STC89C51RC/RD+系列单片机器件手册

---高速，高可靠
---超低功耗，超低价
---超强抗静电，超强抗干扰
---ISP/IAP，在线编程，无需编程器/仿真器

STC89C51RC,	STC89LE51RC
STC89C52RC,	STC89LE52RC
STC89C53RC,	STC89LE53RC
STC89C54RD+,	STC89LE54RD+
STC89C58RD+,	STC89LE58RD+
STC89C510RD+,	STC89LE510RD+
STC89C512RD+,	STC89LE512RD+
STC89C514RD+,	STC89LE514RD+
STC89C516RD+,	STC89LE516RD+

请使用采用最新第六代加密技术的STC11/10xx和STC12C5Axx系列单片机取代全球各厂家均已被解密的89系列单片机

STC-ISP：最方便的在线升级软件，无需编程器，无需仿真器

STC——8051单片机全球第一品牌，全球最大的8051单片机设计公司

请同行不要再抄袭我们的规格、设计和管脚排列，再抄袭就很无耻了

全部中国本土独立自主知识产权，请全体中国人民支持，您的支持是中国本土力量前进的有力保证.

封装后，全部175℃八小时高温烘烤，高品质制造保证

技术支持网站：www.STCMCU.com　www.GXWMCU.com

Update date: 2015/8/10

STC micro™ 宏晶科技

超强抗干扰，加密性强

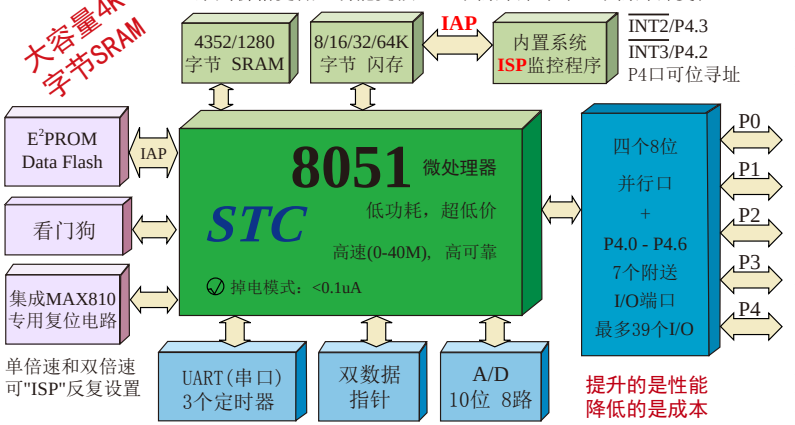
8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

官方网站：www.STCMCU.com www.GXWMCU.com
南通 Tel: 0513-5501 2928 5501 2929
深圳 Tel: 0755-8294 8411 8294 8412

宏晶·STC 89C51 90C51 系列单片机，直接兼容传统8051

欢迎使用STC90C51系列(软硬件完全兼容)取代89C51系列，STC90C51系列抗干扰能力更强，成本更低
或请使用第六代加密技术加密、加密性强的STC11/10xx系列取代89C51系列
或强烈建议使用第九代加密技术加密、无法解密的STC15xx系列取代89C51系列，
STC15系列价格更低、功能更强，且不需外部时钟、不需外部复位

已有大批量供货10年以上的历史
可用IAP15F2K61S2仿真(仅供参考)



选择宏晶·STC90C51系列单片机理由：

★ 超大容量SRAM，最高达4.2K字节SRAM
★ 直接取代传统8051单片机，成本更低
★ 10位A/D，可达25万次/秒
★ 超强抗干扰：
1. 高抗静电(ESD保护，整机轻松过2万伏静电测试)
2. 轻松过4KV快速脉冲干扰(EFT测试)
3. 宽电压，不怕电源抖动
4. 宽温度范围，-40℃~+85℃
★ 三大降低单片机时钟对外部电磁辐射的措施：
----出口欧美的有力保证
1. 禁止ALE输出；
2. 如选6时钟/机器周期，外部时钟频率可降低一半；
3. 单片机时钟振荡器增益可设为1/2gain
★ 超低功耗：
1. 掉电模式：典型功耗功耗 <0.1uA
2. 正常工作模式：4mA -7mA
3. 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。
★ 在系统可编程，无需编程器及仿真器，可远程升级
★ 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
★ 全部175℃，8小时高温烘烤，高可靠制造保证

型号	工作电压(V)	Flash程序存储器(字节)	SRAM字节	EEPROM(字节)	定时器	A/D 10位8路	降低EMI	双倍速	I/O口	支持掉电唤醒外部中断	内置复位	看门狗	ISP	IAP	LQFP44价格	PIDP40价格	PLCC44价格	兼容传统落后型号	价格更低 功能更强 不需外部时钟 不需外部复位的替代型号
STC89C51RC	5.5-3.5	4K	512	9K	3个	-	√	√	39个	4个	有	有	√	√	¥2.8	¥3.3	¥3.4	89C51	
STC90C51RC	5.5-3.5	4K	512	9K	3个	-	√	√	39个	4个	有	有	√	√	¥2.7	¥3.2	¥3.3	89C51	STC15W404S
STC15W404S	5.5-2.5	4K	512	9K	3个	-	√		42个	5个	强	强	√	√	¥2.5	¥3.0			
STC89C52RC	5.5-3.5	8K	512	5K	3个	-	√	√	39个	4个	有	有	√	√	¥2.8	¥3.1	¥3.4	89C52	
STC90C52RC	5.5-3.5	8K	512	5K	3个	-	√	√	39个	4个	有	有	√	√	¥2.7	¥3.0	¥3.3	89C52	STC15W408S
STC15W408S	5.5-2.5	8K	512	5K	3个	-	√		42个	5个	强	强	√	√	¥2.7	¥3.0			
STC89C53RC	5.5-3.5	12K	512	-	3个	-	√	√	39个	4个	有	有	√	√	¥3.5	¥4.0	¥4.1	89C53	
STC90C53RC	5.5-3.5	12K	512	-	3个	-	√	√	39个	4个	有	有	√	√	¥3.4	¥3.9	¥4.0	89C54	IAP15W413S
IAP15W413S	5.5-2.5	13K	512	IAP	3个	-	√		42个	5个	强	强	√	√	¥2.8	¥3.3			
STC89C54RD+	5.5-3.5	16K	1280	45K	3个	-	√	√	39个	4个	有	有	√	√	¥4.5	¥5.0	¥5.1	89C54	
STC90C54RD+	5.5-3.5	16K	1280	45K	3个	-	√	√	39个	4个	有	有	√	√	¥4.4	¥4.9	¥5.0	89C54	STC15W1K16S
STC15W1K16S	5.5-2.6	16K	1024	13K	3个	-	√		42个	5个	强	强	√	√	¥3.5	¥4.0			
STC89C58RD+	5.5-3.5	32K	1280	29K	3个	-	√	√	39个	4个	有	有	√	√	¥4.5	¥5.0	¥5.1	89C58	
STC90C58RD+	5.5-3.5	32K	1280	29K	3个	-	√	√	39个	4个	有	有	√	√	¥4.4	¥4.9	¥5.0	89C58	IAP15W1K29S
IAP15W1K29S	5.5-2.6	29K	1024	IAP	3个	-	√		42个	5个	强	强	√	√	¥3.8	¥4.3			
STC89C516RD+	5.5-3.5	61K	1280		3个	-	√	√	39个	4个	有	有	√	√	¥4.5	¥5.0	¥5.1	89C516	
STC90C516RD+	5.5-3.5	61K	1280		3个	-	√	√	39个	4个	有	有	√	√	¥4.4	¥4.9	¥5.0	89C516	IAP15F2K61S
IAP15F2K61S	5.5-3.8	61K	2048	IAP	3个	-	√		42个	5个	强	强	√	√	¥4.0	¥4.5			
STC90C510RD+	5.5-3.5	40K	1280	21K	3个	-	√	√	39个	4个	有	有	√	√	¥4.4	¥4.9	¥5.0	89C516	
STC90C512RD+	5.5-3.5	48K	1280	13K	3个	-	√	√	39个	4个	有	有	√	√	¥4.4	¥4.9	¥5.0	89C516	
STC90C514RD+	5.5-3.5	56K	1280	5K	3个	-	√	√	39个	4个	有	有	√	√	¥4.4	¥4.9	¥5.0	89C516	
STC90C54AD	5.5-3.8	16K	4352	45K	3个	有	√	√	39个	4个	有	有	√	√	¥7.4	¥7.9	¥8.0	特别要求:A/D输入电压需<1/2Vcc	
STC90C58AD	5.5-3.8	32K	4352	29K	3个	有	√	√	39个	4个	有	有	√	√	¥8.4	¥8.9	¥9.0		
STC90C514AD	5.5-3.8	56K	4352	5K	3个	有	√	√	39个	4个	有	有	√	√	¥9.4	¥9.9	¥10.0		
STC90C516AD	5.5-3.8	61K	4352	-	3个	有	√	√	39个	4个	有	有	√	√	¥10.4	¥10.9	¥11.0		

我们直销，所以低价，**以上单价为10K/M起定量，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**

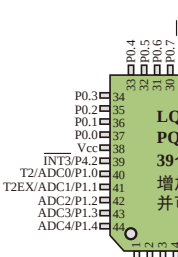
用"不需外部时钟/外部复位"的"STC15W404S/STC15W408S/IAP15W413S, 取代89C51/89C52/89C53, 成本更低
用"不需外部时钟/外部复位"的"STC15W1K16S/STC15W1K24S/IAP15W1K29S" 取代89C54/89C58, 成本更低
用"不需外部时钟/外部复位"的"IAP15F2K61S" 取代89C54/89C58/89C516, 成本更低

宏晶STC90C58AD系列单片机(带A/D转换)的
串口可从[P3.0/RxD，P3.1/TxD]切换到
[P1.6/RxD，P1.7/TxD]，可当双串口使用

宏晶STC90C58AD系列(带A/D转换)P4口地址在C0H，
其它不带A/D转换的89/90系列单片机P4口地址在E8H

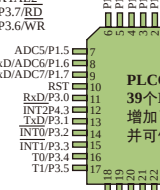
免费索取

从网上下载样品申请单，
传真至深圳申请STC单片机
样片及ISP下载/编程工具



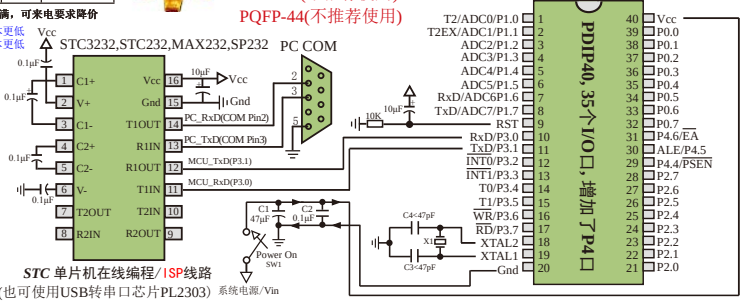
★强烈推荐LQFP44小型封装★
★尽量不选落后的PDIP和PLCC封装★

提供客制化IC设计服务



全部175℃
8小时高温烘烤

封装形式：
PDIP40(正常生产，不推荐使用)
LQFP44(12x12mm，强烈推荐)
PLCC44(不推荐使用)
PQFP-44(不推荐使用)



宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

STC micro 宏晶科技

超强抗干扰，无法解密

8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

官方网站：www.STCMCU.com　www.GXWMCU.com
南通 Tel: 0513-5501 2928　5501 2929
深圳 Tel: 0755-8294 8411　8294 8412

STC15系列1T 8051单片机，超高速双串口，高速A/D，比较器，宽电压，低功耗

不需外部晶振的单片机
不需外部复位的单片机

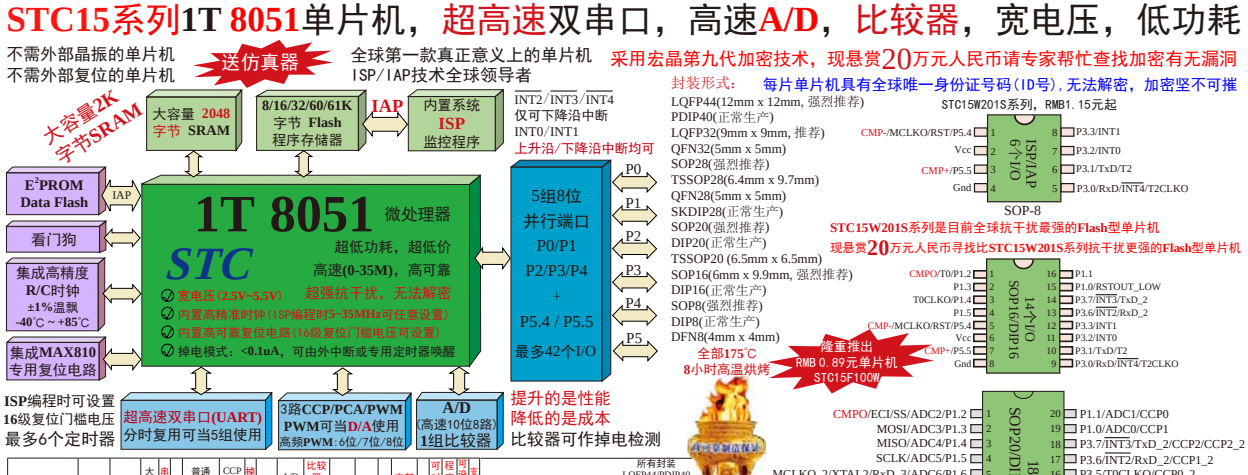
送仿真器

全球第一款真正意义上的单片机
ISP/IAP技术全球领导者

采用宏晶第九代加密技术，现悬赏20万元人民币请专家帮忙查找加密有无漏洞

封装形式：
LQFP44(12mm x 12mm, 强烈推荐)
PDIP40(正常生产)
LQFP32(9mm x 9mm, 推荐)
QFN32(5mm x 5mm)
SOP28(强烈推荐)
TSSOP28(6.4mm x 9.7mm)
QFN28(5mm x 5mm)
SKDIP28(正常生产)
SOP20(强烈推荐)
DIP20(正常生产)
TSSOP20 (6.5mm x 6.5mm)
SOP16(6mm x 9.9mm, 强烈推荐)
DIP16(正常生产)
SOP8(强烈推荐)
DIP8(正常生产)
DFN8(4mm x 4mm)

每片单片机具有全球唯一身份证号码(ID号)，无法解密，加密坚不可摧

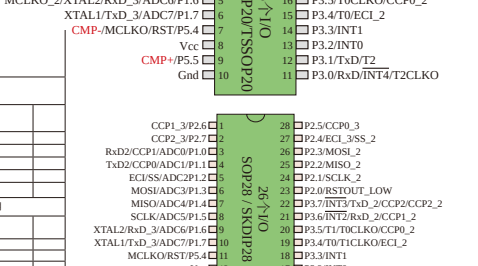


STC15W201S系列是目前全球抗干扰最强的Flash型单片机
现悬赏20万元人民币寻找比STC15W201S系列抗干扰更强的Flash型单片机

隆重推出 RMB 0.89元单片机 STC15F100W

全部175℃ 8小时高温烘烤

所有封装
LQFP44/PDIP40
LQFP32/QFN32
SOP28/SKDIP28/TSSOP28/QFN28
SOP20/DIP20/TSSOP20
SOP16/DIP16
SOP8/DIP8/DFN8
(现此系列未生产PLCC44和SOP32封装)



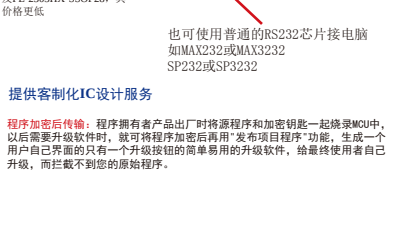
型号	工作电压(V)	Flash程序存储器(byte)	大容量SRAM字节	串行口可掉电唤醒	SPI	普通定时器计数器	CCP/PCA/PWM可当外部中断掉电唤醒	掉电唤醒专用定时器	标准(外部中断4路,3路PWM可当外部中断掉电唤醒,3路D/A可当)	比较器(可当1路ADC使用,可作外部掉电检测)	EEP ROM	内部低压检测中断	内部看门狗	内部高可靠复位(可选复位门槛电压)	可对外输出时钟	可设下次更新程序口令	程序加密后传输(防拦截)	支持RS485下载	SOP8	DIP8	DFN8	SOP16	DIP16	SOP20	DIP20	TSSOP20	SOP28	SKDIP28	TSSOP28	QFN28
大批量现货供应	STC15F100W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																												
STC15F100W	5.5-3.8	0.5K	128	-	-	2(T0/T2)	-	有	5	-	-	-	有	有	8级	有	是	有	是	¥0.89									
STC15F101W	5.5-3.8	1K	128	-	-	2(T0/T2)	-	有	5	-	4K	有	有	8级	有	是	有	是	¥0.99	¥1.1									
STC15F102W	5.5-3.8	2K	128	-	-	2(T0/T2)	-	有	5	-	3K	有	有	8级	有	是	有	是	¥1.1	¥1.2									
STC15F104W	5.5-3.8	4K	128	-	-	2(T0/T2)	-	有	5	-	1K	有	有	8级	有	是	有	是	¥1.2	¥1.3	¥1.4								
IRC15F107W 默认使用内部24MHz时钟	5.5-3.8	7K	128	-	-	2(T0/T2)	-	有	5	-	IAP	有	有	固定	有	是	无	否	¥1.2	¥1.3									
用户可将用户程序区不用的程序Flash当EEPROM使用																													
大批量现货供应中	STC15W10x系列单片机选型价格一览表 串行口功能可由[P3.0/INT4, P3.1]结合定时器实现																		SOP8	DIP8	DFN8								
STC15W100	2.5-5.5	0.5K	128	-	-	2(T0/T2)	-	有	5	-	-	有	有	16级	有	是	有	是	¥0.99										
STC15W101	2.5-5.5	1K	128	-	-	2(T0/T2)	-	有	5	-	4K	有	有	16级	有	是	有	是	¥1.1	¥1.2									
STC15W102	2.5-5.5	2K	128	-	-	2(T0/T2)	-	有	5	-	3K	有	有	16级	有	是	有	是	¥1.2	¥1.3									
STC15W104	2.5-5.5	4K	128	-	-	2(T0/T2)	-	有	5	-	1K	有	有	16级	有	是	有	是	¥1.3	¥1.4	¥1.5								
IRC15W107 默认使用内部24MHz时钟	2.5-5.5	7K	128	-	-	2(T0/T2)	-	有	5	-	IAP	有	有	固定	有	是	无	否	¥1.3	¥1.4									
用户可将用户程序区不用的程序Flash当EEPROM使用																													
大批量现货供应中	STC15W201S系列单片机选型价格一览表，大批量现货供应中																		SOP8	SOP16	DIP16								
STC15W201S	2.5-5.5	1K	256	1	-	2(T0/T2)	-	有	5	-	4K	有	有	16级	有	是	有	是	¥1.15	¥1.2	¥1.35								
STC15W202S	2.5-5.5	2K	256	1	-	2(T0/T2)	-	有	5	-	3K	有	有	16级	有	是	有	是	¥1.25	¥1.3	¥1.45								
STC15W204S	2.5-5.5	4K	256	1	-	2(T0/T2)	-	有	5	-	1K	有	有	16级	有	是	有	是	¥1.35	¥1.4	¥1.55								
IRC15W207S 默认使用内部24MHz时钟	2.5-5.5	7.5K	256	1	-	2(T0/T2)	-	有	5	-	有	IAP	有	有	固定	有	是	无	否	¥1.35	¥1.4	¥1.55								
用户可将用户程序区不用的程序Flash当EEPROM使用																													
大批量现货供应中	STC15W401AS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																		SOP16	DIP16	SOP20	DIP20	TSSOP20	SOP28	SKDIP28	TSSOP28	QFN28
STC15W401AS	2.5-5.5	1K	512	1	有	2(T0/T2)	3-ch	有	5	10-bit	有	5K	有	有	16级	有	是	有	是	¥1.6	¥1.8	¥1.8	¥2.0	¥1.9	¥2.0	¥2.2	¥2.1	¥2.0
STC15W402AS	2.5-5.5	2K	512	1	有	2(T0/T2)	3-ch	有	5	10-bit	有	5K	有	有	16级	有	是	有	是	¥1.8	¥2.0	¥2.0	¥2.2	¥2.1	¥2.2	¥2.4	¥2.3	¥2.2
STC15W404AS	2.5-5.5	4K	512	1	有	2(T0/T2)	3-ch	有	5	10-bit	有	9K	有	有	16级	有	是	有	是	¥1.99	¥2.2	¥2.2	¥2.4	¥2.3	¥2.5	¥2.7	¥2.6	¥2.5
STC15W408AS	2.5-5.5	8K	512	1	有	2(T0/T2)	3-ch	有	5	10-bit	有	5K	有	有	16级	有	是	有	是	¥2.2	¥2.4	¥2.4	¥2.6	¥2.5	¥2.7	¥2.9	¥2.8	¥2.7
IAP15W413AS	2.5-5.5	13K	512	1	有	2(T0/T2)	3-ch	有	5	10-bit	有	IAP	有	有	16级	有	是	有	是	¥2.3	¥2.5	¥2.5	¥2.7	¥2.6	¥2.8	¥3.0	¥2.9	¥2.8
用户可将用户程序区不用的程序Flash当EEPROM使用																			¥2.3	¥2.5	¥2.5	¥2.7	¥2.6	¥2.8	¥3.0	¥2.9	¥2.8		
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(T0/T2)	3-ch	有	5	10-bit	有	IAP	有	有	固定	有	是	无	否	用户可将用户程序区不用的程序Flash当EEPROM使用									
STC15W404AS系列单片机选型价格一览表，大批量现货供应中																			SOP28	SKDIP28	LQFP32	QFN32	PDIP40	LQFP44					
STC15W404S	2.5-5.5	4K	512	1	有	3	-	有	5	-	有	9K	有	有	16级	有	是	有	是	¥2.3	¥2.5	¥2.4		¥3.0	¥2.5					
STC15W408S	2.5-5.5	8K	512	1	有	3	-	有	5	-	有	5K	有	有	16级	有	是	有	是	¥2.5	¥2.7	¥2.6		¥3.0	¥2.7					
IAP15W413S	2.5-5.5	13K	512	1	有	3	-	有	5	-	有	IAP	有	有	16级	有	是	有	是	¥2.6	¥2.8	¥2.7	¥2.65	¥3.3	¥2.8					
用户可将用户程序区不用的程序Flash当EEPROM使用																			¥2.6	¥2.8	¥2.7		¥3.3	¥2.8					
IRC15W415S 默认使用内部24MHz时钟	2.5-5.5	15.5K	512	1	有	3	-	有	5	-	有	IAP	有	有	固定	有	是	无	否	用户可将用户程序区不用的程序Flash当EEPROM使用									
STC15W1K16S系列单片机选型价格一览表，大批量现货供应中																			TSSOP20	SOP28	SKDIP28	LQFP32	QFN32	PDIP40	LQFP44				
STC15W1K16S	2.6-5.5	16K	1K	1	有	3	-	有	5	-	有	13K	有	有	16级	有	是	有	是	¥3.3	¥3.5	¥3.4			¥4.0	¥3.5				
STC15W1K24S	2.6-5.5	24K	1K	1	有	3	-	有	5	-	有	5K	有	有	16级	有	是	有	是	¥3.6	¥3.6	¥3.8	¥3.7		¥4.3	¥3.8				
IAP15W1K29S	2.6-5.5	29K	1K	1	有	3	-	有	5	-	有	IAP	有	有	16级	有	是	有	是	¥3.6	¥3.6	¥3.8	¥3.7	¥3.65	¥4.3	¥3.8				
用户可将用户程序区不用的程序Flash当EEPROM使用																			¥3.6	¥3.6	¥3.8	¥3.7		¥4.3	¥3.8				
IRC15W1K31S 默认使用内部24MHz时钟	2.6-5.5	31.5K	1K	1	有	3	-	有	5	-	有	IAP	有	有	固定	有	是	无	否	用户可将用户程序区不用的程序Flash当EEPROM使用									
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																		TSSOP20	SOP28	SKDIP28	LQFP32	QFN32	PDIP40	LQFP44				
STC15F2K08S2	5.5-4.5	8K	2K	2	有	3	3-ch	有	5	10位	-	53K	有	有	8级	有	是	有	是	¥3.8	¥4.0	¥3.9			¥4.5	¥4.0				
STC15F2K16S2	5.5-4.5	16K	2K	2	有	3	3-ch	有	5	10位	-	45K	有	有	8级	有	是	有	是	¥4.0	¥4.2	¥4.1			¥4.7	¥4.2				
STC15F2K32S2	5.5-4.5	32K	2K	2	有	3	3-ch	有	5	10位	-	29K	有	有	8级	有	是	有	是	¥4.3	¥4.5	¥4.4			¥4.9	¥4.5				
STC15F2K48S2	5.5-4.5	48K	2K	2	有	3	3-ch	有	5	10位	-	13K	有	有	8级	有	是	有	是	¥4.4	¥4.6	¥4.5			¥4.9	¥4.6				
STC15F2K60S2	5.5-4.5	60K	2K	2	有	3	3-ch	有	5	10位	-	1K	有	有	8级	有	是	有	是	¥4.4	¥4.6	¥4.5			¥4.9	¥4.6				
IAP15F2K61S2 本身就是仿真器	5.5-4.5	61K	2K	2	有	3	3-ch	有	5	10位	-	IAP	有	有	8级	有	是	有	是	¥4.4	¥4.4	¥4.6	¥4.5	¥4.45	¥4.9	¥4.6				
用户可将用户程序区不用的程序Flash当EEPROM使用																								¥4.9	¥4.6				
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	-	IAP	有	有	固定	有	是	无	否	用户可将用户程序区不用的程序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						¥4.4	¥3.9				
STC15F2K60S	5.5-4.5	60K	2K	1	有	3	-	有	5	-	-	1K	有	有	8级	有	是	有	是						¥4.4	¥4.5				
IAP15F2K61S	5.5-4.5	61K	2K	1	有	3	-	有	5	-	-	IAP	有	有	8级	有	是	有	是						¥4.5	¥4.0				
用户可将用户程序区不用的程序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							¥3.9				
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							¥3.9				

封装价格 (RMB ¥)

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

特别声明：以15F和15L开头具有SPI功能的芯片只支持"SPI主机模式"；以15W开头且有SPI功能的芯片，SPI主/从机模式均支持
以15F和15L开头的芯片不支持"串行口掉电唤醒"功能，而以15W开头的芯片支持该功能

STC 单片机在线编程ISP/仿真线路 *USB*转串口



另有PL-2303HXD-SSOP28及PL-2303HX-SSOP28，其价格更低

也可使用普通的RS232芯片接电脑
如MAX232或MAX3232
SP232或SP3232

提供客制化IC设计服务

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。

STC micro 宏晶科技

超强抗干扰，无法解密

8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

官方网站：www.STCMCU.com / www.GXWMCU.com
南通 Tel: 0513-5501 2928 5501 2929
深圳 Tel: 0755-8294 8411 8294 8412

STC15W4K32S4系列1T 8051单片机，4K字节SRAM，超高速四串口，6路15位PWM

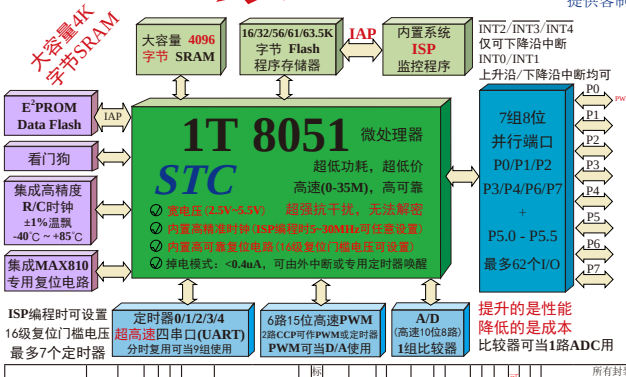
不需外部晶振的单片机
不需外部复位的单片机

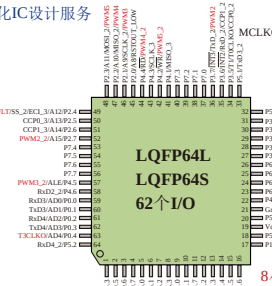
送仿真器

全球第一款真正意义上的单片机
ISP/IAP技术全球领导者

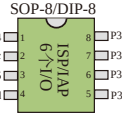
采用宏晶第九代加密技术，现悬赏20万元人民币请专家帮忙查找加密有无漏洞

提供客制化IC设计服务





SOP-8/DIP-8



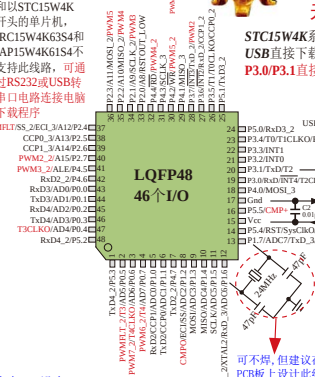
STC15F100W系列, 大批量供货中
RMB0. 89元起 (STC15F100W)

封装形式：
LQFP64L(16mm x 16mm)
LQFP64S(12mm x 12mm)
QFN64(9mm x 9mm)
LQFP48(9mm x 9mm, 强烈推荐)
QFN48(7mm x 7mm)
LQFP44(12mm x 12mm, 强烈推荐)
PDIP40(正常生产，但不推荐)
LQFP32(9mm x 9mm)
SOP28
SKDIP28
强烈推荐选择SOP28,
LQFP44/48/64贴片封装

全部175℃ 8小时高温烘烤

每片单片机具有全球唯一身份证号码(ID号)，无法解密，加密坚不可摧

*STC15W4K*系列及*IAP15W4K58S4*单片机
*USB*直接下载编程线路，*USB-ISP*
P3.0/P3.1直接连接电脑USB的D-/D+



USB-ISP下载时单片机可直接由电脑USB供电，也可不用电脑USB供电，而由系统电源供电。

支持程序加密后传输(防拦截)
也可支持RS485下载

可不焊，但建议在PCB板上设计此线路，如需USB下载建议焊接上此电路以防止USB下载时内部时钟精度不够准

大陆本土宏晶STC姚永平独立创新设计，请不要再抄袭我们的设计、规格和管脚排列，再抄袭就很无...

此线路只针对IAP15W4K58S4和以STC15W4K开头的单片机，IRC15W4K63S4和IAP15W4K61S4不支持此线路，可通过RS232或USB转串口电路连接电脑下载程序

型号	工作电压(V)	Flash程序存储器(byte)	大容量SRAM字节	串行口	片内EEPROM	SPI	普通定时器	8路 PWM		掉电唤醒专用定时器	比较器(可当1路ADC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	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)	支持下载	支持5V单片机直接下载	所有封装 LQFP64S/LQFP64L/QFN64/ LQFP48/QFN48 LQFP44/PDIP40 LQFP32/SOP28/SKDIP28	
							外部管脚也能掉电唤醒	6路15位专门的PWM(带死区控制)	2路CCP 10位PWM可当外部中断并可掉电唤醒												部分封装 价格(RMB ¥)	
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																					PDIP40	LQFP44	LQFP48	LQFP64S
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	¥5.7	¥5.2	¥5.2	¥5.4
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	¥5.9	¥5.5	¥5.5	¥5.7
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	¥5.9	¥5.6	¥5.6	¥5.8
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	¥5.9	¥5.6	¥5.6	¥5.8
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	¥5.9	¥5.6	¥5.6	¥5.8
IAP15W4K58S4 本列就是仿真器	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	¥5.9	¥5.6	¥5.6	¥5.8
用户可将用户程序区的程序Flash当EEPROM使用																						
IAP15W4K61S4 本列就是仿真器	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	¥5.9	¥5.6	¥5.6	¥5.8
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	无	否	否	¥5.9	¥5.6	¥5.6	¥5.8
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	¥3.6	¥3.7		
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	¥3.7	¥3.8		

建议用户将串口1设在[P3.6/RxD_2,P3.7/TxD_2]或[P1.6/RxD_3, P1.7/TxD_3]

IAP15W4K58S4和IAP15W4K61S4单片机可作仿真芯片

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**，以上价格还费由客户承担，零售10片起，如对价格不满，可来电要求降价

IRC15W4K63S4**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，且P3. 2/P3. 3与下载无关，且不支持"程序加密后传输"功能。**
IAP15W4K58S4**型号单片机的不能设置"P3. 2/P3. 3同时为0/0才能下载程序"，P5. 4不可当复位管脚RST使用。**

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。

宏晶•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
- 每个外部晶振的单片机，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%温飘(-40℃~+85℃)，常温下温飘±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

选择宏晶•STC15W4K32S4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4096字节SRAM
- ★ 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)
- ★ 无法解密，宏晶第九代加密技术，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：
  1. 高抗静电(ESD保护)整机轻松过2万伏静电测试
  2. 轻松过4kV快速脉冲干扰(EFT测试)
  3. 宽电压，不怕电源抖动
  4. 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：
  1. **掉电模式：外部中断唤醒功耗** <0.4uA
  2. 空闲模式：典型功耗 <1mA，
  3. 正常工作模式：4mA ~ 6mA
  4. **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

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

STC micro™
宏 晶 科 技

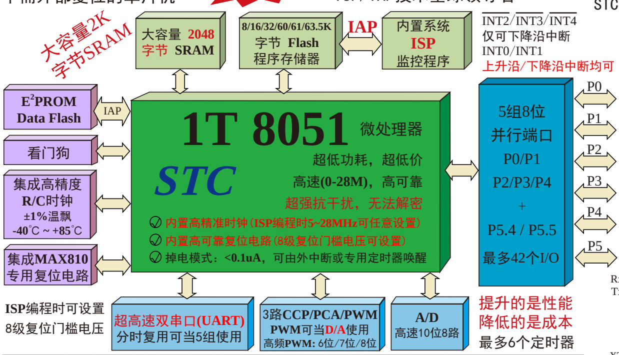
超强抗干扰，无法解密

8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

官方网站：www.STCMCU.com　www.GXWMCU.com
南通 Tel: 0513-5501 2928　5501 2929
深圳 Tel: 0755-8294 8411　8294 8412

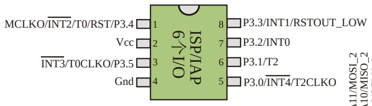
STC15F2K60S2系列1T 8051单片机，2K字节SRAM，超高速双串口，高速A/D

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



采用宏晶第八代加密技术，现悬赏10万元人民币请专家帮忙查找加密有无漏洞

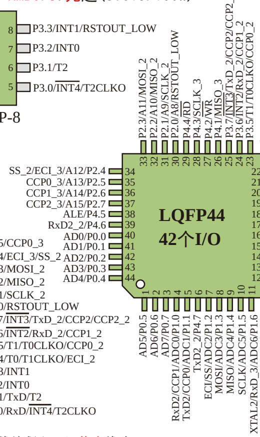
STC15F100W系列，大批量供货中，RMB0.89元起(STC15F100W)



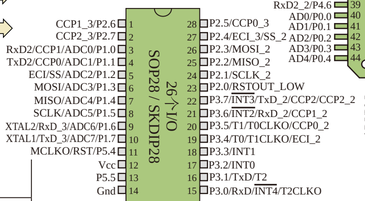
SOP-8/DIP-8

每片单片机具有全球唯一身份证号码(ID号)，无法解密，加密坚不可摧
提供客制化IC设计服务

封装形式：
LQFP44(12mm x 12mm, 强烈推荐)
PDIP40(正常生产)
LQFP32(9mm x 9mm, 推荐)
QFN32(5mm x 5mm)
SOP28(强烈推荐)
SKDIP28(正常生产)
TSSOP20(6.5mm x 6.5mm)



全部175℃
8小时高温烘烤



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 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 ¥)						
																					TSSOP20	SOP28	SKDIP28	LQFP32	QFN32	PDIP40	LQFP44
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		¥3.8	¥4.0	¥3.9		¥4.5	¥4.0
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		¥4.0	¥4.2	¥4.1		¥4.7	¥4.2
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		¥4.3	¥4.5	¥4.4		¥4.9	¥4.5
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		¥4.4	¥4.6	¥4.5		¥4.9	¥4.6
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		¥4.4	¥4.6	¥4.5		¥4.9	¥4.6
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		¥4.4	¥4.6	¥4.5		¥4.9	¥4.6
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		¥4.4	¥4.6	¥4.5		¥4.9	¥4.6
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	¥4.4	¥4.4	¥4.6	¥4.5	¥4.45	¥4.9	¥4.6
																					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						¥4.9	¥4.6
																					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						¥4.4	¥3.9
STC15F2K60S	5.5-4.5	60K	2K	1	有	3	-	有	5	-	-	1K	有	有	8级	有	是	有	是	是						¥4.4	¥3.9
IAP15F2K61S	5.5-4.5	61K	2K	1	有	3	-	有	5	–	2	IAP	有	有	8级	有	是	有	是	是						¥4.5	¥4.0
																					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							¥3.9
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							¥3.9
大批量现货供应中	STC15F100W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/$\overline{INT4}$, P3.1]结合定时器实现																				SOP8	DIP8	DFN8				
STC15F100W	5.5-3.8	0.5K	128	–	–	2(T0/T2)	–	有	5	–	1	–	有	有	8级	有	是	有	是	是	¥0.89						
STC15F101W	5.5-3.8	1K	128	–	–	2(T0/T2)	–	有	5	–	1	4K	有	有	8级	有	是	有	是	是	¥0.99	¥1.1					
STC15F102W	5.5-3.8	2K	128	–	–	2(T0/T2)	–	有	5	–	1	3K	有	有	8级	有	是	有	是	是	¥1.1	¥1.2					
STC15F104W	5.5-3.8	4K	128	–	–	2(T0/T2)	–	有	5	–	1	1K	有	有	8级	有	是	有	是	是	¥1.2	¥1.3	¥1.4				
IRC15F107W 默认使用内部24MHz时钟	5.5-3.8	7K	128	–	–	2(T0/T2)	–	有	5	–	1	IAP	有	有	固定	有	是	无	否	否	¥1.2	¥1.3					
																					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

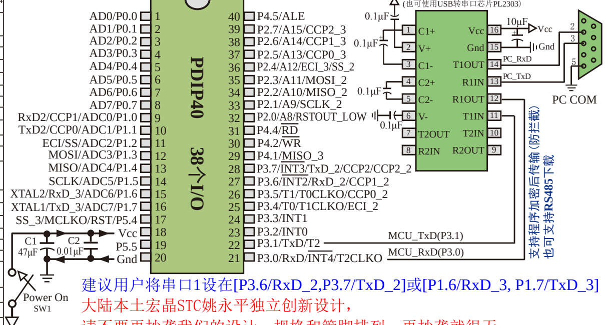
程序加密后传输：程序拥有者产品出厂时将源程序和加密钥匙一起烧录MCU中，以后需要升级软件时，就可将程序加密后再用"发布项目程序"功能，生成一个用户自己界面的只有一个升级按钮的简单易用的升级软件，给最终使用者自己升级，而拦截不到您的原始程序。

特别声明：以15F和15L开头且有SPI功能的芯片不支持"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。
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。

宏晶·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%温飘(-40℃~+85℃)，常温下温飘±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)，$\overline{INT2}$/$\overline{INT3}$/$\overline{INT4}$(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/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、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

STC 单片机在线编程/ISP及仿真线路



建议用户将串口1设在[P3.6/RxD_2,P3.7/TxD_2]或[P1.6/RxD_3, P1.7/TxD_3]
大陆本土宏晶STC姚永平独立创新设计，
请不要再抄袭我们的设计、规格和管脚排列，再抄袭就很无...

- 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，以上3个定时器/计数器输出时钟均可1~65536级分频输出；④ 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/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
- 如果I/O口不够用可用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

选择宏晶·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，宏晶第八代加密技术，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：
1. 高抗静电(ESD保护)整机轻松过2万伏静电测试　2. 轻松过4kV快速脉冲干扰(EFT测试)
3. 宽电压，不怕电源抖动　4. 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：
1. 掉电模式：外部中断唤醒功耗 <0.1uA
2. 空闲模式：典型功耗<1mA，　3. 正常工作模式：4mA～6mA
4. 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

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

STC micro™ 宏晶科技

超强抗干扰，无法解密

8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

官方网站：www.STCMCU.com www.GXWMCU.com

南通 Tel: 0513-5501 2928 5501 2929
深圳 Tel: 0755-8294 8411 8294 8412

STC15W1K16S 系列 1T 8051 单片机，宽电压，超高速串行口，比较器

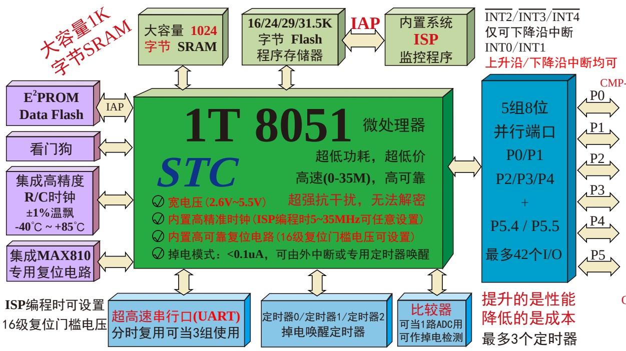
不需外部晶振的单片机
不需外部复位的单片机

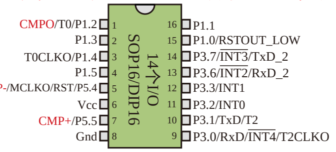
送仿真器

全球第一款真正意义上的单片机
ISP/IAP技术全球领导者

采用宏晶第九代加密技术，现悬赏20万元人民币
请专家帮忙查找STC15W201S系列单片机加密有无漏洞

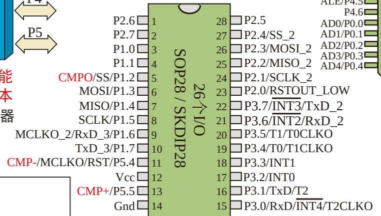
封装形式：
LQFP44(12mm x 12mm, 强烈推荐)
PDIP40(正常生产)
LQFP32(9mm x 9mm, 推荐)
QFN32(5mm x 5mm)
SOP28(强烈推荐)
SKDIP28(正常生产)
TSSOP20(6.5mm x 6.5mm)
SOP16(6mm x 9.9mm, 强烈推荐)
DIP16(正常生产)

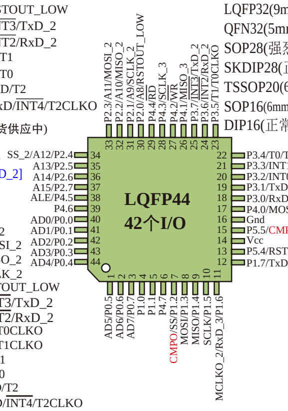




STC15W201S系列(大批量现货供应中)
抗干扰最强的Flash型单片机
RMB1.15元起(STC15W201S)

建议用户将串口1设在[P3.6/RxD_2,P3.7/TxD_2]或[P1.6/RxD_3, P1.7/TxD_3]





全部175℃ 8小时高温烘烤

STC15W201S系列是目前全球抗干扰最强的Flash型单片机

现悬赏20万元人民币寻找比STC15W201S系列抗干扰更强的Flash型单片机

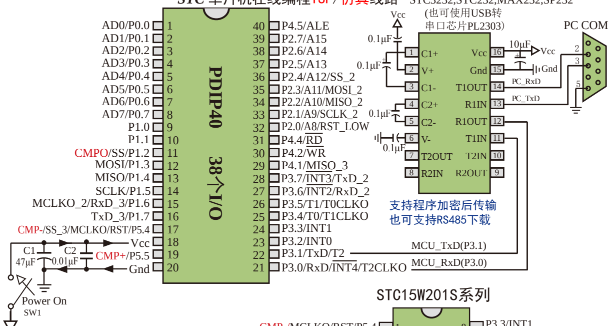
型号	工作电压（V）	Flash程序存储器(byte)	大容量SRAM字节	串行口并可掉电唤醒	SPI	普通定时器计数器T0-T2外部管脚也能掉电唤醒	CCP PCA PWM可当外部中断并可掉电唤醒	掉电唤醒专用定时器	标准外部中断支持掉电唤醒	A/D (3路PWM可当3路D/A使用)	比较器(可当1路ADC使用，可作外部掉电检测)	EEPROM	内部低压检测中断并可掉电唤醒	看门狗	内部高可靠复位(可选复位门槛电压)	内部高精准时钟	可对外输出时钟和复位	程序加密后传输(防拦截)	可设下次更新程序需口令	支持RS485下载	所有封装 LQFP44/PDIP40 LQFP32/QFN32 SOP28/SKDIP28 TSSOP20 SOP16/DIP16 封装价格(RMB ¥)						
STC15W201S系列单片机选型价格一览表，大批量现货供应中																					SOP8	SOP16	DIP16				
STC15W201S	2.5-5.5	1K	256	1	–	T0 / T2	–	有	5	–	有	4K	有	有	16级	有	是	有	是	是	¥1.15	¥1.2	¥1.35				
STC15W202S	2.5-5.5	2K	256	1	–	T0 / T2	–	有	5	–	有	3K	有	有	16级	有	是	有	是	是	¥1.25	¥1.3	¥1.45				
STC15W203S	2.5-5.5	3K	256	1	–	T0 / T2	–	有	5	–	有	2K	有	有	16级	有	是	有	是	是	¥1.35	¥1.4	¥1.55				
STC15W204S	2.5-5.5	4K	256	1	–	T0 / T2	–	有	5	–	有	1K	有	有	16级	有	是	有	是	是	¥1.35	¥1.4	¥1.55				
IAP15W205S	2.5-5.5	5K	256	1	–	T0 / T2	–	有	5	–	有	IAP	有	有	16级	有	是	有	是	是	¥1.35	¥1.4	¥1.55				
																					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	2.5-5.5	7.5K	256	1	–	T0 / T2	–	有	5	–	有	IAP	有	有	固定	有	是	无	否	否	¥1.35	¥1.4	¥1.55				
																					用户可将用户程序区的程序Flash当EEPROM使用						
STC15W404S系列单片机选型价格一览表，大批量现货供应中																						SOP28	SKDIP28	LQFP32	QFN32	PDIP40	LQFP44
STC15W404S	2.5-5.5	4K	512	1	有	3	–	有	5	–	有	9K	有	有	16级	有	是	有	是	是		¥2.3	¥2.5	¥2.4		¥3.0	¥2.5
STC15W408S	2.5-5.5	8K	512	1	有	3	–	有	5	–	有	5K	有	有	16级	有	是	有	是	是		¥2.5	¥2.7	¥2.6		¥3.0	¥2.7
STC15W410S	2.5-5.5	10K	512	1	有	3	–	有	5	–	有	3K	有	有	16级	有	是	有									¥2.8
IAP15W413S	2.5-5.5	13K	512	1	有	3	–	有	5	–	有	IAP	有	有	16级	有	是	有	是	是		¥2.6	¥2.8	¥2.7	¥2.65	¥3.3	¥2.8
																					用户可将用户程序区的程序Flash当EEPROM使用						
IRC15W415S 默认使用内部24MHz时钟	2.5-5.5	15.5K	512	1	有	3	–	有	5	–	有	IAP	有	有	固定	有	是	无	否	否		¥2.6	¥2.8	¥2.7		¥3.3	¥2.8
																					用户可将用户程序区的程序Flash当EEPROM使用						
STC15W1K16S系列单片机选型价格一览表，大批量现货供应中																					TSSOP20	SOP28	SKDIP28	LQFP32	QFN32	PDIP40	LQFP44
STC15W1K16S	2.6-5.5	16K	1K	1	有	3	–	有	5	–	有	13K	有	有	16级	有	是	有		是		¥3.3	¥3.5	¥3.4		¥4.0	¥3.5
STC15W1K24S	2.6-5.5	24K	1K	1	有	3	–	有	5	–	有	5K	有	有	16级	有	是	有		是	¥3.6	¥3.6	¥3.8	¥3.7		¥4.3	¥3.8
IAP15W1K29S	2.6-5.5	29K	1K	1	有	3	–	有	5	–	有	IAP	有	有	16级	有	是	有		是	¥3.6	¥3.6	¥3.8	¥3.7	¥3.65	¥4.3	¥3.8
																					用户可将用户程序区的程序Flash当EEPROM使用						
IRC15W1K31S 默认使用内部24MHz时钟	2.6-5.5	31.5K	1K	1	有	3	–	有	5	–	有	IAP	有	有	固定	有	是	无		否		¥3.6	¥3.8	¥3.7		¥4.3	¥3.8
																					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

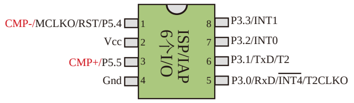
程序加密后传输：程序拥有者产品出厂时将源程序和加密钥匙一起烧录MCU中，以后需要升级软件时，就可将程序加密后再用"发布项目程序"功能，生成一个用户自己界面的只有一个升级按钮的简单易用的升级软件，给最终使用者自己升级，而拦截不到您的原始程序。

上表中以IRC开头的单片机默认使用内部24MHz时钟，其内部复位门槛电压固定，P5.4不可当复位管脚RST使用，P3.2/P3.3与下载无关，且不支持"程序加密后传输"功能。

STC 单片机在线编程ISP/仿真线路



STC15W201S系列



SOP-8

每片单片机具有
全球唯一身份证号码(ID号)，
无法解密，加密坚不可摧

提供客制化IC设计服务

宏晶·STC15W1K16S系列主要性能：

大陆本土宏晶STC姚永平独立创新设计，请不要再抄袭我们的设计、规格和管脚排列，再抄袭就很无...

- 大容量1024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：2.6V ～ 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%温飘(-40℃~+85℃)，常温下温飘±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)；
  RxD/T0/T1/T2管脚；内部掉电唤醒专用定时器
- 16/24/29K/31.5K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
- 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）
- 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(42/38/30/26/18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
- 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

选择宏晶·STC15W1K16S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1024字节SRAM
★ 无法解密，宏晶第九代加密技术，
现悬赏20万元人民币请专家帮忙查找STC15W201S系列和STC15W1K16S系列加密有无漏洞
★ 超强抗干扰：
1. 高抗静电(ESD保护)整机轻松过2万伏静电测试
2. 轻松过4kV快速脉冲干扰(EFT测试)
3. 宽电压，不怕电源抖动
4. 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：
1. 掉电模式:外部中断唤醒功耗 <0.1uA
2. 空闲模式：典型功耗<1mA,
3. 正常工作模式:4mA ~ 6mA
4. 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

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

STC micro™ 宏晶科技

超强抗干扰，无法解密

8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

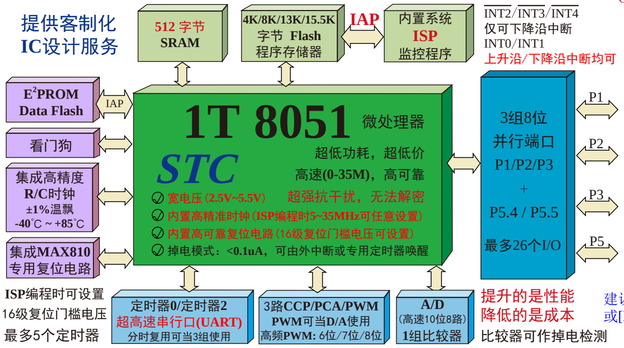
官方网站：www.STCMCU.com www.GXWMCU.com
南通 Tel: 0513-5501 2928 5501 2929
深圳 Tel: 0755-8294 8411 8294 8412

宏晶•STC15W401AS 系列 1T 8051 单片机，超高速串口，高速A/D，比较器，宽电压

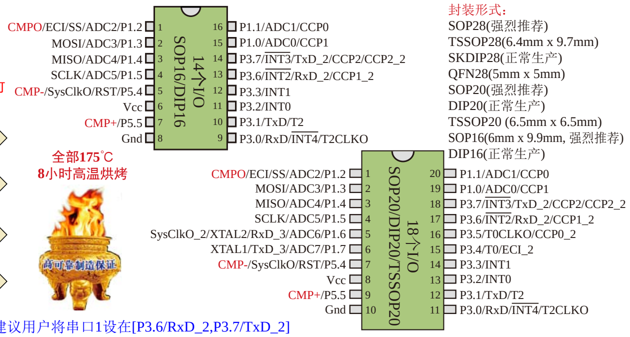
不需外部晶振的单片机
不需外部复位的单片机
送仿真器
全球第一款真正意义上的单片机 ISP/IAP技术全球领导者

采用宏晶第九代加密技术，现悬赏20万元人民币请专家帮忙查找加密有无漏洞

提供客制化IC设计服务



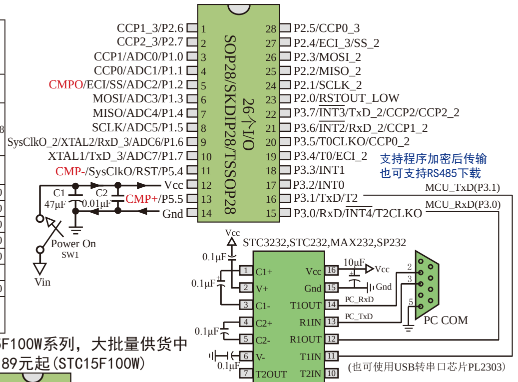
全部175℃ 8小时高温烘烤



封装形式：
SOP28(强烈推荐)
TSSOP28(6.4mm x 9.7mm)
SKDIP28(正常生产)
QFN28(5mm x 5mm)
SOP20(强烈推荐)
DIP20(正常生产)
TSSOP20 (6.5mm x 6.5mm)
SOP16(6mm x 9.9mm, 强烈推荐)
DIP16(正常生产)

建议用户将串口1设在[P3.6/RxD_2, P3.7/TxD_2]
或[P1.6/RxD_3, P1.7/TxD_3]

STC 单片机在线编程/**ISP及仿真**线路(**USB**转串口)



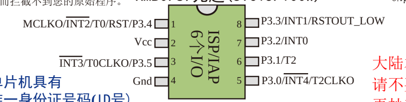
型号	工作电压(V)	Flash程序存储器(byte)	SRAM字节	串行口并可掉电唤醒	SPI	普通定时器计数器T0/T2外部管脚也能掉电唤醒	CCP PCA PWM并可掉电唤醒	掉电唤醒专用定时器	标准外部中断支持掉电唤醒	A/D 8路(3路PWM可当3路D/A使用)	比较器(可当1路ADC使用，可作掉电检测)	DPTR	EEPROM	内部低压检测中断并可掉电唤醒	看门狗	内部高可靠复位(可选复位门槛电压)	内部高精准时钟	可对外输出时钟及复位	程序加密后传输(防拦截)	可设下次更新程序需口令	支持RS485下载	SOP16	DIP16	SOP20	DIP20	TSSOP20	SOP28	SKDIP28	TSSOP28	QFN28
																						所有封装 SOP28/ SKDIP28 / TSSOP28 / QFN28 SOP20 / DIP20 / TSSOP20 SOP16 / DIP16 部分封装 价格(RMB ¥)								
STC15W401AS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	¥1.60	¥1.80	¥1.80	¥2.00	¥1.90	¥2.00	¥2.20	¥2.10	¥2.00
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	¥1.80	¥2.00	¥2.00	¥2.20	¥2.10	¥2.20	¥2.40	¥2.30	¥2.20
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	¥1.99	¥2.20	¥2.20	¥2.40	¥2.30	¥2.50	¥2.70	¥2.60	¥2.50
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	¥2.20	¥2.40	¥2.40	¥2.60	¥2.50	¥2.70	¥2.90	¥2.80	¥2.70
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	¥2.30	¥2.50	¥2.50	¥2.70	¥2.60	¥2.80	¥3.00	¥2.90	¥2.80
																						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	无	否	否	¥2.30	¥2.50	¥2.50	¥2.70	¥2.60	¥2.80	¥3.00	¥2.90	¥2.80
																						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。

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 。

STC15F100W系列，大批量供货中
RMB0.89元起(STC15F100W)



SOP-8/DIP-8

大陆本土宏晶STC姚永平独立创新设计，
请不要再抄袭我们的设计、规格和管脚排列，
再抄袭就很无...

每片单片机具有
全球唯一身份证号码(ID号)
无法解密，加密坚不可摧

宏晶•STC15W401AS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%温飘(-40℃~+85℃)，常温下温飘±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

选择宏晶•STC15W401AS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，宏晶第九代加密技术，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：
1. 高抗静电(ESD保护)整机轻松过2万伏静电测试
2. 轻松过4kV快速脉冲干扰(EFT测试)
3. 宽电压，不怕电源抖动
4. 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：
1. 掉电模式：外部中断唤醒功耗 <0.1uA
2. 空闲模式：典型功耗<1mA，
3. 正常工作模式：4mA～6mA
4. 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

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

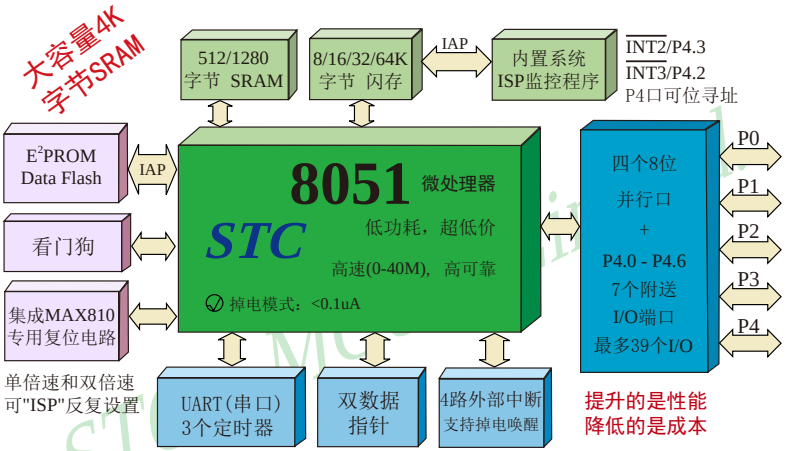
目录

第1章 STC89系列单片机总体介绍

1.1 STC89C51RC/RD+系列单片机简介

STC89C51RC/RD+系列单片机是STC推出的新一代高速/低功耗/超强抗干扰的单片机，指令代码完全兼容传统8051单片机，12时钟/机器周期和6时钟/机器周期可以任意选择，HD版本和90C版本内部集成MAX810专用复位电路。

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



1. 增强型8051单片机，6时钟/机器周期和12时钟/机器周期可任意选择，指令代码完全兼容传统8051
2. 工作电压：5.5V - 3.3V（5V单片机） / 3.8V - 2.0V（3V单片机）
3. 工作频率范围：0～40MHz，相当于普通8051的 0～80MHz，实际工作频率可达48MHz.
4. 用户应用程序空间：4K / 8K / 13K / 16K / 32K / 64K字节
5. 片上集成1280字节或512字节RAM
6. 通用I/O口(35/39个)，复位后为：P1/P2/P3/P4是准双向口/弱上拉(普通8051传统I/O口)；P0口是开漏输出，作为总线扩展用时，不用加上拉电阻，作为I/O口用时，需加上拉电阻。
7. ISP（在系统可编程）/ IAP（在应用可编程），无需专用编程器，无需专用仿真器
   可通过串口（RxD/P3.0, TxD/P3.1）直接下载用户程序，数秒即可完成一片
8. 有EEPROM功能
9. 看门狗
10. 内部集成MAX810专用复位电路(HD版本和90C版本才有)，外部晶体20M以下时，可省外部复位电路。

11. 共3个16位定时器/计数器，其中定时器0还可以当成2个8位定时器使用。

12. 外部中断4路, 下降沿中断或低电平触发中断，Power Down模式可由外部中断低电平触发中断方式唤醒。

13. 通用异步串行口(UART)，还可用定时器软件实现多个UART

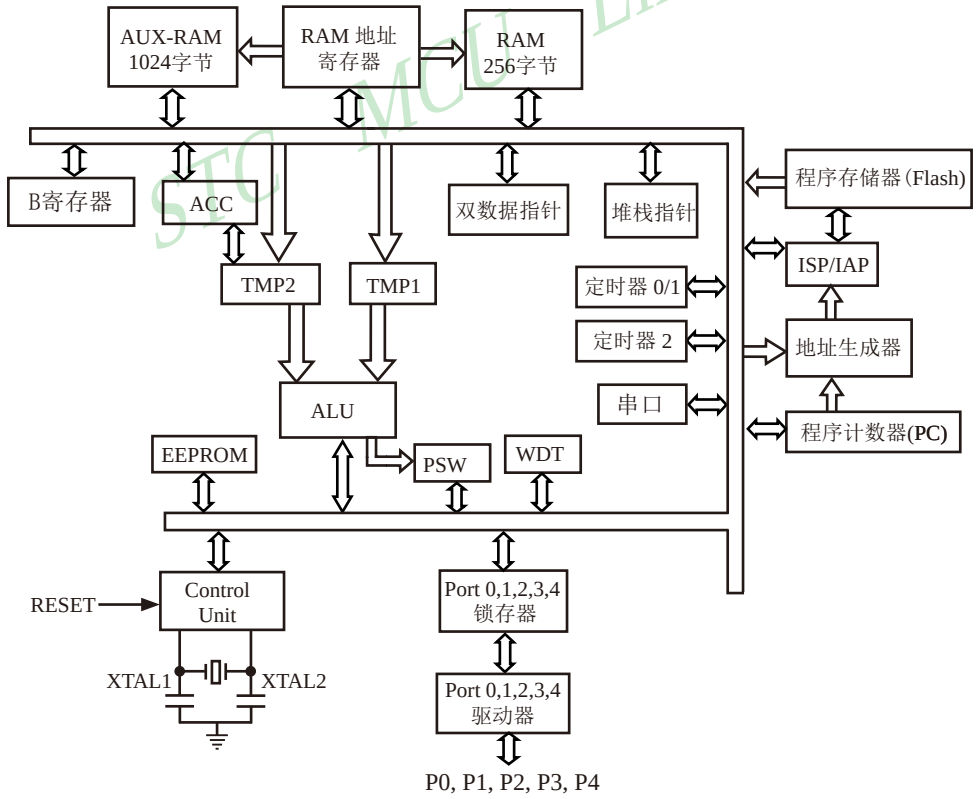
14. 工作温度范围：-40 ~ +85℃(工业级) / 0 ~ 75℃(商业级)

15. 封装：LQFP-44,PDIP-40,PLCC-44,PQFP-44. 如选择STC89系列, 请优先选择LQFP-44封装.

温馨提示：推荐优先选择采用最新第六代加密技术的STC11/10xx系列单片机取代全球各厂家均已被解密的89系列单片机.

1.2 STC89C51RC/RD+系列单片机的内部结构

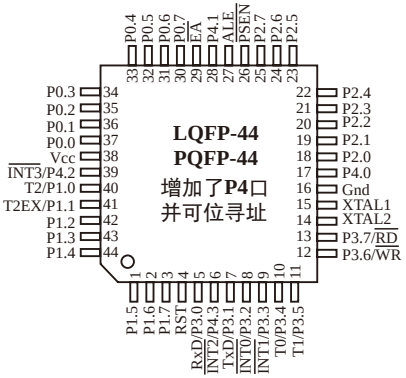
STC89C51RC/RD+系列单片机的内部结构框图如下图所示。STC89C51RC/RD+单片机中包含中央处理器(CPU)、程序存储器(Flash)、数据存储器(SRAM)、定时/计数器、UART串口、I/O接口、EEPROM、看门狗等模块。STC89C51RC/RD+系列单片机几乎包含了数据采集和控制中所需的所有单元模块，可称得上一个片上系统。

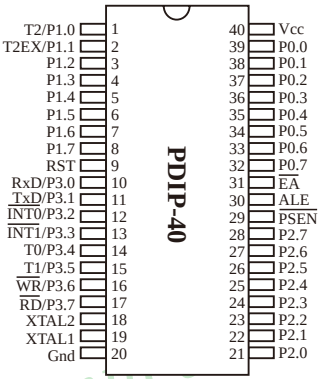


STC89C51RC/RD+系列内部结构框图

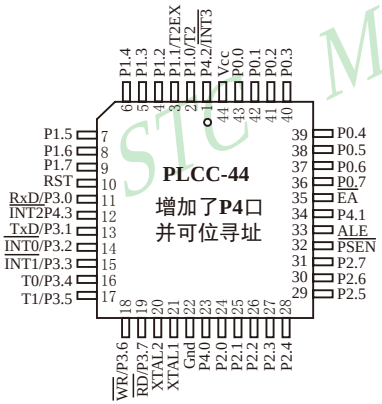
1.3 STC89C51RC/RD+系列单片机管脚图

1.3.1 STC89C51RC/RD+系列HD版本的管脚图





HD版本无P4.6/P4.5/P4.4口



关于编译器/汇编器:

1. 任何老的编译器/汇编器均可使用Keil C51中:Device选择标准的Intel8052头文件包含标准的〈reg52.h〉
2. 新增特殊功能寄存器如要用到,则用“sfr”及“sbit”声明地址即可
3. 汇编中用“data”,或“EQU”声明地址

关于仿真及仿真器:

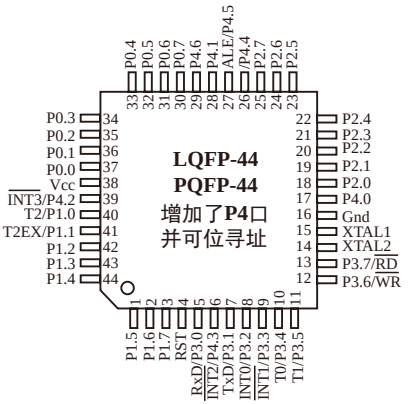
1. 任何老的仿真器均可使用
2. 老的仿真器仿真他可仿真的基本功能
3. 新增特殊功能用ISP直接下载程序看结果即可
5. 其实现在大部分STC用户不用仿真器,用ISP就可调通64K程序

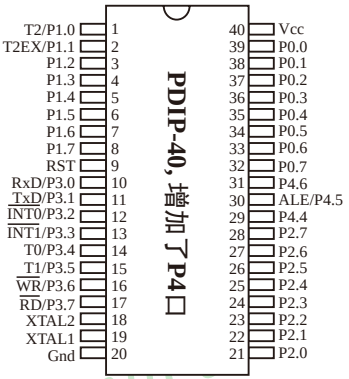
关于工作电压/时钟频率:RC/RD+系列是真正的6T单片机,兼容普通的12 时钟/机器周期

内核实际6T	现有HD版本5V单片机,单倍速工作将外部时钟频率除以2,降频工作					
工作电压	外部时钟	单倍速相当于普通8052	实际内核运行时钟	双倍速相当于普通8052	实际内核运行时钟	IAP/ISP可以
5.5V - 4.5V	0 - 44MHz	0 - 44MHz	0 - 20MHz	0 - 80MHz	0 - 40MHz	读,编程,擦除
5.5V - 3.8V	0 - 33MHz	0 - 33MHz	0 - 16.5MHz	0 - 66MHz	0 - 33MHz	读,编程,擦除
5.5V - 3.6V	0 - 24MHz	0 - 24MHz	0 - 12MHz	0 - 48MHz	0 - 24MHz	读,编程,擦除
5.5V - 3.4V	0 - 20MHz	0 - 20MHz	0 - 10MHz	0 - 40MHz	0 - 20MHz	读(不要编程/擦除)

3V: 3.8 - 2.0V(可外部24MHz,双倍速48MHz),2.3 - 1.9V时不要进行IAP擦除/编程

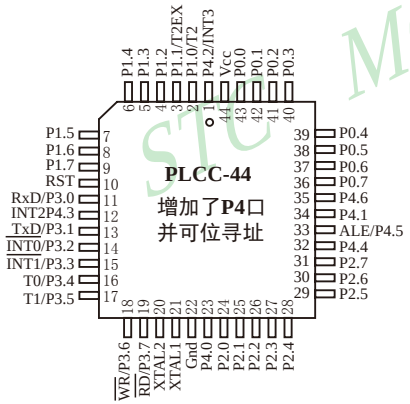
1.3.2 STC89C51RC/RD+系列90C版本的管脚图





90C版本无EA、PSEN管脚,有P4.4/P4.5/P4.6口

90C版本ALE/P4.5管脚默认是作为ALE管脚，如需作为P4.5口使用时，需在烧录用户程序时在STC-ISP编程器中设置



关于编译器/汇编器：

1. 任何老的编译器/汇编器均可使用Keil C51中：Device选择标准的Intel8052头文件包含标准的〈reg52.h〉
2. 新增特殊功能寄存器如要用到，则用“sfr”及“sbit”声明地址即可
3. 汇编中用“data”,或“EQU”声明地址

关于仿真及仿真器：

1. 任何老的仿真器均可使用
2. 老的仿真器仿真他可仿真的基本功能
3. 新增特殊功能用ISP直接下载程序看结果即可
5. 其实现在大部分STC用户不用仿真器,用ISP就可调通64K程序

1.4 STC89C51RC/RD+系列单片机选型一览表

型号	工作电压（V）	最高时钟频率(Hz)		Flash程序存储器(字节)	SRAM字节	定时器	UART串口	DPTR	EEP ROM(字节)	看门狗	A/D	中断源	中断优先级	最多I/O数量	支持掉电唤醒外部中断	内置复位	所有封装 LQFP44 / PDIP40 PLCC44 / PQFP44 封装价格(RMB ¥)			
		5V	3V														LQFP44	PDIP40	PLCC44	PQFP44
STC89C/LE51RC系列单片机选型一览																				
STC89C51RC	5.5 - 3.5	0-80M		4K	512	3	1个	2	9K	有	-	8	4	39	4个	有	¥2.8	¥3.3	¥3.4	
STC89LE51RC	3.6 - 2.2		0-80M	4K	512	3	1个	2	9K	有	-	8	4	39	4个	有	¥2.8		¥3.4	
STC15W404S 不需要外部时钟 不需要外部复位	5.5 - 2.4	5-35M		4K	512	3	1个	2	9K	强	-	12	2	42	5个	强	¥2.5	¥3.0		
STC89C52RC	5.5 - 3.5	0-80M		8K	512	3	1个	2	5K	有	-	8	4	39	4个	有	¥2.8	¥3.1	¥3.4	
STC89LE52RC	3.6 - 2.2		0-80M	8K	512	3	1个	2	5K	有	-	8	4	39	4个	有	¥2.8		¥3.4	¥3.4
STC15W408S 不需要外部时钟 不需要外部复位	5.5 - 2.4	5-35M		8K	512	3	1个	2	5K	强	-	12	2	42	5个	强	¥2.7	¥3.0		
STC89C53RC	5.5 - 3.5	0-80M		12K	512	3	1个	2	-	有	-	8	4	39	4个	有	¥3.5	¥4.0	¥4.1	¥4.1
STC89LE53RC	3.6 - 2.2		0-80M	12K	512	3	1个	2	-	有	-	8	4	39	4个	有	¥3.5		¥4.1	¥4.1
IAP15W413S 不需要外部时钟 不需要外部复位	5.5 - 2.4	5-35M		13K	512	3	1个	2	IAP	强	-	12	2	42	5个	强	¥2.8	¥3.3		
STC89C/LE51RD+系列单片机选型一览																				
STC89C54RD+	5.5 - 3.3	0-80M		16K	1280	3	1个	2	45K	有	-	8	4	39	4个	有	¥4.5	¥5.0	¥5.1	¥5.1
STC89LE54RD+	3.6 - 2.2		0-80M	16K	1280	3	1个	2	45K	有	-	8	4	39	4个	有	¥4.5		¥5.1	¥5.1
STC15W1K16S 不需要外部时钟 不需要外部复位	5.5 - 2.6	5-35M		16K	1024	3	1个	2	13K	强	-	12	2	42	5个	强	¥3.5	¥4.0		
STC89C58RD+	5.5 - 3.3	0-80M		32K	1280	3	1个	2	29K	有	-	8	4	39	4个	有	¥4.5	¥5.0	¥5.1	¥5.1
STC89LE58RD+	3.6 - 2.2		0-80M	32K	1280	3	1个	2	29K	有	-	8	4	39	4个	有	¥4.5		¥5.1	¥5.1
IAP15W1K29S 不需要外部时钟 不需要外部复位	5.5 - 2.6	5-35M		29K	1024	3	1个	2	IAP	强	-	12	2	42	5个	强	¥3.8	¥4.3		
STC89C516RD+	5.5 - 3.3	0-80M		61K	1280	3	1个	2	-	有	-	8	4	39	4个	有	¥4.5	¥5.0	¥5.1	¥5.1
IAP15F2K61S 不需要外部时钟 不需要外部复位	5.5 - 3.8	5-35M		61K	2048	3	1个	2	IAP	强	-	12	2	42	5个	强	¥4.0	¥4.5		
STC89LE516RD+	3.6 - 2.2		0-80M	61K	1280	3	1个	2	-	有	-	8	4	39	4个	有	¥4.5		¥5.1	¥5.1
IAP15L2K61S 不需要外部时钟 不需要外部复位	3.6 - 2.4		5-35M	61K	2048	3	1个	2	IAP	强	-	12	2	42	5个	强	¥4.0			

STC89C51RC/RD+系列单片机44-pin的封装不推荐使用PLCC44和PFQP44封装，建议选用LQFP44的封装。

选用STC单片机的理由：降低成本，提升性能，原有程序直接使用,硬件无需改动。STC公司鼓励您放心大胆选用LQFP44小型封装单片机,使您的产品更小，更轻,功耗更低。

建议：

用"不需外部时钟/外部复位"的"IAP15F2K61S,RMB: LQFP44/RMB4.0,PDIP40/RMB4.5"取代89C54/89C58/89C516,成本更低

用"不需外部时钟/外部复位"的"STC15W1K16S/STC15W1K24S/IAP15W1K29S,RMB: LQFP44 3.5/3.8/3.8"取代89C54/89C58,成本更低

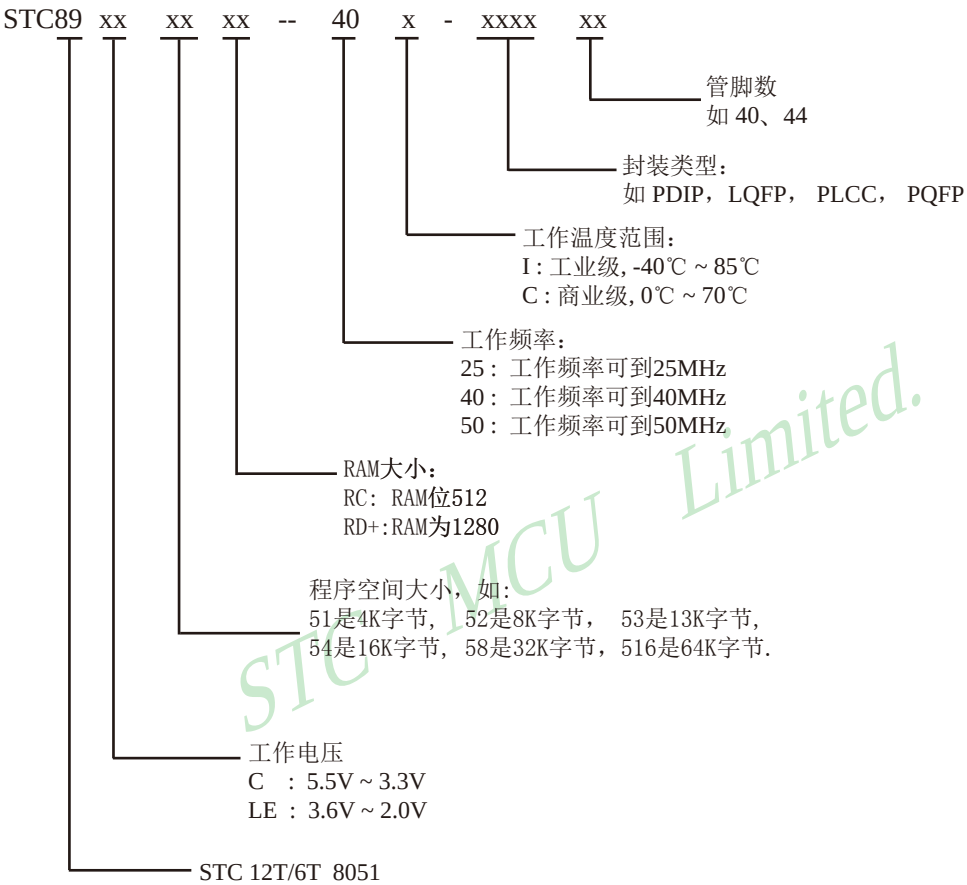
用"不需外部时钟/外部复位"的"STC15W404S/STC15W408S/IAP15W413S,RMB: LQFP44 2.5/2.7/2.8"取代89C51/89C52/89C53,成本更低

RC/RD+系列为真正的看门狗，缺省为关闭(冷启动),启动后无法关闭，可放心省去外部看门狗.

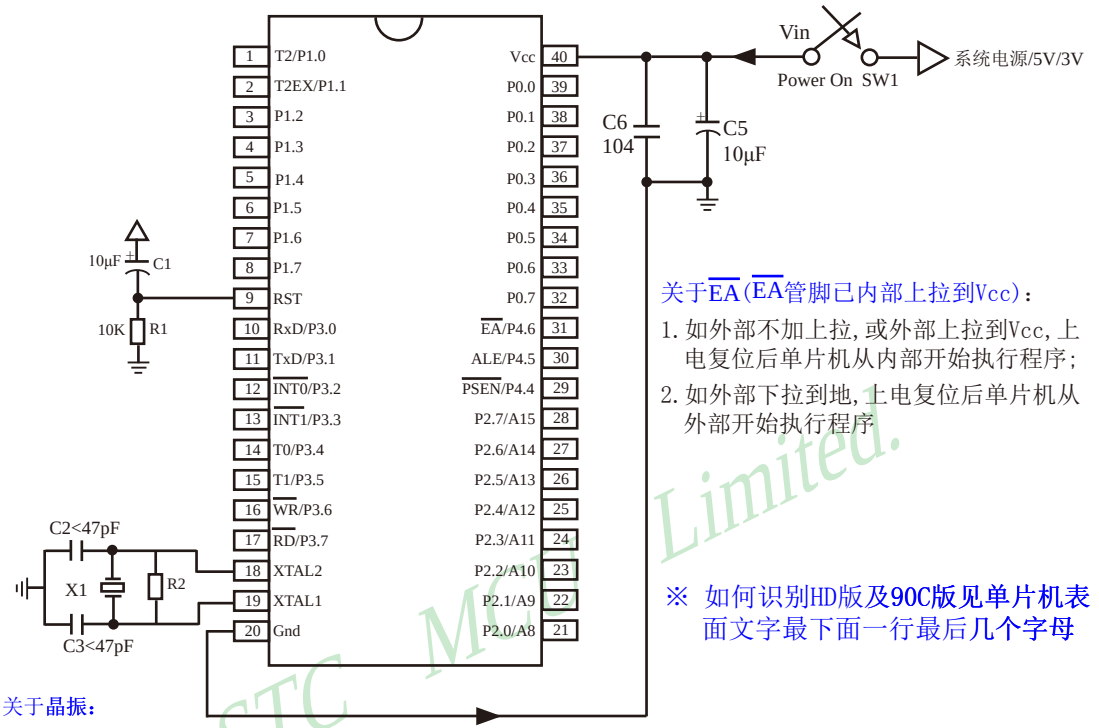
内部Flash擦写次数为10万次以上。

STC89系列的低功耗模式中，仅支持掉电模式，空闲模式不支持(不要用)

1.5 STC89C51RC/RD+系列单片机命名规则



1.6 STC89C51RC/RD+系列单片机最小应用系统



关于$\overline{EA}$($\overline{EA}$管脚已内部上拉到Vcc)：

1. 如外部不加上拉,或外部上拉到Vcc,上电复位后单片机从内部开始执行程序;
2. 如外部下拉到地,上电复位后单片机从外部开始执行程序

※ 如何识别HD版及90C版见单片机表面文字最下面一行最后几个字母

关于晶振：

晶振频率X1为4MHz时，C2、C3应为100pF；晶振频率X1为6MHz时，C2、C3应为47pF~100pF；
晶振频率X1为12M~25MHz时，C2、C3应为47pF

1. 阻容复位时,电容C1为10uF,电阻R1为10K
2. RC/RD+系列单片机HD版本，RESET脚内部已有45K-100K下拉电阻

关于晶振电路：

OSCDN,晶体振荡器增益控制 = full gain								
X1	4MHz	6MHz	12M-25MHz	26M-30MHz	31M-35MHz	36M-39MHz	40M-43MHz	44M-48MHz
C2, C3	= 100pF	47pF~100pF	= 47pF	<= 10pF	<= 10pF	<= 10pF	<= 10pF	<= 5pF
R1	不用	不用	不用	6.8K	5.1K	4.7K	3.3K	3.3K

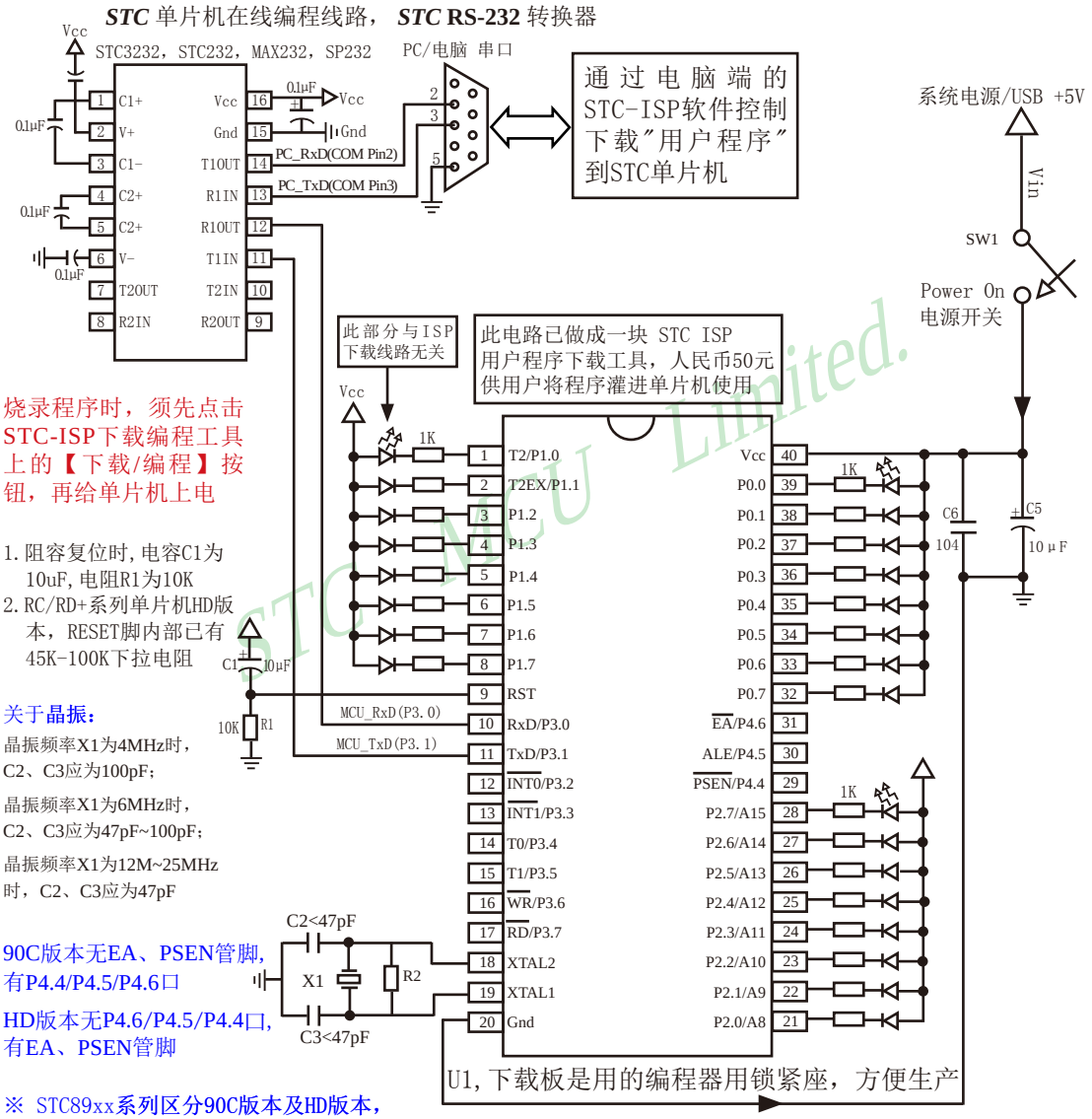
OSCDN(OSC Control),晶体振荡器增益 = 1/2 gain								
X1	4MHz	6MHz	12M-25MHz	26M-30MHz	31M-35MHz	36M-39MHz	40M- 43MHz	44M- 48MHz
C2, C3	= 100pF	47pF~100pF	= 47pF	<= 10pF	不用	不用	不用	不用
R1	不用	不用	不用	6.8K	5.1K	4.7K	3.3K	3.3K

STC89系列HD版本的单片机正常工作时的时钟频率

推荐工作时钟频率(总线) STC单片机RC/RD+系列 (I/O方式可到40M/80M)	内部振荡器产生时钟，外接晶体		外部时钟直接输入，　由XTAL1输入	
	12T模式	6T模式	12T模式	6T模式
5.0V单片机	2MHz - 48MHz	2MHz - 36MHz	2MHz - 48MHz	2MHz - 36MHz
3.3V单片机	2MHz - 48MHz	2MHz - 32MHz	2MHz - 36MHz	2MHz - 18MHz

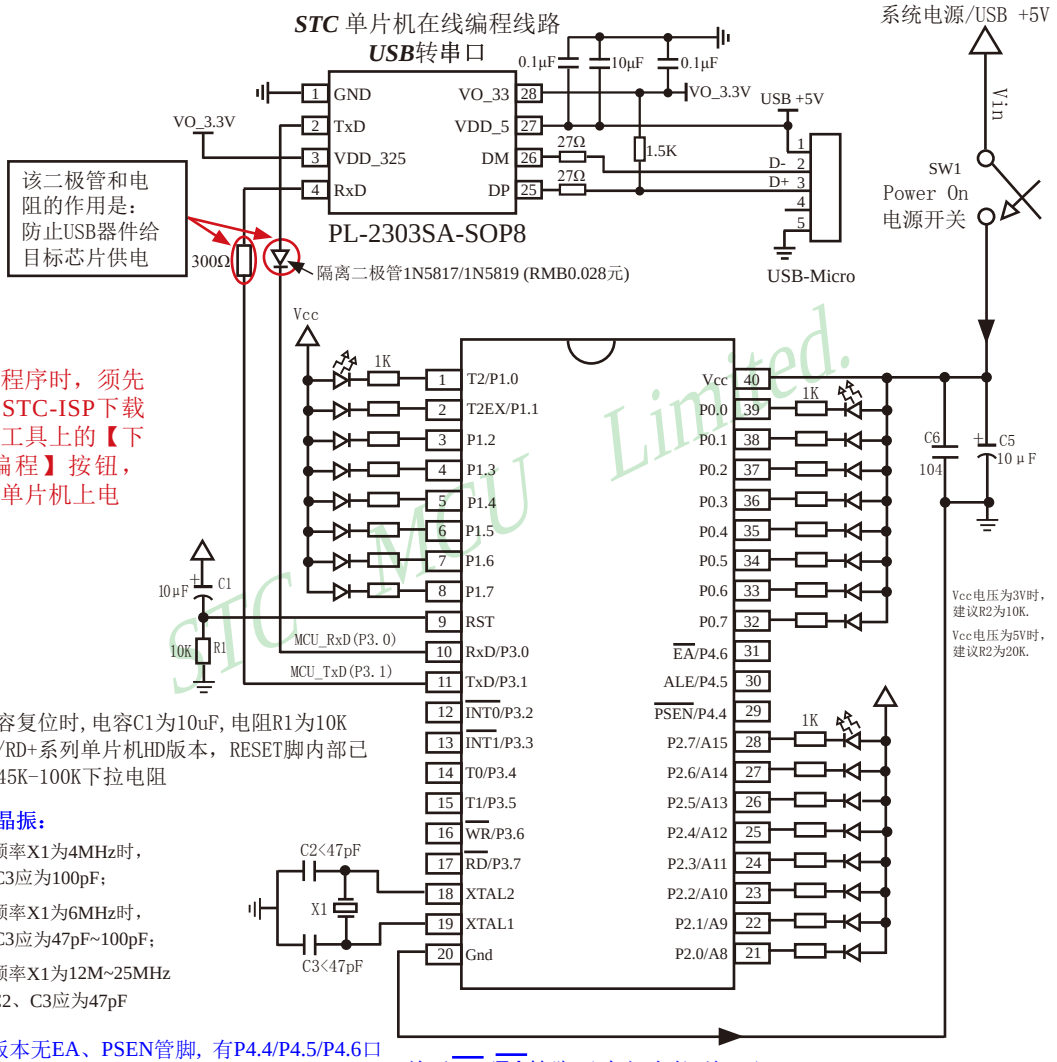
1.7 STC89C51RC/RD+系列在系统可编程(ISP)典型应用线路图

1.7.1 利用RS-232转换器的典型应用线路图



1.7.2 利用USB转串口芯片PL-2303SA的ISP下载编程典型应用线路图

建议选用USB转串口芯片PL-2303HXD-SSOP28 / PL-2303HX-SSOP28 / PL-2303SA-SOP8，国产兼容的PL-2303HX-SSOP28价格便宜，PL-2303HXD-SSOP28 / PL-2303SA-SOP8无需外部晶振，电路简单



烧录程序时，须先点击STC-ISP下载编程工具上的【下载/编程】按钮，再给单片机上电

1. 阻容复位时，电容C1为10uF，电阻R1为10K
2. RC/RD+系列单片机HD版本，RESET脚内部已有45K-100K下拉电阻

关于晶振：

晶振频率X1为4MHz时，C2、C3应为100pF；

晶振频率X1为6MHz时，C2、C3应为47pF~100pF；

晶振频率X1为12M~25MHz时，C2、C3应为47pF

90C版本无EA、PSEN管脚，有P4.4/P4.5/P4.6口

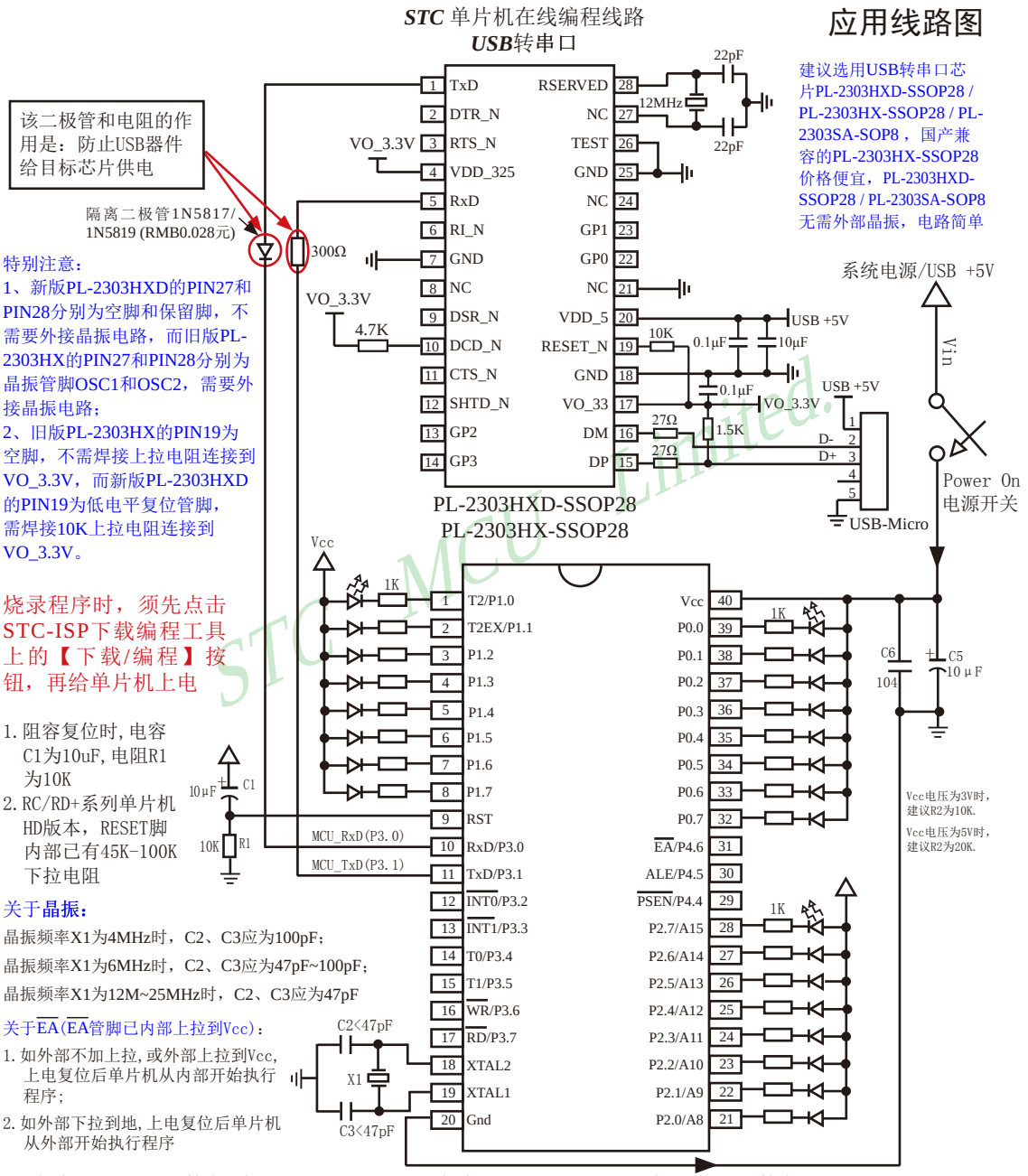
HD版本无P4.6/P4.5/P4.4口，有EA、PSEN管脚

※ STC89xx系列区分90C版本及HD版本，如何识别90C版及HD版：通过查询单片机表面文字最下面一行最后几个字母，最后几个字母为90C，则该单片机为90C版本；最后几个字母为HD，则该单片机为HD版本

关于$\overline{EA}$($\overline{EA}$管脚已内部上拉到Vcc)：

1. 如外部不加上拉，或外部上拉到Vcc，上电复位后单片机从内部开始执行程序；
2. 如外部下拉到地，上电复位后单片机从外部开始执行程序

1.7.3 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



建议选用USB转串口芯片PL-2303HXD-SSOP28 / PL-2303HX-SSOP28 / PL-2303SA-SOP8，国产兼容的PL-2303HX-SSOP28价格便宜，PL-2303HXD-SSOP28 / PL-2303SA-SOP8无需外部晶振，电路简单

特别注意：

1、新版PL-2303HXD的PIN27和PIN28分别为空脚和保留脚，不需要外接晶振电路，而旧版PL-2303HX的PIN27和PIN28分别为晶振管脚OSC1和OSC2，需要外接晶振电路；

2、旧版PL-2303HX的PIN19为空脚，不需焊接上拉电阻连接到VO_3.3V，而新版PL-2303HXD的PIN19为低电平复位管脚，需焊接10K上拉电阻连接到VO_3.3V。

烧录程序时，须先点击STC-ISP下载编程工具上的【下载/编程】按钮，再给单片机上电

1. 阻容复位时，电容C1为10uF，电阻R1为10K
2. RC/RD+系列单片机HD版本，RESET脚内部已有45K-100K下拉电阻

关于晶振：

晶振频率X1为4MHz时，C2、C3应为100pF；

晶振频率X1为6MHz时，C2、C3应为47pF~100pF；

晶振频率X1为12M~25MHz时，C2、C3应为47pF

关于$\overline{EA}$($\overline{EA}$管脚已内部上拉到Vcc)：

1. 如外部不加上拉，或外部上拉到Vcc，上电复位后单片机从内部开始执行程序；
2. 如外部下拉到地，上电复位后单片机从外部开始执行程序

90C版本无EA、PSEN管脚，有P4.4/P4.5/P4.6口；HD版本无P4.6/P4.5/P4.4口，有EA、PSEN管脚

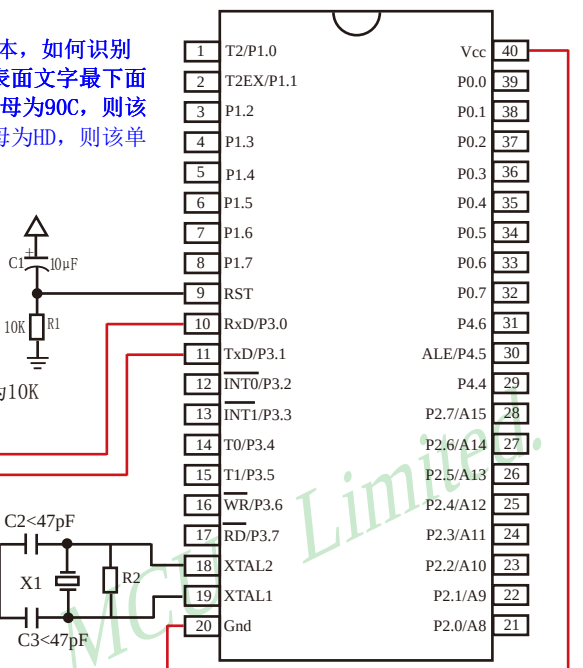
※ STC89xx系列区分90C版本及HD版本，如何识别90C版及HD版：通过查询单片机表面文字最下面一行最后几个字母，最后几个字母为90C，则该单片机为90C版本；最后几个字母为HD，则该单片机为HD版本

1.7.4 利用U8-Mini进行ISP下载的示意图

※ STC89xx系列区分90C版本及HD版本，如何识别90C版及HD版：**通过查询单片机表面文字最下面一行最后几个字母，最后几个字母为90C，则该**单片机为90C版本；最后几个字母为HD，则该单片机为HD版本

90C版本无EA、PSEN管脚，有P4.4/P4.5/P4.6口

HD版本无P4.6/P4.5/P4.4口，有EA、PSEN管脚



关于复位电路：

1. 阻容复位时，电容C1为10uF，电阻R1为10K
2. RESET脚内部已有45K-100K下拉电阻

关于晶振：

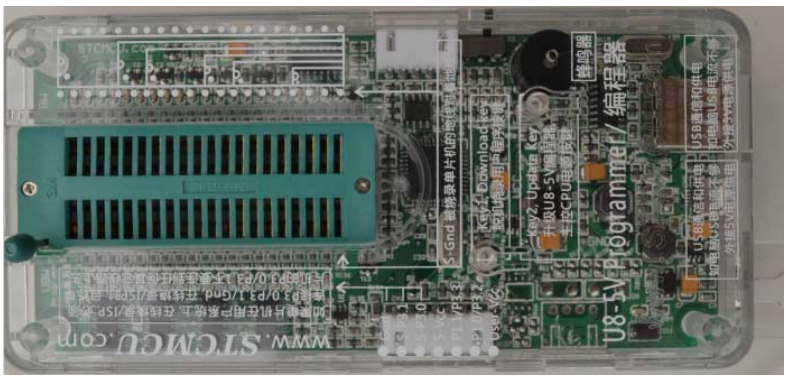
晶振频率X1为4MHz时，C2、C3应为100pF；

晶振频率X1为6MHz时，C2、C3应为47pF~100pF；

晶振频率X1为12M~25MHz时，C2、C3应为47pF

电脑/PC ⟺

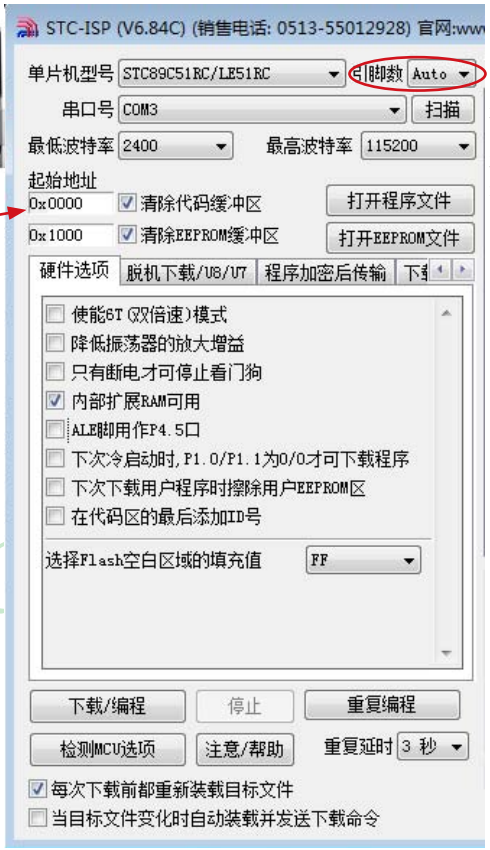
如用户需要将单片机插在锁紧座上进行ISP下载，可用下载工具U8（U8具有锁紧座，除此之外其余功能模块均与U8-Mini相同），U8的实物图如下所示：



在批量下载时，U8还可支持自动烧录机接口

1.7.5 利用U8进行ISP下载的示意图

在批量下载时，U8还可支持自动烧录机接口



ISP下载时，注意选择相应型号单片机的引脚数

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

1.8 STC89C51RC/RD+系列管脚说明

管脚	管脚编号			说明	
	LQFP44	PDIP40	PLCC44		
P0.0 ~ P0.7	37-30	39-32	43~36	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是一个8位准双向口，上电复位后处于开漏模式。P0口内部无上拉电阻，所以作I/O口必须外接10K-4.7K的上拉电阻。当P0作为地址/数据复用总线使用时，是低8位地址线[A0~A7]，数据线的[D0~D7]，此时无需外接上拉电阻。	
P1.0/T2	40	1	2	P1.0	标准I/O口 PORT1[0]
				T2	定时器/计数器2的外部输入
P1.1/T2EX	41	2	3	P1.1	标准I/O口 PORT1[1]
				T2EX	定时器/计数器2捕捉/重装方式的触发控制
P1.2	42	3	4	标准I/O口 PORT1[2]	
P1.3	43	4	5	标准I/O口 PORT1[3]	
P1.4	44	5	6	标准I/O口 PORT1[4]	
P1.5	1	6	7	标准I/O口 PORT1[5]	
P1.6	2	7	8	标准I/O口 PORT1[6]	
P1.7	3	8	9	标准I/O口 PORT1[7]	
P2.0 ~ P2.7	18-25	21-28	24~31	Port2: P2口内部有上拉电阻，既可作为输入/输出口，也可作为高8位地址总线使用(A8 ~ A15)。当P2口作为输入/输出口时，P2是一个8位准双向口。	
P3.0/RxD	5	10	11	P3.0	标准I/O口 PORT3[0]
				RxD	串口1数据接收端
P3.1/TxD	7	11	13	P3.1	标准I/O口 PORT3[1]
				TxD	串口1数据发送端
P3.2/$\overline{\text{INT0}}$	8	12	14	P3.2	标准I/O口 PORT3[2]
				$\overline{\text{INT0}}$	外部中断0, 下降沿中断或低电平中断
P3.3/$\overline{\text{INT1}}$	9	13	15	P3.3	标准I/O口 PORT3[3]
				$\overline{\text{INT1}}$	外部中断1, 下降沿中断或低电平中断
P3.4/T0	10	14	16	P3.4	标准I/O口 PORT3[4]
				T0	定时器/计数器0的外部输入
P3.5/T1	11	15	17	P3.5	标准I/O口 PORT3[5]
				T1	定时器/计数器1的外部输入
P3.6/$\overline{\text{WR}}$	12	16	18	P3.6	标准I/O口 PORT3[6]
				$\overline{\text{WR}}$	外部数据存储器写脉冲
P3.7/$\overline{\text{RD}}$	13	17	19	P3.7	标准I/O口 PORT3[7]
				$\overline{\text{RD}}$	外部数据存储器读脉冲

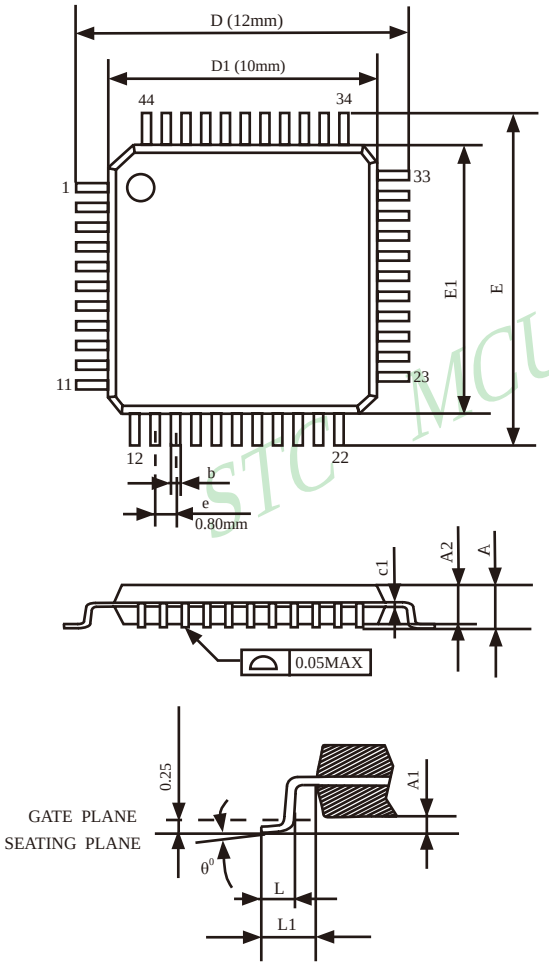
管脚	管脚编号			说明	
	LQFP44	PDIP40	PLCC44		
P4.0	17		23	P4.0	标准I/O口 PORT4[0]
P4.1	28		34	P4.1	标准I/O口 PORT4[1]
P4.2/$\overline{\text{INT3}}$	39		1	P4.2	标准I/O口 PORT4[2]
				$\overline{\text{INT3}}$	外部中断3,下降沿中断或低电平中断
P4.3/$\overline{\text{INT2}}$	6		12	P4.3	标准I/O口 PORT4[3]
				$\overline{\text{INT3}}$	外部中断2,下降沿中断或低电平中断
P4.4/$\overline{\text{PSEN}}$	26	29	32	P4.4	标准I/O口 PORT4[4]
				$\overline{\text{PSEN}}$	外部程序存储器选通信号输出引脚
P4.5/ALE	27	30	33	P4.5	标准I/O口 PORT4[5]
				ALE	地址锁存允许信号输出引脚/编程脉冲输入引脚
P4.6/$\overline{\text{EA}}$	29	31	35	P4.6	标准I/O口 PORT4[6]
				$\overline{\text{EA}}$	内外存储器选择引脚
RST	4	9	10	RST	复位脚
XTAL1	15	19	21	内部时钟电路反相放大器输入端，接外部晶振的一个引脚。当直接使用外部时钟源时，此引脚是外部时钟源的输入端。	
XTAL2	14	18	20	内部时钟电路反相放大器的输出端，接外部晶振的另一端。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。	
VCC	38	40	44	电源正极	
Gnd	16	20	22	电源负极，接地	

1.9 STC89C51RC/RD+系列单片机封装尺寸图

所有封装形式均满足欧盟RoHS要求

LQFP-44 封装尺寸图

LQFP-44 OUTLINE PACKAGE

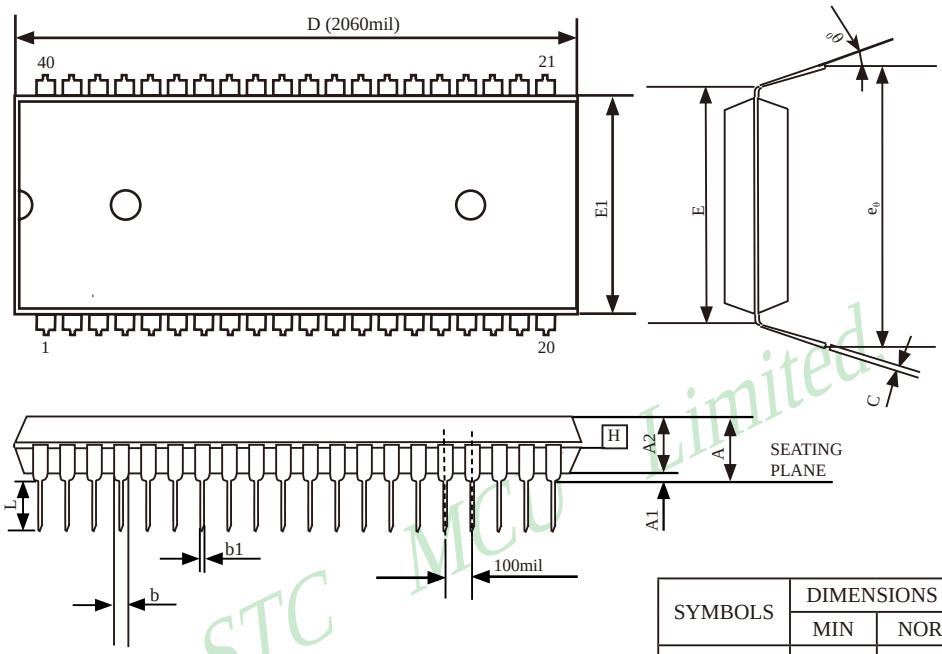


VARIATIONS (ALL DIMENSIONS SHOWN IN MM

SYMBOLS	MIN.	NOM	MAX.
A	-	-	1.60
A1	0.05	-	0.15
A2	1.35	1.40	1.45
c1	0.09	-	0.16
D		12.00	
D1		10.00	
E		12.00	
E1		10.00	
e		0.80	
△1 b(w/o plating)	0.25	0.30	0.35
L	0.45	0.60	0.75
L1		1.00REF	
θ^0	0^0	3.5^0	7^0

PDIP-40 封装尺寸图

PDIP-40 OUTLINE PACKAGE

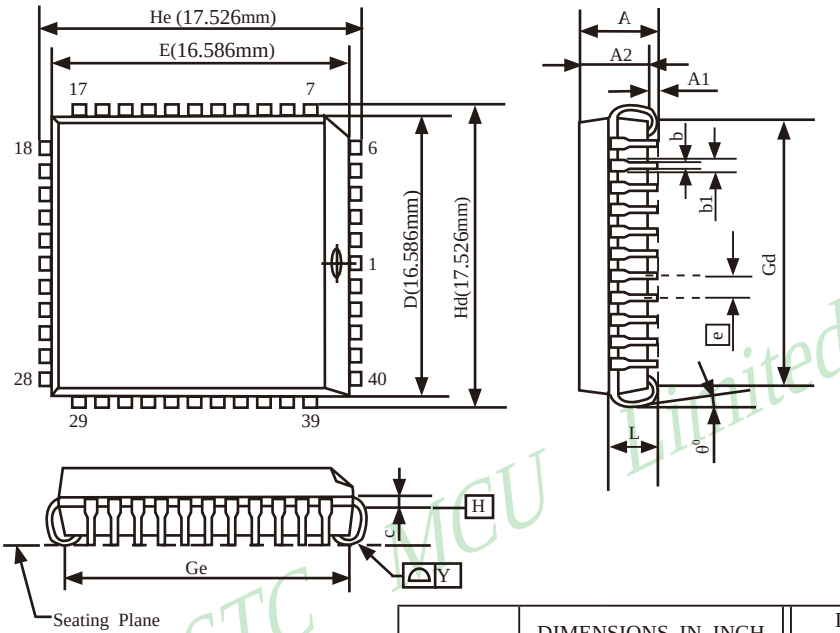


SYMBOLS	DIMENSIONS IN INCH		
	MIN	NOR	MAX
A	-	-	0.190
A1	0.015	-	0.020
A2	0.15	0.155	0.160
C	0.008	-	0.015
D	2.025	2.060	2.070
E	0.600 BSC		
E1	0.540	0.545	0.550
L	0.120	0.130	0.140
b1	0.015	-	0.021
b	0.045	-	0.067
e_θ	0.630	0.650	0.690
θ	0	7	15

UNIT: INCH　　1 inch = 1000mil

PLCC-44 封装尺寸图

PLCC-44 OUTLINE PACKAGE

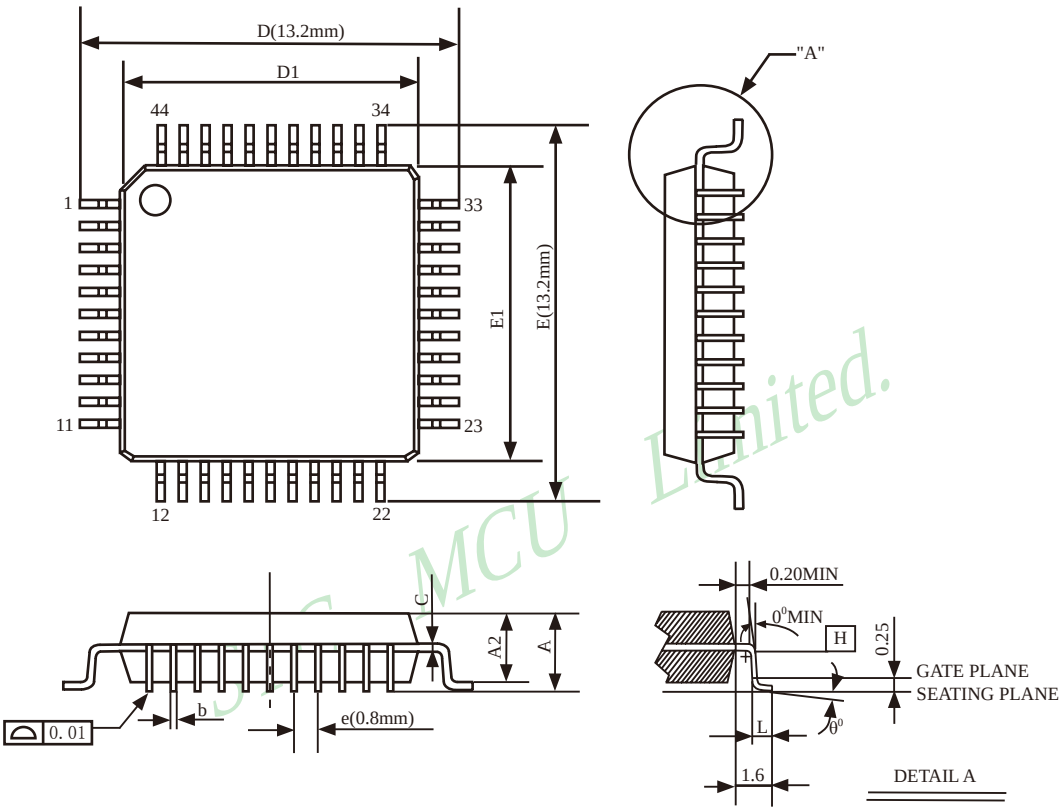


SYMBOLS	DIMENSIONS IN INCH			DIMENSIONS IN MILLMETERS		
	MIN	NOM	MAX	MIN	NOM	MAX
A	0.165	-	0.180	4.191	-	4.572
A1	0.020	-	-	0.508	-	-
A2	0.147	-	0.158	3.734	-	4.013
b1	0.026	0.028	0.032	0.660	0.711	0.813
b	0.013	0.017	0.021	0.330	0.432	0.533
c	0.007	0.010	0.0013	0.178	0.254	0.330
D	0.650	0.653	0.656	16.510	16.586	16.662
E	0.650	0.653	0.656	16.510	16.586	16.662
e	0.050BSC			1.270BSC		
Gd	0.590	0.610	0.630	14.986	15.494	16.002
Ge	0.590	0.610	0.630	14.986	15.494	16.002
Hd	0.685	0.690	0.695	17.399	17.526	17.653
He	0.685	0.690	0.695	17.399	17.526	17.653
L	0.100	-	0.112	2.540	-	2.845
Y	-	-	0.004	-	-	0.102

1 inch = 1000 mil

PQFP-44 封装尺寸图

PQFP-44 OUTLINE PACKAGE



SYMBOLS	MIN.	NOM	MAX.
A	–	–	2.70
A1	0.25	–	0.50
A2	1.80	2.00	2.20
b(w/o plating)	0.25	0.30	0.35
D	13.00	13.20	13.40
D1	9.9	10.00	10.10
E	13.00	13.20	13.40
E1	9.9	10.00	10.10
L	0.73	0.88	0.93
e	0.80 BSC.		
θ°	0	–	7
C	0.1	0.15	0.2

UNIT:mm

NOTES:
1.JEDEC OUTLINE:M0-108 AA-1

2.DATUM PLANE H IS LOCATED AT THE BOTTOM OF THE MOLD PARTING LINE COINCIDENT WITH WHERE THE LAED EXITS THE BODY.

3.DIMENSIONS D1 AND E1 D0 NOT INCLUDE MOLD PROTRUSION. ALLOWABLE PROTRUSION IS 0.25mm PER SIDE. DIMENSIONS D1 AND E1 D0 INCLUDE MOLD MISMATCH AND ARE DETRMINED AT DATUM PLANE H.

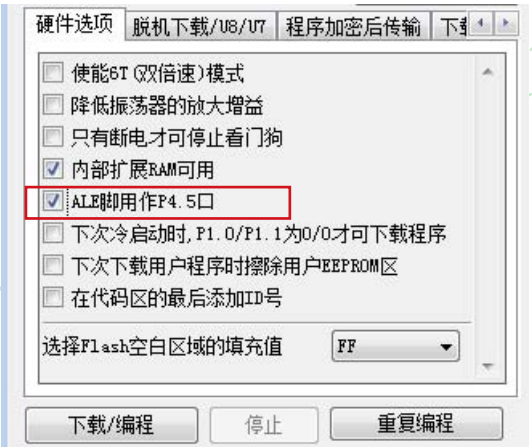
4.DIMENSION b DOES NOT INCLUDE DAMBAR PROTRUSION.

