

STC8 系列单片机 技术参考手册

STCMCU

目录

1	概述.....	1
2	特性及价格.....	3
2.1	STC8A8K64S4A12 系列特性及价格	3
2.2	STC8A4K64S2A12 系列特性及价格	5
2.3	STC8F2K64S4 系列特性及价格	7
2.4	STC8F2K64S2 系列特性及价格	9
2.5	STC8F1K08S2 系列特性及价格	11
2.6	STC8C1K12 系列特性及价格	13
2.7	STC8P1K08S2A10 系列提前通告.....	15
2.8	STC8P1K16S2A10 系列提前通告.....	16
2.9	STC8P2K32S4A12 系列提前通告.....	17
2.10	STC8H04A10 系列提前通告	18
2.11	STC8H04 系列提前通告	19
3	管脚及说明.....	20
3.1	管脚图	20
3.1.1	STC8A8K64S4A12 系列管脚图	20
3.1.2	STC8A4K64S2A12 系列管脚图	24
3.1.3	STC8F2K64S4 系列管脚图	28
3.1.4	STC8F2K64S2 系列管脚图	31
3.1.5	STC8F1K08S2 系列管脚图	35
3.1.6	STC8C1K08S2A10 系列管脚图	37
3.1.7	GX8S003 系列管脚图	38
3.1.8	STC8P1K08S2A10 系列管脚图	39
3.1.9	STC8P1K16S2A10 系列管脚图	40
3.1.10	STC8P2K32S4A12 系列管脚图	41
3.2	管脚说明	42
3.2.1	STC8A8K64S4A12 系列管脚说明	42
3.2.2	STC8A4K64S2A12 系列管脚说明	49
3.2.3	STC8F2K64S4 系列管脚说明	56
3.2.4	STC8F2K64S2 系列管脚说明	60
3.3	功能脚切换	64
3.3.1	功能脚切换相关寄存器	64
3.4	范例程序	67
3.4.1	串口 1 切换	67
3.4.2	串口 2 切换	68
3.4.3	串口 3 切换	68
3.4.4	串口 4 切换	69
3.4.5	SPI 切换	69
3.4.6	PWM 切换	70
3.4.7	PCA/CCP/PWM 切换	72

3.4.8	I2C切换.....	73
3.4.9	比较器输出切换.....	73
3.4.10	主时钟输出切换.....	74
4	封装尺寸图.....	76
4.1	LQFP64S封装尺寸图 (12mm*12mm)	76
4.2	LQFP64L封装尺寸图 (16mm*16mm)	77
4.3	LQFP48 封装尺寸图 (9mm*9mm)	78
4.4	LQFP44 封装尺寸图 (12mm*12mm)	81
4.5	LQFP32 封装尺寸图 (9mm*9mm)	84
4.6	QFN64 封装尺寸图 (8mm*8mm)	85
4.7	QFN48 封装尺寸图 (6mm*6mm)	86
4.8	QFN32 封装尺寸图 (4mm*4mm)	87
4.9	PDIP40 封装尺寸图	88
4.10	TSSOP20 封装尺寸图	89
4.11	SOP16 封装尺寸图.....	90
4.12	STC8 系列单片机命名规则.....	91
5	ISP下载及典型应用线路图.....	92
5.1	STC8F系列ISP下载应用线路图.....	92
5.1.1	使用RS-232 转换器下载.....	92
5.1.2	使用PL2303-SA 下载.....	93
5.1.3	使用PL2303-GL下载.....	94
5.1.4	使用U8-Mini工具下载.....	95
5.1.5	使用U8W工具下载	96
5.1.6	USB直接ISP下载	97
5.2	STC8A系列ISP下载应用线路图.....	98
5.2.1	使用RS-232 转换下载 (使用高精度ADC)	98
5.2.2	使用RS-232 转换下载 (ADC一般应用)	99
5.2.3	使用PL2303-SA 下载	100
5.2.4	使用PL2303-GL下载.....	101
5.2.5	使用U8-Mini工具下载	102
5.2.6	使用U8W工具下载	103
5.2.7	USB直接ISP下载	104
6	时钟、复位与电源管理.....	105
6.1	系统时钟控制	105
6.2	STC8 系列内部IRC频率调整	108
6.3	STC15 系列内部IRC频率调整	110
6.4	系统复位	112
6.5	系统电源管理	114
6.6	范例程序	115
6.6.1	选择系统时钟源	115
6.6.2	主时钟分频输出	116
6.6.3	看门狗定时器应用	117
6.6.4	软复位实现自定义下载	118
6.6.5	低压检测	119

6.6.6	省电模式.....	120
6.6.7	使用INT0/INT1/INT2/INT3/INT4 中断唤醒MCU	121
6.6.8	使用T0/T1/T2/T3/T4 中断唤醒MCU	123
6.6.9	使用RxD/RxD2/RxD3/RxD4 中断唤醒MCU.....	127
6.6.10	使用LVD中断唤醒MCU	129
6.6.11	使用CCP0/CCP1/CCP2/CCP3 中断唤醒MCU.....	130
6.6.12	CMP中断唤醒MCU	133
6.6.13	使用LVD功能检测工作电压（电池电压）	134
7	存储器.....	138
7.1	程序存储器	138
7.2	数据存储器	139
7.2.1	内部RAM.....	139
7.2.2	内部扩展RAM.....	140
7.2.3	外部扩展RAM.....	141
7.3	存储器中的特殊参数	143
7.3.1	读取Bandgap电压值 (从ROM中读取)	144
7.3.2	读取Bandgap电压值 (从RAM中读取)	146
7.3.3	读取全球唯一ID号 (从ROM中读取)	148
7.3.4	读取全球唯一ID号 (从RAM中读取)	151
7.3.5	读取 32K掉电唤醒定时器的频率 (从ROM中读取).....	153
7.3.6	读取 32K掉电唤醒定时器的频率 (从RAM中读取).....	155
7.3.7	用户自定义内部IRC频率 (从ROM中读取).....	157
7.3.8	用户自定义内部IRC频率 (从RAM中读取).....	158
8	特殊功能寄存器.....	160
8.1	STC8A8K64S4A12 系列.....	160
8.2	STC8A4K64S2A12 系列.....	161
8.3	STC8F2K64S4 系列	162
8.4	STC8F2K64S2 系列	163
8.5	特殊功能寄存器列表	164
9	I/O口	169
9.1	I/O口相关寄存器.....	169
9.2	配置I/O口.....	173
9.3	I/O的结构图.....	174
9.3.1	准双向口（弱上拉）	174
9.3.2	推挽输出.....	174
9.3.3	高阻输入.....	174
9.3.4	开漏输出	175
9.4	特殊I/O口说明.....	176
9.4.1	P2.0/RSTCV	176
9.4.2	PWM相关I/O口	176
9.5	范例程序	177
9.5.1	端口模式设置.....	177
9.5.2	双向口读写操作	178
9.5.3	用STC系列MCU的IO口直接驱动段码LCD.....	179

10	指令系统.....	198
11	中断系统.....	202
11.1	STC8 系列中断源.....	202
11.1.1	STC8A8K64S4A12 系列中断源.....	203
11.1.2	STC8A4K64S2A12 系列中断源.....	203
11.1.3	STC8F2K64S4 系列中断源.....	203
11.1.4	STC8F2K64S2 系列中断源.....	204
11.2	STC8 中断结构图.....	205
11.3	STC8 系列中断列表.....	206
11.4	中断相关寄存器.....	209
11.4.1	中断使能寄存器（中断允许位）.....	210
11.4.2	中断请求寄存器（中断标志位）.....	213
11.4.3	中断优先级寄存器.....	216
11.5	范例程序.....	219
11.5.1	INT0 中断（上升沿和下降沿）.....	219
11.5.2	INT0 中断（下降沿）.....	220
11.5.3	INT1 中断（上升沿和下降沿）.....	220
11.5.4	INT1 中断（下降沿）.....	221
11.5.5	INT2 中断（下降沿）.....	222
11.5.6	INT3 中断（下降沿）.....	223
11.5.7	INT4 中断（下降沿）.....	224
11.5.8	定时器 0 中断.....	225
11.5.9	定时器 1 中断.....	226
11.5.10	定时器 2 中断.....	227
11.5.11	定时器 3 中断.....	228
11.5.12	定时器 4 中断.....	229
11.5.13	UART1 中断.....	230
11.5.14	UART2 中断.....	232
11.5.15	UART3 中断.....	234
11.5.16	UART4 中断.....	235
11.5.17	ADC 中断.....	237
11.5.18	LVD 中断.....	238
11.5.19	PCA 中断.....	239
11.5.20	SPI 中断.....	241
11.5.21	CMP 中断.....	242
11.5.22	PWM 中断.....	243
11.5.23	I2C 中断.....	246
12	定时器/计数器.....	249
12.1	定时器的相关寄存器.....	249
12.2	定时器 0/1.....	251
12.3	定时器 2.....	254
12.4	定时器 3/4.....	255
12.5	掉电唤醒定时器.....	257
12.6	范例程序.....	258

12.6.1	定时器 0 (模式 0—16 位自动重载)	258
12.6.2	定时器 0 (模式 1—16 位不自动重载)	259
12.6.3	定时器 0 (模式 2—8 位自动重载)	260
12.6.4	定时器 0 (模式 3—16 位自动重载不可屏蔽中断)	261
12.6.5	定时器 0 (外部计数—扩展T0 为外部下降沿中断)	262
12.6.6	定时器 0 (测量脉宽—INT0 高电平宽度)	263
12.6.7	定时器 0 (时钟分频输出)	264
12.6.8	定时器 1 (模式 0—16 位自动重载)	264
12.6.9	定时器 1 (模式 1—16 位不自动重载)	265
12.6.10	定时器 1 (模式 2—8 位自动重载)	266
12.6.11	定时器 1 (外部计数—扩展T1 为外部下降沿中断)	267
12.6.12	定时器 1 (测量脉宽—INT1 高电平宽度)	268
12.6.13	定时器 1 (时钟分频输出)	270
12.6.14	定时器 1 (模式 0) 做串口 1 波特率发生器	270
12.6.15	定时器 1 (模式 2) 做串口 1 波特率发生器	273
12.6.16	定时器 2 (16 位自动重载)	276
12.6.17	定时器 2 (外部计数—扩展T2 为外部下降沿中断)	277
12.6.18	定时器 2 (时钟分频输出)	279
12.6.19	定时器 2 做串口 1 波特率发生器	279
12.6.20	定时器 2 做串口 2 波特率发生器	282
12.6.21	定时器 2 做串口 3 波特率发生器	285
12.6.22	定时器 2 做串口 4 波特率发生器	288
12.6.23	定时器 3 (16 位自动重载)	291
12.6.24	定时器 3 (外部计数—扩展T3 为外部下降沿中断)	293
12.6.25	定时器 3 (时钟分频输出)	294
12.6.26	定时器 3 做串口 3 波特率发生器	295
12.6.27	定时器 4 (16 位自动重载)	298
12.6.28	定时器 4 (外部计数—扩展T4 为外部下降沿中断)	299
12.6.29	定时器 4 (时钟分频输出)	300
12.6.30	定时器 4 做串口 4 波特率发生器	301
13	串口通信.....	305
13.1	串口相关寄存器	305
13.2	串口 1	306
13.2.1	串口 1 模式 0	307
13.2.2	串口 1 模式 1	308
13.2.3	串口 1 模式 2	311
13.2.4	串口 1 模式 3	311
13.2.5	自动地址识别	312
13.3	串口 2	314
13.3.1	串口 2 模式 0	314
13.3.2	串口 2 模式 1	315
13.4	串口 3	317
13.4.1	串口 3 模式 0	317
13.4.2	串口 3 模式 1	318

13.5	串口 4	320
13.5.1	串口 4 模式 0	320
13.5.2	串口 4 模式 1	321
13.6	串口注意事项	323
13.7	范例程序	324
13.7.1	串口 1 使用定时器 2 做波特率发生器	324
13.7.2	串口 1 使用定时器 1 (模式 0) 做波特率发生器	326
13.7.3	串口 1 使用定时器 1 (模式 2) 做波特率发生器	329
13.7.4	串口 2 使用定时器 2 做波特率发生器	332
13.7.5	串口 3 使用定时器 2 做波特率发生器	335
13.7.6	串口 3 使用定时器 3 做波特率发生器	338
13.7.7	串口 4 使用定时器 2 做波特率发生器	341
13.7.8	串口 4 使用定时器 4 做波特率发生器	344
14	比较器, 掉电检测, 内部固定比较电压	348
14.1	比较器内部结构图	348
14.2	比较器相关的寄存器	349
14.3	范例程序	351
14.3.1	比较器的使用 (中断方式)	351
14.3.2	比较器的使用 (查询方式)	352
14.3.3	比较器作外部掉电检测	354
14.3.4	比较器检测工作电压 (电池电压)	356
15	IAP/EEPROM	360
15.1	EEPROM相关的寄存器	360
15.2	关于EEPROM编程和擦除等待时间的重要说明	362
15.3	EEPROM大小及地址	363
15.4	范例程序	366
15.4.1	EEPROM基本操作	366
15.4.2	使用MOVC读取EEPROM	369
15.4.3	使用串口送出EEPROM数据	372
16	ADC模数转换	376
16.1	ADC相关的寄存器	376
16.2	ADC典型应用线路图	378
16.2.1	高精度ADC应用	378
16.2.2	ADC一般应用 (对ADC精度要求不高的应用)	379
16.3	ADC线性参数	380
16.4	范例程序	381
16.4.1	ADC基本操作 (查询方式)	381
16.4.2	ADC基本操作 (中断方式)	382
16.4.3	格式化ADC转换结果	383
16.4.4	利用ADC第 16 通道测量外部电压或电池电压	385
17	PCA/CCP/PWM应用	388
17.1	PCA相关的寄存器	388
17.2	PCA工作模式	392
17.2.1	捕获模式	392

17.2.2	软件定时器模式.....	392
17.2.3	高速脉冲输出模式.....	393
17.2.4	PWM脉宽调制模式	393
17.3	范例程序.....	397
17.3.1	PCA输出PWM（6/7/8/10 位）	397
17.3.2	PCA捕获测量脉冲宽度	399
17.3.3	PCA实现 16 位软件定时	402
17.3.4	PCA输出高速脉冲	404
17.3.5	PCA扩展外部中断	407
18	增强型PWM.....	410
18.1	PWM相关的寄存器	410
18.2	范例程序	417
18.2.1	输出任意周期和任意占空比的波形	417
18.2.2	两路PWM实现互补对称带死区控制的波形.....	419
18.2.3	PWM实现渐变灯（呼吸灯）	423
19	同步串行外设接口SPI.....	428
19.1	SPI相关的寄存器	428
19.2	SPI通信方式	430
19.2.1	单主单从	430
19.2.2	互为主从	430
19.2.3	单主多从	431
19.3	配置SPI	432
19.4	数据模式	434
19.5	范例程序	435
19.5.1	SPI单主单从系统主机程序（中断方式）	435
19.5.2	SPI单主单从系统从机程序（中断方式）	436
19.5.3	SPI单主单从系统主机程序（查询方式）	437
19.5.4	SPI单主单从系统从机程序（查询方式）	439
19.5.5	SPI互为主从系统程序（中断方式）	440
19.5.6	SPI互为主从系统程序（查询方式）	442
20	I²C总线	444
20.1	I ² C相关的寄存器.....	444
20.2	I ² C主机模式.....	445
20.3	I ² C从机模式.....	449
20.4	范例程序	452
20.4.1	I ² C主机模式访问AT24C256（中断方式）	452
20.4.2	I ² C主机模式访问AT24C256（查询方式）	457
20.4.3	I ² C主机模式访问PCF8563.....	462
20.4.4	I ² C从机模式（中断方式）	466
20.4.5	I ² C从机模式（查询方式）	470
20.4.6	测试I ² C从机模式代码的主机代码	473
21	增强型双数据指针	478
21.1	范例程序	480
21.1.1	示例代码 1	480

21.1.2	示例代码 2	480
附录A	应用注意事项.....	482
A.1	关于STC8A系列、STC8F系列芯片问题总结	482
A.2	关于使用CLR指令关闭EA的重要说明	483
A.3	关于EEPROM编程和擦除等待时间的重要说明	484
A.4	STC8F2K64S4 系列应用注意事项	486
A.5	STC8F2K64S2 系列应用注意事项	489
A.6	STC8A8K64S4A12 系列应用注意事项	491
A.7	STC8A4K64S2A12 系列应用注意事项	495
A.8	使用外部晶振对STC8 系列进行仿真的注意事项	496
附录B	STC仿真器使用指南	497
附录C	STC-USB驱动程序安装说明	502
Windows XP安装方法	502	
Windows 7 (32 位) 安装方法	509	
Windows 7 (64 位) 安装方法	512	
Windows 8 (32 位) 安装方法	521	
Windows 8 (64 位) 安装方法	530	
Windows 8.1 (64 位) 安装方法	538	
Windows10 (64 位) 安装方法	546	
附录D	使用第三方MCU对STC8A/STC8F系列单片机进行ISP下载范例程序	565
附录E	电气特性.....	573
附录F	更新记录.....	575

1 概述

STC8 系列单片机是不需要外部晶振和外部复位的单片机，是以超强抗干扰/超低价/高速/低功耗为目标的 8051 单片机，在相同的工作频率下，STC8 系列单片机比传统的 8051 约快 12 倍(速度快 11.2~13.2 倍)，依次按顺序执行完全部的 111 条指令，STC8 系列单片机仅需 147 个时钟，而传统 8051 则需要 1944 个时钟。STC8 系列单片机是 STC 生产的单时钟/机器周期(1T)的单片机，是宽电压/高速/高可靠/低功耗/强抗静电/较强抗干扰的新一代 8051 单片机，超级加密。指令代码完全兼容传统 8051。

MCU 内部集成高精度 R/C 时钟($\pm 0.3\%$, 常温下 $+25^{\circ}\text{C}$), $-1.8\% \sim +0.8\%$ 温飘($-40^{\circ}\text{C} \sim +85^{\circ}\text{C}$) , $-1.0\% \sim +0.5\%$ 温飘($-20^{\circ}\text{C} \sim +65^{\circ}\text{C}$)。ISP 编程时 5MHz~30MHz 宽范围可设置，可彻底省掉外部昂贵的晶振和外部复位电路(内部已集成高可靠复位电路，ISP 编程时 4 级复位门槛电压可选)。

MCU 内部有 3 个可选时钟源：内部 24MHz 高精度 IRC 时钟（可适当调高或调低）、内部 32KHz 的低速 IRC、外部 4M~33M 晶振或外部时钟信号。用户代码中可自由选择时钟源，时钟源选定后可再经过 8-bit 的分频器分频后再将时钟信号提供给 CPU 和各个外设（如定时器、串口、SPI 等）。

MCU 提供两种低功耗模式：IDLE 模式和 STOP 模式。IDLE 模式下，MCU 停止给 CPU 提供时钟，CPU 无时钟，CPU 停止执行指令，但所有的外设仍处于工作状态，此时功耗约为 1.3mA (6MHz 工作频率)。STOP 模式即为主时钟停振模式，即传统的掉电模式/停电模式/停机模式，此时 CPU 和全部外设都停止工作，功耗可降低到 0.1uA 以下。IDLE 模式可以由外部中断 (INT0/INT1/INT2/INT3/INT4)、定时器中断(定时器 0/定时器 1/定时器 2/定时器 3/定时器 4)、串口中断(串口 1/串口 2/串口 3/串口 4)、PCA/CCP/PWM 中断、增强型 PWM、增强型 PWM 异常检测、ADC 模数转换中断、LVD 低压检测、SPI 中断、I2C 中断、比较器中断唤醒。STOP 模式可以由 INT0/INT1 端口上升沿/下降沿、INT2/INT3/INT4 端口下降沿、T0/T1/T2/T3/T4 端口下降沿、RxD/RxD_2/RxD_3/RxD_4 端口下降沿、RxD2/RxD2_2 端口下降沿、RxD2/RxD2_2 端口下降沿、RxD3/RxD3_2 端口下降沿、RxD4/RxD4_2 端口下降沿、CCP0/CCP0_2/CCP0_3/CCP0_4 端口下降沿、CCP1/CCP1_2/CCP1_3/CCP1_4 端口下降沿、CCP2/CCP2_2/CCP2_3/CCP2_4 端口下降沿、CCP3/CCP3_2/CCP3_3/CCP3_4 端口下降沿、LVD 低压检测以及掉电唤醒定时器（进入掉电模式前需要先使能掉电唤醒定时器）唤醒。

MCU 提供了丰富的数字外设 (4 个串口、5 个定时器、4 组 PCA、8 组增强型 PWM 以及 I²C、SPI) 接口与模拟外设 (速度高达 800K 即每秒 80 万次采样的 12 位*15 路超高速 ADC、比较器)，可满足广大用户的设计需求。

数字功能可使用程序在多个管脚之间进行切换。串口 1 可以组为单位 ([TxR/RxT]为一组) 在 [P3.0/P3.1]、[P3.6/P3.7]、[P1.6/P1.7]、[P4.3/P4.4] 这 4 组之间进行任意切换；串口 2 可以组为单位 ([TxD2/RxD2]为一组) 在 [P1.0/P1.1]、[P4.0/P4.2] 这 2 组之间进行任意切换；串口 3 可以组为单位 ([TxD3/RxD3]为一组) 在 [P0.0/P0.1]、[P5.0/P5.1] 这 2 组之间进行任意切换；串口 4 可以组为单位 ([TxD4/RxD4]为一组) 在 [P0.2/P0.3]、[P5.2/P5.3] 这 2 组之间进行任意切换；PCA 可以组为单位 ([ECI/CCP0/CCP1/CCP2/CCP3] 为一组) 在 [P1.2/P1.7/P1.6/P1.5/P1.4]、[P2.2/P2.3/P2.4/P2.5/P2.6]、[P7.4/P7.0/P7.1/P7.2/P7.3]、[P3.5/P3.3/P3.2/P3.1/P3.0] 这 4 组之间进行任意切换；SPI 可以组为单位 ([SS/MOSI/MISO/SCLK] 为一组) 在 [P1.2/P1.3/P1.4/P1.5]、[P2.2/P2.3/P2.4/P2.5]、[P7.4/P7.5/P7.6/P7.7]、[P3.5/P3.4/P3.3/P3.2] 这 4 组之间进行任意切换；I²C 可以组为单位 ([SCL/SDA] 为一组) 在 [P1.5/P1.4]、[P2.5/P2.4]、[P7.7/P7.6]、[P3.2/P3.3] 这 4 组之间进行任意切换；增强型 PWM 的每一路均可各自独立地在 3 个端口之间任意切换（详细切换介绍请参考第 3.3 章的功能脚切换）。

STC8 系列单片机内部集成了增强型的双数据指针。通过程序控制，可实现数据指针自动递增或递减功能以及两组数据指针的自动切换功能。

产品线	UART	定时器	ADC	增强型 PWM	高级 PWM	RTC	PCA	比较器	SPI	I2C	备注
STC8A8K64S4A12	●	●	●	●			●	●	●	●	
STC8A4K64S2A12	●	●	●	●			●	●	●	●	
STC8F2K64S4	●	●						●	●	●	
STC8F2K64S2	●	●						●	●	●	
STC8H1K64S2A10	●	●	●		●			●	●	●	
STC8H1K08S2A10	●	●	●		●			●	●	●	
STC8H1K08S2	●	●						●	●	●	
STC8H04A10	●	●	●					●	●	●	
STC8H04	●	●						●	●	●	

2 特性及价格

2.1 STC8A8K64S4A12 系列特性及价格

➤ 选型价格 (不需要外部晶振、不需要外部复位, 12 位 ADC, 15 通道)

单片机型号	工作电压 (V)	Flash 程序存储器 10 万次 字节	EEPROM 10 万次 字节	I/O 口最多数量	强大的双 DPTR 可增可减	大容量扩展 SRAM 字节	串口并可掉电唤醒	掉电唤醒专用定时器	PCA/CCP/PWM (可当外部中断并可掉电唤醒)	15 位增强型 PWM (带死区控制)	16 位高级 PWM 定时器 互补对称死区	定时器/计数器 (T0-T4 外部管脚也可掉电唤醒)	看门狗 复位定时器	内部高精准时钟 (24MHz 可调)	内部高压可靠复位 (可选复位门槛电压)	比较器 (可当一路 A/D, 可作外部掉电检测)	15 路高速 ADC (8 路 PWM 可当 8 路 D/A 使用)	内部低压检测中断并可掉电唤醒	外部高压检测中断并可掉电复位 (可选复位门槛电压)	可对外输出时钟及复位	可设置下次更新程序需口号令	程序加密后传输 (防拦截)	支持 RS485 下载	本身即可在线仿真	封装		2017 年新品供货信息	
STC8A8K16S4A12	2.0-5.5	16K	8K	2	48K	59	4	有	有	5	-	8	4	有	12 位	有	有	有	4 级	有	是	有	是	是	是	¥3.4	¥3.2	¥3.2
STC8A8K32S4A12	2.0-5.5	32K	8K	2	32K	59	4	有	有	5	-	8	4	有	12 位	有	有	有	4 级	有	是	有	是	是	是	¥3.6	¥3.3	¥3.3
STC8A8K60S4A12	2.0-5.5	60K	8K	2	4K	59	4	有	有	5	-	8	4	有	12 位	有	有	有	4 级	有	是	有	是	是	是	¥3.8	¥3.4	¥3.4
STC8A8K64S4A12	2.0-5.5	64K	8K	2	IAP	59	4	有	有	5	-	8	4	有	12 位	有	有	有	4 级	有	是	有	是	是	是	¥3.8	¥3.4	¥3.4

各个版本芯片的使用注意事项请参考“附录 A、应用注意事项”

➤ 内核

- ✓ 超高速 8051 内核 (1T), 比传统 8051 约快 12 倍以上
- ✓ 指令代码完全兼容传统 8051
- ✓ 22 个中断源, 4 级中断优先级
- ✓ 支持在线仿真

➤ 工作电压

- ✓ 2.0V~5.5V
- ✓ 内建 LDO

➤ 工作温度

- ✓ -40°C~85°C

➤ Flash 存储器

- ✓ 最大 64K 字节 FLASH 空间, 用于存储用户代码
- ✓ 支持用户配置 EEPROM 大小, 512 字节单页擦除, 擦写次数可达 10 万次以上
- ✓ 支持在系统编程方式 (ISP) 更新用户应用程序, 无需专用编程器
- ✓ 支持单芯片仿真, 无需专用仿真器, 理论断点个数无限制

➤ SRAM

- ✓ 128 字节内部直接访问 RAM (DATA)
- ✓ 128 字节内部间接访问 RAM (IDATA)
- ✓ 8192 字节内部扩展 RAM (内部 XDATA)
- ✓ 外部最大可扩展 64K 字节 RAM (外部 XDATA)

➤ 时钟控制

- ✓ 内部 24MHz 高精度 IRC (ISP 编程时可进行上下调整)
 - ✧ 误差±0.3% (常温下 25°C)
 - ✧ -1.8%~+0.8% 温漂 (全温度范围, -40°C~85°C)
 - ✧ -1.0%~+0.5% 温漂 (温度范围, -20°C~65°C)
- ✓ 内部 32KHz 低速 IRC (误差较大)
- ✓ 外部晶振 (4MHz~33MHz) 和外部时钟
 - 用户可自由选择上面的 3 种时钟源

➤ 复位

- ✓ 硬件复位
 - ✧ 上电复位
 - ✧ 复位脚复位 (高电平复位), 出厂时 P5.4 默认为 IO 口, ISP 下载时可将 P5.4 管脚设置为复位脚
 - ✧ 看门狗溢出复位
 - ✧ 低压检测复位, 提供 4 级低压检测电压: 2.2V、2.4V、2.7V、3.0V
- ✓ 软件复位
 - ✧ 软件方式写复位触发寄存器

➤ 中断

- ✓ 提供 22 个中断源: INT0、INT1、INT2、INT3、INT4、定时器 0、定时器 1、定时器 2、定时器 3、定时器 4、串口 1、串口 2、串口 3、串口 4、ADC 模数转换、LVD 低压检测、SPI、I²C、比较器、PCA/CCP/PWM、增强型 PWM、增强型 PWM 异常检测
- ✓ 提供 4 级中断优先级

➤ 数字外设

- ✓ 5 个 16 位定时器: 定时器 0、定时器 1、定时器 2、定时器 3、定时器 4, 其中定时器 0 的模式 3 具有 NMI (不可屏蔽中断) 功能, 定时器 0 和定时器 1 的模式 0 为 16 位自动重载模式
- ✓ 4 个高速串口: 串口 1、串口 2、串口 3、串口 4, 波特率时钟源最快可为 FOSC/4
- ✓ 4 组 16 位 PCA 模块: CCP0、CCP1、CCP2、CCP3, 可用于捕获、高速脉冲输出, 及 6/7/8/10 位的 PWM 输出
- ✓ 8 组 15 位增强型 PWM, 可实现带死区的控制信号, 并支持外部异常检测功能, 另外还有 4 组传统的 PCA/CCP/PWM 可作 PWM
- ✓ SPI: 支持主机模式和从机模式以及主机/从机自动切换
- ✓ I²C: 支持主机模式和从机模式

➤ 模拟外设

- ✓ 超高速 ADC, 支持 **12 位精度** 15 通道的模数转换, **速度最快可达 800K (即每秒可进行 80 万次模数转换)**
- ✓ 比较器, 一组比较器附近

➤ GPIO

- ✓ 最多可达 59 个 GPIO: P0.0~P0.7、P1.0~P1.7、P2.0~P2.7、P3.0~P3.7、P4.0~P4.4、P5.0~P5.5、P6.0~P6.7、P7.0~P7.7
- ✓ 所有的 GPIO 均支持如下 4 种模式: 准双向口模式、强推挽输出模式、开漏输出模式、高阻输入模式

➤ 封装

- ✓ LQFP64S、LQFP48、LQFP44、**PDIP40 (暂未生产)**

2.2 STC8A4K64S2A12 系列特性及价格

➤ 选型价格（不需要外部晶振、不需要外部复位，12位ADC，15通道）

2017 年新品供货信息	PDI P40	封装	本身即可在线仿真	支持 USB 直接下载	支持 RS485 下载	可设置下次更新程序需口令	程序加密后传输（防拦截）	可对外输出时钟及复位	内部高精准时钟（24MHz 可调）	内部高可靠复位（可选复位门槛电压）	看门狗 复位定时器	内部低压检测中断并可掉电唤醒	比较器（可当 1 路 A/D，可作外部掉电检测）	15 路高速 ADC(8 路 PWM 可当 8 路 D/A 使用)	掉电唤醒专用定时器	PCA/CCP/PWM（可当外部中断并可掉电唤醒）	15 位增强型 PWM（带死区控制）	16 位高级 PWM 定时器 互补对称死区	定时器/计数器（10-T4 外部管脚也可掉电唤醒）	I ² C	SPI	串口并可掉电唤醒	I/O 口最多数量	EEPROM 10 万次 字节	强大的双 DTR 可增可减	大容量扩展 SRAM 字节	Flash 程序存储器 10 万次 字节	工作电压 (V)	单片机型号
	LQFP44																												
STC8A4K16S2A12-0-5.5-16K	4K	2	48K	59	2	有	有	5	-	8	4	有	12 位	有	有	有	4 级	有	是	有	是	是	是	是	\$3.1	\$2.9	\$2.9		
STC8A4K32S2A12-0-5.5-32K	4K	2	32K	59	2	有	有	5	-	8	4	有	12 位	有	有	有	4 级	有	是	有	是	是	是	是	\$3.3	\$3.0	\$3.0		
STC8A4K60S2A12-0-5.5-60K	4K	2	4K	59	2	有	有	5	-	8	4	有	12 位	有	有	有	4 级	有	是	有	是	是	是	是	\$3.6	\$3.2	\$3.2		
STC8A4K64S2A12-0-5.5-64K	4K	2	IAP	59	2	有	有	5	-	8	4	有	12 位	有	有	有	4 级	有	是	有	是	是	是	是	\$3.6	\$3.2	\$3.2		

各个版本芯片的使用注意事项请参考“附录 A、应用注意事项”

➤ 内核

- ✓ 超高速 8051 内核 (1T)，比传统 8051 约快 12 倍以上
 - ✓ 指令代码完全兼容传统 8051
 - ✓ 20 个中断源，4 级中断优先级
 - ✓ 支持在线仿真

➤ 工作电压

- ✓ 2.0V~5.5V
 - ✓ 内建 LDO

➤ 工作温度

- ✓ -40 °C ~ 85 °C

➤ Flash 存储器

- ✓ 最大 64K 字节 FLASH 空间，用于存储用户代码
 - ✓ 支持用户配置 EEPROM 大小，512 字节单页擦除，擦写次数可达 10 万次以上
 - ✓ 支持在系统编程方式 (ISP) 更新用户应用程序，无需专用编程器
 - ✓ 支持单芯片仿真，无需专用仿真器，理论断点个数无限制

➤ SRAM

- ✓ 128 字节内部直接访问 RAM (DATA)
 - ✓ 128 字节内部间接访问 RAM (IDATA)
 - ✓ 4096 字节内部扩展 RAM (内部 XDATA)
 - ✓ 外部最大可扩展 64K 字节 RAM (外部 XDATA)

➤ 时钟控制

- ✓ 内部 24MHz 高精度 IRC (ISP 编程时可进行上下调整)
 - ◆ 误差 $\pm 0.3\%$ (常温下 25°C)

- ◆ -1.8%~+0.8%温漂（全温度范围, -40°C~85°C）
 - ◆ -1.0%~+0.5%温漂（温度范围, -20°C~65°C）
 - ✓ 内部 32KHz 低速 IRC（误差较大）
 - ✓ 外部晶振（4MHz~33MHz）和外部时钟
- 用户可自由选择上面的 3 种时钟源

➤ 复位

- ✓ 硬件复位
 - ◆ 上电复位
 - ◆ 复位脚复位（高电平复位），出厂时 P5.4 默认为 IO 口，ISP 下载时可将 P5.4 管脚设置为复位脚
 - ◆ 看门狗溢出复位
 - ◆ 低压检测复位，提供 4 级低压检测电压：2.2V、2.4V、2.7V、3.0V
- ✓ 软件复位
 - ◆ 软件方式写复位触发寄存器

➤ 中断

- ✓ 提供 20 个中断源：INT0、INT1、INT2、INT3、INT4、定时器 0、定时器 1、定时器 2、定时器 3、定时器 4、串口 1、串口 2、ADC 模数转换、LVD 低压检测、SPI、I²C、比较器、PCA/CCP/PWM、增强型 PWM、增强型 PWM 异常检测
- ✓ 提供 4 级中断优先级

➤ 数字外设

- ✓ 5 个 16 位定时器：定时器 0、定时器 1、定时器 2、定时器 3、定时器 4，其中定时器 0 的模式 3 具有 NMI（不可屏蔽中断）功能，定时器 0 和定时器 1 的模式 0 为 16 位自动重载模式
- ✓ 2 个高速串口：串口 1、串口 2，波特率时钟源最快可为 FOSC/4
- ✓ 4 组 16 位 PCA 模块：CCP0、CCP1、CCP2、CCP3，可用于捕获、高速脉冲输出，及 6/7/8/10 位的 PWM 输出
- ✓ 8 组 15 位增强型 PWM，可实现带死区的控制信号，并支持外部异常检测功能，另外还有 4 组传统的 PCA/CCP/PWM 可作 PWM
- ✓ SPI：支持主机模式和从机模式以及主机/从机自动切换
- ✓ I²C：支持主机模式和从机模式

➤ 模拟外设

- ✓ 超高速 ADC，支持 **12 位精度** 15 通道的模数转换，**速度最快可达 800K**（即每秒可进行 80 万次模数转换）
- ✓ 比较器，一组比较器附近

➤ GPIO

- ✓ 最多可达 59 个 GPIO：P0.0~P0.7、P1.0~P1.7、P2.0~P2.7、P3.0~P3.7、P4.0~P4.4、P5.0~P5.5、P6.0~P6.7、P7.0~P7.7
- ✓ 所有的 GPIO 均支持如下 4 种模式：准双向口模式、强推挽输出模式、开漏输出模式、高阻输入模式

➤ 封装

- ✓ LQFP64S、LQFP48、LQFP44、**PDIP40（暂未生产）**

2.3 STC8F2K64S4 系列特性及价格

➤ 选型价格（不需要外部晶振、不需要外部复位）

各个版本芯片的使用注意事项请参考“附录 A、应用注意事项”

➤ 内核

- ✓ 超高速 8051 内核 (1T)，比传统 8051 约快 12 倍以上
 - ✓ 指令代码完全兼容传统 8051
 - ✓ 18 个中断源，4 级中断优先级
 - ✓ 支持在线仿真

➤ 工作电压

- ✓ 2.0V~5.5V
 - ✓ 内建 LDO

➤ 工作温度

- ✓ -40 °C ~ 85 °C

➤ Flash 存储器

- ✓ 最大 64K 字节 FLASH 空间，用于存储用户代码
 - ✓ 支持用户配置 EEPROM 大小，512 字节单页擦除，擦写次数可达 10 万次以上
 - ✓ 支持在系统编程方式 (ISP) 更新应用程序，无需专用编程器
 - ✓ 支持单芯片仿真，无需专用仿真器，理论断点个数无限制

➤ SRAM

- ✓ 128 字节内部直接访问 RAM (DATA)
 - ✓ 128 字节内部间接访问 RAM (IDATA)
 - ✓ 2048 字节内部扩展 RAM (内部 XDATA)
 - ✓ 外部最大可扩展 64K 字节 RAM (外部 XDATA)

➤ 时钟控制

- ✓ 内部 24MHz 高精度 IRC (ISP 编程时可进行上下调整)
 ◆ 误差±0.3% (常温下 25°C)

- ◆ -1.8%~+0.8%温漂（全温度范围, -40°C~85°C）
- ◆ -1.0%~+0.5%温漂（温度范围, -20°C~65°C）
- ✓ 内部 32KHz 低速 IRC（误差较大）
- ✓ 外部晶振（4MHz~33MHz）和外部时钟输入
- 用户可自由选择上面的 3 种时钟源

➤ 复位

- ✓ 硬件复位
 - ◆ 上电复位
 - ◆ 复位脚复位（高电平复位），出厂时 P5.4 默认为 IO 口，ISP 下载时可将 P5.4 管脚设置为复位脚
 - ◆ 看门狗溢出复位
 - ◆ 低压检测复位，提供 4 级低压检测电压：2.2V、2.4V、2.7V、3.0V
- ✓ 软件复位
 - ◆ 软件方式写复位触发寄存器

➤ 中断

- ✓ 提供 18 个中断源：INT0、INT1、INT2、INT3、INT4、定时器 0、定时器 1、定时器 2、定时器 3、定时器 4、串口 1、串口 2、串口 3、串口 4、LVD 低压检测、**PCA/CCP/PWM**、SPI、I²C、比较器
- ✓ 提供 4 级中断优先级

➤ 数字外设

- ✓ 5 个 16 位定时器：定时器 0、定时器 1、定时器 2、定时器 3、定时器 4，其中定时器 0 的模式 3 具有 NMI（不可屏蔽中断）功能，定时器 0 和定时器 1 的模式 0 为 16 位自动重载模式
- ✓ 4 个高速串口：串口 1、串口 2、串口 3、串口 4，波特率时钟源最快可为 FOSC/4
- ~~✓ 4 组 16 位 PCA 模块：CCP0、CCP1、CCP2、CCP3，可用于捕获、高速脉冲输出，及 6/7/8/10 位的 PWM 输出（A 版和 B 版由此功能，C 版芯片无此功能）~~
- ✓ SPI：支持主机模式和从机模式以及主机/从机自动切换
- ✓ I²C：支持主机模式和从机模式

➤ 模拟外设

- ✓ 比较器

➤ GPIO

- ✓ 最多可达 42 个 GPIO：P0.0~P0.7、P1.0~P1.7、P2.0~P2.7、P3.0~P3.7、P4.0~P4.7、P5.4~P5.5
- ✓ 所有的 GPIO 均支持如下 4 种模式：准双向口模式、强推挽输出模式、开漏输出模式、高阻输入模式

➤ 封装

- ✓ LQFP44、LQFP32、PDIP40

2.4 STC8F2K64S2 系列特性及价格

➤ 选型价格 (不需要外部晶振、不需要外部复位)

单片机型号	2017年新品供货信息																			
	封装						PDI40													
SOP8			SOP16			TSSOP20			QFN32<4x4mm>			大量供货								
LQFP44			LQFP32			LQFP44			LQFP44											
本身即可在线仿真						支持USB直接下载														
可设置下次更新新程序需口号令						支持RS485下载														
程序加密后传输(防拦截)						可对外输出时钟及复位														
内部高精准时钟(24MHz可调)						内部高可靠复位(可选复位门槛电压)														
看门狗 复位定时器						内部低压检测中断并可掉电唤醒														
比较器(可当1路A/D,可作外部掉电检测)						15路高速ADC(8路PWM同时8路D/A使用)														
掉电唤醒专用定时器						15位增强型PWM(带死区控制)														
16位高级PWM定时器互补对称死区						PCA/CCP/PWM(可当外部中断并可掉电唤醒)														
定时器计数器(10-T4外部管脚也可掉电唤醒)						I ² C														
SPI						串口并可掉电唤醒														
I/O口最多数量						EEPROM 10万次字节														
强大的双DPIR可增可减						大容量扩展SRAM字节														
Flash程序存储器 10万次字节						工作电压(V)														

各个版本芯片的使用注意事项请参考“附录 A、应用注意事项”

➤ 内核

- ✓ 超高速 8051 内核 (1T)，比传统 8051 约快 12 倍以上
 - ✓ 指令代码完全兼容传统 8051
 - ✓ 16 个中断源，4 级中断优先级
 - ✓ 支持在线仿真

➤ 工作电压

- ✓ 2.0V~5.5V
 - ✓ 内建LDO

➤ 工作溫度

- ✓ -40 °C ~ 85 °C

➤ Flash 存储器

- ✓ 最大 64K 字节 FLASH 空间，用于存储用户代码
 - ✓ 支持用户配置 EEPROM 大小，512 字节单页擦除，擦写次数可达 10 万次以上
 - ✓ 支持在系统编程方式 (ISP) 更新应用程序，无需专用编程器
 - ✓ 支持单芯片仿真，无需专用仿真器，理论断点个数无限制

➤ SRAM

- ✓ 128 字节内部直接访问 RAM (DATA)
 - ✓ 128 字节内部间接访问 RAM (IDATA)
 - ✓ 2048 字节内部扩展 RAM (内部 XDATA)
 - ✓ 外部最大可扩展 64K 字节 RAM (外部 XDATA)

➤ 时钟控制

- ✓ 内部 24MHz 高精度 IRC (ISP 编程时可进行上下调整)

- ◆ 误差±0.3%（常温下 25°C）
- ◆ -1.8%~+0.8%温漂（全温度范围，-40°C~85°C）
- ◆ -1.0%~+0.5%温漂（温度范围，-20°C~65°C）
- ✓ 内部 32KHz 低速 IRC（误差较大）
- ✓ 外部晶振（4MHz~33MHz）和外部时钟输入
 用户可自由选择上面的 3 种时钟源

➤ 复位

- ✓ 硬件复位
 - ◆ 上电复位
 - ◆ 复位脚复位（高电平复位），出厂时 P5.4 默认为 IO 口，ISP 下载时可将 P5.4 管脚设置为复位脚
 - ◆ 看门狗溢出复位
 - ◆ 低压检测复位，提供 4 级低压检测电压：2.2V、2.4V、2.7V、3.0V
- ✓ 软件复位
 - ◆ 软件方式写复位触发寄存器

➤ 中断

- ✓ 提供 16 个中断源：INT0、INT1、INT2、INT3、INT4、定时器 0、定时器 1、定时器 2、定时器 3、定时器 4、串口 1、串口 2、LVD 低压检测、SPI、I²C、比较器
- ✓ 提供 4 级中断优先级

➤ 数字外设

- ✓ 5 个 16 位定时器：定时器 0、定时器 1、定时器 2、定时器 3、定时器 4，其中定时器 0 的模式 3 具有 NMI（不可屏蔽中断）功能，定时器 0 和定时器 1 的模式 0 为 16 位自动重载模式
- ✓ 2 个高速串口：串口 1、串口 2，波特率时钟源最快可为 FOSC/4
- ✓ SPI：支持主机模式和从机模式以及主机/从机自动切换
- ✓ I²C：支持主机模式和从机模式

➤ 模拟外设

- ✓ 比较器

➤ GPIO

- ✓ 最多可达 42 个 GPIO：P0.0~P0.7、P1.0~P1.7、P2.0~P2.7、P3.0~P3.7、P4.0~P4.7、P5.4~P5.5
- ✓ 所有的 GPIO 均支持如下 4 种模式：准双向口模式、强推挽输出模式、开漏输出模式、高阻输入模式

➤ 封装

- ✓ LQFP44、LQFP32、PDIP40、TSSOP20、SOP16

2.5 STC8F1K08S2 系列特性及价格

➤ 选型价格 (不需要外部复位)

单片机型号	工作电压 (V)	Flash 程序存储器 10 万次 字节	大容量扩展 SRAM 字节	强大的双 DPIR 可增可减	EEPROM 10 万次 字节	IO 口最多数量	串口并可掉电唤醒	I ² C	SPI	掉电唤醒专用定时器	PCA/CCP/PWM (可当外部中断并可掉电唤醒)	15 位增强型 PWM (带死区控制)	16 位高级 PWM 定时器 互补对称死区	定时器计数器 (T0-T4 外部管脚也可掉电唤醒)	看门狗	复位定时器	内部高可靠复位 (可选复位门槛电压)	内部高精准时钟 (24MHz 可调)	可对外输出时钟及复位	本身即可在线仿真	2018 年新品供货信息		2018 年新品供货信息				
																					SOP8	SOP16					
																					QFN20<3mm*3mm>	TSSOP20					
																					本身即可在线仿真	支持 USB 直接下载	可设置下次更新程序需口令	程序加密后传输 (防拦截)			
STC8F1K08S2	2.0-5.5	8K	1.2K	2	3K	18	2	有	有	3	-	-	-	有	4 级	有	是	有	是	是	是	是	是	是	是	-	大量供货
STC8F1K08	2.0-5.5	8K	1.2K	2	3K	6	1	有	有	3	-	-	-	有	-	有	4 级	有	是	有	是	是	是	是	-	?	

➤ 内核

- ✓ 超高速 8051 内核 (1T)，比传统 8051 约快 12 倍以上
- ✓ 指令代码完全兼容传统 8051
- ✓ 14 个中断源，4 级中断优先级
- ✓ 支持在线仿真

➤ 工作电压

- ✓ 2.0V~5.5V
- ✓ 内建 LDO

➤ 工作温度

- ✓ -40°C~85°C

➤ Flash 存储器

- ✓ 最大 11K 字节 FLASH 空间，用于存储用户代码
- ✓ 512 字节单页擦除，擦写次数可达 10 万次以上
- ✓ 支持在系统编程方式 (ISP) 更新应用程序，无需专用编程器
- ✓ 支持单芯片仿真，无需专用仿真器，理论断点个数无限制

➤ SRAM

- ✓ 128 字节内部直接访问 RAM (DATA)
- ✓ 128 字节内部间接访问 RAM (IDATA)
- ✓ 1024 字节内部扩展 RAM (内部 XDATA)

➤ 时钟控制

- ✓ 内部 24MHz 高精度 IRC (ISP 编程时可进行上下调整)
 - ◆ 误差±0.3% (常温下 25°C)
 - ◆ -1.8%~+0.8% 温漂 (全温度范围, -40°C~85°C)
 - ◆ -1.0%~+0.5% 温漂 (温度范围, -20°C~65°C)
- ✓ 内部 32KHz 低速 IRC (误差较大)
- ✓ 注意：此型号不可使用外部晶振作为系统时钟源

➤ 复位

- ✓ 硬件复位
 - ✧ 上电复位
 - ✧ 复位脚复位（高电平复位），出厂时 P5.4 默认为 IO 口，ISP 下载时可将 P5.4 管脚设置为复位脚
 - ✧ 看门狗溢出复位
 - ✧ 低压检测复位，提供 4 级低压检测电压：2.2V、2.4V、2.7V、3.0V
- ✓ 软件复位
 - ✧ 软件方式写复位触发寄存器

➤ 中断

- ✓ 提供 14 个中断源：INT0、INT1、INT2、INT3、INT4、定时器 0、定时器 1、定时器 2、串口 1、串口 2、LVD 低压检测、SPI、I²C、比较器
- ✓ 提供 4 级中断优先级

➤ 数字外设

- ✓ 3 个 16 位定时器：定时器 0、定时器 1、定时器 2，其中定时器 0 的模式 3 具有 NMI（不可屏蔽中断）功能，定时器 0 和定时器 1 的模式 0 为 16 位自动重载模式
- ✓ 2 个高速串口：串口 1、串口 2，波特率时钟源最快可为 FOSC/4
- ✓ SPI：支持主机模式和从机模式以及主机/从机自动切换
- ✓ I²C：支持主机模式和从机模式

➤ 模拟外设

- ✓ 比较器

➤ GPIO

- ✓ 最多可达 18 个 GPIO：P1.0~P1.7、P3.0~P3.7、P5.4~P5.5
- ✓ 所有的 GPIO 均支持如下 4 种模式：准双向口模式、强推挽输出模式、开漏输出模式、高阻输入模式

➤ 封装

- ✓ TSSOP20、SOP16、SOP8、QFN20

2.6 STC8C1K12 系列特性及价格

➤ 选型价格 (不需要外部晶振、不需要外部复位)

➤ 内核

- ✓ 超高速 8051 内核 (1T)，比传统 8051 约快 12 倍以上
 - ✓ 指令代码完全兼容传统 8051
 - ✓ 16 个中断源，4 级中断优先级
 - ✓ 支持在线仿真

➤ 工作电压

- ✓ 1.7V~5.5V
 - ✓ 内建LDO

➤ 工作溫度

- ✓ -40°C ~ 85°C

➤ Flash 存储器

- ✓ 最大 12K 字节 FLASH 空间，用于存储用户代码
 - ✓ 512 字节单页擦除，擦写次数可达 10 万次以上
 - ✓ 支持在系统编程方式（ISP）更新应用程序，无需专用编程器
 - ✓ 支持单芯片仿真，无需专用仿真器，理论断点个数无限制

➤ SRAM

- ✓ 128 字节内部直接访问 RAM (DATA)
 - ✓ 128 字节内部间接访问 RAM (IDATA)
 - ✓ 1024 字节内部扩展 RAM (内部 XDATA)

➤ 时钟控制

- ✓ 内部 20M/35MHz 高精度 IRC (ISP 编程时可进行上下调整)
 - ◆ 误差±0.3% (常温下 25°C)
 - ◆ -1.8%~+0.8% 温漂 (全温度范围, -40°C~85°C)
 - ◆ -1.0%~+0.5% 温漂 (温度范围, -20°C~65°C)
 - ✓ 内部 32KHz 低速 IRC (误差较大)

➤ 复位

- ✓ 硬件复位
 - ✧ 上电复位
 - ✧ 复位脚复位（高电平复位），出厂时 P5.4 默认为 IO 口，ISP 下载时可将 P5.4 管脚设置为复位脚
 - ✧ 看门狗溢出复位
 - ✧ 低压检测复位，提供 4 级低压检测电压：2.2V、2.4V、2.7V、3.0V
- ✓ 软件复位
 - ✧ 软件方式写复位触发寄存器

➤ 中断

- ✓ 提供 14 个中断源：INT0、INT1、INT2、INT3、INT4、定时器 0、定时器 1、定时器 2、串口 1、串口 2、LVD 低压检测、ADC、CCP、SPI、I²C、比较器
- ✓ 提供 4 级中断优先级

➤ 数字外设

- ✓ 3 个 16 位定时器：定时器 0、定时器 1、定时器 2，其中定时器 0 的模式 3 具有 NMI（不可屏蔽中断）功能，定时器 0 和定时器 1 的模式 0 为 16 位自动重载模式
- ✓ 2 个高速串口：串口 1、串口 2，波特率时钟源最快可为 FOSC/4
- ✓ SPI：支持主机模式和从机模式以及主机/从机自动切换
- ✓ 3 组 16 位 PCA 模块：CCP0、CCP1、CCP2，可用于捕获、高速脉冲输出，及 6/7/8/10 位的 PWM 输出
- ✓ I²C：支持主机模式和从机模式

➤ 模拟外设

- ✓ 超高速 ADC，支持 **10 位精度** 15 通道的模数转换，**速度最快可达 800K**（即每秒可进行 80 万次模数转换）
- ✓ 比较器

➤ GPIO

- ✓ 最多可达 18 个 GPIO：P1.0~P1.7、P3.0~P3.7、P5.4~P5.5
- ✓ 所有的 GPIO 均支持如下 4 种模式：准双向口模式、强推挽输出模式、开漏输出模式、高阻输入模式

➤ 封装

- ✓ TSSOP20、SOP16

2.7 STC8P1K08S2A10 系列提前通告

➤ 选型价格 (不需要外部晶振、不需要外部复位)

2018年新品供货信息		SOP16		TSSOP20		8月送样	
封装	本身即可在线仿真	支持USB直接下载	支持RS485下载	可设置下次更新程序需口令	程序加密后传输（防拦截）	可对外输出时钟及复位	内部高精准时钟（3520MHz 可调）
看门狗	复位定时器	内部低压检测中断并可掉电唤醒	比较器（可当1路A/D，可作外部掉电检测）	9路高速ADC	掉电唤醒专用定时器	内部高可靠复位（可选复位门槛电压）	内部高可靠复位（可选复位门槛电压）
15位增强型PWM（带死区控制）	16位高级PWM定时器互补对称死区	定时器计数器(T0-T4外部管脚也可掉电唤醒)	1IC	SPI	串口并可掉电唤醒	I/O口最多数量	EEPROM 10万次字节
大容量扩展SRAM字节	强大的双DPIR可增可减	Flash程序存储器10万次字节	工作电压(V)	单片机型号	本身即可在线仿真	支持USB直接下载	支持RS485下载

2.8 STC8P1K16S2A10 系列提前通告

➤ 选型价格（不需要外部晶振、不需要外部复位）

2.9 STC8P2K32S4A12 系列提前通告

➤ 选型价格 (不需要外部晶振、不需要外部复位)

2018年新品供货信息										8月送样	
封装					LQFP48						
本身即可在线仿真					支持USB直接下载						
支持RS485下载					可设置下次更新程序需口号令						
程序加密后传输（防拦截）					看门狗复位定时器						
可对外输出时钟及复位					内部高精准时钟（35/20MHz可调）					有	
内部高可靠复位（可选复位门槛电压）					内部低压检测中断并可掉电唤醒					有	
比较器（可当1路A/D，可作外部掉电检测）					15路高速ADC					有	
掉电唤醒专用定时器					PCA/CCP/PWM（可当外部中断并可掉电唤醒）					有	
15位增强型PWM（带死区控制）					16位高级PWM定时器互补对称死区					有	
定时器计数器(T0-T4外部管脚也可掉电唤醒)					I _C						
SPI					串口并可掉电唤醒						
I/O口最多数量					强大的双DPIR可增可减						
大容量扩展SRAM字节					Flash程序存储器10万次字节						
工作电压(V)					STC8P2K32S4A12	1.7-5.5	32K	2.2K	2	32K	45
单片机型号											

2.10 STC8H04A10 系列提前通告

➤ 选型价格 (不需要外部晶振、不需要外部复位)

单片机型号	2018 年新品供货信息	
	封装	SOP8
	本身即可在线仿真	
	支持 USB 直接下载	
	可设置下次更新程序需口令	
	程序加密后传输(防拦截)	
	可对外输出时钟及复位	
	内部高精准时钟 (24MHz 可调)	
	内部高可靠复位 (可选复位门槛电压)	
	看门狗 复位定时器	
	内部低压检测中断并可掉电唤醒	
	比较器 (可当 1 路 A/D, 可作外部掉电检测)	
	15 路高速 ADC (8 路 PWM 可当 8 路 D/A 使用)	
	掉电唤醒专用定时器	
	PCA/CCP/PWM (可当外部中断并可掉电唤醒)	
	15 位增强型 PWM (带死区控制)	
	16 位高级 PWM 定时器 互补对称死区	
	定时器/计数器 (T0-T4 外部管脚也可掉电唤醒)	
	I ² C	
	SPI	
	IO □最多数量	
	EEROM 10 万次 字节	
	强大的双 DPTIR 可增可减	
	大容量扩展 SRAM 字节	
	Flash 程序存储器 10 万次 字节	
	工作电压 (V)	

STC8H04A10 1.7-5.5 4K 256 1 3K 6 1 有 有 3 - - 有 10 位 7 月 送样

2.11 STC8H04 系列提前通告

➤ 选型价格 (不需要外部晶振、不需要外部复位)

单片机型号	2018 年新品供货信息	
	封装	SOP8
	本身即可在线仿真	
	支持 USB 直接下载	
	可设置下次更新程序需口令	
	程序加密后传输(防拦截)	
	可对外输出时钟及复位	
	内部高精准时钟 (24MHz 可调)	
	内部高可靠复位 (可选复位门槛电压)	
	看门狗 复位定时器	
	内部低压检测中断并可掉电唤醒	
	比较器 (可当 1 路 A/D, 可作外部掉电检测)	
	15 路高速 ADC (8 路 PWM 可当 8 路 D/A 使用)	
	掉电唤醒专用定时器	
	PCA/CCP/PWM (可当外部中断并可掉电唤醒)	
	15 位增强型 PWM (带死区控制)	
	16 位高级 PWM 定时器 互补对称死区	
	定时器/计数器 (T0-T4 外部管脚也可掉电唤醒)	
	I ² C	
	SPI	
	IO □最多数量	
	EEROM 10 万次 字节	
	强大的双 DPTIR 可增可减	
	大容量扩展 SRAM 字节	
	Flash 程序存储器 10 万次 字节	
	工作电压 (V)	

单片机型号

STC8H04

工作电压 (V)

Flash 程序存储器

10 万次 字节

EEROM 10 万次 字节

强大的双 DPTIR 可增可减

大容量扩展 SRAM 字节

IO □最多数量

I²C

串口并可掉电唤醒

SPI

掉电唤醒专用定时器

15 位增强型 PWM (带死区控制)

16 位高级 PWM 定时器 互补对称死区

定时器/计数器 (T0-T4 外部管脚也可掉电唤醒)

PCA/CCP/PWM (可当外部中断并可掉电唤醒)

15 位增强型 PWM (带死区控制)

16 位高级 PWM 定时器 互补对称死区

定时器/计数器 (T0-T4 外部管脚也可掉电唤醒)

掉电唤醒专用定时器

15 位增强型 PWM (带死区控制)

16 位高级 PWM 定时器 互补对称死区

定时器/计数器 (T0-T4 外部管脚也可掉电唤醒)

掉电唤醒专用定时器

15 位增强型 PWM (带死区控制)

16 位高级 PWM 定时器 互补对称死区

定时器/计数器 (T0-T4 外部管脚也可掉电唤醒)

掉电唤醒专用定时器

15 位增强型 PWM (带死区控制)

16 位高级 PWM 定时器 互补对称死区

定时器/计数器 (T0-T4 外部管脚也可掉电唤醒)

掉电唤醒专用定时器

15 位增强型 PWM (带死区控制)

16 位高级 PWM 定时器 互补对称死区

定时器/计数器 (T0-T4 外部管脚也可掉电唤醒)

掉电唤醒专用定时器

15 位增强型 PWM (带死区控制)

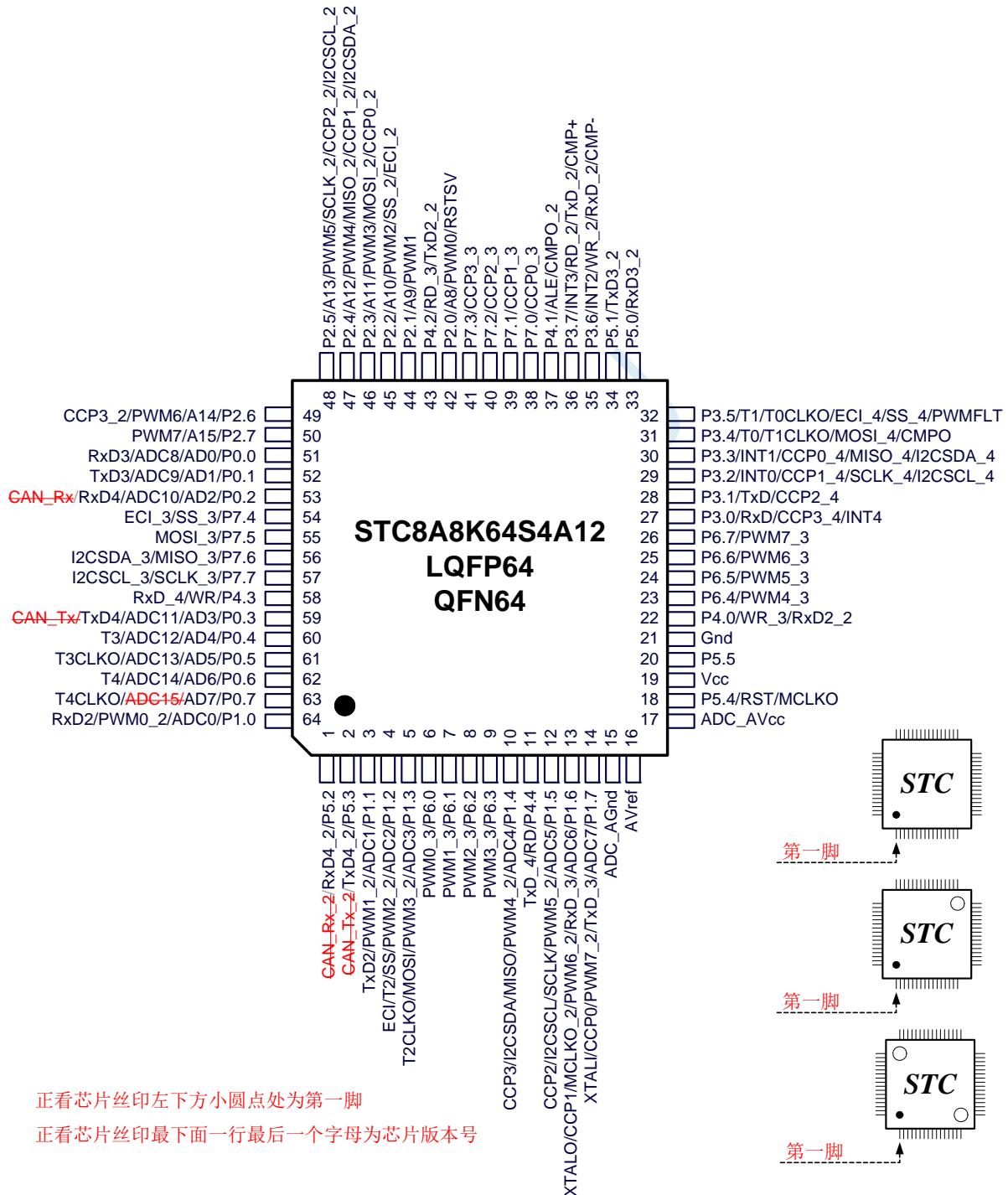
16 位高级 PWM 定时器 互补对称死区

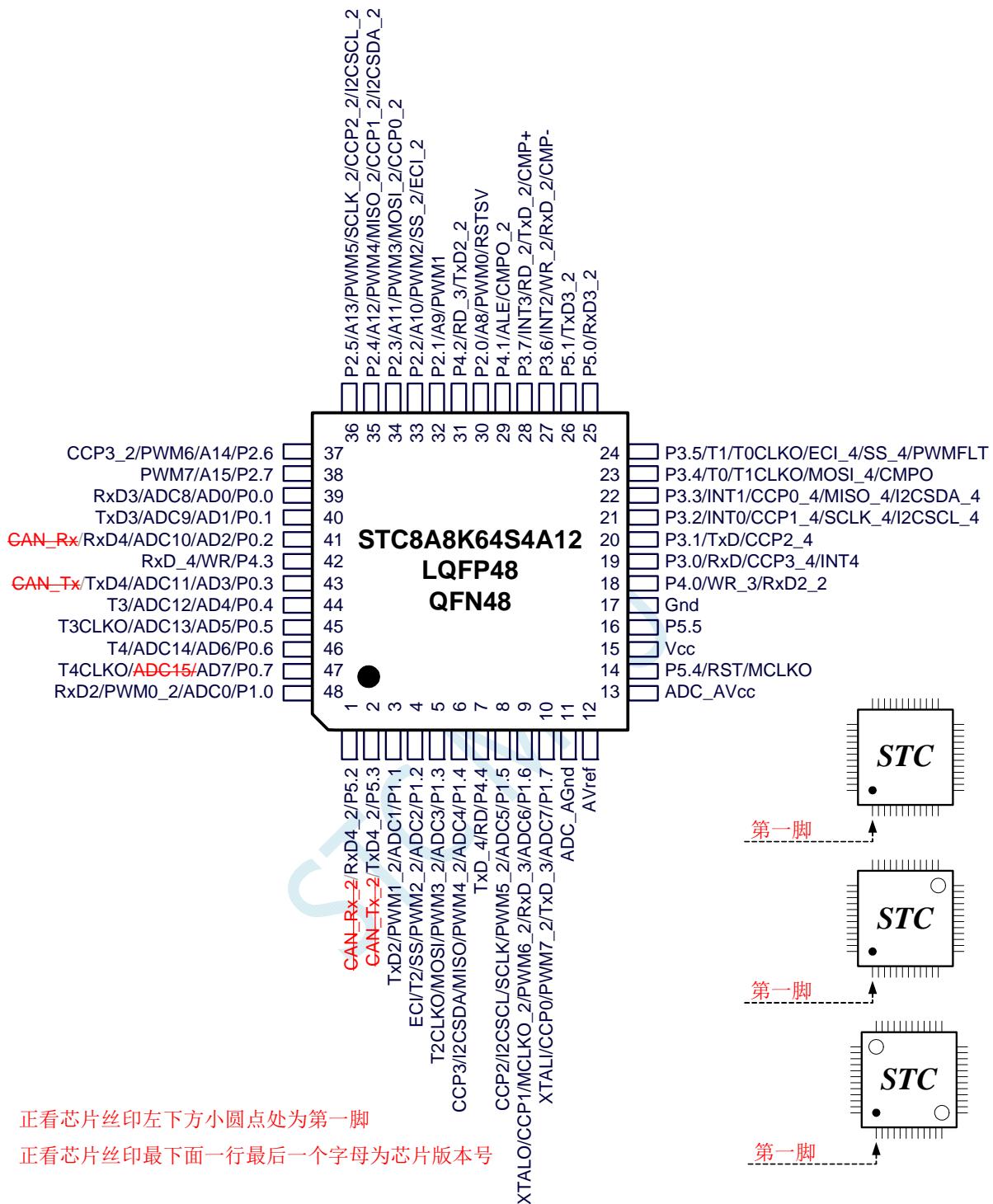
定时器/计数器 (T0-T4 外部管脚也可掉电唤醒)

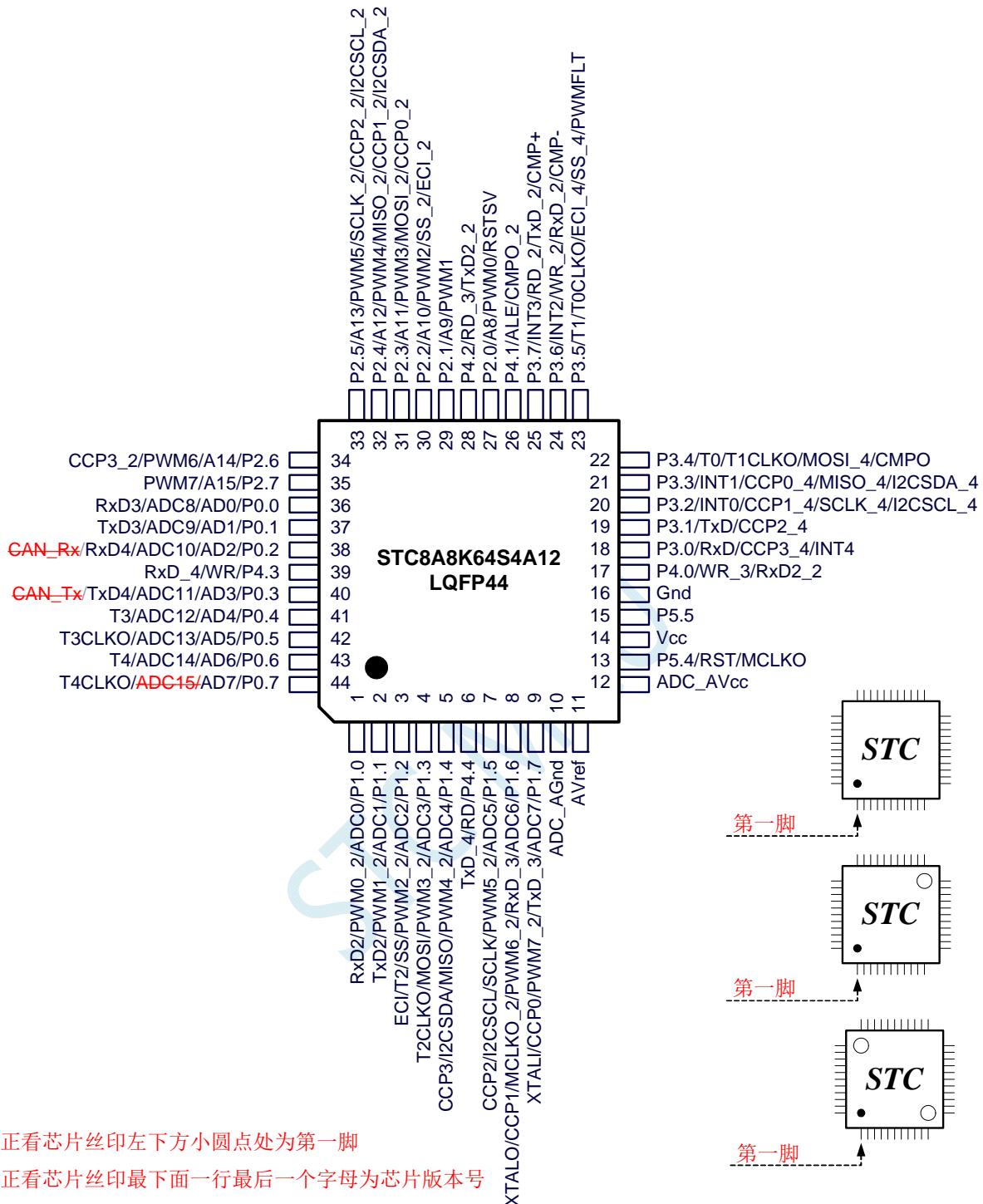
3 管脚及说明

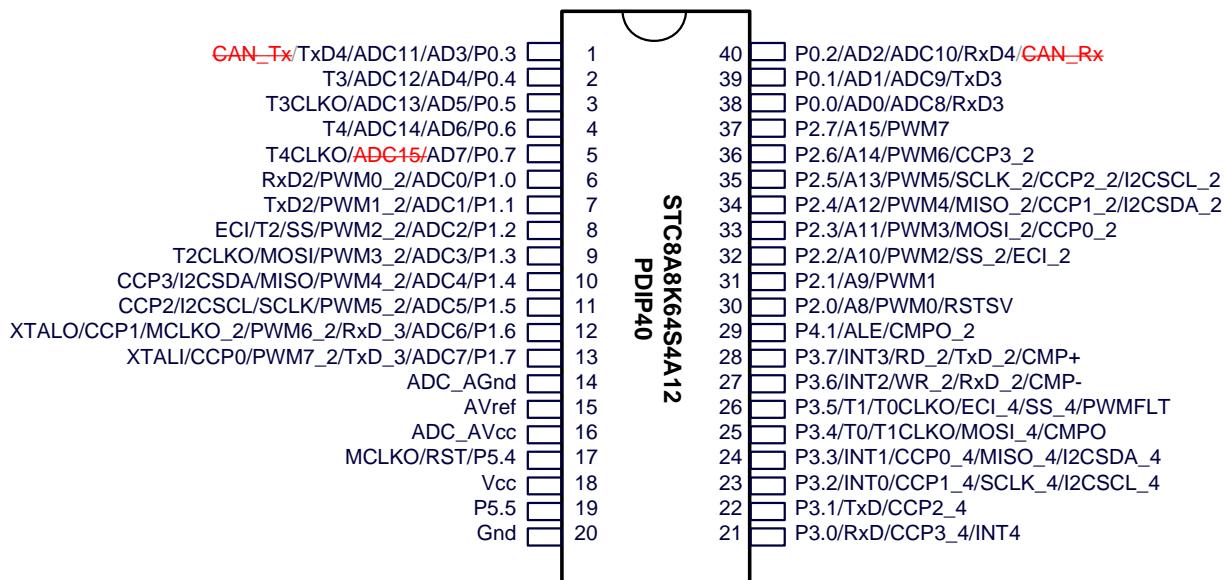
3.1 管脚图

3.1.1 STC8A8K64S4A12 系列管脚图

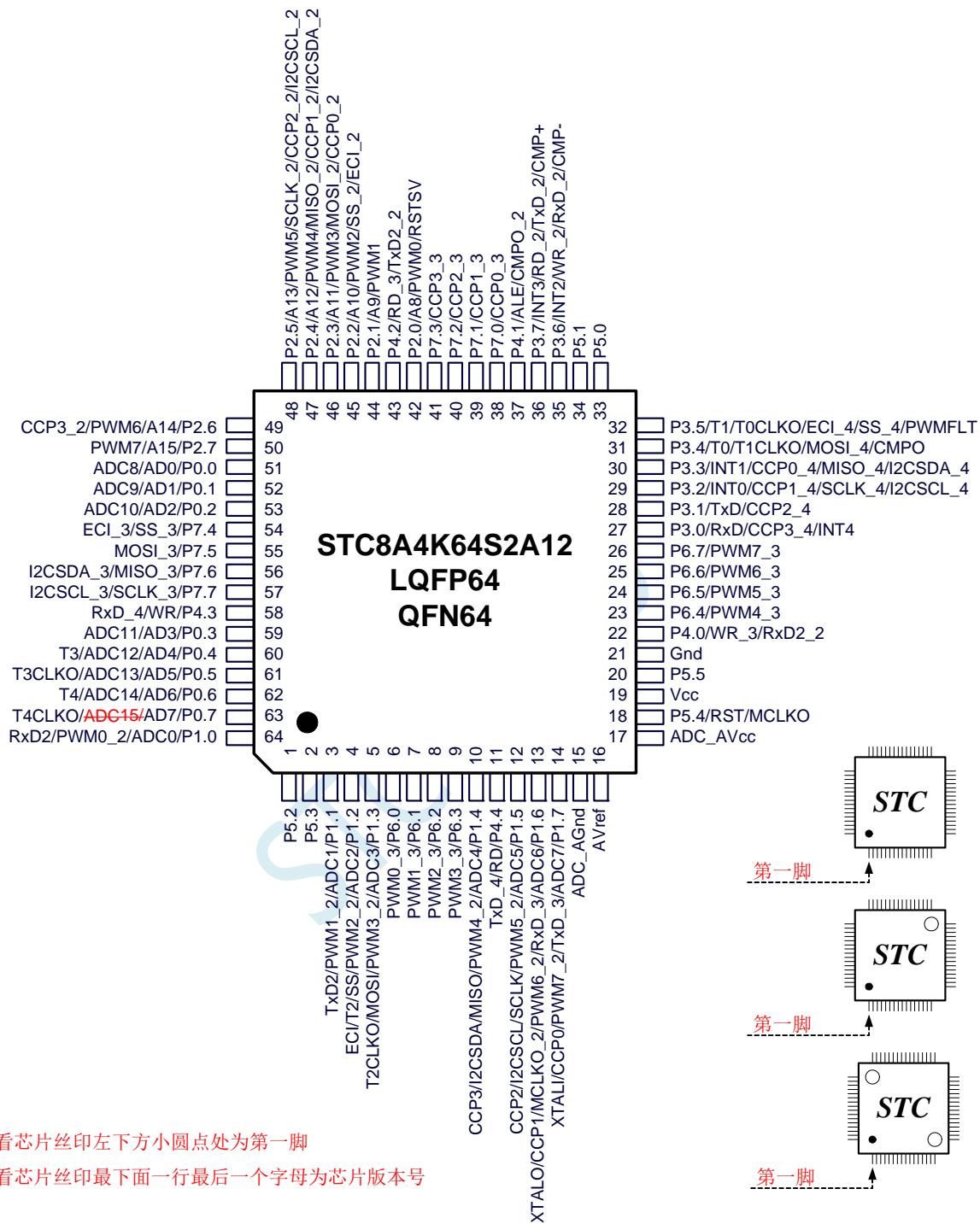


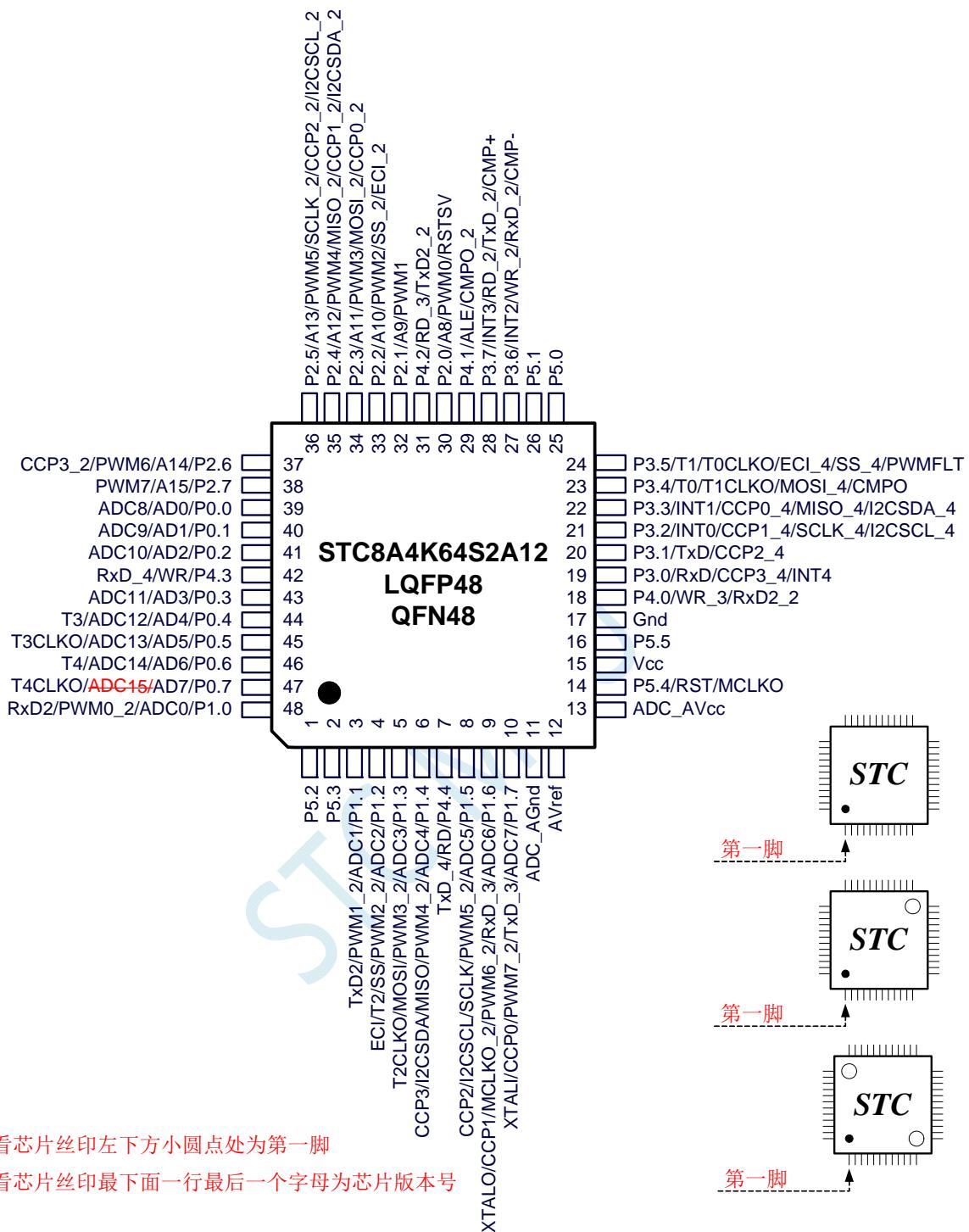


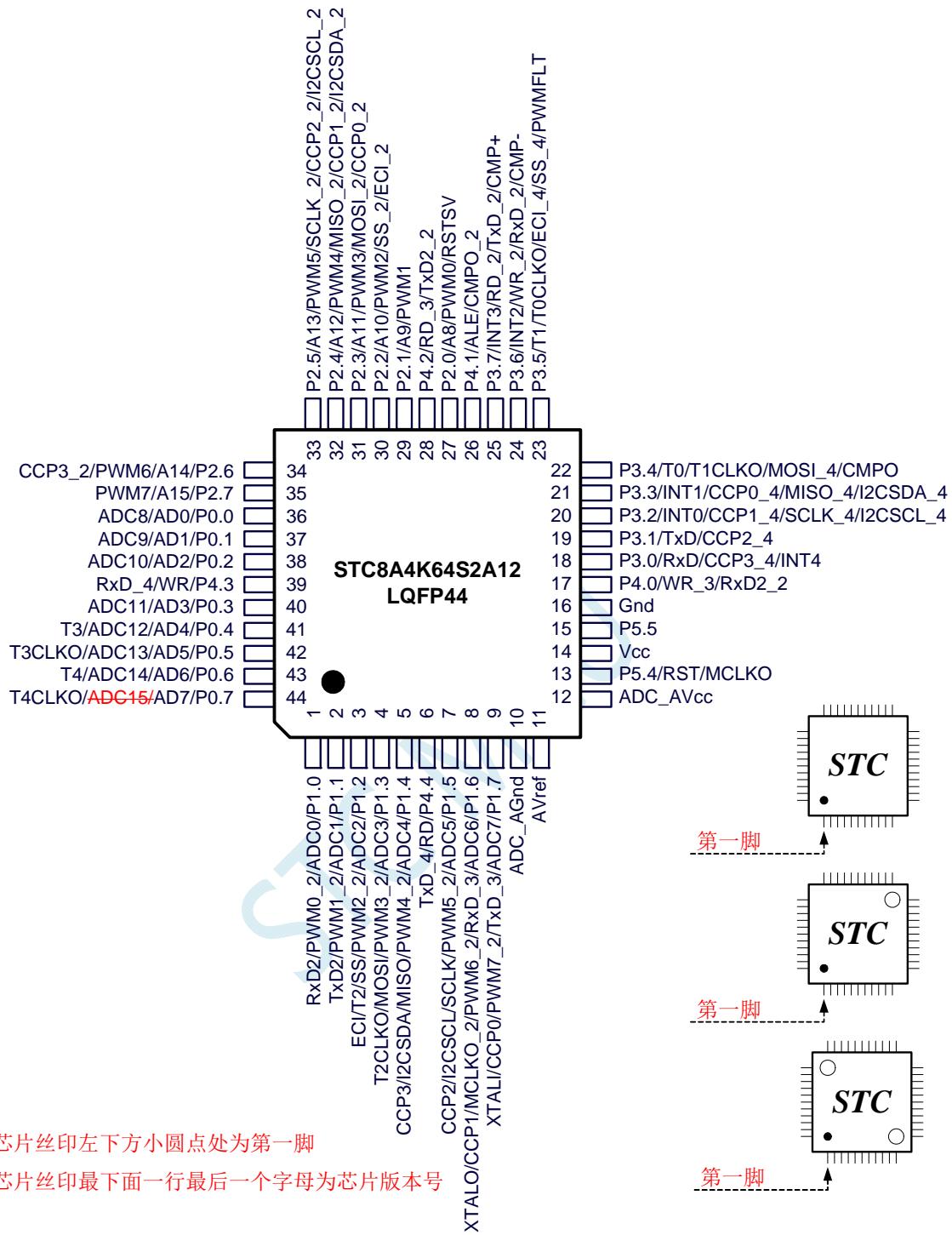


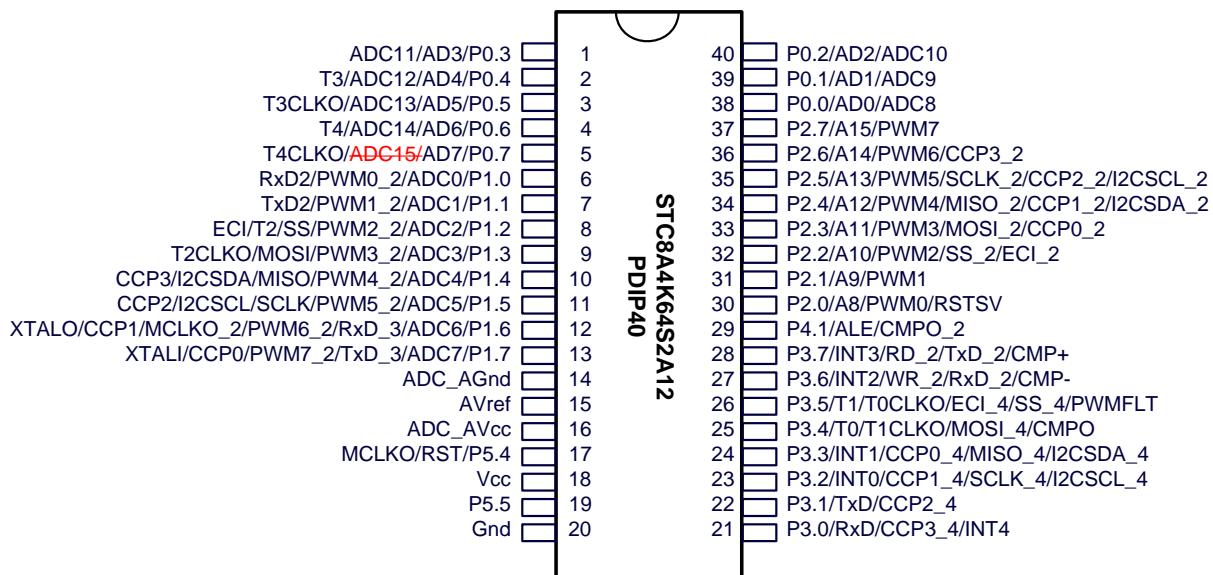


3.1.2 STC8A4K64S2A12 系列管脚图

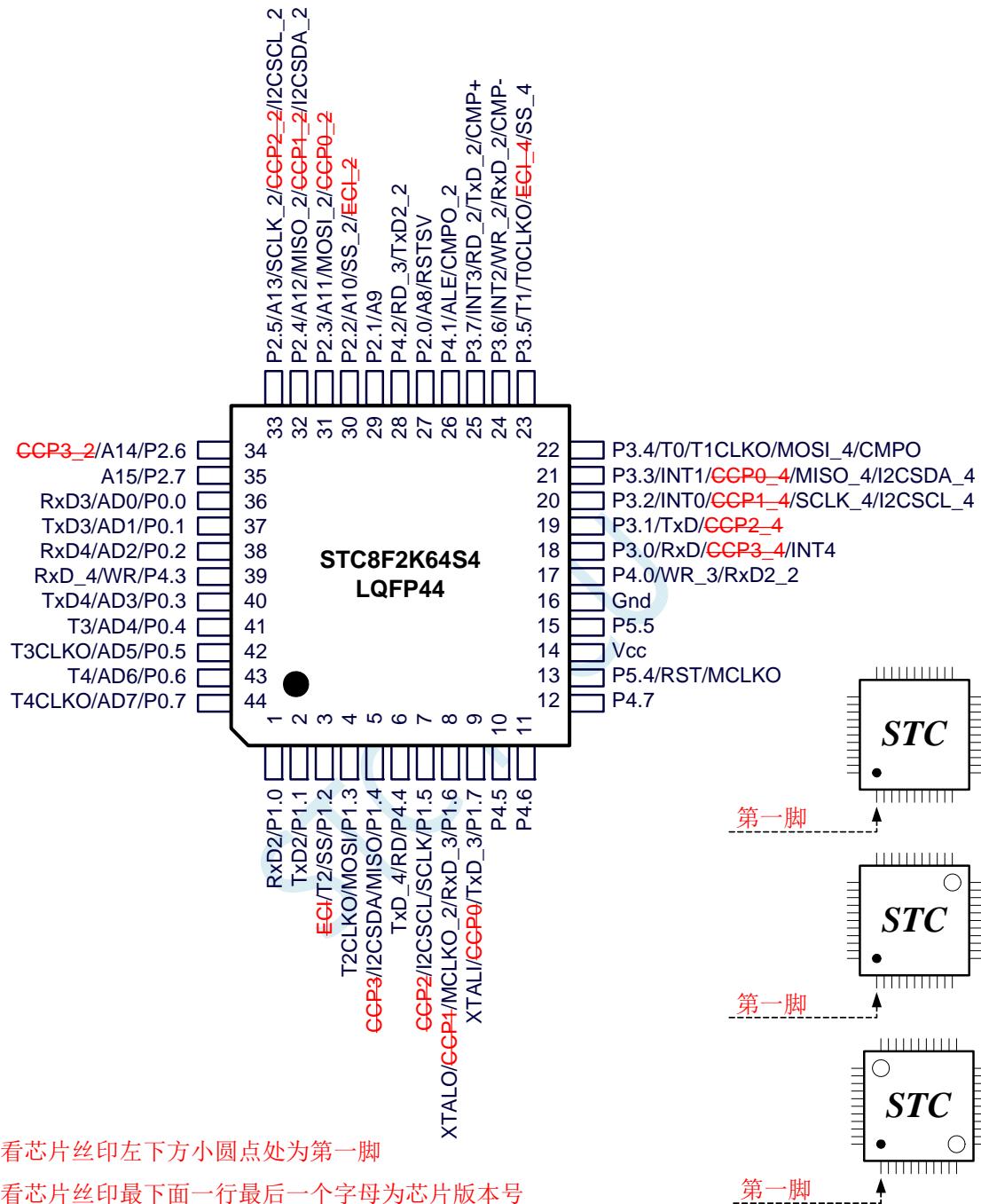


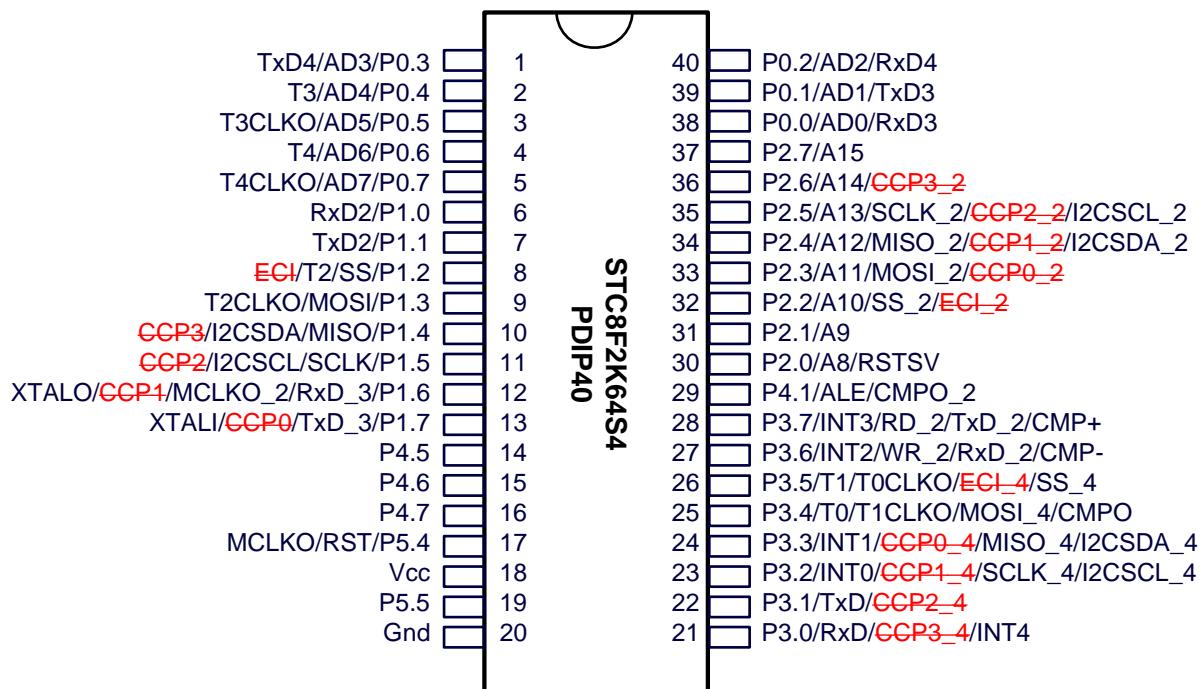


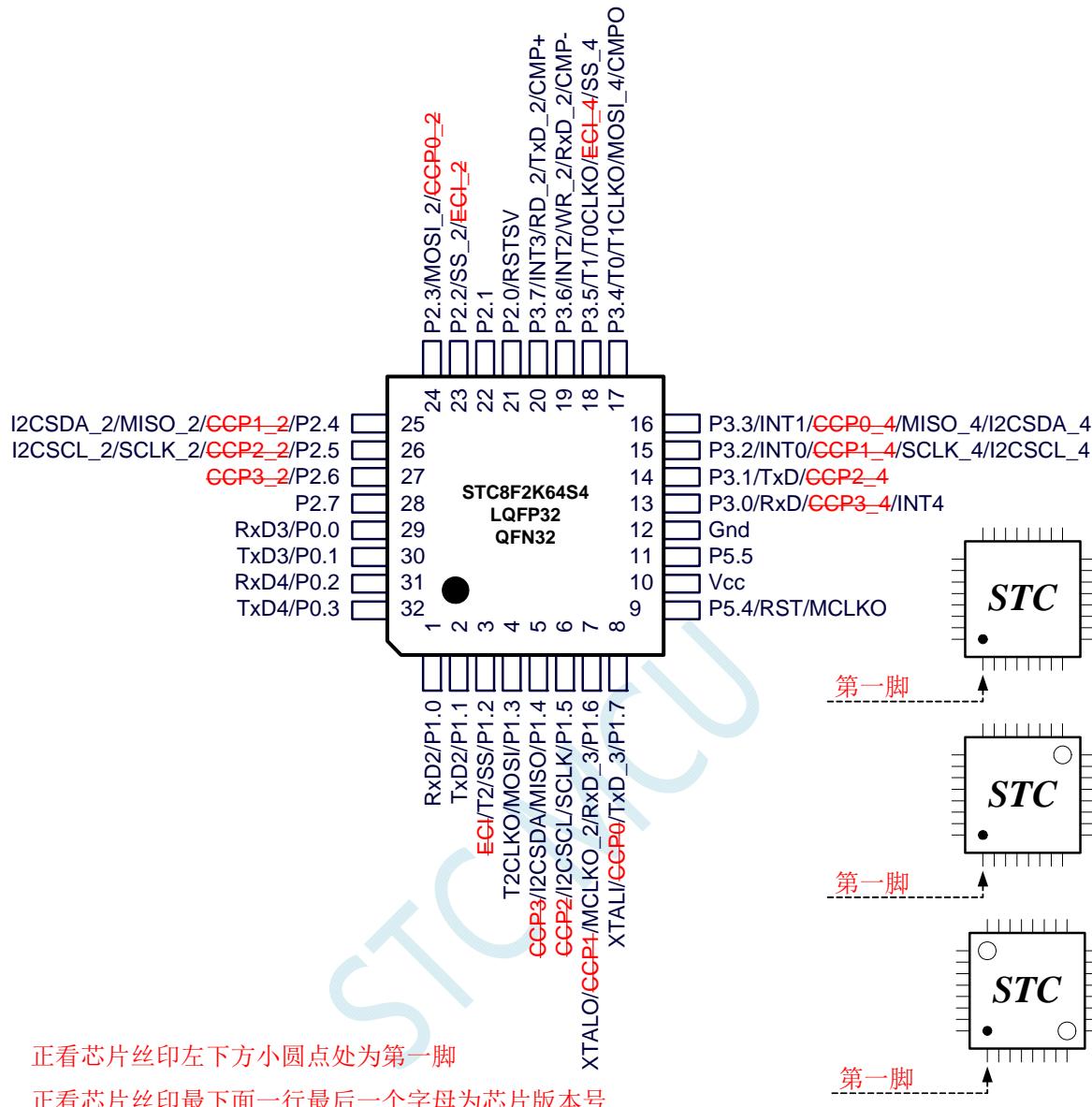




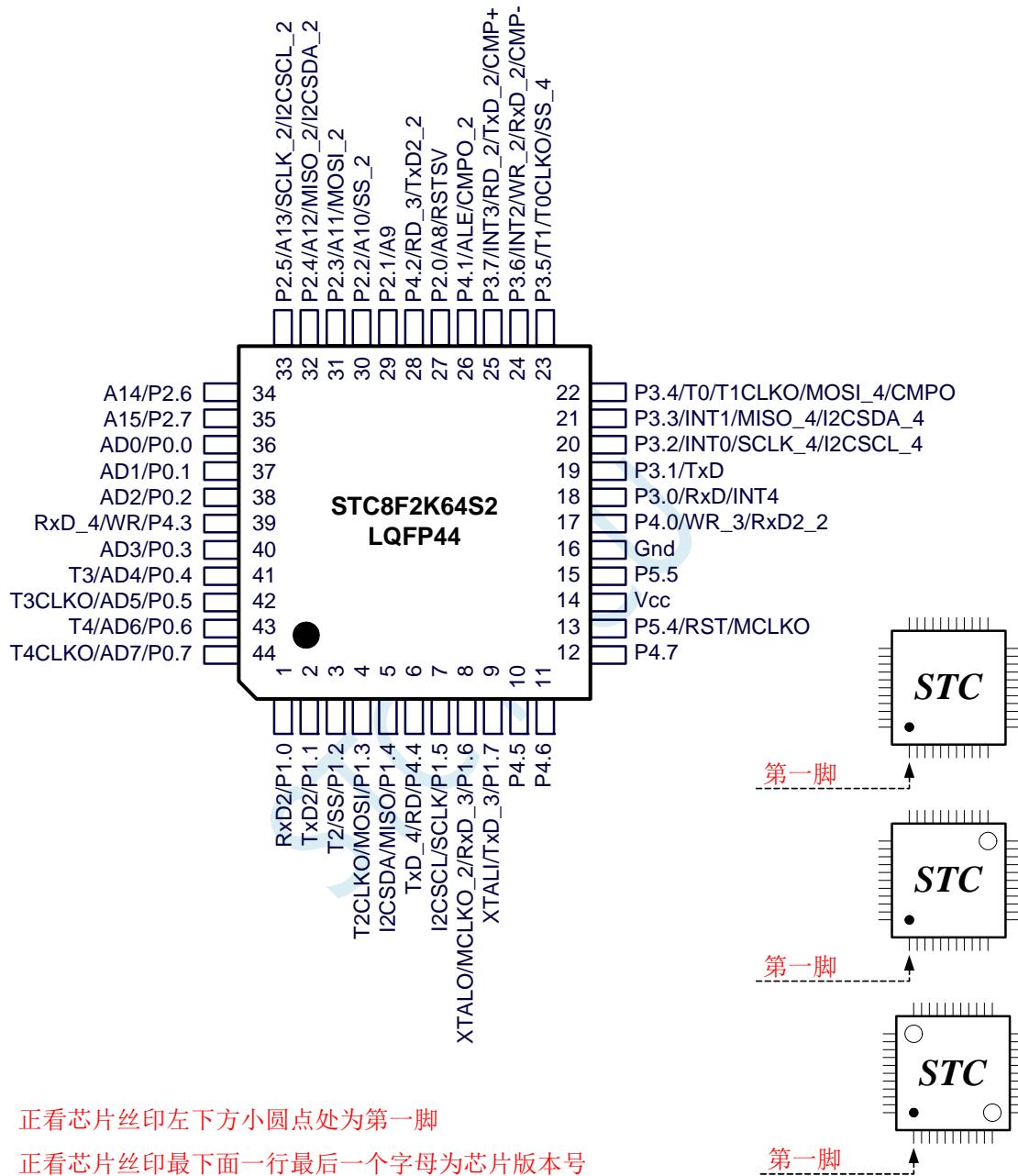
3.1.3 STC8F2K64S4 系列管脚图





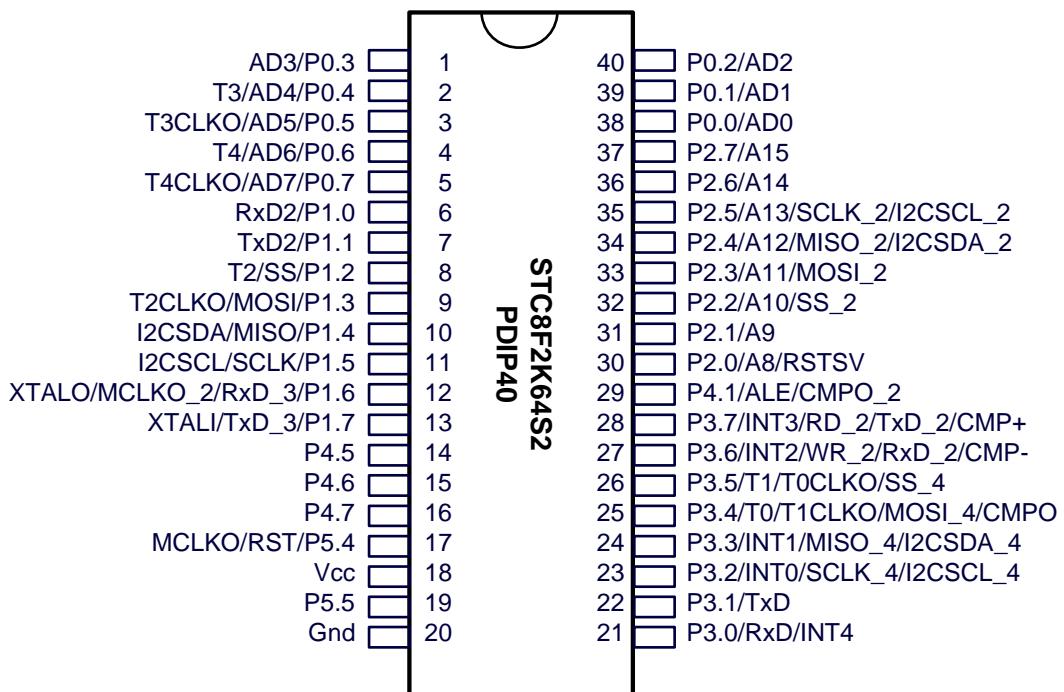


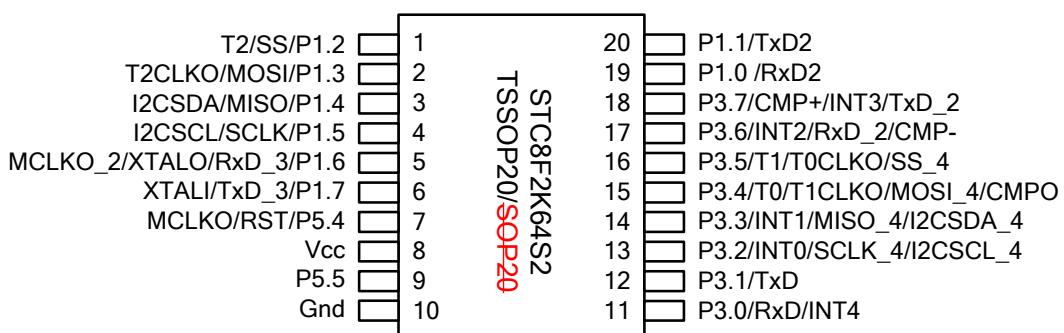
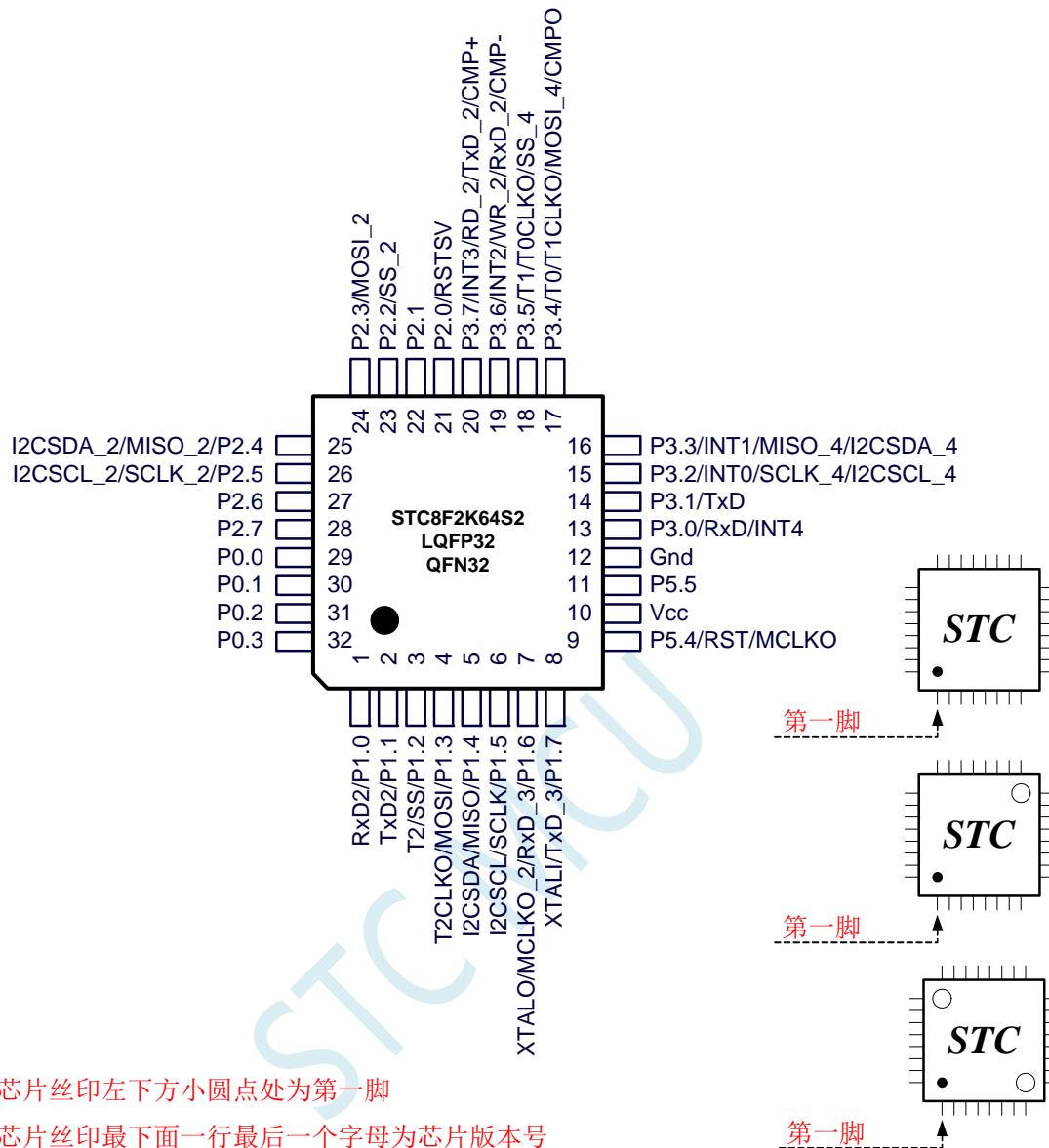
3.1.4 STC8F2K64S2 系列管脚图

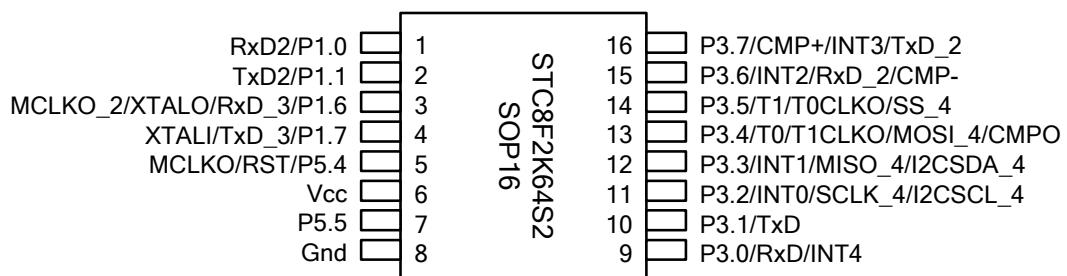


正看芯片丝印左下方小圆点处为第一脚

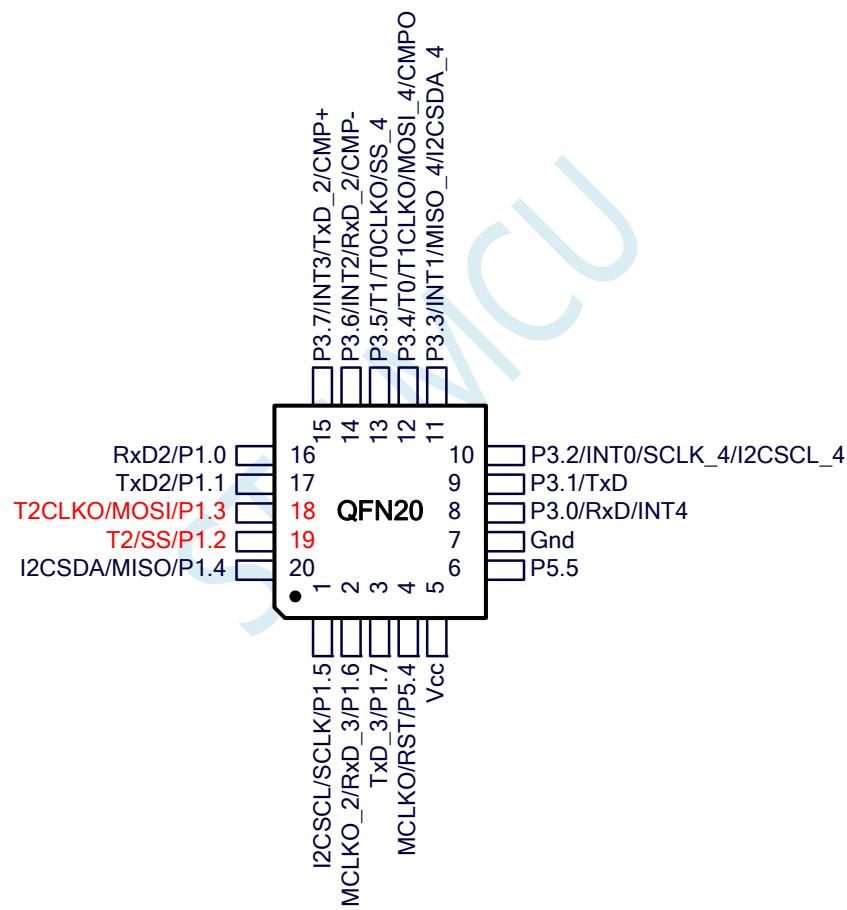
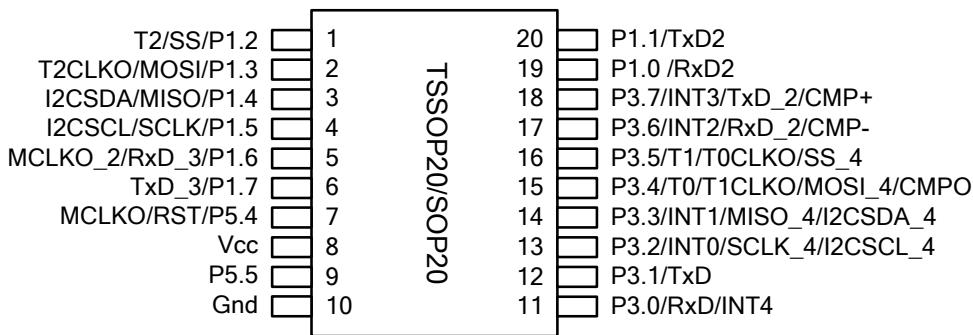
正看芯片丝印最下面一行最后一个字母为芯片版本号



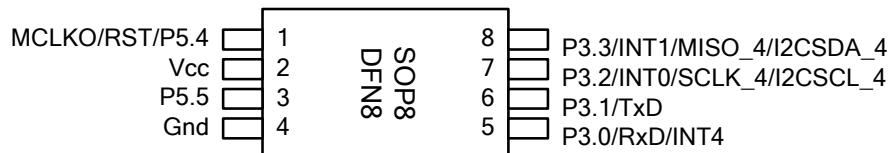
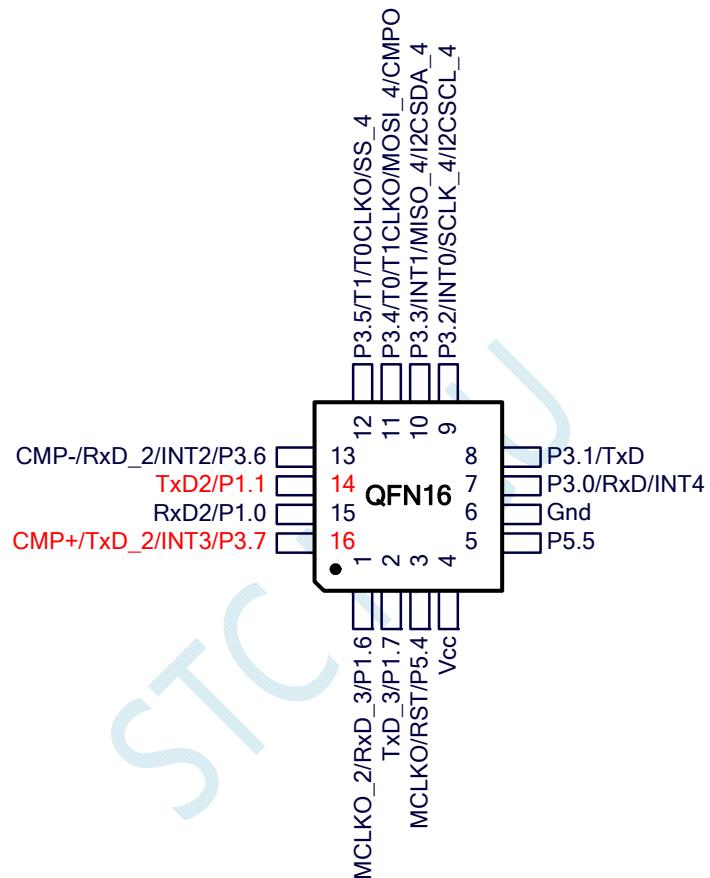
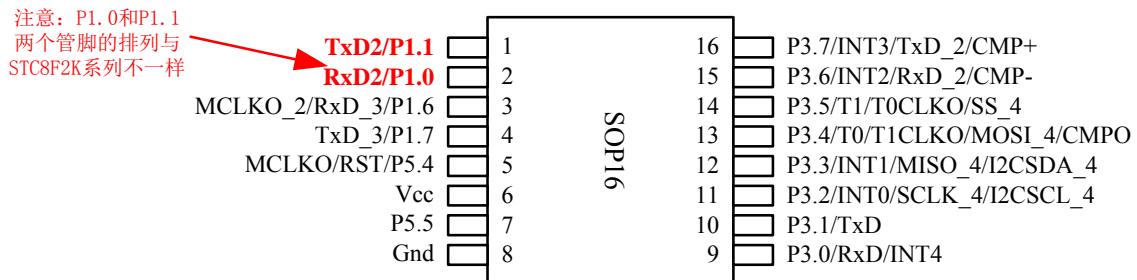




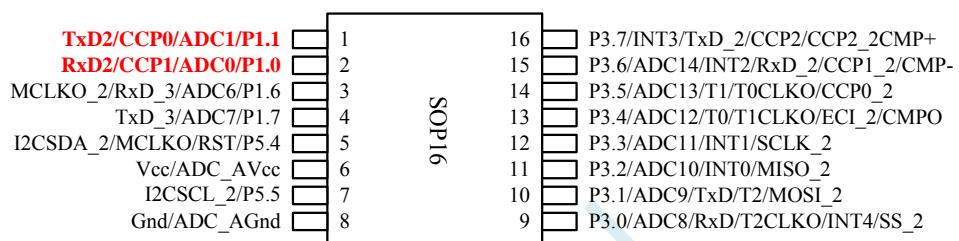
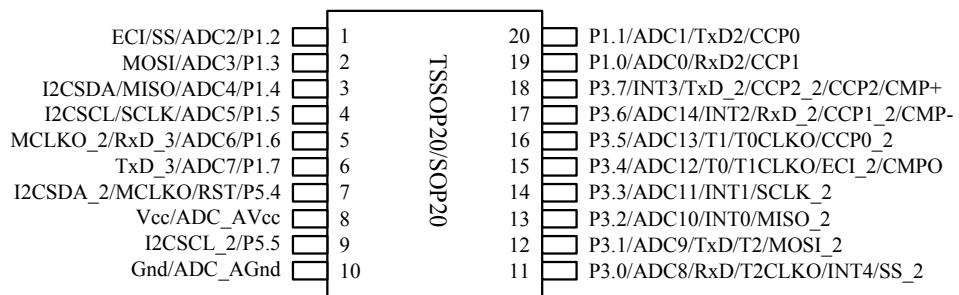
3.1.5 STC8F1K08S2 系列管脚图



QFN20 (3mm*3mm) 正在试产中

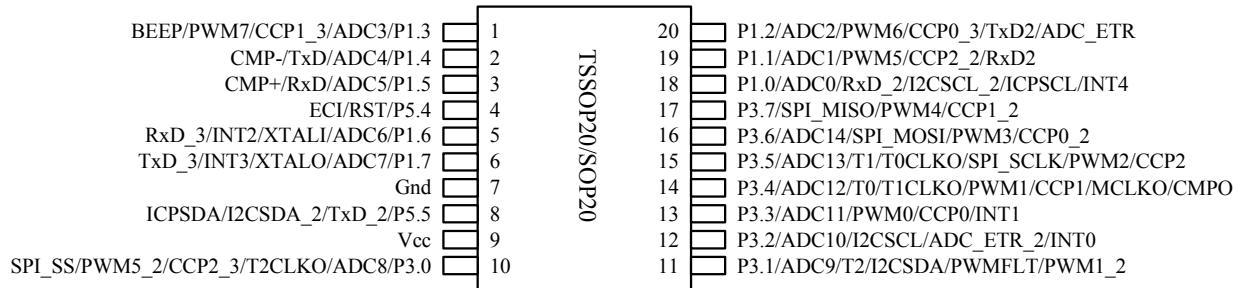


3.1.6 STC8C1K08S2A10 系列管脚图

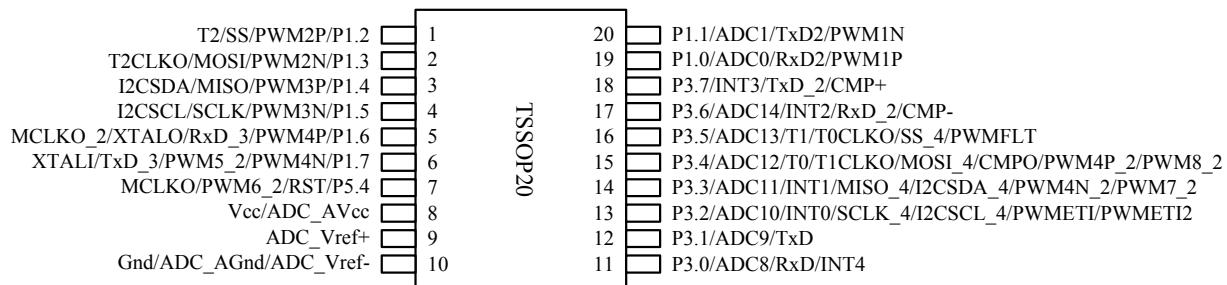


3.1.7 GX8S003 系列管脚图

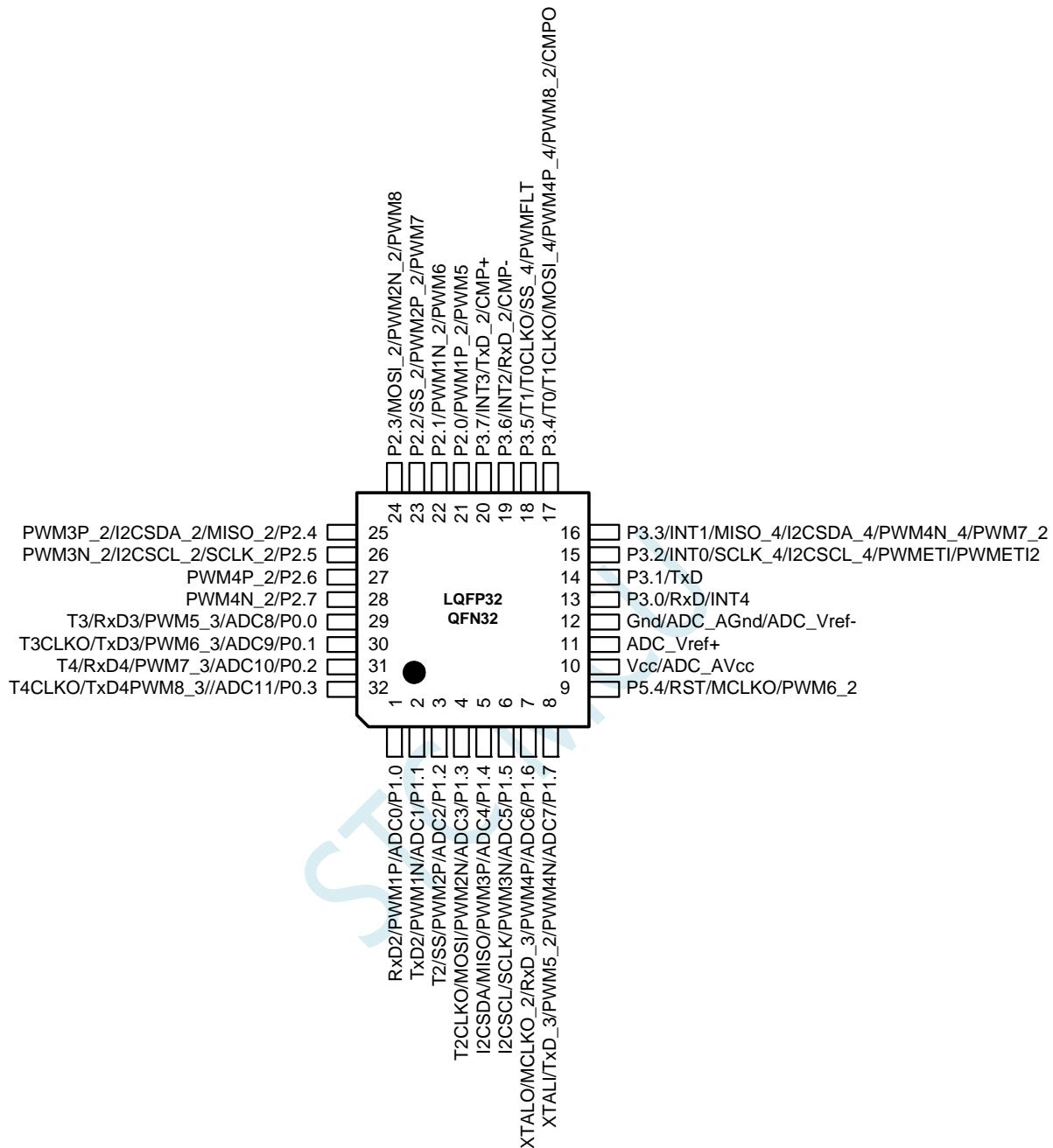
客户需求的特殊管脚封装



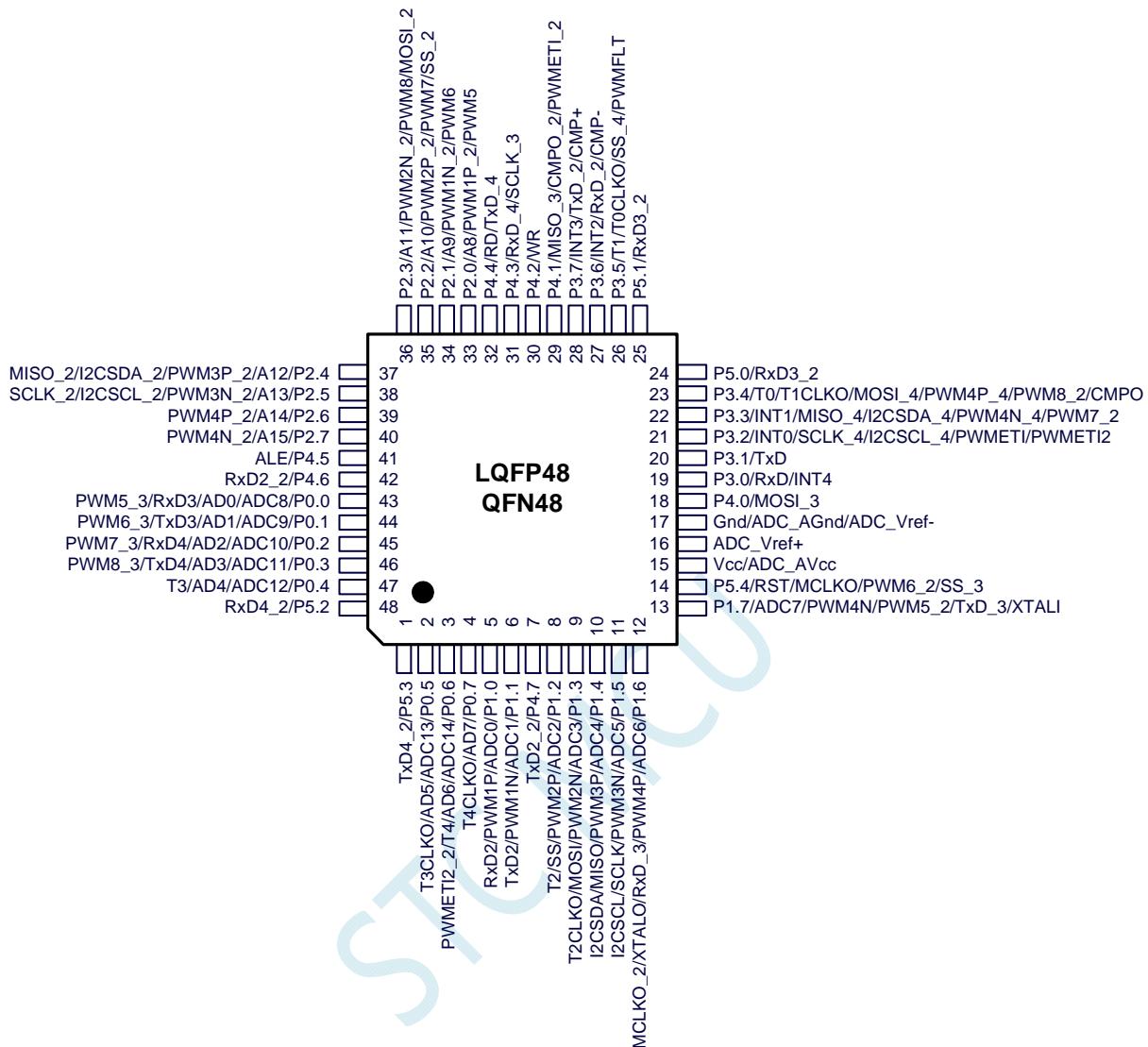
3.1.8 STC8P1K08S2A10 系列管脚图



3.1.9 STC8P1K16S2A10 系列管脚图



3.1.10 STC8P2K32S4A12 系列管脚图



3.2 管脚说明

3.2.1 STC8A8K64S4A12 系列管脚说明

编号				名称	类型	说明
LQFP64S	LQFP48	LQFP44	PDIP40			
1	1			P5.2	I/O	标准 IO 口
				RxD4_2	I	串口 4 的接收脚
2	2			P5.3	I/O	标准 IO 口
				TxD4_2	O	串口 4 的发送脚
3	3	2	7	P1.1	I/O	标准 IO 口
				ADC1	I	ADC 模拟输入通道 1
				PWM1_2	O	增强 PWM 通道 1 输出脚
				TxD2	O	串口 2 的发送脚
4	4	3	8	P1.2	I/O	标准 IO 口
				ADC2	I	ADC 模拟输入通道 2
				PWM2_2	O	增强 PWM 通道 2 输出脚
				SS	I/O	SPI 从机选择
				T2	I	定时器 2 外部时钟输入
				ECI	I	PCA 的外部脉冲输入
5	5	4	9	P1.3	I/O	标准 IO 口
				ADC3	I	ADC 模拟输入通道 3
				PWM3_2	O	增强 PWM 通道 3 输出脚
				MOSI	I/O	SPI 主机输出从机输入
				T2CLKO	O	定时器 2 时钟分频输出
6				P6.0	I/O	标准 IO 口
				PWM0_3	O	增强 PWM 通道 0 输出脚
7				P6.1	I/O	标准 IO 口
				PWM1_3	O	增强 PWM 通道 1 输出脚
8				P6.2	I/O	标准 IO 口
				PWM2_3	O	增强 PWM 通道 2 输出脚
9				P6.3	I/O	标准 IO 口
				PWM3_3	O	增强 PWM 通道 3 输出脚

编号				名称	类型	说明
LQFP64S	LQFP48	LQFP44	PDIP40			
10	6	5	10	P1.4	I/O	标准 IO 口
				ADC4	I	ADC 模拟输入通道 4
				PWM4_2	O	增强 PWM 通道 4 输出脚
				MISO	I/O	SPI 主机输入从机输出
				SDA	I/O	I2C 接口的数据线
				CCP3	I/O	PCA 的捕获输入和脉冲输出
11	7	6		P4.4	I/O	标准 IO 口
				RD	O	外部总线的读信号线
				TxD_4	O	串口 1 的发送脚
12	8	7	11	P1.5	I/O	标准 IO 口
				ADC5	I	ADC 模拟输入通道 5
				PWM5_2	O	增强 PWM 通道 5 输出脚
				SCLK	I/O	SPI 的时钟脚
				SCL	I/O	I2C 的时钟线
				CCP2	I/O	PCA 的捕获输入和脉冲输出
13	9	8	12	P1.6	I/O	标准 IO 口
				ADC6	I	ADC 模拟输入通道 6
				RxD_3	I	串口 1 的接收脚
				PWM6_2	O	增强 PWM 通道 6 输出脚
				MCLKO_2	O	主时钟分频输出
				CCP1	I/O	PCA 的捕获输入和脉冲输出
				XTALO	O	外部晶振的输出脚
14	10	9	13	P1.7	I/O	标准 IO 口
				ADC7	I	ADC 模拟输入通道 7
				TxD_3	O	串口 1 的发送脚
				PWM7_2	O	增强 PWM 通道 7 输出脚
				CCP0	I/O	PCA 的捕获输入和脉冲输出
				XTAL1	I	外部晶振/外部时钟的输入脚
15	11	10	14	ADC_AGnd	GND	ADC 地线
16	12	11	15	AVref	I	ADC 的参考电压脚
17	13	12	16	ADC_AVcc	VCC	ADC 电源脚
18	14	13	17	P5.4	I/O	标准 IO 口
				RST	I	复位引脚
				MCLKO	O	主时钟分频输出
19	15	14	18	Vcc	VCC	电源脚
20	16	15	19	P5.5	I/O	标准 IO 口
21	17	16	20	Gnd	GND	地线

编号				名称	类型	说明
LQFP64S	LQFP48	LQFP44	PDIP40			
22	18	17		P4.0	I/O	标准 IO 口
				WR_3	O	外部总线的写信号线
				RxD2_2	I	串口 2 的接收脚
23				P6.4	I/O	标准 IO 口
				PWM4_3	O	增强 PWM 通道 4 输出脚
24				P6.5	I/O	标准 IO 口
				PWM5_3	O	增强 PWM 通道 5 输出脚
25				P6.6	I/O	标准 IO 口
				PWM6_3	O	增强 PWM 通道 6 输出脚
26				P6.7	I/O	标准 IO 口
				PWM7_3	O	增强 PWM 通道 7 输出脚
27	19	18	21	P3.0	I/O	标准 IO 口
				RxD	I	串口 1 的接收脚
				CCP3_4	I/O	PCA 的捕获输入和脉冲输出
				INT4	I	外部中断 4
28	20	19	22	P3.1	I/O	标准 IO 口
				TxD	O	串口 1 的发送脚
				CCP2_4	I/O	PCA 的捕获输入和脉冲输出
29	21	20	23	P3.2	I/O	标准 IO 口
				INT0	I	外部中断 0
				CCP1_4	I/O	PCA 的捕获输入和脉冲输出
				SCLK_4	I/O	SPI 的时钟脚
				SCL_4	I/O	I2C 的时钟线
30	22	21	24	P3.3	I/O	标准 IO 口
				INT1	I	外部中断 1
				CCP0_4	I/O	PCA 的捕获输入和脉冲输出
				MISO_4	I/O	SPI 主机输入从机输出
				SDA_4	I/O	I2C 接口的数据线
31	23	22	25	P3.4	I/O	标准 IO 口
				T0	I	定时器 0 外部时钟输入
				T1CLKO	O	定时器 1 时钟分频输出
				MOSI_4	I/O	SPI 主机输出从机输入
				CMPO	O	比较器输出

编号				名称	类型	说明
LQFP64S	LQFP48	LQFP44	PDIP40			
32	24	23	26	P3.5	I/O	标准 IO 口
				T1	I	定时器 1 外部时钟输入
				T0CLKO	O	定时器 0 时钟分频输出
				ECI_4	I	PCA 的外部脉冲输入
				SS_4	I	SPI 的从机选择脚（主机为输出）
				PWMFLT	I	增强 PWM 的外部异常检测脚
33	25			P5.0	I/O	标准 IO 口
				RxD3_2	I	串口 3 的接收脚
34	26			P5.1	I/O	标准 IO 口
				TxD3_2	O	串口 3 的发送脚
35	27	24	27	P3.6	I/O	标准 IO 口
				INT2	I	外部中断 2
				WR_2	O	外部总线的写信号线
				RxD_2	I	串口 1 的接收脚
				CMP-	I	比较器负极输入
36	28	25	28	P3.7	I/O	标准 IO 口
				INT3	I	外部中断 3
				RD_2	O	外部总线的读信号线
				TxD_2	O	串口 1 的发送脚
				CMP+	I	比较器正极输入
37	29	26	29	P4.1	I/O	标准 IO 口
				ALE	O	地址锁存信号
				CMPO_2	O	比较器输出
38				P7.0	I/O	标准 IO 口
				CCP0_3	I/O	PCA 的捕获输入和脉冲输出
39				P7.1	I/O	标准 IO 口
				CCP1_3	I/O	PCA 的捕获输入和脉冲输出
40				P7.2	I/O	标准 IO 口
				CCP2_3	I/O	PCA 的捕获输入和脉冲输出
41				P7.3	I/O	标准 IO 口
				CCP3_3	I/O	PCA 的捕获输入和脉冲输出
42	30	27	30	P2.0	I/O	标准 IO 口
				A8	I	地址总线
				PWM0	O	增强 PWM 通道 0 输出脚
				RSTSVD	-	端口的初始电平可 ISP 下载时配置
43	31	28		P4.2	I/O	标准 IO 口
				RD_3	O	外部总线的读信号线
				TxD2_2	O	串口 2 的发送脚

编号				名称	类型	说明
LQFP64S	LQFP48	LQFP44	PDIP40			
44	32	29	31	P2.1	I/O	标准 IO 口
				A9	I	地址总线
				PWM1	O	增强 PWM 通道 1 输出脚
45	33	30	32	P2.2	I/O	标准 IO 口
				A10	I	地址总线
				PWM2	O	增强 PWM 通道 2 输出脚
				SS_2	I	SPI 的从机选择脚 (主机为输出)
				ECI_2	I	PCA 的外部脉冲输入
46	34	31	33	P2.3	I/O	标准 IO 口
				A11	I	地址总线
				PWM3	O	增强 PWM 通道 3 输出脚
				MOSI_2	I/O	SPI 主机输出从机输入
				CCP0_2	I/O	PCA 的捕获输入和脉冲输出
47	35	32	34	P2.4	I/O	标准 IO 口
				A12	I	地址总线
				PWM4	O	增强 PWM 通道 4 输出脚
				MISO_2	I/O	SPI 主机输入从机输出
				SDA_2	I/O	I2C 接口的数据线
				CCP1_2	I/O	PCA 的捕获输入和脉冲输出
48	36	33	35	P2.5	I/O	标准 IO 口
				A13	I	地址总线
				PWM5	O	增强 PWM 通道 5 输出脚
				SCLK_2	I/O	SPI 的时钟脚
				SCL_2	I/O	I2C 的时钟线
				CCP2_2	I/O	PCA 的捕获输入和脉冲输出
49	37	34	36	P2.6	I/O	标准 IO 口
				A14	I	地址总线
				PWM6	O	增强 PWM 通道 6 输出脚
				CCP3_2	I/O	PCA 的捕获输入和脉冲输出
50	38	35	37	P2.7	I/O	标准 IO 口
				A15	I	地址总线
				PWM7	O	增强 PWM 通道 7 输出脚
51	39	36	38	P0.0	I/O	标准 IO 口
				AD0	I	地址总线
				ADC8	I	ADC 模拟输入通道 8
				RxD3	I	串口 3 的接收脚

编号				名称	类型	说明
LQFP64S	LQFP48	LQFP44	PDIP40			
52	40	37	39	P0.1	I/O	标准 IO 口
				AD1	I	地址总线
				ADC9	I	ADC 模拟输入通道 9
				TxD3	O	串口 3 的发送脚
53	41	38	40	P0.2	I/O	标准 IO 口
				AD2	I	地址总线
				ADC10	I	ADC 模拟输入通道 10
				RxD4	I	串口 4 的接收脚
54				P7.4	I/O	标准 IO 口
				SS_3	I	SPI 的从机选择脚 (主机为输出)
				ECI_3	I	PCA 的外部脉冲输入
55				P7.5	I/O	标准 IO 口
				MOSI_3	I/O	SPI 主机输出从机输入
56				P7.6	I/O	标准 IO 口
				MISO_3	I/O	SPI 主机输入从机输出
				SDA_3	I/O	I2C 接口的数据线
57				P7.7	I/O	标准 IO 口
				SCLK_3	I/O	SPI 的时钟脚
				SCL_3	I/O	I2C 的时钟线
58	42	39		P4.3	I/O	标准 IO 口
				WR	O	外部总线的写信号线
				RxD_4	I	串口 1 的接收脚
59	43	40	1	P0.3	I/O	标准 IO 口
				AD3	I	地址总线
				ADC11	I	ADC 模拟输入通道 11
				TxD4	O	串口 4 的发送脚
60	44	41	2	P0.4	I/O	标准 IO 口
				AD4	I	地址总线
				ADC12	I	ADC 模拟输入通道 12
				T3	I	定时器 3 外部时钟输入

编号				名称	类型	说明
LQFP64S	LQFP48	LQFP44	PDIP40			
61	45	42	3	P0.5	I/O	标准 IO 口
				AD5	I	地址总线
				ADC13	I	ADC 模拟输入通道 13
				T3CLKO	O	定时器 3 时钟分频输出
62	46	43	4	P0.6	I/O	标准 IO 口
				AD6	I	地址总线
				ADC14	I	ADC 模拟输入通道 14
				T4	I	定时器 4 外部时钟输入
63	47	44	5	P0.7	I/O	标准 IO 口
				AD7	I	地址总线
				T4CLKO	O	定时器 4 时钟分频输出
64	48	1	6	P1.0	I/O	标准 IO 口
				ADC0	I	ADC 模拟输入通道 0
				PWM0_2	O	增强 PWM 通道 0 输出脚
				RxD2	I	串口 2 的接收脚

3.2.2 STC8A4K64S2A12 系列管脚说明

编号				名称	类型	说明
LQFP64S	LQFP48	LQFP44	PDIP40			
1	1			P5.2	I/O	标准 IO 口
2	2			P5.3	I/O	标准 IO 口
3	3	2	7	P1.1	I/O	标准 IO 口
				ADC1	I	ADC 模拟输入通道 1
				PWM1_2	O	增强 PWM 通道 1 输出脚
				TxD2	O	串口 2 的发送脚
4	4	3	8	P1.2	I/O	标准 IO 口
				ADC2	I	ADC 模拟输入通道 2
				PWM2_2	O	增强 PWM 通道 2 输出脚
				SS	I/O	SPI 从机选择
				T2	I	定时器 2 外部时钟输入
				ECI	I	PCA 的外部脉冲输入
5	5	4	9	P1.3	I/O	标准 IO 口
				ADC3	I	ADC 模拟输入通道 3
				PWM3_2	O	增强 PWM 通道 3 输出脚
				MOSI	I/O	SPI 主机输出从机输入
				T2CLKO	O	定时器 2 时钟分频输出
6				P6.0	I/O	标准 IO 口
				PWM0_3	O	增强 PWM 通道 0 输出脚
7				P6.1	I/O	标准 IO 口
				PWM1_3	O	增强 PWM 通道 1 输出脚
8				P6.2	I/O	标准 IO 口
				PWM2_3	O	增强 PWM 通道 2 输出脚
9				P6.3	I/O	标准 IO 口
				PWM3_3	O	增强 PWM 通道 3 输出脚

编号				名称	类型	说明
LQFP64S	LQFP48	LQFP44	PDIP40			
10	6	5	10	P1.4	I/O	标准 IO 口
				ADC4	I	ADC 模拟输入通道 4
				PWM4_2	O	增强 PWM 通道 4 输出脚
				MISO	I/O	SPI 主机输入从机输出
				SDA	I/O	I2C 接口的数据线
				CCP3	I/O	PCA 的捕获输入和脉冲输出
11	7	6		P4.4	I/O	标准 IO 口
				RD	O	外部总线的读信号线
				TxD_4	O	串口 1 的发送脚
12	8	7	11	P1.5	I/O	标准 IO 口
				ADC5	I	ADC 模拟输入通道 5
				PWM5_2	O	增强 PWM 通道 5 输出脚
				SCLK	I/O	SPI 的时钟脚
				SCL	I/O	I2C 的时钟线
				CCP2	I/O	PCA 的捕获输入和脉冲输出
13	9	8	12	P1.6	I/O	标准 IO 口
				ADC6	I	ADC 模拟输入通道 6
				RxD_3	I	串口 1 的接收脚
				PWM6_2	O	增强 PWM 通道 6 输出脚
				MCLKO_2	O	主时钟分频输出
				CCP1	I/O	PCA 的捕获输入和脉冲输出
				XTALO	O	外部晶振的输出脚
14	10	9	13	P1.7	I/O	标准 IO 口
				ADC7	I	ADC 模拟输入通道 7
				TxD_3	O	串口 1 的发送脚
				PWM7_2	O	增强 PWM 通道 7 输出脚
				CCP0	I/O	PCA 的捕获输入和脉冲输出
				XTAL1	I	外部晶振/外部时钟的输入脚
15	11	10	14	ADC_AGnd	GND	ADC 地线
16	12	11	15	AVref	I	ADC 的参考电压脚
17	13	12	16	ADC_AVcc	VCC	ADC 电源脚
18	14	13	17	P5.4	I/O	标准 IO 口
				RST	I	复位引脚
				MCLKO	O	主时钟分频输出
19	15	14	18	Vcc	VCC	电源脚
20	16	15	19	P5.5	I/O	标准 IO 口
21	17	16	20	Gnd	GND	地线

编号				名称	类型	说明
LQFP64S	LQFP48	LQFP44	PDIP40			
22	18	17		P4.0	I/O	标准 IO 口
				WR_3	O	外部总线的写信号线
				RxD2_2	I	串口 2 的接收脚
23				P6.4	I/O	标准 IO 口
				PWM4_3	O	增强 PWM 通道 4 输出脚
24				P6.5	I/O	标准 IO 口
				PWM5_3	O	增强 PWM 通道 5 输出脚
25				P6.6	I/O	标准 IO 口
				PWM6_3	O	增强 PWM 通道 6 输出脚
26				P6.7	I/O	标准 IO 口
				PWM7_3	O	增强 PWM 通道 7 输出脚
27	19	18	21	P3.0	I/O	标准 IO 口
				RxD	I	串口 1 的接收脚
				CCP3_4	I/O	PCA 的捕获输入和脉冲输出
				INT4	I	外部中断 4
28	20	19	22	P3.1	I/O	标准 IO 口
				TxD	O	串口 1 的发送脚
				CCP2_4	I/O	PCA 的捕获输入和脉冲输出
29	21	20	23	P3.2	I/O	标准 IO 口
				INT0	I	外部中断 0
				CCP1_4	I/O	PCA 的捕获输入和脉冲输出
				SCLK_4	I/O	SPI 的时钟脚
				SCL_4	I/O	I2C 的时钟线
30	22	21	24	P3.3	I/O	标准 IO 口
				INT1	I	外部中断 1
				CCP0_4	I/O	PCA 的捕获输入和脉冲输出
				MISO_4	I/O	SPI 主机输入从机输出
				SDA_4	I/O	I2C 接口的数据线
31	23	22	25	P3.4	I/O	标准 IO 口
				T0	I	定时器 0 外部时钟输入
				T1CLKO	O	定时器 1 时钟分频输出
				MOSI_4	I/O	SPI 主机输出从机输入
				CMPO	O	比较器输出

编号				名称	类型	说明
LQFP64S	LQFP48	LQFP44	PDIP40			
32	24	23	26	P3.5	I/O	标准 IO 口
				T1	I	定时器 1 外部时钟输入
				T0CLKO	O	定时器 0 时钟分频输出
				ECI_4	I	PCA 的外部脉冲输入
				SS_4	I	SPI 的从机选择脚（主机为输出）
				PWMFLT	I	增强 PWM 的外部异常检测脚
33	25			P5.0	I/O	标准 IO 口
34	26			P5.1	I/O	标准 IO 口
35	27	24	27	P3.6	I/O	标准 IO 口
				INT2	I	外部中断 2
				WR_2	O	外部总线的写信号线
				RxD_2	I	串口 1 的接收脚
				CMP-	I	比较器负极输入
36	28	25	28	P3.7	I/O	标准 IO 口
				INT3	I	外部中断 3
				RD_2	O	外部总线的读信号线
				TxD_2	O	串口 1 的发送脚
				CMP+	I	比较器正极输入
37	29	26	29	P4.1	I/O	标准 IO 口
				ALE	O	地址锁存信号
				CMPO_2	O	比较器输出
38				P7.0	I/O	标准 IO 口
				CCP0_3	I/O	PCA 的捕获输入和脉冲输出
39				P7.1	I/O	标准 IO 口
				CCP1_3	I/O	PCA 的捕获输入和脉冲输出
40				P7.2	I/O	标准 IO 口
				CCP2_3	I/O	PCA 的捕获输入和脉冲输出
41				P7.3	I/O	标准 IO 口
				CCP3_3	I/O	PCA 的捕获输入和脉冲输出
42	30	27	30	P2.0	I/O	标准 IO 口
				A8	I	地址总线
				PWM0	O	增强 PWM 通道 0 输出脚
				RSTSV	-	端口的初始电平可 ISP 下载时配置
43	31	28		P4.2	I/O	标准 IO 口
				RD_3	O	外部总线的读信号线
				TxD2_2	O	串口 2 的发送脚

编号				名称	类型	说明
LQFP64S	LQFP48	LQFP44	PDIP40			
44	32	29	31	P2.1	I/O	标准 IO 口
				A9	I	地址总线
				PWM1	O	增强 PWM 通道 1 输出脚
45	33	30	32	P2.2	I/O	标准 IO 口
				A10	I	地址总线
				PWM2	O	增强 PWM 通道 2 输出脚
				SS_2	I	SPI 的从机选择脚 (主机为输出)
				ECI_2	I	PCA 的外部脉冲输入
46	34	31	33	P2.3	I/O	标准 IO 口
				A11	I	地址总线
				PWM3	O	增强 PWM 通道 3 输出脚
				MOSI_2	I/O	SPI 主机输出从机输入
				CCP0_2	I/O	PCA 的捕获输入和脉冲输出
47	35	32	34	P2.4	I/O	标准 IO 口
				A12	I	地址总线
				PWM4	O	增强 PWM 通道 4 输出脚
				MISO_2	I/O	SPI 主机输入从机输出
				SDA_2	I/O	I2C 接口的数据线
				CCP1_2	I/O	PCA 的捕获输入和脉冲输出
48	36	33	35	P2.5	I/O	标准 IO 口
				A13	I	地址总线
				PWM5	O	增强 PWM 通道 5 输出脚
				SCLK_2	I/O	SPI 的时钟脚
				SCL_2	I/O	I2C 的时钟线
				CCP2_2	I/O	PCA 的捕获输入和脉冲输出
49	37	34	36	P2.6	I/O	标准 IO 口
				A14	I	地址总线
				PWM6	O	增强 PWM 通道 6 输出脚
				CCP3_2	I/O	PCA 的捕获输入和脉冲输出
50	38	35	37	P2.7	I/O	标准 IO 口
				A15	I	地址总线
				PWM7	O	增强 PWM 通道 7 输出脚
51	39	36	38	P0.0	I/O	标准 IO 口
				AD0	I	地址总线
				ADC8	I	ADC 模拟输入通道 8

编号				名称	类型	说明
LQFP64S	LQFP48	LQFP44	PDIP40			
52	40	37	39	P0.1	I/O	标准 IO 口
				AD1	I	地址总线
				ADC9	I	ADC 模拟输入通道 9
53	41	38	40	P0.2	I/O	标准 IO 口
				AD2	I	地址总线
				ADC10	I	ADC 模拟输入通道 10
54				P7.4	I/O	标准 IO 口
				SS_3	I	SPI 的从机选择脚 (主机为输出)
				ECI_3	I	PCA 的外部脉冲输入
55				P7.5	I/O	标准 IO 口
				MOSI_3	I/O	SPI 主机输出从机输入
56				P7.6	I/O	标准 IO 口
				MISO_3	I/O	SPI 主机输入从机输出
				SDA_3	I/O	I2C 接口的数据线
57				P7.7	I/O	标准 IO 口
				SCLK_3	I/O	SPI 的时钟脚
				SCL_3	I/O	I2C 的时钟线
58	42	39		P4.3	I/O	标准 IO 口
				WR	O	外部总线的写信号线
				RxD_4	I	串口 1 的接收脚
59	43	40	1	P0.3	I/O	标准 IO 口
				AD3	I	地址总线
				ADC11	I	ADC 模拟输入通道 11
60	44	41	2	P0.4	I/O	标准 IO 口
				AD4	I	地址总线
				ADC12	I	ADC 模拟输入通道 12
				T3	I	定时器 3 外部时钟输入

编号				名称	类型	说明
LQFP64S	LQFP48	LQFP44	PDIP40			
61	45	42	3	P0.5	I/O	标准 IO 口
				AD5	I	地址总线
				ADC13	I	ADC 模拟输入通道 13
				T3CLKO	O	定时器 3 时钟分频输出
62	46	43	4	P0.6	I/O	标准 IO 口
				AD6	I	地址总线
				ADC14	I	ADC 模拟输入通道 14
				T4	I	定时器 4 外部时钟输入
63	47	44	5	P0.7	I/O	标准 IO 口
				AD7	I	地址总线
				T4CLKO	O	定时器 4 时钟分频输出
64	48	1	6	P1.0	I/O	标准 IO 口
				ADC0	I	ADC 模拟输入通道 0
				PWM0_2	O	增强 PWM 通道 0 输出脚
				RxD2	I	串口 2 的接收脚

3.2.3 STC8F2K64S4 系列管脚说明

编号			名称	类型	说明
LQFP44	PDIP40	LQFP32			
2	7	2	P1.1	I/O	标准 IO 口
			TxD2	O	串口 2 的发送脚
3	8	3	P1.2	I/O	标准 IO 口
			SS	I	SPI 的从机选择脚（主机为输出）
			T2	I	定时器 2 外部时钟输入
4	9	4	P1.3	I/O	标准 IO 口
			MOSI	I/O	SPI 主机输出从机输入
			T2CLKO	O	定时器 2 时钟分频输出
5	10	5	P1.4	I/O	标准 IO 口
			MISO	I/O	SPI 主机输入从机输出
			SDA	I/O	I2C 接口的数据线
6			P4.4	I/O	标准 IO 口
			RD	O	外部总线的读信号线
			TxD_4	O	串口 1 的发送脚
7	11	6	P1.5	I/O	标准 IO 口
			SCLK	I/O	SPI 的时钟脚
			SCL	I/O	I2C 的时钟线
8	12	7	P1.6	I/O	标准 IO 口
			RxD_3	I	串口 1 的接收脚
			XTALO	O	外部晶振的输出脚
			MCLKO_2	O	主时钟分频输出
9	13	8	P1.7	I/O	标准 IO 口
			TxD_3	O	串口 1 的发送脚
			XTALI	I	外部晶振/外部时钟的输入脚
10	14		P4.5	I/O	标准 IO 口
11	15		P4.6	I/O	标准 IO 口
12	16		P4.7	I/O	标准 IO 口
13	17	9	P5.4	I/O	标准 IO 口
			RST	I	复位引脚
			MCLKO	O	主时钟分频输出
14	18	10	Vcc	VCC	电源脚

