟或外部引脚 T4 的时钟输入进行可编程时钟分频输出

定时器/计数器 4 的原理框图如下：



定时器/计数器 4 除可当定时器/计数器使用外，还可作可编程时钟输出。当定时器/计数器 4 用作可编程时钟输出时，不要允许相应的定时器中断,免得 CPU 反复进中断。

当 T4CLKO/T4T3M.4=1 时，P0.6 管脚配置为定时器 4 的时钟输出 T4CLKO。

输出时钟频率=T4 溢出率/2

- 如果 T4_C/T=0，定时器/计数器 T4 对内部系统时钟计数，则：
  - ✓ T4 工作在 1T 模式（T4T3M.5/T4x12=1）时的输出时钟频率=（SYSclk）/（65536-[RL_TH4, RL_TL4]）/2
  - ✓ T4 工作在 12T 模式（T4T3M.5/T4x12=0）时的输出时钟频率=（SYSclk）/12/（65536-[RL_TH4, RL_TL4]）/2
- 如果 T4_C/T=1，定时器/计数器 T4 是对外部脉冲输入（P0.7/T4）计数，则：
  - ✓ 输出时钟频率=（T4_Pin_CLK）/（65536-[RL_TH4, RL_TL4]）/2

上面所有的式子中 RL_TH4 是 T4H 的重装载寄存器，RL_TL4 是 T4L 的重装载寄存器。

18.6.3.3 定时器/计数器 4 作串行口 4 的波特率发生器

定时器/计数器 4 除可当定时器/计数器和可编程时钟输出使用外，还可作串行口 4 波特率发生器。串行口 4 默认选择定时器 2 作为其波特率发生器，但通过设置 S4ST4/S4C0N.6，串行口 4 也可以选择定时器 4 作为其波特率发生器。

S4CON：串行口 4 控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
S4CON	84H	name	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI

S4ST4： 串口 4（UART4）选择定时器 4 作波特率发生器的控制位

0，选择定时器 2 作为串口 4（UART4）的波特率发生器；
1，选择定时器 4 作为串口 4（UART4）的波特率发生器

串行口 4 的工作模式只有两种：模式 0（8 位 UART，波特率可变）和模式 1（9 位 UART，波特率可变）。当串行口 4 被设置为选择定时器 4 作为其波特率发生器时，串行口 4 的波特率按如下公式计算：

串行口 4 的波特率=（定时器 T4 的溢出率）/4

- 当 T4 工作在 1T 模式（T4T3M.5/T4x12=1）时，定时器 4 的溢出率=SYSclk/（65536-[RL_TH4, RL_TL4]）；

即此时，串行口 4 的波特率=SYSclk /（65536-[RL_TH4, RL_TL4]）/4

- 当 T4 工作在 12T 模式（T4T3M.5/T4x12=0）时，定时器 4 的溢出率=SYSclk /12/（65536-[RL_TH4, RL_TL4]）；

即此时，串行口 4 的波特率=SYSclk / 12 /（65536-[RL_TH4, RL_TL4]）/4

上面所有的式子中 RL_TH4 是 T4H 的重装载寄存器，RL_TL4 是 T4L 的重装载寄存器。

18.7 如何将定时器 T0/TT1/T2/T3/T4 的速度提高 12 倍

1、定时器 T0/T1/T2 的速度控制寄存器位：T0x12/T1x12/T2x12

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	name	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

定时器 0、定时器 1 和定时器 2：

STC15W4K32S4 系列是 1T 的 8051 单片机，为了兼容传统 8051，定时器 0、和定时器 1 和定时器 2 复位后是传统 8051 的速度，即 12 分频，这是为了兼容传统 8051。但也可不进行 12 分频，实现真正的 1T。

T0x12： 定时器 0 速度控制位
0，定时器 0 是传统 8051 速度，12 分频；
1，定时器 0 的速度是传统 8051 的 12 倍，不分频

T1x12： 定时器 1 速度控制位
0，定时器 1 是传统 8051 速度，12 分频；
1，定时器 1 的速度是传统 8051 的 12 倍，不分频
如果串口 1 用定时器 1 做波特率发生器，T1x12 位就可以控制 UART 异步串口是 12T 还是 1T 了。

T2x12： 定时器 2 速度控制位
0，定时器 2 是传统 8051 速度，12 分频；
1，定时器 2 的速度是传统 8051 的 12 倍，不分频
如果串口 1 或串口 2 用 T2 作为波特率发生器，则由 T2x12 决定串口 1 或串口 2 是 12T

串口 1 的模式 0：

STC15W4K32S4 系列是 1T 的 8051 单片机，为了兼容传统 8051，串口 1 复位后是兼容传统 8051 的

UART_M0x6：串口 1 模式 0 的通信速度设置位。
0，串口 1 模式 0 的速度是传统 8051 单片机串口的速度，12 分频；
1，串口 1 模式 0 的速度是传统 8051 单片机串口速度的 6 倍，2 分频
如果用定时器 T1 做波特率发生器时，串口 1 的速度由 T1 的溢出率决定

T2R： 定时器 2 允许控制位

0，不允许定时器 2 运行；
1，允许定时器 2 运行

T2_C/T：控制定时器 2 用作定时器或计数器。
0，用作定时器（对内部系统时钟进行计数）：
1，用作计数器（对引脚 T2/P3.1 的外部脉冲进行计数）

EXTRAM：内部/外部 RAM 存取控制位
0，允许使用逻辑上在片外、物理上在片内的扩展 RAM；
1，禁止使用逻辑上在片外、物理上在片内的扩展 RAM

S1ST2： 串口 1（UART1）选择定时器 2 作波特率发生器的控制位
0，选择定时器 1 作为串口 1（UART1）的波特率发生器；
1，选择定时器 2 作为串口 1（UART1）的波特率发生器，此时定时器 1 得到释放，可以作为独立定时器使用

注意：有串口 2 的单片机，串口 2 永远是使用定时器 2 作为波特率发生器，串口 2 不能够选择定时器 1 做波特率发生器，串口 1 可以选择定时器 1 做波特率发生器，也可以选择定时器 2 作为波特率发生器。

2、定时器 T4 和 T3 的速度控制寄存器位：T4x12/T3x12

T4T3M（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	name	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO

B5-T4x12：定时器 4 速度控制位。
0，定时器 4 速度是 8051 单片机定时器的速度，即 12 分频；
1，定时器 4 速度是 8051 单片机定时器速度的 12 倍，即不分频。

B1-T3x12：定时器 3 速度控制位。
0，定时器 3 速度是 8051 单片机定时器的速度，即 12 分频；
1，定时器 3 速度是 8051 单片机定时器速度的 12 倍，即不分频。

B7-T4R：定时器 4 运行控制位。
0，不允许定时器 4 运行
1，允许定时器 4 运行。

B6-T4_C/T：控制定时器 4 用作定时器或计数器。
0，用作定时器（对内部系统时钟进行计数）：
1，用作计数器（对引脚 T4/P0.7 的外部脉冲进行计数）

B4-T4CLKO：是否允许将 P0.6 脚配置为定时器 4（T4）的时钟输出 T4CLKO
1，允许将 P0.6 脚配置为定时器 4 的时钟输出 T4CLKO，输出时钟频率=T4 溢出率/2

- 如果 T4C/T=0，定时器/计数器 T4 是对内部系统时钟计数，则：
  - ✓ T4 工作在 1T 模式（T4T3M.5/T4x12=1）时的输出频率=（SYSclk）/（65536-[RL_TH4, RL_TL4]）/2
  - ✓ T4 工作在 12T 模式（T4T3M.5/T4x12=0）时的输出频率=（SYSclk）/12/（65536-[RL_TH4, RL_TL4]）/2
- 如果 T4_C/T=1，定时器/计数器 T4 是对外部脉冲输入（P0.7/T4）计数，则：
  - ✓ 输出时钟频率=（T4_Pin_CLK）/（65536-[RL_TH4, RL_TL4]）/2

0：不允许将 P0.6 脚配置为定时器 4（T4）的时钟输出 T4CLKO

B3-T3R：定时器 3 运行控制位。

0，不允许定时器 3 运行

1，允许定时器 3 运行。

B2-T3_C/T：控制定时器 3 用作定时器或计数器。

0，用作定时器（对内部系统时钟进行计数）：

1，用作计数器（对引脚 T3/P0.5 的外部脉冲进行计数）

B0-T3CLKO：是否允许将 P0.4 脚配置为定时器 3（T3）的时钟输出 T3CLKO

1，允许将 P0.4 脚配置为定时器 3 的时钟输出 T3CLKO，输出时钟频率=T3 溢出率/2

- 如果 T3_C/T=0，定时器/计数器 T3 是对内部系统时钟计数，则：
  - ✓ T3 工作在 1T 模式（T4T3M.1/T3x12=1）时的输出频率=（SYSclk）/（65536-[RL_TH3, RL_TL3]）/2
  - ✓ T3 工作在 12T 模式（T4T3M.1/T3x12=0）时的输出频率=（SYSclk）/12/（65536-[RL_TH3, RL_TL3]）/2
- 如果 T3_C/T=1，定时器/计数器 T3 是对外部脉冲输入（P0.5/T3）计数，则：
  - ✓ 输出时钟频率=（T3_Pin_CLK）/（65536-[RL_TH3, RL_TL3]）/2

0，不允许将 P0.4 脚配置为定时器 3（T3）的时钟输出 T3CLKO

18.8 可编程时钟输出（也可作分频器使用）

STC15 系列单片机最多有六路可编程时钟输出（如 STC15W4K32S4 系列），如下表所示。对于 STC15 系列 5V 单片机，由于 I/O 口的对外输出速度最快不超过 13.5MHz，所以对外可编程时钟输出速度最快也不超过 13.5MHz；对于 3.3V 单片机，由于 I/O 口的对外输出速度最快不超过 8MHZ，所以对外可编程时钟输出速度最快也不超过 8MHz。

STC15 全系列的可编程时钟输出的类型如下表所示。

可编程时钟输出 / 单片机型号	主时钟输出 (MCLKO/P5.4)	系统时钟输出 (SysClkO/P5.4 或 SysClkO_2/P1.6)	定时器/计数器 0 时钟输出 (T0CLKO/P3.5)	定时器/计数器 1 时钟输出 (T1CLKO/P3.4)	定时器/计数器 2 时钟输出 (T2CLKO/P3.0)	定时器/计数器 3 时钟输出 (T3CLKO/P0.4)	定时器/计数器 4 时钟输出 (T4CLKO/P0.6)
STC15F104W 系列	该系列主时钟输出在 MCLKO/P3.4		√		√		
STC15F408AD 系列	√		√		√		
STC15W204S 系列	√		√		√		
STC15W408AS 系列		√	√		√		
STC15W404S 系列	√（该系列主时钟输出还可在 MCLKO_2/P1.6）		√	√	√		
STC15W1K16S 系列	√（该系列主时钟输出还可在 MCLKO_2/ XTAL2/P1.6）		√	√	√		
STC15F2K60S2 系列	√		√	√	√		
STC15W4K32S4 系列		√	√	√	√	√	√
STC15W1K08PWM 系列		√	√	√	√		
STC15W1K20S-LQFP64		√	√	√	√		

上表中 √ 表示对应的系列有相应的可编程时钟输出。

【特别注意】：对于 STC15W1K16S 系列和 STC15W408S 单片机，若要使用 T0CLKO 时钟输出功能，必须将 P3.5 口设置为强推挽输出模式。

18.8.1 与可编程时钟输出有关的特殊功能寄存器

符号	描述	地址	位地址及其符号								复位值
			MSB							LSB	
AUXR	辅助寄存器	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000 0001B
INT_CLKO AUXR2	External Interrupt enable and Clock output register	8FH	-	EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	T0CLKO	x000 x000B
CLK_DIV (PCON2)	时钟分配器	97H	MCKO_S1	MCKO_S1	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000 0000B
T4T3M	T4 和 T3 的控制寄存器	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO	0000 0000B

特殊功能寄存器 INT_CLKO/AUXR/CLK_DIV/T4T3M 的 C 语言声明：

```
sfr     INT_CLKO= 0x8F;     //新增加的特殊功能寄存器 INT_CLKO 的地址声明
sfr     AUXR= 0x8E;         //特殊功能寄存器 AUXR 的地址声明
sfr     CLK_DIV= 0x97;      //特殊功能寄存器 CLK_DIV 的地址声明
sfr     T4T3M= 0xD1;        //新增加的特殊功能寄存器 T4T3M 的地址声明
```

特殊功能寄存器 INT_CLKO/AUXR/CLK_DIV/T4T3M 的汇编语言声明：

```
INT_CLKO    EQU    8FH    ;新增加的特殊功能寄存器 INT_CLKO 的地址声明
AUXR        EQU    8EH    ;特殊功能寄存器 AUXR 的地址声明
CLK_DIV     EQU    97H    ;特殊功能寄存器 CLK_DIV 的地址声明
T4T3M       EQU    D1H    ;新增加的特殊功能寄存器 T4T3M 的地址声明
```

1. CLK_DIV (PCON2)：时钟分频寄存器（不可位寻址）

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CLK_DIV (PCON2)	97H	name	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0

MCKO_S1	MCKO_S0	主时钟对外分频输出控制位 （主时钟对外输出管脚 MCLKO 或 MCLKO_2 既可对外输出内部 R/C 时钟，也可对外输出外部输入的时钟或外部晶体振荡产生的时钟）
0	0	主时钟不对外输出时钟
0	1	主时钟对外输出时钟，但时钟频率不被分频，输出时钟频率 = MCLK / 1
1	0	主时钟对外输出时钟，但时钟频率被 2 分频，输出时钟频率 = MCLK / 2
1	1	主时钟对外输出时钟，但时钟频率被 3 分频，输出时钟频率 = MCLK / 4

主时钟既可以是内部 R/C 时钟，也可以是外部输入的时钟或外部晶体振荡产生的时钟。但对于无外部时钟源的单片机(STC15F104W 系列、STC15W204S 系列、STC15W404S 系列、STC15W1K16S 系列)以及现供货的 STC15F2K60S2 系列 C 版单片机，其主时钟只能是内部 R/C 时钟。

主时钟可在管脚 MCLKO 或 MCLKO_2 对外输出。其中，STC15 系列 8-pin 单片机(如 STC15F104W 系列)在 MCLKO/P3.4 口对外输出时钟：STC15F2K60S2 系列、STC15W204S 系列及 STC15F408AD 系列单片机在 MCLKO/P5.4 口对外输出时钟：而 STC15W404S 系列及 STC15W1K16S 系列单片机除可在 MCLKO/P5.4 口对外输出时钟外，还可在 MCLKO_2/P1.6 口对外输出时钟。

【特意注意】：STC15W4K32S4 系列、STC15W1K08PWM 系列及 STC15W1K20S-LQFP64 单片机是将系统时钟在管脚 SysClkO/P5.4 或 SysCLKO_2/XTAL2/P1.6 对外分频输出，并可如下分频 SysClk/1，SysClk/2，SysClk/4，SysClk/16。

STC15W408AS 系列单片机也是将系统时钟在管脚 SysClkO/P5.4 或 SysCLKO_2 /XTAL2 /P1.6 对外分频输出，但只可如下分频 SysClk/1，SysClk/2，SysClk/4。

系统时钟是指对主时钟进行分频后供给 CPU、定时器、串行口、SPI、CCP/PWM/PCA、A/D 转换的实际工作时钟。STC15W4K32S4 系列、STC15W408AS 系列、STC15W1K08PWM 系列及 STC15W1K20S-LQFP64 单片机的主时钟既可以是内部 R/C 时钟，也可以是外部输入的时钟或外部晶体振荡产生的时钟。

MCLK 是指主时钟频率，MCLKO 是指系统时钟输出。SysClk 是指系统时钟频率，SysClkO 是指系统时钟输出。

STC15W404S 系列及 STC15W1K16S 系列单片机通过 CLK_DIV.3/MCLKO_2 位来选择是在 MCLKO/P5.4 口对外输出主时钟，还是在 MCLKO_2/P1.6 口对外输出主时钟。

MCLKO_2：主时钟对外输出位置的选择位

0：在 MCLKO/P5.4 日对外输出主时钟；

1：在 MCLKO_2/P1.6 口对外输出主时钟

STC15W404S 系列及 STC15W1K16S 系列单片机的主时钟只能是内部 R/C 时钟。

STC15W4K32S4 系列、STC15W408AS 系列、STC15W1K08PWM 系列及 STC15W1K20S-LQFP64 单片机通过 CLK_DIV.3/SysClkO_2 位来选择是在 SysClkO/P5.4 口对外输出系统时钟，还是在 SysClkO_2/P1.6 口对外输出系统时钟。

SysClkO_2：系统时钟对外输出位置的选择位

0：7 在 SysClkO/P5.4 口对外输出系统时钟；

1：在 SysClkO_2/P1.6 口对外输出系统时钟

系统时钟是指对主时钟进行分频后供给 CPU、定时器、串行口、SPI、CCP/PWMPCA、A/D 转换的实际工作时钟。STC15W4K32S4 系列、STC15W408AS 系列、STC15W1K08PWM 系列及 STC15W1K20S-LQFP64 单片机的主时钟既可以是内部 R/C 时钟，也可以是外部输入的时钟或外部晶体振荡产生的时钟。

CLKS2	CLKS1	CLKS0	系统时钟选择控制位 (系统时钟是指对主时钟进行分频后供给 CPU、串行口、SPI、定时器、CCP/PWM/PCA、A/D 转换的实际工作时钟)
0	0	0	主时钟频率/1，不分频
0	0	1	主时钟频率/2
0	1	0	主时钟频率/4
0	1	1	主时钟频率/8
1	0	0	主时钟频率/16
1	0	1	主时钟频率/32
1	1	0	主时钟频率/64
1	1	1	主时钟频率/128

主时钟既可是内部 R/C 时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。

2. INT_CLKO (AUXR2)：External Interrupt Enable and Clock Output register

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
INT_CLKO AUXR2	8FH	name	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO

B0- T0CLKO：是否允许将 P3.5/T1 脚配置为定时器 0（T0）的时钟输出 T0CLKO

1，将 P3.5/T1 管脚配置为定时器 0 的时钟输出 T0CLKO，输出时钟频率=T0 溢出率/2

若定时器/计数器 T0 工作在定时器模式 0（16 位自动重装载模式）时，

- 如果 C/T=0，定时器/计数器 T0 是对内部系统时钟计数，则：
  - ✓ T0 工作在 1T 模式（AUXR.7/T0x12=1）时的输出频率=（SYSclk/（65536-[RL_TH0, RL_TL0]）/2

- ✓ T0 工作在 12T 模式（AUXR.7/T0x12=0）时的输出频率=（SYSclk）/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

若定时器/计数器 T0 工作在定时器模式 2（8 位自动重装模式），

➢ 如果 C/T=0，定时器/计数器 T0 是对内部系统时钟计数，则：

- ✓ T0 工作在 1T 模式（AUXR.7/T0x12=1）时的输出频率=（SYSclk）/（256-TH0）/2
- ✓ T0 工作在 12T 模式（AUXR.7/T0x12=0）时的输出频率=（SYSclk）/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

0，不允许 P3.5/T1 管脚被配置为定时器 0 的时钟输出

B1-T1CLKO：是否允许将 P3.4/T0 脚配置为定时器 1（T1）的时钟输出 T1CLKO

1，将 P3.4/T0 管脚配置为定时器 1 的时钟输出 T1CLKO，输出时钟频率=T1 溢出率/2

若定时器/计数器 T1 工作在定时器模式 0（16 位自动重装载模式），

➢ 如果 C/T=0，定时器/计数器 T1 是对内部系统时钟计数，则：

- ✓ T1 工作在 IT 模式（AUXR.6/T1x12=1）时的输出频率=（SYSclk）/（65536-[RL_TH1, RL_TL1]）/2
- ✓ T1 工作在 12T 模式（AUXR.6/T1x12=0）时的输出频率=（SYSclk）/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

若定时器/计数器 T1 工作在模式 2（8 位自动重装模式），

➢ 如果 C/T=0，定时器/计数器 T1 是对内部系统时钟计数，则：

- ✓ T1 工作在 1T 模式（AUXR.6/T1x12=1）时的输出频率=（SYSclk）/（256-TH1）/2
- ✓ T1 工作在 12T 模式（AUXR.6/T1x12=0）时的输出频率=（SYSclk/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

0，不允许 P3.4/T0 管脚被配置为定时器 1 的时钟输出

B2-T2CLKO：是否允许将 P3.0 脚配置为定时器 2（T2）的时钟输出 T2CLKO

1，允许将 P3.0 脚配置为定时器 2 的时钟输出 T2CLKO，输出时钟频率=T2 溢出率/2

➢ 如果 T2_C/T=0，定时器/计数器 T2 是对内部系统时钟计数，则：

- ✓ T2 工作在 1T 模式（AUXR.2/T2x12=1）时的输出频率=（SYSclk）/（65536-[RL_TH2, RL_TL2]）/2
- ✓ T2 工作在 12T 模式（AUXR.2/T2x12=0）时的输出频率=（SYSclk）/12/（65536-[RL_TH2, RL_TL2]）/2

➢ 如果 T2_C/T=1，定时器/计数器 T2 是对外部脉冲输入（P3.1/T2）计数，则：

- ✓ 输出时钟频率=（T2_Pin_CLK）/（65536-[RL_TH2, RL_TL2]）/2

0，不允许将 P3.0 脚配置为定时器 2（T2）的时钟输出 T2CLKO

B4-EX2：允许外部中断 2（INT2）

B5-EX3：允许外部中断 3（INT3）

B6-EX4：允许外部中断 4（INT4）

3、辅助特殊功能寄存器：AUXR（地址：0x8E）

AUXR：Auxiliary register（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	name	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

B7-T0x12：定时器 0 速度控制位。

0，定时器 0 速度是传统 8051 单片机定时器的速度，即 12 分频；

1，定时器 0 速度是传统 8051 单片机定时器速度的 12 倍，即不分频。

B6-T1x12：定时器 1 速度控制位。

0，定时器 1 速度是传统 8051 单片机定时器的速度，即 12 分频；

1，定时器 1 速度是传统 8051 单片机定时器速度的 12 倍，即不分频。

如果串口 1 用 T1 作为波特率发生器，则由 T1x12 位决定串口 1 是 12T 还是 1T。

B5-UART_M0x6：串口 1 模式 0 的通信速度设置位。

0，串口 1 模式 0 的速度是传统 8051 单片机串口的速度，即 12 分频;

1，串口 1 模式 0 的速度是传统 8051 单片机串口速度的 6 倍，即 2 分频。

B4-T2R：定时器 2 运行控制位。

0，不允许定时器 2 运行；

1，允许定时器 2 运行。

B3-T2_C/T：控制定时器 2 用作定时器或计数器。

0，用作定时器（对内部系统时钟进行计数）；

1，用作计数器（对引脚 T2/P3.1 的外部脉冲进行计数）

B2-T2x12：定时器 2 速度控制位

0，定时器 2 是传统 8051 单片机的速度，12 分频；

1，定时器 2 的速度是传统 8051 单片机速度的 12 倍，不分频

如果串口 1 或串口 2 用 T2 作为波特率发生器,则由 T2x12 决定串口 1 或串口 2 是 12T 还是 1T。

B1-EXTRAM：内部/外部 RAM 存取控制位。

0，允许使用逻辑上在片外、物理上在片内的扩展 RAM；

1，禁止使用逻辑上在片外、物理上在片内的扩展 RAM。

B0-S1ST2：串口 1（UART1）选择定时器 2 作波特率发生器的控制位。

0：选择定时器 1 作为串口 1（UART1）的波特率发生器；

1：选择定时器 2 作为串口 1（UART1）的波特率发生器，此时定时器 1 得到释放，可以作为独立定时器使用。

4、定时器 T4 和 T3 的控制寄存器：T4T3M（地址：0xD1）

T4T3M（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	name	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO

B7-T4R：定时器 4 运行控制位。

0，不允许定时器 4 运行；

1，允许定时器 4 运行。

B6-T4C：控制定时器 4 用作定时器或计数器。

0，用作定时器（对内部系统时钟进行计数）；

1，用作计数器（对引脚 T4/P0.7 的外部脉冲进行计数）

B5-T4x12：定时器 4 速度控制位。

0，定时器 4 速度是 8051 单片机定时器的速度，即 12 分频；

1，定时器 4 速度是 8051 单片机定时器速度的 12 倍，即不分频。

B4-T4CLKO：是否允许将 P0.6 脚配置为定时器 4（T4）的时钟输出 T4CLKO

1，允许将 P0.6 脚配置为定时器 4 的时钟输出 T4CLKO，输出时钟频率=T4 溢出率/2

- 如果 T4_C/T=0，定时器/计数器 T4 是对内部系统时钟计数，则：
  - T4 工作在 1T 模式（T4T3M.5/T4x12=1）时的输出频率=（SYSclk）/（65536-[RL_TH4, RL_TL4]）/2
  - T4 工作在 12T 模式（T4T3M.5/T4x12=0）时的输出频率=（SYSclk）/12/（65536-[RL_TH4, RL_TL4]）/2
- 如果 T4_C/T=1，定时器/计数器 T4 是对外部脉冲输入（P0.7/T4）计数，则：
  - 输出时钟频率=（T4_Pin_CLK）/（65536-[RL_TH4, RL_TL4]）/2

0，不允许将 P0.6 脚配置为定时器 4（T4）的时钟输出 T4CLKO

B3-T3R：定时器 3 运行控制位。

0，不允许定时器 3 运行；

1，允许定时器 3 运行。

B2-T3_C/T：控制定时器 3 用作定时器或计数器。

0，用作定时器（对内部系统时钟进行计数）；

1，用作计数器（对引脚 T3/P0.5 的外部脉冲进行计数）

B1-T3x12：定时器 3 速度控制位。

0，定时器 3 速度是 8051 单片机定时器的速度，即 12 分频；

1，定时器 3 速度是 8051 单片机定时器速度的 12 倍，即不分频。

B0-T3CLKO：是否允许将 P0.4 脚配置为定时器 3（T3）的时钟输出 T3CLKO

1，允许将 P0.4 脚配置为定时器 3 的时钟输出 T3CLKO，输出时钟频率=T3 溢出率/2

- 如果 T3_C/T=0，定时器/计数器 T3 是对内部系统时钟计数，则：
  - T3 工作在 1T 模式（T4T3M.1/T3x12=1）时的输出频率=（SYSclk）/（65536-[RL_TH3, RL_TL3]）/2
  - T3 工作在 12T 模式（T4T3M.1/T3x12=0）时的输出频率=（SYSclk）/12/（65536-[RL_TH3, RL_TL3]）/2
- 如果 T3_C/T=1，定时器/计数器 T3 是对外部脉冲输入（P0.5/T3）计数，则：
  - 输出时钟频率=（T3_Pin_CLK）/（65536-[RL_TH3, RL_TL3]）/2

0，不允许将 P0.4 脚配置为定时器 3（T3）的时钟输出 T3CLKO

18.8.2 主时钟输出及其测试程序（C 和汇编）

主时钟可以是内部高精度 R/C 时钟，也可以是外部输入的时钟或外部晶体振荡产生的时钟。由于 STC15 系列 5V 单片机 I/O 口的对外输出速度最快不超过 13.5MHz，所以 5V 单片机的对外可编程时钟输出速度最快也不超过 13.5Mz，如果频率过高，需进行分频输出；而 3.3V 单片机 I/O 口的对外输出速度最快不超过 8MHz，故 3.3V 单片机的对外可编程时钟输出速度最快也不超过 8MHz，如果频率过高，需进行分频输出。

主时钟对外输出控制寄存器：CLK_DIV（不可位寻址）与 INTCLKO（不可位寻址）

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
CLK_DIV（PCON2）	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000 0000

如何利用 MCLKO/P5.4 或 MCLKO 2/XTAL2/P1.6 管脚输出时钟

MCLKO/P5.4 或 MCLKO 2/XTAL2/P1.6 的时钟输出控制由 CLK_DIV 寄存器的 MCKO_S1 和 MCKO_S0 位控制。通过设置 MCKO_S1（CLK_DIV.7）和 MCKO_S0（CLK_DIV.6）可将 MCLKO/P5.4

管脚配置为主时钟输出同时还可以设置该主时钟的输出频率。

特殊功能寄存器：CLK_DIV（地址：97H）

MCKO_S1	MCKO_S0	主时钟对外分频输出控制位 (主时钟对外输出管脚 MCLKO 或 MCLKO_2 既可对外输出内部 R/C 时钟，也可对外输出外部输入的时钟或外部晶体振荡产生的时钟)
0	0	主时钟不对外输出时钟
0	1	主时钟对外输出时钟，但时钟频率不被分频，输出时钟频率 = MCLK / 1
1	0	主时钟对外输出时钟，但时钟频率被 2 分频，输出时钟频率 = MCLK / 2
1	1	主时钟对外输出时钟，但时钟频率被 4 分频，输出时钟频率 = MCLK / 4

主时钟既可以是内部 R/C 时钟，也可以是外部输入的时钟或外部晶体振荡产生的时钟。但对于无外部时钟源的单片机（STC15F104W 系列、STC15W204S 系列、STC15W404S 系列、STC15W1K16S 系列）以及现供货的 STC15F2K60S2 系列 C 版单片机，其主时钟只能是内部 R/C 时钟。

主时钟可在管脚 MCLKO 或 MCLKO_2 对外输出。

- STC15 系列 8-pin 单片机（如 STC15F104W 系列）在 MCLKO/P3.4 口对外输出时钟；
- STC15F2K60S2 系列、STC15W204S 系列及 STC15F408AD 系列单片机在 MCLKO/P5.4 口对外输出时钟；
- STC15W404S 系列及 STC15W1K16S 系列单片机除可在 MCLKO/P5.4 口对外输出时钟外，还可在 MCLKO_2/P1.6 口对外输出时钟。

【特意注意】：STC15W4K32S4 系列、STC15W1K08PWM 系列及 STC15W1K20S-LQFP64 单片机是将系统时钟在管脚 SysClkO/P5.4 或 SysCLKO_2/XTAL2/P1.6 对外分频输出，并可如下分频 SysClk/1，SysClk/2，SysClk/4，SysClk/16。
STC15W408AS 系列单片机也是将系统时钟在管脚 SysClkO/P5.4 或 SysCLKO_2/XTAL2/P1.6 对外分频输出，但只可如下分频 SysClk/1，SysClk/2，SysClk/4。

系统时钟是指对主时钟进行分频后供给 CPU、定时器、串行口、SPI、CCPIPWM/PCA、A/D 转换的实际工作时钟。STC15W4K32S4 系列、STC15W408AS 系列、STC15W1K08PWM 系列及 STC15W1K20S-LQFP64 单片机的主时钟既可以是内部 R/C 时钟，也可以是外部输入的时钟或外部晶体振荡产生的时钟。

MCLK 是指主时钟频率，MCLKO 是指系统时钟输出。SysClk 是指系统时钟频率，SysClkO 是指系统时钟输出。

STC15W404S 系列及 STC15W1K16S 系列单片机通过 CLK_DIV3/MCLKO_2 位来选择是在 MCLKO/P5.4 口对外输出主时钟，还是在 MCLKO_2/P1.6 口对外输出主时钟。

MCLKO_2：主时钟对外输出位置的选择位

0：在 MCLKO/P5.4 日对外输出主时钟；

1：在 MCLKO_2/P1.6 日对外输出主时钟

STC15W404S 系列及 STC15W1K16S 系列单片机的主时钟只能是内部 R/C 时钟。

STC15W4K32S4 系列、STC15W408AS 系列、STC15W1K08PWM 系列及 STC15W1K20S-LQFP64 单片机通过 CLK_DIV.3/SysClkO_2 位来选择是在 SysClkO/P5.4 口对外输出系统时钟，还是在 SysClkO_2/P1.6 口对外输出系统时钟。

SysClkO_2：系统时钟对外输出位置的选择位

0：在 SysClkO/P5.4 口对外输出系统时钟；

1：在 SysClkO_2/P1.6 口对外输出系统时钟

系统时钟是指对主时钟进行分频后供给 CPU、定时器、串行口、SPI、CCP/PWM/PCA、A/D 转换的

实际工作时钟。STC15W4K32S4 系列、STC15W408AS 系列、STC15W1K08PWM 系列及 STC15W1K20S-LQFP64 单片机的主时钟既可以是内部 R/C 时钟，也可以是外部输入的时钟或外部晶体振荡产生的时钟。

由于 STC15 系列 5V 单片机 I/O 口的对外输出速度最快不超过 13.5MHz，所以 5V 单片机的对外可编程时钟输出速度最快也不超过 13.5MHz，如果频率过高，需进行分频输出。

而 3.3V 单片机 I/O 口的对外输出速度最快不超过 8MHz，故 3.3V 单片机的对外可编程时钟输出速度最快也不超过 8MHz，如果频率过高，需进行分频输出。

下面是主时钟输出的示例程序：

1. C 程序：

```
/*----演示 STC15F2K60S2 系列单片机的主时钟输出 ----------------------------*/
/*----Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"
typedef     unsigned char     BYTE;
typedef     unsigned int      WORD;
#define     FOSC              18432000L
//-----------------------------------------
sfr         CLK_DIV = 0x97;             //时钟分频寄存器
//-----------------------------------------
void main()
{
    CLK_DIV = 0x40;                     //0100,0000 P5.4 输出频率为 SYSclk
//  CLK_DIV = 0x80;                     //1000,0000 P5.4 输出频率为 SYSclk/2
//  CLK_DIV = 0xC0;                     //1100,0000 P5.4 输出频率为 SYSclk/4
    while (1); //程序终止
}
```

2. 汇编程序：

```
/*----演示 STC15F2K60S2 系列单片机的主时钟输出 ----------------------------*/
/*----Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
;假定测试芯片的工作频率为 18.432MHz

CLK_DIV     DATA    097H                ;IRC 时钟输出控制寄存器
;-----------------------------------------
;interrupt vector table
    ORG     0000H
    LJMP    START                       ;复位入口

;-----------------------------------------
    ORG     0100H
START:
    MOV     SP,  #3FH                   ;initial SP
```

```
    MOV     CLK_DIV, #40H           ;0100,0000 P5.4 输出频率为 SYSclk
;   MOV     CLK_DIV, #80H           ;1000,0000 P5.4 输出频率为 SYSclk/2
;   MOV     CLK_DIV, #C0H           ;1100,0000 P5.4 输出频率为 SYSclk/4
    SJMP    $

;---------------------------------------
END
```

18.8.3 定时器 0 对系统时钟或外部引脚 T0 的时钟输入进行可编程分频输出

----及测试程序（C 和汇编）

如何利用 T0CLKO/P3.5 管脚输出时钟

T0CLKO/P3.5 管脚是否输出时钟由 INT_CLKO（AUXR2）寄存器的 T0CLKO 位控制
AUXR2.0-T0CLKO：1，允许时钟输出
0，禁止时钟输出

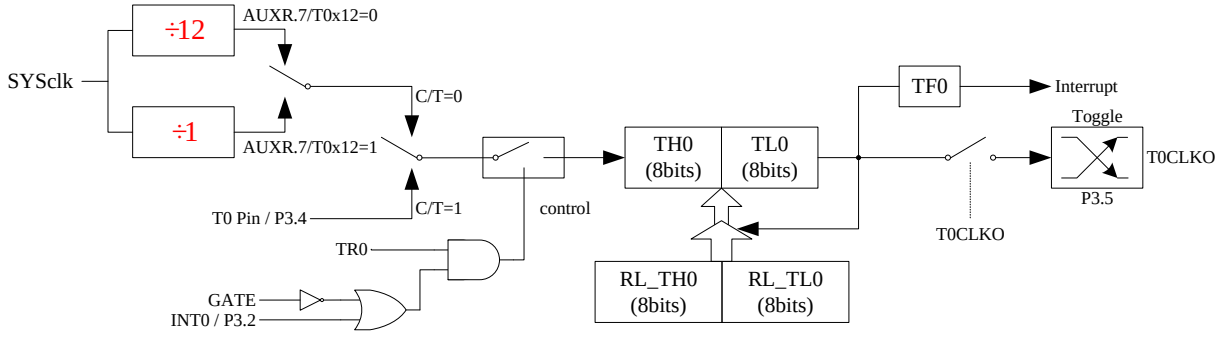
T0CLKO 的输出时钟频率由定时器 0 控制，相应的定时器 0 需要工作在定时器的模式 0（16 位自动重装载模式）或模式 2（8 位自动重装载模式），不要允许相应的定时器中断，免得 CPU 反复进中断，当然在特殊情况下也可允许相应的定时器中断。

新增加的特殊功能寄存器：INT_CLKO（AUXR2）（地址：0x8F）

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=T0 溢出率/2
若定时器/计数器 T0 工作在定时器模式 0（16 位自动重装载模式）时，（如下图所示）

- 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）/（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）/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

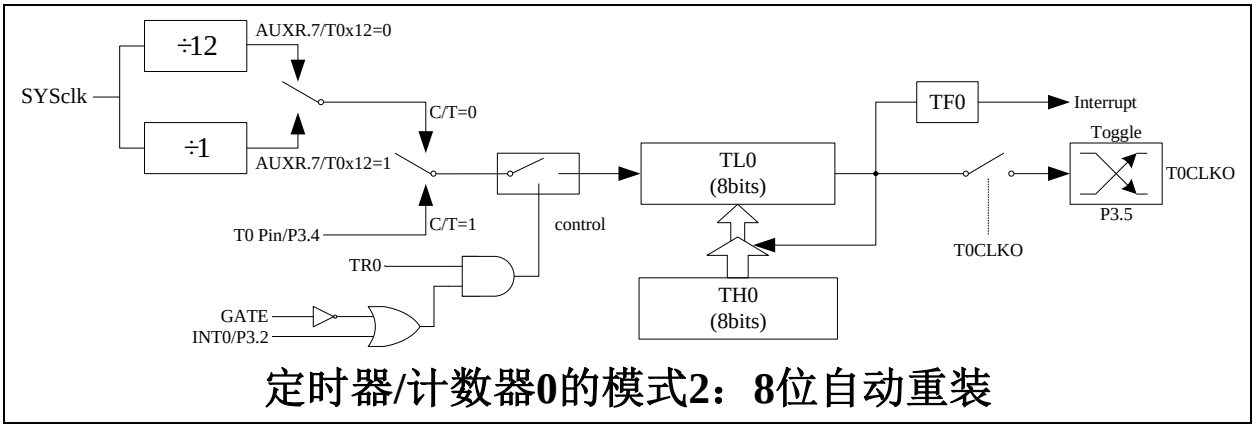
RL_TH0 为 TH0 的重装载寄存器，RL_TL0 为 TL0 的重装载寄存器。



定时器/计数器0的模式0：16位自动重装

当 T0CLKO/INT_CLKO.0=1 且定时器/计数器 T0 工作在定时器模式 2（8 位自动重装模式）时，（如下图所示）

- 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）/（256-TH0）/2
  - ✓ T0 工作在 12T 模式（AUXR.7/T0x12=0）时的输出时钟频率=（SYSclk）/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



【特别注意】：对于 STC15W1K16S 系列和 STC15W408S 单片机，若要使用 T0CLKO 时钟输出功能，必须将 P3.5 口设置为强推挽输出模式。

下面是定时器 0 对内部系统时钟或外部引脚 T0/P3.4 的时钟输入进行可编程时钟分频输出的程序举例（C 和汇编）：

1.C 程序:

```
/*----演示 STC15F2K60S2 系列单片机定时器 0 的可编程时钟分频输出----------*/
/*----在 Kei C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"

typedef     unsigned char     BYTE;
typedef     unsigned int      WORD;

#define     FOSC              18432000L
//-----------------------------------------------
sfr         AUXR = 0x8e;                          //辅助特殊功能寄存器
sfr         INT_CLKO = 0x8f;                      //唤醒和时钟输出功能寄存器
sbit        T0CLKO = P3^5;                        //定时器 0 的时钟输出脚

#define     F38_4KHz          (65536-FOSC/2/38400)      //1T 模式
//#define     F38_4KHz          (65536-FOSC/2/12/38400)   //12T 模式
//-----------------------------------------------
void main()
{
    AUXR |= 0x80;                                 //定时器 0 为 1T 模式
//  AUXR &= ~0x80;                                //定时器 0 为 12T 模式
    TMOD = 0x00;                                  //设置定时器为模式 0(16 位自动重装载)
```

```
    TMOD &= ~0x04;                  //C/T0=0, 对内部时钟进行时钟输出
//  TMOD |= 0x04;                   //C/T0=1, 对 T0 引脚的外部时钟进行时钟输出
    TL0 = F38_4KHz;                 //初始化计时值
    TH0 = F38_4KHz >> 8;
    TR0 = 1;
    INT_CLKO = 0x01;                //使能定时器 0 的时钟输出功能
    while (1);                      //程序终止
}
```

2. 汇编程序：

```
/*----演示 STC15F2K60S2 系列单片机定时器 0 的可编程时钟分频输出-----------*/
/*----在 Kei C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
;假定测试芯片的工作频率为 18.432MHz

AUXR        DATA    08EH        ;辅助特殊功能寄存器
INT_CLKO    DATA    08FH        ;唤醒和时钟输出功能寄存器
T0CLKO      BIT     P3.5        ;定时器 0 的时钟输出脚
F38_4KHz    EQU     0FF10H      ;38.4KHz(1T 模式下, 65536-18432000/2/38400)
;F38_4KHz   EQU     0FFECH      ;38.4KHz(12T 模式下, (65536-18432000/2/12/38400))

;-----------------------------------------------
    ORG     0000H
    LJMP    START               ;复位入口
;---------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    ORL     AUXR, #80H          ;定时器 0 为 1T 模式
;   ANL     AUXR, #7FH          ;定时器 0 为 12T 模式

    MOV     TMOD, #00H          ;设置定时器为模式 0(16 位自动重装载)

    ANL     TMOD, #0FBH         ;C/T0=0, 对内部时钟进行时钟输出
;   ORL     TMOD, #04H          ;C/T0=1, 对 T0 引脚的外部时钟进行时钟输出

    MOV     TL0, #LOW F38_4KHz  ;初始化计时值
    MOV     TH0, #HIGH F38_4KHz
    SETB    TR0
    MOV     INT_CLKO,   #01H    ;使能定时器 0 的时钟输出功能

    SJMP    $                   ;程序终止
;-----------------------------------------------
END
```

18.8.4 定时器 1 对系统时钟或外部引脚 T1 的时钟输入进行可编程分频输出

----及测试程序（C 和汇编）

如何利用 T1CLKO/P3.4 管脚输出时钟

T1CLKO/P3.4 管脚是否输出时钟由 INT_CLKO（AUXR2）寄存器的 T1CLKO 位控制

AUXR2.1-T1CLKO：1，允许时钟输出

0，禁止时钟输出

T1CLKO 的输出时钟频率由定时器 1 控制，相应的定时器 1 需要工作在定时器的模式 0（16 位自动重装载模式）或模式 2（8 位自动重装载模式），不要允许相应的定时器中断，免得 CPU 反复进中断，当然在特殊情况下也可允许相应的定时器中断。

新增加的特殊功能寄存器：INT_CLKO（AUXR2）（地址：0x8F）

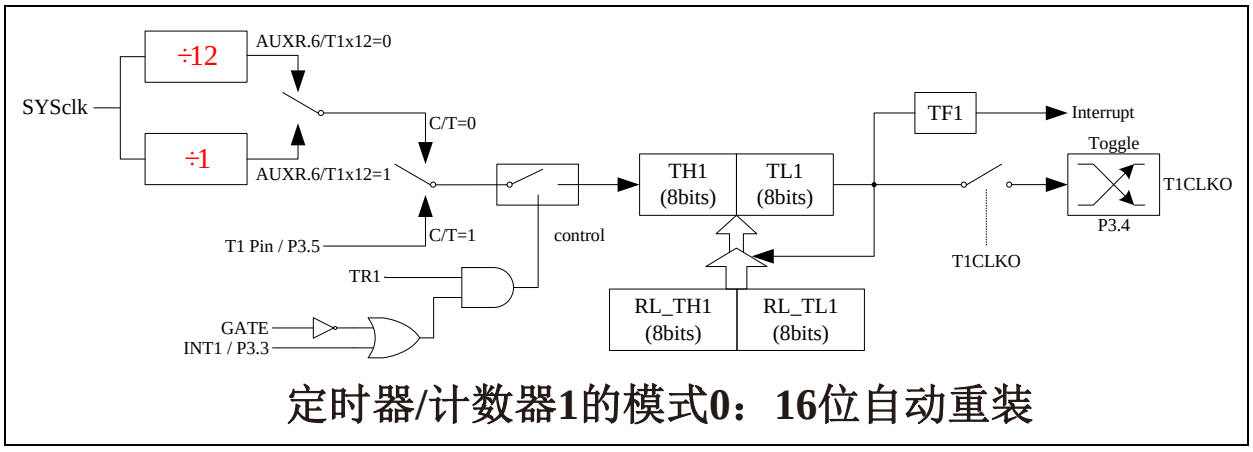
当 T1CLKO/INT_CLKO.1=1 时，P3.4/T0 管脚配置为定时器 1 的时钟输出 T1CLKO。

输出时钟频率=T1 溢出率/2

若定时器/计数器 T1 工作在定时器模式 0（16 位自动重装载模式）时，（如下图所示）

- 如果 C/T=0，定时器/计数器 T1 是对内部系统时钟计数，则：
  - T1 工作在 1T 模式（AUXR.6/T1x12=1）时的输出频率=（SYSclk）/（65536-[RL_TH1, RL_TL1]）/2
  - T1 工作在 12T 模式（AUXR.6/T1x12=0）时的输出频率=（SYSclk）/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

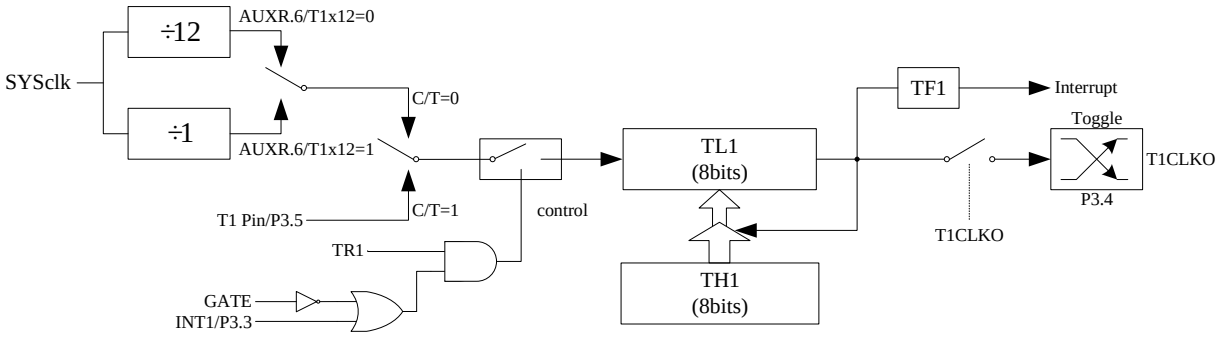
RL_TH0 为 TH1 的重装载寄存器，RL_TL1 为 TL0 的重装载寄存器。



定时器/计数器1的模式0：16位自动重装

当 T1CLKO/INT_CLKO.1=1 且定时器/计数器 T1 工作在定时器模式 2（8 位自动重装模式）时，（如下图所示）

- 如果 C/T=0，定时器/计数器 T1 是对内部系统时钟计数，则：
  - T1 工作在 1T 模式（AUXR.6/T1x12=1）时的输出频率=（SYSclk）/（256-TH1）/2
  - T1 工作在 12T 模式（AUXR.6/T1x12=0）时的输出频率=（SYSclk）/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



定时器/计数器1的模式2：8位自动重装载

下面是定时器 1 对内部系统时钟或外部引脚 T1/P3.5 的时钟输入进行可编程时钟分频输出的程序举例（C 和汇编）：

1.C 程序:

```
/*----演示 STC15 系列单片机定时器 1 的可编程时钟分频输出----------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可.----*/
//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"
typedef     unsigned char     BYTE;
typedef     unsigned int      WORD;

#define     FOSC              18432000L
//-----------------------------------------------
sfr         AUXR = 0x8e;                    //辅助特殊功能寄存器
sfr         INT_CLKO = 0x8f;                //唤醒和时钟输出功能寄存器
sbit        T1CLKO = P3^4;                  //定时器 1 的时钟输出脚

#define     F38_4KHz (65536-FOSC/2/38400)       //1T 模式
//#define     F38_4KHz (65536-FOSC/2/12/38400)    //12T 模式
//-----------------------------------------------
void main()
{
    AUXR |= 0x40;                           //定时器 1 为 1T 模式
//  AUXR &= ~0x40;                          //定时器 1 为 12T 模式
    TMOD = 0x00;                            //设置定时器为模式 1(16 位自动重装载)
    TMOD &= ~0x40;                          //C/T1=0, 对内部时钟进行时钟输出
//  TMOD |= 0x40;                           //C/T1=1, 对 T1 引脚的外部时钟进行时钟输出
    TL1 = F38_4KHz;                         //初始化计时值
    TH1 = F38_4KHz >> 8;
    TR1 = 1;
    INT_CLKO = 0x02;                        //使能定时器 1 的时钟输出功能
    while (1);                              //程序终止
}
```

2. 汇编程序：

```
/*----演示 STC15 系列单片机定时器 1 的可编程时钟分频输出----------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可.-------*/
;假定测试芯片的工作频率为 18.432MHz

AUXR        DATA    08EH                ;辅助特殊功能寄存器
INT_CLKO    DATA    08FH                ;唤醒和时钟输出功能寄存器
T1CLKO      BIT     P3.4                ;定时器 1 的时钟输出脚

F38_4KHz    EQU     0FF10H              ;38.4KHz(1T 模式下, 65536-18432000/2/38400)
;F38_4KHz   EQU     0FFECH              ;38.4KHz(12T 模式下, (65536-18432000/2/12/38400)

    ORG     0000H
    LJMP    START                       ;复位入口
;----------------------------------------------
    ORG     0100H

START:
    MOV     SP, #3FH
    ORL     AUXR, #40H                  ;定时器 1 为 1T 模式
;   ANL     AUXR, #0BFH                 ;定时器 1 为 12T 模式

    MOV     TMOD, #00H                  ;设置定时器为模式 0(16 位自动重装载)

    ANL     TMOD, #0BFH                 ;C/T1=0, 对内部时钟进行时钟输出
;   ORL     TMOD, #40H                  ;C/T1=1, 对 T1 引脚的外部时钟进行时钟输出

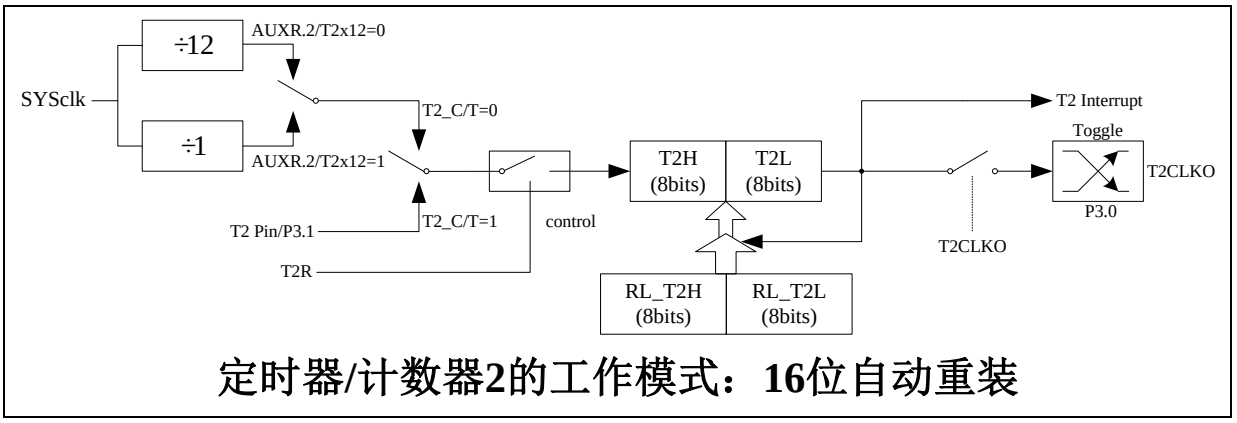
    MOV     TL1, #LOW F38_4KHz          ;初始化计时值
    MOV     TH1, #HIGH F38_4KHz
    SETB    TR1
    MOV     INT_CLKO, #02H              ;使能定时器 1 的时钟输出功能
    SJMP    $                           ;程序终止
;----------------------------------------------
END
```

18.8.5 定时器 2 对系统时钟或外部引脚 T2 的时钟输入进行可编程分频输出

----及测试程序（C 和汇编）

T2 可以当定时器用，也可以当串口的波特率发生器和可编程时钟输出。

定时器 2 的原理框图如下：



如何利用 T2CLKO/P3.0 管脚输出时钟

AUXR2.2-T2CLKO：是否允许将 P3.0 脚配置为定时器 2（T2）的时钟输出 T2CLKO

1，允许将 P3.0 脚配置为定时器 2（T2）的时钟输出 T2CLKO

0：不允许将 P3.0 脚配置为定时器 2（T2）的时钟输出 T2CLKO

当 T2CLKO/INT_CLKO.2=1 时，P3.0 管脚配置为定时器 2 的时钟输出 T2CLKO。

输出时钟频率=T2 溢出率/2

- 如果 T2_C/T=0，定时器/计数器 T2 对内部系统时钟计数，则：
  - ✓ T2 工作在 1T 模式（AUXR.2/T2x12=1）时的输出时钟频率=（SYSclk/（65536-[RL_TH2, RL_TL2]）/2
  - ✓ T2 工作在 12T 模式（AUXR.2/T2x12=0）时的输出时钟频率=（SYSclk/12/（65536-[RL_TH2, RL_TL2]）/2
- 如果 T2_C/T=1，定时器/计数器 T2 是对外部脉冲输入（P3.1/T2）计数，则：
  - ✓ 输出时钟频率=（T2_Pin_CLK）/（65536-[RL_TH2, RL_TL2]）/2

RL_TH2 为 T2H 的重装载寄存器，RL_TL2 为 T2L 的重装载寄存器。

用户在程序中如何具体设置 T2CLKO/P3.0 管脚输出时钟

1.对定时器 2 寄存器 T2H/T2L 送 16 位重装载值，[T2H, T2L] = #reload data

2.对 AUXR 寄存器中的 T2R 位置 1，让定时器 2 运行

3.对 AUXR2/INT_CLKO 寄存器中的 T2CLKO 位置 1，让定时器 2 的溢出在 P3.0 口输出时钟。

注意：当定时器/计数器 2 用作可编程时钟输出时，不要允许相应的定时器中断，免得 CPU 反复进中断，在特殊情况下也可允许定时器/计数器 2 中断。

下面是定时器 2 对内部系统时钟或外部引脚 T2/P3.1 的时钟输入进行可编程时钟分频输出的程序举例（C 和汇编）：

1.C 程序：

```
/*----STC15F2K60S2 系列定时器 2 的可编程时钟分频输出举例--------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-----*/
//假定测试芯片的工作频率为 18.432MHz
```

```
#include "reg51.h"
typedef     unsigned char     BYTE;
typedef     unsigned int      WORD;

#define     FOSC              18432000L
//-----------------------------------------------
sfr         AUXR = 0x8e;                          //辅助特殊功能寄存器
sfr         INT_CLKO = 0x8f;                      //唤醒和时钟输出功能寄存器
sfr         T2H = 0xD6;                           //定时器 2 高 8 位
sfr         T2L = 0xD7;                           //定时器 2 低 8 位
sbit        T2CLKO = P3^0;                        //定时器 2 的时钟输出脚
#define     F38_4KHz          (65536-FOSC/2/38400)      //1T 模式
//#define   F38_4KHz          (65536-FOSC/2/12/38400)   //12T 模式

//-----------------------------------------------
void main()
{
    AUXR |= 0x04;                                 //定时器 2 为 1T 模式
//  AUXR &= ~0x04;                                //定时器 2 为 12T 模式
    AUXR &= ~0x08;                                //T2_C/T=0, 对内部时钟进行时钟输出
//  AUXR |= 0x08;                                 //T2_C/T=1, 对 T2(P3.1)引脚的外部时钟进行时钟输出
    T2L = F38_4KHz;                               //初始化计时值
    T2H = F38_4KHz >> 8;
    AUXR |= 0x10;                                 //定时器 2 开始计时
    INT_CLKO = 0x04;                              //使能定时器 2 的时钟输出功能
    while (1);                                    //程序终止
}
```

2. 汇编程序：

```
/*----STC15F2K60S2 系列定时器 2 的可编程时钟分频输出举例-------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-----------*/
;假定测试芯片的工作频率为 18.432MHz

AUXR        DATA    08EH                      ;辅助特殊功能寄存器
INT_CLKO    DATA    08FH                      ;唤醒和时钟输出功能寄存器
T2H         DATA    0D6H                      ;定时器 2 高 8 位
T2L         DATA    0D7H                      ;定时器 2 低 8 位
T2CLKO      BIT     P3.0                      ;定时器 2 的时钟输出脚
F38_4KHz    EQU     0FF10H                    ;38.4KHz(1T 模式下, 65536-18432000/2/38400)
;F38_4KHz   EQU     0FFECH                    ;38.4KHz(12T 模式下, (65536-18432000/2/12/38400)

;-----------------------------------------------
    ORG     0000H
    LJMP    START                             ;复位入口
```

```
;---------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    ORL     AUXR, #04H                  ;定时器 2 为 1T 模式

;   ANL     AUXR, #0FBH                 ;定时器 2 为 12T 模式
    ANL     AUXR, #0F7H                 ;T2_C/T=0, 对内部时钟进行时钟输出
;   ORL     AUXR, #08H                  ;T2_C/T=1, 对 T2(P3.1)引脚的外部时钟进行时钟输出
    MOV     T2L, #LOW F38_4KHz          ;初始化计时值
    MOV     T2H, #HIGH F38_4KHz
    ORL     AUXR, #10H                  ;定时器 2 开始计时
    MOV     INT_CLKO, #04H              ;使能定时器 2 的时钟输出功能
    SJMP    $                           ;程序终止

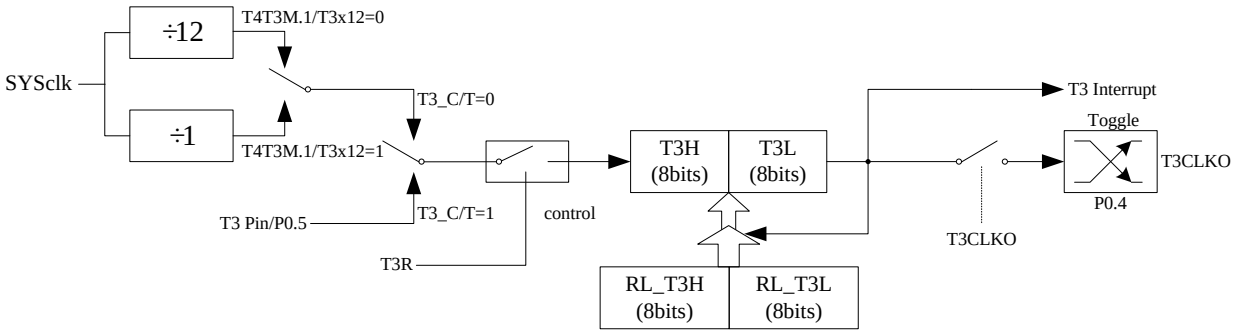
;---------------------------------------------
END
```

18.8.6 定时器 3 对系统时钟或外部引脚 T3 的时钟输入进行可编程分频输出

----及测试程序（C 和汇编）

T3 可以当定时器用，也可以当串口 3 的波特率发生器和可编程时钟输出。

定时器 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

如何利用 T3CLKO/P0.4 管脚输出时钟

T4T3M.0-T3CLKO：是否允许将 P0.4 脚配置为定时器 3（T3）的时钟输出 T3CLKO

1，允许将 P0.4 脚配置为定时器 3（T3）的时钟输出 T3CLKO

0，不允许将 P0.4 脚配置为定时器 3（T3）的时钟输出 T3CLKO

当 T3CLKO/T4T3M.0=1 时，P0.4 管脚配置为定时器 3 的时钟输出 T3CLKO。

输出时钟频率=T3 溢出率/2

- 如果 T3_C/T=0，定时器/计数器 T3 对内部系统时钟计数，则：
  - ✓ T3 工作在 1T 模式（T4T3M.1/T3x12=1）时的输出时钟频率=（SYSclk）/（65536-[RL_TH3, RL_TL3]）/2

- ✓ T3 工作在 12T 模式（T4T3M.1/T3x12=0）时的输出时钟频率=（SYSclk）/12/（65536-[RL_TH3, RL_TL3]）/2

➢ 如果 T3_C/T=1，定时器/计数器 T3 是对外部脉冲输入（P0.5/T3）计数，则：

- ✓ 输出时钟频率=（T3_Pin_CLK）/（65536-[RL_TH3, RL_TL3]）/2

RL_TH3 为 T3H 的重装载寄存器，RL_TL3 为 T3L 的重装载寄存器。

用户在程序中如何具体设置 T3CLKO/P0.4 管脚输出时钟

1.对定时器 3 寄存器 T3H/T3L 送 16 位重装载值，[T3H, T3L] = #reload data

2.对 T4T3M 寄存器中的 T3R 位置 1，让定时器 3 运行

3.对 T4T3M 寄存器中的 T3CLKO 位置 1，让定时器 3 的溢出在 P0.4 口输出时钟。

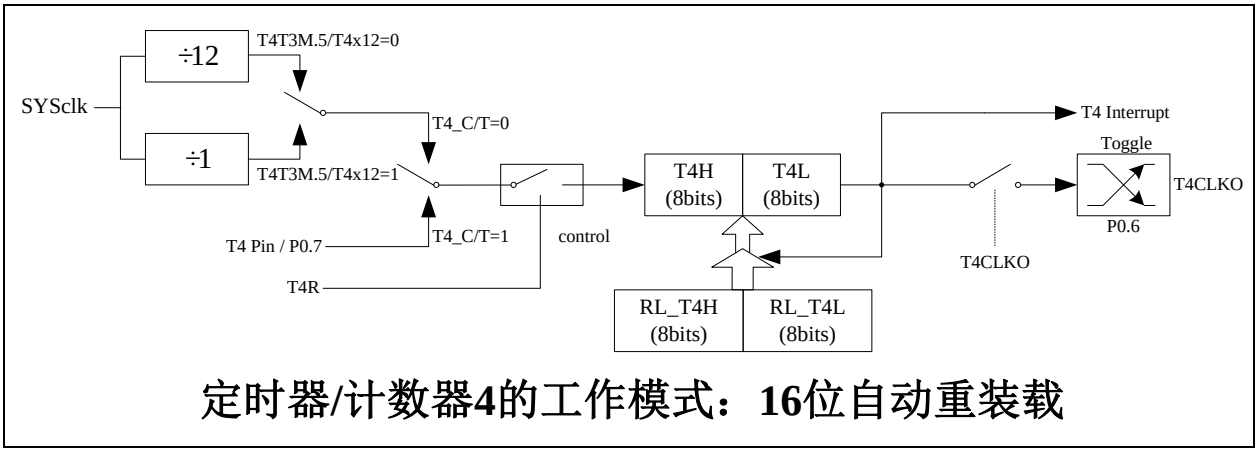
注意：当定时器/计数器 3 用作可编程时钟输出时，不要允许相应的定时器中断，免得 CPU 反复进中断，在特殊情况下也可允许定时器/计数器 3 中断。

18.8.7 定时器 4 对系统时钟或外部引脚 T4 的时钟输入进行可编程分频输出

----及测试程序（C 和汇编）

T4 可以当定时器用，也可以当串口 4 的波特率发生器和可编程时钟输出。

定时器 4 的原理框图如下：



如何利用 T4CLKO/P0.6 管脚输出时钟

T4T3M.4-T4CLKO：是否允许将 P0.6 脚配置为定时器 4（T4）的时钟输出 T4CLKO

1，允许将 P0.6 脚配置为定时器 4（T4）的时钟输出 T4CLKO

0，不允许将 P0.6 脚配置为定时器 4（T4）的时钟输出 T4CLKO

当 T4CLKO/T4T3M.4=1 时，P0.6 管脚配置为定时器 4 的时钟输出 T4CLKO。

输出时钟频率=T4 溢出率/2

➢ 如果 T4_C/T=0，定时器/计数器 T4 对内部系统时钟计数，则：

- ✓ T4 工作在 1T 模式（T4T3M.5/T4x12=1）时的输出时钟频率=（SYSclk）/（65536-[RL_TH4, RL_TL4]）/2
- ✓ T4 工作在 12T 模式（T4T3M.5/T4x12=0）时的输出时钟频率=（SYSclk）/12/（65536-[RL_TH4, RL_TL4]）/2

➢ 如果 T4_C/T=1，定时器/计数器 T4 是对外部脉冲输入（P0.7/T4）计数，则：

- ✓ 输出时钟频率=（T4_Pin_CLK）/（65536-[RL_TH4, RL_TL4]）/2

RL_TH4 为 T4H 的重装载寄存器，RL_TL4 为 T4L 的重装载寄存器。

用户在程序中如何具体设置 T4CLKO/P0.6 管脚输出时钟

1.对定时器 4 寄存器 T4H/T4L 送 16 位重装载值，[T4H, T4L] = #reload data
2.对 T4T3M 寄存器中的 T4R 位置 1，让定时器 4 运行
3.对 T4T3M 寄存器中的 T4CLKO 位置 1，让定时器 4 的溢出在 P0.6 口输出时钟。

注意：当定时器/计数器 4 用作可编程时钟输出时，不要允许相应的定时器中断，免得 CPU 反复进中断，在特殊情况下也可允许定时器/计数器 4 中断。

18.9 掉电唤醒专用定时器及测试程序（C 和汇编）

----进入掉电模式后可将单片机唤醒

----以 15L 开头的单片机进入掉电模式前必须启动掉电唤醒定时器

【特别声明】：以 15L 开头的芯片如需进入“掉电模式”，进入“掉电模式”前必须启动掉电唤醒定时器<3uA>，不超过 1 秒要唤醒一次，以 15F 和 15W 开头的芯片以及新供货的 STC15L2K60S2 系列 D 版本芯片则不需要。

STC15 系列部分单片机新增了内部掉电唤醒定时器，在进入停机模式/掉电模式后，除了可以通过外部中断源进行唤醒外，还可以在无外部中断源的情况下通过使能内部掉电唤醒定时器定期唤醒 CPU，使其恢复到正常工作状态。

掉电唤醒专用定时器的功耗：3V 器件典型值低于 3uA；5V 器件典型值低于 5uA。

STC15 系列单片机的内部低功耗掉电唤醒专用定时器由特殊功能寄存器 WKTCH 和 WKTCL 进行管理和控制。

WKTCL（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
WKTCL	AAH	name									1111 11110B

WKTCH（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
WKTCH	ABH	name	WKTEN								0111 1111B

内部掉电唤醒定时器是一个 15 位定时器，{WKTCH[6：0], WKTCL[7：0]}构成最长 15 位计数值（32768 个），定时从 0 开始计数。

WKTEN：内部停机唤醒定时器的使能控制位。
WKTEN=1，允许内部停机唤醒定时器
WKTEN=0，禁止内部停机唤醒定时器

STC15 系列有内部低功耗掉电唤醒专用定时器的单片机除增加了特殊功能寄存器 WKTCL 和 WKTCH，还设计了了 2 个隐藏的特殊功能寄存器 WKTCL_CNT 和 WKTCH_CNT 来控制内部掉电唤醒专用定时器。

- WKTCL_CNT 与 WKTCL 共用同一个地址，WKTCH_CNT 与 WKTCH 共用同一个地址，WKTCL_CNT 和 WKTCH_CNT 是隐藏的，对用户不可见。
- WKTCL_CNT 和 WKTCH_CNT 实际上是作计数器使用，而 WKTCL 和 WKTCH 实际上作比较器使用。
- 当用户对 WKTCL 和 WKTCH 写入内容时，该内容只写入寄存器 WKTCL 和 WKTCH 中，而不

会写入 WKTCL_CNT 和 WKTCH_CNT 中。

- 当用户读寄存器 WKTCL 和 WKTCH 中的内容时，实际上读的是寄存器 WKTCL_CNT 和 WKTCH_CNT 中的内容，而不是 WKTCL 和 WKTCH 中的内容。

特殊功能寄存器 WKTCL_CNT 和 WKTCH_CNT 的格式如下所示：

WKTCL_CNT

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
WKTCL_CNT	AAH	name									1111 1111B

WKTCH_CNT

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
WKTCH_CNT	ABH	name	-								x111 1111B

通过软件将 WKTCH 寄存器中的 WKTEN（Power Down Wakeup Timer Enable）位置'1'，使能内部掉电唤醒专用定时器。一旦 MCU 进入 Power Down Mode，内部掉电唤醒专用定时器[WKTCH——CNT, WKTCL_CNT]就从 7FFFH 开始计数，直到计数到与{WKTCH[6：0], WKTCL[7：0]}寄存器所设定的计数值相等后就让系统时钟开始振荡。

- 如果主时钟使用的是内部系统时钟（由用户在 ISP 烧录程序时自行设置），MCU 在等待 64 个时钟后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态，就将时钟供给 CPU 工作。
- 如果主时钟使用的是外部晶体或时钟（由用户在 ISP 烧录程序时自行设置），MCU 在等待 1024 个时钟后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态，才将时钟供给 CPU 工作。

CPU 获得时钟后，程序从上次设置单片机进入掉电模式语句的下一条语句开始往下执行。掉电唤醒之后，WKTCH_CNT 和 WKTCL_CNT 的内容保持不变，因此可以通过读[WKTCH, WKTCL]的内容（实际上是读[WKTCH_CNT, WKTCL_CNT]的内容）读出单片机在停机模式/掉电模式所等待的时间。

这里请注意：用户在设置寄存器{WKTCH[6：0], WKTCL[7：0]}的计数值时，要按照所需要的计数次数，在计数次数的基础上减 1 所得的数值才是{WKTCH, WKTCL}的计数值。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 32768 次，则应对{WKTCH[6：0], WKTCL[7：0]}写入 7FFFH（即 32767）。

内部掉电唤醒定时器有自己的内部时钟，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。内部掉电唤醒定时器的时钟频率约为 32768Hz，当然误差较大。

- 对于 16-pin 及其以上的单片机，用户可以通过读 RAM 区 F8 单元和 F9 单元的内容，来获取内部掉电唤醒专用定时器常温下的时钟频率。
- 对于 8-pin 单片机即 STC15F104W 系列，用户可以通过读 RAM 区 78 单元和 79 单元的内容，来获取内部掉电唤醒专用定时器常温下的时钟频率。

下面以 16-pin 及其以上的单片机为例，介绍如何计算内部掉电唤醒专用定时器的计数时间。

假设我们用[WIRC_H, WIRC_L]来表示从 RAM 区 F8 单元和 F9 单元获取到的内部掉电唤醒专用定时器常温下的时钟频率，则内部掉电唤醒专用定时器计数时间按下式计算：

$$\text{内部掉电唤醒专用定时器计数时间} = \frac{10^6\,\text{us}}{\text{WIRC_H, WIRC_L}} \times 16 \times \text{计数次数}$$

例如：假设读到 RAM 区 F8 单元的内容为 80H，F9 单元的内容为 00H，即内部掉电唤醒定时器的时钟频率[WIRC_H, WIRC_L]为 32768Hz，则内部掉电唤醒专用定时器最短计数时间（即计数一次的时间）为：

$$\frac{10^6 \text{us}}{32768} \times 16 \times 1 \approx 488.28\text{us}$$

内部掉电唤醒专用定时器最长计数时间约为 488.28us×32768 = 16S

设定{WKTCH[6：0], WKTCL[7：0]}寄存器的值等于 9（即计数 10 次），且内部掉电唤醒定时器的时钟频率[WIRC_H, WIRC_L]为 32768Hz，则从系统掉电到启动系统振荡器，所需要等待的时间为 488.28us×10 ≈ 4882.8us

设定{WKTCH[6：0], WKTCL[7：0]}寄存器的值等于 32767（即最大计数值=32768=2^{15}），且内部掉电唤醒定时器的时钟频率[WIRC_H, WIRC_L]为 32768Hz，则从系统掉电到启动系统振荡器，所需要等待的时间为 488.28us×32768 = 16s

下面给出了在读到 RAM 区 F8 单元的内容为 80H，F9 单元的内容为 00H，即内部掉电唤醒定时器的时钟频率[WIRC_H, WIRC_L]为 32768Hz 情况下，内部掉电唤醒专用定时器的计数时间：

{WKTCH[6：0], WKTCL[7：0]} = 0， 488.28us×1 = 488.28us
{WKTCH[6：0], WKTCL[7：0]} = 9， 488.28us×10 = 4.8828ms
{WKTCH[6：0], WKTCL[7：0]} = 99， 488.28us×100 = 48.828ms
{WKTCH[6：0], WKTCL[7：0]} = 999， 488.28us×1000 = 488.28ms
{WKTCH[6：0], WKTCL[7：0]} = 4095， 488.28us×4096 = 2.0s
{WKTCH[6：0], WKTCL[7：0]} = 32767， 488.28us×32768 = 16s

再假设读到 RAM 区 F8 单元的内容为 79H，F9 单元的内容为 18H，即内部掉电唤醒定时器的时钟频率[WIRC_H, WIRC_L]为 31000Hz，则内部掉电唤醒专用定时器最短计数时间（即计数一次的时间）为：

$$\frac{10^6 \text{us}}{31000} \times 16 \times 1 \approx 516.13\text{us}$$

内部掉电唤醒专用定时器最长计数时间约为 516.13us×32768 ≈ 16.9s

设定{WKTCH[6：0], WKTCL[7：0]}寄存器的值等于 9（即计数 10 次），且内部掉电唤醒定时器的时钟频率[WIRC_H, WIRC_L]为 31000Hz，则从系统掉电到启动系统振荡器，所需要等待的时间为 516.13us×10 = 5161.3us

下面给出了在读到 RAM 区 F8 单元的内容为 79H，F9 单元的内容为 18H，即内部掉电唤醒定时器的时钟频率[WIRC_H, WIRC_L]为 31000Hz 情况下，内部掉电唤醒专用定时器的计数时间：

{WKTCH[6：0],WKTCL[7：0]} = 0， 516.13us×1 = 516.13us
{WKTCH[6：0],WKTCL[7：0]} = 9， 516.13us×10 = 5.1613ms
{WKTCH[6：0],WKTCL[7：0]} = 99， 516.13us×100 = 51.613ms
{WKTCH[6：0],WKTCL[7：0]} = 999， 516.13us×1000 = 516.13ms
{WKTCH[6：0],WKTCL[7：0]} = 4095， 516.13us×4096 ≈ 2.1s
{WKTCH[6：0],WKTCL[7：0]} = 32767， 516.13us×32768 ≈ 16.9s

又假设读到 RAM 区 F8 单元的内容为 80H，F9 单元的内容为 E8H，即内部掉电唤醒定时器的时钟频率[WIRC_H, WIRC_L]为 33000Hz，则内部掉电唤醒专用定时器最短计数时间（即计数一次的时间）

为： $\frac{10^6 us}{33000} \times 16 \times 1 \approx 484.85us$

内部掉电唤醒专用定时器最长计数时间约为 484.85us×32768≈15.89s

设定{WKTCH[6：0], WKTCL[7：0]} 寄存器的值等于 9（即计数 10 次），且内部掉电唤醒定时器的时钟频率[WIRC_H, WIRC_L]为 33000Hz，则从系统掉电到启动系统振荡器，所需要等待的时间为 484.85us×10 = 4848.5us

下面给出了在读到 RAM 区 F8 单元的内容为 80H，F9 单元的内容为 E8H，即内部掉电唤醒定时器的时钟频率[WIRC_H, WIRC_L]为 33000Hz 情况下，内部掉电唤醒专用定时器的计数时间：

{WKTCH[6：0],WKTCL[7：0]} = 0， 484.85us×1 = 484.85 us
{WKTCH[6：0],WKTCL[7：0]} = 9， 484.85us×10 = 4.8485ms
{WKTCH[6：0],WKTCL[7：0]} = 99， 484.85us×100 = 48.485ms
{WKTCH[6：0],WKTCL[7：0]} = 999， 484.85us×1000 = 484.85ms
{WKTCH[6：0],WKTCL[7：0]} = 4095， 484.85us×4096 ≈ 1.986s
{WKTCH[6：0],WKTCL[7：0]} = 32767， 484.85us×32768≈15.89s

如果掉电唤醒定时器被允许（WKTEN=1），同时用户也将外部中断打开了。进入掉电模式后，当外部中断提前将单片机从停机模式唤醒时，可以通过读 WKTCL 和 WKTCH 的内容（实际是读 WKTCL_CNT 和 WKTCH_CNT 中的内容），可以读出单片机在停机模式/掉电模式等待的时间。

为了降低功耗，未制作掉电唤醒定时器的抗误差和抗温漂的电路，因此，掉电唤醒定时器制造误差较大，压漂（电压抖动）较大。

/*利用内部专用掉电唤醒定时器来唤醒掉电模式的示例程序（C 程序）

1. C 程序：

```
/*----演示 STC15F2K60S2 系列掉电唤醒定时器举例---------------------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-----------*/
//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"
#include "intrins.h"
//-----------------------------------------------
sfr     WKTCL = 0xaa;           //掉电唤醒定时器计时低字节
sfr     WKTCH = 0xab;           //掉电唤醒定时器计时高字节
sbit    P10 = P1^0;
//-----------------------------------------------
void main()
{
    WKTCL = 49;                 //设置唤醒周期为 488us*(49+1) = 24.4ms
    WKTCH = 0x80;               //使能掉电唤醒定时器
    while (1)
    {
        PCON = 0x02;            //进入掉电模式
        _nop_();
        _nop_();
```

```
        P10 = !P10;                     //掉电唤醒后,取反测试口
    }
}
```

2. 汇编程序：

```
/*----演示 STC15F2K60S2 系列掉电唤醒定时器举例----------------------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-----------*/
;假定测试芯片的工作频率为 18.432MHz

WKTCL       DATA    0AAH            ;掉电唤醒定时器计时低字节
WKTCH       DATA    0ABH            ;掉电唤醒定时器计时高字节
;-------------------------------------
    ORG     0000H
    LJMP    START                   ;复位入口
;-------------------------------------
    ORG     0100H
START:
    MOV     SP,   #3FH
    MOV     WKTCL, #49              ;设置唤醒周期为 488us*(49+1) = 24.4ms

    MOV     WKTCH, #80H             ;使能掉电唤醒定时器
LOOP:
    MOV     PCON, #02H              ;进入掉电模式
    NOP
    NOP
    CPL     P1.0                    ;掉电唤醒后,取反测试口
    JMP     LOOP
    SJMP    $
;-------------------------------------
END
```

18.10 外部管脚 T0/T1/T2/T3/T4 如何唤醒掉电模式/停机模式

如果定时器（T0/T1/T2/T3/T4）的中断在进入掉电模式/停机模式前已经被允许，即 ET0/ET1/ET2/ET3/ET4 及 EA 在进入掉电模式/停机模式前已经被置为 1，则进入掉电模式/停机模式后定时器仍继续工作，且定时器 T0/T1/T2/T3/T4 的外部管脚（T0/P3.4，T1/P3.5，T2/P3.1，T3/P0.5，T4/P0.7）如发生由高到低的变化可以将 MCU 从掉电模式/停机模式唤醒。

当 MCU 由定时器 T0/T1/T2/T3/T4 的外部管脚由高到低的变化唤醒时，

- 如果主时钟使用的是内部系统时钟（由用户在 ISP 烧录程序时自行设置），MCU 在等待 64 个时钟后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态，就将时钟供给 CPU 工作；
- 如果主时钟使用的是外部晶体或时钟（由用户在 ISP 烧录程序时自行设置），MCU 在等待 1024 个时钟后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态，就将时钟供给 CPU 工作；

CPU 获得时钟后，程序从上次设置单片机进入掉电模式语句的下一条语句开始往下执行，不进入相应定时器的中断程序。

注意：对于 STC15W4K32S4 系列 A 版本单片机，[T3/P0.5, T4/P0.7]在掉电模式时不要作掉电唤醒。

19 串行口通信

除 STC15F104W 系列无串行口功能外，其他 STC15 系列单片机都有串行口功能，其中 STC15W4K32S4 系列单片机有 4 个高速异步串行通信端口、STC15F2K60S2 系列单片机有 2 个高速异步串行通信端口、STC15W1K16S/STC15W408S/STC15W408AS/STC15W204S/STC15F408AD 系列单片机有 1 个高速异步串行通信端口，如下表所示：

下表总结了 STC15 系列单片机内置了高速异步串行通信端口的单片机型号：

高速异步串行通信端口 / 单片机型号	串行口 1	串行口 2	串行口 3	串行口 4
STC15W4K32S4 系列	√	√	√	√
STC15F2K60S2 系列	√	√		
STC15W1K16S 系列	√			
STC15W404S 系列	√			
STC15W408AS 系列	√			
STC15W204S 系列	√			
STC15F408AD 系列	√			
STC15F104W 系列				

上表中√表示对应的系列有相应的串行口。

现以 STC15W4K32S4 系列单片机为例，介绍 STC15 系列单片机的串行通信端口。

STC15W4K32S4 系列单片机具有 4 个采用 UART（Universal Asychronous Receiver/Transmitter）工作方式的全双工异步串行通信接口（串口 1、串口 2、串口 3 和串口 4）。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。发送缓冲器只能写入而不能读出，接收缓冲器只能读出而不能写入，因而两个缓冲器可以共用一个地址码。

- 串行口 1 的两个缓冲器共用的地址码是 99H；
- 串行口 2 的两个缓冲器共用的地址码是 9BH；
- 串行口 3 的两个缓冲器共用的地址码是 ADH；
- 串行口 4 的两个缓冲器共用的地址码是 85H。

- 串行口 1 的两个缓冲器统称串行通信特殊功能寄存器 SBUF；
- 串行口 2 的两个缓冲器统称串行通信特殊功能寄存器 S2BUF；
- 串行口 3 的两个缓冲器统称串行通信特殊功能寄存器 S3BUF；
- 串行口 4 的两个缓冲器统称串行通信特殊功能寄存器 S4BUF。

STC15W4K32S4 系列单片机的串行口 1 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。串行口 2/串行口 3/串行口 4 都只有两种工作方式，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。

STC15W4K32S4 系列单片机串行口 1 对应的硬件部分是 TxD 和 RxD。串行口 1 可以在 3 组管脚之间进行切换。

- 通过设置特殊功能寄存器 AUXR1/P_SW1 中的位 S1_S1/AUXR1.7 和 S1_S0/PSW1.6，
- 可以将串行口 1 从[RxD/P3.0, TxD/P3.1]切换到[RxD_2/P3.6, TxD_2/P3.7]，

- 还可以切换到[RxD_3/P1.6/XTAL2, TxD_3/P1.7/XTAL1]。

注意，当串行口 1 在[RxD_2/P1.6, TxD_2/P1.7]时，系统要使用内部时钟。串口 1 建议放在[P3.6/RxD_2, P3.7/TxD_2]或[P1.6/RxD_3/XTAL2, P1.7/TxD_3/XTAL1]上。

STC15W4K32S4 系列单片机串行口 2 对应的硬件部分是 TxD2 和 RxD2。串行口 2 可以在 2 组管脚之间进行切换。

- 通过设置特殊功能寄存器 P_SW2 中的位 S2_S/P_SW2.0，
- 可以将串行口 2 从[RxD2/P1.0, TxD2/P1.1]切换到[RxD2_2/P4.6, TxD2_2/P4.7]。

STC15W4K32S4 系列单片机串行口 3 对应的硬件部分是 TxD3 和 RxD3。串行口 3 可以在 2 组管脚之间进行切换。

- 通过设置特殊功能寄存器 P_SW2 中的位 S3_S/P_SW2.1，
- 可以将串行口 3 从[RxD3/P0.0, TxD3/P0.11 切换到[RxD3_2/P5.0, TxD3_2/P5.1]。

STC15W4K32S4 系列单片机串行口 4 对应的硬件部分是 TxD4 和 RxD4。串行口 4 可以在 2 组管脚之间进行切换。

- 通过设置特殊功能寄存器 P_SW2 中的位 S4_S/P_SW2.2，
- 可以将串行口 4 从[RxD4/P0.2, TxD4/P0.3]切换到[RxD4_2/P5.2, TxD4_2/P5.3]。

STC15W4K32S4 系列单片机的串行通信口，除用于数据通信外，还可方便地构成一个或多个并行 I/O 口，或作串----并转换，或用于扩展串行外设等。

19.1 串行口 1 的相关寄存器

符号	描述	地址	位地址及其符号								复位值
			MSB							LSB	
T2H	定时器 2 高 8 位寄存器	D6H									0000 0000B
T2L	定时器 2 低 8 位寄存器	D7H									0000 0000B
AUXR	辅助寄存器	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000 0001B
SCON	Serial Control	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000 0000B
SBUF	Serial Buffer	99H									xxxx xxxxB
PCON	Power Control	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011 0000B
IE	Interrupt Enable	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000 0000B
IP	Interrupt Priority Low	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	0000 0000B
SADEN	Slave Address Mask	B9H									0000 0000B
SADDR	Slave Address	A9H									0000 0000B
AUXR1 P_SW1	辅助寄存器 1	A2H	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000 0000B
CLK_DIV PCON2	时钟分频寄存器	97H	MCKO_S1	MCKO_S1	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000 0000B

1、串行口 1 的控制寄存器 SCON 和 PCON

STC15 系列单片机的串行口 1 设有两个控制寄存器：串行控制寄存器 SCON 和波特率选择特殊功能寄存器 PCON。

串行控制寄存器 SCON 用于选择串行通信的工作方式和某些控制功能。其格式如下：

SCON：串行控制寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
SCON	98H	name	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI

SM0/FE：当 PCON 寄存器中的 SMOD0/PCON.6 位为 1 时，该位用于帧错误检测。当检测到一个无效停止位时，通过 UART 接收器设置该位。它必须由软件清零。

当 PCON 寄存器中的 SMOD0/PCON.6 位为 0 时，该位和 SM1 一起指定串行通信的工作方式，如下表所示。

其中 SM0、SM1 按下列组合确定串行口 1 的工作方式：

SM0	SM1	规则方式	功能说明	波特率
0	0	方式 0	同步移位串行方式：移位寄存器	当 UART_M0x6 = 0 时，波特率是 SYSclk/12, 当 UART_M0x6 = 1 时，波特率是 SYSclk / 2
0	1	方式 1	8 位 UART， 波特率可变	串行口 1 用定时器 1 作为其波特率发生器且定时器 1 工作于模式 0（16 位自动重装载模式）或串行口用定时器 2 作为其波特率发生器时， 波特率 =（定时器 1 的溢出率或定时器 T2 的溢出率）/4 注意：此时波特率与 SMOD 无关。 当串行口 1 用定时器 1 作为其波特率发生器且定时器 1 工作于模式 2（8 位自动重装模式）时，波特率=（2^{SMOD}/32）×（定时器 1 的溢出率）
1	0	方式 2	9 位 UART	（2^{SMOD}/64）×SYSclk 系统工作时钟频率
1	1	方式 3	9 位 UART， 波特率可变	当串行口 1 用定时器 1 作为其波特率发生器且定时器 1 工作于模式 0（16 位自动重装载模式）或串行口用定时器 2 作为其波特率发生器时， 波特率=（定时器 1 的溢出率或定时器 T2 的溢出率）/4 注意：此时波特率与 SMOD 无关。 当串行口 1 用定时器 1 作为其波特率发生器且定时器 1 工作于模式 2（8 位自动重装模式）时，波特率=（2^{SMOD}/32）×（定时器 1 的溢出率）

- 当定时器 1 工作于模式 0（16 位自动重装载模式）且 AUXR.6/T1x12=0 时，
  定时器 1 的溢出率=SYSclk / 12 /（65536 - [RL_TH1, RL_TL1]）；
- 当定时器 1 工作于模式 0（16 位自动重装载模式）且 AUXR.6/T1x12=1 时，
  定时器 1 的溢出率=SYSclk /（65536 - [RL_TH1, RL_TL1]）
- 当定时器 1 工作于模式 2（8 位自动重装模式）且 T1x12=0 时，
  定时器 1 的溢出率=SYSclk / 12 /（256-TH1）；
- 当定时器 1 工作于模式 2（8 位自动重装模式）且 T1x12=1 时
  定时器 1 的溢出率=SYSclk /（256-TH1）
- 当 AUXR.2/T2x12=0 时，定时器 T2 的溢出率=SYSclk / 12 /（65536 - [RL_TH2, RL_TL2]）
- 当 AUXR.2/T2x12=1 时，定时器 T2 的溢出率=SYSclk /（65536 - [RL_TH2, RL_TL2]）

SM2： 允许方式 2 或方式 3 多机通信控制位。

- 在方式 2 或方式 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。
- 在方式 2 或方式 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。

方式 1 和方式 0 是非多机通信方式，在这两种方式时，要设置 SM2 应为 0。

REN： 允许/禁止串行接收控制位。

- 由软件置位 REN，即 REN=1 为允许串行接收状态，可启动串行接收器 RxD，开始接收信息。
- 软件复位 REN，即 REN=0，则禁止接收。

TB8： 在方式 2 或方式 3，它为要发送的第 9 位数据，按需要由软件置位或清 0。例如，可用作数据的校验位或多机通信中表示地址帧/数据帧的标志位。在方式 0 和方式 1 中，该位不用。

RB8： 在方式 2 或方式 3，是接收到的第 9 位数据，作为奇偶校验位或地址帧/数据帧的标志位。方式 0 中不用 RB8（置 SM2=0）。方式 1 中也不用 RB8（置 SM2=0，RB8 是接收到的停止位）。

TL： 发送中断请求标志位。

- 在方式 0，当串行发送数据第 8 位结束时，由内部硬件自动置位即 TI=1，向主机请求中断，响应中断后 TI 必须用软件清零，即 TI=0。
- 在其他方式中，则在停止位开始发送时由内部硬件置位，即 TI=1，响应中断后 TI 必须用软件清零。

RI： 接收中断请求标志位。

- 在方式 0，当串行接收到第 8 位结束时，由内部硬件自动置位 RI=1，向主机请求中断，响应中断后 RI 必须用软件清零，即 RI=0。
- 在其他方式中，串行接收到停止位的中间时刻由内部硬件置位，即 RI=1，向 CPU 发中断申请，响应中断后 RI 必须由软件清零。

SCON 的所有位可通过整机复位信号复位为全“0”。SCON 的字节地址为 98H，可位寻址，各位地址为 98H ~ 9FH，可用软件实现位设置。

串行通信的中断请求：

- 当一帧发送完成，内部硬件自动置位 TI，即 TI=1，请求中断处理；
- 当接收完一帧信息时，内部硬件自动置位 RI，即 RI=1，请求中断处理。

由于 TI 和 RI 以“或逻辑”关系向主机请求中断，所以主机响应中断时事先并不知道是 TI 还是 RI 请求的中断，必须在中断服务程序中查询 TI 和 RI 进行判别，然后分别处理。因此，两个中断请求标志位均不能由硬件自动置位，必须通过软件清 0，否则将出现一次请求多次响应的错误。

电源控制寄存器 PCON 中的 SMOD/PCON.7 用于设置方式 1、方式 2、方式 3 的波特率是否加倍。

电源控制寄存器 PCON 格式如下：

PCON：电源控制寄存器（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
PCON	87H	name	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL

SMOD：波特率选择位。

- 当用软件置位 SMOD，即 SMOD=1，则使串行通信方式 1、2、3 的波特率加倍;
- SMOD=0，则各工作方式的波特率不加倍。复位时 SMOD=0。

SMOD0：帧错误检测有效控制位。

- 当 SMOD0=1，SCON 寄存器中的 SM0/FE 位用于 FE（帧错误检测）功能；
- 当 SMOD0=0，SCON 寄存器中的 SM0/FE 位用于 SM0 功能，和 SM1 一起指定串行口的工作方式。复位时 SMOD0=0

PCON 中的其他位都与串行口 1 无关，在此不作介绍。

2、串行口数据缓冲寄存器 SBUF

STC15 系列单片机的串行口 1 缓冲寄存器（SBUF）的地址是 99H，实际是 2 个缓冲器，写 SBUF 的操作完成待发送数据的加载，读 SBUF 的操作可获得已接收到的数据。两个操作分别对应两个不同的寄存器，1 个是只写寄存器，1 个是只读寄存器。

串行通道内设有数据寄存器。在所有的串行通信方式中，在写入 SBUF 信号（MOV SBUF, A）的控制下，把数据装入相同的 9 位移位寄存器，前面 8 位为数据字节，其最低位为移位寄存器的输出位。根据不同的工作方式会自动将“1”或 TB8 的值装入移位寄存器的第 9 位，并进行发送。

串行通道的接收寄存器是一个输入移位寄存器。在方式 0 时它的字长为 8 位，其他方式时为 9 位。当一帧接收完毕，移位寄存器中的数据字节装入串行数据缓冲器 SBUF 中，其第 9 位则装入 SCON 寄存器中的 RB8 位。如果由于 SM2 使得已接收到的数据无效时，RB8 和 SBUF 中内容不变。

由于接收通道内设有输入移位寄存器和 SBUF 缓冲器，从而能使一帧接收完将数据由移位寄存器装入 SBUF 后，可立即开始接收下一帧信息，主机应在该帧接收结束前从 SBUF 缓冲器中将数据取走，否则前一帧数据将丢失。

SBUF 以并行方式送往内部数据总线。

3、辅助寄存器 AUXR

辅助寄存器 AUXR 的格式及各位含义如下：

AUXR：辅助寄存器（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	name	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

T0x12： 定时器 0 速度控制位
- 0，定时器 0 是传统 8051 速度，12 分频；
- 1，定时器 0 的速度是传统 8051 的 12 倍，不分频

T1x12： 定时器 1 速度控制位
- 0，定时器 1 是传统 8051 速度，12 分频：
- 1，定时器 1 的速度是传统 8051 的 12 倍，不分频

  如果 UART1/串口 1 用 T1 作为波特率发生器，则由 T1x12 决定 UART1/串口是 12T 还是 1T

UART_M0x6：串口模式 0 的通信速度设置位。
- 0，串口 1 模式 0 的速度是传统 8051 单片机串口的速度，12 分频；
- 1，串口 1 模式 0 的速度是传统 8051 单片机串口速度的 6 倍，2 分频

T2R： 定时器 2 允许控制位
- 0，不允许定时器 2 运行；
- 1，允许定时器 2 运行

T2_C/T：控制定时器 2 用作定时器或计数器
- 0，用作定时器（对内部系统时钟进行计数）；
- 1，用作计数器（对引脚 T2/P3.1 的外部脉冲进行计数）

T2x12： 定时器 2 速度控制位
- 0，定时器 2 是传统 8051 速度，12 分频；
- 1，定时器 2 的速度是传统 8051 的 12 倍，不分频

如果串口 1 或串口 2 用 T2 作为波特率发生器，则由 T2x12 决定串口 1 或串口 2 是 12T 还是 1T

EXTRAM：内部/外部 RAM 存取控制位

- 0，允许使用逻辑上在片外、物理上在片内的扩展 RAM；
- 1，禁止使用逻辑上在片外、物理上在片内的扩展 RAM

S1ST2：串口 1（UART1）选择定时器 2 作波特率发生器的控制位

- 0，选择定时器 1 作为串口 1（UART1）的波特率发生器；
- 1，选择定时器 2 作为串口 1（UART1）的波特率发生器，此时定时器 1 得到释放，可以作为独立定时器使用

串口 1 可以选择定时器 1 做波特率发生器，也可以选择定时器 2 作为波特率发生器。当设置 AUXR 寄存器中的 S1ST2 位（串行口波特率选择位）为 1 时，串行口 1 选择定时器 2 作为波特率发生器，此时定时器 1 可以释放出来作为定时器/计数器/时钟输出使用。

【注意】：对于 STC15 系列单片机，

- 串口 2 只能使用定时器 2 作为波特率发生器，不能够选择其他定时器作为其波特率发生器；
- 而串口 1 默认选择定时器 2 作为其波特率发生器，也可以选择定时器 1 作为其波特率发生器；
- 串口 3 默认选择定时器 2 作为其波特率发生器，也可以选择定时器 3 作为其波特率发生器；
- 串口 4 默认选择定时器 2 作为其波特率发生器，也可以选择定时器 4 作为其波特率发生器。

4、定时器 2 的寄存器 T2H, T2L

定时器 2 寄存器 T2H（地址为 D6H，复位值为 00H）及寄存器 T2L（地址为 D7H，复位值为 00H），用于保存重装时间常数。

【注意】：对于 STC15 系列单片机：

- 串口 2 只能使用定时器 2 作为波特率发生器，不能够选择其他定时器作为其波特率发生器；
- 串口 1 默认选择定时器 2 作为其波特率发生器，也可以选择定时器 1 作为其波特率发生器；
- 串口 3 默认选择定时器 2 作为其波特率发生器，也可以选择定时器 3 作为其波特率发生器；
- 串口 4 默认选择定时器 2 作为其波特率发生器，也可以选择定时器 4 作为其波特率发生器。

5、从机地址控制寄存器 SADEN 和 SADDR

为了方便多机通信，STC15 系列单片机设置了从机地址控制寄存器 SADEN 和 SADDR。其中：

- SADEN 是从机地址掩模寄存器（地址为 B9H，复位值为 00H）；
- SADDR 是从机地址寄存器（地址为 A9H，复位值为 00H）。

6、与串行口 1 中断相关的寄存器位 ES 和 PS

串行口中断允许位 ES 位于中断允许寄存器 IE 中，中断允许寄存器的格式如下：

IE：中断允许寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	name	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

EA： CPU 的总中断允许控制位

- EA=1，CPU 开放中断，
- EA=0，CPU 屏蔽所有的中断申请。

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。

ES： 串行口中断允许位

- ES=1，允许串行口中断，
- ES=0，禁止串行口中断。

IP：中断优先级控制寄存器低（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IP	B8H	name	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0

PS： 串行口 1 中断优先级控制位

- 当 PS=0 时，串行口 1 中断为最低优先级中断（优先级 0）
- 当 PS=1 时，串行口 1 中断为最高优先级中断（优先级 1）

7、将串口 1 进行切换的寄存器 AUXR1（P_SW1）

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000 0000

串口 1/S1 可在 3 个地方切换，由 S1_S0 及 S1_S1 控制位来选择		
S1_S1	S1_S0	串口 1/S1 可在 P1/P3 之间来回切换
0	0	串口 1/S1 在[P3.0/RxD，P3.1/TxD]
0	1	串口 1/S1 在[P3.6/RxD_2，P3.7/TxD_2]
1	0	串口 1/S1 在[P1.6/RxD_3/XTAL2，P1.7/TxD_3/XTAL1] 串口 1 在 P1 口时要使用内部时钟
1	1	无效

串口 1 建议放在[P3.6/RxD_2, P3.7/TxD_2]或[P1.6/RxD_3/XTAL2, P1.7/TxD_3/XTAL1]上。

CCP 可在 3 个地方切换，由 CCP_S1 / CCP_S0 两个控制位来选择		
CCP_S1	CCP_S0	CCP 可在 P1/P2/P3 之间来回切换
0	0	CCP 在[P1.2/ECI，P1.1/CCP0，P1.0/CCP1，P3.7/CCP2]
0	1	CCP 在[P3.4/ECI_2，P3.5/CCP0_2，P3.6/CCP1_2，P3.7/CCP2_2]
1	0	CCP 在[P2.4/ECI_3，P2.5/CCP0_3，P2.6/CCP1_3，P2.7/CCP2_3]
1	1	无效

SPI 可在 3 个地方切换，由 SPI_S1 / SPI_S0 两个控制位来选择		
SPI_S1	SPI_S0	SPI 可在 P1/P2/P4 之间来回切换
0	0	SPI 在[P1.2/SS，P1.3/MOSI，P1.4/MISO，P1.5/SCLK]
0	1	SPI 在[P2.4/SS_2，P2.3/MOSI_2，P2.2/MISO_2，P2.1/SCLK_2]
1	0	SPI 在[P5.4/SS_3，P4.0/MOSI_3，P4.1/MISO_3，P4.3/SCLK_3]
1	1	无效

8、串口 1 的中继广播方式设置位----Tx_Rx/CLK_DIV.4

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
CLK_DIV （PCON2）	97H	时中分频 寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000,0000

Tx_Rx：串口 1 的中继广播方式设置

- 0，串口 1 为正常工作方式
- 1，串口 1 为中继广播方式，即将 RxD 端口输入的电平状态实时输出在 TxD 外部管脚上，TxD

外部管脚可以对 RxD 管脚的输入信号进行实时整形放大输出，TxD 管脚的对外输出实时反映 RxD 端口输入的电平状态。

串口 1 的 RxD 管脚和 TxD 管脚可以在 3 组不同管脚之间进行切换：[RxD/P3.0, TxD/P3.1]；
[RxD_2/P3.6, TxD_2/P3.7]；
[RxD_3/P1.6, TxD_3/P1.7]

19.2 串行口 1 工作模式

STC15 系列单片机的串行通信接口有 4 种工作模式，可通过软件编程对 SCON 中的 SM0、SM1 的设置进行选择。其中：

- 模式 1、模式 2 和模式 3 为异步通信，每个发送和接收的字符都带有 1 个启动位和 1 个停止位。
- 在模式 0 中，串行口被作为 1 个简单的移位寄存器使用。

19.2.1 串行口 1 工作模式 0：同步移位寄存器（建议初学者不学）

在模式 0 状态，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/AUXR.5=0 时，其波特率固定为 SYSclk/12。
- 当串行口模式 0 的通信速度设置位 UART_M0x6/AUXR.5=1 时，其波特率固定为 SYSclk/2。

串行口数据由 RxD/P3.0 端输入，同步移位脉冲（SHIFTCLOCK）由 TxD/P3.1 输出，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 / AUXR.5 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（SHIFTCLOCK）。波形如图 8-1 中“发送”所示。

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。

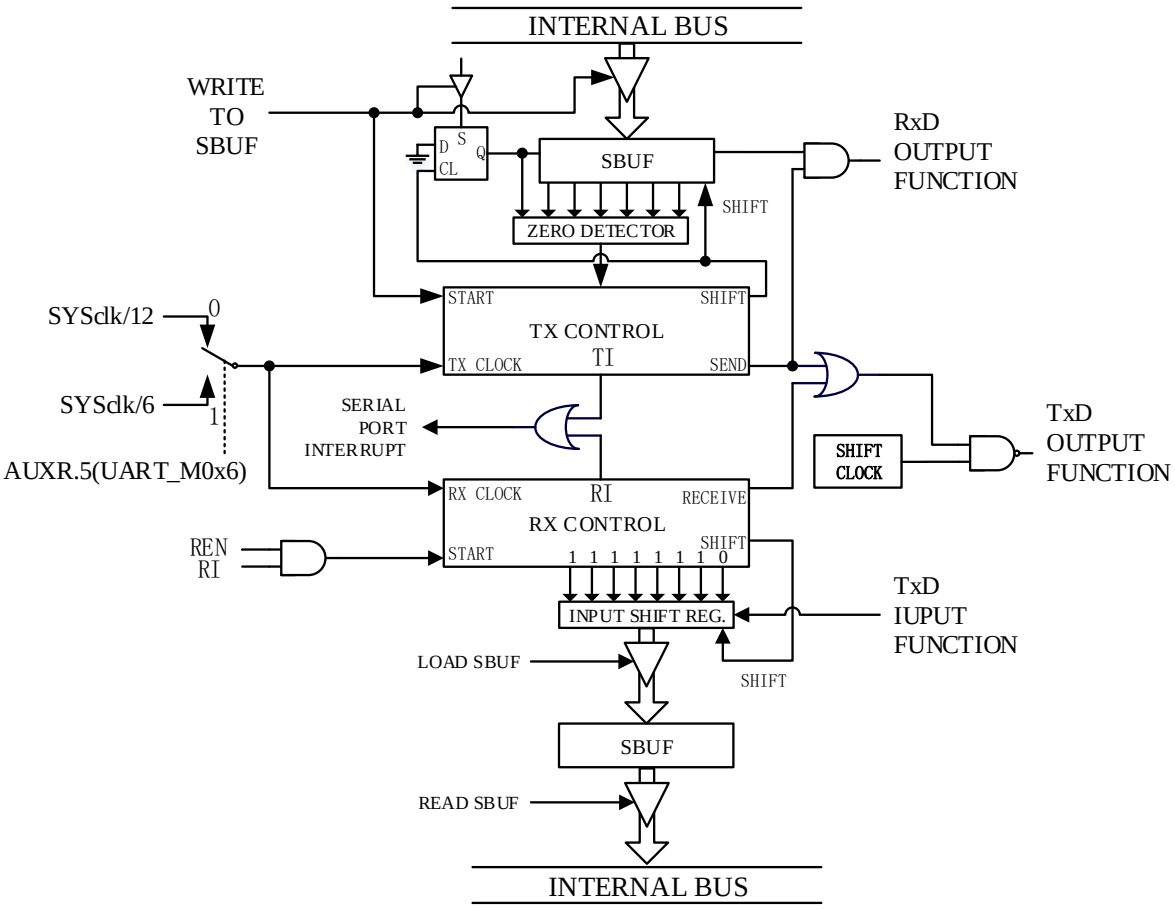
模式 0 接收过程：模式 0 接收时，复位接收中断请求标志 RI，即 RI=0，置位允许接收控制位 REN=1 时启动串行模式 0 接收过程。启动接收过程后，RxD 为串行输入端，TxD 为同步脉冲输出端。串行接收的波特率为 SYSclk/12 或 SYSclk/2（由 UART_M0x6/AUXR.5 确定是 12 分频还是 2 分频）。其时序图如下图中“接收”所示。

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。

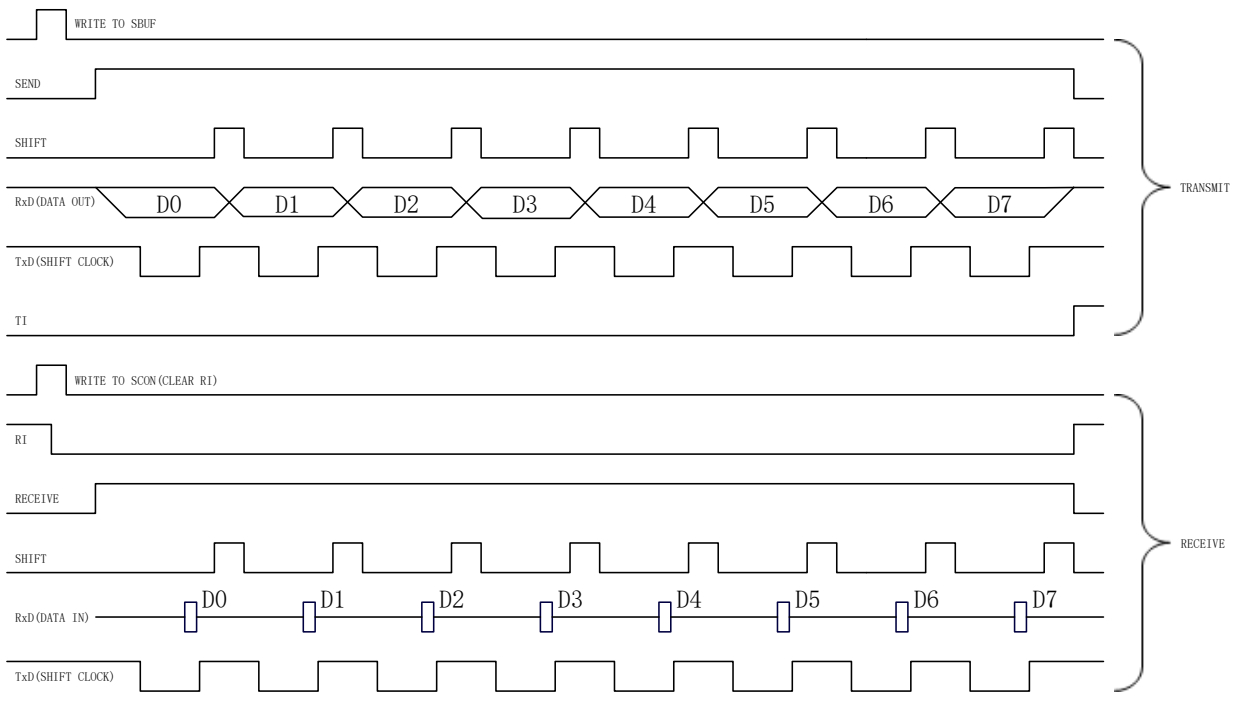
工作于模式 0 时，必须清 0 多机通信控制位 SM2，使不影响 TB8 位和 RB8 位。由于波特率固定为 SYSclk/12 或 SYSclk/2，无需定时器提供，直接由单片机的时钟作为同步移位脉冲。

串行口工作模式 0 的示意图如下图所示。

由示意图中可见，由 TX 和 RX 控制单元分别产生中断请求信号并置位 TI=1 或 RI=1，经“或门”送主机请求中断，所以主机响应中断后必须软件判别是 TI 还是 RI 请求中断，必须软件清 0 中断请求标志位 TI 或 RI。



串行口1模式0 功能结构示意图



串行口1模式0 时序示意图

19.2.2 串行口 1 工作模式 1：8 位 UART，波特率可变

当软件设置 SCON 的 SM0、SM1 为“01”时，串行口 1 则以模式 1 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。TxD/P3.1 为发送信息，RxD/P3.0 为接收端接收信息，串行口为全双工接受/发送串行口。

下图为串行模式 1 的功能结构示意图及接收/发送时序图

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。发送各位的定时是由 16 分频计数器同步。

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。

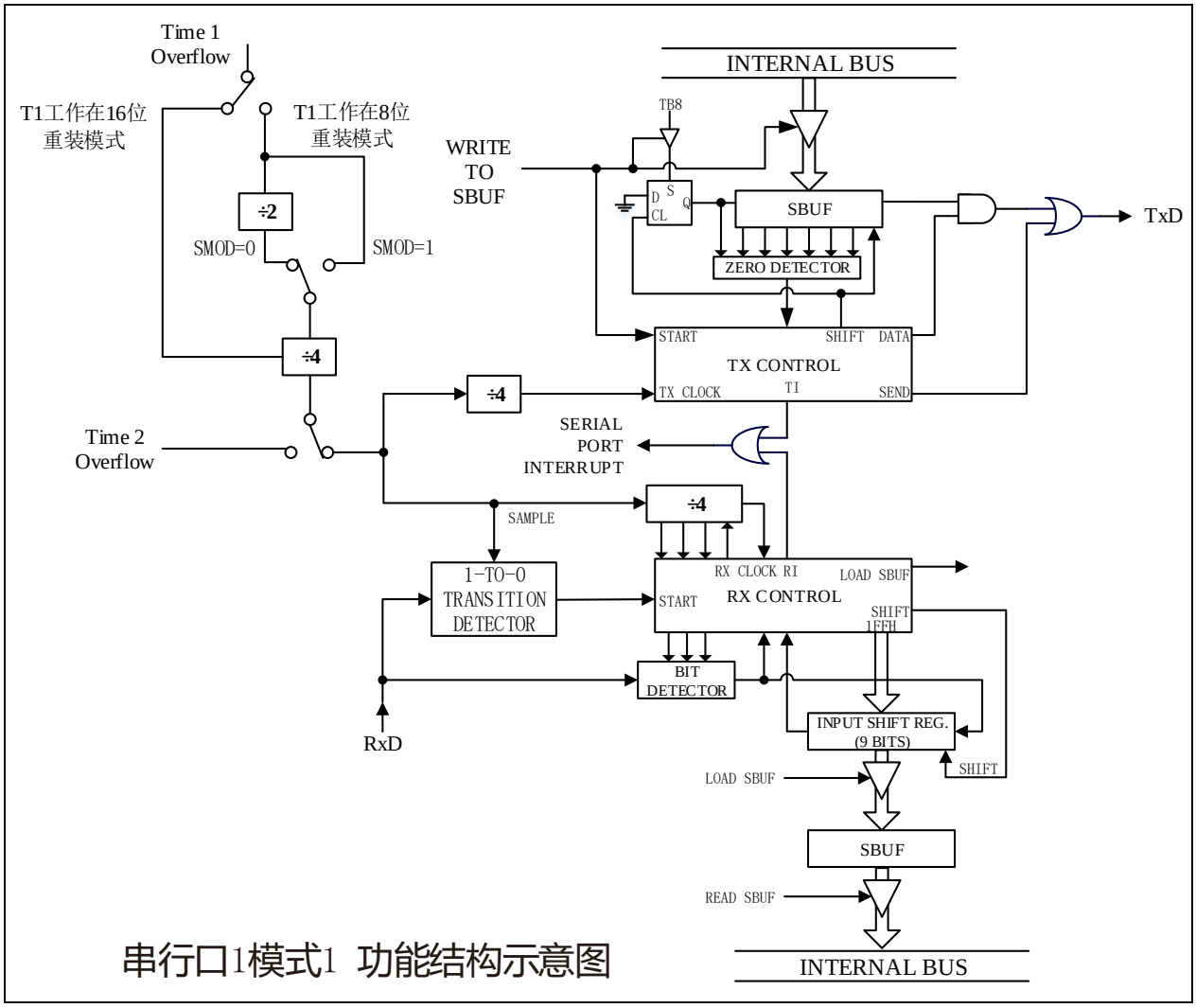
模式 1 的接收过程：当软件置位接收允许标志位 REN，即 REN=1 时，接收器便以选定波特率的 16 分频的速率采样串行接收端口 RxD，当检测到 RxD 端口从“1”→“0”的负跳变时，就启动接收器准备接收数据，并立即复位 16 分频计数器，将 1FFH 植装入移位寄存器。复位 16 分频计数器是使它与输入位时间同步。

16 分频计数器的 16 个状态是将 1 波特率（每位接收时间）均为 16 等份，在每位时间的 7、8、9 状态由检测器对 RxD 端口进行采样，所接收的值是这次采样直经“三中取二”的值，即 3 次采样至少 2 次相同的值，以此消除干扰影响，提高可靠性。在起始位，如果接收到的值不为“0”（低电平），则起始位无效，复位接收电路，并重新检测“1”→“0”的跳变。如果接收到的起始位有效，则将它输入移位寄存器，并接收本帧的其余信息。

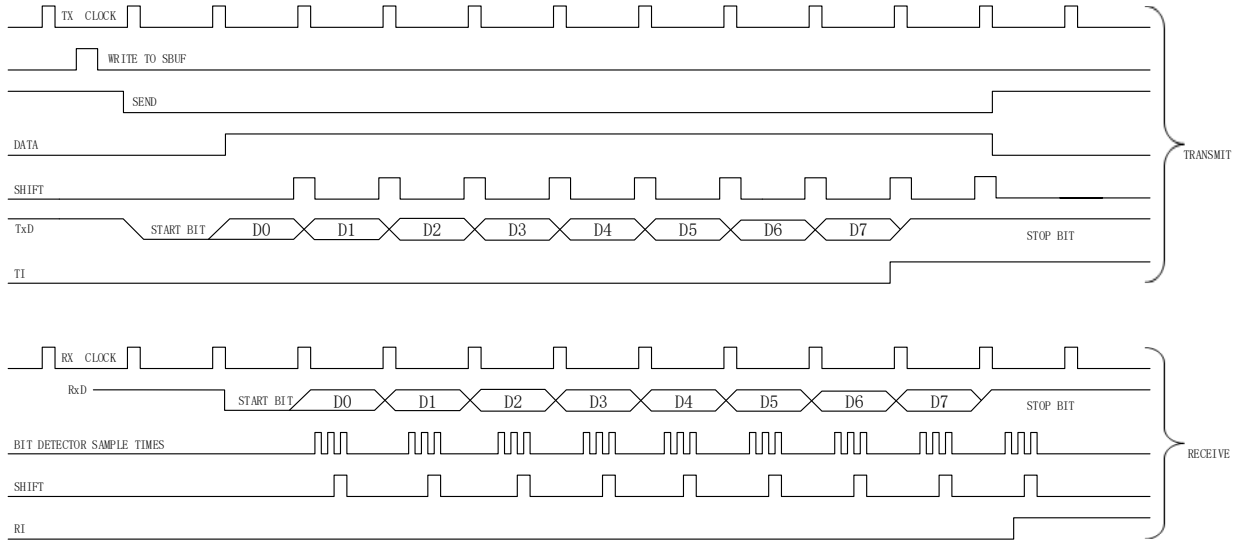
接收的数据从接收移位寄存器的右边移入，已装入的 1FFH 向左边移出，当起始位"0"移到移位寄存器的最左边时，使 RX 控制器作最后一次移位，完成一帧的接收。若同时满足以下两个条件：

- ✓ 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，置位 RI，即 RI=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，必须由软件清 0，即 RI=0。通常情况下，串行通信工作于模式 1 时，SM2 设置为"0”。



串行口1模式1 功能结构示意图



串行口1模式1 接收/发送时序图

串行通信模式 1 的波特率是可变的，可变的波特率由定时器/计数器 1 或定时器 2 产生，优先选择定时器 2 产生波特率。

- 当串行口 1 用定时器 2 作为其波特率发生器时，
  串行口 1 的波特率=（定时器 T2 的溢出率）/4

（注意：此时波特率也与 SMOD 无关。）

✓ 当 T2 工作在 1T 模式（AUXR.2/T2x12=1）时，
- 定时器 2 的溢出率=SYSclk/（65536-[RL_TH2, RL_TL2]）；
- 即此时，串行口 1 的波特率=SYSclk/（65536-[RL_TH2, RL_TL2]）/4

✓ 当 T2 工作在 12T 模式（AUXR.2/T2x12=0）时，
- 定时器 2 的溢出率=SYSclk/12/（65536-[RL_TH2, RL_TL2]）；
- 即此时，串行口 1 的波特率=SYSclk/12/（65536-[RL_TH2, RL_TL2]）/4

RL_TH2 是 T2H 的自动重装载寄存器，RL_TL2 是 T2L 的自动重装载寄存器。

➢ 当串行口 1 用定时器 1 作为其波特率发生器且定时器 1 工作于模式 0（16 位自动重装载模式）时，串行口 1 的波特率=（定时器 1 的溢出率）/4

（注意：此时波特率与 SMOD 无关。）

✓ 当定时器 1 工作于模式 0（16 位自动重装载模式）且 T1x12=0 时，
- 定时器 1 的溢出率=SYSclk/12/（65536-[RL_TH1, RL_TL1]）；
- 即此时，串行口 1 的波特率=SYSclk/12/（65536-[RL_TH1, RL_TL1]）/4

✓ 当定时器 1 工作于模式 0（16 位自动重装载模式）且 T1x12=1 时，
- 定时器 1 的溢出率=SYSclk/（65536-[RL_TH1, RL_TL1]）
- 即此时，串行口 1 的波特率=SYSclk/（65536-[RL_TH1, RL_TL1]）/4

RL_TH1 是 TH1 的自动重装载寄存器，RL_TL1 是 TL1 的自动重装载寄存器。

➢ 当串行口 1 用定时器 1 作为其波特率发生器且定时器 1 工作于模式 2（8 位自动重装模式）时，串行口 1 的波特率=（2^{SMOD}/32）×（定时器 1 的溢出率）

✓ 当定时器 1 工作于模式 2（8 位自动重装模式）且 T1x12=0 时，
- 定时器 1 的溢出率=SYSclk/12/（256-TH1）；
- 即此时，串行口 1 的波特率=（2^{SMOD} / 32）×SYSclk / 12 /（256-TH1）

✓ 当定时器 1 工作于模式 2（8 位自动重装模式）且 T1x12=1 时，
- 定时器 1 的溢出率=SYSclk/（256-TH1）
- 即此时，串行口 1 的波特率=（2^{SMOD} / 32）×SYSclk /（256-TH1）

19.2.3 串行口 1 工作模式 2：9 位 UART，波特率固定（建议不学习）

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/P3.1 为发送端口 RxD/P3.0 为接收端口，以全双工模式进行接收/发送。

模式 2 的波特率为：

串行通信模式 2 波特率=2^{SMOD} / 64×（SYSclk 系统工作时钟频率）

上述波特率可通过软件对 PCON 中的 SMOD 位进行设置，当 SMOD=1 时，选择 SYSclk/32；当 SMOD=0 时，选择 SYSclk/64，故而称 SMOD 为波特率加倍位。可见，模式 2 的波特率基本上是固定的。

下图为串行通信模式 2 的功能结构示意图及其接收/发送时序图。

由下图可知，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。

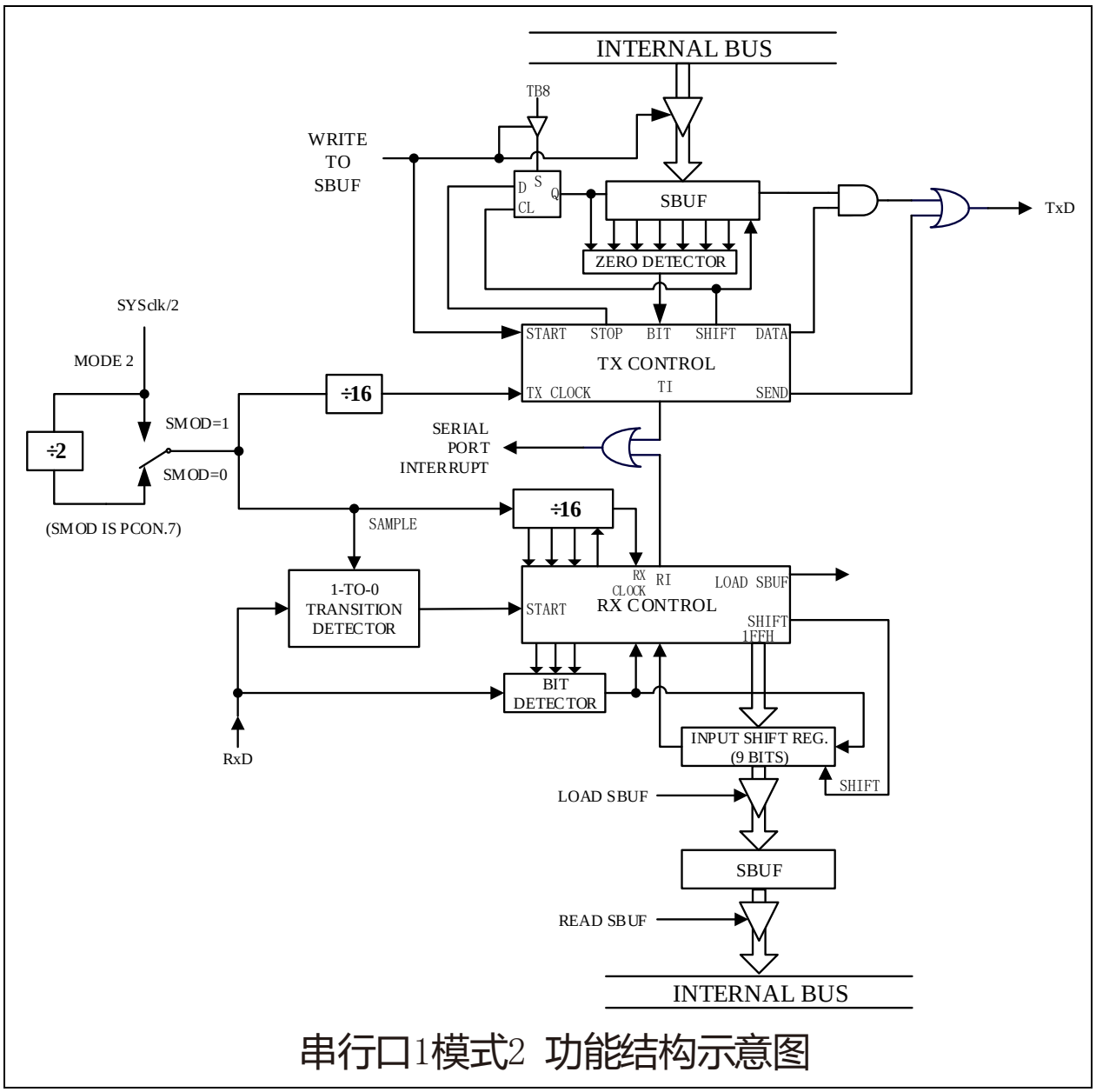
当接收器接收完一帧信息后必须同时满足下列条件：

- ✓ RI=0
- ✓ SM2=0 或者 SM2=1，并且接收到的第 9 数据位 RB8=1。

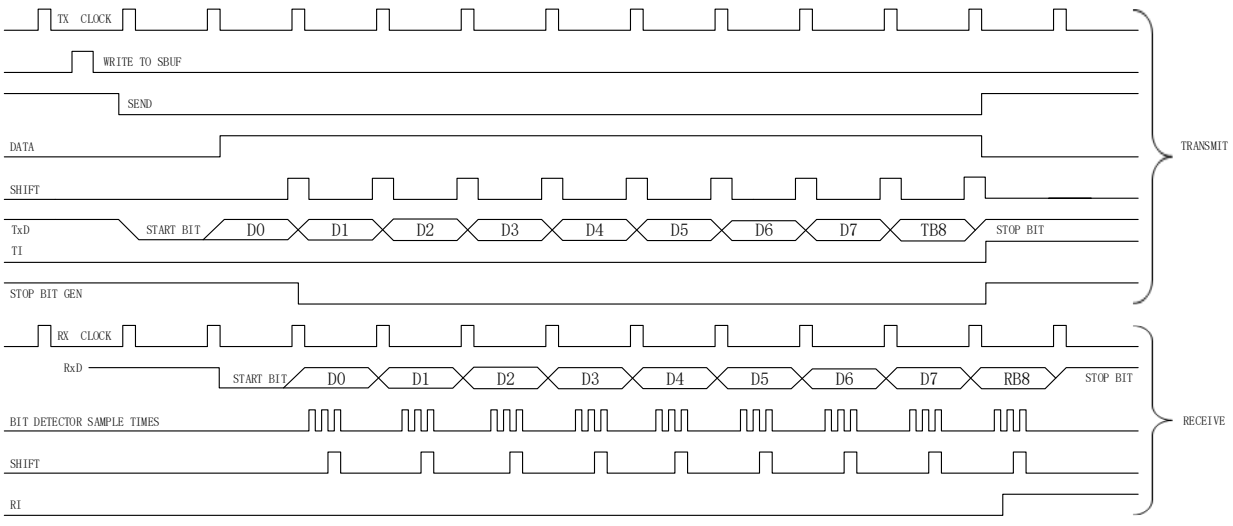
当上述两条件同时满足时，才将接收到的移位寄存器的数据装入 SBUF 和 RB8 中，并置位 RI=1，向主机请求中断处理。如果上述条件有一个不满足，则刚接收到移位寄存器中的数据无效而丢失，也不置位 RI。无论上述条件满足与否，接收器又重新开始检测 RxD 输入端口的跳变信息，接收下一帧的输入信息。

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。



串行口1模式2 功能结构示意图



串行口1模式2 接收/发送时序图

19.2.4 串行口 1 工作模式 3：9 位 UART，波特率可变

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/P3.1 为发送端口，RxD/P3.0 为接收端口，以全双工模式进行接收/发送。

图 8-4 为串行口工作模式 3 的功能结构示意图及其接收/发送时序图。

由图 8-4 可知，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：

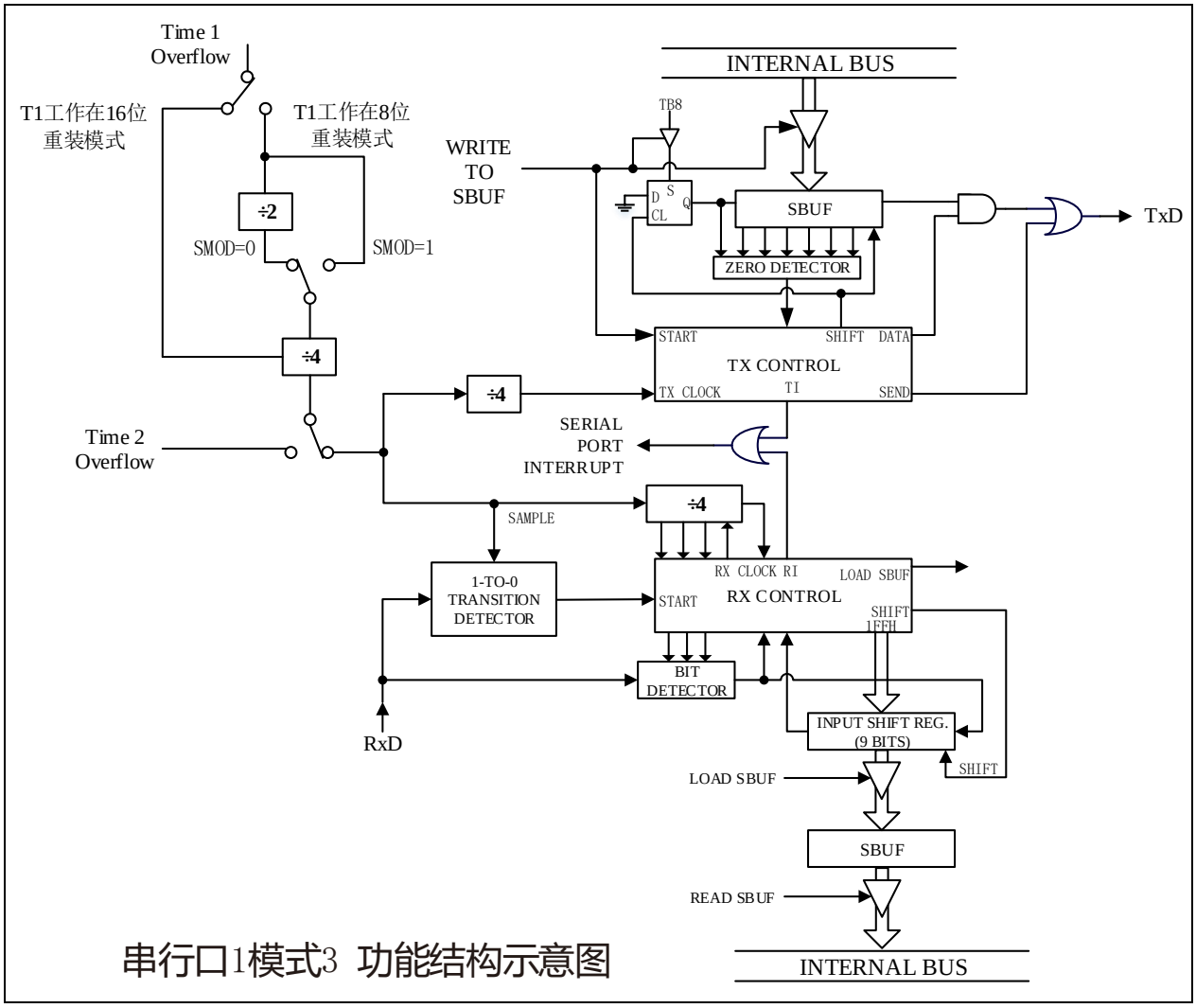
- ✓ RI=0
- ✓ SM2=0 或者 SM2-1，并且接收到的第 9 数据位 RB8=1。

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，并置位 RI=1，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。

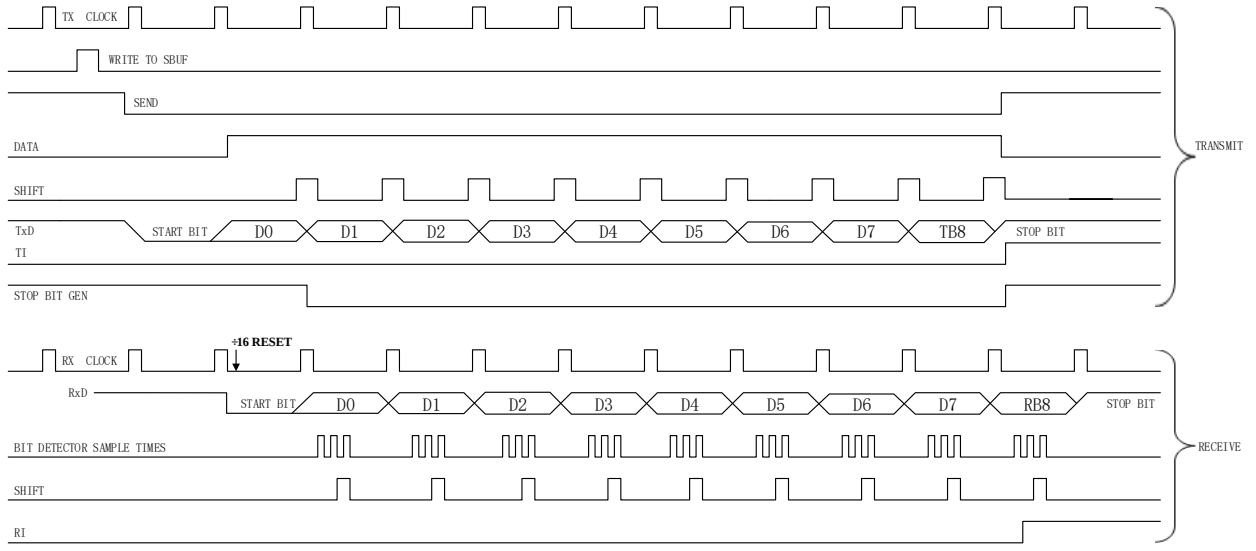
在模式 3 中，接收到的停止位与 SBUF、RB8 和 RI 无关。

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。

串行通信模式 3 的波特率也是可变的，可变的波特由定时器/计数器 1 或定时器 2 产生。



串行口1模式3 功能结构示意图



串行口1模式3 接收/发送时序图

模式 3 的波特率（优先选择定时器 2 产生波特率）为：

- 当串行口 1 用定时器 2 作为其波特率发生器时，
  串行口 1 的波特率=（定时器 T2 的溢出率）/4

（注意：此时波特率也与 SMOD 无关。）

✓ 当 T2 工作在 1T 模式（AUXR.2/T2x12=1）时，
- 定时器 2 的溢出率=SYSclk/（65536-[RL_TH2, RL_TL2]）；
- 即此时，串行口 1 的波特率=SYSclk/（65536-[RL_TH2, RL_TL2]）/4

✓ 当 T2 工作在 12T 模式（AUXR.2/T2x12=0）时，
- 定时器 2 的溢出率=SYSclk /12/（65536-[RL_TH2, RL_TL2]）；
- 即此时，串行口 1 的波特率=SYSclk/12/（65536-[RL_TH2, RL_TL2]）/4

RL_TH2 是 T2H 的自动重装载寄存器，RL_TL2 是 T2L 的自动重装载寄存器。

➢ 当串行口 1 用定时器 1 作为其波特率发生器且定时器 1 工作于模式 0（16 位自动重装载模式）时，串行口 1 的波特率=（定时器 1 的溢出率）/4.

（注意：此时波特率与 SMOD 无关。）

✓ 当定时器 1 工作于模式 0（16 位自动重装载模式）且 T1x12=0 时，
- 定时器 1 的溢出率=SYSclk/12/（65536-[RL_TH1, RL_TL1]）；
- 即此时，串行口 1 的波特率=SYSclk/12/（65536-[RL_TH1, RL_TL1]）/4

✓ 当定时器 1 工作于模式 0（16 位自动重装载模式）且 T1x12=1 时，
- 定时器 1 的溢出率=SYSclk/（65536-RL_TH1, RL_TL1]）
- 即此时，串行口 1 的波特率=SYSclk/（65536-[RL_TH1, RL_TL1]）/4

RL_TH1 是 TH1 的自动重装载寄存器，RL_TL1 是 TL1 的自动重装载寄存器。

➢ 当串行口 1 用定时器 1 作为其波特率发生器且定时器 1 工作于模式 2（8 位自动重装模式）时，串行口 1 的波特率=（2^{SMOD} /32）x（定时器 1 的溢出率）

✓ 当定时器 1 工作于模式 2（8 位自动重装模式）且 T1x12=0 时，
- 定时器 1 的溢出率=SYSclk/12/（256-TH1）；
- 即此时，串行口 1 的波特率=（2^{SMOD} /32）×SYSclk/12/（256-TH1）

✓ 当定时器 1 工作于模式 2（8 位自动重装模式）且 T1x12=1 时，
- 定时器 1 的溢出率=SYSclk/（256-TH1）
- 即此时，串行口 1 的波特率=（2^{SMOD} /32）×SYSclk/（256-TH1）

可见，模式 3 和模式 1 一样，其波特率可通过软件对定时器/计数器 1 或定时器 2 的设置进行波特率的选择，是可变的。

19.3 串行口 1 的波特率设置

----串口 1 和串口 2 的波特率相同时，串口 1 和串口 2 可共享 T2 作波特率发生器

STC15W4K32S4 系列单片机串行口 1 的波特率随所选工作模式的不同而异，

- 对于工作模式 0 和模式 2，其波特率与系统时钟频率 SYSclk 和 PCON 中的波特率选择位 SMOD 有关；
- 而模式 1 和模式 3 的波特率除与 SYSclk 和 PCON 位有关外，还与定时器/计数器 1 或定时器 2 设置有关。通过对定时器/计数器 1 或定时器 2 的设置，可选择不同的波特率，所以这种波特率是可变的。建议用户优先选择定时器 2 作为串行口 1 的波特率发生器。

【说明】:

- ★ 当串口 1 和串口 2 的波特率相同时，串口 1 和串口 2 可以共享波特率发生器，此时建议用户选择定时器 T2 作为串口 1 的波特率发生器；
- ★ 当串口 1 和串口 2 的波特率不同时，才建议选择定时器 T1 作为串口 1 的波特率发生器（因串口 2 固定使用定时器 T2 作波特率发生器）。

- 串行通信模式 0，其波特率与系统时钟频率 SYSclk 有关。
  - ✓ 当模式 0 的通信速度设置位 UART_M0x6/AUXR.5=0 时，其波特率=SYSclk / 12。
  - ✓ 当模式 0 的通信速度设置位 UART_M0x6/AUXR.5=1 时，其波特率=SYSclk / 2。

  一旦 SYSclk 选定且 UART_M0x6/AUXR.5 设置好，则串行通信工作模式 0 的波特率固定不变。

- 串行通信工作模式 2，其波特率除与 SYSclk 有关外，还与 SMOD 位有关。

  其基本表达式为：串行通信模式 2 波特率=2^{SMOD} / 64×（SYSclk 系统工作时钟频率）
  - ✓ 当 SMOD=1 时，波特率=2/64（SYSclk）=1/32（SYSclk）；
  - ✓ 当 SMOD=0 时，波特率=1/64（SYSclk）。

  当 SYSclk 选定后，通过软件设置 PCON 中的 SMOD 位，可选择两种波特率。所以，这种模式的波特率基本固定。

- 串行通信模式 1 和 3，其波特率是可变的（建议用户优先选择定时器 T2 作为串口 1 的波特率发生器）：
  - ✓ 当串行口 1 用定时器 2 作为其波特率发生器时，

    串行口 1 的波特率=（定时器 T2 的溢出率）/4

    （注意：此时波特率也与 SMOD 无关。）
    - ➔ 当 T2 工作在 1T 模式（AUXR.2/T2x12=1）时，
      - 定时器 2 的溢出率=SYSclk /（65536-[RL_TH2, RL_TL2]）；
      - 即此时，串行口 1 的波特率=SYSclk /（65536 - [RL_TH2, RL_TL2]）/ 4
    - ➔ 当 T2 工作在 12T 模式（AUXR.2/T2x12=0）时，
      - 定时器 2 的溢出率=SYSclk /12/（65536-[RL_TH2, RL_TL2]）；
      - 即此时，串行口 1 的波特率=SYSclk / 12 /（65536 - [RL_TH2, RL_TL2]）/ 4
  - ✓ 当串行口 1 用定时器 1 作为其波特率发生器，且定时器 1 工作于模式 0（16 位自动重装载模式）时，

    串行口 1 的波特率=（定时器 1 的溢出率）/4

    （注意：此时波特率与 SM0D 无关。）
    - ➔ 当定时器 1 工作于模式 0（16 位自动重装载模式）且 T1x12=0 时，
      - 定时器 1 的溢出率=SYSclk/12/（65536-[RL_TH1, RL_TL1]）；
      - 即此时，串行口 1 的波特率=SYSclk / 12 /（65536-[RL_TH1, RL_TL1]）/4

➔ 当定时器 1 工作于模式 0（16 位自动重装载模式）且 T1x12=1 时，

- 定时器 1 的溢出率=SYSclk /（65536 - [RL_TH1, RL_TL1]）；
- 即此时，串行口 1 的波特率=SYSclk /（65536 - [RL_TH1, RL_TL1]）/4

✓ 当串行口 1 用定时器 1 作为其波特率发生器且定时器 1 工作于模式 2（8 位自动重装模式）时，

串行口 1 的波特率=（2^{SMOD} / 32）×（定时器 1 的溢出率）

➔ 当定时器 1 工作于模式 2（8 位自动重装模式）且 T1x12=0 时，

- 定时器 1 的溢出率=SYSclk / 12 /（256 - TH1）；
- 即此时，串行口 1 的波特率=（2^{SMOD} / 32）× SYSclk / 12 /（256 - TH1）

➔ 当定时器 1 工作于模式 2（8 位自动重装模式）且 T1x12=1 时，

- 定时器 1 的溢出率=SYSclk /（256 - TH1）
- 即此时，串行口 1 的波特率=（2^{SMOD} / 32）× SYSclk /（256 - TH1）

通过对定时器 1 和定时器 2 的设置，可灵活地选择不同的波特率。在实际应用中多半选用串行模式 1 或串行模式 3。显然，为选择波特率，关键在于定时器 1 和定时器 2 的溢出率的计算。SMOD 的选择，只需根据需要执行下列指令就可实现 SMOD=0 或 1：

```
MOV     PCON, #00H      ;使 SMOD=0
MOV     PCON, #80H      ;使 SMOD=1
```

SMOD 只占用电源控制寄存器 PCON 的最高一位，其他各位的具体设置应根据实际情况而定。

当用户选择定时器/计数器 1 作波特率发生器时，为选择波特率，关键在于定时器/计数器 1 的溢出率。下面介绍如何计算定时器/计数器 1 的溢出率。

定时器/计数器 1 的溢出率定义为：单位时间（秒）内定时器/计数器 1 回 0 溢出的次数，即定时器/计数器 1 的溢出率 = 定时器/计数器 1 的溢出次数/秒。

STC15W4K32S4 系列单片机设有 3 个定时器/计数器，定时器/计数器 1 具有 4 种工作方式，而常选用定时器/计数器 1 的工作方式 0（16 位自动重装载模式）及工作方式 2（8 位自动重装）作为波特率的溢出率。

以定时器/计数器 1 工作于定时模式的工作方式 2（8 位自动重装）为例：

设置定时器/计数器 1 工作于定时模式的工作方式 2（8 位自动重装），TL1 的计数输入来自于 SYSclk 经 12 分频或不分频（由 T1x12/AUXR.6 确定是 12 分频还是不分频）的脉冲。

当 T1x12/AUXR.6=0 时，单片机工作在 12T 模式，TL1 的计数输入来自于 SYSclk 经 12 分频的脉冲；

当 T1x12/AUXR.6=1 时，单片机工作在 1T 模式，TL1 的计数输入来自于 SYSclk 不经过分频的脉冲。

可见，定时器/计数器 1 的溢出率与 SYSclk 和自动重装值 N 有关，SYSclk 越大，特别是 N 越大，溢出率也就越高。

例如：当 N=FFN，则每隔一个时钟即溢出一次（极限情况）；

若 N=00H，则需每隔 256 个时钟才溢出一次；

当 SYSclk=6MHz 且 T1x12/AUXR.6=0 时，一个时钟为 2us；

当 SYSclk=6MHz 且 T1x12/AUXR.6=1 时，一个时钟约为 0.167μs（快 12 倍）。

当 SYSclk=12MHz 且 T1x12/AUXR.6=0 时，则一个时钟为 1μs；

当 SYSclk=12MHz 且 T1x12/AUXR.6=1 时，一个时钟约为 0.083μs（快 12 倍）。

对于一般情况下，

当 T1x12/AUXR.6=0 时，定时器/计数器 1 溢出一次所需的时间为：

$$(2^8 - N) \times 12\text{时钟} = (2^8 - N) \times 12 \times \frac{1}{\text{SYSclk}}$$

当 T1x12/AUXR.6=1 时，定时器/计数器 1 溢出一次所需的时间为：

$$(2^8 - N) \times 1\text{时钟} = (2^8 - N) \times \frac{1}{\text{SYSclk}}$$

于是得定时器/计数器每秒溢出的次数，即：

当 T1x12/AUXR.6=0 时，定时器/计数器 1 的溢出率=SYSclk / 12 ×（2^8 - N）（次/秒）

当 T1x12/AUXR.6=1 时，定时器/计数器 1 的溢出率=SYSclk ×（2^8 - N）（次/秒）

式中 SYSclk 为系统时钟频率，N 为再装入时间常数。

显然，选用定时器/计数器 2 作波特率的溢出率也一样。选用不同工作方式所获得波特率的范围不同。因为不同方式的计数位数不同，N 取值范围不同，且计数方式较复杂。

现以定时器/计数器 1 工作于方式 2（8 位自动重装模式）为例，

设：T1x12/AUXR.6=0，SYSclk=6MHz，N=FFH，

定时器/计数器 1 工作于方式 2 的溢出率为 6×10^6 / {12×（256-255）} = 0.5×10^6（次/秒）；

设：T1x12/AUXR.6=0，SYSclk=12MHz，N=FFH，

定时器/计数器 1 工作于方式 2 的溢出率=1×106（次/秒）；

设：T1x12/AUXR.6=0，SYSclk=12MHz，N=00H，

定时器/计数器 1 工作于方式 2 的溢出率 = 12×106 / 12×256 ≈ 3906（次/秒）

设：T1x12/AUXR.6=1，SYSclk=6MHz，N=FFH，

定时器/计数器 1 工作于方式 2 的溢出率为 6×106（256-255）= 6×106（次/秒）；

设：T1x12/AUXR.6=1，SYSclk=12MHz，N=00H，

定时器/计数器 1 工作于方式 2 的溢出率=12×106 / 256 = 46875（次/秒）

下表给出各种常用波特率与定时器/计数器 1 各参数之间的关系。

常用波特率与定时器/计数器 1 各参数关系(T1x12/AUXR.6=0)

常用波特率		系统时中频率 (MHz)	SMOD	定时器 1		
				C/T	方式	重新装入值
方式 0 MAX：1M		12	×	×	×	×
方式 2 MAX：375K		12	1	×	×	×
方式 1 和 3	62.5K	12	1	0	2	FFH
	19.2K	11.059	1	0	2	FDH
	9.6K	11.059	0	0	2	FDH
	4.8K	11.059	0	0	2	FAH
	2.4K	11.059	0	0	2	F4H
	1.2K	11.059	0	0	2	F8H
	137.5	11.986	0	0	2	1DH
	110	6	0	0	2	72H
	110	12	0	0	1	FFFBH

设置波特率的初始化程序段如下:

```
MOV     TMOD, #20H          ;设置定时器/计数器 1 定时、工作方式 2
MOV     TH1, #xxH           ;设置定时常数 N
MOV     TL1, #xxH
SETB    TR1                 ;启动定时器/计数器 1
MOV     PCON, #80H          ;设置 SMOD=1
MOV     SCON, #50H          ;设置串行通信方式 1
```

执行上述程序段后,即可完成对定时器/计数器 1 的操作方式及串行通信的工作方式和波特率的设置。

由于用其他工作方式设置波特率计算方法较复杂，一般应用较少，故不一一论述。

当用户选择定时器 2 作波特率发生器时，为选择波特率，关键在于定时器 2 的溢出率。当用户选择定时器 2 作波特率发生器时，定时器/计数器 1 可以释放出来作为定时器/计数器/时钟输出使用。

用户在程序中如何具体使用串口 1 和定时器 T2

1.设置串口 1 的工作模式，SCON 寄存器中的 SM0 和 SM1 两位决定了串口 1 的 4 种工作模式。
2.设置串口 1 的波特率，使用定时器 2 寄存器 T2H 及 T2L
3.设置寄存器 AUXR 中的位 T2x12/AUXR.2，确定定时器 2 速度是 1T 还是 12T
4.启动定时器 2，让 T2R 位为 1，T2H/T2L 定时器 2 寄存器就立即开始计数。
5.设置串口 1 的中断优先级，及打开中断相应的控制位是：PS，ES，EA
6.如要串口 1 接收，将 REN 置 1 即可
如要串口 1 发送，将数据送入 SBUF 即可，
接收完成标志 RI，发送完成标志，要由软件清 0。

当串口 1 工作在模式 1 和模式 3 时，计算相应的波特率需要设置的重装载数，结果送入 T2H/T2L 寄存器，计算自动重装数 RELOAD：

1.计算 RELOAD

计算公式：RELOAD = 65536 - INT(SYSclk / Baud0 / 4 + 0.5)

计算出的 RELOAD 数直接送 T2H/T2L 寄存器

式中：INT()表示取整运算即舍去小数，在式中加 0.5 可以达到四舍五入的目的

SYSclk = 晶振频率

Baud0 = 标准波特率

2.设置 AUXR.2/T2x12=1，定时器 2 工作在 1T 模式

3.计算用 RELOAD 产生的波特率：Baud = SYSclk / (65536 - RELOAD) / 4

4.计算误差：error = (Baud - Baud0) / Baud0×100%

5.如果误差绝对值>3%要更换波特率或者更换晶体频率，重复步骤 1 - 4

【例】：SYSclk=22.1184MHz，Baud0=57600

1.RELOAD = 65536 - INT(22118400 / 57600 / 4 + 0.5)
= 65536 - INT(96.5)
=65536-96
=65440
= 0FFA0H

2.设置 AUXR.2/T2x12=1，定时器 2 工作在 1T 模式

3. Baud = 22118400 / (65536-65440) / 4
= 57600

4.误差等于零

【例】: SYSclk=12MHz，Baud0=57600

1.RELOAD = 65536 - INT(12000000 / 57600 / 4 + 0.5)
= 65536 - INT(52.0833 + 0.5)
= 65536 - INT(52.5833)
= 65536 - 52
= 65484
= 0FECCH

2.设置 AUXR.2/T2x12=1，定时器 2 工作在 1T 模式

3.Baud = 12000000 / (66536 - 65484) / 4
= 57692

4.error = (57692-57600) / 57600×100%
= 0.16%

19.4 串行口 1 的测试程序(C 和汇编)

19.4.1 定时器 2 作串口 1 波特率发生器的测试程序(C 和汇编)

1.C 程序:

```
/*----STC15F2K60S2 系列定时器 2 用作串口 1 的波特率发生器举例--------------*/
/*----在 Keil C 开发环境中选择  Intel 8052 编译，头文件包含<reg51.h>即可----*/
//假定测试芯片的工作频率为 18.432MHz

#include "reg51.h"
#include "intrins.h"

typedef     unsigned char     BYTE;
typedef     unsigned int      WORD;

#define     FOSC              18432000L          //系统频率
#define     BAUD              115200             //串口波特率
#define     NONE_PARITY       0                  //无校验
#define     ODD_PARITY        1                  //奇校验
#define     EVEN_PARITY       2                  //偶校验
#define     MARK_PARITY       3                  //标记校验
#define     SPACE_PARITY      4                  //空白校验
#define     PARITYBIT         EVEN_PARITY        //定义校验位

sfr         AUXR = 0x8e;                         //辅助寄存器
sfr         T2H = 0xd6;                          //定时器 2 高 8 位
sfr         T2L = 0xd7;                          //定时器 2 低 8 位

sbit        P22 = P2^2;

bit         busy;

void SendData(BYTE dat);
void SendString(char *s);
void main()
{
#if (PARITYBIT == NONE_PARITY)
    SCON = 0x50;                                 //8 位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
    SCON = 0xda;                                 //9 位可变波特率,校验位初始为 1
#elif (PARITYBIT == SPACE_PARITY)
    SCON = 0xd2; /                               /9 位可变波特率,校验位初始为 0
#endif
```

```
    T2L = (65536 - (FOSC/4/BAUD));                  //设置波特率重装值
    T2H = (65536 - (FOSC/4/BAUD))>>8;
    AUXR = 0x14;                                    //T2 为 1T 模式, 并启动定时器 2
    AUXR |= 0x01;                                   //选择定时器 2 为串口 1 的波特率发生器
    ES = 1;                                         //使能串口 1 中断
    EA = 1;

    SendString("STC15F2K60S2\r\nUart Test !\r\n");
    while(1);
}
/*----------------------------UART 中断服务程序-----------------------------*/
void Uart() interrupt 4 using 1
{
    if (RI)
    {
        RI = 0;                                     //清除 RI 位
        P0 = SBUF;                                  //P0 显示串口数据
        P22 = RB8;                                  //P2.2 显示校验位
    }
    if (TI)
    {
        TI = 0;                                     //清除 TI 位
        busy = 0;                                   //清忙标志
    }
}
/*----------------------------发送串口数据----------------------------*/
void SendData(BYTE dat)
{
    while (busy);                                   //等带前面的数据发送完成
    ACC = dat;                                      //获取校验位 P (PSW.0)
    if (P)                                          //根据 P 来设置校验位
    {
#if (PARITYBIT == ODD_PARITY)
        TB8 = 0;                                    //设置校验位为 0
#elif (PARITYBIT == EVEN_PARITY)
        TB8 = 1;                                    //设置校验位为 1
#endif
    }
    else
    {
#if (PARITYBIT == ODD_PARITY)
        TB8 = 1;                                    //设置校验位为 1
#elif (PARITYBIT == EVEN_PARITY)
        TB8 = 0;                                    //设置校验位为 0
#endif
    }
```

```
    busy = 1;
    SBUF = ACC;                                 //写数据到 UART 数据寄存器
}
/*----------------------------发送字符串----------------------------*/
void SendString(char *s)
{
    while (*s)                                  //检测字符串结束标志
    {
        SendData(*s++);                         //发送当前字符
    }
}
```

2. 汇编程序：

```
/*----STC15F2K60S2 系列定时器 2 用作串口 1 的波特率发生器举例--------------*/
/*----在 Keil C 开发环境中选择  Intel 8052 编译，头文件包含<reg51.h>即可----*/
;本示例在 Keil 开发环境下请选择 Intel 的 8052 芯片型号进行编译
;假定测试芯片的工作频率为 18.432MHz

#define     NONE_PARITY     0               ;无校验
#define     ODD_PARITY      1               ;奇校验
#define     EVEN_PARITY     2               ;偶校验
#define     MARK_PARITY     3               ;标记校验
#define     SPACE_PARITY    4               ;空白校验

#define     PARITYBIT       EVEN_PARITY     ;定义校验位
;---------------------------------------
AUXR        EQU             08EH            ;辅助寄存器
T2H         DATA            0D6H            ;定时器 2 高 8 位
T2L         DATA            0D7H            ;定时器 2 低 8 位
;---------------------------------------
BUSY        BIT             20H.0           ;忙标志位
;---------------------------------------
    ORG     0000H
    LJMP    START
    ORG     0023H
    LJMP    UART_ISR
;---------------------------------------
    ORG     0100H
START:
    CLR     BUSY
    CLR     EA
    MOV     SP, #3FH

#if (PARITYBIT == NONE_PARITY)
    MOV     SCON, #50H                      ;8 位可变波特率
```

```
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
    MOV     SCON, #0DAH                 ;9 位可变波特率,校验位初始为 1
#elif (PARITYBIT == SPACE_PARITY)
    MOV     SCON, #0D2H                 ;9 位可变波特率,校验位初始为 0
#endif
;-------------------------------
    MOV     T2L, #0D8H                  ;设置波特率重装值(65536-18432000/4/115200)
    MOV     T2H, #0FFH
    MOV     AUXR, #14H                  ;T2 为 1T 模式, 并启动定时器 2
    ORL     AUXR, #01H                  ;选择定时器 2 为串口 1 的波特率发生器
    SETB    ES                          ;使能串口中断
    SETB    EA

    MOV     DPTR, #TESTSTR              ;发送测试字符串
    LCALL   SENDSTRING

    SJMP    $
;----------------------------------------
TESTSTR:
    DB      "STC15F2K60S2 Uart1 Test !",0DH,0AH,0
;/*--------------------------UART 中断服务程序--------------------------*/
UART_ISR:
    PUSH    ACC
    PUSH    PSW
    JNB     RI, CHECKTI                 ;检测 RI 位
    CLR     RI                          ;清除 RI 位
    MOV     P0, SBUF                    ;P0 显示串口数据
    MOV     C, RB8
    MOV     P2.2, C                     ;P2.2 显示校验位

CHECKTI:
    JNB     TI, ISR_EXIT                ;检测 TI 位
    CLR     TI                          ;清除 TI 位
    CLR     BUSY                        ;清忙标志

ISR_EXIT:
    POP     PSW
    POP     ACC
    RETI
;/*--------------------------发送串口数据--------------------------*/
SENDDATA:
    JB      BUSY, $                     ;等待前面的数据发送完成
    MOV     ACC, A                      ;获取校验位 P (PSW.0)
    JNB     P, EVEN1INACC               ;根据 P 来设置校验位

ODD1INACC:
```

```
#if (PARITYBIT == ODD_PARITY)
    CLR     TB8                         ;设置校验位为 0
#elif (PARITYBIT == EVEN_PARITY)
    SETB    TB8                         ;设置校验位为 1
#endif
    SJMP    PARITYBITOK

EVEN1INACC:
#if (PARITYBIT == ODD_PARITY)
    SETB    TB8                         ;设置校验位为 1
#elif (PARITYBIT == EVEN_PARITY)
    CLR     TB8                         ;设置校验位为 0
#endif

PARITYBITOK:                            ;校验位设置完成
    SETB    BUSY
    MOV     SBUF, A                     ;写数据到 UART 数据寄存器
    RET
;/*----------------------------发送字符串----------------------------*/
SENDSTRING:
    CLR     A
    MOVC    A, @A+DPTR                  ;读取字符
    JZ      STRINGEND                   ;检测字符串结束标志
    INC     DPTR                        ;字符串地址+1
    LCALL   SENDDATA                    ;发送当前字符
    SJMP    SENDSTRING

STRINGEND:
    RET
;------------------------------------------
END
```

19.4.2 定时器 1 模式 0(16 位自动重装载)作串口 1 波特率发生器程序(C 和汇编)

1.C 程序:

```
/*----STC15F2K60S2 系列定时器 1 用作串口 1 的波特率发生器举例--------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可----------*/
//假定测试芯片的工作频率为 18.432MHz

#include "reg51.h"
#include "intrins.h"

typedef     unsigned char     BYTE;
typedef     unsigned int      WORD;

#define     FOSC              18432000L          //系统频率
#define     BAUD              115200             //串口波特率

#define     NONE_PARITY       0                  //无校验
#define     ODD_PARITY        1                  //奇校验
#define     EVEN_PARITY       2                  //偶校验
#define     MARK_PARITY       3                  //标记校验
#define     SPACE_PARITY      4                  //空白校验

#define     PARITYBIT         EVEN_PARITY        //定义校验位

sfr         AUXR = 0x8e;                         //辅助寄存器

sbit        P22 = P2^2;

bit         busy;

void SendData(BYTE dat);
void SendString(char *s);
void main()
{
#if (PARITYBIT == NONE_PARITY)
    SCON = 0x50;                                 //8 位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
    SCON = 0xda;                                 //9 位可变波特率,校验位初始为 1
#elif (PARITYBIT == SPACE_PARITY)
    SCON = 0xd2;                                 //9 位可变波特率,校验位初始为 0
#endif
    AUXR = 0x40;                                 //定时器 1 为 1T 模式
    TMOD = 0x00;                                 //定时器 1 为模式 0(16 位自动重载)
    TL1 = (65536 - (FOSC/4/BAUD));               //设置波特率重装值
```

```
    TH1 = (65536 - (FOSC/4/BAUD))>>8;
    TR1 = 1;                                    //定时器 1 开始启动
    ES = 1;                                     //使能串口中断
    EA = 1;

    SendString("STC15F2K60S2\r\nUart Test !\r\n");
    while(1);
}
/*----------------------------UART 中断服务程序-----------------------------*/
void Uart() interrupt 4 using 1
{
    if (RI)
    {
        RI = 0;                                 //清除 RI 位
        P0 = SBUF;                              //P0 显示串口数据
        P22 = RB8;                              //P2.2 显示校验位
    }
    if (TI)
    {
        TI = 0;                                 //清除 TI 位
        busy = 0;                               //清忙标志
    }
}
/*----------------------------发送串口数据----------------------------*/
void SendData(BYTE dat)
{
    while (busy);                               //等待前面的数据发送完成
    ACC = dat;                                  //获取校验位 P (PSW.0)
    if (P)                                      //根据 P 来设置校验位
    {
#if (PARITYBIT == ODD_PARITY)
    TB8 = 0;                                    //设置校验位为 0
#elif (PARITYBIT == EVEN_PARITY)
    TB8 = 1;                                    //设置校验位为 1
#endif
    }
    else
    {
#if (PARITYBIT == ODD_PARITY)
    TB8 = 1;                                    //设置校验位为 1
#elif (PARITYBIT == EVEN_PARITY)
    TB8 = 0;                                    //设置校验位为 0
#endif
    }
    busy = 1;
    SBUF = ACC;                                 //写数据到 UART 数据寄存器
```

```
}
/*---------------------------发送字符串---------------------------*/
void SendString(char *s)
{
    while (*s)                                  //检测字符串结束标志
    {
        SendData(*s++);                         //发送当前字符
    }
}
```

2. 汇编程序：

```
/*----STC15F2K60S2 系列定时器 1 用作串口 1 的波特率发生器举例--------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-----------*/
;假定测试芯片的工作频率为 18.432MHz
#define     NONE_PARITY     0               ;无校验
#define     ODD_PARITY      1               ;奇校验
#define     EVEN_PARITY     2               ;偶校验
#define     MARK_PARITY     3               ;标记校验
#define     SPACE_PARITY    4               ;空白校验
#define     PARITYBIT       EVEN_PARITY     ;定义校验位
;---------------------------------------
AUXR        EQU     08EH                    ;辅助寄存器
BUSY        BIT     20H.0                   ;忙标志位
;---------------------------------------
    ORG     0000H
    LJMP    START

    ORG     0023H
    LJMP    UART_ISR
;---------------------------------------
    ORG     0100H
START:
    CLR     BUSY
    CLR     EA
    MOV     SP, #3FH
#if (PARITYBIT == NONE_PARITY)
    MOV     SCON, #50H                      ;8 位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
    MOV     SCON, #0DAH                     ;9 位可变波特率,校验位初始为 1
#elif (PARITYBIT == SPACE_PARITY)
    MOV     SCON, #0D2H                     ;9 位可变波特率,校验位初始为 0
#endif
;-------------------------------
    MOV     AUXR, #40H                      ;定时器 1 为 1T 模式
    MOV     TMOD, #00H                      ;定时器 1 为模式 0(16 位自动重载)
```

```
    MOV     TL1, #0D8H                  ;设置波特率重装值(65536-18432000/4/115200)
    MOV     TH1, #0FFH
    SETB    TR1                         ;定时器 1 开始运行
    SETB    ES                          ;使能串口中断
    SETB    EA

    MOV     DPTR, #TESTSTR              ;发送测试字符串
    LCALL   SENDSTRING

    SJMP    $
;----------------------------------------
TESTSTR:
    DB      "STC15F2K60S2 Uart1 Test !",0DH,0AH,0
;/*----------------------------UART 中断服务程序----------------------------*/
UART_ISR:
    PUSH    ACC
    PUSH    PSW
    JNB     RI, CHECKTI                 ;检测 RI 位
    CLR     RI                          ;清除 RI 位
    MOV     P0, SBUF                    ;P0 显示串口数据
    MOV     C, RB8
    MOV     P2.2, C                     ;P2.2 显示校验位

CHECKTI:
    JNB     TI, ISR_EXIT                ;检测 TI 位
    CLR     TI                          ;清除 TI 位
    CLR     BUSY                        ;清忙标志

ISR_EXIT:
    POP     PSW
    POP     ACC
    RETI
;/*----------------------------发送串口数据----------------------------*/
SENDDATA:
    JB      BUSY, $                     ;等待前面的数据发送完成
    MOV     ACC, A                      ;获取校验位 P (PSW.0)
    JNB     P, EVEN1INACC               ;根据 P 来设置校验位

ODD1INACC:
#if (PARITYBIT == ODD_PARITY)
    CLR     TB8                         ;设置校验位为 0
#elif (PARITYBIT == EVEN_PARITY)
    SETB    TB8                         ;设置校验位为 1
#endif
    SJMP    PARITYBITOK
```

```
EVEN1INACC:
#if (PARITYBIT == ODD_PARITY)
    SETB    TB8                         ;设置校验位为 1
#elif (PARITYBIT == EVEN_PARITY)
    CLR     TB8                         ;设置校验位为 0
#endif

PARITYBITOK:                            ;校验位设置完成
    SETB    BUSY
    MOV     SBUF, A                     ;写数据到 UART 数据寄存器
    RET
;/*----------------------------发送字符串----------------------------*/
SENDSTRING:
    CLR     A
    MOVC    A, @A+DPTR                  ;读取字符
    JZ      STRINGEND                   ;检测字符串结束标志
    INC     DPTR                        ;字符串地址+1
    LCALL   SENDDATA                    ;发送当前字符
    SJMP    SENDSTRING

STRINGEND:
    RET
;----------------------------------------
END
```

19.4.3 定时器 1 模式 2(8 位自动重装载)作串口 1 波特率发生器程序(建议不学)

1.C 程序:

```
/*----STC15F2K60S2 系列定时器 1 用作串口 1 的波特率发生器举例------------*/
/*----在 Kei C 开发环境中选择  Intel 8052 编译，头文件包含<reg51.h>即可---*/
//假定测试芯片的工作频率为 18.432MHz

#include"reg51.h"
#include "intrins.h"

typedef     unsigned char   BYTE;
typedef     unsigned int    WORD;

#define     FOSC            18432000L       //系统频率
#define     BAUD            115200          //串口波特率

#define     NONE_PARITY     0               //无校验
#define     ODD_PARITY      1               //奇校验
#define     EVEN_PARITY     2               //偶校验
```

```
#define     MARK_PARITY   3                   //标记校验
#define     SPACE_PARITY  4                   //空白校验

#define     PARITYBIT     EVEN_PARITY         //定义校验位

sfr         AUXR = 0x8e;                      //辅助寄存器
sbit        P22 = P2^2;
bit         busy;

void SendData(BYTE dat);
void SendString(char *s);
void main()
{
#if (PARITYBIT == NONE_PARITY)
    SCON = 0x50;                              //8 位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
    SCON = 0xda;                              //9 位可变波特率,校验位初始为 1
#elif (PARITYBIT == SPACE_PARITY)
    SCON = 0xd2;                              //9 位可变波特率,校验位初始为 0
#endif
    AUXR = 0x40;                              //定时器 1 为 1T 模式
    TMOD = 0x20;                              //定时器 1 为模式 2(8 位自动重载)
    TL1 = (256 - (FOSC/32/BAUD));             //设置波特率重装值
    TH1 = (256 - (FOSC/32/BAUD));
    TR1 = 1;                                  //定时器 1 开始规则
    ES = 1;                                   //使能串口中断
    EA = 1;

    SendString("STC15F2K60S2\r\nUart Test !\r\n");
    while(1);
}
/*----------------------------UART 中断服务程序-----------------------------*/
void Uart() interrupt 4 using 1
{
    if (RI)
    {
        RI = 0;                               //清除 RI 位
        P0 = SBUF;                            //P0 显示串口数据
        P22 = RB8;                            //P2.2 显示校验位
    }
    if (TI)
    {
        TI = 0;                               //清除 TI 位
        busy = 0;                             //清忙标志
    }
}
```

```
/*---------------------------发送串口数据---------------------------*/
void SendData(BYTE dat)
{
    while (busy);                                   //等待前面的数据发送完成
    ACC = dat;                                      //获取校验位 P (PSW.0)
    if (P)                                          //根据 P 来设置校验位
    {
#if (PARITYBIT == ODD_PARITY)
    TB8 = 0;                                        //设置校验位为 0
#elif (PARITYBIT == EVEN_PARITY)
    TB8 = 1;                                        //设置校验位为 1
#endif
    }
    else
    {
#if (PARITYBIT == ODD_PARITY)
    TB8 = 1;                                        //设置校验位为 1
#elif (PARITYBIT == EVEN_PARITY)
    TB8 = 0;                                        //设置校验位为 0
#endif
}
    busy = 1;
    SBUF = ACC;                                     //写数据到 UART 数据寄存器
}
/*---------------------------发送字符串---------------------------*/
void SendString(char *s)
{
    while (*s)                                      //检测字符串结束标志
    {
        SendData(*s++);                             //发送当前字符
    }
}
```

2. 汇编程序：

```
/*----STC15F2K60S2 系列定时器 1 用作串口 1 的波特率发生器举例------------*/
/*----在 Kei C 开发环境中选择  Intel 8052 编译，头文件包含<reg51.h>即可---*/
;假定测试芯片的工作频率为 18.432MHz

#define     NONE_PARITY     0               ;无校验
#define     ODD_PARITY      1               ;奇校验
#define     EVEN_PARITY     2               ;偶校验
#define     MARK_PARITY     3               ;标记校验
#define     SPACE_PARITY    4               ;空白校验

#define     PARITYBIT       EVEN_PARITY     ;定义校验位
```

```
;---------------------------------------
AUXR        EQU     08EH                    ;辅助寄存器
BUSY        BIT     20H.0                   ;忙标志位
;---------------------------------------
    ORG     0000H
    LJMP    START

    ORG     0023H
    LJMP    UART_ISR
;---------------------------------------
    ORG     0100H
START:
    CLR     BUSY
    CLR     EA
    MOV     SP, #3FH
#if (PARITYBIT == NONE_PARITY)
    MOV     SCON, #50H                      ;8 位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
    MOV     SCON, #0DAH                     ;9 位可变波特率,校验位初始为 1
#elif (PARITYBIT == SPACE_PARITY)
    MOV     SCON, #0D2H                     ;9 位可变波特率,校验位初始为 0
#endif
;------------------------------
    MOV     AUXR, #40H                      ;定时器 1 为 1T 模式
    MOV     TMOD, #20H                      ;定时器 1 为模式 2(8 位自动重载)
    MOV     TL1, #0FBH                      ;设置波特率重装值(256-18432000/32/115200)
    MOV     TH1, #0FBH
    SETB    TR1                             ;定时器 1 开始运行
    SETB    ES                              ;使能串口中断
    SETB    EA

    MOV     DPTR, #TESTSTR                  ;发送测试字符串
    LCALL   SENDSTRING

    SJMP    $
;---------------------------------------
TESTSTR:
    DB      "STC15F2K60S2 Uart1 Test !",0DH,0AH,0
;/*--------------------------UART  中断服务程序--------------------------*/
UART_ISR:
    PUSH    ACC
    PUSH    PSW
    JNB     RI, CHECKTI                     ;检测 RI 位
    CLR     RI                              ;清除 RI 位
    MOV     P0, SBUF                        ;P0 显示串口数据
    MOV     C, RB8
```

```
    MOV     P2.2, C                         ;P2.2 显示校验位

CHECKTI:
    JNB     TI, ISR_EXIT                    ;检测 TI 位
    CLR     TI                              ;清除 TI 位
    CLR     BUSY                            ;清忙标志

ISR_EXIT:
    POP     PSW
    POP     ACC
    RETI
;/*----------------------------发送串口数据----------------------------*/
SENDDATA:
    JB      BUSY, $                         ;等待前面的数据发送完成
    MOV     ACC, A                          ;获取校验位 P (PSW.0)
    JNB     P, EVEN1INACC                   ;根据 P 来设置校验位

ODD1INACC:
#if (PARITYBIT == ODD_PARITY)
    CLR     TB8                             ;设置校验位为 0
#elif (PARITYBIT == EVEN_PARITY)
    SETB    TB8                             ;设置校验位为 1
#endif
    SJMP    PARITYBITOK

EVEN1INACC:
#if (PARITYBIT == ODD_PARITY)
    SETB    TB8                             ;设置校验位为 1
#elif (PARITYBIT == EVEN_PARITY)
    CLR     TB8                             ;设置校验位为 0
#endif

PARITYBITOK:                                ;校验位设置完成
    SETB    BUSY
    MOV     SBUF, A                         ;写数据到 UART 数据寄存器
    RET
;/*----------------------------发送字符串----------------------------*/
SENDSTRING:
    CLR     A
    MOVC    A, @A+DPTR                      ;读取字符
    JZ      STRINGEND                       ;检测字符串结束标志
    INC     DPTR                            ;字符串地址+1
    LCALL   SENDDATA                        ;发送当前字符
    SJMP    SENDSTRING

STRINGEND:
```

```
    RET
;---------------------------------------
END
```

19.5 串行口 2 的相关寄存器

符号	描述	地址	位地址及符号								复位值
			MSB							LSB	
S2CON	Serial 2 Control register	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0100 0000B
S2BUF	Serial 2 Buffer	9BH									xxxx xxxxB
T2H	定时器 2 高 8 位，装入重装数	D6H									0000 0000B
T2L	定时器 2 低 8 位，装入重装数	D7H									0000 0000B
AUXR	辅助寄存器	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000 0001B
IE	Interrupt Enable	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000 0000B
IE2	Interrupt Enable 2	AFH	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000 0000B
IP2	Interrupt Priority 2 Low	B5H	-	-	-	PX4	PPWMFD	PPWM	PSPI	PS2	xxx0 0000B
P_SW2	外围设备功能切换控制寄存器	BAH	EAXSFR	-	-	-	-	S4_S	S3_S	S2_S	0000 x000B

1. 串行口 2 的控制寄存器 S2CON

串行口 2 控制寄存器 S2CON 用于确定串行口 2 的工作方式和某些控制功能。其格式如下：

S2CON：串行口 2 控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
S2CON	9AH	name	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI

S2SM0：指定串行口 2 的工作方式，如下表所示：

S2SM0	工作方式	功能说明	波特率
0	方式 0	8 位 UART，波特率可变	（定时器 T2 的溢出率）/4
1	方式 1	9 位 UART，波特率可变	（定时器 T2 的溢出率）/4
当 AUXR.2/T2x12=1 时，定时器 T2 的溢出率 = SYSclk /（65536 - [RL_TH2, RL_TL2]） 当 AUXR.2/T2x12=0 时，定时器 T2 的溢出率 = SYSclk / 12 /（65536 - [RL_TH2, RL_TL2]） 式中 RL_TH2 是 T2H 的重装载寄存器，RL_TL2 是 T2L 的重装载寄存器。			

B6： 保留，该位复位后为 1

S2SM2：允许方式 1 多机通信控制位。

- 在方式 1 时，如果 S2SM2 位为 1 且 S2REN 位为 1，则接收机处于地址选状态。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。
- 在方式 1 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
- 方式 0 是非多机通信方式，在这种方式时，要设置 S2SM2 应为 0。

S2REN：允许/禁止串行口 2 接收控制位。

- 由软件置位 S2REN，即 S2REN=1 为允许串行接收状态，可启动串行接收器 RxD2，开始接收信息。
- 软件复位 S2REN，即 S2REN=0，则禁止接收。

S2TB8： 在方式 1，S2TB8 为要发送的第 9 位数据，按需要由软件置位或清 0。例如，可用作数据的校验位或多机通信中表示地址帧/数据帧的标志位。在方式 0 中，该位不用。

S2RB8： 在方式 1，S2RB8 是接收到的第 9 位数据，作为奇偶校验位或地址/数据帧的标志位。方式 0 中不用 S2RB8（置 S2SM2=0，S2RB8 是接收到的停止位）。

S2TI： 发送中断请求标志位。在停止位开始发送时由 S2TI 内部硬件置位，即 S2TI=1，响应中断后 S2TI 必须用软件清零。

S2RI： 接收中断请求标志位。在串行接收到停止位的中间时刻 S2RI 由内部硬件置位，即 S2RI=1，向 CPU 发中断申请，响应中断后 S2RI 必须由软件清零。

S2CON 的字节地址为 9AH，不可位寻址。

串行通信的中断请求：

- 当一帧发送完成，内部硬件自动置位 S2TI，即 S2TI=1，请求中断处理；
- 当接收完一帧信息时，内部硬件自动置位 S2RI，即 S2RI=1，请求中断处理。

由于 S2TI 和 S2RI 以“或逻辑”关系向主机请求中断，所以主机响应中断时事先并不知道是 S2TI 还是 S2RI 请求的中断，必须在中断服务程序中查询 S2TI 和 S2RI 进行判别，然后分别处理。因此，两个中断请求标志位均不能由硬件自动置位，必须通过软件清 0，否则将出现一次请求多次响应的错误。

2.串行口 2 的数据缓冲寄存器 S2BUF

STC15 系列单片机的串行口 2 数据缓冲寄存器（S2BUF）的地址是 9BH，实际是 2 个缓冲器，写 S2BUF 的操作完成待发送数据的加载，读 S2BUF 的操作可获得已接收到的数据。两个操作分别对应两个不同的寄存器，1 个是只写寄存器，1 个是只读寄存器。

串行通道内设有数据寄存器。在所有的串行通信方式中，在写入 S2BUF 信号（MOV S2BUF, A）的控制下，把数据装入相同的 9 位移位寄存器，前面 8 位为数据字节，其最低位为移位寄存器的输出位。根据不同的工作方式会自动将“1”或 S2TB8 的值装入移位寄存器的第 9 位，并进行发送。

串行通道的接收寄存器是一个输入移位寄存器。在方式 0 和方式 1 时均为 9 位。当一帧接收完毕，移位寄存器中的数据字节装入串行数据缓冲器 S2BUF 中，其第 9 位则装入 S2CON 寄存器中的 S2RB8 位。如果由于 S2SM2 使得已接收到的数据无效时，S2RB8 和 S2BUF 中内容不变

由于接收通道内设有输入移位寄存器和 S2BUF 缓冲器，从而能使一接收完将数据由移位寄存器装入 S2BUF 后，可立即开始接收下一帧信息，主机应在该帧接收结束前从 S2BUF 缓冲器中将数据取走，否则前一帧数据将丢失。

S2BUF 以并行方式送往内部数据总线。

3.串行口 2 只能选择定时器 2 作为其波特率发生器 ---- 定时器 2 的寄存器 T2H，T2L

定时器 2 寄存器 T2H（地址为 D6H，复位值为 00H）及寄存器 T2L（地址为 D7H，复位值为 00H）用于保存重装时间常数。

【注意】：对于 STC15 系列单片机，

- 串口 2 永远是使用定时器 2 作为波特率发生器，不能够选择其他定时器作其波特率发生器；
- 串口 1 默认选择定时器 2 作为其波特率发生器，也可以选择定时器 1 作为其波特率发生器；
- 串口 3 默认选择定时器 2 作为其波特率发生器，也可以选择定时器 3 作为其波特率发生器；
- 串口 4 默认选择定时器 2 作为其波特率发生器，也可以选择定时器 4 作为其波特率发生器。

4.定时器 2 的控制位 ---- TR2、T2_C/T、T2x12

AUXR：辅助寄存器（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	name	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

T2R： 定时器 2 运行控制位

- 0，不允许定时器 2 运行;
- 1，允许定时器 2 运行

T2_C/T：控制定时器 2 用作定时器或计数器。

- 0，用作定时器（对内部系统时钟进行计数）;
- 1，用作计数器（对引脚 T2/P3.1 的外部脉冲进行计数）

T2x12： 定时器 2 速度控制位

- 0，定时器 2 是传统 8051 速度，12 分频;
- 1，定时器 2 的速度是传统 8051 的 12 倍，不分频

  如果串口 1 或串口 2 用 T2 作为波特率发生器,则由 T2x12 决定串口 1 或串口 2 是 12T 还是 1T。

对于 STC15 系列单片机，

- 串口 2 只能使用定时器 2 作为波特率发生器，不能够选择其他定时器作为其波特率发生器;
- 而串口 1 默认选择定时器 2 作为其波特率发生器，也可以选择定时器 1 作为其波特率发生器;
- 串口 3 默认选择定时器 2 作为其波特率发生器，也可以选择定时器 3 作为其波特率发生器;
- 串口 4 默认选择定时器 2 作为其波特率发生器，也可以选择定时器 4 作为其波特率发生器。

5. 与串行口 2 中断相关的寄存器

串行口 2 中断允许位 ES2 位于中断允许寄存器 IE2 中，中断允许寄存器的格式如下：

IE2：中断允许寄存器 2（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE2	AFH	name	-	-	-	-	-	-	ESPI	ES2

ES2： 串行口 2 中断允许位

- ES2=1，允许串行口 2 中断
- ES2=0，禁止串行口 2 中断。

IE：中断允许寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	name	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

EA： CPU 的总中断允许控制位

- EA=1，CPU 开放中断
- EA=0，CPU 屏蔽所有的中断申请

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。

串行口 2 中断优先级控制位 PS2 位位于中断优先级控制寄存器 IP 中，中断优先级控制寄存器的格式如下：

IP2：中断优先级控制寄存器（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IP2	B5H	name	-	-	-	-	-	-	PSPI	PS2

PS2： 串行口 2 中断优先级控制位

- 当 PS2=0 时，串行口 2 中断为最低优先级中断（优先级 0）
- 当 PS2=1 时，串行口 2 中断为最高优先级中断（优先级 1）

6. 串行口 2 在 2 组管脚之间切换的控制位 ---- S2_S/P_SW2.0

通过设置寄存器 P_SW2 中的 S2_S 位，可以将串口 2 在 2 组管脚之间任意切换，P_SW2 寄存器的格式如下：

P_SW2：外围设备功能切换控制寄存器 2（不可位寻址）

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
P_SW2	BAH	外围设备功能切换 控制寄存器 2						S4_S	S3_S	S2_S	xxxx,x000

串口 2/S2 可在 2 个地方切换，由 S2_S 控制位来选择	
S2_S	S2 可在 P1/P4 之间来回切换
0	串口 2/S2 在[P1.0/RxD2，P1.1/TxD2]
1	串口 2/S2 在[P4.6/RxD2_2，P4.7/TxD2_2]

串口 3/S3 可在 2 个地方切换，由 S3_S 控制位来选择	
S3_S	S3 可在 P0/P5 之间来回切换
0	串口 3/S3 在[P0.0/RxD3，P0.1/TxD3]
1	串口 3/S3 在[P5.0/RxD3_2，P5.1/TxD3_2]

串口 4/S4 可在 2 个地方切换，由 S4_S 控制位来选择	
S4_S	S4 可在 P0/P5 之间来回切换
0	串口 4/S4 在[P0.2/RxD4，P0.3/TxD4]
1	串口 4/S4 在[P5.2/RxD4_2，P5.3/TxD4_2]

19.6 串行口 2 工作模式

----串口 2 固定使用定时器 T2 作波特率发生器
----串口 1/3/4 和串口 2 的波特率相同时，串口 1/3/4 和串口 2 可共享 T2 作波特率发生器

STC15W4K32S4 系列单片机的串行口 2 有两种工作模式，可通过软件编程对 S2CON 中的 S2SM0 的设置进行选择。其中模式 0 和模式 1 都为异步通信，每个发送和接收的字符都带有 1 个启动位和 1 个停止位。

19.6.1 串行口 2 的工作模式 0 ---- 8 位 UART，波特率可变

10 位数据通过 RxD2/P1.0（RxD2_2/P4.6）接收，通过 TxD2/P1.1（TxD2_2/P4.7）发送。一帧数据包含一个起始位（0），8 个数据位和一个停止位（1）。接收时，停止位进入特殊功能寄存器 S2CON 的 S2RB8 位。波特率由定时器 T2 的溢出率决定。

串口 2 在模式 0 的波特率=定时器 T2 的溢出率/4

- 当 T2 工作在 1T 模式（AUXR.2/T2x12=1）时，定时器 2 的溢出率=SYSclk/（65536-[RL_TH2, RL_TL2]）；
  - ✓ 即此时，串行口 2 的波特率 = SYSclk /（65536 - [RL_TH2, RL_TL2]）/ 4
- 当 T2 工作在 12T 模式（AUXR.2/T2x12=0）时，定时器 2 的溢出率=SYSclk/12/（65536-[RL_TH2, RL_TL2]）；
  - ✓ 即此时，串行口 2 的波特率 = SYSclk / 12 /（65536 - [RL_TH2, RL_TL2]）/ 4

上式中 RL_TH2 是 T2H 的重装载寄存器，RL_TL2 是 T2L 的重装载寄存器。

19.6.2 串行口 2 的工作模式 1 ---- 9 位 UART，波特率可变

11 位数据通过 RxD2/P1.0（RxD2_2/P4.6）发送，通过 TxD2/P1.1（TxD2_2/P4.7）接收。一帧数据包含一个起始位（0），8 个数据位，一个可编程的第 9 位，和一个停止位（1）。发送时，第 9 位数据位来自特殊功能寄存器 S2CON 的 S2TB8 位；接收时，第 9 位进入特殊功能寄存器 S2CON 的 S2RB8 位。

串口 2 在模式 1 的波特率=T2 定时器 2 的溢出率/4

- 当 T2 工作在 1T 模式（AUXR.2/T2x12=1）时，定时器 2 的溢出率=SYSclk/（65536-[RL_TH2, RL_TL2]）；
  - ✓ 即此时，串行口 2 的波特率=SYSclk /（65536 -[RL_TH2, RL_TL2]）/ 4
- 当 T2 工作在 12T 模式（AUXR.2/T2x12=0）时，定时器 2 的溢出率=SYSclk /12/（65536-[RL_TH2, RL_TL2]）；
  - ✓ 即此时，串行口 2 的波特率=SYSclk / 12 /（65536 - [RL_TH2, RL_TL2]）/ 4

上式中 RL_TH2 是 T2H 的重装载寄存器，RL_TL2 是 T2L 的重装载寄存器。

可见，模式 1 和模式 0 一样，其波特率可通过软件对定时器 2 的设置进行波特率的选择，是可变的。

【说明】:

- 当串口 1、串口 3 及串口 4 和串口 2 的波特率相同时，串口 1、串口 3 及串口 4 和串口 2 可以共享定时器 T2 作波特率发生器，此时建议串口 1、串口 3 及串口 4 都选择定时器 T2 作为波特率发生器；
- 当串口 1、串口 3 及串口 4 和串口 2 的波特率不同时，串口 1、串口 3 及串口 4 和串口 2 不可以共享定时器 T2 作波特率发生器，这时才建议串口 1 选择定时器 T1 作波特率发生器，串口 3 选择定时器 T3 作波特率发生器，串口 4 选择定时器 T4 作波特率发生器。

用户在程序中如何具体使用串口 2：

1.设置串口 2 的工作模式，S2CON 寄存器中的 S2SM0 决定了串口 2 的 2 种工作模式
2.设置串口 2 的波特率相应的寄存器：定时器 2 寄存器 T2H / T2L
3.启动定时器 2，让 T2R 位为 1，定时器 2 就立即开始计数
4.设置 AUXR.2/T2x12，确定定时器 2 的速度
5.设置串口 2 的中断优先级，及打开中断相应的控制位是：PS2，PS2H，ES2，EA
6.如要串口 2 接收，将 S2REN 置 1 即可
如要串口 2 发送，将数据送入 S2BUF 即可，
接收完成标志 S2RI，发送完成标志 S2T，要由软件清 0。

19.7 串行口 2 的测试程序(C 和汇编)