1.10 如何识别HD版及90C版本

※ 如何识别HD版及90C版见单片机表面文字最下面一行最后几个字母

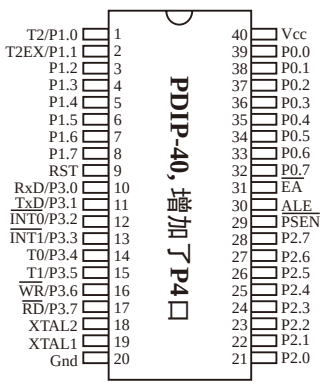
HD版本和90C版本内部都集成了MAX810专用复位电路，当时钟频率为6MHz时内部简单的MAX810专用复位电路是可靠的；当时钟频率为12MHz时勉强可用。在要求不高的情况下，可在复位脚外接电阻电容复位。

HD版有$\overline{\text{PSEN}}$、ALE及$\overline{\text{EA}}$管脚，无P4.4/P4.5/P4.6口。而90C版本有P4.4和P4.6管脚，无$\overline{\text{PSEN}}$和$\overline{\text{EA}}$。90C版本的ALE/P4.5管脚既可作I/O口P4.5使用，也可被复用作ALE管脚，默认是用作ALE管脚。如用户需用到P4.5口，只能选择90C版本的单片机，且需在烧录用户程序时在STC-ISP编程器中将ALE pin选择为用作P4.5，在烧录用户程序时在STC-ISP编程器中该管脚默认的是作为ALE pin。具体设置如下图所示：

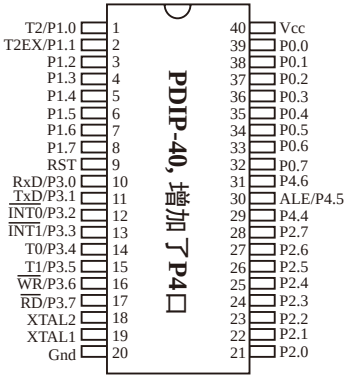


下面是STC89系列单片机HD版和90C版本的管脚图，主要区别在P4.6/P4.5/P4.4三个管脚处。

HD版本的管脚图(PDIP-40)



90C版本的管脚图(PDIP-40)



1.11 降低单片机时钟对外界的电磁辐射(EMI)——三大措施

1. 禁止ALE信号输出，适用型号:

STC89C51RC, STC89C52RC, STC89C53RC, STC89LE51RC, STC89LE52RC, STC89LE53RC
STC89C54RD+, STC89C58RD+, STC89C516RD+, STC89LE54RD+, STC89LE58RD+, STC89LE516RD+

RC/RD+系列8051单片机 扩展RAM管理及禁止ALE输出 特殊功能寄存器(只写)

AUXR : Auxiliary Register(只写)

Mnemonic	Add	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
AUXR	8EH	name	-	-	-	-	-	-	EXTRAM	ALEOFF	xxxx,xx00

禁止ALE信号输出(应用示例供参考,C 语言):

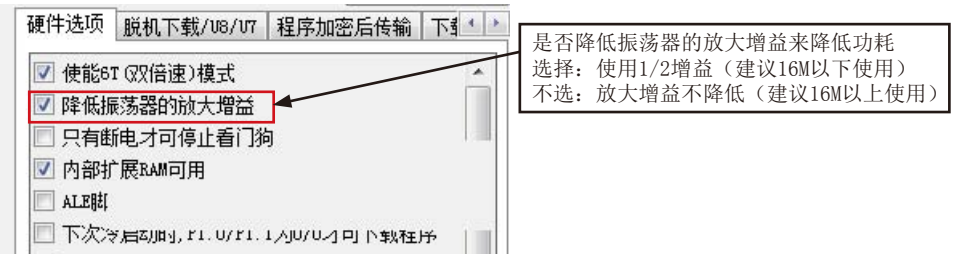
```
sfr AUXR = 0x8e;        /* 声明AUXR 寄存器的地址 */
AUXR = 0x01;
/* ALEOFF位置1,禁止ALE信号输出,提升系统的EMI性能,复位后为0,ALE信号正常输出*/
```

禁止ALE信号输出(应用示例供参考,汇编语言):

```
AUXR    EQU     8Eh             ;或AUXR     DATA    8Eh
MOV     AUXR,   #00000001B      ;ALEOFF位置“1”,禁止ALE信号输出,
                                ;提升了系统的EMI性能
```

2. 外部时钟频率降一半，6T模式：传统的8051为每个机器周期12时钟，如将STC的增强型8051单片机在ISP烧录程序时设为双倍速（及6T模式，每个机器周期6 时钟），则可将单片机外部时钟频率降低一半，有效的降低单片机时钟对外界的干扰

3. 单片机内部时钟振荡器增益降低一半：在ISP烧录程序时将OSCDN设为1/2gain可以有效的降低单片机时钟高频部分对外界的辐射,但此时外部晶振频率尽量不要高于16MHz。单片机外部晶振频率<16MHz时，可将OSCDN设为1/2 gain,有利于降低EMI，16M以上选择full gain。



1.12 超低功耗——STC89C51RC/RD+ 系列单片机

1. 掉电模式：
   典型功耗 < 0.1uA, 可由外部中断唤醒，中断返回后，继续执行原程序
2. 正常工作模式：
   典型功耗 4mA ~ 7mA
3. 掉电模式可由外部中断唤醒，适用于水表、气表等电池供电系统及便携设备

※ STC89系列的低功耗模式中，仅支持掉电模式，空闲模式不支持(不要用)

第2章 复位及省电模式

2.1 复位

STC89C51RC/RD+系列单片机有4种复位方式：外部RST引脚复位，软件复位，掉电复位/上电复位，看门狗复位。

2.1.1 外部RST引脚复位

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。将RST复位管脚拉高并维持至少24个时钟加10us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从用户程序区的0000H处开始正常工作。

2.1.2 软件复位

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ISP_CONTR特殊功能寄存器,实现了此功能。用户只需简单的控制ISP_CONTR特殊功能寄存器的其中两位SWBS/SWRST就可以系统复位了。

ISP_CONTR: ISP/IAP 控制寄存器

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
ISP_CONTR	E7H	name	ISPEN	SWBS	SWRST	-	-	WT2	WT1	WT0

ISPEN：ISP/IAP功能允许位。
　　0：禁止ISP/IAP读/写/擦除Data Flash/EEPROM；
　　1：允许ISP/IAP读/写/擦除Data Flash/EEPROM。
SWBS：软件选择从用户应用程序区启动(0)，还是从ISP程序区启动(1)。要与SWRST直接配合才可以实现
SWRST：0：不操作；1：产生软件系统复位，硬件自动清零。

```
;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 ISP_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 ISP_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 ISP_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 ISP_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

2.1.3 上电复位/掉电复位

当电源电压VCC低于上电复位/掉电复位电路的检测门槛电压时，所有的逻辑电路都会复位。当VCC重新恢复正常电压时，HD版本的单片机延迟2048个时钟(90C版本单片机延迟32768个时钟）后，上电复位/掉电复位结束。进入掉电模式时，上电复位/掉电复位功能被关闭。

2.1.4 看门狗(WDT)复位

适用型号:

STC89C51RC, STC89C52RC, STC89C53RC, STC89LE51RC, STC89LE52RC, STC89LE53RC

STC89C54RD+, STC89C58RD+, STC89C516RD+, STC89LE54RD+, STC89LE58RD+, STC89LE516RD+

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。

STC89C51RC/RD+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。为此功能，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	E1H	name	-	-	EN_WDT	CLR_WDT	IDLE_WDT	PS2	PS1	PS0

Symbol符号 Function功能

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

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

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 x Pre-scale x 32768) / Oscillator frequency

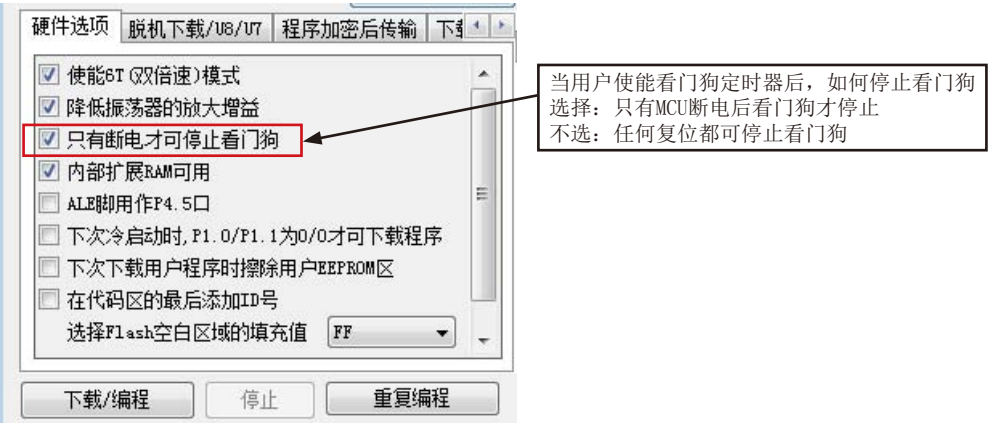
设时钟为12MHz：
看门狗溢出时间 = (12 × Pre-scale × 32768) / 12000000 = Pre-scale× 393216 / 12000000

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

设时钟为11.0592MHz：
看门狗溢出时间= (12 x Pre-scale x 32768) / 11059200 = Pre-scale x 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

当用户启动内部看门狗后，可在烧录用户程序时在STC-ISP编程器中对看门狗作如下所示的设置。



看门狗测试程序，在STC的下载板上可以直接测试

```
/*---------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ----------------------------------------*/
/* --- 演示STC89xx系列单片机 看门狗及其溢出时间计算公式-------*/
/* --- Mobile: (86)13922805190 -------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -----------*/
/*---------------------------------------------------------------------------------*/

;本演示程序在STC-ISP Ver 4.86.PCB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   0E1H        ;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Oscillator 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  MAIN
        ORG   0100H
MAIN:
        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
```

2.1.5 冷启动复位和热启动复位

	复位源	现象
热启动复位	内部看门狗复位	会使单片机直接从用户程序区0000H处开始执行用户程序
	通过控制RESET脚产生的硬复位	会使系统 从用户程序区0000H处开始直接执行用户程序
	通过对ISP_CONTR寄存器送入20H产生的软复位	会使系统从用户程序区0000H处开始直接执行用户程序
	通过对ISP_CONTR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，会软复位到用户程序区执行用户程序
冷启动复位	系统停电后再上电引起的硬复位	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，会软复位到用户程序区执行用户程序

2.2 STC89C51RC/RD+系列单片机的省电模式

——仅支持掉电模式，不支持空闲模式

STC89C51RC/RD+系列单片机可以运行2种省电模式以降低功耗，它们分别是：空闲模式和掉电模式。正常工作模式下，STC89C51RC/RD+系列单片机的典型功耗是4mA ~ 7mA，而掉电模式下的典型功耗是<0.1uA，空闲模式(建议不要使用此模式)下的典型功耗是2mA.

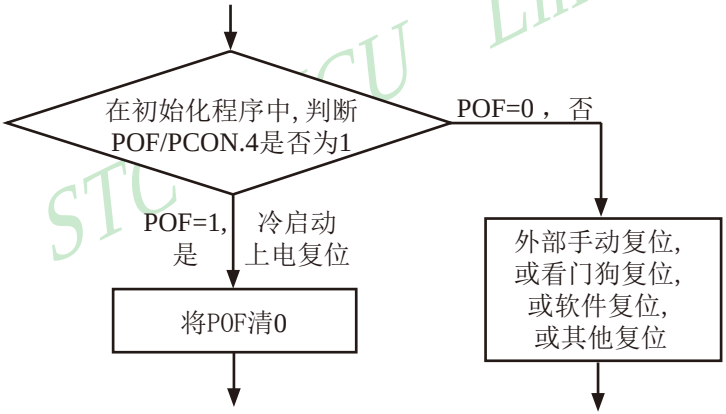
空闲模式和掉电模式的进入由电源控制寄存器PCON的相应位控制。PCON寄存器定义如下：

PCON (Power Control Register) （不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
PCON	87H	name	SMOD	SMOD0	-	POF	GF1	GF0	PD	IDL

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，可由外部中断低电平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CPU、定时器、串行口等功能部件停止工作, 只有外部中断继续工作。掉电模式可由外部中断唤醒，中断返回后，继续执行原程序。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。

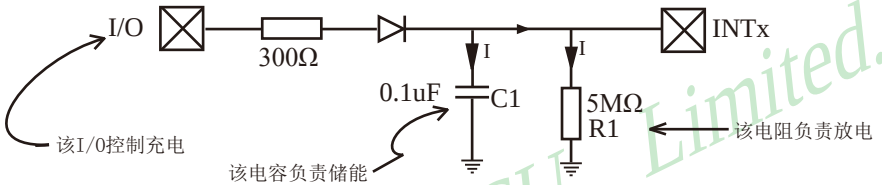
2.2.1 掉电模式/停机模式

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。进入掉电模式后，内部时钟停振，由于无时钟源，CPU、定时器、看门狗、串行口等停止工作，外部中断继续工作。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。进入掉电模式后，所有I/O口、SFRs(特殊功能寄存器)维持进入掉电模式前那一刻的状态不变。

可将CPU从掉电模式唤醒的外部管脚有:$\overline{\text{INT0}}$/P3.2, $\overline{\text{INT1}}$/P3.3,$\overline{\text{INT2}}$/P4.3, $\overline{\text{INT3}}$/P4.2

另外，外部复位也将MCU从掉电模式中唤醒，复位唤醒后的MCU将从用户程序的0000H处开始正常工作。

当用户系统无外部中断源将单片机从掉电模式唤醒时，下面的电路能够定时唤醒掉电模式。



控制充电的I/O口首先配置为推挽/强上拉模式并置高，上面的电路会给储能电容C1充电。在单片机进入掉电模式之前，将控制充电的I/O口拉低，上面电路通过电阻R1给储能电容C1放电。当电容C1的电被放到小于0.8V时，外部中断INTx会产生一个下降沿中断,从而自动地将单片机从掉电模式中唤醒。

2.2.2 掉电模式/停机模式的示例程序(C和汇编)

1. C程序:

```
/*可由外部中断唤醒的掉电唤醒示例程序 ---------------------------------- */
/*-------------------------------------------------------------------------------------------*/
/* --- STC MCU International Limited ----------------------------------------------*/
/* --- 演示STC90C58AD 系列单片机由外部中断唤醒掉电模式 ---------------------*/
/* --- Mobile: (86)13922805190 ----------------------------------------------------*/
/* --- Fax: 86-755-82905966 -------------------------------------------------------*/
/* --- Tel: 86-755-82948412 -------------------------------------------------------*/
/* --- Web: www.STCMCU.com --------------------------------------------------------*/
/*如果要在程序中使用或在文章中引用该程序 ---------------------------------- */
/*请在程序中或文章中注明使用了STC的资料及程序 -------------------------*/
/*-------------------------------------------------------------------------------------------*/
#include <reg51.h>
```

```
#include <intrins.h>
sbit        Begin_LED = P1^2;                       //Begin-LED indicator indicates system start-up
unsigned char       Is_Power_Down = 0;              //Set this bit before go into Power-down mode
sbit        Is_Power_Down_LED_INT0          = P1^7; //Power-Down wake-up LED indicator on INT0
sbit        Not_Power_Down_LED_INT0         = P1^6; //Not  Power-Down wake-up LED indicator on INT0
sbit        Is_Power_Down_LED_INT1          = P1^5; //Power-Down wake-up LED indicator on INT1
sbit        Not_Power_Down_LED_INT1         = P1^4; //Not  Power-Down wake-up LED indicator on INT1
sbit        Power_Down_Wakeup_Pin_INT0      = P3^2; //Power-Down wake-up pin on INT0
sbit        Power_Down_Wakeup_Pin_INT1      = P3^3; //Power-Down wake-up pin on INT1
sbit        Normal_Work_Flashing_LED        = P1^3; //Normal work LED indicator
void  Normal_Work_Flashing (void);
void  INT_System_init (void);
void  INT0_Routine (void);
void  INT1_Routine (void);

void main (void)
{
        unsigned char       j = 0;
        unsigned char       wakeup_counter = 0;
                                        //clear interrupt wakeup counter variable wakeup_counter
        Begin_LED = 0;                  //system start-up LED
        INT_System_init ( );            //Interrupt system initialization
        while(1)
        {
                P2 = wakeup_counter;
                wakeup_counter++;
                for(j=0; j<2; j++)
                {
                        Normal_Work_Flashing( );    //System normal work
                }
                Is_Power_Down = 1;                  //Set this bit before go into Power-down mode
                PCON    = 0x02;             //after this instruction, MCU will be in power-down mode
                                            //external clock stop
                _nop_( );
                _nop_( );
                _nop_( );
                _nop_( );
        }
}
void  INT_System_init (void)
{
        IT0     = 0;                        /* External interrupt 0, low electrical level triggered */
//      IT0     = 1;                        /* External interrupt 0, negative edge triggered */
        EX0     = 1;                        /* Enable external interrupt 0
```

```
        IT1     = 0;                    /* External interrupt 1, low electrical level triggered */
//      IT1     = 1;                    /* External interrupt 1, negative edge triggered */
        EX1     = 1;                    /* Enable external interrupt 1
        EA      = 1;                    /* Set Global Enable bit
}
void  INT0_Routine (void) interrupt 0
{
        if (Is_Power_Down)
        {
                //Is_Power_Down ==1;        /* Power-Down wakeup on INT0 */
                Is_Power_Down = 0;
                Is_Power_Down_LED_INT0 = 0;
                                /*open external interrupt 0 Power-Down wake-up LED indicator */
                while (Power_Down_Wakeup_Pin_INT0 == 0)
                {
                        /* wait higher */
                }
                Is_Power_Down_LED_INT0 = 1;
                                /* close external interrupt 0 Power-Down wake-up LED indicator */
        }
        else
        {
                Not_Power_Down_LED_INT0 = 0;    /* open external interrupt 0 normal work LED */
                while (Power_Down_Wakeup_Pin_INT0 ==0)
                {
                        /* wait higher */
                }
                Not_Power_Down_LED_INT0 = 1;    /* close external interrupt 0 normal work LED */
        }
}

void  INT1_Routine (void)   interrupt 2
{
        if (Is_Power_Down)
        {
                //Is_Power_Down ==1;        /* Power-Down wakeup on INT1 */
                Is_Power_Down = 0;
                Is_Power_Down_LED_INT1= 0;
                                /*open external interrupt 1 Power-Down wake-up LED indicator */
                while (Power_Down_Wakeup_Pin_INT1 == 0)
                {
                        /* wait higher */
                }
                Is_Power_Down_LED_INT1 = 1;
                                /* close external interrupt 1 Power-Down wake-up LED indicator */
```

```
        }
        else
        {
                Not_Power_Down_LED_INT1 = 0;      /* open external interrupt 1 normal work LED */
                while (Power_Down_Wakeup_Pin_INT1 ==0)
                {
                        /* wait higher */
                }
                Not_Power_Down_LED_INT1 = 1;      /* close external interrupt 1 normal work LED */
        }
}

void  delay (void)
{
        unsigned int          j = 0x00;
        unsigned  int         k = 0x00;
        for (k=0; k<2; ++k)
        {
                for (j=0; j<=30000; ++j)
                {
                        _nop_( );
                        _nop_( );
                        _nop_( );
                        _nop_( );
                        _nop_( );
                        _nop_( );
                        _nop_( );
                        _nop_( );
                }
        }
}

void Normal_Work_Flashing (void)
{
        Normal_Work_Flashing_LED = 0;
        delay ( );
        Normal_Work_Flashing_LED = 1;
        delay ( );
}
```

2. 汇编程序：

```
;通过外部中断从掉电模式唤醒单片机（汇编语言）
;*************************************************************
;Wake Up Idle and Wake Up Power Down
;*************************************************************
;/* --- STC MCU International Limited ----------------------------------------------*/
;/* --- 演示STC90C58AD系列单片机由外部中断唤醒掉电模式 ----------------------------*/
;/* --- Mobile: (86)13922805190 ----------------------------------------------------*/
;/* --- Fax: 86-755-82905966 -------------------------------------------------------*/
;/* --- Tel: 86-755-82948412 -------------------------------------------------------*/
;/* --- Web: www.STCMCU.com --------------------------------------------------------*/
;/*如果要在程序中使用或在文章中引用该程序 ------------------------------------- */
;/*请在程序中或文章中注明使用了STC的资料及程序 ---------------------------*/
;/*---------------------------------------------------------------------------------*/
                ORG     0000H
                AJMP    MAIN
                ORG     0003H
int0_interrupt:
                CLR     P1.7                ;open P1.7 LED indicator
                ACALL   delay               ;delay in order to observe
                CLR     EA                  ;clear global enable bit, stop all interrupts
                RETI
                ORG     0013H
int1_interrupt:
                CLR     P1.6                ;open P1.6 LED indicator
                ACALL   delay               ;;delay in order to observe
                CLR     EA                  ;clear global enable bit, stop all interrupts
                RETI
                ORG     0100H
delay:
                CLR     A
                MOV     R0,     A
                MOV     R1,     A
                MOV     R2,     #02
main:
                MOV     R3,     #0          ;P1 LED increment mode changed
                                            ;start to run program
main_loop:
                MOV     A,      R3
                CPL     A
                MOV     P1,     A
                ACALL   delay
                INC     R3
                MOV     A,      R3
                SUBB    A,      #18H
                JC      main_loop
```

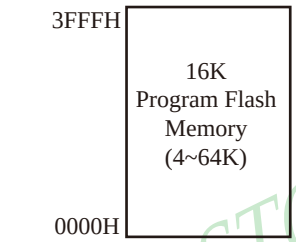
```
                MOV     P1,     #0FFH           ;close all LED, MCU go into power-down mode
                CLR     IT0                     ;low electrical level trigger external interrupt 0
        ;       SETB    IT0                     ;negative edge trigger external interrupt 0
                SETB    EX0                     ;enable external interrupt 0
                CLR     IT1                     ;low electrical level trigger external interrupt 1
        ;       SETB    IT1                     ;negative edge trigger external interrupt 1
                SETB    EX1                     ;enable external interrupt 1
                SETB    EA                      ;set the global enable
                                                ;if don't so, power-down mode cannot be wake up

;MCU will go into idle mode or power-down mode after the following instructions
                MOV     A,      PCON    ;Set PD bit, power-down mode (PD = PCON.1)
                ORL     A,      #02H
                MOV     PCON,   A
        ;       NOP
        ;       MOV     PCON,   #00000001B      ;Set IDL bit, idle mode (IDL = PCON.0)
                MOV     P1,     #0DFH           ;1101,1111
                NOP
                NOP
                NOP
WAIT1:
                SJMP    $                       ;dynamically stop
                END
```

第3章　存储器和特殊功能寄存器(SFRs)

STC89C51RC/RD+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. STC-89C51RC/RD+系列单片机除可以访问片上Flash存储器外，还可以访问64KB的外部程序存储器。STC89C54RD+系列单片机内部有1280字节的数据存储器，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(1024字节)。而STC89C51RC系列单片机内部有512字节的数据存储器，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(256字节)。另外，STC89C51RC/RD+系列单片机还可以访问在片外扩展的64KB外部数据存储器。现以STC89C54RD+系列单片机为例，分别介绍其程序存储器和数据存储器。

3.1　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。STC89C51RC/RD+系列单片机内部集成了4K~64K字节的Flash程序存储器。STC89C51RC/RD+系列各种型号单片机的片内程序Flash存储器的地址如下表所示。



STC89C54RD+单片机程序存储器

Type	Program Memory
STC89C/LE51RC	0000H~0FFFH(4K)
STC89C/LE52RC	0000H~1FFFH(8K)
STC89C/LE53RC	0000H~33FFH(13K)
STC89C/LE54RD+	0000H~3FFFH (16K)
STC89C/LE58RD+	0000H~7FFFH (32K)
STC89C/LE510RD+	0000H~9FFFH(40K)
STC89C/LE512RD+	0000H~BFFFH(48K)
STC89C/LE514RD+	0000H~DFFFH(56K)
STC89C/LE516RD+	0000H~FFFFH (64K)

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。STC89C51RC/RD+单片机利用$\overline{EA}$引脚来确定是访问片内程序存储器还是访问片外程序存储器。当$\overline{EA}$引脚接高电平时，对于STC89C51RC/RD+单片机首先访问片内程序存储器，当PC的内容超过片内程序存储器的地址范围时，系统会自动转到片外程序存储器。以STC89C54RD+单片机为例，当若$\overline{EA}$引脚接高电平，单片机首先从片内程序存储器的0000H单元开始执行程序，当PC的内容超过3FFFH时系统自动转到片外程序存储器中取指令。此时外部程序存储器的地址从4000H开始。

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。

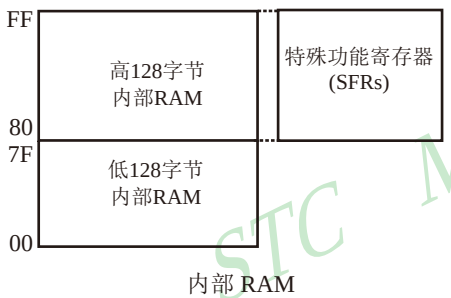
3.2 数据存储器(SRAM)

STC89C54RD+系列单片机内部集成了1280字节RAM，可用于存放程序执行的中间结果和过程数据。内部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(1024字节)。此外，STC89C51RC/RD+系列单片机还可以访问在片外扩展的64KB外部数据存储器。

3.2.1 内部RAM

内部RAM共256字节，可分为3**个部分：低128字节RAM(与传统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。



内部 RAM

低128字节的内部RAM

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(可位寻址)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
PSW	D0H	name	CY	AC	F0	RS1	RS0	OV	F1	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0。

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

F1 ： 用户标志位1。

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。STC89C51RC/RD+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。

3.2.2 内部扩展RAM(物理上是内部，逻辑上是外部，用MOVX访问)

STC89C54RD+单片机片内除了集成256字节的内部RAM外，还集成了1024字节的扩展RAM，地址范围是0000H~03FFH. 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、P2口、P3.6、P3.7和ALE。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。

STC89C51RC/RD+/AD/PWM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	-	-	-	-	-	-	EXTRAM	ALEOFF	xxxx,xx00

EXTRAM:　Internal/External RAM access　内部/外部RAM存取

0: 内部扩展的EXT_RAM可以存取.

RD+系列单片机

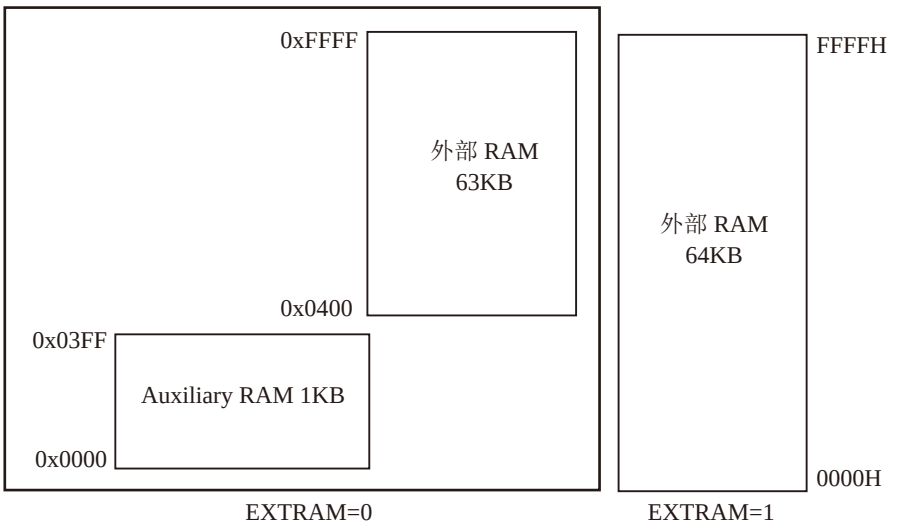
在00H到3FFH单元(1024字节),使用MOVX @DPTR指令访问,超过400H的地址空间总是访问外部数据存储器(含400H单元), MOVX @Ri只能访问00H到FFH单元

RC 系列单片机

在00H到FFH单元(256字节),使用MOVX @DPTR指令访问,超过100H的地址空间总是访问外部数据存储器(含100H单元), 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

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



ALEOFF: Disable/enable ALE.

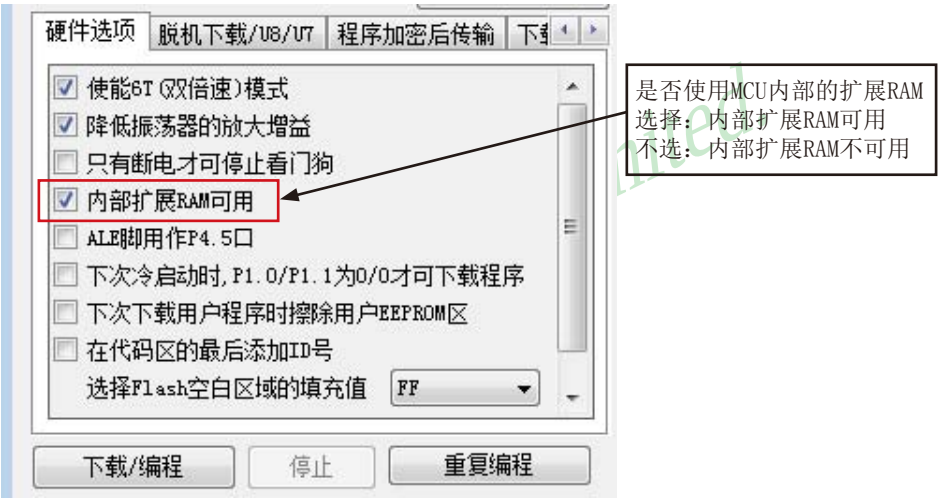
0: ALE is emitted at a constant rate of 1/3 the oscillator frequency in 6 clock mode, 1/6 fosc in 12 clock mode

ALE脚输出固定的1/6晶振频率信号在12时钟模式时,在6时钟模式时输出固定的1/3晶振频率信号.

1: ALE is active only during a MOVX or MOVC instruction.

ALE脚仅在执行MOVX or MOVC指令时才输出信号,好处是：降低了系统对外界的EMI

另外，在访问内部扩展RAM之前，用户还需在烧录用户程序时在STC-ISP编程器中设置允许内部扩展AUX-RAM访问，如下图所示



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”.
        ;MOVX   A,   @DPTR / MOVX       @DPTR,A指令可访问内部扩展的EXTRAM
                        ;RD+ 系列为(00H - 3FFH,共1024 字节)
                        ;RC 系列为(00H - FFH,共256 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 RD+ 系列 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
RD+ 系列
; 如果 #address < 400H，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 >= 400H，则总是访问物理上外部扩展的RAM 或I/O 空间（400H--FFFFH）

RC 系列
; 如果 #address < 100H，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 >= 100H，则总是访问物理上外部扩展的RAM 或I/O 空间（100H--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1024/256 字节,则可直接将此SRAM 省去,比如省去STC62WV256,IS62C256,UT6264 等. 另外尽量用 MOVX A,@Ri / MOVX @Ri, A 指令访问此内部扩展的EXTRAM,这样只能访问256字节的扩展EXTRAM,但可与很多单片机兼容。

应用示例供参考（C 语言）:

```
/* 访问内部扩展的EXTRAM */
 /* RD+ 系列为(00H - 3FFH, 共1024 字节扩展的EXTRAM) */
/* RC 系列为(00H - FFH, 共256 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];
/* RD+ 系列:
      如果 #address < 400H，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>=400H，则总是访问物理上外部扩展的RAM 或I/O空间（400H-FFFFH）

RC 系列:
      如果 #address < 100H，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>=100H，总是访问物理上外部扩展的RAM 或I/O空间（100H--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.

AUXR是只写寄存器

所谓只写，就是直接用“MOV AUXR, #data”去写, 而不要用含读的操作如“或，与，入栈”因为他不让你读，如去读，读出的数值不确定，用含读的操作如“或，与，入栈”，会达不到需要的效果。

单片机HD版本和以前版本的区别(关于内部扩展RAM)

传统的8051，内部无扩展RAM，而STC89C51RC/RD+系列单片机内部均已扩展了RAM,少数客户的老产品P0/P2是作为总线用的而不是作为普通I/O口用，有些需要用软件关闭此内部扩展RAM。而客户的源程序早已遗失，或开发工程师早已离职，所以STC89C51RC/RD+系列单片机为了解决此问题，推出HD版本以供用户在ISP下载程序时就可选择关闭此内部扩展RAM，以达到完全兼容以前的老产品的目的。

一般不要在ISP下载程序时就选择关闭此内部扩展RAM，因为流行用法是复位后缺省是允许访问扩展RAM,复位后AUXR.1/AUXR.EXTRAM = 0,选择关闭此内部扩展RAM,则本来是：

	在ISP下载程序时选择 “允许访问内部扩展RAM”	在ISP下载程序时选择 “禁止访问内部扩展RAM”
AUXR.1/AUXR.EXTRAM = 0	是允许访问内部扩展RAM	是禁止访问内部扩展RAM
AUXR.1/AUXR.EXTRAM = 1	是禁止访问内部扩展RAM	是允许访问内部扩展RAM

另STC89C51RC/RD+系列单片机C版本以前的单片机“AUXR寄存器是只写特性”,现HD版本及以后的版本将都是既可以读又可以写。

STC89C51RC/RD+系列单片机内部扩展RAM演示程序

```
;/* ---  STC International Limited ------------------------------------------ */
;/* ---  STC 姚永平  设计 2006/1/6   V1.0 ----------------------------------- */
;/* ---  演示 STC90C58AD系列单片机 MCU 内部扩展RAM演示程---------------------- */
;/* ---  Mobile: 13922805190 ------------------------------------------------ */
;/* ---  Fax: 0755-82905966 ------------------------------------------------- */
;/* ---  Tel: 0755-82948409 ------------------------------------------------- */
;/* ---  Web: www.STCMCU.com ------------------------------------------------ */
;/* ---  本演示程序在STC-ISP Ver 3.0A.PCB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 ------- */

#include <reg52.h>
#include <intrins.h>        /* use _nop_() function */

sfr AUXR = 0x8e;
sfr AUXR1 = 0xa2;

sfr P4 = 0xc0;
sfr XICON = 0xe8;

sfr IPH = 0xb7;

sfr WDT_CONTR  = 0xe1;
sfr ISP_DATA   = 0xe2;
sfr ISP_ADDRH  = 0xe3;
sfr ISP_ADDRL  = 0xe4;
sfr ISP_CMD    = 0xe5;
sfr ISP_TRIG   = 0xe6;
sfr ISP_CONTR  = 0xe7;

sbit ERROR_LED= P1^5;
sbit OK_LED    = P1^7;

void main()
{
    unsigned int array_point = 0;

    /* 测试数组 Test_array_one[2048],Test_array_two[2048]*/
    unsigned char xdata Test_array_one[2048]        =
    {
       0x00,      0x01,      0x02,      0x03,      0x04,      0x05,      0x06,      0x07,
       0x08,      0x09,      0x0a,      0x0b,      0x0c,      0x0d,      0x0e,      0x0f,
       0x10,      0x11,      0x12,      0x13,      0x14,      0x15,      0x16,      0x17,
```

```
0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f,
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, 0xa1, 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, 0xc4, 0xc5, 0xc6, 0xc7,
0xc8, 0xc9, 0xca, 0xcb, 0xcc, 0xcd, 0xce, 0xcf,
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, 0xe1, 0xe0,
0xdf, 0xde, 0xdd, 0xdc, 0xdb, 0xda, 0xd9, 0xd8,
0xd7, 0xd6, 0xd5, 0xd4, 0xd3, 0xd2, 0xd1, 0xd0,
0xcf, 0xce, 0xcd, 0xcc, 0xcb, 0xca, 0xc9, 0xc8,
0xc7, 0xc6, 0xc5, 0xc4, 0xc3, 0xc2, 0xc1, 0xc0,
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,   0x36,   0x35,   0x34,   0x33,   0x32,   0x31,   0x30,
    0x2f,   0x2e,   0x2d,   0x2c,   0x2b,   0x2a,   0x29,   0x28,
    0x27,   0x26,   0x25,   0x24,   0x23,   0x22,   0x21,   0x20,
    0x1f,   0x1e,   0x1d,   0x1c,   0x1b,   0x1a,   0x19,   0x18,
    0x17,   0x16,   0x15,   0x14,   0x13,   0x12,   0x11,   0x10,
    0x0f,   0x0e,   0x0d,   0x0c,   0x0b,   0x0a,   0x09,   0x08,
    0x07,   0x06,   0x05,   0x04,   0x03,   0x02,   0x01,   0x00
};

unsigned char xdata Test_array_two[2048]        =
{
    0x00,   0x01,   0x02,   0x03,   0x04,   0x05,   0x06,   0x07,
    0x08,   0x09,   0x0a,   0x0b,   0x0c,   0x0d,   0x0e,   0x0f,
    0x10,   0x11,   0x12,   0x13,   0x14,   0x15,   0x16,   0x17,
    0x18,   0x19,   0x1a,   0x1b,   0x1c,   0x1d,   0x1e,   0x1f,
    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,   0xa1,   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,   0xc4,   0xc5,   0xc6,   0xc7,
    0xc8,   0xc9,   0xca,   0xcb,   0xcc,   0xcd,   0xce,   0xcf,
    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,   0xe1,   0xe0,
    0xdf,   0xde,   0xdd,   0xdc,   0xdb,   0xda,   0xd9,   0xd8,
    0xd7,   0xd6,   0xd5,   0xd4,   0xd3,   0xd2,   0xd1,   0xd0,
    0xcf,   0xce,   0xcd,   0xcc,   0xcb,   0xca,   0xc9,   0xc8,
    0xc7,   0xc6,   0xc5,   0xc4,   0xc3,   0xc2,   0xc1,   0xc0,
    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,   0x36,   0x35,   0x34,   0x33,   0x32,   0x31,   0x30,
    0x2f,   0x2e,   0x2d,   0x2c,   0x2b,   0x2a,   0x29,   0x28,
    0x27,   0x26,   0x25,   0x24,   0x23,   0x22,   0x21,   0x20,
    0x1f,   0x1e,   0x1d,   0x1c,   0x1b,   0x1a,   0x19,   0x18,
    0x17,   0x16,   0x15,   0x14,   0x13,   0x12,   0x11,   0x10,
    0x0f,   0x0e,   0x0d,   0x0c,   0x0b,   0x0a,   0x09,   0x08,
    0x07,   0x06,   0x05,   0x04,   0x03,   0x02,   0x01,   0x00
};
```

```
    ERROR_LED = 1;
    OK_LED = 1;
    for(array_point=0; array_point<2048; 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);
}
```


3.2.3 可外部扩展64K Bytes(字节)数据存储器

STC89C51RC/RD+系列单片机具有扩展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，$\overline{WR}$或$\overline{RD}$信号要有效。

当MOVX指令访问物理上在内部，逻辑上在外部的片内扩展的1024字节EXTRAM时，以上设置均被忽略，以上设置只是在访问真正的片外扩展器件时有效。

助记符	功能说明	字节数	STC89C51RC/RD+系列单片机所需时钟
MOVX A,@Ri	逻辑上在外部的片内扩展RAM,(8位地址)送入累加器	1	12
MOVX A,@DPTR	逻辑上在外部的片内扩展RAM,(16位地址)送入累加器	1	12
MOVX @Ri,A	累加器送逻辑上在外部的片内扩展RAM(8位地址)	1	12
MOVX @DPTR ,A	累加器送逻辑上在外部的片内扩展RAM(16位地址)	1	12

3.3 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。STC89C51RC/RD+系列单片机内的特殊功能寄存器(SFR)与内部高128字节RAM貌似共用相同的地址范围，都使用80H~FFH, 但特殊功能寄存器(SFR)必须用直接寻址指令访问。

STC89C51RC/RD+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									0FFH
0F0H	B 0000,0000								0F7H
0E8H	P4 xxxx,1111								0EFH
0E0H	ACC 0000,0000	WDT_CONR xx00,0000	ISP_DATA 1111,1111	ISP_ADDRH 0000,0000	ISP_ADDRL 0000,0000	ISP_CMD 1111,1000	ISP_TRIG xxxx,xxxx	ISP_CONTR 000x,x000	0E7H
0D8H									0DFH
0D0H	PSW 0000,0000								0D7H
0C8H	T2CON 0000,0000	T2MOD xxxx,xx00	RCAP2L 0000,0000	RCAP2H 0000,0000	TL2 0000,0000	TH2 0000,0000			0CFH
0C0H	XICON 0000,0000								0C7H
0B8H	IP xx00,0000	SADEN 0000,0000							0BFH
0B0H	P3 1111,1111							IPH 0000,0000	0B7H
0A8H	IE 0x00,0000	SADDR 0000,0000							0AFH
0A0H	P2 1111,1111		AUXR1 xxxx,0xx0					Don't use	0A7H
098H	SCON 0000,0000	SBUF xxxx,xxxx							09FH
090H	P1 1111,1111								097H
088H	TCON 0000,0000	TMOD 0000,0000	TL0 0000,0000	TL1 0000,0000	TH0 0000,0000	TH1 0000,0000	AUXR xxxx,xx00		08FH
080H	P0 1111,1111	SP 0000,0111	DPL 0000,0000	DPH 0000,0000				PCON 00x1,0000	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

符号	描述	地址	位地址及符号 MSB LSB								复位值
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
DPTR DPH	数据指针(高)	83H									0000 0000B
PCON	电源控制寄存器	87H	SMOD	SMOD0	-	POF	GF1	GF0	PD	IDL	00x1 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/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	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	-	-	-	-	-	-	EXTRAM	ALEOFF	xxxx xx00B
P1	Port 1	90H	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	1111 1111B
SCON	串口控制寄存器	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
SBUF	串口数据缓冲器	99H									xxxx,xxxx
P2	Port 2	A0H	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0	1111 1111B
AUXR1	辅助寄存器1	A2H	-	-	-	-	GF2	-	-	DPS	xxxx 0xx0B
IE	中断允许寄存器	A8H	EA	-	ET2	ES	ET1	EX1	ET0	EX0	0x00 0000B
SADDR	从机地址控制寄存器	A9H									0000 0000B
P3	Port 3	B0H	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0	1111 1111B
IPH	中断优先级寄存器高	B7H	PX3H	PX2H	PT2H	PSH	PT1H	PX1H	PT0H	PX0H	0000 0000B
IP	中断优先级寄存器低	B8H	-	-	PT2	PS	PT1	PX1	PT0	PX0	xx00 0000B
SADEN	从机地址掩模寄存器	B9H									0000 0000B
XICON	Auxiliary Interrupt Control	C0H	PX3	EX3	IE3	IT3	PX2	EX2	IE2	IT2	0000,0000B
T2CON	Timer/Counter 2 Control	C8H	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/$\overline{T2}$	CP/$\overline{RL2}$	0000 0000B
T2MOD	Timer/Counter 2 Mode	C9H	-	-	-	-	-	-	T2OE	DCEN	xxxx xx00B
RCAP2L	Timer/Counter 2 Reload/Capture Low Byte	CAH									0000 0000B
RCAP2H	Timer/Counter 2 Reload/Capture High Byte	CBH									0000 0000B

符号	描述	地址	位地址及符号 MSB LSB								复位值
TL2	Timer/Counter Low Byte	CCH									0000 0000B
TH2	Timer/Counter High Byte	CDH									0000 0000B
PSW	程序状态字寄存器	D0H	CY	AC	F0	RS1	RS0	OV	F1	P	0000 0000B
ACC	累加器	E0H									0000 0000B
WDT_CONTR	看门狗控制寄存器	E1H	-	-	EN_WDT	CLR_WDT	IDLE_WDT	PS2	PS1	PS0	xx00 0000B
ISP_DATA	ISP/IAP 数据寄存器	E2H									1111 1111B
ISP_ADDRH	ISP/IAP 高8位地址寄存器	E3H									0000 0000B
ISP_ADDRL	ISP/IAP 低8位地址寄存器	E4H									0000 0000B
ISP_CMD	ISP/IAP 命令寄存器	E5H	-	-	-	-	-	MS2	MS1	MS0	xxxx x000B
ISP_TRIG	ISP/IAP 命令触发寄存器	E6H									xxxx xxxxB
ISP_CONTR	ISP/IAP控制寄存器	E7H	ISPEN	SWBS	SWRST	-	-	WT2	WT1	WT0	000x x000B
P4	Port 4	E8H	-	-	-	-	P4.3	P4.2	P4.1	P4.0	xxxx 1111B
B	B寄存器	F0H									0000 0000B

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	F1	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0。

RS1、RS0: 工作寄存器组的选择位。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.

F0 ： 用户标志位1。

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。STC89C51RC/RD+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。STC89C51RC/RD+系列单片机有两个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/AUXR1.0来选择具体被使用的数据指针。

STC89C51RC/RD+系列8051 单片机 双数据指针 特殊功能寄存器

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1	A2H	Auxiliary Register 1	-	-	-	-	GF2	-	-	DPS	xxxx,0xx0

DPS　　DPTR registers 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

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
    AUXR1       DATA    0A2H
    MOV         AUXR1,  #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         AUXR1                   ; 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         AUXR1                   ; 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         AUXR1                   ; 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         AUXR1                   ; 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.
```

第4章 STC89C51RC/RD+系列单片机的I/O口结构

4.1 I/O口各种不同的工作模式及配置介绍

I/O口配置

STC89C51RC/RD+系列单片机所有I/O口均(新增P4口)有3种工作类型：准双向口/弱上拉（标准8051输出模式）、仅为输入（高阻）或开漏输出功能。STC89C51RC/RD+系列单片机的P1/P2/P3/P4上电复位后为准双向口/弱上拉（传统8051的I/O口）模式，**P0口上电复位后是**开漏输出。P0口作为总线扩展用时，不用加上拉电阻，作为I/O口用时，需加10K-4.7K上拉电阻。

STC89C51RC/RD+ 的5V单片机的P0口的灌电流最大为12mA,其他I/O口的灌电流最大为6mA。
STC89LE51RC/RD+的3V单片机的P0口的灌电流最大为8mA, 其他I/O口的灌电流最大为4mA。

4.1.1 准双向口/弱上拉输出配置

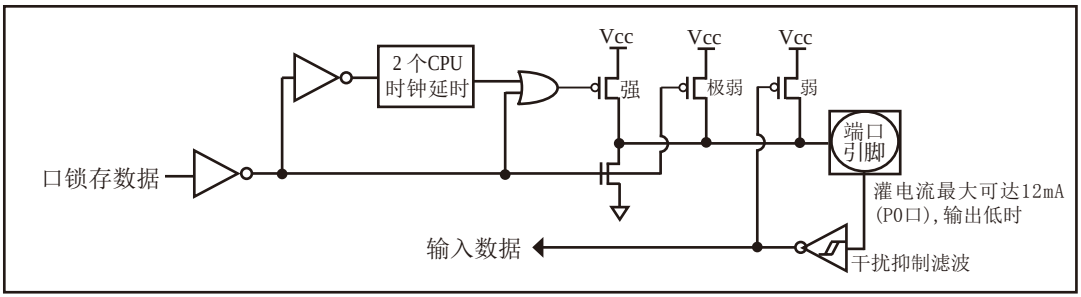
准双向口输出类型可用作输出和输入功能而不需重新配置**端口输出状态。这是因为当端口**输出为1时驱动能力很弱，允许外部装置将其拉低。当引脚输出为低时，它的驱动能力很强，可吸收相当大的电流。准双向口有3个上拉晶体管适应不同的需要。

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。

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。

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。

准双向口输出如下图所示。



准双口输出

STC89LE51RC/RD+系列单片机为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模式中向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。

准双向口带有一个干扰抑制电路。

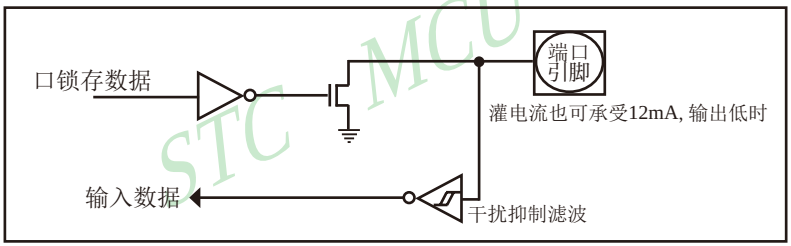
准双向口读外部状态前,要先锁存为 ‘1’,才可读到外部正确的状态.

4.1.2 开漏输出配置(P0口上电复位后处于开漏模式)

P0口上电复位后处于开漏模式，当P0管脚作I/O口时，需外加10K-4.7K的上拉电阻，当P0管脚作为地址/数据复用总线使用时，不用外加上拉电阻。

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。

开漏端口带有一个干扰抑制电路。



开漏输出(P0口上电复位后为开漏模式)

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.

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

4.2 头文件/新增特殊功能寄存器的声明,P4口的使用

对STC89C51RC/RD+系列单片机的P4口的访问，如同访问常规的P1/P2/P3口，并且均可位寻址，P4的地址E8H。

P4端口的地址在E8h ， P4口中的每一位均可位寻址， 位地址如下：								
位	-	P4.6	P4.5	P4.4	P4.3	P4.2	P4.1	P4.0
位地址	EFh	EEh	EDh	ECh	EBh	EAh	E9h	E8h

C语言:

```
        #include<reg51.h>
/*STC所有程序都可只包含以上头文件*/
/*新增特殊功能寄存器按如下方式声明地址即可*/
        sfr     P4    = 0xe8;           /*C语言中声明P4口特殊功能寄存器地址*/
        sbit    P40 = 0xe8;             /*C语言中声明P4.0口位地址*/
        sbit    P41 = 0xe9;             /*C语言中声明P4.1口位地址*/
        sbit    P42 = 0xea;
        sbit    P43 = 0xeb;
        sbit    P44 = 0xec;
        sbit    P45 = 0xed;
        sbit    P46 = 0xee;
        /*以上为P4口的C语言地址声明*/

void main()
{
        unsigned char idata temp = 0;
        P4 = 0xff;

        temp = P4;
        P1 = temp;
        P40 = 1;
        P41 = 0;
        P42 = 1;
        P43 = 0;
        P44 = 1;
        P45 = 0;
        P46 = 1;
        while(1);
}
```

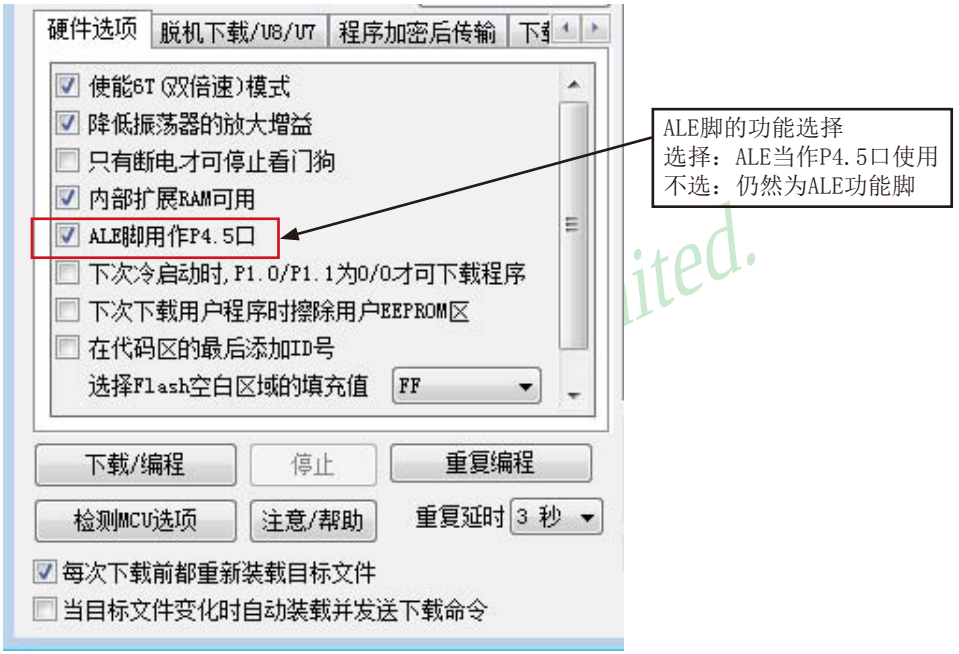
汇编语言：

```
        P4      EQU     0E8H            ; or  P4   DATA   0E8H
        P40     EQU     0E8H            ; or  P40  BIT    0E8H
        P41     EQU     0E9H            ; or  P41  BIT    0E9H
        P42     EQU     0EAH
        P43     EQU     0EBH
        P44     EQU     0ECH
        P45     EQU     0EDH
        P46     EQU     0EEH
        ;以上为P4的地址声明
        P26     EQU     0A6H
        ORG     0000H
        LJMP    MAIN

        ORG     0100H
MAIN:
        MOV     SP,     #0C0H
MAIN_LOOP:
        MOV     A,      P4              ; Read P4 status to Accumulator.
        MOV     P1,     A
        MOV     P4,     #0AH            ; Output data “A”through P4.0 - P4.3
        SETB    P40                     ; P4.0 = 1
        CLR     P41                     ; P4.1 = 0
        SETB    P42                     ; P4.2 = 1
        CLR     P43                     ; P4.3 = 0
        SETB    P44                     ; P4.4 = 1
        CLR     P45                     ; P4.5 = 0
        SETB    P46                     ; P4.6 = 1
        NOP
        MOV     C,      P46
        MOV     P26,    C
        SJMP    MAIN_LOOP
        END
```

注：STC90C58AD/STC90LE58AD系列的P4口地址在C0h。

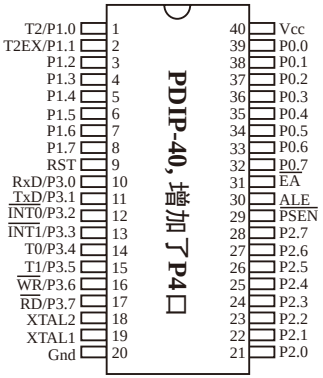
4.3 STC89C51RC系列单片机ALE/P4.5管脚作I/O口使用的设置

STC89C51RC/RD+系列单片机HD版无P4.5口，有ALE管脚。而90C版本的ALE/P4.5管脚既可作I/O口P4.5，也可被复用作ALE管脚，默认是用作ALE管脚。如用户需用到P4.5口，只能选择90C版本的单片机，且需在烧录用户程序时在STC-ISP编程器中将ALE pin选择为用作P4.5，在烧录用户程序时在STC-ISP编程器中该管脚默认的是作为ALE pin。具体设置如下图所示：

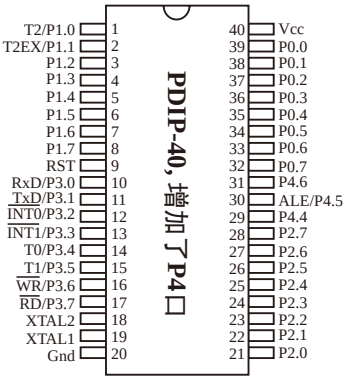


下面是STC89系列单片机HD版和90C版本的管脚图，主要区别在P4.6/P4.5/P4.4三个管脚处。

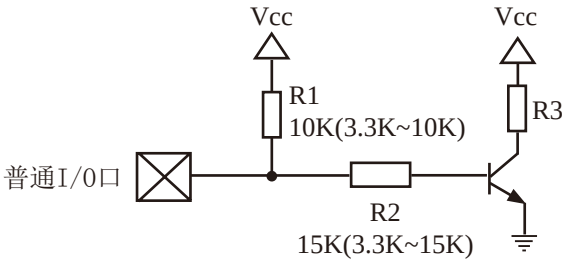
HD版本的管脚图(PDIP-40)



90C版本的管脚图(PDIP-40)



4.4 一种典型三极管控制电路

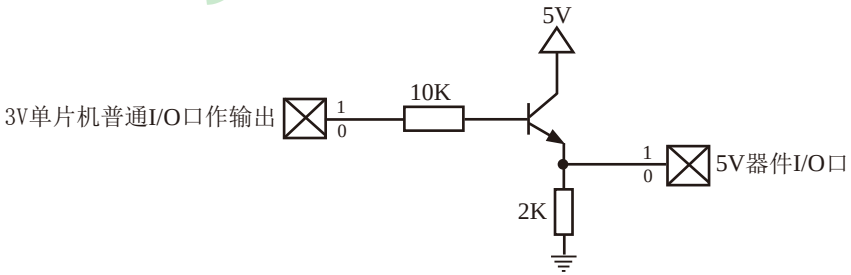


如果用弱上拉控制，建议加上拉电阻R1(3.3K～10K)，如果不加上拉电阻R1(3.3K～10K)，建议R2的值在15K以上。

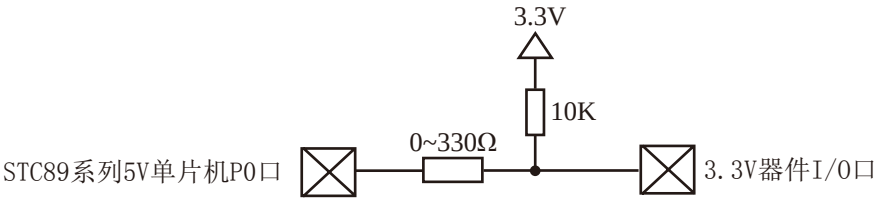
4.5 混合电压供电系统3V/5V器件I/O口互连

3V单片机连接5V器件时作为输入时，为防止3V单片机承受不了5V，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。

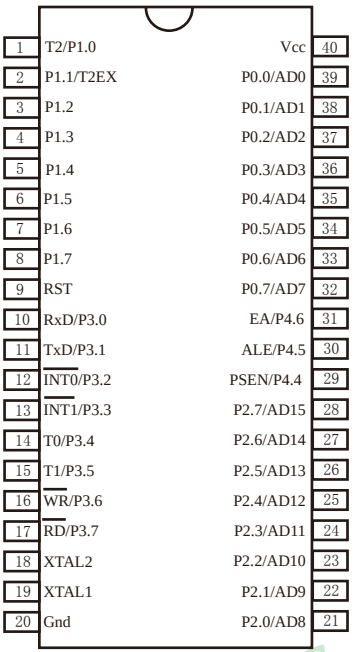
3V单片机连接5V器件时作输出时，为防止3V单片机承受不了5V，可用一个NPN三极管隔离，电路如下：

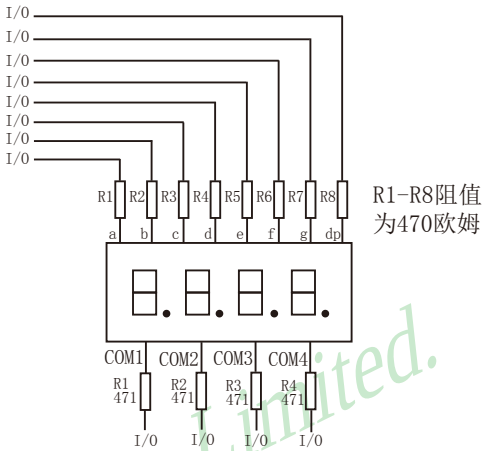


STC89C51RC/RD+系列5V单片机连接3.3V器件时，为防止3.3V器件承受不了5V，可将相应的5V单片机P0口先串一个0~330Ω的限流电阻到3.3V器件I/O口，相应的3.3V器件I/O口外部加10K上拉电阻到3.3V器件的Vcc，这样高电平是3.3V，低电平是0V，输入输出一切正常。



4.6 I/O口直接驱动LED数码管应用线路图



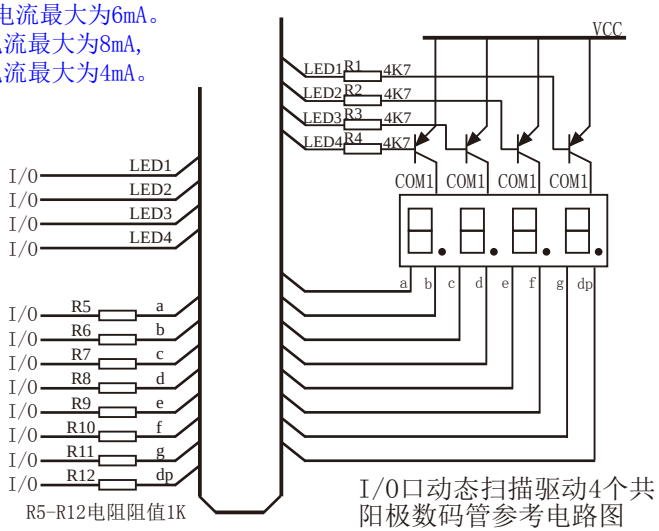


R1-R8阻值为470欧姆

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

STC89C51RC/RD+ 的5V单片机的P0口的灌电流最大为12mA，其他I/O口的灌电流最大为6mA。
STC89LE51RC/RD+的3V单片机的P0口的灌电流最大为8mA，其他I/O口的灌电流最大为4mA。



R5-R12电阻阻值1K

I/O口动态扫描驱动4个共阳极数码管参考电路图

第5章 指令系统

5.1 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

5.1.1 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：

如：MOV A, #70H

这条指令的功能是将立即数70H传送到累加器A中

5.1.2 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。

5.1.3 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, @R1

把数据55H传送到累加器。

5.1.4 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

5.1.5 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。

5.1.6 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。

5.1.7 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

5.2 指令系统分类总结

----与普通8051指令代码完全兼容

如果按功能分类，STC89C51RC/RD+系列单片机指令系统可分为：

1. 数据传送类指令；
2. 算术操作类指令；
3. 逻辑操作类指令；
4. 控制转移类指令；
5. 布尔变量操作类指令。

按功能分类的指令系统表如下表所示。

传统 12T的8051指令执行所需时钟

STC89系列在6T模式时指令执行所需时钟

数据传送类指令

助记符		功能说明	字节数	12时钟/机器周期所需时钟	6时钟/机器周期所需时钟	效率提升
MOV	A, Rn	寄存器内容送入累加器	1	12	6	2倍
MOV	A, direct	直接地址单元中的数据送入累加器	2	12	6	2倍
MOV	A, @Ri	间接RAM中的数据送入累加器	1	12	6	2倍
MOV	A, #data	立即送入累加器	2	12	6	2倍
MOV	Rn, A	累加器内容送入寄存器	1	12	6	2倍
MOV	Rn, direct	直接地址单元中的数据送入寄存器	2	24	12	2倍
MOV	Rn, #data	立即数送入寄存器	2	12	6	2倍
MOV	direct, A	累加器内容送入直接地址单元	2	12	6	2倍
MOV	direct, Rn	寄存器内容送入直接地址单元	2	24	12	2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	12	2倍
MOV	direct, @Ri	间接RAM中的数据送入直接地址单元	2	24	12	2倍
MOV	direct, #data	立即数送入直接地址单元	3	24	12	2倍
MOV	@Ri, A	累加器内容送间接RAM单元	1	12	6	2倍
MOV	@Ri, direct	直接地址单元数据送入间接RAM单元	2	24	12	2倍
MOV	@Ri, #data	立即数送入间接RAM单元	2	12	6	2倍
MOV	DPTR,#data16	16位立即数送入地址寄存器	3	24	12	2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	12	2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	12	2倍
MOVX	A, @Ri	逻辑上在外部的片内扩展RAM,(8位地址）送入累加器	1	24	12	2倍
MOVX	A, @DPTR	逻辑上在外部的片内扩展RAM,(16位地址)送入累加器	1	24	12	2倍
MOVX	@Ri, A	累加器送逻辑上在外部的片内扩展RAM（8位地址）	1	24	12	2倍
MOVX	@DPTR, A	累加器送逻辑上在外部的片内扩展RAM（16位地址）	1	24	12	2倍
PUSH	direct	直接地址单元中的数据压入堆栈	2	24	12	2倍
POP	direcct	出栈送直接地址单元	2	24	12	2倍
XCH	A, Rn	寄存器与累加器交换	1	12	6	2倍
XCH	A,direct	直接地址单元与累加器交换	2	12	6	2倍
XCH	A, @Ri	间接RAM与累加器交换	1	12	6	2倍
XCHD	A, @Ri	间接RAM的低半字节与累加器交换	1	12	6	2倍

算术操作类指令

助记符		功能说明	字节数	12时钟/机器周期所需时钟	6时钟/机器周期所需时钟	效率提升
ADD	A，Rn	寄存器内容送入累加器	1	12	6	2倍
ADD	A，direct	直接地址单元中的数据加到累加器	2	12	6	2倍
ADD	A，@Ri	间接RAM中的数据加到累加器	1	12	6	2倍
ADD	A，#data	立即加到累加器	2	12	6	2倍
ADDC	A，Rn	寄存器内容带进位加到累加器	1	12	6	2倍
ADDC	A，direct	直接地址单元的内容带进位加到累加器	2	12	6	2倍
ADDC	A，@Ri	间接RAM内容带进位加到累加器	1	12	6	2倍
ADDC	A，#data	立即数带进位加到累加器	2	12	6	2倍
SUBB	A，Rn	累加器带借位减寄存器内容	1	12	6	2倍
SUBB	A，direct	累加器带借位减直接地址单元的内容	2	12	6	2倍
SUBB	A，@Ri	累加器带借位减间接RAM中的内容	1	12	6	2倍
SUBB	A，#data	累加器带借位减立即数	2	12	6	2倍
INC	A	累加器加1	1	12	6	2倍
INC	Rn	寄存器加1	1	12	6	2倍
INC	direct	直接地址单元加1	2	12	6	2倍
INC	@Ri	间接RAM单元加1	1	12	6	2倍
DEC	A	累加器减1	1	12	6	2倍
DEC	Rn	寄存器减1	1	12	6	2倍
DEC	direct	直接地址单元减1	2	12	6	2倍
DEC	@Ri	间接RAM单元减1	1	12	6	2倍
INC	DPTR	地址寄存器DPTR加1	1	24	12	2倍
MUL	AB	A乘以B	1	48	24	2倍
DIV	AB	A除以B	1	48	24	2倍
DA	A	累加器十进制调整	1	12	6	2倍

逻辑操作类指令

助记符	功能说明	字节数	12时钟/机器周期所需时钟	6时钟/机器周期所需时钟	效率提升
ANL A，Rn	累加器与寄存器相“ 与”	1	12	6	2倍
ANL A，direct	累加器与直接地址单元相“ 与”	2	12	6	2倍
ANL A，@Ri	累加器与间接RAM单元相“ 与”	1	12	6	2倍
ANL A，#data	累加器与立即数相“ 与”	2	12	6	2倍
ANL direct， A	直接地址单元与累加器相“ 与”	2	12	6	2倍
ANL direct，#data	直接地址单元与立即数相“ 与”	3	24	12	2倍
ORL A， Rn	累加器与寄存器相“ 或”	1	12	6	2倍
ORL A，direct	累加器与直接地址单元相“ 或”	2	12	6	2倍
ORL A，@Ri	累加器与间接RAM单元相“ 或”	1	12	6	2倍
ORL A，#data	累加器与立即数相“ 或”	2	12	6	2倍
ORL direct， A	直接地址单元与累加器相“ 或”	2	12	6	2倍
ORL direct， #data	直接地址单元与立即数相“ 或”	3	24	12	2倍
XRL A， Rn	累加器与寄存器相“ 异或”	1	12	6	2倍
XRL A， d irect	累加器与直接地址单元相“ 异或”	2	12	6	2倍
XRL A， @Ri	累加器与间接RAM单元相“ 异或”	1	12	6	2倍
XRL A， # data	累加器与立即数相“ 异或”	2	12	6	2倍
XRL direct， A	直接地址单元与累加器相“ 异或”	2	12	6	2倍
XRL direct，#data	直接地址单元与立即数相“异或”	3	24	12	2倍
CLR A	累加器清“ 0”	1	12	6	2倍
CPL A	累加器求反	1	12	6	2倍
RL A	累加器循环左移	1	12	6	2倍
RLC A	累加器带进位位循环左移	1	12	6	2倍
RR A	累加器循环右移	1	12	6	2倍
RRC A	累加器带进位位循环右	1	12	6	2倍
SWAP A	累加器半字节交换	1	12	6	2倍

控制转移类指令

助记符		功能说明	字节数	12时钟/机器周期所需时钟	6时钟/机器周期所需时钟	效率提升
ACALL	addr11	绝对（短）调用子程序	2	24	12	2倍
LCALL	addr16	长调用子程序	3	24	12	2倍
RET		子程序返回	1	24	12	2倍
RETI		中断返回	1	24	12	2倍
AJMP	addr11	绝对（短）转移	2	24	12	2倍
LJMP	addr16	长转移	3	24	12	2倍
SJMP	re1	相对转移	2	24	12	2倍
JMP	@A+DPTR	相对于DPTR的间接转移	1	24	12	2倍
JZ	re1	累加器为零转移	2	24	12	2倍
JNZ	re1	累加器非零转移	2	24	12	2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	12	2倍
CJNE	A，#data，re1	累加器与立即数比较，不相等则转移	3	24	12	2倍
CJNE	Rn，#data，re1	寄存器与立即数比较，不相等则转移	3	24	12	2倍
CJNE	@Ri，#data，re1	间接RAM单元与立即数比较，不相等则转移	3	24	12	2倍
DJNZ	Rn，re1	寄存器减1，非零转移	3	24	12	2倍
DJNZ	direct，re1	直接地址单元减1，非零转移	3	24	12	2倍
NOP		空操作	1	12	6	2倍

布尔变量操作类指令

助记符		功能说明	字节数	12时钟/机器周期所需时钟	6时钟/机器周期所需时钟	效率提升
CLR	C	清零进位位	1	12	6	2倍
CLR	bit	清0直接地址位	2	12	6	2倍
SETB	C	置1进位位	1	12	6	2倍
SETB	bit	置1直接地址位	2	12	6	2倍
CPL	C	进位位求反	1	12	6	2倍
CPL	bit	直接地址位求反	2	12	6	2倍
ANL	C, bit	进位位和直接地址位相“与”	2	24	12	2倍
ANL	C, /bit	进位位和直接地址位的反码相“与”	2	24	12	2倍
ORL	C, bit	进位位和直接地址位相“或”	2	24	12	2倍
ORL	C, /bit	进位位和直接地址位的反码相“或”	2	24	12	2倍
MOV	C, bit	直接地址位送入进位位	2	12	6	2倍
MOV	bit, C	进位位送入直接地址位	2	24	12	2倍
JC	rel	进位位为1则转移	2	24	12	2倍
JNC	rel	进位位为0则转移	2	24	12	2倍
JB	bit, rel	直接地址位为1则转移	3	24	12	2倍
JNB	bit, rel	直接地址位为0则转移	3	24	12	2倍
JBC	bit, rel	直接地址位为1则转移，该位清0	3	24	12	2倍

5.3 传统8051单片机指令定义详解(中文&English)

5.3.1 传统8051单片机指令定义详解

ACALL addr 11

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： 2

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$_{7\text{-}0}$)**

(SP)←**(SP) + 1**

((SP))←**(PC$_{15\text{-}8}$)**

(PC$_{10\text{-}0}$)← 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(000011B)，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　addr 11

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：2

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(010101011B)，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：2

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：2

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：2

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

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：2

二进制编码：

1 0 1 1	0 1 0 0	immediata 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

二进制编码：

1 0 1 1	1 r r r	immediata 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

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

$(A) \leftarrow \overline{(A)}$

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

$(C) \leftarrow \overline{(C)}$

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

$(bit) \leftarrow \overline{(bit)}$

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： 1

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0001000B)，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：4

二进制编码：

1 0 0 0	0 1 0 0

操作：DIV
$(A)_{15\text{-}8}$
$(B)_{7\text{-}0}$ ← (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

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

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，该指令步影响标志位。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(0111110B)，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

```
INC   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：2

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：2

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：2

二进制编码：

0 0 0 1	0 0 0 0	bit address	rel. address

操作：

```
JBC
(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： 2

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： 2

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110010108，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：　2

二进制编码：

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

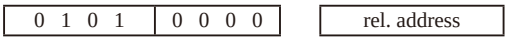
```
JNC   LABEL1
CPL   C
JNC   LABEL2
```

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：　2

二进制编码：



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：　2

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：　2

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：2

二进制编码：

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：2

二进制编码：

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

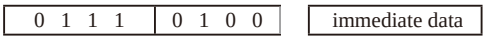
操作：MOV
(A) ← **((Ri))**

MOV A,#data

指令长度(字节): 2

执行周期: 1

二进制编码:



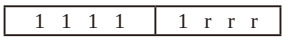
操作: MOV
(A)← #data

MOV Rn, A

指令长度(字节): 1

执行周期: 1

二进制编码:



操作: MOV
(Rn)←(A)

MOV Rn,direct

指令长度(字节): 2

执行周期: 2

二进制编码:

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:

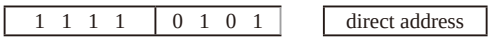
操作: MOV
(Rn) ← #data

MOV direct, A

指令长度(字节): 2

执行周期: 1

二进制编码:



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:

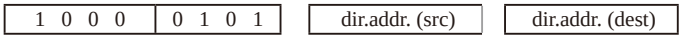
操作: MOV
(direct) ← (Rn)

MOV　direct, direct

指令长度(字节)：3

执行周期：2

二进制编码：



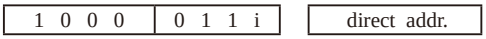
操作：MOV
(direct)← **(direct)**

MOV　direct, @Ri

指令长度(字节)：2

执行周期：2

二进制编码：



操作：MOV
(direct)←**((Ri))**

MOV　direct,#data

指令长度(字节)：3

执行周期：2

二进制编码：

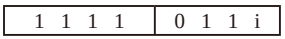
操作：MOV
(direct) ← **#data**

MOV　@Ri, A

指令长度(字节)：1

执行周期：1

二进制编码：



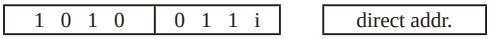
操作：MOV
((Ri)) ← **(A)**

MOV　@Ri, direct

指令长度(字节)：2

执行周期：2

二进制编码：



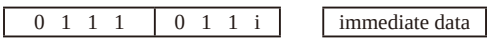
操作：MOV
((Ri)) ← **(direct)**

MOV　@Ri, #data

指令长度(字节)：2

执行周期：1

二进制编码：



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P2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：2

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：2

二进制编码：

1	0	0	1	0	0	0	0	immediate data 15-8	immediate data7-0

操作：MOV
(DPTR) ← **#data**$_{15\text{-}0}$
DPH DPL ← **#data**$_{15\text{-}8}$ #data$_{7\text{-}0}$

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： 2

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： 2

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：Inte1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： 2

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： 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：2

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

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12 800 (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：4

二进制编码：

1 0 1 0	0 1 0 0

操作：MUL
$(A)_{7\text{-}0}$ ← **(A)×(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*R*0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： 2

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：2

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：2

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：2

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：2

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：2

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：2

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) ← (**A**n)　n = 0-6

(A0) ← (**C**)

(C) ← (**A**7)

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) ← (**A**n+1)　n = 0 - 6

(A7) ← (**A**0)

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：2

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

$(A_{3-0}) \rightleftarrows (A_{7-4})$

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

$(A_{3\text{-}0}) \rightleftarrows (Ri_{3\text{-}0})$

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：2

二进制编码：

0 1 1 0	0 0 1 1	direct address	immediate data

操作：XRL

(direct) ← (direct) ⊻ # data

5.3.2 Instruction Definitions of Traditional 8051 MCU

ACALL addr 11

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 suceesively 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 executingthe 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: 2

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 occured.

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 occured.
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 (01101101B) 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 addr 11

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: 2

Encoding:

a10 a9 a8 0	0 0 0 1	a7 a6 a5 a4	a3 a2 a1 a0

Operation: AJMP

(PC)← (PC)+ 2

($PC_{10\text{-}0}$)← 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: 2

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 affceted.* No other flsgs 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: 2

Encoding:

1 0 0 0	0 0 1 0	bit address

Operation: ANL
(C) ← **(C)** ∧ (bit)

ANL C, /bit

Bytes: 2

Cycles: 2

Encoding:

1 0 1 1	0 0 0 0	bit address

Operation: ANL

(C)←**(C)** ∧ $\overline{\text{(bit)}}$

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 dat, 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

Encoding:

1 0 1 1	0 1 0 0	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: 2

Encoding:

1 0 1 1	0 1 0 1	immediata 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

Encoding:

1 0 1 1	1 r r r	immediata 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

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 Aecunmlator 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 (101000011B).

Bytes: 1

Cycles: 1

Encoding:

1 1 1 1	0 1 0 0

Operation: CPL

$(A) \leftarrow \overline{(A)}$

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

$(C) \leftarrow \overline{(C)}$

CPL bit

Bytes: 2

Cycles: 1

Encoding:

1 0 1 1	0 0 1 0	bit address

Operation: CPL

$(bit) \leftarrow \overline{(bit)}$

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-111xxxx), 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: 1

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 00010010B) in B, since 251 = (13×18) + 17. Carry and OV will both be cleared.

Bytes: 1

Cycles: 4

Encoding:

1 0 0 0	0 1 0 0

Operation: DIV

$(A)_{15-8}$
$(B)_{7-0}$ ← (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 afected. 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

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

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 (011111110B). 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,
INC　DPTR
INC　DPTR
INC　DPTR
will change DPH and DPL to 13H and 01H.

Bytes: 1

Cycles: 2

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: 2

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 wili 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: 2

Encoding:

0 0 0 1	0 0 0 0	bit address	rel. address

Operation:

```
JBC
(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      LABEL2s
```

will set the carry and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 2

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: 2

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: 2

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: 2

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: 2

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: 2

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 loated 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 “SUT2N” is assigned to program memory location 1234H. After executing the instruction,

LCALL SUT2N

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: 2

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: 2

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: 2

Encoding:

1 0 1 0	1 r r r	direct addr.

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: 2

Encoding:

1 0 0 0	1 r r r	direct address

Operation: MOV
(direct) ← **(Rn)**

MOV　direct, direct

Bytes: 3

Cycles: 2

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: 2

Encoding:

1 0 0 0	0 1 1 i	direct addr.

Operation: MOV
(direct)←**((Ri))**

MOV　direct,#data

Bytes: 3

Cycles: 2

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: 2

Encoding:

1 0 1 0	0 1 1 i	direct addr.

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: 2

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: 2

Encoding:

1 0 0 1	0 0 0 0	immediate data 15-8	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 defimed 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: 2

Encoding:

1 0 0 1	0 0 1 1

Operation: MOVC
(A) ← ((A)+(DPTR))

MOVC A,@A+PC

Bytes: 1

Cycles: 2

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: 2

Encoding:

1 1 1 0	0 0 1 i

Operation: MOVX
(A) ← ((Ri))

MOVX A,@DPTR

Bytes: 1

Cycles: 2

Encoding:

1 1 1 0	0 0 0 0

Operation: MOVX
(A) ← ((DPTR))

MOVX @Ri, A

Bytes: 1

Cycles: 2

Encoding:

1 1 1 1	0 0 1 i

Operation: MOVX
((Ri))← (A)

MOVX @DPTR, A

Bytes: 1

Cycles: 2

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: 4

Encoding:

1 0 1 0	0 1 0 0

Operation: MUL
$(A)_{7\text{-}0}$ ← (A)×(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: 2

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 P1.0
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: 2

Encoding:

0 1 1 1	0 0 1 0	bit address

Operation: ORL

(C) ← (C)∨(bit)

ORL C, /bit

Bytes: 2

Cycles: 2

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: 2

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 variableis then copied into the internal RAM location addressed by the Stack Pointer. Otherwise no flags are affected.

Example: On entering 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: 2

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: 2

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: 2

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: 2

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

$(A_{3-0}) \rightleftarrows (A_{7-4})$

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) $\rightleftarrows$ (Rn)

XCH A, direct

Bytes: 2

Cycles: 1

Encoding:

1 1 0 0	0 1 0 1	direct address

Operation: XCH

(A) $\rightleftarrows$ (direct)

XCH A, @Ri

Bytes: 1

Cycles: 1

Encoding:

1 1 0 0	0 1 1 i

Operation: XCH

(A) $\rightleftarrows$ ((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}) $\rightleftarrows$ (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 vatue 69H (01101001B).

When the destination is a directly addressed byte, this instruction can complement combinnation 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 outpue 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: 2

Encoding:

0 1 1 0	0 0 1 1	direct address	immediate data

Operation: XRL
(direct) ← (direct) ⅄ # data

第6章 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。

STC89C51RC/RD+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($\overline{\text{INT0}}$)、定时器0中断、外部中断1($\overline{\text{INT1}}$)、定时器1中断、串口(UART)中断、定时器2中断、外部中断2($\overline{\text{INT2}}$)、外部中断3($\overline{\text{INT3}}$)。所有的中断都具有4个中断优先级。用户可以用关总中断允许位(EA/IE.7)或相应中断的允许位来屏蔽所有的中断请求，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；每一个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及同优先级的中断。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。STC89C51RC/RD+系列单片机的各个中断查询次序如下表6-1所示：

表6-1 中断查询次序

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(IPH,IP)	优先级0(最低)	优先级1	优先级2	优先级3(最高)	中断请求标志位	中断允许控制位
$\overline{\text{INT0}}$ (外部中断 0)	0003H	0 (highest)	PX0H, PX0	0, 0	0, 1	1, 0	1, 1	IE0	EX0/EA
Timer 0	000BH	1	PT0H, PT0	0, 0	0, 1	1, 0	1, 1	TF0	ET0/EA
$\overline{\text{INT1}}$ (外部中断1)	0013H	2	PX1H, PX1	0, 0	0, 1	1, 0	1, 1	IE1	EX1/EA
Timer1	001BH	3	PT1H, PT1	0, 0	0, 1	1, 0	1, 1	TF1	ET1/EA
UART	0023H	4	PSH, PS	0, 0	0, 1	1, 0	1, 1	RI+TI	
Timer2	002BH	5	PT2H, PT2	0, 0	0, 1	1, 0	1, 1	TF2 + EXF2	(ET2)/EA
$\overline{\text{INT2}}$ (外部中断2)	0033H	6	PX2H, PX2	0, 0	0, 1	1, 0	1, 1	IE2	EX2/EA
$\overline{\text{INT3}}$ (外部中断3)	003BH	7 (lowest)	PX3H, PX3	0, 0	0, 1	1, 0	1, 1	IE3	EX3/EA

通过设置新增加的特殊功能寄存器IPH中的相应位，可将中断优先级设为四级，如果只设置IP或XICON，那么中断优先级就只有两级，与传统8051单片机两级中断优先级完全兼容。

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    UART_Routine(void)          interrupt 4;
void    Timer2_Routine(void)        interrupt 5;
void    Int2_Routine(void)          interrupt 6;
void    Int3_Routine(void)          interrupt 7;
```

6.1 中断结构

STC89C51RC/RD+系列单片机的中断系统结构示意图如图6-1所示

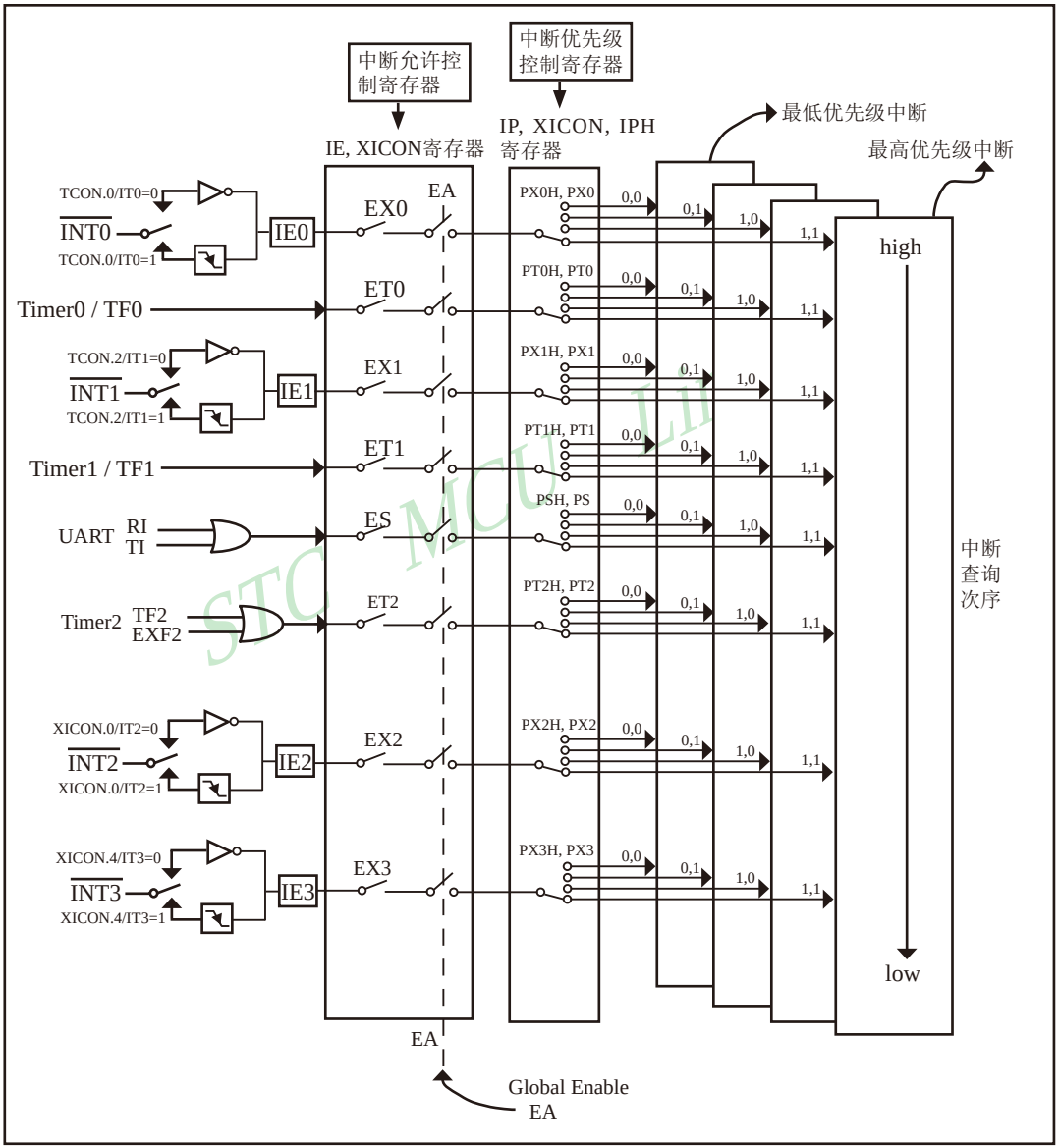


图6-1　STC89C51RC/RD+系列中断系统结构图

外部中断0($\overline{INT0}$)、外部中断1($\overline{INT1}$)、外部中断2($\overline{INT2}$)和外部中断3($\overline{INT3}$)既可低电平触发，也下降沿触发。请求四个外部中断的标志位是位于寄存器TCON中的IE0/TCON.1、IE1/TCON.3、IE2/XICON.2和IE3/XICON.5。当外部中断服务程序被响应后，中断请求标志位IE0、IE1、IE2和IE3会自动被清0。TCON寄存器中的IT0/TCON.0、IT1/TCON.2、IT2/XICON.0和IT3/XICON.4决定了外部中断0、1、2和3是低电平触发方式还是下降沿触发方式。如果ITx = 0(x = 0,1,2,3)，那么系统在INTx(x = 0,1,2,3)脚探测到低电平后可产生外部中断。如果ITx = 1(x = 0,1,2,3)，那么系统在INTx(x= 0,1,2,3)脚探测下降沿后可产生外部中断。外部中断0($\overline{INT0}$)、外部中断1($\overline{INT1}$)、外部中断2($\overline{INT2}$)和外部中断3($\overline{INT3}$)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，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。

当串行口接收中断请求标志位RI和串行口1发送中断请求标志位TI中的任何一个被置为1后，串行口中断都会产生。。

定时器2的中断请求标志位是TF2和EXF2。当定时器寄存器TH2/TL2溢出时，溢出标志位TF2会被置位，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TF2会被硬件清除。当EXEN2=1且T2EX的负跳变产生捕获或重装时，EXF2置位。定时器2中断使能时，EXF2=1也将使CPU从中断向量处执行定时器2中断子程序。

各个中断触发行为总结如下表6-2所示：

表6-2 中断触发

中断源	触发行为
$\overline{INT0}$ (外部中断0)	(IT0/TCON.0 = 1): 下降沿 (IT0/TCON.0 = 0): 低电平
Timer 0	定时器0溢出
$\overline{INT1}$ (外部中断1)	(IT1/TCON.2 = 1): 下降沿 (IT1/TCON.2 = 0): 低电平
Timer1	定时器1溢出
UART	发送或接受完成
Timer2	定时器2溢出
$\overline{INT2}$ (外部中断2)	(IT2/XICON.0 = 1): 下降沿 (IT2/XICON.0 = 0): 低电平
$\overline{INT3}$ (外部中断3)	(IT3/XICON.4 = 1): 下降沿 (IT3/XICON.4 = 0): 低电平

6.2 中断寄存器

符号	描述	地址	位地址及符号 MSB LSB								复位值
IE	Interrupt Enable	A8H	EA	-	ET2	ES	ET1	EX1	ET0	EX0	0x00 0000B
IP	Interrupt Priority Low	B8H	-	-	PT2	PS	PT1	PX1	PT0	PX0	xx00 0000B
IPH	Interrupt Priority High	B7H	PX3H	PX2H	PT2H	PSH	PT1H	PX1H	PT0H	PX0H	0000,0000B
TCON	Timer/Counter 0 and 1 Control	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
T2CON	Timer/Counter 2 Control	C8H	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/$\overline{T2}$	CP/$\overline{RL2}$	0000 0000B
XICON	Auxiliary Interupt Control	C0H	PX3	EX3	IE3	IT3	PX2	EX2	IE2	IT2	0000 0000B

上表中列出了与STC89C51RC/RD+系列单片机中断相关的所有寄存器，下面逐一地对上述寄存器进行介绍。

1. 中断允许寄存器IE和XICON

STC89C51RC/RD+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（地址为A8H）和XICON（地址为C0H）控制的。寄存器IE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	-	ET2	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。

ET2: 定时/计数器T2的溢出中断允许位。ET2=1，允许T2中断；ET2=0，禁止T2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。

寄存器XICON的格式如下：

XICON：辅助中断控制寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
XICON	C0H	name	PX3	EX3	IE3	IT3	PX2	EX2	IE2	IT2

PX3: 置位表明外部中断3的优先级为高,优先级最终由[PX3H,PX3]=[0,0]；[0,1]；[1,0]；[1,1]来决定。

EX3 ：如被设置成1，允许外部中断3中断；如被清成0，禁止外部中断3中断。

IE3：外部中断3中断请求标志位，中断条件成立后，IE3=1,可由硬件自动清零。

IT3：当此位由软件置位时,外部中断3为下降沿触发中断；当此位由软件清零时,为低电平触发中断。

PX2: 置位表明外部中断2的优先级为高，优先级最终由[PX2H,PX2]=[0,0]；[0,1]；[1,0]；[1,1]来决定。

EX2 ：如被设置成1，允许外部中断2中断；如被清成0，禁止外部中断2中断。

IE2：外部中断2中断请求标志位，中断条件成立后，IE2=1,可由硬件自动清零。

IT2：当此位由软件置位时,外部中断2为下降沿触发中断；当此位由软件清零时,为低电平触发中断。

STC89C51RC/RD+系列单片机复位以后，IE和XICON被清0，由用户程序置“1”或清“0”IE和XICON相应的位，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和XICON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）。

2. 中断优先级控制寄存器IP/XICON和IPH

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。STC89C51RC/RD+系列单片机通过设置新增加的特殊功能寄存器(IPH/XICON)中的相应位，可将中断优先级设置为4个中断优先级；如果只设置IP，那么中断优先级只有两级，与传统8051单片机两级中断优先级完全兼容。

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

STC89C51RC/RD+系列单片机的片内各优先级控制寄存器的格式如下:

IPH: 中断优先级控制寄存器高(不可位寻址)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IPH	B7H	name	PX3H	PX2H	PT2	PSH	PT1H	PX1H	PT0H	PX0H

XICON : 辅助中断控制寄存器 (可位寻址)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
XICON	C0H	name	PX3	EX3	IE3	IT3	PX2	EX2	IE2	IT2

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	-	-	PT2	PS	PT1	PX1	PT0	PX0

PX3H, PX3: 外部中断3优先级控制位。

当PX3H=0且PX3=0时，外部中断3为最低优先级中断(优先级0)
当PX3H=0且PX3=1时，外部中断3为较低优先级中断(优先级1)
当PX3H=1且PX3=0时，外部中断3为较高优先级中断(优先级2)
当PX3H=1且PX3=1时，外部中断3为最高优先级中断(优先级3)

PX2H, PX2: 外部中断2优先级控制位。

当PX2H=0且PX2=0时，外部中断2为最低优先级中断(优先级0)
当PX2H=0且PX2=1时，外部中断2为较低优先级中断(优先级1)
当PX2H=1且PX2=0时，外部中断2为较高优先级中断(优先级2)
当PX2H=1且PX2=1时，外部中断2为最高优先级中断(优先级3)

PT2H, PT2: 定时器2中断优先级控制位。

当PT2H=0且PT2=0时，定时器2中断为最低优先级中断(优先级0)
当PT2H=0且PT2=1时，定时器2中断为较低优先级中断(优先级1)
当PT2H=1且PT2=0时，定时器2中断为较高优先级中断(优先级2)
当PT2H=1且PT2=1时，定时器2中断为最高优先级中断(优先级3)

PSH, PS: 串口1中断优先级控制位。

当PSH=0且PS=0时，串口1中断为最低优先级中断(优先级0)
当PSH=0且PS=1时，串口1中断为较低优先级中断(优先级1)
当PSH=1且PS=0时，串口1中断为较高优先级中断(优先级2)
当PSH=1且PS=1时，串口1中断为最高优先级中断(优先级3)

PT1H, PT1: 定时器1中断优先级控制位。

当PT1H=0且PT1=0时，定时器1中断为最低优先级中断(优先级0)
当PT1H=0且PT1=1时，定时器1中断为较低优先级中断(优先级1)
当PT1H=1且PT1=0时，定时器1中断为较高优先级中断(优先级2)
当PT1H=1且PT1=1时，定时器1中断为最高优先级中断(优先级3)

PX1H, PX1: 外部中断1优先级控制位。

当PX1H=0且PX1=0时，外部中断1为最低优先级中断(优先级0)
当PX1H=0且PX1=1时，外部中断1为较低优先级中断(优先级1)
当PX1H=1且PX1=0时，外部中断1为较高优先级中断(优先级2)
当PX1H=1且PX1=1时，外部中断1为最高优先级中断(优先级3)

PT0H, PT0: 定时器0中断优先级控制位。

当PT0H=0且PT0=0时，定时器0中断为最低优先级中断(优先级0)
当PT0H=0且PT0=1时，定时器0中断为较低优先级中断(优先级1)
当PT0H=1且PT0=0时，定时器0中断为较高优先级中断(优先级2)
当PT0H=1且PT0=1时，定时器0中断为最高优先级中断(优先级3)

PX0H, PX0: 外部中断0优先级控制位。

当PX0H=0且PX0=0时，外部中断0为最低优先级中断(优先级0)
当PX0H=0且PX0=1时，外部中断0为较低优先级中断(优先级1)
当PX0H=1且PX0=0时，外部中断0为较高优先级中断(优先级2)
当PX0H=1且PX0=1时，外部中断0为最高优先级中断(优先级3)

中断优先级控制寄存器IP和IPH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IPH寄存器的内容只能用字节操作指令来更新。STC89C51RC/RD+系列单片机复位后IP和IPH均为00H，各个中断源均为低优先级中断。

3. 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请求源（$\overline{\text{INT1}}$/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1中断源类型选择位。IT1=0，$\overline{\text{INT1}}$/P3.3引脚上的低电平信号可触发外部中断1。IT1=1，外部中断1为下降沿触发方式。

IE0：外部中断0请求源（$\overline{\text{INT0}}$/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0中断源类型选择位。IT0=0，$\overline{\text{INT0}}$/P3.2引脚上的低电平可触发外部中断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。

3. 定时器/计数器2控制寄存器T2CON

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
T2CON	C8H	name	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/$\overline{T2}$	CP/$\overline{RL2}$

TF2：定时器2 溢出标志。定时器2溢出时置位，必须由软件清除。当RCLK或TCLK=1 时，TF2将不会置位

EXF2：定时器2外部标志。当EXEN2=1且T2EX的负跳变产生捕获或重装时，EXF2置位。定时器2中断使能时，EXF2=1将使CPU从中断向量处执行定时器2 中断子程序。EXF2位必须用软件清零。在递增/递减计数器模式（DCEN=1）中，EXF2不会引起中断

RCLK：接收时钟标志。RCLK置位时，定时器2的溢出脉冲作为串行口模式1和模式3的接收时钟。RCLK=0时，将定时器1的溢出脉冲作为接收时钟

TCLK：发送时钟标志。TCLK置位时，定时器2的溢出脉冲作为串行口模式1和模式3的发送时钟。TCLK=0时，将定时器1的溢出脉冲作为发送时钟

EXEN2：定时器2外部使能标志。当其置位且定时器2未作为串行口时钟时，允许T2EX的负跳变产生捕获或重装。EXEN2=0 时，T2EX的跳变对定时器2无效

TR2：定时器2启动/停止控制位。置1时启动定时器

C/$\overline{T2}$：定时器/ 计数器选择。（定时器2）

0= 内部定时器（OSC/12 或OSC/6）

1 = 外部事件计数器（下降沿触发）

CP/$\overline{RL2}$：捕获/ 重装标志。置位：EXEN2=1 时，T2EX 的负跳变产生捕获。 清零：EXEN2=0 时，定时器2溢出或T2EX 的负跳变都可使定时器自动重装。当RCLK=1 或TCLK=1时，该位无效且定时器强制为溢出时自动重装

6.3 中断优先级

STC89C51RC/RD+系列单片机的所有的中断都具有4个中断优先级，对于这些中断请求源可编程为高优先级中断或低优先级中断，可实现两级中断服务程序嵌套。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

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，STC89C51RC/RD+系列单片机各中断优先查询次序如下：

	中断源	查询次序
0.	$\overline{\text{INT0}}$	(highest)
1.	Timer 0	↓
2.	$\overline{\text{INT1}}$	↓
3.	Timer 1	↓
4.	UART	↓
5.	Timer 1	↓
6.	$\overline{\text{INT1}}$	↓
7.	$\overline{\text{INT1}}$	(lowest)

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    UART_Routine(void)          interrupt 4;
void    Timer2_Routine(void)        interrupt 5;
void    Int2_Routine(void)          interrupt 6;
void    Int3_Routine(void)          interrupt 7;
```

6.4 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。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，是同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
UART	0023H
Timer 2	002BH
External Interrupt 2	0033H
External Interrupt 3	003BH

当“转去执行中断”时，引起中断的标志位将被硬件自动清零。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 MAIN)。

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。

6.5 外部中断

外部中断0($\overline{INT0}$)、外部中断1($\overline{INT1}$)、外部中断2($\overline{INT2}$)和外部中断3($\overline{INT3}$)有两种触发方式，下降沿触发方式和低电平触发方式。

请求四个外部中断的标志位是位于寄存器TCON中的IE0/TCON.1、IE1/TCON.3、IE2/XICON.2和IE3/XICON.5。当外部中断服务程序被响应后，中断请求标志位IE0、IE1、IE2和IE3会自动被清0。TCON寄存器中的IT0/TCON.0、IT1/TCON.2、IT2/XICON.0和IT3/XICON.4决定了外部中断0、1、2和3是低电平触发方式还是下降沿触发方式。如果ITx = 0(x = 0,1,2,3)，那么系统在INTx(x = 0,1,2,3)脚探测到低电平后可产生外部中断。如果ITx = 1(x = 0,1,2,3)，那么系统在INTx(x= 0,1,2,3)脚探测下降沿后可产生外部中断。外部中断0($\overline{INT0}$)、外部中断1($\overline{INT1}$)、外部中断2($\overline{INT2}$)和外部中断3($\overline{INT3}$)还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个系统时钟。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个系统时钟，而且低电平也要持续至少一个系统时钟，才能确保该下降沿被CPU检测到。同样，如果外部中断是低电平可触发，则要求必须在相应的引脚维持低电平至少2个系统时钟，这样才能确保CPU能够检测到该低电平信号。

6.6 中断测试程序

6.6.1 外部中断0($\overline{\text{INT0}}$)的测试程序(C程序及汇编程序)

1. 程序1——演示外部中断0的下降沿中断

C程序:

```
/*---------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ----------------------------------------*/
/* --- 演示STC89xx系列单片机外部中断0(下降沿) ----------------------*/
/* --- Mobile: (86)13922805190 -------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序，  ------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -----------*/
/*---------------------------------------------------------------------------------*/

#include "reg51.h"

//External interrupt0 service routine
void exint0() interrupt 0                       //INT0, interrupt 0 (location at 0003H)
{
        P0++;
}

void main()
{
        IT0 = 1;                                //set INT0 interrupt type (1:Falling 0:Low level)
        EX0 = 1;                                //enable INT0 interrupt
        EA = 1;                                 //open global interrupt switch

        while (1);
}
```

汇编程序:

```
/*---------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ----------------------------------*/
/* --- 演示STC89xx系列单片机外部中断0(下降沿) ----------------------*/
/* --- Mobile: (86)13922805190 -------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 ----------*/
/*---------------------------------------------------------------------------*/

;-----------------------------------------
;interrupt vector table

        ORG     0000H
        LJMP    MAIN

        ORG     0003H                   ;INT0, interrupt 0 (location at 0003H)
        LJMP    EXINT0

;-----------------------------------------

        ORG     0100H
MAIN:
        MOV     SP,     #7FH            ;initial SP
        SETB    IT0                     ;set INT0 interrupt type (1:Falling 0:Low level)
        SETB    EX0                     ;enable INT0 interrupt
        SETB    EA                      ;open global interrupt switch
        SJMP    $

;-----------------------------------------
;External interrupt0 service routine

EXINT0:
        CPL     P0.0
        RETI

;-----------------------------------------

        END
```

2. 程序2——演示外部中断0的下降沿中断唤醒掉电模式

C程序：

```
/*---------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ----------------------------------------*/
/* --- 演示STC89xx系列单片机外部中断0(下降沿)唤醒掉电模式 ----*/
/* --- Mobile: (86)13922805190 -------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -----------*/
/*---------------------------------------------------------------------------------*/

#include "reg51.h"
#include "intrins.h"

//External interrupt0 service routine
void exint0( )        interrupt 0                   //INT0, interrupt 0 (location at 0003H)
{
}

void main()
{
        IT0 = 1;                                    //set INT0 interrupt type (1:Falling 0:Low level)
        EX0 = 1;                                    //enable INT0 interrupt
        EA = 1;                                     //open global interrupt switch

        while (1)
        {
                INT0 = 1;                           //ready read INT0 port
                while (!INT0);                      //check INT0
                _nop_();
                _nop_();
                PCON = 0x02;                        //MCU power down
                _nop_();
                _nop_();
                P1++;
        }
}
```

汇编程序：

```
/*---------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ----------------------------------*/
/* --- 演示STC89xx系列单片机外部中断0(下降沿)唤醒掉电模式 ----*/
/* --- Mobile: (86)13922805190 -------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 ----------*/
/*---------------------------------------------------------------------------*/

;-----------------------------------------
;interrupt vector table

        ORG     0000H
        LJMP    MAIN

        ORG     0003H                   ;INT0, interrupt 0 (location at 0003H)
        LJMP    EXINT0

;-----------------------------------------

        ORG     0100H
MAIN:
        MOV     SP,     #7FH            ;initial SP
        SETB    IT0                     ;set INT0 interrupt type (1:Falling 0:Low level)
        SETB    EX0                     ;enable INT0 interrupt
        SETB    EA                      ;open global interrupt switch
LOOP:
        SETB    INT0                    ;ready read INT0 port
        JNB     INT0,   $               ;check INT0
        NOP
        NOP
        MOV     PCON,   #02H            ;MCU power down
        NOP
        NOP
        CPL     P1.0
        SJMP    LOOP

;-----------------------------------------
;External interrupt0 service routine

EXINT0:
        RETI

;-----------------------------------------

        END
```

6.6.2 外部中断1($\overline{INT1}$)的测试程序(C程序及汇编程序)

1. 程序1——演示外部中断1的下降沿中断

C程序:

```
/*---------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ----------------------------------------*/
/* --- 演示STC89xx系列单片机外部中断1(下降沿) ----------------------*/
/* --- Mobile: (86)13922805190 -------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -----------*/
/*---------------------------------------------------------------------------------*/

#include "reg51.h"

//External interrupt1 service routine
void exint1() interrupt 2                   //INT1, interrupt 2 (location at 0013H)
{
        P0++;
}

void main()
{
        IT1 = 1;                            //set INT1 interrupt type (1:Falling only 0:Low level)
        EX1 = 1;                            //enable INT1 interrupt
        EA = 1;                             //open global interrupt switch

        while (1);
}
```

汇编程序：

```
/*---------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ----------------------------------------*/
/* --- 演示STC89xx系列单片机外部中断1(下降沿) ----------------------*/
/* --- Mobile: (86)13922805190 -------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序，  ----------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -----------*/
/*---------------------------------------------------------------------------------*/

;-----------------------------------------
;interrupt vector table

        ORG     0000H
        LJMP    MAIN

        ORG     0013H                   ;INT1, interrupt 2 (location at 0013H)
        LJMP    EXINT1

;-----------------------------------------

        ORG     0100H
MAIN:
        MOV     SP,     #7FH            ;initial SP
        SETB    IT1                     ;set INT1 interrupt type (1:Falling 0:Low level)
        SETB    EX1                     ;enable INT1 interrupt
        SETB    EA                      ;open global interrupt switch
        SJMP    $

;-----------------------------------------
;External interrupt1 service routine

EXINT1:
        CPL     P0.0
        RETI

;-----------------------------------------

        END
```

2. 程序2——演示外部中断1的下降沿中断唤醒掉电模式

C程序:

```
/*---------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ----------------------------------*/
/* --- 演示STC89xx系列单片机外部中断1(下降沿)唤醒掉电模式 -----*/
/* --- Mobile: (86)13922805190 -------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -----------*/
/*---------------------------------------------------------------------------*/

#include "reg51.h"
#include "intrins.h"

//External interrupt0 service routine
void exint1( ) interrupt 2                  //INT1, interrupt 2 (location at 0013H)
{
}

void main()
{
    IT1 = 1;                                //set INT1 interrupt type (1:Falling 0:Low level)
    EX1 = 1;                                //enable INT1 interrupt
    EA = 1;                                 //open global interrupt switch

    while (1)
    {
        INT1 = 1;                           //ready read INT1 port
        while (!INT1);                      //check INT1
        _nop_();
        _nop_();
        PCON = 0x02;                        //MCU power down
        _nop_();
        _nop_();
        P1++;
    }
}
```

汇编程序:

```
/*---------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ----------------------------------*/
/* --- 演示STC89xx系列单片机外部中断1(下降沿)唤醒掉电模式 ----*/
/* --- Mobile: (86)13922805190 -------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -----------*/
/*---------------------------------------------------------------------------*/

;-----------------------------------------
;interrupt vector table

        ORG     0000H
        LJMP    MAIN

        ORG     0013H                   ;INT1, interrupt 2 (location at 0013H)
        LJMP    EXINT1

;-----------------------------------------

        ORG     0100H
MAIN:
        MOV     SP,     #7FH            ;initial SP
        SETB    IT1                     ;set INT1 interrupt type (1:Falling 0:Low level)
        SETB    EX1                     ;enable INT1 interrupt
        SETB    EA                      ;open global interrupt switch
LOOP:
        SETB    INT1                    ;ready read INT1 port
        JNB     INT1    ,$              ;check INT1
        NOP
        NOP
        MOV     PCON,   #02H            ;MCU power down
        NOP
        NOP
        CPL     P1.0
        SJMP    LOOP

;-----------------------------------------
;External interrupt1 service routine

EXINT1:
        RETI

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

6.6.3 外部中断2($\overline{\text{INT2}}$)的测试程序(C程序及汇编程序)

1. 程序1——演示外部中断2的下降沿中断

C程序:

```
/*---------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ------------------------------------*/
/* --- 演示STC89xx系列单片机外部中断2(下降沿) ----------------------*/
/* --- Mobile: (86)13922805190 ---------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序，  ----------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -----------*/
/*---------------------------------------------------------------------------------------*/

#include "reg51.h"

sfr P4    = 0xe8;                       //for 90C58AD series, location at 0C0H
sbit INT2 = P4^3;
sbit INT3 = P4^2;

sfr  XICON = 0xc0;                      //for 90C58AD series, location at 0E8H
sbit PX3  = XICON^7;
sbit EX3  = XICON^6;
sbit IE3  = XICON^5;
sbit IT3  = XICON^4;
sbit PX2  = XICON^3;
sbit EX2  = XICON^2;
sbit IE2  = XICON^1;
sbit IT2  = XICON^0;

//External interrupt2 service routine
void exint2() interrupt 6          //INT2, interrupt 6 (location at 0033H)
{
   P0++;
}

void main()
{
   IT2 = 1;                  //set INT2 interrupt type (1:Falling only 0:Low level)
   EX2 = 1;                   //enable INT2 interrupt
   EA = 1;                   //open global interrupt switch

   while (1);
}
```

汇编程序:

```
/*---------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ----------------------------------*/
/* --- 演示STC89xx系列单片机外部中断2(下降沿) ----------------------*/
/* --- Mobile: (86)13922805190 -------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 ----------*/
/*---------------------------------------------------------------------------*/
P4      EQU     0E8H            ;for 90C58AD series, location at 0C0H
INT2    BIT     P4.3
INT3    BIT     P4.2

XICON   EQU     0C0H            ;for 90C58AD series, location at 0E8H
PX3     BIT     XICON.7
EX3     BIT     XICON.6
IE3     BIT     XICON.5
IT3     BIT     XICON.4
PX2     BIT     XICON.3
EX2     BIT     XICON.2
IE2     BIT     XICON.1
IT2     BIT     XICON.0
;-----------------------------------------
;interrupt vector table
        ORG     0000H
        LJMP    MAIN

        ORG     0033H           ;INT2, interrupt 6 (location at 0033H)
        LJMP    EXINT2
;-----------------------------------------
        ORG     0100H
MAIN:
        MOV     SP,     #7FH    ;initial SP
        SETB    IT2             ;set INT2 interrupt type (1:Falling 0:Low level)
        SETB    EX2             ;enable INT2 interrupt
        SETB    EA              ;open global interrupt switch
        SJMP    $
;-----------------------------------------
;External interrupt2 service routine
EXINT2:
        CPL     P0.0
        RETI
;-----------------------------------------
        END
```

2. 程序2——演示外部中断2的下降沿中断唤醒掉电模式

C程序：

```
/*---------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ----------------------------------*/
/* --- 演示STC89xx系列单片机外部中断2(下降沿)唤醒掉电模式 -----*/
/* --- Mobile: (86)13922805190 -------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -----------*/
/*---------------------------------------------------------------------------*/

#include "reg51.h"
#include "intrins.h"

sfr   P4    = 0xe8;                   //for 90C58AD series, location at 0C0H
sbit INT2 = P4^3;
sbit INT3 = P4^2;

sfr   XICON         = 0xc0;           //for 90C58AD series, location at 0E8H
sbit PX3            = XICON^7;
sbit EX3            = XICON^6;
sbit IE3            = XICON^5;
sbit IT3            = XICON^4;
sbit PX2            = XICON^3;
sbit EX2            = XICON^2;
sbit IE2            = XICON^1;
sbit IT2            = XICON^0;

//External interrupt2 service routine
void exint2() interrupt 6             //INT2, interrupt 6 (location at 0033H)
{
}

void main()
{
        IT2 = 1;                      //set INT2 interrupt type (1:Falling 0:Low level)
        EX2 = 1;                      //enable INT2 interrupt
        EA = 1;                       //open global interrupt switch
```

```
        while (1)
        {
                INT2 = 1;               //ready read INT2 port
                while (!INT2);          //check INT2
                _nop_();
                _nop_();
                PCON = 0x02;            //MCU power down
                _nop_();
                _nop_();
                P1++;
        }
}
```

汇编程序:

```
/*---------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ----------------------------------*/
/* --- 演示STC89xx系列单片机外部中断2(下降沿)唤醒掉电模式 -----*/
/* --- Mobile: (86)13922805190 -------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序，  ----------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -----------*/
/*---------------------------------------------------------------------------*/

P4      EQU     0E8H            ;for 90C58AD series, location at 0C0H
INT2    BIT     P4.3
INT3    BIT     P4.2

XICON   EQU     0C0H            ;for 90C58AD series, location at 0E8H
PX3     BIT     XICON.7
EX3     BIT     XICON.6
IE3     BIT     XICON.5
IT3     BIT     XICON.4
PX2     BIT     XICON.3
EX2     BIT     XICON.2
IE2     BIT     XICON.1
IT2     BIT     XICON.0

;------------------------------------------
;interrupt vector table

        ORG     0000H
        LJMP    MAIN
```

```
        ORG     0033H                   ;INT2, interrupt 6 (location at 0033H)
        LJMP    EXINT2
;-----------------------------------------

        ORG     0100H
MAIN:
        MOV     SP,     #7FH            ;initial SP
        SETB    IT2                     ;set INT2 interrupt type (1:Falling 0:Low level)
        SETB    EX2                     ;enable INT2 interrupt
        SETB    EA                      ;open global interrupt switch
LOOP:
        SETB    INT2                    ;ready read INT2 port
        JNB     INT2,   $               ;check INT2
        NOP
        NOP
        MOV     PCON,   #02H            ;MCU power down
        NOP
        NOP
        CPL     P1.0
        SJMP    LOOP

;-----------------------------------------
;External interrupt2 service routine
EXINT2:
        RETI
;-----------------------------------------

        END
```

6.6.4 外部中断3($\overline{INT3}$)的测试程序（C程序及汇编程序）

1. 程序1——演示外部中断3的下降沿中断

C程序：

```
/*---------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ----------------------------------*/
/* --- 演示STC89xx系列单片机外部中断3(下降沿) ----------------------*/
/* --- Mobile: (86)13922805190 -------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -----------*/
/*---------------------------------------------------------------------------*/

#include "reg51.h"

sfr P4      = 0xe8;                 //for 90C58AD series, location at 0C0H
sbit INT2 = P4^3;
sbit INT3 = P4^2;

sfr   XICON = 0xc0;                 //for 90C58AD series, location at 0E8H
sbit PX3  = XICON^7;
sbit EX3  = XICON^6;
sbit IE3   = XICON^5;
sbit IT3   = XICON^4;
sbit PX2  = XICON^3;
sbit EX2  = XICON^2;
sbit IE2   = XICON^1;
sbit IT2   = XICON^0;

//External interrupt3 service routine
void exint3() interrupt 7          //INT3, interrupt 7 (location at 003BH)
{
   P0++;
}

void main()
{
   IT3     = 1;                    //set INT3 interrupt type (1:Falling only 0:Low level)
   EX3     = 1;                    //enable INT3 interrupt
   EA      = 1;                    //open global interrupt switch

   while (1);
}
```

汇编程序：

```
/*---------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ----------------------------------------*/
/* --- 演示STC89xx系列单片机外部中断3(下降沿) ----------------------*/
/* --- Mobile: (86)13922805190 -------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序，  ----------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -----------*/
/*---------------------------------------------------------------------------------*/
P4      EQU     0E8H            ;for 90C58AD series, location at 0C0H
INT2    BIT     P4.3
INT3    BIT     P4.2

XICON   EQU     0C0H            ;for 90C58AD series, location at 0E8H
PX3     BIT     XICON.7
EX3     BIT     XICON.6
IE3     BIT     XICON.5
IT3     BIT     XICON.4
PX2     BIT     XICON.3
EX2     BIT     XICON.2
IE2     BIT     XICON.1
IT2     BIT     XICON.0
;-----------------------------------------
;interrupt vector table
        ORG     0000H
        LJMP    MAIN

        ORG     003BH           ;INT3, interrupt 7 (location at 003BH)
        LJMP    EXINT3

;-----------------------------------------

        ORG     0100H
MAIN:
        MOV     SP,#7FH                 ;initial SP
        SETB    IT3                     ;set INT3 interrupt type (1:Falling 0:Low level)
        SETB    EX3                     ;enable INT3 interrupt
        SETB    EA                      ;open global interrupt switch
        SJMP    $
;-----------------------------------------
;External interrupt3 service routine

EXINT3:
        CPL     P0.0
        RETI
;-----------------------------------------
        END
```

2. 程序2——演示外部中断3的下降沿中断唤醒掉电模式

C程序：

```
/*---------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ----------------------------------*/
/* --- 演示STC89xx系列单片机外部中断3(下降沿)唤醒掉电模式 -----*/
/* --- Mobile: (86)13922805190 -------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -----------*/
/*---------------------------------------------------------------------------*/

#include "reg51.h"
#include "intrins.h"

sfr    P4     = 0xe8;              //for 90C58AD series, location at 0C0H
sbit  INT2 = P4^3;
sbit  INT3 = P4^2;

sfr   XICON = 0xc0;                //for 90C58AD series, location at 0E8H
sbit PX3      = XICON^7;
sbit EX3      = XICON^6;
sbit IE3      = XICON^5;
sbit IT3      = XICON^4;
sbit PX2      = XICON^3;
sbit EX2      = XICON^2;
sbit IE2      = XICON^1;
sbit IT2      = XICON^0;

//External interrupt3 service routine
void exint3() interrupt 7          //INT3, interrupt 7 (location at 003BH)
{
}

void main()
{
        IT3 = 1;                        //set INT3 interrupt type (1:Falling 0:Low level)
        EX3 = 1;                        //enable INT3 interrupt
        EA = 1;                         //open global interrupt switch
```

```
        while (1)
        {
                INT3 = 1;                       //ready read INT3 port

                while (!INT3);                  //check INT3
                _nop_();
                _nop_();
                PCON = 0x02;                    //MCU power down
                _nop_();
                _nop_();
                P1++;
        }
}
```

汇编程序：

```
/*-------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ------------------------------------*/
/* --- 演示STC89xx系列单片机外部中断3(下降沿)唤醒掉电模式 -----*/
/* --- Mobile: (86)13922805190 -------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -----------*/
/*-------------------------------------------------------------------------------*/
P4      EQU     0E8H            ;for 90C58AD series, location at 0C0H
INT2    BIT     P4.3
INT3    BIT     P4.2

XICON   EQU     0C0H            ;for 90C58AD series, location at 0E8H
PX3     BIT     XICON.7
EX3     BIT     XICON.6
IE3     BIT     XICON.5
IT3     BIT     XICON.4
PX2     BIT     XICON.3
EX2     BIT     XICON.2
IE2     BIT     XICON.1
IT2     BIT     XICON.0

;-----------------------------------------
;interrupt vector table

        ORG     0000H
        LJMP    MAIN
```

```
        ORG     003BH               ;INT3, interrupt 7 (location at 003BH)
        LJMP    EXINT3

;---------------------------------------

        ORG     0100H
MAIN:
        MOV     SP,     #7FH        ;initial SP
        SETB    IT3                 ;set INT3 interrupt type (1:Falling 0:Low level)
        SETB    EX3                 ;enable INT3 interrupt
        SETB    EA                  ;open global interrupt switch
LOOP:
        SETB    INT3                ;ready read INT3 port
        JNB     INT3,   $           ;check INT3
        NOP
        NOP
        MOV     PCON,#02H           ;MCU power down
        NOP
        NOP
        CPL     P1.0
        SJMP    LOOP

;---------------------------------------
;External interrupt3 service routine

EXINT3:
        RETI

;---------------------------------------

        END
```

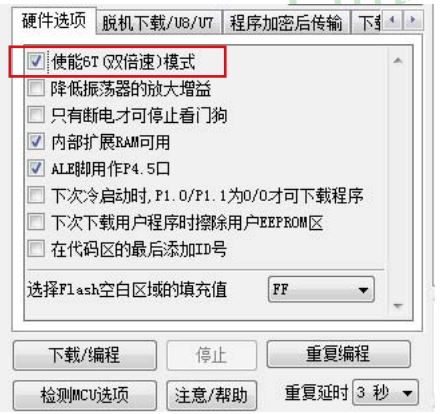
第7章 定时器/计数器

7.1 定时器/计数器0/1

STC89C51RC/RD+系列单片机的定时器0和定时器1，与传统8051的定时器完全兼容，当在定时器1做波特率发生器时，定时器0可以当两个8位定时器用。

STC89C51RC/RD+系列单片机内部设置的两个16位定时器/计数器T0和T1都具有计数方式和定时方式两种工作方式。对每个定时器/计数器(T0和T1)，在特殊功能寄存器TMOD中都有一控制位— C/$\overline{T}$来选择T0或T1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(也有减法)的计数器，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6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)，则为计数方式，每来一个脉冲加1。

当定时器/计数器工作在定时模式时，可在烧录用户程序时在STC-ISP编程器中设置(如下图所示)是系统时钟/12还是系统时钟/6后让T0和T1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(13位定时器/计数器)，模式1(16位定时器/计数器模式)，模式2(8位自动重装模式)，模式3(两个8位定时器/计数器)。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。

7.1.1 定时器/计数器0和1的相关寄存器

符号	描述	地址	位地址及其符号 MSB							LSB	复位值
TCON	定时器控制寄存器	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000 0000B
TMOD	定时器模式寄存器	89H	GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	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

1. 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：定时器/计数器T1溢出标志。T1被允许计数以后，从初值开始加1计数。当最高位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”TF1（TF1也可由程序查询清“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$\overline{\text{INT1}}$输入高电平时，才允许T1计数。

TF0：定时器/计数器T0溢出中断标志。T0被允许计数以后，从初值开始加1计数，当最高位产生溢出时，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”TF0（ TF0也可由程序查询清“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$\overline{\text{INT0}}$输入高电平时，才允许T0计数。

IE1：外部中断1请求源（$\overline{\text{INT1}}$/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1触发方式控制位。IT1=0时，外部中断1为低电平触发方式，当$\overline{\text{INT1}}$（P3.3）输入低电平时，置位IE1 。采用低电平触发方式时，外部中断源（输入到$\overline{\text{INT1}}$）必须保持低电平有效，直到该中断被CPU 响应，同时在该中断服务程序执行完之前，外部中断源必须被清除(P3.3要变高)，否则将产生另一次中断。当IT1=1时，则外部中断1($\overline{\text{INT1}}$)端口由“1”→“0”下降沿跳变，激活中断请求标志位IE1 ，向主机请求中断处理。

IE0：外部中断0请求源（$\overline{\text{INT0}}$/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0触发方式控制位。IT0=0时，外部中断0为低电平触发方式，当$\overline{\text{INT0}}$（P3.2）输入低电平时，置位IE0。采用低电平触发方式时，外部中断源（输入到$\overline{\text{INT0}}$）必须保持低电平有效，直到该中断被CPU响应，同时在该中断服务程序执行完之前，外部中断源必须被清除（P3.2要变高），否则将产生另一次中断。当IT0=1时，则外部中断0($\overline{\text{INT0}}$)端口由“1”→“0”下降沿跳变，激活中断请求标志位IE0 ，向主机请求中断处理。

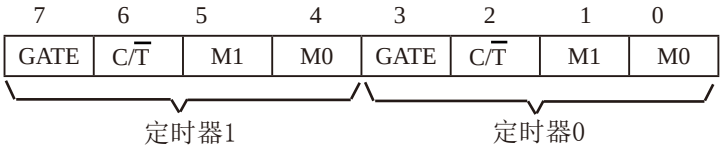
2. 定时器/计数器工作模式寄存器TMOD

定时和计数功能由特殊功能寄存器TMOD的控制位C/$\overline{T}$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，各模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$\overline{INT1}$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$\overline{INT0}$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/$\overline{T}$	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(从内部系统时钟输入)，置1用作计数器(从T1/P3.5脚输入)
TMOD.2/	C/$\overline{T}$	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(从内部系统时钟输入)，置1用作计数器(从T0/P3.4脚输入)
TMOD.5/TMOD.4	M1、M0	定时器定时器/计数器1模式选择
	0 0	13位定时器/计数器，兼容8048定时模式，TL1只用低5位参与分频，TH1整个8位全用。
	0 1	16位定时器/计数器，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	13位定时器/计数器，兼容8048定时模式，TL0只用低5位参与分频，TH0整个8位全用。
	0 1	16位定时器/计数器，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	定时器0此时作为双8位定时器/计数器。TL0作为一个8位定时器/计数器，通过标准定时器0的控制位控制。TH0仅作为一个8位定时器,由定时器1的控制位控制。

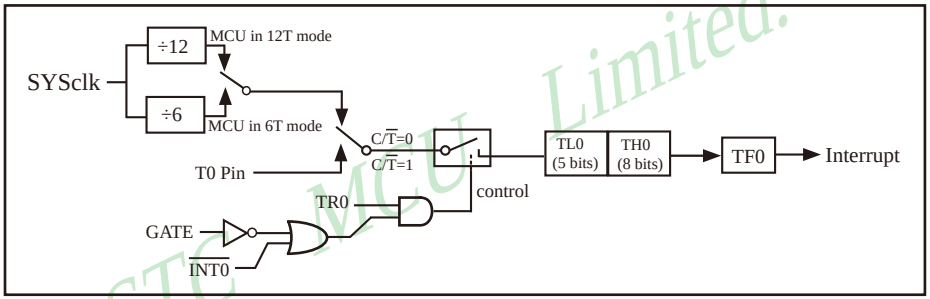
7.1.2 定时器/计数器0工作模式(与传统8051单片机兼容)