编号			名称	类型	说明
LQFP44	PDIP40	LQFP32			
15	19	11	P5.5	I/O	标准 IO 口
16	20	12	Gnd	GND	地线
17			P4.0	I/O	标准 IO 口
			WR_3	O	外部总线的写信号线
			RxD2_2	I	串口 2 的接收脚
18	21	13	P3.0	I/O	标准 IO 口
			RxD	I	串口 1 的接收脚
			INT4	I	外部中断 4
19	22	14	P3.1	I/O	标准 IO 口
			TxD	O	串口 1 的发送脚
20	23	15	P3.2	I/O	标准 IO 口
			INT0	I	外部中断 0
			SCL_4	I/O	I2C 的时钟线
			SCLK_4	I/O	SPI 的时钟脚
21	24	16	P3.3	I/O	标准 IO 口
			INT1	I	外部中断 1
			SDA_4	I/O	I2C 接口的数据线
			MISO_4	I/O	SPI 主机输入从机输出
22	25	17	P3.4	I/O	标准 IO 口
			T0	I	定时器 0 外部时钟输入
			T1CLKO	O	定时器 1 时钟分频输出
			MOSI_4	I/O	SPI 主机输出从机输入
			CMPO	O	比较器输出
23	26	18	P3.5	I/O	标准 IO 口
			T1	I	定时器 1 外部时钟输入
			T0CLKO	O	定时器 0 时钟分频输出
			SS_4	I	SPI 的从机选择脚（主机为输出）
24	27	19	P3.6	I/O	标准 IO 口
			INT2	I	外部中断 2
			WR_2	O	外部总线的写信号线
			RxD_2	I	串口 1 的接收脚
			CMP-	I	比较器负极输入

编号			名称	类型	说明
LQFP44	PDIP40	LQFP32			
25	28	20	P3.7	I/O	标准 IO 口
			INT3	I	外部中断 3
			RD_2	O	外部总线的读信号线
			TxD_2	O	串口 1 的发送脚
			CMP+	I	比较器正极输入
26	29		P4.1	I/O	标准 IO 口
			ALE	O	地址锁存信号
			CMPO_2	O	比较器输出
27	30	21	P2.0	I/O	标准 IO 口
			A8	I	地址总线
			RSTSV	-	端口的初始电平可 ISP 下载时配置
28			P4.2	I/O	标准 IO 口
			RD_3	O	外部总线的读信号线
			TxD2_2	O	串口 2 的发送脚
29	31	22	P2.1	I/O	标准 IO 口
			A9	I	地址总线
30	32	23	P2.2	I/O	标准 IO 口
			A10	I	地址总线
			SS_2	I	SPI 的从机选择脚 (主机为输出)
31	33	24	P2.3	I/O	标准 IO 口
			A11	I	地址总线
			MOSI_2	I/O	SPI 主机输出从机输入
32	34	25	P2.4	I/O	标准 IO 口
			A12	I	地址总线
			MISO_2	I/O	SPI 主机输入从机输出
			SDA_2	I/O	I2C 接口的数据线
33	35	26	P2.5	I/O	标准 IO 口
			A13	I	地址总线
			SCLK_2	I/O	SPI 的时钟脚
			SCL_2	I/O	I2C 的时钟线
34	36	27	P2.6	I/O	标准 IO 口
			A14	I	地址总线
35	37	28	P2.7	I/O	标准 IO 口
			A15	I	地址总线

编号			名称	类型	说明
LQFP44	PDIP40	LQFP32			
36	38	29	P0.0	I/O	标准 IO 口
			AD0	I	地址总线
			RxD3	I	串口 3 的接收脚
37	39	30	P0.1	I/O	标准 IO 口
			AD1	I	地址总线
			TxD3	O	串口 3 的发送脚
38	40	31	P0.2	I/O	标准 IO 口
			AD2	I	地址总线
			RxD4	I	串口 4 的接收脚
39			P4.3	I/O	标准 IO 口
			WR	O	外部总线的写信号线
			RxD_4	I	串口 1 的接收脚
40	1	32	P0.3	I/O	标准 IO 口
			AD3	I	地址总线
			TxD4	O	串口 4 的发送脚
41	2		P0.4	I/O	标准 IO 口
			AD4	I	地址总线
			T3	I	定时器 3 外部时钟输入
42	3		P0.5	I/O	标准 IO 口
			AD5	I	地址总线
			T3CLKO	O	定时器 3 时钟分频输出
43	4		P0.6	I/O	标准 IO 口
			AD6	I	地址总线
			T4	I	定时器 4 外部时钟输入
44	5		P0.7	I/O	标准 IO 口
			AD7	I	地址总线
			T4CLKO	O	定时器 4 时钟分频输出
1	6	1	P1.0	I/O	标准 IO 口
			RxD2	I	串口 2 的接收脚

3.2.4 STC8F2K64S2 系列管脚说明

编号			名称	类型	说明
LQFP44	PDIP40	LQFP32			
2	7	2	P1.1	I/O	标准 IO 口
			TxD2	O	串口 2 的发送脚
3	8	3	P1.2	I/O	标准 IO 口
			SS	I	SPI 的从机选择脚（主机为输出）
			T2	I	定时器 2 外部时钟输入
4	9	4	P1.3	I/O	标准 IO 口
			MOSI	I/O	SPI 主机输出从机输入
			T2CLKO	O	定时器 2 时钟分频输出
5	10	5	P1.4	I/O	标准 IO 口
			MISO	I/O	SPI 主机输入从机输出
			SDA	I/O	I2C 接口的数据线
6			P4.4	I/O	标准 IO 口
			RD	O	外部总线的读信号线
			TxD_4	O	串口 1 的发送脚
7	11	6	P1.5	I/O	标准 IO 口
			SCLK	I/O	SPI 的时钟脚
			SCL	I/O	I2C 的时钟线
8	12	7	P1.6	I/O	标准 IO 口
			RxD_3	I	串口 1 的接收脚
			XTALO	O	外部晶振的输出脚
			MCLKO_2	O	主时钟分频输出
9	13	8	P1.7	I/O	标准 IO 口
			TxD_3	O	串口 1 的发送脚
			XTALI	I	外部晶振/外部时钟的输入脚
10	14		P4.5	I/O	标准 IO 口
11	15		P4.6	I/O	标准 IO 口
12	16		P4.7	I/O	标准 IO 口
13	17	9	P5.4	I/O	标准 IO 口
			RST	I	复位引脚
			MCLKO	O	主时钟分频输出
14	18	10	Vcc	VCC	电源脚

编号			名称	类型	说明
LQFP44	PDIP40	LQFP32			
15	19	11	P5.5	I/O	标准 IO 口
16	20	12	Gnd	GND	地线
17			P4.0	I/O	标准 IO 口
			WR_3	O	外部总线的写信号线
			RxD2_2	I	串口 2 的接收脚
18	21	13	P3.0	I/O	标准 IO 口
			RxD	I	串口 1 的接收脚
			INT4	I	外部中断 4
19	22	14	P3.1	I/O	标准 IO 口
			TxD	O	串口 1 的发送脚
20	23	15	P3.2	I/O	标准 IO 口
			INT0	I	外部中断 0
			SCL_4	I/O	I2C 的时钟线
			SCLK_4	I/O	SPI 的时钟脚
21	24	16	P3.3	I/O	标准 IO 口
			INT1	I	外部中断 1
			SDA_4	I/O	I2C 接口的数据线
			MISO_4	I/O	SPI 主机输入从机输出
22	25	17	P3.4	I/O	标准 IO 口
			T0	I	定时器 0 外部时钟输入
			T1CLKO	O	定时器 1 时钟分频输出
			MOSI_4	I/O	SPI 主机输出从机输入
			CMPO	O	比较器输出
23	26	18	P3.5	I/O	标准 IO 口
			T1	I	定时器 1 外部时钟输入
			T0CLKO	O	定时器 0 时钟分频输出
			SS_4	I	SPI 的从机选择脚（主机为输出）
24	27	19	P3.6	I/O	标准 IO 口
			INT2	I	外部中断 2
			WR_2	O	外部总线的写信号线
			RxD_2	I	串口 1 的接收脚
			CMP-	I	比较器负极输入

编号			名称	类型	说明
LQFP44	PDIP40	LQFP32			
25	28	20	P3.7	I/O	标准 IO 口
			INT3	I	外部中断 3
			RD_2	O	外部总线的读信号线
			TxD_2	O	串口 1 的发送脚
			CMP+	I	比较器正极输入
26	29		P4.1	I/O	标准 IO 口
			ALE	O	地址锁存信号
			CMPO_2	O	比较器输出
27	30	21	P2.0	I/O	标准 IO 口
			A8	I	地址总线
			RSTSV	-	端口的初始电平可 ISP 下载时配置
28			P4.2	I/O	标准 IO 口
			RD_3	O	外部总线的读信号线
			TxD2_2	O	串口 2 的发送脚
29	31	22	P2.1	I/O	标准 IO 口
			A9	I	地址总线
30	32	23	P2.2	I/O	标准 IO 口
			A10	I	地址总线
			SS_2	I	SPI 的从机选择脚 (主机为输出)
31	33	24	P2.3	I/O	标准 IO 口
			A11	I	地址总线
			MOSI_2	I/O	SPI 主机输出从机输入
32	34	25	P2.4	I/O	标准 IO 口
			A12	I	地址总线
			MISO_2	I/O	SPI 主机输入从机输出
			SDA_2	I/O	I2C 接口的数据线
33	35	26	P2.5	I/O	标准 IO 口
			A13	I	地址总线
			SCLK_2	I/O	SPI 的时钟脚
			SCL_2	I/O	I2C 的时钟线
34	36	27	P2.6	I/O	标准 IO 口
			A14	I	地址总线
35	37	28	P2.7	I/O	标准 IO 口
			A15	I	地址总线

编号			名称	类型	说明
LQFP44	PDIP40	LQFP32			
36	38	29	P0.0	I/O	标准 IO 口
			AD0	I	地址总线
37	39	30	P0.1	I/O	标准 IO 口
			AD1	I	地址总线
38	40	31	P0.2	I/O	标准 IO 口
			AD2	I	地址总线
39			P4.3	I/O	标准 IO 口
			WR	O	外部总线的写信号线
40	1	32	P0.3	I/O	标准 IO 口
			AD3	I	地址总线
41	2		P0.4	I/O	标准 IO 口
			AD4	I	地址总线
			T3	I	定时器 3 外部时钟输入
42	3		P0.5	I/O	标准 IO 口
			AD5	I	地址总线
			T3CLKO	O	定时器 3 时钟分频输出
43	4		P0.6	I/O	标准 IO 口
			AD6	I	地址总线
			T4	I	定时器 4 外部时钟输入
44	5		P0.7	I/O	标准 IO 口
			AD7	I	地址总线
			T4CLKO	O	定时器 4 时钟分频输出
1	6	1	P1.0	I/O	标准 IO 口
			RxD2	I	串口 2 的接收脚

3.3 功能脚切换

STC8 系列单片机的特殊外设串口 1、串口 2、串口 3、串口 4、SPI、PCA、PWM、I²C 以及总线控制脚可以在多个 I/O 直接进行切换，以实现一个外设当作多个设备进行分时复用。

3.3.1 功能脚切换相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
BUS_SPEED	总线速度控制寄存器	A1H	RW_S[1:0]						SPEED[1:0]		00xx,xx00
P_SW1	外设端口切换寄存器 1	A2H	S1_S[1:0]		CCP_S[1:0]		SPI_S[1:0]		0	-	nn00,0000x
P_SW2	外设端口切换寄存器 2	BAH	EAXFR	CAN_S	I2C_S[1:0]		CMPO_S	S4_S	S3_S	S2_S	0x00,0000

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
PWM0CR	PWM0 控制寄存器	FF04H	ENC0O	C0INI	-	C0_S[1:0]	EC0I	EC0T2SI	EC0T1SI	00x0,0000	
PWM1CR	PWM1 控制寄存器	FF14H	ENC1O	C1INI	-	C1_S[1:0]	EC1I	EC1T2SI	EC1T1SI	00x0,0000	
PWM2CR	PWM2 控制寄存器	FF24H	ENC2O	C2INI	-	C2_S[1:0]	EC2I	EC2T2SI	EC2T1SI	00x0,0000	
PWM3CR	PWM3 控制寄存器	FF34H	ENC3O	C3INI	-	C3_S[1:0]	EC3I	EC3T2SI	EC3T1SI	00x0,0000	
PWM4CR	PWM4 控制寄存器	FF44H	ENC4O	C4INI	-	C4_S[1:0]	EC4I	EC4T2SI	EC4T1SI	00x0,0000	
PWM5CR	PWM5 控制寄存器	FF54H	ENC5O	C5INI	-	C5_S[1:0]	EC5I	EC5T2SI	EC5T1SI	00x0,0000	
PWM6CR	PWM6 控制寄存器	FF64H	ENC6O	C6INI	-	C6_S[1:0]	EC6I	EC6T2SI	EC6T1SI	00x0,0000	
PWM7CR	PWM7 控制寄存器	FF74H	ENC7O	C7INI	-	C7_S[1:0]	EC7I	EC7T2SI	EC7T1SI	00x0,0000	
CKSEL	时钟选择寄存器	FE00H		MCLKODIV[3:0]		MCLKO_S	-	MCKSEL[1:0]		0000,0000	

总线速度控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
BUS_SPEED	A1H	RW_S[1:0]						SPEED[1:0]	

RW_S[1:0]: 外部总线 RD/WR 控制线选择位

RW_S[1:0]	RD	WR
00	P4.4	P4.3
01	P3.7	P3.6
10	P4.2	P4.0
11	保留	

外设端口切换控制寄存器 1

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW1	A2H	S1_S[1:0]		CCP_S[1:0]		SPI_S[1:0]		0	-

S1_S[1:0]: 串口 1 功能脚选择位

S1_S[1:0]	RxD	TxD
00	P3.0	P3.1
01	P3.6	P3.7
10	P1.6	P1.7
11	P4.3	P4.4

CCP_S[1:0]: PCA 功能脚选择位

CCP_S[1:0]	ECI	CCP0	CCP1	CCP2	CCP3
00	P1.2	P1.7	P1.6	P1.5	P1.4
01	P2.2	P2.3	P2.4	P2.5	P2.6
10	P7.4	P7.0	P7.1	P7.2	P7.3
11	P3.5	P3.3	P3.2	P3.1	P3.0

SPI_S[1:0]: SPI 功能脚选择位

SPI_S[1:0]	SS	MOSI	MISO	SCLK
00	P1.2	P1.3	P1.4	P1.5
01	P2.2	P2.3	P2.4	P2.5
10	P7.4	P7.5	P7.6	P7.7
11	P3.5	P3.4	P3.3	P3.2

外设端口切换控制寄存器 2

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW2	BAH	EAXFR	CAN_S	I2C_S[1:0]	CMPO_S	S4_S	S3_S	S2_S	

I2C_S[1:0]: I²C 功能脚选择位

I2C_S[1:0]	SCL	SDA
00	P1.5	P1.4
01	P2.5	P2.4
10	P7.7	P7.6
11	P3.2	P3.3

CMPO_S: 比较器输出脚选择位

CMPO_S	CMPO
0	P3.4
1	P4.1

S4_S: 串口 4 功能脚选择位

S4_S	RxD4	TxD4
0	P0.2	P0.3
1	P5.2	P5.3

S3_S: 串口 3 功能脚选择位

S3_S	RxD3	TxD3
0	P0.0	P0.1
1	P5.0	P5.1

S2_S: 串口 2 功能脚选择位

S2_S	RxD2	TxD2
0	P1.0	P1.1
1	P4.0	P4.2

时钟选择寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CKSEL	FE00H	MCLKODIV[3:0]				MCLKO_S	-	MCKSEL[1:0]	

MCLKO_S: 主时钟输出脚选择位

MCLKO_S	MCLKO
0	P5.4
1	P1.6

增强型 PWM 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWM0CR	FF04H	ENC0O	C0INI	-	C0_S[1:0]		EC0I	EC0T2SI	EC0T1SI
PWM1CR	FF14H	ENC1O	C1INI	-	C1_S[1:0]		EC1I	EC1T2SI	EC1T1SI
PWM2CR	FF24H	ENC2O	C2INI	-	C2_S[1:0]		EC2I	EC2T2SI	EC2T1SI
PWM3CR	FF34H	ENC3O	C3INI	-	C3_S[1:0]		EC3I	EC3T2SI	EC3T1SI
PWM4CR	FF44H	ENC4O	C4INI	-	C4_S[1:0]		EC4I	EC4T2SI	EC4T1SI
PWM5CR	FF54H	ENC5O	C5INI	-	C5_S[1:0]		EC5I	EC5T2SI	EC5T1SI
PWM6CR	FF64H	ENC6O	C6INI	-	C6_S[1:0]		EC6I	EC6T2SI	EC6T1SI
PWM7CR	FF74H	ENC7O	C7INI	-	C7_S[1:0]		EC7I	EC7T2SI	EC7T1SI

C0_S[1:0]: 增强型 PWM 通道 0 输出脚选择位

C0_S[1:0]	PWM0
00	P2.0
01	P1.0
10	P6.0
11	保留

C1_S[1:0]: 增强型 PWM 通道 1 输出脚选择位

C1_S[1:0]	PWM1
00	P2.1
01	P1.1
10	P6.1
11	保留

C2_S[1:0]: 增强型 PWM 通道 2 输出脚选择位

C2_S[1:0]	PWM2
00	P2.2
01	P1.2
10	P6.2
11	保留

C3_S[1:0]: 增强型 PWM 通道 3 输出脚选择位

C3_S[1:0]	PWM3
00	P2.3
01	P1.3
10	P6.3
11	保留

C4_S[1:0]: 增强型 PWM 通道 4 输出脚选择位

C4_S[1:0]	PWM4
00	P2.4
01	P1.4
10	P6.4

11	保留
----	----

C5_S[1:0]: 增强型 PWM 通道 5 输出脚选择位

C5_S[1:0]	PWM5
00	P2.5
01	P1.5
10	P6.5
11	保留

C6_S[1:0]: 增强型 PWM 通道 6 输出脚选择位

C6_S[1:0]	PWM6
00	P2.6
01	P1.6
10	P6.6
11	保留

C7_S[1:0]: 增强型 PWM 通道 7 输出脚选择位

C7_S[1:0]	PWM7
00	P2.7
01	P1.7
10	P6.7
11	保留

3.4 范例程序

3.4.1 串口 1 切换

汇编代码

<i>P_SW1</i>	<i>DATA</i>	<i>0A2H</i>	
<i>ORG</i>	<i>0000H</i>		
<i>LJMP</i>	<i>MAIN</i>		
<i>ORG</i>	<i>0100H</i>		
<i>MAIN:</i>			
<i>MOV</i>	<i>SP, #3FH</i>		
<i>MOV</i>	<i>P_SW1, #00H</i>	; <i>RXD/P3.0, TXD/P3.1</i>	
;	<i>MOV</i>	<i>P_SW1, #40H</i>	; <i>RXD_2/P3.6, TXD_2/P3.7</i>
;	<i>MOV</i>	<i>P_SW1, #80H</i>	; <i>RXD_3/P1.6, TXD_3/P1.7</i>
;	<i>MOV</i>	<i>P_SW1, #0C0H</i>	; <i>RXD_4/P4.3, TXD_4/P4.4</i>
<i>SJMP</i>	<i>\$</i>		
 <i>END</i>			

C 语言代码

```
#include "reg51.h"

sfr P_SW1 = 0xa2;

void main()
{
    P_SW1 = 0x00;           //RXD/P3.0, TXD/P3.1
}
```

```

//      P_SW1 = 0x40;          //RXD_2/P3.6, TXD_2/P3.7
//      P_SW1 = 0x80;          //RXD_3/P1.6, TXD_3/P1.7
//      P_SW1 = 0xc0;          //RXD_4/P4.3, TXD_4/P4.4

    while (1);
}

```

3.4.2 串口 2 切换

汇编代码

P_SW2	DATA	0BAH
--------------	-------------	-------------

```

ORG      0000H
LJMP    MAIN

ORG      0100H
MAIN:
MOV     SP, #3FH

MOV     P_SW2,#00H           ;RXD2/P1.0, TXD2/P1.1
;      MOV     P_SW2,#01H           ;RXD2_2/P4.0, TXD2_2/P4.2

SJMP    $

END

```

C 语言代码

```

#include "reg51.h"

sfr P_SW2 = 0xba;

void main()
{
    P_SW2 = 0x00;             //RXD2/P1.0, TXD2/P1.1
//    P_SW2 = 0x01;             //RXD2_2/P4.0, TXD2_2/P4.2

    while (1);
}

```

3.4.3 串口 3 切换

汇编代码

P_SW2	DATA	0BAH
--------------	-------------	-------------

```

ORG      0000H
LJMP    MAIN

ORG      0100H
MAIN:
MOV     SP, #3FH

MOV     P_SW2,#00H           ;RXD3/P0.0, TXD3/P0.1
;      MOV     P_SW2,#02H           ;RXD3_2/P5.0, TXD3_2/P5.1

SJMP    $


```

END

C 语言代码

```
#include "reg51.h"

sfr P_SW2 = 0xba;

void main()
{
    P_SW2 = 0x00;           // RXD3/P0.0, TXD3/P0.1
//    P_SW2 = 0x02;           // RXD3_2/P5.0, TXD3_2/P5.1

    while (1);
}
```

3.4.4 串口 4 切换**汇编代码**

```
P_SW2    DATA    0BAH

        ORG      0000H
        LJMP    MAIN

        ORG      0100H
MAIN:
        MOV      SP, #3FH

        MOV      P_SW2, #00H          ;RXD4/P0.2, TXD4/P0.3
;        MOV      P_SW2, #04H          ;RXD4_2/P5.2, TXD4_2/P5.3

        SJMP    $

        END
```

C 语言代码

```
#include "reg51.h"

sfr P_SW2 = 0xba;

void main()
{
    P_SW2 = 0x00;           //RXD4/P0.2, TXD4/P0.3
//    P_SW2 = 0x04;           //RXD4_2/P5.2, TXD4_2/P5.3

    while (1);
}
```

3.4.5 SPI切换**汇编代码**

```
P_SW1    DATA    0A2H

        ORG      0000H
```

```

LJMP    MAIN

ORG    0100H

MAIN:
MOV    SP, #3FH

MOV    P_SW1,#00H          ;SS/P1.2, MOSI/P1.3, MISO/P1.4, SCLK/P1.5
;      MOV    P_SW1,#04H          ;SS_2/P2.2, MOSI_2/P2.3, MISO_2/P2.4, SCLK_2/P2.5
;      MOV    P_SW1,#08H          ;SS_3/P7.4, MOSI_3/P7.5, MISO_3/P7.6, SCLK_3/P7.7
;      MOV    P_SW1,#0CH          ;SS_4/P3.5, MOSI_4/P3.4, MISO_4/P3.3, SCLK_4/P3.2

SJMP    $

END

```

C 语言代码

```

#include "reg51.h"

sfr P_SW1 = 0xa2;

void main()
{
    P_SW1 = 0x00;           //SS/P1.2, MOSI/P1.3, MISO/P1.4, SCLK/P1.5
//    P_SW1 = 0x04;          //SS_2/P2.2, MOSI_2/P2.3, MISO_2/P2.4, SCLK_2/P2.5
//    P_SW1 = 0x08;          //SS_3/P7.4, MOSI_3/P7.5, MISO_3/P7.6, SCLK_3/P7.7
//    P_SW1 = 0x0c;          //SS_4/P3.5, MOSI_4/P3.4, MISO_4/P3.3, SCLK_4/P3.2

    while (1);
}

```

3.4.6 PWM切换

汇编代码

```

P_SW2    DATA    0BAH
PWM0CR  EQU    0FF04H
PWM1CR  EQU    0FF14H
PWM2CR  EQU    0FF24H
PWM3CR  EQU    0FF34H
PWM4CR  EQU    0FF44H
PWM5CR  EQU    0FF54H
PWM6CR  EQU    0FF64H
PWM7CR  EQU    0FF74H

ORG    0000H
LJMP    MAIN

MAIN:
ORG    0100H

MOV    SP, #3FH

MOV    P_SW2,#80H
MOV    A,#00H          ;PWM0/P2.0
;      MOV    A,#08H          ;PWM0_2/P1.0
;      MOV    A,#10H          ;PWM0_3/P6.0
MOV    DPTR,#PWM0CR
MOVX   @DPTR,A
MOV    A,#00H          ;PWM1/P2.1

```

```

;           MOV      A,#08H          ;PWM1_2/P1.1
;           MOV      A,#10H          ;PWM1_3/P6.1
MOV      DPTR,#PWM1CR
MOVX   @DPTR,A
MOV      A,#00H          ;PWM2/P2.2
;           MOV      A,#08H          ;PWM2_2/P1.2
;           MOV      A,#10H          ;PWM2_3/P6.2
MOV      DPTR,#PWM2CR
MOVX   @DPTR,A
MOV      A,#00H          ;PWM3/P2.3
;           MOV      A,#08H          ;PWM3_2/P1.3
;           MOV      A,#10H          ;PWM3_3/P6.3
MOV      DPTR,#PWM3CR
MOVX   @DPTR,A
MOV      A,#00H          ;PWM4/P2.4
;           MOV      A,#08H          ;PWM4_2/P1.4
;           MOV      A,#10H          ;PWM4_3/P6.4
MOV      DPTR,#PWM4CR
MOVX   @DPTR,A
MOV      A,#00H          ;PWM5/P2.5
;           MOV      A,#08H          ;PWM5_2/P1.5
;           MOV      A,#10H          ;PWM5_3/P6.5
MOV      DPTR,#PWM5CR
MOVX   @DPTR,A
MOV      A,#00H          ;PWM6/P2.6
;           MOV      A,#08H          ;PWM6_2/P1.6
;           MOV      A,#10H          ;PWM6_3/P6.6
MOV      DPTR,#PWM6CR
MOVX   @DPTR,A
MOV      A,#00H          ;PWM7/P2.7
;           MOV      A,#08H          ;PWM7_2/P1.7
;           MOV      A,#10H          ;PWM7_3/P6.7
MOV      DPTR,#PWM7CR
MOVX   @DPTR,A
MOV      P_SW2,#00H

SJMP    $

```

END

C 语言代码

#include "reg51.h"

```

#define PWM0CR  (*(unsigned char volatile xdata *)0xff04)
#define PWM1CR  (*(unsigned char volatile xdata *)0xff14)
#define PWM2CR  (*(unsigned char volatile xdata *)0xff24)
#define PWM3CR  (*(unsigned char volatile xdata *)0xff34)
#define PWM4CR  (*(unsigned char volatile xdata *)0xff44)
#define PWM5CR  (*(unsigned char volatile xdata *)0xff54)
#define PWM6CR  (*(unsigned char volatile xdata *)0xff64)
#define PWM7CR  (*(unsigned char volatile xdata *)0xff74)

```

sfr P_SW2 = 0xba;

```

void main()
{
    P_SW2 = 0x80;
    PWM0CR = 0x00;           //PWM0/P2.0
}

```

```

//      PWM0CR = 0x08;          //PWM0_2/P1.0
//      PWM0CR = 0x10;          //PWM0_3/P6.0
PWM1CR = 0x00;          //PWI1/P2.1
//      PWM1CR = 0x08;          //PWM1_2/P1.1
//      PWM1CR = 0x10;          //PWM1_3/P6.1
PWM2CR = 0x00;          //PWM2/P2.2
//      PWM2CR = 0x08;          //PWM2_2/P1.2
//      PWM2CR = 0x10;          //PWM2_3/P6.2
PWM3CR = 0x00;          //PWM3/P2.3
//      PWM3CR = 0x08;          //PWM3_2/P1.3
//      PWM3CR = 0x10;          //PWM3_3/P6.3
PWM4CR = 0x00;          //PWM4/P2.4
//      PWM4CR = 0x08;          //PWM4_2/P1.4
//      PWM4CR = 0x10;          //PWM4_3/P6.4
PWM5CR = 0x00;          //PWM5/P2.5
//      PWM5CR = 0x08;          //PWM5_2/P1.5
//      PWM5CR = 0x10;          //PWM5_3/P6.5
PWM6CR = 0x00;          //PWM6/P2.6
//      PWM6CR = 0x08;          //PWM6_2/P1.6
//      PWM6CR = 0x10;          //PWM6_3/P6.6
PWM7CR = 0x00;          //PWM7/P2.7
//      PWM7CR = 0x08;          //PWM7_2/P1.7
//      PWM7CR = 0x10;          //PWM7_3/P6.7
P_SW2 = 0x00;

while (1);
}

```

3.4.7 PCA/CCP/PWM切换

汇编代码

P_SW1	DATA	0A2H
ORG	0000H	
LJMP	MAIN	
ORG	0100H	
MAIN:		
MOV	SP, #3FH	
MOV	P_SW1,#00H	
;ECI/P1.2, CCP0/P1.7, CCP1/P1.6, CCP2/P1.5, CCP3/P1.4		
;	MOV	P_SW1,#10H
;ECI_2/P2.2, CCP0_2/P2.3, CCP1_2/P2.4, CCP2_2/P2.5, CCP3_2/P2.6		
;	MOV	P_SW1,#20H
;ECI_3/P7.4, CCP0_3/P7.0, CCP1_3/P7.1, CCP2_3/P7.2, CCP3_3/P7.3		
;	MOV	P_SW1,#30H
;ECI_4/P3.5, CCP0_4/P3.3, CCP1_4/P3.2, CCP2_4/P3.1, CCP3_4/P3.0		
SJMP	\$	
END		

C 语言代码

```

#include "reg51.h"

sfr P_SW1 = 0xa2;

void main()
{
    P_SW1 = 0x00;           //ECI/P1.2, CCP0/P1.7, CCP1/P1.6, CCP2/P1.5, CCP3/P1.4
}

```

```

//      P_SW1 = 0x10;          //ECI_2/P2.2, CCP0_2/P2.3, CCP1_2/P2.4, CCP2_2/P2.5, CCP3_2/P2.6
//      P_SW1 = 0x20;          //ECI_3/P7.4, CCP0_3/P7.0, CCP1_3/P7.1, CCP2_3/P7.2, CCP3_3/P7.3
//      P_SW1 = 0x30;          //ECI_4/P3.5, CCP0_4/P3.3, CCP1_4/P3.2, CCP2_4/P3.1, CCP3_4/P3.0

    while (1);
}

```

3.4.8 I2C切换

汇编代码

P_SW2	DATA	0BAH
--------------	-------------	-------------

```

ORG      0000H
LJMP     MAIN

ORG      0100H
MAIN:
MOV      SP, #3FH

MOV      P_SW2,#00H           ;SCL/P1.5, SDA/P1.4
;      MOV      P_SW2,#10H        ;SCL_2/P2.5, SDA_2/P2.4
;      MOV      P_SW2,#20H        ;SCL_3/P7.7, SDA_3/P7.6
;      MOV      P_SW2,#30H        ;SCL_4/P3.2, SDA_4/P3.3

SJMP    $

```

C 语言代码

```

#include "reg51.h"

sfr P_SW2 = 0xa;

void main()
{
    P_SW2 = 0x00;           //SCL/P1.5, SDA/P1.4
//    P_SW2 = 0x10;          //SCL_2/P2.5, SDA_2/P2.4
//    P_SW2 = 0x20;          //SCL_3/P7.7, SDA_3/P7.6
//    P_SW2 = 0x30;          //SCL_4/P3.2, SDA_4/P3.3

    while (1);
}

```

3.4.9 比较器输出切换

汇编代码

P_SW2	DATA	0BAH
--------------	-------------	-------------

```

ORG      0000H
LJMP     MAIN

ORG      0100H
MAIN:
MOV      SP, #3FH

```

```

MOV    P_SW2,#00H          ;CMPO/P3.4
;      MOV    P_SW2,#08H          ;CMPO_2/P4.1
SJMP   $                 

END

```

C 语言代码

```

#include "reg51.h"

sfr P_SW2 = 0xba;

void main()
{
    P_SW2 = 0x00;           //CMPO/P3.4
//    P_SW2 = 0x08;          //CMPO_2/P4.1

    while (1);
}

```

3.4.10 主时钟输出切换

汇编代码

```

P_SW2  DATA  0BAH
CKSEL   EQU   0FE00H

        ORG   0000H
        LJMP  MAIN

        ORG   0100H
MAIN:
        MOV   SP, #3FH

        MOV   P_SW2,#80H
        MOV   A,#40H          ;IRC24M/4 output via MCLKO/P5.4
;        MOV   A,#48H          ;IRC24M/4 output via MCLKO_2/P1.6
;        MOV   A,#0E8H          ;IRC24M/128 output via MCLKO_2/P1.6
        MOV   DPTR,#CKSEL
        MOVX  @DPTR,A
        MOV   P_SW2,#00H

        SJMP  $

```

C 语言代码

```

#include "reg51.h"

#define CKSEL  (*(unsigned char volatile xdata *)0xfe00)

sfr P_SW2 = 0xba;

void main()
{
    P_SW2 = 0x80;           //IRC24M/4 output via MCLKO/P5.4
    CKSEL = 0x40;

```

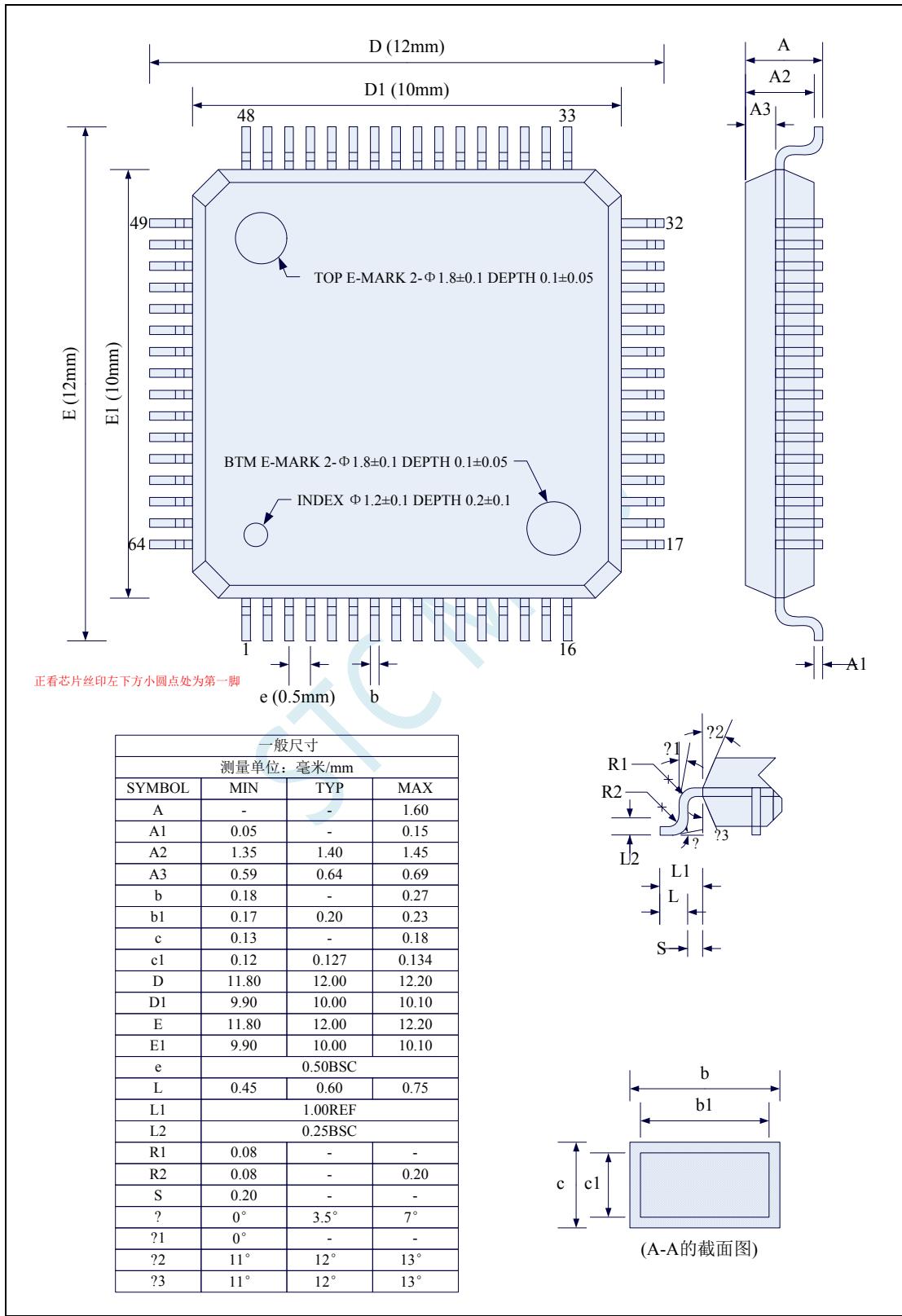
```
//      CKSEL = 0x48;          //IRC24M/4 output via MCLKO_2/P1.6
//      CKSEL = 0xe8;          //IRC24M/128 output via MCLKO_2/P1.6
P_SW2 = 0x00;

while (1);
}
```

STCMCU

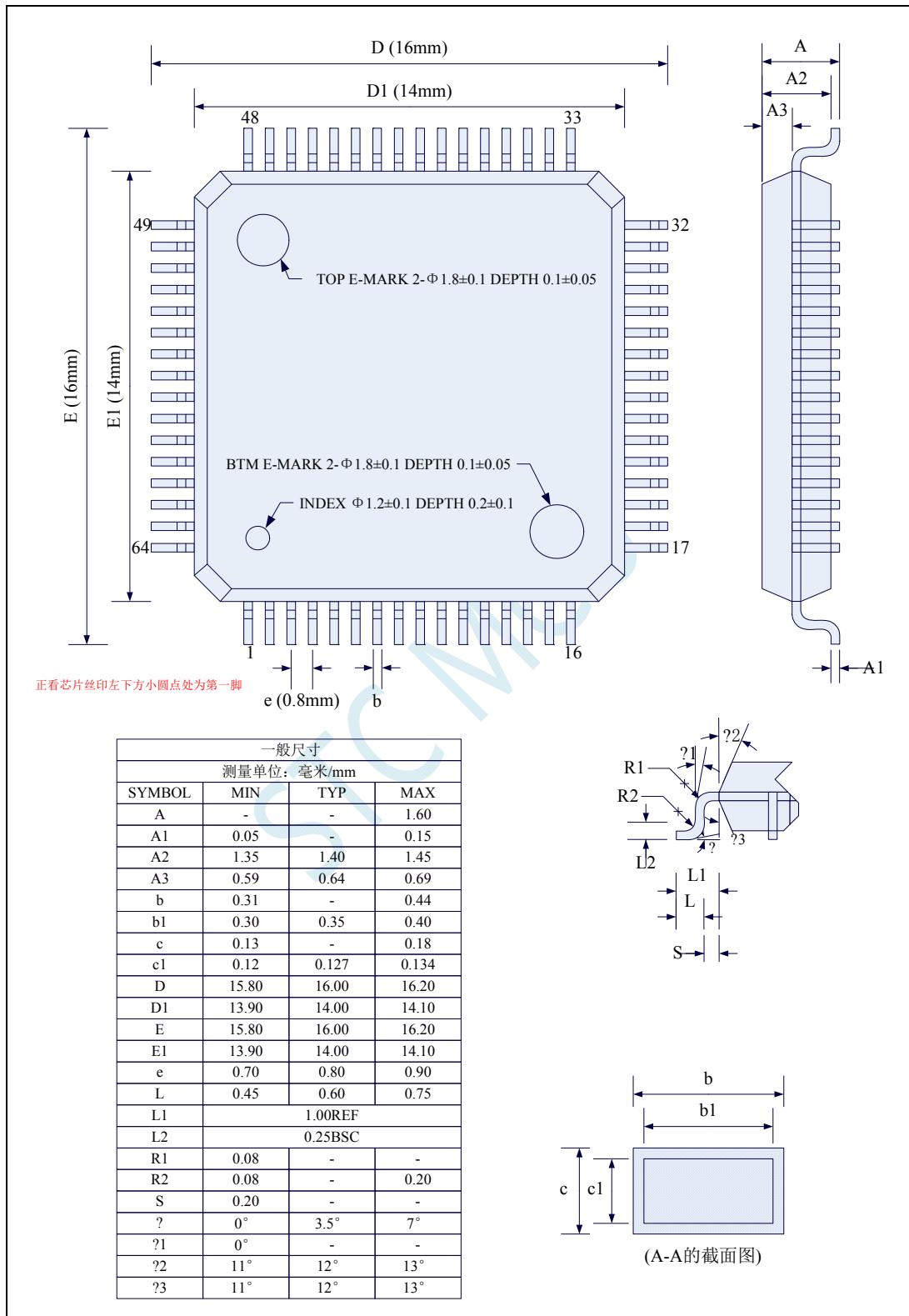
4 封装尺寸图

4.1 LQFP64S封装尺寸图 (12mm*12mm)

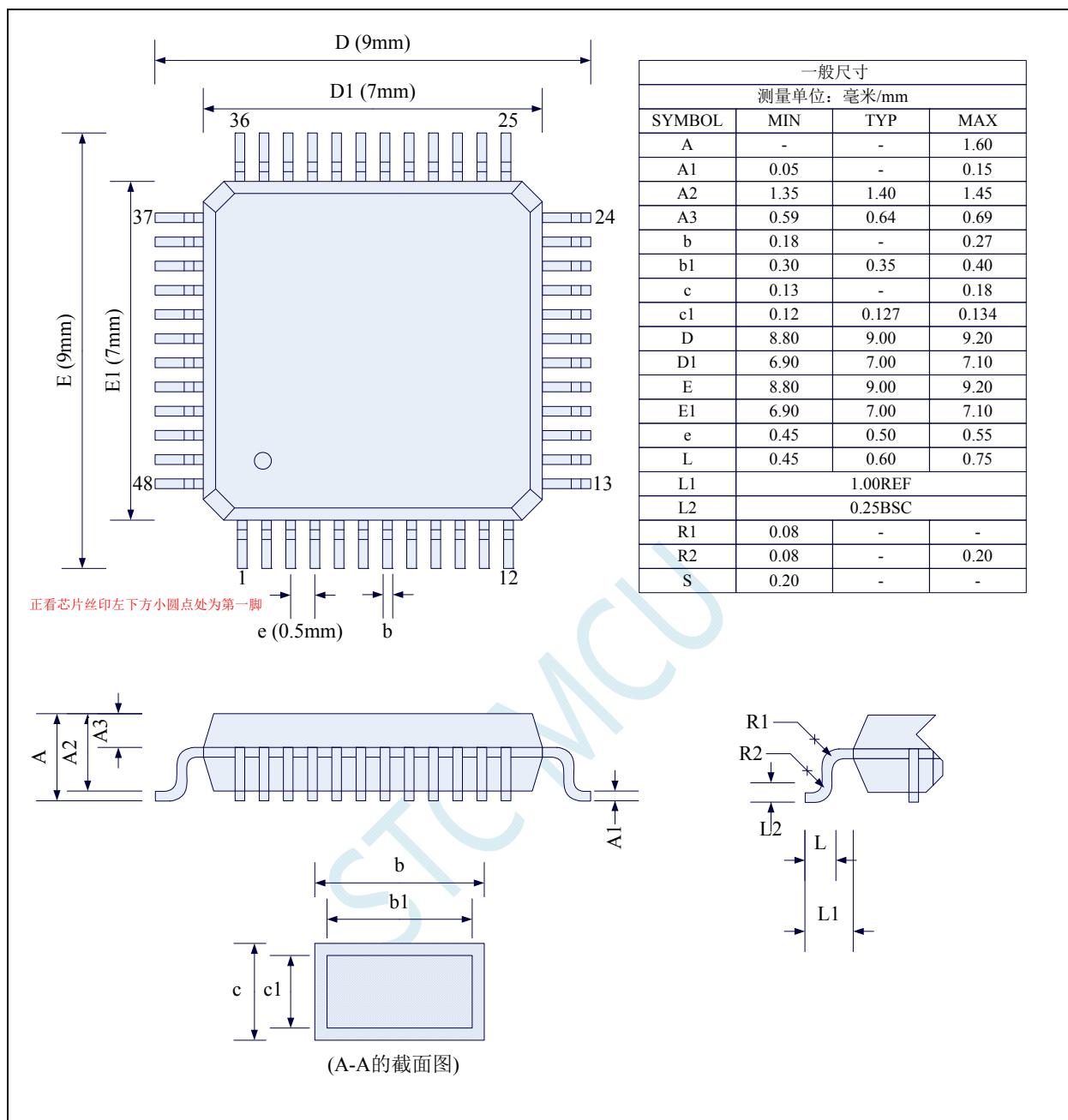


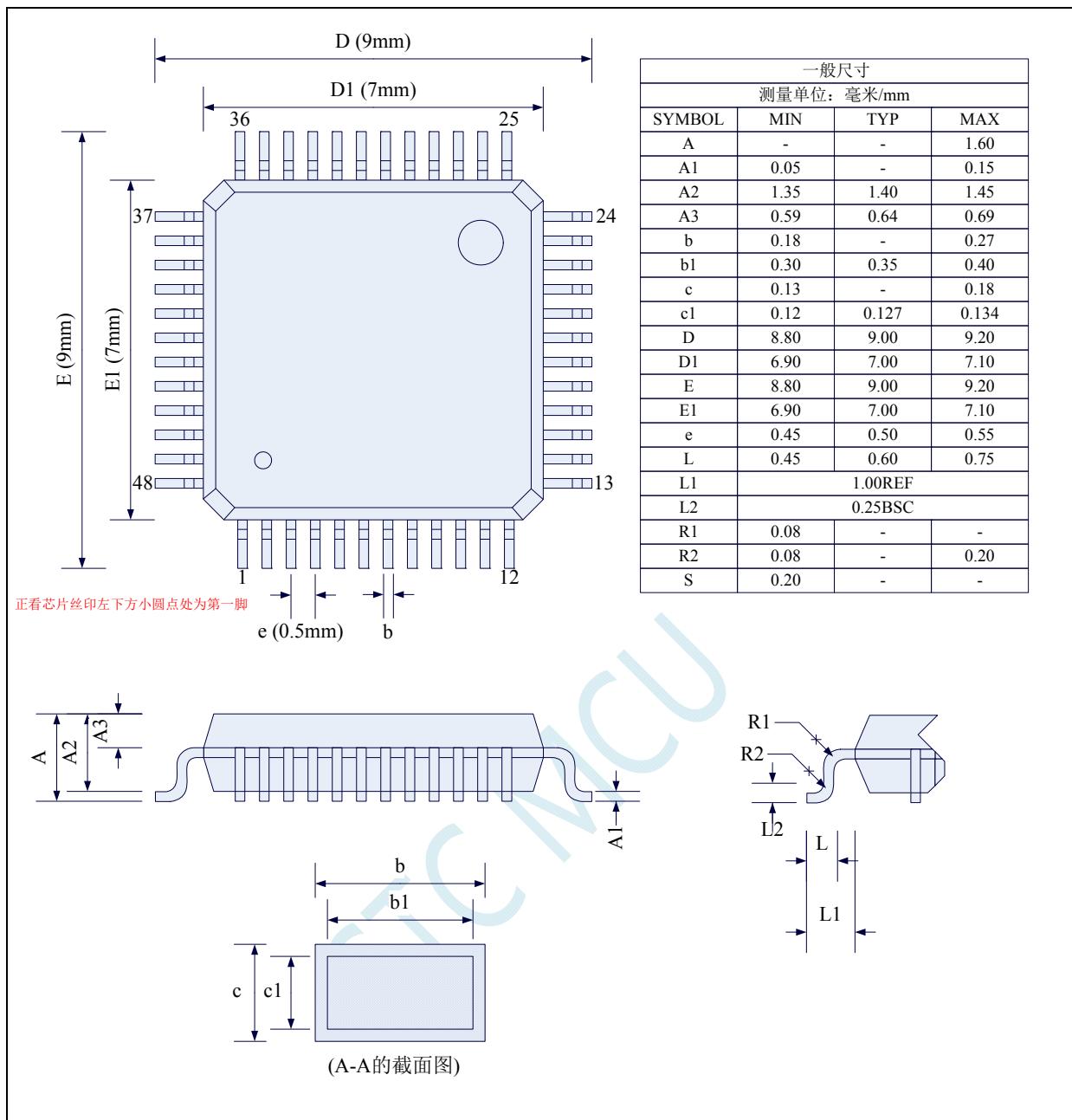
4.2 LQFP64L封装尺寸图 (16mm*16mm)

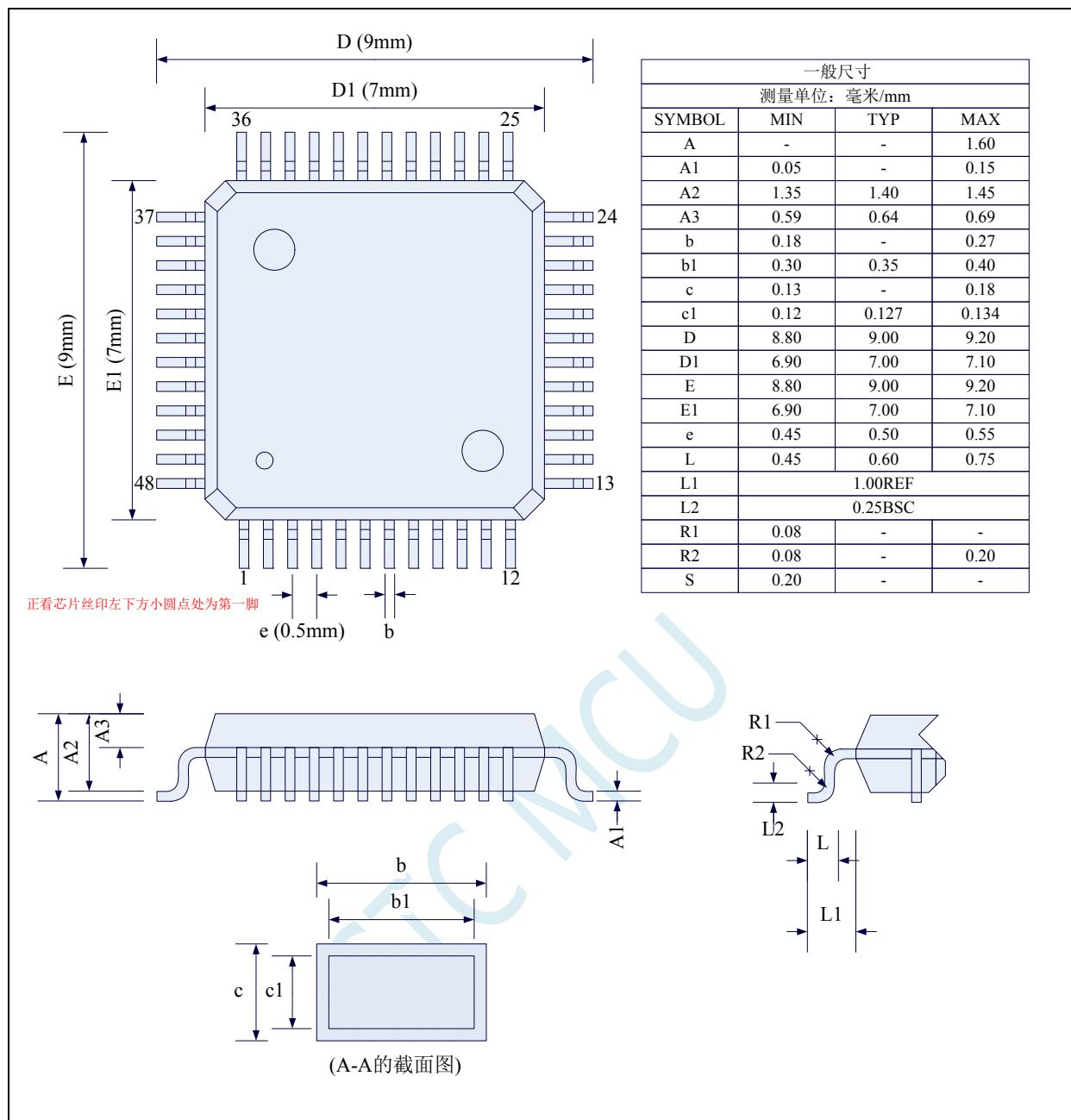
STC8 系列暂无此封装



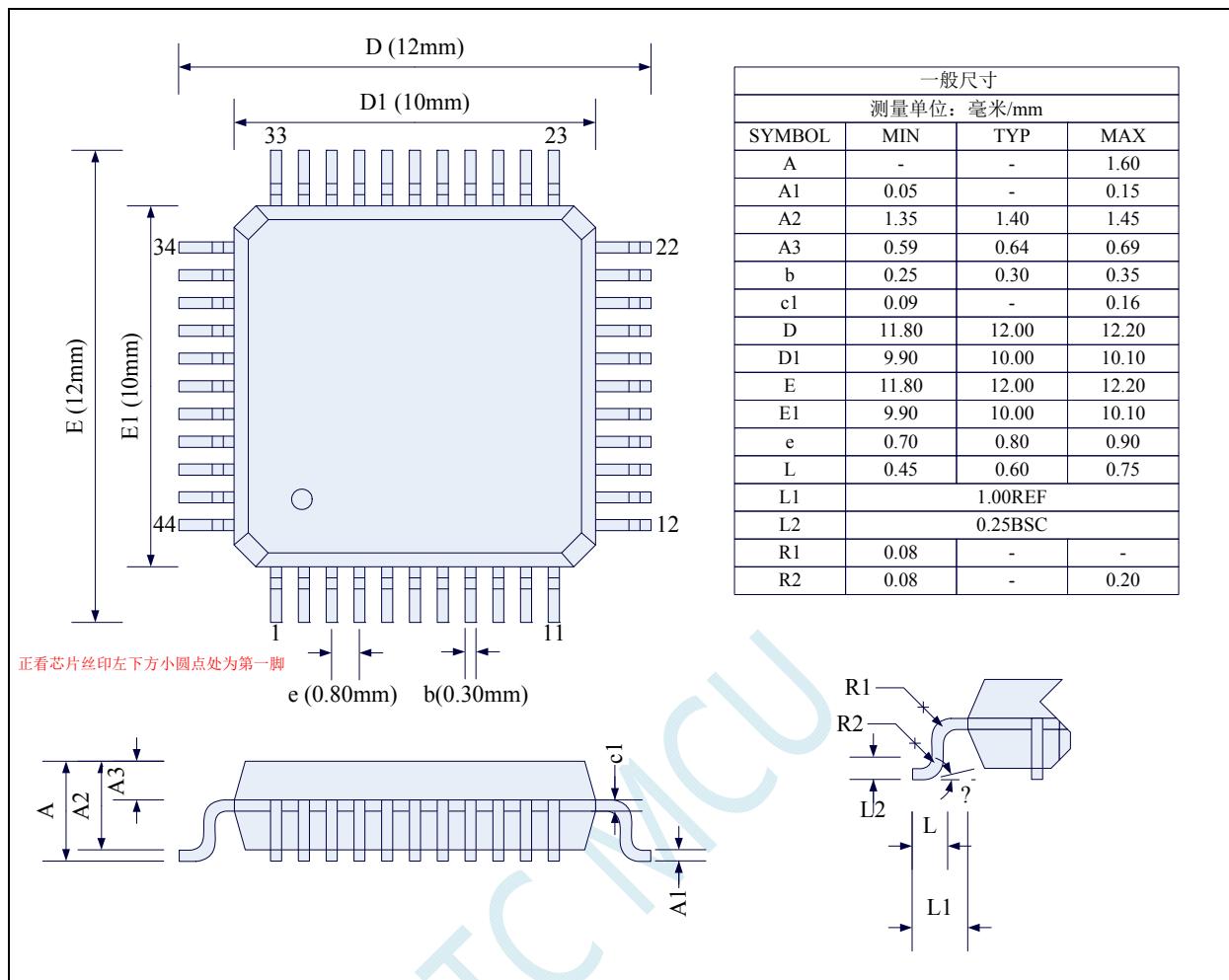
4.3 LQFP48 封装尺寸图 (9mm*9mm)

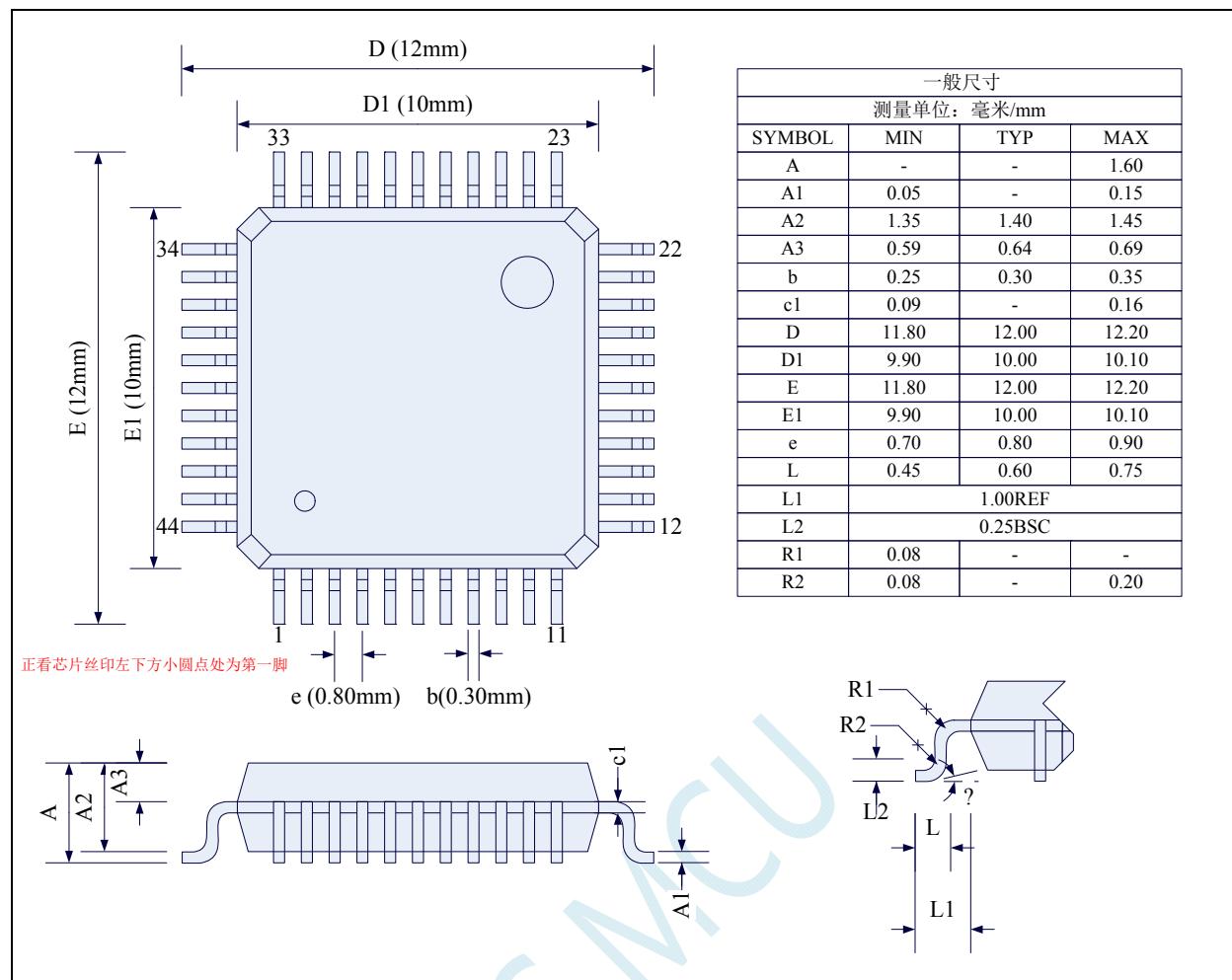


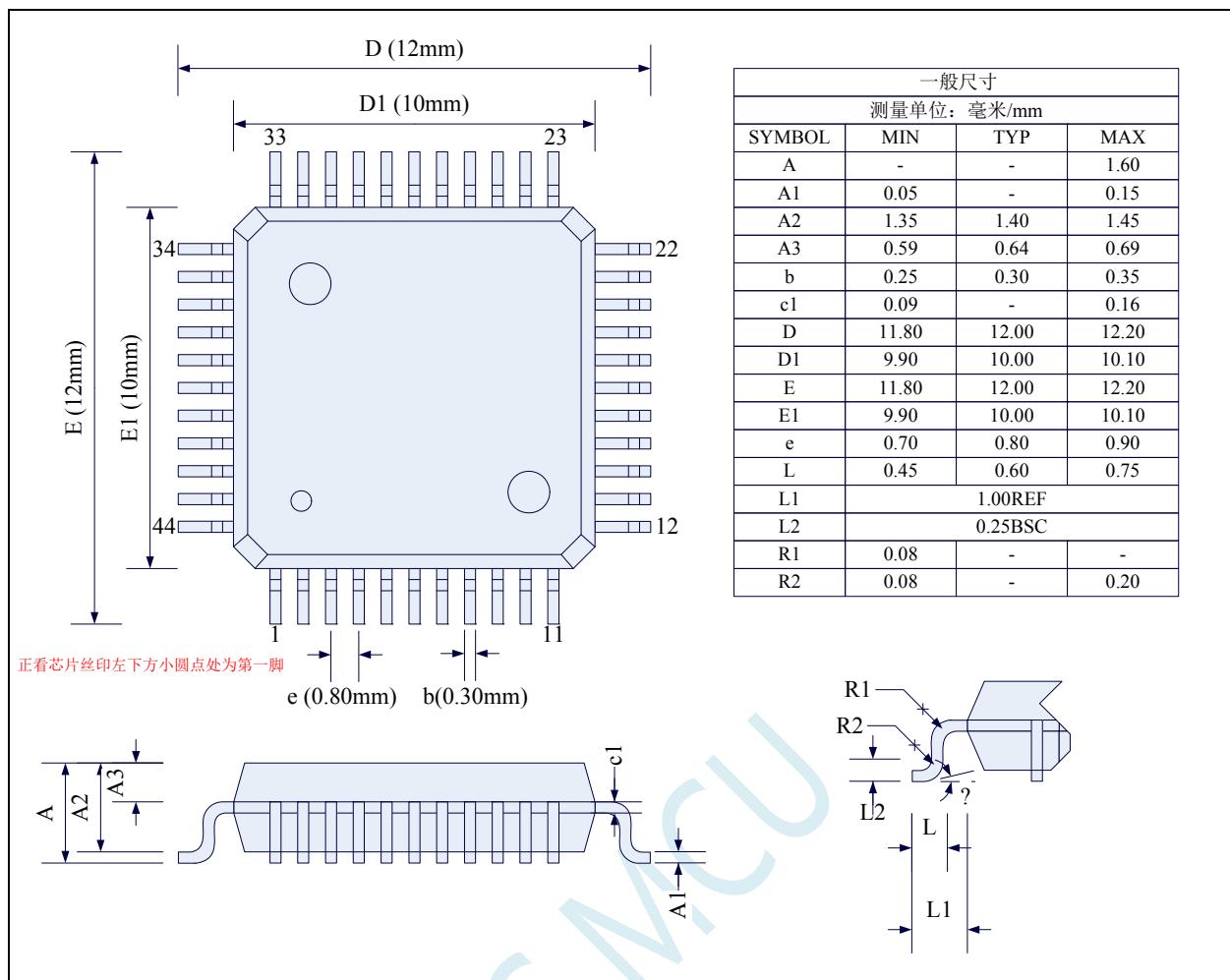




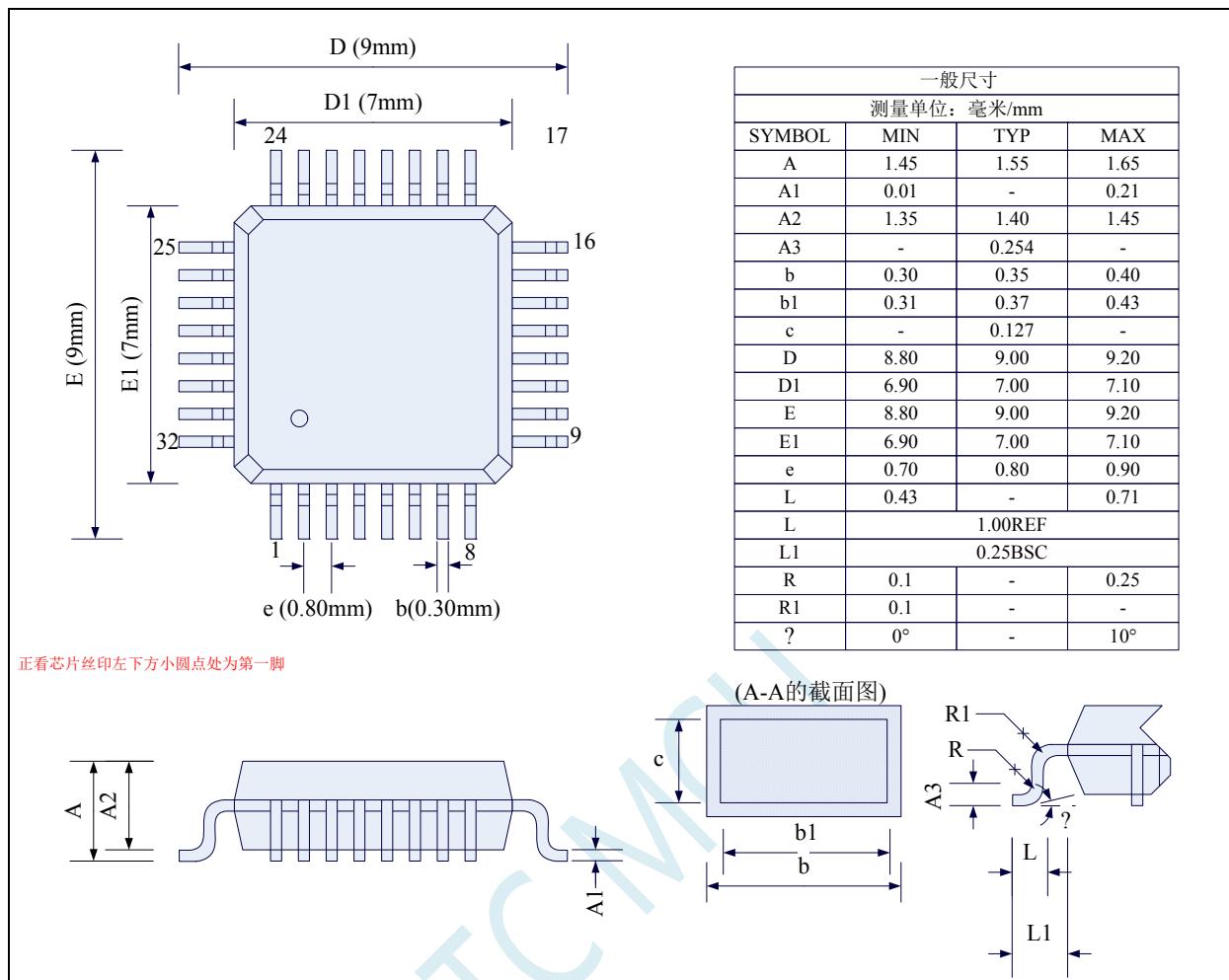
4.4 LQFP44 封装尺寸图 (12mm*12mm)



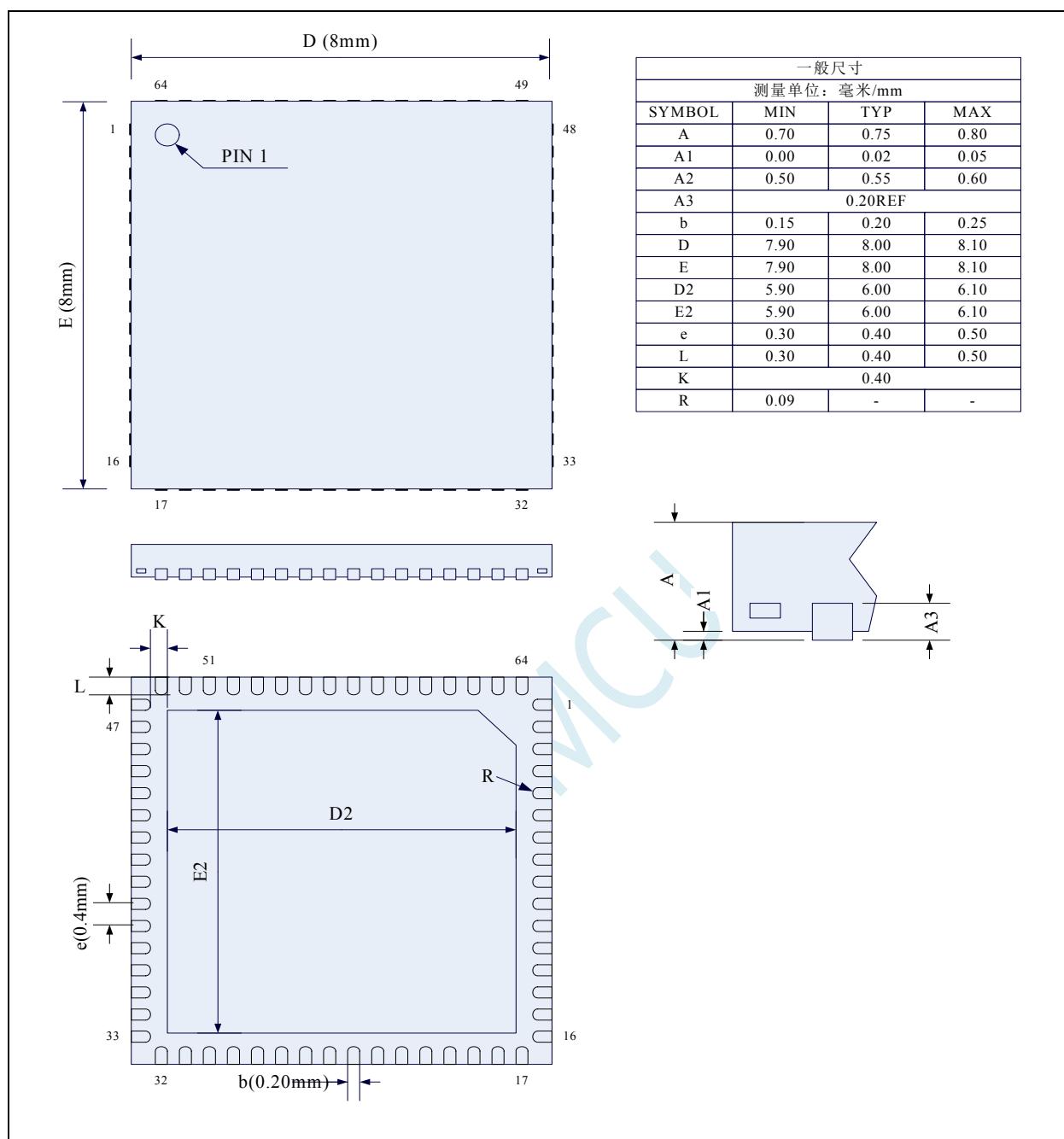




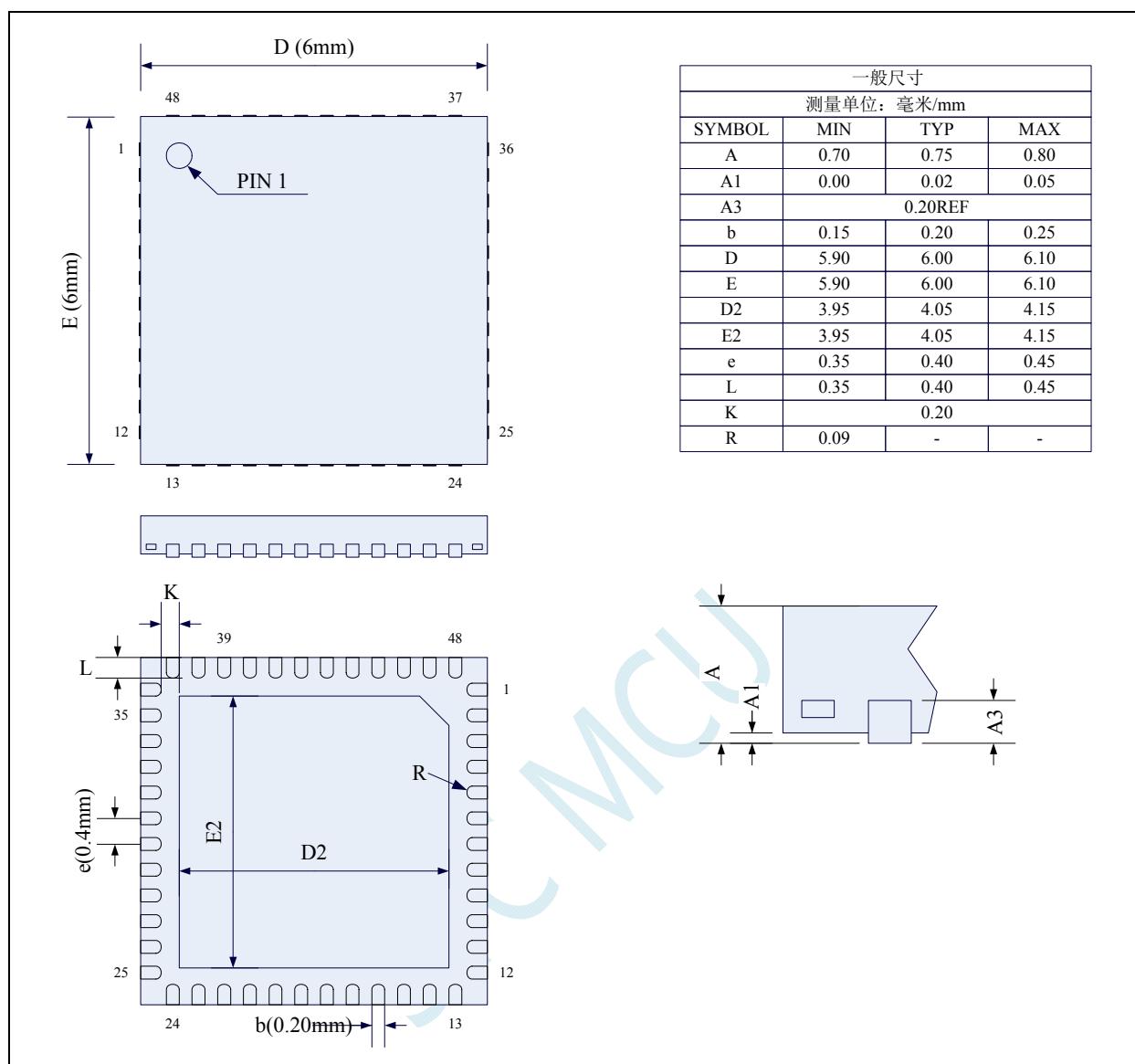
4.5 LQFP32 封装尺寸图 (9mm*9mm)



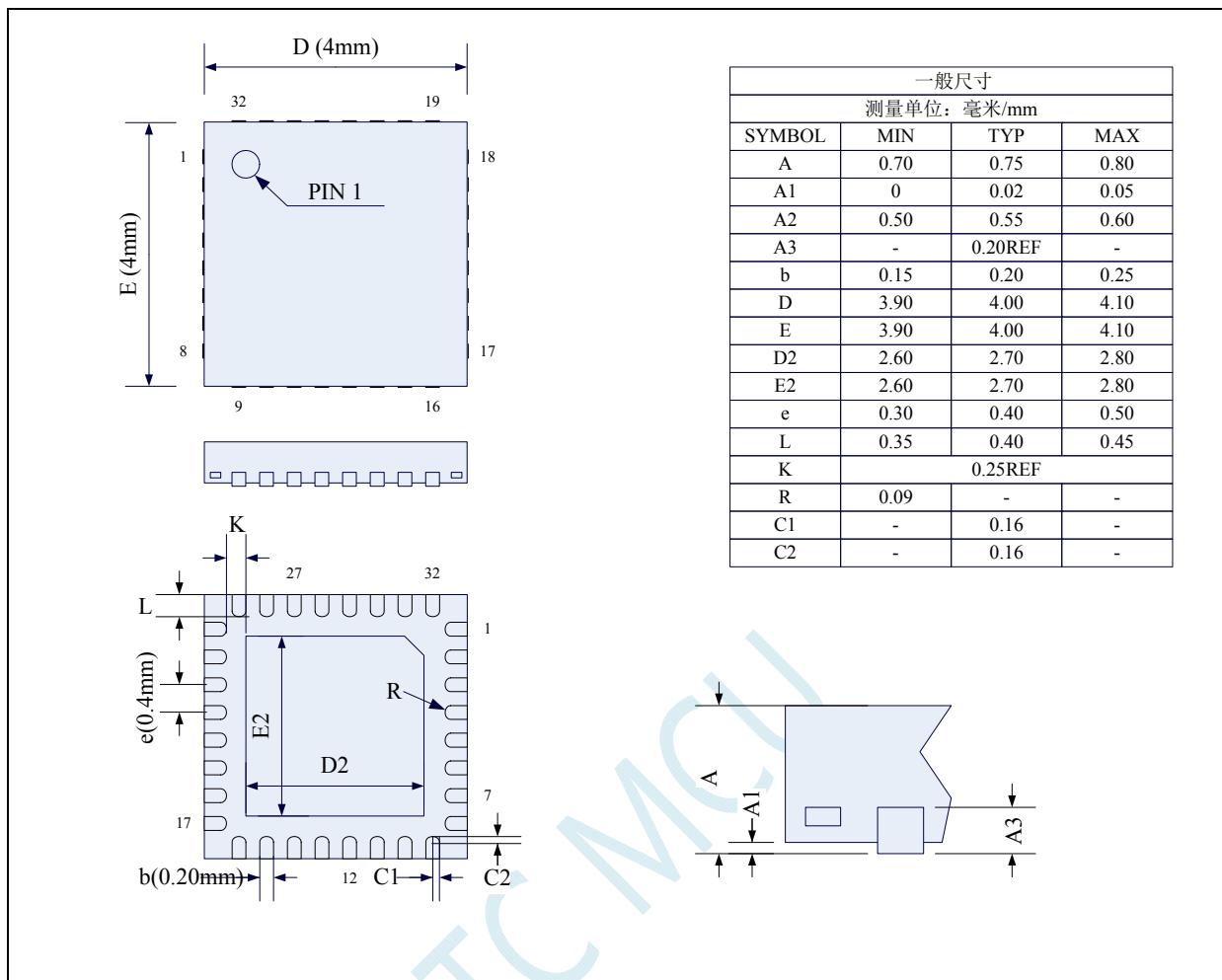
4.6 QFN64 封装尺寸图 (8mm*8mm)



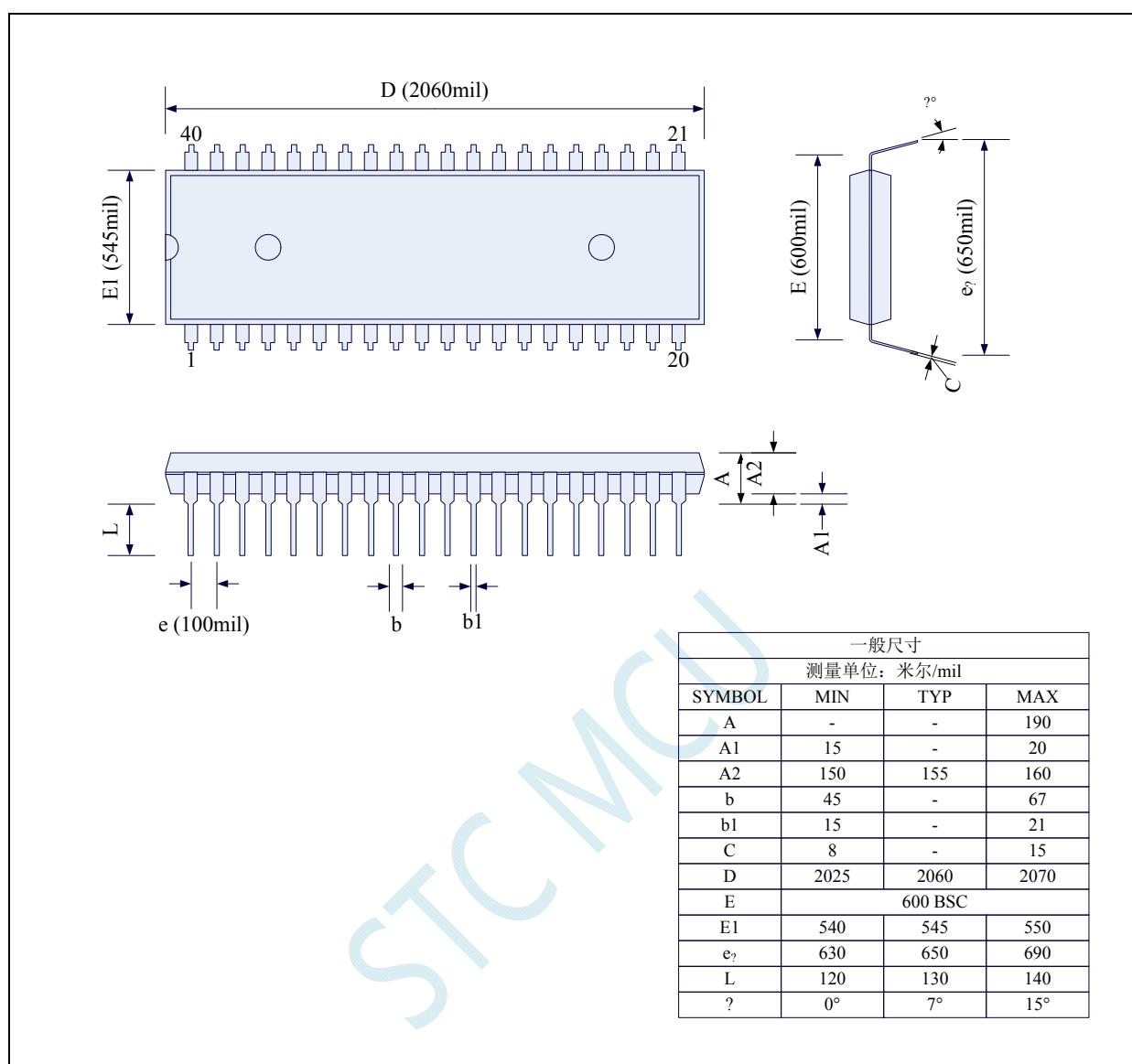
4.7 QFN48 封装尺寸图 (6mm*6mm)



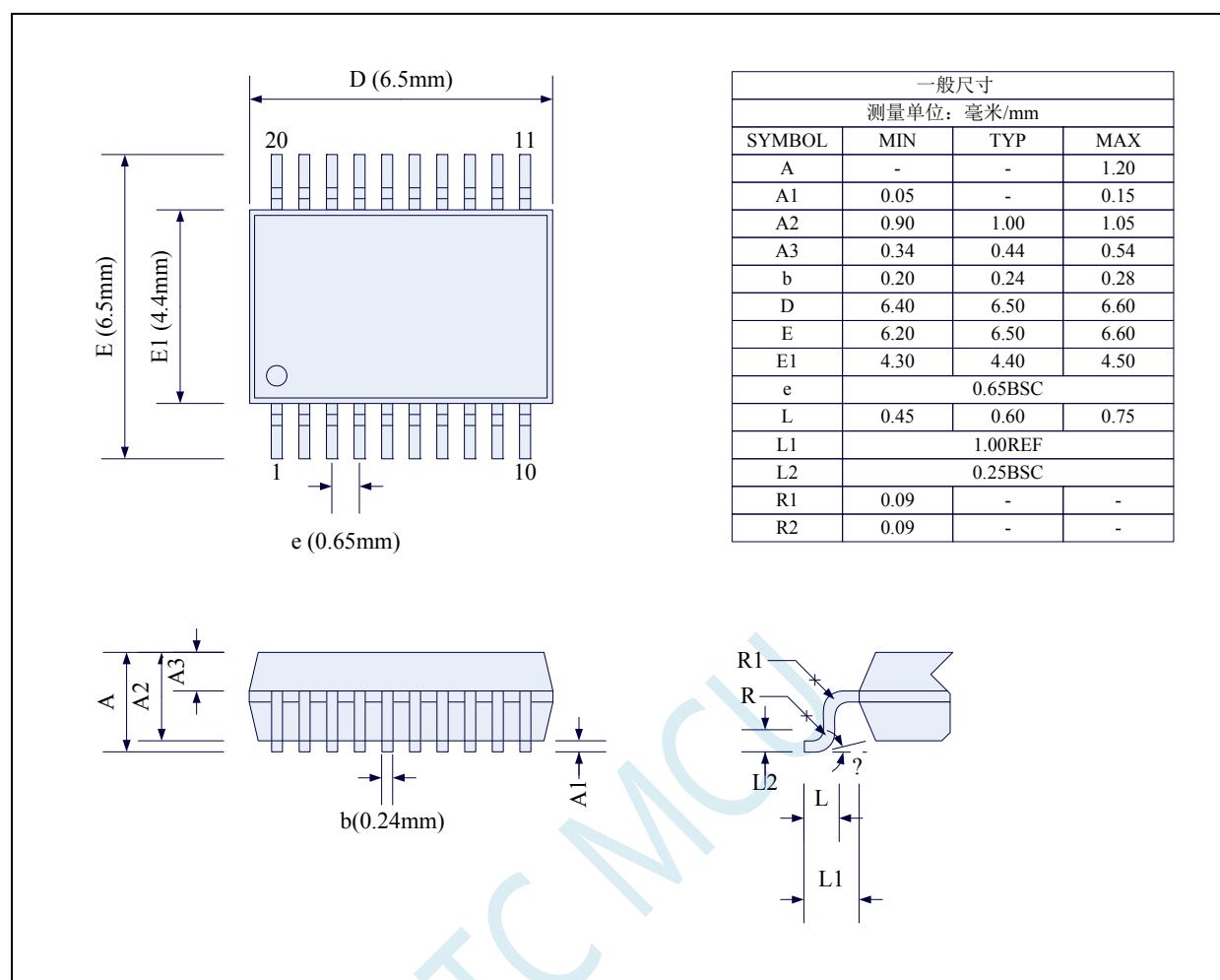
4.8 QFN32 封装尺寸图 (4mm*4mm)



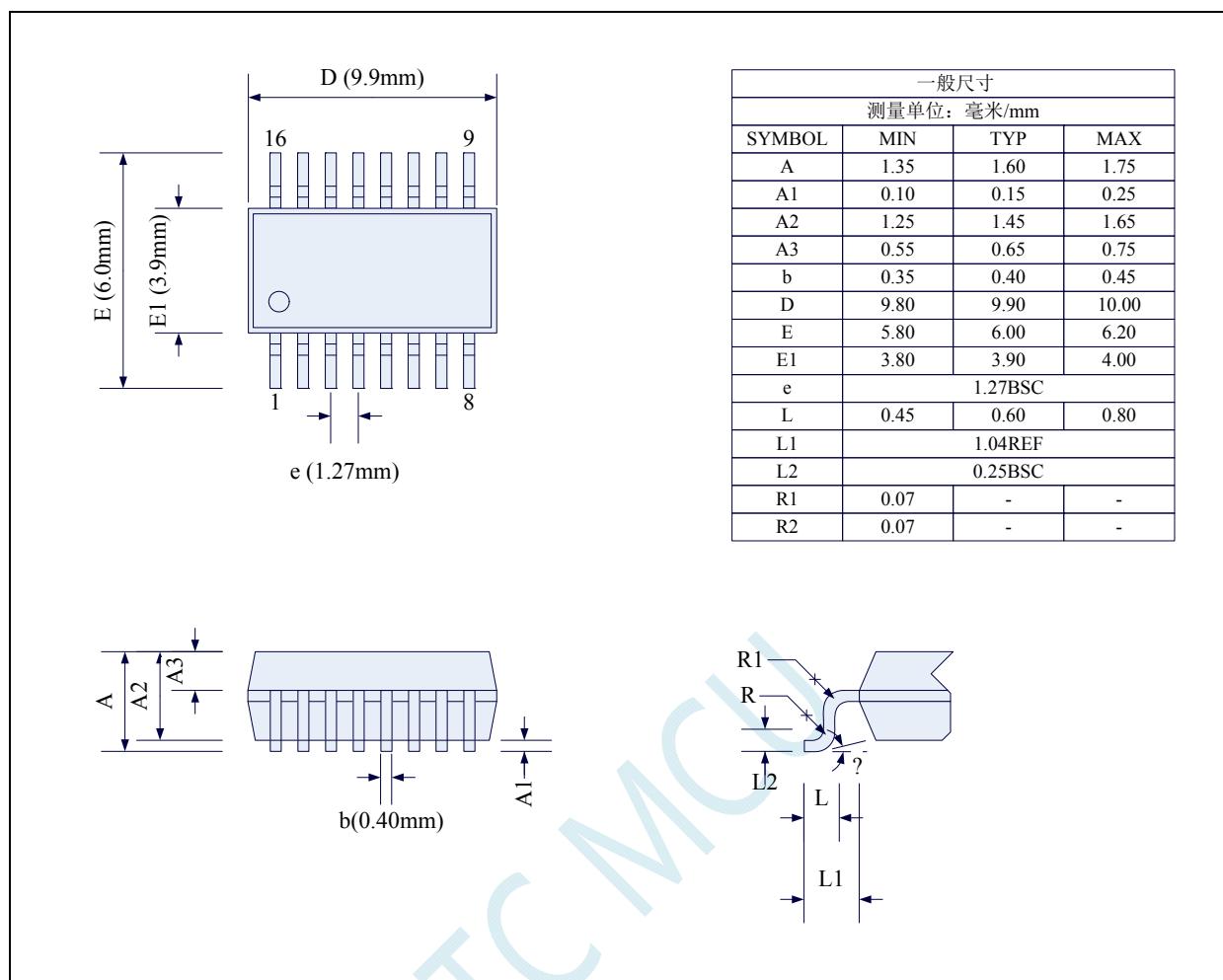
4.9 PDIP40 封装尺寸图



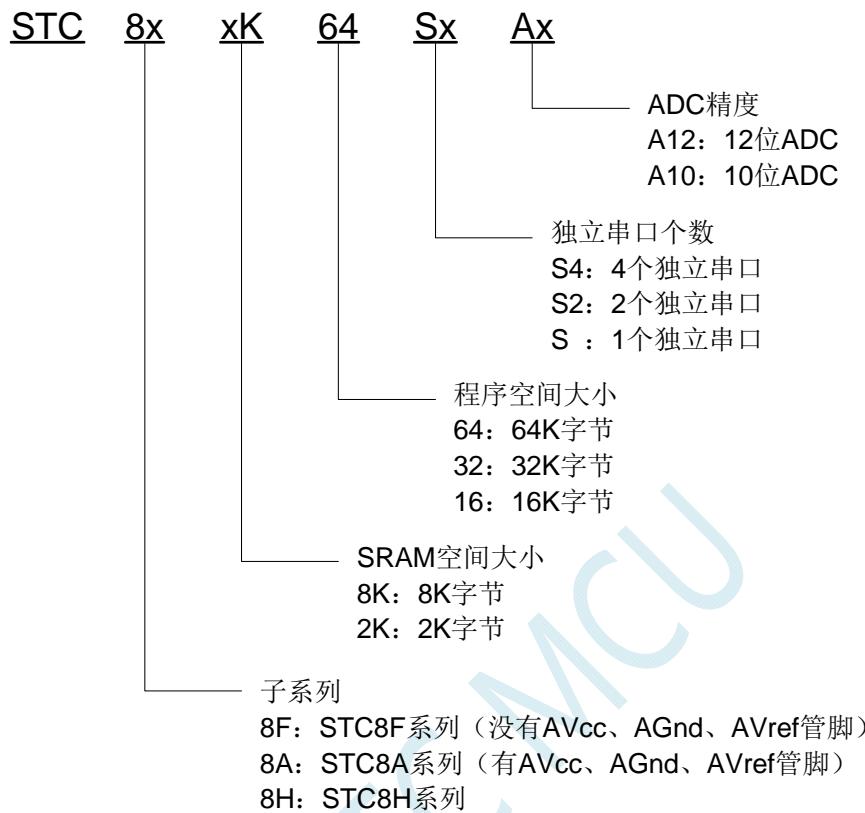
4.10 TSSOP20 封装尺寸图



4.11 SOP16 封装尺寸图



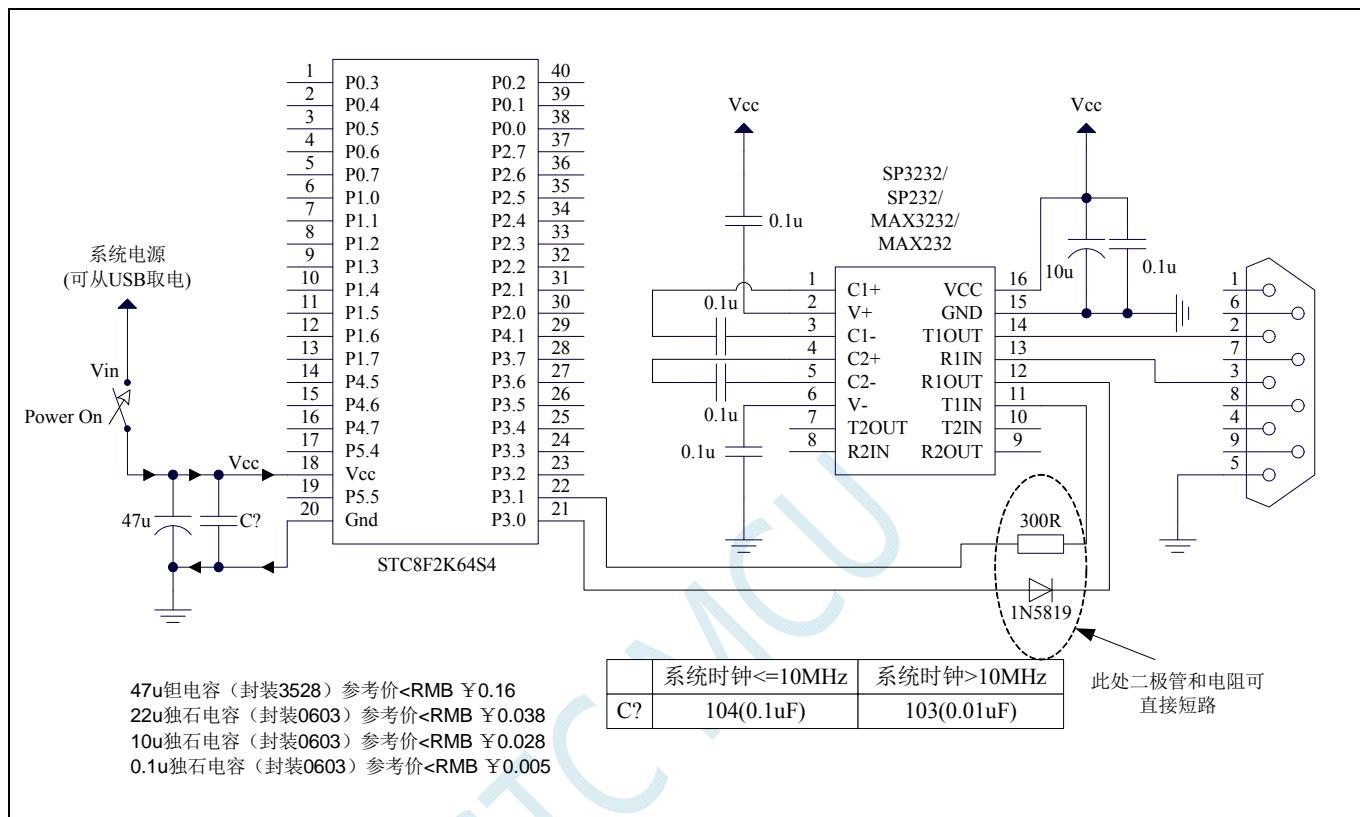
4.12 STC8 系列单片机命名规则



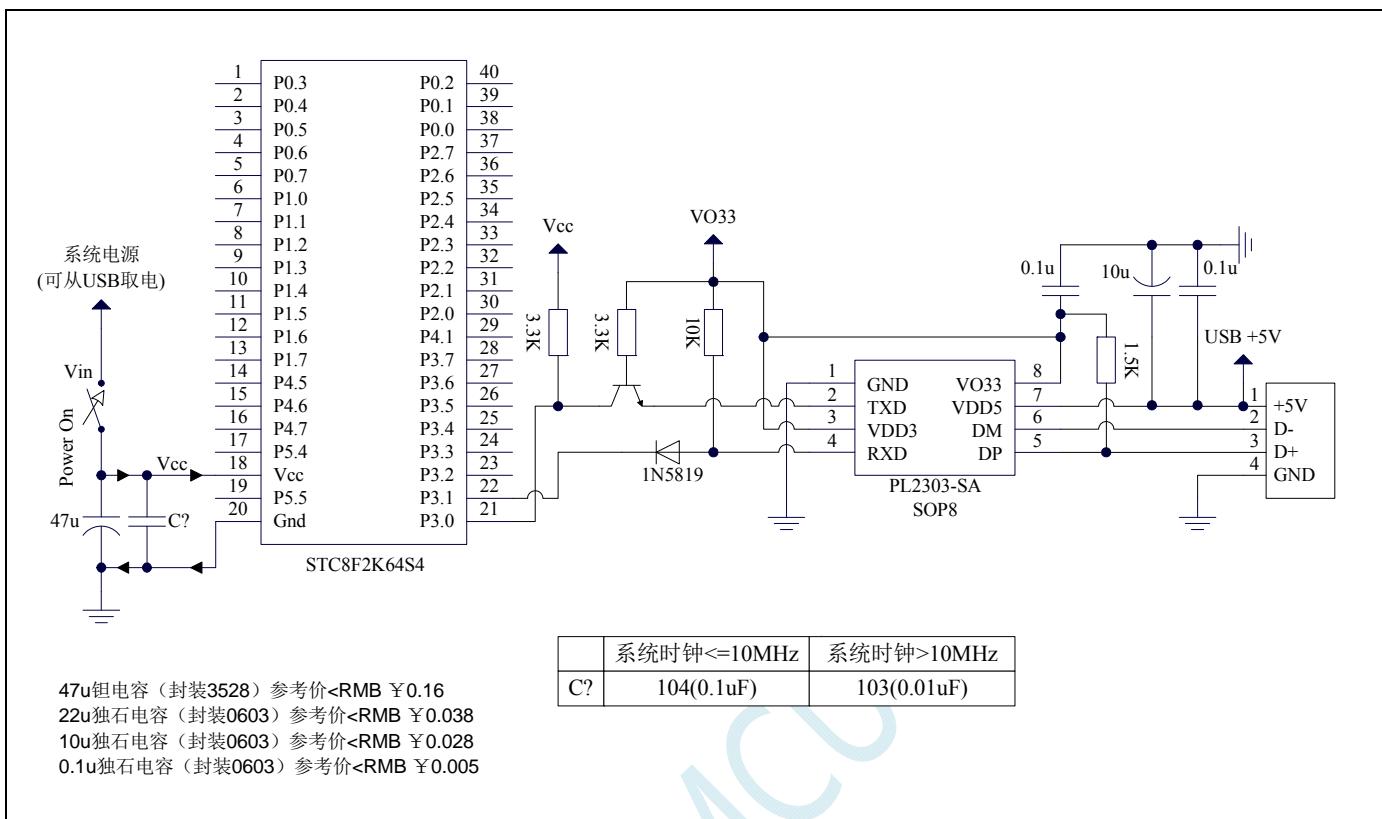
5 ISP下载及典型应用线路图

5.1 STC8F系列ISP下载应用线路图

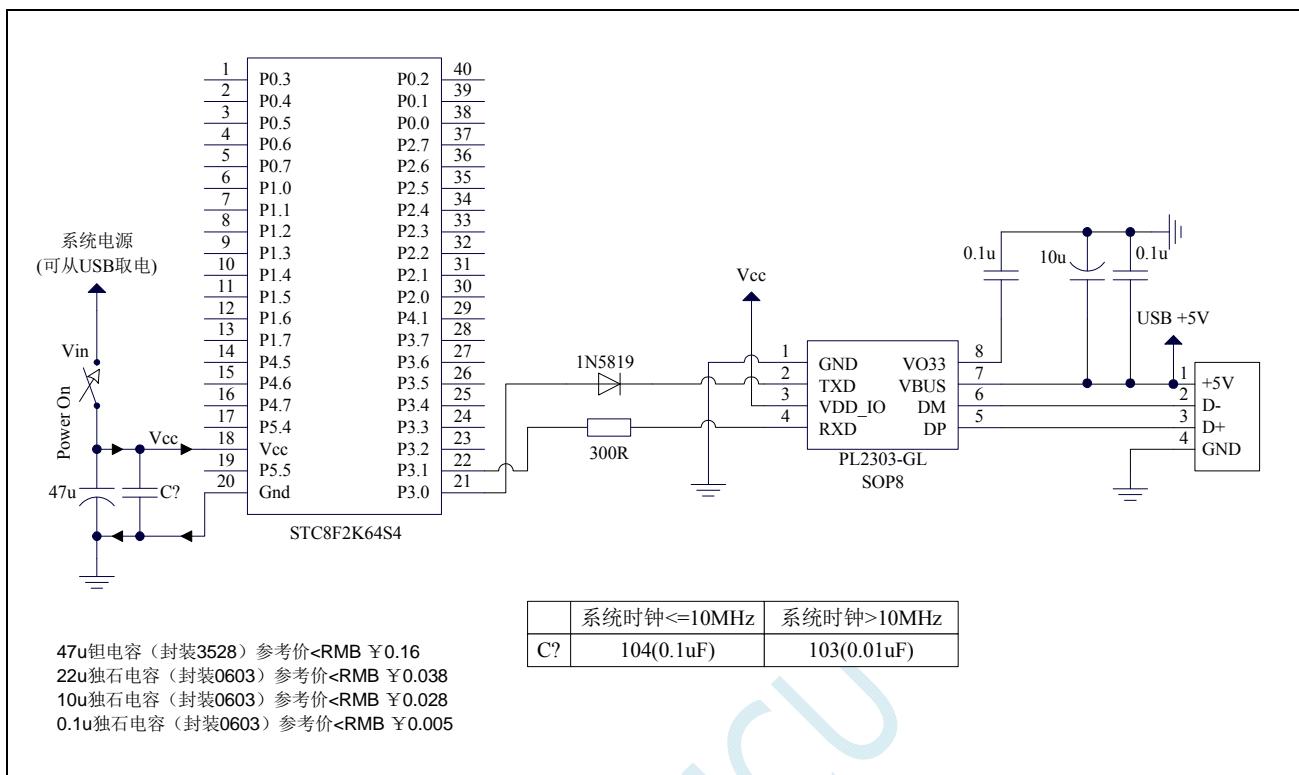
5.1.1 使用RS-232 转换器下载



5.1.2 使用PL2303-SA下载

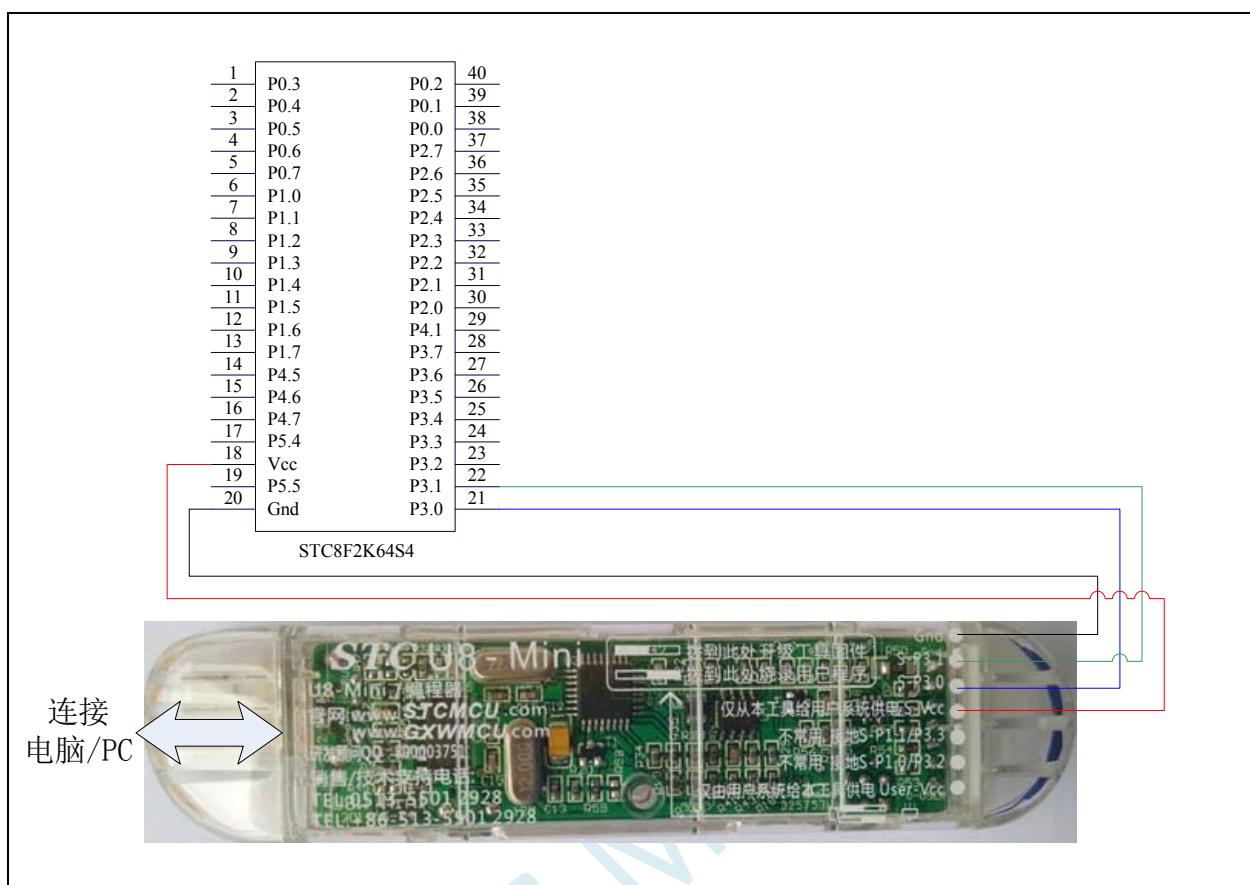


5.1.3 使用PL2303-GL下载

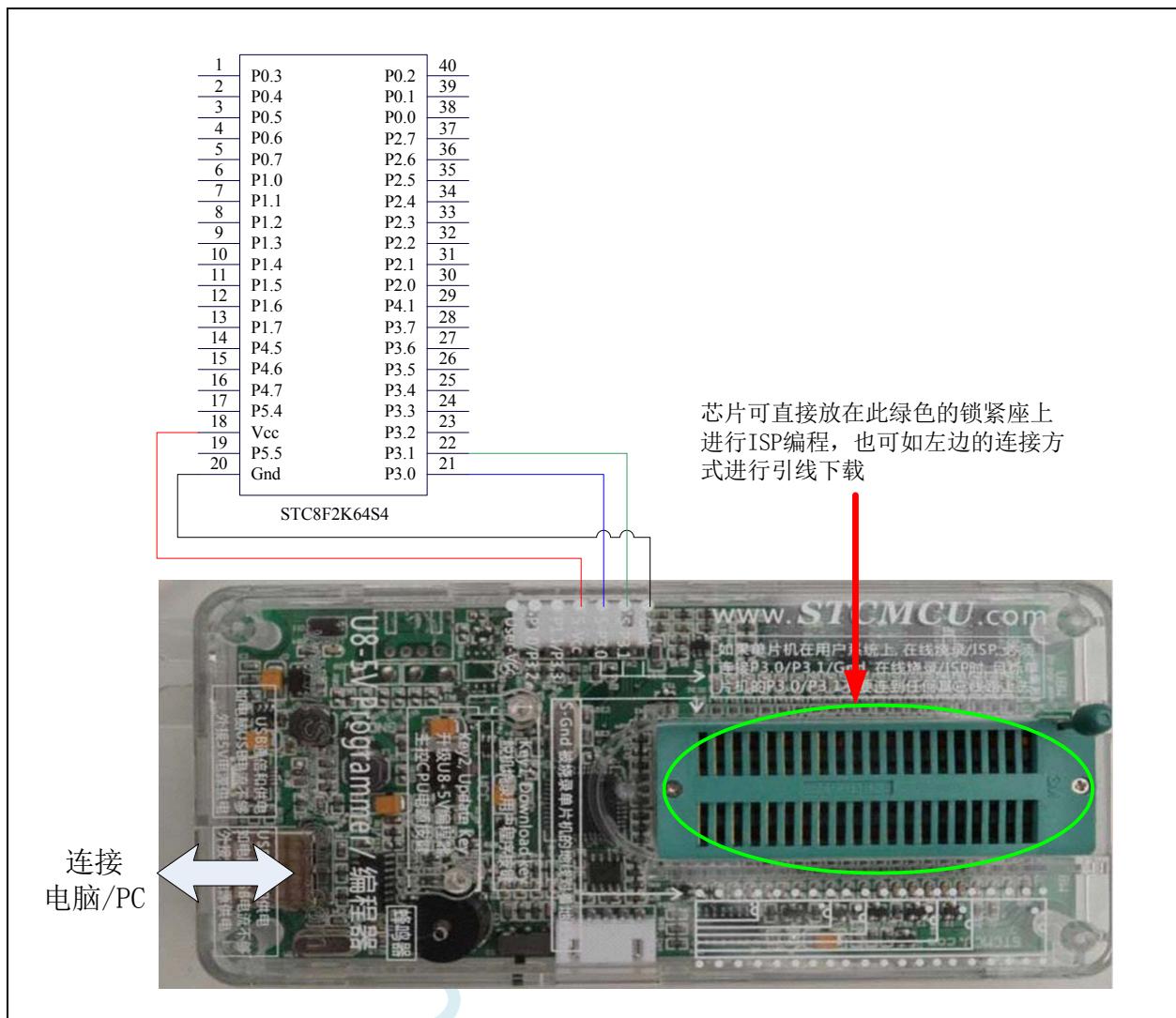


注: 使用 PL2303-GL 进行开发产品时, 必须到 Prolific 官网下载最新驱动进行安装。

5.1.4 使用U8-Mini工具下载

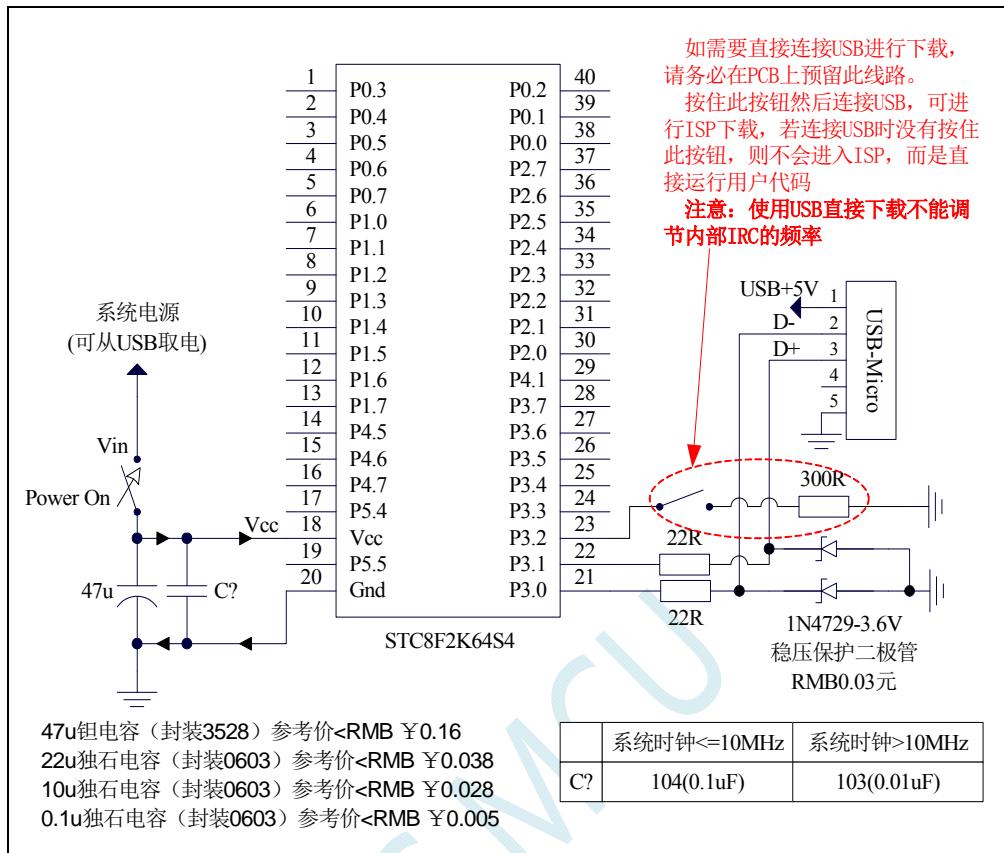


5.1.5 使用U8W工具下载



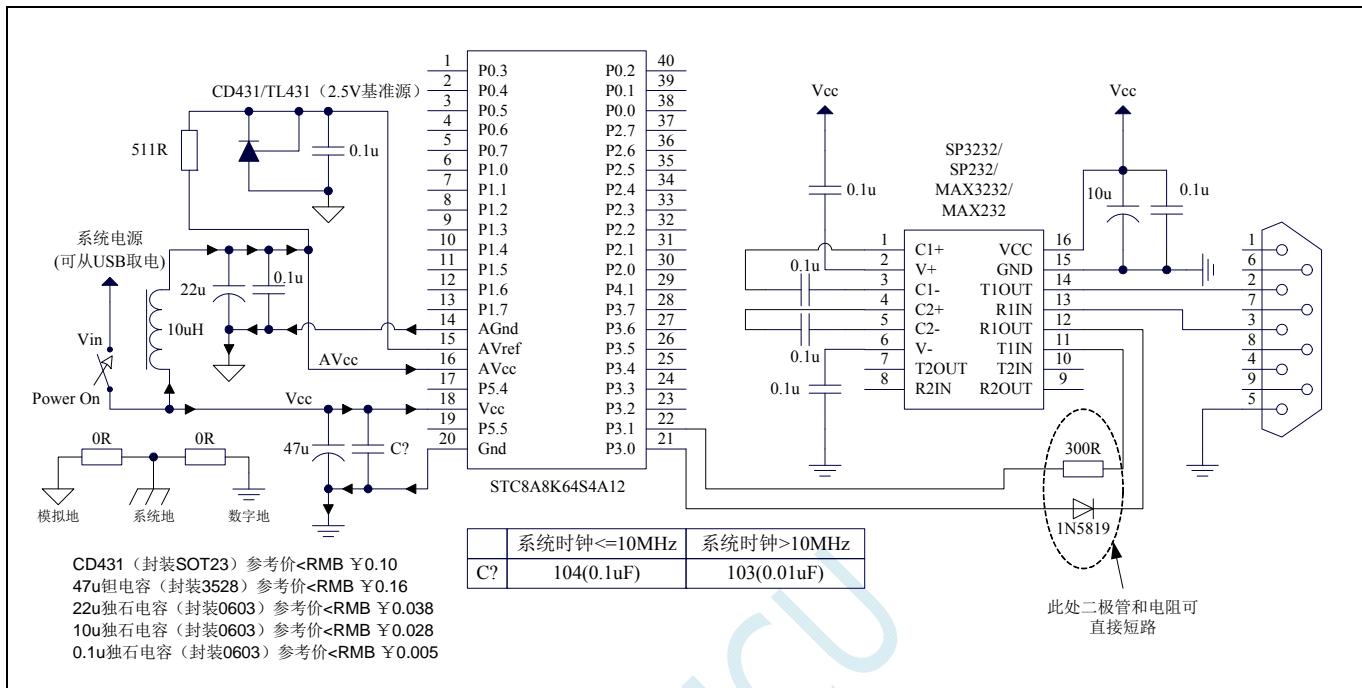
5.1.6 USB直接ISP下载

注: 使用 USB 下载时需要将 P3.2 接 GND 才可进行正常下载

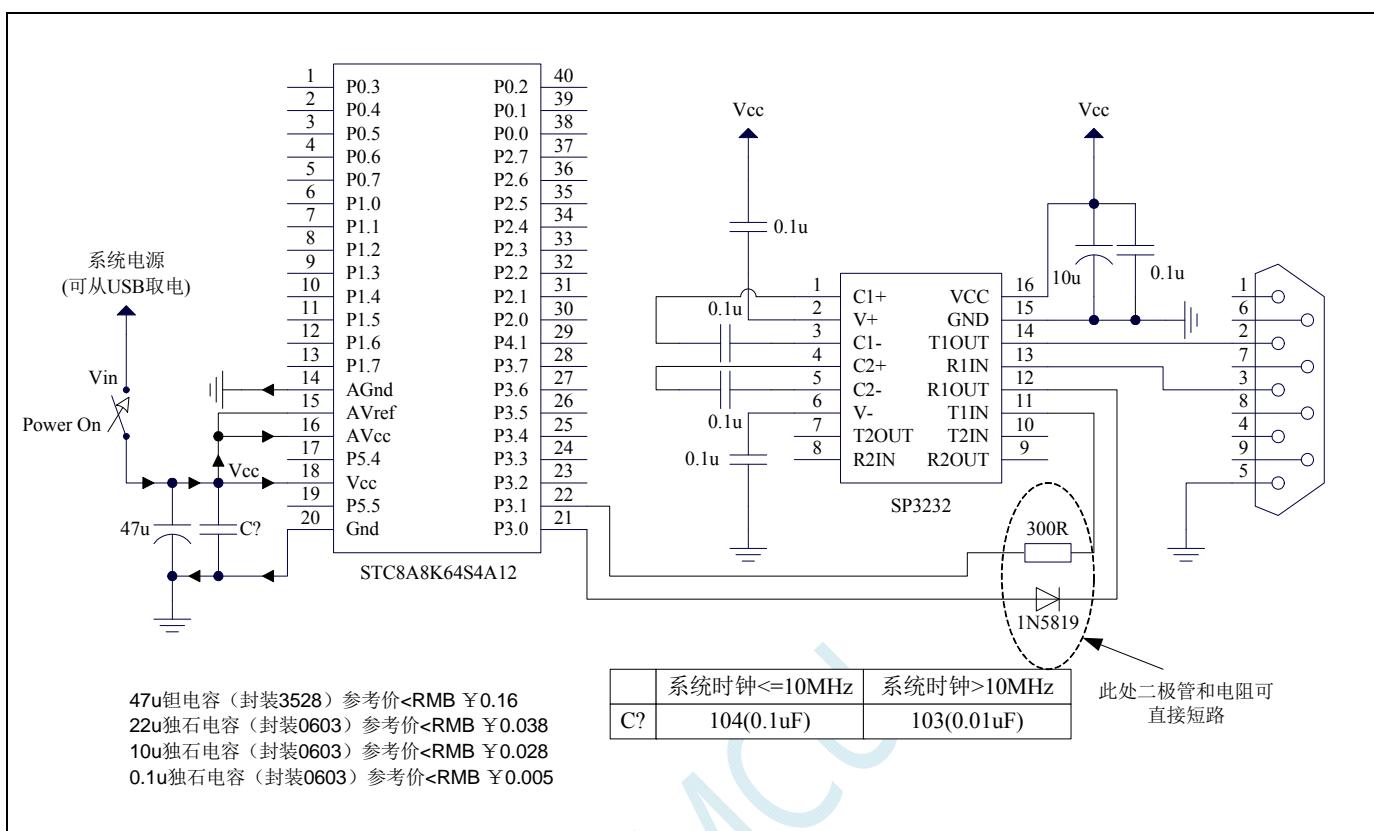


5.2 STC8A系列ISP下载应用线路图

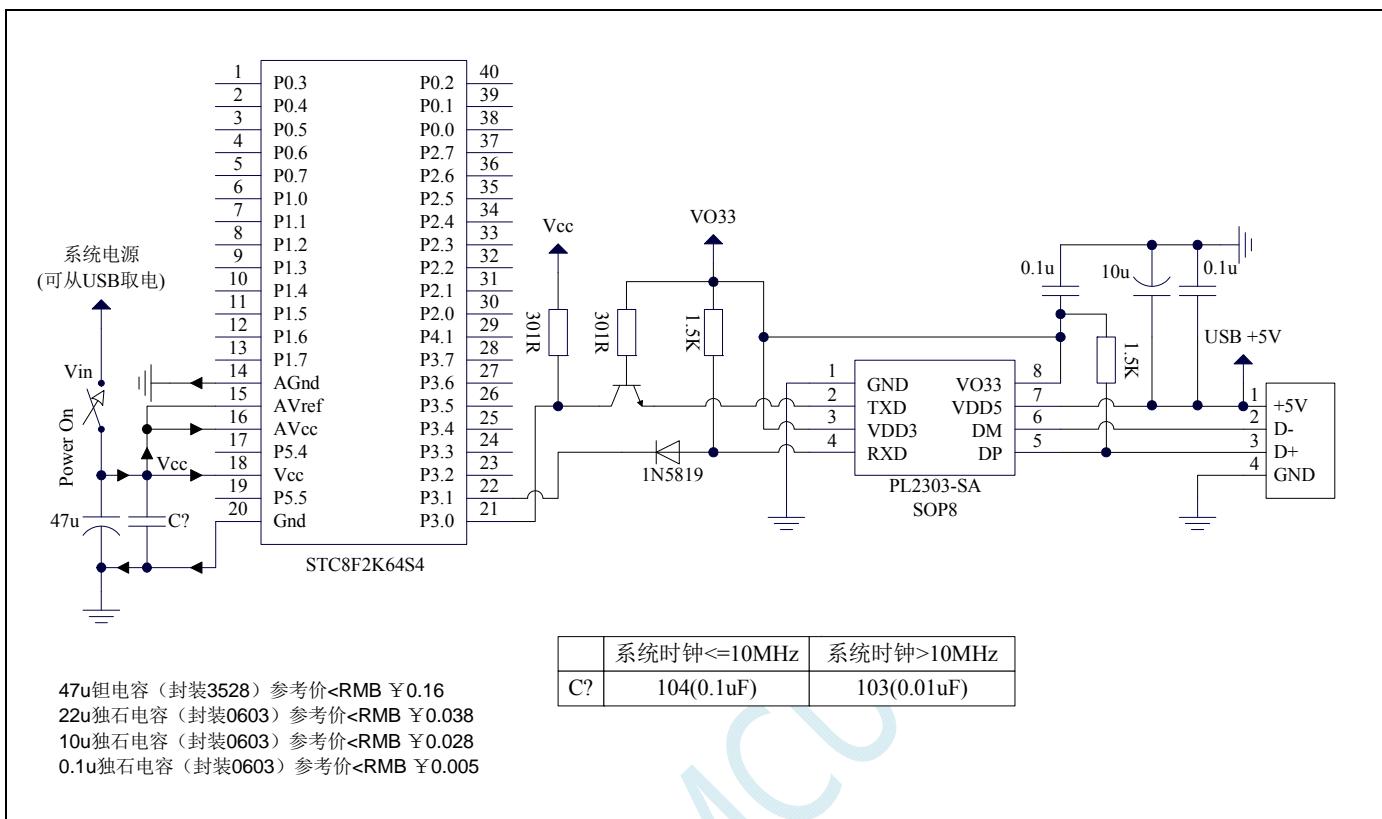
5.2.1 使用RS-232 转换下载（使用高精度ADC）



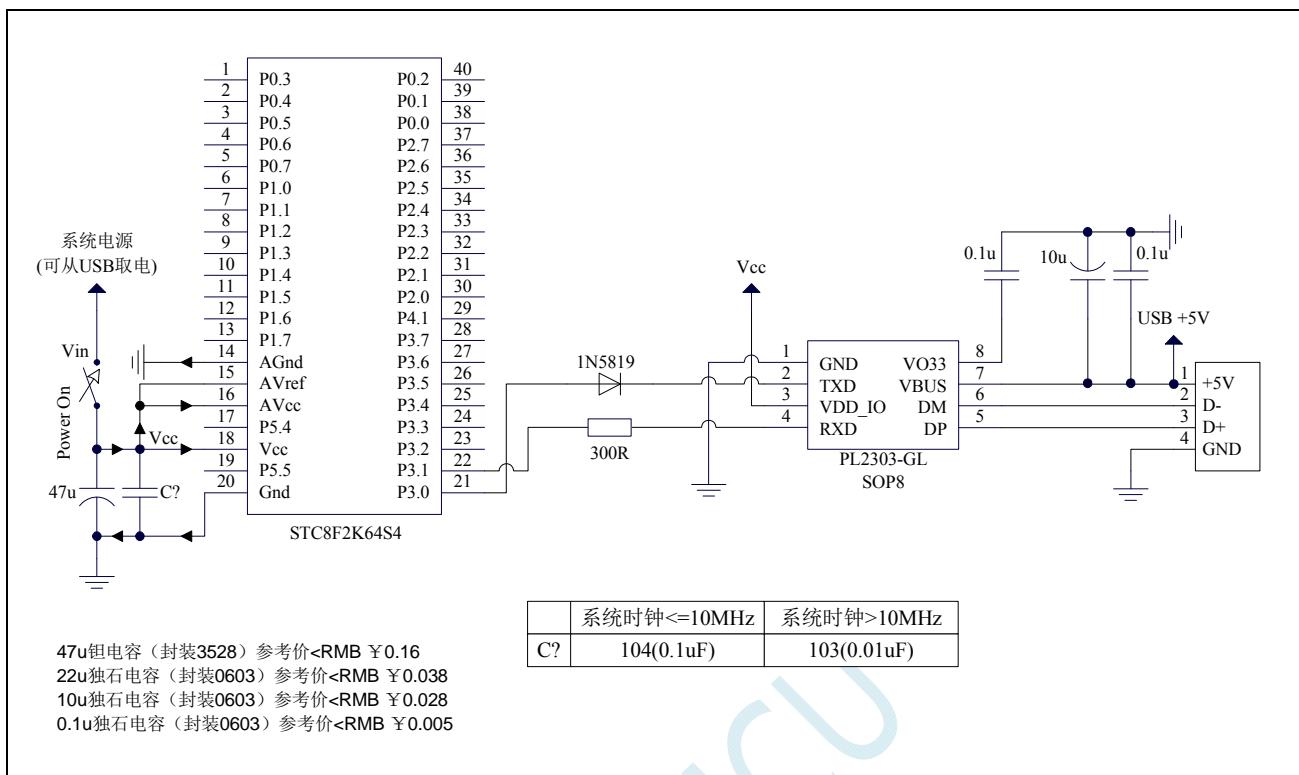
5.2.2 使用RS-232 转换下载 (ADC一般应用)



5.2.3 使用PL2303-SA下载

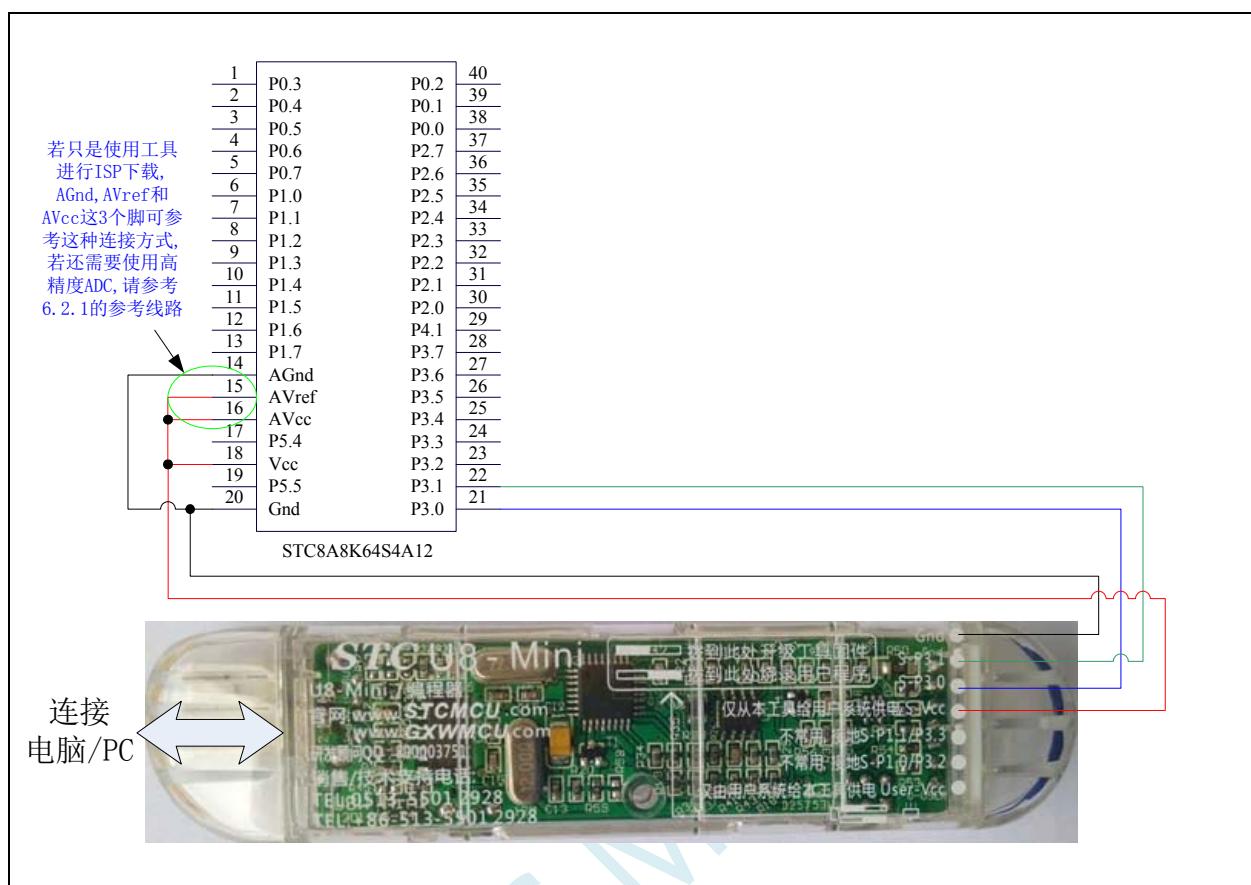


5.2.4 使用PL2303-GL下载

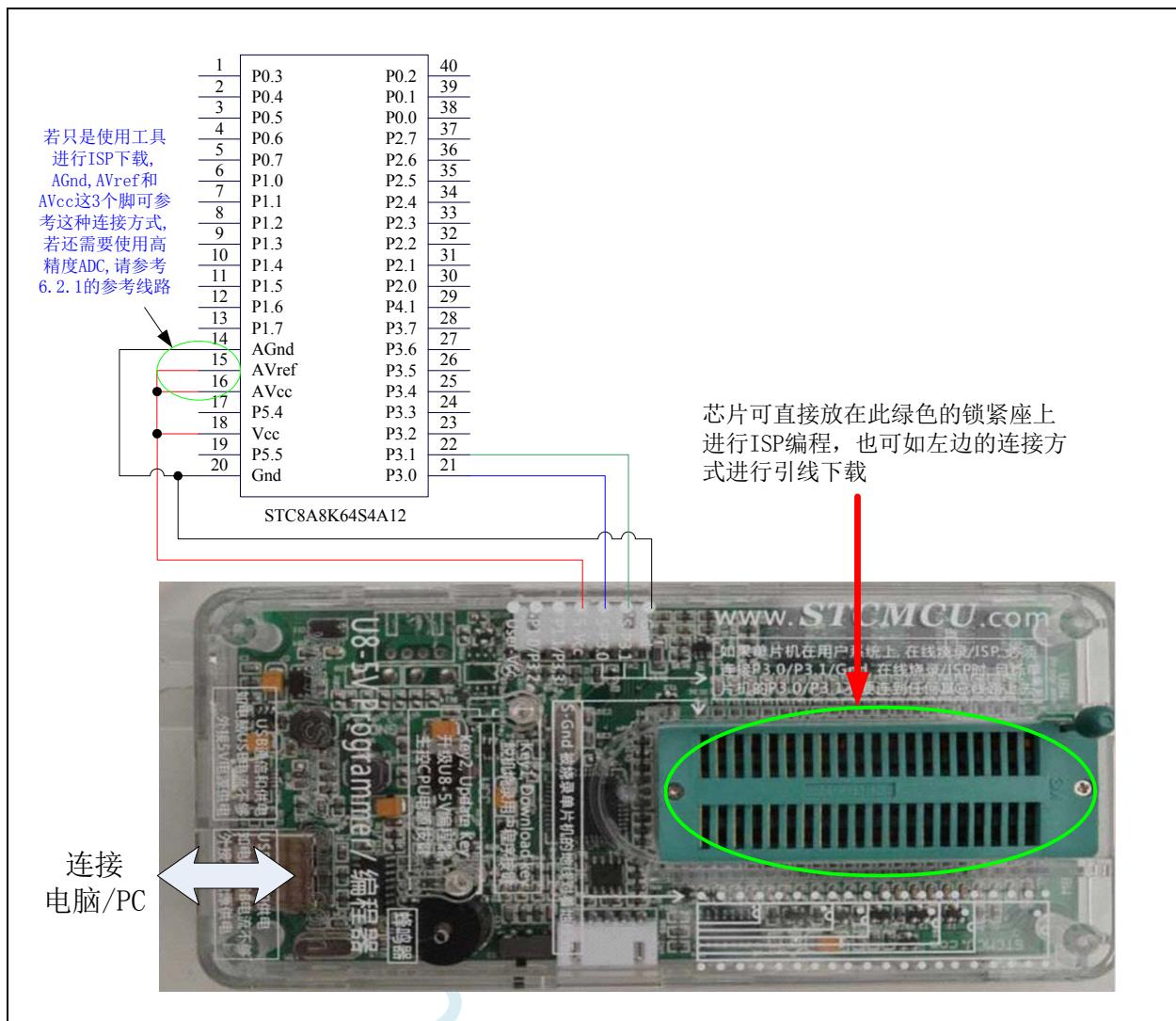


注: 使用 PL2303-GL 进行开发产品时, 必须到 Prolific 官网下载最新驱动进行安装。

5.2.5 使用U8-Mini工具下载

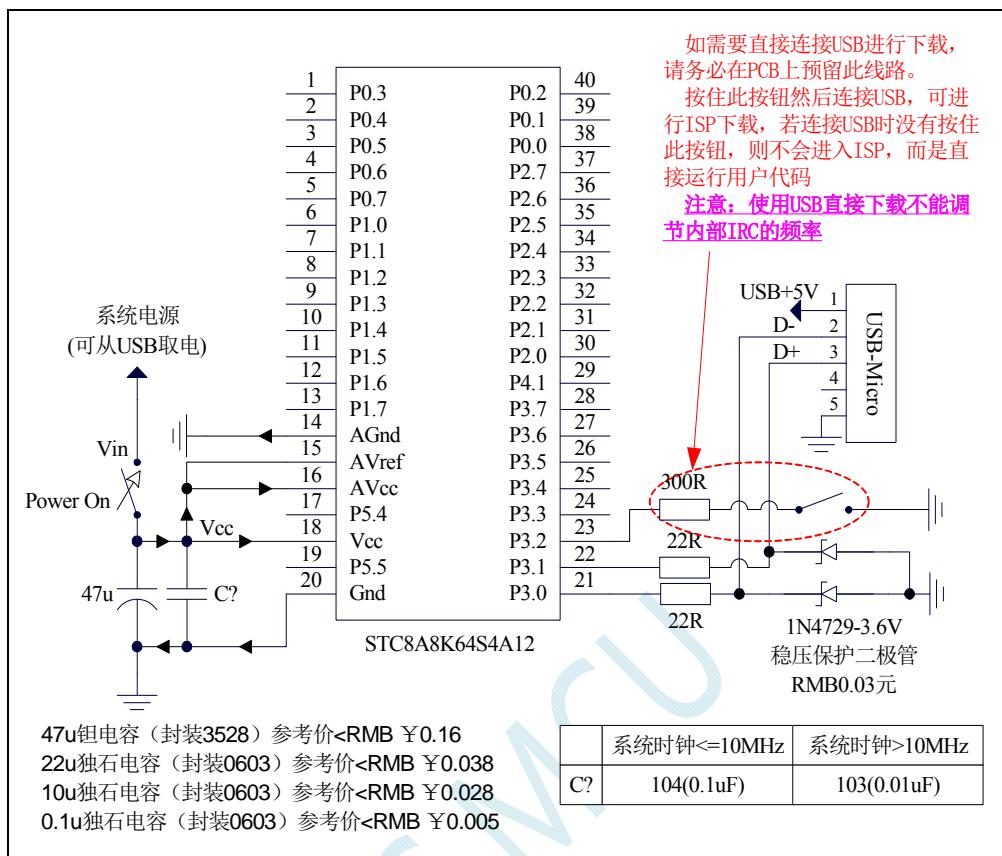


5.2.6 使用U8W工具下载



5.2.7 USB 直接 ISP 下载

注: 使用 USB 下载时需要将 P3.2 接 GND 才可进行正常下载

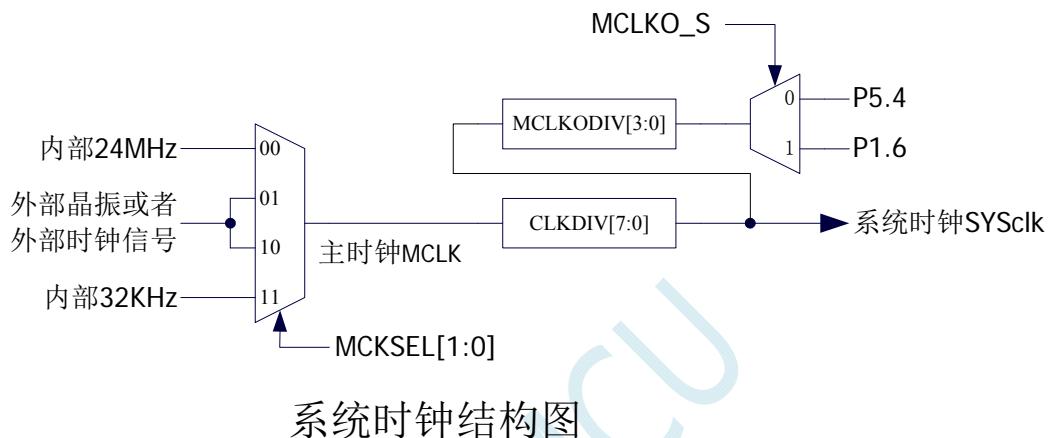


6 时钟、复位与电源管理

6.1 系统时钟控制

系统时钟控制器为单片机的 CPU 和所有外设系统提供时钟源，系统时钟有 3 个时钟源可供选择：内部高精度 24MHz 的 IRC、内部 32KHz 的 IRC（误差较大）、外部晶体振荡器或外部时钟信号。用户可通过程序分别使能和关闭各个时钟源，以及内部提供时钟分频以达到降低功耗的目的。

单片机进入掉电模式后，时钟控制器将会关闭所有的时钟源



相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CKSEL	时钟选择寄存器	FE00H	MCLKODIV[3:0]				MCLKO_S	-	MCKSEL[1:0]		0000,0000
CLKDIV	时钟分频寄存器	FE01H									0000,0100
IRC24MCR	内部 24M 振荡器控制寄存器	FE02H	ENIRC24M	-	-	-	-	-	-	IRC24MST	1xxx,xxx0
XOSCCR	外部晶振控制寄存器	FE03H	ENXOSC	XITYPE	-	-	-	-	-	XOSCST	0xxx,xxx0
IRC32KCR	内部 32K 振荡器控制寄存器	FE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST	0xxx,xxx0

CKSEL (系统时钟选择寄存器)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0	
CKSEL	FE00H	MCLKODIV[3:0]				MCLKO_S	MCKSEL[1:0]			

MCLKODIV[3:0]: 系统时钟输出分频系数

(注意: 系统时钟分频输出的时钟源是主时钟 MCLK 经过 CLKDIV 分频后的系统时钟)

MCLKODIV[3:0]	系统时钟分频输出频率
0000	不输出时钟
0001	SYSclk/1
001x	SYSclk /2
010x	SYSclk /4
011x	SYSclk /8
100x	SYSclk /16
101x	SYSclk /32

110x	SYSclk /64
111x	SYSclk /128

MCLKO_S: 系统时钟输出管脚选择

0: 系统时钟分频输出到 P5.4 口

1: 系统时钟分频输出到 P1.6 口

MCKSEL[1:0]: 主时钟源选择

MCKSEL[1:0]	主时钟源
00	内部 24MHz 高精度 IRC
01	外部晶体振荡器或
10	外部输入时钟信号
11	内部 32KHz 低速 IRC

CLKDIV (时钟分频寄存器)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CLKDIV	FE01H								

CLKDIV: 主时钟分频系数。系统时钟 SYSCLK 是对主时钟 MCLK 进行分频后的时钟信号。

CLKDIV	系统时钟频率
0	MCLK/1
1	MCLK/1
2	MCLK/2
3	MCLK/3
...	...
x	MCLK/x
...	...
255	MCLK/255

IRC24MCR (内部 24M 高精度 IRC 控制寄存器)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IRC24MCR	FE02H	ENIRC24M	-	-	-	-	-	-	IRC24MST

ENIRC24M: 内部 24M 高精度 IRC 使能位

0: 关闭内部 24M 高精度 IRC

1: 使能内部 24M 高精度 IRC

IRC24MST: 内部 24M 高精度 IRC 频率稳定标志位。(只读位)

当内部 24M 的 IRC 从停振状态开始使能后, 必须经过一段时间, 振荡器的频率才会稳定, 当振荡器频率稳定后, 时钟控制器会自动将 IRC24MST 标志位置 1。所以当用户程序需要将时钟切换到使用内部 24M 的 IRC 时, 首先必须设置 ENIRC24M=1 使能振荡器, 然后一直查询振荡器稳定标志位 IRC24MST, 直到标志位变为 1 时, 才可进行时钟源切换。

XOSCCR (外部振荡器控制寄存器)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
XOSCCR	FE03H	ENXOSC	XITYPE	-	-	-	-	-	XOSCST

ENXOSC: 外部晶体振荡器使能位

0: 关闭外部晶体振荡器

1: 使能外部晶体振荡器

XITYPE: 外部时钟源类型

0: 外部时钟源是外部时钟信号（或有源晶振）。信号源只需连接单片机的 XTAL1 (P1.7)

1: 外部时钟源是晶体振荡器。信号源连接单片机的 XTAL1 (P1.7) 和 XTAL0 (P1.6)

XOSCST: 外部晶体振荡器频率稳定标志位。(只读位)

当外部晶体振荡器从停振状态开始使能后，必须经过一段时间，振荡器的频率才会稳定，当振荡器频率稳定后，时钟控制器会自动将 XOSCST 标志位置 1。所以当用户程序需要将时钟切换到使用外部晶体振荡器时，首先必须设置 ENXOSC=1 使能振荡器，然后一直查询振荡器稳定标志位 XOSCST，直到标志位变为 1 时，才可进行时钟源切换。

IRC32KCR (内部 32KHz 低速 IRC 控制寄存器)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IRC32KCR	FE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST

ENIRC32K: 内部 32K 低速 IRC 使能位

0: 关闭内部 32K 低速 IRC

1: 使能内部 32K 低速 IRC

IRC32KST: 内部 32K 低速 IRC 频率稳定标志位。(只读位)

当内部 32K 低速 IRC 从停振状态开始使能后，必须经过一段时间，振荡器的频率才会稳定，当振荡器频率稳定后，时钟控制器会自动将 IRC32KST 标志位置 1。所以当用户程序需要将时钟切换到使用内部 32K 低速 IRC 时，首先必须设置 ENIRC32K=1 使能振荡器，然后一直查询振荡器稳定标志位 IRC32KST，直到标志位变为 1 时，才可进行时钟源切换。

6.2 STC8 系列内部IRC频率调整

STC8 系列单片机内部均集成有一颗高精度内部 IRC 振荡器。在用户使用 ISP 下载软件进行下载时，ISP 下载软件会根据用户所选择/设置的频率自动进行调整，一般频率值可调整到±0.3%以下，调整后的频率在全温度范围内（-40℃~85℃）的温漂可达-1.8%~0.8%。

STC8 系列内部 IRC 只有一个频段，此频段的中心频率约为 24MHz，最小频率约为 16MHz，最大频率约为 30MHz（注意：不同的芯片以及不同的生成批次可能会有约 5% 左右的制造误差）。

内部 IRC 频率调整主要使用下面的 3 个寄存器进行调整

相关寄存器

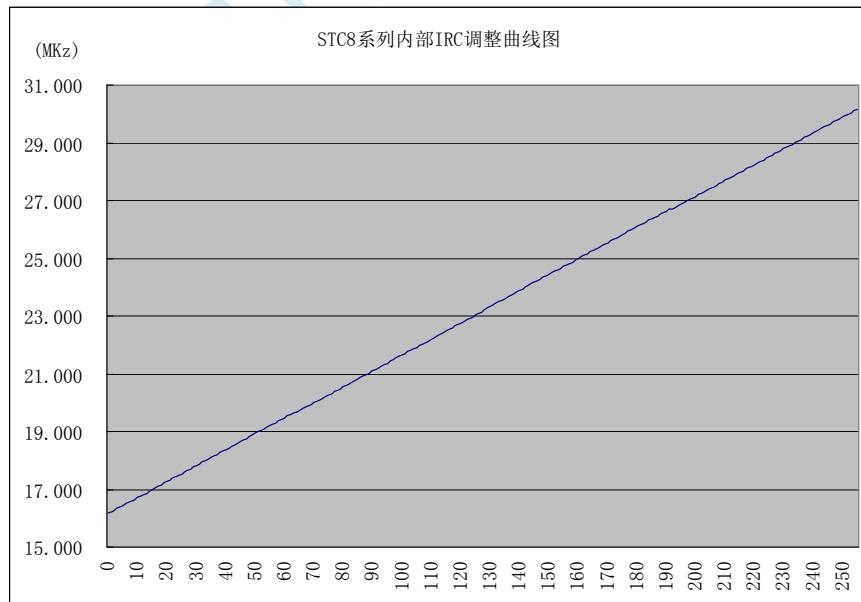
符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
LIRTRIM	IRC 频率微调寄存器	9EH	-	-	-	-	-	-	-	LIRTRIM[1:0]	0000,00nn
IRTRIM	IRC 频率调整寄存器	9FH	IRTRIM[7:0]								nnnn,nnnn
CLKDIV	时钟分频寄存器	FE01H	CLKDIV[7:0]								0000,0100

内部 IRC 频率调整寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IRTRIM	9FH	IRTRIM[7:0]							

IRTRIM[7:0]: 内部高精度 IRC 频率调整寄存器

IRTRIM 可对 IRC 频率进行 256 个等级的调整，每个等级所调整的频率值在整体上呈线性分布，局部会有波动。宏观上，每一级所调整的频率约为 0.24%，即 IRTRIM 为 (n+1) 时的频率比 IRTRIM 为 (n) 时的频率快 0.24%。但由于 IRC 频率调整并非每一级都是 0.24%（每一级所调整频率的最大值约为 0.55%，最小值约为 0.02%，整体平均值约为 0.24%），所以会造成局部波动。STC8 系列内部 IRC 频率调整的曲线图如下图所示：



内部 IRC 频率微调寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LIRTRIM	9EH	-	-	-	-	-	-	-	LIRTRIM[1:0]

LIRTRIM[1:0]: 内部高精度 IRC 频率微调寄存器

LIRTRIM 可对 IRC 频率进行 3 个等级的调整, 3 个等级所调整的频率范围如下表所示:

LIRTRIM[1:0]	调整的频率范围
00	不微调
01	调整约 0.10%
10	调整约 0.04%
11	调整约 0.10%

CLKDIV (时钟分频寄存器)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CLKDIV	FE01H								

CLKDIV: 主时钟分频系数。系统时钟 SYSCLK 是对主时钟 MCLK 进行分频后的时钟信号。

CLKDIV	系统时钟频率
0	MCLK/1
1	MCLK/1
2	MCLK/2
3	MCLK/3
...	...
x	MCLK/x
...	...
255	MCLK/255

由于 STC8 系列内部只有一个以 24MHz 为基准频率的频段, 频率范围只能在 16~30MHz, 所以 STC8 系列内部 IRC 所能调整频率的上限即为 30MHz(由于制造误差, 部分芯片的上限值可能为 28MHz~29MHz); 在未分频的情况下所能调整频率的下限为 16MHz (由于制造误差, 部分芯片的下限值可能为 17MHz)。当用户需要使用低于 16MHz 的频率时, 可将内部 IRC 的频率调整到目标频率的 2 倍或者 3 倍等, 再使用 CLKDIV 进行分频即可得到用户所需要的频率。例如用户需要 11.0592MHz 的频率, 使用内部 IRC 直接调整是无法得到这个频率的, 但可将内部 IRC 调整到 22.1184MHz, 在使用 CLKDIV 进行 2 分频即可得到 11.0592MHz。

注意: 由于 STC8 系列内部 IRC 频段的最小频率比最大频率的 1/2 大, 所以 STC8 系列在进行频率调整时存在盲区。一般 STC8 系列的芯片无法调整到 15MHz~16MHz 之间频率(由于制造误差, 部分芯片的盲区可能会更大一些)。

6.3 STC15 系列内部IRC频率调整

STC15 系列单片机内部均集成有一颗高精度内部 IRC 振荡器。在用户使用 ISP 下载软件进行下载时, ISP 下载软件会根据用户所选择/设置的频率自动进行调整, 一般频率值可调整到±0.3%以下, 调整后的频率在全温度范围内 (-40°C~85°C) 的温漂可达±1%

STC15 系列内部 IRC 有 4 个频段, 4 个频段的中心频率分别约为 5.5MHz、11MHz、22MHz 和 33MHz, 其中 5.5MHz 频段的频率范围约在 3.8MHz~7.1MHz, 11MHz 频段的频率范围约在 7.5MHz~13.8MHz, 22MHz 频段的频率范围约在 15.5MHz~28MHz, 33MHz 频段的频率范围约在 24.3MHz~43MHz(注意: 不同的芯片以及不同的生成批次可能会有约 5% 左右的制造误差)。

注意: STC15 系列内部 IRC 的 4 个频段的选择必须通过 ISP 下载时进行自动选择, 在用户程序中只能使用特殊功能寄存器 IRTRIM 在同一频段内进行微调。

STC15 系列内部 IRC 频率调整主要使用下面的寄存器进行调整

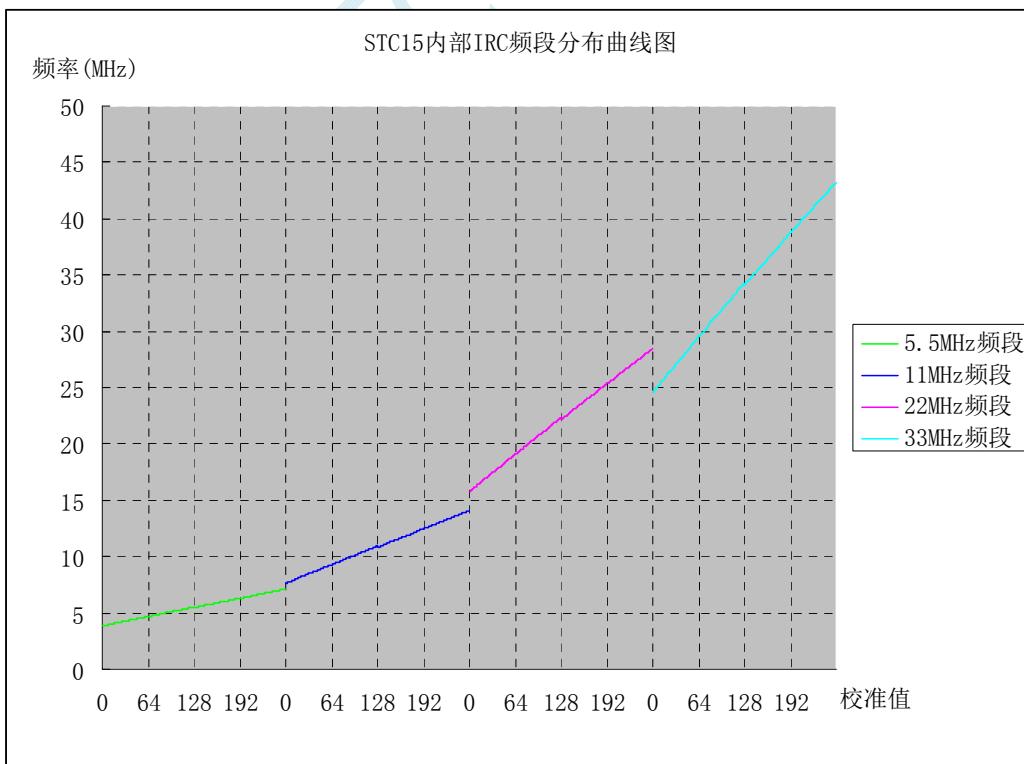
内部 IRC 频率调整寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IRTRIM	9FH					IRTRIM[7:0]			