----使用定时器 2 作串口 2 的波特率发生器

1.C 程序:

```
/*----STC15F2K60S2 系列定时器 2 用作串口 2 的波特率发生器举例-------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可----*/
//假定测试芯片的工作频率为 18.432MHz

#include "reg51.h"
#include "intrins.h"

typedef     unsigned char   BYTE;
typedef     unsigned int    WORD;

#define     FOSC            18432000L           //系统频率
#define     BAUD            115200              //串口波特率
#define     TM              (65536 - (FOSC/4/BAUD))

#define     NONE_PARITY     0                   //无校验
#define     ODD_PARITY      1                   //奇校验
#define     EVEN_PARITY     2                   //偶校验
#define     MARK_PARITY     3                   //标记校验
#define     SPACE_PARITY    4                   //空白校验

#define     PARITYBIT       EVEN_PARITY         //定义校验位

sfr         AUXR = 0x8e;                        //辅助寄存器
sfr         S2CON = 0x9a;                       //UART2  控制寄存器
sfr         S2BUF = 0x9b;                       //UART2  数据寄存器
sfr         T2H = 0xd6;                         //定时器 2 高 8 位
sfr         T2L = 0xd7;                         //定时器 2 低 8 位
sfr         IE2 = 0xaf;                         //中断控制寄存器 2
```

```
#define     S2RI    0x01            //S2CON.0
#define     S2TI    0x02            //S2CON.1
#define     S2RB8   0x04            //S2CON.2
#define     S2TB8   0x08            //S2CON.3

bit         busy;

void SendData(BYTE dat);
void SendString(char *s);
void main()
{
#if (PARITYBIT == NONE_PARITY)
    S2CON = 0x50;                   //8 位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
    S2CON = 0xda;                   //9 位可变波特率,校验位初始为 1
#elif (PARITYBIT == SPACE_PARITY)
    S2CON = 0xd2;                   //9 位可变波特率,校验位初始为 0
#endif

    T2L = TM;                       //设置波特率重装值
    T2H = TM>>8;
    AUXR = 0x14;                    //T2 为 1T 模式, 并启动定时器 2
    IE2 = 0x01;                     //使能串口 2 中断
    EA = 1;

    SendString("STC15F2K60S2\r\nUart2 Test !\r\n");
    while(1);
}
/*----------------------------UART2  中断服务程序-----------------------------*/
void Uart2() interrupt 8 using 1
{
    if (S2CON & S2RI)
    {
        S2CON &= ~S2RI;             //清除 S2RI 位
        P0 = S2BUF;                 //P0 显示串口数据
        P2 = (S2CON & S2RB8);       //P2.2 显示校验位
    }
    if (S2CON & S2TI)
    {
        S2CON &= ~S2TI;             //清除 S2TI 位
        busy = 0;                   //清忙标志
    }
}
/*----------------------------发送串口数据-----------------------------*/
void SendData(BYTE dat)
{
```

```
    while (busy);                                   //等待前面的数据发送完成
    ACC = dat;                                      //获取校验位 P (PSW.0)
    if (P)                                          //根据 P 来设置校验位
    {
#if (PARITYBIT == ODD_PARITY)
    S2CON &= ~S2TB8;                                //设置校验位为 0
#elif (PARITYBIT == EVEN_PARITY)
    S2CON |= S2TB8;                                 //设置校验位为 1
#endif
    }
    else
    {
#if (PARITYBIT == ODD_PARITY)
    S2CON |= S2TB8;                                 //设置校验位为 1
#elif (PARITYBIT == EVEN_PARITY)
    S2CON &= ~S2TB8;                                //设置校验位为 0
#endif
    }
    busy = 1;
    S2BUF = ACC;                                    //写数据到 UART2 数据寄存器
}
/*----------------------------发送字符串----------------------------*/
void SendString(char *s)
{
    while (*s)                                      //检测字符串结束标志
    {
        SendData(*s++);                             //发送当前字符
    }
}
```

2. 汇编程序：

```
/*----STC15F2K60S2 系列定时器 2 用作串口 2 的波特率发生器举例-------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可----*/
;假定测试芯片的工作频率为 18.432MHz
#define     NONE_PARITY     0               ;无校验
#define     ODD_PARITY      1               ;奇校验
#define     EVEN_PARITY     2               ;偶校验
#define     MARK_PARITY     3               ;标记校验
#define     SPACE_PARITY    4               ;空白校验

#define     PARITYBIT       EVEN_PARITY     ;定义校验位
;---------------------------------------

AUXR    EQU     08EH                        ;辅助寄存器
S2CON   EQU     09AH                        ;UART2  控制寄存器
```

```
S2BUF       EQU         09BH            ;UART2  数据寄存器
T2H         DATA        0D6H            ;定时器 2 高 8 位
T2L         DATA        0D7H            ;定时器 2 低 8 位
IE2         EQU         0AFH            ;中断控制寄存器 2

S2RI        EQU         01H             ;S2CON.0
S2TI        EQU         02H             ;S2CON.1
S2RB8       EQU         04H             ;S2CON.2
S2TB8       EQU         08H             ;S2CON.3
;-----------------------------------------
BUSY        BIT         20H.0           ;忙标志位
;-----------------------------------------
    ORG     0000H
    LJMP    START

    ORG     0043H
    LJMP    UART2_ISR
;-----------------------------------------
    ORG     0100H
START:
    CLR     BUSY
    CLR     EA
    MOV     SP, #3FH
#if (PARITYBIT == NONE_PARITY)
    MOV     S2CON, #50H                 ;8 位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
    MOV     S2CON, #0DAH                ;9 位可变波特率,校验位初始为 1
#elif (PARITYBIT == SPACE_PARITY)
    MOV     S2CON, #0D2H                ;9 位可变波特率,校验位初始为 0
#endif
;-------------------------------
    MOV     T2L, #0D8H                  ;设置波特率重装值(65536-18432000/4/115200)
    MOV     T2H, #0FFH
    MOV     AUXR, #14H                  ;T2 为 1T 模式, 并启动定时器 2
    ORL     IE2, #01H                   ;使能串口 2 中断
    SETB    EA

    MOV     DPTR, #TESTSTR              ;发送测试字符串
    LCALL   SENDSTRING

    SJMP    $
;-----------------------------------------
TESTSTR:
    DB      "STC15F2K60S2 Uart2 Test !",0DH,0AH,0
;/*--------------------------UART2  中断服务程序--------------------------*/
UART2_ISR:
```

```
    PUSH    ACC
    PUSH    PSW
    MOV     A, S2CON                ;读取 UART2 控制寄存器
    JNB     ACC.0, CHECKTI          ;检测 S2RI 位
    ANL     S2CON, #NOT S2RI        ;清除 S2RI 位
    MOV     P0, S2BUF               ;P0 显示串口数据
    ANL     A, #S2RB8 ;
    MOV     P2, A                   ;P2.2 显示校验位

CHECKTI:
    MOV     A, S2CON                ;读取 UART2 控制寄存器
    JNB     ACC.1, ISR_EXIT         ;检测 S2TI 位
    ANL     S2CON, #NOT S2TI        ;清除 S2TI 位
    CLR     BUSY                    ;清忙标志

ISR_EXIT:
    POP     PSW
    POP     ACC
    RETI
;/*----------------------------发送串口数据----------------------------*/
SENDDATA:
    JB      BUSY, $                 ;等待前面的数据发送完成
    MOV     ACC, A                  ;获取校验位 P (PSW.0)
    JNB     P, EVEN1INACC           ;根据 P 来设置校验位

ODD1INACC:
#if (PARITYBIT == ODD_PARITY)
    ANL     S2CON, #NOT S2TB8       ;设置校验位为 0
#elif (PARITYBIT == EVEN_PARITY)
    ORL     S2CON, #S2TB8           ;设置校验位为 1
#endif
    SJMP    PARITYBITOK

EVEN1INACC:
#if (PARITYBIT == ODD_PARITY)
    ORL     S2CON, #S2TB8           ;设置校验位为 1
#elif (PARITYBIT == EVEN_PARITY)
    ANL     S2CON, #NOT S2TB8       ;设置校验位为 0
#endif

PARITYBITOK:                        ;校验位设置完成
    SETB    BUSY
    MOV     S2BUF, A                ;写数据到 UART2 数据寄存器
    RET
;/*----------------------------发送字符串----------------------------*/
SENDSTRING:
```

```
    CLR     A
    MOVC    A, @A+DPTR                  ;读取字符
    JZ      STRINGEND                   ;检测字符串结束标志
    INC     DPTR                        ;字符串地址+1
    LCALL   SENDDATA                    ;发送当前字符
    SJMP    SENDSTRING

STRINGEND:
    RET
;------------------------------------------
END
```

19.8 串行口 3 的相关寄存器

符号	描述	地址	位地址及符号								复位值
			MSB LSB								
S3CON	串口 3 控制寄存器	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000B
S3BUF	串口 3 数据缓冲器	ADH									xxxx,xxxxB
T2H	定时器 2 高 8 位，装入重装数	D6H									0000,0000B
T2L	定时器 2 低 8 位，装入重装数	D7H									0000,0000B
AUXR	辅助寄存器	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000,0001B
T3H	定时器 3 高 8 位寄存器	D4H									0000,0000B
T3L	定时器 3 低 8 位寄存器	D5H									0000,0000B
T4T3M	T4 和 T3 的控制寄存器	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO	0000,0000B
IE2	中断允许寄存器	AFH	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000,0000B
P_SW2	外围设备功能切换控制寄存器	BAH	-	-	-	-	-	S4_S	S3_S	S2_S	xxxx,x000B

1. 串行口 3 的控制寄存器 S3CON

串行口 3 控制寄存器 S3CON 用于确定串行口 3 的工作方式和某些控制功能。其格式如下：

S3CON：串行口 3 控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
S3CON	ACH	name	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI

S3SM0：指定串行口 3 的工作方式，如下表所示：

S3SM0	工作方式	功能说明	波特率
0	方式 0	8 位 UART，波特率可变	（定时器 T2 的溢出率）/4 或 （定时器 T3 的溢出率）/4
1	方式 1	9 位 UART，波特率可变	（定时器 T2 的溢出率）/4 或 （定时器 T3 的溢出率）/4

- 当 AUXR.2/T2x12=1 时，定时器 T2 的溢出率=SYSclk /（65536 - [RL_TH2, RL_TL2]）
- 当 AUXR.2/T2x12=0 时，定时器 T2 的溢出率=SYSclk / 12 /（65536 - [RL_TH2, RL_TL2]）

式中 RL_TH2 是 T2H 的重装载寄存器，RL_TL2 是 T2L 的重装载寄存器。

- 当 T4T3M.1/T3x12=1 时，定时器 T3 的溢出率=SYSclk /（65536 - [RL_TH3, RL_TL3]）
- 当 T4T3M.1/T3x12=0 时，定时器 T3 的溢出率=SYSclk / 12 /（65536 - [RL_TH3, RL_TL3]）

式中 RL_TH3 是 T3H 的重装载寄存器，RL_TL3 是 T3L 的重装载寄存器。

S3ST3：串口 3（UART3）选择定时器 3 作波特率发生器的控制位。

- 0，串行口 3 选择定时器 2 作为其波特率发生器；
- 1，串行口 3 选择定时器 3 作为其波特率发生器

S3SM2：允许方式 1 多机通信控制位。

- 在方式 1 时，如果 S3SM2 位为 1 且 S3REN 位为 1，则接收机处于地址选状态。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；
  - ✓ 若 S3RB8=0，说明该帧不是地址帧，应丢掉且保持 S3RI=0。
- 在方式 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。

- 方式 0 是非多机通信方式，在这种方式时，要设置 S3SM2 应为 0。

S3REN：允许/禁止串行口 3 接收控制位。

- 由软件置位 S3REN，即 S3REN=1 为允许串行接收状态，可启动串行接收器 RxD3，开始接收信息。
- 软件复位 S3REN，即 S3REN=0，则禁止接收。

S3TB8：在方式 1，S3TB8 为要发送的第 9 位数据，按需要由软件置位或清 0。例如，可用作数据的校验位或多机通信中表示地址帧/数据帧的标志位。在方式 0 中，该位不用。

S3RB8：在方式 1，S3RB8 是接收到的第 9 位数据，作为奇偶校验位或地址帧/数据帧的标志位。方式 0 中不用 S3RB8（置 S3SM2=0，S3RB8 是接收到的停止位）。

S3TI：发送中断请求标志位。在停止位开始发送时由 S3TI 内部硬件置位，即 S3TI=1，响应中断后 S3TI 必须用软件清零。

S3RI：接收中断请求标志位。在串行接收到停止位的中间时刻 S3RI 由内部硬件置位，即 S3RI=1，向 CPU 发中断申请，响应中断后 S3RI 必须由软件清零。

S3CON 的所有位可通过整机复位信号复位为全“0”。S3CON 的字节地址为 ACH，不可位寻址。

串行通信的中断请求：

- 当一帧发送完成，内部硬件自动置位 S3TI，即 S3TI=1，请求中断处理；
- 当接收完一帧信息时，内部硬件自动置位 S3RI，即 S3RI=1，请求中断处理。

由于 S3TI 和 S3RI 以“或逻辑”关系向主机请求中断，所以主机响应中断时事先并不知道是 S3TI 还是 S3RI 请求的中断，必须在中断服务程序中查询 S3TI 和 S3RI 进行判别，然后分别处理。因此，两个中断请求标志位均不能由硬件自动置位，必须通过软件清 0，否则将出现一次请求多次响应的错误。

2.串行口 3 的数据缓冲寄存器 S3BUF

STC15W4K32S4 系列单片机的串行口 3 数据缓冲寄存器（S3BUF）的地址是 ADH，实际是 2 个缓冲器，写 S3BUF 的操作完成待发送数据的加载，读 S3BUF 的操作可获得已接收到的数据。两个操作分别对应两个不同的寄存器，1 个是只写寄存器，1 个是只读寄存器。

串行通道内设有数据寄存器。在所有的串行通信方式中，在写入 S3BUF 信号（MOV S3BUF, A）的控制下，把数据装入相同的 9 位移位寄存器，前面 8 位为数据字节，其最低位为移位寄存器的输出位。根据不同的工作方式会自动将“1”或 S3TB8 的值装入移位寄存器的第 9 位，并进行发送。

串行通道的接收寄存器是一个输入移位寄存器。在方式 0 和方式 1 时均为 9 位。当一帧接收完毕，移位寄存器中的数据字节装入串行数据缓冲器 S3BUF 中，其第 9 位则装入 S3CON 寄存器中的 S3RB8 位。如果由于 S3SM2 使得已接收到的数据无效时，S3RB8 和 S3BUF 中内容不变。

由于接收通道内设有输入移位寄存器和 S3BUF 缓冲器，从而能使一帧接收完将数据由移位寄存器装入 S3BUF 后，可立即开始接收下一帧信息，主机应在该帧接收结束前从 S3BUF 缓冲器中将数据取走，否则前一帧数据将丢失。

S3BUF 以并行方式送往内部数据总线。

3.串行口 3 既能选择定时器 2 作为其波特率发生器，也能选择定时器 3 作为其波特率发生器

----定时器 2 的寄存器 T2H，T2L 和定时器 3 的寄存器 T3H，T3L

- 定时器 2 寄存器 T2H（地址为 D6H，复位值为 00H）及寄存器 T2L（地址为 D7H，复位值为 00H）用于保存重装时间常数。
- 定时器 3 寄存器 T3H（地址为 D4H，复位值为 00H）及寄存器 T3L（地址为 D5H，复位值为 00H）用于保存重装时间常数。

【注意】:

- 有串口 2 的单片机，串口 2 永远是使用定时器 2 作为波特率发生器，不能够选择定时器 1 做波特率发生器；
- 串口 1 可以选择定时器 1 做波特率发生器，也可以选择定时器 2 作为波特率发生器；
- 串口 3 可以选择定时器 2 做波特率发生器，也可以选择定时器 3 作为波特率发生器；
- 串口 4 可以选择定时器 2 做波特率发生器，也可以选择定时器 4 作为波特率发生器。

4. 定时器 2 的控制位——TR2、T2_C/T、T2x12

AUXR：辅助寄存器 （不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	name	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

T2R： 定时器 2 运行控制位

- 0，不允许定时器 2 运行；
- 1，允许定时器 2 运行

T2_C/T：控制定时器 2 用作定时器或计数器。

- 0，用作定时器（对内部系统时钟进行计数）；
- 1，用作计数器（对引脚 T2/P3.1 的外部脉冲进行计数）

T2x12：定时器 2 速度控制位

- 0，定时器 2 是传统 8051 速度，12 分频；
- 1，定时器 2 的速度是传统 8051 的 12 倍，不分频

如果串口 1 或串口 2 用 T2 作为波特率发生器，则由 T2x12 决定串口 1 或串口 2 是 12T 还是 1T。

5.定时器 3 的控制位 ---- TR3、T3_C/T、T3x12

T4T3M（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	name	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO

B3-T3R：定时器 3 运行控制位。

- 0，不允许定时器 3 运行；
- 1，允许定时器 3 运行。

B2-T3_C/T：控制定时器 3 用作定时器或计数器。

- 0，用作定时器（对内部系统时钟进行计数）；
- 1，用作计数器（对引脚 T3/P0.5 的外部脉冲进行计数）

B1-T3x12：定时器 3 速度控制位。

- 0，定时器 3 速度是 8051 单片机定时器的速度，即 12 分频；
- 1，定时器 3 速度是 8051 单片机定时器速度的 12 倍，即不分频。

6. 与串行口 3 中断相关的寄存器 IE2

串行口 3 中断允许位 ES3 位于中断允许寄存器 IE2 中，中断允许寄存器的格式如下：

IE2：中断允许寄存器 2（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE2	AFH	name	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2

ET4： 定时器 4 的中断允许位。

- 1，允许定时器 4 产生中断
- 0，禁止定时器 4 产生中断

ET3： 定时器 3 的中断允许位。

- 1，允许定时器 3 产生中断
- 0，禁止定时器 3 产生中断

ES4： 串行口 4 中断允许位。

- 1，允许串行口 4 中断
- 0，禁止串行口 4 中断

ES3： 串行口 3 中断允许位。

- 1，允许串行口 3 中断
- 0，禁止串行口 3 中断

ET2： 定时器 2 的中断允许位。

- 1，允许定时器 2 产生中断
- 0，禁止定时器 2 产生中断

ESPI： SPI 中断允许位。

- 1，允许 SPI 中断
- 0，禁止 SPI 中断

ES2： 串行口 2 中断允许位。

- 1，允许串行口 2 中断
- 0，禁止串行口 2 中断

IE：中断允许寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	name	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

EA：CPU 的总中断允许控制位

- 1，CPU 开放中断
- 0，CPU 屏蔽所有的中断申请。

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。

7.串行口 3 在 2 组管脚之间切换的控制位 ---- S3_S/P_SW2.1

通过设置寄存器 P_SW2 中的 S3_S 位，可以将串口 3 在 2 组管脚之间任意切换，P_SW2 寄存器的格式如下：

P_SW2：外围设备切换控制寄存器 2（不可位寻址）

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
P_SW2	BAH	外围设备功能切换控制寄存器 2						S4_S	S3_S	S2_S	xxxx,x000

串口 2/S2 可在 2 个地方切换，由 S2_S 控制位来选择	
S2_S	S2 可在 P1/P4 之间来回切换
0	串口 2/S2 在[P1.0/RxD2，P1.1/TxD2]
1	串口 2/S2 在[P4.6/RxD2_2，P4.7/TxD2_2]

串口 3/S3 可在 2 个地方切换，由 S3_S 控制位来选择	
S3_S	S3 可在 P0/P5 之间来回切换
0	串口 3/S3 在[P0.0/RxD3，P0.1/TxD3]
1	串口 3/S3 在[P5.0/RxD3_2，P5.1/TxD3_2]

串口 4/S4 可在 2 个地方切换，由 S4_S 控制位来选择	
S4_S	S4 可在 P0/P5 之间来回切换
0	串口 4/S4 在[P0.2/RxD4，P0.3/TxD4]
1	串口 4/S4 在[P5.2/RxD4_2，P5.3/TxD4_2]

19.9 串行口 3 工作模式

---串口 3 和串口 2 的波特率相同时，串口 3 和串口 2 可共享 T2 作波特率发生器

STC15W4K32S4 系列单片机的串行口 3 有两种工作模式，可通过软件编程对 S3CON 中的 S3SM0 的设置进行选择。其中模式 0 和模式 1 都为异步通信，每个发送和接收的字符都带有 1 个启动位和 1 个停止位。

19.9.1 串行口 3 的工作模式 0----8 位 UART，波特率可变

10 位数据通过 RxD3/P0.0（RxD3/P5.0）接收，通过 TxD3/P0.1（TxD3/P5.1）发送。一帧数据包含一个起始位（0），8 个数据位和一个停止位（1）。接收时，停止位进入特殊功能寄存器 S3CON 的 S3RB8 位。

串行口 3 既可以选择定时器 2 作其波特率发生器，也可以选择定时器 3 作其波特率发生器。所以串行口 3 的波特率由定时器 T2 的溢出率或定时器 T3 的溢出率决定。

- 当串行口 3 选择定时器 T2 作为其波特率发生器（即 S3ST3/S3SCON.1=0）时：

  串口 3 在模式 0 的波特率 = 定时器 T2 的溢出率 / 4

  - ✓ 当 T2 工作在 1T 模式（AUXR.2/T2x12=1）时，定时器 2 的溢出率=SYSclk/（65536-[RL_TH2, RL_TL2]）；
    - 即此时，串行口 3 的波特率=SYSclk /（65536 - [RL_TH2, RL_TL2]）/4
  - ✓ 当 T2 工作在 12T 模式（AUXR.2/T2x12=0）时，定时器 2 的溢出率=SYSclk /12/（65536-[RL_TH2, RL_TL2]）；
    - 即此时，串行口 3 的波特率=SYSclk / 12 /（65536 - [RL_TH2, RL_TL2]）/4

  上式中 RL_TH2 是 T2H 的重装载寄存器，RL_TL2 是 T2L 的重装载寄存器。

- 当串行口 3 选择定时器 T3 作为其波特率发生器（即 S3ST3/S3SCON.1=1）时：

  串口 3 在模式 0 的波特率 = 定时器 T3 的溢出率 / 4

  - ✓ 当 T3 工作在 1T 模式（T4T3M.1/T3x12=1）时，定时器 3 的溢出率=SYSclk/（65536-[RL_TH3, RL_TL3]）；
    - 即此时，串行口 3 的波特率=SYSclk /（65536 - [RL_TH3, RL_TL3]）/4
  - ✓ 当 T3 工作在 12T 模式（T4T3M.1/T3x12=0）时，定时器 3 的溢出率=SYSclk /12/（65536-[RL_TH3, RL_TL3]）；
    - 即此时，串行口 3 的波特率=SYSclk / 12 /（65536 - [RL_TH3, RL_TL3]）/4

  上式中 RL_TH3 是 T3H 的重装载寄存器，RL_TL3 是 T3L 的重装载寄存器。

【说明】:

- 当串口 3 和串口 2 的波特率相同时，串口 3 和串口 2 可以共享波特率发生器，此时建议用户选择定时器 T2 作为串口 3 的波特率发生器；
- 当串口 3 和串口 2 的波特率不同时，才建议选择定时器 T3 作为串口 3 的波特率发生器（因串口 2 固定使用定时器 T2 作波特率发生器）。

19.9.2 串行口 3 的工作模式 1----9 位 UART，波特率可变

11 位数据通过 TxD3/P0.1（TxD3/P5.1）发送，通过 RxD3/P0.0（RxD3/P5.0）接收。一帧数据包含一个起始位（0），8 个数据位，一个可编程的第 9 位，和一个停止位（1）。发送时，第 9 位数据位来自特殊功能寄存器 S3CON 的 S3TB8 位；接收时，第 9 位进入特殊功能寄存器 S3CON 的 S3RB8 位。

串行口 3 既可以选择定时器 2 作其波特率发生器，也可以选择定时器 3 作其波特率发生器。所以串行口 3 的波特率由定时器 T2 的溢出率或定时器 T3 的溢出率决定。

- 当串行口 3 选择定时器 T2 作为其波特率发生器（即 S3ST3/S3SCON.1=0）时：

  串口 3 在模式 1 的波特率=定时器 T2 的溢出率/4

  - ✓ 当 T2 工作在 1T 模式（AUXR.2/T2x12=1）时，定时器 2 的溢出率=SYSclk/（65536-RL_TH2, RL_TL2]）；
    - 即此时，串行口 3 的波特率=SYSclk /（65536 - [RL_TH2, RL_TL2]）/4
  - ✓ 当 T2 工作在 12T 模式（AUXR.2/T2x12=0）时，定时器 2 的溢出率=SYSclk /12/（65536-[RL_TH2, RL_TL2]）；
    - 即此时，串行口 3 的波特率=SYSclk / 12 /（65536 - [RL_TH2, RL_TL2]）/4

  上式中 RL_TH2 是 T2H 的重装载寄存器，RL_TL2 是 T2L 的重装载寄存器。

- 当串行口 3 选择定时器 T3 作为其波特率发生器（即 S3ST3/S3SCON.1=1）时：

  串口 3 在模式 1 的波特率=定时器 T3 的溢出率/4

  - ✓ 当 T3 工作在 1T 模式（T4T3M.1/T3x12=1）时，定时器 3 的溢出率=SYSclk/（65536-「RL_TH3, RL_TL3]）；
    - 即此时，串行口 3 的波特率=SYSclk /（65536 - [RL_TH3, RL_TL3]）/4
  - ✓ 当 T3 工作在 12T 模式（T4T3M.1/T3x12=0）时，定时器 3 的溢出率=SYSclk /12/（65536-[RL_TH3,RL_TL3]）；
    - 即此时，串行口 3 的波特率=SYSclk / 12 /（65536 - [RL_TH3, RL_TL3]）/4

  上式中 RL_TH3 是 T3H 的重装载寄存器，RL_TL3 是 T3L 的重装载寄存器。

可见，模式 1 和模式 0 一样，其波特率可通过软件对定时器 2 或定时器 3 的设置进行波特率的选择，是可变的。

【说明】:

- 当串口 3 和串口 2 的波特率相同时，串口 3 和串口 2 可以共享波特率发生器，此时建议用户选择定时器 T2 作为串口 3 的波特率发生器；
- 当串口 3 和串口 2 的波特率不同时，才建议选择定时器 T3 作为串口 3 的波特率发生器（因串口 2 固定使用定时器 T2 作波特率发生器）。

用户在程序中如何具体使用串口 3：

1.设置串口 3 的工作模式，S3CON 寄存器中的 S3SM0 决定了串口 3 的 2 种工作模式
2.设置串口 3 的波特率相应的寄存器：定时器 3 寄存器 T3H/T3L
3.启动定时器 3，让 T3R 位为 1，定时器 3 就立即开始计数。
4.设置 T4T3M.1/T3x12，确定定时器 3 的速度
5.打开串口 3 中断，设置相应的控制位是：ES3，EA
6.如要串口 3 接收，将 S3REN 置 1 即可
如要串口 3 发送，将数据送入 S3BUF 即可，
接收完成标志 S3RI，发送完成标志 S3TI，要由软件清 0。

19.10 串行口 4 的相关寄存器

符号	描述	地址	位地址及符号								复位值
			MSB							LSB	
S4CON	串口 4 控制寄存器	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000,0000B
S4BUF	串口 4 数据缓冲器	85H									xxxx,xxxxB
T2H	定时器 2 高 8 位，装入重装数	D6H									0000,0000B
T2L	定时器 2 低 8 位，装入重装数	D7H									0000,0000B
AUXR	辅助寄存器	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000,0001B
T4H	定时器 4 高 8 位寄存器	D2H									0000,0000B
T4L	定时器 4 低 8 位寄存器	D3H									0000,0000B
T4T3M	T4 和 T3 的控制寄存器	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO	0000,0000B
IE2	中断允许寄存器	AFH	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000,0000B
P_SW2	外围设备功能切换控制寄存器	BAH	-	-	-	-	-	S4_S	S3_S	S2_S	xxxx,x000B

1. 串行口 4 的控制寄存器 S4CON

串行口 4 控制寄存器 S4CON 用于确定串行口 4 的工作方式和某些控制功能。其格式如下：

S4CON：串行口 4 控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
S4CON	84H	name	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI

S4SM0：指定串行口 4 的工作方式，如下表所示：

S4SM0	工作方式	功能说明	波特率
0	方式 0	8 位 UART，波特率可变	（定时器 T4 的溢出率）/4
1	方式 1	9 位 UART，波特率可变	（定时器 T4 的溢出率）/4

- 当 T4T3M.5/T4x12=1 时，定时器 T4 的溢出率=SYSclk /（65536 - [RL_TH4, RL_TL4]）
- 当 T4T3M.5/T4x12=0 时，定时器 T4 的溢出率=SYSclk / 12 /（65536 - [RL_TH4, RL_TL4]）

式中 RL_TH4 是 T4H 的重装载寄存器，RL_TL4 是 T4L 的重装载寄存器。

S4ST4：串口 4（UART4）选择定时器 4 作波特率发生器的控制位。

- 0，串行口 4 选择定时器 2 作为其波特率发生器
- 1，串行口 4 选择定时器 4 作为其波特率发生器

S4SM2：允许方式 1 多机通信控制位。

- 在方式 1 时，如果 S4SM2 位为 1 且 S4REN 位为 1，则接收机处于地址选状态。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。
- 在方式 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。
- 方式 0 是非多机通信方式，在这种方式时，要设置 S4SM2 应为 0。

S4REN：允许/禁止串行口 4 接收控制位。

- 由软件置位 S4REN，即 S4REN=1 为允许串行接收状态，可启动串行接收器 RxD4，开始接收信息。
- 软件复位 S4REN，即 S4REN=0，则禁止接收。

S4TB8： 在方式 1，S4TB8 为要发送的第 9 位数据，按需要由软件置位或清 0。例如，可用作数据的校验位或多机通信中表示地址帧/数据帧的标志位。在方式 0 中，该位不用。

S4RB8： 在方式 1，S4RB8 是接收到的第 9 位数据，作为奇偶校验位或地址帧/数据帧的标志位。方式 0 中不用 S4RB8（置 S4SM2=0，S4RB8 是接收到的停止位）。

S4TI： 发送中断请求标志位。在停止位开始发送时由 S4TI 内部硬件置位，即 S4TI=1，响应中断后 S4TI 必须用软件清零。

S4RI： 接收中断请求标志位。在串行接收到停止位的中间时刻 S4RI 由内部硬件置位，即 S4RI=1,向 CPU 发中断申请，响应中断后 S4RI 必须由软件清零。

S4CON 的所有位可通过整机复位信号复位为全“0”。S4CON 的字节地址为 84H，不可位寻址。
串行通信的中断请求：

- 当一帧发送完成，内部硬件自动置位 S4TI，即 S4TI=1，请求中断处理；
- 当接收完一帧信息时，内部硬件自动置位 S4RI，即 S4RI=1，请求中断处理。

由于 S4TI 和 S4RI 以“或逻辑”关系向主机请求中断，所以主机响应中断时事先并不知道是 S4TI 还是 S4RI 请求的中断，必须在中断服务程序中查询 S4TI 和 S4RI 进行判别，然后分别处理。因此，两个中断请求标志位均不能由硬件自动置位，必须通过软件清 0，否则将出现一次请求多次响应的错误。

2. 串行口 4 的数据缓冲寄存器 S4BUF

STC15W4K32S4 系列单片机的串行口 4 数据缓冲寄存器（S4BUF）的地址是 85H，实际是 2 个缓冲器，写 S4BUF 的操作完成待发送数据的加载，读 S4BUF 的操作可获得已接收到的数据。两个操作分别对应两个不同的寄存器，1 个是只写寄存器，1 个是只读寄存器。

串行通道内设有数据寄存器。在所有的串行通信方式中，在写入 S4BUF 信号（MOV S4BUF, A）的控制下，把数据装入相同的 9 位移位寄存器，前面 8 位为数据字节，其最低位为移位寄存器的输出位。根据不同的工作方式会自动将“1”或 S4TB8 的值装入移位寄存器的第 9 位，并进行发送。

串行通道的接收寄存器是一个输入移位寄存器。在方式 0 和方式 1 时均为 9 位。当一帧接收完毕，移位寄存器中的数据字节装入串行数据缓冲器 S4BUF 中，其第 9 位则装入 S4CON 寄存器中的 S4RB8 位。如果由于 S4SM2 使得已接收到的数据无效时，S4RB8 和 S4BUF 中内容不变。

由于接收通道内设有输入移位寄存器和 S4BUF 缓冲器，从而能使一帧接收完将数据由移位寄存器装入 S4BUF 后，可立即开始接收下一帧信息，主机应在该帧接收结束前从 S4BUF 缓冲器中将数据取走，否则前一帧数据将丢失。

S4BUF 以并行方式送往内部数据总线。

3.串行口 4 既能选择定时器 2 作为其波特率发生器，也能选择定时器 4 作为其波特率发生器

----定时器 2 的寄存器 T2H，T2L 和定时器 4 的寄存器 T4H，T4L

- 定时器 2 寄存器 T2H（地址为 D6H，复位值为 00H）及寄存器 T2L（地址为 D7H，复位值为 00H）用于保存重装时间常数。

- 定时器 4 寄存器 T4H（地址为 D2H，复位值为 00H）及寄存器 T4L（地址为 D3H，复位值为 00H）用于保存重装时间常数。

4. 定时器 2 的控制位——TR2、T2_C/T、T2x12

AUXR：辅助寄存器 （不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	name	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

T2R： 定时器 2 允许控制位

- 0，不允许定时器 2 运行；
- 1，允许定时器 2 运行

T2_C/T：控制定时器 2 用作定时器或计数器。

- 0，用作定时器（对内部系统时钟进行计数）；
- 1，用作计数器（对引脚 T2/P3.1 的外部脉冲进行计数）

T2x12： 定时器 2 速度控制位

- 0，定时器 2 是传统 8051 速度，12 分频；
- 1，定时器 2 的速度是传统 8051 的 12 倍，不分频

如果串口 1 或串口 2 用 T2 作为波特率发生器，则由 T2x12 决定串口 1 或串口 2 是 12T 还是 1T.

5. 定时器 4 的控制位--TR4、T4_C/T、T4x12

T4T3M（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	name	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO

B7-T4R：定时器 4 运行控制位

- 0，不允许定时器 4 运行
- 1，允许定时器 4 运行

B6-T4_C/T：控制定时器 4 用作定时器或计数器

- 0，用作定时器（对内部系统时钟进行计数）
- 1，用作计数器（对引脚 T4/P0.7 的外部脉冲进行计数）

B5-T4x12：定时器 4 速度控制位

- 0，定时器 4 速度是 8051 单片机定时器的速度，即 12 分频
- 1，定时器 4 速度是 8051 单片机定时器速度的 12 倍，即不分频

6. 与串行口 4 中断相关的寄存器 IE2

串行口 4 中断允许位 ES4 位于中断允许寄存器 IE2 中，中断允许寄存器的格式如下：

IE2：中断允许寄存器 2（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE2	AFH	name	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2

ET4： 定时器 4 的中断允许位
- 1，允许定时器 4 产生中断
- 0，禁止定时器 4 产生中断

ET3： 定时器 3 的中断允许位
- 1，允许定时器 3 产生中断
- 0，禁止定时器 3 产生中断

ES4： 串行口 4 中断允许位
- 1，允许串行口 4 中断
- 0，禁止串行口 4 中断

ES3： 串行口 3 中断允许位
- 1，允许串行口 3 中断
- 0，禁止串行口 3 中断

ET2： 定时器 2 的中断允许位
- 1，允许定时器 2 产生中断
- 0，禁止定时器 2 产生中断。

ESPI： SPI 中断允许位
- 1，允许 SPI 中断
- 0，禁止 SPI 中断

ES2： 串行口 2 中断允许位
- 1，允许串行口 2 中断
- 0，禁止串行口 2 中断

IE：中断允许寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	name	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

EA：CPU 的总中断允许控制位
- EA=1，CPU 开放中断；
- EA=0，CPU 屏蔽所有的中断申请。

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。

7. 串行口 4 在 2 组管脚之间切换的控制位 ---- S4_S/P_SW2.2

通过设置寄存器 P_SW2 中的 S4_S 位，可以将串口 4 在 2 组管脚之间任意切换，P_SW2 寄存器的格式如下：

P_SW2：外围设备切换控制寄存器 2（不可位寻址）

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
P_SW2	BAH	外围设备功能切换控制寄存器 2						S4_S	S3_S	S2_S	xxxx,x000

串口 2/S2 可在 2 个地方切换，由 S2_S 控制位来选择	
S2_S	S2 可在 P1/P4 之间来回切换
0	串口 2/S2 在[P1.0/RxD2，P1.1/TxD2]
1	串口 2/S2 在[P4.6/RxD2_2，P4.7/TxD2_2]

串口 3/S3 可在 2 个地方切换，由 S3_S 控制位来选择	
S3_S	S3 可在 P0/P5 之间来回切换
0	串口 3/S3 在[P0.0/RxD3，P0.1/TxD3]
1	串口 3/S3 在[P5.0/RxD3_2，P5.1/TxD3_2]

串口 4/S4 可在 2 个地方切换，由 S4_S 控制位来选择	
S4_S	S4 可在 P0/P5 之间来回切换
0	串口 4/S4 在[P0.2/RxD4，P0.3/TxD4]
1	串口 4/S4 在[P5.2/RxD4_2，P5.3/TxD4_2]

19.11 串行口 4 工作模式

----串口 1 和串口 2 的波特率相同时，串口 1 和串口 2 可共享 T2 作波特率发生器

STC15W4K32S4 系列单片机的串行口 4 有两种工作模式，可通过软件编程对 S4CON 中的 S4SM0 的设置进行选择。其中模式 0 和模式 1 都为异步通信，每个发送和接收的字符都带有 1 个启动位和 1 个停止位。

19.11.1 串行口 4 的工作模式 0 ---- 8 位 UART，波特率可变

10 位数据通过 RxD4/P0.2（RxD4_2/P5.2）接收，通过 TxD4/P0.3（TxD4_2/P5.3）发送。一帧数据包含一个起始位（0），8 个数据位和一个停止位（1）。接收时，停止位进入特殊功能寄存器 S4CON 的 S4RB8 位。

串行口 4 既可以选择定时器 2 作其波特率发生器，也可以选择定时器 4 作其波特率发生器。所以串行口 4 的波特率由定时器 T2 的溢出率或定时器 T4 的溢出率决定。

- 当串行口 4 选择定时器 T2 作为其波特率发生器（即 S4ST4/S4SCON.1=0）时：

  串口 4 在模式 0 的波特率=定时器 T2 的溢出率/4

  - ✓ 当 T2 工作在 1T 模式（AUXR.2/T2x12=1）时，定时器 2 的溢出率=SYSclk/（65536-[RL_TH2, RL_TL2]）；
    - 即此时，串行口 4 的波特率 = SYSclk /（65536 - [RL_TH2, RL_TL2]）/ 4
  - ✓ 当 T2 工作在 12T 模式（AUXR.2/T2x12=0）时，定时器 2 的溢出率=SYSclk/12/（65536-RL_TH2, RL_TL2]）；
    - 即此时，串行口 4 的波特率 = SYSclk / 12 /（65536 - [RL_TH2, RL_TL2]）/ 4

  上式中 RL_TH2 是 T2H 的重装载寄存器，RL_TL2 是 T2L 的重装载寄存器。

- 当串行口 4 选择定时器 T4 作为其波特率发生器（即 S4ST4/S4SCON.1=1）时：

  串口 4 在模式 0 的波特率=定时器 T4 的溢出率/4

  - ✓ 当 T4 工作在 1T 模式（T4T3M.5/T4x12=1）时，定时器 4 的溢出率=SYSclk/（65536-[RL_TH4, RL_TL4]）；
    - 即此时，串行口 4 的波特率=SYSclk /（65536 - [RL_TH4, RL_TL4]）/ 4
  - ✓ 当 T4 工作在 12T 模式（T4T3M.5/T4x12=0）时，定时器 4 的溢出率=SYSclk /12/（65536-[RL_TH4, RL_TL4]）；
    - 即此时，串行口 4 的波特率=SYSclk / 12 /（65536 - [RL_TH4, RL_TL4]）/ 4

  上式中 RL_TH4 是 T4H 的重装载寄存器，RL_TL4 是 T4L 的重装载寄存器。

【说明】:

- 当串口 4 和串口 2 的波特率相同时，串口 4 和串口 2 可以共享波特率发生器，此时建议用户选择定时器 T2 作为串口 4 的波特率发生器；
- 当串口 4 和串口 2 的波特率不同时，才建议选择定时器 T4 作为串口 4 的波特率发生器（因串口 2 固定使用定时器 T2 作波特率发生器）。

19.11.2 串行口 4 的工作模式 1 ---- 9 位 UART，波特率可变

11 位数据通过 TxD4/P0.3（TxD4_2/P5.3）发送，通过 RxD4/P0.2（RxD4_2/P5.2）接收。一帧数据包含一个起始位（0），8 个数据位，一个可编程的第 9 位，和一个停止位（1）。

- 发送时，第 9 位数据位来自特殊功能寄存器 S4CON 的 S4TB8 位；
- 接收时，第 9 位进入特殊功能寄存器 S4CON 的 S4RB8 位。

串行口 4 既可以选择定时器 2 作其波特率发生器，也可以选择定时器 4 作其波特率发生器。所以串行口 4 的波特率由定时器 T2 的溢出率或定时器 T4 的溢出率决定。

- 当串行口 4 选择定时器 T2 作为其波特率发生器（即 S4ST4/S4SCON.1=0）时：

串口 4 在模式 1 的波特率=定时器 T2 的溢出率/4

- ✓ 当 T2 工作在 1T 模式（AUXR.2/T2x12=1）时，定时器 2 的溢出率=SYSclk/（65536-[RL_TH2, RL_TL2]）；
  - 即此时，串行口 4 的波特率 = SYSclk /（65536 - [RL_TH2, RL_TL2]）/ 4
- ✓ 当 T2 工作在 12T 模式（AUXR.2/T2x12=0）时，定时器 2 的溢出率=SYSclk /12/（65536-[RL_TH2, RL_TL2]）；
  - 即此时，串行口 4 的波特率 = SYSclk / 12 /（65536 - [RL_TH2, RL_TL2]）/ 4

上式中 RL_TH2 是 T2H 的重装载寄存器，RL_TL2 是 T2L 的重装载寄存器。

- 当串行口 4 选择定时器 T4 作为其波特率发生器（即 S4ST4/S4SCON.1=1）时：

串口 4 在模式 1 的波特率=定时器 T4 的溢出率/4

- ✓ 当 T4 工作在 1T 模式（T4T3M.5/T4x12=1）时，定时器 4 的溢出率=SYSclk/（65536-[RL_TH4, RL_TL4]）；
  - 即此时，串行口 4 的波特率 = SYSclk /（65536 - [RL_TH4, RL_TL4]）/ 4
- ✓ 当 T4 工作在 12T 模式（T4T3M.5/T4x12=0）时，定时器 4 的溢出率=SYSclk /12/（65536-[RL_TH4, RL_TL4]）；
  - 即此时，串行口 4 的波特率 = SYSclk /12 /（65536 - [RL_TH4, RL_TL4]）/ 4

上式中 RL_TH4 是 T4H 的重装载寄存器，RL_TL4 是 T4L 的重装载寄存器。

可见，模式 1 和模式 0 一样，其波特率可通过软件对定时器 2 的设置进行波特率的选择，是可变的。

【说明】:

- 当串口 4 和串口 2 的波特率相同时，串口 4 和串口 2 可以共享波特率发生器，此时建议用户选择定时器 T2 作为串口 4 的波特率发生器；
- 当串口 4 和串口 2 的波特率不同时，才建议选择定时器 T4 作为串口 4 的波特率发生器（因串口 2 固定使用定时器 T2 作波特率发生器）。

用户在程序中如何具体使用串口 4：

1.设置串口 4 的工作模式，S4CON 寄存器中的 S4SM0 决定了串口 4 的 2 种工作模式
2.设置串口 4 的波特率相应的寄存器：定时器 4 寄存器 T4H / T4L
3.启动定时器 4，让 T4R 位为 1，定时器 4 就立即开始计数
4.设置 T4T3M.5/T4x12，确定定时器 4 的速度

5.打开串口 4 中断，设置相应的控制位是：ES4，EA

6.如要串口 4 接收，将 S4REN 置 1 即可

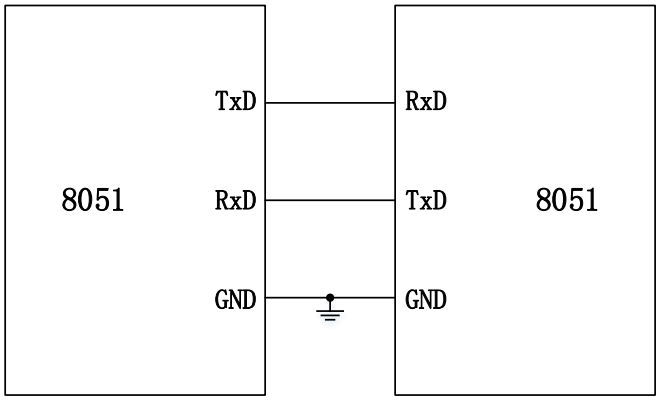
如要串口 4 发送，将数据送入 S4BUF 即可

接收完成标志 S4RI，发送完成标志 S4T1，要由软件清 0。

19.12 双机通信

STC15 系列单片机的串行通信根据其应用可分为双机通信和多机通信两种。下面先介绍双机通信。

如果两个 8051 应用系统相距很近，可将它们的串行端口直接相连（TXD-RXD，RXD-TXD，GND-GND-地），即可实现双机通信。为了增加通信距离，减少通道及电源干扰，可采用 RS-232C 或 RS-422、RS-485 标准进行双机通信，两通信系统之间采用光一电隔离技术，以减少通道及电源的干扰，提高通信可靠性。



为确保通信成功，通信双方必须在软件上有系列的约定通常称为软件通信“协议”。现举例简介双机异步通信软件“协议”如下：

通信双方均选用 2400 波特的传输速率，设系统的主频 SYSclk=6MHz，甲机发送数据，乙机接收数据。在双机开始通信时，先由甲机发送一个呼叫信号（例如“06H”），以询问乙机是否可以接收数据；乙机接收到呼叫信号后，若同意接收数据，则发回“00H”作为应答信号，否则发“05H”表示暂不能接收数据；甲机只有在接收到乙机的应答信号“00H”后才可将存储在外部数据存储器中的内容逐一发送给乙机，否则继续向乙机发呼叫信号，直到乙机同意接收。其发送数据格式如下：

字节数 n	数据 1	数据 2	数据 3	…	数据 n	累加校验和

字节数 n： 甲机向乙机发送的数据个数；

数据 1～ 数据 n： 甲机将向乙机发送的 n 帧数据；

累加校验和： 为字节数 n、数据 1、…、数据 n，这（n+1）个字节内容的算术累加和

乙机根据接收到的“校验和”判断已接收到的 n 个数据是否正确。若接收正确，向甲机回发“0FH”信号，否则回发“F0H”信号。甲机只有在接收到乙机发回的“0FH”信号才算完成发送任务，返回被调用的程序，否则继续呼叫，重发数据。

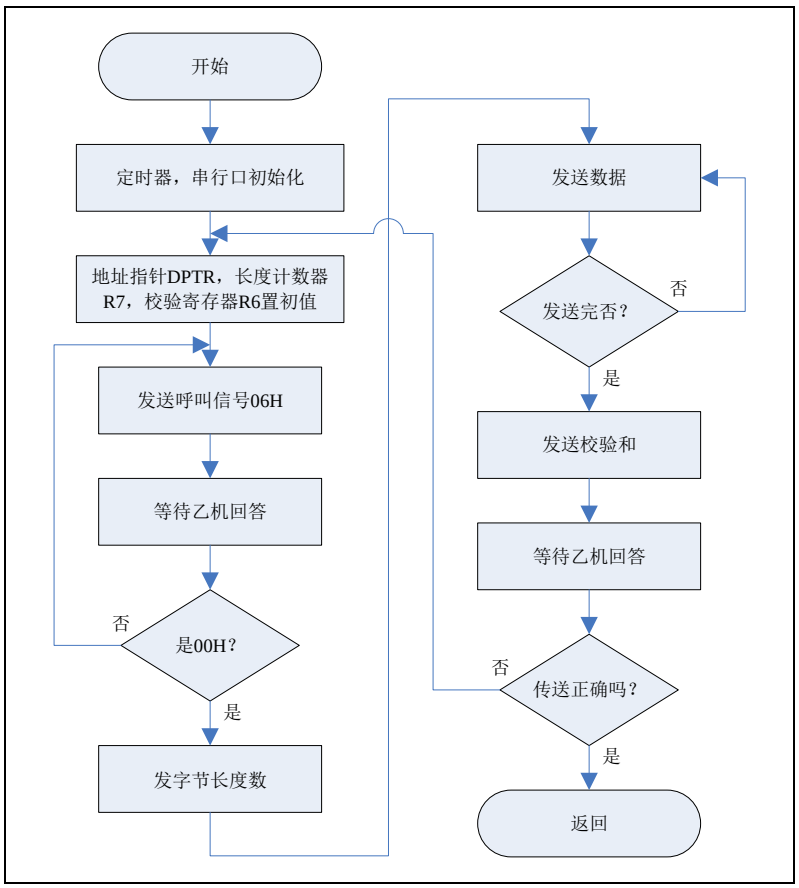
不同的通信要求，软件“协议”内容也不一样，有关需甲、乙双方共同遵守的约定应尽量完善，以防止通信不能正确判别而失败。

STC15 系列单片机的串行通信，可直接采用查询法，也可采用自动中断法。

（1）查询方式双机通信软件举例

①甲机发送子程序段

下图为甲机发送子程序流程图：



甲机发送程序设置：

（a）波特率设置：选用定时器/计数器 1 定时模式、工作方式 2，计数常数 F3H，SMOD=1。波特率为 2400（位/秒）；

（b）串行通信设置：异步通信方式 1，允许接收；

（c）内部 RAM 和工作寄存器设置：

- 31H 和 30H 单元存放发送的数据块首地址；
- 2FH 单元存放发送的数据块个数；
- R6 为累加和寄存器。

甲机发送子程序清单：

```
START:
    MOV     TMOD, #20H          ;设置定时器/计数器 1 定时、工作方式 2
    MOV     TH1, #0F3H          ;设置定时计数常数
    MOV     TL1, #0F3H
    MOV     SCON, #50H          ;串口初始化
    MOV     PCON, #80H          ;设置 SMOD=1
    SETB    TR1                 ;启动定时
ST-RAM:
    MOV     DPH, 31H            ;设置外部 RAM 数据指针
    MOV     DPL, 30H            ;DPTR 初值
```

```
    MOV     R7, 2FH             ;发送数据块数送 R7
    MOV     R6, #00H            ;累加和寄存器 R6 清 0
TX-ACK:
    MOV     A, #06H             ;发送呼叫信号到“06H”
    MOV     SBUF, A
WAIT1:
    JBC     T1, RX-YES          ;等待发送完呼叫信号
    SJMP    WAIT1               ;未发送完转 WATI1
RX-YES:
    JBC     RI, NEXT1           ;判断乙机回答信号
    SJMP    RX-YES              ;未收到回答信号,则等待
NEXT1:
    MOV     A, SBUF             ;接收回答信号送 A
    CJNE    A, #00H, TX-ACK     ;判断是否“00H”,否则重发呼叫信号
TX-BYT:
    MOV     A, R7               ;发送数据块 n
    MOV     SBUF, A
    ADD     A, R6
    MOV     R6, A
WAIT2:
    JBC     TI, TX-NES          ;等待发送完
    JMP     WAIT2
TX-NES:
    MOVX    A, @DPTR            ;从外部 RAM 取发送数据
    MOV     SBUF, A             ;发送数据块
    ADD     A, R6
    MOV     R6, A
    INC     DPTR                ;DPTR 指针加 1
WAIT3:
    JBC     TI, NEXT2           ;判断一数据块发送完否
    SJMP    WAIT3               ;等待发送完
NEXT2:
    DJNZ    R7, TX-NES          ;判断发送全部结束否
TX-SUM:
    MOV     A, R6               ;发送累加和给乙机
    MOV     SBUF, A
WAIT4:
    JBC     TI, RX-0FH          ;等待发送完
    SJMP    WAIT4
RX-0FH:
    JBC     RI, IF-0FH          ;等待接收乙机回答信号
    SJMP    RX-0FH
IF-0FH:
    MOV     A, SBUF             ;判断传输是否正确，否则重新发送
    CJNE    A, #0FH, ST-RAM
```

RET ;返回

乙机接收子程序段接收程序段的设置:

(a)波特率设置初始化：同发送程序;

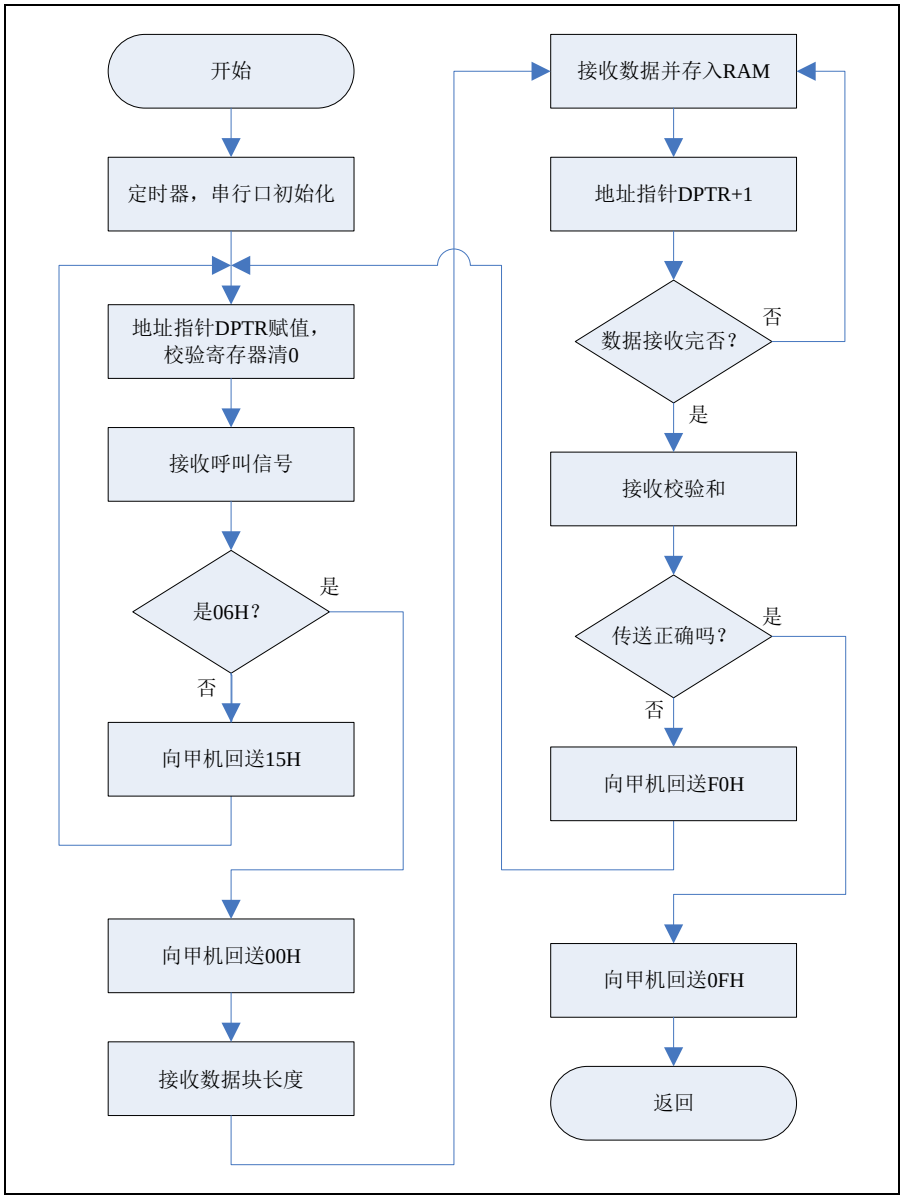
(b)串行通信初始化：同发送程序

(c)寄存器设置:

- 内部 RAM 31H、30H 单元存放接收数据缓冲区首地址。
- R7----数据块个数寄存器。
- R6----累加和寄存器。

（d）向甲机回答信号：“0FH”为接收正确，“F0H”为传送出错，“00H”为同意接收数据，“05H”为暂不接收。

下图为双机通信查询方式乙机接收子程序流程图：



乙机接收子程序清单：

```
START:
    MOV     TMOD, #20H
    MOV     TH1, #0F3H              ;定时器/计数器 1 设置
```

```
    MOV     TL1, #0F3H
    SETB    TR1                 ;启动定时器/计数器 1
    MOV     SCON, #50H          ;置串行通信方式 1,允许接收
    MOV     PCON, #80H          ;SMOD 置位
ST-RAM:
    MOV     DPH, 31H            ;设置 DPTR 首地址
    MOV     DPL, 30H
    MOV     R6, #00H            ;校验和寄存器清 0
RX-ACK:
    JBC     RI, IF-06H          ;判断接收呼叫信号
    SJMP    RX-ACK              ;等待接收呼叫信号
IF-06H:
    MOV     A, SBUF             ;呼叫信号送 A
    CJNEA   #06H, TX-05H        ;判断呼叫信号正确否?
TX-00H:
    MOV     A, #00H             ;向甲机发送“00H”，同意接收
    MOV     SBUF, A
WAIT1:
    JBC     TI, RX-BYS          ;等待应答信号发送完
    SJMP    WAIT1
TX-05H:
    MOV     A, #05H             ;向甲机发送“05H”呼叫
    MOV     SBUF, A             ;不正确信号
WAIT2:
    JBC     TI, HAVE1           ;等待发送完
    SJMP    WAIT2
HAVE1:
    LJMP    RX-ACK              ;因呼叫错,返回重新接收呼叫
RX-BYS:
    JBC     RI, HAVE2           ;等待接收数据块个数
    SJMP    RX-BYS
HAVE2:
    MOV     A, SBUF
    MOV     R7, A               ;数据块个数帧送 R7,R6
    MOV     R6, A
RX-NES:
    JBC     RI, HAVE3           ;接收数据帧
    SJMP    RX-NES
HAVE3:
    MOV     A, SBUF
    MOVX    @DPTR, A            ;接收到的数据存入外部 RAM
    INC     DPTR
    ADD     A, R6               ;形成累加和
    MOV     R6, A
    DJNZ    R7, RX-NES          ;判断数据是否接收完
RX-SUM:
```

```
    JBC     RI, HAVE4           ;等待接收校验和
    SJMP    RX-SUM
HAVE4:
    MOV     A, SBUF             ;判断传输是否正确
    CJNE    A, R6, TX-ERR
TX-RIT:
    MOV     A, #0FH             ;向甲机发送接收正确信息
    MOV     SBUF, A
WAIT3:
    JBC     TI, GOOD            ;等待发送结束
    SJMP    WAIT3
TX-ERR:
    MOV     A, #0F0H            ;向甲机发送传输有误信号
    MOV     SBUF, A
WAIT4:
    JBC     TI, AGAIN           ;等待发送完
    SJMP    WAIT4
AGAIN:
    LJMP    ST-RAM              ;返回重新开始接收
GOOD:
    RET                         ;传输正确返回
```

(2)中断方式双机通信软件举例

在很多应用场合，双机通信的双方或一方采用中断方式以提高通信效率。由于 STC15 系列单片机的串行通信是双工的，且中断系统只提供一个中断矢量入口地址，所以实际上是中断和查询必须相结合，即接收/发送均可各自请求中断，响应中断时主机并不知道是谁请求中断，统一转入同一个中断矢量入口，必须由中断服务程序查询确定并转入对应的服务程序进行处理。

这里，仍以上述协议为例，甲方（发送方）任以查询方式通信（从略），乙方（接收方）则改用中断一查询方式进行通信。

在中断接收服务程序中，需设置三个标志位来判断所接收的信息是呼叫信号还是数据块个数，是数据还是校验和。增设寄存器：内部 RAM32H 单元为数据块个数寄存器，33H 单元为校验和寄存器，位地址 7FH、7EH、7DH 为标志位。

乙机接收中断服务程序清单

采用中断方式时，应在主程序中安排定时器/计数器、串行通信等初始化程序。通信接收的数据存放在外部 RAM 的首地址也需在主程序中确定。

主程序:

```
    ORG     0000H
    AJMP    START               ;转至主程序起始处
    ORG     0023H
    LIMP    SERVE               ;转中断服务程序处
    … …
START：
```

```
    MOV     TMOD, #20H          ;定义定时器/计数器 1 定时、工作方式 2
    MOV     TH1, #0F3H          ;设置波特率为 2400 位/秒
    MOV     TL1, #0F3H
    MOV     SCON, #50H          ;设置串行通信方式 1,允许接收
    MOV     PCON, #80H          ;设置 SMOD=1
    SETB    TR1                 ;启动定时器
    SETB    7FH
    SETB    7EH                 ;设置标志位为 1
    SETB    7DH
    MOV     31H, #10H           ;规定接收的数据存储于外部 RAM 的起始地址 1000H
    MOV     30H, #00H
    MOV     33H, #00H           ;累加和单元清 0
    SETB    EA                  ;开中断
    SETB    ES
    …
```

中断服务程序：

```
SERVE：
    CLR     EA                  ;关中断
    CLR     RI                  ;清除接收中断请求标志
    PUSH    DPH
    PUSH    DPL                 ;现场保护
    PUSH    A
    JB      7FH, RXACK          ;判断是否是呼叫信号
    JB      7EH, RXBYS          ;判断是否是数据块数据
    JB      7DH, RXDATA         ;判断是否是接收数据帧
RXSUM：
    MOV     A, SBUF             ;接收到的校验和
    CJNE    A, 33H, TXERR       ;判断传输是否正确
TXRI：
    MOV     A, #0FH             ;向甲机发送接收正确信号“0FH”
    MOV     SBUF, A
WAIT1：
    JNB     TI, WAIT1           ;等待发送完毕
    CLR     TI                  ;清除发送中断请求标志位
    SJMP    AGAIN               ;转结束处理
TXERR：
    MOV     A, #0F0H            ; 向甲机发送接收出错信号“F0H”
    MOV     SBUF, A
WAIT2：
    JNB     TI, WAIT2           ;等待发送完毕
    CLR     TI                  ;清除发送中断请求标志
    SJMP    AGAIN               ;转结束处理
RXACK：
    MOV     A, SBUF             ;判断是否是呼叫信号“06H”
    XRL     A, #06H             ;异或逻辑处理
```

```
    JZ      TXREE           ;是呼叫,则转 TXREE
TXNACK：
    MOV     A, #05H         ;接收到的不是呼叫信号,则向甲机发送
    MOV     SBUF, A         ;“05H”,要求重发呼叫
WAIT3：
    JNB     TI, WAIT3       ;等待发送结束
    CLR     TI
    SJMP    RETURN          ;转恢复现场处理
TXREE：
    MOV     A, #00H         ;接收到的是呼叫信号,发送“00H”
    MOV     SBUF, A         ;接收到的是呼叫信号,发送“00H”
WAIT4：
    JNB     TI, WAIT4       ;等待发送完毕
    CLR     TI              ;清除 TI 标志
    CLR     7FH             ;清除呼叫标志
    SJMP    RETURN          ;转恢复现场处理
RXBYS：
    MOV     A, SBUF         ;接收到数据块数
    MOV     32H, A          ;存入 32H 单元
    ADD     A, 33H          ;形成累加和
    MOV     33H, A
    CLR     7EH             ;清除数据块数标志
    SJMP    RETURN          ;转恢复现场处理
RXDATA：
    MOV     DPH, 31H        ;设置存储数据地址指针
    MOV     DPL, 30H
    MOV     A, SBUF         ;读取数据帧
    MOVX    @DPTR, A        ;将数据存外部 RAM
    INC     DPTR            ;地址指针加 1
    MOV     31H, DPH        ;保存地址指针值
    MOV     30H, DPL
    ADD     A, 33H          ;形成累加和
    MOV     33H, A
    DJNZ    32H, RETURN     ;判断数据接收完否
    CLR     7DH             ;清数据接收完标志
    SJMP    RETURN          ;转恢复现场处理
AGAIN：
    SETB    7FH
    SETB    7EH             ;恢复标志位
    SETB    7DH
    MOV     33H, #00H       ;累加和单元清 0
    MOV     31H, #10H       ;恢复接收数据缓冲区首地址
    MOV     30H, #00H
RETURN：
    POP     A
    POP     DPL             ;恢复现场
```

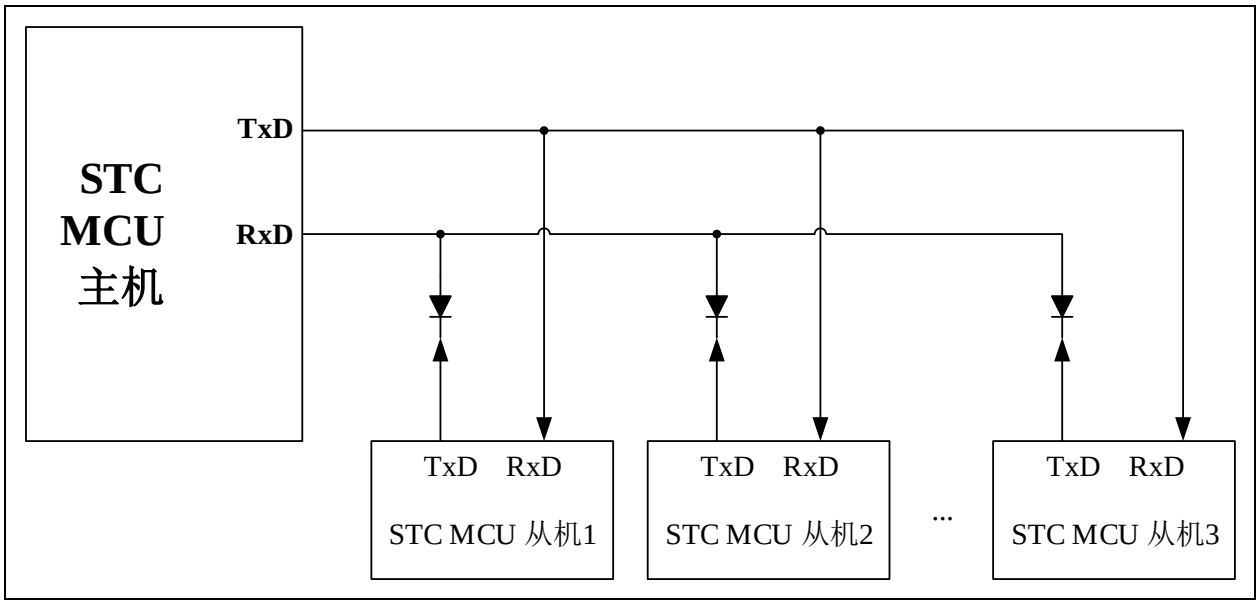
```
POP     DPH
SETB    EA                  ;开中断
RETI                        ;返回
```

上述程序清单中，ORG 为程序段说明伪指令，在程序汇编时，它向汇编程序说明该程序段的起始地址。

在实际应用中情况多种多样，而且是两台独立的计算机之间进行信息传输。因此，应周密考虑通信协议，以保证通信的正确性和成功率。

19.13 多机通信

在很多实际应用系统中，需要多台微计算机协调工作。STC15 系列单片机的串行通信方式 2 和方式 3 具有多机通信功能，可构成各种分布式通信系统。下图为全双工主从式多机通信系统的连接框图。



上图为一台主机和几台从机组成的全双工多机通信系统。主机可与任一台从机通信，而从机之间的通信必须通过主机转发。

（1）多机通信的基本原理

在多机通信系统中，为保证主机（发送）与多台从机（接收）之间能可通信，串行通信必须具备识别能力。

MCS-51 系列单片机的串行通信控制寄存器 SCON 中设有多机通信选择位 SM2。当程序设置 SM2=1，串行通信工作于方式 2 或方式 8，发送端通过对 TB8 的设置以区别于发送的是地址帧（TB8=1）还是数据帧（TB8=0），接收端通过对接收到 RB8 进行识别：

- 当 SM2=1，
  - ✓ 若接收到 RB8=1，则被确认为呼叫地址帧，将该帧内容装入 SBUF 中，并置位 RI=1，向 CPU 请求中断，进行地址呼叫处理；
  - ✓ 若 RB8=0 为数据帧，将不予理睬，接收的信息被丢弃。
- 若 SM2=0，则无论是地址帧还是数据帧均接收，并置位 RI=1，向 CPU 请求中断，将该帧内容装入 SBUF。

据此原理，可实现多机通信。

对于上图的从机式多机通信系统，从机的地址为 0，1，2，…，n。实现多机通信的过程如下：

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 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 TB8=0，以表示为数据帧。

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。

（2）多机通信协议简述

由于串行通信是在二台或多台各自完全独立的系统之间进行信息传输这就需要根据时间通信要求制定某些约定，作为通信规范遵照执行，协议要求严格、完善，不同的通信要求，协议的内容也不相同。在多机通信系统中要考虑的问题较多，协议内容比较复杂。这里仅例举几条作一说明。

上图的主从式多机通信系统，允许配置 255 台从机，各从机的地址分别为 00H ~ FEH。

① 约定地址 FFH 为全部从机的控制命令，命令各从机恢复 SM2=1 状态，准备接收主机的地址呼叫。

② 主机和从机的联络过程约定：主机首先发送地址呼叫帧，被寻址的从机回送本机地址给主机，经验证地址相符后主机再向被寻址的从机发送命令字，被寻址的从机根据命令字要求回送本机的状态，若主机判断状态正常，主机即开始发送或接收数据帧，发送或接收的第一帧为传输数据块长度。

③ 约定主机发送的命令字为:

00H：要求从机接收数据块:
01H：要求从机发送数据块;
…
其他：非法命令。

④ 从机的状态字格式约定为:

B7	B6	B5	B4	B3	B2	B1	B0
ERR	0	0	0	0	0	TRDY	RRDY

定义：若 ERR=1，从机接收到非法命令；
若 TRDY=1，从机发送准备就绪；
若 RRDY=1，从机接收准备就绪

⑤ 其他：如传输出错措施等。

（3）程序举例

在实际应用中如传输波特率不太高，系统实时性有一定要求以及希望提高通信效率，则多半采用中断控制方式，但程序调试较困难，这就要求提高程序编制的正确性。采用查询方式，则程序调试较方便。这里仅以中断控制方式为例简单介绍主----从机之间一对一通信软件。

① 主机发送程序

该主机要发送的数据存放在内部 RAM 中，数据块的首地址为 51H，数据块长度存放做 50H 单元中，有关发送前的初始化、参数设置等采用子程序格式，所有信息发送均由中断服务程序完成。当主机需要发送时，在完成发送子程序的调用之后，随即返回主程序继续执行。以后只需查询 PSW • 5 的 F0 标志位的状态即可知道数据是否发送完毕。

要求主机向#5 从机发送数据，中断服务程序选用工作寄存存器区 1 的 R0 ~ R7。

主机发送程序清单：

```
        ORG     0000H
        AJMP    START                   ;转主程序
        ORG     0023H                   ;发送中断服务程序入口
        LJMP    SERVE                   ;转中断服务程序
        …

START:                                  ;主程序
        …       …
        ORG     1000H                   ;发送子程序入口
TXCALL:
        MOV     TMOD, #20H              ;设置定时器/计数器 1 定时、方式 2
        MOV     TH1, #0F3H              ;设置波特率为 2400 位/秒
        MOV     TL1, #0F3H              ;置位 SMOD
        MOV     PCON, #80H
        SETB    TR1                     ;启动定时器/计数器 1
        MOV     SCON, #0D8H             ;串行方式 8，允许接收, TB8=1
        SETB    EA                      ;开中断总控制位
        CLR     ES                      ;禁止串行通信中断
TXADDR:
        MOV     SBUF, #05H              ;发送呼叫从机地址
WAIT1:
        JNB     TI, WAIT1               ;等待发送完毕
        CLR     TI                      ;复位发送中断请求标志
RXADDR:
        JNB     RI, RXADDR              ;等待从机回答本机地址
        CLR     TI                      ;复位接收中断请求标志
        MOV     A, SBUF                 ;读取从机回答的本机地址
        CJNE    A, #05H, TXADDR         ;判断呼叫地址符否，否则重发
        CLR     TB8                     ;地址相符，复位 TB8=0，准备发数据
        CLR     PSW.5                   ;复位 F0=0 标志位
        MOV     08H, #50H               ;发送数据地址指针送 R0
        MOV     0CH, 50H                ;数据块长度送 R4
        INC     0CH                     ;数据块长度加 1
        SETB    ES                      ;允许串行通信中断
        RET                             ;返回主程序
        . . .
SERVE:
        CLR     TI                      ;中断服务程序段,清中断请求标志 TI
        PUSH    PSW                     ;现场入栈保护
        PUSH    A
        CLR     RS1                     ;选择工作寄存器组 1
        SETB    RS0
TXDATA:
```

```
    MOV     SBUF, @R0               ;发送数据块长度及数据
WAIT2:
    JNB     TI, WAIT2               ;等待发送完毕
    CLR     TI                      ;复位 TI=0
    INC     R0                      ;地址指针加 1
    DJNZ    R4, RETURN              ;数据块未发送完,转返回
    SETB    PSW.5                   ;已发送完毕置位 F0=1
    CLR     ES                      ;关闭串行中断
RETURN:
    POP     A                       ;恢复现场
    POP     PSW
    RETI                            ;返回
```

② 从机接收程序

主机发送的地址呼叫帧，所有的从机均接收，若不是呼叫本机地址即从中断返回；若是本机地址，则回送本机地址给主机作为应答，并开始接收主机发送来的数据块长度帧，并存放于内部 RAM 的 60H 单元中，紧接着接收的数据帧存放于 61H 为首地址的内部 RAM 单元中，程序中还选用 20H.0、20H.1 位作标志位，用来判断接收的是地址、数据块长度还是数据，选用了 2FH、2EH 两个字节单元用于存放数据字节数和存储数据指针。#5 从机的接收程序如下，供参考。

#5 从机接收程序清单：

```
    ORG     0000H
    AJMP    START                   ;转主程序段
    ORG     0023H
    LJMP    SERVE                   ;从中断入口转中断服务程序
    ORG     0100H
START:
    MOV     TMOD, #20H              ;主程序段:初始化程序,设置定时
    MOV     TH1, #0F3H              ;器/计数器 1 定时、工作方式 2,设
    MOV     TL1, #0F3H              ;置波特率为 2400 位/秒的有关初值
    MOV     PCON, #80H              ;置位 SMOD
    MOV     SCON, #0F0H             ;设置串行方式 3,允许接收, SM2=1
    SETB    TR1                     ;启动定时器/计数器 1
    SETB    20H.0                   ;置标志位为 1
    SETB    20H.1
    SETB    EA                      ;开中断
    SETB    ES
    . . .
    ORG     1000H
SERVE:
    CLR     RI                      ;清接收中断请求标志 RI=0
    PUSH    A                       ;现场保护
    PUSH    PSW
    CLR     RS1                     ;选择工作寄存器组 1
    SETB    RS0
    JB      20H.0, ISADDR           ;判断是否是地址帧
```

```
    JB      20H.1, ISBYTE       ;判断是否是数据块长度帧
ISDATA:
    MOV     R0, 2EH             ;数据指针送 R0
    MOV     A, SBUF             ;接收数据
    MOV     @R0, A
    INC     2EH                 ; 数据指针加 1
    DJNZ    2FH, RETURN         ;判断数据接收完否?
    SETB    20H.0
    SETB    20H.1               ;恢复标志位
    SETB    SM2
    SJMP    RETURN              ;转入恢复现场,返回
ISADDR:
    MOV     A, SBUF             ;是地址呼叫, 判断与本机地址相符否，不符则转返回
    CJNE    A, #05H, RETURN
    MOV     SBUF, #01H          ;相符, 发回答信号“01H”
WAIT:
    JNB     TI, WAIT            ;等待发送结束
    CLR     TI                  ;清 0TI, 20H.0, SM2
    CLR     20H.0               ;清 0TI, 20H.0, SM2
    CLR     SM2                 ;清 0TI, 20H.0, SM2
    SJMP    RETURN              ;转返回
ISBYTES:
    MOV     A, SBUF             ;接收数据块长度帧
    MOV     R0, #60H
    MOV     @R0, A              ;将数据块长度存入内部 RAM
    MOV     2FH, A              ;60H 单元及 2FH 单元
    MOV     2EH, #61H           ;置首地址 61H 于 2EH 单元
    CLR     20H.1               ;清 20H.1 标志, 表示以后接收的为数据
RETURN:
    POP     PSW                 ;恢复现场
    POP     A
    RETI                        ;返回
```

多机通信方式可多种多样，上例仅以最简单的住一从式作了简单介绍，仅供参考。

对于串行通信工作方式 0 的同步方式，常用于通过移位寄存器进行扩展并行 I/O 口，或配置某些串行通信接口的外部设备。例如，串行打印机、显示器等。这里就不一一举例了。

19.14 串口 1 作为增强型串口使用时的自动地址识别功能

19.14.1 与串口 1 自动地址识别功能相关的特殊功能寄存器

符号	描述	地址	位地址及符号								复位值
			MSB							LSB	
SCON	串口控制寄存器	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000 0000B
SBUF	串口发送/接收数据缓冲器	99H									xxxx xxxxB
SADEN	从机地址屏蔽位寄存器	B9H									0000 0000B
SADDR	从机地址寄存器	A9H									0000 0000B

1.串行口 1 的控制寄存器 SCON

串行控制寄存器 SCON 用于选择串行通信的工作方式和某些控制功能。其格式如下:

SCON：串行控制寄存器(可位寻址)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
SCON	98H	name	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI

SMO/FE：当 PCON 寄存器中的 SMOD0/PCON.6 位为 1 时，该位用于帧错误检测。当检测到一个无效停止位时，通过 UART 接收器设置该位。它必须由软件清零。

当 PCON 寄存器中的 SMODO/PCON.6 位为 0 时，该位和 SM1 一起指定串行通信的工作方式，如下表所示。

其中 SM0、SM1 按下列组合确定串行口 1 的工作方式：

SM0	SM1	工作方式	功能说明	波特率
0	0	方式 0	同步移位串行方式：移位寄存器	当 UART_M0x6 = 0 时，波特率是 SYSclk / 12, 当 UART_M0x6 = 1 时，波特率是 SYSclk / 2
0	1	方式 1	8 位 UART，波特率可变	➢ 当串行口 1 用定时器 1 作为其波特率发生器且定时器 1 工作于模式 0(16 位自动重装载模式)或串行口用定时器 2 作为其波特率发生器时， ✓ 波特率=(定时器 1 的溢出率或定时器 T2 的溢出率) / 4 注意：此时波特率与 SMOD 无关。 ➢ 当串行口 1 用定时器 1 作为其波特率发生器且定时器 1 工作于模式 2(8 位自动重装模式)时， ✓ 波特率=(2^{SMOD} / 32) ×(定时器 1 的溢出率)
1	0	方式 2	9 位 UART	（2^{SMOD} /64) ×SYSclk 系统工作时钟频率
1	1	方式 3	9 位 UART，波特率可变	➢ 当串行口 1 用定时器 1 作为其波特率发生器且定时器 1 工作于模式 0(16 位自动重装载模式)或串行口用定时器 2 作为其波特率发生器时， ✓ 波特率=(定时器 1 的溢出率或定时器 T2 的溢出率) / 4 注意：此时波特率与 SMOD 无关。 ➢ 当串行口 1 用定时器 1 作为其波特率发生器且定时器 1 工作于模式 2(8 位自动重装模式)时， ✓ 波特率=(2^{SMOD} / 32) ×(定时器 1 的溢出率)

SM2： 允许方式 2 或方式 3 多机通信控制位。

- 在方式 2 或方式 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。
- 在方式 2 或方式 3 中，如果 SM2 位为 0 且 REN 位为 1，接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。方式 1 和方式 0 是非多机通信方式，在这两种方式时，要设置 SM2 应为 0。

REN： 允许/禁止串行接收控制位。

- 由软件置位 REN，即 REN=1 为允许串行接收状态，可启动串行接收器 RxD，开始接收信息。
- 软件复位 REN，即 REN=0，则禁止接收。

TB8： 在方式 2 或方式 3，它为要发送的第 9 位数据，按需要由软件置位或清 0。例如，可用作据的校验位或多机通信中表示地址帧/数据帧的标志位。在方式 0 和方式 1 中，该位不用。

RB8： 在方式 2 或方式 3，是接收到的第 9 位数据，作为奇偶校验位或地址帧/数据帧的标志位。方式 0 中不用 RB8（置 SM2=0）。方式 1 中也不用 RB8（置 SM2=0，RB8 是接收到的停止位）。

TI： 发送中断请求标志位。

- 在方式 0，当串行发送数据第 8 位结束时，由内部硬件自动置位即 TI=1，向主机请求中断，响应中断后 TI 必须用软件清零，即 TI=0。
- 在其他方式中，则在停止位开始发送时由内部硬件置位，即 TI=1，响应中断后 TI 必须用软件清零。

RI： 接收中断请求标志位。

- 在方式 0，当串行接收到第 8 位结束时，由内部硬件自动置位 RI=1，向主机请求中断，响应中断后 RI 必须用软件清零，即 RI=0。
- 在其他方式中，串行接收到停止位的中间时刻由内部硬件置位，即 RI=1，向 CPU 发中断申请，响应中断后 RI 必须由软件清零。

SCON 的所有位可通过整机复位信号复位为全“0”。
SCON 的字节地址为 98H，可位寻址，各位地址为 98H ~ 9FH，可用软件实现位设置。
串行通信的中断请求：

- 当一帧发送完成，内部硬件自动置位 TI，即 TI=1，请求中断处理；
- 当接收完一帧信息时，内部硬件自动置位 RI，即 RI=1，请求中断处理。

由于 TI 和 RI 以“或逻辑”关系向主机请求中断，所以主机响应中断时事先并不知道是 TI 还是 RI 请求的中断，必须在中断服务程序中查询 TI 和 RI 进行判别，然后分别处理。因此，两个中断请求标志位均不能由硬件自动置位，必须通过软件清 0，否则将出现一次请求多次响应的错误。

2.串行口数据缓冲寄存器 SBUF

STC15 系列单片机的串行口 1 缓冲寄存器（SBUF）的地址是 99H，实际是 2 个缓冲器，写 SBUF 的操作完成待发送数据的加载，读 SBUF 的操作可获得已接收到的数据。两个操作分别对应两个不同的寄存器，1 个是只写寄存器，1 个是只读寄存器。

串行通道内设有数据寄存器。在所有的串行通信方式中，在写入 SBUF 信号（MOV SBUF, A）的控制下，把数据装入相同的 9 位移位寄存器，前面 8 位为数据字节，其最低位为移位寄存器的输出位。根据不同的工作方式会自动将“1”或 TB8 的值装入移位寄存器的第 9 位，并进行发送。

串行通道的接收寄存器是一个输入移位寄存器。在方式 0 时它的字长为 8 位，其他方式时为 9 位。当一帧接收完毕，移位寄存器中的数据字节装入串行数据缓冲器 SBUF 中，其第 9 位则装入 SCON 寄存器中的 RB8 位。如果由于 SM2 使得已接收到的数据无效时，RB8 和 SBUF 中内容不变。

由于接收通道内设有输入移位寄存器和 SBUF 缓冲器，从而能使一帧接收完将数据由移位寄存器装入 SBUF 后，可立即开始接收下一帧信息，主机应在该帧接收结束前从 SBUF 缓冲器中将数据取走，否则前一帧数据将丢失。

SBUF 以并行方式送往内部数据总线。

3.从机地址控制寄存器 SADEN 和 SADDR

为了方便多机通信，STC15 系列单片机设置了从机地址控制寄存器 SADEN 和 SADDR。其中 SADEN 是从机地址掩模寄存器（地址为 B9H，复位值为 00H），SADDR 是从机地址寄存器（地址为 A9H，复位值为 00H）。

19.14.2 串口 1 自动地址识别功能的介绍

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=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“don’t care bit”（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

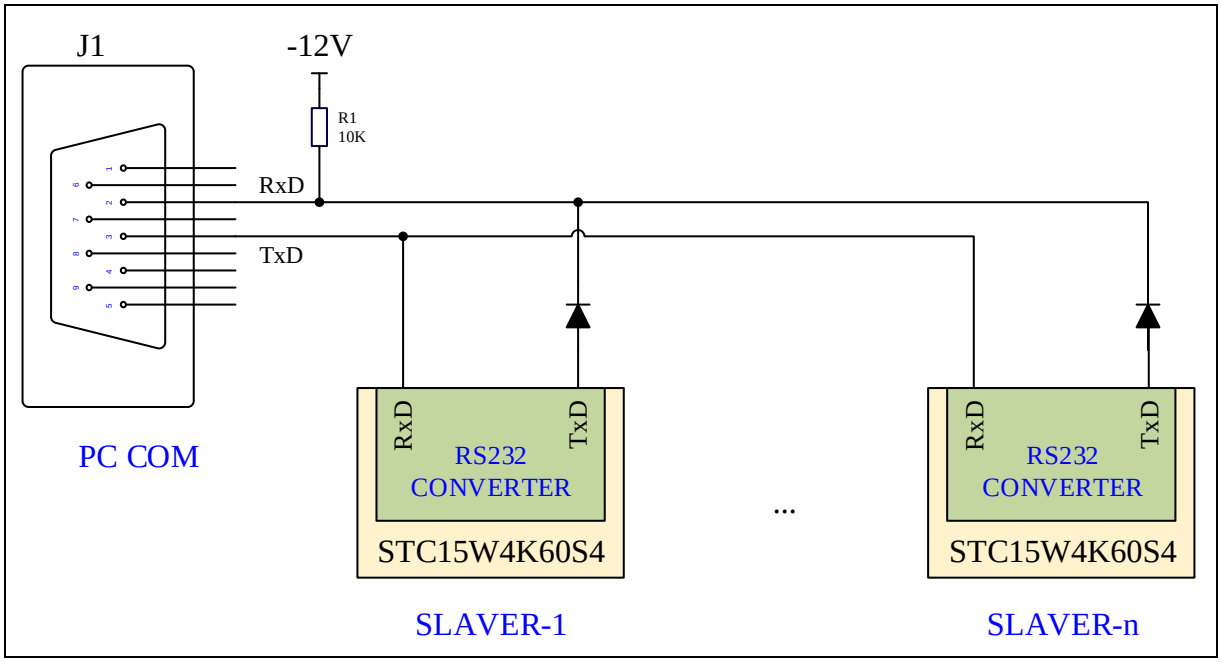
SADDR = 11001010

SADEN = 00001111

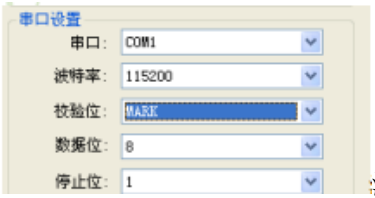
则匹配地址为 xxxx1010

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。

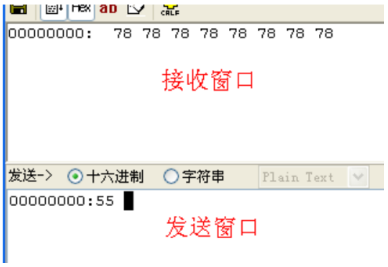
主机可以使用广播地址（FFH）同时选中所有的从机来进行通讯。



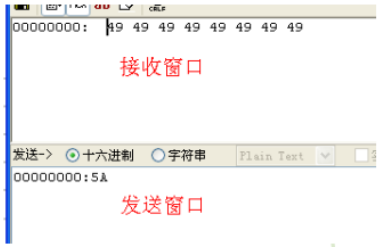
示例代码的测试方法：

1. 首先将两个 MCU 按照上图的链接方法与电脑相连接
2. 将代码中 SLAVE 定义为 0（“#define SLAVER 0”），编译产生的 HEX 文件烧录到 SLAVER-1 的 MCU 中；然后将 SLAVE 定义为 1（“#define SLAVER 1”），编译产生的 HEX 文件烧录到 SLAVER-2 的 MCU 中
3. 在 PC 端，打开串口助手，将串口设置如有图， 

注意校验位
4. 在串口助手终端，发送 0x55，则会选中从机 1，此时从机 1 应答的数据为 8 个 0x78



5、串口助手终端再发送 0x5a，则会选中从机 2，此时从机 2 应答的数据为 8 个 0x49，如图



19.14.3 串口 1 自动地址识别功能的测试程序（C 和汇编）

1.C 程序:

```
/*----STC15F2K60S2 系列串口 1 地址自动识别举例举例----------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可----*/
//假定测试芯片的工作频率为 18.432MHZ

#include "reg51.h"
#include "intrins.h"

typedef   unsigned char     BYTE;
typedef   unsigned int      WORD;
//-----------------------------------------------
#define   SLAVER            0           //定义从机编号,0 为从机 1, 1 为从机 2

#if       SLAVER == 0
#define   SAMASK            0x33        //从机 1 地址屏蔽位
#define   SERADR            0x55        //从机 1 的地址为 xx01,xx01
#define   ACKTST            0x78        //从机 1 应答测试数据
#else
#define   SAMASK            0x3C        //从机 2 地址屏蔽位
#define   SERADR            0x5A        //从机 2 的地址为 xx01,10xx
#define   ACKTST            0x49        //从机 2 应答测试数据
#endif

#define   URMD              0           //0:使用定时器 2 作为波特率发生器
                                        //1:使用定时器 1 的模式 0(16 位自动重载模式)作为波特率发生器
                                        //2:使用定时器 1 的模式 2(8 位自动重载模式)作为波特率发生器

sfr       T2H = 0xd6;                   //定时器 2 高 8 位
sfr       T2L = 0xd7;                   //定时器 2 低 8 位

sfr       AUXR = 0x8e;                  //辅助寄存器

sfr       SADDR = 0xA9;                 //从机地址寄存器
sfr       SADEN = 0xB9;                 //从机地址屏蔽寄存器

void InitUart();

char count;

void main()
{
    InitUart();                         //初始化串口
```

```
    ES = 1;
    EA = 1;
    while (1);
}
/*----------------------------UART 中断服务程序-----------------------------*/
void Uart() interrupt 4 using 1
{
    if (TI)
    {
        TI = 0;                         //清除 TI 位
        if (count != 0)
        {
            count--;
            SBUF = ACKTST;              //继续发送应答数据
        }
        else
        {
            SM2 = 1;                    //若发送完成,则重新开始地址检测
        }
    }
    if (RI)
    {
        RI = 0;                         //清除 RI 位
        SM2 = 0;                        //本机被选中后,进入数据接收状态
        count = 7;
        SBUF = ACKTST;                  //并开发送应答数据
    }
}
/*----------------------------初始化串口-----------------------------*/
void InitUart()
{
    SADDR = SERADR;
    SADEN = SAMASK;
    SCON = 0xf8;                        //设置串口为 9 位可变波特率,使能多机通讯检测,
                                        //(将 TB8 设置为 1, 方便与 PC 直接通讯测试)
#if  URMD == 0
    T2L = 0xd8;                         //设置波特率重装值
    T2H = 0xff;                         //115200 bps(65536-18432000/4/115200)
    AUXR = 0x14;                        //T2 为 1T 模式, 并启动定时器 2
    AUXR |= 0x01;                       //选择定时器 2 为串口 1 的波特率发生器
#elif URMD == 1
    AUXR = 0x40;                        //定时器 1 为 1T 模式
    TMOD = 0x00;                        //定时器 1 为模式 0(16 位自动重载)
    TL1 = 0xd8;                         //设置波特率重装值
    TH1 = 0xff;                         //115200 bps(65536-18432000/4/115200)
    TR1 = 1;                            //定时器 1 开始启动
```

```
#else
    TMOD = 0x20;                    //设置定时器 1 为 8 位自动重装载模式
    AUXR = 0x40;                    //定时器 1 为 1T 模式
    TH1 = TL1 = 0xfb;               //115200 bps(256 - 18432000/32/115200)
    TR1 = 1;
#endif
}
```

2. 汇编程序：

```
/*----STC15F2K60S2 系列串口 1 地址自动识别举例举例----------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可----*/
;假定测试芯片的工作频率为 18.432MHZ

#define     SLAVER      0           ;定义从机编号,0 为从机 1, 1 为从机 2

#if         SLAVER == 0
#define     SAMASK      0x33        ;从机 1 地址屏蔽位
#define     SERADR      0x55        ;从机 1 的地址为 xx01,xx01
#define     ACKTST      0x78        ;从机 1 应答测试数据
#else
#define     SAMASK      0x3C        ;从机 2 地址屏蔽位
#define     SERADR      0x5A        ;从机 2 的地址为 xx01,10xx
#define     ACKTST      0x49        ;从机 2 应答测试数据
#endif

#define     URMD        0           ;0:使用定时器 2 作为波特率发生器
                                    ;1:使用定时器 1 的模式 0(16 位自动重载模式)作为波特率发生器
                                    ;2:使用定时器 1 的模式 2(8 位自动重载模式)作为波特率发生器

T2H         DATA        0D6H        ;定时器 2 高 8 位
T2L         DATA        0D7H        ;定时器 2 低 8 位

AUXR        DATA        08EH        ;辅助寄存器

SADDR       DATA        0A9H        ;从机地址寄存器
SADEN       DATA        0B9H        ;从机地址屏蔽寄存器

COUNT       DATA        20H
;-------------------------------------
    ORG     0000H
    LJMP    START
    ORG     0023H
    LJMP    UART_ISR
;-------------------------------------
    ORG     0100H
```

```
START:
    MOV     SP, #3FH
    LCALL   INIT_UART               ;初始化串口
    SETB    ES
    SETB    EA
    SJMP    $
;-------------------------串口中断服务程序----------------
UART_ISR:
    PUSH    PSW
    PUSH    ACC
    JNB     TI, CHK_RX
    CLR     TI
    MOV     A, COUNT                ;发送完成 8 个数据后,就不再发送
    JZ      RESTART
    DEC     COUNT
    MOV     SBUF, #ACKTST           ;发送应答测试数据
    JMP     UREXIT
RESTART:
    SETB    SM2                     ;若发送完成,则重新开始地址检测
    JMP     UREXIT
CHK_RX:
    JNB     RI, UREXIT
    CLR     RI
    CLR     SM2                     ;本机被选中后,进入数据接收状态
    MOV     SBUF, #ACKTST           ;并开发送应答数据
    MOV     COUNT, #7
UREXIT:
    POP     ACC
    POP     PSW
    RETI
;---------------------------初始化串口---------------------------
INIT_UART:
    MOV     SADDR, #SERADR
    MOV     SADEN, #SAMASK
    MOV     SCON, #0F8H             ;设置串口为 9 位可变波特率,使能多机通讯检测,
                                    ;(将 TB8 设置为 1,方便与 PC 直接通讯测试)
#if   URMD == 0
    MOV     T2L, #0D8H              ;设置波特率重装值(65536-18432000/4/115200)
    MOV     T2H, #0FFH
    MOV     AUXR, #14H              ;T2 为 1T 模式, 并启动定时器 2
    ORL     AUXR, #01H              ;选择定时器 2 为串口 1 的波特率发生器
#elif URMD == 1
    MOV     AUXR, #40H              ;定时器 1 为 1T 模式
    MOV     TMOD, #00H              ;定时器 1 为模式 0(16 位自动重载)
    MOV     TL1, #0D8H              ;设置波特率重装值(65536-18432000/4/115200)
    MOV     TH1, #0FFH
```

```
    SETB    TR1                 ;定时器 1 开始运行
#else
    MOV     TMOD, #20H          ;设置定时器 1 为 8 位自动重装载模式
    MOV     AUXR, #40H          ;定时器 1 为 1T 模式
    MOV     TL1, #0FBH          ;115200 bps(256 - 18432000/32/115200)
    MOV     TH1, #0FBH
    SETB    TR1
#endif
    RET
;----------------------------------------
END
```

19.15 串行口 1 的中继广播方式

串行口 1 可在 3 组管脚间进行切换：[RxD/P3.0, TxD/P3.1];
[RxD_2/P3.6, TxD_2/P3.7];
[RxD_3/P1.6, TxD_3/P1.7]

Mnemonic	Add	Name	7	BB6	B5	B4	B3	B2	B1	B0	Reset Value
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000,0000

Tx_Rx：串口 1 的中继广播方式设置

0：串口 1 为正常工作方式

1：串口 1 为中继广播方式，即将 RxD 端口输入的电平状态实时输出在 TxD 外部管脚上，TxD 外部管脚可以对 RxD 管脚的输入信号进行实时整形放大输出，TxD 管脚的对外输出实时反映 RxD 端口输入的电平状态。

串口 1 的 RxD 管脚和 TxD 管脚可以在 3 组不同管脚之间进行切换：[RxD/P3.0, TxD/P3.1];
[RxD_2/P3.6, TxD_2/P3.7];
[RxD_3/P1.6, TxD_3/P1.7]

串行口 1 的中继广播方式除可以在用户程序中设置 Tx_Rx/CLK_DIV.4 来选择外，还可以在 AIapp-ISP 下载编程软件中设置。

当单片机的工作电压低于上电复位门槛电压（POR，3V 单片机在 1.8V 附近，5V 单片机在 3.2V 附近）时，Tx_Rx 默认为 0，即串行口 1 默认为正常工作方式。

当单片机的工作电压高于上电复位门槛电压（POR，3V 单片机在 1.8V 附近，5V 单片机在 3.2V 附近）时，单片机首先读取用户在 AIapp-ISP 下载编程软件中的设置，

- 如果用户允许了“单片机 TxD 管脚的对外输出实时反映 RxD 端口输入的电平状态”，即中继广播方式，则上电复位后 P3.7/TxD_2 管脚的对外输出可以实时反映 P3.6/RxD_2 端口输入的电平状态；
- 如果用户未选择“单片机 TxD 管脚的对外输出实时反映 RxD 端口输入的电平状态”，则上电复位后串口 1 为正常工作方式，即 P3.7/TxD_2 管脚的对外输出不实时反映 P3.6/RxD_2 端口输入的电平状态。

串行口 1 的位置和中继广播方式除可以在 AIapp-ISP 下载编程软件中设置外，还可以在用户的用户程序中用设置。在 AIapp-ISP 下载编程软件中的设置是在单片机上电复位后就可以执行的，如果用户在

用户程序中的设置与 AIapp-ISP 下载编程软件中的设置不一致时，当执行到相应的用户程序时就会覆盖原来 AIapp-ISP 下载编程软件中的设置。

19.16 用 T0 软件模拟串行口的测试程序(C 及汇编)

----如串行口不够用或无串行口可用[P3.0, P3.1]结合定时器 0 软件模拟串行口

1.C 程序:

```
/*----STC15Fxx 系列软件模拟串口举例------------------------------------------*/
/*----本示例在 Keil 开发环境下请选择 Intel 的 8058 芯片型号进行编译----*/
//假定测试芯片的工作频率为 18.432MHz

#include "reg51.h"
//-----------------------------------------
//define baudrate const
//BAUD = 65536 - FOSC/3/BAUDRATE/M (1T:M=1; 12T:M=12)
//NOTE: (FOSC/3/BAUDRATE) must be greater then 98, (RECOMMEND GREATER THEN 110)

//#define     BAUD     0xF400      // 1200bps @ 11.0592MHz
//#define     BAUD     0xFA00      // 2400bps @ 11.0592MHz
//#define     BAUD     0xFD00      // 4800bps @ 11.0592MHz
//#define     BAUD     0xFE80      // 9600bps @ 11.0592MHz
//#define     BAUD     0xFF40      //19200bps @ 11.0592MHz
//#define     BAUD     0xFFA0      //38400bps @ 11.0592MHz

//#define     BAUD     0xEC00      // 1200bps @ 18.432MHz
//#define     BAUD     0xF600      // 2400bps @ 18.432MHz
//#define     BAUD     0xFB00      // 4800bps @ 18.432MHz
//#define     BAUD     0xFD80      // 9600bps @ 18.432MHz
//#define     BAUD     0xFEC0      //19200bps @ 18.432MHz
#define     BAUD     0xFF60      //38400bps @ 18.432MHz

//#define     BAUD     0xE800      // 1200bps @ 22.1184MHz
//#define     BAUD     0xF400      // 2400bps @ 22.1184MHz
//#define     BAUD     0xFA00      // 4800bps @ 22.1184MHz
//#define     BAUD     0xFD00      // 9600bps @ 22.1184MHz
//#define     BAUD     0xFE80      //19200bps @ 22.1184MHz
//#define     BAUD     0xFF40      //38400bps @ 22.1184MHz
//#define     BAUD     0xFF80      //57600bps @ 22.1184MHz

sfr         AUXR = 0x8E;
sbit        RXB = P3^0;             //define UART TX/RX port
sbit        TXB = P3^1;
```

```
typedef     bit BOOL;
typedef     unsigned char BYTE;
typedef     unsigned int WORD;

BYTE        TBUF, RBUF;
BYTE        TDAT, RDAT;
BYTE        TCNT, RCNT;
BYTE        TBIT, RBIT;
BOOL        TING, RING;
BOOL        TEND, REND;

void UART_INIT();
BYTE        t, r;
BYTE        buf[16];

void main()
{
    TMOD = 0x00;                    //timer0 in 16-bit auto reload mode
    AUXR = 0x80;                    //timer0 working at 1T mode
    TL0 = BAUD;
    TH0 = BAUD>>8;                  //initial timer0 and set reload value
    TR0 = 1;                        //tiemr0 start running
    ET0 = 1;                        //enable timer0 interrupt
    PT0 = 1;                        //improve timer0 interrupt priority
    EA = 1;                         //open global interrupt switch

    UART_INIT();

    while (1)
    {                               //user's function
        if (REND)
        {
            REND = 0;
            buf[r++ & 0x0f] = RBUF;
        }
        if (TEND)
        {
            if (t != r)
            {
                TEND = 0;
                TBUF = buf[t++ & 0x0f];
                TING = 1;
            }
        }
    }
}
```

```
//---------------------------------------
//Timer interrupt routine for UART
void tm0() interrupt 1 using 1
{
    if (RING)
    {
        if (--RCNT == 0)
        {
            RCNT = 3;                   //reset send baudrate counter
            if (--RBIT == 0)
            {
                RBUF = RDAT;            //save the data to RBUF
                RING = 0;               //stop receive
                REND = 1;               //set receive completed flag
            }
            else
            {
                RDAT >>= 1;
                if (RXB) RDAT |= 0x80;  //shift RX data to RX buffer
            }
        }
    }
    else if (!RXB)
    {
        RING = 1;                       //set start receive flag
        RCNT = 4;                       //initial receive baudrate counter
        RBIT = 9;                       //initial receive bit number (8 data bits + 1 stop bit)
    }
    if (--TCNT == 0)
    {
        TCNT = 3;                       //reset send baudrate counter
        if (TING)                       //judge whether sending
        {
            if (TBIT == 0)
            {
                TXB = 0;                //send start bit
                TDAT = TBUF;            //load data from TBUF to TDAT
                TBIT = 9;               //initial send bit number (8 data bits + 1 stop bit)
            }
            else
            {
                TDAT >>= 1;             //shift data to CY
                if (--TBIT == 0)
                {
                    TXB = 1;
                    TING = 0;           //stop send
```

```
                    TEND = 1;               //set send completed flag
                }
                else
                {
                    TXB = CY;               //write CY to TX port
                }
            }
        }
    }
}
//-----------------------------------------
//initial UART module variable
void UART_INIT()
{
    TING = 0;
    RING = 0;
    TEND = 1;
    REND = 0;
    TCNT = 0;
    RCNT = 0;
}
```

2. 汇编程序：

```
/* --- STC15Fxx 系列软件模拟串口举例----------------------------------------------------*/
//本示例在 Keil 开发环境下请选择 Intel 的 8058 芯片型号进行编译
//假定测试芯片的工作频率为 18.432MHz
;-----------------------------------------
;define    baudrate const
;BAUD = 65536 - FOSC/3/BAUDRATE/M (1T:M=1; 12T:M=12)
;NOTE: (FOSC/3/BAUDRATE) must be greater then 75, (RECOMMEND GREATER THEN 100)
;BAUD       EQU     0F400H          ; 1200bps @ 11.0592MHz
;BAUD       EQU     0FA00H          ; 2400bps @ 11.0592MHz
;BAUD       EQU     0FD00H          ; 4800bps @ 11.0592MHz
;BAUD       EQU     0FE80H          ; 9600bps @ 11.0592MHz
;BAUD       EQU     0FF40H          ;19200bps @ 11.0592MHz
;BAUD       EQU     0FFA0H          ;38400bps @ 11.0592MHz
;BAUD       EQU     0FFC0H          ;57600bps @ 11.0592MHz

;BAUD       EQU     0EC00H          ; 1200bps @ 18.432MHz
;BAUD       EQU     0F600H          ; 2400bps @ 18.432MHz
;BAUD       EQU     0FB00H          ; 4800bps @ 18.432MHz
;BAUD       EQU     0FD80H          ; 9600bps @ 18.432MHz
;BAUD       EQU     0FEC0H          ;19200bps @ 18.432MHz
;BAUD       EQU     0FF60H          ;38400bps @ 18.432MHz
BAUD        EQU     0FF95H          ;57600bps @ 18.432MHz
```

```
;BAUD       EQU     0E800H          ; 1200bps @ 22.1184MHz
;BAUD       EQU     0F400H          ; 2400bps @ 22.1184MHz
;BAUD       EQU     0FA00H          ; 4800bps @ 22.1184MHz
;BAUD       EQU     0FD00H          ; 9600bps @ 22.1184MHz
;BAUD       EQU     0FE80H          ;19200bps @ 22.1184MHz
;BAUD       EQU     0FF40H          ;38400bps @ 22.1184MHz
;BAUD       EQU     0FF80H          ;57600bps @ 22.1184MHz

;-------------------------------------------
;define     UART TX/RX port

RXB         BIT     P3.0
TXB         BIT     P3.1
;-------------------------------------------
;define     SFR

AUXR        DATA    8EH
;-------------------------------------------
;define     UART module variable

TBUF        DATA        08H         ;(R0) ready send data buffer (USER WRITE ONLY)
RBUF        DATA        09H         ;(R1) received data buffer (UAER READ ONLY)
TDAT        DATA        0AH         ;(R2) sending data buffer (RESERVED FOR UART MODULE)
RDAT        DATA        0BH         ;(R3) receiving data buffer (RESERVED FOR UART MODULE)
TCNT        DATA        0CH         ;(R4) send baudrate counter (RESERVED FOR UART MODULE)
RCNT        DATA        0DH         ;(R5)receive baudrate counter (RESERVED FOR UART MODULE)
TBIT        DATA        0EH         ;(R6) send bit counter (RESERVED FOR UART MODULE)
RBIT        DATA        0FH         ;(R7) receive bit counter (RESERVED FOR UART MODULE)

TING        BIT         20H.0       ;sending flag (USER WRITE "1" TO TRIGGER SEND DATA,
                                    ;CLEAR BY MODULE)
RING        BIT         20H.1       ;receiving flag (RESERVED FOR UART MODULE)
TEND        BIT         20H.2       ;sent flag (SET BY MODULE AND SHOULD USER CLEAR)
REND        BIT         20H.3       ;received flag (SET BY MODULE AND SHOULD USER CLEAR)

RPTR        DATA        21H         ;circular queue read pointer
WPTR        DATA        22H         ;circular queue write pointer
BUFFER      DATA        23H         ;circular queue buffer (16 bytes)
;-------------------------------------------
    ORG     0000H
    LJMP    RESET
;-------------------------------------------
;Timer0 interrupt routine for UART

    ORG     000BH
```

```
    PUSH    ACC                 ;4 save ACC
    PUSH    PSW                 ;4 save PSW

    MOV     PSW, #08H           ;3 using register group 1
L_UARTSTART:
    JB      RING, L_RING        ;4 judge whether receiving
    JB      RXB, L_REND         ;check start signal
L_RSTART:
    SETB    RING                ;set start receive flag
    MOV     R5,  #4             ;initial receive baudrate counter
    MOV     R7,  #9             ;initial receive bit number (8 data bits + 1 stop bit)
    SJMP    L_REND              ;end this time slice
L_RING:
    DJNZ    R5,  L_REND         ;4 judge whether sending
    MOV     R5,  #3             ;2 reset send baudrate counter
L_RBIT:
    MOV     C, RXB              ;3 read RX port data
    MOV     A, R3               ;1 and shift it to RX buffer
    RRC     A                   ;1
    MOV     R3, A               ;2
    DJNZ    R7, L_REND          ;4 judge whether the data have receive completed
L_RSTOP:
    RLC     A                   ; shift out stop bit
    MOV     R1, A               ; save the data to RBUF
    CLR     RING                ; stop receive
    SETB    REND                ; set receive completed flag
L_REND:
;------------------
L_TING:
    DJNZ    R4, L_TEND          ;4 check send baudrate counter
    MOV     R4,  #3             ;2 reset it
    JNB     TING, L_TEND        ;4 judge whether sending
    MOV     A, R6               ;1 detect the sent bits
    JNZ     L_TBIT              ;3 "0" means start bit not sent
L_TSTART:
    CLR     TXB                 ; send start bit
    MOV     TDAT, R0            ; load data from TBUF to TDAT
    MOV     R6,  #9             ; initial send bit number (8 data bits + 1 stop bit)
    JMP     L_TEND              ; end this time slice
L_TBIT:
    MOV     A, R2               ;1 read data in TDAT
    SETB    C                   ;1 shift in stop bit
    RRC     A                   ;1 shift data to CY
    MOV     R2, A               ;2 update TDAT
    MOV     TXB, C              ;4 write CY to TX port
    DJNZ    R6, L_TEND          ;4 judge whether the data have send completed
```

```
L_TSTOP:
    CLR     TING                ; stop send
    SETB    TEND                ; set send completed flag
L_TEND:
;------------------
L_UARTEND:
    POP     PSW                 ;3 restore PSW
    POP     ACC                 ;3 restore ACC
    RETI                        ;4 (69)
;----------------------------------------
;initial UART module variable
UART_INIT:
    CLR     TING
    CLR     RING
    SETB    TEND
    CLR     REND
    CLR     A
    MOV     TCNT, A
    MOV     RCNT, A
    RET
;----------------------------------------
;main program entry
RESET:
    MOV     R0, #7FH            ;clear RAM
    CLR     A
    MOV     @R0, A
    DJNZ    R0, $-1
    MOV     SP, #7FH            ;initial SP
;------------------
;system initial
    MOV     TMOD, #00H          ;timer0 in 16-bit auto reload mode
    MOV     AUXR, #80H          ;timer0 working at 1T mode
    MOV     TL0, #LOW BAUD      ;initial timer0 and
    MOV     TH0, #HIGH BAUD     ;set reload value
    SETB    TR0                 ;tiemr0 start running
    SETB    ET0                 ;enable timer0 interrupt
    SETB    PT0                 ;improve timer0 interrupt priority
    SETB    EA                  ;open global interrupt switch
    LCALL   UART_INIT
;----------------------------------------
START:
    JNB     REND, CHECKREND     ;if (REND)
    CLR     REND                ;{
    MOV     A, RPTR             ; REND = 0;
    INC     RPTR                ; BUFFER[RPTR++ & 0xf] = RBUF;
    ANL     A, #0FH             ;}
```

```
    ADD     A, #BUFFER
    MOV     R0, A
    MOV     @R0, RBUF
CHECKREND:
    JNB     TEND, START             ;if (TEND)
    MOV     A, RPTR                 ;{
    XRL     A, WPTR                 ; if (WPTR != REND)
    JZ      START                   ; {
    CLR     TEND                    ; TEND = 0;
    MOV     A, WPTR                 ; TBUF = BUFFER[WPTR++ & 0xf];
    INC     WPTR                    ; TING = 1;
    ANL     A, #0FH                 ; }
    ADD     A, #BUFFER              ;}
    MOV     R0, A
    MOV     TBUF, @R0
    SETB    TING
    SJMP    START

;----------------------------------------
END
```

19.17 用 T2 结合 INT4 模拟一个半双工串口的测试程序(C 及汇编)

1.C 程序:

```
/*----使用 STC15 系列的 Timer2 做的模拟串口。P3.0 接收，P3.1 发送，半双工
假定测试芯片的工作频率为 22118400Hz，时钟为 5.5296MHz ~ 35MHz
波特率高，则时钟也要选高，优先使用 22.1184MHz，11.0592MHz
测试方法：上位机发送数据，MCU 收到数据后原样返回.
串口固定设置：1 位起始位，8 位数据位，1 位停止位，波特率在范围如下
1200 ~115200      bps @ 33.1776MHz
600 ~115200       bps @ 22.1184MHz
600 ~76800        bps @ 18.4320MHz
300 ~57600        bps @ 11.0592MHz
150 ~19200        bps @ 5.5296MHz
-----------------------------------------------------------------------------------------*/
#include <reg52.h>

#define   MAIN_Fosc             22118400UL      //定义主时钟
#define   UART3_Baudrate        115200UL        //定义波特率
#define   RX_Lenth              32              //接收长度

#define   UART3_BitTime         (MAIN_Fosc / UART3_Baudrate)

typedef   unsigned char         u8;
```

```
typedef   unsigned int        u16;
typedef   unsigned long       u32;

sfr       IE2 = 0xAF;
sfr       AUXR = 0x8E;
sfr       INT_CLKO = 0x8F;
sfr       T2H = 0xD6;
sfr       T2L = 0xD7;

u8        Tx3_read;                     //发送读指针
u8        Rx3_write;                    //接收写指针
u8        idata buf3[RX_Lenth];         //接收缓冲

u16       RxTimeOut;
bit       B_RxOk;                       //接收结束标志
//=====================  模拟串口相关==========================
sbit      P_RX3 = P3^0;                 //定义模拟串口接收 IO
sbit      P_TX3 = P3^1;                 //定义模拟串口发送 IO

u8        Tx3_DAT;                      //发送移位变量, 用户不可见
u8        Rx3_DAT;                      //接收移位变量, 用户不可见
u8        Tx3_BitCnt;                   //发送数据的位计数器, 用户不可见
u8        Rx3_BitCnt;                   //接收数据的位计数器, 用户不可见
u8        Rx3_BUF;                      //接收到的字节, 用户读取
u8        Tx3_BUF;                      //要发送的字节, 用户写入
bit       Rx3_Ring;                     //正在接收标志, 低层程序使用, 用户程序不可见
bit       Tx3_Ting;                     //正在发送标志, 用户置 1 请求发送, 底层发送完成清 0
bit       RX3_End;                      //接收到一个字节, 用户查询  并清 0
//=============================================================
void UART_Init(void);
/*********************  主函数  **************************/
void main(void)
{
    UART_Init();                                    //PCA 初始化
    EA = 1;
    while (1)                                       //user's function
    {
        if (RX3_End)                                //检测是否收到一个字节
        {
            RX3_End = 0;                            //清除标志
            buf3[Rx3_write] = Rx3_BUF;              //写入缓冲
            if(++Rx3_write >= RX_Lenth) Rx3_write = 0;   //指向下一个位置, 溢出检测
            RxTimeOut = 1000;                       //装载超时时间
        }
        if(RxTimeOut != 0)                          //超时时间是否非 0?
        {
```

```
            if(--RxTimeOut == 0)                        //(超时时间  - 1) == 0?
            {
                B_RxOk = 1;
                AUXR &= ~(1<<4);                        //Timer2  停止运行
                INT_CLKO &= ~(1 << 6);                  //禁止 INT4 中断
                T2H = (65536 - UART3_BitTime) / 256;    //数据位
                T2L = (65536 - UART3_BitTime) % 256;    //数据位
                AUXR |= (1<<4);                         //Timer2  开始运行
            }
        }
        if(B_RxOk)                                      //检测是否接收 OK?
        {
            if (!Tx3_Ting)                              //检测是否发送空闲
            {
                if (Tx3_read != Rx3_write)              //检测是否收到过字符
                {
                    Tx3_BUF = buf3[Tx3_read];           //从缓冲读一个字符发送
                    Tx3_Ting = 1;                       //设置发送标志
                    if(++Tx3_read >= RX_Lenth) Tx3_read = 0;    //指向下一个位置, 溢出检测
                }
                else
                {
                    B_RxOk = 0;
                    AUXR &= ~(1<<4);                    //Timer2  停止运行
                    INT_CLKO |= (1 << 6);               //允许 INT4 中断
                }
            }
        }
    }
}
//========================================================================
// 函数: void UART_Init(void)
// 描述: UART 初始化程序.
// 参数: none
// 返回: none.
//========================================================================
void UART_Init(void)
{
    Tx3_read = 0;
    Rx3_write = 0;
    Tx3_Ting = 0;
    Rx3_Ring = 0;
    RX3_End = 0;
    Tx3_BitCnt = 0;
    RxTimeOut = 0;
    B_RxOk = 0;
```

```
    AUXR &= ~(1<<4);                    // Timer2  停止运行
    T2H = (65536 - UART3_BitTime) / 256;    //  数据位
    T2L = (65536 - UART3_BitTime) % 256;    //  数据位
    INT_CLKO |= (1 << 6);               //  允许 INT4 中断
    IE2 |= (1<<2);                      //  允许 Timer2 中断
    AUXR |= (1<<2);                     // 1T
}
//========================================================================
//  函数: void timer2_int (void) interrupt 12
//  描述: Timer2 中断处理程序.
//  参数: None
//  返回: none.
//========================================================================
void timer2_int (void) interrupt 12
{
    if(Rx3_Ring)                        //已收到起始位
    {
        if (--Rx3_BitCnt == 0)          //接收完一帧数据
        {
            Rx3_Ring = 0;               //停止接收
            Rx3_BUF = Rx3_DAT;          //存储数据到缓冲区
            RX3_End = 1;
            AUXR &= ~(1<<4);            //Timer2  停止运行
            INT_CLKO |= (1 << 6);       //允许 INT4 中断
        }
        else
        {
            Rx3_DAT >>= 1;              //把接收的单 b 数据  暂存到  RxShiftReg(接收缓冲)
            if(P_RX3) Rx3_DAT |= 0x80;  //shift RX data to RX buffer
        }
    }
    if(Tx3_Ting)                        //  不发送, 退出
    {
        if(Tx3_BitCnt == 0)             //发送计数器为 0  表明单字节发送还没开始
        {
            P_TX3 = 0;                  //发送开始位
            Tx3_DAT = Tx3_BUF;          //把缓冲的数据放到发送的 buff
            Tx3_BitCnt = 9;             //发送数据位数  (8 数据位+1 停止位)
        }
        else                            //发送计数器为非 0  正在发送数据
        {
            if (--Tx3_BitCnt == 0)      //发送计数器减为 0  表明单字节发送结束
            {
                P_TX3 = 1;              //送停止位数据
                Tx3_Ting = 0;           //发送停止
            }
```

```
            else
            {
                Tx3_DAT >>= 1;              //把最低位送到 CY(益处标志位)
                P_TX3 = CY;                 //发送一个 bit 数据
            }
        }
    }
}

/********************** INT4 中断函数 *************************/
void Ext_INT4 (void) interrupt 16
{
    AUXR &= ~(1<<4);                                        //Timer2 停止运行
    T2H = (65536 - (UART3_BitTime / 2 + UART3_BitTime)) / 256;      //起始位 + 半个数据位
    T2L = (65536 - (UART3_BitTime / 2 + UART3_BitTime)) % 256;      //起始位 + 半个数据位
    AUXR |= (1<<4);                                         //Timer2 开始运行
    Rx3_Ring = 1;                                           //标志已收到起始位
    Rx3_BitCnt = 9;                         //初始化接收的数据位数(8 个数据位+1 个停止位)

    INT_CLKO &= ~(1 << 6);                  //禁止 INT4 中断
    T2H = (65536 - UART3_BitTime) / 256;    //数据位
    T2L = (65536 - UART3_BitTime) % 256;    //数据位
}
```

2. 汇编程序：

```
;************* 功能说明 **************
;使用 STC15 系列的 Timer2 做的模拟串口。P3.0 接收，P3.1 发送，半双工
;假定测试芯片的工作频率为 22118400Hz，时钟为 5.5296MHz ~ 35MHz
;波特率高，则时钟也要选高，优先使用 22.1184MHz，11.0592MHz
;测试方法：上位机发送数据，MCU 收到数据后原样返回.
;串口固定设置：1 位起始位，8 位数据位，1 位停止位，波特率在范围如下.
;*******************************************
STACK_POIRTER   EQU     0D0H        ;堆栈开始地址
;UART3_BitTime  EQU     9216        ; 1200bps @ 11.0592MHz
                                    ;UART3_BitTime = (MAIN_Fosc / Baudrate)
;UART3_BitTime  EQU     4608        ; 2400bps @ 11.0592MHz
;UART3_BitTime  EQU     2304        ; 4800bps @ 11.0592MHz
;UART3_BitTime  EQU     1152        ; 9600bps @ 11.0592MHz
;UART3_BitTime  EQU     576         ;19200bps @ 11.0592MHz
;UART3_BitTime  EQU     288         ;38400bps @ 11.0592MHz
;UART3_BitTime  EQU     192         ;57600bps @ 11.0592MHz

;UART3_BitTime  EQU     15360       ; 1200bps @ 18.432MHz
;UART3_BitTime  EQU     7680        ; 2400bps @ 18.432MHz
;UART3_BitTime  EQU     3840        ; 4800bps @ 18.432MHz
;UART3_BitTime  EQU     1920        ; 9600bps @ 18.432MHz
```

```
;UART3_BitTime     EQU     960         ;19200bps @ 18.432MHz
;UART3_BitTime     EQU     480         ;38400bps @ 18.432MHz
;UART3_BitTime     EQU     320         ;57600bps @ 18.432MHz

;UART3_BitTime     EQU     18432       ; 1200bps @ 22.1184MHz
;UART3_BitTime     EQU     9216        ; 2400bps @ 22.1184MHz
;UART3_BitTime     EQU     4608        ; 4800bps @ 22.1184MHz
;UART3_BitTime     EQU     2304        ; 9600bps @ 22.1184MHz
;UART3_BitTime     EQU     1152        ;19200bps @ 22.1184MHz
;UART3_BitTime     EQU     576         ;38400bps @ 22.1184MHz
;UART3_BitTime     EQU     384         ;57600bps @ 22.1184MHz
UART3_BitTime      EQU     192         ;115200bps @ 22.1184MHz

;UART3_BitTime     EQU     27648       ; 1200bps @ 33.1776MHz
;UART3_BitTime     EQU     13824       ; 2400bps @ 33.1776MHz
;UART3_BitTime     EQU     6912        ; 4800bps @ 33.1776MHz
;UART3_BitTime     EQU     3456        ; 9600bps @ 33.1776MHz
;UART3_BitTime     EQU     1728        ;19200bps @ 33.1776MHz
;UART3_BitTime     EQU     864         ;38400bps @ 33.1776MHz
;UART3_BitTime     EQU     576         ;57600bps @ 33.1776MHz
;UART3_BitTime     EQU     288         ;115200bps @ 33.1776MHz

IE2                DATA    0AFH
AUXR               DATA    08EH
INT_CLKO           DATA    08FH
T2H                DATA    0D6H
T2L                DATA    0D7H
;====================== 模拟串口相关 ============================
P_RX3              BIT     P3.0        ; 定义模拟串口接收 IO
P_TX3              BIT     P3.1        ; 定义模拟串口发送 IO
Rx3_Ring           BIT     20H.0       ; 正在接收标志, 低层程序使用, 用户程序不可见
Tx3_Ting           BIT     20H.1       ; 正在发送标志, 用户置 1 请求发送, 底层发送完成清 0
RX3_End            BIT     20H.2       ; 接收到一个字节, 用户查询 并清 0
B_RxOk             BIT     20H.3       ; 接收结束标志
Tx3_DAT            DATA    30H         ; 发送移位变量, 用户不可见
Rx3_DAT            DATA    31H         ; 接收移位变量, 用户不可见
Tx3_BitCnt         DATA    32H         ; 发送数据的位计数器, 用户不可见
Rx3_BitCnt         DATA    33H         ; 接收数据的位计数器, 用户不可见
Rx3_BUF            DATA    34H         ; 接收到的字节, 用户读取
Tx3_BUF            DATA    35H         ; 要发送的字节, 用户写入
;================================================================
RxTimeOutH         DATA    36H
RxTimeOutL         DATA    37H
Tx3_read           DATA    38H         ; 发送读指针
Rx3_write          DATA    39H         ; 接收写指针
RX_Lenth           EQU     32          ; 接收长度
```

```
buf3                EQU     40H         ; 40H ~ 5FH  接收缓冲
;*******************************************************************
    ORG     00H                         ;reset
    LJMP    F_Main

    ORG     63H                         ;12 Timer2 interrupt
    LJMP    F_Timer2_Interrupt

    ORG     83H                         ;16 INT4 interrupt
    LJMP    F_INT4_Interrupt
;********************  主程序  ************************/
F_Main:
    MOV     SP, #STACK_POIRTER
    MOV     PSW, #0
    USING   0                           ;选择第 0 组 R0~R7
;================  用户初始化程序  ==================================
    LCALL   F_UART_Init                 ;UART 初始化
    SETB    EA
;==================  主循环  ==================================
L_MainLoop:
    JNB     RX3_End, L_QuitRx3          ; 检测是否收到一个字节
    CLR     RX3_End                     ; 清除标志
    MOV     A, #buf3
    ADD     A, Rx3_write
    MOV     R0, A
    MOV     @R0, Rx3_BUF                ; 写入缓冲
    MOV     RxTimeOutH, #HIGH 1000      ; 装载超时时间
    MOV     RxTimeOutL, #LOW 1000
    INC     Rx3_write                   ; 指向下一个位置
    MOV     A, Rx3_write
    CLR     C
    SUBB    A, #RX_Lenth                ; 溢出检测
    JC      L_QuitRx3
    MOV     Rx3_write, #0

L_QuitRx3:
    MOV     A, RxTimeOutL
    ORL     A, RxTimeOutH
    JZ      L_QuitTimeOut               ; 超时时间是否非 0?
    MOV     A, RxTimeOutL
    DEC     RxTimeOutL                  ; (超时时间  - 1) == 0?
    JNZ     $+4
    DEC     RxTimeOutH
    DEC     A
    ORL     A, RxTimeOutH
    JNZ     L_QuitTimeOut
```

```
    SETB    B_RxOk                          ; 超时, 标志接收完成
    ANL     AUXR, #NOT (1 SHL 4)            ; Timer2  停止运行
    ANL     INT_CLKO, #NOT (1 SHL 6)        ; 禁止 INT4 中断
    MOV     T2H, #HIGH (65536 - UART3_BitTime)  ;数据位
    MOV     T2L, #LOW (65536 - UART3_BitTime) ;
    ORL     AUXR, #(1 SHL 4)                ; Timer2  开始运行

L_QuitTimeOut:
    JNB     B_RxOk, L_QuitTx3               ; 检测是否接收 OK?
    JB      Tx3_Ting, L_QuitTx3             ; 检测是否发送空闲
    MOV     A, Tx3_read
    XRL     A, Rx3_write
    JZ      L_TxFinish                      ; 检测是否发送完毕

    MOV     A, #buf3
    ADD     A, Tx3_read
    MOV     R0, A
    MOV     Tx3_BUF, @R0                    ; 从缓冲读一个字符发送
    SETB    Tx3_Ting                        ; 设置发送标志
    INC     Tx3_read                        ; 指向下一个字符位置
    MOV     A, Tx3_read
    CLR     C
    SUBB    A, #RX_Lenth
    JC      L_QuitTx3                       ; 溢出检测
    MOV     Tx3_read, #0
    SJMP    L_QuitTx3

L_TxFinish:
    CLR     B_RxOk
    ANL     AUXR, #NOT (1 SHL 4)            ; Timer2  停止运行
    ORL     INT_CLKO, #(1 SHL 6)            ; 允许 INT4 中断

L_QuitTx3:
    LJMP    L_MainLoop
;==================== 主程序结束 ================================
;================================================================================
; 函数: F_UART_Init
; 描述: UART 初始化程序.
; 参数: none
; 返回: none.
;================================================================================
F_UART_Init:
    MOV     Tx3_read, #0
    MOV     Rx3_write, #0
    CLR     Tx3_Ting
```

```
    CLR     RX3_End
    CLR     Rx3_Ring
    MOV     Tx3_BitCnt, #0
    MOV     RxTimeOutH, #0
    MOV     RxTimeOutL, #0
    CLR     B_RxOk
    ANL     AUXR, #NOT(1 SHL 4)             ; Timer2  停止运行
    MOV     T2H, #HIGH (65536 - UART3_BitTime)  ; 数据位
    MOV     T2L, #LOW (65536 - UART3_BitTime)   ; 数据位
    ORL     INT_CLKO, #(1 SHL 6)            ; 允许 INT4 中断
    ORL     IE2, #(1 SHL 2)                 ; 允许 Timer2 中断
    ORL     AUXR, #(1 SHL 2)                ; 1T 模式

    RET
;=========================================================================
; 函数: void F_Timer2_Interrupt
; 描述: Timer2 中断处理程序.
; 参数: None
; 返回: none.
;=========================================================================
F_Timer2_Interrupt:
    PUSH    PSW
    PUSH    ACC
    JNB     Rx3_Ring, L_QuitRx              ; 已收到起始位
    DJNZ    Rx3_BitCnt, L_RxBit             ; 接收完一帧数据
    CLR     Rx3_Ring                        ; 停止接收
    MOV     Rx3_BUF, Rx3_DAT                ; 存储数据到缓冲区
    SETB    RX3_End
    ANL     AUXR, #NOT (1 SHL 4)            ; Timer2  停止运行
    ORL     INT_CLKO, #(1 SHL 6)            ; 允许 INT4 中断
    SJMP    L_QuitRx

L_RxBit:
    MOV     A, Rx3_DAT                      ; 把接收的单 b 数据  暂存到  RxShiftReg(接收缓冲)
    MOV     C, P_RX3
    RRC     A
    MOV     Rx3_DAT, A

L_QuitRx:
    JNB     Tx3_Ting, L_QuitTx              ; 不发送, 退出
    MOV     A, Tx3_BitCnt
    JNZ     L_TxData                        ; 发送计数器为 0  表明单字节发送还没开始
    CLR     P_TX3                           ; 发送开始位
    MOV     Tx3_DAT, Tx3_BUF                ; 把缓冲的数据放到发送的 buff
    MOV     Tx3_BitCnt, #9                  ; 发送数据位数  (8 数据位+1 停止位)
    SJMP    L_QuitTx
```

```
L_TxData:                                       ; 发送计数器为非 0  正在发送数据
    DJNZ    Tx3_BitCnt, L_TxBit                 ; 发送计数器减为 0  表明单字节发送结束
    SETB    P_TX3                               ; 送停止位数据
    CLR     Tx3_Ting                            ; 发送停止
    SJMP    L_QuitTx

L_TxBit:
    MOV     A, Tx3_DAT                          ; 把最低位送到 CY(益处标志位)
    RRC     A
    MOV     P_TX3, C                            ; 发送一个 bit 数据
    MOV     Tx3_DAT, A

L_QuitTx:
    POP     ACC
    POP     PSW

    RETI
;====================== INT4 中断函数 ================================
F_INT4_Interrupt:
    PUSH    PSW
    PUSH    ACC
    ANL     AUXR, #NOT(1 SHL 4)                 ;Timer2  停止运行
    MOV     T2H, #HIGH (65536 - (UART3_BitTime / 2 + UART3_BitTime))    ; 起始位 + 半个数据位
    MOV     T2L, #LOW (65536 - (UART3_BitTime / 2 + UART3_BitTime))     ; 起始位 + 半个数据位
    ORL     AUXR, #(1 SHL 4)                    ;Timer2  开始运行
    SETB    Rx3_Ring                            ; 标志已收到起始位
    MOV     Rx3_BitCnt, #9                      ; 初始化接收的数据位数(8 个数据位+1 个停止位)
    ANL     INT_CLKO, #NOT(1 SHL 6)             ; 禁止 INT4 中断
    MOV     T2H, #HIGH (65536 - UART3_BitTime)  ; 数据位
    MOV     T2L, #LOW (65536 - UART3_BitTime)   ; 数据位
    POP     ACC
    POP     PSW

    RETI

END
```

19.18 利用两路 CCP/PCA 模拟一个全双工串口的程序(C 及汇编)

1.C 程序:

```
/*----使用 STC15 系列的 PCA0 和 PCA1 做的模拟串口。PCA0 接收(P2.5)，PCA1 发送(P2.6)
//假定测试芯片的工作频率为 22118400Hz，时钟为 5.5296MHz ~ 35MHz
波特率高，则时钟也要选高，优先使用 22.1184MHz，11.0592MHz
测试方法：上位机发送数据,MCU 收到数据后原样返回.
串口固定设置：1 位起始位，8 位数据位，1 位停止位，波特率在  600 ~ 57600 bps.
1200 ~ 57600        bps @ 33.1776MHz
600 ~ 57600         bps @ 22.1184MHz
600 ~ 38400         bps @ 18.4320MHz
300 ~ 28800         bps @ 11.0592MHz
150 ~ 14400         bps @ 5.5296MHz
*****************************************/
#include <reg52.h>

#define   MAIN_Fosc             22118400UL  //定义主时钟
#define   UART3_Baudrate        57600UL     //定义波特率
#define   RX_Lenth              16          //接收长度

#define   PCA_P12_P11_P10_P37   (0<<4)
#define   PCA_P34_P35_P36_P37   (1<<4)
#define   PCA_P24_P25_P26_P27   (2<<4)

#define   PCA_Mode_Capture      0
#define   PCA_Mode_SoftTimer    0x48

#define   PCA_Clock_1T          (4<<1)
#define   PCA_Clock_2T          (1<<1)
#define   PCA_Clock_4T          (5<<1)
#define   PCA_Clock_6T          (6<<1)
#define   PCA_Clock_8T          (7<<1)
#define   PCA_Clock_12T         (0<<1)
#define   PCA_Clock_ECI         (3<<1)

#define   PCA_Rise_Active       (1<<5)
#define   PCA_Fall_Active       (1<<4)

#define   PCA_PWM_8bit          (0<<6)
#define   PCA_PWM_7bit          (1<<6)
#define   PCA_PWM_6bit          (2<<6)

#define   UART3_BitTime         (MAIN_Fosc / UART3_Baudrate)

#define   ENABLE                1
```

```
#define   DISABLE                 0

typedef   unsigned char           u8;
typedef   unsigned int            u16;
typedef   unsigned long           u32;

sfr       AUXR1 = 0xA2;
sfr       CCON = 0xD8;
sfr       CMOD = 0xD9;
sfr       CCAPM0= 0xDA;           //PCA 模块 0 的工作模式寄存器。
sfr       CCAPM = 0xDB;           //PCA 模块 1 的工作模式寄存器。
sfr       CCAPM2= 0xDC;           //PCA 模块 2 的工作模式寄存器。

sfr       CL = 0xE9;
sfr       CCAP0L = 0xEA;          //PCA 模块 0 的捕捉/比较寄存器低 8 位。
sfr       CCAP1L = 0xEB;          //PCA 模块 1 的捕捉/比较寄存器低 8 位。
sfr       CCAP2L = 0xEC;          //PCA 模块 2 的捕捉/比较寄存器低 8 位。

sfr       CH = 0xF9;
sfr       CCAP0H = 0xFA;          //PCA 模块 0 的捕捉/比较寄存器高 8 位。
sfr       CCAP1H = 0xFB;          //PCA 模块 1 的捕捉/比较寄存器高 8 位。
sfr       CCAP2H = 0xFC;          //PCA 模块 2 的捕捉/比较寄存器高 8 位。

sbit      CCF0 = CCON^0;          //PCA 模块 0 中断标志，由硬件置位，必须由软件清 0。
sbit      CCF1 = CCON^1;          //PCA 模块 1 中断标志，由硬件置位，必须由软件清 0。
sbit      CCF2 = CCON^2;          //PCA 模块 2 中断标志，由硬件置位，必须由软件清 0。
sbit      CR = CCON^6;            //1: 允许 PCA 计数器计数， 0: 禁止计数。
sbit      CF = CCON^7;            //PCA 计数器溢出（CH, CL 由 FFFFH 变为 0000H）标志。
                                  //PCA 计数器溢出后由硬件置位，必须由软件清 0。
sbit      PPCA = IP^7;            //PCA 中断 优先级设定位

u16       CCAP0_tmp;
u16       CCAP1_tmp;
u8        Tx3_read;               //发送读指针
u8        Rx3_write;              //接收写指针
u8        idata buf3[RX_Lenth];   //接收缓冲
//=====================  模拟串口相关============================
sbit      P_RX3 = P2^5;           //定义模拟串口接收 IO
sbit      P_TX3 = P2^6;           //定义模拟串口发送 IO
u8        Tx3_DAT;                //发送移位变量, 用户不可见
u8        Rx3_DAT;                //接收移位变量, 用户不可见
u8        Tx3_BitCnt;             //发送数据的位计数器, 用户不可见
u8        Rx3_BitCnt;             //接收数据的位计数器, 用户不可见
u8        Rx3_BUF;                //接收到的字节, 用户读取
u8        Tx3_BUF;                //要发送的字节, 用户写入
bit       Rx3_Ring;               //正在接收标志, 低层程序使用, 用户程序不可见
```

```
bit     Tx3_Ting;                       //正在发送标志, 用户置 1 请求发送, 底层发送完成清 0
bit     RX3_End;                        //接收到一个字节, 用户查询  并清 0
//=========================================================
void PCA_Init(void);
/********************  主函数  **************************/
void main(void)
{
    PCA_Init();                                             //PCA 初始化
    EA = 1;
    Tx3_read = 0;
    Rx3_write = 0;
    Tx3_Ting = 0;
    Rx3_Ring = 0;
    RX3_End = 0;
    Tx3_BitCnt = 0;
    while (1)                                               //user's function
    {
        if (RX3_End)                                        // 检测是否收到一个字节
        {
            RX3_End = 0;                                    // 清除标志
            buf3[Rx3_write] = Rx3_BUF;                      // 写入缓冲
            if(++Rx3_write >= RX_Lenth) Rx3_write = 0;       // 指向下一个位置, 溢出检测
        }
        if (!Tx3_Ting)                                      // 检测是否发送空闲
        {
            if (Tx3_read != Rx3_write)                      // 检测是否收到过字符
            {
                Tx3_BUF = buf3[Tx3_read];                   // 从缓冲读一个字符发送
                Tx3_Ting = 1;                               // 设置发送标志
                if(++Tx3_read >= RX_Lenth) Tx3_read = 0;    // 指向下一个位置, 溢出检测
            }
        }
    }
}
//========================================================================
// 函数: void PCA_Init(void)
// 描述: PCA 初始化程序.
// 参数: none
// 返回: none.
//========================================================================
void PCA_Init(void)
{
    CR = 0;
    CCAPM0 = (PCA_Mode_Capture | PCA_Fall_Active | ENABLE);   //16 位下降沿捕捉中断模式
    CCAPM1 = PCA_Mode_SoftTimer | ENABLE;
    CCAP1_tmp = UART3_BitTime;
```

```
    CCAP1L = (u8)CCAP1_tmp;                                 //将影射寄存器写入捕获寄存器,先写 CCAP0L
    CCAP1H = (u8)(CCAP1_tmp >> 8);                          //后写 CCAP0H
    CH = 0;
    CL = 0;
    AUXR1 = (AUXR1 & ~(3<<4)) | PCA_P24_P25_P26_P27;        //切换 I/O 口
    CMOD = (CMOD & ~(7<<1)) | PCA_Clock_1T;                 //选择时钟源
    PPCA = 1;                                               // 高优先级中断
    CR = 1;                                                 // 运行 PCA 定时器
}
//========================================================================
// 函数: void PCA_Handler (void) interrupt 7
// 描述: PCA 中断处理程序.
// 参数: None
// 返回: none.
//========================================================================
void PCA_Handler (void) interrupt 7
{
    if(CCF0)                                                //PCA 模块 0 中断
    {
        CCF0 = 0;                                           //清 PCA 模块 0 中断标志
        if(Rx3_Ring)                                        //已收到起始位
        {
            if (--Rx3_BitCnt == 0)                          //接收完一帧数据
            {
                Rx3_Ring = 0;                               //停止接收
                Rx3_BUF = Rx3_DAT;                          //存储数据到缓冲区
                RX3_End = 1;
                CCAPM0 = (PCA_Mode_Capture | PCA_Fall_Active | ENABLE);
                                                            //16 位下降沿捕捉中断模式
            }
            else
            {
                Rx3_DAT >>= 1;                  //把接收的单 b 数据暂存到 RxShiftReg(接收缓冲)
                if(P_RX3) Rx3_DAT |= 0x80;                  //shift RX data to RX buffer
                CCAP0_tmp += UART3_BitTime;                 //数据位时间
                CCAP0L = (u8)CCAP0_tmp;         //将影射寄存器写入捕获寄存//器，先写 CCAP0L
                CCAP0H = (u8)(CCAP0_tmp >> 8);              //后写 CCAP0H
            }
        }
        else
        {
            CCAP0_tmp = ((u16)CCAP0H << 8) + CCAP0L;            //读捕捉寄存器
            CCAP0_tmp += (UART3_BitTime / 2 + UART3_BitTime);   //起始位 + 半个数据位
            CCAP0L = (u8)CCAP0_tmp;             //将影射寄存器写入捕获寄存器，先写 CCAP0L
            CCAP0H = (u8)(CCAP0_tmp >> 8);                  //后写 CCAP0H
            CCAPM0 = (PCA_Mode_SoftTimer | ENABLE);         //16 位软件定时中断模式
```

```
            Rx3_Ring = 1;                                   //标志已收到起始位
            Rx3_BitCnt = 9;                                 //初始化接收的数据位数(8 个数据位+1 个停止位)
        }
    }

    if(CCF1)                                                //PCA 模块 1 中断, 16 位软件定时中断模式
    {
        CCF1 = 0;                                           //清 PCA 模块 1 中断标志
        CCAP1_tmp += UART3_BitTime;
        CCAP1L = (u8)CCAP1_tmp;                             //将影射寄存器写入捕获寄存器，先写 CCAP0L
        CCAP1H = (u8)(CCAP1_tmp >> 8);                      //后写 CCAP0H
        if(Tx3_Ting)                                        // 不发送, 退出
        {
            if(Tx3_BitCnt == 0)                             //发送计数器为 0  表明单字节发送还没开始
            {
                P_TX3 = 0;                                  //发送开始位
                Tx3_DAT = Tx3_BUF;                          //把缓冲的数据放到发送的 buff
                Tx3_BitCnt = 9;                             //发送数据位数  (8 数据位+1 停止位)
            }
            else                                            //发送计数器为非 0  正在发送数据
            {
                if (--Tx3_BitCnt == 0)                      //发送计数器减为 0  表明单字节发送结束
                {
                    P_TX3 = 1;                              //送停止位数据
                    Tx3_Ting = 0;                           //发送停止
                }
                else
                {
                    Tx3_DAT >>= 1;                          //把最低位送到  CY(益处标志位)
                    P_TX3 = CY;                             //发送一个 bit 数据
                }
            }
        }
    }
}
```

2. 汇编程序：

```
;************* 功能说明 **************
;使用 STC15 系列的 PCA0 和 PCA1 做的模拟串口。PCA0 接收(P2.5)，PCA1 发送(P2.6)
;假定测试芯片的工作频率为 22118400Hz，时钟为 5.5296MHz ~ 35MHz
;波特率高，则时钟也要选高，优先使用 22.1184MHz，11.0592MHz
;测试方法：上位机发送数据，MCU 收到数据后原样返回.
;串口固定设置：1 位起始位，8 位数据位，1 位停止位
;*******************************************
STACK_POIRTER           EQU     0D0H            ;堆栈开始地址
```

```
;UART3_BitTime      EQU     9216        ; 1200bps @ 11.0592MHz
                                        ;UART3_BitTime = (MAIN_Fosc / Baudrate)
;UART3_BitTime      EQU     4608        ; 2400bps @ 11.0592MHz
;UART3_BitTime      EQU     2304        ; 4800bps @ 11.0592MHz
;UART3_BitTime      EQU     1152        ; 9600bps @ 11.0592MHz
;UART3_BitTime      EQU     576         ;19200bps @ 11.0592MHz
;UART3_BitTime      EQU     288         ;38400bps @ 11.0592MHz

;UART3_BitTime      EQU     15360       ; 1200bps @ 18.432MHz
;UART3_BitTime      EQU     7680        ; 2400bps @ 18.432MHz
;UART3_BitTime      EQU     3840        ; 4800bps @ 18.432MHz
;UART3_BitTime      EQU     1920        ; 9600bps @ 18.432MHz
;UART3_BitTime      EQU     960         ;19200bps @ 18.432MHz
;UART3_BitTime      EQU     480         ;38400bps @ 18.432MHz
;UART3_BitTime      EQU     320         ;57600bps @ 18.432MHz

;UART3_BitTime      EQU     18432       ; 1200bps @ 22.1184MHz
;UART3_BitTime      EQU     9216        ; 2400bps @ 22.1184MHz
;UART3_BitTime      EQU     4608        ; 4800bps @ 22.1184MHz
;UART3_BitTime      EQU     2304        ; 9600bps @ 22.1184MHz
;UART3_BitTime      EQU     1152        ;19200bps @ 22.1184MHz
;UART3_BitTime      EQU     576         ;38400bps @ 22.1184MHz
UART3_BitTime       EQU     384         ;57600bps @ 22.1184MHz

;UART3_BitTime      EQU     27648       ; 1200bps @ 33.1776MHz
;UART3_BitTime      EQU     13824       ; 2400bps @ 33.1776MHz
;UART3_BitTime      EQU     6912        ; 4800bps @ 33.1776MHz
;UART3_BitTime      EQU     3456        ; 9600bps @ 33.1776MHz
;UART3_BitTime      EQU     1728        ;19200bps @ 33.1776MHz
;UART3_BitTime      EQU     864         ;38400bps @ 33.1776MHz
;UART3_BitTime      EQU     576         ;57600bps @ 33.1776MHz
;UART3_BitTime      EQU     288         ;115200bps @ 33.1776MHz

PCA_P12_P11_P10_P37 EQU     (0 SHL 4)
PCA_P34_P35_P36_P37 EQU     (1 SHL 4)
PCA_P24_P25_P26_P27 EQU     (2 SHL 4)
PCA_Mode_Capture    EQU     0
PCA_Mode_SoftTimer  EQU     048H
PCA_Clock_1T        EQU     (4 SHL 1)
PCA_Clock_2T        EQU     (1 SHL 1)
PCA_Clock_4T        EQU     (5 SHL 1)
PCA_Clock_6T        EQU     (6 SHL 1)
PCA_Clock_8T        EQU     (7 SHL 1)
PCA_Clock_12T       EQU     (0 SHL 1)
PCA_Clock_ECI       EQU     (3 SHL 1)
PCA_Rise_Active     EQU     (1 SHL 5)
```

```
PCA_Fall_Active         EQU     (1 SHL 4)

ENABLE                  EQU     1

AUXR1                   DATA    0xA2
CCON                    DATA    0xD8
CMOD                    DATA    0xD9
CCAPM0                  DATA    0xDA        ; PCA 模块 0 的工作模式寄存器。
CCAPM1                  DATA    0xDB        ; PCA 模块 1 的工作模式寄存器。
CCAPM2                  DATA    0xDC        ; PCA 模块 2 的工作模式寄存器。

CL                      DATA    0xE9
CCAP0L                  DATA    0xEA        ; PCA 模块 0 的捕捉/比较寄存器低 8 位。
CCAP1L                  DATA    0xEB        ; PCA 模块 1 的捕捉/比较寄存器低 8 位。
CCAP2L                  DATA    0xEC        ; PCA 模块 2 的捕捉/比较寄存器低 8 位。

CH                      DATA    0xF9
CCAP0H                  DATA    0xFA        ; PCA 模块 0 的捕捉/比较寄存器高 8 位。
CCAP1H                  DATA    0xFB        ; PCA 模块 1 的捕捉/比较寄存器高 8 位。
CCAP2H                  DATA    0xFC        ; PCA 模块 2 的捕捉/比较寄存器高 8 位。

CCF0                    BIT     CCON.0      ; PCA 模块 0 中断标志，由硬件置位，必须由软件清 0。
CCF1                    BIT     CCON.1      ; PCA 模块 1 中断标志，由硬件置位，必须由软件清 0。
CCF2                    BIT     CCON.2      ; PCA 模块 2 中断标志，由硬件置位，必须由软件清 0。
CR                      BIT     CCON.6      ; 1: 允许 PCA 计数器计数， 0: 禁止计数。
CF                      BIT     CCON.7      ; PCA 计数器溢出（CH, CL 由 FFFFH 变为 0000H）标志。
                                            ;PCA 计数器溢出后由硬件置位，必须由软件清 0。
PPCA                    BIT     IP.7        ; PCA 中断 优先级设定位
;====================== 模拟串口相关 ============================
P_RX3                   BIT     P2.5        ; 定义模拟串口接收 IO
P_TX3                   BIT     P2.6        ; 定义模拟串口发送 IO

Rx3_Ring                BIT     20H.0       ; 正在接收标志, 低层程序使用, 用户程序不可见
Tx3_Ting                BIT     20H.1       ; 正在发送标志, 用户置 1 请求发送, 底层发送完成清 0
RX3_End                 BIT     20H.2       ; 接收到一个字节, 用户查询 并清 0

Tx3_DAT                 DATA    30H         ; 发送移位变量, 用户不可见
Rx3_DAT                 DATA    31H         ; 接收移位变量, 用户不可见
Tx3_BitCnt              DATA    32H         ; 发送数据的位计数器, 用户不可见
Rx3_BitCnt              DATA    33H         ; 接收数据的位计数器, 用户不可见
Rx3_BUF                 DATA    34H         ; 接收到的字节, 用户读取
Tx3_BUF                 DATA    35H         ; 要发送的字节, 用户写入
;================================================================
Tx3_read                DATA    36H         ; 发送读指针
Rx3_write               DATA    37H         ; 接收写指针
```

```
RX_Lenth                EQU     16              ; 接收长度
buf3                    EQU     40H             ; 40H ~ 4FH  接收缓冲
;*******************************************************************
    ORG     00H                                 ;reset
    LJMP    F_Main
    ORG     3BH                                 ;7 PCA interrupt
    LJMP    F_PCA_Interrupt
;********************  主程序  ***************************/
F_Main:
    MOV     SP, #STACK_POIRTER
    MOV     PSW, #0
    USING   0                                   ;选择第 0 组 R0~R7
;================  用户初始化程序  ================================
    LCALL   F_PCA_Init                          ;PCA 初始化
    SETB    EA
    MOV     Tx3_read, #0
    MOV     Rx3_write, #0
    CLR     Tx3_Ting
    CLR     RX3_End
    CLR     Rx3_Ring
    MOV     Tx3_BitCnt, #0
;==================  主循环  ================================
L_MainLoop:
    JNB     RX3_End, L_QuitRx3                  ; 检测是否收到一个字节
    CLR     RX3_End                             ; 清除标志
    MOV     A, #buf3
    ADD     A, Rx3_write
    MOV     R0, A
    MOV     @R0, Rx3_BUF                        ; 写入缓冲
    INC     Rx3_write                           ; 指向下一个位置
    MOV     A, Rx3_write
    CLR     C
    SUBB    A, #RX_Lenth                        ; 溢出检测
    JC      L_QuitRx3
    MOV     Rx3_write, #0
L_QuitRx3:
    JB      Tx3_Ting, L_QuitTx3                 ;检测是否发送空闲
    MOV     A, Tx3_read
    XRL     A, Rx3_write
    JZ      L_QuitTx3                           ;检测是否收到过字符
    MOV     A, #buf3
    ADD     A, Tx3_read
    MOV     R0, A
    MOV     Tx3_BUF, @R0                        ; 从缓冲读一个字符发送
    SETB    Tx3_Ting                            ; 设置发送标志
    INC     Tx3_read                            ; 指向下一个字符位置
```

```
    MOV     A, Tx3_read
    CLR     C
    SUBB    A, #RX_Lenth
    JC      L_QuitTx3                       ; 溢出检测
    MOV     Tx3_read, #0
L_QuitTx3:
    SJMP    L_MainLoop
;==================== 主程序结束 ================================
;================================================================
; 函数: F_PCA_Init
; 描述: PCA 初始化程序.
; 参数: none
; 返回: none.
;================================================================
F_PCA_Init:
    CLR     CR
    MOV     CCAPM0, #(PCA_Mode_Capture OR PCA_Fall_Active OR ENABLE)
                                            ; 16 位下降沿捕捉中断模式
    MOV     CCAPM1, #(PCA_Mode_SoftTimer OR ENABLE)     ; 16 位软件定时器, 中断模式
    MOV     CCAP1L, #LOW UART3_BitTime      ; 将影射寄存器写入捕获寄存器，先写 CCAP0L
    MOV     CCAP1H, #HIGH UART3_BitTime     ; 后写 CCAP0H
    MOV     CH, #0
    MOV     CL, #0
    MOV     A, AUXR1
    ANL     A, # NOT(3 SHL 4)
    ORL     A, #PCA_P24_P25_P26_P27         ;切换 I/O 口
    MOV     AUXR1, A
    ANL     A, #NOT(7 SHL 1)
    ORL     A, #PCA_Clock_1T                ;选择时钟源
    MOV     CMOD, A
    SETB    PPCA                            ; 高优先级中断
    SETB    CR                              ; 运行 PCA 定时器
    RET
;================================================================
; 函数: F_PCA_Interrupt
; 描述: PCA 中断处理程序.
; 参数: None
; 返回: none.
;================================================================
F_PCA_Interrupt:
    PUSH    PSW
    PUSH    ACC
;=============== PCA 模块 0 中断 ===================
    JNB     CCF0, L_QuitPCA0                ; PCA 模块 0 中断
    CLR     CCF0                            ; 清 PCA 模块 0 中断标志
    JNB     Rx3_Ring, L_Rx3_Start           ; 已收到起始位
```

```
    DJNZ    Rx3_BitCnt, L_RxBit             ; 接收完一帧数据
    CLR     Rx3_Ring                        ; 停止接收

    MOV     Rx3_BUF, Rx3_DAT                ; 存储数据到缓冲区
    SETB    RX3_End
    MOV     CCAPM0, #(PCA_Mode_Capture OR PCA_Fall_Active OR ENABLE)
                                            ; 16 位下降沿捕捉中断模式
    SJMP    L_QuitPCA0

L_RxBit:
    MOV     A, Rx3_DAT                      ; 把接收的单 b 数据暂存到 RxShiftReg(接收缓冲)
    MOV     C, P_RX3
    RRC     A
    MOV     Rx3_DAT, A
    MOV     A, CCAP0L ;
    ADD     A, #LOW UART3_BitTime           ; 数据位时间
    MOV     CCAP0L, A                       ; 将影射寄存器写入捕获寄存器，先写 CCAP0L
    MOV     A, CCAP0H                       ; 数据位时间
    ADDC    A, #HIGH UART3_BitTime ;
    MOV     CCAP0H, A                       ; 后写 CCAP0H
    SJMP    L_QuitPCA0

L_Rx3_Start:
    MOV     CCAPM0, #(PCA_Mode_SoftTimer OR ENABLE)     ; 16 位软件定时中断模式
    MOV     A, CCAP0L                       ; 数据位时间
    ADD     A, #LOW (UART3_BitTime / 2 + UART3_BitTime) ;
    MOV     CCAP0L, A                       ; 将影射寄存器写入捕获寄存器，先写 CCAP0L
    MOV     A, CCAP0H                       ; 数据位时间
    ADDC    A, #HIGH (UART3_BitTime / 2 + UART3_BitTime) ;
    MOV     CCAP0H, A                       ; 后写 CCAP0H
    SETB    Rx3_Ring                        ; 标志已收到起始位
    MOV     Rx3_BitCnt, #9                  ; 初始化接收的数据位数(8 个数据位+1 个停止位)

L_QuitPCA0:
;================ PCA 模块 1 中断 ====================
    JNB     CCF1, L_QuitPCA1                ; PCA 模块 1 中断, 16 位软件定时中断模式
    CLR     CCF1                            ; 清 PCA 模块 1 中断标志
    MOV     A, CCAP1L
    ADD     A, #LOW UART3_BitTime           ; 数据位时间
    MOV     CCAP1L, A                       ; 将影射寄存器写入捕获寄存器，先写 CCAP0L
    MOV     A, CCAP1H
    ADDC    A, #HIGH UART3_BitTime          ; 数据位时间
    MOV     CCAP1H, A                       ; 后写 CCAP0H
    JNB     Tx3_Ting, L_QuitPCA1            ; 不发送, 退出
    MOV     A, Tx3_BitCnt
```

```
    JNZ     L_TxData                ; 发送计数器为 0  表明单字节发送还没开始
    CLR     P_TX3                   ; 发送开始位
    MOV     Tx3_DAT, Tx3_BUF        ; 把缓冲的数据放到发送的 buff
    MOV     Tx3_BitCnt, #9          ; 发送数据位数  (8 数据位+1 停止位)
    SJMP    L_QuitPCA1

L_TxData:                           ; 发送计数器为非 0  正在发送数据
    DJNZ    Tx3_BitCnt, L_TxBit     ; 发送计数器减为 0  表明单字节发送结束
    SETB    P_TX3                   ; 送停止位数据
    CLR     Tx3_Ting                ; 发送停止
    SJMP    L_QuitPCA1

L_TxBit:
    MOV     A, Tx3_DAT              ; 把最低位送到  CY(益处标志位)
    RRC     A
    MOV     P_TX3, C                ; 发送一个 bit 数据
    MOV     Tx3_DAT, A

L_QuitPCA1:
    POP     ACC
    POP     PSW

    RETI

END
```

20 STC15 系列单片机 EEPROM 的应用

STC15 系列单片机内部集成了大容量的 EEPROM，其与程序空间是分开的。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。使用时，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。

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 Vcc 偏低时，建议不要进行 EEPROM/IAP 操作。

20.1 IAP 及 EEPROM 新增特殊功能寄存器介绍

符号	描述	地址	位地址及符号								复位值
			MSB							LSB	
IAP_DATA	ISP/IAP Flash Data Register	C2H									1111 1111B
IAP_ADDRH	ISP/IAP Flash Address High	C3H									0000 0000B
IAP_ADDRL	ISP/IAP Flash Address Low	C4H									0000 0000B
IAP_CMD	ISP/IAP Flash Command Register	C5H	-	-	-	-	-	-	MS1	MS0	xxxx x000B
IAP_TRIG	ISP/IAP Flash Command Trigger	C6H									xxxx xxxxB
IAP_CONTR	ISP/IAP_CONTRol Register	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	WT2	WT1	WT0	0000 x000B
PCON	Power Control	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011 0000B

1、ISP/IAP 数据寄存器 IAP_DATA

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IAP_DATA	C2H	name	-	-	-	-	-	-	-	-

IAP_DATA：ISP/IAP 操作时的数据寄存器。

ISP/IAP 从 Flash 读出的数据放在此处，向 Flash 写的数据也需放在此处。

2、ISP/IAP 地址寄存器 IAP_ADDRH 和 IAP_ADDRL

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IAP_ADDRH	C3H	name	-	-	-	-	-	-	-	-
IAP_ADDRL	C4H	name	-	-	-	-	-	-	-	-

IAP_ADDRH：ISP/IAP 操作时的地址寄存器高八位。

IAP_ADDRL：ISP/IAP 操作时的地址寄存器低八位。

3、ISP/IAP 命令寄存器 IAP_CMD

ISP/IAP 命令寄存器 IAP_CMD 格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CMD	C5H	name	-	-	-	-	-	-	MS1	MS0

MS1	MS0	命令 / 操作 模式选择
0	0	Standby 待机模式，无 ISP 操作
0	1	从用户的应用程序区对"Data Flash/EEPROM 区"进行字节读
1	0	从用户的应用程序区对"Data Flash/EEPROM 区"进行字节编程
1	1	从用户的应用程序区对"Data Flash/EEPROM 区"进行扇区擦除

程序在用户应用程序区时，仅可以对数据 Flash 区（EEPROM）进行字节读/字节编程/扇区擦除

IAP15 系列除外，IAP15 系列可在用户应用程序区修改用户应用程序区。

【特别声明】: EEPROM 也可以用 MOVC 指令读（MOVC 访问的是程序存储器），但起始地址不再是 0000H，而是程序存储空间结束地址的下一个地址。

4、ISP/IAP 命令触发寄存器 IAP_TRIG

ISP/IAP 命令触发寄存器 IAP_TRIG 格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IAP_TRIG	C6H	name	–							

IAP_TRIG：ISP/IAP 操作时的命令触发寄存器。

在 IAPEN（IAP_CONTR.7）=1 时，对 IAP_TRIG 先写入 5Ah，再写入 A5h，ISP/IAP 命令才会生效。

ISP/IAP 操作完成后，IAP 地址高八位寄存器 IAP_ADDRH、IAP 地址低八位寄存器 IAP_ADDRL 和 IAP 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 ISP/IAP 操作，需手动将该地址的高 8 位和低 8 位分别写入 IAP_ADDRH 和 IAP_ADDRL 寄存器。

每次 IAP 操作时，都要对 IAP_TRIG 先写入 5AH，再写入 A5H，ISP/IAP 命令才会生效。

在每次触发前，需重新送字节读/字节编程/扇区擦除命令，在命令不改变时，不需重新送命令。

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】
```

5、ISP/IAP 命令寄存器 IAP_CONTR

ISP/IAP 控制寄存器 IAP_CONTR 格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CONTR	C7H	name	IAPEN	SWBS	SWRST	CMD_FAIL	-	WT2	WT1	WT0

IAPEN：ISP/IAP 功能允许位。

- 禁止 IAP 读/写/擦除 DataFlash/EEPROM
- 允许 LAP 读/写/擦除 Data Flash/EEPROM

SWBS： 软件选择复位后从用户应用程序区启动（送 0），还是从系统 ISP 监控程序区启动（送 1）。要与 SWRST 直接配合才可以实现。

SWRST：0：不操作；1：软件控制产生复位，单片机自动复位。

CMDFAIL：如果 IAP 地址（由 IAP 地址寄存器 IAP_ADDRH 和 IAP_ADDRL 的值决定）指向了非法地址或无效地址，且送了 ISP/IAP 命令，并对 IAP_TRIG 送 5AH/A5h 触发失败，则 CMD_FAIL 为 1，需由软件清零。

```
;在用户应用程序区（AP 区）软件复位并从用户应用程序区（AP 区）开始执行程序
MOV     IAP_CONTR, #00100000B       ;SWBS=0（选择 AP 区），SWRST=1（软复位）

;在用户应用程序区（AP 区）软件复位并从系统 ISP 监控程序区开始执行程序
MOV     IAP_CONTR, #01100000B       ;SWBS=1（选择 ISP 区），SWRST =1（软复位）

;在系统 ISP 监控程序区软件复位并从用户应用程序区（AP 区）开始执行程序
MOV     IAP_CONTR, #00100000B       ;SWBS=0（选择 AP 区），SWRST=1（软复位）

;在系统 ISP 监控程序区软件复位并从系统 ISP 监控程序区开始执行程序
MOV     IAP_CONTR, #01100000B       ;SWBS=1（选择 ISP 区），SWRST =1（软复位）
```

设置等待时间			CPU 等待时间（多少个 CPU 工作时钟 ）			
WT2	WT1	WT0	Read/读 （2 个时钟）	Program/编程 （=55us）	Sector Erase 扇区擦除（=21ms）	Recommended System Clock 跟等待参数对应的推荐系统时钟
1	1	1	2 个时钟	55 个时钟	21012 个时钟	≥ 1MHz
1	1	0	2 个时钟	110 个时钟	42024 个时钟	≥ 2MHz
1	0	1	2 个时钟	165 个时钟	63036 个时钟	≥ 3MHz
1	0	0	2 个时钟	330 个时钟	126072 个时钟	≥ 6MHz
0	1	1	2 个时钟	660 个时钟	252144 个时钟	≥ 12MHz
0	1	0	2 个时钟	1100 个时钟	420240 个时钟	≥ 20MHz
0	0	1	2 个时钟	1320 个时钟	504288 个时钟	≥ 24MHz
0	0	0	2 个时钟	1760 个时钟	672384 个时钟	≥ 30MHz

6、工作电压过低判断，此时不要进行 EEPROM/IAP 操作

PCON：电源控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
PCON	87H	name	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL

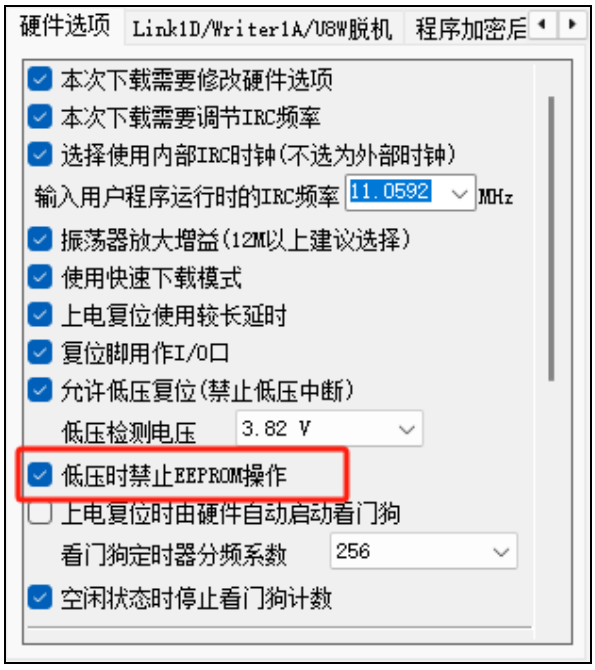
LVDF： 低压检测标志位，当工作电压 Vcc 低于低压检测门槛电压时，该位置 1。该位要由软件清 0。当低压检测电路发现工作电压 Vcc 偏低时，不要进行 EEPROM/IAP 操作。

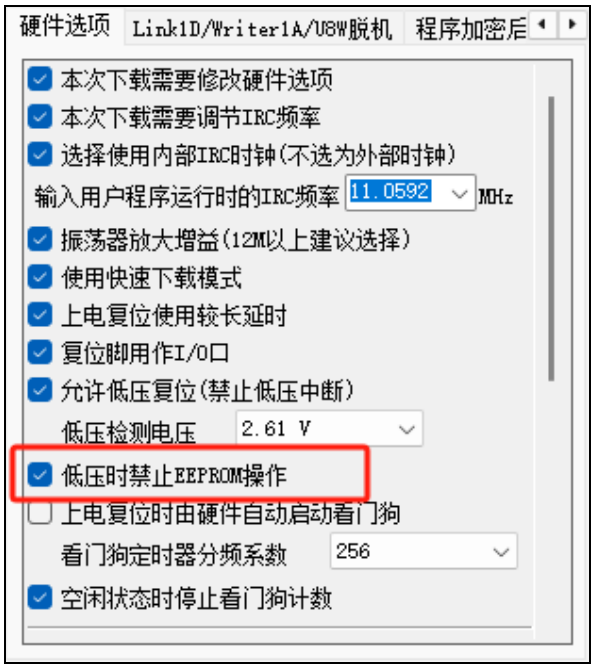
5V 单片机的低压检测门槛电压：

-40 ℃	25 ℃	85 ℃
4.74	4.64	4.60
4.41	4.32	4.27
4.14	4.05	4.00
3.90	3.82	3.77
3.69	3.61	3.56
3.51	3.43	3.38
3.36	3.28	3.23
3.21	3.14	3.09

3.3V 单片机的低压检测门槛电压：

-40 ℃	25 ℃	85 ℃
3.11	3.08	3.09
2.85	2.82	2.83
2.63	2.61	2.61
2.44	2.42	2.43
2.29	2.26	2.26
2.14	2.12	2.12
2.01	2.00	2.00
1.90	1.89	1.89





建议在电压偏低时，不要操作 EEPROM/IAP，烧录时直接选择“低压禁止 EEPROM 操作”

20.2 STC15 系列单片机 EEPROM 空间大小及地址

20.2.1 STC15W4K32S4 系列单片机 EEPROM 空间大小及地址

STC15W4K32S4 系列单片机内部 EEPROM 选型一览表							STC15W4K32S4 系列单片机内部 EEPROM 还可以用 MOVC 指令读，但此时首地址不再是 0000H，而是程序存储空间结束地址的下一个地址。 512 字节为一个扇区。 建议同一次修改的数据放在同一扇区，不是同一次修改的数据放在不同的扇区。
型号	EEPROM 字节数	扇区数	用 IAP 字节读时 EEPROM 起始扇区首地址	用 IAP 字节读时 EEPROM 结束扇区末尾地址	用 MOVC 指令读时 EEPROM 起始扇区首地址	用 MOVC 指令读时 EEPROM 结束扇区末尾地址	
STC15W4K16S4	42K	84	0000h	A7FFh	4000h	E7FFh	
STC15W4K32S4	26K	52	0000h	67FFh	8000h	E7FFh	
STC15W4K40S4	18K	36	0000h	47FFh	A000h	E7FFh	
STC15W4K48S4	10K	20	0000h	27FFh	C000h	E7FFh	
STC15W4K56S4	2K	4	0000h	07FFh	E000h	E7FFh	
以下系列特殊，用户可在用户程序区直接修改用户程序，所有 Flash 空间均可作 EEPROM 修改							
IAP15W4K58S4	-	116	0000h	E7FFh			没有专门的 EEPROM，但用户可将用户程序区的程序 FLASH 当 EEPROM 使用，使用时不要将自己的有效程序擦除。
IAP15W4K61S4	-	122	0000h	F3FFh			没有专门的 EEPROM，但用户可将用户程序区的程序 FLASH 当 EEPROM 使用，使用时不要将自己的有效程序擦除。
IRC15W4K63S4	-	127	0000h	FDFFh			没有专门的 EEPROM，但用户可将用户程序区的程序 FLASH 当 EEPROM 使用，使用时不要将自己的有效程序擦除。

20.2.2 STC15F2K60S2 及 STC15L2K60S2 系列 EEPROM 空间大小及地址

STC15F2K60S2 系列单片机内部 EEPROM 选型一览表 STC15L2K60S2 系列单片机内部 EEPROM 选型一览表							
型号	EEPROM 字节数	扇区数	用 IAP 字节读时 EEPROM 起始扇区首地址	用 IAP 字节读时 EEPROM 结束扇区末尾地址	用 MOVC 指令读时 EEPROM 起始扇区首地址	用 MOVC 指令读时 EEPROM 结束扇区末尾地址	
STC15F2K08S2 STC15L2K08S2	53K	106	0000h	D3FFh	2000h	F3FFh	STC15F2K60S2 及 STC15L2K60S2 系列单片机内部 EEPROM 还可以用 MOVC 指令读，但此时首地址不再是 0000H，而是程序存储空间结束地址的下一个地址。 512 字节为一个扇区 建议同一次修改的数据放在同一扇区，不是同一次修改的数据放在不同的扇区
STC15F2K16S2 STC15L2K16S2	45K	90	0000h	B3FFh	4000h	F3FFh	
STC15F2K24S2 STC15L2K24S2	37K	74	0000h	93FFh	6000h	F3FFh	
STC15F2K32S2 STC15L2K32S2	29K	58	0000h	73FFh	8000h	F3FFh	
STC15F2K40S2 STC15L2K40S2	21K	42	0000h	53FFh	A000h	F3FFh	
STC15F2K48S2 STC15L2K48S2	13K	26	0000h	33FFh	C000h	F3FFh	
STC15F2K56S2 STC15L2K56S2	5K	10	0000h	13FFh	E000h	F3FFh	
STC15F2K60S2 STC15L2K60S2	1K	2	0000h	03FFh	F000h	F3FFh	
STC15F2K32S STC15L2K32S	29K	58	0000h	73FFh	8000h	F3FFh	
STC15F2K60S STC15L2K60S	1K	2	0000h	03FFh	F000h	F3FFh	
STC15F2K24AS STC15L2K24AS	37K	74	0000h	93FFh	6000h	F3FFh	
STC15F2K48AS STC15L2K48AS	13K	26	0000h	33FFh	C000h	F3FFh	
以下系列特殊，用户可在用户程序区直接修改用户程序，所有 Flash 空间均可作 EEPROM 修改							
IAP15F2K61S2 IAP15L2K61S2	-	122	0000h	F3FFh			没有专门的 EEPROM，但用户可将用户程序区的程序 FLASH 当 EEPROM 使用，使用时不要将自己的有效程序擦除。
IRC15F2K63S2	-	126	0000h	FBFFh			没有专门的 EEPROM，但用户可将用户程序区的程序 FLASH 当 EEPROM 使用，使用时不要将自己的有效程序擦除。
IAP15F2K61S IAP15L2K61S	-	122	0000h	F3FFh			没有专门的 EEPROM，但用户可将用户程序区的程序 FLASH 当 EEPROM 使用，使用时不要将自己的有效程序擦除。

20.2.3 STC15W1K08PWM 系列单片机 EEPROM 空间大小及地址

STC15W1K08PWM 系列单片机内部 EEPROM 选型一览表							STC15W1K08PWM 系列单片机内部 EEPROM 不可以用 MOVC 指令读。 512 字节为一个扇区 建议同一次修改的数据放在同一扇区，不是同一次修改的数据放在不同的扇区。
型号	EEPROM 字节数	扇区数	用 IAP 字节读时 EEPROM 起始扇区首地址	用 IAP 字节读时 EEPROM 结束扇区末尾地址	用 MOVC 指令读时 EEPROM 起始扇区首地址	用 MOVC 指令读时 EEPROM 结束扇区末尾地址	
STC15W1K08PWM	19K	38	0000h	4BFFh			
STC15W1K16PWM	11K	22	0000h	2BFFh			

20.2.4 STC15W1K16S 系列单片机 EEPROM 空间大小及地址

STC15W1K16S 系列单片机内部 EEPROM 选型一览表							STC15W1K16S 系列单片机内部 EEPROM 不可以用 MOVC 指令读。 512 字节为一个扇区 建议同一次修改的数据放在同一扇区，不是同一次修改的数据放在不同的扇区
型号	EEPROM 字节数	扇区数	用 IAP 字节读时 EEPROM 起始扇区首地址	用 IAP 字节读时 EEPROM 结束扇区末尾地址	用 MOVC 指令读时 EEPROM 起始扇区首地址	用 MOVC 指令读时 EEPROM 结束扇区末尾地址	
STC15W1K16S	13K	26	0000h	33FFh			
STC15W1K24S	5K	10	0000h	13FFh			
以下系列特殊，用户可在用户程序区直接修改用户程序，所有 Flash 空间均可作 EEPROM 修改							
IAP15W1K29S	-	58	0000h	73FFh			没有专门的 EEPROM，但用户可将用户程序区的程序 FLASH 当 EEPROM 使用，使用时不要将自己的有效程序擦除。
IRC15W1K31S	-	63	0000h	7DFFh			没有专门的 EEPROM，但用户可将用户程序区的程序 FLASH 当 EEPROM 使用，使用时不要将自己的有效程序擦除。

STC15W1K16S 系列中以 STC 开头的单片机不能用 MOVC 读 EEPROM。

20.2.5 STC15W404S 系列单片机 EEPROM 空间大小及地址

STC15W404S 系列单片机内部 EEPROM 选型一览表							STC15W404S 系列单片机内部 EEPROM 不可以用 MOVC 指令读 512 字节为一个扇区 建议同一次修改的数据放在同一扇区，不是同一次修改的数据放在不同的扇区
型号	EEPROM 字节数	扇区数	用 IAP 字节读时 EEPROM 起始扇区首地址	用 IAP 字节读时 EEPROM 结束扇区末尾地址	用 MOVC 指令读时 EEPROM 起始扇区首地址	用 MOVC 指令读时 EEPROM 结束扇区末尾地址	
STC15W404S	9K	18	0000h	23FFh			
STC15W408S	5K	10	0000h	13FFh			
STC15W410S	3K	6	0000h	0BFFh			
以下系列特殊，用户可在用户程序区直接修改用户程序，所有 Flash 空间均可作 EEPROM 修改							
IAP15W413S	-	26	0000h	33FFh			没有专门的 EEPROM，但用户可将用户程序区的程序 FLASH 当 EEPROM 使用，使用时不要将自己的有效程序擦除。
IRC15W415S	-	31	0000h	3DFFh			没有专门的 EEPROM，但用户可将用户程序区的程序 FLASH 当 EEPROM 使用，使用时不要将自己的有效程序擦除。

STC15W404S 系列中以 STC 开头的单片机不能用 MOVC 读 EEPROM。

20.2.6 STC15W408AS 系列单片机 EEPROM 空间大小及地址

STC15W408AS 系列单片机内部 EEPROM 选型一览表							STC15W408AS 系列单片机内部 EEPROM 还可以用 MOVC 指令读，但此时首地址不再是 0000H，而是程序存储空间结束地址的下一个地址。 512 字节为一个扇区 建议同一次修改的数据放在同一扇区，不是同一次修改的数据放在不同的扇区
型号	EEPROM 字节数	扇区数	用 IAP 字节读时 EEPROM 起始扇区首地址	用 IAP 字节读时 EEPROM 结束扇区末尾地址	用 MOVC 指令读时 EEPROM 起始扇区首地址	用 MOVC 指令读时 EEPROM 结束扇区末尾地址	
STC15W401AS	5K	10	0000h	13FFh	400h	17FFh	
STC15W402AS	5K	10	0000h	13FFh	800h	1BFFh	
STC15W404AS	9K	18	0000h	23FFh	1000h	33FFh	
STC15W408AS	5K	10	0000h	13FFh	2000h	33FFh	
STC15W410AS	3K	6	0000h	0BFFh	2800h	33FFh	
STC15W412AS	1K	2	0000h	03FFh	3000h	33FFh	
以下系列特殊，用户可在用户程序区直接修改用户程序，所有 Flash 空间均可作 EEPROM 修改							
IAP15W413AS	-	26	0000h	33FFh			没有专门的 EEPROM，但用户可将用户程序区的程序 FLASH 当 EEPROM 使用，使用时不要将自己的有效程序擦除。
IRC15W415AS	-	31	0000h	3DFFh			没有专门的 EEPROM，但用户可将用户程序区的程序 FLASH 当 EEPROM 使用，使用时不要将自己的有效程序擦除。

20.2.7 STC15W204S 系列单片机 EEPROM 空间大小及地址

STC15W204S 系列单片机内部 EEPROM 选型一览表							STC15W204S 系列单片机内部 EEPROM 不可以用 MOVC 指令读 512 字节为一个扇区 建议同一次修改的数据放在同一扇区，不是同一次修改的数据放在不同的扇区
型号	EEPROM 字节数	扇区数	用 IAP 字节读时 EEPROM 起始扇区首地址	用 IAP 字节读时 EEPROM 结束扇区末尾地址	用 MOVC 指令读时 EEPROM 起始扇区首地址	用 MOVC 指令读时 EEPROM 结束扇区末尾地址	
STC15W201S	4K	8	0000h	0FFFh			
STC15W202S	3K	6	0000h	0BFFh			
STC15W203S	2K	4	0000h	07FFh			
STC15W204S	1K	2	0000h	03FFh			
以下系列特殊，用户可在用户程序区直接修改用户程序，所有 Flash 空间均可作 EEPROM 修改							
IAP15W205S	-	10	0000h	13FFh			没有专门的 EEPROM，但用户可将用户程序区的程序 FLASH 当 EEPROM 使用，使用时不要将自己的有效程序擦除。
IRC15W207S	-	15	0000h	1DFFh			没有专门的 EEPROM，但用户可将用户程序区的程序 FLASH 当 EEPROM 使用，使用时不要将自己的有效程序擦除。

STC15W204S 系列中以 STC 开头的单片机不能用 MOVC 读 EEPROM。

20.2.8 STC15W104 系列 EEPROM 空间大小及地址

STC15W104 系列单片机内部 EEPROM 选型一览表							STC15W104 系列单片机内部 EEPROM 还可以用 MOVC 指令读，但此时首地址不再是 0000H，而是程序存储空间结束地址的下一个地址。 512 字节为一个扇区 建议同一次修改的数据放在同一扇区，不是同一次修改的数据放在不同的扇区
型号	EEPROM 字节数	扇区数	用 IAP 字节读时 EEPROM 起始扇区首地址	用 IAP 字节读时 EEPROM 结束扇区末尾地址	用 MOVC 指令读时 EEPROM 起始扇区首地址	用 MOVC 指令读时 EEPROM 结束扇区末尾地址	
STC15W101	4K	8	0000h	0FFFh	0400h	13FFh	
STC15W102	3K	6	0000h	0BFFh	0800h	13FFh	
STC15W103	2K	4	0000h	07FFh	0C00h	13FFh	
STC15W104	1K	2	0000h	03FFh	1000h	13FFh	
以下系列特殊，用户可在用户程序区直接修改用户程序，所有 Flash 空间均可作 EEPROM 修改							
IAP15W105	-	10	0000h	13FFh			没有专门的 EEPROM，但用户可将用户程序区的程序 FLASH 当 EEPROM 使用，使用时不要将自己的有效程序擦除。
IRC15W107	-	14	0000h	1BFFh			没有专门的 EEPROM，但用户可将用户程序区的程序 FLASH 当 EEPROM 使用，使用时不要将自己的有效程序擦除。

20.2.9 STC15F101W 及 STC15L104W 系列 EEPROM 空间大小及地址

STC15F101W 系列单片机内部 EEPROM 选型一览表 STC15L101W 系列单片机内部 EEPROM 选型一览表							STC15F101W 及 STC15L101W 系列单片机内部 EEPROM 还可以用 MOVC 指令读，但此时首地址不再是 0000H，而是程序存储空间结束地址的下一个地址。 512 字节为一个扇区 建议同一次修改的数据放在同一扇区，不是同一次修改的数据放在不同的扇区
型号	EEPROM 字节数	扇区数	用 IAP 字节读时 EEPROM 起始扇区首地址	用 IAP 字节读时 EEPROM 结束扇区末尾地址	用 MOVC 指令读时 EEPROM 起始扇区首地址	用 MOVC 指令读时 EEPROM 结束扇区末尾地址	
STC15F101W STC15L101W	4K	8	0000h	0FFFh	0400h	13FFh	
STC15F102W STC15L102W	3K	6	0000h	0BFFh	0800h	13FFh	
STC15F103W STC15L103W	2K	4	0000h	07FFh	0C00h	13FFh	
STC15F104W STC15L104W	1K	2	0000h	03FFh	1000h	13FFh	
以下系列特殊，用户可在用户程序区直接修改用户程序，所有 Flash 空间均可作 EEPROM 修改							
IAP15F105W IAP15L105W	-	10	0000h	13FFh			没有专门的 EEPROM，但用户可将用户程序区的程序 FLASH 当 EEPROM 使用，使用时不要将自己的有效程序擦除。
IRC15F107W	-	14	0000h	1BFFh			没有专门的 EEPROM，但用户可将用户程序区的程序 FLASH 当 EEPROM 使用，使用时不要将自己的有效程序擦除。

20.2.10 STC15F408AD 及 STC15L408AD 系列 EEPROM 空间大小及地址

STC15F408AD 系列单片机内部 EEPROM 选型一览表 STC15L408AD 系列单片机内部 EEPROM 选型一览表							STC15F408AD 及 STC15L408AD 系列单片机内部 EEPROM 还可以用 MOVC 指令读，但此时首地址不再是 0000H，而是程序存储空间结束地址的下一个地址。 512 字节为一个扇区 建议同一次修改的数据放在同一扇区，不是同一次修改的数据放在不同的扇区
型号	EEPROM 字节数	扇区数	用 IAP 字节读时 EEPROM 起始扇区首地址	用 IAP 字节读时 EEPROM 结束扇区末尾地址	用 MOVC 指令读时 EEPROM 起始扇区首地址	用 MOVC 指令读时 EEPROM 结束扇区末尾地址	
STC15F408AD STC15L408AD	5K	10	0000h	13FFh	2000h	33FFh	
以下系列特殊，用户可在用户程序区直接修改用户程序，所有 Flash 空间均可作 EEPROM 修改							
IAP15F413AD IAP15L413AD	-	26	0000h	33FFh			没有专门的 EEPROM，但用户可将用户程序区的程序 FLASH 当 EEPROM 使用，使用时不要将自己的有效程序擦除。

STC15 单片机的内部 EEPROM 地址表												
扇区	起始地址	结束地址	扇区	起始地址	结束地址	扇区	起始地址	结束地址	扇区	起始地址	结束地址	
第 1 扇区	0000h	1FFh	第 33 扇区	4000h	41FFh	第 65 扇区	8000h	81FFh	第 97 扇区	C000h	C1FFh	每个扇区 512 字节 建议同一次修改的数据放在同一扇区，不是同一次修改的数据放在不同的扇区，不必用满，当然可全用
第 2 扇区	200h	3FFh	第 34 扇区	4200h	43FFh	第 66 扇区	8200h	83FFh	第 98 扇区	C200h	C3FFh	
第 3 扇区	400h	5FFh	第 35 扇区	4400h	45FFh	第 67 扇区	8400h	85FFh	第 99 扇区	C400h	C5FFh	
第 4 扇区	600h	7FFh	第 36 扇区	4600h	47FFH	第 68 扇区	8600h	87FFh	第 100 扇区	C600h	C7FFh	
第 5 扇区	800h	9FFh	第 37 扇区	4800h	49FFh	第 69 扇区	8800h	89FFh	第 101 扇区	C800h	C9FFh	
第 6 扇区	A00h	BFFh	第 38 扇区	4A00h	4BFFh	第 70 扇区	8A00h	8BFFh	第 102 扇区	CA00h	CBFFh	
第 7 扇区	C00h	DFFh	第 39 扇区	4C00h	4DFFh	第 71 扇区	8C00h	8DFFh	第 103 扇区	CC00h	CDFFh	
第 8 扇区	E00h	FFFh	第 40 扇区	4E00h	4FFFH	第 72 扇区	8E00h	8FFFh	第 104 扇区	CE00h	CFFFH	
第 9 扇区	1000h	11FFh	第 41 扇区	5000h	51FFh	第 73 扇区	9000h	91FFh	第 105 扇区	D000h	D1FFh	
第 10 扇区	1200h	13FFh	第 42 扇区	5200h	53FFh	第 74 扇区	9200h	93FFh	第 106 扇区	D200h	D3FFh	
第 11 扇区	1400h	15FFh	第 43 扇区	5400h	55FFh	第 75 扇区	9400h	95FFh	第 107 扇区	D400h	D5FFh	
第 12 扇区	1600h	17FFh	第 44 扇区	5600h	57FFH	第 76 扇区	9600h	97FFh	第 108 扇区	D600h	D7FFh	
第 13 扇区	1800h	19FFh	第 45 扇区	5800h	59FFh	第 77 扇区	9800h	99FFh	第 109 扇区	D800h	D9FFh	
第 14 扇区	1A00h	1BFFh	第 46 扇区	5A00h	5BFFh	第 78 扇区	9A00h	9BFFh	第 110 扇区	DA00h	DBFFh	
第 15 扇区	1C00h	1DFFh	第 47 扇区	5C00h	5DFFh	第 79 扇区	9C00h	9DFFh	第 111 扇区	DC00h	DDFFh	
第 16 扇区	1E00h	1FFFh	第 48 扇区	5E00h	5FFFH	第 80 扇区	9E00h	9FFFH	第 112 扇区	DE00h	DFFFH	
第 17 扇区	2000h	21FFh	第 49 扇区	6000h	61FFh	第 81 扇区	A000h	A1FFh	第 113 扇区	E000h	E1FFh	
第 18 扇区	2200h	23FFh	第 50 扇区	6200h	63FFh	第 82 扇区	A200h	A3FFh	第 114 扇区	E200h	E3FFh	
第 19 扇区	2400h	25FFh	第 51 扇区	6400h	65FFh	第 83 扇区	A400h	A5FFh	第 115 扇区	E400h	E5FFh	
第 20 扇区	2600h	27FFh	第 52 扇区	6600h	67FFH	第 84 扇区	A600h	A7FFh	第 116 扇区	E600h	E7FFh	
第 21 扇区	2800h	29FFh	第 53 扇区	6800h	69FFh	第 85 扇区	A800h	A9FFh	第 117 扇区	E800h	E9FFh	
第 22 扇区	2A00h	2BFFh	第 54 扇区	6A00h	6BFFh	第 86 扇区	AA00h	ABFFh	第 118 扇区	EA00h	EBFFh	
第 23 扇区	2C00h	2DFFh	第 55 扇区	6C00h	6DFFh	第 87 扇区	AC00h	ADFFh	第 119 扇区	EC00h	EDFFh	
第 24 扇区	2E00h	2FFFH	第 56 扇区	6E00h	6FFFH	第 88 扇区	AE00h	AFFFH	第 120 扇区	EE00h	EFFFH	
第 25 扇区	3000h	31FFh	第 57 扇区	7000h	71FFh	第 89 扇区	B000h	B1FFh	第 121 扇区	F000h	F1FFh	
第 26 扇区	3200h	33FFh	第 58 扇区	7200h	73FFh	第 90 扇区	B200h	B3FFh	第 122 扇区	F200h	F3FFh	
第 27 扇区	3400h	35FFh	第 59 扇区	7400h	75FFh	第 91 扇区	B400h	B5FFh	第 123 扇区	F400h	F5FFh	
第 28 扇区	3600h	37FFH	第 60 扇区	7600h	77FFh	第 92 扇区	B600h	B7FFh	第 124 扇区	F600h	F7FFh	
第 29 扇区	3800h	39FFh	第 61 扇区	7800h	79FFh	第 93 扇区	B800h	B9FFh	第 125 扇区	F800h	F9FFh	
第 30 扇区	3A00h	3BFFh	第 62 扇区	7A00h	7BFFh	第 94 扇区	BA00h	BBFFh	第 126 扇区	FA00h	FBFFh	
第 31 扇区	3C00h	3DFFh	第 63 扇区	7C00h	7DFFh	第 95 扇区	BC00h	BDFFh	第 127 扇区	FC00h	FDFFh	
第 32 扇区	3E00h	3FFFH	第 64 扇区	7E00h	7FFFh	第 96 扇区	BE00h	BFFFH				

20.3 IAP 及 EEPROM 汇编简介

```
;用 DATA 还是 EQU 声明新增特殊功能寄存器地址要看你用的汇编器/编译器
IAP_DATA    DATA  0C2h;  或  IAP_DATA    EQU  0C2h
IAP_ADDRH   DATA  0C3h;  或  IAP_ADDRH   EQU  0C3h
IAP_ADDRL   DATA  0C4h;  或  IAP_ADDRL   EQU  0C4h
IAP_CMD     DATA  0C5h;  或  IAP_CMD     EQU  0C5h
IAP_TRIG    DATA  0C6h;  或  IAP_TRIG    EQU  0C6h
IAP_CONTR   DATA  0C7h;  或  IAP_CONTR   EQU  0C7h

;定义 ISP/IAP 命令及等待时间
ISP_IAP_BYTE_READ           EQU  1   ;字节读
ISP_IAP_BYTE_PROGRAM        EQU  2   ;字节编程，前提是该字节是空，0FFh
ISP_IAP_SECTOR_ERASE        EQU  3   ;扇区擦除，要某字节为空，要擦一扇区
WAIT_TIME                   EQU  0   ;设置等待时间，30MHz 以下 0，24M 以下 1，
                                     ;20MHz 以下 2，12M 以下 3，6M 以下 4，
                                     ;3M 以下 5，2M 以下 6，1M 以下 7.