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

7.1.2.1 模式0(13位定时器/计数器)

将定时器设置成模式0时类似8048定时器，即8位计数器带32分频的预分频器。下图所示为**定时器/计数器的模式0工作方式。此模式下，定时器0配置为13位的计数器，由TL0的低5位**和TH0的8位所构成。TL0低5位溢出向TH0进位，TH0计数溢出置位TCON中的溢出标志位TF0。GATE(TMOD.3)=0 时，如TR0=1，则定时器计数。GATE=1时，允许由外部输入$\overline{INT1}$控制定时器1，$\overline{INT0}$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TCON寄存器各位的具体功能描述表。

在模式0下定时器/**计数器0作为13位定时器/计数器，如下图所示。**



定时器/计数器0的模式 0: 13位定时器/计数器

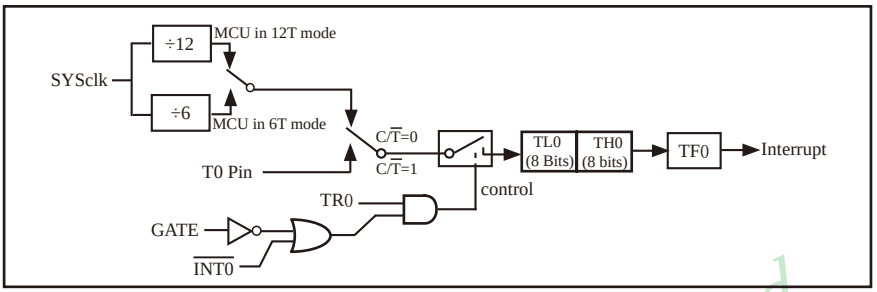
当C/$\overline{T}$=0时，多路开关连接到系统时钟的分频输出，T0对时钟周期计数，T0工作在定时方式。当C/$\overline{T}$=1时，多路开关连接到外部脉冲输入P3.4/T0，即T0工作在计数方式。

STC89C51RC/RD+**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6T模式，每6个时钟加1，速度是传统8051单片机的2倍。**T0的速率在烧录用户程序时在STC-ISP编程器中设置。

该**模式下的13位寄存器包含TH0全部8个位及TL0的低5位。TL0的高3位不定，可将其忽略。**置位运行标志（TR0）不能清零此寄存器。模式0的操作对于定时器0及定时器1都是相同的。2个不同的GATE位（TMOD.7和TMOD.3）分别分配给定时器1及定时器0。

7.1.2.2 模式1(16位定时器/计数器)及其测试程序(C程序及汇编程序)

模式1除了使用了TH0及TL0全部16位外，其他与模式0完全相同。即此模式下定时器/计数器0作为16位定时器/计数器，如下图所示。



定时器/计数器0的模式 1: 16位定时器/计数器

此模式下，定时器配置为16位定时器/计数器，由TL0的8位和TH0的8位所构成。TL0的8位溢出向TH0进位，TH0计数溢出置位TCON中的溢出标志位TF0。

当GATE=0(TMOD.3)时，如TR0=1，则定时器计数。GATE=1时，允许由外部输入INT0控制定时器0，这样可实现脉宽测量。TR0为TCON寄存器内的控制位，TCON寄存器各位的具体功能描述见上节TCON寄存器的介绍。

当C/$\overline{T}$=0时，多路开关连接到系统时钟的分频输出，T0对时钟周期计数，T0工作在定时方式。当C/$\overline{T}$=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。

STC89C51RC/RD+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6T模式，每6个时钟加1，速度是传统8051单片机的2倍。T0的速率在烧录用户程序时在STC-ISP编程器中设置。

定时器0工作在16位定时器/计数器模式的测试程序

1. C程序：

```
/*---------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ----------------------------------*/
/* --- 演示STC89xx系列单片机定时器0的16位定时器/计数器模式 --*/
/* --- Mobile: (86)13922805190 -------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -----------*/
/*---------------------------------------------------------------------------*/
#include "reg51.h"

typedef unsigned char BYTE;
typedef unsigned int WORD;

//-----------------------------------------

/* define constants */
#define    FOSC     18432000L

#define    T1MS     (65536-FOSC/12/1000)          //1ms timer calculation method in 12T mode

/* define SFR */
sbit TEST_LED = P1^0;                             //work LED, flash once per second

/* define variables */
WORD count;                                       //1000 times counter

//-----------------------------------------

/* Timer0 interrupt routine */
void tm0_isr() interrupt 1 using 1
{
        TL0 = T1MS;                               //reload timer0 low byte
        TH0 = T1MS >> 8;                          //reload timer0 high byte
        if (count-- == 0)                         //1ms * 1000 -> 1s
        {
                count = 1000;                     //reset counter
                TEST_LED = ! TEST_LED;            //work LED flash
        }
}

//-----------------------------------------
```

```
/* main program */
void main()
{
        TMOD = 0x01;                    //set timer0 as mode1 (16-bit)
        TL0 = T1MS;                     //initial timer0 low byte
        TH0 = T1MS >> 8;                //initial timer0 high byte
        TR0 = 1;                        //timer0 start running
        ET0 = 1;                        //enable timer0 interrupt
        EA = 1;                         //open global interrupt switch
        count = 0;                      //initial counter

        while (1);                      //loop
}
```

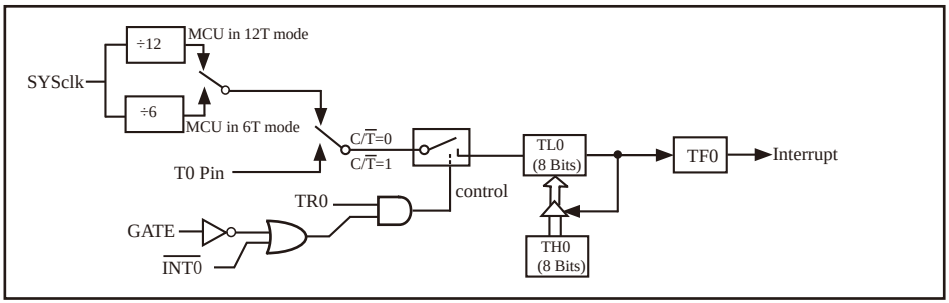
2. 汇编程序：

```
/*------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ------------------------------------*/
/* --- 演示STC89xx系列单片机定时器0的16位定时器/计数器模式 ---*/
/* --- Mobile: (86)13922805190 -------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -----------*/
/*------------------------------------------------------------------------------*/
;/* define constants */
T1MS      EQU     0FA00H            ;1ms timer calculation method in 12T mode is (65536-18432000/12/1000)

;/* define SFR */
TEST_LED    BIT  P1.0               ;work LED, flash once per second

;/* define variables */
COUNT  DATA     20H                 ;1000 times counter (2 bytes)

;------------------------------------------------

        ORG     0000H
        LJMP    MAIN
        ORG     000BH
        LJMP    TM0_ISR

;------------------------------------------------
```

```
;/* main program */
MAIN:
        MOV     TMOD,#01H               ;set timer0 as mode1 (16-bit)
        MOV     TL0,#LOW T1MS           ;initial timer0 low byte
        MOV     TH0,#HIGH T1MS          ;initial timer0 high byte
        SETB    TR0                     ;timer0 start running
        SETB    ET0                     ;enable timer0 interrupt
        SETB    EA                      ;open global interrupt switch
        CLR     A
        MOV     COUNT,A
        MOV     COUNT+1,A               ;initial counter
        SJMP    $

;-----------------------------------------------

;/* Timer0 interrupt routine */
TM0_ISR:
        PUSH    ACC
        PUSH    PSW
        MOV     TL0,    #LOW    T1MS            ;reload timer0 low byte
        MOV     TH0,    #HIGH   T1MS            ;reload timer0 high byte
        MOV     A,      COUNT
        ORL     A,      COUNT+1                 ;check whether count(2byte) is equal to 0
        JNZ     SKIP
        MOV     COUNT,  #LOW 1000               ;1ms * 1000 -> 1s
        MOV     COUNT+1,        #HIGH 1000
        CPL     TEST_LED                        ;work LED flash
SKIP:
        CLR     C
        MOV     A,      COUNT                   ;count--
        SUBB    A,      #1
        MOV     COUNT,  A
        MOV     A,      COUNT+1
        SUBB    A,      #0
        MOV     COUNT+1,A
        POP     PSW
        POP     ACC
        RETI

;-----------------------------------------------

        END
```

7.1.2.3 模式2(8位自动重装模式)及其测试程序(C程序及汇编程序)

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。

;定时器0中断的测试程序，定时器0工作在8位自动重装模式

1. C程序：

```
/*---------------------------------------------------------------------------*/
/* --- STC MCU Limited ------------ ----------------------------------*/
/* --- STC89-90xx Series MCU T0(Falling edge) Demo ---------------*/
/* --- Mobile: (86)13922805190 ----------------------------------------*/
/* --- Fax: 86-755-82905966 -------------------------------------------*/
/* --- Tel: 86-755-82948412 -------------------------------------------*/
/* --- Web: www.STCMCU.com --------------------------------------------*/
/* If you want to use the program or the program referenced in the  */
/* article, please specify in which data and procedures from STC    */
/*---------------------------------------------------------------------------*/
#include "reg51.h"

//T0 interrupt service routine
void t0int() interrupt 1          //T0 interrupt (location at 000BH)
{
        P0++;
}
```

```
void main()
{
        TMOD = 0x06;            //set timer0 as counter mode2 (8-bit auto-reload)
        TL0 = TH0 = 0xff;       //fill with 0xff to count one time
        TR0 = 1;                //timer0 start run
        ET0 = 1;                //enable T0 interrupt
        EA = 1;                 //open global interrupt switch

        while (1);
}
```

2. 汇编程序：

```
;/*---------------------------------------------------------------------------*/
;/* --- STC MCU Limited ---------------- ----------------------------------*/
;/* --- STC89-90xx Series MCU T0(Falling edge) Demo ----------------*/
;/* --- Mobile: (86)13922805190 -------------------------------------*/
;/* --- Fax: 86-755-82905966 ----------------------------------------*/
;/* --- Tel: 86-755-82948412 ----------------------------------------*/
;/* --- Web: www.STCMCU.com -----------------------------------------*/
;/* If you want to use the program or the program referenced in the  ---*/
;/* article, please specify in which data and procedures from STC    ----*/
;/*---------------------------------------------------------------------------*/

;----------------------------------------
;interrupt vector table

   ORG    0000H
   LJMP   MAIN

   ORG    000BH                ;T0 interrupt (location at 000BH)
   LJMP   T0INT

;----------------------------------------

   ORG    0100H
```

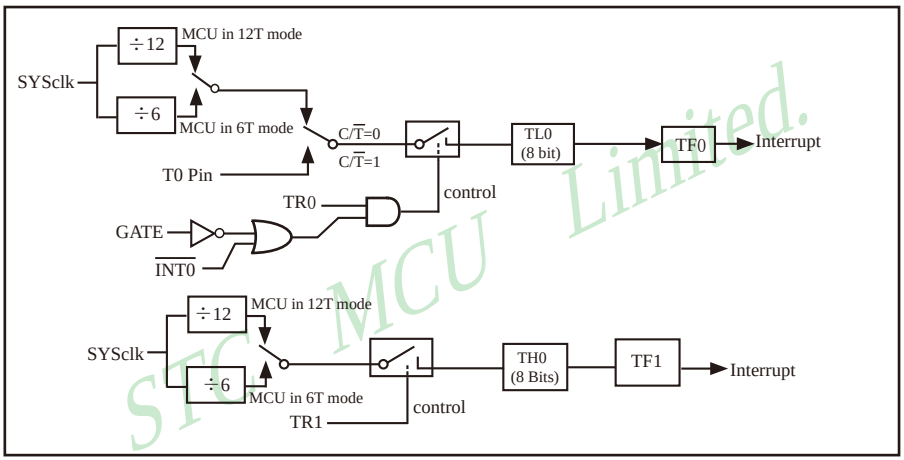
```
MAIN:
        MOV     SP,     #7FH            ;initial SP
        MOV     TMOD,   #06H            ;set timer0 as counter mode2 (8-bit auto-reload)
        MOV     A,      #0FFH
        MOV     TL0,    A               ;fill with 0xff to count one time
        MOV     TH0,    A
        SETB    TR0                     ;timer0 start run
        SETB    ET0                     ;enable T0 interrupt
        SETB    EA                      ;open global interrupt switch
        SJMP    $

;----------------------------------------
;T0 interrupt service routine

T0INT:
        CPL     P0.0
        RETI

;----------------------------------------

        END
```

7.1.2.4 模式3(两个8位计数器)

对定时器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0，此模式下定时器0的TL0及TH0作为2个独立的8位计数器。下图为模式3时的定时器0逻辑图。TL0占用定时器0的控制位：C/$\overline{T}$、GATE、TR0、INT0及TF0。TH0限定为定时器功能（计数器周期），占用定时器1的TR1及TF1。此时，TH0控制定时器1中断。

模式3是为了增加一个附加的8位定时器/计数器而提供的，使单片机具有三个定时器/计数器。模式3只适用于定时器/计数器0，定时器T1处于模式3时相当于TR1=0，停止计数，而T0可作为两个定时器用。



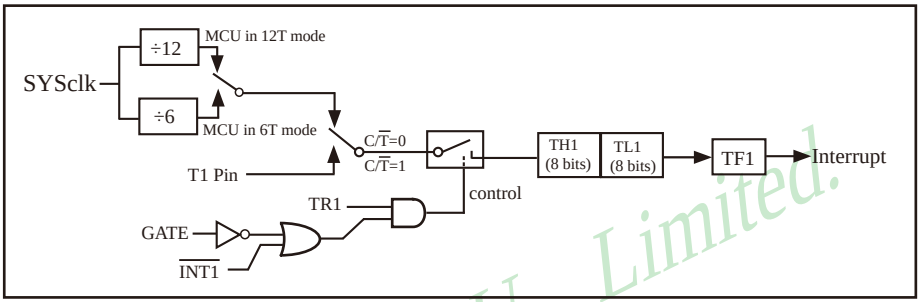
定时/计数器0 模式3: 两个8位计数器

7.1.3 定时器/计数器1工作模式(与传统8051单片机兼容)

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。

7.1.3.1 模式0(13位定时器/计数器)

此模式下定时器/计数器1作为13位定时器/计数器，有TL1的低5位和TH1的8位所构成，如下图所示。模式0的操作对于定时器1和定时器0是相同的。



定时器/计数器1的模式 0: 13位定时器/计数器

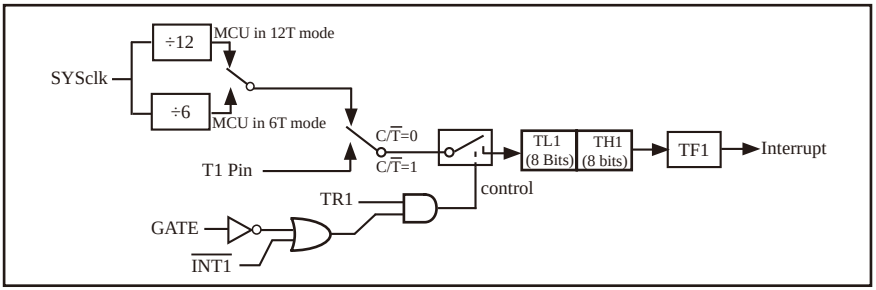
当GATE=0(TMOD.7)时，如TR1=1，则定时器计数。GATE=1时，允许由外部输入$\overline{\text{INT1}}$控制定时器1，这样可实现脉宽测量。TR1为TCON寄存器内的控制位，TCON寄存器各位的具体功能描述见上节TCON寄存器的介绍。

当C/$\overline{\text{T}}$=0时，多路开关连接到系统时钟的分频输出，T1对时钟周期计数，T1工作在定时方式。当C/$\overline{\text{T}}$=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。

STC89C51RC/RD+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6T模式，每6个时钟加1，速度是传统8051单片机的2倍。T1的速率在烧录用户程序时在STC-ISP编程器中设置。

7.1.3.2 模式1(16位定时器/计数器)及其测试程序(C程序及汇编程序)

此模式下定时器/计数器1作为16位定时器/计数器，如下图所示。



定时器/计数器1的模式 1: 16位定时器/计数器

此模式下，定时器1配置为16位定时器/计数器，由TL1的8位和TH1的8位所构成。TL1的8位溢出向TH1进位，TH1计数溢出置位TCON中的溢出标志位TF1。

当GATE=0(TMOD.7)时，如TR1=1，则定时器计数。GATE=1时，允许由外部输入INT1控制定时器1，这样可实现脉宽测量。TR1为TCON寄存器内的控制位，TCON寄存器各位的具体功能描述见上节TCON寄存器的介绍。

当C/$\overline{T}$=0时，多路开关连接到系统时钟的分频输出，T1对时钟周期计数，T1工作在定时方式。当C/$\overline{T}$=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。

STC89C51RC/RD+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6T模式，每6个时钟加1，速度是传统8051单片机的2倍。T1的速率在烧录用户程序时在STC-ISP编程器中设置。

定时器1工作在16位定时器/计数器模式的测试程序

1. C程序：

```
/*---------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ----------------------------------*/
/* --- 演示STC89xx系列单片机定时器1的16位定时器/计数器模式 --*/
/* --- Mobile: (86)13922805190 -------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -----------*/
/*---------------------------------------------------------------------------*/
#include "reg51.h"

typedef    unsigned char       BYTE;
typedef    unsigned int        WORD;

//-----------------------------------------------

/* define constants */
#define    FOSC     18432000L

#define    T1MS (65536-FOSC/12/1000)                //1ms timer calculation method in 12T mode

/* define SFR */
sbit       TEST_LED = P1^0;                         //work LED, flash once per second

/* define variables */
WORD       count;                                   //1000 times counter

//-----------------------------------------------

/* Timer0 interrupt routine */
void tm1_isr() interrupt 3 using 1
{
           TL1 = T1MS;                              //reload timer1 low byte
           TH1 = T1MS >> 8;                         //reload timer1 high byte
           if (count-- == 0)                        //1ms * 1000 -> 1s
           {
                      count = 1000;                 //reset counter
                      TEST_LED = ! TEST_LED;        //work LED flash
           }
}
//-----------------------------------------------
```

```
/* main program */
void main()
{
        TMOD = 0x10;            //set timer1 as mode1 (16-bit)
        TL1 = T1MS;             //initial timer1 low byte
        TH1 = T1MS >> 8;        //initial timer1 high byte
        TR1 = 1;                //timer1 start running
        ET1 = 1;                //enable timer1 interrupt
        EA = 1;                 //open global interrupt switch
        count = 0;              //initial counter

        while (1);              //loop
}
```

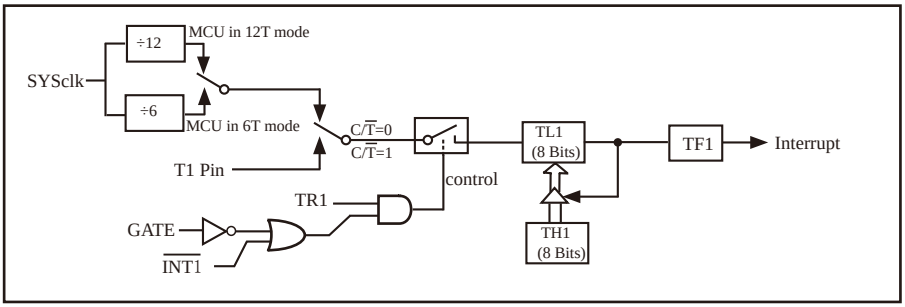
2. 汇编程序：

```
/*------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ----------------------------------*/
/* --- 演示STC89xx系列单片机定时器1的16位定时器/计数器模式 --*/
/* --- Mobile: (86)13922805190 -------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -----------*/
/*------------------------------------------------------------------------------*/
;/* define constants */
T1MS      EQU    0FA00H          ;1ms timer calculation method in 12T mode is (65536-18432000/12/1000)

;/* define SFR */
TEST_LED   BIT  P1.0              ;work LED, flash once per second

;/* define variables */
COUNT  DATA    20H                ;1000 times counter (2 bytes)

;------------------------------------------------

        ORG     0000H
        LJMP    MAIN
        ORG     001BH
        LJMP    TM1_ISR
;------------------------------------------------
```

```
;/* main program */
MAIN:
        MOV     TMOD,   #10H                    ;set timer1 as mode1 (16-bit)
        MOV     TL1,    #LOW T1MS               ;initial timer1 low byte
        MOV     TH1     ,#HIGH T1MS             ;initial timer1 high byte
        SETB    TR1                             ;timer1 start running
        SETB    ET1                             ;enable timer1 interrupt
        SETB    EA                              ;open global interrupt switch
        CLR     A
        MOV     COUNT,  A
        MOV     COUNT+1,        A               ;initial counter
        SJMP    $

;-----------------------------------------------

;/* Timer1 interrupt routine */
TM1_ISR:
        PUSH    ACC
        PUSH    PSW
        MOV     TL1,    #LOW T1MS               ;reload timer1 low byte
        MOV     TH1,    #HIGH T1MS              ;reload timer1 high byte
        MOV     A,      COUNT
        ORL     A,      COUNT+1                 ;check whether count(2byte) is equal to 0
        JNZ     SKIP
        MOV     COUNT,  #LOW 1000               ;1ms * 1000 -> 1s
        MOV     COUNT+1,#HIGH 1000
        CPL     TEST_LED                        ;work LED flash
SKIP:
        CLR     C
        MOV     A,      COUNT                   ;count--
        SUBB    A,      #1
        MOV     COUNT,A
        MOV     A,COUNT+1
        SUBB    A,#0
        MOV     COUNT+1,A
        POP     PSW
        POP     ACC
        RETI

;-----------------------------------------------

         END
```

7.1.3.3 模式2(8位自动重装模式)及其测试程序(C程序及汇编程序)

此模式下定时器/计数器1作为可自动重装载的8位计数器，如下图所示。



定时器/计数器1的模式 2: 8位自动重装

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。

;定时器1中断的测试程序，定时器1工作在8位自动重装模式

1. C程序：

```
/*---------------------------------------------------------------------------*/
/* --- STC MCU Limited ------------------------------------------------------*/
/* --- STC89-90xx Series MCU T1(Falling edge) Demo ---------------------------*/
/* --- Mobile: (86)13922805190 ----------------------------------------------*/
/* --- Fax: 86-755-82905966 -------------------------------------------------*/
/* --- Tel: 86-755-82948412 -------------------------------------------------*/
/* --- Web: www.STCMCU.com --------------------------------------------------*/
/* If you want to use the program or the program referenced in the  */
/* article, please specify in which data and procedures from STC    */
/*---------------------------------------------------------------------------*/

#include "reg51.h"

//T1 interrupt service routine
void t1int() interrupt 3          //T1 interrupt (location at 001BH)
{
   P0++;
}
```

```
void main()
{
   TMOD = 0x60;                    //set timer1 as counter mode2 (8-bit auto-reload)
   TL1 = TH1 = 0xff;               //fill with 0xff to count one time
   TR1 = 1;                        //timer1 start run
   ET1 = 1;                        //enable T1 interrupt
   EA = 1;                         //open global interrupt switch

   while (1);
}
```

2. 汇编程序：

```
;/*---------------------------------------------------------------------------*/
;/* --- STC MCU Limited ---------------- ----------------------------------*/
;/* --- STC89-90xx Series MCU T1(Falling edge) Demo ----------------*/
;/* --- Mobile: (86)13922805190 ------------------------------------*/
;/* --- Fax: 86-755-82905966 ---------------------------------------*/
;/* --- Tel: 86-755-82948412 ---------------------------------------*/
;/* --- Web: www.STCMCU.com ----------------------------------------*/
;/* If you want to use the program or the program referenced in the  */
;/* article, please specify in which data and procedures from STC    */
;/*---------------------------------------------------------------------------*/

;----------------------------------------
;interrupt vector table

   ORG    0000H
   LJMP   MAIN

   ORG    001BH                    ;T1 interrupt (location at 001BH)
   LJMP   T1INT

;----------------------------------------
```

```
        ORG     0100H
MAIN:
        MOV     SP,     #7FH            ;initial SP
        MOV     TMOD,   #60H            ;set timer1 as counter mode2 (8-bit auto-reload)
        MOV     A,      #0FFH
        MOV     TL1,    A               ;fill with 0xff to count one time
        MOV     TH1,    A
        SETB    TR1                     ;timer1 start run
        SETB    ET1                     ;enable T1 interrupt
        SETB    EA                      ;open global interrupt switch
        SJMP

;---------------------------------------
;T1 interrupt service routine

T1INT:
        CPL     P0.0
        RETI

;---------------------------------------

        END
```

7.1.4 古老Intel 8051单片机定时器0/1的应用举例

【例1】 定时/计数器编程，定时/计数器的应用编程主要需考虑：根据应用要求，通过程序初始化，正确设置控制字，正确计算和计算计数初值，编写中断服务程序，适时设置控制位等。通常情况下，设置顺序大致如下：

1)工作方式控制字（TMOD、T2CON）的设置；

2)计数初值的计算并装入THx、TLx、RCAP2H、RCAP2L；

3)中断允许位ETx、EA的设置，使主机开放中断；

4) 启/停位TRx的设置等。

现以定时/计数器0或1为例作一简要介绍。

8051系列单片机的定时器/计数器0或1是以不断加1进行计数的，即属加1计数器，因此，就不能直接将实际的计数值作为计数初值送入计数寄存器THx、TLx中去，而必须将实际计数值以2^8、2^{13}、2^{16}为模求补，以其补码作为计数初值设置THx和TLx。

设：实际计数值为X，计数器长度为n（n=8、13、16），则应装入计数器THx、TLx中的计数初值为2^n-x，式中2^n为取模值。例如，工作方式0的计数长度为13位，则n=13，以2^{13}为模，工作方式1的计数长度为16，则n=16，以2^{16}为模等等。所以，计数初值为（x）= 2^n-x。

对于定时模式，是对机器周期计数，而机器周期与选定的主频密切相关。因此，需根据应用系统所选定的主频计算出机器周期值。现以主频6MHz为例，则机器周期为：

$$\text{一个机器周期}=\frac{12}{\text{主振频率}}=\frac{12}{6\times10^6}\ \mu s=2\ \mu s$$

实际定时时间Tc = x • Tp

式中Tp为机器周期，Tc为所需定时时间，x为所需计数次数。Tp和Tp一般为已知值，在求出Tp后即可求得所需计数值x，再将x求补码，即求得定时计数初值。即

（x）补= 2^n - x

例如，设定时时间Tc = 5ms，机器周期TP = 2μs，可求得定时计数次数

$$x=\frac{5ms}{2\ \mu s}=2500次$$

设选用工作方式1，则n=16，则应设置的定时时间计数初值为：

（x）补= 2^{16} - x=65536-2500=63036，还需将它分解成两个8位十六进制数，分别求得低8位为3CH装入TLx，高8位为F6H装入THx中。

工作方式0、1、2的最大计数次数分别为8192、65536和256。

对外部事件计数模式，只需根据实际计数次数求补后变换成两个十六进制码即可。

【例2】 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：

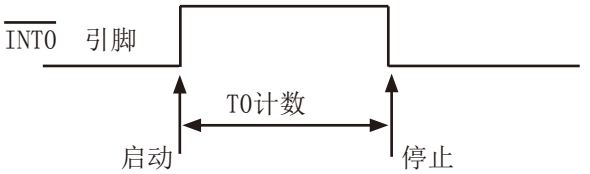
```
START：
        ⋮                          ；主程序段
    MOV     SP，#60H               ；设置堆栈区域
    MOV     TMOD，#10H             ；选择T1、定时模式，工作方式1
    MOV     TH1，#0D8H             ；设置高字节计数初值
    MOV     TL1，#0F0H             ；设置低字节计数初值
    SETB    EA                     ；}
    SETB    ET1                    ；} 开中断
        ⋮                          ；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           ；}
       MOV     TH1,#0D8H           ；} 重新置初值
           ⋮
                                   ；中断处理主体程序
       POP     DPH                 ；
       POP     DPL                 ；} 现场恢复
       POP     A                   ；
       RETI                        ；返回
```

这里展示了中断服务子程序的基本格式。STC89C51RC/RD+系列单片机的中断属于矢量中断，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3】 对外部正脉冲测宽。选择定时/计数器2进行脉宽测试较方便，但也可选用定时/计数器0或定时/计数器1进行测宽操作。本例选用定时/计数器0（T0）以定时模式，工作方式1对INT0引脚上的正脉冲进行脉宽测试。



设置GATE为1，机器周期TP为1μs。本例程序段编制如下：

```
INTT0:    MOV    TMOD，#09H        ；设T0为定时方式1，GATE为1
          MOV    TL0，#00H         ；
          MOV    TH0，#00H         ；} TH0,TL0清0
          CLR    EX0               ；关INT0中断
LOP1:     JB     P3.2，LOP1        ；等待INT0引低电平
LOP2:     JNB    P3.2，LOP2        ；等待INT0引脚高电平
          SETB   TR0               ；启动T0开始计数
LOP3:     JB     P3.2，LOP3        ；等待INT0低电平
          CLR    TR0               ；停止T0计数
          MOV    A，TL0            ；低字节计数值送A
          MOV    B，TH0            ；高字节计数值送B
          ⋮                        ；计算脉宽和处理
```

【例4】 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）的中断矢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    MAIN                ；转主程序
        ORG     001BH
        LJMP    INTER               ；转T1中断服务程序
        ⋮
        ORG     0100                ；主程序入口
MAIN:   …
        ⋮
        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              ;
                POP     DPL              ;  } 现场出栈复原
                POP     A                ;
                RETI                     ; 返回
```

这是中断服务程序的基本格式。

【例5】 某应用系统需通过P1.0和P1.1分别输出周期为200μs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:   MOV     TMOD,#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               ;
        SETB    ET0              ;  } 开中断
        SETB    ET1              ;
        SETB    TR0              ; 启动
        SETB    TR1              ; 启动
         ⋮
```

(2)中断服务程序段

1)

```
INT0P:  ⋮
        ⋮
        MOV     TL0，#9CH                ；重新设置初值
        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                  ；禁止中断
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
       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
         ⋮
```

7.2 定时器/计数器T2

定时器2是一个16位定时/计数器。通过设置特殊功能寄存器T2CON中的C/$\overline{T2}$位，可将其作为定时器或计数器（特殊功能寄存器T2CON的描述如下所示）。

定时器/计数器2的相关寄存器表：

符号	描述	地址	位地址及其符号 MSB LSB								复位值
T2CON	定时器2控制寄存器	C8H	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/$\overline{T2}$	CP/$\overline{RL2}$	0000 0000B
T2MOD	定时器2模式寄存器	C9H	-	-	-	-	-	-	T2OE	DECN	xxxx xx00B
RCAP2L	Timer / Counter 2 Reload/Capture Low Byte	CAH									0000 0000B
RCAP2H	Timer / Counter 2 Reload/Capture High Byte	CBH									0000 0000B
TL2	Timer / Counter 2 Low Byte	CCH									0000 0000B
TH2	Timer/Counter 2 High Byte	CDH									0000 0000B

寄存器T2CON(定时器2的控制寄存器）各位的功能描述

T2CON　地址：0C8H　复位值：00H

可位寻址

7	6	5	4	3	2	1	0
TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/$\overline{T2}$	CP/$\overline{RL2}$

位	符号	功能
T2CON.7/	TF2	定时器2溢出标志。定时器2溢出时置位，必须由软件清除。当RCLK或TCLK=1 时，TF2将不会置位
T2CON.6/	EXF2	定时器2外部标志。当EXEN2=1且T2EX的负跳变产生捕获或重装时，EXF2置位。定时器2中断使能时，EXF2=1将使CPU从中断向量处执行定时器2中断子程序。EXF2位必须用软件清零。在递增/递减计数器模式（DCEN=1）中，EXF2不会引起中断
T2CON.5/	RCLK	接收时钟标志。RCLK置位时，定时器2的溢出脉冲作为串行口模式1和模式3的接收时钟。RCLK=0时，将定时器1的溢出脉冲作为串行口模式1和模式3的接收时钟
T2CON.4/	TCLK	发送时钟标志。TCLK置位时，定时器2的溢出脉冲作为串行口模式1和模式3的发送时钟。TCLK=0时，将定时器1的溢出脉冲作为串行口模式1和模式3发送时钟
T2CON.3/	EXEN2	定时器2外部使能标志。当其置位且定时器2未作为串行口时钟时，允许T2EX的负跳变产生捕获或重装。EXEN2=0时，T2EX的跳变对定时器2无效

位	符号	功能
T2CON.2/	TR2	定时器2 启动/停止控制位。置1 时启动定时器
T2CON.1/	C/$\overline{T2}$	定时器/ 计数器选择。（定时器2 ） 0 = 内部定时器（SYSclk/12 或SYSclk/6） 1 = 外部事件计数器（下降沿触发）
T2CON.0/	CP/$\overline{RL2}$	捕获/重装标志。置位:**EXEN2=1**时,T2EX的负跳变产生捕获. 清零：EXEN2=0时，定时器2溢出或T2EX的负跳变都可使定时器自动重装。当RCLK=1 或TCLK=1 时，该位无效且定时器强制为溢出时自动重装

定时器2有3种操作模式：捕获、自动重新装载（递增或递减计数）和波特率发生器。这3种模式由T2CON中的位进行选择（如下表所列）。

定时器2的工作方式			
RCLK+TCLK	CP/$\overline{RL2}$	TR2	模式
0	0	1	16位自动重装
0	1	1	16位捕获
1	x	1	波特率发生器
x	x	0	（关闭）

T2MOD：定时器/计数器2模式控制寄存器（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
T2MOD	0C9H	name	-	-	-	-	-	-	T2OE	DCEN

T2OE：定时器2 输出使能位

DECN：向下计数使能位。定时器2 可配置成向上/ 向下计数器

＊ 用户勿将其置1。这些位在将来80C51 系列产品中用来实现新的特性。在这种情况下，以后用到保留位，复位时或非有效状态时，它的值应为0；而这些位为有效状态时，它的值为1。从保留位读到的值是不确定的。

7.2.1 定时器2的捕获模式

在捕获模式中，通过T2CON中的EXEN2设置2个选项。如果EXEN2=0，定时器2作为一个16位定时器或计数器（由T2CON中C/T2位选择），溢出时置位TF2（定时器2溢出标志位）。该位可用于产生中断（通过使能IE寄存器中的定时器2中断使能位ET2）。如果EXEN2=1，与以上描述相同，但增加了一个特性，即外部输入T2EX由1变零时，将定时器2中TL2和TH2的当前值各自捕获到RCAP2L和RCAP2H。另外，T2EX的负跳变使T2CON中的EXF2置位，EXF2也像TF2一样能够产生中断（其向量与定时器2溢出中断地址相同，定时器2中断服务程序通过查询TF2和EXF2来确定引起中断的事件），捕获模式如下图所示。在该模式中，TL2和TH2无重新装载值，甚至当T2EX产生捕获事件时，计数器仍以T2EX 的负跳变或振荡频率的1/12（12时钟模式）或1/6（6时钟模式）计数。

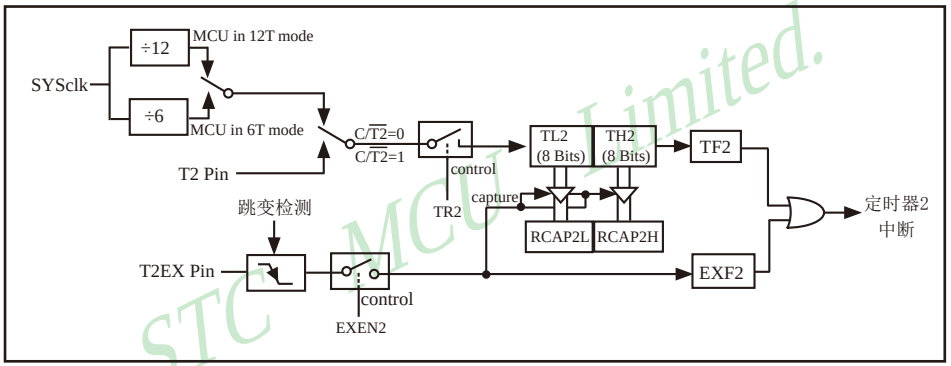


图1 定时器2捕获模式

7.2.2 定时器2的自动重装模式(递增/递减计数器)

16位自动重装模式中，定时器2 可通过C/T2配置为定时器/计数器，编程控制递增/递减计数。计数的方向是由DCEN（递减计数使能位）确定的，DCEN位于T2MOD寄存器中，T2MOD寄存器各位的功能描述如表3 所示。当DCEN=0 时，定时器2 默认为向上计数；当DCEN=1 时，定时器2 可通过T2EX 确定递增或递减计数。图2 显示了当DCEN=0 时，定时器2 自动递增计数。在该模式中，通过设置EXEN2位进行选择。如果EXEN2=0，定时器2 递增计数到0FFFFH，并在溢出后将TF2置位，然后将RCAP2L和RCAP2H 中的16位值作为重新装载值装入定时器2。RCAP2L和RCAP2H的值是通过软件预设的。

如果EXEN2=1，16位重新装载可通过溢出或T2EX从1到0的负跳变实现。此负跳变同时EXF2置位。如果定时器2中断被使能，则当TF2或EXF2置1时产生中断。在图3中，DCEN=1时，定时器2可增或递减计数。此模式允许T2EX 控制计数的方向。当T2EX 置1 时，定时器2 递增计数，计数到0FFFFH后溢出并置位TF2，还将产生中断（如果中断被使能）。定时器2的溢出将使RCAP2L 和RCAP2H中的16 位值作为重新装载值放入TL2和TH2。

当T2EX置零时，将使定时器2递减计数。当TL2和TH2计数到等于RCAP2L和RCAP2H时，定时器产生中断。

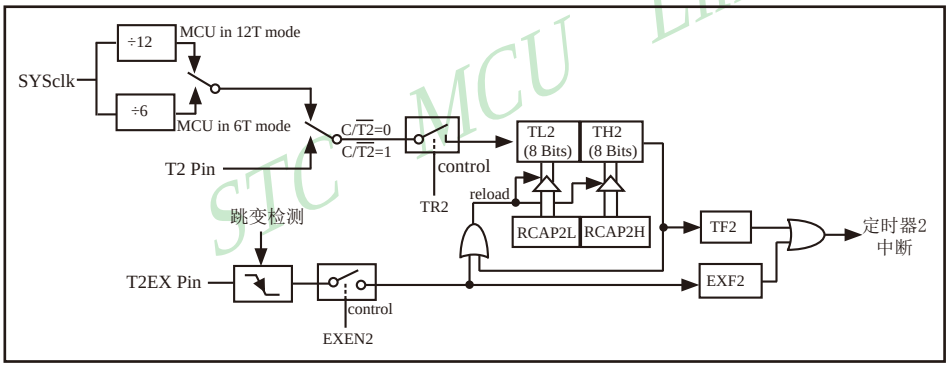


图2 定时器2的自动重装模式 (DCEN=0)

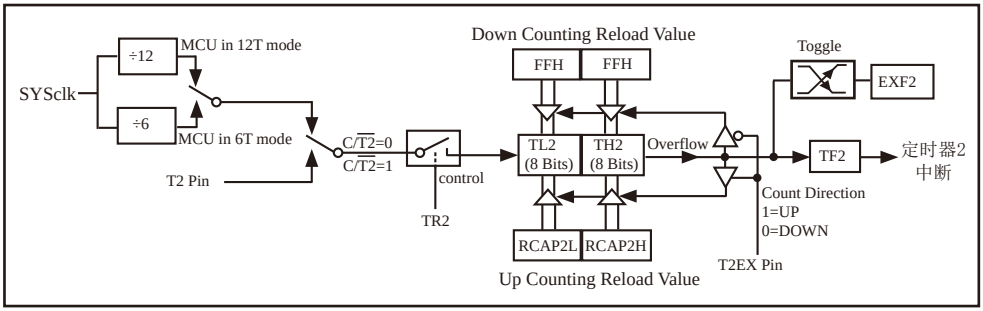


图3 定时器2的自动重装模式 (DCEN=1)

符号	描述	地址	位地址及其符号 MSB LSB								复位值
T2CON	定时器2控制寄存器	C8H	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/$\overline{T2}$	CP/$\overline{RL2}$	0000 0000B
T2MOD	定时器2模式寄存器	C9H	-	-	-	-	-	-	T2OE	DECN	xxxx xx00B

除了波特率发生器模式,T2CON不包括TR2位的设置，TR2位需单独设置来启动定时器。如下表列出了T2作为定时器和计数器的具体设置方法。

T2作定时器T2CON的设置		
模式	T2CON	
	内部控制	外部控制
16位重装	0000 0000B / 00H	0000 1000B / 08H
16位捕获	0000 0001B / 01H	0000 1001B / 09H
波特率发生器接收和发送相同波特率	0011 0100B / 34H	0011 0110B / 36H
只接收	0010 0100B / 24H	0010 0110B / 26H
只发送	0001 0100B / 14H	0001 0110B / 16H

T2作计数器T2MOD的设置		
模式	T2MOD	
	内部控制	外部控制
16位	0000 0010B / 02H	0000 1010B / 0AH
自动重装	0000 0011B / 03H	0000 1011B / 0BH

（1）内部控制：仅当定时器溢出时进行捕获和重装。

（2）外部控制：当定时/计数器溢出并且T2EX(P1.1)发生电平负跳变时产生捕获和重装(定时器2用于波特率发生器模式时除外)。

7.2.3 定时器2作串行口波特率发生器及其测试程序(C程序及汇编程序)

寄存器T2CON 的位TCLK和（或）RCLK允许从定时器1或定时器2获得串行口发送和接收的波特率。当TCLK=0时，定时器1作为串行口发送波特率发生器；当TCLK=1时，定时器2作为串行口发送波特率发生器。RCLK对串行口接收波特率有同样的作用。通过这2位，串行口能得到不同的接收和发送波特率，一个通过定时器1产生，另一个通过定时器2产生。如图4所示为定时器2工作在波特率发生器模式。与自动重装模式相似，当TH2溢出时，波特率发生器模式使定时器2寄存器重新装载来自寄存器RCAP2H 和RCAP2L的16位的值，寄存器RCAP2H和RCAP2L的值由软件预置。当工作于模式1和模式3时，波特率由下面给出的公式所决定：

模式1 和模式3 的波特率=（定时器2溢出速率）/16

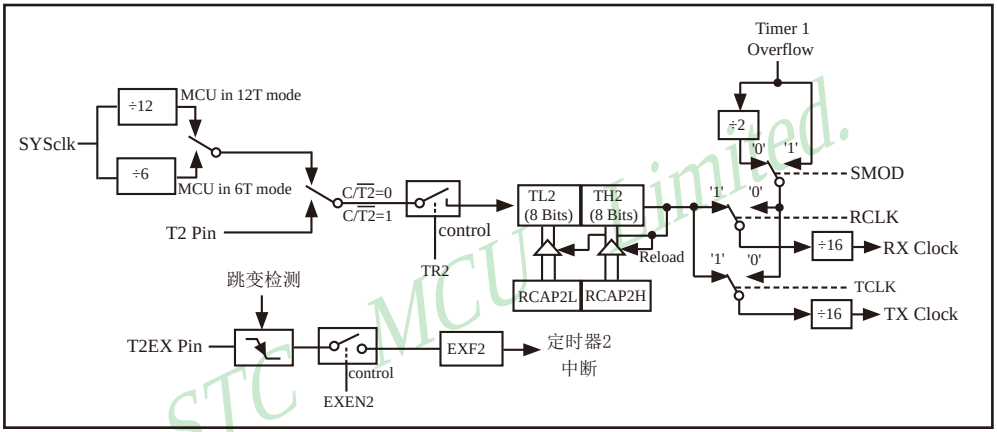


图4 定时器2的独立波特率发生器模式

定时器可配置成“定时”或“计数”方式，在许多应用上，定时器被设置在“定时”方式（C/T2=0）。当定时器2作为定时器时，它的操作不同于波特率发生器。通常定时器2作为定时器，它会在每个机器周期递增（1/6 或1/12 振荡频率）。当定时器2 作为波特率发生器时，它在6 时钟模式下，以振荡器频率递增（12时钟模式时为1/12振荡频率）。

这时的波特率公式如下：

$$\text{模式1和模式3的波特率} = \frac{\text{振荡器频率}}{n\times[65536-(RCAP2H,RCAP2L)]}$$

式中：n=16（6时钟模式）或32（12时钟模式）；[RCAP2H，RCAP2L]是RCAP2H和RCAP2L的内容，为16 位无符号整数。

如图4所示，定时器2是作为波特率发生器，仅当寄存器T2CON中的RCLK和（或）TCLK=1时，定时器2作为波特率发生器才有效。注意：TH2溢出并不置位TF2，也不产生中断。这样当定时器2 作为波特率发生器时，定时器2中断不必被禁止。如果EXEN2（T2外部使能标志）被置位，在T2EX中由1 到0 的转换会置位EXF2（T2 外部标志位），但并不导致（TH2，TL2）重新装载（RCAP2H，RCAP2L）。

当定时器2用作波特率发生器时，如果需要，T2EX可用做附加的外部中断。当计时器工作在波特率发生器模式下，则不要对TH2和TL2 进行读/ 写，每隔一个状态时间（fosc/2)或由T2 进入的异步信号，定时器2 将加1。在此情况下对TH2 和TH1 进行读/ 写是不准确的；可对RCAP2寄存器进行读，但不要进行写，否则将导致自动重装错误。当对定时器2或寄存器RCAP进行访问时，应关闭定时器（清零TR2）。表4列出了常用的波特率和如何用定时器2得到这些波特率。

波特率公式汇总

定时器2工作在波特率发生器模式，外部时钟信号由T2脚进入，这时的波特率公式如下：

模式1和模式3的波特率=（定时器2溢出速率）/16

如果定时器2采用内部时钟信号，则波特率公式如下：

$$\text{波特率} = \frac{\text{SYSclk}}{n\times[65536-(\text{RCAP2H,RCAP2L})]}$$

式中:n=32(12时钟模式)或16(6 时钟模式),SYSclk= 振荡器频率。
自动重装值可由下式得到:

RCAP2H,RCAP2L= 65536 — [SYSclk/(n ×波特率)]

定时器2作串行口波特率发生器的测试程序(C程序及汇编程序)

1. C程序：

```
/*---------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ------------------------------------*/
/* --- 演示STC89xx系列单片机定时器2作波特率发生器测试程序---*/
/* --- Mobile: (86)13922805190 -------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序，  ----------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -----------*/
/*---------------------------------------------------------------------------------------*/


#include "reg51.h"
#include "intrins.h"

sfr T2CON  = 0xC8;                      //timer2 control register
sfr RCAP2L = 0xCA;
sfr RCAP2H = 0xCB;
sfr TL2    = 0xCC;
sfr TH2    = 0xCD;

typedef unsigned char BYTE;
typedef unsigned int WORD;

#define FOSC 18432000L                  //System frequency
#define BAUD 115200                     //UART baudrate

/*Define UART parity mode*/
#define NONE_PARITY     0               //None parity
#define ODD_PARITY      1               //Odd parity
#define EVEN_PARITY     2               //Even parity
#define MARK_PARITY     3               //Mark parity
#define SPACE_PARITY    4               //Space parity

#define PARITYBIT EVEN_PARITY           //Testing even parity

sbit bit9 = P2^2;                       //P2.2 show UART data bit9
bit busy;

void SendData(BYTE dat);
void SendString(char *s);
```

```
void main()
{
#if (PARITYBIT == NONE_PARITY)
   SCON = 0x50;          //8-bit variable UART
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
   SCON = 0xda;          //9-bit variable UART, parity bit initial to 1
#elif (PARITYBIT == SPACE_PARITY)
   SCON = 0xd2;          //9-bit variable UART, parity bit initial to 0
#endif

   TL2 = RCAP2L = (65536-(FOSC/32/BAUD)); //Set auto-reload vaule
   TH2 = RCAP2H = (65536-(FOSC/32/BAUD)) >> 8;
   T2CON = 0x34;              //Timer2 start run
   ES = 1;                    //Enable UART interrupt
   EA = 1;                    //Open master interrupt switch

   SendString("STC89-90xx\r\nUart Test !\r\n");
   while(1);
}

/*----------------------------
UART interrupt service routine
----------------------------*/
void Uart_Isr() interrupt 4 using 1
{
   if (RI)
   {
      RI = 0;                 //Clear receive interrupt flag
      P0 = SBUF;              //P0 show UART data
      bit9 = RB8;             //P2.2 show parity bit
   }
   if (TI)
   {
      TI = 0;                 //Clear transmit interrupt flag
      busy = 0;               //Clear transmit busy flag
   }
}

/*----------------------------
Send a byte data to UART
Input: dat (data to be sent)
Output:None
----------------------------*/
```

```
void SendData(BYTE dat)
{
    while (busy);               //Wait for the completion of the previous data is sent
    ACC = dat;                  //Calculate the even parity bit P (PSW.0)
    if (P)                      //Set the parity bit according to P
    {
#if (PARITYBIT == ODD_PARITY)
        TB8 = 0;                //Set parity bit to 0
#elif (PARITYBIT == EVEN_PARITY)
        TB8 = 1;                //Set parity bit to 1
#endif
    }
    else
    {
#if (PARITYBIT == ODD_PARITY)
        TB8 = 1;                            //Set parity bit to 1
#elif (PARITYBIT == EVEN_PARITY)
        TB8 = 0;                            //Set parity bit to 0
#endif
    }
    busy = 1;
    SBUF = ACC;                             //Send data to UART buffer
}

/*----------------------------
Send a string to UART
Input: s (address of string)
Output:None
----------------------------*/
void SendString(char *s)
{
    while (*s)                              //Check the end of the string
    {
        SendData(*s++);                     //Send current char and increment string ptr
    }
}
```

2. 汇编程序:

```
/*---------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ----------------------------------------*/
/* --- 演示STC89xx系列单片机定时器2作波特率发生器测试程序---*/
/* --- Mobile: (86)13922805190 -------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -----------*/
/*---------------------------------------------------------------------------------*/

T2CON           EQU     0C8H            ;timer2 control register
TR2             BIT     T2CON.2

T2MOD           EQU     0C9H            ;timer2 mode register
RCAP2L          EQU     0CAH
RCAP2H          EQU     0CBH
TL2             EQU     0CCH
TH2             EQU     0CDH

;/*Define UART parity mode*/
#define NONE_PARITY     0               //None parity
#define ODD_PARITY      1               //Odd parity
#define EVEN_PARITY     2               //Even parity
#define MARK_PARITY     3               //Mark parity
#define SPACE_PARITY    4               //Space parity

#define PARITYBIT EVEN_PARITY   //Testing even parity
;-----------------------------------------
BUSY    BIT     20H.0                   ;transmit busy flag
;-----------------------------------------
        ORG     0000H
        LJMP    MAIN

        ORG     0023H
        LJMP    UART_ISR
;-----------------------------------------
        ORG     0100H
MAIN:
        CLR     BUSY
        CLR     EA
        MOV     SP,     #3FH
```

```
#if (PARITYBIT == NONE_PARITY)
        MOV   SCON,     #50H            ;8-bit variable UART
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
        MOV   SCON,     #0DAH           ;9-bit variable UART, parity bit initial to 1
#elif (PARITYBIT == SPACE_PARITY)
        MOV   SCON,     #0D2H           ;9-bit variable UART, parity bit initial to 0
#endif
;---------------------------------
        MOV   A,        #0FBH           ;65536-18432000/32/115200 = 0xfffb
        MOV   TL2,      A
        MOV   RCAP2L,   A
        MOV   A,        #0FFH
        MOV   TH2,      A               ;Set auto-reload vaule
        MOV   RCAP2H,   A
        MOV   T2CON     ,#34H           ;Timer2 start run
        SETB  ES                        ;Enable UART interrupt
        SETB  EA                        ;Open master interrupt switch
;---------------------------------
        MOV   DPTR,     #TESTSTR        ;Load string address to DPTR
        LCALL SENDSTRING                ;Send string
;---------------------------------
        SJMP    $
;-----------------------------------------
TESTSTR:                                ;Test string
   DB  "STC89-90xx Uart Test !",0DH,0AH,0

;/*---------------------------
;UART2 interrupt service routine
;---------------------------*/
UART_ISR:
        PUSH    ACC
        PUSH    PSW
        JNB     RI,CHECKTI              ;Check RI bit
        CLR     RI                      ;Clear RI bit
        MOV     P0,SBUF                 ;P0 show UART data
        MOV     C,RB8
        MOV     P2.2,C                  ;P2.2 show parity bit
CHECKTI:
        JNB     TI,ISR_EXIT             ;Check S2TI bit
        CLR     TI                      ;Clear S2TI bit
        CLR     BUSY                    ;Clear transmit busy flag
ISR_EXIT:
        POP     PSW
        POP     ACC
        RETI
```

```
;/*----------------------------
;Send a byte data to UART
;Input: ACC (data to be sent)
;Output:None
;----------------------------*/
SENDDATA:
        JB      BUSY,$                  ;Wait for the completion of the previous data is sent
        MOV     ACC,A                   ;Calculate the even parity bit P (PSW.0)
        JNB     P,      EVEN1INACC      ;Set the parity bit according to P
ODD1INACC:
#if (PARITYBIT == ODD_PARITY)
        CLR     TB8                     ;Set parity bit to 0
#elif (PARITYBIT == EVEN_PARITY)
        SETB    TB8                     ;Set parity bit to 1
#endif
        SJMP    PARITYBITOK
EVEN1INACC:
#if (PARITYBIT == ODD_PARITY)
        SETB    TB8                     ;Set parity bit to 1
#elif (PARITYBIT == EVEN_PARITY)
        CLR     TB8                     ;Set parity bit to 0
#endif
PARITYBITOK:            ;Parity bit set completed
        SETB    BUSY
        MOV     SBUF,   A               ;Send data to UART buffer
        RET

;/*----------------------------
;Send a string to UART
;Input: DPTR (address of string)
;Output:None
;----------------------------*/
SENDSTRING:
        CLR     A
        MOVC    A,      @A+DPTR         ;Get current char
        JZ      STRINGEND               ;Check the end of the string
        INC     DPTR                    ;increment string ptr
        LCALL   SENDDATA                ;Send current char
        SJMP    SENDSTRING              ;Check next
STRINGEND:
        RET
;----------------------------------------
        END
```

7.2.4 定时器2作可编程时钟输出及其测试程序(C程序及汇编程序)

STC89C51RC/RD+ 系列单片机，可设定定时/计数器2，通过P1.0输出时钟。P1.0除作通用I/O口外还有两个功能可供选用：用于定时/ 计数器2 的外部计数输入和定时/计数器2 时钟信号输出。图5 为时钟输出和外部事件计数方式示意图。

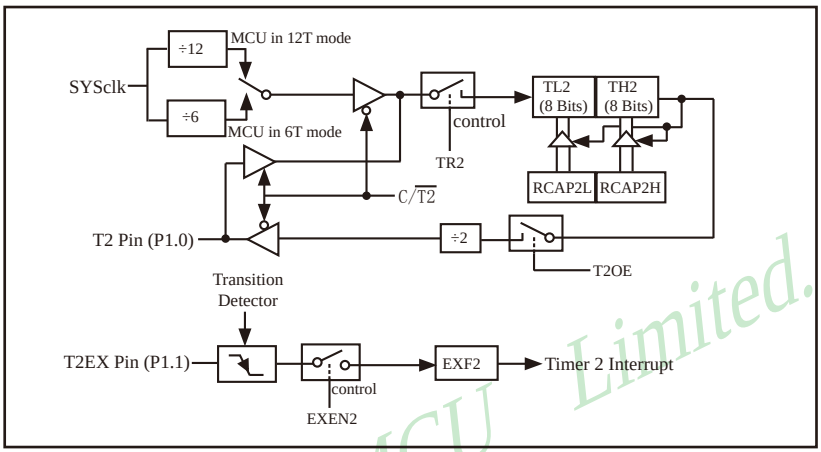


图5 定时器2的可编程时钟输出模式

通过软件对T2CON.1位C/T2复位为0，对T2MOD.1位T2OE置1就可将定时/计数器2选定为时钟信号发生器，而T2CON.2位TR2控制时钟信号输出开始或结束（TR2为启/停控制位）. 由主振频率（SYSclk）和定时/计数器2定时、自动再装入方式的计数初值决定时钟信号的输出频率。其设置公式如下：

$$\text{模式1和模式3的波特率} = \frac{\text{SYSclk}}{n\times[65536-(\text{RCAP2H,RCAP2L})]}$$

* n=2，6时钟/机器周期；n=4，12时钟/机器周期

从公式可见，在主振频率（SYSclk）设定后，时钟信号输出频率就取决于定时计数初值的设定。

在时钟输出模式下，计数器回0 溢出不会产生中断请求。这种功能相当于定时/计数器2用作波特率发生器，同时又可以作时钟发生器。但必须注意，无论如何波特率发生器和时钟发生器不能单独确定各自不同的频率。原因是两者都用同一个陷阱寄存器RCAP2H、RCAP2L，不可能出现两个计数初值。

定时器2作可编程时钟输出演示程序

1、C程序清单：

```
/*---------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ----------------------------------*/
/* --- STC89-90xx Series Programmable Clock Output Demo -------*/
/* --- Mobile: (86)13922805190 -------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------*/
/* If you want to use the program or the program referenced in the  */
/* article, please specify in which data and procedures from STC    */
/*---------------------------------------------------------------------------*/

#include "reg51.h"

typedef unsigned char BYTE;
typedef unsigned int WORD;
//-----------------------------------------------
/* define constants */
#define FOSC 18432000L

#define F38_4KHz    (65536-18432000/4/38400)

/* define SFR */

sfr   T2CON = 0xc8;             //timer2 control register
sbit TF2   = T2CON^7;
sbit TR2  = T2CON^2;

sfr T2MOD = 0xc9;               //timer2 mode register
sfr RCAP2L = 0xca;
sfr RCAP2H = 0xcb;
sfr TL2 = 0xcc;
sfr TH2 = 0xcd;

sbit T2 = P1^0;                 //Clock Output pin
//-----------------------------------------------
/* main program */
void main()
{
   T2MOD = 0x02;                        //enable timer2 output clock
   RCAP2L = TL2 = F38_4KHz;              //initial timer2 low byte
   RCAP2H = TH2 = F38_4KHz >> 8;        //initial timer2 high byte
   TR2 = 1;                             //timer2 start running
   EA = 1;                              //open global interrupt switch

   while (1);                           //loop
}
```

2、汇编程序清单：

```
/*---------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ----------------------------------*/
/* --- STC89-90xx Series Programmable Clock Output Demo -------*/
/* --- Mobile: (86)13922805190 -------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------*/
/* If you want to use the program or the program referenced in the  */
/* article, please specify in which data and procedures from STC    */
/*---------------------------------------------------------------------------*/
;/* define constants */
F38_4KHz    EQU 0FF88H        ;38.4KHz frequency calculation method of 12T mode (65536-18432000/4/38400)

;/* define SFR */

T2CON   EQU     0C8H                    ;timer2 control register
TF2     BIT     T2CON.7
TR2     BIT     T2CON.2

T2MOD           EQU     0C9H            ;timer2 mode register
RCAP2L          EQU     0CAH
RCAP2H          EQU     0CBH
TL2             EQU     0CCH
TH2             EQU     0CDH

T2      BIT  P1.0                       ;Clock Output pin
;-----------------------------------------------
        ORG     0000H
        LJMP    MAIN
;-----------------------------------------------
;/* main program */
MAIN:
        MOV     T2MOD,  #02H                    ;enable timer2 output clock
        MOV     T2CON,  #00H                    ;timer2 stop
        MOV     TL2,    #00H                    ;initial timer2 low byte
        MOV     TH2,    #00H                    ;initial timer2 high byte
        MOV     RCAP2L, #LOW F38_4KHz           ;initial timer2 reload low byte
        MOV     RCAP2H, #HIGH F38_4KHz          ;initial timer2 reload high byte
        SETB    TR2                             ;timer2 start running
        SJMP    $
;-----------------------------------------------
        END
```

7.2.5 定时器/计数器2作定时器的测试程序(C程序及汇编程序)

1、C程序清单:

```
/*---------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ----------------------------------*/
/* --- STC89-90xx Series 16-bit Timer Demo -------------------------*/
/* --- Mobile: (86)13922805190 -------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------*/
/* If you want to use the program or the program referenced in the  */
/* article, please specify in which data and procedures from STC    */
/*---------------------------------------------------------------------------*/

#include "reg51.h"

typedef unsigned char BYTE;
typedef unsigned int WORD;
//-----------------------------------------------

/* define constants */
#define FOSC 18432000L

#define T1MS (65536-FOSC/12/1000)          //1ms timer calculation method in 12T mode

/* define SFR */
sbit ET2 = IE^5;

sfr T2CON = 0xc8;                          //timer2 control register
sbit TF2  = T2CON^7;
sbit TR2  = T2CON^2;

sfr T2MOD = 0xc9;                          //timer2 mode register
sfr RCAP2L = 0xca;
sfr RCAP2H = 0xcb;
sfr TL2 = 0xcc;
sfr TH2 = 0xcd;

sbit TEST_LED = P1^0;                      //work LED, flash once per second

/* define variables */
WORD count;                                //1000 times counter

//-----------------------------------------------
```

```
/* Timer2 interrupt routine */
void tm2_isr() interrupt 5 using 1
{
   TF2 = 0;
   if (count-- == 0)                    //1ms * 1000 -> 1s
   {
      count = 1000;                     //reset counter
      TEST_LED = ! TEST_LED;            //work LED flash
   }
}
//-----------------------------------------------
/* main program */
void main()
{
   RCAP2L = TL2 = T1MS;                 //initial timer2 low byte
   RCAP2H = TH2 = T1MS >> 8;            //initial timer2 high byte
   TR2 = 1;                             //timer2 start running
   ET2 = 1;                             //enable timer2 interrupt
   EA = 1;                              //open global interrupt switch
   count = 0;                           //initial counter

   while (1);                           //loop
}
```


2、汇编程序清单：

```
/*---------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ----------------------------------*/
/* --- STC89-90xx Series 16-bit Timer Demo --------------------------*/
/* --- Mobile: (86)13922805190 -------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------*/
/* If you want to use the program or the program referenced in the  */
/* article, please specify in which data and procedures from STC    */
/*---------------------------------------------------------------------------*/

;/* define constants */
T1MS       EQU     0FA00H              ;1ms(1000Hz) timer (65536-18432000/12/1000)

;/* define SFR */

ET2        BIT        IE.5
```

```
T2CON   EQU     0C8H            ;timer2 control register
TF2     BIT     T2CON.7
TR2     BIT     T2CON.2

T2MOD   EQU     0C9H            ;timer2 mode register
RCAP2L  EQU     0CAH
RCAP2H  EQU     0CBH
TL2     EQU     0CCH
TH2     EQU     0CDH

TEST_LED    BIT  P1.0           ;work LED, flash once per second

;/* define variables */
COUNT DATA 30H                  ;1000 times counter (2 bytes)

;-----------------------------------------------

   ORG    0000H
   LJMP   MAIN
   ORG    002BH
   LJMP   TM2_ISR

;-----------------------------------------------

;/* main program */
MAIN:
        MOV    T2MOD,#00H                   ;initial timer2 mode
        MOV    T2CON,#00H                   ;timer2 stop
        MOV    TL2,#00H                     ;initial timer2 low byte
        MOV    TH2,#00H                     ;initial timer2 high byte
        MOV    RCAP2L,#LOW T1MS             ;initial timer2 reload low byte
        MOV    RCAP2H,#HIGH T1MS            ;initial timer2 reload high byte
        SETB   TR2                          ;timer2 start running
        SETB   ET2                          ;enable timer2 interrupt
        SETB   EA                           ;open global interrupt switch
        CLR    A
        MOV    COUNT,A
        MOV    COUNT+1,A                    ;initial counter
        SJMP   $

;-----------------------------------------------
```

```
;/* Timer2 interrupt routine */
TM2_ISR:
        PUSH    ACC
        PUSH    PSW
        CLR     TF2
        MOV     A,      COUNT
        ORL     A,      COUNT+1             ;check whether count(2byte) is equal to 0
        JNZ     SKIP
        MOV     COUNT,  #LOW 1000           ;1ms * 1000 -> 1s
        MOV     COUNT+1,        #HIGH 1000
        CPL     TEST_LED                    ;work LED flash
SKIP:
        CLR     C
        MOV     A,      COUNT               ;count--
        SUBB    A,      #1
        MOV     COUNT,  A
        MOV     A,      COUNT+1
        SUBB    A,      #0
        MOV     COUNT+1,A
        POP     PSW
        POP     ACC
        RETI
;-----------------------------------------

        END
```

第8章 串行口通信

STC89C51RC/RD+系列单片机内部集成有一个功能很强的全双工串行通信口，与传统8051单片机的串口完全兼容。设有2**个互相独立的接收、发送缓冲器，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（99H）**。两个缓冲器统称串行通信特殊功能寄存器SBUF。**

串行通信设有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。波特率由内部定时器/计数器产生，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。

STC89C51RC/RD+系列单片机串行口对应的硬件部分对应的管脚是P3.0/RxD和P3.1/TxD。

STC89C51RC/RD+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。**

8.1 串行口相关寄存器

符号	描述	地址	位地址及符号 MSB LSB								复位值
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	-	POF	GF1	GF0	PD	IDL	00x1 0000B
IE	Interrupt Enable	A8H	EA	-	ET2	ES	ET1	EX1	ET0	EX0	0x00 0000B
IPH	中断优先级寄存器高	B7H	-	-	PT2H	PSH	PT1H	PX1H	PT0H	PX0H	xx00 0000B
IP	中断优先级寄存器低	B8H	-	-	PT2	PS	PT1	PX1	PT0	PX0	xx00 0000B
SADEN	Slave Address Mask	B9H									0000 0000B
SADDR	Slave Address	A9H									0000 0000B

1. 串行口控制寄存器SCON和PCON

STC89C51RC/RD+系列单片机的串行口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	波特率是SYSclk/12,
0	1	方式1	8位UART，波特率可变	(2^{SMOD}/32)×(定时器1的溢出率)
1	0	方式2	9位UART	(2^{SMOD} / 64) x SYSclk系统工作时钟频率
1	1	方式3	9位UART，波特率可变	(2^{SMOD}/32)x(定时器1的溢出率)
当单片机工作在12T模式时，**定时器1的溢出率** = SYSclk/12/(256 - T H1)； 当单片机工作在6T模式时，**定时器1的溢出率** = SYSclk / 6 / (256 - T H1)				

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SM2位为1，REN位为1，则从机处于只有接收到RB8位为1（地址帧）时才激活中断请求标志位RI为1，并向主机请求中断处理。被确认为寻址的从机则复位SM2位为0，从而才接收RB8为0的数据帧。
在方式1时，如果SM2位为1，则只有在接收到有效的停止位时才置位中断请求标志位RI为1；在方式0时，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。

RB8：在方式2或方式3，是接收到的第9位数据。在方式1，若SM2=0，则RB8是接收到的停止位。方式0不用RB8。

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，响应中断后必须用软件复位，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，必须用软件复位。

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，响应中断后必须用软件复位，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（例外情况见SM2说明），必须由软件复位，即RI=0。

SCON的所有位可通过整机复位信号复位为全“0”。SCON的字节地址尾98H，可位寻址，各位地址为98H~~9FH，可用软件实现位设置。当用指令改变SCON的有关内容时，其改变的状态将在下一条指令的第一个机器周期的S1P1状态发生作用。如果一次串行发送已经开始，则输出TB8将是原先的值，不是新改变的值。

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	-	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

2. 串行口数据缓冲寄存器SBUF

STC89xx系列单片机的串行口缓冲寄存器(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，STC89C51RC/RD+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)。

4. 与串行口中断相关的寄存器IE和IPH、IP

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	-	ET2	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。

串行口中断优先级控制位PS/PSH位于中断优先级控制寄存器IP/IPH中，中断优先级控制寄存器的格式如下：

IPH: 中断优先级控制寄存器高(不可位寻址)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IPH	B7H	name	PX3H	PX2H	PT2H	PSH	PT1H	PX1H	PT0H	PX0H

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	-	-	PT2	PS	PT1	PX1	PT0	PX0

PSH, PS: 串口1中断优先级控制位。
当PSH=0且PS=0时，串口1中断为最低优先级中断(优先级0)
当PSH=0且PS=1时，串口1中断为较低优先级中断(优先级1)
当PSH=1且PS=0时，串口1中断为较高优先级中断(优先级2)
当PSH=1且PS=1时，串口1中断为最高优先级中断(优先级3)

8.2 串行口工作模式

STC89C51RC/RD+系列单片机的串行通信有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。

8.2.1 串行口工作模式0：同步移位寄存器

在模式0状态，串行通信工作在同步移位寄存器模式，当当单片机工作在6T模式时，其波特率固定为SYSclk/6。当单片机工作在12T时，其波特率固定为SYSclk/12。串行口数据由RxD(RxD/P3.0)端输入，同步移位脉冲（SHIFTCLOCK）由TxD(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/6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/6。其时序图如图8-1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/6，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的示意图如图8-1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。

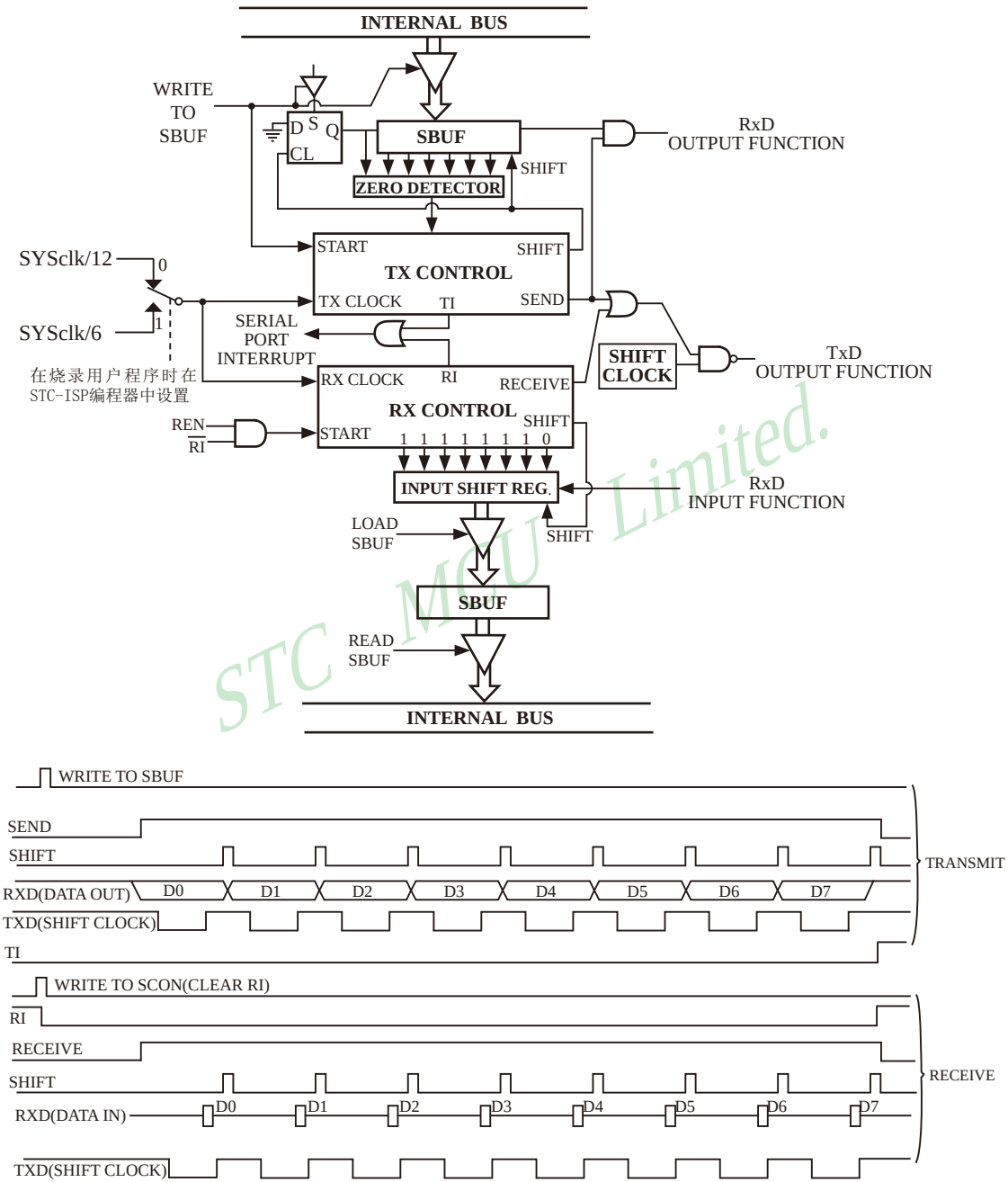


图8-1　串行口模式0功能结构及时序示意图

8.2.2 串行口工作模式1：8位UART，波特率可变

当软件设置SCON的SM0、SM1为“01”时，串行通信则以模式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(TxD/P3.1)为发送信息，RxD(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。

图8-2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″。

串行通信模式1的波特率是可变的，可变的波特由定时器/计数器1或独立波特率发生器产生。

串行通信模式1的波特率=2^{SMOD}/32×(定时器/计数器1溢出率)

当单片机工作在12T模式时，定时器1的溢出率 = SYSclk/12/(256 - TH1);

当单片机工作在6T模式时， 定时器1的溢出率 = SYSclk /6/ (256 - TH1)

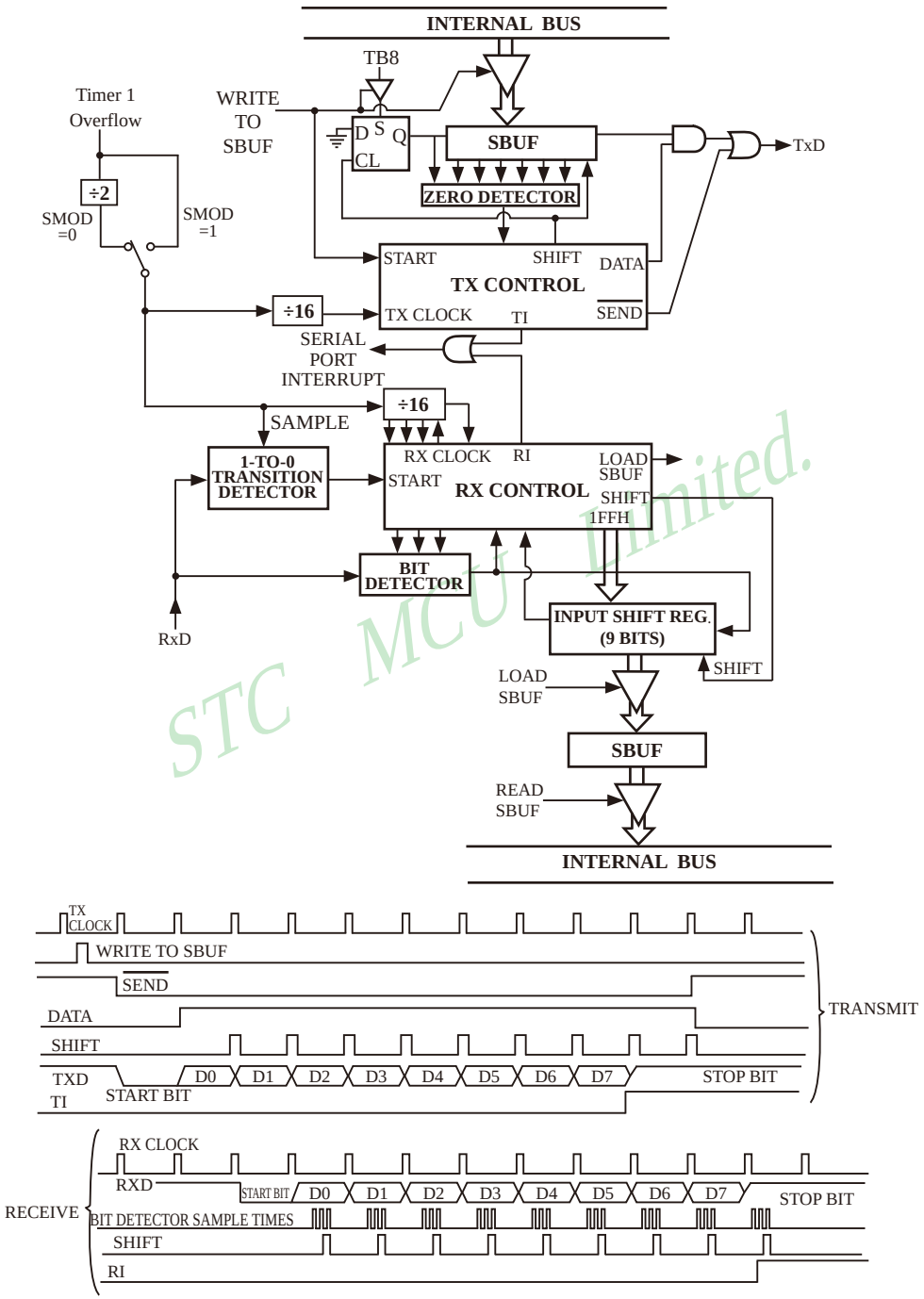


图8-2　串行口模式1功能结构示意图及接收/发送时序图

8.2.3 串行口工作模式2：9位UART，波特率固定

当SM0、SM1两位为10时，串行口工作在模式2。串行口工作模式2为9位数据异步通信UART模式，其一帧的信息由11位组成：1位起始位，8位数据位(低位在先)，1位可编程位(第9位数据)和1位停止位。发送时可编程位(第9位数据)由SCON中的TB8提供，可软件设置为1或0，或者可将PSW中的奇/偶校验位P值装入TB8(TB8既可作为多机通信中的地址数据标志位，又可作为数据的奇偶校验位)。接收时第9位数据装入SCON的RB8。TxD为发送端口，RxD为接收端口，以全双工模式进行接收/发送。

模式2的波特率为：

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1/32(SYSclk)；当SMOD=0时，选择1/64(SYSclk)，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。

图8-3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。

由图8-3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。

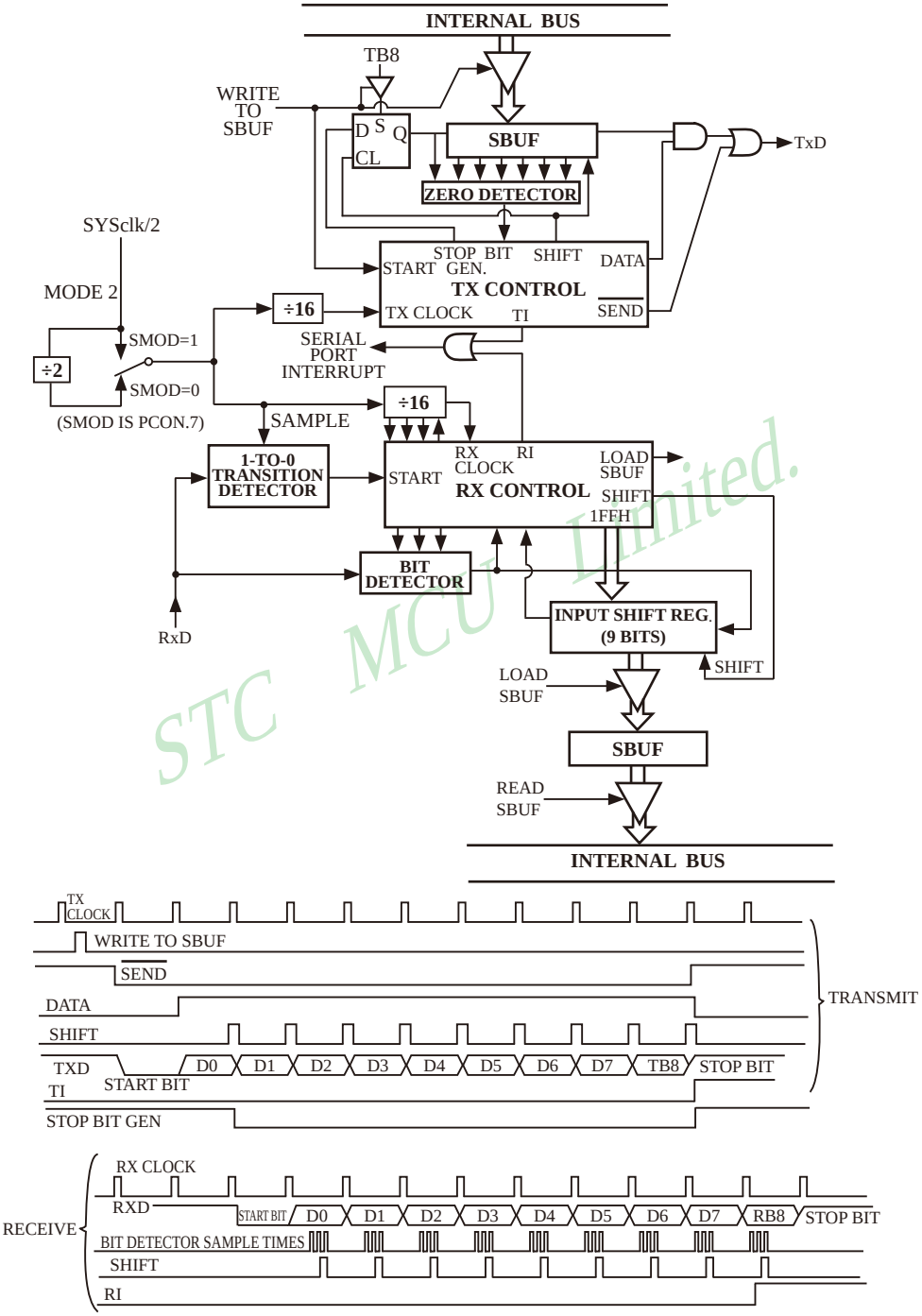


图8-3　串行口模式2功能结构示意图及接收/发送时序图

8.2.4 串行口工作模式3：9位UART，波特率可变

当SM0、SM1两位为11时，串行口工作在模式3。**串行通信模式3**为9位数据异步通信UART模式，**其一帧的信息由**11位组成：1位起始位，8位数据位(低位在先)，1位可编程位(第9位数据)和1位停止位。发送时可编程位(第9位数据)由SCON中的TB8提供，可软件设置为1或0，或者可将PSW中的奇/偶校验位P值装入TB8(TB8既可作为多机通信中的地址数据标志位，又可作为数据的奇偶校验位)。接收时第9位数据装入SCON的RB8。TxD为发送端口，RxD为接收端口，以全双工模式进行接收/发送。

模式3的波特率为：

串行通信模式3波特率=$2^{SMOD}/32\times$(定时器/计数器1的溢出率)

当**单片机工作在12T模式时，定时器1的溢出率** = SYSclk/12/(256 - TH1)；

当**单片机工作在6T模式时， 定时器1的溢出率** = SYSclk /6/ (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或独立波特率发生器的设置进行波特率的选择，是可变的。

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。

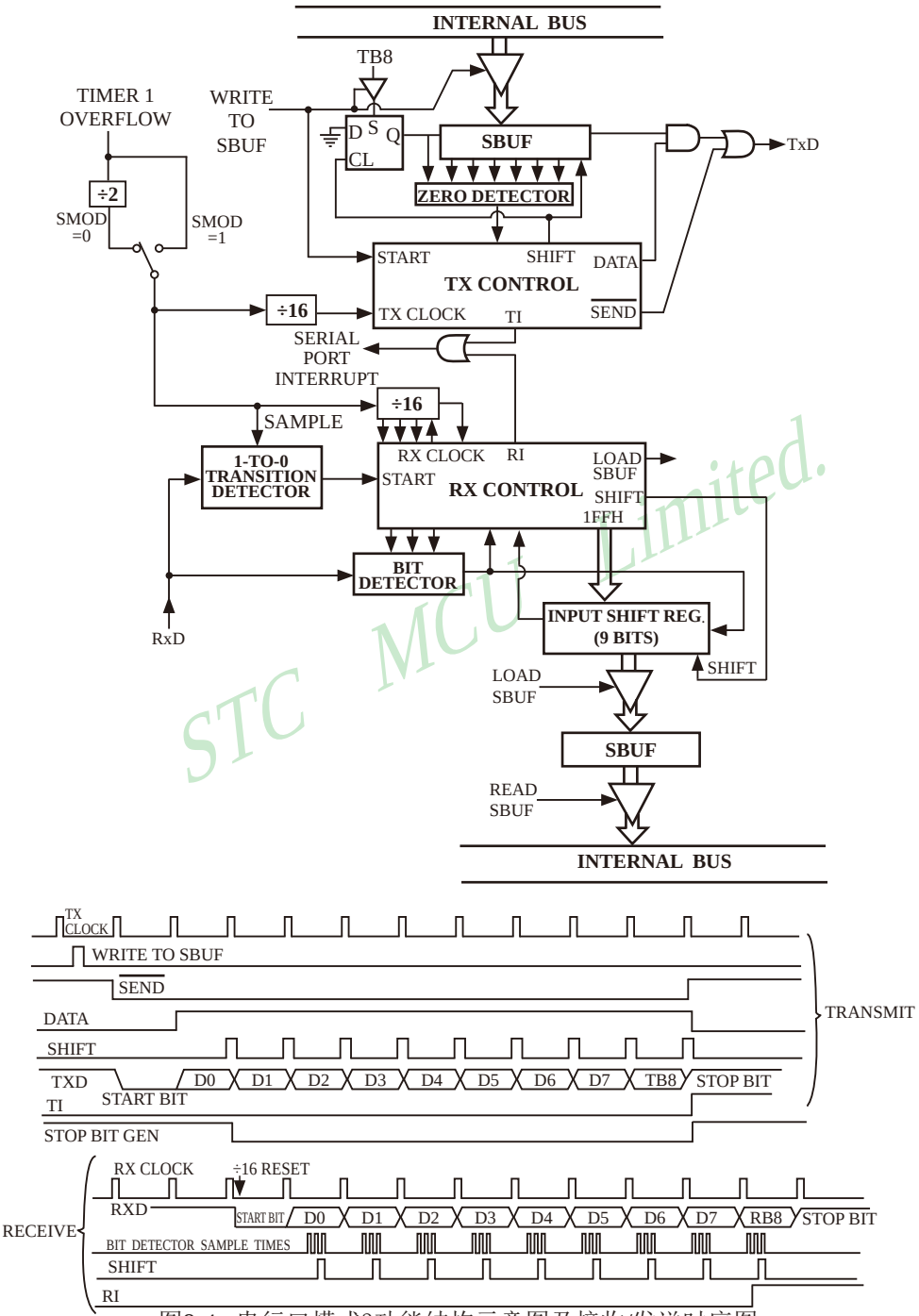


图8-4　串行口模式3功能结构示意图及接收/发送时序图

8.3 串行通信中波特率的设置

STC89C51RC/RD+系列单片机串行通信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BRT独立波特率发生器设置有关。通过对定时器/计数器1或BRT独立波特率发生器的设置，可选择不同的波特率，所以这种波特率是可变的。

串行通信模式0，其波特率与系统时钟频率SYSclk有关 。

当用户在烧录用户程序时在STC-ISP编程器中设置单片机为6T/双倍速时，其波特率 = SYSclk/12。

当用户在烧录用户程序时在STC-ISP编程器中设置单片机为12T/单倍速时，其波特率 = SYSclk/2。

一旦SYSclk选定且单片机在烧录用户程序时在STC-ISP编程器设置好，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=$2^{SMOD}/64\times$(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：

模式1、3波特率=$2^{SMOD}/32\times$(**定时器**/计数器1的溢出率或**BRT独立波特率发生器的溢出率**)

当**单片机工作在12T模式时，定时器1的溢出率** = SYSclk/12/(256 - TH1)；

当**单片机工作在6T模式时， 定时器1的溢出率** = SYSclk /6/ (256 - TH1)

通过对定时器/计数器1和BRT独立波特率发生器的设置，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/计数器1和BRT独立波特率发生器的溢出率的计算。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。

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。

STC89C51RC/RD+系列单片机设有两个定时器/计数器，因定时器/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2（8位自动重装）作为波特率的溢出率。

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分频或不分频的脉冲。当单片机工作在12T模式，TL1的计数输入来自于SYSclk经12分频的脉冲；当单片机工作在6T模式，TL1的计数输入来自于SYSclk经6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。对于一般情况下，

当单片机工作在12T模式时时，定时器/计数器1溢出一次所需的时间为：

$$(2^8-N)\times 12\text{时钟}=(2^8-N)\times 12\times\frac{1}{\text{SYSclk}}$$

当单片机工作在6T模式时，定时器/计数器1溢出一次所需的时间为：

$$(2^8-N)\times 6\text{时钟}=(2^8-N)\times 6\times\frac{1}{\text{SYSclk}}$$

于是得定时器/计数器每秒溢出的次数，即

当单片机工作在12T模式时，定时器/计数器1的溢出率=SYSclk/12×(2^8－N) (次/秒)

当单片机工作在6T模式时， 定时器/计数器1的溢出率=SYSclk×6×(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0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。

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		系统时钟频率(MHz)	SMOD	定时器1		
				C/$\overline{T}$	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，   #××H        ；  设置定时常数N
MOV   TL1，   #××H        ；
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。

8.4 串行口的测试程序(C程序及汇编程序)

1. C程序:

```
/*---------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ----------------------------------*/
/* --- 演示STC89xx系列单片机串行口功能 (8-bit/9-bit) ----------------*/
/* --- Mobile: (86)13922805190 -------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 ----------*/
/*---------------------------------------------------------------------------*/
#include "reg51.h"
#include "intrins.h"

typedef unsigned char BYTE;
typedef unsigned int WORD;

#define FOSC 18432000L                  //System frequency
#define BAUD 9600                       //UART baudrate

/*Define UART parity mode*/
#define NONE_PARITY     0               //None parity
#define ODD_PARITY      1               //Odd parity
#define EVEN_PARITY     2               //Even parity
#define MARK_PARITY     3               //Mark parity
#define SPACE_PARITY    4               //Space parity

#define PARITYBIT EVEN_PARITY           //Testing even parity

sbit bit9 = P2^2;                       //P2.2 show UART data bit9
bit busy;

void SendData(BYTE dat);
void SendString(char *s);

void main()
{
#if (PARITYBIT == NONE_PARITY)
    SCON = 0x50;                        //8-bit variable UART
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
    SCON = 0xda;                        //9-bit variable UART, parity bit initial to 1
#elif (PARITYBIT == SPACE_PARITY)
    SCON = 0xd2;                        //9-bit variable UART, parity bit initial to 0
#endif
```

```
	TMOD = 0x20;                            //Set Timer1 as 8-bit auto reload mode
	TH1 = TL1 = -(FOSC/12/32/BAUD);         //Set auto-reload vaule
	TR1 = 1;                                //Timer1 start run
	ES = 1;                                 //Enable UART interrupt
	EA = 1;                                 //Open master interrupt switch

	SendString("STC89-90xx\r\nUart Test !\r\n");
	while(1);
}

/*----------------------------
UART interrupt service routine
----------------------------*/
void Uart_Isr() interrupt 4 using 1
{
	if (RI)
	{
		RI = 0;                         //Clear receive interrupt flag
		P0 = SBUF;                      //P0 show UART data
		bit9 = RB8;                     //P2.2 show parity bit
	}
	if (TI)
	{
		TI = 0;                         //Clear transmit interrupt flag
		busy = 0;                       //Clear transmit busy flag
	}
}

/*----------------------------
Send a byte data to UART
Input: dat (data to be sent)
Output:None
----------------------------*/
void SendData(BYTE dat)
{
	while (busy);                           //Wait for the completion of the previous data is sent
	ACC = dat;                              //Calculate the even parity bit P (PSW.0)
	if (P)                                  //Set the parity bit according to P
	{
	#if (PARITYBIT == ODD_PARITY)
		TB8 = 0;                        //Set parity bit to 0
	#elif (PARITYBIT == EVEN_PARITY)
		TB8 = 1;                        //Set parity bit to 1
	#endif
	}
```

```
        else
        {
        #if (PARITYBIT == ODD_PARITY)
                TB8 = 1;                    //Set parity bit to 1
        #elif (PARITYBIT == EVEN_PARITY)
                TB8 = 0;                    //Set parity bit to 0
        #endif
        }
        busy = 1;
        SBUF = ACC;                         //Send data to UART buffer
}

/*----------------------------
Send a string to UART
Input: s (address of string)
Output:None
----------------------------*/
void SendString(char *s)
{
        while (*s)                          //Check the end of the string
        {
                SendData(*s++);             //Send current char and increment string ptr
        }
}
```

2. 汇编程序:

```
/*---------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ----------------------------------*/
/* --- 演示STC89xx系列单片机串行口功能 (8-bit/9-bit) ----------------*/
/* --- Mobile: (86)13922805190 -------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -----------*/
/*---------------------------------------------------------------------------*/

;/*Define UART parity mode*/
#define NONE_PARITY     0               //None parity
#define ODD_PARITY      1               //Odd parity
#define EVEN_PARITY     2               //Even parity
#define MARK_PARITY     3               //Mark parity
#define SPACE_PARITY    4               //Space parity

#define PARITYBIT EVEN_PARITY           //Testing even parity
;-----------------------------------------
        BUSY    BIT     20H.0           ;transmit busy flag
;-----------------------------------------
        ORG     0000H
        LJMP    MAIN

        ORG     0023H
        LJMP    UART_ISR
;-----------------------------------------
        ORG     0100H
MAIN:
        CLR     BUSY
        CLR     EA
        MOV     SP,     #3FH
#if (PARITYBIT == NONE_PARITY)
        MOV     SCON,   #50H            ;8-bit variable UART
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
        MOV     SCON,   #0DAH           ;9-bit variable UART, parity bit initial to 1
#elif (PARITYBIT == SPACE_PARITY)
        MOV     SCON,   #0D2H           ;9-bit variable UART, parity bit initial to 0
#endif
;-------------------------------
```

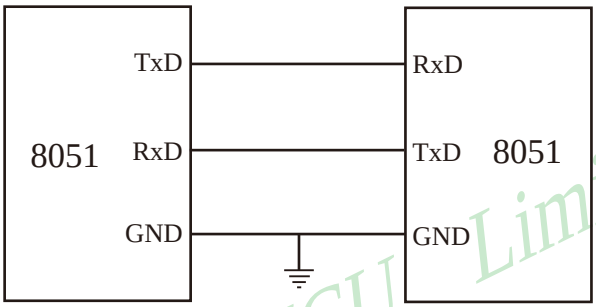
```
        MOV     TMOD,   #20H                    ;Set Timer1 as 8-bit auto reload mode
        MOV     A,      #0FBH                   ;256-18432000/12/32/9600
        MOV     TH1,    A                       ;Set auto-reload vaule
        MOV     TL1,    A
        SETB    TR1                             ;Timer1 start run
        SETB    ES                              ;Enable UART interrupt
        SETB    EA                              ;Open master interrupt switch
;------------------------------
        MOV     DPTR,   #TESTSTR                ;Load string address to DPTR
        LCALL   SENDSTRING                      ;Send string
;------------------------------
        SJMP    $
;----------------------------------------
TESTSTR:                                        ;Test string
        DB  "STC89-90xx Uart Test !",0DH,0AH,0

;/*---------------------------
;UART2 interrupt service routine
;---------------------------*/
UART_ISR:
        PUSH    ACC
        PUSH    PSW
        JNB     RI,     CHECKTI                 ;Check RI bit
        CLR     RI                              ;Clear RI bit
        MOV     P0,     SBUF                    ;P0 show UART data
        MOV     C,      RB8
        MOV     P2.2,   C                       ;P2.2 show parity bit
CHECKTI:
        JNB     TI,     ISR_EXIT                ;Check S2TI bit
        CLR     TI                              ;Clear S2TI bit
        CLR     BUSY                            ;Clear transmit busy flag
ISR_EXIT:
        POP     PSW
        POP     ACC
        RETI

;/*---------------------------
;Send a byte data to UART
;Input: ACC (data to be sent)
;Output:None
;---------------------------*/
```

```
SENDDATA:
        JB      BUSY,   $               ;Wait for the completion of the previous data is sent
        MOV     ACC,    A               ;Calculate the even parity bit P (PSW.0)
        JNB     P,      EVEN1INACC      ;Set the parity bit according to P
ODD1INACC:
#if (PARITYBIT == ODD_PARITY)
        CLR     TB8                     ;Set parity bit to 0
#elif (PARITYBIT == EVEN_PARITY)
        SETB    TB8                     ;Set parity bit to 1
#endif
        SJMP    PARITYBITOK
EVEN1INACC:
#if (PARITYBIT == ODD_PARITY)
        SETB    TB8                     ;Set parity bit to 1
#elif (PARITYBIT == EVEN_PARITY)
        CLR     TB8                     ;Set parity bit to 0
#endif
PARITYBITOK:                            ;Parity bit set completed
        SETB    BUSY
        MOV     SBUF,   A               ;Send data to UART buffer
        RET

;/*----------------------------
;Send a string to UART
;Input: DPTR (address of string)
;Output:None
;----------------------------*/
SENDSTRING:
        CLR     A
        MOVC    A,      @A+DPTR         ;Get current char
        JZ      STRINGEND               ;Check the end of the string
        INC     DPTR                    ;increment string ptr
        LCALL   SENDDATA                ;Send current char
        SJMP    SENDSTRING              ;Check next
STRINGEND:
        RET
;-----------------------------------------
        END
```

8.5 双机通信

STC89C51RC/RD+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—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。

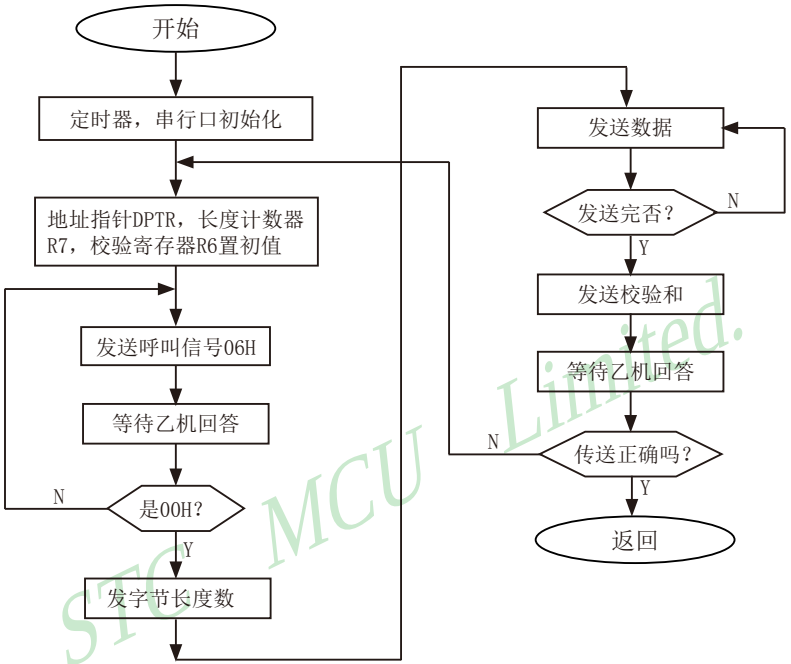
不同的通信要求，软件“协议”内容也不一样，有关需甲、乙双方共同遵守的约定应尽量完善，以防止通信不能正确判别而失败。

STC89C51RC/RD+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            ；
        MOV     SBUF，  A               ；}发送呼叫信号“06H”
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              ；}
        MOV     SBUF,   A               ；}发送数据块数n
        ADD     A,      R6
        MOV     R6,     A
WAIT2：
        JBC     TI,     TX-NES          ；}
        JMP     WAIT2                   ；}等待发送完
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                  ；
        SJMP    WAIT4                   ；等待发送完
RX-0FH：
        JBC     RI,     IF-0FH          ；
        SJMP    RX-0FH                  ；等待接收乙机回答信号
IF-0FH：
        MOV     A,      SBUF；          ；
        CJNE    A,      #0FH,   ST-RAM  ；判断传输是否正确，否则重新发送
        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：

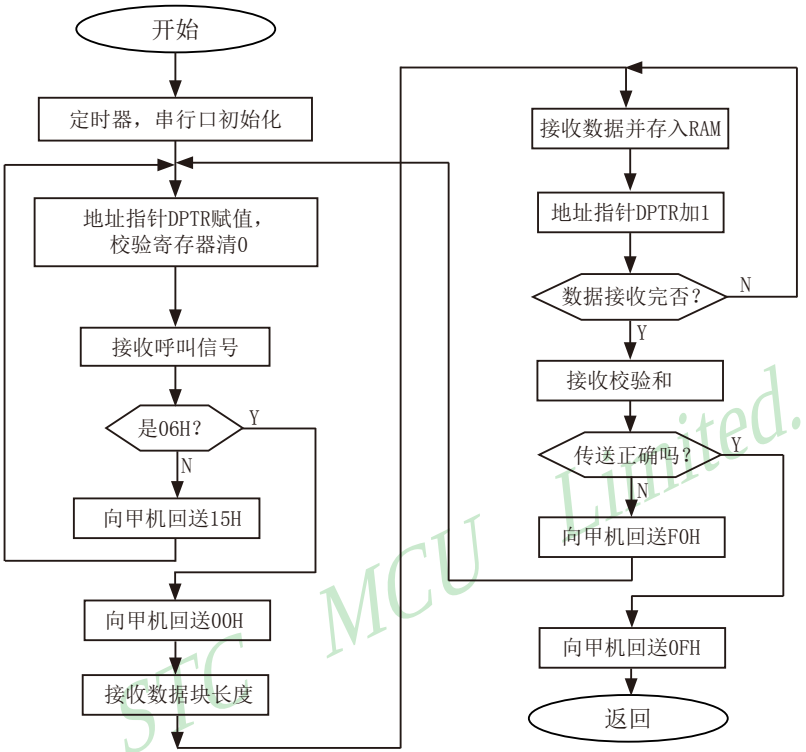
内部RAM 31H、30H单元存放接收数据缓冲区首地址。

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。



接收子程序清单：

```
TART：
        MOV     TMOD，  #20H
        MOV     TH1，   #0F3H           ；
                                        ｝
        MOV     TL1，   #0F3H           ；｝定时器/计数器1设置
        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             ；
        MOV     DPL，   30H             ；｝设置DPTR首地址
        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            ；
        MOV     SBUF，  A               ；} 向甲机发送“00H”，同意接收
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           ；
        SJMP    RX-NES                  ；} 接收数据帧
HAVE3：
        MOV     A，     SBUF            ；
        MOVX    @DPTR，A                ；接收到的数据存入外部RAM
        INC     DPTR                    ；
        ADD     A，     R6              ；
        MOV     R6，    A               ；} 形成累加和
        DJNZ    R7，    RX-NES          ；判断数据是否接收完
```

```
RX-SUM：
        JBC     RI，    HAVE4           ；
        SJMP    RX-SUM                  ；  等待接收校验和
HAVE4：
        MOV     A，     SBUF            ；
        CJNE    A，     R6，    TX-ERR  ；  判断传输是否正确
TX-RIT：
        MOV     A，     #0FH            ；
        MOV     SBUF，  A               ；  向甲机发送接收正确信息
WAIT3：
        JBC     TI，    GOOD            ；
        SJMP    WAIT3                   ；  等待发送结束
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STC-89C51RC/RD+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，任以上述协议为例，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—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           ；
        MOV     TL1，   #0F3H           ；  设置波特率为2400位/秒
        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
                                        ；
        MOV     30H，#00H               ；  起始地址1000H
        MOV     33H，#00H               ；累加和单元清0
        SETB    EA                      ；
        SETB    ES                      ；  开中断
        •
        •
        •
```

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        ；}
        MOV     SBUF，  A           ；} 向甲机发送接收正确信号“0FH”
WAIT1：
        JNB     TI，    WAITI       ；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       ；}
        MOV     SBUF，  A           ；} 向甲机发送接收出错信号“F0H”
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             ；
        MOV     33H，   A               ；  形成累加和
        CLR     7EH                     ；清除数据块数标志
        SJMP    RETURN                  ；转恢复现场处理
RXDATA：
        MOV     DPH，   31H             ；
        MOV     DPL，   30H             ；  设置存储数据地址指针
        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             ；
        MOV     30H，   DPL             ；  保存地址指针值
        ADD     A，     33H             ；
        MOV     33H，   A               ；  形成累加和
        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
```

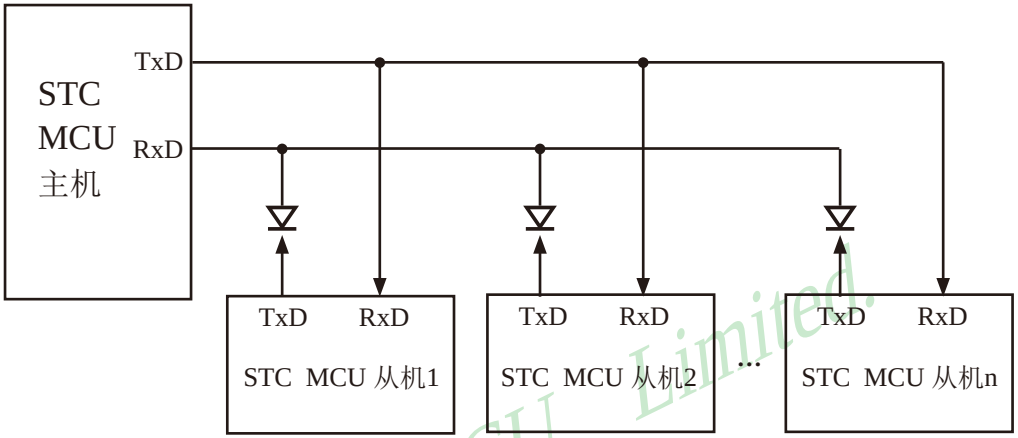
```
AGAIN：
        SETB    7FH             ；
        SETB    7EH             ；恢复标志位
        SETB    7DH             ；
        MOV     33H，   #00H    ；累加和单元清0
        MOV     31H，   #10H    ；}
        MOV     30H，   #00H    ；}恢复接收数据缓冲区首地址
RETURN：
        POP     A               ；
        POP     DPL             ；恢复现场
        POP     DPH             ；
        SETB    EA              ；开中断
        RET1                    ；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

8.6 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。STC89C51RC/RD+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知己转发。

（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）之间能可靠通信，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，任维持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器区1的R0~R7。

主机发送程序清单：

```
        ORG     0000H
        AJMP    MAIN                    ；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
          .
          .
          .
MAIN:   . . .   . . .                   ；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                     ；
        PUSH    A                       ；  现场入栈保护
        CLR     RS1                     ；
        SETB    RS0                     ；  选择工作寄存器区1
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                       ；
        POP     PSW                     ；  恢复现场
        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20·0H、20·1H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    20・0                   ；
        SETB    20・1                   ；  置标志位为1
        SETB    EA                      ；
        SETB    ES                      ；  开中断
        ：
        ORG     1000H
SERVE：
        CLR     RI                      ；清接收中断请求标志RI=0
        PUSH    A                       ；
        PUSH    PSW                     ；  现场保护
        CLR     RS1                     ；
        SETB    RS0                     ；  选择工作寄存器区1
        JB      20・0H，  ISADDR        ；判断是否是地址帧
        JB      20・1H，  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    20·0H                       ；
        SETB    20·1H                       ；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
                                            }
        CJNE    A，     #05H，  RETURN      ；  相符否，不符则转返回
        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，20·0，SM2
        CLR     20·0H                       ；清0TI，20·0，SM2
        CLR     SM2                         ；清0TI，20·0，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     20·1H                       ；清20·1H标志，表示以后接收的为数据
RETURN：
        POP     PSW                         ；
                                            }
        POP     A                           ；  恢复现场
        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。

第9章 STC89C51RC/RD+系列EEPROM的应用

STC89C51RC/RD+系列单片机内部集成了的EEPROM是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。

9.1 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 MSB LSB								复位值
ISP_DATA	ISP/IAP Flash Data Register	E2H									1111 1111B
ISP_ADDRH	ISP/IAP Flash Address High	E3H									0000 0000B
ISP_ADDRL	ISP/IAP Flash Address Low	E4H									0000 0000B
ISP_CMD	ISP/IAP Flash Command Register	E5H	-	-	-	-	-	-	MS1	MS0	xxxx xx00B
ISP_TRIG	ISP/IAP Flash Command Trigger	E6H									xxxx xxxxB
ISP_CONTR	ISP/IAP Control Register	E7H	ISPEN	SWBS	SWRST	-	-	WT2	WT1	WT0	000x x000B

1. ISP/IAP数据寄存器ISP_DATA

ISP_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ISP_ADDRH和ISP_ADDRL

ISP_ADDRH : ISP/IAP 操作时的地址寄存器高八位。**该寄存器地址为E3H, 复位后值为00H.**
ISP_ADDRL : ISP/IAP 操作时的地址寄存器低八位。**该寄存器地址为E4H, 复位后值为00H.**

3. ISP/IAP命令寄存器ISP_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
ISP_CMD	E5H	name	-	-	-	-	-	-	MS1	MS0

MS2	MS1	MS0	命令 / 操作 模式选择
0	0	0	Standby 待机模式，无ISP操作
0	0	1	从用户的应用程序区对"Data Flash/EEPROM区"进行字节读
0	1	0	从用户的应用程序区对"Data Flash/EEPROM区"进行字节编程
0	1	1	从用户的应用程序区对"Data Flash/EEPROM区"进行扇区擦除

程序在系统ISP程序区时可以对用户应用程序区/数据Flash区(EEPROM)进行字节读/字节编程/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。已经固化有ISP引导码,并设置为上电复位进入ISP

4. ISP/IA命令触发寄存器ISP_TRIG

ISP_TRIG: ISP/IAP 操作时的命令触发寄存器。
在ISPEN(ISP_CONTR.7) = 1 时, 对ISP_TRIG先写入46h, 再写入B9h, ISP/IAP 命令才会生效。

ISP/IAP操作完成后，ISP地址高八位寄存器ISP_ADDRH、ISP地址低八位寄存器ISP_ADDRL和ISP命令寄存器ISP_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ISP_ADDRH和ISP_ADDRL寄存器。

每次ISP操作时，都要对ISP_TRIG先写入46H，再写入B9H，ISP/IAP命令才会生效。

5. ISP/IAP命令寄存器ISP_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	E7H	name	ISPEN	SWBS	SWRST	-	-	WT2	WT2	WT0

ISPEN: ISP/IAP功能允许位。0：禁止IAP/ISP读/写/擦除Data Flash/EEPROM
1：允许IAP/ISP读/写/擦除Data Flash/EEPROM

SWBS：软件选择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：产生软件系统复位，硬件自动复位。

```
;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 ISP_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 ISP_CONTR, #00100000B ;SWBS = 0(选择AP 区), SWRST = 1(软复位)
```

设置等待时间			CPU等待时间(机器周期),(1个机器周期=12个CPU工作时钟)			
WT2	WT1	WT0	Read/读	Program/编程 (=72uS)	Sector Erase 扇区擦除 (=13.1304ms)	Recommended System Clock 跟等待参数对应的推荐系统时钟
0	1	1	6个机器周期	30个机器周期	5471个机器周期	5MHz
0	1	0	11个机器周期	60个机器周期	10942个机器周期	10MHz
0	0	1	22个机器周期	120个机器周期	21885个机器周期	20MHz
0	0	0	43个机器周期	240个机器周期	43769个机器周期	40MHz

9.2 STC89C51RC/RD+系列单片机EEPROM空间大小及地址

STC89C51RC/RD+系列单片机内部可用EEPROM的地址与程序空间是分开的：程序在用户应用程序区时,可以对EEPROM 进行IAP/ISP操作。

具体某个型号单片机内部EEPROM大小及详细地址请参阅:

1. STC89C51RC/RD+系列单片机内部EEPROM详细地址表
2. STC89C51RC/RD+系列单片机内部EEPROM空间大小选型一览表

STC89C51RC/RD+系列单片机内部EEPROM选型一览表				
型号	EEPROM字节数	扇区数	起始扇区首地址	结束扇区末尾地址
STC89C51RC STC89LE51RC	4K	8	2000h	2FFFh
STC89C52RC STC89LE52RC	4K	8	2000h	2FFFh
STC89C54RD+ STC89LE54RD+	45K	90	4000h	F3FFh
STC89C58RD+ STC89LE58RD+	29K	58	8000h	F3FFh
STC89C510RD+ STC89LE510RD+	21K	42	A000h	F3FFh
STC89C512RD+ STC89LE512RD+	13K	26	C000h	F3FFh
STC89C514RD+ STC89LE514RD+	5K	10	E000h	F3FFh

STC89C51RC系列单片机内部EEPROM详细地址表 具体某型号有多少扇区的EEPROM,参照前面的EEPROM空间大小选型一览表， 每个扇区0.5 K字节								
第一扇区		第二扇区		第三扇区		第四扇区		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
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	
2000h	21FFh	2200h	23FFh	2400h	25FFh	2600h	27FFH	
第五扇区		第六扇区		第七扇区		第八扇区		
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	
2800h	29FFh	2A00h	2BFFh	2C00h	2DFFh	2E00h	2FFFh	

STC89C58RD+系列单片机内部EEPROM详细地址表

具体某型号有多少扇区的EEPROM,参照前面的EEPROM空间大小选型一览表， 每个扇区0.5 K字节

第一扇区		第二扇区		第三扇区		第四扇区	
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址
8000h	81FFh	8200h	83FFh	8400h	85FFh	8600h	87FFh
第五扇区		第六扇区		第七扇区		第八扇区	
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址
8800h	89FFh	8A00h	8BFFh	8C00h	8DFFh	8E00h	8FFFh
第九扇区		第十扇区		第十一扇区		第十二扇区	
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址
9000h	91FFh	9200h	93FFh	9400h	95FFh	9600h	97FFh
第十三扇区		第十四扇区		第十五扇区		第十六扇区	
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址
9800h	99FFh	1A00h	9BFFh	9C00h	9DFFh	9E00h	9FFFh
第十七扇区		第十八扇区		第十九扇区		第二十扇区	
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址
A000h	A1FFh	A200h	A3FFh	A400h	A5FFh	A600h	A7FFh
第二十一扇区		第二十二扇区		第二十三扇区		第二十四扇区	
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址
A800h	A9FFh	AA00h	ABFFh	AC00h	ADFFh	AE00h	AFFFH
第二十五扇区		第二十六扇区		第二十七扇区		第二十八扇区	
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址
B000h	B1FFh	B200h	B3FFh	B400h	B5FFh	B600h	B7FFH
第二十九扇区		第三十扇区		第三十一扇区		第三十二扇区	
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址
B800h	B9FFh	BA00h	BBFFh	BC00h	BDFFh	BE00h	BFFFH
第三十三扇区		第三十四扇区		第三十五扇区		第三十六扇区	
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址
C000h	C1FFh	C200h	C3FFh	C400h	C5FFh	C600h	C7FFH
第三十七扇区		第三十八扇区		第三十九扇区		第四十扇区	
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址
C800h	C9FFh	CA00h	CBFFh	CC00h	CDFFh	CE00h	CFFFH
第四十一扇区		第四十二扇区		第四十三扇区		第四十四扇区	
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址
D000h	D1FFh	D200h	D3FFh	D400h	D5FFh	D600h	D7FFH
第四十五扇区		第四十六扇区		第四十七扇区		第四十八扇区	
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址
D800h	D9FFh	DA00h	DBFFh	DC00h	DDFFh	DE00h	DFFFH
第四十九扇区		第五十扇区		第五十一扇区		第五十二扇区	
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址
E000h	E1FFh	E200h	E3FFh	E400h	E5FFh	E600h	E7FFH

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

STC89C58RD+系列单片机内部EEPROM详细地址表

具体某型号有多少扇区的EEPROM,参照前面的EEPROM空间大小选型一览表， 每个扇区0.5 K字节

第五十三扇区		第五十四扇区		第五十五扇区		第五十六扇区		每个扇区512字节
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	
E800h	E9FFh	EA00h	EBFFh	EC00h	EDFFh	EE00h	EFFFH	
第五十七扇区		第五十八扇区						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，不必用满， 当然可全用
起始地址	结束地址	起始地址	结束地址					
F000h	F1FFh	F200h	F3FFh					

9.3 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    0E2h;   或      IAP_DATA        EQU     0E2h
IAP_ADDRH       DATA    0E3h;   或      IAP_ADDRH       EQU     0E3h
IAP_ADDRL       DATA    0E4h;   或      IAP_ADDRL       EQU     0E4h
IAP_CMD         DATA    0E5h;   或      IAP_CMD         EQU     0E5h
IAP_TRIG        DATA    0E6h;   或      IAP_TRIG        EQU     0E6h
IAP_CONTR       DATA    0E7h;   或      IAP_CONTR       EQU     0E7h

;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

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,命令不需改变时,不需重新送命令
MOV     IAP_TRIG,       #46h ;先送46h,再送B9h到ISP/IAP触发寄存器,每次都需如此
MOV     IAP_TRIG,       #0B9h           ;送完B9h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00,指向非EEPROM区
    ;MOV    IAP_ADDRL,      #0FFh           ;送地址低字节单元为00,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
    MOV     IAP_CONTR,      #WAIT_TIME          ;设置等待时间
    ORL     IAP_CONTR,      #10000000B          ;允许ISP/IAP操作
    MOV     IAP_CMD,        #ISP_IAP_BYTE_PROGRAM       ;送字节编程命令
    MOV     IAP_TRIG,       #46h    ;先送46h,再送B9h到ISP/IAP触发寄存器,每次都需如此
    MOV     IAP_TRIG,       #0B9h   ;送完B9h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00,;指向非EEPROM区,防止误操作
    ;MOV    IAP_ADDRL,      #0FFh   ;送地址低字节单元为00,指向非EEPROM区,防止误操作
```

地址需要改变时才需重新送地址

此两句可合成一句，并且只送一次就够了

```
;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，操作起来越灵活越快.
;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
        MOV     IAP_CMD,        #ISP_IAP_SECTOR_ERASE
                                        ;送扇区擦除命令,命令不需改变时,不需重新送命令
        MOV     IAP_TRIG,       #46h
                                ;先送46h,再送B9h到ISP/IAP触发寄存器,每次都需如此
        MOV     IAP_TRIG,       #0B9h       ;送完B9h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00,指向非EEPROM区
        ;MOV    IAP_ADDRL,      #0FFh           ;送地址低字节单元为00,防止误操作
```

此两句可以合成一句,并且只送一次就够了

小常识： (STC单片机的Data Flash 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另外的扇区,就不须读出保护。

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10uS/60uS/10mS。

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)。

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46和B9触发后，下一次IAP命令是否还需要送46和B9触发？