IRTRIM[7:0]: 内部高精度 IRC 频率调整寄存器

IRTRIM 可对 IRC 每个频段的频率进行 256 个等级的调整(在进行频率调整前必须先选择相应的目标频段), 每个等级所调整的频率值在整体上呈线性分布, 局部会有波动。宏观上, 每一级所调整的频率约为 0.24%, 即 IRTRIM 为 (n+1) 时的频率比 IRTRIM 为 (n) 时的频率约快 0.24%。但由于 IRC 频率调整并非每一级都是 0.24% (每一级所调整频率的最大值约为 0.55%, 最小值约为 0.02%, 整体平均值约为 0.24%), 所以会造成局部波动。

STC15 系列内部 IRC 各个频段的频率调整的曲线图如下图所示:



从上图可用看出, STC15 系列的 4 个频段中, 存在频率盲区。

5.5MHz 和 11MHz 两个频段的过渡点中存在断层, 即 5.5MHz 频段的可调最大频率值比 11MHz 频

段的可调最小频率值小, 所以 **STC15** 系列在进行频率调整时 **5.5MHz** 和 **11MHz** 之间存在频率盲区(一般在 **7MHz~8MHz** 之间, 由于制造误差, 部分芯片的盲区可能会更大一些)。

同样, 11MHz 和 22MHz 两个频段的过渡点中存在断层, 即 11MHz 频段的可调最大频率值比 22MHz 频段的可调最小频率值小, 所以 **STC15** 系列在进行频率调整时 **11MHz** 和 **22MHz** 之间存在频率盲区(一般在 **13MHz~16MHz** 之间, 由于制造误差, 部分芯片的盲区可能会更大一些);

22MHz 与 33MHz 两个频段的过渡点中不存在断层, 故也不存在频率盲区。

由于有盲区的存在, 所以用户在给目标芯片设置 IRC 频率时, 应注意尽量避免使用可能出现在盲区的频率。

6.4 系统复位

STC8 系列单片机的复位分为硬件复位和软件复位两种。

硬件复位时，所有的寄存器的值会复位到初始值，系统会重新读取所有的硬件选项。同时根据硬件选项所设置的上电等待时间进行上电等待。硬件复位主要包括：

- 上电复位
- 低压复位
- 复位脚复位
- 看门狗复位

软件复位时，除与时钟相关的寄存器保持不变外，其余的所有寄存器的值会复位到初始值，软件复位不会重新读取所有的硬件选项。软件复位主要包括：

- 写 IAP_CONTR 的 SWRST 所触发的复位

相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
WDT CONTR	看门狗控制寄存器	C1H	WDT_FLAG	-	EN_WDT	CLR_WDT	IDL_WDT	WDT_PS[2:0]			0x00,0000
IAP CONTR	IAP 控制寄存器	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	IAP_WT[2:0]			0000,x000
RSTCFG	复位配置寄存器	FFH	-	ENLVR	-	P54RST	-	-	LVDS[1:0]	-	0000,0000

WDT CONTR (看门狗控制寄存器)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
WDT CONTR	C1H	WDT_FLAG	-	EN_WDT	CLR_WDT	IDL_WDT	WDT_PS[2:0]		

WDT_FLAG: 看门狗溢出标志

看门狗发生溢出时，硬件自动将此位置 1，需要软件清零。

EN_WDT: 看门狗使能位

- 0: 对单片机无影响
- 1: 启动看门狗定时器

CLR_WDT: 看门狗定时器清零

- 0: 对单片机无影响
- 1: 清零看门狗定时器，硬件自动将此位复位

IDL_WDT: IDLE 模式时的看门狗控制位

- 0: IDLE 模式时看门狗停止计数
- 1: IDLE 模式时看门狗继续计数

WDT_PS[2:0]: 看门狗定时器时钟分频系数

WDT_PS[2:0]	分频系数	12M 主频时的溢出时间	20M 主频时的溢出时间
000	2	≈ 65.5 毫秒	≈ 39.3 毫秒
001	4	≈ 131 毫秒	≈ 78.6 毫秒
010	8	≈ 262 毫秒	≈ 157 毫秒
011	16	≈ 524 毫秒	≈ 315 毫秒
100	32	≈ 1.05 秒	≈ 629 毫秒
101	64	≈ 2.10 秒	≈ 1.26 秒
110	128	≈ 4.20 秒	≈ 2.52 秒
111	256	≈ 8.39 秒	≈ 5.03 秒

看门狗溢出时间计算公式如下:

$$\text{看门狗溢出时间} = \frac{12 \times 32768 \times 2^{(\text{WDT_PS}+1)}}{\text{SYSclk}}$$

IAP_CONTR (IAP 控制寄存器)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CONTR	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	IAP_CMD[2:0]		

SWBS: 软件复位启动选择

- 0: 软件复位后从用户程序区开始执行代码。用户数据区的数据保持不变。
- 1: 软件复位后从系统 ISP 区开始执行代码。用户数据区的数据会被初始化。

SWRST: 软件复位触发位

- 0: 对单片机无影响
- 1: 触发软件复位

RSTCFG (复位配置寄存器)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
RSTCFG	FFH	-	ENLVR	-	P54RST	-	-	LVDS[1:0]	

ENLVR: 低压复位控制位

- 0: 禁止低压复位。当系统检测到低压事件时，会产生低压中断
- 1: 使能低压复位。当系统检测到低压事件时，自动复位

P54RST: RST 管脚功能选择

- 0: RST 管脚用作普通 I/O 口 (P54)
- 1: RST 管脚用作复位脚

LVDS[1:0]: 低压检测门槛电压设置

LVDS[1:0]	低压检测门槛电压
00	2.2V
01	2.4V
10	2.7V
11	3.0V

注意事项:

1. 复位脚用于复位时，下拉电阻不大于 3K。在 1.5V 时有超过 200uA 的上拉电流，10K 电路拉不低。
2. 使用外部晶振时，要注意下面的要求：AVCC 跟 VCC 电压差不要超过 0.1V，P1.0~P1.7 P0.0~P0.6 电压不能比 AVCC 高，否则晶振不起振，形同死机。

6.5 系统电源管理

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
PCON	电源控制寄存器	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
VOCTRL	电压控制寄存器	BBH	SCC	-	-	-	-	-	0	0	0xxx,xx00

PCON (电源控制寄存器)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PCON	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL

LVDF: 低压检测标志位。当系统检测到低压事件时，硬件自动将此位置 1，并向 CPU 提出中断请求。

此位需要用户软件清零。

POF: 上电标志位。当硬件自动将此位置 1。

PD: 掉电模式控制位

0: 无影响

1: 单片机进入掉电模式，CPU 以及全部外设均停止工作。唤醒后硬件自动清零。

(可将 CPU 从掉电模式唤醒的外部管脚有：INT0/P3.2、INT1/P3.3、INT2/P3.6、INT3/P3.7、INT4/P3.0、T0/P3.4、T1/P3.5、T2/P1.2、T3/P0.4、T4/P0.6、CCP0/P1.7、CCP1/P1.6、CCP2/P1.5、CCP4/P1.4、CCP0_2/P2.3、CCP1_2/P2.4、CCP2_2/P2.5、CCP3_2/P2.6、CCP0_3/P7.0、CCP1_3/P7.1、CCP2_3/P7.2、CCP3_3/P7.3、CCP0_4/P3.3、CCP1_4/P3.2、CCP2_4/P3.1、CCP3_4/P3.0、RxD/P3.0、RxD_2/P3.6、RxD_3/P1.6、RxD_4/P4.3、RxD2/P1.0、RxD2_2/P4.0、RxD3/P0.0、RxD3_2/P5.0、RxD4/P0.2、RxD4_2/P5.2，另外还有掉电唤醒定时器、低电压中断和比较器中断也可将 CPU 从掉电模式唤醒。)

IDL: IDLE (空闲) 模式控制位

0: 无影响

1: 单片机进入 IDLE 模式，只有 CPU 停止工作，其他外设依然在运行。唤醒后硬件自动清零

VOCTRL (电压控制寄存器)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
VOCTRL	BBH	SCC						0	0

SCC: 静态电流控制位

0: 选择内部静态保持电流控制线路，静态电流一般为 1.5uA 左右。

1: 选择外部静态保持电流控制线路，选择此模式时功耗更低。此模式下 STC8A8K 系列的静态电流一般为 0.15uA 以下；STC8F2K 系列的静态电流一般为 0.1uA 以下。注意：选择此模式进入掉电模式后，VCC 管脚的电压不能有较大波动，否则对 MCU 内核可能会有不良影响。

[B1:B0]: 内部测试位，必须写入 0

6.6 范例程序

6.6.1 选择系统时钟源

汇编代码

```

P_SW2      DATA    0BAH
CKSEL      EQU     0FE00H
CLKDIV    EQU     0FE01H
IRC24MCR   EQU     0FE02H
XOSCCR    EQU     0FE03H
IRC32KCR   EQU     0FE04H

        ORG     0000H
        LJMP    MAIN

        ORG     0100H
MAIN:
        MOV     SP,#3FH

        MOV     P_SW2,#80H
        MOV     A,#00H           ;选择内部IRC(默认)
        MOV     DPTR,#CKSEL
        MOVX   @DPTR,A
        MOV     P_SW2,#00H

;
        MOV     P_SW2,#80H
;
        MOV     A,#0C0H           ;启动外部晶振
;
        MOV     DPTR,#XOSCCR
;
        MOVX   @DPTR,A
;
        MOVX   A,@DPTR
;
        JNB    ACC.0,$-1         ;等待时钟稳定
;
        CLR    A                 ;时钟不分频
;
        MOV     DPTR,#CLKDIV
;
        MOVX   @DPTR,A
;
        MOV     A,#01H           ;选择外部晶振
;
        MOV     DPTR,#CKSEL
;
        MOVX   @DPTR,A
;
        MOV     P_SW2,#00H

;
        MOV     P_SW2,#80H
;
        MOV     A,#80H           ;启动内部32KIRC
;
        MOV     DPTR,#IRC32KCR
;
        MOVX   @DPTR,A
;
        MOVX   A,@DPTR
;
        JNB    ACC.0,$-1         ;等待时钟稳定
;
        CLR    A                 ;时钟不分频
;
        MOV     DPTR,#CLKDIV
;
        MOVX   @DPTR,A
;
        MOV     A,#03H           ;选择内部32K
;
        MOV     DPTR,#CKSEL
;
        MOVX   @DPTR,A
;
        MOV     P_SW2,#00H

        JMP    $
END

```

C 语言代码

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

#define CKSEL          (*(unsigned char volatile xdata *)0xfe00)
#define CLKDIV         (*(unsigned char volatile xdata *)0xfe01)
#define IRC24MCR       (*(unsigned char volatile xdata *)0xfe02)
#define XOSCCR         (*(unsigned char volatile xdata *)0xfe03)
#define IRC32KCR       (*(unsigned char volatile xdata *)0xfe04)

sfr P_SW2 = 0xba;

void main()
{
    P_SW2 = 0x80;                                //选择内部IRC ( 默认 )
    CKSEL = 0x00;
    P_SW2 = 0x00;

    /*
    P_SW2 = 0x80;
    XOSCCR = 0xc0;                               //启动外部晶振
    while (!(XOSCCR & 1));                      //等待时钟稳定
    CLKDIV = 0x00;                                //时钟不分频
    CKSEL = 0x01;                                //选择外部晶振
    P_SW2 = 0x00;
    */

    /*
    P_SW2 = 0x80;
    IRC32KCR = 0x80;                            //启动内部32K IRC
    while (!(IRC32KCR & 1));                    //等待时钟稳定
    CLKDIV = 0x00;                                //时钟不分频
    CKSEL = 0x03;                                //选择内部32K
    P_SW2 = 0x00;
    */

    while (1);
}
```

6.6.2 主时钟分频输出

汇编代码

P_SW2	DATA	0BAH
CKSEL	EQU	0FE00H
CLKDIV	EQU	0FE01H
	ORG	0000H
	LJMP	MAIN
MAIN:	ORG	0100H
	MOV	SP,#3FH
	MOV	P_SW2,#80H
;	MOV	A,#10H ;主时钟输出到P5.4 口
;	MOV	A,#20H ;主时钟2 分频输出到P5.4 口
	MOV	A,#40H ;主时钟4 分频输出到P5.4 口
;	MOV	A,#48H ;主时钟4 分频输出到P1.6 口

```

MOV      DPTR,#CKSEL
MOVX    @DPTR,A
MOV      P_SW2,#00H

JMP      $

END

```

C 语言代码

```

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

#define CKSEL      (*(unsigned char volatile xdata *)0xfe00)
#define CLKDIV     (*(unsigned char volatile xdata *)0xfe01)

sfr      P_SW2      = 0xba;

void main()
{
    P_SW2 = 0x80;
    // CKSEL = 0x10;           //主时钟输出到P5.4 口
    // CKSEL = 0x20;           //主时钟2 分频输出到P5.4 口
    CKSEL = 0x40;           //主时钟4 分频输出到P5.4 口
    // CKSEL = 0x48;           //主时钟4 分频输出到P1.6 口
    P_SW2 = 0x00;

    while (1);
}

```

6.6.3 看门狗定时器应用

汇编代码

```

;测试工作频率为11.0592MHz

WDT_CONTR DATA 0CIH

        ORG 0000H
        LJMP MAIN

        ORG 0100H
MAIN:
        MOV SP,#3FH

;        MOV WDT CONTR,#23H      ;使能看门狗,溢出时间为0.5s
;        MOV WDT CONTR,#24H      ;使能看门狗,溢出时间为1s
;        MOV WDT CONTR,#27H      ;使能看门狗,溢出时间为8s
;        CLR P3.2                ;测试端口

LOOP:
;        ORL WDT CONTR,#10H      ;清看门狗,否则系统复位
        JMP LOOP

END

```

C 语言代码

```
#include "reg51.h"
```

```
#include "intrins.h"

//测试工作频率为11.0592MHz

sfr      WDT_CONTR = 0xc1;
sbit     P32        = P3^2;

void main()
{
//  WDT_CONTR = 0x23;           //使能看门狗,溢出时间约为0.5s
//  WDT_CONTR = 0x24;           //使能看门狗,溢出时间约为1s
//  WDT_CONTR = 0x27;           //使能看门狗,溢出时间约为8s
P32 = 0;                      //测试端口

while (1)
{
//    WDT_CONTR |= 0x10;        //清看门狗,否则系统复位
}
}
```

6.6.4 软复位实现自定义下载

汇编代码

```
;测试工作频率为11.0592MHz

IAP CONTR   DATA      0C7H

          ORG      0000H
          LJMP    MAIN

MAIN:
          ORG      0100H
          MOV      SP,#3FH

          SETB    P3.2
          SETB    P3.3

LOOP:
          JB      P3.2,LOOP
          JB      P3.3,LOOP
          MOV      IAP CONTR,#60H      ;检查到P3.2 和 P3.3 同时为0 时复位到ISP
          JMP      $

END
```

C 语言代码

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

//测试工作频率为11.0592MHz

sfr      IAP CONTR = 0xc7;
sbit     P32        = P3^2;
sbit     P33        = P3^3;

void main()
{
```

```

P32 = 1;           //测试端口
P33 = 1;           //测试端口

while (1)
{
    if (!P32 && !P33)
    {
        IAP_CONTR |= 0x60;      //检查到P3.2 和 P3.3 同时为0 时复位到ISP
    }
}
}

```

6.6.5 低压检测

汇编代码

<i>RSTCFG</i>	<i>DATA</i>	<i>0FFH</i>	
<i>ENLVR</i>	<i>EQU</i>	<i>40H</i>	;RSTCFG.6
<i>LVD2V2</i>	<i>EQU</i>	<i>00H</i>	;LVD@2.2V
<i>LVD2V4</i>	<i>EQU</i>	<i>01H</i>	;LVD@2.4V
<i>LVD2V7</i>	<i>EQU</i>	<i>02H</i>	;LVD@2.7V
<i>LVD3V0</i>	<i>EQU</i>	<i>03H</i>	;LVD@3.0V
<i>ELVD</i>	<i>BIT</i>	<i>IE.6</i>	
<i>LVDF</i>	<i>EQU</i>	<i>20H</i>	;PCON.5
	<i>ORG</i>	<i>0000H</i>	
	<i>LJMP</i>	<i>MAIN</i>	
	<i>ORG</i>	<i>0033H</i>	
	<i>LJMP</i>	<i>LVDISR</i>	
	<i>ORG</i>	<i>0100H</i>	
<i>LVDISR:</i>	<i>ANL</i>	<i>PCON,#NOT LVDF</i>	;清中断标志
	<i>CPL</i>	<i>P3.2</i>	;测试端口
	<i>RETI</i>		
<i>MAIN:</i>	<i>MOV</i>	<i>SP,#3FH</i>	
	<i>ANL</i>	<i>PCON,#NOT LVDF</i>	;上电后需要先清LVDF 标志
;	<i>MOV</i>	<i>RSTCFG#ENLVR / LVD3V0</i>	;使能3.0V 时低压复位,不产生LVD 中断
	<i>MOV</i>	<i>RSTCFG#LVD3V0</i>	;使能3.0V 时低压中断
	<i>SETB</i>	<i>ELVD</i>	;使能LVD 中断
	<i>SETB</i>	<i>EA</i>	
	<i>JMP</i>	<i>\$</i>	
 <i>END</i>			

C 语言代码

```

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

sfr     RSTCFG      = 0xff;
#define  ENLVR       0x40      //RSTCFG.6
#define  LVD2V2      0x00      //LVD@2.2V
#define  LVD2V4      0x01      //LVD@2.4V
#define  LVD2V7      0x02      //LVD@2.7V
#define  LVD3V0      0x03      //LVD@3.0V

```

```

sbit    ELVD      =  IE^6;
#define  LVDF      0x20      //PCON.5
sbit    P32       =  P3^2;

void Lvd_Isr() interrupt 6
{
    PCON &= ~LVDF;           //清中断标志
    P32 = ~P32;              //测试端口
}

void main()
{
    PCON &= ~LVDF;           //测试端口
// RSTCFG = ENLVR / LVD3V0;   //使能3.0V时低压复位,不产生LVD中断
// RSTCFG = LVD3V0;          //使能3.0V时低压中断
    ELVD = 1;                //使能LVD中断
    EA = 1;

    while (1);
}

```

6.6.6 省电模式

汇编代码

```

VOCTRL    DATA      0BBH
IDL        EQU       01H
PD         EQU       02H
                  ;PCON.0
                  ;PCON.1

ORG        0000H
LJMP       MAIN
ORG        0003H
LJMP       INT0ISR

ORG        0100H
INT0ISR:
CPL        P3.4      ;测试端口
RETI

MAIN:
MOV        SP,#3FH

MOV        VOCTRL,#00H      ;掉电模式时使用内部SCC模块,功耗约1.5uA
;MOV        VOCTRL,#80H      ;掉电模式时使用外部SCC模块,功耗约0.15uA
SETB       EX0
SETB       EA
NOP
NOP
;MOV        PCON,#IDL      ;MCU进入IDLE模式
MOV        PCON,#PD       ;MCU进入掉电模式
NOP
NOP
CLR        P3.5      ;测试端口
JMP        $

```

END

C 语言代码

```

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

sfr VOCTRL = 0xbb;
#define IDL 0x01 //PCON.0
#define PD 0x02 //PCON.1
sbit P34 = P3^4;
sbit P35 = P3^5;

void INT0_Isr() interrupt 0
{
    P34 = ~P34; //测试端口
}

void main()
{
    VOCTRL = 0x00; //掉电模式时使用内部SCC模块,功耗约1.5uA
//    VOCTRL = 0x80; //掉电模式时使用外部SCC模块,功耗约0.15uA
    EX0 = 1; //使能INT0中断,用于唤醒MCU
    EA = 1;
    _nop_();
    _nop_();
    PCON = IDL; //MCU进入IDLE模式
//    PCON = PD; //MCU进入掉电模式
    _nop_();
    _nop_();
    P35 = 0;

    while (1);
}

```

6.6.7 使用INT0/INT1/INT2/INT3/INT4 中断唤醒MCU

汇编代码

INTCLKO	DATA	8FH
EX2	EQU	10H
EX3	EQU	20H
EX4	EQU	40H
	ORG	0000H
	LJMP	MAIN
	ORG	0003H
	LJMP	INT0ISR
	ORG	0013H
	LJMP	INT1ISR
	ORG	0053H
	LJMP	INT2ISR
	ORG	005BH
	LJMP	INT3ISR
	ORG	0083H
	LJMP	INT4ISR
	ORG	0100H
INT0ISR:	CPL	P1.0 ;测试端口
	RETI	
INT1ISR:		

	CPL	P1.0	;	测试端口
	RETI			
INT2ISR:				
	CPL	P1.0	;	测试端口
	RETI			
INT3ISR:				
	CPL	P1.0	;	测试端口
	RETI			
INT4ISR:				
	CPL	P1.0	;	测试端口
	RETI			
MAIN:				
	MOV	SP,#3FH		
	CLR	IT0	;	使能 INT0 上升沿和下降沿中断
;	SETB	IT0	;	使能 INT0 下降沿中断
	SETB	EX0	;	使能 INT0 中断
	CLR	IT1	;	使能 INT1 上升沿和下降沿中断
;	SETB	IT1	;	使能 INT1 下降沿中断
	SETB	EX1	;	使能 INT1 中断
	MOV	INTCLKO,#EX2	;	使能 INT2 下降沿中断
	ORL	INTCLKO,#EX3	;	使能 INT3 下降沿中断
	ORL	INTCLKO,#EX4	;	使能 INT4 下降沿中断
	SETB	EA		
	MOV	PCON,#02H	;	MCU 进入掉电模式
	NOP		;	掉电唤醒后立即进入中断服务程序
	NOP			
LOOP:				
	CPL	P1.I		
	JMP	LOOP		
	END			

C 语言代码

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

sfr INTCLKO = 0x8f;
#define EX2 0x10
#define EX3 0x20
#define EX4 0x40

sbit P10 = P1^0;
sbit P11 = P1^1;

void INT0_Isr() interrupt 0
{
    P10 = !P10; // 测试端口
}

void INT1_Isr() interrupt 2
{
    P10 = !P10; // 测试端口
}
```

```

}

void INT2_Isr() interrupt 10
{
    P10 = !P10;                                //测试端口
}

void INT3_Isr() interrupt 11
{
    P10 = !P10;                                //测试端口
}

void INT4_Isr() interrupt 16
{
    P10 = !P10;                                //测试端口
}

void main()
{
    IT0 = 0;                                    //使能INT0 上升沿和下降沿中断
//    IT0 = 1;                                    //使能INT0 下降沿中断
    EX0 = 1;                                    //使能INT0 中断

    IT1 = 0;                                    //使能INT1 上升沿和下降沿中断
//    IT1 = 1;                                    //使能INT1 下降沿中断
    EX1 = 1;                                    //使能INT1 中断

    INTCLK0 = EX2;                             //使能INT2 下降沿中断
    INTCLK0 |= EX3;                            //使能INT3 下降沿中断
    INTCLK0 |= EX4;                            //使能INT4 下降沿中断

    EA = 1;

    PCON = 0x02;                               //MCU 进入掉电模式
    _nop_();
    _nop_();

    while (1)
    {
        P11 = ~P11;
    }
}

```

6.6.8 使用T0/T1/T2/T3/T4 中断唤醒MCU

汇编代码

; 测试工作频率为 11.0592MHz

T2L	DATA	0D7H
T2H	DATA	0D6H
T3L	DATA	0D5H
T3H	DATA	0D4H
T4L	DATA	0D3H
T4H	DATA	0D2H
T4T3M	DATA	0D1H
AUXR	DATA	8EH
IE2	DATA	0AFH
ET2	EQU	04H

<i>ET3</i>	<i>EQU</i>	<i>20H</i>
<i>ET4</i>	<i>EQU</i>	<i>40H</i>
<i>AUXINTIF</i>	<i>DATA</i>	<i>0EFH</i>
<i>T2IF</i>	<i>EQU</i>	<i>01H</i>
<i>T3IF</i>	<i>EQU</i>	<i>02H</i>
<i>T4IF</i>	<i>EQU</i>	<i>04H</i>
	<i>ORG</i>	<i>0000H</i>
	<i>LJMP</i>	<i>MAIN</i>
	<i>ORG</i>	<i>000BH</i>
	<i>LJMP</i>	<i>TM0ISR</i>
	<i>ORG</i>	<i>001BH</i>
	<i>LJMP</i>	<i>TM1ISR</i>
	<i>ORG</i>	<i>0063H</i>
	<i>LJMP</i>	<i>TM2ISR</i>
	<i>ORG</i>	<i>009BH</i>
	<i>LJMP</i>	<i>TM3ISR</i>
	<i>ORG</i>	<i>00A3H</i>
	<i>LJMP</i>	<i>TM4ISR</i>
	<i>ORG</i>	<i>0100H</i>
<i>TM0ISR:</i>	<i>CPL</i>	<i>P1.0</i> ; 测试端口
	<i>RETI</i>	
<i>TM1ISR:</i>	<i>CPL</i>	<i>P1.0</i> ; 测试端口
	<i>RETI</i>	
<i>TM2ISR:</i>	<i>CPL</i>	<i>P1.0</i> ; 测试端口
	<i>ANL</i>	<i>AUXINTIF,#NOT T2IF</i> ; 清中断标志
	<i>RETI</i>	
<i>TM3ISR:</i>	<i>CPL</i>	<i>P1.0</i> ; 测试端口
	<i>ANL</i>	<i>AUXINTIF,#NOT T3IF</i> ; 清中断标志
	<i>RETI</i>	
<i>TM4ISR:</i>	<i>CPL</i>	<i>P1.0</i> ; 测试端口
	<i>ANL</i>	<i>AUXINTIF,#NOT T4IF</i> ; 清中断标志
	<i>RETI</i>	
<i>MAIN:</i>	<i>MOV</i>	<i>SP,#3FH</i>
	<i>MOV</i>	<i>TMOD,#00H</i>
	<i>MOV</i>	<i>TL0,#66H</i> ; 65536-11.0592M/12/1000
	<i>MOV</i>	<i>TH0,#0FCH</i>
	<i>SETB</i>	<i>TR0</i> ; 启动定时器
	<i>SETB</i>	<i>ET0</i> ; 使能定时器中断
	<i>MOV</i>	<i>TL1,#66H</i> ; 65536-11.0592M/12/1000
	<i>MOV</i>	<i>TH1,#0FCH</i>
	<i>SETB</i>	<i>TR1</i> ; 启动定时器
	<i>SETB</i>	<i>ET1</i> ; 使能定时器中断
	<i>MOV</i>	<i>T2L,#66H</i> ; 65536-11.0592M/12/1000
	<i>MOV</i>	<i>T2H,#0FCH</i>
	<i>MOV</i>	<i>AUXR,#10H</i> ; 启动定时器
	<i>MOV</i>	<i>IE2,#ET2</i> ; 使能定时器中断

MOV	T3L,#66H	;65536-11.0592M/12/1000
MOV	T3H,#0FCH	
MOV	T4T3M,#08H	;启动定时器
ORL	IE2,#ET3	;使能定时器中断
MOV	T4L,#66H	;65536-11.0592M/12/1000
MOV	T4H,#0FCH	
ORL	T4T3M,#80H	;启动定时器
ORL	IE2,#ET4	;使能定时器中断
SETB	EA	
MOV	PCON,#02H	;MCU 进入掉电模式
NOP		;掉电唤醒后不会立即进入中断服务程序,
		;而是等到定时器溢出后才会进入中断服务程序
NOP		
LOOP:		
CPL	P1.1	
JMP	LOOP	
END		

C 语言代码

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

//测试工作频率为 11.0592MHz

sfr T2L = 0xd7;
sfr T2H = 0xd6;
sfr T3L = 0xd5;
sfr T3H = 0xd4;
sfr T4L = 0xd3;
sfr T4H = 0xd2;
sfr T4T3M = 0xd1;
sfr AUXR = 0x8e;
sfr IE2 = 0xaf;
#define ET2 0x04
#define ET3 0x20
#define ET4 0x40
sfr AUXINTIF = 0xef;
#define T2IF 0x01
#define T3IF 0x02
#define T4IF 0x04

sbit P10 = P1^0;
sbit P11 = P1^1;

void TM0_Isr() interrupt 1
{
    P10 = !P10; //测试端口
}

void TM1_Isr() interrupt 3
{
    P10 = !P10; //测试端口
}
```

```

void TM2_Isr() interrupt 12
{
    P10 = !P10;                      //测试端口
    AUXINTIF &= ~T2IF;                //清中断标志
}

void TM3_Isr() interrupt 19
{
    P10 = !P10;                      //测试端口
    AUXINTIF &= ~T3IF;                //清中断标志
}

void TM4_Isr() interrupt 20
{
    P10 = !P10;                      //测试端口
    AUXINTIF &= ~T4IF;                //清中断标志
}

void main()
{
    TMOD = 0x00;                     //65536-11.0592M/12/1000
    TL0 = 0x66;                      //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                         //启动定时器
    ET0 = 1;                          //使能定时器中断

    TL1 = 0x66;                     //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                         //启动定时器
    ET1 = 1;                          //使能定时器中断

    T2L = 0x66;                     //65536-11.0592M/12/1000
    T2H = 0xfc;
    AUXR = 0x10;                     //启动定时器
    IE2 = ET2;                       //使能定时器中断

    T3L = 0x66;                     //65536-11.0592M/12/1000
    T3H = 0xfc;
    T4T3M = 0x08;                   //启动定时器
    IE2 |= ET3;                      //使能定时器中断

    T4L = 0x66;                     //65536-11.0592M/12/1000
    T4H = 0xfc;
    T4T3M |= 0x80;                  //启动定时器
    IE2 |= ET4;                      //使能定时器中断

    EA = 1;

    PCON = 0x02;                    //MCU 进入掉电模式
    _nop_();                        //掉电唤醒后不会立即进入中断服务程序,
                                    //而是等到定时器溢出后才会进入中断服务程序
    _nop_();

    while (1)
    {
        P11 = ~P11;
    }
}

```

6.6.9 使用RxD/RxD2/RxD3/RxD4 中断唤醒MCU

汇编代码

; 测试工作频率为 11.0592MHz

<i>IE2</i>	<i>DATA</i>	<i>0AFH</i>
<i>ES2</i>	<i>EQU</i>	<i>01H</i>
<i>ES3</i>	<i>EQU</i>	<i>08H</i>
<i>ES4</i>	<i>EQU</i>	<i>10H</i>

<i>P_SW1</i>	<i>DATA</i>	<i>0A2H</i>
<i>P_SW2</i>	<i>DATA</i>	<i>0BAH</i>

<i>ORG</i>	<i>0000H</i>
<i>LJMP</i>	<i>MAIN</i>
<i>ORG</i>	<i>0023H</i>
<i>LJMP</i>	<i>UART1ISR</i>
<i>ORG</i>	<i>0043H</i>
<i>LJMP</i>	<i>UART2ISR</i>
<i>ORG</i>	<i>008BH</i>
<i>LJMP</i>	<i>UART3ISR</i>
<i>ORG</i>	<i>0093H</i>
<i>LJMP</i>	<i>UART4ISR</i>

<i>ORG</i>	<i>0100H</i>
------------	--------------

<i>UART1ISR:</i>	<i>RETI</i>
<i>UART2ISR:</i>	<i>RETI</i>
<i>UART3ISR:</i>	<i>RETI</i>
<i>UART4ISR:</i>	<i>RETI</i>

<i>MAIN:</i>	<i>MOV</i>	<i>SP,#3FH</i>
--------------	------------	----------------

;	<i>MOV</i>	<i>P_SW1,#00H</i>	<i>;RxD/P3.0 下降沿唤醒</i>
;	<i>MOV</i>	<i>P_SW1,#40H</i>	<i>;RxD_2/P3.6 下降沿唤醒</i>
;	<i>MOV</i>	<i>P_SW1,#80H</i>	<i>;RxD_3/P1.6 下降沿唤醒</i>
;	<i>MOV</i>	<i>P_SW1,#0C0H</i>	<i>;RxD_4/P4.3 下降沿唤醒</i>

;	<i>MOV</i>	<i>P_SW2,#00H</i>	<i>;RxD2/P1.0 下降沿唤醒</i>
;	<i>MOV</i>	<i>P_SW2,#01H</i>	<i>;RxD2_2/P4.0 下降沿唤醒</i>

;	<i>MOV</i>	<i>P_SW2,#00H</i>	<i>;RxD3/P0.0 下降沿唤醒</i>
;	<i>MOV</i>	<i>P_SW2,#02H</i>	<i>;RxD3_2/P5.0 下降沿唤醒</i>

;	<i>MOV</i>	<i>P_SW2,#00H</i>	<i>;RxD4/P0.2 下降沿唤醒</i>
;	<i>MOV</i>	<i>P_SW2,#04H</i>	<i>;RxD4_2/P5.2 下降沿唤醒</i>

<i>SETB</i>	<i>ES</i>	<i>;使能串口中断</i>
<i>MOV</i>	<i>IE2,#ES2</i>	<i>;使能串口中断</i>
<i>ORL</i>	<i>IE2,#ES3</i>	<i>;使能串口中断</i>
<i>ORL</i>	<i>IE2,#ES4</i>	<i>;使能串口中断</i>
<i>SETB</i>	<i>EA</i>	

MOV	PCON,#02H	<i>;MCU 进入掉电模式</i>
NOP		<i>;掉电唤醒后不会进入中断服务程序,</i>
NOP		
LOOP:		
CPL	P1.1	
JMP	LOOP	
END		

C 语言代码

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

//测试工作频率为11.0592MHz

sfr IE2 = 0xaf;
#define ES2 0x01
#define ES3 0x08
#define ES4 0x10

sfr P_SW1 = 0xa2;
sfr P_SW2 = 0xba;

sbit P1I = P1^1;

void UART1_Isr() interrupt 4
{
}

void UART2_Isr() interrupt 8
{
}

void UART3_Isr() interrupt 17
{
}

void UART4_Isr() interrupt 18
{
}

void main()
{
    P_SW1 = 0x00; //RXD/P3.0 下降沿唤醒
    // P_SW1 = 0x40; //RXD_2/P3.6 下降沿唤醒
    // P_SW1 = 0x80; //RXD_3/P1.6 下降沿唤醒
    // P_SW1 = 0xc0; //RXD_4/P4.3 下降沿唤醒

    P_SW2 = 0x00; //RXD2/P1.0 下降沿唤醒
    // P_SW2 = 0x01; //RXD2_2/P4.0 下降沿唤醒

    P_SW2 = 0x00; //RXD3/P0.0 下降沿唤醒
    // P_SW2 = 0x02; //RXD3_2/P5.0 下降沿唤醒

    P_SW2 = 0x00; //RXD4/P0.2 下降沿唤醒
    // P_SW2 = 0x04; //RXD4_2/P5.2 下降沿唤醒
}
```

```

ES = I;           //使能串口中断
IE2 = ES2;        //使能串口中断
IE2 |= ES3;       //使能串口中断
IE2 |= ES4;       //使能串口中断
EA = I;           //使能中断

PCon = 0x02;      //MCU 进入掉电模式
_nop_();
_nop_();          //掉电唤醒后不会进入中断服务程序

while (1)
{
    PII = ~PII;
}
}

```

6.6.10 使用LVD中断唤醒MCU

汇编代码

RSTCFG	DATA	0FFH	
ENLVR	EQU	40H	;RSTCFG.6
LVD2V2	EQU	00H	;LVD@2.2V
LVD2V4	EQU	01H	;LVD@2.4V
LVD2V7	EQU	02H	;LVD@2.7V
LVD3V0	EQU	03H	;LVD@3.0V
ELVD	BIT	IE.6	
LVDF	EQU	20H	;PCon.5
	ORG	0000H	
	LJMP	MAIN	
	ORG	0033H	
	LJMP	LVDIR	
LVDIR:	ORG	0100H	
	ANL	PCon,#NOT LVDF	;清中断标志
	CPL	P1.0	;测试端口
	RETI		
MAIN:	MOV	SP,#3FH	
	ANL	PCon,#NOT LVDF	;上电需要清中断标志
	MOV	RSTCFG# LVD3V0	;设置LVD 电压为3.0V
	SETB	ELVD	;使能LVD 中断
	SETB	EA	
	MOV	PCon,#02H	;MCU 进入掉电模式
	NOP		;掉电唤醒后立即进入中断服务程序
	NOP		
LOOP:	CPL	P1.1	
	JMP	LOOP	
	END		

C 语言代码

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

sfr RSTCFG      = 0xff;
#define ENLVR        0x40    //RSTCFG.6
#define LVD2V2       0x00    //LVD@2.2V
#define LVD2V4       0x01    //LVD@2.4V
#define LVD2V7       0x02    //LVD@2.7V
#define LVD3V0       0x03    //LVD@3.0V
sbit ELVD        = IE^6;
#define LVDF         0x20    //PCON.5

sbit P10          = P1^0;
sbit P11          = P1^1;

void LVD_Isr() interrupt 6
{
    PCON &= ~LVDF;           //清中断标志
    P10 = !P10;              //测试端口
}

void main()
{
    PCON &= ~LVDF;          //上电需要清中断标志
    RSTCFG = LVD3V0;         //设置LVD 电压为3.0V
    ELVD = 1;                //使能LVD 中断
    EA = 1;

    PCON = 0x02;             //MCU 进入掉电模式
    _nop_();
    _nop_();                //掉电唤醒后立即进入中断服务程序

    while (1)
    {
        P11 = ~P11;
    }
}
```

6.6.11 使用CCP0/CCP1/CCP2/CCP3 中断唤醒MCU

汇编代码

; 测试工作频率为 11.0592MHz

CCON	DATA	0D8H
CF	BIT	CCON.7
CR	BIT	CCON.6
CCF3	BIT	CCON.3
CCF2	BIT	CCON.2
CCF1	BIT	CCON.1
CCF0	BIT	CCON.0
CMOD	DATA	0D9H
CL	DATA	0E9H
CH	DATA	0F9H
CCAPM0	DATA	0DAH
CCAP0L	DATA	0EAH
CCAP0H	DATA	0FAH

<i>PCA_PWM0</i>	<i>DATA</i>	<i>0F2H</i>
<i>CCAPM1</i>	<i>DATA</i>	<i>0DBH</i>
<i>CCAPIL</i>	<i>DATA</i>	<i>0EBH</i>
<i>CCAPIH</i>	<i>DATA</i>	<i>0FBH</i>
<i>PCA_PWM1</i>	<i>DATA</i>	<i>0F3H</i>
<i>CCAPM2</i>	<i>DATA</i>	<i>0DCH</i>
<i>CCAP2L</i>	<i>DATA</i>	<i>0ECH</i>
<i>CCAP2H</i>	<i>DATA</i>	<i>0FCH</i>
<i>PCA_PWM2</i>	<i>DATA</i>	<i>0F4H</i>
<i>CCAPM3</i>	<i>DATA</i>	<i>0DDH</i>
<i>CCAP3L</i>	<i>DATA</i>	<i>0EDH</i>
<i>CCAP3H</i>	<i>DATA</i>	<i>0FDH</i>
<i>PCA_PWM3</i>	<i>DATA</i>	<i>0F5H</i>
<i>P_SW1</i>	<i>DATA</i>	<i>0A2H</i>
	<i>ORG</i>	<i>0000H</i>
	<i>LJMP</i>	<i>MAIN</i>
	<i>ORG</i>	<i>003BH</i>
	<i>LJMP</i>	<i>PCAISR</i>
	<i>ORG</i>	<i>0100H</i>
<i>PCAISR:</i>		
	<i>ANL</i>	<i>CCON,#NOT 8FH</i>
	<i>CPL</i>	<i>P1.0</i> ;清中断标志
	<i>RETI</i>	；测试端口
<i>MAIN:</i>		
	<i>MOV</i>	<i>SP,#3FH</i>
;	<i>MOV</i>	<i>P_SW1,#00H</i> ; CCP0/P1.7, CCP1/P1.6, CCP2/P1.5, CCP3/P1.4
;	<i>MOV</i>	<i>P_SW1,#10H</i> ; CCP0_2/P2.3, CCP1_2/P2.4, CCP2_2/P2.5, CCP3_2/P2.6
;	<i>MOV</i>	<i>P_SW1,#20H</i> ; CCP0_3/P7.0, CCP1_3/P7.1, CCP2_3/P7.2, CCP3_3/P7.3
;	<i>MOV</i>	<i>P_SW1,#30H</i> ; CCP0_4/P3.3, CCP1_4/P3.2, CCP2_4/P3.1, CCP3_4/P3.0
	<i>MOV</i>	<i>CCON,#00H</i>
	<i>MOV</i>	<i>CMOD,#08H</i> ;PCA 时钟为系统时钟
	<i>MOV</i>	<i>CCAPM0,#31H</i> ;使能 CCP0 口边沿唤醒功能
	<i>MOV</i>	<i>CCAPM1,#31H</i> ;使能 CCP1 口边沿唤醒功能
	<i>MOV</i>	<i>CCAPM2,#31H</i> ;使能 CCP2 口边沿唤醒功能
	<i>MOV</i>	<i>CCAPM3,#31H</i> ;使能 CCP3 口边沿唤醒功能
	<i>SETB</i>	<i>CR</i> ;启动 PCA 计时器
	<i>SETB</i>	<i>EA</i>
	<i>MOV</i>	<i>PCON,#02H</i> ;MCU 进入掉电模式
	<i>NOP</i>	；掉电唤醒后立即进入中断服务程序
	<i>NOP</i>	
<i>LOOP:</i>		
	<i>CPL</i>	<i>P1.1</i>
	<i>JMP</i>	<i>LOOP</i>
	<i>END</i>	

C 语言代码

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

//测试工作频率为 11.0592MHz

```

sfr    CCON      = 0xd8;
sbit   CF        = CCON^7;
sbit   CR        = CCON^6;
sbit   CCF3      = CCON^3;
sbit   CCF2      = CCON^2;
sbit   CCF1      = CCON^1;
sbit   CCF0      = CCON^0;
sfr    CMOD      = 0xd9;
sfr    CL        = 0xe9;
sfr    CH        = 0xf9;
sfr    CCAPM0    = 0xda;
sfr    CCAP0L    = 0xea;
sfr    CCAP0H    = 0xfa;
sfr    PCA_PWM0  = 0xf2;
sfr    CCAPM1    = 0xdb;
sfr    CCAP1L    = 0xeb;
sfr    CCAP1H    = 0xfb;
sfr    PCA_PWM1  = 0xf3;
sfr    CCAPM2    = 0xdc;
sfr    CCAP2L    = 0xec;
sfr    CCAP2H    = 0xfc;
sfr    PCA_PWM2  = 0xf4;
sfr    CCAPM3    = 0xdd;
sfr    CCAP3L    = 0xed;
sfr    CCAP3H    = 0xfd;
sfr    PCA_PWM3  = 0xf5;

sfr    P_SWI     = 0xa2;
sbit   P10       = PI^0;
sbit   P11       = PI^1;

void PCA_Isr() interrupt 7
{
    CCON &= ~0x8f;           //清中断标志
    P10 = !P10;              //测试端口
}

void main()
{
    P_SWI = 0x00;            //CCP0/P1.7, CCP1/P1.6, CCP2/P1.5, CCP3/P1.4
//    P_SWI = 0x10;            //CCP0_2/P2.3, CCP1_2/P2.4, CCP2_2/P2.5, CCP3_2/P2.6
//    P_SWI = 0x20;            //CCP0_3/P7.0, CCP1_3/P7.1, CCP2_3/P7.2, CCP3_3/P7.3
//    P_SWI = 0x30;            //CCP0_4/P3.3, CCP1_4/P3.2, CCP2_4/P3.1, CCP3_4/P3.0

    CCON = 0x00;             //PCA 时钟为系统时钟
    CMOD = 0x08;              //使能 CCP0 口边沿唤醒功能
    CCAPM0 = 0x31;            //使能 CCP1 口边沿唤醒功能
    CCAPM1 = 0x31;            //使能 CCP2 口边沿唤醒功能
    CCAPM2 = 0x31;            //使能 CCP3 口边沿唤醒功能
    CCAPM3 = 0x31;            //使能 CCP3 口边沿唤醒功能

    CR = 1;                  //启动 PCA 计时器
    EA = 1;

    PCON = 0x02;              //MCU 进入掉电模式
    _nop_();
}

```

```

_nop_();
while (1)
{
    PII = ~PII;
}

```

6.6.12 CMP中断唤醒MCU

汇编代码

CMPCR1	DATA	0E6H	
CMPCR2	DATA	0E7H	
	ORG	0000H	
	LJMP	MAIN	
	ORG	00ABH	
	LJMP	CMPISR	
	ORG	0100H	
CMPISR:	ANL	CMPCR1,#NOT 40H	;清中断标志
	CPL	P1.0	;测试端口
	RETI		
MAIN:	MOV	SP,#3FH	
	MOV	CMPCR2,#00H	
	MOV	CMPCR1,#80H	;使能比较器模块
	ORL	CMPCR1,#30H	;使能比较器边沿中断
	ANL	CMPCR1,#NOT 08H	;P3.6 为 CMP+ 输入脚
	ORL	CMPCR1,#04H	;P3.7 为 CMP- 输入脚
	ORL	CMPCR1,#02H	;使能比较器输出
	SETB	EA	
	MOV	PCON,#02H	;MCU 进入掉电模式
	NOP		;掉电唤醒后立即进入中断服务程序
	NOP		
LOOP:	CPL	P1.1	
	JMP	LOOP	
	END		

C 语言代码

```

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

sfr CMPCR1      = 0xe6;
sfr CMPCR2      = 0xe7;

sbit P10         = P1^0;
sbit PII          = P1^1;

```

```

void CMP_Isr() interrupt 21
{
    CMPCR1 &= ~0x40;           //清中断标志
    P10 = !P10;                //测试端口
}

void main()
{
    CMPCR2 = 0x00;            //使能比较器模块
    CMPCR1 = 0x80;            //使能比较器边沿中断
    CMPCR1 |= 0x30;           //P3.6 为 CMP+ 输入脚
    CMPCR1 &= ~0x08;          //P3.7 为 CMP- 输入脚
    CMPCR1 |= 0x04;           //使能比较器输出
    CMPCR1 |= 0x02;           //EA = 1;

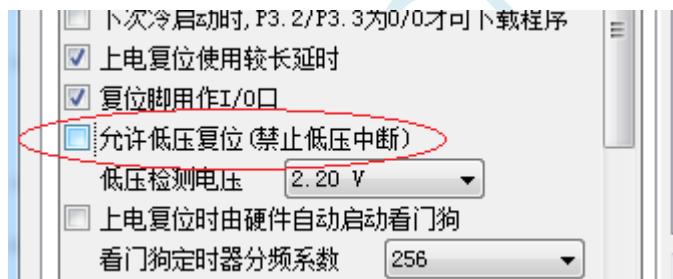
    PCON = 0x02;               //MCU 进入掉电模式
    _nop_();
    _nop_();

    while (1)
    {
        P11 = ~P11;
    }
}

```

6.6.13 使用LVD功能检测工作电压（电池电压）

若需要使用 LVD 功能检测电池电压，则在 ISP 下载时需要将低压复位功能去掉，如下图“允许低压复位（禁止低压中断）”的硬件选项的勾选项需要去掉



汇编代码

```

RSTCFG    DATA    0FFH
LVD2V2    EQU     00H           ;LVD@2.2V
LVD2V4    EQU     01H           ;LVD@2.4V
LVD2V7    EQU     02H           ;LVD@2.7V
LVD3V0    EQU     03H           ;LVD@3.0V
LVDF      EQU     20H           ;PCON.5

        ORG     0000H
        JMP     MAIN

MAIN:
        ORG     0100H
        ANL     PCON,#NOT LVDF
        MOV     RSTCFG,#LVD3V0

LOOP:

```

<i>MOV</i>	<i>B,#0FH</i>
<i>MOV</i>	<i>RSTCFG,#LVD3V0</i>
<i>CALL</i>	<i>DELAY</i>
<i>ANL</i>	<i>PCON,#NOT LVDF</i>
<i>CALL</i>	<i>DELAY</i>
<i>MOV</i>	<i>A,PCON</i>
<i>ANL</i>	<i>A,#LVDF</i>
<i>JZ</i>	<i>SKIP</i>
<i>MOV</i>	<i>A,B</i>
<i>CLR</i>	<i>C</i>
<i>RRC</i>	<i>A</i>
<i>MOV</i>	<i>B,A</i>
<i>MOV</i>	<i>RSTCFG,#LVD2V7</i>
<i>CALL</i>	<i>DELAY</i>
<i>ANL</i>	<i>PCON,#NOT LVDF</i>
<i>CALL</i>	<i>DELAY</i>
<i>MOV</i>	<i>A,PCON</i>
<i>ANL</i>	<i>A,#LVDF</i>
<i>JZ</i>	<i>SKIP</i>
<i>MOV</i>	<i>A,B</i>
<i>CLR</i>	<i>C</i>
<i>RRC</i>	<i>A</i>
<i>MOV</i>	<i>B,A</i>
<i>MOV</i>	<i>RSTCFG,#LVD2V4</i>
<i>CALL</i>	<i>DELAY</i>
<i>ANL</i>	<i>PCON,#NOT LVDF</i>
<i>CALL</i>	<i>DELAY</i>
<i>MOV</i>	<i>A,PCON</i>
<i>ANL</i>	<i>A,#LVDF</i>
<i>JZ</i>	<i>SKIP</i>
<i>MOV</i>	<i>A,B</i>
<i>CLR</i>	<i>C</i>
<i>RRC</i>	<i>A</i>
<i>MOV</i>	<i>B,A</i>
<i>MOV</i>	<i>RSTCFG,#LVD2V2</i>
<i>CALL</i>	<i>DELAY</i>
<i>ANL</i>	<i>PCON,#NOT LVDF</i>
<i>CALL</i>	<i>DELAY</i>
<i>MOV</i>	<i>A,PCON</i>
<i>ANL</i>	<i>A,#LVDF</i>
<i>JZ</i>	<i>SKIP</i>
<i>MOV</i>	<i>A,B</i>
<i>CLR</i>	<i>C</i>
<i>RRC</i>	<i>A</i>
<i>MOV</i>	<i>B,A</i>
<i>SKIP:</i>	
<i>MOV</i>	<i>A,B</i>
<i>CPL</i>	<i>A</i>
<i>MOV</i>	<i>P2,A</i>
	<i>;P2.3~P2.0 显示电池电量</i>
<i>JMP</i>	<i>LOOP</i>
<i>DELAY:</i>	
<i>MOV</i>	<i>R0,#100</i>
<i>NEXT:</i>	

NOP
NOP
NOP
NOP
DJNZ R0,NEXT
RET

END

C 语言代码

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

#define FOSC          24000000UL
#define TIMS          (65536 - FOSC/4/100)

sfr     RSTCFG      = 0xff;
#define LVD2V2        0x00    //LVD@2.2V
#define LVD2V4        0x01    //LVD@2.4V
#define LVD2V7        0x02    //LVD@2.7V
#define LVD3V0        0x03    //LVD@3.0V

#define LVDF          0x20    //PCON.5

void delay()
{
    int i;

    for (i=0; i<100; i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
}

void main()
{
    unsigned char power;

    PCON &= ~LVDF;
    RSTCFG = LVD3V0;

    while (1)
    {
        power = 0x0f;

        RSTCFG = LVD3V0;
        delay();
        PCON &= ~LVDF;
        delay();
        if (PCON & LVDF)
        {
            power >>= 1;

            RSTCFG = LVD2V7;
            delay();
        }
    }
}
```

```
PCON &= ~LVDF;
delay();
if (PCON & LVDF)
{
    power >>= 1;

    RSTCFG = LVD2V4;
    delay();
    PCON &= ~LVDF;
    delay();
    if (PCON & LVDF)
    {
        power >>= 1;

        RSTCFG = LVD2V2;
        delay();
        PCON &= ~LVDF;
        delay();
        if (PCON & LVDF)
        {
            power >>= 1;
        }
    }
}

RSTCFG = LVD3V0;
P2 = ~power; //P2.3~P2.0 显示电池电量
}
```

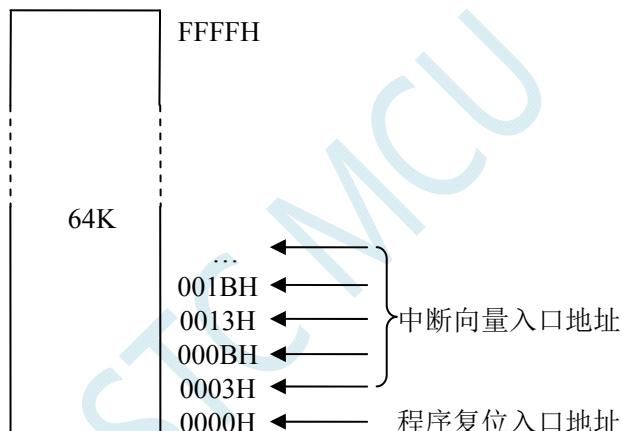
7 存储器

STC8 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。由于没有提供访问外部程序存储器的总线，所有单片机的所有程序存储器都是片上 Flash 存储器，不能访问外部程序存储器。

STC8 系列单片机内部集成了大容量的数据存储器，STC8A8K64S4A12 系列单片机内部有 8192+256 字节的数据存储器、STC8A4K64S2A12 系列单片机内部有 4096+256 字节的数据存储器、STC8F2K64S4 系列单片机内部有 2048+256 字节的数据存储器、STC8F2K64S2 系列单片机内部有 2048+256 字节的数据存储器。STC8 系列单片机内部的数据存储器在物理和逻辑上都分为两个地址空间：内部 RAM(256 字节)和内部扩展 RAM。其中内部 RAM 的高 128 字节的数据存储器与特殊功能寄存器(SFRs)地址重叠，实际使用时通过不同的寻址方式加以区分。另外，STC8 系列封装管脚数为 40 及其以上的单片机还可以访问在片外扩展的 64KB 外部数据存储器。

7.1 程序存储器

程序存储器用于存放用户程序、数据以及表格等信息。STC8 系列单片机内部集成了 64K 字节的 Flash 程序存储器。



单片机复位后，程序计数器(PC)的内容为 0000H，从 0000H 单元开始执行程序。另外中断服务程序的入口地址(又称中断向量)也位于程序存储器单元。在程序存储器中，每个中断都有一个固定的入口地址，当中断发生并得到响应后，单片机就会自动跳转到相应的中断入口地址去执行程序。外部中断 0 (INT0) 的中断服务程序的入口地址是 0003H，定时器/计数器 0 (TIMER0) 中断服务程序的入口地址是 000BH，外部中断 1 (INT1) 的中断服务程序的入口地址是 0013H，定时器/计数器 1 (TIMER1) 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)请参考中断介绍章节。

由于相邻中断入口地址的间隔区间仅有 8 个字节，一般情况下无法保存完整的中断服务程序，因此在中断响应的地址区域存放一条无条件转移指令，指向真正存放中断服务程序的空间去执行。

STC8 系列单片机中都包含有 Flash 数据存储器 (EEPROM)。以字节为单位进行读/写数据，以 512 字节为页单位进行擦除，可在线反复编程擦写 10 万次以上，提高了使用的灵活性和方便性。

7.2 数据存储器

STC8 系列单片机内部集成的 RAM 可用于存放程序执行的中间结果和过程数据。STC8A8K64S4A12 系列和 STC8F2K64S4 系列内部集成的 RAM 有如下差异:

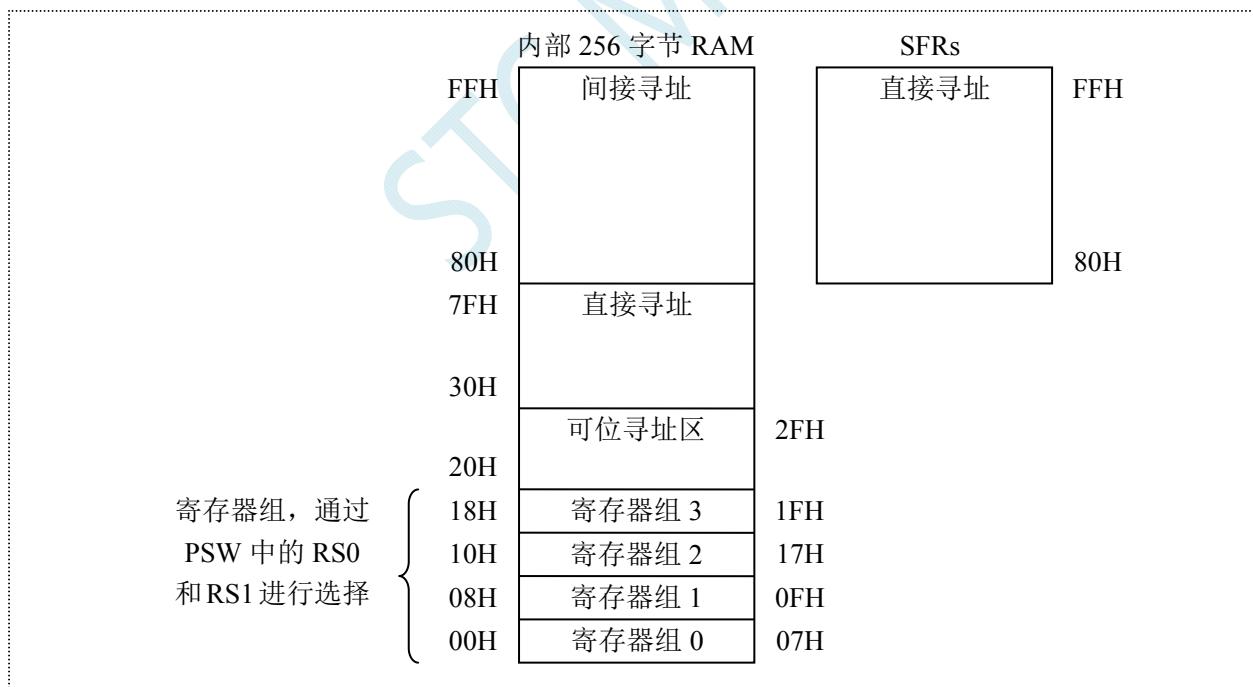
单片机系列	内部直接访问 RAM (DATA)	内部直接访问 RAM (IDATA)	内部扩展 RAM (XDATA)
STC8A8K64S4A12 系列	128 字节	128 字节	8192 字节
STC8A4K64S2A12 系列	128 字节	128 字节	4096 字节
STC8F2K64S4 系列	128 字节	128 字节	2048 字节
STC8F2K64S2 系列	128 字节	128 字节	2048 字节

此外, STC8 系列封装管脚数为 40 及其以上的单片机还可以访问在片外扩展的 64KB 外部数据存储器。

7.2.1 内部RAM

内部 RAM 共 256 字节, 可分为 2 个部分: 低 128 字节 RAM 和高 128 字节 RAM。低 128 字节的数据存储器与传统 8051 兼容, 既可直接寻址也可间接寻址。高 128 字节 RAM(在 8052 中扩展了高 128 字节 RAM)与特殊功能寄存器区共用相同的逻辑地址, 都使用 80H~FFH, 但在物理上是分别独立的, 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:



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 32 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。

PSW (程序状态寄存器)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PSW	D0H	CY	AC	F0	RS1	RS0	OV	-	P

RS1, RS0: 工作寄存器选择位

RS1	RS0	工作寄存器组 (R0~R7)
0	0	第 0 组 (00H~07H)
0	1	第 1 组 (08H~0FH)
1	0	第 2 组 (10H~17H)
1	1	第 3 组 (18H~1FH)

可位寻址区的地址从 20H ~ 2FH 共 16 个字节单元。20H~2FH 单元既可像普通 RAM 单元一样按字节存取，也可以对单元中的任何一位单独存取，共 128 位，所对应的逻辑位地址范围是 00H~7FH。位地址范围是 00H~7FH，内部 RAM 低 128 字节的地址也是 00H~7FH，从外表看，二者地址是一样的，实际上二者具有本质的区别；位地址指向的是一个位，而字节地址指向的是一个字节单元，在程序中使用不同的指令区分。

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。

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。STC8 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。

7.2.2 内部扩展RAM

STC8 系列单片机片内除了集成 256 字节的内部 RAM 外，还集成了内部的扩展 RAM。访问内部扩展 RAM 的方法和传统 8051 单片机访问外部扩展 RAM 的方法相同，但是不影响 P0 口(数据总线和高八位地址总线)、P2 口(低八位地址总线)、以及 RD、WR 和 ALE 等端口上的信号。

在汇编语言中，内部扩展 RAM 通过 MOVX 指令访问，

```
MOVX    A,@DPTR
MOVX    @DPTR,A
MOVX    A,@Ri
MOVX    @Ri,A
```

在 C 语言中，可使用 xdata/pdata 声明存储类型即可。如：

```
unsigned char xdata i;
unsigned int pdata j;
```

注：pdata 即为 xdata 的低 256 字节，在 C 语言中订阅变量为 pdata 类型后，编译器会自动将变量分配在 XDATA 的 0000H~00FFH 区域，并使用 MOVX @Ri,A 和 MOVX A@Ri 进行访问。

单片机内部扩展 RAM 是否可以访问，受辅助寄存器 AUXR 中的 EXTRAM 位控制。

AUXR (辅助寄存器)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
----	----	----	----	----	----	----	----	----	----

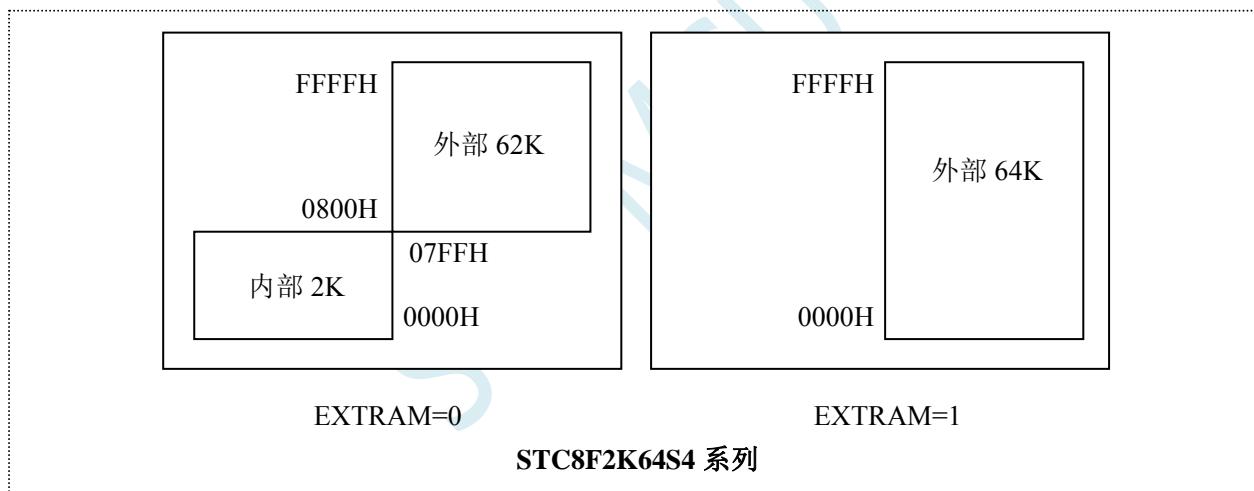
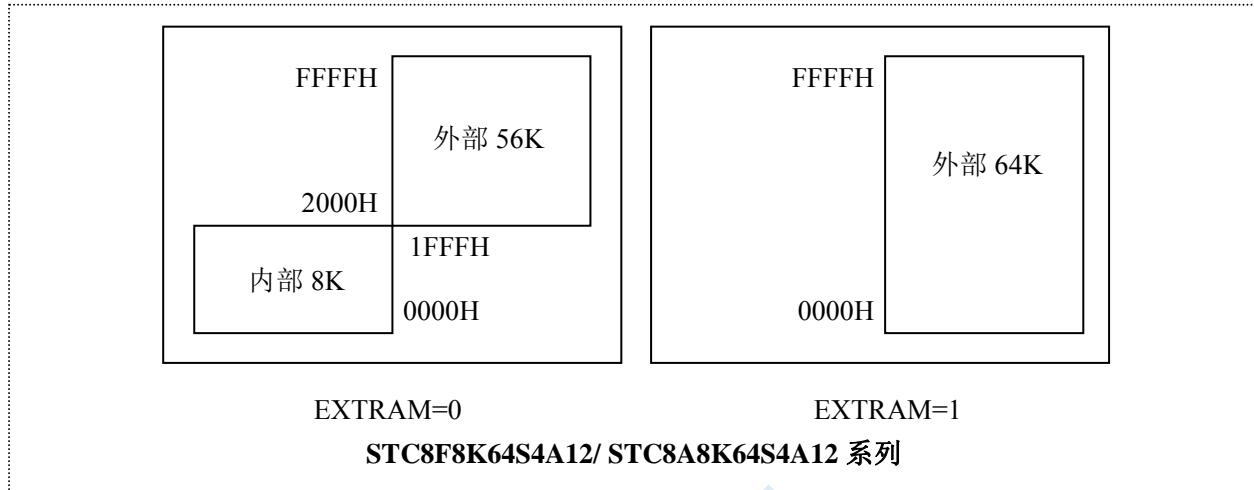
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2
------	-----	-------	-------	-----------	-----	--------	-------	---------------	-------

EXTRAM: 扩展 RAM 访问控制

0: 访问内部扩展 RAM。

当访问地址超出内部扩展 RAM 的地址时, 系统会自动切换到外部扩展 RAM

1: 访问外部扩展 RAM, 内部扩展 RAM 被禁用。



7.2.3 外部扩展RAM

STC8 系列封装管脚数为 40 及其以上的单片机具有扩展 64KB 外部数据存储器的能力。访问外部数据存储器期间, WR/RD/ALE 信号要有效。STC8 系列单片机新增了一个控制外部 64K 字节数据总线速度的特殊功能寄存器 BUS_SPEED, 说明如下:

BUS_SPEED (总线速度控制寄存器)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
BUS_SPEED	A1H	RW_S[1:0]							SPEED[1:0]

RW_S[1:0]: RD/RW 控制线选择位

00: P4.4 为 RD, P4.3 为 WR

01: P3.7 为 RD, P3.6 为 WR

10: P4.2 为 RD, P4.0 为 WR

11: 保留

SPEED[1:0]: 总线读写速度控制 (读写数据时控制信号和数据信号的准备时间和保持时间)

00: 1 个时钟

01: 2 个时钟

10: 4 个时钟

11: 8 个时钟

7.3 存储器中的特殊参数

STC8 系列单片机内部的数据存储器和程序存储器中保存有与芯片相关的一些特殊参数，包括：全球唯一 ID 号、32K 掉电唤醒定时器的频率、内部 Bandgap 电压值以及 IRC 参数。

这些参数在程序存储器（ROM）中的存放地址分别如下：

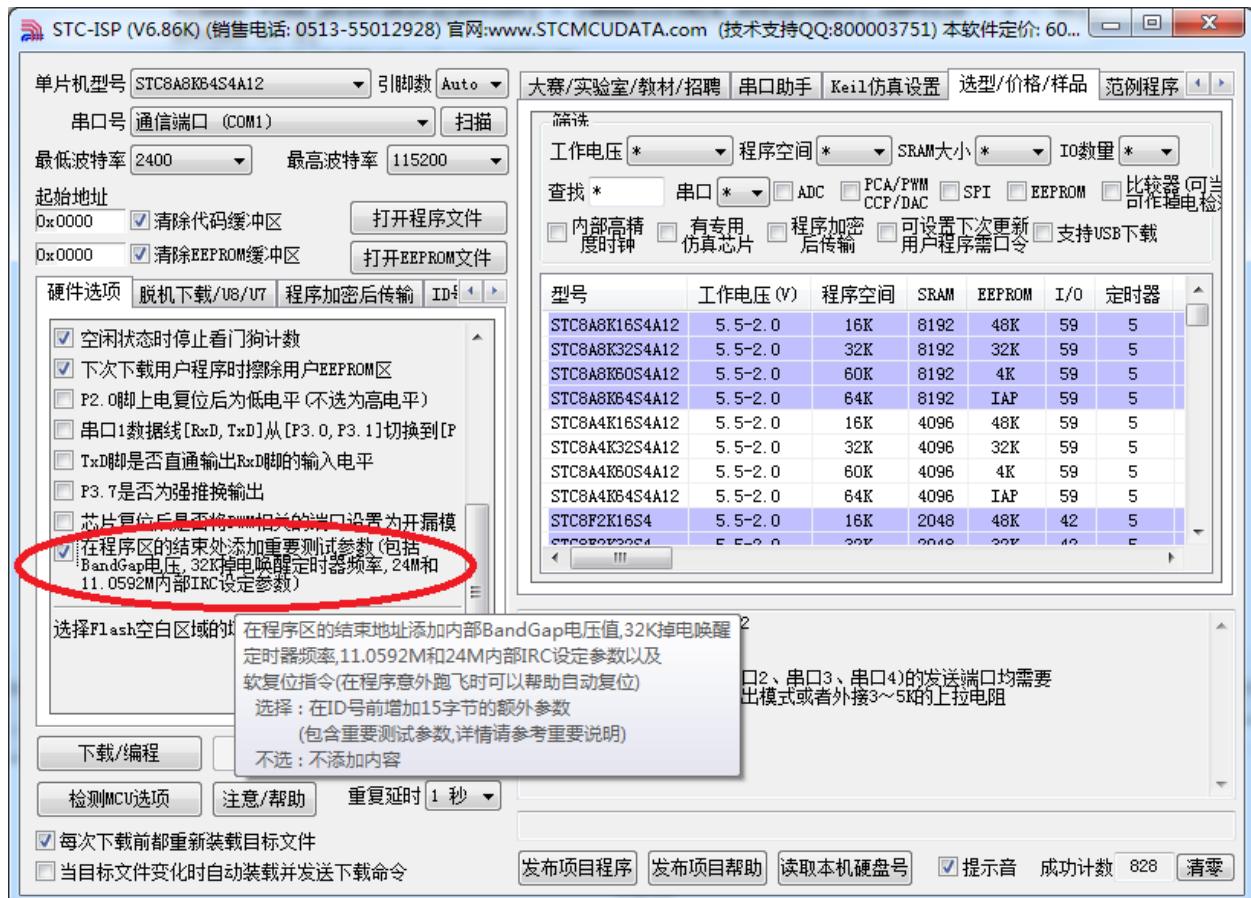
参数名称	保存地址				参数说明
	STC8A8K16S4A12 STC8A4K16S4A12 STC8F2K16S4 STC8F2K16S2	STC8A8K32S4A12 STC8A4K32S4A12 STC8F2K32S4 STC8F2K32S2	STC8A8K60S4A12 STC8A4K60S4A12 STC8F2K60S4 STC8F2K60S2	STC8A8K64S4A12 STC8A4K64S4A12 STC8F2K64S4 STC8F2K64S2	
全球唯一 ID 号	3FF9H~3FFFH	7FF9H~7FFFH	0EFF9H~0EFFFH	0FDF9H~0FDFFFH	7 字节
Bandgap 电压值	3FF7H~3FF8H	7FF7H~7FF8H	0EFF7H~0EFF8H	0FDF7H~0FDF8H	电压单位为毫伏
32K 掉电唤醒定时器的频率	3FF5H~3FF6H	7FF5H~7FF6H	0EFF5H~0EFF6H	0FDF5H~0FDF6H	单位 Hz
22.1184MHz 的 IRC 参数	3FF4H	7FF4H	0EFF4H	0FDF4H	—
24MHz 的 IRC 参数	3FF3H	7FF3H	0EFF3H	0FDF3H	—

这些参数在数据存储器（RAM）中的存放地址分别如下：

参数名称	保存地址	参数说明
Bandgap 电压值	idata: 0EFH~0F0H	电压单位为毫伏，高字节在前
全球唯一 ID 号	idata: 0F1H~0F7H	7 字节
32K 掉电唤醒定时器的频率	idata: 0F8H~0F9H	单位 Hz，高字节在前
22.1184MHz 的 IRC 参数	idata: 0FAH	—
24MHz 的 IRC 参数	idata: 0FBH	—

特别说明

- 由于 RAM 中的参数可能被修改，所以一般不建议用户使用，特别是用户使用 ID 号进行加密时，强烈建议用于读取 ROM 中的 ID 数据。
- 由于 STC8A8K64S4A10、STC8A4K64S2A10、STC8F2K64S4 和 STC8F2K64S2 这 4 个型号的 EEPROM 的大小用户是可以自己设置的，有可能将保存重要参数的 ROM 空间设置为 EEPROM 而人为的将重要参数擦除或修改，所以使用这 4 个型号进行 ID 号进行加密时可能需要考虑这个问题。
- 默认情况下，程序存储器中只有全球唯一 ID 号的数据，而 Bandgap 电压值、32K 掉电唤醒定时器的频率以及 IRC 参数都是没有的，需要在 ISP 下载时选择如下图所示的选项才可用。



7.3.1 读取Bandgap电压值 (从ROM中读取)

汇编代码

```

AUXR      DATA      8EH
BGV       EQU       0FDF7H ;STC8A8K64S4A10
;BGV       EQU       0EFF7H ;STC8A8K60S4A10
;BGV       EQU       07FF7H ;STC8A8K32S4A10
;BGV       EQU       03FF7H ;STC8A8K16S4A10

BUSY      BIT       20H.0

ORG      0000H
LJMP     MAIN

```

ORG 0023H
LJMP *UART_ISR*

ORG 0100H

UART_ISR:

JNB TI,CHKRI
CLR TI
CLR BUSY
CHKRI:
JNB RI,UARTISR_EXIT
CLR RI

UARTISR_EXIT:

RETI

UART_INIT:

MOV SCON,#50H
MOV TMOD,#00H
MOV TLI,#0E8H ;65536-11059200/115200/4=0FFE8H
MOV THI,#0FFH
SETB TRI
MOV AUXR,#40H
CLR BUSY
RET

UART_SEND:

JB BUSY,\$
SETB BUSY
MOV SBUFA,A
RET

MAIN:

MOV SP,#3FH
LCALL *UART_INIT*
SETB ES
SETB EA
MOV DPTR,#BGV
CLR A
MOVC A,@A+DPTR ;读取 Bandgap 电压的高字节
LCALL *UART_SEND*
MOV A,#1
MOVC A,@A+DPTR ;读取 Bandgap 电压的低字节
LCALL *UART_SEND*

LOOP:

JMP LOOP

END

C 语言代码

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

#define FOSC 11059200UL
#define BRT (65536 - FOSC / 115200 / 4)
```

```

sfr      AUXR      = 0x8e;
bit      busy;
int     *BGV;

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void main()
{
    BGV = (int code *)0xfd7;           // STC8A8K64S4A10
//    BGV = (int code *)0xeff;          // STC8A8K60S4A10
//    BGV = (int code *)0x7ff;          // STC8A8K32S4A10
//    BGV = (int code *)0x3ff;          // STC8A8K16S4A10
    UartInit();
    ES = 1;
    EA = 1;
    UartSend(*BGV >> 8);           // 读取 Bandgap 电压的高字节
    UartSend(*BGV);                 // 读取 Bandgap 电压的低字节

    while (1);
}

```

7.3.2 读取Bandgap电压值 (从RAM中读取)

汇编代码

AUXR	DATA	8EH
BGV	DATA	0EFH

BUSY	BIT	20H.0	
	<i>ORG</i>	<i>0000H</i>	
	<i>LJMP</i>	<i>MAIN</i>	
	<i>ORG</i>	<i>0023H</i>	
	<i>LJMP</i>	<i>UART_ISR</i>	
	<i>ORG</i>	<i>0100H</i>	
UART_ISR:			
	<i>JNB</i>	<i>TI,CHKRI</i>	
	<i>CLR</i>	<i>TI</i>	
	<i>CLR</i>	<i>BUSY</i>	
CHKRI:			
	<i>JNB</i>	<i>RI,UARTISR_EXIT</i>	
	<i>CLR</i>	<i>RI</i>	
UARTISR_EXIT:			
		<i>RETI</i>	
UART_INIT:			
	<i>MOV</i>	<i>SCON,#50H</i>	
	<i>MOV</i>	<i>TMOD,#00H</i>	
	<i>MOV</i>	<i>TLI,#0E8H</i>	<i>;65536-11059200/115200/4=0FFE8H</i>
	<i>MOV</i>	<i>THI,#0FFH</i>	
	<i>SETB</i>	<i>TR1</i>	
	<i>MOV</i>	<i>AUXR,#40H</i>	
	<i>CLR</i>	<i>BUSY</i>	
	<i>RET</i>		
UART_SEND:			
	<i>JB</i>	<i>BUSY,\$</i>	
	<i>SETB</i>	<i>BUSY</i>	
	<i>MOV</i>	<i>SBUFA,A</i>	
	<i>RET</i>		
MAIN:			
	<i>MOV</i>	<i>SP,#3FH</i>	
	<i>LCALL</i>	<i>UART_INIT</i>	
	<i>SETB</i>	<i>ES</i>	
	<i>SETB</i>	<i>EA</i>	
	<i>MOV</i>	<i>R0,#BGV</i>	
	<i>MOV</i>	<i>A,@R0</i>	<i>;读取 Bandgap 电压的高字节</i>
	<i>LCALL</i>	<i>UART_SEND</i>	
	<i>INC</i>	<i>R0</i>	
	<i>MOV</i>	<i>A,@R0</i>	<i>;读取 Bandgap 电压的低字节</i>
	<i>LCALL</i>	<i>UART_SEND</i>	
LOOP:			
	<i>JMP</i>	<i>LOOP</i>	
		<i>END</i>	

C 语言代码

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

```

#define  FOSC          11059200UL
#define  BRT           (65536 - FOSC / 115200 / 4)

sfr    AUXR         =  0x8e;

bit    busy;
int   *BGV;

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void main()
{
    BGV = (intidata *)0xef;
    UartInit();
    ES = 1;
    EA = 1;
    UartSend(*BGV >> 8);           //读取 Bandgap 电压的高字节
    UartSend(*BGV);                 //读取 Bandgap 电压的低字节

    while (1);
}

```

7.3.3 读取全球唯一ID号 (从ROM中读取)

汇编代码

AUXR	DATA	8EH
------	------	-----

ID	EQU	0FDF9H	; STC8A8K64S4A10
;ID	EQU	0EFF9H	; STC8A8K60S4A10
;ID	EQU	07FF9H	; STC8A8K32S4A10
;ID	EQU	03FF9H	; STC8A8K16S4A10
BUSY	BIT	20H.0	
	ORG	0000H	
	LJMP	MAIN	
	ORG	0023H	
	LJMP	UART_ISR	
	ORG	0100H	
UART_ISR:			
	JNB	TI,CHKRI	
	CLR	TI	
	CLR	BUSY	
CHKRI:			
	JNB	RI,UARTISR_EXIT	
	CLR	RI	
UARTISR_EXIT:			
	RETI		
UART_INIT:			
	MOV	SCON,#50H	
	MOV	TMOD,#00H	
	MOV	TLI,#0E8H	
	MOV	THI,#0FFH	
	SETB	TR1	
	MOV	AUXR,#40H	
	CLR	BUSY	
	RET		
UART_SEND:			
	JB	BUSY,\$	
	SETB	BUSY	
	MOV	SBUFA	
	RET		
MAIN:			
	MOV	SP,#3FH	
	LCALL	UART_INIT	
	SETB	ES	
	SETB	EA	
	MOV	DPTR,#ID	
	MOV	RI,#7	
NEXT:	CLR	A	
	MOVC	A,@A+DPTR	
	LCALL	UART_SEND	
	INC	DPTR	
	DJNZ	RI,NEXT	
LOOP:			
	JMP	LOOP	
	END		

C 语言代码

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

#define FOSC          11059200UL
#define BRT           (65536 - FOSC / 115200 / 4)

sfr AUXR = 0x8e;

bit busy;
char *ID;

void UartIsr() interrupt 4
{
    if(TI)
    {
        TI = 0;
        busy = 0;
    }
    if(RI)
    {
        RI = 0;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void main()
{
    char i;

    ID = (char code *)0xdf9;           // STC8A8K64S4A10
//    ID = (char code *)0xeff9;         // STC8A8K60S4A10
//    ID = (char code *)0x7ff9;         // STC8A8K32S4A10
//    ID = (char code *)0x3ff9;         // STC8A8K16S4A10
    UartInit();
    ES = 1;
    EA = 1;

    for (i=0; i<7; i++)
    {

```

```

    UartSend(ID[i]);
}

while (1);
}

```

7.3.4 读取全球唯一ID号 (从RAM中读取)

汇编代码

AUXR	DATA	8EH	
ID	DATA	0F1H	
BUSY	BIT	20H.0	
	ORG	0000H	
	LJMP	MAIN	
	ORG	0023H	
	LJMP	UART_ISR	
	ORG	0100H	
UART_ISR:			
	JNB	TI,CHKRI	
	CLR	TI	
	CLR	BUSY	
CHKRI:			
	JNB	RI,UARTISR_EXIT	
	CLR	RI	
UARTISR_EXIT:			
	RETI		
UART_INIT:			
	MOV	SCON,#50H	
	MOV	TMOD,#00H	
	MOV	TLI,#0E8H	;65536-11059200/115200/4=0FFE8H
	MOV	THI,#0FFH	
	SETB	TR1	
	MOV	AUXR,#40H	
	CLR	BUSY	
	RET		
UART_SEND:			
	JB	BUSY,\$	
	SETB	BUSY	
	MOV	SBUF,A	
	RET		
MAIN:			
	MOV	SP,#3FH	
	LCALL	UART_INIT	
	SETB	ES	
	SETB	EA	
	MOV	R0,#ID	
	MOV	RI,#7	

```

NEXT:      MOV      A,@R0
          LCALL    UART_SEND
          INC      R0
          DJNZ    RI,NEXT

LOOP:      JMP      LOOP

END

```

C 语言代码

```

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

#define FOSC           11059200UL
#define BRT            (65536 - FOSC / 115200 / 4)

sfr AUXR = 0x8e;

bit busy;
char *ID;

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void main()
{
    char i;

    ID = (char idata *)0xf1;
}

```

```

UartInit();
ES = 1;
EA = 1;

for (i=0; i<7; i++)
{
    UartSend(ID[i]);
}

while (1);
}

```

7.3.5 读取 32K掉电唤醒定时器的频率 (从ROM中读取)

汇编代码

AUXR	DATA	8EH	
F32K	EQU	0FDF5H	; STC8A8K64S4A10
;F32K	EQU	0EFF5H	; STC8A8K60S4A10
;F32K	EQU	07FF5H	; STC8A8K32S4A10
;F32K	EQU	03FF5H	; STC8A8K16S4A10
 BUSY	BIT	20H.0	
 UART_ISR:	ORG	0000H	
	LJMP	MAIN	
	ORG	0023H	
	LJMP	UART_ISR	
	ORG	0100H	
 CHKRI:	JNB	TI,CHKRI	
	CLR	TI	
	CLR	BUSY	
 UARTISR_EXIT:	JNB	RI,UARTISR_EXIT	
	CLR	RI	
 UART_INIT:	RETI		
 UART_SEND:	MOV	SCON,#50H	
	MOV	TMOD,#00H	
	MOV	T1I,#0E8H	;65536-11059200/115200/4=0FFE8H
	MOV	TH1,#0FFH	
	SETB	TR1	
	MOV	AUXR,#40H	
	CLR	BUSY	
	RET		
 MAIN:	MOV	SP,#3FH	

```

LCALL      UART_INIT
SETB       ES
SETB       EA

MOV        DPTR,#F32K
CLR        A
MOVC       A,@A+DPTR           ;读取32K频率的高字节
LCALL       UART_SEND
INC         DPTR
CLR         A
MOVC       A,@A+DPTR           ;读取32K频率的低字节
LCALL       UART_SEND

LOOP:
JMP        LOOP

END

```

C 语言代码

```

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

#define FOSC          11059200UL
#define BRT           (65536 - FOSC / 115200 / 4)

sfr AUXR = 0x8e;

bit busy;
int *F32K;

void UartIsr() interrupt 4
{
    if(TI)
    {
        TI = 0;
        busy = 0;
    }
    if(RI)
    {
        RI = 0;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TRI = 1;
    AUXR = 0x40;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
}

```

```

busy = 1;
SBUF = dat;
}

void main()
{
    F32K = (int code *)0xdfdf5;           // STC8A8K64S4A10
// F32K = (int code *)0xeaff5;           // STC8A8K60S4A10
// F32K = (int code *)0x7ff5;            // STC8A8K32S4A10
// F32K = (int code *)0x3ff5;            // STC8A8K16S4A10
UartInit();
ES = 1;
EA = 1;

UartSend(*F32K >> 8);                // 读取 32K 频率的高字节
UartSend(*F32K);                      // 读取 32K 频率的低字节

while (1);
}

```

7.3.6 读取 32K掉电唤醒定时器的频率 (从RAM中读取)

汇编代码

AUXR	DATA	8EH
F32K	DATA	0F8H
BUSY	BIT	20H.0
	ORG	0000H
	LJMP	MAIN
	ORG	0023H
	LJMP	UART_ISR
	ORG	0100H
UART_ISR:		
	JNB	TI,CHKRI
	CLR	TI
	CLR	BUSY
CHKRI:		
	JNB	RI,UARTISR_EXIT
	CLR	RI
UARTISR_EXIT:		
	RETI	
UART_INIT:		
	MOV	SCON,#50H
	MOV	TMOD,#00H
	MOV	TL1,#0E8H
	MOV	;65536-11059200/115200/4=0FFE8H
	MOV	TH1,#0FFH
	SETB	TR1
	MOV	AUXR,#40H
	CLR	BUSY
	RET	
UART_SEND:		
	JB	BUSY,\$
	SETB	BUSY

```

MOV      SBUF,A
RET

MAIN:
    MOV      SP,#3FH

    LCALL   UART_INIT
    SETB    ES
    SETB    EA

    MOV      R0,#F32K
    MOV      A,@R0          ;读取32K频率的高字节
    LCALL   UART_SEND
    INC     R0
    MOV      A,@R0          ;读取32K频率的低字节
    LCALL   UART_SEND

LOOP:
    JMP     LOOP

END

```

C 语言代码

```

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

#define FOSC      11059200UL
#define BRT       (65536 - FOSC / 115200 / 4)

sfr     AUXR      = 0x8e;

bit     busy;
int    *F32K;

void UartIsr() interrupt 4
{
    if(TI)
    {
        TI = 0;
        busy = 0;
    }
    if(RI)
    {
        RI = 0;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    busy = 0;
}

```

```

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void main()
{
    F32K = (intidata *)0xf8;
    UartInit();
    ES = 1;
    EA = 1;

    UartSend(*F32K >> 8);           //读取32K 频率的高字节
    UartSend(*F32K);                //读取32K 频率的低字节

    while (1);
}

```

7.3.7 用户自定义内部IRC频率 (从ROM中读取)

汇编代码

<i>P_SW2</i>	<i>DATA</i>	<i>0BAH</i>	
<i>CKSEL</i>	<i>EQU</i>	<i>0FE00H</i>	
<i>CLKDIV</i>	<i>EQU</i>	<i>0FE01H</i>	
<i>IRCCR</i>	<i>DATA</i>	<i>09FH</i>	
<i>IRC22M</i>	<i>EQU</i>	<i>0FDF4H</i>	<i>; STC8A8K64S4A10</i>
<i>IRC24M</i>	<i>EQU</i>	<i>0FDF3H</i>	
<i>;IRC22M</i>	<i>EQU</i>	<i>0EFF4H</i>	<i>; STC8A8K60S4A10</i>
<i>;IRC24M</i>	<i>EQU</i>	<i>0EFF3H</i>	
<i>;IRC22M</i>	<i>EQU</i>	<i>07FF4H</i>	<i>; STC8A8K32S4A10</i>
<i>;IRC24M</i>	<i>EQU</i>	<i>07FF3H</i>	
<i>;IRC22M</i>	<i>EQU</i>	<i>03FF4H</i>	<i>; STC8A8KI6S4A10</i>
<i>;IRC24M</i>	<i>EQU</i>	<i>03FF3H</i>	
	<i>ORG</i>	<i>0000H</i>	
	<i>LJMP</i>	<i>MAIN</i>	
	<i>ORG</i>	<i>0100H</i>	
<i>MAIN:</i>	<i>MOV</i>	<i>SP,#3FH</i>	
<i>;</i>	<i>MOV</i>	<i>DPTR,#IRC22M</i>	<i>;装载 22.1184MHz 的IRC 参数</i>
<i>;</i>	<i>CLR</i>	<i>A</i>	
<i>;</i>	<i>MOVC</i>	<i>A,@A+DPTR</i>	
<i>;</i>	<i>MOV</i>	<i>IRCCR,A</i>	
	<i>MOV</i>	<i>DPTR,#IRC24M</i>	<i>;装载 24MHz 的IRC 参数</i>
	<i>CLR</i>	<i>A</i>	
	<i>MOVC</i>	<i>A,@A+DPTR</i>	
	<i>MOV</i>	<i>IRCCR,A</i>	
	<i>MOV</i>	<i>P_SW2,#80H</i>	
	<i>MOV</i>	<i>A,#0</i>	<i>;主时钟不预分频</i>
	<i>MOV</i>	<i>DPTR,#CLKDIV</i>	
	<i>MOVX</i>	<i>@DPTR,A</i>	

```

MOV      A,#40H           ;主时钟4 分频输出到P5.4 口
MOV      DPTR,#CKSEL
MOVX    @DPTR,A
MOV      P_SW2,#00H

JMP      $

```

END

C 语言代码

```

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

#define CKSEL          (*(unsigned char volatile xdata *)0xfe00)
#define CLKDIV         (*(unsigned char volatile xdata *)0xfe01)

sfr    P_SW2        = 0xba;
sfr    IRCCR        = 0x9f;

char   *IRC22M;
char   *IRC24M;

void main()
{
    IRC22M = (char code *)0xfd4;           // STC8A8K64S4A10
    IRC24M = (char code *) 0xdf3;           // STC8A8K60S4A10
//    IRC22M = (char code *)0xeff4;          // STC8A8K32S4A10
//    IRC24M = (char code *) 0xeff3;          // STC8A8K32S4A10
//    IRC22M = (char code *)0x7ff4;          // STC8A8K16S4A10
//    IRC24M = (char code *) 0x7ff3;          // STC8A8K16S4A10
//    IRC22M = (char code *)0x3ff4;          // STC8A8K16S4A10
//    IRC24M = (char code *) 0x3ff3;          // STC8A8K16S4A10

//    IRCCR = *IRC22M;                      //装载22.1184MHz 的IRC 参数
//    IRCCR = *IRC24M;                      //装载24MHz 的IRC 参数

    P_SW2 = 0x80;                          //主时钟不预分频
    CLKDIV = 0;                            //主时钟4 分频输出到P5.4 口
    CKSEL = 0x40;
    P_SW2 = 0x00;

    while (1);
}

```

7.3.8 用户自定义内部IRC频率 (从RAM中读取)

汇编代码

<i>P_SW2</i>	<i>DATA</i>	<i>0BAH</i>
<i>CKSEL</i>	<i>EQU</i>	<i>0FE00H</i>
<i>CLKDIV</i>	<i>EQU</i>	<i>0FE01H</i>
<i>IRCCR</i>	<i>DATA</i>	<i>09FH</i>
<i>IRC22M</i>	<i>DATA</i>	<i>0FAH</i>
<i>IRC24M</i>	<i>DATA</i>	<i>0FBH</i>

<i>ORG</i>	<i>0000H</i>
<i>LJMP</i>	<i>MAIN</i>
<i>MAIN:</i>	<i>ORG</i> <i>0100H</i>
	<i>MOV</i> <i>SP,#3FH</i>
;	<i>MOV</i> <i>R0,#IRC22M</i>
;	<i>MOV</i> <i>IRCCR,@R0</i>
	<i>MOV</i> <i>R0,#IRC24M</i>
	<i>MOV</i> <i>IRCCR,@R0</i>
	<i>MOV</i> <i>P_SW2,#80H</i>
	<i>MOV</i> <i>A,#0</i>
	<i>MOV</i> <i>DPTR,#CLKDIV</i>
	<i>MOVX</i> <i>@DPTR,A</i>
	<i>MOV</i> <i>A,#40H</i>
	<i>MOV</i> <i>DPTR,#CKSEL</i>
	<i>MOVX</i> <i>@DPTR,A</i>
	<i>MOV</i> <i>P_SW2,#00H</i>
	<i>JMP</i> <i>\$</i>
 <i>END</i>	

C 语言代码

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

#define CKSEL      (*(unsigned char volatile xdata *)0xfe00)
#define CLKDIV     (*(unsigned char volatile xdata *)0xfe01)

sfr P_SW2      = 0xba;
sfr IRCCR      = 0x9f;

char *IRC22M;
char *IRC24M;

void main()
{
    IRC22M = (char idata *)0xfa;
    IRC24M = (char idata *) 0xfb;
//    IRCCR = *IRC22M;           //装载 22.1184MHz 的 IRC 参数
//    IRCCR = *IRC24M;           //装载 24MHz 的 IRC 参数

    P_SW2 = 0x80;
    CLKDIV = 0;                //主时钟不预分频
    CKSEL = 0x40;              //主时钟 4 分频输出到 P5.4 口
    P_SW2 = 0x00;

    while (1);
}
```

8 特殊功能寄存器

8.1 STC8A8K64S4A12 系列

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H	P7	CH	CCAP0H	CCAP1H	CCAP2H	CCAP3H	PWMCR	RSTCFG
F0H	B	PWMCFG	PCA_PWM0	PCA_PWM1	PCA_PWM2	PCA_PWM3	PWMIF	PWMFDCR
E8H	P6	CL	CCAP0L	CCAP1L	CCAP2L	CCAP3L	IP3H	AUXINTIF
E0H	ACC	P7M1	P7M0	DPS	DPL1	DPH1	CMPCCR1	CMPCCR2
D8H	CCON	CMOD	CCAPM0	CCAPM1	CCAPM2	CCAPM3	ADCCFG	IP3
D0H	PSW	T4T3M	T4H	T4L	T3H	T3L	T2H	T2L
C8H	P5	P5M1	P5M0	P6M1	P6M0	SPSTAT	SPCTL	SPDAT
C0H	P4	WDT_CONTR	IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP_CONTR
B8H	IP	SADEN	P_SW2	VOCTRL	ADC_CONTR	ADC_RES	ADC_RESL	ADCRESH
B0H	P3	P3M1	P3M0	P4M1	P4M0	IP2	IP2H	IPH
A8H	IE	SADDR	WKTC	WKTC	S3CON	S3BUF	TA	IE2
A0H	P2	BUS_SPEED	P_SW1	Reserved				Reserved
98H	SCON	SBUF	S2CON	S2BUF	Reserved		LIRTRIM	IRTRIM
90H	P1	P1M1	P1M0	P0M1	P0M0	P2M1	P2M0	
88H	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	INTCLKO
80H	P0	SP	DPL	DPH	S4CON	S4BUF		PCON

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
FFF0H	PWMCH	PWMCL	PWMCKS	TADCPH	TADCPL			
FF70H	PWM7T1H	PWM7T1L	PWM7T2H	PWM7T2L	PWM7CR	PWM7HLD		
FF60H	PWM6T1H	PWM6T1L	PWM6T2H	PWM6T2L	PWM6CR	PWM6HLD		
FF50H	PWM5T1H	PWM5T1L	PWM5T2H	PWM5T2L	PWM5CR	PWM5HLD		
FF40H	PWM4T1H	PWM4T1L	PWM4T2H	PWM4T2L	PWM4CR	PWM4HLD		
FF30H	PWM3T1H	PWM3T1L	PWM3T2H	PWM3T2L	PWM3CR	PWM3HLD		
FF20H	PWM2T1H	PWM2T1L	PWM2T2H	PWM2T2L	PWM2CR	PWM2HLD		
FF10H	PWM1T1H	PWM1T1L	PWM1T2H	PWM1T2L	PWM1CR	PWM1HLD		
FF00H	PWM0T1H	PWM0T1L	PWM0T2H	PWM0T2L	PWM0CR	PWM0HLD		
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE18H	P0NCS	P1NCS	P2NCS	P3NCS	P4NCS	P5NCS	P6NCS	P7NCS
FE10H	P0PU	P1PU	P2PU	P3PU	P4PU	P5PU	P6PU	P7PU
FE00H	CKSEL	CLKDIV	IRC24MCR	XOSCCR	IRC32KCR			

8.2 STC8A4K64S2A12 系列

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H	P7	CH	CCAP0H	CCAP1H	CCAP2H	CCAP3H	PWMCR	RSTCFG
F0H	B	PWMCFG	PCA_PWM0	PCA_PWM1	PCA_PWM2	PCA_PWM3	PWMIF	PWMFDCR
E8H	P6	CL	CCAP0L	CCAP1L	CCAP2L	CCAP3L	IP3H	AUXINTIF
E0H	ACC	P7M1	P7M0	DPS	DPL1	DPH1	CMPCCR1	CMPCCR2
D8H	CCON	CMOD	CCAPM0	CCAPM1	CCAPM2	CCAPM3	ADCCFG	IP3
D0H	PSW	T4T3M	T4H	T4L	T3H	T3L	T2H	T2L
C8H	P5	P5M1	P5M0	P6M1	P6M0	SPSTAT	SPCTL	SPDAT
C0H	P4	WDT_CONTR	IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP CONTR
B8H	IP	SADEN	P_SW2	VOCTRL	ADC_CONTR	ADC_RES	ADC_RESL	ADCRESH
B0H	P3	P3M1	P3M0	P4M1	P4M0	IP2	IP2H	IPH
A8H	IE	SADDR	WKTC	WKTC			TA	IE2
A0H	P2	BUS_SPEED	P_SW1	Reserved				Reserved
98H	SCON	SBUF	S2CON	S2BUF	Reserved		LIRTRIM	IRTRIM
90H	P1	P1M1	P1M0	P0M1	P0M0	P2M1	P2M0	
88H	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	INTCLKO
80H	P0	SP	DPL	DPH				PCON

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
FFF0H	PWMCH	PWMCL	PWMCKS	TADCPH	TADCPL			
FF70H	PWM7T1H	PWM7T1L	PWM7T2H	PWM7T2L	PWM7CR	PWM7HLD		
FF60H	PWM6T1H	PWM6T1L	PWM6T2H	PWM6T2L	PWM6CR	PWM6HLD		
FF50H	PWM5T1H	PWM5T1L	PWM5T2H	PWM5T2L	PWM5CR	PWM5HLD		
FF40H	PWM4T1H	PWM4T1L	PWM4T2H	PWM4T2L	PWM4CR	PWM4HLD		
FF30H	PWM3T1H	PWM3T1L	PWM3T2H	PWM3T2L	PWM3CR	PWM3HLD		
FF20H	PWM2T1H	PWM2T1L	PWM2T2H	PWM2T2L	PWM2CR	PWM2HLD		
FF10H	PWM1T1H	PWM1T1L	PWM1T2H	PWM1T2L	PWM1CR	PWM1HLD		
FF00H	PWM0T1H	PWM0T1L	PWM0T2H	PWM0T2L	PWM0CR	PWM0HLD		
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE18H	P0NCS	P1NCS	P2NCS	P3NCS	P4NCS	P5NCS	P6NCS	P7NCS
FE10H	P0PU	P1PU	P2PU	P3PU	P4PU	P5PU	P6PU	P7PU
FE00H	CKSEL	CLKDIV	IRC24MCR	XOSCCR	IRC32KCR			

8.3 STC8F2K64S4 系列

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H	P7							RSTCFG
F0H	B	PWMCFG						
E8H	P6						IP3H	AUXINTIF
E0H	ACC	P7M1	P7M0	DPS	DPL1	DPH1	CMPCCR1	CMPCCR2
D8H								IP3
D0H	PSW	T4T3M	T4H	T4L	T3H	T3L	T2H	T2L
C8H	P5	P5M1	P5M0	P6M1	P6M0	SPSTAT	SPCTL	SPDAT
C0H	P4	WDT_CONTR	IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP CONTR
B8H	IP	SADEN	P_SW2	VOCTRL				
B0H	P3	P3M1	P3M0	P4M1	P4M0	IP2	IP2H	IPH
A8H	IE	SADDR	WKTCL	WKTCH	S3CON	S3BUF	TA	IE2
A0H	P2	BUS_SPEED	P_SW1	Reserved				Reserved
98H	SCON	SBUF	S2CON	S2BUF	Reserved		LIRTRIM	IRTRIM
90H	P1	P1M1	P1M0	P0M1	P0M0	P2M1	P2M0	
88H	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	INTCLKO
80H	P0	SP	DPL	DPH	S4CON	S4BUF		PCON

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE18H	P0NCS	P1NCS	P2NCS	P3NCS	P4NCS	P5NCS	P6NCS	P7NCS
FE10H	P0PU	P1PU	P2PU	P3PU	P4PU	P5PU	P6PU	P7PU
FE00H	CKSEL	CLKDIV	IRC24MCR	XOSCCR	IRC32KCR			

8.4 STC8F2K64S2 系列

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H	P7							RSTCFG
F0H	B	PWMCFG						
E8H	P6						IP3H	AUXINTIF
E0H	ACC	P7M1	P7M0	DPS	DPL1	DPH1	CMPCCR1	CMPCCR2
D8H								IP3
D0H	PSW	T4T3M	T4H	T4L	T3H	T3L	T2H	T2L
C8H	P5	P5M1	P5M0	P6M1	P6M0	SPSTAT	SPCTL	SPDAT
C0H	P4	WDT_CONTR	IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP CONTR
B8H	IP	SADEN	P_SW2	VOCTRL				
B0H	P3	P3M1	P3M0	P4M1	P4M0	IP2	IP2H	IPH
A8H	IE	SADDR	WKTCL	WKTCH			TA	IE2
A0H	P2	BUS_SPEED	P_SW1	Reserved				Reserved
98H	SCON	SBUF	S2CON	S2BUF	Reserved		LIRTRIM	IRTRIM
90H	P1	P1M1	P1M0	P0M1	P0M0	P2M1	P2M0	
88H	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	INTCLKO
80H	P0	SP	DPL	DPH				PCON

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE18H	P0NCS	P1NCS	P2NCS	P3NCS	P4NCS	P5NCS	P6NCS	P7NCS
FE10H	P0PU	P1PU	P2PU	P3PU	P4PU	P5PU	P6PU	P7PU
FE00H	CKSEL	CLKDIV	IRC24MCR	XOSCCR	IRC32KCR			

8.5 特殊功能寄存器列表

符号	描述	地址	位地址与符号									复位值
			B7	B6	B5	B4	B3	B2	B1	B0		
P0	P0 端口	80H										1111,1111
SP	堆栈指针	81H										0000,0111
DPL	数据指针 (低字节)	82H										0000,0000
DPH	数据指针 (高字节)	83H										0000,0000
S4CON	串口 4 控制寄存器	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI		0000,0000
S4BUF	串口 4 数据寄存器	85H										0000,0000
PCON	电源控制寄存器	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL		0011,0000
TCON	定时器控制寄存器	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0		0000,0000
TMOD	定时器模式寄存器	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0		0000,0000
TL0	定时器 0 低 8 为寄存器	8AH										0000,0000
TL1	定时器 1 低 8 为寄存器	8BH										0000,0000
TH0	定时器 0 高 8 为寄存器	8CH										0000,0000
TH1	定时器 1 高 8 为寄存器	8DH										0000,0000
AUXR	辅助寄存器 1	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXRAM	S1ST2		0000,0001
INTCLKO	中断与时钟输出控制寄存器	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO		x000,x000
P1	P1 端口	90H										1111,1111
P1M1	P1 口配置寄存器 1	91H										0000,0000
P1M0	P1 口配置寄存器 0	92H										0000,0000
P0M1	P0 口配置寄存器 1	93H										0000,0000
P0M0	P0 口配置寄存器 0	94H										0000,0000
P2M1	P2 口配置寄存器 1	95H										0000,0000
P2M0	P2 口配置寄存器 0	96H										0000,0000
SCON	串口 1 控制寄存器	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI		0000,0000
SBUF	串口 1 数据寄存器	99H										0000,0000
S2CON	串口 2 控制寄存器	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI		0100,0000
S2BUF	串口 2 数据寄存器	9BH										0000,0000
LIRTRIM	IRC 频率微调寄存器	9EH	-	-	-	-	-	-	-	LIRTRIM[1:0]		0000,00nn
IRTRIM	IRC 频率调整寄存器	9FH	IRTRIM[7:0]									nnnn,nnnn
P2	P2 端口	A0H										1111,1111
BUS_SPEED	总线速度控制寄存器	A1H	RW_S[1:0]							SPEED[1:0]		00xx,xx00
P_SW1	外设端口切换寄存器 1	A2H	S1_S[1:0]		CCP_S[1:0]		SPI_S[1:0]		0	-		nn00,000x
IE	中断允许寄存器	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0		0000,0000
SADDR	串口 1 从机地址寄存器	A9H										0000,0000
WKTCL	掉电唤醒定时器低字节	AAH										1111,1111
WKTCH	掉电唤醒定时器高字节	ABH	WKTE									0111,1111
S3CON	串口 3 控制寄存器	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI		0000,0000
S3BUF	串口 3 数据寄存器	ADH										0000,0000
TA	DPTR 时序控制寄存器	AEH										0000,0000
IE2	中断允许寄存器 2	AFH	ECA	ET4	ET3	ES4	ES3	ET2	ESPI	ES2		x000,0000

P3	P3 端口	B0H									1111,1111				
P3M1	P3 口配置寄存器 1	B1H									n000,0000				
P3M0	P3 口配置寄存器 0	B2H									n000,0000				
P4M1	P4 口配置寄存器 1	B3H									0000,0000				
P4M0	P4 口配置寄存器 0	B4H									0000,0000				
IP2	中断优先级控制寄存器 2	B5H	PCAN	PI2C	PCMP	PX4	PPWMFD	PPWM	PSPI	PS2	x000,0000				
IP2H	高中断优先级控制寄存器 2	B6H	PCANH	PI2CH	PCMPPH	PX4H	PPWMFDH	PPWMH	PSPIH	PS2H	x000,0000				
IPH	高中断优先级控制寄存器	B7H	PPCAH	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H	0000,0000				
IP	中断优先级控制寄存器	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	0000,0000				
SADEN	串口 1 从机地址屏蔽寄存器	B9H									0000,0000				
P_SW2	外设端口切换寄存器 2	BAH	EAXFR	CAN_S	I2C_S[1:0]		CMPO_S	S4_S	S3_S	S2_S	0x00,0000				
VOCTRL	电压控制寄存器	BBH	SCC	-	-	-	-	-	0	0	0xxx,xx00				
ADC_CONTR	ADC 控制寄存器	BCH	ADC_POWER	ADC_START	ADC_FLAG	-	ADC_CHS[3:0]				000x,0000				
ADC_RES	ADC 转换结果高位寄存器	BDH									0000,0000				
ADC_RESL	ADC 转换结果低位寄存器	BEH									0000,0000				
P4	P4 端口	C0H	P4[7:0]								1111,1111				
P4	P4 端口 注: STC8A 系列没有 P45~P47	C0H	-	-	-	P4[4:0]				1111,1111					
WDT CONTR	看门狗控制寄存器	C1H	WDT_FLAG	-	EN_WDT	CLR_WDT	IDL_WDT	WDT_PS[2:0]			0x00,0000				
IAP DATA	IAP 数据寄存器	C2H									1111,1111				
IAP_ADDRH	IAP 高地址寄存器	C3H									0000,0000				
IAP_ADDRL	IAP 低地址寄存器	C4H									0000,0000				
IAP_CMD	IAP 命令寄存器	C5H	-	-	-	-	-	-	CMD[1:0]	xxxx,xx00					
IAP_TRIG	IAP 触发寄存器	C6H									0000,0000				
IAP CONTR	IAP 控制寄存器	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	IAP_WT[2:0]			0000,x000				
P5	P5 端口	C8H	-	-											
P5M1	P5 口配置寄存器 1	C9H	-	-											
P5M0	P5 口配置寄存器 0	CAH	-	-											
P6M1	P6 口配置寄存器 1	CBH									0000,0000				
P6M0	P6 口配置寄存器 0	CCH									0000,0000				
SPSTAT	SPI 状态寄存器	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx,xxxx				
SPCTL	SPI 控制寄存器	CEH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR[1:0]		0000,0100				
SPDAT	SPI 数据寄存器	CFH									0000,0000				
PSW	程序状态字寄存器	D0H	CY	AC	F0	RS1	RS0	OV	-	P	0000,00x0				
T4T3M	定时器 4/3 控制寄存器	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO	0000,0000				
T4H	定时器 4 高字节	D2H									0000,0000				
T4L	定时器 4 低字节	D3H									0000,0000				
T3H	定时器 3 高字节	D4H									0000,0000				
T3L	定时器 3 低字节	D5H									0000,0000				
T2H	定时器 2 高字节	D6H									0000,0000				
T2L	定时器 2 低字节	D7H									0000,0000				
CCON	PCA 控制寄存器	D8H	CF	CR	-	-	CCF3	CCF2	CCF1	CCF0	00xx,0000				
CMOD	PCA 模式寄存器	D9H	CIDL	-	-	-	CPS[2:0]			ECF	0xxx,0000				
CCAPM0	PCA 模块 0 模式控制寄存器	DAH	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0	x000,0000				

CCAPM1	PCA 模块 1 模式控制寄存器	DBH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1	x000,0000
CCAPM2	PCA 模块 2 模式控制寄存器	DCH	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2	x000,0000
CCAPM3	PCA 模块 3 模式控制寄存器	DDH	-	ECOM3	CCAPP3	CCAPN3	MAT3	TOG3	PWM3	ECCF3	x000,0000
IP3		DFH	-	-	-	-	-	-	PS4	PS3	xxxx,xx00
ADCCFG	ADC 配置寄存器	DEH	-	-	RESFMT	-	SPEED[3:0]				xx0x,0000
ACC	累加器	E0H									0000,0000
P7M1	P7 口配置寄存器 1	E1H									0000,0000
P7M0	P7 口配置寄存器 0	E2H									0000,0000
DPS	DPTR 指针选择器	E3H	ID1	ID0	TSL	AU1	AU0	-	-	SEL	0000,0xx0
DPL1	第二组数据指针 (低字节)	E4H									0000,0000
DPH1	第二组数据指针 (高字节)	E5H									0000,0000
CMPCR1	比较器控制寄存器 1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES	0000,0000
CMPCR2	比较器控制寄存器 2	E7H	INVCMPO	DISFLT							0000,0000
P6	P6 端口	E8H									1111,1111
CL	PCA 计数器低字节	E9H									0000,0000
CCAPOL	PCA 模块 0 低字节	EAH									0000,0000
CCAPIL	PCA 模块 1 低字节	EBH									0000,0000
CCAP2L	PCA 模块 2 低字节	ECH									0000,0000
CCAP3L	PCA 模块 3 低字节	EDH									0000,0000
IP3H	高中断优先级控制寄存器 3	EEH	-	-	-	-	-	-	PS4H	PS3H	xxxx,xx00
AUXINTIF	扩展外部中断标志寄存器	EFH	-	INT4IF	INT3IF	INT2IF	-	T4IF	T3IF	T2IF	x000,x000
B	B 寄存器	F0H									0000,0000
PWMCFG	增强型 PWM 配置寄存器	F1H	CBIF	ETADC	-	-	-	-	-	-	00xx,xxxx
PCA_PWM0	PCA0 的 PWM 模式寄存器	F2H		EBS0[1:0]		XCCAP0H[1:0]		XCCAP0L[1:0]	EPC0H	EPC0L	0000,0000
PCA_PWM1	PCA1 的 PWM 模式寄存器	F3H		EBS1[1:0]		XCCAP1H[1:0]		XCCAP1L[1:0]	EPC1H	EPC1L	0000,0000
PCA_PWM2	PCA2 的 PWM 模式寄存器	F4H		EBS2[1:0]		XCCAP2H[1:0]		XCCAP2L[1:0]	EPC2H	EPC2L	0000,0000
PCA_PWM3	PCA3 的 PWM 模式寄存器	F5H		EBS3[1:0]		XCCAP3H[1:0]		XCCAP3L[1:0]	EPC3H	EPC3L	0000,0000
PWMIF	增强型 PWM 中断标志寄存器	F6H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWMFDRCR	PWM 异常检测控制寄存器	F7H	INVCMPI	INVIO	ENFD	FLTFLO	EFDI	FDCMP	FDIO	FDIF	0000,0000
P7	P7 端口	F8H									1111,1111
CH	PCA 计数器高字节	F9H									0000,0000
CCAP0H	PCA 模块 0 高字节	FAH									0000,0000
CCAP1H	PCA 模块 1 高字节	FBH									0000,0000
CCAP2H	PCA 模块 2 高字节	FCH									0000,0000
CCAP3H	PCA 模块 3 高字节	FDH									0000,0000
PWMCR	PWM 控制寄存器	FEH	ENPWM	ECBI	-	-	-	-	-	-	00xx,xxxx
RSTCFG	复位配置寄存器	FFH	-	ENLVR	-	P54RST	-	-		LVDS[1:0]	0000,0000

下列特殊功能寄存器为扩展 SFR，逻辑地址位于 XDATA 区域，访问前需要将 P_SW2 (BAH) 寄存器的最高位 (EAXFR) 置 1，然后使用 MOVXA,@DPTR 和 MOVX @DPTRA 指令进行访问

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
PWMCH	PWM 计数器高字节	FFF0H	-								x000,0000

PWMCL	PWM 计数器低字节	FFF1H									0000,0000					
PWMCKS	PWM 时钟选择	FFF2H	-	-	-	SELT2	PWM_PS[3:0]				xxx0,0000					
TADCPH	触发 ADC 计数值高字节	FFF3H	-									x000,0000				
TADCPL	触发 ADC 计数值低字节	FFF4H									0000,0000					
PWM0T1H	PWM0T1 计数值高字节	FF00H	-									x000,0000				
PWM0T1L	PWM0T1 计数值低节	FF01H									0000,0000					
PWM0T2H	PWM0T2 数值高字节	FF02H	-									x000,0000				
PWM0T2L	PWM0T2 数值低节	FF03H									0000,0000					
PWM0CR	PWM0 控制寄存器	FF04H	ENC0O	C0INI	-	C0_S[1:0]		EC0I	EC0T2SI	EC0T1SI	00x0,0000					
PWM0HLD	PWM0 电平保持控制寄存器	FF05H	-	-	-	-	-	-	HC0H	HC0L	xxxx,xx00					
PWM1T1H	PWM1T1 计数值高字节	FF10H	-									x000,0000				
PWM1T1L	PWM1T1 计数值低节	FF11H									0000,0000					
PWM1T2H	PWM1T2 数值高字节	FF12H	-									x000,0000				
PWM1T2L	PWM1T2 数值低节	FF13H									0000,0000					
PWM1CR	PWM1 控制寄存器	FF14H	ENC1O	C1INI	-	C1_S[1:0]		EC1I	EC1T2SI	EC1T1SI	00x0,0000					
PWM1HLD	PWM1 电平保持控制寄存器	FF15H	-	-	-	-	-	-	HC1H	HC1L	xxxx,xx00					
PWM2T1H	PWM2T1 计数值高字节	FF20H	-									x000,0000				
PWM2T1L	PWM2T1 计数值低节	FF21H									0000,0000					
PWM2T2H	PWM2T2 数值高字节	FF22H	-									x000,0000				
PWM2T2L	PWM2T2 数值低节	FF23H									0000,0000					
PWM2CR	PWM2 控制寄存器	FF24H	ENC2O	C2INI	-	C2_S[1:0]		EC2I	EC2T2SI	EC2T1SI	00x0,0000					
PWM2HLD	PWM2 电平保持控制寄存器	FF25H	-	-	-	-	-	-	HC2H	HC2L	xxxx,xx00					
PWM3T1H	PWM3T1 计数值高字节	FF30H	-									x000,0000				
PWM3T1L	PWM3T1 计数值低节	FF31H									0000,0000					
PWM3T2H	PWM3T2 数值高字节	FF32H	-									x000,0000				
PWM3T2L	PWM3T2 数值低节	FF33H									0000,0000					
PWM3CR	PWM3 控制寄存器	FF34H	ENC3O	C3INI	-	C3_S[1:0]		EC3I	EC3T2SI	EC3T1SI	00x0,0000					
PWM3HLD	PWM3 电平保持控制寄存器	FF35H	-	-	-	-	-	-	HC3H	HC3L	xxxx,xx00					
PWM4T1H	PWM4T1 计数值高字节	FF40H	-									x000,0000				
PWM4T1L	PWM4T1 计数值低节	FF41H									0000,0000					
PWM4T2H	PWM4T2 数值高字节	FF42H	-									x000,0000				
PWM4T2L	PWM4T2 数值低节	FF43H									0000,0000					
PWM4CR	PWM4 控制寄存器	FF44H	ENC4O	C4INI	-	C4_S[1:0]		EC4I	EC4T2SI	EC4T1SI	00x0,0000					
PWM4HLD	PWM4 电平保持控制寄存器	FF45H	-	-	-	-	-	-	HC4H	HC4L	xxxx,xx00					
PWM5T1H	PWM5T1 计数值高字节	FF50H	-									x000,0000				
PWM5T1L	PWM5T1 计数值低节	FF51H									0000,0000					
PWM5T2H	PWM5T2 数值高字节	FF52H	-									x000,0000				
PWM5T2L	PWM5T2 数值低节	FF53H									0000,0000					
PWM5CR	PWM5 控制寄存器	FF54H	ENC5O	C5INI	-	C5_S[1:0]		EC5I	EC5T2SI	EC5T1SI	00x0,0000					
PWM5HLD	PWM5 电平保持控制寄存器	FF55H	-	-	-	-	-	-	HC5H	HC5L	xxxx,xx00					
PWM6T1H	PWM6T1 计数值高字节	FF60H	-									x000,0000				
PWM6T1L	PWM6T1 计数值低节	FF61H									0000,0000					
PWM6T2H	PWM6T2 数值高字节	FF62H	-									x000,0000				
PWM6T2L	PWM6T2 数值低节	FF63H									0000,0000					

PWM6CR	PWM6 控制寄存器	FF64H	ENC6O	C6INI	-	C6_S[1:0]		EC6I	EC6T2SI	EC6T1SI	00x0,0000
PWM6HLD	PWM6 电平保持控制寄存器	FF65H	-	-	-	-	-	-	HC6H	HC6L	xxxx,xx00
PWM7T1H	PWM7T1 计数值高字节	FF70H	-								x000,0000
PWM7T1L	PWM7T1 计数值低节	FF71H									0000,0000
PWM7T2H	PWM7T2 数值高字节	FF72H	-								x000,0000
PWM7T2L	PWM7T2 数值低节	FF73H									0000,0000
PWM7CR	PWM7 控制寄存器	FF74H	ENC7O	C7INI	-	C7_S[1:0]		EC7I	EC7T2SI	EC7T1SI	00x0,0000
PWM7HLD	PWM7 电平保持控制寄存器	FF75H	-	-	-	-	-	-	HC7H	HC7L	xxxx,xx00
I2CCFG	I ² C 配置寄存器	FE80H	ENI2C	MSSL		MSSPEED[6:1]					0000,0000
I2CMSCR	I ² C 主机控制寄存器	FE81H	EMSI	-	-	-		MSCMD[3:0]			0xxx,0000
I2CMSST	I ² C 主机状态寄存器	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO	00xx,xx00
I2CSLCR	I ² C 从机控制寄存器	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0
I2CSLST	I ² C 从机状态寄存器	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000
I2CSLADR	I ² C 从机地址寄存器	FE85H		SLADR[6:0]						MA	0000,0000
I2CTXD	I ² C 数据发送寄存器	FE86H									0000,0000
I2CRXD	I ² C 数据接收寄存器	FE87H									0000,0000
I2CMSAUX	I ² C 主机辅助控制寄存器	FE88H	-	-	-	-	-	-	-	WDTA	xxxx,xxx0
POPU	P0 口上拉电阻控制寄存器	FE10H									0000,0000
P1PU	P1 口上拉电阻控制寄存器	FE11H									0000,0000
P2PU	P2 口上拉电阻控制寄存器	FE12H									0000,0000
P3PU	P3 口上拉电阻控制寄存器	FE13H									0000,0000
P4PU	P4 口上拉电阻控制寄存器	FE14H									0000,0000
P5PU	P5 口上拉电阻控制寄存器	FE15H									0000,0000
P6PU	P6 口上拉电阻控制寄存器	FE16H									0000,0000
P7PU	P7 口上拉电阻控制寄存器	FE17H									0000,0000
P0NCS	P0 口施密特触发控制寄存器	FE18H									0000,0000
P1NCS	P1 口施密特触发控制寄存器	FE19H									0000,0000
P2NCS	P2 口施密特触发控制寄存器	FE1AH									0000,0000
P3NCS	P3 口施密特触发控制寄存器	FE1BH									0000,0000
P4NCS	P4 口施密特触发控制寄存器	FE1CH									0000,0000
P5NCS	P5 口施密特触发控制寄存器	FE1DH									0000,0000
P6NCS	P6 口施密特触发控制寄存器	FE1EH									0000,0000
P7NCS	P7 口施密特触发控制寄存器	FE1FH									0000,0000
CKSEL	时钟选择寄存器	FE00H		MCLKODIV[3:0]			MCLKO_S	-	MCKSEL[1:0]		0000,0000
CLKDIV	时钟分频寄存器	FE01H									0000,0100
IRC24MCR	内部 24M 振荡器控制寄存器	FE02H	ENIRC24M	-	-	-	-	-	-	IRC24MST	1xxx,xxx0
XOSCCR	外部晶振控制寄存器	FE03H	ENXOSC	XITYPE	-	-	-	-	-	XOSCST	00xx,xxx0
IRC32KCR	内部 32K 振荡器控制寄存器	FE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST	0xxx,xxx0

9 I/O 口

STC8 系列单片机最多有 59 个 I/O 口。所有的 I/O 口均有 4 种工作模式：准双向口/弱上拉（标准 8051 输出口模式）、推挽输出/强上拉、高阻输入（电流既不能流入也不能流出）、开漏输出。可使用软件对 I/O 口的工作模式进行容易配置。

9.1 I/O 口相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
P0	P0 端口	80H									1111,1111
P1	P1 端口	90H									1111,1111
P2	P2 端口	A0H									1111,1111
P3	P3 端口	B0H									1111,1111
P4	P4 端口 注: STC8A 系列没有 P45~P47	C0H									1111,1111
P5	P5 端口	C8H	-	-							xx11,1111
P6	P6 端口	E8H									1111,1111
P7	P7 端口	F8H									1111,1111
P0M1	P0 口配置寄存器 1	93H									0000,0000
P0M0	P0 口配置寄存器 0	94H									0000,0000
P1M1	P1 口配置寄存器 1	91H									0000,0000
P1M0	P1 口配置寄存器 0	92H									0000,0000
P2M1	P2 口配置寄存器 1	95H									0000,0000
P2M0	P2 口配置寄存器 0	96H									0000,0000
P3M1	P3 口配置寄存器 1	B1H									n000,0000
P3M0	P3 口配置寄存器 0	B2H									n000,0000
P4M1	P4 口配置寄存器 1	B3H									0000,0000
P4M0	P4 口配置寄存器 0	B4H									0000,0000
P5M1	P5 口配置寄存器 1	C9H	-	-							xx11,1111
P5M0	P5 口配置寄存器 0	CAH	-	-							xx11,1111
P6M1	P6 口配置寄存器 1	CBH									0000,0000
P6M0	P6 口配置寄存器 0	CCH									0000,0000
P7M1	P7 口配置寄存器 1	E1H									0000,0000
P7M0	P7 口配置寄存器 0	E2H									0000,0000

符号	描述	地址	位地址与符号								复位值	
			B7	B6	B5	B4	B3	B2	B1	B0		
P0PU	P0 口上拉电阻控制寄存器	FE10H										0000,0000
P1PU	P1 口上拉电阻控制寄存器	FE11H										0000,0000
P2PU	P2 口上拉电阻控制寄存器	FE12H										0000,0000
P3PU	P3 口上拉电阻控制寄存器	FE13H										0000,0000
P4PU	P4 口上拉电阻控制寄存器	FE14H										0000,0000
P5PU	P5 口上拉电阻控制寄存器	FE15H										0000,0000

P6PU	P6 口上拉电阻控制寄存器	FE16H	0000,0000						
P7PU	P7 口上拉电阻控制寄存器	FE17H	0000,0000						
P0NCS	P0 口施密特触发控制寄存器	FE18H	0000,0000						
P1NCS	P1 口施密特触发控制寄存器	FE19H	0000,0000						
P2NCS	P2 口施密特触发控制寄存器	FE1AH	0000,0000						
P3NCS	P3 口施密特触发控制寄存器	FE1BH	0000,0000						
P4NCS	P4 口施密特触发控制寄存器	FE1CH	0000,0000						
P5NCS	P5 口施密特触发控制寄存器	FE1DH	0000,0000						
P6NCS	P6 口施密特触发控制寄存器	FE1EH	0000,0000						
P7NCS	P7 口施密特触发控制寄存器	FE1FH	0000,0000						

端口数据寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0	80H	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0
P1	90H	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
P2	A0H	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0
P3	B0H	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0
P4	C0H	P4.7	P4.6	P4.5	P4.4	P4.3	P4.2	P4.1	P4.0
P5	C8H	-	-	P5.5	P5.4	P5.3	P5.2	P5.1	P5.0
P6	E8H	P6.7	P6.6	P6.5	P6.4	P6.3	P6.2	P6.1	P6.0
P7	F8H	P7.7	P7.6	P7.5	P7.4	P7.3	P7.2	P7.1	P7.0

读写端口状态

写 0: 输出低电平到端口缓冲区

写 1: 输出高电平到端口缓冲区

读: 直接读端口管脚上的电平

注: STC8A系列没有P45~P47

端口模式配置寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0M0	94H								
P0M1	93H								
P1M0	92H								
P1M1	91H								
P2M0	96H								
P2M1	95H								
P3M0	B2H								
P3M1	B1H								
P4M0	B4H								
P4M1	B3H								
P5M0	CAH	-	-						
P5M1	C9H	-	-						
P6M0	CCH								
P6M1	CBH								

P7M0	E2H	
P7M1	E1H	

配置端口的模式

PnM1.x	PnM0.x	Pn.x 口工作模式
0	0	准双向口
0	1	推挽输出
1	0	高阻输入
1	1	开漏输出

端口上拉电阻控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0PU	FE10H								
P1PU	FE11H								
P2PU	FE12H								
P3PU	FE13H								
P4PU	FE14H								
P5PU	FE15H	-	-						
P6PU	FE16H								
P7PU	FE17H								

端口内部3.7K上拉电阻控制位（注：P3.0和P3.1口上的上拉电阻可能会略小一些）

- 0: 禁止端口内部的3.7K上拉电阻（实测为4.2K左右）
- 1: 使能端口内部的3.7K上拉电阻（实测为4.2K左右）

端口施密特触发控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0NCS	FE18H								
P1NCS	FE19H								
P2NCS	FE1AH								
P3NCS	FE1BH								
P4NCS	FE1CH								
P5NCS	FE1DH	-	-						
P6NCS	FE1EH								
P7NCS	FE1FH								

端口施密特触发控制位

- 0: **使能**端口的施密特触发功能。（上电复位后默认使能施密特触发）
- 1: **禁止**端口的施密特触发功能。

VCC=5.0V	最小值	最大值	
普通 IO 输入高电平	2.2V	-	打开施密特触发
普通 IO 输入低电平	-	1.4V	
普通 IO 输入高电平	1.6V	-	关闭施密特触发

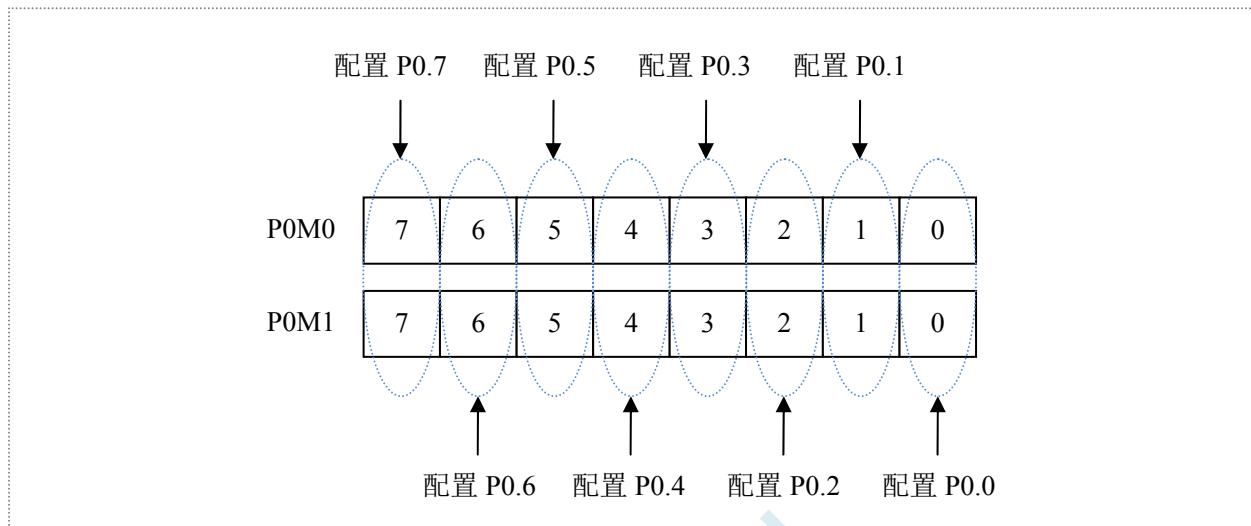
普通 IO 输入低电平	-	1.5V	
复位脚输入高电平	2.2V	-	
复位脚输入低电平	-	1.8V	

VCC=3.3V	最小值	最大值	
普通 IO 输入高电平	1.6V	-	打开施密特触发
普通 IO 输入低电平	-	1.0V	
普通 IO 输入高电平	1.2V		关闭施密特触发
普通 IO 输入低电平		1.1V	
复位脚输入高电平	1.7V	-	
复位脚输入低电平	-	1.3V	

9.2 配置I/O口

每个 I/O 的配置都需要使用两个寄存器进行设置。

以 P0 口为例, 配置 P0 口需要使用 P0M0 和 P0M1 两个寄存器进行配置, 如下图所示:



即 P0M0 的第 0 位和 P0M1 的第 0 位组合起来配置 P0.0 口的模式

即 P0M0 的第 1 位和 P0M1 的第 1 位组合起来配置 P0.1 口的模式

其他所有 I/O 的配置都与此类似。

PnM0 与 PnM1 的组合方式如下表所示

PnM1	PnM0	I/O 口工作模式
0	0	准双向口 (传统8051端口模式, 弱上拉) 灌电流可达20mA, 拉电流为270~150μA (存在制造误差)
0	1	推挽输出 (强上拉输出, 可达20mA, 要加限流电阻)
1	0	高阻输入 (电流既不能流入也不能流出)
1	1	开漏输出 (Open-Drain), 内部上拉电阻断开 开漏模式既可读外部状态也可对外输出 (高电平或低电平)。如要正确读外部状态或需要对外输出高电平, 需外加上拉电阻, 否则读不到外部状态, 也对外输出不出高电平。

注: n = 0, 1, 2, 3, 4, 5, 6, 7

注意:

虽然每个 I/O 口在弱上拉 (准双向口) /强推挽输出/开漏模式时都能承受 20mA 的灌电流 (还是要加限流电阻, 如 1K、560Ω、472Ω 等), 在强推挽输出时能输出 20mA 的拉电流 (也要加限流电阻), 但整个芯片的工作电流推荐不要超过 90mA, 即从 VCC 流入的电流建议不要超过 90mA, 从 GND 流出电流建议不要超过 90mA, 整体流入/流出电流建议都不要超过 90mA。

9.3 I/O的结构图

9.3.1 准双向口（弱上拉）

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

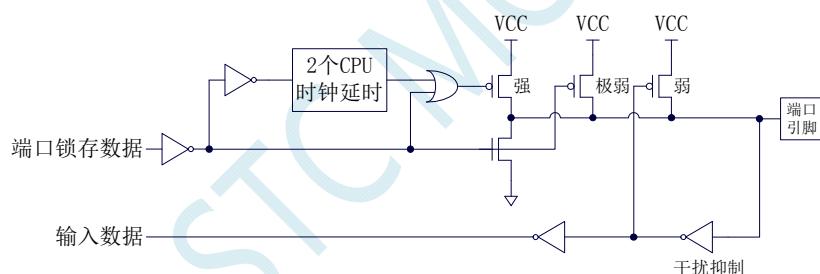
在 3 个上拉晶体管中，有 1 个上拉晶体管称为“弱上拉”，当端口寄存器为 1 且引脚本身也为 1 时打开。此上拉提供基本驱动电流使准双向口输出为 1。如果一个引脚输出为 1 而由外部装置下拉到低时，弱上拉关闭而“极弱上拉”维持开状态，为了把这个引脚强拉为低，外部装置必须有足够的灌电流能力使引脚上的电压降到门槛电压以下。对于 5V 单片机，“弱上拉”晶体管的电流约 250uA；对于 3.3V 单片机，“弱上拉”晶体管的电流约 150uA。

第 2 个上拉晶体管，称为“极弱上拉”，当端口锁存为 1 时打开。当引脚悬空时，这个极弱的上拉源产生很弱的上拉电流将引脚上拉为高电平。对于 5V 单片机，“极弱上拉”晶体管的电流约 18uA；对于 3.3V 单片机，“极弱上拉”晶体管的电流约 5uA。

第 3 个上拉晶体管称为“强上拉”。当端口锁存器由 0 到 1 跳变时，这个上拉用来加快准双向口由逻辑 0 到逻辑 1 转换。当发生这种情况时，强上拉打开约 2 个时钟以使引脚能够迅速地上拉到高电平。

准双向口（弱上拉）带有一个施密特触发输入以及一个干扰抑制电路。准双向口（弱上拉）读外部状态前，要先锁存为‘1’，才可读到外部正确的状态。

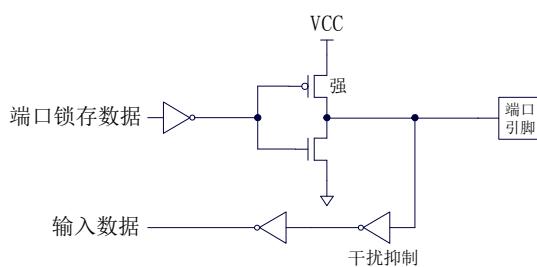
准双向口（弱上拉）输出如下图所示：



9.3.2 推挽输出

强推挽输出配置的下拉结构与开漏输出以及准双向口的下拉结构相同，但当锁存器为 1 时提供持续的强上拉。推挽模式一般用于需要更大驱动电流的情况。

强推挽引脚配置如下图所示：

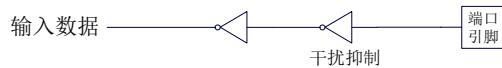


9.3.3 高阻输入

电流既不能流入也不能流出

输入口带有一个施密特触发输入以及一个干扰抑制电路

高阻输入引脚配置如下图所示:



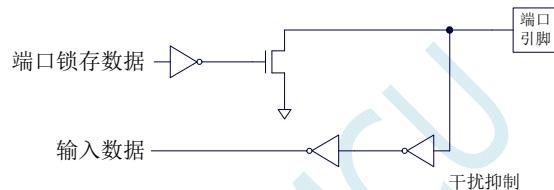
9.3.4 开漏输出

开漏模式既可读外部状态也可对外输出（高电平或低电平）。如要正确读外部状态或需要对外输出高电平，需外加上拉电阻。

当端口锁存器为 0 时，开漏输出关闭所有上拉晶体管。当作为一个逻辑输出高电平时，这种配置方式必须有外部上拉，一般通过电阻外接到 VCC。如果外部有上拉电阻，开漏的 I/O 口还可读外部状态，即此时被配置为开漏模式的 I/O 口还可作为输入 I/O 口。这种方式的下拉与准双向口相同。

开漏端口带有一个施密特触发输入以及一个干扰抑制电路。

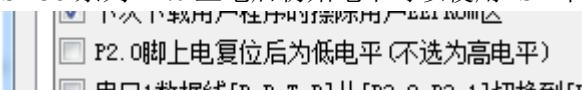
输出端口配置如下图所示:



9.4 特殊I/O口说明

9.4.1 P2.0/RSTCV

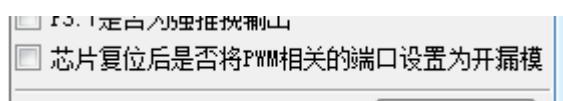
STC8 系列 P2.0 上电后初始电平可以使用 ISP 下载软件中的如下硬件选项进行设置



注意: 当 MCU 的工作电压低于 1.6V 时, P2.0 输出电平高电平, 只有当 MCU 的工作电压上升到 1.6V 以上时, P2.0 才会输出用户硬件选项设置的电平。

9.4.2 PWM相关I/O口

STC8 系列的所有 IO 口上电复位后默认模式均是弱上拉准双向口模式, 用户可以通过 ISP 下载软件中的如下硬件选项将 PWM 相关的 IO 口配置为开漏模式



STC8 系列 PWM 相关的 I/O 口为 P1.0~P1.7, P2.0~P2.7, P6.0~P6.7

9.5 范例程序

9.5.1 端口模式设置

汇编代码

<i>P0M0</i>	<i>DATA</i>	<i>094H</i>	
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>	
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>	
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>	
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>	
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>	
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>	
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>	
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>	
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>	
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</i>	
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>	
<i>P6M0</i>	<i>DATA</i>	<i>0CCH</i>	
<i>P6M1</i>	<i>DATA</i>	<i>0CBH</i>	
<i>P7M0</i>	<i>DATA</i>	<i>0E2H</i>	
<i>P7M1</i>	<i>DATA</i>	<i>0E1H</i>	
	<i>ORG</i>	<i>0000H</i>	
	<i>LJMP</i>	<i>MAIN</i>	
	<i>ORG</i>	<i>0100H</i>	
<i>MAIN:</i>	<i>MOV</i>	<i>SP,#3FH</i>	
	<i>MOV</i>	<i>P0M0,#00H</i>	<i>; 设置 P0.0~P0.7 为双向口模式</i>
	<i>MOV</i>	<i>P0M1,#00H</i>	
	<i>MOV</i>	<i>P1M0,#0FFH</i>	<i>; 设置 P1.0~P1.7 为推挽输出模式</i>
	<i>MOV</i>	<i>P1M1,#00H</i>	
	<i>MOV</i>	<i>P2M0,#00H</i>	<i>; 设置 P2.0~P2.7 为高阻输入模式</i>
	<i>MOV</i>	<i>P2M1,#0FFH</i>	
	<i>MOV</i>	<i>P3M0,#0FFH</i>	<i>; 设置 P3.0~P3.7 为开漏模式</i>
	<i>MOV</i>	<i>P3M1,#0FFH</i>	
	<i>JMP</i>	<i>\$</i>	
 <i>END</i>			

C 语言代码

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

<i>sfr</i>	<i>P0M0</i>	=	<i>0x94;</i>
<i>sfr</i>	<i>P0M1</i>	=	<i>0x93;</i>
<i>sfr</i>	<i>P1M0</i>	=	<i>0x92;</i>
<i>sfr</i>	<i>P1M1</i>	=	<i>0x91;</i>
<i>sfr</i>	<i>P2M0</i>	=	<i>0x96;</i>
<i>sfr</i>	<i>P2M1</i>	=	<i>0x95;</i>
<i>sfr</i>	<i>P3M0</i>	=	<i>0xb2;</i>
<i>sfr</i>	<i>P3M1</i>	=	<i>0xb1;</i>
<i>sfr</i>	<i>P4M0</i>	=	<i>0xb4;</i>
<i>sfr</i>	<i>P4M1</i>	=	<i>0xb3;</i>

```

sfr      P5M0      = 0xca;
sfr      P5M1      = 0xc9;
sfr      P6M0      = 0xcc;
sfr      P6M1      = 0xcb;
sfr      P7M0      = 0xe2;
sfr      P7M1      = 0xe1;

void main()
{
    P0M0 = 0x00;          // 设置 P0.0~P0.7 为双向口模式
    P0M1 = 0x00;
    P1M0 = 0xff;          // 设置 P1.0~P1.7 为推挽输出模式
    P1M1 = 0x00;
    P2M0 = 0x00;          // 设置 P2.0~P2.7 为高阻输入模式
    P2M1 = 0xff;
    P3M0 = 0xff;          // 设置 P3.0~P3.7 为开漏模式
    P3M1 = 0xff;

    while (1);
}

```

9.5.2 双向口读写操作

汇编代码

```

P0M0      DATA      094H
P0M1      DATA      093H

        ORG      0000H
        LJMP     MAIN

        ORG      0100H
MAIN:   MOV      SP,#3FH

        MOV      P0M0,#00H      ; 设置 P0.0~P0.7 为双向口模式
        MOV      P0M1,#00H

        SETB     P0.0          ; P0.0 口输出高电平
        CLR      P0.0          ; P0.0 口输出低电平

        SETB     P0.0          ; 读取端口前先使能内部弱上拉电阻
        NOP      NOP            ; 等待两个时钟
        NOP      NOP
        MOV      C,P0.0         ; 读取端口状态

        JMP     $

END

```

C 语言代码

```

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

sfr      P0M0      = 0x94;
sfr      P0M1      = 0x93;
sbit     P00       = P0^0;

```

```

void main()
{
    P0M0 = 0x00; //设置 P0.0~P0.7 为双向口模式
    P0M1 = 0x00;

    P00 = 1; //P0.0 口输出高电平
    P00 = 0; //P0.0 口输出低电平

    P00 = 1; //读取端口前先使能内部弱上拉电阻
    _nop_(); //等待两个时钟
    _nop_();
    //
    CY = P00; //读取端口状态

    while (1);
}

```

9.5.3 用STC系列MCU的IO口直接驱动段码LCD

当产品需要段码 LCD 显示时, 如果使用不带 LCD 驱动器的 MCU, 则需要外接 LCD 驱动 IC, 这会增加成本和 PCB 面积。事实上, 很多小项目, 比如大量的小家电, 需要显示的段码不多, 常见的是 4 个 8 带小数点或时钟的冒号 “：“, 这样如果使用 IO 口直接扫描显示, 则会减小 PCB 面积, 降低成本。但是, 本方案不合适驱动太多的段 (占用 IO 太多), 也不合适非常低功耗的场合。

段码 LCD 驱动简单原理: 如图 1 所示。

LCD 是一种特殊的液态晶体, 在电场的作用下晶体的排列方向会发生扭转, 因而改变其透光性, 从而可以看到显示内容。LCD 有一个扭转阀值, 当 LCD 两端电压高于此阀值时, 显示内容, 低于此阀值时, 不显示。通常 LCD 有 3 个参数: 工作电压、DUTY (对应 COM 数) 和 BIAS (即偏压, 对应阀值), 比如 4.5V、1/4 DUTY、1/3 BIAS, 表示 LCD 显示电压为 4.5V, 4 个 COM, 阀值大约是 1.5V, 当加在某段 LCD 两端电压大于 1.5V 时 (一般加 4.5V) 显示, 而加 1.5V 时不显示。但是 LCD 对于驱动电压的反应不是很明显的, 比如加 2V 时, 可能会微弱显示, 这就是通常说的“鬼影”。所以要保证驱动显示时, 要大于阀值电压比较多, 而不显示时, 要用比阀值小比较多的电压。

注意: LCD 的两端不能加直流电压, 否则时间稍长就会损坏, 所以要保证加在 LCD 两端的驱动电压的平均电压为 0。LCD 使用时分割扫描法, 任何时候一个 COM 扫描有效, 另外的 COM 处于无效状态。

驱动 1/4Duty 1/2BIAS 3V 的方案电路见图 1, LCD 扫描原理见图 3, MCU 为 3V 工作, 用双向口做 COM, PUSH-PULL 或 STANDARD 输出口接 SEG, 并且每个 COM 都接一个 47K 电阻到一个电容, RC 滤波后得到一个中点电压。在轮到某个 COM 扫描时, 设置成 PUSH-PULL 输出, 如果与本 COM 连接的 SEG 不显示, 则 SEG 输出与 COM 同相, 如果显示, 则反相。扫描完后, 这个 COM 的 IO 就设置成高阻, 这样这个 COM 就通过 47K 电阻连接到 1/2VDD 电压, 而 SEG 继续输出方波, 这样加在 LCD 上的电压, 显示时是+VDD, 不显示时是+1/2VDD, 保证了 LCD 两端平均直流电压为 0。

驱动 1/4Duty 1/3BIAS 3V 的方案电路见图 4, LCD 扫描原理见图 5,, MCU 为 5V 工作, SEG 线通过电阻分压输出 1.5V、3.5V, COM 线通过电阻分压输出 0.5V、2.5V (高阻时)、4.5V。在轮到某个 COM 扫描时, 设置成 PUSH-PULL 输出, 如果与本 COM 连接的 SEG 不显示, 则 SEG 输出与 COM 同相, 如果显示, 则反相。扫描完后, 这个 COM 的 IO 就设置成高阻, 这样这个 COM 就通过 47K 电阻连接到 2.5V 电压, 而 SEG 继续输出方波, 这样加在 LCD 上的电压, 显示时是+3.0V, 不显示时是+1.0V, 完全满足 LCD 的扫描要求。