;字节读，也可以用 MOVC 指令读，但起始地址不再是 0000H，而是程序存储空间结束地址的下一个地址
MOV     IAP_ADDRH, #BYTE_ADDR_HIGH   ;送地址高字节
MOV     IAP_ADDRL, #BYTE_ADDR_LOW    ;送地址低字节
                                     ;地址需要改变时，才需重新送地址

MOV     IAP_CONTR, #WAIT_TIME        ;设置等待时间
ORL     IAP_CONTR, #10000000B        ;允许 ISP/IAP 操作
                                     ;此两句可以合成一句并且只送一次就够了

MOV     IAP_CMD, #ISP_IAP_BYTE_READ  ;送字节读命令，现有 A 版本每次触发前需重新送命令。
                                     ;在命令不需改变时，不需重新送命令

MOV     IAP_TRIG, #5Ah               ;先送 5Ah，再送 A5h 到 ISP/IAP 触发寄存器,每次都需如此
MOV     IAP_TRIG, #0A5h              ;送完 A5h 后，ISP/IAP 命令立即被触发起动

;CPU 等待 IAP 动作完成后，才会继续执行程序。
NOP                                  ;数据读出到 IAP_DATA 寄存器后，CPU 继续执行程序
MOV     A, ISP_DATA                  ;将读出的数据送往 Acc

;以下语句可不用，只是出于安全考虑而已
MOV     IAP_CONTR, #00000000B        ;禁止 ISP/IAP 操作
MOV     IAP_CMD, #00000000B          ;去除 ISP/IAP 命令
;MOV    IAP TRIG, #00000000B         ;防止 ISP/IAP 命令误触发
;MOV    IAP_ADDRH, #0FFh             ;送地址高字节单元为 FF，指向非 EEPROM 区
;MOV    IAP_ADDRL, #0FFh             ;送地址低字节单元为 FF，防止误操作
```

```
;字节编程，该字节为 FFh/空时，可对其编程，否则不行，要先执行扇区擦除
MOV     IAP_DATA, #ONE_DATA                 ;送字节编程数据到 IAP_DATA
                                            ;只有数据改变时才需重新送

MOV     IAP_ADDRH, #BYTE_ADDR_HIGH          ;送地址高字节
MOV     IAP_ADDRL, #BYTE_ADDR_LOW           ;送地址低字节
                                            ;地址需要改变时，才需重新送地址

MOV     IAP_CONTR, #WAIT_TIME               ;设置等待时间
ORL     IAP_CONTR, #10000000B               ;允许 ISP/IAP 操作
                                            ;此两句可合成一句，并且只一送一次就够了
MOV     IAP_CMD, #ISP_IAP_BYTE_PROGRAM      ;送字节编程命令,现有 A 版本每次触发前需重新送命令。
                                            ;在命令不需改变时，不需重新送命令
MOV     IAP_TRIG, #5Ah                      ;先送 5Ah，再送 A5h 到 ISP/IAP 触发寄存器，每次都需如此
MOV     IAP_TRIG, #0A5h                     ;送完 A5h 后，ISP/IAP 命令立即被触发起动

;CPU 等待 IAP 动作完成后，才会继续执行程序。
NOP                                         ;字节编程成功后，CPU 继续执行程序

;以下语句可不用,只是出于安全考虑而已
MOV     IAP_CONTR, #00000000B               ;禁止 ISP/IAP 操作
MOV     IAP_CMD, #00000000B                 ;去除 ISP/IAP 命令
;MOV    IAP_TRIG, #00000000B            ;防止 ISP/IAP 命令误触发
;MOV    IAP_ADDRH, #0FFh                    ;送地址高字节单元为 FF
                                            ;指向非 EEPROM 区，防止误操作
;MOV    IAP_ADDRL, #0FFh                    ;送地址低字节单元为 FF
                                            ;指向非 EEPROM 区，防止误操作

;扇区擦除，没有字节擦除，只有扇区擦除，512 字节/扇区，每个扇区用得越少越方便
;如果要对某个扇区进行擦除，而其中有些字节的内容需要保留，则需将其先读到单片机
;内部的 RAM 中保存，再将该扇区擦除，然后将须保留的数据写回该扇区，所以每个扇区中用的字节数
;越少越好，操作起来越灵活方便。扇区中任意一个字节的地址都是该扇区的地址，无需求出首地址
MOV     IAP_ADDRH, #SECTOR_FIRST_BYTE_ADDR_HIGH     ;送扇区起始地址高字节
MOV     IAP_ADDRL, #SECTOR_FIRST_BYTE_ADDR_LOW      ;送扇区起始地址低字节
                                                    ;地址需要改变时才需重新送地址
MOV     IAP_CONTR, #WAIT_TIME                       ;设置等待时间
ORL     IAP_CONTR, #10000000B                       ;允许 ISP/IAP
                                            ;此两句可以合成一句，并且只送一次就够了
MOV     IAP_CMD, #ISP_IAP_SECTOR_ERASE      ;送扇区擦除命令,现有 A 版本每次触发前需重新送命令。
                                            ;在命令不需改变时,不需重新送命令
MOV     IAP_TRIG, #5Ah                      ;先送 5Ah,再送 A5h 到 ISP/IAP 触发寄存器,每次都需如此
MOV     IAP_TRIG, #0A5h                     ;送完 A5h 后，ISP/IAP 命令立即被触发起动

;CPU 等待 IAP 动作完成后，才会继续执行程序
NOP                                         ;扇区擦除成功后，CPU 继续执行程序
```

```
;以下语句可不用,只是出于安全考虑而已
MOV     IAP_CONTR, #00000000B           ;禁止 ISP/IAP 操作
MOV     IAP_CMD, #00000000B             ;去除 ISP/IAP 命令
;MOV    IAP_TRIG, #00000000B          ;防止 ISP/IAP 命令误触发
;MOV    IAP_ADDRH, #0FFh              ;送地址高字节单元为 FF，指向非 EEPROM 区
;MOV    IAP_ADDRL, #0FFh              ;送地址低字节单元为 FF，防止误操作
```

小常识：(STC 单片机的 DataFlash 当 EEPROM 功能使用)

3 个基本命令----字节读，字节编程，扇区擦除

字节编程：将“1”写成“1”或“0”，将“0”写成“0”。如果某字节是 FFH，才可对其进行字节编程。如果该字节不是 FFH，则须先将整个扇区擦除，因为只有“扇区擦除”才可以将“0”变为“1”。

扇区擦除：只有“扇区擦除”才可能将“0”擦除为“1”。

大建议:

1.同一次修改的数据放在同一扇区中，不是同一次修改的数据放在不同的扇区，就不需读出保护。

2.如果一个扇区只用一个字节，那就是真正的 EEPROM,STC 单片机的 Data Flash 比外部 EEPROM 要快很多，读一个字节/编程一个字节大概是 2 个时钟/55us。

3.如果在一个扇区中存放了大量的数据，某次只需要修改其中的一个字节或一部分字节时，则另外的不需要修改的数据须先读出放在 STC 单片机的 RAM 中，然后擦除整个扇区，再将需要保留的数据和需修改的数据按字节逐字节写回该扇区中(只有字节写命令，无连续字节写命令)。这时每个扇区使用的字节数是使用的越少越方便(不需读出一大堆需保留数据)。

4.以部分字节为一组数据时，可以在该组数据起始时增加一个特殊标志字节，该特殊标志字节用于标志该组数据是否被使用，即该组数据中的各字节是否被写入内容。其中，标志字节可以用 00H 表示该组数据已被使用，用 FFH 表示该组数据未被使用，这样用户就只需读取起始标志字节的内容就可以判断接下来所访问的一组数据是否被使用。当用户读取到某一组数据的起始标志字节为 FFH，就可以往其中写入内容，并且在使用完该组数据后，须将该组数据的起始标志字节修改为 00H，再向下访问。

例如，以 4 个字节一组，并在每 4 个字节数据前都增加一个起始标志字节，现有如下数据：00, xx, xx, xx, xx, ff, xx, xx, xx, xx, 这部分数据表示 00 后的 4 个字节数据已经被使用，而 ff 后的 4 个字节数据未被使用，用户可以往 ff 后的 4 个字节中写入内容，写完后再将标志 ff 改为 00 再往下访问。

常问的问题:

1:IAP 指令完成后，地址是否会自动“加 1”或“减 1”？
答：不会

2:送 5A 和 A5 触发后，下一次 IAP 命令是否还需要送 5A 和 A5 触发？
答：是，一定要。

20.4 EEPROM 测试程序(C 和汇编)

20.4.1 EEPROM 测试程序(不用串口送出数据)(C 和汇编)

1.C 程序:

```
/*----STC15 系列单片机 EEPROM/IAP 功能测试程序演示-------------------------*/
/*----演示 STC15 系列单片机 EEPROM/IAP 功能-----------------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可---*/

//假定测试芯片的工作频率为 18.432MHz

void IapIdle();
BYTE    IapReadByte(WORD addr);

#include "reg51.h"
#include "intrins.h"

typedef   unsigned char     BYTE;
typedef   unsigned int      WORD;
//-----------------------------------------------
sfr         IAP_DATA = 0xC2;            //IAP 数据寄存器
sfr         IAP_ADDRH = 0xC3;           //IAP 地址寄存器高字节
sfr         IAP_ADDRL = 0xC4;           //IAP 地址寄存器低字节
sfr         IAP_CMD = 0xC5;             //IAP 命令寄存器
sfr         IAP_TRIG = 0xC6;            //IAP 命令触发寄存器
sfr         IAP_CONTR = 0xC7;           //IAP 控制寄存器

#define     CMD_IDLE        0       //空闲模式
#define     CMD_READ        1       //IAP 字节读命令
#define     CMD_PROGRAM     2       //IAP 字节编程命令
#define     CMD_ERASE       3       //IAP 扇区擦除命令

//#define     ENABLE_IAP      0x80    //if SYSCLK<30MHz
//#define     ENABLE_IAP      0x81    //if SYSCLK<24MHz
#define     ENABLE_IAP      0x82    //if SYSCLK<20MHz
//#define     ENABLE_IAP      0x83    //if SYSCLK<12MHz
//#define     ENABLE_IAP      0x84    //if SYSCLK<6MHz
//#define     ENABLE_IAP      0x85    //if SYSCLK<3MHz
//#define     ENABLE_IAP      0x86    //if SYSCLK<2MHz
//#define     ENABLE_IAP      0x87    //if SYSCLK<1MHz

//测试地址
#define     IAP_ADDRESS     0x0400
```

```
void Delay(BYTE n);
void IapIdle();
BYTE IapReadByte(WORD addr);
void IapProgramByte(WORD addr, BYTE dat);
void IapEraseSector(WORD addr);

void main()
{
    WORD i;
    P1 = 0xfe;                          //1111,1110  系统 OK
    Delay(10);                          //延时
    IapEraseSector(IAP_ADDRESS);        //扇区擦除
    for (i=0; i<512; i++)               //检测是否擦除成功(全 FF 检测)
    {
        if (IapReadByte(IAP_ADDRESS+i) != 0xff)
            goto Error;                 //如果出错,则退出
    }
    P1 = 0xfc;                          //1111,1100  擦除成功
    Delay(10);                          //延时
    for (i=0; i<512; i++)               //编程 512 字节
    {
        IapProgramByte(IAP_ADDRESS+i, (BYTE)i);
    }
    P1 = 0xf8;                          //1111,1000  编程完成
    Delay(10);                          //延时
    for (i=0; i<512; i++)               //校验 512 字节
    {
        if (IapReadByte(IAP_ADDRESS+i) != (BYTE)i)
            goto Error;                 //如果校验错误,则退出
    }
    P1 = 0xf0;                          //1111,0000  测试完成
    while (1);
Error:
    P1 &= 0x7f;                         //0xxx,xxxx IAP 操作失败
    while (1);
}
/*--------------------------软件延时--------------------------*/
void Delay(BYTE n)
{
    WORD x;
    while (n--)
    {
        x = 0;
        while (++x);
    }
}
```

```
/*----------------------------关闭 IAP----------------------------*/
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_ADDRH = 0x80;               //将地址设置到非 IAP 区域
    IAP_ADDRL = 0;
}
/*----------------------------从 ISP/IAP/EEPROM 区域读取一字节----------------------------*/
BYTE IapReadByte(WORD addr)
{
    BYTE dat;                       //数据缓冲区
    EA = 0;                         //关闭总中断
    IAP_CONTR = ENABLE_IAP;         //使能 IAP
    IAP_CMD = CMD_READ;             //设置 IAP 命令
    IAP_ADDRL = addr;               //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;          //设置 IAP 高地址
    IAP_TRIG = 0x5a;                //写触发命令(0x5a)
    IAP_TRIG = 0xa5;                //写触发命令(0xa5)
    _nop_();                        //等待 ISP/IAP/EEPROM 操作完成
    dat = IAP_DATA;                 //读 ISP/IAP/EEPROM 数据
    IapIdle();                      //关闭 IAP 功能
    EA = 1;                         //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】
    return dat;                     //返回
}
/*----------------------------写一字节数据到 ISP/IAP/EEPROM 区域----------------------------*/
void IapProgramByte(WORD addr, BYTE dat)
{
    EA = 0;                         //关闭总中断
    IAP_CONTR = ENABLE_IAP;         //使能 IAP
    IAP_CMD = CMD_PROGRAM;          //设置 IAP 命令
    IAP_ADDRL = addr;               //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;          //设置 IAP 高地址
    IAP_DATA = dat;                 //写 ISP/IAP/EEPROM 数据
    IAP_TRIG = 0x5a;                //写触发命令(0x5a)
    IAP_TRIG = 0xa5;                //写触发命令(0xa5)
    _nop_();                        //等待 ISP/IAP/EEPROM 操作完成
    IapIdle();
    EA = 1;                         //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】
}
/*----------------------------扇区擦除----------------------------*/
```

```
void IapEraseSector(WORD addr)
{
    EA = 0;                         //关闭总中断
    IAP_CONTR = ENABLE_IAP;         //使能 IAP
    IAP_CMD = CMD_ERASE;            //设置 IAP 命令
    IAP_ADDRL = addr;               //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;          //设置 IAP 高地址
    IAP_TRIG = 0x5a;                //写触发命令(0x5a)
    IAP_TRIG = 0xa5;                //写触发命令(0xa5)
    _nop_();                        //等待 ISP/IAP/EEPROM 操作完成
    IapIdle();
    EA = 1;                         //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】
}
```

2. 汇编程序：

```
;STC15 系列单片机 EEPROM/IAP 功能测试程序演示
/*----演示 STC15 系列单片机 EEPROM/IAP 功能--------------------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-----*/
;假定测试芯片的工作频率为 18.432MHz
IAP_DATA        EQU     0C2H        ;IAP 数据寄存器
IAP_ADDRH       EQU     0C3H        ;IAP 地址寄存器高字
IAP_ADDRL       EQU     0C4H        ;IAP 地址寄存器低字
IAP_CMD         EQU     0C5H        ;IAP 命令寄存器
IAP_TRIG        EQU     0C6H        ;IAP 命令触发寄存器
IAP_CONTR       EQU     0C7H        ;IAP 控制寄存器

CMD_IDLE        EQU     0           ;空闲模式
CMD_READ        EQU     1           ;IAP 字节读命令
CMD_PROGRAM     EQU     2           ;IAP 字节编程命令
CMD_ERASE       EQU     3           ;IAP 扇区擦除命令

;ENABLE_IAP     EQU     80H         ;if SYSCLK<30MHz
;ENABLE_IAP     EQU     81H         ;if SYSCLK<24MHz
ENABLE_IAP      EQU     82H         ;if SYSCLK<20MHz
;ENABLE_IAP     EQU     83H         ;if SYSCLK<12MHz
;ENABLE_IAP     EQU     84H         ;if SYSCLK<6MHz
;ENABLE_IAP     EQU     85H         ;if SYSCLK<3MHz
;ENABLE_IAP     EQU     86H         ;if SYSCLK<2MHz
;ENABLE_IAP     EQU     87H         ;if SYSCLK<1MHz
;测试地址
IAP_ADDRESS     EQU     0400H
;------------------------------------------
    ORG     0000H
```

```
    LJMP    START
;---------------------------------------
    ORG     0100H
START:
    MOV     P1, #0FEH               ;1111,1110  系统 OK
    LCALL   DELAY                   ;延时
;---------------------------------------
    MOV     DPTR, #IAP_ADDRESS      ;设置 ISP/IAP/EEPROM 地址
    LCALL   IAP_ERASE               ;扇区擦除
;---------------------------------------
    MOV     DPTR, #IAP_ADDRESS      ;设置 ISP/IAP/EEPROM 地址
    MOV     R0, #0                  ;检测 512 字节
    MOV     R1, #2

CHECK1:                             ;检测是否擦除成功(全 FF 检测)
    LCALL   IAP_READ                ;读 IAP 数据
    CJNE    A, #0FFH, ERROR         ;如果出错,则退出
    INC     DPTR                    ;IAP 地址+1
    DJNZ    R0, CHECK1
    DJNZ    R1, CHECK1
;------------------------------
    MOV     P1, #0FCH               ;1111,1100  擦除成功
    LCALL   DELAY                   ;延时
;------------------------------
    MOV     DPTR, #IAP_ADDRESS      ;设置 ISP/IAP/EEPROM 地址
    MOV     R0, #0                  ;编程 512 字节
    MOV     R1, #2
    MOV     R2, #0
    NEXT:
    MOV     A, R2                   ;准备数据
    LCALL   IAP_PROGRAM             ;字节编程
    INC     DPTR                    ;IAP 地址+1
    INC     R2                      ;修改测试数据
    DJNZ    R0, NEXT
    DJNZ    R1, NEXT
;------------------------------
    MOV     P1, #0F8H               ;1111,1000  编程完成
    LCALL   DELAY                   ;延时
;------------------------------
    MOV     DPTR, #IAP_ADDRESS      ;设置 ISP/IAP/EEPROM 地址
    MOV     R0, #0                  ;校验 512 字节
    MOV     R1, #2
    MOV     R2, #0

CHECK2:
    LCALL   IAP_READ                ;读 IAP 数据
```

```
    CJNE    A, 2, ERROR             ;如果出错,则退出
    INC     DPTR                    ;IAP 地址+1
    INC     R2
    DJNZ    R0, CHECK2
    DJNZ    R1, CHECK2
;------------------------------
    MOV     P1, #0F0H               ;1111,0000  测试完成
    SJMP    $
;------------------------------
ERROR:
    MOV     P0, R0
    MOV     P2, R1
    MOV     P3, R2
    CLR     P1.7                    ;0xxx,xxxx IAP  测试失败
    SJMP    $
;----------------------------软件延时----------------------------
DELAY:
    CLR     A
    MOV     R0, A
    MOV     R1, A
    MOV     R2, #20H

DELAY1:
    DJNZ    R0, DELAY1
    DJNZ    R1, DELAY1
    DJNZ    R2, DELAY1
    RET
;--------------------------关闭 IAP--------------------------
IAP_IDLE:
    MOV     IAP_CONTR, #0           ;关闭 IAP 功能
    MOV     IAP_CMD, #0             ;清除命令寄存器
    MOV     IAP_TRIG, #0            ;清除触发寄存器
    MOV     IAP_ADDRH, #80H         ;将地址设置到非 IAP 区域
    MOV     IAP_ADDRL, #0

    RET
;-------------------------从 ISP/IAP/EEPROM 区域读取一字节-------------------------
IAP_READ:
    CLR     EA                      ;关闭总中断
    MOV     IAP_CONTR, #ENABLE_IAP  ;使能 IAP
    MOV     IAP_CMD, #CMD_READ      ;设置 IAP 命令
    MOV     IAP_ADDRL,DPL           ;设置 IAP 低地址
    MOV     IAP_ADDRH, DPH          ;设置 IAP 高地址
    MOV     IAP_TRIG, #5AH          ;写触发命令(0x5a)
    MOV     IAP_TRIG, #0A5H         ;写触发命令(0xa5)
    NOP                             ;等待 ISP/IAP/EEPROM 操作完成
```

```
    MOV     A, IAP_DATA             ;度 IAP 数据
    LCALL   IAP_IDLE                ;关闭 IAP 功能
    SETB    EA                      ;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】
    RET
;----------------------------写一字节数据到 ISP/IAP/EEPROM 区域----------------------------
IAP_PROGRAM:
    CLR     EA                      ;关闭总中断
    MOV     IAP_CONTR, #ENABLE_IAP  ;使能 IAP
    MOV     IAP_CMD, #CMD_PROGRAM   ;设置 IAP 命令
    MOV     IAP_ADDRL, DPL          ;设置 IAP 低地址
    MOV     IAP_ADDRH, DPH          ;设置 IAP 高地址
    MOV     IAP_DATA, A             ;写 IAP 数据
    MOV     IAP_TRIG, #5AH          ;写触发命令(0x5a)
    MOV     IAP_TRIG, #0A5H         ;写触发命令(0xa5)
    NOP                             ;等待 ISP/IAP/EEPROM 操作完成
    LCALL   IAP_IDLE                ;关闭 IAP 功能
    SETB    EA                      ;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】
    RET
;----------------------------扇区擦除----------------------------
IAP_ERASE:
    CLR     EA                      ;关闭总中断
    MOV     IAP_CONTR, #ENABLE_IAP  ;使能 IAP
    MOV     IAP_CMD, #CMD_ERASE     ;设置 IAP 命令
    MOV     IAP_ADDRL, DPL          ;设置 IAP 低地址
    MOV     IAP_ADDRH, DPH          ;设置 IAP 高地址
    MOV     IAP_TRIG, #5AH          ;写触发命令(0x5a)
    MOV     IAP_TRIG, #0A5H         ;写触发命令(0xa5)
    NOP                             ;等待 ISP/IAP/EEPROM 操作完成
    LCALL   IAP_IDLE                ;关闭 IAP 功能
    SETB    EA                      ;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】
    RET
END
```

20.4.2 EEPROM 测试程序(使用串口送出数据)(C 和汇编)

1.C 程序:

```
/*STC15 系列单片机 EEPROM/IAP 功能测试程序演示*/
/*----演示 STC15 系列单片机 EEPROM/IAP 功能--------------------------------------*/
```

```
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可------*/

//假定测试芯片的工作频率为 18.432MHz

#include "reg51.h"
#include "intrins.h"

typedef   unsigned char     BYTE;
typedef   unsigned int      WORD;
//-----------------------------------------------
sfr         IAP_DATA = 0xC2;        //IAP 数据寄存器
sfr         IAP_ADDRH = 0xC3;       //IAP 地址寄存器高字节
sfr         IAP_ADDRL = 0xC4;       //IAP 地址寄存器低字节
sfr         IAP_CMD = 0xC5;         //IAP 命令寄存器
sfr         IAP_TRIG = 0xC6;        //IAP 命令触发寄存器
sfr         IAP_CONTR = 0xC7;       //IAP 控制寄存器

#define     CMD_IDLE            0   //空闲模式
#define     CMD_READ            1   //IAP 字节读命令
#define     CMD_PROGRAM         2   //IAP 字节编程命令
#define     CMD_ERASE           3   //IAP 扇区擦除命令
#define     URMD                0   //0:使用定时器 2 作为波特率发生器
                                    //1:使用定时器 1 的模式 0(16 位自动重载模式)作为波特率发生器
                                    //2:使用定时器 1 的模式 2(8 位自动重载模式)作为波特率发生器
sfr     T2H = 0xd6;                 //定时器 2 高 8 位
sfr     T2L = 0xd7;                 //定时器 2 低 8 位
sfr     AUXR = 0x8e;                //辅助寄存器

//#define     ENABLE_IAP      0x80    //if SYSCLK<30MHz
//#define     ENABLE_IAP      0x81    //if SYSCLK<24MHz
#define     ENABLE_IAP      0x82    //if SYSCLK<20MHz
//#define     ENABLE_IAP      0x83    //if SYSCLK<12MHz
//#define     ENABLE_IAP      0x84    //if SYSCLK<6MHz
//#define     ENABLE_IAP      0x85    //if SYSCLK<3MHz
//#define     ENABLE_IAP      0x86    //if SYSCLK<2MHz
//#define     ENABLE_IAP      0x87    //if SYSCLK<1MHz

//测试地址
#define     IAP_ADDRESS     0x0400

void Delay(BYTE n);
void IapIdle();
BYTE IapReadByte(WORD addr);
void IapProgramByte(WORD addr, BYTE dat);
void IapEraseSector(WORD addr);
void InitUart();
```

```
BYTE SendData(BYTE dat);

void main()
{
    WORD i;
    P1 = 0xfe;                          //1111,1110  系统 OK
    InitUart();                         //初始化串口
    Delay(10);                          //延时
    IapEraseSector(IAP_ADDRESS);        //扇区擦除
    for (i=0; i<512; i++)               //检测是否擦除成功(全 FF 检测)
    {
        if (SendData(IapReadByte(IAP_ADDRESS+i)) != 0xff)
            goto Error;                 //如果出错,则退出
    }
    P1 = 0xfc;                          //1111,1100  擦除成功
    Delay(10);                          //延时
    for (i=0; i<512; i++)               //编程 512 字节
    {
        IapProgramByte(IAP_ADDRESS+i, (BYTE)i);
    }
    P1 = 0xf8;                          //1111,1000  编程完成
    Delay(10);                          //延时
    for (i=0; i<512; i++)               //校验 512 字节
    {
        if (SendData(IapReadByte(IAP_ADDRESS+i)) != (BYTE)i)
            goto Error;                 //如果校验错误,则退出
    }
    P1 = 0xf0;                          //1111,0000  测试完成
    while (1);
Error:
    P1 &= 0x7f;                         //0xxx,xxxx IAP 操作失败
    while (1);
}
/*----------------------------软件延时----------------------------*/
void Delay(BYTE n)
{
    WORD x;
    while (n--)
    {
    x = 0;
    while (++x);
    }
}
/*----------------------------关闭 IAP----------------------------*/
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_ADDRH = 0x80;               //将地址设置到非 IAP 区域
    IAP_ADDRL = 0;
}
/*----------------------------从 ISP/IAP/EEPROM 区域读取一字节----------------------------*/
BYTE IapReadByte(WORD addr)
{
    BYTE dat;                       //数据缓冲区
    EA = 0;                         //关闭总中断
    IAP_CONTR = ENABLE_IAP;         //使能 IAP
    IAP_CMD = CMD_READ;             //设置 IAP 命令
    IAP_ADDRL = addr;               //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;          //设置 IAP 高地址
    IAP_TRIG = 0x5a;                //写触发命令(0x5a)
    IAP_TRIG = 0xa5;                //写触发命令(0xa5)
    _nop_();                        //等待 ISP/IAP/EEPROM 操作完成
    dat = IAP_DATA;                 //读 ISP/IAP/EEPROM 数据
    IapIdle();                      //关闭 IAP 功能
    EA = 1;                         //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】
    return dat;                     //返回
}
/*----------------------------写一字节数据到 ISP/IAP/EEPROM 区域----------------------------*/
void IapProgramByte(WORD addr, BYTE dat)
{
    EA = 0;                         //关闭总中断
    IAP_CONTR = ENABLE_IAP;         //使能 IAP
    IAP_CMD = CMD_PROGRAM;          //设置 IAP 命令
    IAP_ADDRL = addr;               //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;          //设置 IAP 高地址
    IAP_DATA = dat;                 //写 ISP/IAP/EEPROM 数据
    IAP_TRIG = 0x5a;                //写触发命令(0x5a)
    IAP_TRIG = 0xa5;                //写触发命令(0xa5)
    _nop_();                        //等待 ISP/IAP/EEPROM 操作完成
    IapIdle();
    EA = 1;                         //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】
}
/*----------------------------扇区擦除----------------------------*/
void IapEraseSector(WORD addr)
{
    EA = 0;                         //关闭总中断
```

```
    IAP_CONTR = ENABLE_IAP;          //使能 IAP
    IAP_CMD = CMD_ERASE;             //设置 IAP 命令
    IAP_ADDRL = addr;                //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;           //设置 IAP 高地址
    IAP_TRIG = 0x5a;                 //写触发命令(0x5a)
    IAP_TRIG = 0xa5;                 //写触发命令(0xa5)
    _nop_();                         //等待 ISP/IAP/EEPROM 操作完成
    IapIdle();
    EA = 1;                          //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】
}
/*----------------------------初始化串口----------------------------*/
void InitUart()
{
    SCON = 0x5a;                     //设置串口为 8 位可变波特率
#if URMD == 0
    T2L = 0xd8;                      //设置波特率重装值
    T2H = 0xff;                      //115200 bps(65536-18432000/4/115200)
    AUXR = 0x14;                     //T2 为 1T 模式, 并启动定时器 2
    AUXR |= 0x01;                    //选择定时器 2 为串口 1 的波特率发生器
#elif URMD == 1
    AUXR = 0x40;                     //定时器 1 为 1T 模式
    TMOD = 0x00;                     //定时器 1 为模式 0(16 位自动重载)
    TL1 = 0xd8;                      //设置波特率重装值
    TH1 = 0xff;                      //115200 bps(65536-18432000/4/115200)
    TR1 = 1;                         //定时器 1 开始启动
#else
    TMOD = 0x20;                     //设置定时器 1 为 8 位自动重装载模式
    AUXR = 0x40;                     //定时器 1 为 1T 模式
    TH1 = TL1 = 0xfb;                //115200 bps(256 - 18432000/32/115200)
    TR1 = 1;
#endif
}
/*----------------------------发送串口数据----------------------------*/
BYTE SendData(BYTE dat)
{
    while (!TI);                     //等待前一个数据发送完成
    TI = 0;                          //清除发送标志
    SBUF = dat;                      //发送当前数据
    return dat;
}
```

2. 汇编程序：

```
;STC15 系列单片机 EEPROM/IAP 功能测试程序演示
/*----演示 STC15 系列单片机 EEPROM/IAP 功能--------------------------------------*/
```

```
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-----*/
//假定测试芯片的工作频率为 18.432MHz
#define         URMD    0       //0:使用定时器 2 作为波特率发生器
//1:使用定时器 1 的模式 0(16 位自动重载模式)作为波特率发生器
//2:使用定时器 1 的模式 2(8 位自动重载模式)作为波特率发生器
T2H             DATA    0D6H    //定时器 2 高 8 位
T2L             DATA    0D7H    //定时器 2 低 8 位
AUXR            DATA    08EH    //辅助寄存器

IAP_DATA        EQU     0C2H    //IAP 数据寄存器
IAP_ADDRH       EQU     0C3H    //IAP 地址寄存器高字
IAP_ADDRL       EQU     0C4H    //IAP 地址寄存器低字
IAP_CMD         EQU     0C5H    //IAP 命令寄存器
IAP_TRIG        EQU     0C6H    //IAP 命令触发寄存器
IAP_CONTR       EQU     0C7H    //IAP 控制寄存器

CMD_IDLE        EQU     0       //空闲模式
CMD_READ        EQU     1       //IAP 字节读命令
CMD_PROGRAM     EQU     2       //IAP 字节编程命令
CMD_ERASE       EQU     3       //IAP 扇区擦除命令

;ENABLE_IAP     EQU     80H     //if SYSCLK<30MHz
;ENABLE_IAP     EQU     81H     //if SYSCLK<24MHz
ENABLE_IAP      EQU     82H     //if SYSCLK<20MHz
;ENABLE_IAP     EQU     83H     //if SYSCLK<12MHz
;ENABLE_IAP     EQU     84H     //if SYSCLK<6MHz
;ENABLE_IAP     EQU     85H     //if SYSCLK<3MHz
;ENABLE_IAP     EQU     86H     //if SYSCLK<2MHz
;ENABLE_IAP     EQU     87H     //if SYSCLK<1MHz

//测试地址
IAP_ADDRESS     EQU     0400H
//----------------------------------------
    ORG     0000H
    LJMP    START
;----------------------------------------
    ORG     0100H

START:
    LCALL   INIT_UART           //初始化串口
    MOV     P1, #0FEH           //1111,1110  系统 OK
    LCALL   DELAY               //延时
;--------------------------------
    MOV     DPTR, #IAP_ADDRESS  //设置 ISP/IAP/EEPROM 地址
    LCALL   IAP_ERASE           //扇区擦除
;--------------------------------
```

```
    MOV     DPTR, #IAP_ADDRESS      //设置 ISP/IAP/EEPROM 地址
    MOV     R0, #0                  //检测 512 字节
    MOV     R1, #2

CHECK1:                             //检测是否擦除成功(全 FF 检测)
    LCALL   IAP_READ                //读 IAP 数据
    LCALL   SEND_DATA
    CJNE    A, #0FFH,ERROR          //如果出错,则退出
    INC     DPTR                    //IAP 地址+1
    DJNZ    R0, CHECK1
    DJNZ    R1, CHECK1
;------------------------------
    MOV     P1, #0FCH               //1111,1100  擦除成功
    LCALL   DELAY                   //延时
;------------------------------
    MOV     DPTR, #IAP_ADDRESS      //设置 ISP/IAP/EEPROM 地址
    MOV     R0, #0                  //编程 512 字节
    MOV     R1, #2
    MOV     R2, #0

NEXT:
    MOV     A,R2                    //准备数据
    LCALL   IAP_PROGRAM             //字节编程
    INC     DPTR                    //IAP 地址+1
    INC     R2                      //修改测试数据
    DJNZ    R0, NEXT
    DJNZ    R1, NEXT
;------------------------------
    MOV     P1, #0F8H               //1111,1000  编程完成
    LCALL   DELAY                   //延时
;------------------------------
    MOV     DPTR, #IAP_ADDRESS      //设置 ISP/IAP/EEPROM 地址
    MOV     R0, #0                  //校验 512 字节
    MOV     R1, #2
    MOV     R2, #0

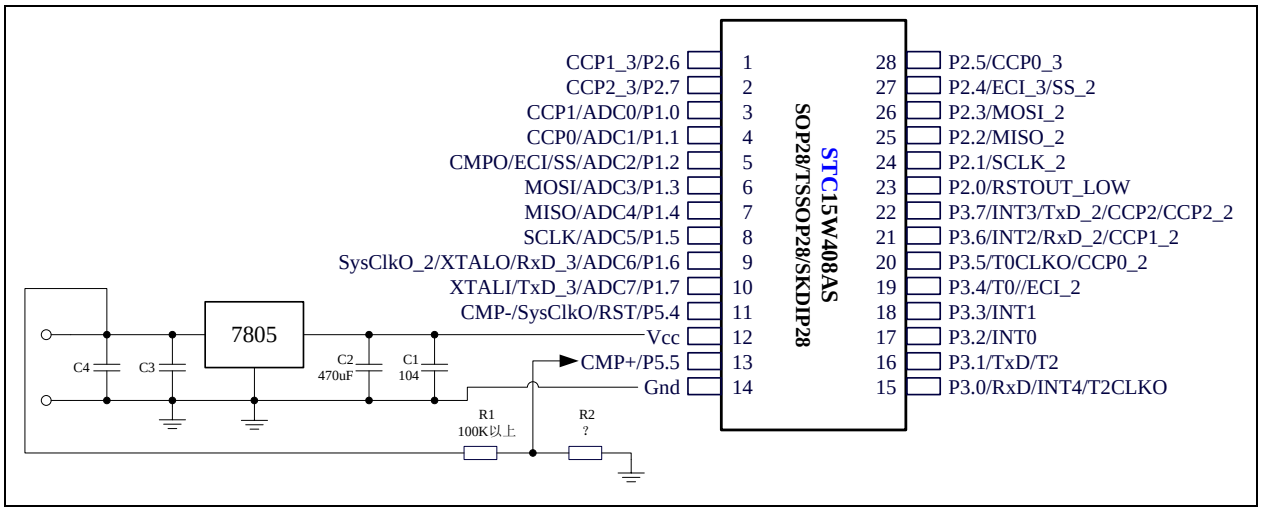
CHECK2:
    LCALL   IAP_READ                //读 IAP 数据
    LCALL   SEND_DATA
    CJNE    A, 2, ERROR             //如果出错,则退出
    INC     DPTR                    //IAP 地址+1
    INC     R2
    DJNZ    R0, CHECK2
    DJNZ    R1, CHECK2
;------------------------------
    MOV     P1, #0F0H               //1111,0000  测试完成
```

```
    SJMP    $
;------------------------------
ERROR:
    MOV     P0, R0
    MOV     P2, R1
    MOV     P3, R2
    CLR     P1.7                    //0xxx,xxxx IAP 测试失败
    SJMP    $
;/*----------------------------软件延时----------------------------*/
DELAY:
    CLR     A
    MOV     R0, A
    MOV     R1, A
    MOV     R2, #20H
DELAY1:
    DJNZ    R0, DELAY1
    DJNZ    R1, DELAY1
    DJNZ    R2, DELAY1
    RET
;/*----------------------------关闭 IAP----------------------------*/
IAP_IDLE:
    MOV     IAP_CONTR, #0           //关闭 IAP 功能
    MOV     IAP_CMD, #0             //清除命令寄存器
    MOV     IAP_TRIG, #0            //清除触发寄存器
    MOV     IAP_ADDRH, #80H         //将地址设置到非 IAP 区域
    MOV     IAP_ADDRL, #0
    RET
;/*----------------------------从 ISP/IAP/EEPROM 区域读取一字节----------------------------*/
IAP_READ:
    CLR     EA                      ;关闭总中断
    MOV     IAP_CONTR, #ENABLE_IAP  //使能 IAP
    MOV     IAP_CMD, #CMD_READ      //设置 IAP 命令
    MOV     IAP_ADDRL, DPL          //设置 IAP 低地址
    MOV     IAP_ADDRH, DPH          //设置 IAP 高地址
    MOV     IAP_TRIG, #5AH          //写触发命令(0x5a)
    MOV     IAP_TRIG, #0A5H         //写触发命令(0xa5)
    NOP                             //等待 ISP/IAP/EEPROM 操作完成
    MOV     A, IAP_DATA             //度 IAP 数据
    LCALL   IAP_IDLE                //关闭 IAP 功能
    SETB    EA                      ;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】
    RET
;/*----------------------------写一字节数据到 ISP/IAP/EEPROM 区域----------------------------*/
IAP_PROGRAM:
    CLR     EA                      ;关闭总中断
```

```
    MOV     IAP_CONTR, #ENABLE_IAP    //使能 IAP
    MOV     IAP_CMD, #CMD_PROGRAM     //设置 IAP 命令
    MOV     IAP_ADDRL, DPL            //设置 IAP 低地址
    MOV     IAP_ADDRH, DPH            //设置 IAP 高地址
    MOV     IAP_DATA, A               //写 IAP 数据
    MOV     IAP_TRIG, #5AH            //写触发命令(0x5a)
    MOV     IAP_TRIG, #0A5H           //写触发命令(0xa5)
    NOP                               //等待 ISP/IAP/EEPROM 操作完成
    LCALL   IAP_IDLE                  //关闭 IAP 功能
    SETB    EA                        ;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】
    RET
;/*---------------------------扇区擦除---------------------------*/
IAP_ERASE:
    CLR     EA                        ;关闭总中断
    MOV     IAP_CONTR, #ENABLE_IAP    //使能 IAP
    MOV     IAP_CMD, #CMD_ERASE       //设置 IAP 命令
    MOV     IAP_ADDRL, DPL            //设置 IAP 低地址
    MOV     IAP_ADDRH, DPH            //设置 IAP 高地址
    MOV     IAP_TRIG, #5AH            //写触发命令(0x5a)
    MOV     IAP_TRIG, #0A5H           //写触发命令(0xa5)
    NOP                               //等待 ISP/IAP/EEPROM 操作完成
    LCAL    L IAP_IDLE                //关闭 IAP 功能
    SETB    EA                        ;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】
    RET
;/*---------------------------初始化串口---------------------------*/
INIT_UART:
    MOV     SCON, #5AH                ;设置串口为 8 位可变波特率
#if URMD == 0
    MOV     T2L, #0D8H                ;设置波特率重装值(65536-18432000/4/115200)
    MOV     T2H, #0FFH
    MOV     AUXR, #14H                ;T2 为 1T 模式, 并启动定时器 2
    ORL     AUXR, #01H                ;选择定时器 2 为串口 1 的波特率发生器
#elif URMD == 1
    MOV     AUXR, #40H                ;定时器 1 为 1T 模式
    MOV     TMOD, #00H                ;定时器 1 为模式 0(16 位自动重载)
    MOV     TL1, #0D8H                ;设置波特率重装值(65536-18432000/4/115200)
    MOV     TH1, #0FFH
    SETB    TR1                       ;定时器 1 开始运行
#else
    MOV     TMOD, #20H                ;设置定时器 1 为 8 位自动重装载模式
    MOV     AUXR, #40H                ;定时器 1 为 1T 模式
    MOV     TL1, #0FBH                ;115200 bps(256 - 18432000/32/115200)
```

```
    MOV     TH1, #0FBH
    SETB    TR1
#endif
    RET
;-------------------------发送串口数据-------------------------
SEND_DATA:
    JNB     TI, $                   ;等待前一个数据发送完成
    CLR     TI                      ;清除发送标志
    MOV     SBUF, A                 ;发送当前数据
    RET
END
```

20.5 比较器作外部掉电检测的参考电路



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 P5.5/CMP+的外部输入与内部 BandGap 参考电压(1.27V 附近)进行比较。

一般当交流电在 220V 时，稳压块 7805 前端的直流电压是 11V，但当交流电压降到 160V 时，稳压块 7805 前端的直流电压是 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 CMP+端(比较器正极输入端)，CMP+端输入电压低于内部 BandGap 参考电压(1.27V 附近)，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 CMP+端(比较器正极输入端)，CMP+端输入电压高于内部 BandGap 参考电压(1.27V 附近)，此时 CPU 可继续正常工作。

内部 BandGap 参考电压约在 1.27V 附近，具体数值要通过读取内部 BandGap 电压在内部 RAM 区或 ROM 区所占用的地址的值获得。

- 对于具有 128 字节 RAM 空间的单片机(如 STC15W104 系列单片机)，其内部 BandGap 参考电压值在 RAM 区占用的地址为 06FH - 070H，在 ROM 区占用的地址为程序空间最后第 8 字节和第 9 字节(如 STC15W104 型号单片机具有 4K 程序空间，则其内部 BandGap 参考电压值在 ROM 区占用的地址为 0FF7H - 0FF8H)，用户只需通过读取 RAM 区 06FH - 070H 地址的值或 ROM 区

OFF7H - 0FF8H 地址的值即可获得 STC15W104 型号单片机的内部 BandGap 参考电压值(毫伏, 高字节在前)。

- 对于具有 256 及其以上字节 RAM 空间的单片机(如 STC15W4K32S4 系列单片机)，其内部 BandGap 参考电压值在 RAM 区占用的地址为 0EFH - 0F0H，在 ROM 区占用的地址为程序空间最后第 8 字节和第 9 字节(如 STC15W4K32S4 型号单片机具有 32K 程序空间，则其内部 BandGap 参考电压值在 ROM 区占用的地址为 7FF7H - 7FF8H)，用户只需通过读取 RAM 区 0EFH - 0F0H 地址的值或 ROM 区 7FF7H - 7FF8H 地址的值即可获得 STC15W4K32S4 型号单片机的内部 BandGap 参考电压值(毫伏，高字节在前)。

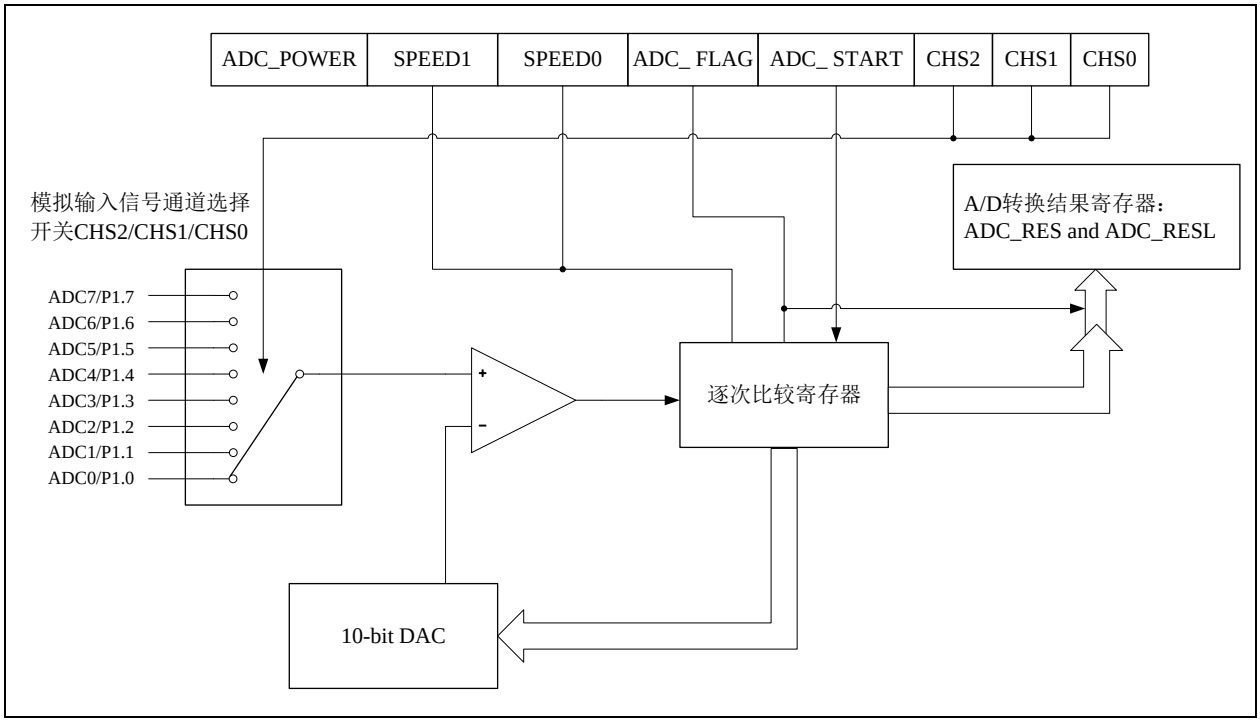
21 STC15 系列单片机的 A/D 转换器

下表总结了 STC15 系列单片机内部集成了 8 路 10 位高速 A/D 转换器的单片机型号:

特殊外围设备 / 单片机型号	8 路 10 位高速 A/D 转换器	CCP/PCA/PWM 功能	1 组高速同步串行口 SPI
STC15W4K32S4 系列	√	√	√
STC15F2K60S2 系列	√	√	√
STC15W1K16S 系列			√
STC15W404S 系列			√
STC15W408AS 系列	√	√	√
STC15W204S 系列			
STC15F408AD 系列	√	√	√
STC15F104W 系列			

21.1 A/D 转换器的结构

STC15 系列单片机 ADC(A/D 转换器)的结构如下图所示。



当 CLK_DIV.5(PCON2.5)/ADRJ = 0 时，A/D 转换结果寄存器格式如下：

ADC_RES[7:0]

ADC_B9	ADC_B8	ADC_B7	ADC_B6	ADC_B5	ADC_B4	ADC_B3	ADC_B2

-	-	-	-	-	-	ADC_B1	ADC_B0

ADC_RESL[1:0]

当 CLK_DIV.5(PCON2.5)/ADRJ = 1 时，A/D 转换结果寄存器格式如下：

ADC_RES[1:0]

-	-	-	-	-	-	ADC_B9	ADC_B8

ADC_B7	ADC_B6	ADC_B5	ADC_B4	ADC_B3	ADC_B2	ADC_B1	ADC_B0

ADC_RESL[7:0]

STC15 系列单片机 ADC 由多路选择开关、比较器、逐次比较寄存器、10 位 DAC、转换结果寄存器（ADC_RES 和 ADC_RESL）以及 ADC_CONTR 构成。

STC15 系列单片机的 ADC 是逐次比较型 ADC。逐次比较型 ADC 由一个比较器和 D/A 转换器构成，通过逐次比较逻辑，从最高位（MSB）开始，顺序地对每一输入电压与内置 D/A 转换器输出进行比较，经过多次比较，使转换所得的数字量逐次通近输入模拟量对应值。逐次比较型 A/D 转换器具有速度高，功耗低等优点。

从上图可以看出，通过模拟多路开关，将通过 ADC0 ~ 7 的模拟量输入送给比较器。用数/模转换器（DAC）转换的模拟量与输入的模拟量通过比较器进行比较，将比较结果保存到逐次比较寄存器，并通过逐次比较寄存器输出转换结果。A/D 转换结束后，最终的转换结果保存到 ADC 转换结果寄存器 ADC_RES 和 ADC_RESL，同时，置位 ADC 控制寄存器 ADC_CONTR 中的 A/D 转换结束标志位 ADC_FLAG，以供程序查询或发出中断申请。模拟通道的选择控制由 ADC 控制寄存器 ADC_CONTR 中的 CHS2 ~ CHS0 确定。ADC 的转换速度由 ADC 控制寄存器中的 SPEED1 和 SPEED0 确定。在使用 ADC 之前，应先给 ADC 上电，也就是置位 ADC 控制寄存器中的 ADC_POWER 位。

- 当 ADRJ=0 时，如果取 10 位结果，则按下面公式计算：

$$10_bit\ A/D\ Conversion\ Result:(ADC_RES[7:0], ADC_RESL[1:0]) = 1024 \times \frac{Vin}{Vcc}$$

- 当 ADRJ=0 时，如果取 8 位结果，则按下面公式计算：

$$8_bit\ A/D\ Conversion\ Result:(ADC_RES[7:0]) = 256 \times \frac{Vin}{Vcc}$$

- 当 ADRJ=1 时，如果取 10 位结果，则按下面公式计算：

$$10_bit\ A/D\ Conversion\ Result:(ADC_RES[1:0], ADC_RESL[7:0]) = 1024 \times \frac{Vin}{Vcc}$$

式中，Vin 为模拟输入通道输入电压，Vcc 为单片机实际工作电压，用单片机工作电压作为模拟参考电压。

21.2 与 A/D 转换相关的寄存器

与 STC15 系列单片机 A/D 转换相关的寄存器列于下表所示。

符号	描述	地址	位地址及其符号								复位值
			MSB							LSB	
P1ASF	P1 Analog Function Configure register	9DH	P17ASF	P16ASF	P15ASF	P14ASF	P13ASF	P12ASF	P11ASF	P10ASF	0000 0000B
ADC_CONTR	ADC Control Register	BCH	ADC_POWER	SPEED1	SPEED0	ADC_FLAG	ADC_START	CHS2	CHS1	CHS0	0000 0000B
ADC_RES	ADC Result high	BDH									0000 0000B
ADC_RESL	ADC Result low	BEH									0000 0000B
CLK_DIV PCON2	时钟分频寄存器	97H	MCKO_S1	MCKO_S1	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000 0000B
IE	Interrupt Enable	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000 0000B
IP	Interrupt Priority Low	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	0000 0000B

1、P1 口模拟功能控制寄存器 P1ASF

STC15 系列单片机的 A/D 转换口在 P1 口（P1.7-P1.0），有 8 路 10 位高速 A/D 转换器，速度可达到 300KHz（30 万次/秒）。8 路电压输入型 A/D，可做温度检测、电池电压检测、按键扫描、频谱检测等。上电复位后 P1 口为弱上拉型 I/O 口，用户可以通过软件设置将 8 路中的任何一路设置为 A/D 转换，不需作为 A/D 使用的 P1 口可继续作为 I/O 口使用（建议只作为输入）。需作为 A/D 使用的口需先将 P1ASF 特殊功能寄存器中的相应位置为'1'，将相应的口设置为模拟功能。P1ASF 寄存器的格式如下：

P1ASF：P1 口模拟功能控制寄存器（该寄存器是只写寄存器，读无效）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P1ASF	9DH	name	P17ASF	P16ASF	P15ASF	P14ASF	P13ASF	P12ASF	P11ASF	P10ASF

P1ASF[7：0]	P1.x 的功能	其中 P1ASF 寄存器地址为：[9DH]（不能够进行位寻址）
P1ASF.0 = 1	P1.0 口作为模拟功能 A/D 使用	
P1ASF.1 = 1	P1.1 口作为模拟功能 A/D 使用	
P1ASF.2 = 1	P1.2 口作为模拟功能 A/D 使用	
P1ASF.3 = 1	P1.3 口作为模拟功能 A/D 使用	
P1ASF.4 = 1	P1.4 口作为模拟功能 A/D 使用	
P1ASF.5 = 1	P1.5 口作为模拟功能 A/D 使用	
P1ASF.6 = 1	P1.6 口作为模拟功能 A/D 使用	
P1ASF.7 = 1	P1.7 口作为模拟功能 A/D 使用	

2、ADC 控制寄存器 ADC_CONTR

ADC_CONTR 寄存器的格式如下：

ADC_CONTR：ADC 控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
ADC_CONTR	BCH	name	ADC_POWER	SPEED1	SPEED0	ADC_FLAG	ADC_START	CHS2	CHS1	CHS0

对 ADC_CONTR 寄存器进行操作，建议直接用 M0V 赋值语句，不要用'与’和'或’语句。

ADC_POWER：ADC 电源控制位。

- 关闭 ADC 电源；
- 打开 A/D 转换器电源

  建议进入空闲模式和掉电模式前，将 ADC 电源关闭，即 ADC_POWER=0，可降低功耗。启动 A/D 转换前一定要确认 A/D 电源已打开，A/D 转换结束后关闭 A/D 电源可降低功耗，也可不关闭。初次打开内部 A/D 转换模拟电源，需适当延时，等内部模拟电源稳定后，再启动 A/D 转换。

  建议启动 A/D 转换后，在 A/D 转换结束之前，不改变任何 I/O 口的状态，有利于高精度 A/D 转换，如能将定时器/串行口/中断系统关闭更好。

SPEED1，SPEED0：模数转换器转换速度控制位

SPEED1	SPEED0	A/D 转换所需时间
1	1	90 个时钟周期转换一次，CPU 工作频率 27MHz 时， A/D 转换速度约 300KHz（=27MHz÷90）
1	0	180 个时钟周期转换一次
0	1	360 个时钟周期转换一次
0	0	540 个时钟周期转换一次

ADC_FLAG：模数转换器转换结束标志位。当 A/D 转换完成后，ADC_FLAG=1，要由软件清 0。不管是 A/D 转换完成后由该位申请产生中断，还是由软件查询该标志位 A/D 转换是否结束，当 A/D 转换完成后，ADC_FLAG=1，一定要软件清 0。

ADC_START：模数转换器（ADC）转换启动控制位。设置为“1”时，开始转换，转换结束后为 0。

CHS2 / CHS1 / CHS0：模拟输入通道选择，CHS2 / CHS1 / CHS0

CHS2	CHS1	CHS0	Analog Channel Select （模拟输入通道选择）
0	0	0	选择 P1.0 作为 A/D 输入来用
0	0	1	选择 P1.1 作为 A/D 输入来用
0	1	0	选择 P1.2 作为 A/D 输入来用
0	1	1	选择 P1.3 作为 A/D 输入来用
1	0	0	选择 P1.4 作为 A/D 输入来用
1	0	1	选择 P1.5 作为 A/D 输入来用
1	1	0	选择 P1.6 作为 A/D 输入来用
1	1	1	选择 P1.7 作为 A/D 输入来用

3、ADC 转换结果调整寄存器位——ADRJ

ADC 转换结果调整寄存器位----ADRJ 位于寄存器 CLK_DIV/PCON 中，用于控制 ADC 转换结果存放的位置。

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
CLK_DIV（PCON2）	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000, x000

ADRJ： ADC 转换结果调整

- 0，ADC_RES[7：0]存放高 8 位 ADC 结果，ADC_RESL[1：0]存放低 2 位 ADC 结果
- 1，ADC_RES[1：0]存放高 2 位 ADC 结果，ADC_RESL[7：0]存放低 8 位 ADC 结果

4、A/D 转换结果寄存器 ADC_RES、ADC_RESL

特殊功能寄存器 ADC_RES 和 ADC_RESL 寄存器用于保存 A/D 转换结果，其格式如下：

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0
ADC_RES	BDh	A/D 转换结果寄存器高								
ADC_RESL	BEh	A/D 转换结果寄存器低								
CLK_DIV（PCON2）	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0

CLK_DIV 寄存器的 ADRJ 位是 A/D 转换结果寄存器（ADC_RES，ADC_RESL）的数据格式调整控制位。

当 ADRJ=0 时，10 位 A/D 转换结果的高 8 位存放在 ADC_RES 中，
低 2 位存放在 ADC_RESL 的低 2 位中。

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0
ADC_RES	BDh	A/D 转换结果寄存器高 8 位	ADC_RES9	ADC_RES8	ADC_RES7	ADC_RES6	ADC_RES5	ADC_RES4	ADC_RES3	ADC_RES2
ADC_RESL	BEh	A/D 转换结果寄存器低 2 位	-	-	-	-	-	-	ADC_RES1	ADC_RES0
CLK_DIV（PCON2）	97H	时钟分频寄存器			ADRJ = 0					

此时，如果用户需取完整 10 位结果，按下面公式计算：

$$10_bit\ A/D\ Conversion\ Result:(ADC_RES[7:0],ADC_RESL[1:0]) = 1024 \times \frac{Vin}{Vcc}$$

如果用户只需取 8 位结果，按下面公式计算：

$$8_bit\ A/D\ Conversion\ Result:(ADC_RES[7:0]) = 256 \times \frac{Vin}{Vcc}$$

式中，Vin 为模拟输入通道输入电压，Vcc 为单片机实际工作电压，用单片机工作电压作为模拟参考电压。

当 ADRJ=1 时，10 位 A/D 转换结果的高 2 位存放在 ADC_RES 的低 2 位中，
低 8 位存放在 ADC_RESL 中。

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0
ADC_RES	BDh	A/D 转换结果寄存器高 2 位	-	-	-	-	-	-	ADC_RES9	ADC_RES8
ADC_RESL	BEh	A/D 转换结果寄存器低 8 位	ADC_RES7	ADC_RES6	ADC_RES5	ADC_RES4	ADC_RES3	ADC_RES2	ADC_RES1	ADC_RES0
CLK_DIV（PCON2）	97H	时钟分频寄存器			ADRJ = 1					

此时，如果用户需取完整 10 位结果，按下面公式计算：

$$10_bit\ A/D\ Conversion\ Result:(ADC_RES[1:0],ADC_RESL[7:0]) = 1024 \times \frac{Vin}{Vcc}$$

式中，Vin 为模拟输入通道输入电压，Vcc 为单片机实际工作电压，用单片机工作电压作为模拟参考电压。

5、中断允许寄存器 IE

IE：中断允许寄存器 （可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	name	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

EA： CPU 的中断开放标志

- EA=1，CPU 开放中断，
- EA=0，CPU 屏蔽所有的中断申请。

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。

EADC： A/D 转换中断允许位

- EADC=1，允许 A/D 转换中断；
- EADC=0，禁止 A/D 转换中断。

6、中断优先级控制寄存器 IP

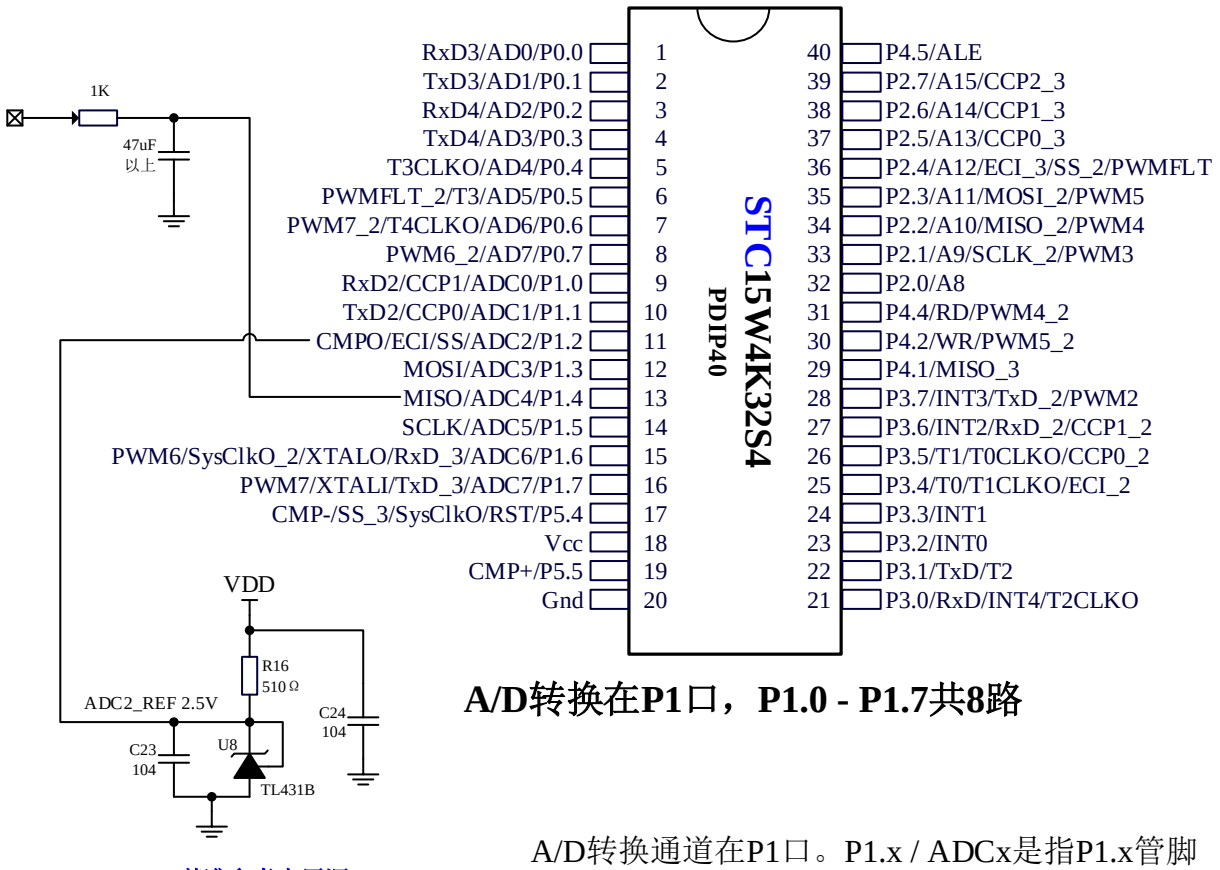
IP：中断优先级控制寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IP	B8H	name	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0

PADC： A/D 转换中断优先级控制位。

- 当 PADC=0 时，A/D 转换中断为最低优先级中断（优先级 0）
- 当 PADC=1 时，A/D 转换中断为最高优先级中断（优先级 1）

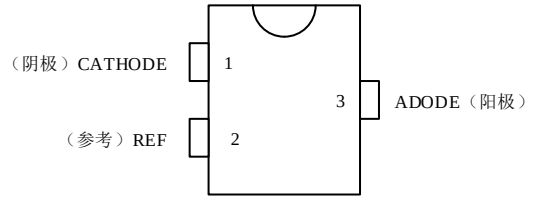
21.3 A/D 转换典型应用线路



A/D转换在P1口，P1.0 - P1.7共8路

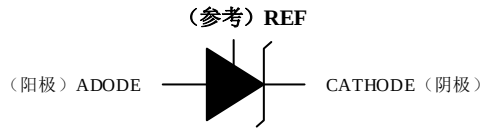
A/D转换通道在P1口。P1.x / ADCx是指P1.x管脚可做为A/D转换通道使用

基准参考电压源TL431B



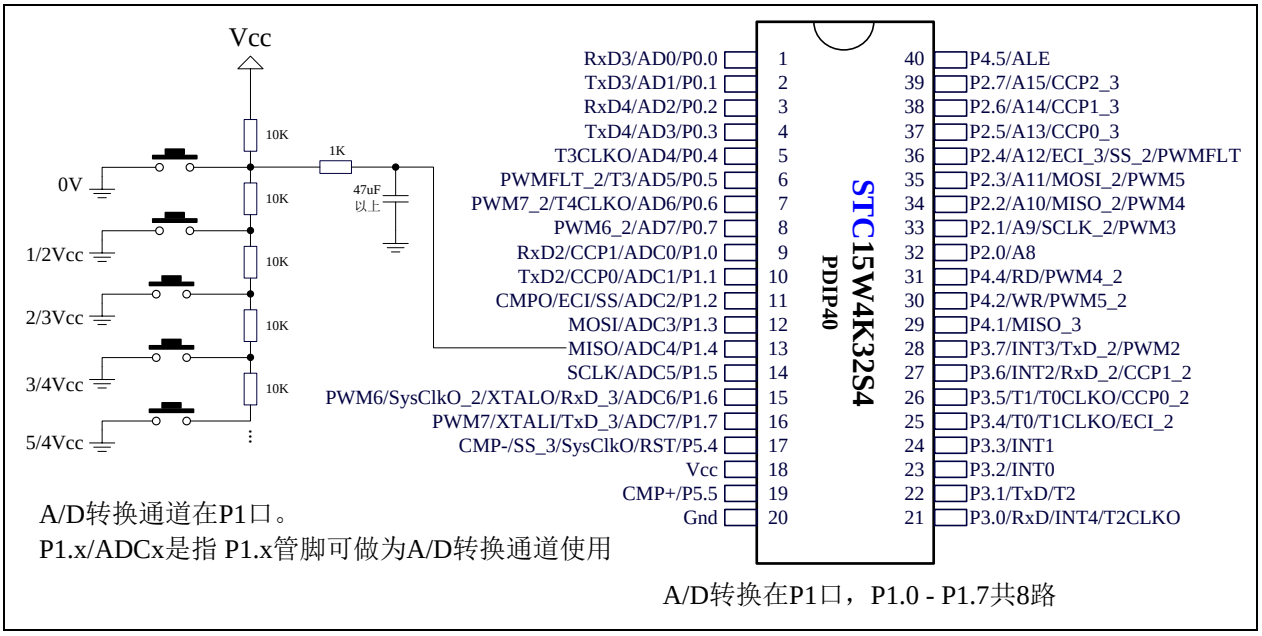
SOT23-3封装，RMB￥0.15~0.3

基准参考电压源TL431B的符号



如应用简单，可无需基准参考电压源，直接与Vcc比较即可。

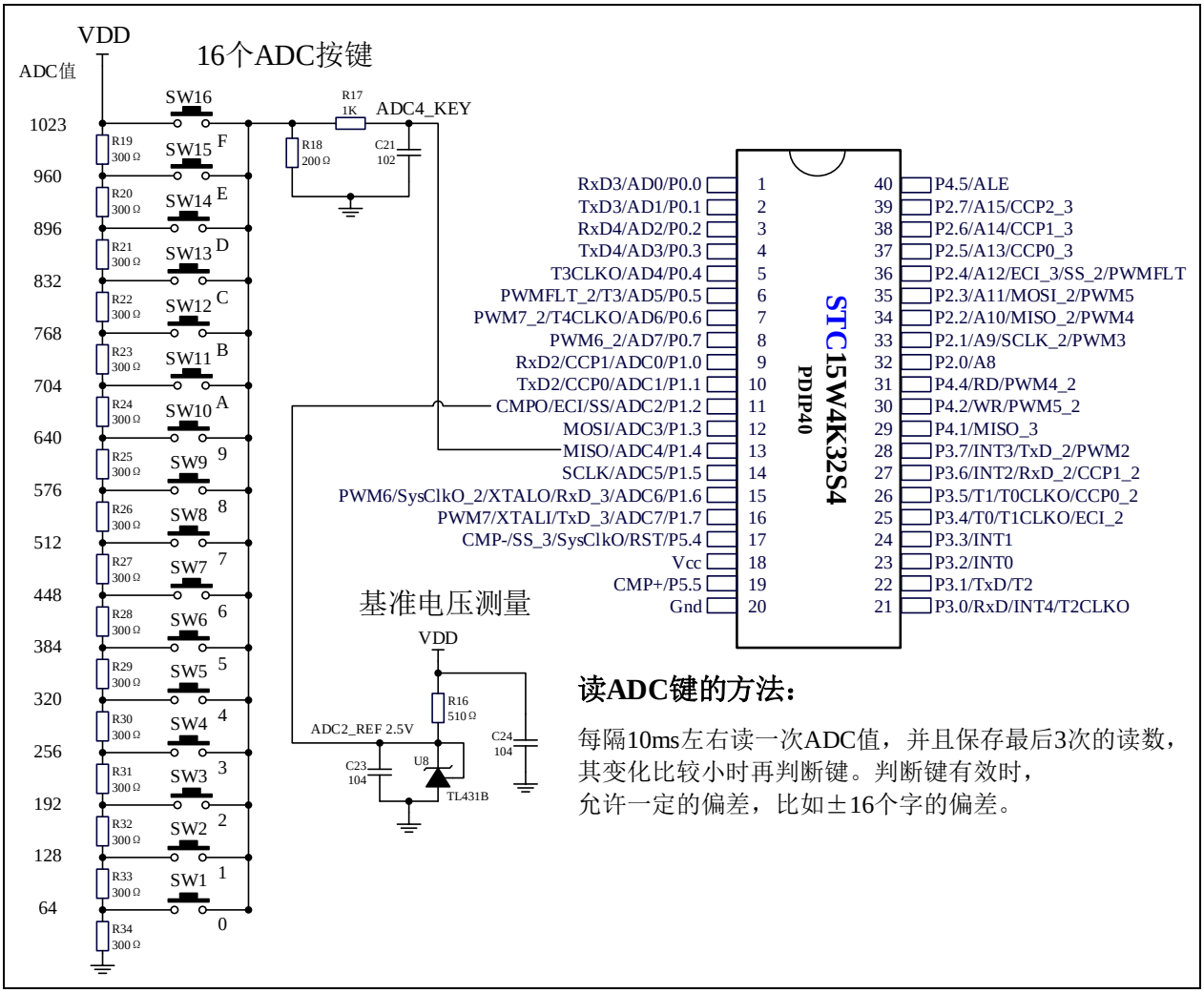
21.4 A/D 作按键扫描应用线路图



A/D转换通道在P1口。
P1.x/ADCx是指 P1.x管脚可做为A/D转换通道使用

A/D转换在P1口，P1.0 - P1.7共8路

读 ADC 键的方法：每隔 10ms 左右读一次 ADC 值，并且保存最后 3 次的读数，其变化比较小时再判断键。判断键有效时，允许一定的偏差，比如±16 个字的偏差。



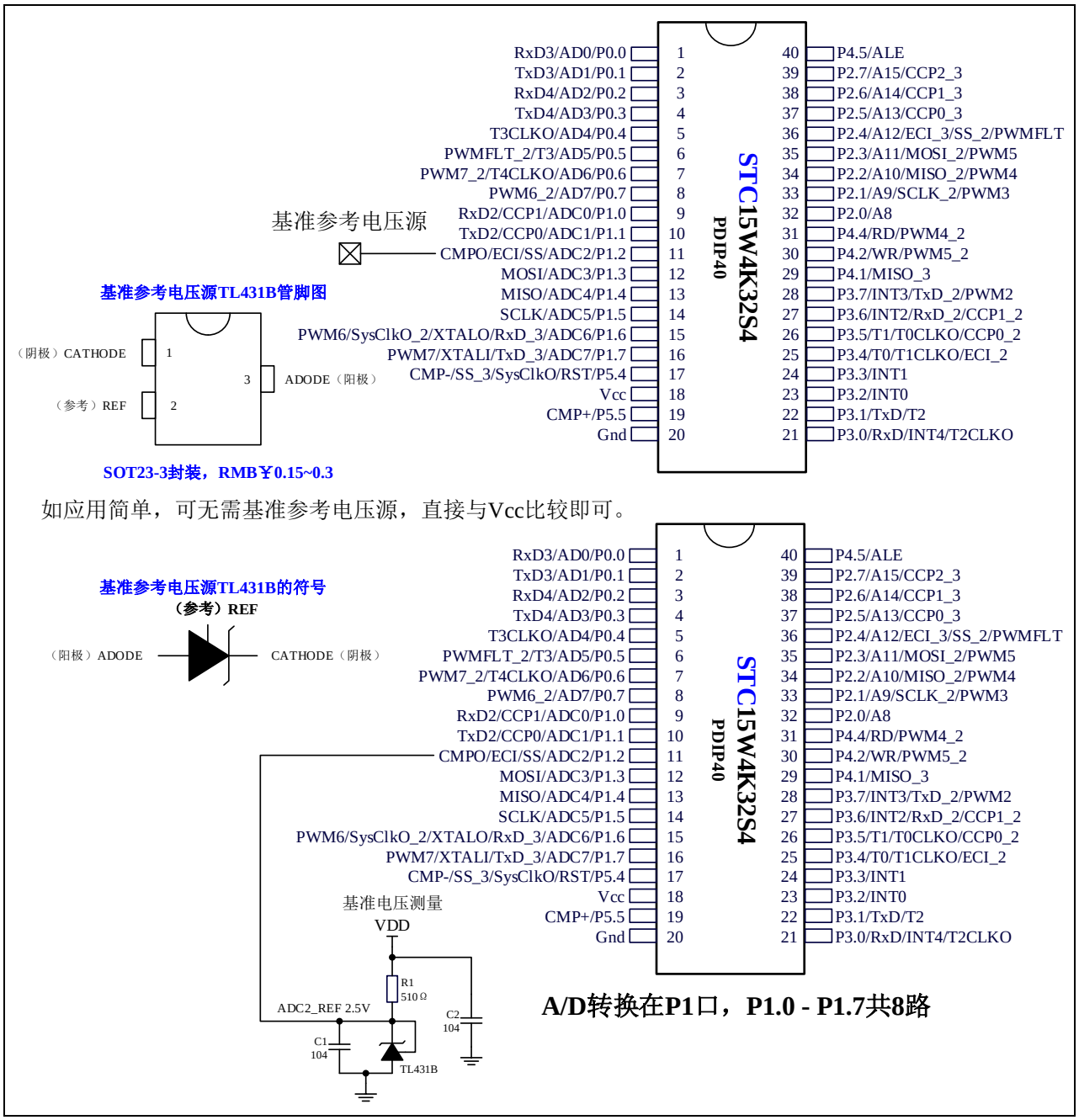
读ADC键的方法：

每隔10ms左右读一次ADC值，并且保存最后3次的读数，其变化比较小时再判断键。判断键有效时，允许一定的偏差，比如±16个字的偏差。

21.5 A/D 转换模块的参考电压源

STC15 系列单片机的参考电压源是输入工作电压 Vcc，所以一般不用外接参考电压源。如 7805 的输出电压是 5V，但实际电压可能是 4.88V 到 4.96V，用户需要精度比较高的话，可在出厂时将实际测出的工作电压值记录在单片机内部的 EEPROM 里面，以供计算。

如果有些用户的 Vcc 不固定，如电池供电，电池电压在 5.3V--4.2V 之间漂移，则 Vcc 不固定，就需要在 8 路 A/D 转换的一个通道外接一个稳定的参考电压源，来计算出此时的工作电压 Vcc，再计算出其他几路 A/D 转换通道的电压。如下图所示，可在 ADC 转换通道的第二通道外接一个 1.25V（或 1V，或...）的基准参考电压源，由此求出此时的工作电压 Vcc，再计算出其它几路 A/D 转换通道的电压（理论依据是短时间之内，Vcc 不变）。



21.6 A/D 转换的测试程序(C 和汇编)

21.6.1 A/D 转换的测试程序(ADC 中断方式)

1.C 程序:

```
/*----STC15F2K60S2 系列 A/D 转换中断方式举例-----------------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可----*/
//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"
#include "intrins.h"

#define   FOSC            18432000L
#define   BAUD            9600

typedef   unsigned char   BYTE;
typedef   unsigned int    WORD;

#define   URMD            0         //0:使用定时器 2 作为波特率发生器
                                    //1:使用定时器 1 的模式 0(16 位自动重载模式)作为波特率发生器
                                    //2:使用定时器 1 的模式 2(8 位自动重载模式)作为波特率发生器

sfr       T2H = 0xd6;               //定时器 2 高 8 位
sfr       T2L = 0xd7;               //定时器 2 低 8 位

sfr       AUXR = 0x8e;              //辅助寄存器

sfr       ADC_CONT = 0xBC;          //ADC 控制寄存器
sfr       ADC_RES = 0xBD;           //ADC 高 8 位结果
sfr       ADC_LOW2 = 0xBE;          //ADC 低 2 位结果
sfr       P1ASF = 0x9D;             //P1 口第 2 功能控制寄存器

#define   ADC_POWER       0x80      //ADC 电源控制位
#define   ADC_FLAG        0x10      //ADC 完成标志
#define   ADC_START       0x08      //ADC 起始控制位

#define   ADC_SPEEDLL     0x00      //540 个时钟
#define   ADC_SPEEDL      0x20      //360 个时钟
#define   ADC_SPEEDH      0x40      //180 个时钟
#define   ADC_SPEEDHH     0x60      //90 个时钟

void InitUart();
void SendData(BYTE dat);
void Delay(WORD n);
```

```
void InitADC();

BYTE    ch = 0;                         //ADC 通道号

void main()
{
    InitUart();                         //初始化串口
    InitADC();                          //初始化 ADC
    IE = 0xa0;                          //使能 ADC 中断
                                        //开始 AD 转换
    while (1);
}
/*----------------------------ADC 中断服务程序----------------------------*/
void adc_isr() interrupt 5 using 1
{
    ADC_CONTR &= !ADC_FLAG;             //清除 ADC 中断标志
    SendData(ch);                       //显示通道号
    SendData(ADC_RES);                  //读取高 8 位结果并发送到串口
    // SendData(ADC_LOW2);              //显示低 2 位结果
    if (++ch > 7) ch = 0;               //切换到下一个通道
    ADC_CONTR = ADC_POWER | ADC_SPEEDLL | ADC_START | ch;
}
/*----------------------------初始化 ADC----------------------------*/
void InitADC()
{
    P1ASF = 0xff;                       //设置 P1 口为 AD 口
    ADC_RES = 0;                        //清除结果寄存器
    ADC_CONTR = ADC_POWER | ADC_SPEEDLL | ADC_START | ch;
    Delay(2);                           //ADC 上电并延时
}
/*----------------------------初始化串口----------------------------*/
void InitUart()
{
    SCON = 0x5a;                        //设置串口为 8 位可变波特率
#if URMD == 0
    T2L = 0xd8;                         //设置波特率重装值
    T2H = 0xff;                         //115200 bps(65536-18432000/4/115200)
    AUXR = 0x14;                        //T2 为 1T 模式, 并启动定时器 2
    AUXR |= 0x01;                       //选择定时器 2 为串口 1 的波特率发生器
#elif URMD == 1
    AUXR = 0x40;                        //定时器 1 为 1T 模式
    TMOD = 0x00;                        //定时器 1 为模式 0(16 位自动重载)
    TL1 = 0xd8;                         //设置波特率重装值
    TH1 = 0xff;                         //115200 bps(65536-18432000/4/115200)
    TR1 = 1;                            //定时器 1 开始启动
#else
```

```
    TMOD = 0x20;                    //设置定时器 1 为 8 位自动重装载模式
    AUXR = 0x40;                    //定时器 1 为 1T 模式
    TH1 = TL1 = 0xfb;               //115200 bps(256 - 18432000/32/115200)
    TR1 = 1;
#endif
}
/*----------------------------发送串口数据----------------------------*/
void SendData(BYTE dat)
{
    while (!TI);                    //等待前一个数据发送完成
    TI = 0;                         //清除发送标志
    SBUF = dat;                     //发送当前数据
}
/*----------------------------软件延时----------------------------*/
void Delay(WORD n)
{
    WORD x;
    while (n--)
    {
        x = 5000;
        while (x--);
    }
}
```

2. 汇编程序：

```
/*----STC15F2K60S2 系列 A/D 转换中断方式举例---------------------------------------*/
/*----在 Keil C 开发环境中选择  Intel 8052 编译，头文件包含<reg51.h>即可--------*/
;本示例在 Keil 开发环境下请选择 Intel 的 8052 芯片型号进行编译
;假定测试芯片的工作频率为 18.432MHz
#define     URMD    0       ;0:使用定时器 2 作为波特率发生器
                            ;1:使用定时器 1 的模式 0(16 位自动重载模式)作为波特率发生器
                            ;2:使用定时器 1 的模式 2(8 位自动重载模式)作为波特率发生器
T2H         DATA    0D6H    ;定时器 2 高 8 位
T2L         DATA    0D7H    ;定时器 2 低 8 位

AUXR        DATA    08EH    ;辅助寄存器

ADC_CONTR   EQU     0BCH    ;ADC 控制寄存器
ADC_RES     EQU     0BDH    ;ADC 高 8 位结果
ADC_LOW2    EQU     0BEH    ;ADC 低 2 位结果

P1ASF       EQU     09DH    ;P1 口第 2 功能控制寄存器

ADC_POWER   EQU     80H     ;ADC 电源控制位
ADC_FLAG    EQU     10H     ;ADC 完成标志
```

```
ADC_START       EQU     08H             ;ADC 起始控制位
ADC_SPEEDLL     EQU     00H             ;540 个时钟
ADC_SPEEDL      EQU     20H             ;360 个时钟
ADC_SPEEDH      EQU     40H             ;180 个时钟
ADC_SPEEDHH     EQU     60H             ;90 个时钟
ADCCH           DATA    20H             ;ADC 通道号
;----------------------------------------
        ORG     0000H
        LJMP    START
        ORG     002BH
        LJMP    ADC_ISR
;----------------------------------------
        ORG     0100H
START:
        MOV     SP, #3FH
        MOV     ADCCH, #0
        LCALL   INIT_UART               ;初始化串口
        LCALL   INIT_ADC                ;初始化 ADC
        MOV     IE, #0A0H               ;使能 ADC 中断
        SJMP    $
;/*---------------------------ADC 中断服务程序---------------------------*/
ADC_ISR:
        PUSH    ACC
        PUSH    PSW
        ANL     ADC_CONTR, #NOT ADC_FLAG ;清除 ADC 中断标志
        MOV     A, ADCCH
        LCALL   SEND_DATA               ;Send channel NO.
        MOV     A, ADC_ RES             ;Get ADC high 8-bit result
        LCALL   SEND_DATA               ;Send to UART
;       MOV     A, ADC_LOW2             ;Get ADC low 2-bit result
;       LCALL   SEND_DATA               ;Send to UART
        INC     ADCCH
        MOV     A, ADCCH
        ANL     A, #07H
        MOV     ADCCH, A
        ORL     A, #ADC_POWER | ADC_SPEEDLL | ADC_START
        MOV     ADC_CONTR, A            ;AD\开始 AD 转换
        POP     PSW
        POP     ACC
        RETI
;/*---------------------------初始化 ADC---------------------------*/
INIT_ADC:
        MOV     P1ASF, #0FFH            ;设置 P1 口为 AD 口
        MOV     ADC_RES, #0             ;清除结果寄存器
        MOV     A, ADCCH
        ORL     A, #ADC_POWER | ADC_SPEEDLL | ADC_START
```

```
    MOV     ADC_CONTR, A            ;ADC 上电并延时
    MOV     A, #2
    LCALL   DELAY
    RET
;/*---------------------------初始化串口---------------------------*/
INIT_UART:
    MOV     SCON, #5AH              ;设置串口为 8 位可变波特率
#if URMD == 0
    MOV     T2L, #0D8H              ;设置波特率重装值(65536-18432000/4/115200)
    MOV     T2H, #0FFH
    MOV     AUXR, #14H              ;T2 为 1T 模式, 并启动定时器 2
    ORL     AUXR, #01H              ;选择定时器 2 为串口 1 的波特率发生器
#elif URMD == 1
    MOV     AUXR, #40H              ;定时器 1 为 1T 模式
    MOV     TMOD, #00H              ;定时器 1 为模式 0(16 位自动重载)
    MOV     TL1, #0D8H              ;设置波特率重装值(65536-18432000/4/115200)
    MOV     TH1, #0FFH
    SETB    TR1                     ;定时器 1 开始运行
#else
    MOV     TMOD, #20H              ;设置定时器 1 为 8 位自动重装载模式
    MOV     AUXR, #40H              ;定时器 1 为 1T 模式
    MOV     TL1, #0FBH              ;115200 bps(256 - 18432000/32/115200)
    MOV     TH1, #0FBH
    SETB    TR1
#endif
    RET
;/*---------------------------发送串口数据---------------------------*/
SEND_DATA:
    JNB     TI, $                   ;等待前一个数据发送完成
    CLR     TI                      ;清除发送标志
    MOV     SBUF, A                 ;发送当前数据
    RET
;/*---------------------------软件延时---------------------------*/
DELAY:
    MOV     R2, A
    CLR     A
    MOV     R0, A
    MOV     R1, A
DELAY1:
    DJNZ    R0, DELAY1
    DJNZ    R1, DELAY1
    DJNZ    R2, DELAY1
    RET
END
```

21.6.2 A/D 转换的测试程序(ADC 查询方式)

1. C 程序：

```
/*----STC15F2K60S2 系列 A/D 转换查询方式举例-------------------------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可--------*/
//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"
#include "intrins.h"

#define   FOSC            18432000L
#define   BAUD            9600

typedef   unsigned char   BYTE;
typedef   unsigned int    WORD;

#define   URMD            0         //0:使用定时器 2 作为波特率发生器
                                    //1:使用定时器 1 的模式 0(16 位自动重载模式)作为波特率发生器
                                    //2:使用定时器 1 的模式 2(8 位自动重载模式)作为波特率发生器

sfr       T2H = 0xd6;               //定时器 2 高 8 位
sfr       T2L = 0xd7;               //定时器 2 低 8 位

sfr       AUXR = 0x8e;              //辅助寄存器

sfr       ADC_CONTR = 0xBC;         //ADC 控制寄存器
sfr       ADC_RES = 0xBD;           //ADC 高 8 位结果
sfr       ADC_LOW2 = 0xBE;          //ADC 低 2 位结果
sfr       P1ASF = 0x9D;             //P1 口第 2 功能控制寄存器

#define   ADC_POWER       0x80      //ADC 电源控制位
#define   ADC_FLAG        0x10      //ADC 完成标志
#define   ADC_START       0x08      //ADC 起始控制位

#define   ADC_SPEEDLL     0x00      //540 个时钟
#define   ADC_SPEEDL      0x20      //360 个时钟
#define   ADC_SPEEDH      0x40      //180 个时钟
#define   ADC_SPEEDHH     0x60      //90 个时钟

void InitUart();
void InitADC();
void SendData(BYTE dat);
BYTE GetADCResult(BYTE ch);
void Delay(WORD n);
void ShowResult(BYTE ch);
```

```
void main()
{
    InitUart();                         //初始化串口
    InitADC();                          //初始化 ADC
    while (1)
    {
        ShowResult(0);                  //显示通道 0
        ShowResult(1);                  //显示通道 1
        ShowResult(2);                  //显示通道 2
        ShowResult(3);                  //显示通道 3
        ShowResult(4);                  //显示通道 4
        ShowResult(5);                  //显示通道 5
        ShowResult(6);                  //显示通道 6
        ShowResult(7);                  //显示通道 7
    }
}
/*----------------------------发送 ADC 结果到 PC----------------------------*/
void ShowResult(BYTE ch)
{
    SendData(ch);                       //显示通道号
    SendData(GetADCResult(ch));         //显示 ADC 高 8 位结果
//  SendData(ADC_LOW2);                 //显示低 2 位结果
}
/*----------------------------读取 ADC 结果----------------------------*/
BYTE GetADCResult(BYTE ch)
{
    ADC_CONTR = ADC_POWER | ADC_SPEEDLL | ch | ADC_START;
    _nop_();                            //等待 4 个 NOP
    _nop_();
    _nop_();
    _nop_();
    while (!(ADC_CONTR & ADC_FLAG));    //等待 ADC 转换完成
    ADC_CONTR &= ~ADC_FLAG;             //Close ADC
    return ADC_RES;                     //返回 ADC 结果
}
/*----------------------------初始化串口----------------------------*/
void InitUart()
{
    SCON = 0x5a;                        //设置串口为 8 位可变波特率
#if URMD == 0
    T2L = 0xd8;                         //设置波特率重装值
    T2H = 0xff;                         //115200 bps(65536-18432000/4/115200)
    AUXR = 0x14;                        //T2 为 1T 模式, 并启动定时器 2
    AUXR |= 0x01;                       //选择定时器 2 为串口 1 的波特率发生器
#elif URMD == 1
    AUXR = 0x40;                        //定时器 1 为 1T 模式
```

```
    TMOD = 0x00;                        //定时器 1 为模式 0(16 位自动重载)
    TL1 = 0xd8;                         //设置波特率重装值
    TH1 = 0xff;                         //115200 bps(65536-18432000/4/115200)
    TR1 = 1;                            //定时器 1 开始启动
#else
    TMOD = 0x20;                        //设置定时器 1 为 8 位自动重装载模式
    AUXR = 0x40;                        //定时器 1 为 1T 模式
    TH1 = TL1 = 0xfb;                   //115200 bps(256 - 18432000/32/115200)
    TR1 = 1;
#endif
}
/*----------------------------初始化 ADC----------------------------*/
void InitADC()
{
    P1ASF = 0xff;                       //设置 P1 口为 AD 口
    ADC_RES = 0;                        //清除结果寄存器
    ADC_CONTR = ADC_POWER | ADC_SPEEDLL;
    Delay(2);                           //ADC 上电并延时
}
/*----------------------------发送串口数据----------------------------*/
void SendData(BYTE dat)
{
    while (!TI);                        //等待前一个数据发送完成
    TI = 0;                             //清除发送标志
    SBUF = dat;                         //发送当前数据
}
/*----------------------------软件延时----------------------------*/
void Delay(WORD n)
{
    WORD x;
    while (n--)
    {
        x = 5000;
        while (x--);
    }
}
```

2. 汇编程序：

```
/*----STC15F2K60S2 系列 A/D 转换查询方式举例------------------------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可----------*/
;假定测试芯片的工作频率为 18.432MHz
#define         URMD    0           ;0:使用定时器 2 作为波特率发生器
                                    ;1:使用定时器 1 的模式 0(16 位自动重载模式)作为波特率发生器
                                    ;2:使用定时器 1 的模式 2(8 位自动重载模式)作为波特率发生器
```

```
T2H             DATA    0D6H        ;定时器 2 高 8 位
T2L             DATA    0D7H        ;定时器 2 低 8 位

AUXR            DATA    08EH        ;辅助寄存器

ADC_CONTR       EQU     0BCH        ;ADC 控制寄存器
ADC_RES         EQU     0BDH        ;ADC 高 8 位结果
ADC_LOW2        EQU     0BEH        ;ADC 低 2 位结果

P1ASF           EQU     09DH        ;P1 口第 2 功能控制寄存器

ADC_POWER       EQU     80H         ;ADC 电源控制位
ADC_FLAG        EQU     10H         ;ADC 完成标志
ADC_START       EQU     08H         ;ADC 起始控制位

ADC_SPEEDLL     EQU     00H         ;540 个时钟
ADC_SPEEDL      EQU     20H         ;360 个时钟
ADC_SPEEDH      EQU     40H         ;180 个时钟
ADC_SPEEDHH     EQU     60H         ;90 个时钟
;----------------------------------------
    ORG     0000H
    LJMP    START
;----------------------------------------
    ORG     0100H
START:
    LCALL   INIT_UART               ;初始化串口
    LCALL   INIT_ADC                ;初始化 ADC
;------------------------------
NEXT:
    MOV     A, #0
    LCALL   SHOW_RESULT             ;显示通道 0 的结果
    MOV     A, #1
    LCALL   SHOW_RESULT             ;显示通道 1 的结果
    MOV     A, #2
    LCALL   SHOW_RESULT             ;显示通道 2 的结果
    MOV     A, #3
    LCALL   SHOW_RESULT             ;显示通道 3 的结果
    MOV     A, #4
    LCALL   SHOW_RESULT             ;显示通道 4 的结果
    MOV     A, #5
    LCALL   SHOW_RESULT             ;显示通道 5 的结果
    MOV     A, #6
    LCALL   SHOW_RESULT             ;显示通道 6 的结果
    MOV     A, #7
    LCALL   SHOW_RESULT             ;显示通道 7 的结果
    SJMP    NEXT
```

```
;/*----------------------------发送 ADC 结果到 PC----------------------------*/
SHOW_RESULT:
    LCALL   SEND_DATA               ;显示通道号
    LCALL   GET_ADC_RESULT          ;读取高 8 位结果
    LCALL   SEND_DATA               ;显示结果
;   MOV     A, ADC_LOW2             ;读取低 2 位结果
;   LCALL   SEND_DATA               ;显示结果
    RET
;/*----------------------------读取 ADC 结果----------------------------*/
GET_ADC_RESULT:
    ORL     A, #ADC_POWER | ADC_SPEEDLL | ADC_START
    MOV     ADC_CONTR, A            ;开始 AD 转换
    NOP                             ;等待 4 个 NOP
    NOP
    NOP
    NOP
WAIT:
    MOV     A, ADC_CONTR            ;等待 ADC 转换完成
    JNB     ACC.4, WAIT             ;ADC_FLAG(ADC_CONTR.4)
    ANL     ADC_CONTR, #NOT ADC_FLAG            ;清 ADC 标志
    MOV     A, ADC_RES              ;返回 ADC 结果
    RET
;/*----------------------------初始化 ADC----------------------------*/
INIT_ADC:
    MOV     P1ASF, #0FFH            ;设置 P1 口为 AD 口
    MOV     ADC_RES, #0             ;清除结果寄存器
    MOV     ADC_CONTR, #ADC_POWER | ADC_SPEEDLL
    MOV     A, #2                   ;ADC 上电并延时
    LCALL   DELAY
    RET
;/*----------------------------初始化串口----------------------------*/
INIT_UART:
    MOV     SCON, #5AH              ;设置串口为 8 位可变波特率
#if URMD == 0
    MOV     T2L, #0D8H              ;设置波特率重装值(65536-18432000/4/115200)
    MOV     T2H, #0FFH
    MOV     AUXR, #14H              ;T2 为 1T 模式, 并启动定时器 2
    ORL     AUXR, #01H              ;选择定时器 2 为串口 1 的波特率发生器
#elif URMD == 1
    MOV     AUXR, #40H              ;定时器 1 为 1T 模式
    MOV     TMOD, #00H              ;定时器 1 为模式 0(16 位自动重载)
    MOV     TL1, #0D8H              ;设置波特率重装值(65536-18432000/4/115200)
    MOV     TH1, #0FFH
    SETB    TR1                     ;定时器 1 开始运行
#else
    MOV     TMOD, #20H              ;设置定时器 1 为 8 位自动重装载模式
```

```
    MOV     AUXR, #40H              ;定时器 1 为 1T 模式
    MOV     TL1, #0FBH              ;115200 bps(256 - 18432000/32/115200)
    MOV     TH1, #0FBH
    SETB    TR1
#endif
    RET
;/*----------------------------发送串口数据----------------------------*/
SEND_DATA:
    JNB     TI, $                   ;等待前一个数据发送完成
    CLR     TI                      ;清除发送标志
    MOV     SBUF, A                 ;发送当前数据
    RET
;/*----------------------------软件延时----------------------------*/
DELAY:
    MOV     R2, A
    CLR     A
    MOV     R0, A
    MOV     R1, A
DELAY1:
    DJNZ    R0, DELAY1
    DJNZ    R1, DELAY1
    DJNZ    R2, DELAY1
    RET
END
```

21.7 利用新增的 ADC 第 9 通道测量内部参考电压的测试程序

----所测量的内部参考电压 BandGap 电压用来计算工作电压 Vcc

ADC 的第 9 通道是用来测试内部 BandGap 参考电压的，由于内部 BandGap 参考电压很稳定，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 BandGap 参考电压，然后通过 ADC 的值便可反推出 VCC 的电压，从而用户可以实现自己的低压检测功能。

ADC 的第 9 通道的测量方法：首先将 P1ASF 初始化为 0，即关闭所有 P1 口的模拟功能然后通过正常的 ADC 转换的方法读取第 0 通道的值，即可通过 ADC 的第 9 通道读取当前内部 BandGap 参考电压值。

用户实现自己的低压检测功能的实现方法：首先用户需要在 VCC 很精准的情况下(比如 5.0V)，测量出内部 BandGap 参考电压的 ADC 转换值(比如为 BGV5)，并将这个值保存到 EEPROM 中，然后在低压检测的代码中，在实际 VCC 变化后，测量出的内部 BandGap 参考电压的 ADC 转换值(比如为 BGVx)，最后通过计算公式：实际 VCC=5.0V×BGV5/BGVx，即可计算出实际的 VCC 电压值，需要注意的是，第一步的 BGV5 的基准测量一定要精确。

1.C 程序:

```
/*----STC15W4K60S4 系列 ADC 第 9 通道应用举例----*/
/*----本示例在 Keil 开发环境下请选择 Intel 的 8058 芯片型号进行编译----*/
```

```
//假定测试芯片的工作频率为 18.432MHz
//说明:
//ADC 的第 9 通道是用来测试内部 BandGap 参考电压的，由于内部 BandGap 参考电压很稳定，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 BandGap 参考电压，然后通过 ADC 的值便可反推
//出 VCC 的电压，从而用户可以实现自己的低压检测功能。
//ADC 的第 9 通道的测量方法：首先将 P1ASF 初始化为 0，即关闭所有 P1 口的模拟功能然后通过正常
//的 ADC 转换的方法读取第 0 通道的值,即可通过 ADC 的第 9 通道读取当前内部 BandGap 参考电压值。
//用户实现自己的低压检测功能的实现方法：首先用户需要在 VCC 很精准的情况下(比如 5.0V)，测量
//出内部 BandGap 参考电压的 ADC 转换值(比如为 BGV5)，并将这个值保存到 EEPROM 中，然后在低压
//检测的代码中，在实际 VCC 变化后，测量出的内部 BandGap 参考电压的 ADC 转换值(比如为 BGVx)，
//最后通过计算公式：实际 VCC=5.0V×BGV5/BGVx，即可计算出实际的 VCC 电压值/需要注意的是，
//第一步的 BGV5 的基准测量一定要精确.
#include "reg51.h"
#include "intrins.h"

#define   FOSC            18432000L
#define   BAUD            115200

typedef   unsigned char   BYTE;
typedef   unsigned int    WORD;

#define   URMD            0         //0:使用定时器 2 作为波特率发生器
                                    //1:使用定时器 1 的模式 0(16 位自动重载模式)作为波特率发生器
                                    //2:使用定时器 1 的模式 2(8 位自动重载模式)作为波特率发生器

sfr       T2H = 0xd6;               //定时器 2 高 8 位
sfr       T2L = 0xd7;               //定时器 2 低 8 位
sfr       AUXR = 0x8e;              //辅助寄存器
sfr       ADC_CONTR = 0xBC;         //ADC 控制寄存器
sfr       ADC_RES = 0xBD;           //ADC 高 8 位结果
sfr       ADC_LOW2 = 0xBE;          //ADC 低 2 位结果
sfr       P1ASF = 0x9D;             //P1 口第 2 功能控制寄存器

#define   ADC_POWER       0x80      //ADC 电源控制位
#define   ADC_FLAG        0x10      //ADC 完成标志
#define   ADC_START       0x08      //ADC 起始控制位
#define   ADC_SPEEDLL     0x00      //540 个时钟
#define   ADC_SPEEDL      0x20      //360 个时钟
#define   ADC_SPEEDH      0x40      //180 个时钟
#define   ADC_SPEEDHH     0x60      //90 个时钟

void InitUart();
void InitADC();
void SendData(BYTE dat);
BYTE GetADCResult();
void Delay(WORD n);
```

```
void ShowResult();

void main()
{
    InitUart();                         //初始化串口
    InitADC();                          //初始化 ADC
    while (1)
    {
        ShowResult();                   //显示 ADC 结果
    }
}
/*----------------------------发送 ADC 结果到 PC----------------------------*/
void ShowResult()
{
    SendData(GetADCResult());           //显示 ADC 高 8 位结果
//  SendData(ADC_LOW2);                 //显示低 2 位结果
}
/*----------------------------读取 ADC 结果----------------------------*/
BYTE GetADCResult()
{
    ADC_CONTR = ADC_POWER | ADC_SPEEDLL | 0 | ADC_START;
    _nop_();                            //等待 4 个 NOP
    _nop_();
    _nop_();
    _nop_();
    while (!(ADC_CONTR & ADC_FLAG));    //等待 ADC 转换完成
    ADC_CONTR &= ~ADC_FLAG;             //Close ADC
    P2 = ADC_RES;
    return ADC_RES;                     //返回 ADC 结果
}
/*----------------------------初始化串口----------------------------*/
void InitUart()
{
    SCON =  0x5a;                       //设置串口为 8 位可变波特率
#if URMD == 0
    T2L = 0xd8;                         //设置波特率重装值
    T2H = 0xff;                         //115200 bps(65536-18432000/4/115200)
    AUXR = 0x14;                        //T2 为 1T 模式, 并启动定时器 2
    AUXR |= 0x01;                       //选择定时器 2 为串口 1 的波特率发生器
#elif URMD == 1
    AUXR = 0x40;                        //定时器 1 为 1T 模式
    TMOD = 0x00;                        //定时器 1 为模式 0(16 位自动重载)
    TL1 = 0xd8;                         //设置波特率重装值
    TH1 = 0xff;                         //115200 bps(65536-18432000/4/115200)
    TR1 = 1;                            //定时器 1 开始启动
#else
```

```
    TMOD = 0x20;                        //设置定时器 1 为 8 位自动重装载模式
    AUXR = 0x40;                        //定时器 1 为 1T 模式
    TH1 = TL1 =  0xfb;                  //115200 bps(256 - 18432000/32/115200)
    TR1 = 1;
#endif
}
/*----------------------------初始化 ADC----------------------------*/
void InitADC()
{
    P1ASF = 0x00;                       //不设置 P1 口为模拟口
    ADC_RES = 0;                        //清除结果寄存器
    ADC_CONTR = ADC_POWER | ADC_SPEEDLL;
    Delay(2);                           //ADC 上电并延时
}
/*----------------------------发送串口数据----------------------------*/
void SendData(BYTE dat)
{
    while (!TI);                        //等待前一个数据发送完成
    TI = 0;                             //清除发送标志
    SBUF = dat;                         //发送当前数据
}
/*----------------------------软件延时----------------------------*/
void Delay(WORD n)
{
    WORD x;
    while (n--)
    {
        x = 5000;
        while (x--);
    }
}
```

2.汇编程序:

```
/*----STC15W4K60S4 系列 ADC 第 9 通道应用举例----------------------------*/
/*----本示例在 Keil 开发环境下请选择 Intel 的 8058 芯片型号进行编译----*/
//假定测试芯片的工作频率为 18.432MHz
//说明:
//ADC 的第 9 通道是用来测试内部 BandGap 参考电压的，由于内部 BandGap 参考电压很稳定，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 BandGap 参考电压，然后通过 ADC 的值便可反推
//出 VCC 的电压，从而用户可以实现自己的低压检测功能。
//ADC 的第 9 通道的测量方法：首先将 P1ASF 初始化为 0，即关闭所有 P1 口的模拟功能然后通过正常
//的 ADC 转换的方法读取第 0 通道的值，即可通过 ADC 的第 9 通道读取当前内部 BandGap 参考电压值。
//用户实现自己的低压检测功能的实现方法：首先用户需要在 VCC 很精准的情况下(比如 5.0V)，测量
//出内部 BandGap 参考电压的 ADC 转换值(比如为 BGV5)，并将这个值保存到 EEPROM 中，然后在低压
//检测的代码中，在实际 VCC 变化后，测量出的内部 BandGap 参考电压的 ADC 转换值(比如为 BGVx)，
//最后通过计算公式：实际 VCC=5.0V×BGV5/BGVx，即可计算出实际的 VCC 电压值/需要注意的是，
```

```
//第一步的 BGV5 的基准测量一定要精确.
#define         URMD    0       //0:使用定时器 2 作为波特率发生器
                                //1:使用定时器 1 的模式 0(16 位自动重载模式)作为波特率发生器
                                //2:使用定时器 1 的模式 2(8 位自动重载模式)作为波特率发生器

T2H             DATA    0D6H            ;定时器 2 高 8 位
T2L             DATA    0D7H            ;定时器 2 低 8 位

AUXR            DATA    08EH            ;辅助寄存器

ADC_CONTR       EQU     0BCH            ;ADC 控制寄存器
ADC_RES         EQU     0BDH            ;ADC 高 8 位结果
ADC_LOW2        EQU     0BEH            ;ADC 低 2 位结果

P1ASF           EQU     09DH            ;P1 口第 2 功能控制寄存器

ADC_POWER       EQU     80H             ;ADC 电源控制位
ADC_FLAG        EQU     10H             ;ADC 完成标志
ADC_START       EQU     08H             ;ADC 起始控制位
ADC_SPEEDLL     EQU     00H             ;540 个时钟
ADC_SPEEDL      EQU     20H             ;360 个时钟
ADC_SPEEDH      EQU     40H             ;180 个时钟
ADC_SPEEDHH     EQU     60H             ;90 个时钟
;----------------------------------------
    ORG     0000H
    LJMP    START
;----------------------------------------
    ORG     0100H
START:
    LCALL   INIT_UART               ;初始化串口
    LCALL   INIT_ADC                ;初始化 ADC
;------------------------------
NEXT:
    LCALL   SHOW_RESULT             ;显示通道 0 的结果
    SJMP    NEXT
;/*--------------------------发送 ADC 结果到 PC--------------------------*/
SHOW_RESULT:
    LCALL   GET_ADC_RESULT          ;读取高 8 位结果
    LCALL   SEND_DATA               ;显示结果
;   MOV     A, ADC_LOW2             ;读取低 2 位结果
;   LCALL   SEND_DATA               ;显示结果
    RET
;/*--------------------------读取 ADC 结果--------------------------*/
GET_ADC_RESULT:
    MOV     A, #ADC_POWER | ADC_SPEEDLL | 0 | ADC_START
    MOV     ADC_CONTR, A            ;开始 AD 转换
```

```
    NOP                                     ;等待 4 个 NOP
    NOP
    NOP
    NOP
WAIT:
    MOV     A, ADC_CONTR                    ;等待 ADC 转换完成
    JNB     ACC.4, WAIT                     ;ADC_FLAG(ADC_CONTR.4)
    ANL     ADC_CONTR, #NOT ADC_FLAG        ;清 ADC 标志
    MOV     A, ADC_RES                      ;返回 ADC 结果
    RET
;/*----------------------------初始化 ADC----------------------------*/
INIT_ADC:
    MOV     P1ASF, #00H                     ;不设置 P1 口为模拟口
    MOV     ADC_RES, #0                     ;清除结果寄存器
    MOV     ADC_CONTR, #ADC_POWER | ADC_SPEEDLL
    MOV     A, #2                           ;ADC 上电并延时
    LCALL   DELAY
    RET
;/*----------------------------初始化串口----------------------------*/
INIT_UART:
    MOV     SCON, #5AH                      ;设置串口为 8 位可变波特率
#if URMD == 0
    MOV     T2L, #0D8H                      ;设置波特率重装值(65536-18432000/4/115200)
    MOV     T2H, #0FFH
    MOV     AUXR, #14H                      ;T2 为 1T 模式, 并启动定时器 2
    ORL     AUXR, #01H                      ;选择定时器 2 为串口 1 的波特率发生器
#elif URMD == 1
    MOV     AUXR, #40H                      ;定时器 1 为 1T 模式
    MOV     TMOD, #00H                      ;定时器 1 为模式 0(16 位自动重载)
    MOV     TL1, #0D8H                      ;设置波特率重装值(65536-18432000/4/115200)
    MOV     TH1, #0FFH
    SETB    TR1                             ;定时器 1 开始运行
#else
    MOV     TMOD, #20H                      ;设置定时器 1 为 8 位自动重装载模式
    MOV     AUXR, #40H                      ;定时器 1 为 1T 模式
    MOV     TL1, #0FBH                      ;115200 bps(256 - 18432000/32/115200)
    MOV     TH1, #0FBH
    SETB    TR1
#endif
    RET

;/*----------------------------发送串口数据----------------------------*/
SEND_DATA:
    JNB     TI, $                           ;等待前一个数据发送完成
    CLR     TI                              ;清除发送标志
    MOV     SBUF, A                         ;发送当前数据
```

```
    RET
;/*----------------------------软件延时----------------------------*/
DELAY:
    MOV     R2, A
    CLR     A
    MOV     R0, A
    MOV     R1, A
DELAY1:
    DJNZ    R0, DELAY1
    DJNZ    R1, DELAY1
    DJNZ    R2, DELAY1
    RET
END
```

21.8 利用新增的 ADC 第 9 通道测量外部电压或外部电池电压

----利用内部参考电压 BandGap 电压测量

ADC 的第 9 通道是用来测试内部 BandGap 参考电压的，由于内部 BandGap 参考电压很稳定，约为 1.27V，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 BandGap 参考电压，然后通过 ADC 的值便可反推出外部电压或外部电池电压，从而用户可以测量外部电压或外部电池电压。

ADC 的第 9 通道的测量方法：首先将 P1ASF 初始化为 0，即关闭所有 P1 口的模拟功能然后通过正常的 ADC 转换的方法读取第 0 通道的值，即可迪过 ADC 的第 9 迪道读取当前内部 BandGap 参考电压值，约为 1.27V。

测量外部电压或外部电池电压的方法：首先用户需要在外部电压或外部电池电压很精准的情况下(比如 5.0V)，测量出内部 BandGap 参考电压的 ADC 转换值(比如为 BGV5)，并将这个值保存到 EEPROM 中，然后在实际的外部电压或外部电池电压变化后，测量出的内部 BandGap 参考电压的 ADC 转换值(比如为 BGVx)，最后通过计算公式：实际外部电压或外部电池电压=5.0V×BGV5/BGVx，即可计算出实际的外部电压或外部电池电压值，需要注意的是，第一步的 BGV5 的基准测量一定要精确。

21.9 利用外部 TL431 基准测量外部输入电压值的测试程序

1、C 语言程序

```
/*----STC 1T Series MCU 利用 TL431 基准测量外部输入电压值的 Demo Programme-----*/
/*-------------------本程序功能说明--------------------------
读 ADC 测量外部电压，使用外部 TL431 基准计算电压
用 STC 的 MCU 的 IO 方式控制 74HC595 驱动 8 位数码管。
用户可以修改宏来选择时钟频率.
用户可以在"用户定义宏"中选择共阴或共阳，推荐尽量使用共阴数码管
使用 Timer0 的 16 位自动重装来产生 1ms 节拍，程序运行于这个节拍下，用户修改 MCU 主时钟频率时，
自动定时于 1ms.
右边 4 位数码管显示测量的电压值值
```

```
外部电压从板上测温电阻两端输入，输入电压 0 ~ VDD，不要超过 VDD 或低于 0V.
实际项目使用请串一个 1K 的电阻到 ADC 输入口，ADC 输入口再并一个电容到地
-----------------------------------------------------------------*/
#include "config.H"
#include "adc.h"
#define     P1n_pure_input(bitn)    P1M1 |= (bitn), P1M0 &= ~(bitn)
/****************************** 用户定义宏 ***********************************/
#define     Cal_MODE        0           //每次测量只读 1 次 ADC. 分辨率 0.01V
// #define     Cal_MODE        1           //每次测量连续读 16 次 ADC 再平均计算. 分辨率 0.01V
#define     LED_TYPE        0x00        //定义 LED 类型, 0x00--共阴, 0xff--共阳
#define     Timer0_Reload   (65536UL -(MAIN_Fosc / 1000))       //Timer 0 中断频率, 1000 次/秒
/*****************************************************************************/
/************* 本地常量声明 **************/
u8 code t_display[]={                   //标准字库
// 0    1     2     3    4     5     6     7     8    9    A    B     C    D    E     F
0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F, 0x77, 0x7C, 0x39, 0x5E, 0x79, 0x71,
//  black   -    H    J    K     L    N    o    P     U    t    G     Q    r    M    y
    0x00, 0x40, 0x76, 0x1E, 0x70, 0x38, 0x37, 0x5C, 0x73, 0x3E, 0x78, 0x3d, 0x67, 0x50, 0x37, 0x6e,
    0xBF, 0x86, 0xDB, 0xCF, 0xE6, 0xED, 0xFD, 0x87, 0xFF, 0xEF, 0x46};  //0. 1. 2. 3. 4. 5. 6. 7. 8. 9. -1
u8 code T_COM[]={0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};    //位码

/************* I/O 口定义 **************/
sbit    P_HC595_SER = P4^0;             //pin 14 SER data input
sbit    P_HC595_RCLK = P5^4;            //pin 12 RCLk store (latch) clock
sbit    P_HC595_SRCLK = P4^3;           //pin 11 SRCLK Shift data clock
/*************    本地变量声明    **************/
u8      LED8[8];                        //显示缓冲
u8      display_index;                  //显示位索引
bit     B_1ms;                          //1ms 标志
u16     msecond;
u16     Bandgap;
/*************本地函数声明**************/
u16     get_temperature(u16 adc);

/**************** 外部函数声明和外部变量声明 *****************/
/************************ ADC 配置函数 ************************/
void ADC_config(void)
{
    ADC_InitTypeDef ADC_InitStructure;              //结构定义
    ADC_InitStructure.ADC_Px = ADC_P12 | ADC_P13;
                                    //设置要做 ADC 的 IO, ADC_P10 ~ ADC_P17(或操作), ADC_P1_All
    ADC_InitStructure.ADC_Speed = ADC_90T;
                                    //ADC 速度: ADC_90T, ADC_180T, ADC_360T, ADC_540T
    ADC_InitStructure.ADC_Power = ENABLE;
                                    //ADC 功率允许/关闭: ENABLE, DISABLE
    ADC_InitStructure.ADC_AdjResult = ADC_RES_H8L2;
```

```
                                    //ADC 结果调整, ADC_RES_H2L8, ADC_RES_H8L2
    ADC_InitStructure.ADC_Polity = PolityLow;           //优先级设置: PolityHigh, PolityLow
    ADC_InitStructure.ADC_Interrupt = DISABLE;          //中断允许: ENABLE, DISABLE
    ADC_Inilize(&ADC_InitStructure);                    //初始化
    ADC_PowerControl(ENABLE);                           //单独的 ADC 电源操作函数, ENABLE 或 DISABLE
    P1n_pure_input((1<<2) | (1<<3));                    //把 ADC 口设置为高阻输入
}
/*********************************************/
void main(void)
{
    u8 i;
    u16 j;
    display_index = 0;
    ADC_config();

    Timer0_1T();
    Timer0_AsTimer();
    Timer0_16bitAutoReload();
    Timer0_Load(Timer0_Reload);
    Timer0_InterruptEnable();
    Timer0_Run();
    EA = 1;                                             //打开总中断

    for(i=0; i<8; i++)    LED8[i] = 0x10;               //上电消隐
    while(1)
    {
        if(B_1ms)                                       //1ms 到
        {
            B_1ms = 0;
            if(++msecond >= 300)                        //300ms 到
            {
                msecond = 0;
                #if (Cal_MODE == 0)
                    //================  只读 1 次 ADC, 10bit ADC.  分辨率 0.01V =======
                    Get_ADC10bitResult(2);
                    //通道改变, 先读一次并丢弃结果, 让内部的采样电容的电压等于输入值.
                    Bandgap = Get_ADC10bitResult(2);    //读外部基准 TL431 对应的 ADC
                    Get_ADC10bitResult(3);
                    //通道改变, 先读一次并丢弃结果, 让内部的采样电容的电压等于输入值.
                    j = Get_ADC10bitResult(3);          //读外部电压 ADC
                    j = (u16)((u32)j * 250 / Bandgap);
                    //计算外部电压, TL431 电压为 2.50V, 测电压分辨率 0.01V
                #endif
                //================================================
                //=====  连续读 16 次 ADC  再平均计算.  分辨率 0.01V ========
                #if (Cal_MODE == 1)
```

```
                    Get_ADC10bitResult(2);
                        //通道改变, 先读一次并丢弃结果, 让内部的采样电容的电压等于输入值.
                    for(j=0, i=0; i<16; i++)
                    {
                        j += Get_ADC10bitResult(2);       //读外部基准 TL431 对应的 ADC
                    }
                    Bandgap = j >> 4; //16 次平均
                    Get_ADC10bitResult(3);
                        //通道改变, 先读一次并丢弃结果, 让内部的采样电容的电压等于输入值.
                    for(j=0, i=0; i<16; i++)
                    {
                        j += Get_ADC10bitResult(3);       //读外部电压 ADC
                    }
                    j = j >> 4;                           //16 次平均
                    j = (u16)((u32)j * 250 / Bandgap);
                        //计算外部电压, TL431 电压为 2.50V, 测电压分辨率 0.01V
                #endif
                //=====================================================
                    LED8[5] = j / 100 + DIS_DOT;    //显示外部电压值
                    LED8[6] = (j % 100) / 10;
                    LED8[7] = j % 10;
                //  j = Bandgap;
                //  LED8[0] = j / 1000; //显示 Bandgap ADC 值
                //  LED8[1] = (j % 1000) / 100;
                //  LED8[2] = (j % 100) / 10;
                //  LED8[3] = j % 10;
            }
        }
    }
}
/***************** 向 HC595 发送一个字节函数 ******************/
void Send_595(u8 dat)
{
    u8 i;
    for(i=0; i<8; i++)
    {
        dat <<= 1;
        P_HC595_SER = CY;
        P_HC595_SRCLK = 1;
        P_HC595_SRCLK = 0;
    }
}
/************************ 显示扫描函数 ************************/
void DisplayScan(void)
{
    Send_595(~LED_TYPE ^ T_COM[display_index]);              //输出位码
```

```
    Send_595( LED_TYPE ^ t_display[LED8[display_index]]);      //输出段码
    P_HC595_RCLK = 1;
    P_HC595_RCLK = 0;                                          //锁存输出数据
    if(++display_index >= 8)     display_index = 0;            //8 位结束回 0
}
/********************** Timer0 1ms 中断函数  ************************/
void timer0 (void) interrupt TIMER0_VECTOR
{
    DisplayScan();                                             //1ms 扫描显示一位
    B_1ms = 1;                                                 //1ms 标志
}
```

2 汇编程序

```
/*----STC 1T Series MCU 利用 TL431 基准测量外部输入电压值的 Demo Programme ----*/
/*************  本程序功能说明  **************
读 ADC 测量外部电压，使用外部 TL431 基准计算电压.
用 STC 的 MCU 的 IO 方式控制 74HC595 驱动 8 位数码管。
;用户可以修改宏来选择时钟频率.
;用户可以在"用户定义宏"中选择共阴或共阳. 推荐尽量使用共阴数码管.
;使用 Timer0 的 16 位自动重装来产生 1ms 节拍，程序运行于这个节拍下，用户修改 MCU 主时钟频率时，
自动定时于 1ms.
;右边 4 位数码管显示测量的电压值值.
;外部电压从板上测温电阻两端输入，输入电压 0 ~ VDD，不要超过 VDD 或低于 0V.
;实际项目使用请串一个 1K 的电阻到 ADC 输入口，ADC 输入口再并一个电容到地.
;*****************************************/
;/*****************************  用户定义宏  ***********************************/
Fosc_KHZ        EQU     22118       ;22118KHz

STACK_POIRTER   EQU     0D0H        ;堆栈开始地址

LED_TYPE        EQU     000H        ;定义 LED 类型, 000H --  共阴, 0FFH --  共阳

Timer0_Reload   EQU     (65536 - Fosc_KHZ)     ; Timer 0  中断频率, 1000 次/秒

DIS_DOT         EQU     020H
DIS_BLACK       EQU     010H
DIS_            EQU     011H
;*************************************************************************
ADC_P10         EQU     0x01        ;I/O 引脚  Px.0
ADC_P11         EQU     0x02        ;I/O 引脚  Px.1
ADC_P12         EQU     0x04        ;I/O 引脚  Px.2
ADC_P13         EQU     0x08        ;I/O 引脚  Px.3
ADC_P14         EQU     0x10        ;I/O 引脚  Px.4
ADC_P15         EQU     0x20        ;I/O 引脚  Px.5
ADC_P16         EQU     0x40        ;I/O 引脚  Px.6
```

```
ADC_P17         EQU     0x80        ;I/O 引脚  Px.7
ADC_P1_All      EQU     0xFF        ;I/O 所有引脚

ADC_PowerOn     EQU     (1 SHL 7)
ADC_90T         EQU     (3 SHL 5)
ADC_180T        EQU     (2 SHL 5)
ADC_360T        EQU     (1 SHL 5)
ADC_540T        EQU     0
ADC_FLAG        EQU     (1 SHL 4)   ;软件清 0
ADC_START       EQU     (1 SHL 3)   ;自动清 0
ADC_CH0         EQU     0
ADC_CH1         EQU     1
ADC_CH2         EQU     2
ADC_CH3         EQU     3
ADC_CH4         EQU     4
ADC_CH5         EQU     5
ADC_CH6         EQU     6
ADC_CH7         EQU     7

ADC_RES_H2L8    EQU     (1 SHL 5)
ADC_RES_H8L2    EQU     NOT (1 SHL 5)

;*******************************************************************
AUXR            DATA    08EH
P4              DATA    0C0H
P5              DATA    0C8H
ADC_CONTR       DATA    0BCH        ;带 AD 系列
ADC_RES         DATA    0BDH        ;带 AD 系列

ADC_RESL        DATA    0BEH        ;带 AD 系列
P1ASF           DATA    09DH
PCON2           DATA    097H
P1M1            DATA    091H        ; P1M1.n,P1M0.n =00--->Standard, 01--->push-pull
                                    ;实际上 1T 的都一样
P1M0            DATA    092H        ; =10--->pure input, 11--->open drain
;************* I/O 口定义  ***************/
P_HC595_SER     BIT     P4.0        ; //pin 14 SER data input
P_HC595_RCLK    BIT     P5.4        ; //pin 12 RCLk store (latch) clock
P_HC595_SRCLK   BIT     P4.3        ; //pin 11 SRCLK Shift data clock
;*************    本地变量声明      ***************/
Flag0           DATA    20H
B_1ms           BIT     Flag0.0     ; 1ms 标志

LED8            DATA    30H         ;  显示缓冲  30H ~ 37H
display_index   DATA    38H         ;  显示位索引
```

```
msecond_H       DATA    39H
msecond_L       DATA    3AH
BandgapH        DATA    3BH
BandgapL        DATA    3CH
ADC3_H          DATA    3DH
ADC3_L          DATA    3EH
;*******************************************************
    ORG     00H                             ;reset
    LJMP    F_Main
    ORG     0BH                             ;1 Timer0 interrupt
    LJMP    F_Timer0_Interrupt
;********************  主程序  ***************************/
F_Main:
    MOV     SP, #STACK_POIRTER
    MOV     PSW, #0
    USING   0                               ;选择第 0 组 R0~R7
;================  用户初始化程序  ================
    MOV     display_index, #0
    MOV     R0, #LED8
    MOV     R2, #8
L_ClearLoop:
    MOV     @R0, #DIS_BLACK                 ;上电消隐
    INC     R0
    DJNZ    R2, L_ClearLoop

    CLR     TR0
    ORL     AUXR, #(1 SHL 7)                ; Timer0_1T();
    ANL     TMOD, #NOT 04H                  ; Timer0_AsTimer();
    ANL     TMOD, #NOT 03H                  ; Timer0_16bitAutoReload();
    MOV     TH0, #Timer0_Reload / 256       ;Timer0_Load(Timer0_Reload);
    MOV     TL0, #Timer0_Reload MOD 256
    SETB    ET0                             ; Timer0_InterruptEnable();
    SETB    TR0                             ; Timer0_Run();
    SETB    EA                              ;  打开总中断

    LCALL   F_ADC_config                    ; ADC 初始化
;=======================================================
L_Main_Loop:
    JNB     B_1ms, L_Main_Loop              ;1ms 未到
    CLR     B_1ms
;==================  检测 300ms 是否到  ==================
    INC     msecond_L                       ;msecond + 1
    MOV     A, msecond_L
    JNZ     $+4
    INC     msecond_H
```

```
    CLR     C
    MOV     A, msecond_L                ;msecond - 300
    SUBB    A, #LOW 300
    MOV     A, msecond_H
    SUBB    A, #HIGH 300
    JC      L_Main_Loop                 ;if(msecond < 300), jmp
;================= 300ms 到 ============================
    MOV     msecond_L, #0               ;if(msecond >= 1000)
    MOV     msecond_H, #0

    MOV     A, #ADC_CH2
    LCALL   F_Get_ADC10bitResult        ;读外部基准 ADC, 查询方式做一次 ADC,
                                        ;返回值(R6 R7)就是 ADC 结果, == 1024 为错误
    MOV     BandgapH, R6                ;保存 Bandgap
    MOV     BandgapL, R7

    MOV     A, #ADC_CH3
    LCALL   F_Get_ADC10bitResult        ; 读外部电压 ADC, 查询方式做一次 ADC,
                                        ;返回值(R6 R7)就是 ADC 结果, == 1024 为错误
    MOV     ADC3_H, R6                  ;保存 adc
    MOV     ADC3_L, R7

    MOV     R4, ADC3_H                  ; adc * 123 / Bandgap, 计算外部电压,
                                        ;Bandgap 为 1.23V, 测电压分辨率 0.01V
    MOV     R5, ADC3_L
    MOV     R7, #250                    ; TL431 为 2.50V, 定点计算, 放大 100 倍
    LCALL   F_MUL16x8                   ;(R4, R5)* R7 -->(R5,R6,R7)
    MOV     R4, #0
    MOV     R2, BandgapH
    MOV     R3, BandgapL
    LCALL   F_ULDIV                     ; (R4,R5,R6,R7)/(R2,R3)=(R4,R5,R6,R7),
                                        ;余数在(R2,R3),use R0~R7,B,DPL
    LCALL   F_HEX2_DEC                  ;(R6 R7) HEX Change to DEC ---> (R3 R4 R5), use (R2~R7)
    MOV     LED8+4, #DIS_BLACK
    MOV     A, R4
    ANL     A, #0x0F
    ADD     A, #DIS_DOT                 ;显示电压值, 小数点
    MOV     LED8+5, A
    MOV     A, R5
    SWAP    A
    ANL     A, #0x0F
    MOV     LED8+6, A
    MOV     A, R5
    ANL     A, #0x0F
    MOV     LED8+7, A
L_Quit_Check_300ms:
```

```
;======================================================
    LJMP    L_Main_Loop
;*********************************************/
;//=========================================================================
;// 函数: F_HEX2_DEC
;// 描述: 把双字节十六进制数转成十进制 BCD 数.
;// 参数: (R6 R7): 要转换的双字节十六进制数.
;// 返回: (R3 R4 R5) = BCD 码.
;//=========================================================================
F_HEX2_DEC:                             ;(R6 R7) HEX Change to DEC ---> (R3 R4 R5), use (R2~R7)
    MOV     R2, #16
    MOV     R3, #0
    MOV     R4, #0
    MOV     R5, #0
L_HEX2_DEC:
    CLR     C
    MOV     A, R7
    RLC     A
    MOV     R7, A
    MOV     A, R6
    RLC     A
    MOV     R6, A
    MOV     A, R5
    ADDC    A, R5
    DA      A
    MOV     R5, A
    MOV     A, R4
    ADDC    A, R4
    DA      A
    MOV     R4, A
    MOV     A, R3
    ADDC    A, R3
    DA      A
    MOV     R3, A
    DJNZ    R2, L_HEX2_DEC
    RET
;*********************************************/
F_ADC_config:
    MOV     P1ASF, #(ADC_P12 + ADC_P13)
                                    ; 设置要做 ADC 的 IO,ADC_P10 ~ ADC_P17(或操作),ADC_P1_All
    MOV     ADC_CONTR, #(ADC_PowerOn + ADC_90T)     ;打开 ADC, 设置速度
    ORL     PCON2, #ADC_RES_H2L8
                                    ;10 位 AD 结果的高 2 位放 ADC_RES 的低 2 位,低 8 位在 ADC_RESL。
    ORL     P1M1, #(ADC_P12 + ADC_P13)              ; 把 ADC 口设置为高阻输入
    ANL     P1M0, #NOT (ADC_P12 + ADC_P13)
;   SETB    EADC                                    ;中断允许
```

```
;     SETB    PADC                                    ;优先级设置
      RET
;//=========================================================================
;// 函数: F_Get_ADC10bitResult
;// 描述: 查询法读一次 ADC 结果.
;// 参数: ACC: 选择要转换的 ADC.
;// 返回: (R6 R7) = 10 位 ADC 结果.
;//=========================================================================
F_Get_ADC10bitResult:
                ;ACC - 通道 0~7, 查询方式做一次 ADC, 返回值(R6 R7)就是 ADC 结果, == 1024 为错误
      MOV     R7, A                                   //channel
      MOV     ADC_RES, #0;
      MOV     ADC_RESL, #0;
      MOV     A, ADC_CONTR        ;ADC_CONTR = (ADC_CONTR & 0xe0) | ADC_START | channel;
      ANL     A, #0xE0
      ORL     A, #ADC_START
      ORL     A, R7
      MOV     ADC_CONTR, A
      NOP
      NOP
      NOP
      NOP
      MOV     R3, #100
L_WaitAdcLoop:
      MOV     A, ADC_CONTR
      JNB     ACC.4, L_CheckAdcTimeOut
      ANL     ADC_CONTR, #NOT ADC_FLAG            ;清除完成标志
      MOV     A, ADC_RES      ;10 位 AD 结果的高 2 位放 ADC_RES 的低 2 位，低 8 位在 ADC_RESL。
      ANL     A, #3
      MOV     R6, A
      MOV     R7, ADC_RESL
      SJMP    L_QuitAdc
L_CheckAdcTimeOut:
      DJNZ    R3, L_WaitAdcLoop
      MOV     R6, #HIGH 1024                          ;超时错误,返回 1024,调用的程序判断
      MOV     R7, #LOW 1024
L_QuitAdc:
      RET
; ************************ 显示相关程序 ****************************************
T_Display: ;标准字库
;           0   1   2   3       4   5   6   7       8   9   A   B       C   D   E   F
DB  03FH,006H,05BH,04FH,066H,06DH,07DH,007H,07FH,06FH,077H,07CH,039H,05EH,079H,071H

;     black -       H   J   K   L       N   o   P   U       t   G   Q   r       M   y
DB 000H,040H,076H,01EH,070H,038H,037H,05CH,073H,03EH,078H,03dH,067H,050H,037H,06EH
;           0.  1.  2.  3.      4.  5.  6.      7.  8.  9.      -1
```

```
DB  0BFH,086H,0DBH,0CFH,0E6H,0EDH,0FDH,087H,0FFH,0EFH,046H
T_COM:
DB  001H,002H,004H,008H,010H,020H,040H,080H       ;     位码
;//========================================================================
;// 函数: F_Send_595
;// 描述: 向 HC595 发送一个字节子程序。
;// 参数: ACC: 要发送的字节数据.
;// 返回: none.
;// 备注: 除了 ACCC 和 PSW 外, 所用到的通用寄存器都入栈
;//========================================================================
F_Send_595:
    PUSH    02H                         ;R2 入栈
    MOV     R2, #8
L_Send_595_Loop:
    RLC     A
    MOV     P_HC595_SER, C
    SETB    P_HC595_SRCLK
    CLR     P_HC595_SRCLK
    DJNZ    R2, L_Send_595_Loop
    POP     02H                         ;R2 出栈
    RET
;//========================================================================
;// 函数: F_DisplayScan
;// 描述: 显示扫描子程序。
;// 参数: none.
;// 返回: none.
;// 备注: 除了 ACCC 和 PSW 外, 所用到的通用寄存器都入栈
;//========================================================================
F_DisplayScan:
    PUSH    DPH                         ;DPH 入栈
    PUSH    DPL                         ;DPL 入栈
    PUSH    00H                         ;R0  入栈
    MOV     DPTR, #T_COM
    MOV     A, display_index
    MOVC    A, @A+DPTR
    XRL     A, #NOT LED_TYPE
    LCALL   F_Send_595                  ;输出位码
    MOV     DPTR, #T_Display
    MOV     A, display_index
    ADD     A, #LED8
    MOV     R0, A
    MOV     A, @R0
    MOVC    A, @A+DPTR
    XRL     A, #LED_TYPE
    LCALL   F_Send_595                  ;输出段码
    SETB    P_HC595_RCLK
```

```
    CLR     P_HC595_RCLK            ;锁存输出数据
    INC     display_index
    MOV     A, display_index
    ANL     A, #0F8H                ; if(display_index >= 8)
    JZ      L_QuitDisplayScan
    MOV     display_index, #0       ;8 位结束回 0
L_QuitDisplayScan:
    POP     00H                     ;R0  出栈
    POP     DPL                     ;DPL 出栈
    POP     DPH                     ;DPH 出栈
    RET
;***************  中断函数  ************************************************
F_Timer0_Interrupt:                 ;Timer0 1ms 中断函数
    PUSH    PSW                     ;PSW 入栈
    PUSH    ACC                     ;ACC 入栈
    LCALL   F_DisplayScan           ; 1ms 扫描显示一位
    SETB    B_1ms                   ; 1ms 标志
    POP     ACC                     ;ACC 出栈
    POP     PSW                     ;PSW 出栈
    RETI
;********************************************************************************
F_ULDIV:
F_DIV32:            ; (R4,R5,R6,R7)/(R2,R3)=(R4,R5,R6,R7),余数在(R2,R3),use R0~R7,B,DPL
    CJNE    R2, #0, F_DIV32_16      ; L_0075
F_DIV32_8:          ;R3 非 0，  (R4,R5,R6,R7)/R3=(R4,R5,R6,R7),余数在  R3,use R0~R7,B
    MOV     A, R4
    MOV     B, R3
    DIV     AB
    XCH     A, R7
    XCH     A, R6
    XCH     A, R5
    MOV     R4, A
    MOV     A, B
    XCH     A, R3
    MOV     R1, A
    MOV     R0, #24
L_0056:
    MOV     A, R7
    ADD     A, R7
    MOV     R7, A
    MOV     A, R6
    RLC     A
    MOV     R6, A
    MOV     A, R5
    RLC     A
    MOV     R5, A
```

```
    MOV     A, R4
    RLC     A
    MOV     R4, A
    MOV     A, R3
    RLC     A
    MOV     R3, A
    JBC     CY, L_006B
    SUBB    A, R1
    JC      L_006F
L_006B:
    MOV     A, R3
    SUBB    A, R1
    MOV     R3, A
    INC     R7
L_006F:
    DJNZ    R0, L_0056
    CLR     A
    MOV     R1, A
    MOV     R2, A
    RET
;*************************************************************************
F_DIV32_16:             ;R2 非 0，  (R4,R5,R6,R7)/(R2,R3)=(R5,R6,R7),余数在  (R2,R3),use R0~R7
L_0075:
    MOV     R0,#24                  ;开始 R1=0
L_0077:
    MOV     A,R7                    ;左移一位
    ADD     A, R7
    MOV     R7, A
    MOV     A, R6
    RLC     A
    MOV     R6, A
    MOV     A, R5
    RLC     A
    MOV     R5, A
    MOV     A, R4
    RLC     A
    MOV     R4, A
    XCH     A, R1
    RLC     A
    XCH     A, R1
    JBC     CY, L_008E              ;如果 C=1，肯定够减
    SUBB    A, R3
    MOV     A, R1                   ;测试是否够减
    SUBB    A, R2
    JC      L_0095
L_008E:
```

```
    MOV     A, R4
    SUBB    A, R3
    MOV     R4, A
    MOV     A, R1
    SUBB    A, R2
    MOV     R1, A
    INC     R7
L_0095:
    DJNZ    R0, L_0077
    CLR     A
    XCH     A, R1
    MOV     R2, A
    CLR     A
    XCH     A, R4
    MOV     R3, A
    RET
;*******************************************************************************
F_MUL16x8:                          ;(R4, R5)* R7 -->(R5,R6,R7)
    MOV     A, R7                   ;1T 1
    MOV     B, R5                   ;2T 3
    MUL     AB                      ;4T R3*R7 4
    MOV     R6, B                   ;1T 4
    XCH     A, R7                   ;2T 3
    MOV     B, R4                   ;1T 3
    MUL     AB                      ;4T R3*R6 4
    ADD     A, R6                   ;1T 2
    MOV     R6, A                   ;1T 3
    CLR     A                       ;1T 1
    ADD     C A, B                  ;1T 3
    MOV     R5, A                   ;1T 2
    RET                             ;4T 10
    END
```

21.10 利用 BandGap 电压精确测量外部输入电压值及测试程序

ADC 的第 9 通道是用来测试内部 BandGap 参考电压的，由于内部 BandGap 参考电压很稳定，不会随芯片的工作电压的改变而变化，所以可以通过两次测量和一次计算便可得到外部的精确电压，公式如下：

$$\frac{ADC_{bg}}{V_{bg}} = \frac{1023}{Vcc}$$

$$\frac{ADCx}{Vx} = \frac{1023}{Vcc}$$

由于两次测量的时间间隔很短，Vcc 的电压在此期间的波动可忽略不计从而可推出：

$\frac{ADC_{bg}}{V_{bg}} = \frac{ADCx}{Vx}$，进一步得出 $Vx = \frac{V_{bg} \times ADCx}{ADC_{bg}}$

其中：ADC_{bg} 为 Bandgap 电压的 ADC 测量值

V_{bg} 为实际 Bandgap 的电压值，在单片机进行 CP 测试时记录的参数，单位为毫伏(mV)

ADCx 为外部输入电压的 ADC 测量值

Vx 外部输入电压的实际电压值，单位为毫伏(mV)

具体的测试方法：首先将 P1ASF 初始化为 0，即关闭所有 P1 口的模拟功能然后通过正常的 ADC 转换的方法读取第 0 通道的值，即可通过 ADC 的第 9 通道读取当前内部 BandGap 参考电压值 ADC_{bg}，然后测量有外部电压输入的 ADC 通道，测量出外部输入电压的 ADC 测量值 ADCx，接下来从 RAM 区或者 ROM 区读取实际 Bandgap 的电压值 V_{bg}，最后通过公式：

$$Vx = \frac{V_{bg} \times ADCx}{ADC_{bg}}$$

即可计 ADC 算出外部输入电压的实际电压值 Vx。

利用 BandGap 电压精确测量外部输入电压值的测试程序如下:

```
/*----STC15W4K60S4 系列 通过 BandGap 电压精确测量外部输入电压值举例-----*/
//本示例在 Keil 开发环境下请选择 Intel 的 8058 芯片型号进行编译
//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"
#include "intrins.h"
typedef        unsigned char        BYTE;
typedef        unsigned int         WORD;
//说明:
// ADC 的第 9 通道是用来测试内部 BandGap 参考电压的,由于内部 BandGap 参考电
//压很稳定,不会随芯片的工作电压的改变而变化,所以可以通过两次测量和一次计算
//便可得到外部的精确电压.公式如下:
//ADCbg / Vbg = 1023 / VCC
//ADCx / Vx = 1023 / VCC
```

```
//由于两次测量的时间间隔很短,VCC 的电压在此期间的波动可忽略不计
//从而可推出  ADCbg / Vbg = ADCx / Vx
//进一步得出  Vx = Vbg * ADCx / ADCbg
//其中:ADCbg 为 Bandgap 电压的 ADC 测量值
// Vbg 为实际 Bandgap 的电压值,在单片机进行 CP 测试时记录的参数单位为毫伏(mV)
// ADCx 为外部输入电压的 ADC 测量值
// Vx 外部输入电压的实际电压值,单位为毫伏(mV)
//
//具体的测试方法:首先将 P1ASF 初始化为 0,即关闭所有 P1 口的模拟功能
//然后通过正常的 ADC 转换的方法读取第 0 通道的值,即可通过 ADC 的第 9 通道读取当前
//内部 BandGap 参考电压值 ADCbg,然后测量有外部电压输入的 ADC 通道,测量出
//外部输入电压的 ADC 测量值 ADCx,接下来从 RAM 区或者 ROM 区读取实际 Bandgap 的电压值 Vbg,
//最后通过公式 Vx = Vbg * ADCx / ADCbg,即可计算出外部输入电压的实际电压值 Vx
//----------------------------------------

WORD        idata     Vbg_RAM _at_ 0xef;      //对于只有 256 字节 RAM 的 MCU 存放地址为 0EFH
//WORD      idata     Vbg_RAM _at_ 0x6f;      //对于只有 128 字节 RAM 的 MCU 存放地址为 06FH

//注意:需要在下载代码时选择"在 ID 号前添加重要测试参数"选项,才可在程序中获取此参数
//WORD      code      Vbg_ROM _at_ 0x03f7;    //1K 程序空间的 MCU
//WORD      code      Vbg_ROM _at_ 0x07f7;    //2K 程序空间的 MCU
//WORD      code      Vbg_ROM _at_ 0x0bf7;    //3K 程序空间的 MCU
//WORD      code      Vbg_ROM _at_ 0x0ff7;    //4K 程序空间的 MCU
//WORD      code      Vbg_ROM _at_ 0x13f7;    //5K 程序空间的 MCU
//WORD      code      Vbg_ROM _at_ 0x1ff7;    //8K 程序空间的 MCU
//WORD      code      Vbg_ROM _at_ 0x27f7;    //10K 程序空间的 MCU
//WORD      code      Vbg_ROM _at_ 0x2ff7;    //12K 程序空间的 MCU
//WORD      code      Vbg_ROM _at_ 0x3ff7;    //16K 程序空间的 MCU
//WORD      code      Vbg_ROM _at_ 0x4ff7;    //20K 程序空间的 MCU
//WORD      code      Vbg_ROM _at_ 0x5ff7;    //24K 程序空间的 MCU
//WORD      code      Vbg_ROM _at_ 0x6ff7;    //28K 程序空间的 MCU
//WORD      code      Vbg_ROM _at_ 0x7ff7;    //32K 程序空间的 MCU
//WORD      code      Vbg_ROM _at_ 0x9ff7;    //40K 程序空间的 MCU
//WORD      code      Vbg_ROM _at_ 0xbff7;    //48K 程序空间的 MCU
//WORD      code      Vbg_ROM _at_ 0xcff7;    //52K 程序空间的 MCU
//WORD      code      Vbg_ROM _at_ 0xdff7;    //56K 程序空间的 MCU
WORD        code      Vbg_ROM _at_ 0xeff7;    //60K 程序空间的 MCU
//----------------------------------------
sfr         ADC_CONTR = 0xBC;                 //ADC 控制寄存器
sfr         ADC_RES = 0xBD;                   //ADC 高 8 位结果
sfr         ADC_LOW2 = 0xBE;                  //ADC 低 2 位结果
sfr         P1ASF = 0x9D;                     //P1 口第 2 功能控制寄存器

#define     ADC_POWER       0x80              //ADC 电源控制位
#define     ADC_FLAG        0x10              //ADC 完成标志
#define     ADC_START       0x08              //ADC 起始控制位
```

```
#define     ADC_SPEEDLL     0x00        //540 个时钟
#define     ADC_SPEEDL      0x20        //360 个时钟
#define     ADC_SPEEDH      0x40        //180 个时钟
#define     ADC_SPEEDHH     0x60        //90 个时钟

/*----------------------------软件延时----------------------------*/
void Delay(WORD n)
{
    WORD x;
    while (n--)
    {
        x = 5000;
        while (x--);
    }
}

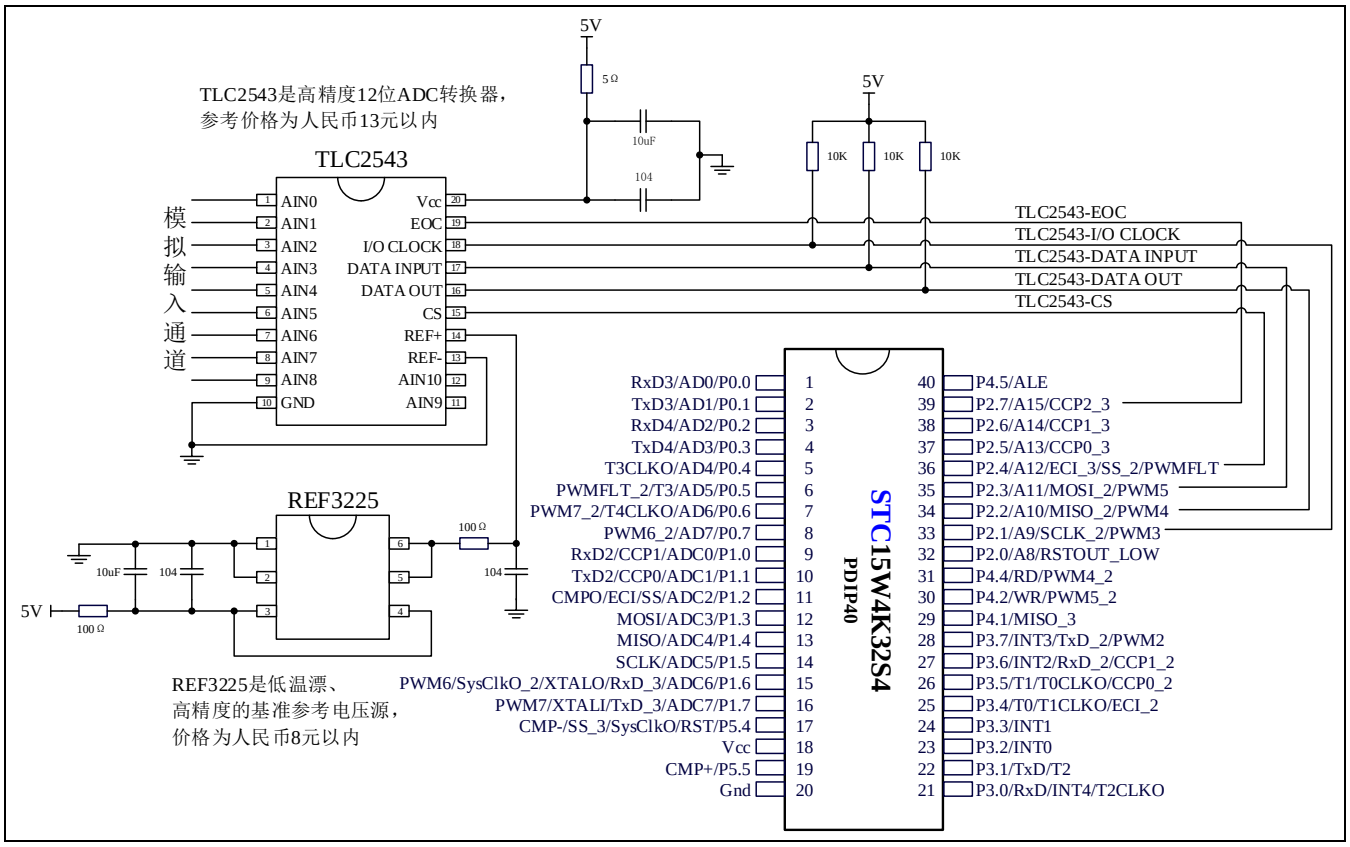
void main()
{
    BYTE    ADCbg;
    BYTE    ADCx;
    WORD    Vx;
//第一步:通过 ADC 的第 9 通道测试 Bandgap 电压的 ADC 测量值
    ADC_RES = 0;                                //清除结果寄存器
    P1ASF = 0x00;     //不设置 P1ASF,即可从 ADC 的第 9 通道读取内部 Bandgap 电压的 ADC 测量值
    ADC_CONTR = ADC_POWER | ADC_SPEEDLL;
    Delay(2);                                   //ADC 上电并延时
    ADC_CONTR = ADC_POWER | ADC_SPEEDLL | 0 | ADC_START;
    _nop_();                                    //等待 4 个 NOP
    _nop_();
    _nop_();
    _nop_();
    while (!(ADC_CONTR & ADC_FLAG));            //等待 ADC 转换完成
    ADC_CONTR &= ~ADC_FLAG;                     //清除 ADC 标志
    ADCbg = ADC_RES;

//第二步:通过 ADC 的第 2 通道测试外部输入电压的 ADC 测量值
    ADC_RES = 0;                                //清除结果寄存器
    P1ASF = 0x02;                               //设置 P1.1 口为模拟通道
    ADC_CONTR = ADC_POWER | ADC_SPEEDLL;
    Delay(2);                                   //ADC 上电并延时
    ADC_CONTR = ADC_POWER | ADC_SPEEDLL | 1 | ADC_START;
    _nop_(); //等待 4 个 NOP
    _nop_();
    _nop_();
    _nop_();
    while (!(ADC_CONTR & ADC_FLAG));            //等待 ADC 转换完成
```

```
    ADC_CONTR &= ~ADC_FLAG;                 //清除 ADC 标志
    ADCx = ADC_RES;
//第三步:通过公式计算外部输入的实际电压值
    Vx = Vbg_RAM * ADCx / ADCbg;            //使用 RAM 中的 Bandgap 电压参数进行计算
//  Vx = Vbg_ROM * ADCx / ADCbg;            //使用 ROM 中的 Bandgap 电压参数进行计算

    while (1);
}
```

21.11 利用 SPI 接口扩展 12 位 ADC(TLC2543)的应用线路图



22 STC15 系列 CCP/PCA/PWMIDAC 应用

STC15 系列部分单片机集成了 3 路可编程计数器阵列(CCP/PCA)模块(STC15W4K32S4 系列单片机只有两路 CCP/PCA)，可用于软件定时器、外部脉冲的捕捉、高速脉冲输出以及脉宽调制(PWM)输出。

下表总结了 STC15 系列单片机内部集成了 CCP/PCA/PWM 功能的单片机型号：

特殊外围设备 / 单片机型号	8 路 10 位高速 A/D 转换器	CCP/PCA/PWM 功能	1 组高速同步串行口 SPI
STC15W4K32S4 系列	√	√ （该系列只有两路 CCP/PCA，无 CCP2）	√
STC15F2K60S2 系列	√	√	√
STC15W1K16S 系列			√
STC15W404S 系列			√
STC15W408AS 系列	√	√	√
STC15W204S 系列			
STC15F408AD 系列	√	√	√
STC15F104W 系列			

上表中√表示对应的系列有相应的功能。

STC15F2K60S2 系列和 STC15F408AD 系列单片机的 CCP/PWM/PCA 均可以在 3 组不同管脚之间进行切换：

[CCP0/P1.1, CCP1/P1.0, CCP2/CCP2_2/P3.7]；
[CCP0_2/P3.5, CCP1_2/P3.6, CCP2/CCP2_2/P3.7]；
[CCP0_3/P2.5, CCP1_3/P2.6, CCP2_3/P2.7]。

STC15W408AS 系列单片机的 CCP/PWM/PCA 可以在 2 组不同管脚之间进行切换：

[CCP0/P1.1, CCP1/P1.0, CCP2/CCP2_2/P3.7]；
[CCP0_2/P3.5, CCP1_2/P3.6, CCP2/CCP2_2/P3.7]。

STC15W4K32S4 系列单片机只有两路 CCP/PWM/PCA，该两路 CCP/PWM/PCA 均可以在 3 组不同管脚之间进行切换：

[CCP0/P1.1, CCP1/P1.0]；
[CCP0_2/P3.5, CCP1_2/P3.6]；
[CCP0_3/P2.5, CCP1_3/P2.6]。

STC15W1K16S 系列、STC15W404S 系列、STC15W204S 系列和 STC15F104W 单片机没有 CCP/PWM/PCA 功能。

22.1 与 CCP/PWM/PCA 应用有关的特殊功能寄存器

STC15 系列 1T 8051 单片机 CCP/PCA/PWM 特殊功能寄存器表 CCP/PCA/PWM SFRs

符号	描述	地址	位地址及其符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CCON	PCA Control Register	D8H	CF	CR	-	-	-	CCF2	CCF1	CCF0	00xx,x000
CMOD	PCA Mode Register	D9H	CIDL	-	-	-	CPS2	CPS1	CPS0	ECF	0xxx,0000
CCAPM0	PCA Module 0 Mode Register	DAH	-	ECOM0	CAPP0	CAPN0	MAT0	TOG0	PWM0	ECCF0	x000,0000
CCAPM1	PCA Module 1 Mode Register	DBH	-	ECOM1	CAPP1	CAPN1	MAT1	TOG1	PWM1	ECCF1	x000,0000
CCAPM2	PCA Module 2 Mode Register	DCH	-	ECOM2	CAPP2	CAPN2	MAT2	TOG2	PWM2	ECCF2	x000,0000
CL	PCA Base Timer Low	E9H									0000,0000
CH	PCA Base Timer High	F9H									0000,0000
CCAP0L	PCA Module-0 Capture Register Low	EAH									0000,0000
CCAP0H	PCA Module-0 Capture Register High	FAH									0000,0000
CCAP1L	PCA Module-1 Capture Register Low	EBH									0000,0000
CCAP1H	PCA Module-1 Capture Register High	FBH									0000,0000
CCAP2L	PCA Module-2 Capture Register Low	ECH									0000,0000
CCAP2H	PCA Module-2 Capture Register High	FCH									0000,0000
PCA_PWM0	PCA PWM Mode Auxiliary Register 0	F2H	EBS0_1	EBS0_0	-	-	-	-	EPC0H	EPC0L	00xx,xx00
PCA_PWM1	PCA PWM Mode Auxiliary Register 1	F3H	EBS1_1	EBS1_0	-	-	-	-	EPC1H	EPC1L	00xx,xx00
PCA_PWM2	PCA PWM Mode Auxiliary Register 2	F4H	EBS2_1	EBS2_0	-	-	-	-	EPC2H	EPC2L	00xx,xx00
AUXR1 P_SW1	Auxiliary Register 1	A2H	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	-	DPS	0000,0000

1、PCA 工作模式寄存器 CMOD

PCA 工作模式寄存器的格式如下：

CMOD：PCA 工作模式寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CMOD	D9H	name	CIDL	-	-	-	CPS2	CPS1	CPS0	ECF

CIDL： 空闲模式下是否停止 PCA 计数的控制位。

- 当 CIDL=0 时，空闲模式下 PCA 计数器继续工作；
- 当 CIDL=1 时，空闲模式下 PCA 计数器停止工作。

CPS2、CPS1、CPS0：PCA 计数脉冲源选择控制位。PCA 计数脉冲选择如下表所示。

CPS2	CPS1	CPS0	选择 CCP/PCA/PWM 时钟源输入
0	0	0	0，系统时钟，SYSclk/12
0	0	1	1，系统时钟，SYSclk/2
0	1	0	2，定时器 0 的溢出脉冲。由于定时器 0 可以工作在 1T 模式，所以可以达到计一个时钟就溢出，从而达到最高频率 CPU 工作时钟 SYSclk。通过改变定时器 0 的溢出率，可以实现可调频率的 PWM 输出
0	1	1	3，ECI/P1.2（或 P3.4 或 P2.4）脚输入的外部时钟（最大速率＝SYSclk/2）
1	0	0	4，系统时钟，SYSclk
1	0	1	5，系统时钟/4，SYSclk/4
1	1	0	6，系统时钟/6，SYSclk/6
1	1	1	7，系统时钟/8，SYSclk/8

例如，CPS2/CPS1/CPS0=1/0/0 时，CCP/PCA/PWM 的时钟源是 SYSclk，不用定时器 0，PWM 的频率为 SYSclk/256

如果要用系统时钟/3 来作为 PCA 的时钟源，应选择 T0 的溢出作为 CCP/PCA/PWM 的时钟源，此时应让 T0 工作在 1T 模式，计数 3 个脉冲即产生溢出。用 T0 的溢出可对系统时钟进行 1 ~ 65536 级分频（T0 工作在 16 位重装载模式）。

ECF： PCA 计数溢出中断使能位。

- 当 ECF=0 时，禁止寄存器 CCON 中 CF 位的中断；
- 当 ECF=1 时，允许寄存器 CCON 中 CF 位的中断。

2、PCA 控制寄存器 CCON

PCA 控制寄存器的格式如下：

CCON：PCA 控制控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CCON	D8H	name	CF	CR	-	-	-	CCF2	CCF1	CCF0

CF： PCA 计数器阵列溢出标志位。当 PCA 计数器溢出时，CF 由硬件置位。如果 CMOD 寄存器的 ECF 位置位，则 CF 标志可用来产生中断。CF 位可通过硬件或软件置位，但只可通过软件清零。

CR： PCA 计数器阵列运行控制位。该位通过软件置位，用来起动 PCA 计数器阵列计数。该位通过软件清零，用来关闭 PCA 计数器。

CCF2： PCA 模块 2 中断标志。当出现匹配或捕获时该位由硬件置位。该位必须通过软件清零。

CCF1： PCA 模块 1 中断标志。当出现匹配或捕获时该位由硬件置位。该位必须通过软件清零。

CCF0： PCA 模块 0 中断标志。当出现匹配或捕获时该位由硬件置位。该位必须通过软件清零。

3、PCA 比较/捕获寄存器 CCAPM0、CCAPM1 和 CCAPM2

PCA 模块 0 的比较/捕获寄存器的格式如下：

CCAPM0：PCA 模块 0 的比较/捕获寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CCAPM0	DAH	name	-	ECOM0	CAPP0	CAPN0	MAT0	TOG0	PWM0	ECCF0

B7： 保留为将来之用

ECOM0：允许比较器功能控制位。

- 当 ECOM0＝1 时，允许比较器功能。

CAPP0：正捕获控制位

- 当 CAPP0＝1 时，允许上升沿捕获。

CAPN0：负捕获控制位

- 当 CAPN0＝1 时，允许下降沿捕获。

MAT0： 匹配控制位

- 当 MAT0＝1 时，PCA 计数值与模块的比较/捕获寄存器的值的匹配将置位 CCON 寄存器的中断标志位 CCF0。

TOG0： 翻转控制位

- 当 TOG0＝1 时，工作在 PCA 高速脉冲输出模式， PCA 计数器的值与模块的比较/捕获寄存器的值的匹配将使 CCP0 脚翻转。
  （CCP0/PCA0/PWM0/P1.1 或 CCP0_2/PCA0/PWM0/P3.5 或 CCP0_3/PCA0/PWM0/P2.5）

PWM0： 脉宽调节模式

- 当 PWM0＝1 时，允许 CCP0 脚用作脉宽调节输出。
  （CCP0/PCA0/PWM0/P1.1 或 CCP0_2/PCA0/PWM0/P3.5 或 CCP0_3/PCA0/PWM0/P2.5）

ECCF0：使能 CCF0 中断。使能寄存器 CCON 的比较/捕获标志 CCF0，用来产生中断。

PCA 模块 1 的比较/捕获寄存器的格式如下：

CCAPM1：PCA 模块 1 的比较/捕获寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CCAPM1	DBH	name	-	ECOM1	CAPP1	CAPN1	MAT1	TOG1	PWM1	ECCF1

B7： 保留为将来之用

ECOM1：允许比较器功能控制位。

- 当 ECOM1＝1 时，允许比较器功能。

CAPP1：正捕获控制位

- 当 CAPP1＝1 时，允许上升沿捕获。

CAPN1：负捕获控制位

- 当 CAPN1＝1 时，允许下降沿捕获。

MAT1： 匹配控制位

- 当 MAT1＝1 时，PCA 计数值与模块的比较/捕获寄存器的值的匹配将置位 CCON 寄存器的中断标志位 CCF1。

TOG1： 翻转控制位

- 当 TOG1＝1 时，工作在 PCA 高速脉冲输出模式，PCA 计数器的值与模块的比较/捕获寄存器的值的匹配将使 CCP1 脚翻转。
  （CCP1/PCA1/PWM1/P1.0 或 CCP1_2/PCA1/PWM1/P3.6 或 CCP1_3/PCA1/PWM1/P2.6）

PWM1： 脉宽调节模式。

- 当 PWM1＝1 时，允许 CCP1 脚用作脉宽调节输出。
  （CCP1/PCA1/PWM1/P1.0 或 CCP1_2/PCA1/PWM1/P3.6 或 CCP1_3/PCA1/PWM1/P2.6）

ECCF1：使能 CCF1 中断。使能寄存器 CCON 的比较/捕获标志 CCF1，用来产生中断。

PCA 模块 2 的比较/捕获寄存器的格式如下：

CCAPM2：PCA 模块 2 的比较/捕获寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CCAPM2	DCH	name	-	ECOM2	CAPP2	CAPN2	MAT2	TOG2	PWM2	ECCF2

B7： 保留为将来之用

ECOM2：允许比较器功能控制位
- 当 ECOM2＝1 时，允许比较器功能。

CAPP2：正捕获控制位
- 当 CAPP2＝1 时，允许上升沿捕获。

CAPN2：负捕获控制位
- 当 CAPN2＝1 时，允许下降沿捕获。

MAT2： 匹配控制位
- 当 MAT2＝1 时，PCA 计数值与模块的比较/捕获寄存器的值的匹配将置位 CCON 寄存器的中断标志位 CCF2。

TOG2： 翻转控制位
- 当 TOG2＝1 时，工作在 PCA 高速脉冲输出模式，PCA 计数器的值与模块的比较/捕获寄存器的值的匹配将使 CCP2 脚翻转。
  （CCP2/PCA2/PWM2/P3.7 或 CCP2/PCA2/PWM2/P2.7）

PWM2：脉宽调节模式
- 当 PWM2＝1 时，允许 CCP2 脚用作脉宽调节输出。
  （CCP2/PCA2/PWM2/P3.7 或 CCP2/PCA2/PWM2/P2.7）

ECCF2：使能 CCF2 中断。使能寄存器 CCON 的比较/捕获标志 CCF2，用来产生中断。

4、PCA 的 16 位计数器 — 低 8 位 CL 和高 8 位 CH

CL 和 CH 地址分别为 E9H 和 F9H，复位值均为 00H，用于保存 PCA 的装载值。

5、PCA 捕捉/比较寄存器 — CCAPnL（低位字节）和 CCAPnH（高位字节）

- 当 PCA 模块用于捕获或比较时，它们用于保存各个模块的 16 位捕捉计数值；
- 当 PCA 模块用于 PWM 模式时，它们用来控制输出的占空比。

其中，n=0、1、2，分别对应模块 0、模块 1 和模块 2。复位值均为 00H。它们对应的地址分别为：

CCAP0L -- EAH、CCAP0H -- FAH：模块 0 的捕捉/比较寄存器。
CCAP1L -- EBH、CCAP1H -- FBH：模块 1 的捕捉/比较寄存器。
CCAP2L -- ECH、CCAP2H -- FCH：模块 2 的捕捉/比较寄存器。

6、PCA 模块 PWM 寄存器 PCA_PWM0、PCA_PWM1 和 PCA_PWM2

PCA 模块 0 的 PWM 寄存器的格式如下：

PCA_PWM0：PCA 模块 0 的 PWM 寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
PCA_PWM0	F2H	name	EBS0_1	EBS0_0	-	-	-	-	EPC0H	EPC0L

EBS0_1，EBS0_0：当 PCA 模块 0 工作于 PWM 模式时的功能选择位。

0， 0：PCA 模块 0 工作于 8 位 PWM 功能；

0， 1：PCA 模块 0 工作于 7 位 PWM 功能；

1， 0：PCA 模块 0 工作于 6 位 PWM 功能；

1， 1：无效，PCA 模块 0 仍工作于 8 位 PWM 模式

EPC0H：在 PWM 模式下，与 CCAP0H 组成 9 位数。

EPC0L：在 PWM 模式下，与 CCAP0L 组成 9 位数。

PCA 模块 1 的 PWM 寄存器的格式如下：

PCA_PWM1：PCA 模块 1 的 PWM 寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
PCA_PWM1	F3H	name	EBS1_1	EBS1_0	-	-	-	-	EPC1H	EPC1L

EBS1_1，EBS1_0：当 PCA 模块 1 工作于 PWM 模式时的功能选择位。

0， 0：PCA 模块 1 工作于 8 位 PWM 功能；

0， 1：PCA 模块 1 工作于 7 位 PWM 功能；

1， 0：PCA 模块 1 工作于 6 位 PWM 功能；

1， 1：无效，PCA 模块 1 仍工作于 8 位 PWM

EPC1H：在 PWM 模式下，与 CCAP1H 组成 9 位数。

EPC1L：在 PWM 模式下，与 CCAP1L 组成 9 位数。

PCA 模块 2 的 PWM 寄存器的格式如下：

PCA_PWM2：PCA 模块 2 的 PWM 寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
PCA_PWM2	F4H	name	EBS2_1	EBS2_0	-	-	-	-	EPC2H	EPC2L

EBS2_1，EBS2_0：当 PCA 模块 2 工作于 PWM 模式时的功能选择位。

0， 0：PCA 模块 2 工作于 8 位 PWM 模式；

0， 1：PCA 模块 2 工作于 7 位 PWM 模式；

1， 0：PCA 模块 2 工作于 6 位 PWM 模式；

1， 1：无效，PCA 模块 2 仍工作于 8 位 PWM

EPC2H：在 PWM 模式下，与 CCAP2H 组成 9 位数。

EPC2L：在 PWM 模式下，与 CCAP2L 组成 9 位数。

PCA 模块的工作模式设定表如下表所列：

PCA 模块工作模式设定（CCAPMn 寄存器，n=0, 1, 2）

EBSn_1	EBSn_0	-	ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn	模块功能
X	X		0	0	0	0	0	0	0	无此操作
0	0		1	0	0	0	0	1	0	8 位 PWM，无中断
0	1		1	0	0	0	0	1	0	7 位 PWM，无中断
1	0		1	0	0	0	0	1	0	6 位 PWM，无中断
1	1		1	0	0	0	0	1	0	8 位 PWM，无中断
0	0		1	1	0	0	0	1	1	8 位 PWM 输出， 由低变高可产生中断
0	1		1	1	0	0	0	1	1	7 位 PWM 输出， 由低变高可产生中断
1	0		1	1	0	0	0	1	1	6 位 PWM 输出， 由低变高可产生中断
1	1		1	1	0	0	0	1	1	8 位 PWM 输出， 由低变高可产生中断
0	0		1	0	1	0	0	1	1	8 位 PWM 输出， 由高变低可产生中断
0	1		1	0	1	0	0	1	1	7 位 PWM 输出， 由高变低可产生中断
1	0		1	0	1	0	0	1	1	6 位 PWM 输出， 由高变低可产生中断
1	1		1	0	1	0	0	1	1	8 位 PWM 输出， 由高变低可产生中断
0	0		1	1	1	0	0	1	1	8 位 PWM 输出，由低变高 或者由高变低均可产生中断
0	1		1	1	1	0	0	1	1	7 位 PWM 输出，由低变高 或者由高变低均可产生中断
1	0		1	1	1	0	0	1	1	6 位 PWM 输出，由低变高 或者由高变低均可产生中断
1	1		1	1	1	0	0	1	1	8 位 PWM 输出，由低变高 或者由高变低均可产生中断
X	X		X	1	0	0	0	0	X	16 位捕获模式， 由 CCPn/PCAn 的上升沿触发
X	X		X	0	1	0	0	0	X	16 位捕获模式， 由 CCPn/PCAn 的下降沿触发
X	X		X	1	1	0	0	0	X	16 位捕获模式， 由 CCPn/PCAn 的跳变触发
X	X		1	0	0	1	0	0	X	16 位软件定时器
X	X		1	0	0	1	1	0	X	16 位高速脉冲输出

7、将单片机的 CCP/PWM/PCA 功能在 3 组管脚之间切换的寄存器 AUXR1（P_SW1）

辅助寄存器 1 的格式如下：

AUXR1/P_SW1：外围设备切换控制寄存器（不可位寻址）

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000,0000

CCP 可在 3 个地方切换，由 CCP_S1 / CCP_S0 两个控制位来选择		
CCP_S1	CCP_S0	CCP 可在 P1/P2/P3 之间来回切换
0	0	CCP 在[P1.2/ECI，P1.1/CCP0，P1.0/CCP1，P3.7/CCP2]
0	1	CCP 在[P3.4/ECI_2，P3.5/CCP0_2，P3.6/CCP1_2，P3.7/CCP2_2]
1	0	CCP 在[P2.4/ECI_3，P2.5/CCP0_3，P2.6/CCP1_3，P2.7/CCP2_3]
1	1	无效

串口 1/S1 可在 3 个地方切换，由 S1_S0 及 S1_S1 控制位来选择		
S1_S1	S1_S0	串口 1/S1 可在 P1/P3 之间来回切换
0	0	串口 1/S1 在[P3.0/RxD，P3.1/TxD]
0	1	串口 1/S1 在[P3.6/RxD_2，P3.7/TxD_2]
1	0	串口 1/S1 在[P1.6/RxD_3/XTAL2，P1.7/TxD_3/XTAL1] 串口 1 在 P1 口时要使用内部时钟
1	1	无效

SPI 可在 3 个地方切换，由 SPI_S1 / SPI_S0 两个控制位来选择		
SPI_S1	SPI_S0	SPI 可在 P1/P2/P4 之间来回切换
0	0	SPI 在[P1.2/SS，P1.3/MOSI，P1.4/MISO，P1.5/SCLK]
0	1	SPI 在[P2.4/SS_2，P2.3/MOSI_2，P2.2/MISO_2，P2.1/SCLK_2]
1	0	SPI 在[P5.4/SS_3，P4.0/MOSI_3，P4.1/MISO_3，P4.3/SCLK_3]
1	1	无效

DPS：DPTR registers select b select bit. DPTR 寄存器选择位

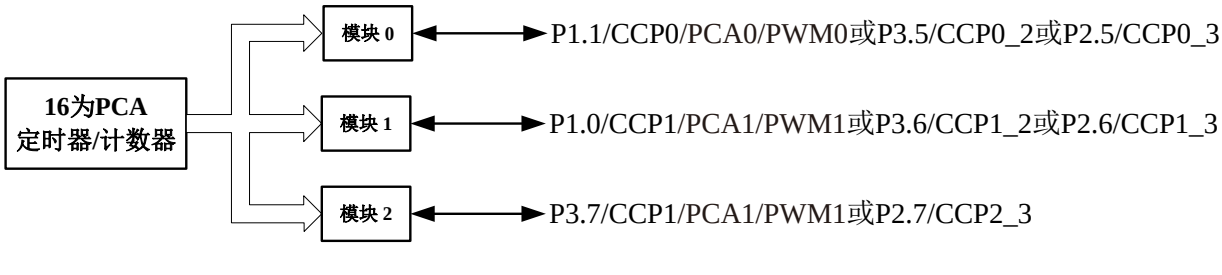
0：DPTR0 is selected DPTR0 被选择

1：DPTR1 is selected DPTR1 被选择

22.2 CCP/PWM/PCA 模块的结构

STC15 系列部分单片机有 3 路可编程计数器阵列 CCP/PCA/PWM（通过 AUXR1/PSW1 寄存器可以设置 CCP/PCA/PWM 从 P1 日切换到 P2 日切换到 P3 口）。

PCA 含有一个特殊的 16 位定时器，有 3 个 16 位的捕获/比较模块与之相连，如下图所示。

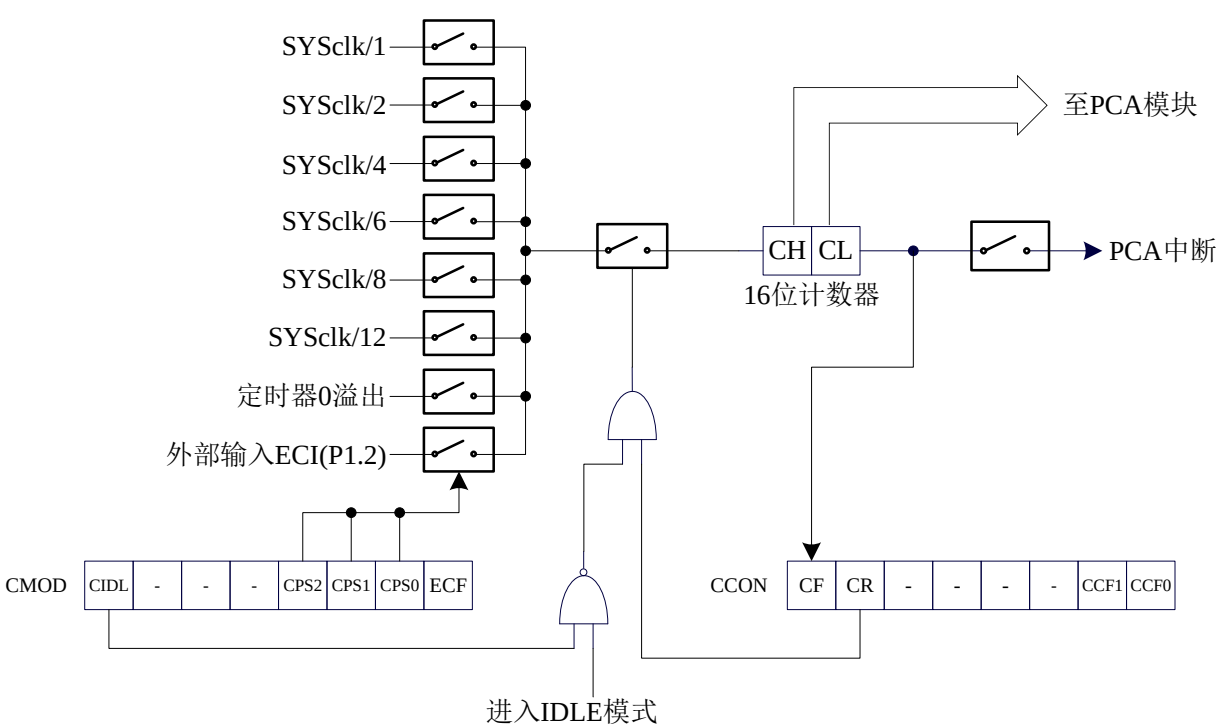


PCA模块结构

每个模块可编程工作在 4 种模式下：上升/下降沿捕获、软件定时器、高速脉冲输出或可调制脉冲输出。

STC15F2K60S2 系列：模块 0 连接到 P1.1/CCP0 或 P3.5/CCP0_2 或 P2.5/CCP0_3；
模块 1 连接到 P1.0/CCP1 或 P3.6/CCP1_2 或 P2.6/CCP1_3；
模块 2 连接到 P3.7/CCP2 或 P3.7/CCP2_2 或 P2.7/CCP2_3

16 位 PCA 定时器/计数器是 3 个模块的公共时间基准，其结构如下图所示。



PCA 定时器/计数器结构

寄存器 CH 和 CL 的内容是正在自由递增计数的 16 位 PCA 定时器的值。PCA 定时器是 3 个模块的

公共时间基准，可通过编程工作在：1/12 系统时钟、1/8 系统时钟、1/6 系统时钟、1/4 系统时钟、1/2 系统时钟、系统时钟、定时器 0 溢出或 ECI 脚的输入（STC15W4K32S4 系列在 P1.2 或 P2.4 或 P3.4 口）。定时器的计数源由 CMOD 特殊功能寄存器中的 CPS2，CPS1 和 CPS0 位来确定（见 CMOD 特殊功能寄存器说明）。

CMOD 特殊功能寄存器还有 2 个位与 PCA 相关。它们分别是：

- CIDL，空闲模式下允许停止 PCA；
- ECF，置位时，使能 PCA 中断，当 PCA 定时器溢出将 PCA 计数溢出标志 CF（CCON.7）置位。

CCON 特殊功能寄存器包含 PCA 的运行控制位（CR）和 PCA 定时器标志（CF）以及各个模块的标志（CCF2/CCF1/CCF0）。通过软件置位 CR 位（CCON.6）来运行 PCA。CR 位被清零时 PCA 关闭。当 PCA 计数器溢出时，CF 位（CCON.7）置位，如果 CMOD 寄存器的 ECF 位置位，就产生中断。CF 位只可通过软件清除。CCON 寄存器的位 0～2 是 PCA 各个模块的标志（位 0 对应模块 0，位 1 对应模块 1，位 2 对应模块 2），当发生匹配或比较时由硬件置位。这些标志也只能通过软件清除。所有模块共用一个中断向量。PCA 的中断系统如图所示。

PCA 的每个模块都对应一个特殊功能寄存器。它们分别是：模块 0 对应 CCAPM0，模块 1 对应 CCAPM1，模块 2 对应 CCAPM2，特殊功能寄存器包含了相应模块的工作模式控制位。

当模块发生匹配或比较时，ECCFn 位（CCAPMn.0，n=0，1，2 由工作的模块决定）使能 CCON 特殊功能寄存器的 CCFn 标志来产生中断。

PWM（CCAPMn.1）用来使能脉宽调制模式。

- 当 PCA 计数值与模块的捕获/比较寄存器的值相匹配时，如果 TOG 位（CCAPMn.2）置位，模块的 CCPn 输出将发生翻转。
- 当 PCA 计数值与模块的捕获/比较寄存器的值相匹配时，如果匹配位 MATn（CCAPMn.3）置位，CCON 寄存器的 CCFn 位将被置位。

CAPNn（CCAPMn.4）和 CAPPn（CCAPMn.5）用来设置捕获输入的有效沿。CAPNn 位使能下降沿有效，CAPPn 位使能上升沿有效。如果两位都置位，则两种跳变沿都被使能，捕获可在两种跳变沿产生。

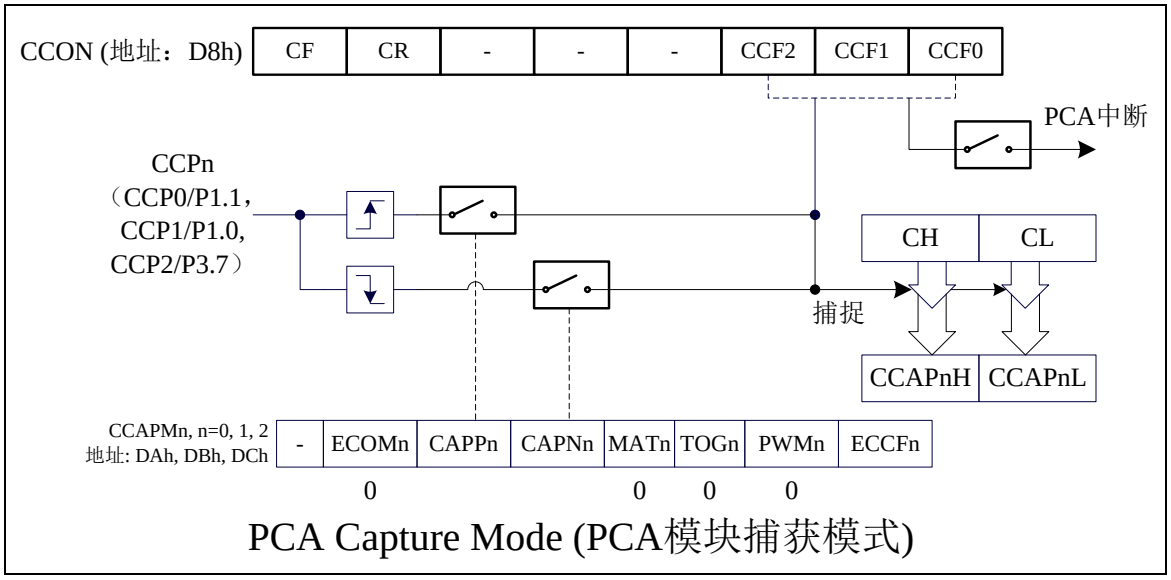
通过置位 CCAPMn 寄存器的 ECOMn 位（CCAPMn.6）来使能比较器功能。

每个 PCA 模块还对应另外两个寄存器，CCAPnH 和 CCAPnL。当出现捕获或比较时，它们用来保存 16 位的计数值。当 PCA 模块用在 PW 模式中时，它们用来控制输出的占空比。

22.3 CCP/PCA 模块的工作模式

22.3.1 捕获模式

PCA 模块工作于捕获模式的结构图如下图所示。要使一个 PCA 模块工作在捕获模式，寄存器 CCAPMn 的两位（CAPNn 和 CAPPn）或其中任何一位必须置 1。PCA 模块工作于捕获模式时，对模块的外部 CCPn 输入（CCP0/P1.1，CCP1/P1.0，CCP2/P3.7）的跳变进行采样。当采样到有效跳变时，PCA 硬件就将 PCA 计数器阵列寄存器（CH 和 CL）的值装载到模块的捕获寄存器中（CCAPnL 和 CCAPnH）

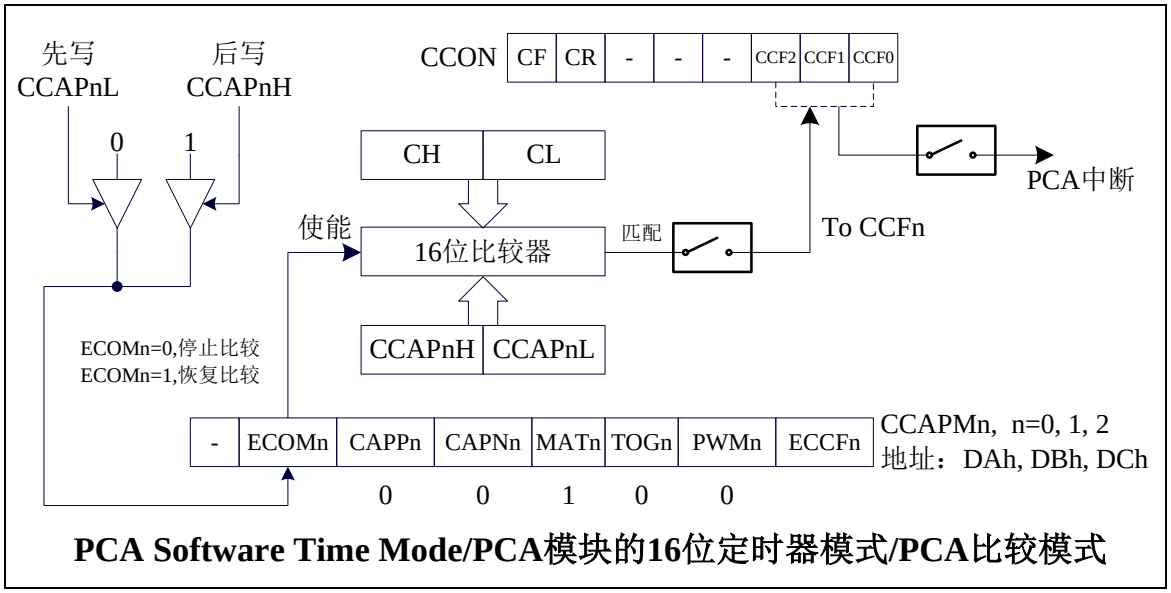


PCA Capture Mode (PCA模块捕获模式)

如果 CCON 特殊功能寄存器中的位 CCFn 和 CCAPMn 特殊功能寄存器中的位 ECCFn 位被置位，将产生中断。可在中断服务程序中判断哪一个模块产生了中断，并注意中断标志位的软件清零问题。

22.3.2 16 位软件定时器模式

16 位软件定时器模式结构图如下图所示。



PCA Software Time Mode/PCA模块的16位定时器模式/PCA比较模式

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（上图）。PCA 定时器的值与模块捕获寄存器的值相比较，当两者相等时，如果位 CCFn（在 CCON 特殊功能寄存器中）和位 ECCFn（在 CCAPMn 特殊功能寄存器中）都置位，将产生中断。

[CH, CL]每隔一定的时间自动加 1，时间间隔取决于选择的时钟源。例如，当选择的时钟源为 SYSclk/12，每 12 个时钟周期[CH, CL]加 1。当[CH, CL]增加到等于[CCAPnH, CCAPnL]时，CCFn=1，产生中断请求。如果每次 PCA 模块中断后，在中断服务程序中断给[CCAPnH, CCAPnL]增加一个相同的数值，那么下次中断来临的间隔时间 T 也是相同的，从而实现了定时功能。定时时间的长短取决于时钟源的选择以及 PCA 计数器计数值的设置。下面举例说明 PCA 计数器计数值的计算方法。

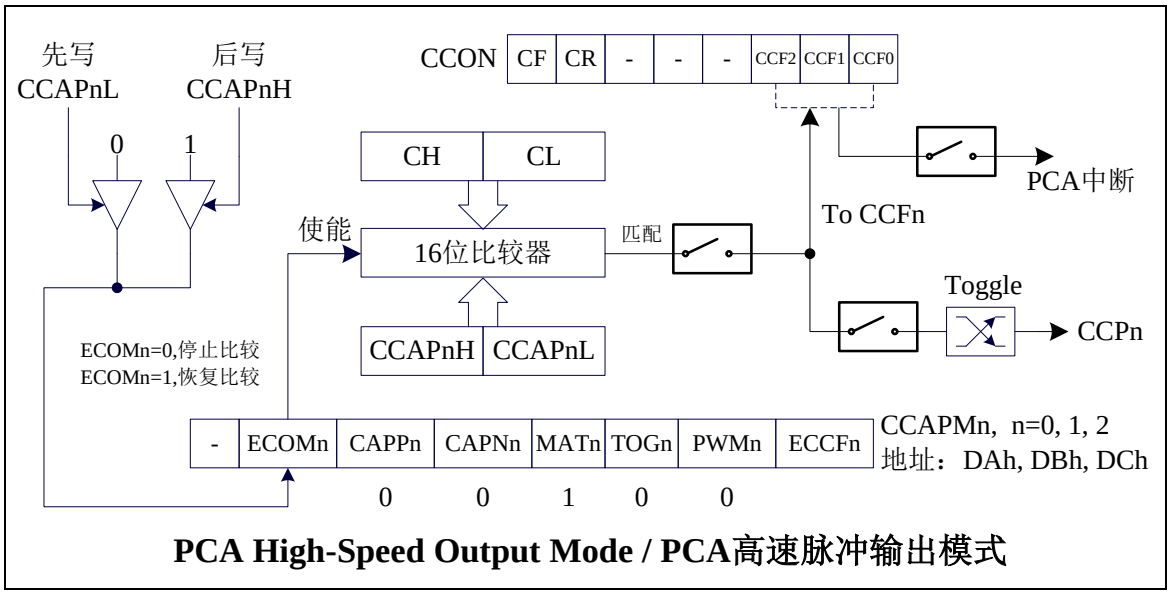
假设，系统时钟频率 SYSclk = 18.432MHz，选择的时钟源为 SYSclk/12，定时时间 T 为 5ms，则 PCA 计数器计数值为：

PCA 计数器的计数值 = T /（（1 / SYSclk ）×12）= 0.005 /（（1 / 18432000）×12）= 7680（10 进制数）
= 1E00H（16 进制数）

也就是说，PCA 计数器计数 1E00H 次，定时时间才是 5ms，这也就是每次给[CCAPnH, CCAPnL]增加的数值（步长）。

22.3.3 高速脉冲输出模式

该模式中（下图），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。要激活高速脉冲输出模式，CCAPMn 寄存器的 TOGn，MATn 和 ECOMn 位必须都置位。



PCA High-Speed Output Mode / PCA高速脉冲输出模式

CCAPnL 的值决定了 PCA 模块 n 的输出脉冲频率。当 PCA 时钟源是 SYSclk/2 时，输出脉冲的频率 F 为：

f = SYSclk /（4×CCAPnL）

其中，SYSclk 为系统时钟频率。由此，可以得到 CCAPnL 的值 CCAPnL = SYSclk /（4×f）

如果计算出的结果不是整数，则进行四舍五入取整，即

CCAPnL = INT（SYSclk /（ 4×f ） + 0.5 ）

其中，INT（）为取整运算，直接去掉小数。例如，假设 SYSclk = 20MHz，要求 PCA 高速脉冲输出 125kHz 的方波，则 CCAPnL 中的值应为：

CCAPnL = INT（20000000 /（ 4×125000 ） + 0.5 ）= INT（40 + 0.5）= 40 = 28H

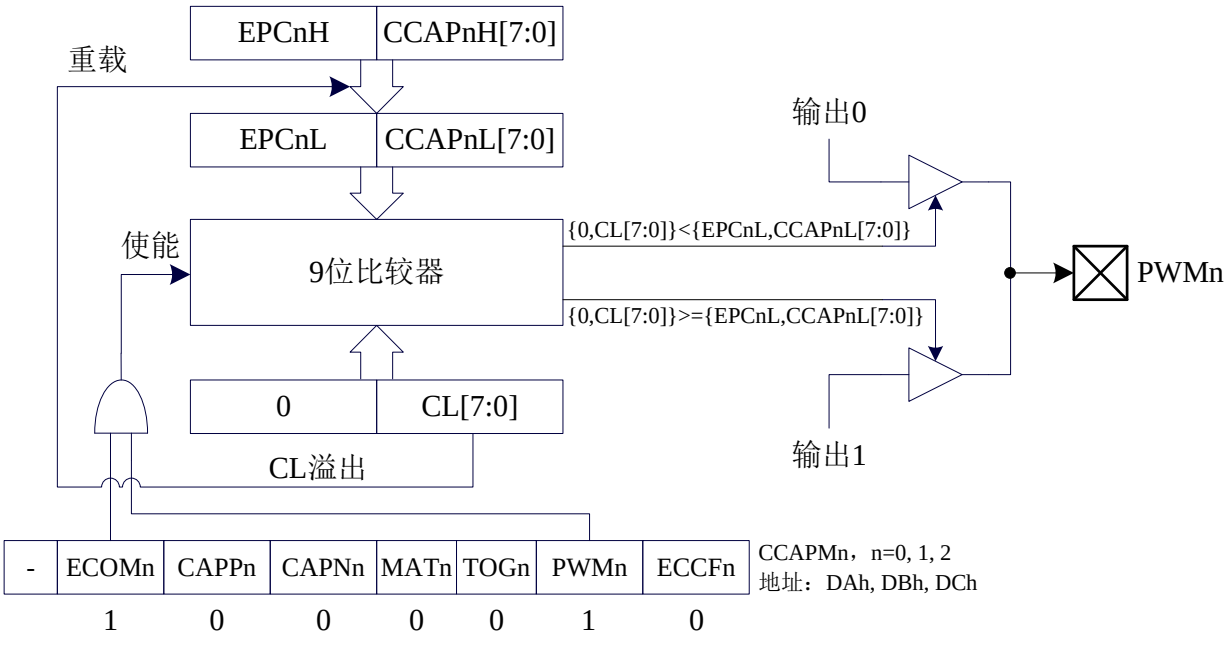
22.3.4 脉宽调节模式（PWM）

脉宽调制（PWM，Pulse Width Modulation）是一种使用程序来控制波形占空比、周期、相位波形的技术，在三相电机驱动、D/A 转换等场合有广泛的应用。

STC15 系列单片机的 PCA 模块可以通过设定各自的寄存器 PCA_PWMn（n=0，1，2.下同）中的位 EBSn_1/PCA_PWMn.7 及 EBSn_0/PCA_PWMn.6，使其工作于 8 位 PWM 或 7 位 PWM 或 6 位 PWM 模式。

22.3.4.1 8 位脉宽调节模式(PWM)

当[EBSn_1, EBSn_0] = [0, 0]或[1, 1]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。PWM 模式的结构如下图所示。



PCA PWM Mode / 可调制脉冲宽度输出模式(PCA模块工作于8位PWM模式)

当 PCA 模块工作于 8 位 PWM 模式时，由于所有模块共用仅有的 PCA 定时器，所有它们的输出频率相同。各个模块的输出占空比是独立变化的，与使用的捕获寄存器{EPCnL, CCAPnL[7: 0]}有关。

- 当{0, CL[7: 0]}的值小于{EPCnL, CCAPnL[7: 0]}时，输出为低；
- 当{0, CL[7: 0]}的值等于或大于{EPCnL, CCAPnL[7:0]}时，输出为高。

当 CL 的值由 FF 变为 00 溢出时，{EPCnH, CCAPnH[7:0]}的内容装载到{EPCnL, CCAPnL[7: 0]}中。这样就可实现无干扰地更新 PWM。要使能 PWM 模式，模块 CCAPMn 寄存器的 PWMn 和 ECOMn 位必须置位。

当 PWM 是 8 位的时：$\text{PWM的频率} = \dfrac{\text{PCA时钟输入源频率}}{256}$

PCA 时钟输入源可以从以下 8 种中选择一种：SYSclk，SYSclk/2，SYSclk/4，SYSclk/6，SYSclk/8，SYSclk/12，定时器 0 的溢出，ECI/P1.2 输入。

【举例】: 设 PCA 模块工作于 8 位 PWM 模式。要求 PWM 输出频率为 38KHz，选 SYSclk 为 PCA/PWM 时钟输入源，求出 SYSclk 的值。

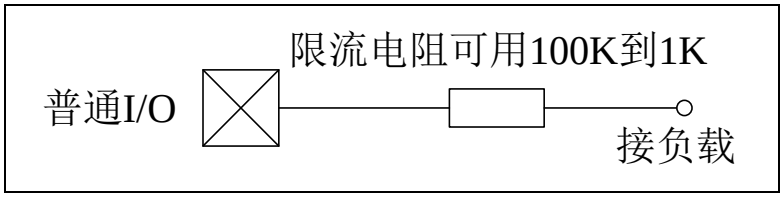
由计算公式 38000 = SYSclk/256，得到外部时钟频率 SYSclk = 38000×256×1 = 9,728,000

如果要实现可调频率的 PWM 输出，可选择定时器 0 的溢出率或者 ECI 脚的输入作为 PCA/PWM 的时钟输入源

- 当 EPCnL = 0 及 CCAPnL = 00H 时，PWM 固定输出高
- 当 EPCnL = 1 及 CCAPnL = FFH 时，PWM 固定输出低

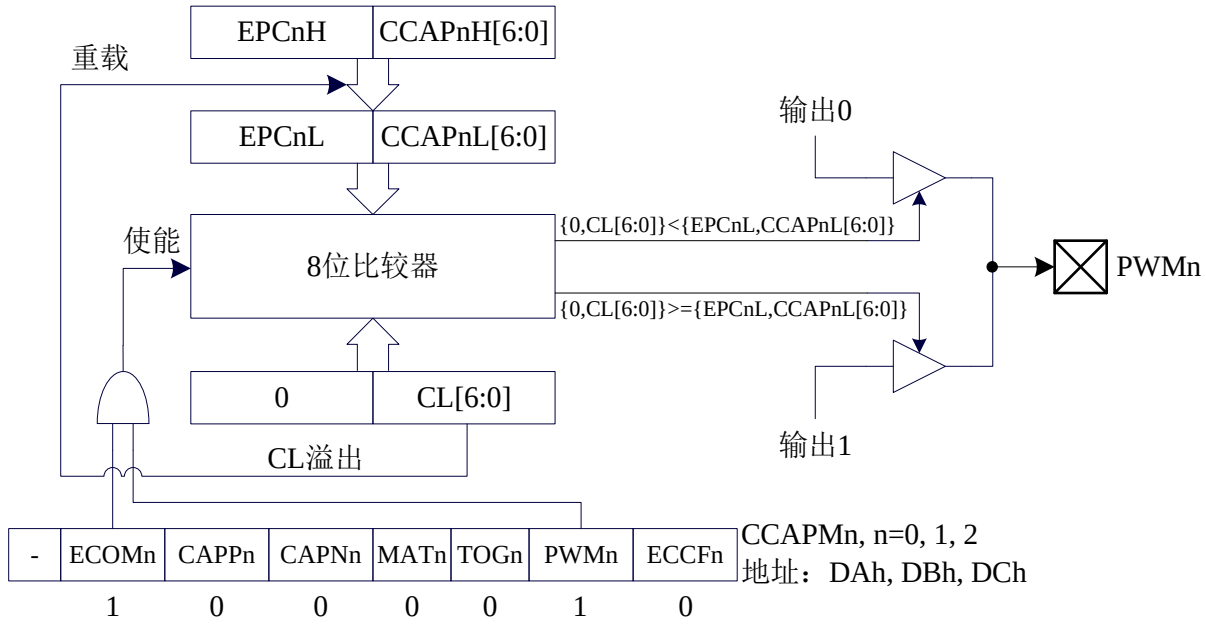
当某个 I/O 口作为 PWM 使用时，该口的状态：

PWM 之前口的状态	PWM 输出时口的状态
弱上拉/准双向	强推挽输出/强上拉输出，要加输出限流电阻 1K-10K
强推挽输出/强上拉输出	强推挽输出/强上拉输出，要加输出限流电阻 1K-10K
仅为输入/高阻	PWM 无效
开漏	开



22.3.4.2 7 位脉宽调节模式(PWM)(STC 创新设计)

当[EBSn_1, EBSn_0] = [0, 1]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。PWM 模式的结构如下图所示。



PCA PWM Mode / 可调制脉冲宽度输出模式(PCA模块工作于7位PWM模式)

当 PCA 模块工作于 7 位 PWM 模式时，由于所有模块共用仅有的 PCA 定时器，所有它们的输出频率相同。各个模块的输出占空比是独立变化的，与使用的捕获寄存器{EPCnL, CCAPnL[6:0]}有关。

- 当{0, CL[6: 0]}的值小于{EPCnL, CCAPnL[6: 0]}时，输出为低；
- 当{0, CL[6: 0]}的值等于或大于{EPCnL, CCAPnL[6:0]}时，输出为高。

当 CL 的值由 7F 变为 00 溢出时，{EPCnH, CCAPnH[6: 0]}的内容装载到{EPCnL, CCAPnL[6: 0]}中。这样就可实现无干扰地更新 PWM。要使能 PWM 模式，模块 CCAPMn 寄存器的 PWMn 和 ECOMn 位必须置位。

当 PWM 是 7 位的时：$\text{PWM的频率} = \dfrac{\text{PCA时钟输入源频率}}{128}$

PCA 时钟输入源可以从以下 8 种中选择一种：SYSclk，SYSclk/2，SYSclk/4，SYSclk/6，SYSclk/8，SYSclk/12，定时器 0 的溢出，ECI/P1.2 输入。

【举例】: 设 PCA 模块工作于 7 位 PWM 模式。要求 PWM 输出频率为 38KHz，选 SYSclk 为 PCA/PWM 时钟输入源，求出 SYSclk 的值。

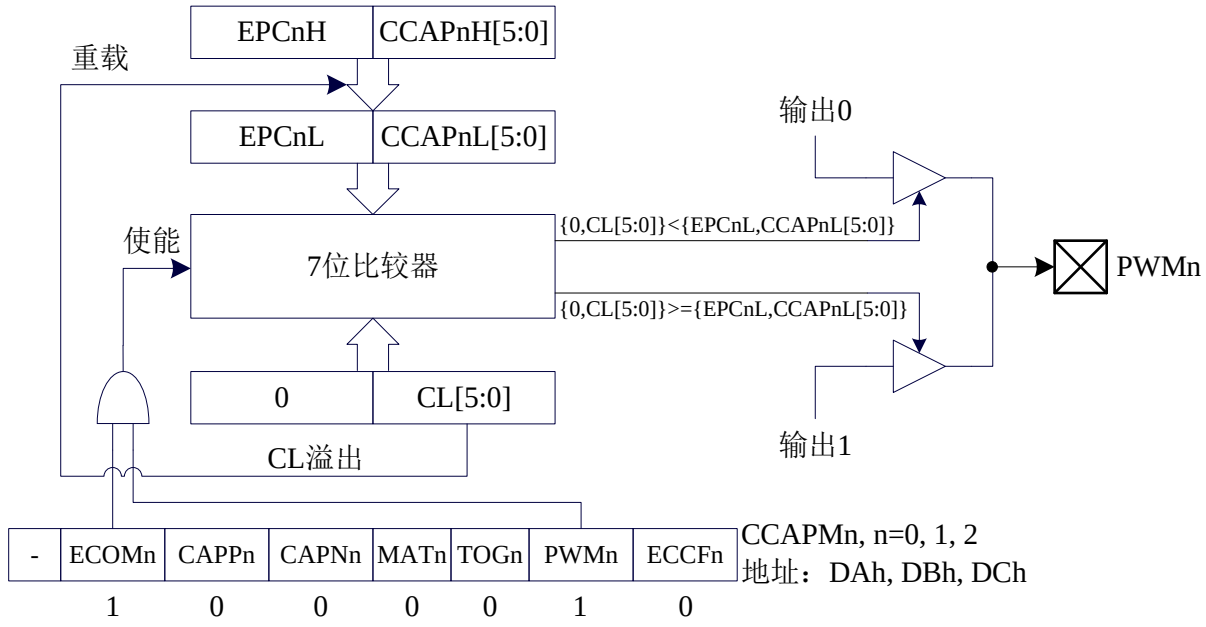
由计算公式 38000 = SYSclk / 128，得到外部时钟频率 SYSclk = 38000×128×1 = 4,864,000

如果要实现可调频率的 PWM 输出，可选择定时器 0 的溢出率或者 ECI 脚的输入作为 PCA/PWM 的时钟输入源。

- 当 EPCnL = 0 及 CCAPnL = 80H 时，PWM 固定输出高
- 当 EPCnL = 1 及 CCAPnL = 0FFH 时，PWM 固定输出低

22.3.4.3 6 位脉宽调节模式(PWM)(STC 创新设计)

当[EBSn_1, EBSn_0] = [1, 0]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。PWM 模式的结构如下图所示



PCA PWM Mode / 可调制脉冲宽度输出模式(PCA模块工作于6位PWM模式)

当 PCA 模块工作于 6 位 PWM 模式时，由于所有模块共用仅有的 PCA 定时器，所有它们的输出频率相同。各个模块的输出占空比是独立变化的，与使用的捕获寄存器{EPCnL, CCAPnL[5: 0]}有关。

- 当{0, CL[5: 0]}的值小于{EPCnL, CCAPnL[5: 0]}时，输出为低；
- 当{0, CL[5: 0]}的值等于或大于{EPCnL, CCAPnL[5: 0]}时，输出为高。

当 CL 的值由 3F 变为 00 溢出时，{EPCnH, CCAPnH[5:0]}的内容装载到{EPCnL, CCAPnL[5: 0]}中。这样就可实现无干扰地更新 PWM。要使能 PWM 模式，模块 CCAPMn 寄存器的 PWMn 和 ECOMn 位必须置位。

当 PWM 是 6 位的时：$\text{PWM的频率} = \dfrac{\text{PCA时钟输入源频率}}{64}$

PCA 时钟输入源可以从以下 8 种中选择一种：SYSclk，SYSclk/2，SYSclk/4，SYSclk/6，SYSclk/8，SYSclk/12，定时器 0 的溢出，ECI/P1.2 输入。

【举例】：设 PCA 模块工作于 6 位 PWM 模式。要求 PWM 输出频率为 38KHz，选 SYSclk 为 PCA/PWM 时钟输入源，求出 SYSclk 的值。

由计算公式 38000 = SYSclk/64，得到外部时钟频率 SYSclk = 38000×64×1 = 2,432,000

如果要实现可调频率的 PWM 输出，可选择定时器 0 的溢出率或者 ECI 脚的输入作为 PCA/PWM 的时钟输入源

- 当 EPCnL = 0 及 CCAPnL = 0C0H 时，PWM 固定输出高
- 当 EPCnL = 1 及 CCAPnL = 0FFH 时，PWM 固定输出低

22.4 用 CCP/PCA 功能扩展外部中断的测试程序(C 和汇编)

1.C 程序:

```
/*----演示 STC1T 系列单片机用 PCA 功能扩展外部中断----------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可----*/
//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"
#include "intrins.h"

typedef   unsigned char     BYTE;
typedef   unsigned int      WORD;

sfr       P_SW1 = 0xA2;                 //外设功能切换寄存器 1

#define   CCP_S0            0x10        //P_SW1.4
#define   CCP_S1            0x20        //P_SW1.5

sfr       CCON = 0xD8;                  //PCA 控制寄存器
sbit      CCF0 = CCON^0;                //PCA 模块 0 中断标志
sbit      CCF1 = CCON^1;                //PCA 模块 1 中断标志
sbit      CR = CCON^6;                  //PCA 定时器运行控制位
sbit      CF = CCON^7;                  //PCA 定时器溢出标志
sfr       CMOD = 0xD9;                  //PCA 模式寄存器
sfr       CL = 0xE9;                    //PCA 定时器低字节
sfr       CH = 0xF9;                    //PCA 定时器高字节
sfr       CCAPM0 = 0xDA;                //PCA 模块 0 模式寄存器
sfr       CCAP0L = 0xEA;                //PCA 模块 0 捕获寄存器 LOW
sfr       CCAP0H = 0xFA;                //PCA 模块 0 捕获寄存器 HIGH
sfr       CCAPM1 = 0xDB;                //PCA 模块 1 模式寄存器
sfr       CCAP1L = 0xEB;                //PCA 模块 1 捕获寄存器 LOW
sfr       CCAP1H = 0xFB;                //PCA 模块 1 捕获寄存器 HIGH
sfr       PCAPWM0 = 0xf2;
sfr       PCAPWM1 = 0xf3;

sbit      PCA_LED = P1^0;               //PCA 测试 LED

void PCA_isr() interrupt 7 using 1
{
    CCF0 = 0;                           //清中断标志
    PCA_LED = !PCA_LED;                 //测试 LED 取反
}

void main()
{
```

```
    ACC = P_SW1;
    ACC &= ~(CCP_S0 | CCP_S1);      //CCP_S0=0 CCP_S1=0
    P_SW1 = ACC;                    //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)
//  ACC = P_SW1;
//  ACC &= ~(CCP_S0 | CCP_S1);      //CCP_S0=1 CCP_S1=0
//  ACC |= CCP_S0;                  //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
//  P_SW1 = ACC;
//
//  ACC = P_SW1;
//  ACC &= ~ (CCP_S0 | CCP_S1);     //CCP_S0=0 CCP_S1=1
//  ACC |= CCP_S1;                  //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
//  P_SW1 = ACC;
    CCON = 0;                       //初始化 PCA 控制寄存器
                                    //PCA 定时器停止
                                    //清除 CF 标志
                                    //清除模块中断标志
    CL = 0;                         //复位 PCA 寄存器
    CH = 0;
    CMOD = 0x00;                    //设置 PCA 时钟源
                                    //禁止 PCA 定时器溢出中断
    CCAPM0 = 0x11;                  //PCA 模块 0 为下降沿触发
//  CCAPM0 = 0x21;                  //PCA 模块 0 为上升沿沿触发
//  CCAPM0 = 0x31;                  //PCA 模块 0 为上升沿/下降沿触发
    CR = 1;                         //PCA 定时器开始工作
    EA = 1;
    while (1);
}
```

2. 汇编程序：

```
/*----演示 STC 1T 系列单片机用 PCA 功能扩展外部中断  ---------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-----*/
//假定测试芯片的工作频率为 18.432MHz
P_SW1       EQU     0A2H        ;外设功能切换寄存器 1

CCP_S0      EQU     10H         ;P_SW1.4
CCP_S1      EQU     20H         ;P_SW1.5

CCON        EQU     0D8H        ;PCA 控制寄存器
CCF0        BIT     CCON.0      ;PCA 模块 0 中断标志
CCF1        BIT     CCON.1      ;PCA 模块 1 中断标志
CR          BIT     CCON.6      ;PCA 定时器运行控制位
CF          BIT     CCON.7      ;PCA 定时器溢出标志
CMOD        EQU     0D9H        ;PCA 模式寄存器
CL          EQU     0E9H        ;PCA 定时器低字节
```

```
CH          EQU     0F9H        ;PCA 定时器高字节
CCAPM0      EQU     0DAH        ;PCA 模块 0 模式寄存器
CCAP0L      EQU     0EAH        ;PCA 模块 0 捕获寄存器 LOW
CCAP0H      EQU     0FAH        ;PCA 模块 0 捕获寄存器 HIGH
CCAPM1      EQU     0DBH        ;PCA 模块 1 模式寄存器
CCAP1L      EQU     0EBH        ;PCA 模块 1 捕获寄存器 LOW
CCAP1H      EQU     0FBH        ;PCA 模块 1 捕获寄存器 HIGH

PCA_LED     BIT     P1.0        ;PCA 测试 LED
;---------------------------------------
    ORG     0000H
    LJMP    START

    ORG     003BH
PCA_ISR:
    CLR     CCF0                ;清中断标志
    CPL     PCA_LED             ;测试 LED 取反
    RETI
;---------------------------------------
    ORG     0100H
START:
    MOV     A, P_SW1
    ANL     A, #0CFH            ;CCP_S0=0 CCP_S1=0
    MOV     P_SW1, A            ;(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)
;   MOV     A, P_SW1
;   ANL     A, #0CFH            ;CCP_S0=1 CCP_S1=0
;   ORL     A, #CCP_S0          ;(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
;   MOV     P_SW1, A
;   MOV     A, P_SW1
;   ANL     A, #0CFH            ;CCP_S0=0 CCP_S1=1
;   ORL     A, #CCP_S1          ;(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
;   MOV     P_SW1, A
    MOV     CCON, #0            ;初始化 PCA 控制寄存器
                                ;PCA 定时器停止
                                ;清除 CF 标志
                                ;清除模块中断标志
    CLR     A
    MOV     CL, A               ;复位 PCA 寄存器
    MOV     CH, A
    MOV     CMOD, #00H          ;设置 PCA 时钟源
                                ;禁止 PCA 定时器溢出中断
    MOV     CCAPM0, #11H        ;PCA 模块 0 捕获 CCP0(P1.3)的下降沿
;   MOV     CCAPM0, #21H        ;PCA 模块 0 捕获 CCP0(P1.3)的上升沿
;   MOV     CCAPM0, #31H        ;PCA 模块 0 捕获 CCP0(P1.3)的上升沿/下降沿
;-------------------------------
    SETB    CR                  ;PCA 定时器开始工作
```

```
    SETB    EA
    SJMP    $
;---------------------------------------
END
```

22.5 用 CCP/PCA 功能实现 16 位定时器的测试程序(C 和汇编)

1.C 程序:

```
/*----演示 STC1T 系列单片机用 PCA 功能实现 16 位定时器----------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可----*/
//假定测试芯片的工作频率为 18.432MHz

#include "reg51.h"
#include "intrins.h"

#define     FOSC            18432000L
#define     T100Hz          (FOSC / 12 / 100)

typedef     unsigned char   BYTE;
typedef     unsigned int    WORD;

sfr         P_SW1 = 0xA2;                   //外设功能切换寄存器 1

#define     CCP_S0          0x10            //P_SW1.4
#define     CCP_S1          0x20            //P_SW1.5

sfr         CCON = 0xD8;                    //PCA 控制寄存器
sbit        CCF0 = CCON^0;                  //PCA 模块 0 中断标志
sbit        CCF1 = CCON^1;                  //PCA 模块 1 中断标志
sbit        CR = CCON^6;                    //PCA 定时器运行控制位
sbit        CF = CCON^7;                    //PCA 定时器溢出标志
sfr         CMOD = 0xD9;                    //PCA 模式寄存器
sfr         CL = 0xE9;                      //PCA 定时器低字节
sfr         CH = 0xF9;                      //PCA 定时器高字节
sfr         CCAPM0 = 0xDA;                  //PCA 模块 0 模式寄存器
sfr         CCAP0L = 0xEA;                  //PCA 模块 0 捕获寄存器 LOW
sfr         CCAP0H = 0xFA;                  //PCA 模块 0 捕获寄存器 HIGH
sfr         CCAPM1 = 0xDB;                  //PCA 模块 1 模式寄存器
sfr         CCAP1L = 0xEB;                  //PCA 模块 1 捕获寄存器 LOW
sfr         CCAP1H = 0xFB;                  //PCA 模块 1 捕获寄存器 HIGH
sfr         PCAPWM0 = 0xf2;
sfr         PCAPWM1 = 0xf3;

sbit        PCA_LED = P1^0;                 //PCA 测试 LED
```

```
BYTE        cnt;
WORD        value;
void PCA_isr() interrupt 7 using 1
{
    CCF0 = 0;                           //清中断标志
    CCAP0L = value;

    CCAP0H = value >> 8;                //更新比较值
    value += T100Hz;
    if (cnt-- == 0)
    {
        cnt = 100;                      //记数 100 次
        PCA_LED = !PCA_LED;             //每秒闪烁一次
    }
}

void main()
{
    ACC = P_SW1;
    ACC &= ~(CCP_S0 | CCP_S1);          //CCP_S0=0 CCP_S1=0
    P_SW1 = ACC;                        //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)
//  ACC = P_SW1;
//  ACC &= ~(CCP_S0 | CCP_S1);          //CCP_S0=1 CCP_S1=0
//  ACC |= CCP_S0;                      //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
//  P_SW1 = ACC;
//
//  ACC = P_SW1;
//  ACC &= ~(CCP_S0 | CCP_S1);          //CCP_S0=0 CCP_S1=1
//  ACC |= CCP_S1;                      //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
//  P_SW1 = ACC;
    CCON = 0;                           //初始化 PCA 控制寄存器
                                        //PCA 定时器停止
                                        //清除 CF 标志
                                        //清除模块中断标志
    CL = 0;                             //复位 PCA 寄存器
    CH = 0;
    CMOD = 0x00;                        //设置 PCA 时钟源
                                        //禁止 PCA 定时器溢出中断
    value = T100Hz;
    CCAP0L = value;
    CCAP0H = value >> 8;                //初始化 PCA 模块 0
    value += T100Hz;

    CCAPM0 = 0x49;                      //PCA 模块 0 为 16 位定时器模式
    CR = 1;                             //PCA 定时器开始工作
    EA = 1;
```

```
    cnt = 0;
    while (1);
}
```

2. 汇编程序：

```
/*----演示 STC 1T 系列单片机用 PCA 功能实现 16 位定时器 -------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可--------*/
//假定测试芯片的工作频率为 18.432MHz
T100Hz      EQU     3C00H           ;(18432000 / 12 / 100)
P_SW1       EQU     0A2H            ;外设功能切换寄存器 1

CCP_S0      EQU     10H             ;P_SW1.4
CCP_S1      EQU     20H             ;P_SW1.5

CCON        EQU     0D8H            ;PCA 控制寄存器
CCF0        BIT     CCON.0          ;PCA 模块 0 中断标志
CCF1        BIT     CCON.1          ;PCA 模块 1 中断标志
CR          BIT     CCON.6          ;PCA 定时器运行控制位
CF          BIT     CCON.7          ;PCA 定时器溢出标志
CMOD        EQU     0D9H            ;PCA 模式寄存器
CL          EQU     0E9H            ;PCA 定时器低字节
CH          EQU     0F9H            ;PCA 定时器高字节
CCAPM0      EQU     0DAH            ;PCA 模块 0 模式寄存器
CCAP0L      EQU     0EAH            ;PCA 模块 0 捕获寄存器 LOW
CCAP0H      EQU     0FAH            ;PCA 模块 0 捕获寄存器 HIGH
CCAPM1      EQU     0DBH            ;PCA 模块 1 模式寄存器
CCAP1L      EQU     0EBH            ;PCA 模块 1 捕获寄存器 LOW
CCAP1H      EQU     0FBH            ;PCA 模块 1 捕获寄存器 HIGH

PCA_LED     BIT     P1.0            ;PCA 测试 LED

CNT         EQU     20H
;---------------------------------------
    ORG     0000H
    LJMP    START
    ORG     003BH
    LJMP    PCA_ISR
;---------------------------------------
    ORG     0100H
START:
    MOV     SP, #3FH
    MOV     A, P_SW1
    ANL     A, #0CFH            ;CCP_S0=0 CCP_S1=0
    MOV     P_SW1, A            ;(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)
;   MOV     A, P_SW1
```

```
;       ANL     A, #0CFH                ;CCP_S0=1 CCP_S1=0
;       ORL     A, #CCP_S0              ;(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
;       MOV     P_SW1, A
;       MOV     A, P_SW1
;       ANL     A, #0CFH                ;CCP_S0=0 CCP_S1=1
;       ORL     A, #CCP_S1              ;(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
;       MOV     P_SW1, A
        MOV     CCON, #0                ;初始化 PCA 控制寄存器
                                        ;PCA 定时器停止
                                        ;清除 CF 标志
                                        ;清除模块中断标志
        CLR     A
        MOV     CL, A                       ;复位 PCA 寄存器
        MOV     CH, A
        MOV     CMOD, #00H              ;设置 PCA 时钟源
                                        ;禁止 PCA 定时器溢出中断
;-------------------------------
        MOV     CCAP0L, #LOW T100Hz
        MOV     CCAP0H, #HIGH T100Hz    ;初始化 PCA 模块 0
        MOV     CCAPM0, #49H            ;PCA 模块 0 为 16 位定时器模式
;-------------------------------
        SETB    CR                      ;PCA 定时器开始工作
        SETB    EA
        MOV     CNT, #100
        SJMP    $
;---------------------------------------
PCA_ISR:
        PUSH    PSW
        PUSH    ACC
        CLR     CCF0                    ;清中断标志
        MOV     A, CCAP0L
        ADD     A, #LOW T100Hz          ;更新比较值
        MOV     CCAP0L,A
        MOV     A, CCAP0H
        ADDC    A, #HIGH T100Hz
        MOV     CCAP0H, A
        DJNZ    CNT, PCA_ISR_EXIT       ;记数 100 次
        MOV     CNT, #100
        CPL     PCA_LED                 ;每秒闪烁一次
PCA_ISR_EXIT:
        POP     ACC
        POP     PSW
        RETI
;---------------------------------------
END
```

22.6 CCP/PCA 输出高速脉冲的测试程序(C 和汇编)

1.C 程序:

```
/*----演示 STC1T 系列单片机 PCA 输出高速脉冲----------------------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
//假定测试芯片的工作频率为 18.432MHz

#include "reg51.h"
#include "intrins.h"

#define     FOSC            18432000L
#define     T100KHz         (FOSC / 4 / 100000)

typedef     unsigned char   BYTE;
typedef     unsigned int    WORD;

sfr         P_SW1 = 0xA2;               //外设功能切换寄存器 1

#define     CCP_S0      0x10            //P_SW1.4
#define     CCP_S1      0x20            //P_SW1.5

sfr         CCON = 0xD8;                //PCA 控制寄存器
sbit        CCF0 = CCON^0;              //PCA 模块 0 中断标志
sbit        CCF1 = CCON^1;              //PCA 模块 1 中断标志
sbit        CR = CCON^6;                //PCA 定时器运行控制位
sbit        CF = CCON^7;                //PCA 定时器溢出标志
sfr         CMOD = 0xD9;                //PCA 模式寄存器
sfr         CL = 0xE9;                  //PCA 定时器低字节
sfr         CH = 0xF9;                  //PCA 定时器高字节
sfr         CCAPM0 = 0xDA;              //PCA 模块 0 模式寄存器
sfr         CCAP0L = 0xEA;              //PCA 模块 0 捕获寄存器 LOW
sfr         CCAP0H = 0xFA;              //PCA 模块 0 捕获寄存器 HIGH
sfr         CCAPM1 = 0xDB;              //PCA 模块 1 模式寄存器
sfr         CCAP1L = 0xEB;              //PCA 模块 1 捕获寄存器 LOW
sfr         CCAP1H = 0xFB;              //PCA 模块 1 捕获寄存器 HIGH
sfr         PCAPWM0 = 0xf2;
sfr         PCAPWM1 = 0xf3;

sbit        PCA_LED = P1^0;             //PCA 测试 LED

BYTE        cnt;
WORD        value;

void PCA_isr() interrupt 7 using 1
```

```
{
    CCF0 = 0;                           //清中断标志

    CCAP0L = value;
    CCAP0H = value >> 8;                //更新比较值
    value += T100KHz;
}

void main()
{
    ACC = P_SW1;
    ACC &= ~(CCP_S0 | CCP_S1);          //CCP_S0=0 CCP_S1=0
    P_SW1  = ACC;                       //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)
//  ACC = P_SW1;
//  ACC &= ~(CCP_S0 | CCP_S1);          //CCP_S0=1 CCP_S1=0
//  ACC |= CCP_S0;                      //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
//  P_SW1 = ACC;
//
//  ACC = P_SW1;
//  ACC &= ~(CCP_S0 | CCP_S1);          //CCP_S0=0 CCP_S1=1
//  ACC |= CCP_S1;                      //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
//  P_SW1 = ACC;
    CCON = 0;                           //初始化 PCA 控制寄存器
                                        //PCA 定时器停止
                                        //清除 CF 标志
                                        //清除模块中断标志
    CL = 0;                             //复位 PCA 寄存器
    CH = 0;
    CMOD = 0x02;                        //设置 PCA 时钟源
                                        //禁止 PCA 定时器溢出中断
    value = T100KHz;
    CCAP0L = value;                     //P1.1 输出 100KHz 方波
    CCAP0H = value >> 8;                //初始化 PCA 模块 0
    value += T100KHz;
    CCAPM0 = 0x4d;                      //PCA 模块 0 为 16 位定时器模式,同时反转 CCP0(P1.1)口
    CR = 1;                             //PCA 定时器开始工作
    EA = 1;
    cnt = 0;
    while (1);
}
```

2. 汇编程序：

```
/*----演示 STC 1T 系列单片机 PCA 输出高速脉冲  ------------------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可------*/
//假定测试芯片的工作频率为 18.432MHz
```

```
T100KHz     EQU     2EH         ;(18432000 / 4 / 100000)

P_SW1       EQU     0A2H        ;外设功能切换寄存器 1

CCP_S0      EQU     10H         ;P_SW1.4
CCP_S1      EQU     20H         ;P_SW1.5

CCON        EQU     0D8H        ;PCA 控制寄存器
CCF0        BIT     CCON.0      ;PCA 模块 0 中断标志
CCF1        BIT     CCON.1      ;PCA 模块 1 中断标志
CR          BIT     CCON.6      ;PCA 定时器运行控制位
CF          BIT     CCON.7      ;PCA 定时器溢出标志
CMOD        EQU     0D9H        ;PCA 模式寄存器
CL          EQU     0E9H        ;PCA 定时器低字节
CH          EQU     0F9H        ;PCA 定时器高字节
CCAPM0      EQU     0DAH        ;PCA 模块 0 模式寄存器
CCAP0L      EQU     0EAH        ;PCA 模块 0 捕获寄存器  LOW
CCAP0H      EQU     0FAH        ;PCA 模块 0 捕获寄存器  HIGH
CCAPM1      EQU     0DBH        ;PCA 模块 1 模式寄存器
CCAP1L      EQU     0EBH        ;PCA 模块 1 捕获寄存器  LOW
CCAP1H      EQU     0FBH        ;PCA 模块 1 捕获寄存器  HIGH
;---------------------------------------
    ORG     0000H
    LJMP    START

    ORG     003BH
PCA_ISR:
    PUSH    PSW
    PUSH    ACC
    CLR     CCF0                ;清中断标志
    MOV     A, CCAP0L
    ADD     A, #T100KHz
    MOV     CCAP0L, A
    CLR     A
    ADDC    A, CCAP0H
    MOV     CCAP0H, A
PCA_ISR_EXIT:
    POP     ACC
    POP     PSW
    RETI
;---------------------------------------
    ORG     0100H
START:
    MOV     A, P_SW1
    ANL     A, #0CFH            ;CCP_S0=0 CCP_S1=0
    MOV     P_SW1, A            ;(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)
```

```
;       MOV     A, P_SW1
;       ANL     A, #0CFH                ;CCP_S0=1 CCP_S1=0
;       ORL     A, #CCP_S0              ;(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
;       MOV     P_SW1, A
;
;       MOV     A, P_SW1
;       ANL     A, #0CFH                ;CCP_S0=0 CCP_S1=1
;       ORL     A, #CCP_S1              ;(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
;       MOV     P_SW1, A
        MOV     CCON, #0                ;初始化 PCA 控制寄存器
                                        ;PCA 定时器停止
                                        ;清除 CF 标志
                                        ;清除模块中断标志
        CLR     A
        MOV     CL, A                   ;复位 PCA 寄存器
        MOV     CH, A
        MOV     CMOD, #02H              ;设置 PCA 时钟源
                                        ;禁止 PCA 定时器溢出中断
;------------------------------
        MOV     CCAP0L, #T100KHz        ;P1.3 输出 100KHz 方波
        MOV     CCAP0H, #0              ;初始化 PCA 模块 0
        MOV     CCAPM0, #4dH            ;PCA 模块 0 为 16 位定时模式并使能 PCA 中断
;------------------------------
        SETB    CR                      ;PCA 定时器开始工作
        SETB    EA
        SJMP    $
;---------------------------------------
END
```

22.7 CCP/PCA 输出 PWM(6 位+7 位+8 位)的测试程序(C 和汇编)

1.C 程序:

```
/*----STC15F2K60S2 系列 PCA 输出 6/7/8 位 PWM 举例------------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
//假定测试芯片的工作频率为 18.432MHz

#include "reg51.h"
#include "intrins.h"

#define     FOSC            18432000L

typedef     unsigned char   BYTE;
typedef     unsigned int    WORD;
```

```
sfr         P_SW1 = 0xA2;           //外设功能切换寄存器 1

#define     CCP_S0 0x10             //P_SW1.4
#define     CCP_S1 0x20             //P_SW1.5

sfr         CCON = 0xD8;            //PCA 控制寄存器

sbit        CCF0 = CCON^0;          //PCA 模块 0 中断标志
sbit    CCF1 = CCON^1;              //PCA 模块 1 中断标志
sbit    CR = CCON^6;                //PCA 定时器运行控制位
sbit    CF = CCON^7;                //PCA 定时器溢出标志

sfr     CMOD = 0xD9;                //PCA 模式寄存器
sfr     CL = 0xE9;                  //PCA 定时器低字节
sfr     CH  = 0xF9;                 //PCA 定时器高字节
sfr     CCAPM0= 0xDA;               //PCA 模块 0 模式寄存器
sfr     CCAP0L = 0xEA;              //PCA 模块 0 捕获寄存器 LOW
sfr     CCAP0H = 0xFA;              //PCA 模块 0 捕获寄存器 HIGH
sfr     CCAPM1= 0xDB;               //PCA 模块 1 模式寄存器
sfr     CCAP1L = 0xEB;              //PCA 模块 1 捕获寄存器 LOW
sfr     CCAP1H = 0xFB;              //PCA 模块 1 捕获寄存器 HIGH
sfr     CCAPM2 = 0xDC;              //PCA 模块 2 模式寄存器
sfr     CCAP2L = 0xEC;              //PCA 模块 2 捕获寄存器 LOW
sfr     CCAP2H = 0xFC;              //PCA 模块 2 捕获寄存器 HIGH
sfr     PCA_PWM0 = 0xf2;            //PCA 模块 0 的 PWM 寄存器
sfr     PCA_PWM1 = 0xf3;            //PCA 模块 1 的 PWM 寄存器
sfr     PCA_PWM2 = 0xf4;            //PCA 模块 2 的 PWM 寄存器

void main()
{
    ACC = P_SW1;
    ACC &= ~(CCP_S0 | CCP_S1);      //CCP_S0=0 CCP_S1=0
    P_SW1 = ACC;                    //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)
//  ACC = P_SW1;

//  ACC &= ~(CCP_S0 | CCP_S1);      //CCP_S0=1 CCP_S1=0
//  ACC |= CCP_S0;                  //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
//  P_SW1 = ACC;
//
//  ACC = P_SW1;
//  ACC &= ~(CCP_S0 | CCP_S1);      //CCP_S0=0 CCP_S1=1
//  ACC |= CCP_S1;                  //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
//  P_SW1 = ACC;
    CCON = 0;                       //初始化 PCA 控制寄存器
                                    //PCA 定时器停止
                                    //清除 CF 标志
```

```
                                    //清除模块中断标志
    CL = 0;                         //复位 PCA 寄存器
    CH = 0;
    CMOD = 0x02;                    //设置 PCA 时钟源
                                    //禁止 PCA 定时器溢出中断
    PCA_PWM0 = 0x00;                //PCA 模块 0 工作于 8 位 PWM
    CCAP0H = CCAP0L = 0x20;         //PWM0 的占空比为 87.5% ((100H-20H)/100H)
    CCAPM0 = 0x42;                  //PCA 模块 0 为 8 位 PWM 模式
    PCA_PWM1 = 0x40;                //PCA 模块 1 工作于 7 位 PWM
    CCAP1H = CCAP1L = 0x20;         //PWM1 的占空比为 75% ((80H-20H)/80H)
    CCAPM1 = 0x42;                  //PCA 模块 1 为 7 位 PWM 模式
    PCA_PWM2 = 0x80;                //PCA 模块 2 工作于 6 位 PWM
    CCAP2H = CCAP2L = 0x20;         //PWM2 的占空比为 50% ((40H-20H)/40H)
    CCAPM2 = 0x42;                  //PCA 模块 2 为 6 位 PWM 模式
    CR = 1;                         //PCA 定时器开始工作
    while (1);
}
```

2. 汇编程序：

```
/*----STC15F2K60S2 系列 PCA 输出 6/7/8 位 PWM 举例------------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-------*/
//假定测试芯片的工作频率为 18.432MHz

P_SW1       EQU     0A2H        ;外设功能切换寄存器 1

CCP_S0      EQU     10H         ;P_SW1.4
CCP_S1      EQU     20H         ;P_SW1.5

CCON        EQU     0D8H        ;PCA 控制寄存器
CCF0        BIT     CCON.0      ;PCA 模块 0 中断标志
CCF1        BIT     CCON.1      ;PCA 模块 1 中断标志
CR          BIT     CCON.6      ;PCA 定时器运行控制位
CF          BIT     CCON.7      ;PCA 定时器溢出标志
CMOD        EQU     0D9H        ;PCA 模式寄存器
CL          EQU     0E9H        ;PCA 定时器低字节
CH          EQU     0F9H        ;PCA 定时器高字节
CCAPM0      EQU     0DAH        ;PCA 模块 0 模式寄存器
CCAP0L      EQU     0EAH        ;PCA 模块 0 捕获寄存器  LOW
CCAP0H      EQU     0FAH        ;PCA 模块 0 捕获寄存器  HIGH
CCAPM1      EQU     0DBH        ;PCA 模块 1 模式寄存器
CCAP1L      EQU     0EBH        ;PCA 模块 1 捕获寄存器  LOW
CCAP1H      EQU     0FBH        ;PCA 模块 1 捕获寄存器  HIGH
CCAPM2      EQU     0DCH        ;PCA 模块 2 模式寄存器
CCAP2L      EQU     0ECH        ;PCA 模块 2 捕获寄存器  LOW
CCAP2H      EQU     0FCH        ;PCA 模块 2 捕获寄存器  HIGH
```

```
PCA_PWM0    EQU   0F2H      ;PCA 模块 0 的 PWM 寄存器
PCA_PWM1    EQU   0F3H      ;PCA 模块 1 的 PWM 寄存器
PCA_PWM2    EQU   0F4H      ;PCA 模块 2 的 PWM 寄存器
;--------------------------------------
    ORG   0000H
    LJMP  START
;--------------------------------------
    ORG   0100H
START:
    MOV   A, P_SW1
    ANL   A, #0CFH          ;CCP_S0=0 CCP_S1=0
    MOV   P_SW1, A          ;(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)
;   MOV   A, P_SW1
;   ANL   A, #0CFH          ;CCP_S0=1 CCP_S1=0
;   ORL   A, #CCP_S0        ;(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
;   MOV   P_SW1, A

;   MOV   A, P_SW1
;   ANL   A, #0CFH          ;CCP_S0=0 CCP_S1=1
;   ORL   A, #CCP_S1        ;(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
;   MOV   P_SW1, A
    MOV   CCON, #0          ;初始化 PCA 控制寄存器
                            ;PCA 定时器停止
                            ;清除 CF 标志
                            ;清除模块中断标志
    CLR   A
    MOV   CL, A             ;复位 PCA 计数器
    MOV   CH, A
    MOV   CMOD, #02H        ;设置 PCA 时钟源
                            ;禁止 PCA 定时器溢出中断
;------------------------------
    MOV   PCA_PWM0, #00H    ;PCA 模块 0 工作于 8 位 PWM
    MOV   A, #020H
    MOV   CCAP0H, A         ;PWM0 的占空比为 87.5% ((100H-20H)/100H)
    MOV   CCAP0L, A
    MOV   CCAPM0, #42H      ;PCA 模块 0 为 8 位 PWM 模式
;------------------------------
    MOV   PCA_PWM1, #40H    ;PCA 模块 1 工作于 7 位 PWM
    MOV   A, #020H
    MOV   CCAP1H, A         ;PWM1 的占空比为 75% ((80H-20H)/80H)
    MOV   CCAP1L, A
    MOV   CCAPM1, #42H      ;PCA 模块 1 为 7 位 PWM 模式
;------------------------------
    MOV   PCA_PWM2, #80H    ;PCA 模块 2 工作于 6 位 PWM
    MOV   A, #020H
    MOV   CCAP2H, A         ;PWM2 的占空比为 50% ((40H-20H)/40H)
```

```
    MOV     CCAP2L, A
    MOV     CCAPM2, #42H            ;PCA 模块 2 为 6 位 PWM 模式
;-------------------------------
    SETB    CR                      ;PCA 定时器开始工作
    SJMP    $
;----------------------------------------
END
```

22.8 用 CCP/PCA 高速脉冲输出功能实现 3 路 9 ~ 16 位 PWM 的程序

----每通道占用系统时间小于 0.6%

下文为利用 CCP/PCA 高速脉冲输出功能实现 3 路 9 ~ 16 位 PWM(每通道占用系统时间小于 0.6%)的测试程序(包括 C 语言程序和汇编程序)。

1、C 语言程序

```
#include <reg52.h>
/*************************功能说明*********************
输出 3 路 9-16 位 PWM 信号:
PWM 频率=MAIN FOSC/PWM DUTY
假设 MAIN FOSc=24MHz, PWM DUTY=6000, 则输出 PWM 频率为 4000Hz
*************************************************************/
/************************用户宏定义*******************/
#define   MAIN_Fosc             24000000UL  //定义主时钟
#define   PWM_DUTY              6000        //定义 PWM 的周期，数值为 PCA 所选择的时钟脉冲个数。
#define   PWM_HIGH_MIN          80          //限制 PWM 输出的最小占空比，避免中断里重装参数时间不够。
#define   PWM_HIGH_MAX          (PWM_DUTY - PWM_HIGH_MIN)
                                            //限制 PWM 输出的最大占空比。
/*************************************************************/
#define   PCA0                          0
#define   PCA1                          1
#define   PCA2                          2
#define   PCA_Counter                   3
#define   PCA_P12_P11_P10_P37           (0<<4)
#define   PCA_P34_P35_P36_P37           (1<<4)
#define   PCA_P24_P25_P26_P27           (2<<4)
#define   PCA_Mode_PWM                  0x42
#define   PCA_Mode_Capture              0
#define   PCA_Mode_SoftTimer            0x48
#define   PCA_Mode_HighPulseOutput      0x4c
#define   PCA_Clock_1T                  (4<<1)
#define   PCA_Clock_2T                  (1<<1)
#define   PCA_Clock_4T                  (5<<1)
#define   PCA_Clock_6T                  (6<<1)
#define   PCA_Clock_8T                  (7<<1)
```

```
#define   PCA_Clock_12T          (0<<1)
#define   PCA_Clock_Timer0_OF    (2<<1)
#define   PCA_Clock_ECI          (3<<1)
#define   PCA_Rise_Active        (1<<5)
#define   PCA_Fall_Active        (1<<4)
#define   PCA_PWM_8bit           (0<<6)
#define   PCA_PWM_7bit           (1<<6)
#define   PCA_PWM_6bit           (2<<6)

#define   ENABLE                 1
#define   DISABLE                0

typedef   unsigned char          u8;
typedef   unsigned int           u16;
typedef   unsigned long          u32;

sfr       AUXR1 = 0xA2;
sfr       CCON = 0xD8;
sfr       CMOD = 0xD9;
sfr       CCAPM0 = 0xDA;         //PCA 模块 0 的工作模式寄存器。
sfr       CCAPM1 = 0xDB;         //PCA 模块 1 的工作模式寄存器。
sfr       CCAPM2 = 0xDC;         //PCA 模块 2 的工作模式寄存器。

sfr       CL = 0xE9;
sfr       CCAP0L = 0xEA;         //PCA 模块 0 的捕捉/比较寄存器低 8 位。
sfr       CCAP1L = 0xEB;         //PCA 模块 1 的捕捉/比较寄存器低 8 位。
sfr       CCAP2L = 0xEC;         //PCA 模块 2 的捕捉/比较寄存器低 8 位。

sfr       CH = 0xF9;
sfr       CCAP0H = 0xFA;         //PCA 模块 0 的捕捉/比较寄存器高 8 位。
sfr       CCAP1H = 0xFB;         //PCA 模块 1 的捕捉/比较寄存器高 8 位。
sfr       CCAP2H = 0xFC;         //PCA 模块 2 的捕捉/比较寄存器高 8 位。

sbit      CCF0 = CCON^0;         //PCA 模块 0 中断标志,由硬件置位,必须由软件清 0
sbit      CCF1 = CCON^1;         //PCA 模块 1 中断标志,由硬件置位,必须由软件清 0
sbit      CCF2 = CCON^2;         //PCA 模块 2 中断标志,由硬件置位,必须由软件清 0
sbit      CR = CCON^6;           //1: 允许 PCA 计数器计数， 0: 禁止计数。
sbit      CF = CCON^7;           //PCA 计数器溢出（CH， CL 由 FFFFH 变为 0000H）标志。
                                 //PCA 计数器溢出后由硬件置位，必须由软件清 0。
sbit      PPCA = IP^7;           //PCA 中断 优先级设定位
sfr       P2M1 = 0x95;           //P2M1.n, P2M0.n =00→Standard, 01→push-pull
sfr       P2M0 = 0x96;           //                =10→pure input, 11→open drain
//=========================================================================
sbit      P25 = P2^5;
sbit      P26 = P2^6;
sbit      P27 = P2^7;
```

```
u16     CCAP0_tmp,PWM0_high,PWM0_low;
u16     CCAP1_tmp,PWM1_high,PWM1_low;
u16     CCAP2_tmp,PWM2_high,PWM2_low;
u16     pwm0,pwm1,pwm2;

void PWMn_Update(u8 PCA_id, u16 pwm);
void PCA_Init(void);
void delay_ms(u8 ms);

/********************  主函数  **************************/
void main(void)
{
    PCA_Init();                         //PCA 初始化
    EA = 1;

    P2M1 &= ~(0xe0);                    //P2.7 P2.6 P2.5  设置为推挽输出
    P2M0 |= (0xe0);
    while (1)
    {
        delay_ms(2);
        if(++pwm0 >= PWM_HIGH_MAX) pwm0 = PWM_HIGH_MIN;
        PWMn_Update(PCA0,pwm0);
        if(++pwm1 >= PWM_HIGH_MAX) pwm1 = PWM_HIGH_MIN;
        PWMn_Update(PCA1,pwm1);
        if(++pwm2 >= PWM_HIGH_MAX) pwm2 = PWM_HIGH_MIN;
        PWMn_Update(PCA2,pwm2);
    }
}
//========================================================================
// 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.
//========================================================================
void delay_ms(u8 ms)
{
    unsigned int i;
    do
    {
        i = MAIN_Fosc / 13000;
        while(--i) ;
    }while(--ms);
}
//========================================================================
// 函数: void PWMn_SetHighReg(unsigned int high)
// 描述:  更新占空比数据。
```

```
// 参数: high:  占空比数据，即 PWM 输出高电平的 PCA 时钟脉冲个数。
// 返回: 无
//========================================================================
void PWMn_Update(u8 PCA_id, u16 pwm)
{
    if(pwm > PWM_HIGH_MAX) pwm = PWM_HIGH_MAX;
                                        //如果写入大于最大占空比数据，强制为最大占空比。
    if(pwm < PWM_HIGH_MIN) pwm = PWM_HIGH_MIN;
                                        //如果写入小于最小占空比数据，强制为最小占空比。
    if(PCA_id == PCA0)
    {
        CR = 0;                         //停止 PCA 一会，  一般不会影响 PWM。
        PWM0_high = pwm;                //数据在正确范围，则装入占空比寄存器。
        PWM0_low = PWM_DUTY - pwm;
                                        //计算并保存 PWM 输出低电平的 PCA 时钟脉冲个数。
        CR = 1;                         //启动 PCA。
    }
    else if(PCA_id == PCA1)
    {
        CR = 0;                         //停止 PCA。
        PWM1_high = pwm;                //数据在正确范围，则装入占空比寄存器。
        PWM1_low = PWM_DUTY - pwm;
                                        //计算并保存 PWM 输出低电平的 PCA 时钟脉冲个数。
        CR = 1;                         //启动 PCA。
    }
    else if(PCA_id == PCA2)
    {
        CR = 0;                         //停止 PCA。
        PWM2_high = pwm;                //数据在正确范围，则装入占空比寄存器。
        PWM2_low = PWM_DUTY - pwm;
                                        //计算并保存 PWM 输出低电平的 PCA 时钟脉冲个数。
        CR = 1; //启动 PCA。
    }
}
//========================================================================
// 函数: void PCA_Init(void)
// 描述: PCA 初始化程序.
// 参数: none
// 返回: none.
//========================================================================
void PCA_Init(void)
{
    CR = 0;
    AUXR1 = (AUXR1 & ~(3<<4)) | PCA_P24_P25_P26_P27;      //切换 I/O 口
    CCAPM0 = (PCA_Mode_HighPulseOutput | ENABLE);
                                        //16 位软件定时、高速脉冲输出、中断模式
```

```
    CCAPM1 = (PCA_Mode_HighPulseOutput | ENABLE);
    CCAPM2 = (PCA_Mode_HighPulseOutput | ENABLE);
    CH = 0;
    CL = 0;
    CMOD = (CMOD & ~(7<<1)) | PCA_Clock_1T;          //选择时钟源
    PPCA = 1;                                        // 高优先级中断
    pwm0 = (PWM_DUTY / 4 * 1);                       //给 PWM 一个初值
    pwm1 = (PWM_DUTY / 4 * 2);
    pwm2 = (PWM_DUTY / 4 * 3);

    PWMn_Update(PCA0,pwm0);
    PWMn_Update(PCA1,pwm1);
    PWMn_Update(PCA2,pwm2);

    CR = 1;                                          // 运行 PCA 定时器
}
//========================================================================
// 函数: void PCA_Handler (void) interrupt 7
// 描述: PCA 中断处理程序.
// 参数: None
// 返回: none.
//========================================================================
void PCA_Handler (void) interrupt 7
{
    if(CCF0)                                //PCA 模块 0 中断
    {
        CCF0 = 0;                           //清 PCA 模块 0 中断标志
        if(P25) CCAP0_tmp += PWM0_high;
                                            //输出为高电平，则给影射寄存器装载高电平时间长度
        else CCAP0_tmp += PWM0_low;         //输出为低电平，则给影射寄存器装载低电平时间长度
        CCAP0L = (u8)CCAP0_tmp;             //将影射寄存器写入捕获寄存器，先写 CCAP0L
        CCAP0H = (u8)(CCAP0_tmp >> 8);      //后写 CCAP0H
    }
    if(CCF1)                                //PCA 模块 1 中断
    {
        CCF1 = 0;                           //清 PCA 模块 1 中断标志
        if(P26) CCAP1_tmp += PWM1_high;     //输出为高电平，则给影射寄存器装载高电平时间长度
        else CCAP1_tmp += PWM1_low;         //输出为低电平，则给影射寄存器装载低电平时间长度
        CCAP1L = (u8)CCAP1_tmp;             //将影射寄存器写入捕获寄存器，先写 CCAP0L
        CCAP1H = (u8)(CCAP1_tmp >> 8);      //后写 CCAP0H
    }
    if(CCF2)                                //PCA 模块 2 中断
    {
        CCF2 = 0;                           //清 PCA 模块 1 中断标志
        if(P27) CCAP2_tmp += PWM2_high;     //输出为高电平，则给影射寄存器装载高电平时间长度
        else CCAP2_tmp += PWM2_low;         //输出为低电平，则给影射寄存器装载低电平时间长度
```

```
            CCAP2L = (u8)CCAP2_tmp;             //将影射寄存器写入捕获寄存器，先写 CCAP0L
            CCAP2H = (u8)(CCAP2_tmp >> 8);      //后写 CCAP0H
        }
}
```

2、汇编语言程序

```
;************* 功能说明 **************
; 输出 3 路 9~16 位 PWM 信号。
; PWM 频率 = MAIN_Fosc / PWM_DUTY, 假设 MAIN_Fosc = 24MHz, PWM_DUTY = 6000, 则输出 PWM 频率为 4000Hz.
; ****************************************
;*************************用户宏定义*************************************************
Fosc_KHZ                    EQU     24000           //定义主时钟, KHz
PWM_DUTY                    EQU     6000
                            //定义 PWM 的周期，数值为 PCA 所选择的时钟脉冲个数。
PWM_HIGH_MIN                EQU     80
                            //限制 PWM 输出的最小占空比, 避免中断里重装参数时间不够。
PWM_HIGH_MAX                EQU     (PWM_DUTY - PWM_HIGH_MIN)
                            //限制 PWM 输出的最大占空比。
;************************************************************************************
PCA0                        EQU     0
PCA1                        EQU     1
PCA2                        EQU     2
PCA_Counter                 EQU     3
PCA_P12_P11_P10_P37         EQU     (0 SHL 4)
PCA_P34_P35_P36_P37         EQU     (1 SHL 4)
PCA_P24_P25_P26_P27         EQU     (2 SHL 4)
PCA_Mode_Capture            EQU     0
PCA_Mode_SoftTimer          EQU     048H
PCA_Mode_HighPulseOutput    EQU     04CH
PCA_Clock_1T                EQU     (4 SHL 1)
PCA_Clock_2T                EQU     (1 SHL 1)
PCA_Clock_4T                EQU     (5 SHL 1)
PCA_Clock_6T                EQU     (6 SHL 1)
PCA_Clock_8T                EQU     (7 SHL 1)
PCA_Clock_12T               EQU     (0 SHL 1)
PCA_Clock_ECI               EQU     (3 SHL 1)
PCA_Rise_Active             EQU     (1 SHL 5)
PCA_Fall_Active             EQU     (1 SHL 4)

ENABLE                      EQU     1

AUXR1                       DATA    0xA2
CCON        DATA    0xD8
CMOD        DATA    0xD9
```

```
CCAPM0          DATA    0xDA        ; PCA 模块 0 的工作模式寄存器。
CCAPM1          DATA    0xDB        ; PCA 模块 1 的工作模式寄存器。
CCAPM2          DATA    0xDC        ; PCA 模块 2 的工作模式寄存器。

CL              DATA    0xE9
CCAP0L          DATA    0xEA        ; PCA 模块 0 的捕捉/比较寄存器低 8 位。
CCAP1L          DATA    0xEB        ; PCA 模块 1 的捕捉/比较寄存器低 8 位。
CCAP2L          DATA    0xEC        ; PCA 模块 2 的捕捉/比较寄存器低 8 位。

CH              DATA    0xF9
CCAP0H          DATA    0xFA        ; PCA 模块 0 的捕捉/比较寄存器高 8 位。
CCAP1H          DATA    0xFB        ; PCA 模块 1 的捕捉/比较寄存器高 8 位。
CCAP2H          DATA    0xFC        ; PCA 模块 2 的捕捉/比较寄存器高 8 位。

CCF0            BIT     CCON.0      ; PCA 模块 0 中断标志，由硬件置位，必须由软件清 0。
CCF1            BIT     CCON.1      ; PCA 模块 1 中断标志，由硬件置位，必须由软件清 0。
CCF2            BIT     CCON.2      ; PCA 模块 2 中断标志，由硬件置位，必须由软件清 0。
CR              BIT     CCON.6      ; 1: 允许 PCA 计数器计数，0: 禁止计数。
CF              BIT     CCON.7      ; PCA 计数器溢出（CH，CL 由 FFFFH 变为 0000H）标志。
                                    ;PCA 计数器溢出后由硬件置位，必须由软件清 0。
PPCA            BIT     IP.7        ; PCA 中断 优先级设定位

P2M1            DATA    095H        ; P2M1.n, P2M0.n = 00→Standard, 01→push-pull
P2M0            DATA    096H        ;                = 10→pure input, 11→open drain
;=========================================================================
P25             BIT     P2.5
P26             BIT     P2.6
P27             BIT     P2.7

PWM0_high_H     DATA    030H
PWM0_high_L     DATA    031H
PWM0_low_H      DATA    032H
PWM0_low_L      DATA    033H

PWM1_high_H     DATA    034H
PWM1_high_L     DATA    035H
PWM1_low_H      DATA    036H
PWM1_low_L      DATA    037H

PWM2_high_H     DATA    038H
PWM2_high_L     DATA    039H
PWM2_low_H      DATA    03AH
PWM2_low_L      DATA    03BH

pwm0_H          DATA    03CH
pwm0_L          DATA    03DH
```

```
pwm1_H          DATA    03EH
pwm1_L          DATA    03FH
pwm2_H          DATA    040H
pwm2_L          DATA    041H

STACK_POIRTER       EQU     0D0H        ;堆栈开始地址
;*******************************************************************
    ORG     00H                         ;reset
    LJMP    F_Main

    ORG     3BH                         ;7 PCA interrupt
    LJMP    F_PCA_Interrupt
;*********************  主程序  *************************/
F_Main:
    MOV     SP, #STACK_POIRTER
    MOV     PSW, #0
    USING   0                           ;选择第 0 组 R0~R7
;================  用户初始化程序  ====================================
    LCALL   F_PCA_Init                  ;PCA 初始化
    SETB    EA
    ANL     P2M1, #NOT 0E0H             ; P2.7 P2.6 P2.5  设置为推挽输出
    ORL     P2M0, #0E0H
;==================  主循环  ==================================
L_MainLoop:
    MOV     R7, #2
    LCALL   F_delay_ms

    MOV     A, pwm0_L               ; if(++pwm0 >= PWM_HIGH_MAX) pwm0 = PWM_HIGH_MIN
    ADD     A, #1
    MOV     pwm0_L, A
    MOV     A, pwm0_H
    ADDC    A, #0
    MOV     pwm0_H, A
    MOV     A, pwm0_L
    CLR     C
    SUBB    A, #LOW PWM_HIGH_MAX
    MOV     A, pwm0_H
    SUBB    A, #HIGH PWM_HIGH_MAX
    JC      L_PWM0_NotOverFollow
    MOV     pwm0_H, #HIGH PWM_HIGH_MIN
    MOV     pwm0_L, #LOW PWM_HIGH_MIN

L_PWM0_NotOverFollow:
    MOV     R5, #PCA0
    MOV     R6, pwm0_H
    MOV     R7, pwm0_L
```

```
    LCALL   F_PWMn_Update

    MOV     A, pwm1_L               ; if(++pwm1 >= PWM_HIGH_MAX) pwm1 = PWM_HIGH_MIN
    ADD     A, #1
    MOV     pwm1_L, A
    MOV     A, pwm1_H
    ADDC    A, #0
    MOV     pwm1_H, A
    MOV     A, pwm1_L
    CLR     C
    SUBB    A, #LOW PWM_HIGH_MAX
    MOV     A, pwm1_H
    SUBB    A, #HIGH PWM_HIGH_MAX
    JC      L_PWM1_NotOverFollow
    MOV     pwm1_H, #HIGH PWM_HIGH_MIN
    MOV     pwm1_L, #LOW PWM_HIGH_MIN

L_PWM1_NotOverFollow:
    MOV     R5, #PCA1
    MOV     R6, pwm1_H
    MOV     R7, pwm1_L
    LCALL   F_PWMn_Update

    MOV     A, pwm2_L               ; if(++pwm2 >= PWM_HIGH_MAX) pwm2 = PWM_HIGH_MIN
    ADD     A, #1
    MOV     pwm2_L, A
    MOV     A, pwm2_H
    ADDC    A, #0
    MOV     pwm2_H, A
    MOV     A, pwm2_L
    CLR     C
    SUBB    A, #LOW PWM_HIGH_MAX
    MOV     A, pwm2_H
    SUBB    A, #HIGH PWM_HIGH_MAX
    JC      L_PWM2_NotOverFollow
    MOV     pwm2_H, #HIGH PWM_HIGH_MIN
    MOV     pwm2_L, #LOW PWM_HIGH_MIN

L_PWM2_NotOverFollow:
    MOV     R5, #PCA2
    MOV     R6, pwm2_H
    MOV     R7, pwm2_L
    LCALL   F_PWMn_Update
    LJMP    L_MainLoop
;//==========================================================================
;// 函数: F_delay_ms
```

```
;// 描述: 延时子程序。
;// 参数: R7: 延时 ms 数.
;// 返回: none.
;// 备注: 除了 ACCC 和 PSW 外, 所用到的通用寄存器都入栈
;//=========================================================================
F_delay_ms:
    PUSH    AR3                 ;入栈 R3
    PUSH    AR4                 ;入栈 R4

L_delay_ms_1:
    MOV     R3, #HIGH (Fosc_KHZ / 13)
    MOV     R4, #LOW (Fosc_KHZ / 13)

L_delay_ms_2:
    MOV     A, R4               ;1T Total 13T/loop
    DEC     R4                  ;2T
    JNZ     L_delay_ms_3        ;4T
    DEC     R3

L_delay_ms_3:
    DEC     A                   ;1T
    ORL     A, R3               ;1T
    JNZ     L_delay_ms_2        ;4T
    DJNZ    R7, L_delay_ms_1
    POP     AR4                 ;出栈 R2
    POP     AR3                 ;出栈 R3
    RET
;=========================================================================
; 函数: F_PWMn_Update
; 描述: 更新占空比数据。
; 参数: R5: PCA 通道数。
; R6,R7: PWM 值.
; 返回: 无
;=========================================================================
F_PWMn_Update:
    PUSH    AR3
    PUSH    AR4
    CLR     C
    MOV     A, R7
    SUBB    A, #LOW PWM_HIGH_MAX
    MOV     A, R6
    SUBB    A, #HIGH PWM_HIGH_MAX
    JC      L_QuitCheckPwm_1
    MOV     R6, #HIGH PWM_HIGH_MAX
                                ; 如果写入大于最大占空比数据，强制为最大占空比。
    MOV     R7, #LOW PWM_HIGH_MAX
```

```
L_QuitCheckPwm_1:
    CLR     C
    MOV     A, R7
    SUBB    A, #LOW PWM_HIGH_MIN
    MOV     A, R6
    SUBB    A, #HIGH PWM_HIGH_MIN
    JNC     L_QuitCheckPwm_2
    MOV     R6, #HIGH PWM_HIGH_MIN
                                    ; 如果写入小于最小占空比数据，强制为最小占空比。
    MOV     R7, #LOW PWM_HIGH_MIN

L_QuitCheckPwm_2:
    CLR     C
    MOV     A, #LOW PWM_DUTY        ;计算并保存 PWM 输出低电平的 PCA 时钟脉冲个数
    SUBB    A, R7
    MOV     R4, A
    MOV     A, #HIGH PWM_DUTY
    SUBB    A, R6
    MOV     R3, A
    CJNE    R5, #PCA0, L_NotLoadPCA0
    CLR     CR                      ; 停止 PCA 一会， 一般不会影响 PWM。
    MOV     PWM0_high_H, R6         ; 数据装入占空比变量。
    MOV     PWM0_high_L, R7
    MOV     PWM0_low_H, R3
    MOV     PWM0_low_L, R4
    SETB    CR                      ; 启动 PCA。

L_NotLoadPCA0:
    CJNE    R5, #PCA1, L_NotLoadPCA1
    CLR     CR                      ; 停止 PCA 一会， 一般不会影响 PWM。
    MOV     PWM1_high_H, R6         ; 数据装入占空比变量。
    MOV     PWM1_high_L, R7
    MOV     PWM1_low_H, R3
    MOV     PWM1_low_L, R4
    SETB    CR                      ; 启动 PCA。

L_NotLoadPCA1:
    CJNE    R5, #PCA2, L_NotLoadPCA2
    CLR     CR                      ; 停止 PCA 一会， 一般不会影响 PWM。
    MOV     PWM2_high_H, R6         ; 数据装入占空比变量。
    MOV     PWM2_high_L, R7
    MOV     PWM2_low_H, R3
    MOV     PWM2_low_L, R4
    SETB    CR                      ; 启动 PCA。
```

```
L_NotLoadPCA2:
    POP     AR4
    POP     AR3
RET
;========================================================================
; 函数: F_PCA_Init
; 描述: PCA 初始化程序.
; 参数: none
; 返回: none.
;========================================================================
F_PCA_Init:
    CLR     CR
    MOV     CH, #0
    MOV     CL, #0
    MOV     A, AUXR1
    ANL     A, # NOT(3 SHL 4)
    ORL     A, #PCA_P24_P25_P26_P27     ;切换 I/O 口
    MOV     AUXR1, A
    ANL     A, #NOT(7 SHL 1)
    ORL     A, #PCA_Clock_1T            ;选择时钟源
    MOV     CMOD, A
    MOV     CCAPM0, #(PCA_Mode_HighPulseOutput OR ENABLE)
                                        ; 16 位软件定时、高速脉冲输出、中断模式
    MOV     CCAPM1, #(PCA_Mode_HighPulseOutput OR ENABLE)
                                        ; 16 位软件定时、高速脉冲输出、中断模式
    MOV     CCAPM2, #(PCA_Mode_HighPulseOutput OR ENABLE)
                                        ; 16 位软件定时、高速脉冲输出、中断模式
    MOV     pwm0_H, #HIGH (PWM_DUTY / 4 * 1)        ;给 PWM 一个初值
    MOV     pwm0_L, #LOW (PWM_DUTY / 4 * 1)
    MOV     pwm1_H, #HIGH (PWM_DUTY / 4 * 2)
    MOV     pwm1_L, #LOW (PWM_DUTY / 4 * 2)
    MOV     pwm2_H, #HIGH (PWM_DUTY / 4 * 3)
    MOV     pwm2_L, #LOW (PWM_DUTY / 4 * 3)
    MOV     R5, #PCA0
    MOV     R6, pwm0_H
    MOV     R7, pwm0_L
    LCALL   F_PWMn_Update
    MOV     R5, #PCA1
    MOV     R6, pwm1_H
    MOV     R7, pwm1_L
    LCALL   F_PWMn_Update
    MOV     R5, #PCA2
    MOV     R6, pwm2_H
    MOV     R7, pwm2_L
    LCALL   F_PWMn_Update
    SETB    PPCA                        ; 高优先级中断
```

```
    SETB    CR                      ; 运行 PCA 定时器
RET
;=========================================================================
; 函数: F_PCA_Interrupt
; 描述: PCA 中断处理程序.
; 参数: None
; 返回: none.
;=========================================================================
F_PCA_Interrupt:
    PUSH    PSW
    PUSH    ACC
;=============== PCA 模块 0 中断 ===================
    JNB     CCF0, L_QuitPCA0        ; PCA 模块 0 中断
    CLR     CCF0                    ; 清 PCA 模块 0 中断标志
    JNB     P25, L_PCA0_LoadLow
    MOV     A, CCAP0L               ; 输出为高电平，则给影射寄存器装载高电平时间长度
    ADD     A, PWM0_high_L          ; 加上高电平时间，
    MOV     CCAP0L, A               ; 先写 CCAP0L
    MOV     A, CCAP0H
    ADDC    A, PWM0_high_H
    MOV     CCAP0H, A               ; 后写 CCAP0H
    SJMP    L_QuitPCA0

L_PCA0_LoadLow:
    MOV     A, CCAP0L               ; 输出为低电平，则给影射寄存器装载低电平时间长度
    ADD     A, PWM0_low_L           ; 加上低电平时间，
    MOV     CCAP0L, A               ; 先写 CCAP0L
    MOV     A, CCAP0H
    ADDC    A, PWM0_low_H
    MOV     CCAP0H, A               ; 后写 CCAP0H

L_QuitPCA0:
;=============== PCA 模块 1 中断 ===================
    JNB     CCF1, L_QuitPCA1        ; PCA 模块 1 中断
    CLR     CCF1                    ; 清 PCA 模块 1 中断标志
    JNB     P26, L_PCA1_LoadLow
    MOV     A, CCAP1L               ; 输出为高电平，则给影射寄存器装载高电平时间长度
    ADD     A, PWM1_high_L          ; 加上高电平时间，
    MOV     CCAP1L, A               ; 先写 CCAP1L
    MOV     A, CCAP1H
    ADDC    A, PWM1_high_H
    MOV     CCAP1H, A               ; 后写 CCAP1H
    SJMP    L_QuitPCA1

L_PCA1_LoadLow:
    MOV     A, CCAP1L               ; 输出为低电平，则给影射寄存器装载低电平时间长度
```

```
    ADD     A, PWM1_low_L           ; 加上低电平时间，
    MOV     CCAP1L, A               ; 先写 CCAP1L
    MOV     A, CCAP1H
    ADDC    A, PWM1_low_H
    MOV     CCAP1H, A               ; 后写 CCAP1H

L_QuitPCA1:
;=============== PCA 模块 2 中断 ===================
    JNB     CCF2, L_QuitPCA2        ; PCA 模块 2 中断
    CLR     CCF2                    ; 清 PCA 模块 2 中断标志
    JNB     P27, L_PCA2_LoadLow
    MOV     A, CCAP2L               ; 输出为高电平，则给影射寄存器装载高电平时间长度
    ADD     A, PWM2_high_L          ; 加上高电平时间，
    MOV     CCAP2L, A               ; 先写 CCAP2L
    MOV     A, CCAP2H
    ADDC    A, PWM2_high_H
    MOV     CCAP2H, A               ; 后写 CCAP2H
    SJMP    L_QuitPCA2

L_PCA2_LoadLow:
    MOV     A, CCAP2L               ; 输出为低电平，则给影射寄存器装载低电平时间长度
    ADD     A, PWM2_low_L           ; 加上低电平时间，
    MOV     CCAP2L, A               ; 先写 CCAP2L
    MOV     A, CCAP2H
    ADDC    A, PWM2_low_H
    MOV     CCAP2H, A               ; 后写 CCAP2H

L_QuitPCA2:
    POP     ACC
    POP     PSW
    RETI

END
```

22.9 用 CCP/PCA 的 16 位捕获模式测脉冲宽度的程序(C 和汇编)

1.C 程序:

```
/*----STC15F2K60S2 系列 PCA 实现 16 位捕获举例-------------------------------------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可--------*/
//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"
#include "intrins.h"

#define     FOSC                18432000L

typedef     unsigned char       BYTE;
typedef     unsigned int        WORD;
typedef     unsigned long       DWORD;

sfr         P_SW1 = 0xA2;       //外设功能切换寄存器 1

#define     CCP_S0 0x10         //P_SW1.4
#define     CCP_S1 0x20         //P_SW1.5

sfr         CCON = 0xD8;        //PCA 控制寄存器

sbit        CCF0 = CCON^0;      //PCA 模块 0 中断标志
sbit        CCF1 = CCON^1;      //PCA 模块 1 中断标志
sbit        CR = CCON^6;        //PCA 定时器运行控制位
sbit        CF = CCON^7;        //PCA 定时器溢出标志

sfr         CMOD = 0xD9;        //PCA 模式寄存器
sfr         CL = 0xE9;          //PCA 定时器低字节
sfr         CH = 0xF9;          //PCA 定时器高字节
sfr         CCAPM0 = 0xDA;      //PCA 模块 0 模式寄存器
sfr         CCAP0L = 0xEA;      //PCA 模块 0 捕获寄存器 LOW
sfr         CCAP0H = 0xFA;      //PCA 模块 0 捕获寄存器 HIGH
sfr         CCAPM1 = 0xDB;      //PCA 模块 1 模式寄存器
sfr         CCAP1L = 0xEB;      //PCA 模块 1 捕获寄存器 LOW
sfr         CCAP1H = 0xFB;      //PCA 模块 1 捕获寄存器 HIGH
sfr         CCAPM2 = 0xDC;      //PCA 模块 2 模式寄存器
sfr         CCAP2L = 0xEC;      //PCA 模块 2 捕获寄存器 LOW
sfr         CCAP2H = 0xFC;      //PCA 模块 2 捕获寄存器 HIGH
sfr         PCA_PWM0 = 0xf2;    //PCA 模块 0 的 PWM 寄存器
sfr         PCA_PWM1 = 0xf3;    //PCA 模块 1 的 PWM 寄存器
sfr         PCA_PWM2 = 0xf4;    //PCA 模块 2 的 PWM 寄存器

BYTE        cnt;                //存储 PCA 计时溢出次数
```

```
DWORD    count0;                    //记录上一次的捕获值
DWORD    count1;                    //记录本次的捕获值
DWORD    length;                    //存储信号的时间长度(count1 - count0)

void main()
{
    ACC = P_SW1;
    ACC &= ~(CCP_S0 | CCP_S1);      //CCP_S0=0 CCP_S1=0
    P_SW1 = ACC;                    //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)
//  ACC = P_SW1;

//  ACC &= ~(CCP_S0 | CCP_S1);      //CCP_S0=1 CCP_S1=0
//  ACC |= CCP_S0;                  //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
//  P_SW1 = ACC;
//
//  ACC = P_SW1;
//  ACC &= ~(CCP_S0 | CCP_S1);      //CCP_S0=0 CCP_S1=1
//  ACC |= CCP_S1;                  //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
//  P_SW1 = ACC;
    CCON = 0;                       //初始化 PCA 控制寄存器
                                    //PCA 定时器停止
                                    //清除 CF 标志
                                    //清除模块中断标志
    CL = 0;                         //复位 PCA 寄存器
    CH = 0;
    CCAP0L = 0;
    CCAP0H = 0;
    CMOD = 0x09;                    //设置 PCA 时钟源为系统时钟,且使能 PCA 计时溢出中断
    CCAPM0= 0x21;                   //PCA 模块 0 为 16 位捕获模式(上升沿捕获,
                                    //可测从高电平开始的整个周期),且产生捕获中断
//  CCAPM0= 0x11;                   //PCA 模块 0 为 16 位捕获模式(下降沿捕获,
                                    //可测从低电平开始的整个周期),且产生捕获中断
//  CCAPM0= 0x31;                   //PCA 模块 0 为 16 位捕获模式(上升沿/下降沿捕获,
                                    //可测高电平或者低电平宽度),且产生捕获中断
    CR = 1;                         //PCA 定时器开始工作
    EA = 1;
    cnt = 0;

    count0 = 0;
    count1 = 0;
    while (1);
}

void PCA_isr() interrupt 7 using 1
{
    if (CF)
```

```
    {
        CF = 0;
        cnt++;                          //PCA 计时溢出次数+1
    }
    if (CCF0)
    {
        CCF0 = 0;
        count0 = count1;                //备份上一次的捕获值
        ((BYTE *)&count1)[3] = CCAP0L;  //保存本次的捕获值
        ((BYTE *)&count1)[2] = CCAP0H;
        ((BYTE *)&count1)[1] = cnt;
        ((BYTE *)&count1)[0] = 0;
        length = count1 - count0;       //计算两次捕获的差值,即得到时间长度
    }
}
```

2. 汇编程序：

```
/*----STC15F2K60S2 系列 PCA 实现 16 位捕获举例----------------------------------------*/
/*-----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-----------*/
;假定测试芯片的工作频率为 18.432MHz
P_SW1       EQU   0A2H       ;外设功能切换寄存器 1
CCP_S0      EQU   10H        ;P_SW1.4

CCP_S1      EQU   20H        ;P_SW1.5
CCON        EQU   0D8H       ;PCA 控制寄存器

CCF0        BIT   CCON.0     ;PCA 模块 0 中断标志
CCF1        BIT   CCON.1     ;PCA 模块 1 中断标志
CR          BIT   CCON.6     ;PCA 定时器运行控制位
CF          BIT   CCON.7     ;PCA 定时器溢出标志
CMOD        EQU   0D9H       ;PCA 模式寄存器
CL          EQU   0E9H       ;PCA 定时器低字节
CH          EQU   0F9H       ;PCA 定时器高字节

CCAPM0      EQU   0DAH       ;PCA 模块 0 模式寄存器
CCAP0L      EQU   0EAH       ;PCA 模块 0 捕获寄存器 LOW
CCAP0H      EQU   0FAH       ;PCA 模块 0 捕获寄存器 HIGH
CCAPM1      EQU   0DBH       ;PCA 模块 1 模式寄存器
CCAP1L      EQU   0EBH       ;PCA 模块 1 捕获寄存器 LOW
CCAP1H      EQU   0FBH       ;PCA 模块 1 捕获寄存器 HIGH
CCAPM2      EQU   0DCH       ;PCA 模块 2 模式寄存器
CCAP2L      EQU   0ECH       ;PCA 模块 2 捕获寄存器 LOW
CCAP2H      EQU   0FCH       ;PCA 模块 2 捕获寄存器 HIGH

PCA_PWM0    EQU   0F2H       ;PCA 模块 0 的 PWM 寄存器
```

```
PCA_PWM1    EQU    0F3H        ;PCA 模块 1 的 PWM 寄存器
PCA_PWM2    EQU    0F4H        ;PCA 模块 2 的 PWM 寄存器

CNT         EQU    30H         ;存储 PCA 计时溢出次数
COUNT0      EQU    31H         ;记录上一次的捕获值,3 字节
COUNT1      EQU    34H         ;记录本次的捕获值,3 字节
LENGTH      EQU    37H         ;存储信号的时间长度,3 字节(count1 - count0)

    ORG    0000H
    LJMP   START
    ORG    003BH

PCA_ISR:
    PUSH   PSW
    PUSH   ACC
    JNB    CF, CKECK_CCF0      ;判断是否为 PCA 计时溢出中断
    CLR    CF
    INC    CNT                 ;PCA 计时溢出次数+1

CKECK_CCF0:
    JNB    CCF0, PCA_ISR_EXIT  ;判断是否为捕获中断
    CLR    CCF0
    MOV    COUNT0, COUNT1      ;备份上一次的捕获值
    MOV    COUNT0+1, COUNT1+1
    MOV    COUNT0+2, COUNT1+2
    MOV    COUNT1, CNT         ;保存本次的捕获值
    MOV    COUNT1+1, CCAP0H
    MOV    COUNT1+2, CCAP0L
    CLR    C                   ;计算两次捕获的差值
    MOV    A, COUNT1+2
    SUBB   A, COUNT0+2
    MOV    LENGTH+2, A
    MOV    A, COUNT1+1
    SUBB   A, COUNT0+1
    MOV    LENGTH+1, A
    MOV    A, COUNT1
    SUBB   A, COUNT0
    MOV    LENGTH, A           ;LENGTH 存放的即为时间长度

PCA_ISR_EXIT:
    POP    ACC
    POP    PSW
    RETI
;---------------------------------------

    ORG    0100H
```

```
START:
    MOV     SP, #5FH
    MOV     A, P_SW1

    ANL     A, #0CFH                ;CCP_S0=0 CCP_S1=0
    MOV     P_SW1, A                ;(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)

;   MOV     A, P_SW1
;   ANL     A, #0CFH                ;CCP_S0=1 CCP_S1=0
;   ORL     A, #CCP_S0              ;(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
;   MOV     P_SW1, A

;   MOV     A, P_SW1
;   ANL     A, #0CFH                ;CCP_S0=0 CCP_S1=1
;   ORL     A, #CCP_S1              ;(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
;   MOV     P_SW1, A

    MOV     CCON, #0                ;初始化 PCA 控制寄存器
                                    ;PCA 定时器停止
                                    ;清除 CF 标志
                                    ;清除模块中断标志
    CLR     A

    MOV     CL, A                   ;复位 PCA 计时器
    MOV     CH, A
    MOV     CCAP0L, A
    MOV     CCAP0H, A
    MOV     CMOD, #09H              ;设置 PCA 时钟源为系统时钟
                                    ;使能 PCA 定时器溢出中断
    MOV     CCAPM0, #21H            ;PCA 模块 0 为 16 位捕获模式(上升沿捕获,
                                    ;可测从高电平开始的整个周期),且产生捕获中断
    MOV     CCAPM0, #11H            ;PCA 模块 0 为 16 位捕获模式(下降沿捕获,
                                    ;可测从低电平开始的整个周期),且产生捕获中断
    MOV     CCAPM0, #31H            ;PCA 模块 0 为 16 位捕获模式(上升沿/下降沿捕获,
                                    ;可测高电平或者低电平宽度),且产生捕获中断

    SETB    CR                      ;PCA 定时器开始工作
    SETB    EA

    CLR     A                       ;初始化变量
    MOV     CNT, A
    MOV     COUNT0, A
    MOV     COUNT0+1, A
    MOV     COUNT0+2, A
    MOV     COUNT1, A
    MOV     COUNT1+1, A
```

```
    MOV     COUNT1+2, A
    MOV     LENGTH, A
    MOV     LENGTH+1, A
    MOV     LENGTH+2, A
    SJMP    $
;----------------------------------------
END
```

22.10 用 T0 软硬结合模拟 16 路软件 PWM 的程序(C 及汇编)

1.C 程序:

```
/*----STC 1T Series MCU Demo Programme----------------*/
/*********************功能说明**********************
使用 Timer0 模拟 16 通道 PWM 驱动程序。
输出为 P1.0 ~ P1.7，P2.0 ~ P2.7，对应 PWM0 ~ PWM15。
定时器中断频率一般不要超过 100KHz，留足够的时间给别的程序运行。
本例子使用 22.1184MHz 时钟，25K 的中断频率，250 级 PWM，周期为 10ms。
中断里处理的时间不超过 6us，占 CPU 时间大约为 15%。
*****************************************************/
#include <reg52.h>

#define   MAIN_Fosc       22118400UL          //定义主时钟
#define   Timer0_Rate     25000               //中断频率

typedef   unsigned char   u8;
typedef   unsigned int    u16;
typedef   unsigned long   u32;

sfr       AUXR = 0x8E;

#define   Timer0_Reload   (65536UL -(MAIN_Fosc / Timer0_Rate))    //Timer 0  重装值
//**************PWM8  变量和常量以及 I/O 口定义  ***************
//*********************8 通道 8 bit 软 PWM*********************
#define   PWM_DUTY_MAX    250                 // 0~255 PWM 周期, 最大 255
#define   PWM_ON          1                   //  定义占空比的电平, 1  或  0

#define   PWM_OFF         (!PWM_ON)
#define   PWM_ALL_ON      (0xff * PWM_ON)

u8        bdata PWM_temp1, PWM_temp2;         //影射一个 RAM，可位寻址，输出时同步刷新

sbit      P_PWM0 = PWM_temp1^0;               //  定义影射 RAM 每位对应的 IO
sbit      P_PWM1 = PWM_temp1^1;
sbit      P_PWM2 = PWM_temp1^2;
```

```
sbit    P_PWM3 = PWM_temp1^3;
sbit    P_PWM4 = PWM_temp1^4;
sbit    P_PWM5 = PWM_temp1^5;
sbit    P_PWM6 = PWM_temp1^6;
sbit    P_PWM7 = PWM_temp1^7;
sbit    P_PWM8 = PWM_temp2^0;
sbit    P_PWM9 = PWM_temp2^1;
sbit    P_PWM10 = PWM_temp2^2;
sbit    P_PWM11 = PWM_temp2^3;
sbit    P_PWM12 = PWM_temp2^4;
sbit    P_PWM13 = PWM_temp2^5;
sbit    P_PWM14 = PWM_temp2^6;
sbit    P_PWM15 = PWM_temp2^7;

u8      pwm_duty;                       //周期计数值
u8      pwm[16];                        //pwm0~pwm15  为 0 至 15 路 PWM 的宽度值

bit     B_1ms;

u8      cnt_1ms;
u8      cnt_20ms;
/*********************************************/
void main(void)
{
    u8 i;
    AUXR |= (1<<7);                     // Timer0 set as 1T mode
    TMOD &= ~(1<<2);                    // Timer0 set as Timer
    TMOD &= ~0x03;                      // Timer0 set as 16 bits Auto Reload
    TH0 = Timer0_Reload / 256;          //Timer0 Load
    TL0 = Timer0_Reload % 256;
    ET0 = 1;                            //Timer0 Interrupt Enable
    PT0 = 1;                            //高优先级
    TR0 = 1;                            //Timer0 Run
    EA = 1;                             //打开总中断
    cnt_1ms = Timer0_Rate / 1000;       //1ms 计数
    cnt_20ms = 20;

    for(i=0; i<16; i++) pwm[i] = i * 15 + 15;       //给 PWM 一个初值
    while(1)
    {
        if(B_1ms)                       //1ms 到
        {
            B_1ms = 0;
            if(--cnt_20ms == 0)         //PWM 20ms 改变一阶
            {
                cnt_20ms = 20;
```

```
                    for(i=0; i<16; i++) pwm[i]++;
                }
            }
        }
}
/********************** Timer0 1ms 中断函数  ************************/
void timer0 (void) interrupt 1
{
    P1 = PWM_temp1;                         //影射 RAM 输出到实际的 PWM 端口
    P2 = PWM_temp2;
    if(++pwm_duty == PWM_DUTY_MAX)          //PWM 周期结束，重新开始新的周期
    {
        pwm_duty = 0;
        PWM_temp1 = PWM_ALL_ON;
        PWM_temp2 = PWM_ALL_ON;
    }
    ACC = pwm_duty;

    if(ACC == pwm[0]) P_PWM0 = PWM_OFF;     //判断 PWM 占空比是否结束
    if(ACC == pwm[1]) P_PWM1 = PWM_OFF;
    if(ACC == pwm[2]) P_PWM2 = PWM_OFF;
    if(ACC == pwm[3]) P_PWM3 = PWM_OFF;
    if(ACC == pwm[4]) P_PWM4 = PWM_OFF;
    if(ACC == pwm[5]) P_PWM5 = PWM_OFF;
    if(ACC == pwm[6]) P_PWM6 = PWM_OFF;
    if(ACC == pwm[7]) P_PWM7 = PWM_OFF;
    if(ACC == pwm[8]) P_PWM8 = PWM_OFF;
    if(ACC == pwm[9]) P_PWM9 = PWM_OFF;
    if(ACC == pwm[10]) P_PWM10 = PWM_OFF;
    if(ACC == pwm[11]) P_PWM11 = PWM_OFF;
    if(ACC == pwm[12]) P_PWM12 = PWM_OFF;
    if(ACC == pwm[13]) P_PWM13 = PWM_OFF;
    if(ACC == pwm[14]) P_PWM14 = PWM_OFF;
    if(ACC == pwm[15]) P_PWM15 = PWM_OFF;
    if(--cnt_1ms == 0)
    {
        cnt_1ms = Timer0_Rate / 1000;
        B_1ms = 1;                          // 1ms 标志
    }
}
```

2. 汇编程序：

```
/*----STC 1T Series MCU Demo Programme----------------*/
/*********************功能说明**********************
```

```
;******************** 功能说明 ********************
;使用 Timer0 模拟 16 通道 PWM 驱动程序。
;输出为 P1.0 ~ P1.7, P2.0 ~ P2.7, 对应 PWM0 ~ PWM15.
;定时器中断频率一般不要超过 100KHZ, 留足够的时间给别的程序运行.
;本例子使用 22.1184MHZ 时钟, 25K 的中断频率, 250 级 PWM, 周期为 10ms.
;中断里处理的时间不超过 6us, 占 CPU 时间大约为 15%.
;*****************************************
Fosc_KHZ        EQU     22118       ; 定义主时钟 KHz
Timer0_Rate     EQU     25          ; 中断频率, KHz
PWM_DUTY_MAX    EQU     250         ; 0~255 PWM 周期, 最大 255

AUXR            DATA    08EH
Timer0_Reload   EQU     (65536 - (Fosc_KHZ / Timer0_Rate))      ;Timer 0 重装值
;************** PWM8 变量和常量以及 I/O 口定义 ***************
;********************* 8 通道 8 bit 软 PWM*********************
PWM_temp1       DATA    20H         ; 影射一个 RAM，可位寻址，输出时同步刷新
PWM_temp2       DATA    21H         ; 影射一个 RAM，可位寻址，输出时同步刷新
P_PWM0          BIT     PWM_temp1.0 ; 定义影射 RAM 每位对应的 I/O
P_PWM1          BIT     PWM_temp1.1
P_PWM2          BIT     PWM_temp1.2
P_PWM3          BIT     PWM_temp1.3
P_PWM4          BIT     PWM_temp1.4
P_PWM5          BIT     PWM_temp1.5
P_PWM6          BIT     PWM_temp1.6
P_PWM7          BIT     PWM_temp1.7
P_PWM8          BIT     PWM_temp2.0
P_PWM9          BIT     PWM_temp2.1
P_PWM10         BIT     PWM_temp2.2
P_PWM11         BIT     PWM_temp2.3
P_PWM12         BIT     PWM_temp2.4
P_PWM13         BIT     PWM_temp2.5
P_PWM14         BIT     PWM_temp2.6
P_PWM15         BIT     PWM_temp2.7

B_1ms           BIT     22H.0

cnt_1ms         DATA    30H
cnt_20ms        DATA    31H

pwm_duty        DATA    32H         ; 周期计数值
pwm             EQU     40H         ; 40H ~ 4FH 为 0 至 15 路 PWM 的宽度值

STACK_POIRTER   EQU     0D0H        ;堆栈开始地址
;*******************************************************************
;*******************************************************************
    ORG     00H                     ;reset
```

```
    LJMP    F_Main
    ORG     0BH                             ;1 Timer0 interrupt
    LJMP    F_Timer0_Interrupt
;********************  主程序  ***************************/
F_Main:
    MOV     SP, #STACK_POIRTER
    MOV     PSW, #0
    USING   0                               ;选择第 0 组 R0~R7
;================  用户初始化程序  ==================================
    ORL     AUXR, #(1<<7)                   ; Timer0 set as 1T mode
    ANL     TMOD, #NOT (1 SHL 2)            ; Timer0 set as Timer
    ANL     TMOD, #NOT 0x03                 ; Timer0 set as 16 bits Auto Reload

    MOV     TH0, #HIGH Timer0_Reload        ; Timer0 Load
    MOV     TL0, #LOW Timer0_Reload

    SETB    ET0                             ; Timer0 Interrupt Enable
    SETB    PT0                             ;  高优先级
    SETB    TR0                             ; Timer0 Run
    SETB    EA                              ;  打开总中断

    MOV     cnt_1ms, #(Timer0_Rate / 1000)  ; 1ms 计数
    MOV     cnt_20ms, #20
    MOV     R0, #pwm
    MOV     R2, #15

L_DefaultPWM:
    MOV     A, R2
    MOV     @R0, A                          ;给 PWM 一个初值
    ADD     A, #15
    MOV     R2, A
    INC     R0
    CJNE    R0, #(pwm+16), L_DefaultPWM
;==================  主循环  ==================================
L_MainLoop:
    JNB     B_1ms, $                        ;  等待 1ms 到
    CLR     B_1ms ;
    DJNZ    cnt_20ms, L_MainLoop
    MOV     cnt_20ms, #20                   ; PWM 20ms 改变一阶
    MOV     R0, #pwm

L_SetPWM_Loop:
    INC     @R0                             ; pwm + 1
    INC     R0
    CJNE    R0, #(pwm+16), L_SetPWM_Loop
    SJMP    L_MainLoop
```

```
;********************** Timer0 1ms 中断函数 **********************/
F_Timer0_Interrupt:
    PUSH    PSW
    PUSH    ACC

    MOV     P1, PWM_temp1               ; 影射 RAM 输出到实际的 PWM 端口
    MOV     P2, PWM_temp2

    INC     pwm_duty
    MOV     A, pwm_duty
    CJNE    A, #PWM_DUTY_MAX, L_CheckPwm
    MOV     pwm_duty, #0                ; PWM 周期结束，重新开始新的周期
    CLR     A
    MOV     PWM_temp1, #0FFH
    MOV     PWM_temp2, #0FFH

L_CheckPwm:
    CJNE    A, pwm+0, $+5               ;判断 PWM 占空比是否结束
    CLR     P_PWM0
    CJNE    A, pwm+1, $+5
    CLR     P_PWM1
    CJNE    A, pwm+2, $+5
    CLR     P_PWM2
    CJNE    A, pwm+3, $+5
    CLR     P_PWM3
    CJNE    A, pwm+4, $+5
    CLR     P_PWM4
    CJNE    A, pwm+5, $+5
    CLR     P_PWM5
    CJNE    A, pwm+6, $+5
    CLR     P_PWM6
    CJNE    A, pwm+7, $+5
    CLR     P_PWM7
    CJNE    A, pwm+8, $+5
    CLR     P_PWM8
    CJNE    A, pwm+9, $+5
    CLR     P_PWM9
    CJNE    A, pwm+10, $+5
    CLR     P_PWM10
    CJNE    A, pwm+11, $+5
    CLR     P_PWM11
    CJNE    A, pwm+12, $+5
    CLR     P_PWM12
    CJNE    A, pwm+13, $+5
    CLR     P_PWM13
    CJNE    A, pwm+14, $+5
```

```
    CLR     P_PWM14
    CJNE    A, pwm+15, $+5
    CLR     P_PWM15

    DJNZ    cnt_1ms, L_QuitCheck_1ms
    MOV     cnt_1ms, #Timer0_Rate
    SETB    B_1ms                       ; 1ms 标志

L_QuitCheck_1ms:
    POP     ACC
    POP     PSW

    RETI

    END
```

22.11 用 T0 的时钟输出功能实现 8~16 位 PWM 的程序(C 及汇编)

----占用系统时间小于 0.4%

下文为利用 T0 的时钟输出功能实现 8 ~ 16 位 PWM(占用系统时间小于 0.4%)的测试程序(包括 C 语言程序和汇编程序)。

1、C 语言程序

```
/*----STC 1T Series MCU RC Demo----*/
#include <reg52.h>
/*************************功能说明*************************
本程序演示使用定时器做软件 PWM。
定时器 0 做 16 位自动重装，中断，从 T0CLKO 高速输出 PWM。
本例程是使用 STC15F/L 系列 MCU 的定时器 T0 做模拟 PWM 的例程。
PWM 可以是任意的量程。但是由于软件重装需要一点时间，所以 PWM 占空比最小为 32T/周期，最大为(周期-32T)周期，T 为时钟周期。
PWM 频率为周期的倒数。假如周期为 6000，使用 24MHz 的主频，则 PWM 频率为 4000Hz。
*****************************************************************/
#define   MAIN_Fosc             24000000UL  //定义主时钟
#define   PWM_DUTY              6000        //定义 PWM 的周期，数值为时钟周期数，假如使用
                                            //24.576MHZ 的主频，则 PWM 频率为 6000HZ。
#define   PWM_HIGH_MIN          32          //限制 PWM 输出的最小占空比。用户请勿修改。

#define   PWM_HIGH_MAX          (PWM_DUTY-PWM_HIGH_MIN)
                                            //限制 PWM 输出的最大占空比。用户请勿修改。
typedef   unsigned char         u8;
typedef   unsigned int          u16;
typedef   unsigned long         u32;

sfr       P3M1 = 0xB1;                      //P3M1.n,P3M0.n =00--->Standard, 01--->push-pull
```

```
sfr         P3M0 = 0xB2;                    // =10--->pure input, 11--->open drain
sfr         AUXR = 0x8E;
sfr         INT_CLKO = 0x8F;

sbit        P_PWM = P3^5;                   //定义 PWM 输出引脚。
//sbit      P_PWM = P1^4;                   //定义 PWM 输出引脚。  STC15W204S

u16         pwm;                            //定义 PWM 输出高电平的时间的变量。用户操作 PWM 的变量。
u16         PWM_high,PWM_low;               //中间变量，用户请勿修改。

void delay_ms(unsigned char ms);
void LoadPWM(u16 i);
/*********************  主函数**************************/
void main(void)
{
    P_PWM = 0;
    P3M1 &= ~(1 << 5);                      //P3.5  设置为推挽输出
    P3M0 |= (1 << 5);

//  P1M1 &= ~(1 << 4);                      //P1.4  设置为推挽输出  STC15W204S
//  P1M0 |= (1 << 4);

    TR0 = 0;                                //停止计数
    ET0 = 1;                                //允许中断
    PT0 = 1;                                //高优先级中断
    TMOD &= ~0x03;                          //工作模式,0: 16 位自动重装
    AUXR |= 0x80;                           //1T
    TMOD &= ~0x04;                          //定时
    INT_CLKO |= 0x01;                       //输出时钟

    TH0 = 0;
    TL0 = 0;
    TR0 = 1;                                //开始运行

    EA = 1;

    pwm = PWM_DUTY / 10;                    //给 PWM 一个初值，这里为 10%占空比
    LoadPWM(pwm);                           //计算 PWM 重装值

    while (1)
    {
        while(pwm < (PWM_HIGH_MAX-8))
        {
            pwm += 8;                       //PWM 逐渐加到最大
            LoadPWM(pwm);
            delay_ms(8);
```

```
            }
            while(pwm > (PWM_HIGH_MIN+8))
            {
                pwm -= 8;                       //PWM 逐渐减到最小
                LoadPWM(pwm);
                delay_ms(8);
            }
        }
}
//========================================================================
// 函数: void delay_ms(unsigned char 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.
//========================================================================
void delay_ms(unsigned char ms)
{
    unsigned int i;
    do   {
            i = MAIN_Fosc / 13000;
            while(--i) ;
        }while(--ms);
}
/**************** 计算 PWM 重装值函数 ******************/
void LoadPWM(u16 i)
{
    u16 j;

    if(i > PWM_HIGH_MAX) i = PWM_HIGH_MAX;
                                            //如果写入大于最大占空比数据，则强制为最大占空比。
    if(i < PWM_HIGH_MIN) i = PWM_HIGH_MIN;
                                            //如果写入小于最小占空比数据，则强制为最小占空比。
    j = 65536UL - PWM_DUTY + i;             //计算 PWM 低电平时间
    i = 65536UL - i;                        //计算 PWM 高电平时间
    EA = 0;
    PWM_high = i;                           //装载 PWM 高电平时间
    PWM_low = j;                            //装载 PWM 低电平时间
    EA = 1;
}
/********************** Timer0 中断函数************************/
void timer0_int (void) interrupt 1
{
    if(P_PWM)
    {
        TH0 = (u8)(PWM_low >> 8);           //如果是输出高电平，则装载低电平时间。
        TL0 = (u8)PWM_low;
```

```
    }
    else
    {
        TH0 = (u8)(PWM_high >> 8);          //如果是输出低电平，则装载高电平时间。
        TL0 = (u8)PWM_high;
    }
    }
```

2、汇编语言程序

```
/*----STC 1T Series MCU RC Demo----*/
;/************* 功能说明 **************
;本程序演示使用定时器做软件 PWM。
;定时器 0 做 16 位自动重装，中断，从 T0CLKO 高速输出 PWM。
;本例程是使用 STC15F/L 系列 MCU 的定时器 T0 做模拟 PWM 的例程。
;PWM 可以是任意的量程。但是由于软件重装需要一点时间，所以 PWM 占空比最小为 32T/周期，最大为(周期-32T)/周期, T 为时钟周期。
;PWM 频率为周期的倒数。假如周期为 6000，使用 24MHz 的主频，则 PWM 频率为 4000Hz。
;*****************************************/
;**** *****************用户宏定义*****************************
Fosc_KHZ        EQU     24000   //定义主时钟, KHZ
PWM_DUTY        EQU     6000    //定义 PWM 的周期，数值为 PCA 所选择的时钟脉冲个数。

PWM_HIGH_MIN    EQU     32      //限制 PWM 输出的最小占空比，避免中断里重装参数时间不够。
PWM_HIGH_MAX    EQU     (PWM_DUTY - PWM_HIGH_MIN)
                        //限制 PWM 输出的最大占空比。
;************************************************************
P3M1            DATA    0B1H        ; P3M1.n,P3M0.n =00--->Standard, 01--->push-pull
P3M0            DATA    0B2H        ; =10--->pure input, 11--->open drain
AUXR            DATA    08EH
INT_CLKO        DATA    08FH

P_PWM           BIT     P3.5        ; 定义 PWM 输出引脚。
;P_PWM          BIT     P1.4        ; 定义 PWM 输出引脚。 STC15W204S

pwm_H           DATA    030H        ; 定义 PWM 输出高电平的时间的变量。用户操作 PWM 的变量。
pwm_L           DATA    031H
PWM_high_H      DATA    032H        ; 中间变量，用户请勿修改
PWM_high_L      DATA    033H
PWM_low_H       DATA    034H
PWM_low_L       DATA    035H

STACK_POIRTER EQU       0D0H        ;堆栈开始地址
;************************************************************
    ORG     00H                     ;reset
```

```
    LJMP    F_Main

    ORG     0BH                 ;1 Timer0 interrupt
    LJMP    F_Timer0_Interrupt
;********************  主程序  ***************************/
F_Main:
    MOV     SP, #STACK_POIRTER
    MOV     PSW, #0
    USING   0                   ;选择第 0 组 R0~R7
;================  用户初始化程序  ==================================
    CLR     P_PWM
    ANL     P3M1, #NOT (1 SHL 5)    ; P3.5  设置为推挽输出
    ORL     P3M0, #(1 SHL 5)

;   ANL     P1M1, #NOT (1 SHL 4)    ; P1.4  设置为推挽输出  STC15W204S
;   ORL     P1M0, #(1 SHL 4)

    CLR     TR0                 ; 停止计数
    SETB    ET0                 ; 允许中断
    SETB    PT0                 ; 高优先级中断
    ANL     TMOD, #NOT 003H     ; 工作模式,0: 16 位自动重装
    ORL     AUXR, #080H         ; 1T
    ANL     TMOD, #NOT 004H     ; 定时
    ORL     INT_CLKO, #001H     ; 输出时钟

    MOV     TH0, #0
    MOV     TL0, #0
    SETB    TR0                 ; 开始运行

    SETB    EA

    MOV     pwm_H, #HIGH (PWM_DUTY / 10)    ;给 PWM 一个初值，这里为 10%占空比
    MOV     pwm_L, #LOW (PWM_DUTY / 10)
    MOV     R6, pwm_H
    MOV     R7, pwm_L
    LCALL   F_PWMn_Update       ; 计算 PWM 重装值
;==================  主循环  ==================================
L_MainLoop1:
    MOV     R7, #2
    LCALL   F_delay_ms

    MOV     A, pwm_L            ; if(++pwm >= PWM_HIGH_MAX)
    ADD     A, #1
    MOV     pwm_L, A
    MOV     A, pwm_H
    ADDC    A, #0
```

```
    MOV     pwm_H, A
    MOV     A, pwm_L
    CLR     C
    SUBB    A, #LOW PWM_HIGH_MAX
    MOV     A, pwm_H
    SUBB    A, #HIGH PWM_HIGH_MAX
    JC      L_PWM_NotUpOverFollow           ; PWM 逐渐加到最大
    MOV     pwm_H, #HIGH PWM_HIGH_MAX
    MOV     pwm_L, #LOW PWM_HIGH_MAX
    SJMP    L_MainLoop2

L_PWM_NotUpOverFollow:
    MOV     R6, pwm_H
    MOV     R7, pwm_L
    LCALL   F_PWMn_Update
    SJMP    L_MainLoop1

L_MainLoop2:
    MOV     R7, #2
    LCALL   F_delay_ms

    MOV     A, pwm_L                ; if(++pwm < PWM_HIGH_MIN)
    CLR     C
    SUBB    A, #1
    MOV     pwm_L, A
    MOV     A, pwm_H
    SUBB    A, #0
    MOV     pwm_H, A
    MOV     A, pwm_L
    CLR     C
    SUBB    A, #LOW PWM_HIGH_MIN
    MOV     A, pwm_H
    SUBB    A, #HIGH PWM_HIGH_MIN
    JNC     L_PWM_NotDnOverFollow           ; PWM 逐渐减到最小
    MOV     pwm_H, #HIGH PWM_HIGH_MIN
    MOV     pwm_L, #LOW PWM_HIGH_MIN
    SJMP    L_MainLoop1

L_PWM_NotDnOverFollow:
    MOV     R6, pwm_H
    MOV     R7, pwm_L
    LCALL   F_PWMn_Update
    SJMP    L_MainLoop2
;//==========================================================================
;// 函数: F_delay_ms
;// 描述: 延时子程序。
```

```
;// 参数: R7: 延时 ms 数.
;// 返回: none.
;// 备注: 除了 ACCC 和 PSW 外, 所用到的通用寄存器都入栈
;//=========================================================================
F_delay_ms:
    PUSH    AR3                 ;入栈 R3
    PUSH    AR4                 ;入栈 R4

L_delay_ms_1:
    MOV     R3, #HIGH (Fosc_KHZ / 13)
    MOV     R4, #LOW (Fosc_KHZ / 13)

L_delay_ms_2:
    MOV     A, R4               ;1T Total 13T/loop
    DEC     R4                  ;2T
    JNZ     L_delay_ms_3        ;4T
    DEC     R3

L_delay_ms_3:
    DEC     A                   ;1T
    ORL     A, R3               ;1T
    JNZ     L_delay_ms_2        ;4T
    DJNZ    R7, L_delay_ms_1
    POP     AR4                 ;出栈 R2
    POP     AR3                 ;出栈 R3
RET
;=========================================================================
; 函数: F_PWMn_Update
; 描述: 更新占空比数据。
; 参数: R6,R7: PWM 值。
; 返回: 无
;=========================================================================
F_PWMn_Update:
    PUSH    AR3
    PUSH    AR4
    CLR     C
    MOV     A, R7
    SUBB    A, #LOW PWM_HIGH_MAX
    MOV     A, R6
    SUBB    A, #HIGH PWM_HIGH_MAX
    JC      L_QuitCheckPwm_1
    MOV     R6, #HIGH PWM_HIGH_MAX      ; 如果写入大于最大占空比数据，强制为最大占空比。
    MOV     R7, #LOW PWM_HIGH_MAX

L_QuitCheckPwm_1:
    CLR     C
```

```
    MOV     A, R7
    SUBB    A, #LOW PWM_HIGH_MIN
    MOV     A, R6
    SUBB    A, #HIGH PWM_HIGH_MIN
    JNC     L_QuitCheckPwm_2
    MOV     R6, #HIGH PWM_HIGH_MIN      ; 如果写入小于最小占空比数据，强制为最小占空比。
    MOV     R7, #LOW PWM_HIGH_MIN

L_QuitCheckPwm_2:
    CLR     C
    MOV     A, R7                       ;计算并保存 PWM 输出低电平的 T0 时钟脉冲个数
    SUBB    A, #LOW PWM_DUTY
    MOV     R4, A
    MOV     A, R6
    SUBB    A, #HIGH PWM_DUTY
    MOV     R3, A

    CLR     C
    MOV     A, #0                       ;计算并保存 PWM 输出高电平的 T0 时钟脉冲个数
    SUBB    A, R7
    MOV     R7, A
    MOV     A, #0
    SUBB    A, R6
    MOV     R6, A

    CLR     EA                          ; 禁止一会中断， 一般不会影响 PWM。
    MOV     PWM_high_H, R6              ; 数据装入占空比变量。
    MOV     PWM_high_L, R7
    MOV     PWM_low_H, R3
    MOV     PWM_low_L, R4
    SETB    EA

    POP     AR4
    POP     AR3
    RET
;********************** Timer0 中断函数************************/
F_Timer0_Interrupt:
    PUSH    PSW
    PUSH    ACC
    JNB     P_PWM, L_T0_LoadLow
    MOV     TH0, PWM_low_H              ; 如果是输出高电平，则装载低电平时间。
    MOV     TL0, PWM_low_L
    SJMP    L_QuitTimer0

L_T0_LoadLow:
    MOV     TH0, PWM_high_H             ; 如果是输出低电平，则装载高电平时间。
```

```
    MOV     TL0, PWM_high_L

L_QuitTimer0:
    POP     ACC
    POP     PSW
    RETI

END
```

22.12 用 T1 的时钟输出功能实现 8~16 位 PWM 的程序(C 及汇编)

----占用系统时间小于 0.4%

下文为利用 T1 的时钟输出功能实现 8~16 位 PWM(占用系统时间小于 0.4%)的测试程序(包括 C 语言程序和汇编程序)。

1、C 语言程序

```
/*----STC 1T Series MCU RC Demo ----*/
#include <reg52.h>
/*************************功能说明*************************
本程序演示使用定时器做软件 PWM。
定时器 1 做 16 位自动重装，中断，从 T1CLKO 高速输出 PWM。
本例程是使用 STC15FI 系列 MCU 的定时器 T1 做模拟 PWM 的例程。
PWM 可以是任意的量程。但是由于软件重装需要一点时间，所以 PWM 占空比最小为 32T/周期，最大为(周期-32TV 周期，T 为时钟周期。
PWM 频率为周期的倒数。假如周期为 6000，使用 24MHz 的主频，则 PWM 频率为 4000Hz。
*************************************************************/
#define   MAIN_Fosc             24000000UL  //定义主时钟
#define   PWM_DUTY              6000        //定义 PWM 的周期，数值为时钟周期数，假如使用
                                            //24.576MHZ 的主频，则 PWM 频率为 6000HZ。
#define   PWM_HIGH_MIN          32          //限制 PWM 输出的最小占空比。用户请勿修改。
#define   PWM_HIGH_MAX          (PWM_DUTY-PWM_HIGH_MIN)
                                            //限制 PWM 输出的最大占空比。用户请勿修改。
typedef   unsigned char         u8;
typedef   unsigned int          u16;
typedef   unsigned long         u32;

sfr       P3M1 = 0xB1;                      //P3M1.n,P3M0.n =00--->Standard, 01--->push-pull
sfr       P3M0 = 0xB2;                      // =10--->pure input, 11--->open drain
sfr       AUXR = 0x8E;
sfr       INT_CLKO= 0x8F;

sbit      P_PWM = P3^4;                     //定义 PWM 输出引脚。

u16       pwm;                              //定义 PWM 输出高电平的时间的变量。用户操作 PWM 的变量。
u16       PWM_high, PWM_low;                //中间变量，用户请勿修改。
```

```
void    delay_ms(u8 ms);
void    LoadPWM(u16 i);
/*********************  主函数**************************/
void main(void)
{
    P_PWM = 0;
    P3M1 &= ~(1 << 4);                  //P3.4  设置为推挽输出
    P3M0 |= (1 << 4);

    TR1 = 0;                            //停止计数
    ET1 = 1;                            //允许中断
    PT1 = 1;                            //高优先级中断
    TMOD &= ~0x30;                      //工作模式,0: 16 位自动重装
    AUXR |= 0x40;                       //1T
    TMOD &= ~0x40;                      //定时
    INT_CLKO |= 0x02;                   //输出时钟

    TH1 = 0;
    TL1 = 0;
    TR1 = 1;                            //开始运行

    EA = 1;

    pwm = PWM_DUTY / 10;                //给 PWM 一个初值，这里为 10%占空比

    LoadPWM(pwm);                       //计算 PWM 重装值

    while (1)
    {
        while(pwm < (PWM_HIGH_MAX-8))
        {
            pwm += 8;                   //PWM 逐渐加到最大
            LoadPWM(pwm);
            delay_ms(8);
        }
        while(pwm > (PWM_HIGH_MIN+8))
        {
            pwm -= 8;                   //PWM 逐渐减到最小
            LoadPWM(pwm);
            delay_ms(8);
        }
    }
}
//========================================================================
// 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.
//==========================================================================
void delay_ms(u8 ms)
{
     unsigned int i;
     do   {
               i = MAIN_Fosc / 13000;
               while(--i) ;
          }while(--ms);
}
/***************** 计算 PWM 重装值函数 ******************/
void LoadPWM(u16 i)
{
     u16 j;
     if(i > PWM_HIGH_MAX) i = PWM_HIGH_MAX;
                                        //如果写入大于最大占空比数据，则强制为最大占空比。
     if(i < PWM_HIGH_MIN) i = PWM_HIGH_MIN;
                                        //如果写入小于最小占空比数据，则强制为最小占空比。
     j = 65536UL - PWM_DUTY + i;        //计算 PWM 低电平时间
     i = 65536UL - i;                   //计算 PWM 高电平时间
     EA = 0;
     PWM_high = i;                      //装载 PWM 高电平时间
     PWM_low = j;                       //装载 PWM 低电平时间
     EA = 1;
}
/********************** Timer0 中断函数************************/
void timer0_int (void) interrupt 3
{
     if(P_PWM)
     {
          TH1 = (u8)(PWM_low >> 8);     //如果是输出高电平，则装载低电平时间。
          TL1 = (u8)PWM_low;
     }
     else
     {
          TH1 = (u8)(PWM_high >> 8);    //如果是输出低电平，则装载高电平时间。
          TL1 = (u8)PWM_high;
     }
}
```

2、汇编语言程序

```
/*----STC 1T Series MCU RC Demo----*/
;/************* 功能说明 **************
;本程序演示使用定时器做软件 PWM。
```

```
;定时器 1 做 16 位自动重装，中断，从 T1CLKO 高速输出 PWM。
;本例程是使用 STC15F/L 系列 MCU 的定时器 T1 做模拟 PWM 的例程。
;PWM 可以是任意的量程。但是由于软件重装需要一点时间，所以 PWM 占空比最小为 32T/周期，最大为(周期-32T)/周期，T 为时钟周期。
;PWM 频率为周期的倒数。假如周期为 6000, 使用 24MHz 的主频，则 PWM 频率为 4000Hz。
;*************************************/
;*************************用户宏定义*************************
Fosc_KHZ        EQU     24000   //定义主时钟, KHZ

PWM_DUTY        EQU     6000    //定义 PWM 的周期，数值为 PCA 所选择的时钟脉冲个数。
PWM_HIGH_MIN    EQU     32      //限制 PWM 输出的最小占空比，避免中断里重装参数时间不够。
PWM_HIGH_MAX    EQU     (PWM_DUTY - PWM_HIGH_MIN)   //限制 PWM 输出的最大占空比。
;************************************************************
P3M1            DATA    0B1H    ; P3M1.n,P3M0.n =00--->Standard, 01--->push-pull
P3M0            DATA    0B2H    ; =10--->pure input, 11--->open drain
AUXR            DATA    08EH
INT_CLKO        DATA    08FH

P_PWM  BIT      P3.4            ; 定义 PWM 输出引脚。

pwm_H           DATA    030H    ; 定义 PWM 输出高电平的时间的变量。用户操作 PWM 的变量。
pwm_L           DATA    031H
PWM_high_H      DATA    032H    ; 中间变量，用户请勿修改
PWM_high_L      DATA    033H
PWM_low_H       DATA    034H
PWM_low_L       DATA    035H

STACK_POIRTER   EQU     0D0H    ;堆栈开始地址
;*************************************************************************
    ORG     00H                             ;reset
    LJMP    F_Main

    ORG     1BH                             ;3 Timer1 interrupt
    LJMP    F_Timer1_Interrupt
;********************  主程序  ***************************/
F_Main:
    MOV     SP, #STACK_POIRTER
    MOV     PSW, #0
    USING   0                               ;选择第 0 组 R0~R7
;=================  用户初始化程序  ==================================
    CLR     P_PWM
    ANL     P3M1, #NOT (1 SHL 4)            ; P3.5  设置为推挽输出
    ORL     P3M0, #(1 SHL 4)

    CLR     TR1                             ; 停止计数
    SETB    ET1                             ; 允许中断
```

```
    SETB    PT1                     ; 高优先级中断
    ANL     TMOD, #NOT 030H         ; 工作模式,0: 16 位自动重装
    ORL     AUXR, #040H             ; 1T
    ANL     TMOD, #NOT 040H         ; 定时
    ORL     INT_CLKO, #002H         ; 输出时钟

    MOV     TH1, #0
    MOV     TL1, #0
    SETB    TR1                     ; 开始运行

    SETB    EA

    MOV     pwm_H, #HIGH (PWM_DUTY / 10)        ;给 PWM 一个初值，这里为 10%占空比
    MOV     pwm_L, #LOW (PWM_DUTY / 10)
    MOV     R6, pwm_H
    MOV     R7, pwm_L
    LCALL   F_PWMn_Update           ; 计算 PWM 重装值
;================== 主循环 ================================
L_MainLoop1:
    MOV     R7, #2
    LCALL   F_delay_ms

    MOV     A, pwm_L                ; if(++pwm >= PWM_HIGH_MAX)
    ADD     A, #1
    MOV     pwm_L, A
    MOV     A, pwm_H
    ADDC    A, #0
    MOV     pwm_H, A
    MOV     A, pwm_L
    CLR     C
    SUBB    A, #LOW PWM_HIGH_MAX
    MOV     A, pwm_H
    SUBB    A, #HIGH PWM_HIGH_MAX
    JC      L_PWM_NotUpOverFollow   ; PWM 逐渐加到最大
    MOV     pwm_H, #HIGH PWM_HIGH_MAX
    MOV     pwm_L, #LOW PWM_HIGH_MAX
    SJMP    L_MainLoop2

L_PWM_NotUpOverFollow:
    MOV     R6, pwm_H
    MOV     R7, pwm_L
    LCALL   F_PWMn_Update
    SJMP    L_MainLoop1

L_MainLoop2:
    MOV     R7, #2
```

```
    LCALL   F_delay_ms

    MOV     A, pwm_L                ; if(++pwm < PWM_HIGH_MIN)
    CLR     C
    SUBB    A, #1
    MOV     pwm_L, A
    MOV     A, pwm_H
    SUBB    A, #0
    MOV     pwm_H, A
    MOV     A, pwm_L
    CLR     C
    SUBB    A, #LOW PWM_HIGH_MIN
    MOV     A, pwm_H
    SUBB    A, #HIGH PWM_HIGH_MIN
    JNC     L_PWM_NotDnOverFollow   ; PWM 逐渐减到最小
    MOV     pwm_H, #HIGH PWM_HIGH_MIN
    MOV     pwm_L, #LOW PWM_HIGH_MIN
    SJMP    L_MainLoop1

L_PWM_NotDnOverFollow:
    MOV     R6, pwm_H
    MOV     R7, pwm_L
    LCALL   F_PWMn_Update
    SJMP    L_MainLoop2
;//==========================================================================
;// 函数: F_delay_ms
;// 描述: 延时子程序。
;// 参数: R7: 延时 ms 数.
;// 返回: none.
;// 备注: 除了 ACCC 和 PSW 外, 所用到的通用寄存器都入栈
;//==========================================================================
F_delay_ms:
    PUSH    AR3                     ;入栈 R3
    PUSH    AR4                     ;入栈 R4

L_delay_ms_1:
    MOV     R3, #HIGH (Fosc_KHZ / 13)
    MOV     R4, #LOW (Fosc_KHZ / 13)

L_delay_ms_2:
    MOV     A, R4                   ;1T Total 13T/loop
    DEC     R4                      ;2T
    JNZ     L_delay_ms_3            ;4T
    DEC     R3

L_delay_ms_3:
```

```
    DEC     A                       ;1T
    ORL     A, R3                   ;1T
    JNZ     L_delay_ms_2            ;4T
    DJNZ    R7, L_delay_ms_1
    POP     AR4                     ;出栈 R2
    POP     AR3                     ;出栈 R3

RET
;========================================================================
; 函数: F_PWMn_Update
; 描述: 更新占空比数据。
; 参数: R6,R7: PWM 值。
; 返回: 无
;========================================================================
F_PWMn_Update:
    PUSH    AR3
    PUSH    AR4
    CLR     C
    MOV     A, R7
    SUBB    A, #LOW PWM_HIGH_MAX
    MOV     A, R6
    SUBB    A, #HIGH PWM_HIGH_MAX
    JC      L_QuitCheckPwm_1
    MOV     R6, #HIGH PWM_HIGH_MAX      ; 如果写入大于最大占空比数据，强制为最大占空比。
    MOV     R7, #LOW PWM_HIGH_MAX

L_QuitCheckPwm_1:
    CLR     C
    MOV     A, R7
    SUBB    A, #LOW PWM_HIGH_MIN
    MOV     A, R6
    SUBB    A, #HIGH PWM_HIGH_MIN
    JNC     L_QuitCheckPwm_2
    MOV     R6, #HIGH PWM_HIGH_MIN      ; 如果写入小于最小占空比数据，强制为最小占空比。
    MOV     R7, #LOW PWM_HIGH_MIN

L_QuitCheckPwm_2:
    CLR     C
    MOV     A, R7                       ;计算并保存 PWM 输出低电平的 T0 时钟脉冲个数
    SUBB    A, #LOW PWM_DUTY
    MOV     R4, A
    MOV     A, R6
    SUBB    A, #HIGH PWM_DUTY
    MOV     R3, A

    CLR     C
```

```
    MOV     A, #0                   ;计算并保存 PWM 输出高电平的 T0 时钟脉冲个数
    SUBB    A, R7
    MOV     R7, A
    MOV     A, #0
    SUBB    A, R6
    MOV     R6, A

    CLR     EA                      ; 禁止一会中断， 一般不会影响 PWM。
    MOV     PWM_high_H, R6          ; 数据装入占空比变量。
    MOV     PWM_high_L, R7
    MOV     PWM_low_H, R3
    MOV     PWM_low_L, R4
    SETB    EA

    POP     AR4
    POP     AR3
    RET
;********************** Timer0 中断函数************************/
F_Timer1_Interrupt:
    PUSH    PSW
    PUSH    ACC
    JNB     P_PWM, L_T1_LoadLow
    MOV     TH1, PWM_low_H          ; 如果是输出高电平，则装载低电平时间。
    MOV     TL1, PWM_low_L
    SJMP    L_QuitTimer1

L_T1_LoadLow:
    MOV     TH1, PWM_high_H         ; 如果是输出低电平，则装载高电平时间。
    MOV     TL1, PWM_high_L

L_QuitTimer1:
    POP     ACC
    POP     PSW
    RETI
    END
```

22.13 用 T2 的时钟输出功能实现 8~16 位 PWM 的程序(C 及汇编)

----占用系统时间小于 0.4%

下文为利用 T2 的时钟输出功能实现 8~16 位 PWM(占用系统时间小于 0.4%)的测试程序(包括 C 语言程序和汇编程序)。

1、C 语言程序

```
/*----STC 1T Series MCU RC Demo----*/
#include <reg52.h>
/*****************功能说明*****************
本程序演示使用定时器做软件 PWM。
定时器 2 做 16 位自动重装，中断，从 T2CLKO 高速输出 PWM。
本例程是使用 STC15FL 系列 MCU 的定时器 T2 微模拟 PWM 的例程。
PWM 可以是任意的量程。但是由于软件重装需要一点时间，所以 PWM 占空比最小为 32T/周期，最大为(周期-32T/周期,T 为时钟周期。
PWM 频率为周期的倒数。假如周期为 6000,使用 24MHz 的主频，则 PWM 频率为 4000Hz。
*****************************************/
#define   MAIN_Fosc          24000000UL      //定义主时钟
#define   PWM_DUTY           6000            //定义 PWM 的周期，数值为时钟周期数，假如使用
                                             //24.576MHZ 的主频，则 PWM 频率为 6000Hz。
#define   PWM_HIGH_MIN       32              //限制 PWM 输出的最小占空比。用户请勿修改。
#define   PWM_HIGH_MAX       (PWM_DUTY-PWM_HIGH_MIN)
                                             //限制 PWM 输出的最大占空比。用户请勿修改。
typedef   unsigned char      u8;
typedef   unsigned int       u16;
typedef   unsigned long      u32;

sfr       IE2 = 0xAF;                        //STC12C5A60S2 系列
sfr       P3M1 = 0xB1;                       //P3M1.n,P3M0.n =00--->Standard, 01--->push-pull
sfr       P3M0 = 0xB2;                       // =10--->pure input, 11--->open drain
sfr       AUXR = 0x8E;
sfr       INT_CLKO = 0x8F;

sbit      P_PWM = P3^0;                      //定义 PWM 输出引脚。

u16       pwm;                               //定义PWM输出高电平的时间的变量。用户操作PWM的变量。
u16       PWM_high,PWM_low;                  //中间变量，用户请勿修改。

void delay_ms(u8 ms);
void LoadPWM(u16 i);
/*********************  主函数**************************/
void main(void)
{
    P_PWM = 0;
    P3M1 &= ~1;                              //P3.0  设置为推挽输出
```

```
    P3M0 |= 1;

    AUXR &= ~(1<<4);                    //停止计数
    IE2 |= (1<<2);                      //允许中断
    AUXR |= (1<<2);                     //1T
    AUXR &= ~(1<<3);                    //定时
    INT_CLKO |= 0x04;                   //输出时钟

    TH2 = 0;
    TL2 = 0;
    AUXR |= (1<<4);                     //开始运行

    EA = 1;

    pwm = PWM_DUTY / 10;                //给 PWM 一个初值，这里为 10%占空比
    LoadPWM(pwm);                       //计算 PWM 重装值

    while (1)
    {
        while(pwm < (PWM_HIGH_MAX-8))
        {
            pwm += 8;                   //PWM 逐渐加到最大
            LoadPWM(pwm);
            delay_ms(8);
        }
        while(pwm > (PWM_HIGH_MIN+8))
        {
            pwm -= 8;                   //PWM 逐渐减到最小
            LoadPWM(pwm);
            delay_ms(8);
        }
    }
}

//========================================================================
// 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.
//========================================================================
void delay_ms(u8 ms)
{
    unsigned int i;
    do{
        i = MAIN_Fosc / 13000;
        while(--i) ;
```

```
        }while(--ms);
}
/***************** 计算 PWM 重装值函数 ******************/
void LoadPWM(u16 i)
{
    u16 j;
    if(i > PWM_HIGH_MAX) i = PWM_HIGH_MAX;
                                        //如果写入大于最大占空比数据，则强制为最大占空比。
    if(i < PWM_HIGH_MIN) i = PWM_HIGH_MIN;
                                        //如果写入小于最小占空比数据，则强制为最小占空比。
    j = 65536UL - PWM_DUTY + i;         //计算 PWM 低电平时间
    i = 65536UL - i;                    //计算 PWM 高电平时间
    EA = 0;
    PWM_high = i;                       //装载 PWM 高电平时间
    PWM_low = j;                        //装载 PWM 低电平时间
    EA = 1;
}
/********************** Timer0 中断函数************************/
void timer2_int (void) interrupt 12
{
    if(P_PWM)
    {
        TH2 = (u8)(PWM_low >> 8);       //如果是输出高电平，则装载低电平时间。
        TL2 = (u8)PWM_low;
    }
    else
    {
        TH2 = (u8)(PWM_high >> 8);      //如果是输出低电平，则装载高电平时间。
        TL2 = (u8)PWM_high;
    }
}
```

2、汇编语言程序

```
/*----STC 1T Series MCU RC Demo----*/
;/************* 功能说明 ***************
;本程序演示使用定时器做软件 PWM。
;定时器 2 做 16 位自动重装，中断，从 T2CLKO 高速输出 PWM。
;本例程是使用 STC15F/L 系列 MCU 的定时器 T2 做模拟 PWM 的例程。
;PWM 可以是任意的量程。但是由于软件重装需要一点时间，所以 PWM 占空比最小为 32T/周期，最大为(周期-32T)/周期, T 为时钟周期。
;PWM 频率为周期的倒数。假如周期为 6000, 使用 24MHz 的主频，则 PWM 频率为 4000Hz。
;*******************************************/
;*************************用户宏定义***********************
Fosc_KHZ                EQU     24000   //定义主时钟, KHz
```

```
PWM_DUTY        EQU     6000    //定义 PWM 的周期，数值为 PCA 所选择的时钟脉冲个数。
PWM_HIGH_MIN    EQU     32      //限制 PWM 输出的最小占空比，避免中断里重装参数时间不够。
PWM_HIGH_MAX    EQU     (PWM_DUTY - PWM_HIGH_MIN)
                                //限制 PWM 输出的最大占空比。
;******* ********************************************************
P3M1            DATA    0B1H    ; P3M1.n,P3M0.n =00--->Standard, 01--->push-pull
P3M0            DATA    0B2H    ; =10--->pure input, 11--->open drain
AUXR            DATA    08EH
INT_CLKO        DATA    08FH

IE2             DATA    0AFH
T2H             DATA    0D6H
T2L             DATA    0D7H
P_PWM           BIT     P3.0    ; 定义 PWM 输出引脚。

pwm_H           DATA    030H    ; 定义 PWM 输出高电平的时间的变量。用户操作 PWM 的变量。
pwm_L           DATA    031H
PWM_high_H      DATA    032H    ; 中间变量，用户请勿修改
PWM_high_L      DATA    033H
PWM_low_H       DATA    034H
PWM_low_L       DATA    035H

STACK_POIRTER   EQU     0D0H    ;堆栈开始地址
;*************************************************************************
;*************************************************************************
    ORG     00H                         ;reset
    LJMP    F_Main

    ORG     63H                         ;12 Timer2 interrupt
    LJMP    F_Timer2_Interrupt
;********************  主程序  ***************************/
F_Main:
    MOV     SP, #STACK_POIRTER
    MOV     PSW, #0
    USING   0 ;选择第 0 组 R0~R7
;==================  用户初始化程序  ====================================
    CLR     P_PWM
    ANL     P3M1, #NOT 1            ; P3.0  设置为推挽输出
    ORL     P3M0, #1

    ANL     AUXR, #NOT (1 SHL 4)    ;停止计数
    ORL     IE2, #(1 SHL 2)         ; 允许中断
    ORL     AUXR, #(1 SHL 2)        ; 1T
    ANL     AUXR, #NOT (1 SHL 3)    ; 定时
    ORL     INT_CLKO, #0x04         ; 输出时钟
```

```
    MOV     T2H, #0;
    MOV     T2L, #0;
    ORL     AUXR, #(1 SHL 4)            ; 开始运行

    SETB    EA

    MOV     pwm_H, #HIGH (PWM_DUTY / 10)        ;给 PWM 一个初值，这里为 10%占空比
    MOV     pwm_L, #LOW (PWM_DUTY / 10)
    MOV     R6, pwm_H
    MOV     R7, pwm_L
    LCALL   F_PWMn_Update               ; 计算 PWM 重装值
;================== 主循环 ==================================
L_MainLoop1:
    MOV     R7, #2
    LCALL   F_delay_ms

    MOV     A, pwm_L                    ; if(++pwm >= PWM_HIGH_MAX)
    ADD     A, #1
    MOV     pwm_L, A
    MOV     A, pwm_H
    ADDC    A, #0
    MOV     pwm_H, A
    MOV     A, pwm_L
    CLR     C
    SUBB    A, #LOW PWM_HIGH_MAX
    MOV     A, pwm_H
    SUBB    A, #HIGH PWM_HIGH_MAX
    JC      L_PWM_NotUpOverFollow       ; PWM 逐渐加到最大
    MOV     pwm_H, #HIGH PWM_HIGH_MAX
    MOV     pwm_L, #LOW PWM_HIGH_MAX
    SJMP    L_MainLoop2

L_PWM_NotUpOverFollow:
    MOV     R6, pwm_H
    MOV     R7, pwm_L
    LCALL   F_PWMn_Update
    SJMP    L_MainLoop1

L_MainLoop2:
    MOV     R7, #2
    LCALL   F_delay_ms

    MOV     A, pwm_L                    ; if(++pwm < PWM_HIGH_MIN)
    CLR     C
    SUBB    A, #1
    MOV     pwm_L, A
```

```
    MOV     A, pwm_H
    SUBB    A, #0
    MOV     pwm_H, A
    MOV     A, pwm_L
    CLR     C
    SUBB    A, #LOW PWM_HIGH_MIN
    MOV     A, pwm_H
    SUBB    A, #HIGH PWM_HIGH_MIN
    JNC     L_PWM_NotDnOverFollow       ; PWM 逐渐减到最小
    MOV     pwm_H, #HIGH PWM_HIGH_MIN
    MOV     pwm_L, #LOW PWM_HIGH_MIN
    SJMP    L_MainLoop1

L_PWM_NotDnOverFollow:
    MOV     R6, pwm_H
    MOV     R7, pwm_L
    LCALL   F_PWMn_Update
    SJMP    L_MainLoop2
;//=========================================================================
;// 函数: F_delay_ms
;// 描述: 延时子程序。
;// 参数: R7: 延时 ms 数.
;// 返回: none.
;// 备注: 除了 ACCC 和 PSW 外, 所用到的通用寄存器都入栈
;//=========================================================================
F_delay_ms:
    PUSH    AR3                         ;入栈 R3
    PUSH    AR4                         ;入栈 R4

L_delay_ms_1:
    MOV     R3, #HIGH (Fosc_KHZ / 13)
    MOV     R4, #LOW (Fosc_KHZ / 13)

L_delay_ms_2:
    MOV     A, R4                       ;1T Total 13T/loop
    DEC     R4                          ;2T
    JNZ     L_delay_ms_3                ;4T
    DEC     R3

L_delay_ms_3:
    DEC     A                           ;1T
    ORL     A, R3                       ;1T
    JNZ     L_delay_ms_2                ;4T
    DJNZ    R7, L_delay_ms_1
    POP     AR4                         ;出栈 R2
    POP     AR3                         ;出栈 R3
```

```
RET
;========================================================================
; 函数: F_PWMn_Update
; 描述: 更新占空比数据。
; 参数: R6,R7: PWM 值。
; 返回: 无
;========================================================================
F_PWMn_Update:
    PUSH    AR3
    PUSH    AR4

    CLR     C
    MOV     A, R7
    SUBB    A, #LOW PWM_HIGH_MAX
    MOV     A, R6
    SUBB    A, #HIGH PWM_HIGH_MAX
    JC      L_QuitCheckPwm_1
    MOV     R6, #HIGH PWM_HIGH_MAX      ; 如果写入大于最大占空比数据，强制为最大占空比。
    MOV     R7, #LOW PWM_HIGH_MAX

L_QuitCheckPwm_1:
    CLR     C
    MOV     A, R7
    SUBB    A, #LOW PWM_HIGH_MIN
    MOV     A, R6
    SUBB    A, #HIGH PWM_HIGH_MIN
    JNC     L_QuitCheckPwm_2
    MOV     R6, #HIGH PWM_HIGH_MIN      ; 如果写入小于最小占空比数据，强制为最小占空比。
    MOV     R7, #LOW PWM_HIGH_MIN

L_QuitCheckPwm_2:
    CLR     C
    MOV     A, R7                       ;计算并保存 PWM 输出低电平的 T0 时钟脉冲个数
    SUBB    A, #LOW PWM_DUTY
    MOV     R4, A
    MOV     A, R6
    SUBB    A, #HIGH PWM_DUTY
    MOV     R3, A

    CLR     C
    MOV     A, #0                       ;计算并保存 PWM 输出高电平的 T0 时钟脉冲个数
    SUBB    A, R7
    MOV     R7, A
    MOV     A, #0
    SUBB    A, R6
    MOV     R6, A
```

```
    CLR     EA                          ; 禁止一会中断， 一般不会影响 PWM。
    MOV     PWM_high_H, R6              ; 数据装入占空比变量。
    MOV     PWM_high_L, R7
    MOV     PWM_low_H, R3
    MOV     PWM_low_L, R4
    SETB    EA
    POP     AR4
    POP     AR3
RET
;********************** Timer0 中断函数************************/
F_Timer2_Interrupt:
    PUSH    PSW
    PUSH    ACC
    JNB     P_PWM, L_T2_LoadLow
    MOV     T2H, PWM_low_H              ; 如果是输出高电平，则装载低电平时间。
    MOV     T2L, PWM_low_L
    SJMP    L_QuitTimer2

L_T2_LoadLow:
    MOV     T2H, PWM_high_H             ; 如果是输出低电平，则装载高电平时间。
    MOV     T2L, PWM_high_L

L_QuitTimer2:
    POP     ACC
    POP     PSW
    RETI

END
```

22.14 利用两路 CCP/PCA 模拟一个全双工串口的程序(C 及汇编)

1.C 程序:

```
/**********************功能说明*****************
使用 STC15 系列的 PCA0 和 PCA1 做的模拟串口.PCA0 接收(P2.5),PCA1 发送(P2.6).
假定测试芯片的工作频率为 22118400Hz，时钟为 5.5296MHz ~ 35MHz.
波特率高，则时钟也要选高,优先使用 22.1184MHz，11.0592MHz
测试方法：上位机发送数据，MCU 收到数据后原样返回.
串口固定设置：1 位起始位，8 位数据位，1 位停止位，波特率在 600~57600 bps.
1200 ~ 57600 bps   @ 33.1776MHz
600 ~ 57600 bps    @ 22.1184MHz
600 ~ 38400 bps    @ 18.4320MHz
300 ~ 28800 bps    @ 11.0592MHz
150 ~ 14400 bps    @ 5.5296MHz
*************************************************/
#include <reg52.h>

#define   MAIN_Fosc              22118400UL      //定义主时钟
#define   UART3_Baudrate         57600UL         //定义波特率
#define   RX_Lenth               16              //接收长度

#define   PCA_P12_P11_P10_P37    (0<<4)
#define   PCA_P34_P35_P36_P37    (1<<4)
#define   PCA_P24_P25_P26_P27    (2<<4)
#define   PCA_Mode_Capture       0
#define   PCA_Mode_SoftTimer     0x48
#define   PCA_Clock_1T           (4<<1)
#define   PCA_Clock_2T           (1<<1)
#define   PCA_Clock_4T           (5<<1)
#define   PCA_Clock_6T           (6<<1)
#define   PCA_Clock_8T           (7<<1)
#define   PCA_Clock_12T          (0<<1)
#define   PCA_Clock_ECI          (3<<1)
#define   PCA_Rise_Active        (1<<5)
#define   PCA_Fall_Active        (1<<4)
#define   PCA_PWM_8bit           (0<<6)
#define   PCA_PWM_7bit           (1<<6)
#define   PCA_PWM_6bit           (2<<6)

#define   UART3_BitTime          (MAIN_Fosc / UART3_Baudrate)

#define   ENABLE                 1
#define   DISABLE                0
```

```
typedef   unsigned char          u8;
typedef   unsigned int           u16;
typedef   unsigned long          u32;

sfr       AUXR1 = 0xA2;
sfr       CCON = 0xD8;
sfr       CMOD = 0xD9;
sfr       CCAPM0= 0xDA;          //PCA 模块 0 的工作模式寄存器。
sfr       CCAPM = 0xDB;          //PCA 模块 1 的工作模式寄存器。
sfr       CCAPM2= 0xDC;          //PCA 模块 2 的工作模式寄存器。

sfr       CL = 0xE9;
sfr       CCAP0L = 0xEA;         //PCA 模块 0 的捕捉/比较寄存器低 8 位。
sfr       CCAP1L = 0xEB;         //PCA 模块 1 的捕捉/比较寄存器低 8 位。
sfr       CCAP2L = 0xEC;         //PCA 模块 2 的捕捉/比较寄存器低 8 位。

sfr       CH = 0xF9;
sfr       CCAP0H = 0xFA;         //PCA 模块 0 的捕捉/比较寄存器高 8 位。
sfr       CCAP1H = 0xFB;         //PCA 模块 1 的捕捉/比较寄存器高 8 位。
sfr       CCAP2H = 0xFC;         //PCA 模块 2 的捕捉/比较寄存器高 8 位。

sbit      CCF0 = CCON^0;         //PCA  模块 0 中断标志，由硬件置位，必须由软件清 0。
sbit      CCF1 = CCON^1;         //PCA  模块 1 中断标志，由硬件置位，必须由软件清 0。
sbit      CCF2 = CCON^2;         //PCA  模块 2 中断标志，由硬件置位，必须由软件清 0。
sbit      CR = CCON^6;           //1: 允许 PCA 计数器计数，0: 禁止计数。
sbit      CF = CCON^7;           //PCA 计数器溢出（CH，CL 由 FFFFH 变为 0000H）标志。
                                 //PCA 计数器溢出后由硬件置位，必须由软件清 0。
sbit      PPCA = IP^7;           //PCA  中断  优先级设定位

u16       CCAP0_tmp;
u16       CCAP1_tmp;

u8        Tx3_read;              //发送读指针
u8        Rx3_write;             //接收写指针
u8        idata buf3[RX_Lenth];  //接收缓冲
//=====================  模拟串口相关==========================
sbit      P_RX3 = P2^5;          //定义模拟串口接收 IO
sbit      P_TX3 = P2^6;          //定义模拟串口发送 IO

u8        Tx3_DAT;               // 发送移位变量, 用户不可见
u8        Rx3_DAT;               // 接收移位变量, 用户不可见
u8        Tx3_BitCnt;            // 发送数据的位计数器, 用户不可见
u8        Rx3_BitCnt;            // 接收数据的位计数器, 用户不可见
u8        Rx3_BUF;               // 接收到的字节, 用户读取
u8        Tx3_BUF;               // 要发送的字节, 用户写入
```

```
bit     Rx3_Ring;                   // 正在接收标志, 低层程序使用, 用户程序不可见
bit     Tx3_Ting;                   // 正在发送标志, 用户置 1 请求发送, 底层发送完成清 0
bit     RX3_End;                    // 接收到一个字节, 用户查询 并清 0
//=========================================================
void PCA_Init(void);
/********************  主函数  ***************************/
void main(void)
{
    PCA_Init();                                         //PCA 初始化
    EA = 1;
    Tx3_read = 0;
    Rx3_write = 0;
    Tx3_Ting = 0;
    Rx3_Ring = 0;
    RX3_End = 0;
    Tx3_BitCnt = 0;

    while (1)                                           //user's function
    {
        if (RX3_End)                                    // 检测是否收到一个字节
        {
            RX3_End = 0;                                // 清除标志
            buf3[Rx3_write] = Rx3_BUF;                  // 写入缓冲
            if(++Rx3_write >= RX_Lenth) Rx3_write = 0;   // 指向下一个位置, 溢出检测
        }

        if (!Tx3_Ting)                                  // 检测是否发送空闲
        {
            if (Tx3_read != Rx3_write)                  // 检测是否收到过字符
            {
                Tx3_BUF = buf3[Tx3_read];               // 从缓冲读一个字符发送
                Tx3_Ting = 1;                           // 设置发送标志
                if(++Tx3_read >= RX_Lenth) Tx3_read = 0;  // 指向下一个位置, 溢出检测
            }
        }
    }
}
//========================================================================
// 函数: void PCA_Init(void)
// 描述: PCA 初始化程序.
// 参数: none
// 返回: none.
//========================================================================
void PCA_Init(void)
{
    CR = 0;
```

```
    CCAPM0 = (PCA_Mode_Capture | PCA_Fall_Active | ENABLE);   //16 位下降沿捕捉中断模式
    CCAPM1 = PCA_Mode_SoftTimer | ENABLE;

    CCAP1_tmp = UART3_BitTime;

    CCAP1L = (u8)CCAP1_tmp;                 //将影射寄存器写入捕获寄存器，先写 CCAP0L
    CCAP1H = (u8)(CCAP1_tmp >> 8);          //后写 CCAP0H

    CH = 0;
    CL = 0;
    AUXR1 = (AUXR1 & ~(3<<4)) | PCA_P24_P25_P26_P27;        //切换 I/O 口
    CMOD = (CMOD & ~(7<<1)) | PCA_Clock_1T;     //选择时钟源
    PPCA = 1;                                   // 高优先级中断
    CR = 1;                                     // 运行 PCA 定时器
}
//========================================================================
// 函数: void PCA_Handler (void) interrupt 7
// 描述: PCA 中断处理程序.
// 参数: None
// 返回: none.
//========================================================================
void PCA_Handler (void) interrupt 7
{
    if(CCF0)                                    //PCA 模块 0 中断
    {
        CCF0 = 0;                               //清 PCA 模块 0 中断标志
        if(Rx3_Ring)                            //已收到起始位
        {
            if (--Rx3_BitCnt == 0)              //接收完一帧数据
            {
                Rx3_Ring = 0;                   //停止接收
                Rx3_BUF = Rx3_DAT;              //存储数据到缓冲区
                RX3_End = 1;
                CCAPM0 = (PCA_Mode_Capture | PCA_Fall_Active | ENABLE);
                                                //16 位下降沿捕捉中断模式
            }
            else
            {
                Rx3_DAT >>= 1;                  //把接收的单 b 数据 暂存到 RxShiftReg(接收缓冲)
                if(P_RX3) Rx3_DAT |= 0x80;      //shift RX data to RX buffer
                CCAP0_tmp += UART3_BitTime;     //数据位时间
                CCAP0L = (u8)CCAP0_tmp;         //将影射寄存器写入捕获寄存器，先写 CCAP0L
                CCAP0H = (u8)(CCAP0_tmp >> 8);  //后写 CCAP0H
            }
        }
        else
```

```
        {
            CCAP0_tmp = ((u16)CCAP0H << 8) + CCAP0L;                //读捕捉寄存器
            CCAP0_tmp += (UART3_BitTime / 2 + UART3_BitTime);       //起始位 + 半个数据位
            CCAP0L = (u8)CCAP0_tmp;                 //将影射寄存器写入捕获寄存器，先写 CCAP0L
            CCAP0H = (u8)(CCAP0_tmp >> 8);          //后写 CCAP0H
            CCAPM0 = (PCA_Mode_SoftTimer | ENABLE);                 //16 位软件定时中断模式
            Rx3_Ring = 1;                           //标志已收到起始位
            Rx3_BitCnt = 9;                         //初始化接收的数据位数(8 个数//据位+1 个停止位)
        }
    }

    if(CCF1)                                        //PCA 模块 1 中断, 16 位软件定时中断模式
    {
        CCF1 = 0;                                   //清 PCA 模块 1 中断标志
        CCAP1_tmp += UART3_BitTime;
        CCAP1L = (u8)CCAP1_tmp;                     //将影射寄存器写入捕获寄存器，先写 CCAP0L
        CCAP1H = (u8)(CCAP1_tmp >> 8);              //后写 CCAP0H
        if(Tx3_Ting)                                //  不发送, 退出
        {
            if(Tx3_BitCnt == 0)                     //发送计数器为 0  表明单字节发送还没开始
            {
                P_TX3 = 0;                          //发送开始位
                Tx3_DAT = Tx3_BUF;                  //把缓冲的数据放到发送的 buff
                Tx3_BitCnt = 9;                     //发送数据位数  (8 数据位+1 停止位)
            }
            else                                    //发送计数器为非 0  正在发送数据
            {
                if (--Tx3_BitCnt == 0)              //发送计数器减为 0  表明单字节发送结束
                {
                    P_TX3 = 1;                      //送停止位数据
                    Tx3_Ting = 0;                   //发送停止
                }
                else
                {
                    Tx3_DAT >>= 1;                  //把最低位送到  CY(益处标志位)
                    P_TX3 = CY;                     //发送一个 bit 数据
                }
            }
        }
    }
}
```

2. 汇编程序：

```
;************* 功能说明 **************
;使用 STC15 系列的 PCA0 和 PCA1 做的模拟串口. PCA0 接收(P2.5), PCA1 发送(P2.6).
```

```
;假定测试芯片的工作频率为 22118400Hz. 时钟为 5.5296MHz ~ 35MHz.
;波特率高，则时钟也要选高, 优先使用 22.1184MHz, 11.0592MHz.
;测试方法: 上位机发送数据,MCU 收到数据后原样返回.
;串口固定设置: 1 位起始位, 8 位数据位, 1 位停止位.
;*****************************************
STACK_POIRTER       EQU     0D0H    ;堆栈开始地址

;UART3_BitTime      EQU     9216    ; 1200bps @ 11.0592MHz
                                    ;UART3_BitTime = (MAIN_Fosc / Baudrate)
;UART3_BitTime      EQU     4608    ; 2400bps @ 11.0592MHz
;UART3_BitTime      EQU     2304    ; 4800bps @ 11.0592MHz
;UART3_BitTime      EQU     1152    ; 9600bps @ 11.0592MHz
;UART3_BitTime      EQU     576     ;19200bps @ 11.0592MHz
;UART3_BitTime      EQU     288     ;38400bps @ 11.0592MHz

;UART3_BitTime      EQU     15360   ; 1200bps @ 18.432MHz
;UART3_BitTime      EQU     7680    ; 2400bps @ 18.432MHz
;UART3_BitTime      EQU     3840    ; 4800bps @ 18.432MHz
;UART3_BitTime      EQU     1920    ; 9600bps @ 18.432MHz
;UART3_BitTime      EQU     960     ;19200bps @ 18.432MHz
;UART3_BitTime      EQU     480     ;38400bps @ 18.432MHz
;UART3_BitTime      EQU     320     ;57600bps @ 18.432MHz

;UART3_BitTime      EQU     18432   ; 1200bps @ 22.1184MHz
;UART3_BitTime      EQU     9216    ; 2400bps @ 22.1184MHz
;UART3_BitTime      EQU     4608    ; 4800bps @ 22.1184MHz
;UART3_BitTime      EQU     2304    ; 9600bps @ 22.1184MHz
;UART3_BitTime      EQU     1152    ;19200bps @ 22.1184MHz
;UART3_BitTime      EQU     576     ;38400bps @ 22.1184MHz
UART3_BitTime       EQU     384     ;57600bps @ 22.1184MHz

;UART3_BitTime      EQU     27648   ; 1200bps @ 33.1776MHz
;UART3_BitTime      EQU     13824   ; 2400bps @ 33.1776MHz
;UART3_BitTime      EQU     6912    ; 4800bps @ 33.1776MHz
;UART3_BitTime      EQU     3456    ; 9600bps @ 33.1776MHz
;UART3_BitTime      EQU     1728    ;19200bps @ 33.1776MHz
;UART3_BitTime      EQU     864     ;38400bps @ 33.1776MHz
;UART3_BitTime      EQU     576     ;57600bps @ 33.1776MHz
;UART3_BitTime      EQU     288     ;115200bps @ 33.1776MHz

PCA_P12_P11_P10_P37 EQU     (0 SHL 4)
PCA_P34_P35_P36_P37 EQU     (1 SHL 4)
PCA_P24_P25_P26_P27 EQU     (2 SHL 4)
PCA_Mode_Capture    EQU     0
PCA_Mode_SoftTimer  EQU     048H
PCA_Clock_1T        EQU     (4 SHL 1)
```

```
PCA_Clock_2T        EQU     (1 SHL 1)
PCA_Clock_4T        EQU     (5 SHL 1)
PCA_Clock_6T        EQU     (6 SHL 1)
PCA_Clock_8T        EQU     (7 SHL 1)
PCA_Clock_12T       EQU     (0 SHL 1)
PCA_Clock_ECI       EQU     (3 SHL 1)
PCA_Rise_Active     EQU     (1 SHL 5)
PCA_Fall_Active     EQU     (1 SHL 4)

ENABLE              EQU     1

AUXR1               DATA    0xA2
CCON                DATA    0xD8
CMOD                DATA    0xD9
CCAPM0              DATA    0xDA    ; PCA 模块 0 的工作模式寄存器。
CCAPM1              DATA    0xDB    ; PCA 模块 1 的工作模式寄存器。
CCAPM2              DATA    0xDC    ; PCA 模块 2 的工作模式寄存器。

CL                  DATA    0xE9
CCAP0L              DATA    0xEA    ; PCA 模块 0 的捕捉/比较寄存器低 8 位。
CCAP1L              DATA    0xEB    ; PCA 模块 1 的捕捉/比较寄存器低 8 位。
CCAP2L              DATA    0xEC    ; PCA 模块 2 的捕捉/比较寄存器低 8 位。

CH                  DATA    0xF9
CCAP0H              DATA    0xFA    ; PCA 模块 0 的捕捉/比较寄存器高 8 位。
CCAP1H              DATA    0xFB    ; PCA 模块 1 的捕捉/比较寄存器高 8 位。
CCAP2H              DATA    0xFC    ; PCA 模块 2 的捕捉/比较寄存器高 8 位。

CCF0        BIT     CCON.0      ; PCA 模块 0 中断标志，由硬件置位，必须由软件清 0。
CCF1        BIT     CCON.1      ; PCA 模块 1 中断标志，由硬件置位，必须由软件清 0。
CCF2        BIT     CCON.2      ; PCA 模块 2 中断标志，由硬件置位，必须由软件清 0。
CR          BIT     CCON.6      ; 1: 允许 PCA 计数器计数， 0: 禁止计数。
CF          BIT     CCON.7      ; PCA 计数器溢出（CH，CL 由 FFFFH 变为 0000H）标志。
                                ;PCA 计数器溢出后由硬件置位，必须由软件清 0。
PPCA        BIT     IP.7        ; PCA 中断 优先级设定位
;====================== 模拟串口相关 ============================
P_RX3       BIT     P2.5        ; 定义模拟串口接收 IO
P_TX3       BIT     P2.6        ; 定义模拟串口发送 IO

Rx3_Ring    BIT     20H.0       ; 正在接收标志, 低层程序使用, 用户程序不可见
Tx3_Ting    BIT     20H.1       ; 正在发送标志, 用户置 1 请求发送, 底层发送完成清 0
RX3_End     BIT     20H.2       ; 接收到一个字节, 用户查询 并清 0

Tx3_DAT     DATA    30H         ; 发送移位变量, 用户不可见
Rx3_DAT     DATA    31H         ; 接收移位变量, 用户不可见
Tx3_BitCnt  DATA    32H         ; 发送数据的位计数器, 用户不可见
```

```
Rx3_BitCnt      DATA    33H         ; 接收数据的位计数器, 用户不可见
Rx3_BUF         DATA    34H         ; 接收到的字节, 用户读取
Tx3_BUF         DATA    35H         ; 要发送的字节, 用户写入
;=================================================================
Tx3_read        DATA    36H         ; 发送读指针
Rx3_write       DATA    37H         ; 接收写指针
RX_Lenth        EQU     16          ; 接收长度

buf3            EQU     40H         ; 40H ~ 4FH  接收缓冲
;*****************************************************************
    ORG     00H                     ;reset
    LJMP    F_Main

    ORG     3BH                     ;7 PCA interrupt
    LJMP    F_PCA_Interrupt
;********************  主程序  **************************/
F_Main:
    MOV     SP, #STACK_POIRTER
    MOV     PSW, #0
    USING   0                       ;选择第 0 组 R0~R7
;================  用户初始化程序  ===================================
    LCALL   F_PCA_Init              ;PCA 初始化
    SETB    EA

    MOV     Tx3_read, #0
    MOV     Rx3_write, #0
    CLR     Tx3_Ting
    CLR     RX3_End
    CLR     Rx3_Ring
    MOV     Tx3_BitCnt, #0
;==================  主循环  ==================================
L_MainLoop:
    JNB     RX3_End, L_QuitRx3      ; 检测是否收到一个字节
    CLR     RX3_End                 ; 清除标志
    MOV     A, #buf3
    ADD     A, Rx3_write
    MOV     R0, A
    MOV     @R0, Rx3_BUF            ; 写入缓冲
    INC     Rx3_write               ; 指向下一个位置
    MOV     A, Rx3_write
    CLR     C
    SUBB    A, #RX_Lenth            ; 溢出检测
    JC      L_QuitRx3
    MOV     Rx3_write, #0

L_QuitRx3:
```

```
    JB      Tx3_Ting, L_QuitTx3     ;检测是否发送空闲
    MOV     A, Tx3_read
    XRL     A, Rx3_write
    JZ      L_QuitTx3               ;检测是否收到过字符

    MOV     A, #buf3
    ADD     A, Tx3_read
    MOV     R0, A
    MOV     Tx3_BUF, @R0            ; 从缓冲读一个字符发送
    SETB     Tx3_Ting               ; 设置发送标志
    INC     Tx3_read                ; 指向下一个字符位置
    MOV     A, Tx3_read
    CLR     C
    SUBB    A, #RX_Lenth
    JC      L_QuitTx3               ; 溢出检测
    MOV     Tx3_read, #0

L_QuitTx3:
    SJMP    L_MainLoop
;=================== 主程序结束 ================================
; 函数: F_PCA_Init
; 描述: PCA 初始化程序.
; 参数: none
; 返回: none.
;================================================================================
F_PCA_Init:
    CLR     CR
    MOV     CCAPM0, #(PCA_Mode_Capture OR PCA_Fall_Active OR ENABLE)
                                    ; 16 位下降沿捕捉中断模式
    MOV     CCAPM1, #(PCA_Mode_SoftTimer OR ENABLE)
                                    ; 16 位软件定时器, 中断模式
    MOV     CCAP1L, #LOW UART3_BitTime
                                    ; 将影射寄存器写入捕获寄存器，先写 CCAP0L
    MOV     CCAP1H, #HIGH UART3_BitTime    ; 后写 CCAP0H
    MOV     CH, #0
    MOV     CL, #0
    MOV     A, AUXR1
    ANL     A, # NOT(3 SHL 4)
    ORL     A, #PCA_P24_P25_P26_P27         ;切换 I/O 口
    MOV     AUXR1, A
    ANL     A, #NOT(7 SHL 1)
    ORL     A, #PCA_Clock_1T                ;选择时钟源
    MOV     CMOD, A
    SETB    PPCA                            ; 高优先级中断
    SETB    CR                              ; 运行 PCA 定时器
    RET
```

```
;========================================================================
; 函数: F_PCA_Interrupt
; 描述: PCA 中断处理程序.
; 参数: None
; 返回: none.
;========================================================================
F_PCA_Interrupt:
    PUSH    PSW
    PUSH    ACC
;=============== PCA 模块 0 中断 ===================
    JNB     CCF0, L_QuitPCA0                ; PCA 模块 0 中断
    CLR     CCF0                            ; 清 PCA 模块 0 中断标志
    JNB     Rx3_Ring, L_Rx3_Start           ; 已收到起始位

    DJNZ    Rx3_BitCnt, L_RxBit             ; 接收完一帧数据
    CLR     Rx3_Ring                        ; 停止接收

    MOV     Rx3_BUF, Rx3_DAT                ; 存储数据到缓冲区
    SETB    RX3_End
    MOV     CCAPM0, #(PCA_Mode_Capture OR PCA_Fall_Active OR ENABLE)
                                            ; 16 位下降沿捕捉中断模式
    SJMP    L_QuitPCA0

L_RxBit:
    MOV     A, Rx3_DAT                      ; 把接收的单 b 数据  暂存到  RxShiftReg(接收缓冲)
    MOV     C, P_RX3
    RRC     A
    MOV     Rx3_DAT, A
    MOV     A, CCAP0L ;
    ADD     A, #LOW UART3_BitTime           ; 数据位时间
    MOV     CCAP0L, A                       ; 将影射寄存器写入捕获寄存器，先写 CCAP0L
    MOV     A, CCAP0H                       ; 数据位时间
    ADDC    A, #HIGH UART3_BitTime
    MOV     CCAP0H, A                       ; 后写 CCAP0H
    SJMP    L_QuitPCA0

L_Rx3_Start:
    MOV     CCAPM0, #(PCA_Mode_SoftTimer OR ENABLE)     ; 16 位软件定时中断模式
    MOV     A, CCAP0L                       ; 数据位时间
    ADD     A, #LOW (UART3_BitTime / 2 + UART3_BitTime)
    MOV     CCAP0L, A                       ; 将影射寄存器写入捕获寄存器，先写 CCAP0L
    MOV     A, CCAP0H                       ; 数据位时间
    ADDC    A, #HIGH (UART3_BitTime / 2 + UART3_BitTime)
    MOV     CCAP0H, A                       ; 后写 CCAP0H
    SETB    Rx3_Ring                        ; 标志已收到起始位
    MOV     Rx3_BitCnt, #9                  ; 初始化接收的数据位数(8 个数据位+1 个停止位)
```

```
L_QuitPCA0:
;================ PCA 模块 1 中断  ====================
    JNB     CCF1, L_QuitPCA1            ; PCA 模块 1 中断, 16 位软件定时中断模式
    CLR     CCF1                        ; 清 PCA 模块 1 中断标志
    MOV     A, CCAP1L
    ADD     A, #LOW UART3_BitTime       ; 数据位时间
    MOV     CCAP1L, A                   ; 将影射寄存器写入捕获寄存器，先写 CCAP0L
    MOV     A, CCAP1H
    ADDC    A, #HIGH UART3_BitTime      ; 数据位时间
    MOV     CCAP1H, A                   ; 后写 CCAP0H

    JNB     Tx3_Ting, L_QuitPCA1        ; 不发送, 退出
    MOV     A, Tx3_BitCnt
    JNZ     L_TxData                    ; 发送计数器为 0  表明单字节发送还没开始
    CLR     P_TX3                       ; 发送开始位
    MOV     Tx3_DAT, Tx3_BUF            ; 把缓冲的数据放到发送的 buff
    MOV     Tx3_BitCnt, #9              ; 发送数据位数  (8 数据位+1 停止位)
    SJMP    L_QuitPCA1

L_TxData:                               ; 发送计数器为非 0  正在发送数据
    DJNZ    Tx3_BitCnt, L_TxBit         ; 发送计数器减为 0  表明单字节发送结束
    SETB    P_TX3                       ; 送停止位数据
    CLR     Tx3_Ting                    ; 发送停止
    SJMP    L_QuitPCA1

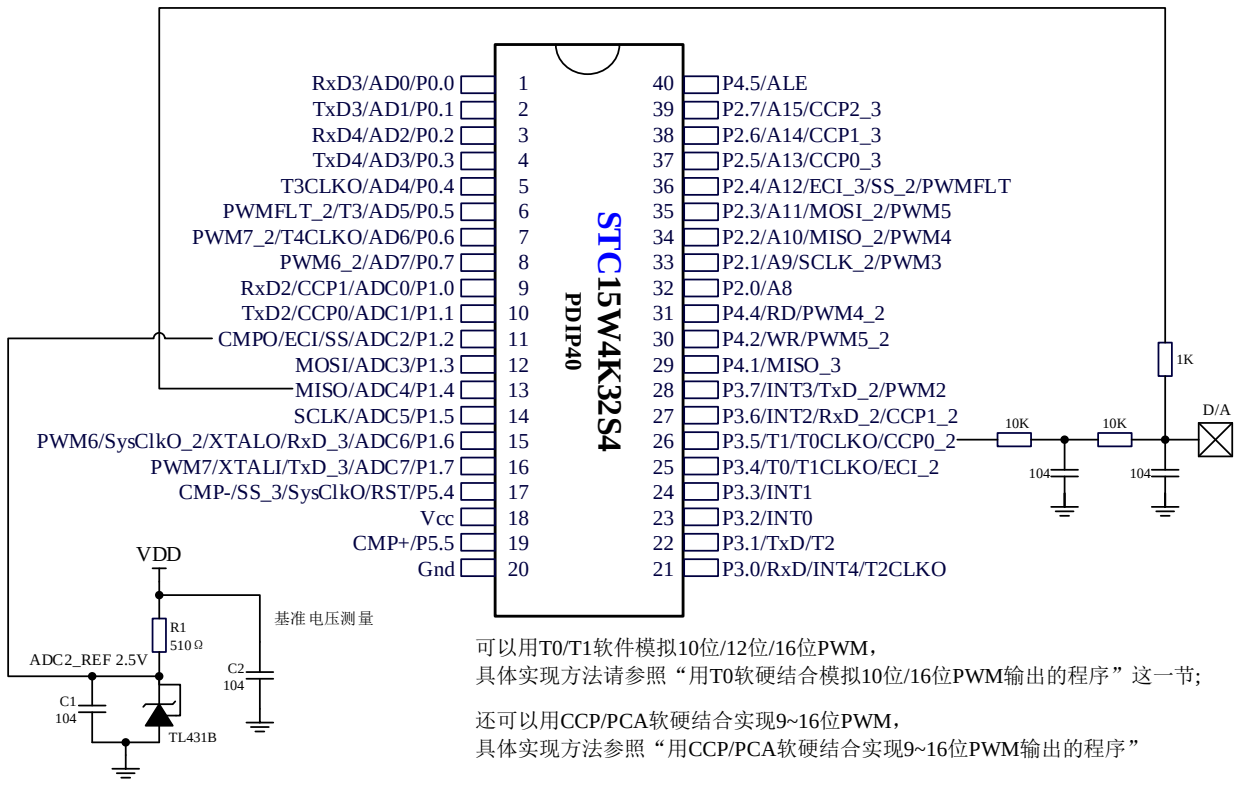
L_TxBit:
    MOV     A, Tx3_DAT                  ; 把最低位送到  CY(益处标志位)
    RRC     A
    MOV     P_TX3, C                    ; 发送一个 bit 数据
    MOV     Tx3_DAT, A

L_QuitPCA1:
    POP     ACC
    POP     PSW
    RETI
END
```

22.15 利用 CCP/PCA 模块实现 8~16 位 DAC 的参考线路图

CCP:是英文单词的缩写

Capture(捕获)，Compare(比较)，PWM(脉宽调制)



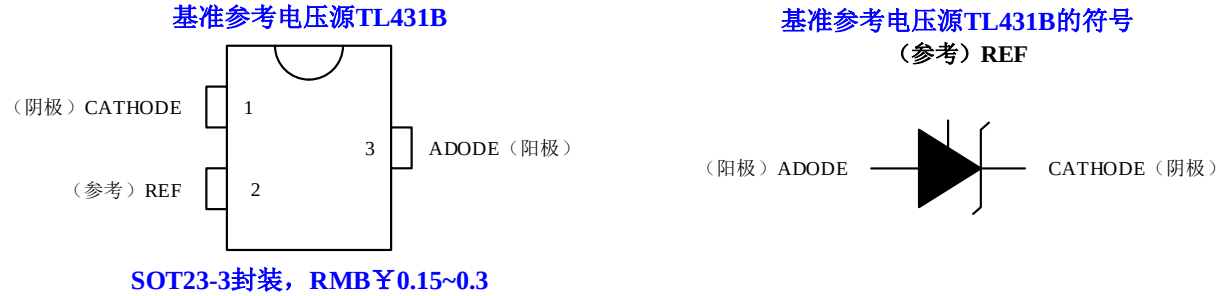
利用 CCP/PCA 模块的高速脉冲输出功能实现 9~16 位 PWM 来实现 9~16 位 DAC，或用本身的硬件 8 位 PWM 来实现 8 位 DAC，单片机本身也有 10 位 ADC。

提示：

（1）PWM 频率越高，输出波形越平滑。

（2）如果工作电压为 5V，需输出 1V 电压，则设置高电平为 1/5，低电平为 4/5，则 PWM 输出电压就为 1V。

（3）如果要输出高精准电压，建议用 A/D 检测输出的电压值，然后根据 A/D 检测的电压值逐步调整到所需要的电压。



SOT23-3封装，RMB￥0.15~0.3

如应用简单，可无需基准参考电压源，直接与Vcc比较即可。

23 STC15W4K32S4 系列新增 6 通道高精度 PWM

----带死区控制的增强型 PWM 波形发生器

STC15W4K32S4 系列的单片机集成了一组(各自独立 6 路)增强型的 PWM 波形发生器。PWM 波形发生器内部有一个 15 位的 PWM 计数器供 6 路 PWM 使用，用户可以设置每路 PWM 的初始电平。另外，PWM 波形发生器为每路 PWM 又设计了两个用于控制波形翻转的计数器 T1/T2，可以非常灵活的每路 PWM 的高低电平宽度，从而达到对 PWM 的占空比以及 PWM 的输出延迟进行控制的目的。由于 6 路 PWM 是各自独立的，且每路 PWM 的初始状态可以进行设定，所以用户可以将其中的任意两路配合起来使用，即可实现互补对称输出以及死区控制等特殊应用。

增强型的 PWM 波形发生器还设计了对外部异常事件(包括外部端口 P2.4 的电平异常、比较器比较结果异常)进行监控的功能，可用于紧急关闭 PWM 输出。PWM 波形发生器还可在 15 位的 PWM 计数器归零时触发外部事件(ADC 转换)。

STC15W4K32S4 系列增强型 PWM 输出端口定义如下：

[PWM2:P3.7, PWM3:P2.1, PWM4:P2.2, PWM5:P2.3, PWM6:P1.6, PWM7:P1.7]

每路 PWM 的输出端口都可使用特殊功能寄存器位 CnPINSEL 分别独立的切换到第二组

[PWM2_2:P2.7, PWM3_2:P4.5, PWM4_2:P4.4, PWM5_2:P4.2, PWM6_2:P0.7, PWM7_2:P0.6]

所有与 PWM 相关的端口，在上电后均为高阻输入态，必须在程序中将这些口设置为准双向口或强推挽模式才可正常输出波形。

端口模式设置相关特殊功能寄存器

符号	描述	地址	位址及符号								初始值
			B7	B6	B5	B4	B3	B2	B1	B0	
P1M1	P1 模式配置 1	91H									0000,0000
P1M0	P1 模式配置 0	92H									0000,0000
P0M1	P0 模式配置 1	93H									0000,0000
P0M0	P0 模式配置 0	94H									0000,0000
P2M1	P2 模式配置 1	95H									0000,0000
P2M0	P2 模式配置 0	96H									0000,0000
P3M1	P3 模式配置 1	B1H									0000,0000
P3M0	P3 模式配置 0	B2H									0000,0000
P4M1	P4 模式配置 1	B3H									0000,0000
P4M0	P4 模式配置 0	B4H									0000,0000

端口模式设置

PxM1	PxM0	模式
0	0	准双向口
0	1	强推挽输出
1	0	高阻输入
1	1	开漏输出

若需要正常使用与 PWM 相关的端口，则需要将相应的端口设置为准双向口或强推挽输出口。
例如将端口均设置为准双向口的汇编代码如下：