答：是，一定要。

9.4 EEPROM测试程序（C程序及汇编程序）

1. C程序：

```
;STC89C51RC/RD+系列单片机EEPROM/IAP 功能测试程序演示
/*---------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ----------------------------------*/
/* --- 演示STC89xx系列单片机  EEPROM/IAP功能-------------------*/
/* --- Mobile: (86)13922805190 -------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序，  ----------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -----------*/
/*---------------------------------------------------------------------------*/

#include "reg51.h"
#include "intrins.h"

typedef unsigned char BYTE;
typedef unsigned int WORD;

/*Declare SFR associated with the IAP */
sfr IAP_DATA    =  0xE2;        //Flash data register
sfr IAP_ADDRH   =  0xE3;        //Flash address HIGH
sfr IAP_ADDRL   =  0xE4;        //Flash address LOW
sfr IAP_CMD     =  0xE5;        //Flash command register
sfr IAP_TRIG    =  0xE6;        //Flash command trigger
sfr IAP_CONTR   =  0xE7;        //Flash control register

/*Define ISP/IAP/EEPROM command*/
#define CMD_IDLE    0                   //Stand-By
#define CMD_READ    1                   //Byte-Read
#define CMD_PROGRAM 2                   //Byte-Program
#define CMD_ERASE   3                   //Sector-Erase

/*Define ISP/IAP/EEPROM operation const for IAP_CONTR*/
//#define  ENABLE_IAP 0x80           //if SYSCLK<40MHz
#define    ENABLE_IAP   0x81         //if SYSCLK<20MHz
//#define  ENABLE_IAP x82            //if SYSCLK<10MHz
//#define  ENABLE_IAP 0x83           //if SYSCLK<5MHz

//Start address for STC89C58xx EEPROM
#define IAP_ADDRESS 0x08000

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 System Reset OK
    Delay(10);                          //Delay
    IapEraseSector(IAP_ADDRESS);        //Erase current sector
    for (i=0; i<512; i++)               //Check whether all sector data is FF
    {
        if (IapReadByte(IAP_ADDRESS+i) != 0xff)
            goto Error;                 //If error, break
    }
    P1 = 0xfc;                          //1111,1100 Erase successful
    Delay(10);                          //Delay
    for (i=0; i<512; i++)               //Program 512 bytes data into data flash
    {
        IapProgramByte(IAP_ADDRESS+i, (BYTE)i);
    }
    P1 = 0xf8;                          //1111,1000 Program successful
    Delay(10);                          //Delay
    for (i=0; i<512; i++)               //Verify 512 bytes data
    {
        if (IapReadByte(IAP_ADDRESS+i) != (BYTE)i)
            goto Error;                 //If error, break
    }
    P1 = 0xf0;                          //1111,0000 Verify successful
    while (1);
Error:
    P1 &= 0x7f;                         //0xxx,xxxx IAP operation fail
    while (1);
}

/*-----------------------------
Software delay function
-----------------------------*/
void Delay(BYTE n)
{
    WORD x;

    while (n--)
    {
        x = 0;
        while (++x);
    }
}
```

```
/*---------------------------
Disable ISP/IAP/EEPROM function
Make MCU in a safe state
---------------------------*/
void IapIdle()
{
    IAP_CONTR = 0;                  //Close IAP function
    IAP_CMD = 0;                    //Clear command to standby
    IAP_TRIG = 0;                   //Clear trigger register
    IAP_ADDRH = 0x80;               //Data ptr point to non-EEPROM area
    IAP_ADDRL = 0;                  //Clear IAP address to prevent misuse
}

/*---------------------------
Read one byte from ISP/IAP/EEPROM area
Input: addr (ISP/IAP/EEPROM address)
Output:Flash data
---------------------------*/
BYTE IapReadByte(WORD addr)
{
    BYTE dat;                       //Data buffer

    IAP_CONTR = ENABLE_IAP;         //Open IAP function, and set wait time
    IAP_CMD = CMD_READ;             //Set ISP/IAP/EEPROM READ command
    IAP_ADDRL = addr;               //Set ISP/IAP/EEPROM address low
    IAP_ADDRH = addr >> 8;          //Set ISP/IAP/EEPROM address high
    IAP_TRIG = 0x46;                //Send trigger command1 (0x46)
    IAP_TRIG = 0xb9;                //Send trigger command2 (0xb9)
    _nop_();                        //MCU will hold here until ISP/IAP/EEPROM operation complete
    dat = IAP_DATA;                 //Read ISP/IAP/EEPROM data
    IapIdle();                      //Close ISP/IAP/EEPROM function

    return dat;                     //Return Flash data
}

/*---------------------------
Program one byte to ISP/IAP/EEPROM area
Input: addr (ISP/IAP/EEPROM address)
       dat (ISP/IAP/EEPROM data)
Output:-
---------------------------*/
```

```
void IapProgramByte(WORD addr, BYTE dat)
{
    IAP_CONTR = ENABLE_IAP;                 //Open IAP function, and set wait time
    IAP_CMD = CMD_PROGRAM;                  //Set ISP/IAP/EEPROM PROGRAM command
    IAP_ADDRL = addr;                       //Set ISP/IAP/EEPROM address low
    IAP_ADDRH = addr >> 8;                  //Set ISP/IAP/EEPROM address high
    IAP_DATA = dat;                         //Write ISP/IAP/EEPROM data
    IAP_TRIG = 0x46;                        //Send trigger command1 (0x46)
    IAP_TRIG = 0xb9;                        //Send trigger command2 (0xb9)
    _nop_();                        //MCU will hold here until ISP/IAP/EEPROM operation complete
    IapIdle();
}

/*-----------------------------
Erase one sector area
Input: addr (ISP/IAP/EEPROM address)
Output:-
-----------------------------*/
void IapEraseSector(WORD addr)
{
    IAP_CONTR = ENABLE_IAP;                 //Open IAP function, and set wait time
    IAP_CMD = CMD_ERASE;                    //Set ISP/IAP/EEPROM ERASE command
    IAP_ADDRL = addr;                       //Set ISP/IAP/EEPROM address low
    IAP_ADDRH = addr >> 8;                  //Set ISP/IAP/EEPROM address high
    IAP_TRIG = 0x46;                        //Send trigger command1 (0x46)
    IAP_TRIG = 0xb9;                        //Send trigger command2 (0xb9)
    _nop_();                        //MCU will hold here until ISP/IAP/EEPROM operation complete
    IapIdle();
}
```

2. 汇编程序：

```
;STC89C51RC/RD+系列单片机EEPROM/IAP 功能测试程序演示
/*---------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------- ----------------------------------*/
/* --- 演示STC89xx系列单片机  EEPROM/IAP功能-------------------------*/
/* --- Mobile: (86)13922805190 -------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序，  ----------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -----------*/
/*---------------------------------------------------------------------------*/

;/*Declare SFR associated with the IAP */
IAP_DATA    EQU 0E2H            ;Flash data register
IAP_ADDRH   EQU 0E3H              ;Flash address HIGH
IAP_ADDRL   EQU 0E4H              ;Flash address LOW
IAP_CMD     EQU 0E5H            ;Flash command register
IAP_TRIG    EQU 0E6H            ;Flash command trigger
IAP_CONTR   EQU 0E7H              ;Flash control register

;/*Define ISP/IAP/EEPROM command*/
CMD_IDLE    EQU 0               ;Stand-By
CMD_READ    EQU 1               ;Byte-Read
CMD_PROGRAM EQU 2                 ;Byte-Program
CMD_ERASE   EQU 3               ;Sector-Erase

;/*Define ISP/IAP/EEPROM operation const for IAP_CONTR*/
;ENABLE_IAP EQU 80H              ;if SYSCLK<40MHz
ENABLE_IAP  EQU 81H             ;if SYSCLK<20MHz
;ENABLE_IAP EQU 82H              ;if SYSCLK<10MHz
;ENABLE_IAP EQU 83H              ;if SYSCLK<5MHz
;//Start address for STC89C58xx EEPROM

IAP_ADDRESS EQU 08000H
;------------------------------------------
   ORG   0000H
   LJMP  MAIN
;------------------------------------------

          ORG   0100H
MAIN:
   MOV   P1,#0FEH              ;1111,1110 System Reset OK
   LCALL DELAY                  ;Delay
```

```
;------------------------------
        MOV     DPTR,   #IAP_ADDRESS        ;Set ISP/IAP/EEPROM address
        LCALL   IAP_ERASE                   ;Erase current sector
;------------------------------
        MOV     DPTR,   #IAP_ADDRESS        ;Set ISP/IAP/EEPROM address
        MOV     R0,     #0                  ;Set counter (512)
        MOV     R1,     #2
CHECK1:                                     ;Check whether all sector data is FF
        LCALL   IAP_READ                    ;Read Flash
        CJNE    A,      #0FFH,  ERROR       ;If error, break
        INC     DPTR                        ;Inc Flash address
        DJNZ    R0,     CHECK1              ;Check next
        DJNZ    R1,     CHECK1              ;Check next
;------------------------------
        MOV     P1,     #0FCH               ;1111,1100 Erase successful
        LCALL DELAY                         ;Delay
;------------------------------
        MOV     DPTR,   #IAP_ADDRESS        ;Set ISP/IAP/EEPROM address
        MOV     R0,     #0                  ;Set counter (512)
        MOV     R1,     #2
        MOV     R2,     #0                  ;Initial test data
NEXT:                                       ;Program 512 bytes data into data flash
        MOV     A,      R2                  ;Ready IAP data
        LCALL   IAP_PROGRAM                 ;Program flash
        INC     DPTR                        ;Inc Flash address
        INC     R2                          ;Modify test data
        DJNZ    R0,     NEXT                ;Program next
        DJNZ    R1,     NEXT                ;Program next
;------------------------------
        MOV     P1,     #0F8H               ;1111,1000 Program successful
        LCALL   DELAY                       ;Delay
;------------------------------
        MOV     DPTR,   #IAP_ADDRESS        ;Set ISP/IAP/EEPROM address
        MOV     R0,     #0                  ;Set counter (512)
        MOV     R1,     #2
        MOV     R2,     #0
CHECK2:                                     ;Verify 512 bytes data
        LCALL   IAP_READ                    ;Read Flash
        CJNE    A,2,ERROR                   ;If error, break
        INC     DPTR                        ;Inc Flash address
        INC     R2                          ;Modify verify data
        DJNZ    R0,     CHECK2              ;Check next
        DJNZ    R1,     CHECK2              ;Check next
```

```
;-------------------------------
        MOV     P1,     #0F0H           ;1111,0000 Verify successful
        SJMP    $
;-------------------------------
ERROR:
        MOV     P0,     R0
        MOV     P2,     R1
        MOV     P3,     R2
        CLR     P1.7                    ;0xxx,xxxx IAP operation fail
        SJMP    $

;/*---------------------------
;Software delay function
;---------------------------*/
DELAY:
        CLR     A
        MOV     R0,     A
        MOV     R1,     A
        MOV     R2,     #20H
DELAY1:
        DJNZ    R0,     DELAY1
        DJNZ    R1,     DELAY1
        DJNZ    R2,     DELAY1
        RET

;/*---------------------------
;Disable ISP/IAP/EEPROM function
;Make MCU in a safe state
;---------------------------*/
IAP_IDLE:
        MOV   IAP_CONTR,        #0              ;Close IAP function
        MOV   IAP_CMD, #0                       ;Clear command to standby
        MOV   IAP_TRIG, #0                      ;Clear trigger register
        MOV   IAP_ADDRH,        #80H            ;Data ptr point to non-EEPROM area
        MOV   IAP_ADDRL,        #0              ;Clear IAP address to prevent misuse
        RET

;/*---------------------------
;Read one byte from ISP/IAP/EEPROM area
;Input: DPTR(ISP/IAP/EEPROM address)
;Output:ACC (Flash data)
;---------------------------*/
```

```
IAP_READ:
        MOV   IAP_CONTR,        #ENABLE_IAP             ;Open IAP function, and set wait time
        MOV   IAP_CMD,          #CMD_READ               ;Set ISP/IAP/EEPROM READ command
        MOV   IAP_ADDRL,        DPL                     ;Set ISP/IAP/EEPROM address low
        MOV   IAP_ADDRH,        DPH                     ;Set ISP/IAP/EEPROM address high
        MOV   IAP_TRIG,         #46H                    ;Send trigger command1 (0x46)
        MOV   IAP_TRIG,         #0B9H                   ;Send trigger command2 (0xb9)
        NOP                     ;MCU will hold here until ISP/IAP/EEPROM operation complete
        MOV   A,          IAP_DATA                      ;Read ISP/IAP/EEPROM data
        LCALL IAP_IDLE                                  ;Close ISP/IAP/EEPROM function
        RET

;/*---------------------------
;Program one byte to ISP/IAP/EEPROM area
;Input: DPAT(ISP/IAP/EEPROM address)
;       ACC (ISP/IAP/EEPROM data)
;Output:-
;---------------------------*/
IAP_PROGRAM:
        MOV   IAP_CONTR,        #ENABLE_IAP     ;Open IAP function, and set wait time
        MOV   IAP_CMD, #CMD_PROGRAM             ;Set ISP/IAP/EEPROM PROGRAM command
        MOV   IAP_ADDRL,        DPL             ;Set ISP/IAP/EEPROM address low
        MOV   IAP_ADDRH,        DPH             ;Set ISP/IAP/EEPROM address high
        MOV   IAP_DATA,         A               ;Write ISP/IAP/EEPROM data
        MOV   IAP_TRIG, #46H                    ;Send trigger command1 (0x46)
        MOV   IAP_TRIG, #0B9H                   ;Send trigger command2 (0xb9)
        NOP             ;MCU will hold here until ISP/IAP/EEPROM operation complete
        LCALL IAP_IDLE  ;Close ISP/IAP/EEPROM function
        RET

;/*---------------------------
;Erase one sector area
;Input: DPTR(ISP/IAP/EEPROM address)
;Output:-
;---------------------------*/
IAP_ERASE:
        MOV   IAP_CONTR,        #ENABLE_IAP     ;Open IAP function, and set wait time
        MOV   IAP_CMD, #CMD_ERASE               ;Set ISP/IAP/EEPROM ERASE command
        MOV   IAP_ADDRL,        DPL             Set ISP/IAP/EEPROM address low
        MOV   IAP_ADDRH,        DPH             ;Set ISP/IAP/EEPROM address high
        MOV   IAP_TRIG, #46H                    ;Send trigger command1 (0x46)
        MOV   IAP_TRIG, #0B9H                   ;Send trigger command2 (0xb9)
        NOP             ;MCU will hold here until ISP/IAP/EEPROM operation complete
        LCALL IAP_IDLE                          ;Close ISP/IAP/EEPROM function
        RET

        END
```

第10章 编译器(汇编器)/ISP编程器(烧录)/仿真器说明

10.1 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Philips的P87C52/P87C54/P87C58编译，头文件包含<reg51.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:

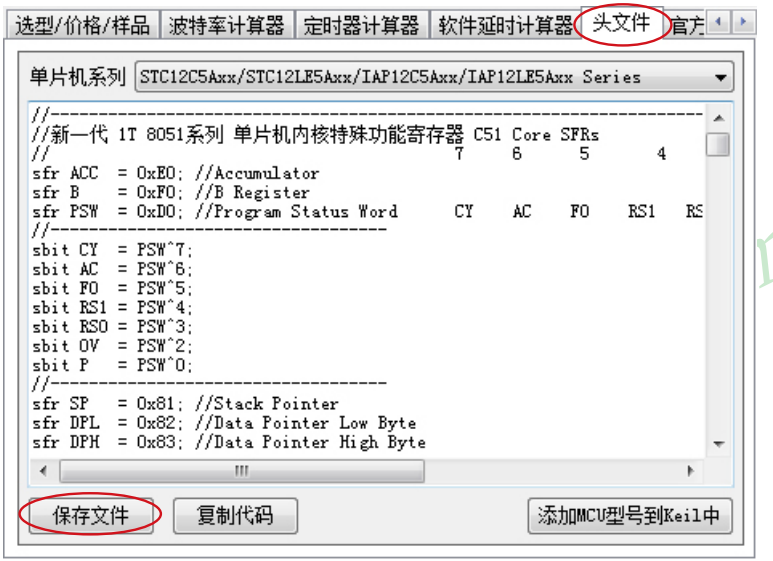
```
        P4      EQU     0C0H            ; or  P4        DATA    0C0H
        P4M1    EQU     0B3H            ; or  P4M1      DATA    0B3H
        P4M0    EQU     0B4H            ; or  P4M1      DATA    0B4H

        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STC官方网站www.STCMCU.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快捷键″ctrl+F″查找)，STC12系列单片机还可以用最新的ISP下载工具STC-ISP-15xx-V6.58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″。

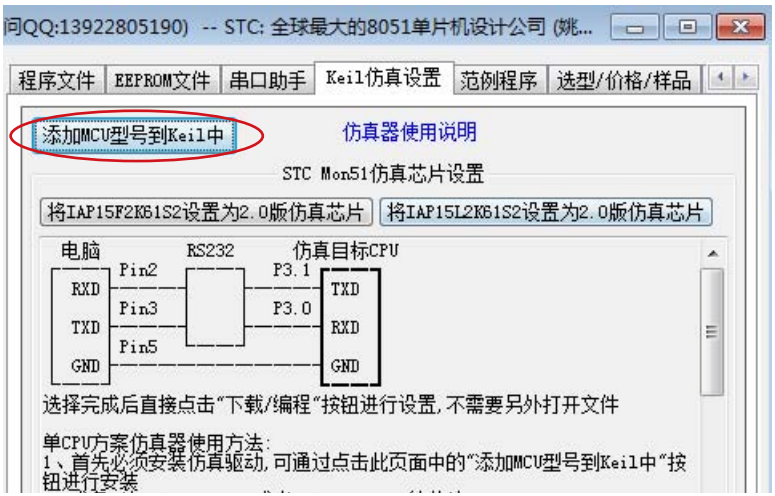


Keil C51集成开发环境有许多版本，而对于8051单片机最常用的版本有Keil μVision4、Keil μVision3及Keil μVision4.

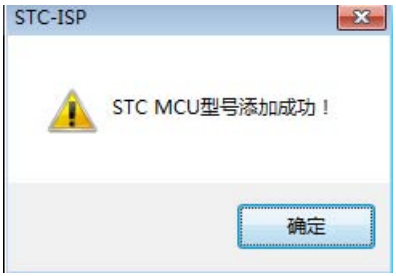
注意：由于STC系列单片机是新发展的芯片，一般情况下在Keil μVision2设备库中没有STC系列单片机。在编辑、编译STC系列单片机应用程序时，可选任何厂家的51或52系列单片机，再用汇编或C语言对STC系列单片机新增特殊功能寄存器进行定义，也可以通过STC-ISP下载编程工具将STC型号MCU添加到Keil μVision4或Keil μVision3或Keil μVision2的设备库中。

如果用户需在Keil μVision4或Keil μVision3或Keil μVision2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STC-ISP下载编程工具的最新软件STC-ISP-V6.58，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。



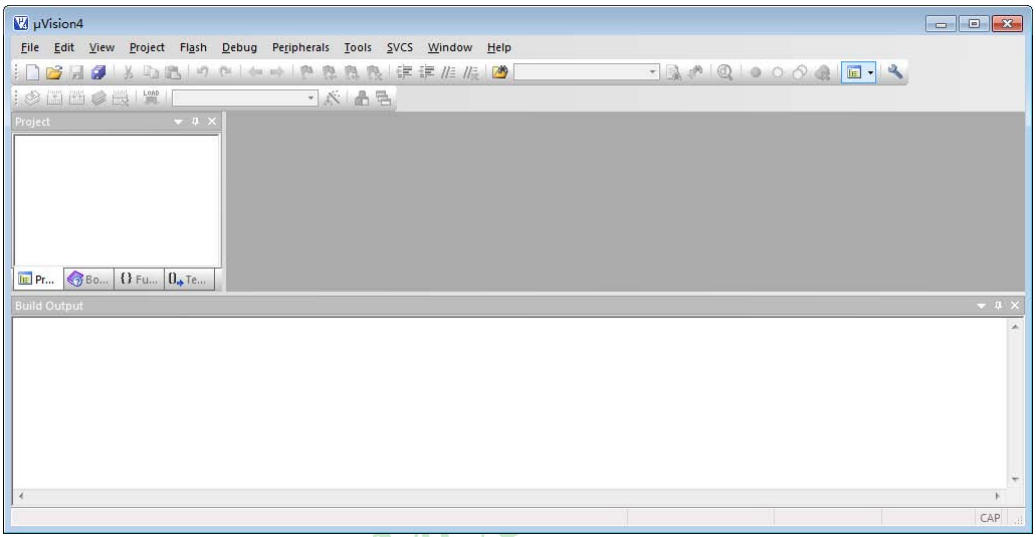
② 在弹出的“浏览文件夹”对话框中选择Keil安装目录（一般可能为“C:\keil”），然后单击【确定】，这样就将STC型号的MCU成功添加到Keil μVision2设备库中了。



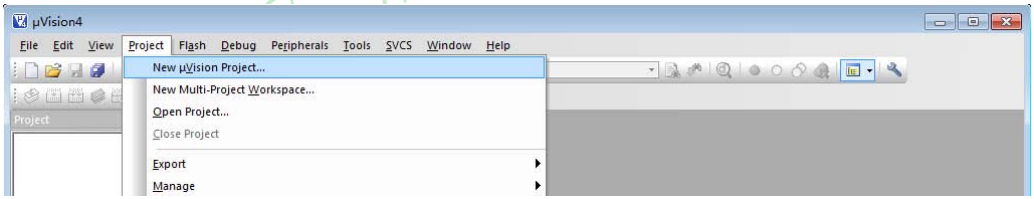
下面以Keil μVision4为例，详细介绍如何使用Keil μVision4开发、编译、调试用户程序。

一、如何新建项目及在所新建的项目中添加STC型号MCU进行开发、编译、调试用户程序：

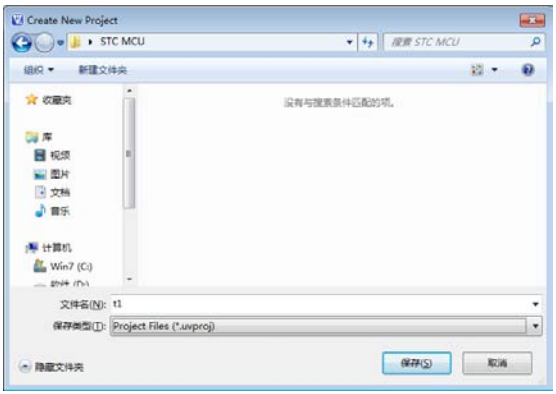
（1）启动Keil μVision4，进入Keil μVision4后的编辑界面如下所示：



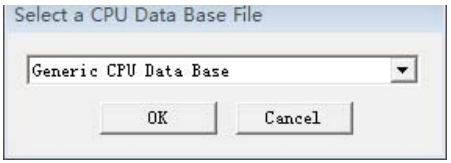
（2）建立一个新工程：单击Project菜单，在弹出的下拉菜单中选中New Project选项

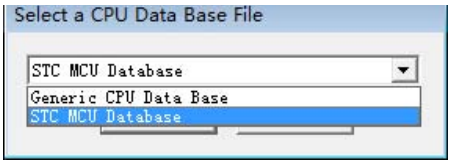


（3）在弹出的对话框中选择新项目要保存的路径和文件名，例如：保存路径为C:\Users\THINK\Documents\STC MCU，项目名为t1，单击保存即可。Keil μVision4的项目文件扩展名为.uvproj

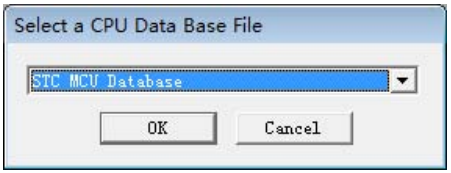


（4）因之前已经通过STC-ISP下载编程工具将STC型号MCU添加到Keil μVision2的设备库中，所以在上一步【保存】之后会弹出“选择设备数据库”的对话框，如下图所示。该“选择设备数据库”的对话框中有“通用CPU数据库(Generic CPU Database)”和“STC MCU数据库(STC MCU Database)”两个选项。

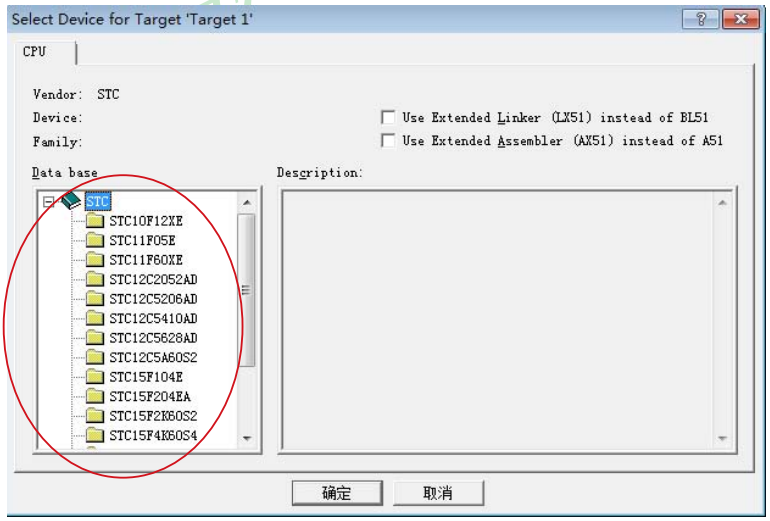




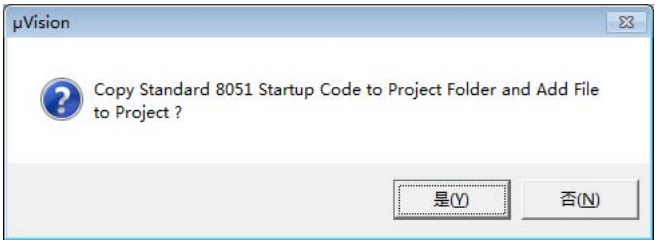
如用户所使用的单片机是STC系列单片机，则在这里选择“STC MCU数据库(STC MCU Database)”，点击【OK】按钮确定。



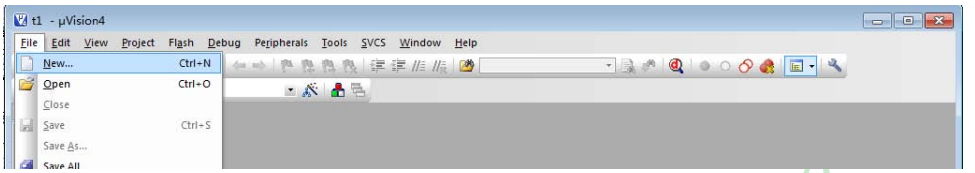
（5）在上一步“选择设备数据库”后会弹出"Select Device for Target"对话框，如下所示。因上一步中我们选择了“STC MCU数据库(STC MCU Database)”，所以这里的MCU型号都是STC型号，用户可在左侧的数据列表(Data base)选择自己所使用的具体单片机型号。



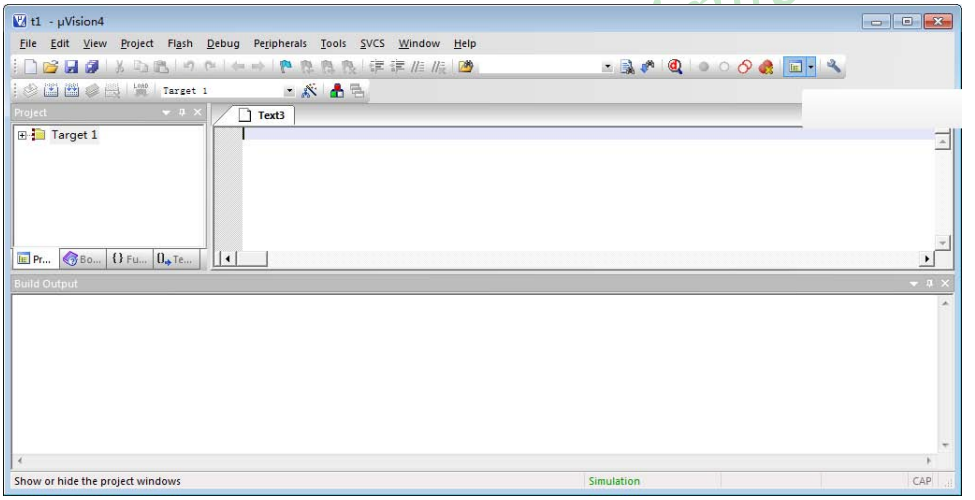
（6）选择好单片机型号并点击确定后，程序会询问是否将标准51初始化程序(STARTUP.51)加入到项目中，如下图所示。选择【是】按钮，程序会自动复制标准51初始化程序到项目所在目录并将其加入项目中。一般情况下，选择【否】按钮



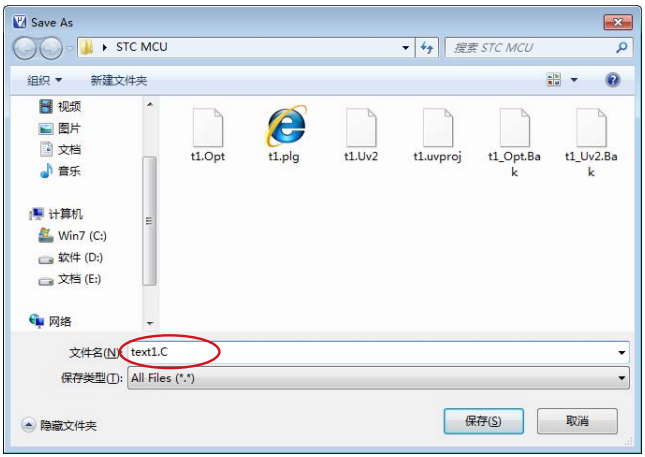
（7）项目建好后开始编写程序了，选择“File”菜单，再在下拉菜单中单击“New”选项



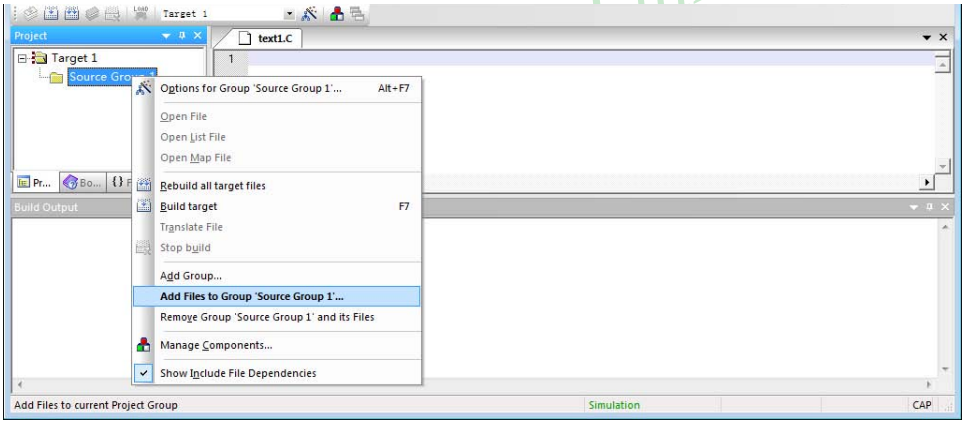
新建文件后界面如下图所示



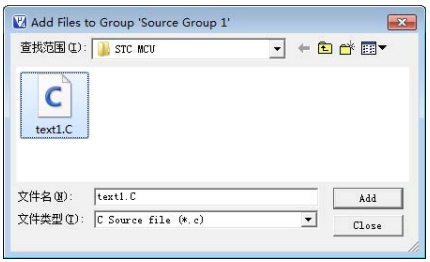
此时光标在编辑窗口里闪烁，这时可以键入用户的应用程序了，输入程序后单击菜单上的“File”，在下拉菜单中选中“Save As”选项单击，弹出如下图所示的界面，在“文件名”栏右侧的编辑框中，键入欲使用的文件名，同时必须键入正确的扩展名。注意，如果用C语言编写程序，则扩展名为(.C)；如果用汇编语言编写程序，则扩展名必须为(.ASM)，扩展名不分大小写。然后，单击“保存”按钮。



（8）将应用程序添加到项目中：单击“Target 1”前面的“＋”号，然后在“Source Group 1”上单击右键，弹出如下菜单

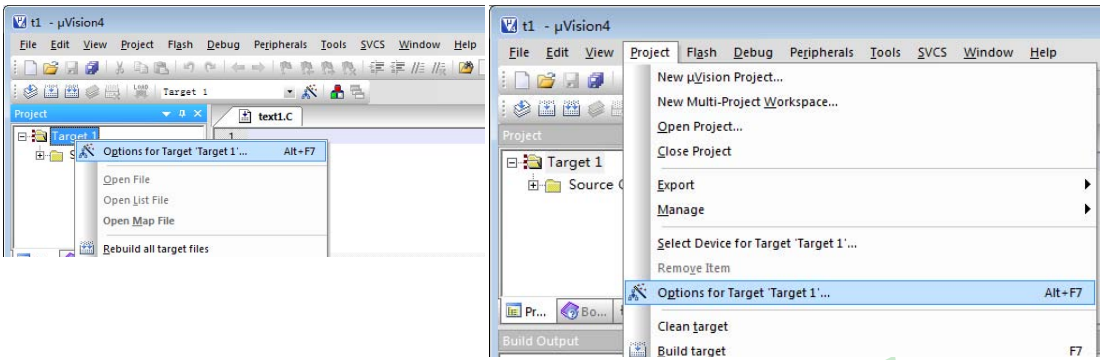


然后单击“Add File to Group ‘Source Group 1’ ”，弹出如下图所示的界面

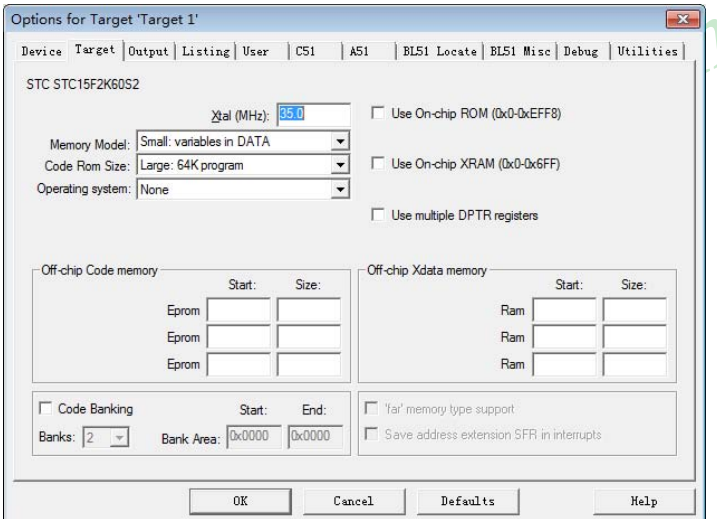


选中text1.c，然后单击“Add”添加成功。

（9）环境设置：在″Target 1"上单击右键选择Options for Target 'Target1'或选择菜单命令Project → Options for Target 'Target1'，弹出Options for Target 'Target1'对话框。

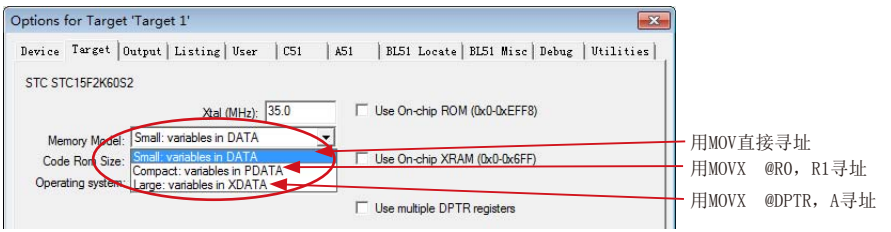


使用Options for Target 'Target1'对话框设定目标的硬件环境。

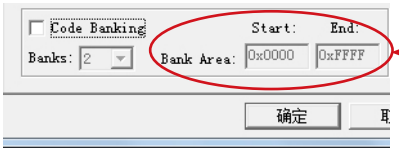


Options for Target 'Target1'对话框有多个选项页，用于设备(Device)选择、目标(Target)属性、输出(Output)属性、C51编译器属性、A51编译器属性、BL51连接器属性、调试(Debug)属性等信息的设置。一般情况下按缺省设置，下面介绍几个需用户自己设置的选项。

① 数据存储器的选择

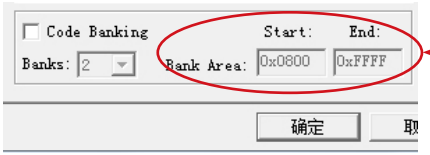


② 程序代码区的起始地址和结束地址默认如下图所示，默认的起始地址或结束地址是合法的。



←合法的起始地址和结束地址

但下图的起始地址或结束地址是不合法的，用户须将其修改成为合法的起始地址和结束地址。

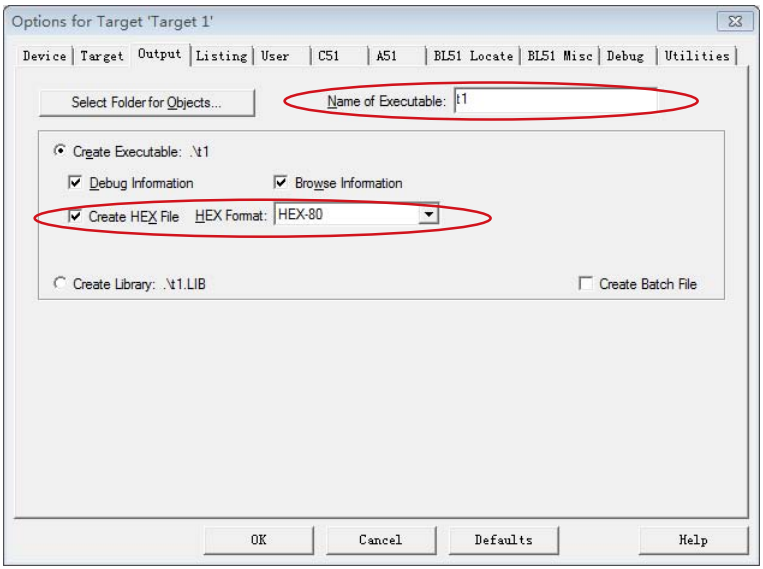


←非法的起始地址和结束地址

具体做法如下：先勾选“Code Banking”选项，然后修改“Bank Area”的起始地址和结束地址，最后去选“Code Banking”选项(记住一定要去选此项)，点击【确定】，这样程序代码区的起始地址和结束地址就设置好了。

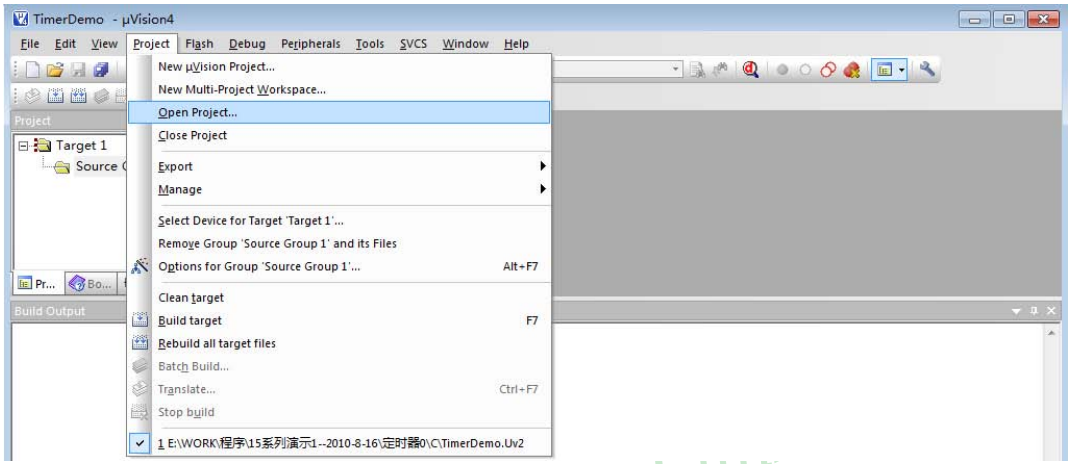
③ 设置在编译、连接程序时自动生成机器代码文件(.HEX)，一定要设置此项，因为默认是不输出HEX代码的，所以需用户设置。

单击“Output”中选项，在弹出的Output对话框中勾选“Create HEX File”选项(如下图所示)，使程序编译后产生HEX代码文件(默认文件名为项目文件名，也可以在“Name of Executable”信息框中输入HEX文件的文件名)，点击【确定】按钮结束设置。

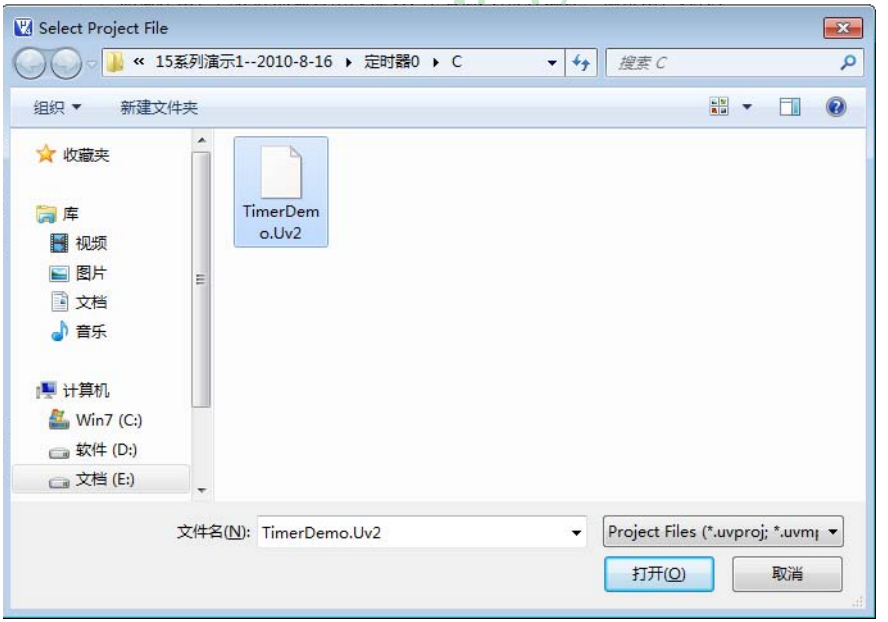


二、如何在用户已建好的项目中改选STC型号MCU进行编译、调试用户程序：

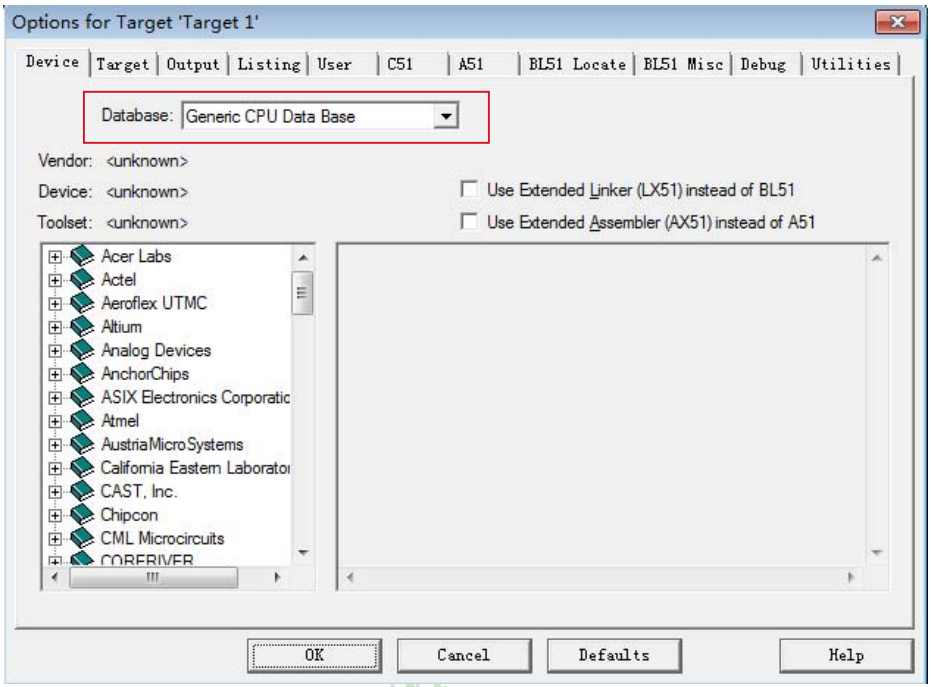
（1）启动Keil μVision4，并打开已建好的项目，如下图所示：



（2）启动Keil μVision4，并打开已建好的项目，在弹出的对话框“Select Project File”中选择目标项目文件，点击【打开】，如下图所示：

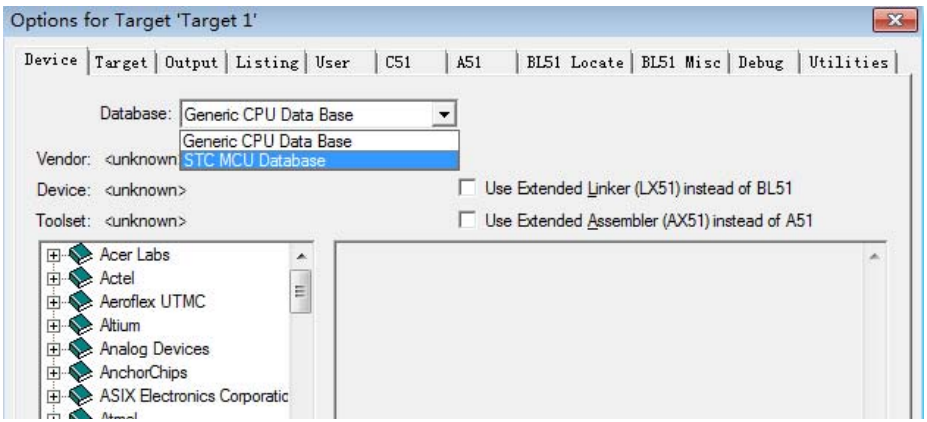


（2）在″Target 1"上单击右键选择Options for Target 'Target1'或选择菜单命令Project→ Options for Target 'Target1'，弹出Options for Target 'Target1'对话框，选择该对话框中“Device”页面，如下图所示：

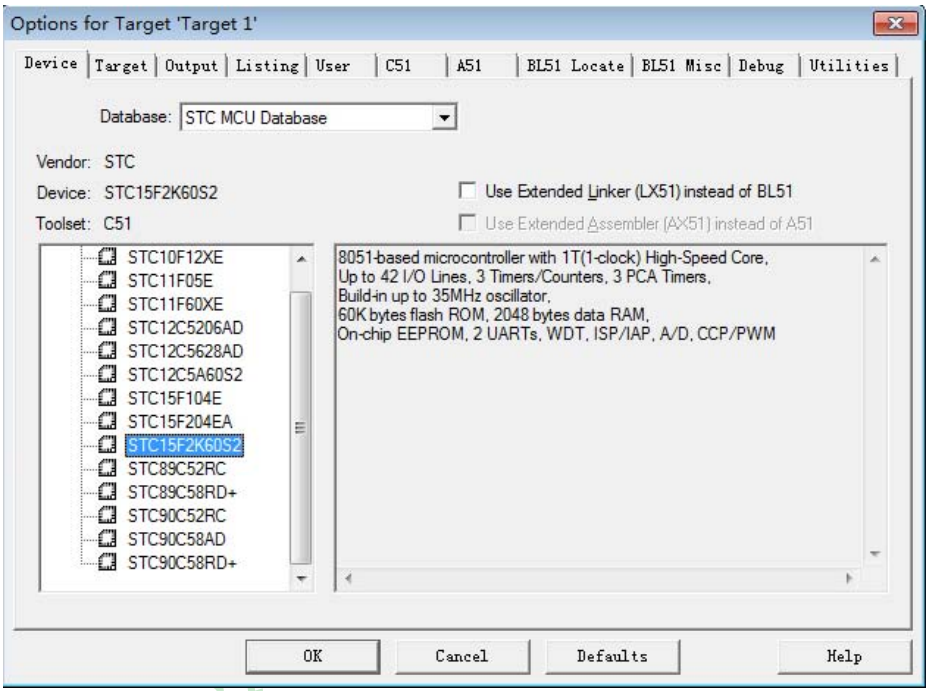


可以看到此时所使用的设备数据库为“通用CPU数据库(Generic CPU Database)”，如用户所使用的单片机为STC单片机，则需更改所使用的设备数据库，具体操作见以下步骤。

（3）因之前已经通过STC-ISP下载编程工具将STC型号MCU添加到Keil μVision4的设备库中(添加方法见上文)，所以此时“Device”页面的中“Database(数据库)”有两个下拉选项“通用CPU数据库(Generic CPU Database)”和“STC MCU数据库(STC MCU Database)”，如下图所示。



在下拉选项中选择“STC MCU数据库(STC MCU Database)”，确定后用户可在左下侧的设备列表选择自己所使用的具体单片机型号，如下图所示。



这样就成功地在已建好的项目中将原MCU改选成了STC型号MCU，接来接用户就可以进行编译、调试用户程序了。

10.2 USB型联机/脱机下载工具U8W/U8W-Mini/U8/U8-Mini

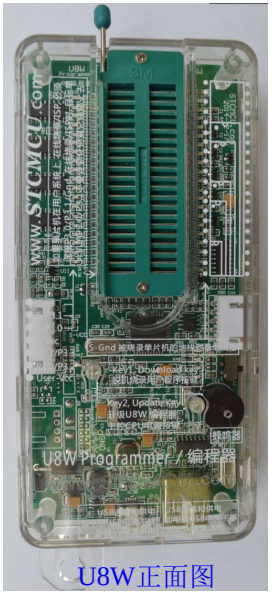
U8W/U8W-Mini及U8/U8-Mini是一款集在线联机下载和脱机下载于一体的编程工具系列。其中，U8编程工具分5V工具和3.3V工具，分别为U8-5V及U8-3.3V。U8W/U8W-Mini及U8/U8-Mini的应用范围可支持STC目前的全部系列的MCU, Flash程序空间和EEPROM数据空间不受限制。支持包括如下和即将推出的STC全系列芯片：

STC15W4K32S4系列
STC15F2K60S2/STC15L2K60S2系列
STC15W201S系列
STC15W401AS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/STC11Lxx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：



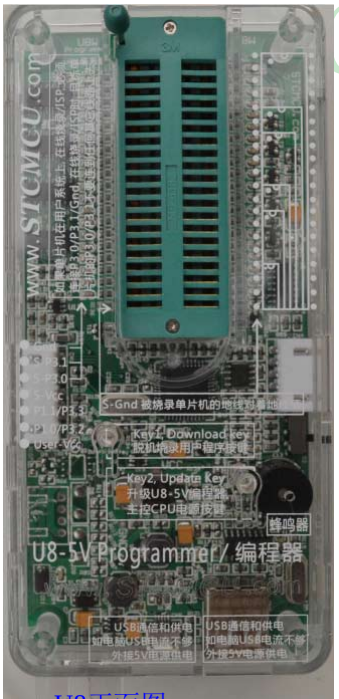
U8W正面图

U8W反面图

U8W-Mini正反面图

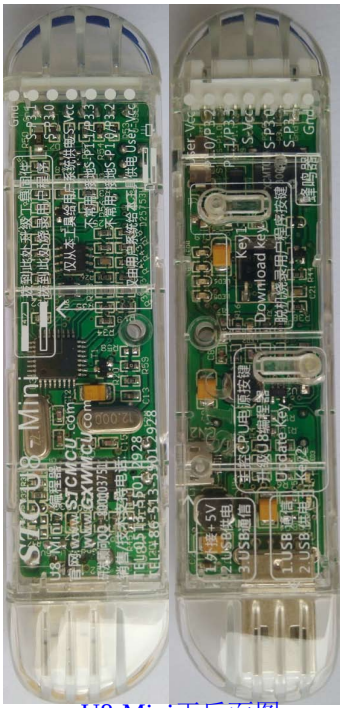
U8W-Mini工具的体积仅有一个U盘大小　其功能与U8W相同　但无锁紧座　价格仅为RMB 50元　欢迎来电订购

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：



U8正面图

U8反面图



U8-Mini正反面图

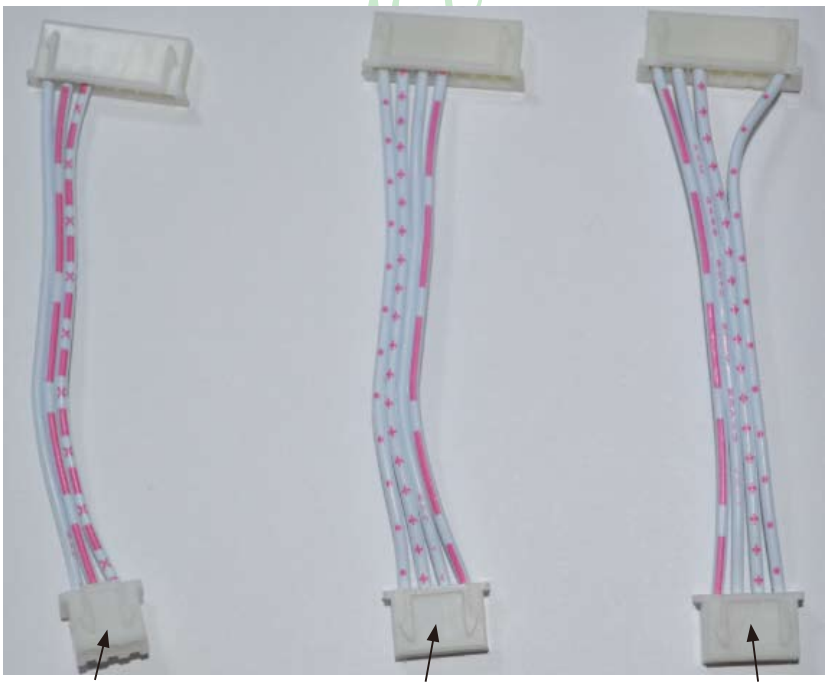
U8-Mini工具的体积仅有一个U盘大小　其功能与U8相同　但无锁紧座　价格仅为RMB 50元　欢迎来电订购

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

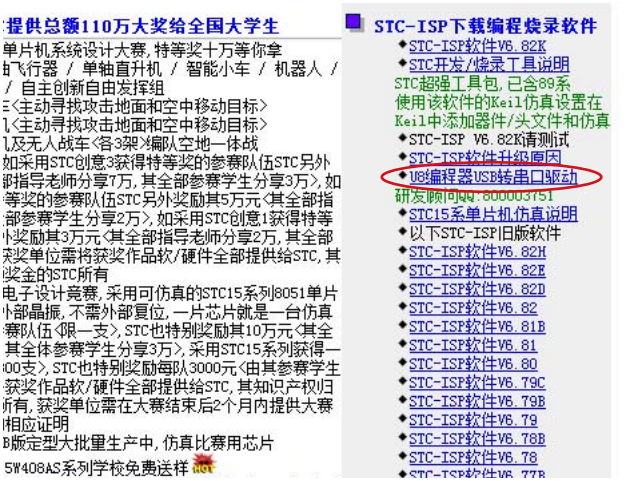
用户系统给U8W/U8W-Mini/U8/U8-Mini供电的连接线

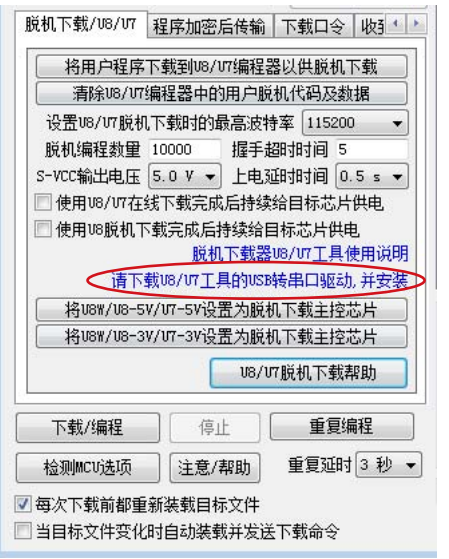
10.2.1 如何安装下载工具U8W/U8W-Mini/U8/U8-Mini的驱动程序

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STC-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.stcmcu.com/STCISP/CH341SER.exe）。网站上及STC-ISP下载软件上的驱动地址如下图所示：

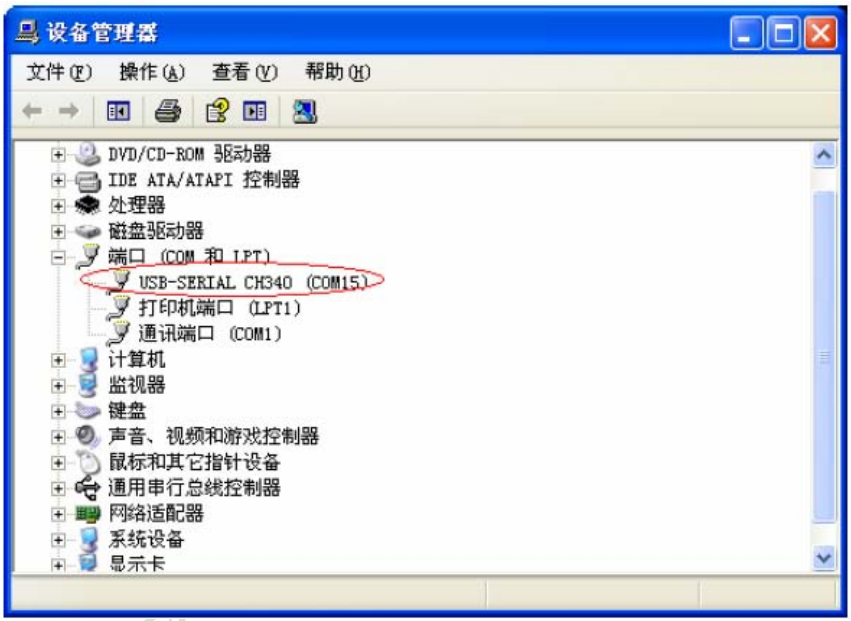




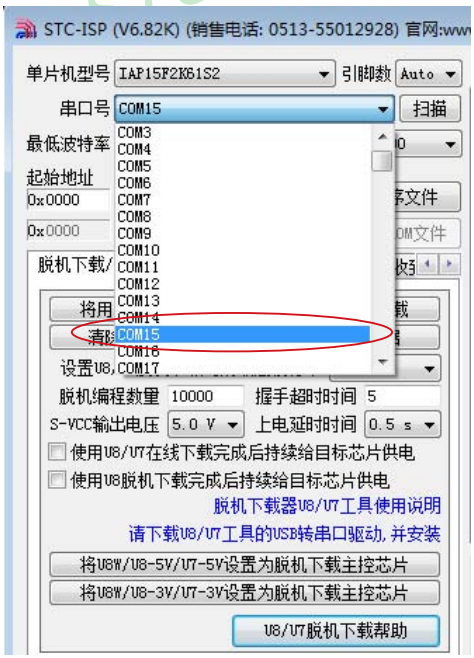
驱动程序下载到本机后，直接双击可执行程序并运行，出现下图所示的界面，点击“安装”按钮开始自动安装驱动

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）

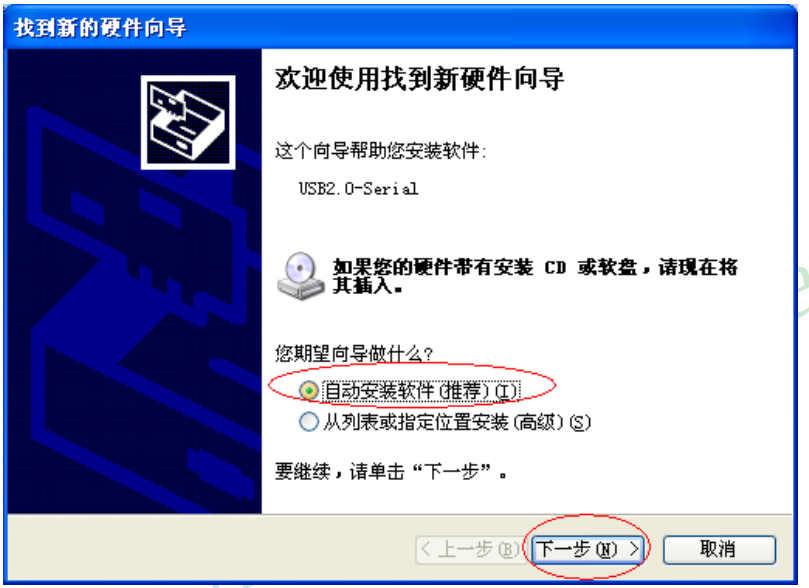


注意：在后面使用STC-ISP下载软件时，选择的串口号必须选择与此相对应的串口号，如下图所示



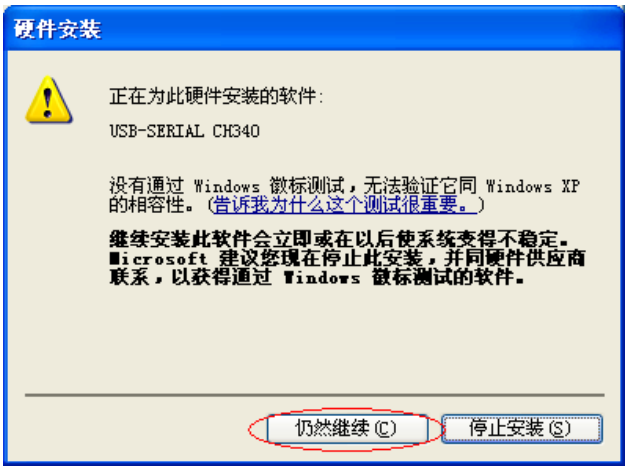
2、自动安装USB型联机/脱机下载工具U8的驱动程序

若用户所使用的 STC-ISP 下载软件为 V6.85K 及以上版本，则打开该 STC-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：

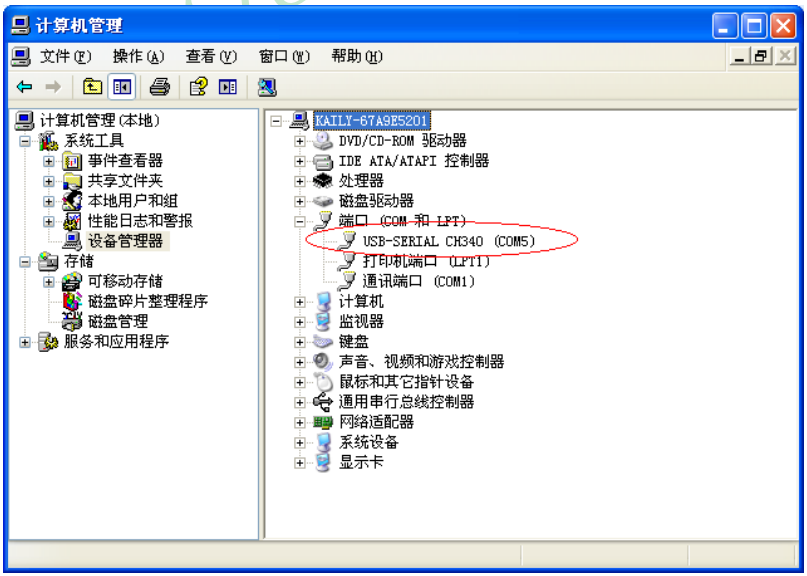
在接下来出现的如下面的对话框中，选中【仍然继续】：



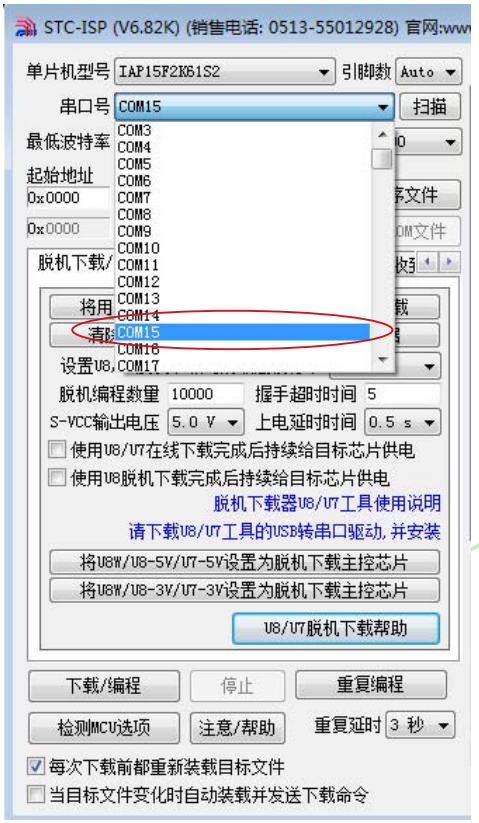
系统便会开始自动安装驱动，如下图所示：

直至出现如下画面，点击【完成】按钮：

至此，U8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）：

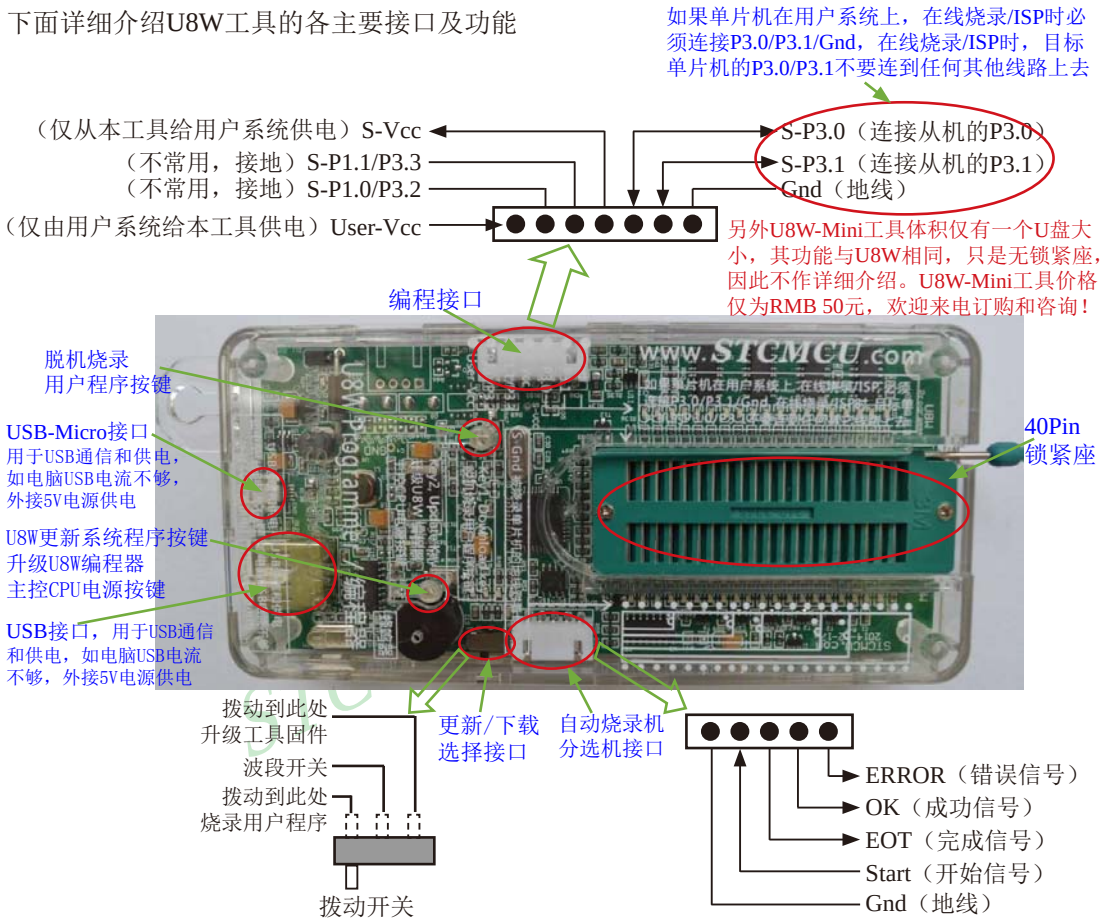


注意：在后面使用STC-ISP下载软件时，选择的串口号必须选择与此相对应的串口号，如下图所示：



10.2.2 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



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）。

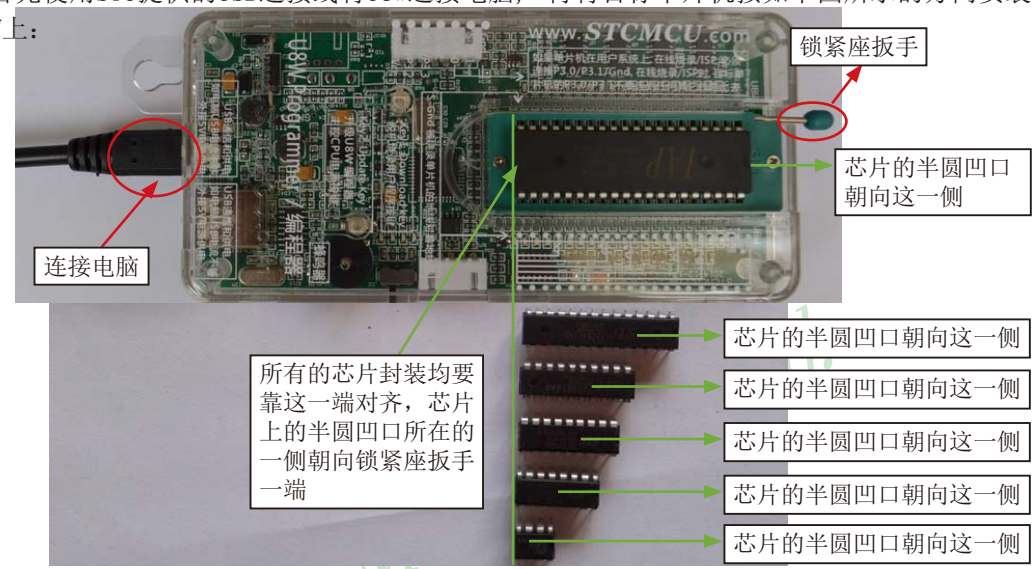
更新/下载选择接口：当需要对U8W的底层固件进行升级时，需将此拨动开关拨动到升级工具固件处，当需通过U8W对目标芯片进行烧录程序，则需将拨动开关拨动到烧录用户程序处。（拨动开关连接方式请参考上图）

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

10.2.3 U8W的在线联机下载使用说明

10.2.3.1 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STC-ISP下载软件下载程序时，在STC-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。

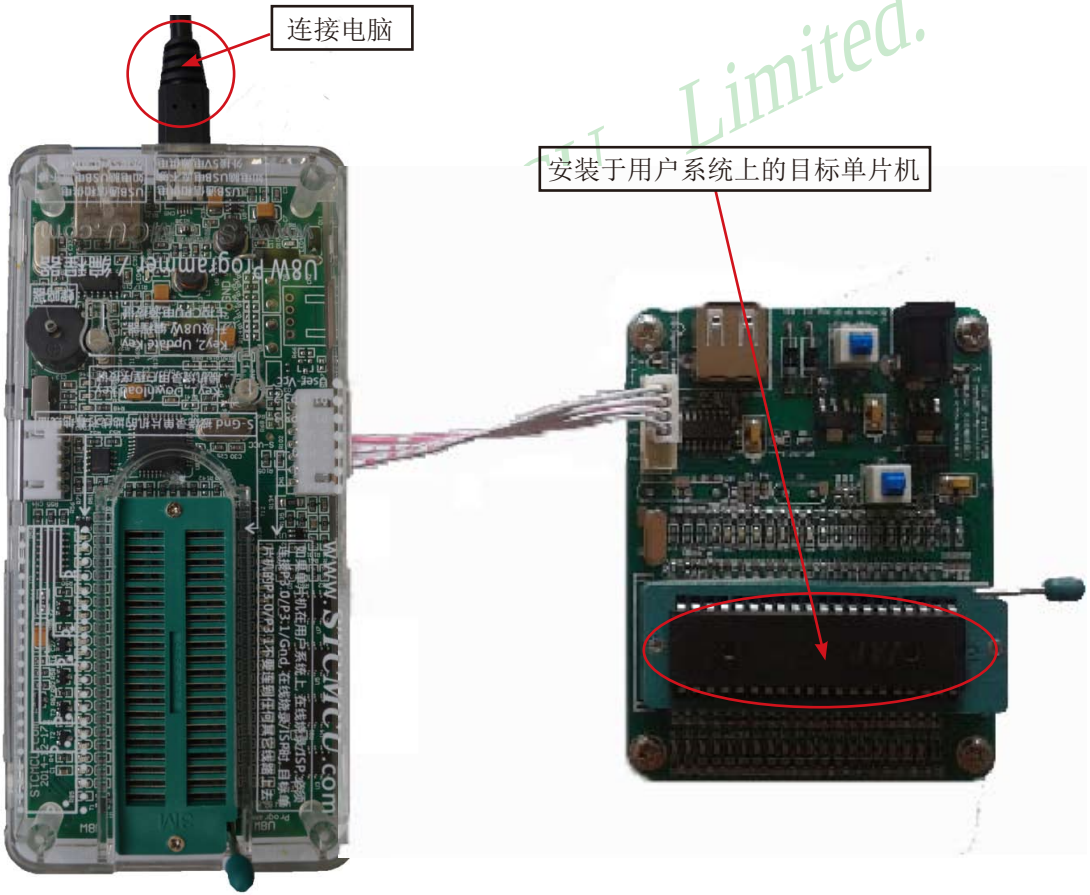
当信息框中有输出下载板的版本号信息以及外挂Flash的相应信息时，表示已正确检测到U8W下载工具。

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。

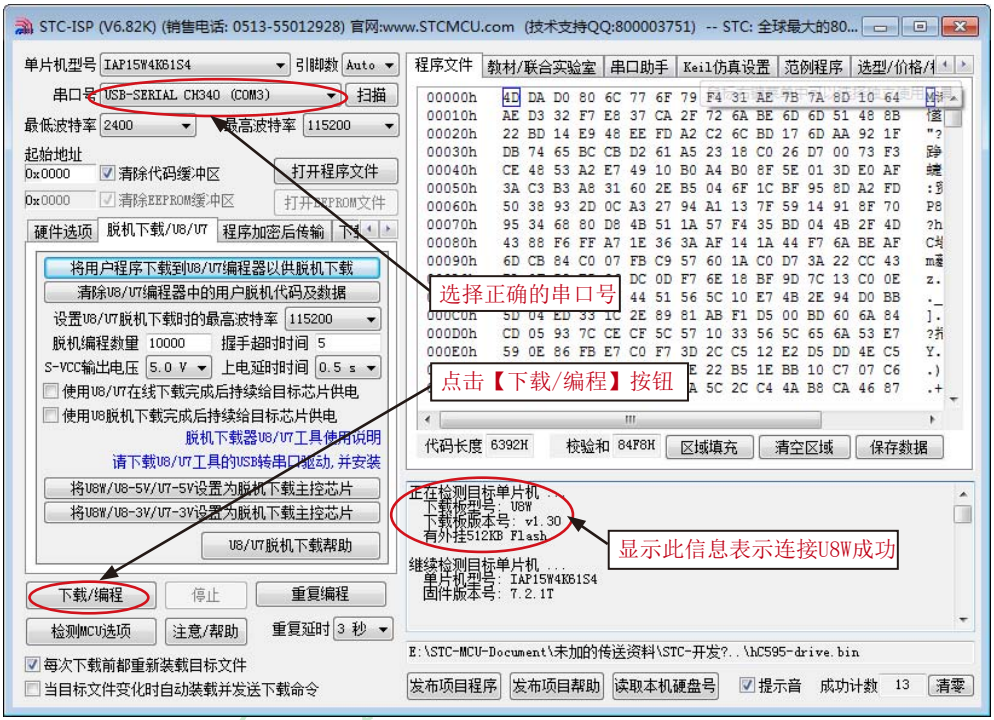
建议用户用最新版本的STC-ISP下载软件“STC-ISP (V6.85K).exe”（请随时留意STC官方网站http://www.STCMCU.com中STC-ISP下载软件的更新，强烈建议用户在官方网站http://www.STCMCU.com中下载最新版本的软件使用）。

10.2.3.2 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STC-ISP下载软件下载程序时，在STC-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。

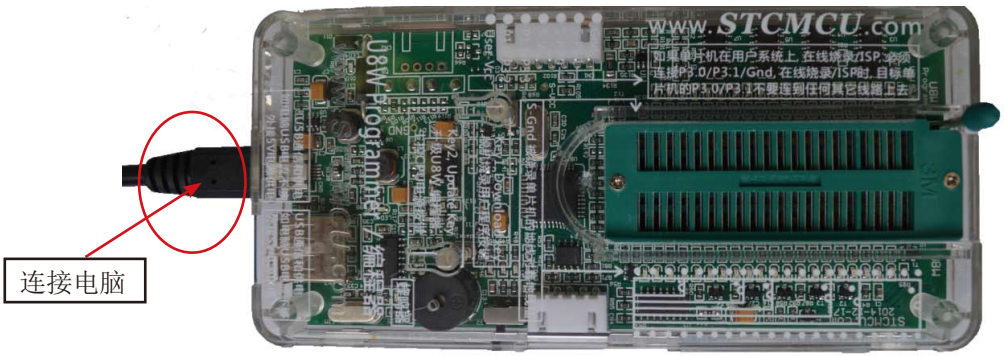
建议用户用最新版本的STC-ISP下载软件“STC-ISP (V6.85K).exe”（请随时留意STC官方网站http://www.STCMCU.com中STC-ISP下载软件的更新，强烈建议用户在官方网站http://www.STCMCU.com中下载最新版本的软件使用）。

10.2.4 U8W的脱机下载使用说明

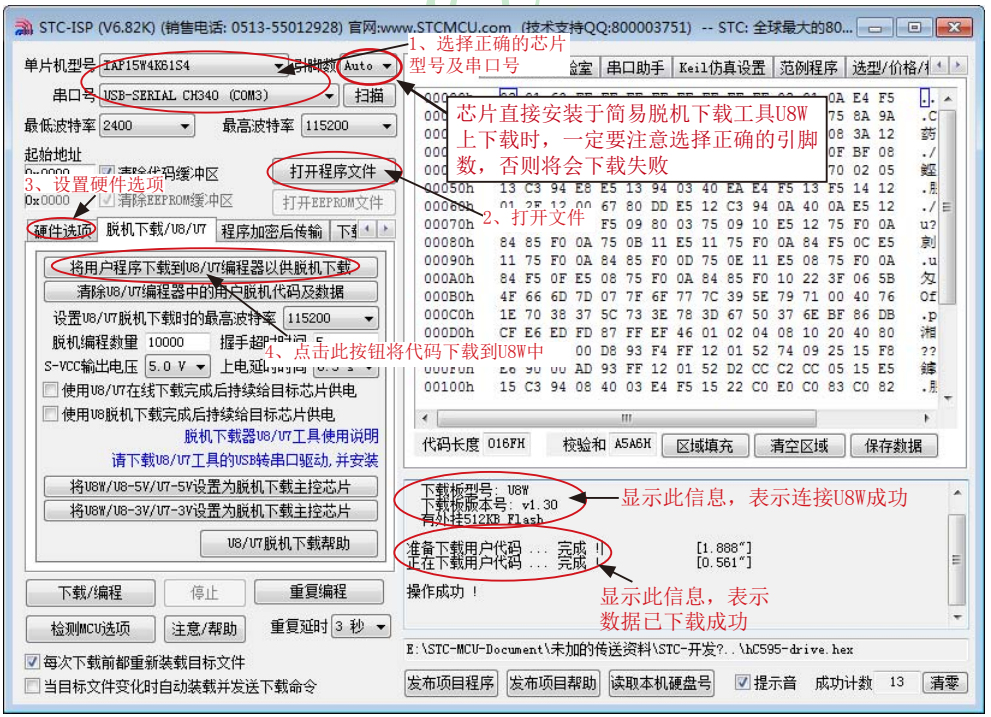
10.2.4.1 目标芯片直接安装于U8W座锁紧上并通过USB连接电脑给U8W供电进行脱机下载

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：



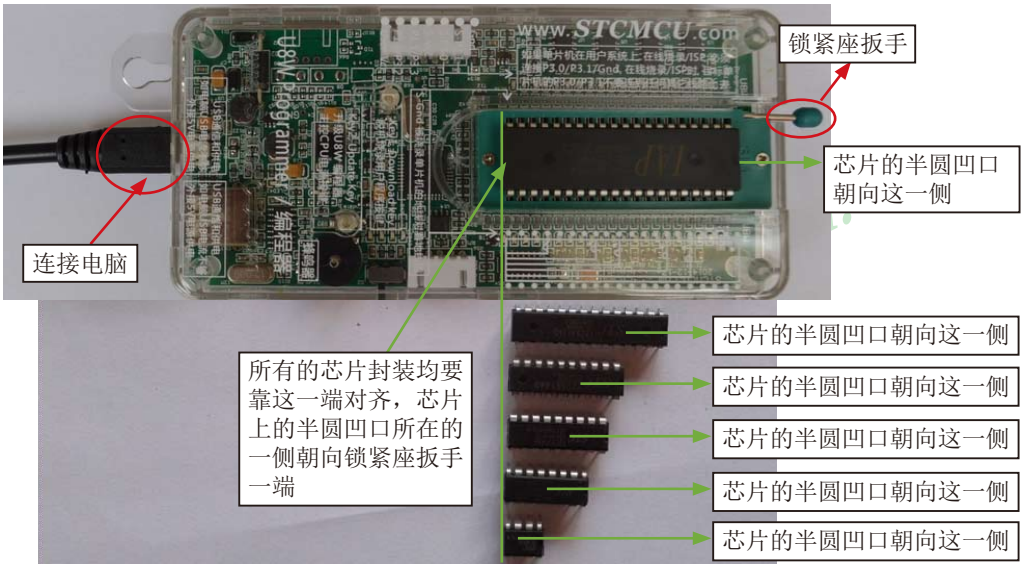
（2）在ISP下载软件“STC-ISP (V6.85K).exe”以上版本中按如下图所示的步骤进行设置：



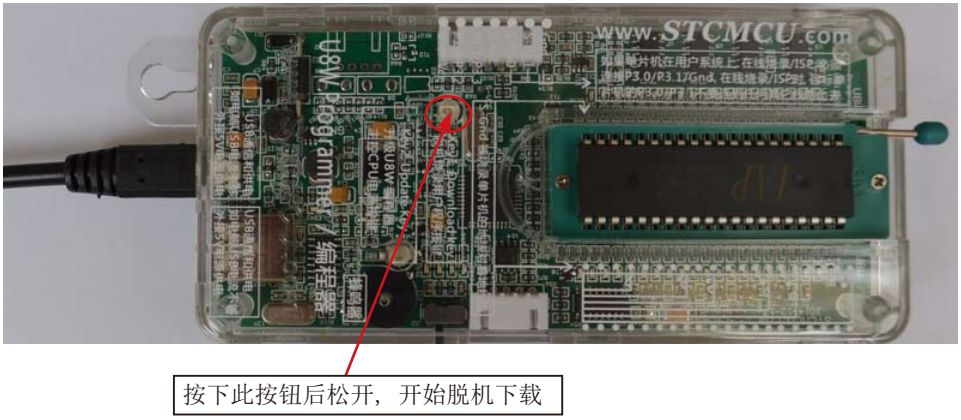
按照上图的步骤，操作完成后，若下载成功则表示用户代码和相关的设置选项都已下载到U8W下载工具中。

建议用户用最新版本的STC-ISP下载软件“STC-ISP (V6.85K).exe”（请随时留意STC官方网站http://www.STCMCU.com中STC-ISP下载软件的更新，强烈建议用户在官方网站http://www.STCMCU.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。

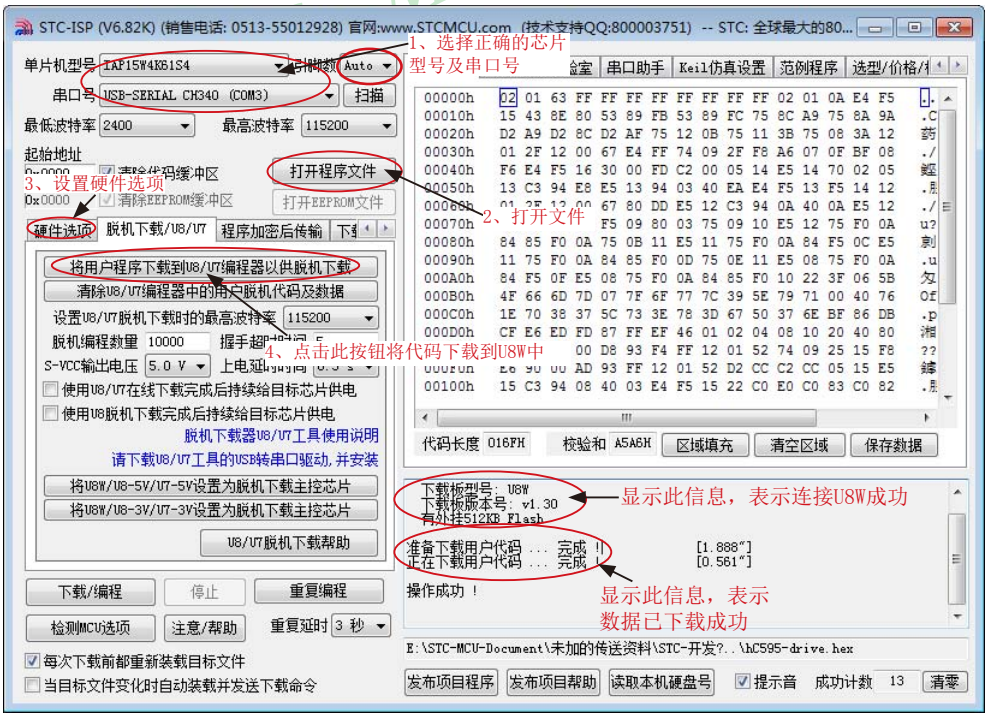
10.2.4.2 目标芯片由用户系统引线连接U8W并通过USB连接电脑给U8W供电进行脱机下载

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：

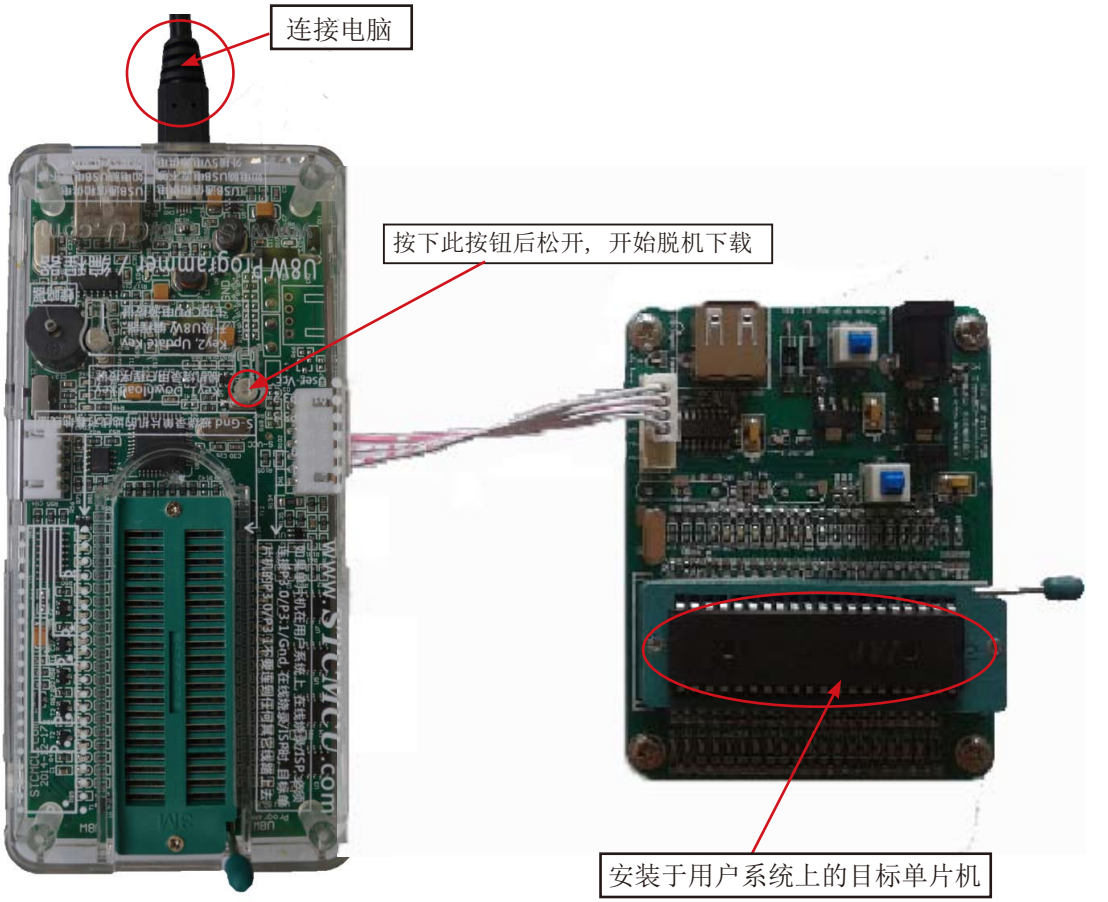
（2）在ISP下载软件“STC-ISP (V6.85K).exe”以上版本中按如下图所示的步骤进行设置：

建议用户用最新版本的STC-ISP下载软件“STC-ISP (V6.85K).exe”（请随时留意STC官方网站http://www.STCMCU.com中STC-ISP下载软件的更新，强烈建议用户在官方网站http://www.STCMCU.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。

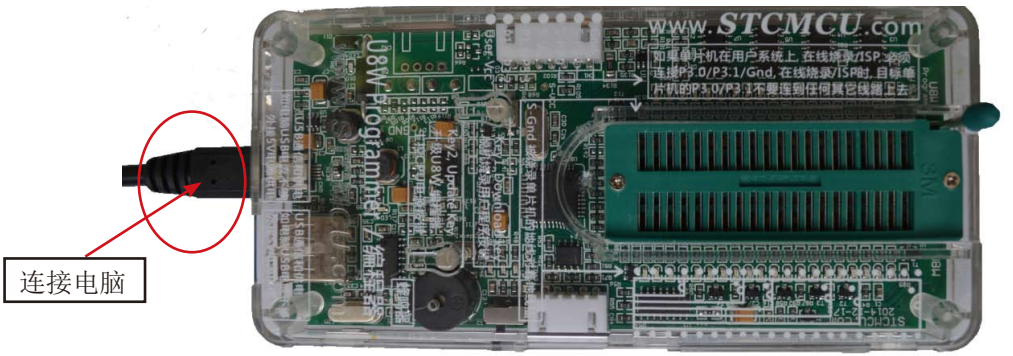
（3）然后使用连接线连接电脑、将U8W下载工具以及用户系统（目标单片机）如下图所示的方式连接起来，并按下图所示的按钮后松开，即可开始脱机下载



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。

10.2.4.3 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：

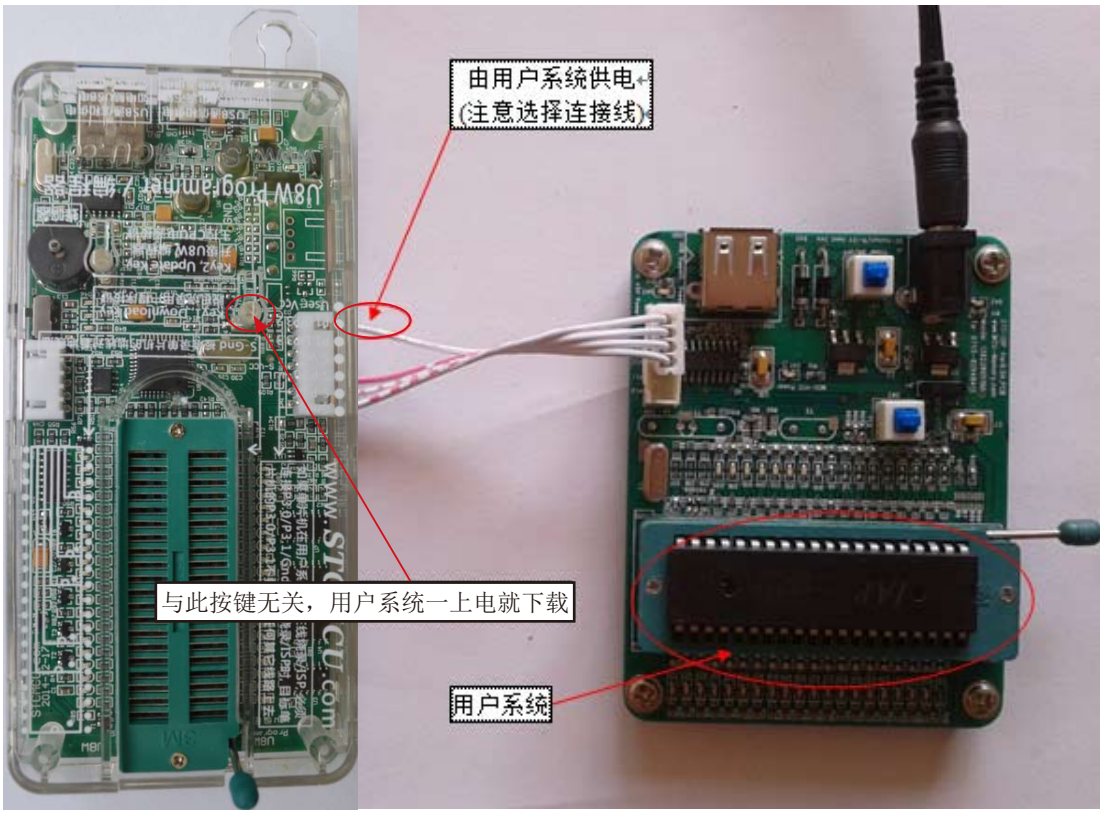


（2）在ISP下载软件“STC-ISP (V6.85K).exe”以上版本中按如下图所示的步骤进行设置：

建议用户用最新版本的STC-ISP下载软件“STC-ISP (V6.85K).exe”（请随时留意STC官方网站http://www.STCMCU.com中STC-ISP下载软件的更新，强烈建议用户在官方网站http://www.STCMCU.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：



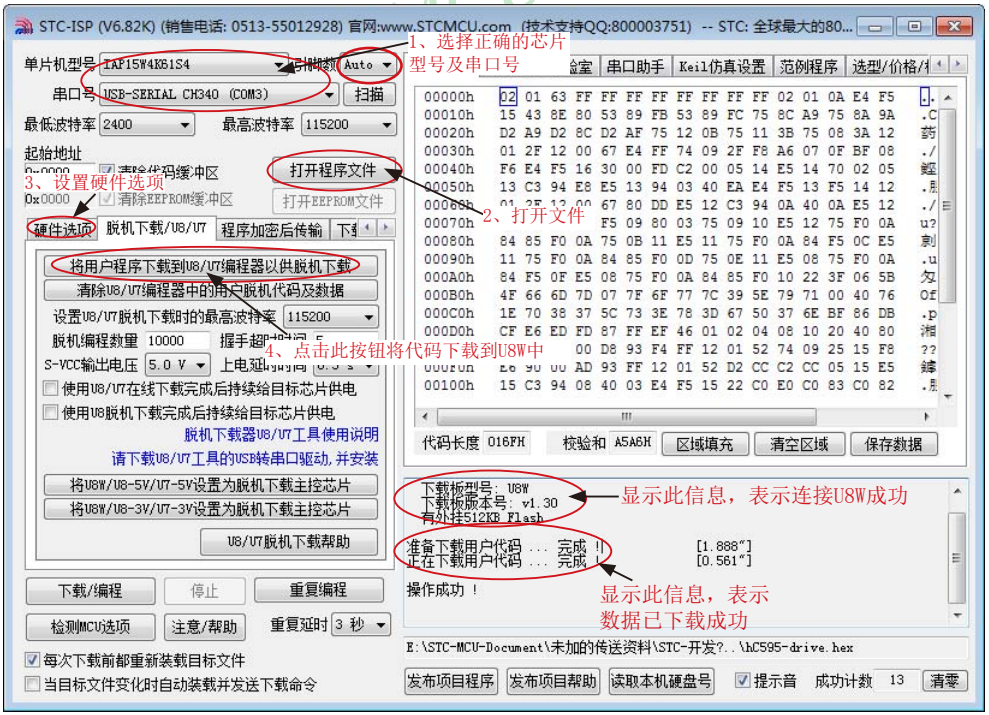
下载的过程中，U8W下载工具上的4个LED会以跑马灯的模式显示。下载完成后，若下载成功，则4个LED会同时亮、同时灭；若下载失败，则4个LED全部不亮。

10.2.4.4 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：

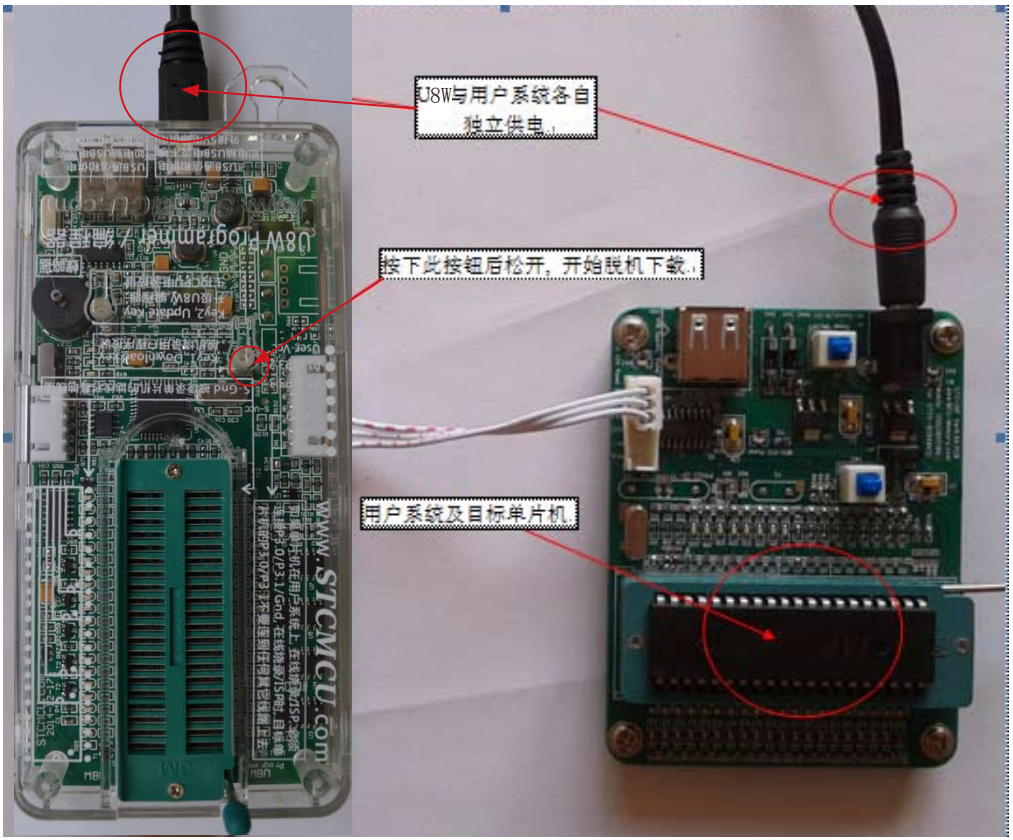
（2）在ISP下载软件“STC-ISP (V6.85K).exe”以上版本中按如下图所示的步骤进行设置：

建议用户用最新版本的STC-ISP下载软件“STC-ISP (V6.85K).exe”（请随时留意STC官方网站http://www.STCMCU.com中STC-ISP下载软件的更新，强烈建议用户在官方网站http://www.STCMCU.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

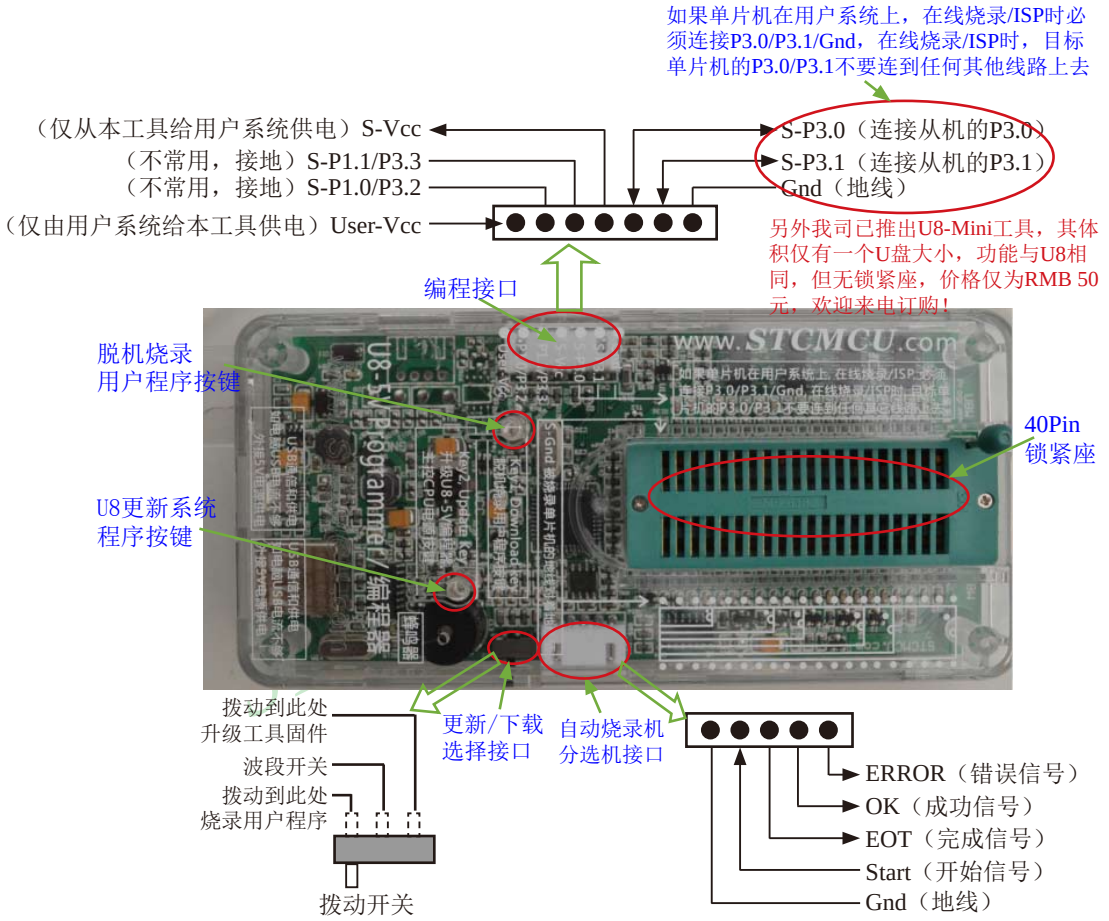
（3）然后按下图所示的方式连接U8W与用户系统，并将图中所示按钮先按下后松开，准备开始脱机下载，最后给用户系统上电/开电源，下载用户程序正式开始：



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。

10.2.5 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）。

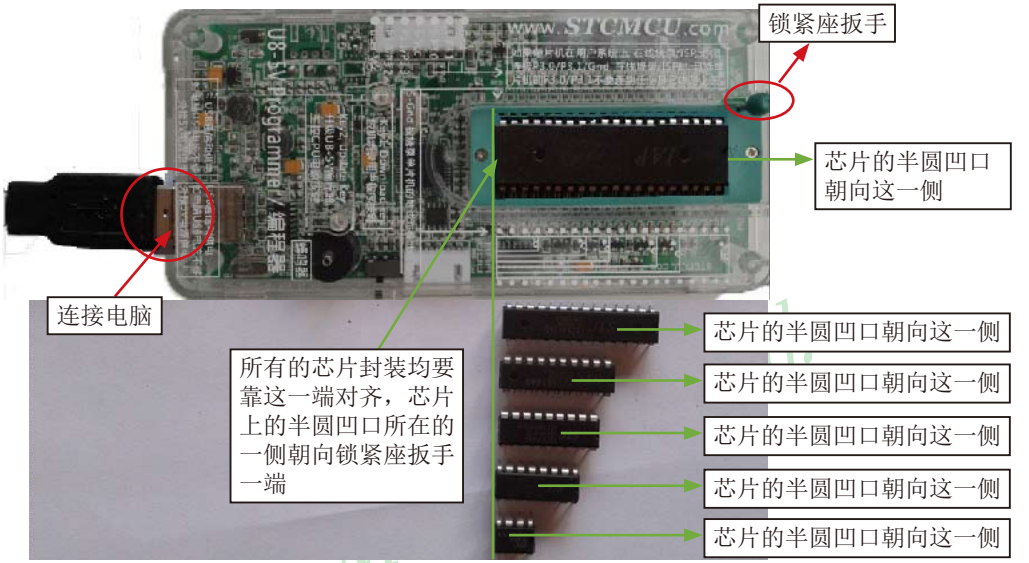
更新/下载选择接口：当需要对U8的底层固件进行升级时，需将此拨动开关拨动到升级工具固件处，当需通过U8对目标芯片进行烧录程序，则需将拨动开关拨动到烧录用户程序处。（拨动开关连接方式请参考上图）

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

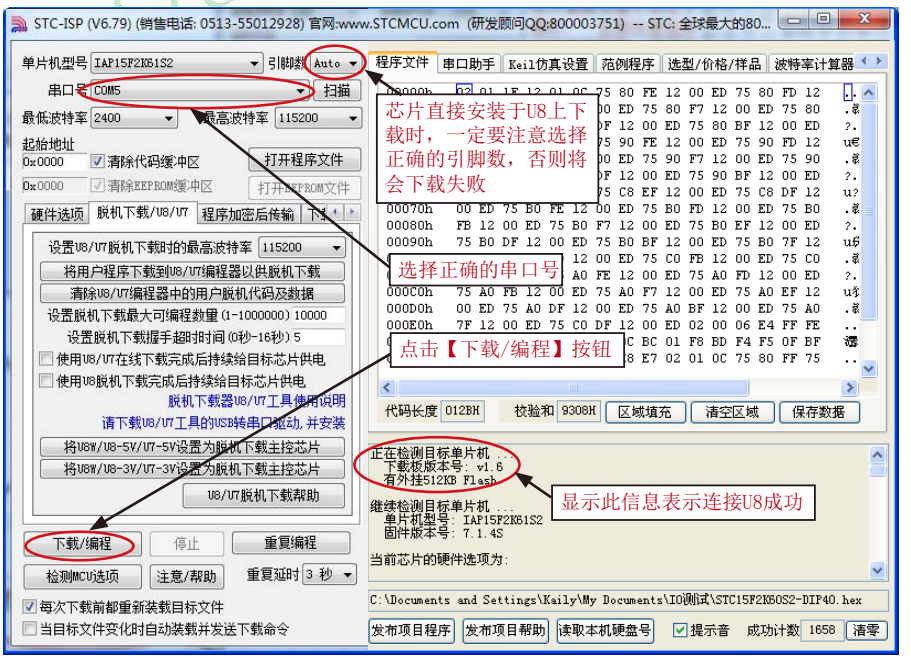
10.2.6 U8的在线联机下载使用说明

10.2.6.1 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STC-ISP下载软件下载程序时，在STC-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。



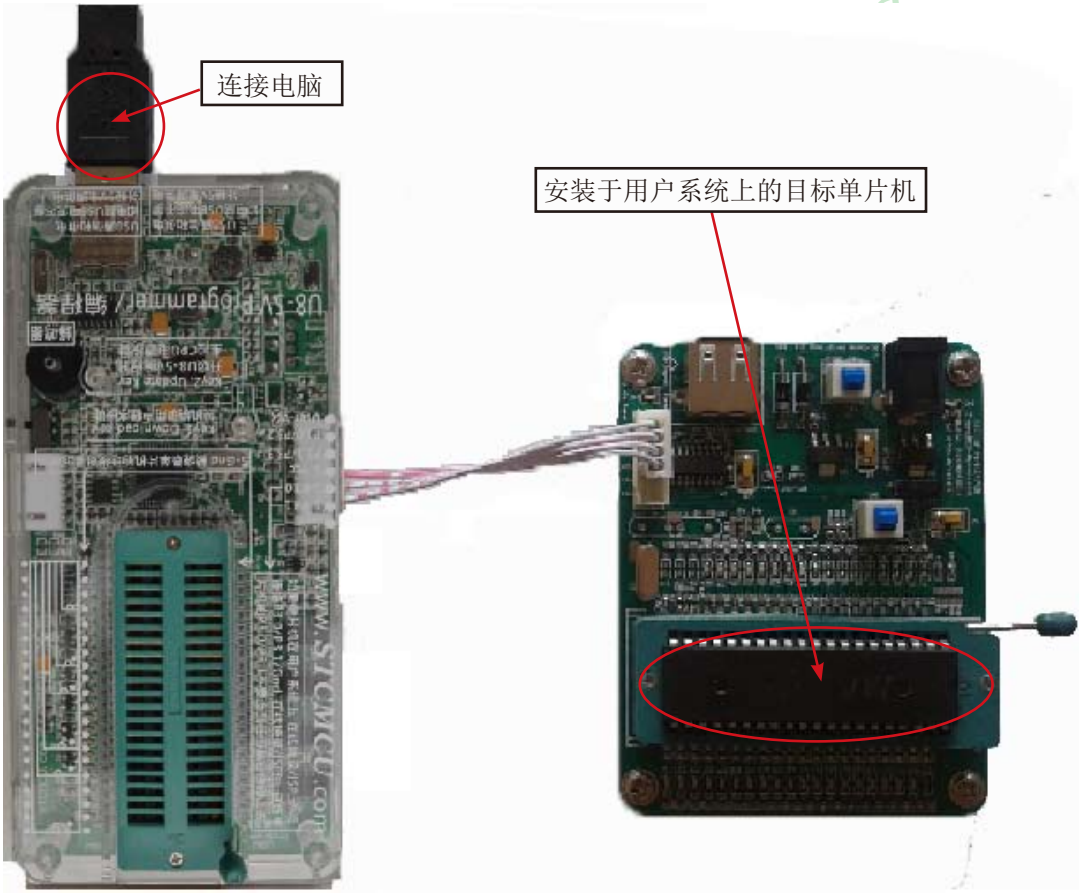
当信息框中有输出下载板的版本号信息以及外挂Flash的相应信息时，表示已正确检测到U8下载工具。

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。

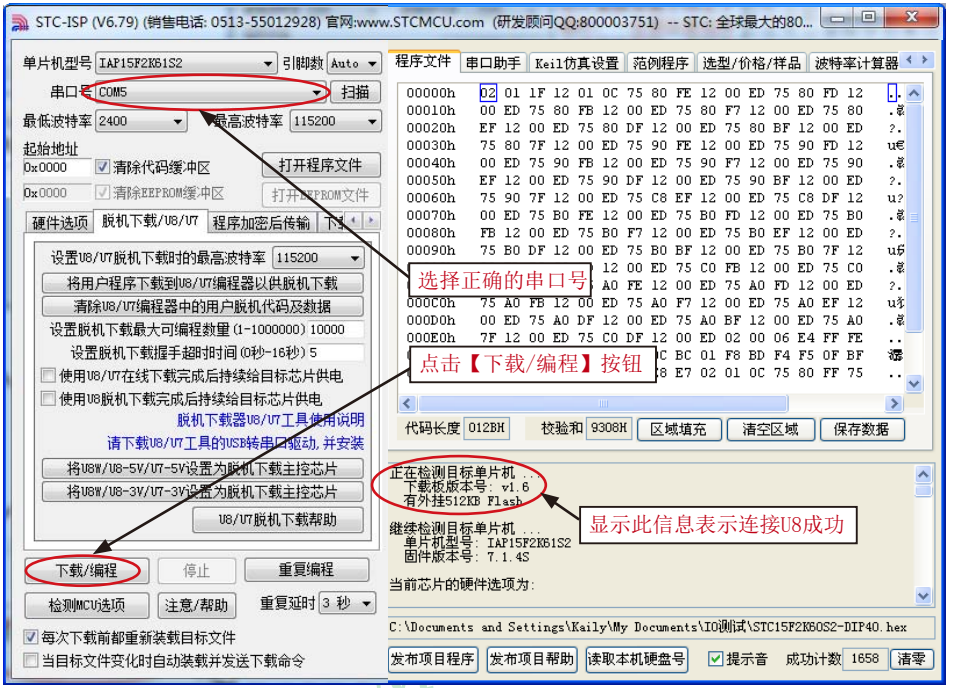
建议用户用最新版本的STC-ISP下载软件“STC-ISP (V6.85).exe”（请随时留意STC官方网站http://www.STCMCU.com中STC-ISP下载软件的更新，强烈建议用户在官方网站http://www.STCMCU.com中下载最新版本的软件使用）。

10.2.6.2 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STC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STC-ISP下载软件下载程序时，在STC-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。

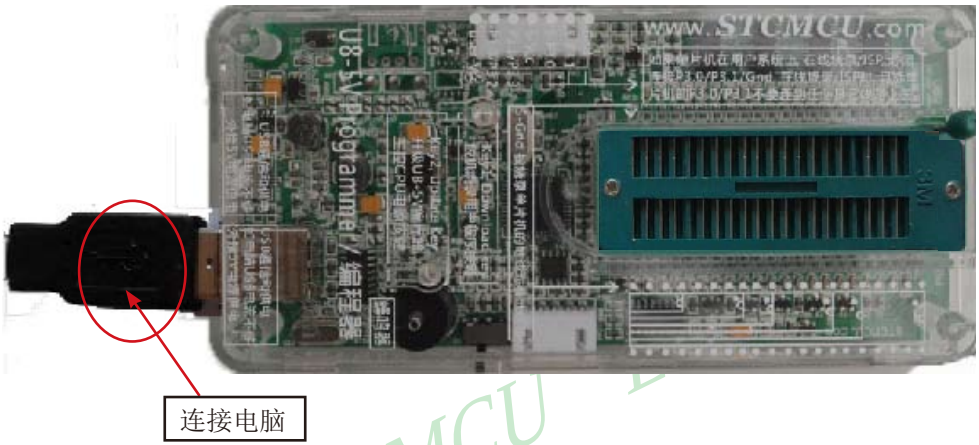
建议用户用最新版本的STC-ISP下载软件“STC-ISP (V6.85).exe”（请随时留意STC官方网站http://www.STCMCU.com中STC-ISP下载软件的更新，强烈建议用户在官方网站http://www.STCMCU.com中下载最新版本的软件使用）。

10.2.7 U8的脱机下载使用说明

10.2.7.1 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：

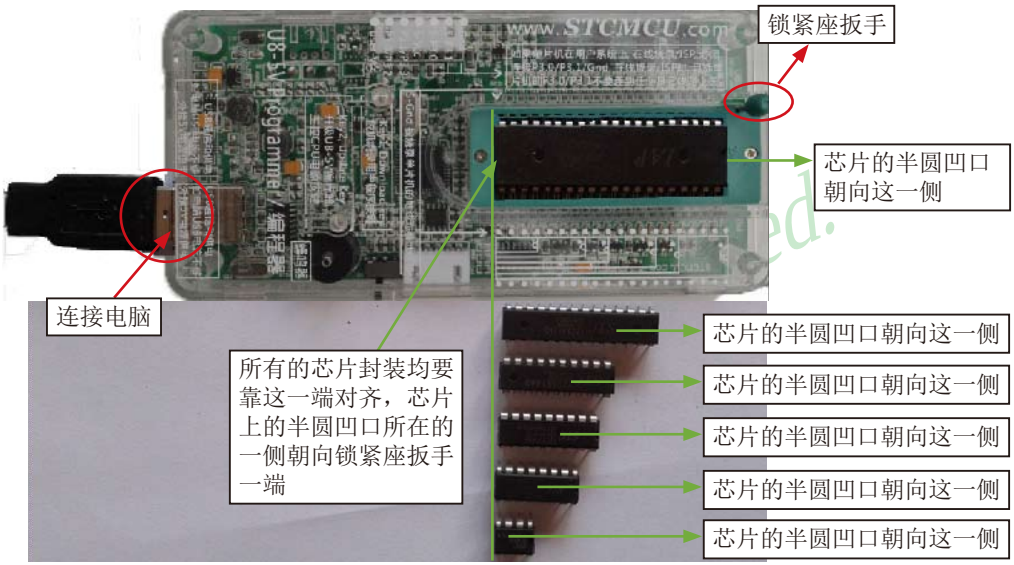


（2）在ISP下载软件“STC-ISP (V6.85).exe”以上版本中按如下图所示的步骤进行设置：

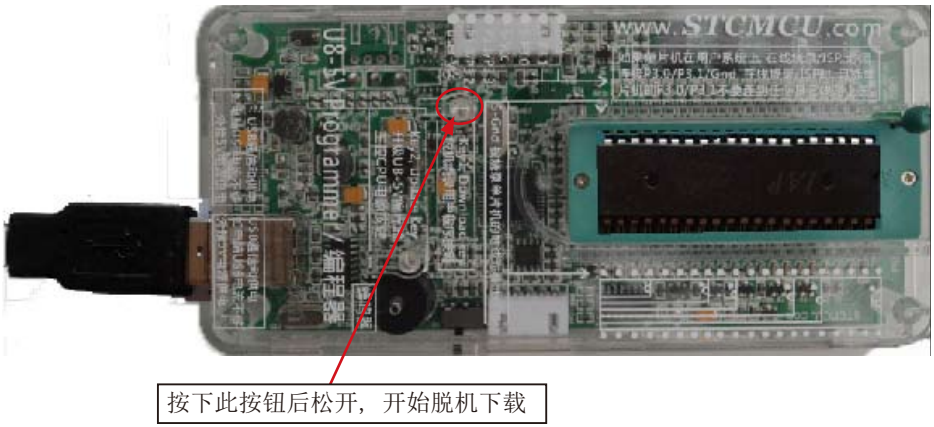
按照上图的步骤，操作完成后，若下载成功则表示用户代码和相关的设置选项都已下载到U8下载工具中。

建议用户用最新版本的STC-ISP下载软件“STC-ISP (V6.85).exe”（请随时留意STC官方网站http://www.STCMCU.com中STC-ISP下载软件的更新，强烈建议用户在官方网站http://www.STCMCU.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。

10.2.7.2 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：

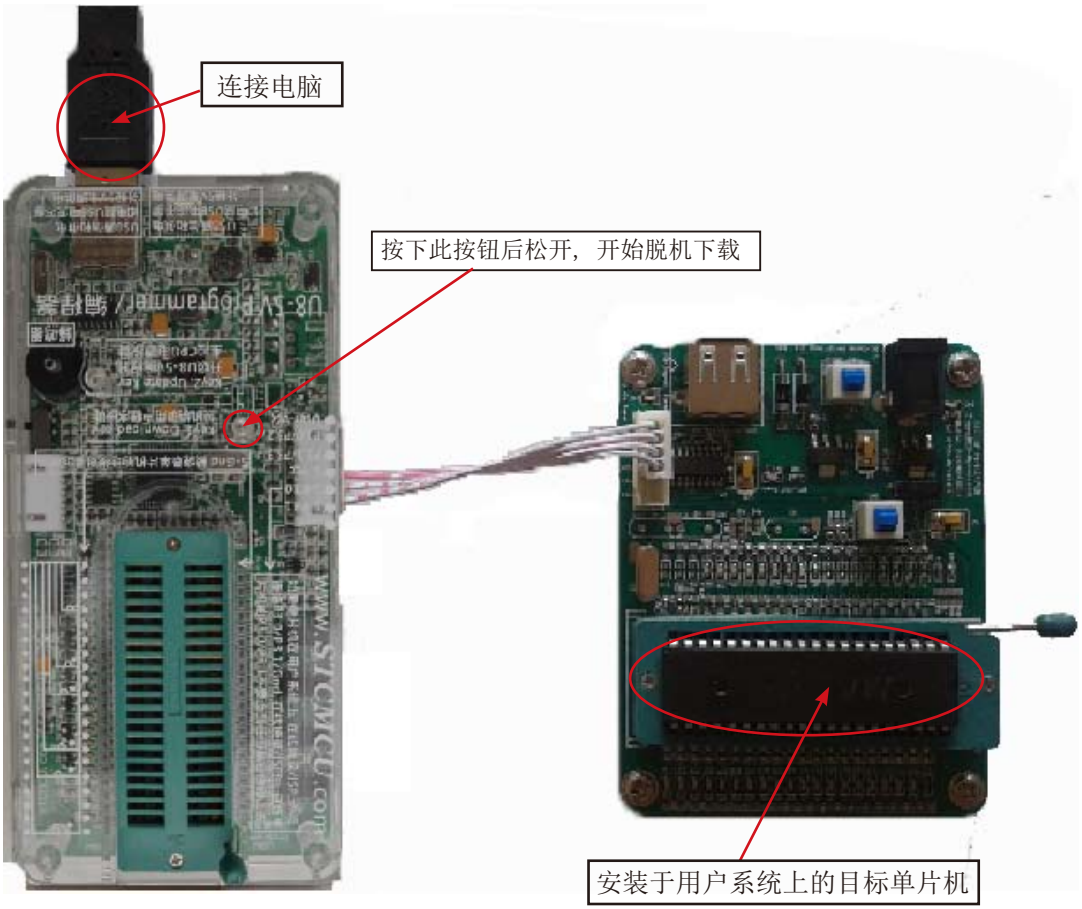
（1）使用STC提供的USB连接线将U8下载板连接到电脑，如下图：

（2）在ISP下载软件“STC-ISP (V6.85).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STC-ISP下载软件“STC-ISP (V6.85).exe”（请随时留意STC官方网站http://www.STCMCU.com中STC-ISP下载软件的更新，强烈建议用户在官方网站http://www.STCMCU.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



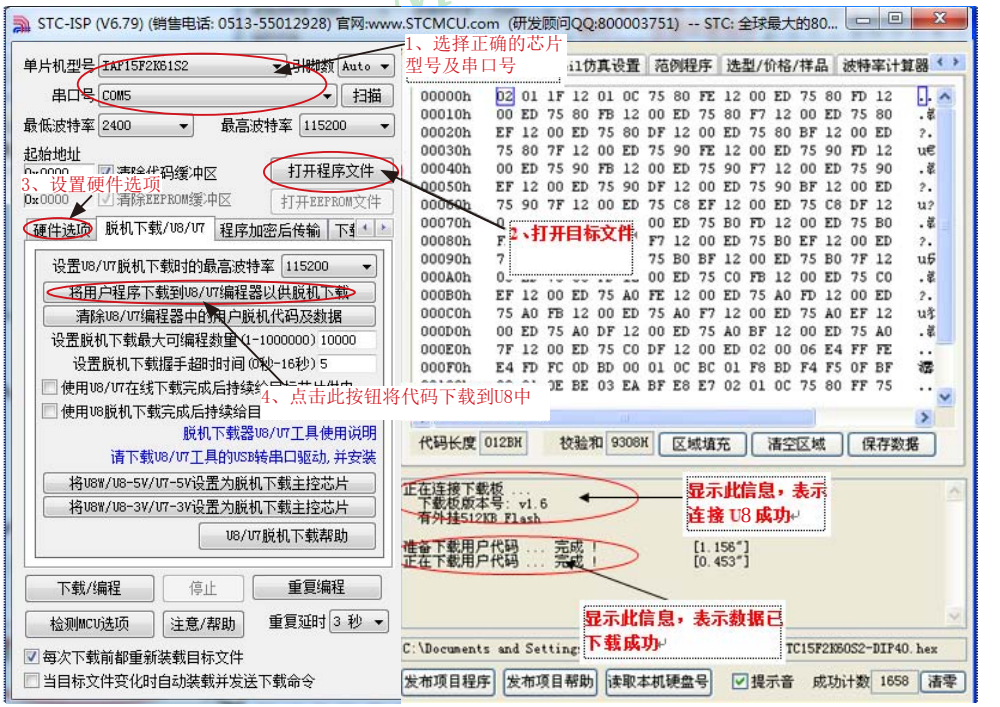
下载的过程中，U8下载工具上的4个LED会以跑马灯的模式显示。下载完成后，若下载成功，则4个LED会同时亮、同时灭；若下载失败，则4个LED全部不亮。

10.2.7.3 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：

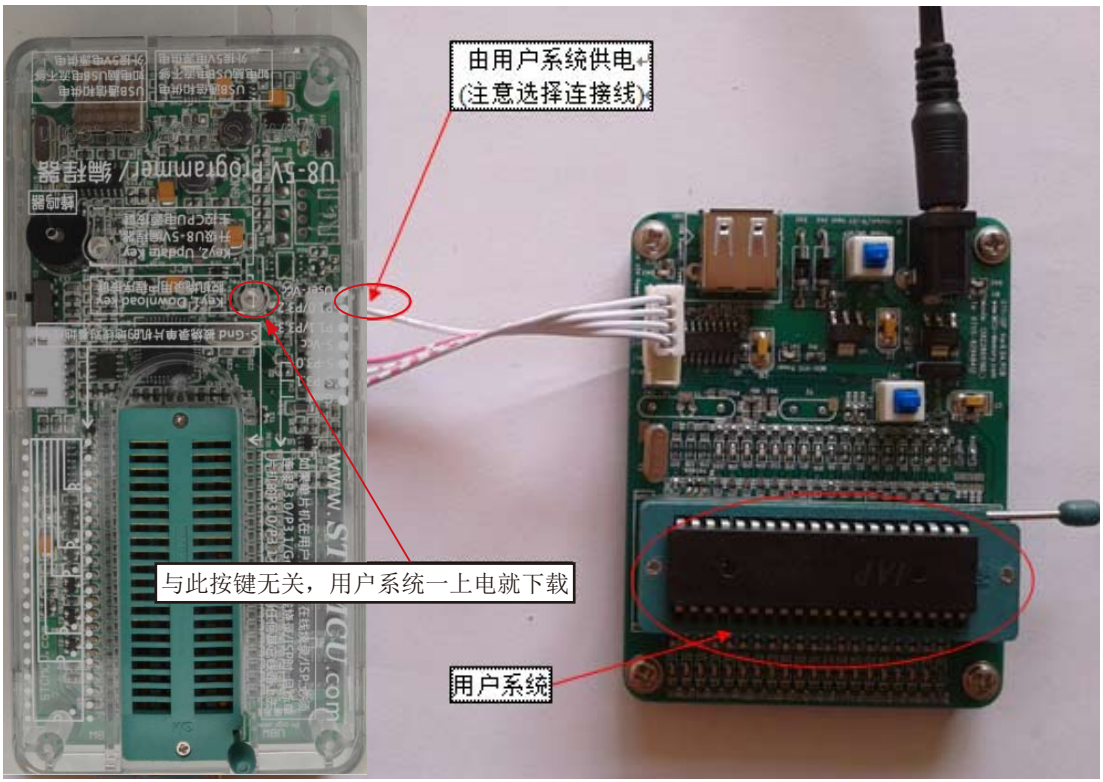
（2）在ISP下载软件“STC-ISP (V6.85).exe”以上版本中按如下图所示的步骤进行设置：

建议用户用最新版本的STC-ISP下载软件“STC-ISP (V6.85).exe”（请随时留意STC官方网站http://www.STCMCU.com中STC-ISP下载软件的更新，强烈建议用户在官方网站http://www.STCMCU.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：



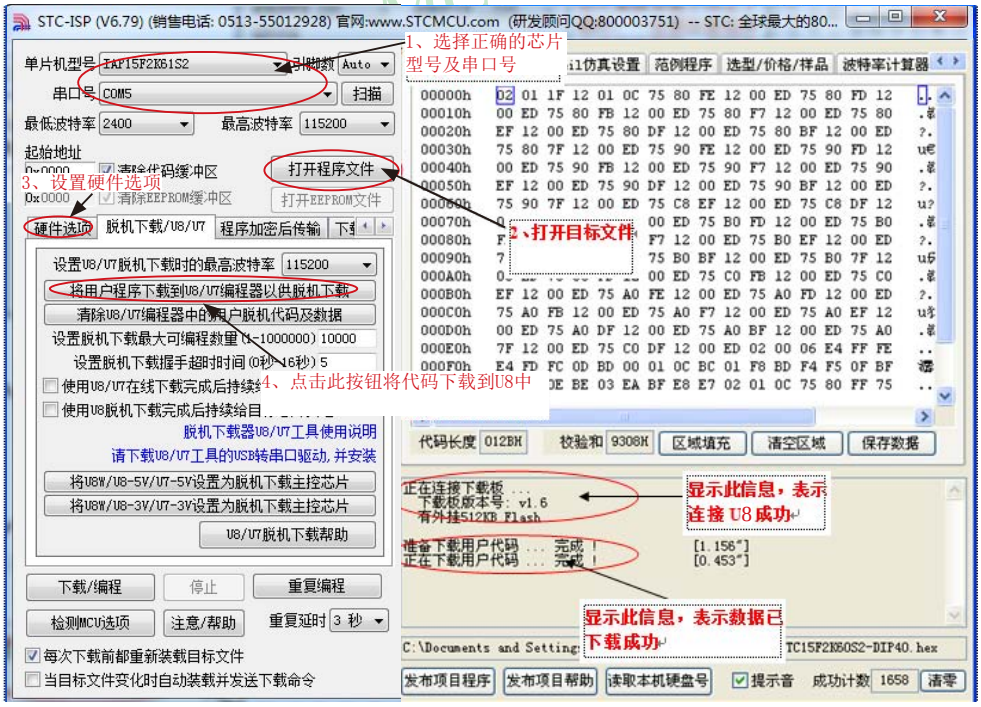
下载的过程中，U8下载工具上的4个LED会以跑马灯的模式显示。下载完成后，若下载成功，则4个LED会同时亮、同时灭；若下载失败，则4个LED全部不亮。

10.2.7.4 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：

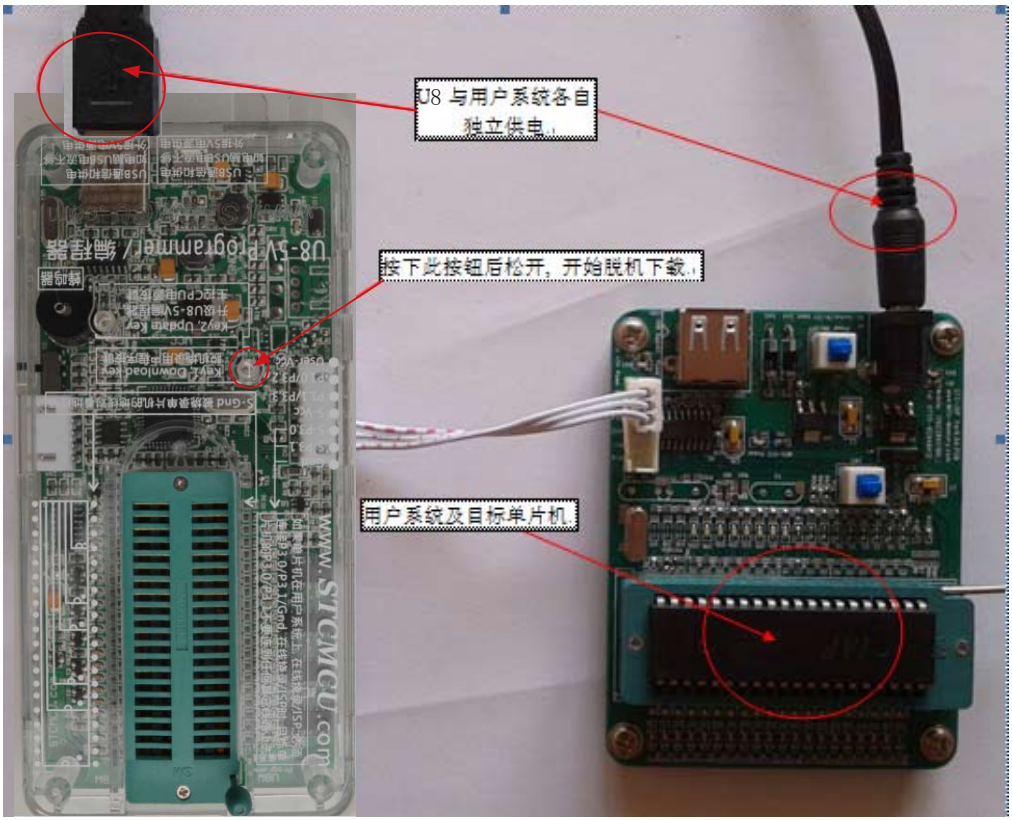
（2）在ISP下载软件“STC-ISP (V6.85).exe”以上版本中按如下图所示的步骤进行设置：

建议用户用最新版本的STC-ISP下载软件“STC-ISP (V6.85).exe”（请随时留意STC官方网站http://www.STCMCU.com中STC-ISP下载软件的更新，强烈建议用户在官方网站http://www.STCMCU.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：



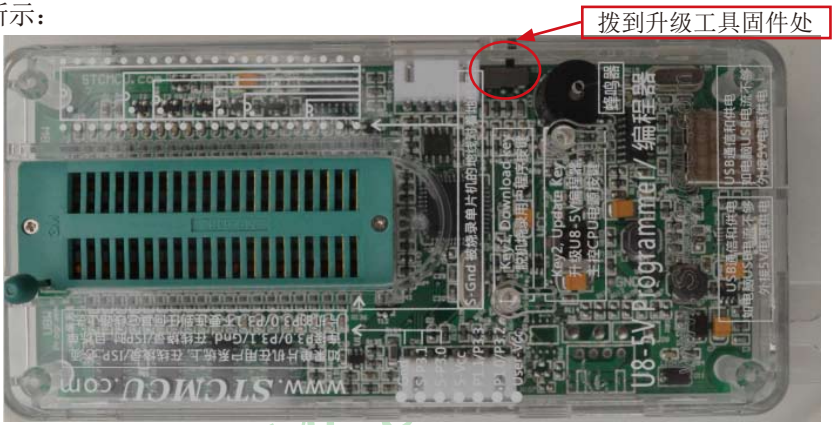
下载的过程中，U8下载工具上的4个LED会以跑马灯的模式显示。下载完成后，若下载成功，则4个LED会同时亮、同时灭；若下载失败，则4个LED全部不亮。

10.2.8 制作/更新USB型联机/脱机下载工具U8W/U8W-Mini/U8/U8-Mini

10.2.8.1 制作U8W/U8W-Mini/U8/U8-Mini下载母片(控制母片)

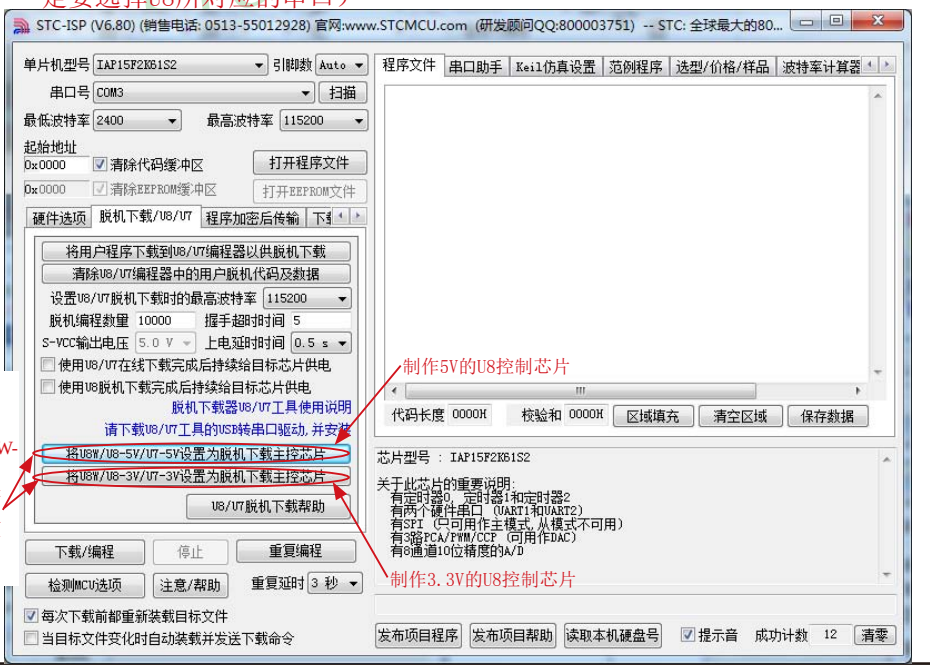
制作U8W/U8W-Mini及U8/U8-Mini下载母片的过程类似，为节约篇幅，下文以U8为例，详述如何制作U8下载母片，U8W/U8W-Mini及U8-Mini不作赘述。

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：

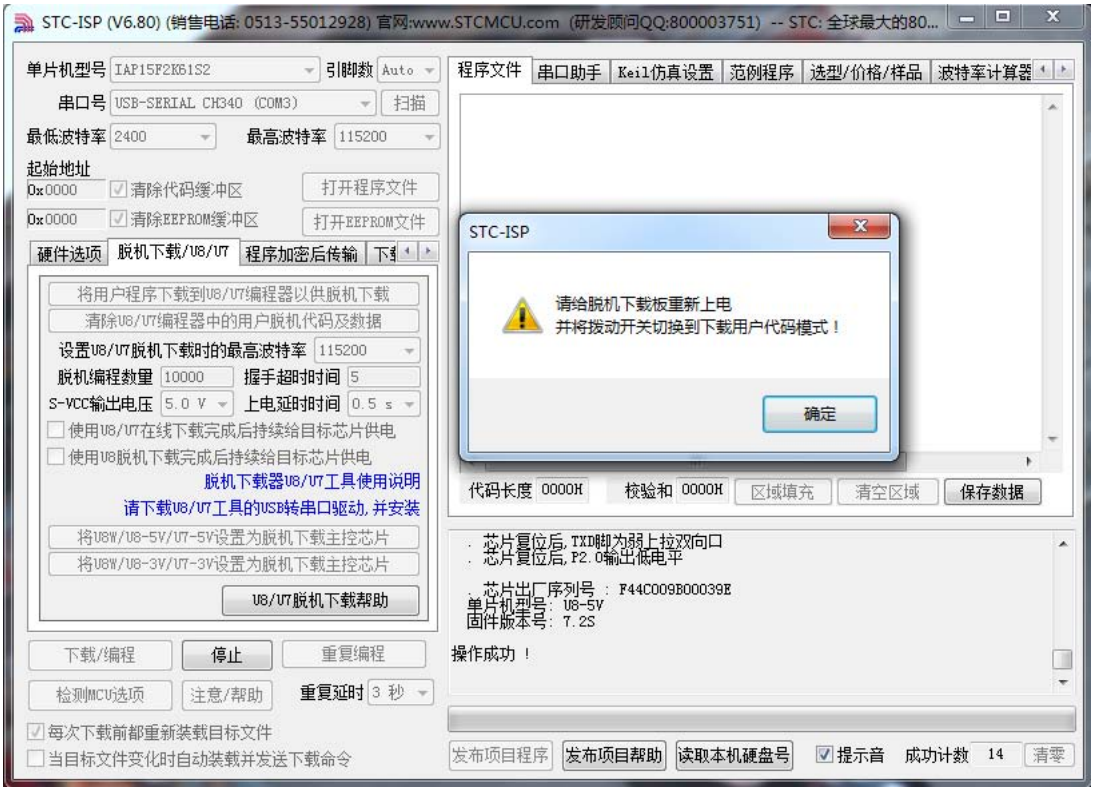


然后在ISP下载程序“STC-ISP (V6.80).exe”中的“脱机下载/U8/U7”页面中点击“将U8W/U8-5V/U7-5V设置为脱机下载主控芯片”按钮（5V下载板）或者点击“将U8W/U8-3V/U7-3V设置为脱机下载主控芯片”按钮（3.3V下载板），如下图：

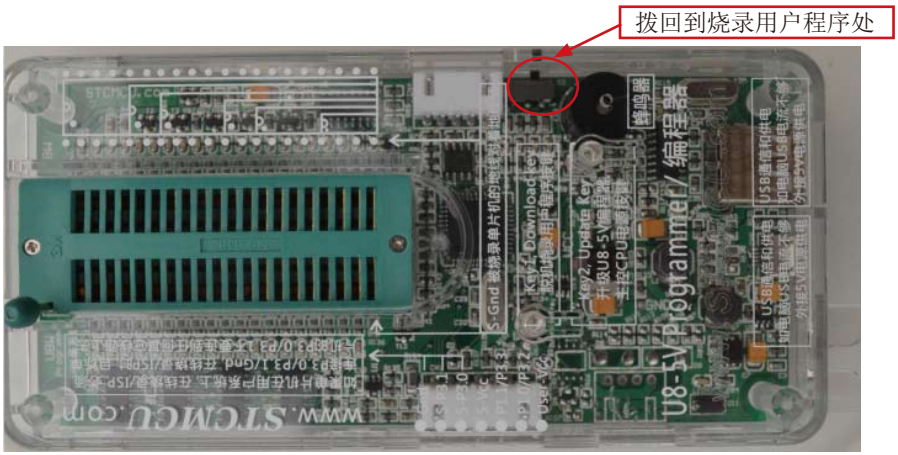
（注意：一定要选择U8所对应的串口）



在出现如下画面表示U8控制芯片制作完成：



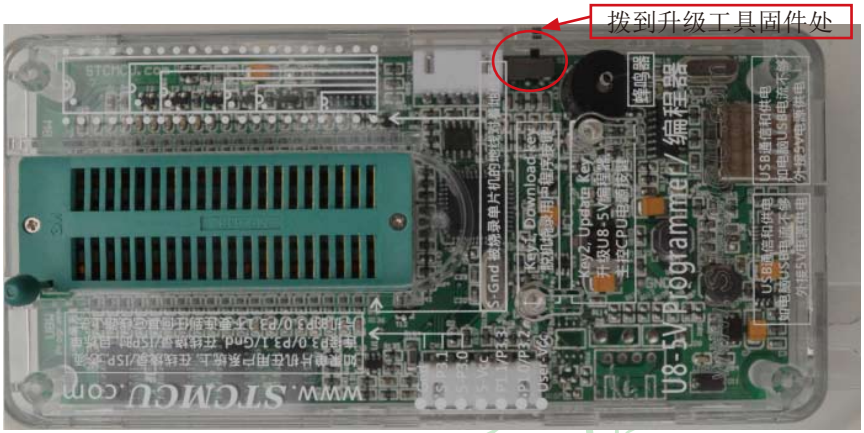
制作完成后，一定不要忘记将U8的“更新/下载选择接口”拨回到“烧录用户程序”模式，并将U8下载工具重新上电，如下图所示：（否则将不能正常进行下载）



10.2.8.2 手动升级U8W/U8W-Mini/U8/U8-Mini

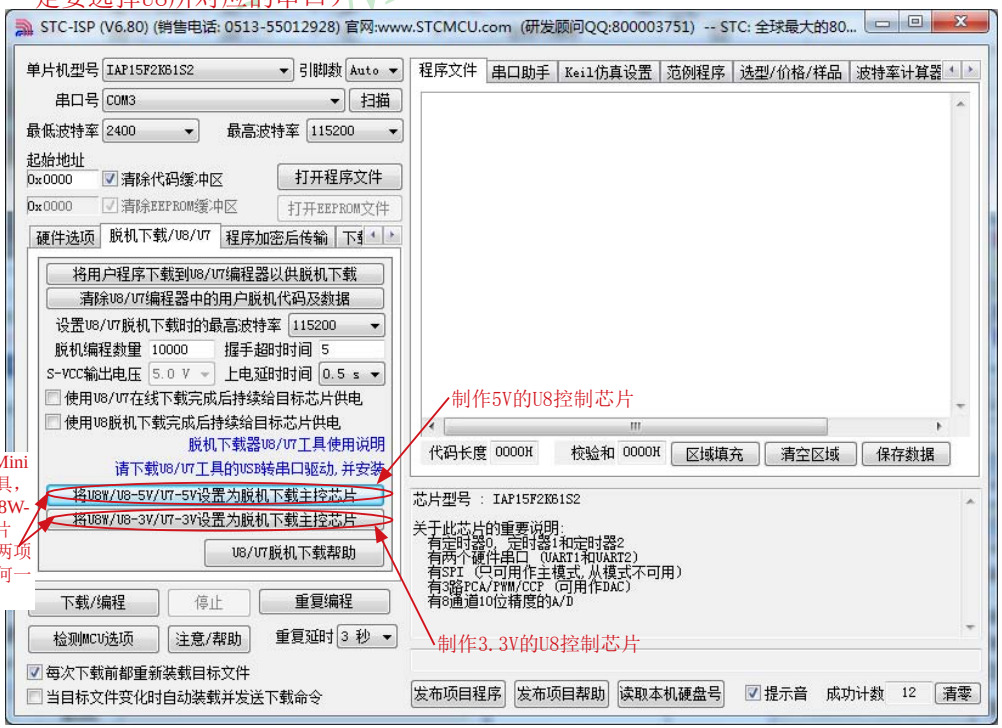
手动升级U8W/U8W-Mini及U8/U8-Mini的过程类似，为节约篇幅，下文以U8为例，详述如何手动升级U8，U8W/U8W-Mini及U8-Mini不作赘述。

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：

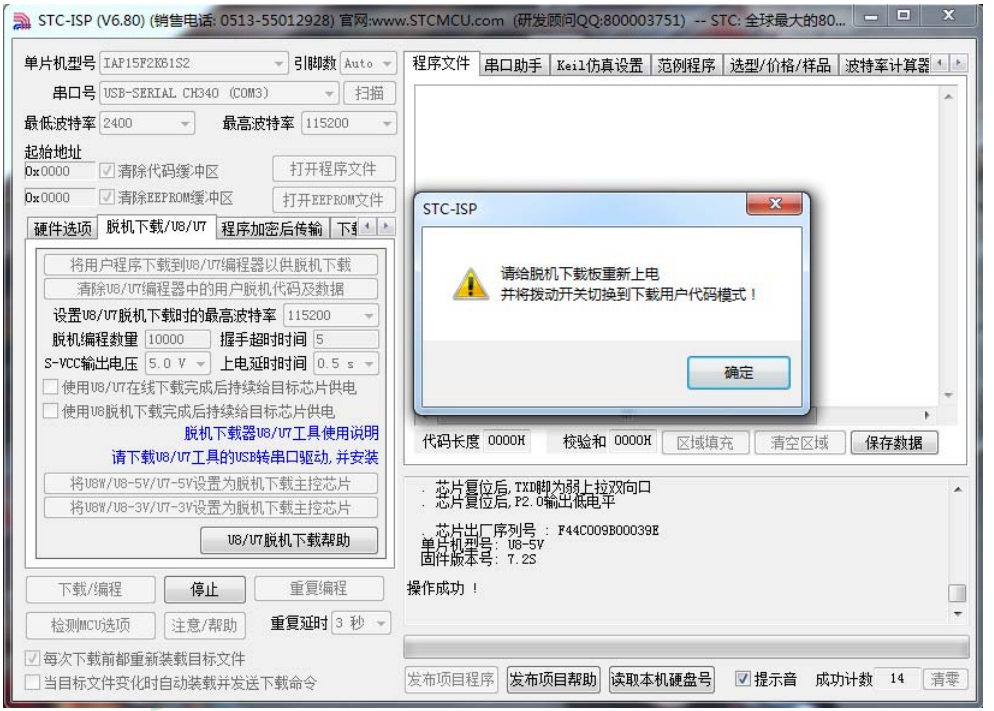


然后在ISP下载程序“STC-ISP (V6.80).exe”中的“脱机下载/U8/U7”页面中点击“将U8W/U8-5V/U7-5V设置为脱机下载主控芯片”按钮（5V下载板）或者点击“将U8W/U8-3V/U7-3V设置为脱机下载主控芯片”按钮（3.3V下载板），如下图：

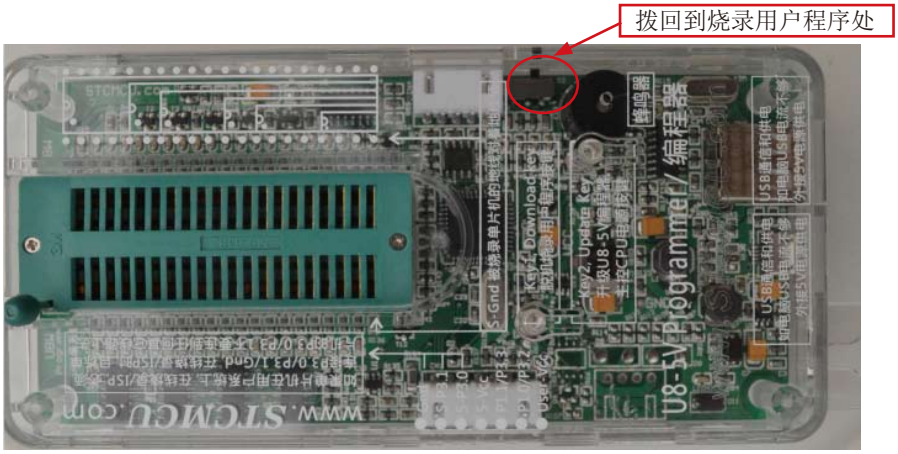
（注意：一定要选择U8所对应的串口）



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）

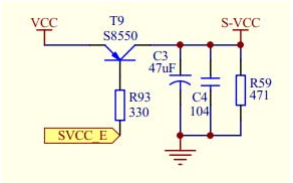


10.2.9 USB型联机/脱机下载板U8W/U8W-Mini/U8/U8-Mini的参考电路

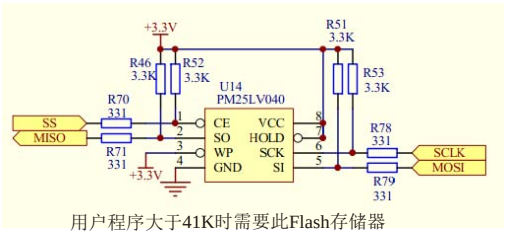
USB型联机/脱机下载板 U8W/U8W-Mini/U8/U8-Mini 为用户提供了如下的常用控制接口（Ver6.85版）：

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

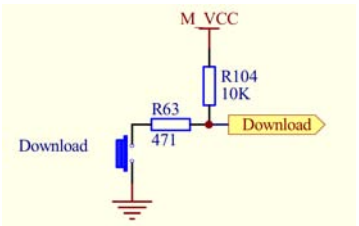


Flash控制部分参考电路图

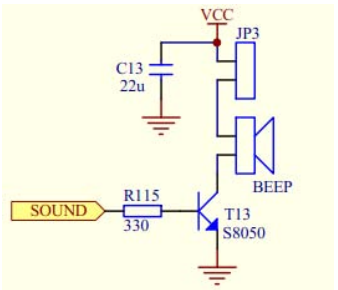


用户程序大于41K时需要此Flash存储器

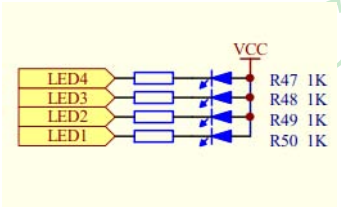
按键部分参考电路图



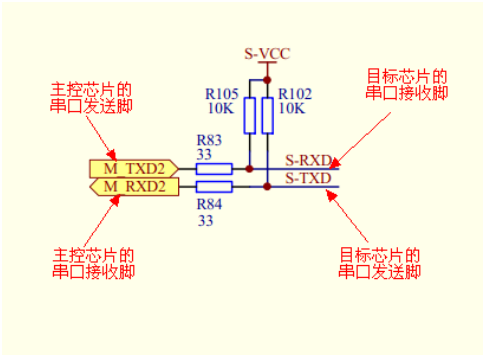
蜂鸣器部分参考电路图



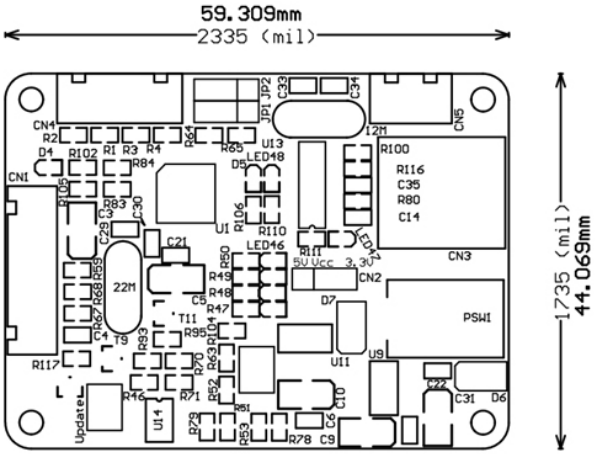
LED显示部分参考电路图



串口通讯脚连接部分参考电路图

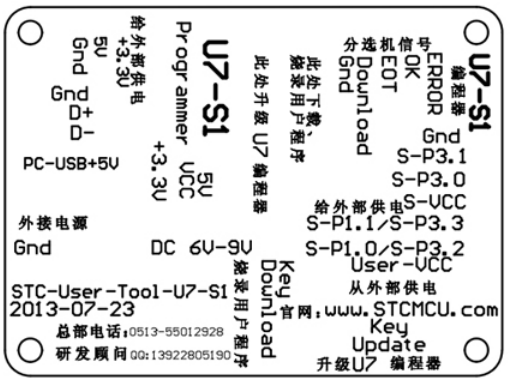


U8 PCB板正面丝印图：

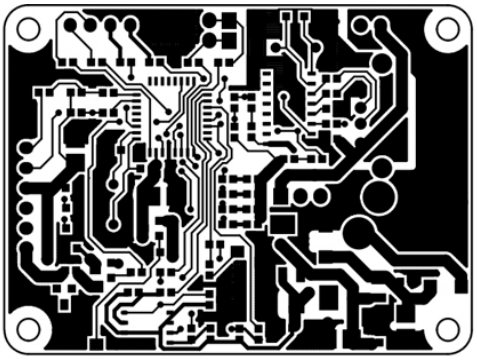


四角的安装孔直径：2.8mm

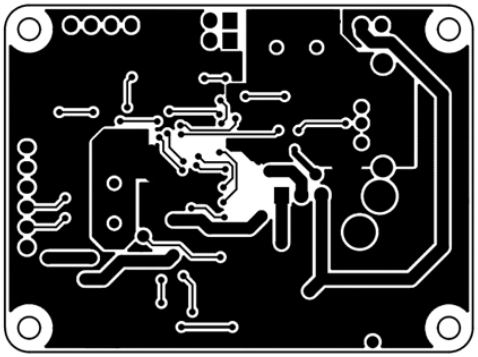
U8 PCB板反面丝印图：



U8 PCB板走线图（正面）：



U8 PCB板走线图（反面）：



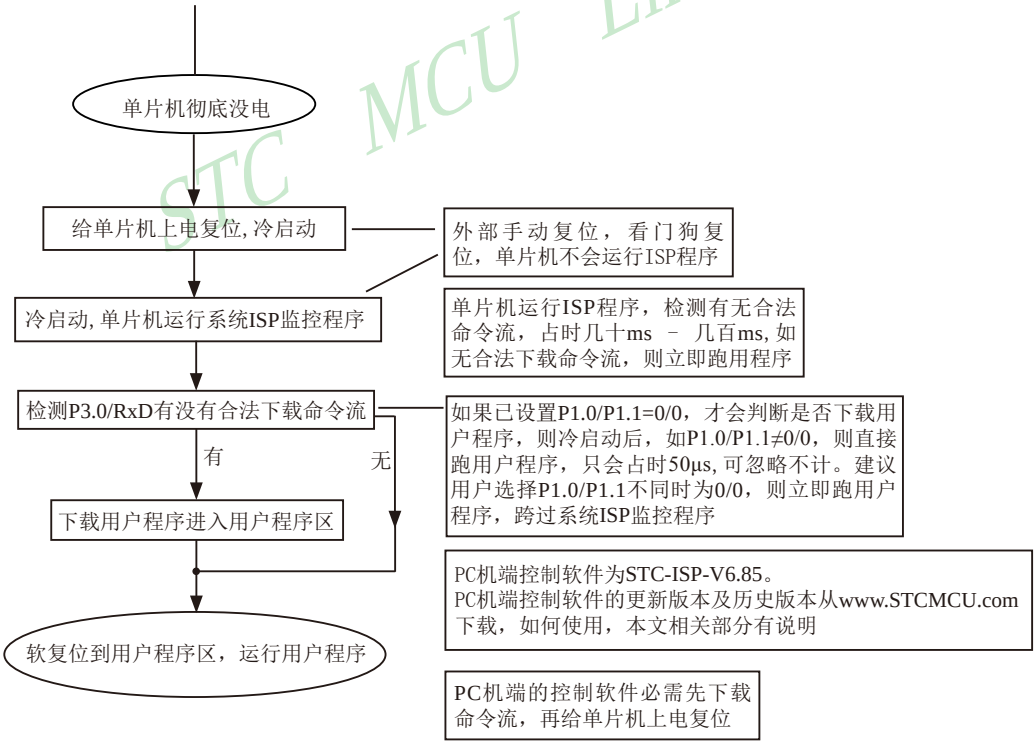
10.3 ISP编程器/烧录器的说明

我们有：STC-ISP经济型下载编程工具

所有STC-ISP编程工具的分类如下：

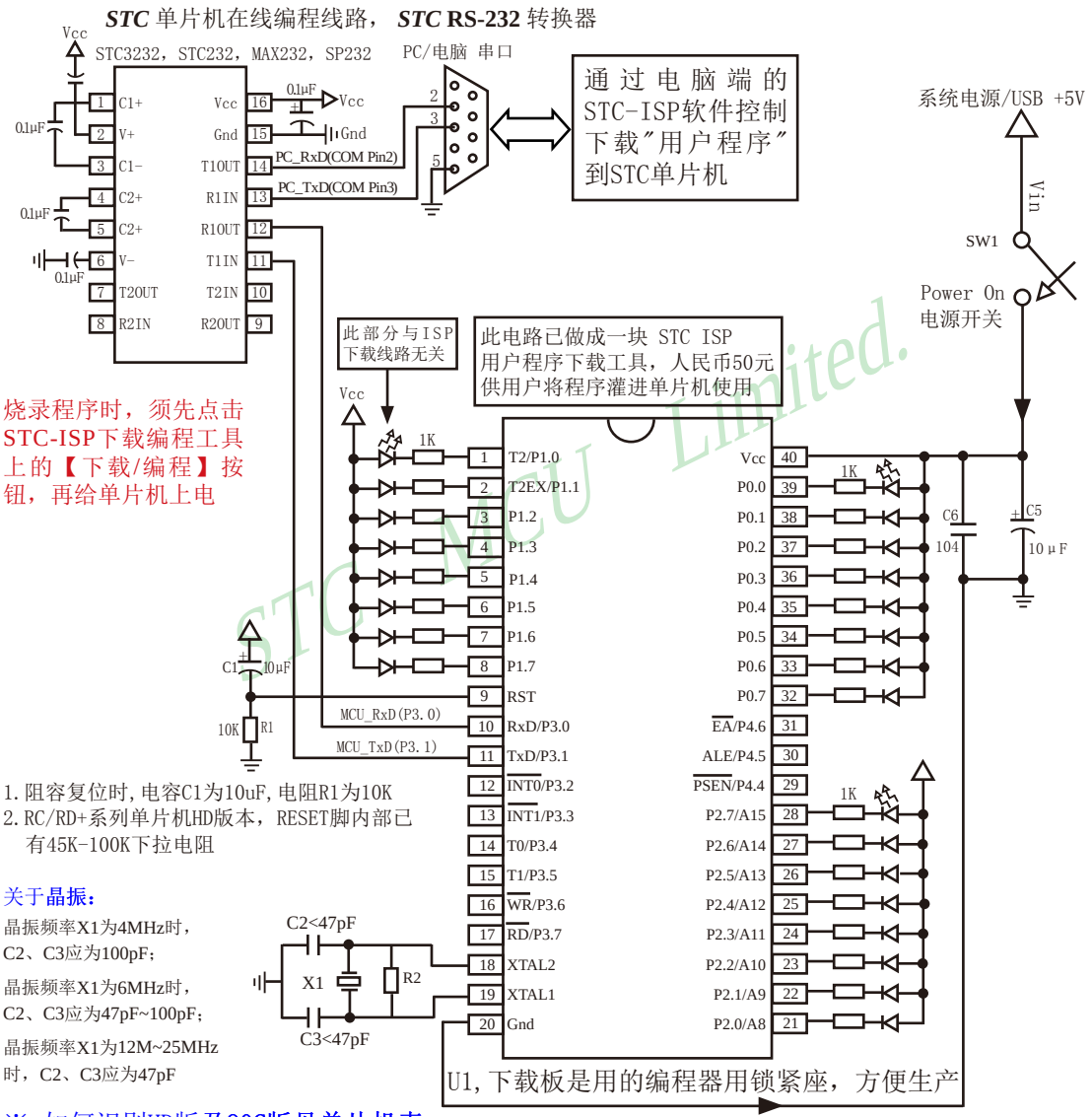
- STC-ISP经济型下载编程工具
  - STC12/11/10/89/90系列专用ISP下载编程工具
    - STC12/11/10/89/90系列40-pin单片机专用ISP下载编程工具
    - STC12/11/10/89/90系列32-pin单片机专用ISP下载编程工具
    - STC12/11/10/89/90系列28-pin单片机专用ISP下载编程工具
    - STC12/11/10/89/90系列20-pin单片机专用ISP下载编程工具
    - STC12/11/10/89/90系列18-pin单片机专用ISP下载编程工具
    - STC12/11/10/89/90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

10.3.1 在系统可编程(ISP)原理使用说明



10.3.2 STC89xx系列在系统可编程(ISP)典型应用线路图

10.3.2.1 利用RS-232转换器的典型应用线路图



STC89C51RC/RD+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。

STC89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（STC-ISP.exe 软件）：

(1). 获得STC提供的ISP下载工具（软件）

登陆 www.STCMCU.com 网站，从STC半导体专栏下载PC（电脑）端的ISP程序，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STC-ISP下载工具（软件）,请随时更新，目前已到Ver6.85版本以上,

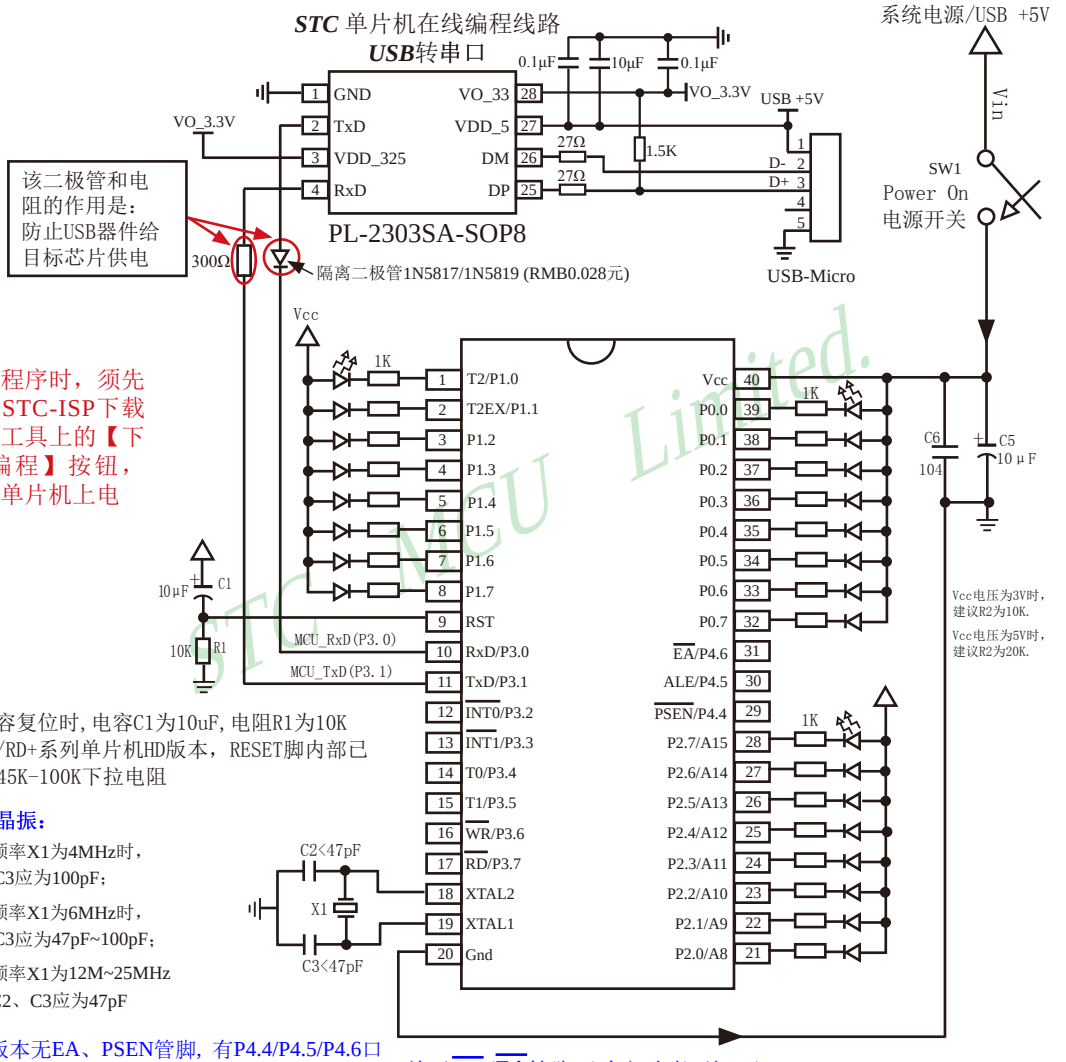
支持*.bin,*.hex(Intel 16 进制格式)文件，少数*.hex 文件不支持的话，请转换成*.bin 文件，**请随时注意升级PC（电脑）端的STC-ISP.EXE** 程序。

(3).已经固化有ISP 引导码,并设置为上电复位进入ISP 的STC89C51RC/RD+ 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 程序。

(4).可能用户板上P3.0/RxD，P3.1/Txd 除了接RS-232 转换器外，还接了RS-485 等电路，需要将其断开。用户系统接了RS-485 电路的，推荐在选项中选择下次冷启动时需P1.0/P1.1=0.0才判是否下载程序。

10.3.2.2 利用USB转串口芯片PL-2303SA的ISP下载编程典型应用线路图

建议选用USB转串口芯片PL-2303HXD-SSOP28 / PL-2303HX-SSOP28 / PL-2303SA-SOP8，国产兼容的PL-2303HX-SSOP28价格便宜，PL-2303HXD-SSOP28 / PL-2303SA-SOP8无需外部晶振，电路简单



1. 阻容复位时，电容C1为10uF，电阻R1为10K
2. RC/RD+系列单片机HD版本，RESET脚内部已有45K-100K下拉电阻

关于晶振：

晶振频率X1为4MHz时，C2、C3应为100pF；

晶振频率X1为6MHz时，C2、C3应为47pF~100pF；

晶振频率X1为12M~25MHz时，C2、C3应为47pF

90C版本无EA、PSEN管脚，有P4.4/P4.5/P4.6口

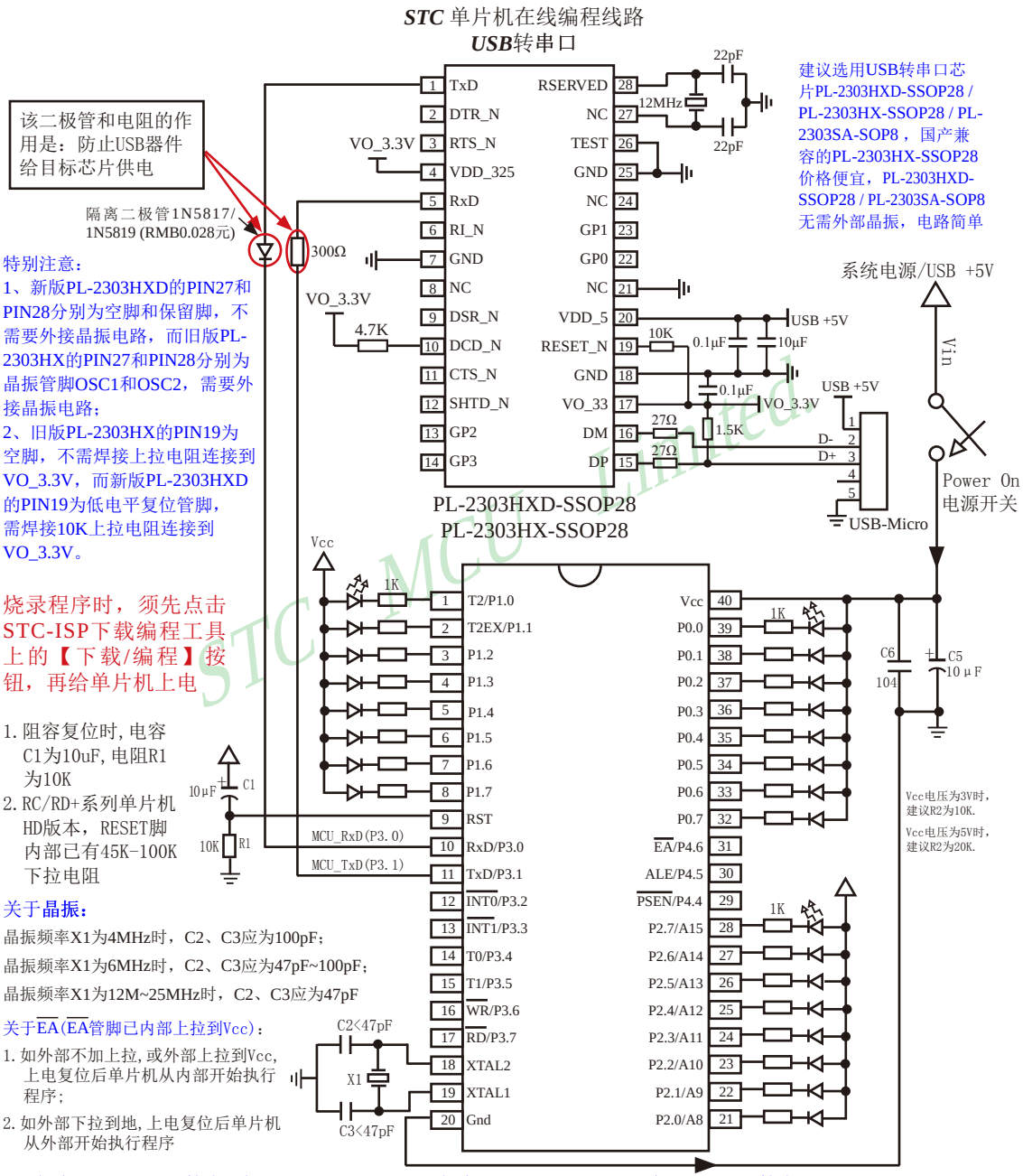
HD版本无P4.6/P4.5/P4.4口，有EA、PSEN管脚

※ STC89xx系列区分90C版本及HD版本，如何识别90C版及HD版：通过查询单片机表面文字最下面一行最后几个字母，最后几个字母为90C，则该单片机为90C版本；最后几个字母为HD，则该单片机为HD版本

关于$\overline{EA}$($\overline{EA}$管脚已内部上拉到Vcc)：

1. 如外部不加上拉，或外部上拉到Vcc，上电复位后单片机从内部开始执行程序；
2. 如外部下拉到地，上电复位后单片机从外部开始执行程序

10.3.2.3 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



建议选用USB转串口芯片PL-2303HXD-SSOP28 / PL-2303HX-SSOP28 / PL-2303SA-SOP8，国产兼容的PL-2303HX-SSOP28价格便宜，PL-2303HXD-SSOP28 / PL-2303SA-SOP8无需外部晶振，电路简单

特别注意：

1、新版PL-2303HXD的PIN27和PIN28分别为空脚和保留脚，不需要外接晶振电路，而旧版PL-2303HX的PIN27和PIN28分别为晶振管脚OSC1和OSC2，需要外接晶振电路；

2、旧版PL-2303HX的PIN19为空脚，不需焊接上拉电阻连接到VO_3.3V，而新版PL-2303HXD的PIN19为低电平复位管脚，需焊接10K上拉电阻连接到VO_3.3V。

烧录程序时，须先点击STC-ISP下载编程工具上的【下载/编程】按钮，再给单片机上电

1. 阻容复位时，电容C1为10uF，电阻R1为10K
2. RC/RD+系列单片机HD版本，RESET脚内部已有45K-100K下拉电阻

关于晶振：

晶振频率X1为4MHz时，C2、C3应为100pF；

晶振频率X1为6MHz时，C2、C3应为47pF~100pF；

晶振频率X1为12M~25MHz时，C2、C3应为47pF

关于$\overline{EA}$($\overline{EA}$管脚已内部上拉到Vcc)：

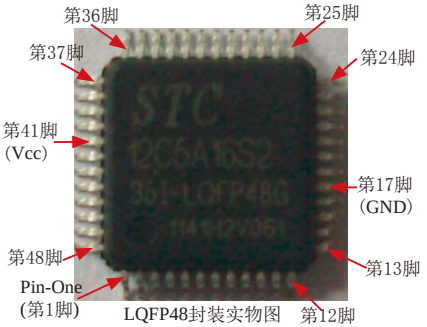
1. 如外部不加上拉，或外部上拉到Vcc，上电复位后单片机从内部开始执行程序；
2. 如外部下拉到地，上电复位后单片机从外部开始执行程序

90C版本无EA、PSEN管脚，有P4.4/P4.5/P4.6口；HD版本无P4.6/P4.5/P4.4口，有EA、PSEN管脚

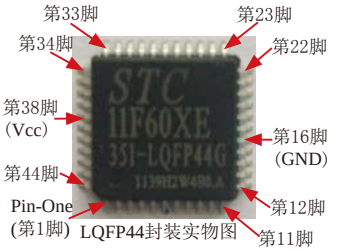
※ STC89xx系列区分90C版本及HD版本，如何识别90C版及HD版：通过查询单片机表面文字最下面一行最后几个字母，最后几个字母为90C，则该单片机为90C版本；最后几个字母为HD，则该单片机为HD版本

10.3.3 所有STC系列单片机封装实物图

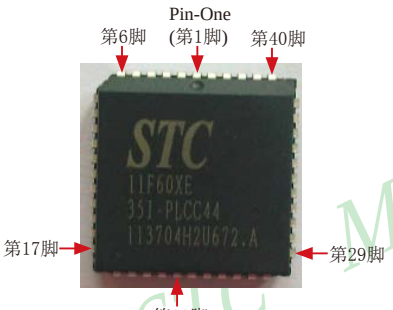
STC12/11/10/89/90系列单片机的封装实物图:



LQFP48封装实物图

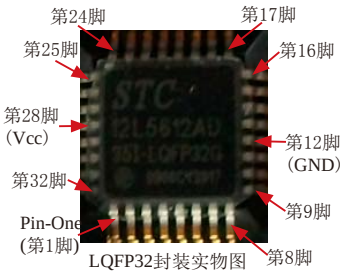


LQFP44封装实物图

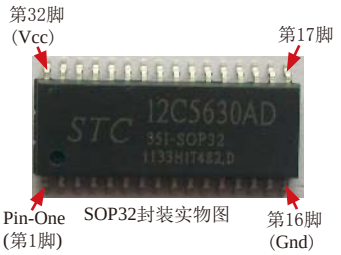


PLCC44封装实物图

40-Pin单片机PDIP40的封装实物图



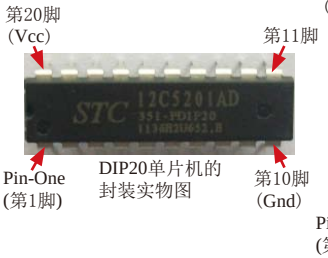
LQFP32封装实物图

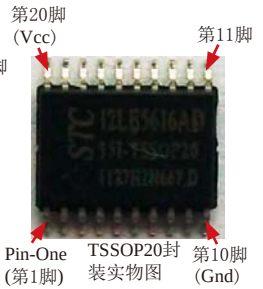


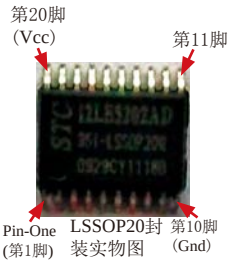
SOP32封装实物图

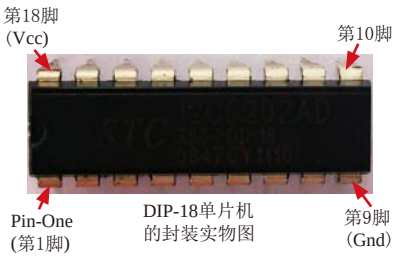
SOP28封装实物图

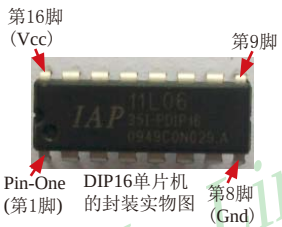
SKDIP28单片机的封装实物图

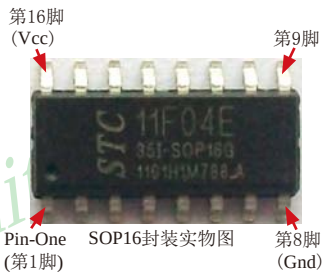




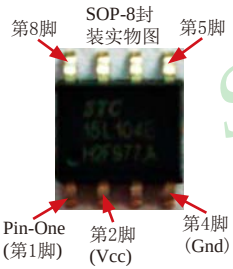


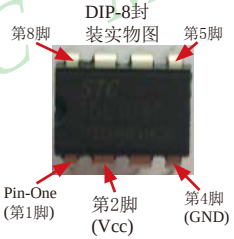


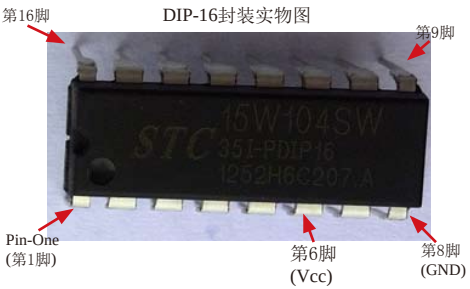




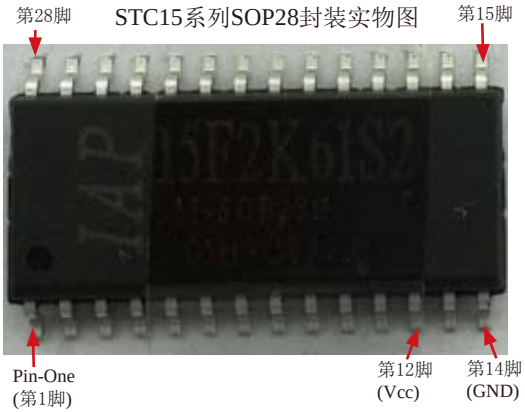
STC15系列单片机的封装实物图：





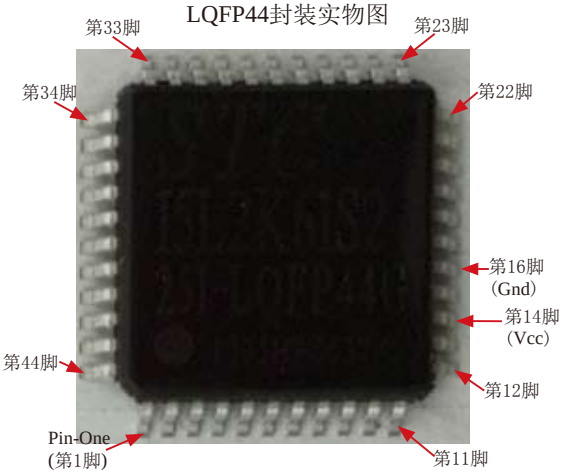


STC15系列SOP28封装实物图



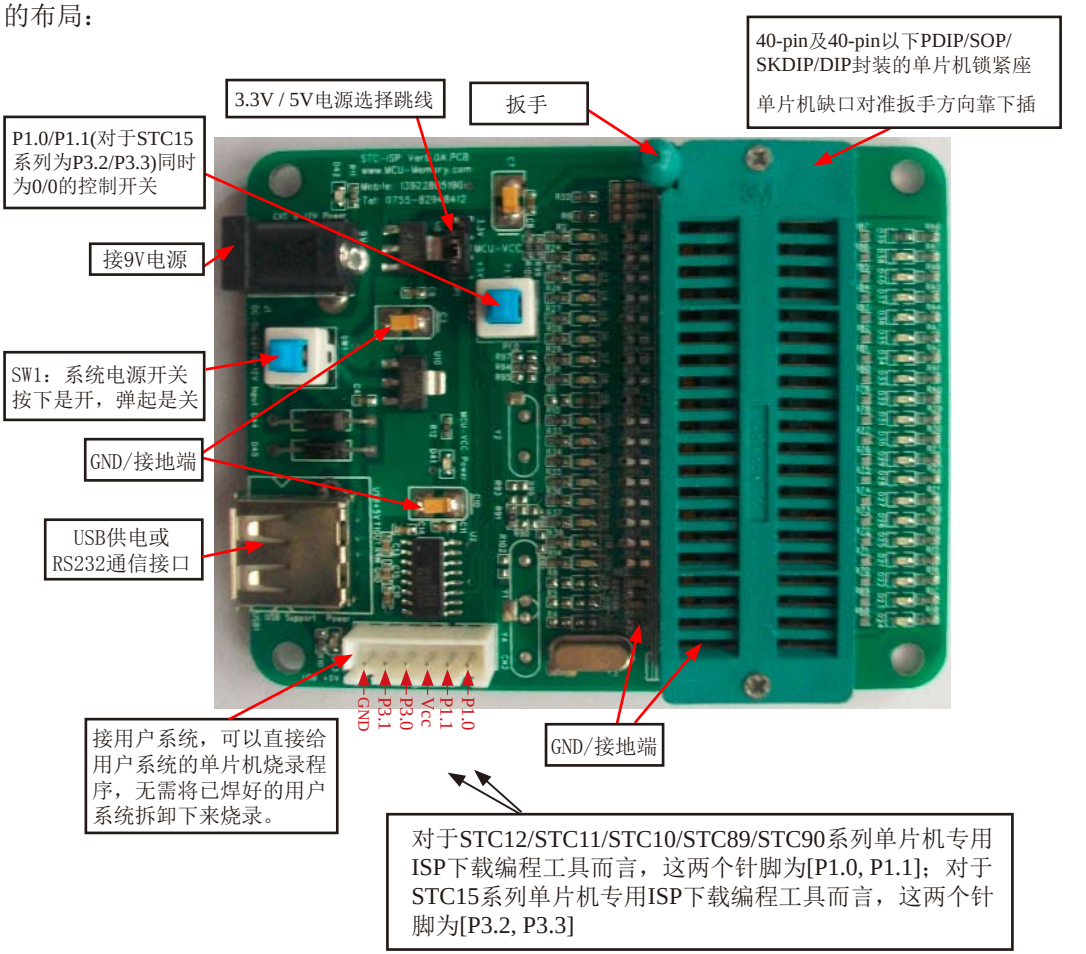
STC15系列PDIP40封装实物图

LQFP44封装实物图

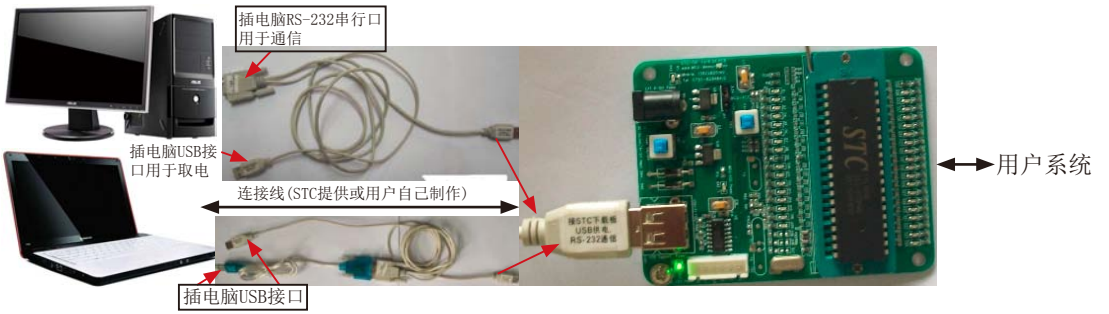


10.3.4 STC-ISP下载编程工具硬件——STC-ISP下载板

下面是STC12/11/10/89/90系列40-pin单片机专用ISP下载编程硬件工具——STC-ISP下载板的布局：



STC-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。



台式电脑或笔记本电脑
PC端控制软件:
V4.86或STC-ISP-15XX-v6.58

连接线(STC提供或用户自己制作)
对于没有标准RS232串口的笔记本电脑，此处除需STC提供或用户自己制作的连接线外，还需一条USB-RS232转换线来扩展一个RS232串口

STC-ISP下载编程工具

用户系统或
STC下载板

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：

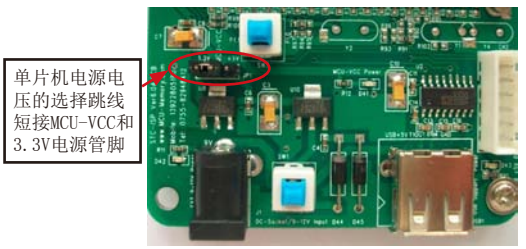
(1). MCU/单片机　RXD(P3.0)　---　RS-232转换器　---　电脑　TXD(COM Port Pin3)
(2). MCU/单片机　TXD(P3.1)　---　RS-232转换器　---　电脑　RXD(COM Port Pin2)
(3). MCU/单片机　GND　----------　电脑　GND(COM Port Pin5)
(4). 如果您的系统P3.0/P3.1连接到RS-485电路，推荐
在选项里选择“下次冷启动需要[P1.0, P1.1] = [0, 0]才可以下载用户程序”
这样冷启动后如[P1.0, P1.1]不同时为[0, 0]单片机直接运行用户程序，免得由于RS-485总线上的乱码造成单片机反复判断乱码是否为合法，浪费几百mS的时间，其实如果你的系统本身[P3.0, P3.1]就是做串口使用，也建议选择[P1.0, P1.1] = [0, 0]才可下载用户程序，以便下次冷启动直接运行用户程序。
(5). RS-232转换器可选用MAX232/SP232(4.5-5.5V),MAX3232/SP3232(3V-5.5V).

STC-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STC-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

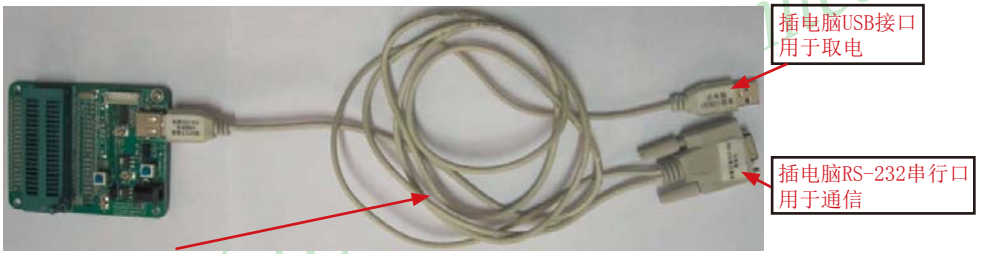


(2).将STC-ISP下载板连接到电脑端

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STC-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:

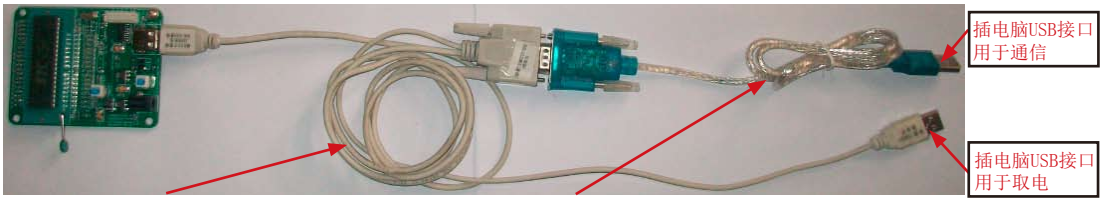
①.将一端有9芯连接座的插头插入电脑RS-232串行接口插座用于通信;

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STC-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STC-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STC-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).“[P1.0, P1.1](对于STC15系列为[P3.2, P3.3])同时为[0, 0]的控制开关“
处于非按下状态，[P1.0, P1.1] = [1, 1], 不短接到地；
处于按下状态， [P1.0, P1.1] = [0, 0], 短接到地。
如果单片机已被设成“下次冷启动[P1.0, P1.1] =[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**[P1.0, P1.1]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。**

10.3.5 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）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。

（2）尝试在STC-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。

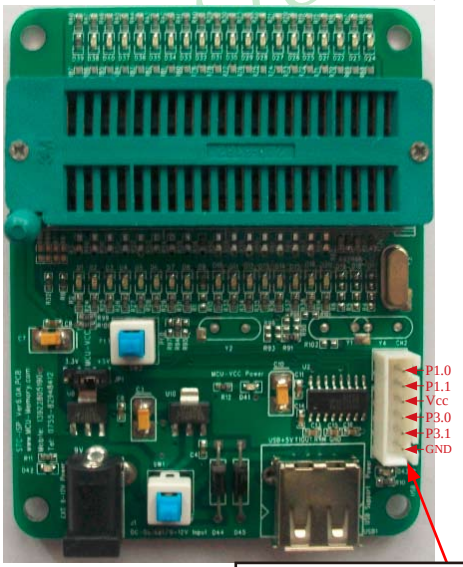
（3）**让**STC**帮您购买经过测试的转换线。**

10.3.6 如何用STC-ISP下载板给在用户系统上的单片机烧录用户程序

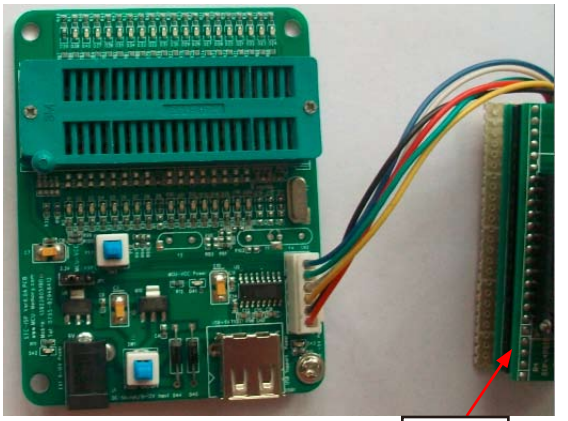
利用STC系列ISP下载编程工具(其实就是单片机通过RS-232转换器连接到电脑)进行RS-232转换。

单片机在用户自己的板上完成下载/烧录：

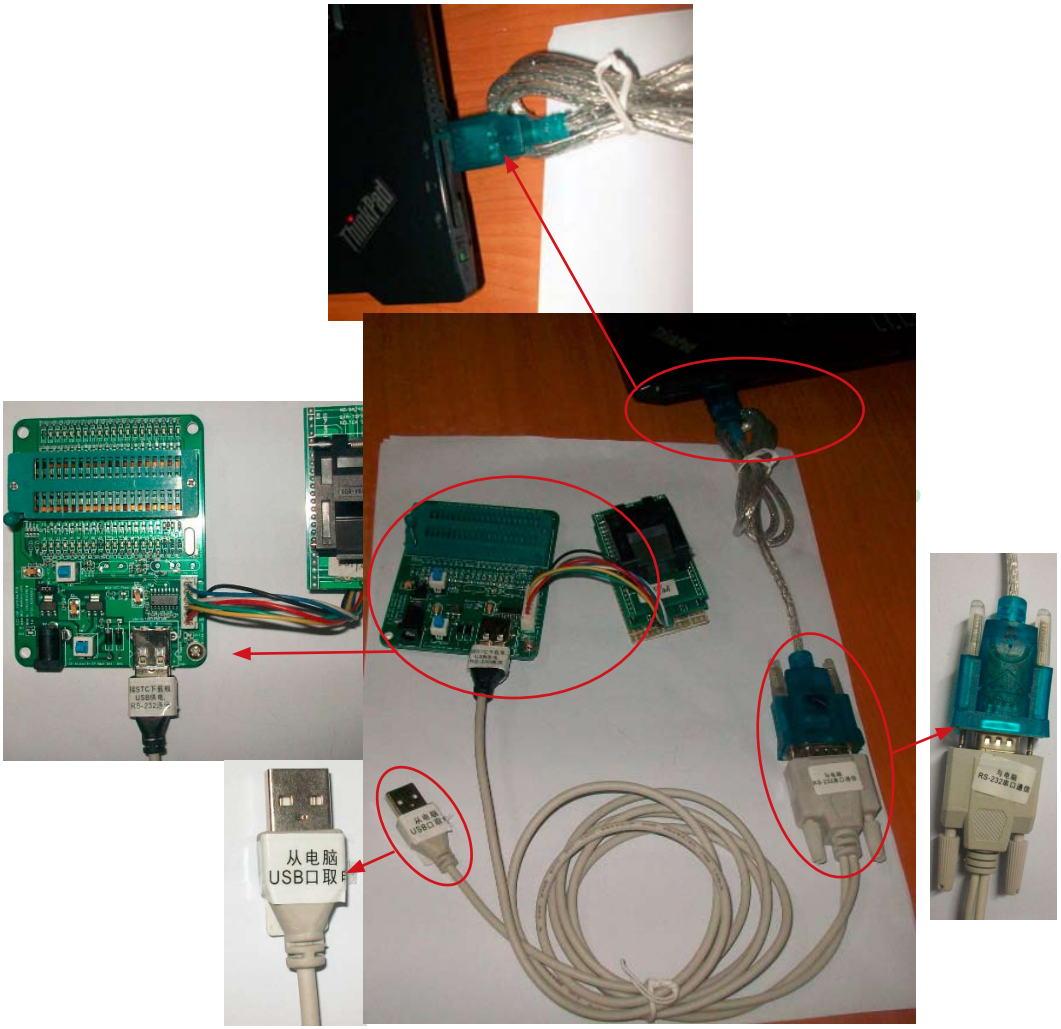
1. U1-Socket锁紧座不得插入单片机
2. 将用户系统上的电源(MCU-VCC,GND)及单片机的[P3.0, P3.1]接入转换板的“白色六芯插座”，如下图所示，这样用户系统上的单片机就具备了与电脑进行通信的能力
3. 将用户系统的单片机的[P1.0, P1.1] (对于STC15系列为[P3.2, P3.3]) 接入转换板“白色六芯插座”(如果需要的话)
4. 如须[P1.0, P1.1] =[0, 0], 短接到地，可在用户系统上将其短接到地，或将[P1.0, P1.1]也从用户系统引到STC系列ISP下载编程工具(其实就是单片机通过RS-232转换器连接到电脑)上，将“控制[P1.0, P1.1]同时为[0, 0]的开关”按下，则[P1.0, P1.1] = [0, 0]。
5. 将STC-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. 如有RS-485晶片连到P3.0/P3.1,或其他线路，在下载时应将其断开。



接用户系统，可以直接给用户系统的单片机烧录程序，无需将已焊好的用户系统拆卸下来烧录。



将用户系统按上图所示连接到STC-ISP下载板上



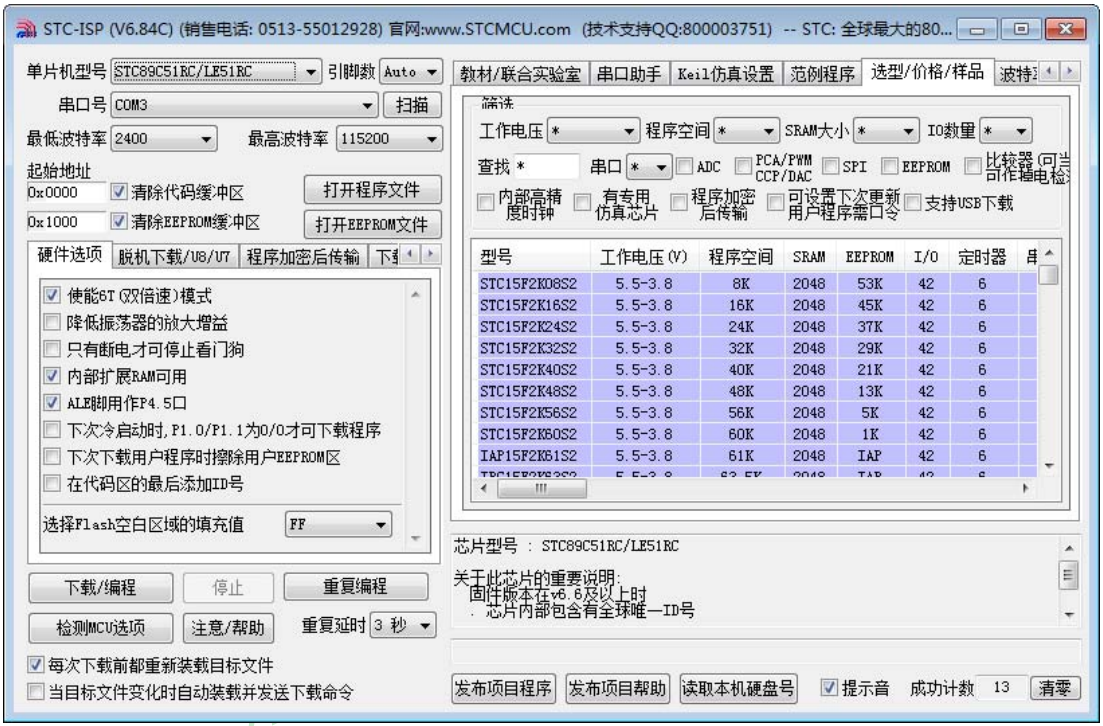
将连有用户系统的STC-ISP下载板按左图所示连接到电脑上，注意以下几点：

（1）STC-ISP下载板的锁紧座不得插入单片机；

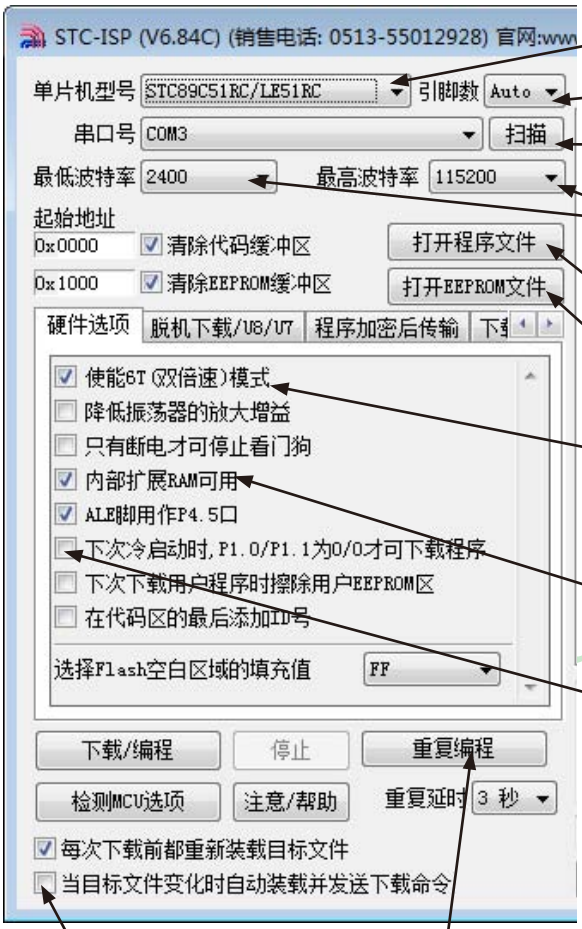
（2）“从电脑USB口取电”的USB插头悬空，不要插入电脑，因为是**从用户系统自供电的。**

（3）接STC下载板的USB插头仅用于RS232通信。

10.3.7 电脑端的STC-ISP控制软件(Ver6.85)的界面使用说明



最新的ISP下载控制软件V6.85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STC-ISP-V6.85软件的各个功能。



选择STC系列单片机的型号

选择STC89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115200,38400,19200,2400或Auto Buad

打开用户的程序代码文件

打开EEPROM数据文件

系统频率是否加倍
选择：　采用6T(双倍速)模式
不选择：不加倍，采用12T(单倍速)模式

是否使用MCU内部的扩展RAM
选择：　内部扩展RAM可用
不选择：内部扩展RAM不可用

下次是否需要[P1.0, P1.1]同时为低电平时才可下载程序
选择：　[P1.0, P1.1]同时为低电平时才可下载程序
不选择：下载时不检测[P1.0, P1.1]的电平

如P3.0/P3.1外接RS-485/RS-232等通信电路，建议选择P1.0/P1.1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。

新的设置冷启动后（彻底停电后再上电），才生效

大批量生产时使用

如P3.0/P3.1外接RS-485/RS-232等通信电路，建议选择P1.0/P1.1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。

Step1/步骤1：选择你所使用的单片机型号，如STC89C51RC等

Step2/步骤2：打开文件，要烧录用户程序，必须调入用户的程序代码（*.bin, *.hex）

Step3/步骤3：选择串行口，你所使用的电脑串口，如串行口1--COM1,串行口2--COM2,...

有些新式笔记本电脑没有RS-232串行口,可买一条USB-RS232转接器，人民币50元左右。

有些USB-RS232转接器，不能兼容，可让STC帮你购买经过测试的转换器。

Step4/步骤4：选择下次冷启动后，时钟源为“内部R/C振荡器”还是“外部晶体或时钟”

Step5/步骤5：选择“Download/下载”按钮下载用户的程序进单片机内部，可重复执行Step5/步骤5， 也可选择“Re-Download/重复下载”按钮

下载时注意看提示，主要看是否要给单片机上电或复位，下载速度比一般通用编程器快。

一定要先选择“Download/下载”按钮，然后再给单片机上电复位(先彻底断电)，而不要先上电，先上电，检测不到合法的下载命令流，单片机就直接跑用户程序了。

关于硬件连接：

(1). MCU/单片机　RXD(P3.0)　---　RS-232转换器　---　PC/电脑　TXD(COM Port Pin3)

(2). MCU/单片机　TXD(P3.1)　---　RS-232转换器　---　PC/电脑　RXD(COM Port Pin2)

(3). MCU/单片机　GND　----------　PC/电脑　GND(COM Port Pin5)

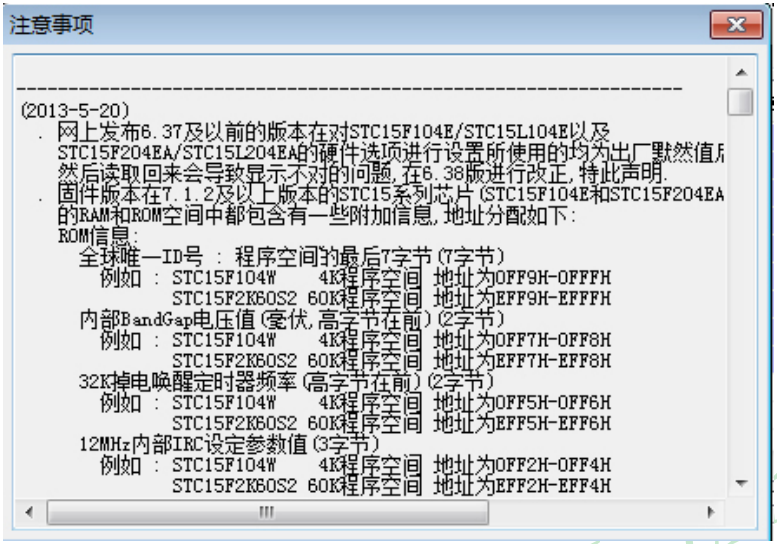
(4). 如果您的系统P3.0/P3.1连接到RS-485电路，推荐

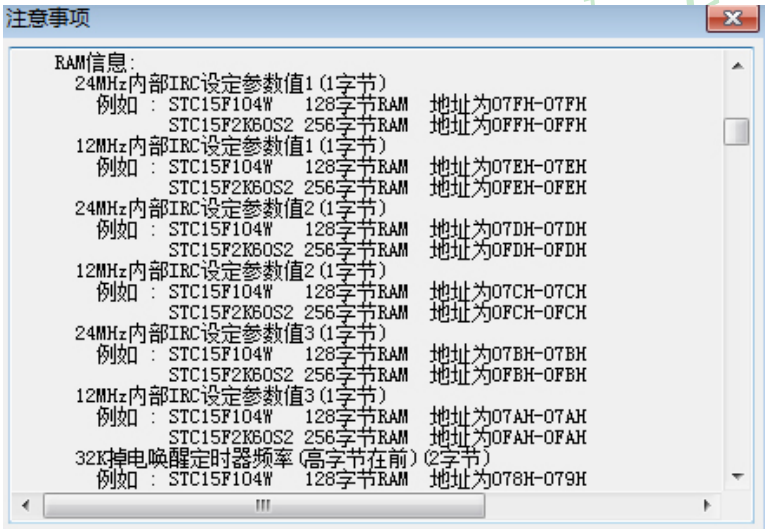
在选项里选择“下次冷启动需要P1.0/P1.1 = 0,0才可以下载用户程序”

这样冷启动后如 P1.0, P1.1不同时为0,单片机直接运行用户程序，免得由于RS-485总线上的乱码造成单片机反复判断乱码是否为合法，浪费几百mS的时间，其实如果你的系统本身P3.0,P3.1就是做串口使用，也建议选择P1.0/P1.1 = 0/0才可下载用户程序，以便下次冷启动直接运行用户程序。

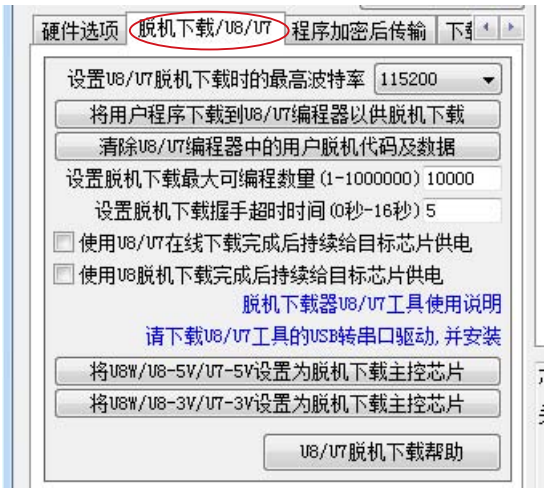
(5). RS-232转换器可选用MAX232/SP232(4.5-5.5V),MAX3232/SP3232(3V-5.5V).

点击界面上的注意/帮助按钮后出现下面的对话框：

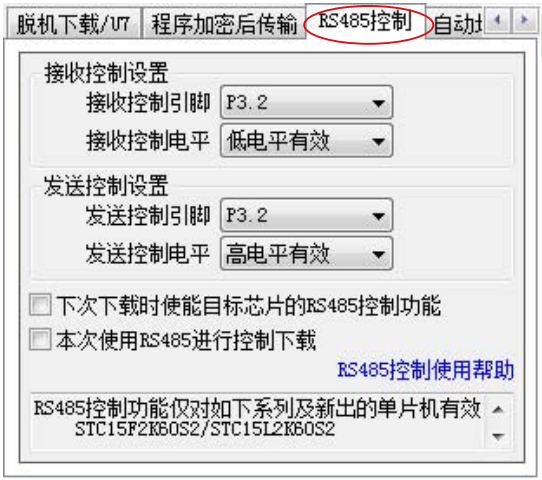




脱机下载界面:

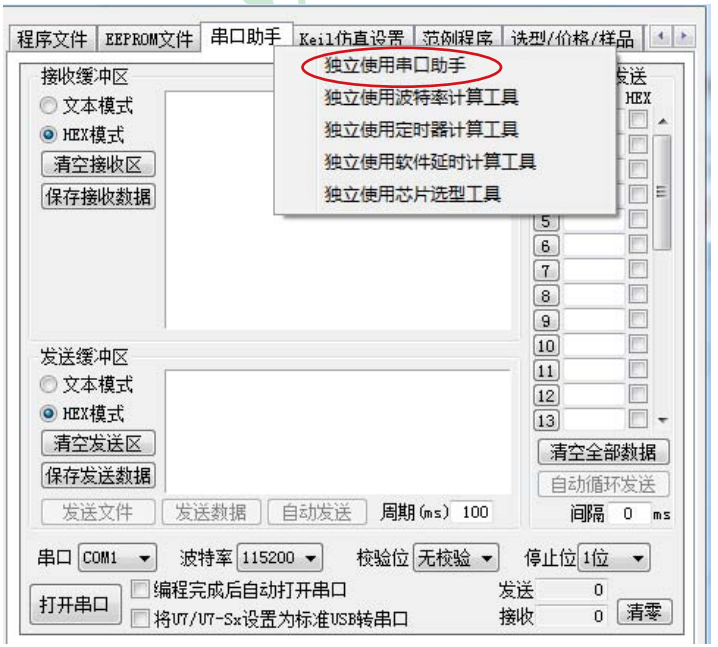


RS485控制界面:



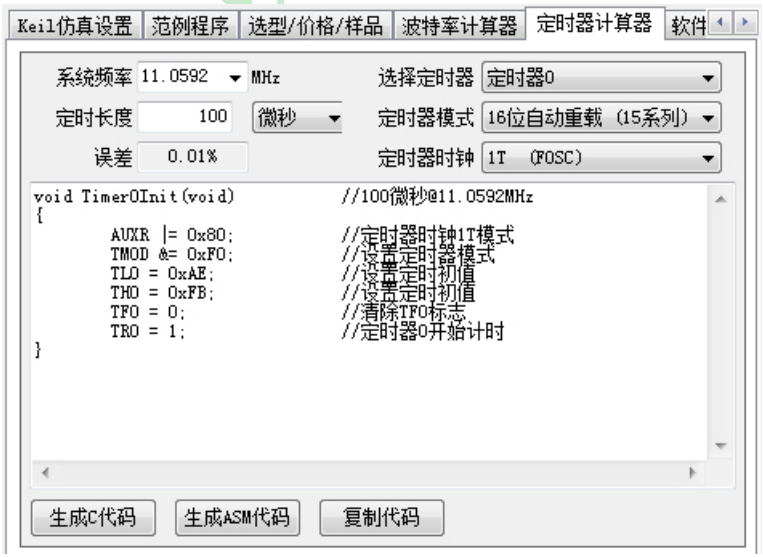
串口助手界面：

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STC-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STC-ISP-V6.85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：

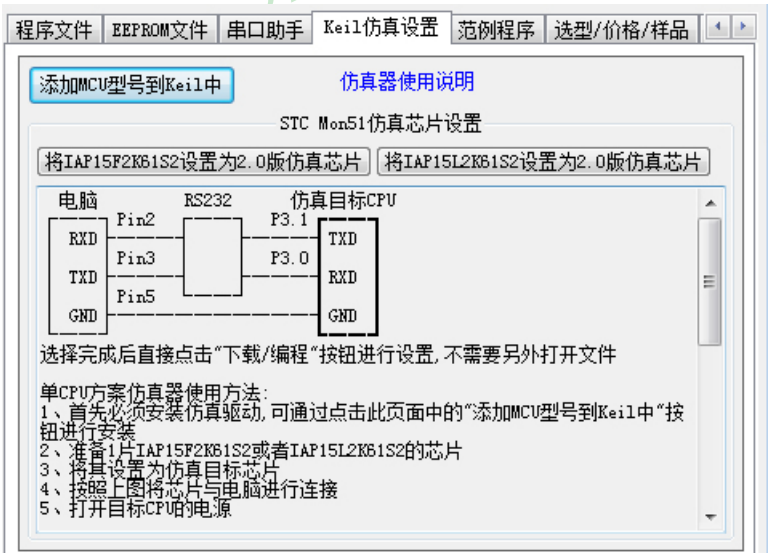
最新的STC-ISP-V6.85软件还集成了定时器计算器，定时器计算器也可以生成相应的代码(C或ASM代码)，根据用户的设置对定时器的各相关寄存器进行初始化。定时器计算器界面如下所示：



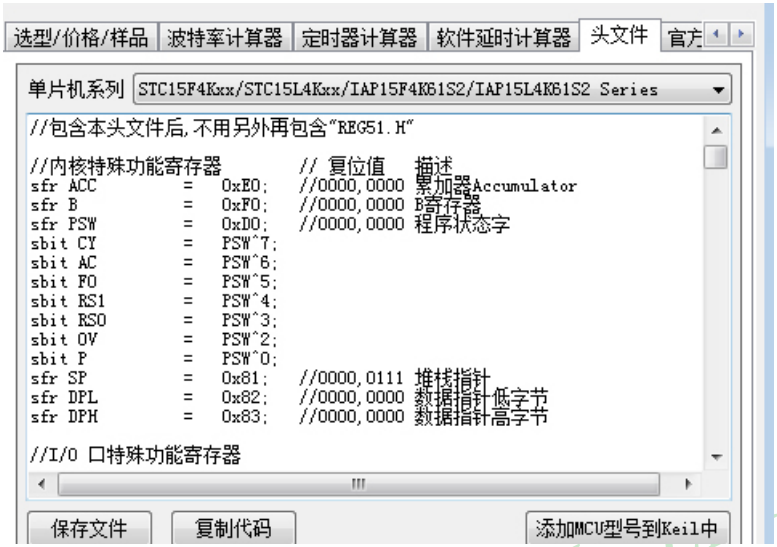
另外，最新的STC-ISP-V6.85软件还集成了软件延时计算器，软件延时计算器也可以生成相应的代码(C或ASM代码)，根据用户的设置可以生成相应的延时子函数。软件延时计算器界面如下所示：

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STC-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。

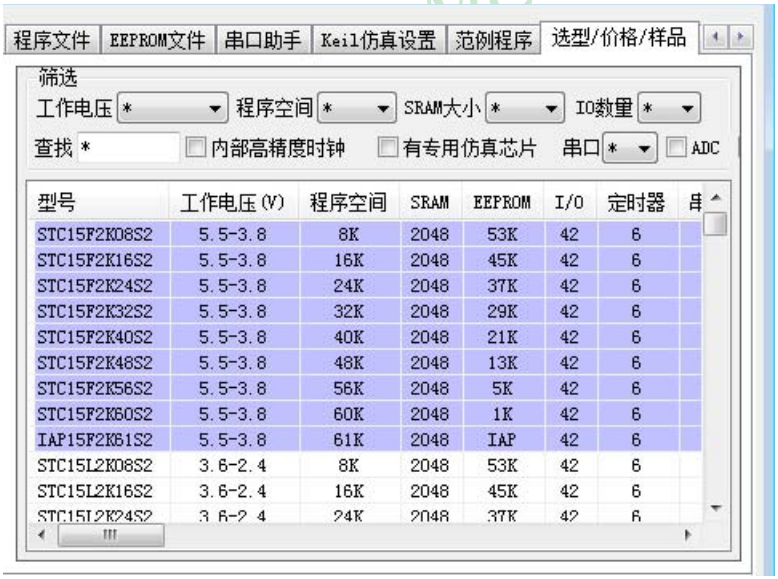
最新的STC-ISP-V6.85软件还设计了“Keil仿真设置”选项，如下图所示



最新的STC-ISP-V6.85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STC-ISP-V6.85软件中查询STC系列单片机的选型和价格。



10.3.8 STC-ISP控制软件(Ver6.85)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。

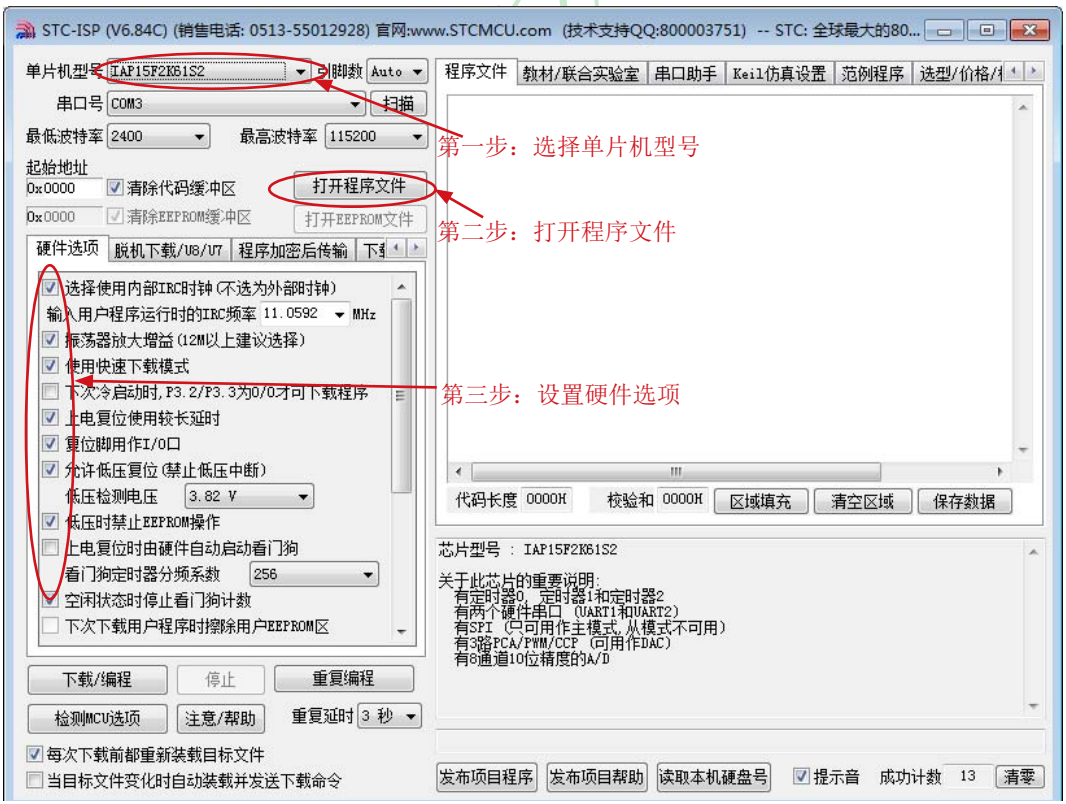
关于界面，用户可以自己进行定制（用户可以自行修改发布项目程序的标题、按钮名称以及帮助信息），同时用户还可以指定目标电脑的硬盘号和目标芯片的ID号，指定目标电脑的硬盘号后，便可以控制发布应用程序只能在指定的电脑上运行(防止烧录人员将程序轻易从电脑盗走,如通过网络发走,如通过U盘烤走,防不胜防,当然盗走你的电脑那就没办法那,所以STC的脱机下载工具比电脑烧录安全,能限制可烧录芯片数量,让前台文员小姐烧,让老板娘烧都可以)，拷贝到其它电脑，应用程序不能运行。同样的，当指定了目标芯片的ID号后，那么用户代码只能下载到具有相应ID号的目标芯片中(对于一台设备要卖几千万的产品特别有用---坦克,可以发给客户自己升级,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：

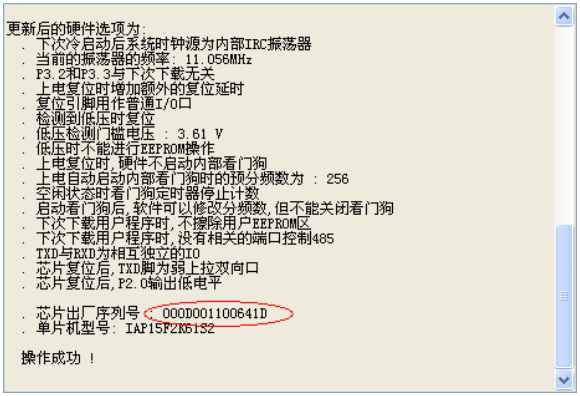
1、首先选择目标芯片的型号

2、打开程序代码文件

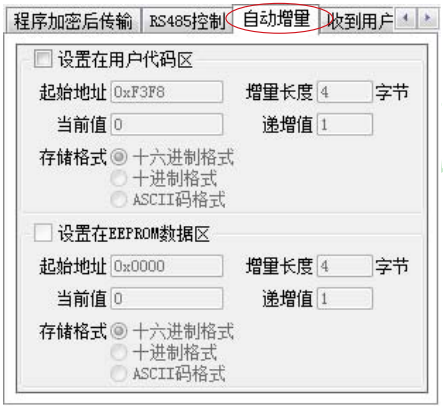
3、设置好相应的硬件选项



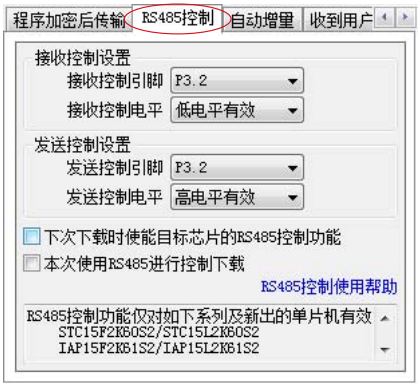
4、试烧一下芯片，并记下目标芯片的ID号，如下图所示，该芯片的ID号即为“000D001100641D”（如不需要对目标芯片的ID号进行校验，可跳过此步）



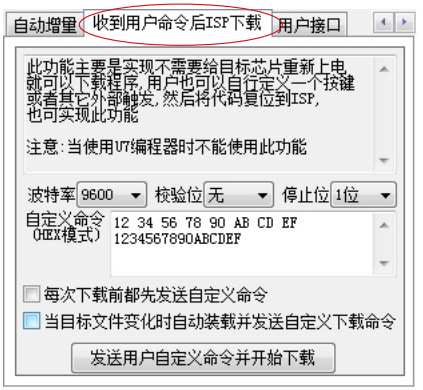
5、设置自动增量（如不需要自动增量，可跳过此步



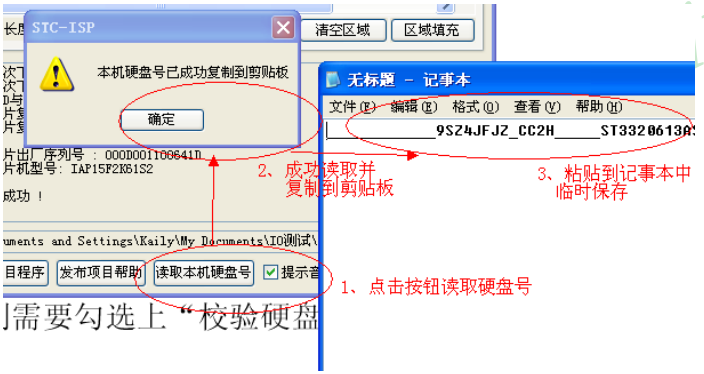
6、设置RS485控制信息（如不需要RS485控制，可跳过此步）



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）

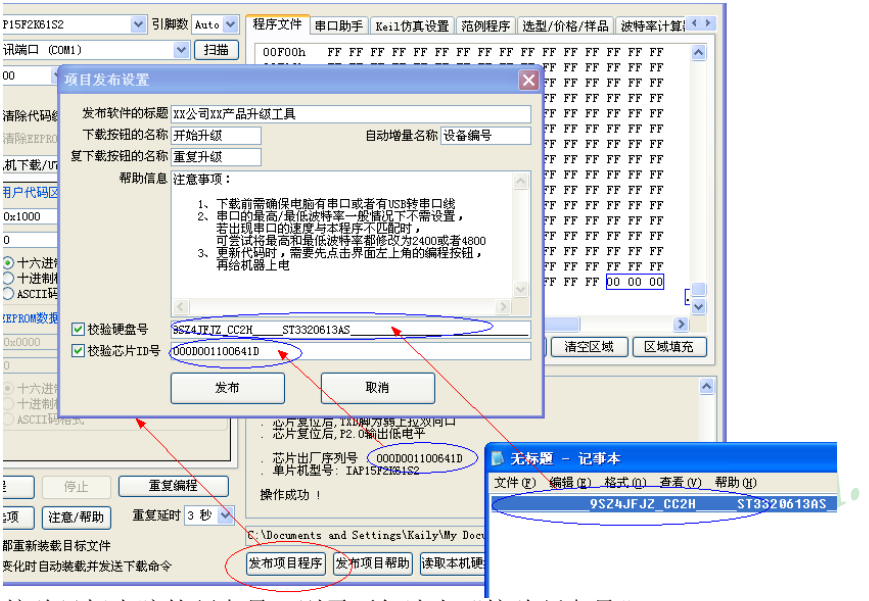


9、点击“发布项目程序”按钮，进入发布应用程序的设置界面。

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

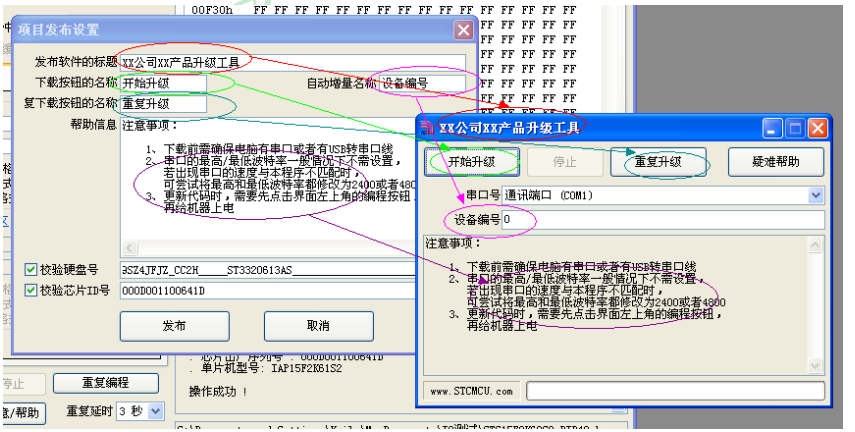
11、若需要校验目标电脑的硬盘号,则需要勾选上“校验硬盘号”，并在后面的文本框内输入前面所记下的目标电脑的硬盘号

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

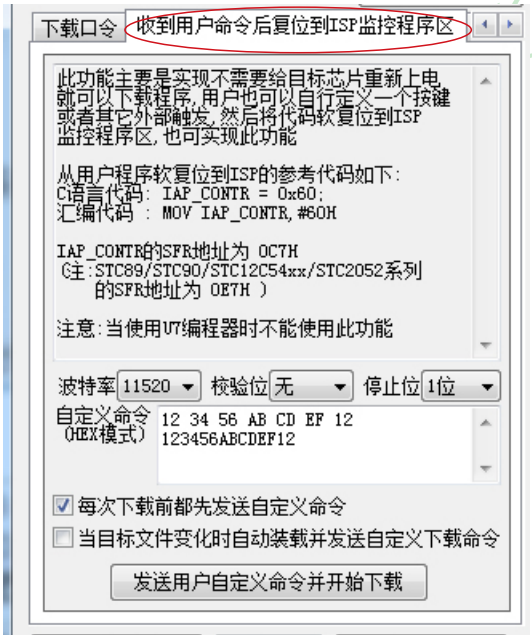
10.3.9 运行用户程序时收到用户命令后自动启动ISP下载(不停电)

“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。

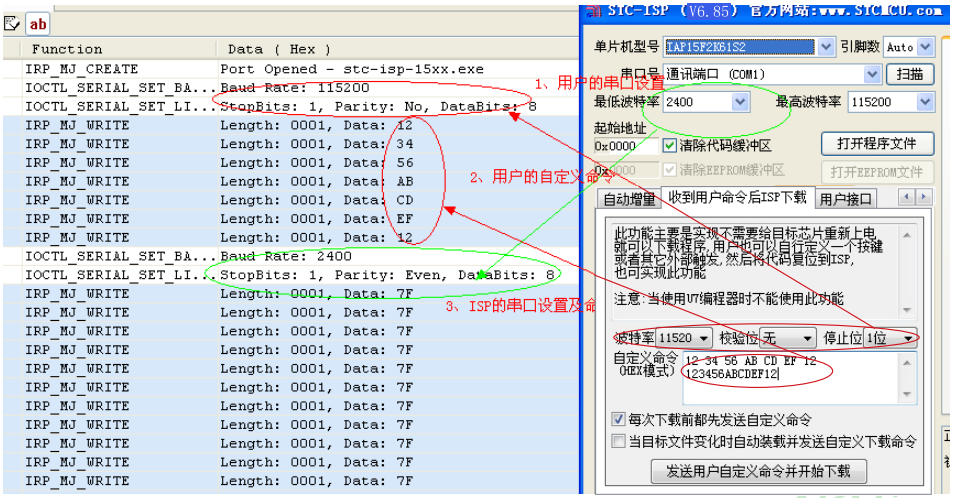
具体的功能为：电脑或脱机下载板在开始发送真正的ISP下载编程握手命令前，先发送用户自定义的一串命令（关于这一串串口命令，用户可以根据自己在应用程序中的串口设置来设置波特率、校验位以及停止位），然后再立即发送ISP下载编程握手命令。

“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



10.3.10 用户接口

STC-ISP-V6.85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。

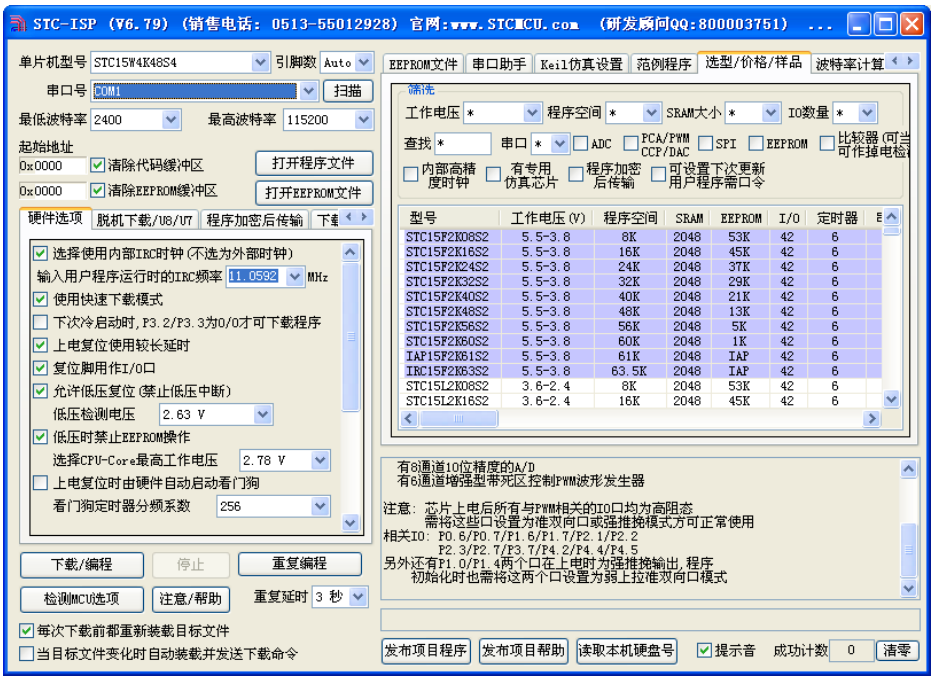
使用用户接口功能时，PC机或者U7编程器首先与用户单片机通讯，将单片机的重要信息(如：序列号等)读取出来并保存，然后用户可以设置发送给用户代码的自定义命令(如设置发送给用户代码的读数据命令或复位命令)，用户代码可以接收自定义命令。当用户代码收到复位命令后可以控制目标单片机自动复位，若用户未设置复位命令，则用户需手动给目标单片机重新上电，当目标单片机上电复位后，就开始更新代码了，此时更新的代码包括上述PC机或U7编程器所保存的重要信息和用户新代码，即将PC机或U7编程器所保存的重要信息和用户新代码一并写入了目标单片机中，从而实现了保留目标单片机中的重要信息不被破坏的目的。

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。

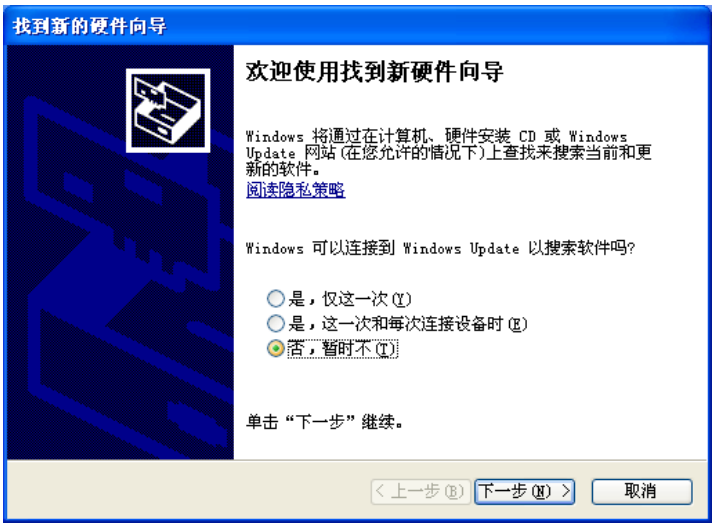
10.3.11 STC-USB驱动程序安装说明

10.3.11.1 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STC-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。

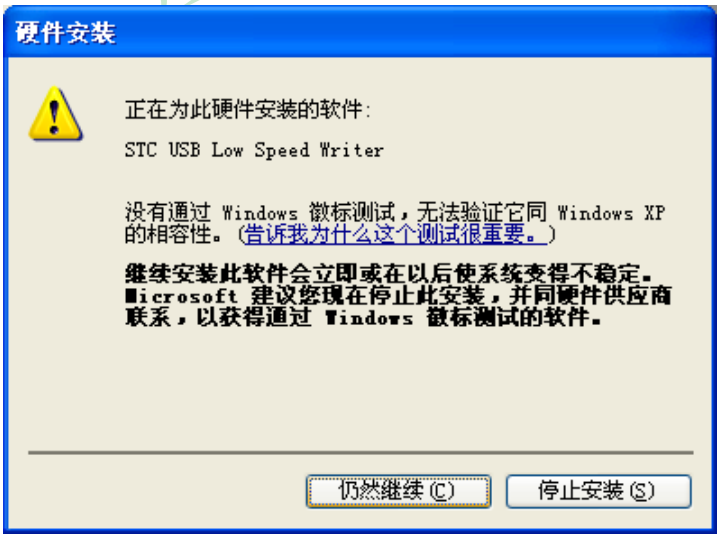


插入USB设备，系统找到设备后自动弹出如下对话框，选择其中的“否，暂时不”项



在下面的对话框中选择“自动安装软件(推荐)”项

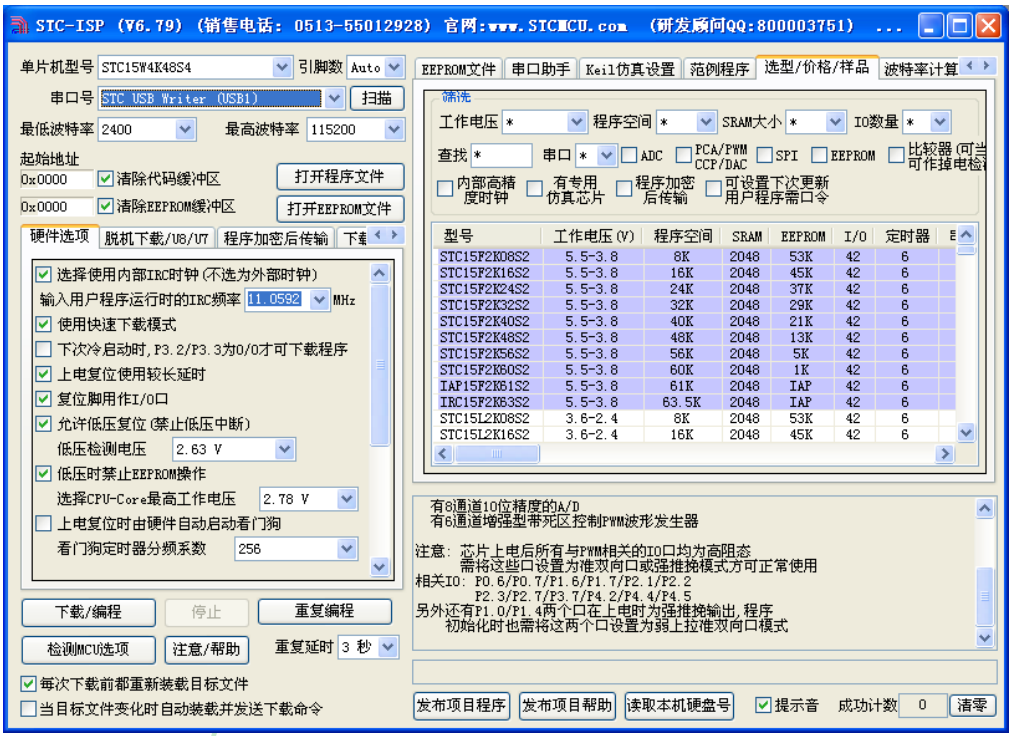
在弹出的下列对话框中，选择“仍然继续”按钮



接下系统会自动安装驱动，如下图

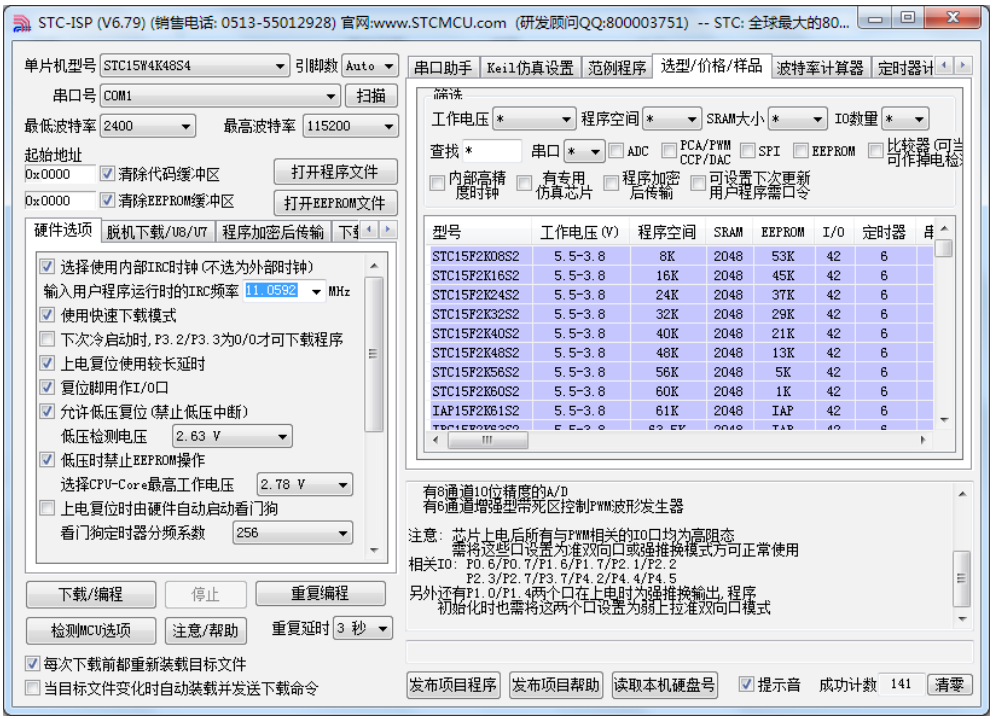
出现下面的的对话框表示驱动安装完成

此时，之前打开的STC-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：

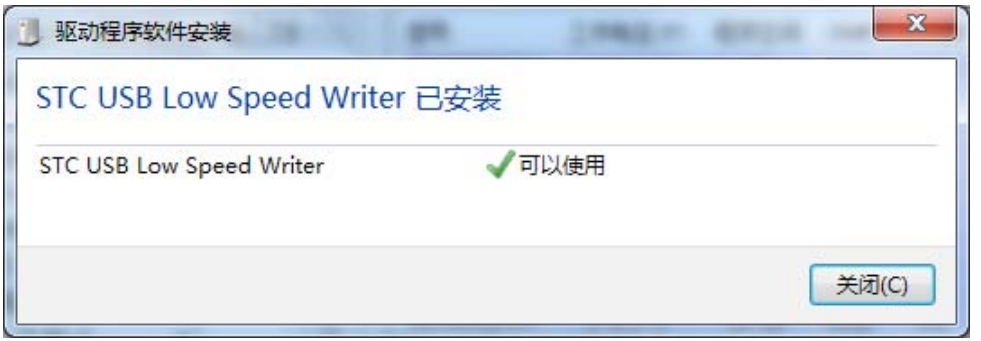


10.3.11.2 Windows 7（32位）操作系统下的STC-USB驱动程序安装说明

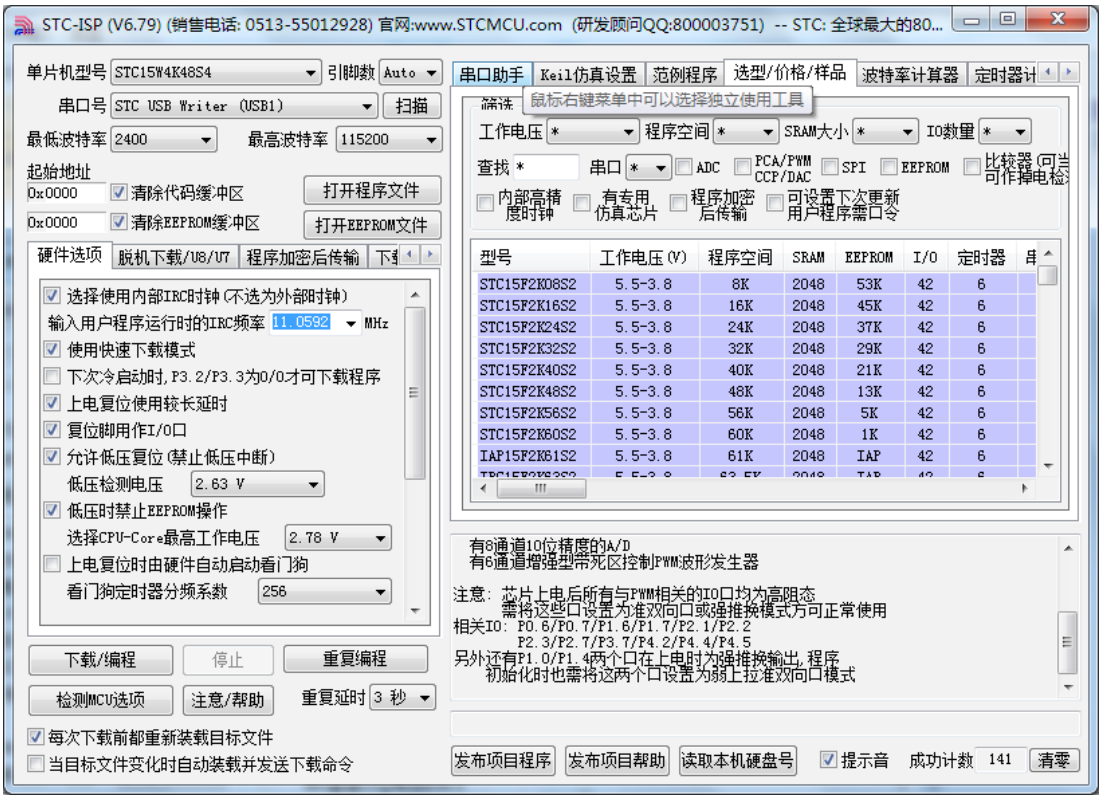
打开V6.85版（或者更新的版本）的STC-ISP下载软件，下载软件会自动将驱动文件复制到相关的系统目录



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。



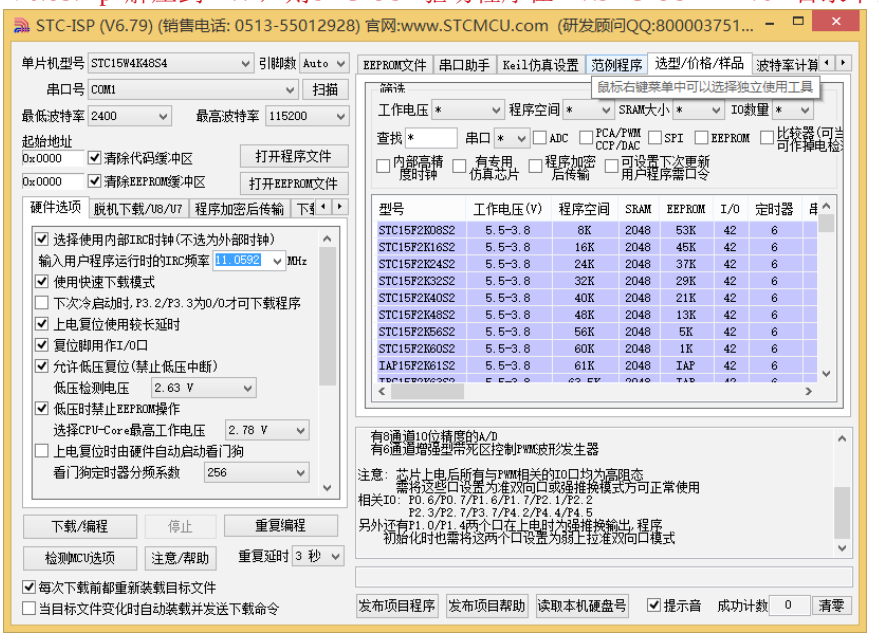
此时，之前打开的STC-ISP下载软件中的串口号列表会自动选择所插入的USB设备，并显示设备名称为“STC USB Writer (USB1)”，如下图：



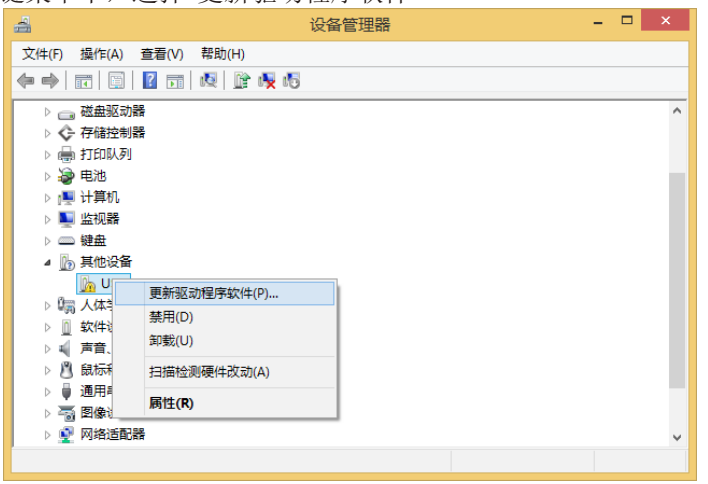
注：若Windows 7下，系统并没有自动安装驱动，则驱动的安装方法请参考Windows 8（32位）的安装方法

10.3.11.3 Windows 8（32位）操作系统下的STC-USB驱动程序安装说明

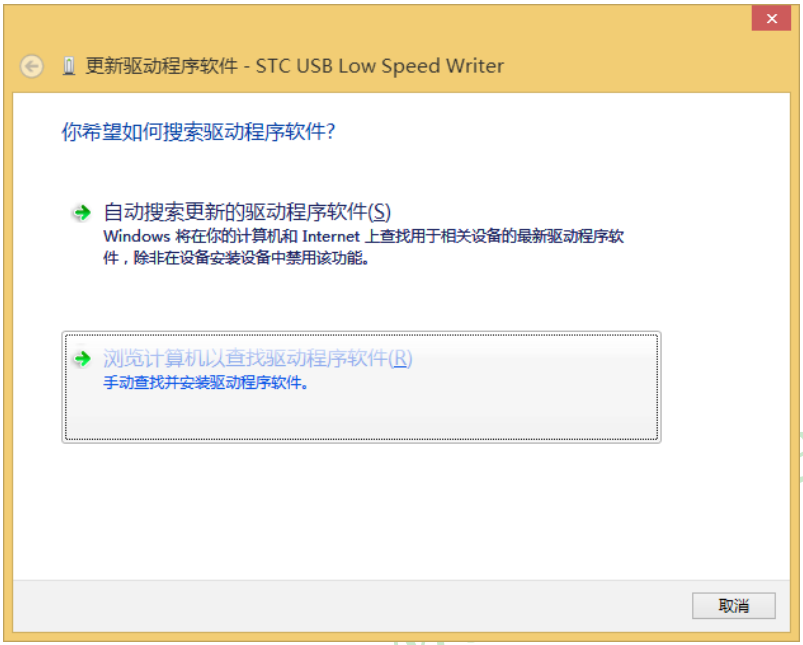
打开V6.85版（或者更新的版本）的STC-ISP下载软件（由于权限的原因，在Windows 8中下载软件不会将驱动文件复制到相关的系统目录，需要用户手动安装。首先从STC官方网站下载“stc-isp-15xx-V6.85.zip”（或更新版本），下载后解压到本地磁盘，则STC-USB的驱动文件也会被解压到当前解压目录中的“STC-USB Driver”中（例如将下载的压缩文件“stc-isp-15xx-V6.85.zip”解压到“F:\”，则STC-USB驱动程序在“F:\STC-USB Driver”目录中））



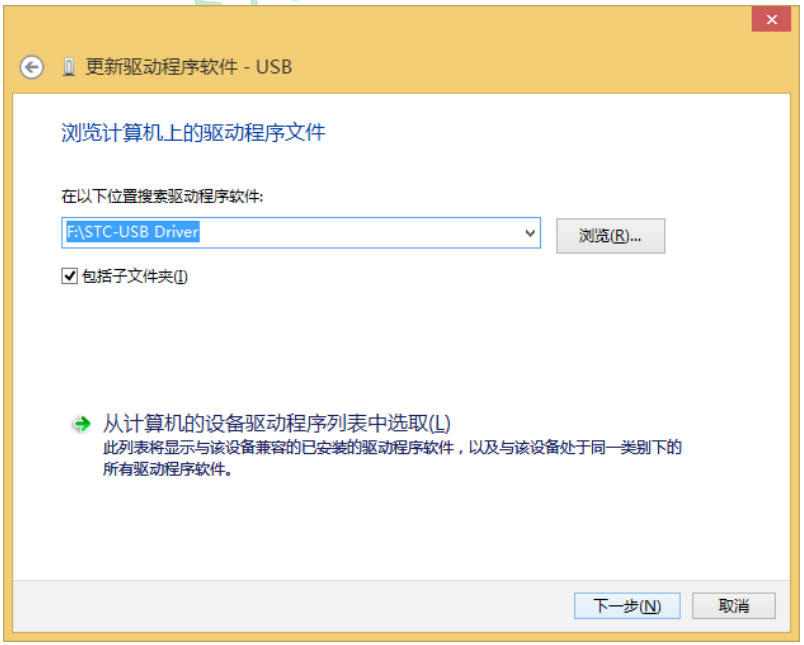
插入USB设备，并打开“设备管理器”。找到设备列表中带黄色感叹号的USB设备，在设备的右键菜单中，选择“更新驱动程序软件”



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”

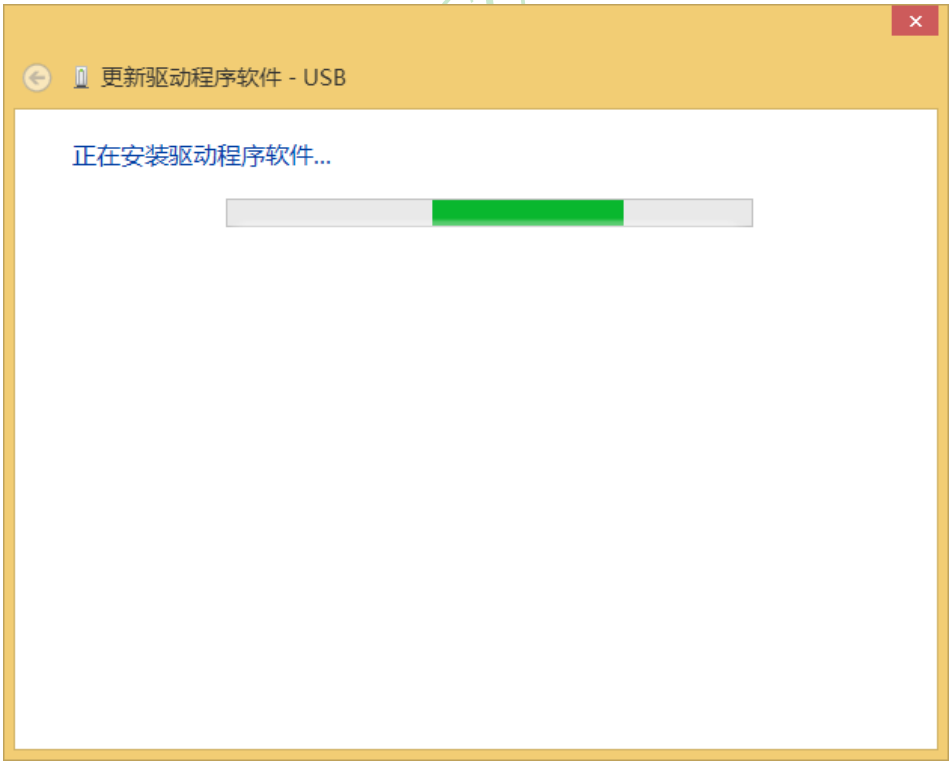


单击下面对话框中的“浏览”按钮，找到之前STC-USB驱动程序的存放目录（例如：之前的示例目录为“F:\STC-USB Driver”，用户将路径定位到实际的解压目录）

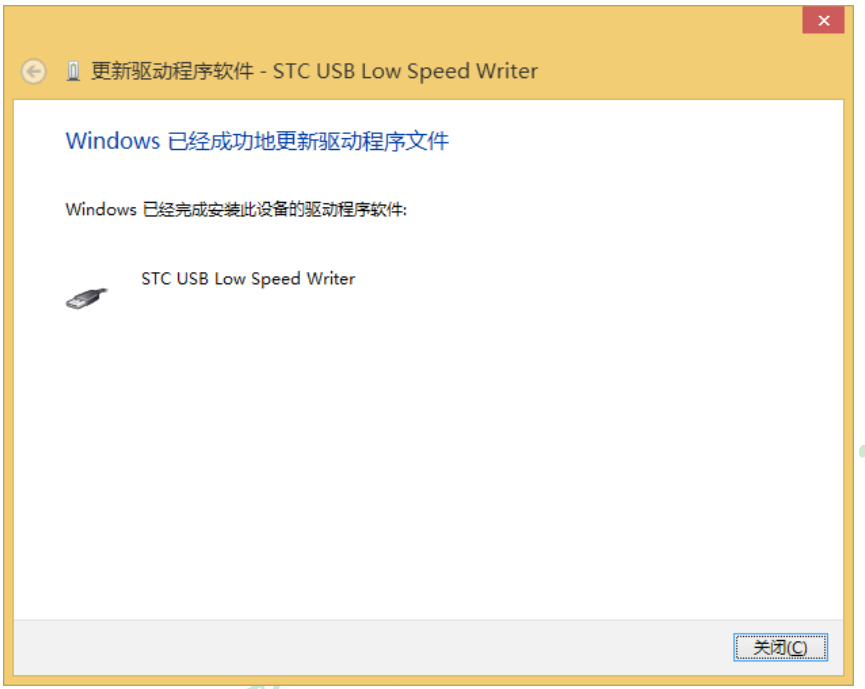


驱动程序开始安装时，会弹出如下对话框，选择“始终安装此驱动程序软件”

接下来，系统会自动安装驱动，如下图

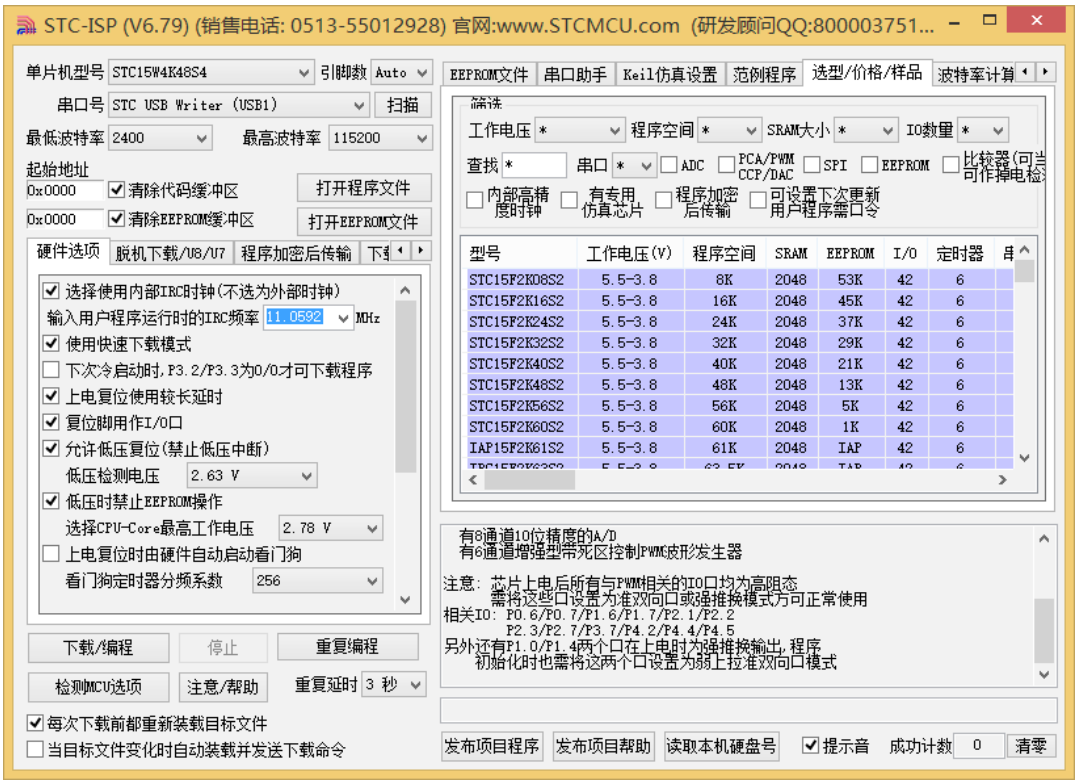


出现下面的的对话框表示驱动安装完成



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STC-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：

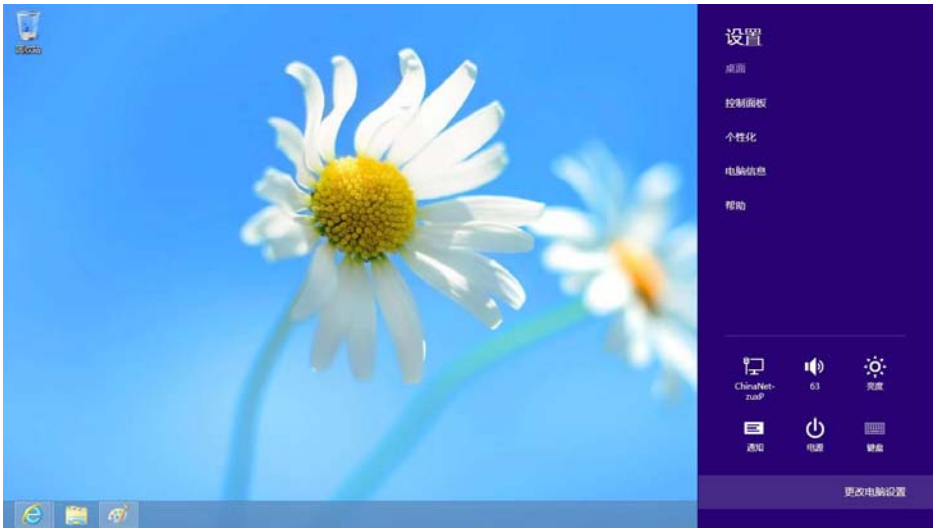


10.3.11.4 Windows 8（64位）操作系统下的STC-USB驱动程序安装说明

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

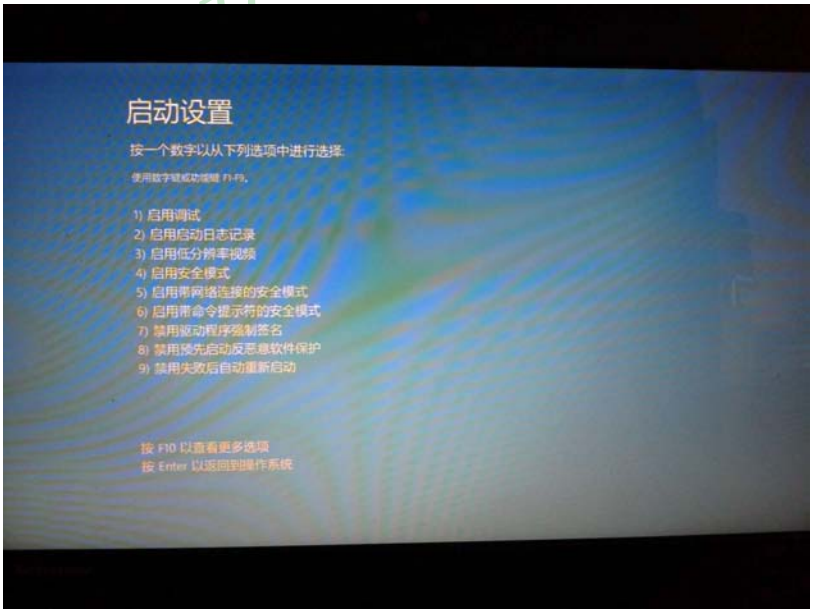
在下面的界面中，选择“疑难解答”项

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



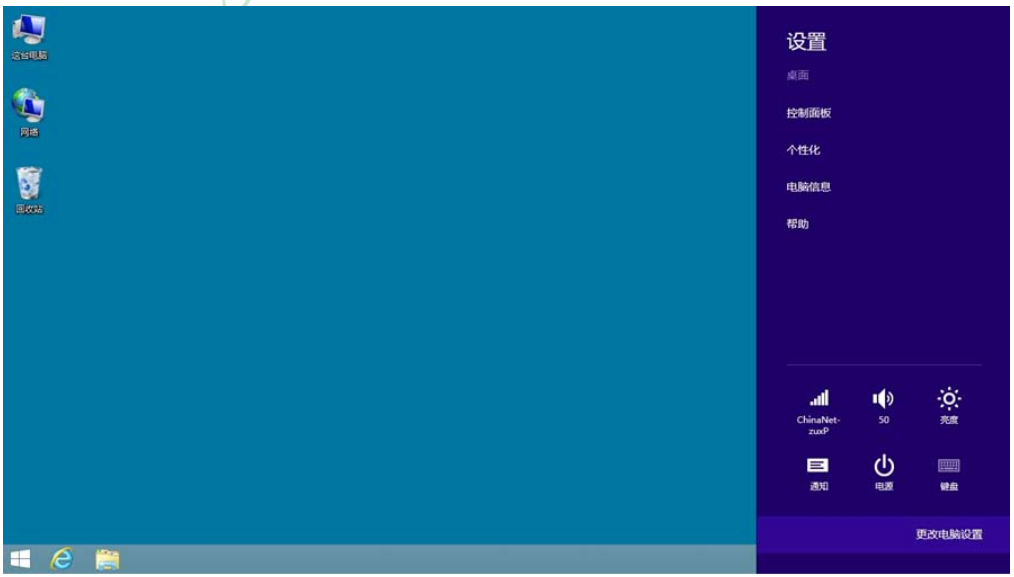
启动到Windows 8后，按照Windows 8（32位）的安装说明即可完成驱动的安装

10.3.11.5 Windows 8.1（64位）操作系统下的STC-USB驱动程序安装说明

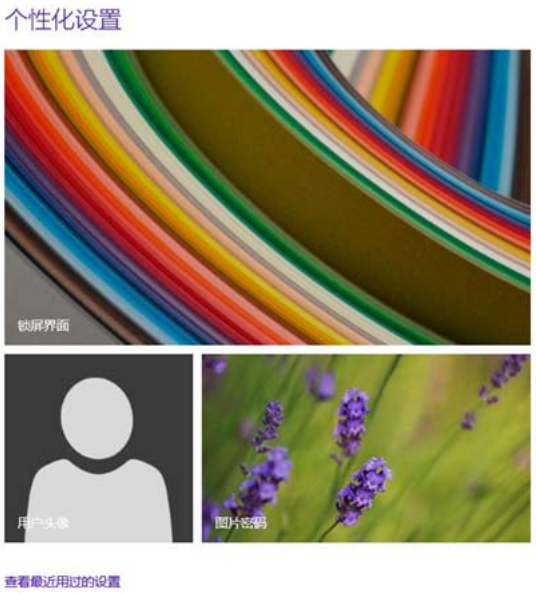
Windows 8.1与Windows 8进入高级启动菜单的方法不一样,在此专门进行说明。

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

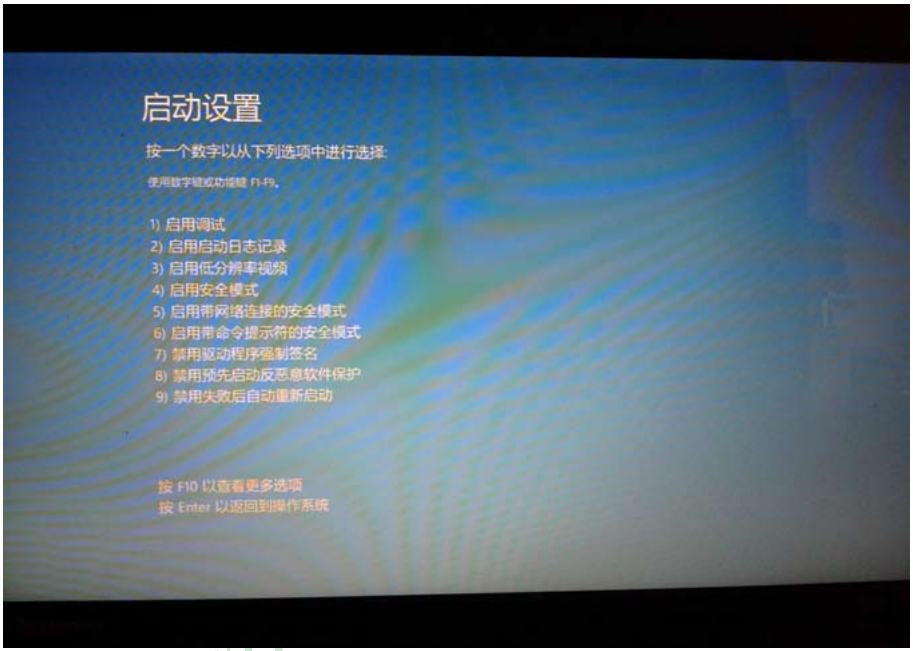
在下面的界面中，选择“疑难解答”项

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 8.1后，按照Windows 8（32位）的安装方法即可完成驱动的安装

10.4 STC仿真器说明指南

1. 仿真器的参考硬件图

（1）利用RS-232转换器连接电脑的的仿真应用线路图

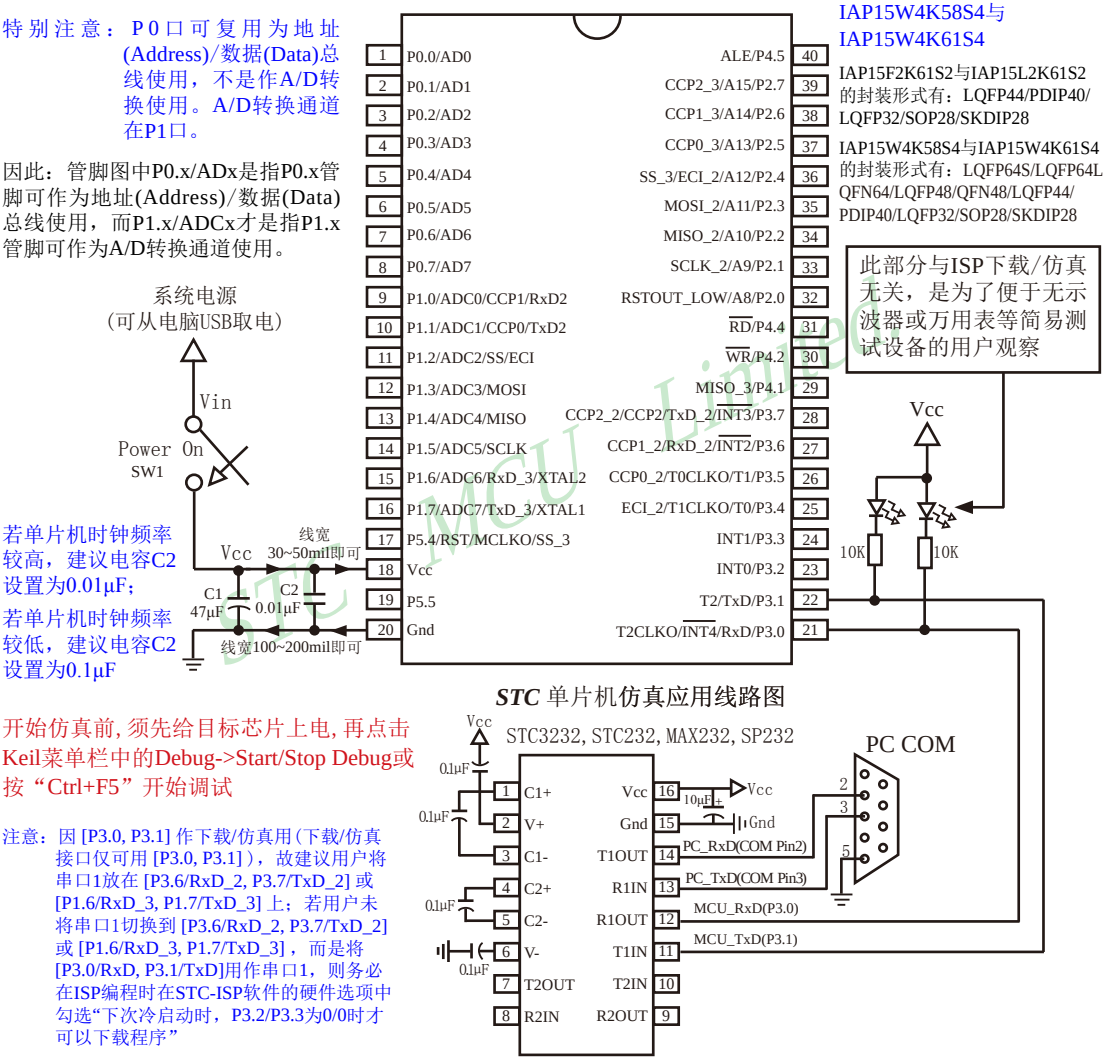
特别注意：P0口可复用为地址(Address)/数据(Data)总线使用，不是作A/D转换使用。A/D转换通道在P1口。

因此：管脚图中P0.x/ADx是指P0.x管脚可作为地址(Address)/数据(Data)总线使用，而P1.x/ADCx才是指P1.x管脚可作为A/D转换通道使用。

目前支持仿真的芯片只有IAP15F2K61S2, IAP15L2K61S2, IAP15W4K58S4与IAP15W4K61S4

IAP15F2K61S2与IAP15L2K61S2的封装形式有：LQFP44/PDIP40/LQFP32/SOP28/SKDIP28

IAP15W4K58S4与IAP15W4K61S4的封装形式有：LQFP64S/LQFP64L QFN64/LQFP48/QFN48/LQFP44/PDIP40/LQFP32/SOP28/SKDIP28



STC 单片机仿真应用线路图

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

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/RxD_2, P3.7/TxD_2] 或 [P1.6/RxD_3, P1.7/TxD_3] 上；若用户未将串口1切换到 [P3.6/RxD_2, P3.7/TxD_2] 或 [P1.6/RxD_3, P1.7/TxD_3]，而是将 [P3.0/RxD, P3.1/TxD]用作串口1，则务必在ISP编程时在STC-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”

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 STC-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%温飘(-40℃~+85℃)，常温下温飘±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

目前支持仿真的芯片只有IAP15F2K61S2, IAP15L2K61S2, IAP15W4K58S4与IAP15W4K61S4

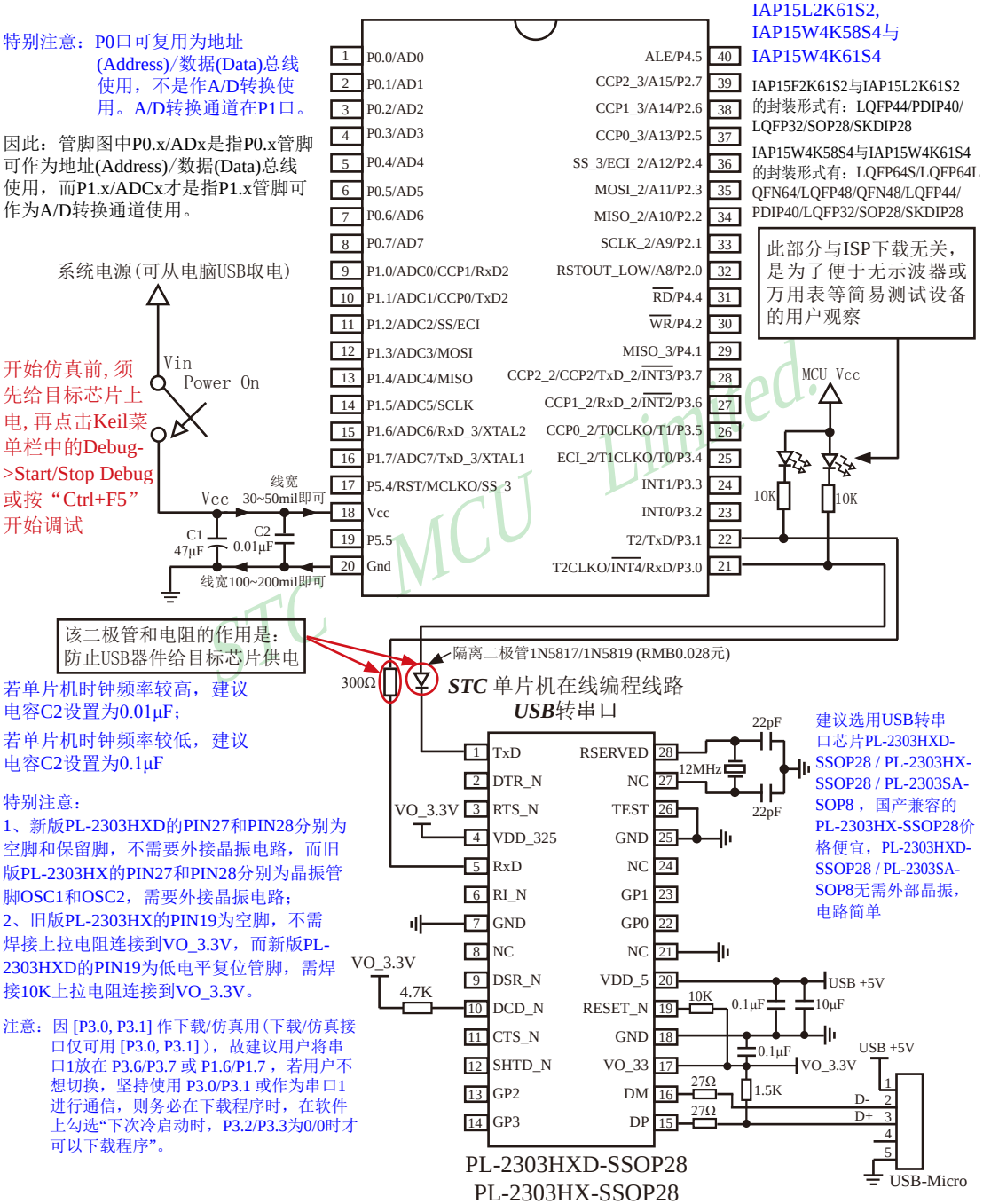
特别注意：P0口可复用为地址(Address)/数据(Data)总线使用，不是作A/D转换使用。A/D转换通道在P1口。

因此：管脚图中P0.x/ADx是指P0.x管脚可作为地址(Address)/数据(Data)总线使用，而P1.x/ADCx才是指P1.x管脚可作为A/D转换通道使用。

IAP15F2K61S2与IAP15L2K61S2的封装形式有：LQFP44/PDIP40/LQFP32/SOP28/SKDIP28

IAP15W4K58S4与IAP15W4K61S4的封装形式有：LQFP64S/LQFP64L QFN64/LQFP48/QFN48/LQFP44/PDIP40/LQFP32/SOP28/SKDIP28

此部分与ISP下载无关，是为了便于无示波器或万用表等简易测试设备的用户观察



若单片机时钟频率较高，建议电容C2设置为0.01μF；

若单片机时钟频率较低，建议电容C2设置为0.1μF

建议选用USB转串口芯片PL-2303HXD-SSOP28 / PL-2303HX-SSOP28 / PL-2303SA-SOP8，国产兼容的PL-2303HX-SSOP28价格便宜，PL-2303HXD-SSOP28 / PL-2303SA-SOP8无需外部晶振，电路简单

特别注意：

1、新版PL-2303HXD的PIN27和PIN28分别为空脚和保留脚，不需要外接晶振电路，而旧版PL-2303HX的PIN27和PIN28分别为晶振管脚OSC1和OSC2，需要外接晶振电路；

2、旧版PL-2303HX的PIN19为空脚，不需焊接上拉电阻连接到VO_3.3V，而新版PL-2303HXD的PIN19为低电平复位管脚，需焊接10K上拉电阻连接到VO_3.3V。

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”。

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)

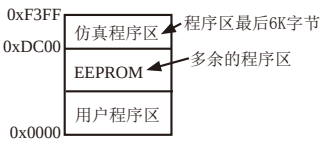
I/O : P3.0 / P3.1

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

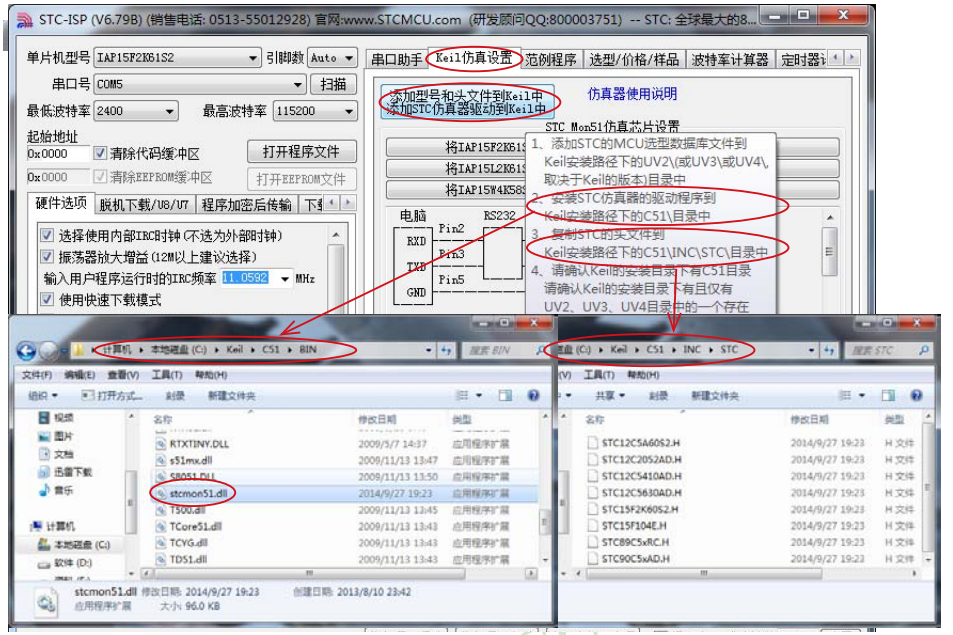
对于IAP型号单片机，对EEPROM的操作是通过对多余不用的程序区进行IAP模拟实现的，此部分要修改程序(IAP起始地址)。

如IAP15F2K61S2单片机的EEPROM区的位置如右图所示。



4. 仿真器操作步骤

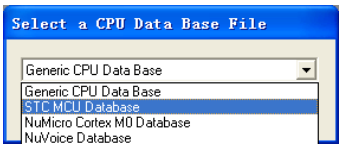
（1）安装Keil版本的仿真驱动，如下图所示:



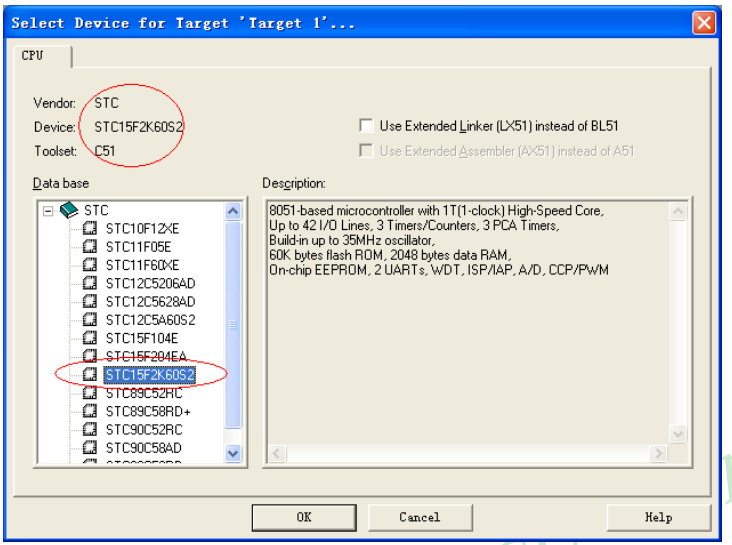
如上图，首先选择“Keil仿真设置”页面，点击“添加MCU型号到Keil中”，在出现的如下的目录选择窗口中，定位到Keil的安装目录(一般可能为“C:\Keil\”)，“确定”后出现下图中右边所示的提示信息，表示安装成功。添加头文件的同时也会安装STC的Monitor51仿真驱动STCMON51.DLL，驱动与头文件的的安装目录如上图所示。

（2）在Keil中创建项目

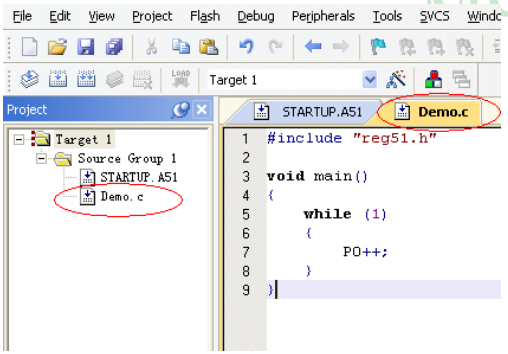
若第一步的驱动安装成功，则在Keil中新建项目时选择芯片型号时，便会有“STC MCU Database”的选择项，如下图



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



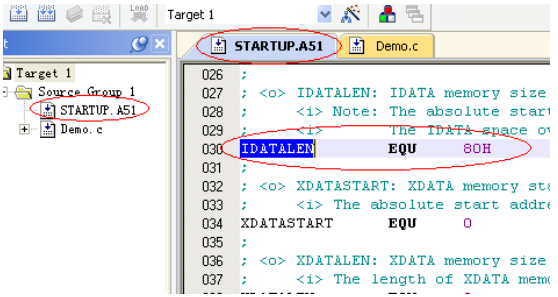
添加源代码文件到项目中，如下图：



保存项目，若编译无误，则可以进行下面的项目设置了

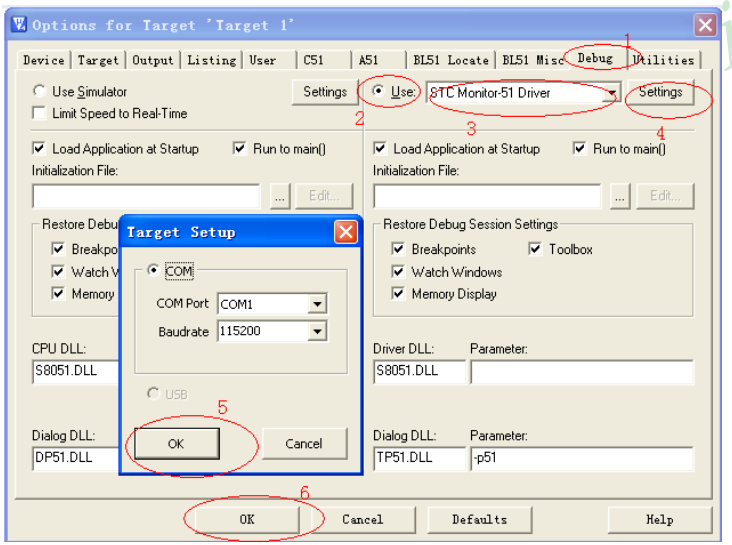
附加说明一点：

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。



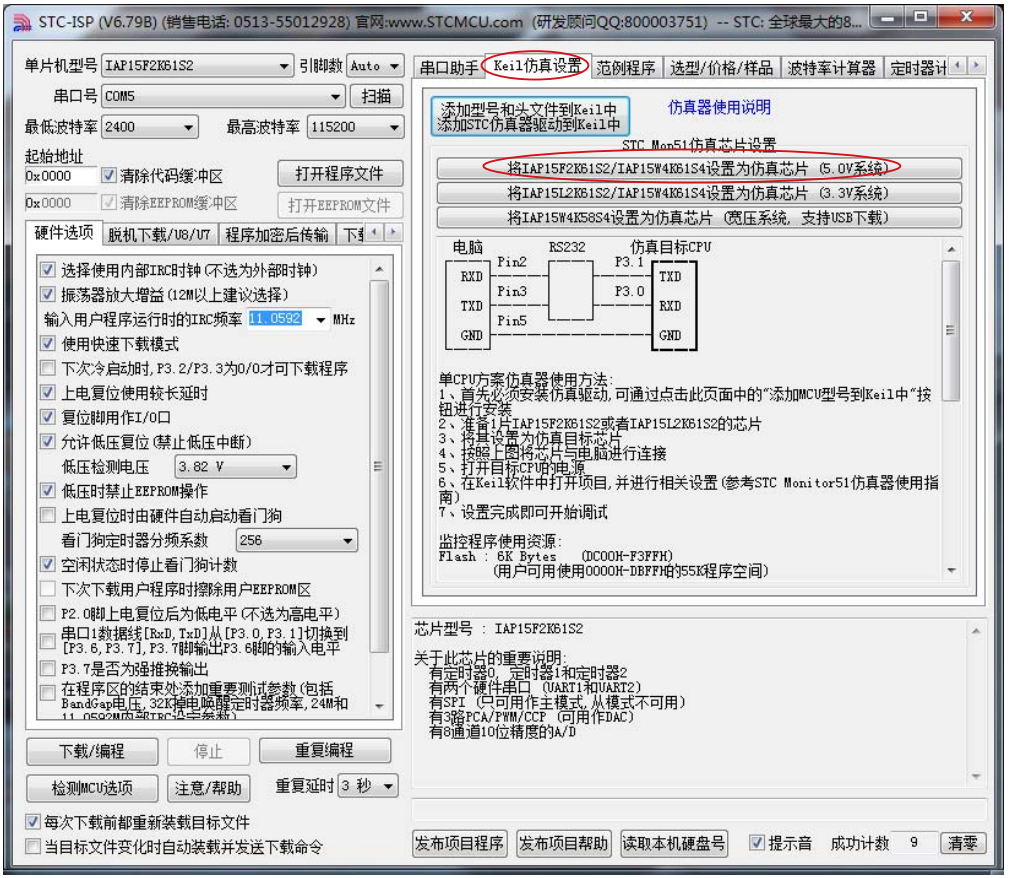
虽然STC15F2K60S2系列的单片机的IDATA大小为256字节（00-7F的DATA和80H-FFH的IDATA），但由于STC15F2K60S2在RAM的最后17个字节有写入ID号以及相关的测试参数，若用户在程序中需要使用这一部分数据，则一定不要将IDATALEN定义为256。

（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

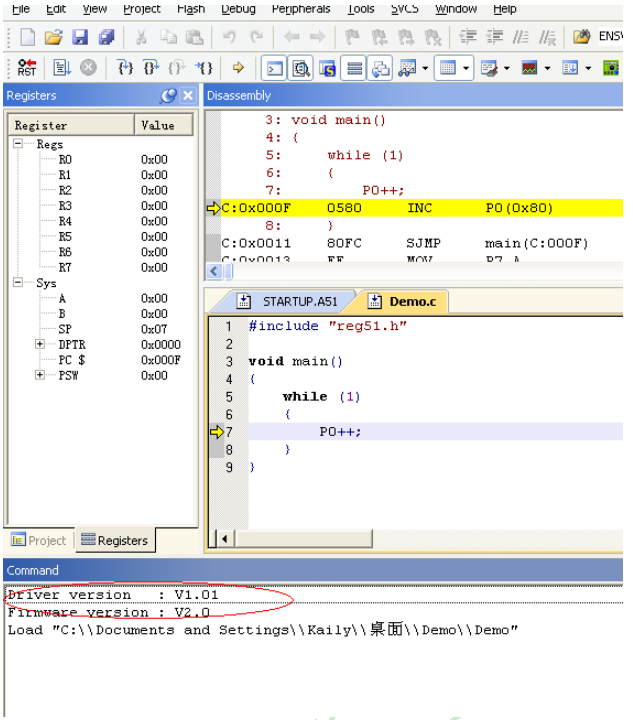
将制作完成的仿真芯片通过串口与电脑相连接，并给目标芯片上电。

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

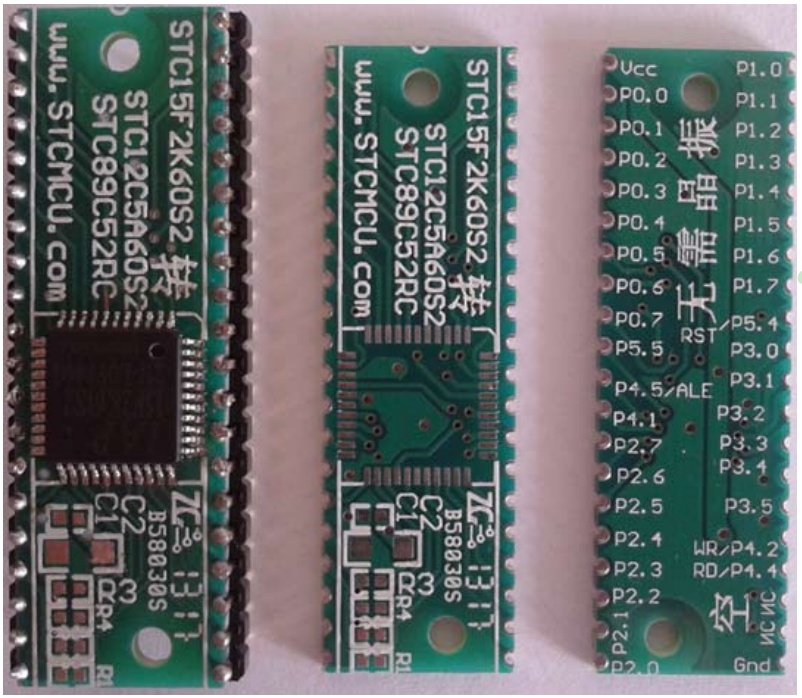
断点设置的个数目前最大允许20个（理论上可设置任意个，但是断点设置得过多会影响调试的速度）。

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


File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
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\\Kaily\\桌面\\Demo\\Demo"

10.5 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

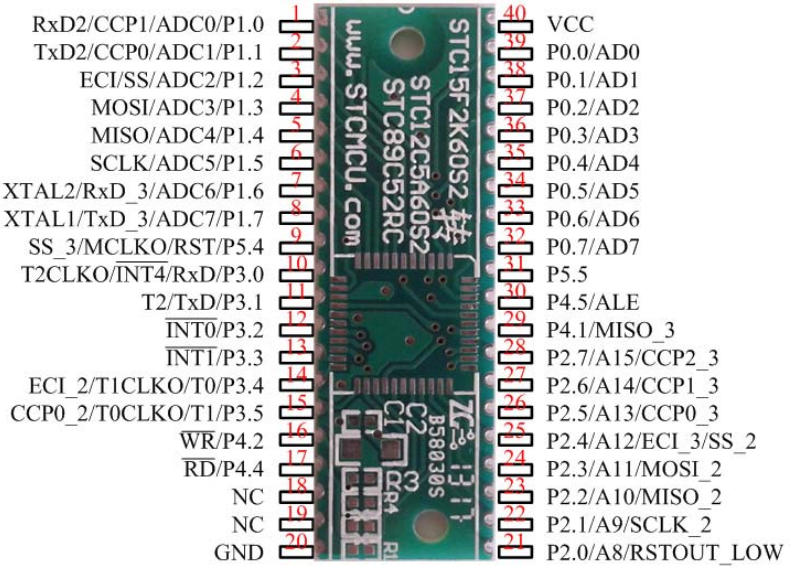
该转换板可进行IAP15F2K61S2/STC15F2K60S2转STC89C52RC/STC89C58RD+系列仿真用、IAP15F2K61S2/STC15F2K60S2转STC90C52RC/STC90C58RD+系列仿真用、IAP15F2K61S2/STC15F2K60S2转STC10F08XE/STC11F60XE系列仿真用、及IAP15F2K61S2/STC15F2K60S2转STC12C5A60S2系列仿真用。

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）



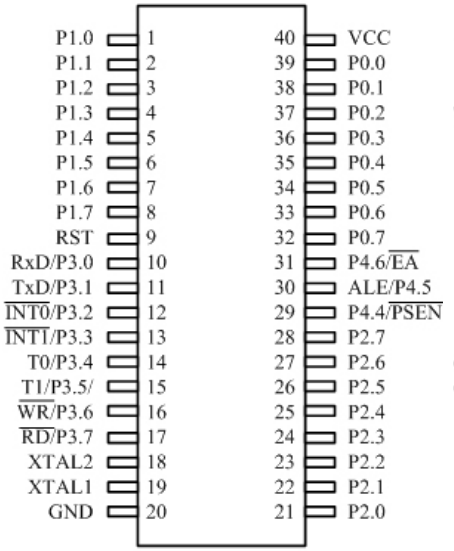
注意：

由于内置高精准R/C时钟(5MHz ~ 40MHz可设)，因此不需要外部晶振；

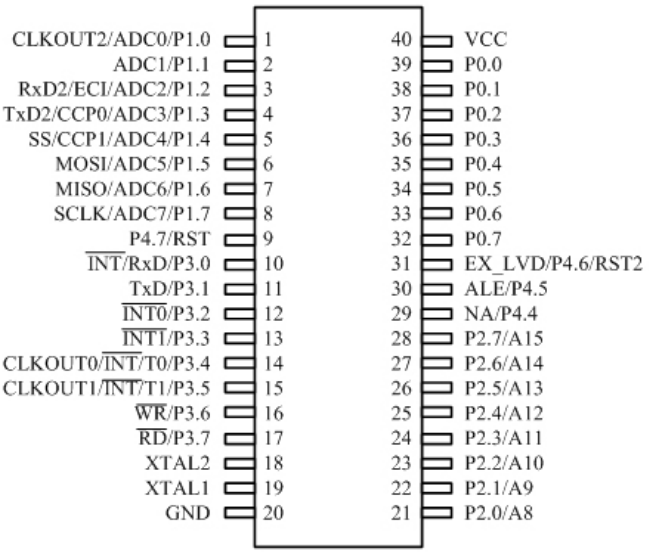
XTAL1和XTAL2是空的

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

10.6 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、STC12xx系列、STC11xx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、STC12xx系列、STC11xx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STC-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。

附录A：汇编语言编程

INTRODUCTION

Assembly language is a computer language lying between the extremes of machine language and high-level language like Pascal or C use words and statements that are easily understood by humans, although still a long way from "natural" language.Machine language is the binary language of computers.A machine language program is a series of binary bytes representing instructions the computer can execute.

Assembly language replaces the binary codes of machine language with easy to remember "mnemonics"that facilitate programming.For example, an addition instruction in machine language might be represented by the code "10110011".It might be represented in assembly language by the mnemonic "ADD".Programming with mnemonics is obviously preferable to programming with binary codes.

Of course, this is not the whole story. Instructions operate on data, and the location of the data is specified by various "addressing modes" emmbeded in the binary code of the machine language instruction. So, there may be several variations of the ADD instruction, depending on what is added. The rules for specifying these variations are central to the theme of assembly language programming.

An assembly language program is not executable by a computer. Once written, the program must undergo translation to machine language. In the example above, the mnemonic "ADD" must be translated to the binary code "10110011". Depending on the complexity of the programming environment, this translation may involve one or more steps before an executable machine language program results. As a minimum, a program called an "assembler" is required to translate the instruction mnemonics to machine language binary codes. Afurther step may require a "linker" to combine portions of program from separate files and to set the address in memory at which th program may execute. We begin with a few definitions.

An assembly language program i a program written using labels, mnemonics, and so on, in which each statement corresponds to a machine instruction. Assembly language programs, often called source code or symbolic code, cannot be executed by a computer.

A machine language program is a program containing binary codes that represent instructions to a computer. Machine language programs, often called object code, are executable by a computer.

A assembler is a program that translate an assembly language program into a machine language program. The machine language program (object code) may be in "absolute" form or in "relocatable" form. In the latter case, "linking" is required to set the absolute address for execution.

A linker is a program that combines relocatable object programs (modules) and produces an absolute object program that is executable by a computer. A linker is sometimes called a "linker/locator" to reflect its separate functions of combining relocatable modules (linking) and setting the address for execution (locating).

A segment is a unit of code or data memory. A segment may be relocatable or absolute. A relocatable segment has a name, type, and other attributes that allow the linker to combine it with other paritial segments, if required, and to correctly locate the segment. An absolute segment has no name and cannot be combined with other segments.

A module contains one or more segments or partial segments. A module has a name assigned by the user. The module definitions determine the scope of local symbols. An object file contains one or more modules. A module may be thought of as a "file" in many instances.

A program consists of a single absolute module, merging all absolute and relocatable segments from all input modules. A program contains only the binary codes for instructions (with address and data constants) that are understood by a computer.

ASSEMBLER OPERATION

There are many assembler programs and other support programs available to facilitate the development of applications for the 8051 microcontroller. Intel's original MCS-51 family assembler, ASM51, is no longer available commercially. However, it set the standard to which the others are compared.

ASM51 is a powerful assembler with all the bells and whistles. It is available on Intel development systems and on the IBM PC family of microcomputers. Since these "host" computers contain a CPU chip other than the 8051, ASM51 is called a cross assembler. An 8051 source program may be written on the host computer (using any text editor) and may be assembled to an object file and listing file (using ASM51), but the program may not be executed. Since the host system's CPU chip is not an 8051, it does not understand the binary instruction in the object file. Execution on the host computer requires either hardware emulation or software simulation of the target CPU. A third possibility is to download the object program to an 8051-based target system for execution.

ASM51 is invoked from the system prompt by

ASM51 source_file [assembler_controls]

The source file is assembled and any assembler controls specified take effect. The assembler receives a source file as input (e.g., PROGRAM.SRC) and generates an object file (PROGRAM.OBJ) and listing file (PROGRAM.LST) as output. This is illustrated in Figure 1.

Since most assemblers scan the source program twice in performing the translation to machine language, they are described as two-pass assemblers. The assembler uses a location counter as the address of instructions and the values for labels. The action of each pass is described below.

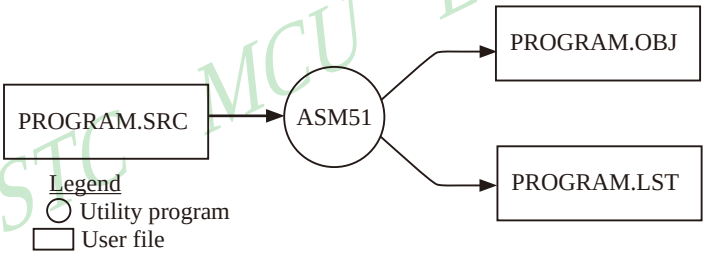


Figure 1 Assembling a source program

Pass one

During the first pass, the source file is scanned line-by-line and a symbol table is built. The location counter defaults to 0 or is set by the ORG (set origin) directive. As the file is scanned, the location counter is incremented by the length of each instruction. Define data directives (DBs or DWs) increment the location counter by the number of bytes defined. Reserve memory directives (DSs) increment the location counter by the number of bytes reserved.

Each time a label is found at the beginning of a line, it is placed in the symbol table along with the current value of the location counter. Symbols that are defined using equate directives (EQUs) are placed in the symbol table along with the "equated" value. The symbol table is saved and then used during pass two.

Pass two

During pass two, the object and listing files are created. Mnemonics are converted to opcodes and placed in the output files. Operands are evaluated and placed after the instruction opcodes. Where symbols appear in the operand field, their values are retrieved from the symbol table (created during pass one) and used in calculating the correct data or addresses for the instructions.

Since two passes are performed, the source program may use "forward references", that is, use a symbol before it is defined. This would occur, for example, in branching ahead in a program.

The object file, if it is absolute, contains only the binary bytes (00H-0FH) of the machine language program. A relocatable object file will also contain a sysmbol table and other information required for linking and locating. The listing file contains ASCII text codes (02H-7EH) for both the source program and the hexadecimal bytes in the machine language program.

A good demonstration of the distinction between an object file and a listing file is to display each on the host computer's CRT display (using, for example, the TYPE command on MS-DOS systems). The listing file clearly displays, with each line of output containing an address, opcode, and perhaps data, followed by the program statement from the source file. The listing file displays properly because it contains only ASCII text codes. Displaying the object file is a problem, however. The output will appear as "garbage", since the object file contains binary codes of an 8051 machine language program, rather than ASCII text codes.

ASSEMBLY LANGUAGE PROGRAM FORMAT

Assembly language programs contain the following:

Machine instructions
Assembler directives
Assembler controls
Comments

Machine instructions are the familiar mnemonics of executable instructions (e.g., ANL). Assembler directives are instructions to the assembler program that define program structure, symbols, data, constants, and so on (e.g., ORG). Assembler controls set assembler modes and direct assembly flow (e.g., $TITLE). Comments enhance the readability of programs by explaining the purpose and operation of instruction sequences.

Those lines containing machine instructions or assembler directives must be written following specific rules understood by the assembler. Each line is divided into "fields" separated by space or tab characters. The general format for each line is as follows:

[label:] mnemonic [operand] [, operand] […] [;commernt]

Only the mnemonic field is mandatory. Many assemblers require the label field, if present, to begin on the left in column 1, and subsequent fields to be separated by space or tab charecters. With ASM51, the label field needn't begin in column 1 and the mnemonic field needn't be on the same line as the label field. The operand field must, however, begin on the same line as the mnemonic field. The fields are described below.

Label Field

A label represents the address of the instruction (or data) that follows. When branching to this instruction, this label is usded in the operand field of the branch or jump instruction (e.g., SJMP SKIP).

Whereas the term "label" always represents an address, the term "symbol" is more general. Labels are one type of symbol and are identified by the requirement that they must terminate with a colon(:). Symbols are assigned values or attributes, using directives such as EQU, SEGMENT, BIT, DATA, etc. Symbols may be addresses, data constants, names of segments, or other constructs conceived by the programmer. Symbols do not terminate with a colon. In the example below, PAR is a symbol and START is a label (which is a type of symbol).