当需要睡眠省电时, 把所有 COM 和 SEG 驱动 IO 全部输出低电平, LCD 驱动部分不会增加额外电流。

图 1: 驱动 1/4Duty 1/2BIAS 3V LCD 的电路

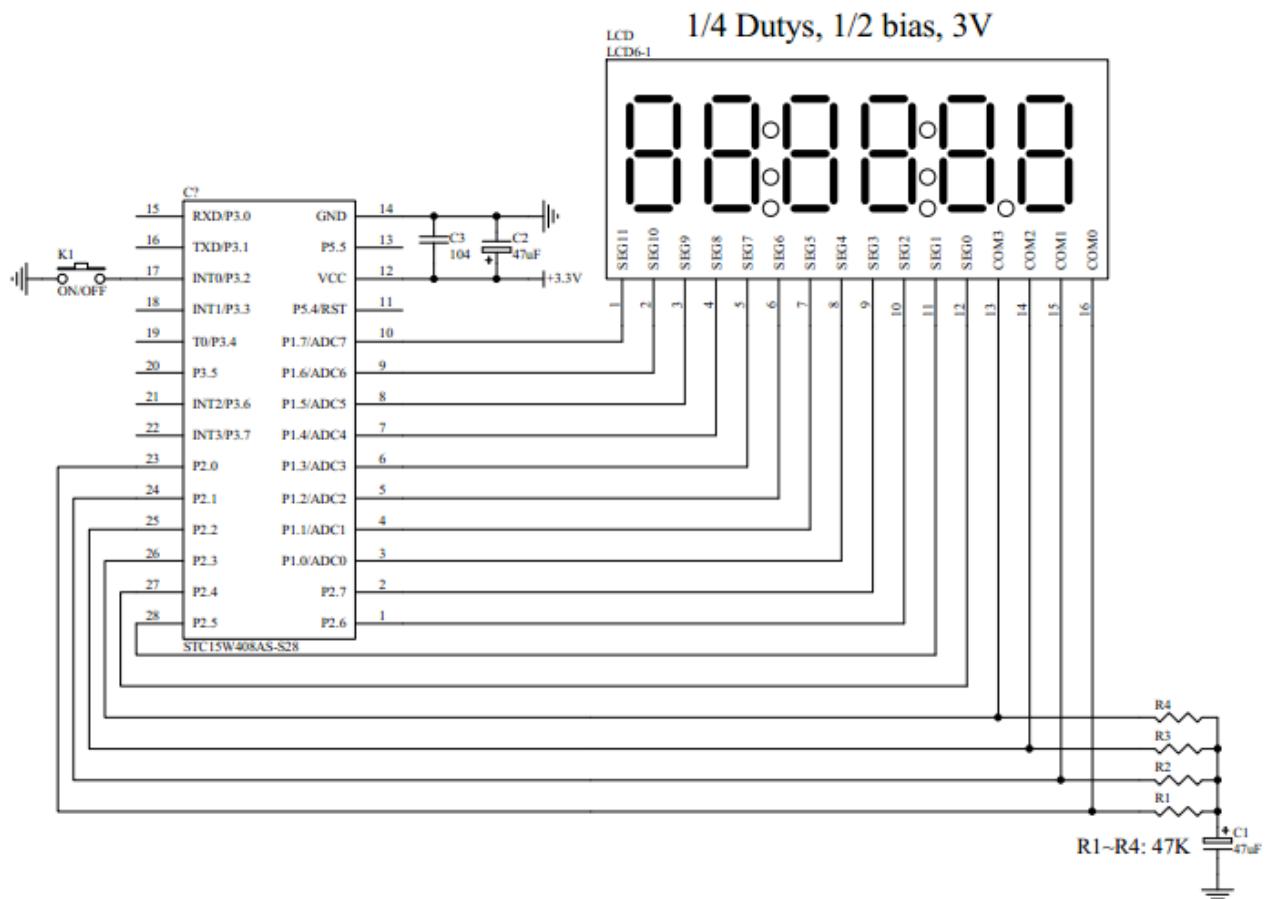


图 2: 段码名称图

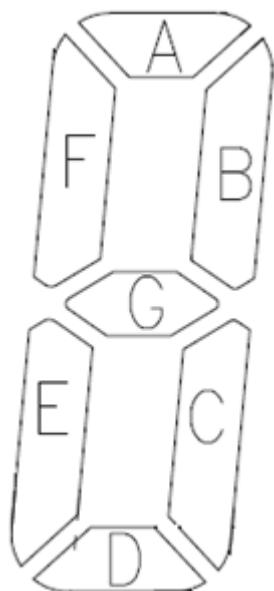


图 3: 1/4Duty 1/2BIAS 扫描原理图

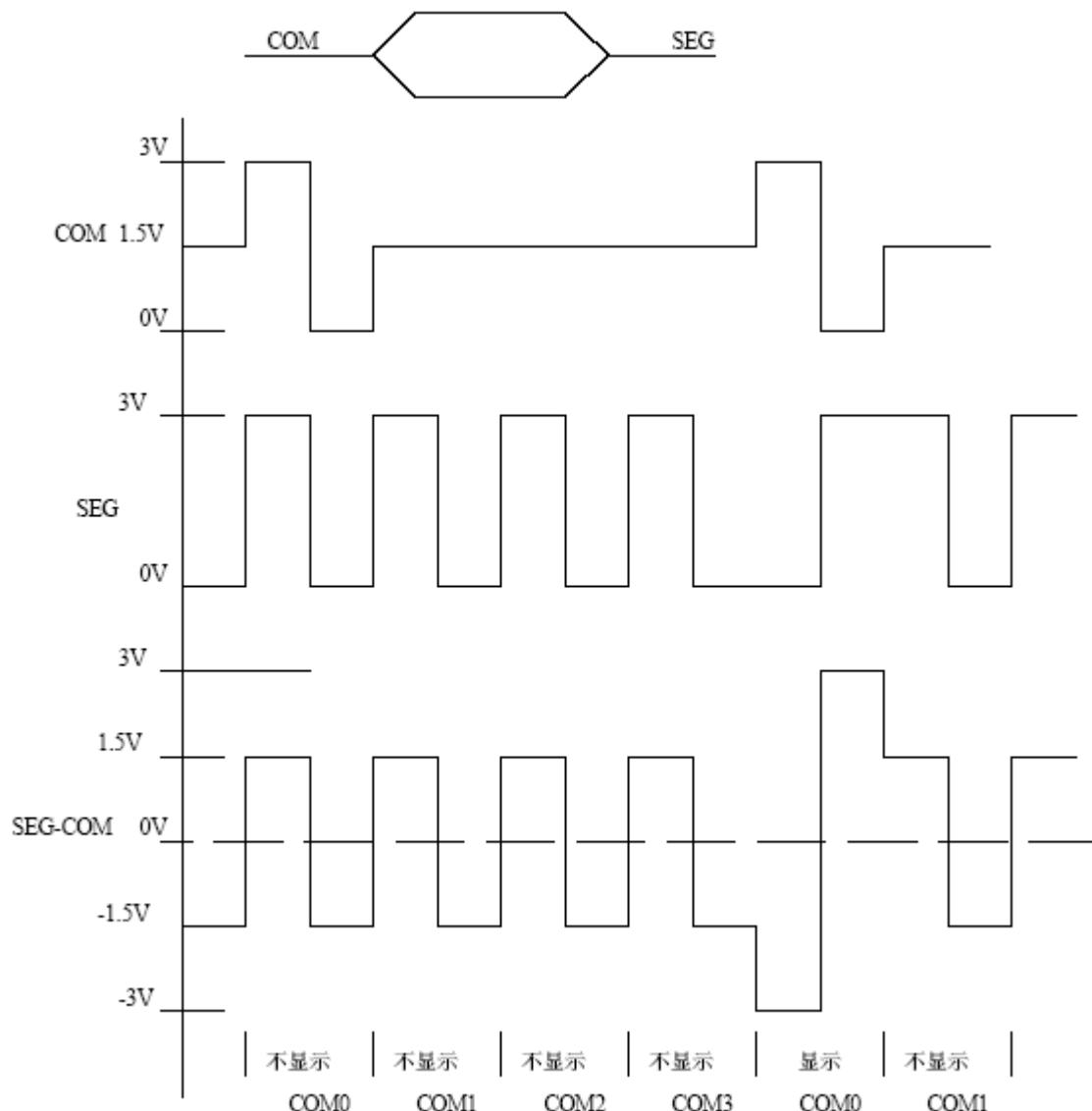


图 4: 驱动 1/4Duty 1/3BIAS 3V LCD 的电路

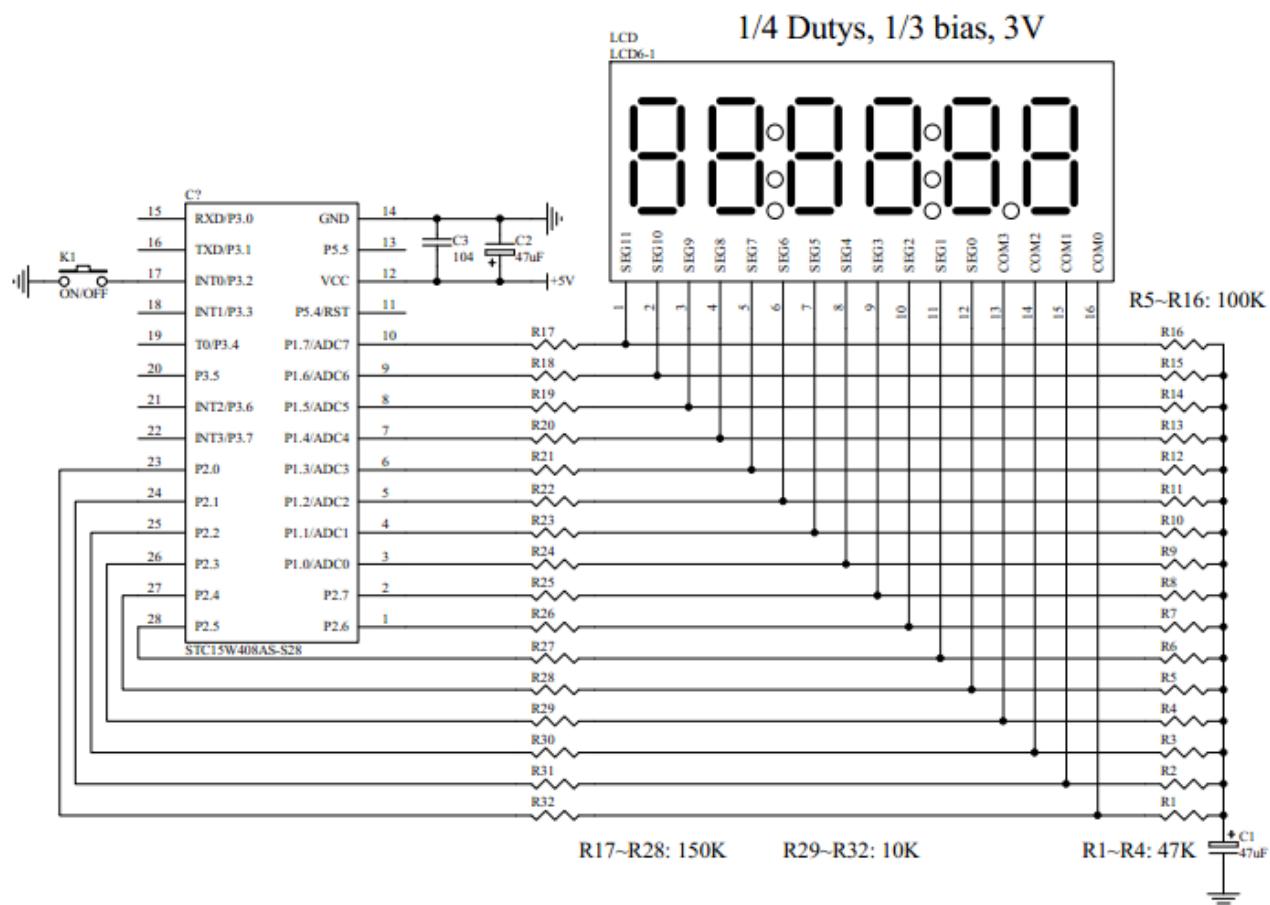
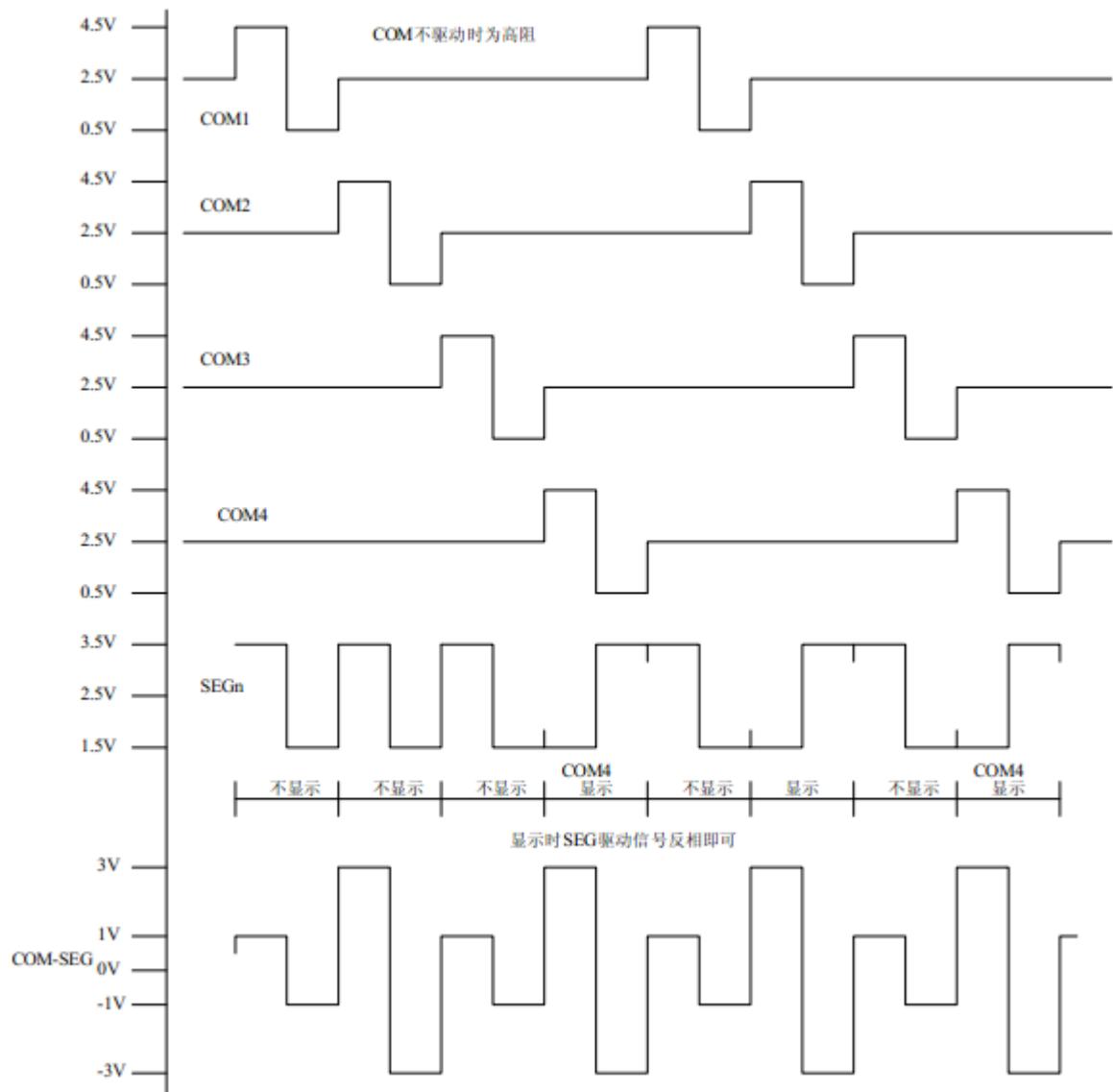


图 5: 1/4Duty 1/3BIAS 扫描原理图



为了使用方便，显示内容放在一个显存中，其中的各个位与 LCD 的段一一对应，见图 6。

图 6: LCD 真值表和显存影射表

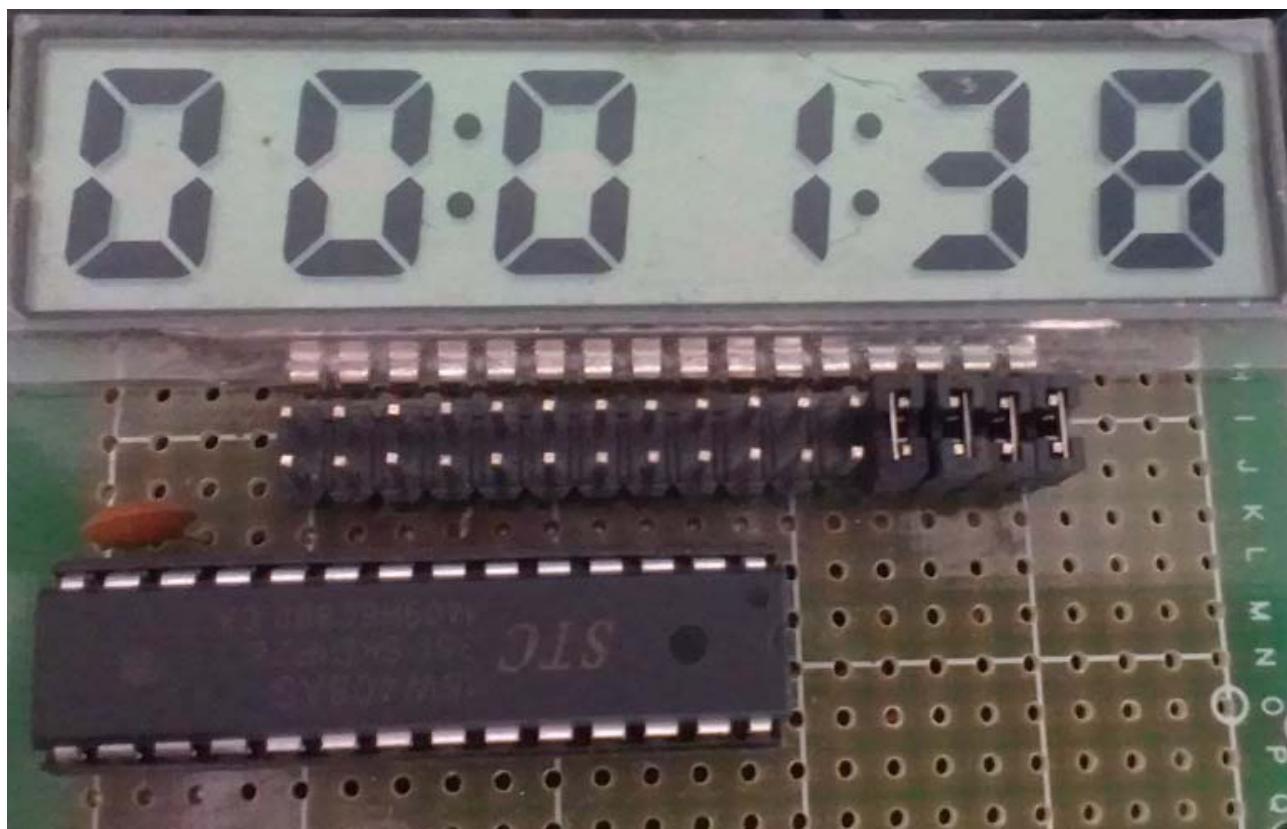
LCD 真值表:

MCU PIN	P17	P16	P15	P14	P13	P12	P11	P10	P27	P26	P25	P24	P23	P22	P21	P20
LCD PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
LCD PIN name	SEG11	SEG10	SEG9	SEG8	SEG7	SEG6	SEG5	SEG4	SEG3	SEG2	SEG1	SEG0	COM3	COM2	COM1	COM0
--	1D	2D	2.	3D	4:	4D	4.	5D	5.	6D	5.	6D	COM3			
1E	1C	2E	2C	3E	3C	4E	4C	5E	5C	6E	6C		COM2			
1G	1B	2G	2B	3G	3B	4G	4B	5G	5B	6G	6B			COM1		
1F	1A	2F	2A	3F	3A	4F	4A	5F	5A	6F	6A				COM0	

显存影射表:

	B7	B6	B5	B4	B3	B2	B1	B0
buff[0]:	--	1D	2:	2D	2.	3D	4:	4D
buff[1]:	1E	1C	2E	2C	3E	3C	4E	4C
buff[2]:	1G	1B	2G	2B	3G	3B	4G	4B
buff[3]:	1F	1A	2F	2A	3F	3A	4F	4A
buff[4]:	4.	5D	5.	6D	--	--	--	--
buff[5]:	5E	5C	6E	6C	--	--	--	--
buff[6]:	5G	5B	6G	6B	--	--	--	--
buff[7]:	5F	5A	6F	6A	--	--	--	--

图 7: 驱动效果照片



本 LCD 扫描程序仅需要两个函数:

1、LCD 段码扫描函数 void LCD_scan(void)

程序隔一定的时间调用这个函数，就会将 LCD 显示缓冲的内容显示到 LCD 上，全部扫描一次需要 8 个调用周期，调用间隔一般是 1~2ms，假如使用 1ms，则扫描周期就是 8ms，刷新率就是 125HZ。

2、LCD 段码显示缓冲装载函数 void LCD_load(u8 n,u8 dat)

本函数用来将显示的数字或字符放在 LCD 显示缓冲中，比如 LCD_load(1,6)，就是要在第一个数位位置显示数字 6，支持显示 0~9，A~F，其它字符用户可以自己添加。

另外，用宏来显示、熄灭或闪烁冒号或小数点。

汇编代码

```
;用 STC8 系列测试 IO 直接驱动段码 LCD(6 个 8 字 LCD, 1/4 Dutys, 1/3 bias)。
;上电后显示一个时间(时分秒).
```

```
,*****
```

P0M1	DATA	0x93
P0M0	DATA	0x94
P1M1	DATA	0x91
P1M0	DATA	0x92
P2M1	DATA	0x95
P2M0	DATA	0x96
P3M1	DATA	0xB1
P3M0	DATA	0xB2
P4M1	DATA	0xB3
P4M0	DATA	0xB4
P5M1	DATA	0xC9
P5M0	DATA	0xC

<i>P6M1</i>	<i>DATA</i>	<i>0xCB</i>
<i>P6M0</i>	<i>DATA</i>	<i>0xCC</i>
<i>P7M1</i>	<i>DATA</i>	<i>0xE1</i>
<i>P7M0</i>	<i>DATA</i>	<i>0xE2</i>
<i>AUXR</i>	<i>DATA</i>	<i>0x8E</i>
<i>INT_CLKO</i>	<i>DATA</i>	<i>0x8F</i>
<i>IE2</i>	<i>DATA</i>	<i>0xAF</i>
<i>P4</i>	<i>DATA</i>	<i>0xC0</i>
<i>T2H</i>	<i>DATA</i>	<i>0xD6</i>
<i>T2L</i>	<i>DATA</i>	<i>0xD7</i>

,*****

<i>DIS_BLACK</i>	<i>EQU</i>	<i>010H</i>
<i>DIS_</i>	<i>EQU</i>	<i>011H</i>
<i>DIS_A</i>	<i>EQU</i>	<i>00AH</i>
<i>DIS_B</i>	<i>EQU</i>	<i>00BH</i>
<i>DIS_C</i>	<i>EQU</i>	<i>00CH</i>
<i>DIS_D</i>	<i>EQU</i>	<i>00DH</i>
<i>DIS_E</i>	<i>EQU</i>	<i>00EH</i>
<i>DIS_F</i>	<i>EQU</i>	<i>00FH</i>

<i>B_2ms</i>	<i>BIT</i>	<i>20H.0</i>	<i>;2ms 信号</i>
<i>B_Second</i>	<i>BIT</i>	<i>20H.1</i>	<i>;秒信号</i>
<i>cnt_500ms</i>	<i>DATA</i>	<i>30H</i>	
<i>second</i>	<i>DATA</i>	<i>31H</i>	
<i>minute</i>	<i>DATA</i>	<i>32H</i>	
<i>hour</i>	<i>DATA</i>	<i>33H</i>	
<i>scan_index</i>	<i>DATA</i>	<i>34H</i>	
<i>LCD_buff</i>	<i>DATA</i>	<i>40H</i>	<i>;40H~47H</i>

,*****

<i>ORG</i>	<i>0000H</i>
<i>LJMP</i>	<i>F_Main</i>
<i>ORG</i>	<i>000BH</i>
<i>LJMP</i>	<i>F_Timer0_Interrupt</i>

,*****

<i>ORG</i>	<i>0100H</i>
------------	--------------

F_Main:

<i>CLR</i>	<i>A</i>	
<i>MOV</i>	<i>P3M1, A</i>	<i>; 设置为准双向口</i>
<i>MOV</i>	<i>P3M0, A</i>	
<i>MOV</i>	<i>P5M1, A</i>	<i>; 设置为准双向口</i>
<i>MOV</i>	<i>P5M0, A</i>	
<i>MOV</i>	<i>P1M1, #0</i>	<i>; segment 设置为推挽输出</i>
<i>MOV</i>	<i>P1M0, #0ffh</i>	
<i>ANL</i>	<i>P2M1, #NOT 0f0h</i>	<i>; segment 设置为推挽输出</i>
<i>ORL</i>	<i>P2M0, #0f0h</i>	
<i>ORL</i>	<i>P2M1, #00fH</i>	<i>; 全部 COM 输出高阻, COM 为中点电压</i>
<i>ANL</i>	<i>P2M0, #0f0H</i>	
<i>MOV</i>	<i>SP, #0D0H</i>	
<i>MOV</i>	<i>PSW, #0</i>	
<i>USING</i>	<i>0</i>	<i>; 选择第 0 组 R0~R7</i>

,*****

<i>MOV</i>	<i>R2, #8</i>
------------	---------------

```

        MOV      R0, #LCD_buff
L_ClearLcdRam:
        MOV      @R0, #0
        INC      R0
        DJNZ    R2, L_ClearLcdRam

        LCALL   F_Timer0_init
        SETB    EA

;
;          ORL      LCD_buff, #020H           ;显示时分间隔:
;          ORL      LCD_buff, #002H           ;显示分秒间隔:

        MOV      hour, #12
        MOV      minute, #00
        MOV      second, #00
        LCALL   F_LoadRTC                  ;显示时间

,******



L_Main_Loop:
        JNB     B_2ms, L_Main_Loop        ;2ms 节拍到
        CLR     B_2ms

        INC      cnt_500ms
        MOV      A, cnt_500ms
        CJNE    A, #250, L_Main_Loop      ;500ms 到

        MOV      cnt_500ms, #0;

        XRL      LCD_buff, #020H          ;闪烁时分间隔:
        XRL      LCD_buff, #002H          ;闪烁分秒间隔:

        CPL     B_Second
        JNB     B_Second, L_Main_Loop

        INC      second
        MOV      A, second
        CJNE    A, #60, L_Main_Load      ; 1 分钟到
        INC      minute
        MOV      A, minute
        CJNE    A, #60, L_Main_Load
        MOV      minute, #0;
        INC      hour
        MOV      A, hour
        CJNE    A, #24, L_Main_Load      ;24 小时到
        MOV      hour, #0

L_Main_Load:
        LCALL   F_LoadRTC                ;显示时间
        LJMP    L_Main_Loop

,******



F_Timer0_init:
        CLR      TR0                   ; 停止计数
        ANL      TMOD, #0f0H
        SETB    ET0                   ; 允许中断
        ORL      TMOD, #0
        ANL      INT_CLKO, #NOT 0x01   ; 工作模式 0: 16 位自动重装
        ORL      AUXR, #0x80            ; 不输出时钟
                                ; 1T mode

```

```

MOV      TH0, #HIGH (-22118)      ; 2ms
MOV      TL0, #LOW  (-22118)      ;
SETB    TR0                      ; 开始运行
RET

```

,*****

F_Timer0_Interrupt: ;Timer0 1ms 中断函数

```

PUSH   PSW          ;PSW 入栈
PUSH   ACC          ;ACC 入栈
PUSH   AR0
PUSH   AR7
PUSH   DPH
PUSH   DPL

LCALL  F_LCD_scan
SETB   B_2ms

POP    DPL
POP    DPH
POP    AR7
POP    AR0
POP    ACC          ;ACC 出栈
POP    PSW          ;PSW 出栈
RETI

```

,***** 显示时间 *****

F_LoadRTC:

```

MOV    R6, #1          ;LCD_load(1,hour/10);
MOV    A, hour
MOV    B, #10
DIV    AB
MOV    R7, A
LCALL  F_LCD_load    ;R6 为第几个数字, 为1~6, R7 为要显示的数字

MOV    R6, #2          ;LCD_load(2,hour%10);
MOV    A, hour
MOV    B, #10
DIV    AB
MOV    R7, B
LCALL  F_LCD_load    ;R6 为第几个数字, 为1~6, R7 为要显示的数字

MOV    R6, #3          ;LCD_load(3,minute/10);
MOV    A, minute
MOV    B, #10
DIV    AB
MOV    R7, A
LCALL  F_LCD_load    ;R6 为第几个数字, 为1~6, R7 为要显示的数字

MOV    R6, #4          ;LCD_load(4,minute%10);
MOV    A, minute
MOV    B, #10
DIV    AB
MOV    R7, B
LCALL  F_LCD_load    ;R6 为第几个数字, 为1~6, R7 为要显示的数字

MOV    R6, #5          ;LCD_load(5,second/10);
MOV    A, second
MOV    B, #10
DIV    AB

```

MOV	R7, A	
LCALL	F_LCD_load	<i>;R6 为第几个数字, 为1~6, R7 为要显示的数字</i>
MOV	R6, #6	<i>;LCD_load(6,second%10);</i>
MOV	A, second	
MOV	B, #10	
DIV	AB	
MOV	R7, B	
LCALL	F_LCD_load	<i>;R6 为第几个数字, 为1~6, R7 为要显示的数字</i>
RET		

,*****

T_COM:

DB	008H, 004H, 002H, 001H
-----------	-------------------------------

F_LCD_scan:

MOV	A, scan_index	<i>;j = scan_index >> 1;</i>
CLR	C	
RRC	A	
MOV	R7, A	<i>;R7 = j</i>
ADD	A, #LCD_buff	
MOV	R0, A	<i>;R0 = LCD_buff[j]</i>
ORL	P2M1, #00fH	<i>;全部 COM 输出高阻, COM 为中点电压</i>
ANL	P2M0, #0f0H	
MOV	A, scan_index	
JNB	ACC.0, L_LCD_Scan2	<i>;if(scan_index & 1) //反相扫描</i>
MOV	A, @R0	<i>;P1 = ~LCD_buff[j];</i>
CPL	A	
MOV	P1, A	
MOV	A, R0	<i>;P2 = ~(LCD_buff[j/4] & 0xf0);</i>
ADD	A, #4	
MOV	R0, A	
MOV	A, @R0	
ANL	A, #0f0H	
CPL	A	
MOV	P2, A	
SJMP	L_LCD_Scan3	

L_LCD_Scan2:

MOV	A, @R0	<i>;正相扫描</i>
MOV	P1, A	<i>;P1 = LCD_buff[j];</i>
MOV	A, R0	<i>;P2 = (LCD_buff[j/4] & 0xf0);</i>
ADD	A, #4	
MOV	R0, A	
MOV	A, @R0	
ANL	A, #0f0H	
MOV	P2, A	

L_LCD_Scan3:

MOV	DPTR, #T_COM	<i>;某个 COM 设置为推挽输出</i>
MOV	A, R7	
MOVC	A, @A+DPTR	
ORL	P2M0, A	
CPL	A	
ANL	P2M1, A	
INC	scan_index	<i>;if(++scan_index == 8) scan_index = 0;</i>

```

MOV      A, scan_index
CJNE    A, #8, L_QuitLcdScan
MOV      scan_index, #0

```

L_QuitLcdScan:**RET**

,***** 标准字库 *****

T_Display:

;	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DB	03FH,006H,05BH,04FH,066H,06DH,07DH,007H,07FH,06FH,077H,07CH,039H,05EH,079H,071H															
;	black -															
DB	000H,040H															

,***** 对第1~6 数字装载显示函数 算法简单 *****

F_LCD_load:

```

MOV      DPTR, #T_Display           ;R6 为第几个数字, 为1~6, R7 为要显示的数字
MOV      A, R7
MOVC    A, @A+DPTR
MOV      B, A                   ;要显示的数字

MOV      A, R6
CJNE    A, #1, L_NotLoadChar1
MOV      R0,          #LCD_buff
MOV      A, @R0
MOV      C, B.3                ;D
MOV      ACC.6, C
MOV      @R0, A

INC     R0
MOV      A, @R0
MOV      C, B.2                ;C
MOV      ACC.6, C
MOV      C, B.4                ;E
MOV      ACC.7, C
MOV      @R0, A

INC     R0
MOV      A, @R0
MOV      C, B.1                ;B
MOV      ACC.6, C
MOV      C, B.6                ;G
MOV      ACC.7, C
MOV      @R0, A

INC     R0
MOV      A, @R0
MOV      C, B.0                ;A
MOV      ACC.6, C
MOV      C, B.5                ;F
MOV      ACC.7, C
MOV      @R0, A
RET

```

L_NotLoadChar1:

```

CJNE    A, #2, L_NotLoadChar2
MOV      R0,#LCD_buff
MOV      A, @R0
MOV      C, B.3                ;D

```

```

MOV      ACC.4, C
MOV      @R0, A

INC      R0
MOV      A, @R0
MOV      C, B.2          ;C
MOV      ACC.4, C
MOV      C, B.4          ;E
MOV      ACC.5, C
MOV      @R0, A

INC      R0
MOV      A, @R0
MOV      C, B.1          ;B
MOV      ACC.4, C
MOV      C, B.6          ;G
MOV      ACC.5, C
MOV      @R0, A

INC      R0
MOV      A, @R0
MOV      C, B.0          ;A
MOV      ACC.4, C
MOV      C, B.5          ;F
MOV      ACC.5, C
MOV      @R0, A
RET

```

L_NotLoadChar2:

```

CJNE    A, #3, L_NotLoadChar3
MOV      R0,#LCD_buff
MOV      A, @R0
MOV      C, B.3          ;D
MOV      ACC.2, C
MOV      @R0, A

INC      R0
MOV      A, @R0
MOV      C, B.2          ;C
MOV      ACC.2, C
MOV      C, B.4          ;E
MOV      ACC.3, C
MOV      @R0, A

INC      R0
MOV      A, @R0
MOV      C, B.1          ;B
MOV      ACC.2, C
MOV      C, B.6          ;G
MOV      ACC.3, C
MOV      @R0, A

INC      R0
MOV      A, @R0
MOV      C, B.0          ;A
MOV      ACC.2, C
MOV      C, B.5          ;F
MOV      ACC.3, C
MOV      @R0, A

```

RET***L_NotLoadChar3:***

```

CJNE      A, #4, L_NotLoadChar4
MOV       R0,#LCD_buff
MOV       A, @R0
MOV       C, B.3           ;D
MOV       ACC.0, C
MOV       @R0, A

INC       R0
MOV       A, @R0
MOV       C, B.2           ;C
MOV       ACC.0, C
MOV       C, B.4           ;E
MOV       ACC.1, C
MOV       @R0, A

INC       R0
MOV       A, @R0
MOV       C, B.1           ;B
MOV       ACC.0, C
MOV       C, B.6           ;G
MOV       ACC.1, C
MOV       @R0, A

INC       R0
MOV       A, @R0
MOV       C, B.0           ;A
MOV       ACC.0, C
MOV       C, B.5           ;F
MOV       ACC.1, C
MOV       @R0, A
RET

```

L_NotLoadChar4:

```

CJNE      A, #5, L_NotLoadChar5
MOV       R0,#LCD_buff+4
MOV       A, @R0
MOV       C, B.3           ;D
MOV       ACC.6, C
MOV       @R0, A

INC       R0
MOV       A, @R0
MOV       C, B.2           ;C
MOV       ACC.6, C
MOV       C, B.4           ;E
MOV       ACC.7, C
MOV       @R0, A

INC       R0
MOV       A, @R0
MOV       C, B.1           ;B
MOV       ACC.6, C
MOV       C, B.6           ;G
MOV       ACC.7, C
MOV       @R0, A

```

```

INC      R0
MOV      A, @R0
MOV      C, B.0          ;A
MOV      ACC.6, C
MOV      C, B.5          ;F
MOV      ACC.7, C
MOV      @R0, A
RET

```

L_NotLoadChar5:

```

CJNE    A, #6, L_NotLoadChar6
MOV     R0,#LCD_buff+4
MOV     A, @R0
MOV     C, B.3          ;D
MOV     ACC.4, C
MOV     @R0, A

INC      R0
MOV     A, @R0
MOV     C, B.2          ;C
MOV     ACC.4, C
MOV     C, B.4          ;E
MOV     ACC.5, C
MOV     @R0, A

INC      R0
MOV     A, @R0
MOV     C, B.1          ;B
MOV     ACC.4, C
MOV     C, B.6          ;G
MOV     ACC.5, C
MOV     @R0, A

INC      R0
MOV     A, @R0
MOV     C, B.0          ;A
MOV     ACC.4, C
MOV     C, B.5          ;F
MOV     ACC.5, C
MOV     @R0, A
RET

```

L_NotLoadChar6:

```
RET
```

```
END
```

C 语言代码

```
*****功能说明*****
用STC15 系列测试IO 直接驱动段码LCD(6 个8 字LCD, 1/4 Dutys, 1/3 bias)。
上电后显示一个时间(时分秒)。
P3.2 对地接一个开关,用来进入睡眠或唤醒。
*****/
```

```

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

typedef unsigned char      u8;
typedef unsigned int        u16;

```

```

typedef     unsigned long      u32;

sfr AUXR = 0x8e;
sfr PIM1 = 0x91;
sfr PIM0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;

/****************本地常量声明********************/
#define MAIN_Fosc      11059200L           //定义主时钟

#define DIS_BLACK      0x10
#define DIS_          0x11
#define DIS_A         0x0A
#define DIS_B         0x0B
#define DIS_C         0x0C
#define DIS_D         0x0D
#define DIS_E         0x0E
#define DIS_F         0x0F

#define LCD_SET_DP2    LCD_buff[0] /= 0x08
#define LCD_CLR_DP2    LCD_buff[0] &= ~0x08
#define LCD_FLASH_DP2   LCD_buff[0] ^= 0x08

#define LCD_SET_DP4    LCD_buff[4] /= 0x80
#define LCD_CLR_DP4    LCD_buff[4] &= ~0x80
#define LCD_FLASH_DP4   LCD_buff[4] ^= 0x80

#define LCD_SET_2M     LCD_buff[0] /= 0x20
#define LCD_CLR_2M     LCD_buff[0] &= ~0x20
#define LCD_FLASH_2M    LCD_buff[0] ^= 0x20

#define LCD_SET_4M     LCD_buff[0] /= 0x02
#define LCD_CLR_4M     LCD_buff[0] &= ~0x02
#define LCD_FLASH_4M    LCD_buff[0] ^= 0x02

#define LCD_SET_DP5    LCD_buff[4] /= 0x20
#define LCD_CLR_DP5    LCD_buff[4] &= ~0x20
#define LCD_FLASH_DP5   LCD_buff[4] ^= 0x20

#define PIn_standard(bitn)  PIM1 &= ~(bitn), PIM0 &= ~(bitn)
#define PIn_push_pull(bitn) PIM1 &= ~(bitn), PIM0 /= (bitn)
#define PIn_pure_input(bitn) PIM1 /= (bitn), PIM0 &= ~(bitn)
#define PIn_open_drain(bitn) PIM1 /= (bitn), PIM0 /= (bitn)

#define P2n_standard(bitn)  P2M1 &= ~(bitn), P2M0 &= ~(bitn)
#define P2n_push_pull(bitn) P2M1 &= ~(bitn), P2M0 /= (bitn)
#define P2n_pure_input(bitn) P2M1 /= (bitn), P2M0 &= ~(bitn)
#define P2n_open_drain(bitn) P2M1 /= (bitn), P2M0 /= (bitn)

/****************本地变量声明********************/
u8 cnt_500ms;
u8 second,minute,hour;
bit B_Second;
bit B_2ms;
u8 LCD_buff[8];
u8 scan_index;

/****************本地函数声明********************/

```

```

void LCD_load(u8 n,u8 dat);
void LCD_scan(void);
void LoadRTC(void);
void delay_ms(u8 ms);

/******主函数*****/
void main(void)
{
    u8 i;

    AUXR = 0x80;
    TMOD = 0x00;
    TL0 = (65536 - (MAIN_Fosc / 500));
    TH0 = (65536 - (MAIN_Fosc / 500)) >> 8;
    TR0 = 1;
    ET0 = 1;
    EA = 1;

                                            //初始化LCD 显存
    for(i=0; i<8; i++) LCD_buff[i] = 0;
    P2n_push_pull(0xf0);                      //segment 设置为推挽输出
    P1n_push_pull(0xff);

    LCD_SET_2M;                                //显示时分间隔:
    LCD_SET_4M;                                //显示分秒间隔:
    LoadRTC();                                  //显示时间

    while (1)
    {
        PCON |= 0x01;                          //进入空闲模式, 由 Timer0 2ms 唤醒退出
        _nop_();
        _nop_();
        _nop_();

        if(B_2ms)                                //2ms 节拍到
        {
            B_2ms = 0;

            if(++cnt_500ms >= 250)              //500ms 到
            {
                cnt_500ms = 0;
                // LCD_FLASH_2M;                  //闪烁时分间隔:
                // LCD_FLASH_4M;                  //闪烁分秒间隔:

                B_Second = ~B_Second;
                if(B_Second)
                {
                    if(++second >= 60)           //1 分钟到
                    {
                        second = 0;
                        if(++minute >= 60)         //1 小时到
                        {
                            minute = 0;
                            if(++hour >= 24) hour = 0; //24 小时到
                        }
                    }
                }
                LoadRTC();                      //显示时间
            }
        }
    }
}

```

```

if(!INT0)                                //键按下, 准备睡眠
{
    LCD_CLR_2M;                          //显示时分间隔.
    LCD_CLR_4M;                          //显示分秒间隔.
    LCD_load(1,DIS_BLACK);
    LCD_load(2,DIS_BLACK);
    LCD_load(3,0);
    LCD_load(4,0x0F);
    LCD_load(5,0x0F);
    LCD_load(6,DIS_BLACK);

    while(!INT0) delay_ms(10);           //等待释放按键
    delay_ms(50);                      //再次等待释放按键

    TR0 = 0;                            //关闭定时器
    IE0 = 0;                            //外中断0 标志位
    EX0 = 1;                            //INT0 Enable
    IT0 = 1;                            //INT0 下降沿中断

    P1n_push_pull(0xff);                //com 和 seg 全部输出0
    P2n_push_pull(0xff);
    P1 = 0;
    P2 = 0;

    PCON |= 0x02;                      //Sleep
    _nop_();
    _nop_();
    _nop_();

    LCD_SET_2M;                        //显示时分间隔.
    LCD_SET_4M;                        //显示分秒间隔.
    LoadRTC();                         //显示时间
    TR0 = 1;                            //打开定时器
    while(!INT0) delay_ms(10);           //等待释放按键
    delay_ms(50);                      //再次等待释放按键
}
}

}

*******/

void delay_ms(u8 ms)
{
    unsigned int i;
    do{
        i = MAIN_Fosc / 13000;
        while(--i);                      //14T per loop
    }while(--ms);
}

*******/

void timer0_int (void) interrupt 1
{
    LCD_scan();
    B_2ms = 1;
}

```

```

/********************* INT0 中断函数 *****/
void INT0_int (void) interrupt 0
{
    EX0 = 0;
    IE0 = 0;
}

/******************* LCD 段码扫描函数 *****/
void LCD_scan(void) //5us @22.1184MHZ
{
    u8 code T_COM[4]={0x08,0x04,0x02,0x01};
    u8 j;

    j = scan_index >> 1;
    P2n_pure_input(0x0f);
    if(scan_index & 1) //全部 COM 输出高阻, COM 为中点电压
        //反相扫描
    {
        P1 = ~LCD_buff[j];
        P2 = ~(LCD_buff[j]/4) & 0xf0;
    }
    else //正相扫描
    {
        P1 = LCD_buff[j];
        P2 = LCD_buff[j]/4 & 0xf0;
    }
    P2n_push_pull(T_COM[j]);
    if(++scan_index >= 8) scan_index = 0;
}

/****************** 对第1~6 数字装载显示函数 *****/
void LCD_load(u8 n, u8 dat) //n 为第几个数字, dat 为要显示的数字
{
    u8 code t_display[]={ //标准字库
        0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x77,0x7C,0x39,0x5E,0x79,0x71,
        //black -
        0x00,0x40
    };
    u8 code T_LCD_mask[4] = {~0xc0,~0x30,~0x0c,~0x03};
    u8 code T_LCD_mask4[4] = {~0x40,~0x10,~0x04,~0x01};
    u8 i,k;
    u8 *p;

    if((n == 0) || (n > 6)) return;
    i = t_display[dat];

    if(n <= 4) //1~4
    {
        n--;
        p = LCD_buff;
    }
    else
    {
        n = n - 5;
        p = &LCD_buff[4];
    }

    k = 0;
}

```

```
if(i & 0x08) k |= 0x40;                                //D
*p = (*p & T_LCD_mask4[n]) | (k>>2*n);
p++;

k = 0;
if(i & 0x04) k |= 0x40;                                //C
if(i & 0x10) k |= 0x80;                                //E
*p = (*p & T_LCD_mask[n]) | (k>>2*n);
p++;

k = 0;
if(i & 0x02) k |= 0x40;                                //B
if(i & 0x40) k |= 0x80;                                //G
*p = (*p & T_LCD_mask[n]) | (k>>2*n);
p++;

k = 0;
if(i & 0x01) k |= 0x40;                                //A
if(i & 0x20) k |= 0x80;                                //F
*p = (*p & T_LCD_mask[n]) | (k>>2*n);
}

/*********************************************************/  
void LoadRTC(void)
{
    LCD_load(1,hour/10);
    LCD_load(2,hour%10);
    LCD_load(3,minute/10);
    LCD_load(4,minute%10);
    LCD_load(5,second/10);
    LCD_load(6,second%10);
}
```

10 指令系统

助记符	指令说明	字节	时钟
ADD A,Rn	寄存器内容加到累加器	1	1
ADD A,direct	直接地址单元的数据加到累加器	2	1
ADD A,@Ri	间接地址单元的数据加到累加器	1	1
ADD A,#data	立即数加到累加器	2	1
ADDC A,Rn	寄存器带进位加到累加器	1	1
ADDC A,direct	直接地址单元的数据带进位加到累加器	2	1
ADDC A,@Ri	间接地址单元的数据带进位加到累加器	1	1
ADDC A,#data	立即数带进位加到累加器	2	1
SUBB A,Rn	累加器带借位减寄存器内容	1	1
SUBB A,direct	累加器带借位减直接地址单元的内容	2	1
SUBB A,@Ri	累加器带借位减间接地址单元的内容	1	1
SUBB A,#data	累加器带借位减立即数	2	1
INC A	累加器加1	1	1
INC Rn	寄存器加1	1	1
INC direct	直接地址单元加1	2	1
INC @Ri	间接地址单元加1	1	1
DEC A	累加器减1	1	1
DEC Rn	寄存器减1	1	1
DEC direct	直接地址单元减1	2	1
DEC @Ri	间接地址单元减1	1	1
INC DPTR	地址寄存器DPTR加1	1	1
MUL AB	A乘以B, B存放高字节, A存放低字节	1	2
DIV AB	A除以B, B存放余数, A存放商	1	6
DA A	累加器十进制调整	1	3
ANL A,Rn	累加器与寄存器相与	1	1
ANL A,direct	累加器与直接地址单元相与	2	1
ANL A,@Ri	累加器与间接地址单元相与	1	1
ANL A,#data	累加器与立即数相与	2	1
ANL direct,A	直接地址单元与累加器相与	2	1
ANL direct,#data	直接地址单元与立即数相与	3	1
ORL A,Rn	累加器与寄存器相或	1	1
ORL A,direct	累加器与直接地址单元相或	2	1
ORL A,@Ri	累加器与间接地址单元相或	1	1
ORL A,#data	累加器与立即数相或	2	1

ORL	direct,A	直接地址单元与累加器相或	2	1
ORL	direct,#data	直接地址单元与立即数相或	3	1
XRL	A,Rn	累加器与寄存器相异或	1	1
XRL	A,direct	累加器与直接地址单元相异或	2	1
XRL	A,@Ri	累加器与间接地址单元相异或	1	1
XRL	A,#data	累加器与立即数相异或	2	1
XRL	direct,A	直接地址单元与累加器相异或	2	1
XRL	direct,#data	直接地址单元与立即数相异或	3	1
CLR	A	累加器清0	1	1
CPL	A	累加器取反	1	1
RL	A	累加器循环左移	1	1
RLC	A	累加器带进位循环左移	1	1
RR	A	累加器循环右移	1	1
RRC	A	累加器带进位循环右移	1	1
SWAP	A	累加器高低半字节交换	1	1
CLR	C	清零进位位	1	1
CLR	bit	清0直接地址位	2	1
SETB	C	置1进位位	1	1
SETB	bit	置1直接地址位	2	1
CPL	C	进位位求反	1	1
CPL	bit	直接地址位求反	2	1
ANL	C,bit	进位位和直接地址位相与	2	1
ANL	C,/bit	进位位和直接地址位的反码相与	2	1
ORL	C,bit	进位位和直接地址位相或	2	1
ORL	C,/bit	进位位和直接地址位的反码相或	2	1
MOV	C,bit	直接地址位送入进位位	2	1
MOV	bit,C	进位位送入直接地址位	2	1
MOV	A,Rn	寄存器内容送入累加器	1	1
MOV	A,direct	直接地址单元中的数据送入累加器	2	1
MOV	A,@Ri	间接地址中的数据送入累加器	1	1
MOV	A,#data	立即数送入累加器	2	1
MOV	Rn,A	累加器内容送入寄存器	1	1
MOV	Rn,direct	直接地址单元中的数据送入寄存器	2	1
MOV	Rn,#data	立即数送入寄存器	2	1
MOV	direct,A	累加器内容送入直接地址单元	2	1
MOV	direct,Rn	寄存器内容送入直接地址单元	2	1
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	1

MOV	direct,@Ri	间接地址中的数据送入直接地址单元	2	1
MOV	direct,#data	立即数送入直接地址单元	3	1
MOV	@Ri,A	累加器内容送间接地址单元	1	1
MOV	@Ri,direct	直接地址单元数据送入间接地址单元	2	1
MOV	@Ri,#data	立即数送入间接地址单元	2	1
MOV	DPTR,#data16	16位立即数送入数据指针	3	1
MOVC	A,@A+DPTR	以DPTR为基址址变址寻址单元中的数据送入累加器	1	4
MOVC	A,@A+PC	以PC为基址变址寻址单元中的数据送入累加器	1	3
MOVX	A,@Ri	扩展地址(8位地址)的内容送入累加器A中	1	3 ^[1]
MOVX	A,@DPTR	扩展RAM(16位地址)的内容送入累加器A中	1	2 ^[1]
MOVX	@Ri,A	将累加器A的内容送入扩展RAM(8位地址)中	1	3 ^[1]
MOVX	@DPTR,A	将累加器A的内容送入扩展RAM(16位地址)中	1	2 ^[1]
PUSH	direct	直接地址单元中的数据压入堆栈	2	1
POP	direct	栈底数据弹出送入直接地址单元	2	1
XCH	A,Rn	寄存器与累加器交换	1	1
XCH	A,direct	直接地址单元与累加器交换	2	1
XCH	A,@Ri	间接地址与累加器交换	1	1
XCHD	A,@Ri	间接地址的低半字节与累加器交换	1	1
ACALL	addr11	短调用子程序	2	3
LCALL	addr16	长调用子程序	3	3
RET		子程序返回	1	3
RETI		中断返回	1	3
AJMP	addr11	短跳转	2	3
LJMP	addr16	长跳转	3	3
SJMP	rel	相对跳转	2	3
JMP	@A+DPTR	相对于DPTR的间接跳转	1	4
JZ	rel	累加器为零跳转	2	1/3 ^[2]
JNZ	rel	累加器非零跳转	2	1/3 ^[2]
JC	rel	进位位为1跳转	2	1/3 ^[2]
JNC	rel	进位位为0跳转	2	1/3 ^[2]
JB	bit,rel	直接地址位为1则跳转	3	1/3 ^[2]
JNB	bit,rel	直接地址位为0则跳转	3	1/3 ^[2]
JBC	bit,rel	直接地址位为1则跳转, 该位清0	3	1/3 ^[2]
CJNE	A,direct,rel	累加器与直接地址单元不相等跳转	3	2/3 ^[3]
CJNE	A,#data,rel	累加器与立即数不相等跳转	3	1/3 ^[2]
CJNE	Rn,#data,rel	寄存器与立即数不相等跳转	3	2/3 ^[3]
CJNE	@Ri,#data,rel	间接地址单元与立即数不相等跳转	3	2/3 ^[3]

DJNZ Rn,rel	寄存器减1后非零跳转	2	2/3 ^[3]
DJNZ direct,rel	直接地址单元减1后非零跳转	3	2/3 ^[3]
NOP	空操作	1	1

[1]: 访问外部扩展 RAM 时, 指令的执行周期与寄存器 BUS_SPEED 中的 SPEED[1:0]位有关

[2]: 对于条件跳转语句的执行时间会依据条件是否满足而不同。当条件不满足时, 不会发生跳转而继续执行下一条指令, 此时条件跳转语句的执行时间为 1 个时钟; 当条件满足时, 则会发生跳转, 此时条件跳转语句的执行时间为 3 个时钟。

[3]: 对于条件跳转语句的执行时间会依据条件是否满足而不同。当条件不满足时, 不会发生跳转而继续执行下一条指令, 此时条件跳转语句的执行时间为 2 个时钟; 当条件满足时, 则会发生跳转, 此时条件跳转语句的执行时间为 3 个时钟。

11 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。

用户可以用关总中断允许位（EA/IE.7）或相应中断的允许位屏蔽相应的中断请求，也可以用打开相应的中断允许位来使 CPU 响应相应的中断申请，每一个中断源可以用软件独立地控制为开中断或关中断状态，部分中断的优先级别均可用软件设置。高优先级的中断请求可以打断低优先级的中断，反之，低优先级的中断请求不可以打断高优先级的中断。当两个相同优先级的中断同时产生时，将由查询次序来决定系统先响应哪个中断。

11.1 STC8 系列中断源

下表中 √ 表示对应的系列有相应的中断源

中断源	STC8A8K64S4A12 系列	STC8A4K64S2A12 系列	STC8F2K64S4 系列	STC8F2K64S2 系列
外部中断 0 中断 (INT0)	√	√	√	√
定时器 0 中断 (Timer0)	√	√	√	√
外部中断 1 中断 (INT1)	√	√	√	√
定时器 1 中断 (Timer1)	√	√	√	√
串口 1 中断 (UART1)	√	√	√	√
模数转换中断 (ADC)	√	√		
低压检测中断 (LVD)	√	√	√	√
捕获中断 (CCP/PCA/PWM)	√	√	—	
串口 2 中断 (UART2)	√	√	√	√
串行外设接口中断 (SPI)	√	√	√	√
外部中断 2 中断 (INT2)	√	√	√	√
外部中断 3 中断 (INT3)	√	√	√	√
定时器 2 中断 (Timer2)	√	√	√	√
外部中断 4 中断 (INT4)	√	√	√	√
串口 3 中断 (UART3)	√		√	

串口 4 中断 (UART4)	√		√	
定时器 3 中断 (Timer3)	√	√	√	√
定时器 4 中断 (Timer4)	√	√	√	√
比较器中断 (CMP)	√	√	√	√
增强型 PWM 中断	√	√		
PWM 异常检测中断 (PWMFD)	√	√		
I2C 总线中断	√	√	√	√
CAN 总线中断	✗	✗		

11.1.1 STC8A8K64S4A12 系列中断源

STC8A8K64S4A12 系列单片机提供了 22 个中断请求源，它们分别是：外部中断 0 中断 (INT0)，定时器 0 中断 (Timer0)，外部中断 1 中断 (INT1)，定时器 1 中断 (Timer1)，串口 1 中断 (UART1)，模数转换中断 (ADC)，低压检测中断 (LVD)，捕获中断 (CCP/PCA/PWM)，串口 2 中断 (UART2)，串行外设接口中断 (SPI)，外部中断 2 中断 (INT2)，外部中断 3 中断 (INT3)，定时器 2 中断 (Timer2)，外部中断 4 中断 (INT4)，串口 3 中断 (UART3)，串口 4 中断 (UART4)，定时器 3 中断 (Timer3)，定时器 4 中断 (Timer4)，比较器中断 (CMP)，增强型 PWM 中断，PWM 异常检测中断 (PWMFD)，I2C 总线中断。

除外部中断 2、外部中断 3、串口 3 中断、串口 4 中断、定时器 2 中断、定时器 3 中断、定时器 4 中断固定是最低优先级中断外，其它的中断都具有 4 个中断优先级可以设置。（注：STC8 系列的比较器中断可设置 4 级中断优先级，STC15 系列的比较器中断固定是最低优先级。之前的资料有误，特此进行更正说明）

11.1.2 STC8A4K64S2A12 系列中断源

STC8A4K64S2A12 系列单片机提供了 20 个中断请求源，它们分别是：外部中断 0 中断 (INT0)，定时器 0 中断 (Timer0)，外部中断 1 中断 (INT1)，定时器 1 中断 (Timer1)，串口 1 中断 (UART1)，模数转换中断 (ADC)，低压检测中断 (LVD)，捕获中断 (CCP/PCA/PWM)，串口 2 中断 (UART2)，串行外设接口中断 (SPI)，外部中断 2 中断 (INT2)，外部中断 3 中断 (INT3)，定时器 2 中断 (Timer2)，外部中断 4 中断 (INT4)，定时器 3 中断 (Timer3)，定时器 4 中断 (Timer4)，比较器中断 (CMP)，增强型 PWM 中断，PWM 异常检测中断 (PWMFD)，I2C 总线中断。

除外部中断 2、外部中断 3、串口 3 中断、串口 4 中断、定时器 2 中断、定时器 3 中断、定时器 4 中断固定是最低优先级中断外，其它的中断都具有 4 个中断优先级可以设置。（注：STC8 系列的比较器中断可设置 4 级中断优先级，STC15 系列的比较器中断固定是最低优先级。之前的资料有误，特此进行更正说明）

11.1.3 STC8F2K64S4 系列中断源

STC8F2K64S4 系列单片机提供了 18 个中断请求源，它们分别是：外部中断 0 中断 (INT0)，定时器 0 中断 (Timer0)，外部中断 1 中断 (INT1)，定时器 1 中断 (Timer1)，串口 1 中断 (UART1)，，低压检测中断 (LVD)，捕获中断 (CCP/PCA/PWM)，串口 2 中断 (UART2)，串行外设接口中断 (SPI)，外部中断 2 中断 (INT2)，外部中断 3 中断 (INT3)，定时器 2 中断 (Timer2)，外部中断 4 中断 (INT4)，串口 3 中断 (UART3)，串口 4 中断 (UART4)，定时器 3 中断 (Timer3)，定时器 4 中断 (Timer4)，

比较器中断 (CMP), I2C 总线中断。

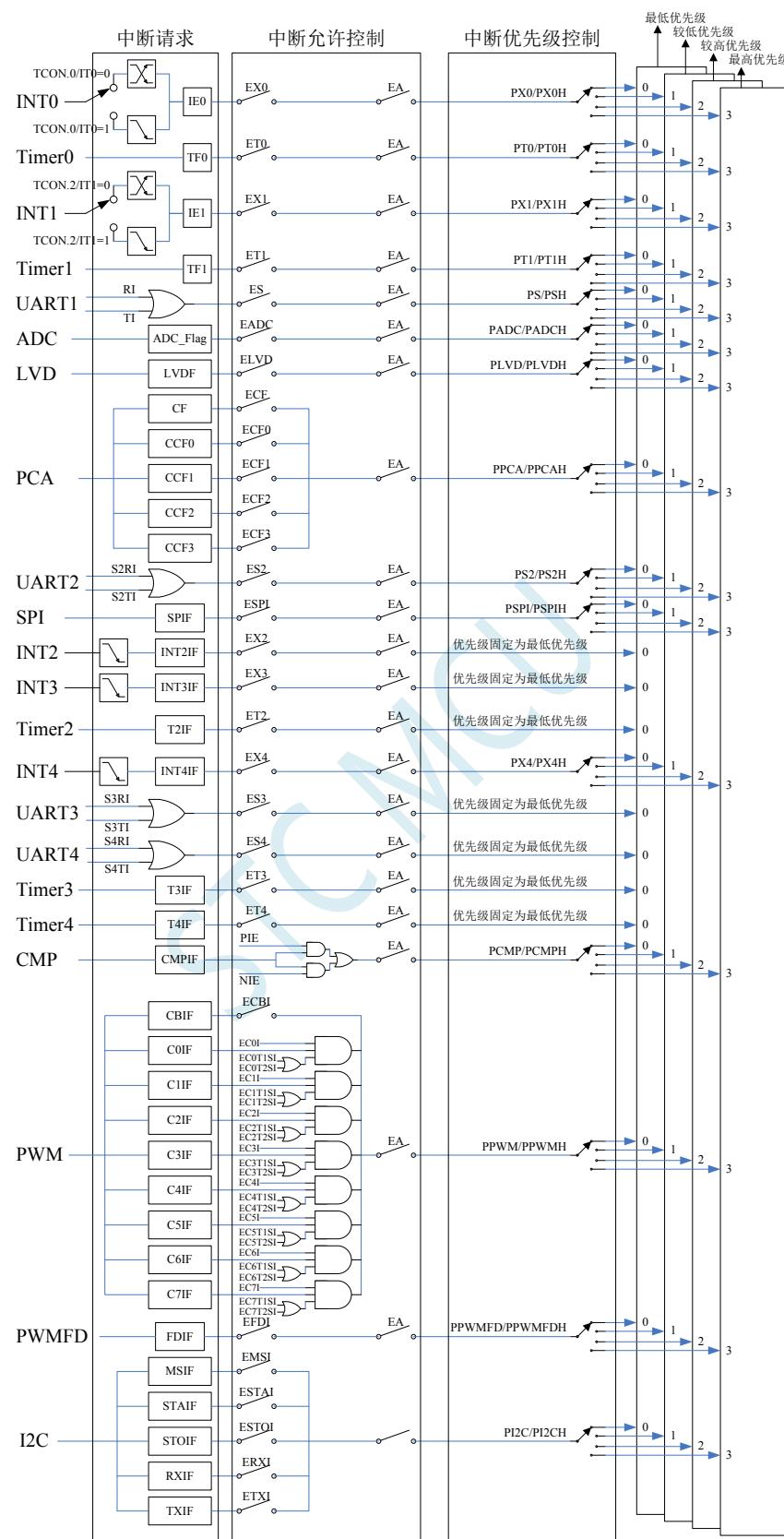
除外部中断 2、外部中断 3、串口 3 中断、串口 4 中断、定时器 2 中断、定时器 3 中断、定时器 4 中断固定是最低优先级中断外，其它的中断都具有 4 个中断优先级可以设置。（**注：STC8 系列的比较器中断可设置 4 级中断优先级，STC15 系列的比较器中断固定是最低优先级。之前的资料有误，特此进行更正说明**）

11.1.4 STC8F2K64S2 系列中断源

STC8F2K64S2 系列单片机提供了 16 个中断请求源，它们分别是：外部中断 0 中断 (INT0)，定时器 0 中断 (Timer0)，外部中断 1 中断 (INT1)，定时器 1 中断 (Timer1)，串口 1 中断 (UART1)，，低压检测中断 (LVD)，串口 2 中断 (UART2)，串行外设接口中断 (SPI)，外部中断 2 中断 (INT2)，外部中断 3 中断 (INT3)，定时器 2 中断 (Timer2)，外部中断 4 中断 (INT4)，定时器 3 中断 (Timer3)，定时器 4 中断 (Timer4)，比较器中断 (CMP)，I2C 总线中断。

除外部中断 2、外部中断 3、定时器 2 中断、定时器 3 中断、定时器 4 中断固定是最低优先级中断外，其它的中断都具有 4 个中断优先级可以设置。（**注：STC8 系列的比较器中断可设置 4 级中断优先级，STC15 系列的比较器中断固定是最低优先级。之前的资料有误，特此进行更正说明**）

11.2 STC8 中断结构图



11.3 STC8 系列中断列表

中断源	中断向量	次序	优先级设置	优先级	中断请求位	中断允许位
INT0	0003H	0	PX0,PX0H	0/1/2/3	IE0	EX0
Timer0	000BH	1	PT0,PT0H	0/1/2/3	TF0	ET0
INT1	0013H	2	PX1,PX1H	0/1/2/3	IE1	EX1
Timer1	001BH	3	PT1,PT1H	0/1/2/3	TF1	ET1
UART1	0023H	4	PS,PSH	0/1/2/3	RI TI	ES
ADC	002BH	5	PADC,PADCH	0/1/2/3	ADC_FLAG	EADC
LVD	0033H	6	PLVD,PLVDH	0/1/2/3	LVDF	ELVD
PCA	003BH	7	PPCA,PPCAH	0/1/2/3	CF	ECF
					CCF0	ECCF0
					CCF1	ECCF1
					CCF2	ECCF2
					CCF3	ECCF3
UART2	0043H	8	PS2,PS2H	0/1/2/3	S2RI S2TI	ES2
SPI	004BH	9	PSPI,PSPIH	0/1/2/3	SPIF	ESPI
INT2	0053H	10		0	INT2IF	EX2
INT3	005BH	11		0	INT3IF	EX3
Timer2	0063H	12		0	T2IF	ET2
INT4	0083H	16	PX4,PX4H	0/1/2/3	INT4IF	EX4
UART3	008BH	17	PS3,PS3H	0/1/2/3	S3RI S3TI	ES3
UART4	0093H	18	PS4,PS4H	0/1/2/3	S4RI S4TI	ES4
Timer3	009BH	19		0	T3IF	ET3
Timer4	00A3H	20		0	T4IF	ET4
CMP	00ABH	21	PCMP,PCMPH	0/1/2/3	CMPIF	PIE NIE

中断源	中断向量	次序	优先级设置	优先级	中断请求位	中断允许位
PWM	00B3H	22	PPWM,PPWMH	0/1/2/3	CBIF	ECBI
					C0IF	EC0I && EC0T1SI
						EC0I && EC0T2SI
					C1IF	EC1I && EC1T1SI
						EC1I && EC1T2SI
					C2IF	EC2I && EC2T1SI
						EC2I && EC2T2SI
					C3IF	EC3I && EC3T1SI
						EC3I && EC3T2SI
					C4IF	EC4I && EC4T1SI
						EC4I && EC4T2SI
					C5IF	EC5I && EC5T1SI
						EC5I && EC5T2SI
					C6IF	EC6I && EC6T1SI
						EC6I && EC6T2SI
					C7IF	EC7I && EC7T1SI
						EC7I && EC7T2SI
PWMFD	00BBH	23	PPWMFD,PPWMFDH	0/1/2/3	FDIF	FDI
I2C	00C3H	24	PI2C,PI2CH	0/1/2/3	MSIF	MSI
					STAIF	ESTAI
					RXIF	ERXI
					TXIF	ETXI
					STOIF	ESTOI
CAN	00CBH	25	PCAN,PCANH	0/1/2/3	CANIF	ECAN

在 C 语言中声明中断服务程序

```

void INT0_Routine(void)           interrupt 0;
void TM0_Routine(void)            interrupt 1;
void INT1_Routine(void)            interrupt 2;
void TM1_Routine(void)             interrupt 3;
void UART1_Routine(void)           interrupt 4;
void ADC_Routine(void)             interrupt 5;
void LVD_Routine(void)             interrupt 6;
void PCA_Routine(void)             interrupt 7;
void UART2_Routine(void)           interrupt 8;
void SPI_Routine(void)              interrupt 9;
void INT2_Routine(void)             interrupt 10;
void INT3_Routine(void)             interrupt 11;
void TM2_Routine(void)              interrupt 12;

```

```
void INT4_Routine(void)    interrupt 16;
void UART3_Routine(void)   interrupt 17;
void UART4_Routine(void)   interrupt 18;
void TM3_Routine(void)    interrupt 19;
void TM4_Routine(void)    interrupt 20;
void CMP_Routine(void)    interrupt 21;
void PWM_Routine(void)    interrupt 22;
void PWMFD_Routine(void)  interrupt 23;
void I2C_Routine(void)    interrupt 24;
```

STCMCU

11.4 中断相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
IE	中断允许寄存器	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000,0000
IE2	中断允许寄存器 2	AFH	PCAN	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000,0000
INTCLKO	中断与时钟输出控制寄存器	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000
IP	中断优先级控制寄存器	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	0000,0000
IPH	高中断优先级控制寄存器	B7H	PPCAH	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H	0000,0000
IP2	中断优先级控制寄存器 2	B5H	PCAN	PI2C	PCMP	PX4	PPWMFD	PPWM	PSPI	PS2	x000,0000
IP2H	高中断优先级控制寄存器 2	B6H	PCANH	PI2CH	PCMPH	PX4H	PPWMFDH	PPWMH	PSPIH	PS2H	x000,0000
IP3	中断优先级控制寄存器 3	D8H	-	-	-	-	-	-	PS4	PS3	xxxx,xx00
IP3H	高中断优先级控制寄存器 3	EEH	-	-	-	-	-	-	PS4H	PS3H	xxxx,xx00
TCON	定时器控制寄存器	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
AUXINTIF	扩展外部中断标志寄存器	EFH	-	INT4IF	INT3IF	INT2IF	-	T4IF	T3IF	T2IF	x000,x000
SCON	串口 1 控制寄存器	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
S2CON	串口 2 控制寄存器	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0100,0000
S3CON	串口 3 控制寄存器	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000
S4CON	串口 4 控制寄存器	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000,0000
PCON	电源控制寄存器	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
ADC_CONTR	ADC 控制寄存器	BCH	ADC_POWER	ADC_START	ADC_FLAG	-	ADC_CHS[3:0]				000x,0000
SPSTAT	SPI 状态寄存器	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx,xxxx
CCON	PCA 控制寄存器	D8H	CF	CR	-	-	CCF3	CCF2	CCF1	CCF0	00xx,0000
CMOD	PCA 模式寄存器	D9H	CIDL	-	-	-	CPS[2:0]			ECF	0xxx,0000
CCAPM0	PCA 模块 0 模式控制寄存器	DAH	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0	x000,0000
CCAPM1	PCA 模块 1 模式控制寄存器	DBH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1	x000,0000
CCAPM2	PCA 模块 2 模式控制寄存器	DCH	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2	x000,0000
CCAPM3	PCA 模块 3 模式控制寄存器	DDH	-	ECOM3	CCAPP3	CCAPN3	MAT3	TOG3	PWM3	ECCF3	x000,0000
CMPCR1	比较器控制寄存器 1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES	0000,0000
PWMCFG	增强型 PWM 配置寄存器	F1H	CBIF	ETADC	-	-	-	-	-	-	00xx,xxxx
PWMCR	PWM 控制寄存器	FEH	ENPWM	ECBI	-	-	-	-	-	-	00xx,xxxx
PWMIF	增强型 PWM 中断标志寄存器	F6H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWMFDRCR	PWM 异常检测控制寄存器	F7H	INVCM	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	0000,0000

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
PWM0CR	PWM0 控制寄存器	FF04H	ENC0O	C0INI	-	C0_S[1:0]		EC0I	EC0T2SI	EC0T1SI	00x0,0000
PWM1CR	PWM1 控制寄存器	FF14H	ENC1O	C1INI	-	C1_S[1:0]		EC1I	EC1T2SI	EC1T1SI	00x0,0000
PWM2CR	PWM2 控制寄存器	FF24H	ENC2O	C2INI	-	C2_S[1:0]		EC2I	EC2T2SI	EC2T1SI	00x0,0000
PWM3CR	PWM3 控制寄存器	FF34H	ENC3O	C3INI	-	C3_S[1:0]		EC3I	EC3T2SI	EC3T1SI	00x0,0000
PWM4CR	PWM4 控制寄存器	FF44H	ENC4O	C4INI	-	C4_S[1:0]		EC4I	EC4T2SI	EC4T1SI	00x0,0000
PWM5CR	PWM5 控制寄存器	FF54H	ENC5O	C5INI	-	C5_S[1:0]		EC5I	EC5T2SI	EC5T1SI	00x0,0000

PWM6CR	PWM6 控制寄存器	FF64H	ENC6O	C6INI	-	C6_S[1:0]		EC6I	EC6T2SI	EC6T1SI	00x0,0000
PWM7CR	PWM7 控制寄存器	FF74H	ENC7O	C7INI	-	C7_S[1:0]		EC7I	EC7T2SI	EC7T1SI	00x0,0000
I2CMSCR	I ² C 主机控制寄存器	FE81H	EMSI	-	-	-	-	MSCMD[2:0]			0xxx,x000
I2CMSST	I ² C 主机状态寄存器	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO	00xx,xx00
I2CSLCR	I ² C 从机控制寄存器	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0
I2CSLST	I ² C 从机状态寄存器	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000

11.4.1 中断使能寄存器（中断允许位）

IE (中断使能寄存器)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

EA: 总中断允许控制位。EA 的作用是使中断允许形成多级控制。即各中断源首先受 EA 控制;其次还受各中断源自己的中断允许控制位控制。

0: CPU 屏蔽所有的中断申请

1: CPU 开放中断

ELVD: 低压检测中断允许位。

0: 禁止低压检测中断

1: 允许低压检测中断

EADC: A/D 转换中断允许位。

0: 禁止 A/D 转换中断

1: 允许 A/D 转换中断

ES: 串行口 1 中断允许位。

0: 禁止串行口 1 中断

1: 允许串行口 1 中断

ET1: 定时/计数器 T1 的溢出中断允许位。

0: 禁止 T1 中断

1: 允许 T1 中断

EX1: 外部中断 1 中断允许位。

0: 禁止 INT1 中断

1: 允许 INT1 中断

ET0: 定时/计数器 T0 的溢出中断允许位。

0: 禁止 T0 中断

1: 允许 T0 中断

EX0: 外部中断 0 中断允许位。

0: 禁止 INT0 中断

1: 允许 INT0 中断

IE2 (中断使能寄存器 2)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IE2	AFH	ECAN	ET4	ET3	ES4	ES3	ET2	ESPI	ES2

ECAN: CAN 中断允许位。（暂无此功能）

0: 禁止 CAN 中断

1: 允许 CAN 中断

ET4: 定时/计数器 T4 的溢出中断允许位。

0: 禁止 T4 中断

1: 允许 T4 中断

ET3: 定时/计数器 T3 的溢出中断允许位。

0: 禁止 T3 中断

1: 允许 T3 中断

ES4: 串行口 4 中断允许位。

0: 禁止串行口 4 中断

1: 允许串行口 4 中断

ES3: 串行口 3 中断允许位。

0: 禁止串行口 3 中断

1: 允许串行口 3 中断

ET2: 定时/计数器 T2 的溢出中断允许位。

0: 禁止 T2 中断

1: 允许 T2 中断

ESPI: SPI 中断允许位。

0: 禁止 SPI 中断

1: 允许 SPI 中断

ES2: 串行口 2 中断允许位。

0: 禁止串行口 2 中断

1: 允许串行口 2 中断

INTCLKO (外部中断与时钟输出控制寄存器)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INTCLKO	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO

EX4: 外部中断 4 中断允许位。

0: 禁止 INT4 中断

1: 允许 INT4 中断

EX3: 外部中断 3 中断允许位。

0: 禁止 INT3 中断

1: 允许 INT3 中断

EX2: 外部中断 2 中断允许位。

0: 禁止 INT2 中断

1: 允许 INT2 中断

PCA/CCP/PWM 中断控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMOD	D9H	CIDL	-	-	-	CPS[2:0]			
CCAPM0	DAH	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0
CCAPM1	DBH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1
CCAPM2	DCH	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2
CCAPM3	DDH	-	ECOM3	CCAPP3	CCAPN3	MAT3	TOG3	PWM3	ECCF3

ECF: PCA 计数器中断允许位。

0: 禁止 PCA 计数器中断

1: 允许 PCA 计数器中断

ECCF0: PCA 模块 0 中断允许位。

0: 禁止 PCA 模块 0 中断

1: 允许 PCA 模块 0 中断

ECCF1: PCA 模块 1 中断允许位。

0: 禁止 PCA 模块 1 中断

1: 允许 PCA 模块 1 中断

ECCF2: PCA 模块 2 中断允许位。

0: 禁止 PCA 模块 2 中断

1: 允许 PCA 模块 2 中断

ECCF3: PCA 模块 3 中断允许位。

0: 禁止 PCA 模块 3 中断

1: 允许 PCA 模块 3 中断

CMPCR1 (比较器控制寄存器 1)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES

PIE: 比较器上升沿中断允许位。

0: 禁止比较器上升沿中断

1: 允许比较器上升沿中断

NIE: 比较器下降沿中断允许位。

0: 禁止比较器下降沿中断

1: 允许比较器下降沿中断

PWMCR (PWM 控制寄存器)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMCR	FEH	ENPWM	ECBI	-	-	-	-	-	-

ECBI: 增强PWM计数器中断允许位。

0: 禁止 PWM 计数器中断

1: 允许 PWM 计数器中断

PWMFDCR (PWM 异常检测控制寄存器)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMFDCR	F7H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF

EFDI: PWM外部异常事件中断允许位。

0: 禁止 PWM 外部异常事件中断

1: 允许 PWM 外部异常事件中断

增强型 PWM 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWM0CR	FF04H	ENC0O	C0INI	-	C0_S[1:0]		EC0I	EC0T2SI	EC0T1SI
PWM1CR	FF14H	ENC1O	C1INI	-	C1_S[1:0]		EC1I	EC1T2SI	EC1T1SI

PWM2CR	FF24H	ENC2O	C2INI	-	C2_S[1:0]	EC2I	EC2T2SI	EC2T1SI
PWM3CR	FF34H	ENC3O	C3INI	-	C3_S[1:0]	EC3I	EC3T2SI	EC3T1SI
PWM4CR	FF44H	ENC4O	C4INI	-	C4_S[1:0]	EC4I	EC4T2SI	EC4T1SI
PWM5CR	FF54H	ENC5O	C5INI	-	C5_S[1:0]	EC5I	EC5T2SI	EC5T1SI
PWM6CR	FF64H	ENC6O	C6INI	-	C6_S[1:0]	EC6I	EC6T2SI	EC6T1SI
PWM7CR	FF74H	ENC7O	C7INI	-	C7_S[1:0]	EC7I	EC7T2SI	EC7T1SI

ECnI: PWM通道n电平翻转中断允许位。

- 0: 禁止第 n 通道 PWM 中断
- 1: 允许第 n 通道 PWM 中断

ECnT2SI: PWM通道n第2个翻转点中断允许位。

- 0: 禁止第 n 通道 PWM 的第 2 个翻转点中断
- 1: 允许第 n 通道 PWM 的第 2 个翻转点中断

ECnT1SI: PWM通道n第1个翻转点中断允许位。

- 0: 禁止第 n 通道 PWM 的第 1 个翻转点中断
- 1: 允许第 n 通道 PWM 的第 1 个翻转点中断

I2C 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSCR	FE81H	EMSI	-	-	-	-	MSCMD[2:0]		
I2CSLCR	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST

EMSI: I²C主机模式中断允许位。

- 0: 禁止 I²C 主机模式中断
- 1: 允许 I²C 主机模式中断

ESTAI: I²C从机接收START事件中断允许位。

- 0: 禁止 I²C 从机接收 START 事件中断
- 1: 允许 I²C 从机接收 START 事件中断

ERXI: I²C从机接收数据完成事件中断允许位。

- 0: 禁止 I²C 从机接收数据完成事件中断
- 1: 允许 I²C 从机接收数据完成事件中断

ETXI: I²C从机发送数据完成事件中断允许位。

- 0: 禁止 I²C 从机发送数据完成事件中断
- 1: 允许 I²C 从机发送数据完成事件中断

ESTOI: I²C从机接收STOP事件中断允许位。

- 0: 禁止 I²C 从机接收 STOP 事件中断
- 1: 允许 I²C 从机接收 STOP 事件中断

11.4.2 中断请求寄存器（中断标志位）

定时器控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TCON	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1: 定时器1溢出中断标志。中断服务程序中，硬件自动清零。

TF0: 定时器0溢出中断标志。中断服务程序中，硬件自动清零。

IE1: 外部中断1中断请求标志。中断服务程序中，硬件自动清零。

IE0: 外部中断0中断请求标志。中断服务程序中，硬件自动清零。

中断标志辅助寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AUXINTIF	EFH	-	INT4IF	INT3IF	INT2IF	-	T4IF	T3IF	T2IF

INT4IF: 外部中断4中断请求标志。需要软件清零。

INT3IF: 外部中断3中断请求标志。需要软件清零。

INT2IF: 外部中断2中断请求标志。需要软件清零。

T4IF: 定时器4溢出中断标志。需要软件清零。

T3IF: 定时器3溢出中断标志。需要软件清零。

T2IF: 定时器2溢出中断标志。需要软件清零。

串口控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SCON	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI
S2CON	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI
S3CON	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI
S4CON	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI

TI: 串口1发送完成中断请求标志。需要软件清零。

RI: 串口1接收完成中断请求标志。需要软件清零。

S2TI: 串口2发送完成中断请求标志。需要软件清零。

S2RI: 串口2接收完成中断请求标志。需要软件清零。

S3TI: 串口3发送完成中断请求标志。需要软件清零。

S3RI: 串口3接收完成中断请求标志。需要软件清零。

S4TI: 串口4发送完成中断请求标志。需要软件清零。

S4RI: 串口4接收完成中断请求标志。需要软件清零。

电源管理寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PCON	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL

LVDF: 低压检测中断请求标志。需要软件清零。

ADC 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0	
ADC_CONTR	BCH	ADC_POWER	ADC_START	ADC_FLAG	-	ADC_CHS[3:0]				

ADC_FLAG: ADC转换完成中断请求标志。需要软件清零。

SPI 状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0

SPSTAT	CDH	SPIF	WCOL	-	-	-	-	-	-
--------	-----	------	------	---	---	---	---	---	---

SPIF: SPI数据传输完成中断请求标志。需要软件清零。

PCA 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CCON	D8H	CF	CR	-	-	CCF3	CCF2	CCF1	CCF0

CF: PCA计数器中断请求标志。需要软件清零。

CCF3: PCA模块3中断请求标志。需要软件清零。

CCF2: PCA模块2中断请求标志。需要软件清零。

CCF1: PCA模块1中断请求标志。需要软件清零。

CCF0: PCA模块0中断请求标志。需要软件清零。

比较器控制寄存器 1

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES

CMPIF: 比较器中断请求标志。需要软件清零。

增强型 PWM 配置寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMCFG	F1H	CBIF	ETADC	-	-	-	-	-	-

CBIF: 增强型PWM计数器中断请求标志。需要软件清零。

增强型 PWM 中断标志寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMIF	F6H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF

C7IF: 增强型PWM通道7中断请求标志。需要软件清零。

C6IF: 增强型PWM通道6中断请求标志。需要软件清零。

C5IF: 增强型PWM通道5中断请求标志。需要软件清零。

C4IF: 增强型PWM通道4中断请求标志。需要软件清零。

C3IF: 增强型PWM通道3中断请求标志。需要软件清零。

C2IF: 增强型PWM通道2中断请求标志。需要软件清零。

C1IF: 增强型PWM通道1中断请求标志。需要软件清零。

C0IF: 增强型PWM通道0中断请求标志。需要软件清零。

增强型 PWM 异常检测控制决寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMFDCR	F7H	INVCM	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF

FDIF: 增强型PWM异常检测中断请求标志。需要软件清零。

I2C 状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0

I2CMSST	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO
I2CSLST	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO

MSIF: I²C主机模式中断请求标志。需要软件清零。

ESTAI: I²C从机接收START事件中断请求标志。需要软件清零。

ERXI: I²C从机接收数据完成事件中断请求标志。需要软件清零。

ETXI: I²C从机发送数据完成事件中断请求标志。需要软件清零。

ESTOI: I²C从机接收STOP事件中断请求标志。需要软件清零。

11.4.3 中断优先级寄存器

中断优先级控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IP	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0
IPH	B7H	PPCAH	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H
IP2	B5H	PCAN	PI2C	PCMP	PX4	PPWMFD	PPWM	PSPI	PS2
IP2H	B6H	PCANH	PI2CH	PCMPH	PX4H	PPWMFDH	PPWMH	PSPIH	PS2H
IP3	DFFH	-	-	-	-	-	-	PS4	PS3
IP3H	EEH	-	-	-	-	-	-	PS4H	PS3H

PX0H,PX0: 外部中断0中断优先级控制位

- 00: INT0 中断优先级为 0 级 (最低级)
- 01: INT0 中断优先级为 1 级 (较低级)
- 10: INT0 中断优先级为 2 级 (较高级)
- 11: INT0 中断优先级为 3 级 (最高级)

PT0H,PT0: 定时器0中断优先级控制位

- 00: 定时器 0 中断优先级为 0 级 (最低级)
- 01: 定时器 0 中断优先级为 1 级 (较低级)
- 10: 定时器 0 中断优先级为 2 级 (较高级)
- 11: 定时器 0 中断优先级为 3 级 (最高级)

PX1H,PX1: 外部中断1中断优先级控制位

- 00: INT1 中断优先级为 0 级 (最低级)
- 01: INT1 中断优先级为 1 级 (较低级)
- 10: INT1 中断优先级为 2 级 (较高级)
- 11: INT1 中断优先级为 3 级 (最高级)

PT1H,PT1: 定时器1中断优先级控制位

- 00: 定时器 1 中断优先级为 0 级 (最低级)
- 01: 定时器 1 中断优先级为 1 级 (较低级)
- 10: 定时器 1 中断优先级为 2 级 (较高级)
- 11: 定时器 1 中断优先级为 3 级 (最高级)

PSH,PS: 串口1中断优先级控制位

- 00: 串口 1 中断优先级为 0 级 (最低级)
- 01: 串口 1 中断优先级为 1 级 (较低级)
- 10: 串口 1 中断优先级为 2 级 (较高级)

11: 串口 1 中断优先级为 3 级 (最高级)

PADCH,PADC: ADC中断优先级控制位

00: ADC 中断优先级为 0 级 (最低级)

01: ADC 中断优先级为 1 级 (较低级)

10: ADC 中断优先级为 2 级 (较高级)

11: ADC 中断优先级为 3 级 (最高级)

PLVDH,PLVD: 低压检测中断优先级控制位

00: LVD 中断优先级为 0 级 (最低级)

01: LVD 中断优先级为 1 级 (较低级)

10: LVD 中断优先级为 2 级 (较高级)

11: LVD 中断优先级为 3 级 (最高级)

PPCAH,PPCA: CCP/PCA/PWM中断优先级控制位

00: CCP/PCA/PWM 中断优先级为 0 级 (最低级)

01: CCP/PCA/PWM 中断优先级为 1 级 (较低级)

10: CCP/PCA/PWM 中断优先级为 2 级 (较高级)

11: CCP/PCA/PWM 中断优先级为 3 级 (最高级)

PS2H,PS2: 串口2中断优先级控制位

00: 串口 2 中断优先级为 0 级 (最低级)

01: 串口 2 中断优先级为 1 级 (较低级)

10: 串口 2 中断优先级为 2 级 (较高级)

11: 串口 2 中断优先级为 3 级 (最高级)

PSPIH,PSPI: SPI中断优先级控制位

00: SPI 中断优先级为 0 级 (最低级)

01: SPI 中断优先级为 1 级 (较低级)

10: SPI 中断优先级为 2 级 (较高级)

11: SPI 中断优先级为 3 级 (最高级)

PPWMH,PPWM: 增强型PWM中断优先级控制位

00: 增强型 PWM 中断优先级为 0 级 (最低级)

01: 增强型 PWM 中断优先级为 1 级 (较低级)

10: 增强型 PWM 中断优先级为 2 级 (较高级)

11: 增强型 PWM 中断优先级为 3 级 (最高级)

PPWMFDH,PPWMFD: 增强型PWM异常检测中断优先级控制位

00: PWMFD 中断优先级为 0 级 (最低级)

01: PWMFD 中断优先级为 1 级 (较低级)

10: PWMFD 中断优先级为 2 级 (较高级)

11: PWMFD 中断优先级为 3 级 (最高级)

PX4H,PX4: 外部中断4中断优先级控制位

00: INT4 中断优先级为 0 级 (最低级)

01: INT4 中断优先级为 1 级 (较低级)

10: INT4 中断优先级为 2 级 (较高级)

11: INT4 中断优先级为 3 级 (最高级)

PCMPH,PCMP: 比较器中断优先级控制位

00: CMP 中断优先级为 0 级 (最低级)

01: CMP 中断优先级为 1 级 (较低级)

10: CMP 中断优先级为 2 级 (较高级)

11: CMP 中断优先级为 3 级 (最高级)

PI2CH,PI2C: I2C中断优先级控制位

00: I2C 中断优先级为 0 级 (最低级)

01: I2C 中断优先级为 1 级 (较低级)

10: I2C 中断优先级为 2 级 (较高级)

11: I2C 中断优先级为 3 级 (最高级)

PCANH,PCAN: CAN中断优先级控制位 (暂无此功能)

00: CAN 中断优先级为 0 级 (最低级)

01: CAN 中断优先级为 1 级 (较低级)

10: CAN 中断优先级为 2 级 (较高级)

11: CAN 中断优先级为 3 级 (最高级)

PS3H,PS3: 串口3中断优先级控制位 (暂无此功能)

00: 串口 3 中断优先级为 0 级 (最低级)

01: 串口 3 中断优先级为 1 级 (较低级)

10: 串口 3 中断优先级为 2 级 (较高级)

11: 串口 3 中断优先级为 3 级 (最高级)

PS4H,PS4: 串口4中断优先级控制位 (暂无此功能)

00: 串口 4 中断优先级为 0 级 (最低级)

01: 串口 4 中断优先级为 1 级 (较低级)

10: 串口 4 中断优先级为 2 级 (较高级)

11: 串口 4 中断优先级为 3 级 (最高级)

11.5 范例程序

11.5.1 INT0 中断（上升沿和下降沿）

汇编代码

<i>ORG</i>	<i>0000H</i>	
<i>LJMP</i>	<i>MAIN</i>	
<i>ORG</i>	<i>0003H</i>	
<i>LJMP</i>	<i>INT0ISR</i>	
<i>ORG</i>	<i>0100H</i>	
<i>INT0ISR:</i>		
<i>JB</i>	<i>INT0,RISING</i>	;判断上升沿和下降沿
<i>CPL</i>	<i>P1.0</i>	;测试端口
<i>RETI</i>		
<i>RISING:</i>		
<i>CPL</i>	<i>P1.1</i>	;测试端口
<i>RETI</i>		
 <i>MAIN:</i>		
<i>MOV</i>	<i>SP,#3FH</i>	
<i>CLR</i>	<i>IT0</i>	;使能 INT0 上升沿和下降沿中断
<i>SETB</i>	<i>EX0</i>	;使能 INT0 中断
<i>SETB</i>	<i>EA</i>	
<i>JMP</i>	<i>\$</i>	
 <i>END</i>		

C 语言代码

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

sbit P10 = P1^0;
sbit P1I = P1^1;

void INT0_Isr() interrupt 0
{
    if (INT0) //判断上升沿和下降沿
    {
        P10 = !P10; //测试端口
    }
    else
    {
        P1I = !P1I; //测试端口
    }
}

void main()
{
    IT0 = 0; //使能 INT0 上升沿和下降沿中断
    EX0 = 1; //使能 INT0 中断
    EA = 1;

    while (1);
}
```

11.5.2 INT0 中断（下降沿）

汇编代码

	<i>ORG</i>	<i>0000H</i>
	<i>LJMP</i>	<i>MAIN</i>
	<i>ORG</i>	<i>0003H</i>
	<i>LJMP</i>	<i>INT0ISR</i>
	 <i>INT0ISR:</i>	
	<i>ORG</i>	<i>0100H</i>
	<i>CPL</i>	<i>P1.0</i>
		; 测试端口
	<i>RETI</i>	
	 <i>MAIN:</i>	
	<i>MOV</i>	<i>SP,#3FH</i>
	<i>SETB</i>	<i>IT0</i>
	<i>SETB</i>	<i>EX0</i>
		; 使能 INT0 下降沿中断
	<i>SETB</i>	<i>EA</i>
	<i>JMP</i>	<i>\$</i>
	 <i>END</i>	

C 语言代码

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

sbit P10 = P1^0;

void INT0_Isr() interrupt 0
{
    P10 = !P10;
    // 测试端口
}

void main()
{
    IT0 = 1;           // 使能 INT0 下降沿中断
    EX0 = 1;           // 使能 INT0 中断
    EA = 1;
    while (1);
}
```

11.5.3 INT1 中断（上升沿和下降沿）

汇编代码

	<i>ORG</i>	<i>0000H</i>
	<i>LJMP</i>	<i>MAIN</i>
	<i>ORG</i>	<i>0013H</i>
	<i>LJMP</i>	<i>INT1ISR</i>
	 <i>INT1ISR:</i>	
	<i>ORG</i>	<i>0100H</i>
	<i>JB</i>	<i>INT1,RISING</i>
		; 判断上升沿和下降沿

<i>CPL</i>	<i>P1.0</i>	; 测试端口
<i>RETI</i>		

RISING:

<i>CPL</i>	<i>P1.1</i>	; 测试端口
<i>RETI</i>		

MAIN:

<i>MOV</i>	<i>SP, #3FH</i>	
------------	-----------------	--

<i>CLR</i>	<i>IT1</i>	; 使能 INT1 上升沿和下降沿中断
<i>SETB</i>	<i>EXI</i>	; 使能 INT1 中断
<i>SETB</i>	<i>EA</i>	
<i>JMP</i>	<i>\$</i>	

END

C 语言代码

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

sbit P10 = P1^0;
sbit P11 = P1^1;

void INT1_Isr() interrupt 2
{
    if (INT1) // 判断上升沿和下降沿
    {
        P10 = !P10; // 测试端口
    }
    else
    {
        P11 = !P11; // 测试端口
    }
}

void main()
{
    IT1 = 0; // 使能 INT1 上升沿和下降沿中断
    EXI = 1; // 使能 INT1 中断
    EA = 1;

    while (1);
}
```

11.5.4 INT1 中断（下降沿）

汇编代码

<i>ORG</i>	<i>0000H</i>	
<i>LJMP</i>	<i>MAIN</i>	
<i>ORG</i>	<i>0013H</i>	
<i>LJMP</i>	<i>INT1ISR</i>	

<i>INT1ISR:</i>	<i>ORG</i>	<i>0100H</i>	
	<i>CPL</i>	<i>P1.0</i>	; 测试端口
	<i>RETI</i>		

MAIN:

```

MOV      SP,#3FH

SETB    IT1          ;使能INT1 下降沿中断
SETB    EX1          ;使能INT1 中断
SETB    EA
JMP     $

```

END

C 语言代码

```

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

sbit   P10      =  P1^0;

void INT1_Isr() interrupt 2
{
    P10 = !P10;           //测试端口
}

void main()
{
    IT1 = 1;            //使能INT1 下降沿中断
    EX1 = 1;            //使能INT1 中断
    EA = 1;

    while (1);
}

```

11.5.5 INT2 中断（下降沿）**汇编代码**

INTCLKO	DATA	8FH
EX2	EQU	10H
EX3	EQU	20H
EX4	EQU	40H
ORG	0000H	
LJMP	MAIN	
ORG	0053H	
LJMP	INT2ISR	
INT2ISR:	ORG	0100H
	CPL	P1.0 ;测试端口
	RETI	
MAIN:	MOV	SP,#3FH
	MOV	INTCLKO,#EX2 ;使能INT2 中断
	SETB	EA
	JMP	\$

END

C 语言代码

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

sfr INTCLKO = 0x8f;
#define EX2 0x10
#define EX3 0x20
#define EX4 0x40
sbit P10 = P1^0;

void INT2_Isr() interrupt 10
{
    P10 = !P10; //测试端口
}

void main()
{
    INTCLKO = EX2; //使能INT2 中断
    EA = 1;
    while (1);
}
```

11.5.6 INT3 中断（下降沿）

汇编代码

<i>INTCLKO</i>	<i>DATA</i>	<i>8FH</i>
<i>EX2</i>	<i>EQU</i>	<i>10H</i>
<i>EX3</i>	<i>EQU</i>	<i>20H</i>
<i>EX4</i>	<i>EQU</i>	<i>40H</i>
<i>ORG</i>	<i>0000H</i>	
<i>LJMP</i>	<i>MAIN</i>	
<i>ORG</i>	<i>005BH</i>	
<i>LJMP</i>	<i>INT3ISR</i>	
<i>ORG</i>	<i>0100H</i>	
<i>INT3ISR:</i>	<i>CPL</i>	<i>P1.0</i> ;测试端口
	<i>RETI</i>	
<i>MAIN:</i>		
<i>MOV</i>	<i>SP,#3FH</i>	
<i>MOV</i>	<i>INTCLKO,#EX3</i>	;使能INT3 中断
<i>SETB</i>	<i>EA</i>	
<i>JMP</i>	<i>\$</i>	
<i>END</i>		

C 语言代码

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

```

sfr      INTCLKO      = 0x8f;
#define  EX2          0x10
#define  EX3          0x20
#define  EX4          0x40
sbit    P10          = P1^0;

void INT3_Isr() interrupt 11
{
    P10 = !P10;           //测试端口
}

void main()
{
    INTCLKO = EX3;        //使能INT3 中断
    EA = 1;

    while (1);
}

```

11.5.7 INT4 中断（下降沿）

汇编代码

```

INTCLKO      DATA      8FH
EX2          EQU       10H
EX3          EQU       20H
EX4          EQU       40H

ORG          0000H
LJMP         MAIN
ORG          0083H
LJMP         INT4ISR

ORG          0100H
INT4ISR:
    CPL        P1.0           ;测试端口
    RETI

MAIN:
    MOV        SP,#3FH
    MOV        INTCLKO,#EX4      ;使能INT4 中断
    SETB       EA
    JMP        $

```

END

C 语言代码

```

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

sfr      INTCLKO      = 0x8f;
#define  EX2          0x10
#define  EX3          0x20
#define  EX4          0x40
sbit    P10          = P1^0;

```

```

void INT4_Isr() interrupt 16
{
    P10 = !P10;                                //测试端口
}

void main()
{
    INTCLKO = EX4;                            //使能INT4 中断
    EA = 1;

    while (1);
}

```

11.5.8 定时器 0 中断

汇编代码

//测试工作频率为 11.0592MHz

```

        ORG      0000H
        LJMP    MAIN
        ORG      000BH
        LJMP    TM0ISR

        ORG      0100H
TM0ISR:
        CPL     P1.0           ;测试端口
        RETI

MAIN:
        MOV     SP,#3FH
        MOV     TMOD,#00H
        MOV     TL0,#66H          ;65536-11.0592M/12/1000
        MOV     TH0,#0FCH
        SETB   TR0              ;启动定时器
        SETB   ET0              ;使能定时器中断
        SETB   EA

        JMP     $

END

```

C 语言代码

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

//测试工作频率为 11.0592MHz

```

sbit      P10      =  P1^0;

void TM0_Isr() interrupt 1
{
    P10 = !P10;                                //测试端口
}

```

```

void main()
{
    TMOD = 0x00;
    TL0 = 0x66;                                //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                                    //启动定时器
    ET0 = 1;                                    //使能定时器中断
    EA = 1;

    while (1);
}

```

11.5.9 定时器 1 中断

汇编代码

//测试工作频率为 11.0592MHz

```

        ORG      0000H
        LJMP    MAIN
        ORG      001BH
        LJMP    TMIISR

        ORG      0100H
TMIISR:
        CPL      P1.0           ; 测试端口
        RETI

MAIN:
        MOV      SP,#3FH

        MOV      TMOD,#00H
        MOV      TLI,#66H          ;65536-11.0592M/12/1000
        MOV      THI,#0FCH
        SETB    TRI               ;启动定时器
        SETB    ETI               ;使能定时器中断
        SETB    EA

        JMP      $

END

```

C 语言代码

```

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

//测试工作频率为 11.0592MHz

sbit     P10      =  P1^0;

void TM1_Isr() interrupt 3
{
    P10 = !P10;           // 测试端口
}

void main()
{

```

```

TMOD = 0x00;
TL1 = 0x66;                                //65536-11.0592M/12/1000
TH1 = 0xfc;
TR1 = 1;                                 //启动定时器
ET1 = 1;                                 //使能定时器中断
EA = 1;

while (1);
}

```

11.5.10 定时器 2 中断

汇编代码

//测试工作频率为 11.0592MHz

```

T2L      DATA    0D7H
T2H      DATA    0D6H
AUXR     DATA    8EH
IE2      DATA    0AFH
ET2      EQU     04H
AUXINTIF  DATA   0EFH
T2IF     EQU     01H

ORG      0000H
LJMP    MAIN
ORG      0063H
LJMP    TM2ISR

ORG      0100H
TM2ISR:
CPL      P1.0
ANL      AUXINTIF,#NOT T2IF          ;测试端口
RETI

MAIN:
MOV      SP,#3FH
MOV      T2L,#66H                  ;65536-11.0592M/12/1000
MOV      T2H,#0FCH
MOV      AUXR,#10H                ;启动定时器
MOV      IE2,#ET2                 ;使能定时器中断
SETB    EA

JMP      $

```

END

C 语言代码

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

//测试工作频率为 11.0592MHz

```
sfr      T2L      = 0xd7;
sfr      T2H      = 0xd6;
sfr      AUXR     = 0x8e;
```

```

sfr     IE2          = 0xaf;
#define ET2          0x04
sfr     AUXINTIF    = 0xef;
#define T2IF         0x01

sbit    P10          = P1^0;

void TM2_Isr() interrupt 12
{
    P10 = !P10;           //测试端口
    AUXINTIF &= ~T2IF;   //清中断标志
}

void main()
{
    T2L = 0x66;          //65536-11.0592M/12/1000
    T2H = 0xfc;
    AUXR = 0x10;          //启动定时器
    IE2 = ET2;            //使能定时器中断
    EA = 1;

    while (1);
}

```

11.5.11 定时器 3 中断

汇编代码

; 测试工作频率为 11.0592MHz

```

T3L      DATA      0D5H
T3H      DATA      0D4H
T4T3M    DATA      0DIH
IE2      DATA      0AFH
ET3      EQU       20H
AUXINTIF DATA      0EFH
T3IF    EQU       02H

        ORG      0000H
        LJMP    MAIN
        ORG      009BH
        LJMP    TM3ISR

        ORG      0100H
TM3ISR:
        CPL      P1.0          ; 测试端口
        ANL      AUXINTIF,#NOT T3IF ; 清中断标志
        RETI

MAIN:
        MOV      SP,#3FH

        MOV      T3L,#66H          ;65536-11.0592M/12/1000
        MOV      T3H,#0FCH
        MOV      T4T3M,#08H          ;启动定时器
        MOV      IE2,#ET3            ;使能定时器中断
        SETB    EA

        JMP      $

```

END

C 语言代码

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

//测试工作频率为11.0592MHz

sfr      T3L          = 0xd5;
sfr      T3H          = 0xd4;
sfr      T4T3M        = 0xd1;
sfr      IE2          = 0xaf;
#define   ET3          0x20
sfr      AUXINTIF     = 0xef;
#define   T3IF         0x02

sbit     P10          = P1^0;

void TM3_Isr() interrupt 19
{
    P10 = !P10;           //测试端口
    AUXINTIF &= ~T3IF;    //清中断标志
}

void main()
{
    T3L = 0x66;          //65536-11.0592M/12/1000
    T3H = 0xfc;
    T4T3M = 0x08;        //启动定时器
    IE2 = ET3;           //使能定时器中断
    EA = 1;

    while (1);
}
```

11.5.12 定时器 4 中断

汇编代码

;测试工作频率为11.0592MHz

```
T4L      DATA      0D3H
T4H      DATA      0D2H
T4T3M   DATA      0DIH
IE2      DATA      0AFH
ET4      EQU       40H
AUXINTIF DATA      0EFH
T4IF    EQU       04H

ORG      0000H
LJMP    MAIN
ORG      00A3H
LJMP    TM4ISR

ORG      0100H
```

TM4ISR:

<i>CPL</i>	<i>P1.0</i>	; 测试端口
<i>ANL</i>	<i>AUXINTIF,#NOT T4IF</i>	; 清中断标志
<i>RETI</i>		

MAIN:

<i>MOV</i>	<i>SP,#3FH</i>	
<i>MOV</i>	<i>T4L,#66H</i>	; 65536-11.0592M/12/1000
<i>MOV</i>	<i>T4H,#0FC</i>	
<i>MOV</i>	<i>T4T3M,#80H</i>	; 启动定时器
<i>MOV</i>	<i>IE2,#ET4</i>	; 使能定时器中断
<i>SETB</i>	<i>EA</i>	
<i>JMP</i>	\$	

END**C 语言代码**

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

// 测试工作频率为 11.0592MHz

sfr T4L = 0xd3;
sfr T4H = 0xd2;
sfr T4T3M = 0xd1;
sfr IE2 = 0xaf;
#define ET4 0x40
sfr AUXINTIF = 0xef;
#define T4IF 0x04

sbit P10 = P1^0;

void TM4_Isr() interrupt 20
{
    P10 = !P10; // 测试端口
    AUXINTIF &= ~T4IF; // 清中断标志
}

void main()
{
    T4L = 0x66; // 65536-11.0592M/12/1000
    T4H = 0xfc;
    T4T3M = 0x80; // 启动定时器
    IE2 = ET4; // 使能定时器中断
    EA = 1;

    while (1);
}
```

11.5.13 UART1 中断**汇编代码**

; 测试工作频率为 11.0592MHz

T2L DATA 0D7H

T2H	DATA	0D6H	
AUXR	DATA	8EH	
	ORG	0000H	
	LJMP	MAIN	
	ORG	0023H	
	LJMP	UARTIISR	
	ORG	0100H	
UARTIISR:	JNB	TI,CHECKRI	
	CLR	TI	;清中断标志
	CPL	P1.0	;测试端口
CHECKRI:	JNB	RI,ISREXIT	
	CLR	RI	;清中断标志
	CPL	P1.1	;测试端口
ISREXIT:	RETI		
MAIN:	MOV	SP,#3FH	
	MOV	SCON,#50H	
	MOV	T2L,#0E8H	;65536-11059200/115200/4=0FFE8H
	MOV	T2H,#0FFH	
	MOV	AUXR,#15H	;启动定时器
	SETB	ES	;使能串口中断
	SETB	EA	
	MOV	SBUF,#5AH	;发送测试数据
	JMP	\$	
 END			

C 语言代码

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

//测试工作频率为11.0592MHz

sfr T2L = 0xd7;
sfr T2H = 0xd6;
sfr AUXR = 0x8e;

sbit P10 = P1^0;
sbit P11 = P1^1;

void UART1_Isr() interrupt 4
{
    if(TI)
    {
        TI = 0; //清中断标志
        P10 = !P10; //测试端口
    }
    if(RI)
    {
        RI = 0; //清中断标志
    }
}
```

```

    P1I = !P1I;
    }
}

void main()
{
    SCON = 0x50;                                //测试端口
    T2L = 0xe8;                                 //65536-11059200/115200/4=0FFE8H
    T2H = 0xff;
    AUXR = 0x15;                                //启动定时器
    ES = 1;                                     //使能串口中断
    EA = 1;
    SBUF = 0x5a;                                //发送测试数据

    while (1);
}

```

11.5.14 UART2 中断

汇编代码

; 测试工作频率为 11.0592MHz

<i>T2L</i>	<i>DATA</i>	<i>0D7H</i>
<i>T2H</i>	<i>DATA</i>	<i>0D6H</i>
<i>AUXR</i>	<i>DATA</i>	<i>8EH</i>
<i>S2CON</i>	<i>DATA</i>	<i>9AH</i>
<i>S2BUF</i>	<i>DATA</i>	<i>9BH</i>
<i>IE2</i>	<i>DATA</i>	<i>0AFH</i>
<i>ES2</i>	<i>EQU</i>	<i>01H</i>
<i>ORG</i>	<i>0000H</i>	
<i>LJMP</i>	<i>MAIN</i>	
<i>ORG</i>	<i>0043H</i>	
<i>LJMP</i>	<i>UART2ISR</i>	
<i>ORG</i>	<i>0100H</i>	
UART2ISR:		
<i>PUSH</i>	<i>ACC</i>	
<i>PUSH</i>	<i>PSW</i>	
<i>MOV</i>	<i>A,S2CON</i>	
<i>JNB</i>	<i>ACC.1,CHECKRI</i>	
<i>ANL</i>	<i>S2CON,#NOT 02H</i> ;清中断标志	
<i>CPL</i>	<i>P1.2</i> ;测试端口	
CHECKRI:		
<i>MOV</i>	<i>A,S2CON</i>	
<i>JNB</i>	<i>ACC.0,ISREXIT</i>	
<i>ANL</i>	<i>S2CON,#NOT 01H</i> ;清中断标志	
<i>CPL</i>	<i>P1.3</i> ;测试端口	
ISREXIT:		
<i>POP</i>	<i>PSW</i>	
<i>POP</i>	<i>ACC</i>	
<i>RETI</i>		
MAIN:		
<i>MOV</i>	<i>SP,#3FH</i>	
<i>MOV</i>	<i>S2CON,#10H</i>	
<i>MOV</i>	<i>T2L,#0E8H</i> ;65536-11059200/115200/4=0FFE8H	

```

MOV      T2H,#0FFH
MOV      AUXR,#14H          ;启动定时器
MOV      IE2,#ES2           ;使能串口中断
SETB    EA
MOV      S2BUF,#5AH         ;发送测试数据

JMP      $

```

END**C 语言代码**

```

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

//测试工作频率为11.0592MHz

sfr      T2L      = 0xd7;
sfr      T2H      = 0xd6;
sfr      AUXR     = 0x8e;
sfr      S2CON    = 0x9a;
sfr      S2BUF    = 0x9b;
sfr      IE2      = 0xaf;
#define  ES2      0x01

sbit     P12      = P1^2;
sbit     P13      = P1^3;

void UART2_Isr() interrupt 8
{
    if(S2CON & 0x02)
    {
        S2CON &= ~0x02;          //清中断标志
        P12 = !P12;              //测试端口
    }
    if(S2CON & 0x01)
    {
        S2CON &= ~0x01;          //清中断标志
        P13 = !P13;              //测试端口
    }
}

void main()
{
    S2CON = 0x10;
    T2L = 0xe8;                //65536-11059200/115200/4=0FFE8H
    T2H = 0xff;
    AUXR = 0x14;                //启动定时器
    IE2 = ES2;                  //使能串口中断
    EA = 1;                     //发送测试数据

    while (1);
}

```

11.5.15 UART3 中断

汇编代码

//测试工作频率为 11.0592MHz

<i>T2L</i>	<i>DATA</i>	<i>0D7H</i>	
<i>T2H</i>	<i>DATA</i>	<i>0D6H</i>	
<i>AUXR</i>	<i>DATA</i>	<i>8EH</i>	
<i>S3CON</i>	<i>DATA</i>	<i>0ACh</i>	
<i>S3BUF</i>	<i>DATA</i>	<i>0ADH</i>	
<i>IE2</i>	<i>DATA</i>	<i>0AFH</i>	
<i>ES3</i>	<i>EQU</i>	<i>08H</i>	
	<i>ORG</i>	<i>0000H</i>	
	<i>LJMP</i>	<i>MAIN</i>	
	<i>ORG</i>	<i>008BH</i>	
	<i>LJMP</i>	<i>UART3ISR</i>	
	<i>ORG</i>	<i>0100H</i>	
<i>UART3ISR:</i>	<i>PUSH</i>	<i>ACC</i>	
	<i>PUSH</i>	<i>PSW</i>	
	<i>MOV</i>	<i>A,S3CON</i>	
	<i>JNB</i>	<i>ACC.1,CHECKRI</i>	
	<i>ANL</i>	<i>S3CON,#NOT 02H</i>	;清中断标志
	<i>CPL</i>	<i>P1.0</i>	;测试端口
<i>CHECKRI:</i>	<i>MOV</i>	<i>A,S3CON</i>	
	<i>JNB</i>	<i>ACC.0,ISREXIT</i>	
	<i>ANL</i>	<i>S3CON,#NOT 01H</i>	;清中断标志
	<i>CPL</i>	<i>P1.1</i>	;测试端口
<i>ISREXIT:</i>	<i>POP</i>	<i>PSW</i>	
	<i>POP</i>	<i>ACC</i>	
	<i>RETI</i>		
<i>MAIN:</i>	<i>MOV</i>	<i>SP,#3FH</i>	
	<i>MOV</i>	<i>S3CON,#10H</i>	
	<i>MOV</i>	<i>T2L,#0E8H</i>	;65536-11059200/115200/4=0FFE8H
	<i>MOV</i>	<i>T2H,#0FFH</i>	
	<i>MOV</i>	<i>AUXR,#14H</i>	;启动定时器
	<i>MOV</i>	<i>IE2,#ES3</i>	;使能串口中断
	<i>SETB</i>	<i>EA</i>	
	<i>MOV</i>	<i>S3BUF,#5AH</i>	;发送测试数据
	<i>JMP</i>	\$	
	<i>END</i>		

C 语言代码

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

//测试工作频率为 11.0592MHz

```

sfr      T2L      =  0xd7;
sfr      T2H      =  0xd6;
sfr      AUXR     =  0x8e;
sfr      S3CON     =  0xac;
sfr      S3BUF     =  0xad;
sfr      IE2       =  0xaf;
#define  ES3        0x08

sbit     P10      =  P1^0;
sbit     P1I      =  P1^1;

void UART3_Isr() interrupt 17
{
    if(S3CON & 0x02)
    {
        S3CON &= ~0x02;           //清中断标志
        P10 = !P10;              //测试端口
    }
    if(S3CON & 0x01)
    {
        S3CON &= ~0x01;           //清中断标志
        P1I = !P1I;              //测试端口
    }
}

void main()
{
    S3CON = 0x10;             //65536-11059200/115200/4=0FFE8H
    T2L = 0xe8;
    T2H = 0xff;
    AUXR = 0x14;              //启动定时器
    IE2 = ES3;                //使能串口中断
    EA = 1;
    S3BUF = 0x5a;              //发送测试数据

    while (1);
}

```

11.5.16 UART4 中断

汇编代码

; 测试工作频率为 11.0592MHz

```

T2L      DATA      0D7H
T2H      DATA      0D6H
AUXR     DATA      8EH
S4CON    DATA      084H
S4BUF    DATA      085H
IE2      DATA      0AFH
ES4      EQU       10H

                    ORG      0000H
                    LJMP   MAIN
                    ORG      0093H
                    LJMP   UART4ISR

                    ORG      0100H

UART4ISR:

```

<i>PUSH</i>	<i>ACC</i>
<i>PUSH</i>	<i>PSW</i>
<i>MOV</i>	<i>A,S4CON</i>
<i>JNB</i>	<i>ACC.1,CHECKRI</i>
<i>ANL</i>	<i>S4CON,#NOT 02H</i>
	;清中断标志
<i>CPL</i>	<i>P1.0</i>
	;测试端口
CHECKRI:	
<i>MOV</i>	<i>A,S4CON</i>
<i>JNB</i>	<i>ACC.0,ISREXIT</i>
<i>ANL</i>	<i>S4CON,#NOT 01H</i>
	;清中断标志
<i>CPL</i>	<i>P1.1</i>
	;测试端口
ISREXIT:	
<i>POP</i>	<i>PSW</i>
<i>POP</i>	<i>ACC</i>
<i>RETI</i>	
MAIN:	
<i>MOV</i>	<i>SP,#3FH</i>
<i>MOV</i>	<i>S4CON,#10H</i>
<i>MOV</i>	<i>T2L,#0E8H</i>
	;65536-11059200/115200/4=0FFE8H
<i>MOV</i>	<i>T2H,#0FFH</i>
<i>MOV</i>	<i>AUXR,#14H</i>
	;启动定时器
<i>MOV</i>	<i>IE2,#ES4</i>
	;使能串口中断
<i>SETB</i>	<i>EA</i>
<i>MOV</i>	<i>S4BUF,#5AH</i>
	;发送测试数据
<i>JMP</i>	\$
END	

C 语言代码

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

//测试工作频率为11.0592MHz

sfr T2L = 0xd7;
sfr T2H = 0xd6;
sfr AUXR = 0x8e;
sfr S4CON = 0x84;
sfr S4BUF = 0x85;
sfr IE2 = 0xaf;
#define ES4 0x10

sbit P10 = P1^0;
sbit P11 = P1^1;

void UART4_Isr() interrupt 18
{
    if(S4CON & 0x02)
    {
        S4CON &= ~0x02;          //清中断标志
        P10 = !P10;              //测试端口
    }
    if(S4CON & 0x01)
    {
        S4CON &= ~0x01;          //清中断标志
    }
}
```

```

    P1I = !P1I;           //测试端口
}
}

void main()
{
    S4CON = 0x10;
    T2L = 0xe8;          //65536-11059200/115200/4=0FFE8H
    T2H = 0xff;
    AUXR = 0x14;         //启动定时器
    IE2 = ES4;           //使能串口中断
    EA = 1;
    S4BUF = 0x5a;        //发送测试数据

    while (1);
}

```

11.5.17 ADC中断

汇编代码

<i>ADC_CONTR</i>	<i>DATA</i>	<i>0BCH</i>
<i>ADC_RES</i>	<i>DATA</i>	<i>0BDH</i>
<i>ADC_RESL</i>	<i>DATA</i>	<i>0BEH</i>
<i>ADCCFG</i>	<i>DATA</i>	<i>0DEH</i>
<i>EADC</i>	<i>BIT</i>	<i>IE.5</i>
<i>ORG</i>	<i>0000H</i>	
<i>LJMP</i>	<i>MAIN</i>	
<i>ORG</i>	<i>002BH</i>	
<i>LJMP</i>	<i>ADCISR</i>	
<i>ORG</i>	<i>0100H</i>	
<i>ADCISR:</i>	<i>ANL</i>	<i>ADC_CONTR,#NOT 20H</i> ;清中断标志
	<i>MOV</i>	<i>P0,ADC_RES</i> ;测试端口
	<i>MOV</i>	<i>P2,ADC_RESL</i> ;测试端口
	<i>RETI</i>	
<i>MAIN:</i>	<i>MOV</i>	<i>SP,#3FH</i>
	<i>MOV</i>	<i>ADCCFG,#00H</i>
	<i>MOV</i>	<i>ADC_CONTR,#0C0H</i> ;使能并启动ADC 模块
	<i>SETB</i>	<i>EADC</i> ;使能ADC 中断
	<i>SETB</i>	<i>EA</i>
	<i>JMP</i>	\$
<i>END</i>		

C 语言代码

```

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

sfr      ADC_CONTR  =  0xbc;
sfr      ADC_RES    =  0xbd;

```

```

sfr      ADC_RESL    = 0xbe;
sfr      ADCCFG     = 0xde;
sbit     EADC        = IE^5;

void ADC_Isr() interrupt 5
{
    ADC_CONTR &= ~0x20;           //清中断标志
    P0 = ADC_RES;                //测试端口
    P2 = ADC_RESL;               //测试端口
}

void main()
{
    ADCCFG = 0x00;              //使能并启动ADC 模块
    ADC_CONTR = 0xc0;            //使能ADC 中断
    EA = 1;                     //EA = 1;

    while (1);
}

```

11.5.18 LVD中断

汇编代码

RSTCFG	DATA	0FFH	
ENLVR	EQU	40H	;RSTCFG.6
LVD2V2	EQU	00H	;LVD@2.2V
LVD2V4	EQU	01H	;LVD@2.4V
LVD2V7	EQU	02H	;LVD@2.7V
LVD3V0	EQU	03H	;LVD@3.0V
ELVD	BIT	IE.6	
LVDF	EQU	20H	;PCON.5
ORG	0000H		
LJMP	MAIN		
ORG	0033H		
LJMP	LVDISR		
ORG	0100H		
LVDISR:	ANL	PCON,#NOT LVDF	;清中断标志
	CPL	P1.0	;测试端口
	RETI		
MAIN:	MOV	SP,#3FH	
	ANL	PCON,#NOT LVDF	;上电需要清中断标志
	MOV	RSTCFG# LVD3V0	;设置LVD 电压为3.0V
	SETB	ELVD	;使能LVD 中断
	SETB	EA	
	JMP	\$	
END			

C 语言代码

```

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

sfr RSTCFG = 0xff;
#define ENLVR 0x40 //RSTCFG6
#define LVD2V2 0x00 //LVD@2.2V
#define LVD2V4 0x01 //LVD@2.4V
#define LVD2V7 0x02 //LVD@2.7V
#define LVD3V0 0x03 //LVD@3.0V
sbit ELVD = IE^6;
#define LVDF 0x20 //PCON.5
sbit P10 = P1^0;

void LVD_Isr() interrupt 6
{
    PCON &= ~LVDF;           //清中断标志
    P10 = !P10;              //测试端口
}

void main()
{
    PCON &= ~LVDF;          //上电需要清中断标志
    RSTCFG = LVD3V0;         //设置LVD 电压为3.0V
    ELVD = 1;                //使能LVD 中断
    EA = 1;

    while (1);
}

```

11.5.19 PCA中断

汇编代码

;测试工作频率为 11.0592MHz

CCON	DATA	0D8H
CF	BIT	CCON.7
CR	BIT	CCON.6
CCF3	BIT	CCON.3
CCF2	BIT	CCON.2
CCF1	BIT	CCON.1
CCF0	BIT	CCON.0
CMOD	DATA	0D9H
CL	DATA	0E9H
CH	DATA	0F9H
CCAPM0	DATA	0DAH
CCAP0L	DATA	0EAH
CCAP0H	DATA	0FAH
PCA_PWM0	DATA	0F2H
CCAPM1	DATA	0DBH
CCAPIL	DATA	0EBH
CCAPIH	DATA	0FBH
PCA_PWM1	DATA	0F3H
CCAPM2	DATA	0DCBH
CCAP2L	DATA	0ECH
CCAP2H	DATA	0FCH
PCA_PWM2	DATA	0F4H
CCAPM3	DATA	0DDH
CCAP3L	DATA	0EDH

CCAP3H	DATA	0FDH	
PCA_PWM3	DATA	0F5H	
	ORG	0000H	
	LJMP	MAIN	
	ORG	003BH	
	LJMP	PCAIISR	
	ORG	0100H	
PCAIISR:	JNB	CF,ISREXIT	
	CLR	CF	;清中断标志
	CPL	P1.0	;测试端口
ISREXIT:	RETI		
MAIN:	MOV	SP,#3FH	
	MOV	CCON,#00H	
	MOV	CMOD,#09H	;PCA 时钟为系统时钟,使能PCA 计时中断
	SETB	CR	;启动PCA 计时器
	SETB	EA	
	JMP	\$	
 END			

C 语言代码

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

```
//测试工作频率为11.0592MHz
```

```
sfr CCON = 0xd8;
sbit CF = CCON^7;
sbit CR = CCON^6;
sbit CCF3 = CCON^3;
sbit CCF2 = CCON^2;
sbit CCF1 = CCON^1;
sbit CCF0 = CCON^0;
sfr CMOD = 0xd9;
sfr CL = 0xe9;
sfr CH = 0xf9;
sfr CCAPM0 = 0xda;
sfr CCAP0L = 0xea;
sfr CCAP0H = 0xfa;
sfr PCA_PWM0 = 0xf2;
sfr CCAPM1 = 0xdb;
sfr CCAP1L = 0xeb;
sfr CCAPIH = 0xfb;
sfr PCA_PWM1 = 0xf3;
sfr CCAPM2 = 0xdc;
sfr CCAP2L = 0xec;
sfr CCAP2H = 0xfc;
sfr PCA_PWM2 = 0xf4;
sfr CCAPM3 = 0xdd;
sfr CCAP3L = 0xed;
```

```

sfr CCAP3H = 0xfd;
sfr PCA_PWM3 = 0xf5;

sbit P10 = P1^0;

void PCA_Isr() interrupt 7
{
    if (CF)
    {
        CF = 0; //清中断标志
        P10 = !P10; //测试端口
    }
}

void main()
{
    CCON = 0x00;
    CMOD = 0x09; //PCA 时钟为系统时钟,使能PCA 计时中断
    CR = 1; //启动PCA 计时器
    EA = 1;

    while (1);
}

```

11.5.20 SPI中断

汇编代码

SPSTAT	DATA	0CDH	
SPCTL	DATA	0CEH	
SPDAT	DATA	0CFH	
IE2	DATA	0AFH	
ESPI	EQU	02H	
	ORG	0000H	
	LJMP	MAIN	
	ORG	004BH	
	LJMP	SPIISR	
	ORG	0100H	
SPIISR:	MOV	SPSTAT,#0C0H	;清中断标志
	CPL	P1.0	;测试端口
	RETI		
MAIN:	MOV	SP,#3FH	
	MOV	SPCTL,#50H	;使能SPI 主机模式
	MOV	SPSTAT,#0C0H	;清中断标志
	MOV	IE2,#ESPI	;使能SPI 中断
	SETB	EA	
	MOV	SPDAT,#5AH	;发送测试数据
	JMP	\$	
 END			

C 语言代码

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

sfr SPSTAT      = 0xcd;
sfr SPCTL       = 0xce;
sfr SPDAT       = 0xcf;
sfr IE2          = 0xaf;
#define ESPI        0x02

sbit P10         = P1^0;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;           //清中断标志
    P10 = !P10;              //测试端口
}

void main()
{
    SPCTL = 0x50;           //使能SPI 主机模式
    SPSTAT = 0xc0;           //清中断标志
    IE2 = ESPI;              //使能SPI 中断
    EA = 1;                  //EA = 1;
    SPDAT = 0x5a;            //发送测试数据

    while (1);
}
```

11.5.21 CMP中断

汇编代码

CMPCR1	DATA	0E6H
CMPCR2	DATA	0E7H
	ORG	0000H
	LJMP	MAIN
	ORG	00ABH
	LJMP	CMPISR
	ORG	0100H
CMPISR:	ANL	CMPCR1,#NOT 40H ;清中断标志
	CPL	P1.0 ;测试端口
	RETI	
MAIN:	MOV	SP,#3FH
	MOV	CMPCR2,#00H
	MOV	CMPCR1,#80H ;使能比较器模块
	ORL	CMPCR1,#30H ;使能比较器边沿中断
	ANL	CMPCR1,#NOT 08H ;P3.6 为 CMP+ 输入脚
	ORL	CMPCR1,#04H ;P3.7 为 CMP- 输入脚
	ORL	CMPCR1,#02H ;使能比较器输出
	SETB	EA

JMP \$**END**

C 语言代码

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

sfr      CMPCR1      = 0xe6;
sfr      CMPCR2      = 0xe7;

sbit     P10          = P1^0;

void CMP_Isr() interrupt 21
{
    CMPCR1 &= ~0x40;           //清中断标志
    P10 = !P10;                //测试端口
}

void main()
{
    CMPCR2 = 0x00;            //使能比较器模块
    CMPCR1 = 0x80;            //使能比较器边沿中断
    CMPCR1 |= 0x30;           //P3.6 为 CMP+ 输入脚
    CMPCR1 &= ~0x08;          //P3.7 为 CMP- 输入脚
    CMPCR1 |= 0x04;           //使能比较器输出
    CMPCR1 |= 0x02;
    EA = 1;

    while (1);
}
```

11.5.22 PWM中断

汇编代码

; 测试工作频率为 11.0592MHz

P_SW2	DATA	0BAH
PWMCFG	DATA	0F1H
PWMIF	DATA	0F6H
PWMFDCR	DATA	0F7H
PWMCR	DATA	0FEH
PWMCH	XDATA	0FFF0H
PWMCL	XDATA	0FFF1H
PWMCKS	XDATA	0FFF2H
TADCPH	XDATA	0FFF3H
TADCPL	XDATA	0FFF4H
PWM0T1H	XDATA	0FF00H
PWM0T1L	XDATA	0FF01H
PWM0T2H	XDATA	0FF02H
PWM0T2L	XDATA	0FF03H
PWM0CR	XDATA	0FF04H
PWM0HLD	XDATA	0FF05H
PWM1T1H	XDATA	0FF10H
PWM1T1L	XDATA	0FF11H

<i>PWM1T2H</i>	<i>XDATA</i>	<i>0FF12H</i>
<i>PWM1T2L</i>	<i>XDATA</i>	<i>0FF13H</i>
<i>PWM1CR</i>	<i>XDATA</i>	<i>0FF14H</i>
<i>PWM1HLD</i>	<i>XDATA</i>	<i>0FF15H</i>
<i>PWM2T1H</i>	<i>XDATA</i>	<i>0FF20H</i>
<i>PWM2T1L</i>	<i>XDATA</i>	<i>0FF21H</i>
<i>PWM2T2H</i>	<i>XDATA</i>	<i>0FF22H</i>
<i>PWM2T2L</i>	<i>XDATA</i>	<i>0FF23H</i>
<i>PWM2CR</i>	<i>XDATA</i>	<i>0FF24H</i>
<i>PWM2HLD</i>	<i>XDATA</i>	<i>0FF25H</i>
<i>PWM3T1H</i>	<i>XDATA</i>	<i>0FF30H</i>
<i>PWM3T1L</i>	<i>XDATA</i>	<i>0FF31H</i>
<i>PWM3T2H</i>	<i>XDATA</i>	<i>0FF32H</i>
<i>PWM3T2L</i>	<i>XDATA</i>	<i>0FF33H</i>
<i>PWM3CR</i>	<i>XDATA</i>	<i>0FF34H</i>
<i>PWM3HLD</i>	<i>XDATA</i>	<i>0FF35H</i>
<i>PWM4T1H</i>	<i>XDATA</i>	<i>0FF40H</i>
<i>PWM4T1L</i>	<i>XDATA</i>	<i>0FF41H</i>
<i>PWM4T2H</i>	<i>XDATA</i>	<i>0FF42H</i>
<i>PWM4T2L</i>	<i>XDATA</i>	<i>0FF43H</i>
<i>PWM4CR</i>	<i>XDATA</i>	<i>0FF44H</i>
<i>PWM4HLD</i>	<i>XDATA</i>	<i>0FF45H</i>
<i>PWM5T1H</i>	<i>XDATA</i>	<i>0FF50H</i>
<i>PWM5T1L</i>	<i>XDATA</i>	<i>0FF51H</i>
<i>PWM5T2H</i>	<i>XDATA</i>	<i>0FF52H</i>
<i>PWM5T2L</i>	<i>XDATA</i>	<i>0FF53H</i>
<i>PWM5CR</i>	<i>XDATA</i>	<i>0FF54H</i>
<i>PWM5HLD</i>	<i>XDATA</i>	<i>0FF55H</i>
<i>PWM6T1H</i>	<i>XDATA</i>	<i>0FF60H</i>
<i>PWM6T1L</i>	<i>XDATA</i>	<i>0FF61H</i>
<i>PWM6T2H</i>	<i>XDATA</i>	<i>0FF62H</i>
<i>PWM6T2L</i>	<i>XDATA</i>	<i>0FF63H</i>
<i>PWM6CR</i>	<i>XDATA</i>	<i>0FF64H</i>
<i>PWM6HLD</i>	<i>XDATA</i>	<i>0FF65H</i>
<i>PWM7T1H</i>	<i>XDATA</i>	<i>0FF70H</i>
<i>PWM7T1L</i>	<i>XDATA</i>	<i>0FF71H</i>
<i>PWM7T2H</i>	<i>XDATA</i>	<i>0FF72H</i>
<i>PWM7T2L</i>	<i>XDATA</i>	<i>0FF73H</i>
<i>PWM7CR</i>	<i>XDATA</i>	<i>0FF74H</i>
<i>PWM7HLD</i>	<i>XDATA</i>	<i>0FF75H</i>
	<i>ORG</i>	<i>0000H</i>
	<i>LJMP</i>	<i>MAIN</i>
	<i>ORG</i>	<i>00B3H</i>
	<i>LJMP</i>	<i>PWMISR</i>
	<i>ORG</i>	<i>00BBH</i>
	<i>LJMP</i>	<i>PWMFDISR</i>
	<i>ORG</i>	<i>0100H</i>
<i>PWMISR:</i>		
	<i>PUSH</i>	<i>ACC</i>
	<i>PUSH</i>	<i>PSW</i>
	<i>MOV</i>	<i>A,PWMCFG</i>
	<i>JNB</i>	<i>ACC.7,ISREXIT</i>
	<i>ANL</i>	<i>PWMCFG,#NOT 80H</i>
		;清中断标志
	<i>CPL</i>	<i>P1.0</i>
		;测试端口
<i>ISREXIT:</i>		
	<i>POP</i>	<i>PSW</i>
	<i>POP</i>	<i>ACC</i>

RETI**PWMFDISR:**

ANL	PWMFDCR,#NOT 01H	;清中断标志
CPL	P1.1	;测试端口
RETI		

MAIN:

MOV	SP,#3FH	
MOV	P_SW2,#80H	
MOV	A,#0FH	
MOV	DPTR,#PWMCKS	
MOVX	@DPTR,A	;PWM 时钟为系统时钟/16
MOV	A,#01H	
MOV	DPTR,#PWMCH	;设置 PWM 周期为 256 个 PWM 时钟
MOVX	@DPTR,A	
MOV	A,#00H	
MOV	DPTR,#PWMCL	
MOVX	@DPTR,A	
MOV	P_SW2,#00H	
MOV	PWMFDCR,#7AH	;使能 IO 口异常检测中断
MOV	PWMCR,#0C0H	;启动 PWM 模块并使能 PWM 计数器中断
SETB	EA	
JMP	\$	

END**C 语言代码**

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

//测试工作频率为 11.0592MHz

sfr P_SW2 = 0xba;
sfr PWMCFG = 0xf1;
sfr PWMIF = 0xf6;
sfr PWMFDCR = 0xf7;
sfr PWMCR = 0xfe;

#define PWMC (*(unsigned int volatile xdata *)0xffff0)
#define PWMCKS (*(unsigned char volatile xdata *)0xffff2)
#define TADCP (*(unsigned int volatile xdata *)0xffff3)
#define PWM0T1 (*(unsigned int volatile xdata *)0xff00)
#define PWM0T2 (*(unsigned int volatile xdata *)0xff02)
#define PWM0CR (*(unsigned char volatile xdata *)0xff04)
#define PWM0HLD (*(unsigned char volatile xdata *)0xff05)
#define PWM1T1 (*(unsigned int volatile xdata *)0xff10)
#define PWM1T2 (*(unsigned int volatile xdata *)0xff12)
#define PWM1CR (*(unsigned char volatile xdata *)0xff14)
#define PWM1HLD (*(unsigned char volatile xdata *)0xff15)
#define PWM2T1 (*(unsigned int volatile xdata *)0xff20)
#define PWM2T2 (*(unsigned int volatile xdata *)0xff22)
#define PWM2CR (*(unsigned char volatile xdata *)0xff24)
#define PWM2HLD (*(unsigned char volatile xdata *)0xff25)
#define PWM3T1 (*(unsigned int volatile xdata *)0xff30)
```

```

#define PWM3T2      (*(unsigned int volatile xdata *)0xff32)
#define PWM3CR      (*(unsigned char volatile xdata *)0xff34)
#define PWM3HLD     (*(unsigned char volatile xdata *)0xff35)
#define PWM4T1      (*(unsigned int volatile xdata *)0xff40)
#define PWM4T2      (*(unsigned int volatile xdata *)0xff42)
#define PWM4CR      (*(unsigned char volatile xdata *)0xff44)
#define PWM4HLD     (*(unsigned char volatile xdata *)0xff45)
#define PWM5T1      (*(unsigned int volatile xdata *)0xff50)
#define PWM5T2      (*(unsigned int volatile xdata *)0xff52)
#define PWM5CR      (*(unsigned char volatile xdata *)0xff54)
#define PWM5HLD     (*(unsigned char volatile xdata *)0xff55)
#define PWM6T1      (*(unsigned int volatile xdata *)0xff60)
#define PWM6T2      (*(unsigned int volatile xdata *)0xff62)
#define PWM6CR      (*(unsigned char volatile xdata *)0xff64)
#define PWM6HLD     (*(unsigned char volatile xdata *)0xff65)
#define PWM7T1      (*(unsigned int volatile xdata *)0xff70)
#define PWM7T2      (*(unsigned int volatile xdata *)0xff72)
#define PWM7CR      (*(unsigned char volatile xdata *)0xff74)
#define PWM7HLD     (*(unsigned char volatile xdata *)0xff75)

sbit P10      = P1^0;
sbit P11      = P1^1;

void PWM_Isr() interrupt 22
{
    if (PWMCIFG & 0x80)
    {
        PWMCIFG &= ~0x80;          //清中断标志
        P10 = !P10;                //测试端口
    }
}

void PWMFD_Isr() interrupt 23
{
    PWMFDCR &= ~0x01;          //清中断标志
    P11 = !P11;                //测试端口
}

void main()
{
    P_SW2 = 0x80;              // PWM 时钟为系统时钟/16
    PWMCKS = 0x0f;             // 设置 PWM 周期为 256 个 PWM 时钟
    PWMC = 0x0100;
    P_SW2 = 0x00;

    PWMFDCR = 0x7a;           //能IO 口异常检测中断
    PWMCR = 0xc0;              //启动 PWM 模块并使能 PWM 计数器中断
    EA = 1;

    while (1);
}

```

11.5.23 I2C中断

汇编代码

P_SW2	DATA	0BAH
I2CCFG	XDATA	0FE80H

<i>I2CMSCR</i>	<i>XDATA</i>	<i>0FE81H</i>
<i>I2CMSST</i>	<i>XDATA</i>	<i>0FE82H</i>
<i>I2CSLCR</i>	<i>XDATA</i>	<i>0FE83H</i>
<i>I2CSLST</i>	<i>XDATA</i>	<i>0FE84H</i>
<i>I2CSLADR</i>	<i>XDATA</i>	<i>0FE85H</i>
<i>I2CTXD</i>	<i>XDATA</i>	<i>0FE86H</i>
<i>I2CRXD</i>	<i>XDATA</i>	<i>0FE87H</i>
	<i>ORG</i>	<i>0000H</i>
	<i>LJMP</i>	<i>MAIN</i>
	<i>ORG</i>	<i>00C3H</i>
	<i>LJMP</i>	<i>I2CISR</i>
	<i>ORG</i>	<i>0100H</i>
<i>I2CISR:</i>		
	<i>PUSH</i>	<i>ACC</i>
	<i>PUSH</i>	<i>DPL</i>
	<i>PUSH</i>	<i>DPH</i>
	<i>PUSH</i>	<i>P_SW2</i>
	<i>MOV</i>	<i>P_SW2,#80H</i>
	<i>MOV</i>	<i>DPTR,#I2CMSST</i>
	<i>MOVX</i>	<i>A,@DPTR</i>
	<i>ANL</i>	<i>A,#NOT 40H</i> ;清中断标志
	<i>MOVX</i>	<i>@DPTR,A</i>
	<i>CPL</i>	<i>P1.0</i> ;测试端口
	<i>POP</i>	<i>P_SW2</i>
	<i>POP</i>	<i>DPH</i>
	<i>POP</i>	<i>DPL</i>
	<i>POP</i>	<i>ACC</i>
	<i>RETI</i>	
<i>MAIN:</i>		
	<i>MOV</i>	<i>SP,#3FH</i>
	<i>MOV</i>	<i>P_SW2,#80H</i>
	<i>MOV</i>	<i>A,#0C0H</i> ;使能 I2C 主机模式
	<i>MOV</i>	<i>DPTR,#I2CCFG</i>
	<i>MOVX</i>	<i>@DPTR,A</i>
	<i>MOV</i>	<i>A,#80H</i> ;使能 I2C 中断
	<i>MOV</i>	<i>DPTR,#I2CMSCR</i>
	<i>MOVX</i>	<i>@DPTR,A</i>
	<i>MOV</i>	<i>P_SW2,#00H</i>
	<i>SETB</i>	<i>EA</i>
	<i>MOV</i>	<i>P_SW2,#80H</i>
	<i>MOV</i>	<i>A,#081H</i> ;发送起始命令
	<i>MOV</i>	<i>DPTR,#I2CMSCR</i>
	<i>MOVX</i>	<i>@DPTR,A</i>
	<i>MOV</i>	<i>P_SW2,#00H</i>
	<i>JMP</i>	\$
	<i>END</i>	

C 语言代码

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

```
sfr      P_SW2      = 0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR     (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST      (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR      (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST      (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR     (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD       (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD       (*(unsigned char volatile xdata *)0xfe87)

sbit     P10        = P1^0;

void I2C_Isr() interrupt 24
{
    _push_(P_SW2);
    P_SW2 |= 0x80;
    if (I2CMSST & 0x40)
    {
        I2CMSST &= ~0x40;           //清中断标志
        P10 = !P10;                //测试端口
    }
    _pop_(P_SW2);
}

void main()
{
    P_SW2 = 0x80;
    I2CCFG = 0xc0;               //使能I2C 主机模式
    I2CMSCR = 0x80;              //使能I2C 中断;
    P_SW2 = 0x00;
    EA = 1;

    P_SW2 = 0x80;
    I2CMSCR = 0x81;              //发送起始命令
    P_SW2 = 0x00;

    while (1);
}
```

12 定时器/计数器

STC8 系列单片机内部设置了 5 个 16 位定时器/计数器。5 个 16 位定时器 T0、T1、T2、T3 和 T4 都具有计数方式和定时方式两种工作方式。对定时器/计数器 T0 和 T1，用它们在特殊功能寄存器 TMOD 中相对应的控制位 C/T 来选择 T0 或 T1 为定时器还是计数器。对定时器/计数器 T2，用特殊功能寄存器 AUXR 中的控制位 T2_C/T 来选择 T2 为定时器还是计数器。对定时器/计数器 T3，用特殊功能寄存器 T4T3M 中的控制位 T3_C/T 来选择 T3 为定时器还是计数器。对定时器/计数器 T4，用特殊功能寄存器 T4T3M 中的控制位 T4_C/T 来选择 T4 为定时器还是计数器。定时器/计数器的核心部件是一个加法计数器，其本质是对脉冲进行计数。只是计数脉冲来源不同：如果计数脉冲来自系统时钟，则为定时方式，此时定时器/计数器每 12 个时钟或者每 1 个时钟得到一个计数脉冲，计数值加 1；如果计数脉冲来自单片机外部引脚（T0 为 P3.4，T1 为 P3.5，T2 为 P1.2，T3 为 P0.4，T4 为 P0.6），则为计数方式，每来一个脉冲加 1。

当定时器/计数器 T0、T1 及 T2 工作在定时模式时，特殊功能寄存器 AUXR 中的 T0x12、T1x12 和 T2x12 分别决定是系统时钟/12 还是系统时钟/1（不分频）后让 T0、T1 和 T2 进行计数。当定时器/计数器 T3 和 T4 工作在定时模式时，特殊功能寄存器 T4T3M 中的 T3x12 和 T4x12 分别决定是系统时钟/12 还是系统时钟/1（不分频）后让 T3 和 T4 进行计数。当定时器/计数器工作在计数模式时，对外部脉冲计数不分频。

定时器/计数器 0 有 4 种工作模式：模式 0（16 位自动重装载模式），模式 1（16 位不可重装载模式），模式 2（8 位自动重装模式），模式 3（不可屏蔽中断的 16 位自动重装载模式）。定时器/计数器 1 除模式 3 外，其他工作模式与定时器/计数器 0 相同。T1 在模式 3 时无效，停止计数。定时器 T2 的工作模式固定为 16 位自动重装载模式。T2 可以当定时器使用，也可以当串口的波特率发生器和可编程时钟输出。定时器 3、定时器 4 与定时器 T2 一样，它们的工作模式固定为 16 位自动重装载模式。T3/T4 可以当定时器使用，也可以当串口的波特率发生器和可编程时钟输出。

12.1 定时器的相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
TCON	定时器控制寄存器	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
TMOD	定时器模式寄存器	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	0000,0000
TL0	定时器 0 低 8 位寄存器	8AH									0000,0000
TL1	定时器 1 低 8 位寄存器	8BH									0000,0000
TH0	定时器 0 高 8 位寄存器	8CH									0000,0000
TH1	定时器 1 高 8 位寄存器	8DH									0000,0000
AUXR	辅助寄存器 1	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000,0001
INTCLKO	中断与时钟输出控制寄存器	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000
WKTCLE	掉电唤醒定时器低字节	AAH									1111,1111
WKTCH	掉电唤醒定时器高字节	ABH	WKTE								0111,1111
T4T3M	定时器 4/3 控制寄存器	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO	0000,0000
T4H	定时器 4 高字节	D2H									0000,0000
T4L	定时器 4 低字节	D3H									0000,0000
T3H	定时器 3 高字节	D4H									0000,0000
T3L	定时器 3 低字节	D5H									0000,0000
T2H	定时器 2 高字节	D6H									0000,0000

T2L	定时器 2 低字节	D7H		0000,0000
-----	-----------	-----	--	-----------

STCMCU

12.2 定时器 0/1

定时器 0/1 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TCON	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1: T1溢出中断标志。T1被允许计数以后, 从初值开始加1计数。当产生溢出时由硬件将TF1位置“1”, 并向CPU请求中断, 一直保持到CPU响应中断时, 才由硬件清“0”(也可由查询软件清“0”)。

TR1: 定时器T1的运行控制位。该位由软件置位和清零。当GATE (TMOD.7) =0, TR1=1时就允许T1开始计数, TR1=0时禁止T1计数。当GATE (TMOD.7) =1, TR1=1且INT1输入高电平时, 才允许T1计数。

TF0: T0溢出中断标志。T0被允许计数以后, 从初值开始加1计数, 当产生溢出时, 由硬件置“1”TF0, 向CPU请求中断, 一直保持CPU响应该中断时, 才由硬件清0(也可由查询软件清0)。

TR0: 定时器T0的运行控制位。该位由软件置位和清零。当GATE (TMOD.3) =0, TR0=1时就允许T0开始计数, TR0=0时禁止T0计数。当GATE (TMOD.3) =1, TR0=1且INT0输入高电平时, 才允许T0计数, TR0=0时禁止T0计数。

IE1: 外部中断1请求源 (INT1/P3.3) 标志。IE1=1, 外部中断向CPU请求中断, 当CPU响应该中断时由硬件清“0”IE1。

IT1: 外部中断源1触发控制位。IT1=0, 上升沿或下降沿均可触发外部中断1。IT1=1, 外部中断1程控为下降沿触发方式。

IE0: 外部中断0请求源 (INT0/P3.2) 标志。IE0=1外部中断0向CPU请求中断, 当CPU响应外部中断时, 由硬件清“0”IE0(边沿触发方式)。

IT0: 外部中断源0触发控制位。IT0=0, 上升沿或下降沿均可触发外部中断0。IT0=1, 外部中断0程控为下降沿触发方式。

定时器 0/1 模式寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TMOD	89H	T1_GATE	T1_C/T	T1_M1	T1_M0	T0_GATE	T0_C/T	T0_M1	T0_M0

T1_GATE: 控制定时器1, 置1时只有在INT1脚为高及TR1控制位置1时才可打开定时器/计数器1。

T0_GATE: 控制定时器0, 置1时只有在INT0脚为高及TR0控制位置1时才可打开定时器/计数器0。

T1_C/T: 控制定时器1用作定时器或计数器, 清0则用作定时器(对内部系统时钟进行计数), 置1用作计数器(对引脚T1/P3.5外部脉冲进行计数)。

T0_C/T: 控制定时器0用作定时器或计数器, 清0则用作定时器(对内部系统时钟进行计数), 置1用作计数器(对引脚T0/P3.4外部脉冲进行计数)。

T1_M1/T1_M0: 定时器定时器/计数器1模式选择

		定时器/计数器1工作模式
T1_M1 0	T1_M0 0	16位自动重载模式 当[TH1,TL1]中的16位计数值溢出时, 系统会自动将内部16位重载寄存器中的重载值装入[TH1,TL1]中。

0	1	16位不自动重载模式 当[TH1,TL1]中的16位计数值溢出时, 定时器1将从0开始计数
1	0	8位自动重载模式 当TL1中的8位计数值溢出时, 系统会自动将TH1中的重载值装入TL1中。
1	1	T1停止工作

T0_M1/T0_M0: 定时器定时器/计数器0模式选择

T0_M1	T0_M0	定时器/计数器0工作模式
0	0	16位自动重载模式 当[TH0,TL0]中的16位计数值溢出时, 系统会自动将内部16位重载寄存器中的重载值装入[TH0,TL0]中。
0	1	16位不自动重载模式 当[TH0,TL0]中的16位计数值溢出时, 定时器0将从0开始计数
1	0	8位自动重载模式 当TL0中的8位计数值溢出时, 系统会自动将TH0中的重载值装入TL0中。
1	1	16位自动重载模式 与模式0相同, 产生不可屏蔽中断

定时器 0 计数寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TL0	8AH								
TH0	8CH								

当定时器/计数器0工作在16位模式(模式0、模式1、模式3)时, TL0和TH0组合成为一个16位寄存器, TL0为低字节, TH0为高字节。若为8位模式(模式2)时, TL0和TH0为两个独立的8位寄存器。

定时器 1 计数寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TL1	8BH								
TH1	8DH								

当定时器/计数器1工作在16位模式(模式0、模式1)时, TL1和TH1组合成为一个16位寄存器, TL1为低字节, TH1为高字节。若为8位模式(模式2)时, TL1和TH1为两个独立的8位寄存器。

辅助寄存器 1 (AUXR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

T0x12: 定时器0速度控制位

0: 12T 模式, 即 CPU 时钟 12 分频 (FOSC/12)

1: 1T 模式, 即 CPU 时钟不分频分频 (FOSC/1)

T1x12: 定时器1速度控制位

0: 12T 模式, 即 CPU 时钟 12 分频 (FOSC/12)

1: 1T 模式, 即 CPU 时钟不分频分频 (FOSC/1)

中断与时钟输出控制寄存器 (INTCLKO)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INTCLKO	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO

T0CLKO: 定时器0时钟输出控制

0: 关闭时钟输出

1: 使能 P3.5 口的是定时器 0 时钟输出功能

当定时器 0 计数发生溢出时, P3.5 口的电平自动发生翻转。

T1CLKO: 定时器1时钟输出控制

0: 关闭时钟输出

1: 使能 P3.4 口的是定时器 1 时钟输出功能

当定时器 1 计数发生溢出时, P3.4 口的电平自动发生翻转。

12.3 定时器 2

辅助寄存器 1 (AUXR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

TR2: 定时器2的运行控制位

- 0: 定时器 2 停止计数
- 1: 定时器 2 开始计数

T2_C/T: 控制定时器0用作定时器或计数器, 清0则用作定时器(对内部系统时钟进行计数), 置1用作计数器(对引脚T2/P1.2外部脉冲进行计数)。

T2x12: 定时器2速度控制位

- 0: 12T 模式, 即 CPU 时钟 12 分频 (FOSC/12)
- 1: 1T 模式, 即 CPU 时钟不分频分频 (FOSC/1)

中断与时钟输出控制寄存器 (INTCLKO)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INTCLKO	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO

T2CLKO: 定时器2时钟输出控制

- 0: 关闭时钟输出
- 1: 使能 P1.3 口的是定时器 2 时钟输出功能
当定时器 2 计数发生溢出时, P1.3 口的电平自动发生翻转。

定时器 2 计数寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T2L	D7H								
T2H	D6H								

定时器/计数器2的工作模式固定为16位重载模式, T2L和T2H组合成为一个16位寄存器, T2L为低字节,

T2H为高字节。当[T2H,T2L]中的16位计数值溢出时, 系统会自动将内部16位重载寄存器中的重载值装入[T2H,T2L]中。

12.4 定时器 3/4

定时器 4/3 控制寄存器 (T4T3M)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO

TR4: 定时器4的运行控制位

- 0: 定时器 4 停止计数
- 1: 定时器 4 开始计数

T4_C/T: 控制定时器4用作定时器或计数器, 清0则用作定时器(对内部系统时钟进行计数), 置1用作计数器(对引脚T4/P0.6外部脉冲进行计数)。

T4x12: 定时器4速度控制位

- 0: 12T 模式, 即 CPU 时钟 12 分频 (FOSC/12)
- 1: 1T 模式, 即 CPU 时钟不分频分频 (FOSC/1)

T4CLKO: 定时器4时钟输出控制

- 0: 关闭时钟输出
- 1: 使能 P0.7 口的是定时器 4 时钟输出功能
当定时器 4 计数发生溢出时, P0.7 口的电平自动发生翻转。

TR3: 定时器3的运行控制位

- 0: 定时器 3 停止计数
- 1: 定时器 3 开始计数

T3_C/T: 控制定时器3用作定时器或计数器, 清0则用作定时器(对内部系统时钟进行计数), 置1用作计数器(对引脚T3/P0.4外部脉冲进行计数)。

T3x12: 定时器3速度控制位

- 0: 12T 模式, 即 CPU 时钟 12 分频 (FOSC/12)
- 1: 1T 模式, 即 CPU 时钟不分频分频 (FOSC/1)

T3CLKO: 定时器3时钟输出控制

- 0: 关闭时钟输出
- 1: 使能 P0.5 口的是定时器 3 时钟输出功能
当定时器 3 计数发生溢出时, P0.5 口的电平自动发生翻转。

定时器 3 计数寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T3L	D5H								
T3H	D4H								

定时器/计数器3的工作模式固定为16位重载模式, T3L和T3H组合成为一个16位寄存器, T3L为低字节, T3H为高字节。当[T3H,T3L]中的16位计数值溢出时, 系统会自动将内部16位重载寄存器中的重载值装入[T3H,T3L]中。

定时器 4 计数寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T4L	D3H								
T4H	D2H								

定时器/计数器4的工作模式固定为16位重载模式，T4L和T4H组合成为一个16位寄存器，T4L为低字节，T4H为高字节。当[T4H,T4L]中的16位计数值溢出时，系统会自动将内部16位重载寄存器中的重载值装入[T4H,T4L]中。

注意事项:

1. 定时器 0 从 P3.5 高速输出、定时器 1 从 P3.4 高速输出、定时器 2 从 P1.3 高速输出、定时器 3 从 P0.5 高速输出、定时器 4 从 P0.7 高速输出，这些 IO 设置为准双向口或推挽输出，脉冲输出正常，均为推挽输出，并且再直接操作 IO 将不影响输出波形。但是如果将 IO 设置为开漏输出（允许内部上拉电阻或外接上拉电阻），则如果对应的 IO 输出高电平，波形无输出（IO 为高阻），而对应的 IO 输出低电平，波形有推挽输出。

12.5 掉电唤醒定时器

内部掉电唤醒定时器是一个 15 位的计数器（由{WKTCH[6:0], WKTCL[7:0]}组成 15 位）。用于唤醒处于掉电模式的 MCU。

掉电唤醒定时器计数寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
WKTCL	AAH								
WKTCH	ABH	WKTEN							

WKTEN: 掉电唤醒定时器的使能控制位

0: 停用掉电唤醒定时器

1: 启用掉电唤醒定时器

如果 STC8 系列单片机内置掉电唤醒专用定时器被允许（通过软件将 WKTCH 寄存器中的 WKTEN 位置 1），当 MCU 进入掉电模式/停机模式后，掉电唤醒专用定时器开始计数，当计数值与用户所设置的值相等时，掉电唤醒专用定时器将 MCU 唤醒。MCU 唤醒后，程序从上次设置单片机进入掉电模式语句的下一条语句开始往下执行。掉电唤醒之后，可以通过读 WKTCH 和 WKTCL 中的内容获取单片机在掉电模式中的睡眠时间。

这里请注意：用户在寄存器{WKTCH[6:0], WKTCL[7:0]}中写入的值必须比实际计数值少 1。如用户需计数 10 次，则将 9 写入寄存器{WKTCH[6:0], WKTCL[7:0]}中。同样，如果用户需计数 32768 次，则应对{WKTCH[6:0], WKTCL[7:0]}写入 7FFFH（即 32767）。

内部掉电唤醒定时器有自己的内部时钟，其中掉电唤醒定时器计数一次的时间就是由该时钟决定的。内部掉电唤醒定时器的时钟频率约为 32KHz，当然误差较大。用户可以通过读 RAM 区 F8H 和 F9H 的内容（F8H 存放频率的高字节，F9H 存放低字节）来获取内部掉电唤醒专用定时器出厂时所记录的时钟频率。

掉电唤醒专用定时器计数时间的计算公式如下所示：（ F_{wt} 为我们从 RAM 区 F8H 和 F9H 获取到的内部掉电唤醒专用定时器的时钟频率）

$$\text{掉电唤醒定时器定时时间} = \frac{10^6 \times 16 \times \text{计数次数}}{F_{wt}} \text{ (微秒)}$$

假设 $F_{wt}=32\text{KHz}$ ，则有：

{WKTCH[6:0], WKTCL[7:0]}	掉电唤醒专用定时器计数时间
0	$10^6 \div 32\text{K} \times 16 \times (1+0) \approx 0.5 \text{ 毫秒}$
9	$10^6 \div 32\text{K} \times 16 \times (1+9) \approx 5 \text{ 毫秒}$
99	$10^6 \div 32\text{K} \times 16 \times (1+99) \approx 50 \text{ 毫秒}$
999	$10^6 \div 32\text{K} \times 16 \times (1+999) \approx 0.5 \text{ 秒}$
4095	$10^6 \div 32\text{K} \times 16 \times (1+4095) \approx 2 \text{ 秒}$
32767	$10^6 \div 32\text{K} \times 16 \times (1+32767) \approx 16 \text{ 秒}$

12.6 范例程序

12.6.1 定时器 0 (模式 0—16 位自动重载)

汇编代码

; 测试工作频率为 11.0592MHz

```

ORG      0000H
LJMP    MAIN
ORG      000BH
LJMP    TM0ISR

ORG      0100H
TM0ISR:
CPL     P1.0          ; 测试端口
RETI

MAIN:
MOV     SP,#3FH

MOV     TMOD,#00H      ; 模式 0
MOV     TL0,#66H        ; 65536-11.0592M/12/1000
MOV     TH0,#0FCH
SETB    TR0            ; 启动定时器
SETB    ET0            ; 使能定时器中断
SETB    EA

JMP     $

END

```

C 语言代码

```

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

// 测试工作频率为 11.0592MHz

sbit     P10      =  P1^0;

void TM0_Isr() interrupt 1
{
    P10 = !P10;          // 测试端口
}

void main()
{
    TMOD = 0x00;          // 模式 0
    TL0 = 0x66;           // 65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;              // 启动定时器
    ET0 = 1;              // 使能定时器中断
    EA = 1;

    while (1);
}

```

12.6.2 定时器 0 (模式 1—16 位不自动重载)

汇编代码

//测试工作频率为 11.0592MHz

```

ORG      0000H
LJMP    MAIN
ORG      000BH
LJMP    TM0ISR

ORG      0100H
TM0ISR:
MOV      TL0,#66H          ;重设定时参数
MOV      TH0,#0FCH
CPL      P1.0              ;测试端口
RETI