```
MOV     P0M0, #00H
MOV     P0M1, #00H
MOV     P1M0, #00H
MOV     P1M1, #00H
MOV     P2M0, #00H
MOV     P2M1, #00H
MOV     P3M0, #00H
MOV     P3M1, #00H
MOV     P4M0, #00H
MOV     P4M1, #00H
```

23.1 增强型 PWM 波形发生器相关功能寄存器

增强型 PWM 波形发生器相关的特殊功能寄存器

符号	描述	地址	位址及符号								初始值
			B7	B6	B5	B4	B3	B2	B1	B0	
P_SW2	端口配置寄存器	BAH	EAXSFR	DBLPWR	P31PU	P30PU	-	S4_S	S3_S	S2_S	0000,0000
PWMCFG	PWM 配置	F1H	-	CBTADC	C7INI	C6INI	C5INI	C4INI	C3INI	C2INI	0000,0000
PWMCR	PWM 控制	F5H	ENPWM	ECBI	ENC7O	ENC6O	ENC5O	ENC4O	ENC3O	ENC2O	0000,0000
PWMIF	PWM 中断标志	F6H	-	CBIF	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	x000,0000
PWMFDCR	PWM 外部异常控制	F7H	-	-	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	xx00,0000
PWMCH	PWM 计数器高位	FFF0H	-	PWMCH[14:8]							x000,0000
PWMCL	PWM 计数器低位	FFF1H	PWMCL[7:0]								0000,0000
PWMCKS	PWM 时钟选择	FFF2H	-	-	-	SELT2	PS[3:0]				xxx0,0000
PWM2T1H	PWM2T1 计数高位	FF00H	-	PWM2T1H[14:8]							x000,0000
PWM2T1L	PWM2T1 计数低位	FF01H	PWM2T1L[7:0]								0000,0000
PWM2T2H	PWM2T2 计数高位	FF02H	-	PWM2T2H[14:8]							x000,0000
PWM2T2L	PWM2T2 计数低位	FF03H	PWM2T2L[7:0]								0000,0000
PWM2CR	PWM2 控制	FF04H	-	-	-	-	PWM2_PS	EPWM2I	EC2T2SI	EC2T1SI	xxxx,0000
PWM3T1H	PWM3T1 计数高位	FF10H	-	PWM3T1H[14:8]							x000,0000
PWM3T1L	PWM3T1 计数低位	FF11H	PWM3T1L[7:0]								0000,0000
PWM3T2H	PWM3T2 计数高位	FF12H	-	PWM3T2H[14:8]							x000,0000
PWM3T2L	PWM3T2 计数低位	FF13H	PWM3T2L[7:0]								0000,0000
PWM3CR	PWM3 控制	FF14H	-	-	-	-	PWM3_PS	EPWM3I	EC3T2SI	EC3T1SI	xxxx,0000
PWM4T1H	PWM4T1 计数高位	FF20H	-	PWM4T1H[14:8]							x000,0000
PWM4T1L	PWM4T1 计数低位	FF21H	PWM4T1L[7:0]								0000,0000
PWM4T2H	PWM4T2 计数高位	FF22H	-	PWM4T2H[14:8]							x000,0000
PWM4T2L	PWM4T2 计数低位	FF23H	PWM4T2L[7:0]								0000,0000
PWM4CR	PWM4 控制	FF24H	-	-	-	-	PWM4_PS	EPWM4I	EC4T2SI	EC4T1SI	xxxx,0000
PWM5T1H	PWM5T1 计数高位	FF30H	-	PWM5T1H[14:8]							x000,0000
PWM5T1L	PWM5T1 计数低位	FF31H	PWM5T1L[7:0]								0000,0000
PWM5T2H	PWM5T2 计数高位	FF32H	-	PWM5T2H[14:8]							x000,0000

符号	描述	地址	位址及符号								初始值
			B7	B6	B5	B4	B3	B2	B1	B0	
PWM5T2L	PWM5T2 计数低位	FF33H	PWM5T2L[7:0]								0000,0000
PWM5CR	PWM5 控制	FF34H	-	-	-	-	PWM5_PS	EPWM5I	EC5T2SI	EC5T1SI	xxxx,0000
PWM6T1H	PWM6T1 计数高位	FF40H	-	PWM6T1H[14:8]							x000,0000
PWM6T1L	PWM6T1 计数低位	FF41H	PWM6T1L[7:0]								0000,0000
PWM6T2H	PWM6T2 计数高位	FF42H	-	PWM6T2H[14:8]							x000,0000
PWM6T2L	PWM6T2 计数低位	FF43H	PWM6T2L[7:0]								0000,0000
PWM6CR	PWM6 控制	FF44H	-	-	-	-	PWM6_PS	EPWM6I	EC6T2SI	EC6T1SI	xxxx,0000
PWM7T1H	PWM7T1 计数高位	FF50H	-	PWM7T1H[14:8]							x000,0000
PWM7T1L	PWM7T1 计数低位	FF51H	PWM7T1L[7:0]								0000,0000
PWM7T2H	PWM7T2 计数高位	FF52H	-	PWM7T2H[14:8]							x000,0000
PWM7T2L	PWM7T2 计数低位	FF53H	PWM7T2L[7:0]								0000,0000
PWM7CR	PWM7 控制	FF54H	-	-	-	-	PWM7_PS	EPWM7I	EC7T2SI	EC7T1SI	xxxx,0000

1、端口配置寄存器：P_SW2

端口配置寄存器的格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
P_SW2	BAH	name	EAXSFR	DBLPWR	P31PU	P30PU	-	S4_S	S3_S	S2_S	0000,0000B

EAXSFR：扩展 SFR 访问控制使能

- 0：MOVX A, @DPTR / MOVX @DPTR, A 指令的操作对象为扩展 RAM（XRAM）
- 1：MOVX A, @DPTR / MOVX @DPTR, A 指令的操作对象为扩展 SFR（XSFR）

  注意：若要访问 PWM 在扩展 RAM 区的特殊功能寄存器，必须先将 EAXSFR 位置为 1。

2、PWM 配置寄存器：PWMCFG

PWM 配置寄存器的格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWMCFG	F1H	name	-	CBTADC	C7INI	C6INI	C5INI	C4INI	C3INI	C2INI	0000,0000

CBTADC：PWM 计数器归零时(CBIF == 1 时)触发 ADC 转换

- 0：PWM 计数器归零时不触发 ADC 转换
- 1：PWM 计数器归零时自动触发 ADC 转换。

  (注：前提条件是 PWM 和 ADC 必须被使能，即 ENPWM==1，且 ADCON==1)

C7INI： 设置 PWM7 输出端口的初始电平

- 0：PWM7 输出端口的初始电平为低电平
- 1：PWM7 输出端口的初始电平为高电平

C6INI： 设置 PWM6 输出端口的初始电平

- 0：PWM6 输出端口的初始电平为低电平
- 1：PWM6 输出端口的初始电平为高电平

C5INI： 设置 PWM5 输出端口的初始电平

- 0：PWM5 输出端口的初始电平为低电平
- 1：PWM5 输出端口的初始电平为高电平

C4INI： 设置 PWM4 输出端口的初始电平

- 0：PWM4 输出端口的初始电平为低电平
- 1：PWM4 输出端口的初始电平为高电平

C3INI： 设置 PWM3 输出端口的初始电平

- 0：PWM3 输出端口的初始电平为低电平
- 1：PWM3 输出端口的初始电平为高电平

C2INI： 设置 PWM2 输出端口的初始电平

- 0：PWM2 输出端口的初始电平为低电平
- 1：PWM2 输出端口的初始电平为高电平

3、PWM 控制寄存器：PWMCR

PWM 控制寄存器的格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWMCR	F5H	name	ENPWM	ECBI	ENC7O	ENC6O	ENC5O	ENC4O	ENC3O	ENC2O	0000,0000B

ENPWM：使能增强型 PWM 波形发生器

- 0：3 关闭 PWM 波形发生器
- 1：使能 PWM 波形发生器，PWM 计数器开始计数

【关于 ENPWM 控制位的重要说明】:

1、ENPWMI 一旦被使能后，内部的 PWM 计数器会立即开始计数，并与 T1/T2 两个翻转点的值进行比较。所有 ENPWM 必须在其他所有的 PWMI 设置(包括 T1/T2 翻转点的设置、初始电平的设置、PWM 异常检测的设置以及 PWM 中断设置)都完成后，最后才能使能 ENPWM 位。

2、ENPWMI 控制位既是整个 PWM 模块的的使能位，也是 PWM 计数器开始计数的控制位。在 PWM 计数器计数的过程中，ENPWM 控制位被关闭时，PWM 计数会立即停止，当再次使能 ENPWM 控制位时，PWM 的计数会从 0 开始重新计数，而不会记忆 PWM 停止计数前的计数值

ECBI： PWM 计数器归零中断使能位

- 0：关闭 PWM 计数器归零中断(CBIF 依然会被硬件置位)
- 1：使能 PWM 计数器归零中断

ENC7O：PWM7 输出使能位

- 0：PWM 通道 7 的端口为 GPIO
- 1：PWM 通道 7 的端口为 PWM 输出口，受 PWM 波形发生器控制

ENC6O：PWM6 输出使能位

- 0：PWM 通道 6 的端口为 GPIO
- 1：PWM 通道 6 的端口为 PWM 输出口，受 PWM 波形发生器控制

ENC5O：PWM5 输出使能位

- 0：PWM 通道 5 的端口为 GPIO
- 1：PWM 通道 5 的端口为 PWM 输出口，受 PWM 波形发生器控制

ENC4O：PWM4 输出使能位

- 0：PWM 通道 4 的端口为 GPIO
- 1：PWM 通道 4 的端口为 PWM 输出口，受 PWM 波形发生器控制

ENC3O：PWM3 输出使能位

- 0：PWM 通道 3 的端口为 GPIO
- 1：PWM 通道 3 的端口为 PWM 输出口，受 PWM 波形发生器控制

ENC2O：PWM2 输出使能位

- 0：PWM 通道 2 的端口为 GPIO
- 1：PWM 通道 2 的端口为 PWM 输出口，受 PWM 波形发生器控制

4、PWM 中断标志寄存器：PWMIF

PWM 中断标志寄存器的格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWMIF	F6H	name	-	CBIF	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	x000,0000B

CBIF： PWM 计数器归零中断标志位

- 当 PWM 计数器归零时，硬件自动将此位置 1。当 ECBI==1 时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

C7IF： 第 7 通道的 PWM 中断标志位

- 可设置在翻转点 1 和翻转点 2 触发 C7IF(详见 EC7TISI 和 EC7T2SI)。当 PWM 发生翻转时，硬件自动将此位置 1。当 EPWM7I==1 时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

C6IF： 第 6 通道的 PWM 中断标志位

- 可设置在翻转点 1 和翻转点 2 触发 C6IF(详见 EC6TISI 和 EC6T2SI)。当 PWM 发生翻转时，硬件自动将此位置 1。当 EPWM6I==1 时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

C5IF： 第 5 通道的 PWM 中断标志位

- 可设置在翻转点 1 和翻转点 2 触发 C5IF(详见 EC5TISI 和 EC5T2SI)。当 PWM 发生翻转时，硬件自动将此位置 1。当 EPWM5I==1 时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

C4IF： 第 4 通道的 PWM 中断标志位

- 可设置在翻转点 1 和翻转点 2 触发 C4IF(详见 EC4TISI 和 EC4T2SI)。当 PWM 发生翻转时，硬件自动将此位置 1。当 EPWM4I==1 时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

C3IF： 第 3 通道的 PWM 中断标志位

- 可设置在翻转点 1 和翻转点 2 触发 C3IF(详见 EC3TISI 和 EC3T2SI)。当 PWM 发生翻转时，硬件自动将此位置 1。当 EPWM3I==1 时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

C2IF： 第 2 通道的 PWM 中断标志位

- 可设置在翻转点 1 和翻转点 2 触发 C2IF(详见 EC2TISI 和 EC2T2SI)。当 PWM 发生翻转时，硬件自动将此位置 1。当 EPWM2I==1 时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

5、PWM 外部异常控制寄存器：PWMFDCR

PWM 外部异常控制寄存器的格式如下

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWMFDCR	F7H	name	-	-	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	xx00,0000B

ENFD： PWM 外部异常检测功能控制位

- 0：关闭 PWM 的外部异常检测功能
- 1：使能 PWM 的外部异常检测功能

FLTFLIO：发生 PWM 外部异常时对 PWM 输出口控制位

- 0：发生 PWM 外部异常时，PWM 的输出口不作任何改变
- 1：发生 PWM 外部异常时，PWM 的输出口立即被设置为高阻输入模式(既不对外输出电流，也不对内输出电流)。(注：只有 ENCnO==1 所对应的端口才会被强制悬空；当 PWM 外部异常状态取消时，相应的 PWM 的输出口会自动恢复以前的 I/O 设置)。

EFDI： PWM 异常检测中断使能位

- 0：关闭 PWM 异常检测中断(FDIF 依然会被硬件置位)
- 1：使能 PWM 异常检测中断

FDCMP：设定 PWM 异常检测源为比较器的输出

- 0：比较器与 PWM 无关
- 1：当比较器正极 P5.5/CMP+的电平比比较器负极 P5.4/CMP-的电平高或者比较器正极 P5.5/CMP+的电平比内部参考电压源 1.28V 高时，触发 PWM 异常。

FDIO： 设定 PWM 异常检测源为端口 P2.4 的状态

- 0：P2.4 的状态与 PWM 无关
- 1：当 P2.4 的电平为高时，触发 PWM 异常

FDIF： PWM 异常检测中断标志位

- 当发生 PWM 异常(比较器正极 P5.5/CMP+的电平比比较器负极 P5.4/CMP-的电平高，或比较器正极 P5.5/CMP+的电平比内部参考电压源 1.28V 高或者 P2.4 的电平为高)时，硬件自动将此位置 1。当 EFDI==1 时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

6、PWM 计数器

PWM 计数器高字节：PWMCH(高 7 位)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWMCH	FFF0H (XSFR)	name	-	PWMCH[14：8]							x000,0000B

PWM 计数器低字节：PWMCL（低 8 位）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWMCL	FFF1H (XSFR)	name	PWMCL[7：0]								0000,0000B

PWM 计数器位一个 15 位的寄存器，可设定 1～32767 之间的任意值作为 PWM 的周期。PWM 波形发生器内部的计数器从 0 开始计数，每个 PWM 时钟周期递增 1，当内部计数器的计数值达到[PWMCH，PWMCL]所设定的 PWM 周期时，PWM 波形发生器内部的计数器将会从 0 重新开始开始计数，硬件会自动将 PWM 归零中断中断标志位 CBIF 置 1，若 ECBI==1，程序将跳转到相应中断入口执行中断服务程序。

7、PWM 时钟选择寄存器：PWMCKS

PWM 时钟选择寄存器的格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWMCKS	FFF2H (XSFR)	name	-	-	-	SELT2	PS[3：0]				0000,0000B

SELT2： PWM 时钟源选择

- 0：PWM 时钟源为系统时钟经分频器分频之后的时钟
- 1：PWM 时钟源为定时器 2 的溢出脉冲

PS[3：0]：系统时钟预分频参数。当 SELT2==0 时，PWM 时钟为系统时钟 / (PS[3：0]+1)

8、PWM2 的翻转计数器

PWM2 的第一次翻转计数器的高字节：PWM2T1H

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM2T1H	FF00H (XSFR)	name	-	PWM2T1H[14：8]							x000,0000B

PWM2 的第一次翻转计数器的低字节：PWM2T1L

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM2T1L	FF01H (XSFR)	name	PWM2T1L[7：0]								0000,0000B

PWM2 的第二次翻转计数器的高字节：PWM2T2H

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM2T2H	FF02H (XSFR)	name	-	PWM2T2H[14：8]							x000,0000B

PWM2 的第二次翻转计数器的低字节：PWM2T2L

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM2T2L	FF03H (XSFR)	name	PWM2T2L[7：0]								0000,0000B

PWM 波形发生器设计了两个用于控制 PWM 波形翻转的 15 位计数器，可设定 1 ~ 32767 之间的任意值。PWM 波形发生器内部的计数器的计数值与 T1/T2 所设定的值相匹配时，PWM 的输出波形将发生翻转。

9、PWM2 的控制寄存器：PWM2CR

PWM2 的控制寄存器的格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM2CR	FF04H (XSFR)	name	-	-	-	-	PWM2_PS	EPWM2I	EC2T2SI	EC2T1SI	xxxx,0000B

PWM2_PS：PWM2 输出管脚选择位

- 0：PWM2 的输出管脚为 PWM2：P3.7
- 1：PWM2 的输出管脚为 PWM2_2：P2.7

EPWM2I：PWM2 中断使能控制位

- 0：关闭 PWM2 中断

- 1：使能 PWM2 中断，当 C2IF 被硬件置 1 时，程序将跳转到相应中断入口执行中断服务程序。

EC2T2SI：PWM2 的 T2 匹配发生波形翻转时的中断控制位

- 0：关闭 T2 翻转时中断
- 1：使能 T2 翻转时中断，当 PWM 波形发生器内部计数值与 T2 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C2IF 置 1，此时若 EPWM2I==1，则程序将跳转到相应中断入口执行中断服务程序。

EC2T1SI：PWM2 的 T1 匹配发生波形翻转时的中断控制位

- 0：关闭 T1 翻转时中断
- 1：使能 T1 翻转时中断，当 PWM 波形发生器内部计数值与 T1 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C2IF 置 1，此时若 EPWM2I==1，则程序将跳转到相应中断入口执行中断服务程序。

10、PWM3 的翻转计数器

PWM3 的第一次翻转计数器的高字节：PWM3T1H

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM3T1H	FF10H (XSFR)	name	-	PWM3T1H[14：8]							x000,0000B

PWM3 的第一次翻转计数器的低字节：PWM3T1L

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM3T1L	FF11H (XSFR)	name	PWM3T1L[7：0]								0000,0000B

PWM3 的第二次翻转计数器的高字节：PWM3T2H

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM3T2H	FF12H (XSFR)	name	-	PWM3T2H[14：8]							x000,0000B

PWM3 的第二次翻转计数器的低字节：PWM3T2L

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM3T2L	FF13H (XSFR)	name	PWM3T2L[7：0]								0000,0000B

PWM 波形发生器设计了两个用于控制 PWM 波形翻转的 15 位计数器，可设定 1 ~ 32767 之间的任意值。PWM 波形发生器内部的计数器的计数值与 T1/T2 所设定的值相匹配时，PWM 的输出波形将发生翻转。

11、PWM3 的控制寄存器：PWM3CR

PWM3 的控制寄存器的格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM3CR	FF14H (XSFR)	name	-	-	-	-	PWM3_PS	EPWM3I	EC3T2SI	EC3T1SI	xxxx,0000B

PWM3_PS：PWM3 输出管脚选择位

- 0：PWM3 的输出管脚为 PWM3：P2.1
- 1：PWM3 的输出管脚为 PWM3_2：P4.5

EPWM3I：PWM3 中断使能控制位

- 0：关闭 PWM3 中断
- 1：使能 PWM3 中断，当 C3IF 被硬件置 1 时，程序将跳转到相应中断入口执行中断服务程序。

EC3T2SI：PWM3 的 T2 匹配发生波形翻转时的中断控制位

- 0：关闭 T2 翻转时中断
- 1：使能 T2 翻转时中断，当 PWM 波形发生器内部计数值与 T2 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C3IF 置 1，此时若 EPWM3I==1，则程序将跳转到相应中断入口执行中断服务程序。

EC3T1SI：PWM3 的 T1 匹配发生波形翻转时的中断控制位

- 0：关闭 T1 翻转时中断
- 1：使能 T1 翻转时中断，当 PWM 波形发生器内部计数值与 T1 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C3IF 置 1，此时若 EPWM3I==1，则程序将跳转到相应中断入口执行中断服务程序。

12、PWM4 的翻转计数器

PWM4 的第一次翻转计数器的高字节：PWM4T1H

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM4T1H	FF20H (XSFR)	name	-	PWM4T1H[14：8]							x000,0000B

PWM4 的第一次翻转计数器的低字节：PWM4T1L

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM4T1L	FF21H (XSFR)	name	PWM4T1L[7：0]								0000,0000B

PWM4 的第二次翻转计数器的高字节：PWM4T2H

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM4T2H	FF22H (XSFR)	name	-	PWM4T2H[14：8]							x000,0000B

PWM4 的第二次翻转计数器的低字节：PWM4T2L

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM4T2L	FF23H (XSFR)	name	PWM4T2L[7：0]								0000,0000B

PWM 波形发生器设计了两个用于控制 PWM 波形翻转的 15 位计数器，可设定 1 ~ 32767 之间的任意值。PWM 波形发生器内部的计数器的计数值与 T1/T2 所设定的值相匹配时，PWM 的输出波形将发生翻转。

13、PWM4 的控制寄存器：PWM4CR

PWM4 的控制寄存器的格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM4CR	FF24H (XSFR)	name	-	-	-	-	PWM4_PS	EPWM4I	EC4T2SI	EC4T1SI	xxxx,0000B

PWM4_PS：PWM4 输出管脚选择位

- 0：PWM4 的输出管脚为 PWM4：P2.2
- 1：PWM4 的输出管脚为 PWM4_2：P4.4

EPWM4I：PWM4 中断使能控制位

- 0：关闭 PWM4 中断
- 1：使能 PWM4 中断，当 C4IF 被硬件置 1 时，程序将跳转到相应中断入口执行中断服务程序。

EC4T2SI：PWM4 的 T2 匹配发生波形翻转时的中断控制位

- 0：关闭 T2 翻转时中断
- 1：使能 T2 翻转时中断，当 PWM 波形发生器内部计数值与 T2 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C4IF 置 1，此时若 EPWM4I==1，则程序将跳转到相应中断入口执行中断服务程序。

EC4T1SI：PWM4 的 T1 匹配发生波形翻转时的中断控制位

- 0：关闭 T1 翻转时中断
- 1：使能 T1 翻转时中断，当 PWM 波形发生器内部计数值与 T1 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C4IF 置 1，此时若 EPWM4I==1，则程序将跳转到相应中断入口执行中断服务程序。

14、PWM5 的翻转计数器

PWM5 的第一次翻转计数器的高字节：PWM5T1H

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM5T1H	FF30H (XSFR)	name	-	PWM5T1H[14：8]							x000,0000B

PWM5 的第一次翻转计数器的低字节：PWM5T1L

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM5T1L	FF31H (XSFR)	name	PWM5T1L[7：0]								0000,0000B

15、PWM5 的控制寄存器：PWM5CR

PWM5 的控制寄存器的格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM5CR	FF34H (XSFR)	name	-	-	-	-	PWM5_PS	EPWM5I	EC5T2SI	EC5T1SI	xxxx,0000B

PWM5_PS：PWM5 输出管脚选择位

- 0：PWM5 的输出管脚为 PWM5：P2.3
- 1：PWM5 的输出管脚为 PWM5_2：P4.2

EPWM5I：PWM5 中断使能控制位

- 0：关闭 PWM5 中断
- 1：使能 PWM5 中断，当 C5IF 被硬件置 1 时，程序将跳转到相应中断入口执行中断服务程序。

EC5T2SI：PWM5 的 T2 匹配发生波形翻转时的中断控制位

- 0：关闭 T2 翻转时中断
- 1：使能 T2 翻转时中断，当 PWM 波形发生器内部计数值与 T2 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C5IF 置 1，此时若 EPWM5I==1，则程序将跳转到相应

中断入口执行中断服务程序。

EC5T1SI：PWM5 的 T1 匹配发生波形翻转时的中断控制位

- 0：关闭 T1 翻转时中断
- 1：使能 T1 翻转时中断，当 PWM 波形发生器内部计数值与 T1 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C5IF 置 1，此时若 EPWM5I==1，则程序将跳转到相应中断入口执行中断服务程序。

16、PWM6 的翻转计数器

PWM6 的第一次翻转计数器的高字节：PWM6T1H

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM6T1H	FF40H (XSFR)	name	-	PWM6T1H[14：8]							x000,0000B

PWM6 的第一次翻转计数器的低字节：PWM6T1L

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM6T1L	FF41H (XSFR)	name	PWM6T1L[7：0]								0000,0000B

PWM6 的第二次翻转计数器的高字节：PWM6T2H

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM6T2H	FF42H (XSFR)	name	-	PWM6T2H[14：8]							x000,0000B

PWM6 的第二次翻转计数器的低字节：PWM6T2L

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM6T2L	FF43H (XSFR)	name	PWM6T2L[7：0]								0000,0000B

PWM 波形发生器设计了两个用于控制 PWM 波形翻转的 15 位计数器，可设定 1 ~ 32767 之间的任意值。PWM 波形发生器内部的计数器的计数值与 T1/T2 所设定的值相匹配时，PWM 的输出波形将发生翻转。

17、PWM6 的控制寄存器：PWM6CR

PWM6 的控制寄存器的格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM6CR	FF44H (XSFR)	name	-	-	-	-	PWM6_PS	EPWM6I	EC6T2SI	EC6T1SI	xxxx,0000B

PWM6_PS：PWM6 输出管脚选择位

- 0：PWM6 的输出管脚为 PWM6：P1.6
- 1：PWM6 的输出管脚为 PWM6_2：P0.7

EPWM6I：PWM6 中断使能控制位

- 0：关闭 PWM6 中断
- 1：使能 PWM6 中断，当 C6IF 被硬件置 1 时，程序将跳转到相应中断入口执行中断服务程序。

EC6T2SI：PWM6 的 T2 匹配发生波形翻转时的中断控制位

- 0：关闭 T2 翻转时中断

- 1：使能 T2 翻转时中断，当 PWM 波形发生器内部计数值与 T2 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C6IF 置 1，此时若 EPWM6I==1，则程序将跳转到相应中断入口执行中断服务程序。

EC6T1SI：PWM6 的 T1 匹配发生波形翻转时的中断控制位

- 0：关闭 T1 翻转时中断
- 1：使能 T1 翻转时中断，当 PWM 波形发生器内部计数值与 T1 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C6IF 置 1，此时若 EPWM6I==1，则程序将跳转到相应中断入口执行中断服务程序。

18、PWM7 的翻转计数器

PWM7 的第一次翻转计数器的高字节：PWM7T1H

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM7T1H	FF50H (XSFR)	name	-	PWM7T1H[14：8]							x000,0000B

PWM7 的第一次翻转计数器的低字节：PWM7T1L

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM7T1L	FF51H (XSFR)	name	PWM7T1L[7：0]								0000,0000B

PWM7 的第二次翻转计数器的高字节：PWM7T2H

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM7T2H	FF52H (XSFR)	name	-	PWM7T2H[14：8]							x000,0000B

PWM7 的第二次翻转计数器的低字节：PWM7T2L

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM7T2L	FF53H (XSFR)	name	PWM7T2L[7：0]								0000,0000B

PWM 波形发生器设计了两个用于控制 PWM 波形翻转的 15 位计数器，可设定 1 ~ 32767 之间的任意值。PWM 波形发生器内部的计数器的计数值与 T1/T2 所设定的值相匹配时，PWM 的输出波形将发生翻转。

19、PWM7 的控制寄存器：PWM7CR

PWM7 的控制寄存器的格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM7CR	FF54H (XSFR)	name	-	-	-	-	PWM7_PS	EPWM7I	EC7T2SI	EC7T1SI	xxxx,0000B

PWM7_PS：PWM7 输出管脚选择位

- 0：PWM7 的输出管脚为 PWM7：P1.7
- 1：PWM7 的输出管脚为 PWM7_2：P0.6

EPWM7I：PWM7 中断使能控制位

- 0：关闭 PWM7 中断
- 1：使能 PWM7 中断，当 C7IF 被硬件置 1 时，程序将跳转到相应中断入口执行中断服务程序。

EC7T2SI：PWM7 的 T2 匹配发生波形翻转时的中断控制位

- 0：关闭 T2 翻转时中断
- 1：使能 T2 翻转时中断，当 PWM 波形发生器内部计数值与 T2 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C7IF 置 1，此时若 EPWM7I==1，则程序将跳转到相应中断入口执行中断服务程序。

EC7T1SI：PWM7 的 T1 匹配发生波形翻转时的中断控制位

- 0：关闭 T1 翻转时中断
- 1：使能 T1 翻转时中断，当 PWM 波形发生器内部计数值与 T1 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C7IF 置 1，此时若 EPWM7I==1，则程序将跳转到相应中断入口执行中断服务程序。

23.2 增强型 PWM 波形发生器的中断控制

PWM 波形发生器中断相关的特殊功能寄存器

符号	描述	地址	位址及符号								初始值
			B7	B6	B5	B4	B3	B2	B1	B0	
IP2	中断优先级控制	B5H	-	-	-	PX4	PPWMFD	PPWM	PSPI	PS2	xxx0,0000
PWMCR	PWM 控制	F5H	ENPWM	ECBI	ENC7O	ENC6O	ENC5O	ENC4O	ENC3O	ENC2O	0000,0000
PWMIF	PWM 中断标志	F6H	-	CBIF	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	x000,0000
PWMFDCR	PWM 外部异常控制	F7H	-	-	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	xx00,0000
PWM2CR	PWM2 控制	FF04H	-	-	-	-	PWM2_PS	EPWM2I	EC2T2SI	EC2T1SI	xxxx,0000
PWM3CR	PWM3 控制	FF14H	-	-	-	-	PWM3_PS	EPWM3I	EC3T2SI	EC3T1SI	xxxx,0000
PWM4CR	PWM4 控制	FF24H	-	-	-	-	PWM4_PS	EPWM4I	EC4T2SI	EC4T1SI	xxxx,0000
PWM5CR	PWM5 控制	FF34H	-	-	-	-	PWM5_PS	EPWM5I	EC5T2SI	EC5T1SI	xxxx,0000
PWM6CR	PWM6 控制	FF44H	-	-	-	-	PWM6_PS	EPWM6I	EC6T2SI	EC6T1SI	xxxx,0000
PWM7CR	PWM7 控制	FF54H	-	-	-	-	PWM7_PS	EPWM7I	EC7T2SI	EC7T1SI	xxxx,0000

1、PWM 中断优先级控制寄存器：IP2

IP2：中断优先级控制寄存器 （不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
IP2	B5H	name	-	-	-	PX4	PPWMFD	PPWM	PSPI	PS2	0000,0000B

PPWMFD：PWM 异常检测中断优先级控制位。

- 当 PPWMFD=0 时，PWM 异常检测中断为最低优先级中断（优先级 0）
- 当 PPWMFD=1 时，PWM 异常检测中断为最高优先级中断（优先级 1）

PPWM：PWM 中断优先级控制位。

- 当 PPWM=0 时，PWM 中断为最低优先级中断（优先级 0）
- 当 PPWM=1 时，PWM 中断为最高优先级中断（优先级 1）

中断优先级控制寄存器 IP 和 IP2 的各位都由可用户程序置“1”和清“0”。但 IP 寄存器可位操作，所以可用位操作指令或字节操作指令更新 IP 的内容。而 IP2 寄存器的内容只能用字节操作指令来更新。STC15 系列单片机复位后 IP 和 IP2 均为 00H，各个中断源均为低优先级中断。

2、PWM 控制寄存器：PWMCR

PWM 控制寄存器的格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWMCR	F5H	name	ENPWM	ECBI	ENC70	ENC60	ENC50	ENC40	ENC30	ENC20	0000,0000B

ECBI： PWM 计数器归零中断使能位

- 0：关闭 PWM 计数器归零中断（CBIF 依然会被硬件置位）
- 1：使能 PWM 计数器归零中断

3、PWM 中断标志寄存器：PWMIF

PWM 中断标志寄存器的格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWMIF	F6H	name	-	CBIF	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	x000,0000B

CBIF： PWM 计数器归零中断标志位

- 当 PWM 计数器归零时，硬件自动将此位置 1。当 ECBI==1 时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

C7IF： 第 7 通道的 PWM 中断标志位

- 可设置在翻转点 1 和翻转点 2 触发 C7IF（详见 EC7T1SI 和 EC7T2SI）。当 PWM 发生翻转时，硬件自动将此位置 1。当 EPWM7I==1 时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

C6IF： 第 6 通道的 PWM 中断标志位

- 可设置在翻转点 1 和翻转点 2 触发 C6IF（详见 EC6T1SI 和 EC6T2SI）。当 PWM 发生翻转时，硬件自动将此位置 1。当 EPWM6I==1 时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

C5IF： 第 5 通道的 PWM 中断标志位

- 可设置在翻转点 1 和翻转点 2 触发 C5IF（详见 EC5T1SI 和 EC5T2SI）。当 PWM 发生翻转时，硬件自动将此位置 1。当 EPWM5I==1 时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

C4IF： 第 4 通道的 PWM 中断标志位

- 可设置在翻转点 1 和翻转点 2 触发 C4IF（详见 EC4T1SI 和 EC4T2SI）。当 PWM 发生翻转时，硬件自动将此位置 1。当 EPWM4I==1 时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

C3IF： 第 3 通道的 PWM 中断标志位

- 可设置在翻转点 1 和翻转点 2 触发 C3IF（详见 EC3T1SI 和 EC3T2SI）。当 PWM 发生翻转时，硬件自动将此位置 1。当 EPWM3I==1 时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

C2IF： 第 2 通道的 PWM 中断标志位

- 可设置在翻转点 1 和翻转点 2 触发 C2IF（详见 EC2T1SI 和 EC2T2SI）。当 PWM 发生翻转时，硬件自动将此位置 1。当 EPWM2I==1 时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

4、PWM 外部异常控制寄存器：PWMFDCR

PWM 外部异常控制寄存器的格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWMFDCR	F7H	name	-	-	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	xx00,0000B

EFDI： PWM 异常检测中断使能位

- 0：关闭 PWM 异常检测中断（FDIF 依然会被硬件置位）
- 1：使能 PWM 异常检测中断

FDIF： PWM 异常检测中断标志位

- 当发生 PWM 异常（比较器正极 P5.5/CMP+的电平比比较器负极 P5.4/CMP-的电平高或比较器正极 P5.5/CMP+的电平比内部参考电压源 1.28V 高或者 P2.4 的电平为高）时，硬件自动将此位置 1。当 EFDI==1 时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

5、PWM2 的控制寄存器：PWM2CR

PWM2 的控制寄存器的格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM2CR	FF04H（XSFR）	name	-	-	-	-	PWM2_PS	EPWM2I	EC2T2SI	EC2T1SI	xxxx,0000B

EPWM2I：PWM2 中断使能控制位

- 0：关闭 PWM2 中断
- 1：使能 PWM2 中断，当 C2IF 被硬件置 1 时，程序将跳转到相应中断入口执行中断服务程序。

EC2T2SI：PWM2 的 T2 匹配发生波形翻转时的中断控制位

- 0：关闭 T2 翻转时中断
- 1：使能 T2 翻转时中断，当 PWM 波形发生器内部计数值与 T2 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C2IF 置 1，此时若 EPWM2I==1，则程序将跳转到相应中断入口执行中断服务程序。

EC2T1SI：PWM2 的 T1 匹配发生波形翻转时的中断控制位

- 0：关闭 T1 翻转时中断
- 1：使能 T1 翻转时中断，当 PWM 波形发生器内部计数值与 T1 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C2IF 置 1，此时若 EPWM2I==1，则程序将跳转到相应中断入口执行中断服务程序。

6、PWM3 的控制寄存器：PWM3CR

PWM3 的控制寄存器的格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM3CR	FF14H（XSFR）	name	-	-	-	-	PWM3_PS	EPWM3I	EC3T2SI	EC3T1SI	xxxx,0000B

EPWM3I：PWM3 中断使能控制位

- 0：关闭 PWM3 中断
- 1：使能 PWM3 中断，当 C3IF 被硬件置 1 时，程序将跳转到相应中断入口执行中断服务程序。

EC3T2SI：PWM3 的 T2 匹配发生波形翻转时的中断控制位

- 0：关闭 T2 翻转时中断
- 1：使能 T2 翻转时中断，当 PWM 波形发生器内部计数值与 T2 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C3IF 置 1，此时若 EPWM3I==1，则程序将跳转到相应中断入口执行中断服务程序。

EC3T1SI：PWM3 的 T1 匹配发生波形翻转时的中断控制位

- 0：关闭 T1 翻转时中断
- 1：使能 T1 翻转时中断，当 PWM 波形发生器内部计数值与 T1 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C3IF 置 1，此时若 EPWM3I==1，则程序将跳转到相应中断入口执行中断服务程序。

7、PWM4 的控制寄存器：PWM4CR

PWM4 的控制寄存器的格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM4CR	FF24H（XSFR）	name	-	-	-	-	PWM4_PS	EPWM4I	EC4T2SI	EC4T1SI	xxxx,0000B

EPWM4I：PWM4 中断使能控制位

- 0：关闭 PWM4 中断
- 1：使能 PWM4 中断，当 C4IF 被硬件置 1 时，程序将跳转到相应中断入口执行中断服务程序。

EC4T2SI：PWM4 的 T2 匹配发生波形翻转时的中断控制位

- 0：关闭 T2 翻转时中断
- 1：使能 T2 翻转时中断，当 PWM 波形发生器内部计数值与 T2 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C4IF 置 1，此时若 EPWM4I==1，则程序将跳转到相应中断入口执行中断服务程序。

EC4T1SI：PWM4 的 T1 匹配发生波形翻转时的中断控制位

- 0：关闭 T1 翻转时中断
- 1：使能 T1 翻转时中断，当 PWM 波形发生器内部计数值与 T1 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C4IF 置 1，此时若 EPWM4I==1，则程序将跳转到相应中断入口执行中断服务程序。

8、PWM5 的控制寄存器： PWM5CR

PWM5 的控制寄存器的格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM5CR	FF34H（XSFR）	name	-	-	-	-	PWM5_PS	EPWM5I	EC5T2SI	EC5T1SI	xxxx,0000B

EPWM5I：PWM5 中断使能控制位

- 0：关闭 PWM5 中断
- 1：使能 PWM5 中断，当 C5IF 被硬件置 1 时，程序将跳转到相应中断入口执行中断服务程序。

EC5T2SI：PWM5 的 T2 匹配发生波形翻转时的中断控制位

- 0：关闭 T2 翻转时中断
- 1：使能 T2 翻转时中断，当 PWM 波形发生器内部计数值与 T2 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C5IF 置 1，此时若 EPWM5I==1，则程序将跳转到相应中

断入口执行中断服务程序。

EC5T1SI：PWM5 的 T1 匹配发生波形翻转时的中断控制位

- 0：关闭 T1 翻转时中断
- 1：使能 T1 翻转时中断，当 PWM 波形发生器内部计数值与 T1 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C5IF 置 1，此时若 EPWM5I==1，则程序将跳转到相应中断入口执行中断服务程序。

9、PWM6 的控制寄存器：PWM6CR

PWM6 的控制寄存器的格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM6CR	FF44H（XSFR）	name	-	-	-	-	PWM6_PS	EPWM6I	EC6T2SI	EC6T1SI	xxxx,0000B

EPWM6I：PWM6 中断使能控制位

- 0：关闭 PWM6 中断
- 1：使能 PWM6 中断，当 C6IF 被硬件置 1 时，程序将跳转到相应中断入口执行中断服务程序。

EC6T2SI：PWM6 的 T2 匹配发生波形翻转时的中断控制位

- 0：关闭 T2 翻转时中断
- 1：使能 T2 翻转时中断，当 PWM 波形发生器内部计数值与 T2 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C6IF 置 1，此时若 EPWM6I==1，则程序将跳转到相应中断入口执行中断服务程序。

EC6T1SI：PWM6 的 T1 匹配发生波形翻转时的中断控制位

- 0：关闭 T1 翻转时中断
- 1：使能 T1 翻转时中断，当 PWM 波形发生器内部计数值与 T1 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C6IF 置 1，此时若 EPWM6I==1，则程序将跳转到相应中断入口执行中断服务程序。

10、PWM7 的控制寄存器：PWM7CR

PWM7 的控制寄存器的格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM7CR	FF54H（XSFR）	name	-	-	-	-	PWM7_PS	EPWM7I	EC7T2SI	EC7T1SI	xxxx,0000B

EPWM7I：PWM7 中断使能控制位

- 0：关闭 PWM7 中断
- 1：使能 PWM7 中断，当 C7IF 被硬件置 1 时，程序将跳转到相应中断入口执行中断服务程序。

EC7T2SI：PWM7 的 T2 匹配发生波形翻转时的中断控制位

- 0：关闭 T2 翻转时中断
- 1：使能 T2 翻转时中断，当 PWM 波形发生器内部计数值与 T2 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C7IF 置 1，此时若 EPWM7I==1，则程序将跳转到相应中断入口执行中断服务程序。

EC7T1SI：PWM7 的 T1 匹配发生波形翻转时的中断控制位

- 0：关闭 T1 翻转时中断

- 1：使能 T1 翻转时中断，当 PWM 波形发生器内部计数值与 T1 计数器所设定的值相匹配时，PWM 的波形发生翻转，同时硬件将 C7IF 置 1，此时若 EPWM7I==1，则程序将跳转到相应中断入口执行中断服务程序。

中断向量地址及中断控制

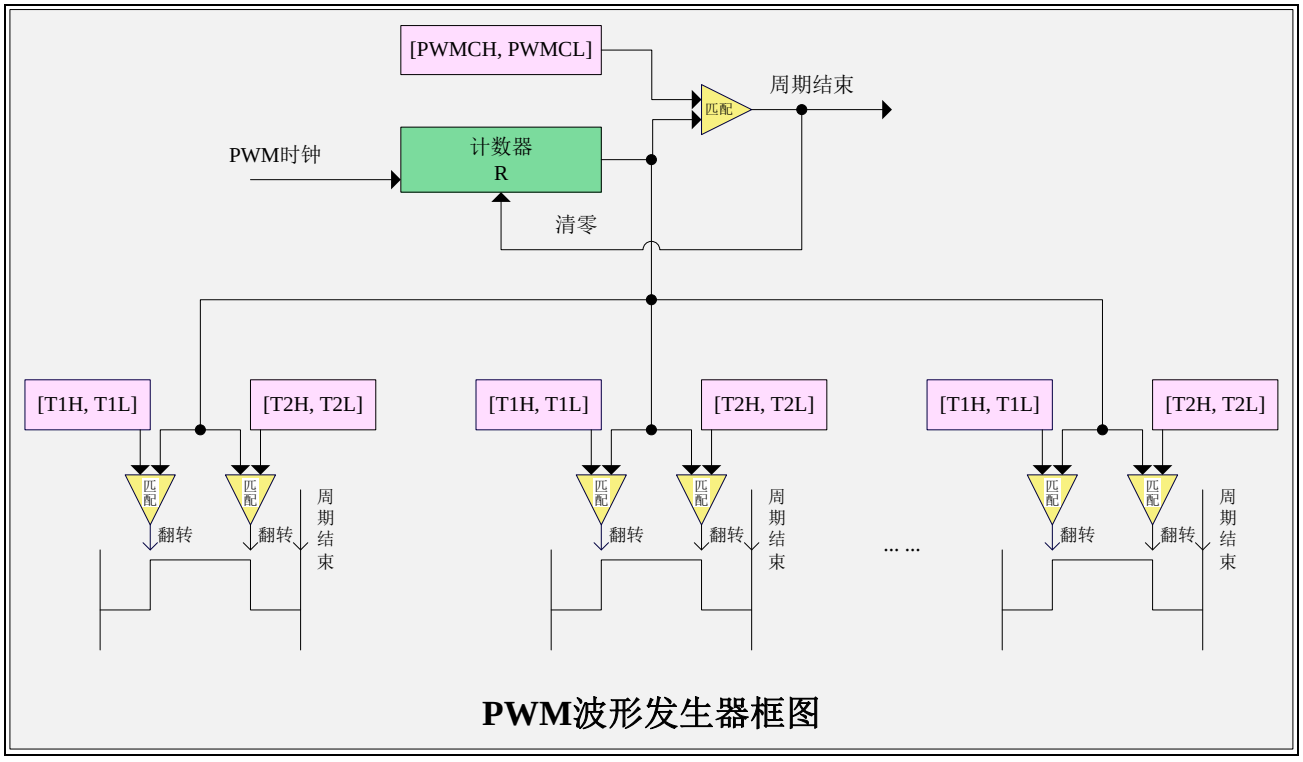
中断名称	入口地址	优先级设置	中断请求位	中断允许位	中断标志清除方式
PWM 中断	00B3H（22）	PPWM	CBIF	ENPWM/ECBI/EA	需软件清除
			C2IF	ENPWM / EPWM2I / EC2T2SI || EC2T1SI / EA	需软件清除
			C3IF	ENPWM / EPWM3I / EC3T2SI || EC3T1SI / EA	需软件清除
			C4IF	ENPWM / EPWM4I / EC4T2SI || EC4T1SI / EA	需软件清除
			C5IF	ENPWM / EPWM5I / EC5T2SI || EC5T1SI / EA	需软件清除
			C6IF	ENPWM / EPWM6I / EC6T2SI || EC6T1SI / EA	需软件清除
			C7IF	ENPWM / EPWM7I / EC7T2SI || EC7T1SI / EA	需软件清除
PWM 异常检测中断	00BBH（23）	PPWMFD	FDIF	ENPWM / ENFD / EFDI / EA	需软件清除

在 Kci1C 中声明中断函数

void PWM_Routine(void) interrupt 22;

void PWMFD_Routine(void) interrupt 23;

PTI 波形发生器的结构框图

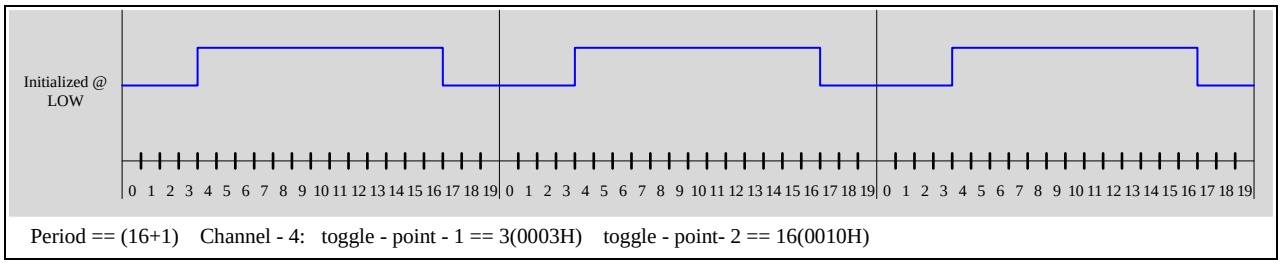


PWM波形发生器框图

汇编示例代码 1

假如要生成一个重复的 PWM 波形，波形如下：

PWM 波形发生器的时钟频率为系统时钟/4，波形由通道 4 输出，周期为 20 个 PWM 时钟，占空比为 1/3，由 4 个 PWM 时钟的相位延迟（波形如下图所示）



汇编代码可以如下设计：

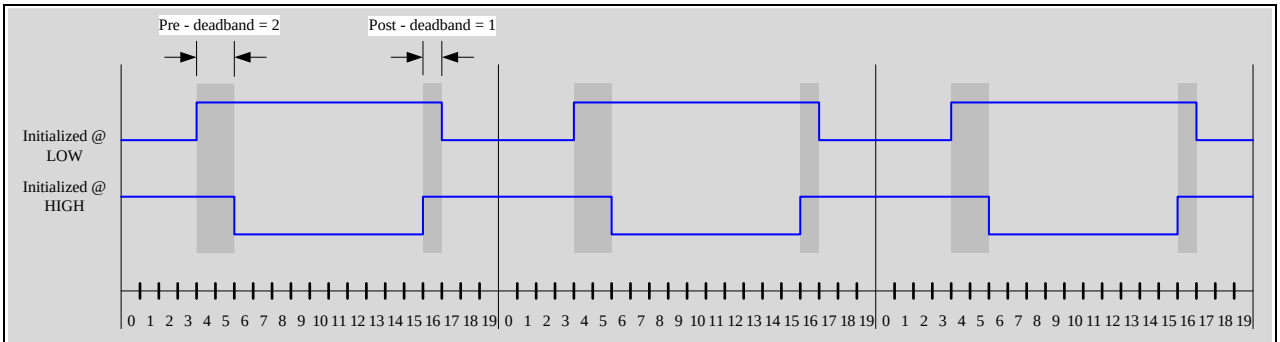
```
;; |   Global Configuration   |
; Set EAXSFR to enable xSFR writing against XRAM writing
    MOV     A, P_SW2
    ORL     A, #10000000B
    MOV     P_SW2, A
;
; Set channel-4 output register start at LOW
    MOV     A, PWMCFG
    ANL     A, #11111011B           ; channel-4 start at LOW
    MOV     PWMCFG, A
;
; Set a clock of the waveform generator consists of 4 Fosc
    MOV     DPTR, #PWMCKS           ; FFF2H
    MOV     A, #00000011B
    MOVX    @DPTR, A
;
; Set period as 20
; {PWMCH, PWMCL} <= 19
    MOV     DPTR, #PWMCH            ; FFF0H
    MOV     A, #00H                 ; PWMCH should be changed first
    MOVX    @DPTR, A
    MOV     DPTR, #PWMCL            ; FFF1H
    MOV     A, #13H                 ; Write PWMCL simultaneous update PWMCH
    MOVX    @DPTR, A
;
;; | Channel-4 Configuration   |
; Set toggle point 1 of Channel-4 as 3
    MOV     DPTR, #PWM4T1H          ; FF20H
    MOV     A, #00H
    MOVX    @DPTR, A
;
    MOV     DPTR,   #PWM4T1L        ; FF21H
    MOV     A,   #03H
    MOVX    @DPTR, A
;
; Set toggle point 2 of Channel-4 as 16
    MOV     DPTR, #PWM4T2H          ; FF22H
```

```
    MOV     A, #00H
    MOVX    @DPTR, A
;
    MOV     DPTR, #PWM4T2L          ; FF23H
    MOV     A, #10H
    MOVX    @DPTR, A
;
; Set Channel-4 output pin as default, and disable interrupting
    MOV     DPTR, #PWM4CR           ; FF24H
    MOV     A, #00H
    MOVX    @DPTR, A
;
; Clear EAXSFR to disable xSFR, return MOVX-DPTR to normal XRAM access
    MOV     A, P_SW2
    ANL     A, #01111111B
    MOV     P_SW2, A
;
; | Operate PWM output |
; Enable counter counting, and enable Channel-4 output
    MOV     A, PWMCR
    ORL     A, #10000100B
    MOV     PWMCR, A
;
```

汇编示例代码 2

假如要生成两个互补对称输出的 PWM 波形，波形如下：

PWM 波形发生器的时钟频率为系统时钟/4，波形由信道 4 和信道 5 输出，周期为 20 个 PWM 时钟，通道 4 的有效高电平为 13 个 PWM 时钟，通道 5 的有效低电平为 10 个 PWM 时钟，信道 4 和信道 5 前端死区为 2 个 PWM 时钟，末端死区为 1 个 PWM 时钟（波形如下图所示）



Period == (16+1) Channel - 4: toggle - point -1 == 3(0003H) toggle - point - 2 == 16(0010H)
Channel - 5: toggle - point -1 == 3(0005H) toggle - point - 2 == 16(000FH)

汇编代码可以如下设计：

```
; |   Global Configuration   |
;
; Set EAXSFR to enable xSFR writing against XRAM writing
;
    MOV     A, P_SW2
```

```
    ORL     A, #10000000B
    MOV     P_SW2, A
;
; Set channel-4 output register start at LOW, channel-5 at HIGH
    MOV     A, PWMCFG
    ANL     A, #11111011B           ; channel-4 start at LOW
    ORL     A, #00001000B           ; channel-5 start at HIGH
    MOV     PWMCFG, A
;
; Set a clock of the waveform generator consists of 4 Fosc
    MOV     DPTR, #PWMCKS           ; FFF2H
    MOV     A, #00000011B
    MOVX    @DPTR, A
;
; Set period as 20
; {PWMCH,PWMCL} <= 19
    MOV     DPTR, #PWMCH            ; FFF0H
    MOV     A, #00H                 ; PWMCH should be changed first
    MOVX    @DPTR, A
    MOV     DPTR, #PWMCL            ; FFF1H
    MOV     A, #13H                 ; Write PWMCL simultaneous update PWMCH
    MOVX    @DPTR, A
;
;; |   Channel-4 Configuration   |
; Set toggle point 1 of Channel-4 as 3
    MOV     DPTR, #PWM4T1H          ; FF20H
    MOV     A, #00H
    MOVX    @DPTR, A
;
    MOV     DPTR, #PWM4T1L          ; FF21H
    MOV     A, #03H
    MOVX    @DPTR, A
;
; Set toggle point 2 of Channel-4 as 16
    MOV     DPTR, #PWM4T2H          ; FF22H
    MOV     A, #00H
    MOVX    @DPTR, A
;
    MOV     DPTR, #PWM4T2L          ; FF23H
    MOV     A, #10H
    MOVX    @DPTR, A
;
; Set Channel-4 output pin as default, and disable interrupting
    MOV     DPTR, #PWM4CR           ; FF24H
    MOV     A, #00H
    MOVX    @DPTR, A
```

```
;
;; | Channel-5 Configuration |
; Set toggle point 1 of Channel-5 as 5
    MOV     DPTR, #PWM5T1H          ; FF30H
    MOV     A, #00H
    MOVX    @DPTR, A
;
    MOV     DPTR, #PWM5T1L          ; FF31H
    MOV     A, #03H
    MOVX    @DPTR, A
;
; Set toggle point 3 of Channel-5 as 15
    MOV     DPTR, #PWM5T2H          ; FF32H
    MOV     A, #00H
    MOVX    @DPTR, A
;
    MOV     DPTR, #PWM5T2L          ; FF33H
    MOV     A, #0FH
    MOVX    @DPTR, A
;
; Set Channel-5 output pin as default, and disable interrupting
    MOV     DPTR, #PWM5CR           ; FF34H
    MOV     A, #00H
    MOVX    @DPTR, A
;
;;; Clear EAXSFR to disable xSFR, return MOVX-DPTR to normal XRAM access
    MOV     A, P_SW2
    ANL     A, #01111111B
    MOV     P_SW2, A
;
;; |   Operate PWM output   |
; Enable counter counting, and enable Channel-4 and Channel-5 output
    MOV     A, PWMCR
    ORL     A, #10001100B
    MOV     PWMCR, A
;
```

23.3 利用 PWM 波形发生器控制舞台灯光的示例程序(C 和汇编)

1、使用 PWM 波形发生器控制舞台灯光-C 语言程序

```
/*----STC15xx 系列使用增强型 PWM 控制舞台灯光示例----*/
//本示例在 Keil 开发环境下请选择 Intel 的 8058 芯片型号进行编译
//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"
#define   CYCLE     0x1000L          //定义 PWM 周期(最大值为 32767)
#define   PWMC      (*(unsigned int  volatile xdata *)0xfff0)
#define   PWMCH     (*(unsigned char volatile xdata *)0xfff0)
#define   PWMCL     (*(unsigned char volatile xdata *)0xfff1)
#define   PWMCKS    (*(unsigned char volatile xdata *)0xfff2)
#define   PWM2T1    (*(unsigned int  volatile xdata *)0xff00)
#define   PWM2T1H   (*(unsigned char volatile xdata *)0xff00)
#define   PWM2T1L   (*(unsigned char volatile xdata *)0xff01)
#define   PWM2T2    (*(unsigned int  volatile xdata *)0xff02)
#define   PWM2T2H   (*(unsigned char volatile xdata *)0xff02)
#define   PWM2T2L   (*(unsigned char volatile xdata *)0xff03)
#define   PWM2CR    (*(unsigned char volatile xdata *)0xff04)
#define   PWM3T1    (*(unsigned int  volatile xdata *)0xff10)
#define   PWM3T1H   (*(unsigned char volatile xdata *)0xff10)
#define   PWM3T1L   (*(unsigned char volatile xdata *)0xff11)
#define   PWM3T2    (*(unsigned int  volatile xdata *)0xff12)
#define   PWM3T2H   (*(unsigned char volatile xdata *)0xff12)
#define   PWM3T2L   (*(unsigned char volatile xdata *)0xff13)
#define   PWM3CR    (*(unsigned char volatile xdata *)0xff14)
#define   PWM4T1    (*(unsigned int  volatile xdata *)0xff20)
#define   PWM4T1H   (*(unsigned char volatile xdata *)0xff20)
#define   PWM4T1L   (*(unsigned char volatile xdata *)0xff21)
#define   PWM4T2    (*(unsigned int  volatile xdata *)0xff22)
#define   PWM4T2H   (*(unsigned char volatile xdata *)0xff22)
#define   PWM4T2L   (*(unsigned char volatile xdata *)0xff23)
#define   PWM4CR    (*(unsigned char volatile xdata *)0xff24)
#define   PWM5T1    (*(unsigned int  volatile xdata *)0xff30)
#define   PWM5T1H   (*(unsigned char volatile xdata *)0xff30)
#define   PWM5T1L   (*(unsigned char volatile xdata *)0xff31)
#define   PWM5T2    (*(unsigned int  volatile xdata *)0xff32)
#define   PWM5T2H   (*(unsigned char volatile xdata *)0xff32)
#define   PWM5T2L   (*(unsigned char volatile xdata *)0xff33)
#define   PWM5CR    (*(unsigned char volatile xdata *)0xff34)
#define   PWM6T1    (*(unsigned int  volatile xdata *)0xff40)
#define   PWM6T1H   (*(unsigned char volatile xdata *)0xff40)
#define   PWM6T1L   (*(unsigned char volatile xdata *)0xff41)
#define   PWM6T2    (*(unsigned int  volatile xdata *)0xff42)
```

```
#define   PWM6T2H   (*(unsigned char   volatile xdata *)0xff42)
#define   PWM6T2L   (*(unsigned char   volatile xdata *)0xff43)
#define   PWM6CR    (*(unsigned char   volatile xdata *)0xff44)
#define   PWM7T1    (*(unsigned int    volatile xdata *)0xff50)
#define   PWM7T1H   (*(unsigned char   volatile xdata *)0xff50)
#define   PWM7T1L   (*(unsigned char   volatile xdata *)0xff51)
#define   PWM7T2    (*(unsigned int    volatile xdata *)0xff52)
#define   PWM7T2H   (*(unsigned char   volatile xdata *)0xff52)
#define   PWM7T2L   (*(unsigned char   volatile xdata *)0xff53)
#define   PWM7CR    (*(unsigned char   volatile xdata *)0xff54)

sfr       P_SW2 = 0xba;

sfr       P0M1 = 0x93;
sfr       P0M0 = 0x94;
sfr       P1M1 = 0x91;
sfr       P1M0 = 0x92;
sfr       P2M1 = 0x95;
sfr       P2M0 = 0x96;
sfr       P3M1 = 0xb1;
sfr       P3M0 = 0xb2;
sfr       P4M1 = 0xb3;
sfr       P4M0 = 0xb4;
sfr       P5M1 = 0xC9;
sfr       P5M0 = 0xCA;
sfr       P6M1 = 0xCB;
sfr       P6M0 = 0xCC;
sfr       P7M1 = 0xE1;
sfr       P7M0 = 0xE2;
sfr       PWMCFG = 0xf1;
sfr       PWMCR = 0xf5;
sfr       PWMIF = 0xf6;
sfr       PWMFDCR = 0xf7;

sbit      PWM2 = P3^7;
sbit      PWM3 = P2^1;
sbit      PWM4 = P2^2;
sbit      PWM5 = P2^3;
sbit      PWM6 = P0^7;
sbit      PWM7 = P0^6;

void PWM_config(void);

void PWM2_SetPwmWide(unsigned short Wide);
void PWM3_SetPwmWide(unsigned short Wide);
void PWM4_SetPwmWide(unsigned short Wide);
```

```
void PWM5_SetPwmWide(unsigned short Wide);
void PWM6_SetPwmWide(unsigned short Wide);
void PWM7_SetPwmWide(unsigned short Wide);

void main()
{
    PWM_config();
    PWM2_SetPwmWide(0);             //输出全低电平
    PWM3_SetPwmWide(1);             //输出 1/2550 高电平
    PWM4_SetPwmWide(CYCLE);         //输出全高电平
    PWM5_SetPwmWide(CYCLE-1);       //输出 2549/2550 低电平
    PWM6_SetPwmWide(CYCLE/2);       //输出 1/2 高电平
    PWM7_SetPwmWide(CYCLE/3);       //输出 1/3 高电平
    while (1);
}

void PWM_config(void)
{
    P0M0 &= ~0xc0;
    P0M1 &= ~0xc0;
    P0 &= ~0xc0;                    //设置 P0.6/.P0.7 电平
    P2M0 &= ~0x0e;
    P2M1 &= ~0x0e;
    P2 &= ~0x0e;                    //设置 P2.1/P2.2/P2.3 电平
    P3M0 &= ~0x80;
    P3M1 &= ~0x80;
    P3 &= ~0x80;                    //设置 P3.7 电平

    P_SW2 |= 0x80;

    PWMCKS = 0x00;
    PWMC = CYCLE;                   //设置 PWM 周期
    PWM2T1 = 1;
    PWM2T2 = 0;
    PWM2CR = 0x00;                  //PWM2 输出到 P3.7
    PWM3T1 = 1;
    PWM3T2 = 0;
    PWM3CR = 0x00;                  //PWM3 输出到 P2.1
    PWM4T1 = 1;
    PWM4T2 = 0;
    PWM4CR = 0x00;                  //PWM4 输出到 P2.2
    PWM5T1 = 1;
    PWM5T2 = 0;
    PWM5CR = 0x00;                  //PWM5 输出到 P2.3
    PWM6T1 = 1;
    PWM6T2 = 0;
```

```
    PWM6CR = 0x08;              //PWM6 输出到 P0.7
    PWM7T1 = 1;
    PWM7T2 = 0;
    PWM7CR = 0x08;              //PWM7 输出到 P0.6
    PWMCFG = 0x00;              //配置 PWM 的输出初始电平
    PWMCR = 0x3f;               //使能 PWM 信号输出
    PWMCR |= 0x80;              //使能 PWM 模块

    P_SW2 &= ~0x80;
}

void PWM2_SetPwmWide(unsigned short Wide)
{
    if (Wide == 0)
    {
        PWMCR &= ~0x01;
        PWM2 = 0;
    }
    else if (Wide == CYCLE)
        {
            PWMCR &= ~0x01;
            PWM2 = 1;
        }
        else
        {
            P_SW2 |= 0x80;
            PWM2T1 = Wide;
            P_SW2 &= ~0x80;
            PWMCR |= 0x01;
        }
}

void PWM3_SetPwmWide(unsigned short Wide)
{
    if (Wide == 0)
    {
        PWMCR &= ~0x02;
        PWM3 = 0;
    }
    else if (Wide == CYCLE)
        {
            PWMCR &= ~0x02;
            PWM3 = 1;
        }
        else
        {
```

```
            P_SW2 |= 0x80;
            PWM3T1 = Wide;
            P_SW2 &= ~0x80;
            PWMCR |= 0x02;
        }
}

void PWM4_SetPwmWide(unsigned short Wide)
{
    if (Wide == 0)
    {
        PWMCR &= ~0x04;
        PWM4 = 0;
    }
    else if (Wide == CYCLE)
        {
            PWMCR &= ~0x04;
            PWM4 = 1;
        }
        else
        {
            P_SW2 |= 0x80;
            PWM4T1 = Wide;
            P_SW2 &= ~0x80;
            PWMCR |= 0x04;
        }
}

void PWM5_SetPwmWide(unsigned short Wide)
{
    if (Wide == 0)
    {
        PWMCR &= ~0x08;
        PWM5 = 0;
    }
    else if (Wide == CYCLE)
        {
            PWMCR &= ~0x08;
            PWM5 = 1;
        }
        else
        {
            P_SW2 |= 0x80;
            PWM5T1 = Wide;
            P_SW2 &= ~0x80;
            PWMCR |= 0x08;
```

```
        }
}

void PWM6_SetPwmWide(unsigned short Wide)
{
    if (Wide == 0)
    {
        PWMCR &= ~0x10;
        PWM6 = 0;
    }
    else if (Wide == CYCLE)
        {
            PWMCR &= ~0x10;
            PWM6 = 1;
        }
        else
        {
            P_SW2 |= 0x80;
            PWM6T1 = Wide;
            P_SW2 &= ~0x80;
            PWMCR |= 0x10;
        }
}

void PWM7_SetPwmWide(unsigned short Wide)
{
    if (Wide == 0)
    {
        PWMCR &= ~0x20;
        PWM7 = 0;
    }
    else if (Wide == CYCLE)
        {
            PWMCR &= ~0x20;
            PWM7 = 1;
        }
        else
        {
            P_SW2 |= 0x80;
            PWM7T1 = Wide;
            P_SW2 &= ~0x80;
            PWMCR |= 0x20;
        }
}
```

2 使用 PWM 波形发生器控制舞台灯光-汇编程序

```
/* --- STC15Fxx 系列 使用增强型 PWM 控制舞台灯光示例----------------*/
//本示例在 Keil 开发环境下请选择 Intel 的 8058 芯片型号进行编译
//假定测试芯片的工作频率为 18.432MHz
CYCLE       EQU     1000H       ;定义 PWM 周期(最大值为 32767)

PWMC        EQU     0FFF0H
PWMCH       EQU     0FFF0H
PWMCL       EQU     0FFF1H
PWMCKS      EQU     0FFF2H
PWM2T1      EQU     0FF00H
PWM2T1H     EQU     0FF00H
PWM2T1L     EQU     0FF01H
PWM2T2      EQU     0FF02H
PWM2T2H     EQU     0FF02H
PWM2T2L     EQU     0FF03H
PWM2CR      EQU     0FF04H
PWM3T1      EQU     0FF10H
PWM3T1H     EQU     0FF10H
PWM3T1L     EQU     0FF11H
PWM3T2      EQU     0FF12H
PWM3T2H     EQU     0FF12H
PWM3T2L     EQU     0FF13H
PWM3CR      EQU     0FF14H
PWM4T1      EQU     0FF20H
PWM4T1H     EQU     0FF20H
PWM4T1L     EQU     0FF21H
PWM4T2      EQU     0FF22H
PWM4T2H     EQU     0FF22H
PWM4T2L     EQU     0FF23H
PWM4CR      EQU     0FF24H
PWM5T1      EQU     0FF30H
PWM5T1H     EQU     0FF30H
PWM5T1L     EQU     0FF31H
PWM5T2      EQU     0FF32H
PWM5T2H     EQU     0FF32H
PWM5T2L     EQU     0FF33H
PWM5CR      EQU     0FF34H
PWM6T1      EQU     0FF40H
PWM6T1H     EQU     0FF40H
PWM6T1L     EQU     0FF41H
PWM6T2      EQU     0FF42H
PWM6T2H     EQU     0FF42H
PWM6T2L     EQU     0FF43H
PWM6CR      EQU     0FF44H
```

```
PWM7T1      EQU     0FF50H
PWM7T1H     EQU     0FF50H
PWM7T1L     EQU     0FF51H
PWM7T2      EQU     0FF52H
PWM7T2H     EQU     0FF52H
PWM7T2L     EQU     0FF53H
PWM7CR      EQU     0FF54H

P_SW2       DATA    0BAH

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0b1H
P3M0        DATA    0b2H
P4M1        DATA    0b3H
P4M0        DATA    0b4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH
P6M1        DATA    0CBH
P6M0        DATA    0CCH
P7M1        DATA    0E1H
P7M0        DATA    0E2H

PWMCFG      DATA    0F1H
PWMCR       DATA    0F5H
PWMIF       DATA    0F6H
PWMFDCR     DATA    0F7H

PWM2        BIT     P3.7
PWM3        BIT     P2.1
PWM4        BIT     P2.2
PWM5        BIT     P2.3
PWM6        BIT     P0.7
PWM7        BIT     P0.6

MOVXB       MACRO ADR, DAT
    MOV     DPTR, #ADR
    MOV     A, #DAT
    MOVX    @DPTR, A
    ENDM

MOVXW       MACRO ADR, DAT
```

```
    MOV     DPTR, #ADR
    MOV     A, #HIGH DAT
    MOVX    @DPTR, A
    INC     DPTR
    MOV     A, #LOW DAT
    MOVX    @DPTR, A
    ENDM

MOVXWD  MACRO ADR
    MOV     DPTR, #ADR
    MOV     A, DPH
    MOVX    @DPTR, A
    INC     DPTR
    MOV     A, DPL
    MOVX    @DPTR, A
    ENDM

    ORG     0000H
    LJMP    START

    ORG     0100H

START:
    CALL    PWM_config

    MOV     DPTR, #0
    CALL    PWM2_SetPwmWide     //输出全低电平
    MOV     DPTR, #1
    CALL    PWM3_SetPwmWide     //输出 1/2550 高电平
    MOV     DPTR, #CYCLE
    CALL    PWM4_SetPwmWide     //输出全高电平
    MOV     DPTR, #CYCLE-1
    CALL    PWM5_SetPwmWide     //输出 2549/2550 低电平
    MOV     DPTR, #CYCLE/2
    CALL    PWM6_SetPwmWide     //输出 1/2 高电平
    MOV     DPTR, #CYCLE/3
    CALL    PWM7_SetPwmWide     //输出 1/3 高电平
    JMP     $

PWM_config:
    MOV     A, #NOT 0xc0
    ANL     P0M0, A             //设置 P0.6/.P0.7 电平
    ANL     P0M1, A
    ANL     P0, A

    MOV     A, #NOT 0x0e
```

```
    ANL     P2M0, A             //设置 P2.1/P2.2/P2.3 电平
    ANL     P2M1, A
    ANL     P2, A

    MOV     A, #NOT 0x80
    ANL     P3M0, A             //设置 P3.7 电平
    ANL     P3M1, A
    ANL     P3, A
    ORL     P_SW2, #0x80

    MOVXB   PWMCKS, 0x00
    MOVXW   PWMC, CYCLE         //设置 PWM 周期
    MOVXW   PWM2T1,1
    MOVXW   PWM2T2,0
    MOVXB   PWM2CR,0x00         //PWM2 输出到 P3.7

    MOVXW   PWM3T1,1
    MOVXW   PWM3T2,0
    MOVXB   PWM3CR,0x00         //PWM3 输出到 P2.1

    MOVXW   PWM4T1,1
    MOVXW   PWM4T2,0
    MOVXB   PWM4CR,0x00         //PWM4 输出到 P2.2

    MOVXW   PWM5T1,1
    MOVXW   PWM5T2,0
    MOVXB   PWM5CR,0x00         //PWM5 输出到 P2.3

    MOVXW   PWM6T1,1
    MOVXW   PWM6T2,0
    MOVXB   PWM6CR,0x08         //PWM6 输出到 P0.7

    MOVXW   PWM7T1,1
    MOVXW   PWM7T2,0
    MOVXB   PWM7CR,0x08         //PWM7 输出到 P0.6

    MOV     PWMCFG,#0x00        //配置 PWM 的输出初始电平
    MOV     PWMCR,#0x3f         //使能 PWM 信号输出
    ORL     PWMCR,#0x80         //使能 PWM 模块

    ANL     P_SW2, #NOT 0x80
    RET

PWM2_SetPwmWide:                //DPTR 存放脉冲宽度
    MOV     A, DPH
    ORL     A, DPL
```

```
    JNZ     PWM2NOT0
    ANL     PWMCR, #NOT 0x01
    CLR     PWM2
    RET

PWM2NOT0:
    MOV     A, DPH
    CJNE    A, #HIGH CYCLE, $+3
    JC      PWM2NOT1
    MOV     A, DPL
    CJNE    A, #LOW CYCLE, $+3
    JC      PWM2NOT1
    ANL     PWMCR, #NOT 0x01
    SETB    PWM2
    RET

PWM2NOT1:
    ORL     P_SW2, #0x80
    ORL     PWMCR, #0x01
    MOVXWD  PWM2T1
    ANL     P_SW2, #NOT 0x80
    RET

PWM3_SetPwmWide:                    //DPTR 存放脉冲宽度
    MOV     A, DPH
    ORL     A, DPL
    JNZ     PWM3NOT0
    ANL     PWMCR, #NOT 0x02
    CLR     PWM3
    RET

PWM3NOT0:
    MOV     A, DPH
    CJNE    A, #HIGH CYCLE, $+3
    JC      PWM3NOT1
    MOV     A, DPL
    CJNE    A, #LOW CYCLE, $+3
    JC      PWM3NOT1
    ANL     PWMCR, #NOT 0x02
    SETB    PWM3
    RET

PWM3NOT1:
    ORL     P_SW2, #0x80
    ORL     PWMCR, #0x02
    MOVXWD  PWM3T1
```

```
    ANL     P_SW2, #NOT 0x80
    RET

PWM4_SetPwmWide:                    //DPTR 存放脉冲宽度
    MOV     A, DPH
    ORL     A, DPL
    JNZ     PWM4NOT0
    ANL     PWMCR, #NOT 0x04
    CLR     PWM4
    RET

PWM4NOT0:
    MOV     A, DPH
    CJNE    A, #HIGH CYCLE, $+3
    JC      PWM4NOT1
    MOV     A, DPL
    CJNE    A, #LOW CYCLE, $+3
    JC      PWM4NOT1
    ANL     PWMCR, #NOT 0x04
    SETB    PWM4
    RET

PWM4NOT1:
    ORL     P_SW2, #0x80
    ORL     PWMCR, #0x04
    MOVXWD  PWM4T1
    ANL     P_SW2, #NOT 0x80
    RET

PWM5_SetPwmWide:                    //DPTR 存放脉冲宽度
    MOV     A, DPH
    ORL     A, DPL
    JNZ     PWM5NOT0
    ANL     PWMCR, #NOT 0x08
    CLR     PWM5
    RET

PWM5NOT0:
    MOV     A, DPH
    CJNE    A, #HIGH CYCLE, $+3
    JC      PWM5NOT1
    MOV     A, DPL
    CJNE    A, #LOW CYCLE, $+3
    JC      PWM5NOT1
    ANL     PWMCR, #NOT 0x08
    SETB    PWM5
```

```
    RET

PWM5NOT1:
    ORL     P_SW2, #0x80
    ORL     PWMCR, #0x08
    MOVXWD  PWM5T1
    ANL     P_SW2, #NOT 0x80
    RET

PWM6_SetPwmWide:                    //DPTR 存放脉冲宽度
    MOV     A, DPH
    ORL     A, DPL
    JNZ     PWM6NOT0
    ANL     PWMCR, #NOT 0x10
    CLR     PWM6
    RET

PWM6NOT0:
    MOV     A, DPH
    CJNE    A, #HIGH CYCLE, $+3
    JC      PWM6NOT1
    MOV     A, DPL
    CJNE    A, #LOW CYCLE, $+3
    JC      PWM6NOT1
    ANL     PWMCR, #NOT 0x10
    SETB    PWM6
    RET

PWM6NOT1:
    ORL     P_SW2, #0x80
    ORL     PWMCR, #0x10
    MOVXWD  PWM6T1
    ANL     P_SW2, #NOT 0x80
    RET

PWM7_SetPwmWide:                    //DPTR 存放脉冲宽度
    MOV     A, DPH
    ORL     A, DPL
    JNZ     PWM7NOT0
    ANL     PWMCR, #NOT 0x20
    CLR     PWM7
    RET

PWM7NOT0:
    MOV     A, DPH
    CJNE    A, #HIGH CYCLE, $+3
```

```
    JC      PWM7NOT1
    MOV     A, DPL
    CJNE    A, #LOW CYCLE, $+3
    JC      PWM7NOT1
    ANL     PWMCR, #NOT 0x20
    SETB    PWM7
    RET

PWM7NOT1:
    ORL     P_SW2, #0x80
    ORL     PWMCR, #0x20
    MOVXWD  PWM7T1
    ANL     P_SW2, #NOT 0x80
    RET
END
```

23.4 两通道 CCP/PCA/增强型 PWM

两通道 CCP/PCA/增强 PWM 相关寄存器

符号	描述	地址	位地址及其符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CCON	PCA Control Register	D8H	CF	CR	-	-	-	-	CCF1	CCF0	00xx,xx00
CMOD	PCA Mode Register	D9H	CIDL	-	-	-	CPS2	CPS1	CPS0	ECF	0xxx,0000
CCAPM0	PCA Module 0 Mode Register	DAH	-	ECOM0	CAPP0	CAPN0	MAT0	TOG0	PWM0	ECCF0	x000,0000
CCAPM1	PCA Module 1 Mode Register	DBH	-	ECOM1	CAPP1	CAPN1	MAT1	TOG1	PWM1	ECCF1	x000,0000
CL	PCA Base Timer Low	E9H									0000,0000
CH	PCA Base Timer High	F9H									0000,0000
CCAP0L	PCA Module-0 Capture Register Low	EAH									0000,0000
CCAP0H	PCA Module-0 Capture Register High	FAH									0000,0000
CCAP1L	PCA Module-1 Capture Register Low	EBH									0000,0000
CCAP1H	PCA Module-1 Capture Register High	FBH									0000,0000
PCA_PWM0	PCA PWM Mode Auxiliary Register 0	F2H	EBS0_1	EBS0_0	PWM0_B9H	PWM0_B8H	PWM0_B9L	PWM0_B8L	EPC0H	EPC0L	0000,0000
PCA_PWM1	PCA PWM Mode Auxiliary Register 1	F3H	EBS1_1	EBS1_0	PWM1_B9H	PWM1_B8H	PWM1_B9L	PWM1_B8L	EPC1H	EPC1L	0000,0000

STC15W4K32S4 的两路 CCP 与 STC12F2K60S2 的 CCP 完全兼容，并在 STC12F2K60S2 的 CCP 的基础上对 PWM 的功能进行增强，不仅可将 STC15W4K32S4 的 CCP 设置为 6/7/8 位 PWM，还可设置为 10 位 PWM。10 位 PWM 的低字节仍用 CCAP0L/CCAP1L 设置（CCAP0H/CCAP1H 为重装值），10 位 PWM 的高两位使用 [PWM0_B9L, PWM0_B8L] / [PWM1_B9L, PWM1_B8L] 进行设置（[PWM0_B9H, PWM0_B8H] / [PWM1_B9H, PWM1_B8H]为重装值）。

[EBS0_1, EBS0_0]：

00：PWM0 为 8 位 PWM 模式
01：PWM0 为 7 位 PWM 模式
10：PWM0 为 6 位 PWM 模式
11：PWM0 为 10 位 PWM 模式

[EBS1_1, EBS1_0]：

00：PWM1 为 8 位 PWM 模式
01：PWM1 为 7 位 PWM 模式
10：PWM1 为 6 位 PWM 模式
11：PWM1 为 10 位 PWM 模式

10 位 PWM 的比较值由{PWMn_B9L, PWMn_B8L, CCAPnL[7:0]}组成，10 重装值由{PWMn_B9H, PWMn_B8H, CCAPnH[7:0]}组成

【注意】：在更新重装值时，必须先写高两位 PWMn_B9H, PWMn_B8H，后写低八位 CCAPnH

23.5 用 STC15W4KxxS4 系列单片机输出两路互补 SPWM

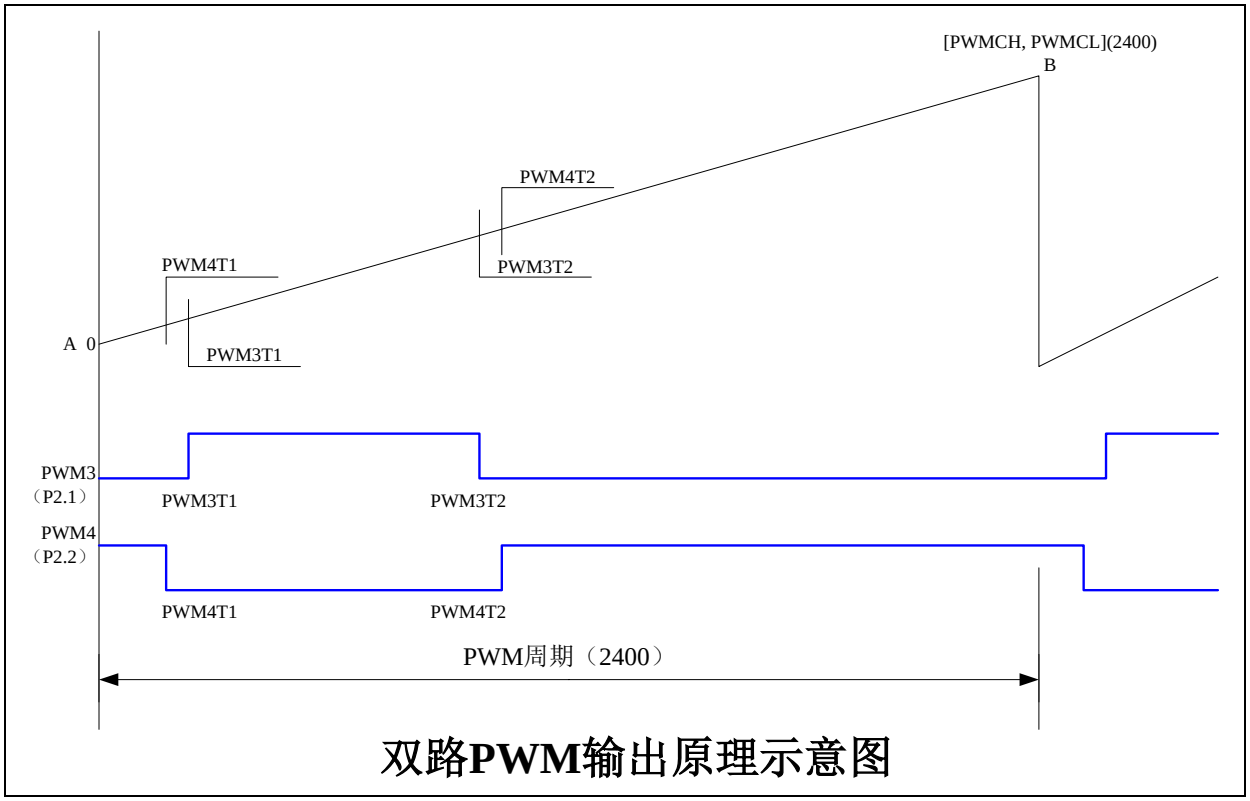
SPWM 是使用 PWM 来获得正弦波输出效果的一种技术，在交流驱动或变频领域应用广泛。

SPWM 知识是一个专门的学科，不了解的用户可以自己上网搜索相关的知识，本文档不做说明(要说明得比较大篇幅，各种图文说明等等)，默认用户已掌握。

本公司 STC15W4KxxS4 系列 MCU 内带 6 通道 15 位 PWM，各路 PWM 周期(频率)相同，输出的占空比独立可调，并且输出始终保持同步，输出相位可设置。这些特性使得设计 SPWM 成为可能，并且可方便设置死区时间，对于驱动桥式电路，死区时间至关重要。不过本 MCU 没有专门的死区控制寄存器，通过设置 PWM 占空比参数来达到。

本程序只演示两路互补 SPWM 的例子(单相)，如需要三相 SPWM，则相同方法设置另外 4 路 PWM，相位差为 120 度即可。

SPWM 产生原理如图 1：



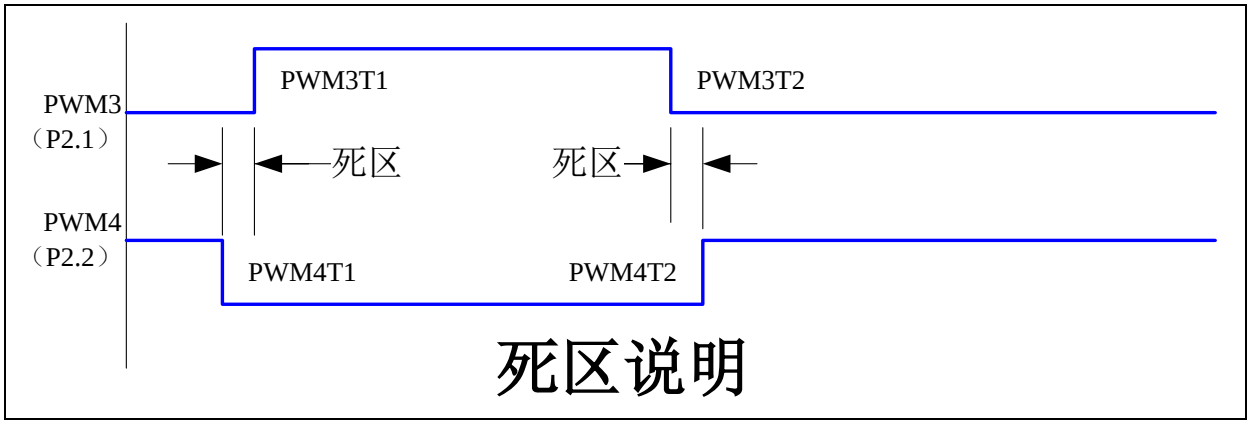
双路PWM输出原理示意图

内部 15 位的 PWM 计数器一旦运行，就会从 0 开始在每个 PWM 时钟到来时加 1，其值线性上升，当计数到与 15 位的周期设置寄存器[PWMCH，PWMCL]相等时（图中斜线 A 到 B），内部 PWM 计数器归 0，并产生中断，称为“归 0 中断”。本例周期设置为 2400，内部计数器计到 2400 就归 0，即 2399，下一个时钟就归 0。

6 路 PWM（PWM2 ~ PWM7）每路的结构一样，都包含两个 15 位的对输出 I/O 翻转的时刻设置寄存器 PWMnT1 和 PWMnT2，本例使用 PWM3 和 PWM4，对应 PWM3T1、PWM3T2 和 PWM4T1、PWM4T2。当内部计数器的值与某个翻转寄存器的值相等时，就对对应的输出 I/O 取反，本例中，PWM3 从 P2.1 输出，PWM4 从 P2.2 输出。假设 PWM3T1=65，PWM3T2=800，PWM4T1=53，PWM4T2=812，并且 PWM3 输出的 P2.1 初始电平为 0，PWM4 输出的 P2.2 初始电平为 1，则，当内部 PWM 计数器计到等于 PWM4T1=53 时，P2.2 由高输出低，计到等于 PWM3T1=65 时，P2.1 由低输出高，计到等于 PWM3T2=800

时， P2.1 由高输出低，计到等于 PWM4T2=812 时，P2.2 由低输出高。

从图中看到，两路输出是互补的，用于驱动一些 MOSFET 的半桥式驱动 IC。细心的用户可以看到，这两路 PWM 的翻转时刻有一点差别，相差 12 个时钟，为什么要这样设计呢？这就是传说中的死区。为了方便说明，把这两路 PWM 放大如图 2：



P2.2 输出低电平后，再过 12 个时钟（在 24MHZ 时，对应 0.5us），P2.1 输出高电平。
P2.1 输出低电平后，再过 12 个时钟（在 24MHZ 时，对应 0.5us），P2.2 输出高电平。

这个 12 个时钟就是死区时间，本例 PWM 时钟为 1T 模式，对应 0.5us。假设 P2.1 驱动的是半桥的下臂，P2.2 驱动的是上臂，则 P2.2 输出低电平后，上臂开始关闭，经过 0.5us，上臂关闭完毕，P2.1 输出高电平，下臂打开。P2.1 输出低电平后，下臂开始关闭，经过 0.5us，下臂关闭完毕，P2.2 输出高电平，上臂打开。这样，死区时间的设置，可以避免上下臂同时打开造成烧毁 MOSFET。

有人会说，一路输出关闭的同时，另一路大开，不会烧管子啊？
错啦，MOSFET 打开快，关闭慢（相关知识请翻翻书），所以需要一段时间关闭。

P2.1 或 P2.2 如果直接用示波器观察，会看到比我们的思绪还凌乱的波形，因为 PWM 一直在变化，但是通过 RC（1K+1uF）低通滤波再观察的话，就会看到两个反相的正弦波，神奇吧，呵呵！

本例使用 24MHz 时钟，PWM 时钟为 1T 模式，PWM 周期 2400，正弦表幅度为 2300，往上偏移 60 个时钟（方便过 0 中断重装数据）。正弦采样为 200 点，则输出正弦波频率 = 24000000/2400/200=50Hz。

下面为实际测量的波形。

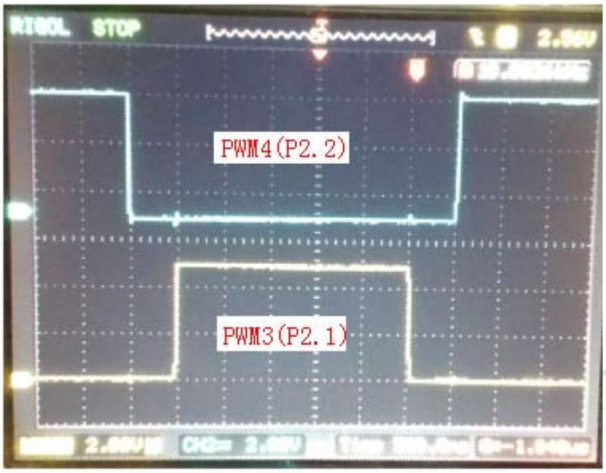


图3： 某个时刻PWM波形，2V/DIV，500nS/DIV

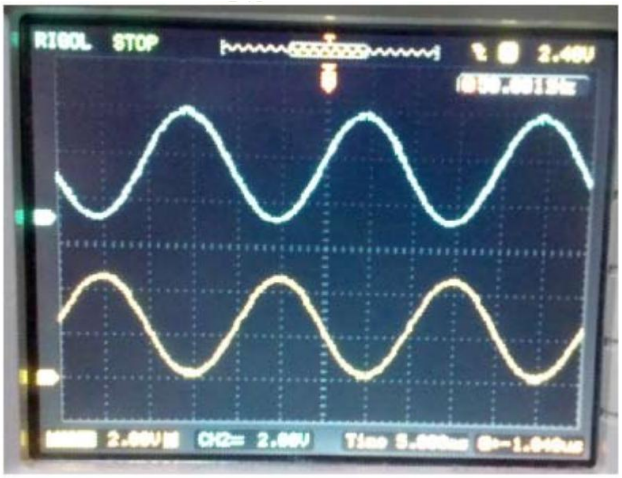


图4：经过RC滤波后两路反相的正弦波50.001HZ

下面是用 STC15W4KxxS4 系列单片机输出两路互补 SPWM 的参考程序（包括 C 语言程序和汇编语言程序）

1、C 语言程序：

```
/* --- STC 1T Series MCU RC Demo -----------------------------------*/

#define    MAIN_Fosc          24000000L          //定义主时钟

#include "STC15Fxxxx.H"
#include "T_SineTable.h"

#define    PWM_DeadZone    12                    //*死区时钟数, 6 ~ 24 之间

/******************* 功能说明 **************
演示使用 2 路 PWM 产生互补或同相的 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.
本程序从 P2.1(PWM3)输出正相脉冲, 从 P2.2(PWM4)输出反相脉冲(互补).
如果需要 P2.2 输出同相的，请在初始化配置中"PWMCFG"项选择设置 1(设置 PWM 输出端口的初始电平，0 或 1).
*****************************************/
u8    PWM_Index;                              //SPWM 查表索引
//========================================================================
// 函数: void PWM_config(void)
// 描述: PWM 配置函数。
// 参数: none.
// 返回: none.
//========================================================================
void PWM_config(void)
{
    u8    xdata    *px;
    EAXSFR();                                 // 访问 XFR

    px = PWM3T1H;                             // 指针指向 PWM3
    *px = 0;                                  // 第一个翻转计数高字节
    px++;
    *px = 65;                                 // 第一个翻转计数低字节
    px++;
    *px = 1220 / 256;                         // 第二个翻转计数高字节
    px++;
    *px = 1220 % 256;                         // 第二个翻转计数低字节
    px++;
    *px = 0;                                  // PWM3 输出选择 P2.1, 无中断
```

```
    PWMCR |= 0x02;              // 相应 PWM 通道的端口为 PWM 输出口，受 PWM 波形发生器控制
    PWMCFG &= ~0x02;                    // 设置 PWM 输出端口的初始电平为 0
//  PWMCFG |= 0x02;                     // 设置 PWM 输出端口的初始电平为 1
    P21 = 0;
    P2n_push_pull(1<<1);                //IO 初始化, 上电时为高阻

    px = PWM4T1H;                       // 指针指向 PWM4
    *px = 0;                            // 第一个翻转计数高字节
    px++;
    *px = 65-PWM_DeadZone;              // 第一个翻转计数低字节
    px++;
    *px = (1220+PWM_DeadZone) / 256;    // 第二个翻转计数高字节
    px++;
    *px = (1220+PWM_DeadZone) % 256;    // 第二个翻转计数低字节
    px++;
    *px = 0;                            // PWM4 输出选择 P2.2, 无中断
    PWMCR |= 0x04;              // 相应 PWM 通道的端口为 PWM 输出口，受 PWM 波形发生器控制
    //  PWMCFG &= ~0x04;                // 设置 PWM 输出端口的初始电平为 0
    PWMCFG |= 0x04;                     // 设置 PWM 输出端口的初始电平为 1
    P22 = 1;
    P2n_push_pull(1<<2);                //IO 初始化, 上电时为高阻

    px = PWMCH;                         // PWM 计数器的高字节
    *px = 2400 / 256;
    px++;
    *px = 2400 % 256;                   // PWM 计数器的低字节
    px++;                               // PWMCKS, PWM 时钟选择
    *px = PwmClk_1T;            // 时钟源: PwmClk_1T,PwmClk_2T, ... PwmClk_16T, PwmClk_Timer2

    EAXRAM();                           // 恢复访问 XRAM

    PWMCR |= ENPWM;                     // 使能 PWM 波形发生器， PWM 计数器开始计数
//  PWMCR &= ~ECBI;                     // 禁止 PWM 计数器归零中断
    PWMCR |= ECBI;                      // 允许 PWM 计数器归零中断
//  PWMFDCR = ENFD | FLTFLIO | FDIO;
                                //  PWM 失效中断控制, ENFD | FLTFLIO | EFDI | FDCMP | FDIO
}
/*******************************************/
void main(void)
{
    PWM_config();                       //初始化 PWM
    EA = 1;                             //允许全局中断
    while (1)
    {
    }
}
```

```
/********************* PWM 中断函数************************/
void PWM_int (void) interrupt PWM_VECTOR
{
    u8      xdata *px;
    u16     j;
    u8      SW2_tmp;
    if(PWMIF & CBIF)                    //PWM 计数器归零中断标志
    {
        PWMIF &= ~CBIF;                 //清除中断标志

        SW2_tmp = P_SW2;                //保存 SW2 设置
        EAXSFR();                       //访问 XFR
        px = PWM3T2H;                   //  指向 PWM3
        j = T_SinTable[PWM_Index];
        *px = (u8)(j >> 8);             //第二个翻转计数高字节
        px++;
        *px = (u8)j;                    //第二个翻转计数低字节

        j += PWM_DeadZone;              //死区
        px = PWM4T2H;                   //  指向 PWM4
        *px = (u8)(j >> 8);             //第二个翻转计数高字节
        px++;
        *px = (u8)j;                    //第二个翻转计数低字节
        P_SW2 = SW2_tmp;                //恢复 SW2 设置
        if(++PWM_Index >= 200)  PWM_Index = 0;
    }
/*  if(PWMIF & C2IF)                    //PWM2 中断标志
    {
        PWMIF &= ~C2IF;                 //清除中断标志
    }
    if(PWMIF & C3IF)                    //PWM3 中断标志
    {
        PWMIF &= ~C3IF;                 //清除中断标志
    }
    if(PWMIF & C4IF)                    //PWM4 中断标志
    {
        PWMIF &= ~C4IF;                 //清除中断标志
    }
    if(PWMIF & C5IF)                    //PWM5 中断标志
    {
        PWMIF &= ~C5IF;                 //清除中断标志
    }
    if(PWMIF & C6IF)                    //PWM6 中断标志
    {
        PWMIF &= ~C6IF;                 //清除中断标志
    }
```

```
    if(PWMIF & C7IF)                    //PWM7 中断标志
    {
        PWMIF &= ~C7IF;                 //清除中断标志
    }
    */
}
```

2、汇编语言程序：

```
;/* --- STC 1T Series MCU RC Demo -----------------------------------*/
PWM_DeadZone    EQU     12          ; 死区时钟数, 6 ~ 24 之间
;******************  功能说明  **************
;演示使用 2 路 PWM 产生互补或同相的 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.
;本程序从 P2.1(PWM3)输出正相脉冲, 从 P2.2(PWM4)输出反相脉冲(互补).
;如果需要 P2.2 输出同相的, 请在初始化配置中"PWMCFG"项选择设置 1(设置 PWM 输出端口的初始电平, 0 或 1).
;*************************************************/
P2M1            DATA    0x95        ; P2M1.n, P2M0.n =00--->Standard, 01--->push-pull
P2M0            DATA    0x96        ;                 =10--->pure input, 11--->open drain
P_SW2           DATA    0xBA
PWMCFG          DATA    0xF1        ; PWM 配置寄存器
PWMCR           DATA    0xF5        ; PWM 控制寄存器
PWMIF           DATA    0xF6        ; PWM 中断标志寄存器
PWMFDCR         DATA    0xF7        ; PWM 外部异常控制寄存器

PWMCH           EQU     0xFFF0      ; PWM 计数器高字节
PWMCL           EQU     0xFFF1      ; PWM 计数器低字节
PWMCKS          EQU     0xFFF2      ; PWM 时钟选择
PWM2T1H         EQU     0xFF00      ; PWM2T1 计数高字节
PWM2T1L         EQU     0xFF01      ; PWM2T1 计数低字节
PWM2T2H         EQU     0xFF02      ; PWM2T2 计数高字节
PWM2T2L         EQU     0xFF03      ; PWM2T2 计数低字节
PWM2CR          EQU     0xFF04      ; PWM2 控制

PWM3T1H         EQU     0xFF10      ; PWM3T1 计数高字节
PWM3T1L         EQU     0xFF11      ; PWM3T1 计数低字节
PWM3T2H         EQU     0xFF12      ; PWM3T2 计数高字节
PWM3T2L         EQU     0xFF13      ; PWM3T2 计数低字节
PWM3CR          EQU     0xFF14      ; PWM3 控制

PWM4T1H         EQU     0xFF20      ; PWM4T1 计数高字节
```

```
PWM4T1L         EQU     0xFF21          ; PWM4T1 计数低字节
PWM4T2H         EQU     0xFF22          ; PWM4T2 计数高字节
PWM4T2L         EQU     0xFF23          ; PWM4T2 计数低字节
PWM4CR          EQU     0xFF24          ; PWM4 控制

PWM5T1H         EQU     0xFF30          ; PWM5T1 计数高字节
PWM5T1L         EQU     0xFF31          ; PWM5T1 计数低字节
PWM5T2H         EQU     0xFF32          ; PWM5T2 计数高字节
PWM5T2L         EQU     0xFF33          ; PWM5T2 计数低字节
PWM5CR          EQU     0xFF34          ; PWM5 控制

PWM6T1H         EQU     0xFF40          ; PWM6T1 计数高字节
PWM6T1L         EQU     0xFF41          ; PWM6T1 计数低字节
PWM6T2H         EQU     0xFF42          ; PWM6T2 计数高字节
PWM6T2L         EQU     0xFF43          ; PWM6T2 计数低字节
PWM6CR          EQU     0xFF44          ; PWM6 控制

PWM7T1H         EQU     0xFF50          ; PWM7T1 计数高字节
PWM7T1L         EQU     0xFF51          ; PWM7T1 计数低字节
PWM7T2H         EQU     0xFF52          ; PWM7T2 计数高字节
PWM7T2L         EQU     0xFF53          ; PWM7T2 计数低字节
PWM7CR          EQU     0xFF54          ; PWM7 控制

PwmClk_1T       EQU     0
PwmClk_2T       EQU     1
PwmClk_3T       EQU     2
PwmClk_4T       EQU     3
PwmClk_5T       EQU     4
PwmClk_6T       EQU     5
PwmClk_7T       EQU     6
PwmClk_8T       EQU     7
PwmClk_9T       EQU     8
PwmClk_10T      EQU     9
PwmClk_11T      EQU     10
PwmClk_12T      EQU     11
PwmClk_13T      EQU     12
PwmClk_14T      EQU     13
PwmClk_15T      EQU     14
PwmClk_16T      EQU     15
PwmClk_Timer2   EQU     16
;*******************************************************************
PWM_Index           DATA    30H         ; SPWM 查表索引
STACK_POIRTER       EQU     0C0H        ;堆栈开始地址
;*******************************************************************
    ORG         00H                     ;reset
    LJMP        F_Main
```

```
    ORG     00B3H                   ; PWM interrupt
    LJMP    F_PWM_Interrupt
;******************** 主程序 **************************/
F_Main:
    MOV     SP, #STACK_POIRTER
    MOV     PSW, #0
    USING   0                       ;选择第 0 组 R0~R7
;================= 用户初始化程序 ====================================
    ORL     P_SW2, #0x80            ; 访问 XFR

    MOV     DPTR, #PWM3T1H          ; 指针指向 PWM3
    CLR     A
    MOVX    @DPTR, A                ; 第一个翻转计数高字节
    INC     DPTR
    MOV     A, #65                  ; 第一个翻转计数低字节
    MOVX    @DPTR, A
    INC     DPTR
    MOV     A, #HIGH 1220           ; 第二个翻转计数高字节
    MOVX    @DPTR, A
    INC     DPTR
    MOV     A, #LOW 1220            ; 第二个翻转计数低字节
    MOVX    @DPTR, A

    INC     DPTR
    CLR     A                       ; PWM3 输出选择 P2.1, 无中断
    MOVX    @DPTR, A
    ORL     PWMCR, #0x02            ; 相应 PWM 通道的端口为 PWM 输出口，
                                    ; 受 PWM 波形发生器控制
    ANL     PWMCFG, #NOT 0x02       ; 设置 PWM 输出端口的初始电平为 0
;   ORL     PWMCFG, #0x02           ; 设置 PWM 输出端口的初始电平为 1
    CLR     P2.1                    ; P2.1 输出低电平
    ANL     P2M1, #NOT 0x02         ; P2.1 设置为推挽输出
    ORL     P2M0, #0x02

    MOV     DPTR, #PWM4T1H          ; 指针指向 PWM4
    CLR     A
    MOVX    @DPTR, A                ; 第一个翻转计数高字节
    INC     DPTR
    MOV     A, #(65-PWM_DeadZone)   ; 第一个翻转计数低字节
    MOVX    @DPTR, A
    INC     DPTR
    MOV     A, #HIGH (1220+PWM_DeadZone)        ; 第二个翻转计数高字节
    MOVX    @DPTR, A
    INC     DPTR
    MOV     A, #LOW (1220+PWM_DeadZone)         ; 第二个翻转计数低字节
    MOVX    @DPTR, A
```

```
    INC     DPTR

    CLR     A                   ; PWM4 输出选择 P2.2, 无中断
    MOVX    @DPTR, A
    ORL     PWMCR, #0x04        ; 相应 PWM 通道的端口为 PWM 输出口，
                                ; 受 PWM 波形发生器控制
    ANL     PWMCFG, #NOT 0x04   ; 设置 PWM 输出端口的初始电平为 0
    ; ORL   PWMCFG, #0x04       ; 设置 PWM 输出端口的初始电平为 1
    SETB    P2.2                ; P2.2 输出高电平
    ANL     P2M1, #NOT 0x04     ; P2.2 设置为推挽输出
    ORL     P2M0, #0x04

    MOV     DPTR, #PWMCH        ; 指针指向 PWMCH
    MOV     A, #HIGH 2400       ; PWM 计数器的高字节
    MOVX    @DPTR, A
    INC     DPTR
    MOV     A, #LOW 2400        ; PWM 计数器的低字节
    MOVX    @DPTR, A
    INC     DPTR
    MOV     A, #PwmClk_1T       ; 时钟源: PwmClk_1T,PwmClk_2T, ... PwmClk_16T, PwmClk_Timer2
    MOVX    @DPTR, A            ; PWMCKS, PWM 时钟选择

    ANL     P_SW2, #NOT 0x80    ; 恢复访问 XRAM

    ORL     PWMCR, #0xC0        ; 允许 PWM 计数器归零中断, 使能 PWM 波形发生
                                ; 器， PWM 计数器开始计数

;   MOV     PWMFDCR, # (ENFD + FLTFLIO + FDIO)
                                ; PWM 失效中断控制, ENFD | FLTFLIO | EFDI | FDCMP | FDIO
    SETB    EA                  ; 允许全局中断
;==================== 主循环 ================================
L_MainLoop:
    LJMP    L_MainLoop
;======================================================================
;********************** PWM 中断函数************************/
F_PWM_Interrupt:
    PUSH    PSW
    PUSH    ACC
    PUSH    DPH
    PUSH    DPL
    PUSH    AR2
    PUSH    AR3

    MOV     DPTR, #T_SinTable   ; 读正弦波表
    MOV     A, PWM_Index
    ADD     A, DPL
```

```
MOV     DPL, A
CLR     A
ADDC    A, DPH
MOV     DPH, A
MOV     A, PWM_Index
ADD     A, DPL
MOV     DPL, A
CLR     A
ADDC    A, DPH
MOV     DPH, A
CLR     A
MOVC    A, @A+DPTR
MOV     R2, A
MOV     A, #1
MOVC    A, @A+DPTR
MOV     R3, A

MOV     A, PWMIF                ; 读中断标志寄存器
JNB     ACC.6, L_Int_NotCBIF    ; PWM 计数器归零中断标志
ANL     PWMIF, #NOT (1 SHL 6)   ; 清除中断标志
PUSH    P_SW2                   ; 保存 SW2 设置
ORL     P_SW2, #0x80            ; 访问 XFR
MOV     DPTR, #PWM3T2H          ; 指针指向 PWM3

MOV     A, R2                   ; 第二个翻转计数高字节
MOVX    @DPTR, A
INC     DPTR
MOV     A, R3                   ; 第二个翻转计数低字节
MOVX    @DPTR, A
MOV     A, R3

ADD     A, #PWM_DeadZone        ; 加死区
MOV     R3, A
CLR     A
ADDC    A, R2
MOV     R2, A

MOV     DPTR, #PWM4T2H          ; 指针指向 PWM4
MOV     A, R2                   ; 第二个翻转计数高字节
MOVX    @DPTR, A
INC     DPTR
MOV     A, R3                   ; 第二个翻转计数低字节
MOVX    @DPTR, A
POP     P_SW2                   ; 恢复访问 P_SW2

INC     PWM_Index
```

```
    MOV     A, PWM_Index
    CLR     C
    SUBB    A, #200
    JC      L_Int_NotCBIF
    MOV     PWM_Index, #0

L_Int_NotCBIF:
/*
    MOV     A, PWMIF            ; 读中断标志寄存器
    JNB     ACC.0, L_Int_NotC2IF    ; PWM2 中断标志
    ANL     PWMIF, #NOT 1       ; 清除中断标志

L_Int_NotC2IF:
    MOV     A, PWMIF            ; 读中断标志寄存器
    JNB     ACC.1, L_Int_NotC3IF    ; PWM3 中断标志
    ANL     PWMIF, #NOT (1 SHL 1)   ; 清除中断标志

L_Int_NotC3IF:
    MOV     A, PWMIF            ; 读中断标志寄存器
    JNB     ACC.2, L_Int_NotC4IF    ; PWM4 中断标志
    ANL     PWMIF, #NOT (1 SHL 2)   ; 清除中断标志

L_Int_NotC4IF:
    MOV     A, PWMIF            ; 读中断标志寄存器
    JNB     ACC.3, L_Int_NotC5IF    ; PWM5 中断标志
    ANL     PWMIF, #NOT (1 SHL 3)   ; 清除中断标志

L_Int_NotC5IF:
    MOV     A, PWMIF            ; 读中断标志寄存器
    JNB     ACC.4, L_Int_NotC6IF    ; PWM6 中断标志
    ANL     PWMIF, #NOT (1 SHL 4)   ; 清除中断标志

L_Int_NotC6IF:
    MOV     A, PWMIF            ; 读中断标志寄存器
    JNB     ACC.5, L_Int_NotC7IF    ; PWM7 中断标志
    ANL     PWMIF, #NOT (1 SHL 5)   ; 清除中断标志

L_Int_NotC7IF:
*/
    POP     AR3
    POP     AR2
    POP     DPL
    POP     DPH
    POP     ACC
    POP     PSW
    RETI
```

```
T_SinTable:
    DW 1220,1256,1292,1328,1364,1400,1435,1471,1506,1541,1575,1610,1643,1677,1710,1742,1774,1805,1836,1866
    DW 1896,1925,1953,1981,2007,2033,2058,2083,2106,2129,2150,2171,2191,2210,2228,2245,2261,2275,2289,2302
    DW 2314,2324,2334,2342,2350,2356,2361,2365,2368,2369,2370,2369,2368,2365,2361,2356,2350,2342,2334,2324
    DW 2314,2302,2289,2275,2261,2245,2228,2210,2191,2171,2150,2129,2106,2083,2058,2033,2007,1981,1953,1925
    DW 1896,1866,1836,1805,1774,1742,1710,1677,1643,1610,1575,1541,1506,1471,1435,1400,1364,1328,1292,1256
    DW 1220,1184,1148,1112,1076,1040,1005,969,934,899,865,830,797,763,730,698,666,635,604,574
    DW 544,515,487,459,433,407,382,357,334,311,290,269,249,230,212,195,179,165,151,138
    DW 126,116,106,98,90,84,79,75,72,71,70,71,72,75,79,84,90,98,106,116
    DW 126,138,151,165,179,195,212,230,249,269,290,311,334,357,382,407,433,459,487,515
    DW 544,574,604,635,666,698,730,763,797,830,865,899,934,969,1005,1040,1076,1112,1148,1184

END
```

23.6 用 STC15W4K 系列的 PWM 实现渐变灯的示例程序

1、C 语言程序

```
/* --- STC15 系列  使用 PWM 实现渐变灯实例----------------------------------------*/
//本示例在 Keil 开发环境下请选择 Intel 的 8058 芯片型号进行编译
//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"

#define   CYCLE      0x1000L                          //定义 PWM 周期

#define   PWMC       (*(unsigned int     volatile xdata *)0xfff0)
#define   PWMCH      (*(unsigned char    volatile xdata *)0xfff0)
#define   PWMCL      (*(unsigned char    volatile xdata *)0xfff1)
#define   PWMCKS     (*(unsigned char    volatile xdata *)0xfff2)

#define   PWM2T1     (*(unsigned int     volatile xdata *)0xff00)
#define   PWM2T1H    (*(unsigned char    volatile xdata *)0xff00)
#define   PWM2T1L    (*(unsigned char    volatile xdata *)0xff01)
#define   PWM2T2     (*(unsigned int     volatile xdata *)0xff02)
#define   PWM2T2H    (*(unsigned char    volatile xdata *)0xff02)
#define   PWM2T2L    (*(unsigned char    volatile xdata *)0xff03)
#define   PWM2CR     (*(unsigned char    volatile xdata *)0xff04)

#define   PWM3T1     (*(unsigned int     volatile xdata *)0xff10)
#define   PWM3T1H    (*(unsigned char    volatile xdata *)0xff10)
#define   PWM3T1L    (*(unsigned char    volatile xdata *)0xff11)
#define   PWM3T2     (*(unsigned int     volatile xdata *)0xff12)
#define   PWM3T2H    (*(unsigned char    volatile xdata *)0xff12)
#define   PWM3T2L    (*(unsigned char    volatile xdata *)0xff13)
#define   PWM3CR     (*(unsigned char    volatile xdata *)0xff14)

#define   PWM4T1     (*(unsigned int     volatile xdata *)0xff20)
#define   PWM4T1H    (*(unsigned char    volatile xdata *)0xff20)
#define   PWM4T1L    (*(unsigned char    volatile xdata *)0xff21)
#define   PWM4T2     (*(unsigned int     volatile xdata *)0xff22)
#define   PWM4T2H    (*(unsigned char    volatile xdata *)0xff22)
#define   PWM4T2L    (*(unsigned char    volatile xdata *)0xff23)
#define   PWM4CR     (*(unsigned char    volatile xdata *)0xff24)

#define   PWM5T1     (*(unsigned int     volatile xdata *)0xff30)
#define   PWM5T1H    (*(unsigned char    volatile xdata *)0xff30)
#define   PWM5T1L    (*(unsigned char    volatile xdata *)0xff31)
#define   PWM5T2     (*(unsigned int     volatile xdata *)0xff32)
#define   PWM5T2H    (*(unsigned char    volatile xdata *)0xff32)
```

```
#define   PWM5T2L    (*(unsigned char   volatile xdata *)0xff33)
#define   PWM5CR     (*(unsigned char   volatile xdata *)0xff34)

#define   PWM6T1     (*(unsigned int    volatile xdata *)0xff40)
#define   PWM6T1H    (*(unsigned char   volatile xdata *)0xff40)
#define   PWM6T1L    (*(unsigned char   volatile xdata *)0xff41)
#define   PWM6T2     (*(unsigned int    volatile xdata *)0xff42)
#define   PWM6T2H    (*(unsigned char   volatile xdata *)0xff42)
#define   PWM6T2L    (*(unsigned char   volatile xdata *)0xff43)
#define   PWM6CR     (*(unsigned char   volatile xdata *)0xff44)

#define   PWM7T1     (*(unsigned int    volatile xdata *)0xff50)
#define   PWM7T1H    (*(unsigned char   volatile xdata *)0xff50)
#define   PWM7T1L    (*(unsigned char   volatile xdata *)0xff51)
#define   PWM7T2     (*(unsigned int    volatile xdata *)0xff52)
#define   PWM7T2H    (*(unsigned char   volatile xdata *)0xff52)
#define   PWM7T2L    (*(unsigned char   volatile xdata *)0xff53)
#define   PWM7CR     (*(unsigned char   volatile xdata *)0xff54)

sfr       PIN_SW2 = 0xba;

sfr       P0M1 = 0x93;
sfr       P0M0 = 0x94;
sfr       P1M1 = 0x91;
sfr       P1M0 = 0x92;
sfr       P2M1 = 0x95;
sfr       P2M0 = 0x96;
sfr       P3M1 = 0xb1;
sfr       P3M0 = 0xb2;
sfr       P4M1 = 0xb3;
sfr       P4M0 = 0xb4;
sfr       P5M1 = 0xC9;
sfr       P5M0 = 0xCA;
sfr       P6M1 = 0xCB;
sfr       P6M0 = 0xCC;
sfr       P7M1 = 0xE1;
sfr       P7M0 = 0xE2;

sfr       PWMCFG = 0xf1;
sfr       PWMCR = 0xf5;
sfr       PWMIF = 0xf6;
sfr       PWMFDCR = 0xf7;

void pwm_isr() interrupt 22 using 1
{
    static bit dir = 1;
```

```
    static int val = 0;

    if (PWMIF & 0x40)
    {
        PWMIF &= ~0x40;
        if (dir)
        {
            val++;
            if (val >= CYCLE) dir = 0;
        }
        else
        {
            val--;
            if (val <= 1) dir = 1;
        }
        PIN_SW2 |= 0x80;
        PWM2T2 = val;
        PIN_SW2 &= ~0x80;
    }
}

void main()
{
    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;

    PIN_SW2 |= 0x80;            //使能访问 XSFR
    PWMCFG = 0x00;              //配置 PWM 的输出初始电平为低电平
    PWMCKS = 0x00;              //选择 PWM 的时钟为 Fosc/1
    PWMC = CYCLE;               //设置 PWM 周期
    PWM2T1 = 0x0000;            //设置 PWM2 第 1 次反转的 PWM 计数
    PWM2T2 = 0x0001;            //设置 PWM2 第 2 次反转的 PWM 计数
```

```
                                //占空比为(PWM2T2-PWM2T1)/PWMC
    PWM2CR = 0x00;              //选择 PWM2 输出到 P3.7,不使能 PWM2 中断
    PWMCR = 0x01;               //使能 PWM 信号输出
    PWMCR |= 0x40;              //使能 PWM 归零中断
    PWMCR |= 0x80;              //使能 PWM 模块

    PIN_SW2 &= ~0x80;
    EA = 1;
    while (1);
}
```

2、汇编语言程序

```
/* --- STC15 系列  使用 PWM 实现渐变灯实例----------------------------------------------*/
//本示例在 Keil 开发环境下请选择 Intel 的 8058 芯片型号进行编译
//假定测试芯片的工作频率为 18.432MHz

CYCLE       EQU     1000H       ;定义 PWM 周期(最大值为 32767)

PWMC        EQU     0FFF0H
PWMCH       EQU     0FFF0H
PWMCL       EQU     0FFF1H
PWMCKS      EQU     0FFF2H

PWM2T1      EQU     0FF00H
PWM2T1H     EQU     0FF00H
PWM2T1L     EQU     0FF01H
PWM2T2      EQU     0FF02H
PWM2T2H     EQU     0FF02H
PWM2T2L     EQU     0FF03H
PWM2CR      EQU     0FF04H

PWM3T1      EQU     0FF10H
PWM3T1H     EQU     0FF10H
PWM3T1L     EQU     0FF11H
PWM3T2      EQU     0FF12H
PWM3T2H     EQU     0FF12H
PWM3T2L     EQU     0FF13H
PWM3CR      EQU     0FF14H

PWM4T1      EQU     0FF20H
PWM4T1H     EQU     0FF20H
PWM4T1L     EQU     0FF21H
PWM4T2      EQU     0FF22H
PWM4T2H     EQU     0FF22H
PWM4T2L     EQU     0FF23H
```

```
PWM4CR      EQU     0FF24H

PWM5T1      EQU     0FF30H
PWM5T1H     EQU     0FF30H
PWM5T1L     EQU     0FF31H
PWM5T2      EQU     0FF32H
PWM5T2H     EQU     0FF32H
PWM5T2L     EQU     0FF33H
PWM5CR      EQU     0FF34H

PWM6T1      EQU     0FF40H
PWM6T1H     EQU     0FF40H
PWM6T1L     EQU     0FF41H
PWM6T2      EQU     0FF42H
PWM6T2H     EQU     0FF42H
PWM6T2L     EQU     0FF43H
PWM6CR      EQU     0FF44H

PWM7T1      EQU     0FF50H
PWM7T1H     EQU     0FF50H
PWM7T1L     EQU     0FF51H
PWM7T2      EQU     0FF52H
PWM7T2H     EQU     0FF52H
PWM7T2L     EQU     0FF53H
PWM7CR      EQU     0FF54H

PIN_SW2     DATA    0BAH

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0b1H
P3M0        DATA    0b2H
P4M1        DATA    0b3H
P4M0        DATA    0b4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH
P6M1        DATA    0CBH
P6M0        DATA    0CCH
P7M1        DATA    0E1H
P7M0        DATA    0E2H

PWMCFG      DATA    0F1H
```

```
PWMCR       DATA    0F5H
PWMIF       DATA    0F6H
PWMFDCR     DATA    0F7H

DIR         BIT     20H.0
VALL        DATA    21H
VALH        DATA    22H

MOVXB       MACRO ADR, DAT
    MOV     DPTR, #ADR
    MOV     A, #DAT
    MOVX    @DPTR, A
    ENDM

MOVXW       MACRO ADR, DAT
    MOV     DPTR, #ADR
    MOV     A, #HIGH DAT
    MOVX    @DPTR, A
    INC     DPTR
    MOV     A, #LOW DAT
    MOVX    @DPTR, A
    ENDM

    ORG     0000H
    LJMP    START

    ORG     00BBH              ;PWM interrupt entry
    LJMP    PWM_ISR

    ORG     0100H
START:
    MOV     SP, #5FH

    MOV     P0M0, #00H
    MOV     P0M1, #00H
    MOV     P1M0, #00H
    MOV     P1M1, #00H
    MOV     P2M0, #00H
    MOV     P2M1, #00H
    MOV     P3M0, #00H
    MOV     P3M1, #00H
    MOV     P4M0, #00H
    MOV     P4M1, #00H
    MOV     P5M0, #00H
    MOV     P5M1, #00H
    MOV     P6M0, #00H
```

```
    MOV     P6M1, #00H
    MOV     P7M0, #00H
    MOV     P7M1, #00H

    ORL     PIN_SW2, #80H       ;使能访问 XSFR
    MOV     PWMCFG, #00H        ;配置 PWM 的输出初始电平为低电平
    MOVXB   PWMCKS, 00H         ;选择 PWM 的时钟为 Fosc/(0+1)
    MOVXW   PWMC, CYCLE         ;设置 PWM 周期
    MOVXW   PWM2T1, 0000H       ;设置 PWM2 第 1 次反转的 PWM 计数
    MOVXW   PWM2T2, 0001H       ;设置 PWM2 第 2 次反转的 PWM 计数
                                ;占空比为(PWM2T2-PWM2T1)/PWMC
    MOVXB   PWM2CR, 00H         ;选择 PWM2 输出到 P3.7,不使能 PWM2 中断
    MOV     PWMCR, #01H         ;使能 PWM 信号输出
    ORL     PWMCR, #40H         ;使能 PWM 归零中断
    ORL     PWMCR, #80H         ;使能 PWM 模块
    ANL     PIN_SW2, #NOT 80H

    SETB    DIR
    CLR     A
    MOV     VALH, A
    MOV     VALL, A
    SETB    EA
    JMP     $

PWM_ISR:
    PUSH    ACC
    PUSH    PSW
    PUSH    DPH
    PUSH    DPL
    MOV     PSW, #08H
    MOV     A, PWMIF
    ANL     A, #40H
    JZ      PWMISREXIT
    XRL     PWMIF, A
    JNB     DIR, PWMDN

PWMUP:
    MOV     A, VALL
    ADD     A, #1
    MOV     VALL, A
    MOV     A, VALH
    ADDC    A, #0
    MOV     VALH, A
    CJNE    A, #HIGH CYCLE, SETPWM
    MOV     A, VALL
    CJNE    A, #LOW CYCLE, SETPWM
```

```
    CLR     DIR
    JMP     SETPWM

PWMDN:
    MOV     A, VALL
    ADD     A, #0FFH
    MOV     VALL, A
    MOV     A, VALH
    ADDC    A, #0FFH
    MOV     VALH, A
    JNZ     SETPWM
    MOV     A, VALL
    CJNE    A, #1, SETPWM
    SETB    DIR

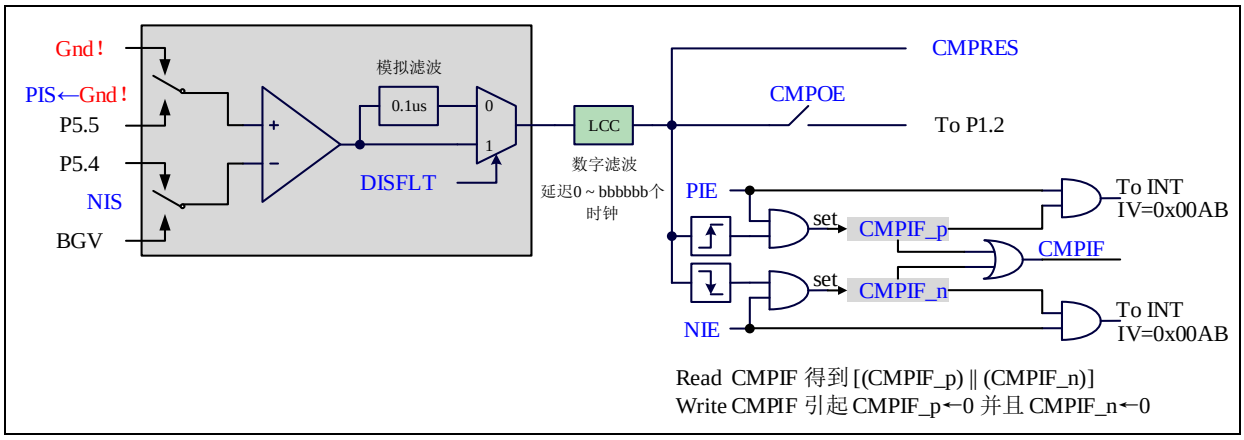
SETPWM:
    ORL     PIN_SW2, #80H
    MOV     DPTR, #PWM2T2
    MOV     A, VALH
    MOVX    @DPTR, A
    INC     DPTR
    MOV     A, VALL
    MOVX    @DPTR, A
    ANL     PIN_SW2, #7FH

PWMISREXIT:
    POP     DPL
    POP     DPH
    POP     PSW
    POP     ACC

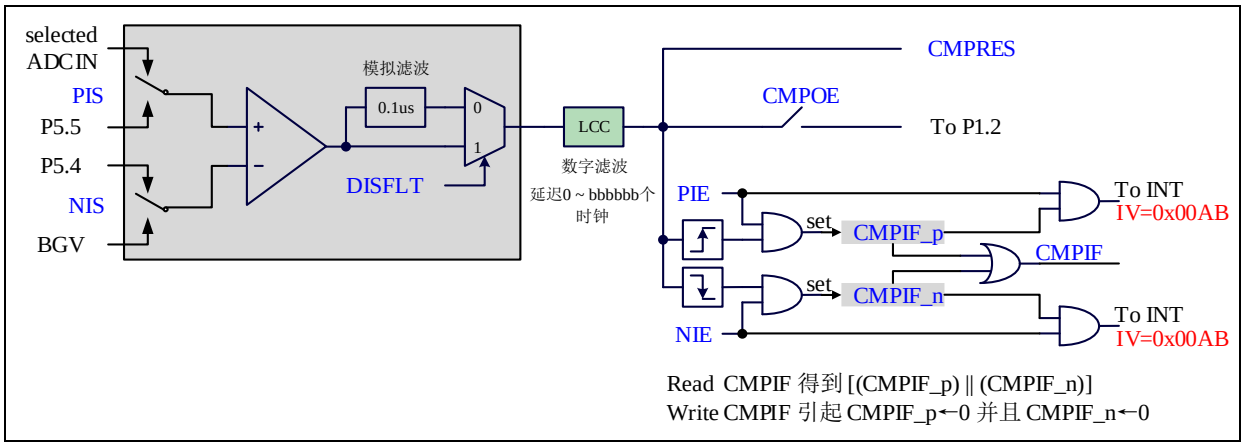
    RETI
    END
```

24 STC15W 系列的比较器

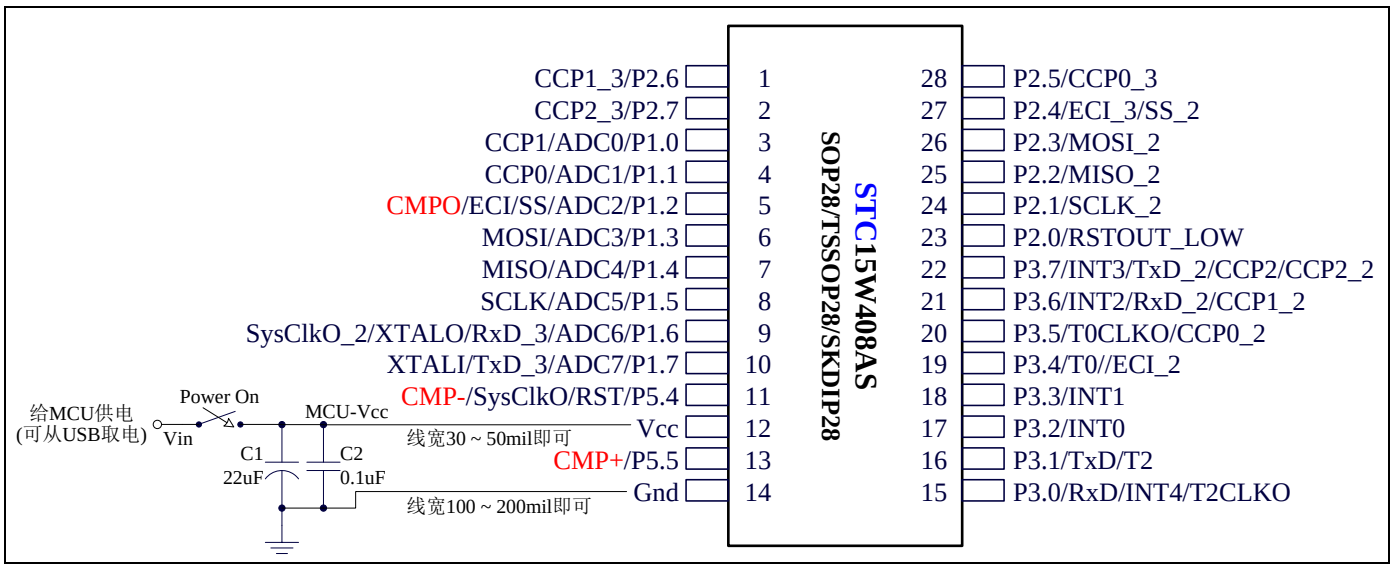
STC15W 系列单片机(如 STC15W408AS 系列、STC15W204S 系列、STC15W404S 系列、STC15W1K16S 系列及 STC15W4K32S4 系列)内置比较器功能。其中 STC15W204S 系列、STC15W404S 系列及 STC15W1K16S 系列的比较器内部规划如下图所示：



其中，有 ADC 的单片机 STC15W408AS 系列及 STC15W4K32S4 系列的比较器内部规划如下图所示：



比较器相关管脚在单片机管脚图中的位置：



比较器正极输入端 CMP+电平可以与比较器负极输入端 CMP-的电平进行比较，也可以与内部 BandGap 参考电呀（1.27V 附近）进行比较

STC15W 系列与比较器相关的特殊功能寄存器（STC15W SFRs associated with comparator）

符号	描述	地址	位地址及其符号								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	PIS	NIS	CMPOE	CMPRES	0000,0000
CMPCR2	比较器控制寄存器 2	E7H	INVCMPO	DISFLT	LCDTY[5:0]						0000,1001

1、比较器控制寄存器 1：CMPCR1

比较器控制寄存器 1 的格式如下：

CMPCR1：比较器控制寄存器 1

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR1	E6H	name	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES

CMPEN：比较器模块使能位

- CMPEN=1，使能比较器模块；
- CMPEN=0，禁用比较器模块，比较器的电源关闭。

CMPIF：比较器中断标志位（Interrupt Flag）

- 在 CMPEN 为 1 的情况下：

  当比较器的比较结果由 LOW 变成 HIGH 时，若是 PIE 被设置成 1，那么内建的某一个叫做 CMPIF_p 的寄存器会被设置成 1；

  当比较器的比较结果由 HIGH 变成 LOW 时，若是 NIE 被设置成 1，那么内建的某一个叫做 CMPIF_n 的寄存器会被设置成 1；

  当 CPU 去读取 CMPIF 的数值时，会读到（CMPIF_p || CMPIF_n）；

  当 CPU 对 CMPIF 写 0 后，CMPIF_p 以及 CMPIF_n 都会被清除为 0。

  而中断产生的条件是[(EA==1)&&(((PIE==1)&&(CMPIF_p==1))||((NIE==1)&&(CMPIF_n==1)))]

  CPU 接受中断后，并不会自动清除此 CMPIF 标志，用户必须用软件写”0”去清除它。

PIE： 比较器上升沿中断使能位（Pos-edge Interrupt Enabling）

- PIE = 1，使能比较器由 LOW 变 HIGH 的事件 设定 CMPIF_p/产生中断；
- PIE = 0，禁用比较器由 LOW 变 HIGH 的事件 设定 CMPIF_p/产生中断。

NIE： 比较器下降沿中断使能位（Neg-edge Interrupt Enabling）

- NIE = 1，使能比较器由 HIGH 变 LOW 的事件 设定 CMPIF_n/产生中断；
- NIE = 0，禁用比较器由 HIGH 变 LOW 的事件 设定 CMPIF_n/产生中断。

PIS： 比较器正极选择位

- PIS = 1，选择 ADCIS[2:0]所选择到的 ADCIN 做为比较器的正极输入源；
- PIS = 0，选择外部 P5.5 为比较器的正极输入源。

NIS： 比较器负极选择位

- NIS = 1，选择外部管脚 P5.4 为比较器的负极输入源；
- NIS = 0，选择内部 BandGap 电压 BGV 为比较器的负极输入源。

CMPOE：比较结果输出控制位

- CMPOE = 1，使能比较器的比较结果输出到 P1.2；
- CMPOE = 0，禁止比较器的比较结果输出。

CMPRES：比较器比较结果 （Comparator Result）标志位

- CMPRES = 1，CMP+的电平高于 CMP-的电平（或内部 BandGap 参考电压的电平）；
- CMPRES = 0，CMP+的电平低于 CMP-的电平（或内部 BandGap 参考电压的电平）。

此 bit 是一个”只读（read-only）”的 bit；软件对它做写入的动作没有任何意义。软件所读到的结果是“经过 ENLCCTL 控制后的结果”，而非 Analog 比较器的直接输出结果.

2、比较器控制寄存器 2：CMPCR2

比较器控制寄存器 2 的格式如下：

CMPCR2：比较器控制寄存器 2

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR2	E7H	name	INVCMPO	DISFLT	LCDTY[5:0]					

INVCMPO：比较器输出取反控制位 （Inverse Comparator Output）

- INVCMPO = 1，比较器取反后再输出到 P1.2；
- INVCMPO = 0，比较器正常输出。

比较器的输出，采用“经过 ENLCCTL 控制后的结果”，而非 Analog 比较器的直接输出结果。

DISFLT：去除比较器输出的 0.1uS Filter

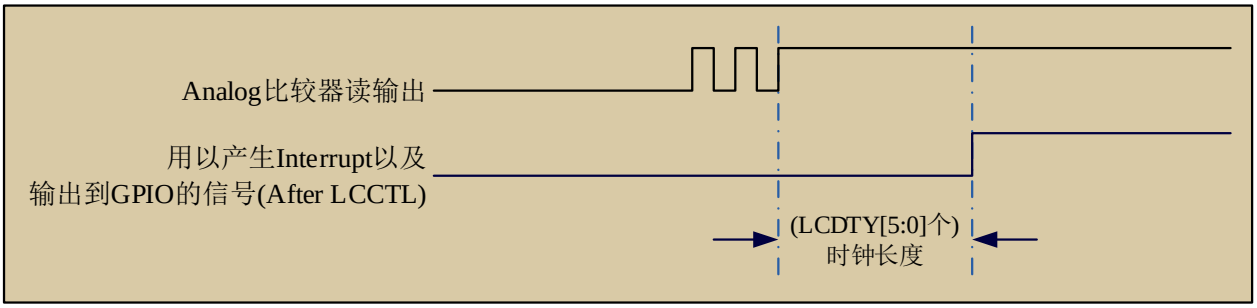
- DISFLT = 1，关掉比较器输出的 0.1uS Filter （可以让比较器速度有少许提升）；
- DISFLT = 0，比较器的输出有 0.1uS 的 Filter。

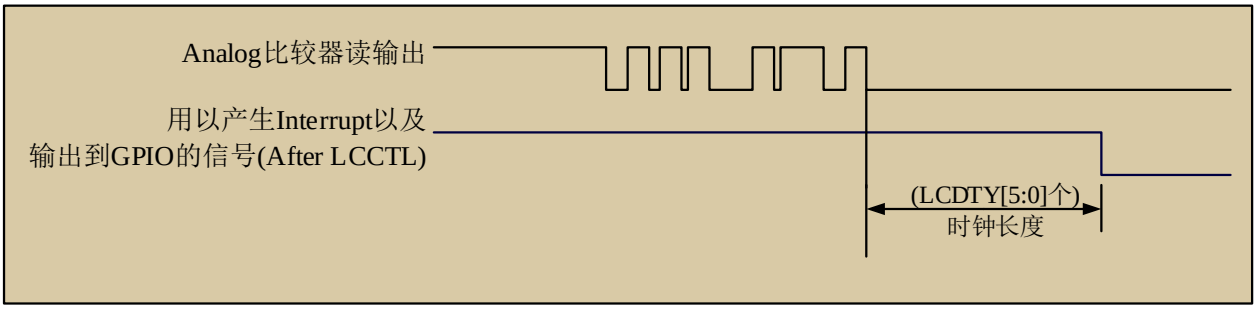
LCDTY[5:0]：比较器输出端 Level-Change control 的 filter 长度（Duty）选择

bbbbbb：=

- 当比较器由 LOW 变 HIGH，必须侦测到该后来的 HIGH 持续至少 bbbbbb 个时钟，此芯片线路才认定比较器的输出是由 LOW 转成 HIGH；如果在 bbbbbb 个时钟内，Analog 比较器的输出又回复到 LOW，此芯片线路认为甚么都没发生，视同比较器的输出一直维持在 LOW；
- 当比较器由 HIGH 变 LOW，必须侦测到该后来的 LOW 持续至少 bbbbbb 个时钟，此芯片线路才认定比较器的输出是由 HIGH 转成 LOW；如果在 bbbbbb 个时钟内，Analog 比较器的输出又回复到 HIGH, 此芯片线路认为甚么都没发生，视同比较器的输出一直维持在 HIGH。

若是设定成 000000，代表没有 Level-Change Control（LCC）.





24.1 比较器中断方式程序举例(C 及汇编)

1.C 语言程序

```
/*----STC15W4K60S4 系列比较器中断方式举例----*/
//本示例在 Keil 开发环境下请选择 Intel 的 8052 芯片型号进行编译
//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"
#include "intrins.h"
sfr         CMPCR1 = 0xE6;          //比较器控制寄存器 1

#define     CMPEN     0x80          //CMPCR1.7 : 比较器模块使能位
#define     CMPIF     0x40          //CMPCR1.6 : 比较器中断标志位
#define     PIE       0x20          //CMPCR1.5 : 比较器上升沿中断使能位
#define     NIE       0x10          //CMPCR1.4 : 比较器下降沿中断使能位
#define     PIS       0x08          //CMPCR1.3 : 比较器正极选择位
#define     NIS       0x04          //CMPCR1.2 : 比较器负极选择位
#define     CMPOE     0x02          //CMPCR1.1 : 比较结果输出控制位
#define     CMPRES    0x01          //CMPCR1.0 : 比较器比较结果标志位

sfr         CMPCR2 = 0xE7;          //比较器控制寄存器 2

#define     INVCMPO   0x80          //CMPCR2.7 : 比较结果反向输出控制位
#define     DISFLT    0x40          //CMPCR2.6 : 比较器输出端虑波使能控制位
#define     LCDTY     0x3F          //CMPCR2.[5:0] : 比较器输出的区抖时间控制

sbit        LED =   P1^1;           //测试脚

void cmp_isr() interrupt 21 using 1     //比较器中断向量入口
{
    CMPCR1 &= ~CMPIF;                   //清除完成标志
    LED = !!(CMPCR1 & CMPRES);          //将比较器结果 CMPRES 输出到测试口显示
}

void main()
{
    CMPCR1 = 0;                         //初始化比较器控制寄存器 1
    CMPCR2 = 0;                         //初始化比较器控制寄存器 2

    CMPCR1 &= ~PIS;                     //选择外部管脚 P5.5（CMP+）为比较器的正极输入源
//  CMPCR1 |= PIS;                      //选择 ADCIS[2:0]所选的 ADCIN 为比较器的正极输入源

    CMPCR1 &= ~NIS;                     //选择内部 BandGap 电压 BGV 为比较器的负极输入源
//  CMPCR1 |= NIS;                      //选择外部管脚 P5.4（CMP-）为比较器的负极输入源
```

```
    CMPCR1 &= ~CMPOE;               //禁用比较器的比较结果输出
//  CMPCR1 |= CMPOE;                //使能比较器的比较结果输出到 P1.2

    CMPCR2 &= ~INVCMPO;             //比较器的比较结果正常输出到 P1.2
//  CMPCR2 |= INVCMPO;              //比较器的比较结果取反后输出到 P1.2

    CMPCR2 &= ~DISFLT;              //不禁用(使能)比较器输出端的 0.1uS 虑波电路
//  CMPCR2 |= DISFLT;               //禁用比较器输出端的 0.1uS 虑波电路

    CMPCR2 &= ~LCDTY;               //比较器结果不去抖动,直接输出
//  CMPCR2 |= (DISFLT & 0x10);      //比较器结果在经过 16 个时钟后再输出

    CMPCR1 |= PIE;                  //使能比较器的上升沿中断
//  CMPCR1 |= NIE;                  //使能比较器的下降沿中断

    CMPCR1 |= CMPEN;                //使能比较器

    EA = 1;
    while (1);
}
```

2.汇编程序

```
/*----STC15W4K60S4 系列 比较器中断方式举例----*/
//本示例在 Keil 开发环境下请选择 Intel 的 8052 芯片型号进行编译
//假定测试芯片的工作频率为 18.432MHz

CMPCR1  DATA  0E6H        ;比较器控制寄存器 1
CMPEN   EQU   080H        ;CMPCR1.7 : 比较器模块使能位
CMPIF   EQU   040H        ;CMPCR1.6 : 比较器中断标志位
PIE     EQU   020H        ;CMPCR1.5 : 比较器上升沿中断使能位
NIE     EQU   010H        ;CMPCR1.4 : 比较器下降沿中断使能位
PIS     EQU   008H        ;CMPCR1.3 : 比较器正极选择位
NIS     EQU   004H        ;CMPCR1.2 : 比较器负极选择位
CMPOE   EQU   002H        ;CMPCR1.1 : 比较结果输出控制位
CMPRES  EQU   001H        ;CMPCR1.0 : 比较器比较结果标志位

CMPCR2  DATA  0E7H        ;比较器控制寄存器 2
INVCMPO EQU   080H        ;CMPCR2.7 : 比较结果反向输出控制位
DISFLT  EQU   040H        ;CMPCR2.6 : 比较器输出端虑波使能控制位
LCDTY   EQU   03FH        ;CMPCR2.[5:0] : 比较器输出的区抖时间控制

LED     BIT   P1.1        ;测试脚
;---------------------------------------
    ORG   0000H
    LJMP  START
```

```
    ORG     00ABH
    LJMP    CMP_ISR                 ;比较器中断向量入口
;---------------------------------------
    ORG     0100H
START:
    MOV     CMPCR1,     #0          ;初始化比较器控制寄存器
    MOV     CMPCR2,     #0          ;初始化比较器控制寄存器

    ANL     CMPCR1, #NOT PIS        ;选择外部管脚 P5.5（CMP+）为比较器的正极输入源
//  ORL     CMPCR1, #PIS            ;选择 ADCIS[2:0]所选的 ADCIN 为比较器的正极输入源

    ANL     CMPCR1, #NOT NIS        ;选择内部 BandGap 电压 BGV 为比较器的负极输入源
//  ORL     CMPCR1, #NIS            ;选择外部管脚 P5.4（CMP-）为比较器的负极输入源

    ANL     CMPCR1, #NOT CMPOE      ;禁用比较器的比较结果输出
//  ORL     CMPCR1, #CMPOE          ;使能比较器的比较结果输出到 P1.2

    ANL     CMPCR2, #NOT INVCMPO        ;比较器的比较结果正常输出到 P1.2
//  ORL     CMPCR2, #INVCMPO            ;比较器的比较结果取反后输出到 P1.2

    ANL     CMPCR2, #NOT DISFLT         ;不禁用(使能)比较器输出端的 0.1uS 虑波电路
//  ORL     CMPCR2, #DISFLT             ;禁用比较器输出端的 0.1uS 虑波电路

    ANL     CMPCR2, #NOT LCDTY          ;比较器结果不去抖动,直接输出
//  ORL     CMPCR2, #(DISFLT AND 0x10)  ;比较器结果在经过 16 个时钟后再输出

    ORL     CMPCR1, #PIE                ;使能比较器的上升沿中断
//  ORL     CMPCR1, #NIE                ;使能比较器的下降沿中断

    ORL     CMPCR1, #CMPEN              ;使能比较器
    SETB    EA

    SJMP    $
;---------------------------------------
CMP_ISR:
    PUSH    PSW
    PUSH    ACC

    ANL     CMPCR1, #NOT CMPIF          ;清除完成标志

    MOV     A, CMPCR1
    MOV     C, ACC.0                    ;将比较器结果 CMPRES 输出到测试口显示
    MOV     LED, C
    POP     ACC
    POP     PSW
```

```
    RETI
;----------------------------------------
    END
```

24.2 比较器查询方式程序举例(C 及汇编)

1.C 语言程序

```
/*---STC15W4K60S4 系列比较器查询方式举例-----*/
//本示例在 Keil 开发环境下请选择 Intel 的 8052 芯片型号进行编译
//假定测试芯片的工作频率为 18.432MHZ

#include "reg51.h"
#include "intrins.h"

sfr         CMPCR1= 0xE6;           //比较器控制寄存器 1
#define     CMPEN       0x80        //CMPCR1.7 : 比较器模块使能位
#define     CMPIF       0x40        //CMPCR1.6 : 比较器中断标志位
#define     PIE         0x20        //CMPCR1.5 : 比较器上升沿中断使能位
#define     NIE         0x10        //CMPCR1.4 : 比较器下降沿中断使能位
#define     PIS         0x08        //CMPCR1.3 : 比较器正极选择位
#define     NIS         0x04        //CMPCR1.2 : 比较器负极选择位
#define     CMPOE       0x02        //CMPCR1.1 : 比较结果输出控制位
#define     CMPRES      0x01        //CMPCR1.0 : 比较器比较结果标志位

sfr         CMPCR2 = 0xE7;          //比较器控制寄存器 2
#define     INVCMPO     0x80        //CMPCR2.7 : 比较结果反向输出控制位
#define     DISFLT      0x40        //CMPCR2.6 : 比较器输出端 0.1us 虑波控制位
#define     LCDTY       0x3F        //CMPCR2.[5:0] : 比较器输出的区抖时间控制

sbit        LED = P1^1;             //测试脚

void main()
{
    CMPCR1 = 0;                     //初始化比较器控制寄存器 1
    CMPCR2 = 0;                     //初始化比较器控制寄存器 2

    CMPCR1 &= ~PIS;                 //选择外部管脚 P5.5（CMP+）为比较器的正极输入源
//  CMPCR1 |= PIS;                  //选择 ADCIS[2:0]所选的 ADCIN 为比较器的正极输入源

    CMPCR1 &= ~NIS;                 //选择内部 BandGap 电压 BGV 为比较器的负极输入源
//  CMPCR1 |= NIS;                  //选择外部管脚 P5.4（CMP-）为比较器的负极输入源
```

```
    CMPCR1 &= ~CMPOE;               //禁用比较器的比较结果输出
//  CMPCR1 |= CMPOE;                //使能比较器的比较结果输出到 P1.2

    CMPCR2 &= ~INVCMPO;             //比较器的比较结果正常输出到 P1.2
//  CMPCR2 |= INVCMPO;              //比较器的比较结果取反后输出到 P1.2

    CMPCR2 &= ~DISFLT;              //不禁用(使能)比较器输出端的 0.1uS 虑波电路
//  CMPCR2 |= DISFLT;               //禁用比较器输出端的 0.1uS 虑波电路

    CMPCR2 &= ~LCDTY;               //比较器结果不去抖动,直接输出
//  CMPCR2 |= (DISFLT & 0x10);      //比较器结果在经过 16 个时钟后再输出

    CMPCR1 |= CMPEN;                //使能比较器
    while (!(CMPCR1 & CMPIF));      //查询比较完成标志
    CMPCR1 &= ~CMPIF;               //清除完成标志
    LED = !!(CMPCR1 & CMPRES);      //将比较器结果 CMPRES 输出到测试口显示

    while (1);
}
```

2.汇编程序

```
/*----STC15W4K60S4 系列比较器查询方式举例----*/
//本示例在 Keil 开发环境下请选择 Intel 的 8052 芯片型号进行编译
//假定测试芯片的工作频率为 18.432MHz

CMPCR1  DATA  0E6H    ;比较器控制寄存器 1
CMPEN   EQU   080H    ;CMPCR1.7 : 比较器模块使能位
CMPIF   EQU   040H    ;CMPCR1.6 : 比较器中断标志位
PIE     EQU   020H    ;CMPCR1.5 : 比较器上升沿中断使能位
NIE     EQU   010H    ;CMPCR1.4 : 比较器下降沿中断使能位
PIS     EQU   008H    ;CMPCR1.3 : 比较器正极选择位
NIS     EQU   004H    ;CMPCR1.2 : 比较器负极选择位
CMPOE   EQU   002H    ;CMPCR1.1 : 比较结果输出控制位
CMPRES  EQU   001H    ;CMPCR1.0 : 比较器比较结果标志位

CMPCR2  DATA  0E7H    ;比较器控制寄存器 2
INVCMPO EQU   080H    ;CMPCR2.7 : 比较结果反向输出控制位
DISFLT  EQU   040H    ;CMPCR2.6 : 比较器输出端虑波使能控制位
LCDTY   EQU   03FH    ;CMPCR2.[5:0] : 比较器输出的区抖时间控制

LED     BIT   P1.1    ;测试脚
;---------------------------------------
    ORG   0000H
    LJMP  START
;---------------------------------------
```

```
    ORG     0100H
START:
    MOV     CMPCR1, #0                  ;初始化比较器控制寄存器
    MOV     CMPCR2, #0                  ;初始化比较器控制寄存器

    ANL     CMPCR1, #NOT PIS            ;选择外部管脚 P5.5（CMP+）为比较器的正极输入源
//  ORL     CMPCR1, #PIS                ;选择 ADCIS[2:0]所选的 ADCIN 为比较器的正极输入源

    ANL     CMPCR1, #NOT NIS            ;选择内部 BandGap 电压 BGV 为比较器的负极输入源
//  ORL     CMPCR1, #NIS                ;选择外部管脚 P5.4（CMP-）为比较器的负极输入源

    ANL     CMPCR1, #NOT CMPOE          ;禁用比较器的比较结果输出
//  ORL     CMPCR1, #CMPOE              ;使能比较器的比较结果输出到 P1.2

    ANL     CMPCR2, #NOT INVCMPO        ;比较器的比较结果正常输出到 P1.2
//  ORL     CMPCR2, #INVCMPO            ;比较器的比较结果取反后输出到 P1.2

    ANL     CMPCR2, #NOT DISFLT         ;不禁用(使能)比较器输出端的 0.1uS 虑波电路
//  ORL     CMPCR2, #DISFLT             ;禁用比较器输出端的 0.1uS 虑波电路

    ANL     CMPCR2, #NOT LCDTY          ;比较器结果不去抖动,直接输出
//  ORL     CMPCR2, #(DISFLT AND 0x10)  ;比较器结果在经过 16 个时钟后再输出

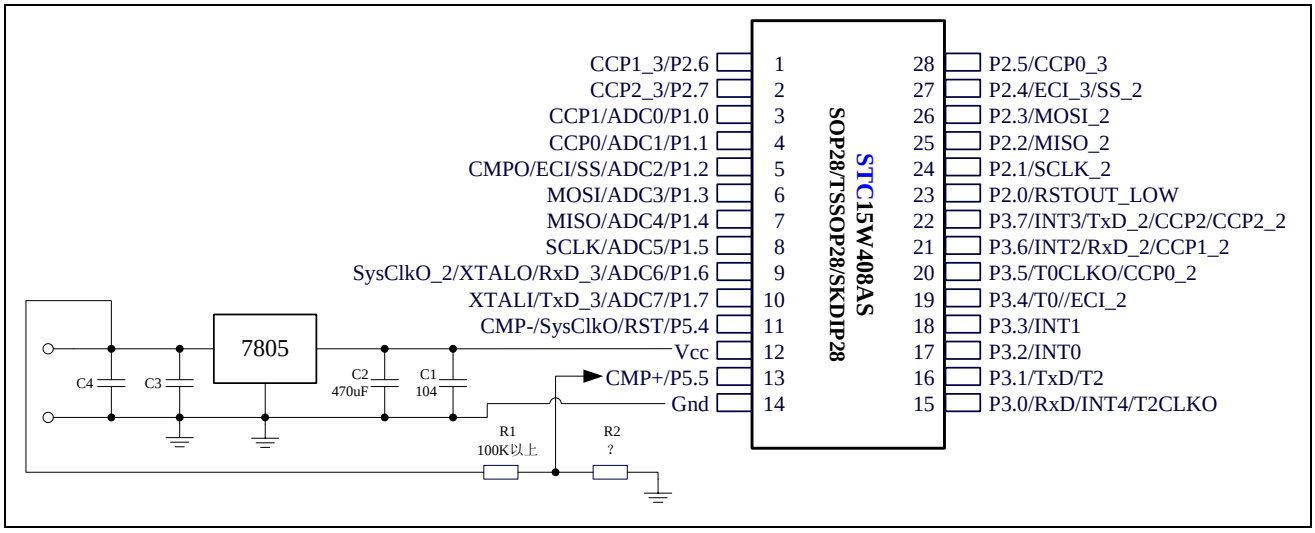
    ORL     CMPCR1, #CMPEN              ;使能比较器

WAIT:
    MOV     A, CMPCR1                   ;查询比较完成标志
    ANL     A, #CMPIF
    JZ      WAIT
    ANL     CMPCR1, #NOT CMPIF          ;清除完成标志
    MOV     A, CMPCR1
    MOV     C, ACC.0                    ;将比较器结果 CMPRES 输出到测试口显示
    MOV     LED, C

    SJMP    $

    END
```

24.3 比较器作外部掉电检测的参考电路



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 P5.5/CMP+的外部输入与内部 BandGap 参考电压（1.27V 附近）进行比较。

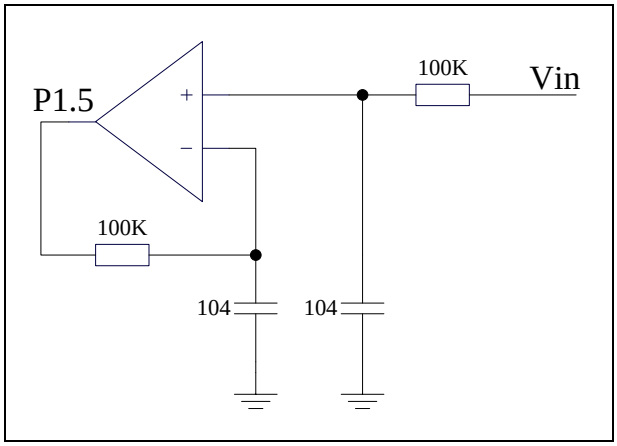
一般当交流电在 220V 时，稳压块 7805 前端的直流电压是 11V，但当交流电压降到 160V 时，稳压块 7805 前端的直流电压是 8.5V。当稳压块 7805 前端的直流电压低于或等于 8.5V 时，该前端输入的直流电压被电阻 R1 和 R2 分压到 CMP+端（比较器正极输入端），CMP+端输入电压低于内部 BandGap 参考电压(1.27V 附近),此时可产生比较器中断,这样在掉电检测时就有充足的时间将数据保存到 EEPROM 中。当稳压块 7805 前端的直流电压高于 8.5V 时，该前端输入的直流电压被电阻 R1 和 R2 分压到 CMP+端（比较器正极输入端），CMP+端输入电压高于内部 BandGap 参考电压（1.27V 附近），此时 CPU 可继续正常工作。

内部 BandGap 参考电压约在 1.27V 附近，具体数值要通过读取内部 BandGap 电压在内部 RAM 区或 ROM 区所占用的地址的值获得。

- 对于具有 128 字节 RAM 空间的单片机（如 STC15W104 系列单片机），其内部 BandGap 参考电压值在 RAM 区占用的地址为 06FH--070H，在 ROM 区占用的地址为程序空间最后第 8 字节和第 9 字节(如 STC15W104 型号单片机具有 4K 程序空间,则其内部 BandGap 参考电压值在 ROM 区占用的地址为 0FF7H--0FF8H)，用户只需通过读取 RAM 区 06FH--070H 地址的值或 ROM 区 0FF7H--0FF8H 地址的值即可获得 STC15W104 型号单片机的内部 BandGap 参考电压值（毫伏，高字节在前）。
- 对于具有 256 及其以上字节 RAM 空间的单片机（如 STC15W4K32S4 系列单片机），其内部 BandGap 参考电压值在 RAM 区占用的地址为 0EFH--0F0H，在 ROM 区占用的地址为程序空间最后第 8 字节和第 9 字节(如 STC15W4K32S4 型号单片机具有 32K 程序空间,则其内部 BandGap 参考电压值在 ROM 区占用的地址为 7FF7H--7FF8H)，用户只需通过读取 RAM 区 0EFH--0F0H 地址的值或 ROM 区 7FF7H--7FF8H 地址的值即可获得 STC15W4K32S4 型号单片机的内部 BandGap 参考电压值（毫伏，高字节在前）。

24.4 STC15W 系列比较器作 ADC 的程序举例（C 语言）

/*----STC15W4K60S4 系列比较器作 ADC 的程序举例----*/
//本示例在 Keil 开发环境下请选择 Intel 的 8052 芯片型号进行编译
//假定测试芯片的工作频率为 22.1184MHz
//使用 MCU 自带的比较器进行 ADC 转换,并经过模拟串口输出结果。
/**********************************
使用比较器做 ADC，原理图如下：
做 ADC 的原理是基于电荷平衡的计数式 ADC。
电乐从 Vin 输入，通过 100K+104 滤波，进入比较器的 P5.5 正输入端，经过比较器的比较，将结果输出到 P1.5 再通过 100K+104 滤波后送比较器 P5.4 负输入端，跟输入电压平衡。
设置两个变量：计数周期(量程)adc_duty 和比较结果高电平的计数值 adc，adc 严格比例于输入电压。
ADC 的基准就是 P1.5 的高电平，如果高电平准确，比较器的放大倍数足够大，则 ADC 结果会很准确。
当比较结果为高电平，则 P1.5 输出 1，并且 adc+1。
当比较结果为低电平，则 P1.5 输出 0。
每一次比较都判断计数周期是否完成，完成则 adc 里的值就是 ADC 结果。
电荷平衡计数式 ADC 的性能类似数字万用表用的双积分 ADC，当计数周期为 20ms 的倍数时，具有很强的抗工频干扰能力，很好的线性和精度。
原理可以参考 ADD3501(3 1/2 位数字万用表)或 ADD3701(3 3/4 位数字万用表),也可以参考 AD7740 VFC 电路。
例：比较一次的时间间隔为 10us，量程为 10000，则做 1 次 ADC 的时间为 100ms。比较器的响应时间越短，则完成 ADC 就越快。
由于要求每次比较时间间隔都要相等，所以用 C 编程最好在定时器中断里进行，定时器设置为自动重装，高优先级中断，其它中断均低优先级。
用汇编的话，保证比较输出电平处理的时间要相等。



**********/

```
#include "reg51.h"
#include "intrins.h"

typedef     unsigned char     u8;
typedef     unsigned int      u16;
typedef     unsigned long     u32;

#define     MAIN_Fosc         22118400L     //定义主时钟
```

```
sfr         P1M1 = 0x91;      //P1M1.n,P1M0.n = 00--->Standard, 01--->push-pull,实际上 1T 的都一样
sfr         P1M0 = 0x92;      //                =10--->pure input,11--->open drain
sfr         AUXR = 0x8E;

sfr         CMPCR1 = 0xE6;          //比较器控制寄存器 1
#define     CMPEN       0x80        //CMPCR1.7 : 比较器模块使能位
#define     CMPIF       0x40        //CMPCR1.6 : 比较器中断标志位
#define     PIE         0x20        //CMPCR1.5 : 比较器上升沿中断使能位
#define     NIE         0x10        //CMPCR1.4 : 比较器下降沿中断使能位
#define     PIS         0x08        //CMPCR1.3 : 比较器正极选择位
#define     NIS         0x04        //CMPCR1.2 : 比较器负极选择位
#define     CMPOE       0x02        //CMPCR1.1 : 比较结果输出控制位
#define     CMPRES      0x01        //CMPCR1.0 : 比较器比较结果标志位

sfr         CMPCR2 = 0xE7;          //比较器控制寄存器 2
#define     INVCMPO     0x80        //CMPCR2.7 : 比较结果反向输出控制位
#define     DISFLT      0x40        //CMPCR2.6 : 比较器输出端 0.1us 虑波控制位
#define     LCDTY       0x3F        //CMPCR2.[5:0] : 比较器输出的区抖时间控制

sbit        LED     =   P1^1;       //测试脚

#define     ADC_SCALE       50000   //ADC 满量程, 根据需要设置
sbit        P_ADC = P1^5;           //P1.2  比较器输出端
unsigned    int   adc;              //ADC 中间值, 用户层不可见
unsigned    int   adc_duty;         //ADC 计数周期, 用户层不可见
unsigned    int   adc_value;        //ADC 值, 用户层使用
bit         adc_ok;                 //ADC 结束标志, 为 1 则 adc_value 的值可用.
                                    //此标志给用户层查询,并且清 0
sbit        P_TXD   =   P3^1;       //定义模拟串口发送端,可以是任意 IO

void TxSend(u8 dat);
void PrintString(unsigned char code *puts);

void main()
{
    u8   i;
    u8   tmp[5];
    CMPCR1 = 0;                     //初始化比较器控制寄存器 1
    CMPCR2 = 0;                     //初始化比较器控制寄存器 2

    CMPCR1 &= ~PIS;                 //选择外部管脚 P5.5（CMP+）为比较器的正极输入源
//  CMPCR1 |= PIS;                  //选择 ADCIS[2:0]所选的 ADCIN 为比较器的正极输入源

//  CMPCR1 &= ~NIS;                 //选择内部 BandGap 电压 BGV 为比较器的负极输入源
    CMPCR1 |= NIS;                  //选择外部管脚 P5.4（CMP-）为比较器的负极输入源
```

```
    CMPCR1 &= ~CMPOE;                   //禁用比较器的比较结果输出
//  CMPCR1 |= CMPOE;                    //使能比较器的比较结果输出到 P1.2

    CMPCR2 &= ~INVCMPO;                 //比较器的比较结果正常输出到 P1.2
//  CMPCR2 |= INVCMPO;                  //比较器的比较结果取反后输出到 P1.2

    CMPCR2 &= ~DISFLT;                  //使能比较器输出端的 0.1uS 虑波电路
//  CMPCR2 |= DISFLT;                   //禁用比较器输出端的 0.1uS 虑波电路

    CMPCR2 &= ~LCDTY;                   //比较器结果不去抖动,直接输出
//  CMPCR2 |= (DISFLT & 0x10);          //比较器结果在经过 16 个时钟后再输出

    CMPCR1 |= CMPEN;                    //使能比较器
//  while (!(CMPCR1 & CMPIF));          //查询比较完成标志
//  CMPCR1 &= ~CMPIF;                   //清除完成标志
//  LED = !!(CMPCR1 & CMPRES);          //将比较器结果 CMPRES 输出到测试口显示

    ET0 = 1;                            //允许中断
    PT0 = 1;                            //高优先级中断
    TMOD &= ~0x03;                      //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
    AUXR |= 0x80;                       //1T

    TH0 = (u8)((-(MAIN_Fosc*10)/1000000) >> 8);     //10us
    TL0 = (u8)(-(MAIN_Fosc*10)/1000000);

    P1M1 &= ~(1<<5);                    //P1.5 设置为 push pull 输出
    P1M0 |= (1<<5);

    adc_duty = ADC_SCALE;               //周期计数赋初值

    adc = 0;
    TR0 = 1;                            //开始运行
    EA = 1;

    PrintString("\r\n 使用比较器做 ADC 例子\r\n");

    while (1)
    {
        if(adc_ok)
        {
            adc_ok = 0;                             //清除 ADC 已结束标志
            PrintString("ADC = ");                  //转十进制
            tmp[0] = adc_value / 10000 + '0';
            tmp[1] = adc_value % 10000 / 1000 + '0';
            tmp[2] = adc_value % 1000 / 100 + '0';
```

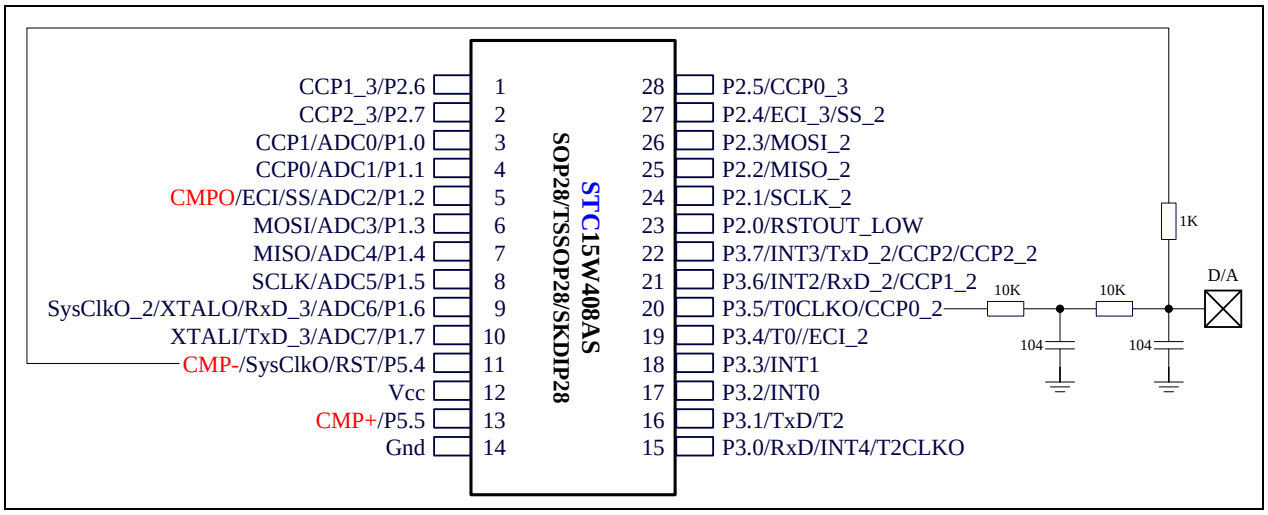
```
            tmp[3] = adc_value % 100 / 10 + '0';
            tmp[4] = adc_value % 10 + '0';
            for(i=0; i<4; i++)                      //消无效 0
            {
                if(tmp[i] != '0') break;
                tmp[i] = ' ';
            }
            for(i=0; i<5; i++) TxSend(tmp[i]);      //发串口
            PrintString("\r\n");
        }
    }
}
/********************** Timer0 中断函数************************/
void timer0_int (void) interrupt 1
{
    if(CMPCR1 & CMPRES)                             //比较器输出高电平
    {
        P_ADC = 1;                                  //P_ADC 输出高电平, 给负输入端做反馈.
        adc ++;                                     //ADC 计数+1
    }
    else P_ADC = 0;                                 //P_ADC 输出高电平, 给负输入端做反馈.

    if(--adc_duty == 0)                             //ADC 周期-1, 到 0 则 ADC 结束
    {
        adc_duty = ADC_SCALE;                       //周期计数赋初值
        adc_value = adc;                            //保存 ADC 值
        adc = 0;                                    //清除 ADC 值
        adc_ok = 1;                                 //标志 ADC 已结束
    }
}
//========================================================================
// 函数: void BitTime(void)
// 描述: 位时间函数。
// 参数: none.
// 返回: none.
//========================================================================
void BitTime(void)
{
    u16 i;
    i = ((MAIN_Fosc / 100) * 104) / 130000L - 1;    //根据主时钟来计算位时间
    while(--i);
}
//========================================================================
// 函数: void TxSend(uchar dat)
// 描述: 模拟串口发送一个字节。9600，N，8，1
// 参数: dat: 要发送的数据字节.
```

```
// 返回: none.
//==========================================================================
void TxSend(u8 dat)
{
    u8    i;
    EA = 0;
    P_TXD = 0;
    BitTime();
    for(i=0; i<8; i++)
    {
        if(dat & 1)     P_TXD = 1;
        else            P_TXD = 0;
        dat >>= 1;
        BitTime();
    }
    P_TXD = 1;
    EA = 1;
    BitTime();
    BitTime();
}
//==========================================================================
// 函数: void PrintString(unsigned char code *puts)
// 描述: 模拟串口发送一串字符串。9600，N，8，1
// 参数: *puts: 要发送的字符指针.
// 返回: none.
//==========================================================================
void PrintString(unsigned char code *puts)
{
    for (; *puts != 0; puts++) TxSend(*puts);
}
```

24.5 在比较器负端产生不同的电压由比较器正端进行比较

对于无 ADC 功能的单片机，如要进行电流检测或检测外部电池电压等，可以利用 CCP/PWM 内部产生一个电压输入到比较器负端（CPM-），然后由比较器的正端（CMP+）将其与外部电压进行比较，从而达到检测外部电压的目的。具体实现的参考电路图如下：



PWM 功能可以利用 T0/T1 软件模拟 10 位/12 位/16 位 PWM 来实现，具体实现方法请参照“用 T0 软硬结合模拟 10 位/16 位 PWM 输出的程序”这一节；还可以利用 CCP/PCA 软硬结合实现 9 ~ 16 位 PWM 来实现，具体实现方法参照“用 CCP/PCA 软硬结合实现 9 ~ 16 位 PWM 输出的程序”。

24.6 现供货的 STC15W204S 系列 A 版本比较器下降沿中断不响应

----将在 STC15W204S 系列 B 版本中修正

STC15W204S 系列的 A 版本芯片正大批量现货供应中，当仅允许该版本的比较器下降沿中断时，该比较器的下降沿中断暂不能使用。但是，该版本的比较器下降沿中断不是绝对不能使用，用户可以通过以下两种办法解决这一问题：

一、STC15W204S 系列 A 版本的比较器上升沿中断是可正常使用的，且当用户将其比较器上升沿中断和下降沿中断都允许后，该比较器上升沿中断和下降沿中断都可以正常使用。由于比较器比较结果标志位 CMPRES（CMPCR1.0）是正确的，因此在比较器中断服务程序中查询比较器比较结果标志位 CMPRES（CMPCR1.0）的值可判断单片机进入的是比较器上升沿中断还是比较器下降沿中断。

- 如果 CMPRES/CMPCR1.0 = 1，即 CMP+的电平高于 CMP-的电平（或内部 BandGap 参考电压的电平），则表示单片机进入的是比较器上升沿中断；
- 反之，如果 CMPRES（CMPCR1.0）= 0，即 CMP+的电平低于 CMP-的电平（或内部 BandGap 参考电压的电平），则表示单片机进入的是比较器下降沿中断，此时比较器下降沿中断是可正常使用的。

这是解决办法之一。

二、STC15W204S 系列 A 版本的比较器比较结果标志位（CMPRES）是正确的，因此用户还可用软件查询方式解决该这一问题。

对于上述问题，敬请广大客户留意！对于给各位客户带来的不便，我们深表歉意，请各位客户谅解！我们将在 STC15W204S 系列的下一版本，即 STC15W204S 系列 B 版本中修正这一 BUG.

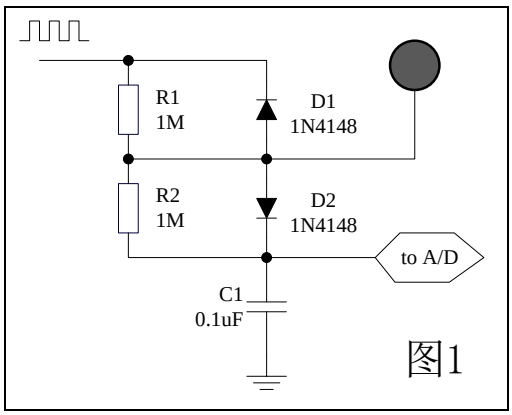
25 使用 STC15 系列单片机的 ADC 做电容感应触摸按键

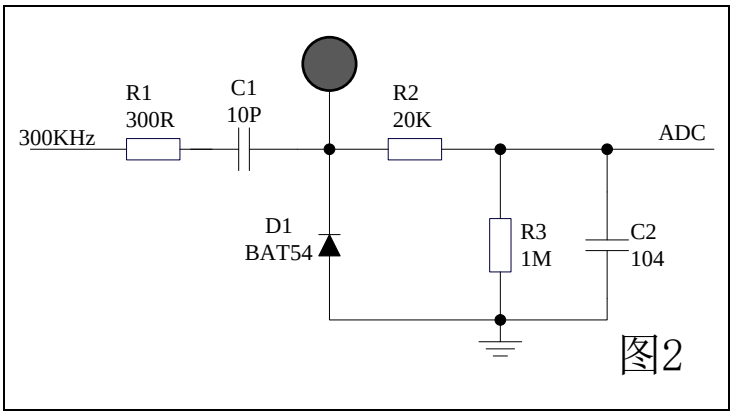
按键是电路最常用的零件之一，是人机界面重要的输入方式，我们最熟悉的是机械式按键，但是机械按键有一个缺点（特别是便宜的按键），触点有寿命，很容易出现接触不良而失效。而非接触的按键则没有机械触点，寿命长，使用方便。

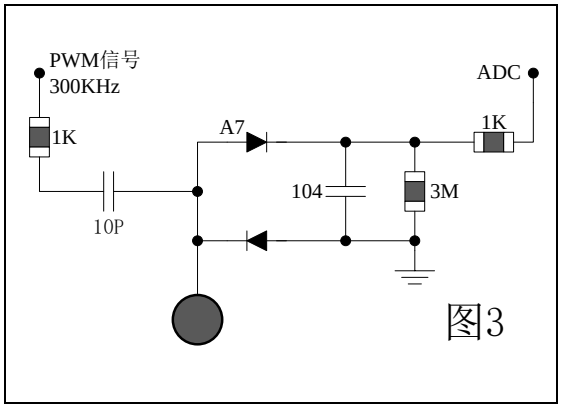
非接触的按键有多种方案，而电容感应按键则是低成本的方案，多年前一般是使用专门的 IC 来实现，随着 MCU 功能的加强，以及广大用户的实践经验，直接使用 MCU 来做电容感应按键的技术已经成熟，其中最典型最可靠的是使用 ADC 做的方案。

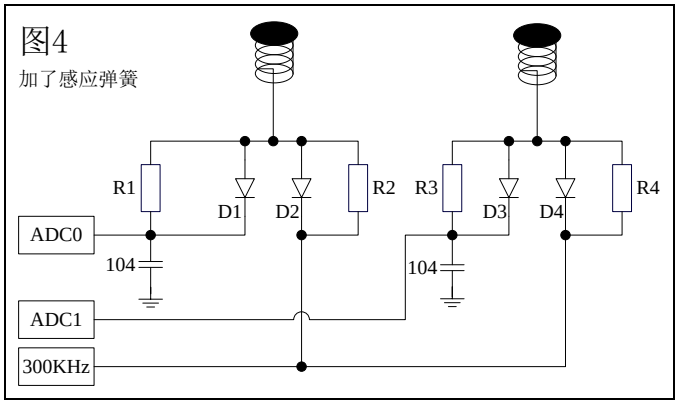
本文档详述使用 STC 带 ADC 的系列 MCU 做的方案，可以使用任何带 ADC 功能的 MCU 来实现。

下面前 3 个图是用得最多的方式，原理都一样，本文使用第 2 个图。

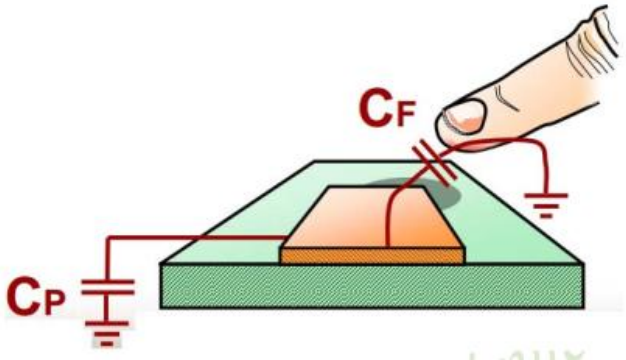




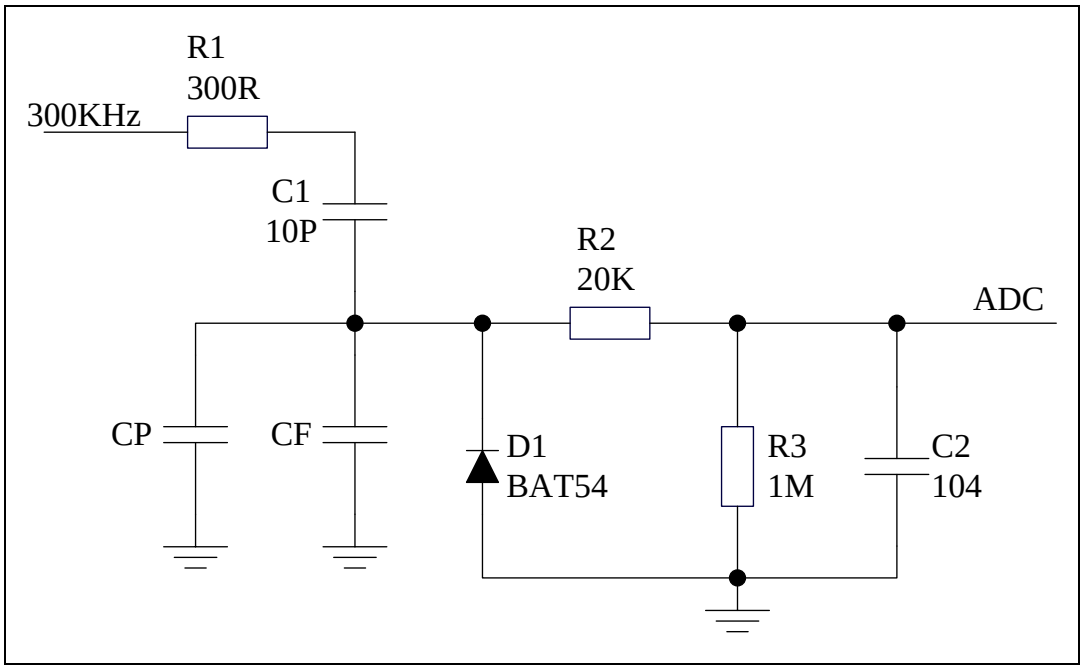




一般实际应用时，都使用图 4 所示的感应弹簧来加大手指按下的面积。感应弹簧等效一块对地的金属板，对地有一个电容 CP，而手指按下后，则再并联一个对地的电容 CF，如下图所示。



下面为电路图的说明，CP 为金属板和分布电容，CF 为手指电容，并联在一起与 C1 对输入的 300KHz 方波进行分压，经过 D1 整流，R2、C2 滤波后送 ADC，当手指压上去后，送去 ADC 的电压降低，程序就可以检测出按键动作。



具体的处理请参考下文的 C 语言或汇编程序：

1、C 语言程序

```
/*----STC15W4K60S4 系列 ADC 做电容触摸按键程序举例----*/
/************* 功能说明 ***************
测试使用 STC15W408AS 的 ADC 做的电容感应触摸键.
假定测试芯片的工作频率为 24MHz
*****************************************/
#include <reg51.h>
#include <intrins.h>

#define     MAIN_Fosc       24000000UL          //定义主时钟

typedef     unsigned char   u8;
typedef     unsigned int    u16;
typedef     unsigned long   u32;

#define     Timer0_Reload   (65536UL -(MAIN_Fosc / 600000))     //Timer 0 重装值，对应 300KHz

sfr         P1ASF = 0x9D;                       //只写，模拟输入选择
sfr         ADC_CONTR = 0xBC;                   //带 AD 系列
sfr         ADC_RES = 0xBD;                     //带 AD 系列
sfr         ADC_RESL = 0xBE;                    //带 AD 系列
sfr         AUXR = 0x8E;
sfr         AUXR2 = 0x8F;
```

```
/************* 本地常量声明 **************/
#define     TOUCH_CHANNEL   8               //ADC 通道数
#define     ADC_90T     (3<<5)              //ADC 时间 90T
#define     ADC_180T    (2<<5)              //ADC 时间 180T
#define     ADC_360T    (1<<5)              //ADC 时间 360T
#define     ADC_540     T0                  //ADC 时间 540T
#define     ADC_FLAG    (1<<4)              //软件清 0
#define     ADC_START   (1<<3)              //自动清 0
**************/

/************* 本地变量声明 sbit P_LED7 = P2^7;
sbit        P_LED6 = P2^6;
sbit        P_LED5 = P2^5;
sbit        P_LED4 = P2^4;
sbit        P_LED3 = P2^3;
sbit        P_LED2 = P2^2;
sbit        P_LED1 = P2^1;
sbit        P_LED0 = P2^0;

u16         idata adc[TOUCH_CHANNEL];       //当前 ADC 值
u16         idata adc_prev[TOUCH_CHANNEL];  //上一个 ADC 值
u16         idata TouchZero[TOUCH_CHANNEL]; //0 点 ADC 值

u8          idata TouchZeroCnt[TOUCH_CHANNEL];  //0 点自动跟踪计数
u8          cnt_250ms;
/*************  本地函数声明  **************/
void        delay_ms(u8 ms);
void        ADC_init(void);
u16         Get_ADC10bitResult(u8 channel);
void        AutoZero(void);
u8          check_adc(u8 index);
void        ShowLED(void);
/********************** 主函数 **************************/
void main(void)
{
    u8  i;
    delay_ms(50);

    ET0 = 0;                                //初始化 Timer0 输出一个 300KHZ 时钟
    TR0 = 0;
    AUXR |= 0x80;                           //Timer0 set as 1T mode
    AUXR2 |= 0x01;                          //允许输出时钟
    TMOD = 0;                               //Timer0 set as Timer, 16 bits Auto Reload.
    TH0 = (u8)(Timer0_Reload >> 8);
    TL0 = (u8)Timer0_Reload;
    TR0 = 1;
```

```
    ADC_init();                         //ADC 初始化
    delay_ms(50);                       //延时 50ms

    for(i=0; i<TOUCH_CHANNEL; i++)      //初始化 0 点和上一个值和 0 点自动跟踪计数
    {
        adc_prev[i] = 1023;
        TouchZero[i] = 1023;
        TouchZeroCnt[i] = 0;
    }

    cnt_250ms = 0;

    while (1)
    {
        delay_ms(50);                   //每隔 50ms 处理一次按键
        ShowLED();
        if(++cnt_250ms >= 5)
        {
            cnt_250ms = 0;
            AutoZero();                 //每隔 250ms 处理一次 0 点自动跟踪
        }
    }
}
/********************************************/
//==========================================================================
// 函数: void delay_ms(unsigned char 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.
// 备注:
//==========================================================================
void delay_ms(u8 ms)
{
    unsigned int i;
    do {
            i = MAIN_Fosc / 13000;
            while(--i) ;
    }while(--ms);
}
/************* ADC 初始化函数 *****************/
void ADC_init(void)
{
    P1ASF = 0xff;                       //8 路 ADC
    ADC_CONTR = 0x80;                   //允许 ADC
}
//==========================================================================
```

```
// 函数: u16 Get_ADC10bitResult(u8 channel)
// 描述: 查询法读一次 ADC 结果.
// 参数: channel: 选择要转换的 ADC.
// 返回: 10 位 ADC 结果.
//========================================================================
u16 Get_ADC10bitResult(u8 channel)                          //channel = 0~7
{
    ADC_RES = 0;
    ADC_RESL = 0;
    ADC_CONTR = 0x80 | ADC_90T | ADC_START | channel;       //触发 ADC
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    while((ADC_CONTR & ADC_FLAG) == 0)      ;               //等待 ADC 转换结束
    ADC_CONTR = 0x80;                                       //清除标志
    return(((u16)ADC_RES << 2) | ((u16)ADC_RESL & 3));      //返回 ADC 结果
}
/********************** 自动 0 点跟踪函数 **************************/
void AutoZero(void)                 //250ms 调用一次 这是使用相邻 2 个采样的差的绝对值之和来检测。
{
    u8      i;
    u16     j, k;
    for(i=0; i<TOUCH_CHANNEL; i++)              //处理 8 个通道
    {
        j = adc[i];
        k = j - adc_prev[i];                    //减前一个读数
        F0 = 0;                                 //按下
        if(k & 0x8000)      F0 = 1,    k = 0 - k;       //释放, 求出两次采样的差值
        if(k >= 20)                             //变化比较大
        {
            TouchZeroCnt[i] = 0;                //如果变化比较大，则清 0 计数器
            if(F0)      TouchZero[i] = j;       //如果是释放，并且变化比较大，则直接替代
        }
        else                                    //变化比较小，则蠕动，自动 0 点跟踪
        {
            if(++TouchZeroCnt[i] >= 20)         //连续检测到小变化 20 次/4 = 5 秒.
            {
                TouchZeroCnt[i] = 0;
                TouchZero[i] = adc_prev[i];     //变化缓慢的值作为 0 点
            }
        }
        adc_prev[i] = j;                        //保存这一次的采样值
    }
}
/********************** 获取触摸信息函数 50ms 调用 1 次 **************************/
```

```
u8 check_adc(u8 index)                                  //判断键按下或释放, 有回差控制
{
    u16     delta;
    adc[index] = 1023 - Get_ADC10bitResult(index);      //获取 ADC 值, 转成按下键, ADC 值增加
    if(adc[index] < TouchZero[index])       return 0;   //比 0 点还小的值，则认为是键释放
    delta = adc[index] - TouchZero[index];
    if(delta >= 40)     return 1;                       //键按下
    if(delta <= 20)     return 0;                       //键释放
    return    2;                                        //保持原状态
}
/********************** 键处理 50ms 调用 1 次 ************************/
void ShowLED(void)
{
    u8   i;
    i = check_adc(0);
    if(i == 0)      P_LED0 = 1;                         //指示灯灭
    if(i == 1)      P_LED0 = 0;                         //指示灯亮

    i = check_adc(1);
    if(i == 0)      P_LED1 = 1;                         //指示灯灭
    if(i == 1)      P_LED1 = 0;                         //指示灯亮

    i = check_adc(2);
    if(i == 0)      P_LED2 = 1;                         //指示灯灭
    if(i == 1)      P_LED2 = 0;                         //指示灯亮

    i = check_adc(3);
    if(i == 0)      P_LED3 = 1;                         //指示灯灭
    if(i == 1)      P_LED3 = 0;                         //指示灯亮

    i = check_adc(4);
    if(i == 0)      P_LED4 = 1;                         //指示灯灭
    if(i == 1)      P_LED4 = 0;                         //指示灯亮

    i = check_adc(5);
    if(i == 0)      P_LED5 = 1;                         //指示灯灭
    if(i == 1)      P_LED5 = 0;                         //指示灯亮

    i = check_adc(6);
    if(i == 0)      P_LED6 = 1;                         //指示灯灭
    if(i == 1)      P_LED6 = 0;                         //指示灯亮

    i = check_adc(7);
    if(i == 0)      P_LED7 = 1;                         //指示灯灭
    if(i == 1)      P_LED7 = 0;                         //指示灯亮
}
```

2、汇编程序

```
/*----STC15W4K60S4 系列 ADC 做电容触摸按键程序举例----*/
;测试使用 STC15W408AS 的 ADC 做的电容感应触摸键.
;假定测试芯片的工作频率为 24MHz
;/********************** 用户定义宏 **********************/
Fosc_KHZ        EQU     24000       ;定义主时钟 KHz
STACK_POIRTER   EQU     0D0H        ;堆栈开始地址
Timer0_Reload   EQU     (65536 - Fosc_KHZ/600)    ;Timer 0 重装值，对应 300KHz
;*******************************************************
P1ASF           DATA    0x9D;       //只写，模拟输入选择
ADC_CONTR       DATA    0xBC;       //带 AD 系列
ADC_RES         DATA    0xBD;       //带 AD 系列
ADC_RESL        DATA    0xBE;       //带 AD 系列
AUXR            DATA    0x8E;
AUXR2           DATA    0x8F;
;*************  本地变量声明    ***************/
/*************  本地常量声明    ***************/
TOUCH_CHANNEL   EQU     8           ;ADC 通道数
ADC_90T         EQU     (3 SHL 5)   ;ADC 时间 90T
ADC_180T        EQU     (2 SHL 5)   ;ADC 时间 180T
ADC_360T        EQU     (1 SHL 5)   ;ADC 时间 360T
ADC_540T        EQU     0           ;ADC 时间 540T
ADC_FLAG        EQU     (1 SHL 4)   ;软件清 0
ADC_START       EQU     (1 SHL 3)   ;自动清 0
/*************  本地变量声明    ***************/
P_LED7          BIT     P2.7
P_LED6          BIT     P2.6
P_LED5          BIT     P2.5
P_LED4          BIT     P2.4
P_LED3          BIT     P2.3
P_LED2          BIT     P2.2
P_LED1          BIT     P2.1
P_LED0          BIT     P2.0

adc             EQU     30H         ;当前 ADC 值 30H ~ 3FH, 两字节一个值
adc_prev        EQU     40H         ;上一个 ADC 值 40H ~ 4FH, 两字节一个值
TouchZero       EQU     50H         ; 0 点 ADC 值 50H ~ 5FH, 两字节一个值
TouchZeroCnt    EQU     60H         ; 0 点自动跟踪计数 60H ~ 67H

cnt_250ms       DATA    68H
;*******************************************************************
    ORG     00H                     ;reset
    LJMP    F_Main
    ORG     03H                     ;0 INT0 interrupt
    RETI
```

```
    LJMP    F_INT0_Interrupt

    ORG     0BH                 ;1 Timer0 interrupt
    LJMP    F_Timer0_Interrupt

    ORG     13H                 ;2 INT1 interrupt
    LJMP    F_INT1_Interrupt

    ORG     1BH                 ;3 Timer1 interrupt
    LJMP    F_Timer1_Interrupt

    ORG     23H                 ;4 UART1 interrupt
    LJMP    F_UART1_Interrupt

    ORG     2BH                 ;5 ADC and SPI interrupt
    LJMP    F_ADC_Interrupt

    ORG     33H                 ;6 Low Voltage Detect interrupt
    LJMP    F_LVD_Interrupt

    ORG     3BH                 ;7 PCA interrupt
    LJMP    F_PCA_Interrupt

    ORG     43H                 ;8 UART2 interrupt
    LJMP    F_UART2_Interrupt

    ORG     4BH                 ;9 SPI interrupt
    LJMP    F_SPI_Interrupt

    ORG     53H                 ;10 INT2 interrupt
    LJMP    F_INT2_Interrupt

    ORG     5BH                 ;11 INT3 interrupt
    LJMP    F_INT3_Interrupt

    ORG     63H                 ;12 Timer2 interrupt
    LJMP    F_Timer2_Interrupt

    ORG     83H                 ;16 INT4 interrupt
    LJMP    F_INT4_Interrupt
;********************  主程序  ***************************/
F_Main:
    MOV     R0, #1

L_ClearRamLoop:                 ;清除 RAM
    MOV     @R0, #0
```

```
    INC     R0
    MOV     A, R0
    CJNE    A, #0FFH, L_ClearRamLoop
    MOV     SP, #STACK_POIRTER
    MOV     PSW, #0
    USING   0                       ;选择第 0 组 R0~R7
;================ 用户初始化程序 ================================
    MOV     R7, #50
    LCALL   F_delay_ms
    CLR     ET0                     ; 初始化 Timer0 输出一个 300KHZ 时钟
    CLR     TR0
    ORL     AUXR, #080H             ; Timer0 set as 1T mode
    ORL     AUXR2, #01H             ; 允许输出时钟
    MOV     TMOD, #0                ; Timer0 set as Timer, 16 bits Auto Reload.
    MOV     TH0, #HIGH Timer0_Reload
    MOV     TL0, #LOW Timer0_Reload
    SETB    TR0
    LCALL   F_ADC_init
    MOV     R7, #50
    LCALL   F_delay_ms
    MOV     R0, #adc_prev           ; 初始化上一个 ADC 值

L_Init_Loop1:
    MOV     @R0, #03H
    INC     R0
    MOV     @R0, #0FFH
    INC     R0
    MOV     A, R0
    CJNE    A, #(adc_prev + TOUCH_CHANNEL * 2), L_Init_Loop1
    MOV     R0, #TouchZero          ; 初始化 0 点 ADC 值

L_Init_Loop2:
    MOV     @R0, #03H
    INC     R0
    MOV     @R0, #0FFH
    INC     R0
    MOV     A, R0
    CJNE    A, #(TouchZero + TOUCH_CHANNEL * 2), L_Init_Loop2
    MOV     R0, #TouchZeroCnt       ; 初始化自动跟踪计数值

L_Init_Loop3:
    MOV     @R0, #0
    INC     R0
    MOV     A, R0
    CJNE    A, #(TouchZeroCnt + TOUCH_CHANNEL), L_Init_Loop3
    MOV     cnt_250ms, #5
```

```
;=================== 主循环 ==================================
L_MainLoop:
    MOV     R7, #50                 ;延时 50ms
    LCALL   F_delay_ms
    LCALL   F_ShowLED               ; 处理一次触摸键值
    DJNZ    cnt_250ms, L_MainLoop
    MOV     cnt_250ms, #5           ;250ms 处理一次 0 点自动跟踪
    LCALL   F_AutoZero              ;自动跟踪零点
    SJMP    L_MainLoop
;=================== 主程序结束 ==================================
; /************* ADC 初始化函数 *****************/
F_ADC_init:
    MOV     P1ASF,#0FFH             ;8 路 ADC
    MOV     ADC_CONTR,#080H         ;允许 ADC
    RET
; END OF ADC_init
; //========================================================================
; //  函数: F_Get_ADC10bitResult
; //  描述:  查询法读一次 ADC 结果.
; //  参数: R7:  选择要转换的 ADC.
; //  返回: R6 R7 == 10 位 ADC 结果.
; //========================================================================
F_Get_ADC10bitResult:
    USING   0                       ;选择第 0 组 R0~R7
    MOV     ADC_RES, #0
    MOV     ADC_RESL,#0
    MOV     A, R7
    ORL     A, #0E8H ;(0x80 OR ADC_90T OR ADC_START)        ;触发 ADC
    MOV     ADC_CONTR, A
    NOP
    NOP
    NOP
    NOP

L_10bitADC_Loop1:
    MOV     A, ADC_CONTR
    JNB     ACC.4, L_10bitADC_Loop1 ;等待 ADC 转换结束
    MOV     ADC_CONTR, #080H        ;清除标志
    MOV     A, ADC_RES
    MOV     B, #04H
    MUL     AB
    MOV     R7, A
    MOV     R6, B
    MOV     A, ADC_RESL
    ANL     A, #03H
    ORL     A, R7
```

```
    MOV     R7, A
    RET
; END OF _Get_ADC10bitResult
; /*********************  自动 0 点跟踪函数  ***************************/
F_AutoZero:                     ;250ms 调用一次  这是使用相邻 2 个采样的差的绝对值之和来检测。
    USING    0                  ;选择第 0 组 R0~R7
    CLR     A
    MOV     R5,A

L_AutoZero_Loop:                ;[R6 R7] = adc[i], (j = adc[i])
    MOV     A, R5
    ADD     A, ACC
    ADD     A, #LOW (adc)
    MOV     R0, A
    MOV     A, @R0
    MOV     R6, A
    INC     R0
    MOV     A, @R0
    MOV     R7, A

; 计算差值  [R2 R3] = adc[i] - adc_prev[i], (k = j - adc_prev[i];)      //减前一个读数
    MOV     A, R5
    ADD     A, ACC
    ADD     A, #LOW (adc_prev+01H)
    MOV     R0, A
    CLR     C
    MOV     A, R7
    SUBB    A, @R0
    MOV     R3, A
    MOV     A, R6
    DEC     R0
    SUBB    A, @R0
    MOV     R2, A

; 求差值的绝对值  [R2 R3], if(k & 0x8000) F0 = 1, k = 0 - k;        //释放  求出两次采样的差值
    CLR     F0                      ;按下
    JNB     ACC.7, L_AutoZero_1
    SETB    F0
    CLR     C
    CLR     A
    SUBB    A, R3
    MOV     R3, A
    MOV     A, R3
    CLR     A
    SUBB    A, R2
    MOV     R2, A
```

```
L_AutoZero_1:
    CLR     C                       ;计算  [R2 R3] - #20, if(k >= 20)  //变化比较大
    MOV     A, R3
    SUBB    A, #2
    MOV     A, R2
    SUBB    A, #00H
    JC      L_AutoZero_2            ;[R2 R3] , 20,  转
    MOV     A,#LOW (TouchZeroCnt)   ;如果变化比较大，则清 0 计数器  TouchZeroCnt[i] = 0;
    ADD     A, R5
    MOV     R0, A
    MOV     @R0, #0
;   if(F0)  TouchZero[i] = j;       //如果是释放，并且变化比较大，则直接替代
    JNB     F0, L_AutoZero_3
    MOV     A, R5
    ADD     A, ACC
    ADD     A, #LOW (TouchZero)
    MOV     R0, A
    MOV     @R0, AR6
    INC     R0
    MOV     @R0, AR7
    SJMP    L_AutoZero_3

L_AutoZero_2:                       ;变化比较小，则蠕动，自动 0 点跟踪
;   if(++TouchZeroCnt[i] >= 20)     //连续检测到小变化 20 次/4 = 5 秒

    MOV     A, #LOW (TouchZeroCnt)
    ADD     A, R5
    MOV     R0, A
    INC     @R0
    MOV     A, @R0
    CLR     C
    SUBB    A, #20
    JC      L_AutoZero_3            ;if(TouchZeroCnt[i] < 20),  转
    MOV     @R0, #0                 ;TouchZeroCnt[i] = 0;
    MOV     A, R5                   ;TouchZero[i] = adc_prev[i]; //变化缓慢的值作为 0 点
    ADD     A, ACC
    ADD     A, #LOW (adc_prev)
    MOV     R0, A
    MOV     A, @R0
    MOV     R2, A
    INC     R0
    MOV     A, @R0
    MOV     R3, A
    MOV     A, R5
    ADD     A, ACC
    ADD     A, #LOW (TouchZero)
```

```
        MOV     R0, A
        MOV     @R0, AR2
        INC     R0
        MOV     @R0, AR3

L_AutoZero_3:                           ; 保存采样值       adc_prev[i] = j;
        MOV     A, R5
        ADD     A, ACC
        ADD     A, #LOW (adc_prev)
        MOV     R0, A
        MOV     @R0, AR6
        INC     R0
        MOV     @R0, AR7

        INC     R5
        MOV     A, R5
        XRL     A, #08H
        JZ      $ + 5H
        LJMP    L_AutoZero_Loop
        RET
; END OF AutoZero
; /********************** 获取触摸信息函数  50ms 调用 1 次  ************************/
F_check_adc:                            ;判断键按下或释放,有回差控制
        USING   0                       ;选择第 0 组 R0~R7
        MOV     R4, AR7
;       adc[index] = 1023 - Get_ADC10bitResult(index)       ;获取 ADC 值, 转成按下键, ADC 值增加
        LCALL   F_Get_ADC10bitResult    ;返回的 ADC 值在  [R6 R7]
        CLR     C
        MOV     A, #0FFH                ;1023 - [R6 R7]
        SUBB    A, R7
        MOV     R7, A
        MOV     A, #03H
        SUBB    A, R6
        MOV     R6, A

        MOV     A, R4                   ;保存  adc[index]
        ADD     A, ACC
        ADD     A, #LOW (adc)
        MOV     R0, A
        MOV     @R0, AR6
        INC     R0
        MOV     @R0, AR7

;       if(adc[index] < TouchZero[index]) return 0      ;比 0 点还小的值，则认为是键释放
        MOV     A, R4
        ADD     A, ACC
```

```
    ADD     A, #LOW (TouchZero+01H)
    MOV     R1, A
    MOV     A, R4
    ADD     A, ACC
    ADD     A, #LOW (adc)
    MOV     R0, A
    MOV     A, @R0
    MOV     R6, A
    INC     R0
    MOV     A, @R0
    CLR     C
    SUBB    A, @R1                  ;计算  adc[index] - TouchZero[index]
    MOV     A, R6
    DEC     R1
    SUBB    A, @R1
    JNC     L_check_adc_1           ;if(adc[index] >= TouchZero[index]), 转
    MOV     R7, #00H                ;if(adc[index] < TouchZero[index]), 比 0 点还小的值，
                                    ;则认为是键释放, 返回 0
    RET

L_check_adc_1:              ; 计算差值  [R6 R7] = delta = adc[index] - TouchZero[index];
    MOV     A, R4
    ADD     A, ACC
    ADD     A, #LOW (TouchZero+01H)
    MOV     R1, A
    MOV     A, R4
    ADD     A, ACC
    ADD     A, #LOW (adc+01H)
    MOV     R0, A
    CLR     C
    MOV     A, @R0
    SUBB    A, @R1
    MOV     R7, A
    DEC     R0
    MOV     A, @R0
    DEC     R1
    SUBB    A, @R1
    MOV     R6, A
;---- Variable 'delta' assigned to Register 'R6/R7' ----
    CLR     C
    MOV     A, R7
    SUBB    A, #40
    MOV     A, R6
    SUBB    A, #00H
    JC      L_check_adc_2           ;if(delta < 40), 转
    MOV     R7, #1                  ;if(delta >= 40)      return 1; //键按下  返回 1
```

```
    RET

L_check_adc_2:
    SETB    C
    MOV     A, R7
    SUBB    A, #20
    MOV     A, R6
    SUBB    A, #00H
    JNC     L_check_adc_3
    MOV     R7, #0                  ;if(delta <= 20)      return 0; //键释放  返回 0
    RET

L_check_adc_3:
    MOV     R7, #2                  ;if((delta > 20) && (delta < 40))  保持原状态  返回 2
    RET
; END OF _check_adc
/*********************  键处理  50ms 调用 1 次  ************************/
F_ShowLED:
    USING   0                       ;选择第 0 组 R0~R7
    MOV     R7, #0
    LCALL   F_check_adc
    MOV     A, R7
    ANL     A, #0FEH
    JNZ     L_QuitCheck0
    MOV     A, R7
    MOV     C, ACC.0
    CPL     C
    MOV     P_LED0, C               ;if(i == 0)  指示灯灭, if(i == 1)  指示灯亮

L_QuitCheck0:
    MOV     R7, #1
    LCALL   F_check_adc
    MOV     A, R7
    ANL     A, #0FEH
    JNZ     L_QuitCheck1
    MOV     A, R7
    MOV     C, ACC.0
    CPL     C
    MOV     P_LED1, C               ;if(i == 0)  指示灯灭, if(i == 1)  指示灯亮

L_QuitCheck1:
    MOV     R7, #2
    LCALL   F_check_adc
    MOV     A,R7
    ANL     A, #0FEH
    JNZ     L_QuitCheck2
```

```
    MOV     A, R7
    MOV     C, ACC.0
    CPL     C
    MOV     P_LED2, C               ;if(i == 0)  指示灯灭, if(i == 1)  指示灯亮

L_QuitCheck2:
    MOV     R7, #3
    LCALL   F_check_adc
    MOV     A, R7
    ANL     A, #0FEH
    JNZ     L_QuitCheck3
    MOV     A, R7
    MOV     C, ACC.0
    CPL     C
    MOV     P_LED3, C               ;if(i == 0)  指示灯灭, if(i == 1)  指示灯亮

L_QuitCheck3:
    MOV     R7, #4
    LCALL   F_check_adc
    MOV     A, R7
    ANL     A, #0FEH
    JNZ     L_QuitCheck4
    MOV     A, R7
    MOV     C, ACC.0
    CPL     C
    MOV     P_LED4, C               ;if(i == 0)  指示灯灭, if(i == 1)  指示灯亮

L_QuitCheck4:
    MOV     R7, #5
    LCALL   F_check_adc
    MOV     A, R7
    ANL     A, #0FEH
    JNZ     L_QuitCheck5
    MOV     A, R7
    MOV     C, ACC.0
    CPL     C
    MOV     P_LED5, C               ;if(i == 0)  指示灯灭, if(i == 1)  指示灯亮

L_QuitCheck5:
    MOV     R7, #6
    LCALL   F_check_adc
    MOV     A, R7
    ANL     A, #0FEH
    JNZ     L_QuitCheck6
    MOV     A, R7
    MOV     C, ACC.0
```

```
    CPL     C
    MOV     P_LED6, C                   ;if(i == 0)  指示灯灭, if(i == 1)  指示灯亮

L_QuitCheck6:
    MOV     R7, #7
    LCALL   F_check_adc
    MOV     A, R7
    ANL     A, #0FEH
    JNZ     L_QuitCheck7
    MOV     A, R7
    MOV     C, ACC.0
    CPL     C
    MOV     P_LED7, C                   ;if(i == 0)  指示灯灭, if(i == 1)  指示灯亮

L_QuitCheck7:
    RET
; END OF ShowLED
;//==========================================================================
;//  函数: F_delay_ms
;//  描述:  延时子程序。
;//  参数: R7:  延时 ms 数.
;//  返回: none.
;//  备注:  除了 ACCC 和 PSW 外,  所用到的通用寄存器都入栈
;//==========================================================================
F_delay_ms:
    PUSH    AR3                         ;入栈 R3
    PUSH    AR4                         ;入栈 R4

L_delay_ms_1:
    MOV     R3, #HIGH (Fosc_KHZ / 13)
    MOV     R4, #LOW (Fosc_KHZ / 13)

L_delay_ms_2:
    MOV     A, R4                       ;1T  Total 13T/loop
    DEC     R4                          ;2T
    JNZ     L_delay_ms_3                ;4T
    DEC     R3

L_delay_ms_3:
    DEC     A                           ;1T
    ORL     A, R3                       ;1T
    JNZ     L_delay_ms_2                ;4T
    DJNZ    R7, L_delay_ms_1
    POP     AR4                         ;出栈 R2
    POP     AR3                         ;出栈 R3
    RET
```

```
;**********************  中断函数  **********************
F_Timer0_Interrupt:
    RETI