```
PAR     EQU   500                ;"PAR" IS A SYMBOL WHICH
                                 ;REPRESENTS THE VALUE 500
START:  MOV   A,      #0FFH      ;"START" IS A LABEL WHICH
                                 ;REPRESENTS THE ADDRESS OF
                                 ;THE MOV INSTRUCTION
```

A symbol (or label) must begin with a letter, question mark, or underscore (_); must be followed by letters, digit, "?", or "_"; and can contain up to 31 characters. Symbols may use upper- or lowercase characters, but they are treated the same. Reserved words (mnemonics, operators, predefined symbols, and directives) may not be used.

Mnemonic Field

Intruction mnemonics or assembler directives go into mnemonic field, which follows the label field. Examples of instruction mnemonics are ADD, MOV, DIV, or INC. Examples of assembler directives are ORG, EQU, or DB.

Operand Field

The operand field follows the mnemonic field. This field contains the address or data used by the instruction. A label may be used to represent the address of the data, or a symbol may be used to represent a data constant. The possibilities for the operand field are largely dependent on the operation. Some operations have no operand (e.g., the RET instruction), while others allow for multiple operands separated by commas. Indeed, the possibilties for the operand field are numberous, and we shall elaborate on these at length. But first, the comment field.

Comment Field

Remarks to clarify the program go into comment field at the end of each line. Comments must begin with a semicolon (;). Each lines may be comment lines by beginning them with a semicolon. Subroutines and large sections of a program generally begin with a comment block—serveral lines of comments that explain the general properties of the section of software that follows.

Special Assembler Symbols

Special assembler symbols are used for the register-specific addressing modes. These include A, R0 through R7, DPTR, PC, C and AB. In addition, a dollar sign ($) can be used to refer to the current value of the location counter. Some examples follow.

```
        SETB    C
        INC     DPTR
        JNB     TI , $
```

The last instruction above makes effective use of ASM51's location counter to avoid using a label. It could also be written as

```
HERE:   JNB     TI , HERE
```

Indirect Address

For certain instructions, the operand field may specify a register that contains the address of the data. The commercial "at" sign (@) indicates address indirection and may only be used with R0, R1, the DPTR, or the PC, depending on the instruction. For example,

```
        ADD     A , @R0
        MOVC    A , @A+PC
```

The first instruction above retrieves a byte of data from internal RAM at the address specified in R0. The second instruction retrieves a byte of data from external code memory at the address formed by adding the contents of the accumulator to the program counter. Note that the value of the program counter, when the add takes place, is the address of the instruction following MOVC. For both instruction above, the value retrieved is placed into the accumulator.

Immediate Data

Instructions using immediate addressing provide data in the operand field that become part of the instruction. Immediate data are preceded with a pound sign (#). For example,

```
CONSTANT    EQU     100
            MOV     A ,     #0FEH
            ORL     40H ,   #CONSTANT
```

All immediate data operations (except MOV DPTR,#data) require eight bits of data. The immediate data are evaluated as a 16-bit constant, and then the low-byte is used. All bits in the high-byte must be the same (00H or FFH) or the error message "value will not fit in a byte" is generated. For example, the following instructions are syntactically correct:

```
MOV     A ,     #0FF00H
MOV     A ,     #00FFH
```

But the following two instructions generate error messages:

```
MOV     A ,     #0FE00H
MOV     A ,     #01FFH
```

If signed decimal notation is used, constants from -256 to +255 may also be used. For example, the following two instructions are equivalent (and syntactically correct):

```
MOV     A ,     #-256
MOV     A ,     #0FF00H
```

Both instructions above put 00H into accumulator A.

Data Address

Many instructions access memory locations using direct addressing and require an on-chip data memory address (00H to 7FH) or an SFR address (80H to 0FFH) in the operand field. Predefined symbols may be used for the SFR addresses. For example,

```
MOV     A ,     45H
MOV     A ,     SBUF            ;SAME AS MOV A, 99H
```

Bit Address

One of the most powerful features of the 8051 is the ability to access individual bits without the need for masking operations on bytes. Instructions accessing bit-addressable locations must provide a bit address in internal data memory (00h to 7FH) or a bit address in the SFRs (80H to 0FFH).

There are three ways to specify a bit address in an instruction: (a) explicitly by giving the address, (b) using the dot operator between the byte address and the bit position, and (c) using a predefined assembler symbol. Some examples follow.

```
SETB    0E7H            ;EXPLICIT BIT ADDRESS
SETB    ACC.7           ;DOT OPERATOR (SAME AS ABOVE)
JNB     TI ,    $       ;"TI" IS A PRE-DEFINED SYMBOL
JNB     99H,    $       ;(SAME AS ABOVE)
```

Code Address

A code address is used in the operand field for jump instructions, including relative jumps (SJMP and conditional jumps), absolute jumps and calls (ACALL, AJMP), and long jumps and calls (LJMP, LCALL).

The code address is usually given in the form of a label.

ASM51 will determine the correct code address and insert into the instruction the correct 8-bit signed offset, 11-bit page address, or 16-bit long address, as appropriate.

Generic Jumps and Calls

ASM51 allows programmers to use a generic JMP or CALL mnemonic. "JMP" can be used instead of SJMP, AJMP or LJMP; and "CALL" can be used instead of ACALL or LCALL. The assembler converts the generic mnemonic to a "real" instruction following a few simple rules. The generic mnemonic converts to the short form (for JMP only) if no forward references are used and the jump destination is within -128 locations, or to the absolute form if no forward references are used and the instruction following the JMP or CALL instruction is in the same 2K block as the destination instruction. If short or absolute forms cannot be used, the conversion is to the long form.

The conversion is not necessarily the best programming choice. For example, if branching ahead a few instrucions, the generic JMP will always convert to LJMP even though an SJMP is probably better. Consider the following assembled instructions sequence using three generic jumps.

```
LOC     OBJ       LINE     SOURCE
1234              1                    ORG     1234H
1234    04        2        START:      INC     A
1235    80FD      3                    JMP     START          ;ASSEMBLES AS SJMP
12FC              4                    ORG     START + 200
12FC    4134      5                    JMP     START          ;ASSEMBLES AS AJMP
12FE    021301    6                    JMP     FINISH         ;ASSEMBLES AS LJMP
1301    04        7        FINISH:     INC     A
                  8                    END
```

The first jump (line 3) assembles as SJMP because the destination is before the jump (i.e., no forward reference) and the offset is less than -128. The ORG directive in line 4 creates a gap of 200 locations between the label START and the second jump, so the conversion on line 5 is to AJMP because the offset is too great for SJMP. Note also that the address following the second jump (12FEH) and the address of START (1234H) are within the same 2K page, which, for this instruction sequence, is bounded by 1000H and 17FFH. This criterion must be met for absolute addressing. The third jump assembles as LJMP because the destination (FINISH) is not yet defined when the jump is assembled (i.e., a forward reference is used). The reader can verify that the conversion is as stated by examining the object field for each jump instruction.

ASSEMBLE-TIME EXPRESSION EVALUATION

Values and constants in the operand field may be expressed three ways: (a) explicitly (e.g.,0EFH), (b) with a predefined symbol (e.g., ACC), or (c) with an expression (e.g.,2 + 3). The use of expressions provides a powerful technique for making assembly language programs more readable and more flexible. When an expression is used, the assembler calculates a value and inserts it into the instruction.

All expression calculations are performed using 16-bit arithmetic; however, either 8 or 16 bits are inserted into the instruction as needed. For example, the following two instructions are the same:

```
MOV     DPTR,   #04FFH + 3
MOV     DPTR,   #0502H          ;ENTIRE 16-BIT RESULT USED
```

If the same expression is used in a "MOV A,#data" instruction, however, the error message "value will not fit in a byte" is generated by ASM51. An overview of the rules for evaluateing expressions follows.

Number Bases

The base for numeric constants is indicated in the usual way for Intel microprocessors. Constants must be followed with "B" for binary, "O" or "Q" for octal, "D" or nothing for decimal, or "H" for hexadecimal. For example, the following instructions are the same:

```
MOV     A , #15H
MOV     A , #1111B
MOV     A , #0FH
MOV     A , #17Q
MOV     A , #15D
```

Note that a digit must be the first character for hexadecimal constants in order to differentiate them from labels (i.e., "0A5H" not "A5H").

Charater Strings

Strings using one or two characters may be used as operands in expressions. The ASCII codes are converted to the binary equivalent by the assembler. Character constants are enclosed in single quotes ('). Some examples follow.

```
CJNE    A ,  # 'Q', AGAIN
SUBB    A ,  # '0'                ;CONVERT ASCII DIGIT TO BINARY DIGIT
MOV     DPTR, # 'AB'
MOV     DPTR, #4142H               ;SAME AS ABOVE
```

Arithmetic Operators

The arithmetic operators are

+	addition
-	subtraction
*	multiplication
/	division
MOD	modulo (remainder after division)

For example, the following two instructions are same:

```
MOV     A,    10 +10H
MOV     A,    #1AH
```

The following two instructions are also the same:

```
MOV     A,    #25 MOD 7
MOV     A,    #4
```

Since the MOD operator could be confused with a symbol, it must be seperated from its operands by at least one space or tab character, or the operands must be enclosed in parentheses. The same applies for the other operators composed of letters.

Logical Operators

The logical operators are

OR	logical OR
AND	logical AND
XOR	logical Exclusive OR
NOT	logical NOT (complement)

The operation is applied on the corresponding bits in each operand. The operator must be separated from the operands by space or tab characters. For example, the following two instructions are the same:

```
MOV     A, # '9'   AND   0FH
MOV     A, #9
```

The NOT operator only takes one operand. The following three MOV instructions are the same:

```
THREE           EQU     3
MINUS_THREE     EQU     -3
                MOV     A,      # (NOT THREE) + 1
                MOV     A,      #MINUS_THREE
                MOV     A,      #11111101B
```

Special Operators

The sepcial operators are

SHR	shift right
SHL	shift left
HIGH	high-byte
LOW	low-byte
()	evaluate first

For example, the following two instructions are the same:

```
MOV     A, #8  SHL  1
MOV     A, #10H
```

The following two instructions are also the same:

```
MOV     A, #HIGH  1234H
MOV     A, #12H
```


Relational Operators

When a relational operator is used between two operands, the result is alwalys false (0000H) or true (FFFFH). The operators are

EQ	=	equals
NE	< >	not equals
LT	<	less than
LE	<=	less than or equal to
GT	>	greater than
GE	>=	greater than or equal to

Note that for each operator, two forms are acceptable (e.g., "EQ" or "="). In the following examples, all relational tests are "true":

```
MOV     A, #5 = 5
MOV     A,#5 NE 4
MOV     A,# 'X'  LT  'Z'
MOV     A,# 'X'  >=  'X'
MOV     A,#$ > 0
MOV     A,#100 GE 50
```

So, the assembled instructions are equal to

```
MOV     A, #0FFH
```

Even though expressions evaluate to 16-bit results (i.e., 0FFFFH), in the examples above only the low-order eight bits are used, since the instruction is a move byte operation. The result is not considered too big in this case, because as signed numbers the 16-bit value FFFFH and the 8-bit value FFH are the same (-1).

Expression Examples

The following are examples of expressions and the values that result:

Expression	Result
'B' - 'A'	0001H
8/3	0002H
155 MOD 2	0001H
4 * 4	0010H
8 AND 7	0000H
NOT 1	FFFEH
'A' SHL 8	4100H
LOW 65535	00FFH
(8 + 1) * 2	0012H
5 EQ 4	0000H
'A' LT 'B'	FFFFH
3 <= 3	FFFFHss

A practical example that illustrates a common operation for timer initialization follows: Put -500 into Timer 1 registers TH1 and TL1. In using the HIGH and LOW operators, a good approach is

```
VALUE       EQU     -500
            MOV     TH1, #HIGH  VALUE
            MOV     TL1, #LOW VALUE
```

The assembler converts -500 to the corresponding 16-bit value (FE0CH); then the HIGH and LOW operators extract the high (FEH) and low (0CH) bytes. as appropriate for each MOV instruction.

Operator Precedence

The precedence of expression operators from highest to lowest is

```
( )
HIGH LOW
* / MOD SHL SHR
+ -
EQ NE LT LE GT GE = <> < <= > >=
NOT
AND
OR XOR
```

When operators of the same precedence are used, they are evaluated left to right.
Examples:

Expression	Value
HIGH ('A' SHL 8)	0041H
HIGH 'A' SHL 8	0000H
NOT 'A' - 1	FFBFH
'A' OR 'A' SHL 8	4141H

ASSEMBLER DIRECTIVES

Assembler directives are instructions to the assembler program. They are not assembly language instructions executable by the target microprocessor. However, they are placed in the mnemonic field of the program. With the exception of DB and DW, they have no direct effect on the contents of memory.

ASM51 provides several catagories of directives:

Assembler state control (ORG, END, USING)
Symbol definition (SEGMENT, EQU, SET, DATA, IDATA, XDATA, BIT, CODE)
Storage initialization/reservation (DS, DBIT, DB, DW)
Program linkage (PUBLIC, EXTRN,NAME)
Segment selection (RSEG, CSEG, DSEG, ISEG, ESEG, XSEG)

Each assembler directive is presented below, ordered by catagory.

Assembler State Control

ORG (Set Origin)　The format for the ORG (set origin) directive is

```
ORG     expression
```

The ORG directive alters the location counter to set a new program origin for statements that follow. A label is not permitted. Two examples follow.

```
ORG     100H                    ;SET LOCATION COUNTER TO 100H
ORG     ($ + 1000H) AND 0F00H   ;SET TO NEXT 4K BOUNDARY
```

The ORG directive can be used in any segment type. If the current segment is absolute, the value will be an absolute address in the current segment. If a relocatable segment is active, the value of the ORG expression is treated as an offset from the base address of the current instance of the segment.

End　The format of the END directive is

```
END
```

END should be the last statement in the source file. No label is permitted and nothing beyond the END statement is processed by the assembler.

Using　The format of the END directive is

```
USING   expression
```

This directive informs ASM51 of the currently active register bank. Subsequent uses of the predefined symbolic register addresses AR0 to AR7 will convert to the appropriate direct address for the active register bank. Consider the following sequence:

```
USING   3
PUSH    AR7
USING   1
PUSH    AR7
```

The first push above assembles to PUSH 1FH (R7 in bank 3), whereas the second push assembles to PUSH 0FH (R7 in bank 1).

Note that USING does not actually switch register banks; it only informs ASM51 of the active bank. Executing 8051 instructions is the only way to switch register banks. This is illustrated by modifying the example above as follows:

```
MOV     PSW, #00011000B         ;SELECT REGISTER BANK 3
USING   3
PUSH    AR7                     ;ASSEMBLE TO PUSH 1FH
MOV     PSW, #00001000B         ;SELECT REGISTER BANK 1
USING   1
PUSH    AR7                     ;ASSEMBLE TO PUSH 0FH
```

Symbol Definition

The symbol definition directives create symbols that represent segment, registers, numbers, and addresses. None of these directives may be preceded by a label. Symbols defined by these directives may not have been previously defined and may not be redefined by any means. The SET directive is the only exception. Symbol definiton directives are described below.

Segment The format for the SEGMENT directive is shown below.

```
symbol          SEGMENT         segment_type
```

The symbol is the name of a relocatable segment. In the use of segments, ASM51 is more complex than conventional assemblers, which generally support only "code" and "data" segment types. However, ASM51 defines additional segment types to accommodate the diverse memory spaces in the 8051. The following are the defined 8051 segment types (memory spaces):

CODE (the code segment)
XDATA (the external data space)
DATA (the internal data space accessible by direct addressing, 00H–07H)
IDATA (the entire internal data space accessible by indirect addressing, 00H–07H)
BIT (the bit space; overlapping byte locations 20H–2FH of the internal data space)

For example, the statement

```
EPROM           SEGMENT         CODE
```

declares the symbol EPROM to be a SEGMENT of type CODE. Note that this statement simply declares what EPROM is. To actually begin using this segment, the RSEG directive is used (see below).

EQU (Equate) The format for the EQU directive is

```
Symbol          EQU     expression
```

The EQU directive assigns a numeric value to a specified symbol name. The symbol must be a valid symbol name, and the expression must conform to the rules described earlier.

The following are examples of the EQU directive:

```
N27             EQU     27              ;SET N27 TO THE VALUE 27
HERE            EQU     $               ;SET "HERE" TO THE VALUE OF
                                        ;THE LOCATION COUNTER
CR              EQU     0DH             ;SET CR (CARRIAGE RETURN) TO 0DH
MESSAGE:        DB 'This is a message'
LENGTH          EQU     $ - MESSAGE     ;"LENGTH" EQUALS LENGTH OF "MESSAGE"
```

Other Symbol Definition Directives The SET directive is similar to the EQU directive except the symbol may be redefined later, using another SET directive.

The DATA, IDATA, XDATA, BIT, and CODE directives assign addresses of the corresponding segment type to a symbol. These directives are not essential. A similar effect can be achieved using the EQU directive; if used, however, they evoke powerful type-checking by ASM51. Consider the following two directives and four instructions:

```
FLAG1       EQU     05H
FLAG2       BIT     05H
            SETB    FLAG1
            SETB    FLAG2
            MOV     FLAG1, #0
            MOV     FLAG2, #0
```

The use of FLAG2 in the last instruction in this sequence will generate a "data segment address expected" error message from ASM51. Since FLAG2 is defined as a bit address (using the BIT directive), it can be used in a set bit instruction, but it cannot be used in a move byte instruction. Hence, the error. Even though FLAG1 represents the same value (05H), it was defined using EQU and does not have an associated address space. This is not an advantage of EQU, but rather, a disadvantage. By properly defining address symbols for use in a specific memory space (using the directives BIT, DATA, XDATA,ect.), the programmer takes advantage of ASM51's powerful type-checking and avoids bugs from the misuse of symbols.

Storage Initialization/Reservation

The storage initialization and reservation directives initialize and reserve space in either word, byte, or bit units. The space reserved starts at the location indicated by the current value of the location counter in the currently active segment. These directives may be preceded by a label. The storage initialization/reservation directives are described below.

DS (Define Storage) The format for the DS (define storage) directive is

```
[label:]    DS      expression
```

The DS directive reserves space in byte units. It can be used in any segment type except BIT. The expression must be a valid assemble-time expression with no forward references and no relocatable or external references. When a DS statement is encountered in a program, the location counter of the current segment is incremented by the value of the expression. The sum of the location counter and the specified expression should not exceed the limitations of the current address space.

The following statement create a 40-byte buffer in the internal data segment:

```
            DSEG    AT      30H     ;PUT IN DATA SEGMENT (ABSOLUTE, INTERNAL)
LENGTH      EQU     40
BUFFER:     DS      LENGRH          ;40 BYTES RESERVED
```

The label BUFFER represents the address of the first location of reserved memory. For this example, the buffer begins at address 30H because "AT 30H" is specified with DSEG. The buffer could be cleared using the following instruction sequence:

```
            MOV     R7,     #LENGTH
            MOV     R0,     #BUFFER
LOOP:       MOV     @R0,    #0
            DJNZ    R7,     LOOP
            (continue)
```

To create a 1000-byte buffer in external RAM starting at 4000H, the following directives could be used:

```
XSTART      EQU     4000H
XLENGTH     EQU     1000
            XSEG      AT    XSTART
XBUFFER:    DS  XLENGTH
```

This buffer could be cleared with the following instruction sequence:

```
            MOV     DPTR, #XBUFFER
LOOP:       CLR     A
            MOVX    @DPTR, A
            INC     DPTR
            MOV     A,      DPL
            CJNE    A,      #LOW (XBUFFER + XLENGTH + 1), LOOP
            MOV     A,      DPH
            CJNE    A,      #HIGH (XBUFFER + XLENGTH + 1), LOOP
            (continue)
```

This is an excellent example of a powerful use of ASM51's operators and assemble-time expressions. Since an instruction does not exist to compare the data pointer with an immediate value, the operation must be fabricated from available instructions. Two compares are required, one each for the high- and low-bytes of the DPTR. Furthermore, the compare-and-jump-if-not-equal instruction works only with the accumulator or a register, so the data pointer bytes must be moved into the accumulator before the CJNE instruction. The loop terminates only when the data pointer has reached XBUFFER + LENGTH + 1. (The "+1" is needed because the data pointer is incremented after the last MOVX instruction.)

DBIT The format for the DBIT (define bit) directive is,

```
[label:]    DBIT    expression
```

The DBIT directive reserves space in bit units. It can be used only in a BIT segment. The expression must be a valid assemble-time expression with no forward references. When the DBIT statement is encountered in a program, the location counter of the current (BIT) segment is incremented by the value of the expression. Note that in a BIT segment, the basic unit of the location counter is bits rather than bytes. The following directives creat three flags in a absolute bit segment:

```
            BSEG            ;BIT SEGMENT (ABSOLUTE)
KEFLAG:     DBIT    1       ;KEYBOARD STATUS
PRFLAG:     DBIT    1       ;PRINTER STATUS
DKFLAG:     DBIT    1       ;DISK STATUS
```

Since an address is not specified with BSEG in the example above, the address of the flags defined by DBIT could be determined (if one wishes to to so) by examining the symbol table in the .LST or .M51 files. If the definitions above were the first use of BSEG, then KBFLAG would be at bit address 00H (bit 0 of byte address 20H). If other bits were defined previously using BSEG, then the definitions above would follow the last bit defined.

DB (Define Byte) The format for the DB (define byte) directive is,

```
[label:]    DB      expression  [, expression]  […]
```

The DB directive initializes code memory with byte values. Since it is used to actually place data constants in code memory, a CODE segment must be active. The expression list is a series of one or more byte values (each of which may be an expression) separated by commas.

The DB directive permits character strings (enclosed in single quotes) longer than two characters as long as they are not part of an expression. Each character in the string is converted to the corresponding ASCII code. If a label is used, it is assigned the address of th first byte. For example, the following statements

```
        CSEG  AT      0100H
SQUARES:  DB    0, 1, 4, 9, 16, 25          ;SQUARES OF NUMBERS 0-5
MESSAGE:  DB    'Login:', 0                 ;NULL-TERMINATED CHARACTER STRING
```

When assembled, result in the following hexadecimal memory assignments for external code memory:

Address	Contents
0100	00
0101	01
0102	04
0103	09
0104	10
0105	19
0106	4C
0107	6F
0108	67
0109	69
010A	6E
010B	3A
010C	00

DW (Define Word)　The format for the DW (define word) directive is

```
[label:]    DW      expression      [, expression]  […]
```

The DW directive is the same as the DB directive except two memory locations (16 bits) are assigned for each data item. For example, the statements

```
CSEG    AT      200H
DW      $, 'A', 1234H, 2, 'BC'
```

result in the following hexadecimal memory assignments:

Address	Contents
0200	02
0201	00
0202	00
0203	41
0204	12
0205	34
0206	00
0207	02
0208	42
0209	43

Program Linkage

Program linkage directives allow the separately assembled modules (files) to communicate by permitting inter-module references and the naming of modules. In the following discussion, a "module" can be considered a "file." (In fact, a module may encompass more than one file.)

Public The format for the PUBLIC (public symbol) directive is

```
PUBLIC      symbol  [, symbol]  […]
```

The PUBLIC directive allows the list of specified symbols to known and used outside the currently assembled module. A symbol declared PUBLIC must be defined in the current module. Declaring it PUBLIC allows it to be referenced in another module. For example,

```
PUBLIC  INCHAR, OUTCHR, INLINE, OUTSTR
```

Extrn The format for the EXTRN (external symbol) directive is

```
EXTRN       segment_type  (symbol  [, symbol]  […], …)
```

The EXTRN directive lists symbols to be referenced in the current module that are defined in other modules. The list of external symbols must have a segment type associated with each symbol in the list. (The segment types are CODE, XDATA, DATA, IDATA, BIT, and NUMBER. NUMBER is a type-less symbol defined by EQU.) The segment type indicates the way a symbol may be used. The information is important at link-time to ensure symbols are used properly in different modules.

The PUBLIC and EXTRN directives work together. Consider the two files, MAIN.SRC and MESSAGES.SRC. The subroutines HELLO and GOOD_BYE are defined in the module MESSAGES but are made available to other modules using the PUBLIC directive. The subroutines are called in the module MAIN even though they are not defined there. The EXTRN directive declares that these symbols are defined in another module.

MAIN.SRC:

```
            EXTRN       CODE (HELLO, GOOD_BYE)
            …
            CALL        HELLO
            …
            CALL        GOOD_BYE
            …
            END
```

MESSAGES.SRC:

```
            PUBLIC      HELLO, GOOD_BYE
            …
HELLO:      (begin subroutine)
            …
            RET
GOOD_BYE:   (begin subroutine)
            …
            RET
            …
            END
```

Neither MAIN.SRC nor MESSAGES.SRC is a complete program; they must be assembled separately and linked together to form an executable program. During linking, the external references are resolved with correct addresses inserted as the destination for the CALL instructions.

Name The format for the NAME directive is

```
NAME    module_name
```

All the usual rules for symbol names apply to module names. If a name is not provided, the module takes on the file name (without a drive or subdirectory specifier and without an extension). In the absence of any use of the NAME directive, a program will contain one module for each file. The concept of "modules," therefore, is somewhat cumbersome, at least for relatively small programming problems. Even programs of moderate size (encompassing, for example, several files complete with relocatable segments) needn't use the NAME directive and needn't pay any special attention to the concept of "modules." For this reason, it was mentioned in the definition that a module may be considered a "file," to simplify learning ASM51. However, for very large programs (several thousand lines of code, or more), it makes sense to partition the problem into modules, where, for example, each module may encompass several files containing routines having a common purpose.

Segment Selection Directives

When the assembler encounters a segment selection directive, it diverts the following code or data into the selected segment until another segment is selected by a segment selection directive. The directive may select may select a previously defined relocatable segment or optionally create and select absolute segments.

RSEG (Relocatable Segment) The format for the RSEG (relocatable segment) directive is

```
RSEG        segment_name
```

Where "segment_name" is the name of a relocatable segment previously defined with the SEGMENT directive. RSEG is a "segment selection" directive that diverts subsequent code or data into the named segment until another segment selection directive is encountered.

Selecting Absolute Segments RSEG selects a relocatable segment. An "absolute" segment, on the other hand, is selected using one of the directives:

```
CSEG    (AT  address)
DSEG    (AT  address)
ISEG    (AT  address)
BSEG    (AT  address)
XSEG    (AT  address)
```

These directives select an absolute segment within the code, internal data, indirect internal data, bit, or external data address spaces, respectively. If an absolute address is provided (by indicating "AT address"), the assembler terminates the last absolute address segment, if any, of the specified segment type and creates a new absolute segment starting at that address. If an absolute address is not specified, the last absolute segment of the specified type is continued. If no absolute segment of this type was previously selected and the absolute address is omitted, a new segment is created starting at location 0. Forward references are not allowed and start addresses must be absolute.

Each segment has its own location counter, which is always set to 0 initially. The default segment is an absolute code segment; therefore, the initial state of the assembler is location 0000H in the absolute code segment. When another segment is chosen for the first time, the location counter of the former segment retains the last active value. When that former segment is reselected, the location counter picks up at the last active value. The ORG directive may be used to change the location counter within the currently selected segment.

ASSEMBLER CONTROLS

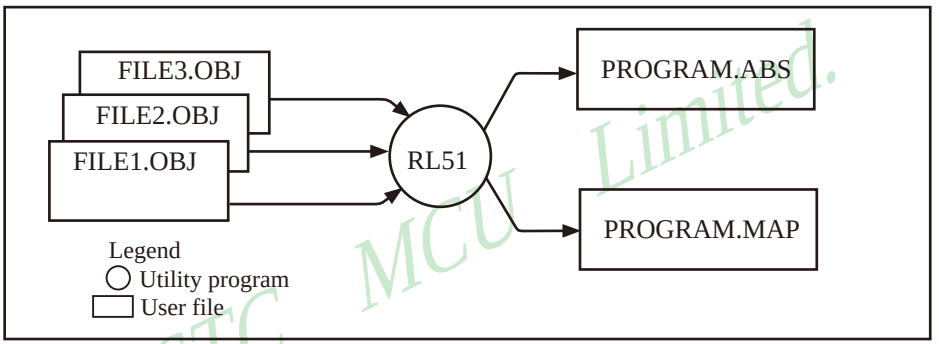
Assembler controls establish the format of the listing and object files by regulating the actions of ASM51. For the most part, assembler controls affect the look of the listing file, without having any affect on the program itself. They can be entered on the invocation line when a program is assembled, or they can be placed in the source file. Assembler controls appearing in the source file must be preceded with a dollor sign and must begin in column 1.

There are two categories of assembler controls: primary and general. Primary controls can be placed in the invocation line or at the beginnig of the source program. Only other primary controls may precede a primary control. General controls may be placed anywhere in the source program.

LINKER OPERATION

In developing large application programs, it is common to divide tasks into subprograms or modules containing sections of code (usually subroutines) that can be written separately from the overall program. The term "modular programming" refers to this programming strategy. Generally, modules are relocatable, meaning they are not intended for a specific address in the code or data space. A linking and locating program is needed to combine the modules into one absolute object module that can be executed.

Intel's RL51 is a typical linker/locator. It processes a series of relocatable object modules as input and creates an executable machine language program (PROGRAM, perhaps) and a listing file containing a memory map and symbol table (PROGRAM.M51). This is illustrated in following figure.



Linker operation

As relocatable modules are combined, all values for external symbols are resolved with values inserted into the output file. The linker is invoked from the system prompt by

RL51 input_list [T0 output_file] [location_controls]

The input_list is a list of relocatable object modules (files) separated by commas. The output_list is the name of the output absolute object module. If none is supplied, it defaults to the name of the first input file without any suffix. The location_controls set start addresses for the named segments.

For example, suppose three modules or files (MAIN.OBJ, MESSAGES.OBJ, and SUBROUTINES.OBJ) are to be combined into an executable program (EXAMPLE), and that these modules each contain two relocatable segments, one called EPROM of type CODE, and the other called ONCHIP of type DATA. Suppose further that the code segment is to be executable at address 4000H and the data segment is to reside starting at address 30H (in internal RAM). The following linker invocation could be used:

```
RS51 MAIN.OBJ, MESSAGES.OBJ, SUBROUTINES.OBJ TO EXAMPLE & CODE
(EPROM (4000H) DATA (ONCHIP (30H))
```

Note that the ampersand character "&" is used as the line continuaton character.

If the program begins at the label START, and this is the first instruction in the MAIN module, then execution begins at address 4000H. If the MAIN module was not linked first, or if the label START is not at the beginning of MAIN, then the program's entry point can be determined by examining the symbol table in the listing file EXAMPLE.M51 created by RL51. By default, EXAMPLE.M51 will contain only the link map. If a symbol table is desired, then each source program must have used the SDEBUG control. The following table shows the assembler controls supported by ASM51.

Assembler controls supported by ASM51				
NAME	PRIMARY/ GENERAL	DEFAULT	ABBREV.	MEANING
DATE (date)	P	DATE( )	DA	Place string in header (9 char. max.)
DEBUG	P	NODEBUG	DB	Outputs debug symbol information to object file
EJECT	G	not applicable	EJ	Continue listing on next page
ERRORPRINT (file)	P	NOERRORPRINT	EP	Designates a file to receive error messages in addition to the listing file (defauts to console)
NOERRORPRINT	P	NOERRORPRINT	NOEP	Designates that error messages will be printed in listing file only
GEN	G	GENONLY	GO	List only the fully expanded source as if all lines generated by a macro call were already in the source file
GENONLY	G	GENONLY	NOGE	List only the original source text in the listing file
INCLUED(file)	G	not applicable	IC	Designates a file to be included as part of the program
LIST	G	LIST	LI	Print subsequent lines of source code in listing file
NOLIST	G	LIST	NOLI	Do not print subsequent lines of source code in lisitng file
MACRO (men_precent)	P	MACRO(50)	MR	Evaluate and expand all macro calls. Allocate percentage of free memory for macro processing
NOMACRO	P	MACRO(50)	NOMR	Do not evalutate macro calls
MOD51	P	MOD51	MO	Recognize the 8051-specific predefined special function registers
NOMOD51	P	MOD51	NOMO	Do not recognize the 8051-specific predefined special function registers
OBJECT(file)	P	OBJECT(source.OBJ)	OJ	Designates file to receive object code
NOOBJECT	P	OBJECT(source.OBJ)	NOOJ	Designates that no object file will be created
PAGING	P	PAGING	PI	Designates that listing file be broken into pages and each will have a header
NOPAGING	P	PAGING	NOPI	Designates that listing file will contain no page breaks
PAGELENGTH (N)	P	PAGELENGT(60)	PL	Sets maximun number of lines in each page of listing file (range=10 to 65536)
PAGE WIDTH (N)	P	PAGEWIDTH(120)	PW	Set maximum number of characters in each line of listing file (range = 72 to 132)
PRINT(file)	P	PRINT(source.LST)	PR	Designates file to receive source listing
NOPRINT	P	PRINT(source.LST)	NOPR	Designates that no listing file will be created
SAVE	G	not applicable	SA	Stores current control settings from SAVE stack
RESTORE	G	not applicable	RS	Restores control settings from SAVE stack
REGISTERBANK (rb,...)	P	REGISTERBANK(0)	RB	Indicates one or more banks used in program module
NOREGISTER-BANK	P	REGISTERBANK(0)	NORB	Indicates that no register banks are used
SYMBOLS	P	SYMBOLS	SB	Creates a formatted table of all symbols used in program
NOSYMBOLS	P	SYMBOLS	NOSB	Designates that no symbol table is created
TITLE(string)	G	TITLE( )	TT	Places a string in all subsequent page headers (max.60 characters)
WORKFILES (path)	P	same as source	WF	Designates alternate path for temporay workfiles
XREF	P	NOXREF	XR	Creates a cross reference listing of all symbols used in program
NOXREF	P	NOXREF	NOXR	Designates that no cross reference list is created

MACROS

The macro processing facility (MPL) of ASM51 is a "string replacement" facility. Macros allow frequently used sections of code be defined once using a simple mnemonic and used anywhere in the program by inserting the mnemonic. Programming using macros is a powerful extension of the techniques described thus far. Macros can be defined anywhere in a source program and subsequently used like any other instruction. The syntax for macro definition is

```
%*DEFINE        (call_pattern)        (macro_body)
```

Once defined, the call pattern is like a mnemonic; it may be used like any assembly language instruction by placing it in the mnemonic field of a program. Macros are made distinct from "real" instructions by preceding them with a percent sign, "%". When the source program is assembled, everything within the macro-body, on a character-by-character basis, is substituted for the call-pattern. The mystique of macros is largely unfounded. They provide a simple means for replacing cumbersome instruction patterns with primitive, easy-to-remember mnemonics. The substitution, we reiterate, is on a character-by-character basis—nothing more, nothing less.

For example, if the following macro definition appears at the beginning of a source file,

```
%*DEFINE        (PUSH_DPTR)
                        (PUSH   DPH
                         PUSH   DPL
                         )
```

then the statement

```
%PUSH_DPTR
```

will appear in the .LST file as

```
PUSH    DPH
PUSH    DPL
```

The example above is a typical macro. Since the 8051 stack instructions operate only on direct addresses, pushing the data pointer requires two PUSH instructions. A similar macro can be created to POP the data pointer.

There are several distinct advantages in using macros:

A source program using macros is more readable, since the macro mnemonic is generally more indicative of the intended operation than the equivalent assembler instructions.
The source program is shorter and requires less typing.
Using macros reduces bugs
Using macros frees the programmer from dealing with low-level details.

The last two points above are related. Once a macro is written and debugged, it is used freely without the worry of bugs. In the PUSH_DPTR example above, if PUSH and POP instructions are used rather than push and pop macros, the programmer may inadvertently reverse the order of the pushes or pops. (Was it the high-byte or low-byte that was pushed first?) This would create a bug. Using macros, however, the details are worked out once—when the macro is written—and the macro is used freely thereafter, without the worry of bugs.

Since the replacement is on a character-by-character basis, the macro definition should be carefully constructed with carriage returns, tabs, ect., to ensure proper alignment of the macro statements with the rest of the assembly language program. Some trial and error is required.

There are advanced features of ASM51's macro-processing facility that allow for parameter passing, local labels, repeat operations, assembly flow control, and so on. These are discussed below.

Parameter Passing

A macro with parameters passed from the main program has the following modified format:

```
%*DEFINE        (macro_name  (parameter_list))    (macro_body)
```

For example, if the following macro is defined,

```
%*DEFINE        (CMPA#  (VALUE))
                (CJNE    A, #%VALUE, $ + 3
                 )
```

then the macro call

```
%CMPA#  (20H)
```

will expand to the following instruction in the .LST file:

```
CJNE     A, #20H, $ + 3
```

Although the 8051 does not have a "compare accumulator" instruction, one is easily created using the CJNE instruction with "$+3" (the next instruction) as the destination for the conditional jump. The CMPA# mnemonic may be easier to remember for many programmers. Besides, use of the macro unburdens the programmer from remembering notational details, such as "$+3."

Let's develop another example. It would be nice if the 8051 had instructions such as

```
JUMP    IF  ACCUMULATOR  GREATER  THAN  X
JUMP    IF  ACCUMULATOR  GREATER  THAN  OR  EQUAL  TO  X
JUMP    IF  ACCUMULATOR  LESS  THAN  X
JUMP    IF  ACCUMULATOR  LESS  THAN  OR  EQUAL  TO  X
```

but it does not. These operations can be created using CJNE followed by JC or JNC, but the details are tricky. Suppose, for example, it is desired to jump to the label GREATER_THAN if the accumulator contains an ASCII code greater than "Z" (5AH). The following instruction sequence would work:

```
CJNE     A, #5BH, $÷3
JNC      GREATER_THAN
```

The CJNE instruction subtracts 5BH (i.e., "Z" + 1) from the content of A and sets or clears the carry flag accordingly. CJNE leaves C=1 for accumulator values 00H up to and including 5AH. (Note: 5AH-5BH<0, therefore C=1; but 5BH-5BH=0, therefore C=0.) Jumping to GREATER_THAN on the condition "not carry" correctly jumps for accumulator values 5BH, 5CH, 5DH, and so on, up to FFH. Once details such as these are worked out, they can be simplified by inventing an appropriate mnemonic, defining a macro, and using the macro instead of the corresponding instruction sequence. Here's the definition for a "jump if greater than" macro:

```
%*DEFINE        (JGT (VALUE,  LABEL))
                    (CJNE    A, #%VALUE+1, $+3      ;JGT
                     JNC      %LABEL
                    )
```

To test if the accumulator contains an ASCII code greater than "Z," as just discussed,the macro would be called as

```
%JGT     ('Z', GREATER_THAN)
```

ASM51 would expand this into

```
CJNE     A, #5BH, $+3        ;JGT
JNC      GREATER_THAN
```

The JGT macro is an excellent example of a relevant and powerful use of macros. By using macros, the programmer benefits by using a meaningful mnemonic and avoiding messy and potentially bug-ridden details.

Local Labels

Local labels may be used within a macro using the following format:

```
%*DEFINE        (macro_name  [(parameter_list)])
                        [LOCAL   list_of_local_labels]    (macro_body)
```

For example, the following macro definition

```
%*DEFINE    (DEC_DPTR)    LOCAL  SKIP
                  (DEC      DPL                          ;DECREMENT DATA POINTER
                   MOV      A,          DPL
                   CJNE     A,          #0FFH,    %SKIP
                   DEC      DPL
%SKIP:             )
```

would be called as

```
%DEC_DPTR
```

and would be expanded by ASM51 into

```
            DEC     DPL                     ;DECREMENT  DATA  POINTER
            MOV     A,          DPL
            CJNE    A,          #0FFH,    SKIP00
            DEC     DPH
SKIP00:
```

Note that a local label generally will not conflict with the same label used elsewhere in the source program, since ASM51 appends a numeric code to the local label when the macro is expanded. Furthermore, the next use of the same local label receives the next numeric code, and so on.

The macro above has a potential "side effect." The accumulator is used as a temporary holding place for DPL. If the macro is used within a section of code that uses A for another purpose, the value in A would be lost. This side effect probably represents a bug in the program. The macro definition could guard against this by saving A on the stack. Here's an alternate definition for the DEC_DPTR macro:

```
%*DEFINE        (DEC_DPTR)      LOCAL SKIP
                  (PUSHACC
                  DEC     DPL                       ;DECREMENT DATA POINTER
                  MOV     A,          DPL
                  CJNE    A,          #0FFH, %SKIP
                  DEC     DPH
%SKIP:            POP     ACC
                  )
```

Repeat Operations

This is one of several built-in (predefined) macros. The format is

```
%REPEAT         (expression)        (text)
```

For example, to fill a block of memory with 100 NOP instructions,

```
%REPEAT    (100)
 (NOP
 )
```

Control Flow Operations

The conditional assembly of section of code is provided by ASM51's control flow macro definition. The format is

```
%IF  (expression)   THEN  (balanced_text)
[ELSE     (balanced_text)]  FI
```

For example,

```
INTRENAL        EQU     1       ;1 = 8051 SERIAL I/O DRIVERS
                                ;0 = 8251 SERIAL I/O DRIVERS
                .
                .
                %IF (INTERNAL) THEN
(INCHAR:        .               ;8051 DRIVERS
                .
OUTCHR:         .
                .
                ) ELSE
(INCHAR:        .               ;8251 DRIVERS
                .
OUTCHR:         .
                .
                )
```

In this example, the symbol INTERNAL is given the value 1 to select I/O subroutines for the 8051's serial port, or the value 0 to select I/O subroutines for an external UART, in this case the 8251. The IF macro causes ASM51 to assemble one set of drivers and skip over the other. Elsewhere in the program, the INCHAR and OUTCHR subroutines are used without consideration for the particular hardware configuration. As long as the program as assembled with the correct value for INTERNAL, the correct subroutine is executed.

附录B：C语言编程

ADVANTAGES AND DISADVANTAGES OF 8051 C

The advantages of programming the 8051 in C as compared to assembly are:

- Offers all the benefits of high-level, structured programming languages such as C, including the ease of writing subroutines
- Often relieves the programmer of the hardware details that the complier handles on behalf of the programmer
- Easier to write, especially for large and complex programs
- Produces more readable program source codes

Nevertheless, 8051 C, being very similar to the conventional C language, also suffers from the following disadvantages:

- Processes the disadvantages of high-level, structured programming languages.
- Generally generates larger machine codes
- Programmer has less control and less ability to directly interact with hardware

To compare between 8051 C and assembly language, consider the solutions to the Example—Write a program using Timer 0 to create a 1KHz square wave on P1.0.
A solution written below in 8051 C language:

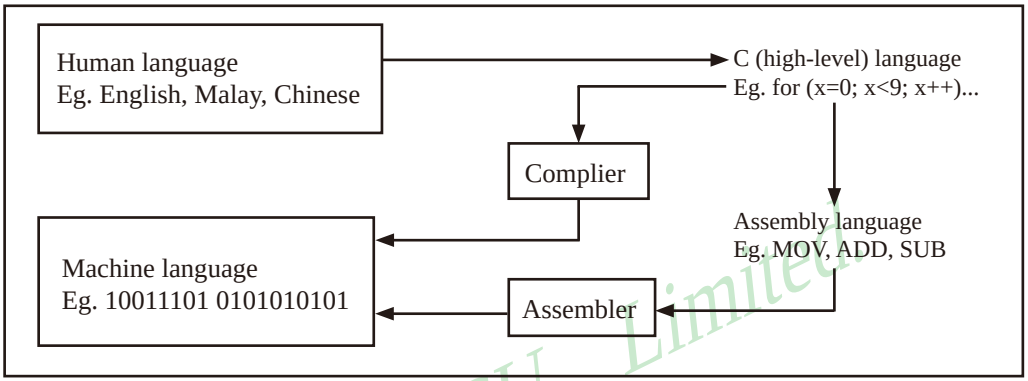
```
sbit portbit = P1^0;            /*Use variable portbit to refer to P1.0*/
main ( )
{
TMOD = 1;
while (1)
      {
          TH0 = 0xFE;
          TL0 = 0xC;
          TR0 = 1;
          while (TF0 !=1);
          TR0 = 0;
          TF0 = 0;
          portbit = !(P1.^0);
       }
}
```

A solution written below in assembly language:

```
            ORG     8100H
            MOV     TMOD,   #01H        ;16-bit timer mode
LOOP:       MOV     TH0,    #0FEH       ;-500 (high byte)
            MOV     TL0,    #0CH        ;-500 (low byte)
            SETB    TR0                 ;start timer
WAIT:       JNB     TF0,    WAIT        ;wait for overflow
            CLR     TR0                 ;stop timer
            CLR     TF0                 ;clear timer overflow flag
            CPL     P1.0                ;toggle port bit
            SJMP    LOOP                ;repeat
            END
```

Notice that both the assembly and C language solutions for the above example require almost the same number of lines. However, the difference lies in the readability of these programs. The C version seems more human than assembly, and is hence more readable. This often helps facilitate the human programmer's efforts to write even very complex programs. The assembly language version is more closely related to the machine code, and though less readable, often results in more compact machine code. As with this example, the resultant machine code from the assembly version takes 83 bytes while that of the C version requires 149 bytes, an increase of 79.5%!

The human programmer's choice of either high-level C language or assembly language for talking to the 8051, whose language is machine language, presents an interesting picture, as shown in following figure.



Conversion between human, high-level, assembly, and machine language

8051 C COMPILERS

We saw in the above figure that a complier is needed to convert programs written in 8051 C language into machine language, just as an assembler is needed in the case of programs written in assembly language. A complier basically acts just like an assembler, except that it is more complex since the difference between C and machine language is far greater than that between assembly and machine language. Hence the complier faces a greater task to bridge that difference.

Currently, there exist various 8051 C complier, which offer almost similar functions. All our examples and programs have been compiled and tested with Keil's µ Vision 2 IDE by Keil Software, an integrated 8051 program development envrionment that includes its C51 cross compiler for C. A cross compiler is a compiler that normally runs on a platform such as IBM compatible PCs but is meant to compile programs into codes to be run on other platforms such as the 8051.

DATA TYPES

8051 C is very much like the conventional C language, except that several extensions and adaptations have been made to make it suitable for the 8051 programming environment. The first concern for the 8051 C programmer is the data types. Recall that a data type is something we use to store data. Readers will be familiar with the basic C data types such as int, char, and float, which are used to create variables to store integers, characters, or floating-points. In 8051 C, all the basic C data types are supported, plus a few additional data types meant to be used specifically with the 8051.

The following table gives a list of the common data types used in 8051 C. The ones in bold are the specific 8051 extensions. The data type **bit** can be used to declare variables that reside in the 8051's bit-addressable locations (namely byte locations 20H to 2FH or bit locations 00H to 7FH). Obviously, these bit variables can only store bit values of either 0 or 1. As an example, the following C statement:

```
bit   flag = 0;
```

declares a bit variable called flag and initializes it to 0.

Data types used in 8051 C language

Data Type	Bits	Bytes	Value Range
bit	1		0 to 1
signed char	8	1	-128 to +127
unsigned char	8	1	0 to 255
enum	16	2	-32768 to +32767
signed short	16	2	-32768 to +32767
unsigned short	16	2	0 to 65535
signed int	16	2	-32768 to +32767
unsigned int	16	2	0 to 65535
signed long	32	4	-2,147,483,648 to +2,147,483,647
unsigned long	32	4	0 to 4,294,967,295
float	32	4	±1.175494E-38 to ±3.402823E+38
sbit	1		0 to 1
sfr	8	1	0 to 255
sfr16	16	2	0 to 65535

The data type **sbit** is somewhat similar to the bit data type, except that it is normally used to declare 1-bit variables that reside in special function registes (SFRs). For example:

```
sbit    P = 0xD0;
```

declares the **sbit** variable P and specifies that it refers to bit address D0H, which is really the LSB of the PSW SFR. Notice the difference here in the usage of the assignment ("=") operator. In the context of **sbit** declarations, it indicatess what address the **sbit** variable resides in, while in **bit** declarations, it is used to specify the initial value of the **bit** variable.

Besides directly assigning a bit address to an **sbit** variable, we could also use a previously defined **sfr** variable as the base address and assign our **sbit** variable to refer to a certain bit within that **sfr**. For example:

```
sfr     PSW = 0xD0;
sbit    P = PSW^0;
```

This declares an **sfr** variable called PSW that refers to the byte address D0H and then uses it as the base address to refer to its LSB (bit 0). This is then assigned to an **sbit** variable, P. For this purpose, the carat symbol (^) is used to specify bit position 0 of the PSW.

A third alternative uses a constant byte address as the base address within which a certain bit is referred. As an illustration, the previous two statements can be replaced with the following:

```
sbit    P = 0xD0 ^ 0;
```

Meanwhile, the **sfr** data type is used to declare byte (8-bit) variables that are associated with SFRs. The statement:

```
sfr     IE = 0xA8;
```

declares an **sfr** variable IE that resides at byte address A8H. Recall that this address is where the Interrupt Enable (IE) SFR is located; therefore, the sfr data type is just a means to enable us to assign names for SFRs so that it is easier to remember.

The **sfr16** data type is very similar to **sfr** but, while the **sfr** data type is used for 8-bit SFRs, **sfr16** is used for 16-bit SFRs. For example, the following statement:

```
sfr16   DPTR = 0x82;
```

declares a 16-bit variable DPTR whose lower-byte address is at 82H. Checking through the 8051 architecture, we find that this is the address of the DPL SFR, so again, the **sfr16** data type makes it easier for us to refer to the SFRs by name rather than address. There's just one thing left to mention. When declaring **sbit**, **sfr**, or **sfr16** variables, remember to do so outside main, otherwise you will get an error.

In actual fact though, all the SFRs in the 8051, including the individual flag, status, and control bits in the bit-addressable SFRs have already been declared in an include file, called reg51.h, which comes packaged with most 8051 C compilers. By using reg51.h, we can refer for instance to the interrupt enable register as simply IE rather than having to specify the address A8H, and to the data pointer as DPTR rather than 82H. All this makes 8051 C programs more human-readable and manageable. The contents of reg51.h are listed below.