MAIN:
MOV      SP,#3FH
MOV      TMOD,#01H          ;模式 1
MOV      TL0,#66H          ;65536-11.0592M/12/1000
MOV      TH0,#0FCH
SETB    TR0                ;启动定时器
SETB    ET0                ;使能定时器中断
SETB    EA

JMP      $

END

```

C 语言代码

```

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

//测试工作频率为 11.0592MHz

sbit     P10      =  P1^0;

void TM0_Isr() interrupt 1
{
    TL0 = 0x66;           //重设定时参数
    TH0 = 0xfc;
    P10 = !P10;           //测试端口
}

void main()
{
    TMOD = 0x01;          //模式 1
    TL0 = 0x66;          //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;              //启动定时器
    ET0 = 1;              //使能定时器中断
    EA = 1;

    while (1);
}

```

12.6.3 定时器 0 (模式 2—8 位自动重载)

汇编代码

//测试工作频率为 11.0592MHz

```

ORG      0000H
LJMP    MAIN
ORG      000BH
LJMP    TM0ISR

TM0ISR:
ORG      0100H
        CPL     P1.0          ;测试端口
        RETI

MAIN:
MOV      SP,#3FH
MOV      TMOD,#02H      ;模式 2
MOV      TL0,#0F4H      ;256-11.0592M/12/76K
MOV      TH0,#0F4H
SETB    TR0             ;启动定时器
SETB    ET0             ;使能定时器中断
SETB    EA

JMP      $

END

```

C 语言代码

```

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

//测试工作频率为 11.0592MHz

sbit     P10      =  P1^0;

void TM0_Isr() interrupt 1
{
    P10 = !P10;          //测试端口
}

void main()
{
    TMOD = 0x02;        //模式 2
    TL0 = 0xf4;         //256-11.0592M/12/76K
    TH0 = 0xf4;
    TR0 = 1;            //启动定时器
    ET0 = 1;            //使能定时器中断
    EA = 1;

    while (1);
}

```

12.6.4 定时器 0 (模式 3—16 位自动重载不可屏蔽中断)

汇编代码

; 测试工作频率为 11.0592MHz

```

ORG      0000H
LJMP    MAIN
ORG      000BH
LJMP    TM0ISR

ORG      0100H
TM0ISR:
CPL     P1.0          ; 测试端口
RETI

MAIN:
MOV     SP,#3FH

MOV     TMOD,#03H      ; 模式 3
MOV     TL0,#66H        ; 65536-11.0592M/12/1000
MOV     TH0,#0FCH
SETB    TR0            ; 启动定时器
SETB    ET0            ; 使能定时器中断
;      SETB    EA             ; 不受 EA 控制

JMP     $

END

```

C 语言代码

```

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

// 测试工作频率为 11.0592MHz

sbit    P10      =  P1^0;

void TM0_Isr() interrupt 1
{
    P10 = !P10;          // 测试端口
}

void main()
{
    TMOD = 0x03;        // 模式 3
    TL0 = 0x66;          // 65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;            // 启动定时器
    ET0 = 1;            // 使能定时器中断
    // EA = 1;           // 不受 EA 控制

    while (1);
}

```

12.6.5 定时器 0 (外部计数—扩展T0 为外部下降沿中断)

汇编代码

; 测试工作频率为 11.0592MHz

```

ORG      0000H
LJMP    MAIN
ORG      000BH
LJMP    TM0ISR

TM0ISR:
ORG      0100H
CPL      P1.0          ; 测试端口
RETI

MAIN:
MOV      SP,#3FH

MOV      TMOD,#04H      ; 外部计数模式
MOV      TL0,#0FFH
MOV      TH0,#0FFH
SETB    TR0              ; 启动定时器
SETB    ET0              ; 使能定时器中断
SETB    EA

JMP      $

END

```

C 语言代码

```

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

// 测试工作频率为 11.0592MHz

sbit     P10      =  P1^0;

void TM0_Isr() interrupt 1
{
    P10 = !P10;          // 测试端口
}

void main()
{
    TMOD = 0x04;        // 外部计数模式
    TL0 = 0xff;
    TH0 = 0xff;
    TR0 = 1;            // 启动定时器
    ET0 = 1;            // 使能定时器中断
    EA = 1;

    while (1);
}

```

12.6.6 定时器 0 (测量脉宽—INT0 高电平宽度)

汇编代码

//测试工作频率为 11.0592MHz

```

AUXR      DATA      8EH
ORG        0000H
LJMP      MAIN
ORG        0003H
LJMP      INT0ISR

INT0ISR:
ORG        0100H
MOV        P0,TL0          ;TL0 为测量值低字节
MOV        P1,TH0          ;TH0 为测量值低高字节
RETI

MAIN:
MOV        SP,#3FH
MOV        AUXR,#80H        ;IT 模式
MOV        TMOD,#08H        ;使能 GATE,INT0 为 1 时使能计时
MOV        TL0,#00H
MOV        TH0,#00H
JB         INT0,$          ;等待 INT0 为低
SETB      TR0              ;启动定时器
SETB      IT0              ;使能 INT0 下降沿中断
SETB      EX0
SETB      EA

JMP        $
END

```

C 语言代码

```

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

//测试工作频率为 11.0592MHz

sfr      AUXR      = 0x8e;

void INT0_Isr() interrupt 0
{
    P0 = TL0;            //TL0 为测量值低字节 (会有约 10 个时钟的误差)
    P1 = TH0;            //TH0 为测量值低高字节
}

void main()
{
    AUXR = 0x80;          //IT 模式
    TMOD = 0x08;          //使能 GATE,INT0 为 1 时使能计时
    TL0 = 0x00;
    TH0 = 0x00;
    while (INT0);          //等待 INT0 为低
    TR0 = 1;              //启动定时器
}

```

```

IT0 = 1;           //使能INT0 下降沿中断
EX0 = 1;
EA = 1;

while (1);
}

```

12.6.7 定时器 0 (时钟分频输出)

汇编代码

; 测试工作频率为 11.0592MHz

```

INTCLKO    DATA      8FH

        ORG      0000H
        LJMP    MAIN

        ORG      0100H
MAIN:
        MOV      SP,#3FH

        MOV      TMOD,#00H          ;模式0
        MOV      TL0,#66H          ;65536-11.0592M/12/1000
        MOV      TH0,#0FCH
        SETB    TR0                ;启动定时器
        MOV      INTCLKO,#01H       ;使能时钟输出

        JMP      $

        END

```

C 语言代码

```

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

// 测试工作频率为 11.0592MHz

sfr     INTCLKO      = 0x8f;

void main()
{
    TMOD = 0x00;           //模式0
    TL0 = 0x66;           //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;               //启动定时器
    INTCLKO = 0x01;       //使能时钟输出

    while (1);
}

```

12.6.8 定时器 1 (模式 0—16 位自动重载)

汇编代码

; 测试工作频率为 11.0592MHz

	<i>ORG</i>	<i>0000H</i>
	<i>LJMP</i>	<i>MAIN</i>
	<i>ORG</i>	<i>001BH</i>
	<i>LJMP</i>	<i>TMIISR</i>
	 <i>TMISR:</i>	
	<i>ORG</i>	<i>0100H</i>
	<i>CPL</i>	<i>P1.0</i>
	<i>RETI</i>	; 测试端口
	 <i>MAIN:</i>	
	<i>MOV</i>	<i>SP,#3FH</i>
	<i>MOV</i>	<i>TMOD,#00H</i> ; 模式0
	<i>MOV</i>	<i>TLI,#66H</i> ; 65536-11.0592M/12/1000
	<i>MOV</i>	<i>THI,#0FC</i>
	<i>SETB</i>	<i>TR1</i> ; 启动定时器
	<i>SETB</i>	<i>ET1</i> ; 使能定时器中断
	<i>SETB</i>	<i>EA</i>
	<i>JMP</i>	\$
		<i>END</i>

C 语言代码

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

// 测试工作频率为 11.0592MHz

sbit P10 = P1^0;

void TMI_Isr() interrupt 3
{
    P10 = !P10; // 测试端口
}

void main()
{
    TMOD = 0x00; // 模式0
    TL1 = 0x66; // 65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1; // 启动定时器
    ET1 = 1; // 使能定时器中断
    EA = 1;

    while (1);
}
```

12.6.9 定时器 1 (模式 1—16 位不自动重载)

汇编代码

; 测试工作频率为 11.0592MHz

<i>ORG</i>	<i>0000H</i>
<i>LJMP</i>	<i>MAIN</i>

<i>ORG</i>	<i>001BH</i>
<i>LJMP</i>	<i>TMISR</i>
<i>ORG</i>	<i>0100H</i>
<i>TMISR:</i>	
<i>MOV</i>	<i>TLI,#66H</i>
<i>MOV</i>	<i>THI,#0FC</i>
<i>CPL</i>	<i>P1.0</i>
<i>RETI</i>	
<i>MAIN:</i>	
<i>MOV</i>	<i>SP,#3FH</i>
<i>MOV</i>	<i>TMOD,#10H</i>
<i>MOV</i>	<i>TLI,#66H</i>
<i>MOV</i>	<i>THI,#0FC</i>
<i>SETB</i>	<i>TR1</i>
<i>SETB</i>	<i>ET1</i>
<i>SETB</i>	<i>EA</i>
<i>JMP</i>	<i>\$</i>
 <i>END</i>	

C 语言代码

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

//测试工作频率为 11.0592MHz

sbit P10 = P1^0;

void TM1_Isr() interrupt 3
{
    TL1 = 0x66;           //重设定时参数
    TH1 = 0xfc;
    P10 = !P10;           //测试端口
}

void main()
{
    TMOD = 0x10;          //模式 1
    TL1 = 0x66;
    TH1 = 0xfc;
    TR1 = 1;              //启动定时器
    ET1 = 1;              //使能定时器中断
    EA = 1;

    while (1);
}
```

12.6.10 定时器 1 (模式 2—8 位自动重载)

汇编代码

; 测试工作频率为 11.0592MHz

	<i>ORG</i>	<i>0000H</i>
	<i>LJMP</i>	<i>MAIN</i>
	<i>ORG</i>	<i>001BH</i>
	<i>LJMP</i>	<i>TMIISR</i>
	 <i>TMISR:</i>	
	<i>ORG</i>	<i>0100H</i>
	<i>CPL</i>	<i>P1.0</i>
	<i>RETI</i>	; 测试端口
	 <i>MAIN:</i>	
	<i>MOV</i>	<i>SP,#3FH</i>
	<i>MOV</i>	<i>TMOD,#20H</i>
	<i>MOV</i>	; 模式 2
	<i>MOV</i>	<i>TLI,#0F4H</i>
	<i>MOV</i>	; 256-11.0592M/12/76K
	<i>MOV</i>	<i>THI,#0F4H</i>
	<i>SETB</i>	<i>TR1</i>
	<i>SETB</i>	; 启动定时器
	<i>SETB</i>	<i>ET1</i>
	<i>SETB</i>	; 使能定时器中断
	<i>SETB</i>	<i>EA</i>
	<i>JMP</i>	\$
	 <i>END</i>	

C 语言代码

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

// 测试工作频率为 11.0592MHz

sbit P10 = P1^0;

void TMI_Isr() interrupt 3
{
    P10 = !P10; // 测试端口
}

void main()
{
    TMOD = 0x20; // 模式 2
    TL1 = 0xf4; // 256-11.0592M/12/76K
    TH1 = 0xf4;
    TR1 = 1; // 启动定时器
    ET1 = 1; // 使能定时器中断
    EA = 1;

    while (1);
}
```

12.6.11 定时器 1 (外部计数—扩展T1 为外部下降沿中断)

汇编代码

; 测试工作频率为 11.0592MHz

<i>ORG</i>	<i>0000H</i>
<i>LJMP</i>	<i>MAIN</i>

```

          001BH
ljmp      TMISR

ORG        0100H
TMISR:
cpl        P1.0
            ; 测试端口
reti

MAIN:
mov        SP,#3FH

mov        TMOD,#40H      ; 外部计数模式
mov        TL1,#0FFH
mov        TH1,#0FFH
setb      TRI
            ; 启动定时器
setb      ET1
            ; 使能定时器中断
setb      EA

jmp        $

END

```

C 语言代码

```

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

// 测试工作频率为 11.0592MHz

sbit      P10          =  P1^0;

void TM1_Isr() interrupt 3
{
    P10 = !P10;           // 测试端口
}

void main()
{
    TMOD = 0x40;          // 外部计数模式
    TL1 = 0xff;
    TH1 = 0xff;
    TRI = 1;              // 启动定时器
    ET1 = 1;              // 使能定时器中断
    EA = 1;

    while (1);
}

```

12.6.12 定时器 1 (测量脉宽—INT1 高电平宽度)

汇编代码

; 测试工作频率为 11.0592MHz

```

auxr      data      8EH

org        0000H
ljmp      main

```

<i>ORG</i>	<i>0013H</i>	
<i>LJMP</i>	<i>INTISR</i>	
<i>ORG</i>	<i>0100H</i>	
<i>INTISR:</i>		
<i>MOV</i>	<i>P0,TLI</i>	<i>;TLI</i> 为测量值低字节
<i>MOV</i>	<i>P1,THI</i>	<i>;THI</i> 为测量值低高字节
<i>RETI</i>		
<i>MAIN:</i>		
<i>MOV</i>	<i>SP,#3FH</i>	
<i>MOV</i>	<i>AUXR,#40H</i>	<i>;IT</i> 模式
<i>MOV</i>	<i>TMOD,#80H</i>	<i>;使能 GATE,INT1 为 1 时使能计时</i>
<i>MOV</i>	<i>TLI,#00H</i>	
<i>MOV</i>	<i>THI,#00H</i>	
<i>JB</i>	<i>INT1,\$</i>	<i>;等待 INT1 为低</i>
<i>SETB</i>	<i>TRI</i>	<i>;启动定时器</i>
<i>SETB</i>	<i>IT1</i>	<i>;使能 INT1 下降沿中断</i>
<i>SETB</i>	<i>EXI</i>	
<i>SETB</i>	<i>EA</i>	
<i>JMP</i>	<i>\$</i>	
 <i>END</i>		

C 语言代码

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

//测试工作频率为 11.0592MHz

sfr AUXR = 0x8e;

void INT1_Isr() interrupt 2
{
    P0 = TLI; //TLI 为测量值低字节 (会有约 11 个时钟的误差)
    P1 = THI; //THI 为测量值低高字节
}

void main()
{
    AUXR = 0x40; //IT 模式
    TMOD = 0x80; //使能 GATE,INT1 为 1 时使能计时
    TLI = 0x00;
    THI = 0x00;
    while (INT1); //等待 INT1 为低
    TRI = 1; //启动定时器
    IT1 = 1; //使能 INT1 下降沿中断
    EXI = 1;
    EA = 1;

    while (1);
}
```

12.6.13 定时器 1 (时钟分频输出)

汇编代码

//测试工作频率为 11.0592MHz

```

INTCLKO    DATA      8FH
            ORG      0000H
            LJMP    MAIN

            ORG      0100H
MAIN:      MOV      SP,#3FH
            MOV      TMOD,#00H          ;模式 0
            MOV      TL1,#66H          ;65536-11.0592M/12/1000
            MOV      TH1,#0FCH
            SETB    TRI                ;启动定时器
            MOV      INTCLKO,#02H        ;使能时钟输出
            JMP      $
END

```

C 语言代码

```

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

//测试工作频率为 11.0592MHz

sfr     INTCLKO      = 0x8f;

void main()
{
    TMOD = 0x00;           //模式 0
    TL1 = 0x66;           //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TRI = 1;              //启动定时器
    INTCLKO = 0x02;        //使能时钟输出

    while (1);
}

```

12.6.14 定时器 1 (模式 0) 做串口 1 波特率发生器

汇编代码

```

AUXR      DATA      8EH
            BUSY    BIT      20H.0
            WPTR    DATA      21H
            RPTR    DATA      22H
            BUFFER  DATA      23H          ;16 bytes

            ORG      0000H
            LJMP    MAIN

```

ORG *0023H*
LJMP *UART_ISR*

ORG *0100H*

UART_ISR:

PUSH *ACC*
PUSH *PSW*
MOV *PSW,#08H*

JNB *TI,CHKRI*
CLR *TI*
CLR *BUSY*

CHKRI:

JNB *RI,UARTISR_EXIT*
CLR *RI*
MOV *A,WPTR*
ANL *A,#0FH*
ADD *A,#BUFFER*
MOV *R0,A*
MOV *@R0,SBUF*
INC *WPTR*

UARTISR_EXIT:

POP *PSW*
POP *ACC*
RETI

UART_INIT:

MOV *SCON,#50H*
MOV *TMOD,#00H*
MOV *TLI,#0E8H*
MOV *THI,#0FFH*
SETB *TR1*
MOV *AUXR,#40H*
CLR *BUSY*
MOV *WPTR,#00H*
MOV *RPTR,#00H*
RET

;65536-11059200/115200/4=0FFE8H

UART_SEND:

JB *BUSY,\$*
SETB *BUSY*
MOV *SBUFA,A*
RET

UART_SENDSTR:

CLR *A*
MOVC *A,@A+DPTR*
JZ *SENDEND*
LCALL *UART_SEND*
INC *DPTR*
JMP *UART_SENDSTR*

SENDEND:

RET

MAIN:

MOV *SP,#3FH*
LCALL *UART_INIT*

<i>SETB</i>	<i>ES</i>
<i>SETB</i>	<i>EA</i>
<i>MOV</i>	<i>DPTR,#STRING</i>
<i>LCALL</i>	<i>UART_SENDSTR</i>

LOOP:

<i>MOV</i>	<i>A,RPTR</i>
<i>XRL</i>	<i>A,WPTR</i>
<i>ANL</i>	<i>A,#0FH</i>
<i>JZ</i>	<i>LOOP</i>
<i>MOV</i>	<i>A,RPTR</i>
<i>ANL</i>	<i>A,#0FH</i>
<i>ADD</i>	<i>A,#BUFFER</i>
<i>MOV</i>	<i>R0,A</i>
<i>MOV</i>	<i>A,@R0</i>
<i>LCALL</i>	<i>UART_SEND</i>
<i>INC</i>	<i>RPTR</i>
<i>JMP</i>	<i>LOOP</i>

STRING: *DB* 'Uart Test !',0DH,0AH,00H

END

C 语言代码

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

#define FOSC 11059200UL
#define BRT (65536 - FOSC / 115200 / 4)

sfr AUXR = 0x8e;

bit busy;
char wptr;
char rptr;
char buffer[16];

void UartIsr() interrupt 4
{
    if(TI)
    {
        TI = 0;
        busy = 0;
    }
    if(RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
}
```

```

TH1 = BRT >> 8;
TR1 = 1;
AUXR = 0x40;
wptr = 0x00;
rptr = 0x00;
busy = 0;
}

```

```

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

```

```

void UartSendStr(char *p)
{
    while (*p)
    {
        UartSend(*p++);
    }
}

```

```

void main()
{
    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !r\n");
    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}

```

12.6.15 定时器 1（模式 2）做串口 1 波特率发生器

汇编代码

AUXR	DATA	8EH
BUSY	BIT	20H.0
WPTR	DATA	21H
RPTR	DATA	22H
BUFFER	DATA	23H ;16 bytes
	ORG	0000H
	LJMP	MAIN
	ORG	0023H
	LJMP	UART_ISR
	ORG	0100H

UART_ISR:

<i>PUSH</i>	<i>ACC</i>	
<i>PUSH</i>	<i>PSW</i>	
<i>MOV</i>	<i>PSW,#08H</i>	
<i>JNB</i>	<i>TI,CHKRI</i>	
<i>CLR</i>	<i>TI</i>	
<i>CLR</i>	<i>BUSY</i>	
<i>CHKRI:</i>		
<i>JNB</i>	<i>RI,UARTISR_EXIT</i>	
<i>CLR</i>	<i>RI</i>	
<i>MOV</i>	<i>A,WPTR</i>	
<i>ANL</i>	<i>A,#0FH</i>	
<i>ADD</i>	<i>A,#BUFFER</i>	
<i>MOV</i>	<i>R0,A</i>	
<i>MOV</i>	<i>@R0,SBUF</i>	
<i>INC</i>	<i>WPTR</i>	
<i>UARTISR_EXIT:</i>		
<i>POP</i>	<i>PSW</i>	
<i>POP</i>	<i>ACC</i>	
<i>RETI</i>		
 <i>UART_INIT:</i>		
<i>MOV</i>	<i>SCON,#50H</i>	
<i>MOV</i>	<i>TMOD,#20H</i>	
<i>MOV</i>	<i>TLI,#0FDH</i>	;256-11059200/115200/32=0FDH
<i>MOV</i>	<i>THI,#0FDH</i>	
<i>SETB</i>	<i>TRI</i>	
<i>MOV</i>	<i>AUXR,#40H</i>	
<i>CLR</i>	<i>BUSY</i>	
<i>MOV</i>	<i>WPTR,#00H</i>	
<i>MOV</i>	<i>RPTR,#00H</i>	
<i>RET</i>		
 <i>UART_SEND:</i>		
<i>JB</i>	<i>BUSY,\$</i>	
<i>SETB</i>	<i>BUSY</i>	
<i>MOV</i>	<i>SBUF,A</i>	
<i>RET</i>		
 <i>UART_SENDSTR:</i>		
<i>CLR</i>	<i>A</i>	
<i>MOVC</i>	<i>A,@A+DPTR</i>	
<i>JZ</i>	<i>SENDEND</i>	
<i>LCALL</i>	<i>UART_SEND</i>	
<i>INC</i>	<i>DPTR</i>	
<i>JMP</i>	<i>UART_SENDSTR</i>	
<i>SENDEND:</i>		
<i>RET</i>		
 <i>MAIN:</i>		
<i>MOV</i>	<i>SP,#3FH</i>	
 <i>LCALL</i>	<i>UART_INIT</i>	
<i>SETB</i>	<i>ES</i>	
<i>SETB</i>	<i>EA</i>	
 <i>MOV</i>	<i>DPTR,#STRING</i>	
<i>LCALL</i>	<i>UART_SENDSTR</i>	

LOOP:

MOV	A,RPTR
XRL	A,WPTR
ANL	A,#0FH
JZ	LOOP
MOV	A,RPTR
ANL	A,#0FH
ADD	A,#BUFFER
MOV	R0,A
MOV	A,@R0
LCALL	UART_SEND
INC	RPTR
JMP	LOOP

STRING: **DB** 'Uart Test !',0DH,0AH,00H

END

C 语言代码

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

#define FOSC 11059200UL
#define BRT (256 - FOSC / 115200 / 32)

sfr AUXR = 0x8e;

bit busy;
char wptr;
char rptr;
char buffer[16];

void UartIsr() interrupt 4
{
    if(TI)
    {
        TI = 0;
        busy = 0;
    }
    if(RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x20;
    TL1 = BRT;
    TH1 = BRT;
    TR1 = 1;
    AUXR = 0x40;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}
```

```

}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void UartSendStr(char *p)
{
    while (*p)
    {
        UartSend(*p++);
    }
}

void main()
{
    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}

```

12.6.16 定时器 2 (16 位自动重载)

汇编代码

; 测试工作频率为 11.0592MHz

T2L	DATA	0D7H
T2H	DATA	0D6H
AUXR	DATA	8EH
IE2	DATA	0AFH
ET2	EQU	04H
AUXINTIF	DATA	0EFH
T2IF	EQU	01H
ORG	0000H	
LJMP	MAIN	
ORG	0063H	
LJMP	TM2ISR	
ORG	0100H	
TM2ISR:	CPL	P1.0
	ANL	AUXINTIF,#NOT T2IF
	RETI	

MAIN:

```

MOV      SP,#3FH
MOV      T2L,#66H          ;65536-11.0592M/12/1000
MOV      T2H,#0FCH
MOV      AUXR,#10H          ;启动定时器
MOV      IE2,#ET2           ;使能定时器中断
SETB    EA
JMP      $

```

END**C 语言代码**

```

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

//测试工作频率为 11.0592MHz

sfr      T2L      = 0xd7;
sfr      T2H      = 0xd6;
sfr      AUXR     = 0x8e;
sfr      IE2      = 0xaf;
#define   ET2      0x04
sfr      AUXINTIF = 0xef;
#define   T2IF     0x01

sbit     P10      = P1^0;

void TM2_Isr() interrupt 12
{
    P10 = !P10;           //测试端口
    AUXINTIF &= ~T2IF;   //清中断标志
}

void main()
{
    T2L = 0x66;          //65536-11.0592M/12/1000
    T2H = 0xfc;
    AUXR = 0x10;          //启动定时器
    IE2 = ET2;            //使能定时器中断
    EA = 1;

    while (1);
}

```

12.6.17 定时器 2 (外部计数—扩展T2 为外部下降沿中断)**汇编代码****;测试工作频率为 11.0592MHz**

T2L	DATA	0D7H
T2H	DATA	0D6H
AUXR	DATA	8EH
IE2	DATA	0AFH
ET2	EQU	04H

```

AUXINTIF    DATA      0EFH
T2IF        EQU       01H

        ORG      0000H
        LJMP    MAIN
        ORG      0063H
        LJMP    TM2ISR

        ORG      0100H
TM2ISR:
        CPL      P1.0          ; 测试端口
        ANL      AUXINTIF,#NOT T2IF ; 清中断标志
        RETI

MAIN:
        MOV      SP,#3FH

        MOV      T2L,#0FFH
        MOV      T2H,#0FFH
        MOV      AUXR,#18H          ; 设置外部计数模式并启动定时器
        MOV      IE2,#ET2           ; 使能定时器中断
        SETB     EA

        JMP      $

END

```

C 语言代码

```

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

// 测试工作频率为 11.0592MHz

sfr      T2L      = 0xd7;
sfr      T2H      = 0xd6;
sfr      AUXR     = 0x8e;
sfr      IE2      = 0xaf;
#define   ET2      0x04
sfr      AUXINTIF = 0xef;
#define   T2IF     0x01

sbit     P10      = P1^0;

void TM2_Isr() interrupt 12
{
    P10 = !P10;                  // 测试端口
    AUXINTIF &= ~T2IF;          // 清中断标志
}

void main()
{
    T2L = 0xff;
    T2H = 0xff;
    AUXR = 0x18;                // 设置外部计数模式并启动定时器
    IE2 = ET2;                  // 使能定时器中断
    EA = 1;

    while (1);
}

```

}

12.6.18 定时器 2 (时钟分频输出)

汇编代码

; 测试工作频率为 11.0592MHz

```

T2L      DATA    0D7H
T2H      DATA    0D6H
AUXR    DATA    8EH
INTCLKO  DATA    8FH

        ORG    0000H
        LJMP   MAIN

MAIN:
        ORG    0100H
        MOV    SP,#3FH

        MOV    T2L,#66H           ;65536-11.0592M/12/1000
        MOV    T2H,#0FCH
        MOV    AUXR,#10H          ;启动定时器
        MOV    INTCLKO,#04H       ;使能时钟输出

        JMP    $

        END

```

C 语言代码

```

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

// 测试工作频率为 11.0592MHz

sfr    T2L      = 0xd7;
sfr    T2H      = 0xd6;
sfr    AUXR    = 0x8e;
sfr    INTCLKO = 0x8f;

void main()
{
    T2L = 0x66;           //65536-11.0592M/12/1000
    T2H = 0xfc;
    AUXR = 0x10;          //启动定时器
    INTCLKO = 0x04;       //使能时钟输出

    while (1);
}

```

12.6.19 定时器 2 做串口 1 波特率发生器

汇编代码

AUXR	DATA	8EH
T2H	DATA	0D6H
T2L	DATA	0D7H

BUSY	BIT	20H.0
WPTR	DATA	21H
RPTR	DATA	22H
BUFFER	DATA	23H

;16 bytes

ORG	0000H
LJMP	MAIN
ORG	0023H
LJMP	UART_ISR
 ORG	 0100H

UART_ISR:

PUSH	ACC
PUSH	PSW
MOV	PSW,#08H
 JNB	 TI,CHKRI
CLR	TI
CLR	BUSY

CHKRI:

JNB	RI,UARTISR_EXIT
CLR	RI
MOV	A,WPTR
ANL	A,#0FH
ADD	A,#BUFFER
MOV	R0,A
MOV	@R0,SBUF
INC	WPTR

UARTISR_EXIT:

POP	PSW
POP	ACC
RETI	

UART_INIT:

MOV	SCON,#50H
MOV	T2L,#0E8H
MOV	T2H,#0FFH
MOV	AUXR,#15H
CLR	BUSY
MOV	WPTR,#00H
MOV	RPTR,#00H
RET	

;65536-11059200/115200/4=0FFE8H

UART_SEND:

JB	BUSY,\$
SETB	BUSY
MOV	SBUF,A
RET	

UART_SENDSTR:

CLR	A
MOVC	A,@A+DPTR
JZ	SENDEND
LCALL	UART_SEND
INC	DPTR
JMP	UART_SENDSTR

SENDEND:

RET***MAIN:***

```

MOV      SP,#3FH
LCALL    UART_INIT
SETB     ES
SETB     EA

MOV      DPTR,#STRING
LCALL    UART_SENDSTR

```

LOOP:

```

MOV      A,RPTR
XRL     A,WPTR
ANL     A,#0FH
JZ       LOOP
MOV      A,RPTR
ANL     A,#0FH
ADD     A,#BUFFER
MOV      R0,A
MOV      A,@R0
LCALL   UART_SEND
INC     RPT
JMP     LOOP

```

STRING: **DB** *'Uart Test !',0DH,0AH,00H*

END**C 语言代码**

```

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

#define FOSC        11059200UL
#define BRT        (65536 - FOSC / 115200 / 4)

sfr     AUXR     = 0x8e;
sfr     T2H     = 0xd6;
sfr     T2L     = 0xd7;

bit     busy;
char    wptr;
char    rptr;
char    buffer[16];

void UartIsr() interrupt 4
{
    if(TI)
    {
        TI = 0;
        busy = 0;
    }
    if(RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

```

```

        }

}

void UartInit()
{
    SCON = 0x50;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x15;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void UartSendStr(char *p)
{
    while (*p)
    {
        UartSend(*p++);
    }
}

void main()
{
    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}

```

12.6.20 定时器 2 做串口 2 波特率发生器

汇编代码

AUXR	DATA	8EH
T2H	DATA	0D6H
T2L	DATA	0D7H
S2CON	DATA	9AH
S2BUF	DATA	9BH
IE2	DATA	0AFH
BUSY	BIT	20H.0
WPTR	DATA	21H

RPTR	DATA	22H	
BUFFER	DATA	23H	<i>;16 bytes</i>
	ORG	0000H	
	LJMP	MAIN	
	ORG	0043H	
	LJMP	UART2_ISR	
	ORG	0100H	
UART2_ISR:			
	PUSH	ACC	
	PUSH	PSW	
	MOV	PSW,#08H	
	MOV	A,S2CON	
	JNB	ACC.1,CHKRI	
	ANL	S2CON,#NOT 02H	
	CLR	BUSY	
CHKRI:			
	JNB	ACC.0,UART2ISR_EXIT	
	ANL	S2CON,#NOT 01H	
	MOV	A,W PTR	
	ANL	A,#0FH	
	ADD	A,#BUFFER	
	MOV	R0,A	
	MOV	@R0,S2BUF	
	INC	WPTR	
UART2ISR_EXIT:			
	POP	PSW	
	POP	ACC	
	RETI		
UART2_INIT:			
	MOV	S2CON,#10H	
	MOV	T2L,#0E8H	<i>;65536-11059200/115200/4=0FFE8H</i>
	MOV	T2H,#0FFH	
	MOV	AUXR,#14H	
	CLR	BUSY	
	MOV	WPTR,#00H	
	MOV	RPTR,#00H	
	RET		
UART2_SEND:			
	JB	BUSY,\$	
	SETB	BUSY	
	MOV	S2BUFA	
	RET		
UART2_SENDSTR:			
	CLR	A	
	MOVC	A,@A+DPTR	
	JZ	SEND2END	
	LCALL	UART2_SEND	
	INC	DPTR	
	JMP	UART2_SENDSTR	
SEND2END:			
	RET		

MAIN:

```

MOV      SP,#3FH
LCALL    UART2_INIT
MOV      IE2,#01H
SETB    EA
MOV      DPTR,#STRING
LCALL    UART2_SENDSTR

```

LOOP:

```

MOV      A,RPTR
XRL      A,WPTR
ANL      A,#0FH
JZ       LOOP
MOV      A,RPTR
ANL      A,#0FH
ADD      A,#BUFFER
MOV      R0,A
MOV      A,@R0
LCALL   UART2_SEND
INC      RPTR
JMP      LOOP

```

STRING: DB 'Uart Test !',0DH,0AH,00H

END

C 语言代码

```

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

#define FOSC      11059200UL
#define BRT       (65536 - FOSC / 115200 / 4)

sfr     AUXR      = 0x8e;
sfr     T2H       = 0xd6;
sfr     T2L       = 0xd7;
sfr     S2CON     = 0x9a;
sfr     S2BUF     = 0x9b;
sfr     IE2       = 0xaf;

bit    busy;
char   wptr;
char   rptr;
char   buffer[16];

void Uart2Isr() interrupt 8
{
    if(S2CON & 0x02)
    {
        S2CON &= ~0x02;
        busy = 0;
    }
    if(S2CON & 0x01)
    {
        S2CON &= ~0x01;
        buffer[wptr++] = S2BUF;
    }
}

```

```
wptr &= 0x0f;
}
}
```

```
void Uart2Init()
{
    S2CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}
```

```
void Uart2Send(char dat)
{
    while (busy);
    busy = 1;
    S2BUF = dat;
}
```

```
void Uart2SendStr(char *p)
{
    while (*p)
    {
        Uart2Send(*p++);
    }
}
```

```
void main()
{
    Uart2Init();
    IE2 = 0x01;
    EA = 1;
    Uart2SendStr("Uart Test !r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart2Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

12.6.21 定时器 2 做串口 3 波特率发生器

汇编代码

AUXR	DATA	8EH
T2H	DATA	0D6H
T2L	DATA	0D7H
S3CON	DATA	0ACh
S3BUF	DATA	0ADH
IE2	DATA	0AFH

BUSY	BIT	20H.0
-------------	------------	--------------

WPTR DATA 21H
RPTR DATA 22H
BUFFER DATA 23H ;16 bytes

ORG 0000H
LJMP MAIN
ORG 008BH
LJMP UART3_ISR
ORG 0100H

UART3_ISR:

PUSH ACC
PUSH PSW
MOV PSW,#08H

MOV A,S3CON
JNB ACC.1,CHKRI
ANL S3CON,#NOT 02H
CLR BUSY

CHKRI:

JNB ACC.0,UART3ISR_EXIT
ANL S3CON,#NOT 01H
MOV A,WPTR
ANL A,#0FH
ADD A,#BUFFER
MOV R0,A
MOV @R0,S3BUF
INC WPTR

UART3ISR_EXIT:

POP PSW
POP ACC
RETI

UART3_INIT:

MOV S3CON,#10H
MOV T2L,#0E8H ;65536-11059200/115200/4=0FFE8H
MOV T2H,#0FFH
MOV AUXR,#14H
CLR BUSY
MOV WPTR,#00H
MOV RPTR,#00H
RET

UART3_SEND:

JB BUSY,\$
SETB BUSY
MOV S3BUF,A
RET

UART3_SENDSTR:

CLR A
MOVC A,@A+DPTR
JZ SEND3END
LCALL UART3_SEND
INC DPTR
JMP UART3_SENDSTR

SEND3END:

RET

MAIN:

```

MOV      SP,#3FH
LCALL    UART3_INIT
MOV      IE2,#08H
SETB    EA
MOV      DPTR,#STRING
LCALL    UART3_SENDSTR

```

LOOP:

```

MOV      A,RPTR
XRL      A,WPTR
ANL      A,#0FH
JZ       LOOP
MOV      A,RPTR
ANL      A,#0FH
ADD      A,#BUFFER
MOV      R0,A
MOV      A,@R0
LCALL   UART3_SEND
INC      RPT
JMP      LOOP

```

STRING: DB 'Uart Test !,0DH,0AH,00H

END

C 语言代码

```

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

#define FOSC      11059200UL
#define BRT       (65536 - FOSC / 115200 / 4)

sfr     AUXR      = 0x8e;
sfr     T2H       = 0xd6;
sfr     T2L       = 0xd7;
sfr     S3CON     = 0xac;
sfr     S3BUF     = 0xad;
sfr     IE2       = 0xaf;

bit     busy;
char   wptr;
char   rptr;
char   buffer[16];

void Uart3Isr() interrupt 17
{
    if(S3CON & 0x02)
    {
        S3CON &= ~0x02;
        busy = 0;
    }
    if(S3CON & 0x01)
    {
        S3CON &= ~0x01;
    }
}

```

```

        buffer[wptr++] = S3BUF;
        wptr &= 0x0f;
    }
}

void Uart3Init()
{
    S3CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart3Send(char dat)
{
    while (busy);
    busy = 1;
    S3BUF = dat;
}

void Uart3SendStr(char *p)
{
    while (*p)
    {
        Uart3Send(*p++);
    }
}

void main()
{
    Uart3Init();
    IE2 = 0x08;
    EA = 1;
    Uart3SendStr("Uart Test !r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart3Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}

```

12.6.22 定时器 2 做串口 4 波特率发生器

汇编代码

AUXR	DATA	8EH
T2H	DATA	0D6H
T2L	DATA	0D7H
S4CON	DATA	84H
S4BUF	DATA	085H
IE2	DATA	0AFH

<i>BUSY</i>	<i>BIT</i>	<i>20H.0</i>
<i>WPTR</i>	<i>DATA</i>	<i>21H</i>
<i>RPTR</i>	<i>DATA</i>	<i>22H</i>
<i>BUFFER</i>	<i>DATA</i>	<i>23H</i>

;16 bytes

<i>ORG</i>	<i>0000H</i>
<i>LJMP</i>	<i>MAIN</i>
<i>ORG</i>	<i>0093H</i>
<i>LJMP</i>	<i>UART4_ISR</i>
 <i>ORG</i>	 <i>0100H</i>

UART4_ISR:

<i>PUSH</i>	<i>ACC</i>
<i>PUSH</i>	<i>PSW</i>
<i>MOV</i>	<i>PSW,#08H</i>
 <i>MOV</i>	<i>A,S4CON</i>
<i>JNB</i>	<i>ACC.1,CHKRI</i>
<i>ANL</i>	<i>S4CON,#NOT 02H</i>
<i>CLR</i>	<i>BUSY</i>

CHKRI:

<i>JNB</i>	<i>ACC.0,UART4ISR_EXIT</i>
<i>ANL</i>	<i>S4CON,#NOT 01H</i>
<i>MOV</i>	<i>A,WPTR</i>
<i>ANL</i>	<i>A,#0FH</i>
<i>ADD</i>	<i>A,#BUFFER</i>
<i>MOV</i>	<i>R0,A</i>
<i>MOV</i>	<i>@R0,S4BUF</i>
<i>INC</i>	<i>WPTR</i>

UART4ISR_EXIT:

<i>POP</i>	<i>PSW</i>
<i>POP</i>	<i>ACC</i>
<i>RETI</i>	

UART4_INIT:

<i>MOV</i>	<i>S4CON,#10H</i>
<i>MOV</i>	<i>T2L,#0E8H</i>
<i>MOV</i>	<i>T2H,#0FFH</i>
<i>MOV</i>	<i>AUXR,#14H</i>
<i>CLR</i>	<i>BUSY</i>
<i>MOV</i>	<i>WPTR,#00H</i>
<i>MOV</i>	<i>RPTR,#00H</i>
<i>RET</i>	

*;65536-11059200/115200/4=0FFE8H****UART4_SEND:***

<i>JB</i>	<i>BUSY,\$</i>
<i>SETB</i>	<i>BUSY</i>
<i>MOV</i>	<i>S4BUF,A</i>
<i>RET</i>	

UART4_SENDSTR:

<i>CLR</i>	<i>A</i>
<i>MOVC</i>	<i>A,@A+DPTR</i>
<i>JZ</i>	<i>SEND4END</i>
<i>LCALL</i>	<i>UART4_SEND</i>
<i>INC</i>	<i>DPTR</i>
<i>JMP</i>	<i>UART4_SENDSTR</i>

SEND4END:

RET***MAIN:***

```

MOV      SP,#3FH
LCALL    UART4_INIT
MOV      IE2,#10H
SETB    EA

MOV      DPTR,#STRING
LCALL    UART4_SENDSTR

```

LOOP:

```

MOV      A,RPTR
XRL     A,WPTR
ANL     A,#0FH
JZ      LOOP
MOV      A,RPTR
ANL     A,#0FH
ADD     A,#BUFFER
MOV      R0,A
MOV      A,@R0
LCALL   UART4_SEND
INC     RPT
JMP     LOOP

```

STRING: **DB** *'Uart Test !',0DH,0AH,00H*

END**C 语言代码**

```

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

#define FOSC            11059200UL
#define BRT            (65536 - FOSC / 115200 / 4)

sfr     AUXR        = 0x8e;
sfr     T2H          = 0xd6;
sfr     T2L          = 0xd7;
sfr     S4CON        = 0x84;
sfr     S4BUF        = 0x85;
sfr     IE2          = 0xaf;

bit     busy;
char    wptr;
char    rptr;
char    buffer[16];

void Uart4Isr() interrupt 18
{
    if(S4CON & 0x02)
    {
        S4CON &= ~0x02;
        busy = 0;
    }
    if(S4CON & 0x01)
    {

```

```

S4CON &= ~0x01;
buffer[wptr++] = S4BUF;
wptr &= 0x0f;
}
}

void Uart4Init()
{
    S4CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart4Send(char dat)
{
    while (busy);
    busy = 1;
    S4BUF = dat;
}

void Uart4SendStr(char *p)
{
    while (*p)
    {
        Uart4Send(*p++);
    }
}

void main()
{
    Uart4Init();
    IE2 = 0x10;
    EA = 1;
    Uart4SendStr("Uart Test !r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart4Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}

```

12.6.23 定时器 3 (16 位自动重载)

汇编代码

; 测试工作频率为 11.0592MHz

T3L	DATA	0D5H
T3H	DATA	0D4H
T4T3M	DATA	0DIH
IE2	DATA	0AFH

<i>ET3</i>	<i>EQU</i>	<i>20H</i>
<i>AUXINTIF</i>	<i>DATA</i>	<i>0EFH</i>
<i>T3IF</i>	<i>EQU</i>	<i>02H</i>
	<i>ORG</i>	<i>0000H</i>
	<i>LJMP</i>	<i>MAIN</i>
	<i>ORG</i>	<i>009BH</i>
	<i>LJMP</i>	<i>TM3ISR</i>
	<i>ORG</i>	<i>0100H</i>
<i>TM3ISR:</i>		
	<i>CPL</i>	<i>P1.0</i> ; 测试端口
	<i>ANL</i>	<i>AUXINTIF,#NOT T3IF</i> ; 清中断标志
	<i>RETI</i>	
<i>MAIN:</i>		
	<i>MOV</i>	<i>SP,#3FH</i>
	<i>MOV</i>	<i>T3L,#66H</i> ;65536-11.0592M/12/1000
	<i>MOV</i>	<i>T3H,#0FCH</i>
	<i>MOV</i>	<i>T4T3M,#08H</i> ;启动定时器
	<i>MOV</i>	<i>IE2,#ET3</i> ;使能定时器中断
	<i>SETB</i>	<i>EA</i>
	<i>JMP</i>	<i>\$</i>
	<i>END</i>	

C 语言代码

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

//测试工作频率为11.0592MHz

sfr    T3L      = 0xd5;
sfr    T3H      = 0xd4;
sfr    T4T3M    = 0xd1;
sfr    IE2      = 0xaf;
#define ET3      0x20
sfr    AUXINTIF = 0xef;
#define T3IF     0x02

sbit   P10      = P1^0;

void TM3_Isr() interrupt 19
{
    P10 = !P10;          //测试端口
    AUXINTIF &= ~T3IF;    //清中断标志
}

void main()
{
    T3L = 0x66;          //65536-11.0592M/12/1000
    T3H = 0xfc;
    T4T3M = 0x08;        //启动定时器
    IE2 = ET3;           //使能定时器中断
    EA = 1;
}
```

```

    while (1);
}

```

12.6.24 定时器 3 (外部计数—扩展T3 为外部下降沿中断)

汇编代码

// 测试工作频率为 11.0592MHz

```

T3L      DATA      0D5H
T3H      DATA      0D4H
T4T3M    DATA      0DIH
IE2      DATA      0AFH
ET3      EQU       20H
AUXINTIF  DATA      0EFH
T3IF     EQU       02H

ORG      0000H
LJMP    MAIN
ORG      009BH
LJMP    TM3ISR

ORG      0100H
TM3ISR:
CPL      P1.0
ANL      AUXINTIF,#NOT T3IF      ; 测试端口
RETI

MAIN:
MOV      SP,#3FH
MOV      T3L,#0FFH
MOV      T3H,#0FFH
MOV      T4T3M,#0CH      ; 设置外部计数模式并启动定时器
MOV      IE2,#ET3      ; 使能定时器中断
SETB    EA
JMP      $

END

```

C 语言代码

```

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

```

// 测试工作频率为 11.0592MHz

```

sfr      T3L      = 0xd5;
sfr      T3H      = 0xd4;
sfr      T4T3M    = 0xd1;
sfr      IE2      = 0xaf;
#define  ET3      0x20
sfr      AUXINTIF = 0xef;
#define  T3IF     0x02

sbit     P10      = P1^0;

```

```

void TM3_Isr() interrupt 19
{
    P10 = !P10;                      //测试端口
    AUXINTIF &= ~T3IF;                //清中断标志
}

void main()
{
    T3L = 0xff;
    T3H = 0xff;
    T4T3M = 0x0c;                   //设置外部计数模式并启动定时器
    IE2 = ET3;                      //使能定时器中断
    EA = 1;

    while (1);
}

```

12.6.25 定时器 3（时钟分频输出）

汇编代码

```

;测试工作频率为 11.0592MHz

T3L      DATA    0D5H
T3H      DATA    0D4H
T4T3M   DATA    0DIH

        ORG     0000H
        LJMP   MAIN

        ORG     0100H
MAIN:   MOV     SP,#3FH
        MOV     T3L,#66H           ;65536-11.0592M/12/1000
        MOV     T3H,#0FCH
        MOV     T4T3M,#09H         ;使能时钟输出并启动定时器

        JMP     $

```

END

C 语言代码

```

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

//测试工作频率为 11.0592MHz

sfr      T3L      = 0xd5;
sfr      T3H      = 0xd4;
sfr      T4T3M   = 0xd1;

void main()
{
    T3L = 0x66;                  //65536-11.0592M/12/1000
    T3H = 0xfc;
    T4T3M = 0x09;               //使能时钟输出并启动定时器

```

```

    while (1);
}

```

12.6.26 定时器3做串口3波特率发生器

汇编代码

<i>T4T3M</i>	<i>DATA</i>	<i>0D1H</i>	
<i>T3H</i>	<i>DATA</i>	<i>0D4H</i>	
<i>T3L</i>	<i>DATA</i>	<i>0D5H</i>	
<i>S3CON</i>	<i>DATA</i>	<i>0ACh</i>	
<i>S3BUF</i>	<i>DATA</i>	<i>0ADH</i>	
<i>IE2</i>	<i>DATA</i>	<i>0AFH</i>	
<i>BUSY</i>	<i>BIT</i>	<i>20H.0</i>	
<i>WPTR</i>	<i>DATA</i>	<i>21H</i>	
<i>RPTR</i>	<i>DATA</i>	<i>22H</i>	
<i>BUFFER</i>	<i>DATA</i>	<i>23H</i>	<i>;16 bytes</i>
	<i>ORG</i>	<i>0000H</i>	
	<i>LJMP</i>	<i>MAIN</i>	
	<i>ORG</i>	<i>008BH</i>	
	<i>LJMP</i>	<i>UART3_ISR</i>	
	<i>ORG</i>	<i>0100H</i>	
UART3_ISR:			
	<i>PUSH</i>	<i>ACC</i>	
	<i>PUSH</i>	<i>PSW</i>	
	<i>MOV</i>	<i>PSW,#08H</i>	
	<i>MOV</i>	<i>A,S3CON</i>	
	<i>JNB</i>	<i>ACC.1,CHKRI</i>	
	<i>ANL</i>	<i>S3CON,#NOT 02H</i>	
	<i>CLR</i>	<i>BUSY</i>	
CHKRI:			
	<i>JNB</i>	<i>ACC.0,UART3ISR_EXIT</i>	
	<i>ANL</i>	<i>S3CON,#NOT 01H</i>	
	<i>MOV</i>	<i>A,WPTR</i>	
	<i>ANL</i>	<i>A,#0FH</i>	
	<i>ADD</i>	<i>A,#BUFFER</i>	
	<i>MOV</i>	<i>R0,A</i>	
	<i>MOV</i>	<i>@R0,S3BUF</i>	
	<i>INC</i>	<i>WPTR</i>	
UART3ISR_EXIT:			
	<i>POP</i>	<i>PSW</i>	
	<i>POP</i>	<i>ACC</i>	
	<i>RETI</i>		
UART3_INIT:			
	<i>MOV</i>	<i>S3CON,#50H</i>	
	<i>MOV</i>	<i>T3L,#0E8H</i>	<i>;65536-11059200/115200/4=0FFE8H</i>
	<i>MOV</i>	<i>T3H,#0FFH</i>	
	<i>MOV</i>	<i>T4T3M,#0AH</i>	
	<i>CLR</i>	<i>BUSY</i>	
	<i>MOV</i>	<i>WPTR,#00H</i>	
	<i>MOV</i>	<i>RPTR,#00H</i>	
	<i>RET</i>		

UART3_SEND:

<i>JB</i>	<i>BUSY,\$</i>
<i>SETB</i>	<i>BUSY</i>
<i>MOV</i>	<i>S3BUF,A</i>
<i>RET</i>	

UART3_SENDSTR:

<i>CLR</i>	<i>A</i>
<i>MOVC</i>	<i>A,@A+DPTR</i>
<i>JZ</i>	<i>SEND3END</i>
<i>LCALL</i>	<i>UART3_SEND</i>
<i>INC</i>	<i>DPTR</i>
<i>JMP</i>	<i>UART3_SENDSTR</i>

SEND3END:

	<i>RET</i>
--	------------

MAIN:

<i>MOV</i>	<i>SP,#3FH</i>
<i>LCALL</i>	<i>UART3_INIT</i>
<i>MOV</i>	<i>IE2,#08H</i>
<i>SETB</i>	<i>EA</i>
<i>MOV</i>	<i>DPTR,#STRING</i>
<i>LCALL</i>	<i>UART3_SENDSTR</i>

LOOP:

<i>MOV</i>	<i>A,RPTR</i>
<i>XRL</i>	<i>A,WPTR</i>
<i>ANL</i>	<i>A,#0FH</i>
<i>JZ</i>	<i>LOOP</i>
<i>MOV</i>	<i>A,RPTR</i>
<i>ANL</i>	<i>A,#0FH</i>
<i>ADD</i>	<i>A,#BUFFER</i>
<i>MOV</i>	<i>R0,A</i>
<i>MOV</i>	<i>A,@R0</i>
<i>LCALL</i>	<i>UART3_SEND</i>
<i>INC</i>	<i>RPTR</i>
<i>JMP</i>	<i>LOOP</i>

STRING: ***DB*** *'Uart Test !',0DH,0AH,00H*

END

C 语言代码

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

#define FOSC 11059200UL
#define BRT (65536 - FOSC / 115200 / 4)

sfr T4T3M = 0xd1;
sfr T3H = 0xd4;
sfr T3L = 0xd5;
sfr S3CON = 0xac;
sfr S3BUF = 0xad;
sfr IE2 = 0xaf;
```

```
bit      busy;
char     wptr;
char     rptr;
char     buffer[16];

void Uart3Isr() interrupt 17
{
    if(S3CON & 0x02)
    {
        S3CON &= ~0x02;
        busy = 0;
    }
    if(S3CON & 0x01)
    {
        S3CON &= ~0x01;
        buffer[wptr++] = S3BUF;
        wptr &= 0x0f;
    }
}

void Uart3Init()
{
    S3CON = 0x50;
    T3L = BRT;
    T3H = BRT >> 8;
    T4T3M = 0x0a;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart3Send(char dat)
{
    while (busy);
    busy = 1;
    S3BUF = dat;
}

void Uart3SendStr(char *p)
{
    while (*p)
    {
        Uart3Send(*p++);
    }
}

void main()
{
    Uart3Init();
    IE2 = 0x08;
    EA = 1;
    Uart3SendStr("Uart Test !\r\n");

    while (1)
    {
        if(rptr != wptr)
        {
            Uart3Send(buffer[rptr++]);
        }
    }
}
```

```

        rptr &= 0x0f;
    }
}
}

```

12.6.27 定时器 4 (16 位自动重载)

汇编代码

//测试工作频率为 11.0592MHz

```

T4L      DATA    0D3H
T4H      DATA    0D2H
T4T3M    DATA    0D1H
IE2      DATA    0AFH
ET4      EQU     40H
AUXINTIF DATA    0EFH
T4IF     EQU     04H

ORG      0000H
LJMP    MAIN
ORG      00A3H
LJMP    TM4ISR

ORG      0100H
TM4ISR:
CPL      P1.0      ; 测试端口
ANL      AUXINTIF, #NOT T4IF ; 清中断标志
RETI

MAIN:
MOV      SP, #3FH
MOV      T4L, #66H      ; 65536-11.0592M/12/1000
MOV      T4H, #0FCH
MOV      T4T3M, #80H    ; 启动定时器
MOV      IE2, #ET4      ; 使能定时器中断
SETB    EA

JMP      $

END

```

C 语言代码

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

//测试工作频率为 11.0592MHz

```
sfr    T4L      = 0xd3;
sfr    T4H      = 0xd2;
sfr    T4T3M    = 0xd1;
sfr    IE2      = 0xaf;
#define ET4      0x40
sfr    AUXINTIF = 0xef;
#define T4IF     0x04
```

```

sbit      P10      =  P1^0;

void TM4_Isr() interrupt 20
{
    P10 = !P10;           //测试端口
    AUXINTIF &= ~T4IF;   //清中断标志
}

void main()
{
    T4L = 0x66;          //65536-11.0592M/12/1000
    T4H = 0xfc;
    T4T3M = 0x80;        //启动定时器
    IE2 = ET4;           //使能定时器中断
    EA = 1;

    while (1);
}

```

12.6.28 定时器 4 (外部计数—扩展T4 为外部下降沿中断)

汇编代码

; 测试工作频率为 11.0592MHz

```

T4L      DATA    0D3H
T4H      DATA    0D2H
T4T3M   DATA    0D1H
IE2      DATA    0AFH
ET4      EQU     40H
AUXINTIF DATA    0EFH
T4IF    EQU     04H

ORG      0000H
LJMP    MAIN
ORG      00A3H
LJMP    TM4ISR

ORG      0100H

TM4ISR:
    CPL      P1.0           ; 测试端口
    ANL      AUXINTIF,#NOT T4IF ; 清中断标志
    RETI

MAIN:
    MOV      SP,#3FH

    MOV      T4L,#0FFH
    MOV      T4H,#0FFH
    MOV      T4T3M,#0C0H      ; 设置外部计数模式并启动定时器
    MOV      IE2,#ET4         ; 使能定时器中断
    SETB    EA

    JMP      $

END

```

C 语言代码

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

//测试工作频率为 11.0592MHz

sfr T4L = 0xd3;
sfr T4H = 0xd2;
sfr T4T3M = 0xd1;
sfr IE2 = 0xaf;
#define ET4 0x40
sfr AUXINTIF = 0xef;
#define T4IF 0x04

sbit P10 = P1^0;

void TM4_Isr() interrupt 20
{
    P10 = !P10; //测试端口
    AUXINTIF &= ~T4IF; //清中断标志
}

void main()
{
    T4L = 0xff;
    T4H = 0xff;
    T4T3M = 0xc0; //设置外部计数模式并启动定时器
    IE2 = ET4; //使能定时器中断
    EA = 1;

    while (1);
}
```

12.6.29 定时器 4 (时钟分频输出)

汇编代码

```
;测试工作频率为 11.0592MHz

T4L DATA 0D3H
T4H DATA 0D2H
T4T3M DATA 0D1H

ORG 0000H
LJMP MAIN

ORG 0100H
MAIN:
MOV SP,#3FH
MOV T4L,#66H ;65536-11.0592M/12/1000
MOV T4H,#0FCH
MOV T4T3M,#90H ;使能时钟输出并启动定时器
JMP $
END
```

C 语言代码

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

//测试工作频率为 11.0592MHz

sfr      T4L          = 0xd3;
sfr      T4H          = 0xd2;
sfr      T4T3M        = 0xd1;

void main()
{
    T4L = 0x66;           //65536-11.0592M/12/1000
    T4H = 0xfc;
    T4T3M = 0x90;        //使能时钟输出并启动定时器

    while (1);
}
```

12.6.30 定时器 4 做串口 4 波特率发生器

汇编代码

T4T3M	DATA	0D1H
T4H	DATA	0D2H
T4L	DATA	0D3H
S4CON	DATA	84H
S4BUF	DATA	085H
IE2	DATA	0AFH
BUSY	BIT	20H.0
WPTR	DATA	21H
RPTR	DATA	22H
BUFFER	DATA	23H
		<i>;16 bytes</i>
ORG		0000H
LJMP		MAIN
ORG		0093H
LJMP		UART4_ISR
ORG		0100H
UART4_ISR:		
PUSH		ACC
PUSH		PSW
MOV		PSW,#08H
MOV		A,S4CON
JNB		ACC.1,CHKRI
ANL		S4CON,#NOT 02H
CLR		BUSY
CHKRI:		
JNB		ACC.0,UART4ISR_EXIT
ANL		S4CON,#NOT 01H
MOV		A,WPTR
ANL		A,#0FH
ADD		A,#BUFFER
MOV		R0,A

```

        MOV      @R0,S4BUF
        INC      WPTR
UART4ISR_EXIT:
        POP      PSW
        POP      ACC
        RETI

UART4_INIT:
        MOV      S4CON,#50H
        MOV      T4L,#0E8H           ;65536-11059200/115200/4=0FFE8H
        MOV      T4H,#0FFH
        MOV      T4T3M,#0A0H
        CLR      BUSY
        MOV      WPTR,#00H
        MOV      RPTR,#00H
        RET

UART4_SEND:
        JB      BUSY,$
        SETB    BUSY
        MOV      S4BUFA,A
        RET

UART4_SENDSTR:
        CLR      A
        MOVC   A,@A+DPTR
        JZ      SEND4END
        LCALL  UART4_SEND
        INC     DPTR
        JMP     UART4_SENDSTR
SEND4END:
        RET

MAIN:
        MOV      SP,#3FH
        LCALL  UART4_INIT
        MOV      IE2,#10H
        SETB    EA
        MOV      DPTR,#STRING
        LCALL  UART4_SENDSTR

LOOP:
        MOV      A,RPTR
        XRL      A,WPTR
        ANL      A,#0FH
        JZ      LOOP
        MOV      A,RPTR
        ANL      A,#0FH
        ADD      A,#BUFFER
        MOV      R0,A
        MOV      A,@R0
        LCALL  UART4_SEND
        INC     RPTR
        JMP     LOOP

STRING:    DB      'Uart Test !',0DH,0AH,00H

```

END

C 语言代码

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

#define FOSC          11059200UL
#define BRT           (65536 - FOSC / 115200 / 4)

sfr T4T3M = 0xd1;
sfr T4H   = 0xd2;
sfr T4L   = 0xd3;
sfr S4CON = 0x84;
sfr S4BUF = 0x85;
sfr IE2   = 0xaf;

bit busy;
char wptr;
char rptr;
char buffer[16];

void Uart4Isr() interrupt 18
{
    if(S4CON & 0x02)
    {
        S4CON &= ~0x02;
        busy = 0;
    }
    if(S4CON & 0x01)
    {
        S4CON &= ~0x01;
        buffer[wptr++] = S4BUF;
        wptr &= 0x0f;
    }
}

void Uart4Init()
{
    S4CON = 0x50;
    T4L = BRT;
    T4H = BRT >> 8;
    T4T3M = 0xa0;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart4Send(char dat)
{
    while (busy);
    busy = 1;
    S4BUF = dat;
}

void Uart4SendStr(char *p)
{
    while (*p)
    {
```

```
Uart4Send(*p++);
}

}

void main()
{
    Uart4Init();
    IE2 = 0x10;
    EA = 1;
    Uart4SendStr("Uart Test !r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart4Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

13 串口通信

STC8 系列单片机具有 4 个全双工异步串行通信接口(串口 1、串口 2、串口 3 和串口 4)。每个串行口由 2 个数据缓冲器、一个移位寄存器、一个串行控制寄存器和一个波特率发生器等组成。每个串行口的数据缓冲器由 2 个互相独立的接收、发送缓冲器构成，可以同时发送和接收数据。

STC8 系列单片机的串口 1 有 4 种工作方式，其中两种方式的波特率是可变的，另两种是固定的，以供不同应用场合选用。串口 2/串口 3/串口 4 都只有两种工作方式，这两种方式的波特率都是可变的。用户可用软件设置不同的波特率和选择不同的工作方式。主机可通过查询或中断方式对接收/发送进行程序处理，使用十分灵活。

串口 1、串口 2、串口 3、串口 4 的通讯口均可以通过功能管脚的切换功能切换到多组端口，从而可以将一个通讯口分时复用为多个通讯口。

13.1 串口相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
SCON	串口 1 控制寄存器	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
SBUF	串口 1 数据寄存器	99H									0000,0000
S2CON	串口 2 控制寄存器	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0100,0000
S2BUF	串口 2 数据寄存器	9BH									0000,0000
S3CON	串口 3 控制寄存器	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000
S3BUF	串口 3 数据寄存器	ADH									0000,0000
S4CON	串口 4 控制寄存器	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000,0000
S4BUF	串口 4 数据寄存器	85H									0000,0000
PCON	电源控制寄存器	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
AUXR	辅助寄存器 1	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000,0001
SADDR	串口 1 从机地址寄存器	A9H									0000,0000
SADEN	串口 1 从机地址屏蔽寄存器	B9H									0000,0000

13.2 串口 1

串口 1 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SCON	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI

SM0/FE: 当PCON寄存器中的SMOD0位为1时, 该位为帧错误检测标志位。当UART在接收过程中检测到一个无效停止位时, 通过UART接收器将该位置1, 必须由软件清零。当PCON寄存器中的SMOD0位为0时, 该位和SM1一起指定串口1的通信工作模式, 如下表所示:

SM0	SM1	串口1工作模式	功能说明
0	0	模式0	同步移位串行方式
0	1	模式1	可变波特率8位数据方式
1	0	模式2	固定波特率9位数据方式
1	1	模式3	可变波特率9位数据方式

SM2: 允许模式2或模式3多机通信控制位。当串口1使用模式2或模式3时, 如果SM2位为1且REN位为1, 则接收机处于地址帧筛选状态。此时可以利用接收到的第9位(即RB8)来筛选地址帧, 若RB8=1, 说明该帧是地址帧, 地址信息可以进入SBUF, 并使RI为1, 进而在中断服务程序中再进行地址号比较; 若RB8=0, 说明该帧不是地址帧, 应丢掉且保持RI=0。在模式2或模式3中, 如果SM2位为0且REN位为1, 接收机处于地址帧筛选被禁止状态, 不论收到的RB8为0或1, 均可使接收到的信息进入SBUF, 并使RI=1, 此时RB8通常为校验位。模式1和模式0为非多机通信方式, 在这两种方式时, SM2应设置为0。

REN: 允许/禁止串口接收控制位

- 0: 禁止串口接收数据
- 1: 允许串口接收数据

TB8: 当串口1使用模式2或模式3时, TB8为要发送的第9位数据, 按需要由软件置位或清0。在模式0和模式1中, 该位不用。

RB8: 当串口1使用模式2或模式3时, RB8为接收到的第9位数据, 一般用作校验位或者地址帧/数据帧标志位。在模式0和模式1中, 该位不用。

TI: 串口1发送中断请求标志位。在模式0中, 当串口发送数据第8位结束时, 由硬件自动将TI置1, 向主机请求中断, 响应中断后TI必须用软件清零。在其他模式中, 则在停止位开始发送时由硬件自动将TI置1, 向CPU发请求中断, 响应中断后TI必须用软件清零。

RI: 串口1接收中断请求标志位。在模式0中, 当串口接收第8位数据结束时, 由硬件自动将RI置1, 向主机请求中断, 响应中断后RI必须用软件清零。在其他模式中, 串行接收到停止位的中间时刻由硬件自动将RI置1, 向CPU发中断申请, 响应中断后RI必须由软件清零。

串口 1 数据寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SBUF	99H								

SBUF: 串口1数据接收/发送缓冲区。SBUF实际是2个缓冲器, 读缓冲器和写缓冲器, 两个操作分别对应两个不同的寄存器, 1个是只写寄存器(写缓冲器), 1个是只读寄存器(读缓冲器)。对SBUF进行读操作, 实际是读取串口接收缓冲区, 对SBUF进行写操作则是触发串口开始发送数据。

电源管理寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PCON	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL

SMOD: 串口 1 波特率控制位

0: 串口 1 的各个模式的波特率都不加倍

1: 串口 1 模式 1、模式 2、模式 3 的波特率加倍

SMOD0: 帧错误检测控制位

0: 无帧错检测功能

1: 使能帧错误检测功能。此时 SCON 的 SM0/FE 为 FE 功能，即为帧错误检测标志位。

辅助寄存器 1

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

UART_M0x6: 串口 1 模式 0 的通讯速度控制

0: 串口 1 模式 0 的波特率不加倍，固定为 Fosc/12

1: 串口 1 模式 0 的波特率 6 倍速，即固定为 Fosc/12*6 = Fosc/2

S1ST2: 串口 1 波特率发射器选择位

0: 选择定时器 1 作为波特率发射器

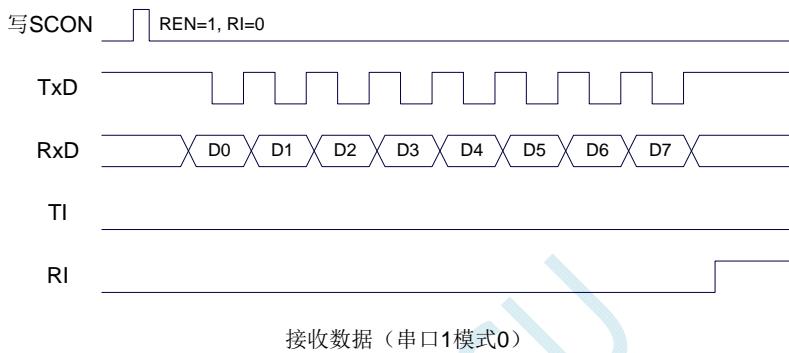
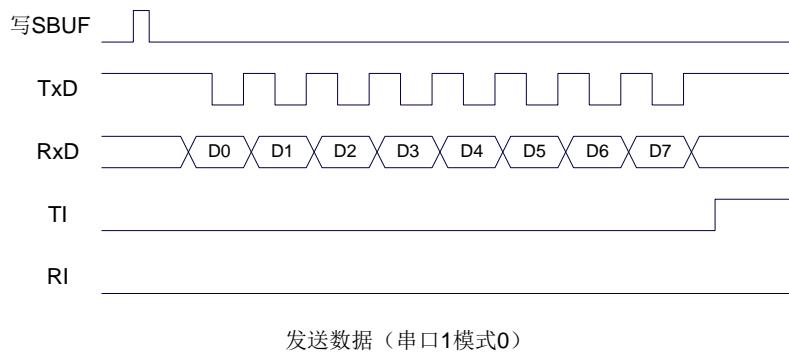
1: 选择定时器 2 作为波特率发射器

13.2.1 串口 1 模式 0

当串口 1 选择工作模式为模式 0 时，串行通信接口工作在同步移位寄存器模式，当串行口模式 0 的通信速度设置位 UART_M0x6 为 0 时，其波特率固定为系统时钟时钟的 12 分频 (SYSclk/12)；当设置 UART_M0x6 为 1 时，其波特率固定为系统时钟频率的 2 分频 (SYSclk/2)。RxD 为串行通讯的数据口，TxD 为同步移位脉冲输出脚，发送、接收的是 8 位数据，低位在先。

模式 0 的发送过程：当主机执行将数据写入发送缓冲器 SBUF 指令时启动发送，串行口即将 8 位数据以 SYSclk/12 或 SYSclk/2 (由 UART_M0x6 确定是 12 分频还是 2 分频) 的波特率从 RxD 管脚输出(从低位到高位)，发送完中断标志 TI 置 1，TxD 管脚输出同步移位脉冲信号。当写信号有效后，相隔一个时钟，发送控制端 SEND 有效(高电平)，允许 RxD 发送数据，同时允许 TxD 输出同步移位脉冲。一帧(8 位)数据发送完毕时，各控制端均恢复原状态，只有 TI 保持高电平，呈中断申请状态。在再次发送数据前，必须用软件将 TI 清 0。

模式 0 的接收过程：首先将接收中断请求标志 RI 清零并置位允许接收控制位 REN 时启动模式 0 接收过程。启动接收过程后，RxD 为串行数据输入端，TxD 为同步脉冲输出端。串行接收的波特率为 SYSclk/12 或 SYSclk/2 (由 UART_M0x6 确定是 12 分频还是 2 分频)。当接收完成一帧数据 (8 位) 后，控制信号复位，中断标志 RI 被置 1，呈中断申请状态。当再次接收时，必须通过软件将 RI 清 0



工作于模式 0 时，必须清 0 多机通信控制位 SM2，使之不影响 TB8 位和 RB8 位。由于波特率固定为 SYScclk/12 或 SYScclk/2，无需定时器提供，直接由单片机的时钟作为同步移位脉冲。

串口 1 模式 0 的波特率计算公式如下表所示 (SYScclk 为系统工作频率)：

UART_M0x6	波特率计算公式
0	波特率 = $\frac{\text{SYScclk}}{12}$
1	波特率 = $\frac{\text{SYScclk}}{2}$

13.2.2 串口 1 模式 1

当软件设置 SCON 的 SM0、SM1 为“01”时，串行口 1 则以模式 1 进行工作。此模式为 8 位 UART 格式，一帧信息为 10 位：1 位起始位，8 位数据位（低位在先）和 1 位停止位。波特率可变，即可根据需要进行设置波特率。TxD 为数据发送口，RxD 为数据接收口，串行口全双工接受/发送。

模式 1 的发送过程：串行通信模式发送时，数据由串行发送端 TxD 输出。当主机执行一条写 SBUF 的指令就启动串行通信的发送，写“SBUF”信号还把“1”装入发送移位寄存器的第 9 位，并通知 TX 控制单元开始发送。移位寄存器将数据不断右移送 TxD 端口发送，在数据的左边不断移入“0”作补充。当数据的最高位移到移位寄存器的输出位置，紧跟其后的是第 9 位“1”，在它的左边各位全为“0”，这个状态条件，使 TX 控制单元作最后一次移位输出，然后使允许发送信号“SEND”失效，完成一帧信息的发送，并置位中断请求位 TI，即 TI=1，向主机请求中断处理。

模式 1 的接收过程: 当软件置位接收允许标志位 REN, 即 REN=1 时, 接收器便对 RxD 端口的信号进行检测, 当检测到 RxD 端口发送从“1”→“0”的下降沿跳变时就启动接收器准备接收数据, 并立即复位波特率发生器的接收计数器, 将 1FFH 装入移位寄存器。接收的数据从接收移位寄存器的右边移入, 已装入的 1FFH 向左边移出, 当起始位“0”移到移位寄存器的最左边时, 使 RX 控制器作最后一次移位, 完成一帧的接收。若同时满足以下两个条件:

- RI=0;
- SM2=0 或接收到的停止位为 1。

则接收到的数据有效, 实现装载入 SBUF, 停止位进入 RB8, RI 标志位被置 1, 向主机请求中断, 若上述两条件不能同时满足, 则接收到的数据作废并丢失, 无论条件满足与否, 接收器重又检测 RxD 端口上的“1”→“0”的跳变, 继续下一帧的接收。接收有效, 在响应中断后, RI 标志位必须由软件清 0。通常情况下, 串行通信工作于模式 1 时, SM2 设置为“0”。



串口 1 的波特率是可变的, 其波特率可由定时器 1 或者定时器 2 产生。当定时器采用 1T 模式时(12 倍速), 相应的波特率的速度也会相应提高 12 倍。

串口 1 模式 1 的波特率计算公式如下表所示: (SYSclk 为系统工作频率)

选择定时器	定时器速度	波特率计算公式
定时器2	1T	定时器2重载值 = $65536 - \frac{SYSclk}{4 \times \text{波特率}}$
	12T	定时器2重载值 = $65536 - \frac{SYSclk}{12 \times 4 \times \text{波特率}}$
定时器1模式0	1T	定时器1重载值 = $65536 - \frac{SYSclk}{4 \times \text{波特率}}$
	12T	定时器1重载值 = $65536 - \frac{SYSclk}{12 \times 4 \times \text{波特率}}$
定时器1模式2	1T	定时器1重载值 = $256 - \frac{2^{\text{SMOD}} \times SYSclk}{32 \times \text{波特率}}$
	12T	定时器1重载值 = $256 - \frac{2^{\text{SMOD}} \times SYSclk}{12 \times 32 \times \text{波特率}}$

下面为常用频率与常用波特率所对应定时器的重载值

频率 (MHz)	波特率	定时器 2		定时器 1 模式 0		定时器 1 模式 2			
		1T 模式	12T 模式	1T 模式	12T 模式	SMOD=1		SMOD=0	
						1T 模式	12T 模式	1T 模式	12T 模式
11.0592	115200	FFE8H	FFFEH	FFE8H	FFFEH	FAH	-	FDH	-
	57600	FFD0H	FFFCH	FFD0H	FFFCH	F4H	FFH	FAH	-
	38400	FFB8H	FFFAH	FFB8H	FFFAH	EEH	-	F7H	-
	19200	FF70H	FFF4H	FF70H	FFF4H	DCH	FDH	EEH	-
	9600	FEE0H	FFE8H	FEE0H	FFE8H	B8H	FAH	DCH	FDH
18.432	115200	FFD8H	-	FFD8H	-	F6H	-	FBH	-
	57600	FFB0H	-	FFB0H	-	ECH	-	F6H	-
	38400	FF88H	FFF6H	FF88H	FFF6H	E2H	-	F1H	-
	19200	FF10H	FFECH	FF10H	FFECH	C4H	FBH	E2H	-
	9600	FE20H	FFD8H	FE20H	FFD8H	88H	F6H	C4H	FBH
22.1184	115200	FFD0H	FFFCH	FFD0H	FFFCH	F4H	FFH	FAH	-
	57600	FFA0H	FFF8H	FFA0H	FFF8H	E8H	FEH	F4H	FFH
	38400	FF70H	FFF4H	FF70H	FFF4H	DCH	FDH	EEH	-
	19200	FEE0H	FFE8H	FEE0H	FFE8H	B8H	FAH	DCH	FDH
	9600	FDC0H	FFD0H	FDC0H	FFD0H	70H	F4H	B8H	FAH

13.2.3 串口 1 模式 2

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

模式 2 的波特率固定为系统时钟的 64 分频或 32 分频（取决于 PCON 中 SMOD 的值）

串口 1 模式 2 的波特率计算公式如下表所示（SYSclk 为系统工作频率）：

SMOD	波特率计算公式
0	波特率 = $\frac{\text{SYSclk}}{64}$
1	波特率 = $\frac{\text{SYSclk}}{32}$

模式 2 和模式 1 相比，除波特率发生源略有不同，发送时由 TB8 提供给移位寄存器第 9 数据位不同外，其余功能结构均基本相同，其接收/发送操作过程及时序也基本相同。

当接收器接收完一帧信息后必须同时满足下列条件：

- RI=0
- SM2=0 或者 SM2=1 且接收到的第 9 数据位 RB8=1。

当上述两条件同时满足时，才将接收到的移位寄存器的数据装入 SBUF 和 RB8 中，RI 标志位被置 1，并向主机请求中断处理。如果上述条件有一个不满足，则刚接收到移位寄存器中的数据无效而丢失，也不置位 RI。无论上述条件满足与否，接收器又重新开始检测 Rx 为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 D 协议的约定，为多机通信提供了方便。



13.2.4 串口 1 模式 3

当 SM0、SM1 两位为 11 时，串行口 1 工作在模式 3。串行通信模式 3 为 9 位数据异步通信 UART

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

模式 3 和模式 1 相比，除发送时由 TB8 提供给移位寄存器第 9 数据位不同外，其余功能结构均基本相同，其接收‘发送操作过程及时序也基本相同。

当接收器接收完一帧信息后必须同时满足下列条件：

- RI=0
- SM2=0 或者 SM2=1 且接收到的第 9 数据位 RB8=1。

当上述两条件同时满足时，才将接收到的移位寄存器的数据装入 SBUF 和 RB8 中，RI 标志位被置 1，并向主机请求中断处理。如果上述条件有一个不满足，则刚接收到移位寄存器中的数据无效而丢失，也不置位 RI。无论上述条件满足与否，接收器又重新开始检测 RxD 输入端口的跳变信息，接收下一帧的输入信息。在模式 3 中，接收到的停止位与 SBUF、RB8 和 RI 无关。

通过软件对 SCON 中的 SM2、TB8 的设置以及通信协议的约定，为多机通信提供了方便。



串口 1 模式 3 的波特率计算公式与模式 1 是完全相同的。请参考模式 1 的波特率计算公式。

13.2.5 自动地址识别

串口 1 从机地址控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SADDR	A9H								
SADEN	B9H								

SADDR: 从机地址寄存器

SADEN: 从机地址屏蔽位寄存器

自动地址识别功能典型应用在多机通讯领域，其主要原理是从机系统通过硬件比较功能来识别来自于主机串口数据流中的地址信息，通过寄存器 SADDR 和 SADEN 设置的本机的从机地址，硬件自动对从机地址进行过滤，当来自于主机的从机地址信息与本机所设置的从机地址相匹配时，硬件产生串口中

断; 否则硬件自动丢弃串口数据, 而不产生中断。当众多处于空闲模式的从机链接在一起时, 只有从机地址相匹配的从机才会从空闲模式唤醒, 从而可以大大降低从机 MCU 的功耗, 即使从机处于正常工作状态也可避免不停地进入串口中断而降低系统执行效率。

要使用串口的自动地址识别功能, 首先需要将参与通讯的 MCU 的串口通讯模式设置为模式 2 或者模式 3 (通常都选择波特率可变的模式 3, 因为模式 2 的波特率是固定的, 不便于调节), 并开启从机的 SCON 的 SM2 位。对于串口模式 2 或者模式 3 的 9 位数据位中, 第 9 位数据 (存放在 RB8 中) 为地址/数据的标志位, 当第 9 位数据为 1 时, 表示前面的 8 位数据 (存放在 SBUF 中) 为地址信息。当 SM2 被设置为 1 时, 从机 MCU 会自动过滤掉非地址数据 (第 9 位为 0 的数据), 而对 SBUF 中的地址数据 (第 9 位为 1 的数据) 自动与 SADDR 和 SALEN 所设置的本机地址进行比较, 若地址相匹配, 则会将 RI 置 “1”, 并产生中断, 否则不予处理本次接收的串口数据。

从机地址的设置是通过 SADDR 和 SALEN 两个寄存器进行设置的。SADDR 为从机地址寄存器, 里面存放本机的从机地址。SALEN 为从机地址屏蔽位寄存器, 用于设置地址信息中的忽略位, 设置方法如下:

例如

SADDR = 11001010

SALEN = 10000001

则匹配地址为 1xxxxxx0

即, 只要主机送出的地址数据中的 bit0 为 0 且 bit7 为 1 就可以和本机地址相匹配

再例如

SADDR = 11001010

SALEN = 00001111

则匹配地址为 xxxx1010

即, 只要主机送出的地址数据中的低 4 位为 1010 就可以和本机地址相匹配, 而高 4 为被忽略, 可以为任意值。

主机可以使用广播地址 (FFH) 同时选中所有的从机来进行通讯。

13.3 串口 2

串口 2 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S2CON	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI

S2SM0: 指定串口2的通信工作模式, 如下表所示:

S2SM0	串口2工作模式	功能说明
0	模式0	可变波特率8位数据方式
1	模式1	可变波特率9位数据方式

S2SM2: 允许串口 2 在模式 1 时允许多机通信控制位。在模式 1 时, 如果 S2SM2 位为 1 且 S2REN 位为 1, 则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位 (即 S2RB8) 来筛选地址帧:

若 S2RB8=1, 说明该帧是地址帧, 地址信息可以进入 S2BUF, 并使 S2RI 为 1, 进而在中断服务程序中再进行地址号比较; 若 S2RB8=0, 说明该帧不是地址帧, 应丢掉且保持 S2RI=0。在模式 1 中, 如果 S2SM2 位为 0 且 S2REN 位为 1, 接收机处于地址帧筛选被禁止状态。不论收到的 S2RB8 为 0 或 1, 均可使接收到的信息进入 S2BUF, 并使 S2RI=1, 此时 S2RB8 通常为校验位。

模式 0 为非多机通信方式, 在这种方式时, 要设置 S2SM2 应为 0。

S2REN: 允许/禁止串口接收控制位

0: 禁止串口接收数据

1: 允许串口接收数据

S2TB8: 当串口 2 使用模式 1 时, S2TB8 为要发送的第 9 位数据, 一般用作校验位或者地址帧/数据帧标志位, 按需要由软件置位或清 0。在模式 0 中, 该位不用。

S2RB8: 当串口 2 使用模式 1 时, S2RB8 为接收到的第 9 位数据, 一般用作校验位或者地址帧/数据帧标志位。在模式 0 中, 该位不用。

S2TI: 串口 2 发送中断请求标志位。在停止位开始发送时由硬件自动将 S2TI 置 1, 向 CPU 发请求中断, 响应中断后 S2TI 必须用软件清零。

S2RI: 串口 2 接收中断请求标志位。串行接收到停止位的中间时刻由硬件自动将 S2RI 置 1, 向 CPU 发中断申请, 响应中断后 S2RI 必须由软件清零。

串口 2 数据寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S2BUF	9BH								

S2BUF: 串口 1 数据接收/发送缓冲区。S2BUF 实际是 2 个缓冲器, 读缓冲器和写缓冲器, 两个操作分别对应两个不同的寄存器, 1 个是只写寄存器 (写缓冲器), 1 个是只读寄存器 (读缓冲器)。对 S2BUF 进行读操作, 实际是读取串口接收缓冲区, 对 S2BUF 进行写操作则是触发串口开始发送数据。

13.3.1 串口 2 模式 0

串行口 2 的模式 0 为 8 位数据位可变波特率 UART 工作模式。此模式一帧信息为 10 位: 1 位起始位, 8 位数据位 (低位在先) 和 1 位停止位。波特率可变, 可根据需要进行设置波特率。TxD2 为数据发送口, RxD2 为数据接收口, 串行口全双工接受/发送。



串口 2 的波特率是可变的，其波特率由定时器 2 产生。当定时器采用 1T 模式时（12 倍速），相应的波特率的速度也会相应提高 12 倍。

串口 2 模式 0 的波特率计算公式如下表所示：(SYSclk 为系统工作频率)

选择定时器	定时器速度	波特率计算公式
定时器2	1T	定时器2重载值 = $65536 - \frac{SYSclk}{4 \times \text{波特率}}$
	12T	定时器2重载值 = $65536 - \frac{SYSclk}{12 \times 4 \times \text{波特率}}$

13.3.2 串口 2 模式 1

串行口 2 的模式 1 为 9 位数据位可变波特率 UART 工作模式。此模式一帧信息为 11 位：1 位起始位，9 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD2 为数据发送口，RxD2 为数据接收口，串行口全双工接受/发送。



串口 2 模式 1 的波特率计算公式与模式 0 是完全相同的。请参考模式 0 的波特率计算公式。

STCMCU

13.4 串口 3

串口 3 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S3CON	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI

S3SM0: 指定串口3的通信工作模式, 如下表所示:

S3SM0	串口3工作模式	功能说明
0	模式0	可变波特率8位数据方式
1	模式1	可变波特率9位数据方式

S3ST3: 选择串口 3 的波特率发生器

- 0: 选择定时器 2 为串口 3 的波特率发生器
- 1: 选择定时器 3 为串口 3 的波特率发生器

S3SM2: 允许串口 3 在模式 1 时允许多机通信控制位。在模式 1 时, 如果 S3SM2 位为 1 且 S3REN 位为

1, 则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位 (即 S3RB8) 来筛选地址帧:
若 S3RB8=1, 说明该帧是地址帧, 地址信息可以进入 S3BUF, 并使 S3RI 为 1, 进而在中断服务
程序中再进行地址号比较; 若 S3RB8=0, 说明该帧不是地址帧, 应丢掉且保持 S3RI=0。在模式
1 中, 如果 S3SM2 位为 0 且 S3REN 位为 1, 接收机处于地址帧筛选被禁止状态。不论收到的
S3RB8 为 0 或 1, 均可使接收到的信息进入 S3BUF, 并使 S3RI=1, 此时 S3RB8 通常为校验位。
模式 0 为非多机通信方式, 在这种方式时, 要设置 S3SM2 应为 0。

S3REN: 允许/禁止串口接收控制位

- 0: 禁止串口接收数据
- 1: 允许串口接收数据

S3TB8: 当串口 3 使用模式 1 时, S3TB8 为要发送的第 9 位数据, 一般用作校验位或者地址帧/数据帧
标志位, 按需要由软件置位或清 0。在模式 0 中, 该位不用。

S3RB8: 当串口 3 使用模式 1 时, S3RB8 为接收到的第 9 位数据, 一般用作校验位或者地址帧/数据帧
标志位。在模式 0 中, 该位不用。

S3TI: 串口 3 发送中断请求标志位。在停止位开始发送时由硬件自动将 S3TI 置 1, 向 CPU 发请求中断,
响应中断后 S3TI 必须用软件清零。

S3RI: 串口 3 接收中断请求标志位。串行接收到停止位的中间时刻由硬件自动将 S3RI 置 1, 向 CPU 发
中断申请, 响应中断后 S3RI 必须由软件清零。

串口 3 数据寄存器

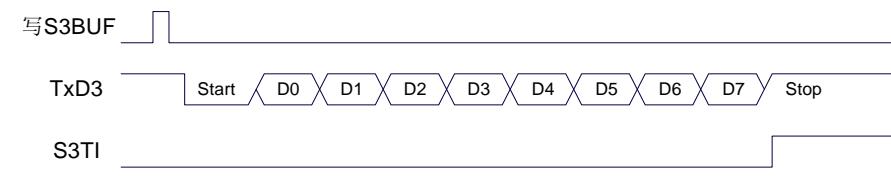
符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S3BUF	ADH								

S3BUF: 串口 1 数据接收/发送缓冲区。S3BUF 实际是 2 个缓冲器, 读缓冲器和写缓冲器, 两个操作分
别对应两个不同的寄存器, 1 个是只写寄存器 (写缓冲器), 1 个是只读寄存器 (读缓冲器)。对
S3BUF 进行读操作, 实际是读取串口接收缓冲区, 对 S3BUF 进行写操作则是触发串口开始发送
数据。

13.4.1 串口 3 模式 0

串行口 3 的模式 0 为 8 位数据位可变波特率 UART 工作模式。此模式一帧信息为 10 位: 1 位起始位, 8 位数据位 (低位在先) 和 1 位停止位。波特率可变, 可根据需要进行设置波特率。TxD3 为数据

发送口, RxD3 为数据接收口, 串行口全双工接受/发送。



串口 3 的波特率是可变的, 其波特率可由定时器 2 或定时器 3 产生。当定时器采用 1T 模式时 (12 倍速), 相应的波特率的速度也会相应提高 12 倍。

串口 3 模式 0 的波特率计算公式如下表所示: (SYSclk 为系统工作频率)

选择定时器	定时器速度	波特率计算公式
定时器2	1T	定时器2重载值 = $65536 - \frac{SYSclk}{4 \times \text{波特率}}$
	12T	定时器2重载值 = $65536 - \frac{SYSclk}{12 \times 4 \times \text{波特率}}$
定时器3	1T	定时器3重载值 = $65536 - \frac{SYSclk}{4 \times \text{波特率}}$
	12T	定时器3重载值 = $65536 - \frac{SYSclk}{12 \times 4 \times \text{波特率}}$

13.4.2 串口 3 模式 1

串行口 3 的模式 1 为 9 位数据位可变波特率 UART 工作模式。此模式一帧信息为 11 位: 1 位起始位, 9 位数据位 (低位在先) 和 1 位停止位。波特率可变, 可根据需要进行设置波特率。TxD3 为数据发送口, RxD3 为数据接收口, 串行口全双工接受/发送。





串口 3 模式 1 的波特率计算公式与模式 0 是完全相同的。请参考模式 0 的波特率计算公式。

13.5 串口 4

串口 4 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S4CON	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI

S4SM0: 指定串口4的通信工作模式, 如下表所示:

S4SM0	串口4工作模式	功能说明
0	模式0	可变波特率8位数据方式
1	模式1	可变波特率9位数据方式

S4ST4: 选择串口 4 的波特率发生器

- 0: 选择定时器 2 为串口 4 的波特率发生器
- 1: 选择定时器 4 为串口 4 的波特率发生器

S4SM2: 允许串口 4 在模式 1 时允许多机通信控制位。在模式 1 时, 如果 S4SM2 位为 1 且 S4REN 位为 1, 则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位 (即 S4RB8) 来筛选地址帧:

若 S4RB8=1, 说明该帧是地址帧, 地址信息可以进入 S4BUF, 并使 S4RI 为 1, 进而在中断服务程序中再进行地址号比较; 若 S4RB8=0, 说明该帧不是地址帧, 应丢掉且保持 S4RI=0。在模式 1 中, 如果 S4SM2 位为 0 且 S4REN 位为 1, 接收机处于地址帧筛选被禁止状态。不论收到的 S4RB8 为 0 或 1, 均可使接收到的信息进入 S4BUF, 并使 S4RI=1, 此时 S4RB8 通常为校验位。

模式 0 为非多机通信方式, 在这种方式时, 要设置 S4SM2 应为 0。

S4REN: 允许/禁止串口接收控制位

- 0: 禁止串口接收数据
- 1: 允许串口接收数据

S4TB8: 当串口 4 使用模式 1 时, S4TB8 为要发送的第 9 位数据, 一般用作校验位或者地址帧/数据帧标志位, 按需要由软件置位或清 0。在模式 0 中, 该位不用。

S4RB8: 当串口 4 使用模式 1 时, S4RB8 为接收到的第 9 位数据, 一般用作校验位或者地址帧/数据帧标志位。在模式 0 中, 该位不用。

S4TI: 串口 4 发送中断请求标志位。在停止位开始发送时由硬件自动将 S4TI 置 1, 向 CPU 发请求中断, 响应中断后 S4TI 必须用软件清零。

S4RI: 串口 4 接收中断请求标志位。串行接收到停止位的中间时刻由硬件自动将 S4RI 置 1, 向 CPU 发中断申请, 响应中断后 S4RI 必须由软件清零。

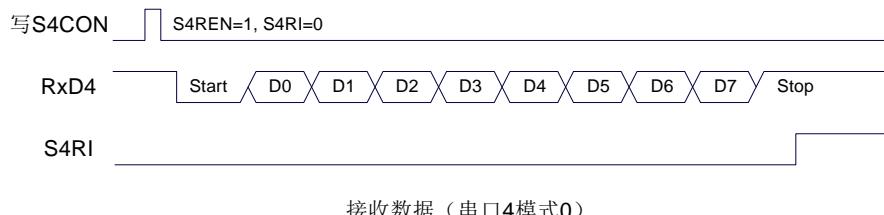
串口 4 数据寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S4BUF	85H								

S4BUF: 串口 1 数据接收/发送缓冲区。S4BUF 实际是 2 个缓冲器, 读缓冲器和写缓冲器, 两个操作分别对应两个不同的寄存器, 1 个是只写寄存器 (写缓冲器), 1 个是只读寄存器 (读缓冲器)。对 S4BUF 进行读操作, 实际是读取串口接收缓冲区, 对 S4BUF 进行写操作则是触发串口开始发送数据。

13.5.1 串口 4 模式 0

串行口 4 的模式 0 为 8 位数据位可变波特率 UART 工作模式。此模式一帧信息为 10 位: 1 位起始位, 8 位数据位 (低位在先) 和 1 位停止位。波特率可变, 可根据需要进行设置波特率。TxD4 为数据发送口, RxD4 为数据接收口, 串行口全双工接受/发送。



串口 4 的波特率是可变的，其波特率可由定时器 2 或定时器 4 产生。当定时器采用 1T 模式时（12 倍速），相应的波特率的速度也会相应提高 12 倍。

串口 4 模式 0 的波特率计算公式如下表所示：(SYSclk 为系统工作频率)

选择定时器	定时器速度	波特率计算公式
定时器2	1T	定时器2重载值 = $65536 - \frac{SYSclk}{4 \times \text{波特率}}$
	12T	定时器2重载值 = $65536 - \frac{SYSclk}{12 \times 4 \times \text{波特率}}$
定时器4	1T	定时器4重载值 = $65536 - \frac{SYSclk}{4 \times \text{波特率}}$
	12T	定时器4重载值 = $65536 - \frac{SYSclk}{12 \times 4 \times \text{波特率}}$

13.5.2 串口 4 模式 1

串行口 4 的模式 1 为 9 位数据位可变波特率 UART 工作模式。此模式一帧信息为 11 位：1 位起始位，9 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD4 为数据发送口，RxD4 为数据接收口，串行口全双工接受/发送。



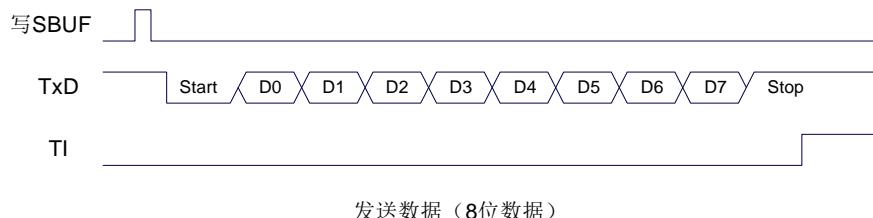


串口 4 模式 1 的波特率计算公式与模式 0 是完全相同的。请参考模式 0 的波特率计算公式。

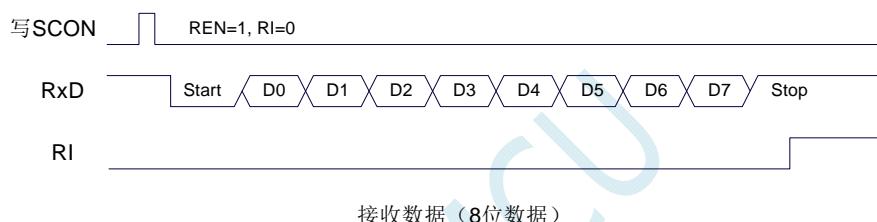
13.6 串口注意事项

关于串口中断请求有如下问题需要注意: (串口 1、串口 2、串口 3、串口 4 均类似, 下面以串口 1 为例进行说明)

8 位数据模式时, 发送完成整个停止位后产生 TI 中断请求, 如下图所示:



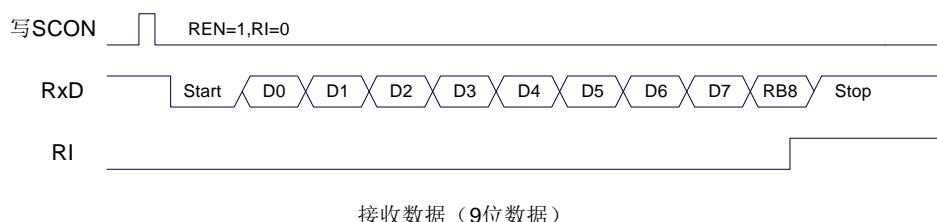
8 位数据模式时, 接收完成一半个停止位后产生 RI 中断请求, 如下图所示:



9 位数据模式时, 发送完成整个第 9 位数据位后产生 TI 中断请求, 如下图所示:



9 位数据模式时, 接收完成一半个第 9 位数据位后产生 RI 中断请求, 如下图所示:



13.7 范例程序

13.7.1 串口 1 使用定时器 2 做波特率发生器

汇编代码

<i>AUXR</i>	<i>DATA</i>	<i>8EH</i>	
<i>T2H</i>	<i>DATA</i>	<i>0D6H</i>	
<i>T2L</i>	<i>DATA</i>	<i>0D7H</i>	
<i>BUSY</i>	<i>BIT</i>	<i>20H.0</i>	
<i>WPTR</i>	<i>DATA</i>	<i>21H</i>	
<i>RPTR</i>	<i>DATA</i>	<i>22H</i>	
<i>BUFFER</i>	<i>DATA</i>	<i>23H</i>	<i>;16 bytes</i>
	<i>ORG</i>	<i>0000H</i>	
	<i>LJMP</i>	<i>MAIN</i>	
	<i>ORG</i>	<i>0023H</i>	
	<i>LJMP</i>	<i>UART_ISR</i>	
	<i>ORG</i>	<i>0100H</i>	
<i>UART_ISR:</i>			
	<i>PUSH</i>	<i>ACC</i>	
	<i>PUSH</i>	<i>PSW</i>	
	<i>MOV</i>	<i>PSW,#08H</i>	
	<i>JNB</i>	<i>TI,CHKRI</i>	
	<i>CLR</i>	<i>TI</i>	
	<i>CLR</i>	<i>BUSY</i>	
<i>CHKRI:</i>			
	<i>JNB</i>	<i>RI,UARTISR_EXIT</i>	
	<i>CLR</i>	<i>RI</i>	
	<i>MOV</i>	<i>A,WPTR</i>	
	<i>ANL</i>	<i>A,#0FH</i>	
	<i>ADD</i>	<i>A,#BUFFER</i>	
	<i>MOV</i>	<i>R0,A</i>	
	<i>MOV</i>	<i>@R0,SBUF</i>	
	<i>INC</i>	<i>WPTR</i>	
<i>UARTISR_EXIT:</i>			
	<i>POP</i>	<i>PSW</i>	
	<i>POP</i>	<i>ACC</i>	
	<i>RETI</i>		
<i>UART_INIT:</i>			
	<i>MOV</i>	<i>SCON,#50H</i>	
	<i>MOV</i>	<i>T2L,#0E8H</i>	<i>;65536-11059200/115200/4=0FFE8H</i>
	<i>MOV</i>	<i>T2H,#0FFH</i>	
	<i>MOV</i>	<i>AUXR,#15H</i>	
	<i>CLR</i>	<i>BUSY</i>	
	<i>MOV</i>	<i>WPTR,#00H</i>	
	<i>MOV</i>	<i>RPTR,#00H</i>	
	<i>RET</i>		
<i>UART_SEND:</i>			
	<i>JB</i>	<i>BUSY,\$</i>	
	<i>SETB</i>	<i>BUSY</i>	
	<i>MOV</i>	<i>SBUF,A</i>	
	<i>RET</i>		

UART_SENDSTR:

<i>CLR</i>	<i>A</i>
<i>MOVC</i>	<i>A,@A+DPTR</i>
<i>JZ</i>	<i>SENDEND</i>
<i>LCALL</i>	<i>UART_SEND</i>
<i>INC</i>	<i>DPTR</i>
<i>JMP</i>	<i>UART_SENDSTR</i>

SENDEND:

	<i>RET</i>
--	------------

MAIN:

<i>MOV</i>	<i>SP,#3FH</i>
<i>LCALL</i>	<i>UART_INIT</i>
<i>SETB</i>	<i>ES</i>
<i>SETB</i>	<i>EA</i>
<i>MOV</i>	<i>DPTR,#STRING</i>
<i>LCALL</i>	<i>UART_SENDSTR</i>

LOOP:

<i>MOV</i>	<i>A,RPTR</i>
<i>XRL</i>	<i>A,WPTR</i>
<i>ANL</i>	<i>A,#0FH</i>
<i>JZ</i>	<i>LOOP</i>
<i>MOV</i>	<i>A,RPTR</i>
<i>ANL</i>	<i>A,#0FH</i>
<i>ADD</i>	<i>A,#BUFFER</i>
<i>MOV</i>	<i>R0,A</i>
<i>MOV</i>	<i>A,@R0</i>
<i>LCALL</i>	<i>UART_SEND</i>
<i>INC</i>	<i>RPT</i>
<i>JMP</i>	<i>LOOP</i>

STRING: ***DB*** *'Uart Test !',0DH,0AH,00H*

	<i>END</i>
--	------------

C 语言代码

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

#define FOSC 11059200UL
#define BRT (65536 - FOSC / 115200 / 4)

sfr AUXR = 0x8e;
sfr T2H = 0xd6;
sfr T2L = 0xd7;

bit busy;
char wptr;
char rptr;
char buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
```

```

    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

void UartInit()
{
    SCON = 0x50;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x15;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void UartSendStr(char *p)
{
    while (*p)
    {
        UartSend(*p++);
    }
}

void main()
{
    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}

```

13.7.2 串口 1 使用定时器 1（模式 0）做波特率发生器

汇编代码

<i>AUXR</i>	<i>DATA</i>	<i>8EH</i>
<i>BUSY</i>	<i>BIT</i>	<i>20H.0</i>
<i>WPTR</i>	<i>DATA</i>	<i>21H</i>
<i>RPTR</i>	<i>DATA</i>	<i>22H</i>
<i>BUFFER</i>	<i>DATA</i>	<i>23H</i> <i>;16 bytes</i>
	<i>ORG</i>	<i>0000H</i>
	<i>LJMP</i>	<i>MAIN</i>
	<i>ORG</i>	<i>0023H</i>
	<i>LJMP</i>	<i>UART_ISR</i>
	<i>ORG</i>	<i>0100H</i>
<i>UART_ISR:</i>		
	<i>PUSH</i>	<i>ACC</i>
	<i>PUSH</i>	<i>PSW</i>
	<i>MOV</i>	<i>PSW,#08H</i>
	<i>JNB</i>	<i>TI,CHKRI</i>
	<i>CLR</i>	<i>TI</i>
	<i>CLR</i>	<i>BUSY</i>
<i>CHKRI:</i>		
	<i>JNB</i>	<i>RI,UARTISR_EXIT</i>
	<i>CLR</i>	<i>RI</i>
	<i>MOV</i>	<i>A,WPTR</i>
	<i>ANL</i>	<i>A,#0FH</i>
	<i>ADD</i>	<i>A,#BUFFER</i>
	<i>MOV</i>	<i>R0,A</i>
	<i>MOV</i>	<i>@R0,SBUF</i>
	<i>INC</i>	<i>WPTR</i>
<i>UARTISR_EXIT:</i>		
	<i>POP</i>	<i>PSW</i>
	<i>POP</i>	<i>ACC</i>
	<i>RETI</i>	
<i>UART_INIT:</i>		
	<i>MOV</i>	<i>SCON,#50H</i>
	<i>MOV</i>	<i>TMOD,#00H</i>
	<i>MOV</i>	<i>TLI,#0E8H</i> <i>;65536-11059200/115200/4=0FFE8H</i>
	<i>MOV</i>	<i>THI,#0FFH</i>
	<i>SETB</i>	<i>TRI</i>
	<i>MOV</i>	<i>AUXR,#40H</i>
	<i>CLR</i>	<i>BUSY</i>
	<i>MOV</i>	<i>WPTR,#00H</i>
	<i>MOV</i>	<i>RPTR,#00H</i>
	<i>RET</i>	
<i>UART_SEND:</i>		
	<i>JB</i>	<i>BUSY,\$</i>
	<i>SETB</i>	<i>BUSY</i>
	<i>MOV</i>	<i>SBUF,A</i>
	<i>RET</i>	
<i>UART_SENDSTR:</i>		
	<i>CLR</i>	<i>A</i>
	<i>MOVC</i>	<i>A,@A+DPTR</i>
	<i>JZ</i>	<i>SENDEND</i>
	<i>LCALL</i>	<i>UART_SEND</i>

<i>INC</i>	<i>DPTR</i>	
<i>JMP</i>	<i>UART_SENDSTR</i>	
SENDEND:		
<i>RET</i>		
MAIN:		
<i>MOV</i>	<i>SP,#3FH</i>	
<i>LCALL</i>	<i>UART_INIT</i>	
<i>SETB</i>	<i>ES</i>	
<i>SETB</i>	<i>EA</i>	
<i>MOV</i>	<i>DPTR,#STRING</i>	
<i>LCALL</i>	<i>UART_SENDSTR</i>	
LOOP:		
<i>MOV</i>	<i>A,RPTR</i>	
<i>XRL</i>	<i>A,WPTR</i>	
<i>ANL</i>	<i>A,#0FH</i>	
<i>JZ</i>	<i>LOOP</i>	
<i>MOV</i>	<i>A,RPTR</i>	
<i>ANL</i>	<i>A,#0FH</i>	
<i>ADD</i>	<i>A,#BUFFER</i>	
<i>MOV</i>	<i>R0,A</i>	
<i>MOV</i>	<i>A,@R0</i>	
<i>LCALL</i>	<i>UART_SEND</i>	
<i>INC</i>	<i>RPTR</i>	
<i>JMP</i>	<i>LOOP</i>	
STRING:	<i>DB</i>	'Uart Test !',0DH,0AH,00H
END		

C 语言代码

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

#define FOSC          11059200UL
#define BRT           (65536 - FOSC / 115200 / 4)

sfr AUXR = 0x8e;

bit busy;
char wptr;
char rptr;
char buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
    }
}
```

```

        wptr &= 0x0f;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TRI = 1;
    AUXR = 0x40;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void UartSendStr(char *p)
{
    while (*p)
    {
        UartSend(*p++);
    }
}

void main()
{
    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}

```

13.7.3 串口 1 使用定时器 1（模式 2）做波特率发生器

汇编代码

AUXR	DATA	8EH
BUSY	BIT	20H.0
WPTR	DATA	21H
RPTR	DATA	22H
BUFFER	DATA	23H
		;16 bytes

ORG *0000H*
LJMP *MAIN*
ORG *0023H*
LJMP *UART_ISR*

ORG *0100H*

UART_ISR:

PUSH *ACC*
PUSH *PSW*
MOV *PSW,#08H*

JNB *TI,CHKRI*
CLR *TI*
CLR *BUSY*

CHKRI:

JNB *RI,UARTISR_EXIT*
CLR *RI*
MOV *A,WPTR*
ANL *A,#0FH*
ADD *A,#BUFFER*
MOV *R0,A*
MOV *@R0,SBUF*
INC *WPTR*

UARTISR_EXIT:

POP *PSW*
POP *ACC*
RETI

UART_INIT:

MOV *SCON,#50H*
MOV *TMOD,#20H*
MOV *TLL,#0FDH*
MOV *THI,#0FDH* ;
SETB *TRI*
MOV *AUXR,#40H*
CLR *BUSY*
MOV *WPTR,#00H*
MOV *RPTR,#00H*
RET

UART_SEND:

JB *BUSY,\$*
SETB *BUSY*
MOV *SBUF,A*
RET

UART_SENDSTR:

CLR *A*
MOVC *A,@A+DPTR*
JZ *SENDEND*
LCALL *UART_SEND*
INC *DPTR*
JMP *UART_SENDSTR*

SENDEND:

RET

MAIN:

```

        MOV      SP,#3FH

        LCALL    UART_INIT
        SETB    ES
        SETB    EA

        MOV      DPTR,#STRING
        LCALL    UART_SENDSTR

LOOP:
        MOV      A,RPTR
        XRL      A,WPTR
        ANL      A,#0FH
        JZ       LOOP
        MOV      A,RPTR
        ANL      A,#0FH
        ADD      A,#BUFFER
        MOV      R0,A
        MOV      A,@R0
        LCALL    UART_SEND
        INC      R PTR
        JMP      LOOP

STRING:   DB      'Uart Test !',0DH,0AH,00H

END

```

C 语言代码

```

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

#define FOSC          11059200UL
#define BRT           (256 - FOSC / 115200 / 32)

sfr AUXR = 0x8e;

bit busy;
char wptr;
char rptr;
char buffer[16];

void UartIsr() interrupt 4
{
    if(TI)
    {
        TI = 0;
        busy = 0;
    }
    if(RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

void UartInit()
{

```

```

    SCON = 0x50;
    TMOD = 0x20;
    TL1 = BRT;
    TH1 = BRT;
    TR1 = 1;
    AUXR = 0x40;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

```

```
void UartSend(char dat)
```

```
{
    while (busy);
    busy = 1;
    SBUF = dat;
}
```

```
void UartSendStr(char *p)
```

```
{
    while (*p)
    {
        UartSend(*p++);
    }
}
```

```
void main()
```

```
{
    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

13.7.4 串口 2 使用定时器 2 做波特率发生器

汇编代码

AUXR	DATA	8EH
T2H	DATA	0D6H
T2L	DATA	0D7H
S2CON	DATA	9AH
S2BUF	DATA	9BH
IE2	DATA	0AFH
BUSY	BIT	20H.0
WPTR	DATA	21H
RPTR	DATA	22H
BUFFER	DATA	23H
		;16 bytes

<i>ORG</i>	<i>0000H</i>
<i>LJMP</i>	<i>MAIN</i>
<i>ORG</i>	<i>0043H</i>
<i>LJMP</i>	<i>UART2_ISR</i>
<i>ORG</i>	<i>0100H</i>

UART2_ISR:

<i>PUSH</i>	<i>ACC</i>
<i>PUSH</i>	<i>PSW</i>
<i>MOV</i>	<i>PSW,#08H</i>
<i>MOV</i>	<i>A,S2CON</i>
<i>JNB</i>	<i>ACC.1,CHKRI</i>
<i>ANL</i>	<i>S2CON,#NOT 02H</i>
<i>CLR</i>	<i>BUSY</i>

CHKRI:

<i>JNB</i>	<i>ACC.0,UART2ISR_EXIT</i>
<i>ANL</i>	<i>S2CON,#NOT 01H</i>
<i>MOV</i>	<i>A,WPTR</i>
<i>ANL</i>	<i>A,#0FH</i>
<i>ADD</i>	<i>A,#BUFFER</i>
<i>MOV</i>	<i>R0,A</i>
<i>MOV</i>	<i>@R0,S2BUF</i>
<i>INC</i>	<i>WPTR</i>

UART2ISR_EXIT:

<i>POP</i>	<i>PSW</i>
<i>POP</i>	<i>ACC</i>
<i>RETI</i>	

UART2_INIT:

<i>MOV</i>	<i>S2CON,#10H</i>
<i>MOV</i>	<i>T2L,#0E8H</i>
<i>MOV</i>	<i>T2H,#0FFH</i>
<i>MOV</i>	<i>AUXR,#14H</i>
<i>CLR</i>	<i>BUSY</i>
<i>MOV</i>	<i>WPTR,#00H</i>
<i>MOV</i>	<i>RPTR,#00H</i>
<i>RET</i>	

;65536-11059200/115200/4=0FFE8H

UART2_SEND:

<i>JB</i>	<i>BUSY,\$</i>
<i>SETB</i>	<i>BUSY</i>
<i>MOV</i>	<i>S2BUFA</i>
<i>RET</i>	

UART2_SENDSTR:

<i>CLR</i>	<i>A</i>
<i>MOVC</i>	<i>A,@A+DPTR</i>
<i>JZ</i>	<i>SEND2END</i>
<i>LCALL</i>	<i>UART2_SEND</i>
<i>INC</i>	<i>DPTR</i>
<i>JMP</i>	<i>UART2_SENDSTR</i>

SEND2END:

<i>RET</i>	
------------	--

MAIN:

<i>MOV</i>	<i>SP,#3FH</i>
------------	----------------

```

LCALL    UART2_INIT
MOV      IE2,#01H
SETB    EA

MOV      DPTR,#STRING
LCALL    UART2_SENDSTR

```

LOOP:

```

MOV      A,RPTR
XRL    A,WPTR
ANL    A,#0FH
JZ     LOOP
MOV      A,RPTR
ANL    A,#0FH
ADD    A,#BUFFER
MOV      R0,A
MOV      A,@R0
LCALL   UART2_SEND
INC    RPT
JMP    LOOP

```

STRING: **DB** 'Uart Test !',0DH,0AH,00H

END

C 语言代码

```

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

#define FOSC          11059200UL
#define BRT           (65536 - FOSC / 115200 / 4)

sfr    AUXR        = 0x8e;
sfr    T2H         = 0xd6;
sfr    T2L         = 0xd7;
sfr    S2CON        = 0x9a;
sfr    S2BUF        = 0x9b;
sfr    IE2          = 0xaf;

bit    busy;
char   wptr;
char   rptr;
char   buffer[16];

void Uart2Isr() interrupt 8
{
    if(S2CON & 0x02)
    {
        S2CON &= ~0x02;
        busy = 0;
    }
    if(S2CON & 0x01)
    {
        S2CON &= ~0x01;
        buffer[wptr++] = S2BUF;
        wptr &= 0x0f;
    }
}

```

```

void Uart2Init()
{
    S2CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart2Send(char dat)
{
    while (busy);
    busy = 1;
    S2BUF = dat;
}

void Uart2SendStr(char *p)
{
    while (*p)
    {
        Uart2Send(*p++);
    }
}

void main()
{
    Uart2Init();
    IE2 = 0x01;
    EA = 1;
    Uart2SendStr("Uart Test !r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart2Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}

```

13.7.5 串口 3 使用定时器 2 做波特率发生器

汇编代码

AUXR	DATA	8EH
T2H	DATA	0D6H
T2L	DATA	0D7H
S3CON	DATA	0ACh
S3BUF	DATA	0ADH
IE2	DATA	0AFH
BUSY	BIT	20H.0
WPTR	DATA	21H
RPTR	DATA	22H
BUFFER	DATA	23H
		<i>;16 bytes</i>

<i>ORG</i>	0000H
<i>LJMP</i>	MAIN
<i>ORG</i>	008BH
<i>LJMP</i>	UART3_ISR
<i>ORG</i>	0100H

UART3_ISR:

<i>PUSH</i>	ACC
<i>PUSH</i>	PSW
<i>MOV</i>	PSW,#08H
<i>MOV</i>	A,S3CON
<i>JNB</i>	ACC.1,CHKRI
<i>ANL</i>	S3CON,#NOT 02H
<i>CLR</i>	BUSY

CHKRI:

<i>JNB</i>	ACC.0,UART3ISR_EXIT
<i>ANL</i>	S3CON,#NOT 01H
<i>MOV</i>	A,WPTR
<i>ANL</i>	A,#0FH
<i>ADD</i>	A,#BUFFER
<i>MOV</i>	R0,A
<i>MOV</i>	@R0,S3BUF
<i>INC</i>	WPTR

UART3ISR_EXIT:

<i>POP</i>	PSW
<i>POP</i>	ACC
<i>RETI</i>	

UART3_INIT:

<i>MOV</i>	S3CON,#10H
<i>MOV</i>	T2L,#0E8H
<i>MOV</i>	T2H,#0FFH
<i>MOV</i>	AUXR,#14H
<i>CLR</i>	BUSY
<i>MOV</i>	WPTR,#00H
<i>MOV</i>	RPTR,#00H
<i>RET</i>	

;65536-11059200/115200/4=0FFE8H

UART3_SEND:

<i>JB</i>	BUSY,\$
<i>SETB</i>	BUSY
<i>MOV</i>	S3BUF,A
<i>RET</i>	

UART3_SENDSTR:

<i>CLR</i>	A
<i>MOVC</i>	A,@A+DPTR
<i>JZ</i>	SEND3END
<i>LCALL</i>	UART3_SEND
<i>INC</i>	DPTR
<i>JMP</i>	UART3_SENDSTR

SEND3END:

<i>RET</i>	
------------	--

MAIN:

<i>MOV</i>	SP,#3FH
------------	----------------

```

LCALL      UART3_INIT
MOV        IE2,#08H
SETB      EA

MOV        DPTR,#STRING
LCALL      UART3_SENDSTR

LOOP:
MOV        A,RPTR
XRL        A,WPTR
ANL        A,#0FH
JZ         LOOP
MOV        A,RPTR
ANL        A,#0FH
ADD        A,#BUFFER
MOV        R0,A
MOV        A,@R0
LCALL    UART3_SEND
INC        RPTR
JMP        LOOP

STRING:    DB        'Uart Test !',0DH,0AH,00H

END

```

C 语言代码

```

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

#define FOSC          11059200UL
#define BRT           (65536 - FOSC / 115200 / 4)

sfr AUXR        = 0x8e;
sfr T2H          = 0xd6;
sfr T2L          = 0xd7;
sfr S3CON         = 0xac;
sfr S3BUF         = 0xad;
sfr IE2          = 0xaf;

bit busy;
char wptr;
char rptr;
char buffer[16];

void Uart3Isr() interrupt 17
{
    if(S3CON & 0x02)
    {
        S3CON &= ~0x02;
        busy = 0;
    }
    if(S3CON & 0x01)
    {
        S3CON &= ~0x01;
        buffer[wptr++] = S3BUF;
        wptr &= 0x0f;
    }
}

```

```

}

void Uart3Init()
{
    S3CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart3Send(char dat)
{
    while (busy);
    busy = 1;
    S3BUF = dat;
}

void Uart3SendStr(char *p)
{
    while (*p)
    {
        Uart3Send(*p++);
    }
}

void main()
{
    Uart3Init();
    IE2 = 0x08;
    EA = 1;
    Uart3SendStr("Uart Test !r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart3Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}

```

13.7.6 串口 3 使用定时器 3 做波特率发生器

汇编代码

T4T3M	DATA	0D1H
T4H	DATA	0D2H
T4L	DATA	0D3H
T3H	DATA	0D4H
T3L	DATA	0D5H
S3CON	DATA	0ACh
S3BUF	DATA	0ADH
IE2	DATA	0AFH
BUSY	BIT	20H.0

WPTR DATA 21H
RPTR DATA 22H
BUFFER DATA 23H ;16 bytes

ORG 0000H
LJMP MAIN
ORG 008BH
LJMP UART3_ISR
ORG 0100H

UART3_ISR:

PUSH ACC
PUSH PSW
MOV PSW,#08H

MOV A,S3CON
JNB ACC.1,CHKRI
ANL S3CON,#NOT 02H
CLR BUSY

CHKRI:

JNB ACC.0,UART3ISR_EXIT
ANL S3CON,#NOT 01H
MOV A,WPTR
ANL A,#0FH
ADD A,#BUFFER
MOV R0,A
MOV @R0,S3BUF
INC WPTR

UART3ISR_EXIT:

POP PSW
POP ACC
RETI

UART3_INIT:

MOV S3CON,#50H
MOV T3L,#0E8H ;65536-11059200/115200/4=0FFE8H
MOV T3H,#0FFH
MOV T4T3M,#0AH
CLR BUSY
MOV WPTR,#00H
MOV RPTR,#00H
RET

UART3_SEND:

JB BUSY,\$
SETB BUSY
MOV S3BUF,A
RET

UART3_SENDSTR:

CLR A
MOVC A,@A+DPTR
JZ SEND3END
LCALL UART3_SEND
INC DPTR
JMP UART3_SENDSTR

SEND3END:

RET

MAIN:

```

MOV      SP,#3FH
LCALL    UART3_INIT
MOV      IE2,#08H
SETB    EA
MOV      DPTR,#STRING
LCALL    UART3_SENDSTR

```

LOOP:

```

MOV      A,RPTR
XRL      A,WPTR
ANL      A,#0FH
JZ       LOOP
MOV      A,RPTR
ANL      A,#0FH
ADD      A,#BUFFER
MOV      R0,A
MOV      A,@R0
LCALL   UART3_SEND
INC      RPTR
JMP      LOOP

```

STRING: DB 'Uart Test !,0DH,0AH,00H

END

C 语言代码

```

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

#define FOSC 11059200UL
#define BRT (65536 - FOSC / 115200 / 4)

sfr T4T3M = 0xd1;
sfr T4H = 0xd2;
sfr T4L = 0xd3;
sfr T3H = 0xd4;
sfr T3L = 0xd5;
sfr S3CON = 0xac;
sfr S3BUF = 0xad;
sfr IE2 = 0xaf;

bit busy;
char wptr;
char rptr;
char buffer[16];

void Uart3Isr() interrupt 17
{
    if(S3CON & 0x02)
    {
        S3CON &= ~0x02;
        busy = 0;
    }
    if(S3CON & 0x01)

```

```

    {
        S3CON &= ~0x01;
        buffer[wptr++] = S3BUF;
        wptr &= 0x0f;
    }
}

```

```

void Uart3Init()
{
    S3CON = 0x50;
    T3L = BRT;
    T3H = BRT >> 8;
    T4T3M = 0x0a;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

```

```

void Uart3Send(char dat)
{
    while (busy);
    busy = 1;
    S3BUF = dat;
}

```

```

void Uart3SendStr(char *p)
{
    while (*p)
    {
        Uart3Send(*p++);
    }
}

```

```

void main()
{
    Uart3Init();
    IE2 = 0x08;
    EA = 1;
    Uart3SendStr("Uart Test !r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart3Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}

```

13.7.7 串口 4 使用定时器 2 做波特率发生器

汇编代码

AUXR	DATA	8EH
T2H	DATA	0D6H
T2L	DATA	0D7H
S4CON	DATA	84H
S4BUF	DATA	085H

IE2	DATA	0AFH
BUSY	BIT	20H.0
WPTR	DATA	21H
RPTR	DATA	22H
BUFFER	DATA	23H
		<i>;16 bytes</i>
	ORG	0000H
	LJMP	MAIN
	ORG	0093H
	LJMP	UART4_ISR
	ORG	0100H
UART4_ISR:		
	PUSH	ACC
	PUSH	PSW
	MOV	PSW,#08H
	MOV	A,S4CON
	JNB	ACC.1,CHKRI
	ANL	S4CON,#NOT 02H
	CLR	BUSY
CHKRI:		
	JNB	ACC.0,UART4ISR_EXIT
	ANL	S4CON,#NOT 01H
	MOV	A,WPTR
	ANL	A,#0FH
	ADD	A,#BUFFER
	MOV	R0,A
	MOV	@R0,S4BUF
	INC	WPTR
UART4ISR_EXIT:		
	POP	PSW
	POP	ACC
	RETI	
UART4_INIT:		
	MOV	S4CON,#10H
	MOV	T2L,#0E8H
	MOV	T2H,#0FFH
	MOV	AUXR,#14H
	CLR	BUSY
	MOV	WPTR,#00H
	MOV	RPTR,#00H
	RET	
UART4_SEND:		
	JB	BUSY,\$
	SETB	BUSY
	MOV	S4BUFA
	RET	
UART4_SENDSTR:		
	CLR	A
	MOVC	A,@A+DPTR
	JZ	SEND4END
	LCALL	UART4_SEND
	INC	DPTR

	<i>JMP</i>	<i>UART4_SENDSTR</i>
<i>SEND4END:</i>		<i>RET</i>
 <i>MAIN:</i>		
	<i>MOV</i>	<i>SP,#3FH</i>
	<i>LCALL</i>	<i>UART4_INIT</i>
	<i>MOV</i>	<i>IE2,#10H</i>
	<i>SETB</i>	<i>EA</i>
	<i>MOV</i>	<i>DPTR,#STRING</i>
	<i>LCALL</i>	<i>UART4_SENDSTR</i>
 <i>LOOP:</i>		
	<i>MOV</i>	<i>A,RPTR</i>
	<i>XRL</i>	<i>A,WPTR</i>
	<i>ANL</i>	<i>A,#0FH</i>
	<i>JZ</i>	<i>LOOP</i>
	<i>MOV</i>	<i>A,RPTR</i>
	<i>ANL</i>	<i>A,#0FH</i>
	<i>ADD</i>	<i>A,#BUFFER</i>
	<i>MOV</i>	<i>R0,A</i>
	<i>MOV</i>	<i>A,@R0</i>
	<i>LCALL</i>	<i>UART4_SEND</i>
	<i>INC</i>	<i>RPTR</i>
	<i>JMP</i>	<i>LOOP</i>
<i>STRING:</i>	<i>DB</i>	'Uart Test !',0DH,0AH,00H
		<i>END</i>

C 语言代码

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

#define FOSC 11059200UL
#define BRT (65536 - FOSC / 115200 / 4)

sfr AUXR = 0x8e;
sfr T2H = 0xd6;
sfr T2L = 0xd7;
sfr S4CON = 0x84;
sfr S4BUF = 0x85;
sfr IE2 = 0xaf;

bit busy;
char wptr;
char rptr;
char buffer[16];

void Uart4Isr() interrupt 18
{
    if(S4CON & 0x02)
    {
        S4CON &= ~0x02;
        busy = 0;
    }
}
```

```

if (S4CON & 0x01)
{
    S4CON &= ~0x01;
    buffer[wptr++] = S4BUF;
    wptr &= 0x0f;
}
}

```

```
void Uart4Init()
```

```
{
    S4CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}
```

```
void Uart4Send(char dat)
```

```
{
    while (busy);
    busy = 1;
    S4BUF = dat;
}
```

```
void Uart4SendStr(char *p)
```

```
{
    while (*p)
    {
        Uart4Send(*p++);
    }
}
```

```
void main()
```

```
{
    Uart4Init();
    IE2 = 0x10;
    EA = 1;
    Uart4SendStr("Uart Test !r\n");
}
```

```

while (1)
{
    if (rptr != wptr)
    {
        Uart4Send(buffer[rptr++]);
        rptr &= 0x0f;
    }
}
}

```

13.7.8 串口 4 使用定时器 4 做波特率发生器

汇编代码

T4T3M	DATA	0D1H
T4H	DATA	0D2H
T4L	DATA	0D3H
T3H	DATA	0D4H

<i>T3L</i>	<i>DATA</i>	<i>0D5H</i>	
<i>S4CON</i>	<i>DATA</i>	<i>84H</i>	
<i>S4BUF</i>	<i>DATA</i>	<i>085H</i>	
<i>IE2</i>	<i>DATA</i>	<i>0AFH</i>	
<i>BUSY</i>	<i>BIT</i>	<i>20H.0</i>	
<i>WPTR</i>	<i>DATA</i>	<i>21H</i>	
<i>RPTR</i>	<i>DATA</i>	<i>22H</i>	
<i>BUFFER</i>	<i>DATA</i>	<i>23H</i>	<i>;16 bytes</i>
	<i>ORG</i>	<i>0000H</i>	
	<i>LJMP</i>	<i>MAIN</i>	
	<i>ORG</i>	<i>0093H</i>	
	<i>LJMP</i>	<i>UART4_ISR</i>	
	<i>ORG</i>	<i>0100H</i>	
 <i>UART4_ISR:</i>			
	<i>PUSH</i>	<i>ACC</i>	
	<i>PUSH</i>	<i>PSW</i>	
	<i>MOV</i>	<i>PSW,#08H</i>	
	<i>MOV</i>	<i>A,S4CON</i>	
	<i>JNB</i>	<i>ACC.1,CHKRI</i>	
	<i>ANL</i>	<i>S4CON,#NOT 02H</i>	
	<i>CLR</i>	<i>BUSY</i>	
 <i>CHKRI:</i>			
	<i>JNB</i>	<i>ACC.0,UART4ISR_EXIT</i>	
	<i>ANL</i>	<i>S4CON,#NOT 01H</i>	
	<i>MOV</i>	<i>A,WPTR</i>	
	<i>ANL</i>	<i>A,#0FH</i>	
	<i>ADD</i>	<i>A,#BUFFER</i>	
	<i>MOV</i>	<i>R0,A</i>	
	<i>MOV</i>	<i>@R0,S4BUF</i>	
	<i>INC</i>	<i>WPTR</i>	
 <i>UART4ISR_EXIT:</i>			
	<i>POP</i>	<i>PSW</i>	
	<i>POP</i>	<i>ACC</i>	
	<i>RETI</i>		
 <i>UART4_INIT:</i>			
	<i>MOV</i>	<i>S4CON,#50H</i>	
	<i>MOV</i>	<i>T4L,#0E8H</i>	<i>;65536-11059200/115200/4=0FFE8H</i>
	<i>MOV</i>	<i>T4H,#0FFH</i>	
	<i>MOV</i>	<i>T4T3M,#0A0H</i>	
	<i>CLR</i>	<i>BUSY</i>	
	<i>MOV</i>	<i>WPTR,#00H</i>	
	<i>MOV</i>	<i>RPTR,#00H</i>	
	<i>RET</i>		
 <i>UART4_SEND:</i>			
	<i>JB</i>	<i>BUSY,\$</i>	
	<i>SETB</i>	<i>BUSY</i>	
	<i>MOV</i>	<i>S4BUFA</i>	
	<i>RET</i>		
 <i>UART4_SENDSTR:</i>			
	<i>CLR</i>	<i>A</i>	
	<i>MOVC</i>	<i>A,@A+DPTR</i>	

<i>JZ</i>	<i>SEND4END</i>	
<i>LCALL</i>	<i>UART4_SEND</i>	
<i>INC</i>	<i>DPTR</i>	
<i>JMP</i>	<i>UART4_SENDSTR</i>	
<i>SEND4END:</i>		
<i>RET</i>		
<i>MAIN:</i>		
<i>MOV</i>	<i>SP,#3FH</i>	
<i>LCALL</i>	<i>UART4_INIT</i>	
<i>MOV</i>	<i>IE2,#10H</i>	
<i>SETB</i>	<i>EA</i>	
<i>MOV</i>	<i>DPTR,#STRING</i>	
<i>LCALL</i>	<i>UART4_SENDSTR</i>	
<i>LOOP:</i>		
<i>MOV</i>	<i>A,RPTR</i>	
<i>XRL</i>	<i>A,WPTR</i>	
<i>ANL</i>	<i>A,#0FH</i>	
<i>JZ</i>	<i>LOOP</i>	
<i>MOV</i>	<i>A,RPTR</i>	
<i>ANL</i>	<i>A,#0FH</i>	
<i>ADD</i>	<i>A,#BUFFER</i>	
<i>MOV</i>	<i>R0,A</i>	
<i>MOV</i>	<i>A,@R0</i>	
<i>LCALL</i>	<i>UART4_SEND</i>	
<i>INC</i>	<i>RPTR</i>	
<i>JMP</i>	<i>LOOP</i>	
<i>STRING:</i>	<i>DB</i>	'Uart Test !',0DH,0AH,00H
<i>END</i>		

C 语言代码

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

#define FOSC 11059200UL
#define BRT (65536 - FOSC / 115200 / 4)

sfr T4T3M = 0xd1;
sfr T4H = 0xd2;
sfr T4L = 0xd3;
sfr T3H = 0xd4;
sfr T3L = 0xd5;
sfr S4CON = 0x84;
sfr S4BUF = 0x85;
sfr IE2 = 0xaf;

bit busy;
char wptr;
char rptr;
char buffer[16];

void Uart4Isr() interrupt 18
{

```

```
if (S4CON & 0x02)
{
    S4CON &= ~0x02;
    busy = 0;
}
if (S4CON & 0x01)
{
    S4CON &= ~0x01;
    buffer[wptr++] = S4BUF;
    wptr &= 0x0f;
}
}
```

```
void Uart4Init()
```

```
{
    S4CON = 0x50;
    T4L = BRT;
    T4H = BRT >> 8;
    T4T3M = 0xa0;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}
```

```
void Uart4Send(char dat)
```

```
{
    while (busy);
    busy = 1;
    S4BUF = dat;
}
```

```
void Uart4SendStr(char *p)
```

```
{
    while (*p)
    {
        Uart4Send(*p++);
    }
}
```

```
void main()
```

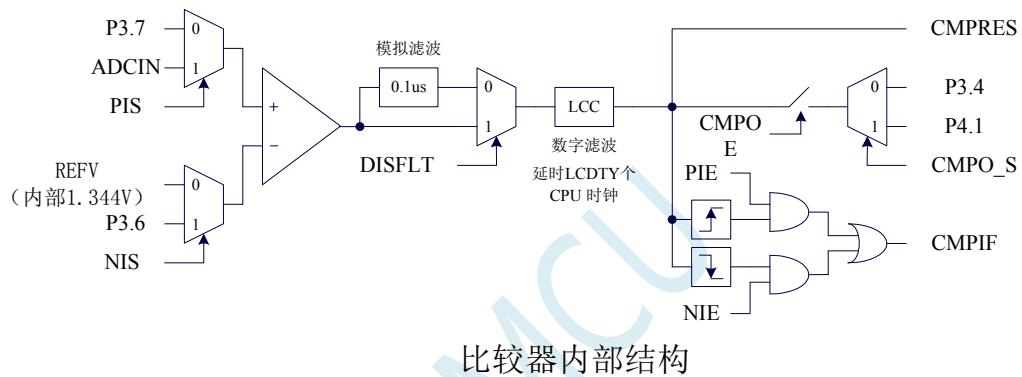
```
{
    Uart4Init();
    IE2 = 0x10;
    EA = 1;
    Uart4SendStr("Uart Test !r\n");
    while (1)
    {
        if (rptr != wptr)
        {
            Uart4Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

14 比较器，掉电检测，内部固定比较电压

STC8 系列单片机内部集成了一个比较器。比较器的正极可以是 P3.7 端口或者 ADC 的模拟输入通道，而负极可以 P3.6 端口或者是内部 BandGap 经过 OP 后的 REFV 电压（内部固定比较电压）。

比较器内部有可程序控制的两级滤波：模拟滤波和数字滤波。模拟滤波可以过滤掉比较输入信号中的毛刺信号，数字滤波可以等待输入信号更加稳定后再进行比较。比较结果可直接通过读取内部寄存器位获得，也可将比较器结果正向或反向输出到外部端口。将比较结果输出到外部端口可用作外部事件的触发信号和反馈信号，可扩大比较的应用范围。

14.1 比较器内部结构图



比较器内部结构

14.2 比较器相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CMPCR1	比较器控制寄存器 1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES	0000,0000
CMPCR2	比较器控制寄存器 2	E7H	INVCMPO	DISFLT	LCDTY[5:0]						0000,0000

比较器控制寄存器 1

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES

CMPEN: 比较器模块使能位

0: 关闭比较功能

1: 使能比较功能

CMPIF: 比较器中断标志位。当 PIE 或 NIE 被使能后, 若产生相应的中断信号, 硬件自动将 CMPIF 置 1, 并向 CPU 提出中断请求。此标志位必须用户软件清零。

(注意: 没有使能比较器中断时, 硬件不会设置此中断标志, 即使用查询方式访问比较器时, 不能查询此中断标志)

PIE: 比较器上升沿中断使能位。

0: 禁止比较器上升沿中断。

1: 使能比较器上升沿中断。使能比较器的比较结果由 0 变成 1 时产生中断请求。

NIE: 比较器下降沿中断使能位。

0: 禁止比较器下降沿中断。

1: 使能比较器下降沿中断。使能比较器的比较结果由 1 变成 0 时产生中断请求。

PIS: 比较器的正极选择位

0: 选择外部端口 P3.7 为比较器正极输入源。

1: 通过 ADC_CONTR 中的 ADC_CHS 位选择 ADC 的模拟输入端作为比较器正极输入源。

NIS: 比较器的负极选择位

0: 选择内部 BandGap 经过 OP 后的电压 REFV 作为比较器负极输入源 (REFV 的电压值为 1.344V, 由于制造误差, 实际电压值可能在 1.34V~1.35V 之间)。

1: 选择外部端口 P3.6 为比较器负极输入源。

CMPOE: 比较器结果输出控制位

0: 禁止比较器结果输出

1: 使能比较器结果输出。比较器结果输出到 P3.4 或者 P4.1 (由 P_SW2 中的 CMPO_S 进行设定)

CMPRES: 比较器的比较结果。此位为只读。

0: 表示 CMP+ 的电平低于 CMP- 的电平

1: 表示 CMP+ 的电平高于 CMP- 的电平

CMPRES 是经过数字滤波后的输出信号, 而不是比较器的直接输出结果。

比较器控制寄存器 2

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR2	E7H	INVCMPO	DISFLT	LCDTY[5:0]					

INVCMPO: 比较器结果输出控制

0: 比较器结果正向输出。若 CMPRES 为 0, 则 P3.4/P4.1 输出低电平, 反之输出高电平。

1: 比较器结果反向输出。若 CMPRES 为 0, 则 P3.4/P4.1 输出高电平, 反之输出低电平。

DISFLT: 模拟滤波功能控制

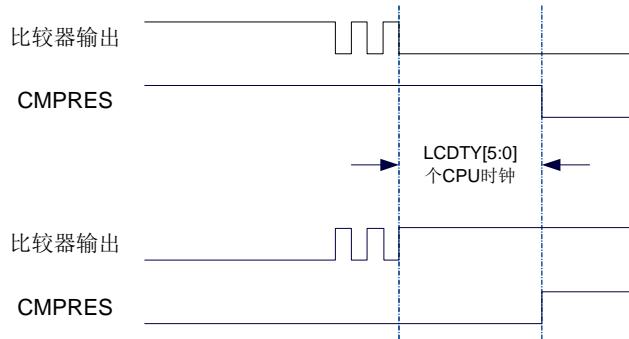
0: 使能 0.1us 模拟滤波功能

1: 关闭 0.1us 模拟滤波功能, 可略微提高比较器的比较速度。

LCDTY[5:0]: 数字滤波功能控制

数字滤波功能即为数字信号去抖动功能。当比较结果发生上升沿或者下降沿变化时, 比较器侦测变化后的信号必须维持 LCDTY 所设置的 CPU 时钟数不发生变化, 才认为数据变化是有效的; 否则将视同信号无变化。

若 LCDTY 设置为 0 时表示关闭数字滤波功能。



14.3 范例程序

14.3.1 比较器的使用（中断方式）

汇编代码

<i>CMPCRI</i>	<i>DATA</i>	<i>0E6H</i>	
<i>CMPCR2</i>	<i>DATA</i>	<i>0E7H</i>	
	<i>ORG</i>	<i>0000H</i>	
	<i>LJMP</i>	<i>MAIN</i>	
	<i>ORG</i>	<i>00ABH</i>	
	<i>LJMP</i>	<i>CMPISR</i>	
	<i>ORG</i>	<i>0100H</i>	
<i>CMPISR:</i>	<i>PUSH</i>	<i>ACC</i>	
	<i>ANL</i>	<i>CMPCRI,#NOT 40H</i>	;清中断标志
	<i>MOV</i>	<i>A,CMPCRI</i>	
	<i>JB</i>	<i>ACC.0,RSING</i>	
<i>FALLING:</i>	<i>CPL</i>	<i>P1.0</i>	;下降沿中断测试端口
	<i>POP</i>	<i>ACC</i>	
	<i>RETI</i>		
<i>RSING:</i>	<i>CPL</i>	<i>P1.1</i>	;上升沿中断测试端口
	<i>POP</i>	<i>ACC</i>	
	<i>RETI</i>		
<i>MAIN:</i>	<i>MOV</i>	<i>SP,#3FH</i>	
	<i>MOV</i>	<i>CMPCR2,#00H</i>	
	<i>ANL</i>	<i>CMPCR2,#NOT 80H</i>	;比较器正向输出
;	<i>ORL</i>	<i>CMPCR2,#80H</i>	;比较器反向输出
	<i>ANL</i>	<i>CMPCR2,#NOT 40H</i>	;禁止 0.1us 滤波
;	<i>ORL</i>	<i>CMPCR2,#40H</i>	;使能 0.1us 滤波
;	<i>ANL</i>	<i>CMPCR2,#NOT 3FH</i>	;比较器结果直接输出
	<i>ORL</i>	<i>CMPCR2,#10H</i>	;比较器结果经过 16 个去抖时钟后输出
	<i>MOV</i>	<i>CMPCRI,#00H</i>	
	<i>ORL</i>	<i>CMPCRI,#30H</i>	;使能比较器边沿中断
;	<i>ANL</i>	<i>CMPCRI,#NOT 20H</i>	;禁止比较器上升沿中断
;	<i>ORL</i>	<i>CMPCRI,#20H</i>	;使能比较器上升沿中断
;	<i>ANL</i>	<i>CMPCRI,#NOT 10H</i>	;禁止比较器下降沿中断
;	<i>ORL</i>	<i>CMPCRI,#10H</i>	;使能比较器下降沿中断
	<i>ANL</i>	<i>CMPCRI,#NOT 08H</i>	;P3.7 为 CMP+ 输入脚
;	<i>ORL</i>	<i>CMPCRI,#08H</i>	;ADC 输入脚为 CMP+ 输入脚
;	<i>ANL</i>	<i>CMPCRI,#NOT 04H</i>	;内部参考电压为 CMP- 输入脚
	<i>ORL</i>	<i>CMPCRI,#04H</i>	;P3.6 为 CMP- 输入脚
;	<i>ANL</i>	<i>CMPCRI,#NOT 02H</i>	;禁止比较器输出
	<i>ORL</i>	<i>CMPCRI,#02H</i>	;使能比较器输出
	<i>ORL</i>	<i>CMPCRI,#80H</i>	;使能比较器模块
	<i>SETB</i>	<i>EA</i>	
	<i>JMP</i>	\$	
	<i>END</i>		

C 语言代码

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

sfr CMPCR1 = 0xe6;
sfr CMPCR2 = 0xe7;

sbit P10 = P1^0;
sbit P11 = P1^1;

void CMP_Isr() interrupt 21
{
    CMPCR1 &= ~0x40; //清中断标志
    if(CMPCR1 & 0x01)
    {
        P10 = !P10; //下降沿中断测试端口
    }
    else
    {
        P11 = !P11; //上升沿中断测试端口
    }
}

void main()
{
    CMPCR2 = 0x00; //比较器正向输出
//    CMPCR2 &= ~0x80; //比较器反向输出
//    CMPCR2 |= 0x80; //禁止 0.1us 滤波
//    CMPCR2 |= 0x40; //使能 0.1us 滤波
//    CMPCR2 &= ~0x3f; //比较器结果直接输出
//    CMPCR2 |= 0x10; //比较器结果经过 16 个去抖时钟后输出
    CMPCR1 = 0x00; //使能比较器边沿中断
    CMPCR1 |= 0x30; //禁止比较器上升沿中断
//    CMPCR1 &= ~0x20; //使能比较器上升沿中断
//    CMPCR1 |= 0x20; //禁止比较器下降沿中断
//    CMPCR1 &= ~0x10; //使能比较器下降沿中断
//    CMPCR1 |= 0x10; //P3.7 为 CMP+ 输入脚
//    CMPCR1 &= ~0x08; //ADC 输入脚为 CMP+ 输入脚
//    CMPCR1 |= 0x08; //内部参考电压为 CMP- 输入脚
//    CMPCR1 &= ~0x04; //P3.6 为 CMP- 输入脚
//    CMPCR1 |= 0x04; //禁止比较器输出
//    CMPCR1 &= ~0x02; //使能比较器输出
//    CMPCR1 |= 0x02; //使能比较器模块
    CMPCR1 |= 0x80;

    EA = 1;

    while (1);
}

```

14.3.2 比较器的使用（查询方式）

汇编代码

CMPCR1	DATA	0E6H
CMPCR2	DATA	0E7H

```

ORG      0000H
LJMP     MAIN

MAIN:
ORG      0100H
MOV      SP,#3FH

MOV      CMPCR2,#00H
ANL      CMPCR2,#NOT 80H      ; 比较器正向输出
; ORL      CMPCR2,#80H      ; 比较器反向输出
; ANL      CMPCR2,#NOT 40H      ; 禁止 0.1us 滤波
; ORL      CMPCR2,#40H      ; 使能 0.1us 滤波
; ANL      CMPCR2,#NOT 3FH      ; 比较器结果直接输出
; ORL      CMPCR2,#10H      ; 比较器结果经过 16 个去抖时钟后输出
MOV      CMPCRI,#00H
ORL      CMPCRI,#30H      ; 使能比较器边沿中断
; ANL      CMPCRI,#NOT 20H      ; 禁止比较器上升沿中断
; ORL      CMPCRI,#20H      ; 使能比较器上升沿中断
; ANL      CMPCRI,#NOT 10H      ; 禁止比较器下降沿中断
; ORL      CMPCRI,#10H      ; 使能比较器下降沿中断
; ANL      CMPCRI,#NOT 08H      ; P3.7 为 CMP+ 输入脚
; ORL      CMPCRI,#08H      ; ADC 输入脚为 CMP+ 输入脚
; ANL      CMPCRI,#NOT 04H      ; 内部参考电压为 CMP- 输入脚
; ORL      CMPCRI,#04H      ; P3.6 为 CMP- 输入脚
; ANL      CMPCRI,#NOT 02H      ; 禁止比较器输出
; ORL      CMPCRI,#02H      ; 使能比较器输出
; ORL      CMPCRI,#80H      ; 使能比较器模块

LOOP:
MOV      A,CMPCRI
MOV      C,ACC.0
MOV      P1.0,C      ; 读取比较器比较结果
JMP      LOOP

END

```

C 语言代码

```

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

sfr    CMPCRI    = 0xe6;
sfr    CMPCR2    = 0xe7;

sbit   P10        = P1^0;
sbit   P11        = P1^1;

void main()
{
    CMPCR2 = 0x00;
    CMPCR2 &= ~0x80;          // 比较器正向输出
//    CMPCR2 |= 0x80;          // 比较器反向输出
    CMPCR2 &= ~0x40;          // 禁止 0.1us 滤波
//    CMPCR2 |= 0x40;          // 使能 0.1us 滤波
//    CMPCR2 &= ~0x3f;          // 比较器结果直接输出
    CMPCR2 |= 0x10;          // 比较器结果经过 16 个去抖时钟后输出
    CMPCRI = 0x00;
    CMPCRI |= 0x30;          // 使能比较器边沿中断
//    CMPCRI &= ~0x20;          // 禁止比较器上升沿中断
}

```

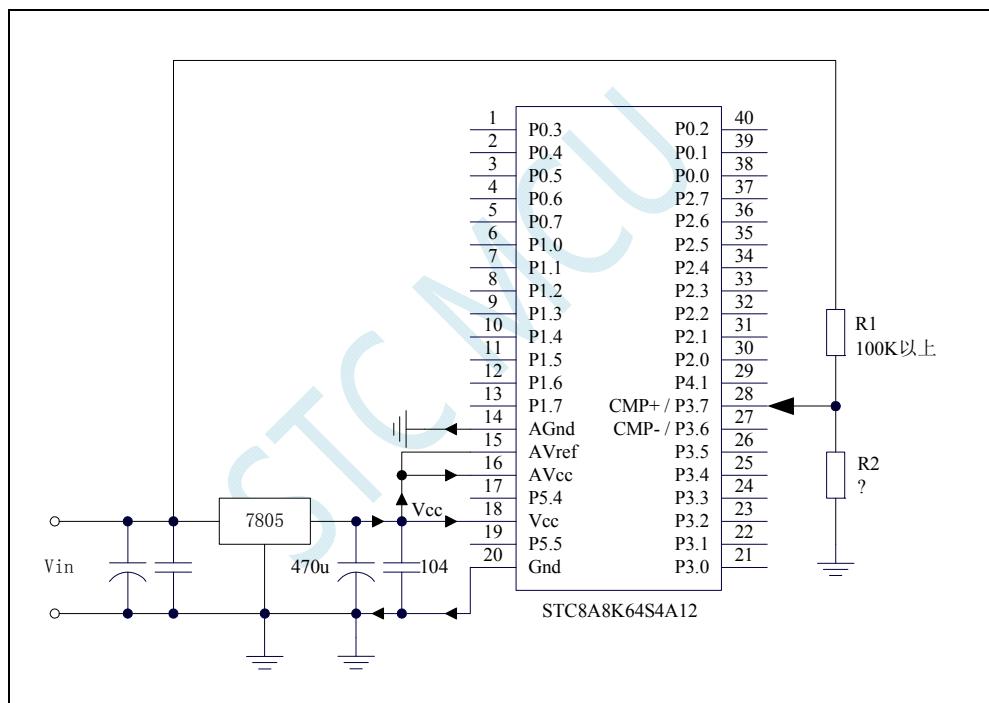
```