F_Timer1_Interrupt:
    RETI

F_Timer2_Interrupt:
    RETI

F_INT0_Interrupt:
    RETI

F_INT1_Interrupt:
    RETI

F_INT2_Interrupt:
    RETI

F_INT3_Interrupt:
    RETI

F_INT4_Interrupt:
    RETI

F_UART1_Interrupt:
    RETI

F_UART2_Interrupt:
    RETI

F_ADC_Interrupt:
    RETI

F_LVD_Interrupt:
    RETI

F_PCA_Interrupt:
    RETI

F_SPI_Interrupt:
    RETI

END
```

26 同步串行外围接口(SPI 接口)

STC15 系列单片机还提供另一种高速串行通信接口--SPI 接口。SPI 是一种全双工、高速、同步的通信总线，有两种操作模式：主模式和从模式。在主模式中支持高达 3Mbps 的速率（工作频率为 12MHz 时，如果 CPU 主频采用 20MHz 到 36MHz，则可更高，从模式时速度无法太快，SYSclk/4 以内较好），还具有传输完成标志和写冲突标志保护。

下表总结了 STC15 系列单片机内部集成了 SPI 功能的单片机型号：

特殊外围设备 / 单片机型号	8 路 10 位高速 A/D 转换器	CCP/PCA/PWM 功能	1 组高速同步串行口 SPI
STC15W4K32S4 系列	√	√	√
STC15F2K60S2 系列	√	√	√
STC15W1K16S 系列			√
STC15W404S 系列			√
STC15W408AS 系列	√	√	√
STC15W204S 系列			
STC15F408AD 系列	√	√	√
STC15F104W 系列			

上表中 √ 表示对应的系列有相应的功能。

STC15W4K32S4 系列、STC15F2K60S2 系列、STC15W1K16S 系列和 STC15W404S 系列单片机的 SPI 可以在 3 组不同管脚之间进行切换：

- [SS/P1.2, MOSI/P1.3, MISO/P1.4, SCLK/P1.5]；
- [SS_2/P2.4, MOSI_2/P2.3, MISO_2/P2.2, SCLK_2/P2.1]；
- [SS_3/P5.4, MOSI_3/P4.0, MISO_3/P4.1, SCLK_3/P4.3]

注意：现 STC15F2K60S2 及 STC15L2K60S2 系列 C 版本的 SPI 工作于主模式时正常，但工作于从模式时有问题，建议不要使用该版本的 SPI 从模式。

STC15F408AD 系列和 STC15W408AS 系列单片机的 SPI 可以在 2 组不同管脚之间进行切换：

- [SS/P1.2, MOSIP1.3, MISO/P1.4, SCLK/P1.5]；
- [SS_2/P2.4, MOSI_2/P2.3, MISO_2/P2.2, SCLK_2/P2.1]

注意：现 STC15F408AD 及 STC15L408AD 系列 C 版本的 SPI 工作于主模式时正常，但工作于从模式时有问题，建议不要使用该版本的 SPI 从模式。

STC15W204S 系列和 STC15F104W 系列单片机没有 SPI 功能。

特别声明：以 15F 和 15L 开头且有 SPI 功能的芯片，只支持“SPI 主机模式”，不支持“SPI 从机模式”以 15W 开头且有 SPI 功能的芯片，SPI 主/从机模式均支持。

26.1 与 SPI 功能模块相关的特殊功能寄存器

STC15 系列 1T 8051 单片机 SPI 功能模块特殊功能寄存器 SPI Management SFRs

符号	描述	地址	位地址及其符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
SPCTL	SPI Control Register	CEH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	0000,0100
SPSTAT	SPI Status Register	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx,xxxx
SPDAT	SPI Data Register	CFH									0000,0000
IE	Interrupt Enable	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000,0000
IE2	Interrupt Enable 2	AFH	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000,0000
IP2	Interrupt Priority	B5H	-	-	-	-	-	-	PSPI	PS2	xxxx,xx00
AUXR1 P_SW1	Auxiliary Register 1	A2H	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	-	DPS	0000,0000

1、SPI 控制寄存器 SPCTL

SPI 控制寄存器的格式如下：

SPCTL：SPI 控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
SPCTL	CEH	name	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR1	SPR0

SSIG： SS 引脚忽略控制位。

- SSIG=1，MSTR（位 4）确定器件为主机还是从机；
- SSIG=0，SS 脚用于确定器件为主机还是从机.SS 脚可作为 I/O 口使用（见 SPI 主从选择表）

SPEN： SPI 使能位。

- SPEN=1，SPI 使能；
- SPEN=0，SPI 被禁止，所有 SPI 引脚都作为 I/O 口使用。

DORD： 设定 SPI 数据发送和接收的位顺序。

- DORD=1，数据字的 LSB（最低位）最先发送；
- DORD=0，数据字的 MSB（最高位）最先发送。

MSTR： 主/从模式选择位（见 SPI 主从选择表）

CPOL： SPI 时钟极性。

- CPOL=1，SCLK 空闲时为高电平。SCLK 的前时钟沿为下降沿而后沿为上升沿。
- CPOL=0，SCLK 空闲时为低电平。SCLK 的前时钟沿为上升沿而后沿为下降沿。

CPHA： SPI 时钟相位选择。

- CPHA=1，数据在 SCLK 的前时钟沿驱动，并在后时钟沿采样。
- CPHA=0，数据在 SS 为低（SSIG=0）时被驱动，在 SCLK 的后时钟沿被改变，并在前时钟沿被采样。（注：SSIG=1 时的操作未定义）

SPR1、SPR0：SPI 时钟频率选择控制位。

- STC15W 系列与 STC15F/L 系列具有不同的 SPI 时钟频率，其中，STC15W 系列单片机的 SPI 时钟频率选择如下表所列：

SPI 时钟频率的选择（表中，CPU_CLK 是 CPU 时钟。）

SPR1	SPR0	时钟（ SCLK ）
0	0	CPU_CLK/4
0	1	CPU_CLK/8
1	0	CPU_CLK/16
1	1	CPU_CLK/32

- STC15F/L 系列单片机的 SPI 时钟频率选择如下表所列：

SPI 时钟频率的选择（表中，CPU_CLK 是 CPU 时钟。）

SPR1	SPR0	时钟（ SCLK ）
0	0	CPU_CLK/4
0	1	CPU_CLK/16
1	0	CPU_CLK/64
1	1	CPU_CLK/128

2、SPI 状态寄存器 SPSTAT

SPI 状态寄存器的格式如下：

SPSTAT：SPI 状态寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
SPSTAT	CDH	name	SPIF	WCOL	-	-	-	-	-	-

SPIF： SPI 传输完成标志。

- 当一次串行传输完成时，SPIF 置位。此时，如果 SPI 中断被打开（即 ESPI（IE2.1）和 EA（IE.7）都置位），则产生中断。当 SPI 处于主模式且 SSIG=0 时，如果 SS 为输入并被驱动为低电平，SPIF 也将置位,表示“模式改变”。SPIF 标志通过软件向其写入“1”清零。

WCOL：SPI 写冲突标志。

- 在数据传输的过程中如果对 SPI 数据寄存器 SPDAT 执行写操作，WCOL 将置位。WCOL 标志通过软件向其写入"1"清零。

3、SPI 数据寄存器 SPDAT

SPI 数据寄存器的格式如下：

SPDAT：SPI 数据寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
SPDAT	CFH	name								

SPDAT.7 - SPDAT.0：传输的数据位 Bit7 ～ Bit0

4、中断允许寄存器 IE 及 IE2

IE：中断允许寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	name	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

EA： CPU 的中断开放标志

- EA=1，CPU 开放中断，
- EA=0，CPU 屏蔽所有的中断申请。

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。

IE2：中断允许寄存器 2

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE2	AFH	name	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2

ESPI： SPI 中断允许位

- ESPI=1，允许 SPI 中断，
- ESPI=0，禁止 SPI 中断。

5、中断优先级控制寄存器 IP2

IP2：中断优先级控制寄存器 2

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IP2	B5H	name	-	-	-	-	-	-	PSPI	PS2

PSPI： SPI 中断优先级控制位。

- 当 PSPI=0 时，SPI 中断为最低优先级中断（优先级 0）
- 当 PSPI=1 时，SPI 中断为最高优先级中断（优先级 1）

6、控制 SPI 功能切换的寄存器 AUXR1（P_SW1）

外围设备切换控制寄存器 1 的格式如下：

AUXR1/P_SW1：外围设备切换控制寄存器 1（不可位寻址）

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000,0000

SPI 可在 3 个地方切换，由 SPI_S1 / SPI_S0 两个控制位来选择		
SPI_S1	SPI_S0	SPI 可在 P1/P2/P4 之间来回切换
0	0	SPI 在[P1.2/SS，P1.3/MOSI，P1.4/MISO，P1.5/SCLK]
0	1	SPI 在[P2.4/SS_2，P2.3/MOSI_2，P2.2/MISO_2，P2.1/SCLK_2]
1	0	SPI 在[P5.4/SS_3，P4.0/MOSI_3，P4.1/MISO_3，P4.3/SCLK_3]
1	1	无效

CCP 可在 3 个地方切换，由 CCP_S1 / CCP_S0 两个控制位来选择		
CCP_S1	CCP_S0	CCP 可在 P1/P2/P3 之间来回切换
0	0	CCP 在[P1.2/ECI，P1.1/CCP0，P1.0/CCP1，P3.7/CCP2]
0	1	CCP 在[P3.4/ECI_2，P3.5/CCP0_2，P3.6/CCP1_2，P3.7/CCP2_2]
1	0	CCP 在[P2.4/ECI_3，P2.5/CCP0_3，P2.6/CCP1_3，P2.7/CCP2_3]
1	1	无效

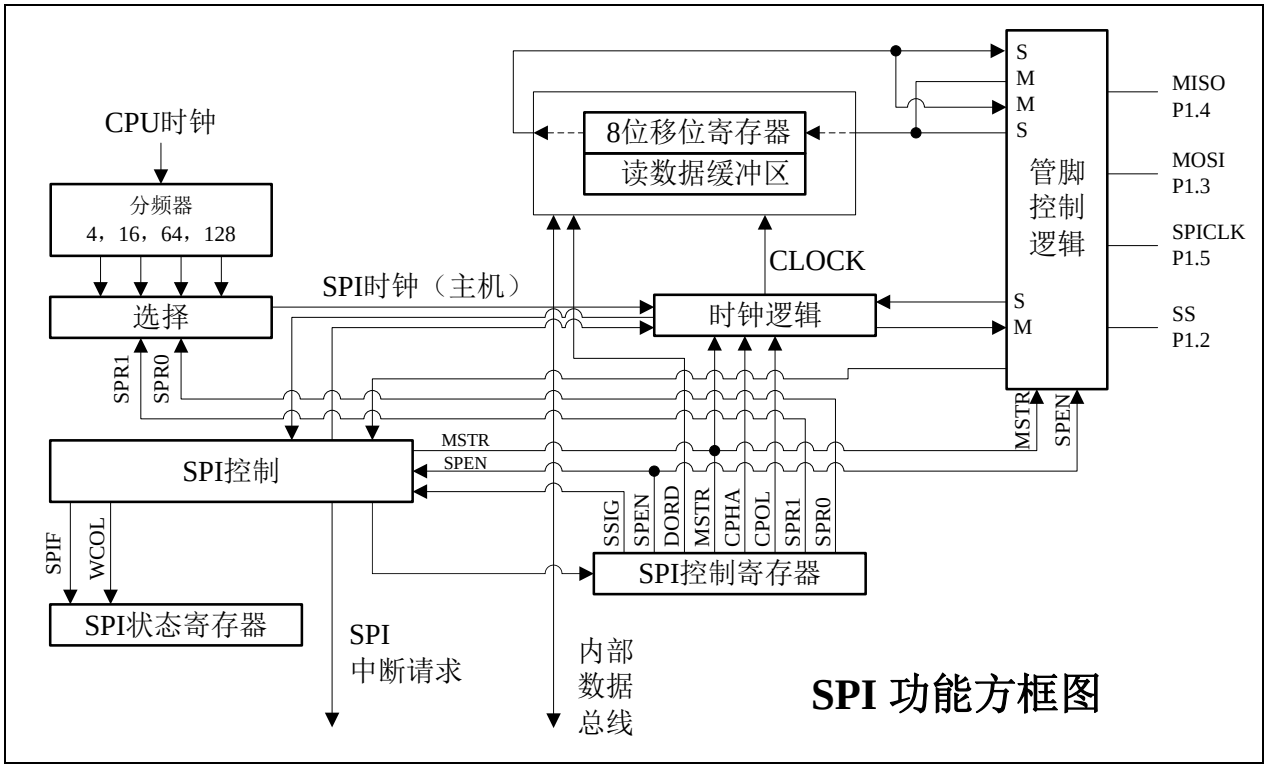
串口 1/S1 可在 3 个地方切换，由 S1_S0 及 S1_S1 控制位来选择		
S1_S1	S1_S0	串口 1/S1 可在 P1/P3 之间来回切换
0	0	串口 1/S1 在[P3.0/RxD，P3.1/TxD]
0	1	串口 1/S1 在[P3.6/RxD_2，P3.7/TxD_2]
1	0	串口 1/S1 在[P1.6/RxD_3/XTAL2，P1.7/TxD_3/XTAL1] 串口 1 在 P1 口时要使用内部时钟
1	1	无效

DPS： DPTR registers select bit. DPTR 寄存器选择位

- 0，使用缺省数据指针 DPTR0
- 1，使用另一个数据指针 DPTR1

26.2 SPI 接口的结构

STC15 系列单片机的 SPI 功能方框图如下图所示。



SPI 的核心是一个 8 位移位寄存器和数据缓冲器，数据可以同时发送和接收。在 SPI 数据的传输过程中，发送和接收的数据都存储在数据缓冲器中。

对于主模式，若要发送一字节数据，只需将这个数据写到 SPDAT 寄存器中。主模式下 SS 信号不是必需的；但是在从模式下，必须在 SS 信号变为有效并接收到合适的时钟信号后，方可进行数据传输。在从模式下，如果一个字节传输完成后，SS 信号变为高电平，这个字节立即被硬件逻辑标志为接收完成，SPI 接口准备接收下一个数据。

26.3 SPI 接口的数据通信

SPI 接口有 4 个管脚：SCLK，MISO，MOSI 和 SS，可在 3 组管脚之间进行切换：

- [SCLK/P1.5，MISO/P1.4，MOSI/P13 和 SS/P1.2]；
- [SCLK_2/P2.1，MISO_2/P2.2，MOSI_2/P2.3 和 SS_2/P2.4]；
- [SCLK_3/P4.3，MISO_3/P4.1，MOSI_3/P4.0 和 SS_3/P5.4]

MOSI（Master Out Slave In，主出从入）：主器件的输出和从器件的输入，用于主器件到从器件的串行数据传输。

- 当 SPI 作为主器件时，该信号是输出；
- 当 SPI 作为从器件时，该信号是输入。

数据传输时最高位在先，低位在后。根据 SPI 规范，多个从机可以共享一根 MOSI 信号线。在时钟边界的前半周期，主机将数据放在 MOSI 信号线上，从机在该边界处获取该数据。

MISO（Master In Slave Out，主入从出）：从器件的输出和主器件的输入，用于实现从器件到主器件的数据传输。

- 当 SPI 作为主器件时，该信号是输入；
- 当 SPI 作为从器件时，该信号是输出。

数据传输时最高位在先，低位在后。SPI 规范中，一个主机可连接多个从机，因此，主机的 MISO 信号线会连接到多个从机上，或者说，多个从机共享一根 MISO 信号线。当主机与一个从机通信时，其他从机应将其 MISO 引脚驱动置为高阻状态。

SCLK（SPI Clock，串行时钟信号）：串行时钟信号是主器件的输出和从器件的输入，用于同步主器件和从器件之间在 MOSI 和 MISO 线上的串行数据传输。当主器件启动一次数据传输时，自动产生 8 个 SCLK 时钟周期信号给从机。在 SCLK 的每个跳变处（上升沿或下降沿）移出一位数据。所以，一次数据传输可以传输一个字节的数据。

SCLK、MOSI 和 MISO 通常和两个或更多 SPI 器件连接在一起。数据通过 MOSI 由主机传送到从机，通过 MISO 由从机传送到主机。SCLK 信号在主模式时为输出，在从模式时为输入。如果 SPI 系统被禁止，即 SPEN（SPCTL.6）= 0（复位值），这些管脚都可作为 I/O 口使用。

SS（Slave Selcct，从机选择信号）：这是一个输入信号，主器件用它来选择处于从模式的 SPI 模块。主模式和从模式下，SS 的使用方法不同。

- 在主模式下，SPI 接口只能有一个主机，不存在主机选择问题，该模式下 SS 不是必需的。主模式下通常将主机的 SS 管脚通过 10KΩ 的电阻上拉高电平。每一个从机的 SS 接主机的 I/O 口，由主机控制电平高低，以便主机选择从机。
- 在从模式下，不管发送还是接收，SS 信号必须有效。因此在一次数据传输开始之前必须将 SS 为低电平。SPI 主机可以使用 I/O 口选择一个 SPI 器件作为当前的从机。

SPI 从器件通过其 SS 脚确定是否被选择。如果满足下面的条件之一，SS 就被忽略：

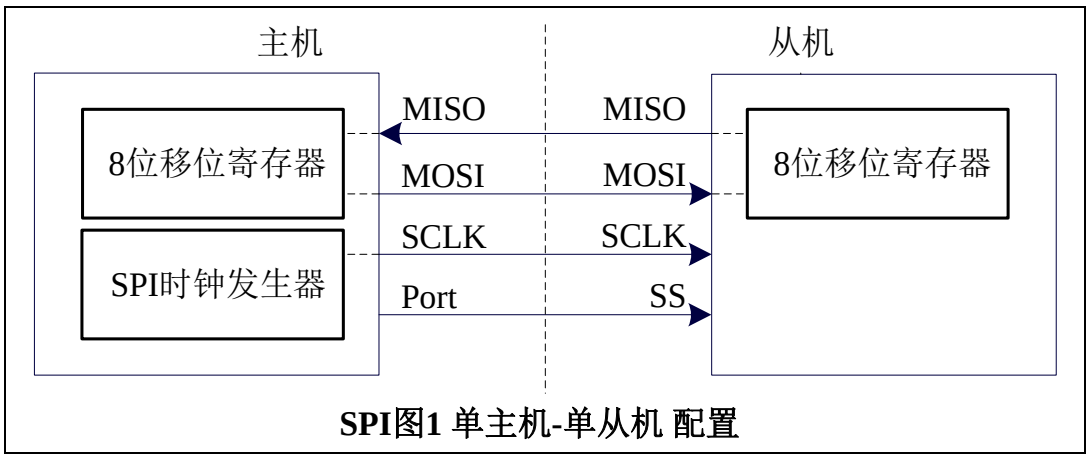
- 如果 SPI 系统被禁止，即 SPEN（SPCTL.6）= 0（复位值）
- 如果 SPI 配置为主机，即 MSTR（SPCTL.4）= 1，并且 P1.2/SS 配置为输出（通过 P1M0.2 和 P1M1.2）
- 如果 SS 脚被忽略，即 SSIG（SPCTL.7）=1，该脚配置用于 I/O 口功能。

注：即使 SPI 被配置为主机（MSTR=1），它仍然可以通过拉低 SS 脚配置为从机（如果 P1.2/SS 配置为输入且 SSIG=0）。要使能该特性，应当置位 SPIF（SPSTAT.7）。

26.3.1 SPI 接口的数据通信方式

STC15 系列单片机的 SPI 接口的数据通信方式有 3 种：单主机—从机方式、双器件方式（器件可互为主机和从机）和单主机—多从机方式。

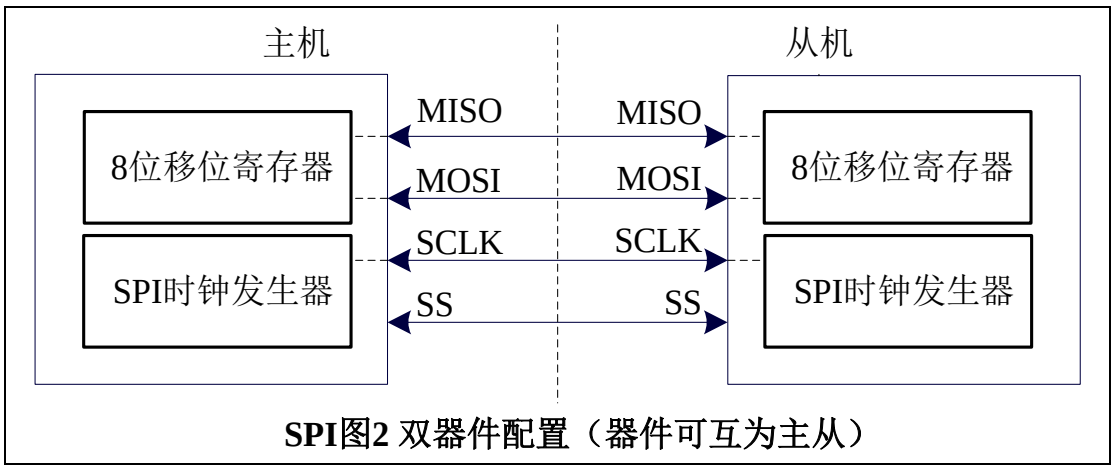
单主机—单从机方式的连接图如下 SPI 图 1 所示。



SPI图1 单主机-单从机 配置

在上图 SPI 图 1 中，从机的 SSIG（SPCTL.7）为 0，SS 用于选择从机。SPI 主机可使用任何端口（包括 P1.2/SS）来驱动 SS 脚。主机 SPI 与从机 SPI 的 8 位移位寄存器连接成一个循环的 16 位移位寄存器。当主机程序向 SPDAT 寄存器写入一个字节时，立即启动一个连续的 8 位移位通信过程：主机的 SCLK 引脚向从机的 SCLK 引脚发出一串脉冲，在这串脉冲的驱动下，主机 SPI 的 8 位移位寄存器中的数据移动到了从机 SPI 的 8 移位寄存器中。与此同时，从机 SPI 的 8 位移位寄存器中的数据移动到了主机 SPI 的 8 位移位寄存器中。由此，主机既可向从机发送数据，又可读从机中的数据。

双器件方式（器件可互为主机和从机）的连接图如下 SPI 图 2 所示。



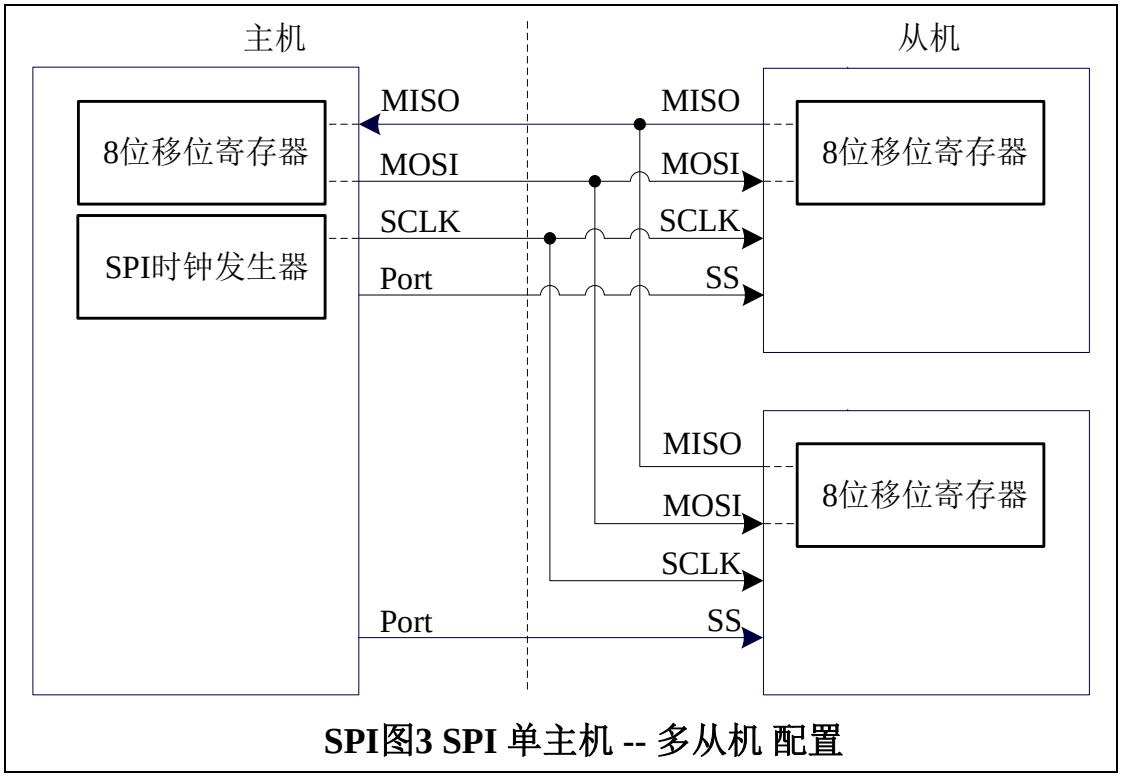
SPI图2 双器件配置（器件可互为主从）

上图 SPI 图 2 所示为两个器件互为主从的情况。当没有发生 SPI 操作时，两个器件都可配置为主机（MSTR=1），将 SSIG 清零并将 P1.2（SS）配置为准双向模式。当其中一个器件启动传输时，它可将 P1.2/SS 配置为输出并驱动为低电平，这样就强制另一个器件变为从机。

双方初始化时将自己设置成忽略 SS 脚的 SPI 从模式。当一方要主动发送数据时，先检测 SS 脚的电平，如果 SS 脚是高电平，就将自己设置成忽略 SS 脚的主模式。通信双方平时将 SPI 设置成没有被选中的从模式。在该模式下，MISO、MOSI、SCLK 均为输入，当多个 MCU 的 SPI 接口以此模式并联时不会发生总线冲突。这种特性在互为主/从、一主多从等应用中很有用。

注意：互为主/从模式时，双方的 SPI 速率必须相同。如果使用外部晶体振荡器，双方的晶体频率也要相同。

双器件方式（器件可互为主机和从机）的连接图如下 SPI 图 3 所示。



SPI图3 SPI 单主机 -- 多从机 配置

在上图 SPI 图 3 中，从机的 SSIG（SPCTL.7）为 0，从机通过对应的 SS 信号被选中。SPI 主机可使用任何端口（包括 P1.2/SS）来驱动 SS 脚。

26.3.2 对 SPI 进行配置

STC15 系列单片机进行 SPI 通信时，主机和从机的选择由 SPEN、SSIG、SS 引脚（P1.2）和 MSTR 联合控制。下表所示为主/从模式的配置以及模式的使用和传输方向。

SPI 主从模式选择

SPEN	SSIG	SS 脚 P1.2	MSTR	主或从模式	MISO P1.4	MOSI P1.3	SCLK P1.5	备注
0	X	P1.2/SS	X	SPI 功能禁止	P1.4/MISO	P1.3/MOSI	P1.5/SCLK	SPI 禁止。 P1.2/SS，P1.3/MOSI，P1.4/MISO 和 P1.5/SCLK 作为普通 I/O 口使用
1	0	0	0	从机模式	输出	输入	输入	选择作为从机
1	0	1	0	从机模式 未被选中	高阻	输入	输入	未被选中。 免总线冲突 MISO 为高阻状态，以避
1	0	0	1—>0	从机模式	输出	输入	输入	P1.2/SS 配置为输入或准双向口。 SSIG 为 0。如果选择 SS 被驱动为低电平，则被选择作为从机。当 SS 变为低电平时，MSTR 将清零。 注：当 SS 处于输入模式时，如被驱动为低电平且 SSIG=0 时，MSTR 位自动清零。
1	0	1	1	主（空闲）	输入	高阻	高阻	当主机空闲时 MOSI 和 SCLK 为高阻态以避免总线冲突。用户必须将 SCLK 上拉或下拉（根据 CPOL/SPCTL.3 的取值）以避免 SCLK 出现悬浮状态。
				主（激活）		输出	输出	作为主机激活时，MOSI 和 SCLK 为推挽输出
1	1	P1.2/SS	0	从	输出	输入	输入	
1	1	P1.2/SS	1	主	输入	输出	输出	

26.3.3 作为主机/从机时的额外注意事项

作为从机时的额外注意事项：

当 CPHA=0 时，SSIG 必须为 0（也就是不能忽略 SS 脚），SS 脚必须置低并且在每个连续的串行字节发送完后须重新设置为高电平。如果 SPDAT 寄存器在 SS 有效（低电平）时执行写操作，那么将导致一个写冲突错误。CPHA=0 且 SSIG=0 时的操作未定义。

当 CPHA=1 时，SSIG 可以置 1（即可以忽略 SS 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）。这种方式有时适用于具有单固定主机和单从机驱动 MISO 数据线的系统。

作为主机时的额外注意事项：

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，主机对 SPI 数据寄存器的写操作将启动 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。

需要注意的是，主机可以通过将对应器件的 SS 脚驱动为低电平实现与之通信。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）置位并产生一个中断（如果 SPI 中断使能）。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。

26.3.4 通过 SS 改变模式

如果 SPEN=1，SSIG=0 且 MSTR=1，SPI 使能为主机模式。

SS 脚可配置为输入（[P2M1.2，P2M0.2] = [1，0]）或准双向模式（[P2M1.2，P2M0.2] = [0，0]）。这种情况下，另外一个主机可将该 SS 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 系统执行以下动作：

1）MSTR 清零并且 CPU 变成从机。这样 SPI 就变成从机。MOSI 和 SCLK 强制变为输入模式，而 MISO 则变为输出模式。

2）SPSTAT 的 SPIF 标志位置位。如果 SPI 中断已被使能，则产生 SPI 中断。

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所清零而用户想继续将 SPI 作为主机，这时就必须重新置位 MSTR，否则就进入从机模式。

26.3.5 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进行写操作时，WCOL 位（SPSTAT.6）将置位以指示数据冲突。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。

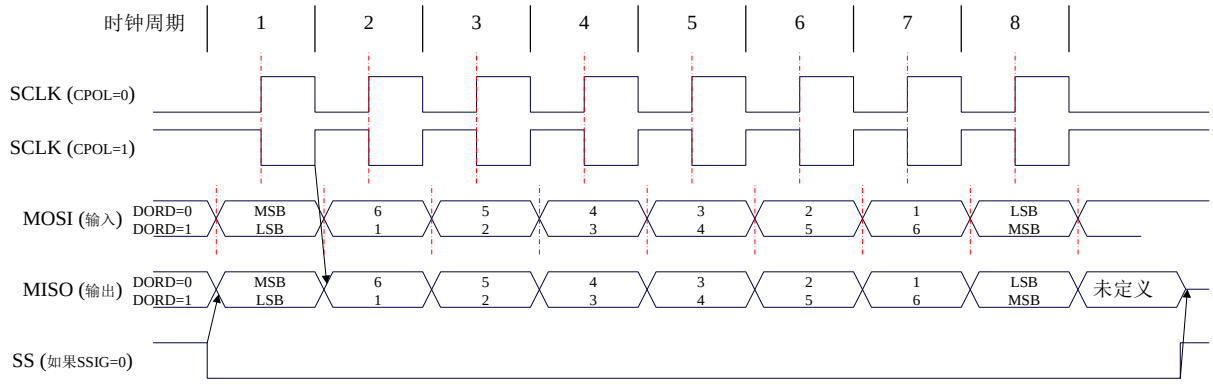
接收数据时，接收到的数据传送到一个并行读数据缓冲区，这样将释放移位寄存器以进行下一个数据的接收。但必须在下个字符完全移入之前从数据寄存器中读出接收到的数据，否则，前一个接收数据将丢失。

WCOL 可通过软件向其写入“1”清零。

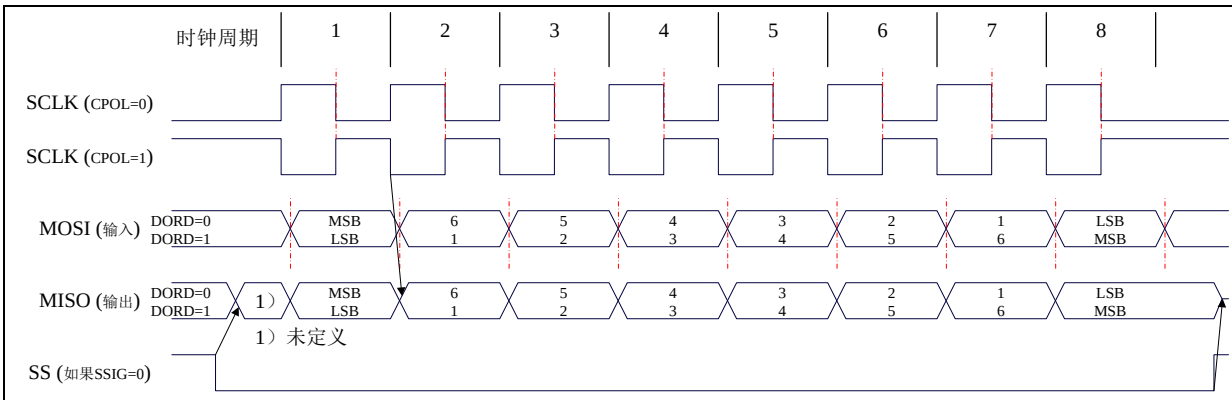
26.3.6 数据模式

时钟相位位（CPHA）允许用户设置采样和改变数据的时钟边沿。时钟极性位 CPOL 允许用户设置时钟极性。

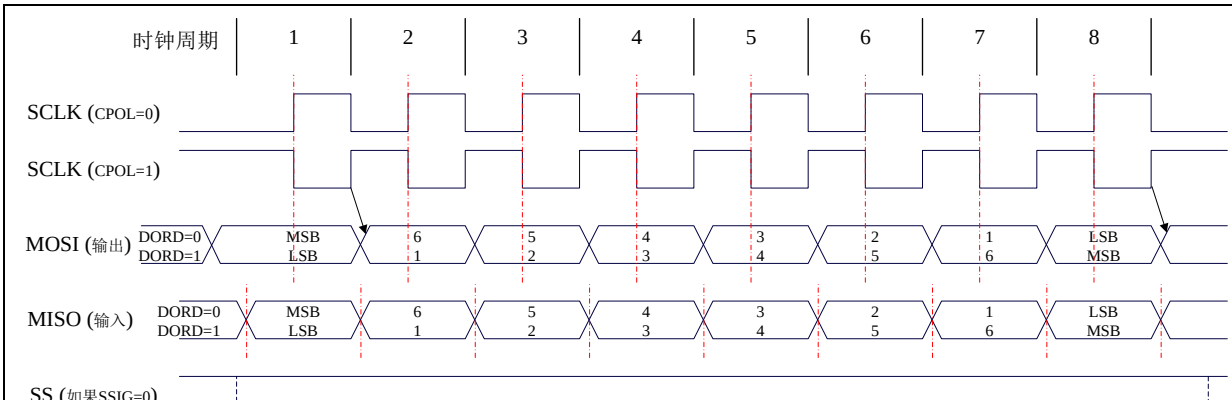
SPI 图 4 一图 7 所示为时钟相位位 CPHA 的不同设定。



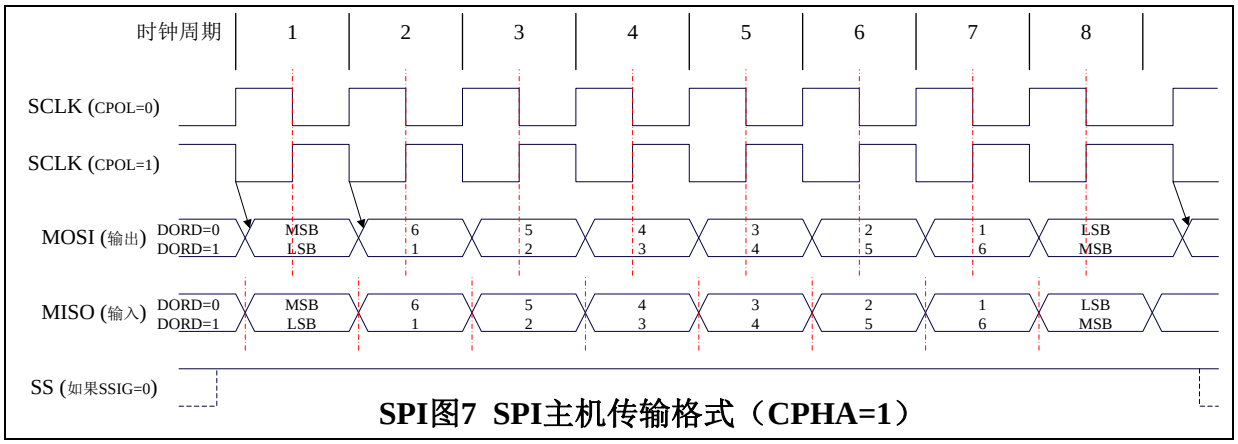
SPI图4 SPI从机传输格式（CPHA=0）



SPI图5 SPI从机传输格式（CPHA=1）



SPI图6 SPI主机传输格式（CPHA=0）



SPI图7 SPI主机传输格式（CPHA=1）

SPI 接口的时钟信号线 SCLK 有 Idle 和 Active 两种状态：

- Ide 状态时指在不进行数据传输的时候（或数据传输完成后）SCLK 所处的状态；
- Active 是与 Idle 相对的一种状态。

时钟相位位（CPHA）允许用户设置采样和改变数据的时钟边沿。时钟极性 CPOL 允许用户设置时钟极性。

- 如果 CPOL=0，Idle 状态=低电平，Active 状态=高电平；
- 如果 CPOL=1，Idle 状态=高电平，Active 状态=低电平。

主机总是在 SCLK=ldle 状态时，将下一位要发送的数据置于数据线 MOSI 上。

从 Idle 状态到 Active 状态的转变，称为 SCLK 前沿；从 Active 状态到 Idle 状态的转变，称为 SCLK 后沿。一个 SCLK 前沿和后沿构成一个 SCLK 时钟周期，一个 SCLK 时钟周期传输一位数据。

SPI 时钟预分频器选择

SPI 时钟预分频器选择是通过 SPCTL 寄存器中的 SPR1-SPRO 位实现的

SPI 时钟频率的选择（其中，CPU_CLK 是 CPU 时钟。）

SPR1	SPR0	时钟（ SCLK ）
0	0	CPU_CLK/4
0	1	CPU_CLK/8
1	0	CPU_CLK/16
1	1	CPU_CLK/32

26.4 适用单主单从系统的 SPI 功能测试程序(C 和汇编)

26.4.1 中断方式

1. C 程序

```
/*----演示 STC 1T 系列单片机 SPI 功能(适用单主单从，中断方式)----*/
/*---- 在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可-----------*/
//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"

//#define MASTER                    //define:master undefine:slave
#define   FOSC              18432000L
#define   BAUD              (256 - FOSC / 32 / 115200)

typedef   unsigned char     BYTE;
typedef   unsigned int      WORD;
typedef   unsigned long     DWORD;

#define   URMD              0       //0:使用定时器 2 作为波特率发生器
                                    //1:使用定时器 1 的模式 0(16 位自动重载模式)作为波特率发生器
                                    //2:使用定时器 1 的模式 2(8 位自动重载模式)作为波特率发生器

sfr       T2H = 0xd6;               //定时器 2 高 8 位
sfr       T2L = 0xd7;               //定时器 2 低 8 位
sfr       AUXR = 0x8e;              //辅助寄存器
sfr       SPSTAT = 0xcd;            //SPI 状态寄存器

#define   SPIF      0x80            //SPSTAT.7
#define   WCOL      0x40            //SPSTAT.6

sfr       SPCTL = 0xce;             //SPI 控制寄存器

#define   SSIG      0x80            //SPCTL.7
#define   SPEN      0x40            //SPCTL.6
#define   DORD      0x20            //SPCTL.5
#define   MSTR      0x10            //SPCTL.4
#define   CPOL      0x08            //SPCTL.3
#define   CPHA      0x04            //SPCTL.2
#define   SPDHH     0x00            //CPU_CLK/4
#define   SPDH      0x01            //CPU_CLK/8
#define   SPDL      0x02            //CPU_CLK/16
#define   SPDLL     0x03            //CPU_CLK/32
```

```
sfr         SPDAT = 0xcf;               //SPI 数据寄存器
sbit        SPISS =  P1^1;              //SPI 从机选择口, 连接到其它 MCU 的 SS 口
                                        //当 SPI 为一主多从模式时,
                                        //请使用主机的普通 IO 口连接到从机的 SS 口

sfr         IE2  =    0xAF;             //中断控制寄存器 2
#define     ESPI      0x02      //IE2.1

void InitUart();
void InitSPI();
void SendUart(BYTE dat);                //发送数据到 PC

BYTE RecvUart();                        //从 PC 接收数据
//////////////////////////////////////////////////////////
void main()
{
    InitUart();                         //初始化串口
    InitSPI();                          //初始化 SPI
    IE2 |= ESPI;
    EA = 1;
    while (1)
    {
#ifdef MASTER                           //对于主机(接收串口数据并发送给从机,同时
                                        //  从即接收 SPI 数据并回传给 PC)
        ACC = RecvUart();
        SPISS = 0;                      //拉低从机的 SS
        SPDAT = ACC;                    //触发 SPI 发送数据
#endif
    }
}
//////////////////////////////////////////////////////////
void spi_isr() interrupt 9 using 1      //SPI 中断服务程序  9 (004BH)
{
    SPSTAT = SPIF | WCOL;               //清除 SPI 状态位
#ifdef MASTER
    SPISS = 1;                          //拉高从机的 SS
    SendUart(SPDAT);                    //返回 SPI 数据
#else                                   //对于从机(从主机接收 SPI 数据,同时
    SPDAT = SPDAT;                      //发送前一个 SPI 数据给主机)
#endif
}
//////////////////////////////////////////////////////////
void InitUart()
{
    SCON = 0x5a;                        //设置串口为 8 位可变波特率
#if URMD == 0
```

```
    T2L = 0xd8;                 //设置波特率重装值
    T2H = 0xff;                 //115200 bps(65536-18432000/4/115200)
    AUXR = 0x14;                //T2 为 1T 模式, 并启动定时器 2
    AUXR |= 0x01;               //选择定时器 2 为串口 1 的波特率发生器
#elif URMD == 1
    AUXR = 0x40;                //定时器 1 为 1T 模式
    TMOD = 0x00;                //定时器 1 为模式 0(16 位自动重载)
    TL1 = 0xd8;                 //设置波特率重装值
    TH1 = 0xff;                 //115200 bps(65536-18432000/4/115200)
    TR1 = 1;                    //定时器 1 开始启动
#else
    TMOD = 0x20;                //设置定时器 1 为 8 位自动重装载模式
    AUXR = 0x40;                //定时器 1 为 1T 模式
    TH1 = TL1 = 0xfb;           //115200 bps(256 - 18432000/32/115200)
    TR1 = 1;
#endif
}
//////////////////////////////////////////////////////////
void InitSPI()
{
    SPDAT = 0;                  //初始化 SPI 数据
    SPSTAT = SPIF | WCOL;       //清除 SPI 状态位
#ifdef MASTER
    SPCTL = SPEN | MSTR;        //主机模式
#else
    SPCTL = SPEN;               //从机模式
#endif
}
//////////////////////////////////////////////////////////
void SendUart(BYTE dat)
{
    while (!TI);                //等待发送完成
    TI = 0;                     //清除发送标志
    SBUF = dat;                 //发送串口数据
}
//////////////////////////////////////////////////////////
BYTE RecvUart()
{
    while (!RI);                //等待串口数据接收完成
    RI = 0;                     //清除接收标志
    return SBUF;                //返回串口数据
}
```

2. 汇编程序

```
/*----演示 STC 1T 系列单片机 SPI 功能(适用单主单从，中断方式)----*/
/*---- 在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可------------*/
//假定测试芯片的工作频率为 18.432MHz
#define     MASTER              //define:master undefine:slave
#define     URMD    0           //0:使用定时器 2 作为波特率发生器
                                //1:使用定时器 1 的模式 0(16 位自动重载模式)作为波特率发生器
                                //2:使用定时器 1 的模式 2(8 位自动重载模式)作为波特率发生器

T2H         DATA    0D6H        //定时器 2 高 8 位
T2L         DATA    0D7H        //定时器 2 低 8 位

AUXR        DATA    08EH        ;辅助寄存器
SPSTAT      DATA    0CDH        ;SPI 状态寄存器
SPIF        EQU     080H        ;SPSTAT.7
WCOL        EQU     040H        ;SPSTAT.6
SPCTL       DATA    0CEH        ;SPI 控制寄存器
SSIG        EQU     080H        ;SPCTL.7
SPEN        EQU     040H        ;SPCTL.6
DORD        EQU     020H        ;SPCTL.5
MSTR        EQU     010H        ;SPCTL.4
CPOL        EQU     008H        ;SPCTL.3
CPHA        EQU     004H        ;SPCTL.2
SPDHH       EQU     000H        ;CPU_CLK/4
SPDH        EQU     001H        ;CPU_CLK/8
SPDL        EQU     002H        ;CPU_CLK/16
SPDLL       EQU     003H        ;CPU_CLK/32
SPDAT       DATA    0CFH        ;SPI 数据寄存器
SPISS       BIT     P1.1        ;SPI 从机选择口，连接到其它 MCU 的 SS 口
                                ;当 SPI 为一主多从模式时,请使用主机的普通 IO 口连接到从机的 SS 口

IE2         EQU     0AFH        ;中断控制寄存器 2
ESPI        EQU     02H         ;IE2.1
;///////////////////////////////////////////////////////////
    ORG     0000H
    LJMP    RESET
    ORG     004BH               ;SPI 中断服务程序

SPI_ISR:
    PUSH    ACC
    PUSH    PSW
    MOV     SPSTAT, #SPIF | WCOL    ;清除 SPI 状态位
#ifdef  MASTER
    SETB    SPISS               ;拉高从机的 SS
    MOV     A, SPDAT            ;返回 SPI 数据
```

```
    LCALL   SEND_UART
#else                               ;对于从机(从主机接收 SPI 数据,同时发送前一个 SPI 数据给主机)
    MOV     SPDAT, SPDAT
#endif
    POP     PSW
    POP     ACC
    RETI
;///////////////////////////////////////////////////////
    ORG     0100H

RESET:
    LCALL   INIT_UART           ;初始化串口
    LCALL   INIT_SPI            ;初始化 SPI
    ORL     IE2, #ESPI
    SETB    EA

START:
#ifdef MASTER                   ;对于主机(接收串口数据并发送给从机,同时
    LCALL   RECV_UART           ; 从从即接收 SPI 数据并回传给 PC)
    CLR     SPISS               ;拉低从机的 SS
    MOV     SPDAT, A            ;触发 SPI 发送数据
#endif
    SJMP    START
;///////////////////////////////////////////////////////
INIT_UART:
    MOV     SCON, #5AH          ;设置串口为 8 位可变波特率
#if URMD == 0
    MOV     T2L, #0D8H          ;设置波特率重装值(65536-18432000/4/115200)
    MOV     T2H, #0FFH
    MOV     AUXR, #14H          ;T2 为 1T 模式, 并启动定时器 2
    ORL     AUXR, #01H          ;选择定时器 2 为串口 1 的波特率发生器
#elif URMD == 1
    MOV     AUXR, #40H          ;定时器 1 为 1T 模式
    MOV     TMOD, #00H          ;定时器 1 为模式 0(16 位自动重载)
    MOV     TL1, #0D8H          ;设置波特率重装值(65536-18432000/4/115200)
    MOV     TH1, #0FFH
    SETB    TR1                 ;定时器 1 开始运行
#else
    MOV     TMOD, #20H          ;设置定时器 1 为 8 位自动重装载模式
    MOV     AUXR, #40H          ;定时器 1 为 1T 模式
    MOV     TL1, #0FBH          ;115200 bps(256 - 18432000/32/115200)
    MOV     TH1, #0FBH
    SETB    TR1
#endif
    RET
;///////////////////////////////////////////////////////
```

```
INIT_SPI:
    MOV     SPDAT, #0               ;初始化 SPI 数据
    MOV     SPSTAT, #SPIF | WCOL    ;清除 SPI 状态位
#ifdef MASTER
    MOV     SPCTL, #SPEN | MSTR     ;主机模式
#else
    MOV     SPCTL, #SPEN            ;从机模式
#endif
    RET
;////////////////////////////////////////////////////////
SEND_UART:
    JNB     TI, $                   ;等待发送完成
    CLR     TI                      ;清除发送标志
    MOV     SBUF, A                 ;发送串口数据
    RET
;////////////////////////////////////////////////////////
RECV_UART:
    JNB     RI, $                   ;等待串口数据接收完成
    CLR     RI                      ;清除接收标志
    MOV     A, SBUF                 ;返回串口数据
    RET
    RET
;////////////////////////////////////////////////////////
    END
```

26.4.2 查询方式

1. C 程序

```
/*----演示 STC 1T 系列单片机 SPI 功能(适用单主单从，查询方式)----*/
/*---- 在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可------------*/
//假定测试芯片的工作频率为 18.432MHz

#include "reg51.h"

//#define MASTER                    //define:master undefine:slave
#define     FOSC            18432000L
#define     BAUD            (256 - FOSC / 32 / 115200)

typedef     unsigned char   BYTE;
typedef     unsigned int    WORD;
typedef     unsigned long   DWORD;

#define     URMD            0   //0:使用定时器 2 作为波特率发生器
                                //1:使用定时器 1 的模式 0(16 位自动重载模式)作为波特率发生器
                                //2:使用定时器 1 的模式 2(8 位自动重载模式)作为波特率发生器

sfr         T2H = 0xd6;         //定时器 2 高 8 位
sfr         T2L = 0xd7;         //定时器 2 低 8 位
sfr         AUXR = 0x8e;        //辅助寄存器
sfr         SPSTAT = 0xcd;      //SPI 状态寄存器

#define     SPIF    0x80        //SPSTAT.7
#define     WCOL    0x40        //SPSTAT.6

sfr         SPCTL = 0xce;       //SPI 控制寄存器

#define     SSIG    0x80        //SPCTL.7
#define     SPEN    0x40        //SPCTL.6
#define     DORD    0x20        //SPCTL.5
#define     MSTR    0x10        //SPCTL.4
#define     CPOL    0x08        //SPCTL.3
#define     CPHA    0x04        //SPCTL.2
#define     SPDHH   0x00        //CPU_CLK/4
#define     SPDH    0x01        //CPU_CLK/8
#define     SPDL    0x02        //CPU_CLK/16
#define     SPDLL   0x03        //CPU_CLK/32

sfr         SPDAT = 0xcf;       //SPI 数据寄存器
sbit        SPISS = P1^1;       //SPI 从机选择口，连接到其它 MCU 的 SS 口
```

```
                                    //当 SPI 为一主多从模式时,
                                    //请使用主机的普通 IO 口连接到从机的 SS 口
void InitUart();
void InitSPI();
void SendUart(BYTE dat);            //发送数据到 PC

BYTE RecvUart();                    //从 PC 接收数据
BYTE SPISwap(BYTE dat);             //主机与从机之间交换数据
/////////////////////////////////////////////////////////
void main()
{
    InitUart();                     //初始化串口
    InitSPI();                      //初始化 SPI
    while (1)
    {
#ifdef MASTER                       //对于主机(接收串口数据  并发送给从机,同时
                                    //  从从即接收 SPI 数据并回传给 PC)
    SendUart(SPISwap(RecvUart()));
#else                               //对于从机(从主机接收 SPI 数据,同时
    ACC = SPISwap(ACC);             //  发送前一个 SPI 数据给主机)
#endif
    }
}
/////////////////////////////////////////////////////////
void InitUart()
{
    SCON = 0x5a;                    //设置串口为 8 位可变波特率
#if URMD == 0
    T2L = 0xd8;                     //设置波特率重装值
    T2H = 0xff;                     //115200 bps(65536-18432000/4/115200)
    AUXR = 0x14;                    //T2 为 1T 模式, 并启动定时器 2
    AUXR |= 0x01;                   //选择定时器 2 为串口 1 的波特率发生器
#elif URMD == 1
    AUXR = 0x40;                    //定时器 1 为 1T 模式
    TMOD = 0x00;                    //定时器 1 为模式 0(16 位自动重载)
    TL1 = 0xd8;                     //设置波特率重装值
    TH1 = 0xff;                     //115200 bps(65536-18432000/4/115200)
    TR1 = 1;                        //定时器 1 开始启动
#else
    TMOD = 0x20;                    //设置定时器 1 为 8 位自动重装载模式
    AUXR = 0x40;                    //定时器 1 为 1T 模式
    TH1 = TL1 = 0xfb;               //115200 bps(256 - 18432000/32/115200)
    TR1 = 1;
    #endif
}
/////////////////////////////////////////////////////////
```

```
void InitSPI()
{
    SPDAT = 0;                  //初始化 SPI 数据
    SPSTAT = SPIF | WCOL;       //清除 SPI 状态位
#ifdef MASTER
    SPCTL = SPEN | MSTR;        //主机模式
#else
    SPCTL = SPEN;               //从机模式
#endif
}
//////////////////////////////////////////////////////////
void SendUart(BYTE dat)
{
    while (!TI);                //等待发送完成
    TI = 0;                     //清除发送标志
    SBUF = dat;                 //发送串口数据
}
//////////////////////////////////////////////////////////
BYTE RecvUart()
{
    while (!RI);                //等待串口数据接收完成
    RI = 0;                     //清除接收标志
    return SBUF;                //返回串口数据
}
//////////////////////////////////////////////////////////
BYTE SPISwap(BYTE dat)
{
#ifdef MASTER
    SPISS = 0;                  //拉低从机的 SS
#endif
    SPDAT = dat;                //触发 SPI 发送数据
    while (!(SPSTAT & SPIF));   //等待发送完成
    SPSTAT = SPIF | WCOL;       //清除 SPI 状态位
#ifdef MASTER
    SPISS = 1;                  //拉高从机的 SS
#endif
    return SPDAT;               //返回 SPI 数据
}
```

2. 汇编程序

```
/*----演示 STC 1T 系列单片机 SPI 功能(适用单主单从，查询方式)----*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可------------*/
//假定测试芯片的工作频率为 18.432MHz
//#define MASTER                //define:master undefine:slave
```

```
#define     URMD   0        ;0:使用定时器 2 作为波特率发生器
                            ;1:使用定时器 1 的模式 0(16 位自动重载模式)作为波特率发生器
                            ;2:使用定时器 1 的模式 2(8 位自动重载模式)作为波特率发生器

T2H         DATA    0D6H    ;定时器 2 高 8 位
T2L         DATA    0D7H    ;定时器 2 低 8 位

AUXR        DATA    08EH    ;Auxiliary register
SPSTAT      DATA    0CDH    ;SPI status register
SPIF        EQU     080H    ;SPSTAT.7
WCOL        EQU     040H    ;SPSTAT.6
SPCTL       DATA    0CEH    ;SPI control register
SSIG        EQU     080H    ;SPCTL.7
SPEN        EQU     040H    ;SPCTL.6
DORD        EQU     020H    ;SPCTL.5
MSTR        EQU     010H    ;SPCTL.4
CPOL        EQU     008H    ;SPCTL.3
CPHA        EQU     004H    ;SPCTL.2

SPDHH       EQU     000H    ;CPU_CLK/4
SPDH        EQU     001H    ;CPU_CLK/8
SPDL        EQU     002H    ;CPU_CLK/16
SPDLL       EQU     003H    ;CPU_CLK/32
SPDAT       DATA    0CFH    ;SPI data register
SPISS       BIT     P1.1    ;SPI 从机选择口, 连接到其它 MCU 的 SS 口
                            ;当 SPI 为一主多从模式时,
                            ;请使用主机的普通 IO 口连接到从机的 SS 口

;////////////////////////////////////////////////////////
    ORG     0000H
    LJMP    RESET

    ORG     0100H

RESET:
    LCALL   INIT_UART       ;初始化串口
    LCALL   INIT_SPI        ;初始化 SPI
START:
#ifdef      MASTER          ;对于主机(接收串口数据  并发送给从机,同时
    LCALL   RECV_UART       ; 从从即接收 SPI 数据并回传给 PC)
    LCALL   SPI_SWAP
    LCALL   SEND_UART
#else                       ;对于从机(从主机接收 SPI 数据,同时
    LCALL   SPI_SWAP        ; 发送前一个 SPI 数据给主机)
#endif
    SJMP    START
```

```
;/////////////////////////////////////////////////////////
INIT_UART:
    MOV     SCON, #5AH              ;设置串口为 8 位可变波特率
#if         URMD == 0
    MOV     T2L, #0D8H              ;设置波特率重装值(65536-18432000/4/115200)
    MOV     T2H, #0FFH
    MOV     AUXR, #14H              ;T2 为 1T 模式, 并启动定时器 2
    ORL     AUXR, #01H              ;选择定时器 2 为串口 1 的波特率发生器
#elif       URMD == 1
    MOV     AUXR, #40H              ;定时器 1 为 1T 模式
    MOV     TMOD, #00H              ;定时器 1 为模式 0(16 位自动重载)
    MOV     TL1, #0D8H              ;设置波特率重装值(65536-18432000/4/115200)
    MOV     TH1, #0FFH
    SETB    TR1                     ;定时器 1 开始运行
#else
    MOV     TMOD, #20H              ;设置定时器 1 为 8 位自动重装载模式
    MOV     AUXR, #40H              ;定时器 1 为 1T 模式
    MOV     TL1, #0FBH              ;115200 bps(256 - 18432000/32/115200)
    MOV     TH1, #0FBH
    SETB    TR1
#endif
    RET
;/////////////////////////////////////////////////////////
INIT_SPI:
    MOV     SPDAT, #0               ;初始化 SPI 数据
    MOV     SPSTAT, #SPIF | WCOL    ;清除 SPI 状态位
#ifdef      MASTER
    MOV     SPCTL, #SPEN | MSTR     ;主机模式
#else
    MOV     SPCTL, #SPEN            ;从机模式
#endif
    RET
;/////////////////////////////////////////////////////////
SEND_UART:
    JNB     TI, $                   ;等待发送完成
    CLR     TI                      ;清除发送标志
    MOV     SBUF, A                 ;发送串口数据
    RET
;/////////////////////////////////////////////////////////
RECV_UART:
    JNB     RI, $                   ;等待串口数据接收完成
    CLR     RI                      ;清除接收标志
    MOV     A, SBUF                 ;返回串口数据
    RET
;/////////////////////////////////////////////////////////
SPI_SWAP:
```

```
#ifdef      MASTER
    CLR     SPISS                   ;拉低从机的 SS
#endif
    MOV     SPDAT, A                ;触发 SPI 发送数据
WAIT:
    MOV     A, SPSTAT
    JNB     ACC.7, WAIT             ;等待发送完成
    MOV     SPSTAT, #SPIF | WCOL    ;清除 SPI 状态位
#ifdef      MASTER
    SETB    SPISS                   ;拉高从机的 SS
#endif
    MOV     A, SPDAT                ;返回 SPI 数据
    RET
;//////////////////////////////////////////////////////
END
```

26.5 适用互为主从系统的 SPI 功能测试程序(C 和汇编)

26.5.1 中断方式

1. C 程序

```
/*----演示 STC 1T 系列单片机 SPI 功能(适用互为主从系统，中断方式)---------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可----*/
//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"

#define     FOSC            18432000L
#define     BAUD            (256 - FOSC / 32 / 115200)

typedef     unsigned char   BYTE;
typedef     unsigned int    WORD;
typedef     unsigned long   DWORD;

#define     URMD            0       //0:使用定时器 2 作为波特率发生器
                                    //1:使用定时器 1 的模式 0(16 位自动重载模式)作为波特率发生器
                                    //2:使用定时器 1 的模式 2(8 位自动重载模式)作为波特率发生器

sfr         T2H = 0xd6;             //定时器 2 高 8 位
sfr         T2L = 0xd7;             //定时器 2 低 8 位
sfr         AUXR = 0x8e;            //辅助寄存器
sfr         SPSTAT = 0xcd;          //SPI 状态寄存器

#define     SPIF        0x80        //SPSTAT.7
```

```
#define     WCOL        0x40        //SPSTAT.6

sfr         SPCTL = 0xce;           //SPI 控制寄存器

#define     SSIG        0x80        //SPCTL.7
#define     SPEN        0x40        //SPCTL.6
#define     DORD        0x20        //SPCTL.5
#define     MSTR        0x10        //SPCTL.4
#define     CPOL        0x08        //SPCTL.3
#define     CPHA        0x04        //SPCTL.2
#define     SPDHH       0x00        //CPU_CLK/4
#define     SPDH        0x01        //CPU_CLK/8
#define     SPDL        0x02        //CPU_CLK/16
#define     SPDLL       0x03        //CPU_CLK/32

sfr         SPDAT = 0xcf;           //SPI 数据寄存器

sbit        SPISS = P1^1;           //SPI 从机选择口, 连接到其它 MCU 的 SS 口
                                    //当 SPI 为一主多从模式时,
                                    //请使用主机的普通 IO 口连接到从机的 SS 口

sfr         IE2 = 0xAF;             //中断控制寄存器 2

#define     ESPI        0x02        //IE2.1

void InitUart();
void InitSPI();
void SendUart(BYTE dat);            //发送数据到 PC

BYTE    RecvUart();                 //从 PC 接收数据
bit     MSSEL;                      //1: master 0:slave
//////////////////////////////////////////////////////////
void main()
{
    InitUart();                     //初始化串口
    InitSPI();                      //初始化 SPI
    IE2 |= ESPI;
    EA = 1;
    while (1)
    {
        if (RI)
        {
            SPCTL = SPEN | MSTR;    //设置为主机模式
            MSSEL = 1;
            ACC = RecvUart();
            SPISS = 0;              //拉低从机的 SS
```

```
            SPDAT = ACC;                //触发 SPI 发送数据
        }
    }
}
//////////////////////////////////////////////////////////
void spi_isr() interrupt 9 using 1      //SPI 中断服务程序  9 (004BH)
{
    SPSTAT = SPIF | WCOL;               //清除 SPI 状态位
    if (MSSEL)
    {
        SPCTL = SPEN;                   //重置为从机模式
        MSSEL = 0;
        SPISS = 1;                      //拉高从机的 SS
        SendUart(SPDAT);                //返回 SPI 数据
    }
    else
    {                                   //对于从机(从主机接收 SPI 数据,同时
        SPDAT = SPDAT;                  //发送前一个 SPI 数据给主机)
    }
}
//////////////////////////////////////////////////////////
void InitUart()
{
    SCON = 0x5a;                        //设置串口为 8 位可变波特率
#if URMD == 0
    T2L = 0xd8;                         //设置波特率重装值
    T2H = 0xff;                         //115200 bps(65536-18432000/4/115200)
    AUXR = 0x14;                        //T2 为 1T 模式, 并启动定时器 2
    AUXR |= 0x01;                       //选择定时器 2 为串口 1 的波特率发生器
#elif URMD == 1
    AUXR = 0x40;                        //定时器 1 为 1T 模式
    TMOD = 0x00;                        //定时器 1 为模式 0(16 位自动重载)
    TL1 = 0xd8;                         //设置波特率重装值
    TH1 = 0xff;                         //115200 bps(65536-18432000/4/115200)
    TR1 = 1;                            //定时器 1 开始启动
#else
    TMOD = 0x20;                        //设置定时器 1 为 8 位自动重装载模式
    AUXR = 0x40;                        //定时器 1 为 1T 模式
    TH1 = TL1 = 0xfb;                   //115200 bps(256 - 18432000/32/115200)
    TR1 = 1;
#endif
}
//////////////////////////////////////////////////////////
void InitSPI()
{
    SPDAT = 0;                          //初始化 SPI 数据
```

```
    SPSTAT = SPIF | WCOL;                   //清除 SPI 状态位
    SPCTL = SPEN;                           //从机模式
}
//////////////////////////////////////////////////////////
void SendUart(BYTE dat)
{
    while (!TI);                            //等待发送完成
    TI = 0;                                 //清除发送标志
    SBUF = dat;                             //发送串口数据
}
//////////////////////////////////////////////////////////
BYTE RecvUart()
{
    while (!RI);                            //等待串口数据接收完成
    RI = 0;                                 //清除接收标志
    return SBUF;                            //返回串口数据
    }
```

2. 汇编程序

```
/*----演示 STC 1T 系列单片机 SPI 功能(适用互为主从系统，中断方式)--------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可----*/
//假定测试芯片的工作频率为 18.432MHz
#define   URMD 0                ;0:使用定时器 2 作为波特率发生器
                                ;1:使用定时器 1 的模式 0(16 位自动重载模式)作为波特率发生器
                                ;2:使用定时器 1 的模式 2(8 位自动重载模式)作为波特率发生器
T2H         DATA    0D6H        ;定时器 2 高 8 位
T2L         DATA    0D7H        ;定时器 2 低 8 位

AUXR        DATA    08EH        ;辅助寄存器
SPSTAT      DATA    0CDH        ;SPI 状态寄存器
SPIF        EQU     080H        ;SPSTAT.7
WCOL        EQU     040H        ;SPSTAT.6
SPCTL       DATA    0CEH        ;SPI 控制寄存器
SSIG        EQU     080H        ;SPCTL.7
SPEN        EQU     040H        ;SPCTL.6
DORD        EQU     020H        ;SPCTL.5
MSTR        EQU     010H        ;SPCTL.4
CPOL        EQU     008H        ;SPCTL.3
CPHA        EQU     004H        ;SPCTL.2

SPDHH       EQU     000H        ;CPU_CLK/4
SPDH        EQU     001H        ;CPU_CLK/8
SPDL        EQU     002H        ;CPU_CLK/16
SPDLL       EQU     003H        ;CPU_CLK/32
SPDAT       DATA    0CFH        ;SPI 数据寄存器
```

```
SPISS       BIT     P1.1        ;SPI 从机选择口, 连接到其它 MCU 的 SS 口
                                ;当 SPI 为一主多从模式时,请使用主机的普通 IO 口连接到从机的 SS 口
IE2         EQU     0AFH        ;中断控制寄存器 2
ESPI        EQU     02H         ;IE2.1
MSSEL       BIT     20H.0       ;1: master 0:slave
;/////////////////////////////////////////////////////////
    ORG     0000H
    LJMP    RESET
    ORG     004BH                       ;SPI 中断服务程序

SPI_ISR:
    PUSH    ACC
    PUSH    PSW
    MOV     SPSTAT, #SPIF | WCOL        ;清除 SPI 状态位
    JBC     MSSEL, MASTER_SEND

SLAVE_RECV:                             ;对于从机(从主机接收 SPI 数据,同时
    MOV     SPDAT, SPDAT                ; 发送前一个 SPI 数据给主机)
    JMP     SPI_EXIT

MASTER_SEND:
    SETB    SPISS                       ;拉高从机的 SS
    MOV     SPCTL, #SPEN                ;重置为从机模式
    MOV     A, SPDAT                    ;返回 SPI 数据
    LCALL   SEND_UART

SPI_EXIT:
    POP     PSW
    POP     ACC
    RETI
;/////////////////////////////////////////////////////////
    ORG     0100H
RESET:
    MOV     SP, #3FH
    LCALL   INIT_UART                   ;初始化串口
    LCALL   INIT_SPI                    ;初始化 SPI
    ORL     IE2, #ESPI
    SETB    EA

START:
    JNB     RI, $                       ;等待串口数据
    MOV     SPCTL, #SPEN | MSTR         ;设置为主机模式
    SETB    MSSEL
    LCALL   RECV_UART                   ;接收串口数据
    CLR     SPISS                       ;拉低从机的 SS
    MOV     SPDAT, A                    ;触发 SPI 发送数据
```

```
    SJMP    START
;/////////////////////////////////////////////////////////
INIT_UART:
    MOV     SCON, #5AH              ;设置串口为 8 位可变波特率
#if URMD == 0
    MOV     T2L, #0D8H              ;设置波特率重装值(65536-18432000/4/115200)
    MOV     T2H, #0FFH
    MOV     AUXR, #14H              ;T2 为 1T 模式, 并启动定时器 2
    ORL     AUXR, #01H              ;选择定时器 2 为串口 1 的波特率发生器
#elif URMD == 1
    MOV     AUXR, #40H              ;定时器 1 为 1T 模式
    MOV     TMOD, #00H              ;定时器 1 为模式 0(16 位自动重载)
    MOV     TL1, #0D8H              ;设置波特率重装值(65536-18432000/4/115200)
    MOV     TH1, #0FFH
    SETB    TR1                     ;定时器 1 开始运行
#else
    MOV     TMOD, #20H              ;设置定时器 1 为 8 位自动重装载模式
    MOV     AUXR, #40H              ;定时器 1 为 1T 模式
    MOV     TL1, #0FBH              ;115200 bps(256 - 18432000/32/115200)
    MOV     TH1, #0FBH
    SETB    TR1
#endif
    RET
;/////////////////////////////////////////////////////////
INIT_SPI:
    MOV     SPDAT, #0               ;初始化 SPI 数据
    MOV     SPSTAT, #SPIF | WCOL    ;清除 SPI 状态位
    MOV     SPCTL, #SPEN            ;从机模式
    RET
;/////////////////////////////////////////////////////////
SEND_UART:
    JNB     TI, $                   ;等待发送完成
    CLR     TI                      ;清除发送标志
    MOV     SBUF, A                 ;发送串口数据
    RET
;/////////////////////////////////////////////////////////
RECV_UART:
    JNB     RI, $                   ;等待串口数据接收完成
    CLR     RI                      ;清除接收标志
    MOV     A, SBUF                 ;返回串口数据
    RET
;/////////////////////////////////////////////////////////
END
```

26.5.2 查询方式

1. C 程序

```
/*----演示 STC 1T 系列单片机 SPI 功能(适用互为主从系统，查询方式)---------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可----*/
//假定测试芯片的工作频率为 18.432MHz
#include "reg51.h"

#define     FOSC            18432000L
#define     BAUD            (256 - FOSC / 32 / 115200)

typedef     unsigned char   BYTE;
typedef     unsigned int    WORD;
typedef     unsigned long   DWORD;

#define     URMD 0              //0:使用定时器 2 作为波特率发生器
                                //1:使用定时器 1 的模式 0(16 位自动重载模式)作为波特率发生器
                                //2:使用定时器 1 的模式 2(8 位自动重载模式)作为波特率发生器

sfr         T2H = 0xd6;         //定时器 2 高 8 位
sfr         T2L = 0xd7;         //定时器 2 低 8 位
sfr         AUXR = 0x8e;        //辅助寄存器
sfr         SPSTAT = 0xcd;      //SPI 状态寄存器

#define     SPIF        0x80    //SPSTAT.7
#define     WCOL        0x40    //SPSTAT.6

sfr         SPCTL = 0xce;       //SPI 控制寄存器

#define     SSIG        0x80    //SPCTL.7
#define     SPEN        0x40    //SPCTL.6
#define     DORD        0x20    //SPCTL.5
#define     MSTR        0x10    //SPCTL.4
#define     CPOL        0x08    //SPCTL.3
#define     CPHA        0x04    //SPCTL.2
#define     SPDHH       0x00    //CPU_CLK/4
#define     SPDH        0x01    //CPU_CLK/8
#define     SPDL        0x02    //CPU_CLK/16
#define     SPDLL       0x03    //CPU_CLK/32

sfr         SPDAT = 0xcf;       //SPI 数据寄存器
sbit        SPISS = P1^1;       //SPI 从机选择口，连接到其它 MCU 的 SS 口
                                //当 SPI 为一主多从模式时，
                                //请使用主机的普通 IO 口连接到从机的 SS 口
```

```
void InitUart();
void InitSPI();
void SendUart(BYTE dat);              //发送数据到 PC

BYTE RecvUart();                      //从 PC 接收数据
BYTE SPISwap(BYTE dat);               //主机与从机之间交换数据
//////////////////////////////////////////////////////////
void main()
{
    InitUart();                               //初始化串口
    InitSPI();                                //初始化 SPI
    while (1)
    {
        if (RI)
        {
            SPCTL = SPEN | MSTR;              //设置为主机模式
            SendUart(SPISwap(RecvUart()));
            SPCTL = SPEN;                     //重置为从机模式
        }
        if (SPSTAT & SPIF)
        {
            SPSTAT = SPIF | WCOL;             //清除 SPI 状态位
            SPDAT = SPDAT;                    //数据从接收缓冲区移到发送缓冲区
        }
    }
}
//////////////////////////////////////////////////////////
void InitUart()
{
    SCON = 0x5a;                              //设置串口为 8 位可变波特率
#if URMD == 0
    T2L = 0xd8;                               //设置波特率重装值
    T2H = 0xff;                               //115200 bps(65536-18432000/4/115200)
    AUXR = 0x14;                              //T2 为 1T 模式, 并启动定时器 2
    AUXR |= 0x01;                             //选择定时器 2 为串口 1 的波特率发生器
#elif URMD == 1
    AUXR = 0x40;                              //定时器 1 为 1T 模式
    TMOD = 0x00;                              //定时器 1 为模式 0(16 位自动重载)
    TL1 = 0xd8;                               //设置波特率重装值
    TH1 = 0xff;                               //115200 bps(65536-18432000/4/115200)
    TR1 = 1;                                  //定时器 1 开始启动
#else
    TMOD = 0x20;                              //设置定时器 1 为 8 位自动重装载模式
    AUXR = 0x40;                              //定时器 1 为 1T 模式
    TH1 = TL1 = 0xfb;                         //115200 bps(256 - 18432000/32/115200)
    TR1 = 1;
```

```
#endif
}
//////////////////////////////////////////////////////
void InitSPI()
{
    SPDAT = 0;                          //初始化 SPI 数据
    SPSTAT = SPIF | WCOL;               //清除 SPI 状态位
    SPCTL = SPEN;                       //从机模式
}
//////////////////////////////////////////////////////
void SendUart(BYTE dat)
{
    while (!TI);                        //等待发送完成
    TI = 0;                             //清除发送标志
    SBUF = dat;                         //发送串口数据
}
//////////////////////////////////////////////////////
BYTE RecvUart()
{
    while (!RI);                        //等待串口数据接收完成
    RI = 0;                             //清除接收标志
    return SBUF;                        //返回串口数据
}
//////////////////////////////////////////////////////
BYTE SPISwap(BYTE dat)
{
    SPISS = 0;                          //拉低从机的 SS
    SPDAT = dat;                        //触发 SPI 发送数据
    while (!(SPSTAT & SPIF));           //等待发送完成
    SPSTAT = SPIF | WCOL;               //清除 SPI 状态位
    SPISS = 1;                          //拉高从机的 SS
    return SPDAT;                       //返回 SPI 数据
}
```

2. 汇编程序

```
/*----演示 STC 1T 系列单片机 SPI 功能(适用互为主从系统，查询方式)-------*/
/*----在 Keil C 开发环境中选择 Intel 8052 编译，头文件包含<reg51.h>即可--*/
//假定测试芯片的工作频率为 18.432MHz

#define   URMD 0                    ;0:使用定时器 2 作为波特率发生器
                                    ;1:使用定时器 1 的模式 0(16 位自动重载模式)作为波特率发生器
                                    ;2:使用定时器 1 的模式 2(8 位自动重载模式)作为波特率发生器

T2H         DATA    0D6H            ;定时器 2 高 8 位
```

```
T2L         DATA    0D7H        ;定时器 2 低 8 位

AUXR        DATA    08EH        ;辅助寄存器

SPSTAT      DATA    0CDH        ;SPI 状态寄存器
SPIF        EQU     080H        ;SPSTAT.7
WCOL        EQU     040H        ;SPSTAT.6
SPCTL       DATA    0CEH        ;SPI 控制寄存器

SSIG        EQU     080H        ;SPCTL.7
SPEN        EQU     040H        ;SPCTL.6
DORD        EQU     020H        ;SPCTL.5
MSTR        EQU     010H        ;SPCTL.4
CPOL        EQU     008H        ;SPCTL.3
CPHA        EQU     004H        ;SPCTL.2

SPDHH       EQU     000H        ;CPU_CLK/4
SPDH        EQU     001H        ;CPU_CLK/8
SPDL        EQU     002H        ;CPU_CLK/16
SPDLL       EQU     003H        ;CPU_CLK/32

SPDAT       DATA    0CFH        ;SPI 数据寄存器
SPISS       BIT     P1.1        ;SPI 从机选择口, 连接到其它 MCU 的 SS 口
                                ;当 SPI 为一主多从模式时,请使用主机的普通 IO 口连接到从机的 SS 口
;////////////////////////////////////////////////////////
    ORG     0000H

    LJMP    RESET
    ORG     0100H

RESET:
    LCALL   INIT_UART               ;初始化串口
    LCALL   INIT_SPI                ;初始化 SPI

START:
    JB  RI, MASTER_MODE

SLAVE_MODE:
    MOV     A, SPSTAT
    JNB     ACC.7, START
    MOV     SPSTAT, #SPIF | WCOL    ;清除 SPI 状态位
    MOV     SPDAT, SPDAT            ;返回 SPI 数据
    SJMP    START

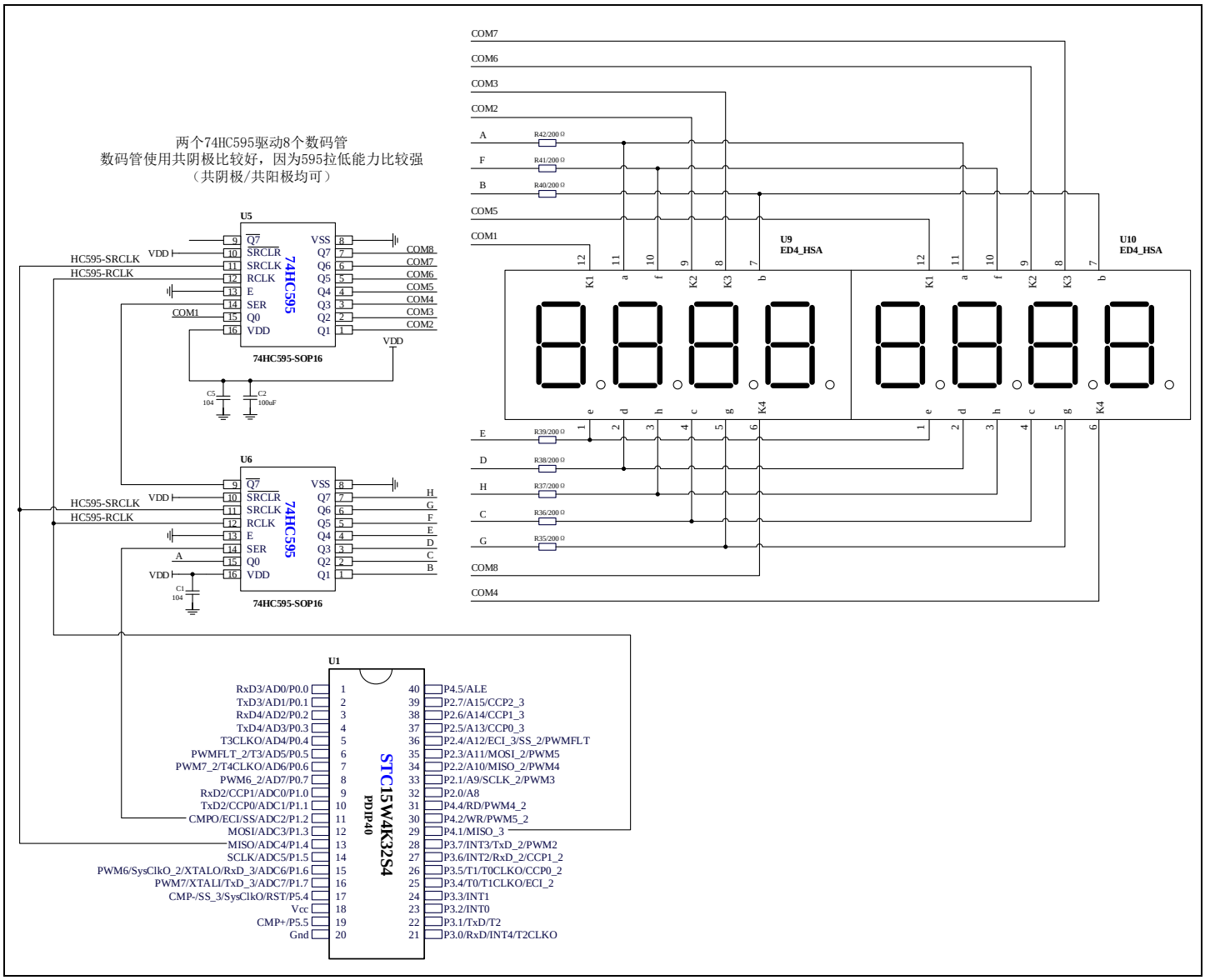
MASTER_MODE:
    MOV     SPCTL, #SPEN | MSTR     ;设置为主机模式
```

```
    LCALL   RECV_UART           ;接收串口数据
    LCALL   SPI_SWAP            ;发送串口数据给从机,同时从从机接收 SPI 数据
    LCALL   SEND_UART           ;发送 SPI 数据到 PC
    MOV     SPCTL, #SPEN;       ;重置为从机模式
    SJMP    START
;/////////////////////////////////////////////////////////
INIT_UART:
    MOV     SCON, #5AH          ;设置串口为 8 位可变波特率
#if URMD == 0
    MOV     T2L, #0D8H          ;设置波特率重装值(65536-18432000/4/115200)
    MOV     T2H, #0FFH
    MOV     AUXR, #14H          ;T2 为 1T 模式, 并启动定时器 2
    ORL     AUXR, #01H          ;选择定时器 2 为串口 1 的波特率发生器
#elif URMD == 1
    MOV     AUXR, #40H          ;定时器 1 为 1T 模式
    MOV     TMOD, #00H          ;定时器 1 为模式 0(16 位自动重载)
    MOV     TL1, #0D8H          ;设置波特率重装值(65536-18432000/4/115200)
    MOV     TH1, #0FFH
    SETB    TR1                 ;定时器 1 开始运行
#else
    MOV     TMOD, #20H          ;设置定时器 1 为 8 位自动重装载模式
    MOV     AUXR, #40H          ;定时器 1 为 1T 模式
    MOV     TL1, #0FBH          ;115200 bps(256 - 18432000/32/115200)
    MOV     TH1, #0FBH
    SETB    TR1
#endif
RET
;/////////////////////////////////////////////////////////
INIT_SPI:
    MOV     SPDAT, #0           ;初始化 SPI 数据
    MOV     SPSTAT, #SPIF | WCOL ;清除 SPI 状态位
    MOV     SPCTL, #SPEN        ;从机模式
    RET
;/////////////////////////////////////////////////////////
SEND_UART:
    JNB     TI, $               ;等待发送完成
    CLR     TI                  ;清除发送标志
    MOV     SBUF, A             ;发送串口数据
    RET
;/////////////////////////////////////////////////////////
RECV_UART:
    JNB     RI, $               ;等待串口数据接收完成
    CLR     RI                  ;清除接收标志
    MOV     A, SBUF             ;返回串口数据
    RET
;/////////////////////////////////////////////////////////
```

```
SPI_SWAP:
    CLR     SPISS                   ;拉低从机的 SS
    MOV     SPDAT, A                ;触发 SPI 发送数据

WAIT:
    MOV     A,   SPSTAT
    JNB     ACC.7, WAIT             ;等待发送完成
    MOV     SPSTAT, #SPIF | WCOL    ;清除 SPI 状态位
    SETB    SPISS                   ;拉高从机的 SS
    MOV     A, SPDAT                ;返回 SPI 数据
    RET
;////////////////////////////////////////////////////////
END
```

26.6 利用 SPI 控制 74HC595 驱动 8 位数码管及测试程序(C 和汇编)



下面是用 STC 的 MCU 的 SPI 方式控制 74HC595 驱动 8 位数码管(串口扩展，3 根线)的测试程序：

1. C 程序

```
/*---------------------------------------------------*/
/*----STC 1T Series MCU Programme Demo----*/
/*---------------------------------------------------*/

/************** 本程序功能说明 ***************
用 STC 的 MCU 的 SPI 方式控制 74HC595 驱动 8 位数码管。
用户可以修改宏来选择时钟频率, 可以修改寄存器定义是 STC12C5A60S2 系列还是 STC12C5628AD、
STC12C5410AD、STC12C4052AD 系列.
用户可以在显示函数里修改成共阴或共阳，推荐尽量使用共阴数码管.
显示效果为: 8 个数码管循环显示 0,1,2...,A,B..F,消隐.
*****************************************/

#include "reg52.h"

/******************************** 用户定义宏 ***********************************/
#define     MAIN_Fosc   11059200UL      //定义主时钟
//#define     MAIN_Fosc   22118400UL      //定义时钟
/*****************************************************************************/

sfr         SPSTAT = 0xCD;              //STC12C5A60S2 系列
sfr         SPCTL = 0xCE;               //STC12C5A60S2 系列
sfr         SPDAT = 0xCF;               //STC12C5A60S2 系列

/*
sfr         SPSTAT = 0x84;              //STC12C5628AD STC12C5410AD STC12C4052AD 系列
sfr         SPCTL = 0x85;               //STC12C5628AD STC12C5410AD STC12C4052AD 系列
sfr         SPDAT = 0x86;               //STC12C5628AD STC12C5410AD STC12C4052AD 系列
*/

/********************* 下面的宏自动生成, 用户不可修改 ***************************/
#define Timer0_Reload (MAIN_Fosc / 12000)
/*****************************************************************************/

//SPCTL     SPI 控制寄存器
// 7    6       5       4       3       2       1       0       Reset Value
//SSIG  SPEN    DORD    MSTR    CPOL    CPHA    SPR1    SPR0    0x00

#define SSIG        1               //1: 忽略 SS 脚，由 MSTR 位决定主机还是从机
                                    //0: SS 脚用于决定主从机。
#define SPEN        1               //1: 允许 SPI，
                                    //0：禁止 SPI，所有 SPI 管脚均为普通 IO
#define DORD        0               //1： LSB 先发，
                                    //0：MSB 先发
```

```
#define   MSTR      1                   //1：设为主机
                                        //0：设为从机
#define   CPOL      1                   //1: 空闲时 SCLK 为高电平，
                                        //0：空闲时 SCLK 为低电平
#define   CPHA      1
#define   SPR1      0                   //SPR1,SPR0  00: fosc/4,    01: fosc/16
#define   SPR0      0                   //           10: fosc/64,   11: fosc/128
#define   SPEED_4   0                   // fosc/4
#define   SPEED_16  1                   // fosc/16
#define   SPEED_64  2                   // fosc/64
#define   SPEED_128 3                   // fosc/128

//SPSTATSPI 状态寄存器
// 7      6      5   4   3   2   1   0   Reset Value
// SPIF   WCOL   -   -   -   -   -   -
#define   SPIF      0x80                //SPI 传输完成标志。写入 1 清 0。
#define   WCOL      0x40                //SPI 写冲突标志。写入 1 清 0。

/************* 本地常量声明 **************/
unsigned char code t_display[]={
// 0     1   2   3   4   5   6    7     8   9   A   B   C   D     E     F   消隐
0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x77,0x7C,0x39,0x5E,0x79,0x71,0x00};
//段码

unsigned char code T_COM[]={0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};      //位码
/************* 本地变量声明 **************/
sbit      SPI_SCL = P1^5;               //SPI 同步时钟  P_HC595_SRCLK pin 11 SRCLK Shift data clock
//sbit    SPI_MISO = P1^6;              //SPI 同步数据输入  本例不用
sbit      SPI_MOSI = P1^3;              //SPI 同步数据输出  P_HC595_SER pin 14 SER data input
sbit      P_HC595_RCLK = P4^1;          //SPI 片选(任意 IO) pin 12 RCLk store (latch) clock

unsigned      char      LED8[8];        //显示缓冲
unsigned      char      display_index;  //显示位索引
bit           B_1ms;                    //1ms 标志
/*********************************************/
void main(void)                         //主函数
{
    unsigned char i,k;
    unsigned int j;
    SPCTL = (SSIG << 7) + (SPEN << 6) + (DORD << 5) + (MSTR << 4) + (CPOL << 3) + (CPHA << 2)+ SPEED_4;
                                        //配置 SPI

    TMOD = 0x01;                        //Timer 0 config as 16bit timer, 12T
    TH0 = (65536 - Timer0_Reload) / 256;
    TL0 = (65536 - Timer0_Reload) % 256;
    ET0 = 1;
```

```
    TR0 = 1;
    EA = 1;

    for(i=0; i<8; i++) LED8[i] = 0x10;          //上电消隐
    j = 0;
    k = 0;

    while(1)
    {
        while(!B_1ms) ;                         //等待 1ms 到
        B_1ms = 0;
        if(++j >= 500)                          //500ms 到
        {
            j = 0;
            for(i=0; i<8; i++) LED8[i] = k;     //刷新显示
            if(++k > 0x10) k = 0;               //8 个数码管循环显示 0,1,2...,A,B..F,消隐.
        }
    }
}
/*********************************************/
void SPI_SendByte(unsigned char dat)            //SPI 发送一个字节
{
    SPSTAT = SPIF + WCOL;                       //清 0 SPIF 和 WCOL 标志
    SPDAT = dat;                                //发送一个字节
    while((SPSTAT & SPIF) == 0) ;               //等待发送完成
    SPSTAT = SPIF + WCOL;                       //清 0 SPIF 和 WCOL 标志
}
/*********************************************/
void DisplayScan(void)                                  //显示扫描函数
{
//  SPI_SendByte(~T_COM[display_index]);                //共阴  输出位码
//  SPI_SendByte(t_display[LED8[display_index]]);       //共阴  输出段码
    SPI_SendByte(T_COM[display_index]);                 //共阳  输出位码
    SPI_SendByte(~t_display[LED8[display_index]]);      //共阳  输出段码
    P_HC595_RCLK = 1;
    P_HC595_RCLK = 0;                                   //锁存输出数据
    if(++display_index >= 8) display_index = 0;         //8 位结束回 0
}
/*********************************************/
void timer0 (void) interrupt 1                          //Timer0 1ms 中断函数
{
    TH0 = (65536 - Timer0_Reload) / 256;                //重装定时值
    TL0 = (65536 - Timer0_Reload) % 256;
    DisplayScan();                                      //1ms 扫描显示一位
    B_1ms = 1;                                          //1ms 标志
}
```

2. 汇编程序

```
;/*---------------------------------------------------*/
;/*----STC 1T Series MCU Programme Demo----*/
;/*---------------------------------------------------*/

;/************* 本程序功能说明 **************
;用 STC 的 MCU 的 SPI 方式控制 74HC595 驱动 8 位数码管。
;用户可以修改宏来选择时钟频率, 可以修改寄存器定义是 STC12C5A60S2 系列还是 STC12C5628AD、
STC12C5410AD、STC12C4052AD 系列.
;用户可以在显示函数里修改成共阴或共阳.推荐尽量使用共阴数码管.
;显示效果为: 8 个数码管循环显示 0,1,2...,A,B..F,消隐.
;*******************************************/

;定义 Timer0 1ms 重装值
D_Timer0_Reload     EQU     (0-921)         ;1ms for 11.0592MHZ
;D_Timer0_Reload    EQU     (0-1832)        ;1ms for 22.1184MHZ

SPSTAT              DATA    0CDH            ;STC12C5A60S2 系列
SPCTL               DATA    0CEH            ;STC12C5A60S2 系列
SPDAT               DATA    0CFH            ;STC12C5A60S2 系列

;SPSTAT             DATA    084H            ;STC12C5628AD STC12C5410AD STC12C4052AD 系列
;SPCTL              DATA    085H            ;STC12C5628AD STC12C5410AD STC12C4052AD 系列
;SPDAT              DATA    086H            ;STC12C5628AD STC12C5410AD STC12C4052AD 系列

;/************* 本地变量声明 **************/
SPI_SCL             BIT     P1^5    ;SPI 同步时钟  P_HC595_SRCLK pin11 SRCLK Shift data clock
;SPI_MISO           BIT     P1^6    ;SPI 同步数据输入  本例不用
SPI_MOSI            BIT     P1^3    ;SPI 同步数据输出  P_HC595_SER pin 14 SER data input
P_HC595_RCLK        BIT     P4^1    ;SPI 片选(任意 IO) pin 12 RCLk store (latch) clock

LED8                EQU     030H
display_index       DATA    038H
FLAG0               DATA    20H
B_1ms               BIT     FLAG0.0

;SPCTL SPI 控制寄存器
; 7         6       5       4       3       2       1       0       Reset Value
; SSIG      SPEN    DORD    MSTR    CPOL    CPHA    SPR1    SPR0    0x00
SSIG                EQU     1               ;1: 忽略 SS 脚，由 MSTR 位决定主机还是从机
                                            ;0: SS 脚用于决定主从机。
SPEN                EQU     1               ;1: 允许 SPI，
                                            ;0：禁止 SPI，所有 SPI 管脚均为普通 IO
DORD                EQU     0               ;1： LSB 先发，
                                            ;0： MSB 先发
```

```
MSTR            EQU     1               ;1：设为主机
                                        ;0：设为从机
CPOL            EQU     1               ;1: 空闲时 SCLK 为高电平，
                                        ;0：空闲时 SCLK 为低电平
CPHA            EQU     1
SPR1            EQU     0               ;SPR1,SPR0 00: fosc/4, 01: fosc/16
SPR0            EQU     0               ; 10: fosc/64, 11: fosc/128
SPEED_4         EQU     0               ; fosc/4
SPEED_16        EQU     1               ; fosc/16
SPEED_64        EQU     2               ; fosc/64
SPEED_128       EQU     3               ; fosc/128

;SPSTAT SPI 状态寄存器
; 7       6       5   4   3   2   1   0   Reset Value
; SPIF    WCOL    -   -   -   -   -   -
SPIF            EQU     80H             ;SPI 传输完成标志。写入 1 清 0。
WCOL            EQU     40H             ;SPI 写冲突标志。写入 1 清 0。
;*******************************************************************
    ORG     00H                         ;reset
    LJMP    F_MAIN_FUNC

    ORG     03H                         ;INT0 interrupt
;   LJMP    F_INT0_interrupt
    RETI

    ORG     0BH                         ;Timer0 interrupt
    LJMP    F_Timer0_interrupt
    RETI

    ORG     13H                         ;INT1 interrupt
;   LJMP    F_INT1_interrupt

    ORG     1BH                         ;Timer1 interrupt
;   LJMP    F_Timer1_interrupt
    RETI
;*******************************************************************
F_MAIN_FUNC:
    MOV     SP, #50H

    MOV     SPCTL, #((SSIG SHL 7) + (SPEN SHL 6) + (DORD SHL 5) + (MSTR SHL 4))     ;配置 SPI
    ORL     SPCTL, #( (CPOL SHL 3) + (CPHA SHL 2) + SPEED_4)        ; 配置 SPI

    MOV     TMOD, #01H                  ;Timer 0 config as 16bit timer, 12T
    MOV     TH0, #HIGH D_Timer0_Reload  ;1ms
    MOV     TL0, #LOW D_Timer0_Reload
```

```
    SETB    ET0
    SETB    TR0
    SETB    EA

    MOV     R0, #LED8

L_InitLoop1:
    MOV     @R0, #10H                   ;上电消隐
    INC     R0
    MOV     A, R0
    CJNE    A, #(LED8+8), L_InitLoop1
    MOV     R2,  #HIGH 500              ;500ms

    MOV     R3, #LOW 500
    MOV     R4, #0

L_MainLoop:
    JNB     B_1ms, $                    ;等待 1ms 到
    CLR     B_1ms                       ;清 1ms 标志
    MOV     A, R3
    CLR     C
    SUBB    A, #1
    MOV     R3, A
    MOV     A, R2
    SUBB    A, #0
    MOV     R2, A
    ORL     A, R3
    JNZ     L_MainLoop

    MOV     R2, #HIGH 500               ;500ms
    MOV     R3, #LOW 500

    MOV     R0, #LED8                   ;刷新显示

L_OptionLoop1:
    MOV     A, R4
    MOV     @R0, A
    INC     R0
    MOV     A, R0
    CJNE    A, #(LED8+8), L_OptionLoop1
    INC     R4
    MOV     A, R4
    CJNE    A, #11H,L_MainLoop          ;8 个数码管循环显示 0,1,2...,A,B..F,消隐.
    MOV     R4, #0
    SJMP    L_MainLoop
;/*********************************************/
```

```
t_display:
;       0   1       2   3   4   5   6       7   8   9   A   B       C   D   E   F   消隐
    DB 03FH,006H,05BH,04FH,066H,06DH,07DH,007H,07FH,06FH,077H,07CH,039H,05EH,079H,071H,000H
                                                ;段码
T_COM:
    DB      01H,02H,04H,08H,10H,20H,40H,80H     ;位码
;/*********************************************/
F_SPI_SendByte:                                 ;SPI 发送一个字节
    MOV     SPSTAT, #(SPIF + WCOL)              ;清 0 SPIF 和 WCOL 标志
    MOV     SPDAT, A                            ;发送一个字节

L_SPI_SendByteWait:                             ;等待发送完成
    MOV     A, SPSTAT
    ANL     A, #SPIF
    JZ      L_SPI_SendByteWait
    MOV     SPSTAT, #(SPIF + WCOL)              ;清 0 SPIF 和 WCOL 标志
    RET
;/*********************************************/
F_DisplayScan:                                  ;显示扫描函数
    MOV     DPTR, #T_COM
    MOV     A, display_index
    MOVC    A, @A+DPTR
    ; CPL   A                                   ;共阴  共阳时注释掉本句
    LCALL   F_SPI_SendByte                      ;输出位码

    MOV     DPTR, #t_display
    MOV     A, #LED8
    ADD     A, display_index
    MOV     R0, A
    MOV     A, @R0
    MOVC    A, @A+DPTR
    CPL     A                                   ;共阳  共阴时注释掉本句
    LCALL   F_SPI_SendByte                      ;输出段码
    SETB    P_HC595_RCLK
    CLR     P_HC595_RCLK                        ;锁存输出数据
    INC     display_index
    MOV     A, display_index
    CJNE    A, #8, L_QuitDisplayScan
    MOV     display_index, #0                   ;8 位结束回 0

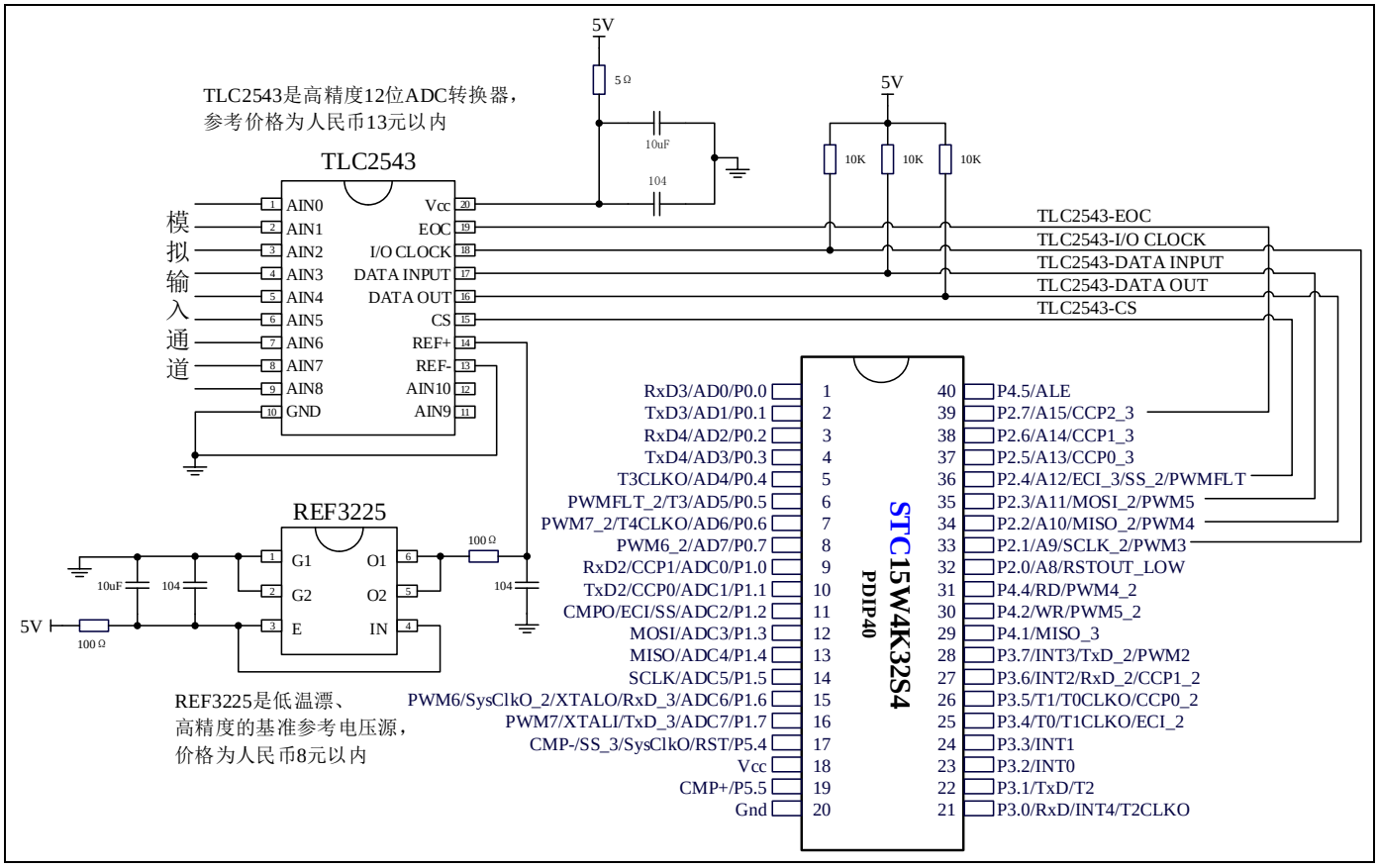
L_QuitDisplayScan:
    RET
;*********************************************************************
F_Timer0_interrupt:                             ;Timer0 1ms 中断函数
    PUSH    PSW                                 ;现场保护
    PUSH    ACC
```

```
    MOV     A, R0
    PUSH    ACC
    PUSH    DPH
    PUSH    DPL
    MOV     TH0, #HIGH D_Timer0_Reload      ;1ms  重装定时值
    MOV     TL0, #LOW D_Timer0_Reload
    LCALL   F_DisplayScan                   ;1ms 扫描显示一位
    SETB    B_1ms                           ;1ms 标志

L_QuitT0Interrupt:
    POP     DPL                             ;现场恢复
    POP     DPH
    POP     ACC
    MOV     R0, A
    POP     ACC
    POP     PSW
    RETI

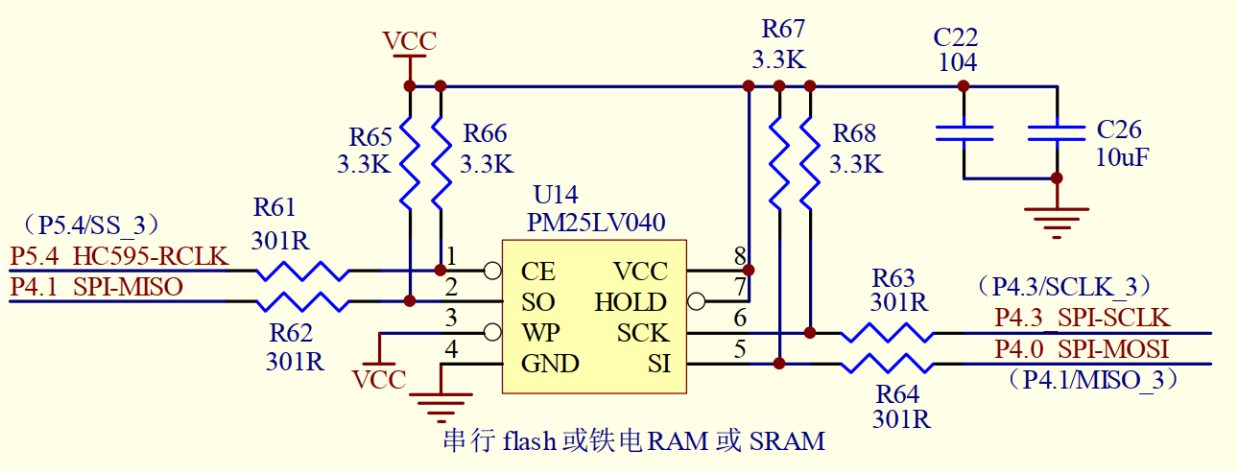
    END
```

26.7 利用 SPI 接口扩展 12 位 ADC(TLC2543)的应用线路图



26.8 利用 STC15 系列单片机 SPI 的主模式读写外部串行 Flash

26.8.1 利用 STC15 系列 SPI 的主模式读写外部串行 Flash 的参考电路图



26.8.2 利用 STC15 系列 SPI 的主模式读写外部串行 Flash 的测试程序

26.8.2.1 通过中断方式利用 SPI 的主模式读写外部串行 Flash 的测试程序(C 和汇编)

1、C 语言程序

```
/*---------------------------------------------------------------------------------------------------*/
/*----STC15W4K60S4 系列 SPI 的主模式读写外部串行 Flash 举例(中断方式)----*/
/*---------------------------------------------------------------------------------------------------*/

//本示例在 Keil 开发环境下请选择 Intel 的 8058 芯片型号进行编译
//假定测试芯片的工作频率为 18.432MHz
//本示例所读写的目标 Flash 为 PM25LV040,本代码已使用 U7 编程器测试通过
#include "reg51.h"

typedef     bit             BOOL;
typedef     unsigned char   BYTE;
typedef     unsigned short  WORD;
typedef     unsigned long   DWORD;

#define     FOSC            18432000L
#define     BAUD            (65536 - FOSC / 4 / 115200)
#define     NULL            0
#define     FALSE           0
#define     TRUE            1
```

```
sfr         AUXR = 0x8e;                //辅助寄存器
sfr         P_SW1 = 0xa2;               //外设功能切换寄存器 1

#define     SPI_S0          0x04
#define     SPI_S1          0x08

sfr         SPSTAT = 0xcd;          //SPI 状态寄存器

#define     SPIF            0x80    //SPSTAT.7
#define     WCOL            0x40    //SPSTAT.6

sfr         SPCTL = 0xce;           //SPI 控制寄存器

#define     SSIG            0x80    //SPCTL.7
#define     SPEN            0x40    //SPCTL.6
#define     DORD            0x20    //SPCTL.5
#define     MSTR            0x10    //SPCTL.4
#define     CPOL            0x08    //SPCTL.3
#define     CPHA            0x04    //SPCTL.2

#define     SPDHH           0x00    //CPU_CLK/4
#define     SPDH            0x01    //CPU_CLK/8
#define     SPDL            0x02    //CPU_CLK/16
#define     SPDLL           0x03    //CPU_CLK/32

sfr         SPDAT = 0xcf;           //SPI 数据寄存器

sbit        SS = P2^4;              //SPI 的 SS 脚,连接到 Flash 的 CE

sfr         IE2 = 0xAF;             //中断控制寄存器 2
#define     ESPI            0x02    //IE2.1

#define     SFC_WREN        0x06    //串行 Flash 命令集
#define     SFC_WRDI        0x04
#define     SFC_RDSR        0x05
#define     SFC_WRSR        0x01
#define     SFC_READ        0x03
#define     SFC_FASTREAD    0x0B
#define     SFC_RDID        0xAB
#define     SFC_PAGEPROG    0x02
#define     SFC_RDCR        0xA1
#define     SFC_WRCR        0xF1
#define     SFC_SECTORER    0xD7
#define     SFC_BLOCKER     0xD8
#define     SFC_CHIPER      0xC7
```

```
void InitUart();
void SendUart(BYTE dat);
void InitSpi();
BYTE SpiShift(BYTE dat);
BOOL FlashCheckID();
BOOL IsFlashBusy();

void FlashWriteEnable();
void FlashErase();
void FlashRead(DWORD addr, DWORD size, BYTE *buffer);
void FlashWrite(DWORD addr, DWORD size, BYTE *buffer);

#define     BUFFER_SIZE     1024    //缓冲区大小
#define     TEST_ADDR       0       //Flash 测试地址

BYTE xdata g_Buffer[BUFFER_SIZE];   //Flash 读写缓冲区
BOOL g_fFlashOK;                    //Flash 状态
BOOL g_fSpiBusy;                    //SPI 的工作状态

void main()
{
    int i;

    //初始化 Flash 状态
    g_fFlashOK = FALSE;
    g_fSpiBusy = FALSE;

    //初始化串口和 SPI
    InitUart();
    InitSpi();

    //使能 SPI 传输中断
    IE2 |= ESPI;
    EA = 1;

    //检测 Flash 状态
    FlashCheckID();

    //擦除 Flash
    FlashErase();

    //读取测试地址的数据
    FlashRead(TEST_ADDR, BUFFER_SIZE, g_Buffer);

    //发送到串口
    for (i=0; i<BUFFER_SIZE; i++) SendUart(g_Buffer[i]);
```

```
    //修改置缓冲区
    for (i=0; i<BUFFER_SIZE; i++) g_Buffer[i] = i>>2;

    //将缓冲区的数据写到 Flash 中
    FlashWrite(TEST_ADDR, BUFFER_SIZE, g_Buffer);

    //读取测试地址的数据
    FlashRead(TEST_ADDR, BUFFER_SIZE, g_Buffer);
    //发送到串口
    for (i=0; i<BUFFER_SIZE; i++) SendUart(g_Buffer[i]);
    FlashErase();

    //读取测试地址的数据
    FlashRead(TEST_ADDR, BUFFER_SIZE, g_Buffer);

    //发送到串口
    for (i=0; i<BUFFER_SIZE; i++) SendUart(g_Buffer[i]);

    //修改置缓冲区
    for (i=0; i<BUFFER_SIZE; i++) g_Buffer[i]= 255-(i>>2);

    //将缓冲区的数据写到 Flash 中
    FlashWrite(TEST_ADDR, BUFFER_SIZE, g_Buffer);

    //读取测试地址的数据
    FlashRead(TEST_ADDR, BUFFER_SIZE, g_Buffer);

    //发送到串口
    for (i=0; i<BUFFER_SIZE; i++) SendUart(g_Buffer[i]);
    while (1);
}
/*************************************************
SPI 中断服务程序
*************************************************/
void spi_isr() interrupt 9 using 1
{
    SPSTAT = SPIF | WCOL;                   //清除 SPI 状态位
    g_fSpiBusy = FALSE;
}
/*************************************************
串口初始化
入口参数: 无
出口参数: 无
*************************************************/
void InitUart()
```

```
{
    AUXR = 0x40;                    //设置定时器 1 为 1T 模式
    TMOD = 0x00;                    //定时器 1 为 16 位重载模式
    TH1 = BAUD >> 8;                //设置波特率
    TL1 = BAUD;
    TR1 = 1;
    SCON = 0x5a;                    //设置串口为 8 位数据位,波特率可变模式
}

/*************************************************
发送数据到串口
入口参数:
    dat : 准备发送的数据
出口参数: 无
*************************************************/
void SendUart(BYTE dat)
{
    while (!TI);                    //等待上一个数据发送完成
    TI = 0;                         //清除发送完成标志
    SBUF = dat;                     //触发本次的数据发送

}

/*************************************************
SPI 初始化
入口参数: 无
出口参数: 无
*************************************************/
void InitSpi()
{
//  ACC = P_SW1;                    //切换到第一组 SPI
//  ACC &= ~(SPI_S0 | SPI_S1);      //SPI_S0=0 SPI_S1=0
//  P_SW1 = ACC;                    //(P1.2/SS, P1.3/MOSI, P1.4/MISO, P1.5/SCLK)
    ACC = P_SW1;                    //可用于测试 U7,U7 使用的是第二组 SPI 控制 Flash
    ACC &= ~(SPI_S0 | SPI_S1);      //SPI_S0=1 SPI_S1=0
    ACC |= SPI_S0;                  //(P2.4/SS_2, P2.3/MOSI_2, P2.2/MISO_2, P2.1/SCLK_2)

    P_SW1 = ACC;
//  ACC = P_SW1;                    //切换到第三组 SPI
//  ACC &= ~(SPI_S0 | SPI_S1);      //SPI_S0=0 SPI_S1=1
//  ACC |= SPI_S1;                  //(P5.4/SS_3, P4.0/MOSI_3, P4.1/MISO_3, P4.3/SCLK_3)
//  P_SW1 = ACC;

    SPSTAT = SPIF | WCOL;           //清除 SPI 状态

    SS = 1;
```

```
    SPCTL = SSIG | SPEN | MSTR;         //设置 SPI 为主模式
}

/*************************************************
使用 SPI 方式与 Flash 进行数据交换
入口参数:
    dat : 准备写入的数据
出口参数:
    从 Flash 中读出的数据
*************************************************/
BYTE SpiShift(BYTE dat)
{
    g_fSpiBusy = TRUE;
    SPDAT = dat;                        //触发 SPI 发送
    while (g_fSpiBusy);                 //等待 SPI 数据传输完成

    return SPDAT;
}

/*************************************************
检测 Flash 是否准备就绪
入口参数: 无
出口参数:
    0 : 没有检测到正确的 Flash
    1 : Flash 准备就绪
*************************************************/
BOOL FlashCheckID()
{
    BYTE dat1, dat2;

    SS = 0;
    SpiShift(SFC_RDID);                 //发送读取 ID 命令
    SpiShift(0x00);                     //空读 3 个字节
    SpiShift(0x00);
    SpiShift(0x00);
    dat1 = SpiShift(0x00);              //读取制造商 ID1
    SpiShift(0x00);                     //读取设备 ID
    dat2 = SpiShift(0x00);              //读取制造商 ID2
    SS = 1;                             //检测是否为 PM25LVxx 系列的 Flash

    g_fFlashOK = ((dat1 == 0x9d) && (dat2 == 0x7f));

    return g_fFlashOK;
}

/*************************************************
```

```
检测 Flash 的忙状态
入口参数: 无
出口参数:
    0 : Flash 处于空闲状态
    1 : Flash 处于忙状态
*************************************************/
BOOL IsFlashBusy()
{
    BYTE dat;
    SS = 0;
    SpiShift(SFC_RDSR);             //发送读取状态命令
    dat = SpiShift(0);              //读取状态
    SS = 1;
    return (dat & 0x01);            //状态值的 Bit0 即为忙标志
}

/*************************************************
使能 Flash 写命令
入口参数: 无
出口参数: 无
*************************************************/
void FlashWriteEnable()
{   while (IsFlashBusy());          //Flash 忙检测
    SS = 0;
    SpiShift(SFC_WREN);             //发送写使能命令
    SS = 1;
}

/*************************************************
擦除整片 Flash
入口参数: 无
出口参数: 无
*************************************************/
void FlashErase()
{
    if (g_fFlashOK)
    {
        FlashWriteEnable();         //使能 Flash 写命令
        SS = 0;
        SpiShift(SFC_CHIPER);       //发送片擦除命令
        SS = 1;
    }
}

/*************************************************
从 Flash 中读取数据
```

```
入口参数:
    addr : 地址参数
    size : 数据块大小
    buffer : 缓冲从 Flash 中读取的数据
出口参数: 无
*************************************************/
void FlashRead(DWORD addr, DWORD size, BYTE *buffer)
{
    if (g_fFlashOK)
    {
        while (IsFlashBusy());              //Flash 忙检测
        SS = 0;
        SpiShift(SFC_FASTREAD);             //使用快速读取命令
        SpiShift(((BYTE *)&addr)[1]);       //设置起始地址
        SpiShift(((BYTE *)&addr)[2]);
        SpiShift(((BYTE *)&addr)[3]);
        SpiShift(0);                        //需要空读一个字节
        while (size)
        {
            *buffer = SpiShift(0);          //自动连续读取并保存
            addr++;
            buffer++;
            size--;
        }
        SS = 1;
    }
}

/*************************************************
写数据到 Flash 中
入口参数:
    addr : 地址参数
    size : 数据块大小
    buffer : 缓冲需要写入 Flash 的数据
出口参数: 无
*************************************************/
void FlashWrite(DWORD addr, DWORD size, BYTE *buffer)
{
    if (g_fFlashOK)
    while (size)
    {
        FlashWriteEnable();                 //使能 Flash 写命令
        SS = 0;
        SpiShift(SFC_PAGEPROG);             //发送页编程命令
        SpiShift(((BYTE *)&addr)[1]);       //设置起始地址
        SpiShift(((BYTE *)&addr)[2]);
```

```
        SpiShift(((BYTE *)&addr)[3]);
        while (size)
        {
            SpiShift(*buffer);              //连续页内写
            addr++;
            buffer++;
            size--;
            if ((addr & 0xff) == 0) break;
        }
        SS = 1;
    }
}
```

2、汇编程序

```
/*---------------------------------------------------------------------------------------------------*/
/*----STC15W4K60S4 系列 SPI 的主模式读写外部串行 Flash 举例(中断方式)----*/
/*---------------------------------------------------------------------------------------------------*/
//本示例在 Keil 开发环境下请选择 Intel 的 8058 芯片型号进行编译
//假定测试芯片的工作频率为 18.432MHz
//本示例所读写的目标 Flash 为 PM25LV040,本代码已使用 U7 编程器测试通过
//BAUD      EQU     0FFE8H          //(65536 - 11059200 / 4 / 115200)
BAUD        EQU     0FFD8H          //(65536 - 18432000 / 4 / 115200)
//BAUD      EQU     0FFD0H          //(65536 - 22118400 / 4 / 115200)

AUXR        DATA    08EH            //辅助寄存器
P_SW1       DATA    0A2H            //外设功能切换寄存器 1
SPI_S0      EQU     04H
SPI_S1      EQU     08H

SPSTAT      DATA    0CDH            //SPI 状态寄存器
SPIF        EQU     080H            //SPSTAT.7
WCOL        EQU     040H            //SPSTAT.6
SPCTL       DATA    0CEH            //SPI 控制寄存器

SSIG        EQU     080H            //SPCTL.7
SPEN        EQU     040H            //SPCTL.6
DORD        EQU     020H            //SPCTL.5
MSTR        EQU     010H            //SPCTL.4
CPOL        EQU     008H            //SPCTL.3
CPHA        EQU     004H            //SPCTL.2

SPDHH       EQU     000H            //CPU_CLK/4
SPDH        EQU     001H            //CPU_CLK/8
SPDL        EQU     002H            //CPU_CLK/16
SPDLL       EQU     003H            //CPU_CLK/32
```

```
SPDAT       DATA    0CFH            //SPI 数据寄存器
SS          BIT     P2.4            //SPI 的 SS 脚,连接到 Flash 的 CE

IE2         DATA    0AFH            //中断控制寄存器 2
ESPI        EQU     002H            //IE2.1

SFC_WREN            EQU     0x06    //串行 Flash 命令集
SFC_WRDI            EQU     0x04
SFC_RDSR            EQU     0x05
SFC_WRSR            EQU     0x01
SFC_READ            EQU     0x03
SFC_FASTREAD        EQU     0x0B
SFC_RDID            EQU     0xAB
SFC_PAGEPROG        EQU     0x02
SFC_RDCR            EQU     0xA1
SFC_WRCR            EQU     0xF1
SFC_SECTORER        EQU     0xD7
SFC_BLOCKER         EQU     0xD8
SFC_CHIPER          EQU     0xC7

BUFFER_SIZE         EQU     1024    //缓冲区大小
TEST_ADDR           EQU     0       //Flash 测试地址

G_BUFFER            XDATA   0       //Flash 读写缓冲区
G_FLASHOK           BIT     20H.0   //Flash 状态
G_SPIBUSY           BIT     20H.1   //SPI 的工作状态

ADDR                DATA    21H     //地址变量,3 字节
SIZE                DATA    24H     //大小变量,3 字节

    ORG     0000H
    LJMP    START

    ORG     004BH
    LJMP    SPI_ISR

    ORG     0100H

START:
    MOV     SP, #3FH
    CLR     G_FLASHOK               //初始化 Flash 状态
    CLR     G_SPIBUSY

    LCALL   INITUART                //初始化串口和 SPI
    LCALL   INITSPI
```

```
ORL     IE2, #ESPI                  //使能 SPI 中断
SETB    EA

LCALL   FLASHCHECKID                //检测 Flash 状态

LCALL   FLASHERASE                  //擦除 Flash

MOV     ADDR+2, #0
MOV     ADDR+1, #HIGH TEST_ADDR
MOV     ADDR+0, #LOW TEST_ADDR
MOV     SIZE+2, #0
MOV     SIZE+1, #HIGH BUFFER_SIZE
MOV     SIZE+0, #LOW BUFFER_SIZE
MOV     DPTR, #G_BUFFER
LCALL   FLASHREAD                   //读取测试地址的数据
MOV     SIZE+2, #0
MOV     SIZE+1, #HIGH BUFFER_SIZE
MOV     SIZE+0, #LOW BUFFER_SIZE
MOV     DPTR, #G_BUFFER
LCALL   SENDUARTBLOCK               //发送到串口


MOV     SIZE+2, #0
MOV     SIZE+1, #HIGH BUFFER_SIZE
MOV     SIZE+0, #LOW BUFFER_SIZE
MOV     DPTR, #G_BUFFER
MOV     B, #55H
LCALL   XMEMSET                     //修改置缓冲区
MOV     ADDR+2, #0
MOV     ADDR+1, #HIGH TEST_ADDR
MOV     ADDR+0, #LOW TEST_ADDR
MOV     SIZE+2, #0
MOV     SIZE+1, #HIGH BUFFER_SIZE
MOV     SIZE+0, #LOW BUFFER_SIZE
MOV     DPTR, #G_BUFFER
LCALL   FLASHWRITE                  //将缓冲区的数据写到 Flash 中

MOV     ADDR+2, #0
MOV     ADDR+1, #HIGH TEST_ADDR
MOV     ADDR+0, #LOW TEST_ADDR
MOV     SIZE+2, #0
MOV     SIZE+1, #HIGH BUFFER_SIZE
MOV     SIZE+0, #LOW BUFFER_SIZE
MOV     DPTR, #G_BUFFER
LCALL   FLASHREAD                   //读取测试地址的数据
MOV     SIZE+2, #0
```

```
MOV     SIZE+1, #HIGH BUFFER_SIZE
MOV     SIZE+0, #LOW BUFFER_SIZE
MOV     DPTR, #G_BUFFER
LCALL   SENDUARTBLOCK           //发送到串口

LCALL   FLASHERASE              //擦除 Flash
MOV     ADDR+2, #0
MOV     ADDR+1, #HIGH TEST_ADDR
MOV     ADDR+0, #LOW TEST_ADDR
MOV     SIZE+2, #0
MOV     SIZE+1, #HIGH BUFFER_SIZE
MOV     SIZE+0, #LOW BUFFER_SIZE
MOV     DPTR, #G_BUFFER
LCALL   FLASHREAD               //读取测试地址的数据
MOV     SIZE+2, #0
MOV     SIZE+1, #HIGH BUFFER_SIZE
MOV     SIZE+0, #LOW BUFFER_SIZE
MOV     DPTR, #G_BUFFER
LCALL   SENDUARTBLOCK           //发送到串口

MOV     SIZE+2, #0
MOV     SIZE+1, #HIGH BUFFER_SIZE
MOV     SIZE+0, #LOW BUFFER_SIZE
MOV     DPTR, #G_BUFFER
MOV     B,    #0AAH
LCALL   XMEMSET                 //修改置缓冲区
MOV     ADDR+2, #0
MOV     ADDR+1, #HIGH TEST_ADDR
MOV     ADDR+0, #LOW TEST_ADDR
MOV     SIZE+2, #0
MOV     SIZE+1, #HIGH BUFFER_SIZE
MOV     SIZE+0, #LOW BUFFER_SIZE
MOV     DPTR, #G_BUFFER
LCALL   FLASHWRITE              //将缓冲区的数据写到 Flash 中

MOV     ADDR+2, #0
MOV     ADDR+1, #HIGH TEST_ADDR
MOV     ADDR+0, #LOW TEST_ADDR
MOV     SIZE+2, #0
MOV     SIZE+1, #HIGH BUFFER_SIZE
MOV     SIZE+0, #LOW BUFFER_SIZE
MOV     DPTR, #G_BUFFER
LCALL   FLASHREAD               //读取测试地址的数据
MOV     SIZE+2, #0
MOV     SIZE+1, #HIGH BUFFER_SIZE
MOV     SIZE+0, #LOW BUFFER_SIZE
```

```
    MOV     DPTR, #G_BUFFER
    LCALL   SENDUARTBLOCK                   //发送到串口

    SJMP    $

/*************************************************
SPI 中断服务程序
*************************************************/
SPI_ISR:
    MOV     SPSTAT, #SPIF | WCOL            //清除 SPI 状态位
    CLR     G_SPIBUSY
    RETI

/*************************************************
发送数据到串口
入口参数:
    SIZE : 数据块大小
    DPTR : 数据缓冲区
    B : 设置的值
出口参数: 无
*************************************************/
XMEMSET:
    MOV     A, SIZE
    ORL     A, SIZE+1
    ORL     A, SIZE+2
    JZ      XMSEND
    MOV     A, B
    MOVX    @DPTR, A                    //自动连续设置 XDATA 数据
    INC     DPTR
    CLR     C
    MOV     A, SIZE+0
    SUBB    A, #1
    MOV     SIZE+0, A
    MOV     A, SIZE+1
    SUBB    A, #0
    MOV     SIZE+1, A
    MOV     A, SIZE+2
    SUBB    A, #0
    MOV     SIZE+2, A
    LJMP    XMEMSET

XMSEND:
    RET

/*************************************************
串口初始化
```

```
入口参数: 无
出口参数: 无
***********************************************/
INITUART:
    MOV     AUXR, #40H              //设置定时器 1 为 1T 模式
    MOV     TMOD, #00H              //定时器 1 为 16 位重载模式
    MOV     TH1, #HIGH BAUD         //设置波特率
    MOV     TL1, #LOW BAUD
    SETB    TR1
    MOV     SCON,   #5AH            //设置串口为 8 位数据位,波特率可变模式

    RET

/***********************************************
发送数据到串口
入口参数:
    ACC : 准备发送的数据
出口参数: 无
***********************************************/
SENDUART:
    JNB     TI, $                   //等待上一个数据发送完成
    CLR     TI                      //清除发送完成标志
    MOV     SBUF, A                 //触发本次的数据发送

    RET

/***********************************************
发送数据块到串口
入口参数:
    SIZE : 数据块大小
    DPTR : 数据缓冲区
出口参数: 无
***********************************************/
SENDUARTBLOCK:
    MOV     A, SIZE
    ORL     A, SIZE+1
    ORL     A, SIZE+2
    JZ      SUBEND
    MOVX    A, @DPTR
    LCALL   SENDUART                //自动连续发送串口数据
    INC     DPTR
    CLR     C
    MOV     A, SIZE+0
    SUBB    A, #1
    MOV     SIZE+0, A
    MOV     A, SIZE+1
```

```
    SUBB    A, #0
    MOV     SIZE+1, A
    MOV     A, SIZE+2
    SUBB    A, #0
    MOV     SIZE+2, A
    LJMP    SENDUARTBLOCK

SUBEND:
    RET

/*************************************************
SPI 初始化
入口参数: 无
出口参数: 无
*************************************************/
INITSPI:
//  MOV     A, P_SW1                    //切换到第一组 SPI
//  ANL     A, #NOT (SPI_S0 | SPI_S1)   //SPI_S0=0 SPI_S1=0
//  MOV     P_SW1, A                    //(P1.2/SS,  P1.3/MOSI,  P1.4/MISO,  P1.5/SCLK)
    MOV     A, P_SW1                    //可用于测试 U7, U7 使用的是第二组 SPI 控制 Flash
    ANL     A, #NOT (SPI_S0 | SPI_S1)   //SPI_S0=1 SPI_S1=0
    ORL     A, #SPI_S0                  //(P2.4/SS_2,  P2.3/MOSI_2,  P2.2/MISO_2,  P2.1/SCLK_2)
    MOV     P_SW1, A

//  MOV     A, P_SW1                    //切换到第三组 SPI
//  ANL     A, #NOT (SPI_S0 | SPI_S1)   //SPI_S0=0 SPI_S1=1
//  ORL     A, #SPI_S1                  //(P5.4/SS_3,  P4.0/MOSI_3,  P4.1/MISO_3,  P4.3/SCLK_3)
//  MOV     P_SW1, A
    MOV     SPSTAT, #SPIF | WCOL        //清除 SPI 状态
    SETB    SS
    MOV     SPCTL, #SSIG | SPEN | MSTR  //设置 SPI 为主模式

    RET

/*************************************************
使用 SPI 方式与 Flash 进行数据交换
入口参数:
    ACC : 准备写入的数据
出口参数:
    ACC : 从 Flash 中读出的数据
*************************************************/
SPISHIFT:
    SETB    G_SPIBUSY
    MOV     SPDAT, A                //触发 SPI 发送
    JB      G_SPIBUSY, $            //等待 SPI 数据传输完成
    MOV     A, SPDAT
```

```
    RET

/*********************************************
检测 Flash 是否准备就绪
入口参数: 无
出口参数: CY
    0 : 没有检测到正确的 Flash
    1 : Flash 准备就绪
*********************************************/
FLASHCHECKID:
    CLR     SS
    MOV     A, #SFC_RDID            //发送读取 ID 命令
    LCALL   SPISHIFT
    LCALL   SPISHIFT                //空读 3 个字节
    LCALL   SPISHIFT
    LCALL   SPISHIFT
    LCALL   SPISHIFT                //读取制造商 ID1
    MOV     R7, A
    LCALL   SPISHIFT                //读取设备 ID
    LCALL   SPISHIFT                //读取制造商 ID2
    MOV     R6, A
    SETB    SS
    CLR     G_FLASHOK
    CJNE    R7, #9DH, $             //检测是否为 PM25LVxx 系列的 Flash
    CJNE    R6, #7FH, $
    SETB    G_FLASHOK

FLASHERROR:
    MOV     C, G_FLASHOK

RET

/*********************************************
检测 Flash 的忙状态
入口参数: 无
出口参数: CY
    0 : Flash 处于空闲状态
    1 : Flash 处于忙状态
*********************************************/
ISFLASHBUSY:
    CLR     SS
    MOV     A, #SFC_RDSR            //发送读取状态命令
    LCALL   SPISHIFT
    LCALL   SPISHIFT                //读取状态
    SETB    SS
    MOV     C, ACC.0                //状态值的 Bit0 即为忙标志
```

```
    RET

/*************************************************
使能 Flash 写命令
入口参数: 无
出口参数: 无
*************************************************/
FALSHWRITEENABLE:
    LCALL   ISFLASHBUSY
    JC      $-3                     //Flash 忙检测
    CLR     SS
    MOV     A, #SFC_WREN            //发送写使能命令
    LCALL   SPISHIFT
    SETB    SS
    RET

/*************************************************
擦除整片 Flash
入口参数: 无
出口参数: 无
*************************************************/
FLASHERASE:
    JNB     G_FLASHOK, FEREXIT
    LCALL   FALSHWRITEENABLE    //使能 Flash 写命令
    CLR     SS
    MOV     A, #SFC_CHIPER          //发送片擦除命令
    LCALL   SPISHIFT
    SETB    SS

FEREXIT:
    RET

/*************************************************
从 Flash 中读取数据
入口参数:
    ADDR : FALSH 地址参数
    SIZE :  数据块大小
    DPTR :  缓冲区首地址
出口参数: 无
*************************************************/
FLASHREAD:
    JNB     G_FLASHOK, FRDEXIT
    LCALL   ISFLASHBUSY
    JC      $-3 //Flash 忙检测
    CLR     SS
    MOV     A, #SFC_FASTREAD            //使用快速读取命令
```

```
    LCALL   SPISHIFT

    MOV     A, ADDR+2               //设置起始地址
    LCALL   SPISHIFT
    MOV     A, ADDR+1
    LCALL   SPISHIFT
    MOV     A, ADDR+0
    LCALL   SPISHIFT

    LCALL   SPISHIFT                //需要空读一个字节

FRDNEXTBYTE:
    MOV     A, SIZE
    ORL     A, SIZE+1
    ORL     A, SIZE+2
    JZ      FRDEND
    LCALL   SPISHIFT                //自动连续读取并保存
    MOVX    @DPTR, A
    INC     DPTR
    MOV     A, ADDR+0
    ADD     A, #1
    MOV     ADDR+0, A
    MOV     A, ADDR+1
    ADDC    A, #0
    MOV     ADDR+1, A
    MOV     A, ADDR+2
    ADDC    A, #0
    MOV     ADDR+2, A
    CLR     C
    MOV     A, SIZE+0
    SUBB    A, #1
    MOV     SIZE+0, A
    MOV     A, SIZE+1
    SUBB    A, #0
    MOV     SIZE+1, A
    MOV     A, SIZE+2
    SUBB    A, #0
    MOV     SIZE+2, A
    LJMP    FRDNEXTBYTE

FRDEND:
    SETB    SS

FRDEXIT:
    RET
```

```
/*************************************************
写数据到 Flash 中
入口参数:
    ADDR : 地址参数
    SIZE : 数据块大小
    DPTR : 缓冲需要写入 Flash 的数据
出口参数: 无
*************************************************/
FLASHWRITE:
    JNB     G_FLASHOK, FWREXIT

FWRNEXTPAGE:
    MOV     A, SIZE
    ORL     A, SIZE+1
    ORL     A, SIZE+2
    JZ      FWREXIT
    LCALL   FALSHWRITEENABLE        //使能 Flash 写命令
    CLR     SS
    MOV     A, #SFC_PAGEPROG        //发送页编程命令
    LCALL   SPISHIFT
    MOV     A, ADDR+2               //设置起始地址
    LCALL   SPISHIFT
    MOV     A, ADDR+1
    LCALL   SPISHIFT
    MOV     A, ADDR+0
    LCALL   SPISHIFT

FWRNEXTBYTE:
    MOV     A, SIZE
    ORL     A, SIZE+1
    ORL     A, SIZE+2
    JZ      FRDEND
    MOVX    A, @DPTR
    LCALL   SPISHIFT                //连续页内写
    INC     DPTR
    MOV     A, ADDR+0
    ADD     A, #1
    MOV     ADDR+0, A
    MOV     A, ADDR+1
    ADDC    A, #0
    MOV     ADDR+1, A
    MOV     A, ADDR+2
    ADDC    A, #0
    MOV     ADDR+2, A
    CLR     C
    MOV     A, SIZE+0
```

```
    SUBB    A, #1
    MOV     SIZE+0,   A
    MOV     A, SIZE+1
    SUBB    A, #0
    MOV     SIZE+1, A
    MOV     A, SIZE+2
    SUBB    A, #0
    MOV     SIZE+2, A
    MOV     A, ADDR+0
    JZ      FWREND
    LJMP    FWRNEXTBYTE

FWREND:
    SETB    SS
    LJMP    FWRNEXTPAGE

FWREXIT:
    RET

    END
```

26.8.2.2 通过查询方式利用 SPI 的主模式读写外部串行 Flash 的测试程序(C 和汇编)

1、C 语言程序

```
/*---------------------------------------------------------------------------------------------*/
/*----STC15W4K60S4 系列 SPI 的主模式读写外部串行 Flash 举例(查询方式)----*/
/*---------------------------------------------------------------------------------------------*/
//本示例在 Keil 开发环境下请选择 Intel 的 8058 芯片型号进行编译
//假定测试芯片的工作频率为 18.432MHz
//本示例所读写的目标 Flash 为 PM25LV040,本代码已使用 U7 编程器测试通过
#include "reg51.h"

typedef     bit             BOOL;
typedef     unsigned char   BYTE;
typedef     unsigned short  WORD;
typedef     unsigned long   DWORD;

#define     FOSC            18432000L
#define     BAUD            (65536 - FOSC / 4 / 115200)
#define     NULL            0
#define     FALSE           0
#define     TRUE            1
```

```
sfr         AUXR = 0x8e;            //辅助寄存器
sfr         P_SW1 = 0xa2;           //外设功能切换寄存器 1

#define     SPI_S0          0x04
#define     SPI_S1          0x08

sfr         SPSTAT = 0xcd;          //SPI 状态寄存器

#define     SPIF            0x80    //SPSTAT.7
#define     WCOL            0x40    //SPSTAT.6

sfr         SPCTL = 0xce;           //SPI 控制寄存器

#define     SSIG            0x80    //SPCTL.7
#define     SPEN            0x40    //SPCTL.6
#define     DORD            0x20    //SPCTL.5
#define     MSTR            0x10    //SPCTL.4
#define     CPOL            0x08    //SPCTL.3
#define     CPHA            0x04    //SPCTL.2

#define     SPDHH           0x00    //CPU_CLK/4
#define     SPDH            0x01    //CPU_CLK/8
#define     SPDL            0x02    //CPU_CLK/16
#define     SPDLL           0x03    //CPU_CLK/32

sfr         SPDAT = 0xcf;           //SPI 数据寄存器
sbit        SS = P2^4;              //SPI 的 SS 脚,连接到 Flash 的 CE

#define     SFC_WREN        0x06    //串行 Flash 命令集
#define     SFC_WRDI        0x04
#define     SFC_RDSR        0x05
#define     SFC_WRSR        0x01
#define     SFC_READ        0x03
#define     SFC_FASTREAD    0x0B
#define     SFC_RDID        0xAB
#define     SFC_PAGEPROG    0x02
#define     SFC_RDCR        0xA1
#define     SFC_WRCR        0xF1
#define     SFC_SECTORER    0xD7
#define     SFC_BLOCKER     0xD8
#define     SFC_CHIPER      0xC7

void InitUart();
void SendUart(BYTE dat);
void InitSpi();
BYTE SpiShift(BYTE dat);
```

```
BOOL FlashCheckID();
BOOL IsFlashBusy();

void FlashWriteEnable();
void FlashErase();
void FlashRead(DWORD addr, DWORD size, BYTE *buffer);
void FlashWrite(DWORD addr, DWORD size, BYTE *buffer);

#define   BUFFER_SIZE     1024          //缓冲区大小
#define   TEST_ADDR       0             //Flash 测试地址

BYTE xdata g_Buffer[BUFFER_SIZE];       //Flash 读写缓冲区
BOOL g_fFlashOK;                        //Flash 状态

void main()
{
    int i;

    //初始化 Flash 状态
    g_fFlashOK = FALSE;

    //初始化串口和 SPI
    InitUart();
    InitSpi();

    //检测 Flash 状态
    FlashCheckID();

    //擦除 Flash
    FlashErase();

    //读取测试地址的数据
    FlashRead(TEST_ADDR, BUFFER_SIZE, g_Buffer);

    //发送到串口
    for (i=0; i<BUFFER_SIZE; i++) SendUart(g_Buffer[i]);

    //修改置缓冲区
    for (i=0; i<BUFFER_SIZE; i++) g_Buffer[i] = i>>2;

    //将缓冲区的数据写到 Flash 中
    FlashWrite(TEST_ADDR, BUFFER_SIZE, g_Buffer);

    //读取测试地址的数据
    FlashRead(TEST_ADDR, BUFFER_SIZE, g_Buffer);
```

```
    //发送到串口
    for (i=0; i<BUFFER_SIZE; i++) SendUart(g_Buffer[i]);
    FlashErase();

    //读取测试地址的数据
    FlashRead(TEST_ADDR, BUFFER_SIZE, g_Buffer);

    //发送到串口
    for (i=0; i<BUFFER_SIZE; i++) SendUart(g_Buffer[i]);

    //修改置缓冲区
    for (i=0; i<BUFFER_SIZE; i++) g_Buffer[i]= 255-(i>>2);

    //将缓冲区的数据写到 Flash 中
    FlashWrite(TEST_ADDR, BUFFER_SIZE, g_Buffer);

    //读取测试地址的数据
    FlashRead(TEST_ADDR, BUFFER_SIZE, g_Buffer);

    //发送到串口
    for (i=0; i<BUFFER_SIZE; i++) SendUart(g_Buffer[i]);
    while (1);
}

/*************************************************
串口初始化
入口参数: 无
出口参数: 无
*************************************************/
void InitUart()
{
    AUXR = 0x40;                //设置定时器 1 为 1T 模式
    TMOD = 0x00;                //定时器 1 为 16 位重载模式
    TH1 = BAUD >> 8;            //设置波特率
    TL1 = BAUD;
    TR1 = 1;
    SCON = 0x5a;                //设置串口为 8 位数据位,波特率可变模式
}

/*************************************************
发送数据到串口
入口参数:
    dat : 准备发送的数据
出口参数: 无
*************************************************/
void SendUart(BYTE dat)
```

```
{
    while (!TI);                    //等待上一个数据发送完成
    TI = 0;                         //清除发送完成标志
    SBUF = dat;                     //触发本次的数据发送
}

/*************************************************
SPI 初始化
入口参数: 无
出口参数: 无
*************************************************/
void InitSpi()
{
//  ACC = P_SW1;                    //切换到第一组 SPI
//  ACC &= ~(SPI_S0 | SPI_S1);      //SPI_S0=0 SPI_S1=0
//  P_SW1 = ACC;                    //(P1.2/SS, P1.3/MOSI, P1.4/MISO, P1.5/SCLK)
    ACC = P_SW1;                    //可用于测试 U7,U7 使用的是第二组 SPI 控制 Flash
    ACC &= ~(SPI_S0 | SPI_S1);      //SPI_S0=1 SPI_S1=0
    ACC |= SPI_S0;                  //(P2.4/SS_2, P2.3/MOSI_2, P2.2/MISO_2, P2.1/SCLK_2)
    P_SW1 = ACC;

//  ACC = P_SW1;                    //切换到第三组 SPI
//  ACC &= ~(SPI_S0 | SPI_S1);      //SPI_S0=0 SPI_S1=1
//  ACC |= SPI_S1;                  //(P5.4/SS_3, P4.0/MOSI_3, P4.1/MISO_3, P4.3/SCLK_3)
//  P_SW1 = ACC;
    SPSTAT = SPIF | WCOL;           //清除 SPI 状态
    SS = 1;
    SPCTL = SSIG | SPEN | MSTR;     //设置 SPI 为主模式
}

/*************************************************
使用 SPI 方式与 Flash 进行数据交换
入口参数:
    dat : 准备写入的数据
出口参数:
    从 Flash 中读出的数据
*************************************************/
BYTE SpiShift(BYTE dat)
{
    SPDAT = dat;                    //触发 SPI 发送
    while (!(SPSTAT & SPIF));       //等待 SPI 数据传输完成
    SPSTAT = SPIF | WCOL;           //清除 SPI 状态

    return SPDAT;
}
```

```
/*************************************************
检测 Flash 是否准备就绪
入口参数: 无
出口参数:
    0 : 没有检测到正确的 Flash
    1 : Flash 准备就绪
*************************************************/
BOOL FlashCheckID()
{
    BYTE dat1, dat2;

    SS = 0;
    SpiShift(SFC_RDID);             //发送读取 ID 命令
    SpiShift(0x00);                 //空读 3 个字节
    SpiShift(0x00);
    SpiShift(0x00);
    dat1 = SpiShift(0x00);          //读取制造商 ID1
    SpiShift(0x00);                 //读取设备 ID
    dat2 = SpiShift(0x00);          //读取制造商 ID2
    SS = 1;
                                    //检测是否为 PM25LVxx 系列的 Flash
    g_fFlashOK = ((dat1 == 0x9d) && (dat2 == 0x7f));

    return g_fFlashOK;
}

/*************************************************
检测 Flash 的忙状态
入口参数: 无
出口参数:
    0 : Flash 处于空闲状态
    1 : Flash 处于忙状态
*************************************************/
BOOL IsFlashBusy()
{
    BYTE dat;
    SS = 0;
    SpiShift(SFC_RDSR);             //发送读取状态命令
    dat = SpiShift(0);              //读取状态
    SS = 1;
    return (dat & 0x01);            //状态值的 Bit0 即为忙标志
}

/*************************************************
使能 Flash 写命令
入口参数: 无
```

```
出口参数: 无
*************************************************/
void FlashWriteEnable()
{
    while (IsFlashBusy());              //Flash 忙检测
    SS = 0;
    SpiShift(SFC_WREN);                 //发送写使能命令
    SS = 1;
}

/*************************************************
擦除整片 Flash
入口参数: 无
出口参数: 无
*************************************************/
void FlashErase()
{
    if (g_fFlashOK)
    {
        FlashWriteEnable();             //使能 Flash 写命令
        SS = 0;
        SpiShift(SFC_CHIPER);           //发送片擦除命令
    SS = 1;
    }
}

/*************************************************
从 Flash 中读取数据
入口参数:
    addr : 地址参数
    size : 数据块大小
    buffer : 缓冲从 Flash 中读取的数据
出口参数: 无
*************************************************/
void FlashRead(DWORD addr, DWORD size, BYTE *buffer)
{
    if (g_fFlashOK)
    {
        while (IsFlashBusy());              //Flash 忙检测
        SS = 0;
        SpiShift(SFC_FASTREAD);             //使用快速读取命令
        SpiShift(((BYTE *)&addr)[1]);       //设置起始地址
        SpiShift(((BYTE *)&addr)[2]);
        SpiShift(((BYTE *)&addr)[3]);
        SpiShift(0);                        //需要空读一个字节
        while (size)
```

```
        {
            *buffer = SpiShift(0);          //自动连续读取并保存
            addr++;
            buffer++;
            size--;
        }
        SS = 1;
    }
}

/*************************************************
写数据到 Flash 中
入口参数:
    addr : 地址参数
    size : 数据块大小
    buffer : 缓冲需要写入 Flash 的数据
出口参数: 无
*************************************************/
void FlashWrite(DWORD addr, DWORD size, BYTE *buffer)
{
    if (g_fFlashOK)
    while (size)
    {
        FlashWriteEnable();             //使能 Flash 写命令
        SS = 0;
        SpiShift(SFC_PAGEPROG);         //发送页编程命令
        SpiShift(((BYTE *)&addr)[1]);   //设置起始地址
        SpiShift(((BYTE *)&addr)[2]);
        SpiShift(((BYTE *)&addr)[3]);
        while (size)
        {
            SpiShift(*buffer);          //连续页内写
            addr++;
            buffer++;
            size--;
            if ((addr & 0xff) == 0) break;
        }
        SS = 1;
    }
}
```

2、汇编程序

```
/*---------------------------------------------------------------------------------------------*/
/*----STC15W4K60S4 系列 SPI 的主模式读写外部串行 Flash 举例(查询方式)----*/
/*---------------------------------------------------------------------------------------------*/

//本示例在 Keil 开发环境下请选择 Intel 的 8058 芯片型号进行编译
//假定测试芯片的工作频率为 18.432MHz
//本示例所读写的目标 Flash 为 PM25LV040,本代码已使用 U7 编程器测试通过

//BAUD      EQU   0FFE8H    //(65536 - 11059200 / 4 / 115200)
BAUD        EQU   0FFD8H    //(65536 - 18432000 / 4 / 115200)
//BAUD      EQU   0FFD0H    //(65536 - 22118400 / 4 / 115200)

AUXR        DATA  08EH      //辅助寄存器
P_SW1       DATA  0A2H      //外设功能切换寄存器 1

SPI_S0      EQU   04H
SPI_S1      EQU   08H

SPSTAT      DATA  0CDH      //SPI 状态寄存器
SPIF        EQU   080H      //SPSTAT.7
WCOL        EQU   040H      //SPSTAT.6
SPCTL       DATA  0CEH      //SPI 控制寄存器

SSIG        EQU   080H      //SPCTL.7
SPEN        EQU   040H      //SPCTL.6
DORD        EQU   020H      //SPCTL.5
MSTR        EQU   010H      //SPCTL.4
CPOL        EQU   008H      //SPCTL.3
CPHA        EQU   004H      //SPCTL.2

SPDHH       EQU   000H      //CPU_CLK/4
SPDH        EQU   001H      //CPU_CLK/8
SPDL        EQU   002H      //CPU_CLK/16
SPDLL       EQU   003H      //CPU_CLK/32
SPDAT       DATA  0CFH      //SPI 数据寄存器

SS          BIT   P2.4      //SPI 的 SS 脚,连接到 Flash 的 CE

SFC_WREN        EQU   0x06      //串行 Flash 命令集
SFC_WRDI        EQU   0x04
SFC_RDSR        EQU   0x05
SFC_WRSR        EQU   0x01
SFC_READ        EQU   0x03
SFC_FASTREAD    EQU   0x0B
```

```
SFC_RDID        EQU     0xAB
SFC_PAGEPROG    EQU     0x02
SFC_RDCR        EQU     0xA1
SFC_WRCR        EQU     0xF1
SFC_SECTORER    EQU     0xD7
SFC_BLOCKER     EQU     0xD8
SFC_CHIPER      EQU     0xC7

BUFFER_SIZE     EQU     1024    //缓冲区大小
TEST_ADDR       EQU     0       //Flash 测试地址
G_BUFFER        XDATA   0       //Flash 读写缓冲区
G_FLASHOK       BIT     20H.0   //Flash 状态

ADDR            DATA    21H     //地址变量,3 字节
SIZE            DATA    24H     //大小变量,3 字节

    ORG     0000H
    LJMP    START

    ORG     0100H

START:
    MOV     SP, #3FH
    CLR     G_FLASHOK           //初始化 Flash 状态

    LCALL   INITUART            //初始化串口和 SPI
    LCALL   INITSPI

    LCALL   FLASHCHECKID        //检测 Flash 状态

    LCALL   FLASHERASE          //擦除 Flash

    MOV     ADDR+2, #0
    MOV     ADDR+1, #HIGH TEST_ADDR
    MOV     ADDR+0, #LOW TEST_ADDR
    MOV     SIZE+2, #0
    MOV     SIZE+1, #HIGH BUFFER_SIZE
    MOV     SIZE+0, #LOW BUFFER_SIZE
    MOV     DPTR, #G_BUFFER
    LCALL   FLASHREAD           //读取测试地址的数据
    MOV     SIZE+2, #0
    MOV     SIZE+1, #HIGH BUFFER_SIZE
    MOV     SIZE+0, #LOW BUFFER_SIZE
    MOV     DPTR, #G_BUFFER
    LCALL   SENDUARTBLOCK       //发送到串口
```

```
MOV     SIZE+2, #0
MOV     SIZE+1, #HIGH BUFFER_SIZE
MOV     SIZE+0, #LOW BUFFER_SIZE
MOV     DPTR, #G_BUFFER
MOV     B, #33H
LCALL   XMEMSET                 //修改置缓冲区
MOV     ADDR+2, #0
MOV     ADDR+1, #HIGH TEST_ADDR
MOV     ADDR+0, #LOW TEST_ADDR
MOV     SIZE+2, #0
MOV     SIZE+1, #HIGH BUFFER_SIZE
MOV     SIZE+0, #LOW BUFFER_SIZE
MOV     DPTR, #G_BUFFER
LCALL   FLASHWRITE              //将缓冲区的数据写到 Flash 中

MOV     ADDR+2, #0
MOV     ADDR+1, #HIGH TEST_ADDR
MOV     ADDR+0, #LOW TEST_ADDR
MOV     SIZE+2, #0
MOV     SIZE+1, #HIGH BUFFER_SIZE
MOV     SIZE+0, #LOW BUFFER_SIZE
MOV     DPTR, #G_BUFFER
LCALL   FLASHREAD               //读取测试地址的数据
MOV     SIZE+2, #0
MOV     SIZE+1, #HIGH BUFFER_SIZE
MOV     SIZE+0, #LOW BUFFER_SIZE
MOV     DPTR, #G_BUFFER
LCALL   SENDUARTBLOCK           //发送到串口

LCALL   FLASHERASE              //擦除 Flash
MOV     ADDR+2, #0
MOV     ADDR+1, #HIGH TEST_ADDR
MOV     ADDR+0, #LOW TEST_ADDR
MOV     SIZE+2, #0
MOV     SIZE+1, #HIGH BUFFER_SIZE
MOV     SIZE+0, #LOW BUFFER_SIZE
MOV     DPTR, #G_BUFFER
LCALL   FLASHREAD               //读取测试地址的数据
MOV     SIZE+2, #0
MOV     SIZE+1, #HIGH BUFFER_SIZE
MOV     SIZE+0, #LOW BUFFER_SIZE
MOV     DPTR, #G_BUFFER
LCALL   SENDUARTBLOCK           //发送到串口

MOV     SIZE+2, #0
MOV     SIZE+1, #HIGH BUFFER_SIZE
```

```
    MOV     SIZE+0, #LOW BUFFER_SIZE
    MOV     DPTR, #G_BUFFER
    MOV     B, #099H
    LCALL   XMEMSET                 //修改置缓冲区
    MOV     ADDR+2, #0
    MOV     ADDR+1, #HIGH TEST_ADDR
    MOV     ADDR+0, #LOW TEST_ADDR
    MOV     SIZE+2, #0
    MOV     SIZE+1, #HIGH BUFFER_SIZE
    MOV     SIZE+0, #LOW BUFFER_SIZE
    MOV     DPTR, #G_BUFFER
    LCALL   FLASHWRITE              //将缓冲区的数据写到 Flash 中

    MOV     ADDR+2, #0
    MOV     ADDR+1, #HIGH TEST_ADDR
    MOV     ADDR+0, #LOW TEST_ADDR
    MOV     SIZE+2, #0
    MOV     SIZE+1, #HIGH BUFFER_SIZE
    MOV     SIZE+0, #LOW BUFFER_SIZE
    MOV     DPTR, #G_BUFFER
    LCALL   FLASHREAD               //读取测试地址的数据
    MOV     SIZE+2, #0
    MOV     SIZE+1, #HIGH BUFFER_SIZE
    MOV     SIZE+0, #LOW BUFFER_SIZE
    MOV     DPTR, #G_BUFFER
    LCALL   SENDUARTBLOCK           //发送到串口

    SJMP    $

/***********************************************
发送数据到串口
入口参数:
    SIZE : 数据块大小
    DPTR : 数据缓冲区
    B : 设置的值
出口参数: 无
***********************************************/
XMEMSET:
    MOV     A, SIZE
    ORL     A, SIZE+1
    ORL     A, SIZE+2
    JZ      XMSEND
    MOV     A, B
    MOVX    @DPTR, A                //自动连续设置 XDATA 数据
    INC     DPTR
    CLR     C
```

```
    MOV     A, SIZE+0
    SUBB    A, #1
    MOV     SIZE+0, A
    MOV     A, SIZE+1
    SUBB    A, #0
    MOV     SIZE+1, A
    MOV     A, SIZE+2
    SUBB    A, #0
    MOV     SIZE+2, A
    LJMP    XMEMSET

XMSEND:
    RET

/*********************************************
串口初始化
入口参数: 无
出口参数: 无
*********************************************/
INITUART:
    MOV     AUXR, #40H              //设置定时器 1 为 1T 模式
    MOV     TMOD, #00H              //定时器 1 为 16 位重载模式
    MOV     TH1, #HIGH BAUD         //设置波特率
    MOV     TL1, #LOW BAUD
    SETB    TR1
    MOV     SCON, #5AH              //设置串口为 8 位数据位, 波特率可变模式
    RET

/*********************************************
发送数据到串口
入口参数:
    ACC : 准备发送的数据
出口参数: 无
*********************************************/
SENDUART:
    JNB     TI, $                   //等待上一个数据发送完成
    CLR     TI                      //清除发送完成标志
    MOV     SBUF, A                 //触发本次的数据发送
    RET

/*********************************************
发送数据块到串口
入口参数:
    SIZE : 数据块大小
    DPTR : 数据缓冲区
出口参数: 无
```

```
**********************************************/
SENDUARTBLOCK:
    MOV     A, SIZE
    ORL     A, SIZE+1
    ORL     A, SIZE+2
    JZ      SUBEND
    MOVX    A, @DPTR
    LCALL   SENDUART                //自动连续发送串口数据
    INC     DPTR
    CLR     C
    MOV     A, SIZE+0
    SUBB    A, #1
    MOV     SIZE+0, A
    MOV     A, SIZE+1
    SUBB    A, #0
    MOV     SIZE+1, A
    MOV     A, SIZE+2
    SUBB    A, #0
    MOV     SIZE+2, A
    LJMP    SENDUARTBLOCK

SUBEND:
    RET

/***********************************************
SPI 初始化
入口参数: 无
出口参数: 无
**********************************************/
INITSPI:
//  MOV     A, P_SW1                //切换到第一组 SPI
//  ANL     A, #NOT (SPI_S0 | SPI_S1)   //SPI_S0=0 SPI_S1=0
//  MOV     P_SW1, A                //(P1.2/SS,  P1.3/MOSI,  P1.4/MISO,  P1.5/SCLK)

    MOV     A, P_SW1                //可用于测试 U7, U7 使用的是第二组 SPI 控制 Flash
    ANL     A, #NOT (SPI_S0 | SPI_S1)   //SPI_S0=1 SPI_S1=0
    ORL     A, #SPI_S0              //(P2.4/SS_2,  P2.3/MOSI_2,  P2.2/MISO_2,  P2.1/SCLK_2)
    MOV     P_SW1, A

//  MOV     A, P_SW1                //切换到第三组 SPI
//  ANL     A, #NOT (SPI_S0 | SPI_S1)   //SPI_S0=0 SPI_S1=1
//  ORL     A, #SPI_S1              //(P5.4/SS_3,  P4.0/MOSI_3,  P4.1/MISO_3,  P4.3/SCLK_3)
//  MOV     P_SW1, A

    MOV     SPSTAT, #SPIF | WCOL    //清除 SPI 状态
    SETB    SS
```

```
    MOV     SPCTL, #SSIG | SPEN | MSTR    //设置 SPI 为主模式

    RET

/*********************************************
使用 SPI 方式与 Flash 进行数据交换
入口参数:
    ACC : 准备写入的数据
出口参数:
    ACC : 从 Flash 中读出的数据
*********************************************/
SPISHIFT:
    MOV     SPDAT, A                    //触发 SPI 发送

WAITSPI:
    MOV     A, SPSTAT                   //等待 SPI 数据传输完成
    ANL     A, #SPIF
    JZ      WAITSPI
    MOV     SPSTAT, #SPIF | WCOL        //清除 SPI 状态
    MOV     A, SPDAT
    RET

/*********************************************
检测 Flash 是否准备就绪
入口参数: 无
出口参数: CY
    0 : 没有检测到正确的 Flash
    1 : Flash 准备就绪
*********************************************/
FLASHCHECKID:
    CLR     SS
    MOV     A, #SFC_RDID                //发送读取 ID 命令
    LCALL   SPISHIFT

    LCALL   SPISHIFT                    //空读 3 个字节
    LCALL   SPISHIFT
    LCALL   SPISHIFT
    LCALL   SPISHIFT                    //读取制造商 ID1
    MOV     R7, A
    LCALL   SPISHIFT                    //读取设备 ID
    LCALL   SPISHIFT                    //读取制造商 ID2
    MOV     R6, A
    SETB    SS

    CLR     G_FLASHOK
    CJNE    R7, #9DH, $                 //检测是否为 PM25LVxx 系列的 Flash
```

```
    CJNE    R6, #7FH, $
    SETB    G_FLASHOK

FLASHERROR:
    MOV     C, G_FLASHOK
    RET

/*************************************************
检测 Flash 的忙状态
入口参数: 无
出口参数: CY
    0 : Flash 处于空闲状态
    1 : Flash 处于忙状态
*************************************************/
ISFLASHBUSY:
    CLR     SS
    MOV     A, #SFC_RDSR            //发送读取状态命令
    LCALL   SPISHIFT
    LCALL   SPISHIFT                //读取状态
    SETB    SS
    MOV     C, ACC.0                //状态值的 Bit0 即为忙标志
    RET

/*************************************************
使能 Flash 写命令
入口参数: 无
出口参数: 无
*************************************************/
FALSHWRITEENABLE:
    LCALL   ISFLASHBUSY
    JC      $-3                     //Flash 忙检测
    CLR     SS
    MOV     A, #SFC_WREN            //发送写使能命令
    LCALL   SPISHIFT
    SETB    SS
    RET

/*************************************************
擦除整片 Flash
入口参数: 无
出口参数: 无
*************************************************/
FLASHERASE:
    JNB     G_FLASHOK, FEREXIT
    LCALL   FALSHWRITEENABLE        //使能 Flash 写命令
    CLR     SS
```

```
    MOV     A, #SFC_CHIPER          //发送片擦除命令
    LCALL   SPISHIFT
    SETB    SS

FEREXIT:
    RET

/************************************************
从 Flash 中读取数据
入口参数:
    ADDR : FALSH 地址参数
    SIZE :  数据块大小
    DPTR :  缓冲区首地址
出口参数:  无
************************************************/
FLASHREAD:
    JNB     G_FLASHOK, FRDEXIT
    LCALL   ISFLASHBUSY
    JC      $-3                     //Flash 忙检测
    CLR     SS

    MOV     A, #SFC_FASTREAD        //使用快速读取命令
    LCALL   SPISHIFT

    MOV     A, ADDR+2               //设置起始地址
    LCALL   SPISHIFT
    MOV     A, ADDR+1
    LCALL   SPISHIFT
    MOV     A, ADDR+0
    LCALL   SPISHIFT

    LCALL   SPISHIFT                //需要空读一个字节

FRDNEXTBYTE:
    MOV     A, SIZE
    ORL     A, SIZE+1
    ORL     A, SIZE+2
    JZ      FRDEND

    LCALL   SPISHIFT                //自动连续读取并保存
    MOVX    @DPTR, A
    INC     DPTR
    MOV     A, ADDR+0
    ADD     A, #1
    MOV     ADDR+0, A
    MOV     A, ADDR+1
```

```
    ADDC    A, #0
    MOV     ADDR+1, A
    MOV     A, ADDR+2
    ADDC    A, #0
    MOV     ADDR+2, A
    CLR     C
    MOV     A, SIZE+0
    SUBB    A, #1
    MOV     SIZE+0, A
    MOV     A, SIZE+1
    SUBB    A, #0
    MOV     SIZE+1, A
    MOV     A, SIZE+2
    SUBB    A, #0
    MOV     SIZE+2, A
    LJMP    FRDNEXTBYTE

FRDEND:
    SETB    SS

FRDEXIT:
    RET

/***********************************************
写数据到 Flash 中
入口参数:
    ADDR : 地址参数
    SIZE : 数据块大小
    DPTR : 缓冲需要写入 Flash 的数据
出口参数: 无
***********************************************/
FLASHWRITE:
    JNB     G_FLASHOK, FWREXIT

FWRNEXTPAGE:
    MOV     A, SIZE
    ORL     A, SIZE+1
    ORL     A, SIZE+2
    JZ      FWREXIT

    LCALL   FALSHWRITEENABLE        //使能 Flash 写命令
    CLR     SS
    MOV     A, #SFC_PAGEPROG        //发送页编程命令
    LCALL   SPISHIFT

    MOV     A, ADDR+2               //设置起始地址
```

```
    LCALL   SPISHIFT
    MOV     A, ADDR+1
    LCALL   SPISHIFT
    MOV     A, ADDR+0
    LCALL   SPISHIFT

FWRNEXTBYTE:
    MOV     A, SIZE
    ORL     A, SIZE+1
    ORL     A, SIZE+2
    JZ      FRDEND
    MOVX    A, @DPTR
    LCALL   SPISHIFT                    //连续页内写
    INC     DPTR
    MOV     A, ADDR+0
    ADD     A, #1
    MOV     ADDR+0, A
    MOV     A, ADDR+1
    ADDC    A, #0
    MOV     ADDR+1, A
    MOV     A, ADDR+2
    ADDC    A, #0
    MOV     ADDR+2, A
    CLR     C
    MOV     A, SIZE+0
    SUBB    A, #1
    MOV     SIZE+0, A
    MOV     A, SIZE+1
    SUBB    A, #0
    MOV     SIZE+1, A
    MOV     A, SIZE+2
    SUBB    A, #0
    MOV     SIZE+2, A
    MOV     A, ADDR+0
    JZ      FWREND
    LJMP    FWRNEXTBYTE

FWREND:
    SETB    SS
    LJMP    FWRNEXTPAGE

FWREXIT:
    RET

    END
```

26.9 SPI 的特别注意事项(仅针对以 15F 和 15L 开头的单片机)

----只支持 SPI 主机模式，不支持 SPI 从机模式

STC 单片机中以 15F 和 15L 开头且有 SPI 功能的单片机（如 STC15F2K60S2 型号及 STC15L408AD 单片机）的 SPI 从机模式暂不能使用，但它们的 SPI 主机模式可正常使用。因此，建议用户不要使用以 15F 和 15L 开头且有 SPI 功能的单片机的 SPI 从机模式。

注意，以 15W 开头的单片机不存在上述问题，以 15W 开头且有 SPI 功能的单片机既支持 SPI 主机模式，也支持 SPI 从机模式。如，STC15W408S、STC15W1K16S 等型号单片机既支持 SPI 主机模式，也支持 SPI 从机模式。

27 编译器(汇编器)/ISP 编程器(烧录)/仿真器说明

27.1 编译器/汇编器的说明及头文件

STC 单片机应使用何种编译器/汇编器:

1.任何老的编译器/汇编器都可以支持，流行用 Keil C51

2.把 STC 单片机当成 Intel 的 8052/87C52/87C54/87C58 或 Phiips 的 P87C52/P87C54/P87C58 编译，头文件包含<rcg51.h>即可。新增特殊功能寄存器用 sfr 声明，新增特殊功能寄存器位用 sbit 声明。例如，对要用到的新增 P4 口特殊功能寄存器及特殊功能寄存器位的地址声明如下：

C 语言地址声明：

```
sfr     P4 = 0xC0;      //8 bit Port4 P4.7 P4.6 P4.5 P4.4 P4.3 P4.2 P4.1 P4.0 1111,1111
sfr     P4M0 = 0xB4;    // 0000,0000
sfr     P4M1 = 0xB3;    // 0000,0000
sbit    P40 = P4^0;
sbit    P41 = P4^1;
sbit    P42 = P4^2;
sbit    P43 = P4^3;
sbit    P44 = P4^4;
sbit    P45 = P4^5;
sbit    P46 = P4^6;
sbit    P47 = P4^7;
```

汇编语言地址声明：