```
/*--------------------------------------------------------------------------------------------------------------------
REG51.H
Header file for generic 8051 microcontroller.
--------------------------------------------------------------------------------------------------------------------*/

/* BYTE Register */
sfr     P0      = 0x80;
sfr     P1      = 0x90;
sfr     P2      = 0xA0;
sfr     P3      = 0xB0;
sfr     PSW     = 0xD0;
sfr     ACC     = 0xE0;
sfr     B       = 0xF0;
sfr     SP      = 0x81;
sfr     DPL     = 0x82;
sfr     DPH     = 0x83;
sfr     PCON    = 0x87;
sfr     TCON    = 0x88;
sfr     TMOD    = 0x89;
sfr     TL0     = 0x8A;
sfr     TL1     = 0x8B;
sfr     TH0     = 0x8C;
sfr     TH1     = 0x8D;
sfr     IE      = 0xA8;
sfr     IP      = 0xB8;
sfr     SCON    = 0x98;
sfr     SBUF    = 0x99;
/* BIT Register */
/* PSW */
sbit    CY      = 0xD7;
sbit    AC      = 0xD6;
sbit    F0      = 0xD5;
sbit    RS1     = 0xD4;
sbit    RS0     = 0xD3;
sbit    OV      = 0xD2;
sbit    P       = 0xD0;
/* TCON */
sbit    TF1     = 0x8F;
sbit    TR1     = 0x8E;
sbit    TF0     = 0x8D;
sbit    TR0     = 0x8C;
sbit    IE1     = 0x8B;
sbit    IT1     = 0x8A;
sbit    IE0     = 0x89;
sbit    IT0     = 0x88;
/* IE */
sbit    EA      = 0xAF;
sbit    ES      = 0xAC;
sbit    ET1     = 0xAB;
sbit    EX1     = 0xAA;
sbit    ET0     = 0xA9;
sbit    EX0     = 0xA8;
/* IP */
sbit    PS      = 0xBC;
sbit    PT1     = 0xBB;
sbit    PX1     = 0xBA;
sbit    PT0     = 0xB9;
sbit    PX0     = 0xB8;
/* P3 */
sbit    RD      = 0xB7;
sbit    WR      = 0xB6;
sbit    T1      = 0xB5;
sbit    T0      = 0xB4;
sbit    INT1    = 0xB3;
sbit    INT0    = 0xB2;
sbit    TXD     = 0xB1;
sbit    RXD     = 0xB0;
/* SCON */
sbit    SM0     = 0x9F;
sbit    SM1     = 0x9E;
sbit    SM2     = 0x9D;
sbit    REN     = 0x9C;
sbit    TB8     = 0x9B;
sbit    RB8     = 0x9A;
sbit    TI      = 0x99;
sbit    RI      = 0x98;
```

MEMORY TYPES AND MODELS

The 8051 has various types of memory space, including internal and external code and data memory. When declaring variables, it is hence reasonable to wonder in which type of memory those variables would reside. For this purpose, several memory type specifiers are available for use, as shown in following table.

Memory types used in 8051 C language	
Memory Type	Description (Size)
code	Code memory (64 Kbytes)
data	Directly addressable internal data memory (128 bytes)
idata	Indirectly addressable internal data memory (256 bytes)
bdata	Bit-addressable internal data memory (16 bytes)
xdata	External data memory (64 Kbytes)
pdata	Paged external data memory (256 bytes)

The first memory type specifier given in above table is **code**. This is used to specify that a variable is to reside in code memory, which has a range of up to 64 Kbytes. For example:

```
char    code    errormsg[ ] = "An error occurred" ;
```

declares a char array called errormsg that resides in code memory.

If you want to put a variable into data memory, then use either of the remaining five data memory specifiers in above table. Though the choice rests on you, bear in mind that each type of data memory affect the speed of access and the size of available data memory. For instance, consider the following declarations:

```
signed int   data   num1;
bit bdata    numbit;
unsigned int   xdata   num2;
```

The first statement creates a signed int variable num1 that resides in inernal **data** memory (00H to 7FH). The next line declares a bit variable numbit that is to reside in the bit-addressable memory locations (byte addresses 20H to 2FH), also known as **bdata**. Finally, the last line declares an unsigned int variable called num2 that resides in external data memory, **xdata**. Having a variable located in the directly addressable internal data memory speeds up access considerably; hence, for programs that are time-critical, the variables should be of type **data**. For other variants such as 8052 with internal data memory up to 256 bytes, the **idata** specifier may be used. Note however that this is slower than data since it must use indirect addressing. Meanwhile, if you would rather have your variables reside in external memory, you have the choice of declaring them as **pdata** or **xdata**. A variable declared to be in **pdata** resides in the first 256 bytes (a page) of external memory, while if more storage is required, **xdata** should be used, which allows for accessing up to 64 Kbytes of external data memory.

What if when declaring a variable you forget to explicitly specify what type of memory it should reside in, or you wish that all variables are assigned a default memory type without having to specify them one by one? In this case, we make use of **memory models**. The following table lists the various memory models that you can use.

Memory models used in 8051 C language	
Memory Model	Description
Small	Variables default to the internal data memory (data)
Compact	Variables default to the first 256 bytes of external data memory (pdata)
Large	Variables default to external data memory (xdata)

A program is explicitly selected to be in a certain memory model by using the C directive, #pragma. Otherwise, the default memory model is **small**. It is recommended that programs use the small memory model as it allows for the fastest possible access by defaulting all variables to reside in internal data memory.

The **compact** memory model causes all variables to default to the first page of external data memory while the **large** memory model causes all variables to default to the full external data memory range of up to 64 Kbytes.

ARRAYS

Often, a group of variables used to store data of the same type need to be grouped together for better readability. For example, the ASCII table for decimal digits would be as shown below.

ASCII table for decimal digits	
Decimal Digit	ASCII Code In Hex
0	30H
1	31H
2	32H
3	33H
4	34H
5	35H
6	36H
7	37H
8	38H
9	39H

To store such a table in an 8051 C program, an array could be used. An array is a group of variables of the same data type, all of which could be accessed by using the name of the arrary along with an appropriate index.

The array to store the decimal ASCII table is:

```
int         table [10] =
{0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39};
```

Notice that all the elements of an array are separated by commas. To access an individul element, an index starting from 0 is used. For instance, table[0] refers to the first element while table[9] refers to the last element in this ASCII table.

STRUCTURES

Sometime it is also desired that variables of different data types but which are related to each other in some way be grouped together. For example, the name, age, and date of birth of a person would be stored in different types of variables, but all refer to the person's personal details. In such a case, a structure can be declared. A structure is a group of related variables that could be of different data types. Such a structure is declared by:

```
struct      person  {
                        char name;
                        int age;
                        long DOB;
                        };
```

Once such a structure has been declared, it can be used like a data type specifier to create structure variables that have the member's name, age, and DOB. For example:

```
struct      person      grace = {"Grace", 22, 01311980};
```

would create a structure variable grace to store the name, age, and data of birth of a person called Grace. Then in order to access the specific members within the person structure variable, use the variable name followed by the dot operator (.) and the member name. Therefore, grace.name, grace.age, grace.DOB would refer to Grace's name, age, and data of birth, respectively.

POINTERS

When programming the 8051 in assembly, sometimes register such as R0, R1, and DPTR are used to store the addresses of some data in a certain memory location. When data is accessed via these registers, indirect addressing is used. In this case, we say that R0, R1, or DPTR are used to point to the data, so they are essentially pointers.

Correspondingly in C, indirect access of data can be done through specially defined pointer variables. Pointers are simply just special types of variables, but whereas normal variables are used to directly store data, pointer variables are used to store the addresses of the data. Just bear in mind that whether you use normal variables or pointer variables, you still get to access the data in the end. It is just whether you go directly to where it is stored and get the data, as in the case of normal variables, or first consult a directory to check the location of that data before going there to get it, as in the case of pointer variables.

Declaring a pointer follows the format:

```
data_type     *pointer_name;
```

where

data_type	refers to which type of data that the pointer is pointing to
*	denotes that this is a pointer variable
pointer_name	is the name of the pointer

As an example, the following declarations:

```
int   * numPtr
int   num;
numPtr = &num;
```

first declares a pointer variable called numPtr that will be used to point to data of type int. The second declaration declares a normal variable and is put there for comparison. The third line assigns the address of the num variable to the numPtr pointer. The address of any variable can be obtained by using the address operator, &, as is used in this example. Bear in mind that once assigned, the numPtr pointer contains the address of the num variable, not the value of its data.

The above example could also be rewritten such that the pointer is straightaway initialized with an address when it is first declared:

```
int   num;
int   * numPtr = &num;
```

In order to further illustrate the difference between normal variables and pointer variables, consider the following, which is not a full C program but simply a fragment to illustrate our point:

```
int   num = 7;
int   * numPtr = &num;
printf ("%d\n", num);
printf ("%d\n", numPtr);
printf ("%d\n", &num);
printf ("%d\n", *numPtr);
```

The first line declare a normal variable, num, which is initialized to contain the data 7. Next, a pointer variable, numPtr, is declared, which is initialized to point to the address of num. The next four lines use the printf( ) function, which causes some data to be printed to some display terminal connected to the serial port. The first such line displays the contents of the num variable, which is in this case the value 7. The next displays the contents of the numPtr pointer, which is really some weird-looking number that is the address of the num variable.The third such line also displays the addresss of the num variable because the address operator is used to obtain num's address. The last line displays the actual data to which the numPtr pointer is pointing, which is 7. The * symbol is called the indirection operator, and when used with a pointer, indirectly obtains the data whose address is pointed to by the pointer. Therefore, the output display on the terminal would show:

7
13452 (or some other weird-looking number)
13452 (or some other weird-looking number)
7

A Pointer's Memory Type

Recall that pointers are also variables, so the question arises where they should be stored. When declaring pointers, we can specify different types of memory areas that these pointers should be in, for example:

```
int  * xdata numPtr = & num;
```

This is the same as our previous pointer examples. We declare a pointer numPtr, which points to data of type int stored in the num variable. The difference here is the use of the memory type specifier **xdata** after the *. This is specifies that pointer numPtr should reside in external data memory (**xdata**), and we say that the pointer's memory type is **xdata**.

Typed Pointers

We can go even further when declaring pointers. Consider the example:

```
int  data  * xdata  numPtr = &num;
```

The above statement declares the same pointer numPtr to reside in external data memory (**xdata**), and this pointer points to data of type int that is itself stored in the variable num in internal data memory (**data**). The memory type specifier, **data**, before the * specifies the *data memory type* while the memory type specifier, **xdata**, after the * specifies the pointer memory type.

Pointer declarations where the data memory types are explicitly specified are called typed pointers. Typed pointers have the property that you specify in your code where the data pointed by pointers should reside. The size of typed pointers depends on the data memory type and could be one or two bytes.

Untyped Pointers

When we do not explicitly state the data memory type when declaring pointers, we get untyped pointers, which are generic pointers that can point to data residing in any type of memory. Untyped pointers have the advantage that they can be used to point to any data independent of the type of memory in which the data is stored. All untyped pointers consist of 3 bytes, and are hence larger than typed pointers. Untyped pointers are also generally slower because the data memory type is not determined or known until the complied program is run at runtime. The first byte of untyped pointers refers to the data memory type, which is simply a number according to the following table. The second and third bytes are,respectively,the higher-order and lower-order bytes of the address being pointed to.

An untyped pointer is declared just like normal C, where:

```
int  * xdata  numPtr = &num;
```

does not explicitly specify the memory type of the data pointed to by the pointer. In this case, we are using untyped pointers.

Data memory type values stored in first byte of untyped pointers	
Value	Data Memory Type
1	idata
2	xdata
3	pdata
4	data/bdata
5	code

FUNCTIONS

In programming the 8051 in assembly, we learnt the advantages of using subroutines to group together common and frequently used instructions. The same concept appears in 8051 C, but instead of calling them subroutines, we call them **functions**. As in conventional C, a function must be declared and defined. A function definition includes a list of the number and types of inputs, and the type of the output (return type), puls a description of the internal contents, or what is to be done within that function.

The format of a typical function definition is as follows:

```
return_type  function_name (arguments)  [memory] [reentrant] [interrupt] [using]
{
        …
}
```

where

return_type	refers to the data type of the return (output) value
function_name	is any name that you wish to call the function as
arguments	is the list of the type and number of input (argument) values
memory	refers to an explicit memory model (small, compact or large)
reentrant	refers to whether the function is reentrant (recursive)
interrupt	indicates that the function is acctually an ISR
using	explicitly specifies which register bank to use

Consider a typical example, a function to calculate the sum of two numbers:

```
int  sum (int a, int b)
{
        return  a + b;
}
```

This function is called sum and takes in two arguments, both of type int. The return type is also int, meaning that the output (return value) would be an int. Within the body of the function, delimited by braces, we see that the return value is basically the sum of the two agruments. In our example above, we omitted explicitly specifying the options: memory, reentrant, interrupt, and using. This means that the arguments passed to the function would be using the default small memory model, meaning that they would be stored in internal data memory. This function is also by default non-recursive and a normal function, not an ISR. Meanwhile, the default register bank is bank 0.

Parameter Passing

In 8051 C, parameters are passed to and from functions and used as function arguments (inputs). Nevertheless, the technical details of where and how these parameters are stored are transparent to the programmer, who does not need to worry about these techinalities. In 8051 C, parameters are passed through the register or through memory. Passing parameters through registers is faster and is the default way in which things are done. The registers used and their purpose are described in more detail below.

Registers used in parameter passing				
Number of Argument	Char / 1-Byte Pointer	INT / 2-Byte Pointer	Long/Float	Generic Pointer
1	R7	R6 & R7	R4–R7	R1–R3
2	R5	R4 &R5	R4–R7	
3	R3	R2 & R3		

Since there are only eight registers in the 8051, there may be situations where we do not have enough registers for parameter passing. When this happens, the remaining parameters can be passed through fixed memory loacations. To specify that all parameters will be passed via memory, the NOREGPARMs control directive is used. To specify the reverse, use the REGPARMs control directive.

Return Values

Unlike parameters, which can be passed by using either registers or memory locations, output values must be returned from functions via registers. The following table shows the registers used in returning different types of values from functions.

Registers used in returning values from functions		
Return Type	Register	Description
bit	Carry Flag (C)	
char/unsigned char/1-byte pointer	R7	
int/unsigned int/2-byte pointer	R6 & R7	MSB in R6, LSB in R7
long/unsigned long	R4–R7	MSB in R4, LSB in R7
float	R4–R7	32-bit IEEE format
generic pointer	R1–R3	Memory type in R3, MSB in R2, LSB in R1

附录C：STC89C51RC/RD+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	+6.0	V
DC power supply (3V)	VDD - VSS	-0.3	+4.0	V
Voltage on any pin	-	-0.5	+ 5.5	V

DC Specification (5V MCU)

Sym	Parameter	Specification				Test Condition
		Min.	Typ	Max.	Unit	
V_{DD}	Operating Voltage	3.8	5.0	5.5	V	
I_{PD}	Power Down Current	-	< 0.1	-	uA	5V
I_{IDL}	Idle Current	-	2.0	-	mA	5V
I_{CC}	Operating Current	-	4	20	mA	5V
V_{IL1}	Input Low (P0,P1,P2,P3, P4)	-	-	0.8	V	5V
V_{IL2}	Input Low voltage (RESET, XTAL1)	-	-	1.5	V	5V
V_{IH1}	Input High (P0,P1,P2,P3, P4, /EA)	2.0	-	-	V	5V
V_{IH2}	Input High (RESET)	3.0	-	-	V	5V
I_{OL1}	Sinking Current for output low (P1,P2,P3,P4)	4	6	-	mA	5V
I_{OL2}	Sinking Current for output low(P0,ALE,PSEN)	8	12		mA	5V
I_{OH1}	Sourcing Current for output high (P1,P2,P3,P4)	150	220	-	uA	5V
I_{OH2}	Sourcing Current for output high (ALE,PSEN)	14	20	-	mA	5V
I_{IL}	Logic 0 input current (P1,P2,P3,P4)	-	18	50	uA	Vpin=0V
I_{TL}	Logic 1 to 0 transition current (P1,P2,P3,P4)	-	270	600	uA	Vpin=2.0V

DC Specification (3V MCU)

Sym	Parameter	Specification				Test Condition
		Min.	Typ	Max.	Unit	
V_{DD}	Operating Voltage	2.4	3.3	3.8	V	
I_{PD}	Power Down Current	-	<0.1	-	uA	3.3V
I_{IDL}	Idle Current	-	2.0	-	mA	3.3V
I_{CC}	Operating Current	-	4	15	mA	3.3V
V_{IL1}	Input Low (P0,P1,P2,P3,P4)	-	-	0.8	V	3.3V
V_{IL2}	Input Low (RESET, XTAL1)	-	-	1.5	V	3.3V
V_{IH1}	Input High (P0,P1,P2,P3)	2.0	-	-	V	3.3V
V_{IH2}	Input High (RESET)	3.0	-	-	V	3.3V
I_{OL1}	Sink Current for output low (P1,P2,P3,P4)	2.5	4	-	mA	3.3V
I_{OL2}	Sink Current for output low (P0,ALE,PSEN)	5	8	-	mA	3.3V
I_{OH1}	Sourcing Current for output high (P1,P2,P3,P4)	40	70	-	uA	3.3V
I_{OH2}	Sourcing Current for output high (ALE,PSEN)	8	13	-	mA	3.3V
I_{IL}	Logic 0 input current (P1,P2,P3,P4)	-	8	50	uA	Vpin=0V
I_{TL}	Logic 1 to 0 transition current (P1,P2,P3,P4)	-	110	600	uA	Vpin=2.0V

附录D：内部常规256字节RAM间接寻址测试程序

```
;/* ---  STC International Limited ------------------------------------------ */
;/* ---  STC     姚永平     2006/1/6     V1.0 ---------------------------------- */
;/* --- STC89C51RC/RD+ 系列单片机 内部常规RAM间接寻址测试序---------------------- */
;/* ---  Mobile: 13922805190 ------------------------------------------------- */
;/* ---  Fax: 0755-82905966 -------------------------------------------------- */
;/* ---  Tel: 0755-82948409 -------------------------------------------------- */
;/* ---  Web: www.STCMCU.com ------------------------------------------------- */
;/* ---  本演示程序在STC-ISP Ver 3.0A.PCB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
```

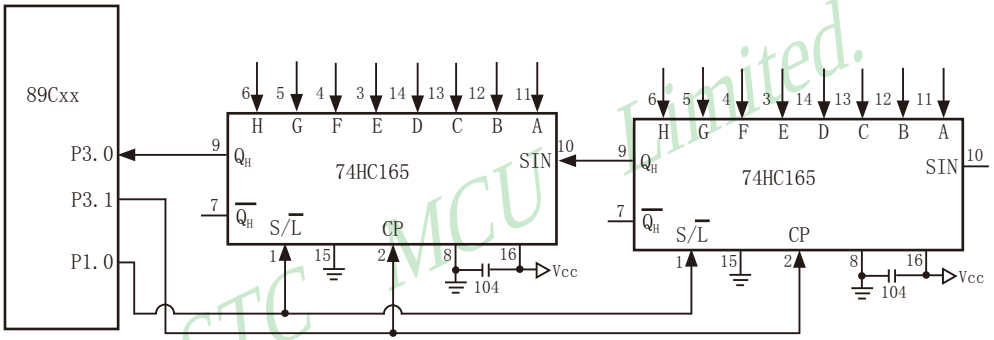
附录E：用串口扩展I/O接口

STC89C51RC/RD+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。

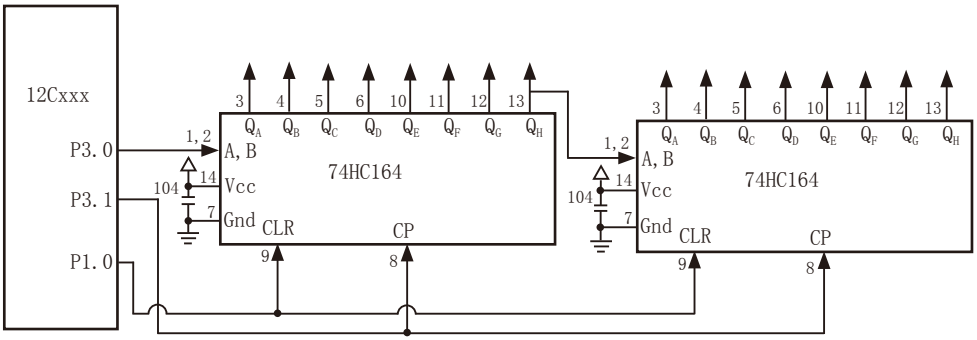
下面的程序是从16位扩展口读入5组数据（每组二个字节），并把它们转存到内部RAM 20H开始的单元中。

```
                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组数据未读完重新并行置入
                ……                              ；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。