// CMPCRI |= 0x20;           //使能比较器上升沿中断
// CMPCRI &= ~0x10;         //禁止比较器下降沿中断
// CMPCRI |= 0x10;           //使能比较器下降沿中断
CMPCRI &= ~0x08;        //P3.7 为 CMP+ 输入脚
// CMPCRI |= 0x08;           //ADC 输入脚为 CMP+ 输入脚
// CMPCRI &= ~0x04;          //内部参考电压为 CMP- 输入脚
CMPCRI |= 0x04;          //P3.6 为 CMP- 输入脚
// CMPCRI &= ~0x02;          //禁止比较器输出
CMPCRI |= 0x02;          //使能比较器输出
CMPCRI |= 0x80;          //使能比较器模块

while (1)
{
    P10 = CMPCRI & 0x01;      //读取比较器比较结果
}
}

```

14.3.3 比较器作外部掉电检测



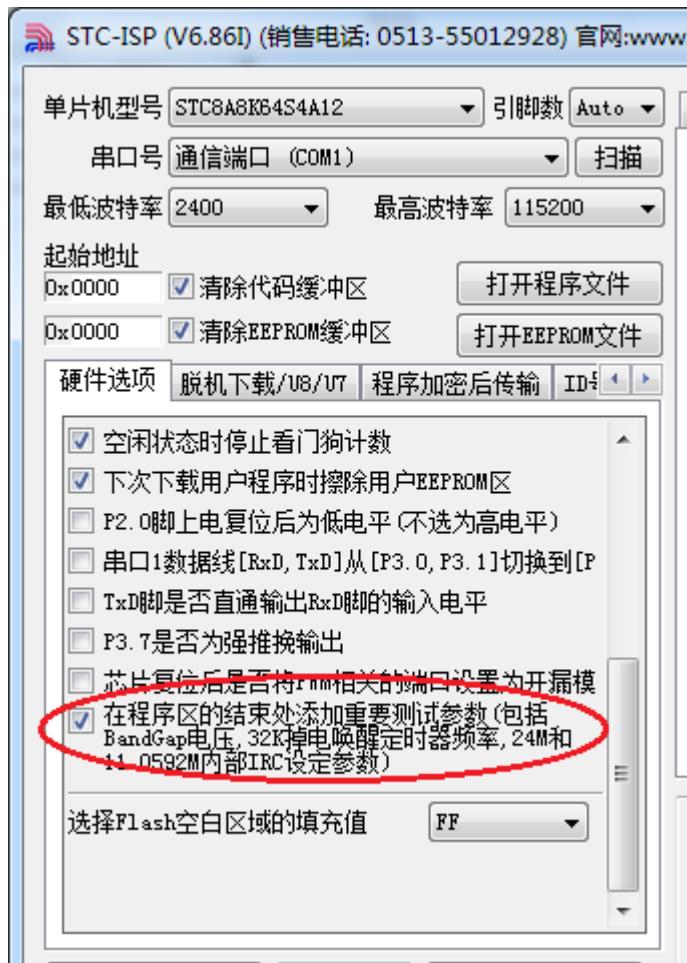
上图中电阻 R1 和 R2 对稳压块 7805 的前端电压进行分压，分压后的电压作为比较器 CMP+的外部输入与内部参考电压（约 1.344V 附近）进行比较。

一般当交流电在 220V 时，稳压块 7805 前端的直流电压为 11V，但当交流电压降到 160V 时，稳压块 7805 前端的直留电压为 8.5V。当稳压块 7805 前端的直留电压低于或等于 8.5V 时，该前端输入的直留电压被电阻 R1 和 R2 分压到比较器正极输入端 CMP+，CMP+端输入电压低于内部参考电压，此时可产生比较器中断，这样在掉电检测时就有充足的时间将数据保存到 EEPROM 中。当稳压块 7805 前端的直留电压高高于 8.5V 时，该前端输入的直留电压被电阻 R1 和 R2 分压到比较器正极输入端 CMP+，CMP+ 端输入电压高于内部参考电压，此时 CPU 可继续正常工作。

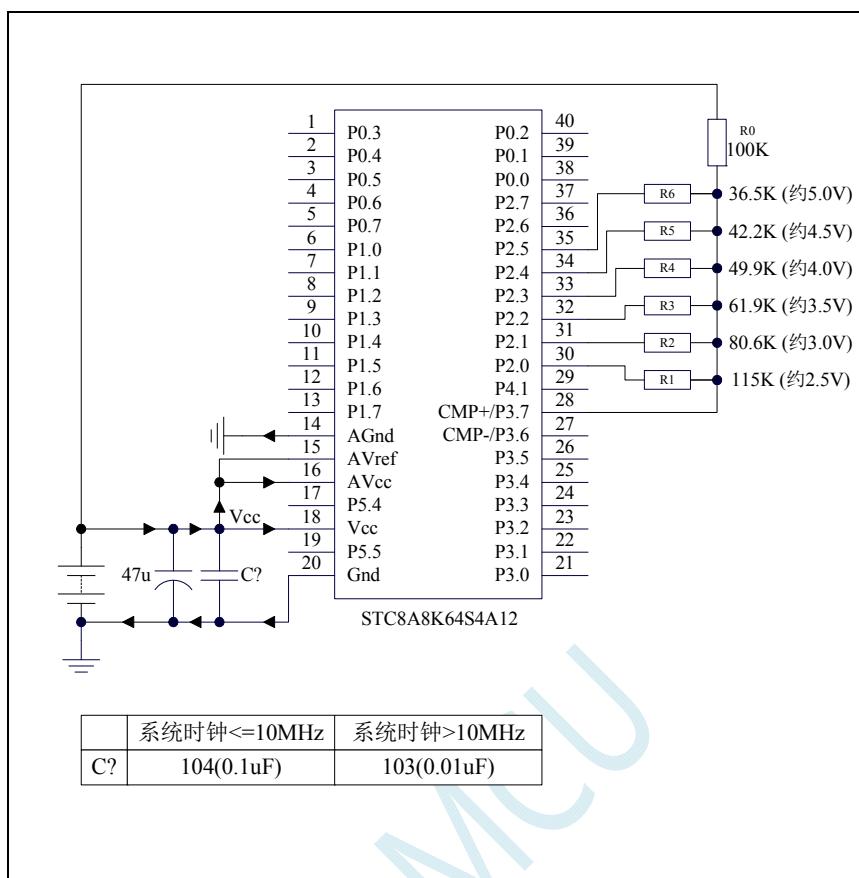
内部参考电压即为内部 BandGap 经过 OP 后的电压 REFV，REFV 的电压值约在 1.344V 附近，由于制造误差，实际电压值可能在 1.34V~1.35V 之间。具体的数值要通过读取内部参考电压在内部 RAM 区或者 ROM 区所占用的地址的值获得。对于 STC8 系列，内部参考电压值在 RAM 和 ROM 中的存储地址如下表所示：

单片机型号	RAM 中存储的地址 (高字节在前)	ROM 中存储的地址 (高字节在前)
STC8A8K16S4A12/STC8A4K16S4A12 STC8F2K16S4/STC8F2K16S2	0EFH-0F0H	3FF7H-3FF8H
STC8A8K32S4A12/STC8A4K32S4A12 STC8F2K32S4/STC8F2K32S2	0EFH-0F0H	7FF7H-7FF8H
STC8A8K60S4A12/STC8A4K60S4A12 STC8F2K60S4/STC8F2K60S2	0EFH-0F0H	EFF7H-EFF8H
STC8A8K64S4A12/STC8A4K64S4A12 STC8F2K64S4/STC8F2K64S2	0EFH-0F0H	FDF7H-FDF8H

注: 若需要从 ROM 中读取参考电压值, 则需要在进行 ISP 下载时勾选下列选项



14.3.4 比较器检测工作电压（电池电压）



上图中，利用电阻分压的原理可以近似的测量出 MCU 的工作电压（选通的通道，MCU 的 IO 口输出低电平，端口电压值接近 GND，未选通的通道，MCU 的 IO 口输出开漏模式的高，不影响其他通道）。

比较器的负端选择内部参考电压（约 1.344V），正端选择通过电阻分压后输入到 CMP+管脚的电压值。

初始化时 P2.5~P2.0 口均设置为开漏模式，并输出高。首先 P2.0 口输出低电平，此时若 VCC 电压低于 2.5V 则比较器的比较值为 0，反之若 VCC 电压高于 2.5V 则比较器的比较值为 1；

若确定 VCC 高于 2.5V，则将 P2.0 口输出高，P2.1 口输出低电平，此时若 VCC 电压低于 3.0V 则比较器的比较值为 0，反之若 VCC 电压高于 3.0V 则比较器的比较值为 1；

若确定 VCC 高于 3.0V，则将 P2.1 口输出高，P2.2 口输出低电平，此时若 VCC 电压低于 3.5V 则比较器的比较值为 0，反之若 VCC 电压高于 3.5V 则比较器的比较值为 1；

若确定 VCC 高于 3.5V，则将 P2.2 口输出高，P2.3 口输出低电平，此时若 VCC 电压低于 4.0V 则比较器的比较值为 0，反之若 VCC 电压高于 4.0V 则比较器的比较值为 1；

若确定 VCC 高于 4.0V，则将 P2.3 口输出高，P2.4 口输出低电平，此时若 VCC 电压低于 4.5V 则比较器的比较值为 0，反之若 VCC 电压高于 4.5V 则比较器的比较值为 1；

若确定 VCC 高于 4.5V，则将 P2.4 口输出高，P2.5 口输出低电平，此时若 VCC 电压低于 5.0V 则比较器的比较值为 0，反之若 VCC 电压高于 5.0V 则比较器的比较值为 1。

汇编代码

CMPCR1	DATA	0E6H
CMPCR2	DATA	0E7H
P2M0	DATA	96H

P2M1	DATA	95H	
	ORG	0000H	
	LJMP	MAIN	
	ORG	0100H	
MAIN:	MOV	SP,#3FH	
	MOV	P2M0,#0011111B	;P2.5~P2.0 初始化为开漏模式
	MOV	P2M1,#0011111B	
	MOV	P2,#0FFH	
	MOV	CMPCR2,#10H	;比较器结果经过 16 个去抖时钟后输出
	MOV	CMPCR1,#00H	
	ANL	CMPCR1,#NOT 08H	;P3.7 为 CMP+ 输入脚
	ANL	CMPCR1,#NOT 04H	;内部参考电压为 CMP- 输入脚
	ANL	CMPCR1,#NOT 02H	;禁止比较器输出
	ORL	CMPCR1,#80H	;使能比较器模块
LOOP:	MOV	R0,#0000000B	;电压<2.5V
	MOV	P2,#1111110B	;P2.0 输出 0
	CALL	DELAY	
	MOV	A,CMPCR1	
	JNB	ACC.0,SKIP	
	MOV	R0,#00000001B	;电压>2.5V
	MOV	P2,#11111101B	;P2.1 输出 0
	CALL	DELAY	
	MOV	A,CMPCR1	
	JNB	ACC.0,SKIP	
	MOV	R0,#00000011B	;电压>3.0V
	MOV	P2,#111111011B	P2.2 输出 0
	CALL	DELAY	
	MOV	A,CMPCR1	
	JNB	ACC.0,SKIP	
	MOV	R0,#00000111B	;电压>3.5V
	MOV	P2,#11110111B	;P2.3 输出 0
	CALL	DELAY	
	MOV	A,CMPCR1	
	JNB	ACC.0,SKIP	
	MOV	R0,#00001111B	;电压>4.0V
	MOV	P2,#11101111B	;P2.4 输出 0
	CALL	DELAY	
	MOV	A,CMPCR1	
	JNB	ACC.0,SKIP	
	MOV	R0,#00011111B	;电压>4.5V
	MOV	P2,#11011111B	;P2.5 输出 0
	CALL	DELAY	
	MOV	A,CMPCR1	
	JNB	ACC.0,SKIP	
	MOV	R0,#00111111B	;电压>5.0V
SKIP:	MOV	P2,#1111111B	
	MOV	A,R0	
	CPL	A	
	MOV	P0,A	;P0.5~P0.0 口显示电压
	JMP	LOOP	
DELAY:	MOV	R0,#20	

**DJNZ R0,\$
RET**

END

C 语言代码

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

sfr CMPCR1 = 0xe6;
sfr CMPCR2 = 0xe7;

sfr P2M0 = 0x96;
sfr P2M1 = 0x95;

void delay()
{
    char i;

    for (i=0; i<20; i++);
}

void main()
{
    unsigned char v;

    P2M0 = 0x3f; // P2.5~P2.0 初始化为开漏模式
    P2M1 = 0x3f;
    P2 = 0xff;

    CMPCR2 = 0x10; // 比较器结果经过16个去抖时钟后输出
    CMPCR1 = 0x00;
    CMPCR1 &= ~0x08; // P3.6 为 CMP+ 输入脚
    CMPCR1 &= ~0x04; // 内部参考电压为 CMP- 输入脚
    CMPCR1 &= ~0x02; // 禁止比较器输出
    CMPCR1 |= 0x80; // 使能比较器模块

    while (1)
    {
        v = 0x00; // 电压<2.5V
        P2 = 0xfe; // P2.0 输出0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x01; // 电压>2.5V
        P2 = 0xfd; // P2.1 输出0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x03; // 电压>3.0V
        P2 = 0xfb; // P2.2 输出0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x07; // 电压>3.5V
        P2 = 0xf7; // P2.3 输出0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x0f; // 电压>4.0V
        P2 = 0xef; // P2.4 输出0
        delay();
    }
}
```

```
if (!(CMPCR1 & 0x01)) goto ShowVol;  
v = 0x1f; //电压>4.5V  
P2 = 0xdf; //P2.5 输出0  
delay();  
if (!(CMPCR1 & 0x01)) goto ShowVol;  
v = 0x3f; //电压>5.0V  
ShowVol:  
P2 = 0xff;  
P0 = ~v;  
}  
}
```

15 IAP/EEPROM

STC8 系列单片机内部集成了大容量的 EEPROM。利用 ISP/IAP 技术可将内部 Data Flash 当 EEPROM，擦写次数在 10 万次以上。EEPROM 可分为若干个扇区，每个扇区包含 512 字节。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。在工作电压偏低时，建议不要进行 EEPROM 操作，以免发送数据丢失的情况。

15.1 EEPROM相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
IAP_DATA	IAP 数据寄存器	C2H									1111,1111
IAP_ADDRH	IAP 高地址寄存器	C3H									0000,0000
IAP_ADDRL	IAP 低地址寄存器	C4H									0000,0000
IAP_CMD	IAP 命令寄存器	C5H	-	-	-	-	-	-	-	CMD[1:0]	xxxx,xx00
IAP_TRIG	IAP 触发寄存器	C6H									0000,0000
IAP_CONTR	IAP 控制寄存器	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	-	-	IAP_WT[2:0]	0000,x000

EEPROM 数据寄存器 (IAP_DATA)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_DATA	C2H								

在进行 EEPROM 的读操作时，命令执行完成后读出的 EEPROM 数据保存在 IAP_DATA 寄存器中。在进行 EEPROM 的写操作时，在执行写命令前，必须将待写入的数据存放在 IAP_DATA 寄存器中，再发送写命令。擦除 EEPROM 命令与 IAP_DATA 寄存器无关。

EEPROM 地址寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_ADDRH	C3H								
IAP_ADDRL	C4H								

EEPROM 进行读、写、擦除操作的目标地址寄存器。IAP_ADDRH 保存地址的高字节，IAP_ADDRL 保存地址的低字节

EEPROM 命令寄存器 (IAP_CMD)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CMD	C5H	-	-	-	-	-	-	-	CMD[1:0]

CMD[1:0]: 发送EEPROM操作命令

- 00: 空操作
- 01: 读 EEPROM 命令。读取目标地址所在的 1 字节。
- 10: 写 EEPROM 命令。写目标地址所在的 1 字节。
- 11: 擦除 EEPROM。擦除目标地址所在的 1 页（1 扇区/512 字节）。

EEPROM 触发寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_TRIG	C6H								

设置完成 EEPROM 读、写、擦除的命令寄存器、地址寄存器、数据寄存器以及控制寄存器后，需要向触发寄存器 IAP_TRIG 依次写入 5AH、A5H（顺序不能交换）两个触发命令来触发相应的读、写、擦除操作。操作完成后，EEPROM 地址寄存器 IAP_ADDRH、IAP_ADDRL 和 EEPROM 命令寄存器 IAP_CMD 的内容不变。如果接下来要对下一个地址的数据进行操作，需手动更新地址寄存器 IAP_ADDRH 和寄存器 IAP_ADDRL 的值。

注意：每次 EEPROM 操作时，都要对 IAP_TRIG 先写入 5AH，再写入 A5H，相应的命令才会生效。写完触发命令后，CPU 会处于 IDLE 等待状态，直到相应的 IAP 操作执行完成后 CPU 才会从 IDLE 状态返回正常状态继续执行 CPU 指令。

EEPROM 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CONTR	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-		IAP_WT[2:0]	

IAPEN: EEPROM 操作使能控制位

- 0: 禁止 EEPROM 操作
- 1: 使能 EEPROM 操作

SWBS: 软件复位选择控制位，(需要与 SWRST 配合使用)

- 0: 软件复位后从用户代码开始执行程序
- 1: 软件复位后从系统 ISP 监控代码区开始执行程序

SWRST: 软件复位控制位

- 0: 无动作
- 1: 产生软件复位

CMD_FAIL: EEPROM 操作失败状态位，需要软件清零

- 0: EEPROM 操作正确
- 1: EEPROM 操作失败

IAP_WT[2:0]: 设置 EEPROM 操作的等待时间

IAP_WT[2:0]			读字节 (2 个时钟)	写字节 (约 7us)	擦除扇区 (约 5ms)	时钟频率
1	1	1	2 个时钟	7 个时钟	5000 个时钟	1MHz
1	1	0	2 个时钟	14 个时钟	10000 个时钟	2MHz
1	0	1	2 个时钟	21 个时钟	15000 个时钟	3MHz
1	0	0	2 个时钟	42 个时钟	30000 个时钟	6MHz
0	1	1	2 个时钟	84 个时钟	60000 个时钟	12MHz
0	1	0	2 个时钟	140 个时钟	100000 个时钟	20MHz
0	0	1	2 个时钟	168 个时钟	120000 个时钟	24MHz
0	0	0	2 个时钟	301 个时钟	215000 个时钟	30MHz

此时 MCU 系统不给 CPU 供应时钟，CPU 没有时钟，所以无法工作，也就是说，针对 EEPROM 操作所需要的等待时间是硬件自动完成的，用户不需要加额外的软件延时。但 MCU 给串口、定时器、SPI、I2C 等外设供应时钟，故除 CPU 无法执行指令外，其他外设仍然继续工作。

EEPROM 的读操作其实可不用 IAP 读方式，可用 MOVC 指令进行读取，CPU 就可继续执行指令，不用等待两个时钟。(汇编、C 如何读待后续介绍)

15.2 关于EEPROM编程和擦除等待时间的重要说明

表一 (STC8A 系列和 STC8F 系列 EEPROM 操作时间需求)

EEPROM 操作	最短时间	最长时间
编程	6us	7.5us
擦除	4ms	6ms

表二 (STC8A 系列和 STC8F 系列 EEPROM 操作相应等待参数的时间等待周期)

IAP_WT[2:0]			编程等待时钟数	擦除等待时钟数	适合的频率
1	1	1	7 个时钟	5000 个时钟	1MHz
1	1	0	14 个时钟	10000 个时钟	2MHz
1	0	1	21 个时钟	15000 个时钟	3MHz
1	0	0	42 个时钟	30000 个时钟	6MHz
0	1	1	84 个时钟	60000 个时钟	12MHz
0	1	0	140 个时钟	100000 个时钟	20MHz
0	0	1	168 个时钟	120000 个时钟	24MHz
0	0	0	301 个时钟	215000 个时钟	30MHz

STC8A 系列和 STC8F 系列 MCU 的内部 EEPROM 的编程和擦除等待时间必须达到表一中的要求，等待时间不可过短，也不可过长。

编程的等待时间必须在 6us~7.5us 之间，编程等待时间过小（小于最短时间 6us），则被编程的目标存储单元内部的数据可能不可靠（数据的保存期限可能达不到 25 年）；若等待时间过长（大于最长时间 7.5us 的 1.5 倍，即大于 11.25us），也可能由于有数据干扰而导致写入的数据不正确。在确保编程的等待时间要求，并在编程完成后进行数据读出对校验，若校验正确，数据便编程正确了。

擦除的等待时间必须在 4ms~6ms 之间，擦除等待时间过小（小于最短时间 4ms），则被擦除的目标存储扇区可能没有被擦除干净；若等待时间过长（大于最长时间 6ms 的 1.5 倍，即大于 9ms），则会缩短 EEPROM 的使用寿命，即原本 10 万次的擦除寿命可能会缩短为 5 万次。

编程与擦除的等待时间请严格按照表二所给的推荐频率进行合适的选择，假如工作频率为 12MHz，请按照表二推荐将等待参数设置为 011B，若 CPU 实际的工作频率并不在表二所推荐的频率之列，则需要根据实际的频率以及表二中实际的等待时钟数进行计算，找出满足表一时间需求的等待时间参数。

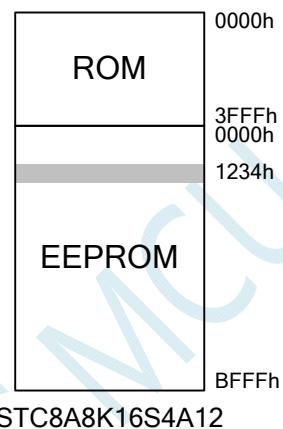
例如：工作频率为 4MHz，若选择等待参数为 101B，则编程时间为 $21/4\text{MHz} = 5.25\text{us}$ ，擦除时间为 $15000/4\text{MHz} = 3.75\text{ms}$ ，时间明显不够，所以应该选择等待参数为 100B，则编程时间为 $42/4\text{MHz} = 10.5\text{us}$ ，擦除时间为 $30000/4\text{MHz} = 7.5\text{ms}$ ，时间均在最短时间和最长时间的 1.5 倍之间。

注意：EEPROM 等待操作的时钟是指对主时钟进行分频后的系统时钟，即 CPU 实际的工作时钟。若单片机使用的是内部高精度 IRC，则 EEPROM 等待操作的时钟为使用 ISP 下载软件下载时经过调节后的频率；若单片机使用的外部晶振，则 EEPROM 等待操作的时钟为外部晶振频率经过 CLKDIV 寄存器分频后的时钟（例如：若单片机使用外部晶振，且外部晶振的频率为 24MHz，CLKDIV 寄存器的值设置为 4，则 EEPROM 等待操作的时钟频率为 $24\text{MHz}/4 = 6\text{MHz}$ ，此时等待参数应选择 100B，而不能选择 001B）。

15.3 EEPROM大小及地址

STC8 系列单片机内部均有用于保存用户数据的 EEPROM。内部的 EEPROM 有 3 操作方式：读、写和擦除，其中擦除操作是以扇区为单位进行操作，每扇区为 512 字节，即每执行一次擦除命令就会擦除一个扇区，而读数据和写数据都是以字节为单位进行操作的，即每执行一次读或者写命令时只能读出或者写入一个字节。

STC8 系列单片机内部的 EEPROM 的访问方式有两种：IAP 方式和 MOVC 方式。IAP 方式可对 EEPROM 执行读、写、擦除操作，但 MOVC 只能对 EEPROM 进行读操作，而不能进行写和擦除操作。无论是使用 IAP 方式还是使用 MOVC 方式访问 EEPROM，首先都需要设置正确的目标地址。IAP 方式时，目标地址与 EEPROM 实际的物理地址是一致的，均是从地址 0000H 开始访问，但若要使用 MOVC 指令进行读取 EEPROM 数据时，目标地址必须是在 EEPROM 实际的物理地址的基础上还有加上程序大小的偏移。下面以 STC8A8K16S4A12 这个型号为例，对目标地址进行详细说明：



STC8A8K16S4A12 的程序空间为 16K 字节(0000h~3FFh)，EEPROM 空间为 48K(0000h~BFFFh)。当需要对 EEPROM 物理地址 1234h 的单元进行读、写、擦除时，若使用 IAP 方式进行访问时，设置的目标地址为 1234h，即 IAP_ADDRH 设置 12h，IAP_ADDRL 设置 34h，然后设置相应的触发命令即可对 1234h 单元进行正确操作了。但若是使用 MOVC 方式读取 EEPROM 的 1234h 单元，则必须在 1234h 的基础上还有加上 ROM 空间的大小 4000h，即必须将 DPTR 设置为 5234h，然后才能使用 MOVC 指令进行读取。

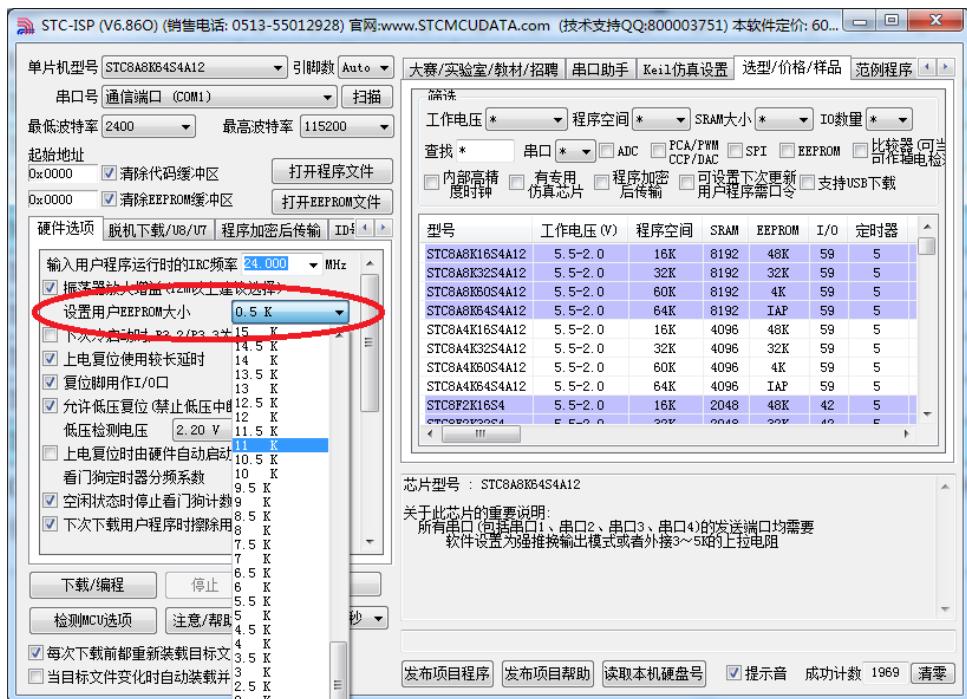
注意：由于擦除是以 512 字节为单位进行操作的，所以执行擦除操作时所设置的目标地址的低 9 位是无意义的。例如：执行擦除命令时，设置地址 1234H/1200H/1300H/13FFH，最终执行擦除的动作都是相同的，都是擦除 1200H~13FFH 这 512 字节。

不同型号内部 EEPROM 的大小及访问地址会存在差异，针对各个型号 EEPROM 的详细大小和地址请参考下表

型号	大小	扇区	IAP 方式读/写/擦除		MOVC 读取	
			起始地址	结束地址	起始地址	结束地址
STC8A8K08S4A12	56K	112	0000h	DFFFh	2000h	FFFFh
STC8A8K16S4A12	48K	96	0000h	BFFFh	4000h	FFFFh
STC8A8K24S4A12	40K	80	0000h	9FFFh	6000h	FFFFh
STC8A8K32S4A12	32K	64	0000h	7FFFh	8000h	FFFFh
STC8A8K40S4A12	24K	48	0000h	5FFFh	A000h	FFFFh

STC8A8K48S4A12	16K	32	0000h	3FFFh	C000h	FFFFh
STC8A8K56S4A12	8K	16	0000h	1FFFh	E000h	FFFFh
STC8A8K60S4A12	4K	8	0000h	0FFFh	F000h	FFFFh
STC8A8K64S4A12	用户自定义 ^[1]					
STC8A4K08S2A12	56K	112	0000h	DFFFh	2000h	FFFFh
STC8A4K16S2A12	48K	96	0000h	BFFFh	4000h	FFFFh
STC8A4K24S2A12	40K	80	0000h	9FFFh	6000h	FFFFh
STC8A4K32S2A12	32K	64	0000h	7FFFh	8000h	FFFFh
STC8A4K40S2A12	24K	48	0000h	5FFFh	A000h	FFFFh
STC8A4K48S2A12	16K	32	0000h	3FFFh	C000h	FFFFh
STC8A4K56S2A12	8K	16	0000h	1FFFh	E000h	FFFFh
STC8A4K60S2A12	4K	8	0000h	0FFFh	F000h	FFFFh
STC8A4K64S2A12	用户自定义 ^[1]					
STC8F2K08S4	56K	112	0000h	DFFFh	2000h	FFFFh
STC8F2K16S4	48K	96	0000h	BFFFh	4000h	FFFFh
STC8F2K24S4	40K	80	0000h	9FFFh	6000h	FFFFh
STC8F2K32S4	32K	64	0000h	7FFFh	8000h	FFFFh
STC8F2K40S4	24K	48	0000h	5FFFh	A000h	FFFFh
STC8F2K48S4	16K	32	0000h	3FFFh	C000h	FFFFh
STC8F2K56S4	8K	16	0000h	1FFFh	E000h	FFFFh
STC8F2K60S4	4K	8	0000h	0FFFh	F000h	FFFFh
STC8F2K64S4	用户自定义 ^[1]					
STC8F2K08S2	56K	112	0000h	DFFFh	2000h	FFFFh
STC8F2K16S2	48K	96	0000h	BFFFh	4000h	FFFFh
STC8F2K24S2	40K	80	0000h	9FFFh	6000h	FFFFh
STC8F2K32S2	32K	64	0000h	7FFFh	8000h	FFFFh
STC8F2K40S2	24K	48	0000h	5FFFh	A000h	FFFFh
STC8F2K48S2	16K	32	0000h	3FFFh	C000h	FFFFh
STC8F2K56S2	8K	16	0000h	1FFFh	E000h	FFFFh
STC8F2K60S2	4K	8	0000h	0FFFh	F000h	FFFFh
STC8F2K64S2	用户自定义 ^[1]					
STC8F1K02S2	10K	20	0000h	27FFh	0800h	2FFFh
STC8F1K04S2	8K	16	0000h	1FFFh	1000h	2FFFh
STC8F1K06S2	6K	12	0000h	17FFh	1800h	2FFFh
STC8F1K08S2	4K	8	0000h	0FFFh	2000h	2FFFh
STC8F1K10S2	2K	4	0000h	07FFh	2800h	2FFFh
STC8F1K12S2	用户自定义 ^[1]					
STC8F1K17S2	用户自定义 ^[1]					

^[1]: STC8A8K64S4A12 / STC8A4K64S2A12 / STC8F2K64S4 / STC8F2K64S2 / STC8F1K12S2 / STC8F1K17S2 这几个为特殊型号,这个几个型号的 EEPROM 大小是可用在 ISP 下载时用户自己设置的。如下图所示:



用户可根据自己的需要在整个 FLASH 空间中规划出任意不超过 FLASH 大小的 EEPROM 空间，但需要注意：**EEPROM 总是从后向前进行规划的。**

例如：STC8A8K64S4A12 这个型号的 FLASH 为 64K，此时若用户想分出其中的 8K 作为 EEPROM 使用，则 EEPROM 的物理地址则为 64K 的最后 8K，物理地址为 E000h~FFFFh，当然，用户若使用 IAP 的方式进行访问，目标地址仍然从 0000h 开始，到 1FFFh 结束，当使用 MOVC 读取则需要从 E000h 开始，到 FFFFh 结束。

15.4 范例程序

15.4.1 EEPROM基本操作

汇编代码

; 测试工作频率为 11.0592MHz

```

IAP_DATA      DATA      0C2H
IAP_ADDRH     DATA      0C3H
IAP_ADDRL     DATA      0C4H
IAP_CMD       DATA      0C5H
IAP_TRIG      DATA      0C6H
IAP CONTR     DATA      0C7H

WT_30M        EQU       80H
WT_24M        EQU       81H
WT_20M        EQU       82H
WT_12M        EQU       83H
WT_6M         EQU       84H
WT_3M         EQU       85H
WT_2M         EQU       86H
WT_1M         EQU       87H

ORG          0000H
LJMP         MAIN

ORG          0100H

IAP_IDLE:
    MOV    IAP CONTR,#0           ; 关闭 IAP 功能
    MOV    IAP CMD,#0            ; 清除命令寄存器
    MOV    IAP TRIG,#0           ; 清除触发寄存器
    MOV    IAP ADDRH,#80H        ; 将地址设置到非 IAP 区域
    MOV    IAP ADDRL,#0
    RET

IAP_READ:
    MOV    IAP CONTR,#WT_12M    ; 使能 IAP
    MOV    IAP CMD,#1            ; 设置 IAP 读命令
    MOV    IAP ADDRL,DPL        ; 设置 IAP 低地址
    MOV    IAP ADDRH,DPH        ; 设置 IAP 高地址
    MOV    IAP TRIG,#5AH         ; 写触发命令(0x5a)
    MOV    IAP TRIG,#0A5H        ; 写触发命令(0xa5)
    NOP
    MOV    A,IAP DATA           ; 读取 IAP 数据
    LCALL   IAP_IDLE            ; 关闭 IAP 功能
    RET

IAP_PROGRAM:
    MOV    IAP CONTR,#WT_12M    ; 使能 IAP
    MOV    IAP CMD,#2            ; 设置 IAP 写命令
    MOV    IAP ADDRL,DPL        ; 设置 IAP 低地址
    MOV    IAP ADDRH,DPH        ; 设置 IAP 高地址
    MOV    IAP DATA,A           ; 写 IAP 数据
    MOV    IAP TRIG,#5AH         ; 写触发命令(0x5a)
    MOV    IAP TRIG,#0A5H        ; 写触发命令(0xa5)

```

<i>NOP</i>		
<i>LCALL</i>	<i>IAP_IDLE</i>	;关闭IAP 功能
<i>RET</i>		

IAP_ERASE:

<i>MOV</i>	<i>IAP_CONTR,#WT_12M</i>	;使能IAP
<i>MOV</i>	<i>IAP_CMD,#3</i>	;设置IAP 擦除命令
<i>MOV</i>	<i>IAP_ADDRH,DPL</i>	;设置IAP 低地址
<i>MOV</i>	<i>IAP_ADDRH,DPH</i>	;设置IAP 高地址
<i>MOV</i>	<i>IAP_TRIG,#5AH</i>	;写触发命令(0x5a)
<i>MOV</i>	<i>IAP_TRIG,#0A5H</i>	;写触发命令(0xa5)
<i>NOP</i>		
<i>LCALL</i>	<i>IAP_IDLE</i>	;关闭IAP 功能
<i>RET</i>		

MAIN:

<i>MOV</i>	<i>SP,#3FH</i>	
<i>MOV</i>	<i>DPTR,#0400H</i>	
<i>LCALL</i>	<i>IAP_ERASE</i>	
<i>MOV</i>	<i>DPTR,#0400H</i>	
<i>LCALL</i>	<i>IAP_READ</i>	
<i>MOV</i>	<i>P0,A</i>	;P0=0FFH
<i>MOV</i>	<i>DPTR,#0400H</i>	
<i>MOV</i>	<i>A,#12H</i>	
<i>LCALL</i>	<i>IAP_PROGRAM</i>	
<i>MOV</i>	<i>DPTR,#0400H</i>	
<i>LCALL</i>	<i>IAP_READ</i>	
<i>MOV</i>	<i>P1,A</i>	;P1=12H
<i>SJMP</i>	\$	

END**C 语言代码**

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

```
//测试工作频率为11.0592MHz
```

```
sfr IAP_DATA = 0xC2;
sfr IAP_ADDRH = 0xC3;
sfr IAP_ADDRL = 0xC4;
sfr IAP_CMD = 0xC5;
sfr IAP_TRIG = 0xC6;
sfr IAP CONTR = 0xC7;

#define WT_30M 0x80
#define WT_24M 0x81
#define WT_20M 0x82
#define WT_12M 0x83
#define WT_6M 0x84
#define WT_3M 0x85
#define WT_2M 0x86
#define WT_1M 0x87
```

```
void IapIdle()
{
```

```

IAP CONTR = 0;           //关闭IAP 功能
IAP CMD = 0;             //清除命令寄存器
IAP TRIG = 0;             //清除触发寄存器
IAP ADDRH = 0x80;         //将地址设置到非IAP 区域
IAP ADDRL = 0;
}

char IapRead(int addr)
{
    char dat;

    IAP CONTR = WT_I2M;      //使能IAP
    IAP CMD = 1;              //设置IAP 读命令
    IAP ADDRL = addr;          //设置IAP 低地址
    IAP ADDRH = addr >> 8;        //设置IAP 高地址
    IAP TRIG = 0x5a;            //写触发命令(0x5a)
    IAP TRIG = 0xa5;            //写触发命令(0xa5)
    _nop_();
    dat = IAP DATA;            //读IAP 数据
    IapIdle();                  //关闭IAP 功能

    return dat;
}

void IapProgram(int addr, char dat)
{
    IAP CONTR = WT_I2M;      //使能IAP
    IAP CMD = 2;              //设置IAP 写命令
    IAP ADDRL = addr;          //设置IAP 低地址
    IAP ADDRH = addr >> 8;        //设置IAP 高地址
    IAP DATA = dat;            //写IAP 数据
    IAP TRIG = 0x5a;            //写触发命令(0x5a)
    IAP TRIG = 0xa5;            //写触发命令(0xa5)
    _nop_();
    IapIdle();                  //关闭IAP 功能
}

void IapErase(int addr)
{
    IAP CONTR = WT_I2M;      //使能IAP
    IAP CMD = 3;              //设置IAP 擦除命令
    IAP ADDRL = addr;          //设置IAP 低地址
    IAP ADDRH = addr >> 8;        //设置IAP 高地址
    IAP TRIG = 0x5a;            //写触发命令(0x5a)
    IAP TRIG = 0xa5;            //写触发命令(0xa5)
    _nop_();
    IapIdle();                  //关闭IAP 功能
}

void main()
{
    IapErase(0x0400);
    P0 = IapRead(0x0400);        //P0=0xff
    IapProgram(0x0400, 0x12);
    P1 = IapRead(0x0400);        //P1=0x12

    while (1);
}

```

15.4.2 使用MOVC读取EEPROM

汇编代码

; 测试工作频率为 11.0592MHz

```

IAP_DATA      DATA      0C2H
IAP_ADDRH     DATA      0C3H
IAP_ADDRL     DATA      0C4H
IAP_CMD       DATA      0C5H
IAP_TRIG      DATA      0C6H
IAP_CONTR     DATA      0C7H

WT_30M        EQU       80H
WT_24M        EQU       81H
WT_20M        EQU       82H
WT_12M        EQU       83H
WT_6M         EQU       84H
WT_3M         EQU       85H
WT_2M         EQU       86H
WT_1M         EQU       87H

IAP_OFFSET    EQU       2000H           ;STC8A8K08S4A12
;IAP_OFFSET   EQU       4000H           ;STC8A8K16S4A12
;IAP_OFFSET   EQU       6000H           ;STC8A8K24S4A12
;IAP_OFFSET   EQU       8000H           ;STC8A8K32S4A12
;IAP_OFFSET   EQU       0A000H          ;STC8A8K40S4A12
;IAP_OFFSET   EQU       0C000H          ;STC8A8K48S4A12
;IAP_OFFSET   EQU       0E000H          ;STC8A8K56S4A12
;IAP_OFFSET   EQU       0F000H          ;STC8A8K60S4A12

ORG           0000H
LJMP          MAIN

ORG           0100H

IAP_IDLE:
    MOV    IAP_CONTR,#0           ;关闭 IAP 功能
    MOV    IAP_CMD,#0             ;清除命令寄存器
    MOV    IAP_TRIG,#0            ;清除触发寄存器
    MOV    IAP_ADDRH,#80H          ;将地址设置到非 IAP 区域
    MOV    IAP_ADDRL,#0
    RET

IAP_READ:
    MOV    A,#LOW IAP_OFFSET      ;使用 MOVC 读取 EEPROM 需要加上相应的偏移
    ADD    A,DPL
    MOV    DPL,A
    MOV    A,@HIGH IAP_OFFSET
    ADDC   A,DPH
    MOV    DPH,A
    CLR    A
    MOVC   A,@A+DPTR             ;使用 MOVC 读取数据
    RET

IAP_PROGRAM:
    MOV    IAP_CONTR,WT_12M        ;使能 IAP
    MOV    IAP_CMD,#2              ;设置 IAP 写命令
    MOV    IAP_ADDRL,DPL           ;设置 IAP 低地址

```

<i>MOV</i>	<i>IAP_ADDRH,DPH</i>	; 设置 IAP 高地址
<i>MOV</i>	<i>IAP_DATA,A</i>	; 写 IAP 数据
<i>MOV</i>	<i>IAP_TRIG,#5AH</i>	; 写触发命令(0x5a)
<i>MOV</i>	<i>IAP_TRIG,#A5H</i>	; 写触发命令(0xa5)
<i>NOP</i>		
<i>LCALL</i>	<i>IAP_IDLE</i>	; 关闭 IAP 功能
<i>RET</i>		

IAP_ERASE:

<i>MOV</i>	<i>IAP_CONTR,#WT_12M</i>	; 使能 IAP
<i>MOV</i>	<i>IAP_CMD,#3</i>	; 设置 IAP 擦除命令
<i>MOV</i>	<i>IAP_ADDRL,DPL</i>	; 设置 IAP 低地址
<i>MOV</i>	<i>IAP_ADDRH,DPH</i>	; 设置 IAP 高地址
<i>MOV</i>	<i>IAP_TRIG,#5AH</i>	; 写触发命令(0x5a)
<i>MOV</i>	<i>IAP_TRIG,#A5H</i>	; 写触发命令(0xa5)
<i>NOP</i>		
<i>LCALL</i>	<i>IAP_IDLE</i>	; 关闭 IAP 功能
<i>RET</i>		

MAIN:

<i>MOV</i>	<i>SP,#3FH</i>	
<i>MOV</i>	<i>DPTR,#0400H</i>	
<i>LCALL</i>	<i>IAP_ERASE</i>	
<i>MOV</i>	<i>DPTR,#0400H</i>	
<i>LCALL</i>	<i>IAP_READ</i>	
<i>MOV</i>	<i>P0,A</i>	; P0=0FFH
<i>MOV</i>	<i>DPTR,#0400H</i>	
<i>MOV</i>	<i>A,#12H</i>	
<i>LCALL</i>	<i>IAP_PROGRAM</i>	
<i>MOV</i>	<i>DPTR,#0400H</i>	
<i>LCALL</i>	<i>IAP_READ</i>	
<i>MOV</i>	<i>P1,A</i>	; P1=12H
<i>SJMP</i>	\$	
		<i>END</i>

C 语言代码

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

```
// 测试工作频率为 11.0592MHz
```

```
sfr IAP_DATA = 0xC2;
sfr IAP_ADDRH = 0xC3;
sfr IAP_ADDRL = 0xC4;
sfr IAP_CMD = 0xC5;
sfr IAP_TRIG = 0xC6;
sfr IAP CONTR = 0xC7;

#define WT_30M 0x80
#define WT_24M 0x81
#define WT_20M 0x82
#define WT_12M 0x83
#define WT_6M 0x84
#define WT_3M 0x85
#define WT_2M 0x86
```

```

#define WT_IM      0x87

#define IAP_OFFSET 0x2000H      //STC8A8K08S4A12
#define IAP_OFFSET 0x4000H      //STC8A8K16S4A12
#define IAP_OFFSET 0x6000H      //STC8A8K24S4A12
#define IAP_OFFSET 0x8000H      //STC8A8K32S4A12
#define IAP_OFFSET 0xA000H      //STC8A8K40S4A12
#define IAP_OFFSET 0xC000H      //STC8A8K48S4A12
#define IAP_OFFSET 0xE000H      //STC8A8K56S4A12
#define IAP_OFFSET 0xF000H      //STC8A8K60S4A12

void IapIdle()
{
    IAP_CONTR = 0;           //关闭IAP 功能
    IAP_CMD = 0;             //清除命令寄存器
    IAP_TRIG = 0;            //清除触发寄存器
    IAP_ADDRH = 0x80;        //将地址设置到非IAP 区域
    IAP_ADDRL = 0;
}

char IapRead(int addr)
{
    addr += IAP_OFFSET;     //使用MOVC 读取EEPROM 需要加上相应的偏移
    return *(char code *)(addr); //使用MOVC 读取数据
}

void IapProgram(int addr, char dat)
{
    IAP_CONTR = WT_I2M;     //使能IAP
    IAP_CMD = 2;             //设置IAP 写命令
    IAP_ADDRL = addr;        //设置IAP 低地址
    IAP_ADDRH = addr >> 8;   //设置IAP 高地址
    IAP_DATA = dat;          //写IAP 数据
    IAP_TRIG = 0x5a;         //写触发命令(0x5a)
    IAP_TRIG = 0xa5;         //写触发命令(0xa5)
    _nop_();
    IapIdle();               //关闭IAP 功能
}

void IapErase(int addr)
{
    IAP_CONTR = WT_I2M;     //使能IAP
    IAP_CMD = 3;             //设置IAP 擦除命令
    IAP_ADDRL = addr;        //设置IAP 低地址
    IAP_ADDRH = addr >> 8;   //设置IAP 高地址
    IAP_TRIG = 0x5a;         //写触发命令(0x5a)
    IAP_TRIG = 0xa5;         //写触发命令(0xa5)
    _nop_();
    IapIdle();               //关闭IAP 功能
}

void main()
{
    IapErase(0x0400);
    P0 = IapRead(0x0400);    //P0=0xff
    IapProgram(0x0400, 0x12);
    P1 = IapRead(0x0400);    //P1=0x12

    while (1);
}

```

}

15.4.3 使用串口送出EEPROM数据

汇编代码

<i>AUXR</i>	<i>DATA</i>	<i>8EH</i>
<i>T2H</i>	<i>DATA</i>	<i>0D6H</i>
<i>T2L</i>	<i>DATA</i>	<i>0D7H</i>
<i>IAP_DATA</i>	<i>DATA</i>	<i>0C2H</i>
<i>IAP_ADDRH</i>	<i>DATA</i>	<i>0C3H</i>
<i>IAP_ADDRL</i>	<i>DATA</i>	<i>0C4H</i>
<i>IAP_CMD</i>	<i>DATA</i>	<i>0C5H</i>
<i>IAP_TRIG</i>	<i>DATA</i>	<i>0C6H</i>
<i>IAP_CONTR</i>	<i>DATA</i>	<i>0C7H</i>
<i>WT_30M</i>	<i>EQU</i>	<i>80H</i>
<i>WT_24M</i>	<i>EQU</i>	<i>81H</i>
<i>WT_20M</i>	<i>EQU</i>	<i>82H</i>
<i>WT_12M</i>	<i>EQU</i>	<i>83H</i>
<i>WT_6M</i>	<i>EQU</i>	<i>84H</i>
<i>WT_3M</i>	<i>EQU</i>	<i>85H</i>
<i>WT_2M</i>	<i>EQU</i>	<i>86H</i>
<i>WT_1M</i>	<i>EQU</i>	<i>87H</i>
<i>ORG</i>	<i>0000H</i>	
<i>LJMP</i>	<i>MAIN</i>	
<i>ORG</i>	<i>0100H</i>	
<i>UART_INIT:</i>		
<i>MOV</i>	<i>SCON,#5AH</i>	
<i>MOV</i>	<i>T2L,#0E8H</i>	<i>;65536-11059200/115200/4=0FFE8H</i>
<i>MOV</i>	<i>T2H,#0FFH</i>	
<i>MOV</i>	<i>AUXR,#15H</i>	
<i>RET</i>		
<i>UART_SEND:</i>		
<i>JNB</i>	<i>TI,\$</i>	
<i>CLR</i>	<i>TI</i>	
<i>MOV</i>	<i>SBUFA,A</i>	
<i>RET</i>		
<i>IAP_IDLE:</i>		
<i>MOV</i>	<i>IAP_CONTR,#0</i>	<i>;关闭IAP 功能</i>
<i>MOV</i>	<i>IAP_CMD,#0</i>	<i>;清除命令寄存器</i>
<i>MOV</i>	<i>IAP_TRIG,#0</i>	<i>;清除触发寄存器</i>
<i>MOV</i>	<i>IAP_ADDRH,#80H</i>	<i>;将地址设置到非IAP 区域</i>
<i>MOV</i>	<i>IAP_ADDRL,#0</i>	
<i>RET</i>		
<i>IAP_READ:</i>		
<i>MOV</i>	<i>IAP_CONTR,#WT_12M</i>	<i>;使能IAP</i>
<i>MOV</i>	<i>IAP_CMD,#1</i>	<i>;设置IAP 读命令</i>
<i>MOV</i>	<i>IAP_ADDRL,DPL</i>	<i>;设置IAP 低地址</i>
<i>MOV</i>	<i>IAP_ADDRH,DPH</i>	<i>;设置IAP 高地址</i>
<i>MOV</i>	<i>IAP_TRIG,#5AH</i>	<i>;写触发命令(0x5a)</i>
<i>MOV</i>	<i>IAP_TRIG,#0A5H</i>	<i>;写触发命令(0xa5)</i>

<i>NOP</i>		
<i>MOV</i>	<i>A,IAP DATA</i>	; 读取 IAP 数据
<i>LCALL</i>	<i>IAP_IDLE</i>	; 关闭 IAP 功能
<i>RET</i>		

IAP_PROGRAM:

<i>MOV</i>	<i>IAP_CONTR,#WT_12M</i>	; 使能 IAP
<i>MOV</i>	<i>IAP_CMD,#2</i>	; 设置 IAP 写命令
<i>MOV</i>	<i>IAP_ADDRL,DPL</i>	; 设置 IAP 低地址
<i>MOV</i>	<i>IAP_ADDRH,DPH</i>	; 设置 IAP 高地址
<i>MOV</i>	<i>IAP_DATA,A</i>	; 写 IAP 数据
<i>MOV</i>	<i>IAP_TRIG,#5AH</i>	; 写触发命令(0x5a)
<i>MOV</i>	<i>IAP_TRIG,#0A5H</i>	; 写触发命令(0xa5)
<i>NOP</i>		
<i>LCALL</i>	<i>IAP_IDLE</i>	; 关闭 IAP 功能
<i>RET</i>		

IAP_ERASE:

<i>MOV</i>	<i>IAP_CONTR,#WT_12M</i>	; 使能 IAP
<i>MOV</i>	<i>IAP_CMD,#3</i>	; 设置 IAP 擦除命令
<i>MOV</i>	<i>IAP_ADDRL,DPL</i>	; 设置 IAP 低地址
<i>MOV</i>	<i>IAP_ADDRH,DPH</i>	; 设置 IAP 高地址
<i>MOV</i>	<i>IAP_TRIG,#5AH</i>	; 写触发命令(0x5a)
<i>MOV</i>	<i>IAP_TRIG,#0A5H</i>	; 写触发命令(0xa5)
<i>NOP</i>		
<i>LCALL</i>	<i>IAP_IDLE</i>	; 关闭 IAP 功能
<i>RET</i>		

MAIN:

<i>MOV</i>	<i>SP,#3FH</i>	
<i>LCALL</i>	<i>UART_INIT</i>	
<i>MOV</i>	<i>DPTR,#0400H</i>	
<i>LCALL</i>	<i>IAP_ERASE</i>	
<i>MOV</i>	<i>DPTR,#0400H</i>	
<i>LCALL</i>	<i>IAP_READ</i>	
<i>LCALL</i>	<i>UART_SEND</i>	
<i>MOV</i>	<i>DPTR,#0400H</i>	
<i>MOV</i>	<i>A,#12H</i>	
<i>LCALL</i>	<i>IAP_PROGRAM</i>	
<i>MOV</i>	<i>DPTR,#0400H</i>	
<i>LCALL</i>	<i>IAP_READ</i>	
<i>LCALL</i>	<i>UART_SEND</i>	
<i>SJMP</i>	\$	

END**C 语言代码**

#include "reg51.h"

#include "intrins.h"

#define	<i>FOSC</i>	11059200UL
#define	<i>BRT</i>	(65536 - FOSC / 115200 / 4)
<i>sfr</i>	<i>AUXR</i>	= 0x8e;
<i>sfr</i>	<i>T2H</i>	= 0xd6;
<i>sfr</i>	<i>T2L</i>	= 0xd7;

```
sfr IAP_DATA = 0xC2;
sfr IAP_ADDRH = 0xC3;
sfr IAP_ADDRL = 0xC4;
sfr IAP_CMD = 0xC5;
sfr IAP_TRIG = 0xC6;
sfr IAP_CONTR = 0xC7;

#define WT_30M 0x80
#define WT_24M 0x81
#define WT_20M 0x82
#define WT_12M 0x83
#define WT_6M 0x84
#define WT_3M 0x85
#define WT_2M 0x86
#define WT_1M 0x87

void UartInit()
{
    SCON = 0x5a;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x15;
}

void UartSend(char dat)
{
    while (!TI);
    TI = 0;
    SBUF = dat;
}

void IapIdle()
{
    IAP_CONTR = 0;           //关闭IAP 功能
    IAP_CMD = 0;             //清除命令寄存器
    IAP_TRIG = 0;            //清除触发寄存器
    IAP_ADDRH = 0x80;        //将地址设置到非IAP 区域
    IAP_ADDRL = 0;
}

char IapRead(int addr)
{
    char dat;

    IAP_CONTR = WT_12M;      //使能IAP
    IAP_CMD = 1;              //设置IAP 读命令
    IAP_ADDRL = addr;         //设置IAP 低地址
    IAP_ADDRH = addr >> 8;   //设置IAP 高地址
    IAP_TRIG = 0x5a;          //写触发命令(0x5a)
    IAP_TRIG = 0xa5;          //写触发命令(0xa5)
    _nop_();
    dat = IAP_DATA;           //读IAP 数据
    IapIdle();                //关闭IAP 功能

    return dat;
}

void IapProgram(int addr, char dat)
```

```
{  
    IAP_CONTR = WT_I2M;          //使能IAP  
    IAP_CMD = 2;                 //设置IAP 写命令  
    IAP_ADDRL = addr;           //设置IAP 低地址  
    IAP_ADDRH = addr >> 8;      //设置IAP 高地址  
    IAP_DATA = dat;              //写IAP 数据  
    IAP_TRIG = 0x5a;             //写触发命令(0x5a)  
    IAP_TRIG = 0xa5;             //写触发命令(0xa5)  
    _nop_();  
    IapIdle();                  //关闭IAP 功能  
  
}  
  
void IapErase(int addr)  
{  
    IAP_CONTR = WT_I2M;          //使能IAP  
    IAP_CMD = 3;                 //设置IAP 擦除命令  
    IAP_ADDRL = addr;           //设置IAP 低地址  
    IAP_ADDRH = addr >> 8;      //设置IAP 高地址  
    IAP_TRIG = 0x5a;             //写触发命令(0x5a)  
    IAP_TRIG = 0xa5;             //写触发命令(0xa5)  
    _nop_();  
    IapIdle();                  //关闭IAP 功能  
  
}  
  
void main()  
{  
    UartInit();  
    IapErase(0x0400);  
    UartSend(IapRead(0x0400));  
    IapProgram(0x0400, 0x12);  
    UartSend(IapRead(0x0400));  
  
    while (1);  
}
```

16 ADC 模数转换

STC8 系列单片机内部集成了一个 12 位 15 通道的高速 A/D 转换器（**注：第 16 通道只能用于检测内部 REFV 参考电压，REFV 的电压值为 1.344V，由于制造误差，实际电压值可能在 1.34V~1.35V 之间**）。ADC 的时钟频率为系统频率 2 分频再经过用户设置的分频系数进行再次分频（ADC 的时钟频率范围为 SYSclk/2/1~SYSclk/2/16）。每固定 16 个 ADC 时钟可完成一次 A/D 转换。**ADC 的速度最快可达 800K（即每秒可进行 80 万次模数转换）**

ADC 转换结果的数据格式有两种：左对齐和右对齐。可方便用户程序进行读取和引用。

16.1 ADC 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
ADC_CONTR	ADC 控制寄存器	BCH	ADC_POWER	ADC_START	ADC_FLAG	-	ADC_CHS[3:0]		000x,0000		
ADC_RES	ADC 转换结果高位寄存器	BDH							0000,0000		
ADC_RESL	ADC 转换结果低位寄存器	BEH							0000,0000		
ADCCFG	ADC 配置寄存器	DEH	-	-	RESFMT	-	SPEED[3:0]		xx0x,0000		

ADC 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADC_CONTR	BCH	ADC_POWER	ADC_START	ADC_FLAG	-	ADC_CHS[3:0]			

ADC_POWER: ADC 电源控制位

0: 关闭 ADC 电源

1: 打开 ADC 电源。

建议进入空闲模式和掉电模式前将 ADC 电源关闭，以降低功耗

ADC_START: ADC 转换启动控制位。写入 1 后开始 ADC 转换，转换完成后硬件自动将此位清零。

0: 无影响。即使 ADC 已经开始转换工作，写 0 也不会停止 A/D 转换。

1: 开始 ADC 转换，转换完成后硬件自动将此位清零。

ADC_FLAG: ADC 转换结束标志位。当 ADC 完成一次转换后，硬件会自动将此位置 1，并向 CPU 提出中断请求。此标志位必须软件清零。

ADC_CHS[3:0]: ADC 模拟通道选择位

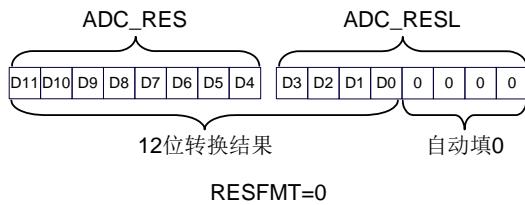
ADC_CHS[3:0]	ADC 通道	ADC_CHS[3:0]	ADC 通道
0000	P1.0	1000	P0.0
0001	P1.1	1001	P0.1
0010	P1.2	1010	P0.2
0011	P1.3	1011	P0.3
0100	P1.4	1100	P0.4
0101	P1.5	1101	P0.5
0110	P1.6	1110	P0.6
0111	P1.7	1111	测试内部 1.344V 的 REFV 电压

ADC 配置寄存器

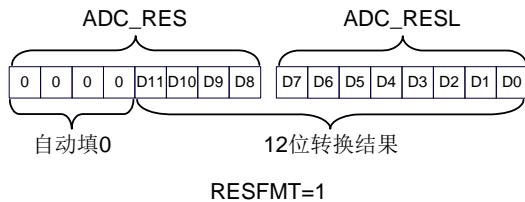
符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADCCFG	DEH	-	-	RESFMT	-	SPEED[3:0]			

RESFMT: ADC 转换结果格式控制位

0: 转换结果左对齐。ADC_RES 保存结果的高 8 位, ADC_RESL 保存结果的低 4 位。格式如下:



1: 转换结果右对齐。ADC_RES 保存结果的高 4 位, ADC_RESL 保存结果的低 8 位。格式如下:



SPEED[3:0]: ADC 时钟控制 ($F_{ADC} = SYSelk/2/16/SPEED$)

SPEED[3:0]	ADC 转换时间 (CPU 时钟数)	SPEED[3:0]	ADC 转换时间 (CPU 时钟数)
0000	32	1000	288
0001	64	1001	320
0010	96	1010	352
0011	128	1011	384
0100	160	1100	416
0101	192	1101	448
0110	224	1110	480
0111	256	1111	512

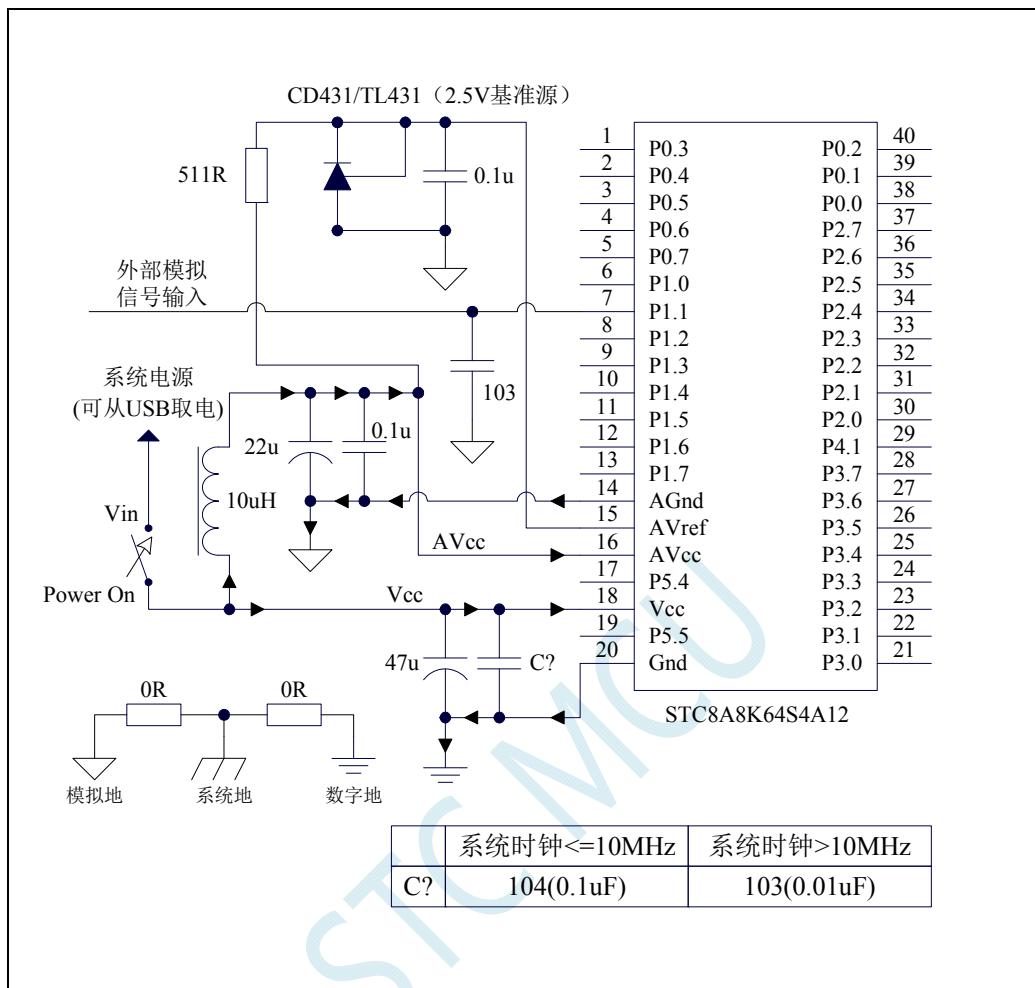
ADC 转换结果寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADC_RES	BDH								
ADC_RESL	BEH								

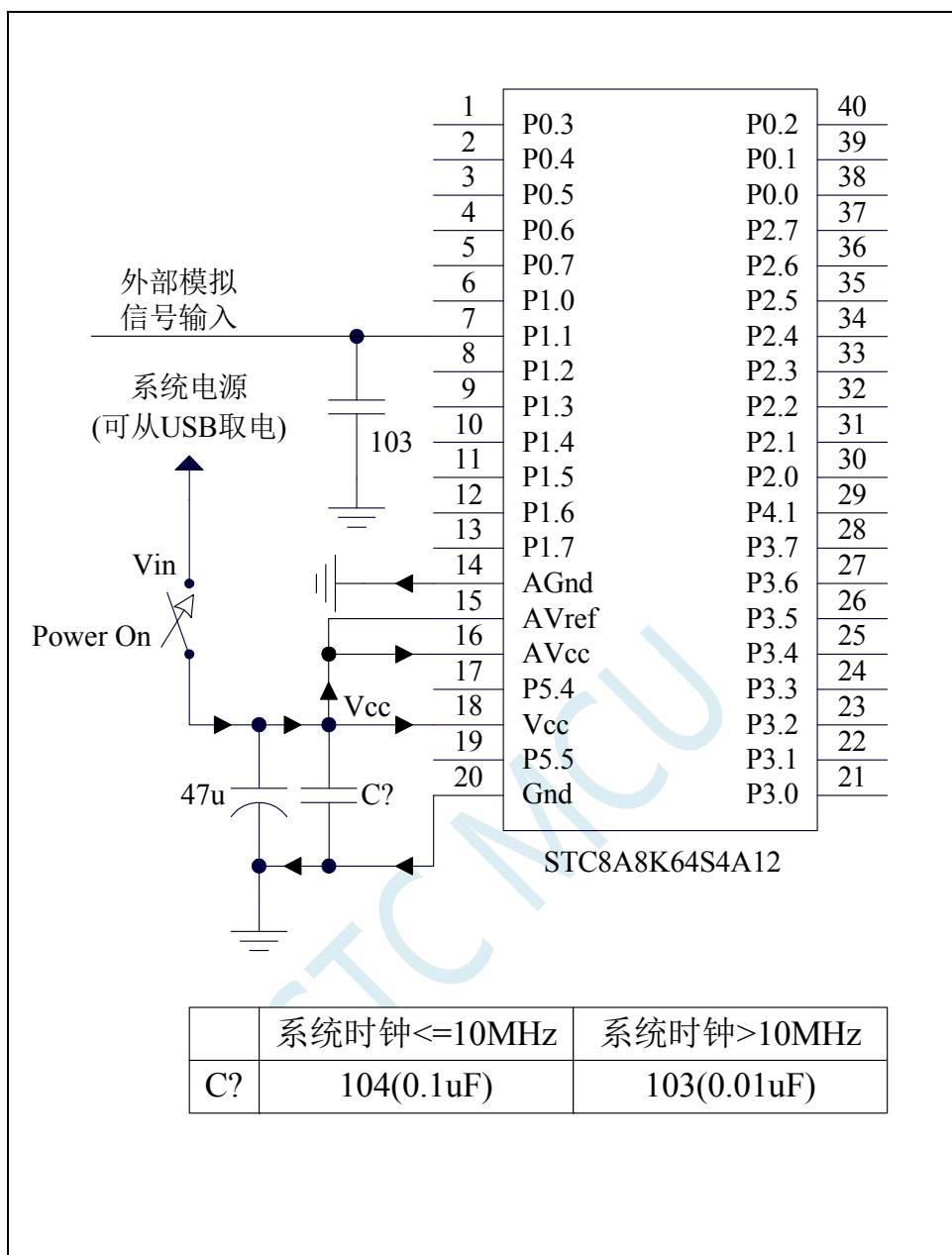
当 A/D 转换完成后, 12 位的转换结果会自动保存到 ADC_RES 和 ADC_RESL 中。保存结果的数据格式请参考 ADC_CFG 寄存器中的 RESFMT 设置。

16.2 ADC典型应用线路图

16.2.1 高精度ADC应用



16.2.2 ADC一般应用（对ADC精度要求不高的应用）



16.3 ADC线性参数

符号	描述	最小值	典型值	最大值	单位
Temp	测试温度	-40	25	85	℃
A _{in}	模拟输入范围	REFN	-	REFP	V
V _{REFP}	REFP 参考电压	1.2	3.0	5.5	V
V _{REFN}	REFN 参考电压	0	0	0	V
RES	分辨率	-	12	-	Bits
OFFSET	偏移误差	-3.0	±1.5	3.0	LSB
GAIN	增益误差		±2	±5	LSB
INL	积分非线性误差	-3.0	±1.0	2.0	LSB
DNL	微分非线性误差	-1.0	±0.9	1.5	LSB
SPS	采样率(转换速度)	30	-	800	KSPS
I _{REFP}	参考电压信号电流	-	100	-	uA
SNDR	信噪比失真率	-	64	-	dB
THD	总谐波失真	-	65	-	dB
SFDR	杂散动态范围	-	64	-	dB
R _{REFP}	REFP 输入等效电阻	-	700	-	Ω
R _{in}	模拟输入等效电阻	-	500	-	Ω
C _{in}	模拟输入等效电容	-	26	30	pF
C _{load}	数字输出负载电容	-	-	0.1	pF

注意事项:

- 使用 ADC 时, 要注意下面的要求: AVCC 跟 VCC 电压差不要超过 0.1V, VREF 在 2.0~AVCC 之间, P1.0~P1.7 P0.0~P0.6 电压不能比 AVCC 高。
- 使用外部晶振时, 要注意下面的要求: AVCC 跟 VCC 电压差不要超过 0.1V, P1.0~P1.7 P0.0~P0.6 电压不能比 AVCC 高, 否则晶振不起振, 形同死机。

16.4 范例程序

16.4.1 ADC基本操作（查询方式）

汇编代码

; 测试工作频率为 11.0592MHz

```

ADC_CONTR    DATA      0BCH
ADC_RES      DATA      0BDH
ADC_RESL     DATA      0BEH
ADCCFG       DATA      0DEH

PIM0         DATA      092H
PIM1         DATA      091H

        ORG      0000H
        LJMP    MAIN

MAIN:
        ORG      0100H
        MOV     SP,#3FH

        MOV     PIM0,#00H          ; 设置 P1.0 为 ADC 口
        MOV     PIM1,#01H
        MOV     ADCCFG,#0FH        ; 设置 ADC 时钟为系统时钟/2/16/16
        MOV     ADC_CONTR,#80H      ; 使能 ADC 模块

LOOP:
        ORL     ADC_CONTR,#40H      ; 启动 AD 转换
        NOP
        NOP
        MOV     A,ADC_CONTR        ; 查询 ADC 完成标志
        JNB     ACC.5,$-2
        ANL     ADC_CONTR,#NOT 20H   ; 清完成标志
        MOV     P2,ADC_RES          ; 读取 ADC 结果

        SJMP    LOOP

END

```

C 语言代码

```

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

// 测试工作频率为 11.0592MHz

sfr    ADC_CONTR  =  0xbc;
sfr    ADC_RES    =  0xbd;
sfr    ADC_RESL   =  0xbe;
sfr    ADCCFG    =  0xde;

sfr    PIM0        =  0x92;
sfr    PIM1        =  0x91;

void main()
{
    PIM0 = 0x00;           // 设置 P1.0 为 ADC 口

```

```

PIMI = 0x01;
ADCCFG = 0x0f;           //设置ADC 时钟为系统时钟/2/16/16
ADC_CONTR = 0x80;         //使能ADC 模块

while (1)
{
    ADC_CONTR |= 0x40;      //启动AD 转换
    _nop_();
    _nop_();
    while (!(ADC_CONTR & 0x20)); //查询ADC 完成标志
    ADC_CONTR &= ~0x20;       //清完成标志
    P2 = ADC_RES;            //读取ADC 结果
}
}

```

16.4.2 ADC基本操作（中断方式）

汇编代码

;测试工作频率为 11.0592MHz

<i>ADC_CONTR</i>	<i>DATA</i>	<i>0BCH</i>
<i>ADC_RES</i>	<i>DATA</i>	<i>0BDH</i>
<i>ADC_RESL</i>	<i>DATA</i>	<i>0BEH</i>
<i>ADCCFG</i>	<i>DATA</i>	<i>0DEH</i>
<i>EADC</i>	<i>BIT</i>	<i>IE.5</i>
<i>PIM0</i>	<i>DATA</i>	<i>092H</i>
<i>PIMI</i>	<i>DATA</i>	<i>091H</i>
	<i>ORG</i>	<i>0000H</i>
	<i>LJMP</i>	<i>MAIN</i>
	<i>ORG</i>	<i>002BH</i>
	<i>LJMP</i>	<i>ADCISR</i>
<i>ADCISR:</i>	<i>ORG</i>	<i>0100H</i>
	<i>ANL</i>	<i>ADC_CONTR,#NOT 20H</i> ;清完成标志
	<i>MOV</i>	<i>P2,ADC_RES</i> ;读取ADC 结果
	<i>ORL</i>	<i>ADC_CONTR,#40H</i> ;继续AD 转换
	<i>RETI</i>	
<i>MAIN:</i>	<i>MOV</i>	<i>SP,#3FH</i>
	<i>MOV</i>	<i>PIM0,#00H</i> ;设置P1.0 为ADC 口
	<i>MOV</i>	<i>PIMI,#01H</i>
	<i>MOV</i>	<i>ADCCFG,#0FH</i> ;设置ADC 时钟为系统时钟/2/16/16
	<i>MOV</i>	<i>ADC_CONTR,#80H</i> ;使能ADC 模块
	<i>SETB</i>	<i>EADC</i> ;使能ADC 中断
	<i>SETB</i>	<i>EA</i>
	<i>ORL</i>	<i>ADC_CONTR,#40H</i> ;启动AD 转换
	<i>SJMP</i>	\$
<i>END</i>		

C 语言代码

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

//测试工作频率为 11.0592MHz

sfr ADC_CONTR = 0xbc;
sfr ADC_RES = 0xbd;
sfr ADC_RESL = 0xbe;
sfr ADCCFG = 0xde;

sbit EADC = IE^5;

sfr P1M0 = 0x92;
sfr P1M1 = 0x91;

void ADC_Isr() interrupt 5
{
    ADC_CONTR &= ~0x20;           //清中断标志
    P2 = ADC_RES;                //读取ADC 结果
    ADC_CONTR |= 0x40;           //继续AD 转换
}

void main()
{
    P1M0 = 0x00;                //设置P1.0 为ADC 口
    P1M1 = 0x01;                //设置ADC 时钟为系统时钟/2/16/16
    ADCCFG = 0x0f;               //使能ADC 模块
    ADC_CONTR = 0x80;             //使能ADC 中断
    EADC = 1;                   //启动AD 转换
    ADC_CONTR |= 0x40;           //启动AD 转换

    while (1);
}
```

16.4.3 格式化ADC转换结果

汇编代码

; 测试工作频率为 11.0592MHz

ADC_CONTR	DATA	0BCH
ADC_RES	DATA	0BDH
ADC_RESL	DATA	0BEH
ADCCFG	DATA	0DEH
P1M0	DATA	092H
P1M1	DATA	091H
ORG	0000H	
LJMP	MAIN	
ORG	0100H	
MAIN:	MOV	SP,#3FH
MOV	P1M0,#00H	; 设置P1.0 为ADC 口

MOV	PIMI,#01H		
MOV	ADCCFG,#0FH	; 设置ADC 时钟为系统时钟/2/16/16	
MOV	ADC_CONTR,#80H	; 使能ADC 模块	
ORL	ADC_CONTR,#40H	; 启动AD 转换	
NOP			
NOP			
MOV	A,ADC CONTR	; 查询ADC 完成标志	
JNB	ACC.5,\$-2		
ANL	ADC CONTR,#NOT 20H	; 清完成标志	
MOV	ADCCFG#00H	; 设置结果左对齐	
MOV	A,ADC RES	; A 存储ADC 的12 位结果的高8 位	
MOV	B,ADC RESL	; B[7:4]存储ADC 的12 位结果的低4 位,B[3:0]为0	
;	MOV	ADCCFG#20H	; 设置结果右对齐
;	MOV	A,ADC RES	; A[3:0]存储ADC 的12 位结果的高4 位,A[7:4]为0
;	MOV	B,ADC RESL	; B 存储ADC 的12 位结果的低8 位
SJMP	\$		
END			

C 语言代码

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

// 测试工作频率为 11.0592MHz

sfr ADC_CONTR = 0xbc;
sfr ADC_RES = 0xbd;
sfr ADC_RESL = 0xbe;
sfr ADCCFG = 0xde;

sfr PIM0 = 0x92;
sfr PIMI = 0x91;

void main()
{
    PIM0 = 0x00; // 设置P1.0 为ADC 口
    PIMI = 0x01;
    ADCCFG = 0x0f; // 设置ADC 时钟为系统时钟/2/16/16
    ADC_CONTR = 0x80; // 使能ADC 模块
    ADC_CONTR |= 0x40; // 启动AD 转换
    _nop_();
    _nop_();
    while (!(ADC_CONTR & 0x20)); // 查询ADC 完成标志
    ADC_CONTR &= ~0x20; // 清完成标志

    ADCCFG = 0x00; // 设置结果左对齐
    ACC = ADC_RES; // A 存储ADC 的12 位结果的高8 位
    B = ADC_RESL; // B[7:4]存储ADC 的12 位结果的低4 位,B[3:0]为0

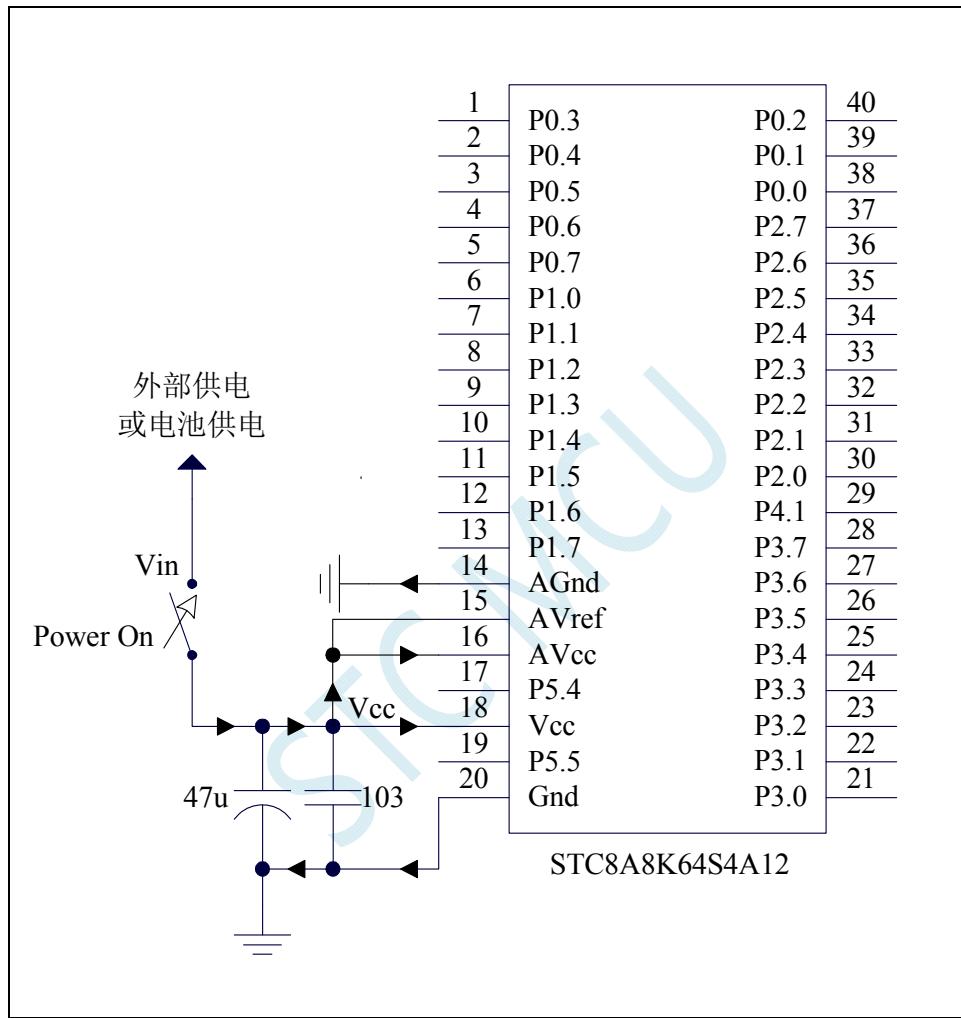
    // ADCCFG = 0x20; // 设置结果右对齐
    // ACC = ADC_RES; // A[3:0]存储ADC 的12 位结果的高4 位,A[7:4]为0
    // B = ADC_RESL; // B 存储ADC 的12 位结果的低8 位

    while (1);
}
```

16.4.4 利用ADC第 16 通道测量外部电压或电池电压

STC8 系列 ADC 的第 16 通道是用来测试内部 BandGap 参考电压的, 由于内部 BandGap 参考电压很稳定, 约为 1.35V, 且不会随芯片的工作电压的改变而变化, 所以可以通过测量内部 BandGap 参考电压, 然后通过 ADC 的值便可反推出外部电压或外部电池电压。

下图为参考线路图:



C 语言代码

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

// 测试工作频率为 11.0592MHz

#define FOSC      11059200UL
#define BRT       (65536 - FOSC / 115200 / 4)

sfr     AUXR      = 0x8e;
sfr     ADC_CONTR = 0xbc;
```

```

sfr    ADC_RES      = 0xbd;
sfr    ADC_RESL     = 0xbe;
sfr    ADCCFG       = 0xde;

sfr    PIM0         = 0x92;
sfr    PIMI         = 0x91;

int   *BGV;          //内部 Bandgap 电压值存放在 idata 中
                     //idata 的 EFH 地址存放高字节
                     //idata 的 F0H 地址存放低字节
                     //电压单位为毫伏(mV)

bit   busy;

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void ADCInit()
{
    ADCCFG = 0x2f;           //设置ADC 时钟为系统时钟/2/16/16
    ADC_CONTR = 0x8f;         //使能ADC 模块并选择第16 通道
}

int  ADCRead()
{
    int res;

    ADC_CONTR |= 0x40;        //启动AD 转换
    _nop_();
    _nop_();
    while (!(ADC_CONTR & 0x20)); //查询ADC 完成标志
    ADC_CONTR &= ~0x20;       //清完成标志
}

```

```
res = (ADC_RES << 8) / ADC_RESL; //读取ADC 结果

return res;
}

void main()
{
    int res;
    int vcc;
    int i;

    BGV = (intidata *)0xfe;
    ADCInit();           //ADC 初始化
    UartInit();          //串口初始化

    ES = 1;
    EA = 1;

    ADCRead();
    ADCRead();           //前两个数据丢弃
    res = 0;
    for (i=0; i<8; i++)
    {
        res += ADCRead();           //读取 8 次数据
    }
    res >>= 3;            //取平均值

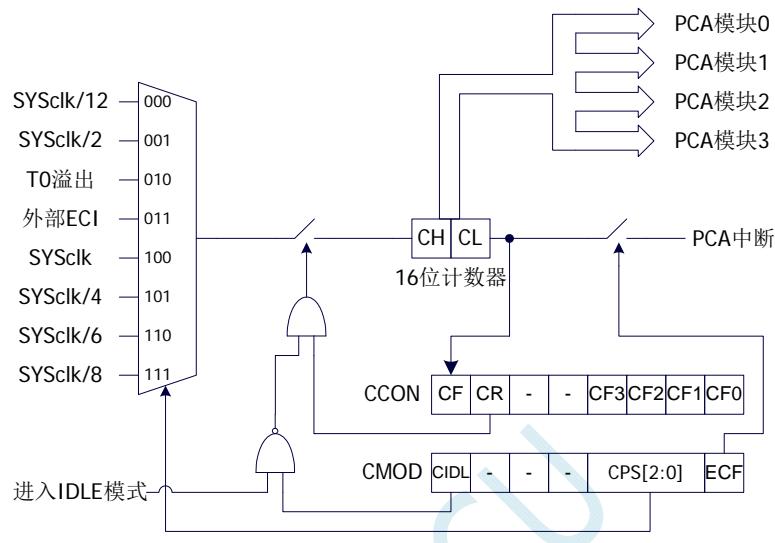
    vcc = (int)(4095L * *BGV / res); //计算 VREF 管脚电压,即电池电压
    //注意,此电压的单位为毫伏(mV)
    UartSend(vcc >> 8);          //输出电压值到串口
    UartSend(vcc);

    while (1);
}
```

17 PCA/CCP/PWM应用

STC8 系列单片机内部集成了 4 组可编程计数器阵列 (PCA/CCP/PWM) 模块, 可用于软件定时器、外部脉冲捕获、高速脉冲输出和 PWM 脉宽调制输出。

PCA 内部含有一个特殊的 16 位计数器, 4 组 PCA 模块均与之相连接。PCA 计数器的结构图如下:



PCA计数器结构图

17.1 PCA相关的寄存器

符号	描述	地址	位地址与符号								复位值				
			B7	B6	B5	B4	B3	B2	B1	B0					
CCON	PCA 控制寄存器	D8H	CF	CR	-	-	CCF3	CCF2	CCF1	CCF0	00xx,0000				
CMOD	PCA 模式寄存器	D9H	CIDL	-	-	-	CPS[2:0]			ECF	0xxx,0000				
CCAPM0	PCA 模块 0 模式控制寄存器	DAH	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0	x000,0000				
CCAPM1	PCA 模块 1 模式控制寄存器	DBH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1	x000,0000				
CCAPM2	PCA 模块 2 模式控制寄存器	DCH	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2	x000,0000				
CCAPM3	PCA 模块 3 模式控制寄存器	DDH	-	ECOM3	CCAPP3	CCAPN3	MAT3	TOG3	PWM3	ECCF3	x000,0000				
CL	PCA 计数器低字节	E9H									0000,0000				
CCAP0L	PCA 模块 0 低字节	EAH									0000,0000				
CCAP1L	PCA 模块 1 低字节	EBH									0000,0000				
CCAP2L	PCA 模块 2 低字节	ECH									0000,0000				
CCAP3L	PCA 模块 3 低字节	EDH									0000,0000				
PCA_PWM0	PCA0 的 PWM 模式寄存器	F2H	EBS0[1:0]		XCCAP0H[1:0]		XCCAP0L[1:0]		EPC0H	EPC0L	0000,0000				
PCA_PWM1	PCA1 的 PWM 模式寄存器	F3H	EBS1[1:0]		XCCAP1H[1:0]		XCCAP1L[1:0]		EPC1H	EPC1L	0000,0000				
PCA_PWM2	PCA2 的 PWM 模式寄存器	F4H	EBS2[1:0]		XCCAP2H[1:0]		XCCAP2L[1:0]		EPC2H	EPC2L	0000,0000				
PCA_PWM3	PCA3 的 PWM 模式寄存器	F5H	EBS3[1:0]		XCCAP3H[1:0]		XCCAP3L[1:0]		EPC3H	EPC3L	0000,0000				
CH	PCA 计数器高字节	F9H									0000,0000				
CCAP0H	PCA 模块 0 高字节	FAH									0000,0000				
CCAP1H	PCA 模块 1 高字节	FBH									0000,0000				

CCAP2H	PCA 模块 2 高字节	FCH								0000,0000
CCAP3H	PCA 模块 3 高字节	FDH								0000,0000

PCA 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CCON	D8H	CF	CR	-	-	CCF3	CCF2	CCF1	CCF0

CF: PCA 计数器溢出中断标志。当 PCA 的 16 位计数器计数发生溢出时，硬件自动将此位置 1，并向 CPU 提出中断请求。此标志位需要软件清零。

CR: PCA 计数器允许控制位。

0: 停止 PCA 计数

1: 启动 PCA 计数

CCFn (n=0,1,2,3): PCA 模块中断标志。当 PCA 模块发生匹配或者捕获时，硬件自动将此位置 1，并向 CPU 提出中断请求。此标志位需要软件清零。

PCA 模式寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMOD	D9H	CIDL	-	-	-	CPS[2:0]			ECF

CIDL: 空闲模式下是否停止 PCA 计数。

0: 空闲模式下 PCA 继续计数

1: 空闲模式下 PCA 停止计数

CPS[2:0]: PCA 计数脉冲源选择位

CPS[2:0]	PCA 的输入时钟源
000	系统时钟/12
001	系统时钟/2
010	定时器 0 的溢出脉冲
011	ECI 脚的外部输入时钟
100	系统时钟
101	系统时钟/4
110	系统时钟/6
111	系统时钟 8

ECF: PCA 计数器溢出中断允许位。

0: 禁止 PCA 计数器溢出中断

1: 使能 PCA 计数器溢出中断

PCA 计数器寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CL	E9H								
CH	F9H								

由 CL 和 CH 两个字节组合成一个 16 位计数器，CL 为低 8 位计数器，CH 为高 8 位计数器。每个 PCA 时钟 16 位计数器自动加 1。

PCA 模块模式控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0

CCAPM0	DAH	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0
CCAPM1	DBH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1
CCAPM2	DCH	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2
CCAPM3	DDH	-	ECOM3	CCAPP3	CCAPN3	MAT3	TOG3	PWM3	ECCF3

ECOMn: 允许 PCA 模块 n 的比较功能

CCAPPn: 允许 PCA 模块 n 进行上升沿捕获

CCAPNn: 允许 PCA 模块 n 进行下降沿捕获

MATn: 允许 PCA 模块 n 的匹配功能

TOGn: 允许 PCA 模块 n 的高速脉冲输出功能

PWMn: 允许 PCA 模块 n 的脉宽调制输出功能

ECCFn: 允许 PCA 模块 n 的匹配/捕获中断

PCA 模块模式捕获值/比较值寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CCAP0L	EAH								
CCAP1L	EBH								
CCAP2L	ECH								
CCAP3L	EDH								
CCAP0H	FAH								
CCAP1H	FBH								
CCAP2H	FCH								
CCAP3H	FDH								

当 PCA 模块捕获功能使能时, CCAPnL 和 CCAPnH 用于保存发生捕获时的 PCA 的计数值(CL 和 CH);

当 PCA 模块比较功能使能时, PCA 控制器会将当前 CL 和 CH 中的计数值与保存在 CCAPnL 和 CCAPnH 中的值进行比较, 并给出比较结果; 当 PCA 模块匹配功能使能时, PCA 控制器会将当前 CL 和 CH 中的计数值与保存在 CCAPnL 和 CCAPnH 中的值进行比较, 看是否匹配(相等), 并给出匹配结果。

PCA 模块 PWM 模式控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PCA_PWM0	F2H	EBS0[1:0]		XCCAP0H[1:0]		XCCAP0L[1:0]		EPC0H	EPC0L
PCA_PWM1	F3H	EBS1[1:0]		XCCAP1H[1:0]		XCCAP1L[1:0]		EPC1H	EPC1L
PCA_PWM2	F4H	EBS2[1:0]		XCCAP2H[1:0]		XCCAP2L[1:0]		EPC2H	EPC2L
PCA_PWM3	F5H	EBS3[1:0]		XCCAP3H[1:0]		XCCAP3L[1:0]		EPC3H	EPC3L

EBSn[1:0]: PCA 模块 n 的 PWM 位数控制

EBSn[1:0]	PWM 位数	重载值	比较值
00	8 位 PWM	{EPCnH, CCAPnH[7:0]}	{EPCnL, CCAPnL[7:0]}
01	7 位 PWM	{EPCnH, CCAPnH[6:0]}	{EPCnL, CCAPnL[6:0]}
10	6 位 PWM	{EPCnH, CCAPnH[5:0]}	{EPCnL, CCAPnL[5:0]}
11	10 位 PWM	{EPCnH, XCCAPnH[1:0], CCAPnH[7:0]}	{EPCnL, XCCAPnL[1:0], CCAPnL[7:0]}

XCCAPnH[1:0]: 10 位 PWM 的第 9 位和第 10 位的重载值

XCCAPnL[1:0]: 10 位 PWM 的第 9 位和第 10 位的比较值

EPCnH: PWM 模式下, 重载值的最高位(8 为 PWM 的第 9 位, 7 位 PWM 的第 8 位, 6 位 PWM 的第

7 位, 10 位 PWM 的第 11 位)

EPCnL: PWM 模式下, 比较值的最高位 (8 为 PWM 的第 9 位, 7 位 PWM 的第 8 位, 6 位 PWM 的第 7 位, 10 位 PWM 的第 11 位)

注意: 在更新 10 位 PWM 的重载值时, 必须先写高两位 XCCAPnH[1:0], 再写低 8 位 CCAPnH[7:0]。

STCMCU

17.2 PCA工作模式

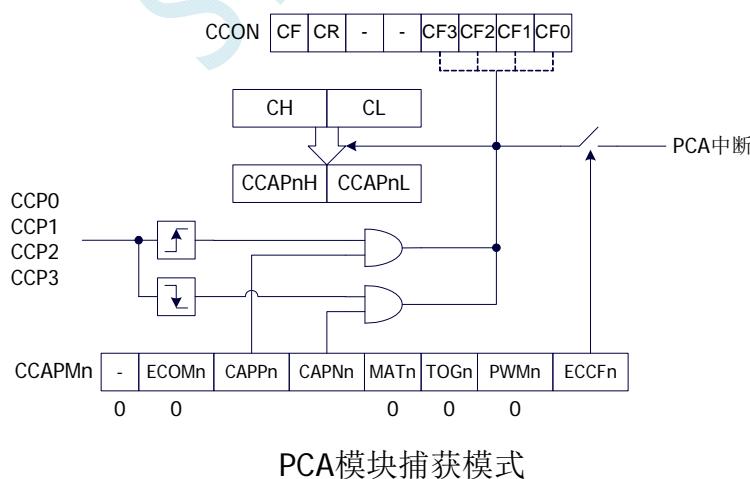
STC8 系列单片机共有 4 组 PCA 模块，每组模块都可独立设置工作模式。模式设置如下所示：

CCAPMn							模块功能	
-	ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn	
-	0	0	0	0	0	0	0	无操作
-	1	0	0	0	0	1	0	6/7/8/10 位 PWM 模式，无中断
-	1	1	0	0	0	1	1	6/7/8/10 位 PWM 模式，产生上升沿中断
-	1	0	1	0	0	1	1	6/7/8/10 位 PWM 模式，产生下降沿中断
-	1	1	1	0	0	1	1	6/7/8/10 位 PWM 模式，产生边沿中断
-	0	1	0	0	0	0	x	16 位上升沿捕获
-	0	0	1	0	0	0	x	16 位下降沿捕获
-	0	1	1	0	0	0	x	16 位边沿捕获
-	1	0	0	1	0	0	x	16 位软件定时器
-	1	0	0	1	1	0	x	16 为高速脉冲输出

17.2.1 捕获模式

要使一个 PCA 模块工作在捕获模式，寄存器 CCAPMn 中的 CAPNn 和 CAPPn 至少有一位必须置 1（也可两位都置 1）。PCA 模块工作于捕获模式时，对模块的外部 CCP0/CCP1/CCP2/CCP3 管脚的输入跳变进行采样。当采样到有效跳变时，PCA 控制器立即将 PCA 计数器 CH 和 CL 中的计数值装载到模块的捕获寄存器中 CCAPnL 和 CCAPnH，同时将 CCON 寄存器中相应的 CCFn 置 1。若 CCAPMn 中的 ECCFn 位被设置为 1，将产生中断。由于所有 PCA 模块的中断入口地址是共享的，所以在中断服务程序中需要判断是哪一个模块产生了中断，并注意中断标志位需要软件清零。

PCA 模块工作于捕获模式的结构图如下图所示：

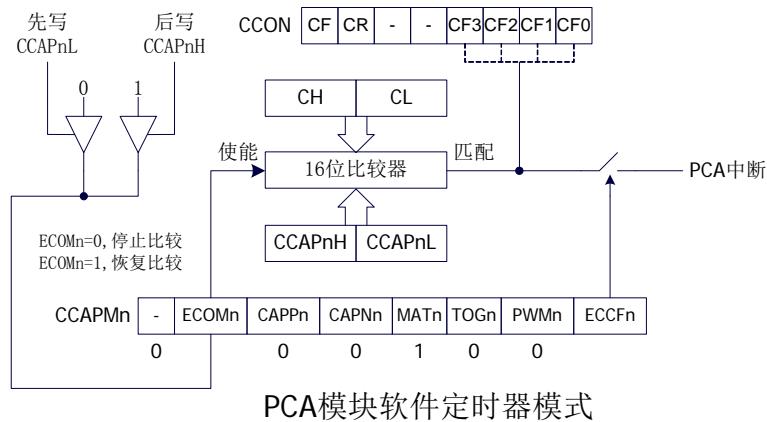


PCA模块捕获模式

17.2.2 软件定时器模式

通过置位 CCAPMn 寄存器的 ECOM 和 MAT 位，可使 PCA 模块用作软件定时器。PCA 计数器值 CL 和 CH 与模块捕获寄存器的值 CCAPnL 和 CCAPnH 相比较，当两者相等时，CCON 中的 CCFn 会被置 1，若 CCAPMn 中的 ECCFn 被设置为 1 时将产生中断。CCFn 标志位需要软件清零。

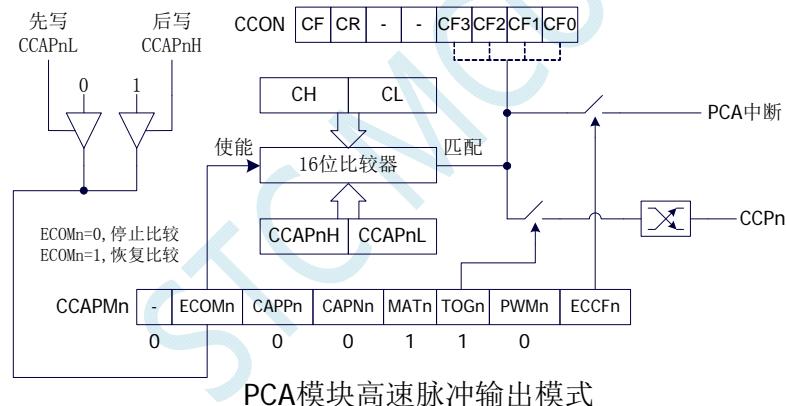
PCA 模块工作于软件定时器模式的结构图如下图所示：



17.2.3 高速脉冲输出模式

当 PCA 计数器的计数值与模块捕获寄存器的值相匹配时,PCA 模块的 CCPn 输出将发生翻转。要激活高速脉冲输出模式,CCAPMn 寄存器的 TOGn、MATn 和 ECOMn 位必须都置 1。

PCA 模块工作于高速脉冲输出模式的结构图如下图所示:



17.2.4 PWM脉宽调制模式

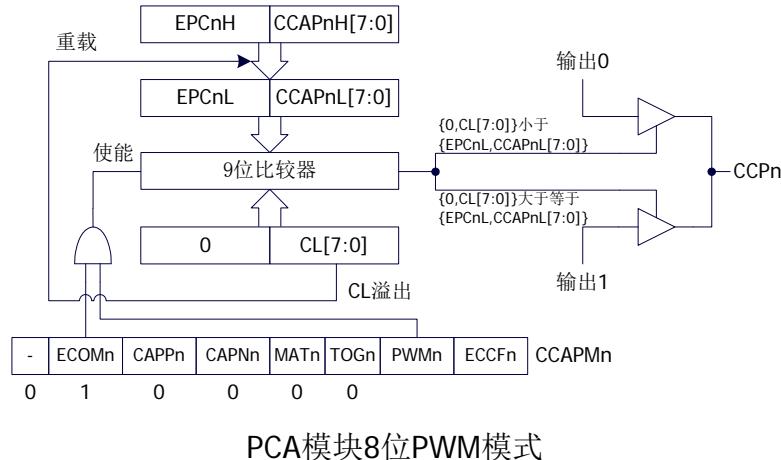
17.2.4.1 8 位PWM模式

脉宽调制是使用程序来控制波形的占空比、周期、相位波形的一种技术,在三相电机驱动、D/A 转换等场合有广泛的应用。STC8 系列单片机的 PCA 模块可以通过设定各自的 PCA_PWMn 寄存器使其工作于 8 位 PWM 或 7 位 PWM 或 6 位 PWM 或 10 位 PWM 模式。要使能 PCA 模块的 PWM 功能,模块寄存器 CCAPMn 的 PWMn 和 ECOMn 位必须置 1。

PCA_PWMn 寄存器中的 EBSn[1:0]设置为 00 时,PCA 模块 n 工作于 8 位 PWM 模式,此时将{0,CL[7:0]}与捕获寄存器{EPCnL,CCAPnL[7:0]}进行比较。当 PCA 模块工作于 8 位 PWM 模式时,由于所有模块共用一个 PCA 计数器,所有它们的输出频率相同。各个模块的输出占空比使用寄存器{EPCnL,CCAPnL[7:0]}进行设置。当{0,CL[7:0]}的值小于{EPCnL,CCAPnL[7:0]}时,输出为低电平;当

{0,CL[7:0]}的值等于或大于{EPCnL,CCAPnL[7:0]}时，输出为高电平。当CL[7:0]的值由FF变为00溢出时，{EPCnH,CCAPnH[7:0]}的内容重新装载到{EPCnL,CCAPnL[7:0]}中。这样就可实现无干扰地更新PWM。

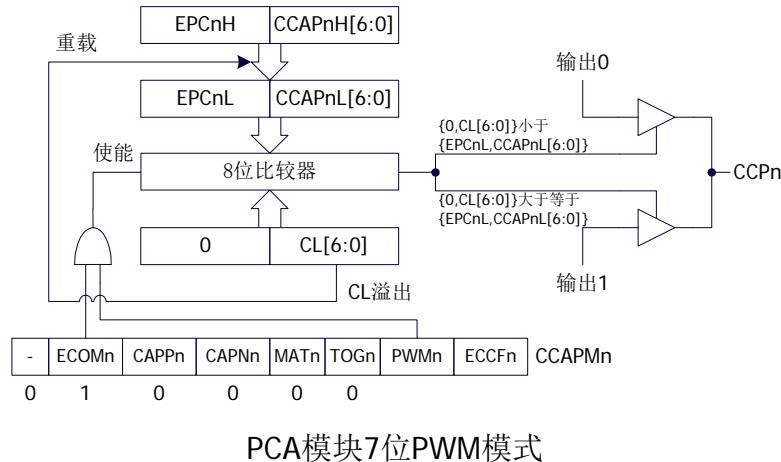
PCA 模块工作于 8 位 PWM 模式的结构图如下图所示：



17.2.4.2 7 位 PWM 模式

PCA_PWMn 寄存器中的 EBSn[1:0]设置为 01 时，PCA 模块 n 工作于 7 位 PWM 模式，此时将{0,CL[6:0]}与捕获寄存器{EPCnL,CCAPnL[6:0]}进行比较。当 PCA 模块工作于 7 位 PWM 模式时，由于所有模块共用一个 PCA 计数器，所有它们的输出频率相同。各个模块的输出占空比使用寄存器{EPCnL,CCAPnL[6:0]}进行设置。当{0,CL[6:0]}的值小于{EPCnL,CCAPnL[6:0]}时，输出为低电平；当{0,CL[6:0]}的值等于或大于{EPCnL,CCAPnL[6:0]}时，输出为高电平。当 CL[6:0]的值由 7F 变为 00 溢出时，{EPCnH,CCAPnH[6:0]}的内容重新装载到{EPCnL,CCAPnL[6:0]}中。这样就可实现无干扰地更新 PWM。

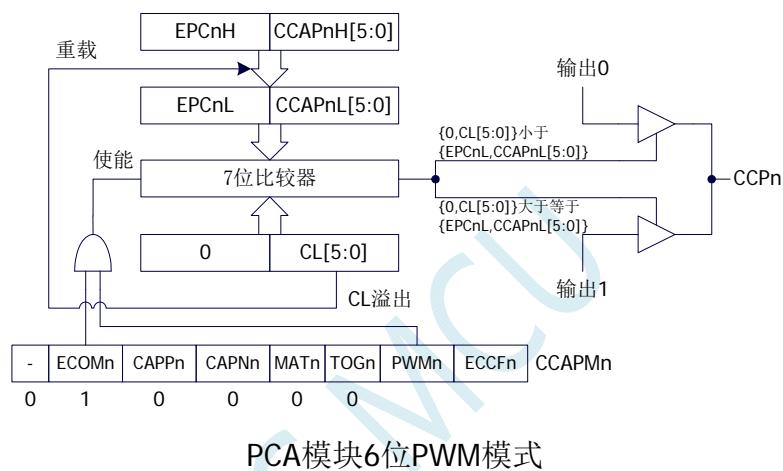
PCA 模块工作于 7 位 PWM 模式的结构图如下图所示：



17.2.4.3 6位PWM模式

PCA_PWMn 寄存器中的 EBSn[1:0]设置为 10 时, PCA 模块 n 工作于 6 位 PWM 模式, 此时将 {0,CL[5:0]}与捕获寄存器{EPCnL,CCAPnL[5:0]}进行比较。当 PCA 模块工作于 6 位 PWM 模式时, 由于所有模块共用一个 PCA 计数器, 所有它们的输出频率相同。各个模块的输出占空比使用寄存器{EPCnL,CCAPnL[5:0]}进行设置。当{0,CL[5:0]}的值小于{EPCnL,CCAPnL[5:0]}时, 输出为低电平; 当{0,CL[5:0]}的值等于或大于{EPCnL,CCAPnL[5:0]}时, 输出为高电平。当 CL[5:0]的值由 3F 变为 00 溢出时, {EPCnH,CCAPnH[5:0]}的内容重新装载到{EPCnL,CCAPnL[5:0]}中。这样就可实现无干扰地更新 PWM。

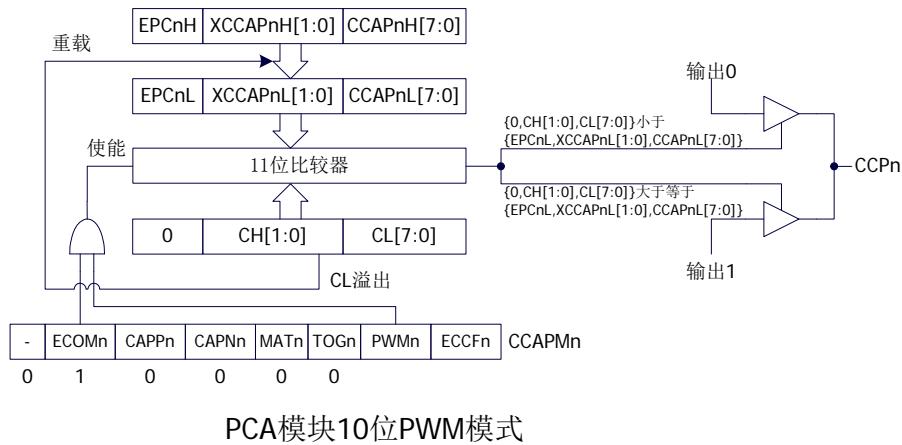
PCA 模块工作于 6 位 PWM 模式的结构图如下图所示:



17.2.4.4 10位PWM模式

PCA_PWMn 寄存器中的 EBSn[1:0]设置为 11 时, PCA 模块 n 工作于 10 位 PWM 模式, 此时将 {CH[1:0],CL[7:0]}与捕获寄存器{EPCnL,XCCAPnL[1:0],CCAPnL[7:0]}进行比较。当 PCA 模块工作于 10 位 PWM 模式时, 由于所有模块共用一个 PCA 计数器, 所有它们的输出频率相同。各个模块的输出占空比使用寄存器{EPCnL,XCCAPnL[1:0],CCAPnL[7:0]}进行设置。当 {CH[1:0],CL[7:0]} 的值小于 {EPCnL,XCCAPnL[1:0],CCAPnL[7:0]} 时, 输出为低电平; 当 {CH[1:0],CL[7:0]} 的值等于或大于 {EPCnL,XCCAPnL[1:0],CCAPnL[7:0]} 时, 输出为高电平。当 {CH[1:0],CL[7:0]} 的值由 3FF 变为 00 溢出时, {EPCnH,XCCAPnH[1:0],CCAPnH[7:0]} 的内容重新装载到 {EPCnL,XCCAPnL[1:0],CCAPnL[7:0]} 中。这样就可实现无干扰地更新 PWM。

PCA 模块工作于 10 位 PWM 模式的结构图如下图所示:



注意事项:

1. 4 通道 PCA 工作于 PWM 时, 切换到 P1.4~P1.7 或 P2.3~P2.6 或 P3.3~P3.0.,, 情况同上类似, IO 设置为准双向口或推挽输出, PWM 输出正常, 均为推挽输出, 并且再直接操作 IO 将不影响 PWM 波形。但是如果将 IO 设置为开漏输出 (允许内部上拉电阻或外接上拉电阻), 则如果对应的 IO 输出高电平, PWM 无输出 (IO 为高阻), 而对应的 IO 输出低电平, PWM 有推挽输出。
2. 4 通道 PCA 工作于高速输出时, 受到同一个端口其余 IO 翻转的影响, 比如 PCA3 从 P2.6 高速输出脉冲, P2 口除了 P2.6 外任何一个 P2 口的改变都会影响到 P2.6 波形, 哪怕是将 IO 设置为高阻。

17.3 范例程序

17.3.1 PCA输出PWM（6/7/8/10位）

汇编代码

;测试工作频率为11.0592MHz

```

CCON      DATA    0D8H
CF        BIT     CCON.7
CR        BIT     CCON.6
CCF3     BIT     CCON.3
CCF2     BIT     CCON.2
CCF1     BIT     CCON.1
CCF0     BIT     CCON.0
CMOD     DATA    0D9H
CL        DATA    0E9H
CH        DATA    0F9H
CCAPM0   DATA    0DAH
CCAP0L   DATA    0EAH
CCAP0H   DATA    0FAH
PCA_PWM0  DATA    0F2H
CCAPM1   DATA    0DBH
CCAP1L   DATA    0EBH
CCAPIH   DATA    0FBH
PCA_PWM1  DATA    0F3H
CCAPM2   DATA    0DCH
CCAP2L   DATA    0ECH
CCAP2H   DATA    0FCH
PCA_PWM2  DATA    0F4H
CCAPM3   DATA    0DDH
CCAP3L   DATA    0EDH
CCAP3H   DATA    0FDH
PCA_PWM3  DATA    0F5H

        ORG    0000H
        LJMP  MAIN

        ORG    0100H
MAIN:
        MOV    SP,#3FH

        MOV    CCON,#00H
        MOV    CMOD,#08H          ;PCA 时钟为系统时钟
        MOV    CL,#00H
        MOV    CH,#0H
        MOV    CCAPM0,#42H         ;PCA 模块0 为 PWM 工作模式
        MOV    PCA_PWM0,#80H        ;PCA 模块0 输出6位 PWM
        MOV    CCAP0L,#20H          ;PWM 占空比为50%[(40H-20H)/40H]
        MOV    CCAP0H,#20H
        MOV    CCAPM1,#42H         ;PCA 模块1 为 PWM 工作模式
        MOV    PCA_PWM1,#40H        ;PCA 模块1 输出7位 PWM
        MOV    CCAP1L,#20H          ;PWM 占空比为75%[(80H-20H)/80H]
        MOV    CCAPIH,#20H
        MOV    CCAPM2,#42H         ;PCA 模块2 为 PWM 工作模式
        MOV    PCA_PWM2,#00H        ;PCA 模块2 输出8位 PWM
        MOV    CCAP2L,#20H          ;PWM 占空比为87.5%[(100H-20H)/100H]
        MOV    CCAP2H,#20H
        MOV    CCAPM3,#42H         ;PCA 模块3 为 PWM 工作模式

```

MOV	PCA_PWM3,#0C0H	;PCA 模块3 输出10位PWM
MOV	CCAP3L,#20H	;PWM 占空比为96.875%[(400H-20H)/400H]
MOV	CCAP3H,#20H	
SETB	CR	;启动PCA 计时器
JMP	\$	
END		

C 语言代码

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

//测试工作频率为11.0592MHz

sfr CCON = 0xd8;
sbit CF = CCON^7;
sbit CR = CCON^6;
sbit CCF3 = CCON^3;
sbit CCF2 = CCON^2;
sbit CCF1 = CCON^1;
sbit CCF0 = CCON^0;
sfr CMOD = 0xd9;
sfr CL = 0xe9;
sfr CH = 0xf9;
sfr CCAPM0 = 0xda;
sfr CCAP0L = 0xea;
sfr CCAP0H = 0xfa;
sfr PCA_PWM0 = 0xf2;
sfr CCAPM1 = 0xdb;
sfr CCAP1L = 0xeb;
sfr CCAP1H = 0xfb;
sfr PCA_PWM1 = 0xf3;
sfr CCAPM2 = 0xdc;
sfr CCAP2L = 0xec;
sfr CCAP2H = 0xfc;
sfr PCA_PWM2 = 0xf4;
sfr CCAPM3 = 0xdd;
sfr CCAP3L = 0xed;
sfr CCAP3H = 0xfd;
sfr PCA_PWM3 = 0xf5;

void main()
{
    CCON = 0x00;
    CMOD = 0x08; //PCA 时钟为系统时钟
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x42; //PCA 模块0 为 PWM 工作模式
    PCA_PWM0 = 0x80; //PCA 模块0 输出6位PWM
    CCAP0L = 0x20; //PWM 占空比为50%[(40H-20H)/40H]
    CCAP0H = 0x20;
    CCAPM1 = 0x42; //PCA 模块1 为 PWM 工作模式
    PCA_PWM1 = 0x40; //PCA 模块1 输出7位PWM
    CCAP1L = 0x20; //PWM 占空比为75%[(80H-20H)/80H]
    CCAP1H = 0x20;
    CCAPM2 = 0x42; //PCA 模块2 为 PWM 工作模式
    PCA_PWM2 = 0x00; //PCA 模块2 输出8位PWM
}
```

```

CCAP2L = 0x20;           //PWM 占空比为87.5%[(100H-20H)/100H]
CCAP2H = 0x20;
CCAPM3 = 0x42;           //PCA 模块3 为 PWM 工作模式
PCA_PWM3 = 0xc0;          //PCA 模块3 输出10位 PWM
CCAP3L = 0x20;           //PWM 占空比为96.875%[(400H-20H)/400H]
CCAP3H = 0x20;
CR = 1;                  //启动PCA 计时器

while (1);
}

```

17.3.2 PCA捕获测量脉冲宽度

汇编代码

;测试工作频率为11.0592MHz

<i>CCON</i>	<i>DATA</i>	<i>0D8H</i>	
<i>CF</i>	<i>BIT</i>	<i>CCON.7</i>	
<i>CR</i>	<i>BIT</i>	<i>CCON.6</i>	
<i>CCF3</i>	<i>BIT</i>	<i>CCON.3</i>	
<i>CCF2</i>	<i>BIT</i>	<i>CCON.2</i>	
<i>CCF1</i>	<i>BIT</i>	<i>CCON.1</i>	
<i>CCF0</i>	<i>BIT</i>	<i>CCON.0</i>	
<i>CMOD</i>	<i>DATA</i>	<i>0D9H</i>	
<i>CL</i>	<i>DATA</i>	<i>0E9H</i>	
<i>CH</i>	<i>DATA</i>	<i>0F9H</i>	
<i>CCAPM0</i>	<i>DATA</i>	<i>0DAH</i>	
<i>CCAP0L</i>	<i>DATA</i>	<i>0EAH</i>	
<i>CCAP0H</i>	<i>DATA</i>	<i>0FAH</i>	
<i>PCA_PWM0</i>	<i>DATA</i>	<i>0F2H</i>	
<i>CCAPM1</i>	<i>DATA</i>	<i>0DBH</i>	
<i>CCAPIL</i>	<i>DATA</i>	<i>0EBH</i>	
<i>CCAPIH</i>	<i>DATA</i>	<i>0FBH</i>	
<i>PCA_PWM1</i>	<i>DATA</i>	<i>0F3H</i>	
<i>CCAPM2</i>	<i>DATA</i>	<i>0DCH</i>	
<i>CCAP2L</i>	<i>DATA</i>	<i>0ECH</i>	
<i>CCAP2H</i>	<i>DATA</i>	<i>0FCH</i>	
<i>PCA_PWM2</i>	<i>DATA</i>	<i>0F4H</i>	
<i>CCAPM3</i>	<i>DATA</i>	<i>0DDH</i>	
<i>CCAP3L</i>	<i>DATA</i>	<i>0EDH</i>	
<i>CCAP3H</i>	<i>DATA</i>	<i>0FDH</i>	
<i>PCA_PWM3</i>	<i>DATA</i>	<i>0F5H</i>	
<i>CNT</i>	<i>DATA</i>	<i>20H</i>	
<i>COUNT0</i>	<i>DATA</i>	<i>21H</i>	;3 bytes
<i>COUNT1</i>	<i>DATA</i>	<i>24H</i>	;3 bytes
<i>LENGTH</i>	<i>DATA</i>	<i>27H</i>	;3 bytes, (COUNT1-COUNT0)
	<i>ORG</i>	<i>0000H</i>	
	<i>LJMP</i>	<i>MAIN</i>	
	<i>ORG</i>	<i>003BH</i>	
	<i>LJMP</i>	<i>PCAISR</i>	
<i>PCAISR:</i>	<i>ORG</i>	<i>0100H</i>	
	<i>PUSH</i>	<i>ACC</i>	
	<i>PUSH</i>	<i>PSW</i>	
	<i>JNB</i>	<i>CF,CHECKCCF0</i>	

CLR	CF	;清中断标志
INC	CNT	; PCA 计时溢出次数+1

CHECKCCF0:

JNB	CCF0,ISREXIT	
CLR	CCF0	
MOV	COUNT0,COUNT1	;备份上一次的捕获值
MOV	COUNT0+1,COUNT1+1	
MOV	COUNT0+2,COUNT1+2	
MOV	COUNT1,CNT	;保存本次的捕获值
MOV	COUNT1+1,CCAP0H	
MOV	COUNT1+2,CCAP0L	
CLR	C	;计算两次的捕获差值
MOV	A,COUNT1+2	
SUBB	A,COUNT0+2	
MOV	LENGTH+2,A	
MOV	A,COUNT1+1	
SUBB	A,COUNT0+1	
MOV	LENGTH+1,A	
MOV	A,COUNT1	
SUBB	A,COUNT0	
MOV	LENGTH,A	;LENGTH 保存的即为捕获的脉冲宽度

ISREXIT:

POP	PSW	
POP	ACC	
RETI		

MAIN:

MOV	SP,#3FH		
CLR	A		
MOV	CNT,A		
MOV	COUNT0,A	;用户变量初始化	
MOV	COUNT0+1,A		
MOV	COUNT0+2,A		
MOV	COUNT1,A		
MOV	COUNT1+1,A		
MOV	COUNT1+2,A		
MOV	LENGTH,A		
MOV	LENGTH+1,A		
MOV	LENGTH+2,A		
MOV	CCON,#00H		
MOV	CMOD,#09H	;PCA 时钟为系统时钟,使能PCA 计时中断	
MOV	CL,#00H		
MOV	CH,#0H		
MOV	CCAPM0,#11H	;PCA 模块0 为16 位捕获模式 (下降沿捕获)	
MOV	CCAPM0,#21H	;PCA 模块0 为16 位捕获模式 (上升沿捕获)	
;	MOV	CCAPM0,#31H	;PCA 模块0 为16 位捕获模式 (边沿捕获)
;	MOV	CCAP0L,#00H	
MOV	CCAP0H,#00H		
SETB	CR	;启动PCA 计时器	
SETB	EA		
JMP	\$		

END**C 语言代码**

```

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

//测试工作频率为 11.0592MHz

sfr    CCON      = 0xd8;
sbit   CF        = CCON^7;
sbit   CR        = CCON^6;
sbit   CCF3      = CCON^3;
sbit   CCF2      = CCON^2;
sbit   CCF1      = CCON^1;
sbit   CCF0      = CCON^0;
sfr    CMOD      = 0xd9;
sfr    CL        = 0xe9;
sfr    CH        = 0xf9;
sfr    CCAPM0    = 0xda;
sfr    CCAP0L    = 0xea;
sfr    CCAP0H    = 0xfa;
sfr    PCA_PWM0  = 0xf2;
sfr    CCAPM1    = 0xdb;
sfr    CCAP1L    = 0xeb;
sfr    CCAPIH    = 0xfb;
sfr    PCA_PWM1  = 0xf3;
sfr    CCAPM2    = 0xdc;
sfr    CCAP2L    = 0xec;
sfr    CCAP2H    = 0xfc;
sfr    PCA_PWM2  = 0xf4;
sfr    CCAPM3    = 0xdd;
sfr    CCAP3L    = 0xed;
sfr    CCAP3H    = 0xfd;
sfr    PCA_PWM3  = 0xf5;

unsigned char  cnt;           //存储PCA 计时溢出次数
unsigned long  count0;        //记录上一次的捕获值
unsigned long  count1;        //记录本次的捕获值
unsigned long  length;         //存储信号的时间长度

void PCA_Isr() interrupt 7
{
    if (CF)
    {
        CF = 0;
        cnt++;           //PCA 计时溢出次数+1
    }
    if (CCF0)
    {
        CCF0 = 0;
        count0 = count1;          //备份上一次的捕获值
        ((unsigned char *)&count1)[3] = CCAP0L;
        ((unsigned char *)&count1)[2] = CCAP0H;
        ((unsigned char *)&count1)[1] = cnt;
        ((unsigned char *)&count1)[0] = 0;
        length = count1 - count0; //length 保存的即为捕获的脉冲宽度
    }
}

void main()
{
    cnt = 0;           //用户变量初始化
}

```

```

count0 = 0;
count1 = 0;
length = 0;
CCON = 0x00;
CMOD = 0x09;           //PCA 时钟为系统时钟,使能PCA 计时中断
CL = 0x00;
CH = 0x00;
CCAPM0 = 0x11;         //PCA 模块0 为16 位捕获模式 (下降沿捕获)
CCAPM0 = 0x21;         //PCA 模块0 为16 位捕获模式 (下降沿捕获)
CCAPM0 = 0x31;         //PCA 模块0 为16 位捕获模式 (下降沿捕获)
CCAP0L = 0x00;
CCAP0H = 0x00;
CR = 1;                //启动PCA 计时器
EA = 1;

while (1);
}

```

17.3.3 PCA实现 16 位软件定时

汇编代码

;测试工作频率为 11.0592MHz

<i>CCON</i>	<i>DATA</i>	<i>0D8H</i>	
<i>CF</i>	<i>BIT</i>	<i>CCON.7</i>	
<i>CR</i>	<i>BIT</i>	<i>CCON.6</i>	
<i>CCF3</i>	<i>BIT</i>	<i>CCON.3</i>	
<i>CCF2</i>	<i>BIT</i>	<i>CCON.2</i>	
<i>CCFI</i>	<i>BIT</i>	<i>CCON.1</i>	
<i>CCF0</i>	<i>BIT</i>	<i>CCON.0</i>	
<i>CMOD</i>	<i>DATA</i>	<i>0D9H</i>	
<i>CL</i>	<i>DATA</i>	<i>0E9H</i>	
<i>CH</i>	<i>DATA</i>	<i>0F9H</i>	
<i>CCAPM0</i>	<i>DATA</i>	<i>0DAH</i>	
<i>CCAP0L</i>	<i>DATA</i>	<i>0EAH</i>	
<i>CCAP0H</i>	<i>DATA</i>	<i>0FAH</i>	
<i>PCA_PWM0</i>	<i>DATA</i>	<i>0F2H</i>	
<i>CCAPM1</i>	<i>DATA</i>	<i>0DBH</i>	
<i>CCAP1L</i>	<i>DATA</i>	<i>0EBH</i>	
<i>CCAPIH</i>	<i>DATA</i>	<i>0FBH</i>	
<i>PCA_PWM1</i>	<i>DATA</i>	<i>0F3H</i>	
<i>CCAPM2</i>	<i>DATA</i>	<i>0DCH</i>	
<i>CCAP2L</i>	<i>DATA</i>	<i>0ECH</i>	
<i>CCAP2H</i>	<i>DATA</i>	<i>0FCH</i>	
<i>PCA_PWM2</i>	<i>DATA</i>	<i>0F4H</i>	
<i>CCAPM3</i>	<i>DATA</i>	<i>0DDH</i>	
<i>CCAP3L</i>	<i>DATA</i>	<i>0EDH</i>	
<i>CCAP3H</i>	<i>DATA</i>	<i>0FDH</i>	
<i>PCA_PWM3</i>	<i>DATA</i>	<i>0F5H</i>	
<i>T50HZ</i>	<i>EQU</i>	<i>2400H</i>	<i>;11059200/12/2/50</i>
	<i>ORG</i>	<i>0000H</i>	
	<i>LJMP</i>	<i>MAIN</i>	
	<i>ORG</i>	<i>003BH</i>	
	<i>LJMP</i>	<i>PCAISR</i>	
	<i>ORG</i>	<i>0100H</i>	

PCAISR:

<i>PUSH</i>	<i>ACC</i>
<i>PUSH</i>	<i>PSW</i>
<i>CLR</i>	<i>CCF0</i>
<i>MOV</i>	<i>A,CCAP0L</i>
<i>ADD</i>	<i>A,#LOW T50HZ</i>
<i>MOV</i>	<i>CCAP0L,A</i>
<i>MOV</i>	<i>A,CCAP0H</i>
<i>ADDC</i>	<i>A,#HIGH T50HZ</i>
<i>MOV</i>	<i>CCAP0H,A</i>
<i>CPL</i>	<i>P1.0</i> ; 测试端口, 闪烁频率为 50Hz
<i>POP</i>	<i>PSW</i>
<i>POP</i>	<i>ACC</i>
<i>RETI</i>	

MAIN:

<i>MOV</i>	<i>SP,#3FH</i>
<i>MOV</i>	<i>CCON,#00H</i>
<i>MOV</i>	<i>CMOD,#00H</i> ; PCA 时钟为系统时钟/12
<i>MOV</i>	<i>CL,#00H</i>
<i>MOV</i>	<i>CH,#0H</i>
<i>MOV</i>	<i>CCAPM0,#49H</i> ; PCA 模块 0 为 16 位定时器模式
<i>MOV</i>	<i>CCAP0L,#LOW T50HZ</i>
<i>MOV</i>	<i>CCAP0H,#HIGH T50HZ</i>
<i>SETB</i>	<i>CR</i> ; 启动 PCA 计时器
<i>SETB</i>	<i>EA</i>
<i>JMP</i>	\$

END**C 语言代码**

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

// 测试工作频率为 11.0592MHz

#define T50HZ (11059200L / 12 / 2 / 50)

sfr CCON = 0xd8;
sbit CF = CCON^7;
sbit CR = CCON^6;
sbit CCF3 = CCON^3;
sbit CCF2 = CCON^2;
sbit CCF1 = CCON^1;
sbit CCF0 = CCON^0;
sfr CMOD = 0xd9;
sfr CL = 0xe9;
sfr CH = 0xf9;
sfr CCAPM0 = 0xda;
sfr CCAP0L = 0xea;
sfr CCAP0H = 0xfa;
sfr PCA_PWM0 = 0xf2;
sfr CCAPM1 = 0xdb;
sfr CCAPIL = 0xeb;
sfr CCAPIH = 0xfb;
sfr PCA_PWM1 = 0xf3;
```

```

sfr CCAPM2      = 0xdc;
sfr CCAP2L      = 0xec;
sfr CCAP2H      = 0xfc;
sfr PCA_PWM2    = 0xf4;
sfr CCAPM3      = 0xdd;
sfr CCAP3L      = 0xed;
sfr CCAP3H      = 0xfd;
sfr PCA_PWM3    = 0xf5;

sbit P10         = P1^0;

unsigned int value;

void PCA_Isr() interrupt 7
{
    CCF0 = 0;
    CCAP0L = value;
    CCAP0H = value >> 8;
    value += T50HZ;

    P10 = !P10;           //测试端口
}

void main()
{
    CCON = 0x00;
    CMOD = 0x00;          //PCA 时钟为系统时钟/12
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x49;        //PCA 模块0 为16 位定时器模式
    value = T50HZ;
    CCAP0L = value;
    CCAP0H = value >> 8;
    value += T50HZ;
    CR = 1;               //启动PCA 计时器
    EA = 1;

    while (1);
}

```

17.3.4 PCA输出高速脉冲

汇编代码

;测试工作频率为11.0592MHz

CCON	DATA	0D8H
CF	BIT	CCON.7
CR	BIT	CCON.6
CCF3	BIT	CCON.3
CCF2	BIT	CCON.2
CCF1	BIT	CCON.1
CCF0	BIT	CCON.0
CMOD	DATA	0D9H
CL	DATA	0E9H
CH	DATA	0F9H
CCAPM0	DATA	0DAH
CCAP0L	DATA	0EAH
CCAP0H	DATA	0FAH

<i>PCA_PWM0</i>	<i>DATA</i>	<i>0F2H</i>	
<i>CCAPM1</i>	<i>DATA</i>	<i>0DBH</i>	
<i>CCAPIL</i>	<i>DATA</i>	<i>0EBH</i>	
<i>CCAPIH</i>	<i>DATA</i>	<i>0FBH</i>	
<i>PCA_PWM1</i>	<i>DATA</i>	<i>0F3H</i>	
<i>CCAPM2</i>	<i>DATA</i>	<i>0DCH</i>	
<i>CCAP2L</i>	<i>DATA</i>	<i>0ECH</i>	
<i>CCAP2H</i>	<i>DATA</i>	<i>0FCH</i>	
<i>PCA_PWM2</i>	<i>DATA</i>	<i>0F4H</i>	
<i>CCAPM3</i>	<i>DATA</i>	<i>0DDH</i>	
<i>CCAP3L</i>	<i>DATA</i>	<i>0EDH</i>	
<i>CCAP3H</i>	<i>DATA</i>	<i>0FDH</i>	
<i>PCA_PWM3</i>	<i>DATA</i>	<i>0F5H</i>	
<i>T38K4HZ</i>	<i>EQU</i>	<i>90H</i>	<i>;11059200/2/38400</i>
	<i>ORG</i>	<i>0000H</i>	
	<i>LJMP</i>	<i>MAIN</i>	
	<i>ORG</i>	<i>003BH</i>	
	<i>LJMP</i>	<i>PCAISR</i>	
	<i>ORG</i>	<i>0100H</i>	
<i>PCAISR:</i>			
	<i>PUSH</i>	<i>ACC</i>	
	<i>PUSH</i>	<i>PSW</i>	
	<i>CLR</i>	<i>CCF0</i>	
	<i>MOV</i>	<i>A,CCAP0L</i>	
	<i>ADD</i>	<i>A,#LOW T38K4HZ</i>	
	<i>MOV</i>	<i>CCAP0L,A</i>	
	<i>MOV</i>	<i>A,CCAP0H</i>	
	<i>ADDC</i>	<i>A,#HIGH T38K4HZ</i>	
	<i>MOV</i>	<i>CCAP0H,A</i>	
	<i>POP</i>	<i>PSW</i>	
	<i>POP</i>	<i>ACC</i>	
	<i>RETI</i>		
<i>MAIN:</i>			
	<i>MOV</i>	<i>SP,#3FH</i>	
	<i>MOV</i>	<i>CCON,#00H</i>	
	<i>MOV</i>	<i>CMOD,#08H</i>	<i>;PCA 时钟为系统时钟</i>
	<i>MOV</i>	<i>CL,#00H</i>	
	<i>MOV</i>	<i>CH,#0H</i>	
	<i>MOV</i>	<i>CCAPM0,#4DH</i>	<i>;PCA 模块 0 为 16 位定时器模式并使能脉冲输出</i>
	<i>MOV</i>	<i>CCAP0L,#LOW T38K4HZ</i>	
	<i>MOV</i>	<i>CCAP0H,#HIGH T38K4HZ</i>	
	<i>SETB</i>	<i>CR</i>	<i>;启动 PCA 计时器</i>
	<i>SETB</i>	<i>EA</i>	
	<i>JMP</i>	\$	
	<i>END</i>		

C 语言代码

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

//测试工作频率为11.0592MHz

```

#define T38K4HZ      (11059200L / 2 / 38400)

sfr CCON      = 0xd8;
sbit CF        = CCON^7;
sbit CR        = CCON^6;
sbit CCF3      = CCON^3;
sbit CCF2      = CCON^2;
sbit CCF1      = CCON^1;
sbit CCF0      = CCON^0;
sfr CMOD      = 0xd9;
sfr CL         = 0xe9;
sfr CH         = 0xf9;
sfr CCAPM0     = 0xda;
sfr CCAP0L     = 0xea;
sfr CCAP0H     = 0xfa;
sfr PCA_PWM0   = 0xf2;
sfr CCAPM1     = 0xdb;
sfr CCAP1L     = 0xeb;
sfr CCAP1H     = 0xfb;
sfr PCA_PWM1   = 0xf3;
sfr CCAPM2     = 0xdc;
sfr CCAP2L     = 0xec;
sfr CCAP2H     = 0xfc;
sfr PCA_PWM2   = 0xf4;
sfr CCAPM3     = 0xdd;
sfr CCAP3L     = 0xed;
sfr CCAP3H     = 0xfd;
sfr PCA_PWM3   = 0xf5;

unsigned int value;

void PCA_Isr() interrupt 7
{
    CCF0 = 0;
    CCAP0L = value;
    CCAP0H = value >> 8;
    value += T38K4HZ;
}

void main()
{
    CCON = 0x00;                                //PCA 时钟为系统时钟
    CMOD = 0x08;
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x4d;                             //PCA 模块0 为16位定时器模式并使能脉冲输出
    value = T38K4HZ;
    CCAP0L = value;
    CCAP0H = value >> 8;
    value += T38K4HZ;
    CR = 1;                                     //启动PCA 计时器
    EA = 1;

    while (1);
}

```

17.3.5 PCA扩展外部中断

汇编代码

; 测试工作频率为 11.0592MHz

```

CCON      DATA      0D8H
CF        BIT       CCON.7
CR        BIT       CCON.6
CCF3     BIT       CCON.3
CCF2     BIT       CCON.2
CCF1     BIT       CCON.1
CCF0     BIT       CCON.0
CMOD     DATA      0D9H
CL        DATA      0E9H
CH        DATA      0F9H
CCAPM0   DATA      0DAH
CCAP0L   DATA      0EAH
CCAP0H   DATA      0FAH
PCA_PWM0  DATA      0F2H
CCAPM1   DATA      0DBH
CCAP1L   DATA      0EBH
CCAPIH   DATA      0FBH
PCA_PWM1  DATA      0F3H
CCAPM2   DATA      0DCH
CCAP2L   DATA      0ECH
CCAP2H   DATA      0FCH
PCA_PWM2  DATA      0F4H
CCAPM3   DATA      0DDH
CCAP3L   DATA      0EDH
CCAP3H   DATA      0FDH
PCA_PWM3  DATA      0F5H

ORG      0000H
LJMP    MAIN
ORG      003BH
LJMP    PCAISR

ORG      0100H
PCAISR:
CLR      CCF0
CPL      P1.0
RETI

MAIN:
MOV      SP,#3FH

MOV      CCON,#00H
MOV      CMOD,#08H          ;PCA 时钟为系统时钟
MOV      CL,#00H
MOV      CH,#0H
MOV      CCAPM0,#11H         ;扩展外部端口 CCP0 为下降沿中断口
;      MOV      CCAPM0,#21H         ;扩展外部端口 CCP0 为上升沿中断口
;      MOV      CCAPM0,#31H         ;扩展外部端口 CCP0 为边沿中断口
MOV      CCAP0L,#0
MOV      CCAP0H,#0
SETB    CR                  ;启动 PCA 计时器
SETB    EA

```

JMP \$**END**

C 语言代码

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

//测试工作频率为 11.0592MHz

sfr CCON = 0xd8;
sbit CF = CCON^7;
sbit CR = CCON^6;
sbit CCF3 = CCON^3;
sbit CCF2 = CCON^2;
sbit CCF1 = CCON^1;
sbit CCF0 = CCON^0;
sfr CMOD = 0xd9;
sfr CL = 0xe9;
sfr CH = 0xf9;
sfr CCAPM0 = 0xda;
sfr CCAP0L = 0xea;
sfr CCAP0H = 0xfa;
sfr PCA_PWM0 = 0xf2;
sfr CCAPM1 = 0xdb;
sfr CCAPIL = 0xeb;
sfr CCAPIH = 0xfb;
sfr PCA_PWM1 = 0xf3;
sfr CCAPM2 = 0xdc;
sfr CCAP2L = 0xec;
sfr CCAP2H = 0xfc;
sfr PCA_PWM2 = 0xf4;
sfr CCAPM3 = 0xdd;
sfr CCAP3L = 0xed;
sfr CCAP3H = 0xfd;
sfr PCA_PWM3 = 0xf5;

sbit PI0 = PI^0;

void PCA_Isr() interrupt 7
{
    CCF0 = 0;
    PI0 = !PI0;
}

void main()
{
    CCON = 0x00;                                //PCA 时钟为系统时钟
    CMOD = 0x08;
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x11;                                //扩展外部端口 CCP0 为下降沿中断口
//    CCAPM0 = 0x21;                                //扩展外部端口 CCP0 为上升沿中断口
//    CCAPM0 = 0x31;                                //扩展外部端口 CCP0 为边沿中断口
    CCAP0L = 0;
    CCAP0H = 0;
    CR = 1;                                         //启动 PCA 计时器
    EA = 1;
}
```

```
    while (1);  
}
```

STCMCU

18 增强型PWM

STC8 系列单片机集成了一组（各自独立 8 路）增强型的 PWM 波形发生器。PWM 波形发生器内部有一个 15 位的 PWM 计数器供 8 路 PWM 使用，用户可以设置每路 PWM 的初始电平。另外，PWM 波形发生器为每路 PWM 又设计了两个用于控制波形翻转的计数器 T1/T2，可以非常灵活的每路 PWM 的高低电平宽度，从而达到对 PWM 的占空比以及 PWM 的输出延迟进行控制的目的。由于 8 路 PWM 是各自独立的，且每路 PWM 的初始状态可以进行设定，所以用户可以将其中的任意两路配合起来使用，即可实现互补对称输出以及死区控制等特殊应用。

增强型的 PWM 波形发生器还设计了对外部异常事件（包括外部端口 P3.5 电平异常、比较器比较结果异常）进行监控的功能，可用于紧急关闭 PWM 输出。PWM 波形发生器还可与 ADC 相关联，设置 PWM 周期的任一时间点触发 ADC 转换事件。

18.1 PWM相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
PWMCFG	增强型 PWM 配置寄存器	F1H	CBIF	ETADC	-	-	-	-	-	-	00xx,xxxx
PWMIF	增强型 PWM 中断标志寄存器	F6H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWMFDCR	PWM 异常检测控制寄存器	F7H	INVCOMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	0000,0000
PWMCR	PWM 控制寄存器	FEH	ENPWM	ECBI	-	-	-	-	-	-	00xx,xxxx

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
PWMCH	PWM 计数器高字节	FFF0H	-								x000,0000
PWMCL	PWM 计数器低字节	FFF1H									0000,0000
PWMCKS	PWM 时钟选择	FFF2H	-	-	-	SELT2	PWM_PS[3:0]				xxx0,0000
TADCPH	触发 ADC 计数值高字节	FFF3H	-								x000,0000
TADCPL	触发 ADC 计数值低字节	FFF4H									0000,0000
PWM0T1H	PWM0T1 计数值高字节	FF00H	-								x000,0000
PWM0T1L	PWM0T1 计数值低节	FF01H									0000,0000
PWM0T2H	PWM0T2 数值高字节	FF02H	-								x000,0000
PWM0T2L	PWM0T2 数值低节	FF03H									0000,0000
PWM0CR	PWM0 控制寄存器	FF04H	ENC0O	C0INI	-	C0_S[1:0]		EC0I	EC0T2SI	EC0T1SI	00x0,0000
PWM0HLD	PWM0 电平保持控制寄存器	FF05H	-	-	-	-	-	-	HC0H	HC0L	xxxx,xx00
PWM1T1H	PWM1T1 计数值高字节	FF10H	-								x000,0000
PWM1T1L	PWM1T1 计数值低节	FF11H									0000,0000
PWM1T2H	PWM1T2 数值高字节	FF12H	-								x000,0000
PWM1T2L	PWM1T2 数值低节	FF13H									0000,0000
PWM1CR	PWM1 控制寄存器	FF14H	ENC1O	C1INI	-	C1_S[1:0]		EC1I	EC1T2SI	EC1T1SI	00x0,0000
PWM1HLD	PWM1 电平保持控制寄存器	FF15H	-	-	-	-	-	-	HC1H	HC1L	xxxx,xx00
PWM2T1H	PWM2T1 计数值高字节	FF20H	-								x000,0000
PWM2T1L	PWM2T1 计数值低节	FF21H									0000,0000
PWM2T2H	PWM2T2 数值高字节	FF22H	-								x000,0000

PWM2T2L	PWM2T2 数值低节	FF23H	0000,0000									
PWM2CR	PWM2 控制寄存器	FF24H	ENC2O	C2INI	-	C2_S[1:0]		EC2I	EC2T2SI	EC2T1SI	00x0,0000	
PWM2HLD	PWM2 电平保持控制寄存器	FF25H	-	-	-	-	-	-	HC2H	HC2L	xxxx,xx00	
PWM3T1H	PWM3T1 计数值高字节	FF30H	-	x000,0000								
PWM3T1L	PWM3T1 计数值低节	FF31H	0000,0000									
PWM3T2H	PWM3T2 数值高字节	FF32H	-	x000,0000								
PWM3T2L	PWM3T2 数值低节	FF33H	0000,0000									
PWM3CR	PWM3 控制寄存器	FF34H	ENC3O	C3INI	-	C3_S[1:0]		EC3I	EC3T2SI	EC3T1SI	00x0,0000	
PWM3HLD	PWM3 电平保持控制寄存器	FF35H	-	-	-	-	-	-	HC3H	HC3L	xxxx,xx00	
PWM4T1H	PWM4T1 计数值高字节	FF40H	-	x000,0000								
PWM4T1L	PWM4T1 计数值低节	FF41H	0000,0000									
PWM4T2H	PWM4T2 数值高字节	FF42H	-	x000,0000								
PWM4T2L	PWM4T2 数值低节	FF43H	0000,0000									
PWM4CR	PWM4 控制寄存器	FF44H	ENC4O	C4INI	-	C4_S[1:0]		EC4I	EC4T2SI	EC4T1SI	00x0,0000	
PWM4HLD	PWM4 电平保持控制寄存器	FF45H	-	-	-	-	-	-	HC4H	HC4L	xxxx,xx00	
PWM5T1H	PWM5T1 计数值高字节	FF50H	-	x000,0000								
PWM5T1L	PWM5T1 计数值低节	FF51H	0000,0000									
PWM5T2H	PWM5T2 数值高字节	FF52H	-	x000,0000								
PWM5T2L	PWM5T2 数值低节	FF53H	0000,0000									
PWM5CR	PWM5 控制寄存器	FF54H	ENC5O	C5INI	-	C5_S[1:0]		EC5I	EC5T2SI	EC5T1SI	00x0,0000	
PWM5HLD	PWM5 电平保持控制寄存器	FF55H	-	-	-	-	-	-	HC5H	HC5L	xxxx,xx00	
PWM6T1H	PWM6T1 计数值高字节	FF60H	-	x000,0000								
PWM6T1L	PWM6T1 计数值低节	FF61H	0000,0000									
PWM6T2H	PWM6T2 数值高字节	FF62H	-	x000,0000								
PWM6T2L	PWM6T2 数值低节	FF63H	0000,0000									
PWM6CR	PWM6 控制寄存器	FF64H	ENC6O	C6INI	-	C6_S[1:0]		EC6I	EC6T2SI	EC6T1SI	00x0,0000	
PWM6HLD	PWM6 电平保持控制寄存器	FF65H	-	-	-	-	-	-	HC6H	HC6L	xxxx,xx00	
PWM7T1H	PWM7T1 计数值高字节	FF70H	-	x000,0000								
PWM7T1L	PWM7T1 计数值低节	FF71H	0000,0000									
PWM7T2H	PWM7T2 数值高字节	FF72H	-	x000,0000								
PWM7T2L	PWM7T2 数值低节	FF73H	0000,0000									
PWM7CR	PWM7 控制寄存器	FF74H	ENC7O	C7INI	-	C7_S[1:0]		EC7I	EC7T2SI	EC7T1SI	00x0,0000	
PWM7HLD	PWM7 电平保持控制寄存器	FF75H	-	-	-	-	-	-	HC7H	HC7L	xxxx,xx00	

PWM 配置寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMCFG	F1H	CBIF	ETADC	-	-	-	-	-	-

CBIF: PWM 计数器归零中断标志位。

当 15 位的 PWM 计数器记满溢出归零时，硬件自动将此位置 1，并向 CPU 提出中断请求，此标志位需要软件清零。

ETADC: PWM 是否与 ADC 关联

0: PWM 与 ADC 不关联

1: PWM 与 ADC 相关联。允许在 PWM 周期中某个时间点触发 A/D 转换。使用 TADCPH 和 TADCPL 进行设置。

PWM 中断标志寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMIF	F6H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF

CnIF: 第 n 通道 PWM 的中断标志位。

可设置在各路 PWM 的翻转点 1 和翻转点 2。当所设置的翻转点发生翻转事件时，硬件自动将此位置 1，并向 CPU 提出中断请求，此标志位需要软件清零。

PWM 异常检测控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMFDCR	F7H	INVCMP	INVIO	ENFD	FLTFLO	EFDI	FDCMP	FDIO	FDIF

INVCMP: 比较器器结果异常信号处理

- 0: 比较器器结果由低变高为异常信号
- 1: 比较器器结果由高变低为异常信号

INVIO: 外部端口 P3.5 异常信号处理

- 0: 外部端口 P3.5 信号由低变高为异常信号
- 1: 外部端口 P3.5 信号由高变低为异常信号

ENFD: PWM 外部异常检测控制位

- 0: 关闭 PWM 外部异常检测功能
- 1: 使能 PWM 外部异常检测功能

FLTFLO: 发生 PWM 外部异常时对 PWM 输出口控制位

- 0: 发生 WM 外部异常时，PWM 的输出口不作任何改变
- 1: 发生 WM 外部异常时，PWM 的输出口立即被设置为高阻输入模式。(注：只有 ENCnO=1 所对应的端口才会被强制悬空)

EFDI: PWM 异常检测中断使能位

- 0: 关闭 PWM 异常检测中断 (FDIF 依然会被硬件置位)
- 1: 使能 PWM 异常检测中断

FDCMP: 比较器输出异常检测使能位

- 0: 比较器与 PWM 无关
- 1: 设定 PWM 异常检测源为比较器输出 (异常类型有 INVCMP 设定)

FDIO: P3.5 口电平异常检测使能位

- 0: P3.5 口电平与 PWM 无关
- 1: 设定 PWM 异常检测源为 P3.5 口 (异常类型有 INVIO 设定)

FDIF: PWM 异常检测中断标志位

当发生 PWM 异常 (比较器的输出由低变高或者 P3.5 的电平由低变高) 时，硬件自动将此位置 1。

当 EFDI==1 时，程序会跳转到相应入口执行中断服务程序。需要软件清零。

PWM 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMCR	FEH	ENPWM	ECBI	-	-	-	-	-	-

ENPWM: 使能增强型 PWM 波形发生器

- 0: 关闭 PWM 波形发生器
- 1: 使能 PWM 波形发生器，PWM 计数器开始计数

关于 ENPWM 控制位的重要说明:

- ENPWM 一旦被使能后，内部的 PWM 计数器会立即开始计数，并与 T1/T2 两个翻转点的值进行比较。所以 ENPWM 必须在其他所有的 PWM 设置（包括 T1/T2 翻转点的设置、初始电平的设置、PWM 异常检测的设置以及 PWM 中断设置）都完成后，最后才能使能 ENPWM 位。
- ENPWM 控制位既是整个 PWM 模块的使能位，也是 PWM 计数器开始计数的控制位。在 PWM 计数器计数的过程中，ENPWM 控制位被关闭时，PWM 计数会立即停止，当再次使能 ENPWM 控制位时，PWM 的计数会从 0 开始重新计数，而不会记忆 PWM 停止计数前的计数值

ECBI: PWM 计数器归零中断使能位

- 0: 关闭 PWM 计数器归零中断 (CBIF 依然会被硬件置位)
1: 使能 PWM 计数器归零中断

PWM 计数器寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMCH	FFF0H	-							
PWMCL	FFF1H								

PWM 计数器位一个 15 位的寄存器，可设定 1~32767 之间的任意值作为 PWM 的周期。PWM 波形发生器内部的计数器从 0 开始计数，每个 PWM 时钟周期递增 1，当内部计数器的计数值达到[PWMCH, PWMCL]所设定的 PWM 周期时，PWM 波形发生器内部的计数器将会从 0 重新开始开始计数，硬件会自动将 PWM 归零中断标志位 CBIF 置 1，若 ECBI=1，程序将跳转到相应中断入口执行中断服务程序。

PWM 时钟选择寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMCKS	FFF2H	-	-	-	SELT2				PWM_PS[3:0]

SELT2: PWM 时钟源选择

- 0: PWM 时钟源为系统时钟经分频器分频之后的时钟
1: PWM 时钟源为定时器 2 的溢出脉冲

PWM_PS[3:0]: 系统时钟预分频参数

SELT2	PWM_PS[3:0]	PWM 输入时钟源频率
1	xxxx	定时器 2 的溢出脉冲
0	0000	SYSclk/1
0	0001	SYSclk/2
0	0010	SYSclk/3
...
0	x	SYSclk/(x+1)
...
0	1111	SYSclk/16

PWM 触发 ADC 计数器寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TADCPH	FFF3H	-							
TADCPL	FFF4H								

在 ETADC=1 且 ADC_POWER=1 时，{TADCPH,TADCPL} 组成一个 15 位的寄存器。在 PWM 的计数周期中，当 PWM 的内部计数值与{TADCPH,TADCPL}的值相等时，硬件自动触发 A/D 转换。

PWM 翻转点设置计数值寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWM0T1H	FF00H	-							
PWM0T1L	FF01H								
PWM0T2H	FF02H	-							
PWM0T2L	FF03H								
PWM1T1H	FF10H	-							
PWM1T1L	FF11H								
PWM1T2H	FF12H	-							
PWM1T2L	FF13H								
PWM2T1H	FF20H	-							
PWM2T1L	FF21H								
PWM2T2H	FF22H	-							
PWM2T2L	FF23H								
PWM3T1H	FF30H	-							
PWM3T1L	FF31H								
PWM3T2H	FF32H	-							
PWM3T2L	FF33H								
PWM4T1H	FF40H	-							
PWM4T1L	FF41H								
PWM4T2H	FF42H	-							
PWM4T2L	FF43H								
PWM5T1H	FF50H	-							
PWM5T1L	FF51H								
PWM5T2H	FF52H	-							
PWM5T2L	FF53H								
PWM6T1H	FF60H	-							
PWM6T1L	FF61H								
PWM6T2H	FF62H	-							
PWM6T2L	FF63H								
PWM7T1H	FF70H	-							
PWM7T1L	FF71H								
PWM7T2H	FF72H	-							
PWM7T2L	FF73H								

PWM 每个通道的 {PWMnT1H, PWMnT1L} 和 {PWMnT2H, PWMnT2L} 分别组合成两个 15 位的寄存器，用于控制各路 PWM 每个周期中输出 PWM 波形的两个翻转点。在 PWM 的计数周期中，当 PWM 的内部计数值与所设置的第 1 个翻转点的值 {PWMnT1H, PWMnT1L} 相等时，PWM 的输出波形会自动翻转为低电平；当 PWM 的内部计数值与所设置的第 2 个翻转点的值 {PWMnT2H, PWMnT2L} 相等时，PWM 的输出波形会自动翻转为高电平。

注意：当 {PWMnT1H, PWMnT1L} 与 {PWMnT2H, PWMnT2L} 的值设置相等时，第 2 组翻转点的匹配将被忽略，即只会翻转为低电平。

PWM 通道控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
----	----	----	----	----	----	----	----	----	----

PWM0CR	FF04H	ENC0O	C0INI	-	C0_S[1:0]	EC0I	EC0T2SI	EC0T1SI
PWM1CR	FF14H	ENC1O	C1INI	-	C1_S[1:0]	EC1I	EC1T2SI	EC1T1SI
PWM2CR	FF24H	ENC2O	C2INI	-	C2_S[1:0]	EC2I	EC2T2SI	EC2T1SI
PWM3CR	FF34H	ENC3O	C3INI	-	C3_S[1:0]	EC3I	EC3T2SI	EC3T1SI
PWM4CR	FF44H	ENC4O	C4INI	-	C4_S[1:0]	EC4I	EC4T2SI	EC4T1SI
PWM5CR	FF54H	ENC5O	C5INI	-	C5_S[1:0]	EC5I	EC5T2SI	EC5T1SI
PWM6CR	FF64H	ENC6O	C6INI	-	C6_S[1:0]	EC6I	EC6T2SI	EC6T1SI
PWM7CR	FF74H	ENC7O	C7INI	-	C7_S[1:0]	EC7I	EC7T2SI	EC7T1SI

ENCnO: PWM 输出使能位

0: 相应 PWM 通道的端口为 GPIO

1: 相应 PWM 通道的端口为 PWM 输出口, 受 PWM 波形发生器控制

CnINI: 设置 PWM 输出端口的初始电平

0: 第 n 通道的 PWM 初始电平为低电平

1: 第 n 通道的 PWM 初始电平为高电平

Cn_S[1:0]: PWM 输出功能脚切换选择, 请参考功能脚切换章节。

ECnI: 第 n 通道的 PWM 中断使能控制位

0: 关闭第 n 通道的 PWM 中断

1: 使能第 n 通道的 PWM 中断

ECnT2SI: 第 n 通道的 PWM 在第 2 个翻转点中断使能控制位

0: 关闭第 n 通道的 PWM 在第 2 个翻转点中断

1: 使能第 n 通道的 PWM 在第 2 个翻转点中断

ECnT1SI: 第 n 通道的 PWM 在第 1 个翻转点中断使能控制位

0: 关闭第 n 通道的 PWM 在第 1 个翻转点中断

1: 使能第 n 通道的 PWM 在第 1 个翻转点中断

PWM 通道电平保持控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWM0HLD	FF05H	-	-	-	-	-	-	HC0H	HC0L
PWM1HLD	FF15H	-	-	-	-	-	-	HC1H	HC1L
PWM2HLD	FF25H	-	-	-	-	-	-	HC2H	HC2L
PWM3HLD	FF35H	-	-	-	-	-	-	HC3H	HC3L
PWM4HLD	FF45H	-	-	-	-	-	-	HC4H	HC4L
PWM5HLD	FF55H	-	-	-	-	-	-	HC5H	HC5L
PWM6HLD	FF65H	-	-	-	-	-	-	HC6H	HC6L
PWM7HLD	FF75H	-	-	-	-	-	-	HC7H	HC7L

HCnH: 第 n 通道 PWM 强制输出高电平控制位

0: 第 n 通道 PWM 正常输出

1: 第 n 通道 PWM 强制输出高电平

HCnL: 第 n 通道 PWM 强制输出低电平控制位

0: 第 n 通道 PWM 正常输出

1: 第 n 通道 PWM 强制输出低电平

注意事项:

1. 8 通道增强型 PWM 的输出, 切换到 P1.0~P1.7 或 P2.0~P2.7, IO 设置为准双向口或推挽输出, PWM 输出正常, 均为推挽输出, 并且再直接操作 IO 将不影响 PWM 波形。但是如果将 IO 设置为开漏输出(允许内部上拉电阻或外接上拉电阻), 则如果对应的 IO 输出高电平, PWM 无输出 (IO 为高阻), 而对应的 IO 输出低电平, PWM 有推挽输出。

STCMCU

18.2 范例程序

18.2.1 输出任意周期和任意占空比的波形

汇编代码

; 测试工作频率为 11.0592MHz

P_SW2	DATA	0BAH
PWMCFG	DATA	0F1H
PWMIF	DATA	0F6H
PWMFDCR	DATA	0F7H
PWMCR	DATA	0FEH
PWMCH	XDATA	0FFF0H
PWMCL	XDATA	0FFF1H
PWMCKS	XDATA	0FFF2H
TADCPH	XDATA	0FFF3H
TADCPL	XDATA	0FFF4H
PWM0T1H	XDATA	0FF00H
PWM0T1L	XDATA	0FF01H
PWM0T2H	XDATA	0FF02H
PWM0T2L	XDATA	0FF03H
PWM0CR	XDATA	0FF04H
PWM0HLD	XDATA	0FF05H
PWM1T1H	XDATA	0FF10H
PWM1T1L	XDATA	0FF11H
PWM1T2H	XDATA	0FF12H
PWM1T2L	XDATA	0FF13H
PWM1CR	XDATA	0FF14H
PWM1HLD	XDATA	0FF15H
PWM2T1H	XDATA	0FF20H
PWM2T1L	XDATA	0FF21H
PWM2T2H	XDATA	0FF22H
PWM2T2L	XDATA	0FF23H
PWM2CR	XDATA	0FF24H
PWM2HLD	XDATA	0FF25H
PWM3T1H	XDATA	0FF30H
PWM3T1L	XDATA	0FF31H
PWM3T2H	XDATA	0FF32H
PWM3T2L	XDATA	0FF33H
PWM3CR	XDATA	0FF34H
PWM3HLD	XDATA	0FF35H
PWM4T1H	XDATA	0FF40H
PWM4T1L	XDATA	0FF41H
PWM4T2H	XDATA	0FF42H
PWM4T2L	XDATA	0FF43H
PWM4CR	XDATA	0FF44H
PWM4HLD	XDATA	0FF45H
PWM5T1H	XDATA	0FF50H
PWM5T1L	XDATA	0FF51H
PWM5T2H	XDATA	0FF52H
PWM5T2L	XDATA	0FF53H
PWM5CR	XDATA	0FF54H
PWM5HLD	XDATA	0FF55H
PWM6T1H	XDATA	0FF60H
PWM6T1L	XDATA	0FF61H
PWM6T2H	XDATA	0FF62H
PWM6T2L	XDATA	0FF63H

```

PWM6CR      XDATA    0FF64H
PWM6HLD     XDATA    0FF65H
PWM7T1H     XDATA    0FF70H
PWM7T1L     XDATA    0FF71H
PWM7T2H     XDATA    0FF72H
PWM7T2L     XDATA    0FF73H
PWM7CR      XDATA    0FF74H
PWM7HLD     XDATA    0FF75H

        ORG      0000H
        LJMP     MAIN

        ORG      0100H

MAIN:
        MOV      P_SW2,#80H
        CLR      A
        MOV      DPTR,#PWMCKS
        MOVX    @DPTR,A           ; PWM 时钟为系统时钟
        MOV      A,#10H
        MOV      DPTR,#PWMCH       ; 设置 PWM 周期为 1000H 个 PWM 时钟
        MOVX    @DPTR,A
        MOV      A,#00H
        MOV      DPTR,#PWMCL
        MOVX    @DPTR,A
        MOV      A,#01H
        MOV      DPTR,#PWM0T1H       ; 在计数值为 100H 地方输出低电平
        MOVX    @DPTR,A
        MOV      A,#00H
        MOV      DPTR,#PWM0T1L
        MOVX    @DPTR,A
        MOV      A,#05H
        MOV      DPTR,#PWM0T2H       ; 在计数值为 500H 地方输出高电平
        MOVX    @DPTR,A
        MOV      A,#00H
        MOV      DPTR,#PWM0T2L
        MOVX    @DPTR,A
        MOV      A,#80H
        MOV      DPTR,#PWM0CR        ; 使能 PWM0 输出
        MOVX    @DPTR,A
        MOV      P_SW2,#00H

        MOV      PWMCR,#080H        ; 启动 PWM 模块

        JMP     $

END