```
P4      EQU     0C0H    ; or P4         DATA    0C0H
P4M1    EQU     0B3H    ; or P4M1       DATA    0B3H
P4M0    EQU     0B4H    ; or P4M1       DATA    0B4H
P40     EQU     0C0H
P41     EQU     0C1H
P42     EQU     0C2H
P43     EQU     0C3H
P44     EQU     0C4H
P45     EQU     0C5H
P46     EQU     0C6H
P47     EQU     0C7H
```

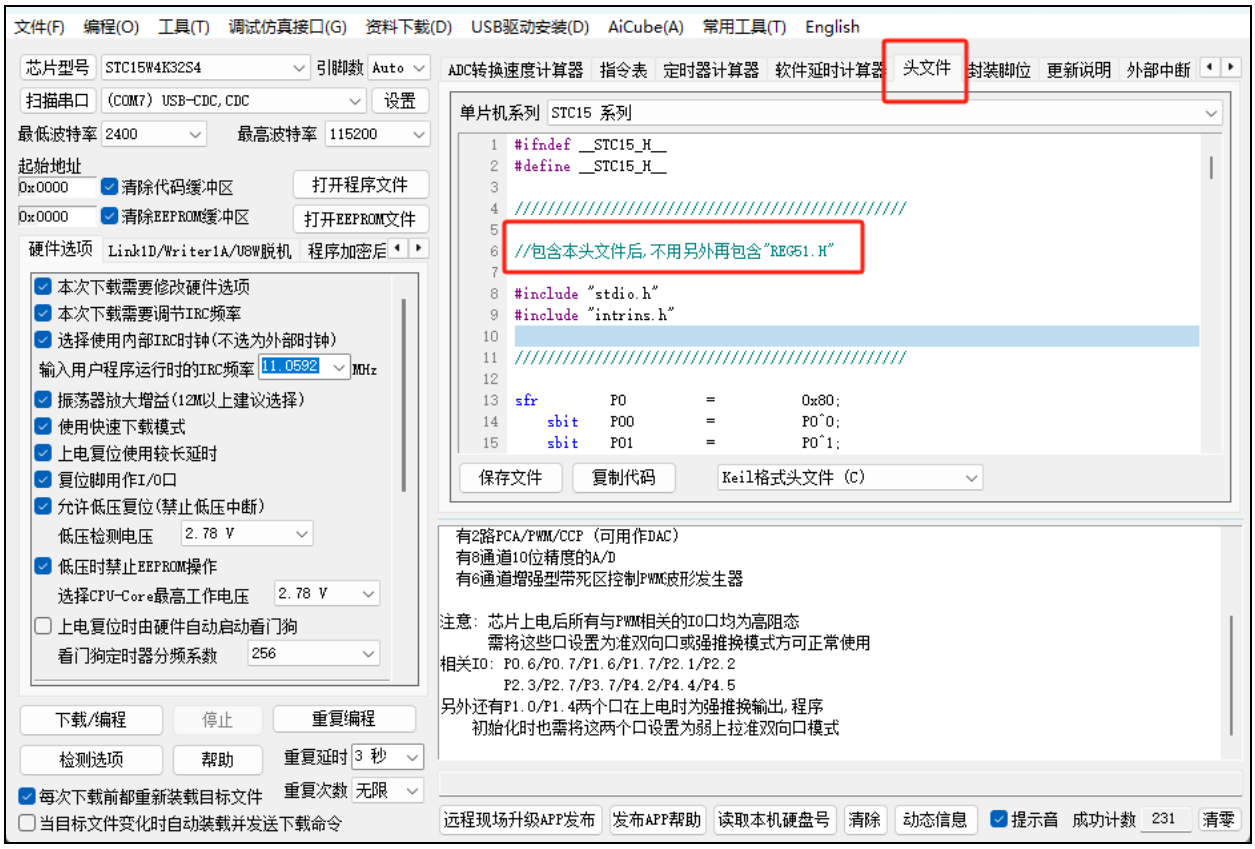
;以上为 P4 口新增功能寄存器的地址声明

当然如果新增功能寄存器在用户程序中用不到的话，也可以不声明。

【注意】: 如果用户所需包含的头文件不在 Keil C 的系统目录(C:keil\C51\INC)下，用" "将该头文件名包含进来，如果所需的头文件在 Keil C 的系统目录下，既可用" "，也可用< >包含进来。

对于 STC 部分单片机，可以到 STCAI 官方网站 wwwSTCAI.com 下载用户所使用的相应系列单片机的头文件(如果找不到所需的文件用 ctrl+F 查找)，STC15 系列单片机还可以用最新的 ISP 下载工具 AIapp-ISP 最新版本生成相应的头文件并保存，如下图所示。在编译具体 STC 系列单片机程序时，这些

相应的头文件可以代替“reg51.h”。

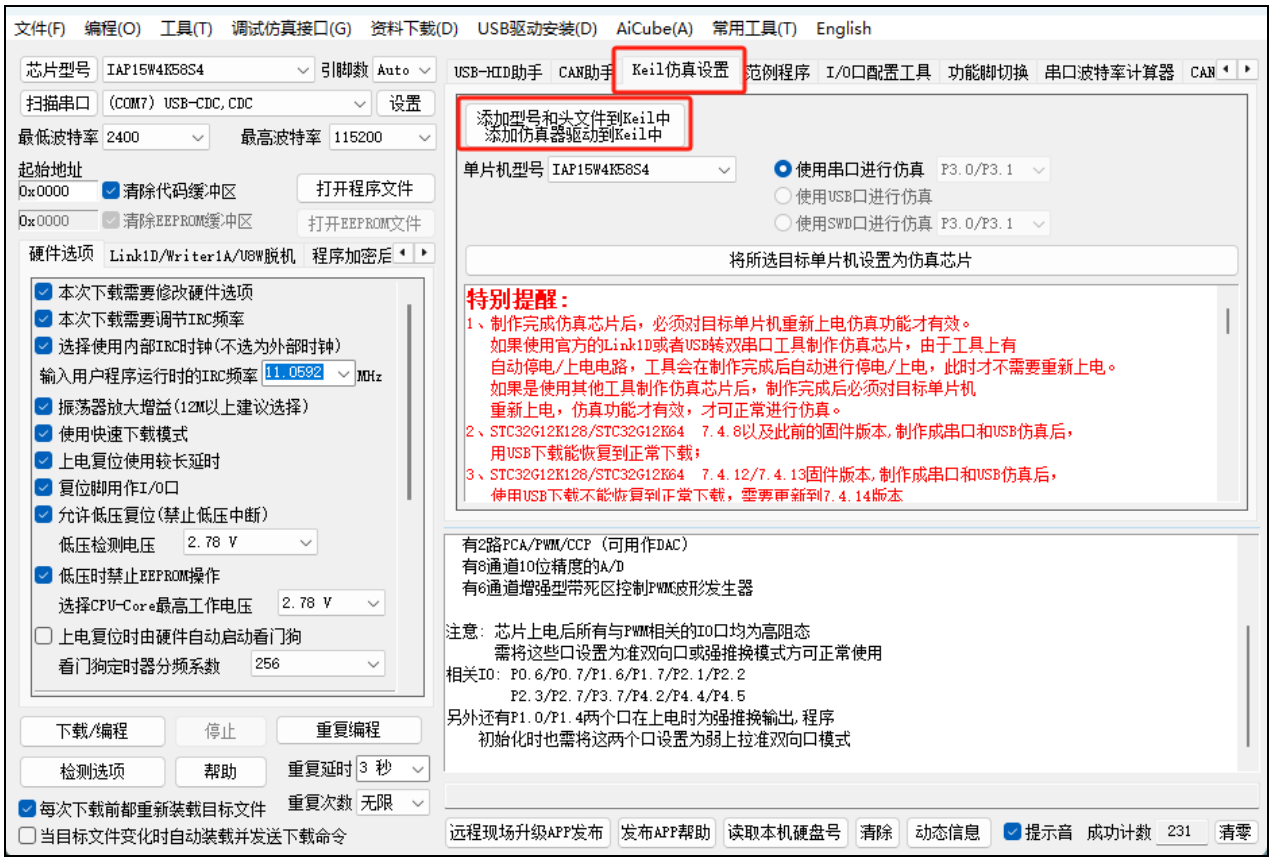


Keil C51 集成开发环境有许多版本，而对于 8051 单片机最常用的版本有 Keil μVision2、Keil uVision3 及 Keil μVision4.

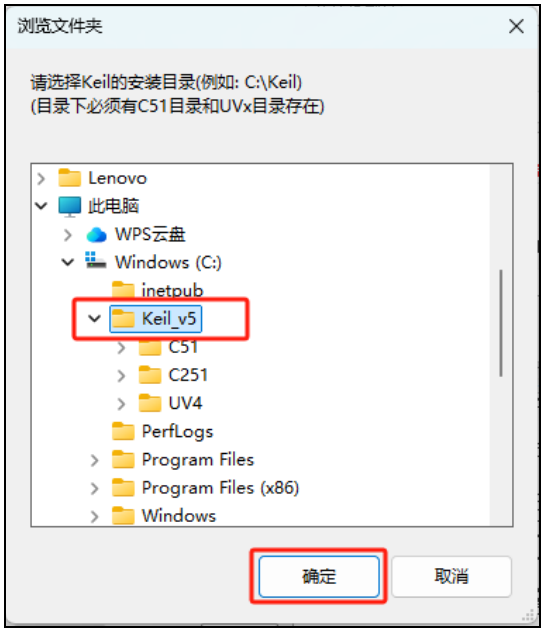
【注意】: 由于 STC 系列单片机是新发展的芯片，一般情况下在 Keil uVision 设备库中没有 STC 系列单片机。在编辑、编译 STC 系列单片机应用程序时，可选任何厂家的 51 或 52 系列单片机，再用汇编或 C 语言对 STC 系列单片机新增特殊功能寄存器进行定义，也可以通过 AIapp-ISP 下载编程工具将 STC 型号 MCU 添加到 Keil μVision5 或 Keil μVision4 或 Keil μVision3 或 KeiluVision2 的设备库中。

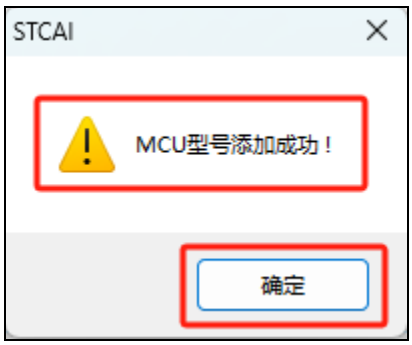
如果用户需在 Keil μVision5 或 Keil μVision4 或 Keil μVision3 或 Keil uVision2 的设备库中增加 STC 型号 MCU，则可按如下步骤进行设置：

（1）打开 AIapp-ISP 下载编程工具的最新软件 AIapp-ISP v6.95N，选择“Keil 仿真设置”页面，点击该页面中的【添加 MCU 型号到 Keil 中】按钮。



在弹出的“浏览文件夹”对话框中选择 Keil 安装目录(一般可能为“C:\keil”)，然后单击【确定】，这样就将 STC 型号的 MCU 成功添加到 Keil pVision2 设备库中了。





27.2 USB 型联机/脱机下载工具 U8W/U8W-Mini/U8/U8-Mini

U8W/U8W-Mini 及 U8/U8-Mini 是一款集在线联机下载和脱机下载于一体的编程工具系列。其中，U8 编程工具分 5V 工具和 3.3V 工具，分别为 U8-5V 及 U8-3.3V。U8W/U8W-Mini 及 U8/U8Mini 的应用范围可支持 STC 目前的全部系列的 MCU,Flash 程序空间和 EEPROM 数据空间不受限制。支持包括如下和即将推出的 STC 全系列芯片:

STC15W4K32S4 系列
STC15F2K60S2/STC15L2K60S2 系列
STC15W204S 系列
STC15W408AS 系列
STC15W404S 系列
STC15W1K16S 系列
STC15F408AD/STC15L408AD 系列
STC15F104W/STC15L104W 系列
STC15F104E/STC15L104E
STC15F204EA/STC15L204EA
STC10Fxx/STC10Lxx 系列
STC11Fxx/STCH1Lxx 系列
STC12C5Axx/STC12LE5Axx 系列
STC12C52xx/STC12LE52xx 系列
STC12C56xx/STC12LE56xx 系列
STC12C54xx/STC12LE54xx 系列

STC12Cx052/STC12Cx052AD/STC12LEx052/STC12LEx052AD 系列 STC90xx/STC89xx 系列脱机下载工具可以在脱离电脑的情况下进行下载工作，可用于批量生产和远程升级。脱机下载板可支持自动增量、下载次数限制以及用户程序加密后传输等多种功能。

U8W/U8W-Mini 工具及 U8/U8-Mini 工具的实物图如下页所示。

下图为 U8W 工具的正反面图以及 U8W-Mini 的正反面图：



目前的 U8 系列工具均分为 5V 工具和 3.3V 工具两种，本文以 U8 系列具 5V 工具(U8-5V)为例，下图为 U8 的 5V 工具(U8-5V)的正反面图以及 U8-Mini 的正反面图：



另外还有如下的一些线材与工具相搭配使用，如：

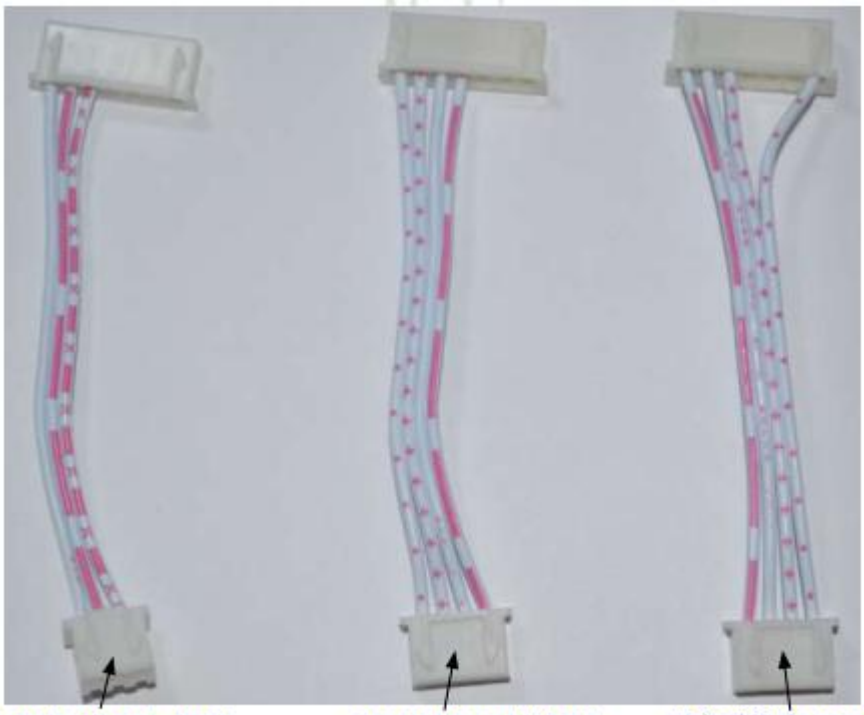
（1）两头公的 USB 连接线(如下图左所示）及 USB-Micro 连接线(如下图右所示)：





【注意】: 此 USB 线为我公司特别定制的 USB 加强线，可确保直接用 USB 供电时能够下载成功。而市面上一些比较劣质的两头公的 USB 线，内阻太大而导致压降很大（如 USB 空载时的电压为 5.0V 左右，当使用劣质的 USB 线连接 U8W/U8W-Mini/U8/U8-Mini，到我们的下载板上的电压可能降到 4.2V 或者更低，从而导致芯片处于复位状态而无法成功下载）

（2）U8W/U8W-Mini/U8/U8-Mini 与用户系统连接的下载连接线(即 U8W/U8W-Mini/U8/U8-Mini 与用户板上的目标单片机的连接线)，如下图所示



U8W/U8W-Mini/U8/U8-Mini与用户系统各自独立供电的连接线

U8W/U8W-Mini/U8/U8-Mini给用户系统供电的连接线

用户系统给U8W/U8W-Mini/U8/U8-Mini供电的连接线

27.2.1 如何安装下载工具 U8W/U8W-Mini/U8/U8-Mini 的驱动程序

U8W/U8W-Mini/U8/U8-Mini 下载板上使用了一颗 CH340 的 USB 转串口通用芯片。这样可以省去部分没有串口的电脑必须额外买一条 USB 转串端口才可下载的麻烦。但 CH340 和其它 USB 转串端口一样，在使用之前必须先安装驱动程序。驱动程序可以进行手动安装，也可以自动安装。

1、手动安装 USB 型联机/脱机下载工具 U8W/U8W-Mini/U8/U8-Mini 的驱动程序

在 STC 的官方网站上或在最新的 AIapp-ISP 下载软件中手动下载驱动程序，驱动的下载链接为：U8 编程器 USB 转串口驱动 （http://www.stcai.com/STCISP/CH341SER.exe）。网站上及 AIapp-ISP 下载软件上的驱动地址如下图所示：



驱动程序下载到本机后，直接双击可执行程序并运行，出现下图所示的界面，点击“安装”按钮开始自动安装驱动



直至弹出右边的画面表示驱动已成功安装



然后使用 STC 提供的 USB 连接线将 U8W/U8W-Mini/U8/U8-Mini 下载板连接到电脑，打开电脑的设备管理器，在端口设备类下面，如果有类似“USB-SERIAL CH340 (COMx)”的设备，就表示 U8W/U8W-Mini/U8/U8-Mini 可以正常使用了。如下图所示（不同的电脑，串口号可能会不同）

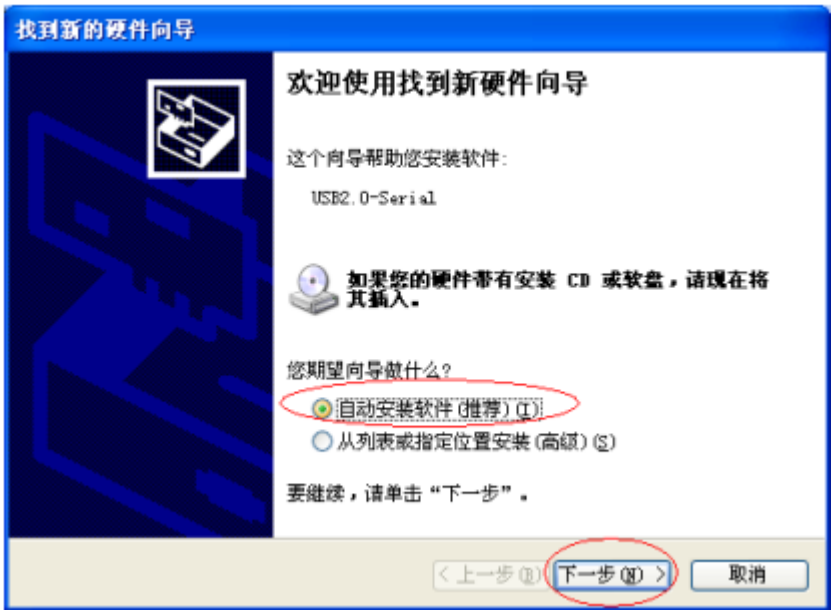


注意：在后面使用 AIapp-ISP 下载软件时，选择的串口号必须选择与此相对应的串口号，如下图所示



2、自动安装 USB 型联机/脱机下载工具 U8 的驱动程序

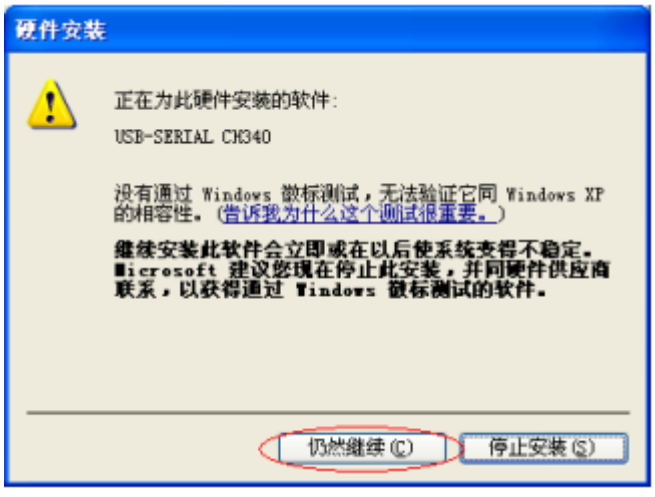
若用户所使用的 AIapp-ISP 下载软件为 V6.85K 及以上版本，则打开该 AIapp-ISP 下载软件时软件会自动检测本机的 U8W/U8W-Mini/U8/U8-Mini 驱动程序的安装情况，若没有安装驱动程序，软件会自动将相应的驱动程序复制到系统目录，此时拔出上一次插入的 U8W/U8W-Mini/U8/U8-Mini 工具并再次将其插上时，会出现如下提示框：



选“自动安装软件(推荐)(I)”选项，并点击【下一步】按钮，会出现如下画面：



在接下来出现的如下面的对话框中，选中【仍然继续】:



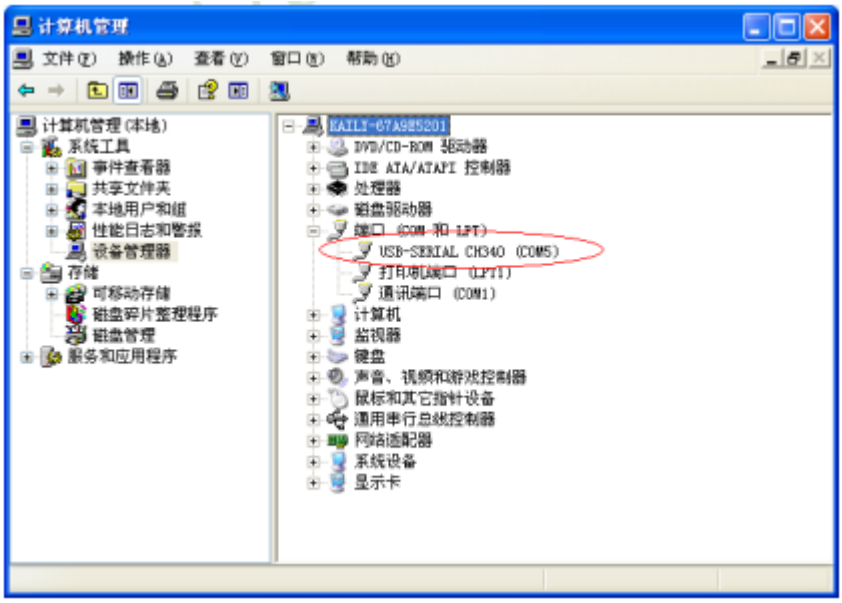
系统便会开始自动安装驱动，如下图所示：



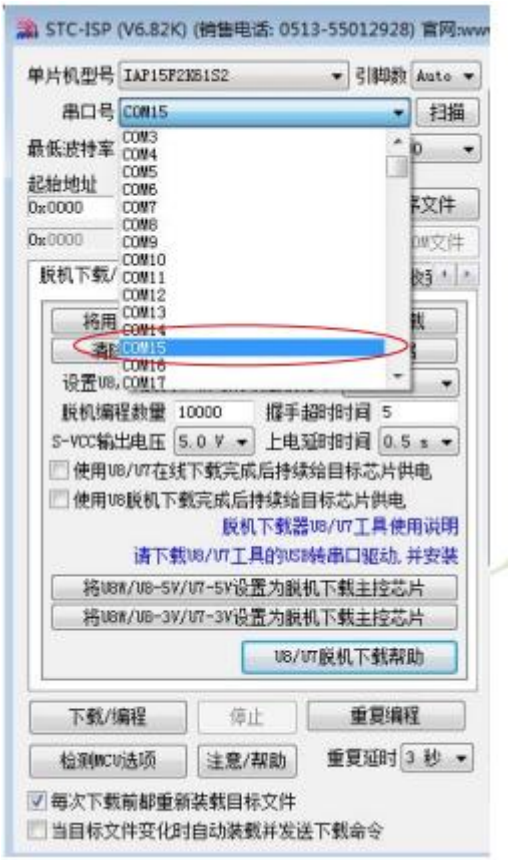
直至出现如下画面，点击【完成】按钮：



至此，8 的驱动程序便自动安装完成了。如手动安装驱动程序一样，也会如下图所示(不同的电脑，串口号可能会不同):

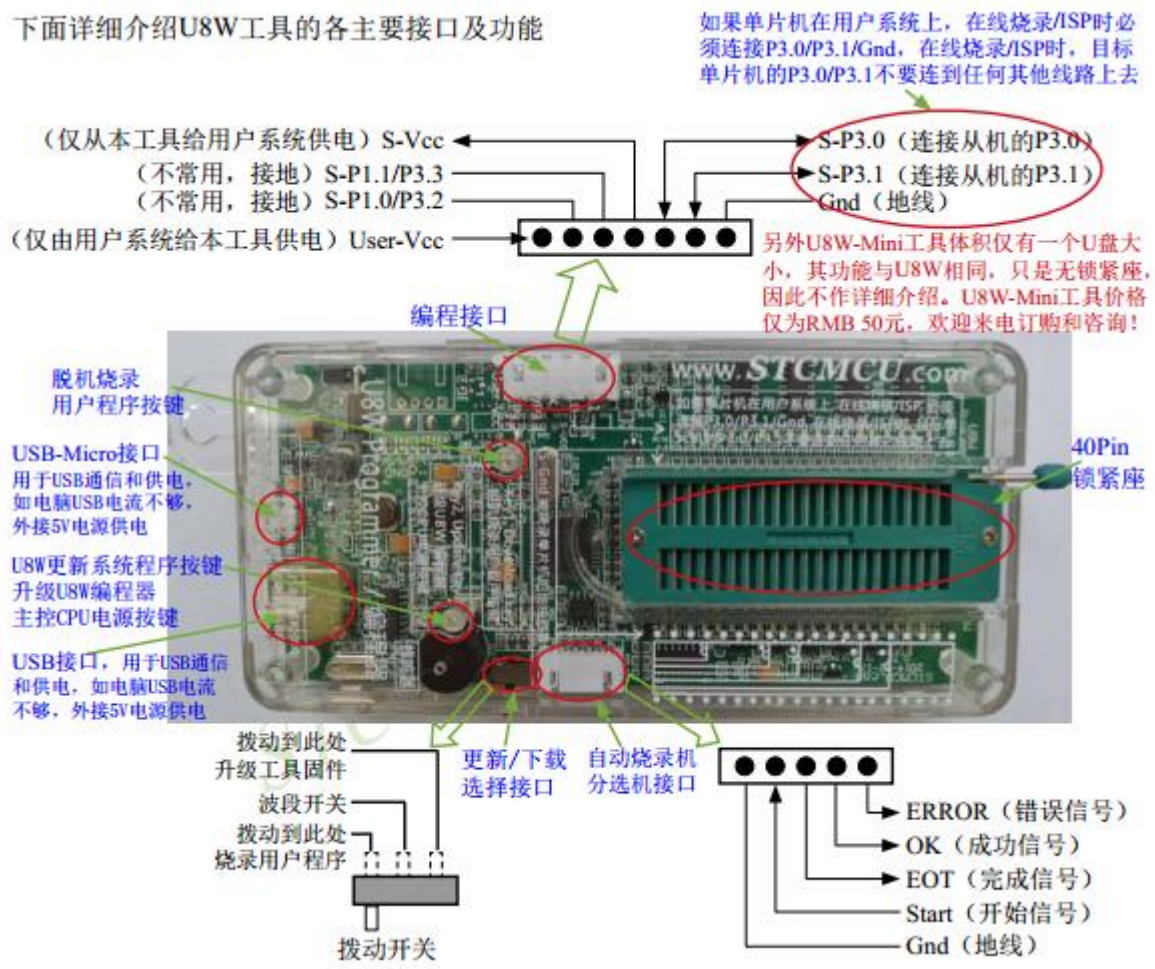


注意：在后面使用 AIapp-ISP 下载软件时，选择的串口号必须选择与此相对应的串口号，如下图所示：



27.2.2 USB 型联机/脱机下载工具 U8W 的功能介绍(价格为人民币 100 元)

下面详细介绍 **U8W** 工具的各主要接口及功能**如果单片机在用户系统上，在线烧录/ISP 时必须连接 P3.0/P3.1/Gnd，在线烧录/ISP 时，目标单片机的 P3.0/P3.1 不要连到任何其他线路上去**



编程接口：根据不同的供电方式，使用不同的下载连接线连接 U8W 下载板和用户系统。

U8W 更新系统程序按键：用于更新 U8W 工具，当有新版本的 U8W 固件时，需要按下此按键对 U8W 的主控芯片进行更新（注意：必须先将更新/下载选择接口上的拨动开关拨动到升级工具固件）。

脱机下载用户程序按钮：开始脱机下载按钮。首先 PC 将脱机代码下载到 U8W 板上，然后使用下载连接线将用户系统连接到 U8W，再按下此按钮即可开始脱机下载（每次上电时也会立即开始下载用户代码）。

更新/下载选择接口：当需要对 U8W 的底层固件进行升级时，需将此拨动开关拨动到升级工具固件处，当需通过 U8W 对目标芯片进行烧录程序，则需将拨动开关拨动到烧录用户程序处。

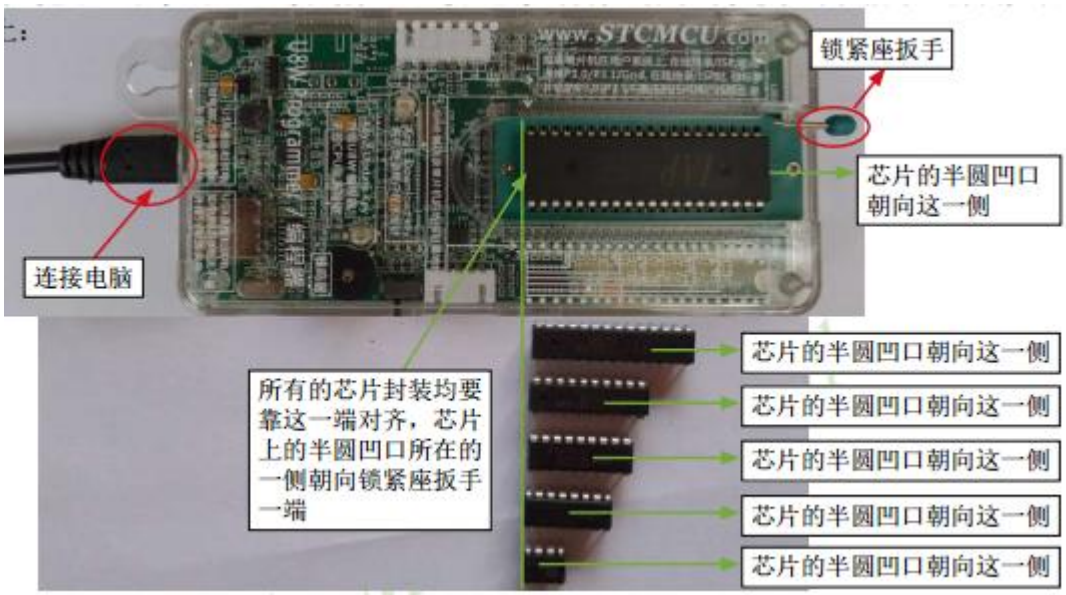
（拨动开关连接方式请参考上图）

自动烧录机/分选机接口：是用于控制自动烧录机/分选机进行自动生产的控制接口

27.2.3 U8W 的在线联机下载使用说明

27.2.3.1 目标芯片直接安装于U8W座锁紧上并由U8W连接电脑进行在线联机下载的说明

首先使用 STC 提供的 USB 连接线将 U8W 连接电脑，再将目标单片机按如下图所示的方向安装在 U8W 上：



然后在用 AIapp-ISP 下载软件下载程序时，在 AIapp-ISP 下载软件中选择正确的串口号(USB 转串口扩展的)，点击【下载/编程】按钮即可开始在线下载。



当信息框中有输出下载板的版本号信息以及外挂 Flash 的相应信息时，表示已正确检测到 U8W 下载

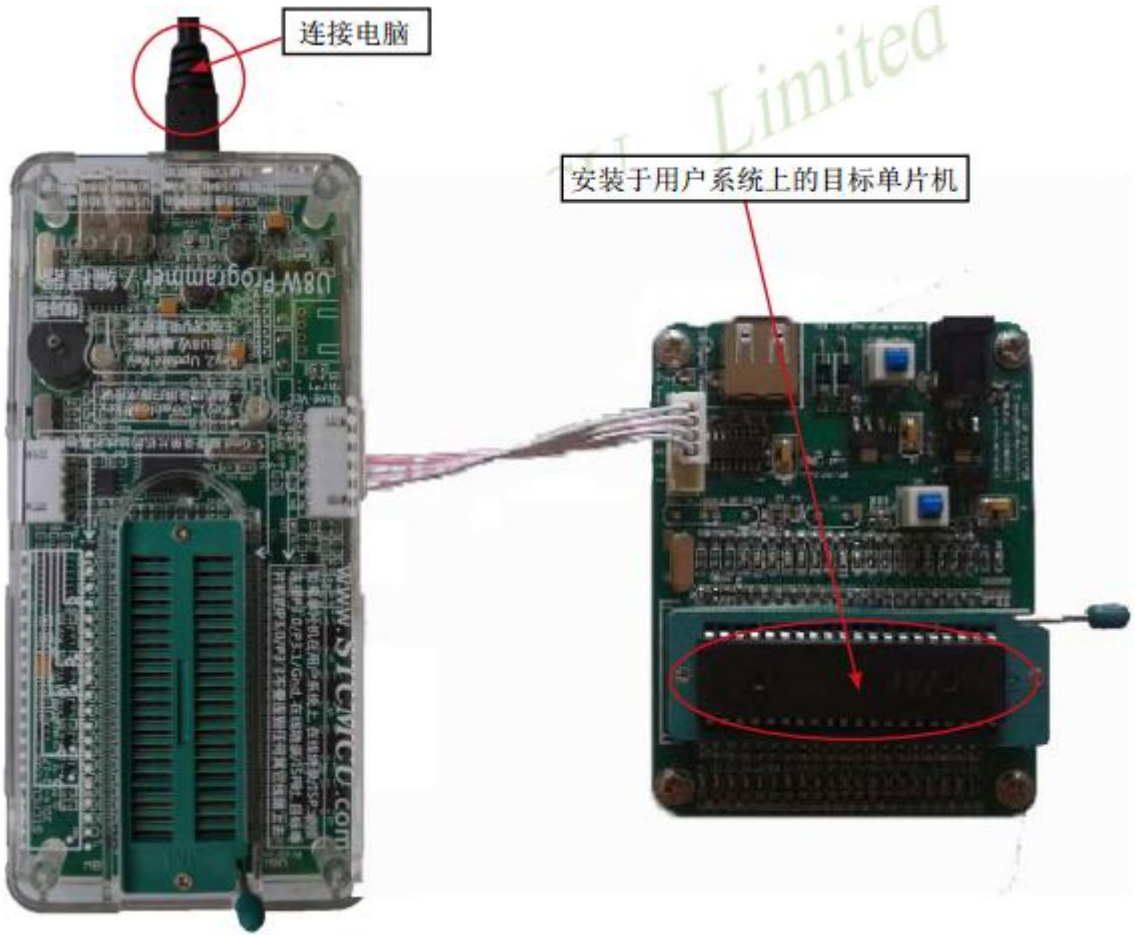
工具。

下载的过程中，U8W 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

建议用户用最新版本的 AIapp-ISP 下载软件“AIapp-ISP (V6.95G).exe”或以上版本（请随时留意 STCAI 官方网站 http://www.STCAI.com 中 AIapp-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCAI.com 中下载最新版本的软件使用）。

27.2.3.2 目标芯片通过用户系统引线连接 U8W 并由 U8W 连接电脑进行在线联机下载的说明

首先使用 STC 提供的 USB 连接线将 U8W 连接电脑，再将 U8W 通过下载线与用户系统的目标单片机相连接，连接方式如下图所示：



然后在用 AIapp-ISP 下载软件下载程序时，在 AIapp-ISP 下载软件中选择正确的串口号(USB 转串口扩展的)，点击【下载/编程】按钮即可开始在线下载。



当信息框中有输出下载板的版本号信息以及外挂 Flash 的相应信息时，表示已正确检测到 U8W 下载工具。

下载的过程中，U8W 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

建议用户用最新版本的 AIapp-ISP 下载软件“AIapp-ISP (V6.95G).exe”或以上版本（请随时留意 STCAI 官方网站 http://www.STCAI.com 中 AIapp-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCAI.com 中下载最新版本的软件使用）。

27.2.4 U8W 的脱机下载使用说明

27.2.4.1 目标芯片直接安装于 U8W 座锁紧上并通过 USB 连接电脑给 U8W 供电进行脱机下载

使用 USB 给 U8W 从而进行脱机下载的步骤如下：

（1）使用 STC 提供的 USB 连接线将 U8W 下载板连接到电脑，如下图：



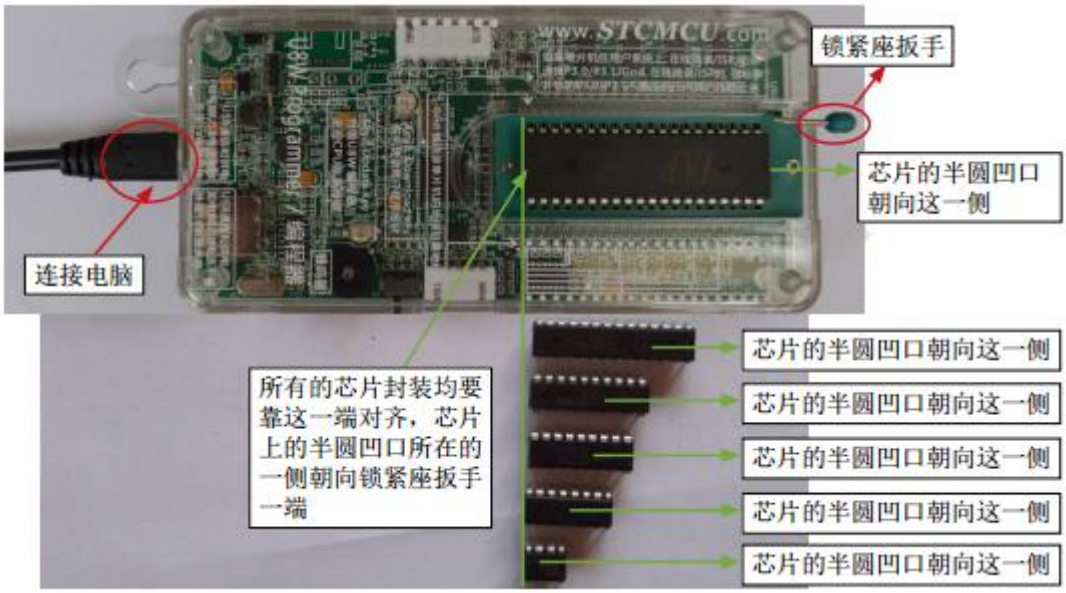
（2）在 ISP 下载软件“AIapp-ISP (V6.95G).exe”以上版本中按如下图所示的步骤进行设置：



按照上图的步骤，操作完成后，若下载成功则表示用户代码和相关的设置选项都已下载到 U8W 下载工具中。

建议用户用最新版本的 AIapp-ISP 下载软件“AIapp-ISP (V6.95G).exe”（请随时留意 STCAI 官方网站 http://www.STCAI.com 中 AIapp-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCAI.com 中下载最新版本的软件使用）。

（3）再将目标单片机如下图所示的方向放在 U8W 下载工具，如下图所示



（4）然后按下如下图所示的按钮后松开，即可开始脱机下载：



下载的过程中，U8W 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

27.2.4.2 目标芯片由用户系统引线连接 U8W 并通过 USB 连接电脑给 U8W 供电进行脱机下载

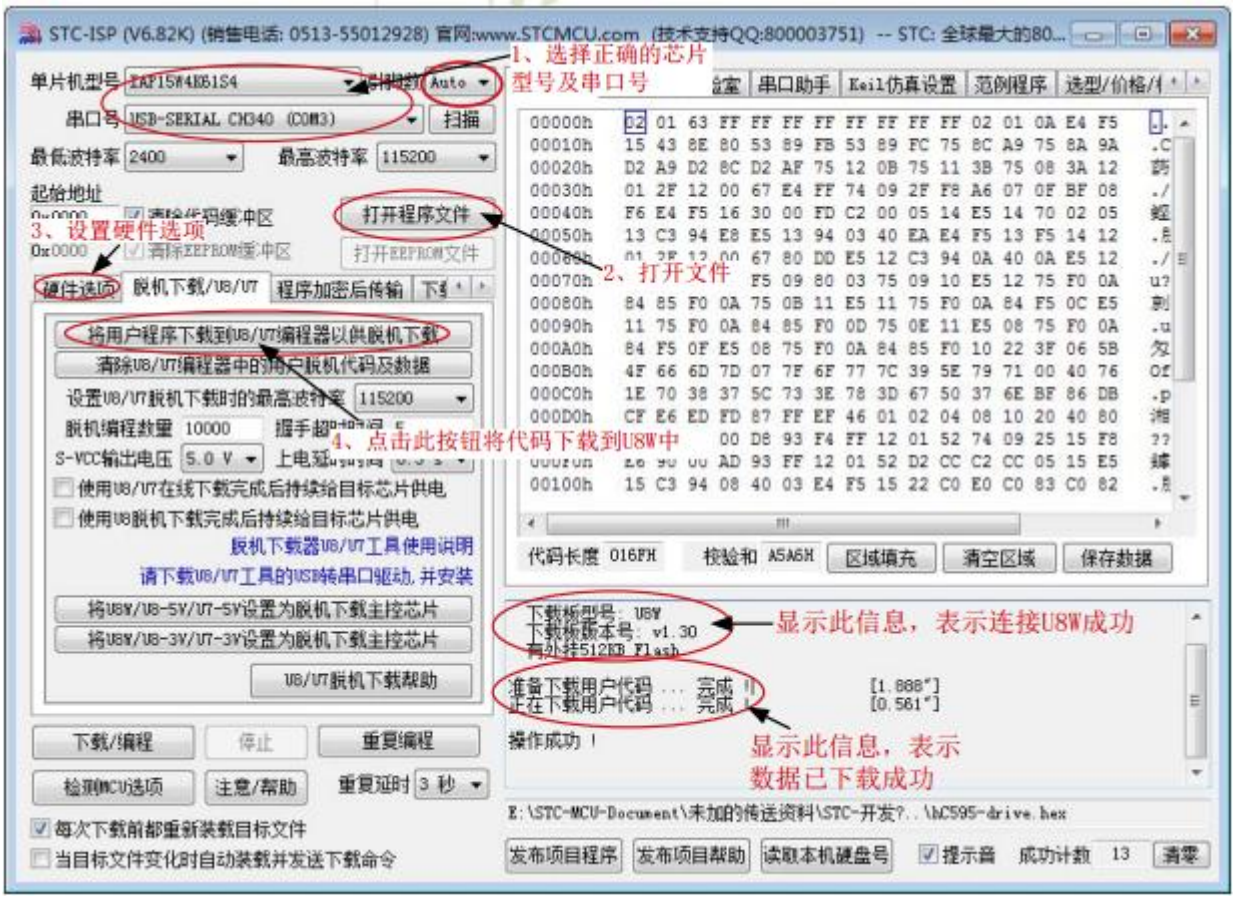
使用 USB 给 U8W 从而进行脱机下载的步骤如下：

（1）使用 STC 提供的 USB 连接线将 U8W 下载板连接到电脑，如下图:



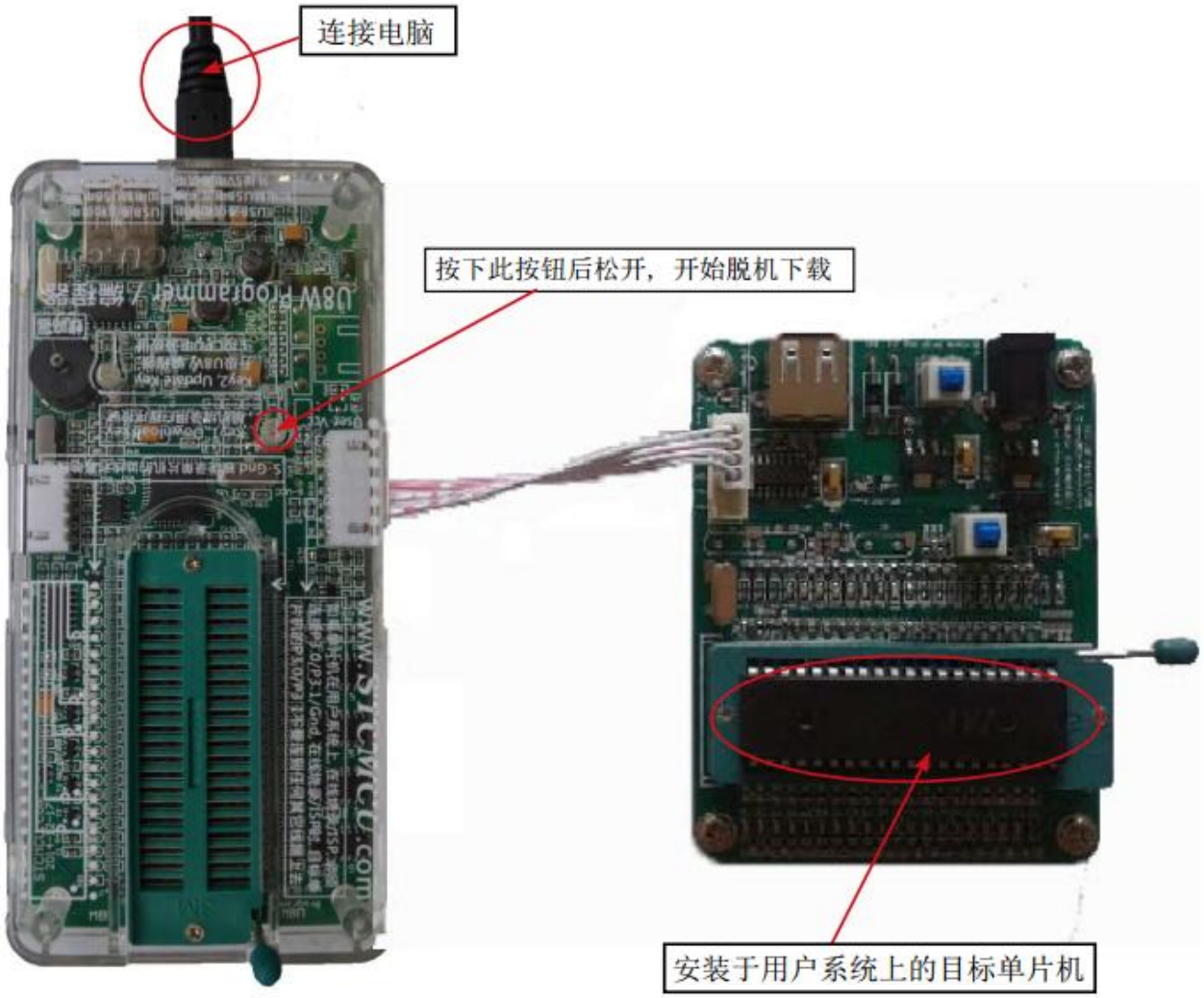
（2）在 ISP 下载软件“AIapp-ISP（V6.95G）.exe”以上版本中按如下图所示的步骤进行设置：

建议用户用最新版本的 AIapp-ISP 下载软件“AIapp-ISP（V6.95G）.exe”（请随时留意 STCAI 官方网站 http://www.STCAI.com 中 AIapp-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCAI.com 中下载最新版本的软件使用）。



按照上图的步骤，操作完成后，若下载成功则表示用户代码和相关的设置选项都已下载到 U8W 下载工具中。

（3）然后使用连接线连接电脑、将 U8W 下载工具以及用户系统（目标单片机）如下图所示的方式连接起来，并按下图所示的按钮后松开，即可开始脱机下载



下载的过程中，U8W 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

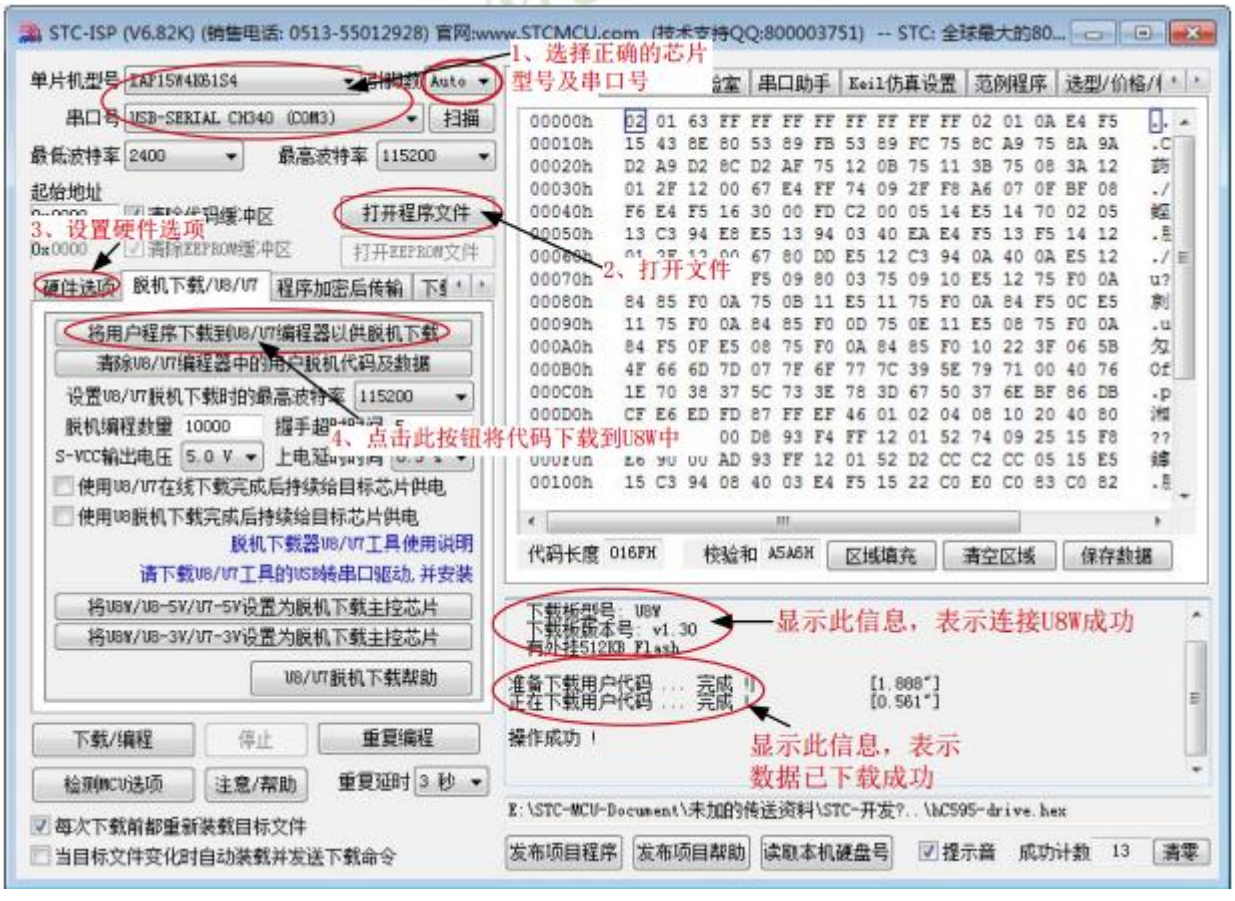
27.2.4.3 目标芯片由用户系统引线连接 U8W 并通过用户系统给 U8W 供电进行脱机下载

（1）首先使用 STC 提供的 USB 连接线将 U8W 下载板连接到电脑，如下图：



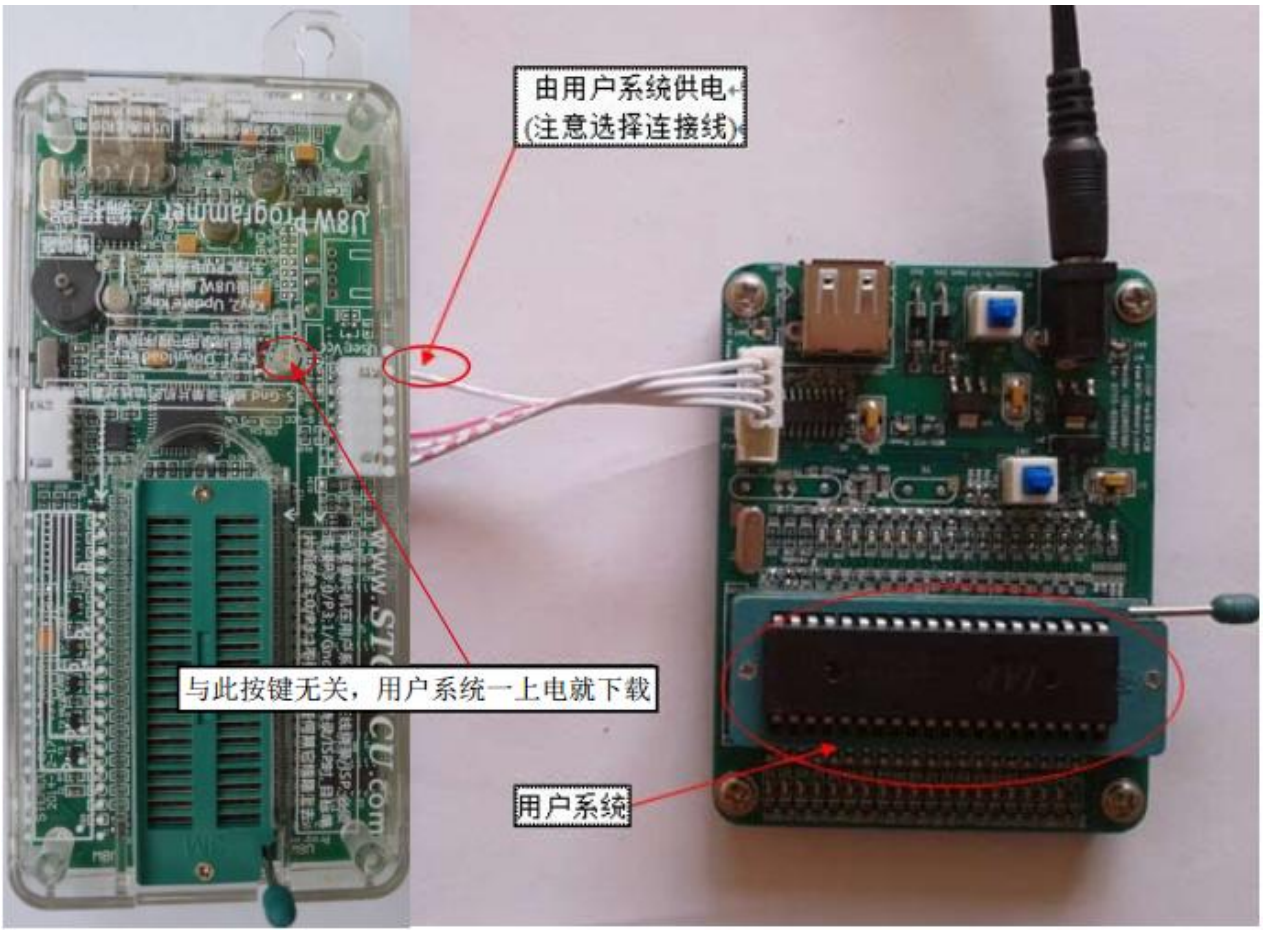
（2）在 ISP 下载软件“AIapp-ISP（V6.95G）.exe”以上版本中按如下图所示的步骤进行设置：

建议用户用最新版本的 AIapp-ISP 下载软件“AIapp-ISP（V6.95G）.exe”（请随时留意 STCAI 官方网站 http://www.STCAI.com 中 AIapp-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCAI.com 中下载最新版本的软件使用）。



按照上图的步骤，操作完成后，若下载成功则表示用户代码和相关的设置选项都已下载到 U8 下载工具中

（3）然后按下图所示的方式连接 U8W 与用户系统，并按下图中所示按钮后松开，即可开始脱机下载:



下载的过程中，U8W 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

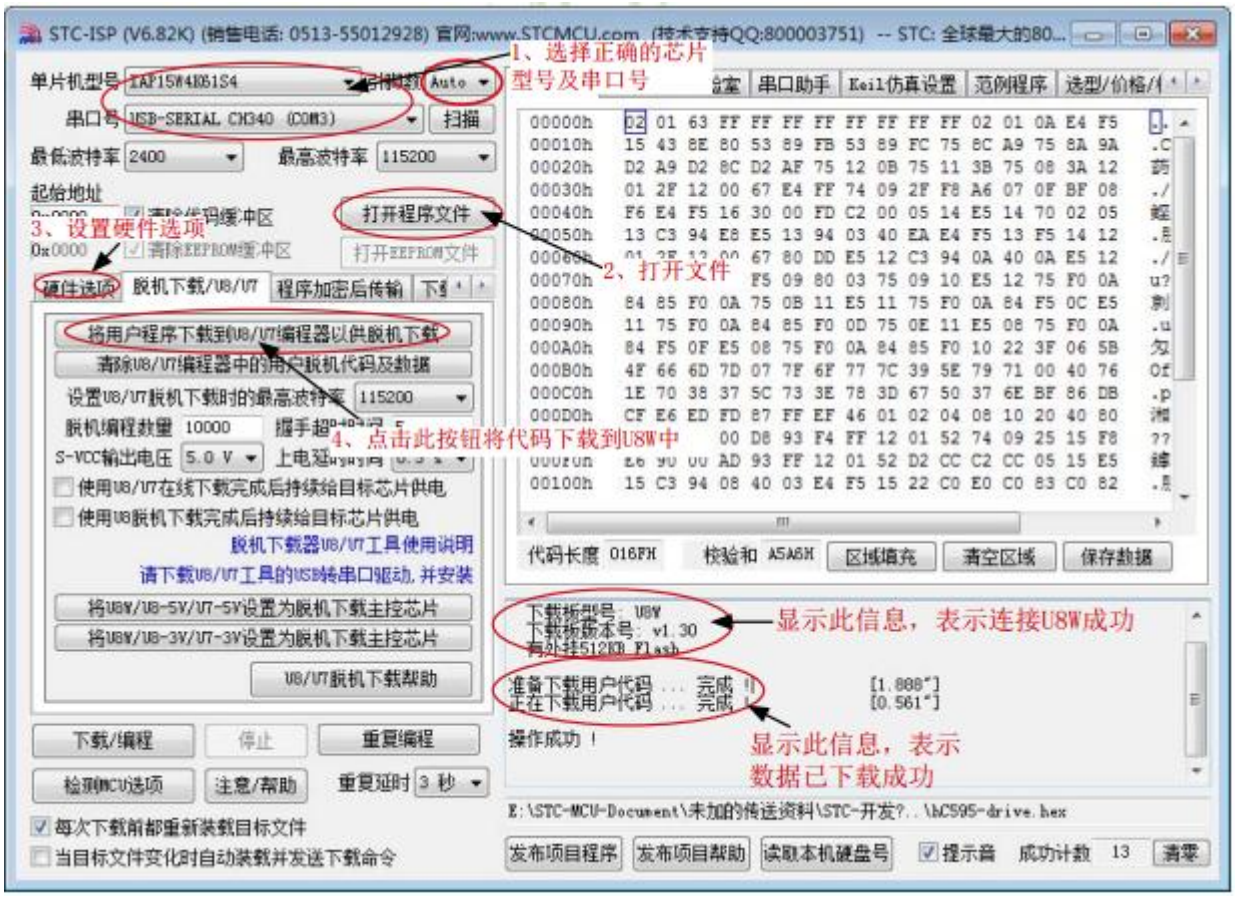
27.2.4.4 目标芯片由用户系统引线连接 U8W 且 U8W 与用户系统各自独立供电进行脱机下载

（1）首先使用 STC 提供的 USB 连接线将 U8W 下载板连接到电脑，如下图:



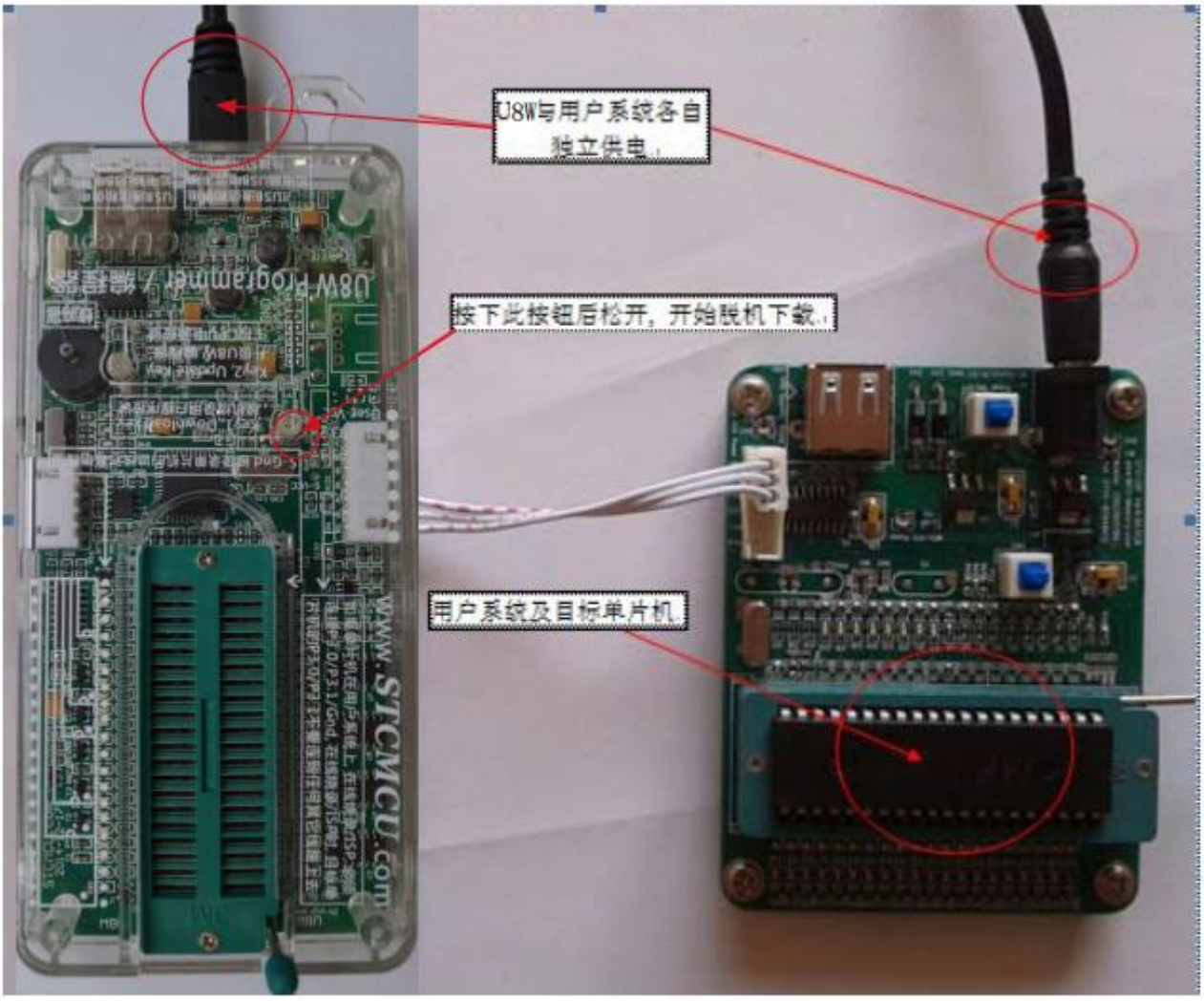
（2）在 ISP 下载软件“AIapp-ISP（V6.95G）.exe”以上版本中按如下图所示的步骤进行设置：

建议用户用最新版本的 AIapp-ISP 下载软件“AIapp-ISP（V6.95G）.exe”（请随时留意 STCAI 官方网站 http://www.STCAI.com 中 AIapp-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCAI.com 中下载最新版本的软件使用）。



按照上图的步骤，操作完成后，若下载成功则表示用户代码和相关的设置选项都已下载到 U8 下载工具中

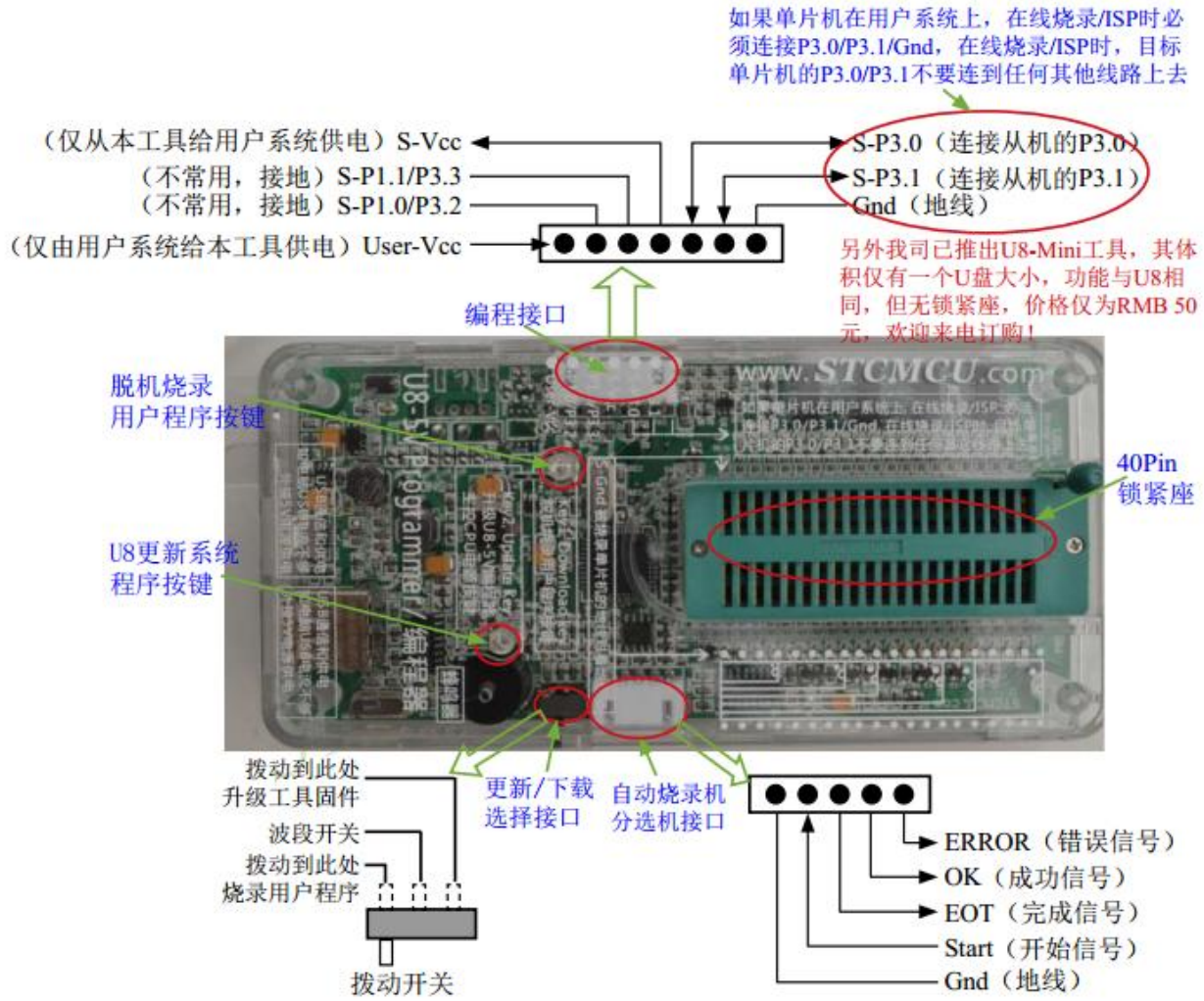
（3）然后按下图所示的方式连接 U8W 与用户系统，并将图中所示按钮先按下后松开，准备开始脱机下载，最后给用户系统上电/开电源，下载用户程序正式开始：



下载的过程中，U8W 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

27.2.5 USB 型联机/脱机下载工具 U8 的功能介绍（U8 的价格为人民币 100 元）

下面以 5V 工具为例，详细介绍 U8 工具的各主要接口及功能



编程接口：根据不同的供电方式，使用不同的下载连接线连接 U8 下载板和用户系统。

U8 更新系统程序按键：用于更新 U8 工具，当有新版本的 U8 固件时，需要按下此按键对 U8 的主控芯片进行更新（注意：必须先将更新/下载选择接口上的拨动开关拨动到升级工具固件）。

脱机下载用户程序按钮：开始脱机下载按钮。首先 PC 将脱机代码下载到 U8 板上，然后使用下载连接线将用户系统连接到 U8，再按下此按钮即可开始脱机下载（每次上电时也会立即开始下载用户代码）。

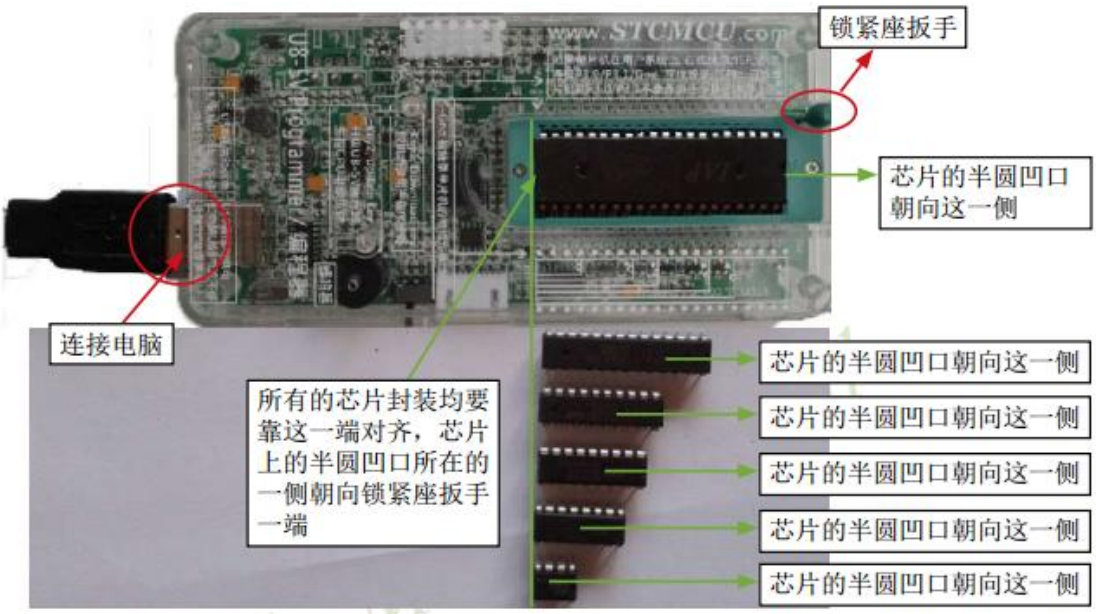
更新/下载选择接口：当需要对 U8 的底层固件进行升级时，需将此拨动开关拨动到升级工具固件处，当需通过 U8 对目标芯片进行烧录程序，则需将拨动开关拨动到烧录用户程序处。（拨动开关连接方式请参考上图）

自动烧录机/分选机接口：是用于控制自动烧录机/分选机进行自动生产的控制接口

27.2.6 U8 的在线联机下载使用说明

27.2.6.1 目标芯片直接安装于 U8 的座锁紧上并由 U8 连接电脑进行在线联机下载的说明

首先使用 STC 提供的 USB 连接线将 U8 连接电脑，再将目标单片机按如下图所示的方向安装在 U8 上：



然后在用 AIapp-ISP 下载软件下载程序时，在 AIapp-ISP 下载软件中选择正确的串口号（USB 转串口扩展的），点击【下载/编程】按钮即可开始在线下载。



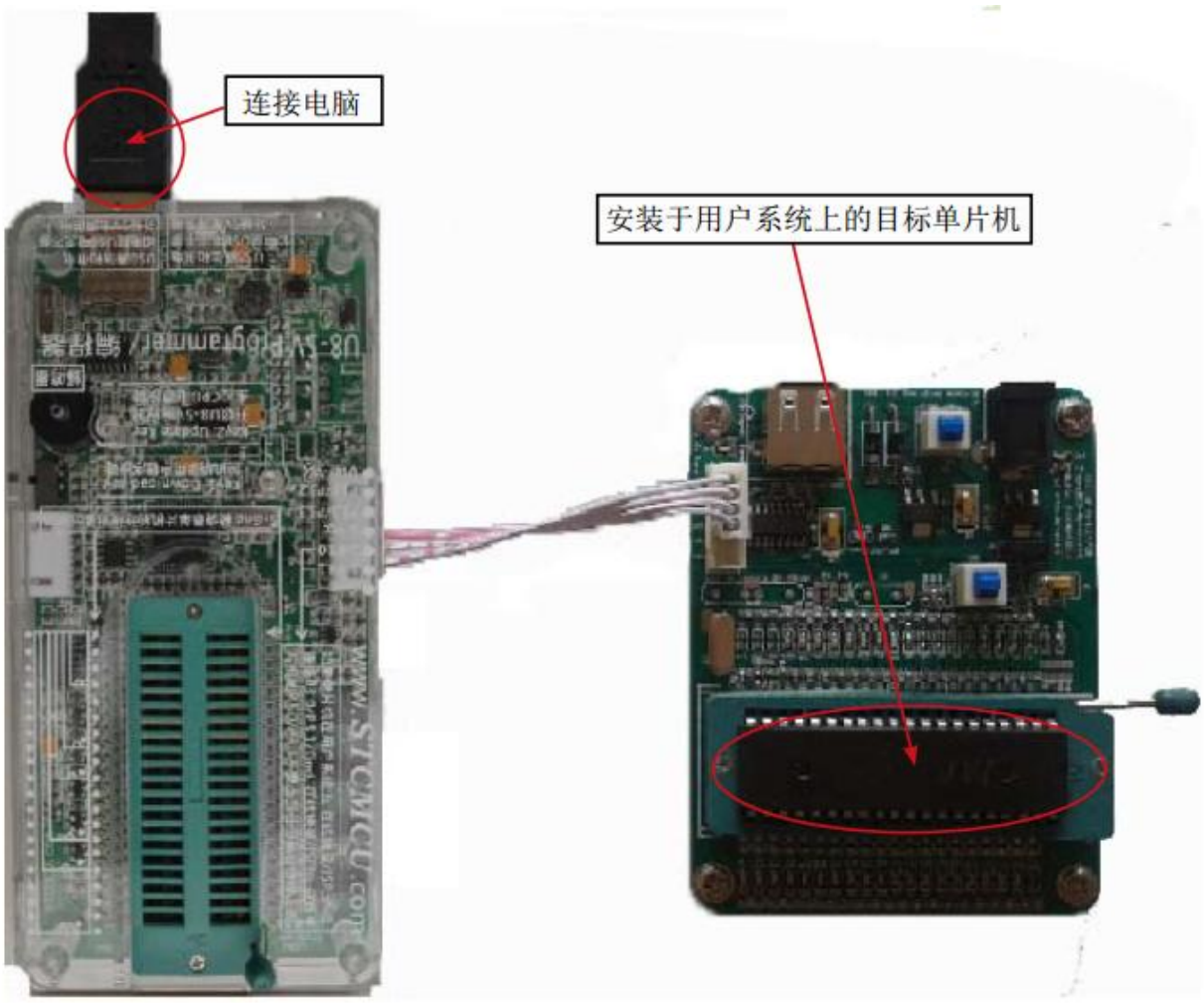
当信息框中有输出下载板的版本号信息以及外挂 Flash 的相应信息时，表示已正确检测到 U8 下载工具。

下载的过程中，U8 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

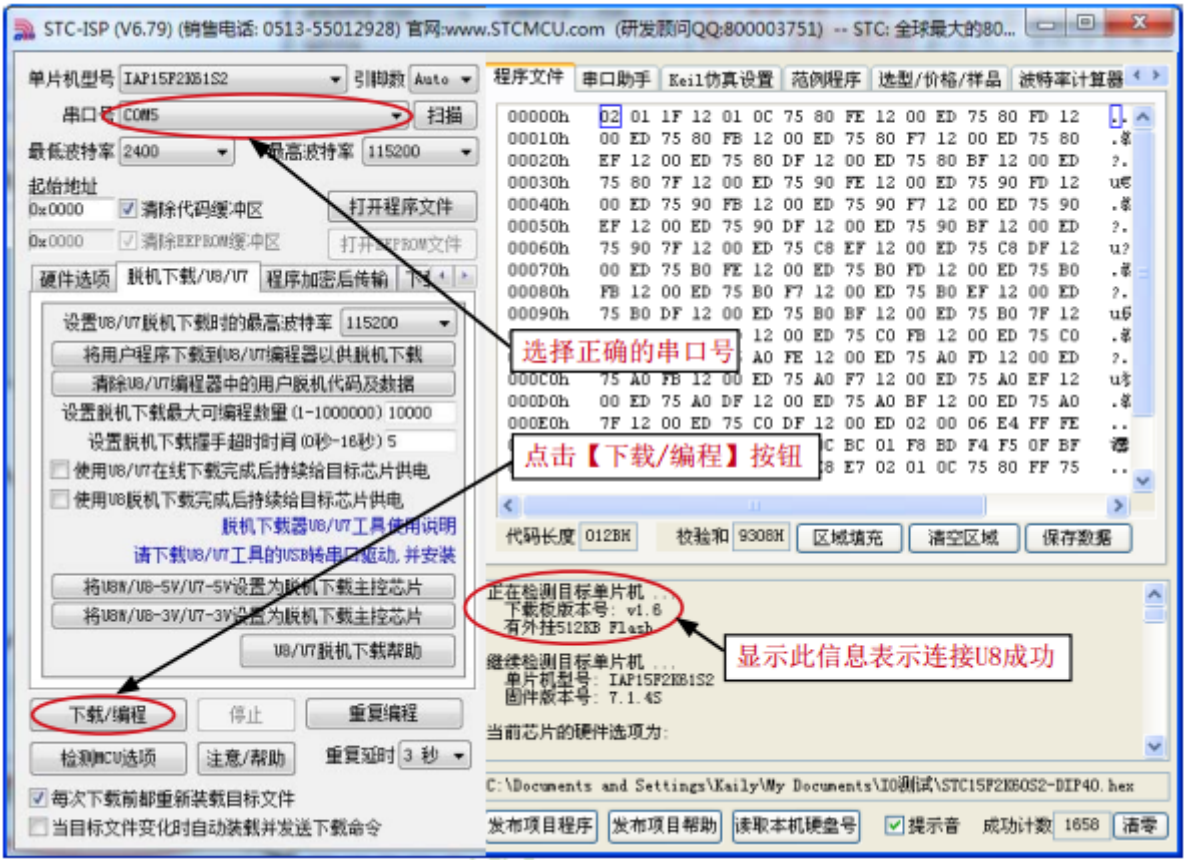
建议用户用最新版本的 AIapp-ISP 下载软件“AIapp-ISP（V6.95G）.exe”（请随时留意 STCAI 官方网站 http://www.STCAI.com 中 AIapp-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCAI.com 中下载最新版本的软件使用）。

27.2.6.2 目标芯片通过用户系统引线连接 U8 并由 U8 连接电脑进行在线联机下载的说明

首先使用 STCAI 提供的 USB 连接线将 U8 连接电脑，再将 U8 通过下载线与用户系统的目标单片机相连接，连接方式如下图所示：



然后在用 AIapp-ISP 下载软件下载程序时，在 AIapp-ISP 下载软件中选择正确的串口号（USB 转串口扩展的），点击【下载/编程】按钮即可开始在线下载。



当信息框中有输出下载板的版本号信息以及外挂 Flash 的相应信息时，表示已正确检测到 U8 下载工具。

下载的过程中，U8 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮，

建议用户用最新版本的 AIapp-ISP 下载软件“AIapp-ISP（V6.95G）.exe”（请随时留意 STCAI 官方网站 http://www.STCAI.com 中 AIapp-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCAI.com 中下载最新版本的软件使用）。

27.2.7 U8 的脱机下载使用说明

27.2.7.1 目标芯片直接安装于 U8 座锁紧上并通过 USB 连接电脑给 U8 供电进行脱机下载

使用 USB 给 U8 从而进行脱机下载的步骤如下:

（1）使用 STC 提供的 USB 连接线将 U8 下载板连接到电脑，如下图:



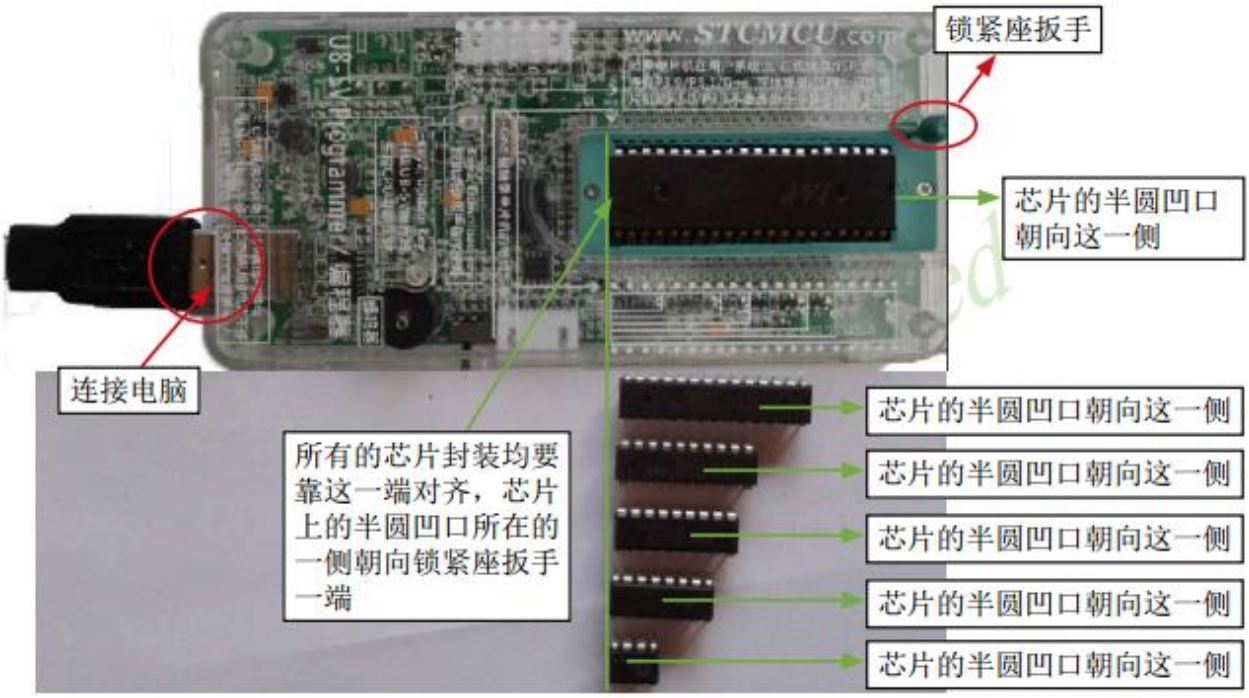
（2）在 ISP 下载软件“AIapp-ISP（V6.95G）.exe”以上版本中按如下图所示的步骤进行设置：



按照上图的步骤，操作完成后，若下载成功则表示用户代码和相关的设置选项都已下载到 U8 下载工具中。

建议用户用最新版本的 AIapp-ISP 下载软件“AIapp-ISP（V6.95G）.exe”（请随时留意 STCAI 官方网站 http://www.STCAI.com 中 AIapp-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCAI.com 中下载最新版本的软件使用）。

（3）再将目标单片机如下图所示的方向放在 U8 下载工具，如下图所示



（4）然后按下如下图所示的按钮后松开，即可开始脱机下载：



下载的过程中，U8 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

27.2.7.2 目标芯片由用户系统引线连接 U8 并通过 USB 连接电脑给 U8 供电进行脱机下载

使用 USB 给 U8 从而进行脱机下载的步骤如下:

（1）使用 STC 提供的 USB 连接线将 U8 下载板连接到电脑，如下图:



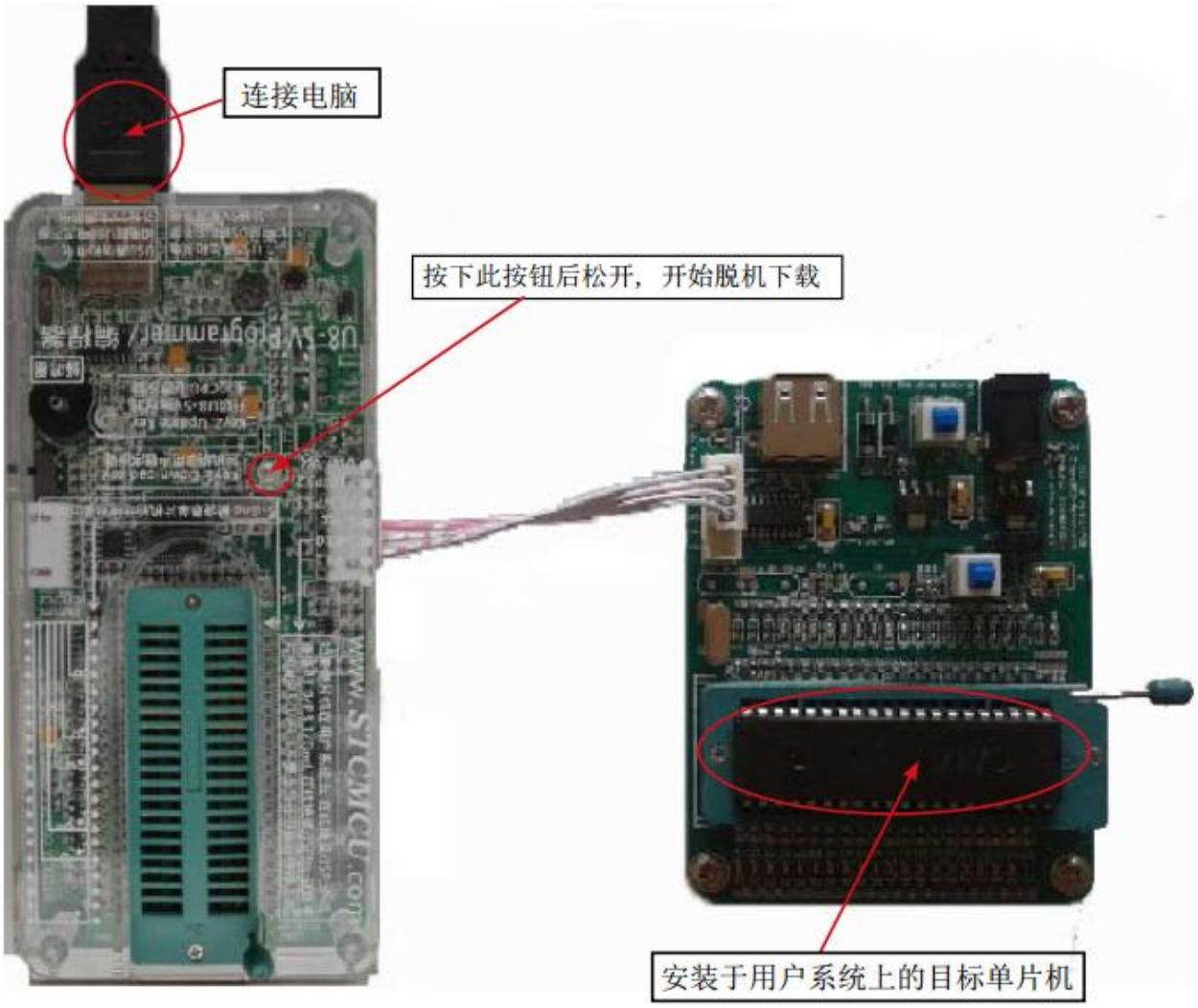
（2）在 ISP 下载软件“'AIapp-ISP（V6.95G）.exe”以上版本中按如下图所示的步骤进行设置：

建议用户用最新版本的 AIapp-ISP 下载软件“AIapp-ISP（V6.95G）.exe”（请随时留意 STCAI 官方网站 http://www.STCAI.com 中 AIapp-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCAI.com 中下载最新版本的软件使用）。



按照上图的步骤，操作完成后，若下载成功则表示用户代码和相关的设置选项都已下载到 U8 下载工具中。

（3）然后使用连接线连接电脑、将 U8 下载工具以及用户系统（目标单片机）如下图所示的方式连接起来，并按下图所示的按钮后松开，即可开始脱机下载



下载的过程中，U8 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

27.2.7.3 目标芯片由用户系统引线连接 U8 并通过用户系统给 U8 供电进行脱机下载

（1）首先使用 STCAI 提供的 USB 连接线将 U8 下载板连接到电脑，如下图：



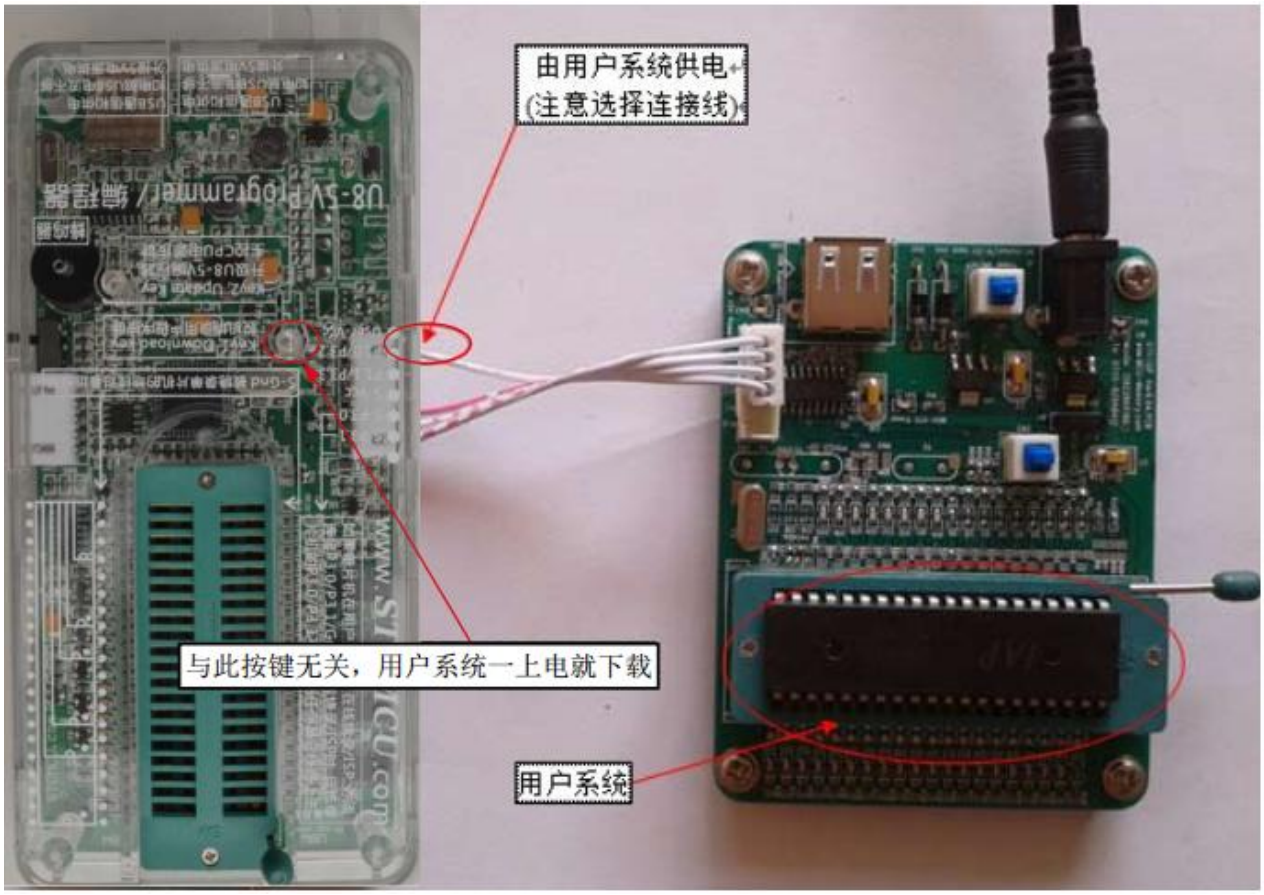
（2）在 ISP 下载软件“AIapp-ISP（V6.95G）.exe”以上版本中按如下图所示的步骤进行设置：

建议用户用最新版本的 AIapp-ISP 下载软件“AIapp-ISP（V6.95G）.exe”（请随时留意 STCAI 官方网站 http://www.STCAI.com 中 AIapp-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCAI.com 中下载最新版本的软件使用）。



按照上图的步骤，操作完成后，若下载成功则表示用户代码和相关的设置选项都已下载到 U8 下载工具中

（3）然后按下图所示的方式连接 U8 与用户系统，并按下图中所示按钮后松开，即可开始脱机下载：



下载的过程中，U8 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

27.2.7.4 目标芯片由用户系统引线连接 U8 且 U8 与用户系统各自独立供电进行脱机下载

（1）首先使用 STC 提供的 USB 连接线将 U8 下载板连接到电脑，如下图:



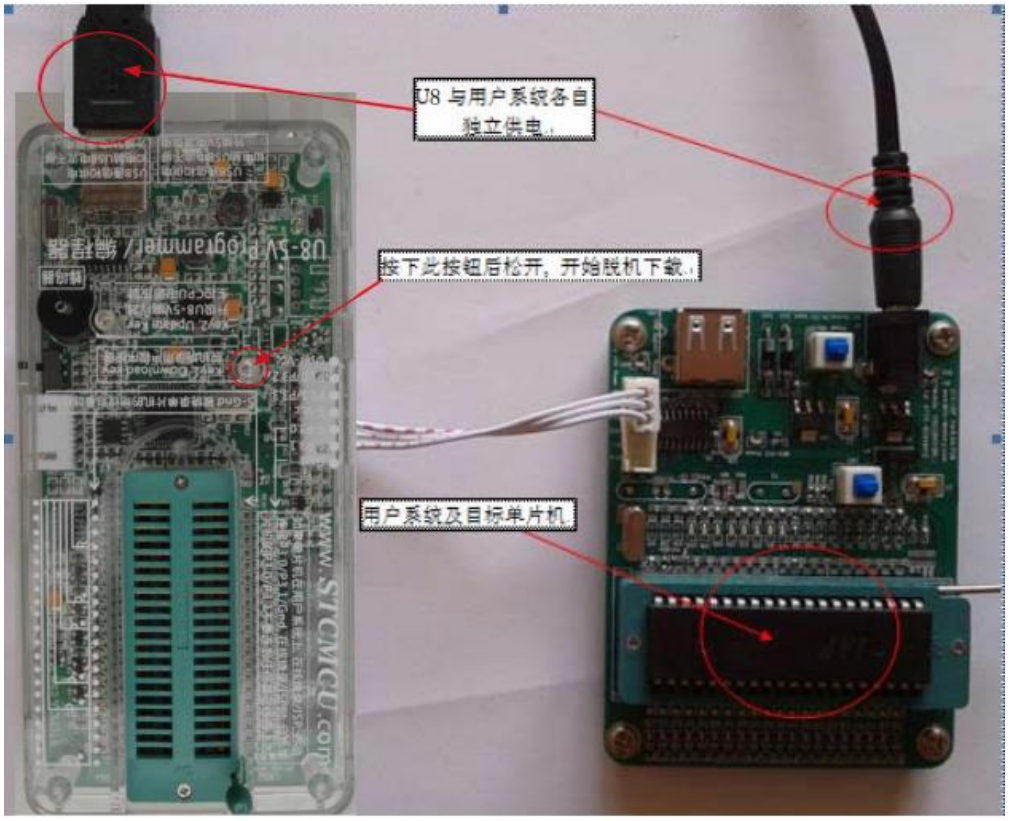
（2）在 ISP 下载软件“AIapp-ISP（V6.95G）.exe”以上版本中按如下图所示的步骤进行设置：

建议用户用最新版本的 AIapp-ISP 下载软件“AIapp-ISP（V6.95G）.exe”（请随时留意 STCAI 官方网站 http://www.STCAI.com 中 AIapp-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCAI.com 中下载最新版本的软件使用）。



按照上图的步骤，操作完成后，若下载成功则表示用户代码和相关的设置选项都已下载到 U8 下载工具中

（3）然后按下图所示的方式连接 U8 与用户系统，并将图中所示按钮先按下后松开，准备开始脱机下载，最后给用户系统上电/开电源，下载用户程序正式开始：



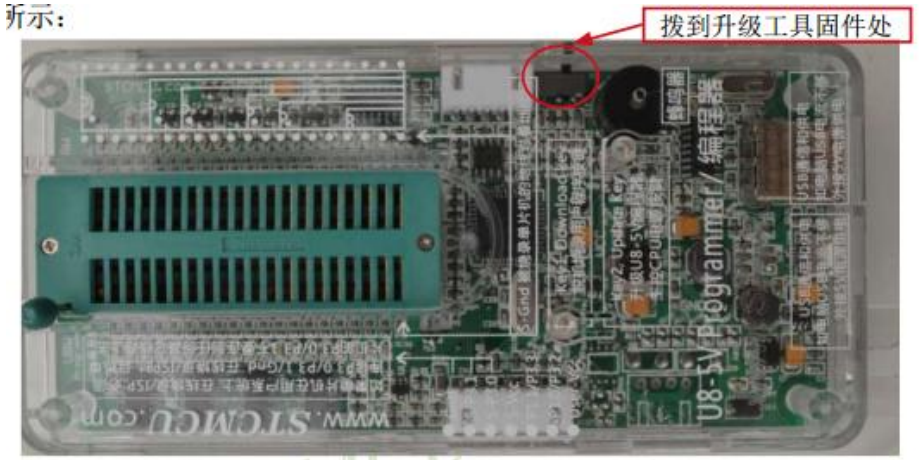
下载的过程中，U8 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

27.2.8 制作/更新 USB 型联机/脱机下载工具 U8W/U8W-Mini/U8/U8-Mini

27.2.8.1 制作 U8W/U8W-Mini/U8/U8-Mini 下载母片（控制母片）

制作 U8W/U8W-Mini 及 U8/U8-Mini 下载母片的过程类似，为节约篇幅，下文以 U8 为例，详述如何制作 U8 下载母片，U8W/U8W-Mini 及 U8-Mini 不作赘述。

在制作 U8 下载母片之前需要将 U8 下载板的“更新/下载选择接口”拨到“升级工具固件”，如下图所示:

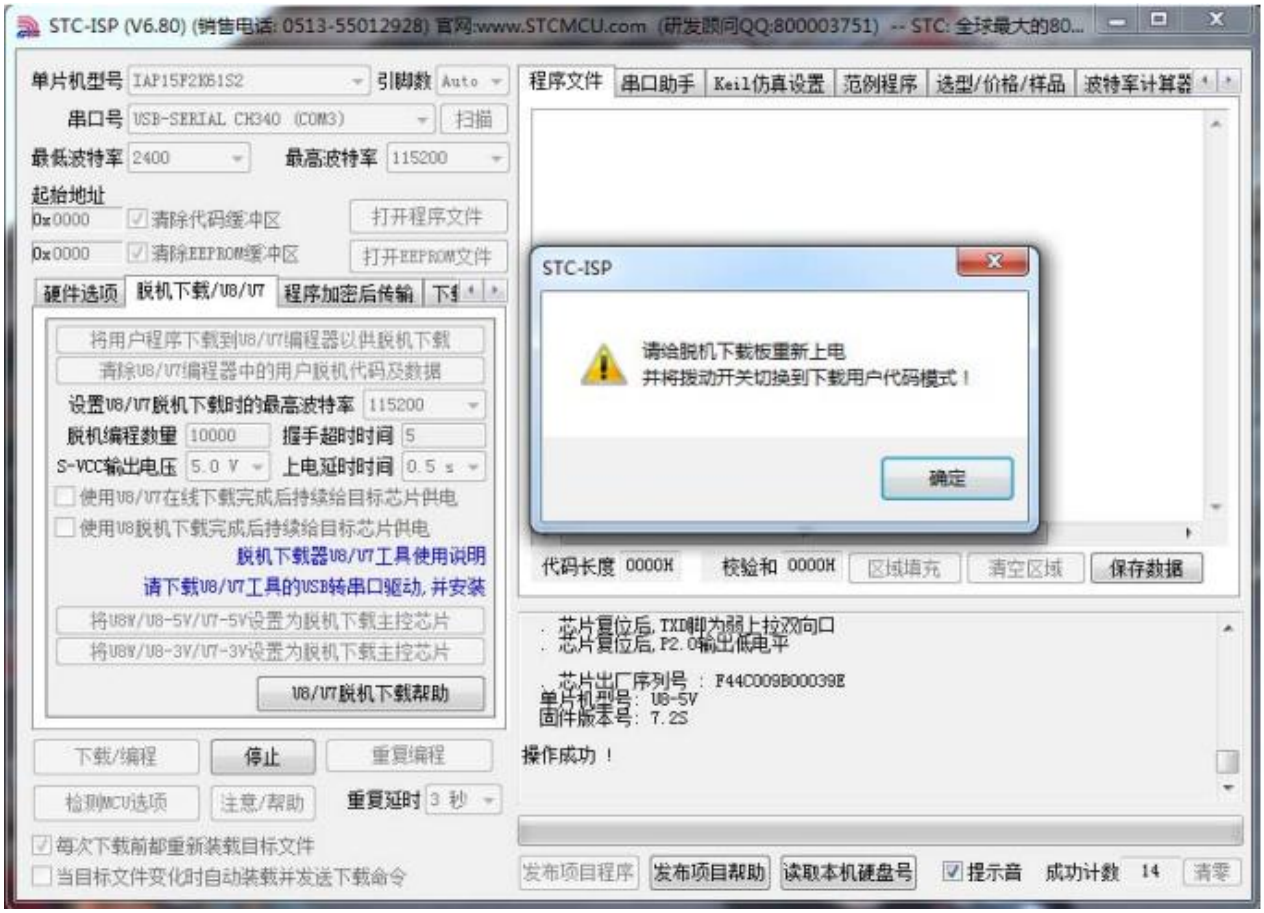


然后在 ISP 下载程序“AIapp-ISP（V6.95G）.exe”中的“脱机下载/U8/U7”页面中点击“将 U8W/U8-5V/U7-5V 设置为脱机下载主控芯片”按钮（5V 下载板）或者点击“将 U8W/U8-3V/U7-3V 设置为脱机下载主控芯片”按钮（3.3V 下载板），如下图：

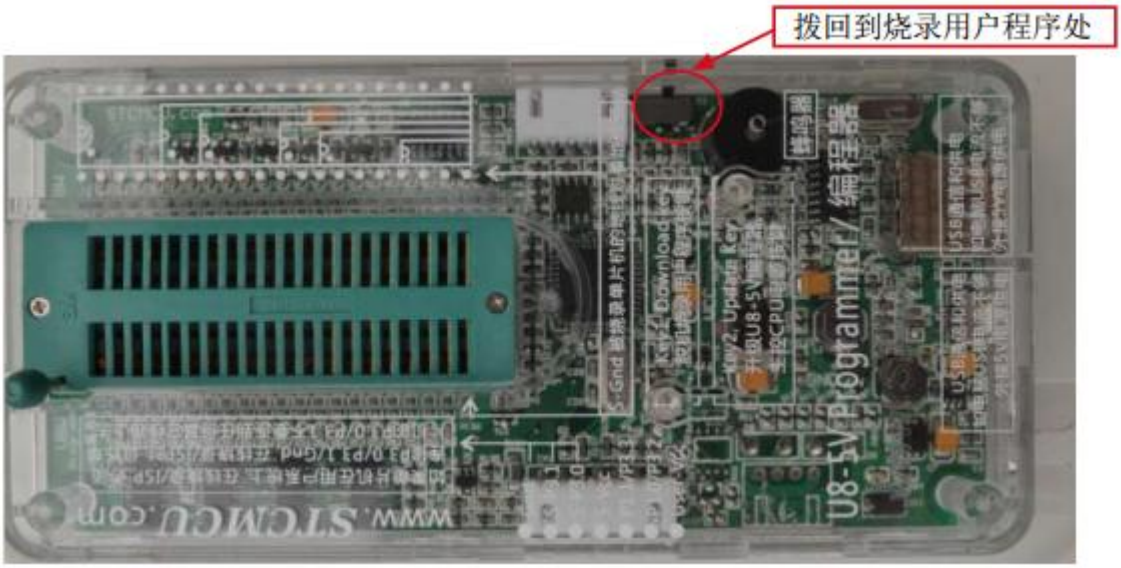
（注意：一定要选择 U8 所对应的串口）



在出现如下画面表示 U8 控制芯片制作完成：



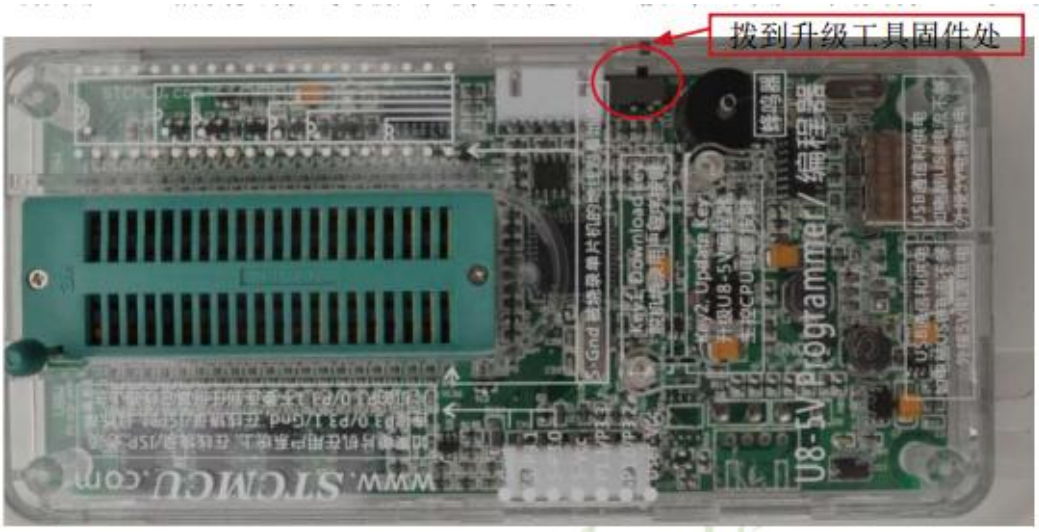
制作完成后，一定不要忘记将 U8 的“更新/下载选择接口”拨回到“烧录用户程序”模式，并将 U8 下载工具重新上电，如下图所示：（否则将不能正常进行下载）



27.2.8.2 手动升级 U8W/U8W-Mini/U8/U8-Mini

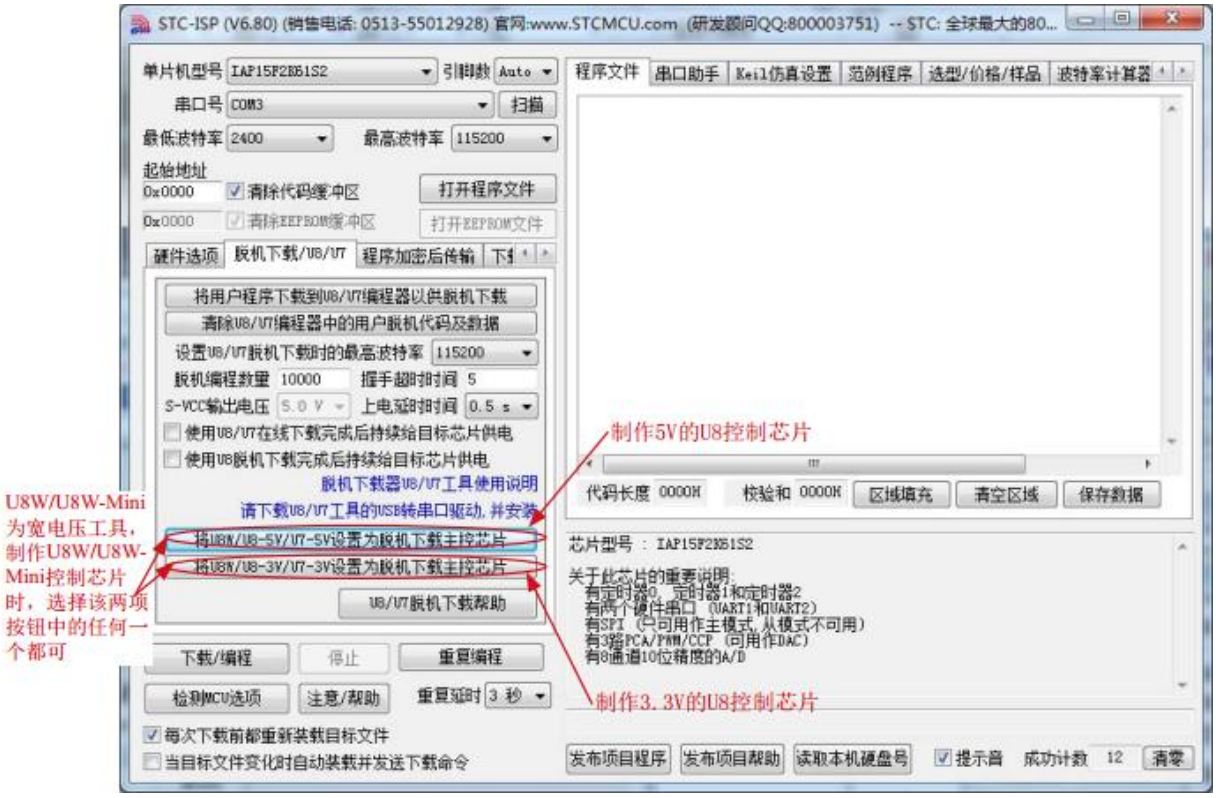
手动升级 U8W/U8W-Mini 及 U8/U8-Mini 的过程类似，为节约篇幅，下文以 U8 为例，详述如何手动升级 U8，U8W/U8W-Mini 及 U8-Mini 不作赘述。

在手动升级 U8 之前需要将“更新/下载选择接口”拨到“升级工具固件”，如下图所示:

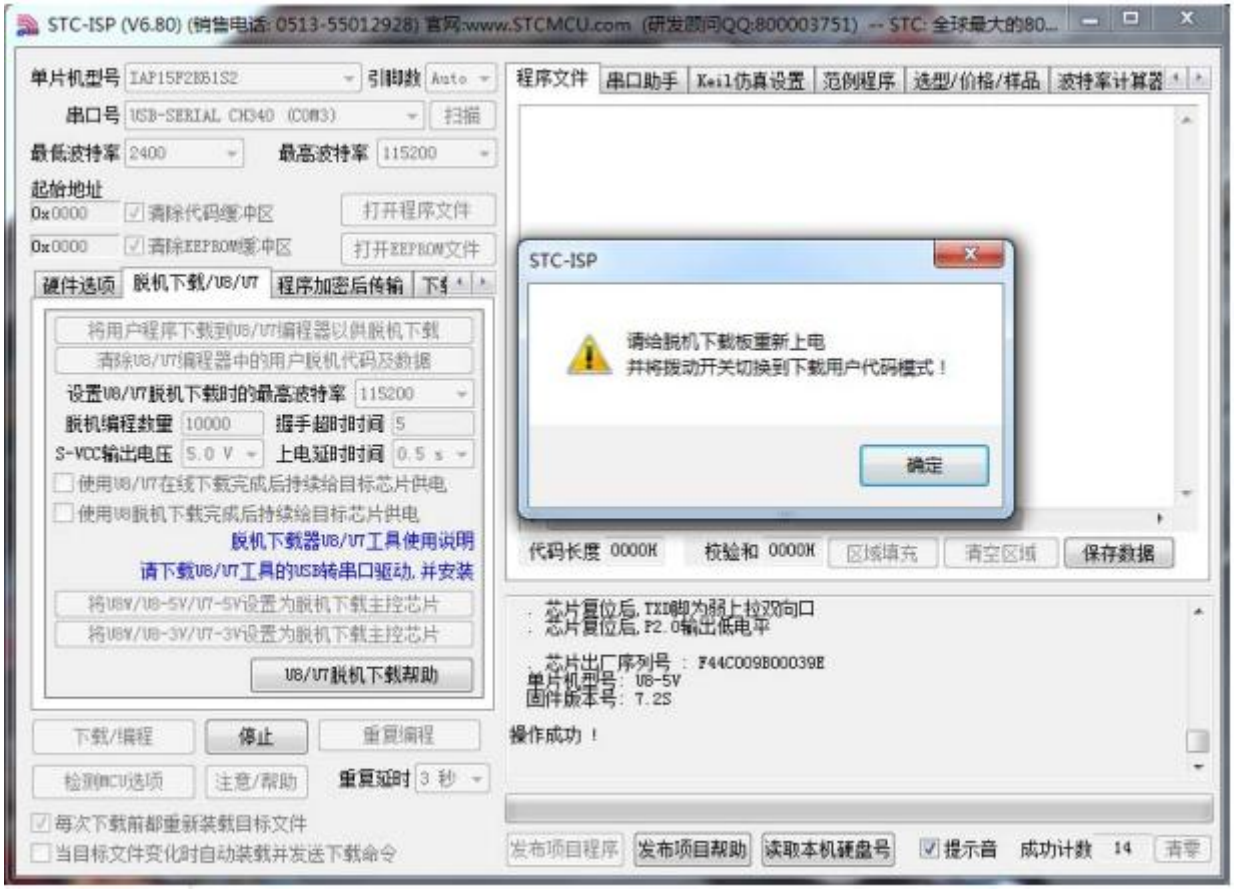


然后在 ISP 下载程序“AIapp-ISP（V6.95G）.exe”中的“脱机下载/U8/U7”页面中点击“将 U8W/U8-5V/U7-5V 设置为脱机下载主控芯片”按钮（5V 下载板）或者点击“将 U8W/U8-3V/U7-3V 设置为脱机下载主控芯片”按钮（3.3V 下载板），如下图:

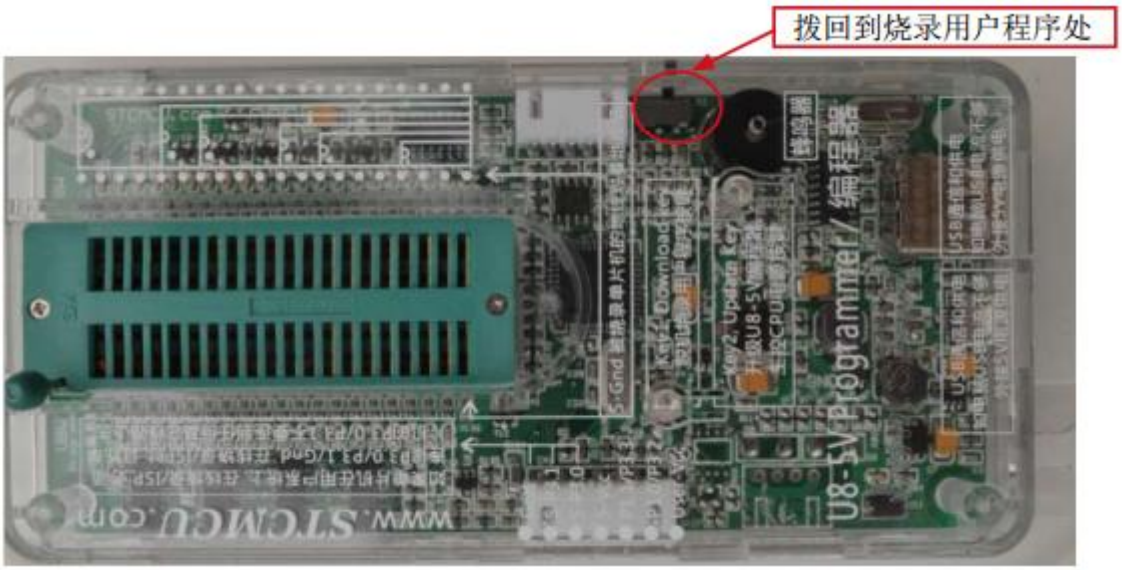
（注意:一定要选择 U8 所对应的串口）



此时由于主控芯片已经被设置为 U8 的下载母片，则点击上面对应的按钮后，芯片会自动进行更新，中间不需要按其它的按键（特殊情况：若软件一直没有反应，则需要用户手动按一下“更新/update”按钮，芯片才能进行更新），直至出现如下画面表示 U8 控制芯片升级完成：



升级完成后，一定不要忘记将 U8 的“更新/下载选择接口”拨回到“烧录用户程序”模式，并将 U8 下载工具重新上电，如下图所示：（否则将不能正常进行下载）

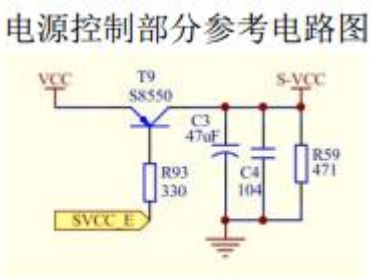


27.2.9 USB 型联机/脱机下载板 U8W/U8W-Mini/U8/U8-Mini 的参考电路

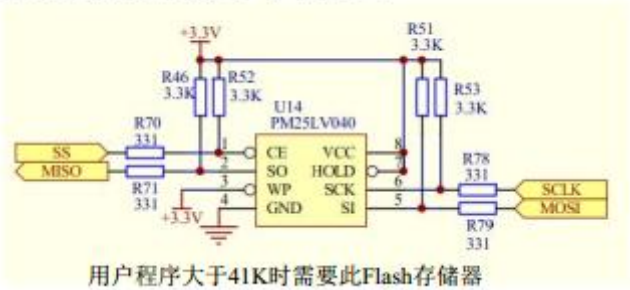
USB 型联机/脱机下载板 U8W/U8W-Mini/U8/U8-Mini 为用户提供了如下的常用控制接口（Ver6.85 版）：

脚位功能	端口	功能描述
电源控制脚	P2.6	低位有效
下载通讯脚	P1.0	串口 RXD，连接目标芯片的 TXD（P3.1）
	P1.1	串口 TXD，连接目标芯片的 RXD（P3.0）
编程按键	P3.6	低有效
显示	P3.2	LED1
	P3.3	LED2
	P3.4	LED3
	P5.5	LED4
外挂串行 Flash 控制脚	P2.4	Flash 的 CE 脚
	P2.2	Flash 的 SO 脚
	P2.3	Flash 的 SI 脚
	P2.1	Flash 的 SCLK 脚
全自动烧录工具分选机信号	P3.6	起始信号
	P1.5	完成信号
	P5.4	OK 信号（良品信号）
	P3.7	ERROR 信号（不良品信号）
蜂鸣器（BEEP）控制	P2.5	高有效（高电平发出声音）

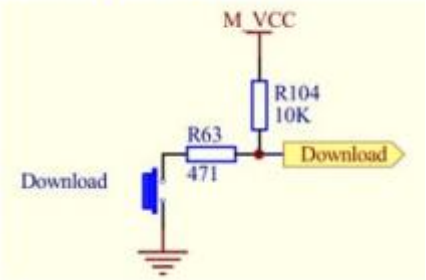
电源控制部分参考电路图



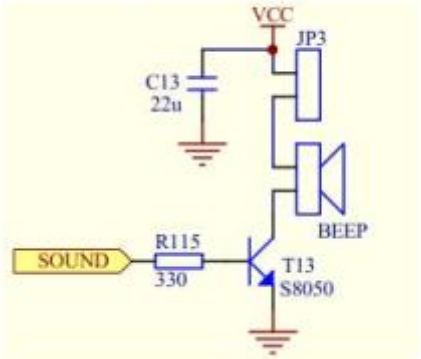
Flash控制部分参考电路图



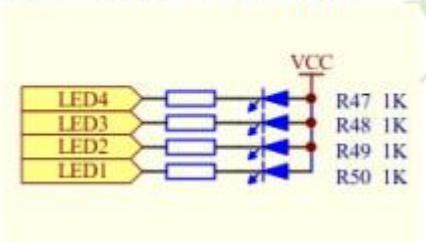
按键部分参考电路图



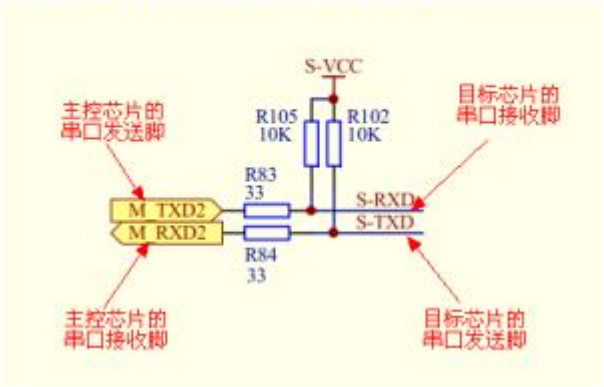
蜂鸣器部分参考电路图



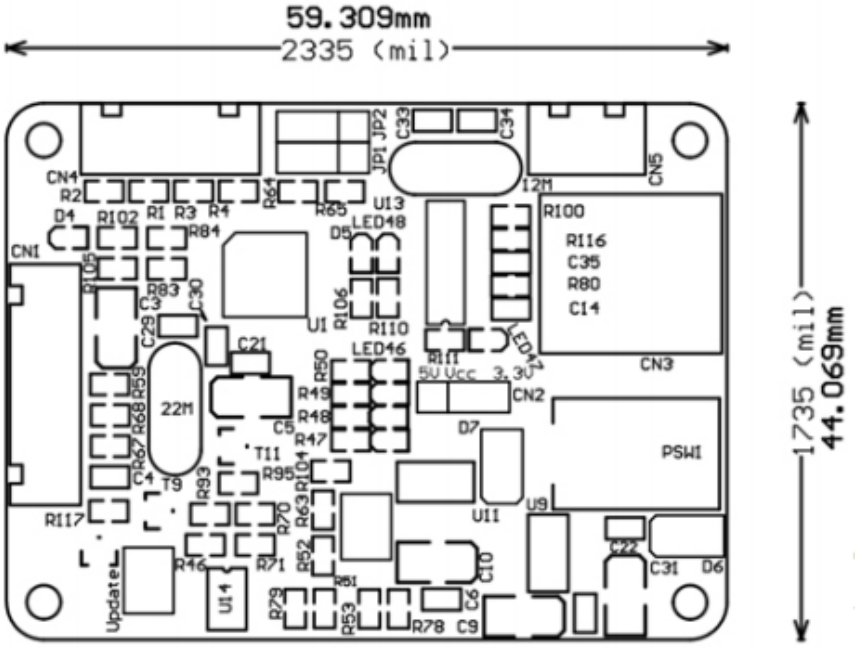
LED显示部分参考电路图



串口通讯脚连接部分参考电路图

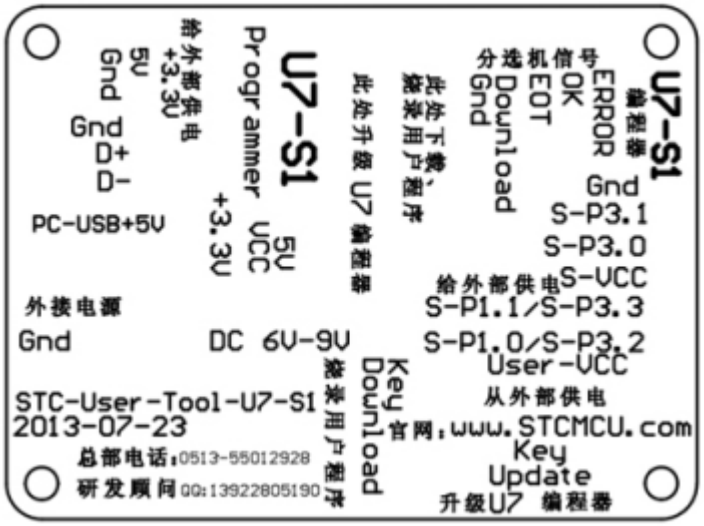


U8 PCB 板正面丝印图：

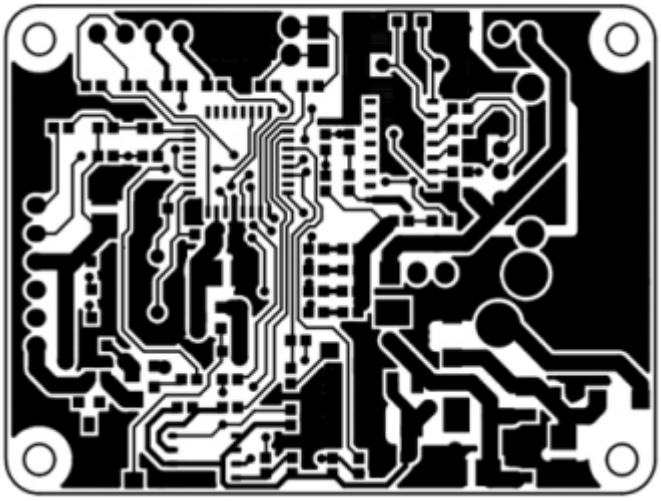


四角的安装孔直径：2．8mm

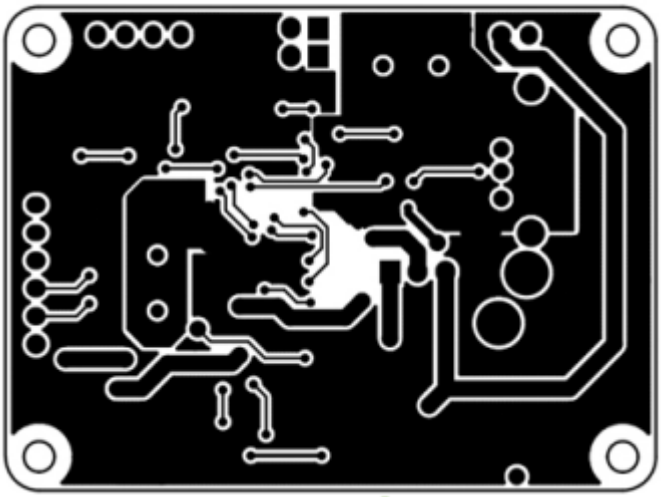
U8 PCB 板反面丝印图：



U8 PCB板走线图（正面）：



U8 PCB板走线图（反面）：



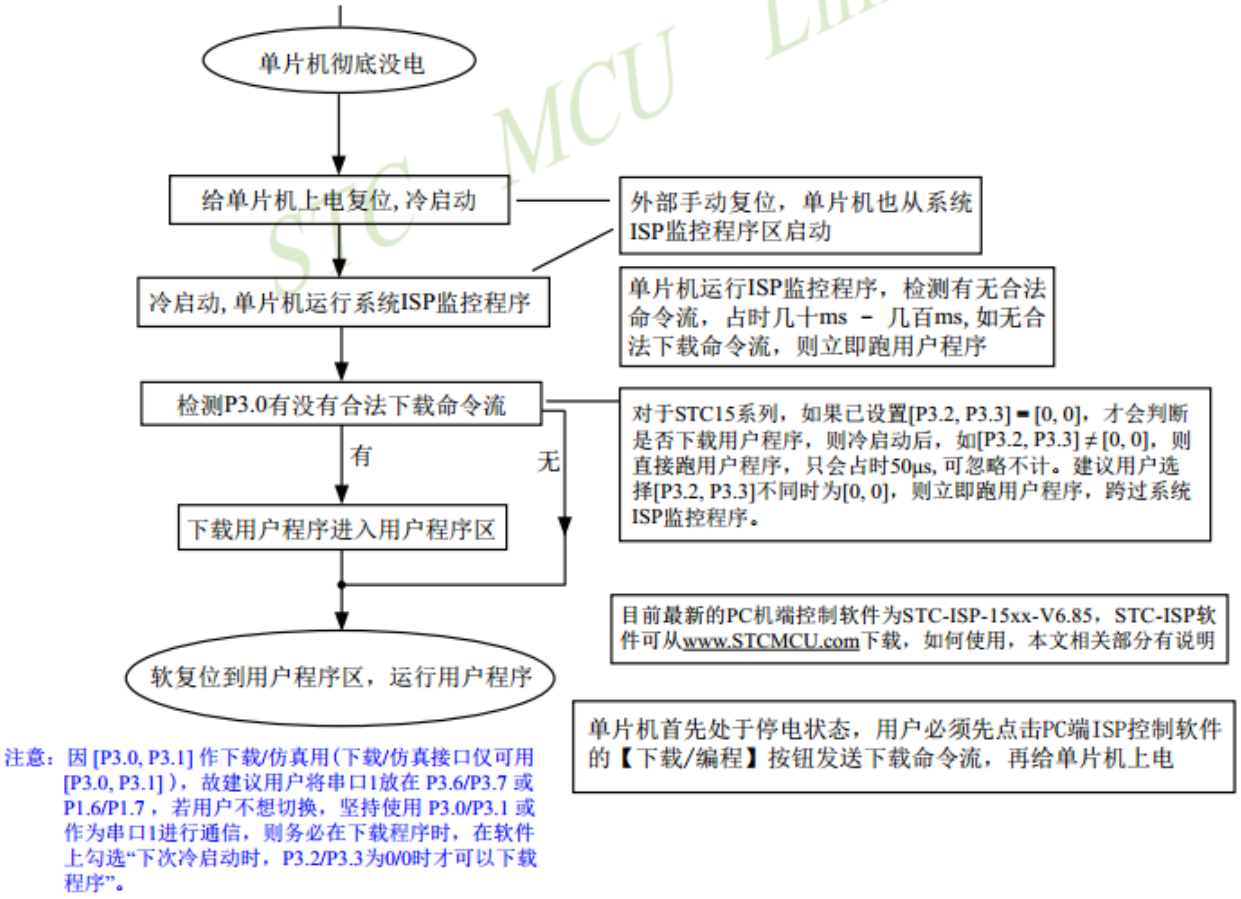
27.3 ISP 编程器/烧录器的说明

我们有：AIapp-ISP 经济型下载编程工具

所有 AIapp-ISP 编程工具的分类如下

- AIapp-ISP 经济型下载编程工具
  - STC12/11/10/89/90 系列专用 ISP 下载编程工具
    - STC12/11/10/89/90 系列 40-pin 单片机专用 ISP 下载编程工具
    - STC12/11/10/89/90 系列 32-pin 单片机专用 ISP 下载编程工具
    - STC12/11/10/89/90 系列 28-pin 单片机专用 ISP 下载编程工具
    - STC12/11/10/89/90 系列 20-pin 单片机专用 ISP 下载编程工具
    - STC12/11/10/89/90 系列 18-pin 单片机专用 ISP 下载编程工具
    - STC12/11/10/89/90 系列 16-pin 单片机专用 ISP 下载编程工具
  - STC15 系列专用 ISP 下载编程工具

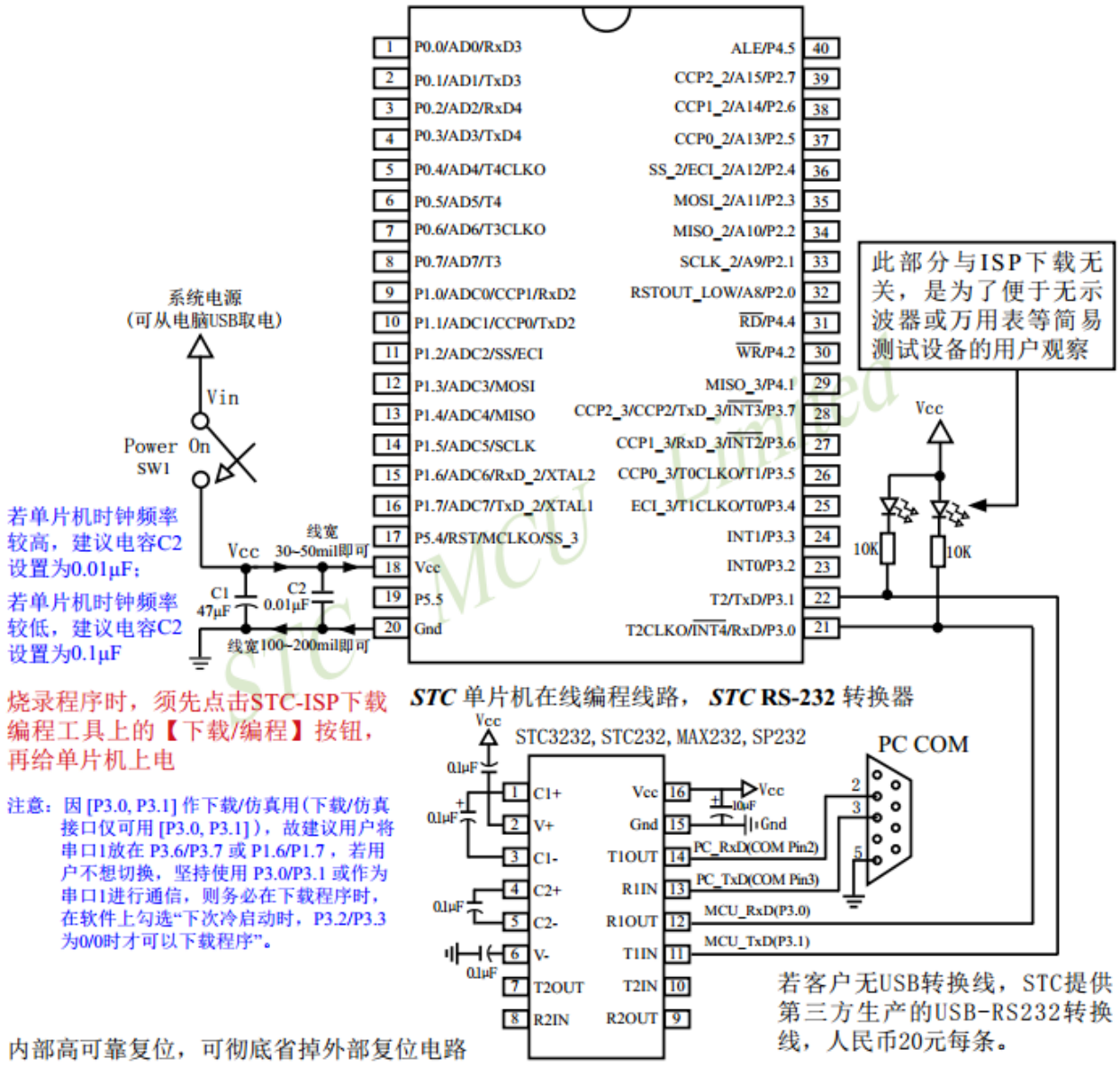
27.3.1 在系统可编程(ISP)原理使用说明



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 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”。

27.3.2 STC15 系列在系统可编程(ISP)典型应用线路图

27.3.2.1 利用 RS-232 转换器的 ISP 下载典型应用线路图



P5.4/RST/MCLKO 脚出厂时默认为 I/O 口，可以通过 AIapp-ISP 编程器将其设置为 RST 复位脚.

内部集成高精度 R/C 时钟（±0.3%），±1%温漂（-40℃～+85℃），常温下温漂±0.6%（-20℃～+65℃），5MHz～35MHz 宽范围可设置，可彻底省掉外部昂贵的晶振

建议在 Vcc 和 Gnd 之间就近加上电源去耦电容 C1（47μF），C2（0.01μF），可去除电源线噪声，提高抗干扰能力。

如何产生虚拟串口：①安装 Windows 驱动程序；②插上 USB-RS232 转换线（若客户无 USB 转换线，STC 提供第三方生产的 USB-RS232 转换线，人民币 20 元每条.）；③确定 PC 端口 COM：右击我的电脑→属性→硬件→设备管理器→确定所扩展的串口是 PC 电脑虚拟的第几个 COM。

STC 系列单片机具有在系统可编程（ISP）特性，ISP 的好处是：省去购买通用编程器。

单片机在用户系统上即可下载/烧录用户程序，而无须将单片机从已生产好的产品上拆下，再用通用编程器将程序代码烧录进单片机内部。有些程序尚未定型的产品可以一边生产，一边完善，加快了产品进入市场的速度，减小了新产品由于软件缺陷带来的风险。由于可以在用户的目标系统上将程序直接下载进单片机看运行结果对错，故无须仿真器。

STC 系列单片机内部固化有 ISP 系统引导固件，配合 PC 端的控制程序即可将用户的程序代码下载进单片机内部，故无须编程器（速度比通用编程器快，几秒一片）。

如何获得及使用 STC 提供的 ISP 下载工具（AIapp-ISP.exe 软件）：

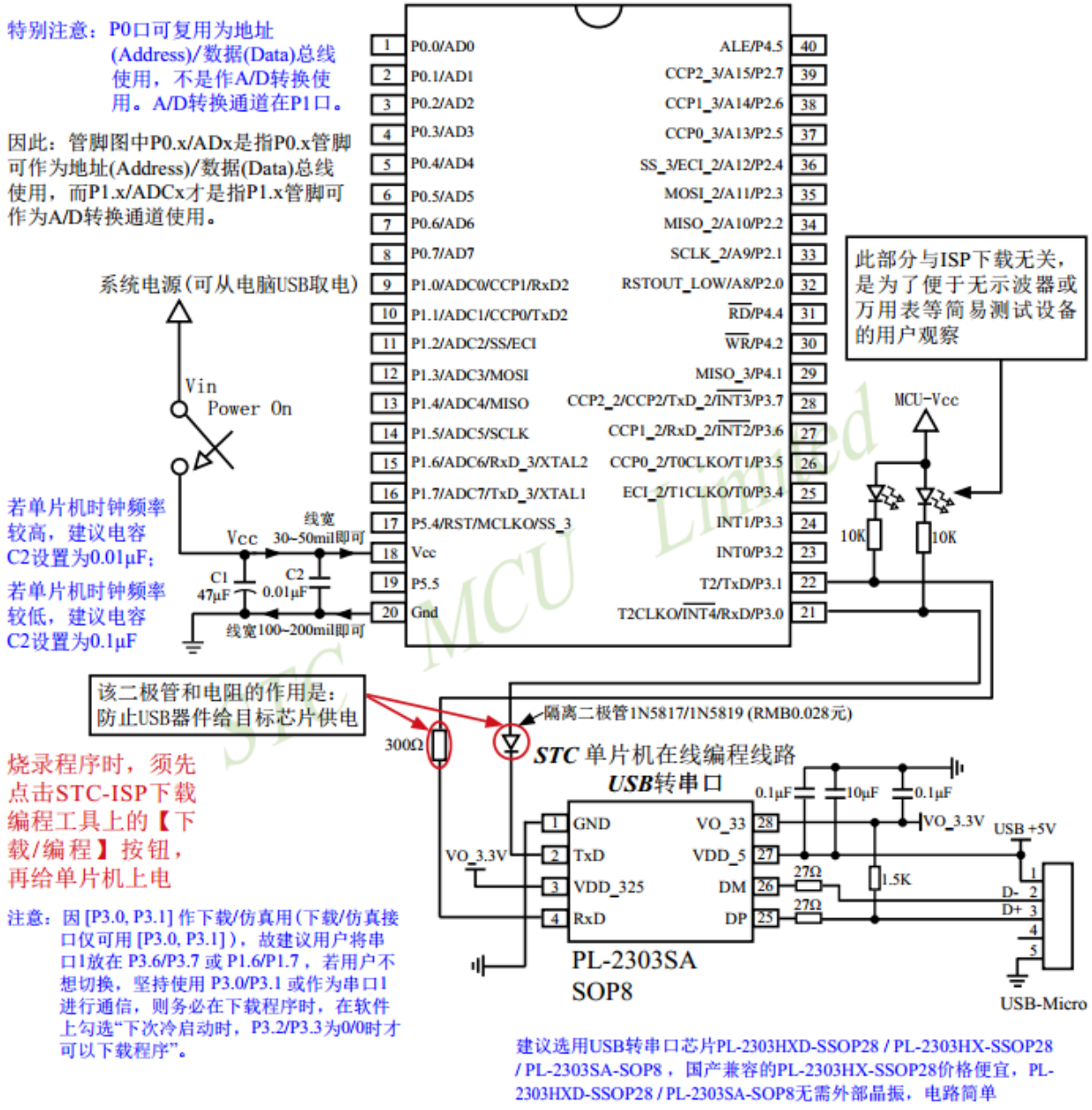
（1）.获得 STC 提供的 ISP 下载工具（软件）
登陆 www.STCAI.com 网站，下载 PC（电脑）端的 ISP 下载工具（软件），然后将其自解压，再安装即可（执行 setup.exe），注意随时更新软件。

（2）.使用 AIapp-ISP 下载工具（软件），请随时更新，AIapp-ISP 下载工具目前已到 Ver6.95G 版本。
支持*.bin，*.hex（Intel 16 进制格式）文件，少数*.hex 文件不支持的话，请转换成*.bin 文件，请随时注意升级 PC（电脑）端的 AIapp-ISP.exe 软件。

（3）.STC 系列单片机出厂时就已完全加密。需要单片机内部的电放光后上电复位（冷起动）才运行系统 ISP 监控程序，如从 P3.0 检测到合法的下载命令流就下载用户程序，如检测不到就复位到用户程序区，运行用户程序。

（4）.如果用户系统接了 RS-485 通信电路，推荐将 RS-485 电路接到[P1.6, P1.7]或[P3.6, P3.7]上，这样既方便又安全，且不用在 AIapp-ISP 下载编程工具中选择“下次冷启动时需[P3.2, P3.3] = [0, 0]才可以下载程序”。

27.3.2.2 利用 USB 转串口芯片 PL-2303SA 的 ISP 下载编程典型应用线路图



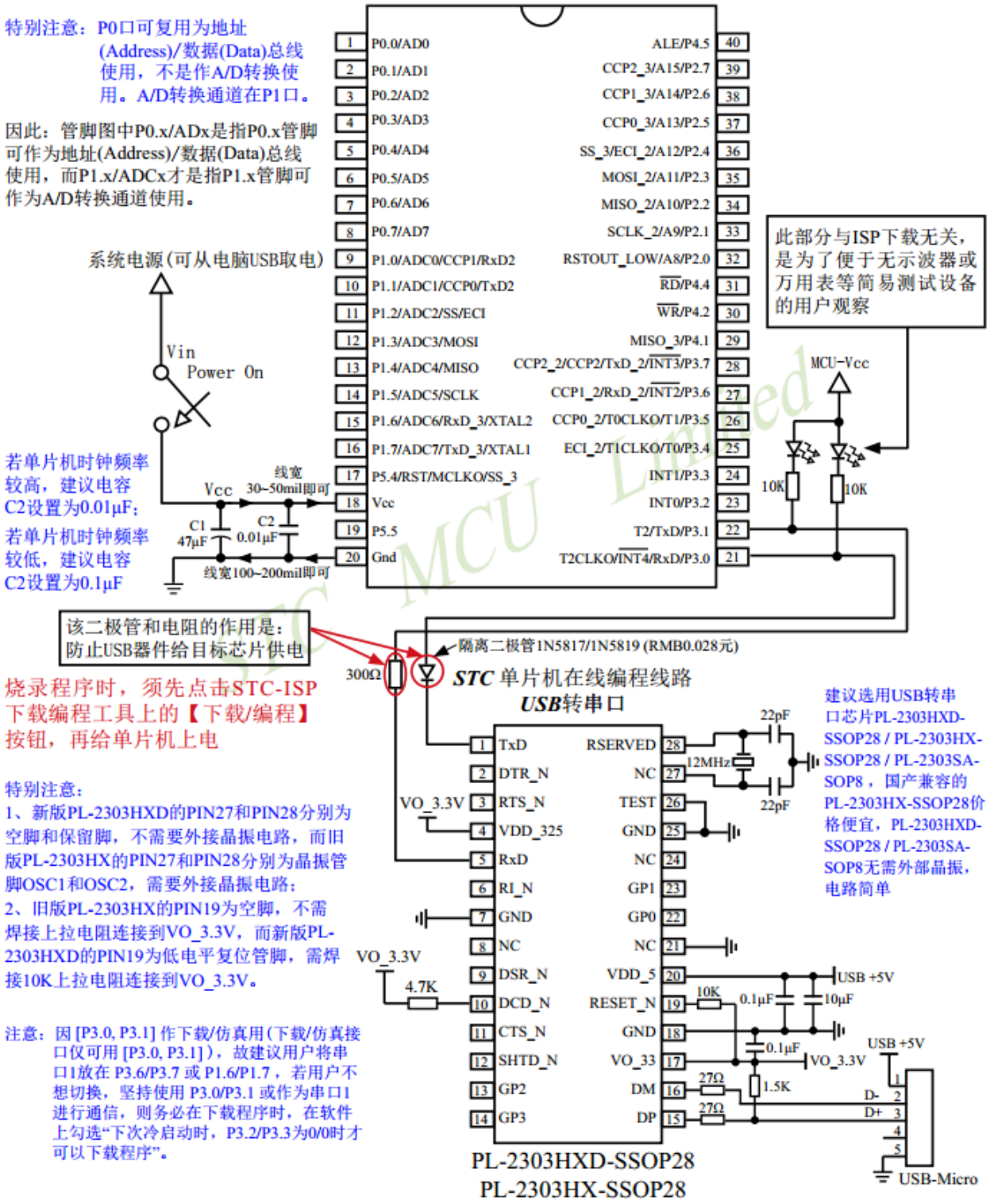
内部高可靠复位，可彻底省掉外部复位电路

P5.4/RST/MCLKO 脚出厂时默认为 I/O 口，可以通过 AIapp-ISP 编程器将其设置为 RST 复位脚（高电平复位）。

内部集成高精度 R/C 时钟（±0.3%），　±1%温漂（-40℃～+85℃），常温下温漂±0.6%（-20℃～+65℃），5MHz~35MHz 宽范围可设置，可彻底省掉外部昂贵的晶振。

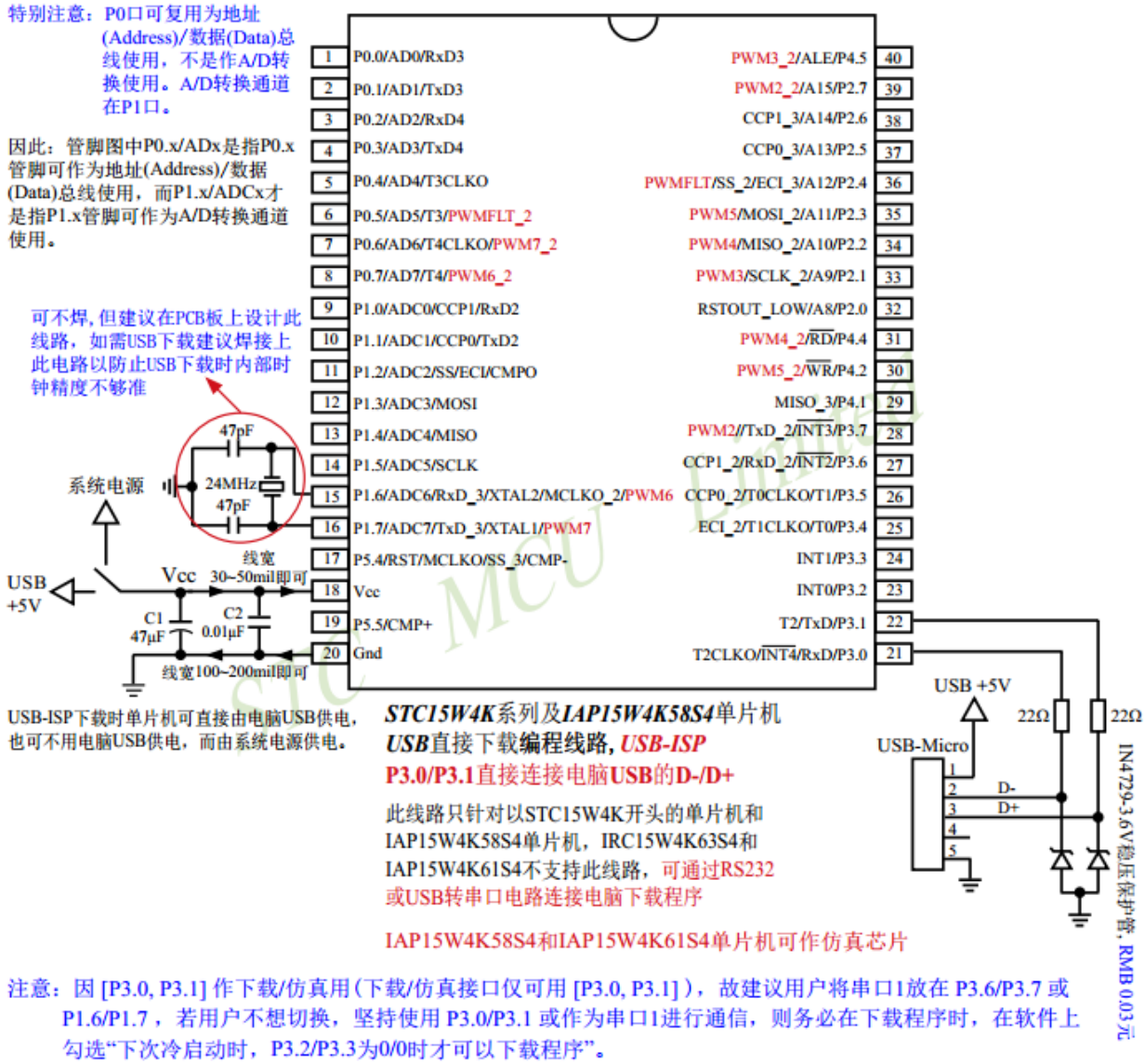
建议在 Vcc 和 Gnd 之间就近加上电源去耦电容 C1（47μF），C2（0.01μF），可去除电源线噪声，提高抗干扰能力。

27.3.2.3 利用 USB 转串口芯片 PL-2303HXD/PL-2303HX 的 ISP 下载编程典型应用线路图



27.3.2.4 STC15W4K 系列及 IAP15W4K58S4 单片机的 USB 直接下载编程线路, USB-ISP

----单片机的 P3.0/P3.1 直接连接电脑 USB 的 D-/D+



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 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”。

内部高可靠复位，可彻底省掉外部复位电路

P5.4/RST/MCLKO 脚出厂时默认为 I/O 口，可以通过 AIapp-ISP 编程器将其设置为 RST 复位脚（高电平复位）。

建议在 Vcc 和 Gnd 之间就近加上电源去耦电容 C1（47μF），C2（0.01μF），可去除电源线噪声，提高抗干扰能力

关于电源：

用户系统的电源可以直接由电脑 USB 供电，也可不用电脑 USB 供电，而由系统电源供电。

若用户单片机系统直接使用电脑 USB 供电，则在用户单片机系统插上电脑 USB 口时，电脑就会检测到 STC15W4K 系列或 IAP15W4K58S4 单片机插入到了电脑 USB 口，如果用户第一次使用该电脑对 STC15W4K 系列或 IAP15W4K58S4 单片机进行 ISP 下载，则该电脑会自动安装 USB 驱动程序，而 STC15W4K 系列或 IAP15W4K58S4 单片机则自动处于等待状态，直到电脑安装完驱动程序并发送【下载/编程】命令给它。

若用户单片机系统使用系统电源供电，则用户单片机系统须在停电（即关闭系统电源）后才能插上电脑 USB 口；在用户单片机系统插上电脑 USB 口并打开系统电源后，电脑会检测到 STC15W4K 系列或 IAP15W4K58S4 单片机插入到了电脑 USB 口，如果用户第一次使用该电脑对 STC15W4K 系列或 IAP15W4K58S4 单片机进行 ISP 下载，则该电脑会自动安装 USB 驱动程序，而 STC15W4K 系列或 IAP15W4K58S4 单片机则自动处于等待状态，直到电脑安装完驱动程序并发送【下载/编程】命令给它。

目前，我司针对 STC15W4K 系列或 IAP15W4K58S4 单片机的 USB 驱动程序只适用于 WinXP 操作系统及 Win7/Win8 的 32 位操作系统，支持 Win7/Win8 的 64 操作系统的 USB 驱动程序尚待进一步开发，建议 Win7/Win8 的 64 操作系统使用 USB 转串口进行 ISP 下载。

关于晶振：

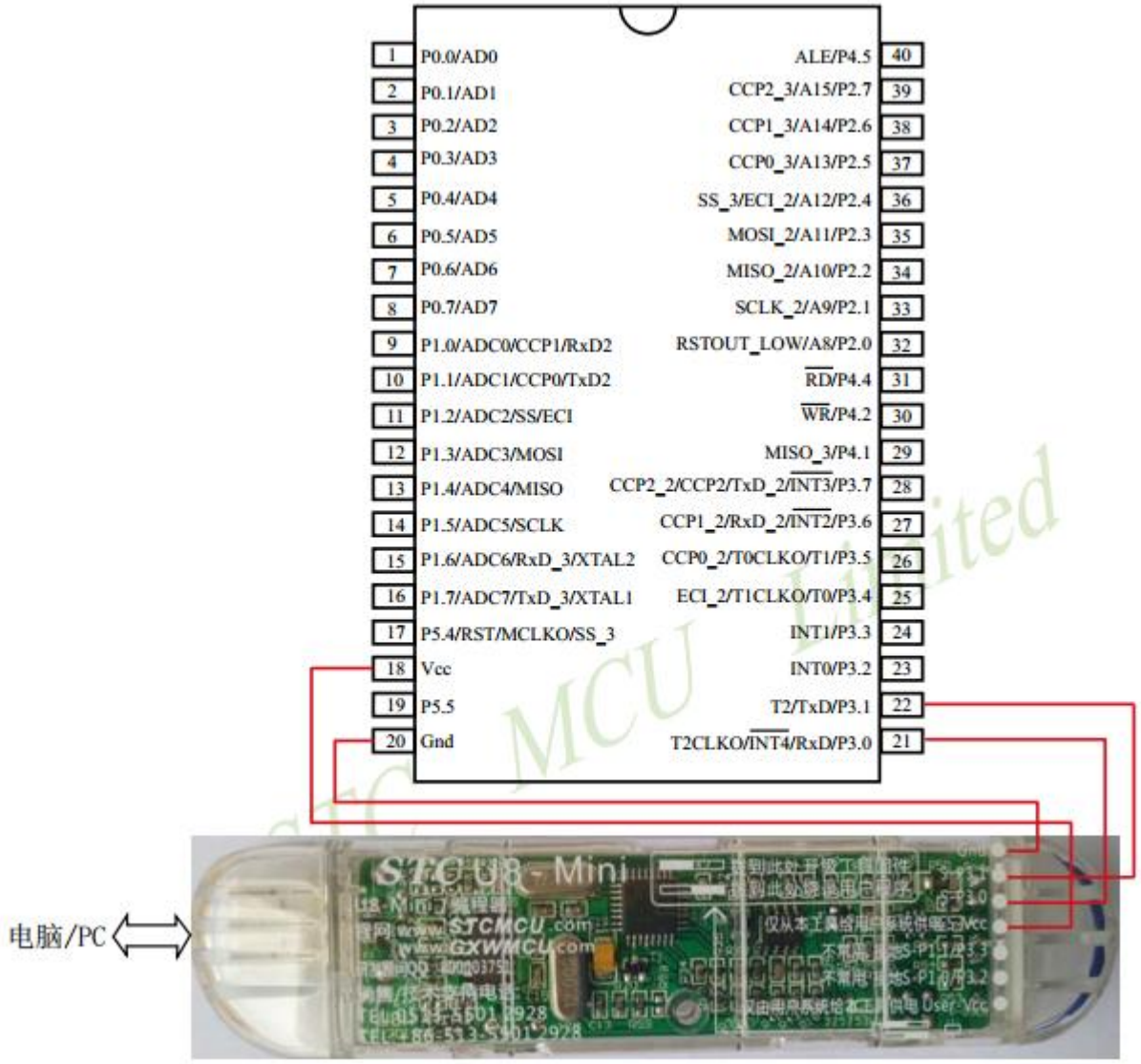
如果用户单片机系统需用外部晶振，则晶振值必须为 24MHz；

如果用户要将用户单片机系统设置成使用内部时钟，则该单片机系统最好不要外接外部晶振；但是如果用户既想将用户单片机系统设置成使用内部时钟，又想外挂外部晶振（24MHz），则该单片机系统上电复位的额外延时<180ms>不能设。



USB-Micro 实物图

27.3.2.5 利用 U8-Mini 进行 ISP 下载的示意图



如用户需要将单片机插在锁紧座上进行 ISP 下载，可用下载工具 U8（U8 具有锁紧座，除此之外其余功能模块均与 U8-Mini 相同），U8 的实物图如下所示：



27.3.2.6 利用 U8 进行 ISP 下载的示意图



ISP 下载时，

（1）首先将单片机直接插在 U8 的锁紧座上；

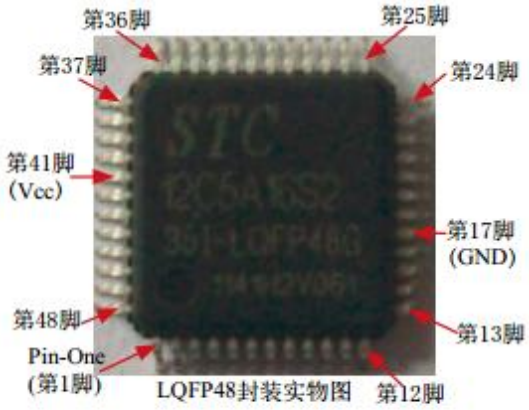
（2）然后通过两头公的 USB 下载线或 Micro USB 下载线将 U8 下载工具连接到电脑 USB 口；

（3）再打开电脑端的 ISP 下载软件，设置好相应单片机型号的参数；

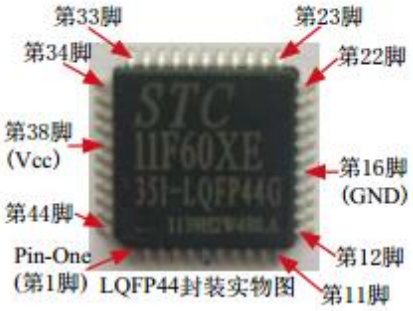
（4）最后，点击 ISP 软件的“打开程序文件”按钮打开待下载的程序文件并点击“下载/编程”按钮后给单片机上电，即可利用 U8 对单片机进行 ISP 下载。

27.3.3 所有 STC 系列单片机封装实物图

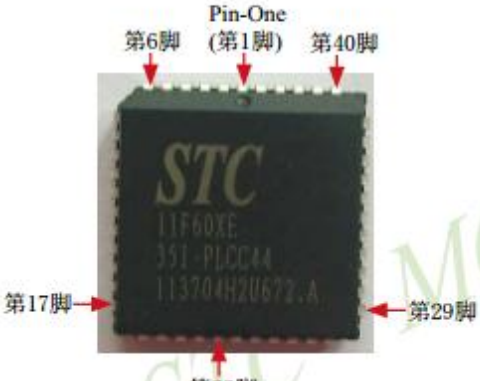
STC12/11/10/89/90 系列单片机的封装实物图：



LQFP48封装实物图



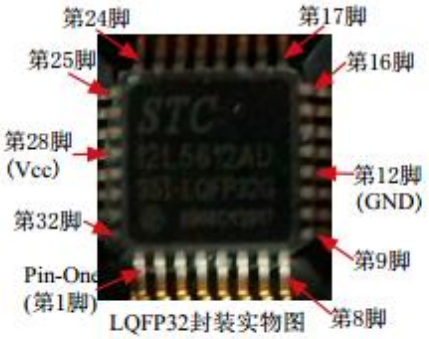
LQFP44封装实物图



PLCC44封装实物图



40-Pin单片机PDIP40的封装实物图



LQFP32封装实物图



SOP32封装实物图



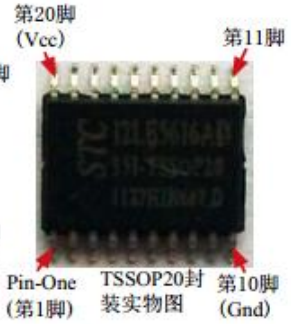
SOP28封装实物图



SKDIP28单片机
的封装实物图

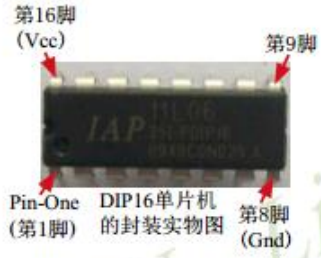


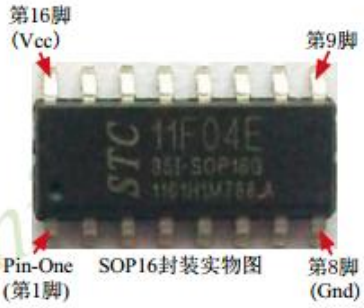




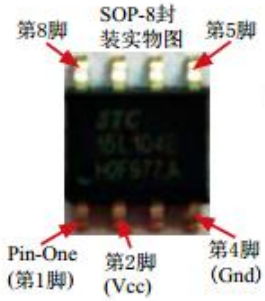


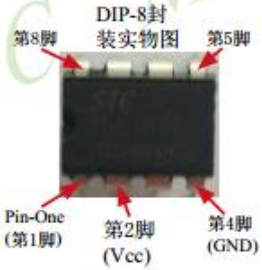






STC15系列单片机的封装实物图：





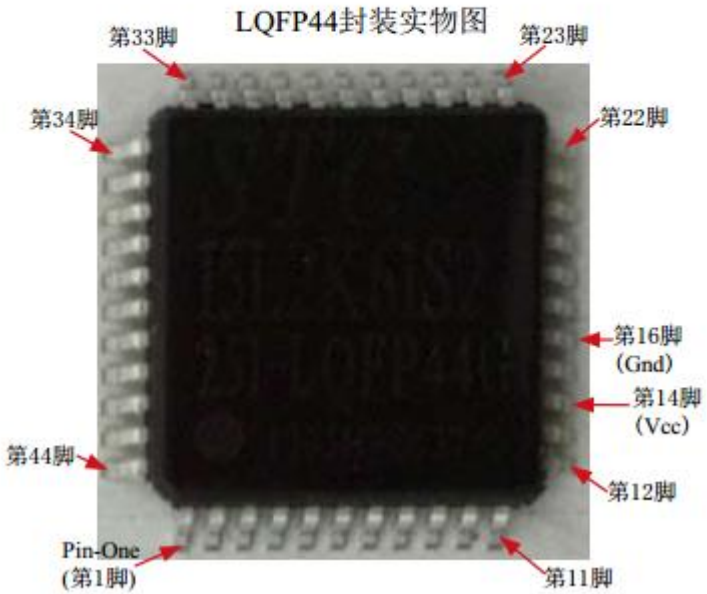







第28脚
STC15系列SOP28封装实物图
第15脚
Pin-One
(第1脚)
第12脚
(Vcc)
第14脚
(GND)


第40脚
STC15系列PDIP40封装实物图
第21脚
Pin-One
(第1脚)
第18脚
(Vcc)
第20脚
(GND)


LQFP44封装实物图
第33脚
第23脚
第34脚
第22脚
第16脚
(Gnd)
第14脚
(Vcc)
第44脚
第12脚
Pin-One
(第1脚)
第11脚

27.3.4 AIapp-ISP 下载编程工具硬件——AIapp-ISP 下载板

27.3.4.1 STC15 系列 ISP 下载板实物图

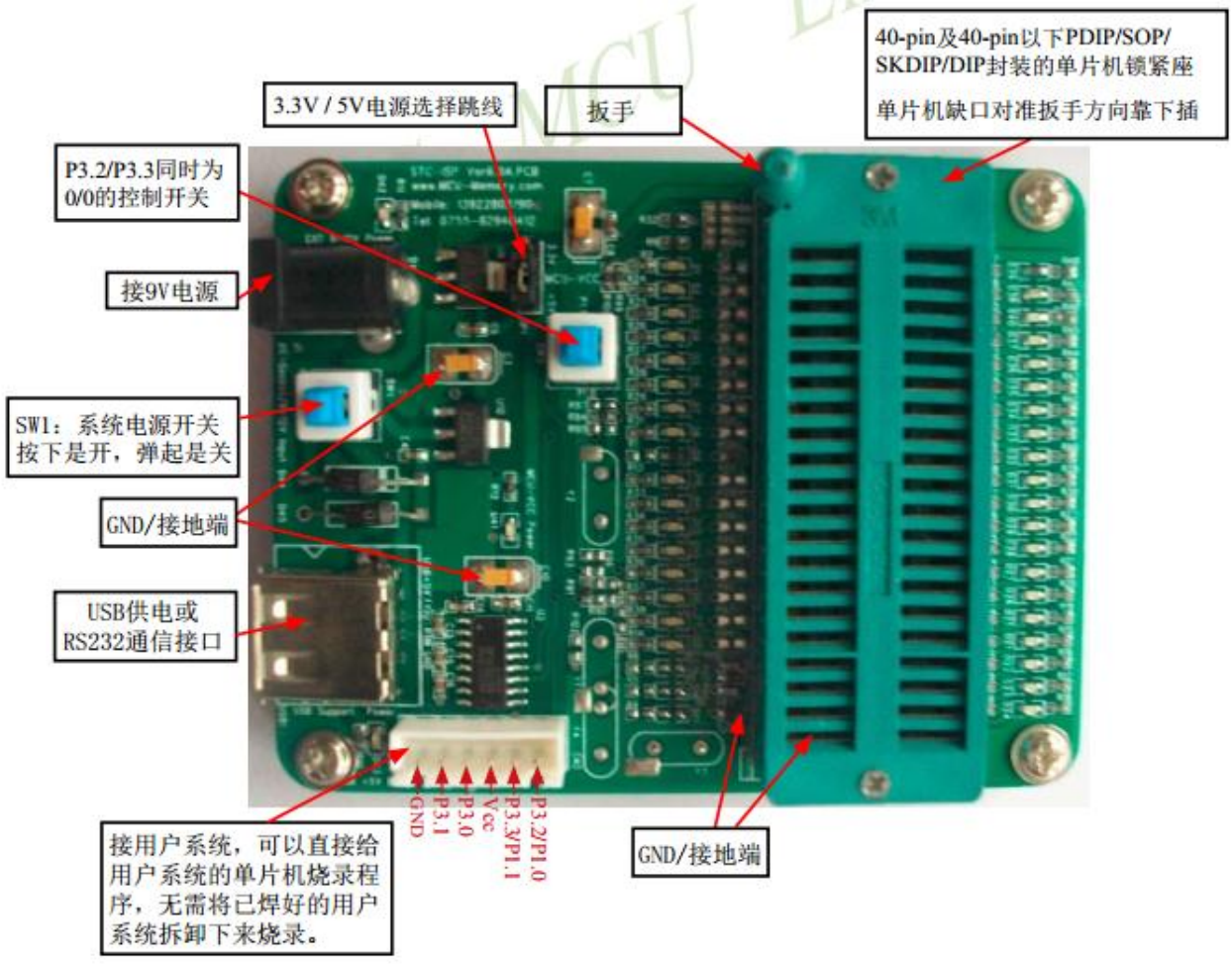
STC15 系列单片机专用 ISP 下载编程工具实物图





STC15系列ISP下载编程工具与STC12/11/10/89/90系列的ISP下载编程工具不兼容，因此注意此ISP下载编程工具适用的单片机型号

STC15 系列专用 ISP 下载编程工具为例详细介绍 AIapp-ISP 下载板的布局：



27.3.4.2 如何将单片机安装到 AIapp-ISP 下载板上

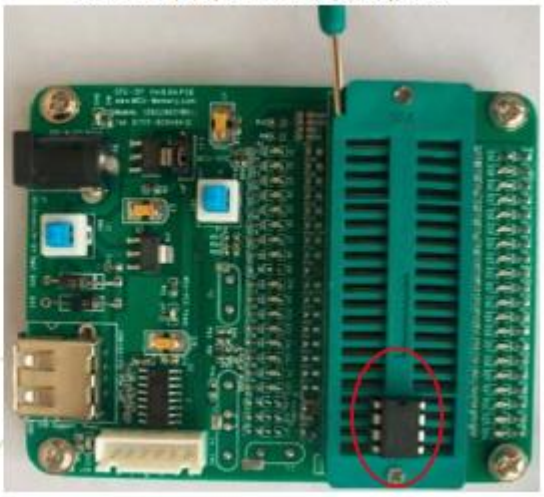
根据用户所使用的单片机型号及管脚选择相应的 AIapp-ISP 下载板，先将下载板上的扳手向上弹起，然后将单片机插入相应的 AIapp-ISP 下载板的锁紧座上（具体做法是：将芯片的半圆缺口对准扳手的方向靠下插），最后将扳手向下按锁紧单片机。

注意：不管是哪种 AIapp-ISP 下载编程工具，其正面焊的编程烧录用锁紧座都是 40Pin 的，锁紧座第 20-Pin 接的是地线（GND），所以请将单片机的地线对着锁紧座的地线插即将芯片的半圆缺口对准扳手的方向靠下插。

STC15系列SKDIP28封装的插法



STC15系列DIP-8封装的插法



27.3.4.3 如何使用转换座将贴片封装的单片机安装到 AIapp-ISP 下载板上

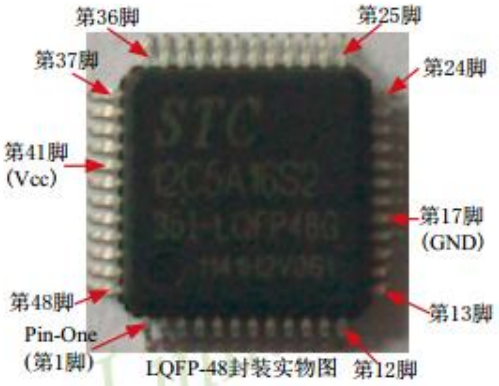
AIapp-ISP 下载板的编程烧录锁紧座只能插入 40 Pin 及 40 Pin 以下的直插式的单片机，对于 LQFP、PLCC、SOP 等封装的单片机需转换座将这些封装转换成直插式的封装才能插入 AIapp-ISP 下载板中。下面介绍几种常用的转换座以及如何使用这些转换座。



LQFP-48的转换座(正面)



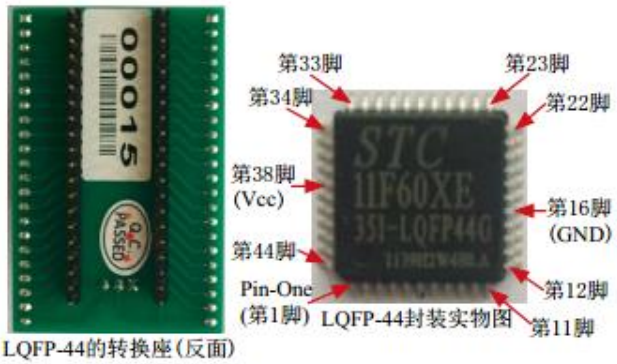
LQFP-48的转换座(内部)



LQFP-48封装实物图


LQFP-44的转换座(正面)


LQFP-44的转换座(内部)



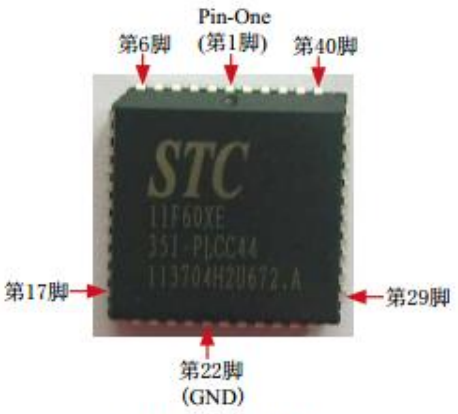
LQFP-44的转换座(反面)
LQFP-44封装实物图



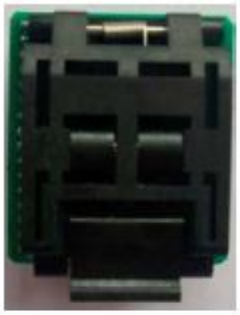
PLCC-44的转换座(正面)



PLCC-44的转换座(反面)



PLCC-44封装实物图



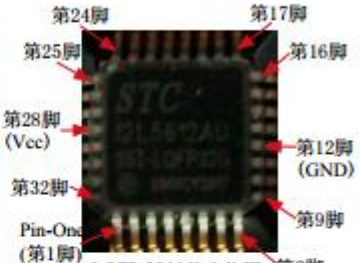
LQFP-32的转换座(正面)



LQFP-32的转换座(内部)



LQFP-32的转换座(反面)



LQFP-32封装实物图



SOP-32的转换座(正面)



SOP-32的转换座(反面)



SOP-32封装实物图



SOP-28和SOP-20
的转换座(正面)



SOP-28和SOP-20
的转换座(反面)



SOP-28封装实物图



SOP-20封
装实物图



SOP-16和SOP-8
的转换座(正面)



SOP-16和SOP-8
的转换座(反面)



SOP-16封装实物图



SOP-8封装
实物图

给需转换座的单片机烧录程序的具体步骤如下：

（1）根据单片机的封装选择转换座，并将单片机安装进转换座中：

LQFP-48/LQFP-44/LQFP-32 封装的单片机按下图所示安装；



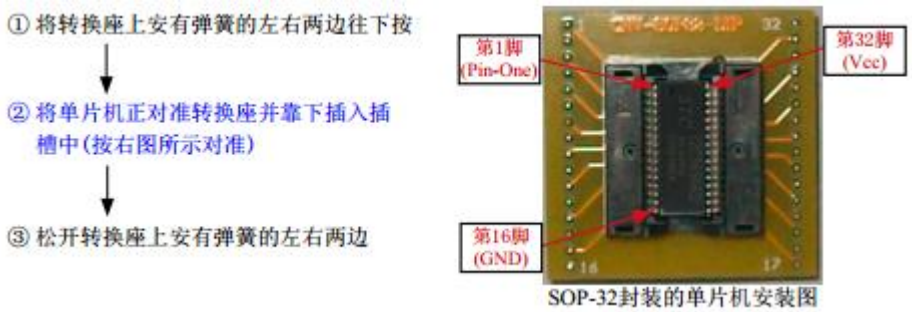
LQFP-48封装的单片机安装图　LQFP-44封装的单片机安装图　LQFP-32封装的单片机安装图

PLCC-44 封装的单片机按下图所示安装：



PLCC-44封装的单片机安装图

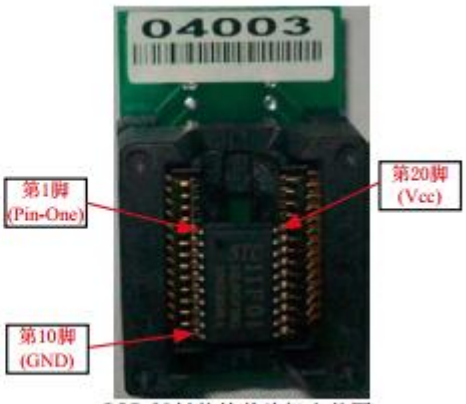
SOP-32 和 SOP-28/20 以及 SOP-16/8 封装的单片机按下图所示安装；



SOP-32封装的单片机安装图

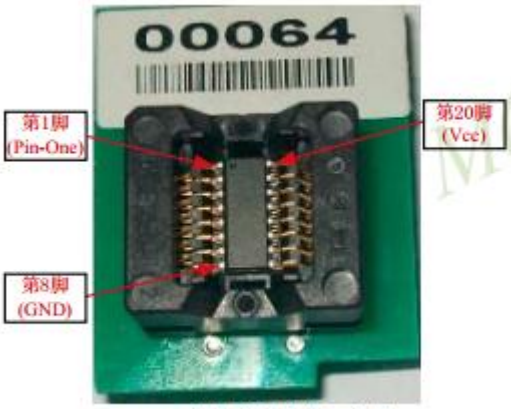


SOP-28封装的单片机安装图

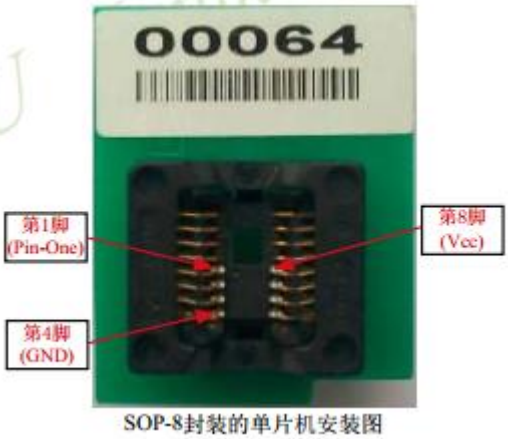


SOP-20封装的单片机安装图

SOP-28 和 SOP-20 封装的单片机用同一个转换座（SOP-28 转换座），将单片机正对准转换座并靠下插入转换座的插槽中。



SOP-16封装的单片机安装图



SOP-8封装的单片机安装图

SOP-16 和 SOP-8 封装的单片机用同一个转换座（SOP-16 转换座），将单片机正对准转换座并靠下插入转换座的插槽中。

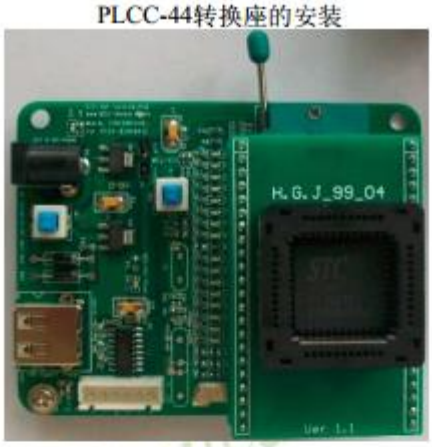
（2）将安有单片机的转换座安装在与单片机相对应的 AIapp-ISP 下载板锁紧座上，具体做法是：

将转换座正对准扳手的方向靠下插。

LQFP-48/LQFP-44转换座的安装



PLCC-44转换座的安装



LQFP-32转换座的安装



SOP-32转换座的安装



SOP-28转换座的安装

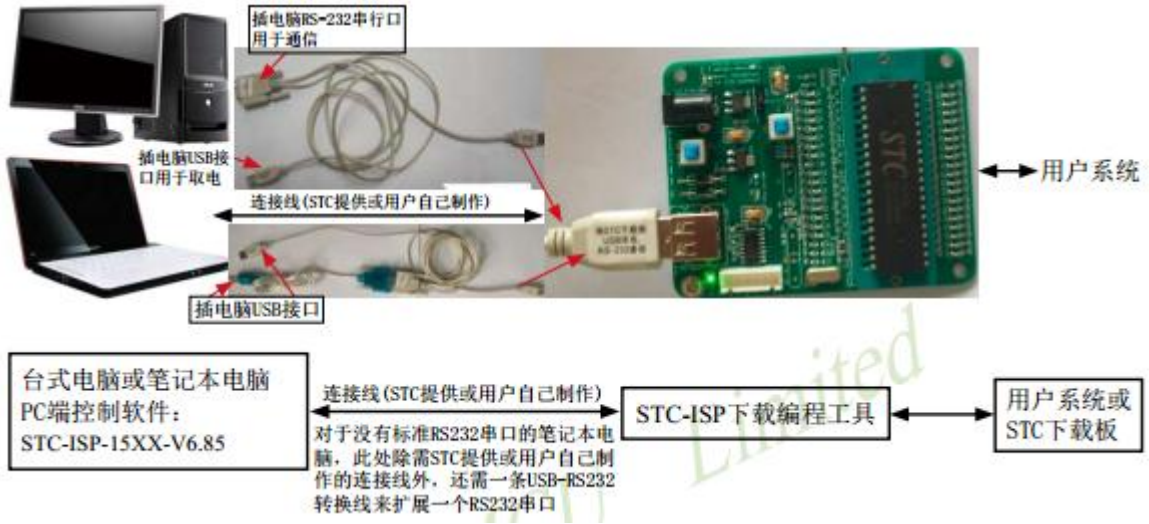


SOP-16转换座的安装



27.3.4.4 如何将 AIapp-ISP 下载板连接到电脑

AIapp-ISP 下载编程工具其实就是单片机通过 RS-232 转换器连接到电脑完成下载编程用户程序工作的。



有些笔记本电脑没有标准 RS-232 串行口，需一条 USB-RS232 转换线来扩展一个 RS-232 串行口。市场上有很多种 USB-RS232 转换线，有的是不能与 STC 下载板或电脑操作系统兼容的。请尽量选择用 CH340/CH341 做的 USB-RS232 转换线或让 STC 帮你购买经过测试的转换线。如果是用 PL2303 或 CP2102 制作的 USB-RS232 转换线，请尝试安装不同版本的驱动程序解决它们的不兼容问题。

关于硬件连接:

（1）.MCU/单片机 RXD（P3.0）---- RS-232 转换器 ---- 电脑 TXD（C0M Port Pin3）

（2）.MCU/单片机 TXD（P3.1）---- RS-232 转换器 ---- 电脑 RXD（C0M Port Pin2）

（3）.MCU/单片机 GND ---------- 电脑 GND（COM Port Pin5）

（4）.如果您的系统接了 RS-485 通信电路，推荐将 RS-485 电路接到「P1.6, P1.7]或[P3.6, P3.7]上，这样既方便又安全，且不用在 AIapp-ISP 下载编程工具中选择“下次冷启动时需[P3.2，P3.3] = [0, 0]才可以下载程序”。

（5）.RS-232 转换器可选用 MAX232/SP232（4.5-5.5V），MAX3232/SP3232（3V-5.5V）.

AIapp-ISP 下载板连接电脑的具体方式：

（1）根据单片机的工作电压在 AIapp-ISP 下载板上选择单片机电源电压

A）5V 单片机，将 MCU-VCC 和+5V 电源管脚短接

B）3V 单片机，将 MCU-VCC 和 3.3V 电源管脚短接

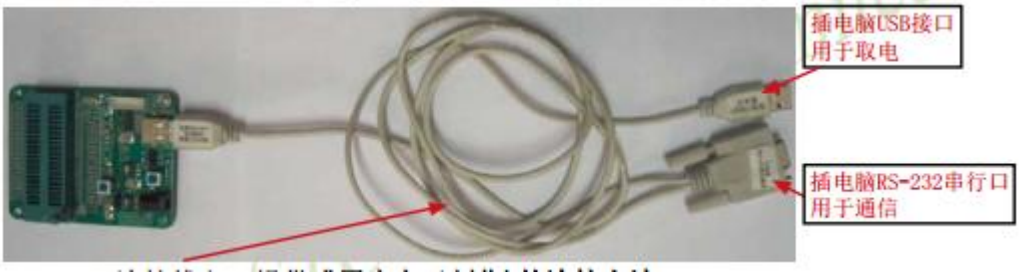


（2）将 AIapp-ISP 下载板连接到电脑端

根据用户所使用的电脑是否有 RS-232 串行口选择连接电脑的方式。

A）如果用户电脑有 RS-232 串行口，参照下图连接。

下面是 AIapp-ISP 下载板连接有 RS-232 串行口电脑的方式:



连接线（STC 提供或用户自己制作）的连接方法：

① 将一端有 9 芯连接座的插头插入电脑 RS-232 串行接口插座用于通信；

② 将连接线的“从电脑 USB 口取电”的 USB 插头插入电脑 USB 接口用于取电；

③ 将连接线中“接 STC 下载板”的 USB 插头插入 AIapp-ISP 下载编程工具的 PCB 板 USB1 插座用于 RS-232 通信和供电

B）.如果用户电脑没有 RS-232 串行口，参照下图连接。

下面是 AIapp-ISP 下载板连接没有 RS-232 串行口电脑（需一条 USB-RS232 转换线扩展一个 RS232 串行口）的方式：



连接线（STC 提供或用户自己制作）和 USB-RS232 转换线的连接方法：

① 将连接线中一端有 9 芯连接座的插头插入 USB-RS232 转换线的相应插座中；

② 将连接线的“从电脑 USB 口取电”的 USB 插头插入电脑 USB 接口用于取电；

③ 将 USB-RS232 转换线中的 USB 插头插入电脑 USB 接口用于通信

④ 将连接线中“接 STC 下载板”的 USB 插头插入 AIapp-ISP 下载编程工具的 PCB 板 USB1 插座用于 RS-232 通信和供电

（3）.其他插座不需连接

（4）.“系统电源开关 Power ON” 开关处于非按下状态，此时 MCU-VCC Power 灯不亮，没有给单片机通电

（5）.通过“[P3.2, P3.3] = [0,0]（对于 STC12 系列、STC11 系列、STC10 系列、STC89 系列及 STC90 系列为[P1.0, P1.1] = [0,0]）”控制开关：

处于非按下状态，[P3.2, P3.3] = [1, 1]，不短接到地；

处于按下状态，[P3.2, P3.3] = [0, 0]，短接到地。

如果单片机已被设成“下次冷启动[P3.2, P3.3] = [0, 0] 才判 P3.0 有无合法下载命令流”

就必须将此开关处于按下状态，让单片机的[P3.2, P3.3]短接到地

（6）.将单片机插进锁紧座，锁紧单片机，注意单片机是 8-Pin/20-Pin/28-Pin/32-Pin/40-Pin 的，锁紧座是 40-Pin，我们的设计是靠下插，单片机地线（Gnd）对准锁紧座的地线（Gnd）插。

27.3.5 针对 USB-RS232 转换线不兼容问题的几点说明

有些新式笔记本电脑没有标准 RS-232 串行口，则需要一条 USB-RS232 转换线来扩展一个 RS-232 串行口。但有些 USB-RS232 转换线与 STC 下载板或电脑操作系统是不能兼容的，这里针对这些不兼容问题提出几点解决方法：

（1）请尽量选择用 CH340/CH341 制作的 USB-RS232 转换线

（2）对于市场上有些用 PL2303 或 CP2102 制作的 USB-RS232 转换线，尝试安装不同版本的驱动程序解决它的不兼容问题。

（3）尝试在 AIapp-ISP 控制下载软件中，将最高波特率和最低波特率设置为相等且都为 2400，重新连接。



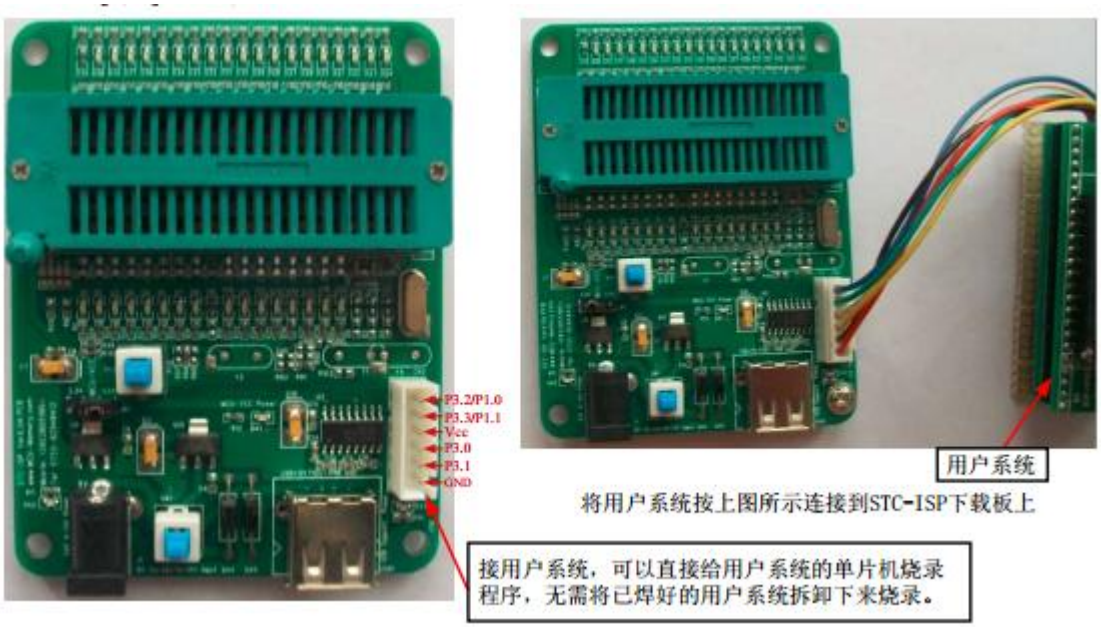
（4）让 STC 帮您购买经过测试的转换线。

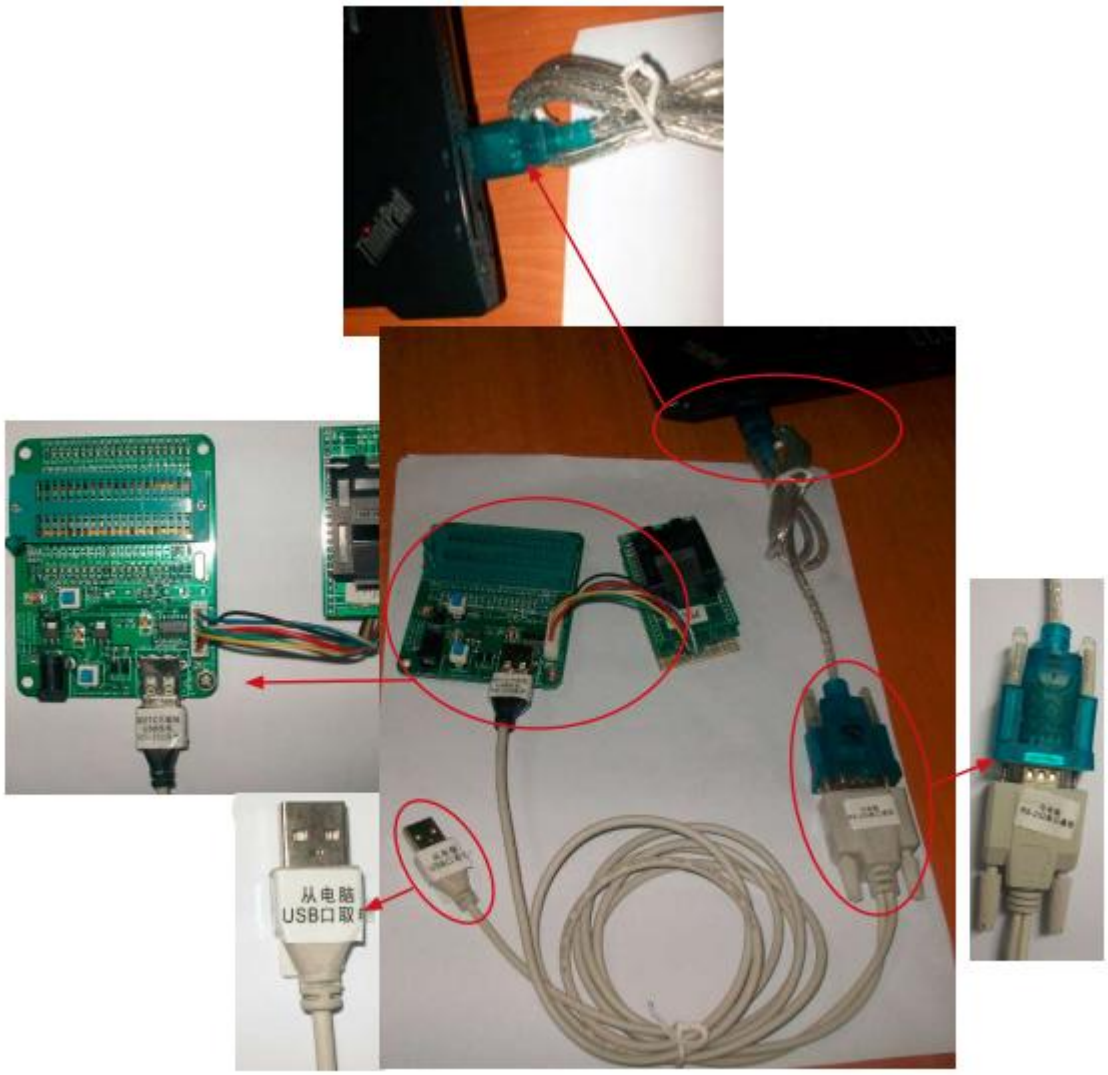
27.3.6 如何用 AIapp-ISP 下载板给在用户系统上的单片机烧录用户程序

利用 STC 系列 ISP 下载编程工具（其实就是单片机通过 RS-232 转换器连接到电脑）进行 RS-232 转换。

单片机在用户自己的板上完成下载/烧录：

1. U1-Socket 锁紧座不得插入单片机
2. 将用户系统上的电源（MCU-VCC, GND）及单片机的[P3.0, P3.1]接入转换板的“白色六芯插座”，如下图所示，这样用户系统上的单片机就具备了与电脑进行通信的能力
3. 将用户系统的单片机的[P3.2, P3.3]（对于 STC12 系列、STC11 系列、STC10 系列、STC89 系列及 STC90 系列为[P1.0, P1.1]）接入转换板“白色六芯插座” （如果需要的话）
4. 如须[P3.2, P3.3] = [0, 0]，短接到地，可在用户系统上将其短接到地，或将[P3.2, P3.3]也从用户系统引到 STC 系列 ISP 下载编程工具（其实就是单片机通过 RS-232 转换器连接到电脑）上，将“控制[P3.2, P3.3]同时为[0, 0]的开关”按下，则[P3.2, P3.3] = [0, 0]。
5. 将 AIapp-ISP 下载板连接到电脑上进行 RS232 通信（具体连接方式见下页图）
6. 给单片机上电复位（注意是从用户系统自供电，不要从电脑 USB 取电，电脑 USB 座不插）
7. 关于软件：选择“Download/下载”
8. 下载程序时，如用户板有外部看门狗电路，不得启动，单片机必须有正确的复位，但不能在 ISP 下载程序时被外部看门狗复位，如有，可将外部看门狗电路 WDI 端或 WDO 端浮空。
9. 如系统接了 RS-485 通信电路，推荐将 RS-485 电路接到[P1.6, P1.7]或[P3.6, P3.7]上，这样既方便又安全，且不用在 AIapp-ISP 下载编程工具中选择“下次冷启动时需[P3.2, P3.3] = [0, 0]才可以下载程序”。





将连有用户系统的 AIapp-ISP 下载板按左图所示连接到电脑上，注意以下几点：

（1）AIapp-ISP 下载板的锁紧座不得插入单片机；

（2）“从电脑 USB 口取电”的 USB 插头悬空，不要插入电脑，因为是从用户系统自供电的。

（3）接 STC 下载板的 USB 插头仅用于 RS232 通信。

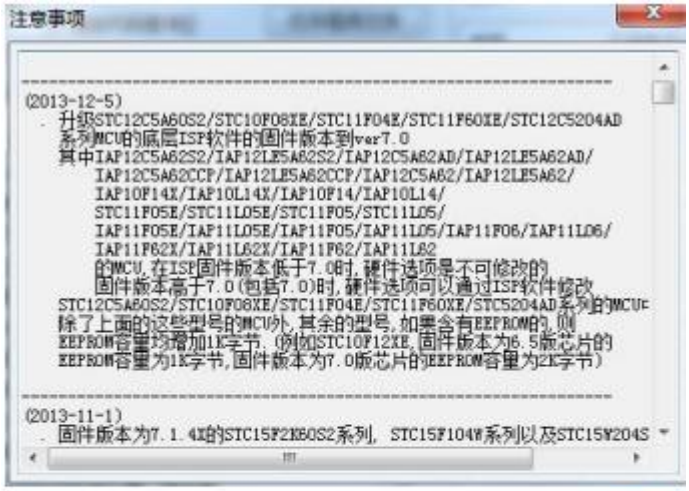
27.3.7 电脑端的 AIapp-ISP 控制软件（Ver6.95G）的界面使用说明



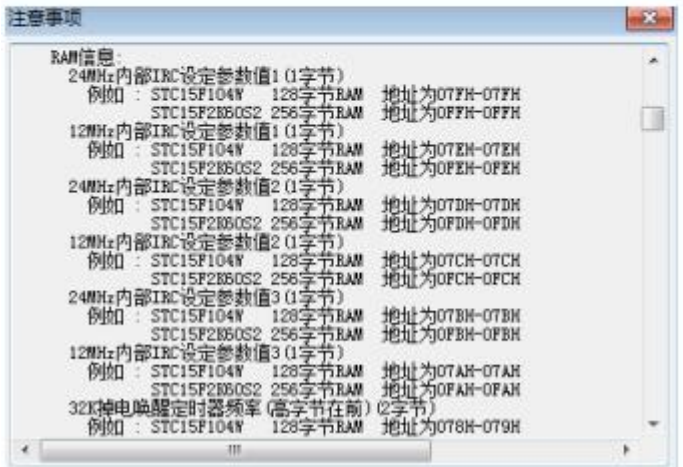
最新的 ISP 下载控制软件 V6.95G 的界面如上图所示。该软件新增了许多新功能（如扫描当前系统中可用的串口、波特率计算器、软件延时计算器、选型/价格/样品表等）。下文将详细介绍该 AIapp-ISP-V6.95G 软件的各个功能。


STC-ISP (V6.79) (销售电话: 0513-55012928) 官网:www
单片机型号 STC15F2K60S2
引脚数 Auto
串口号 COM5
扫描
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
脱机下载/U8/U7
程序加密后传输
选择使用内部IRC时钟(不选为外部时钟)
输入用户程序运行时的IRC频率 11.0592 MHz
使用快速下载模式
下次冷启动时, P3.2/P3.3为0/0才可下载程序
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 3.82 V
低压时禁止EEPROM操作
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
下次下载用户程序时擦除用户EEPROM区
P2.0脚上电复位后为低电平(不选为高电平)
串口1数据线[RxD, TxD]从[P3.0, P3.1]切换到[P3.6, P3.7], P3.7脚输出P3.6脚的输入电平
P3.7是否为强推挽输出
在程序区的结束处添加重要测试参数(包括BandGap电压, 32K掉电唤醒定时器频率, 24M和11.0592M内部IRC设定参数)
选择Flash空白区域的填充值 FF
下载/编程
停止
重复编程
检测MCU选项
注意/帮助
重复延时 3 秒
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
选择STC系列单片机的型号
选择STC15系列单片机的封装
扫描当前系统中可用的串口
用户根据实际使用效果选择限制最高或最低波特率，如57600, 38400,19200,2400或Auto Buad
打开用户的程序代码文件
打开EEPROM数据文件
选择时钟(内部R/C时钟)频率(可输入)
是否使用较快速度的内部振荡器频率进行下载
选择： 使用较快频率的内部振动器
不选择：使用较慢频率的内部振动器
下次是否需要[P3.2, P3.3]同时为低电平时才可下载程序
选择： [P3.2, P3.3]同时为低电平时才可下载程序
不选择：下载时不检测[P3.2, P3.3]的电平
上电复位时，是否需要额外的复位延时
选择： 需要额外的复位延时
不选择：一般长度的复位延时
是否需要将复位引脚当作普通I/O口来使用
选择： 复位引脚当作普通I/O口
不选择：复位引脚仍为复位脚
当电压低于设定的低压检测门槛电压时，芯片是复位还是中断
选择： 检测到低压时复位
不选择：检测到低压时不复位而产生低压中断
建议：当振动器频率高于20MHz时
对于3V的芯片，低压检测门槛电压建议选择2.5V以上
对于5V的芯片，低压检测门槛电压建议选择4.11V以上
当芯片处于空闲状态时，是否需要停止内部看门狗计数
选择： 空闲状态时停止计数
不选择：空闲状态时继续计数
新的设置冷启动后（彻底停电后再上电），才生效
如P3.0/P3.1外接RS-485/RS-232等通信电路，建议选择P3.2/P3.3等于0/0才可以下载程序，如不同时为0/0，则跨过系统ISP引导程序，直接运行用户程序。
大批量生产时使用

点击界面上的注意/帮助按钮后出现下面的对话框：







脱机下载界面：



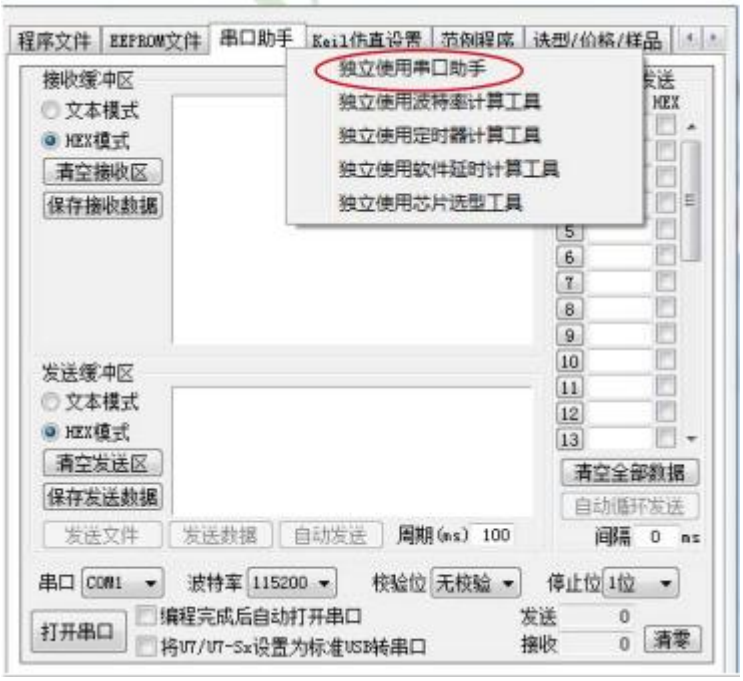
RS485控制界面：



串口助手界面：



在串口助手工具选择页上单击鼠标右键进行选择，可以将串行口助手从 AIapp-ISP 下载编程软件的主界面中独立出来（如下所示），关闭独立使用的工具可以再次返回主界面。



最新的 AIapp-ISP-V6.95G 软件集成了波特率计算器，利用波特率计算器可以很方便地求出波特率，并可以生成相应的代码（C 或 ASM 代码）。波特率计算器界面如下所示：



最新的 AIapp-ISP-V6.95G 软件还集成了定时器计算器，定时器计算器也可以生成相应的代码（C 或 ASM 代码），根据用户的设置对定时器的各相关寄存器进行初始化。定时器计算器界面如下所示：

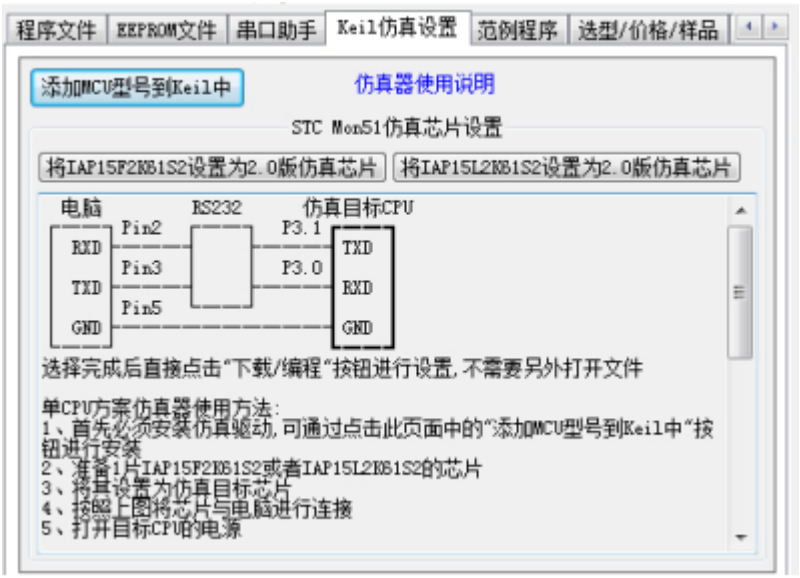


另外，最新的 AIapp-ISP-V6.95G 软件还集成了软件延时计算器，软件延时计算器也可以生成相应的代码（C 或 ASM 代码），根据用户的设置可以生成相应的延时子函数。软件延时计算器界面如下所示：

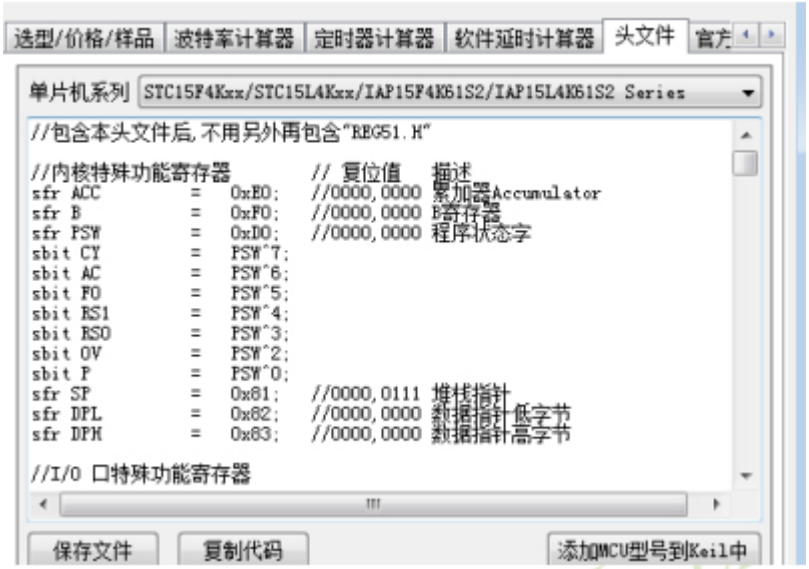


除串口助手外，波特率计算器、定时器计算器、软件延时计算器都可以从 AIapp-ISP 下载编程软件的主界面中独立出来，关闭独立使用的工具可以再次返回主界面。

最新的 AIapp-ISP-V6.85 软件还设计了“Keil 仿真设置”选项，如下图所示



最新的 AIapp-ISP-V6.95G 软件还包含了头文件，供用户查询和复制。头文件如下所示：



另外，用户还可以在最新的 AIapp-ISP-V6.85 软件中查询 STC 系列单片机的选型和价格。



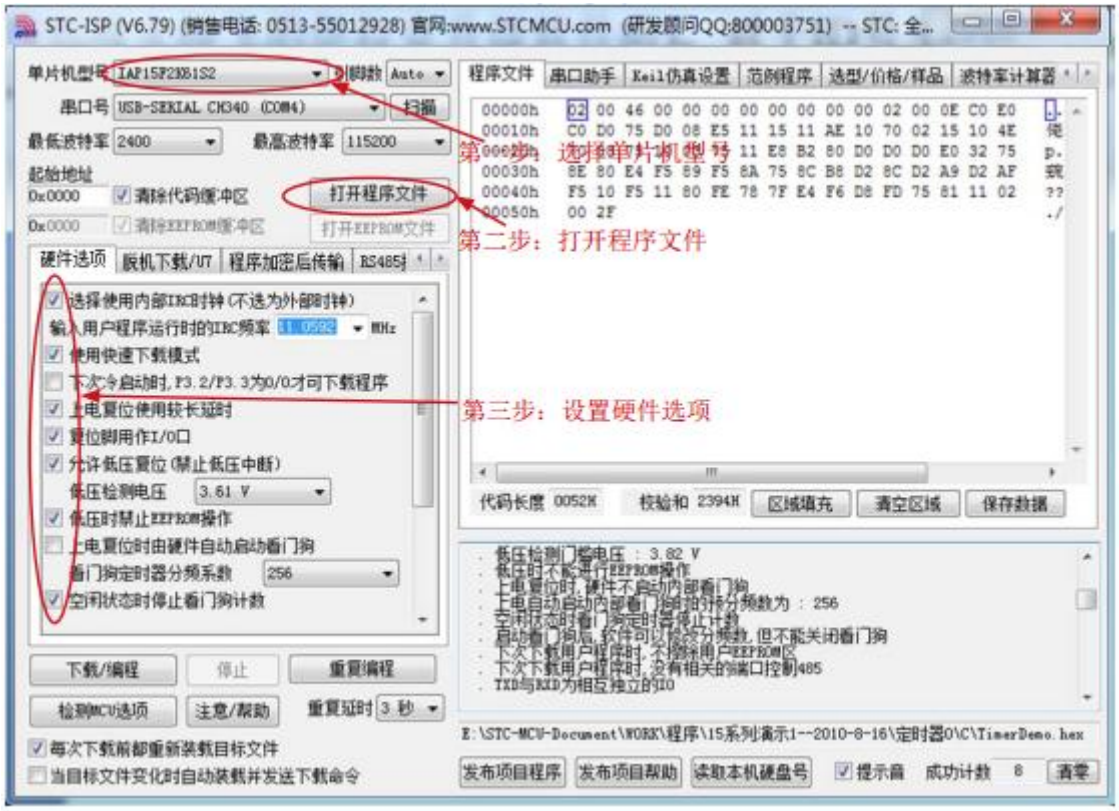
27.3.8 AIapp-ISP 控制软件（Ver6.95G）发布项目程序使用说明

发布项目程序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。

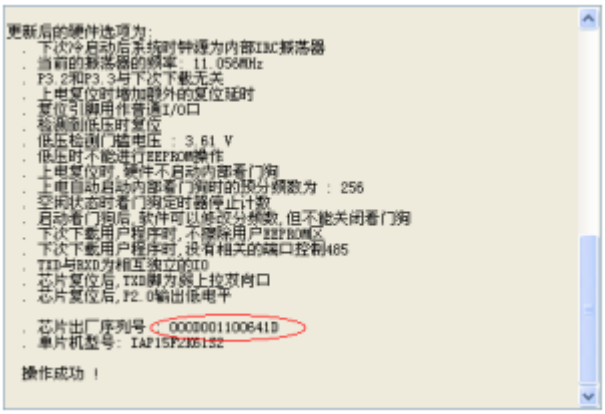
关于界面，用户可以自己进行定制（用户可以自行修改发布项目程序的标题、按钮名称以及帮助信息），同时用户还可以指定目标电脑的硬盘号和目标芯片的 ID 号，指定目标电脑的硬盘号后，便可以控制发布应用程序只能在指定的电脑上运行（防止烧录人员将程序轻易从电脑盗走，如通过网络发走，如通过 U 盘烤走，防不胜防，当然盗走你的电脑那就没办法了，所以 STC 的脱机下载工具比电脑烧录安全，能限制可烧录芯片数量，让前台文员小姐烧，让老板娘烧都可以），拷贝到其它电脑，应用程序不能运行。同样的，当指定了目标芯片的 ID 号后，那么用户代码只能下载到具有相应 ID 号的目标芯片中（对于一台设备要卖几千万的产品特别有用---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，该芯片的 ID 号即为“000D001100641D”（如不需要对目标芯片的 ID 号进行校验，可跳过此步）



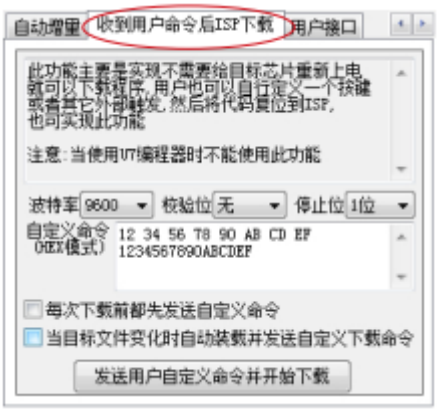
5、设置自动增量（如不需要自动增量，可跳过此步



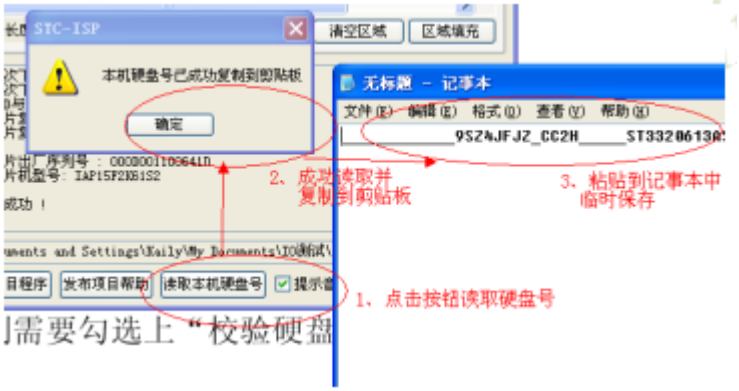
6、设置 RS485 控制信息（如不需要 RS485 控制，可跳过此步）



7、设置“收到用户命令后 ISP 下载”（如不需要此功能，可跳过此步）



8、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）

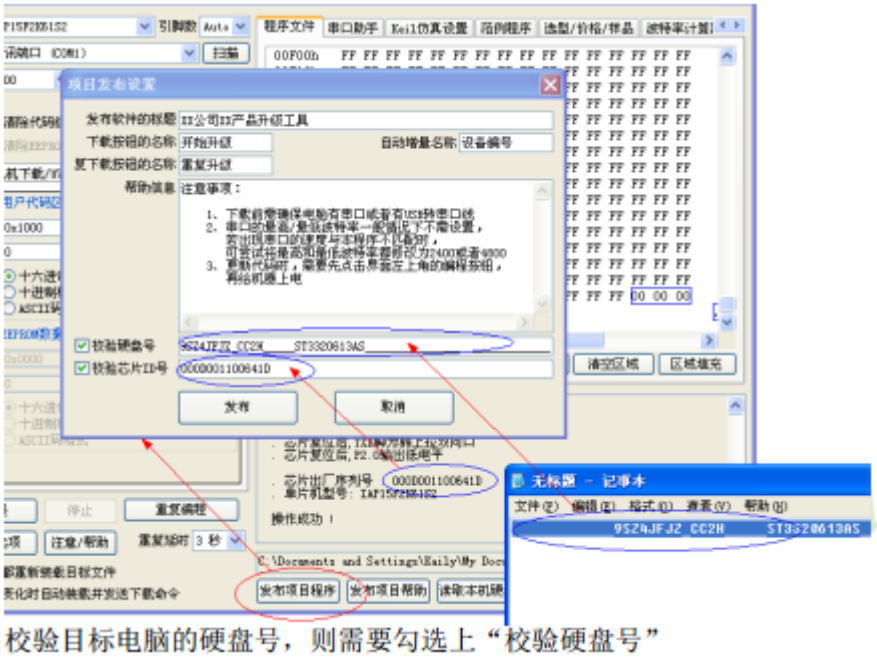


9、点击“发布项目程序”按钮，进入发布应用程序的设置界面。

10、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

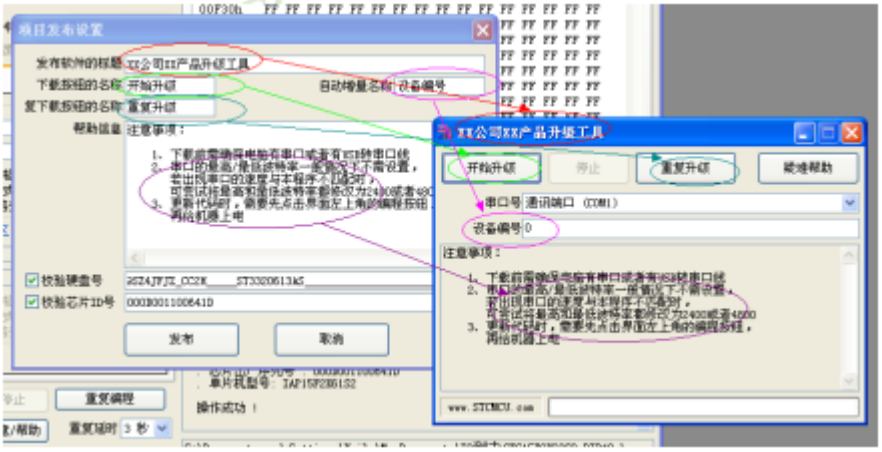
11、若需要校验目标电脑的硬盘号,则需要勾选上“校验硬盘号”，并在后面的文本框内输入前面所记下的目标电脑的硬盘号

12、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



校验目标电脑的硬盘号，则需要勾选上“校验硬盘号”

13、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。如下图，设置界面中所定制的内容与发布文件是一一对应的。



注意：

校验硬盘号与校验目标芯片 ID 号的功能仅对如下系列及新出的单片机有效：

STC15F2K60S2/STC15L2K60S2

IAP15F2K61S2/IAP15L2K61S2

STC15F101W/STC15L101W

IAP15F105W/STC15L105W

STC15W104SW/IAP15W105W

27.3.9 “程序加密后传输”功能说明

----防止烧录时通过串口分析出程序代码

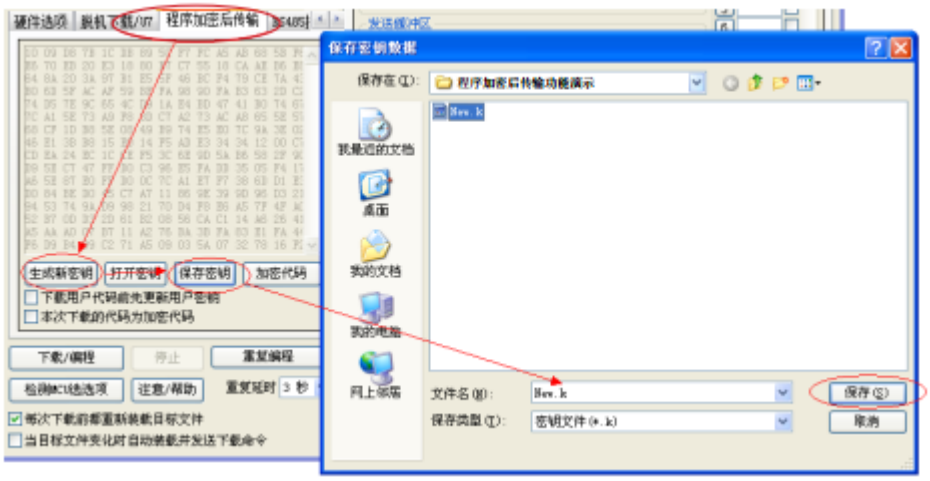
目前，所有的普通串口下载烧录编程都是采用明码通信的（电脑和目标芯片通信时，或脱机下载板和目标芯片通信时），问题：如果烧录人员通过分析下载烧录编程时串口通信的数据，高手是可以在烧录时在串口上引 2 根线出来，通过分析串口通信的数据分析出实际的用户程序代码的。当然用 STC 的脱机下载板烧程序总比用电脑烧程序强（防止烧录人员将程序轻易从电脑盗走，如通过网络发走，如通过 U 盘烤走，防不胜防，当然盗走你的电脑那就没办法了，所以 STC 的脱机下载工具比电脑烧录安全，让前台文员小姐烧，让老板娘烧都可以）。即使是 STC 全球首创的脱机下载工具，对于要防止天才的不法分子在脱机下载工具烧录的过程中通过分析串口通信的数据，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 STC15 系列单片机所提供的“程序加密后传输”功能。目前，我司是全球第一家可以防范用户将程序代码给烧录人员烧录时烧录人员通过串口分析出目标程序代码的公司。

“程序加密后传输”功能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：

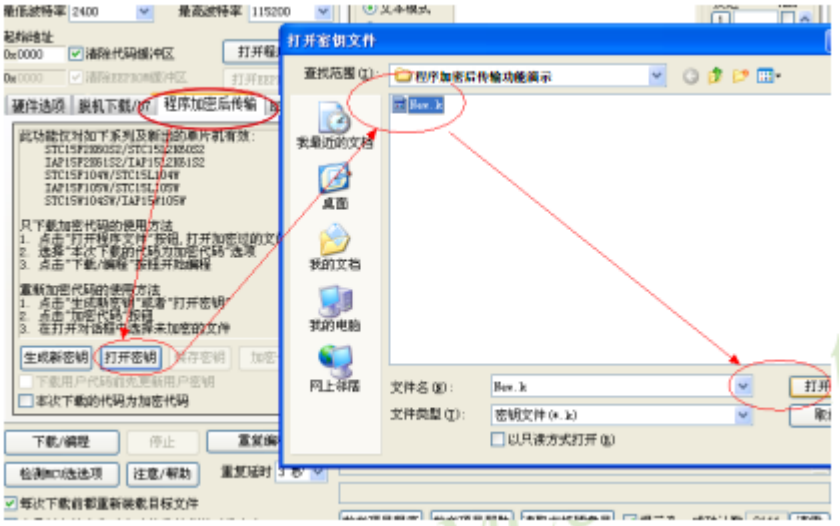
1、生成并保存新的密钥

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”。

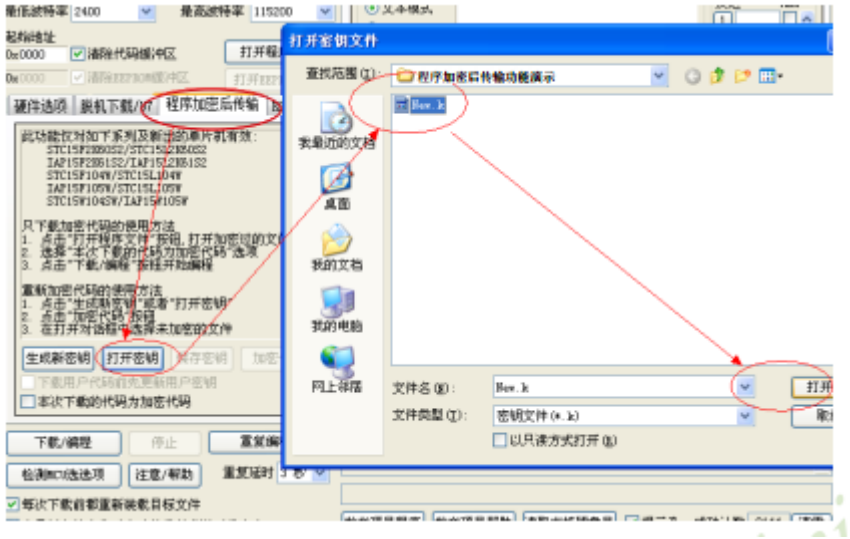


然后返回到“程序加密后传输”页面中点击“加密代码”按钮，如下图所示，首先会弹出“打开源文件（未加密）”的对话框，此时选择的是原始的未加密的代码文件。

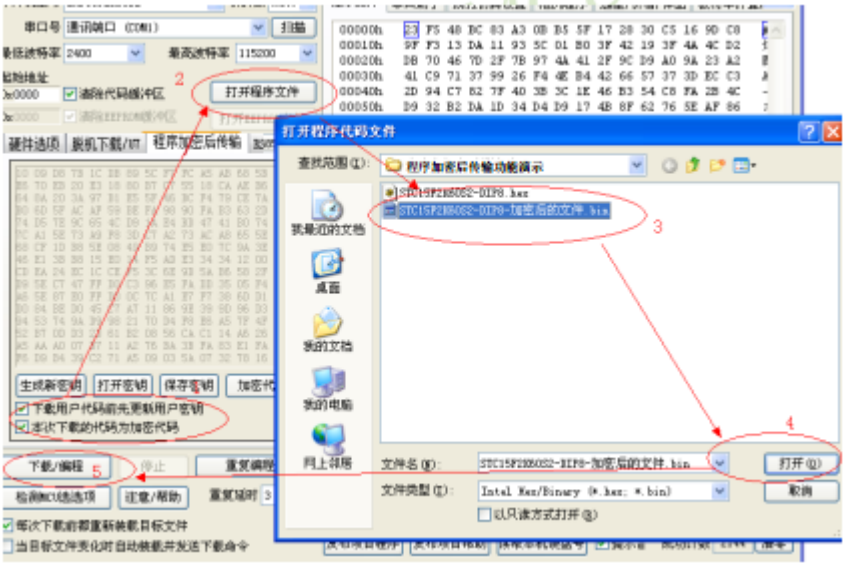


3、将用户密钥更新到目标芯片中

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“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”。

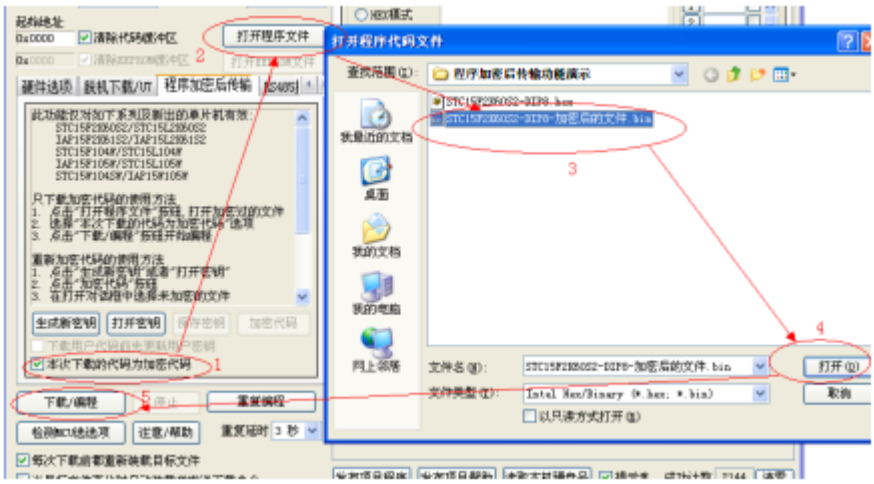


密钥打开后，如下图所示，勾选上“下载用户代码前先更新用户密钥”选项和“本次下载的代码为加密代码”的选项，然后打开我们之前加密过后的文件，打开后点击界面左下角的“下载/编程”按钮，按正常方式对目标芯片下载完成即可更新用户密钥。



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：



首先在“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）。

注意：

“程序加密后传输”功能仅对如下系列及新出的单片机有效：
STC15F2K60S2/STC15L2K60S2
IAP15F2K61S2/IAP15L2K61S2
STC15F101W/STC15L101W
IAP15F105W/STC15L105W
STC15W104SW/IAP15W105W

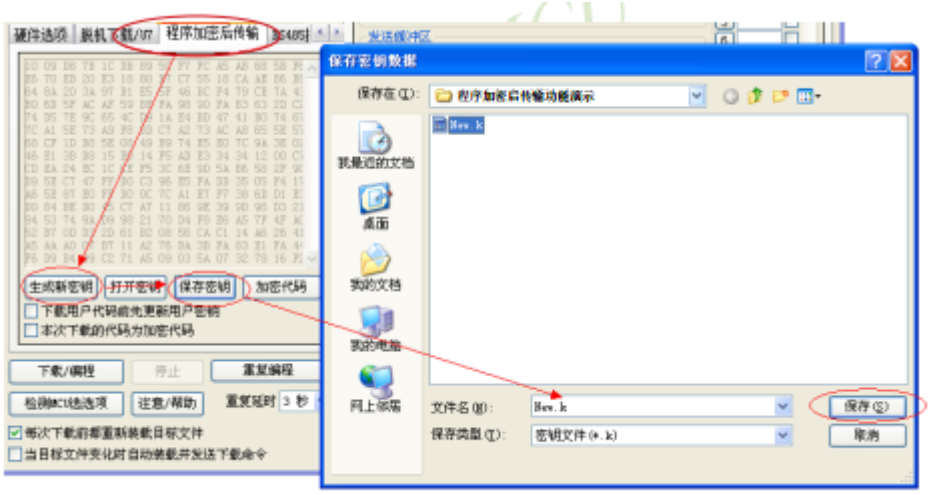
27.3.10 "发布项目程序"+"程序加密后传输"结合使用

“发布项目程序”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”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。

下面用具体的实例来举例说明"发布项目程序"与“程序加密后传输”结合使用的方法，首先讲解代码的加密以及加密芯片的制作方法

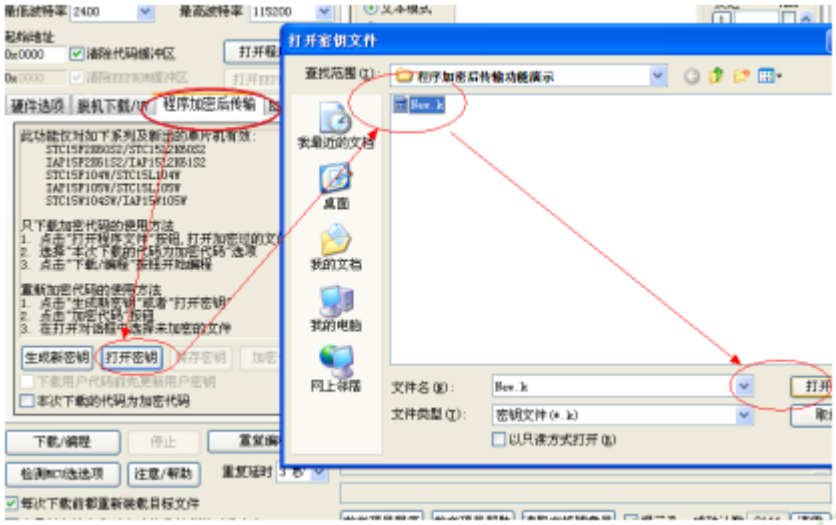
1、生成并保存新的密钥

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）。比如我们将密钥保存为“New.k”。

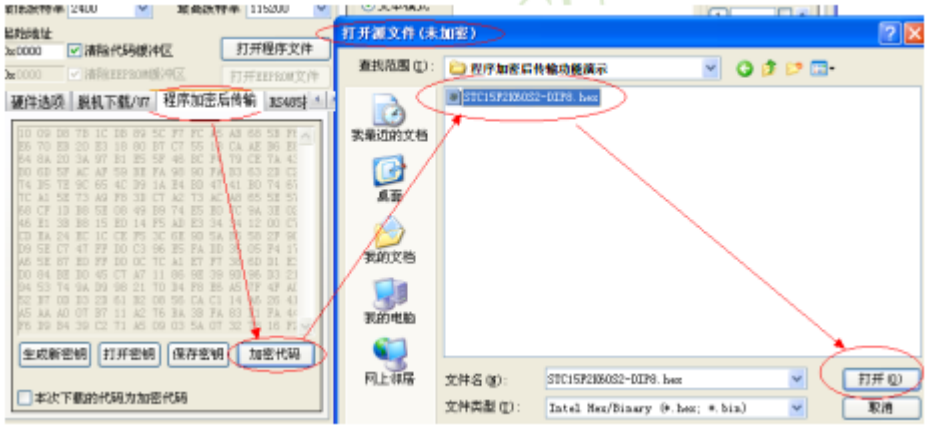


2、代码文件加密

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“程序加密后传输载”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

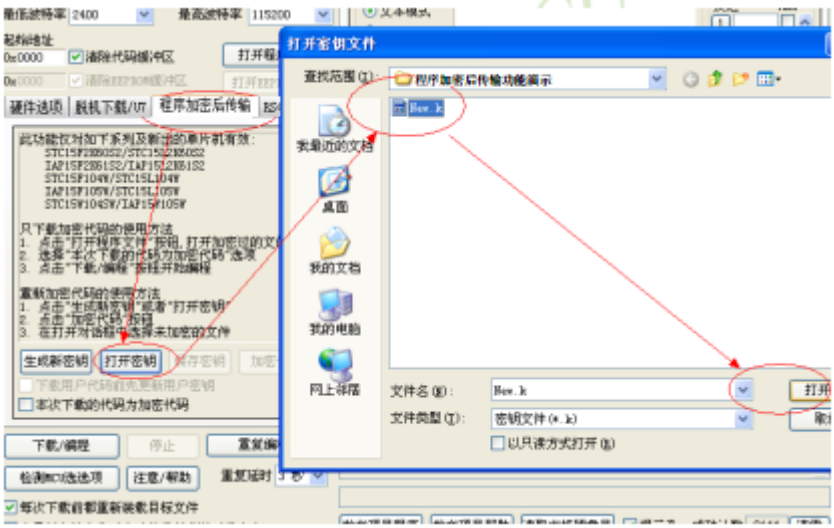


点击打开按钮后，马上有会弹出一个类似的对话框，但此时是对加密后的文件进行保存的对话框。如下图所示，点击保存按钮即可保存加密后的文件。

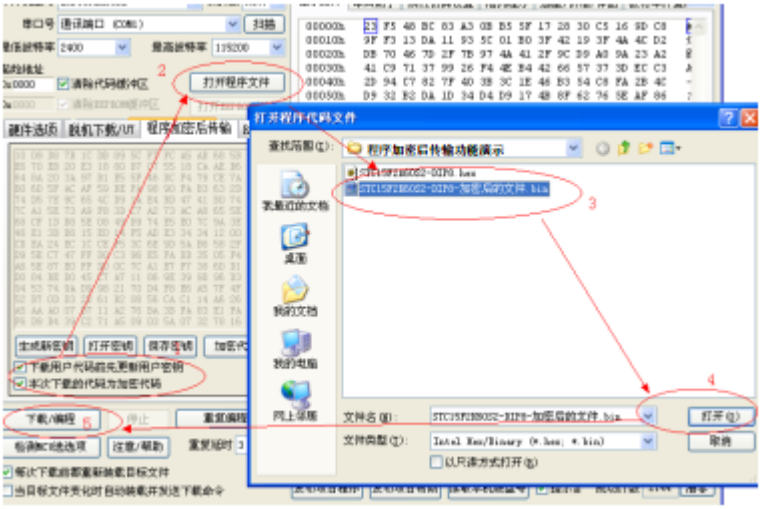


3、将用户密钥更新到目标芯片中

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“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”。



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。



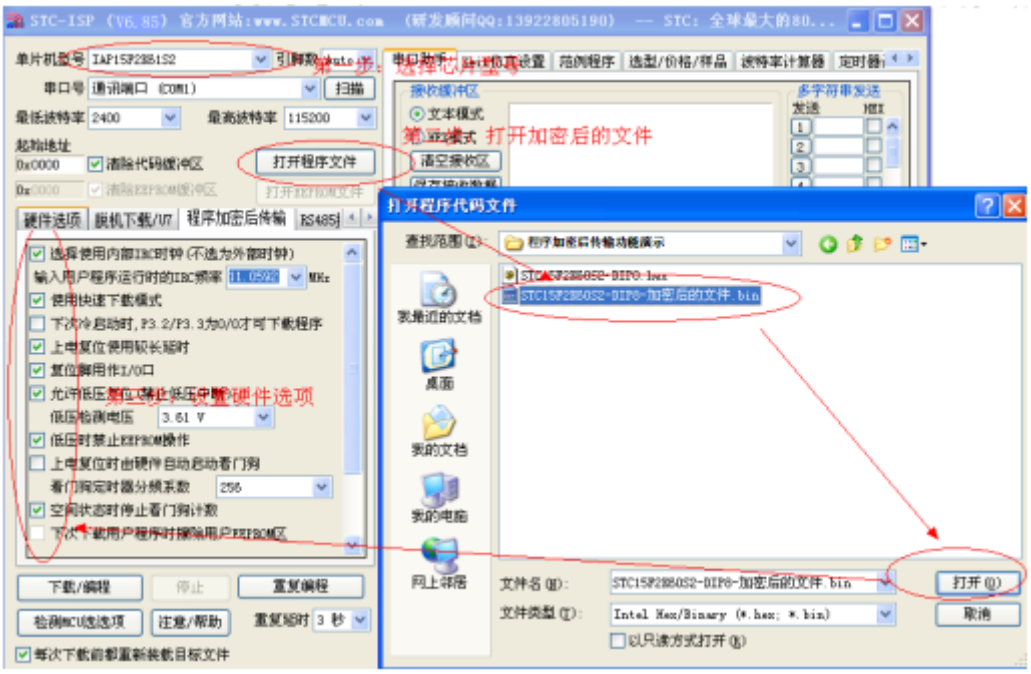
经过上面的三步，此时的目标芯片便具有还原加密代码的功能。便可将目标芯片提供给终端客户使用。

下面讲解如何发布加密项目程序

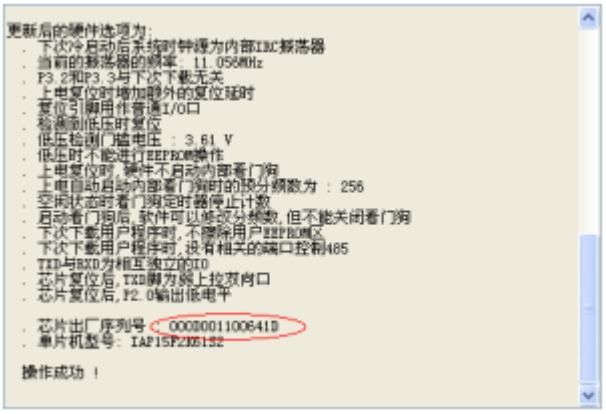
1、首先选择目标芯片的型号

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，该芯片的 ID 号即为“000D001100641D”（如不需要对目标芯片的 ID 号进行校验，可跳过此步）



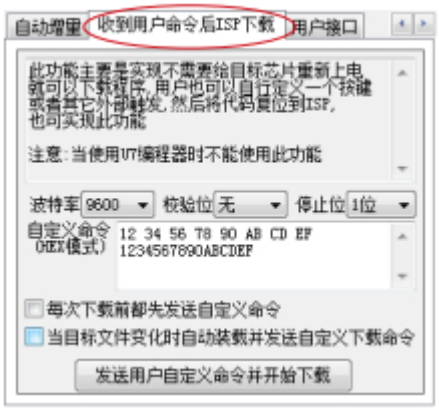
5、在“程序加密后传输”页面中选择“本次下载的代码为加密代码”选项（注意：加密下载时不支持自动增量）



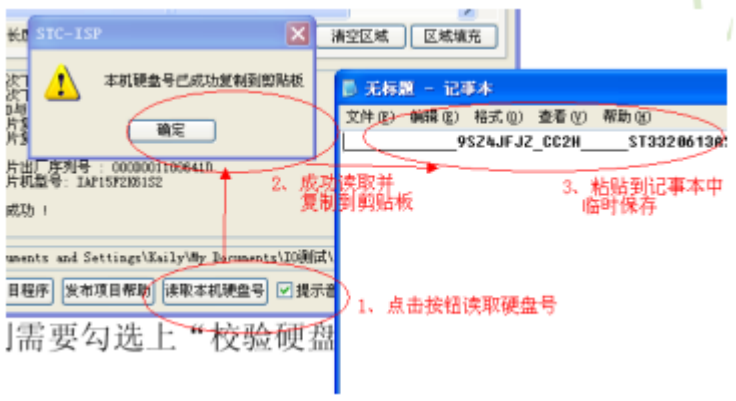
6、设置 RS485 控制信息（如不需要 RS485 控制，可跳过此步）



7、设置“收到用户命令后 ISP 下载”（如不需要此功能，可跳过此步）



8、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）

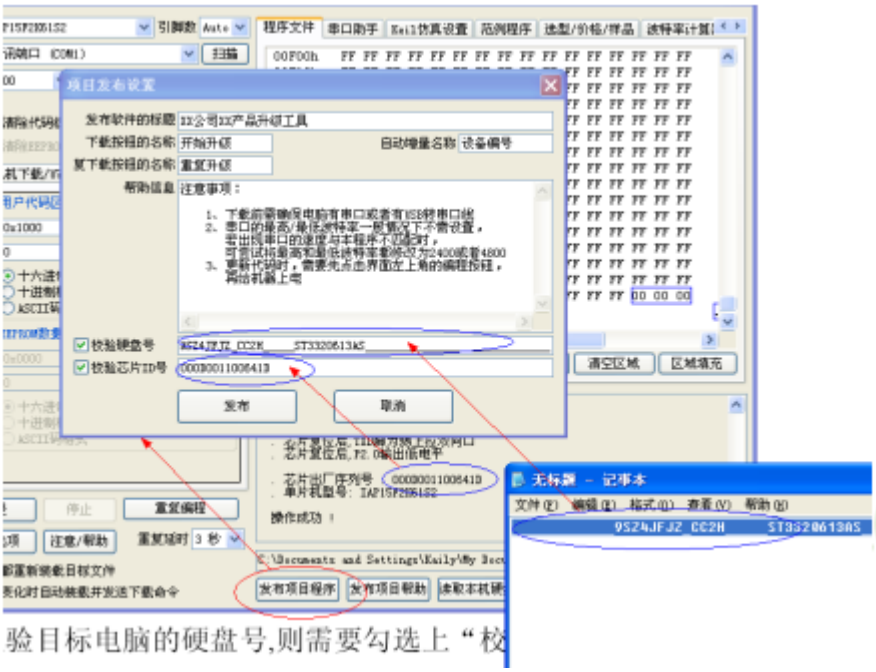


9、点击“发布项目程序”按钮，进入发布应用程序的设置界面。

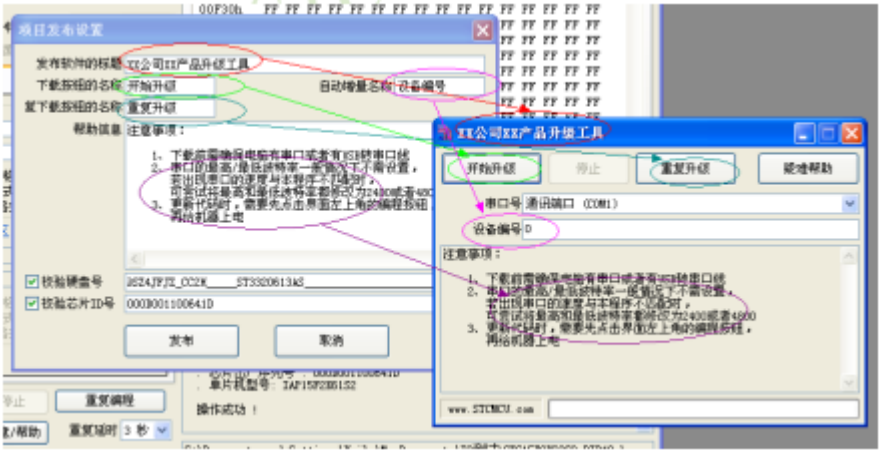
10、根据各自的需要，修改发布软件的标题、下载按钮的名称、重复下载按钮的名称、自动增量的名称以及帮助信息

11、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

12、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



13、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。如下图，设置界面中所定制的内容与发布文件是一一对应的。



上面的整个步骤基本与发布项目程序的步骤相一致，唯一不同的地方是打开的不是原始文件，而是加密后的文件，而且一定要勾选上“本次下载的代码为加密代码”的选项。

27.3.11 运行用户程序时收到用户命令后自动启动 ISP 下载（不停电）

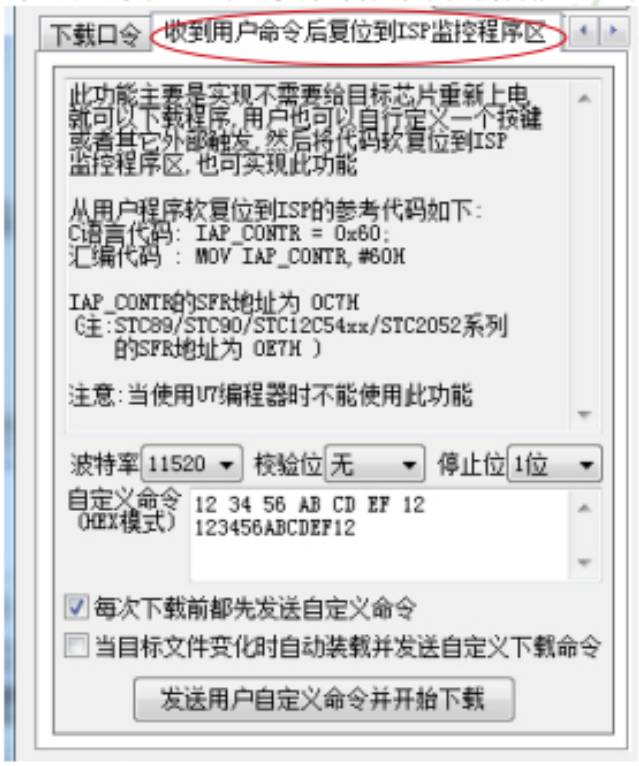
“运行用户程序时收到用户命令后自动启动 ISP 下载”（即软件中的“收到用户命令后 ISP 下载”）与“程序加密后传输”是两种完全不同功能。相对“程序加密后传输”的功能而言，“运行用户程序时收到用户命令后自动启动 ISP 下载”的功能要简单一些。

具体的功能为：电脑或脱机下载板在开始发送真正的 ISP 下载编程握手命令前，先发送用户自定义的一串命令（关于这一串串口命令，用户可以根据自己在应用程序中的串口设置来设置波特率、校验位以及停止位），然后再立即发送 ISP 下载编程握手命令。

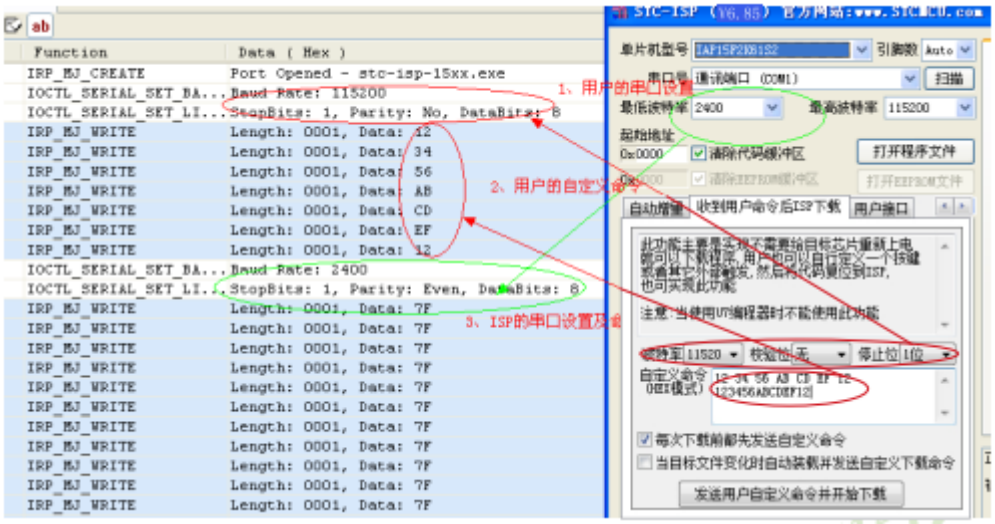
“运行用户程序时收到用户命令后自动启动 ISP 下载”这一功能主要是在项目的早期开发阶段，实现不断电（不用给目标芯片重新上电）即可下载用户代码。具体的实现方法是：

用户需要在自己的程序中加入一段检测自定义命令的代码，当检测到后，执行一句“MOV IAP_CONTR, #60H”的汇编代码或者“IAP_CONTR = 0x60;”的 C 语言代码，MCU 就会自动复位到 ISP 区域执行 ISP 代码。

如下图所示，将自定义命令设置为波特率为 115200、无校验位、一位停止位的命令序列：0x12、0x34、0x56、0xAB、0xCD、0xEF、0x12。当勾选上“每次下载前都先发送自定义命令”的选项后，即可实现自定义下载功能



点击“发送用户自定义命令开始下载”或者点击界面左下角的“下载/编程”按钮，应用程序便会发送如下所示的串口数据。


Function
Data (Hex)
IRP_MJ_CREATE Port Opened - stc-isp-15xx.exe
IOCTL_SERIAL_SET_BA... Baud Rate: 115200
IOCTL_SERIAL_SET_LI... StopBits: 1, Parity: No, DataBits: 8
IRP_MJ_WRITE Length: 0001, Data: 12
IRP_MJ_WRITE Length: 0001, Data: 34
IRP_MJ_WRITE Length: 0001, Data: 56
IRP_MJ_WRITE Length: 0001, Data: AB
IRP_MJ_WRITE Length: 0001, Data: CD
IRP_MJ_WRITE Length: 0001, Data: EF
IRP_MJ_WRITE Length: 0001, Data: 12
IOCTL_SERIAL_SET_BA... Baud Rate: 2400
IOCTL_SERIAL_SET_LI... StopBits: 1, Parity: Even, DataBits: 8
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
1、用户的串口设置
2、用户的自定义命令
3、ISP的串口设置及
单片机型号
引脚数
扫描
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
自动增量
收到用户命令后ISP下载
用户接口
波特率
11520
校验位
无
停止位
1位
自定义命令
(HEX模式)
12 34 56 AB CD EF 12
123456ABCDEF12
每次下载前都先发送自定义命令
当目标文件变化时自动装载并发送自定义下载命令
发送用户自定义命令并开始下载

27.3.12 用户接口

AIapp-ISP-V6.95G 下载编程软件新增了用户接口软件，如下图所示。用户接口功能主要实现了保留用户芯片中的重要信息（如：序列号）不被破坏的作用。



使用用户接口功能时，PC 机或者 U7 编程器首先与用户单片机通讯，将单片机的重要信息（如：序列号等）读取出来并保存，然后用户可以设置发送给用户代码的自定义命令（如设置发送给用户代码的读数据命令或复位命令），用户代码可以接收自定义命令。当用户代码收到复位命令后可以控制目标单片机自动复位，若用户未设置复位命令，则用户需手动给目标单片机重新上电，当目标单片机上电复位后，就开始更新代码了，此时更新的代码包括上述 PC 机或 U7 编程器所保存的重要信息和用户新代码，即将 PC 机或 U7 编程器所保存的重要信息和用户新代码一并写入了目标单片机中，从而实现了保留目标单片机中的重要信息不被破坏的目的。

注意：只有使用普通串口或 USB 转串口直接对单片机进行在线下载或者使用 U8 编程器进行脱机下载时，用户接口功能才可用；当使用 U8 编程器在线联机下载时，用户接口功能并不可用。

27.3.13 RS485 控制

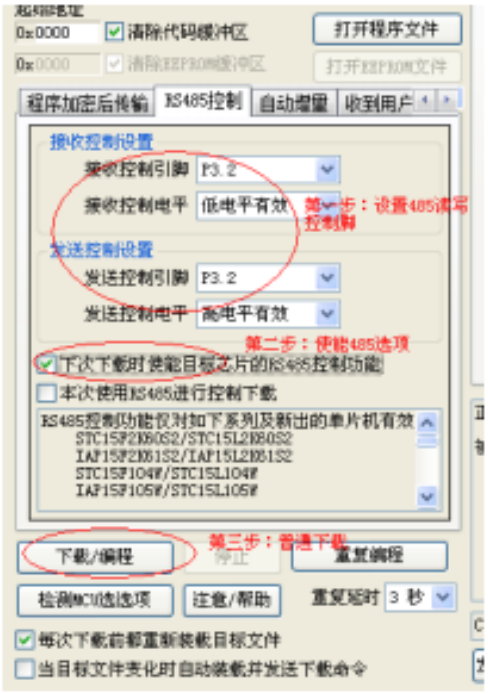
27.3.13.1 RS485 控制使用说明

由于 RS485 相比 RS232 具有抑制共模干扰、传输距离长等优点，所以许多大型的工业设备都采用 RS485 进行串口通讯。但由于 RS485 采用的是差分信号，所以在进行串口通讯时，只能采用半双工的工作方式，必须使用 1 个或 2 个 I/O 口来控制 RS485 的发送和接收状态。当需要采用 RS485 来对 STC 的新版 IC（支持 RS485 下载的单片机系列在后面会详细列出）进行 ISP 下载时，必须进行一些设置才可下载代码。

具体的操作步骤如下:

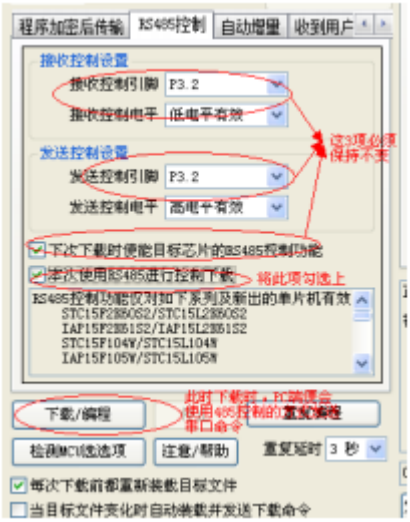
1、首先需要设置好相应的 RS485 控制端口，并勾选上“下次下载时使能目标芯片的 RS485 控制功能”这个选项

2、然后使用普通下载方式将 RS485 相关的硬件选项写入到目标芯片



3、经过前面两步的设置和编程，此时的目标芯片便具有了对 RS485 的控制功能。接下来仍需要保持 RS485 的控制选项不变，并勾选上"本次使用 RS485 进行控制下载"的选项（此选项的作用是使 PC 端也采用 RS485 的控制方式进行发送/接收串口数据）

4、再点击下载编程按钮，并对目标芯片重新上电即可实现使用 **RS485** 进行通信下载的功能



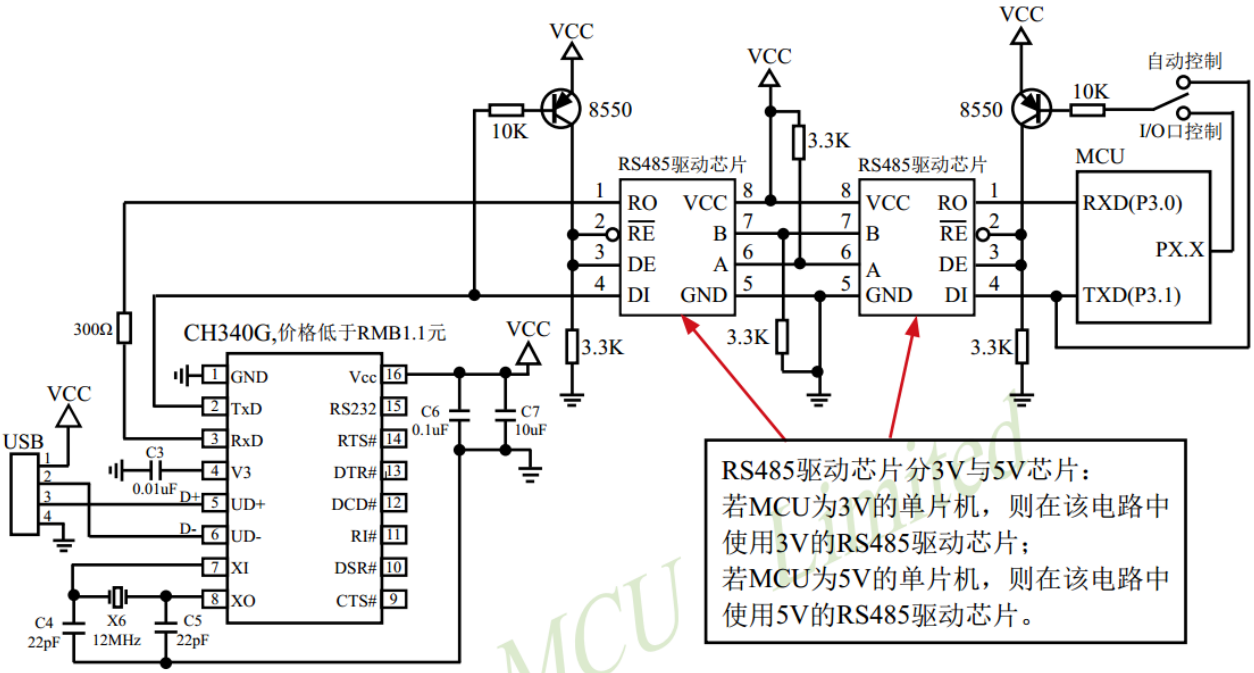
RS485 控制功能仅对如下系列及新出的单片机有效：

STC15F2K60S2/STC15L2K60S2
IAP15F2K61S2/IAP15L2K61S2
STC15F101W/STC15L101W
IAP15F105W/STC15L105W
STC15W104SW/IAP15W105W

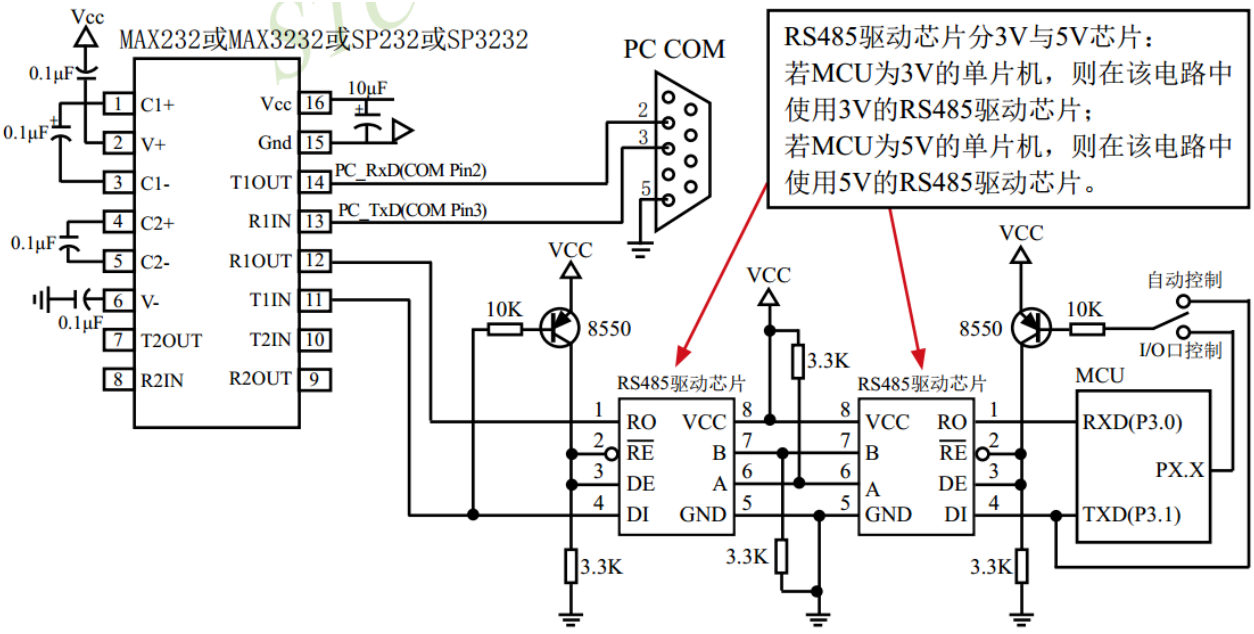
【特别注意】：若需要 RS485 控制功能，则每次都需要将 RS485 相关的配置设置正确，并勾选上“下次下载时使能目标芯片的 RS485 控制功能”这个选项，否则在下一次下载时将不具有 RS485 控制功能了

27.3.13.2 RS485 自动控制或 I/O 口控制下载线路图

1、利用 USB 转串口连接电脑的 RS485 控制下载线路图（自动控制或 I/O 口控制）



2、利用 RS232 转串口连接电脑的 RS485 控制下载线路图（自动控制或 I/O 口控制）



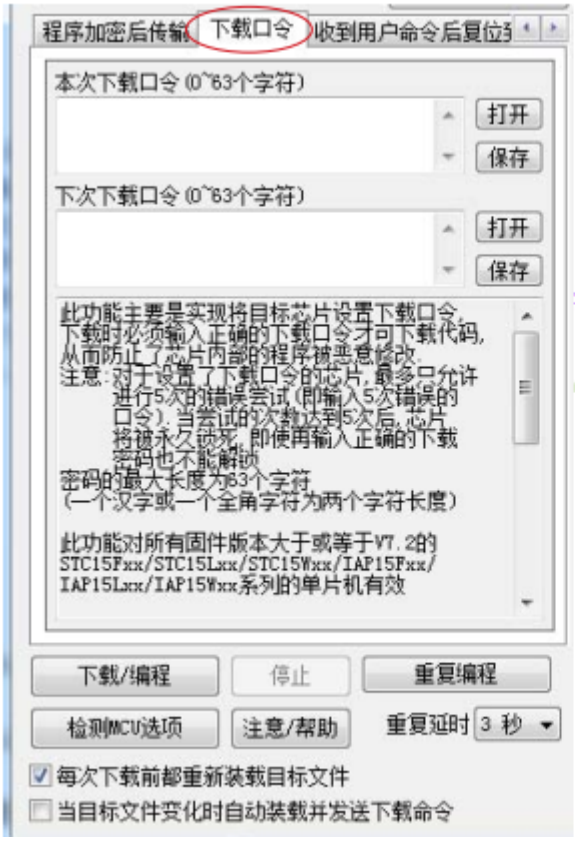
注意：如果要设置单片机某个 I/O 口控制 RS485 发送或接收命令有效，则必须将单片机焊入电路板之前先用 U8 下载工具结合电脑 ISP 软件对该单片机进行“RS485 控制”设置并烧录一下（如上节所述），否则将单片机实现不了 RS485 控制功能。

建议用户将本节所述“RS485 控制下载线路图（自动控制或 I/O 口控制）”设计到您的用户板上

27.3.14 “可设下次更新程序时需口令”功能使用说明

固件版本大于或等于 V7.2 的 STC15Fxx/STC15Lxx/STC15Wxx/IAP15Fxx/IAP15Lxx/IAP15Wxx 系列的单片机还具有“可设下次更新程序时需口令”功能。该功能主要是实现将目标芯片设置下载口令,下载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。

如用户需使用固件版本大于或等于 V7.2 的 STC15Fxx/STC15Lxx/STC15Wxx/IAP15Fxx/IAP15Lxx/IAP15Wxx 系列的单片机的“可设下次更新程序时需口令”功能，则可在 AIapp-ISP 烧录软件中的如下位置进行设置：



注意: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,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密码的最大长度为63个字符(一个汉字或一个全角字符为两个字符长度)。

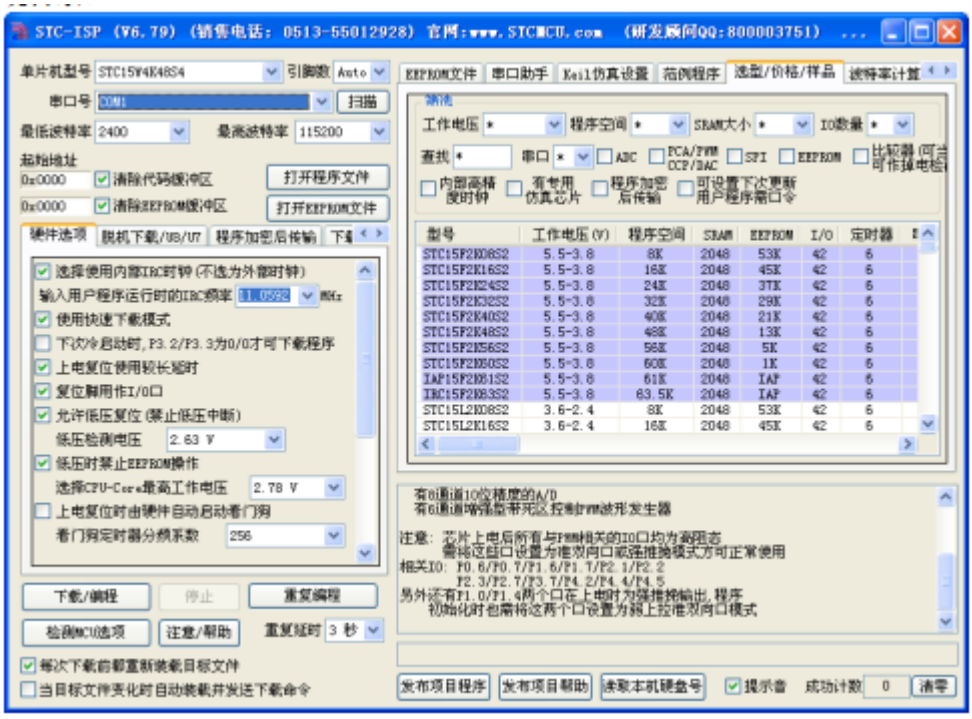
使用方法（分如下 4 种情况）：

1. 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，在下次下载口令输入框内输入初始的下载口令，然后正常下载即可
2. 对已设置下载口令的芯片进行正常下载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，然后正常下载即可
3. 对已设置下载口令的芯片进行修改下载口令在本次下载口令输入框内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，然后正常下载即可
4. 对已设置下载口令的芯片进行取消下载口令在本次下载口令输入框内输入之前设置的下载口令，在下次下载口令输入框内不输入任何内容，然后正常下载即可

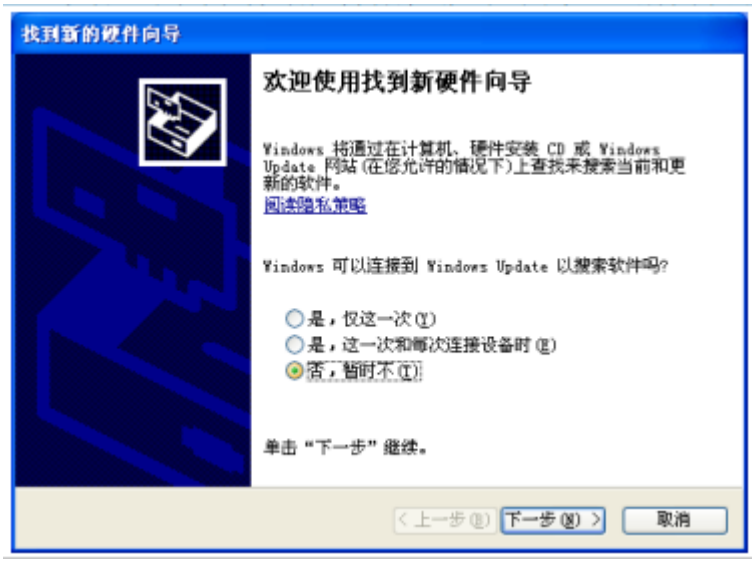
27.3.15 STC-USB 驱动程序安装说明

27.3.15.1 Windows XP 操作系统下的 STC-USB 驱动程序安装说明

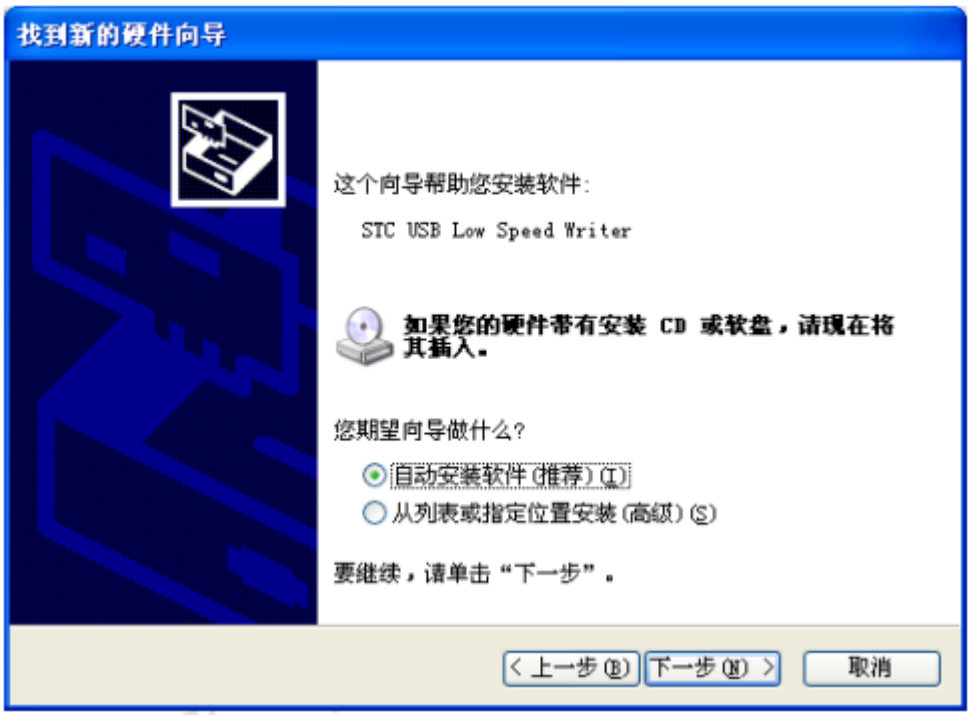
打开 V6.85 版（或者更新的版本）的 AIapp-ISP 下载软件，下载软件会自动将驱动文件复制到相关的系统目录。



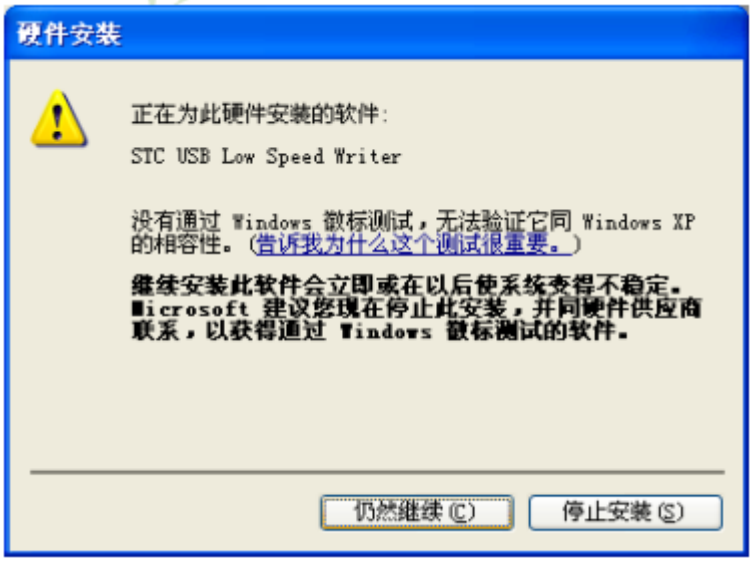
插入 USB 设备，系统找到设备后自动弹出如下对话框，选择其中的“否，暂时不”项



在下面的对话框中选择“自动安装软件（推荐）”项



在弹出的下列对话框中，选择“仍然继续”按钮



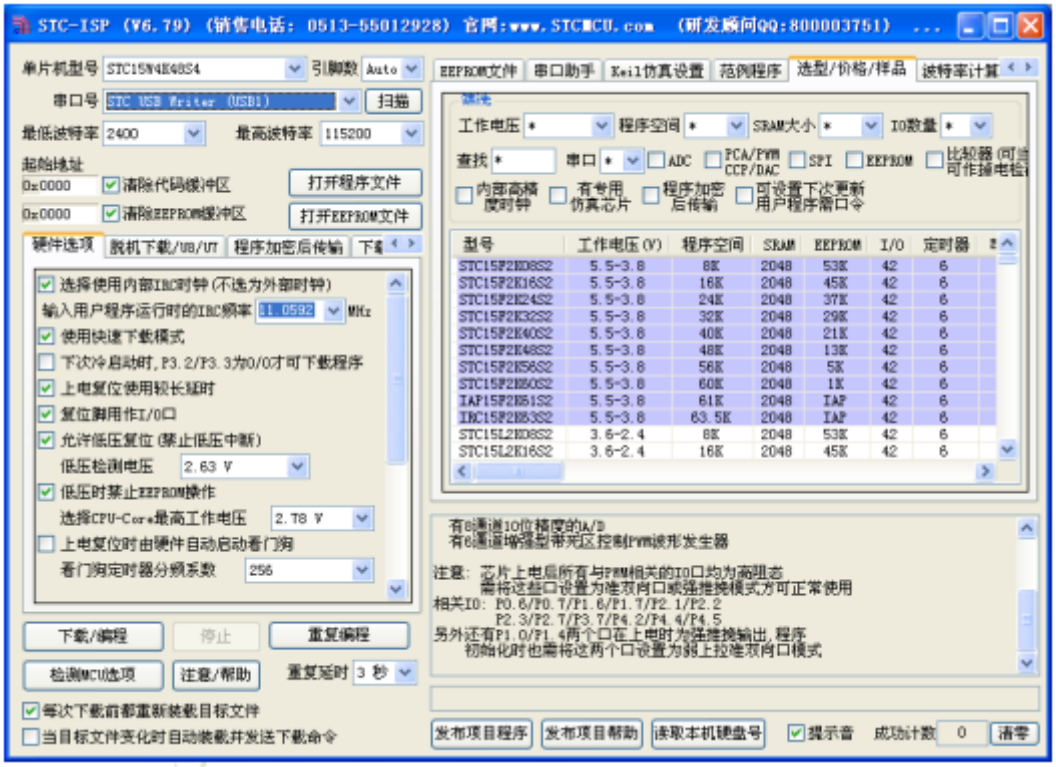
接下系统会自动安装驱动，如下图



出现下面的的对话框表示驱动安装完成

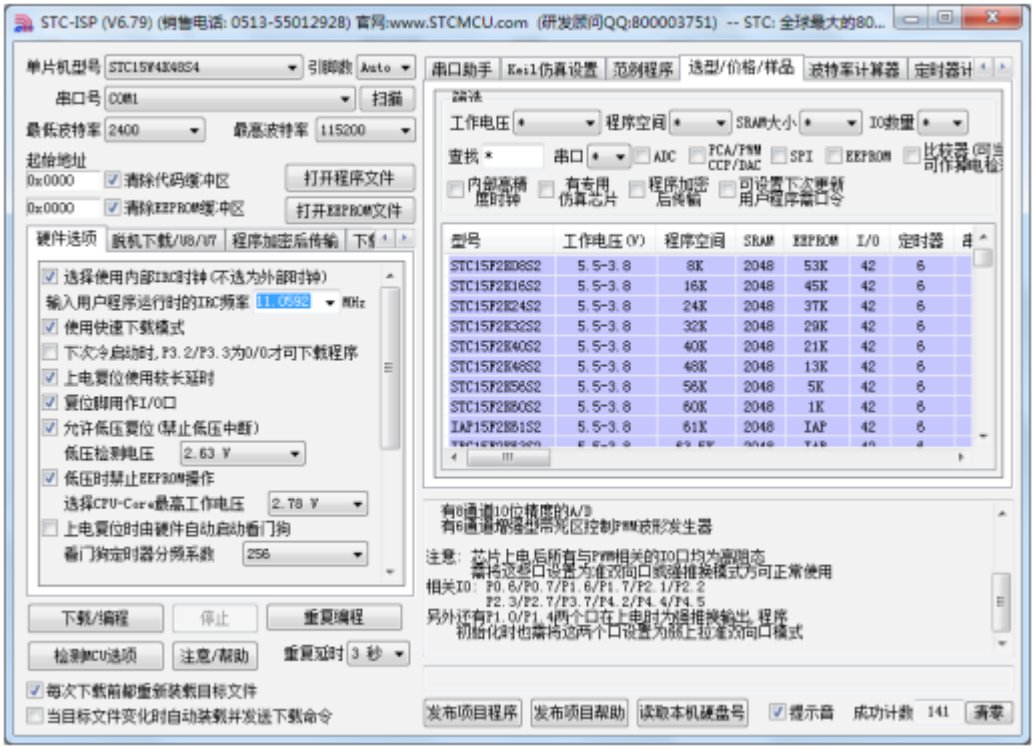


此时，之前打开的 AIapp-ISP 下载软件中的串口号列表会自动选择所插入的 USB 设备，并显示设备名称为“STC USB Writer （USB1）”，如下图：



27.3.15.2 Windows7（32 位）操作系统下的 STC-USB 驱动程序安装说明

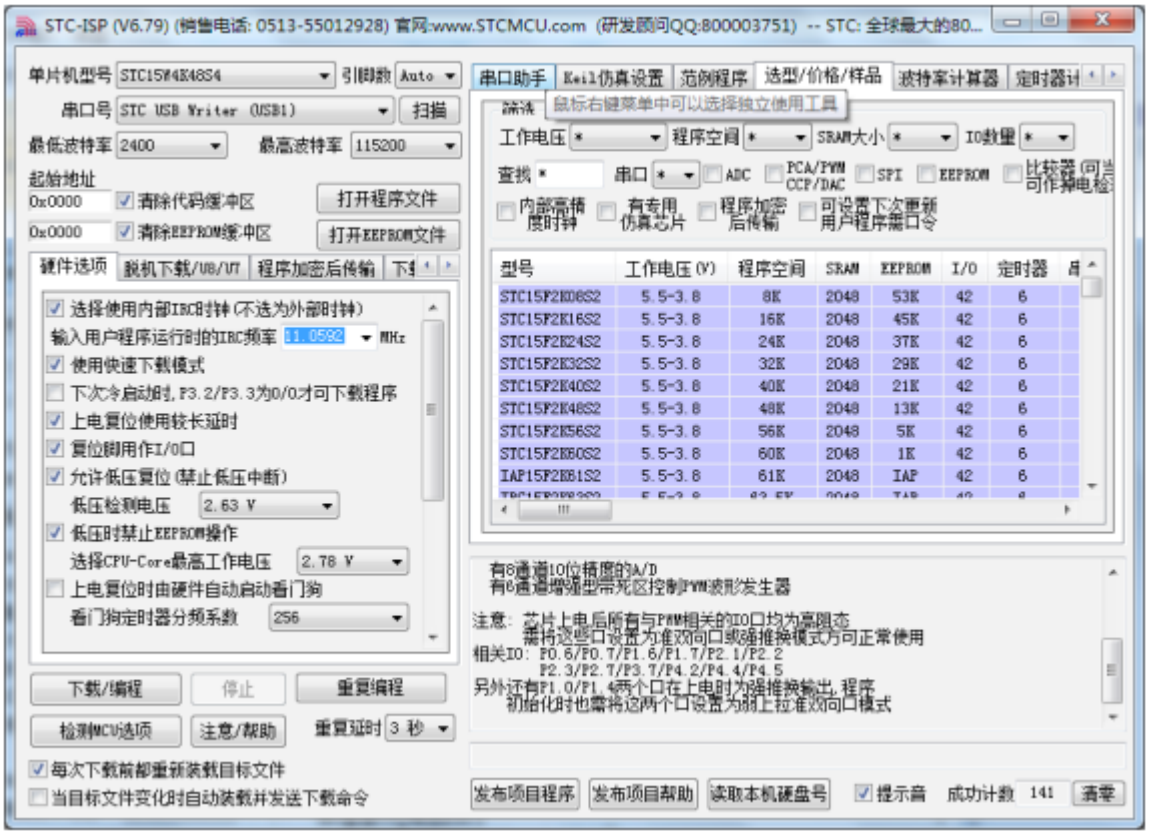
打开 V6.95G 版（或者更新的版本）的 AIapp-ISP 下载软件，下载软件会自动将驱动文件复制到相关的系统目录



插入 USB 设备，系统找到设备后会自动安装驱动。安装完成后会有如下的提示框。



此时，之前打开的 AIapp-ISP 下载软件中的串口号列表会自动选择所插入的 USB 设备，并显示设备名称为“STC USB Writer （USB1）”，如下图：



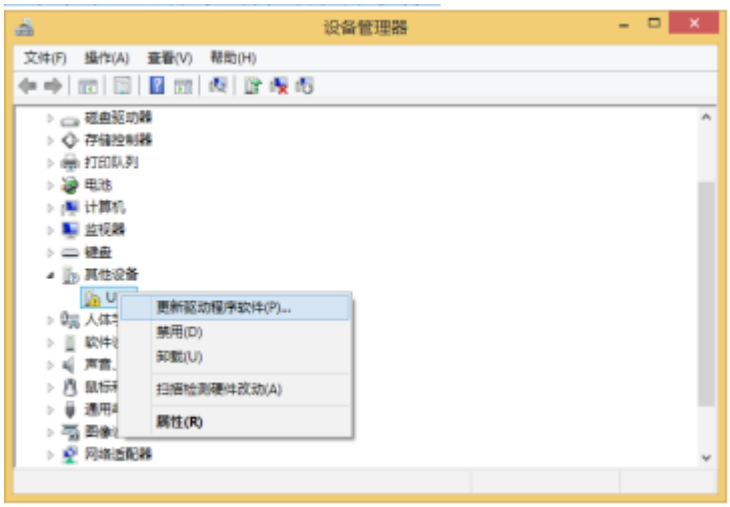
注：若 Windows 7 下，系统并没有自动安装驱动，则驱动的安装方法请参考 Windows 8（32 位）的安装方法

27.3.15.3 Windows8（32 位）操作系统下的 STC-USB 驱动程序安装说明

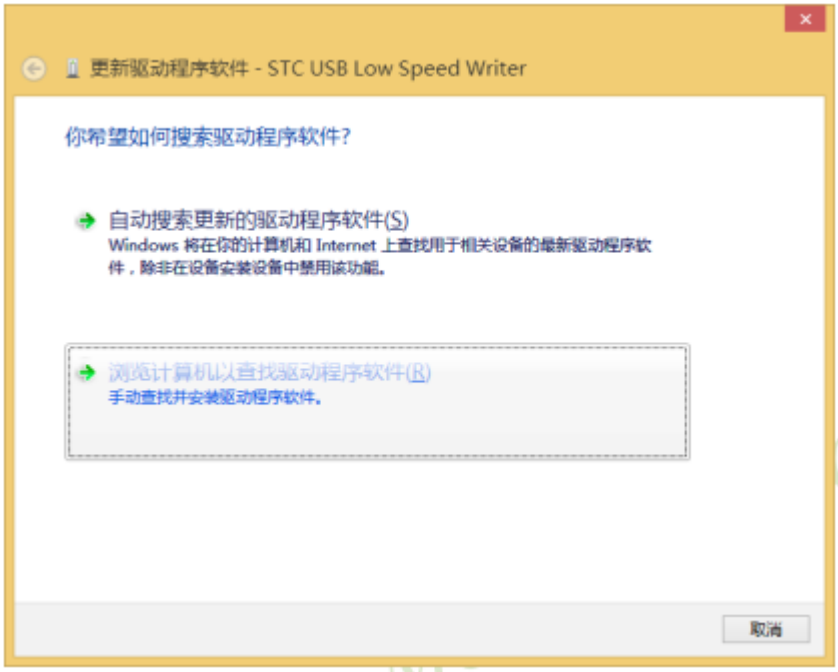
打开 V6.95G 版（或者更新的版本）的 AIapp-ISP 下载软件（由于权限的原因，在 Windows8 中下载软件不会将驱动文件复制到相关的系统目录，需要用户手动安装。首先从 STCAI 官方网站下载“stc-isp-15xx-V6.95g.zip”（或更新版本），下载后解压到本地磁盘，则 STC-USB 的驱动文件也会被解压到当前解压目录中的“STC-USB Driver”中（例如将下载的压缩文件“stc-isp15xx-V6.95g.zip”解压到“F:\”，则 STC-USB 驱动程序在“F:STC-USB Driver”目录中））



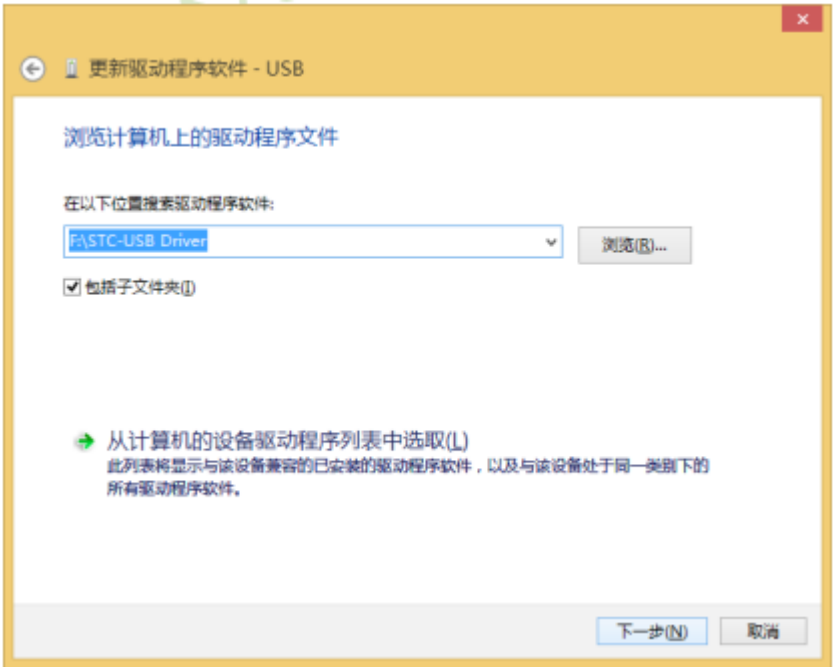
插入 USB 设备，并打开“设备管理器”。找到设备列表中带黄色感叹号的 USB 设备，在设备的右键菜单中，选择“更新驱动程序软件”



在下面的的对话框中选择“浏览计算机以查找驱动程序软件”



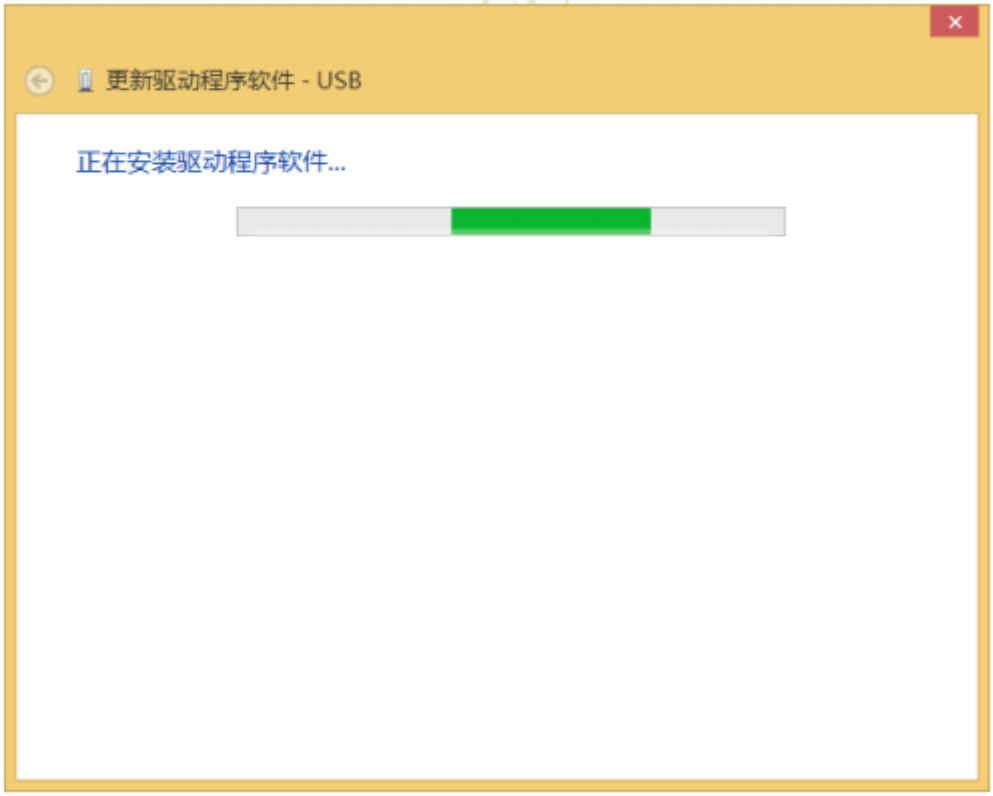
单击下面对话框中的“浏览”按钮，找到之前 STC-USB 驱动程序的存放目录（例如：之前的示例目录为“F:\STC-USB Driver”，用户将路径定位到实际的解压目录）



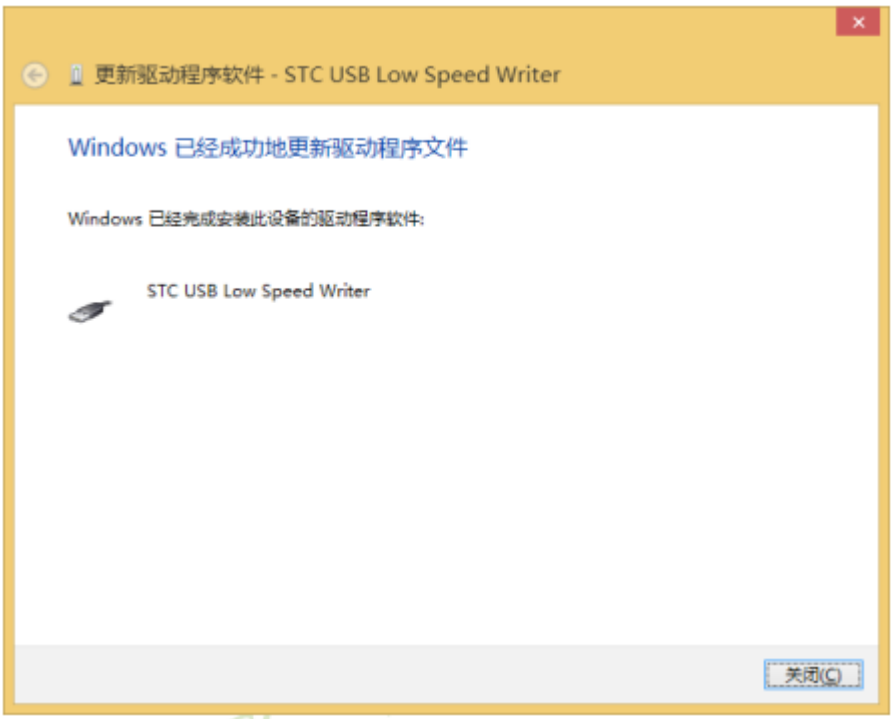
驱动程序开始安装时，会弹出如下对话框，选择“始终安装此驱动程序软件”



接下来，系统会自动安装驱动，如下图



出现下面的的对话框表示驱动安装完成



此时在设备管理器中，之前带有黄色感叹号的设备，此时会显示为“STC USB Low Speed Writer”的设备名



在之前打开的 AIapp-ISP 下载软件中的串口号列表会自动选择所插入的 USB 设备，并显示设备名称为“STC USB Writer （USB1）”，如下图：



27.3.15.4 Windows 8（64 位）操作系统下的 STC-USB 驱动程序安装说明

由于 Windows8 64 位操作系统在默认状态下，对于没有数字签名的驱动程序是不能安装成功的。所以在安装 STC-USB 驱动前，需要按照如下步骤，暂时跳过数字签名，即可顺利安装成功。

首先将鼠标移动到屏幕的右下角，选择其中的“设置”按钮



然后在设置界面中选择“更改电脑设置”项



在电脑设置中，选择“常规”属性页中“高级启动”项下面的“立即启动”按钮



在下面的界面中，选择“疑难解答”项



然后选择“疑难解答”中的“高级选项”



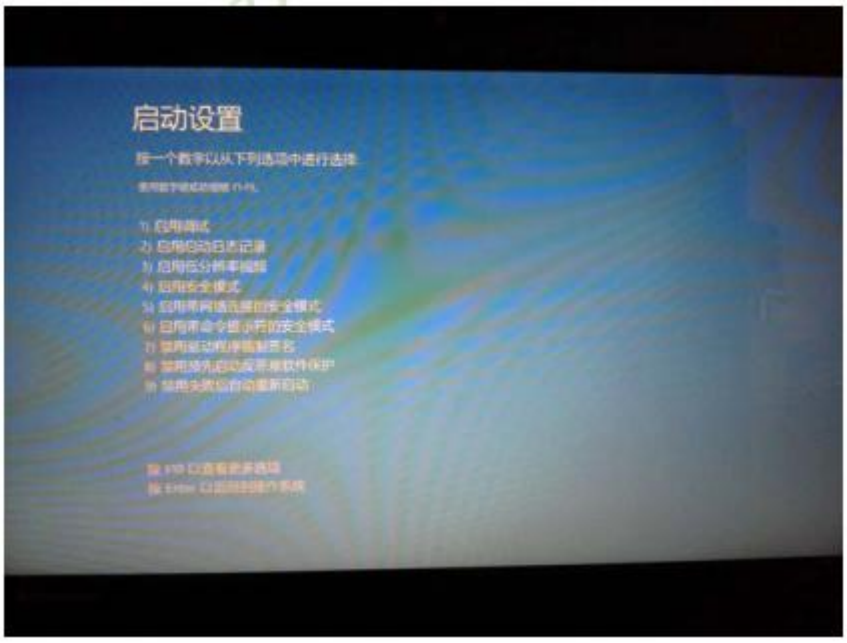
在下面的“高级选项”界面中，选择“启动设置”



在下面的“启动设置”界面中，单击“重启”按钮对电脑进行重新启动



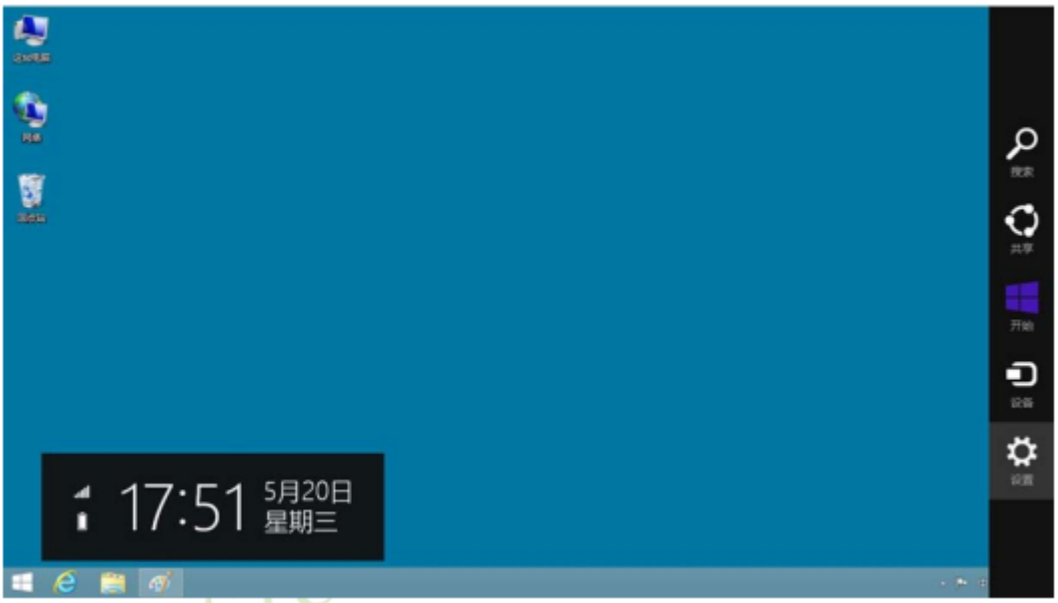
在电脑重新启动后会自动进入如下图所示的“启动设置”界面，按数字键“7”或者按功能键“F7”选择“禁用驱动程序强制签名”进行启动



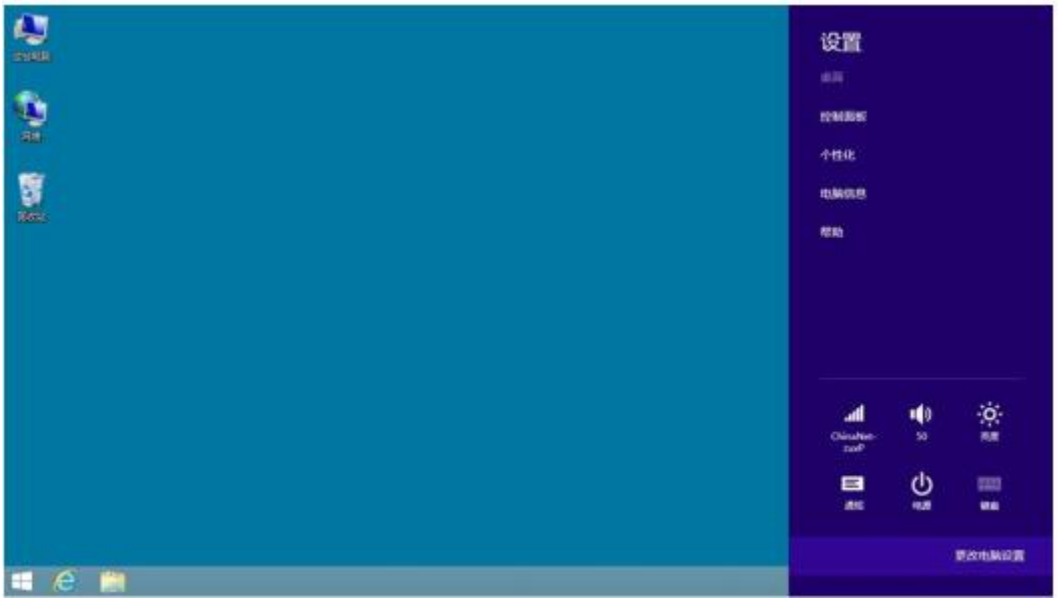
启动到 Windows 8 后，按照 Windows 8（32 位）的安装说明即可完成驱动的安装

27.3.15.5 Windows 8.1（64 位）操作系统下的 STC-USB 驱动程序安装说明

Windows 8.1 与 Windows 8 进入高级启动菜单的方法不一样，在此专门进行说明。
首先将鼠标移动到屏幕的右下角，选择其中的“设置”按钮



然后在设置界面中选择“更改电脑设置”项



在电脑设置中，选择“更新和恢复”（这里与 Windows 8 不一样，Windows 8 选择的是“常规”）



在更新和恢复页面中选择“恢复”属性页，单击“高级启动”项下面的“立即启动”按钮



接下来的操作与 Window 8 的步骤相同
在下面的界面中，选择“疑难解答”项



然后选择“疑难解答”中的“高级选项”



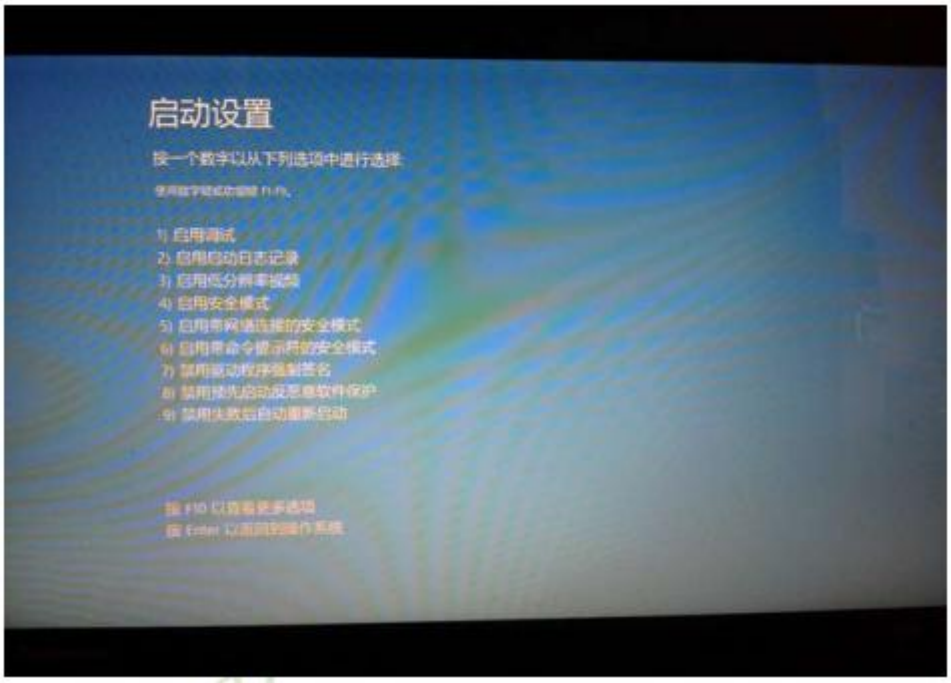
在下面的“高级选项”界面中，选择“启动设置”



在下面的“启动设置”界面中，单击“重启”按钮对电脑进行重新启动



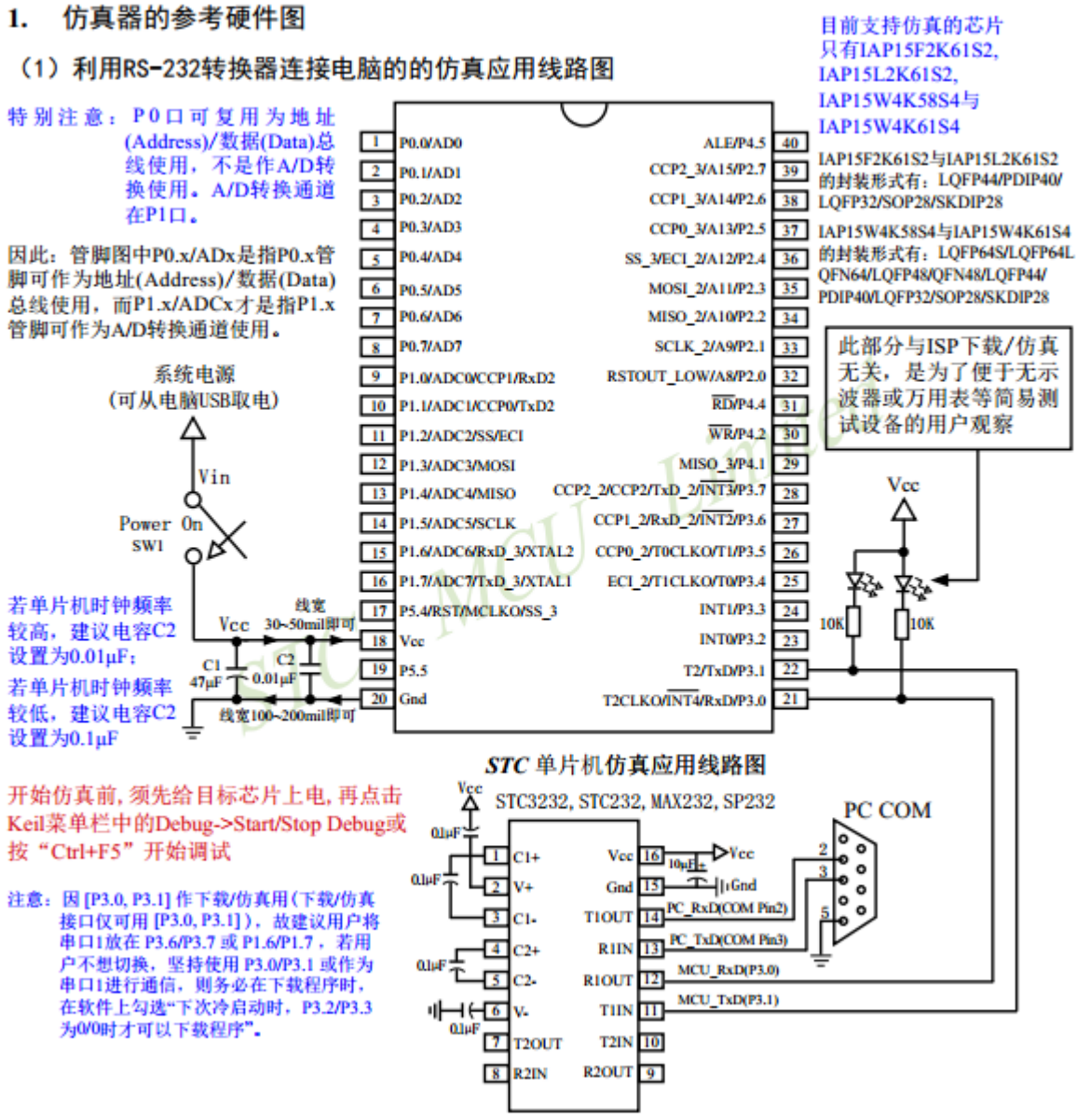
在电脑重新启动后会自动进入如下图所示的“启动设置”界面，按数字键“7”或者按功能键“F7”选择“禁用驱动程序强制签名”进行启动



启动到 Windows 8 后，按照 Windows 8（32 位）的安装方法即可完成驱动的安装

27.4 STC 仿真器说明指南（建议用户串口放在 P3.6/P3.7 或 P1.6/P1.7 上）

1.仿真器的参考硬件图



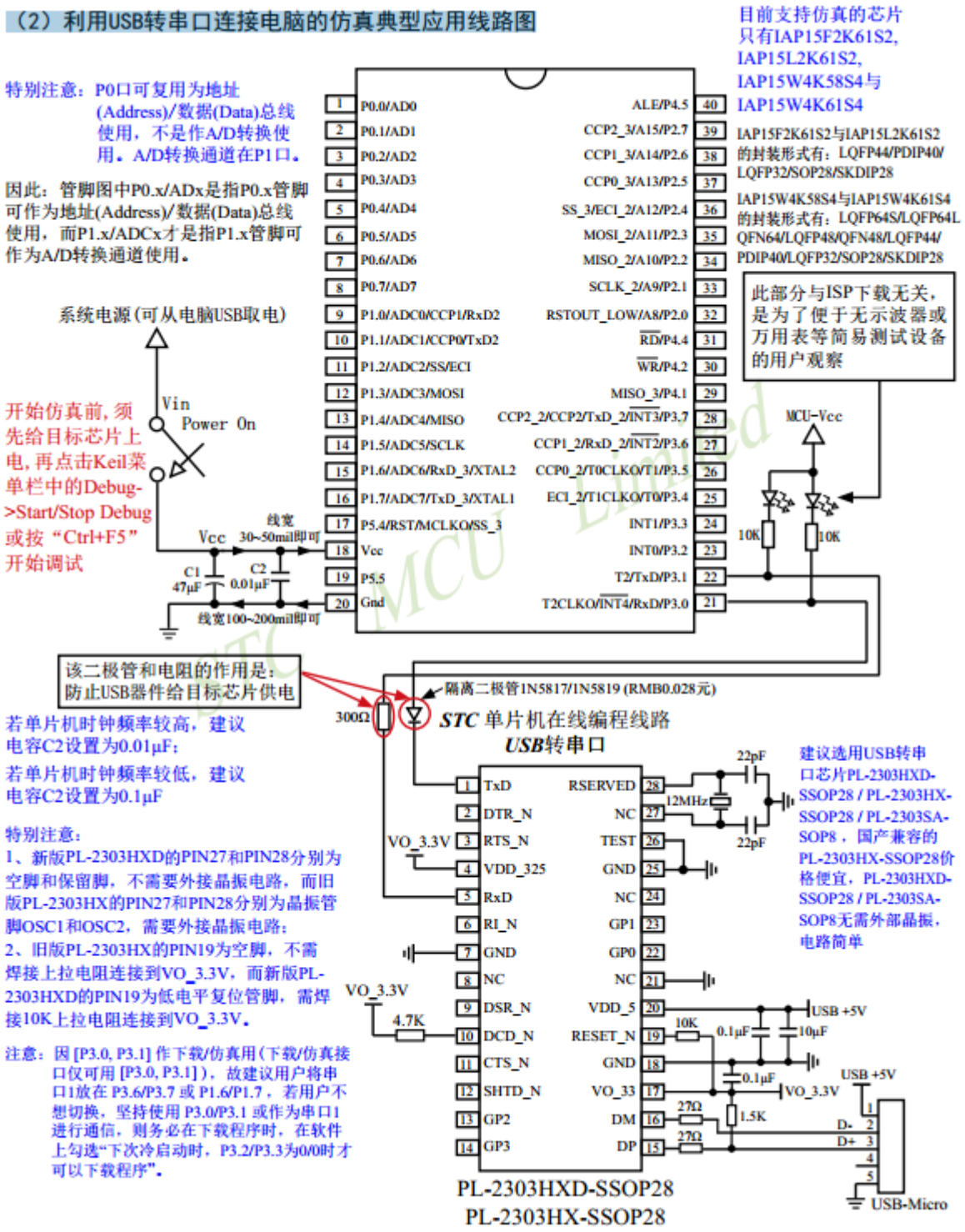
内部高可靠复位，可彻底省掉外部复位电路，当然也可以使用外部复位电路

P5.4/RST/MCLKO 脚出厂时默认为 I/O 口，可以通过 AIapp-ISP 编程器将其设置为 RST 复位脚（高电平复位）.

内部集成高精度 R/C 时钟（±0.3%），±1%温漂（-40℃ ~ +85℃），常温下温漂±0.6%（-20℃ ~ +65℃），5MHz~35MHz 宽范围可设置，可彻底省掉外部昂贵的晶振，当然也可以使用外部晶振

建议在 Vcc 和 Gnd 之间就近加上电源去耦电容 C1（47μF），C2（0.01μF），可去除电源线噪声，提高抗干扰能力

（2）利用 USB 转串口连接电脑的仿真典型应用线路图



2. 软件环境

对于汇编语言程序,复位入口的程序必须是长跳转指令，可使用如下语句