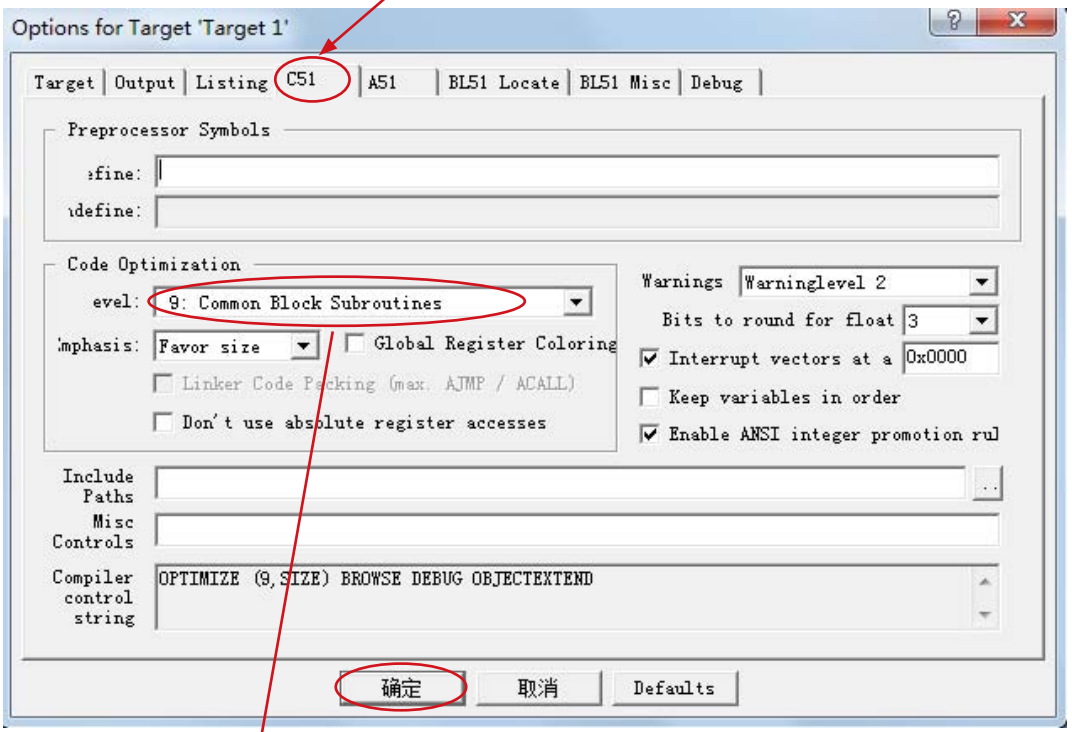
下面是将RAM缓冲区30H、31H的内容串行口由74HC164并行输出的子程序。

```
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F：如何利用Keil C软件减少代码长度

在Keil 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 Target”
2. 在“Options for Target”中选择“C51”



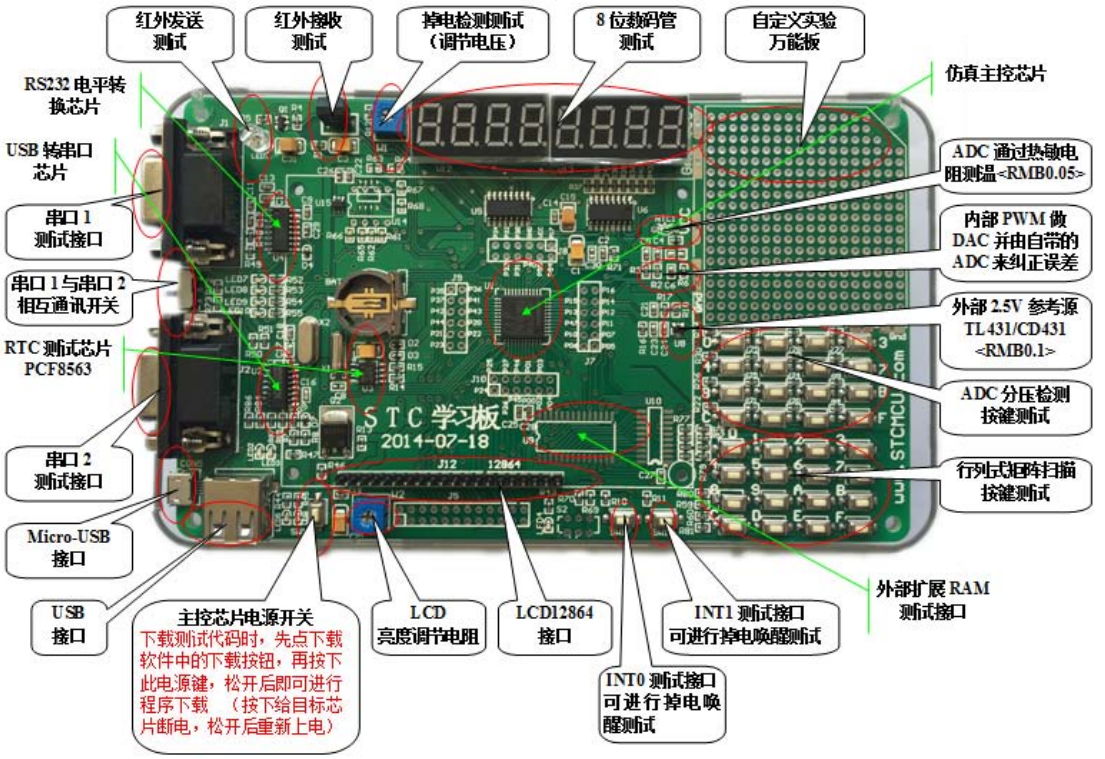
3. 选择按空间大小，9级优化程序
4. 点击“确定”后，重新编译程序即可。

附录G：STC实验箱4使用说明

G.1 实验箱4外观图

打开方式：双手捏住如图红圈所示的实验箱的把手处，双手分别向两边用力即可打开实验箱。

G.2 实验板布局图



在此，需要对“主控芯片电源开关”进行说明。“主控芯片电源开关”按钮的原理是按住此开关时主控芯片将会处于停电状态，放开此开关时主控芯片会被重新上电而进行上电复位。而对于STC的单片机，要想进行ISP下载，则必须是在上电复位时接收到串口命令才会开始执行ISP程序，所以下载程序到实验箱4的正确步骤为：

1、使用USB线将实验箱4与电脑进行连接；

2、打开STC的ISP下载软件；

3、选择单片机型号为“IAP15W4K58S4”；

4、选择实验箱4所对应的串口；

5、打开目标文件（HEX格式或者BIN格式）；

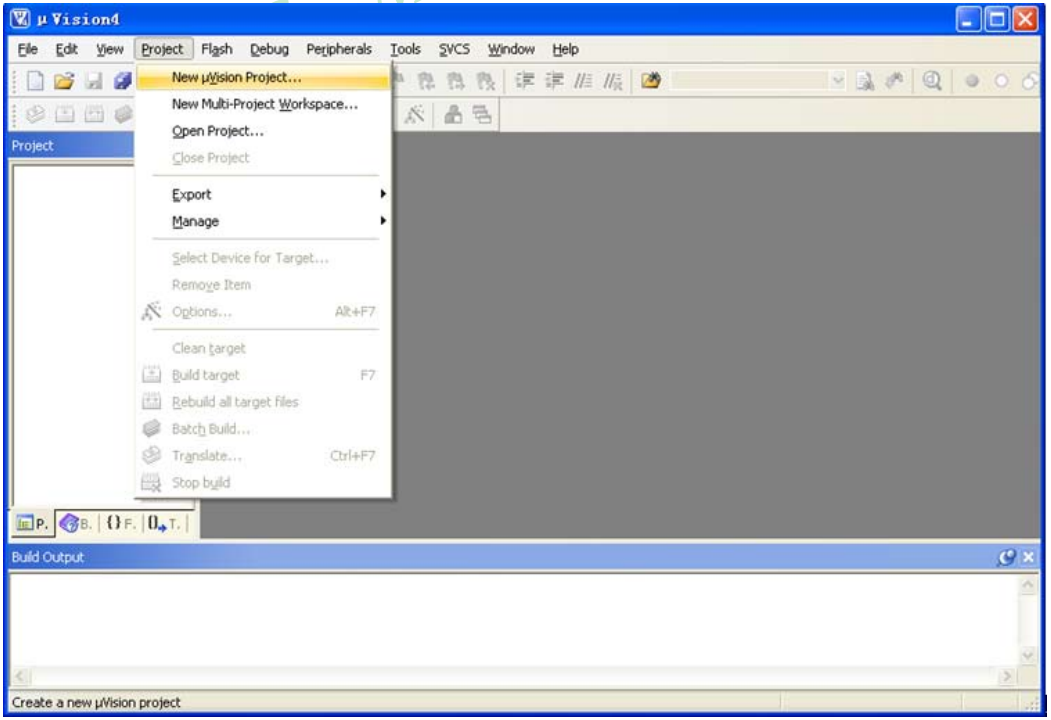
6、点击ISP下载软件中的“下载/编程”按钮；

7、按下实验箱4上的“主控芯片电源开关”，然后松开即可开始下载。

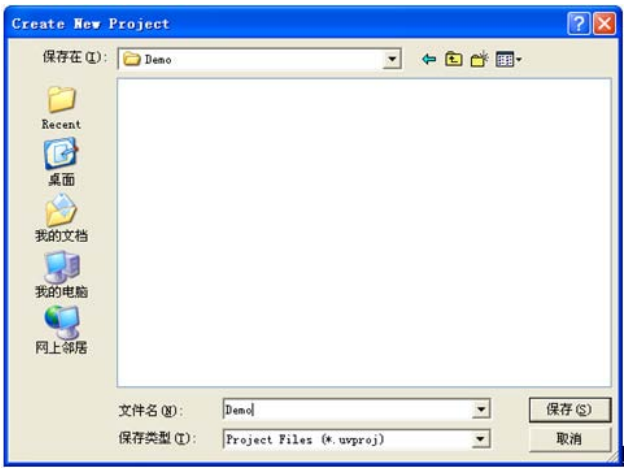
G.3 新建Keil项目

（由于Keil的版本比较多，本说明书将只使用Keil的uVersion4为例进行介绍，Keil的其他版本与之类似）

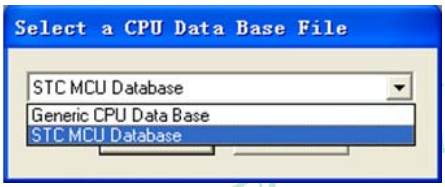
首先打开Keil软件，并打开“Project”菜单中的“New uVersion Project …”项

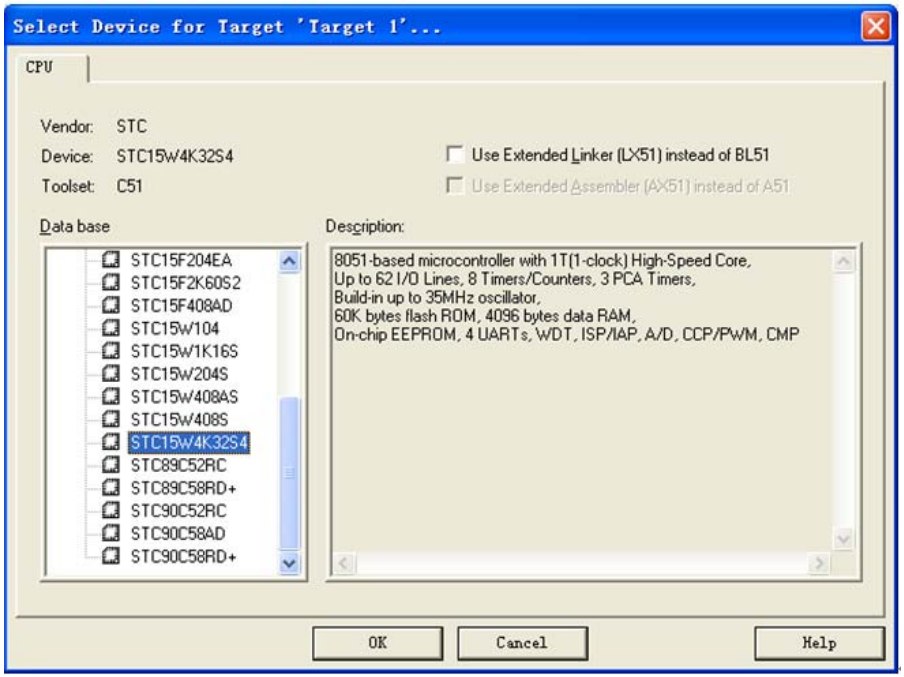


在下面的对话框中输入新建的项目名称，然后保存



接下来需要在如下的对话框内选择芯片型号

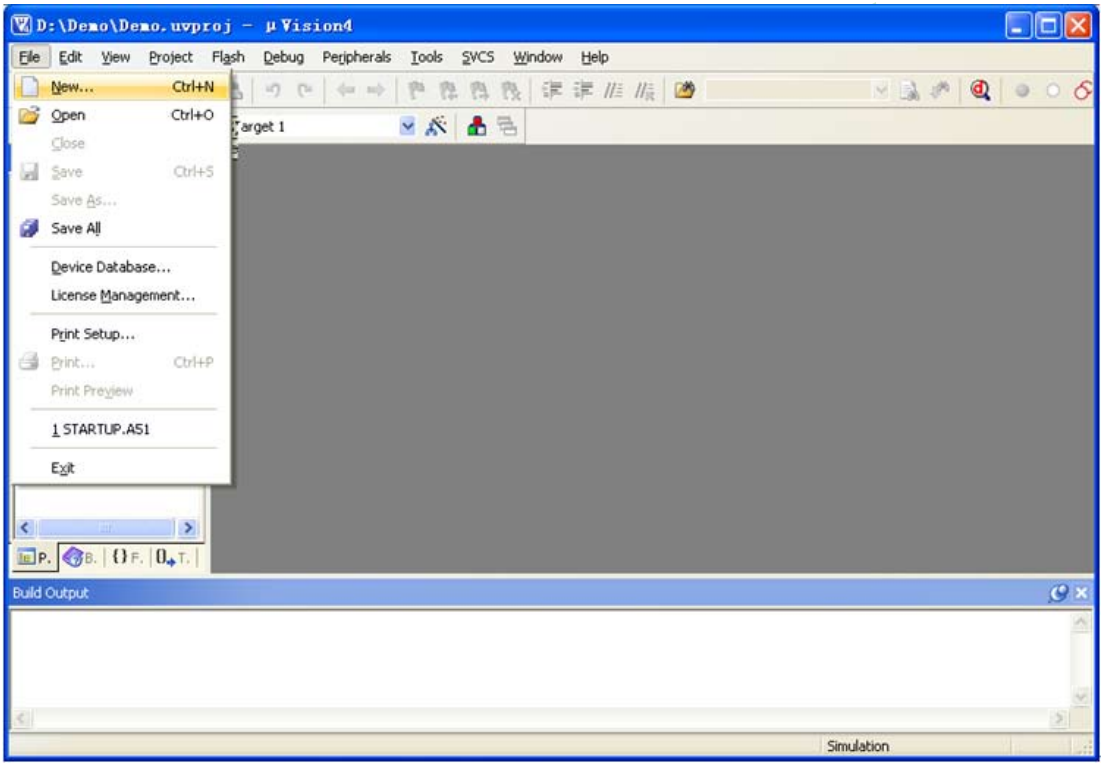




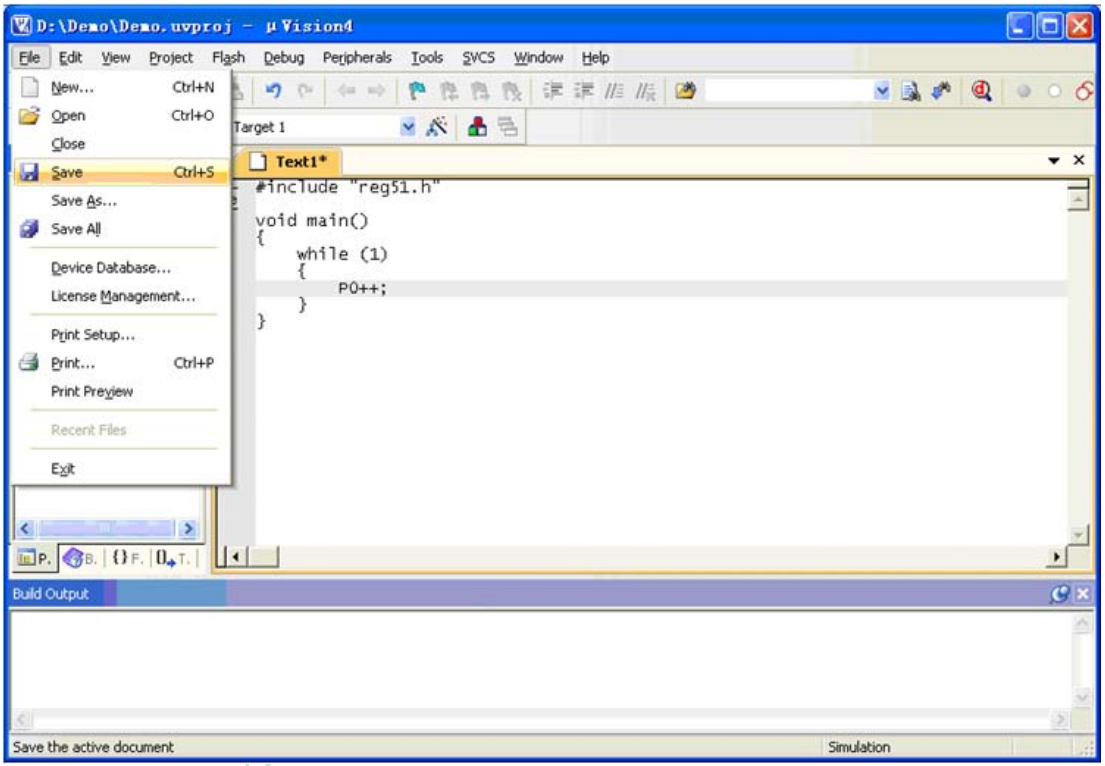
型号确定后，Keil会弹出下面的对话框，问是否需要将启动代码文件添加到项目中。
一般建议选择“是”（也可选择“否”）

至此，基本的项目文件已基本建立。

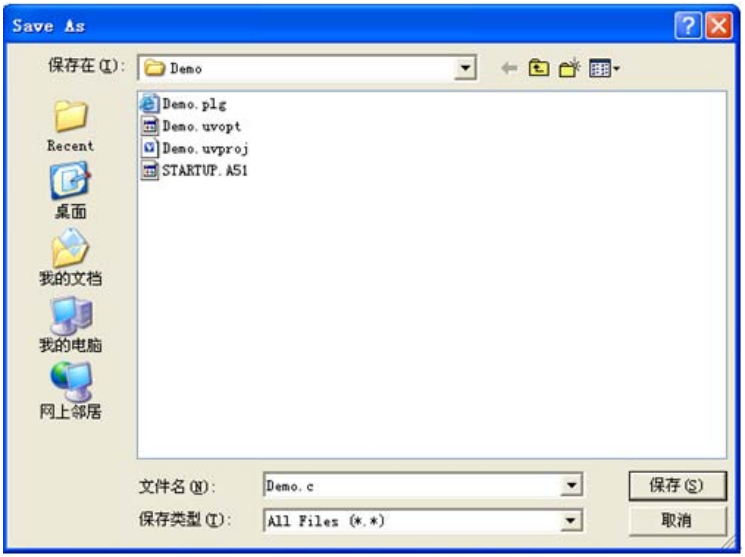
接下来需要新建源代码文件，打开“File”菜单中的“New …”项



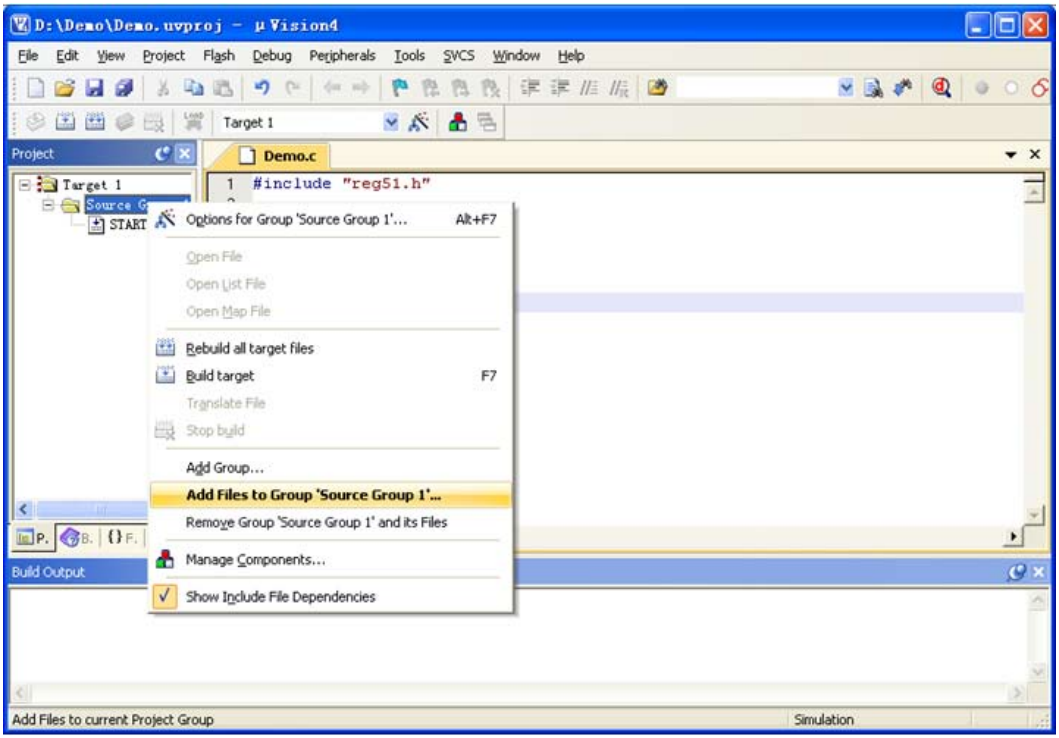
在新建的文件中输入相应的源代码，然后选择“File”菜单中的“Save”项对文件进行保存



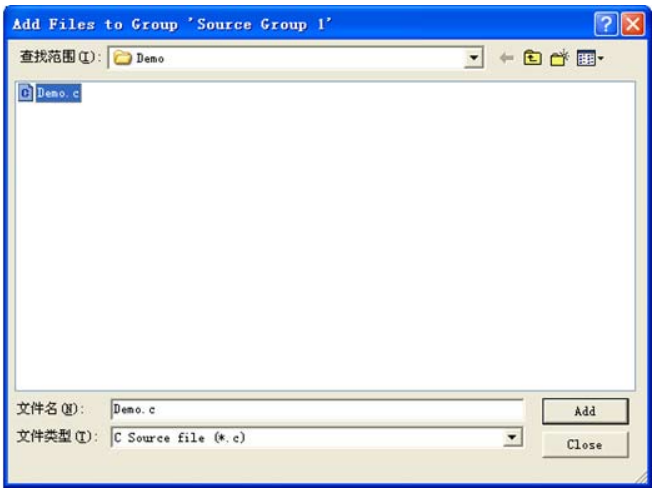
如下图



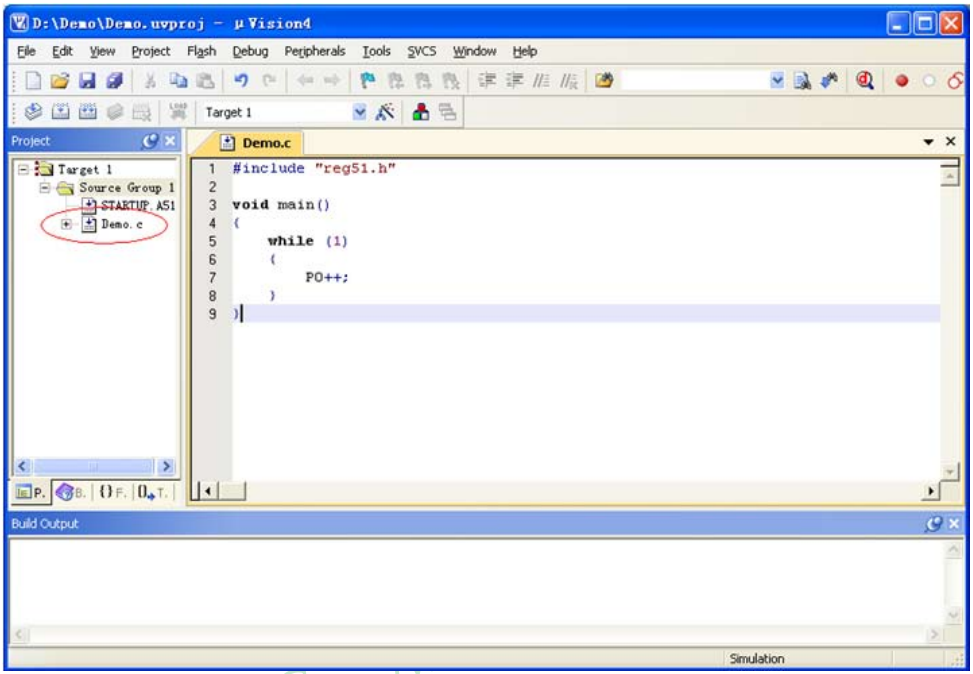
文件保存完成后需要使用下面的操作将源代码文件添加到项目中来，具体的操作方法是：使用鼠标右键单击“Project”列表中的“Source Group 1”项，在出现的右键菜单中选择“Add Files to Group ‘Source Group 1’ ”项目



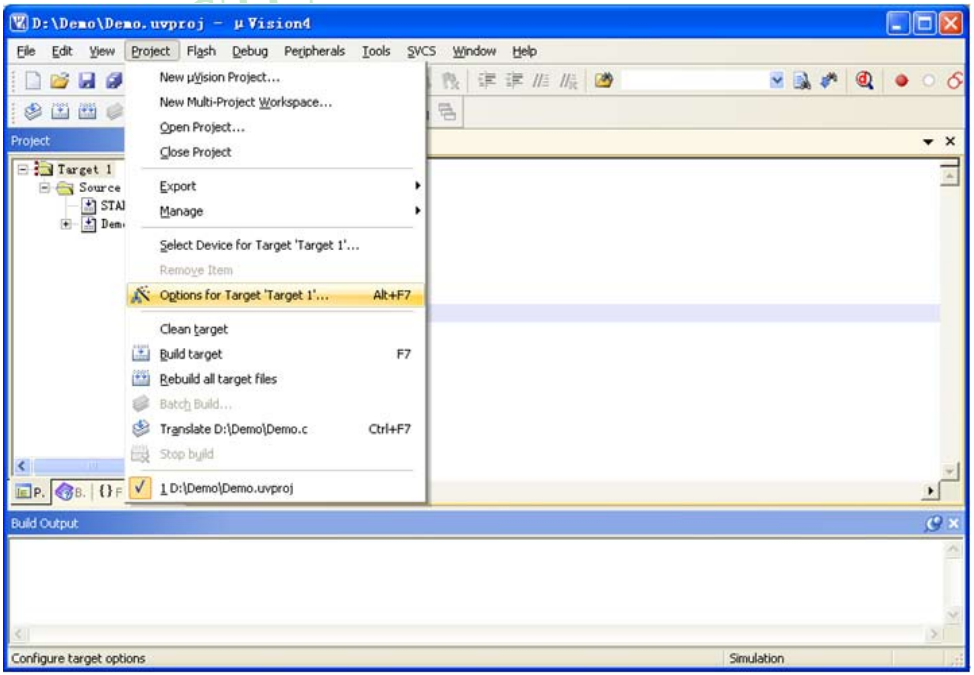
在下面的对话框中选择我们刚才保存的文件，并点击“Add”按钮即可将文件添加到项目中，完成后按下“Close”按钮关闭对话框



此时我们可以看到在项目中已经多了我们刚才添加的代码文件

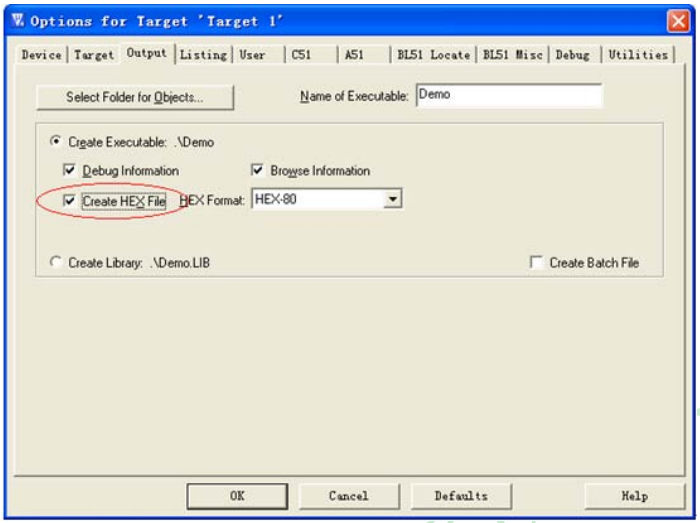


按下快捷键“Alt+F7”或者选择菜单“Project”中的“Option for Target ‘Target1’”

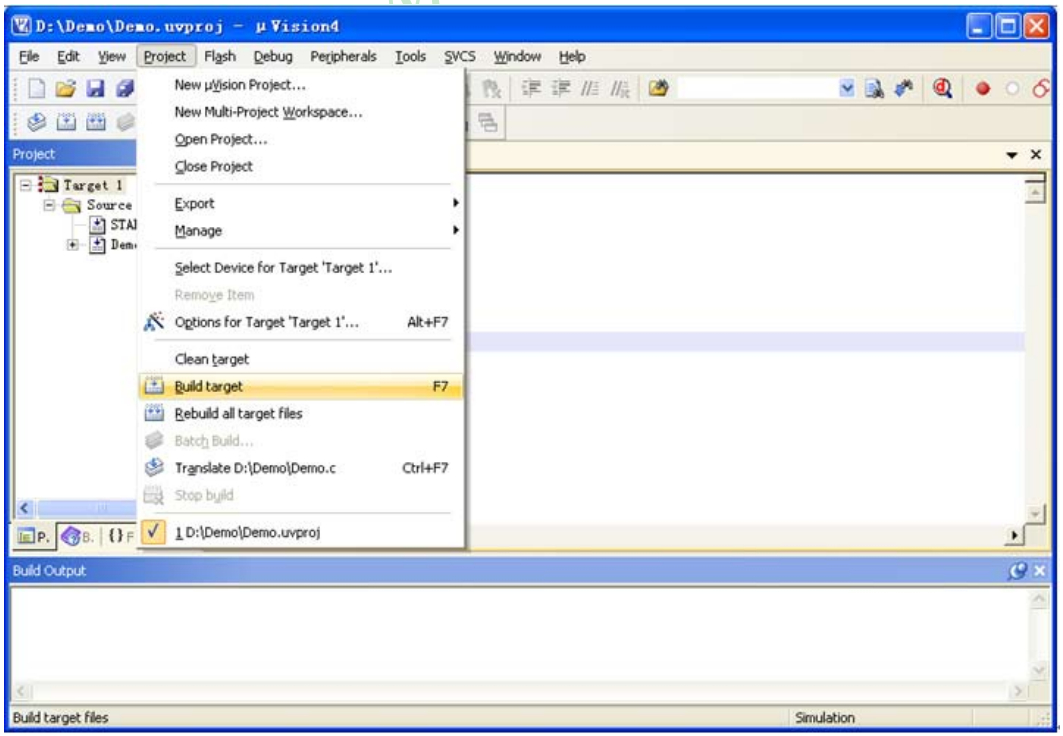


在如下的对话框中对项目进行配置：

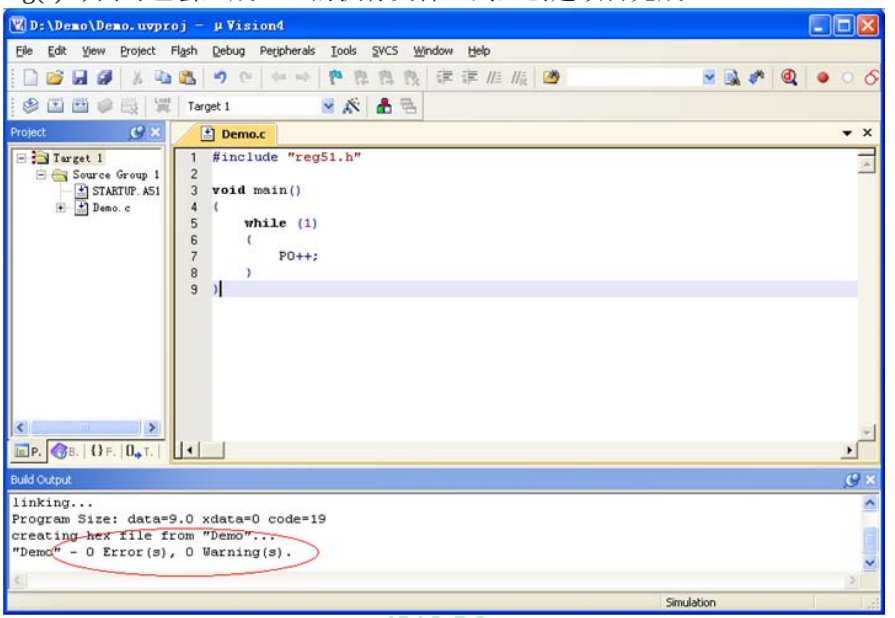
在“Output”属性页中，将“Create HEX FIle”选项打上勾，即可在项目编译完成后自动生成HEX格式的目标文件，按“OK”保存。



按下快捷键“F7”或者选择菜单“Project”中的“Build Target”项对当前项目进行编译

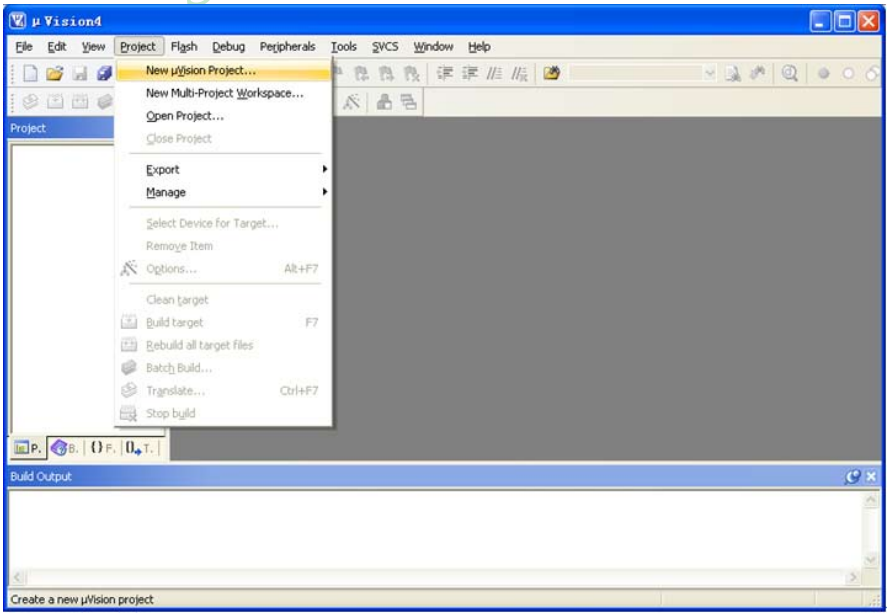


若代码中没有错误，编译完成后则会在“Build Output”的信息输出框中显示 “ 0 Error(s), 0 Warning(s) ”, 同时也会生成HEX的执行文件。到此创建项目完成。

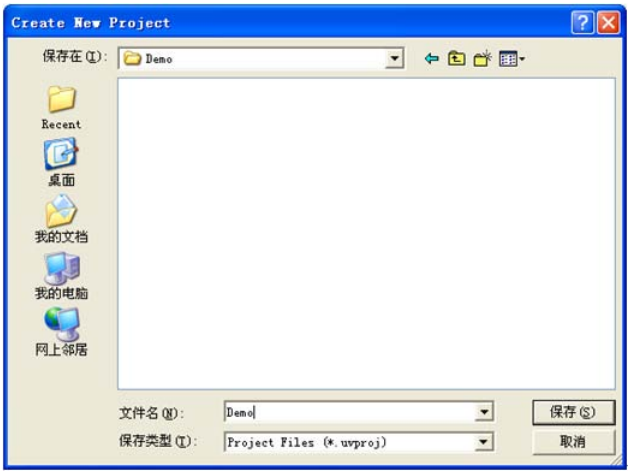


G.4 保存STC-ISP范例程序到Keil项目

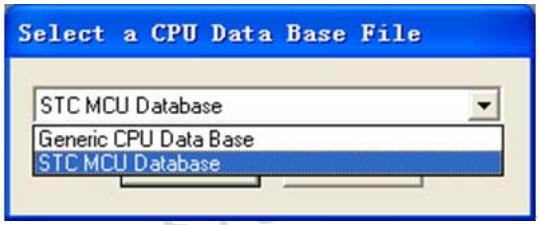
首先打开Keil软件，并打开“Project”菜单中的“New uVersion Project …”项

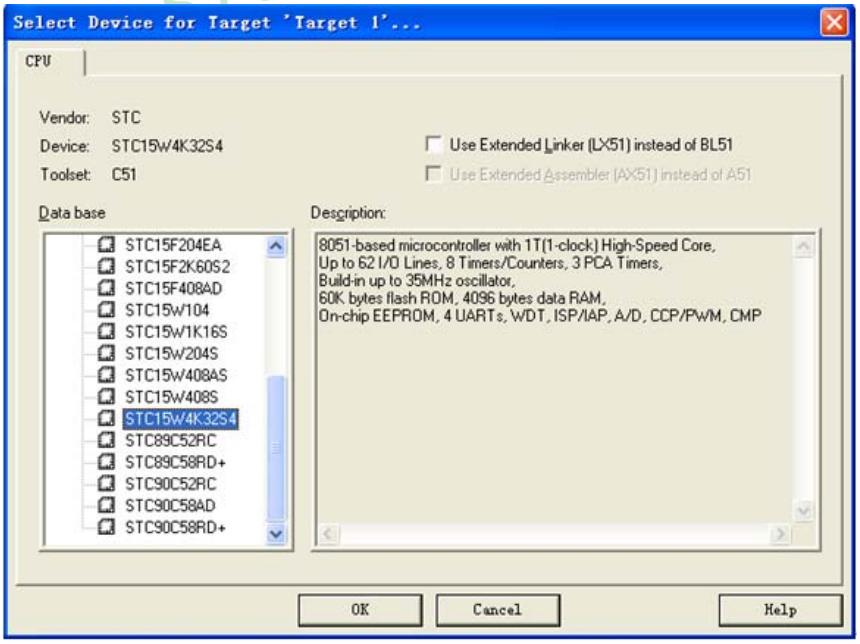


在下面的对话框中输入新建的项目名称，然后保存



接下来需要在如下的对话框内选择芯片型号



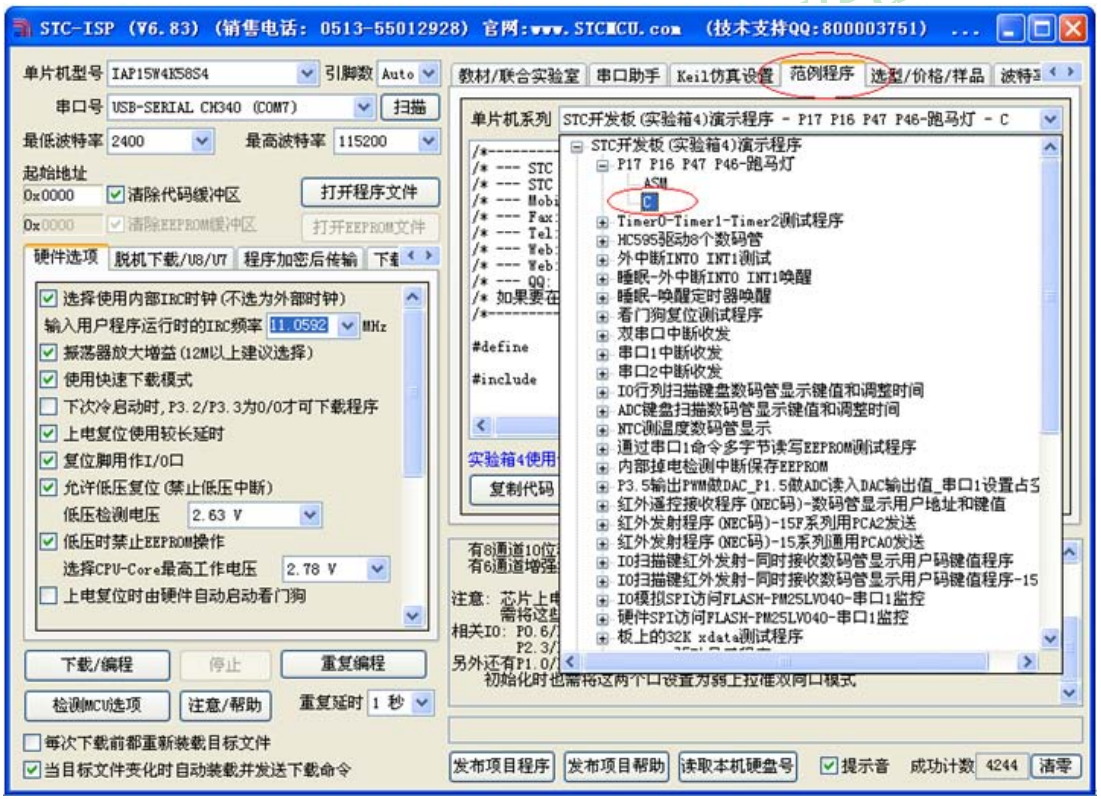


型号确定后，Keil会弹出下面的对话框，问是否需要将启动代码文件添加到项目中。一般建议选择“是”（也可选择“否”）

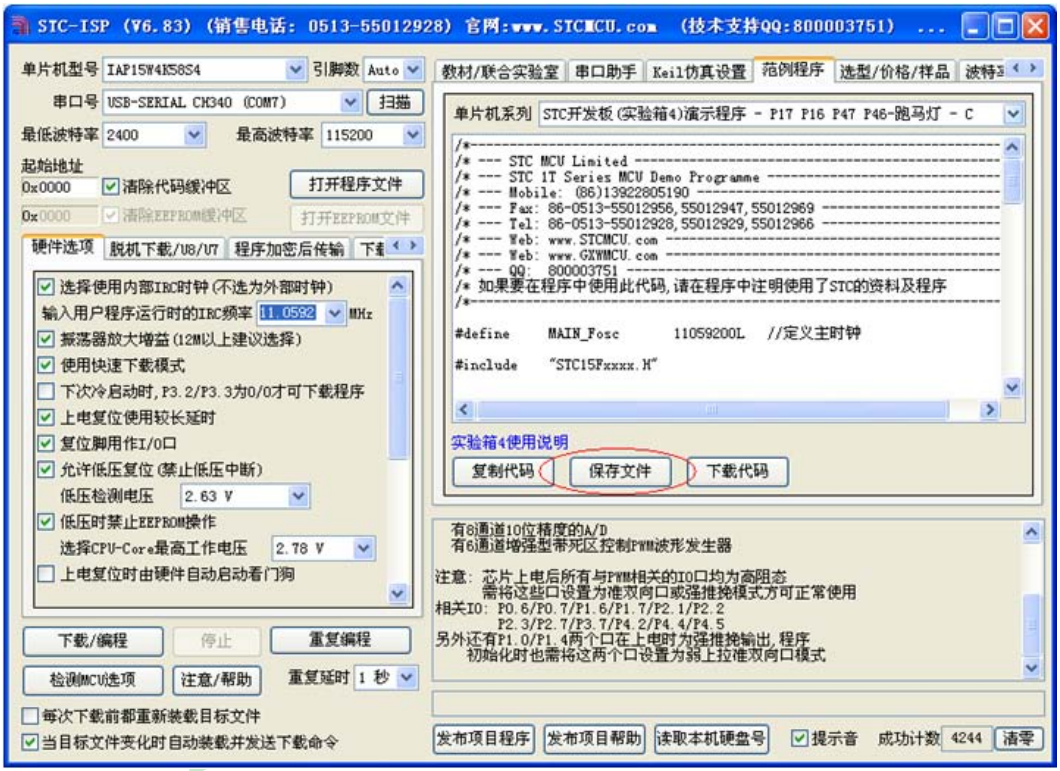
至此，基本的项目文件已基本建立。

接下来打开STC的ISP下载软件，如下图：

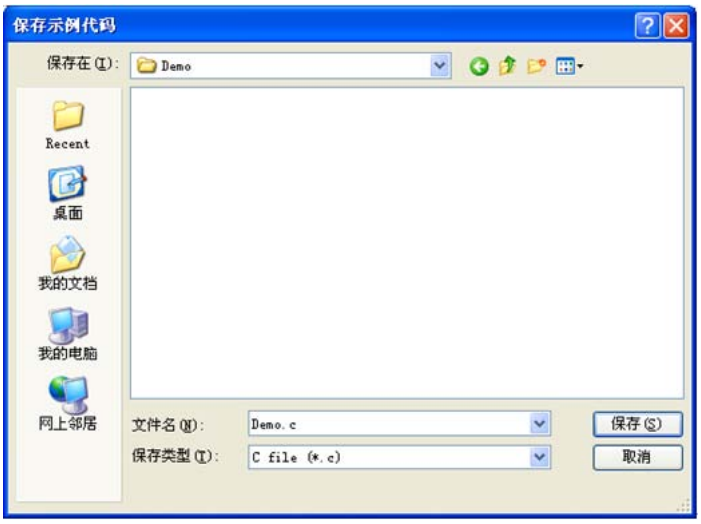
选择软件右边功能模块中的“范例程序”页，然后在下列列表中选择一个范例（我们以“STC开发板（实验箱4）演示程序”中“P17 P16 P47 P46-跑马灯”的C语言代码为例）



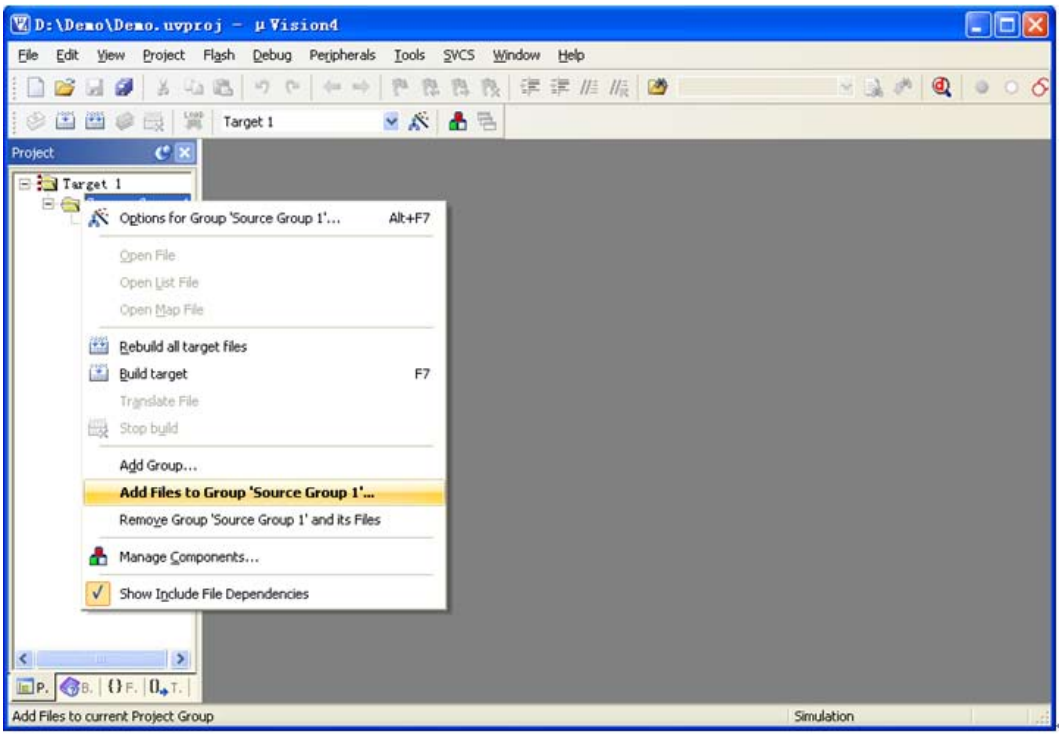
选择完成后点击“范例程序”页中的“保存文件”按钮对文件进行保存，如下图：



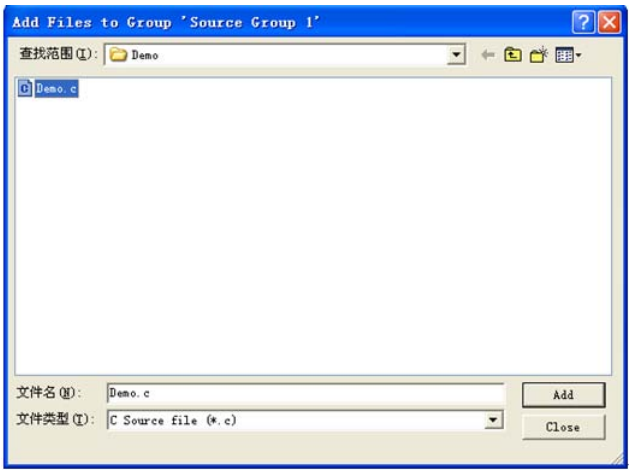
将文件保存到我们前面所建项目的目录中



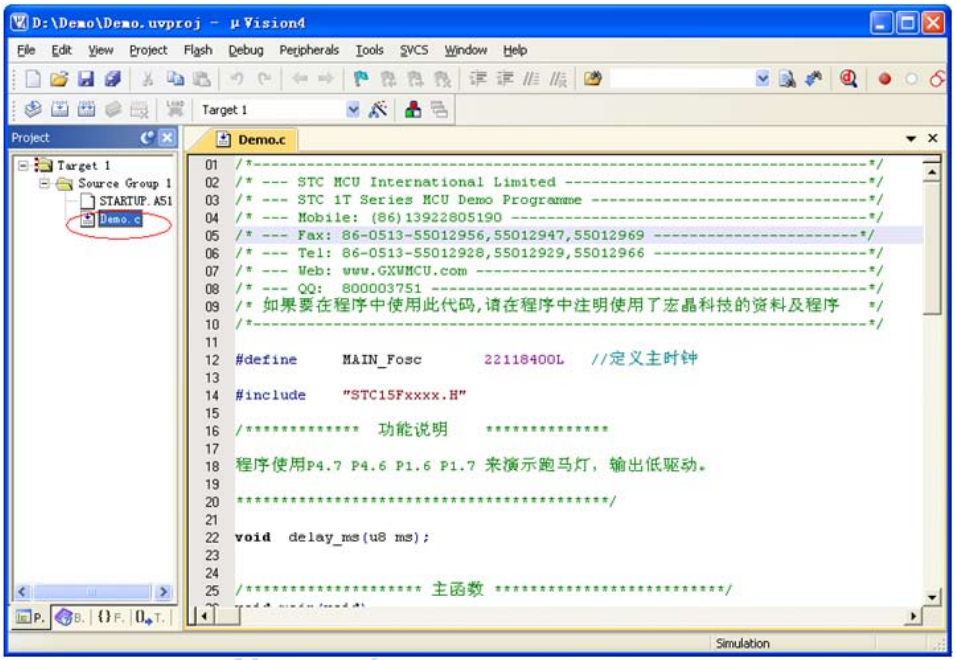
文件保存完成后需要使用下面的操作将源代码文件添加到项目中来，具体的操作方法是：使用鼠标右键单击“Project”列表中的“Source Group 1”项，在出现的右键菜单中选择 “Add Files to Group ‘Source Group 1’ ” 项目



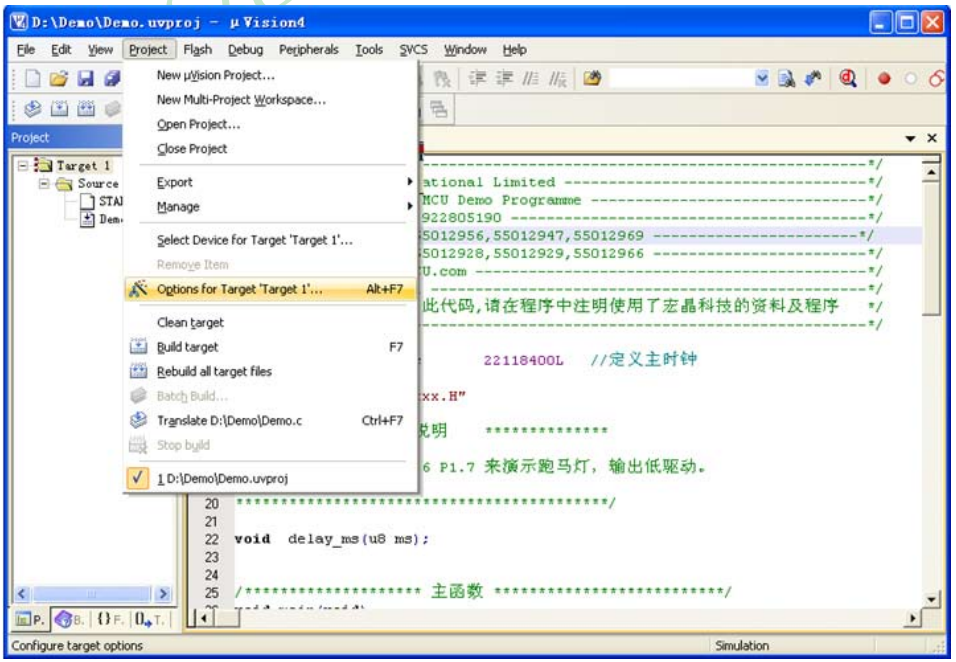
在下面的对话框中选择我们刚才保存的文件，并点击“Add”按钮即可将文件添加到项目中，完成后按下“Close”按钮关闭对话框



此时我们可以看到在项目中已经多了我们刚才添加的代码文件，打开文件即可看到我们刚才从ISP下载软件中保存的文件，如下图

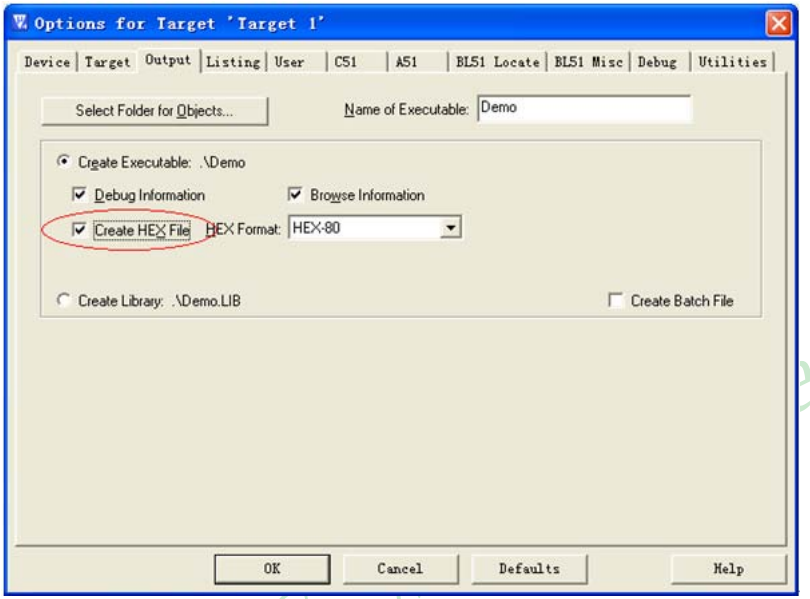


按下快捷键“Alt+F7”或者选择菜单“Project”中的“Option for Target ‘Target1’”

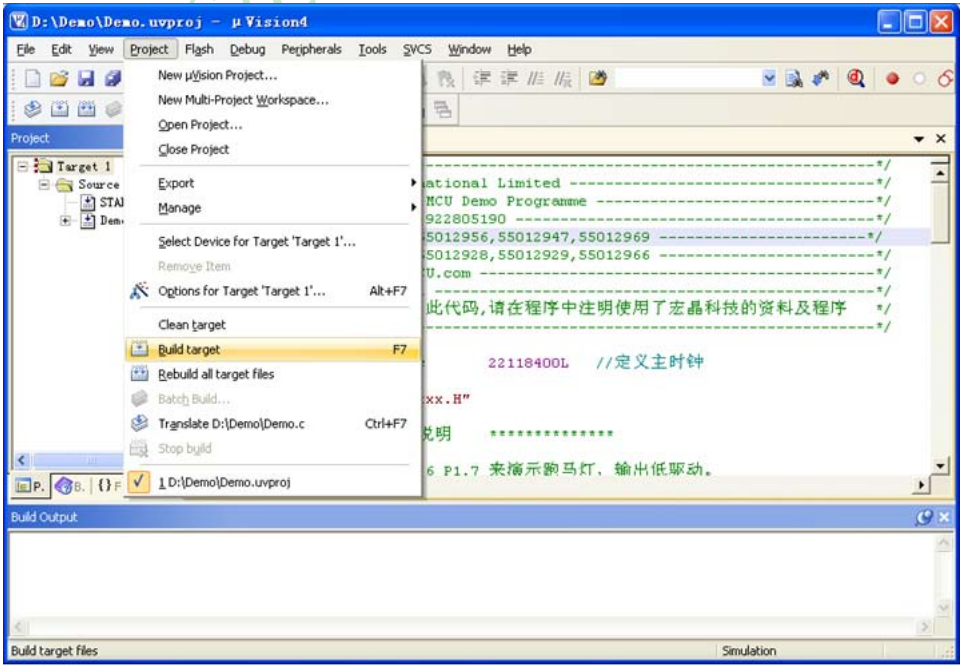


在如下的对话框中对项目进行配置：

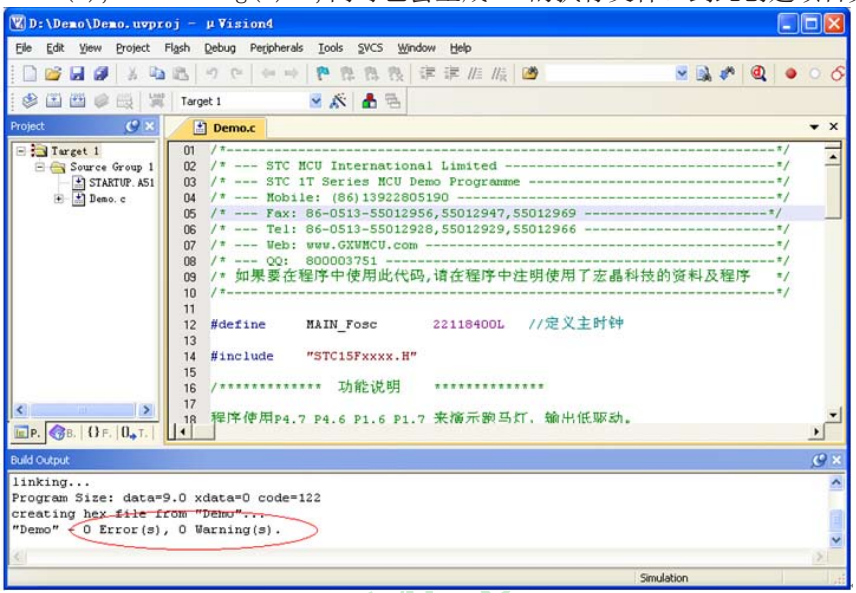
在“Output”属性页中，将“Create HEX FIle”选项打上勾，即可在项目编译完成后自动生成HEX格式的目标文件，按“OK”保存。



按下快捷键“F7”或者选择菜单“Project”中的“Build Target”项对当前项目进行编译



若代码中没有错误，编译完成后则会在“Build Output”的信息输出框中显示“0 Error(s), 0 Warning(s)”,同时也会生成HEX的执行文件。到此创建项目完成。



G.5 下载用户程序到STC实验箱4

下面我们以刚才新建的项目“Demo”为例，将编译后生成的HEX文件下载到STC实验箱4。

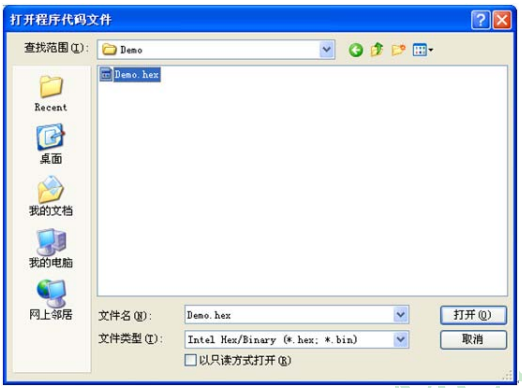
首先使用USB线将STC实验箱4与电脑正确连接，然后打开STC的ISP下载软件（例如：“STC-ISP (Ver6.83)”）

在上面的界面中，下面几点需要注意：

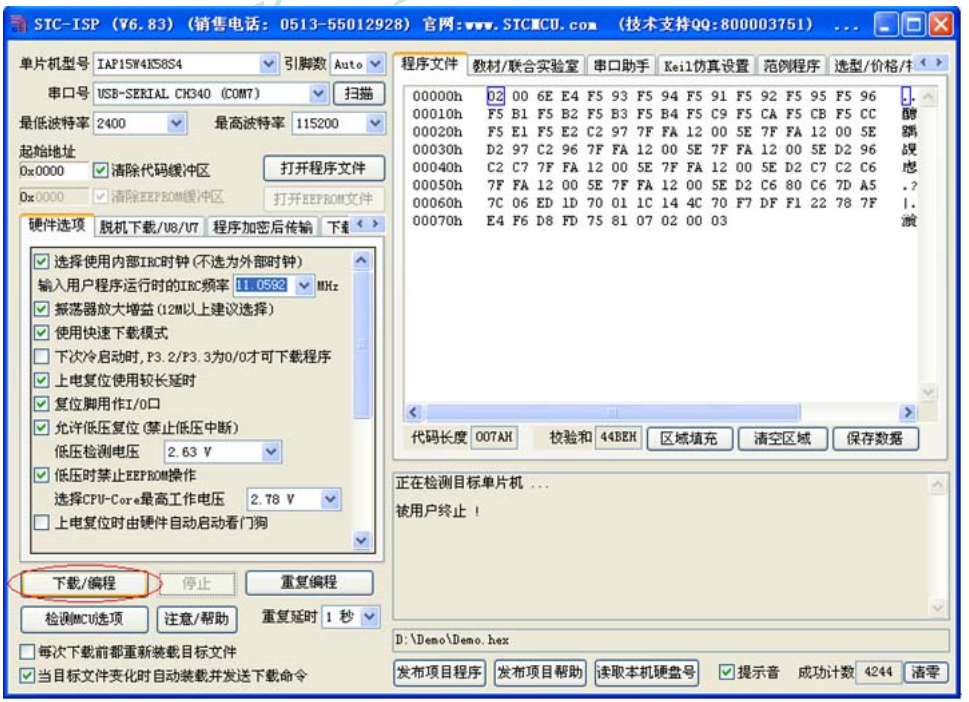
1、单片机型号必须选择“IAP15W4K58S4”(因为实验箱4中的主控芯片都是IAP15W4K58S4)

2、 串口口必须选择实验箱4所对应的串口号（当实验箱4与电脑正确连接后，软件会自动扫描并识别名称为“USB-SERIAL CH340 (COMx)”串口，具体的COM编号会因电脑不同而不同）。当有多个CH340类型的USB转串端口与电脑相连时，则必须手动选择。

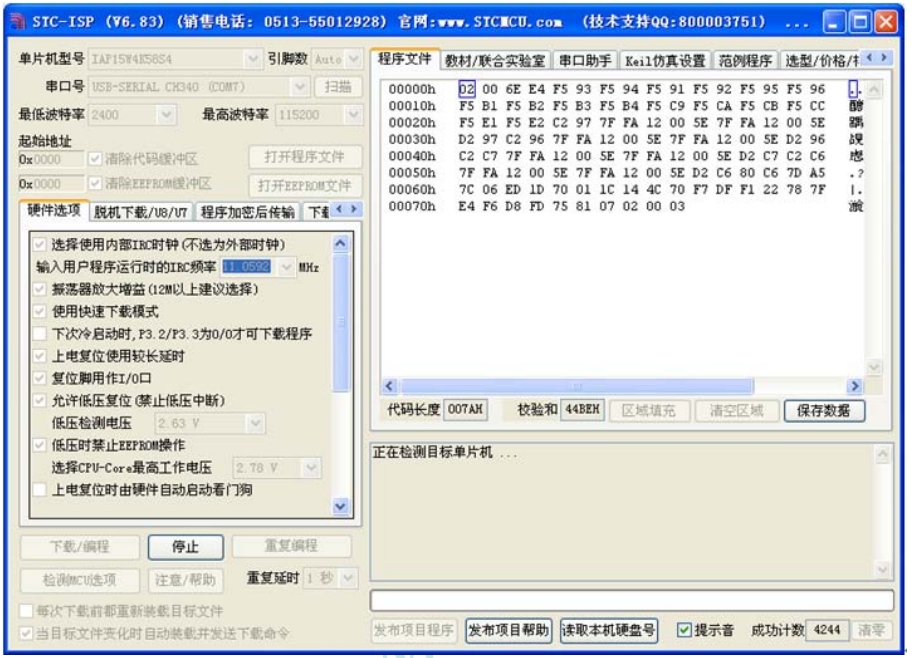
点击界面中的“打开程序文件”按钮，在出现的打开程序代码文件的对话框中选择需要下载的文件（这里以我们前面所建立的项目为例）



文件正确打开后，点击界面中的“下载/编程”按钮开始下载代码

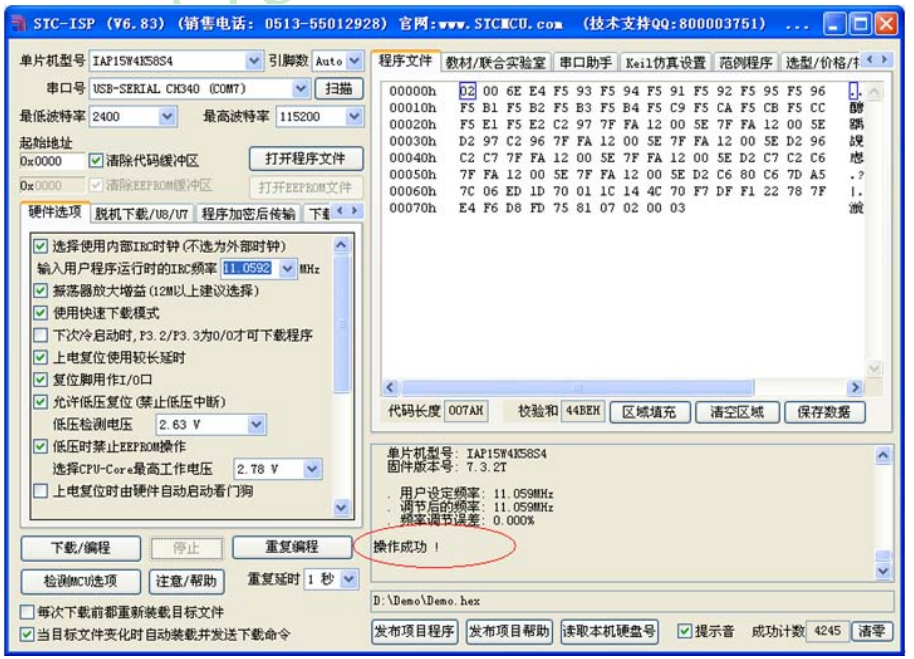


如下图：



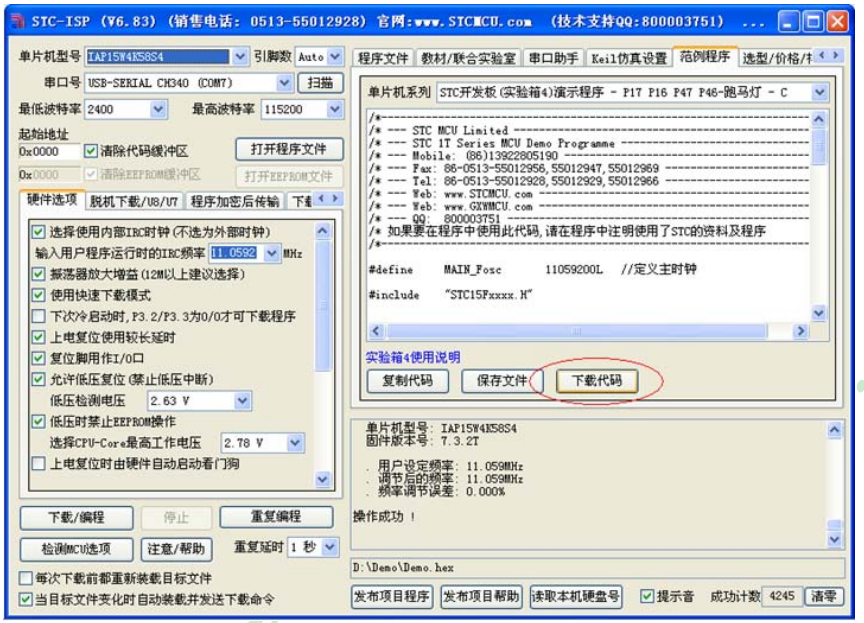
接下来需要按下实验箱4上的“主控芯片电源开关”，然后松开即可开始下载

若下载成功，会出现如下的画面

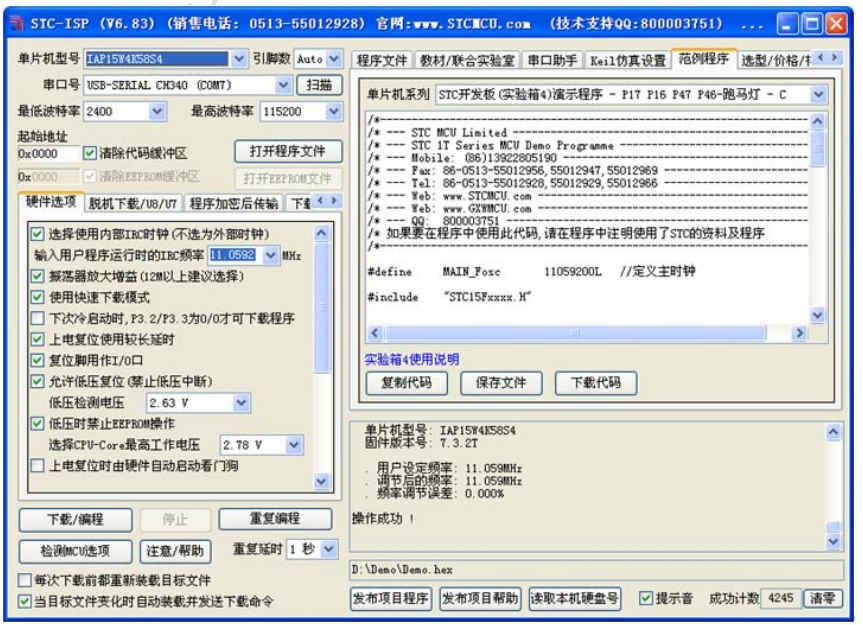


G.6 直接下载STC-ISP范例程序到STC实验箱4

STC的ISP软件中，支持对软件中的范例程序进行直接下载，直接下载按钮如下图所示



在下载之前依然需要进行如下步骤的操作：首先使用USB线将STC实验箱4与电脑正确连接，然后打开STC的ISP下载软件（例如：“STC-ISP (Ver6.83)”）

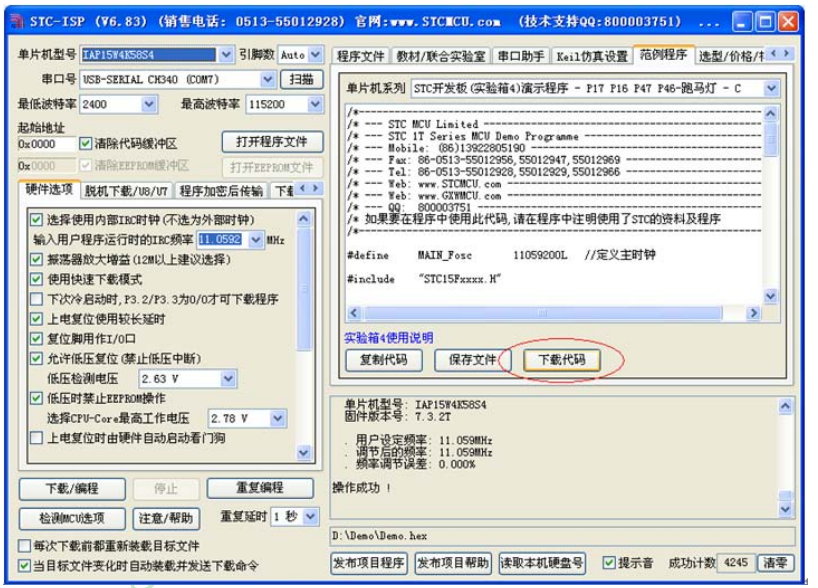


在上面的界面中，下面几点需要注意：

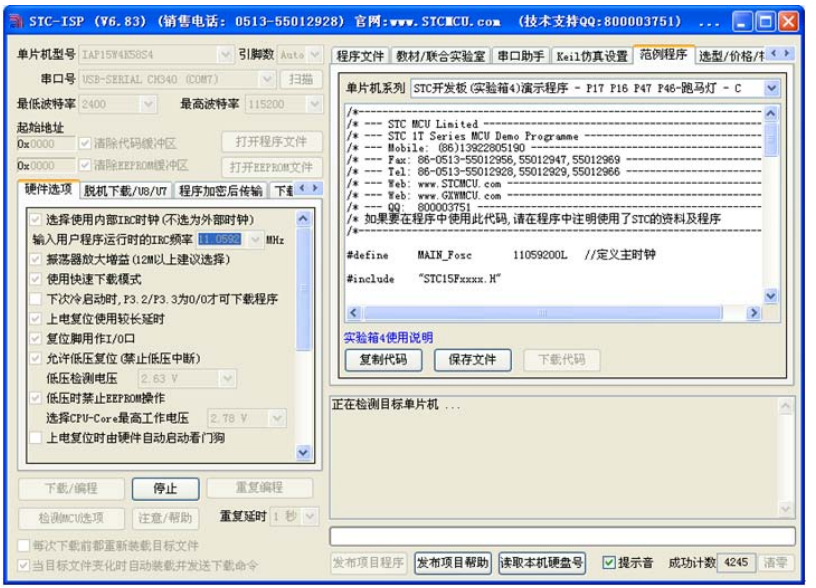
1、 单片机型号必须选择“IAP15W4K58S4”(因为实验箱4中的主控芯片都是IAP15W4K58S4)

2、 串口口必须选择实验箱4所对应的串口号（当实验箱4与电脑正确连接后，软件会自动扫描并识别名称为“USB-SERIAL CH340 (COMx)”串口，具体的COM编号会因电脑不同而不同）。当有多个CH340类型的USB转串端口与电脑相连时，则必须手动选择。

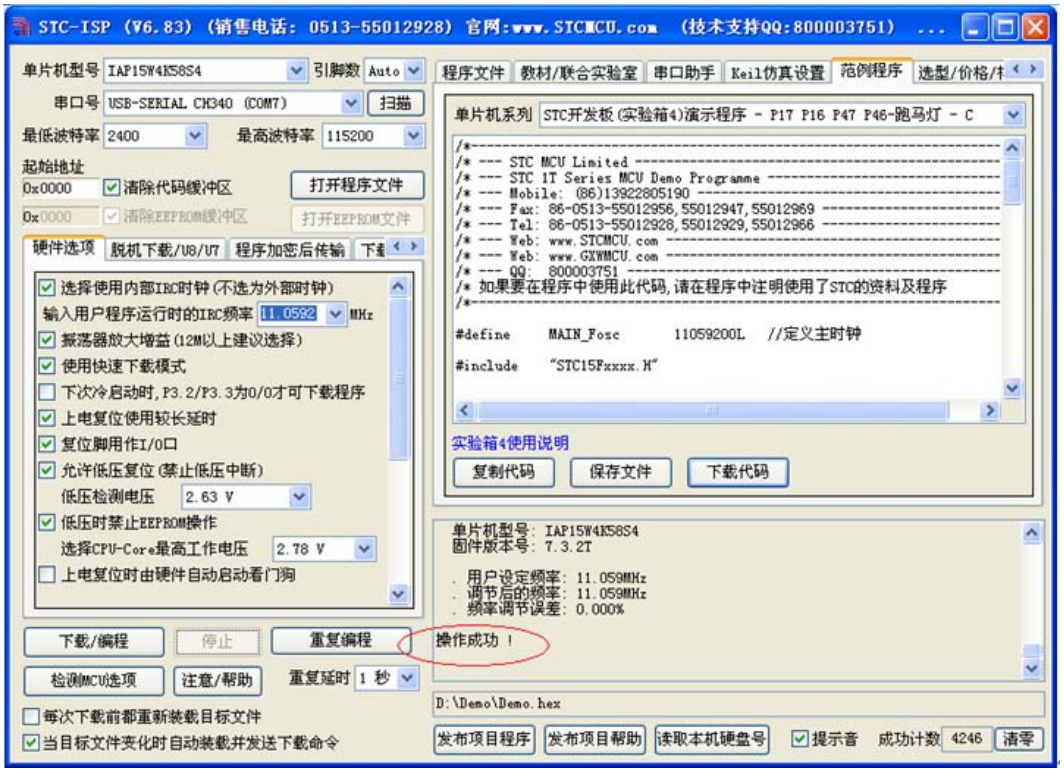
然后点击界面中“范例程序”页面中的“下载代码”按钮开始下载代码



如下图



接下来需要按下实验箱4上的“主控芯片电源开关”，然后松开即可开始下载。若下载成功，会出现如下的画面：



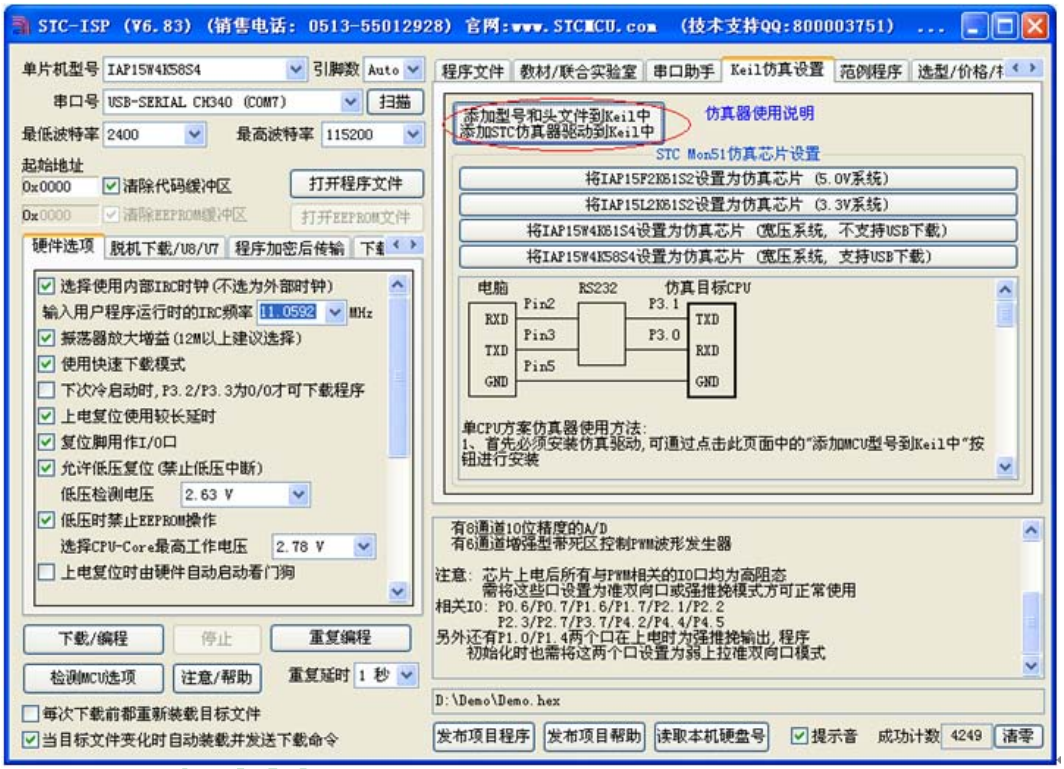
G.7 使用STC实验箱4仿真用户代码

目前STC的仿真都是基于Keil环境的，所以若需要使用STC实验箱4仿真用户代码，则必须要安装Keil软件。Keil的uVersion2、uVersion3和uVersion4都可以（uVersion5好像不支持8051）。建议安装使用uVersion4。

Keil软件安装完成后，还需要安装STC的仿真驱动。STC的仿真驱动的安装步骤如下：

首先开STC的ISP下载软件；

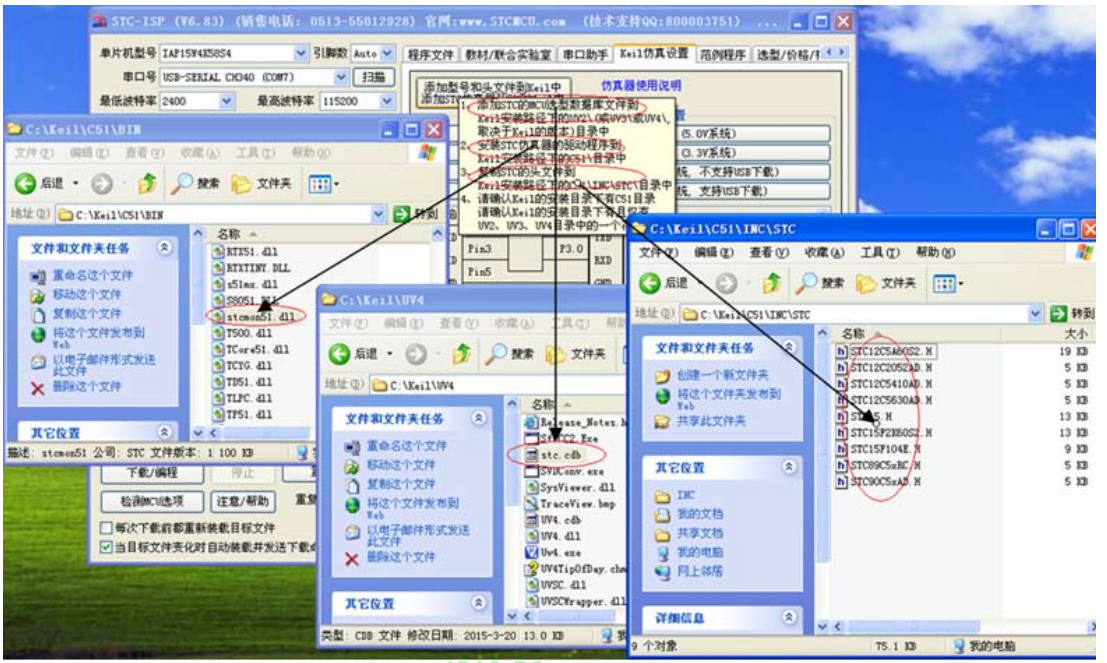
然后在软件右边功能区的“Keil 仿真设置”页面中点击“将IAP15W4K58S4设置为仿真芯片”按钮。



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右图的提示框

在Keil的相关目录中可以看到如下的文件，即表示驱动正确安装了

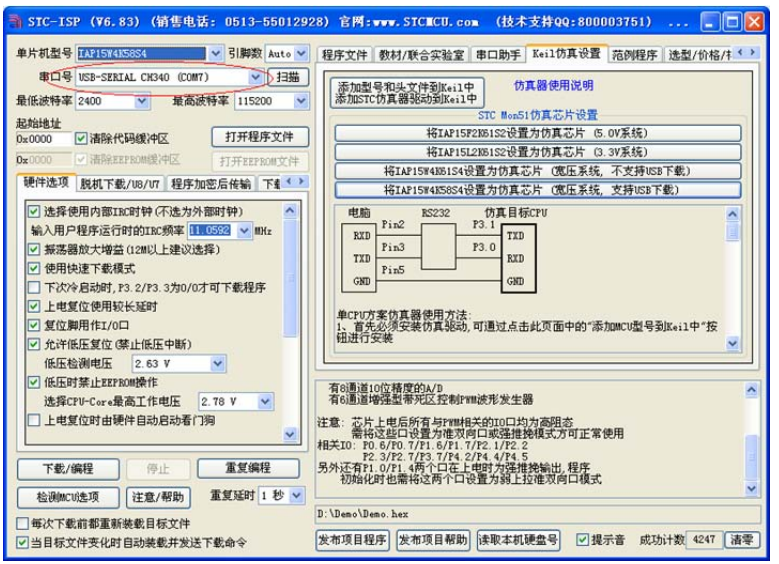


由于在默然状态下，实验箱4的主控芯片并不是一颗仿真芯片，不具有仿真功能，所以若需要使用实验箱4进行仿真，则还需要将实验箱4的主控芯片设置为仿真芯片。

制作仿真芯片步骤如下：

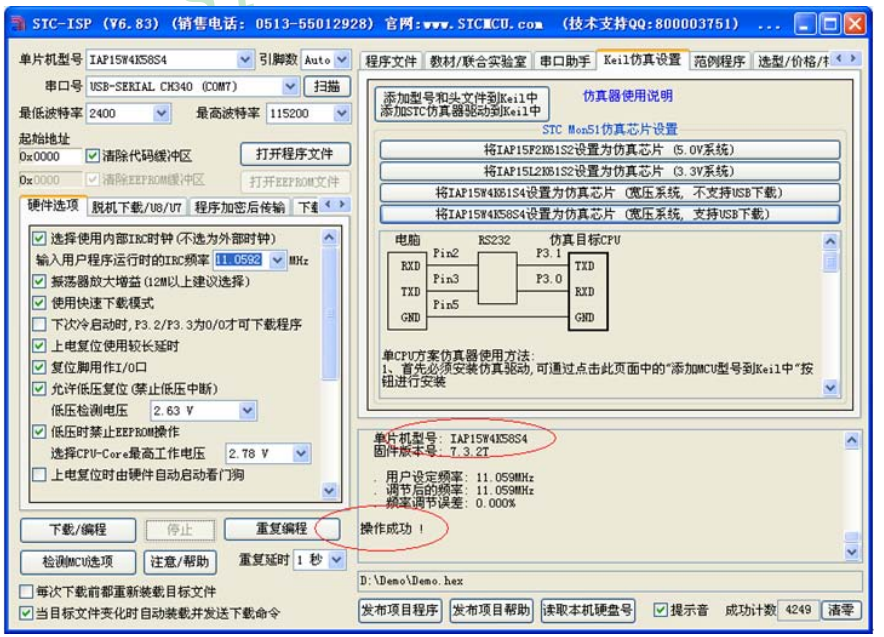
首先使用USB线将实验箱4与电脑进行连接；

然后打开STC的ISP下载软件，并在串口号的下拉列表中选择实验箱4所对应的串口号。单片机型号可以不进行选择，在制作仿真芯片时，软件会自动选择“IAP15W4K58S4”型号。



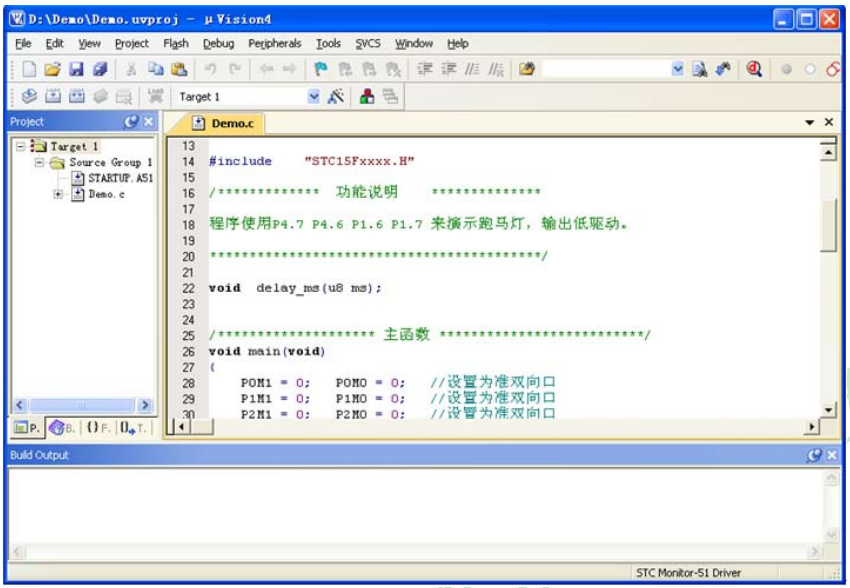
然后在软件右边功能区的“Keil 仿真设置”页面中点击“将IAP15W4K58S4设置为仿真芯片”按钮，按下后会出现如下画面

接下来需要按下实验箱4上的“主控芯片电源开关”，然后松开即可开始制作仿真芯片。若设置成功，会出现如下的画面



到此，仿真芯片便制作成功了。

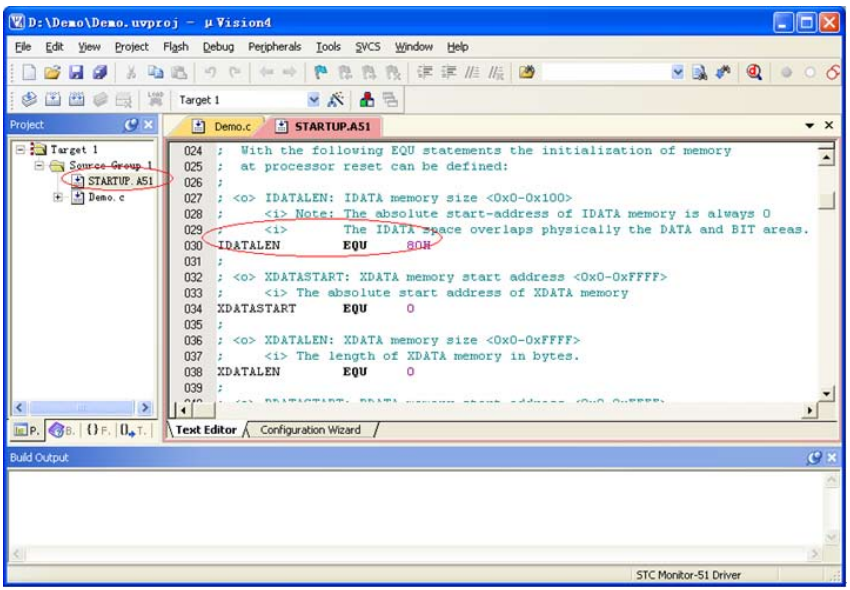
接下来我们打开之前建立的项目



然后进行下面的项目设置

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。



我们所选的STC15W4K32S4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。

按下快捷键“Alt+F7”或者选择菜单“Project”中的“Option for Target ‘Target1’”，在“Option for Target ‘Target1’”对话框中对项目进行配置

第1步、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 …”

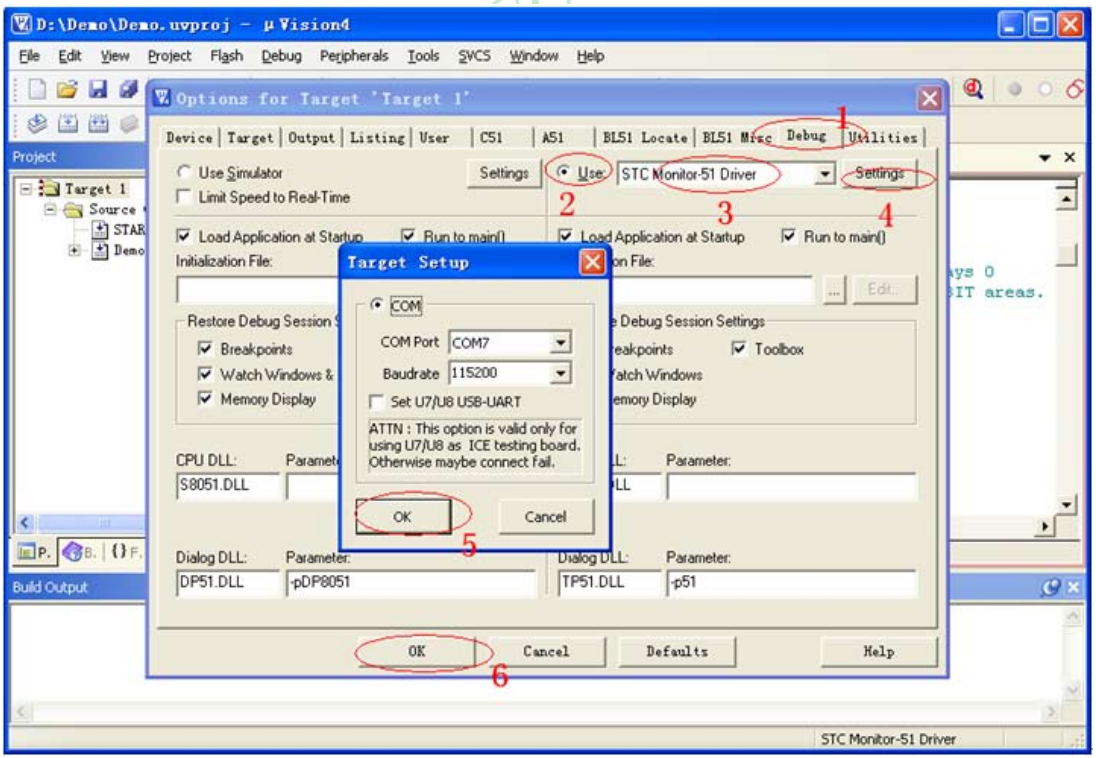
第3步、在仿真驱动下拉列表中选择“STC Monitor-51 Driver”项

第4步、点击“Settings”按钮，进入串口的设置画面

第5步、对串口的端口号和波特率进行设置，串口号要选择实验箱4所对应的串口，波特率一般选择115200或者57600。

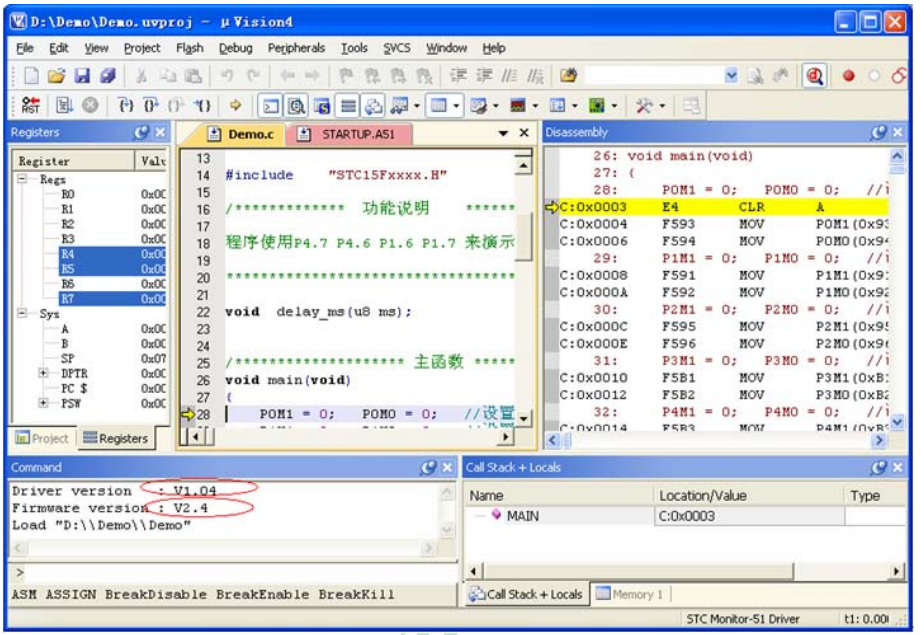
第6步、点击“确定”完成仿真设置。

详细步骤如下图所示：

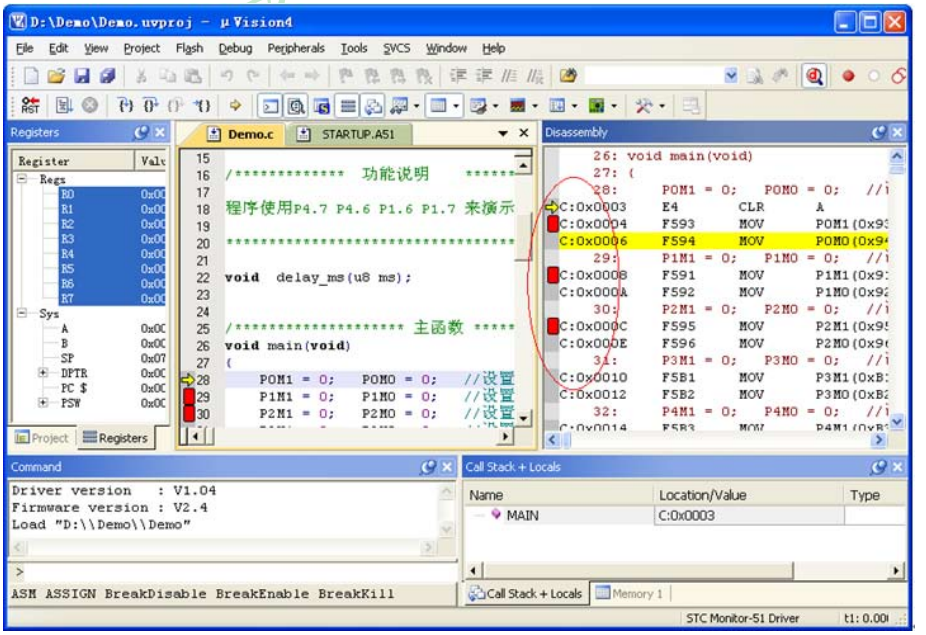


完成了上面所有的工作后，即可在Keil软件中按“Ctrl+F5”开始仿真调试。

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，如下图所示：



仿真调试过程中，可执行复位、全速运行、单步运行、设置断点等多中操作。

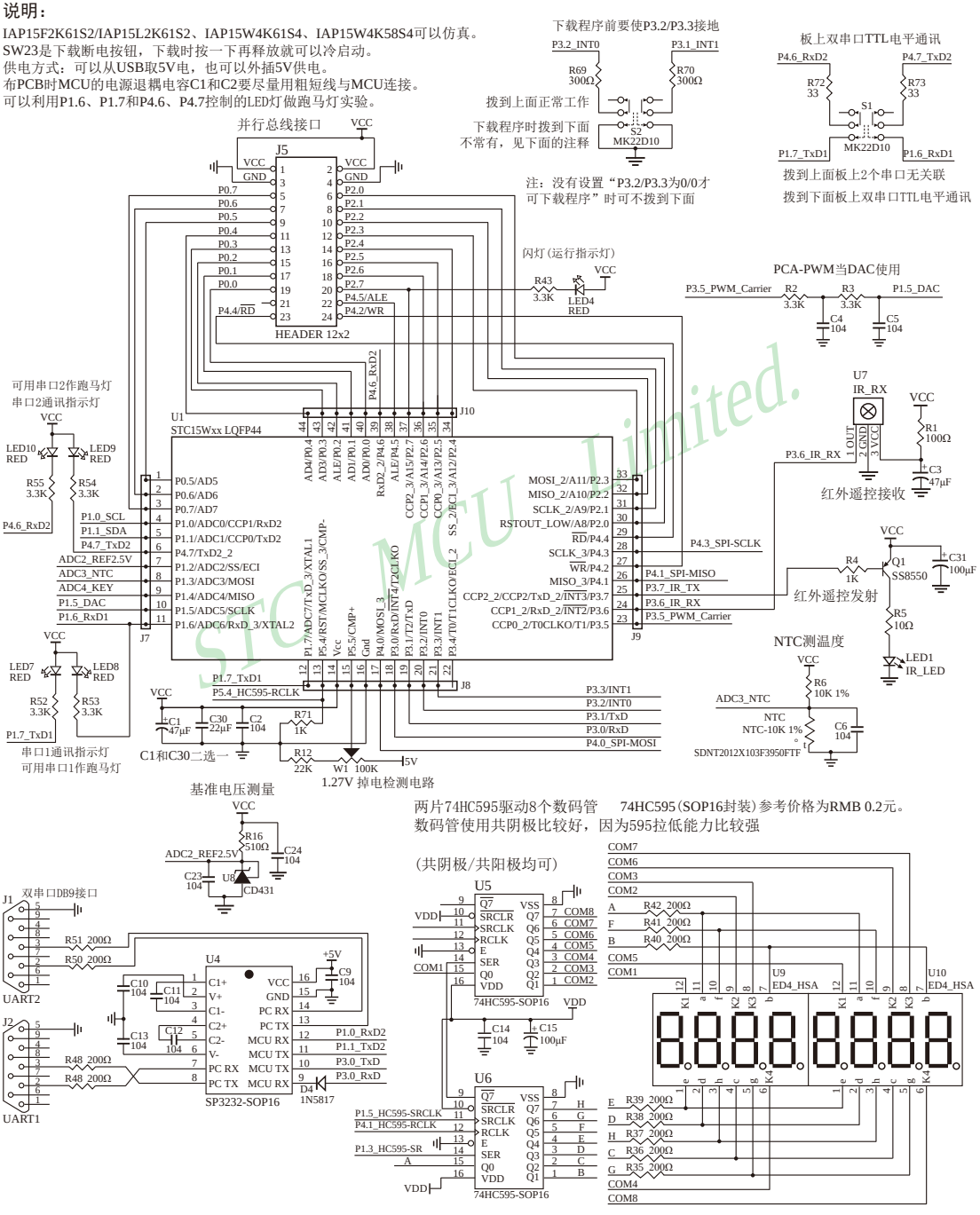


如上图所示，可在程序中设置多个断点，断点设置的个数目前最大允许20个（理论上可设置任意个，但是断点设置得过多会影响调试的速度）。

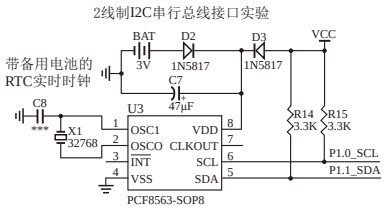
G.8 STC实验箱4参考线路图

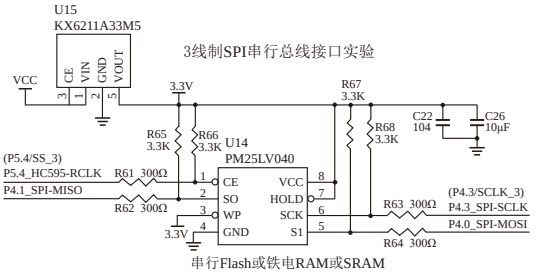
说明:

IAP15F2K61S2/IAP15L2K61S2、IAP15W4K61S4、IAP15W4K58S4可以仿真。
SW23是下载断电按钮，下载时按一下再释放就可以冷启动。
供电方式：可以从USB取5V电，也可以外插5V供电。
布PCB时MCU的电源退耦电容C1和C2要尽量用粗短线与MCU连接。
可以利用P1.6、P1.7和P4.6、P4.7控制的LED灯做跑马灯实验。

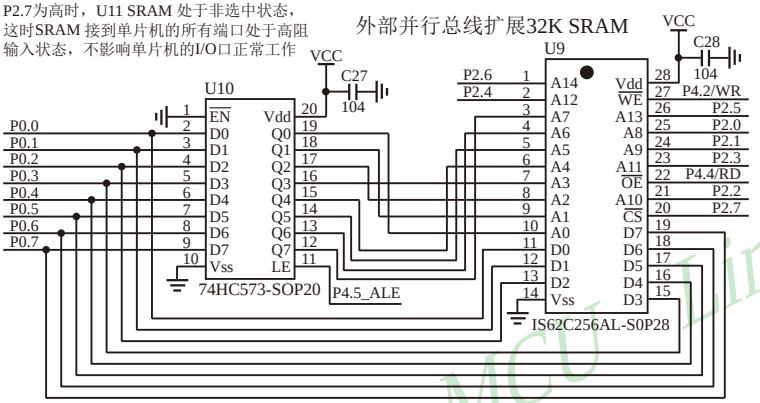


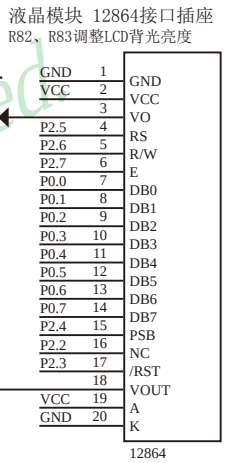
STC实验箱4参考线路图续



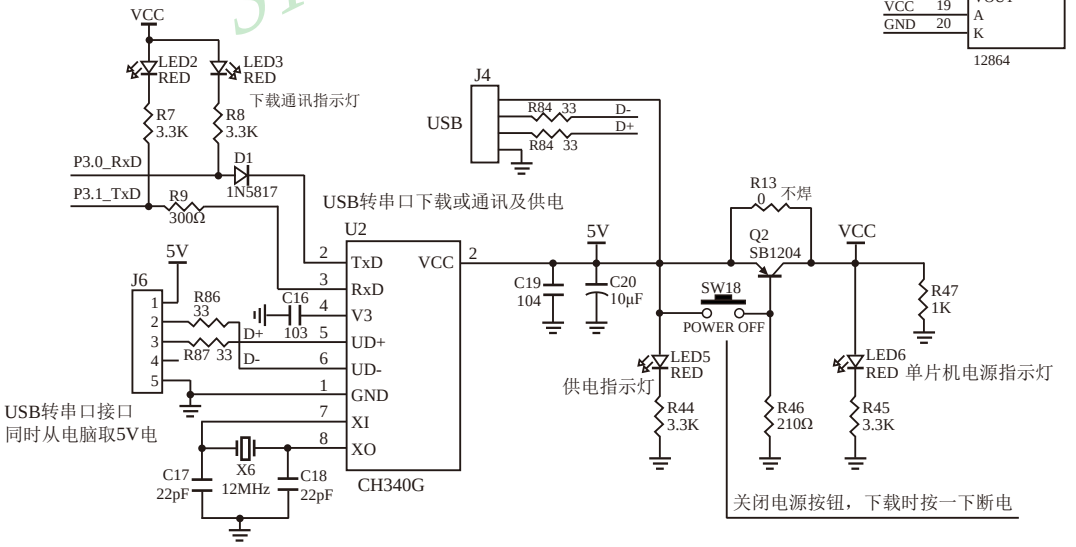


P2.7为高时，U11 SRAM 处于非选中状态，这时SRAM 接到单片机的所有端口处于高阻输入状态，不影响单片机的I/O口正常工作

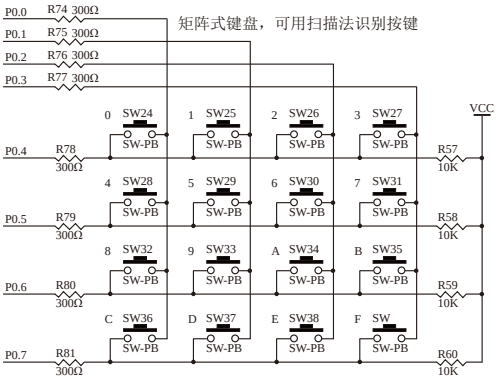




注意：由于历史原因，IS62C256AL-S0P28封装比STC的SOP28封装要宽

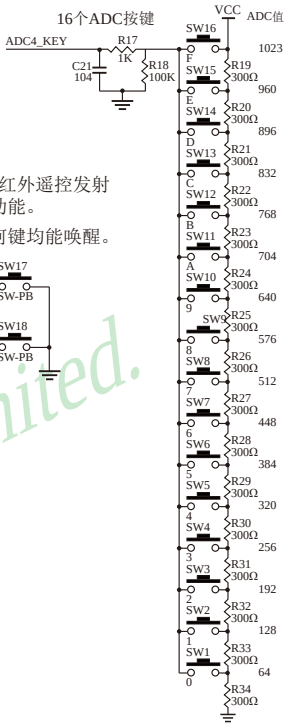


STC实验箱4参考线路图续



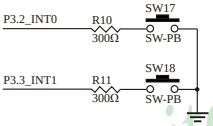
读ADC键的方法：

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。

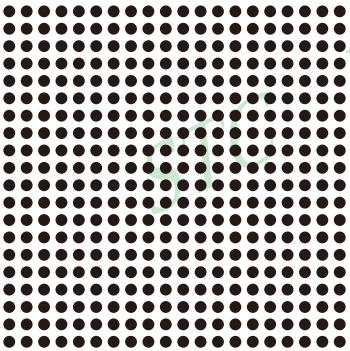


2个可唤醒按键，可作红外遥控发射或接收学习，或别的功能。

进入睡眠之后按下任何键均能唤醒。



PCB板上根据情况留一些过孔焊盘方便做实验(图例20x20)



附录H：STC大学计划—联合实验室

STC创始人决定投入巨资在全国高校建立1000所"STC高性能单片机联合实验室"，第一批院校300所,主要面向一本/二本院校,申请院校满足:

1.每年有100名以上新生学习STC可仿真的 1T 8051单片机

2.采用STC推荐的基于可仿真芯片的教材<或您自己编写>

3.有固定场所挂牌作为 "STC高性能单片机联合实验室"

4.学院盖章提供基本情况证明

5.即可按4:1的比率获赠基于STC可仿真的IAP15F2K61S2单片机学习实验箱<学生4:实验箱1>

6.如愿意导入STC制定/教育部EITP中心推广的MCU考试标准<我们免费提供考题,学校自己考>,将可获得长期赞助

联系邮箱: STC13922805190@qq.com

联系电话:13922805190,下面是推荐教材和实验箱资料

在国内多所大学建立了联合实验室，已建或在建STC高性能单片机联合实验室高校名单：

上海交通大学、复旦大学、同济大学、浙江大学、南京大学、东南大学、吉林大学、中山大学、山东大学、四川大学、中南大学、湖南大学、哈尔滨工业大学、南开大学、天津大学、东北大学、厦门大学、兰州大学、西安交通大学、西北工业大学、西北农林科技大学、中国农业大学、中国海洋大学、中央民族大学、北京航空航天大学、南京航空航天大学、沈阳航空航天大学、南昌航空大学、北京理工大学、大连理工大学、南京理工大学、武汉理工大学、华东理工大学、太原理工大学、上海理工大学、东华理工大学、哈尔滨理工大学、哈尔滨工程大学、合肥工业大学、天津工业大学、河南工业大学、北京工业大学、北京化工大学、北京工商大学、华北电力大学、西南交通大学、东华大学、上海大学、长安大学、南昌大学、福州大学、安徽大学、苏州大学、江南大学、河海大学、扬州大学、南通大学、宁波大学、深圳大学、东北林业大学、大连海事大学、杭州电子科技大学、桂林电子科技大学、成都电子科技大学、南京邮电大学、西安邮电大学、天津财经大学、中国石油大学、中国矿业大学等国内著名高校。上海交通大学/西安交通大学/浙江大学/山东大学/成都电子科技大学等著名高校的多位知名教授使用STC 1T 8051创作的全新教材也在陆续推出中。多所高校每年都有用STC单片机进行的全校创新竞赛，如杭州电子科技大学/湖南大学/东南大学/山东大学等。


STC高性能微控制器
联合实验室
STC High Performance
Microcontroller Joint Lab


天津大学
Tianjin University
STC micro
宏晶科技
STC高性能单片机
联合实验室
STC High Performance MCU
Joint Laboratory


STC micro
宏晶科技
STC高性能单片机
联合实验室
STC High Performance MCU
Joint Laboratory


STC高性能单片机
联合创新实验室
STC High Performance MCU
Joint Innovation Laboratory


STC高性能单片机
联合实验室
STC High Performance MCU
Joint Laboratory


STC micro
宏晶科技
STC高性能微控制器
联合实验室
STC High Performance
Microcontroller Joint Lab


杭州电子科技大学
STC高性能单片机联合实验室


STC micro
宏晶科技
STC高性能微控制器
联合实验室
STC High Performance
Microcontroller Joint Lab


STC micro
宏晶科技
STC高性能单片机
联合实验室
STC High Performance MCU
Joint Laboratory


STC micro
宏晶科技
STC高性能单片机
联合实验室
STC High Performance MCU
Joint Laboratory

附录I：逻辑代数的基础

——无微机原理基础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。

I.1 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。

I.1.1 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=k0
2 | 106 ------ 余数=0=k1
2 |  53 ------ 余数=1=k2
2 |  26 ------ 余数=0=k3
2 |  13 ------ 余数=1=k4
2 |   6 ------ 余数=0=k5
2 |   3 ------ 余数=1=k6
2 |   1 ------ 余数=1=k7
    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2^4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(010111011110.11010010)B转换为十六进制数：

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

于是，(010111011110.11010010)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。

I.1.2 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

I.1.3 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。

I.2 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： 

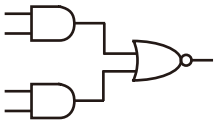
六：与或非运算及与或非图形符号

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

(接上表)

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： 

附录J：STC对单片机相关学科群部分课程改革呼吁

工信部一位对推动中国单片机教学改革有兴趣的朋友约我对这方面提一点看法，与高校及企业合作多年的经验，我确实也深有感触。

STC自成立起就一直专注于国内流行的8051单片机的研发工作，和各高校以及企业保持着密切的联系和紧密的合作。根据我们STC与高校及企业合作多年的经验，我们深刻感受到当前大学里单片机课程教育存在的弊端，主要是开设的时间和顺序以及完整性，不需要增加教学时间，还可适当减少课时，当前单片机课程教育的改革迫在眉睫。对此我们提出以下建议。

单片机相关课程教学之最小系统课程：

《C语言程序设计》	大学一年级第一学期
《数字电路基础》	大学一年级第一学期
《微机原理及单片机应用》	大学一年级第二学期
《数据结构》	大学一年级第二学期
《实时操作系统》	大学二年级第一学期
《模拟电路》	大学二年级第一学期

首先，我们建议《C语言程序设计》课程必须保证在大学一年级第一学期开设，很多高校已这样做了，但还有部分学校滞后了，我的观念晚学不如早学，工科非工科都应将此门课作为进入信息时代的计算机扫盲课（word/excel/打字就由学生自学吧），学好了《C语言程序设计》课程就打好了单片机学习中的部分软件基础。相对于用汇编语言开发程序，C语言也将程序开发难度降低了很多，对开发人员来说，效率得到极大的提高，打个不恰当的比方，

大专生就相当于本科生的能力了，本科生就相当于研究生的能力了。

其次，我们强烈建议将《数字电路》课程提前放在大学一年级第一学期开设，《数字电路》能帮助学生理解微机原理和基本的数字系统，并能设计一些简单的硬件电路系统，是单片机学习的硬件基础。

再次，就是要学习《微机原理及单片机应用》了，我们强烈建议将《微机原理及单片机应用》课程提前到大学一年级第二学期开设。许多学校的相关工科专业都是在大学三年级第二学期才开设这门课程。由于接触单片机的时间晚，学生根本没时间认真学习和研究单片机，更没时间应用所学的知识开发一些单片机项目。在许多学生只学到一些单片机皮毛的情况下，他们便匆匆步入社会进入了相关行业。这种情况往往导致很多学生不能胜任自己的岗位，工作起来很吃力，而且极容易被淘汰。这就有了“学生抱怨就业难，企业却反映招不到人才”的怪现象。如果将《微机原理及单片机应用》课程提前至大学一年级第二学期开设，学生及早接触单片机的话，那么对单片机有兴趣或以后想从事相关行业的同学在意识到单片机的重要性后，仍然有充足的时间去研究单片机，从而将单片机系统设计搞熟。这样在他们毕业以后进入相关行

业及企业工作就会很快能胜任。单片机课程不是一门纯理论性的课程，更多的需要学生去动手实践和开发，所以学习单片机以及开发单片机项目能提高学生的创新能力，如果学得太晚就很难学以致用。其实很多工科的专业课程都如此，很多学生在学习完这些专业课程后之所以云里雾里、不知所云，就是因为没有花时间或没有足够的时间去研究和实践应用所学的内容。如果这些动手实践型的专业课程开设的太晚，学生在学习完之后根本没有时间去动手应用的话，那么这些课程就会形同虚设，对学生以后的就业将起不到应有的帮助作用。

另外，为了使学生在开发程序时，能得心应手，我们建议在大学一年级第二学期应再开设《数据结构》课程，提高开发效率，降低开发难度，同时如果《C语言程序设计》没学好，通过对《数据结构》的学习，也可算对《C语言程序设计》的复习，为单片机开发打好软件基础。还是那个不恰当的比方，有《数据结构》作为基础比没有《数据结构》作为基础的，大专生就相当于本科生的能力了，本科生就相当于研究生的能力了。

最后，我们建议在大学二年级第一学期开设《实时操作系统》和《模拟电路》等后续课程。这些课程是对前面所学课程的补充和提高，如果《数据结构》没学好，学习《实时操作系统》也可算对《数据结构》的复习，为单片机开发打好软件基础。基于实时操作系统的单片机开发，对开发者的能力要求可大幅降低，能较容易开发出大型、复杂的单片机项目。还是那个不恰当的比方，有《实时操作系统》作为基础比没有《实时操作系统》作为基础的，大专生就相当于本科生的能力了，本科生就相当于研究生的能力了。

有了以上基础，相信大学生创新竞赛会更有价值！

乘风破浪会有时，直挂云帆济沧海。

相信单片机课程教育的改革会给当前的单片机教育和科技创新开创一片新的天地，相信各高校一定能培养出更多、更专业、更符合社会需求的优秀人才。

附录K：STC对单片机课程教育改革的贡献

当前绝大部分高校都以8051单片机作为单片机课程教育的基础，但普通的8051单片机诞生与上世纪70年代，不可避免地面临着落伍的危险。为此，STC宏晶科技对8051单片机进行了全面的技术升级与创新，对当前单片机课程教育的改革有巨大的贡献。

宏晶科技STC单片机设计公司是全球最大的8051单片机设计公司，致力于开发设计1T增强型8051单片机，速度平均比普通8051快7~12倍，指令代码完全兼容普通8051单片机。STC单片机全部采用Flash技术(可反复编程10万次以上)和ISP/IAP(在系统可编程/在应用可编程)技术，并对传统8051进行了全面提速，指令最快提高了24倍。另外，STC针对抗干扰进行了专门设计，针对加密进行了特别加密设计，如宏晶STC15系列超强抗干扰，现无法解密。其次，STC单片机大幅提高了集成度，如集成了A/D、CCP/PCA/PWM(PWM还可当D/A使用)、高速同步串行通信端口SPI、高速异步串行通信端口UART(如宏晶STC15F2K60S2系列集成了两个串行口，分时复用可当5组串口使用)、定时器(宏晶STC15F2K60S2系列最多可达到6个定时器)、看门狗、高可靠复位电路(可彻底省掉外部复位)、内部高精准时钟(-40℃ ~ +85℃之间最大只有±1%的温飘，可彻底省掉外部昂贵的晶振)、大容量SRAM(如宏晶STC15F2K60S2系列集成了2K字节的SRAM)、大容量EEPROM、大容量Flash程序存储器等。STC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,这是宏晶科技STC名称的由来)。可以说，STC单片机来源于普通8051单片机，却又高于普通8051单片机。如果学生或单片机爱好者以STC单片机作为学习的工具，将会对8051单片机有个更全面更透彻的掌握。

STC单片机除对普通8051单片机的技术技术进行升级外，还对普通8051单片机一些复杂难懂的知识点进行了简化。例如，对初学者而言，定时器T0/T1的四种工作模式只需学习其中的模式0(16位自动重装载模式)，定时器0的模式3(不可屏蔽中断的16位自动重装载模式)还可作实时操作系统节拍定时器，定时器2也只需学习一种模式。另外，串行口的波特率计算公式也比普通8051的计算公式简单的多。例如，我们用定时器2作为串行口的波特率发生器，则串行口的波特率计数公式为：

串行口的波特率=(定时器T2的溢出率)/4 ； 注意：此波特率与SMOD无关

……对教学而言，这些简化减少了教学的课时，从而大大减轻了教学压力。

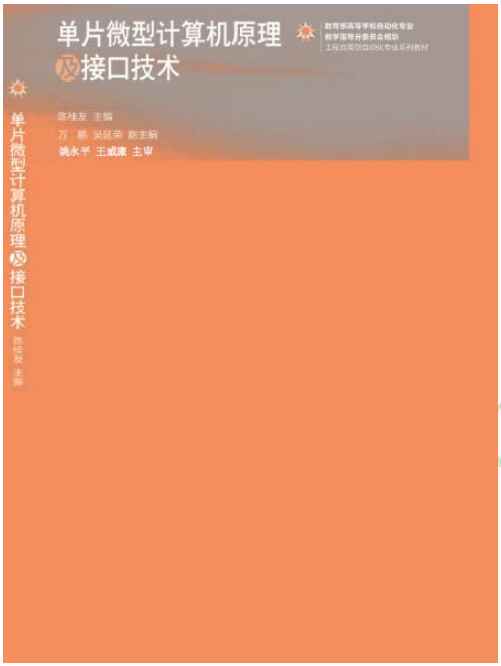
STC公司设计生产的一些辅助工具，如STC-ISP下载编程工具，能够更好地帮助学生和单片机爱好者更容易地学习和理解STC单片机。对于单片机学习者而言，波特率的计算、定时器的计算、头文件的编写等一直是重点和难点，很多学习者在学习完单片机后不知道如何计算波特率、定时器的参数等等。最新的STC15系列ISP下载编程工具集成了波特率计算器、定时器计算器、软件延时器、头文件等有用的工具，所以使用STC单片机的用户再也不用担心不会计算波特率、定时器等参数了。

从宏晶科技STC单片机设计公司多年的发展来看，使用STC单片机的用户越来越多，可见STC对高校课程教育改革乃至对社会贡献的巨大。

附录L：STC推荐的单片机教材

L.1 两本基于可仿真的STC15F2K60S2系列单片机的本科教材

一、《单片微型计算机原理及接口技术》，作者陈桂友，高等教育出版社出版



《单片微型计算机原理及接口技术》，山东大学陈桂友教授主编，姚永平、王威康主审，由原教育部副部长吴启迪教授和教育部高等学校自动化专业分委员会主任、中国工程院院士清华大学吴澄教授共同作序，高等教育出版社出版，于2012年4月出版，受到热烈欢迎，至今已第三次印刷，得到了国内众多高校的教师普遍认可，计划2014年推出第二版。

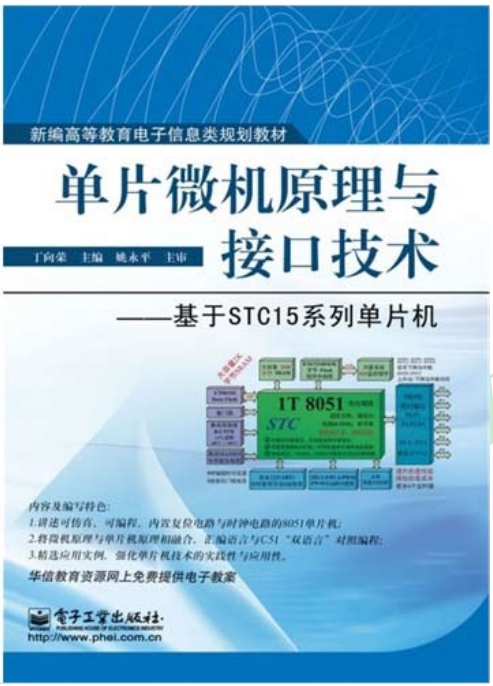
此教材以可仿真的STC15F2K60S2系列单片机为背景，从介绍微型计算机的基本结构和工作原理入手，介绍单片微型计算机（单片机）的构成、各个模块的工作过程、接口原理、应用电路设计、汇编语言和C语言程序设计，精选应用实例，强化单片机技术的实践性与应用性，内容尽可能地选择了目前实际工程中常用的新技术、新器件，力图达到学以致用的根本目的。

全书共分12章，第1章简要介绍微型计算机的发展历史及应用；第2章介绍微型计算机的基础知识，包括数制编码、微型计算机的常见电路、常见技术术语等；第3章介绍计算机系统的组成与工作原理，介绍模型机的构成及工作过程，并介绍单片机的内部结构及典型系统构成；第4章介绍单片机的指令系统及汇编语言程序设计，介绍单片机程序仿真调试和下载的方法；第5章介绍单片机的C语言程序设计与调试，介绍C语言与单片机汇编语言之间的联系；第6章介绍中断的概念和单片机的中断系统；第7章介绍定时器计数器与可编程计数器阵列；第8章介绍数据通信技术，主要介绍常用的并行接口和串行接口工作原理、接口方法以及常用的数据接口芯片及其使用实例；第9章是模数转换器与数模转换器，分别介绍两种转换器的原理和典型芯片的应用；第10章介绍人机交互接口设计，人机交互接口是单片机应用系统必不可少的接口应用；第11章介绍单片机系统的看门狗技术、时钟选择及省电方式的原理和技术；第12章介绍应用系统的设计实例，从硬件和软件两个方面介绍应用系统的设计。每章都有配套的习题，所举例程均经调试通过，很多程序均来自科研和实际应用系统。为了便于学习，开发了与教材配套的综合教学实验平台，该平台提供了20余种实验供学生选用学习，也为善于思考、乐于动手实践的学生提供了自学习实验手段。

本书深入浅出，层次分明，实例丰富，通俗易懂，突出实用，可操作性强，特别适合于作为普通高校计算机类、电子类、电气自动化及机械专业的教学用书。还可作为高职高专以及培训班的教材使用。同时，也可作为从事单片机应用领域的工程技术人员的参考书。

网上订购：当当网、京东商城、亚马逊等

二、《单片微机原理及接口技术》——基于可仿真的STC15系列单片机，作者丁向荣



《单片微机原理与接口技术(ASM+C) -----基于STC15系列单片机》 ，作者丁向荣，本教材于2012年9月出版，于2013年8月第二次印刷，使用本教材的大学有中国矿业大学、深圳大学、河海大学、九江学院、苏州大学、北方理工大学、东北石油大学、北方工业大学等。

此教材选用可在线仿真、在线编程、内置复位电路与时钟电路的STC15系列单片机；将微机原理与单片机技术有机结合，汇编语言与C语言“双语言”对照编程； 精选应用实例，强化单片机技术的实践性与应用性。

在全国各大书店和当当网、京东商城、亚马逊等网店有售

该书内容简介：

STC15系列增强型8051单片机集成了上电复位电路与高精准R/C振荡器，给单片机芯片加上电源就可跑程序；集成了大容量的程序存储器、数据存储器以及EEPRM，集成了A/D、PWM、SPI等高功能接口部件，可大大地简化单片机应用系统的外围电路，促使单片机应用系统的设计更加简捷，系统性能更加高效、可靠。本教材以STC15F2K60S2单片机为主线，系统地介绍了STC15F2K60S2单片机的硬件结构、指令系统与应用编程，系统地介绍了单片机应用系统的开发流程与接口设计，同时，提出多种实践模式：Keil C集成开发环境、Proteus仿真软件以及实物运行开发环境，使得单片机的学习与应用变得更简单、更清晰。

本书可作为普通高校计算机类、电子信息类、电气自动化与机电一体化等专业的教学用书，基础较好的高职高专也可选用本书。此外，可作为电子设计竞赛、电子设计工程师考证的培训教材。也是传统8051单片机应用工程师升级转型的重要参考书籍。

L.2 一本基于可仿真的STC15系列单片机的高职高专教材

《增强型单片机8051单片机原理与系统开发（C51版）》，作者丁向荣，于2013年9月清华大学出版。本教材可作为高职或应用本科计算机类、电子信息类、电气自动化与机电一体化等专业的教学用书。此外，本书可作为电子设计竞赛、电子设计工程师考证的培训教材，也是传统8051单片机应用工程师升级转型的重要参考书。

此教材基于可在线仿真、在线编程、内置复位电路与时钟电路的STC15系列单片机，工学结合，采用任务驱动模式组织教材内容，将教学内容嵌入到一个个单片机应用系统中，学习单片机就是在做单片机应用系统，可实施“教、学、做”一体化教学模式，能有效提高单片机应用实践能力与编程能力。

本教材在当当网、京东商城、亚马逊等网店有售。

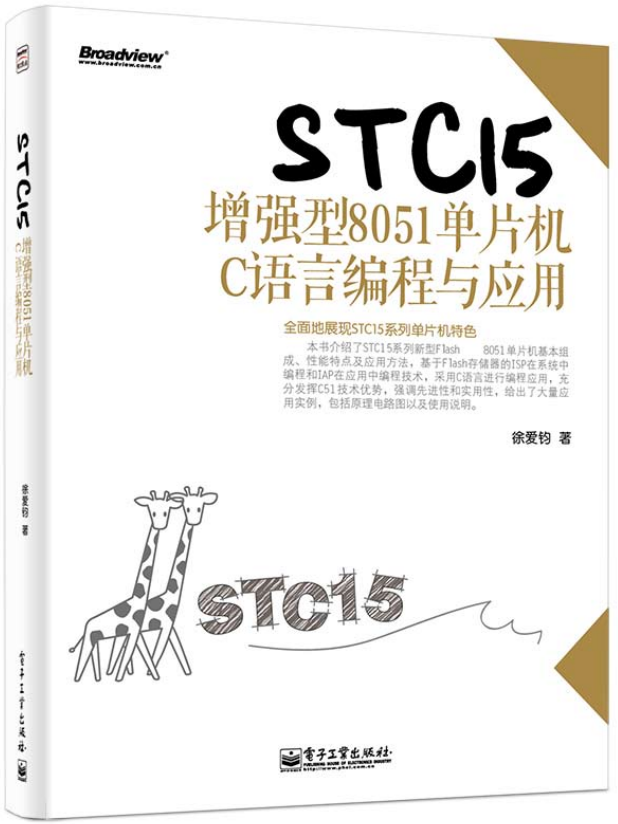
STC15F2K60S2系列增强型8051单片机集成了上电复位电路与高精准RC振荡器，给单片机芯片加上电源就可以运行程序；集成了大容量的程序存储器、数据存储器以及E2PROM，集成了A/D、PWM、SPI等高功能接口部件，可大大地简化单片机应用系统的外围电路，促使单片机应用系统的设计更加简便、快捷，系统性能更加高效、可靠。

STC15F2K60S2单片机的可仿真技术是STC系列单片机的一大创举，它可自定义为仿真芯片或目标应用芯片，仿真时无需增加任何电路，使得单片机仿真变得简单而实用。

本教材按照“教、学、做”一体化教学模式组织教学内容，分基础篇与应用篇，共17个项目，42个任务，兼顾少学时与多学时教学体系。

本书可作为高职或应用本科计算机类、电子信息类、电气自动化与机电一体化等专业的教学用书。此外，本书可作为电子设计竞赛、电子设计工程师考证的培训教材，也是传统8051单片机应用工程师升级转型的重要参考书。

L.3 一本基于STC15系列增强型8051单片机的软件研发参考用书



《STC15增强型单片机8051单片机C语言编程与应用》，基于可仿真的STC15F2K60S2系列单片机，研发人员第一手必备资料，从工程师到资深研发经理的葵花宝典，中国最资深的C51单片机教育专家徐爱钧教授历时20年倾力之作，电子工业出版社于2014年10月出版。

Keil创始人1994年在香港委托徐教授将C51引入中国，STC创始人姚永平看着徐老师的书长大，2004年STC横空出世，2006年STC一统8051江湖，2014年徐老师最新巨著来贺中国创造，历时20年，我们已走完“别人欺负我们中国，不卖单片机/MCU给中国，到可以卖，但只卖落后的MCU，到我们只喜欢国产STC单片机，因为STC抗干扰全球最强”的历程，祖国强大，人人有责。

本书详细介绍了宏晶科技有限公司研发的STC15系列新型Flash 8051单片机基本组成、性能特点及应用方法，基于Flash存储器的ISP在系统中编程和IAP在应用中编程技术，结合目前流行的Keil C51编译器环境，对STC15系列单片机采用C语言进行编程应用，充分发挥C51技术优势，全面地展现STC15系列单片机本身特色，如片内晶体振荡器时钟调整、将芯片配置成为具有仿真功能的单片机、Keil uVision4环境下在线仿真调试、1T单周期指令高速执行、I2C、SPI、ADC等片内资源以及多种片外扩展应用，为读者快速上手提供方便。

本书的特点是强调先进性和实用性，给出了大量应用实例，包括原理电路图以及使用说明，并配套一张CD-ROM光盘，其中包括Keil公司提供的C51全功能评估软件包及全书各章所有程序代码。本书适合于从事单片机应用系统开发研制的广大工程技术人员阅读，也可以作为高等院校相关专业大学生或研究生的教学参考书。本书在当当网、京东商城、亚马逊、电子工业出版社官网等网店有售。

L.4 一本基于STC15系列增强型8051单片机的实用范例参考用书

《MCS-51单片机C语言编程100例》，作者孙安青，中国电力出版社出版。本书以STC15F2K60S2增强型的单周期单片机为核心器件，通过100个应用实例，以单片机的功能模块为起点，以每个模块的硬件、软件设计为主线，通过大量实例的介绍来达到理论联系实际的目的，让读者能迅速开发出实际的应用系统。

本书讲解深入浅出，实例内容翔实，绝大多数模、实例都有Proteus仿真验证。

本书适合各类高校自动控制、电气工程、自动化、机械电子专业的学生使用，同时也可供从事单片机系统开发的广大技术人员参考阅读。

本书在全国各大书店和当当网、京东商城、亚马逊等网店有售。

附录M：每日更新内容的备忘录

2015-8-10更新内容：

1. 更新彩色宣传页”；
2. 删除″利用USB转串口芯片CH340G的ISP下载编程典型应用线路图″；
3. 新增利用″USB转串口芯片PL-2303HXD/PL-2303HX的ISP下载编程典型应用线路图″；
4. 新增″利用USB转串口芯片PL-2303SA的ISP下载编程典型应用线路图″；
5. 修正“传统8051单片机指令定义详解”中的错误；

2015-4-13更新内容：

1. 更新了附录“STC彩色宣传资料”；
2. 更新了10章“编译器（汇编器）/ISP编程器(烧录)/仿真器说明”；
3. 更新了附录L“STC推荐的单片机教材”；
4. 更新了4.3节“I/O口工作模式”；
5. 修正“利用USB转串口的典型应用线路图”中的错误；

2014-2-12更新内容：

1. 更新了STC89系列单片机价格；
2. 更新了10章“编译器（汇编器）/ISP编程器(烧录)/仿真器说明”；
3. 更新了附录“STC彩色宣传资料”；
4. 新增了附录“STC对单片机相关学科群部分课程改革呼吁”；
5. 新增了附录“STC对单片机课程教育改革的贡献”；
6. 新增了附录“STC推荐的单片机教材”；

2013-10-19更新内容：

1. 更新了附录“STC彩色宣传资料”
2. 新增技术支持QQ：800003751；
3. 新增官方网站：www.GXWMCU.com
4. 修正了5.3.1节及5.3.2节“传统8051单片机指令定义详解”中的错误；
5. 更新了10章“编译器(汇编器)/编程器(烧录)/仿真器”的介绍；
6. 增加“利用USB转串口的在系统可编程(ISP)典型应用线路图”

2012-5-13更新内容：

1. 增加了附录“STC彩色宣传资料”
2. 调整的第二章“时钟、复位及省电模式”的章节顺序；
3. 修改了“多机通信”中的错误；

4.将文中“请求中断标志位”的错误更正为“中断请求标志位”；
5.将研发顾问的号码更改为了13922805190；
6.第三章的标题改为“存储器和特殊功能寄存器(SFRs)”。

2011-9-8更新内容：

增加了5.3.1节“（中文的）传统8051单片机指令定义详解”，并附有5.3.2节的英文文本参考。

2011-2-21更新内容：

1.更新了各系列的选型指南

2011-1-21更新内容：

1.调整了第6章的中断系统结构图
2.调整了4.2节的内容

2011-1-20更新内容：

1.增加了在7.2.3节“定时器2作波特率发生器的测试程序”
2.增加了定时器/计数器2作定时器的测试程序
3.取消了“定时器/计数器2的设置”其作为一独立小节，并将其调整在了“自动重装模式”小节之后
4.修改了第6章的中断系统结构图
5.更正了90C版本的ALE/P4.5管脚的设置
6.增加了“STC89C51RC/RD+系列ALE/P4.5的设置”这一节
7.调整了“STC89C51RC/RD+系列单片机内部EEPROM选型一览表”
8.增加了“STC89C51RC系列单片机内部EEPROM详细地址表”
9.添加了P4口的程序
10.删除了总线扩展图

2011-1-19更新内容：

1.将第6章中断系统的第1段中的“异步事件处理”修改为了“紧急事件的实时处理”
2.修改了第6章的中断系统结构图
3.修改了中断系统程序注解
4.增加了"如何利用Keil C软件减少代码长度"和"如何实现运行中自定义下载"两个附录
5.将原10.3节的"自定义下载演示程序"移动到了附录H中
6.更正了90C版本的P4.5/ALE管脚

附录N：STC彩色宣传资料

STC micro™ 宏晶科技

超强抗干扰，无法解密

8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

官方网站：www.STCMCU.com www.GXWMCU.com
南通 Tel: 0513-5501 2928 5501 2929
深圳 Tel: 0755-8294 8411 8294 8412

STC15系列1T 8051单片机，超高速双串口，高速A/D，比较器，宽电压，低功耗

不需外部晶振的单片机
不需外部复位的单片机

送仿真器

全球第一款真正意义上的单片机
ISP/IAP技术全球领导者

采用宏晶第九代加密技术，现悬赏20万元人民币请专家帮忙查找加密有无漏洞

大容量2K字节SRAM

大容量 2048 字节 SRAM

8/16/32/60/61K 字节 Flash 程序存储器

IAP

内置系统 ISP 监控程序

INT2/INT3/INT4 仅可下降沿中断
INT0/INT1 上升沿/下降沿中断均可

E²PROM Data Flash

IAP

看门狗

集成高精度 R/C时钟 ±1%温飘 -40℃～+85℃

集成MAX810 专用复位电路

1T 8051 微处理器

STC

超低功耗，超低价
高速(0-35M)，高可靠
超强抗干扰，无法解密

- 宽电压(2.5V~5.5V)
- 内置高精准时钟(ISP编程时5~35MHz可任意设置)
- 内置高可靠复位电路(16级复位门槛电压可设置)
- 掉电模式：<0.1uA，可由外中断或专用定时器唤醒

5组8位 并行端口 P0/P1 P2/P3/P4 + P5.4 / P5.5 最多42个I/O

P0 P1 P2 P3 P4 P5

ISP编程时可设置16级复位门槛电压 最多6个定时器

超高速双串口(UART) 分时复用可当5组使用

3路CCP/PCA/PWM PWM可当D/A使用 高频PWM：6位/7位/8位

A/D (高速10位8路) 1组比较器

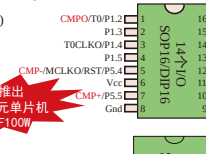
提升的是性能 降低的是成本 比较器可作掉电检测

封装形式：

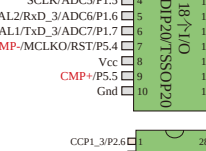
LQFP44(12mm x 12mm, 强烈推荐)
PDIP40(正常生产)
LQFP32(9mm x 9mm, 推荐)
QFN32(5mm x 5mm)
SOP28(强烈推荐)
TSSOP28(6.4mm x 9.7mm)
QFN28(5mm x 5mm)
SKDIP28(正常生产)
SOP20(强烈推荐)
DIP20(正常生产)
TSSOP20 (6.5mm x 6.5mm)
SOP16(6mm x 9.9mm, 强烈推荐)
DIP16(正常生产)
SOP8(强烈推荐)
DIP8(正常生产)
DFN8(4mm x 4mm)

每片单片机具有全球唯一身份证号码(ID号)，无法解密，加密坚不可摧

STC15W201S系列，RMB1.15元起

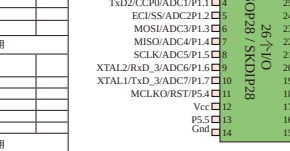


STC15W201S系列是目前全球抗干扰最强的Flash型单片机
现悬赏20万元人民币寻找比STC15W201S系列抗干扰更强的Flash型单片机



隆重推出 RMB 0.89元单片机 STC15F100W

全部175℃ 8小时高温烘烤



所有封装
LQFP44/PDIP40
LQFP32/QFN32
SOP28/SKDIP28/TSSOP28/QFN28
SOP20/DIP20/TSSOP20
SOP16/DIP16
SOP8/DIP8/DFN8
(现此系列未生产PLCC44和SOP32封装)

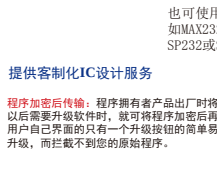
封装价格 (RMB ¥)

型号	工作电压(V)	Flash程序存储器(byte)	大容量SRAM字节	串行口可掉电唤醒	SPI	普通定时器计数器	CCP PCA PWM可当外部中断掉电唤醒	掉电唤醒专用定时器	A/D 8路(3路PWM可当外部中断掉电唤醒，3路D/A使用)	比较器(可当1路ADC使用，可作外部掉电检测)	EEP ROM	内部低压检测中断	内部看门狗	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	程序加密后传输(防拦截)	可设下次更新程序口令	支持RS485需下载	SOP8	DIP8	DFN8							
大批量现货供应中	STC15F100W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	-	-	2(T0/T2)	-	有	5	-	-	-	有	有	8级	有	是	有	是	¥0.89									
STC15F101W	5.5-3.8	1K	128	-	-	2(T0/T2)	-	有	5	-	4K	有	有	8级	有	是	有	是	是	¥0.99	¥1.1								
STC15F102W	5.5-3.8	2K	128	-	-	2(T0/T2)	-	有	5	-	3K	有	有	8级	有	是	有	是	是	¥1.1	¥1.2								
STC15F104W	5.5-3.8	4K	128	-	-	2(T0/T2)	-	有	5	-	1K	有	有	8级	有	是	有	是	是	¥1.2	¥1.3	¥1.4							
IRC15F107W 默认使用内部24MHz时钟	5.5-3.8	7K	128	-	-	2(T0/T2)	-	有	5	-	IAP	有	有	固定	有	是	无	否	否	¥1.2	¥1.3								
用户可将用户程序区不用的程序Flash当EEPROM使用																													
大批量现货供应中	STC15W10x系列单片机选型价格一览表 串行口功能可由[P3.0/INT4, P3.1]结合定时器实现																			SOP8	DIP8	DFN8							
STC15W100	2.5-5.5	0.5K	128	-	-	2(T0/T2)	-	有	5	-	-	有	有	16级	有	是	有	是	是	¥0.99									
STC15W101	2.5-5.5	1K	128	-	-	2(T0/T2)	-	有	5	-	4K	有	有	16级	有	是	有	是	是	¥1.1	¥1.2								
STC15W102	2.5-5.5	2K	128	-	-	2(T0/T2)	-	有	5	-	3K	有	有	16级	有	是	有	是	是	¥1.2	¥1.3								
STC15W104	2.5-5.5	4K	128	-	-	2(T0/T2)	-	有	5	-	1K	有	有	16级	有	是	有	是	是	¥1.3	¥1.4	¥1.5							
IRC15W107 默认使用内部24MHz时钟	2.5-5.5	7K	128	-	-	2(T0/T2)	-	有	5	-	IAP	有	有	固定	有	是	无	否	否	¥1.3	¥1.4								
用户可将用户程序区不用的程序Flash当EEPROM使用																													
	STC15W201S系列单片机选型价格一览表，大批量现货供应中																			SOP8	SOP16	DIP16							
STC15W201S	2.5-5.5	1K	256	1	-	2(T0/T2)	-	有	5	-	4K	有	有	16级	有	是	有	是	是	¥1.15	¥1.2	¥1.35							
STC15W202S	2.5-5.5	2K	256	1	-	2(T0/T2)	-	有	5	-	3K	有	有	16级	有	是	有	是	是	¥1.25	¥1.3	¥1.45							
STC15W204S	2.5-5.5	4K	256	1	-	2(T0/T2)	-	有	5	-	1K	有	有	16级	有	是	有	是	是	¥1.35	¥1.4	¥1.55							
IRC15W207S 默认使用内部24MHz时钟	2.5-5.5	7.5K	256	1	-	2(T0/T2)	-	有	5	-	有	IAP	有	有	固定	有	是	无	否	¥1.35	¥1.4	¥1.55							
用户可将用户程序区不用的程序Flash当EEPROM使用																													
	STC15W401AS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																			SOP16	DIP16	SOP20	DIP20	TSSOP20	SOP28	SKDIP28	TSSOP28	QFN28
STC15W401AS	2.5-5.5	1K	512	1	有	2(T0/T2)	3-ch	有	5	10-bit	有	5K	有	有	16级	有	是	有	是	¥1.6	¥1.8	¥1.8	¥2.0	¥1.9	¥2.0	¥2.2	¥2.1	¥2.0
STC15W402AS	2.5-5.5	2K	512	1	有	2(T0/T2)	3-ch	有	5	10-bit	有	5K	有	有	16级	有	是	有	是	¥1.8	¥2.0	¥2.0	¥2.2	¥2.1	¥2.2	¥2.4	¥2.3	¥2.2
STC15W404AS	2.5-5.5	4K	512	1	有	2(T0/T2)	3-ch	有	5	10-bit	有	9K	有	有	16级	有	是	有	是	¥1.99	¥2.2	¥2.2	¥2.4	¥2.3	¥2.5	¥2.7	¥2.6	¥2.5
STC15W408AS	2.5-5.5	8K	512	1	有	2(T0/T2)	3-ch	有	5	10-bit	有	5K	有	有	16级	有	是	有	是	¥2.2	¥2.4	¥2.4	¥2.6	¥2.5	¥2.7	¥2.9	¥2.8	¥2.7
IAP15W413AS	2.5-5.5	13K	512	1	有	2(T0/T2)	3-ch	有	5	10-bit	有	IAP	有	有	16级	有	是	有	是	¥2.3	¥2.5	¥2.5	¥2.7	¥2.6	¥2.8	¥3.0	¥2.9	¥2.8
用户可将用户程序区不用的程序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(T0/T2)	3-ch	有	5	10-bit	有	IAP	有	有	固定	有	是	无	否	¥2.3	¥2.5	¥2.5	¥2.7	¥2.6	¥2.8	¥3.0	¥2.9	¥2.8
用户可将用户程序区不用的程序Flash当EEPROM使用																													
	STC15W404S系列单片机选型价格一览表，大批量现货供应中																			SOP28	SKDIP28	LQFP32	QFN32	PDIP40	LQFP44				
STC15W404S	2.5-5.5	4K	512	1	有	3	-	有	5	-	有	9K	有	有	16级	有	是	有	是	¥2.3	¥2.5	¥2.4		¥3.0	¥2.5				
STC15W408S	2.5-5.5	8K	512	1	有	3	-	有	5	-	有	5K	有	有	16级	有	是	有	是	¥2.5	¥2.7	¥2.6		¥3.0	¥2.7				
IAP15W413S	2.5-5.5	13K	512	1	有	3	-	有	5	-	有	IAP	有	有	16级	有	是	有	是	¥2.6	¥2.8	¥2.7	¥2.65	¥3.3	¥2.8				
用户可将用户程序区不用的程序Flash当EEPROM使用																													
IRC15W415S 默认使用内部24MHz时钟	2.5-5.5	15.5K	512	1	有	3	-	有	5	-	有	IAP	有	有	固定	有	是	无	否	¥2.6	¥2.8	¥2.7		¥3.3	¥2.8				
用户可将用户程序区不用的程序Flash当EEPROM使用																													
	STC15W1K16S系列单片机选型价格一览表，大批量现货供应中																			TSSOP20	SOP28	SKDIP28	LQFP32	QFN32	PDIP40	LQFP44			
STC15W1K16S	2.6-5.5	16K	1K	1	有	3	-	有	5	-	有	13K	有	有	16级	有	是	有	是	¥3.3	¥3.5	¥3.4			¥4.0	¥3.5			
STC15W1K24S	2.6-5.5	24K	1K	1	有	3	-	有	5	-	有	5K	有	有	16级	有	是	有	是	¥3.6	¥3.6	¥3.8	¥3.7		¥4.3	¥3.8			
IAP15W1K29S	2.6-5.5	29K	1K	1	有	3	-	有	5	-	有	IAP	有	有	16级	有	是	有	是	¥3.6	¥3.6	¥3.8	¥3.7	¥3.65	¥4.3	¥3.8			
用户可将用户程序区不用的程序Flash当EEPROM使用																													
IRC15W1K31S 默认使用内部24MHz时钟	2.6-5.5	31.5K	1K	1	有	3	-	有	5	-	有	IAP	有	有	固定	有	是	无	否	¥3.6	¥3.8	¥3.7			¥4.3	¥3.8			
用户可将用户程序区不用的程序Flash当EEPROM使用																													
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																			TSSOP20	SOP28	SKDIP28	LQFP32	QFN32	PDIP40	LQFP44			
STC15F2K08S2	5.5-4.5	8K	2K	2	有	3	3-ch	有	5	10位	-	53K	有	有	8级	有	是	有	是	¥3.8	¥4.0	¥3.9			¥4.5	¥4.0			
STC15F2K16S2	5.5-4.5	16K	2K	2	有	3	3-ch	有	5	10位	-	45K	有	有	8级	有	是	有	是	¥4.0	¥4.2	¥4.1			¥4.7	¥4.2			
STC15F2K32S2	5.5-4.5	32K	2K	2	有	3	3-ch	有	5	10位	-	29K	有	有	8级	有	是	有	是	¥4.3	¥4.5	¥4.4			¥4.9	¥4.5			
STC15F2K48S2	5.5-4.5	48K	2K	2	有	3	3-ch	有	5	10位	-	13K	有	有	8级	有	是	有	是	¥4.4	¥4.6	¥4.5			¥4.9	¥4.6			
STC15F2K60S2	5.5-4.5	60K	2K	2	有	3	3-ch	有	5	10位	-	1K	有	有	8级	有	是	有	是	¥4.4	¥4.6	¥4.5			¥4.9	¥4.6			
IAP15F2K61S2 本身就是仿真器	5.5-4.5	61K	2K	2	有	3	3-ch	有	5	10位	-	IAP	有	有	8级	有	是	有	是	¥4.4	¥4.4	¥4.6	¥4.5	¥4.45	¥4.9	¥4.6			
用户可将用户程序区不用的程序Flash当EEPROM使用																								¥4.9	¥4.6				
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	-	IAP	有	有	固定	有	是	无	否	否									
用户可将用户程序区不用的程序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						¥4.4	¥3.9			
STC15F2K60S	5.5-4.5	60K	2K	1	有	3	-	有	5	-	-	1K	有	有	8级	有	是	有	是						¥4.4	¥3.9			
IAP15F2K61S	5.5-4.5	61K	2K	1	有	3	-	有	5	-	-	IAP	有	有	8级	有	是	有	是						¥4.5	¥4.0			
用户可将用户程序区不用的程序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							¥3.9			
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							¥3.9			

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

特别声明：以15F和15L开头具有SPI功能的芯片只支持"SPI主机模式"；以15W开头且有SPI功能的芯片，SPI主/从机模式均支持
以15F和15L开头的芯片不支持"串行口掉电唤醒"功能，而以15W开头的芯片支持该功能

STC 单片机在线编程ISP/仿真线路 *USB*转串口



另有PL-2303HXD-SSOP28及PL-2303HX-SSOP28，其价格更低

也可使用普通的RS232芯片接电脑
如MAX232或MAX3232
SP232或SP3232

提供客制化IC设计服务

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。

STC micro™ 宏晶科技

超强抗干扰，无法解密

8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

官方网站：www.STCMCU.com www.GXWMCU.com
南通 Tel: 0513-5501 2928 5501 2929
深圳 Tel: 0755-8294 8411 8294 8412

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

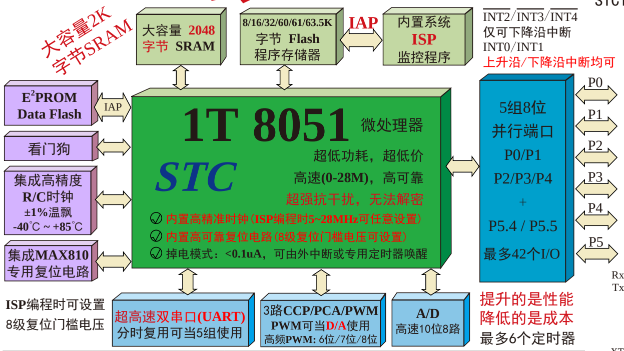
不需外部晶振的单片机
不需外部复位的单片机

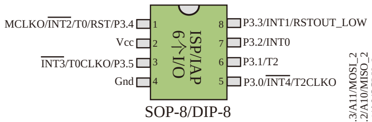
送仿真器

全球第一款真正意义上的单片机
ISP/IAP技术全球领导者

采用宏晶第八代加密技术，现悬赏10万元人民币请专家帮忙查找加密有无漏洞

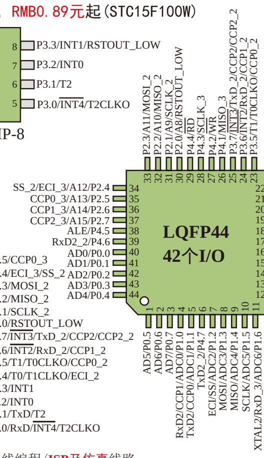
STC15F100W系列，大批量供货中，RMB0.89元起(STC15F100W)

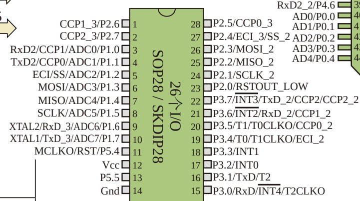




每片单片机具有全球唯一
身份证号码(ID号)，
无法解密，加密坚不可摧
提供客制化IC设计服务

封装形式：
LQFP44(12mm x 12mm, 强烈推荐)
PDIP40(正常生产)
LQFP32(9mm x 9mm, 推荐)
QFN32(5mm x 5mm)
SOP28(强烈推荐)
SKDIP28(正常生产)
TSSOP20(6.5mm x 6.5mm)





全部175℃
8小时高温烘烤

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	TSSOP20	SOP28	SKDIP28	LQFP32	QFN32	PDIP40	LQFP44
																					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 ¥)						
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		¥3.8	¥4.0	¥3.9		¥4.5	¥4.0
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		¥4.0	¥4.2	¥4.1		¥4.7	¥4.2
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		¥4.3	¥4.5	¥4.4		¥4.9	¥4.5
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		¥4.4	¥4.6	¥4.5		¥4.9	¥4.6
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		¥4.4	¥4.6	¥4.5		¥4.9	¥4.6
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		¥4.4	¥4.6	¥4.5		¥4.9	¥4.6
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		¥4.4	¥4.6	¥4.5		¥4.9	¥4.6
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	¥4.4	¥4.4	¥4.6	¥4.5	¥4.45	¥4.9	¥4.6
																					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						¥4.9	¥4.6
																					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						¥4.4	¥3.9
STC15F2K60S	5.5-4.5	60K	2K	1	有	3	-	有	5	-	-	1K	有	有	8级	有	是	有	是	是						¥4.4	¥3.9
IAP15F2K61S	5.5-4.5	61K	2K	1	有	3	-	有	5	–	2	IAP	有	有	8级	有	是	有	是	是						¥4.5	¥4.0
																					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							¥3.9
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							¥3.9
大批量 现货供应中	STC15F100W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	–	–	2(T0/T2)	–	有	5	–	1	–	有	有	8级	有	是	有	是	是	¥0.89						
STC15F101W	5.5-3.8	1K	128	–	–	2(T0/T2)	–	有	5	–	1	4K	有	有	8级	有	是	有	是	是	¥0.99	¥1.1					
STC15F102W	5.5-3.8	2K	128	–	–	2(T0/T2)	–	有	5	–	1	3K	有	有	8级	有	是	有	是	是	¥1.1	¥1.2					
STC15F104W	5.5-3.8	4K	128	–	–	2(T0/T2)	–	有	5	–	1	1K	有	有	8级	有	是	有	是	是	¥1.2	¥1.3	¥1.4				
IRC15F107W 默认使用内部24MHz时钟	5.5-3.8	7K	128	–	–	2(T0/T2)	–	有	5	–	1	IAP	有	有	固定	有	是	无	否	否	¥1.2	¥1.3					
																					用户可将用户程序区的程序Flash当EEPROM使用						

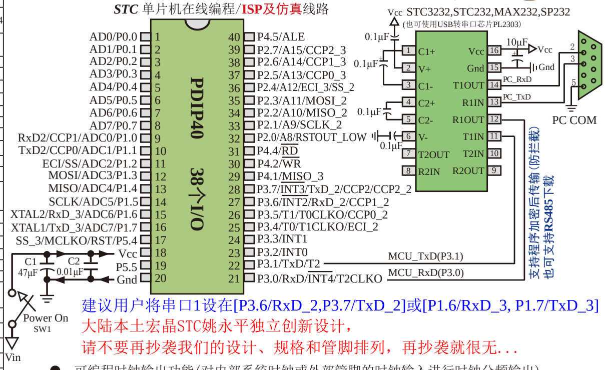
我们直销，所以低价，**以上单价为10K/M起定量**，量小每片需加0.1元，以上价格运费由客户承担，零售10片起，如对价格不满，可来电要求降价

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。

特别声明：以15F和15L开头且有SPI功能的芯片不支持"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。

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。

STC 单片机在线编程/ISP及仿真线路



建议用户将串口1设在[P3.6/RxD_2, P3.7/TxD_2]或[P1.6/RxD_3, P1.7/TxD_3]
大陆本土宏晶STC姚永平独立创新设计，
请不要再抄袭我们的设计、规格和管脚排列，再抄袭就很无...

宏晶·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%温飘(-40℃~+85℃)，常温下温飘±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/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、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)：
  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，以上3个定时器/计数器输出时钟可1~65536级分频输出；④ 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/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
- 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

选择宏晶·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，宏晶第八代加密技术，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：
1. 高抗静电(ESD保护)整机轻松过2万伏静电测试　2. 轻松过4kV快速脉冲干扰(EFT测试)
3. 宽电压，不怕电源抖动　4. 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：
1. **掉电模式：外部中断唤醒功耗** <0.1uA
2. 空闲模式：典型功耗<1mA,　3. 正常工作模式：4mA～6mA
4. **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

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

STC micro™ 宏晶科技

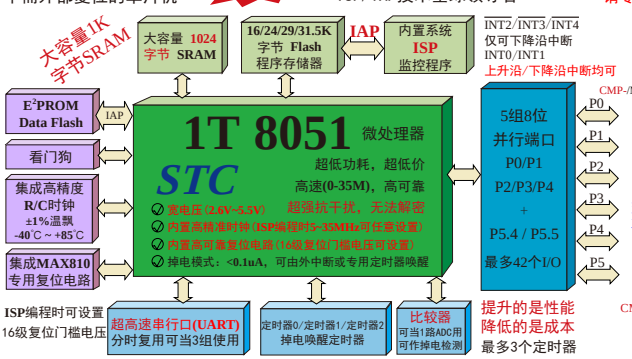
超强抗干扰，无法解密

8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

官方网站：www.STCMCU.com / www.GXWMCU.com
南通 Tel: 0513-5501 2928 5501 2929
深圳 Tel: 0755-8294 8411 8294 8412

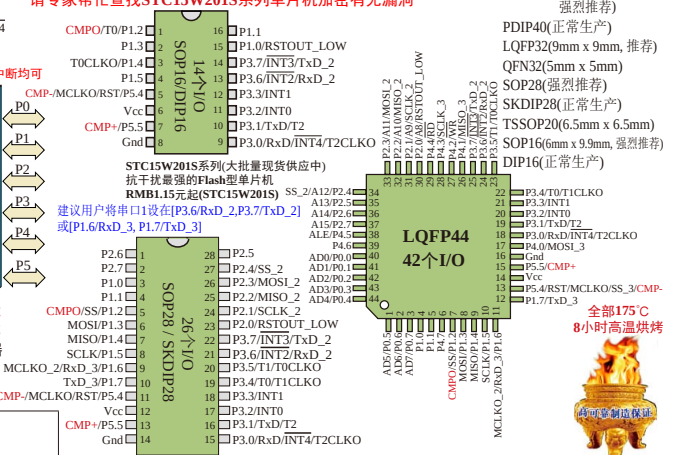
STC15W1K16S 系列 1T 8051 单片机，宽电压，超高速串行口，比较器

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



提升的是性能 降低的是成本 最多3个定时器

采用宏晶第九代加密技术，现悬赏20万元人民币
请专家帮忙查找STC15W201S系列单片机加密有无漏洞



全部175℃ 8小时高温烘烤

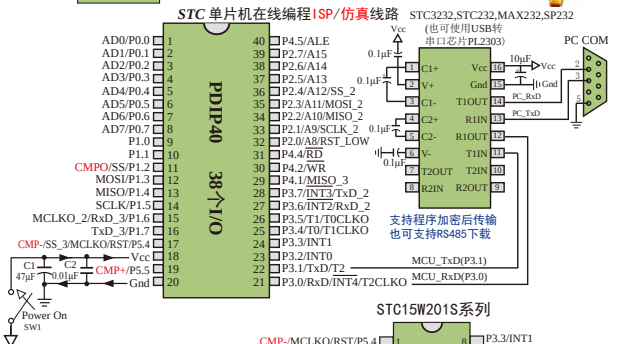
STC15W201S系列是目前全球抗干扰最强的Flash型单片机
现悬赏20万元人民币寻找比STC15W201S系列抗干扰更强的Flash型单片机

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	CCP/PCA/PWM可当外部中断用	可当掉电唤醒	标准外部中断	支持掉电唤醒	可当定时器	A/D(3路PWM可当3路D/A使用)	比较器(可当1路ADC使用，可作外部掉电检测)	内部低压检测中断	内置看门狗	内部高可靠复位电路	内部高精准时钟	可对外输出时钟	可编程加密和传统防拷贝	可掉电唤醒中断复位	可下载用户程序	支持RS485下载	支持5V单片机系统	所有封装 LQFP44/PDIP40 LQFP32/QFN32 SOP28/SKDIP28 TSSOP20 SOP16/DIP16 封装价格(RMB ¥)		
STC15W201S系列单片机选型价格一览表，大批量现货供应中																									SOP8	SOP16	DIP16
STC15W201S	2.5-5.5	1K	256	1	–	T0 / T2	–	有	5	–	有	4K	有	有	16级	有	是	有	是	是	是	是	是	是	¥1.15	¥1.2	¥1.35
STC15W202S	2.5-5.5	2K	256	1	–	T0 / T2	–	有	5	–	有	3K	有	有	16级	有	是	有	是	是	是	是	是	是	¥1.25	¥1.3	¥1.45
STC15W203S	2.5-5.5	3K	256	1	–	T0 / T2	–	有	5	–	有	2K	有	有	16级	有	是	有	是	是	是	是	是	是	¥1.35	¥1.4	¥1.55
STC15W204S	2.5-5.5	4K	256	1	–	T0 / T2	–	有	5	–	有	1K	有	有	16级	有	是	有	是	是	是	是	是	是	¥1.35	¥1.4	¥1.55
IAP15W205S	2.5-5.5	5K	256	1	–	T0 / T2	–	有	5	–	有	IAP	有	有	16级	有	是	有	是	是	是	是	是	是	¥1.35	¥1.4	¥1.55
																									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	2.5-5.5	7.5K	256	1	–	T0 / T2	–	有	5	–	有	IAP	有	有	固定	有	是	无	否	否					¥1.35	¥1.4	¥1.55
																									用户可将用户程序区的程序Flash当EEPROM使用		
STC15W404S系列单片机选型价格一览表，大批量现货供应中																									SOP28	SKDIP28	LQFP32	QFN32	PDIP40	LQFP44
STC15W4045	2.5-5.5	4K	512	1	有	3	–	有	5	–	有	9K	有	有	16级	有	是	有	是	是					¥2.3	¥2.5	¥2.4		¥3.0	¥2.5
STC15W408S	2.5-5.5	8K	512	1	有	3	–	有	5	–	有	5K	有	有	16级	有	是	有	是	是					¥2.5	¥2.7	¥2.6		¥3.0	¥2.7
STC15W410S	2.5-5.5	10K	512	1	有	3	–	有	5	–	有	3K	有	有	16级	有	是	有	是	是										¥2.8
IAP15W413S	2.5-5.5	13K	512	1	有	3	–	有	5	–	有	IAP	有	有	16级	有	是	有	是	是					¥2.6	¥2.8	¥2.7	¥2.65	¥3.3	¥2.8
																									用户可将用户程序区的程序Flash当EEPROM使用					
IRC15W415S 默认使用内部24MHz时钟	2.5-5.5	15.5K	512	1	有	3	–	有	5	–	有	IAP	有	有	固定	有	是	无	否	否					¥2.6	¥2.8	¥2.7		¥3.3	¥2.8
																									用户可将用户程序区的程序Flash当EEPROM使用					
STC15W1K16S系列单片机选型价格一览表，大批量现货供应中																									TSSOP20	SOP28	SKDIP28	LQFP32	QFN32	PDIP40	LQFP44
STC15W1K16S	2.6-5.5	16K	1K	1	有	3	–	有	5	–	有	13K	有	有	16级	有	是	有	是						¥3.3	¥3.5	¥3.4		¥4.0	¥3.5
STC15W1K24S	2.6-5.5	24K	1K	1	有	3	–	有	5	–	有	5K	有	有	16级	有	是	有	是		¥3.6	¥3.6	¥3.8	¥3.7		¥4.3	¥3.8
IAP15W1K29S	2.6-5.5	29K	1K	1	有	3	–	有	5	–	有	IAP	有	有	16级	有	是	有	是		¥3.6	¥3.6	¥3.8	¥3.7	¥3.65	¥4.3	¥3.8
																									用户可将用户程序区的程序Flash当EEPROM使用					
IRC15W1K31S 默认使用内部24MHz时钟	2.6-5.5	31.5K	1K	1	有	3	–	有	5	–	有	IAP	有	有	固定	有	是	无	否						¥3.6	¥3.8	¥3.7		¥4.3	¥3.8
																									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，就可将程序用加密后再用"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IRC开头的单片机默认使用内部24MHz时钟，其内部复位门槛电压固定，P5.4不可当复位管脚RST使用，P3.2/P3.3与下载无关，且不支持"程序加密后传输"功能。



每片单片机具有全球唯一身份证号码(ID号)，无法解密，加密坚不可摧

提供客制化IC设计服务

- 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(42/38/30/26/18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
- 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

宏晶•STC15W1K16S系列主要性能：

大陆本土宏晶STC姚永平独立创新设计，请不要再抄袭我们的设计、规格和管脚排列，再抄袭就很无...

- 大容量1024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：2.6V ～ 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%温飘(-40℃～+85℃)，常温下温飘±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)；
  RxD/T0/T1/T2管脚：内部掉电唤醒专用定时器
- 16/24/29K/31.5K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
- 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）

选择宏晶•STC15W1K16S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1024字节SRAM
- ★ 无法解密，宏晶第九代加密技术，
  现悬赏20万元人民币请专家帮忙查找STC15W201S系列和STC15W1K16S系列加密有无漏洞
- ★ 超强抗干扰：
  1. 高抗静电(ESD保护)整机轻松过2万伏静电测试
  2. 轻松过4kV快速脉冲干扰(EFT测试)
  3. 宽电压，不怕电源抖动
  4. 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：
  1. **掉电模式：外部中断唤醒功耗** <0.1uA
  2. 空闲模式：典型功耗<1mA，
  3. 正常工作模式：4mA～6mA
  4. **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

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

STC micro™ 宏晶科技

超强抗干扰，无法解密

8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

官方网站：www.STCMCU.com www.GXWMCU.com
南通 Tel: 0513-5501 2928 5501 2929
深圳 Tel: 0755-8294 8411 8294 8412

宏晶•STC15W401AS 系列 1T 8051 单片机，超高速串口，高速A/D，比较器，宽电压

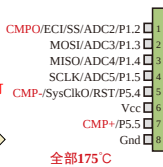
不需外部晶振的单片机
不需外部复位的单片机

送仿真器

全球第一款真正意义上的单片机
ISP/IAP技术全球领导者

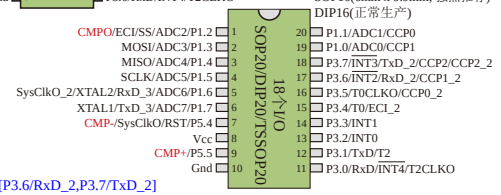
提供客制化
IC设计服务

采用宏晶第九代加密技术，现悬赏20万元人民币请专家帮忙查找加密有无漏洞



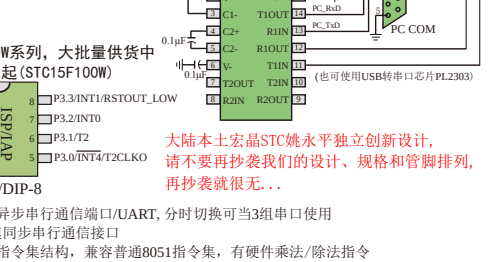
封装形式：
SOP28(强烈推荐)
TSSOP28(6.4mm x 9.7mm)
SKDIP28(正常生产)
QFN28(5mm x 5mm)
SOP20(强烈推荐)
DIP20(正常生产)
TSSOP20 (6.5mm x 6.5mm)
SOP16(6mm x 9.9mm, 强烈推荐)
DIP16(正常生产)

全部175℃ 8小时高温烘烤



建议用户将串口1设在[P3.6/RxD_2,P3.7/TxD_2]
或[P1.6/RxD_3, P1.7/TxD_3]

STC 单片机在线编程/**ISP及仿真**线路(***USB***转串口)



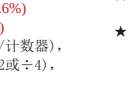
型号	工作电压(V)	Flash程序存储器(byte)	SRAM字节	串行口1	SPI	掉电唤醒	普通定时器(计数器)T0/T2外部管脚也能掉电唤醒	CCP/PCA/PWM	掉电检测	标准外部中断4路,支持掉电唤醒	A/D 8路(3路PWM可当3路D/A使用)	比较器(可当1路ADC使用,可作掉电检测)	DPTR	EEPROM	内部低压检测中断,可掉电唤醒	看门狗	内部高可靠复位(可选16级复位门槛电压)	可对外输出时钟	内置4K字节程序加密后传输(防止反编译)	可设下次程序加密后传输口令	支持RS485下载	所有封装 SOP28/ SKDIP28 / TSSOP28 / QFN28 SOP20 / DIP20 / TSSOP20 SOP16 / DIP16	部分封装价格(RMB ¥)													
																						SOP16	DIP16	SOP20	DIP20	TSSOP20	SOP28	SKDIP28	TSSOP28	QFN28
STC15W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	¥1.60	¥1.80	¥1.80	¥2.00	¥1.90	¥2.00	¥2.20	¥2.10	¥2.00
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	¥1.80	¥2.00	¥2.00	¥2.20	¥2.10	¥2.40	¥2.30	¥2.20	¥2.20
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	¥1.99	¥2.20	¥2.20	¥2.40	¥2.30	¥2.50	¥2.70	¥2.60	¥2.50
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	¥2.20	¥2.40	¥2.40	¥2.60	¥2.50	¥2.70	¥2.90	¥2.80	¥2.70
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	¥2.30	¥2.50	¥2.50	¥2.70	¥2.60	¥2.80	¥3.00	¥2.90	¥2.80
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	无	否	否	¥2.30	¥2.50	¥2.50	¥2.70	¥2.60	¥2.80	¥3.00	¥2.90	¥2.80
																						用户可将用户程序区的程序Flash当EEPROM使用								

我们真销，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。

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 。

STC15F100W系列，大批量供货中
RMB0. 89元起(STC15F100W)



SOP-8/DIP-8

每片单片机具有
全球唯一身份证号码(ID号)
无法解密，加密坚不可摧

大陆本土宏晶STC姚永平独立创新设计，
请不要再抄袭我们的设计、规格和管脚排列，
再抄袭就很无. . .

宏晶•STC15W401AS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%温飘(-40℃~+85℃)，常温下温飘±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

选择宏晶•STC15W401AS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，宏晶第九代加密技术，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：
1. 高抗静电(ESD保护)整机轻松过2万伏静电测试
2. 轻松过4kV快速脉冲干扰(EFT测试)
3. 宽电压，不怕电源抖动
4. 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：
1. **掉电模式：外部中断唤醒功耗** <0.1uA
2. 空闲模式：典型功耗<1mA，　　3. 正常工作模式：4mA～6mA
4. **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

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

STC micro™ 宏晶科技

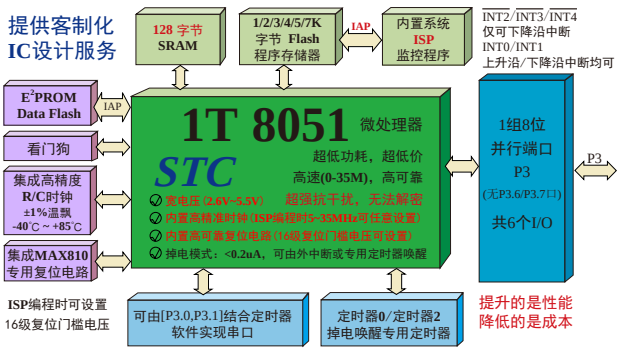
超强抗干扰，无法解密

8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

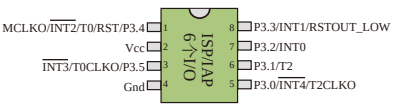
官方网站：www.STCMCU.com www.GXWMCU.com
南通 Tel: 0513-5501 2928 5501 2929
深圳 Tel: 0755-8294 8411 8294 8412

宏晶•STC15W10x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
送仿真器
全球第一款真正意义上的单片机ISP/IAP技术全球领导者
采用宏晶第九代加密技术，现悬赏20万元人民币请专家帮忙查找加密有无漏洞



STC15W10x系列，已开始大批量供货，RMB0.99元起(STC15W100)



SOP-8/DIP-8/DFN-8

封装形式：
SOP-8/DIP-8/DFN-8
强烈推荐选择SOP-8贴片封装

全部175℃ 8小时高温烘烤

注意：STC15W10x系列与STC15F104E系列是不同的两个系列，STC15W10x系列采用STC-Y5超高速CPU内核，在相同的时钟频率下，速度比STC早期的1T系列单片机(如STC12/STC11/STC10系列)的速度快20%，而STC104E系列采用的是STC-Y3 CPU内核，速度没有STC15WF104系列的速度快。

每片单片机具有
全球唯一身份证号码(ID号)
无法解密，加密坚不可摧

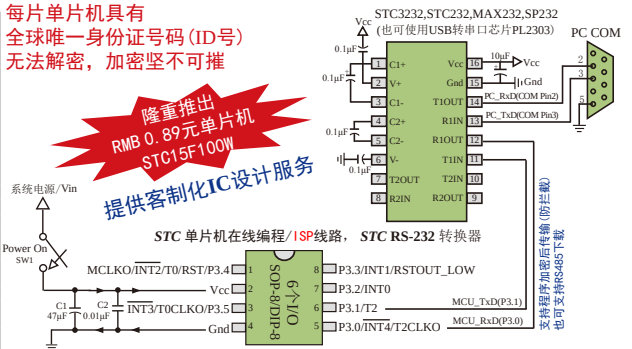
隆重推出 RMB 0.89元单片机 STC15F100W
提供客制化IC设计服务

型号	工作电压(V)	Flash程序存储器(byte)	SRAM字节	EEPROM	串行口	SPI	普通定时器计数器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)	可设下次密钥再升级用户程序	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
STC15W10x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	¥0.99	–	–
STC15W101	5.5-2.5	1K	128	4K	无	无	2	无	有	5个	无	有	有	16级	有	是	有	是	¥1.1	¥1.2	–
STC15W102	5.5-2.5	2K	128	3K	无	无	2	无	有	5个	无	有	有	16级	有	是	有	是	¥1.2	¥1.3	–
STC15W104	5.5-2.5	4K	128	1K	无	无	2	无	有	5个	无	有	有	16级	有	是	有	是	¥1.3	¥1.4	¥1.5
IAP15W105	5.5-2.5	5K	128	IAP	无	无	2	无	有	5个	无	有	有	16级	有	是	有	是	¥1.3	¥1.4	–
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	无	否	用户可将用户程序区的程序FLASH当EEPROM使用 ¥1.3	¥1.4	–
STC15F100W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STC15W10x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	¥0.89	–	–
STC15F101W	5.5-3.8	1K	128	4K	无	无	2	无	有	5个	无	有	有	8级	有	是	有	是	¥0.99	¥1.1	–
STC15F104W	5.5-3.8	4K	128	1K	无	无	2	无	有	5个	无	有	有	8级	有	是	有	是	¥1.2	¥1.3	¥1.4
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	无	否	用户可将用户程序区的程序FLASH当EEPROM使用 ¥1.2	¥1.3	–

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。

IRC15W107型号单片机默认使用内部24MHz时钟，其内部复位门槛电压固定，P5.4不可当复位管脚RST使用，P3.2/P3.3与下载无关，且不支持"程序加密后传输"功能。



大陆本土宏晶STC姚永平独立创新设计，
请不要再抄袭我们的设计、规格和管脚排列，再抄袭就很无...

- 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A/D作按键扫描来节省I/O口

宏晶•STC15W10x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：5.5～2.4V
- 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%温飘(-40℃~+85℃)，常温下温飘±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)；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)：
  ① T0在P3.5输出时钟；② T2在P3.0输出时钟；③ 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)

选择宏晶•STC15W10x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%温飘(-40℃~+85℃)，常温下温飘±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，宏晶第九代加密技术，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：
  1. 高抗静电(ESD保护)整机轻松过2万伏静电测试
  2. 轻松过4kV快速脉冲干扰(EFT测试)
  3. 宽电压，不怕电源抖动　　4. 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：
  1. 掉电模式：外部中断唤醒功耗 <0.2uA
  2. 空闲模式：典型功耗 <1mA，　　3. 正常工作模式：4mA -6mA
  4. 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

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

STC micro™ 宏晶科技

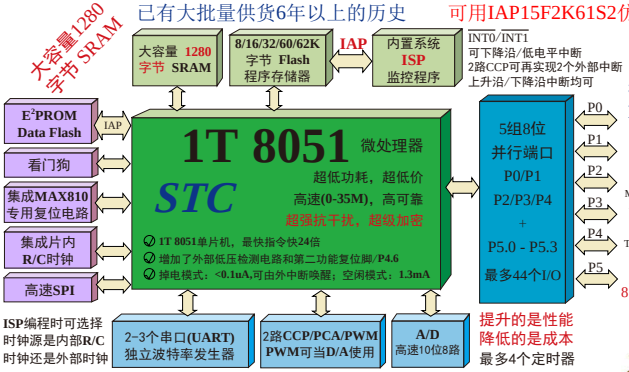
超强抗干扰，超级加密

8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

官方网站：www.STCMCU.com www.GXWMCU.com
南通 Tel: 0513-5501 2928 5501 2929
深圳 Tel: 0755-8294 8411 8294 8412

STC12C5A60S2系列 1T 8051 单片机，2-3个串口，2路CCP/PCA/PWM，高速A/D

已有大批量供货6年以上的历史　可用IAP15F2K61S2仿真(仅供参考)



提供客制化IC设计服务

全部175℃ 8小时高温烘烤

封装形式：
LQFP48(9x9mm)
LQFP44(12x12mm)
PDIP40
PLCC44(不推荐使用)
QFN-40(5x5, 未生产)

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

传统8051单片机划时代升级换代产品，管脚完全兼容，直接取代传统89C51/89S51系列单片机

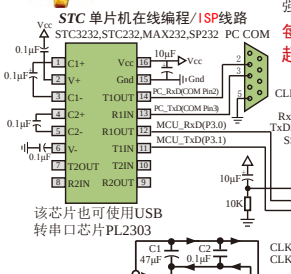
强烈推荐选择LQFP44, LQFP48贴片封装

型号	工作电压（V）	Flash程序存储器(byte)	SRAM 字节	EEP ROM	串行口并可掉电唤醒	SPI	普通定时器 计数器 T0/T1 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	独立波特率发生器	D/A P 8路 25万次每秒 (2路PWM还可当2路D/A使用)	看门狗	内置复位	外部实时低压检测中断	外部复位(可调复位门槛电压)	LQFP48 LQFP44 PDIP40 不推荐PLCC44 部分封装 价格(RMB ¥) LQFP44	LQFP48	PDIP40
STC12C5A60S2系列单片机选型价格一览表 特别提醒：2路CCP/PCA/PWM还可当2路定时器使用，另有STC12LE系列(工作电压2.1V-3.6V)																	
STC12C5A08S2	5.5-4.0	8K	1280	53K	2-3	有	2	2-ch	有	2	10位	有	有	有	有	¥4.2	¥4.2	¥4.5
STC12C5A16S2	5.5-4.0	16K	1280	45K	2-3	有	2	2-ch	有	2	10位	有	有	有	有	¥4.5	¥4.5	¥4.7
STC12C5A32S2	5.5-4.0	32K	1280	29K	2-3	有	2	2-ch	有	2	10位	有	有	有	有	¥4.8	¥4.8	¥5.3
STC12C5A40S2	5.5-4.0	40K	1280	21K	2-3	有	2	2-ch	有	2	10位	有	有	有	有	¥4.9	¥4.9	¥5.4
STC12C5A48S2	5.5-4.0	48K	1280	13K	2-3	有	2	2-ch	有	2	10位	有	有	有	有	¥4.9	¥4.9	¥5.4
STC12C5A56S2	5.5-4.0	56K	1280	5K	2-3	有	2	2-ch	有	2	10位	有	有	有	有	¥4.9	¥4.9	¥5.4
STC12C5A60S2	5.5-4.0	60K	1280	1K	2-3	有	2	2-ch	有	2	10位	有	有	有	有	¥4.9	¥4.9	¥5.4
IAP12C5A62S2	5.5-4.0	62K	1280	IAP	2-3	有	2	2-ch	有	2	10位	有	有	有	有	¥4.9	¥4.9	¥5.4
																用户可将用户程序区的程序Flash当EEPROM使用		

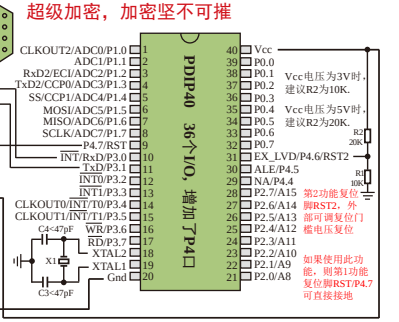
我们直销，所以低价，以上单价为200K/M起定量，以上价格运费由客户承担，零售10片起，如对价格不满，可来电要求降价

宏晶•STC12C5A60S2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
- 宽电压：5.5～4.0V，2.1～3.6V（STC12LE5A60S2系列）
- 增加第二复位功能脚/P4.6(高可靠复位，可调整复位门槛电压，频率<12MHz时，无需此功能)
- 增加外部掉电检测电路/P4.6，可在掉电时，及时将数据保存进EEPROM，正常工作时无需操作EEPROM
- 低功耗设计：空闲模式(可由任意一个中断唤醒)
- 低功耗设计：掉电模式(可由外部中断唤醒)，可支持下降沿/上升沿和远程唤醒
- 支持掉电唤醒的管脚：P3.2/INT0, P3.3/INT1, T0/P3.4, T1/P3.5, RxD/P3.0, P1.3/CCP0（或P4.2/CCP0), P1.4/CCP1（或P4.3/CCP1), EX_LVD/P4.6
- 工作频率：0～35MHz，相当于普通8051：0～420MHz
- 时钟：外部晶体或内部R/C振荡器可选，在ISP下载编程用户程序时设置
- 8/16/32/40/48/56/60/62K字节片内Flash程序存储器，擦写次数10万次以上
- 1280字节片内RAM数据存储器
- 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
- 8通道，10位高速ADC，速度可达25万次/秒，2路PWM还可当2路D/A使用
- 2通道捕获/比较单元(CCP/PCA/PWM)，
  ---也可用来再实现2个定时器或2个外部中断(支持上升沿/下降沿中断)
- 2个16位定时器(兼容普通8051定时器T0/T1)，2路PCA可再实现2个定时器
- 可编程时钟输出功能(T0在P3.4输出时钟，T1在P3.5输出时钟，BRT在P1.0输出时钟
- 硬件看门狗(WDT)
- 独立波特率发生器
- SPI高速同步串行通信接口
- 双串口，全双工异步串行口(UART)，兼容普通8051串口，分时复用可当3组使用
- 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(36/40/44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，但建议整个芯片不要超过120mA



每片单片机具有全球唯一身份证号码(ID号)
超级加密，加密坚不可摧



大客户特价支持

大陆本土宏晶STC姚永平独立创新设计，
请不要再抄袭我们的设计、规格和管脚排列，再抄袭就很无...

复位脚：烧录程序时如设置为I/O口, 可当I/O口使用或浮空
EX_LVD: 是外部低压检测中断/比较器
不用的I/O口：浮空即可

- 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

选择宏晶•STC12C5A60S2系列单片机理由：

★ 超级加密，采用宏晶第六代加密技术
★ 超强抗干扰，超强抗静电，整机轻松过2万伏静电测试
★ 速度快，1个时钟/机器周期，可用低频晶振，大幅降低EMI
　---出口欧美的有力保证
★ 输入/输出口多，最多有44个I/O，A/D做按键扫描还可以节省很多I/O
★ 超低功耗：
　掉电模式：外部中断唤醒功耗 <0.1uA，支持下降沿/上升沿/低电平和远程唤醒
　适用于电池供电系统，如水表、气表、便携设备等。
　空闲模式：典型功耗<1.3mA,
　正常工作模式：2mA -7mA
★ 在系统可编程，无需编程器，无需仿真器，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
★ 内部集成高可靠复位电路，外部复位电路可彻底省掉，当然也可以继续用外部复位电路
★ 全部175℃，8小时高温烘烤，高可靠制造保证

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

STC micro™ 宏晶科技

超强抗干扰，超级加密

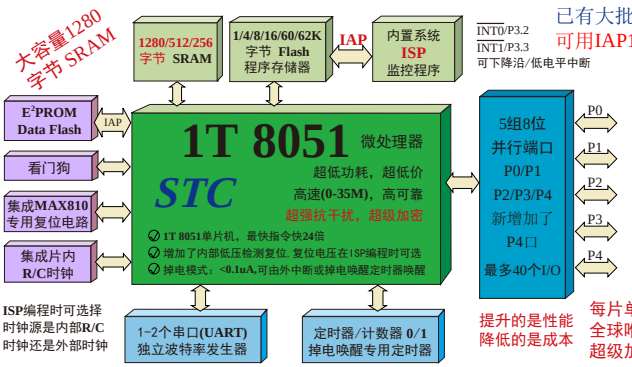
8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

官方网站：www.STCMCU.com www.GXWMCU.com
南通 Tel: 0513-5501 2928 5501 2929
深圳 Tel: 0755-8294 8411 8294 8412

STC11/10xx系列带外部数据总线的 1T 8051 单片机，高速，高可靠，1-2个串口

传统8051单片机划时代升级换代产品，管脚完全兼容，直接取代传统89C51/89S51系列单片机
已有大批量供货6年以上的历史
可用IAP15F2K61S2仿真(仅供参考)

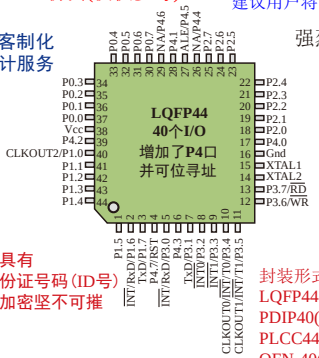
串口作主机通信时，可控制串口通信在[RxD/P3.0,TxD/P3.1]和[RxD/P1.6,TxD/P1.7]之间任意切换，实现2组串口
建议用户将串口设在[RxD/P1.6,TxD/P1.7]



提升的是性能 降低的是成本

每片单片机具有全球唯一身份证号码(ID号)
超级加密，加密坚不可摧

提供客制化IC设计服务



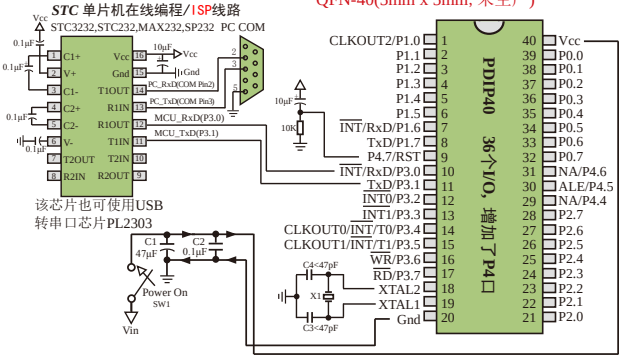
强烈推荐选择LQFP44贴片封装
SOP20/16贴片封装

全部175℃
8小时高温烘烤

封装形式：
LQFP44(12mm x 12mm, 强烈推荐使用)
PDIP40(正常生产但不推荐使用)
PLCC44(不推荐使用)
QFN-40(5mm x 5mm, 未生产)

型号	工作电压(V)	Flash程序存储器(byte)	SRAM字节	EEPROM	串行口并可掉电唤醒	SPI	普通定时器计数器T0/T1外部管脚也能掉电唤醒	独立波特率发生器	掉电唤醒专用定时器	CCP PCA PWM可当外部中断并可掉电唤醒	A/D 8路	看门狗	支持掉电唤醒外部中断	内部低压检测中断	内部复位(可选复位门槛电压)	LQFP44 PDIP40 不推荐PLCC44 部分封装价格(RMB ¥) LQFP44	PDIP40	管脚兼容传统落后型号
STC10Fxx系列单片机选型价格一览表，另有STC10L系列(工作电压2.1V-3.6V)																		
STC10F04	5.5-3.8/3.3	4K	256	-	1-2	无	2	有	-	无	无	有	5个	有	有	¥2.4	¥3.0	89C51
STC10F04XE	5.5-3.8/3.3	4K	512	5K	1-2	无	2	有	-	无	无	有	5个	有	有	¥2.4	¥3.0	89C51
STC10F08	5.5-3.8/3.3	8K	256	-	1-2	无	2	有	-	无	无	有	5个	有	有	¥2.6	¥3.0	89C52
STC10F08XE	5.5-3.8/3.3	8K	512	5K	1-2	无	2	有	-	无	无	有	5个	有	有	¥2.6	¥3.0	89C52
STC10F10XE	5.5-3.8/3.3	10K	512	3K	1-2	无	2	有	-	无	无	有	5个	有	有	¥2.8	¥3.3	89C54
STC10F12XE	5.5-3.8/3.3	12K	512	1K	1-2	无	2	有	-	无	无	有	5个	有	有	¥2.8	¥3.3	89C54
IAP10F14X	5.5-3.8	14K	512	IAP	1-2	无	2	有	-	无	无	有	5个	有	有	¥2.8	¥3.3	用户可将用户程序区的程序Flash当EEPROM用
STC11Fxx系列单片机选型价格一览表，另有STC11L系列(工作电压2.1V-3.6V)																		
STC11F16XE	5.5-4.1/3.7	16K	1280	45K	1-2	无	2	有	有	无	无	有	5个	有	有	¥4.1	¥4.5	89C54
STC11F32XE	5.5-4.1/3.7	32K	1280	29K	1-2	无	2	有	有	无	无	有	5个	有	有	¥4.2	¥4.7	89C58
STC11F56XE	5.5-4.1/3.7	56K	1280	5K	1-2	无	2	有	有	无	无	有	5个	有	有	¥4.3	¥4.8	89C516
STC11F60XE	5.5-4.1/3.7	60K	1280	1K	1-2	无	2	有	有	无	无	有	5个	有	有	¥4.3	¥4.8	89C516
IAP11F62X	5.5-4.1/3.7	62K	1280	IAP	1-2	无	2	有	有	无	无	有	5个	有	有	¥4.3	¥4.8	用户可将用户程序区的程序Flash当EEPROM用

我们直销，所以低价，以上单价为200K/M起定量，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



大陆本土宏晶STC姚永平独立创新设计，
请不要再抄袭我们的设计、规格和管脚排列，再抄袭就很无...

复位脚：烧录程序时如设置为I/O口,可当I/O口使用或浮空
不用的I/O口：浮空即可

- 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

宏晶•STC11/10xx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
- 宽电压：5.5～4.1V/3.7V，2.1/2.4～3.6V(STC11/10L系列)
- 低功耗设计：空闲模式(可由任意一个中断唤醒)
- 低功耗设计：掉电模式(可由任意一个外部中断唤醒，可支持下降沿/低电平和远程唤醒，STC11xx系列还可通过内部掉电唤醒专用定时器唤醒)
- 支持掉电唤醒的管脚：P3.2/INT0,P3.3/INT1,T0/P3.4,T1/P3.5,RxD/P3.0(或RxD/P1.6)
- 内部掉电唤醒专用定时器(只有STC11系列才有，STC10系列无)
- 工作频率：0～35MHz，相当于普通8051：0～420MHz
- 时钟：外部晶体或内部RC振荡器可选，在ISP下载编程用户程序时设置
- 4/8/12/16/32/48/60/62K字节片内Flash程序存储器，擦写次数10万次以上
- 1280/512/256字节片内RAM数据存储器
- 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
- 2个16位定时器，兼容普通8051定时器T0/T1
- 1个独立波特率发生器(故无需T2做波特率发生器)，缺省是T1做波特率发生器
- 可编程时钟输出功能，T0在P3.4输出时钟，T1在P3.5输出时钟，BRT在P1.0输出时钟
- 硬件看门狗（WDT）
- 全双工异步串行口(UART)，兼容普通8051，可当2组串口使用(串口可在P3与P1之间任意切换)
- 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(36/40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，44/40管脚的IC建议整个芯片不要超过120mA

选择宏晶•STC11/10xx系列单片机理由：

★ 超级加密，采用宏晶第六代加密技术
★ 超强抗干扰，超强抗静电，整机轻松过2万伏静电测试
★ 速度快，1个时钟/机器周期，可用低频晶振，大幅降低EMI
  ----出口欧美的有力保证
★ 输入/输出口多，最多有40个I/O，复位脚如当I/O口使用，可省去外部复位电路
★ 超低功耗：
  掉电模式：外部中断唤醒功耗 <0.1uA，支持下降沿/低电平和远程唤醒
  STC11xx系列增加了掉电唤醒专用定时器，启动掉电唤醒定时器典型功耗<2μA
  适用于电池供电系统，如水表、气表、便携设备等。
  空闲模式：典型功耗<1.3mA,
  正常工作模式：2mA -7mA
★ 在系统可编程,无需编程器,无需仿真器，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
★ 内部集成高可靠复位电路,复位脚设置为I/O口使用时，复位脚可浮空

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

STC micro™ 宏晶科技

超强抗干扰，超级加密

8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

官方网站：www.STCMCU.com　www.GXWMCU.com
南通 Tel: 0513-5501 2928　5501 2929
深圳 Tel: 0755-8294 8411　8294 8412

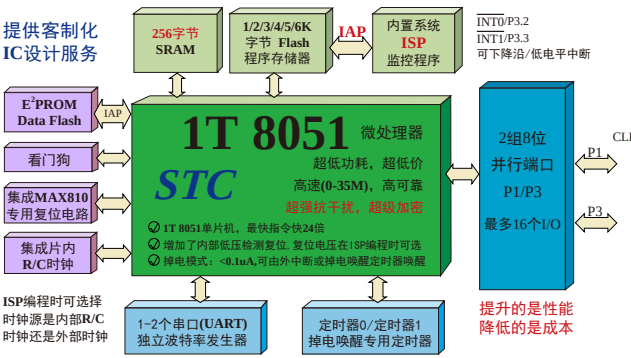
STC11xx系列无外部数据总线 1T 8051 单片机，高速，高可靠，1-2个串口

传统8051单片机划时代升级换代产品，管脚完全兼容，直接取代传统89C51/89S51系列单片机

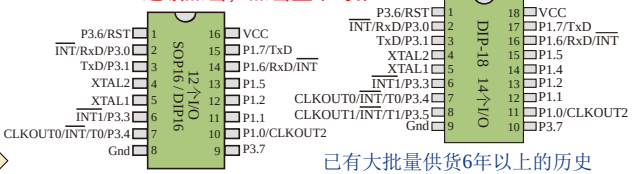
全球最低价：宏晶• STC11F01E-SOP16 RMB 2.00元，STC11F04E RMB 2.3元
宏晶• STC11F02E-SOP16 RMB 2.10元，可取代89C2051/89C4051

每片单片机具有全球唯一身份证号码(ID号)
超级加密，加密坚不可摧

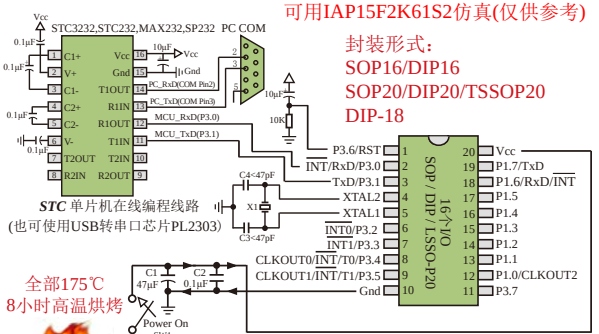
提供客制化
IC设计服务



提升的是性能
降低的是成本



已有大批量供货6年以上的历史
可用IAP15F2K61S2仿真(仅供参考)



封装形式：
SOP16/DIP16
SOP20/DIP20/TSSOP20
DIP-18

STC 单片机在线编程线路
(也可使用USB转串口芯片PL2303)

全部175℃
8小时高温烘烤

强烈推荐选择SOP16/20贴片封装

串口作主机通信时，可控制串口通信在[RxD/P3.0,TxD/P3.1]和[RxD/P1.6,TxD/P1.7]之间任意切换，实现2组串口
建议用户将串口设在[RxD/P1.6,TxD/P1.7]

型号	工作电压(V)	Flash程序存储器(byte)	SRAM字节	EEPROM	串行口并可掉电唤醒	SPI	普通定时器 T0/T1 外部管脚也能掉电唤醒	独立波特率发生器	掉电唤醒专用定时器	CCP PCA PWM 可当外部中断并可掉电唤醒	A/D 8路	看门狗	支持掉电唤醒外部中断	内部低压检测中断	内部复位(可选门槛电压)	SOP20/DIP20/TSSOP20 DIP-18 SOP16/DIP16 部分封装 价格(RMB ¥) SOP16	DIP16	DIP18	SOP20	DIP20	TSSOP20
STC11Fxx系列单片机选型价格一览表																					
STC11F01E	5.5-4.1/3.5	1K	256	2K	1-2	无	2	有	有	无	无	有	5个	有	有	¥2.0	¥2.1	-	¥2.1	¥2.3	¥2.3
STC11F02E	5.5-4.1/3.5	2K	256	2K	1-2	无	2	有	有	无	无	有	5个	有	有	¥2.1	¥2.2	-	¥2.2	¥2.4	¥2.4
STC11F03E	5.5-4.1/3.5	3K	256	2K	1-2	无	2	有	有	无	无	有	5个	有	有	¥2.2	¥2.3	-	¥2.3	¥2.5	¥2.5
STC11F04E	5.5-4.1/3.5	4K	256	1K	1-2	无	2	有	有	无	无	有	5个	有	有	¥2.3	¥2.4	-	¥2.4	¥2.6	¥2.6
STC11F05E	5.5-4.1/3.5	5K	256	1K	1-2	无	2	有	有	无	无	有	5个	有	有	¥2.3	¥2.4	-	¥2.4	¥2.6	¥2.6
IAP11F06	5.5-4.1/3.5	6K	256	IAP	1-2	无	2	有	有	无	无	有	5个	有	有	¥2.3 用户可将用户程序区的程序Flash当EEPROM用	¥2.4	-	¥2.4	¥2.6	¥2.6
STC11Lxx系列单片机选型价格一览表																					
STC11L01E	3.6-2.4/2.1	1K	256	2K	1-2	无	2	有	有	无	无	有	5个	有	有	¥2.0	-	-	¥2.1	-	¥2.3
STC11L02E	3.6-2.4/2.1	2K	256	2K	1-2	无	2	有	有	无	无	有	5个	有	有	¥2.1	-	-	¥2.2	-	¥2.4
STC11L03E	3.6-2.4/2.1	3K	256	2K	1-2	无	2	有	有	无	无	有	5个	有	有	¥2.2	-	-	¥2.3	-	¥2.5
STC11L04E	3.6-2.4/2.1	4K	256	1K	1-2	无	2	有	有	无	无	有	5个	有	有	¥2.3	-	-	¥2.4	-	¥2.6
STC11L05E	3.6-2.4/2.1	5K	256	1K	1-2	无	2	有	有	无	无	有	5个	有	有	¥2.3	-	-	¥2.4	-	¥2.6
IAP11L06	3.6-2.4/2.1	6K	256	IAP	1-2	无	2	有	有	无	无	有	5个	有	有	¥2.3 用户可将用户程序区的程序Flash当EEPROM用	-	-	¥2.4	¥2.6	¥2.6

我们直销，所以低价，**以上单价为200K/M起定量**，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**

大陆本土宏晶STC姚永平独立创新设计，
请不要再抄袭我们的设计、规格和管脚排列，再抄袭就很无. . .

复位脚：烧录程序时如设置为I/O口, 可当I/O口使用或浮空
不用的I/O口：浮空即可

宏晶•STC11xx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
- 宽电压：5.5～4.1V/3.7V，2.1/2.4～3.6V(STC11L系列)
- 低功耗设计：**空闲模式(可由任意一个中断唤醒)**
- 低功耗设计：掉电模式**(可由任意一个外部中断唤醒，可支持下降沿/低电平和**远程唤醒，STC11xx系列还可通过内部掉电唤醒专用定时器唤醒)
- 支持掉电唤醒的管脚：P3.2/INT0,P3.3/INT1,T0/P3.4,T1/P3.5,RxD/P3.0(或RxD/P1.6)
- 内部掉电唤醒专用定时器(只有STC11系列才有，STC10系列无)
- 工作频率：0～35MHz，相当于普通8051：0～420MHz
- 时钟：外部晶体或内部RC振荡器可选，在ISP下载编程用户程序时设置
- 1/2/3/4/5/6K字节片内Flash程序存储器，擦写次数10万次以上
- 256字节片内RAM数据存储器
- 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
- 2个16位定时器，兼容普通8051定时器T0/T1
- 1个独立波特率发生器(故无需T2做波特率发生器)，缺省是T1做波特率发生器
- 可编程时钟输出功能，T0在P3.4输出时钟，T1在P3.5输出时钟
- 硬件看门狗（WDT）
- 全双工异步串行口(UART)，兼容普通8051，可当2组串口使用(串口可在P3与P1之间任意切换)
- 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(16/14/12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, 20/18/16管脚的IC建议整个芯片不要超过70mA
- 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

选择宏晶•STC11xx系列单片机理由：

★ 超级加密，采用宏晶第六代加密技术
★ 超强抗干扰，超强抗静电，整机轻松过2万伏静电测试
★ 速度快，1个时钟/机器周期，可用低频晶振，大幅降低EMI
　----出口欧美的有力保证
★ 输入/输出口多，最多有16个I/O，复位脚如当I/O口使用，可省去外部复位电路
★ 超低功耗：
　掉电模式：**外部中断唤醒功耗** <0.1uA，支持下降沿/低电平和远程唤醒
　STC11xx系列增加了掉电唤醒专用定时器，启动掉电唤醒定时器典型功耗<2μA
　　适用于电池供电系统，如水表、气表、便携设备等。
　空闲模式：**典型功耗**<1.3mA,
　正常工作模式：2mA -7mA
★ 在系统可编程, 无需编程器, 无需仿真器，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
★ 内部集成高可靠复位电路, 复位脚设置为I/O口使用时，复位脚可浮空

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

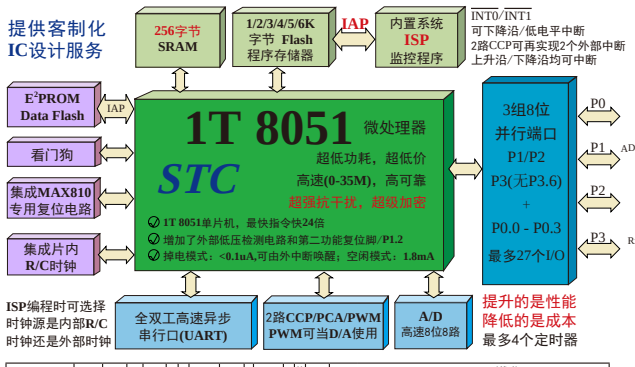
STC micro™ 宏晶科技

超强抗干扰，超级加密

8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

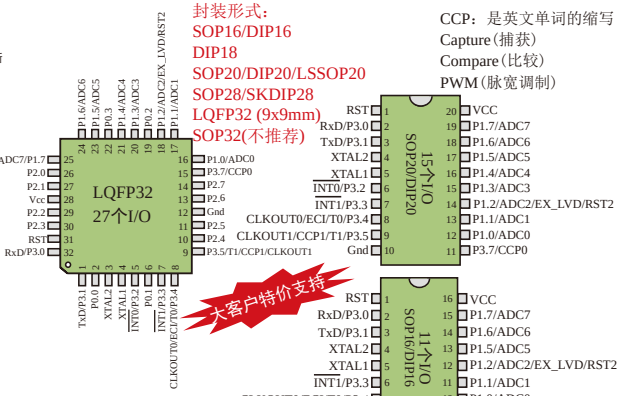
官方网站：www.STCMCU.com　www.GXWMCU.com
南通 Tel: 0513-5501 2928　5501 2929
深圳 Tel: 0755-8294 8411　8294 8412

STC12C5201AD系列 1T 8051 单片机，超低价，2路CCP/PCA/PWM，高速A/D



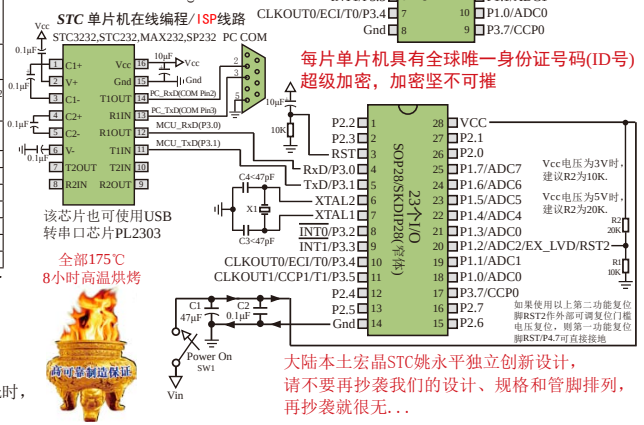
封装形式：
SOP16/DIP16
DIP18
SOP20/DIP20/LSSOP20
SOP28/SKDIP28
LQFP32 (9x9mm)
SOP32(不推荐)

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



型号	工作电压(V)	Flash程序存储器(byte)	SRAM字节	EEPROM	串行口可掉电唤醒	SPI	普通定时器计数器	CCP PCA PWM可当外部中断并可掉电唤醒	DPTR	A/D 8路 30万次每秒(2路PWM可当2路D/A使用)	外部实时掉电检测电压	外部复位可调门槛电压	外部掉电中断	LQFP32/SOP32(不推荐) SOP28/SKDIP28 SOP20/DIP20/TSSOP20 SOP16/DIP16 部分封装价格(RMB ¥) SOP16价格	DIP16价格	SOP20价格	DIP20价格	TSSOP20价格	SOP28价格	SKDIP28价格	LQFP32价格	SOP32价格
STC12C5201AD系列单片机选型价格一览表 特别提醒：2路CCP/PCA/PWM还可当2路定时器使用，另有STC12LE系列(工作电压2.4V-3.6V)																						
STC12C5201AD	5.5-4.0	1K	256	2K	1	无	2	2-ch	1	8位	有	有	有	¥2.3	¥2.4	¥2.4	¥2.6	¥2.6				
STC12C5202AD	5.5-4.0	2K	256	2K	1	无	2	2-ch	1	8位	有	有	有	¥2.4	¥2.5	¥2.5	¥2.7	¥2.7	¥2.8	¥3.0	¥3.0	¥3.1
STC12C5203AD	5.5-4.0	3K	256	2K	1	无	2	2-ch	1	8位	有	有	有	¥2.5	¥2.6	¥2.6	¥2.8	¥2.8	¥2.8	¥3.0	¥3.1	¥3.2
STC12C5204AD	5.5-4.0	4K	256	1K	1	无	2	2-ch	1	8位	有	有	有	¥2.6	¥2.7	¥2.7	¥2.9	¥2.9	¥2.8	¥3.0	¥3.1	¥3.2
STC12C5205AD	5.5-4.0	5K	256	1K	1	无	2	2-ch	1	8位	有	有	有	¥2.6	¥2.7	¥2.7	¥2.9	¥2.9	¥2.8	¥3.0	¥3.1	¥3.2
STC12C5206AD	5.5-4.0	6K	256	IAP	1	无	2	2-ch	1	8位	有	有	有	¥2.6	¥2.7	¥2.7	¥2.9	¥2.9	¥2.8	¥3.0	¥3.1	¥3.2
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200K/M起定量，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



宏晶•STC12C5201AD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
- 宽电压：5.5～4.0V，2.2～3.6V(STC12LE5201AD系列)
- 增加第二复位功能脚/P1.2(高可靠复位，可调整复位门槛电压，频率<12MHz时，无需此功能)
- 增加外部掉电检测电路/P1.2，可在掉电时，及时将数据保存进EEPROM，正常工作时无需操作EEPROM
- 低功耗设计：空闲模式(可由任意一个中断唤醒)，掉电模式(可由外部中断唤醒)
- 支持掉电唤醒的管脚：P3.2/INT0, P3.3/INT1, T0/P3.4, T1/P3.5, RxD/P3.0, P3.7/CCP0, P3.5/CCP1, EX_LVD/P1.2
- 工作频率：0～35MHz，相当于普通8051：0～420MHz
- 时钟：外部晶体或内部RC振荡器可选，在ISP下载编程用户程序时设置
- 1K/2K/3K/4K/5K/6K字节片内Flash程序存储器，擦写次数10万次以上
- 256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
- 8通道，8位高速ADC，速度可达30万次/秒，2路PWM还可当2路D/A使用
- 2通道捕获/比较单元(CCP/PCA/PWM)，
  ---也可用来再实现2个定时器或2个外部中断(支持上升沿/下降沿中断)
- 2个16位定时器，兼容普通8051的定时器T0/T1, 2路PCA可再实现2个定时器
- 可编程时钟输出功能，T0在P3.4输出时钟，T1在P3.5输出时钟
- 硬件看门狗（WDT）
- 全双工异步串行口(UART)，兼容普通8051的串口
- 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(27/23/15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，但建议整个芯片不要超过90mA
- 如选32-Pin，推荐优先选择LQFP32

复位脚：烧录程序时如设置为I/O口, 可当I/O口使用或浮空
EX_LVD：是外部低压检测中断/比较器
不用的I/O口：浮空即可

可用IAP15F2K61S2仿真(仅供参考)

- 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

选择宏晶•STC12C5201AD系列单片机理由：

★ 超级加密，采用宏晶第六代加密技术
★ 超强抗干扰：
1. 高抗静电(ESD保护)，整机轻松过2万伏静电测试
2. 轻松过4KV快速脉冲干扰(EFT测试)
3. 宽电压，不怕电源抖动　　4. 宽温度范围，-40℃～+85℃

★ 速度快，1个时钟/机器周期，可用低频晶振，大幅降低EMI
----出口欧美的有力保证
★ 超低功耗：
1. 掉电模式:外部中断唤醒功耗 <0.1uA
2. 空闲模式：典型功耗 1.8mA,
3. 正常工作模式:2.7mA -7mA
4. 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
★ 在系统可编程，无需编程器，无需仿真器，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
★ 内部集成高可靠复位电路，外部复位电路可彻底省掉，当然也可以继续用外部复位电路
★ 全部175℃，8小时高温烘烤，高可靠制造保证

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

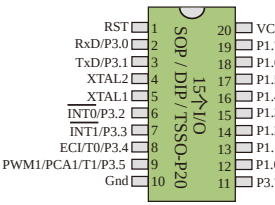
STC micro™ 宏晶科技

超强抗干扰，超级加密

8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

官方网站：www.STCMCU.com www.GXWMCU.com
南通 Tel: 0513-5501 2928 5501 2929
深圳 Tel: 0755-8294 8411 8294 8412

STC12C5620AD系列 1T 8051 单片机，4路PWM/PCA，8路10位A/D转换



超小封装TSSOP20, 6.4mm×6.4mm

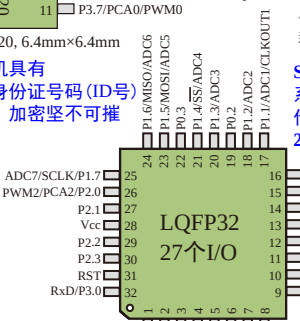
封装形式：
SOP20/DIP20/TSSOP20
SOP28/SKDIP28
LQFP32 (9x9mm)
SOP32(不推荐)

强烈推荐选择SOP20/28, LQFP32贴片封装

传统插件DIP封装稳定供货

每片单片机具有
全球唯一身份证号码(ID号)
超级加密，加密坚不可摧

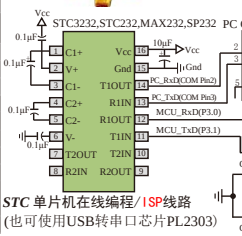
STC12C5620AD系列可直接取代传统12C5410/2052AD系列

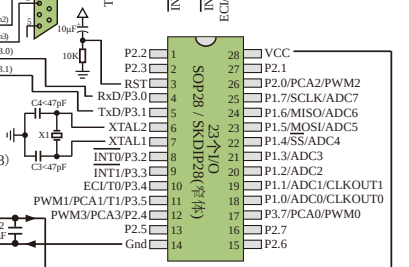


全部175℃ 8小时高温烘烤

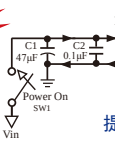
型号	工作电压(V)	Flash程序存储器(byte)	SRAM字节	EEPROM	串行口	串行口可同时工作	SPI	掉电唤醒	普通定时器计数器T0/T1外部管脚也能掉电唤醒	PCA PWM可当外部中断并可掉电唤醒	DPTR	A/D 8路10万次每秒(4路PWM可当4路D/A使用)	看门狗	内部低压检测中断	内部复位(可选复位门槛电压)	LQFP32/SOP32(不推荐) SOP28/SKDIP28 SOP20/DIP20/TSSOP20 部分封装价格(RMB ¥) SOP20价格	DIP20价格	TSSOP20价格	SOP28价格	SKDIP28价格	LQFP32价格	SOP32价格
STC12C5620AD系列单片机选型价格一览表 特别提醒：4路CCP/PCA/PWM还可当4路定时器使用，另有STC12LE系列(工作电压2.4V-3.6V)																					
STC12C5604AD	5.5-3.5	4K	768	4K	1	有	2	4-ch	1	10位	有	有	有	¥3.10	¥3.20	¥3.20	¥3.20	¥3.40	¥3.40	¥3.45
STC12C5608AD	5.5-3.5	8K	768	4K	1	有	2	4-ch	1	10位	有	有	有	¥3.30	¥3.40	¥3.40	¥3.40	¥3.50	¥3.50	¥3.55
STC12C5612AD	5.5-3.5	12K	768	IAP	1	有	2	4-ch	1	10位	有	有	有	¥3.30	¥3.40	¥3.40	¥3.40	¥3.50	¥3.50	¥3.55
														用户可将用户程序区的程序Flash当EEPROM使用						
STC12C5616AD	5.5-3.5	16K	768	4K	1	有	2	4-ch	1	10位	有	有	有	¥3.70	¥3.80	¥3.80	¥3.80	¥4.00	¥4.00	¥4.05
STC12C5620AD	5.5-3.5	20K	768	4K	1	有	2	4-ch	1	10位	有	有	有	¥5.80	¥5.90	¥5.90	¥5.90	¥6.00	¥6.00	¥6.05
STC12C5624AD	5.5-3.5	24K	768	4K	1	有	2	4-ch	1	10位	有	有	有	¥5.80	¥5.90	¥5.90	¥5.90	¥6.00	¥6.00	¥6.05
STC12C5628AD	5.5-3.5	28K	768	IAP	1	有	2	4-ch	1	10位	有	有	有	¥5.80	¥5.90	¥5.90	¥5.90	¥6.00	¥6.00	¥6.05
														用户可将用户程序区的程序Flash当EEPROM使用						
STC12C5630AD	5.5-3.5	30K	768	IAP	1	有	2	4-ch	1	10位	有	有	有	¥5.80	¥5.90	¥5.90	¥5.90	¥6.00	¥6.00	¥6.05
														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200K/M起定量，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**





大客户特价支持



提供客制化IC设计服务

可用IAP15F2K61S2仿真(仅供参考)

大陆本土宏晶STC姚永平独立创新设计，请不要再抄袭
我们的设计、规格和管脚排列，再抄袭就很无...

宏晶•STC12C5620AD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8～12倍
- 宽电压：5.5～3.5V，2.2～3.6V(STC12LE5620AD系列)
- 低功耗设计：**空闲模式(可由任意一个中断唤醒)，掉电模式(可由外部中断唤醒)**
- 工作频率：0～35MHz，相当于普通8051：0～420MHz
- 时钟：外部晶体或内部RC振荡器可选，在ISP下载编程用户程序时设置
- 30K/28K/24K/20K/16K/12K/8K/4K字节片内Flash程序存储器，擦写次数10万次以上
- 768字节(256+512)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
- 8通道，10位ADC，速度可达10万次/秒，4路PWM还可当4路D/A使用
- 4通道捕获/比较单元(PWM/PCA)，
  ---也可用来再实现4个定时器或4个外部中断(支持上升沿/下降沿中断)
- 2个16位定时器，兼容普通8051的定时器T0/T1，4路PCA可再实现4个定时器
- 可编程时钟输出功能，T0在P1.0输出时钟，T1在P1.1输出时钟
- 硬件看门狗(WDT)
- SPI高速**同步串行通信端口**
- 全双工异步串行口(UART)，兼容普通8051的串口
- 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(27/23/15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，但建议整个芯片不要超过90mA
- ★ 如选32-Pin，推荐选LQFP32
- ★ 如果I/O口不够用，可以用3根普通I/O端口外接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，还可用A/D做按键扫描来节省I/O口

选择宏晶•STC12C5620AD系列单片机理由：

- ★ 超级加密，采用宏晶第六代加密技术
- ★ 超强抗干扰：
  1. 高抗静电(ESD保护)，整机轻松过2万伏静电测试
  2. 轻松过4KV快速脉冲干扰(EFT测试)
  3. 宽电压，不怕电源抖动　　4. 宽温度范围，-40℃~+85℃
- ★ 速度快，1个时钟/机器周期，可用低频晶振，大幅降低EMI
  ----出口欧美的有力保证
- ★ 超低功耗：
  1. **掉电模式:外部中断唤醒功耗** <0.1uA
  2. 空闲模式：典型功耗 1.8mA，　　3. 正常工作模式:2.7mA -7mA
  4. 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
- ★ 在系统可编程，无需编程器，无需仿真器，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
- ★ 内部集成专用**复位电路，有2级复位门槛电压可选，**12MHz以下可放心使用内部复位，**外部复位电路可保留，也可以不用(复位脚直接接地)**

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

STC micro™ 宏晶科技

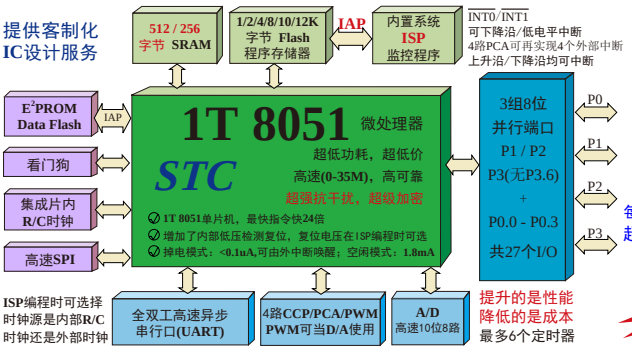
超强抗干扰，超级加密

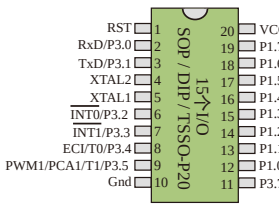
8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

官方网站：www.STCMCU.com www.GXWMCU.com
南通 Tel: 0513-5501 2928 5501 2929
深圳 Tel: 0755-8294 8411 8294 8412

STC12C5410AD/2052AD系列 1T 8051 单片机，4路PWM/PCA，8路10位A/D

提供客制化IC设计服务



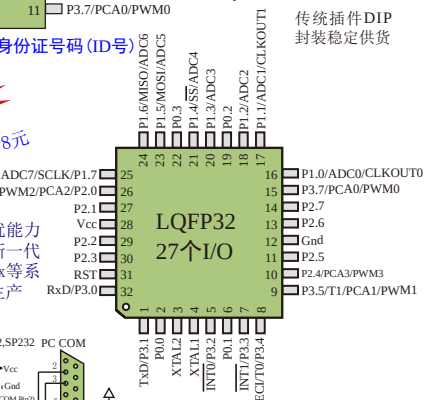


封装形式：
SOP20/DIP20/TSSOP20
SOP28/SKDIP28
LQFP32 (9x9mm)
SOP32(不推荐)
超小封装TSSOP20, 6.4mm x 6.4mm
强烈推荐选择SOP20/28, LQFP32贴片封装

每片单片机具有全球唯一身份证号码(ID号)
超级加密，加密坚不可摧

传统插件DIP 封装稳定供货

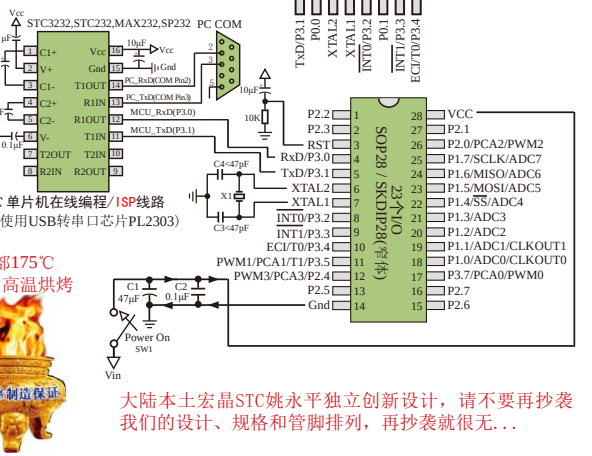
大客户特价支持
STC12C1052, 人民币3.8元



尽量优先选择成本更低，抗干扰能力更强，采用新加密技术的宏晶新一代STC15xx/11xx/12C52xx/12C56xx等系列取代，原有老产品继续长期生产

型号	工作电压(V)	Flash程序存储器(byte)	SRAM字节	EEPROM	串行口并可掉电唤醒	SPI	普通定时器计数器T0/T1外部管脚也能掉电唤醒	PCA PWM可当外部中断并可掉电唤醒	DPTR	A/D 8路10万次每秒(4路PWM可当4路D/A使用)	看门狗	内部复位(可选复位门槛电压)	LQFP32/SOP32(不推荐) SOP28/SKDIP28 SOP20/DIP20/TSSOP20 部分封装价格(RMB ¥) SOP20价格(RMB ¥)	SOP28价格(RMB ¥)	LQFP32价格(RMB ¥)
STC12C5410AD系列单片机选型价格一览表 特别提醒：4路CCP/PCA/PWM还可当4路定时器使用，另有STC12LE系列(工作电压2.2V-3.6V)															
STC12C5401AD	5.5-3.5	1K	512	有	1	有	2	4-ch	1	10位	有	有			
STC12C5402AD	5.5-3.5	2K	512	有	1	有	2	4-ch	1	10位	有	有			
STC12C5404AD	5.5-3.5	4K	512	有	1	有	2	4-ch	1	10位	有	有			
STC12C5406AD	5.5-3.5	6K	512	有	1	有	2	4-ch	1	10位	有	有			
STC12C5408AD	5.5-3.5	8K	512	有	1	有	2	4-ch	1	10位	有	有			
STC12C5410AD	5.5-3.5	10K	512	有	1	有	2	4-ch	1	10位	有	有			
STC12C5412AD	5.5-3.5	12K	512	IAP	1	有	2	4-ch	1	10位	有	有	用户可将用户程序区的程序Flash当EEPROM使用		
STC12C2052AD系列单片机选型价格一览表 特别提醒：2路CCP/PCA/PWM还可当2路定时器使用，另有STC12LE系列(工作电压2.2V-3.6V)															
STC12C1052AD	5.5-3.5	1K	256	有	1	有	2	2-ch	1	8位	有	有	SOP20/DIP20/TSSOP20		
STC12C2052AD	5.5-3.5	2K	256	有	1	有	2	2-ch	1	8位	有	有	SOP20/DIP20/TSSOP20		
STC12C4052AD	5.5-3.5	4K	256	有	1	有	2	2-ch	1	8位	有	有	SOP20/DIP20/TSSOP20		
STC12C5052AD	5.5-3.5	5K	256	IAP	1	有	2	2-ch	1	8位	有	有	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200K/M起定量，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**
封装为SKDIP28的单片机比封装为SOP28的单片机要贵0.2元；
封装为DIP20的单片机比封装为SOP20的单片机要贵0.2元。



STC 单片机在线编程/ISP线路
(也可使用USB转串口芯片PL2303)

可用IAP15F2K61S2仿真(仅供参考)

全部175℃ 8小时高温烘烤

大陆本土宏晶STC姚永平独立创新设计，请不要再抄袭我们的设计、规格和管脚排列，再抄袭就很无...

宏晶•STC12C5410AD/2052AD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
- 宽电压：5.5～3.5V，2.2～3.8V(STC12LE5410AD/2052AD系列)
- 低功耗设计：**空闲模式(可由任意一个中断唤醒)，掉电模式(可由外部中断唤醒)**
- 工作频率：0～35MHz，相当于普通8051：0～420MHz
- 时钟：外部晶体或内部RC振荡器可选，在ISP下载编程用户程序时设置
- 16K/12K/10K/8K/6K/4K/2K/1K节片内Flash程序存储器，擦写次数10万次以上
- 512 / 256 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
- 8通道，10位ADC，STC12C2052AD系列为8位ADC，4路PWM还可当4路D/A使用
- 4通道捕获/比较单元(PWM/PCA)，STC12C2052AD系列为2通道
  ---也可用来再实现4个定时器或4个外部中断(支持上升沿/下降沿中断)
- 2个16位定时器，兼容普通8051的定时器T0/T1，4路PCA可再实现4个定时器
- 可编程时钟输出功能，T0在P1.0输出时钟，T1在P1.1输出时钟
- 硬件看门狗(WDT)
- SPI高速**同步串行通信端口**
- 全双工异步串行口(UART)，兼容普通8051的串口
- 先进的指令集结构，兼容普通8051指令集
  4组8个8位通用工作寄存器(共32个通用寄存器)
  有硬件乘法/除法指令
- 通用I/O口(27/23/15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，但建议整个芯片不要超过90mA

★ 如选32-Pin，推荐选LQFP32
★ 如果I/O口不够用，可以用3根普通I/O端口外接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，还可用A/D做按键扫描来节省I/O口

选择宏晶•STC12C5410AD/2052AD系列单片机理由：

★ 超级加密
★ 超强抗干扰：
1. 高抗静电(ESD保护)　2. 轻松过4KV快速脉冲干扰(EFT测试)
3. 宽电压，不怕电源抖动　4. 宽温度范围，-40℃~+85℃
★ 速度快，1个时钟/机器周期，可用低频晶振，大幅降低EMI
----出口欧美的有力保证
★ 超低功耗：
1. **掉电模式:外部中断唤醒功耗** <0.1uA
2. 空闲模式：典型功耗 1.8mA，　3. 正常工作模式:2.7mA -7mA
4. **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**
★ 所有封装均符合欧盟RoHS要求
★ 在系统可编程，无需编程器，无需仿真器，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

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

STC micro™ 宏晶科技

超强抗干扰，加密性强

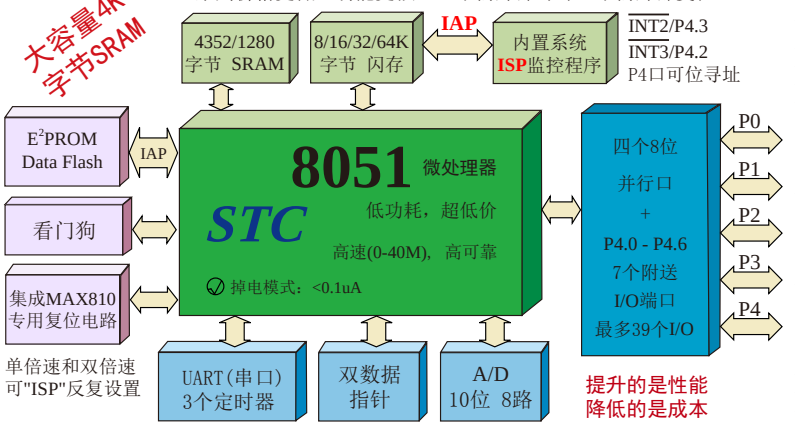
8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

官方网站：www.STCMCU.com　www.GXWMCU.com
南通 Tel: 0513-5501 2928　5501 2929
深圳 Tel: 0755-8294 8411　8294 8412

宏晶·STC 89C51 90C51 系列单片机，直接兼容传统8051

欢迎使用STC90C51系列(软硬件完全兼容)取代89C51系列，STC90C51系列抗干扰能力更强，成本更低
或请使用第六代加密技术加密、加密性强的STC11/10xx系列取代89C51系列
或强烈建议使用第九代加密技术加密、无法解密的STC15xx系列取代89C51系列，
STC15系列价格更低、功能更强，且不需外部时钟、不需外部复位

已有大批量供货10年以上的历史
可用IAP15F2K61S2仿真(仅供参考)



选择宏晶·STC90C51系列单片机理由：

★ 超大容量SRAM，最高达4.2K字节SRAM
★ 直接取代传统8051单片机，成本更低
★ 10位A/D，可达25万次/秒
★ 超强抗干扰：
1. 高抗静电(ESD保护，整机轻松过2万伏静电测试)
2. 轻松过4KV快速脉冲干扰(EFT测试)
3. 宽电压，不怕电源抖动
4. 宽温度范围，-40℃~+85℃
★ 三大降低单片机时钟对外部电磁辐射的措施：
----出口欧美的有力保证
1. 禁止ALE输出；
2. 如选6时钟/机器周期，外部时钟频率可降低一半；
3. 单片机时钟振荡器增益可设为1/2gain
★ 超低功耗：
1. 掉电模式：典型功耗功耗 <0.1uA
2. 正常工作模式：4mA -7mA
3. 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。
★ 在系统可编程，无需编程器及仿真器，可远程升级
★ 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
★ 全部175℃，8小时高温烘烤，高可靠制造保证

型号	工作电压(V)	Flash程序存储器(字节)	SRAM字节	EEPROM(字节)	定时器	A/D 10位8路	降低EMI	双倍速	I/O口	支持掉电唤醒外部中断	内置复位	看门狗	ISP	IAP	LQFP44价格	PDIP40价格	PLCC44价格	兼容传统落后型号	价格更低 功能更强 不需外部时钟 不需外部复位的替代型号
STC89C51RC	5.5-3.5	4K	512	9K	3个	-	√	√	39个	4个	有	有	√	√	¥2.8	¥3.3	¥3.4	89C51	
STC90C51RC	5.5-3.5	4K	512	9K	3个	-	√	√	39个	4个	有	有	√	√	¥2.7	¥3.2	¥3.3	89C51	STC15W404S
STC15W404S	5.5-2.5	4K	512	9K	3个	-	√		42个	5个	强	强	√	√	¥2.5	¥3.0			
STC89C52RC	5.5-3.5	8K	512	5K	3个	-	√	√	39个	4个	有	有	√	√	¥2.8	¥3.1	¥3.4	89C52	
STC90C52RC	5.5-3.5	8K	512	5K	3个	-	√	√	39个	4个	有	有	√	√	¥2.7	¥3.0	¥3.3	89C52	STC15W408S
STC15W408S	5.5-2.5	8K	512	5K	3个	-	√		42个	5个	强	强	√	√	¥2.7	¥3.0			
STC89C53RC	5.5-3.5	12K	512	-	3个	-	√	√	39个	4个	有	有	√	√	¥3.5	¥4.0	¥4.1	89C53	
STC90C53RC	5.5-3.5	12K	512	-	3个	-	√	√	39个	4个	有	有	√	√	¥3.4	¥3.9	¥4.0	89C54	IAP15W413S
IAP15W413S	5.5-2.5	13K	512	IAP	3个	-	√		42个	5个	强	强	√	√	¥2.8	¥3.3			
STC89C54RD+	5.5-3.5	16K	1280	45K	3个	-	√	√	39个	4个	有	有	√	√	¥4.5	¥5.0	¥5.1	89C54	
STC90C54RD+	5.5-3.5	16K	1280	45K	3个	-	√	√	39个	4个	有	有	√	√	¥4.4	¥4.9	¥5.0	89C54	STC15W1K16S
STC15W1K16S	5.5-2.6	16K	1024	13K	3个	-	√		42个	5个	强	强	√	√	¥3.5	¥4.0			
STC89C58RD+	5.5-3.5	32K	1280	29K	3个	-	√	√	39个	4个	有	有	√	√	¥4.5	¥5.0	¥5.1	89C58	
STC90C58RD+	5.5-3.5	32K	1280	29K	3个	-	√	√	39个	4个	有	有	√	√	¥4.4	¥4.9	¥5.0	89C58	IAP15W1K29S
IAP15W1K29S	5.5-2.6	29K	1024	IAP	3个	-	√		42个	5个	强	强	√	√	¥3.8	¥4.3			
STC89C516RD+	5.5-3.5	61K	1280		3个	-	√	√	39个	4个	有	有	√	√	¥4.5	¥5.0	¥5.1	89C516	
STC90C516RD+	5.5-3.5	61K	1280		3个	-	√	√	39个	4个	有	有	√	√	¥4.4	¥4.9	¥5.0	89C516	IAP15F2K61S
IAP15F2K61S	5.5-3.8	61K	2048	IAP	3个	-	√		42个	5个	强	强	√	√	¥4.0	¥4.5			
STC90C510RD+	5.5-3.5	40K	1280	21K	3个	-	√	√	39个	4个	有	有	√	√	¥4.4	¥4.9	¥5.0	89C516	
STC90C512RD+	5.5-3.5	48K	1280	13K	3个	-	√	√	39个	4个	有	有	√	√	¥4.4	¥4.9	¥5.0	89C516	
STC90C514RD+	5.5-3.5	56K	1280	5K	3个	-	√	√	39个	4个	有	有	√	√	¥4.4	¥4.9	¥5.0	89C516	
STC90C54AD	5.5-3.8	16K	4352	45K	3个	有	√	√	39个	4个	有	有	√	√	¥7.4	¥7.9	¥8.0	特别要求:A/D输入电压需<1/2Vcc	
STC90C58AD	5.5-3.8	32K	4352	29K	3个	有	√	√	39个	4个	有	有	√	√	¥8.4	¥8.9	¥9.0		
STC90C514AD	5.5-3.8	56K	4352	5K	3个	有	√	√	39个	4个	有	有	√	√	¥9.4	¥9.9	¥10.0		
STC90C516AD	5.5-3.8	61K	4352	-	3个	有	√	√	39个	4个	有	有	√	√	¥10.4	¥10.9	¥11.0		

我们直销，所以低价，**以上单价为10K/M起定量，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**

用"不需外部时钟/外部复位"的"STC15W404S/STC15W408S/IAP15W413S, 取代89C51/89C52/89C53, 成本更低
用"不需外部时钟/外部复位"的"STC15W1K16S/STC15W1K24S/IAP15W1K29S" 取代89C54/89C58, 成本更低
用"不需外部时钟/外部复位"的"IAP15F2K61S" 取代89C54/89C58/89C516, 成本更低

宏晶STC90C58AD系列单片机(带A/D转换)的
串口可从[P3.0/RxD，P3.1/TxD]切换到
[P1.6/RxD，P1.7/TxD]，可当双串口使用

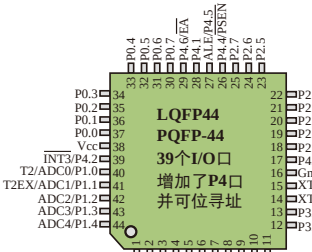
宏晶STC90C58AD系列(带A/D转换)P4口地址在C0H，
其它不带A/D转换的89/90系列单片机P4口地址在E8H

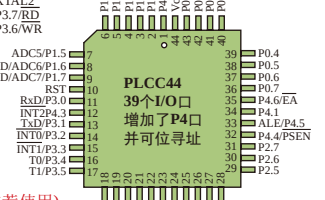
免费索取

从网上下载样品申请单，
传真至深圳申请STC单片机
样片及ISP下载/编程工具

★强烈推荐LQFP44小型封装★
★尽量不选落后的PDIP和PLCC封装★

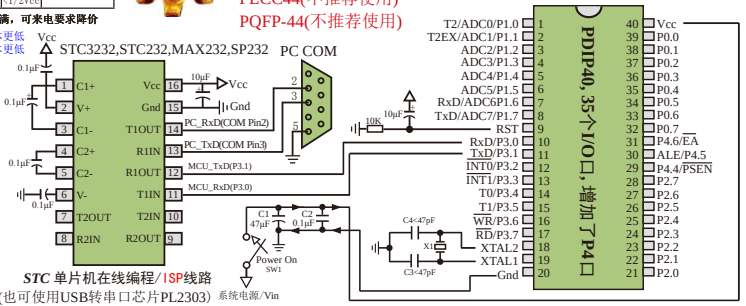
提供客制化IC设计服务





全部175℃
8小时高温烘烤

封装形式：
PDIP40(正常生产，不推荐使用)
LQFP44(12x12mm，强烈推荐)
PLCC44(不推荐使用)
PQFP-44(不推荐使用)



宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！