```

C 语言代码

```

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

// 测试工作频率为 11.0592MHz

sfr P_SW2 = 0xba;
sfr PWMCFG = 0xf1;
sfr PWMIF = 0xf6;
sfr PWMFDCR = 0xf7;
sfr PWMCR = 0xfe;

```

```

#define  PWM_C          (*(unsigned int volatile xdata *)0xffff0)
#define  PWMCKS         (*(unsigned char volatile xdata *)0xffff2)
#define  TADCP          (*(unsigned int volatile xdata *)0xffff3)
#define  PWM0T1          (*(unsigned int volatile xdata *)0xff00)
#define  PWM0T2          (*(unsigned int volatile xdata *)0xff02)
#define  PWM0CR          (*(unsigned char volatile xdata *)0xff04)
#define  PWM0HLD         (*(unsigned char volatile xdata *)0xff05)
#define  PWM1T1          (*(unsigned int volatile xdata *)0xff10)
#define  PWM1T2          (*(unsigned int volatile xdata *)0xff12)
#define  PWM1CR          (*(unsigned char volatile xdata *)0xff14)
#define  PWM1HLD         (*(unsigned char volatile xdata *)0xff15)
#define  PWM2T1          (*(unsigned int volatile xdata *)0xff20)
#define  PWM2T2          (*(unsigned int volatile xdata *)0xff22)
#define  PWM2CR          (*(unsigned char volatile xdata *)0xff24)
#define  PWM2HLD         (*(unsigned char volatile xdata *)0xff25)
#define  PWM3T1          (*(unsigned int volatile xdata *)0xff30)
#define  PWM3T2          (*(unsigned int volatile xdata *)0xff32)
#define  PWM3CR          (*(unsigned char volatile xdata *)0xff34)
#define  PWM3HLD         (*(unsigned char volatile xdata *)0xff35)
#define  PWM4T1          (*(unsigned int volatile xdata *)0xff40)
#define  PWM4T2          (*(unsigned int volatile xdata *)0xff42)
#define  PWM4CR          (*(unsigned char volatile xdata *)0xff44)
#define  PWM4HLD         (*(unsigned char volatile xdata *)0xff45)
#define  PWM5T1          (*(unsigned int volatile xdata *)0xff50)
#define  PWM5T2          (*(unsigned int volatile xdata *)0xff52)
#define  PWM5CR          (*(unsigned char volatile xdata *)0xff54)
#define  PWM5HLD         (*(unsigned char volatile xdata *)0xff55)
#define  PWM6T1          (*(unsigned int volatile xdata *)0xff60)
#define  PWM6T2          (*(unsigned int volatile xdata *)0xff62)
#define  PWM6CR          (*(unsigned char volatile xdata *)0xff64)
#define  PWM6HLD         (*(unsigned char volatile xdata *)0xff65)
#define  PWM7T1          (*(unsigned int volatile xdata *)0xff70)
#define  PWM7T2          (*(unsigned int volatile xdata *)0xff72)
#define  PWM7CR          (*(unsigned char volatile xdata *)0xff74)
#define  PWM7HLD         (*(unsigned char volatile xdata *)0xff75)

void main()
{
    P_SW2 = 0x80;
    PWMCKS = 0x00;           // PWM 时钟为系统时钟
    PWM_C = 0x1000;          // 设置 PWM 周期为 1000H 个 PWM 时钟
    PWM0T1= 0x0100;          // 在计数值为 100H 地方输出低电平
    PWM0T2= 0x0500;          // 在计数值为 500H 地方输出高电平
    PWM0CR= 0x80;            // 使能 PWM0 输出
    P_SW2 = 0x00;

    PWMCR = 0x80;           // 启动 PWM 模块

    while (1);
}

```

18.2.2 两路PWM实现互补对称带死区控制的波形

汇编代码

; 测试工作频率为 11.0592MHz

P_SW2	DATA	0BAH
-------	------	------

PWMCFG	DATA	0F1H
PWMIF	DATA	0F6H
PWMFDCR	DATA	0F7H
PWMCR	DATA	0FEH
PWMCH	XDATA	0FFF0H
PWMCL	XDATA	0FFF1H
PWMCKS	XDATA	0FFF2H
TADCPH	XDATA	0FFF3H
TADCPL	XDATA	0FFF4H
PWM0T1H	XDATA	0FF00H
PWM0T1L	XDATA	0FF01H
PWM0T2H	XDATA	0FF02H
PWM0T2L	XDATA	0FF03H
PWM0CR	XDATA	0FF04H
PWM0HLD	XDATA	0FF05H
PWM1T1H	XDATA	0FF10H
PWM1T1L	XDATA	0FF11H
PWM1T2H	XDATA	0FF12H
PWM1T2L	XDATA	0FF13H
PWM1CR	XDATA	0FF14H
PWM1HLD	XDATA	0FF15H
PWM2T1H	XDATA	0FF20H
PWM2T1L	XDATA	0FF21H
PWM2T2H	XDATA	0FF22H
PWM2T2L	XDATA	0FF23H
PWM2CR	XDATA	0FF24H
PWM2HLD	XDATA	0FF25H
PWM3T1H	XDATA	0FF30H
PWM3T1L	XDATA	0FF31H
PWM3T2H	XDATA	0FF32H
PWM3T2L	XDATA	0FF33H
PWM3CR	XDATA	0FF34H
PWM3HLD	XDATA	0FF35H
PWM4T1H	XDATA	0FF40H
PWM4T1L	XDATA	0FF41H
PWM4T2H	XDATA	0FF42H
PWM4T2L	XDATA	0FF43H
PWM4CR	XDATA	0FF44H
PWM4HLD	XDATA	0FF45H
PWM5T1H	XDATA	0FF50H
PWM5T1L	XDATA	0FF51H
PWM5T2H	XDATA	0FF52H
PWM5T2L	XDATA	0FF53H
PWM5CR	XDATA	0FF54H
PWM5HLD	XDATA	0FF55H
PWM6T1H	XDATA	0FF60H
PWM6T1L	XDATA	0FF61H
PWM6T2H	XDATA	0FF62H
PWM6T2L	XDATA	0FF63H
PWM6CR	XDATA	0FF64H
PWM6HLD	XDATA	0FF65H
PWM7T1H	XDATA	0FF70H
PWM7T1L	XDATA	0FF71H
PWM7T2H	XDATA	0FF72H
PWM7T2L	XDATA	0FF73H
PWM7CR	XDATA	0FF74H
PWM7HLD	XDATA	0FF75H

<i>ORG</i>	<i>0000H</i>	
<i>LJMP</i>	<i>MAIN</i>	
<i>ORG</i>	<i>0100H</i>	
<i>MAIN:</i>		
<i>MOV</i>	<i>P_SW2,#80H</i>	
<i>CLR</i>	<i>A</i>	
<i>MOV</i>	<i>DPTR,#PWMCKS</i>	
<i>MOVX</i>	<i>@DPTR,A</i>	<i>; PWM 时钟为系统时钟</i>
<i>MOV</i>	<i>A,#08H</i>	
<i>MOV</i>	<i>DPTR,#PWMCH</i>	<i>; 设置 PWM 周期为 0800H 个 PWM 时钟</i>
<i>MOVX</i>	<i>@DPTR,A</i>	
<i>MOV</i>	<i>A,#00H</i>	
<i>MOV</i>	<i>DPTR,#PWMCL</i>	
<i>MOVX</i>	<i>@DPTR,A</i>	
<i>MOV</i>	<i>A,#01H</i>	
<i>MOV</i>	<i>DPTR,#PWM0T1H</i>	<i>; PWM0 在计数值为 0100H 地方输出低电平</i>
<i>MOVX</i>	<i>@DPTR,A</i>	
<i>MOV</i>	<i>A,#00H</i>	
<i>MOV</i>	<i>DPTR,#PWM0T1L</i>	
<i>MOVX</i>	<i>@DPTR,A</i>	
<i>MOV</i>	<i>A,#07H</i>	
<i>MOV</i>	<i>DPTR,#PWM0T2H</i>	<i>; PWM0 在计数值为 0700H 地方输出高电平</i>
<i>MOVX</i>	<i>@DPTR,A</i>	
<i>MOV</i>	<i>A,#00H</i>	
<i>MOV</i>	<i>DPTR,#PWM0T2L</i>	
<i>MOVX</i>	<i>@DPTR,A</i>	
<i>MOV</i>	<i>A,#00H</i>	
<i>MOV</i>	<i>DPTR,#PWM1T2H</i>	<i>; PWM1 在计数值为 0080H 地方输出高电平</i>
<i>MOVX</i>	<i>@DPTR,A</i>	
<i>MOV</i>	<i>A,#80H</i>	
<i>MOV</i>	<i>DPTR,#PWM1T2L</i>	
<i>MOVX</i>	<i>@DPTR,A</i>	
<i>MOV</i>	<i>A,#07H</i>	
<i>MOV</i>	<i>DPTR,#PWM1T1H</i>	<i>; PWM1 在计数值为 0780H 地方输出低电平</i>
<i>MOVX</i>	<i>@DPTR,A</i>	
<i>MOV</i>	<i>A,#80H</i>	
<i>MOV</i>	<i>DPTR,#PWM1T1L</i>	
<i>MOVX</i>	<i>@DPTR,A</i>	
<i>MOV</i>	<i>A,#080H</i>	
<i>MOV</i>	<i>DPTR,#PWM0CR</i>	<i>; 使能 PWM0 输出</i>
<i>MOVX</i>	<i>@DPTR,A</i>	
<i>MOV</i>	<i>A,#80H</i>	
<i>MOV</i>	<i>DPTR,#PWM1CR</i>	<i>; 使能 PWM1 输出</i>
<i>MOVX</i>	<i>@DPTR,A</i>	
<i>MOV</i>	<i>P_SW2,#00H</i>	
 <i>MOV</i>	<i>PWMCR,#080H</i>	<i>; 启动 PWM 模块</i>
 <i>JMP</i>	\$	
 <i>END</i>		

C 语言代码

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

```
// 测试工作频率为 11.0592MHz
```

```

sfr P_SW2      = 0xba;
sfr PWMCFG     = 0xf1;
sfr PWMIF      = 0xf6;
sfr PWMFDCR    = 0xf7;
sfr PWMCR      = 0xfe;

#define PWMC      (*(unsigned int volatile xdata *)0xffff0)
#define PWMCKS    (*(unsigned char volatile xdata *)0xffff2)
#define TADCP     (*(unsigned int volatile xdata *)0xffff3)
#define PWM0T1    (*(unsigned int volatile xdata *)0xffff0)
#define PWM0T2    (*(unsigned int volatile xdata *)0xff02)
#define PWM0CR    (*(unsigned char volatile xdata *)0xff04)
#define PWM0HLD   (*(unsigned char volatile xdata *)0xff05)
#define PWM1T1    (*(unsigned int volatile xdata *)0xff10)
#define PWM1T2    (*(unsigned int volatile xdata *)0xff12)
#define PWM1CR    (*(unsigned char volatile xdata *)0xff14)
#define PWM1HLD   (*(unsigned char volatile xdata *)0xff15)
#define PWM2T1    (*(unsigned int volatile xdata *)0xff20)
#define PWM2T2    (*(unsigned int volatile xdata *)0xff22)
#define PWM2CR    (*(unsigned char volatile xdata *)0xff24)
#define PWM2HLD   (*(unsigned char volatile xdata *)0xff25)
#define PWM3T1    (*(unsigned int volatile xdata *)0xff30)
#define PWM3T2    (*(unsigned int volatile xdata *)0xff32)
#define PWM3CR    (*(unsigned char volatile xdata *)0xff34)
#define PWM3HLD   (*(unsigned char volatile xdata *)0xff35)
#define PWM4T1    (*(unsigned int volatile xdata *)0xff40)
#define PWM4T2    (*(unsigned int volatile xdata *)0xff42)
#define PWM4CR    (*(unsigned char volatile xdata *)0xff44)
#define PWM4HLD   (*(unsigned char volatile xdata *)0xff45)
#define PWM5T1    (*(unsigned int volatile xdata *)0xff50)
#define PWM5T2    (*(unsigned int volatile xdata *)0xff52)
#define PWM5CR    (*(unsigned char volatile xdata *)0xff54)
#define PWM5HLD   (*(unsigned char volatile xdata *)0xff55)
#define PWM6T1    (*(unsigned int volatile xdata *)0xff60)
#define PWM6T2    (*(unsigned int volatile xdata *)0xff62)
#define PWM6CR    (*(unsigned char volatile xdata *)0xff64)
#define PWM6HLD   (*(unsigned char volatile xdata *)0xff65)
#define PWM7T1    (*(unsigned int volatile xdata *)0xff70)
#define PWM7T2    (*(unsigned int volatile xdata *)0xff72)
#define PWM7CR    (*(unsigned char volatile xdata *)0xff74)
#define PWM7HLD   (*(unsigned char volatile xdata *)0xff75)

void main()
{
    P_SW2 = 0x80;                                // PWM 时钟为系统时钟
    PWMCKS = 0x00;                               // 设置 PWM 周期为 0800H 个 PWM 时钟
    PWMC = 0x0800;                               // PWM0 在计数值为 100H 地方输出低电平
    PWM0T1= 0x0100;                             // PWM0 在计数值为 700H 地方输出高电平
    PWM0T2= 0x0700;                             // PWM1 在计数值为 0080H 地方输出高电平
    PWM1T2= 0x0080;                             // PWM1 在计数值为 0780H 地方输出低电平
    PWM0CR= 0x80;                                // 使能 PWM0 输出
    PWM1CR= 0x80;                                // 使能 PWM1 输出
    P_SW2 = 0x00;                                // 启动 PWM 模块

    PWMCR = 0x80;                                // 启动 PWM 模块

    while (1);
}

```

}

18.2.3 PWM实现渐变灯（呼吸灯）

汇编代码

; 测试工作频率为 11.0592MHz

CYCLE	EQU	1000H
P_SW2	DATA	0BAH
PWMCFG	DATA	0F1H
PWMIF	DATA	0F6H
PWMFDCR	DATA	0F7H
PWMCR	DATA	0FEH
PWMCH	XDATA	0FFF0H
PWMCL	XDATA	0FFF1H
PWMCKS	XDATA	0FFF2H
TADCPH	XDATA	0FFF3H
TADCPL	XDATA	0FFF4H
PWM0T1H	XDATA	0FF00H
PWM0T1L	XDATA	0FF01H
PWM0T2H	XDATA	0FF02H
PWM0T2L	XDATA	0FF03H
PWM0CR	XDATA	0FF04H
PWM0HLD	XDATA	0FF05H
PWM1T1H	XDATA	0FF10H
PWM1T1L	XDATA	0FF11H
PWM1T2H	XDATA	0FF12H
PWM1T2L	XDATA	0FF13H
PWM1CR	XDATA	0FF14H
PWM1HLD	XDATA	0FF15H
PWM2T1H	XDATA	0FF20H
PWM2T1L	XDATA	0FF21H
PWM2T2H	XDATA	0FF22H
PWM2T2L	XDATA	0FF23H
PWM2CR	XDATA	0FF24H
PWM2HLD	XDATA	0FF25H
PWM3T1H	XDATA	0FF30H
PWM3T1L	XDATA	0FF31H
PWM3T2H	XDATA	0FF32H
PWM3T2L	XDATA	0FF33H
PWM3CR	XDATA	0FF34H
PWM3HLD	XDATA	0FF35H
PWM4T1H	XDATA	0FF40H
PWM4T1L	XDATA	0FF41H
PWM4T2H	XDATA	0FF42H
PWM4T2L	XDATA	0FF43H
PWM4CR	XDATA	0FF44H
PWM4HLD	XDATA	0FF45H
PWM5T1H	XDATA	0FF50H
PWM5T1L	XDATA	0FF51H
PWM5T2H	XDATA	0FF52H
PWM5T2L	XDATA	0FF53H
PWM5CR	XDATA	0FF54H
PWM5HLD	XDATA	0FF55H
PWM6T1H	XDATA	0FF60H
PWM6T1L	XDATA	0FF61H

<i>PWM6T2H</i>	<i>XDATA</i>	<i>0FF62H</i>
<i>PWM6T2L</i>	<i>XDATA</i>	<i>0FF63H</i>
<i>PWM6CR</i>	<i>XDATA</i>	<i>0FF64H</i>
<i>PWM6HLD</i>	<i>XDATA</i>	<i>0FF65H</i>
<i>PWM7T1H</i>	<i>XDATA</i>	<i>0FF70H</i>
<i>PWM7T1L</i>	<i>XDATA</i>	<i>0FF71H</i>
<i>PWM7T2H</i>	<i>XDATA</i>	<i>0FF72H</i>
<i>PWM7T2L</i>	<i>XDATA</i>	<i>0FF73H</i>
<i>PWM7CR</i>	<i>XDATA</i>	<i>0FF74H</i>
<i>PWM7HLD</i>	<i>XDATA</i>	<i>0FF75H</i>
<i>DIR</i>	<i>BIT</i>	<i>20H.0</i>
<i>VALL</i>	<i>DATA</i>	<i>21H</i>
<i>VALH</i>	<i>DATA</i>	<i>22H</i>
	<i>ORG</i>	<i>0000H</i>
	<i>LJMP</i>	<i>MAIN</i>
	<i>ORG</i>	<i>00B3H</i>
	<i>LJMP</i>	<i>PWMISR</i>
	<i>ORG</i>	<i>0100H</i>
<i>PWMISR:</i>	 	
	<i>PUSH</i>	<i>ACC</i>
	<i>PUSH</i>	<i>PSW</i>
	<i>PUSH</i>	<i>DPL</i>
	<i>PUSH</i>	<i>DPH</i>
	<i>PUSH</i>	<i>P_SW2</i>
	<i>MOV</i>	<i>P_SW2,#80H</i>
	<i>MOV</i>	<i>A,PWMCFG</i>
	<i>JNB</i>	<i>ACC.7,ISREXIT</i>
	<i>ANL</i>	<i>PWMCFG,#NOT 80H</i>
	<i>JNB</i>	<i>;清中断标志</i>
<i>PWMUP:</i>	 	
	<i>MOV</i>	<i>A,VALL</i>
	<i>ADD</i>	<i>A,#1</i>
	<i>MOV</i>	<i>VALL,A</i>
	<i>MOV</i>	<i>A,VALH</i>
	<i>ADDC</i>	<i>A,#0</i>
	<i>MOV</i>	<i>VALH,A</i>
	<i>CJNE</i>	<i>A,#HIGH CYCLE,SETPWM</i>
	<i>MOV</i>	<i>A,VALL</i>
	<i>CJNE</i>	<i>A,#LOW CYCLE,SETPWM</i>
	<i>CLR</i>	<i>DIR</i>
	<i>JMP</i>	<i>SETPWM</i>
<i>PWMDN:</i>	 	
	<i>MOV</i>	<i>A,VALL</i>
	<i>ADD</i>	<i>A,#0FFH</i>
	<i>MOV</i>	<i>VALL,A</i>
	<i>MOV</i>	<i>A,VALH</i>
	<i>ADDC</i>	<i>A,#0FFH</i>
	<i>MOV</i>	<i>VALH,A</i>
	<i>JNZ</i>	<i>SETPWM</i>
	<i>MOV</i>	<i>A,VALL</i>
	<i>CJNE</i>	<i>A,#1,SETPWM</i>
	<i>SETB</i>	<i>DIR</i>
<i>SETPWM:</i>	 	
	<i>MOV</i>	<i>A,VALH</i>
	<i>MOV</i>	<i>DPTR,#PWM0T2H</i>

```

MOVX    @DPTR,A
MOV     A,VALL
MOV     DPTR,#PWM0T2L
MOVX    @DPTR,A

ISREXIT:
POP    P_SW2
POP    DPH
POP    DPL
POP    PSW
POP    ACC
RETI

MAIN:
MOV    SP,#3FH

SETB   DIR
MOV    VALH,#00H
MOV    VALL,#0IH

MOV    P_SW2,#80H
CLR    A
MOV    DPTR,#PWMCKS
MOVX   @DPTR,A          ; PWM 时钟为系统时钟
MOV    A,#HIGH CYCLE
MOV    DPTR,#PWMCH        ; 设置 PWM 周期
MOVX   @DPTR,A
MOV    A,#LOW CYCLE
MOV    DPTR,#PWMCL
MOVX   @DPTR,A
MOV    A,#00H
MOV    DPTR,#PWM0TIH
MOVX   @DPTR,A
MOV    A,#00H
MOV    DPTR,#PWM0T1L
MOVX   @DPTR,A
MOV    A,VALH
MOV    DPTR,#PWM0T2H
MOVX   @DPTR,A
MOV    A,VALL
MOV    DPTR,#PWM0T2L
MOVX   @DPTR,A
MOV    A,#80H
MOV    DPTR,#PWM0CR      ; 使能 PWM0 输出
MOVX   @DPTR,A
MOV    P_SW2,#00H

MOV    PWMCR,#0C0H        ; 启动 PWM 模块并使能 PWM 中断
SETB   EA
JMP    $

END

```

C 语言代码

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

```
// 测试工作频率为 11.0592MHz
```

```

#define CYCLE      0x1000

sfr P_SW2      = 0xba;
sfr PWMCFG     = 0xf1;
sfr PWMIF      = 0xf6;
sfr PWMFDCR    = 0xf7;
sfr PWMCR      = 0xfe;

#define PWMC      (*(unsigned int volatile xdata *)0xffff0)
#define PWMCKS    (*(unsigned char volatile xdata *)0xffff2)
#define TADCP     (*(unsigned int volatile xdata *)0xffff3)
#define PWM0T1    (*(unsigned int volatile xdata *)0xff00)
#define PWM0T2    (*(unsigned int volatile xdata *)0xff02)
#define PWM0CR    (*(unsigned char volatile xdata *)0xff04)
#define PWM0HLD   (*(unsigned char volatile xdata *)0xff05)
#define PWM1T1    (*(unsigned int volatile xdata *)0xff10)
#define PWM1T2    (*(unsigned int volatile xdata *)0xff12)
#define PWM1CR    (*(unsigned char volatile xdata *)0xff14)
#define PWM1HLD   (*(unsigned char volatile xdata *)0xff15)
#define PWM2T1    (*(unsigned int volatile xdata *)0xff20)
#define PWM2T2    (*(unsigned int volatile xdata *)0xff22)
#define PWM2CR    (*(unsigned char volatile xdata *)0xff24)
#define PWM2HLD   (*(unsigned char volatile xdata *)0xff25)
#define PWM3T1    (*(unsigned int volatile xdata *)0xff30)
#define PWM3T2    (*(unsigned int volatile xdata *)0xff32)
#define PWM3CR    (*(unsigned char volatile xdata *)0xff34)
#define PWM3HLD   (*(unsigned char volatile xdata *)0xff35)
#define PWM4T1    (*(unsigned int volatile xdata *)0xff40)
#define PWM4T2    (*(unsigned int volatile xdata *)0xff42)
#define PWM4CR    (*(unsigned char volatile xdata *)0xff44)
#define PWM4HLD   (*(unsigned char volatile xdata *)0xff45)
#define PWM5T1    (*(unsigned int volatile xdata *)0xff50)
#define PWM5T2    (*(unsigned int volatile xdata *)0xff52)
#define PWM5CR    (*(unsigned char volatile xdata *)0xff54)
#define PWM5HLD   (*(unsigned char volatile xdata *)0xff55)
#define PWM6T1    (*(unsigned int volatile xdata *)0xff60)
#define PWM6T2    (*(unsigned int volatile xdata *)0xff62)
#define PWM6CR    (*(unsigned char volatile xdata *)0xff64)
#define PWM6HLD   (*(unsigned char volatile xdata *)0xff65)
#define PWM7T1    (*(unsigned int volatile xdata *)0xff70)
#define PWM7T2    (*(unsigned int volatile xdata *)0xff72)
#define PWM7CR    (*(unsigned char volatile xdata *)0xff74)
#define PWM7HLD   (*(unsigned char volatile xdata *)0xff75)

void PWM_Isr() interrupt 22
{
    static bit dir = 1;
    static int val = 0;

    if (PWMCFG & 0x80)
    {
        PWMCFG &= ~0x80;           //清中断标志
        if (dir)
        {
            val++;
            if (val >= CYCLE) dir = 0;
        }
        else
        {

```

```
    val--;
    if (val <= 1) dir = 1;
}
.push_(P_SW2);
P_SW2 /= 0x80;
PWM0T2 = val;
.pop_(P_SW2);
}
}

void main()
{
    P_SW2 = 0x80;                                // PWM 时钟为系统时钟
    PWMCKS = 0x00;                               // 设置 PWM 周期为
    PWMC = CYCLE;
    PWM0T1= 0x0000;
    PWM0T2= 0x0001;                            // 使能 PWM0 输出
    PWM0CR= 0x80;
    P_SW2 = 0x00;                                // 启动 PWM 模块
    EA = 1;

    while (1);
}
```

19 同步串行外设接口SPI

STC8 系列单片机内部集成了一种高速串行通信接口——SPI 接口。SPI 是一种全双工的高速同步通信总线。STC8 系列集成的 SPI 接口提供了两种操作模式：主模式和从模式。

19.1 SPI相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
SPSTAT	SPI 状态寄存器	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx,xxxx
SPCTL	SPI 控制寄存器	CEH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR[1:0]	0000,0100	
SPDAT	SPI 数据寄存器	CFH									0000,0000

SPI 状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPSTAT	CDH	SPIF	WCOL	-	-	-	-	-	-

SPIF: SPI 中断标志位。

当发送/接收完成 1 字节的数据后，硬件自动将此位置 1，并向 CPU 提出中断请求。当 SSIG 位被设置为 0 时，由于 SS 管脚电平的变化而使得设备的主/从模式发生改变时，此标志位也会被硬件自动置 1，以标志设备模式发生变化。

注意：此标志位必须用户通过软件方式向此位写 1 进行清零。

WCOL: SPI 写冲突标志位。

当 SPI 在进行数据传输的过程中写 SPDAT 寄存器时，硬件将此位置 1。

注意：此标志位必须用户通过软件方式向此位写 1 进行清零。

SPI 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPCTL	CEH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR[1:0]	

SSIG: SS 引脚功能控制位

0: SS 引脚确定器件是主机还是从机

1: 忽略 SS 引脚功能，使用 MSTR 确定器件是主机还是从机

SPEN: SPI 使能控制位

0: 关闭 SPI 功能

1: 使能 SPI 功能

DORD: SPI 数据位发送/接收的顺序

0: 先发送/接收数据的高位 (MSB)

1: 先发送/接收数据的低位 (LSB)

MSTR: 器件主/从模式选择位

设置主机模式:

若 SSIG=0，则 SS 管脚必须为高电平且设置 MSTR 为 1

若 SSIG=1，则只需要设置 MSTR 为 1 (忽略 SS 管脚的电平)

设置从机模式:

若 SSIG=0，则 SS 管脚必须为低电平 (与 MSTR 位无关)

若 SSIG=1，则只需要设置 MSTR 为 0 (忽略 SS 管脚的电平)

CPOL: SPI 时钟极性控制

0: SCLK 空闲时为低电平, SCLK 的前时钟沿为上升沿, 后时钟沿为下降沿

1: SCLK 空闲时为高电平, SCLK 的前时钟沿为下降沿, 后时钟沿为上升沿

CPHA: SPI 时钟相位控制

0: 数据 SS 管脚为低电平驱动第一位数据并在 SCLK 的后时钟沿改变数据, 前时钟沿采样数据(必须 SSIG=0)

1: 数据在 SCLK 的前时钟沿驱动, 后时钟沿采样

SPR[1:0]: SPI 时钟频率选择

SPR[1:0]	SCLK 频率
00	SYScclk/4
01	SYScclk/8
10	SYScclk/16
11	SYScclk/32

SPI 数据寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPDAT	CFH								

SPI 发送/接收数据缓冲器。

注意事项:

1. **SPI 的 SCLK 和 MOSI 如果设置为开漏输出并且 IO 输出高, 则 SCLK 和 MOSI 没有输出信号, 但将这两个口输出低电平, 则能正常输出。并且, 这两个信号是推挽输出, 与 IO 设置无关。**

19.2 SPI通信方式

SPI 的通信方式通常有 3 种：单主单从（一个主机设备连接一个从机设备）、互为主从（两个设备连接，设备和互为主机和从机）、单主多从（一个主机设备连接多个从机设备）

19.2.1 单主单从

两个设备相连，其中一个设备固定作为主机，另外一个固定作为从机。

主机设置：SSIG 设置为 1，MSTR 设置为 1，固定为主机模式。主机可以使用任意端口连接从机的 SS 管脚，拉低从机的 SS 脚即可使能从机

从机设置：SSIG 设置为 0，SS 管脚作为从机的片选信号。

单主单从连接配置图如下所示：



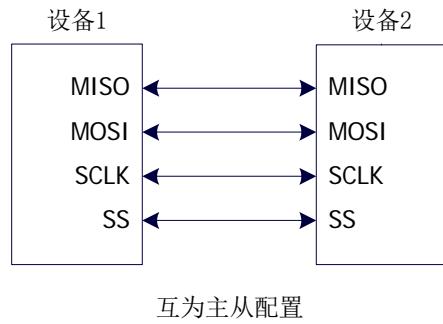
19.2.2 互为主从

两个设备相连，主机和从机不固定。

设置方法 1：两个设备初始化时都设置为 SSIG 设置为 0，MSTR 设置为 1，且将 SS 脚设置为双向口模式输出高电平。此时两个设备都是忽略 SS 的主机模式。当其中一个设备需要启动传输时，可将自己的 SS 脚设置为输出模式并输出低电平，拉低对方的 SS 脚，这样另一个设备就被强行设置为从机模式了。

设置方法 2：两个设备初始化时都将自己设置成忽略 SS 的从机模式，即将 SSIG 设置为 1，MSTR 设置为 0。当其中一个设备需要启动传输时，先检测 SS 管脚的电平，如果时候高电平，就将自己设置成忽略 SS 的主模式，即可进行数据传输了。

互为主从连接配置图如下所示：



互为主从配置

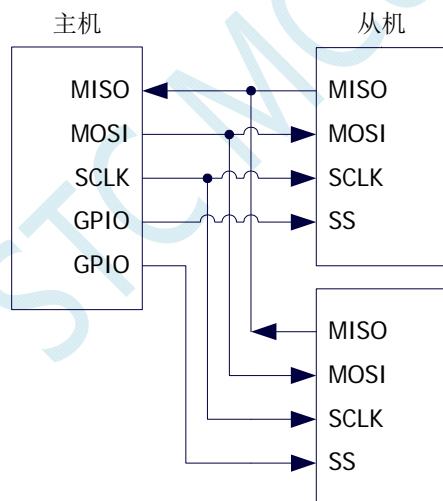
19.2.3 单主多从

多个设备相连，其中一个设备固定作为主机，其他设备固定作为从机。

主机设置：SSIG 设置为 1，MSTR 设置为 1，固定为主机模式。主机可以使用任意端口分别连接各个从机的 SS 管脚，拉低其中一个从机的 SS 脚即可使能相应的从机设备。

从机设置：SSIG 设置为 0，SS 管脚作为从机的片选信号。

单主多从连接配置图如下所示：



单主多从配置

19.3 配置SPI

控制位			通信端口				说明
SPEN	SSIG	MSTR	SS	MISO	MOSI	SCLK	
0	x	x	x	输入	输入	输入	关闭 SPI 功能, SS/MOSI/MISO/SCLK 均为普通 IO
1	0	0	0	输出	输入	输入	从机模式, 且被选中
1	0	0	1	高阻	输入	输入	从机模式, 但未被选中
1	0	1→0	0	输出	输入	输入	从机模式, 不忽略 SS 且 MSTR 为 1 的主机模式, 当 SS 管脚被拉低时, MSTR 将被硬件自动清零, 工作模式将被被动设置为从机模式
1	0	1	1	输入	高阻	高阻	主机模式, 空闲状态
				输出	输出	输出	主机模式, 激活状态
1	1	0	x	输出	输入	输入	从机模式
1	1	1	x	输入	输出	输出	主机模式

从机模式的注意事项:

当 CPHA=0 时, SSIG 必须为 0 (即不能忽略 SS 脚)。在每次串行字节开始还发送前 SS 脚必须拉低, 并且在串行字节发送完后须重新设置为高电平。SS 管脚为低电平时不能对 SPDAT 寄存器执行写操作, 否则将导致一个写冲突错误。CPHA=0 且 SSIG=1 时的操作未定义。

当 CPHA=1 时, SSIG 可以置 1 (即可以忽略脚)。如果 SSIG=0, SS 脚可在连续传输之间保持低有效 (即一直固定为低电平)。这种方式适用于固定单主单从的系统。

主机模式的注意事项:

在 SPI 中, 传输总是由主机启动的。如果 SPI 使能 (SPEN=1) 并选择作为主机时, 主机对 SPI 数据寄存器 SPDAT 的写操作将启动 SPI 时钟发生器和数据的传输。在数据写入 SPDAT 之后的半个到一个 SPI 位时间后, 数据将出现在 MOSI 脚。写入主机 SPDAT 寄存器的数据从 MOSI 脚移出发送到从机的 MOSI 脚。同时从机 SPDAT 寄存器的数据从 MISO 脚移出发送到主机的 MISO 脚。

传输完一个字节后, SPI 时钟发生器停止, 传输完成标志 (SPIF) 置位, 如果 SPI 中断使能则会产生一个 SPI 中断。主机和从机 CPU 的两个移位寄存器可以看作是一个 16 位循环移位寄存器。当数据从主机移位传送到从机的同时, 数据也以相反的方向移入。这意味着在一个移位周期中, 主机和从机的数据相互交换。

通过 SS 改变模式

如果 SPEN=1, SSIG=0 且 MSTR=1, SPI 使能为主机模式, 并将 SS 脚可配置为输入模式化或准双向模式。这种情况下, 另外一个主机可将该脚驱动为低电平, 从而将该器件选择为 SPI 从机并向其发送数据。为了避免争夺总线, SPI 系统将该从机的 MSTR 清零, MOSI 和 SCLK 强制变为输入模式, 而 MISO 则变为输出模式, 同时 SPSTAT 的 SPIF 标志位置 1。

用户软件必须一直对 MSTR 位进行检测, 如果该位被一个从机选择动作而被动清零, 而用户想继续将 SPI 作为主机, 则必须重新设置 MSTR 位, 否则将一直处于从机模式。

写冲突

SPI 在发送时为单缓冲, 在接收时为双缓冲。这样在前一次发送尚未完成之前, 不能将新的数据写

入移位寄存器。当发送过程中对数据寄存器 SPDAT 进行写操作时, WCOL 位将被置 1 以指示发生数据写冲突错误。在这种情况下, 当前发送的数据继续发送, 而新写入的数据将丢失。

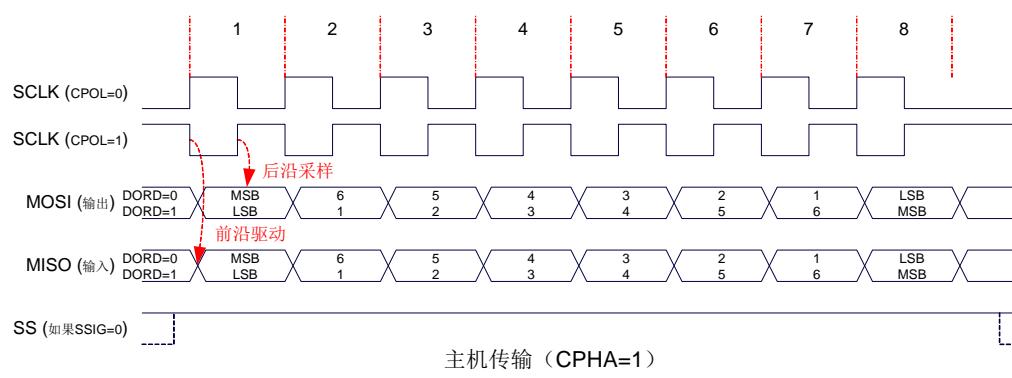
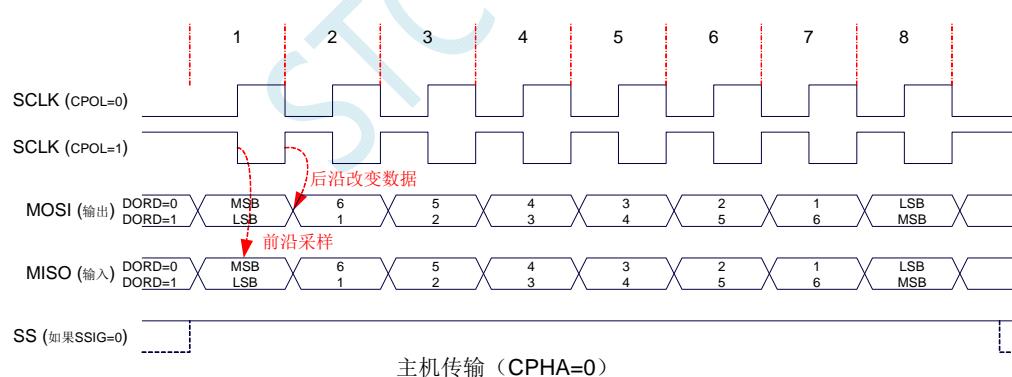
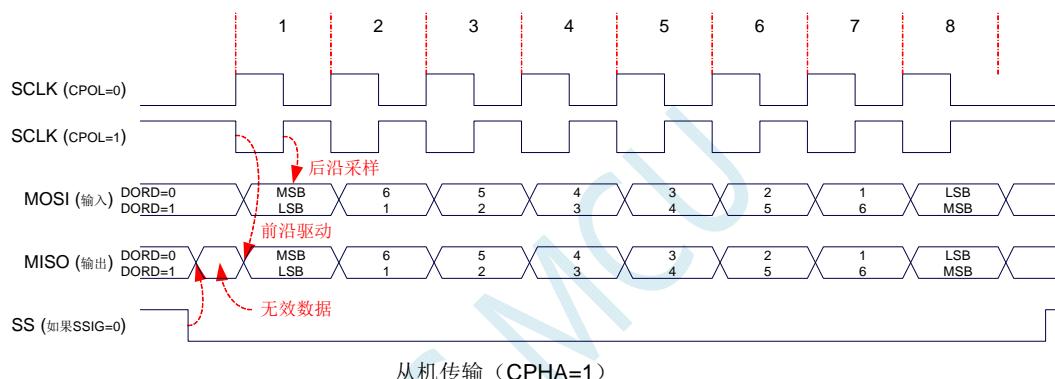
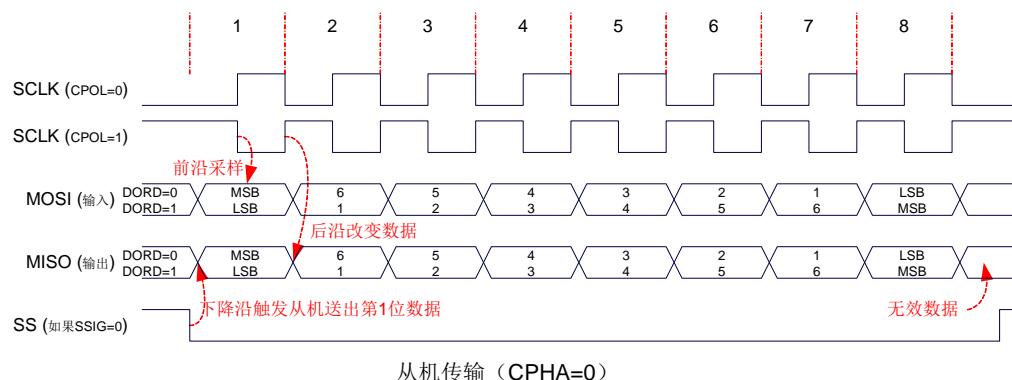
当对主机或从机进行写冲突检测时, 主机发生写冲突的情况是很罕见的, 因为主机拥有数据传输的完全控制权。但从机有可能发生写冲突, 因为当主机启动传输时, 从机无法进行控制。

接收数据时, 接收到的数据传送到一个并行读数据缓冲区, 这样将释放移位寄存器以进行下一个数据的接收。但必须在下个字符完全移入之前从数据寄存器中读出接收到的数据, 否则, 前一个接收数据将丢失。

WCOL 可通过软件向其写入“1”清零。

19.4 数据模式

SPI 的时钟相位控制位 CPHA 可以让用户设定数据采样和改变时的时钟沿。时钟极性位 CPOL 可以让用户设定期钟极性。下面图例显示了不同时钟相位、极性设置下 SPI 通讯时序。



19.5 范例程序

19.5.1 SPI单主单从系统主机程序（中断方式）

汇编代码

<i>SPSTAT</i>	<i>DATA</i>	<i>0CDH</i>	
<i>SPCTL</i>	<i>DATA</i>	<i>0CEH</i>	
<i>SPDAT</i>	<i>DATA</i>	<i>0CFH</i>	
<i>IE2</i>	<i>DATA</i>	<i>0AFH</i>	
<i>ESPI</i>	<i>EQU</i>	<i>02H</i>	
<i>BUSY</i>	<i>BIT</i>	<i>20H.0</i>	
<i>SS</i>	<i>BIT</i>	<i>P1.0</i>	
<i>LED</i>	<i>BIT</i>	<i>P1.1</i>	
	<i>ORG</i>	<i>0000H</i>	
	<i>LJMP</i>	<i>MAIN</i>	
	<i>ORG</i>	<i>004BH</i>	
	<i>LJMP</i>	<i>SPIISR</i>	
	<i>ORG</i>	<i>0100H</i>	
<i>SPIISR:</i>	<i>MOV</i>	<i>SPSTAT,#0C0H</i>	;清中断标志
	<i>SETB</i>	<i>SS</i>	;拉高从机的SS 管脚
	<i>CLR</i>	<i>BUSY</i>	
	<i>CPL</i>	<i>LED</i>	
	<i>RETI</i>		
<i>MAIN:</i>	<i>MOV</i>	<i>SP,#3FH</i>	
	<i>SETB</i>	<i>LED</i>	
	<i>SETB</i>	<i>SS</i>	
	<i>CLR</i>	<i>BUSY</i>	
	<i>MOV</i>	<i>SPCTL,#50H</i>	;使能SPI 主机模式
	<i>MOV</i>	<i>SPSTAT,#0C0H</i>	;清中断标志
	<i>MOV</i>	<i>IE2,#ESPI</i>	;使能SPI 中断
	<i>SETB</i>	<i>EA</i>	
<i>LOOP:</i>	<i>JB</i>	<i>BUSY,\$</i>	
	<i>SETB</i>	<i>BUSY</i>	
	<i>CLR</i>	<i>SS</i>	;拉低从机 SS 管脚
	<i>MOV</i>	<i>SPDAT,#5AH</i>	;发送测试数据
	<i>JMP</i>	<i>LOOP</i>	
	<i>END</i>		

C 语言代码

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

<i>sfr</i>	<i>SPSTAT</i>	=	<i>0xcd;</i>	
<i>sfr</i>	<i>SPCTL</i>	=	<i>0xce;</i>	
<i>sfr</i>	<i>SPDAT</i>	=	<i>0xcf;</i>	

```

sfr     IE2      = 0xaf;
#define  ESPI    0x02

sbit    SS       = P1^0;
sbit    LED      = P1^1;

bit     busy;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;           //清中断标志
    SS = 1;                  //拉高从机的SS 管脚
    busy = 0;
    LED = !LED;              //测试端口
}

void main()
{
    LED = 1;
    SS = 1;
    busy = 0;

    SPCTL = 0x50;            //使能SPI 主机模式
    SPSTAT = 0xc0;            //清中断标志
    IE2 = ESPI;               //使能SPI 中断
    EA = 1;

    while (1)
    {
        while (busy);
        busy = 1;
        SS = 0;                //拉低从机SS 管脚
        SPDAT = 0x5a;           //发送测试数据
    }
}

```

19.5.2 SPI单主单从系统从机程序（中断方式）

汇编代码

SPSTAT	DATA	0CDH
SPCTL	DATA	0CEH
SPDAT	DATA	0CFH
IE2	DATA	0AFH
ESPI	EQU	02H
LED	BIT	P1.1
	ORG	0000H
	LJMP	MAIN
	ORG	004BH
	LJMP	SPIISR
	ORG	0100H
SPIISR:		
	MOV	SPSTAT,#0C0H
	MOV	SPDAT,SPDAT ;清中断标志 ;将接收到的数据回传给主机
	CPL	LED
	RETI	

MAIN:

```

MOV      SP,#3FH
MOV      SPCTL,#40H          ;使能SPI从机模式
MOV      SPSTAT,#0C0H        ;清中断标志
MOV      IE2,#ESPI           ;使能SPI中断
SETB    EA
JMP      $

```

END**C 语言代码**

```

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

sfr    SPSTAT      = 0xcd;
sfr    SPCTL       = 0xce;
sfr    SPDAT       = 0xcf;
sfr    IE2         = 0xaf;
#define ESPI         0x02

sbit   LED          = P1^1;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;           //清中断标志
    SPDAT = SPDAT;          //将接收到的数据回传给主机
    LED = !LED;              //测试端口
}

void main()
{
    SPCTL = 0x40;           //使能SPI从机模式
    SPSTAT = 0xc0;           //清中断标志
    IE2 = ESPI;              //使能SPI中断
    EA = 1;

    while (1);
}

```

19.5.3 SPI单主单从系统主机程序（查询方式）**汇编代码**

SPSTAT	DATA	0CDH
SPCTL	DATA	0CEH
SPDAT	DATA	0CFH
IE2	DATA	0AFH
ESPI	EQU	02H
SS	BIT	P1.0
LED	BIT	P1.1
ORG	0000H	
LJMP	MAIN	

	ORG	0100H	
MAIN:	MOV	SP,#3FH	
	SETB	LED	
	SETB	SS	
	MOV	SPCTL,#50H	;使能 SPI 主机模式
	MOV	SPSTAT,#0C0H	;清中断标志
LOOP:	CLR	SS	;拉低从机 SS 管脚
	MOV	SPDAT,#5AH	;发送测试数据
	MOV	A,SPSTAT	;查询完成标志
	JNB	ACC.7,\$-2	
	MOV	SPSTAT,#0C0H	;清中断标志
	SETB	SS	
	CPL	LED	
	JMP	LOOP	
	END		

C 语言代码

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

sfr SPSTAT      = 0xcd;
sfr SPCTL       = 0xce;
sfr SPDAT       = 0xcf;
sfr IE2          = 0xaf;
#define ESPI        0x02

sbit SS          = P1^0;
sbit LED         = P1^1;

void main()
{
    LED = 1;
    SS = 1;

    SPCTL = 0x50;           //使能 SPI 主机模式
    SPSTAT = 0xc0;           //清中断标志

    while (1)
    {
        SS = 0;                //拉低从机 SS 管脚
        SPDAT = 0x5a;           //发送测试数据
        while (!(SPSTAT & 0x80)); //查询完成标志
        SPSTAT = 0xc0;           //清中断标志
        SS = 1;                 //拉高从机的 SS 管脚
        LED = !LED;              //测试端口
    }
}
```

19.5.4 SPI单主单从系统从机程序（查询方式）

汇编代码

<i>SPSTAT</i>	<i>DATA</i>	<i>0CDH</i>	
<i>SPCTL</i>	<i>DATA</i>	<i>0CEH</i>	
<i>SPDAT</i>	<i>DATA</i>	<i>0CFH</i>	
<i>IE2</i>	<i>DATA</i>	<i>0AFH</i>	
<i>ESPI</i>	<i>EQU</i>	<i>02H</i>	
<i>LED</i>	<i>BIT</i>	<i>P1.1</i>	
	<i>ORG</i>	<i>0000H</i>	
	<i>LJMP</i>	<i>MAIN</i>	
	<i>ORG</i>	<i>0100H</i>	
<i>MAIN:</i>	<i>MOV</i>	<i>SP,#3FH</i>	
	<i>MOV</i>	<i>SPCTL,#40H</i>	;使能SPI从机模式
	<i>MOV</i>	<i>SPSTAT,#0C0H</i>	;清中断标志
<i>LOOP:</i>	<i>MOV</i>	<i>A,SPSTAT</i>	;查询完成标志
	<i>JNB</i>	<i>ACC.7,\$-2</i>	
	<i>MOV</i>	<i>SPSTAT,#0C0H</i>	;清中断标志
	<i>MOV</i>	<i>SPDAT,SPDAT</i>	;将接收到的数据回传给主机
	<i>CPL</i>	<i>LED</i>	
	<i>JMP</i>	<i>LOOP</i>	
	<i>END</i>		

C 语言代码

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

sfr SPSTAT = 0xcd;
sfr SPCTL = 0xce;
sfr SPDAT = 0xcf;
sfr IE2 = 0xaf;
#define ESPI 0x02

sbit LED = P1^1;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0; //清中断标志
}

void main()
{
    SPCTL = 0x40; //使能SPI从机模式
    SPSTAT = 0xc0; //清中断标志

    while (1)
    {
        while (!(SPSTAT & 0x80)); //查询完成标志
        SPSTAT = 0xc0; //清中断标志
    }
}
```

```

SPDAT = SPDAT;           //将接收到的数据回传给主机
LED = !LED;              //测试端口
}
}

```

19.5.5 SPI互为主从系统程序（中断方式）

汇编代码

SPSTAT	DATA	0CDH	
SPCTL	DATA	0CEH	
SPDAT	DATA	0CFH	
IE2	DATA	0AFH	
ESPI	EQU	02H	
SS	BIT	P1.0	
LED	BIT	P1.1	
KEY	BIT	P0.0	
	ORG	0000H	
	LJMP	MAIN	
	ORG	004BH	
	LJMP	SPIISR	
	ORG	0100H	
SPIISR:	PUSH	ACC	
	MOV	SPSTAT,#0C0H	;清中断标志
	MOV	A,SPCTL	
	JB	ACC.4,MASTER	
SLAVE:	MOV	SPDAT,SPDAT	;将接收到的数据回传给主机
	JMP	ISREXIT	
MASTER:	SETB	SS	;拉高从机的 SS 管脚
	MOV	SPCTL,#40H	;重新设置为从机待机
ISREXIT:	CPL	LED	
	POP	ACC	
	RETI		
MAIN:	MOV	SP,#3FH	
	SETB	SS	
	SETB	LED	
	SETB	KEY	
	MOV	SPCTL,#40H	;使能 SPI 从机模式进行待机
	MOV	SPSTAT,#0C0H	;清中断标志
	MOV	IE2,#ESPI	;使能 SPI 中断
	SETB	EA	
LOOP:	JB	KEY,LOOP	;等待按键触发
	MOV	SPCTL,#50H	;使能 SPI 主机模式
	CLR	SS	;拉低从机 SS 管脚
	MOV	SPDAT,#5AH	;发送测试数据
	JNB	KEY,\$;等待按键释放

JMP**LOOP****END****C 语言代码**

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

sfr SPSTAT      = 0xcd;
sfr SPCTL       = 0xce;
sfr SPDAT       = 0xcf;
sfr IE2          = 0xaf;
#define ESPI        0x02

sbit SS          = PI^0;
sbit LED         = PI^1;
sbit KEY         = P0^0;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;           //清中断标志
    if(SPCTL & 0x10)
    {
        SS = 1;             //拉高从机的SS 管脚
        SPCTL = 0x40;        //重新设置为从机待机
    }
    else
    {
        SPDAT = SPDAT;     //从机模式
        //将接收到的数据回传给主机
    }
    LED = !LED;            //测试端口
}

void main()
{
    LED = 1;
    KEY = 1;
    SS = 1;

    SPCTL = 0x40;          //使能SPI 从机模式进行待机
    SPSTAT = 0xc0;          //清中断标志
    IE2 = ESPI;            //使能SPI 中断
    EA = 1;

    while (1)
    {
        if(!KEY)           //等待按键触发
        {
            SPCTL = 0x50;  //使能SPI 主机模式
            SS = 0;          //拉低从机SS 管脚
            SPDAT = 0x5a;    //发送测试数据
            while (!KEY);    //等待按键释放
        }
    }
}
```

19.5.6 SPI互为主从系统程序（查询方式）

汇编代码

<i>SPSTAT</i>	<i>DATA</i>	<i>0CDH</i>	
<i>SPCTL</i>	<i>DATA</i>	<i>0CEH</i>	
<i>SPDAT</i>	<i>DATA</i>	<i>0CFH</i>	
<i>IE2</i>	<i>DATA</i>	<i>0AFH</i>	
<i>ESPI</i>	<i>EQU</i>	<i>02H</i>	
<i>SS</i>	<i>BIT</i>	<i>P1.0</i>	
<i>LED</i>	<i>BIT</i>	<i>P1.1</i>	
<i>KEY</i>	<i>BIT</i>	<i>P0.0</i>	
	<i>ORG</i>	<i>0000H</i>	
	<i>LJMP</i>	<i>MAIN</i>	
	<i>ORG</i>	<i>0100H</i>	
<i>MAIN:</i>	<i>MOV</i>	<i>SP,#3FH</i>	
	<i>SETB</i>	<i>SS</i>	
	<i>SETB</i>	<i>LED</i>	
	<i>SETB</i>	<i>KEY</i>	
	<i>MOV</i>	<i>SPCTL,#40H</i>	;使能 SPI 从机模式进行待机
	<i>MOV</i>	<i>SPSTAT,#0C0H</i>	;清中断标志
<i>LOOP:</i>	<i>JB</i>	<i>KEY,SKIP</i>	;等待按键触发
	<i>MOV</i>	<i>SPCTL,#50H</i>	;使能 SPI 主机模式
	<i>CLR</i>	<i>SS</i>	;拉低从机 SS 管脚
	<i>MOV</i>	<i>SPDAT,#5AH</i>	;发送测试数据
	<i>JNB</i>	<i>KEY,\$</i>	;等待按键释放
<i>SKIP:</i>	<i>MOV</i>	<i>A,SPSTAT</i>	
	<i>JNB</i>	<i>ACC.7,LOOP</i>	
	<i>MOV</i>	<i>SPSTAT,#0C0H</i>	;清中断标志
	<i>MOV</i>	<i>A,SPCTL</i>	
	<i>JB</i>	<i>ACC.4,MASTER</i>	
<i>SLAVE:</i>	<i>MOV</i>	<i>SPDAT,SPDAT</i>	;将接收到的数据回传给主机
	<i>CPL</i>	<i>LED</i>	
	<i>JMP</i>	<i>LOOP</i>	
<i>MASTER:</i>	<i>SETB</i>	<i>SS</i>	;拉高从机的 SS 管脚
	<i>MOV</i>	<i>SPCTL,#40H</i>	;重新设置为从机待机
	<i>CPL</i>	<i>LED</i>	
	<i>JMP</i>	<i>LOOP</i>	
	<i>END</i>		

C 语言代码

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

```
sfr SPSTAT = 0xcd;
sfr SPCTL = 0xce;
```

```
sfr     SPDAT      = 0xcf;
sfr     IE2         = 0xaf;
#define  ESPI        0x02

sbit    SS          = P1^0;
sbit    LED         = P1^1;
sbit    KEY         = P0^0;

void main()
{
    LED = 1;
    KEY = 1;
    SS = 1;

    SPCTL = 0x40;           //使能SPI 从机模式进行待机
    SPSTAT = 0xc0;          //清中断标志

    while (1)
    {
        if (!KEY)           //等待按键触发
        {
            SPCTL = 0x50;   //使能SPI 主机模式
            SS = 0;          //拉低从机SS 管脚
            SPDAT = 0x5a;    //发送测试数据
            while (!KEY);    //等待按键释放
        }
        if (SPSTAT & 0x80)
        {
            SPSTAT = 0xc0;  //清中断标志
            if (SPCTL & 0x10)
            {
                SS = 1;       //主机模式
                SPCTL = 0x40; //重新设置为从机待机
            }
            else
            {
                SPDAT = SPDAT; //将接收到的数据回传给主机
            }
            LED = !LED;      //测试端口
        }
    }
}
```

20 I²C总线

STC8 系列的单片机内部集成了一个 I²C 串行总线控制器。I²C 是一种高速同步通讯总线，通讯使用 SCL（时钟线）和 SDA（数据线）两线进行同步通讯。对于 SCL 和 SDA 的端口分配，STC8 系列的单片机提供了切换模式，可将 SCL 和 SDA 切换到不同的 I/O 口上，以方便用户将一组 I²C 总线当作多组进行分时复用。

与标准 I²C 协议相比较，忽略了如下两种机制：

- 发送起始信号（START）后不进行仲裁
- 时钟信号（SCL）停留在低电平时不进行超时检测

STC8 系列的 I²C 总线提供了两种操作模式：主机模式（SCL 为输出口，发送同步时钟信号）和从机模式（SCL 为输入口，接收同步时钟信号）

20.1 I²C相关的寄存器

符号	描述	地址	位地址与符号								复位值		
			B7	B6	B5	B4	B3	B2	B1	B0			
I2CCFG	I ² C 配置寄存器	FE80H	ENI2C	MSSL	MSSPEED[6:1]				MSCMD[3:0]				0000,0000
I2CMSCR	I ² C 主机控制寄存器	FE81H	EMSI	-	-	-	MSACKI				MSACKO	0xxx,0000	
I2CMSST	I ² C 主机状态寄存器	FE82H	MSBUSY	MSIF	-	-	-	-	-	MSACKI	MSACKO	00xx,xx00	
I2CSLCR	I ² C 从机控制寄存器	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0		
I2CSLST	I ² C 从机状态寄存器	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO		0000,0000	
I2CSLADR	I ² C 从机地址寄存器	FE85H	SLADR[6:0]							MA		0000,0000	
I2CTXD	I ² C 数据发送寄存器	FE86H											0000,0000
I2CRXD	I ² C 数据接收寄存器	FE87H											0000,0000
I2CMSAUX	I ² C 主机辅助控制寄存器	FE88H	-	-	-	-	-	-	-	WDTA		xxxx,xxx0	

20.2 I²C 主机模式

I²C 配置寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CCFG	FE80H	ENI2C	MSSL						MSSPEED[6:1]

ENI2C: I²C 功能使能控制位

0: 禁止 I²C 功能

1: 允许 I²C 功能

MSSL: I²C 工作模式选择位

0: 从机模式

1: 主机模式

MSSPEED[6:1]: I²C 总线速度 (等待时钟数) 控制

MSSPEED[6:1]	对应的时钟数
0	1
1	3
2	5
...	...
x	2x+1
...	...
62	125
63	127

只有当 I²C 模块工作在主机模式时, MSSPEED 参数设置的等待参数才有效。此等待参数主要用于主机模式的以下几个信号:

T_{SSTA}: 起始信号的建立时间 (Setup Time of START)

T_{HSTA}: 起始信号的保持时间 (Hold Time of START)

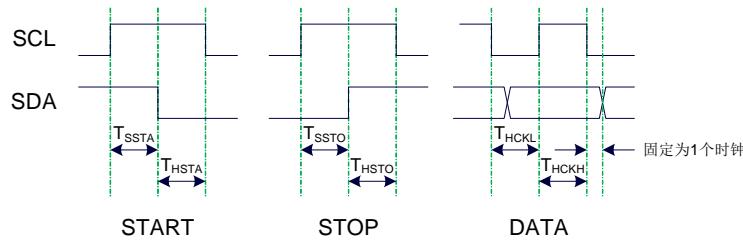
T_{SSTO}: 停止信号的建立时间 (Setup Time of STOP)

T_{HSTO}: 停止信号的保持时间 (Hold Time of STOP)

T_{HCKL}: 时钟信号的低电平保持时间 (Hold Time of SCL Low)

注意:

- 由于需要配合时钟同步机制, 对于时钟信号的高电平保持时间 (T_{HCKH}) 至少为时钟信号的低电平保持时间 (T_{HCKL}) 的 1 倍长, 而 T_{HCKH} 确切的长度取决于 SCL 端口的上拉速度。
- SDA 在 SCL 下降沿后的数据保持时间固定为 1 个时钟



I²C 主机控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSCR	FE81H	EMSI	-	-	-	-			MSCMD[2:0]

EMSI: 主机模式中断使能控制位

0: 关闭主机模式的中断

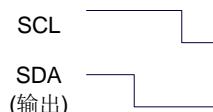
1: 允许主机模式的中断

MSCMD[3:0]: 主机命令

0000: 待机, 无动作。

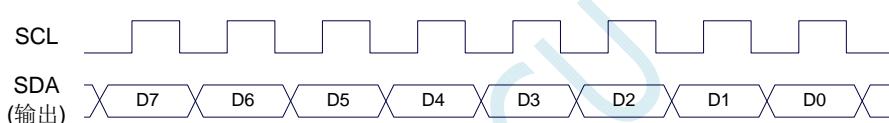
0001: 起始命令。

发送 START 信号。如果当前 I²C 控制器处于空闲状态, 即 MSBUSY (I2CMSST.7) 为 0 时, 写此命令会使控制器进入忙状态, 硬件自动将 MSBUSY 状态位置 1, 并开始发送 START 信号; 若当前 I²C 控制器处于忙状态, 写此命令可触发发送 START 信号。发送 START 信号的波形如下图所示:



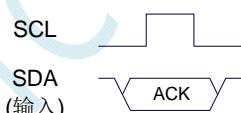
0010: 发送数据命令。

写此命令后, I²C 总线控制器会在 SCL 管脚上产生 8 个时钟, 并将 I2CTXD 寄存器里面数据按位送到 SDA 管脚上 (先发送高位数据)。发送数据的波形如下图所示:



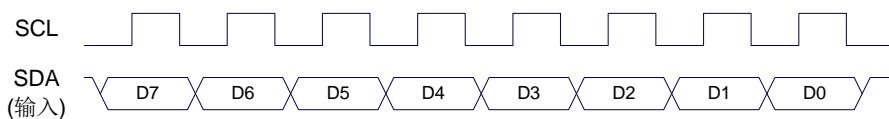
0011: 接收 ACK 命令。

写此命令后, I²C 总线控制器会在 SCL 管脚上产生 1 个时钟, 并将从 SDA 端口上读取的数据保存到 MSACKI (I2CMSST.1)。接收 ACK 的波形如下图所示:



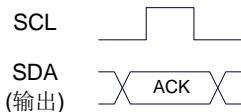
0100: 接收数据命令。

写此命令后, I²C 总线控制器会在 SCL 管脚上产生 8 个时钟, 并将从 SDA 端口上读取的数据依次左移到 I2CRXD 寄存器 (先接收高位数据)。接收数据的波形如下图所示:



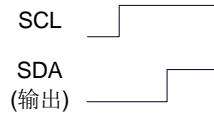
0101: 发送 ACK 命令。

写此命令后, I²C 总线控制器会在 SCL 管脚上产生 1 个时钟, 并将 MSACKO (I2CMSST.0) 中的数据发送到 SDA 端口。发送 ACK 的波形如下图所示:



0110: 停止命令。

发送 STOP 信号。写此命令后, I²C 总线控制器开始发送 STOP 信号。信号发送完成后, 硬件自动将 MSBUSY 状态位清零。STOP 信号的波形如下图所示:



0111: 保留。

1000: 保留。

注: 以下新增的扩展组合命令只对 STC8F2K64S4 系列的 C 版/D 版芯片、STC8A8K64S4A12 系列的 E 版/F 版芯片、STC8F2K64S2 系列的 C 版芯片/D 版、STC8A4K64S2A12 系列的 E 版/F 版芯片有效

1001: 起始命令+发送数据命令+接收 ACK 命令。

此命令为命令 0001、命令 0010、命令 0011 三个命令的组合，下此命令后控制器会依次执行这三个命令。

1010: 发送数据命令+接收 ACK 命令。

此命令为命令 0010、命令 0011 两个命令的组合，下此命令后控制器会依次执行这两个命令。

1011: 接收数据命令+发送 ACK(0)命令。

此命令为命令 0100、命令 0101 两个命令的组合，下此命令后控制器会依次执行这两个命令。

注意：此命令所返回的应答信号固定为 ACK (0)，不受 MSACKO 位的影响。

1100: 接收数据命令+发送 NAK(1)命令。

此命令为命令 0100、命令 0101 两个命令的组合，下此命令后控制器会依次执行这两个命令。

注意：此命令所返回的应答信号固定为 NAK (1)，不受 MSACKO 位的影响。

I²C 主机辅助控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSAUX	FE88H	-	-	-	-	-	-	-	WDTA

WDTA: 主机模式时 I²C 数据自动发送允许位

0: 禁止自动发送

1: 使能自动发送

若自动发送功能被使能，当 MCU 执行完成对 I2CTXD 数据寄存器的写操作后，I²C 控制器会自动触发“1010”命令，即自动发送数据并接收 ACK 信号。

I²C 主机状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSST	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO

MSBUSY: 主机模式时 I²C 控制器状态位（只读位）

0: 控制器处于空闲状态

1: 控制器处于忙碌状态

当 I²C 控制器处于主机模式时，在空闲状态下，发送完成 START 信号后，控制器便进入到忙碌状态，忙碌状态会一直维持到成功发送完成 STOP 信号，之后状态会再次恢复到空闲状态。

MSIF: 主机模式的中断请求位（中断标志位）。当处于主机模式的 I²C 控制器执行完成寄存器 I2CMSCR 中 MSCMD 命令后产生中断信号，硬件自动将此位 1，向 CPU 发请求中断，响应中断后 MSIF 位必须用软件清零。

MSACKI: 主机模式时，发送“0011”命令到 I2CMSCR 的 MSCMD 位后所接收到的 ACK 数据。

MSACKO: 主机模式时，准备将要发送出去的 ACK 信号。当发送“0101”命令到 I2CMSCR 的 MSCMD

位后，控制器会自动读取此位的数据当作 ACK 发送到 SDA。

STCMCU

20.3 I²C从机模式

I²C 从机控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLCR	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST

ESTAI: 从机模式时接收到 START 信号中断允许位

0: 禁止从机模式时接收到 START 信号时发生中断

1: 使能从机模式时接收到 START 信号时发生中断

ERXI: 从机模式时接收到 1 字节数据后中断允许位

0: 禁止从机模式时接收到数据后发生中断

1: 使能从机模式时接收到 1 字节数据后发生中断

ETXI: 从机模式时发送完成 1 字节数据后中断允许位

0: 禁止从机模式时发送完成数据后发生中断

1: 使能从机模式时发送完成 1 字节数据后发生中断

ESTOI: 从机模式时接收到 STOP 信号中断允许位

0: 禁止从机模式时接收到 STOP 信号时发生中断

1: 使能从机模式时接收到 STOP 信号时发生中断

SLRST: 复位从机模式

I²C 从机状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLST	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	-	SLACKI	SLACKO

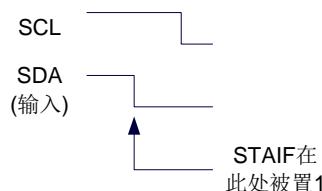
SLBUSY: 从机模式时 I²C 控制器状态位 (只读位)

0: 控制器处于空闲状态

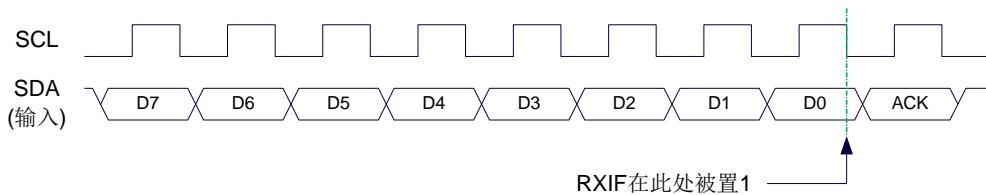
1: 控制器处于忙碌状态

当 I²C 控制器处于从机模式时, 在空闲状态下, 接收到主机发送 START 信号后, 控制器会继续检测之后的设备地址数据, 若设备地址与当前 I2CSLADR 寄存器中所设置的从机地址像匹配时, 控制器便进入到忙碌状态, 忙碌状态会一直维持到成功接收到主机发送 STOP 信号, 之后状态会再次恢复到空闲状态。

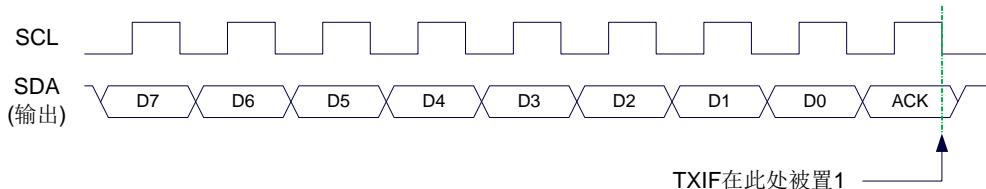
STAIF: 从机模式时接收到 START 信号后的中断请求位。从机模式的 I²C 控制器接收到 START 信号后, 硬件会自动将此位置 1, 并向 CPU 发请求中断, 响应中断后 STAIF 位必须用软件清零。STAIF 被置 1 的时间点如下图所示:



RXIF: 从机模式时接收到 1 字节的数据后的中断请求位。从机模式的 I²C 控制器接收到 1 字节的数据后, 在第 8 个时钟的下降沿时硬件会自动将此位置 1, 并向 CPU 发请求中断, 响应中断后 RXIF 位必须用软件清零。RXIF 被置 1 的时间点如下图所示:



TXIF: 从机模式时发送完成 1 字节的数据后的中断请求位。从机模式的 I²C 控制器发送完成 1 字节的数据并成功接收到 1 位 ACK 信号后，在第 9 个时钟的下降沿时硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 TXIF 位必须用软件清零。TXIF 被置 1 的时间点如下图所示：

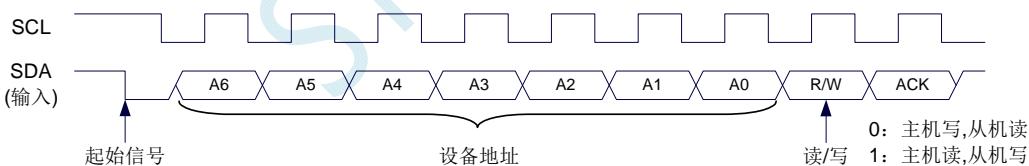


STOIF: 从机模式时接收到 STOP 信号后的中断请求位。从机模式的 I²C 控制器接收到 STOP 信号后，硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 STOIF 位必须用软件清零。STOIF 被置 1 的时间点如下图所示：



SLACKI: 从机模式时，接收到的 ACK 数据。

SLACKO: 从机模式时，准备将要发送出去的 ACK 信号。



I²C 从机地址寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLADR	FE85H								MA

SLADR[6:0]: 从机设备地址

当 I²C 控制器处于从机模式时，控制器在接收到 START 信号后，会继续检测接下来主机发送出的设备地址数据以及读/写信号。当主机发送出的设备地址与 SLADR[6:0]中所设置的从机设备地址相匹配时，控制器才会向 CPU 发出中断请求，请求 CPU 处理 I²C 事件；否则若设备地址不匹配，I²C 控制器继续继续监控，等待下一个起始信号，对下一个设备地址继续匹配。

MA: 从机设备地址匹配控制

- 0: 设备地址必须与 SLADR[6:0]继续匹配
- 1: 忽略 SLADR 中的设置，匹配所有的设备地址

I²C 数据寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CTXD	FE86H								
I2CRXD	FE87H								

I2CTXD 是 I²C 发送数据寄存器，存放将要发送的 I²C 数据

I2CRXD 是 I²C 接收数据寄存器，存放接收完成的 I²C 数据

20.4 范例程序

20.4.1 I²C主机模式访问AT24C256（中断方式）

汇编代码

<i>P_SW2</i>	<i>DATA</i>	<i>0BAH</i>
<i>I2CCFG</i>	<i>XDATA</i>	<i>0FE80H</i>
<i>I2CMSCR</i>	<i>XDATA</i>	<i>0FE81H</i>
<i>I2CMSST</i>	<i>XDATA</i>	<i>0FE82H</i>
<i>I2CSLCR</i>	<i>XDATA</i>	<i>0FE83H</i>
<i>I2CSLST</i>	<i>XDATA</i>	<i>0FE84H</i>
<i>I2CSLADR</i>	<i>XDATA</i>	<i>0FE85H</i>
<i>I2CTXD</i>	<i>XDATA</i>	<i>0FE86H</i>
<i>I2CRXD</i>	<i>XDATA</i>	<i>0FE87H</i>
<i>SDA</i>	<i>BIT</i>	<i>P1.4</i>
<i>SCL</i>	<i>BIT</i>	<i>P1.5</i>
<i>BUSY</i>	<i>BIT</i>	<i>20H.0</i>
	<i>ORG</i>	<i>0000H</i>
	<i>LJMP</i>	<i>MAIN</i>
	<i>ORG</i>	<i>00C3H</i>
	<i>LJMP</i>	<i>I2CISR</i>
	<i>ORG</i>	<i>0100H</i>
<i>I2CISR:</i>	<i>PUSH</i>	<i>ACC</i>
	<i>PUSH</i>	<i>DPL</i>
	<i>PUSH</i>	<i>DPH</i>
	<i>MOV</i>	<i>DPTR,#I2CMSST</i>
	<i>MOVX</i>	<i>A,@DPTR</i>
	<i>ANL</i>	<i>A,#NOT 40H</i>
	<i>MOV</i>	<i>DPTR,#I2CMSST</i>
	<i>MOVX</i>	<i>@DPTR,A</i>
	<i>CLR</i>	<i>BUSY</i>
		;清中断标志
	<i>POP</i>	<i>DPH</i>
	<i>POP</i>	<i>DPL</i>
	<i>POP</i>	<i>ACC</i>
	<i>RETI</i>	
<i>START:</i>	<i>SETB</i>	<i>BUSY</i>
	<i>MOV</i>	<i>A,#10000001B</i>
	<i>MOV</i>	<i>DPTR,#I2CMSCR</i>
	<i>MOVX</i>	<i>@DPTR,A</i>
	<i>JMP</i>	<i>WAIT</i>
		;发送 START 命令
<i>SENDDATA:</i>	<i>MOV</i>	<i>DPTR,#I2CTXD</i>
	<i>MOVX</i>	<i>@DPTR,A</i>
	<i>SETB</i>	<i>BUSY</i>
	<i>MOV</i>	<i>A,#10000010B</i>
	<i>MOV</i>	<i>DPTR,#I2CMSCR</i>
	<i>MOVX</i>	<i>@DPTR,A</i>
	<i>JMP</i>	<i>WAIT</i>
		;写数据到数据缓冲区
		;发送 SEND 命令

RECVACK:

SETB	BUSY	
MOV	A,#10000011B	;发送读ACK命令
MOV	DPTR,#I2CMSCR	
MOVX	@DPTR,A	
JMP	WAIT	

RECVDATA:

SETB	BUSY	
MOV	A,#10000100B	;发送RECV命令
MOV	DPTR,#I2CMSCR	
MOVX	@DPTR,A	
CALL	WAIT	
MOV	DPTR,#I2CRXD	;从数据缓冲区读取数据
MOVX	A,@DPTR	
RET		

SENDACK:

MOV	A,#00000000B	;设置ACK信号
MOV	DPTR,#I2CMSST	
MOVX	@DPTR,A	
SETB	BUSY	
MOV	A,#10000101B	;发送ACK命令
MOV	DPTR,#I2CMSCR	
MOVX	@DPTR,A	
JMP	WAIT	

SENDNAK:

MOV	A,#00000001B	;设置NAK信号
MOV	DPTR,#I2CMSST	
MOVX	@DPTR,A	
SETB	BUSY	
MOV	A,#10000101B	;发送ACK命令
MOV	DPTR,#I2CMSCR	
MOVX	@DPTR,A	
JMP	WAIT	

STOP:

SETB	BUSY	
MOV	A,#10000110B	;发送STOP命令
MOV	DPTR,#I2CMSCR	
MOVX	@DPTR,A	
JMP	WAIT	

WAIT:

JB	BUSY,\$;等待命令发送完成
RET		

DELAY:

MOV	R0,#0	
MOV	R1,#0	

DELAY1:

NOP		
DJNZ	R1,DELAY1	
DJNZ	R0,DELAY1	
RET		

MAIN:

MOV	SP,#3FH	
MOV	P_SW2,#80H	

```

MOV      A,#11100000B          ;设置 I2C 模块为主机模式
MOV      DPTR,#I2CCFG
MOVX    @DPTR,A
MOV      A,#00000000B
MOV      DPTR,#I2CMSST
MOVX    @DPTR,A
SETB    EA

CALL    START                 ;发送起始命令
MOV      A,#0A0H
CALL    SENDDATA              ;发送设备地址+写命令
CALL    RECVACK
MOV      A,#00H                ;发送存储地址高字节
CALL    SENDDATA
CALL    RECVACK
MOV      A,#00H                ;发送存储地址低字节
CALL    SENDDATA
CALL    RECVACK
MOV      A,#12H                ;写测试数据 1
CALL    SENDDATA
CALL    RECVACK
MOV      A,#78H                ;写测试数据 2
CALL    SENDDATA
CALL    RECVACK
CALL    STOP                  ;发送停止命令

CALL    DELAY                 ;等待设备写数据

CALL    START                 ;发送起始命令
MOV      A,#0A0H
CALL    SENDDATA              ;发送设备地址+写命令
CALL    RECVACK
MOV      A,#00H                ;发送存储地址高字节
CALL    SENDDATA
CALL    RECVACK
MOV      A,#00H                ;发送存储地址低字节
CALL    SENDDATA
CALL    RECVACK
CALL    START                 ;发送起始命令
MOV      A,#0AIH                ;发送设备地址+读命令
CALL    SENDDATA
CALL    RECVACK
CALL    RECVDATA              ;读取数据 1
MOV      P0,A
CALL    SENDACK
CALL    RECVDATA              ;读取数据 2
MOV      P2,A
CALL    SENDNAK
CALL    STOP                  ;发送停止命令

JMP    $

```

END**C 语言代码**

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

```

sfr      P_SW2      = 0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR     (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST      (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR      (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST      (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR     (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD       (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD       (*(unsigned char volatile xdata *)0xfe87)

sbit     SDA        = P1^4;
sbit     SCL        = P1^5;

bit      busy;

void I2C_Isr() interrupt 24
{
    _push_(P_SW2);
    P_SW2 /= 0x80;
    if(I2CMSST & 0x40)
    {
        I2CMSST &= ~0x40;           //清中断标志
        busy = 0;
    }
    _pop_(P_SW2);
}

void Start()
{
    busy = 1;
    I2CMSCR = 0x81;             //发送 START 命令
    while (busy);
}

void SendData(char dat)
{
    I2CTXD = dat;              //写数据到数据缓冲区
    busy = 1;
    I2CMSCR = 0x82;             //发送 SEND 命令
    while (busy);
}

void RecvACK()
{
    busy = 1;
    I2CMSCR = 0x83;             //发送读 ACK 命令
    while (busy);
}

char RecvData()
{
    busy = 1;
    I2CMSCR = 0x84;             //发送 RECV 命令
    while (busy);
    return I2CRXD;
}

void SendACK()

```

```

{
    I2CMSST = 0x00; //设置ACK 信号
    busy = 1;
    I2CMSCR = 0x85; //发送ACK 命令
    while (busy);
}

void SendNAK()
{
    I2CMSST = 0x01; //设置NAK 信号
    busy = 1;
    I2CMSCR = 0x85; //发送ACK 命令
    while (busy);
}

void Stop()
{
    busy = 1;
    I2CMSCR = 0x86; //发送STOP 命令
    while (busy);
}

void Delay()
{
    int i;

    for (i=0; i<3000; i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
}

void main()
{
    P_SW2 = 0x80;

    I2CCFG = 0xe0; //使能I2C 主机模式
    I2CMSST = 0x00;
    EA = 1;

    Start(); //发送起始命令
    SendData(0xa0); //发送设备地址+写命令
    RecvACK();
    SendData(0x00); //发送存储地址高字节
    RecvACK();
    SendData(0x00); //发送存储地址低字节
    RecvACK();
    SendData(0x12); //写测试数据1
    RecvACK();
    SendData(0x78); //写测试数据2
    RecvACK();
    Stop(); //发送停止命令

    Delay(); //等待设备写数据

    Start(); //发送起始命令
}

```

```

SendData(0xa0);           //发送设备地址+写命令
RecvACK();
SendData(0x00);           //发送存储地址高字节
RecvACK();
SendData(0x00);           //发送存储地址低字节
RecvACK();
Start();                  //发送起始命令
SendData(0xa1);           //发送设备地址+读命令
RecvACK();
P0 = RecvData();          //读取数据 1
SendACK();
P2 = RecvData();          //读取数据 2
SendNAK();
Stop();                   //发送停止命令

P_SW2 = 0x00;

while (1);
}

```

20.4.2 I²C主机模式访问AT24C256（查询方式）

汇编代码

<i>P_SW2</i>	<i>DATA</i>	<i>0BAH</i>
<i>I2CCFG</i>	<i>XDATA</i>	<i>0FE80H</i>
<i>I2CMSCR</i>	<i>XDATA</i>	<i>0FE81H</i>
<i>I2CMSST</i>	<i>XDATA</i>	<i>0FE82H</i>
<i>I2CSLCR</i>	<i>XDATA</i>	<i>0FE83H</i>
<i>I2CSLST</i>	<i>XDATA</i>	<i>0FE84H</i>
<i>I2CSLADR</i>	<i>XDATA</i>	<i>0FE85H</i>
<i>I2CTXD</i>	<i>XDATA</i>	<i>0FE86H</i>
<i>I2CRXD</i>	<i>XDATA</i>	<i>0FE87H</i>
<i>SDA</i>	<i>BIT</i>	<i>P1.4</i>
<i>SCL</i>	<i>BIT</i>	<i>P1.5</i>
	<i>ORG</i>	<i>0000H</i>
	<i>LJMP</i>	<i>MAIN</i>
	<i>ORG</i>	<i>0100H</i>
<i>START:</i>		
	<i>MOV</i>	<i>A,#00000001B</i> ;发送 START 命令
	<i>MOV</i>	<i>DPTR,#I2CMSCR</i>
	<i>MOVX</i>	<i>@DPTR,A</i>
	<i>JMP</i>	<i>WAIT</i>
<i>SENDDATA:</i>		
	<i>MOV</i>	<i>DPTR,#I2CTXD</i> ;写数据到数据缓冲区
	<i>MOVX</i>	<i>@DPTR,A</i>
	<i>MOV</i>	<i>A,#00000010B</i> ;发送 SEND 命令
	<i>MOV</i>	<i>DPTR,#I2CMSCR</i>
	<i>MOVX</i>	<i>@DPTR,A</i>
	<i>JMP</i>	<i>WAIT</i>
<i>RECVACK:</i>		
	<i>MOV</i>	<i>A,#00000011B</i> ;发送读 ACK 命令
	<i>MOV</i>	<i>DPTR,#I2CMSCR</i>
	<i>MOVX</i>	<i>@DPTR,A</i>
	<i>JMP</i>	<i>WAIT</i>

RECVDATA:

```

MOV      A,#00000100B          ;发送 RECV 命令
MOV      DPTR,#I2CMSCR
MOVX    @DPTR,A
CALL    WAIT
MOV      DPTR,#I2CRXD        ;从数据缓冲区读取数据
MOVX    A,@DPTR
RET

```

SENDACK:

```

MOV      A,#00000000B          ;设置 ACK 信号
MOV      DPTR,#I2CMSST
MOVX    @DPTR,A
MOV      A,#00000101B          ;发送 ACK 命令
MOV      DPTR,#I2CMSCR
MOVX    @DPTR,A
JMP    WAIT

```

SENDNAK:

```

MOV      A,#00000001B          ;设置 NAK 信号
MOV      DPTR,#I2CMSST
MOVX    @DPTR,A
MOV      A,#00000101B          ;发送 ACK 命令
MOV      DPTR,#I2CMSCR
MOVX    @DPTR,A
JMP    WAIT

```

STOP:

```

MOV      A,#00000110B          ;发送 STOP 命令
MOV      DPTR,#I2CMSCR
MOVX    @DPTR,A
JMP    WAIT

```

WAIT:

```

MOV      DPTR,#I2CMSST        ;清中断标志
MOVX    A,@DPTR
JNB    ACC.6,WAIT
ANL    A,#NOT 40H
MOVX    @DPTR,A
RET

```

DELAY:

```

MOV      R0,#0
MOV      R1,#0

```

DELAYI:

```

NOP
NOP
NOP
NOP
DJNZ    R1,DELAYI
DJNZ    R0,DELAYI
RET

```

MAIN:

```

MOV      SP,#3FH
MOV      P_SW2,#80H

MOV      A,#1110000B          ;设置 I2C 模块为主机模式
MOV      DPTR,#I2CCFG
MOVX    @DPTR,A
MOV      A,#00000000B
MOV      DPTR,#I2CMSST
MOVX    @DPTR,A

```

```

CALL      START          ;发送起始命令
MOV      A,#0A0H
CALL      SENDDATA      ;发送设备地址+写命令
CALL      RECVACK
MOV      A,#000H          ;发送存储地址高字节
CALL      SENDDATA
CALL      RECVACK
MOV      A,#000H          ;发送存储地址低字节
CALL      SENDDATA
CALL      RECVACK
MOV      A,#12H           ;写测试数据1
CALL      SENDDATA
CALL      RECVACK
MOV      A,#78H           ;写测试数据2
CALL      SENDDATA
CALL      RECVACK
CALL      STOP            ;发送停止命令

CALL      DELAY          ;等待设备写数据

CALL      START          ;发送起始命令
MOV      A,#0A0H          ;发送设备地址+写命令
CALL      SENDDATA
CALL      RECVACK
MOV      A,#000H          ;发送存储地址高字节
CALL      SENDDATA
CALL      RECVACK
MOV      A,#000H          ;发送存储地址低字节
CALL      SENDDATA
CALL      RECVACK
CALL      START          ;发送起始命令
MOV      A,#0AIH          ;发送设备地址+读命令
CALL      SENDDATA
CALL      RECVACK
CALL      RECVDATA        ;读取数据1
MOV      P0,A
CALL      SENDACK
CALL      RECVDATA        ;读取数据2
MOV      P2,A
CALL      SENDNAK
CALL      STOP            ;发送停止命令

JMP      $

```

END

C 语言代码

```

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

sfr      P_SW2      = 0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSR      (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST     (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR     (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST     (*(unsigned char volatile xdata *)0xfe84)

```

```
#define I2CSLADR      (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD        (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD        (*(unsigned char volatile xdata *)0xfe87)

sbit SDA          = P1^4;
sbit SCL          = P1^5;

void Wait()
{
    while (!(I2CMSST & 0x40));
    I2CMSST &= ~0x40;
}

void Start()
{
    I2CMSCR = 0x01;           //发送START 命令
    Wait();
}

void SendData(char dat)
{
    I2CTXD = dat;           //写数据到数据缓冲区
    I2CMSCR = 0x02;           //发送SEND 命令
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;           //发送读ACK 命令
    Wait();
}

char RecvData()
{
    I2CMSCR = 0x04;           //发送RECV 命令
    Wait();
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;           //设置ACK 信号
    I2CMSCR = 0x05;           //发送ACK 命令
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;           //设置NAK 信号
    I2CMSCR = 0x05;           //发送ACK 命令
    Wait();
}

void Stop()
{
    I2CMSCR = 0x06;           //发送STOP 命令
    Wait();
}
```

```
void Delay()
{
    int i;

    for (i=0; i<3000; i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
}

void main()
{
    P_SW2 = 0x80;

    I2CCFG = 0xe0;           //使能I2C 主机模式
    I2CMSST = 0x00;

    Start();                 //发送起始命令
    SendData(0xa0);          //发送设备地址+写命令
    RecvACK();
    SendData(0x00);          //发送存储地址高字节
    RecvACK();
    SendData(0x00);          //发送存储地址低字节
    RecvACK();
    SendData(0x12);          //写测试数据1
    RecvACK();
    SendData(0x78);          //写测试数据2
    RecvACK();
    Stop();                  //发送停止命令

    Delay();                 //等待设备写数据

    Start();                 //发送起始命令
    SendData(0xa0);          //发送设备地址+写命令
    RecvACK();
    SendData(0x00);          //发送存储地址高字节
    RecvACK();
    SendData(0x00);          //发送存储地址低字节
    RecvACK();
    Start();                 //发送起始命令
    SendData(0xa1);          //发送设备地址+读命令
    RecvACK();
    P0 = RecvData();          //读取数据1
    SendACK();
    P2 = RecvData();          //读取数据2
    SendNAK();
    Stop();                  //发送停止命令

    P_SW2 = 0x00;

    while (1);
}
```

20.4.3 I²C主机模式访问PCF8563

汇编代码

<i>P_SW2</i>	<i>DATA</i>	<i>0BAH</i>
<i>I2CCFG</i>	<i>XDATA</i>	<i>0FE80H</i>
<i>I2CMSCR</i>	<i>XDATA</i>	<i>0FE81H</i>
<i>I2CMSST</i>	<i>XDATA</i>	<i>0FE82H</i>
<i>I2CSLCR</i>	<i>XDATA</i>	<i>0FE83H</i>
<i>I2CSLST</i>	<i>XDATA</i>	<i>0FE84H</i>
<i>I2CSLADR</i>	<i>XDATA</i>	<i>0FE85H</i>
<i>I2CTXD</i>	<i>XDATA</i>	<i>0FE86H</i>
<i>I2CRXD</i>	<i>XDATA</i>	<i>0FE87H</i>
<i>SDA</i>	<i>BIT</i>	<i>P1.4</i>
<i>SCL</i>	<i>BIT</i>	<i>P1.5</i>
	<i>ORG</i>	<i>0000H</i>
	<i>LJMP</i>	<i>MAIN</i>
	<i>ORG</i>	<i>0100H</i>
<i>START:</i>		
	<i>MOV</i>	<i>A,#00000001B</i> ;发送 START 命令
	<i>MOV</i>	<i>DPTR,#I2CMSCR</i>
	<i>MOVX</i>	<i>@DPTR,A</i>
	<i>JMP</i>	<i>WAIT</i>
<i>SENDDATA:</i>		
	<i>MOV</i>	<i>DPTR,#I2CTXD</i> ;写数据到数据缓冲区
	<i>MOVX</i>	<i>@DPTR,A</i>
	<i>MOV</i>	<i>A,#00000010B</i>
	<i>MOV</i>	<i>DPTR,#I2CMSCR</i>
	<i>MOVX</i>	<i>@DPTR,A</i>
	<i>JMP</i>	<i>WAIT</i>
<i>RECVACK:</i>		
	<i>MOV</i>	<i>A,#00000011B</i> ;发送读 ACK 命令
	<i>MOV</i>	<i>DPTR,#I2CMSCR</i>
	<i>MOVX</i>	<i>@DPTR,A</i>
	<i>JMP</i>	<i>WAIT</i>
<i>RECVDATA:</i>		
	<i>MOV</i>	<i>A,#00000100B</i> ;发送 RECV 命令
	<i>MOV</i>	<i>DPTR,#I2CMSCR</i>
	<i>MOVX</i>	<i>@DPTR,A</i>
	<i>CALL</i>	<i>WAIT</i>
	<i>MOV</i>	<i>DPTR,#I2CRXD</i> ;从数据缓冲区读取数据
	<i>MOVX</i>	<i>A,@DPTR</i>
	<i>RET</i>	
<i>SENDACK:</i>		
	<i>MOV</i>	<i>A,#00000000B</i> ;设置 ACK 信号
	<i>MOV</i>	<i>DPTR,#I2CMSST</i>
	<i>MOVX</i>	<i>@DPTR,A</i>
	<i>MOV</i>	<i>A,#00000101B</i> ;发送 ACK 命令
	<i>MOV</i>	<i>DPTR,#I2CMSCR</i>
	<i>MOVX</i>	<i>@DPTR,A</i>
	<i>JMP</i>	<i>WAIT</i>
<i>SENDNAK:</i>		
	<i>MOV</i>	<i>A,#00000001B</i> ;设置 NAK 信号
	<i>MOV</i>	<i>DPTR,#I2CMSST</i>
	<i>MOVX</i>	<i>@DPTR,A</i>

```

        MOV      A,#00000101B          ;发送ACK命令
        MOV      DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP      WAIT

STOP:
        MOV      A,#00000110B          ;发送STOP命令
        MOV      DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP      WAIT

WAIT:
        MOV      DPTR,#I2CMSST          ;清中断标志
        MOVX    A,@DPTR
        JNB      ACC.6,WAIT
        ANL      A,#NOT 40H
        MOVX    @DPTR,A
        RET

DELAY:
        MOV      R0,#0
        MOV      R1,#0

DELAYI:
        NOP
        NOP
        NOP
        NOP
        DJNZ    R1,DELAYI
        DJNZ    R0,DELAYI
        RET

MAIN:
        MOV      SP,#3FH
        MOV      P_SW2,#80H

        MOV      A,#1110000B          ;设置I2C模块为主机模式
        MOV      DPTR,#I2CCFG
        MOVX    @DPTR,A
        MOV      A,#00000000B
        MOV      DPTR,#I2CMSST
        MOVX    @DPTR,A

        CALL    START                ;发送起始命令
        MOV      A,#0A2H
        CALL    SENDDATA             ;发送设备地址+写命令
        CALL    RECVACK
        MOV      A,#002H              ;发送存储地址
        CALL    SENDDATA
        CALL    RECVACK
        MOV      A,#00H                ;设置秒值
        CALL    SENDDATA
        CALL    RECVACK
        MOV      A,#00H                ;设置分钟值
        CALL    SENDDATA
        CALL    RECVACK
        MOV      A,#12H                ;设置小时值
        CALL    SENDDATA
        CALL    RECVACK
        CALL    STOP                 ;发送停止命令

LOOP:
        CALL    START                ;发送起始命令

```

<i>MOV</i>	<i>A,#0A2H</i>	;发送设备地址+写命令
<i>CALL</i>	<i>SENDDATA</i>	
<i>CALL</i>	<i>RECVACK</i>	
<i>MOV</i>	<i>A,#002H</i>	;发送存储地址
<i>CALL</i>	<i>SENDDATA</i>	
<i>CALL</i>	<i>RECVACK</i>	
<i>CALL</i>	<i>START</i>	;发送起始命令
<i>MOV</i>	<i>A,#0A3H</i>	;发送设备地址+读命令
<i>CALL</i>	<i>SENDDATA</i>	
<i>CALL</i>	<i>RECVACK</i>	
<i>CALL</i>	<i>RECVDATA</i>	;读取秒值
<i>MOV</i>	<i>P0,A</i>	
<i>CALL</i>	<i>SENDACK</i>	
<i>CALL</i>	<i>RECVDATA</i>	;读取分钟值
<i>MOV</i>	<i>P2,A</i>	
<i>CALL</i>	<i>SENDACK</i>	
<i>CALL</i>	<i>RECVDATA</i>	;读取小时值
<i>MOV</i>	<i>P3,A</i>	
<i>CALL</i>	<i>SENDNAK</i>	
<i>CALL</i>	<i>STOP</i>	;发送停止命令
<i>CALL</i>	<i>DELAY</i>	
<i>JMP</i>	<i>LOOP</i>	
<i>END</i>		

C 语言代码

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

sfr P_SW2 = 0xba;

#define I2CCFG    (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR   (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST   (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR   (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST   (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR   (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD    (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD    (*(unsigned char volatile xdata *)0xfe87)

sbit SDA = P1^4;
sbit SCL = P1^5;

void Wait()
{
    while (!(I2CMSST & 0x40));
    I2CMSST &= ~0x40;
}

void Start()
{
    I2CMSCR = 0x01;           //发送 START 命令
    Wait();
}

void SendData(char dat)
```

```
{  
    I2CTXD = dat;           //写数据到数据缓冲区  
    I2CMSCR = 0x02;         //发送 SEND 命令  
    Wait();  
}  
  
void RecvACK()  
{  
    I2CMSCR = 0x03;         //发送读 ACK 命令  
    Wait();  
}  
  
char RecvData()  
{  
    I2CMSCR = 0x04;         //发送 RECV 命令  
    Wait();  
    return I2CRXD;  
}  
  
void SendACK()  
{  
    I2CMSST = 0x00;          //设置 ACK 信号  
    I2CMSCR = 0x05;          //发送 ACK 命令  
    Wait();  
}  
  
void SendNAK()  
{  
    I2CMSST = 0x01;          //设置 NAK 信号  
    I2CMSCR = 0x05;          //发送 ACK 命令  
    Wait();  
}  
  
void Stop()  
{  
    I2CMSCR = 0x06;          //发送 STOP 命令  
    Wait();  
}  
  
void Delay()  
{  
    int i;  
  
    for (i=0; i<3000; i++)  
    {  
        _nop_();  
        _nop_();  
        _nop_();  
        _nop_();  
    }  
}  
  
void main()  
{  
    P_SW2 = 0x80;  
  
    I2CCFG = 0xe0;           //使能 I2C 主机模式  
    I2CMSST = 0x00;
```

```

Start();           //发送起始命令
SendData(0xa2);  //发送设备地址+写命令
RecvACK();
SendData(0x02);  //发送存储地址
RecvACK();
SendData(0x00);  //设置秒值
RecvACK();
SendData(0x00);  //设置分钟值
RecvACK();
SendData(0x12);  //设置小时值
RecvACK();
Stop();          //发送停止命令

while (1)
{
    Start();           //发送起始命令
    SendData(0xa2);  //发送设备地址+写命令
    RecvACK();
    SendData(0x02);  //发送存储地址
    RecvACK();
    Start();          //发送起始命令
    SendData(0xa3);  //发送设备地址+读命令
    RecvACK();
    P0 = RecvData(); //读取秒值
    SendACK();
    P2 = RecvData(); //读取分钟值
    SendACK();
    P3 = RecvData(); //读取小时值
    SendNAK();
    Stop();          //发送停止命令

    Delay();
}
}

```

20.4.4 I²C从机模式（中断方式）

汇编代码

<i>P_SW2</i>	<i>DATA</i>	<i>0BAH</i>
<i>I2CCFG</i>	<i>XDATA</i>	<i>0FE80H</i>
<i>I2CMSCR</i>	<i>XDATA</i>	<i>0FE81H</i>
<i>I2CMSST</i>	<i>XDATA</i>	<i>0FE82H</i>
<i>I2CSLCR</i>	<i>XDATA</i>	<i>0FE83H</i>
<i>I2CSLST</i>	<i>XDATA</i>	<i>0FE84H</i>
<i>I2CSLADR</i>	<i>XDATA</i>	<i>0FE85H</i>
<i>I2CTXD</i>	<i>XDATA</i>	<i>0FE86H</i>
<i>I2CRXD</i>	<i>XDATA</i>	<i>0FE87H</i>
<i>SDA</i>	<i>BIT</i>	<i>P1.4</i>
<i>SCL</i>	<i>BIT</i>	<i>P1.5</i>
<i>ISDA</i>	<i>BIT</i>	<i>20H.0</i> ;设备地址标志
<i>ISMA</i>	<i>BIT</i>	<i>20H.1</i> ;存储地址标志
<i>ADDR</i>	<i>DATA</i>	<i>21H</i>
	<i>ORG</i>	<i>0000H</i>
	<i>LJMP</i>	<i>MAIN</i>

	<i>ORG</i>	<i>00C3H</i>	
	<i>LJMP</i>	<i>I2CISR</i>	
	<i>ORG</i>	<i>0100H</i>	
<i>I2CISR:</i>			
	<i>PUSH</i>	<i>ACC</i>	
	<i>PUSH</i>	<i>PSW</i>	
	<i>PUSH</i>	<i>DPL</i>	
	<i>PUSH</i>	<i>DPH</i>	
	<i>MOV</i>	<i>DPTR,#I2CSLST</i>	;检测从机状态
	<i>MOVX</i>	<i>A,@DPTR</i>	
	<i>JB</i>	<i>ACC.6,STARTIF</i>	
	<i>JB</i>	<i>ACC.5,RXIF</i>	
	<i>JB</i>	<i>ACC.4,TXIF</i>	
	<i>JB</i>	<i>ACC.3,STOPIF</i>	
<i>ISRExit:</i>			
	<i>POP</i>	<i>DPH</i>	
	<i>POP</i>	<i>DPL</i>	
	<i>POP</i>	<i>PSW</i>	
	<i>POP</i>	<i>ACC</i>	
	<i>RETI</i>		
<i>STARTIF:</i>			
	<i>ANL</i>	<i>A,#NOT 40H</i>	;处理 START 事件
	<i>MOVX</i>	<i>@DPTR,A</i>	
	<i>JMP</i>	<i>ISRExit</i>	
<i>RXIF:</i>			
	<i>ANL</i>	<i>A,#NOT 20H</i>	;处理 RECV 事件
	<i>MOVX</i>	<i>@DPTR,A</i>	
	<i>MOV</i>	<i>DPTR,#I2CRXD</i>	
	<i>MOVX</i>	<i>A,@DPTR</i>	
	<i>JBC</i>	<i>ISDA,RXDA</i>	
	<i>JBC</i>	<i>ISMA,RXMA</i>	
	<i>MOV</i>	<i>R0,ADDR</i>	;处理 RECV 事件 (RECV DATA)
	<i>MOVX</i>	<i>@R0,A</i>	
	<i>INC</i>	<i>ADDR</i>	
	<i>JMP</i>	<i>ISRExit</i>	
<i>RXDA:</i>			
	<i>JMP</i>	<i>ISRExit</i>	;处理 RECV 事件 (RECV DEVICE ADDR)
<i>RXMA:</i>			
	<i>MOV</i>	<i>ADDR,A</i>	;处理 RECV 事件 (RECV MEMORY ADDR)
	<i>MOV</i>	<i>R0,A</i>	
	<i>MOVX</i>	<i>A,@R0</i>	
	<i>MOV</i>	<i>DPTR,#I2CTXD</i>	
	<i>MOVX</i>	<i>@DPTR,A</i>	
	<i>JMP</i>	<i>ISRExit</i>	
<i>TXIF:</i>			
	<i>ANL</i>	<i>A,#NOT 10H</i>	;处理 SEND 事件
	<i>MOVX</i>	<i>@DPTR,A</i>	
	<i>JB</i>	<i>ACC.1,RXNAK</i>	
	<i>INC</i>	<i>ADDR</i>	
	<i>MOV</i>	<i>R0,ADDR</i>	
	<i>MOVX</i>	<i>A,@R0</i>	
	<i>MOV</i>	<i>DPTR,#I2CTXD</i>	
	<i>MOVX</i>	<i>@DPTR,A</i>	
	<i>JMP</i>	<i>ISRExit</i>	
<i>RXNAK:</i>			
	<i>MOVX</i>	<i>A,#0FFH</i>	
	<i>MOV</i>	<i>DPTR,#I2CTXD</i>	
	<i>MOVX</i>	<i>@DPTR,A</i>	

JMP	ISREXIT	
STOPIF:		
ANL	A,#NOT 08H	; 处理 STOP 事件
MOVX	@DPTR,A	
SETB	ISDA	
SETB	ISMA	
JMP	ISREXIT	
 MAIN:		
MOV	P_SW2,#80H	
MOV	A,#10000001B	; 使能 I2C 从机模式
MOV	DPTR,#I2CCFG	
MOVX	@DPTR,A	
MOV	A,#01011010B	; 设置从机设备地址为 5A
MOV	DPTR,#I2CSLADR	
MOVX	@DPTR,A	
MOV	A,#00000000B	
MOV	DPTR,#I2CSLST	
MOVX	@DPTR,A	
MOV	A,#01111000B	; 使能从机模式中断
MOV	DPTR,#I2CSLCR	
MOVX	@DPTR,A	
SETB	ISDA	; 用户变量初始化
SETB	ISMA	
CLR	A	
MOV	ADDR,A	
MOV	R0,A	
MOVX	A,@R0	
MOV	DPTR,#I2CTXD	
MOVX	@DPTR,A	
SETB	EA	
SJMP	\$	
 END		

C 语言代码

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

sfr P_SW2 = 0xba;

#define I2CCFG (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD (*(unsigned char volatile xdata *)0xfe87)

sbit SDA = P1^4;
sbit SCL = P1^5;

bit isda; // 设备地址标志
```

```

bit           isma;          //存储地址标志
unsigned char  addr;
unsigned char pdata   buffer[256];

void I2C_Isr() interrupt 24
{
    _push_(P_SW2);
    P_SW2 |= 0x80;

    if(I2CSLST & 0x40)
    {
        I2CSLST &= ~0x40;          //处理START 事件
    }
    else if(I2CSLST & 0x20)
    {
        I2CSLST &= ~0x20;          //处理RECV 事件
        if(isda)
        {
            isda = 0;              //处理RECV 事件 (RECV DEVICE ADDR)
        }
        else if(isma)
        {
            isma = 0;              //处理RECV 事件 (RECV MEMORY ADDR)
            addr = I2CRXD;
            I2CTXD = buffer[addr];
        }
        else
        {
            buffer[addr++] = I2CRXD; //处理RECV 事件 (RECV DATA)
        }
    }
    else if(I2CSLST & 0x10)
    {
        I2CSLST &= ~0x10;          //处理SEND 事件
        if(I2CSLST & 0x02)
        {
            I2CTXD = 0xff;          //接收到NAK 则停止读取数据
        }
        else
        {
            I2CTXD = buffer[++addr]; //接收到ACK 则继续读取数据
        }
    }
    else if(I2CSLST & 0x08)
    {
        I2CSLST &= ~0x08;          //处理STOP 事件
        isda = 1;
        isma = 1;
    }

    _pop_(P_SW2);
}

void main()
{
    P_SW2 = 0x80;

    I2CCFG = 0x81;             //使能I2C 从机模式
    I2CSLADR = 0x5a;            //设置从机设备地址为5A
}

```

```

I2CSLST = 0x00;
I2CSLCR = 0x78;          //使能从机模式中断
EA = 1;

isda = 1;                //用户变量初始化
isma = 1;
addr = 0;
I2CTXD = buffer[addr];

while (1);
}

```

20.4.5 I²C从机模式（查询方式）

汇编代码

<i>P_SW2</i>	<i>DATA</i>	<i>0BAH</i>
<i>I2CCFG</i>	<i>XDATA</i>	<i>0FE80H</i>
<i>I2CMSCR</i>	<i>XDATA</i>	<i>0FE81H</i>
<i>I2CMSST</i>	<i>XDATA</i>	<i>0FE82H</i>
<i>I2CSLCR</i>	<i>XDATA</i>	<i>0FE83H</i>
<i>I2CSLST</i>	<i>XDATA</i>	<i>0FE84H</i>
<i>I2CSLADR</i>	<i>XDATA</i>	<i>0FE85H</i>
<i>I2CTXD</i>	<i>XDATA</i>	<i>0FE86H</i>
<i>I2CRXD</i>	<i>XDATA</i>	<i>0FE87H</i>
<i>SDA</i>	<i>BIT</i>	<i>P1.4</i>
<i>SCL</i>	<i>BIT</i>	<i>P1.5</i>
<i>ISDA</i>	<i>BIT</i>	<i>20H.0</i>
<i>ISMA</i>	<i>BIT</i>	<i>20H.1</i> ;设备地址标志 ;存储地址标志
<i>ADDR</i>	<i>DATA</i>	<i>21H</i>
	<i>ORG</i>	<i>0000H</i>
	<i>LJMP</i>	<i>MAIN</i>
	<i>ORG</i>	<i>0100H</i>
<i>MAIN:</i>	<i>MOV</i>	<i>P_SW2,#80H</i>
	<i>MOV</i>	<i>A,#10000001B</i> ;使能I2C 从机模式
	<i>MOV</i>	<i>DPTR,#I2CCFG</i>
	<i>MOVX</i>	<i>@DPTR,A</i>
	<i>MOV</i>	<i>A,#01011010B</i> ;设置从机设备地址为5A
	<i>MOV</i>	<i>DPTR,#I2CSLADR</i>
	<i>MOVX</i>	<i>@DPTR,A</i>
	<i>MOV</i>	<i>A,#00000000B</i>
	<i>MOV</i>	<i>DPTR,#I2CSLST</i>
	<i>MOVX</i>	<i>@DPTR,A</i>
	<i>MOV</i>	<i>A,#00000000B</i> ;禁止从机模式中断
	<i>MOV</i>	<i>DPTR,#I2CSLCR</i>
	<i>MOVX</i>	<i>@DPTR,A</i>
	<i>SETB</i>	<i>ISDA</i> ;用户变量初始化
	<i>SETB</i>	<i>ISMA</i>
	<i>CLR</i>	<i>A</i>
	<i>MOV</i>	<i>ADDR,A</i>
	<i>MOV</i>	<i>R0,A</i>

MOVX A,@R0
MOV DPTR,#I2CTXD
MOVX @DPTR,A

LOOP:

MOV DPTR,#I2CSLST ;检测从机状态
MOVX A,@DPTR
JB ACC.6,STARTIF
JB ACC.5,RXIF
JB ACC.4,TXIF
JB ACC.3,STOPIF
JMP LOOP

STARTIF:

ANL A,#NOT 40H ;处理 START 事件
MOVX @DPTR,A
JMP LOOP

RXIF:

ANL A,#NOT 20H ;处理 RECV 事件
MOVX @DPTR,A
MOV DPTR,#I2CRXD
MOVX A,@DPTR
JBC ISDA,RXDA
JBC ISMA,RXMA
MOV R0,ADDR ;处理 RECV 事件 (RECV DATA)
MOVX @R0,A
INC ADDR
JMP LOOP

RXDA:

JMP LOOP ;处理 RECV 事件 (RECV DEVICE ADDR)

RXMA:

MOV ADDR,A ;处理 RECV 事件 (RECV MEMORY ADDR)
MOV R0,A
MOVX A,@R0
MOV DPTR,#I2CTXD
MOVX @DPTR,A
JMP LOOP

TXIF:

ANL A,#NOT 10H ;处理 SEND 事件
MOVX @DPTR,A
JB ACC.1,RXNAK
INC ADDR
MOV R0,ADDR
MOVX A,@R0
MOV DPTR,#I2CTXD
MOVX @DPTR,A
JMP LOOP

RXNAK:

MOVX A,#0FFH
MOV DPTR,#I2CTXD
MOVX @DPTR,A
JMP LOOP

STOPIF:

ANL A,#NOT 08H ;处理 STOP 事件
MOVX @DPTR,A
SETB ISDA
SETB ISMA
JMP LOOP

END

C 语言代码

```

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

sfr      P_SW2      = 0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR     (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST      (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR      (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST      (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR     (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD       (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD       (*(unsigned char volatile xdata *)0xfe87)

sbit     SDA        = P1^4;
sbit     SCL        = P1^5;

bit      isda;       //设备地址标志
bit      isma;       //存储地址标志
unsigned char   addr;
unsigned char  pdata;  buffer[256];

void main()
{
    P_SW2 = 0x80;

    I2CCFG = 0x81;           //使能I2C 从机模式
    I2CSLADR = 0x5a;         //设置从机设备地址为5A
    I2CSLST = 0x00;          //禁止从机模式中断
    I2CSLCR = 0x00;

    isda = 1;                //用户变量初始化
    isma = 1;
    addr = 0;
    I2CTXD = buffer[addr];

    while (1)
    {
        if (I2CSLST & 0x40)
        {
            I2CSLST &= ~0x40;      //处理START 事件
        }
        else if (I2CSLST & 0x20)
        {
            I2CSLST &= ~0x20;      //处理RECV 事件
            if (isda)
            {
                isda = 0;          //处理RECV 事件 (RECV DEVICE ADDR)
            }
            else if (isma)
            {
                isma = 0;          //处理RECV 事件 (RECV MEMORY ADDR)
                addr = I2CRXD;
                I2CTXD = buffer[addr];
            }
            else
        }
    }
}

```

```

    {
        buffer[addr++] = I2CRXD; //处理RECV 事件 (RECV DATA)
    }
}
else if (I2CSLST & 0x10)
{
    I2CSLST &= ~0x10;      //处理SEND 事件
    if (I2CSLST & 0x02)
    {
        I2CTXD = 0xff;     //接收到NAK 则停止读取数据
    }
    else
    {
        I2CTXD = buffer[++addr]; //接收到ACK 则继续读取数据
    }
}
else if (I2CSLST & 0x08)
{
    I2CSLST &= ~0x08;      //处理STOP 事件
    isda = 1;
    isma = 1;
}
}
}
}

```

20.4.6 测试I²C从机模式代码的主机代码

汇编代码

P_SW2	DATA	0BAH
I2CCFG	XDATA	0FE80H
I2CMSCR	XDATA	0FE81H
I2CMSST	XDATA	0FE82H
I2CSLCR	XDATA	0FE83H
I2CSLST	XDATA	0FE84H
I2CSLADR	XDATA	0FE85H
I2CTXD	XDATA	0FE86H
I2CRXD	XDATA	0FE87H
SDA	BIT	P1.4
SCL	BIT	P1.5
	ORG	0000H
	LJMP	MAIN
	ORG	0100H
START:	MOV	A,#00000001B ;发送START 命令
	MOV	DPTR,#I2CMSCR
	MOVX	@DPTR,A
	JMP	WAIT
SENDDATA:	MOV	DPTR,#I2CTXD ;写数据到数据缓冲区
	MOVX	@DPTR,A
	MOV	A,#00000010B ;发送SEND 命令
	MOV	DPTR,#I2CMSCR
	MOVX	@DPTR,A
	JMP	WAIT

RECVACK:

```

MOV      A,#00000011B          ;发送读ACK 命令
MOV      DPTR,#I2CMSCR
MOVX    @DPTR,A
JMP      WAIT

```

RECVDATA:

```

MOV      A,#00000100B          ;发送RECV 命令
MOV      DPTR,#I2CMSCR
MOVX    @DPTR,A
CALL    WAIT
MOV      DPTR,#I2CRXD        ;从数据缓冲区读取数据
MOVX    A,@DPTR
RET

```

SENDACK:

```

MOV      A,#00000000B          ;设置ACK 信号
MOV      DPTR,#I2CMSST
MOVX    @DPTR,A
MOV      A,#00000101B          ;发送ACK 命令
MOV      DPTR,#I2CMSCR
MOVX    @DPTR,A
JMP      WAIT

```

SENDNAK:

```

MOV      A,#00000001B          ;设置NAK 信号
MOV      DPTR,#I2CMSST
MOVX    @DPTR,A
MOV      A,#00000101B          ;发送ACK 命令
MOV      DPTR,#I2CMSCR
MOVX    @DPTR,A
JMP      WAIT

```

STOP:

```

MOV      A,#00000110B          ;发送STOP 命令
MOV      DPTR,#I2CMSCR
MOVX    @DPTR,A
JMP      WAIT

```

WAIT:

```

MOV      DPTR,#I2CMSST        ;清中断标志
MOVX    A,@DPTR
JNB    ACC.6,WAIT
ANL    A,#NOT 40H
MOVX    @DPTR,A
RET

```

DELAY:

```

MOV      R0,#0
MOV      RI,#0

```

DELAYI:

```

NOP
NOP
NOP
NOP
DJNZ   RI,DELAYI
DJNZ   R0,DELAYI
RET

```

MAIN:

```

MOV      SP,#3FH
MOV      P_SW2,#80H
MOV      A,#11100000B          ;设置I2C 模块为主机模式

```

```

MOV      DPTR,#I2CCFG
MOVX    @DPTR,A
MOV      A,#0000000B
MOV      DPTR,#I2CMSST
MOVX    @DPTR,A

CALL    START          ;发送起始命令
MOV      A,#5AH         ;从机地址为5A
CALL    SENDDATA        ;发送设备地址+写命令
CALL    RECVACK
MOV      A,#00H          ;发送存储地址
CALL    SENDDATA
CALL    RECVACK
MOV      A,#12H          ;写测试数据1
CALL    SENDDATA
CALL    RECVACK
MOV      A,#78H          ;写测试数据2
CALL    SENDDATA
CALL    RECVACK
CALL    STOP            ;发送停止命令

CALL    DELAY           ;等待设备写数据

CALL    START           ;发送起始命令
MOV      A,#5AH         ;发送设备地址+写命令
CALL    SENDDATA
CALL    RECVACK
MOV      A,#00H          ;发送存储地址
CALL    SENDDATA
CALL    RECVACK
CALL    START           ;发送起始命令
MOV      A,#5BH          ;发送设备地址+读命令
CALL    SENDDATA
CALL    RECVACK
CALL    RECVDATA        ;读取数据1
MOV      P0,A
CALL    SENDACK
CALL    RECVDATA        ;读取数据2
MOV      P2,A
CALL    SENDNAK
CALL    STOP            ;发送停止命令

JMP     $

```

END

C 语言代码

```

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

sfr    P_SW2      = 0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR     (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST     (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR     (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST     (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR     (*(unsigned char volatile xdata *)0xfe85)

```

```
#define I2CTXD      (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD      (*(unsigned char volatile xdata *)0xfe87)

sbit SDA      = P1^4;
sbit SCL      = P1^5;

void Wait()
{
    while (!(I2CMSST & 0x40));
    I2CMSST &= ~0x40;
}

void Start()
{
    I2CMSCR = 0x01;           //发送 START 命令
    Wait();
}

void SendData(char dat)
{
    I2CTXD = dat;            //写数据到数据缓冲区
    I2CMSCR = 0x02;          //发送 SEND 命令
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;          //发送读 ACK 命令
    Wait();
}

char RecvData()
{
    I2CMSCR = 0x04;          //发送 RECV 命令
    Wait();
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;          //设置 ACK 信号
    I2CMSCR = 0x05;          //发送 ACK 命令
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;          //设置 NAK 信号
    I2CMSCR = 0x05;          //发送 ACK 命令
    Wait();
}

void Stop()
{
    I2CMSCR = 0x06;          //发送 STOP 命令
    Wait();
}

void Delay()
```

```
{  
    int i;  
  
    for (i=0; i<3000; i++)  
    {  
        _nop_();  
        _nop_();  
        _nop_();  
        _nop_();  
    }  
}  
  
void main()  
{  
    P_SW2 = 0x80;  
  
    I2CCFG = 0xe0;           //使能I2C 主机模式  
    I2CMSST = 0x00;  
  
    Start();                //发送起始命令  
    SendData(0x5a);          //发送设备地址+写命令  
    RecvACK();  
    SendData(0x00);          //发送存储地址  
    RecvACK();  
    SendData(0x12);          //写测试数据1  
    RecvACK();  
    SendData(0x78);          //写测试数据2  
    RecvACK();  
    Stop();                  //发送停止命令  
  
    Start();                //发送起始命令  
    SendData(0x5a);          //发送设备地址+写命令  
    RecvACK();  
    SendData(0x00);          //发送存储地址高字节  
    RecvACK();  
    Start();                //发送起始命令  
    SendData(0x5b);          //发送设备地址+读命令  
    RecvACK();  
    P0 = RecvData();          //读取数据1  
    SendACK();  
    P2 = RecvData();          //读取数据2  
    SendNAK();  
    Stop();                  //发送停止命令  
  
    P_SW2 = 0x00;  
  
    while (1);  
}
```

21 增强型双数据指针

STC8 系列的单片机内部集成了两组 16 位的数据指针。通过程序控制, 可实现数据指针自动递增或递减功能以及两组数据指针的自动切换功能

相关的特殊功能寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
DPL	数据指针 (低字节)	82H									0000,0000
DPH	数据指针 (高字节)	83H									0000,0000
DPL1	第二组数据指针 (低字节)	E4H									0000,0000
DPH1	第二组数据指针 (高字节)	E5H									0000,0000
DPS	DPTR 指针选择器	E3H	ID1	ID0	TSL	AU1	AU0	-	-	SEL	0000,0xx0
TA	DPTR 时序控制寄存器	AEH									0000,0000

第 1 组 16 位数据指针寄存器 (DPTR0)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DPL	82H								
DPH	83H								

DPL为低8位数据 (低字节)

DPH为高8位数据 (高字节)

DPL和DPH组合为第一组16位数据指针寄存器DPTR0

第 2 组 16 位数据指针寄存器 (DPTR1)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DPL1	E4H								
DPH1	E5H								

DPL1为低8位数据 (低字节)

DPH1为高8位数据 (高字节)

DPL1和DPH1组合为第二组16位数据指针寄存器DPTR1

数据指针控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DPS	E3H	ID1	ID0	TSL	AU1	AU0	-	-	SEL

ID1: 控制DPTR1自动递增方式

0: DPTR1 自动递增

1: DPTR1 自动递减

ID0: 控制DPTR0自动递增方式

0: DPTR0 自动递增

1: DPTR0 自动递减

TSL: DPTR0/DPTR1自动切换控制 (自动对SEL进行取反)

0: 关闭自动切换功能

1: 使能自动切换功能

当 TSL 位被置 1 后，每当执行完成相关指令后，系统会自动将 SEL 位取反。

与 TSL 相关的指令包括如下指令：

```
MOV      DPTR,#data16
INC      DPTR
MOVC    A,@A+DPTR
MOVX   A,@DPTR
MOVX  @DPTR,A
```

AU1/AU0：使能DPTR1/DPTR0使用ID1/ID0控制位进行自动递增/递减控制

0：关闭自动递增/递减功能

1：使能自动递增/递减功能

注意：在写保护模式下，AU0 和 AU1 位无法直接单独使能，若单独使能 AU1 位，则 AU0 位也会被自动使能，若单独使能 AU0，没有效果。若需要单独使能 AU1 或者 AU0，则必须使用 TA 寄存器触发 DPS 的保护机制（参考 TA 寄存器的说明）。另外，只有执行下面的 3 条指令后才会对 DPTR0/DPTR1 进行自动递增/递减操作。3 条相关指令如下：

```
MOVC  A,@A+DPTR
MOVX A,@DPTR
MOVX @DPTR,A
```

SEL：选择DPTR0/DPTR1作为当前的目标DPTR

0：选择 DPTR0 作为目标 DPTR

1：选择 DPTR1 作为目标 DPTR

SEL 选择目标 DPTR 对下面指令有效：

```
MOV      DPTR,#data16
INC      DPTR
MOVC    A,@A+DPTR
MOVX   A,@DPTR
MOVX  @DPTR,A
JMP     @A+DPTR
```

数据指针控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TA	AEH								

TA 寄存器是对 DPS 寄存器中的 AU1 和 AU0 进行写保护的。由于程序无法对 DPS 中的 AU1 和 AU0 进行单独的写入，所以当需要单独使能 AU1 或者 AU0 时，必须使用 TA 寄存器进行触发。TA 寄存器是只写寄存器。

当需要对 AU1 或者 AU0 进行单独使能时，必须按照如下的步骤进行操作：

```
CLR      EA          ;关闭中断（必需）
MOV      TA,#0AAH    ;写入触发命令序列 1
                  ;此处不能有其他任何指令
MOV      TA,#55H    ;写入触发命令序列 2
                  ;此处不能有其他任何指令
MOV      DPS,#xxH   ;写保护暂时关闭，可向 DPS 中写入任何值
                  ;DSP 再次进行写保护状态
SETB    EA          ;打开中断（如有必要）
```

21.1 范例程序

21.1.1 示例代码 1

将程序空间 1000H~1003H 的 4 个字节数据反向复制到扩展 RAM 的 0100H~0103H 中, 即

```
C:1000H -> X:0103H
C:1001H -> X:0102H
C:1002H -> X:0101H
C:1003H -> X:0100H
```

汇编代码

```
ORG 0000H
LJMP MAIN

ORG 0100H
MAIN:
    MOV SP, #3FH

    MOV DPS,#00100000B      ;使能TSL,并选择DPTR0
    MOV DPTR,#1000H          ;将1000H写入DPTR0中,执行完成后选择DPTR1为DPTR
    MOV DPTR,#0103H          ;将0103H写入DPTR1中
    MOV DPS,#10111000B       ;设置DPTR1为递减模式,DPTR0为递加模式,使能TSL以及
                           ;AU0和AU1,并选择DPTR0为当前的DPTR
    MOV R7,#4                ;设置数据复制个数

COPY_NEXT:
    CLR A
    MOVC A,@A+DPTR          ;从DPTR0所指的程序空间读取数据
    MOVX @DPTR,A              ;完成后DPTR0自动加1并将DPTR1设置为下一个目标DPTR
    MOVX @DPTR,A              ;将ACC的数据写入到DPTR1所指的XDATA中
    DJNZ R7,COPY_NEXT        ;完成后DPTR1自动减1并将DPTR0设置为下一个目标DPTR

    SJMP $

END
```

21.1.2 示例代码 2

将扩展 RAM 的 0100H~0103H 中的数据依次发送到 P0 口

汇编代码

```
ORG 0000H
LJMP MAIN

ORG 0100H
MAIN:
    MOV SP, #3FH

    CLR EA                  ;关闭中断
    MOV TA,#0AAH              ;写入DPS写保护触发命令1
    MOV TA,#55H                ;写入DPS写保护触发命令2
    MOV DPS,#00001000B         ;DPTR0递增,单独使能AU0,并选择DPTR0
    SETB EA                  ;打开中断
    MOV DPTR,#0100H            ;将0100H写入DPTR0中
```

```
MOVX A,@DPTR ;从 DPTR0 所指的 XRAM 读取数据,完成后DPTR0 自动加1
MOV P0,A ;数据输出到P0 口
MOVX A,@DPTR ;从 DPTR0 所指的 XRAM 读取数据,完成后DPTR0 自动加1
MOV P0,A ;数据输出到P0 口
MOVX A,@DPTR ;从 DPTR0 所指的 XRAM 读取数据,完成后DPTR0 自动加1
MOV P0,A ;数据输出到P0 口
MOVX A,@DPTR ;从 DPTR0 所指的 XRAM 读取数据,完成后DPTR0 自动加1
MOV P0,A ;数据输出到P0 口

SJMP $
```

END

附录A 应用注意事项

A.1 关于STC8A系列、STC8F系列芯片问题总结

1. 复位脚用于复位时，下拉电阻不大于 3K。在 1.5V 时有超过 200uA 的上拉电流，10K 电路拉不低。
2. 4 个串口用 9 位数据时，TB8 要装 2 次。
(STC8A8K 系列的 G 版芯片无此问题、STC8A4K 系列的 G 版芯片无此问题、STC8F2K 系列 E 版芯片无此问题)
3. 使用 ADC 时，要注意下面的要求：AVCC 跟 VCC 电压差不要超过 0.1V，VREF 在 2.0~AVCC 之间，P1.0~P1.7 P0.0~P0.6 电压不能比 AVCC 高。
4. 使用外部晶振时，要注意下面的要求：AVCC 跟 VCC 电压差不要超过 0.1V，P1.0~P1.7 P0.0~P0.6 电压不能比 AVCC 高，否则晶振不起振，形同死机。
5. 4 个串口的发送脚 TXDn 要下面的三选一：
A、设置为推挽输出，
B、允许内部的上拉 3K7 电阻，
C、外加 3K~5.1K 上拉电阻。
(STC8A8K 系列的 G 版芯片无此问题、STC8A4K 系列的 G 版芯片无此问题、STC8F2K 系列 E 版芯片无此问题)
6. SPI 的 SCLK 和 MOSI 如果设置为开漏输出并且 IO 输出高，则 SCLK 和 MOSI 没有输出信号，但将这两个口输出低电平，则能正常输出。并且，这两个信号是推挽输出，与 IO 设置无关。
7. 8 通道增强型 PWM 的输出，切换到 P1.0~P1.7 或 P2.0~P2.7，IO 设置为准双向口或推挽输出，PWM 输出正常，均为推挽输出，并且再直接操作 IO 将不影响 PWM 波形。但是如果将 IO 设置为开漏输出（允许内部上拉电阻或外接上拉电阻），则如果对应的 IO 输出高电平，PWM 无输出（IO 为高阻），而对应的 IO 输出低电平，PWM 有推挽输出。
8. 4 通道 PCA 工作于 PWM 时，切换到 P1.4~P1.7 或 P2.3~P2.6 或 P3.3~P3.0，情况同上类似，IO 设置为准双向口或推挽输出，PWM 输出正常，均为推挽输出，并且再直接操作 IO 将不影响 PWM 波形。但是如果将 IO 设置为开漏输出（允许内部上拉电阻或外接上拉电阻），则如果对应的 IO 输出高电平，PWM 无输出（IO 为高阻），而对应的 IO 输出低电平，PWM 有推挽输出。
9. 4 通道 PCA 工作于高速输出时，受到同一个端口其余 IO 翻转的影响，比如 PCA3 从 P2.6 高速输出脉冲，P2 口除了 P2.6 外任何一个 P2 口的改变都会影响到 P2.6 波形，哪怕是将 IO 设置为高阻。
10. 定时器 0 从 P3.5 高速输出、定时器 1 从 P3.4 高速输出、定时器 2 从 P1.3 高速输出、定时器 3 从 P0.5 高速输出、定时器 4 从 P0.7 高速输出，这些 IO 设置为准双向口或推挽输出，脉冲输出正常，均为推挽输出，并且再直接操作 IO 将不影响输出波形。但是如果将 IO 设置为开漏输出（允许内部上拉电阻或外接上拉电阻），则如果对应的 IO 输出高电平，波形无输出（IO 为高阻），而对应的 IO 输出低电平，波形有推挽输出。

A.2 关于使用CLR指令关闭EA的重要说明

对于 STC8 系列的 MCU，为了加快指令的执行速度，芯片内部对指令采用的是 4 级流水线的取指-解码-执行的并行体系，使得原本需要 12~48 个时钟周期才能完成的指令，在 STC8 系列的 MCU 中除了 MUL、DIV、DA、MOVC、MOVX 以及跳转指令外，其余指令均只需要一个时钟周期就能完成，注意此处所说的完成并不是真正的执行完成，而是在本时钟周期内完成对当前指令的解码后，在下一个时钟周期对下一条指令执行解码操作的同时执行上一条指令的动作。这样操作的好处是在宏观上确实加快了指令的执行速度，但因此而带来的负面影响是对部分指令的执行效果有一个时钟的迟滞现象。

前面所描述的迟滞现象对基于冯·洛伊曼体系结构的 8051 程序代码来说不会有任何问题，因为指令不会出现并发现象，但由于 8051 代码中存在中断机制，中断可能随时会打断当前顺序执行的代码，此时前面所描述的迟滞现象就可能会产生问题，所以一般的做法是当主循环中需要修改的变量可能会与中断中有访问冲突或者主循环中需要修改的变量在中断中需要进行逻辑判断时，就需要在主循环中在对这种变量进行修改前，先使用 CLR EA 指令将中断暂时关闭，等待对变量修改完成后在使用 SETB EA 指令打开中断，从而达到主循环和中断对变量互斥访问的目的。

注意：CLR EA 指令本身也有迟滞现象，即 CLR EA 之后，EA 并不是立即被关闭的，而是需要等待下一个时钟周期完成后 EA 才会被关闭，也就是说，运行完 CLR EA 后，需要再执行一条语句，EA 才会被真正关闭。所以正确的程序代码编写方式是在 CLR EA 指令后加 1~2 个 NOP 指令，因为 NOP 的执行对任何算式逻辑运算和逻辑判断都不会造成影响。

正确的关闭 EA 的汇编代码如下：

```
...
NOP
CLR EA
NOP
...
NOP
SETB EA
NOP
...
```

正确的关闭 EA 的 C 代码如下：

```
...
_nop_0;
EA = 0;
_nop_0;
...
_nop_0;
EA = 1;
_nop_0;
...
```

另外，对于 STC15 系列 MCU 也需要注意此问题，由于 STC15 系列的内部硬件也同样使用的是 4 级流水线的体系，所以也会有类似上面的问题。所以一般正确的做法也需要参考上面的示例代码来编写程序。

A.3 关于EEPROM编程和擦除等待时间的重要说明

表一 (STC8A 系列和 STC8F 系列 EEPROM 操作时间需求)

EEPROM 操作	最短时间	最长时间
编程	6us	7.5us
擦除	4ms	6ms

表二 (STC8A 系列和 STC8F 系列 EEPROM 操作相应等待参数的时间等待周期)

IAP_WT[2:0]			编程等待时钟数	擦除等待时钟数	适合的频率
1	1	1	7 个时钟	5000 个时钟	1MHz
1	1	0	14 个时钟	10000 个时钟	2MHz
1	0	1	21 个时钟	15000 个时钟	3MHz
1	0	0	42 个时钟	30000 个时钟	6MHz
0	1	1	84 个时钟	60000 个时钟	12MHz
0	1	0	140 个时钟	100000 个时钟	20MHz
0	0	1	168 个时钟	120000 个时钟	24MHz
0	0	0	301 个时钟	215000 个时钟	30MHz

STC8A 系列和 STC8F 系列 MCU 的内部 EEPROM 的编程和擦除等待时间必须达到表一中的要求，等待时间不可过短，也不可过长。

编程的等待时间必须在 6us~7.5us 之间，编程等待时间过小（小于最短时间 6us），则被编程的目标存储单元内部的数据可能不可靠（数据的保存期限可能达不到 25 年）；若等待时间过长（大于最长时间 7.5us 的 1.5 倍，即大于 11.25us），也可能由于有数据干扰而导致写入的数据不正确。在确保编程的等待时间要求，并在编程完成后进行数据读出对比校验，若校验正确，数据便编程正确了。

擦除的等待时间必须在 4ms~6ms 之间，擦除等待时间过小（小于最短时间 4ms），则被擦除的目标存储扇区可能没有被擦除干净；若等待时间过长（大于最长时间 6ms 的 1.5 倍，即大于 9ms），则会缩短 EEPROM 的使用寿命，即原本 10 万次的擦除寿命可能会缩短为 5 万次。

编程与擦除的等待时间请严格按照表二所给的推荐频率进行合适的选择，假如工作频率为 12MHz，请按照表二推荐将等待参数设置为 011B，若 CPU 实际的工作频率并不在表二所推荐的频率之列，则需要根据实际的频率以及表二中实际的等待时钟数进行计算，找出满足表一时间需求的等待时间参数。

例如：工作频率为 4MHz，若选择等待参数为 101B，则编程时间为 $21/4\text{MHz} = 5.25\text{us}$ ，擦除时间为 $15000/4\text{MHz} = 3.75\text{ms}$ ，时间明显不够，所以应该选择等待参数为 100B，则编程时间为 $42/4\text{MHz} = 10.5\text{us}$ ，擦除时间为 $30000/4\text{MHz} = 7.5\text{ms}$ ，时间均在最短时间和最长时间的 1.5 倍之间。

注意：EEPROM 等待操作的时钟是指对主时钟进行分频后的系统时钟，即 CPU 实际的工作时钟。若单片机使用的是内部高精度 IRC，则 EEPROM 等待操作的时钟为使用 ISP 下载软件下载时经过调节后的频率；若单片机使用的外部晶振，则 EEPROM 等待操作的时钟为外部晶振频率经过 CLKDIV 寄存器分频后的时钟（例如：若单片机使用外部晶振，且外部晶振的频率为 24MHz，CLKDIV 寄存器的值设置为 4，则 EEPROM 等待操作的时钟频率为 $24\text{MHz}/4 = 6\text{MHz}$ ，此时等待参数应选择 100B，而不能选择 001B）。

下表为之前版本资料错误的部分

IAP_WT[2:0]	读字节	写字节	擦除扇区	时钟频率
-------------	-----	-----	------	------

			(2个时钟)	(约 55us)	(约 21ms)	
1	1	1	2个时钟	55个时钟	21012个时钟	1MHz
1	1	0	2个时钟	110个时钟	42024个时钟	2MHz
1	0	1	2个时钟	165个时钟	63036个时钟	3MHz
1	0	0	2个时钟	330个时钟	126072个时钟	6MHz
0	1	1	2个时钟	660个时钟	252144个时钟	12MHz
0	1	0	2个时钟	1100个时钟	420240个时钟	20MHz
0	0	1	2个时钟	1320个时钟	504288个时钟	24MHz
0	0	0	2个时钟	1760个时钟	672384个时钟	30MHz

下表为针对之前的错误修改正确后的参数

IAP_WT[2:0]			读字节 (2个时钟)	写字节 (约 6~7.5us)	擦除扇区 (约 4~6ms)	时钟频率
1	1	1	2个时钟	7个时钟	5000个时钟	1MHz
1	1	0	2个时钟	14个时钟	10000个时钟	2MHz
1	0	1	2个时钟	21个时钟	15000个时钟	3MHz
1	0	0	2个时钟	42个时钟	30000个时钟	6MHz
0	1	1	2个时钟	84个时钟	60000个时钟	12MHz
0	1	0	2个时钟	140个时钟	100000个时钟	20MHz
0	0	1	2个时钟	168个时钟	120000个时钟	24MHz
0	0	0	2个时钟	301个时钟	215000个时钟	30MHz

A.4 STC8F2K64S4 系列应用注意事项

1. STC8F2K64S4 系列 E 版芯片重要说明 1

修正了 D 版芯片中有关 I2C 的下列问题

- a. 使用 STC8F2K64S4 系列 E 版芯片作从机, 当从机地址与本机不匹配时, MCU 不会返回应答信号 (D 版芯片会错误的回 ACK)
- b. 使用 STC8F2K64S4 系列 E 版芯片作从机, 当主机完成数据读取后给出 NAK 应答信号时, MCU 不会将下一个数据位送到 SDA 总线上 (对于 D 版芯片, 无论主机回 ACK 还是 NAK, MCU 都会错误的将下一个数据位送到 SDA 总线上)
- c. 使用 STC8F2K64S4 系列 E 版芯片作主机, 当 I2C 总线上产生干扰信号时, I2C 通讯会因此而中断, 此时只需要将 I2C 总使能位 ENI2C (I2CCFG 寄存器的 bit7) 关闭, 然后再打开, I2C 主机即可恢复正常工作状态 (当发生此类情况时, D 版芯片必须给 MCU 重新上电才可用)

2. STC8F2K64S4 系列 E 版芯片重要说明 2

修正了 D 版芯片中关于串口 1 的模式 2 和模式 3 时, 在设置发送数据的第 9 位 (TB8) 时, 需要连续设置两次才有效的问题, E 版芯片只需要设置一次即可。

3. STC8F2K64S4 系列 E 版芯片重要说明 3

修正了下列串口发送数据时需要加上拉电阻或者需要设置为强推挽模式的问题, E 版芯片下列串口发送脚在发送数据时不需要额外处理:

- a. TXD (P3.1)、TXD_2 (P3.7)、TXD_3 (P1.7)、TXD_4 (P4.4)
- b. TXD2 (P1.1)、TXD2_2 (P4.2)
- c. TXD3 (P0.1)
- d. TXD4 (P0.3)
- e. 注意: 目前仍有 TXD3_2 (P5.1) 和 TXD4_2 (P5.3) 作为串口发送数据时需要加上拉电阻或者需要设置为推挽模式

4. STC8F2K64S4 系列 D 版芯片重要说明 1

所有串口(包括串口 1、串口 2、串口 3、串口 4)的串口发送端发送串口数据时, 发送端口均需要进行下面的设置: (3 种方式任选其一)

- a. 设置 I/O 口为准双向口模式并打开内部的上拉电阻
- b. 设置 I/O 口为准双向口模式并外接 3~10K 的上拉电阻
- c. 设置 I/O 口为强推挽模式

5. STC8F2K64S4 系列 D 版芯片重要说明 2

串口 1 的模式 2 和模式 3 时, 在设置发送数据的第 9 位 (TB8) 时, 需要连续设置两次才有效。串口 2、串口 3 和串口 4 无此问题

6. STC8F2K64S4 系列 C 版芯片重要说明

- a. 所有串口(包括串口 1、串口 2、串口 3、串口 4)的串口发送端发送串口数据时, 发送端口均需要软件设置为开漏模式并打开内部的上拉电阻或者外接 3~10K 的上拉电阻
- b. 所有的只做为输入的 I/O 口, 建议将其设置为高阻/仅为输入模式, 并打开内部新增加的 上拉电阻/4.2K, 或外加上拉, 也可用传统 8051 的弱上拉模式读外部状态, 只要对外是置“1”的状态, 就可以作为输入, 但新型 8051 有更好的高阻/仅为输入模式模式

- c. 所有的只做为输出的 I/O 口, 建议将其设置为开漏模式, 并打开内部新增加的 上拉电阻/4.2K, 或外接 5~10K 的上拉电阻。
 - 1. A: 要对外输出 “1”, 只要对外置 “1” 即可, 此时需要内部的上拉电阻/4.2K 已打开, 或外部已接 5~10K 的上拉电阻;
 - 2. B: 要对外输出 “0”, 只要对外清 “0” 即可, 此时可再关闭内部的上拉电阻, 可降低功耗 $<5V/4.2K = 1.2mA, 3.3V/4.2K = 0.78mA>$
- d. 所有的既要做为输入又要做为输出的 I/O 口, 建议将其设置为开漏模式, 并打开内部新增加的 上拉电阻/4.2K, 或外接 5~10K 的上拉电阻。
 - 1. A: 做为输入时, 要对外已是输出 “1”的状态, 此时需要内部的上拉电阻/4.2K 已打开, 或外部已接 5~10K 的上拉电阻, 传通 8051 的 P0 口用法;
 - 2. B: 要对外输出 “1”, 只要对外置 “1” 即可, 此时需要内部的上拉电阻/4.2K 已打开, 或外部已接 5~10K 的上拉电阻
 - 3. C, 要对外输出 “0”, 只要对外清 “0” 即可, 此时可再关闭内部的上拉电阻, 可降低功耗 $<5V/4.2K = 1.2mA, 3.3V/4.2K = 0.78mA>$

打开 P0 口内部上拉 4.2K 电阻的寄存器地址, P0PU, 0xFE10

打开 P1 口内部上拉 4.2K 电阻的寄存器地址, P1PU, 0xFE11

打开 P2 口内部上拉 4.2K 电阻的寄存器地址, P2PU, 0xFE12

打开 P3 口内部上拉 4.2K 电阻的寄存器地址, P3PU, 0xFE13

打开 P4 口内部上拉 4.2K 电阻的寄存器地址, P4PU, 0xFE14

打开 P5 口内部上拉 4.2K 电阻的寄存器地址, P5PU, 0xFE15

打开 P6 口内部上拉 4.2K 电阻的寄存器地址, P6PU, 0xFE16

打开 P7 口内部上拉 4.2K 电阻的寄存器地址, P7PU, 0xFE17

//如下特殊功能寄存器位于扩展 RAM 区域

//访问这些寄存器,需先将 P_SW2 的 BIT7 设置为 1,才可正常读写

```
#define P0PU      (*(unsigned char volatile xdata *)0xfe10)
#define P1PU      (*(unsigned char volatile xdata *)0xfe11)
#define P2PU      (*(unsigned char volatile xdata *)0xfe12)
#define P3PU      (*(unsigned char volatile xdata *)0xfe13)
#define P4PU      (*(unsigned char volatile xdata *)0xfe14)
#define P5PU      (*(unsigned char volatile xdata *)0xfe15)
#define P6PU      (*(unsigned char volatile xdata *)0xfe16)
#define P7PU      (*(unsigned char volatile xdata *)0xfe17)
```

- 7. STC8F2K64S4-LQFP44/LQFP32 的 B 版芯片, 送样中, 样品有如下问题: (所有的问题在 C 版芯片中都会进行修正)

=====串口接收需要 2 个停止位(包括串口 1、串口 2、串口 3、串口 4), 系统中如何解决, 发送方如不是 STC 单片机, 如 32-bit 的 CPU/GPU/DSP, 他们的 UART 发送停止位往往有 1-bit/1.5-bit/2-bit 的选择, 直接选择 2 位停止位即可, 如是 STC 量产型单片机, 则发送完成后只有一个停止位, 需等待一个停止位的时间后再发送,, 但此版本的 STC8F2K64S4(B 版)也做成了发送时是固定 2 个停止位, 会在下一版改回成一个停止位

=====当串口 1 使用工作于模式 2 的定时器 1 作为串口的波特率发生器时, SMOD(PCON.7)位必须置 1, 即必须波特率加倍串口 1 才可正常工作, 否则波特率不正确。若使用定时器 2 或者工作于模式 0 的定时器 1 作为串口波特率发生器时无此问题

=====当用户使用扩展 RAM 区的特殊功能寄存器 (XSFR) 时, 数据同时会写到内部扩展 RAM 的 2K 字节的最后 512 字节区, 若用户没有外扩 SRAM, 可在访问 XSFR 前将 EXRAM 设

置为 1, 访问完成后再将 EXTRAM 设置为 0, 这样既可正确访问 XSFR, 又不影响内部扩展 RAM 的使用

=====固件版本为 7.3.5U 及更早固件版本的 STC8F2K 系列的芯片, 在使用仿真功能时, 内部扩展 RAM 只能使用 1K (0000H ~ 03FFH), 即仿真保留区域为 (0400H ~ 07FFH), 固件版本为 7.3.6U 及更新固件版本的 STC8F2K 系列的芯片, 在使用仿真功能时, 内部扩展 RAM 可使用使用 1.25K (0000H ~ 04FFH), 即仿真保留区域为 (0500H ~ 07FFH)

A.5 STC8F2K64S2 系列应用注意事项

1. STC8F2K64S2 系列 E 版芯片重要说明 1

修正了 D 版芯片中有关 I2C 的下列问题

- a. 使用 STC8F2K64S2 系列 E 版芯片作从机, 当从机地址与本机不匹配时, MCU 不会返回应答信号 (D 版芯片会错误的回 ACK)
- b. 使用 STC8F2K64S2 系列 E 版芯片作从机, 当主机完成数据读取后给出 NAK 应答信号时, MCU 不会将下一个数据位送到 SDA 总线上 (对于 D 版芯片, 无论主机回 ACK 还是 NAK, MCU 都会错误的将下一个数据位送到 SDA 总线上)
- c. 使用 STC8F2K64S2 系列 E 版芯片作主机, 当 I2C 总线上产生干扰信号时, I2C 通讯会因此而中断, 此时只需要将 I2C 总使能位 ENI2C (I2CCFG 寄存器的 bit7) 关闭, 然后再打开, I2C 主机即可恢复正常工作状态 (当发生此类情况时, D 版芯片必须给 MCU 重新上电才可用)

2. STC8F2K64S2 系列 E 版芯片重要说明 2

修正了 D 版芯片中关于串口 1 的模式 2 和模式 3 时, 在设置发送数据的第 9 位 (TB8) 时, 需要连续设置两次才有效的问题, E 版芯片只需要设置一次即可。

3. STC8F2K64S2 系列 E 版芯片重要说明 3

修正了下列串口发送数据时需要加上拉电阻或者需要设置为强推挽模式的问题, E 版芯片下列串口发送脚在发送数据时不需要额外处理:

- a. TXD (P3.1)、TXD_2 (P3.7)、TXD_3 (P1.7)、TXD_4 (P4.4)
- b. TXD2 (P1.1)、TXD2_2 (P4.2)
- c. TXD3 (P0.1)
- d. TXD4 (P0.3)
- e. 注意: 目前仍有 TXD3_2 (P5.1) 和 TXD4_2 (P5.3) 作为串口发送数据时需要加上拉电阻或者需要设置为推挽模式

4. STC8F2K64S2 系列 D 版芯片重要说明 1

所有串口(包括串口 1、串口 2、串口 3、串口 4)的串口发送端发送串口数据时, 发送端口均需要进行下面的设置: (3 种方式任选其一)

- a. 设置 I/O 口为准双向口模式并打开内部的上拉电阻
- b. 设置 I/O 口为准双向口模式并外接 3~10K 的上拉电阻
- c. 设置 I/O 口为强推挽模式

5. STC8F2K64S2 系列 D 版芯片重要说明 2

串口 1 的模式 2 和模式 3 时, 在设置发送数据的第 9 位 (TB8) 时, 需要连续设置两次才有效。串口 2、串口 3 和串口 4 无此问题

6. STC8F2K64S2 系列 C 版芯片的重要说明

- a. 所有串口(包括串口 1、串口 2、串口 3、串口 4)的串口发送端发送串口数据时, 发送端口均需要软件设置为开漏模式并打开内部的上拉电阻或者外接 3~10K 的上拉电阻
- b. 所有的只做为输入的 I/O 口, 建议将其设置为高阻/仅为输入模式, 并打开内部新增加的 上拉电阻/4.2K, 或外加上拉, 也可用传统 8051 的弱上拉模式读外部状态, 只要对外是置“1”的状态, 就可以作为输入, 但新型 8051 有更好的高阻/仅为输入模式模式
- c. 所有的只做为输出的 I/O 口, 建议将其设置为开漏模式, 并打开内部新增加的 上拉电

阻/4.2K，或外接5~10K的上拉电阻。

1. A: 要对外输出“1”，只要对外置“1”即可，此时需要内部的上拉电阻/4.2K已打开，或外部已接5~10K的上拉电阻；
2. B: 要对外输出“0”，只要对外清“0”即可，此时可再关闭内部的上拉电阻，可降低功耗<5V/4.2K = 1.2mA, 3.3V/4.2K = 0.78mA>
- d. 所有的既要做为输入又要做为输出的I/O口，建议将其设置为开漏模式，并打开内部新增加的上拉电阻/4.2K，或外接5~10K的上拉电阻。
 1. A: 做为输入时，要对外已是输出“1”的状态，此时需要内部的上拉电阻/4.2K已打开，或外部已接5~10K的上拉电阻，传通8051的P0口用法；
 2. B: 要对外输出“1”，只要对外置“1”即可，此时需要内部的上拉电阻/4.2K已打开，或外部已接5~10K的上拉电阻
 3. C: 要对外输出“0”，只要对外清“0”即可，此时可再关闭内部的上拉电阻，可降低功耗<5V/4.2K = 1.2mA, 3.3V/4.2K = 0.78mA>

打开P0口内部上拉4.2K电阻的寄存器地址，P0PU, 0xFE10

打开P1口内部上拉4.2K电阻的寄存器地址，P1PU, 0xFE11

打开P2口内部上拉4.2K电阻的寄存器地址，P2PU, 0xFE12

打开P3口内部上拉4.2K电阻的寄存器地址，P3PU, 0xFE13

打开P4口内部上拉4.2K电阻的寄存器地址，P4PU, 0xFE14

打开P5口内部上拉4.2K电阻的寄存器地址，P5PU, 0xFE15

打开P6口内部上拉4.2K电阻的寄存器地址，P6PU, 0xFE16

打开P7口内部上拉4.2K电阻的寄存器地址，P7PU, 0xFE17

//如下特殊功能寄存器位于扩展RAM区域

//访问这些寄存器,需先将P_SW2的BIT7设置为1,才可正常读写