```
    ORG     0000H       ;复位入口地址
    LJMP    RESET       ;使用 LJMP 指令
      …                 ;其它中断向量
    ORG     100H        ;用户代码地址
RESET:                  ;复位入口
```

…　　;用户代码

3. 仿真代码占用的资源

程序空间：仿真代码占用程序区最后 6K 字节的空间

如果用 IAP15F2K61S2/IAP15L2K61S2 单片机仿真时，用户程序只能占 55K（0x0000~0xDBFF）空间，用户程序不要使用从 0xDC00 到 0xF3FF 的 6K 字节空间

常规 RAM（data, idata）：0 字节

XRAM（xdata）：768 字节（0x0400 – 0x06FF，用户在程序中不要使用）

I/O：P3.0 / P3.1

用户在程序中不得操作 P3.0/P3.1，不要使用 INT4/T2CLKO/P3.0，不要使用 T2/P3.1

不要使用外部中断 INT4，不要使用 T2 的时钟输出功能，不要使用 T2 的外部计数功能

对于 IAP 型号单片机，对 EEPROM 的操作是通过对多余不用的程序区进行 IAP 模拟实现的，此部分要修改程序（IAP 起始地址）。

如 IAP15F2K61S2 单片机的 EEPROM 区的位置如右图所示。



4. 仿真器操作步骤

（1）安装 Keil 版本的仿真驱动，如下图所示:



如上图，首先选择“Keil 仿真设置”页面，点击“添加 MCU 型号到 Keil 中”，在出现的如下的目录选择窗口中，定位到 Keil 的安装目录（一般可能为“C:\Keil\”），“确定”后出现下图中右边所示的提示信息，表

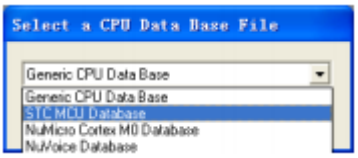
示安装成功。添加头文件的同时也会安装 STC 的 Monitor51 仿真驱动 STCMON51.DLL，驱动与头文件的的安装目录如上图所示。



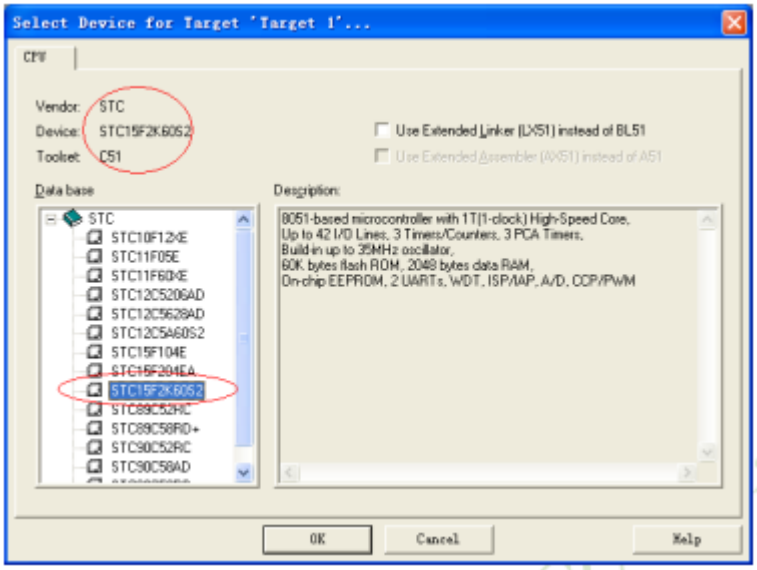


（2）在 Keil 中创建项目

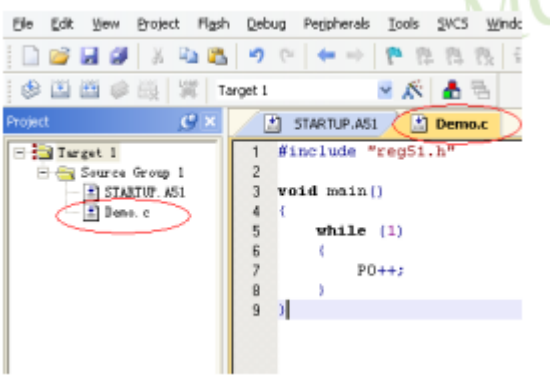
若第一步的驱动安装成功，则在 Keil 中新建项目时选择芯片型号时，便会有“STC MCU Database”的选择项，如下图



然后从列表中选择响应的 MCU 型号（目前 STC 支持仿真的型号只有 STC15F/L2K60S2，IAP15W4K58S4 和 IAP15W4K61S4），我们在此选择“STC15F2K60S2”的型号，点击“确定”完成选择



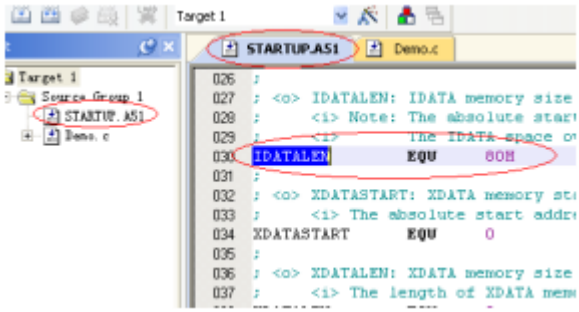
添加源代码文件到项目中，如下图：



保存项目，若编译无误，则可以进行下面的项目设置了

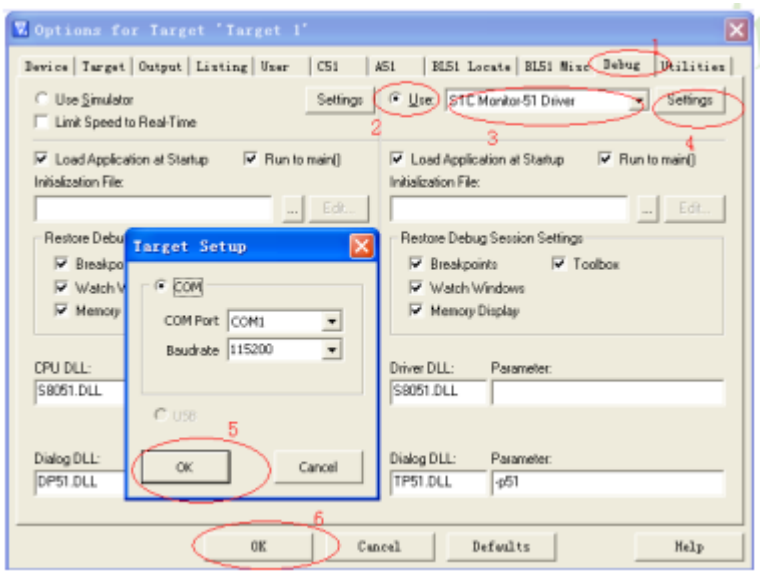
附加说明一点：

当创建的是 C 语言项目，且有将启动文件“STARTUP.A51”添加到项目中时，里面有一个命名为“IDATALEN”的宏定义，它是用来定义 IDATA 大小的一个宏，默认值是 128，即十六进制的 80H，同时它也是启动文件中需要初始化为 0 的 IDATA 的大小。所以当 IDATA 定义为 80H，那么 STARTUP.A51 里面的代码则会将 IDATA 的 00-7F 的 RAM 初始化为 0；同样若将 IDATA 定义为 0FFH，则会将 IDATA 的 00-FF 的 RAM 初始化为 0。



虽然 STC15F2K60S2 系列的单片机的 IDATA 大小为 256 字节（00-7F 的 DATA 和 80H-FFH 的 IDATA），但由于 STC15F2K60S2 在 RAM 的最后 17 个字节有写入 ID 号以及相关的测试参数，若用户在程序中需要使用这一部分数据，则一定不要将 IDATALEN 定义为 256。

（3）项目设置，选择 STC 仿真驱动



如上图，首先进入到项目的设置页面，选择“Debug”设置页，第 2 步选择右侧的硬件仿真“Use…”，第 3 步，在仿真驱动下拉列表中选择“STC Monitor-51 Driver”项，然后点击“Settings”按钮，进入下面的设置画面，对串口的端口号和波特率进行设置，波特率一般选择 115200 或者 57600。到此设置便完成了。

（4）创建仿真芯片



准备一颗 IAP15F2K61S2 或者 IAP15L2K61S2 的芯片，并通过下载板连接到电脑的串口，然后如上图，选择正确的芯片型号，然后进入到“Keil 仿真设置”页面，点击“将 IAP15F2K61S2 设置为 2.0 版仿真芯片”按钮或者“将 IAP15L2K61S2 设置为 2.0 版仿真芯片”按钮，当程序下载完成后仿真器便制作完成了。

（5）开始仿真

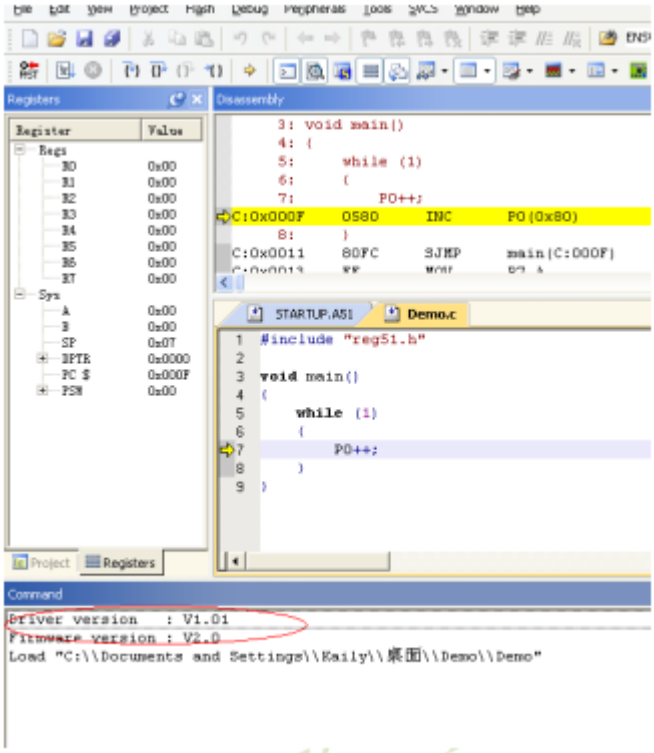
将制作完成的仿真芯片通过串口与电脑相连接，并给目标芯片上电。

将前面我们所创建的项目编译至没有错误后，按“Ctrl+F5”开始调试。

若硬件连接无误的话，将会进入到类似于下面的调试界面，并在命令输出窗口显示当前的仿真驱动版本号和当前仿真监控代码固件的版本号

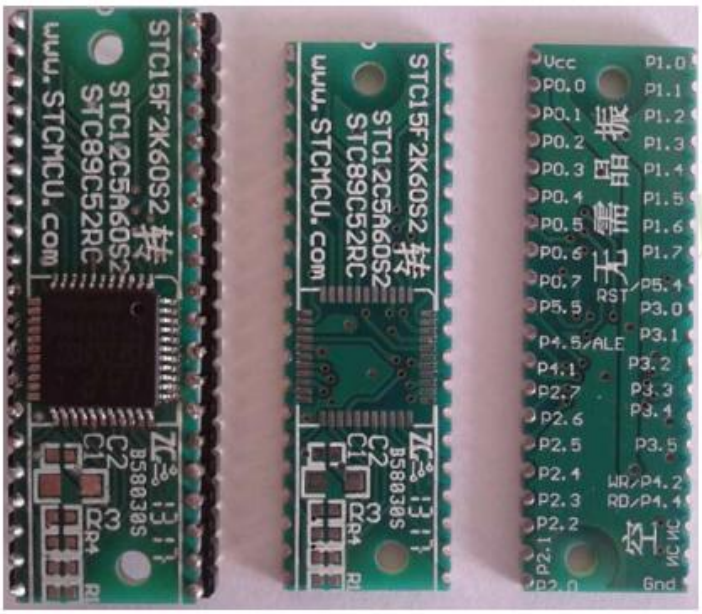
断点设置的个数目前最大允许 20 个（理论上可设置任意个，但是断点设置得过多会影响调试的速度）。

注意：开始仿真前，须先给目标芯片上电，再点击 Keil 菜单栏中的 Debug->Start/Stop Debug 或按“Ctrl+F5”开始调试


Registers
Register
Value
Regs
R0 0x00
R1 0x00
R2 0x00
R3 0x00
R4 0x00
R5 0x00
R6 0x00
R7 0x00
Sys
A 0x00
B 0x00
SP 0x07
DPTR 0x0000
PC $ 0x000F
PSW 0x00
Project
Registers
Disassembly
3: void main()
4: {
5: while (1)
6: {
7: P0++;
C:0x000F 0580 INC P0(0x80)
8: }
C:0x0011 80FC SJMP main(C:000F)
STARTUP.A51
Demo.c
1 #include "reg51.h"
2
3 void main()
4 {
5 while (1)
6 {
7 P0++;
8 }
9 }
Command
Driver version : V1.01
Firmware version : V2.0
Load "C:\\Documents and Settings\\Eaily\\桌面\\Demo\\Demo"

27.5 如何让传统的 8051 单片机学习板可仿真

传统的 8051 单片机学习板不具有仿真功能，让传统的 8051 单片机学习板可仿真需要借助转换板，转换板的实物图如下图所示，转换后的引脚排布与传统 8051 的脚位基本一致，从而可以实现标准 8051 学习板的仿真功能。



完整转换板　　完整转正面　　完整转反面

该转换板可进行：

IAP15F2K61S2/STC15F2K60S2 转 STC89C52RC/STC89C58RD+系列仿真用、

IAP15F2K61S2/STC15F2K60S2 转 STC90C52RC/STC90C58RD+系列仿真用、

IAP15F2K61S2/ STC15F2K60S2 转 STC10F08XE/STC11F60XE 系列仿真用、

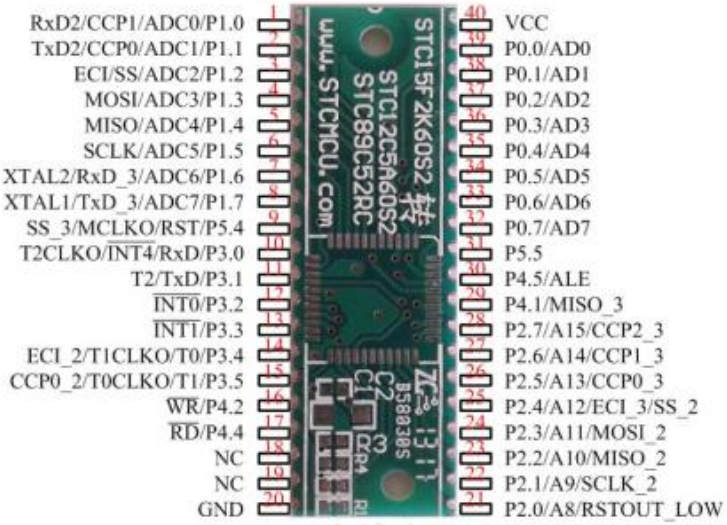
及 IAP15F2K61S2/STC15F2K60S2 转 STC12C5A60S2 系列仿真用。

目前，我公司只是小批量生产此转换板，供客户快速验证用，如需要我们提供样板，售价为：空板：1 元人民币；

转换板+主控芯片（IAP15F2K61S2/STC15F2K60S2）：6 元人民币。

若用户自己批量生产此板，成本价可控制在 0.40 元以下。新产品开发请直接使用 STC15F2K60S2/IAP15F2K61S2 系列来开发

下图为转换板功能示意图（IAP15F2K61S2 转 STC89C52/90C52/12C5A60S2 仿真用转换板）



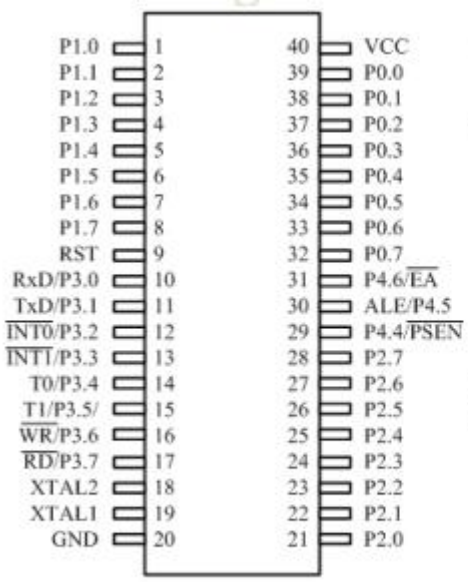
注意：

由于内置高精准 R/C 时钟（5MHz ~ 40MHz 可设），因此不需要外部晶振；

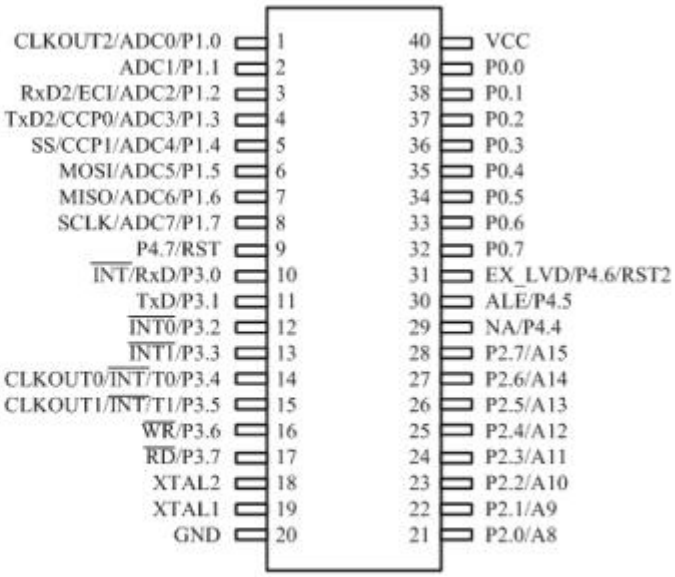
XTAL1 和 XTAL2 是空的

WR 和 RD 是（WR/P4.2 和 RD/P4.4），而不是传统的（WR/P3.6 和 RD/P3.7）

下面为 STC89C52RC 和 STC12C5A60S2 的脚位分布图



STC89C52RC-PDIP40



STC12C5A60S2-PDIP40

27.6 若无仿真器，如何调试/开发用户程序

STC 单片机部分系列无仿真器，如 STC89xx 系列、STC90xx 系列等，但长沙菊阳微电子有限公司以及南京伟福实业有限公司均有通用的 STC89xx、STC90xx 系列单片机仿真器购买。当然部分 STC 单片机也有自己的仿真器，如最新的 STC15 系列。

现介绍在没有仿真器的情况下如何调试和开发用户程序，具体操作步骤如下：

1.首先参照本手册当中的“用定时器 1 做波特率发生器”，调通串口程序，这样，要观察变量就可以自己写一小段测试程序将变量通过串口输出到电脑端的 AIapp-ISP.EXE 的“串口调试助手”来显示，也很方便。

2.调通按键扫描程序（到处都有大量的参考程序）

3.调通用户系统的显示电路程序，此时变量/寄存器也可以通过用户系统的显示电路显示了

4.调通 A/D 检测电路（我们用户手册里面有完整的参考程序）

5.调通 PWM 等电路（我们用户手册里面有完整的参考程序）

这样分步骤模块化调试用户程序，有些系统，熟练的 8051 用户，三天就可以调通了，难度不大的系统，一般一到二周就可以调通。

用户的串口输出显示程序可以在输出变量/寄存器的值之后，继续全速运行用户程序，也可以等待串口送来的“继续运行命令”，方可继续运行用户程序，这就相当于断点。这种断点每设置一个地方，就必须调用一次该显示寄存器/变量的程序，有点麻烦，但却很实用。

28 利用主控芯片对从芯片(限 STC15 系列)进行 ISP 下载

STC15 系列单片机的用户程序，除可以通过专用的下载工具进行在线联机或离线脱机 ISP 下载外，还可以利用其他的 MCU（如单片机，ARM，DSP 等）作主控芯片对其进行 ISP 下载。

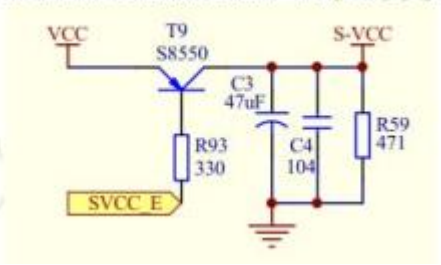
利用其他 MCU（如单片机，ARM，DSP 等）对 STC15 系列单片机进行串口 ISP 下载的系统中，其他 MCU 为主控芯片，STC15 系列单片机为受控的从芯片，主控芯片通过串口控制从芯片进行 ISP 下载。在利用其他 MCU 对 STC15 系列单片机进行 ISP 下载时，

- STC15 系列单片机（即从芯片）先停电，
- 然后，其他 MCU（即主控芯片）发送下载命令给 STC15 系列单片机（即从芯片），
- 最后，其他 MCU 控制 STC15 系列单片机再上电

这样才能正确地对 STC15 系列单片机进行 ISP 下载。

由于在利用其他 MCU（如单片机，ARM，DSP 等）对 STC15 系列单片机进行 ISP 下载过程中，其他 MICU 作为主控芯片需要控制从芯片（即 STC15 系列单片机）电路的电源开关，因此，在进行电路连接时，用户可通过主控芯片的一个 I/O 口控制从芯片电路的电源开关。STC15 系列单片机电路的电源控制部分参考电路图如下:

STC15系列单片机电路的电源控制部分参考电路图



用户可将主控芯片的一个 I/O 口连接到上图中的 SVCC_E 管脚，这样通过主控芯片的该 I/O 口即可控制 STC15 系列单片机电路的电源开关，那么，在利用其他 MCU 对 STC15 系列单片机进行 ISF 下载时，其他 MCU（主控芯片）就可自由控制 STC15 系列单片机（从芯片）停电或上电了。利用其他 MCU（如单片机，ARM，DSP 等）对 STC15 系列单片机进行串口 ISP 下载示例程序（C 语言语言程序）如下:

```
/*----使用主控芯片对从芯片（限 STC15 系列）进行 ISP 下载举例----*/
/本示例在 Keil 开发环境下请选择 Intel 的 8058 芯片型号进行编译
//假定测试芯片的工作频率为 11.0592MHz
//注意:使用本代码对 STC15 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/4/(n))    //主控芯片串口波特率计算公式
#define     T1MS            (65536 - FOSC/1000)     //主控芯片 1ms 定时初值

//#define   FUSER           11059200L           //STC15 系列目标芯片工作频率
//#define   FUSER           12000000L           //STC15 系列目标芯片工作频率
//#define   FUSER           18432000L           //STC15 系列目标芯片工作频率
//#define   FUSER           22118400L           //STC15 系列目标芯片工作频率
#define     FUSER           24000000L           //STC15 系列目标芯片工作频率
#define     RL(n)           (65536 - FUSER/4/(n))    //STC15 系列目标芯片串口波特率计算公式

//SFR 定义
sfr         AUXR = 0x8e;

//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)
{
    while (1)
    {
        Initial();
        if (Download(DEMO, 0x0100))
```

```
        {
            //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;
        }
    }
}

//1ms 定时器中断服务程序
void tm0(void) interrupt 1 using 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 using 1
{
    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 STC15 系列的芯片进行数据下载程序
BOOL Download(BYTE *pdat, long size)
{
    BYTE arg;
    BYTE fwver;
    BYTE offset;
    BYTE cnt;
    WORD addr;

    //握手
    CommInit();
    while (1)
    {
        if (UartRecvStep == 0)
        {
            UartSend(0x7f);
            DelayXms(10);
        }
        if (UartReceived)
        {
            arg = RxBuffer[4];
            fwver = RxBuffer[17];
            if (RxBuffer[0] == 0x50) break;
            return FALSE;
        }
    }

    //设置参数(设置从芯片使用最高的波特率以及擦除等待时间等参数)
    TxBuffer[0] = 0x01;
    TxBuffer[1] = arg;
    TxBuffer[2] = 0x40;
    TxBuffer[3] = HIBYTE(RL(MAXBAUD));
    TxBuffer[4] = LOBYTE(RL(MAXBAUD));
    TxBuffer[5] = 0x00;
    TxBuffer[6] = 0x00;
    TxBuffer[7] = 0xc3;
    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;
if (fwver < 0x72)
{
    CommSend(1);
}
else
{
    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;
if (fwver < 0x72)
{
    CommSend(2);
}
else
{
    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;
    if (fwver < 0x72)
    {
        offset = 3;
    }
    else
    {
        TxBuffer[3] = 0x5a;
        TxBuffer[4] = 0xa5;
        offset = 5;
    }
    while (addr < size)
    {
        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;
    }
```

```
////写硬件选项
////如果不需要修改硬件选项,此步骤可直接跳过,此时所有的硬件选项
////都维持不变,MCU 的频率为上一次所调节频率
////若写硬件选项,MCU 的内部 IRC 频率将被固定写为 24M,
////建议:第一次使用 AIapp-ISP 下载软件将从芯片的硬件选项设置好
////  以后再使用主芯片对从芯片下载程序时不写硬件选项
//DelayXms(10);
//for (cnt=0; cnt<128; cnt++)
//{
//    TxBuffer[cnt] = 0xff;
//}
//TxBuffer[0] = 0x04;
//TxBuffer[1] = 0x00;
//TxBuffer[2] = 0x00;
//if (fwver < 0x72)
//{
//    TxBuffer[34] = 0xfd;
//    TxBuffer[62] = arg;
//    TxBuffer[63] = 0x7f;
//    TxBuffer[64] = 0xf7;
//    TxBuffer[65] = 0x7b;
//    TxBuffer[66] = 0x1f;
//    CommSend(67);
//}
//else
//{
//    TxBuffer[3] = 0x5a;
//    TxBuffer[4] = 0xa5;
//    TxBuffer[36] = 0xfd;
//    TxBuffer[64] = arg;
//    TxBuffer[65] = 0x7f;
//    TxBuffer[66] = 0xf7;
//    TxBuffer[67] = 0x7b;
//    TxBuffer[68] = 0x1f;
//    CommSend(69);
//}
//while (1)
//{
//    if (TimeOut == 0) return FALSE;
//    if (UartReceived)
//    {
//        if ((RxBuffer[0] == 0x04) && (RxBuffer[1] == 'T')) break;
//        return FALSE;
//    }
//}
```

```
    //下载完成
    return TRUE;
}

char code DEMO[256] =
{
    0x02,   0x00,   0x5E,   0x12,   0x00,   0x4B,   0x75,   0xB0,
    0xEF,   0x12,   0x00,   0x2C,   0x75,   0xB0,   0xDF,   0x12,
    0x00,   0x2C,   0x75,   0xB0,   0xFE,   0x12,   0x00,   0x2C,
    0x75,   0xB0,   0xFD,   0x12,   0x00,   0x2C,   0x75,   0xB0,
    0xFB,   0x12,   0x00,   0x2C,   0x75,   0xB0,   0xF7,   0x12,
    0x00,   0x2C,   0x80,   0xDA,   0xE4,   0xFF,   0xFE,   0xE4,
    0xFD,   0xFC,   0x0D,   0xBD,   0x00,   0x01,   0x0C,   0xBC,
    0x01,   0xF8,   0xBD,   0xF4,   0xF5,   0x0F,   0xBF,   0x00,
    0x01,   0x0E,   0xBE,   0x03,   0xEA,   0xBF,   0xE8,   0xE7,
    0x02,   0x00,   0x4B,   0x75,   0x80,   0xFF,   0x75,   0x90,
    0xFF,   0x75,   0xA0,   0xFF,   0x75,   0xB0,   0xFF,   0x75,
    0xC0,   0xFF,   0x75,   0xC8,   0xFF,   0x22,   0x78,   0x7F,
    0xE4,   0xF6,   0xD8,   0xFD,   0x75,   0x81,   0x07,   0x02,
    0x00,   0x03,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
    0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
    0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
    0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
    0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
    0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
    0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
    0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
    0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
    0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
    0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
    0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
    0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
    0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
    0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
    0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
    0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
};
```

附录A STC15 系列单片机电气特性

Absolute Maximum Ratings

Parameter	Symbol	Min	Max	Unit
Srotage temperature	TST	-55	+125	℃
Operating temperature (I)	TA	-40	+85	℃
Operating temperature (C)	TA	0	+70	℃
DC power supply (5V)	VDD - VSS	-0.3	+5.5	V
DC power supply (3V)	VDD - VSS	-0.3	+3.6	V
Voltage on any pin	-	-0.3	VCC + 0.3	V

DC Specification (5V MCU)

Sym	Parameter	Specification				Test Condition
		Min.	Typ	Max.	Unit	
VDD	Operating Voltage	3.3	5.0	5.5	V	
IPD	Power Down Current	-	< 0.1	-	uA	5V
IIDL	Idle Current	-	3.0	-	mA	5V
ICC	Operating Current	-	4	20	mA	5V
VIL1	Input Low (P0,P1,P2,P3)	-	-	0.8	V	5V
VIH1	Input High (P0,P1,P2,P3)	2.0	-	-	V	5V
VIH2	Input High (RESET)	2.2	-	-	V	5V
IOL1	Sink Current for output low (P0,P1,P2,P3)	-	20	-	mA	5V@Vpin=0.45V
IOH1	Sourcing Current for output high (P0,P1,P2,P3)(Quasi-output)	200	270	-	uA	5V
IOH2	Sourcing Current for output high (P0,P1,P2,P3)(Push-Pull, Strong-output)	-	20	-	mA	5V@Vpin=2.4V
IIL	Logic 0 input current (P0,P1,P2,P3)	-	-	50	uA	Vpin=0V
ITL	Logic 1 to 0 transition current (P0,P1,P2,P3)	100	270	600	uA	Vpin=2.0V

DC Specification (3V MCU)

Sym	Parameter	Specification				Test Condition
		Min.	Typ	Max.	Unit	
VDD	Operating Voltage	2.4	3.3	3.6	V	
IPD	Power Down Current	-	<0.1	-	uA	3.3V
IIDL	Idle Current	-	2.0	-	mA	3.3V
ICC	Operating Current	-	4	10	mA	3.3V
VIL1	Input Low (P0,P1,P2,P3)	-	-	0.8	V	3.3V
VIH1	Input High (P0,P1,P2,P3)	2.0	-	-	V	3.3V
VIH2	Input High (RESET)	2.2	-	-	V	3.3V
IOL1	Sink Current for output low (P0,P1,P2,P3)	-	20	-	mA	3.3V@Vpin=0.45V
IOH1	Sourcing Current for output high (P0,P1,P2,P3)(Quasi-output)	140	170	-	uA	3.3V
IOH2	Sourcing Current for output high (P0,P1,P2,P3)(Push-Pull)	-	20	-	mA	3.3V
IIL	Logic 0 input current (P0,P1,P2,P3)	-	8	50	uA	Vpin=0V
ITL	Logic 1 to 0 transition current (P0,P1,P2,P3)	-	110	600	uA	Vpin=2.0V

附录B STC15 系列单片机取代传统 8051 注意事项

STC15 系列单片机的定时器 0/定时器 1 与传统 8051 完全兼容，上电复位后，定时器部分缺省还是除 12 再计数的，所以定时器完全兼容。

STC15 系列单片机对传统 8051 的 111 条指令执行速度全面提速，最快的指令快 24 倍，最慢的指令快 3 倍。靠软件延时实现精确延时的程序需要调整。

其它需注意的细节:

普通 I/O 口既作为输入又作为输出:

传统 8051 单片机执行 I/O 口操作，由高变低或由低变高，以及读外部状态都是 12 个时钟，而现在 STC15 系列单片机执行相应的操作是 4 个时钟。

- 传统 8051 单片机如果对外输出为低，直接读外部状态是读不对的，必须先将 I/O 口智高才能够读对，而传统 8051 单片机由低变高的指令是 12 个时钟，该指令执行完成后，该 I/O 口也确实已变高。故可以紧跟着由低变高的指令后面，直接执行读该 I/O 口状态指令。
- 而 STC15 系列单片机由于执行由低变高的指令是 4 个时钟，太快了，相应的指令执行完以后，I/O 口还没有变高，要再过一个时钟之后，该 I/O 口才可以变高，故建议此状况下增加 2 个空操作延时指令再读外部口的状态。

I/O 口驱动能力:

- 最新 STC15 系列单片机 I/O 口的灌电流是 20mA，驱动能力超强，驱动大电流时，不容易烧坏。
- 传统 STC89Cxx 系列单片机 I/O 口的灌电流是 6mA，驱动能力不够强，不能驱动大电流，建议使用 STC15 系列。

看门狗:

最新 STC15 系列单片机的看门狗寄存器 WDT_CONTR 的地址在 C1H，增加了看门狗复位标志位

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset value
WDT_CONTR	C1h	Watch-Dog-Timer Control register	WDT_FLAG	-	EN_WDT	CLR_WDT	IDLE_WDT	PS2	PS1	PS0	xx00,0000

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset value
WDT_CONTR	E1h	Watch-Dog-Timer Control register	-	-	EN_WDT	CLR_WDT	IDLE_WDT	PS2	PS1	PS0	xx00,0000

- 最新 STC15 系列单片机的看门狗在 ISP 烧录程序可设置上电复位后直接启动看门狗
- 而传统 STC89 系列单片机无此功能，故最新 STC15 系列单片机看门狗更可靠。

与 EEPROM 操作相关的寄存器：

STC15Fxx 单片机 ISP/IAP 控制寄存器地址和 STC89xx 系列单片机 ISP/IAP 控制寄存器地址不同如下：											
Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
STC15Fxx 系列 IAP_DATA STC89xx 系列 ISP_DATA	C2h E2h	ISP/IAP Flash Data Register									1111,1111
STC15Fxx 系列 IAP_ADDRH STC89xx 系列 ISP_ADDRH	C3h E3h	ISP/IAP Flash Address High									0000,0000
STC15Fxx 系列 IAP_ADDRL STC89xx 系列 ISP_ADDRL	C4h E4h	ISP/IAP Flash Address Low									0000,0000
STC15Fxx 系列 IAP_CMD STC89xx 系列 ISP_CMD	C5h E5h	ISP/IAP Flash Command Register	-	-	-	-	-	-	MS1	MS0	xxxx,xx00
STC15Fxx 系列 IAP_TRIG STC89xx 系列 ISP_TRIG	C6h E6h	ISP/IAP Flash Command Trigger									xxxx,xxxx
STC15Fxx 系列 IAP_CONTR STC89xx 系列 ISP_CONTR	C7h E7h	ISP/IAP Control Register	IAPEN	SWBS	SWRST	CMD_FAIL	-	WT2	WT1	WT0	0000,x000

ISP/IAP_TRIG 寄存器有效启动 IAP 操作，需顺序送入的数据不一样：

- STC15 系列单片机的 ISP/IAP 命令要生效，要对 IAP_TRIG 寄存器按顺序先送 5Ah，再送 A5h 方可
- STC89xx 系列单片机的 ISP/IAP 命令要生效，要对 IAP_TRIG 寄存器按顺序先送 46h，再送 B9h 方可

EEPROM 起始地址不一样：

- STC15 系列单片机的 EEPROM 起始地址全部从 0000h 开始，每个扇区 512 字节
- STC89xx 系列单片机的 EEPROM 起始地址分别有从 1000h/2000h/4000h/8000h 开始的，程序兼容性不够好。

外部中断：

- 最新 STC15 系列单片机有 5 个外部中断。其中外部中断 0（INT0）和外部中断 1（INT1）可配置为 2 种中断触发方式：
  - ✓ 第一种方式，仅下降沿触发中断，与传统 8051 的外部中断 0 和 1 的下降沿中断兼容。
  - ✓ 第二种方式，上升沿中断和下降沿中断同时支持。
- 另外相对传统 STC89 系列单片机，最新的 STC15 系列单片机还增加了外部中断 2、外部中断 3 和外部中断 4，这三个新增的外部中断都只能下降沿触发中断。
- 而传统 STC89 系列单片机的外部中断 0 和外部中断 1 只可以配置为下降沿中断或低电平中断。

定时器：

- 最新 STC15 系列单片机的定时器/计数器 0 和定时器/计数器 1 与传统 STC89 系列单片机的定时器/计数器 0 和定时器/计数器 1 的最大不同在于定时器的工作模式 0。
  - ✓ 最新 STC15 系列单片机的定时器/计数器 0 和定时器/计数器 1 的工作模式 0 是 16 位自动重

装载模式

- ✓ 而传统 STC89 系列单片机的定时器/计数器 0 和定时器/计数器 1 的模式 0 是 13 位定时/计数器模式。

- ➢ 最新 STC15 系列单片机的定时器/计数器 0 和定时器/计数器 1 仍保留着其他 3 种工作模式，这 3 中工作模式与传统的 STC89 系列单片机的定时器/计数器 0 和定时器/计数器 1 的工作模式兼容。
- ➢ 另外传统的 STC89 系列单片机还设有定时器 2，而最新 STC15 系列单片机只有定时器 0 和 1

外部时钟和内部时钟：

- ➢ 最新 STC15 系列单片机内部集成了高精度 R/C 振荡器作为系统时钟，省掉了昂贵的外部晶体振荡时钟。
- ➢ 而传统 STC89 系列单片机只能使用外部晶体或时钟作为系统时钟。

功耗：

- ➢ 功耗由 2 部分组成，晶体振荡器放大电路的功耗和单片机的数字电路功耗组成晶体振荡器放大电路的功耗。最新 STC15 系列单片机比 STC89xx 系列低。
- ➢ 单片机的数字电路功耗：时钟频率越高，功耗越大。
  - ✓ 最新 STC15 系列单片机在相同工作频率下，指令执行速度比传统 STC89 系列单片机快 3-24 倍，故可用较低的时钟频率工作，这样功耗更低。
  - ✓ 而且 STC15 系列单片机可以利用内部的时钟分频器对时钟进行分频，以较低的频率工作，使得单片机的功耗更低。

掉电唤醒：

- ➢ 最新 STC15 系列单片机支持外部中断上升沿或下降沿均可唤醒，也可仅下降沿唤醒。另外最新 STC15 系列单片机还内置了掉电唤专用定时器。
- ➢ 传统 STC89 系列单片机是只支持外部中断低电平唤醒。

附录C 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D 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）

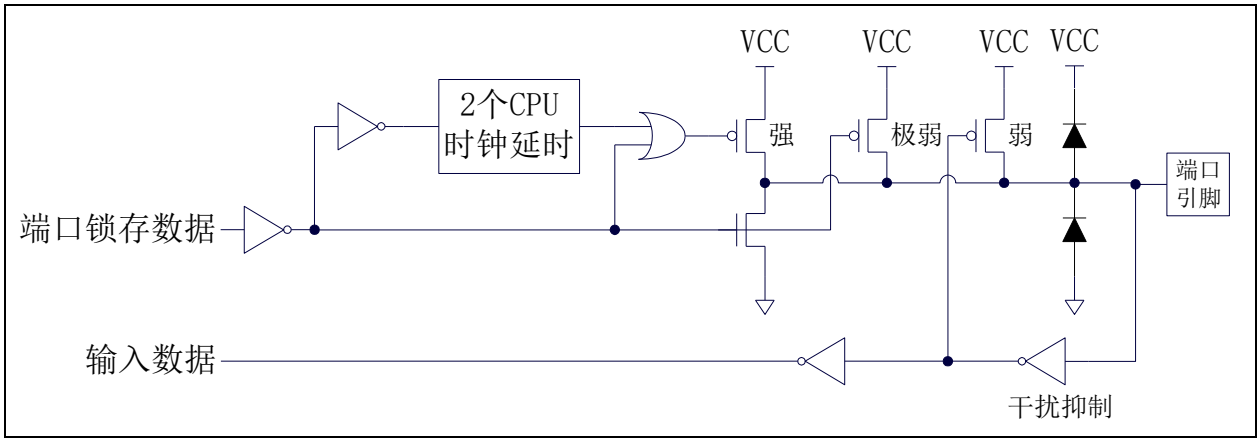
首先将万用表调到二极管检测档，被测芯片不要供电，将万用表的红表笔连接到被测芯片的 **GND 管脚**，黑表笔依次测量每个 I/O 口，如果万用表显示的参数为 0.45～0.75V 左右，则表示芯片的内部 I/O 到 GND 的保护二极管正常，即打线也是完好的，若显示的参数为开路/断开的状态，则表示芯片内部的打线已断路。

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E 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F 内部常规 256 字节 RAM 间接寻址测试程序

```
;/*----STC15 系列单片机内部常规 RAM 间接寻址测试程序----*/
;/*----本演示程序在 STC15 系列 ISP 的下载编程工具上测试通过----*/
;/*----如果要在程序中使用该程序,请在程序中注明使用了 STC 的资料及程序----*/
;/*----如果要在文章中引用该程序,请在文章中注明使用了 STC 的资料及程序----*/
TEST_CONST      EQU 5AH
;TEST_RAM        EQU 03H

        ORG     0000H
        LJMP    INITIAL
        ORG     0050H

INITIAL:
        MOV     R0, #253
        MOV     R1, #3H

TEST_ALL_RAM:
        MOV     R2, #0FFH

TEST_ONE_RAM:
        MOV     A, R2
        MOV     @R1, A
        CLR     A
        MOV     A, @R1
        CJNE    A, 2H, ERROR_DISPLAY
        DJNZ    R2, TEST_ONE_RAM
        INC     R1
        DJNZ    R0, TEST_ALL_RAM

OK_DISPLAY:
        MOV     P1, #11111110B

Wait1:
        SJMP    Wait1

ERROR_DISPLAY:
        MOV     A, R1
        MOV     P1, A

Wait2:
        SJMP    Wait2
END
```

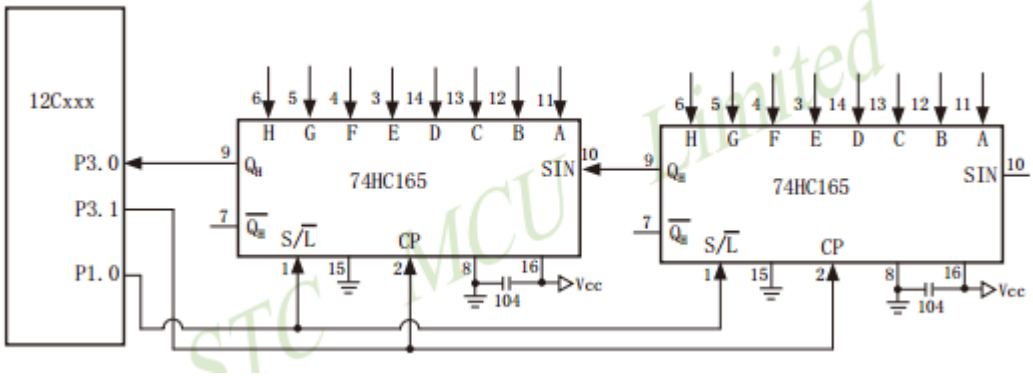
附录G 用串口扩展 I/O 接口

STC15 系列单片机串行口的方式 0 可用于 I/O 扩展。如果在应用系统中，串行口未被占用，那么将它用来扩展并行 I/O 口是一种经济、实用的方法。

在操作方式 0 时，串行口作同步移位寄存器，其波特率是固定的，为 SYSclk/12（SYSclk 为系统时钟频率）。数据由 RXD 端（P3.0）出入，同步移位时钟由 TXD 端（P3.1）输出。发送、接收的是 8 位数据，低位在先。

一、用 74HC165 扩展并行输入口

下图是利用两片 74HC165 扩展二个 8 位并行输入口的接口电路图。



74HC165 是 8 位并行置入移位寄存器。当移位/置入端（S/L）由高到低跳变时，并行输入端的数据置入寄存器；当 S/L=1，且时钟禁止端（第 15 脚）为低电平时，允许时钟输入，这时在时钟脉冲的作用下，数据将由 Q_A 到 Q_H 方向移位。

上图中：

- TXD（P3.1）作为移位脉冲输出端与所有 74HC165 的移位脉冲输入端 CP 相连；
- RXD（P3.0）作为串行输入端与 74HC165 的串行输出端 Q_H 相连；
- P1.0 用来控制 74HC165 的移位与置入而同 S/L 相连；
- 74HC165 的时钟禁止端（15 脚）接地，表示允许时钟输入。

当扩展多个 8 位输入口时，两芯片的首尾（Q_H 与 S_{IN}）相连。

下面的程序是从 16 位扩展口读入 5 组数据（每组二个字节），并把它们转存到内部 RAM 20H 开始的单元中。

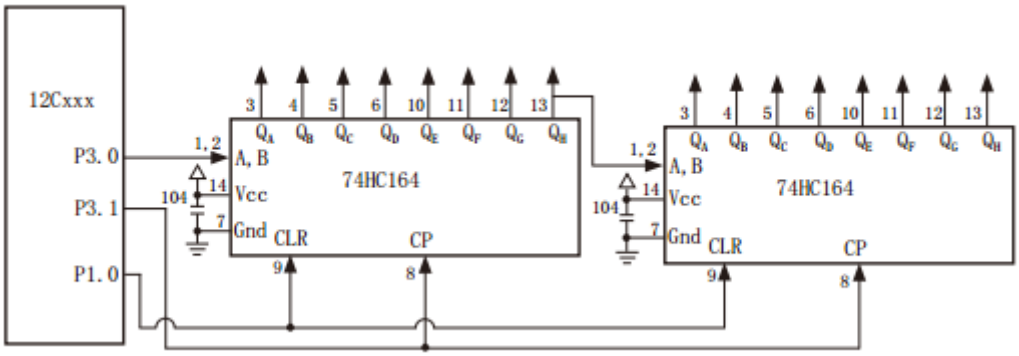
```
    MOV     R7, #05H            ;设置读入组数
    MOV     R0, #20H            ;设置内部 RAM 数据区首址
START:
    CLR     P1.0                ;并行置入数据, S/L=0
    SETB    P1.0                ;允许串行移位 S/L=1
    MOV     R1, #02H            ;设置每组字节数, 即外扩 74LS165 的个数
RXDATA:
    MOV     SCON, #00010000B    ;设串行方式 0, 允许接收, 启动接收过程
WAIT:
    JNB     RI, WAIT            ; 未接收完一帧, 循环等待
```

```
    CLR     RI              ;清 RI 标志, 准备下次接收
    MOV     A, SBUF         ; 读入数据
    MOV     @R0, A          ;送至 RAM 缓冲区
    INC     R0              ;指向下一个地址
    DJNZ    R1, RXDATA      ;为读完一组数据, 继续
    DJNZ    R7, START       ;5 组数据读完重新并行置入
            ……              ;对数据进行处理
```

上面的程序对串行接收过程采用的是查询等待的控制方式，如有必要，也可改用中断方式。从理论上讲，按上图方法扩展的输入口几乎是无限的，但扩展的越多，口的操作速度也就越慢。

二、用 74HC164 扩展并行输出口

74HC164 是 8 位串入并出移位寄存器。下图是利用 74HC164 扩展二个 8 位输出口的接口电路。



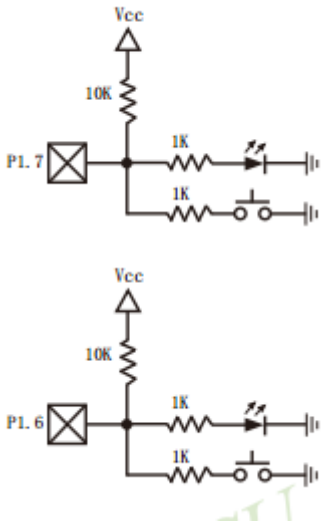
当单片机串行口工作在方式 0 的发送状态时，串行数据由 P3.0（RXD）送出，移位时钟由 P3.1（TXD）送出。在移位时钟的作用下，串行口发送缓冲器的数据一位一位地移入 74HC164 中。需要指出的是，由于 74HC164 无并行输出控制端，因而在串行输入过程中，其输出端的状态会不断变化，故在某些应用场合，在 74HC164 的输出端应加接输出三态门控制，以便保证串行输入结束后再输出数据。

下面是将 RAM 缓冲区 30H、31H 的内容串行口由 74HC164 并行输出的子程序。

```
START:
    MOV     R7,  #02H       ;设置要发送的字节个数
    MOV     R0,  #30H       ;设置地址指针
    MOV     SCON, #00H      ;设置串行口方式 0
SEND:
    MOV     A, @R0
    MOV     SBUF, A         ;启动串行口发送过程
WAIT:
    JNB     TI, WAIT        ;一帧数据未发送完, 循等待
    CLR     TI
    INC     R0              ;取下一个数
    DJNZ    R7, SEND

    RET
```

附录H　一个 I/O 口驱动发光二极管并扫描按键



利用 STC15 系列单片机的 I/O 口可被设置成弱上拉（准双向口），强上拉（推挽）输出，高阻输入（电流既不能流入也不能流出），开漏等四种工作模式的特性，可以将 STC15 系列单片机的 I/O 口同时作为发光二极管驱动及按键检测用，这样可以大幅节省 I/O 口。

当驱动发光二极管时，将该 I/O 口设置成强推挽输出，输出高即可点亮发光二极管当检测按键时，将该 I/O 口设置成弱上拉输入，再读外部口的状态，即可检测按键。

附录I STC15 系列对指令系统的提升

----与普通 8051 指令代码完全兼容，但执行的时间效率大幅提升

----其中 INC DPTR 和 MUL AB 指令的执行速度大幅提升 24 倍

----共有 12 条指令，一个时钟就可以执行完成，平均速度快 8～12 倍

如果按功能分类，STC15 系列单片机指令系统可分为:

- 算术操作类指令；
- 逻辑操作类指令；
- 数据传送类指令；
- 布尔变量操作类指令；
- 控制转移类指令。

按功能分类的指令系统表如下表所示。

指令执行速度效率提升总结(新 15 系列):

指令系统共包括 111 条指令，其中:

- 执行速度快 24 倍的 共 2 条
- 执行速度快 12 倍的 共 28 条
- 执行速度快 8 倍的 共 19 条
- 执行速度快 6 倍的 共 40 条
- 执行速度快 4.8 倍的 共 8 条
- 执行速度快 4 倍的 共 14 条

根据对指令的使用频率分析统计，STC15 系列 1T 的 8051 单片机比普通的 8051 单片机在同样的工作频率下运行速度提升了 8～12 倍。

指令执行时钟数统计(供参考)(新 15 系列):

指令系统共包括 111 条指令，其中:

- 1 个时钟就可执行完成的指令共 22 条
- 2 个时钟就可执行完成的指令共 37 条
- 3 个时钟就可执行完成的指令共 31 条
- 4 个时钟就可执行完成的指令共 12 条
- 5 个时钟就可执行完成的指令共 8 条
- 6 个时钟就可执行完成的指令共 1 条

111 条指令全部执行完一遍所需的时钟为 283 个时钟。

现 STC15 系列单片机采用 STC-Y5 超高速 CPU 内核，在相同的时钟频率下，速度又比 STC 早期的 1T 系列单片机(如 STC12 系列/STC11 系列/STC10 系列)的速度快 20%。

算术操作类指令

助记符	功能说明	字节数	传统 8051 单片机所需时钟	STC15 系列单片机所需时钟 (采用 STC-Y5 超高速 1T 8051 内核)	效率提升
ADD A, Rn	寄存器内容加到累加器	1	12	1	12 倍
ADD A, direct	直接地址单元中的数据加到累加器	2	12	2	6 倍
ADD A, @Ri	间接 RAM 中的数据加到累加器	1	12	2	6 倍
ADD A, #data	立即数加到累加器	2	12	2	6 倍
ADDC A, Rn	寄存器带进位加到累加器	1	12	1	12 倍
ADDC A, direct	直接地址单元的内容带进位加到累加器	2	12	2	6 倍
ADDC A, @Ri	间接 RAM 内容带进位加到累加器	1	12	2	6 倍
ADDC A, #data	立即数带进位加到累加器	2	12	2	6 倍
SUBB A, Rn	累加器带借位减寄存器内容	1	12	1	6 倍
SUBB A, direct	累加器带借位减直接地址单元的内容	2	12	2	6 倍
SUBB A, @Ri	累加器带借位减间接 RAM 中的内容	1	12	2	6 倍
SUBB A, #data	累加器带借位减立即数	2	12	2	6 倍
INC A	累加器加 1	1	12	1	12 倍
INC Rn	寄存器加 1	1	12	2	6 倍
INC direct	直接地址单元加 1	2	12	3	4 倍
INC @Ri	间接 RAM 单元加 1	1	12	3	4 倍
DEC A	累加器减 1	1	12	1	12 倍
DEC Rn	寄存器减 1	1	12	2	6 倍
DEC direct	直接地址单元减 1	2	12	3	4 倍
DEC @Ri	间接 RAM 单元减 1	1	12	3	4 倍
INC DPTR	地址寄存器 DPTR 加 1	1	24	1	24 倍
MUL AB	A 乘以 B	1	48	2	24 倍
DIV AB	A 除以 B	1	48	6	8 倍
DA A	累加器十进制调整	1	12	3	4 倍

逻辑操作类指令

助记符	功能说明	字节数	传统 8051 单片机所需时钟	STC15 系列单片机所需时钟(采用 STC-Y5 超高速 1T 8051 内核)	效率提升
ANL A, Rn	累加器与寄存器相“与”	1	12	1	12 倍
ANL A, direct	累加器与直接地址单元相“与”	2	12	2	6 倍
ANL A, @Ri	累加器与间接 RAM 单元相“与”	1	12	2	6 倍
ANL A, #data	累加器与立即数相“与”	2	12	2	6 倍
ANL direct, A	直接地址单元与累加器相“与”	2	12	3	4 倍
ANL direct, #data	直接地址单元与立即数相“与”	3	24	3	8 倍
ORL A, Rn	累加器与寄存器相“或”	1	12	1	12 倍
ORL A, direct	累加器与直接地址单元相“或”	2	12	2	6 倍
ORL A, @Ri	累加器与间接 RAM 单元相“或”	1	12	2	6 倍
ORL A, # data	累加器与立即数相“或”	2	12	2	6 倍
ORL direct, A	直接地址单元与累加器相“或”	2	12	3	4 倍
ORL direct, #data	直接地址单元与立即数相“或”	3	24	3	8 倍
XRL A, Rn	累加器与寄存器相“异或”	1	12	1	12 倍
XRL A, direct	累加器与直接地址单元相“异或”	2	12	2	6 倍
XRL A, @Ri	累加器与间接 RAM 单元相“异或”	1	12	2	6 倍
XRL A, # data	累加器与立即数相“异或”	2	12	2	6 倍
XRL direct, A	直接地址单元与累加器相“异或”	2	12	3	4 倍
XRL direct, #data	直接地址单元与立即数相“异或”	3	24	3	8 倍
CLR A	累加器清“0”	1	12	1	12 倍
CPL A	累加器求反	1	12	1	12 倍
RL A	累加器循环左移	1	12	1	12 倍
RLC A	累加器带进位位循环左移	1	12	1	12 倍
RR A	累加器循环右移	1	12	1	12 倍
RRC A	累加器带进位位循环右移	1	12	1	12 倍
SWAP A	累加器内高低半字节交换	1	12	1	12 倍

数据传送类指令

助记符	功能说明	字节数	传统 8051 单片机所需时钟	STC15 系列单片机所需时钟 (采用 STC-Y5 超高速 1T 8051 内核)	效率提升
MOV A, Rn	寄存器内容送入累加器	1	12	1	12 倍
MOV A, direct	直接地址单元中的数据送入累加器	2	12	2	6 倍
MOV A, @Ri	间接 RAM 中的数据送入累加器	1	12	2	6 倍
MOV A, #data	立即数送入累加器	2	12	2	6 倍
MOV Rn, A	累加器内容送入寄存器	1	12	1	12 倍
MOV Rn, direct	直接地址单元中的数据送入寄存器	2	24	3	8 倍
MOV Rn, #data	立即数送入寄存器	2	12	2	6 倍
MOV direct, A	累加器内容送入直接地址单元	2	12	2	6 倍
MOV direct, Rn	寄存器内容送入直接地址单元	2	24	2	12 倍
MOV direct, direct	直接地址单元中的数据送入另一个直接地址单元	3	24	3	8 倍
MOV direct, @Ri	间接 RAM 中的数据送入直接地址单元	2	24	3	8 倍
MOV direct, #data	立即数送入直接地址单元	3	24	3	8 倍
MOV @Ri, A	累加器内容送间接 RAM 单元	1	12	2	6 倍
MOV @Ri, direct	直接地址单元数据送入间接 RAM 单元	2	24	3	8 倍
MOV @Ri, #data	立即数送入间接 RAM 单元	2	12	2	6 倍
MOV DPTR,#data16	16 位立即数送入数据指针	3	24	3	8 倍
MOVC A, @A+DPTR	以 DPTR 为基地址变址寻址单元中的数据送入累加器	1	24	5	4.8 倍
MOVC A, @A+PC	以 PC 为基地址变址寻址单元中的数据送入累加器	1	24	4	6 倍
MOVX A, @Ri	将逻辑上在片外、物理上在片内的扩展 RAM(8 位地址)的内容送入累加器 A 中，读操作	1	24	3	8 倍
MOVX @Ri, A	将累加器 A 的内容送入逻辑上在片外、物理上在片内的扩展 RAM(8 位地址)中，写操作	1	24	4	8 倍
MOVX A, @DPTR	将逻辑上在片外、物理上在片内的扩展 RAM(16 位地址)的内容送入累加器 A 中，读操作	1	24	2	12 倍
MOVX @DPTR, A	将累加器 A 的内容送入逻辑上在片外、物理上在片内的扩展 RAM(16 位地址)中，写操作	1	24	3	8 倍
MOVX A, @Ri	将逻辑上在片外、物理上也在片外的扩展 RAM(8 位地址)的内容送入累加器 A 中，读操作	1	24	5×N+2 N 的取值见下列说明	*Note1
MOVX @Ri, A	将累加器 A 的内容送入逻辑上在片外、物理上也在片外的扩展 RAM(8 位地址)中，写操作	1	24	5×N+3 N 的取值见下列说明	*Note1
MOVX A, @DPTR	将逻辑上在片外、物理上也在片外的扩展 RAM(16 位地址)的内容送入累加器 A 中，读操作	1	24	5×N+1 N 的取值见下列说明	*Note1
MOVX @DPTR, A	将累加器 A 的内容送入逻辑上在片外、物理上也在片外的扩展 RAM(16 位地址)中，写操作	1	24	5×N+2 N 的取值见下列说明	*Note1
PUSH direct	直接地址单元中的数据压入堆栈	2	24	3	8 倍
POP direct	栈底数据弹出送入直接地址单元	2	24	2	12 倍
XCH A, Rn	寄存器与累加器交换	1	12	2	6 倍
XCH A,direct	直接地址单元与累加器交换	2	12	3	4 倍
XCH A, @Ri	间接 RAM 与累加器交换	1	12	3	4 倍
XCHD A, @Ri	间接 RAM 的低半字节与累加器交换	1	12	3	4 倍

当 EXRTS[1:0] = [0,0]时，表中 N=1

当 EXRTS[1:0] = [0,1]时，表中 N=2

当 EXRTS[1:0] = [1,0]时，表中 N=4

当 EXRTS[1:0] = [1,1]时，表中 N=8.　　EXRTS[1：0]为寄存器 BUS_SPEED 中的 B1，B0 位

布尔变量操作类指令

助记符	功能说明	字节数	传统 8051 单片机所需时钟	STC15 系列单片机所需时钟 (采用 STC-Y5 超高速 1T 8051 内核)	效率提升
CLR C	清零进位位	1	12	1	12 倍
CLR bit	清 0 直接地址位	2	12	3	4 倍
SETB C	置 1 进位位	1	12	1	12 倍
SETB bit	置 1 直接地址位	2	12	3	4 倍
CPL C	进位位求反	1	12	1	12 倍
CPL bit	直接地址位求反	2	12	3	4 倍
ANL C, bit	进位位和直接地址位相“与”	2	24	2	12 倍
ANL C, /bit	进位位和直接地址位的反码相“与”	2	24	2	12 倍
ORL C, bit	进位位和直接地址位相	2	24	2	12 倍
ORL C, /bit	进位位和直接地址位的反码	2	24	2	12 倍
MOV C, bit	直接地址位送入进位位	2	12	2	12 倍
MOV bit, C	进位位送入直接地址位	2	24	3	8 倍
JC rel	进位位为 1 则转移	2	24	3	8 倍
JNC rel	进位位为 0 则转移	2	24	3	8 倍
JB bit, rel	直接地址位为 1 则转移	3	24	5	4.8 倍
JNB bit, rel	直接地址位为 0 则转移	3	24	5	4.8 倍
JBC bit, rel	直接地址位为 1 则转移，该位清 0	3	24	5	4.8 倍

控制转移类指令

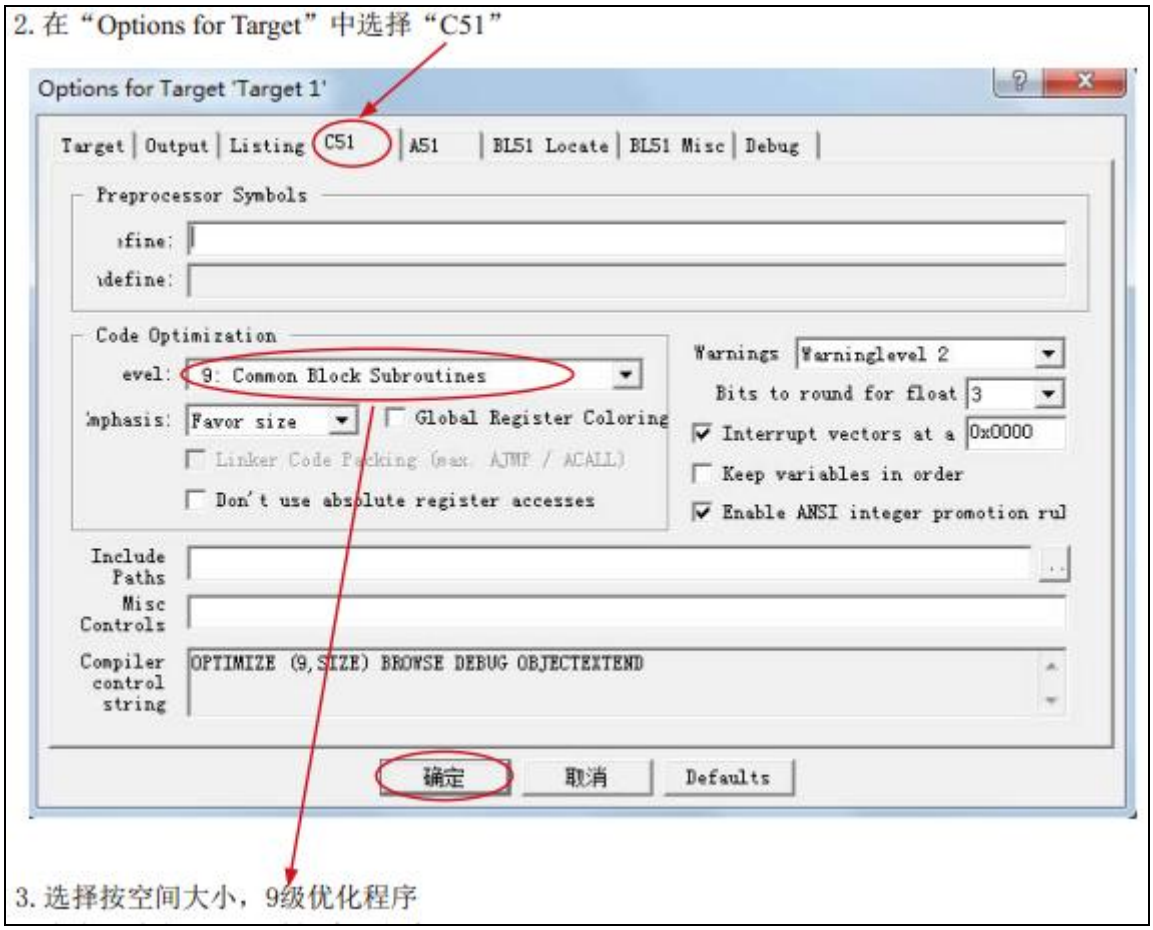
助记符	功能说明	字节数	传统 8051 单片机所需时钟	STC15 系列单片机所需时钟 (采用 STC-Y5 超高速 1T 8051 内核)	效率提升
ACALL addr11	绝对（短）调用子程序	2	24	4	6 倍
LCALL addr16	长调用子程序	3	24	4	6 倍
RET	子程序返回	1	24	4	6 倍
RETI	中断返回	1	24	4	6 倍
AJMP addr11	绝对（短）转移	2	24	3	8 倍
LJMP addr16	长转移	3	24	4	6 倍
SJMP re1	相对转移	2	24	3	8 倍
JMP @A+DPTR	相对于 DPTR 的间接转移	1	24	5	4.8 倍
JZ re1	累加器为零转移	2	24	4	6 倍
JNZ re1	累加器非零转移	2	24	4	6 倍
CJNE A, direct, re1	累加器与直接地址单元比较，不相等则转移	3	24	5	4.8 倍
CJNE A, #data, re1	累加器与立即数比较，不相等则转移	3	24	4	6 倍
CJNE Rn, #data, re1	寄存器与立即数比较，不相等则转移	3	24	4	6 倍
CJNE @Ri, #data, re1	间接 RAM 单元与立即数比较，不相等则转移	3	24	5	4.8 倍
DJNZ Rn, re1	寄存器减 1，非零转移	2	24	4	6 倍
DJNZ direct, re1	直接地址单元减 1，非零转移	3	24	5	4.8 倍
NOP	空操作	1	12	1	12 倍

附录J 如何利用 Keil C 软件减少代码长度

在 Kei1 C 软件中选择作如下设置，能将原代码长度最大减少 10K

1.在“Project”菜单中选择“Options for Target2.在“Options for Target”中选择“C51”

2.在“Options for Target”中选择“C51”



3.选择按空间大小，9 级优化程序

4.点击“确定”后，重新编译程序即可

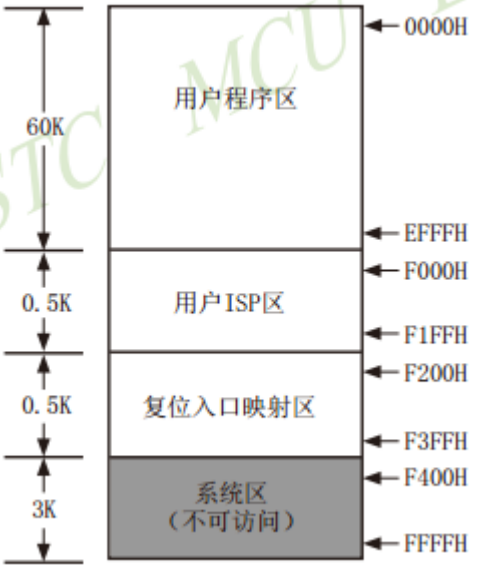
附录K 使用 STC 的 IAP 系列单片机开发自己的 ISP 程序

----基于 IAP15F2K61S2 单片机

随着 IAP（In-Application-Programming）技术在单片机领域的不断发展，给应用系统程序代码升级带来了极大的方便。STC 的串口 ISP（In-System-Programming）程序就是使用 IAP 功能来对用户的程序进行在线升级的，但是出于对用户代码的安全着想，底层代码和上层应用程序都没有开源，为此 STC 推出了 IAP 系列单片机，即整颗 MCU 的 Flash 空间，用户均可在自己的程序中进行改写，从而使得有用户需要开发字节的 ISP 程序的想法得以实现。本文以 STC 的 IAP15F2K61S2 为例，详细说明了使用 STC 的 IAP 单片机开发用户自己的 ISP 程序的方法，并给出了基于 Keil 环境的汇编和 C 源码。

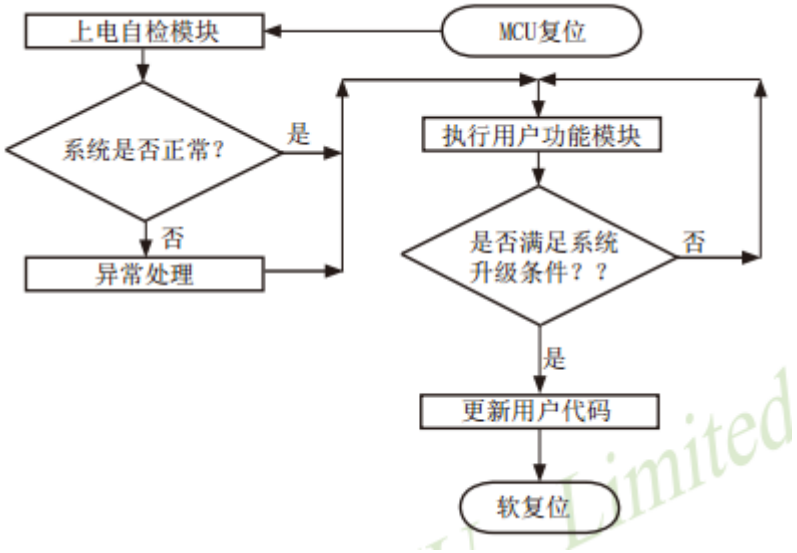
一. 内部 FLASH 规划

示例单片机使用 IAP15F2K61S2，用户可以使用的最大程序空间为 60K 字节，整个 Flash 空间划分如下：



FLASH 空间中，从地址 0000H 开始的连续 60K 字节的空间为用户程序区。当满足特定的下载条件时，用户需要自行将程序跳转到用户 ISP 程序区，此时可对用户程序区进行擦除和改写，以达到更新用户程序的目的。

二. 程序的基本框架



三．下位机固件程序说明

下位机固件程序包括两部分：ISP（ISP 代码）和 AP（用户代码）

ISP 代码（汇编语言）如下：

```
;-------------------------------------------------------------
;----STC IAP 系列单片机实现用户 ISP 演示程序 ----
;-------------------------------------------------------------
;定义常数
UARTBAUD        EQU     0FFE8H      ; 定义串口波特率发生器的重载值(65536-11059200/4/115200)
;ENABLE_IAP     EQU     80H         ;系统工作频率<30MHz
;ENABLE_IAP     EQU     81H         ;系统工作频率<24MHz
;ENABLE_IAP     EQU     82H         ;系统工作频率<20MHz
ENABLE_IAP      EQU     83H         ;系统工作频率<12MHz
;ENABLE_IAP     EQU     84H         ;系统工作频率<6MHz
;ENABLE_IAP     EQU     85H         ;系统工作频率<3MHz
;ENABLE_IAP     EQU     86H         ;系统工作频率<2MHz
;ENABLE_IAP     EQU     87H         ;系统工作频率<1MHz

;-----------------------------------------------
; 定义特殊功能寄存器
AUXR            DATA    08EH        ;附件功能控制寄存器
WDT_CONTR       DATA    0C1H        ;看门狗控制寄存器
IAP_DATA        DATA    0C2H        ;IAP 数据寄存器
IAP_ADDRH       DATA    0C3H        ;IAP 高地址寄存器
IAP_ADDRL       DATA    0C4H        ;IAP 低地址寄存器
IAP_CMD         DATA    0C5H        ;IAP 命令寄存器
IAP_TRIG        DATA    0C6H        ;IAP 命令触发寄存器
IAP_CONTR       DATA    0C7H        ;IAP 控制寄存器

;-----------------------------------------------
; 定义 ISP 模块使用的变量
```

```
ISPCODE         EQU     0F000H      ;ISP 模块入口地址(1 页),同时也是下载接口地址
APENTRY         EQU     0F200H      ;应用程序入口地址数据(1 页)

;-----------------------------------------------
    ORG     0000H

    LJMP    ISP_ENTRY               ;系统复位入口

RESET:
    MOV     SCON, #50H              ;设置串口模式(8 位数据位,波特率可变,无校验位)
    MOV     AUXR, #40H              ;T1 工作于 1T 模式
    MOV     TH1, #HIGH UARTBAUD     ;波特率设置
    MOV     TL1, #LOW UARTBAUD
    SETB    TR1                     ;启动定时器 1

NEXT1:
    MOV     R0, #16

NEXT2:
    JNB     RI, $                   ;等待串口数据
    CLR     RI
    MOV     A, SBUF
    CJNE    A, #7FH,NEXT1           ;判断是否为 7F
    DJNZ    R0, NEXT2
    LJMP    ISPPROGRAM              ;跳转到下载界面
;-----------------------------------------------
;ISP 功能模块
;包括上电自检模块和代码更新模块
;-----------------------------------------------
    ORG     ISPCODE

ISPPROGRAM:
    CLR     A
    MOV     PSW, A                  ;ISP 模块使用第 0 组寄存器
    MOV     IE, A                   ;关闭所有中断
    CLR     RI                      ;清除串口接收标志
    SETB    TI                      ;置串口发送标志
    CLR     TR0
    MOV     SP, #5FH                ; 设置堆栈指针

;-----------------------------------------------
    MOV     A, #5AH                 ;返回 5A 55 到 PC,表示 ISP 擦除模块已准备就绪
    CALL    ISP_SENDUART
    MOV     A, #055H
    CALL    ISP_SENDUART
    CALL    ISP_RECVACK             ;接收应答数据
```

```
    MOV     IAP_ADDRL, #0           ;首先在第 2 页起始地址写  "LJMP ISP_ENTRY"指令
    MOV     IAP_ADDRH, #02H
    CALL    ISP_ERASEIAP
    MOV     A, #02H
    CALL    ISP_PROGRAMIAP          ;编程用户代码复位向量代码
    MOV     A, #HIGH ISP_ENTRY
    CALL    ISP_PROGRAMIAP          ;编程用户代码复位向量代码
    MOV     A, #LOW ISP_ENTRY
    CALL    ISP_PROGRAMIAP          ;编程用户代码复位向量代码

    MOV     IAP_ADDRL, #0           ;用户代码地址从 0 开始
    MOV     IAP_ADDRH, #0
    CALL    ISP_ERASEIAP
    MOV     A, #02H
    CALL    ISP_PROGRAMIAP          ;编程用户代码复位向量代码
    MOV     A, #HIGH ISP_ENTRY
    CALL    ISP_PROGRAMIAP          ;编程用户代码复位向量代码
    MOV     A, #LOW ISP_ENTRY
    CALL    ISP_PROGRAMIAP          ;编程用户代码复位向量代码

    MOV     IAP_ADDRL, #0           ;新代码缓冲区地址
    MOV     IAP_ADDRH, #02H
    MOV     R7,  #119               ;擦除 60K-512 字节

ISP_ERASEAP:
    CALL    ISP_ERASEIAP
    INC     IAP_ADDRH               ;目标地址+512
    INC     IAP_ADDRH
    DJNZ    R7, ISP_ERASEAP         ;判断是否擦除完成

    MOV     IAP_ADDRL, #LOW APENTRY     ;用户代码复位入口页
    MOV     IAP_ADDRH, #HIGH APENTRY
    CALL    ISP_ERASEIAP

    MOV     A, #5AH                 ;返回 5A A5 到 PC,表示 ISP 编程模块已准备就绪
    CALL    ISP_SENDUART
    MOV     A, #0A5H
    CALL    ISP_SENDUART
    CALL    ISP_RECVACK             ;接收应答数据

    CALL    ISP_RECVUART            ;接收长度高字节
    MOV     R0, A
    CALL    ISP_RECVUART            ;接收长度低字节
    MOV     R1, A
    CLR     C                       ;将(总长度-3)的补码存入 DPTR
```

```
    MOV     A, #03H
    SUBB    A, R1
    MOV     DPL, A
    CLR     A
    SUBB    A, R0
    MOV     DPH, A

    CALL    ISP_RECVUART        ;映射用户代码复位入口代码到映射区
    CALL    ISP_PROGRAMIAP      ;0000
    CALL    ISP_RECVUART
    CALL    ISP_PROGRAMIAP      ;0001
    CALL    ISP_RECVUART
    CALL    ISP_PROGRAMIAP      ;0002

    MOV     IAP_ADDRL, #03H     ;用户代码起始地址
    MOV     IAP_ADDRH, #00H

ISP_PROGRAMNEXT:
    CALL    ISP_RECVUART        ;接收代码数据
    CALL    ISP_PROGRAMIAP      ;编程到用户代码区
    INC     DPTR
    MOV     A, DPL
    ORL     A, DPH
    JNZ     ISP_PROGRAMNEXT     ;长度检测

ISP_SOFTRESET:
    MOV     IAP_CONTR, #20H     ;软件复位系统
    SJMP    $

;-------------------------------------------------
;用户 ISP 主程序的入口地址
;-------------------------------------------------
ISP_ENTRY:
    MOV     IAP_ADDRL, #00H     ;电压测试模块
    MOV     IAP_ADDRH, #0F0H    ;测试地址 F000H
    MOV     IAP_DATA, #53H      ;测试数据 1
    CALL    ISP_READIAP
    XRL     A, #53H             ;测试是否可以读出数据
    JZ      ISP_ENTRY           ;若读不出数据,则需等待电压稳定
    INC     IAP_ADDRL           ;测试地址 F001H
    MOV     IAP_DATA, #45H      ;测试数据 2
    CALL    ISP_READIAP
    XRL     A, #45H             ;测试是否可以读出数据
    JZ      ISP_ENTRY           ;若读不出数据,则需等待电压稳定

    JMP     GOTOAP              ;电压稳定后开始运行用户代码
```

```
    MOV     TMOD, #00H
    MOV     SCON, #5AH          ;设置串口模式(8 位数据位,波特率可变,无校验位)
    MOV     AUXR, #40H          ;T1 工作于 1T 模式
    MOV     TH1, #HIGH UARTBAUD ;波特率设置
    MOV     TL1, #LOW UARTBAUD
    SETB    TR1                 ;启动定时器 1

    MOV     R0, #16

ISP_CHECKNEXT:
    CALL    ISP_RECVUART        ;接收同步数据
    JC      GOTOAP
    CJNE    A, #7FH,GOTOAP      ;判断是否为 7F
    DJNZ    R0, ISP_CHECKNEXT
    MOV     A, #5AH             ;返回 5A 69 到 PC,表示 ISP 模块已准备就绪
    CALL    ISP_SENDUART
    MOV     A, #69H
    CALL    ISP_SENDUART
    CALL    ISP_RECVACK         ;接收应答数据
    LJMP    ISPPROGRAM          ;跳转到下载界面

;----------------------------------------------
;跳转到用户程序区,开始正常运行用户程序
;----------------------------------------------
GOTOAP:
    CLR     A                   ;将 SFR 恢复为复位值
    MOV     TCON, A
    MOV     TMOD, A
    MOV     TL0, A
    MOV     TH0, A
    MOV     TL1, A
    MOV     TH1, A
    MOV     SCON, A
    MOV     AUXR, A
    LJMP    APENTRY             ;正常运行用户程序

;----------------------------------------------
;接收来自于上位机的串口应答数据
;----------------------------------------------
ISP_RECVACK:
    CALL    ISP_RECVUART
    JC      GOTOAP
    XRL     A, #7FH
    JZ      ISP_RECVACK         ;跳过同步数据  (7FH)
    CJNE    A, #25H, GOTOAP     ;应答数据 1 检测(5AH)
    CALL    ISP_RECVUART
```

```
    JC      GOTOAP
    XRL     A, #69H                 ;应答数据 2 检测(69H)
    JNZ     GOTOAP
    RET
;----------------------------------------------
;接收 1 字节串口数据
;出口参数: ACC (接收到的数据)
;出口参数: C (1:超时)
;----------------------------------------------
ISP_RECVUART:
    CLR     TR0
    CLR     A
    MOV     TL0, A                  ;初始化超时定时器
    MOV     TH0, A
    CLR     TF0
    SETB    TR0
    MOV     WDT_CONTR, #17H         ;清看门狗

ISP_RECVWAIT:
    JBC     TF0, ISP_RECVTIMEOUT    ;超时检测
    JNB     RI, ISP_RECVWAIT        ;等待接收完成
    MOV     A, SBUF                 ;读取串口数据
    CLR     RI                      ;清除标志
    CLR     C                       ;正确接收串口数据
    RET

ISP_RECVTIMEOUT:
    SETB    C                       ;超时退出
    RET
;----------------------------------------------
;发送 1 字节串口数据
;入口参数: ACC (待发送的数据)
;----------------------------------------------
ISP_SENDUART:
    MOV     WDT_CONTR, #17H         ;清看门狗
    JNB     TI, ISP_SENDUART        ;等待前一个数据发送完成
    CLR     TI                      ;清除标志
    MOV     SBUF, A                 ;发送当前数据
    RET
;----------------------------------------------
;擦除 IAP 扇区
;----------------------------------------------
ISP_ERASEIAP:
    MOV     WDT_CONTR, #17H             ;清看门狗
    MOV     IAP_CONTR, #ENABLE_IAP      ;使能 IAP 功能
    MOV     IAP_CMD, #3                 ;擦除命令
```

```
    MOV     IAP_TRIG, #5AH              ;触发 ISP 命令
    MOV     IAP_TRIG, #0A5H
    RET
;----------------------------------------------
;编程 IAP 字节
;入口参数: ACC (待编程的数据)
;----------------------------------------------
ISP_PROGRAMIAP:
    MOV     WDT_CONTR, #17H             ;清看门狗
    MOV     IAP_CONTR, #ENABLE_IAP      ;使能 IAP 功能
    MOV     IAP_CMD, #2                 ;编程命令
    MOV     IAP_DATA, A                 ;将当前数据送 IAP 数据寄存器
    MOV     IAP_TRIG, #5AH              ;触发 ISP 命令
    MOV     IAP_TRIG, #0A5H
    MOV     A, IAP_ADDRL                ;IAP 地址+1
    ADD     A, #01H
    MOV     IAP_ADDRL, A
    MOV     A, IAP_ADDRH
    ADDC    A, #00H
    MOV     IAP_ADDRH, A
    RET
;----------------------------------------------
;读取 IAP 字节
;出口参数: ACC (读出的数据)
;----------------------------------------------
ISP_READIAP:
    MOV     IAP_CONTR, #ENABLE_IAP      ;使能 IAP 功能
    MOV     IAP_CMD, #1                 ;读命令
    MOV     IAP_TRIG, #5AH              ;触发 ISP 命令
    MOV     IAP_TRIG, #0A5H
    MOV     A, IAP_DATA
    RET
;----------------------------------------------
    ORG     APENTRY
    LJMP    RESET
;----------------------------------------------
    END
```

ISP 代码包括如下外部接口模块：
ISPPROGRAM：程序下载入口地址，绝对地址 F000H
ISP_ENTRY：上电系统自检程序（系统自动调用）
对于用户程序而言，用户只需要在满足下载条件时，将 PC 值跳转到 ISPPROGRAM（即 F000H 的绝对地址），即可实现代码更新。
用户代码(C 语言及汇编语言)示例如下:

1、C 语言用户代码

年 ama:

```
/*---STC IAP 系列单片机实现用户 ISP 演示程序
#include "regs1 .h"
/*定义常数  */
#define   FOSC              11059200L                 //系统时钟频率
#define   BAUD              115200                    //定义串口波特率
#define   RELOAD            (65536 - FOSC/4/BAUD)     //定时器重载值
#define   ISPPROGRAM()      ((void (code *)())0xF000)()  //ISP 下载程序入口地址

/*  定义相关 SFR */
sfr       AUXR = 0x8E;                                //辅助寄存器

void uart() interrupt 4 using 1             //串口中断服务程序
{
    static char cnt7f = 0;                  //7f 的计数器
    if (TI) TI = 0;
    if (RI)
    {
        if (SBUF == 0x7f)
        {
            if (++cnt7f >= 16)
            {
                ISPPROGRAM();       //调用下载模块(****重要语句****)
            }
        }
        else
        {
            cnt7f = 0;
        }
        RI = 0;
    }
}

void main()
{
    SCON =  0x50;                   //设置串口模式(8 位数据位,波特率可变,无校验位)
    AUXR = 0x40;                    //T1 工作于 1T 模式
    TH1 = RELOAD >> 8;              //波特率设置
    TL1 = RELOAD;
    TR1 = 1;                        //启动定时器 1
    ES = 1;                         //使能串口中断
    EA = 1;                         //打开全局中断开关
    while (1)
    {
        P1++;                       //用户示例代码
    }
```

```
}
```

2、汇编语言用户代码

```
;/*-------------------------------------------------------------*/
;/*----STC IAP 系列单片机实现用户 ISP 演示程序----*/
;-------------------------------------------------------------
;/*  定义常数*/
UARTBAUD        EQU     0FFE8H      ;  定义串口波特率  (65536-11059200/4/115200)
ISPPROGRAM      EQU     0F000H      ;ISP 下载程序入口地址

;----------------------------------------------
;/*  定义特殊功能寄存器*/
AUXR            DATA    08EH        ;附件功能控制寄存器
;----------------------------------------------
;/*  定义用户变量*/
CNT7F           DATA    60H         ;接收 7F 的计数器,当连续接收到 16 次 7F 后进入 ISP 下载模式
;----------------------------------------------
;/*中断向量表*/
    ORG     0000H
    LJMP    START                   ;系统复位入口

    ORG     0023H
    LJMP    UART_ISR                ;串口中断入口
;----------------------------------------------
;/*串口中断服务程序*/
UART_ISR:
    PUSH    ACC
    PUSH    PSW
    MOV     PSW, #08H
    JNB     TI, CHECKRI             ;检测发送中断
    CLR     TI                      ;清除标志

CHECKRI:
    JNB     RI,   UARTISR_EXIT      ;检测接收中断
    CLR     RI                      ;清除标志
    MOV     A, SBUF
    CJNE    A, #7FH, ISNOT7F
    INC     CNT7F
    MOV     A, CNT7F
    CJNE    A, #16, UARTISR_EXIT
    LJMP    ISPPROGRAM              ;调用下载模块(****重要语句****)

ISNOT7F:
    MOV     CNT7F, #0

UARTISR_EXIT:
```

```
    POP     PSW
    POP     ACC
    RETI
;-----------------------------------------------
;/*主程序入口*/
START:
    MOV     R0, #7FH                ;清 RAM
    CLR     A
    MOV     @R0, A
    DJNZ    R0, $-1
    MOV     SP, #7FH                ;初始化 SP
    MOV     SCON, #50H              ;设置串口模式(8 位数据位,波特率可变,无校验位)
    MOV     AUXR, #40H              ;T1 工作于 1T 模式
    MOV     TH1, #HIGH UARTBAUD     ;波特率设置
    MOV     TL1, #LOW UARTBAUD
    SETB    TR1                     ;启动定时器 1
    SETB    ES                      ;使能串口中断
    SETB    EA                      ;开中断总开关

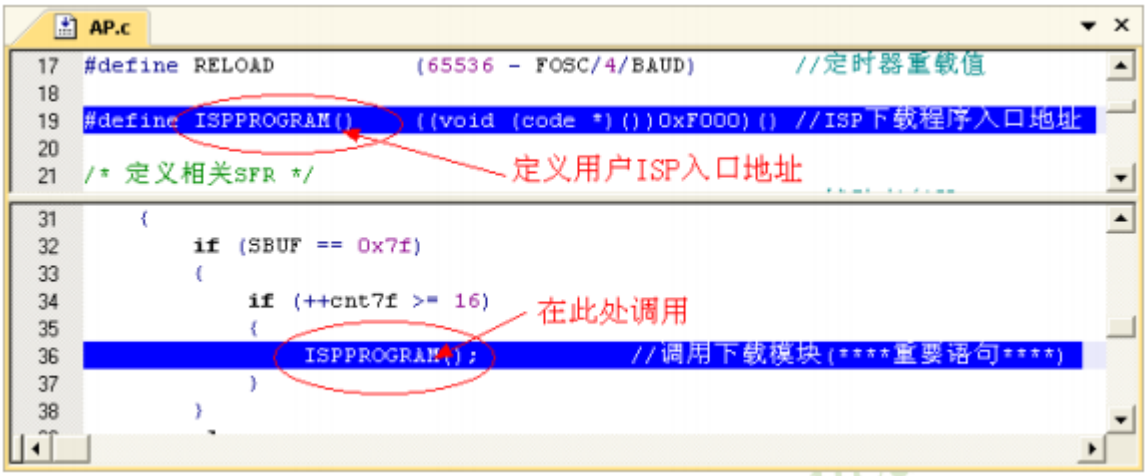
START:
    INC     P0                      ;用户示例代码
    SJMP    START
;-----------------------------------------------
END
```

对于汇编语言编写的用户代码需要注意一点：位于 0000H 的复位入口地址处的指令必须是一个长跳转语句（类似 LJMP START）。在用户代码中，需要设置好串口，并在满足下载条件时，跳转到 ISPPROGRAM（即 F000H 的绝对地址），以实现代码更新。

对于汇编语言编写的用户代码，我们可以使用如下图的方法进行调用：



对于 C 语言编写的用户代码，我们可以使用如下图的方法进行调用：



四．上位机应用程序说明

上位机应用程序请用户在 STCAI 官方网站 www.STCMCU.com 下载。

上位机演示程序是基于 MFC 的对话框项目，对于串口的访问是直接调用 Windows 的 API 函数，而没有使用串口控件，从而省去的控件的注册以及系统版本不兼容的诸多问题。界面较简单，只是为这一功能的实现提供了一个框架，其他的功能及要求均还可以往上面添加。

上位机程序的核心模块是基于类 CISPDlg 的一个友元函数“UINT Download(LPVOIDpParam):

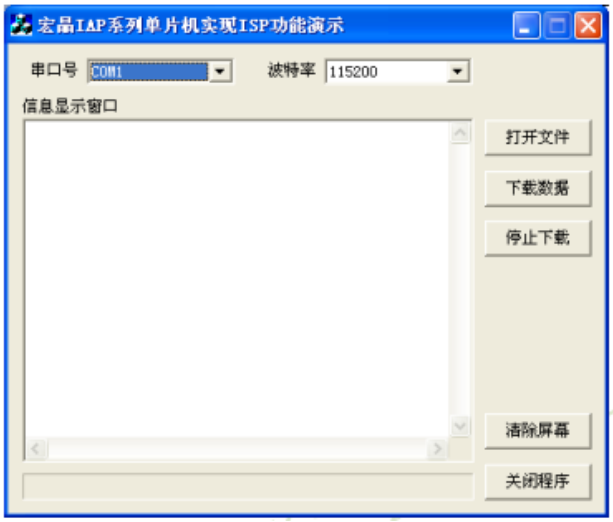
```
// Construction
public:
    CISPDlg(CWnd* pParent = NULL);  // standard constructor

    friend UINT Download(LPVOID pParam);
```

此函数负责与下位机通讯，发送各种通讯命令来完成对用户程序的更新。用户可以根据各自不同的需求增加命令。

五．上位机应用程序的使用方法

1、打开上位机界面，如下图



2、选择串口号，设置与下位机相同的串口波特率

3、打开要下载的源数据文件，Bin 或者 Intelhex 格式均可以

4、点击“下载数据”按钮即可开始下载数据

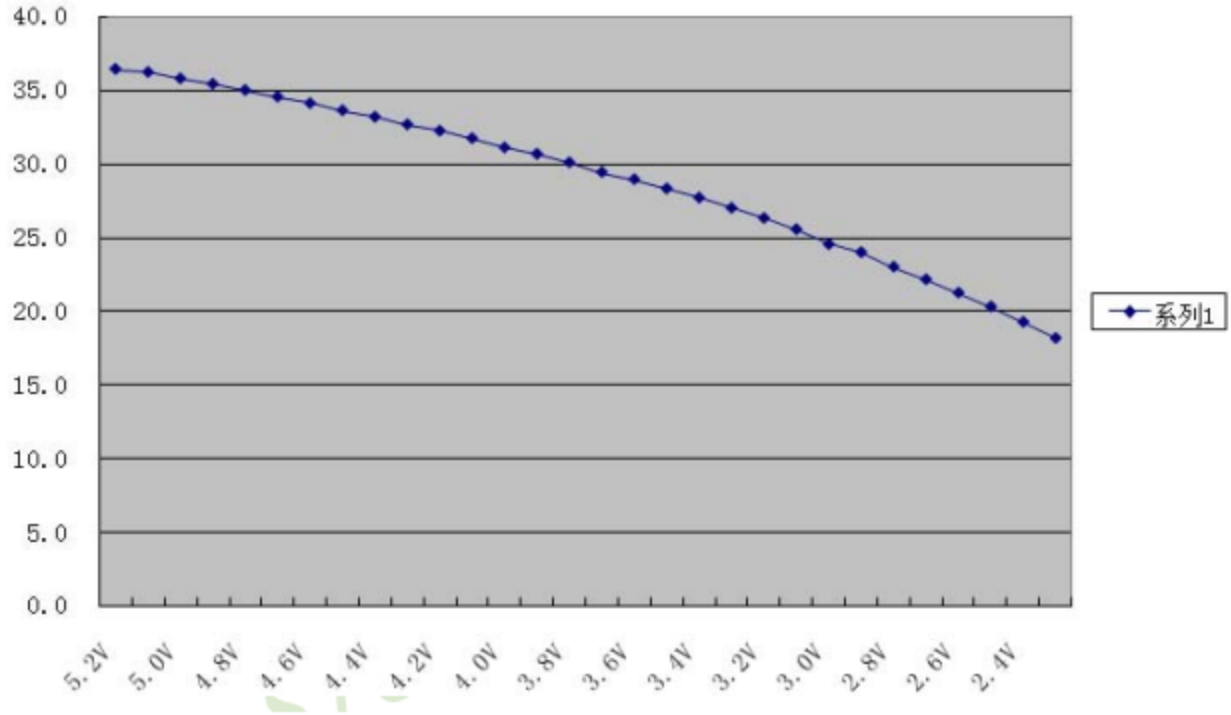
六．下位机固件程序的使用方法

下位机的目标文件有两个：“IAPISP.hex”和“AP.hex”，对于一块新的单片机，第一次必须使用我司的 ISP 下载工具将“IAPISP.hex”写入到芯片内。附件中的“AP.hex”是用户程序模板，当满足下载条件时（模板代码中的下载条件是连续就收到 16 个 7FH 的数据），用户将程序跳转到用户 ISP 入口地址（F000H）处，即可实现代码更新。

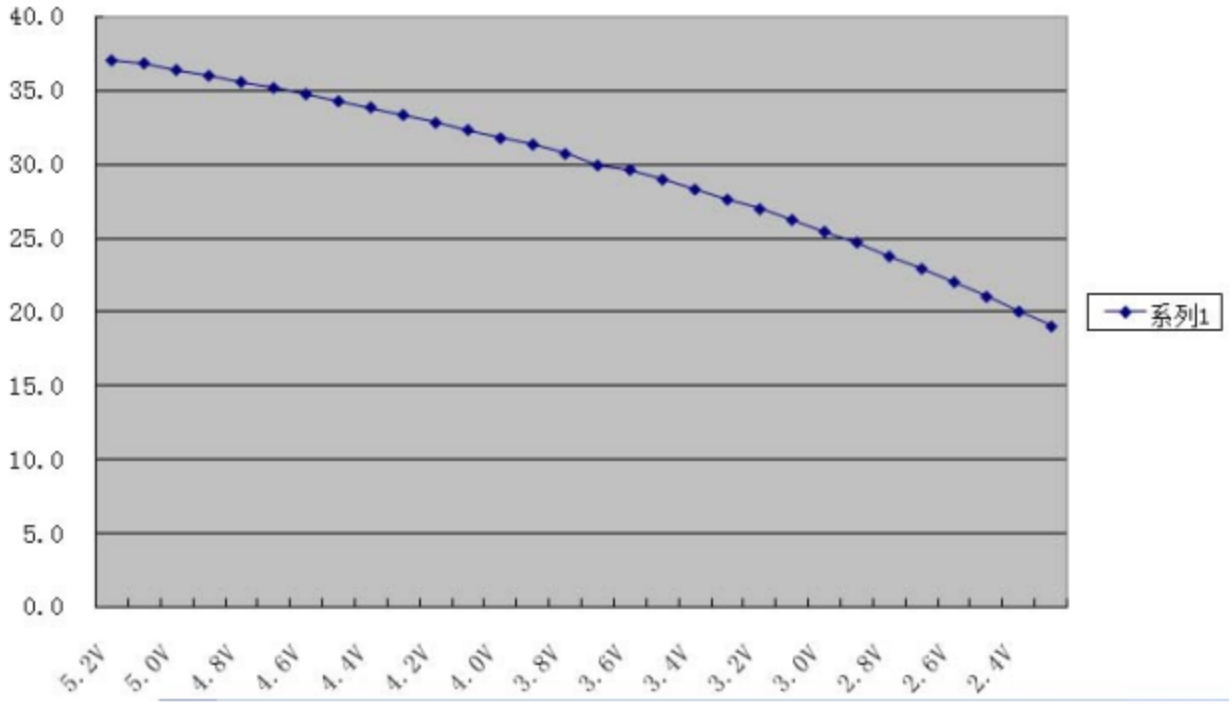
附录L 掉电唤醒定时器频率与电压的关系

----基于 STC15W408AS 和 STC15W204S 系列单片机

STC15W404AS 掉电唤醒定时器频率与电压关系图



STC15W204S掉电唤醒定时器频率与电压关系图



附录M 更新记录

2025/07/02

1. 增加章节：STC15W408AS 系列--管脚图及下载线路图，最小系统（LQFP32）
2. 增加章节：STC15F2K60S2 系列--管脚图及下载线路图，最小系统（LQFP48/QFN48）
3. 增加章节：BL51/LX51 链接器和扩展中断号工具，REMOVEUNUSED 说明
4. 增加章节：USB-TypeC 封装尺寸图，原理图

2025/06/17

1. 增加章节：如何对 IAP15F2K61S2 系列及 IAP15W4K61S4 系列仿真
2. 增加：STC15W4K32S4 系列的下载线路图
3. 更新：EEPROM 触发寄存器（IAP_TRIG）的说明并更新相关范例程序
4. 更新：部分下载线路图及图片

2025/05/16

1. 增加：STC15F104W 系列、STC15W104 系列、STC15W204S 系列、STC15W408AS 系列、STC15F60S2 系列、STC15F408AD 系列的下载线路图

2025/05/09

1. 更新图形并校验至第 26.9 章节

2025/04/30

1. 更新：部分图形

2025/04/21

1. 增加章节：Ai8051U 内部结构图
2. 增加章节：嵌入汇编代码在 C 语言中，Keil C51 编译器

2025/04/18

1. STC15 系列技术手册全新改版

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