```
#define P0PU      (*(unsigned char volatile xdata *)0xfe10)
#define P1PU      (*(unsigned char volatile xdata *)0xfe11)
#define P2PU      (*(unsigned char volatile xdata *)0xfe12)
#define P3PU      (*(unsigned char volatile xdata *)0xfe13)
#define P4PU      (*(unsigned char volatile xdata *)0xfe14)
#define P5PU      (*(unsigned char volatile xdata *)0xfe15)
#define P6PU      (*(unsigned char volatile xdata *)0xfe16)
#define P7PU      (*(unsigned char volatile xdata *)0xfe17)
```

A.6 STC8A8K64S4A12 系列应用注意事项

1. STC8A8K64S4A12 系列 G 版芯片重要说明 1

修正了 F 版芯片中有关 I2C 的下列问题

- a. 使用 STC8A8K64S4A12 系列 G 版芯片作从机, 当从机地址与本机不匹配时, MCU 不会返回应答信号 (F 版芯片会错误的回 ACK)
- b. 使用 STC8A8K64S4A12 系列 G 版芯片作从机, 当主机完成数据读取后给出 NAK 应答信号时, MCU 不会将下一个数据位送到 SDA 总线上 (对于 F 版芯片, 无论主机回 ACK 还是 NAK, MCU 都会错误的将下一个数据位送到 SDA 总线上)
- c. 使用 STC8A8K64S4A12 系列 G 版芯片作主机, 当 I2C 总线上产生干扰信号时, I2C 通讯会因此而中断, 此时只需要将 I2C 总使能位 ENI2C (I2CCFG 寄存器的 bit7) 关闭, 然后再打开, I2C 主机即可恢复正常工作状态 (当发生此类情况时, F 版芯片必须给 MCU 重新上电才可用)

2. STC8A8K64S4A12 系列 G 版芯片重要说明 2

修正了 F 版芯片中关于串口 1 的模式 2 和模式 3 时, 在设置发送数据的第 9 位 (TB8) 时, 需要连续设置两次才有效的问题, G 版芯片只需要设置一次即可。

3. STC8A8K64S4A12 系列 G 版芯片重要说明 3

修正了下列串口发送数据时需要加上拉电阻或者需要设置为强推挽模式的问题, G 版芯片下列串口发送脚在发送数据时不需要额外处理:

- a. TXD (P3.1)、TXD_3 (P1.7)
- b. TXD2 (P1.1)、TXD2_2 (P4.2)
- c. 注意: 目前仍有 TXD_2 (P3.7)、TXD_4 (P4.4)、TXD3 (P0.1)、TXD3_2 (P5.1)、TXD4 (P0.3) 和 TXD4_2 (P5.3) 作为串口发送数据时需要加上拉电阻或者需要设置为推挽模式

4. 增强型 PWM 的中断请求标志位修改为: 当发生中断请求时硬件自动写 1 置中断请求标志, 用户只能写 0 清中断请求标志 (用户程序中不能写 1 设置中断请求标志, 用户写 1 时视为无效)

5. STC8A8K64S4A12 系列 F 版芯片重要说明 1

所有串口(包括串口 1、串口 2、串口 3、串口 4)的串口发送端发送串口数据时, 发送端口均需要进行下面的设置: (3 种方式任选其一)

- a. 设置 I/O 口为准双向口模式并打开内部的上拉电阻
- b. 设置 I/O 口为准双向口模式并外接 3~10K 的上拉电阻
- c. 设置 I/O 口为强推挽模式

6. STC8A8K64S4A12 系列 F 版芯片重要说明 2

串口 1 的模式 2 和模式 3 时, 在设置发送数据的第 9 位 (TB8) 时, 需要连续设置两次才有效。串口 2、串口 3 和串口 4 无此问题

7. STC8A8K64S4A12 系列 E 版芯片重要说明

- a. 所有串口(包括串口 1、串口 2、串口 3、串口 4)的发送端口均需要软件设置为开漏模式并打开内部的上拉电阻或者外接 3~10K 的上拉电阻
- b. 所有的只做为输入的 I/O 口, 建议将其设置为高阻/仅为输入模式, 并打开内部新增加的 上拉电阻/4.2K, 或外加上拉, 也可用传统 8051 的弱上拉模式读外部状态, 只要对

- 外是置“1”的状态，就可以作为输入，但新型 8051 有更好的高阻/仅为输入模式模式
- c. 所有的只做为输出的 I/O 口，建议将其设置为开漏模式，并打开内部新增加的 上拉电阻/4.2K，或外接 5~10K 的上拉电阻
 - 1. A, 要对外输出“1”，只要对外置“1”即可，此时需要内部的上拉电阻/4.2K 已打开，或外部已接 5~10K 的上拉电阻
 - 2. B, 要对外输出“0”，只要对外清“0”即可，此时可再关闭内部的上拉电阻，可降低功耗 $<5V/4.2K = 1.2mA, 3.3V/4.2K = 0.78mA>$
 - d. 所有的既要做为输入又要做为输出的 I/O 口，建议将其设置为开漏模式，并打开内部新增加的 上拉电阻/4.2K，或外接 5~10K 的上拉电阻
 - 1. A, 做为输入时，要对外已是输出“1”的状态，此时需要内部的上拉电阻/4.2K 已打开，或外部已接 5~10K 的上拉电阻，传通 8051 的 P0 口用法
 - 2. B, 要对外输出“1”，只要对外置“1”即可，此时需要内部的上拉电阻/4.2K 已打开，或外部已接 5~10K 的上拉电阻
 - 3. C, 要对外输出“0”，只要对外清“0”即可，此时可再关闭内部的上拉电阻，可降低功耗 $<5V/4.2K = 1.2mA, 3.3V/4.2K = 0.78mA>$

打开 P0 口内部上拉 4.2K 电阻的寄存器地址, P0PU, 0xFE10

打开 P1 口内部上拉 4.2K 电阻的寄存器地址, P1PU, 0xFE11

打开 P2 口内部上拉 4.2K 电阻的寄存器地址, P2PU, 0xFE12

打开 P3 口内部上拉 4.2K 电阻的寄存器地址, P3PU, 0xFE13

打开 P4 口内部上拉 4.2K 电阻的寄存器地址, P4PU, 0xFE14

打开 P5 口内部上拉 4.2K 电阻的寄存器地址, P5PU, 0xFE15

打开 P6 口内部上拉 4.2K 电阻的寄存器地址, P6PU, 0xFE16

打开 P7 口内部上拉 4.2K 电阻的寄存器地址, P7PU, 0xFE17

//如下特殊功能寄存器位于扩展 RAM 区域

//访问这些寄存器,需先将 P_SW2 的 BIT7 设置为 1,才可正常读写

```
#define P0PU      (*(unsigned char volatile xdata *)0xfe10)
#define P1PU      (*(unsigned char volatile xdata *)0xfe11)
#define P2PU      (*(unsigned char volatile xdata *)0xfe12)
#define P3PU      (*(unsigned char volatile xdata *)0xfe13)
#define P4PU      (*(unsigned char volatile xdata *)0xfe14)
#define P5PU      (*(unsigned char volatile xdata *)0xfe15)
#define P6PU      (*(unsigned char volatile xdata *)0xfe16)
#define P7PU      (*(unsigned char volatile xdata *)0xfe17)
```

e. ADC 相关问题

- 1. AVCC 与 VCC 的电压压差要小于 0.3V
- 2. 必须软件将相应的 ADC 转换口设置为 input 高阻输入模式或开漏
- 3. 要想读到 0, ADC 的转换速度要用最快的那一档，在查原因，慢反而不行
- 4. 相关 ADC 口转换时有如下一些口被 IC 内部误设置为高阻模式,软件无法使用和控制，请不要使用被误设为高阻输入模式的口
- 5. 如使用 ADC0/通道 0, 输入口为 P1.0, P1.7 被误设置为高阻, 建议系统中让此端口空着, 不要用 P1.7
- 6. 如使用 ADC1/通道 1, 输入口为 P1.1, P0.0 被误设置为高阻, 建议系统中让此端口空着, 不要用 P0.0
- 7. 如使用 ADC2/通道 2, 输入口为 P1.2, P0.1 被误设置为高阻, 建议系统中让此端

口空着, 不要用 P0.1

8. 如使用 ADC3/通道 3, 输入口为 P1.3, P0.2 被误设置为高阻, 建议系统中让此端口空着, 不要用 P0.2
9. 如使用 ADC4/通道 4, 输入口为 P1.4, P0.3 被误设置为高阻, 建议系统中让此端口空着, 不要用 P0.3
10. 如使用 ADC5/通道 5, 输入口为 P1.5, P0.4 被误设置为高阻, 建议系统中让此端口空着, 不要用 P0.4
11. 如使用 ADC6/通道 6, 输入口为 P1.6, P0.5 被误设置为高阻, 建议系统中让此端口空着, 不要用 P0.5
12. 如使用 ADC7/通道 7, 输入口为 P1.7, P0.6 被误设置为高阻, 建议系统中让此端口空着, 不要用 P0.6
13. 如使用 ADC8/通道 8, 输入口为 P0.0, P1.0 被误设置为高阻, 建议系统中让此端口空着, 不要用 P1.0
14. 如使用 ADC9/通道 9, 输入口为 P0.1, P1.1 被误设置为高阻, 建议系统中让此端口空着, 不要用 P1.1
15. 如使用 ADC10/通道 10, 输入口为 P0.2, P1.2 被误设置为高阻, 建议系统中让此端口空着, 不要用 P1.2
16. 如使用 ADC11/通道 11, 输入口为 P0.3, P1.3 被误设置为高阻, 建议系统中让此端口空着, 不要用 P1.3
17. 如使用 ADC12/通道 12, 输入口为 P0.4, P1.4 被误设置为高阻, 建议系统中让此端口空着, 不要用 P1.4
18. 如使用 ADC13/通道 13, 输入口为 P0.5, P1.5 被误设置为高阻, 建议系统中让此端口空着, 不要用 P1.5
19. 如使用 ADC14/通道 14, 输入口为 P0.6, P1.6 被误设置为高阻, 建议系统中让此端口空着, 不要用 P1.6
20. 如使用 ADC15/通道 15, 输入为内部 Vref/1.344V, P1.7 被误设置为高阻, 建议系统中让此端口空着, 不要用 P1.7

8. STC8A8K64S4A12-LQFP64S/LQFP48/44 的 D 版芯片, 送样中, 样品有如下问题: (所有的问题在 E 版芯片中都会进行修正)

=====串口接收需要 2 个停止位(包括串口 1、串口 2、串口 3、串口 4), 系统中如何解决, 发送方如不是 STC 单片机, 如 32-bit 的 CPU/GPU/DSP, 他们的 UART 发送停止位往往有 1-bit/1.5-bit/2-bit 的选择, 直接选择 2 位停止位即可, 如是 STC 量产型单片机, 则发送完成后只有一个停止位, 需等待一个停止位的时间后再发送,, 但此版本的 STC8F2K64S4(B 版)也做成了发送时是固定 2 个停止位, 会在下一版改回成一个停止位

=====当串口 1 使用工作于模式 2 的定时器 1 作为串口的波特率发生器时, SMOD(PCON.7)位必须置 1, 即必须波特率加倍串口 1 才可正常工作, 否则波特率不正确。若使用定时器 2 或者工作于模式 0 的定时器 1 作为串口波特率发生器时无此问题

=====当用户使用扩展 RAM 区的特殊功能寄存器 (XSFR) 时, 数据同时会写到内部扩展 RAM 的 8K 字节的最后 512 字节区, 若用户没有外扩 SRAM, 可在访问 XSFR 前将 EXRAM 设置为 1, 访问完成后再将 EXRAM 设置为 0, 这样既可正确访问 XSFR, 又不影响内部扩展 RAM 的使用

=====12 位 16 通道 ADC 可达 11.5 位(ADC7), 离 AGnd 管脚最近的 ADC7 通道最好, 其次 ADC6/ADC5/ADC4/ADC3/ADC2/ADC1/ADC0, ADC14/ADC13/ADC12/ADC11/ADC10/ADC9/ADC8, 离 AGnd 管脚最远的 ADC8 最差建议在所用的 ADC 输入通道就近接一个 0.047uF - 0.1uF/0.2uF 电容到模拟地 AGnd, 抵制 MCU 数字电源和地的干扰

=====固件版本为 7.3.5U 及更早固件版本的 STC8A8K 系列和 STC8F8K 系列的芯片, 在使用仿真

功能时，内部扩展 RAM 只能使用 3K (0000H ~ 0BFFH)，即仿真保留区域为 (0C00H ~ 0FFFH)，固件版本为 7.3.6U 的 STC8A8K 系列和 STC8F8K 系列的芯片，在使用仿真功能时，内部扩展 RAM 可使用 7.25K (0000H ~ 0CFFH), (1000H ~ 1FFFH)，即仿真保留区域为 (0D00H ~ 0FFFH)，固件版本为 7.3.7U 及更新固件版本的 STC8A8K 系列和 STC8F8K 系列的芯片，在使用仿真功能时，内部扩展 RAM 可使用 7.25K (0000H ~ 1CFFH)，即仿真保留区域为 (1D00H ~ 1FFFH)

STCMCU

A.7 STC8A4K64S2A12 系列应用注意事项

1. STC8A4K64S2A12 系列 G 版芯片重要说明 1

修正了 F 版芯片中有关 I2C 的下列问题

- a. 使用 STC8A4K64S2A12 系列 G 版芯片作从机, 当从机地址与本机不匹配时, MCU 不会返回应答信号 (F 版芯片会错误的回 ACK)
- b. 使用 STC8A4K64S2A12 系列 G 版芯片作从机, 当主机完成数据读取后给出 NAK 应答信号时, MCU 不会将下一个数据位送到 SDA 总线上 (对于 F 版芯片, 无论主机回 ACK 还是 NAK, MCU 都会错误的将下一个数据位送到 SDA 总线上)
- c. 使用 STC8A4K64S2A12 系列 G 版芯片作主机, 当 I2C 总线上产生干扰信号时, I2C 通讯会因此而中断, 此时只需要将 I2C 总使能位 ENI2C (I2CCFG 寄存器的 bit7) 关闭, 然后再打开, I2C 主机即可恢复正常工作状态 (当发生此类情况时, F 版芯片必须给 MCU 重新上电才可用)

2. STC8A4K64S2A12 系列 G 版芯片重要说明 2

修正了 F 版芯片中关于串口 1 的模式 2 和模式 3 时, 在设置发送数据的第 9 位 (TB8) 时, 需要连续设置两次才有效的问题, G 版芯片只需要设置一次即可。

3. STC8A4K64S2A12 系列 G 版芯片重要说明 3

修正了下列串口发送数据时需要加上拉电阻或者需要设置为强推挽模式的问题, G 版芯片下列串口发送脚在发送数据时不需要额外处理:

- a. TXD (P3.1)、TXD_3 (P1.7)
- b. TXD2 (P1.1)、TXD2_2 (P4.2)
- c. 注意: 目前仍有 TXD_2 (P3.7)、TXD_4 (P4.4)、TXD3 (P0.1)、TXD3_2 (P5.1)、TXD4 (P0.3) 和 TXD4_2 (P5.3) 作为串口发送数据时需要加上拉电阻或者需要设置为推挽模式

4. 增强型 PWM 的中断请求标志位修改为: 当发生中断请求时硬件自动写 1 置中断请求标志, 用户只能写 0 清中断请求标志 (用户程序中不能写 1 设置中断请求标志, 用户写 1 时视为无效)

5. STC8A4K64S2A12 系列 F 版芯片重要说明 1

所有串口(包括串口 1、串口 2、串口 3、串口 4)的串口发送端发送串口数据时, 发送端口均需要进行下面的设置: (3 种方式任选其一)

- a. 设置 I/O 口为准双向口模式并打开内部的上拉电阻
- b. 设置 I/O 口为准双向口模式并外接 3~10K 的上拉电阻
- c. 设置 I/O 口为强推挽模式

6. STC8A4K64S2A12 系列 F 版芯片重要说明 2

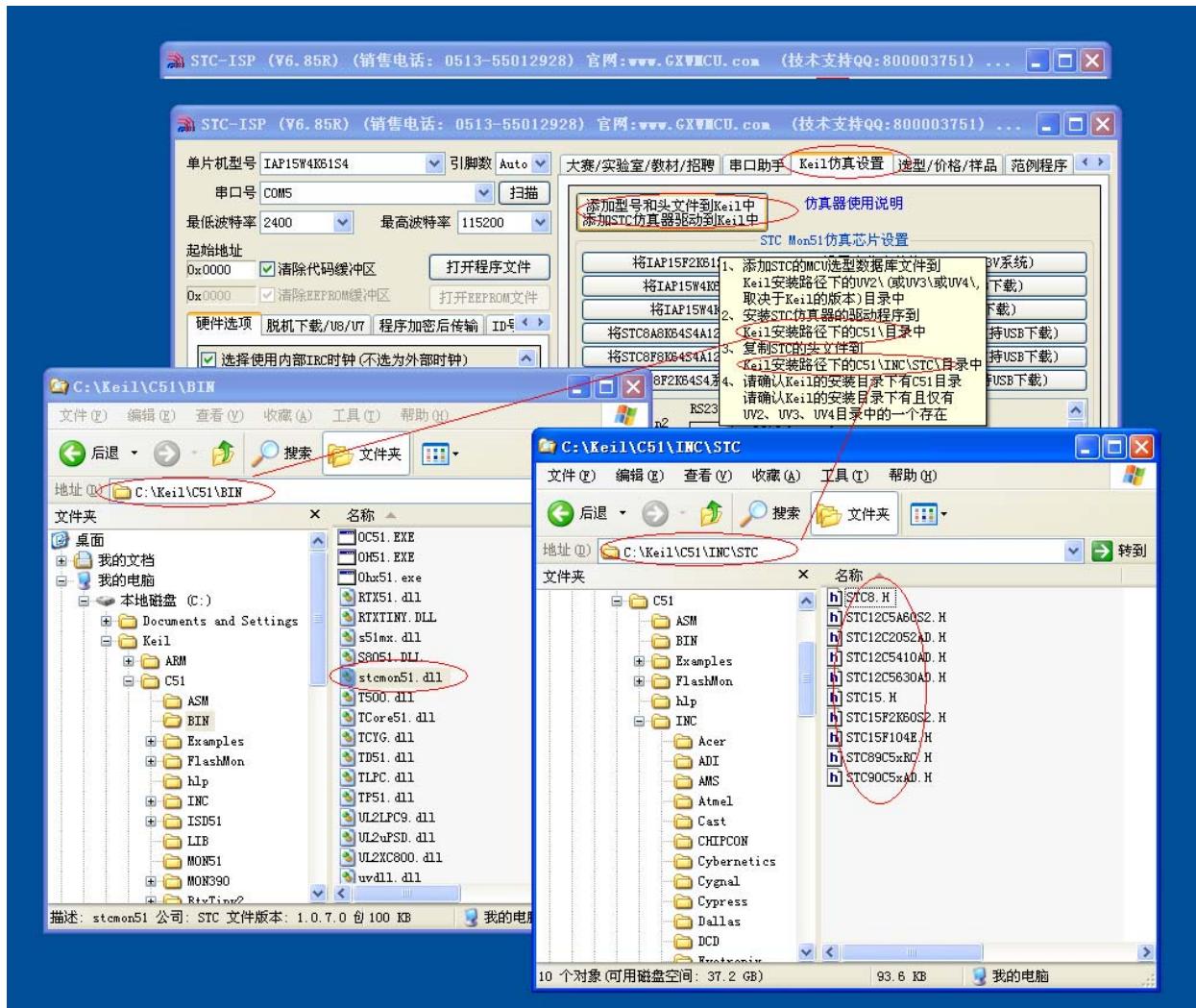
串口 1 的模式 2 和模式 3 时, 在设置发送数据的第 9 位 (TB8) 时, 需要连续设置两次才有效。串口 2、串口 3 和串口 4 无此问题

A.8 使用外部晶振对STC8 系列进行仿真的注意事项

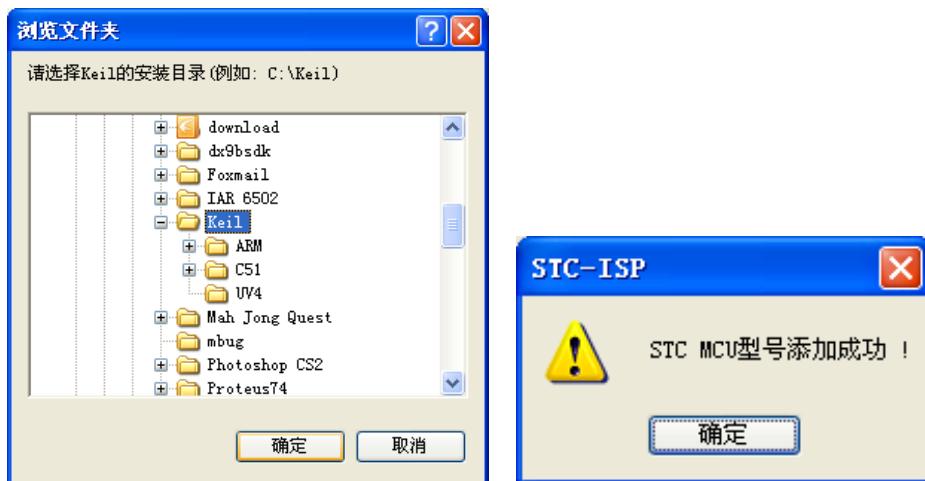
1. 当使用 STC8 系列芯片作仿真器进行仿真时，若需要在用户系统中使用外部晶振，则在制作仿真芯片时，必须将内部 IRC 的频率设置为外部晶振相同的频率。
(例如：用户需要在系统中使用外部 11.0592MHz 晶振，则在制作仿真芯片时，需要将内部 IRC 的频率由默认的 24MHz 改为 11.0592MHz，否则仿真时会出错)

附录B STC仿真器使用指南

1、安装 Keil 版本的仿真驱动

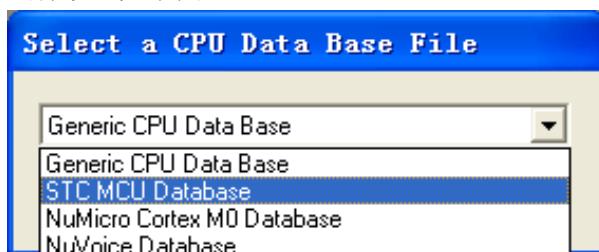


如上图，首先选择“Keil 仿真设置”页面，点击“添加 MCU 型号到 Keil 中”，在出现的如下的目录选择窗口中，定位到 Keil 的安装目录(一般可能为“C:\Keil\”)，“确定”后出现下图中右边所示的提示信息，表示安装成功。添加头文件的同时也会安装 STC 的 Monitor51 仿真驱动 STCMON51.DLL，驱动与头文件的安装目录如上图所示。

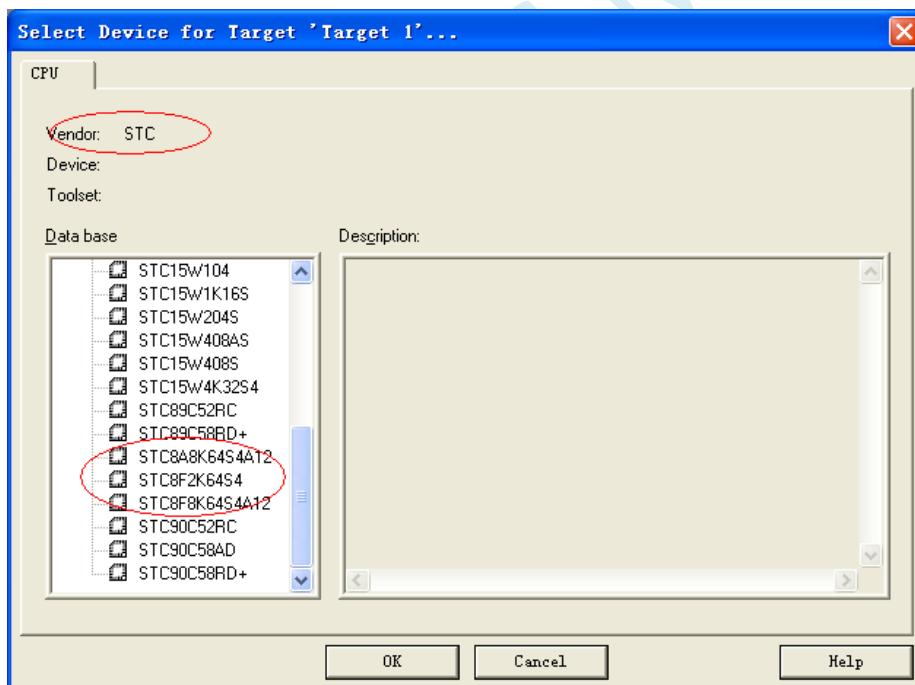


2、在 Keil 中创建项目

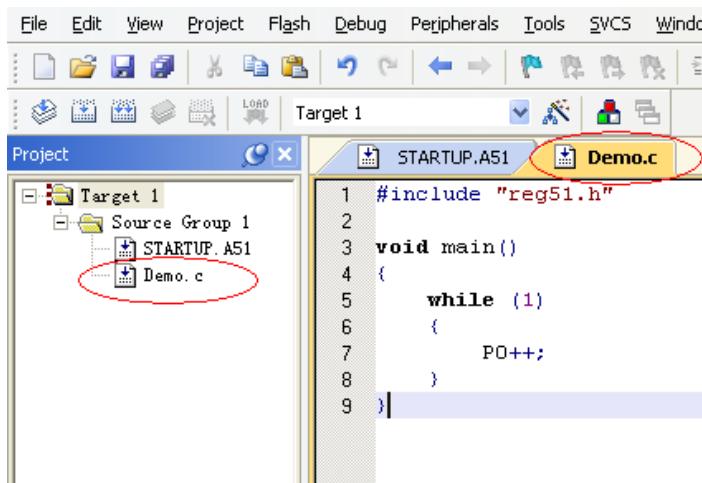
若第一步的驱动安装成功，则在 Keil 中新建项目时选择芯片型号时，便会有“STC MCU Database”的选择项，如下图



然后从列表中选择响应的 MCU 型号，我们在此选择“STC8A8K64S4A12”的型号，点击“确定”完成选择



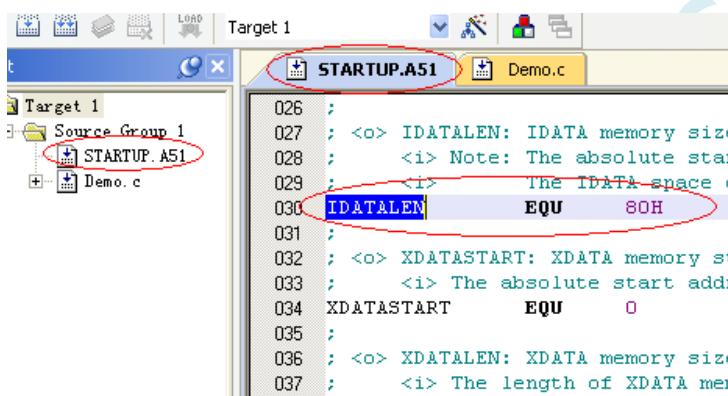
添加源代码文件到项目中，如下图：



保存项目，若编译无误，则可以进行下面的项目设置了

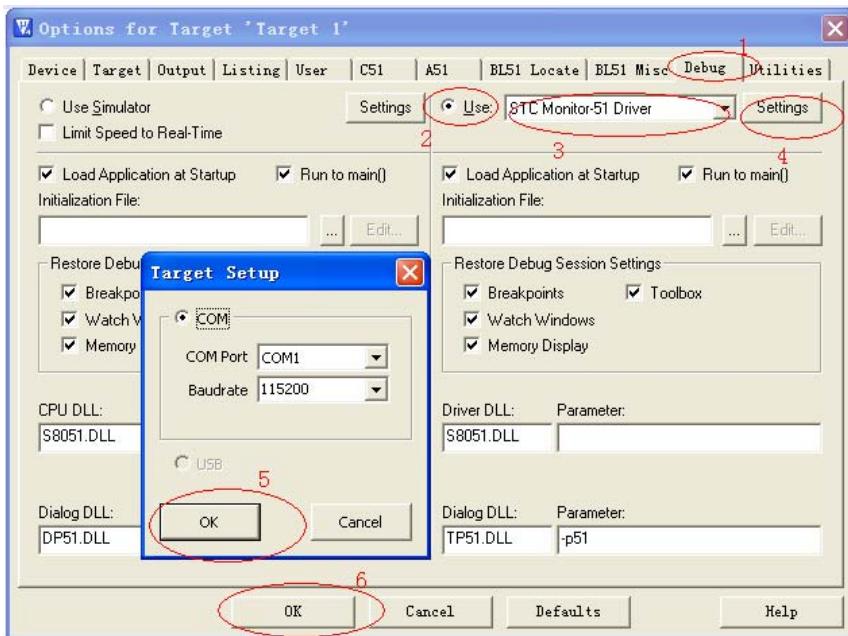
附加说明一点：

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。



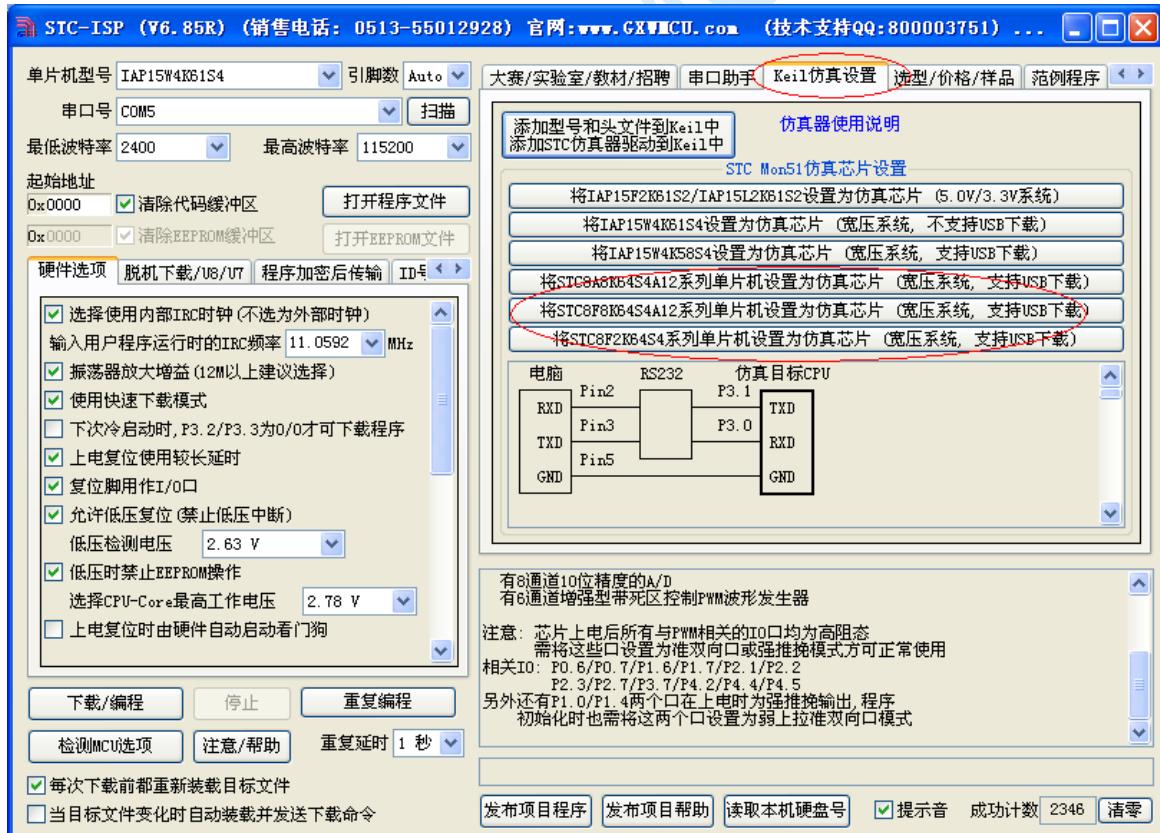
虽然 STC8 系列的单片机的 IDATA 大小为 256 字节 (00-7F 的 DATA 和 80H-FFH 的 IDATA)，但由于在 RAM 的最后 17 个字节有写入 ID 号以及相关的测试参数，若用户在程序中需要使用这一部分数据，则一定不要将 IDATALEN 定义为 256。

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。到此设置便完成了。

4、创建仿真芯片



准备一颗STC8A系列或者STC8F系列的芯片，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，点击相应型号的按钮，当程序下载完成后仿真器便制作完成了。

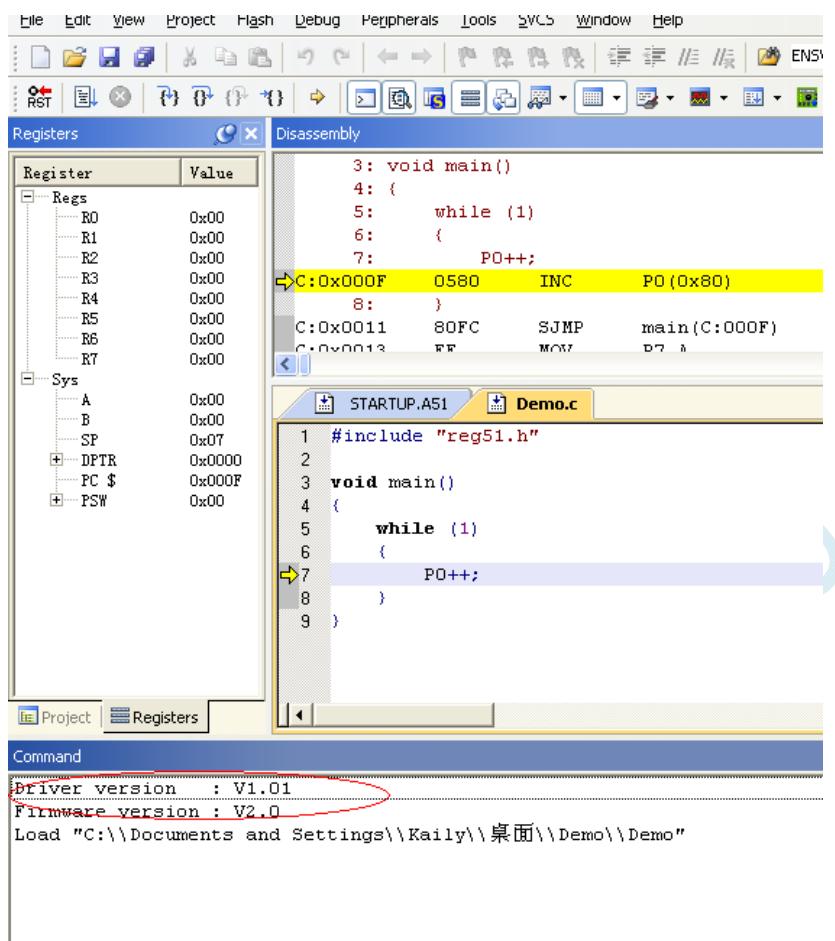
5、开始仿真

将制作完成的仿真芯片通过串口与电脑相连接。

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

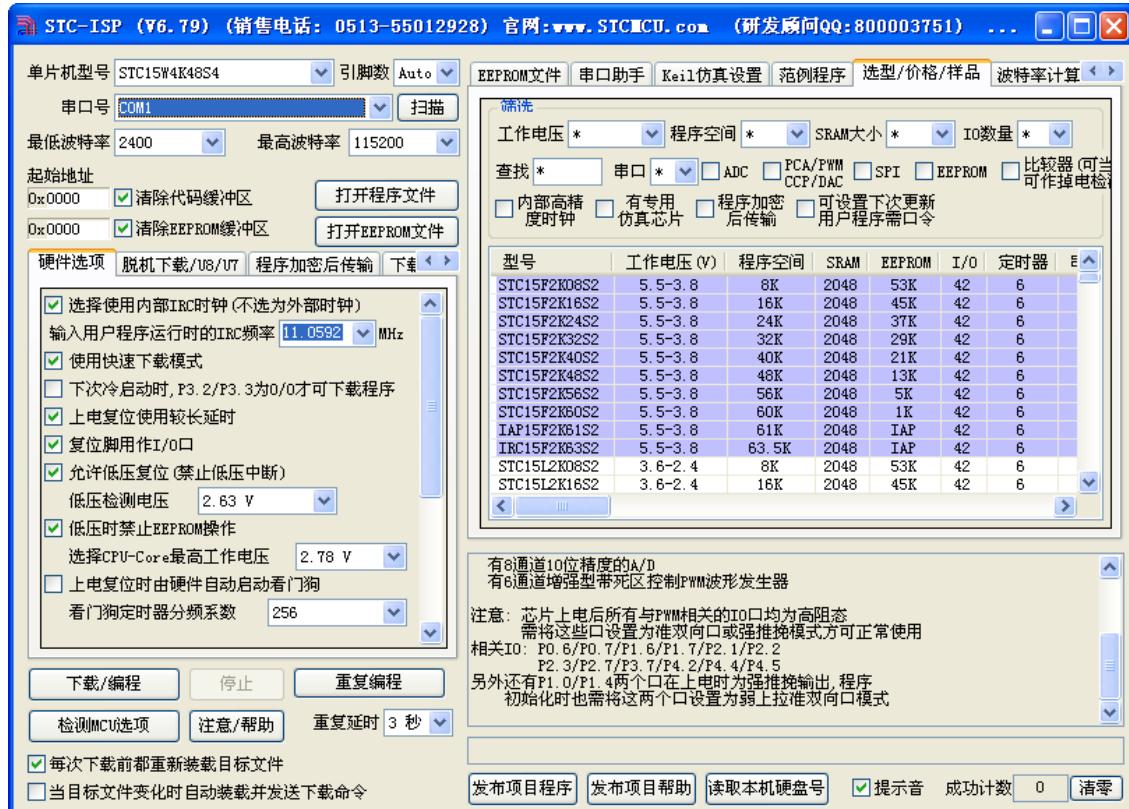
断点设置的个数目前最大允许 20 个（理论上可设置任意个，但是断点设置得过多会影响调试的速度）。



附录C STC-USB驱动程序安装说明

Windows XP安装方法

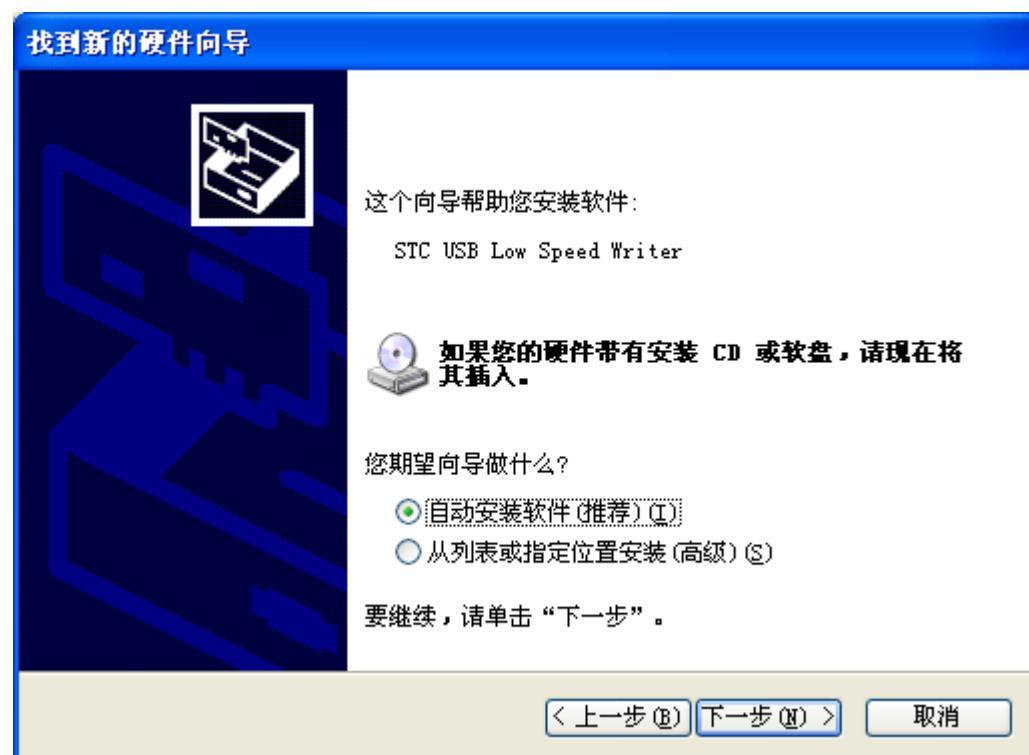
打开 V6.79 版（或者更新的版本）的 STC-ISP 下载软件，下载软件会自动将驱动文件复制到相关的系统目录



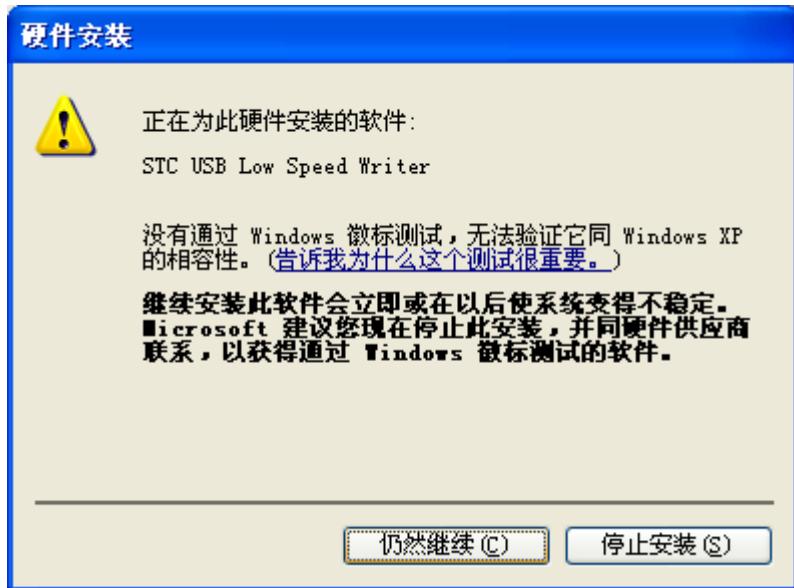
插入 USB 设备，系统找到设备后自动弹出如下对话框，选择其中的“否，暂时不”项



在下面的对话框中选择“自动安装软件(推荐)”项



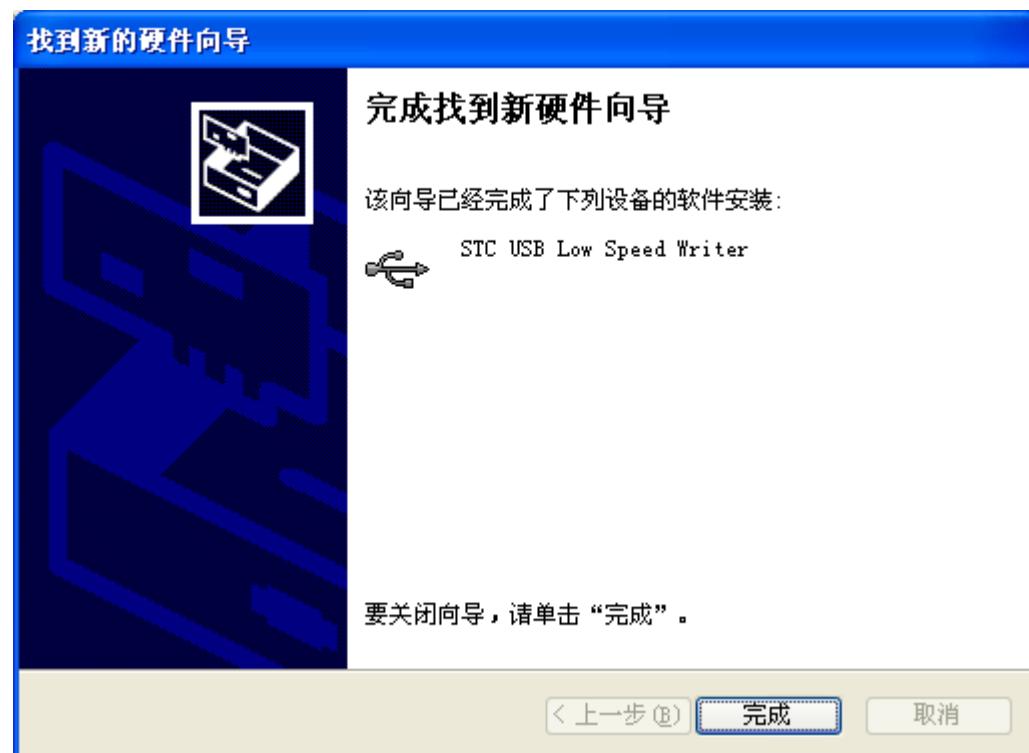
在弹出的下列对话框中，选择“仍然继续”按钮



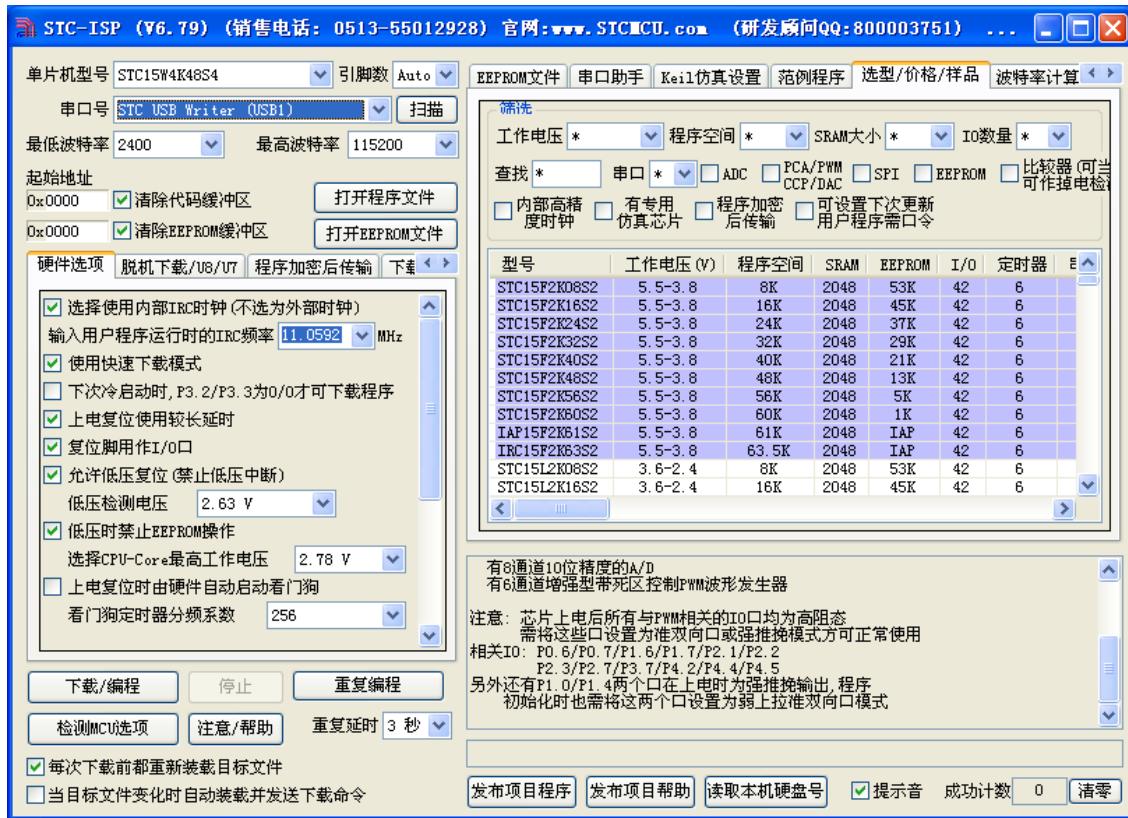
接下系统会自动安装驱动，如下图



出现下面的对话框表示驱动安装完成

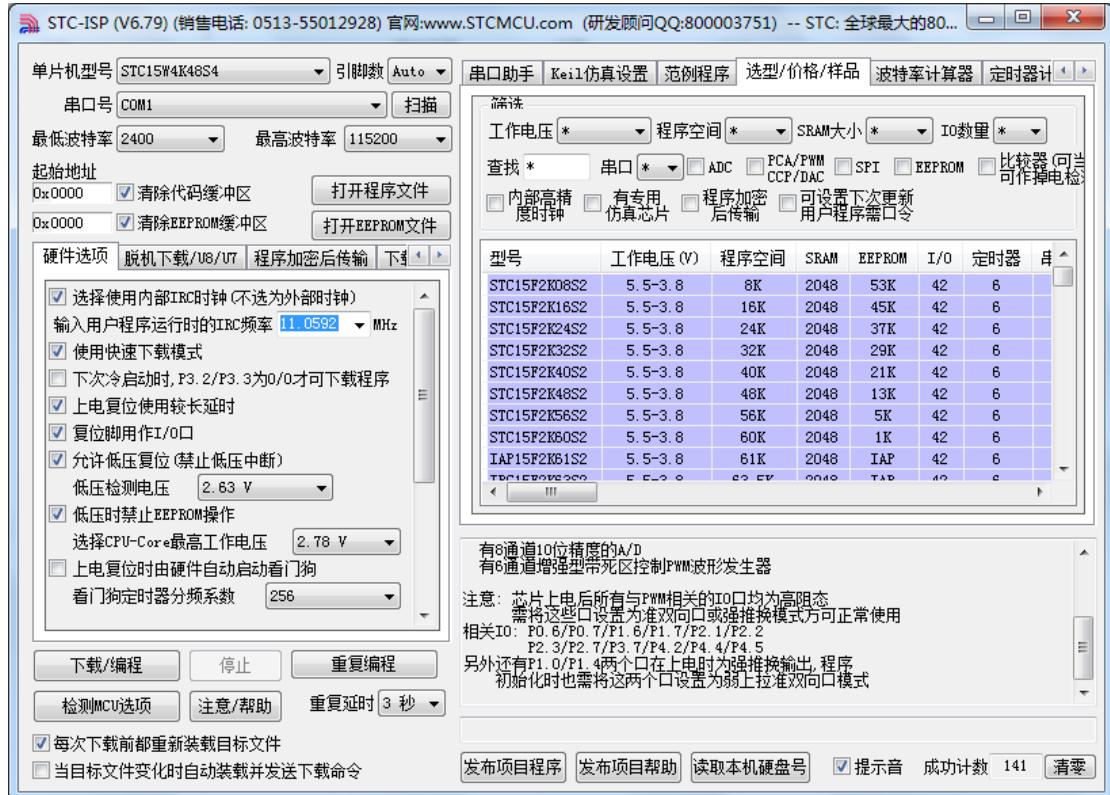


此时，之前打开的 STC-ISP 下载软件中的串口号列表会自动选择所插入的 USB 设备，并显示设备名称为“STC USB Writer (USB1)”，如下图：

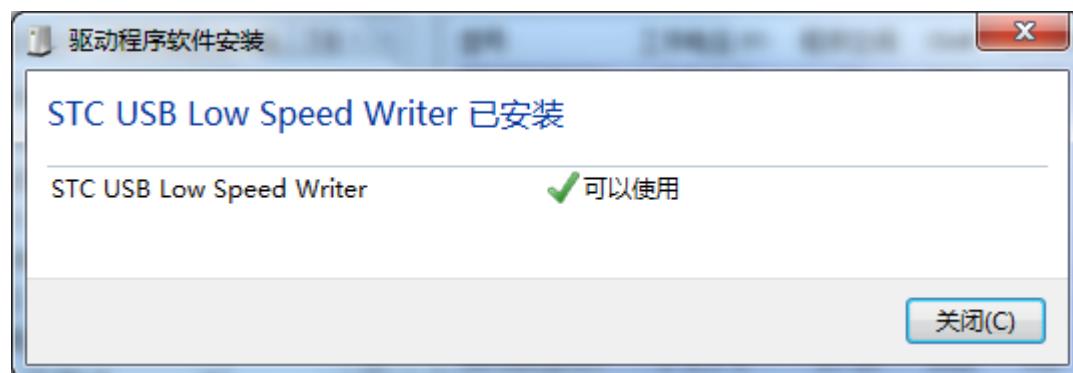


Windows 7 (32位) 安装方法

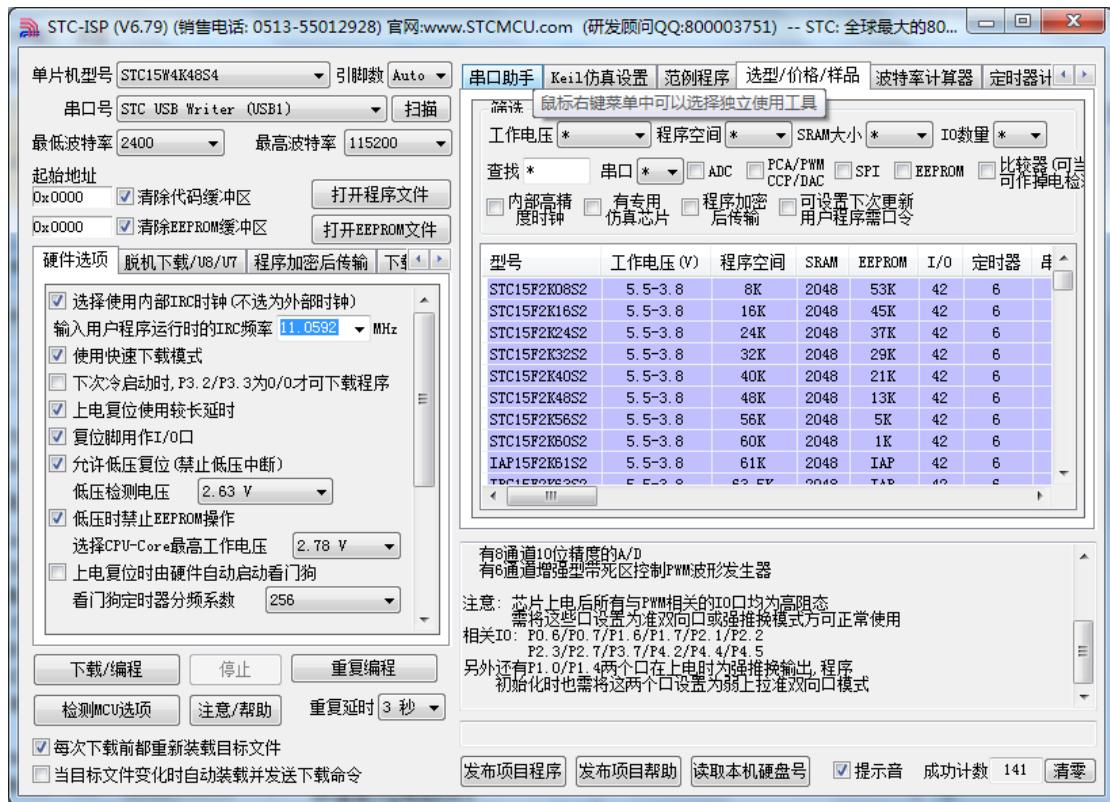
打开 V6.79 版（或者更新的版本）的 STC-ISP 下载软件，下载软件会自动将驱动文件复制到相关的系统目录



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：

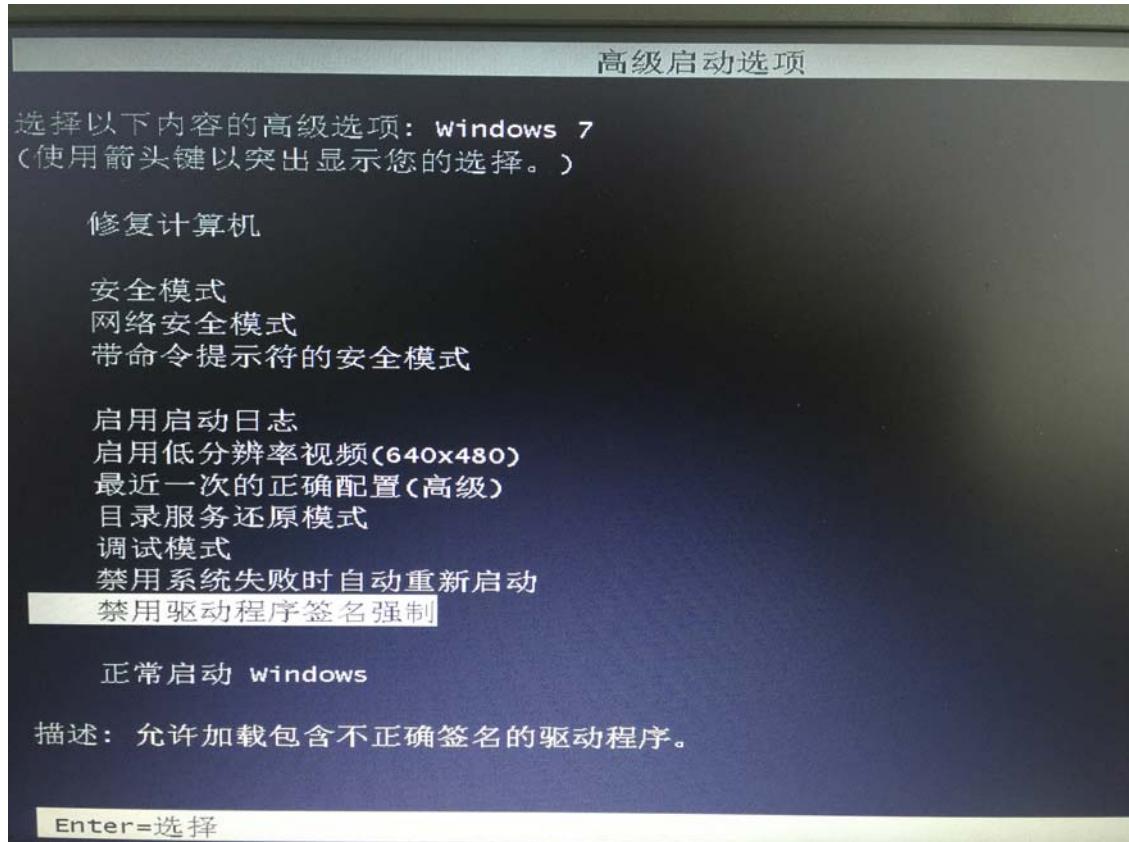


注：若 Windows 7 下，系统并没有自动安装驱动，则驱动的安装方法请参考 [Windows 8 \(32 位\) 的安装方法](#)

Windows 7 (64 位) 安装方法

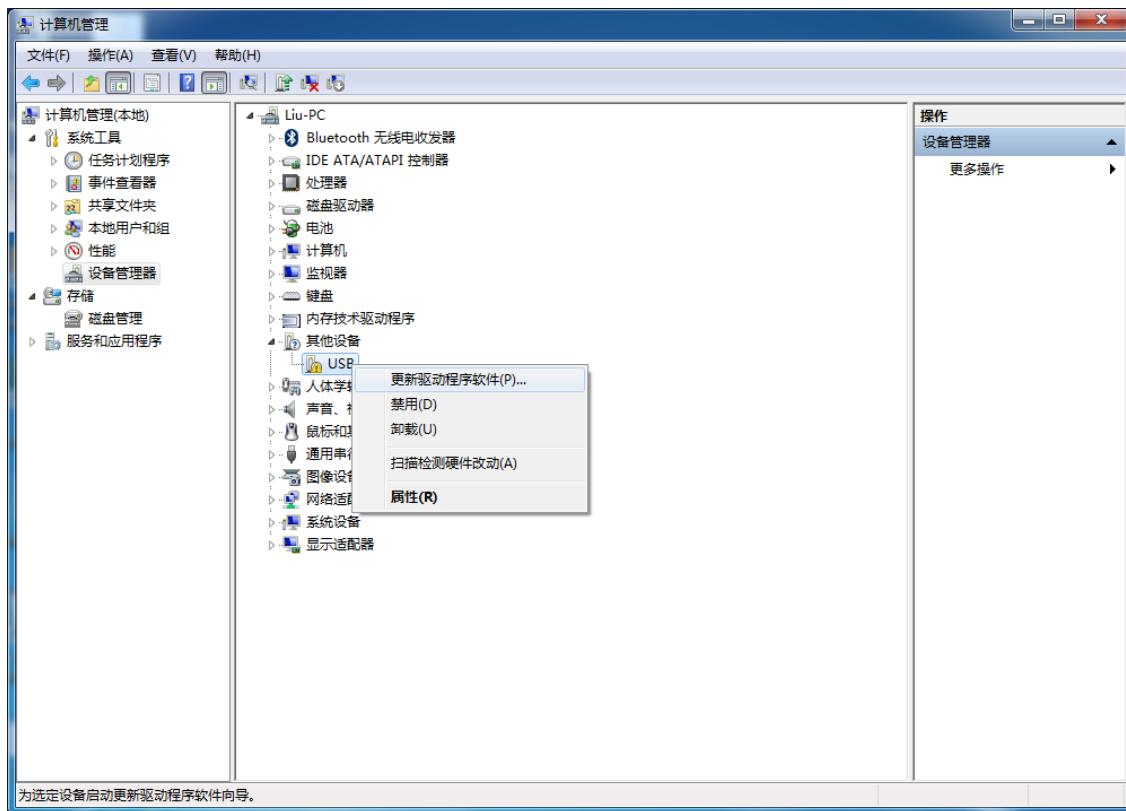
由于 Windows7 64 位操作系统在默认状态下，对于没有数字签名的驱动程序是不能安装成功的。所以在安装 STC-USB 驱动前，需要按照如下步骤，暂时跳过数字签名，即可顺利安装成功。

首先重启电脑，并一直按住 F8，直到出现下面启动画面

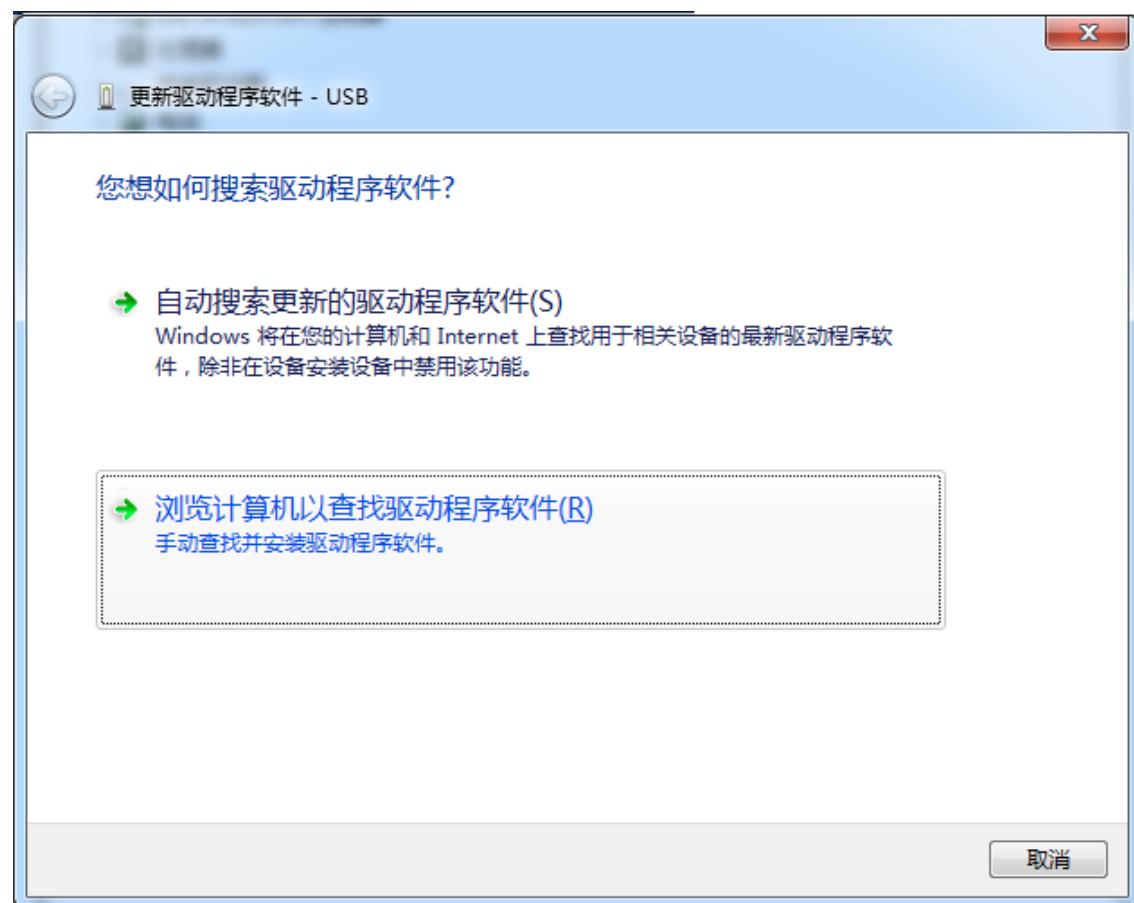


选择“禁用驱动程序签名强制”。启动后即可暂时关闭数字签名验证功能

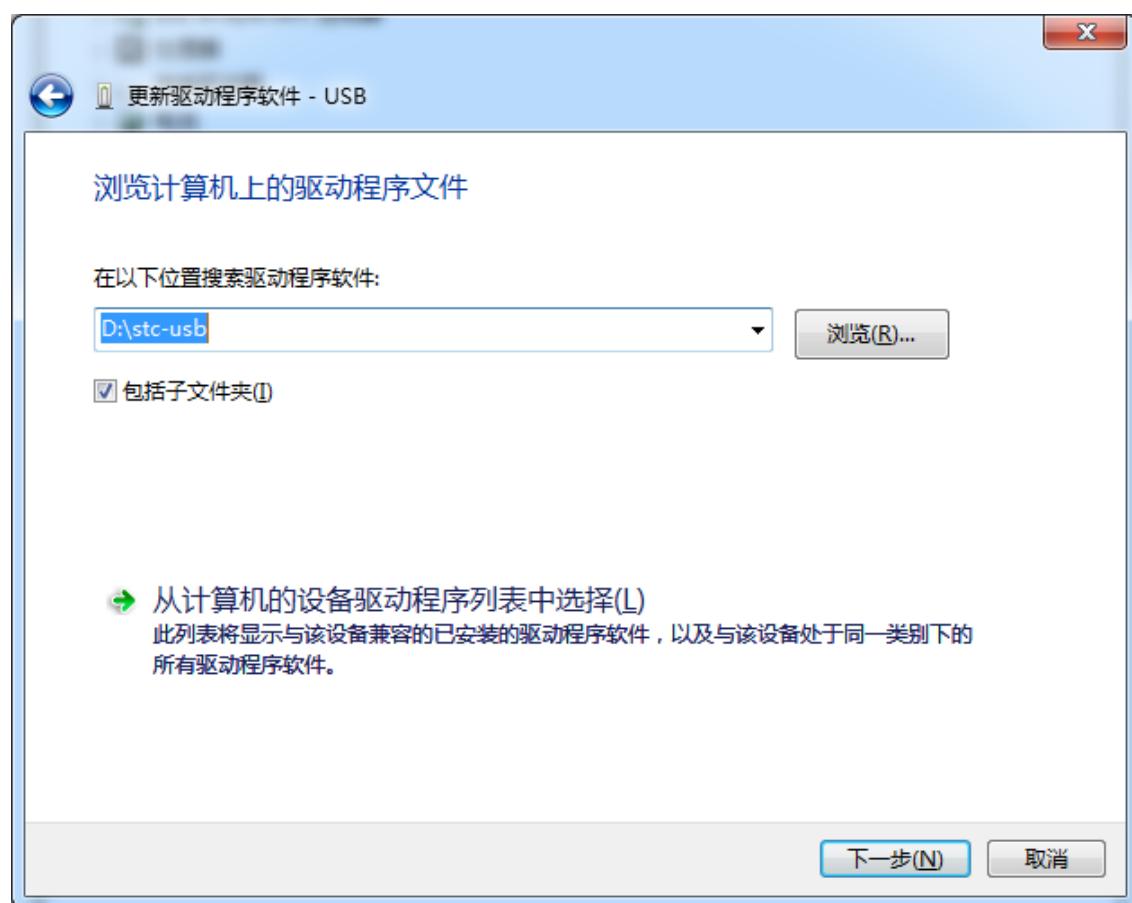
插入 USB 设备，并打开“设备管理器”。找到设备列表中带黄色感叹号的 USB 设备，在设备的右键菜单中，选择“更新驱动程序软件”



在下面的对话框中选择“浏览计算机以查找驱动程序软件”



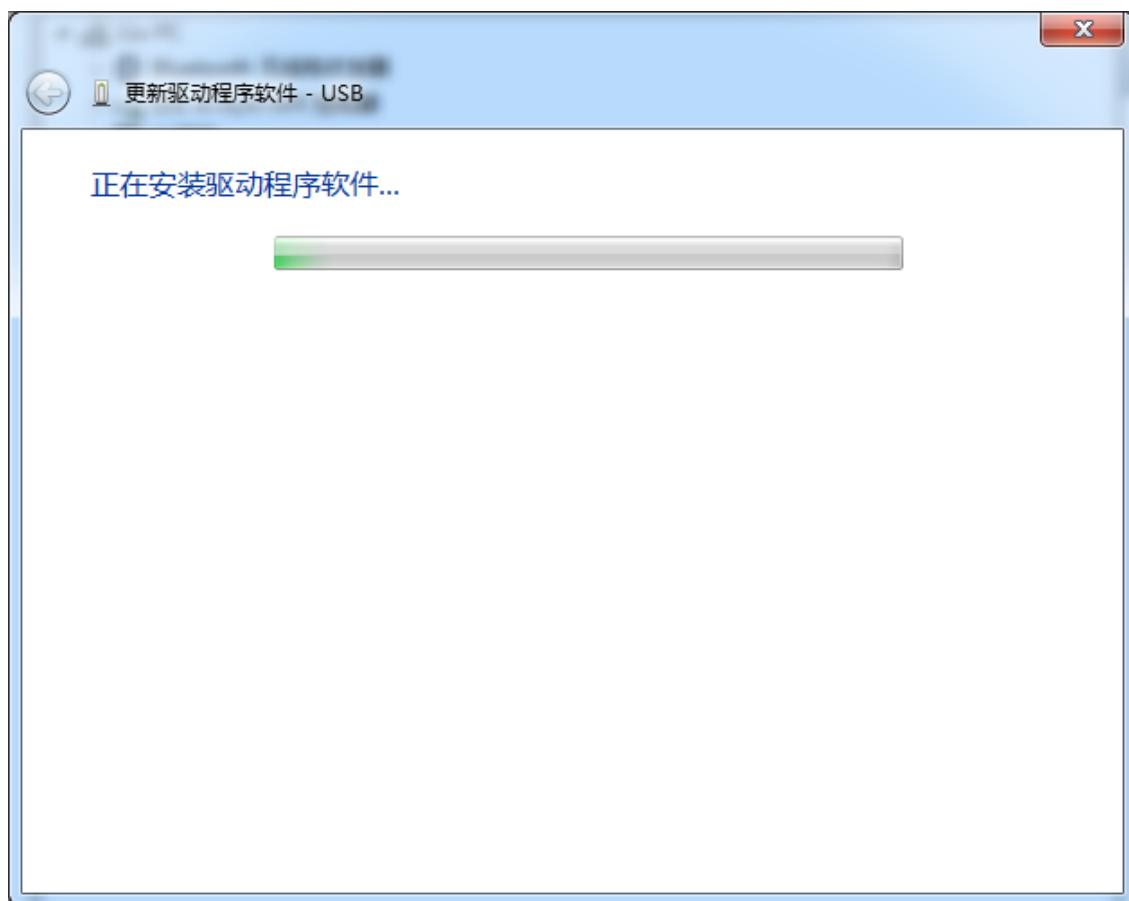
单击下面对话框中的“浏览”按钮，找到之前 STC-USB 驱动程序的存放目录（例如：之前的示例目录为 “ D:\STC-USB ” ， 用户将路径定位到实际的解压目录）



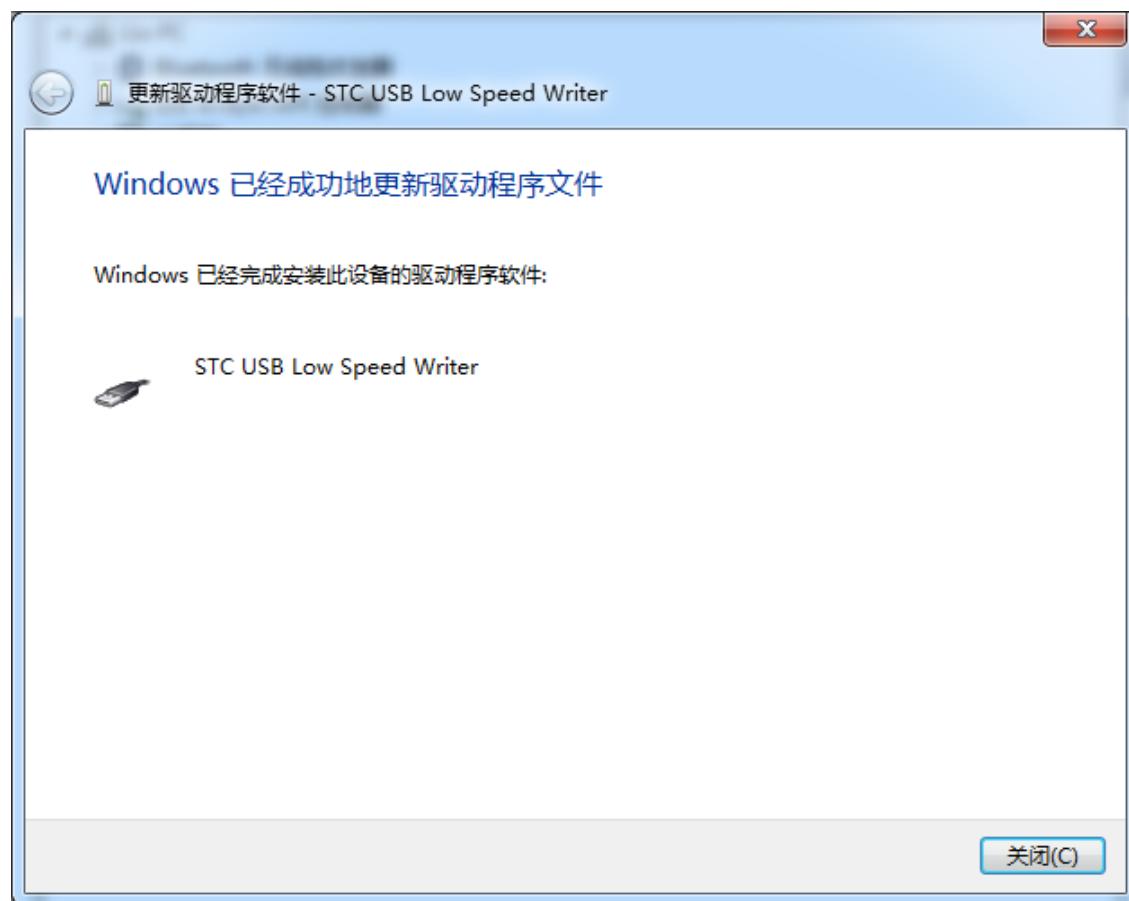
驱动程序开始安装时，会弹出如下对话框，选择“始终安装此驱动程序软件”



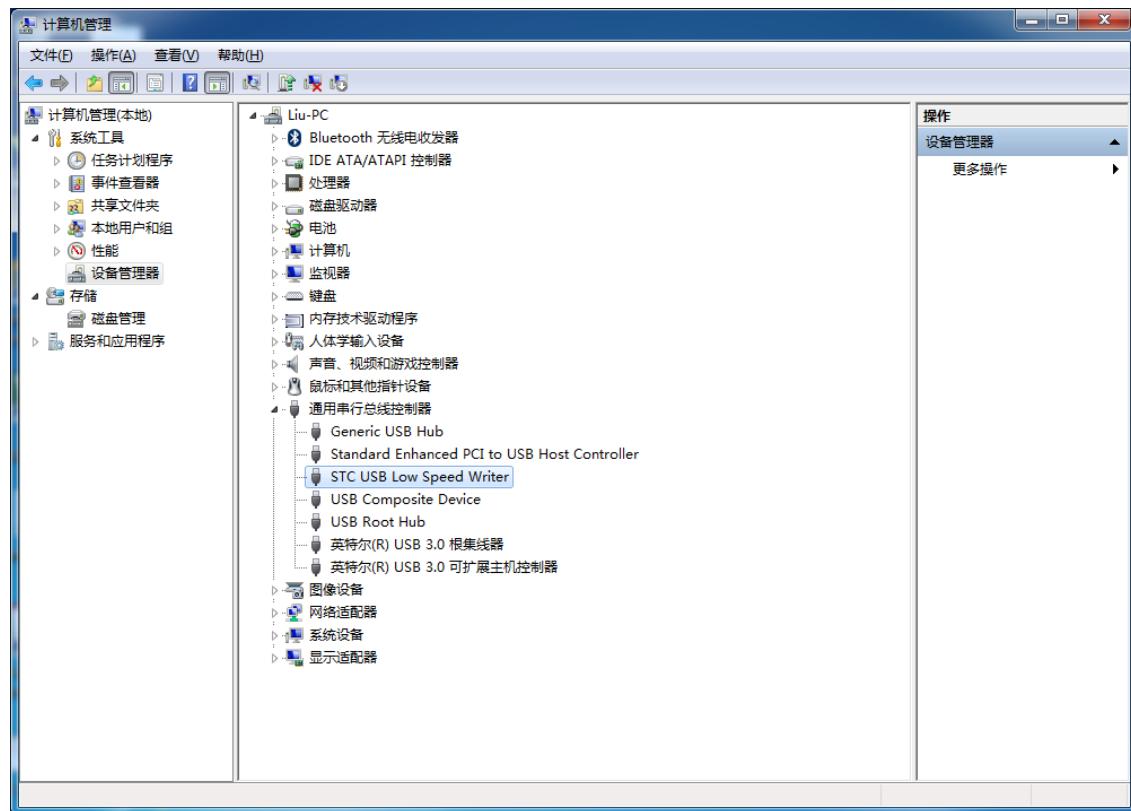
接下来，系统会自动安装驱动，如下图



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



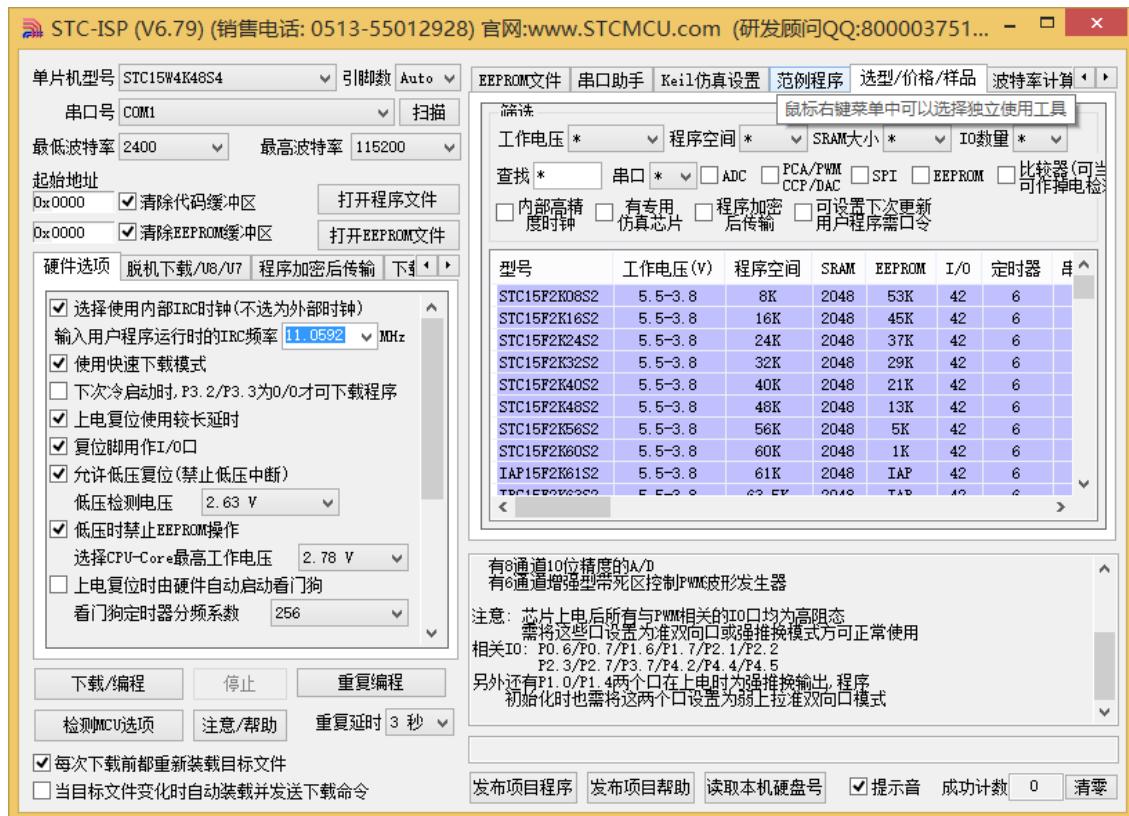
在之前打开的 STC-ISP 下载软件中的串口号列表会自动选择所插入的 USB 设备，并显示设备名称为“STC USB Writer (USB1)”，如下图：



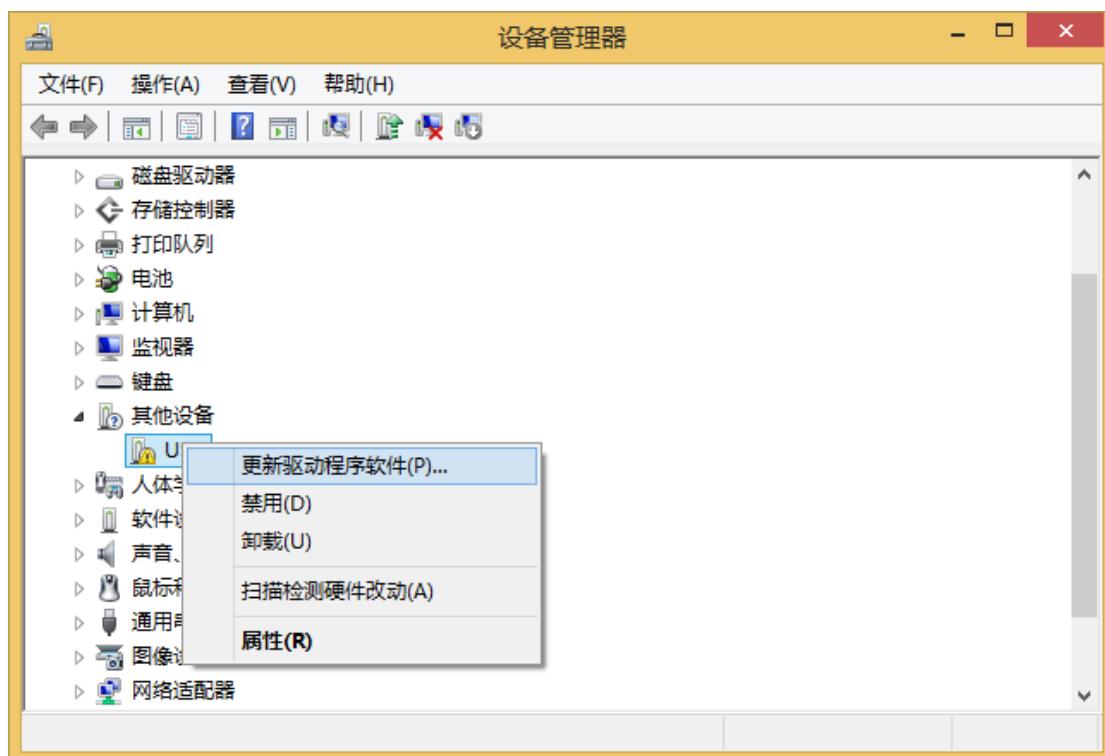
Windows 8 (32位) 安装方法

打开 V6.79 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

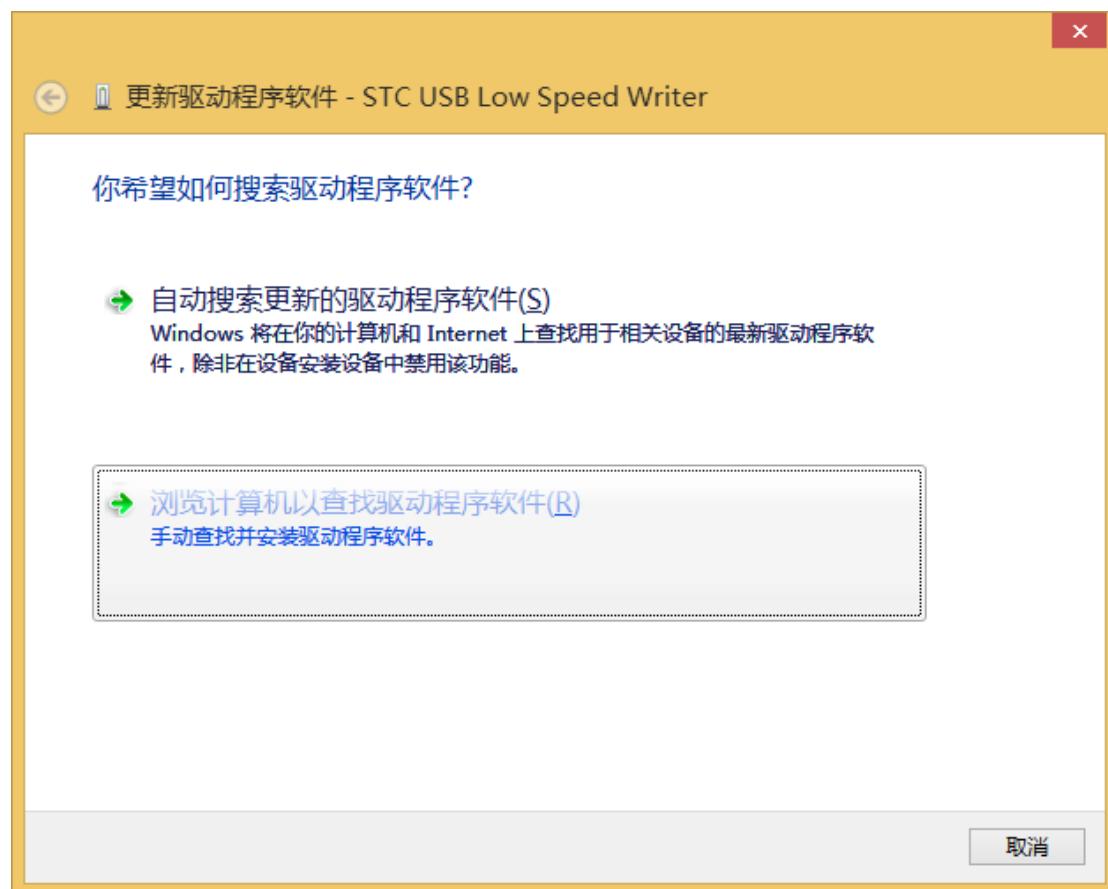
“stc-isp-15xx-v6.79.zip”（或更新版本），下载后解压到本地磁盘，则 STC-USB 的驱动文件也会被解压到当前解压目录中的“STC-USB Driver”中（例如将下载的压缩文件“stc-isp-15xx-v6.79.zip”解压到“F:\”，则 STC-USB 驱动程序在“F:\STC-USB Driver”目录中）



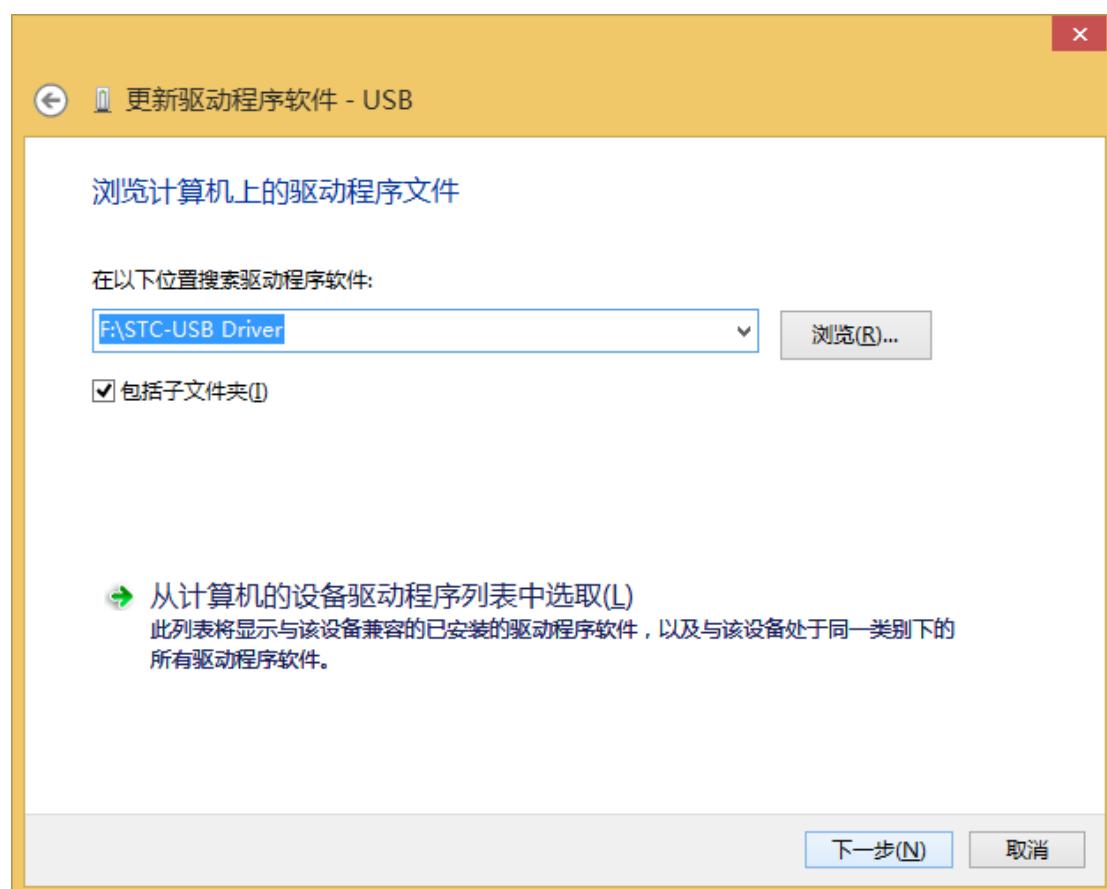
插入 USB 设备，并打开“设备管理器”。找到设备列表中带黄色感叹号的 USB 设备，在设备的右键菜单中，选择“更新驱动程序软件”



在下面的对话框中选择“浏览计算机以查找驱动程序软件”



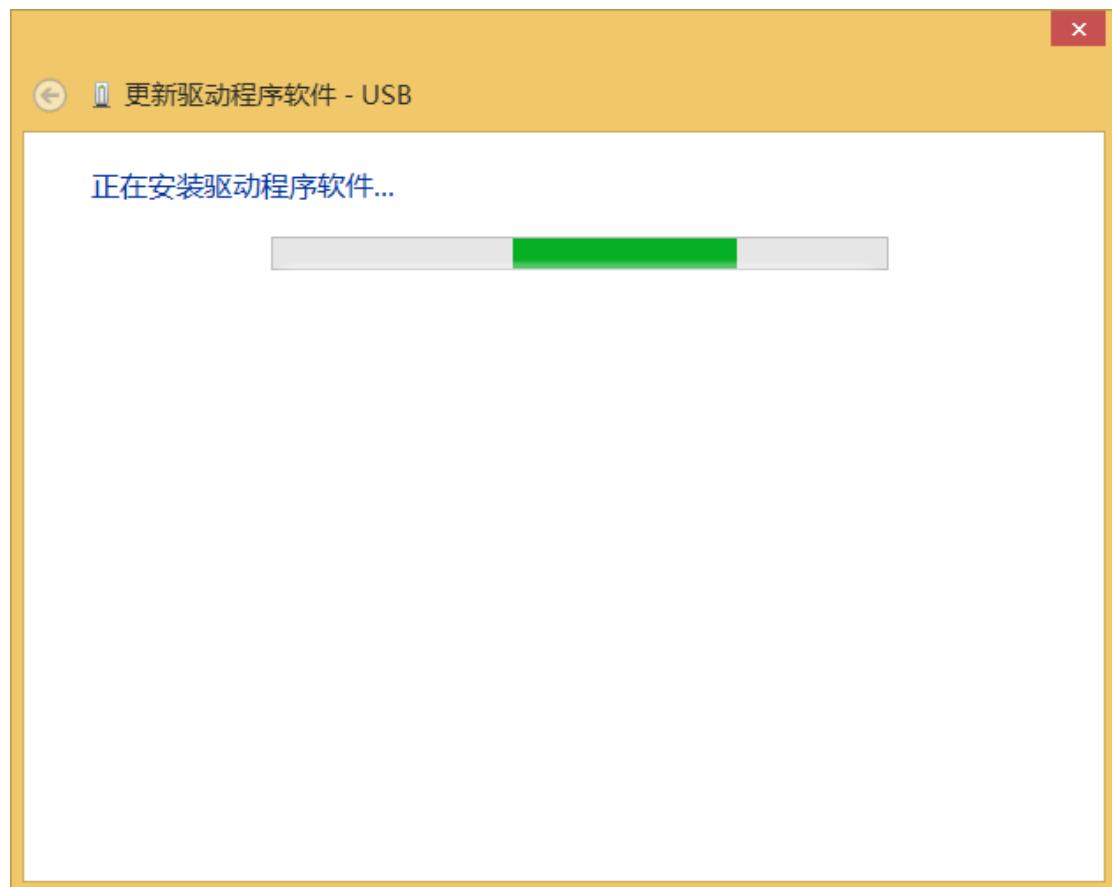
单击下面对话框中的“浏览”按钮，找到之前 STC-USB 驱动程序的存放目录（例如：之前的示例目录为“F:\STC-USB Driver”，用户将路径定位到实际的解压目录）



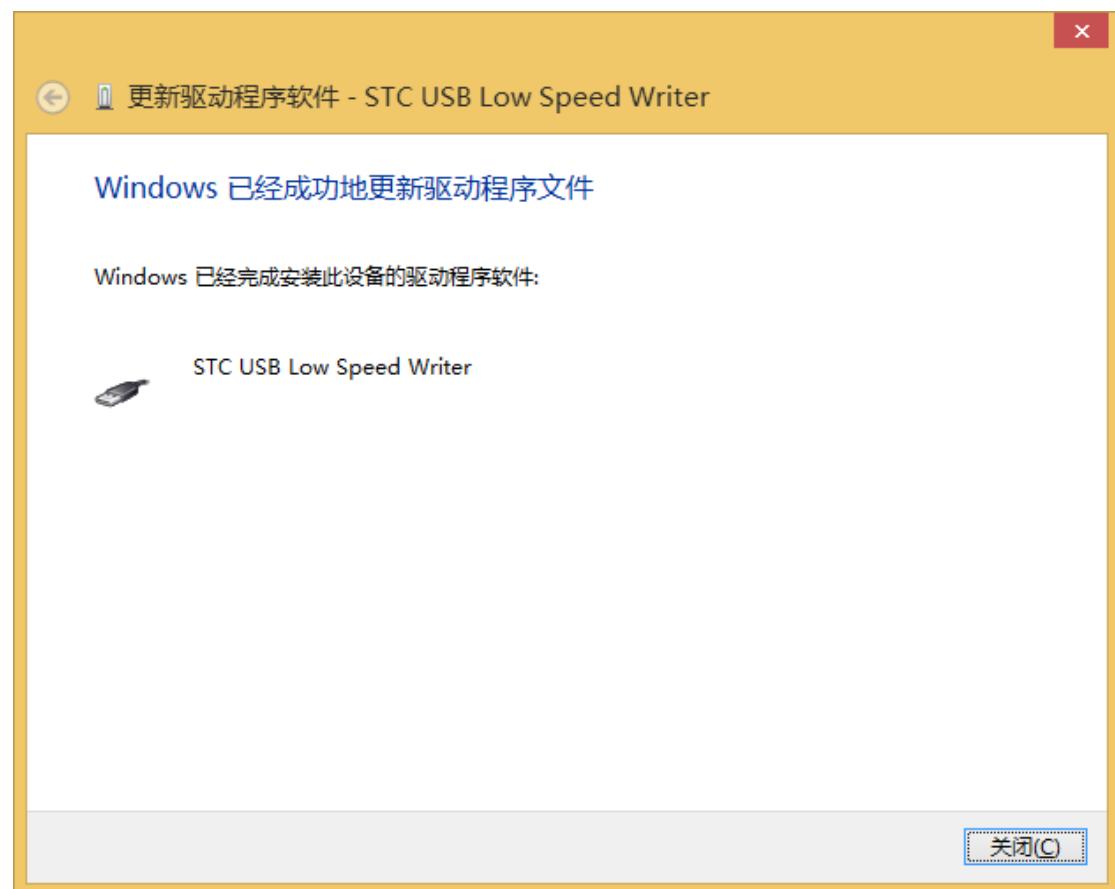
驱动程序开始安装时，会弹出如下对话框，选择“始终安装此驱动程序软件”



接下来，系统会自动安装驱动，如下图



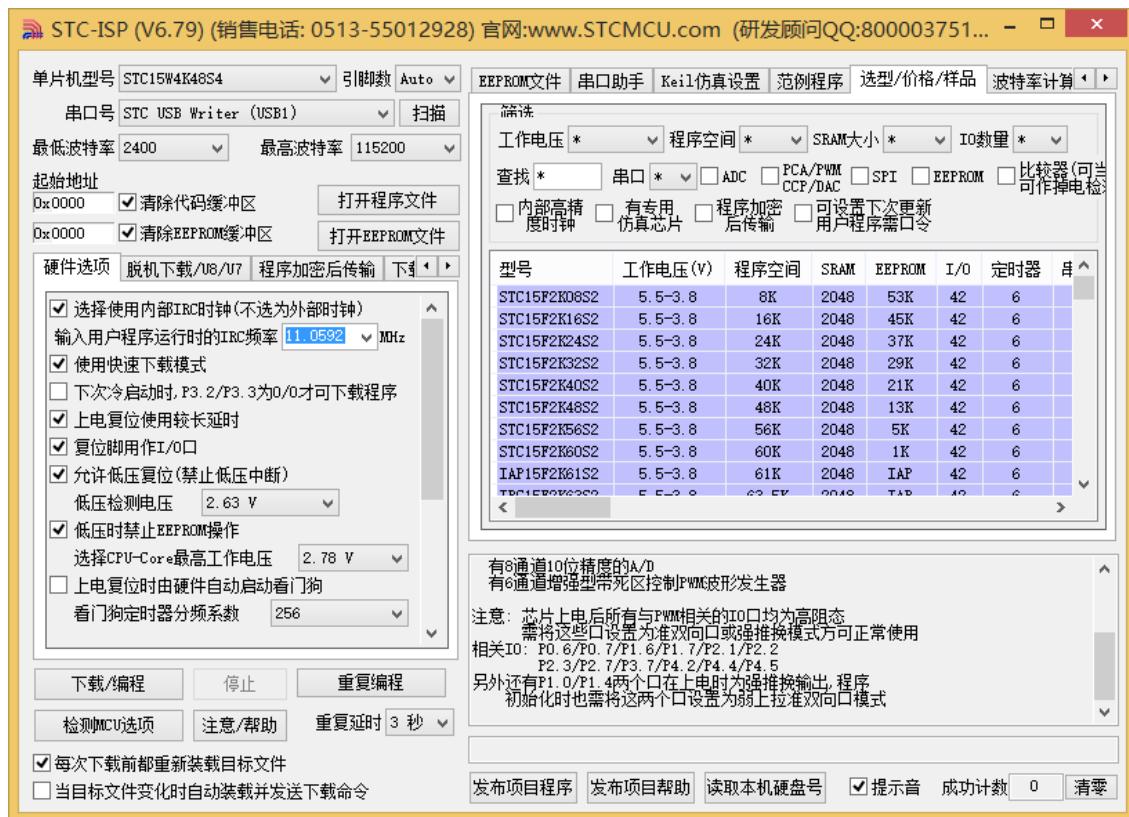
出现下面的对话框表示驱动安装完成



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：



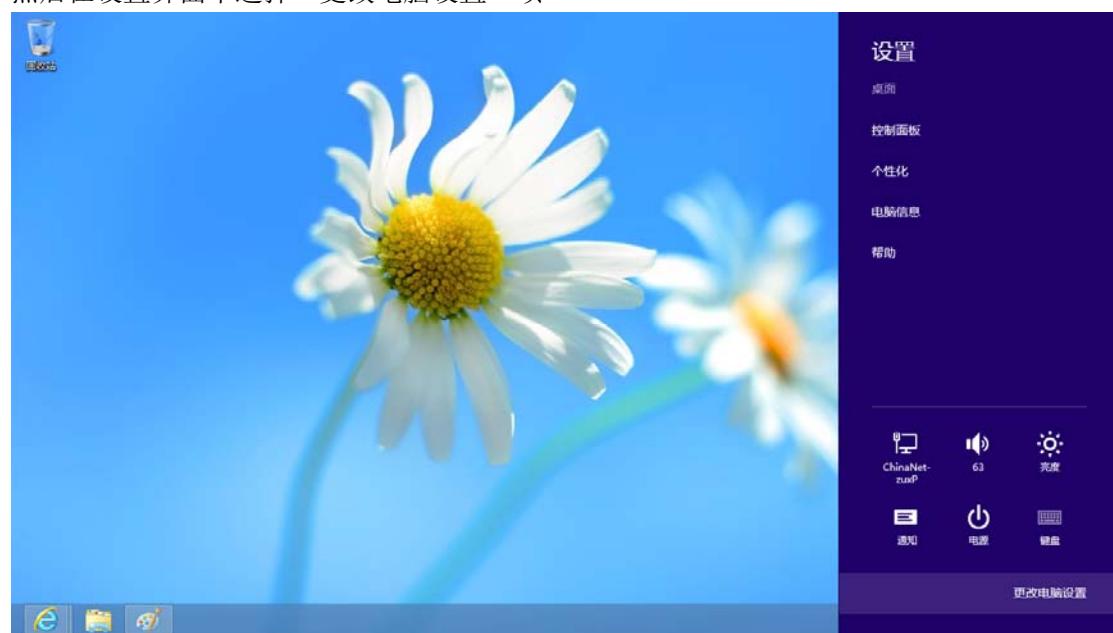
Windows 8 (64 位) 安装方法

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。

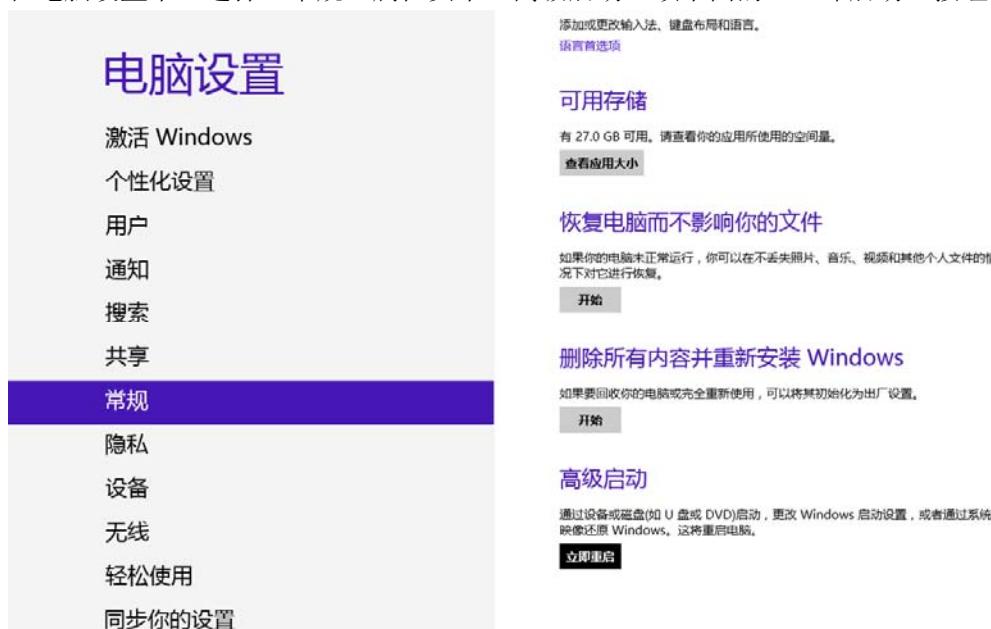
首先将鼠标移动到屏幕的右下角，选择其中的“设置”按钮



然后在设置界面中选择“更改电脑设置”项



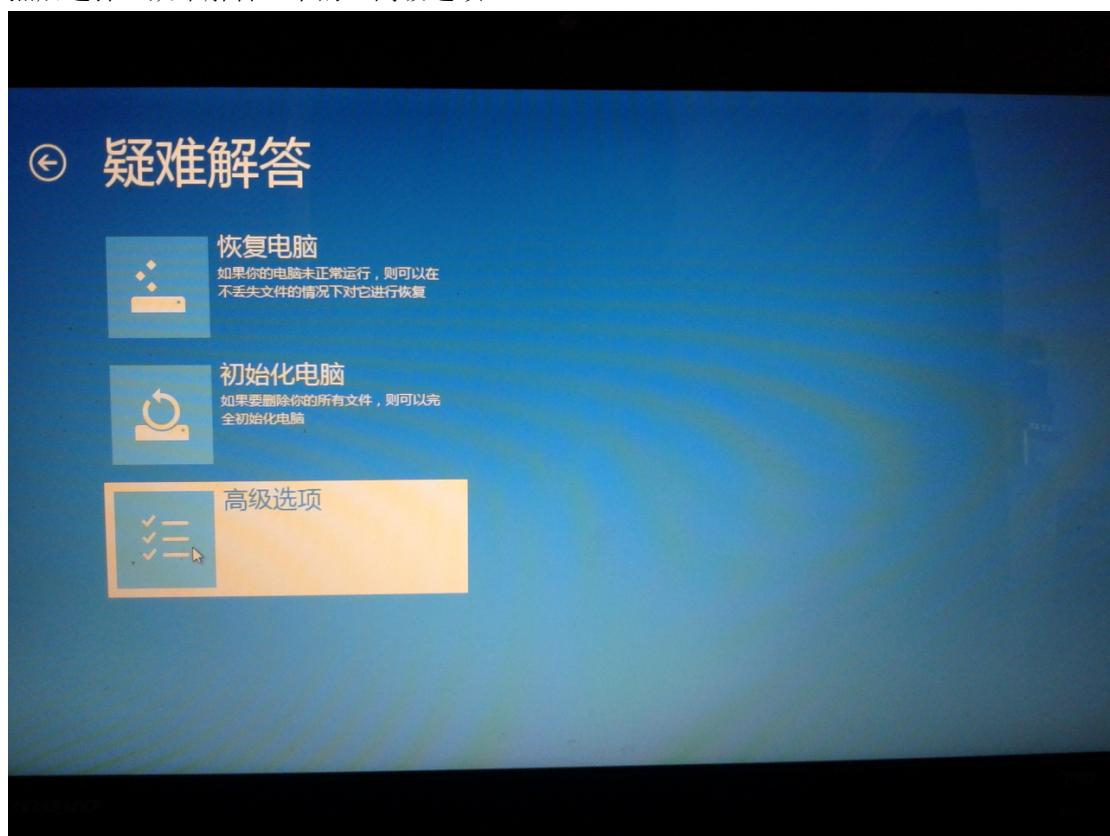
在电脑设置中, 选择“常规”属性页中“高级启动”项下面的“立即启动”按钮



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”



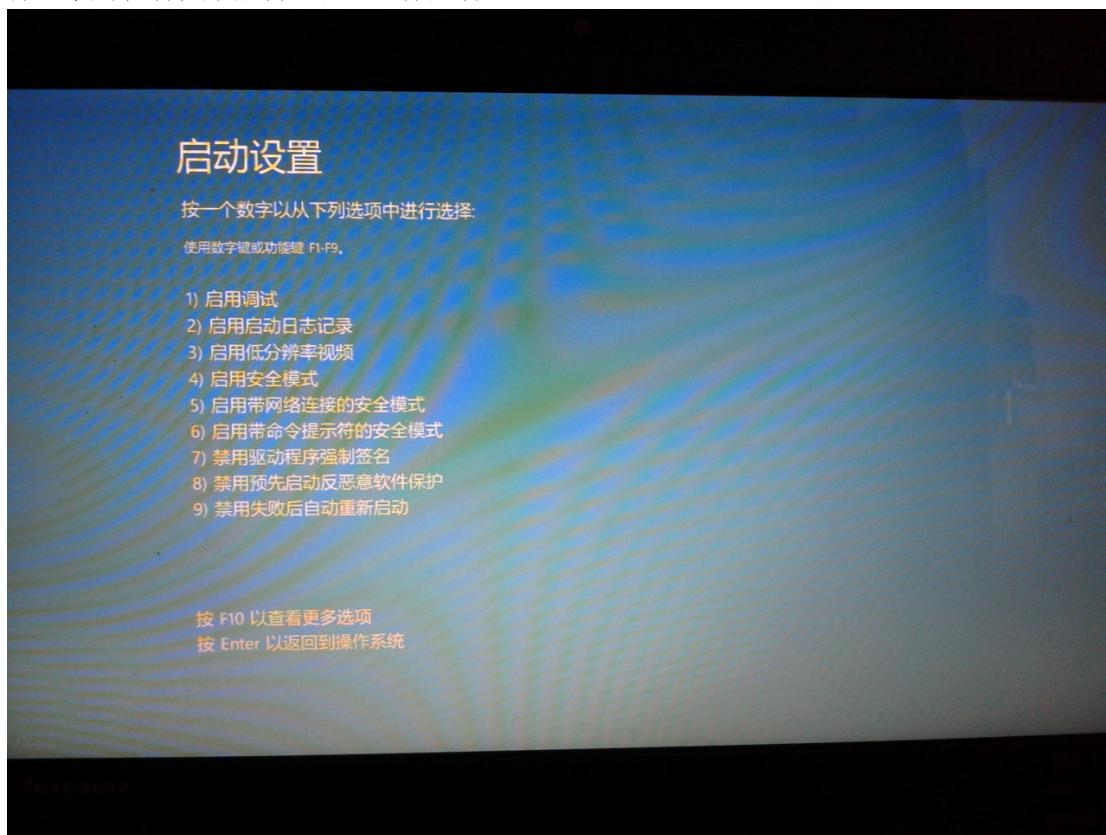
在下面的“高级选项”界面中，选择“启动设置”



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 \(32 位\) 的安装方法](#)即可完成驱动的安装

Windows 8.1 (64 位) 安装方法

Windows 8.1 与 Windows 8 进入高级启动菜单的方法不一样,在此专门进行说明。

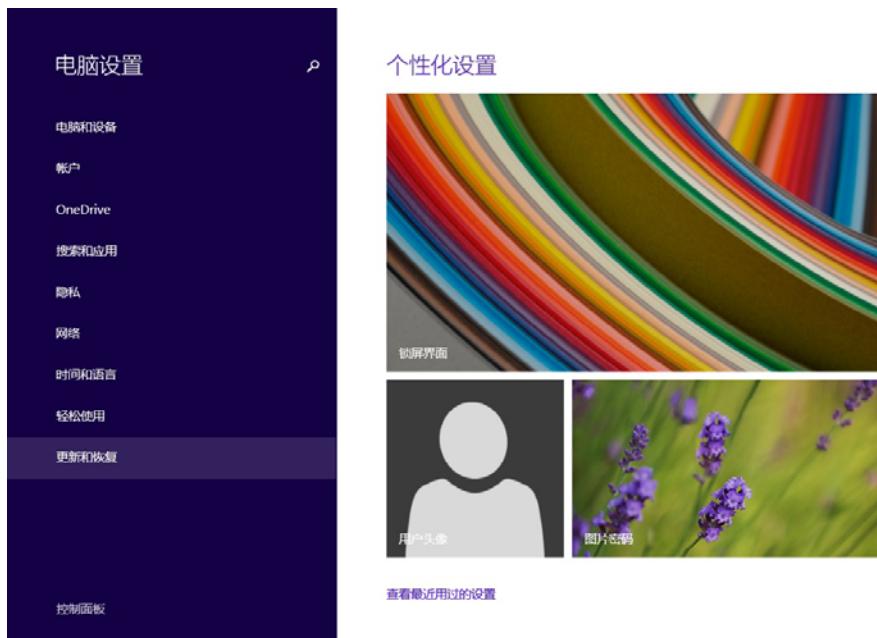
首先将鼠标移动到屏幕的右下角, 选择其中的“设置”按钮



然后在设置界面中选择“更改电脑设置”项



在电脑设置中, 选择“更新和恢复”(这里与 Windows 8 不一样, Windows 8 选择的是“常规”)



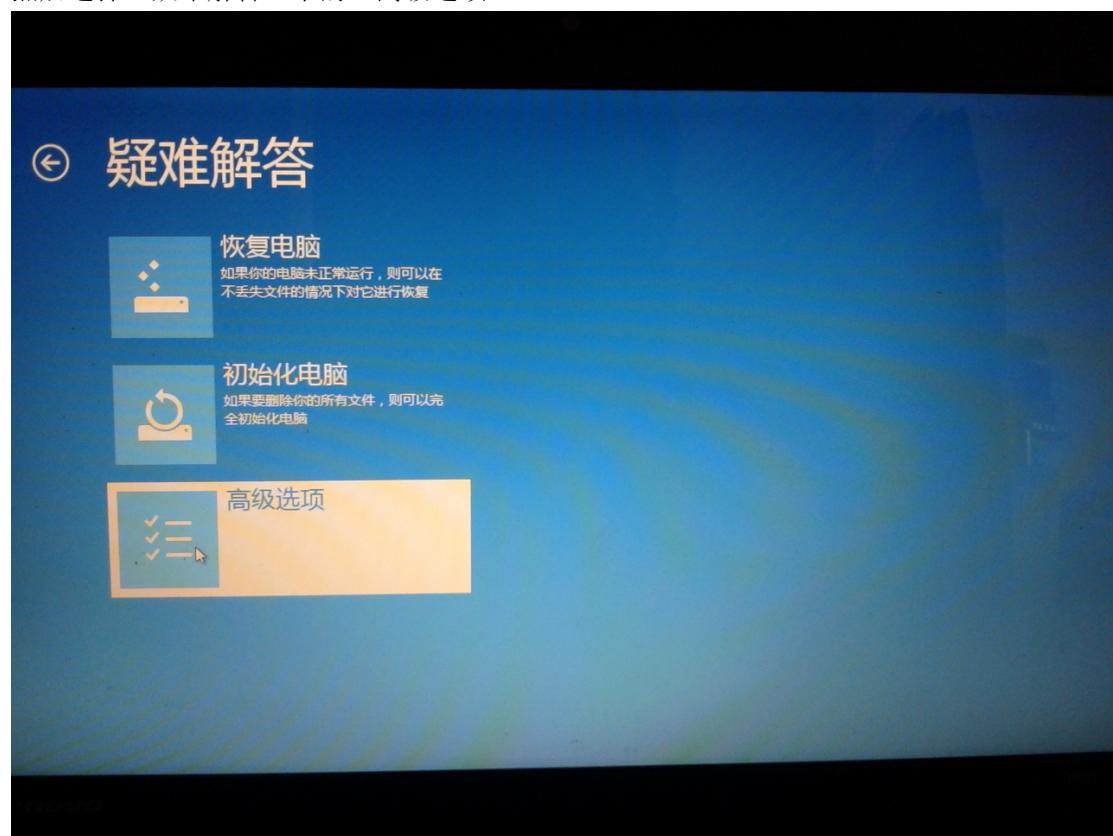
在更新和恢复页面中选择“恢复”属性页，单击“高级启动”项下面的“立即启动”按钮



接下来的操作与 Window 8 的步骤相同
在下面的界面中, 选择“**疑难解答**”项



然后选择“疑难解答”中的“高级选项”



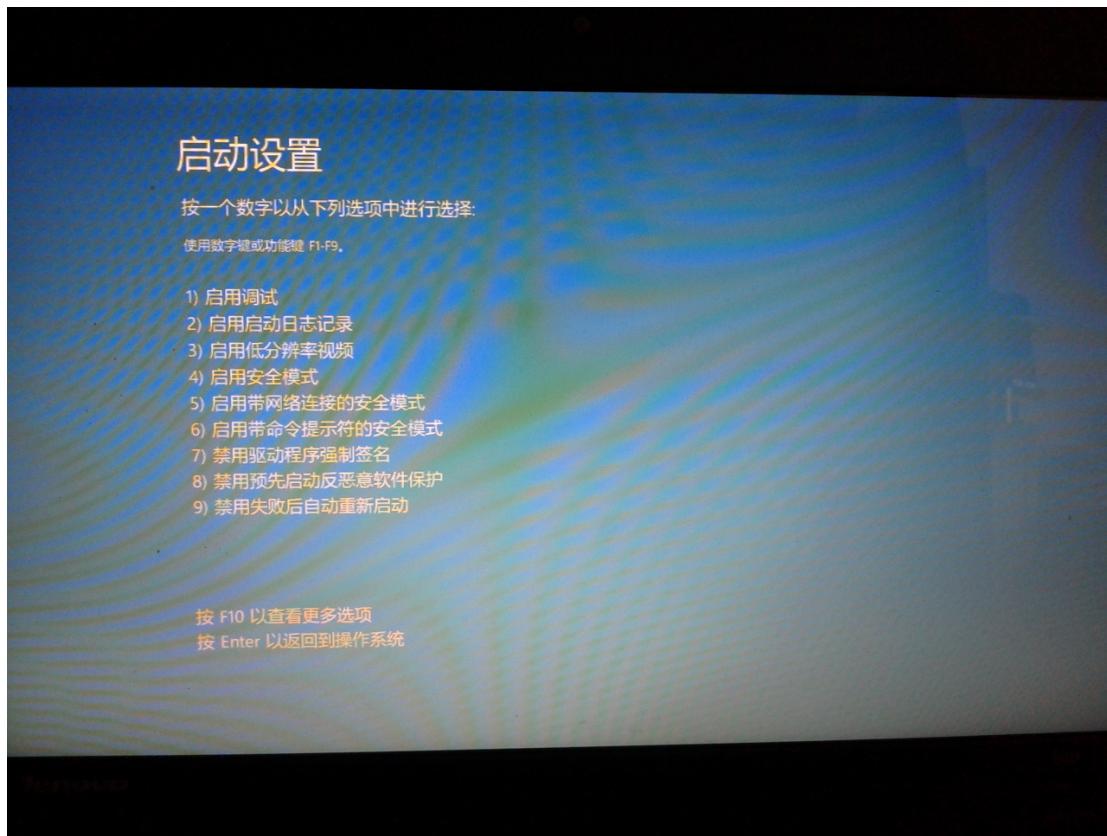
在下面的“高级选项”界面中，选择“启动设置”



在下面的“启动设置”界面中，单击“重启”按钮对电脑进行重新启动



在电脑重新启动后会自动进入如下图所示的“启动设置”界面，按数字键“7”或者按功能键“F7”选择“禁用驱动程序强制签名”进行启动



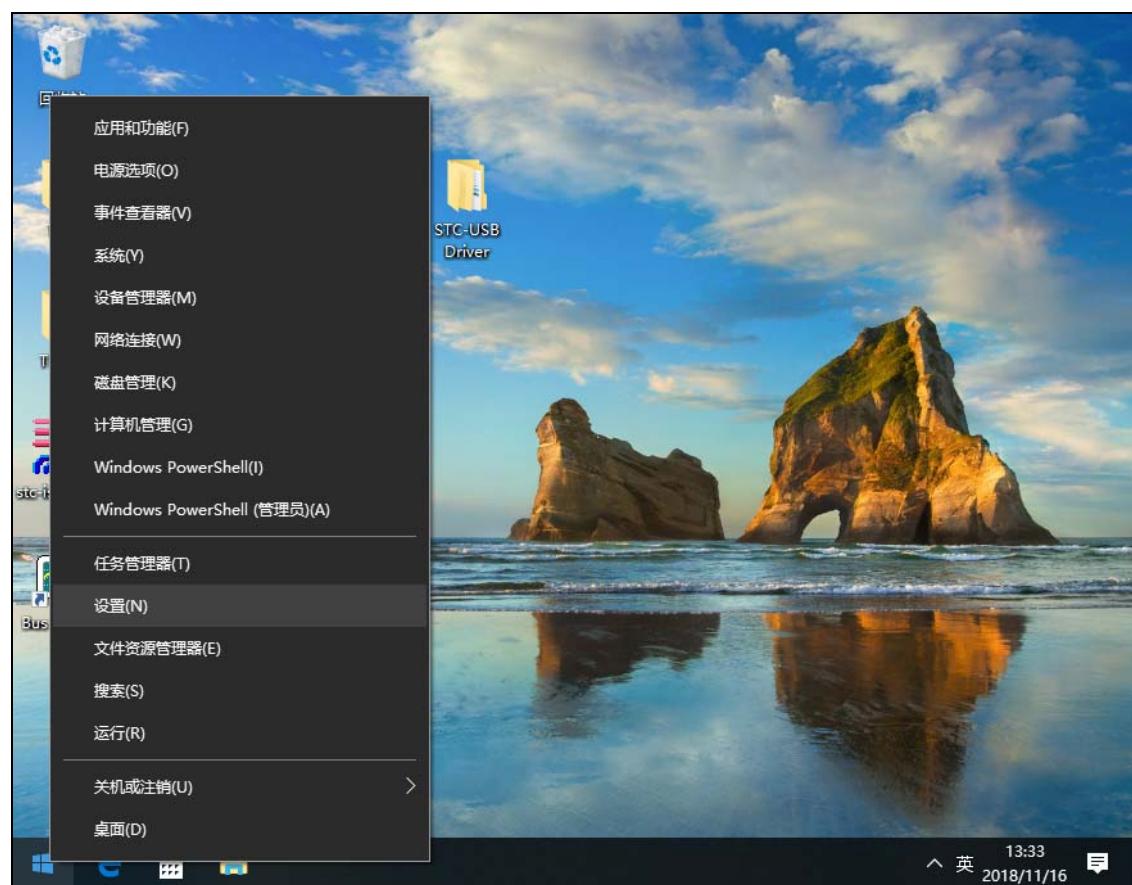
启动到 Windows 8 后，按照 [Windows 8 \(32 位\) 的安装方法](#)即可完成驱动的安装

Windows10 (64 位) 安装方法

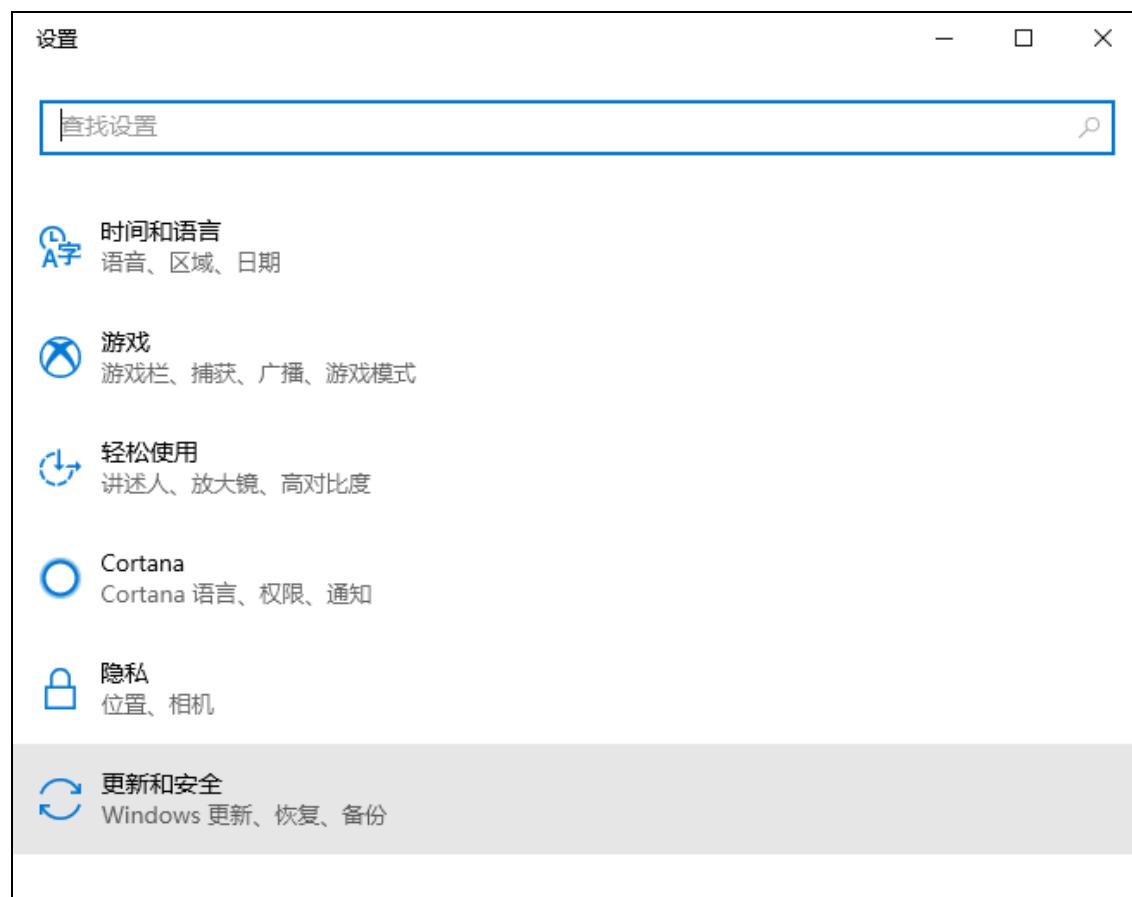
由于 Windows10 64 位操作系统在默认状态下，对于没有数字签名的驱动程序是不能安装成功的。所以在安装 STC-USB 驱动前，需要按照如下步骤，暂时跳过数字签名，即可顺利安装成功。

安装驱动前需要从 STC 官网下载的 STC-ISP 下载软件压缩包中将“STC-USB Driver”文件夹解压缩到硬盘中。将具有 USB 下载功能的芯片准备好，但先不要连接电脑

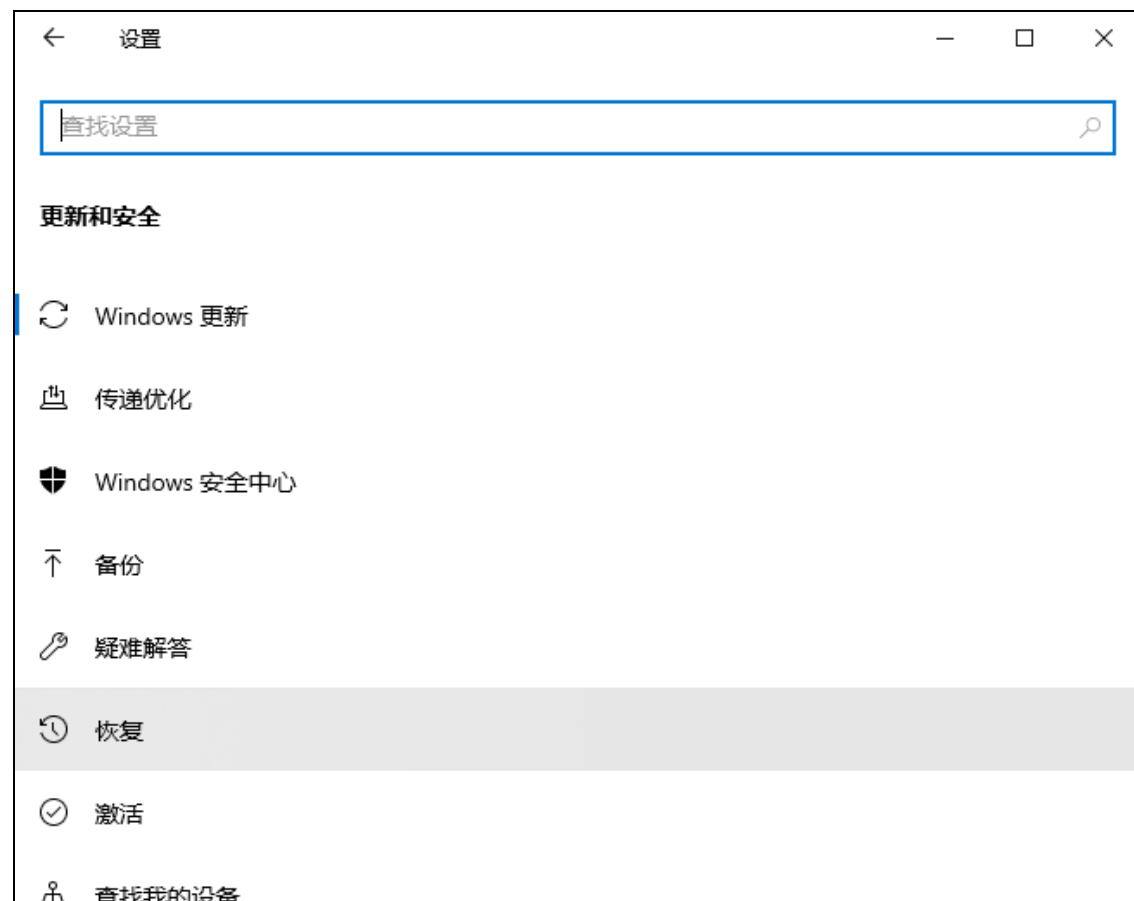
鼠标右键点击“开始”菜单，选择“设置”选项



然后在设置界面中选择“更新和安全”项



然后在设置界面中选择“恢复”项



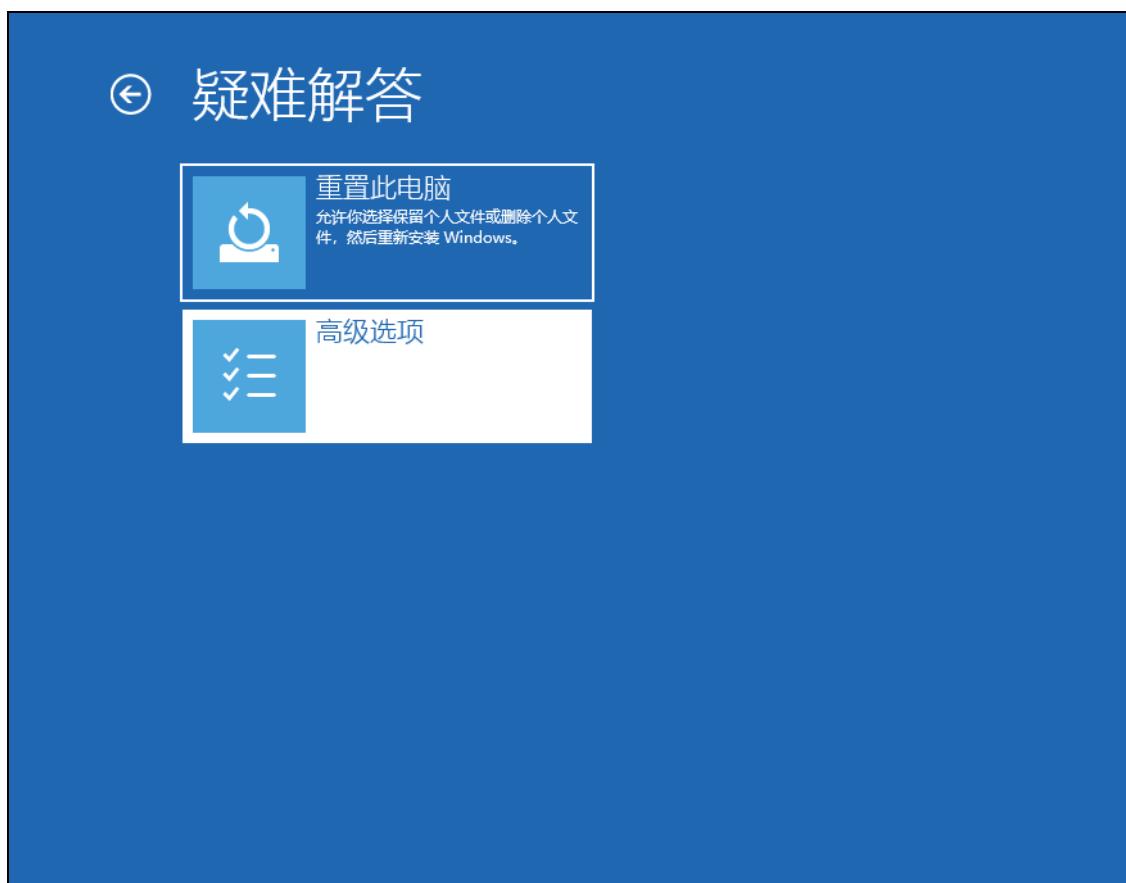
在恢复界面中，点击“高级启动”项中的“立即重新启动”按钮



在电脑重启前, 系统会先进入如下的启动菜单, 选择“疑难解答”项



在疑难解答界面中选择“高级选项”



STCM

然后选择“查看更多恢复选项”



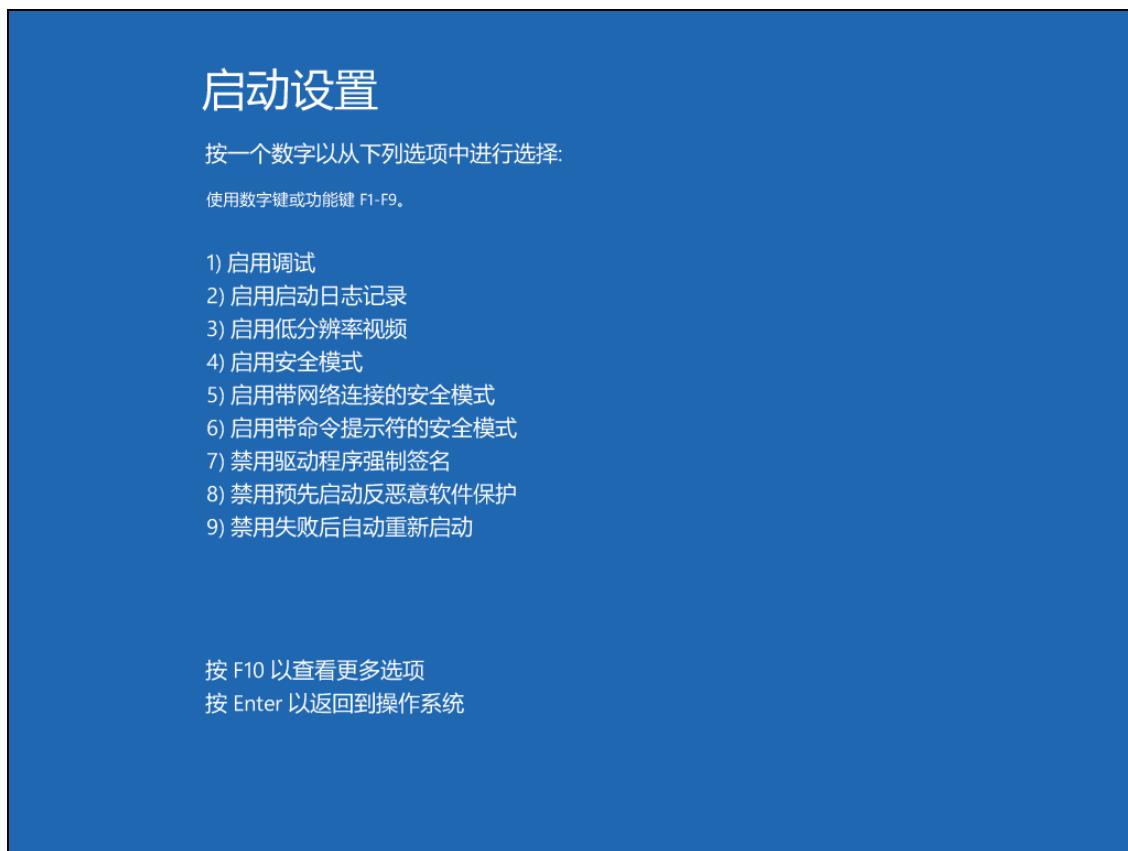
选择“启动设置”项



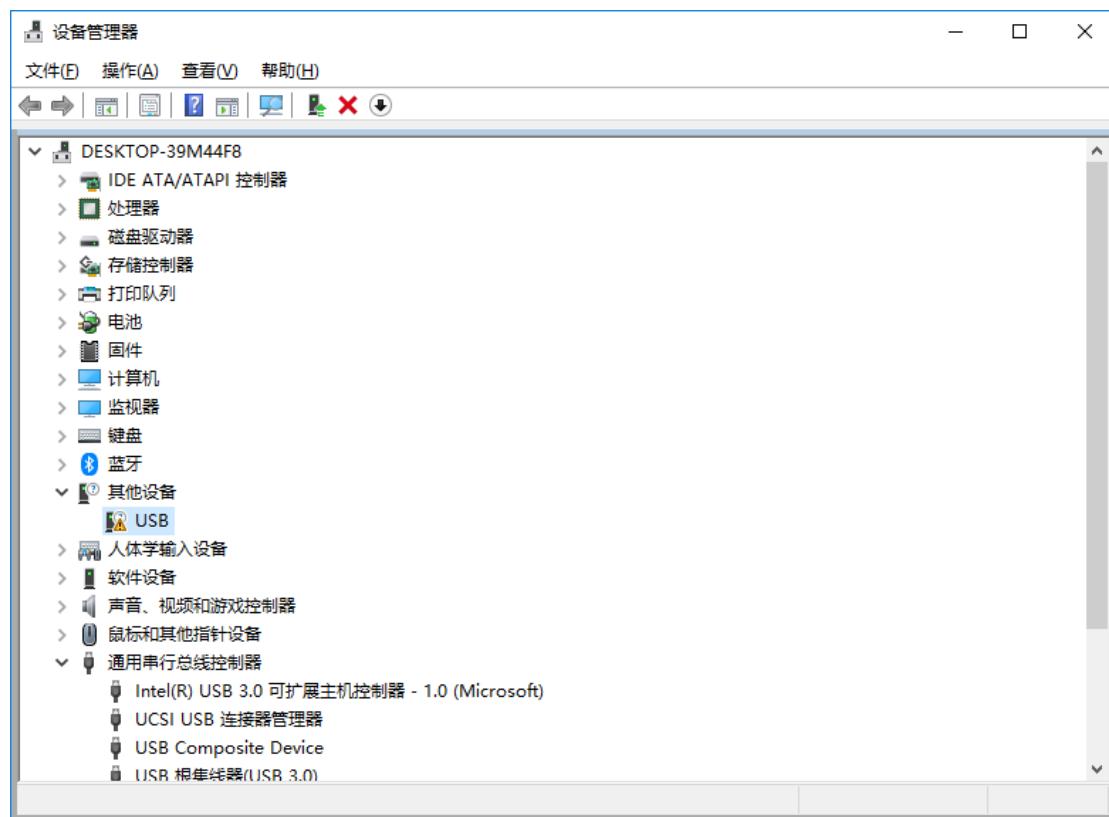
出现如下画面后，点击“重启”按钮重启电脑



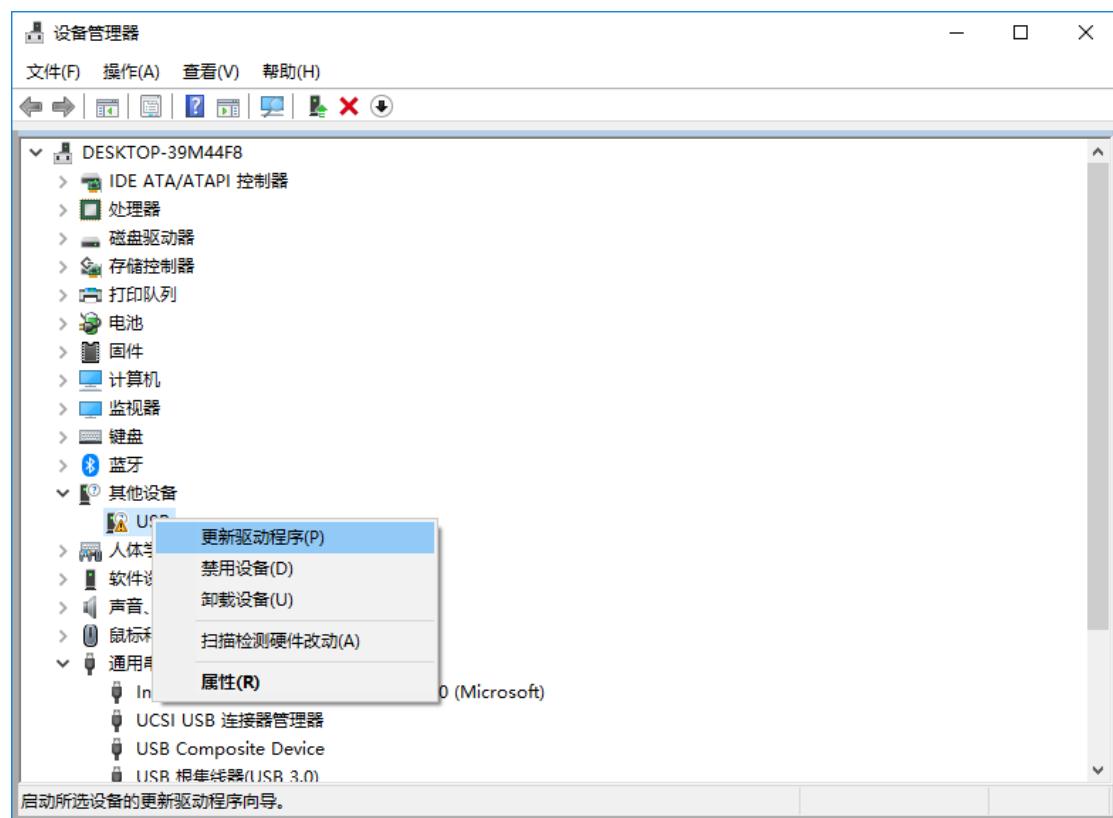
电脑重启后，会弹出“启动设置”界面，按“F7”按钮来选择“禁止驱动程序强制签名”项



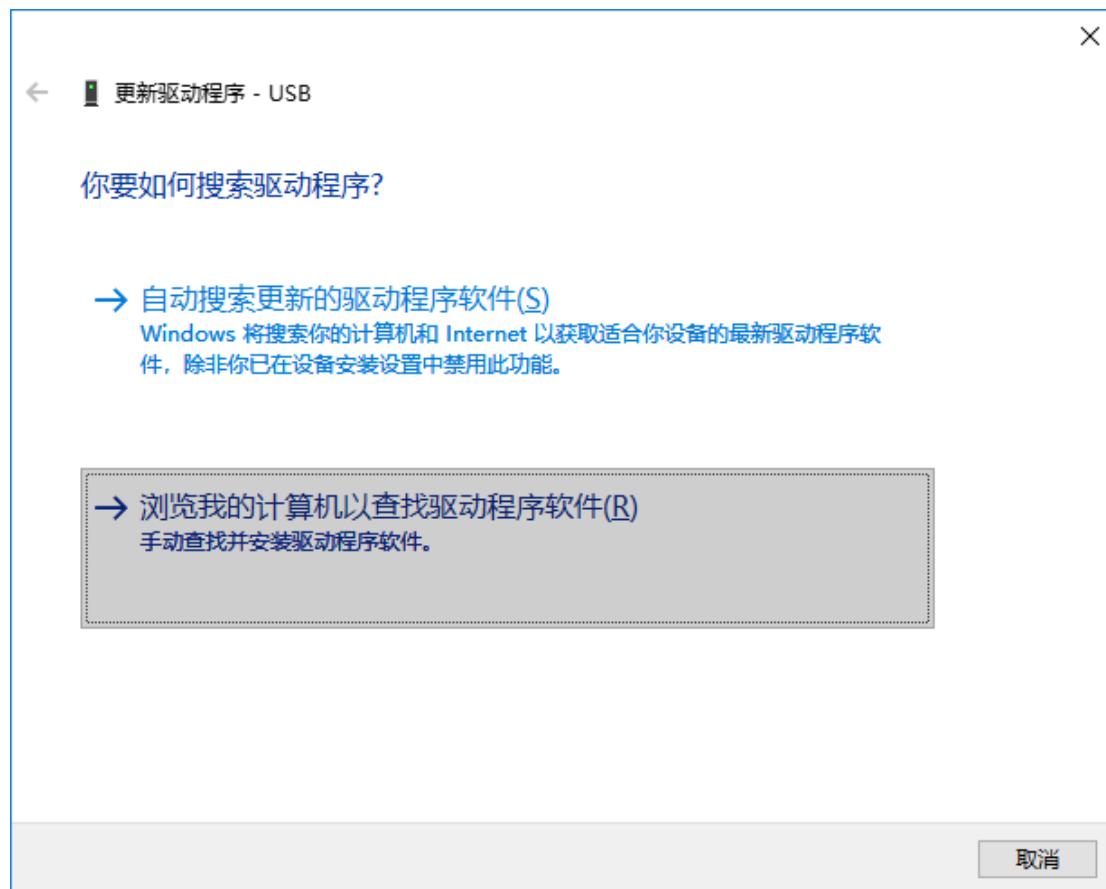
电脑启动完成后, 将准备好的芯片用 USB 线与电脑相连, 并打开“设备管理器”, 此时由于驱动还没有开始安装, 所以在设备管理器中会显示为一个带感叹号的未知设备



鼠标右键单击未知设备，选择右键菜单中的“更新驱动程序”



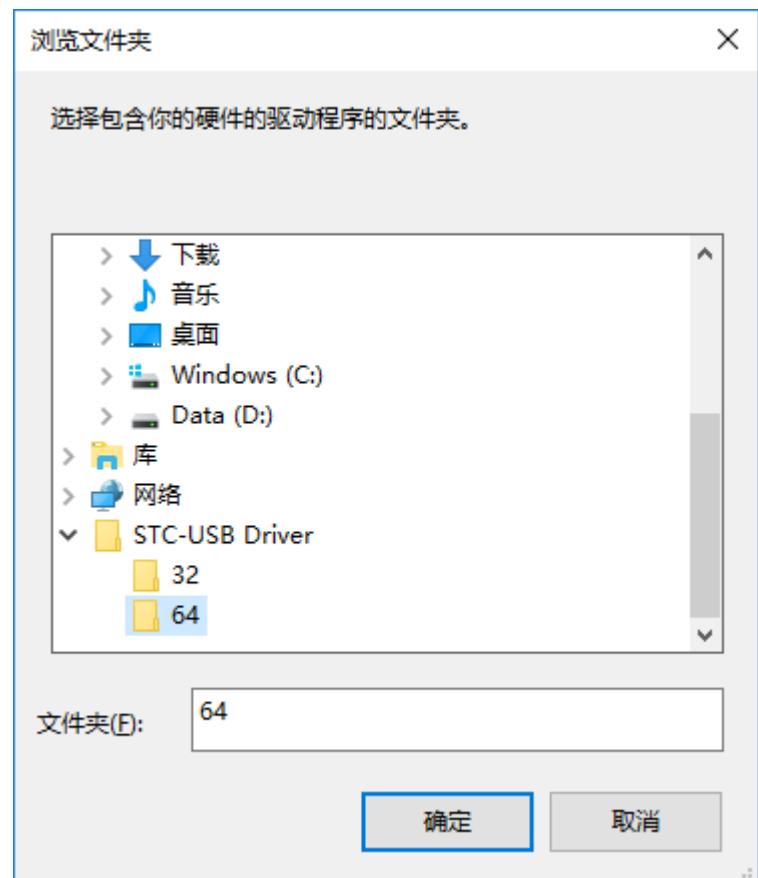
在弹出的驱动安装程序选择画面中, 选择“浏览我的计算机以查找驱动程序软件”项



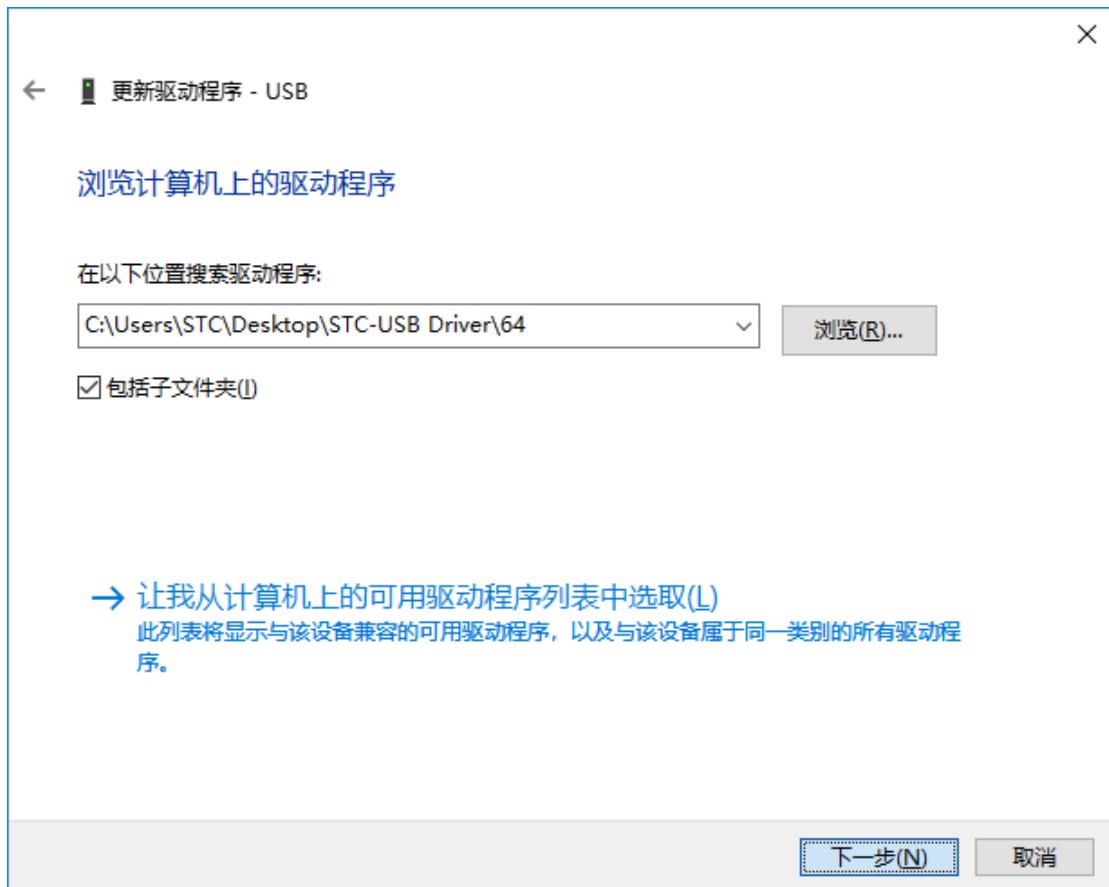
在如下界面中，点击“浏览”按钮



找到之前解压缩到硬盘中的“STC-USB Driver”目录，选择目录中的“64”目录，并确定



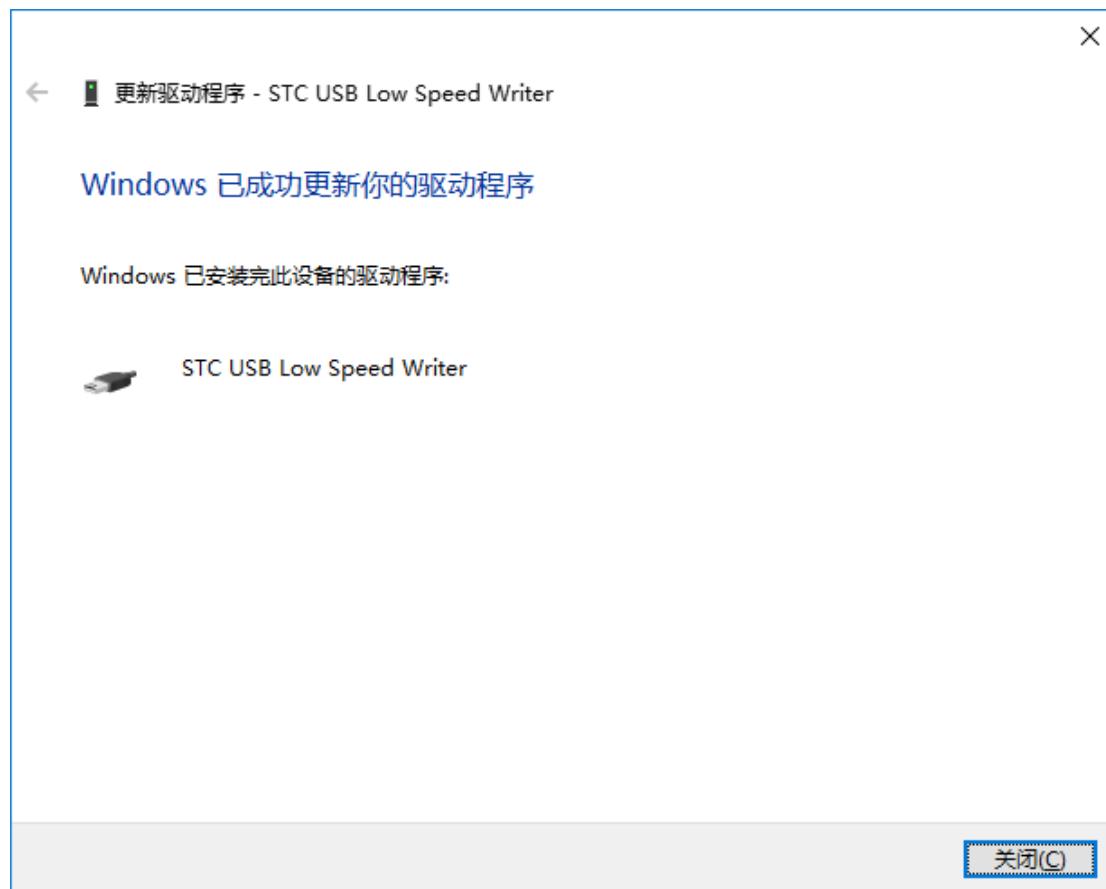
点击“下一步”开始安装驱动



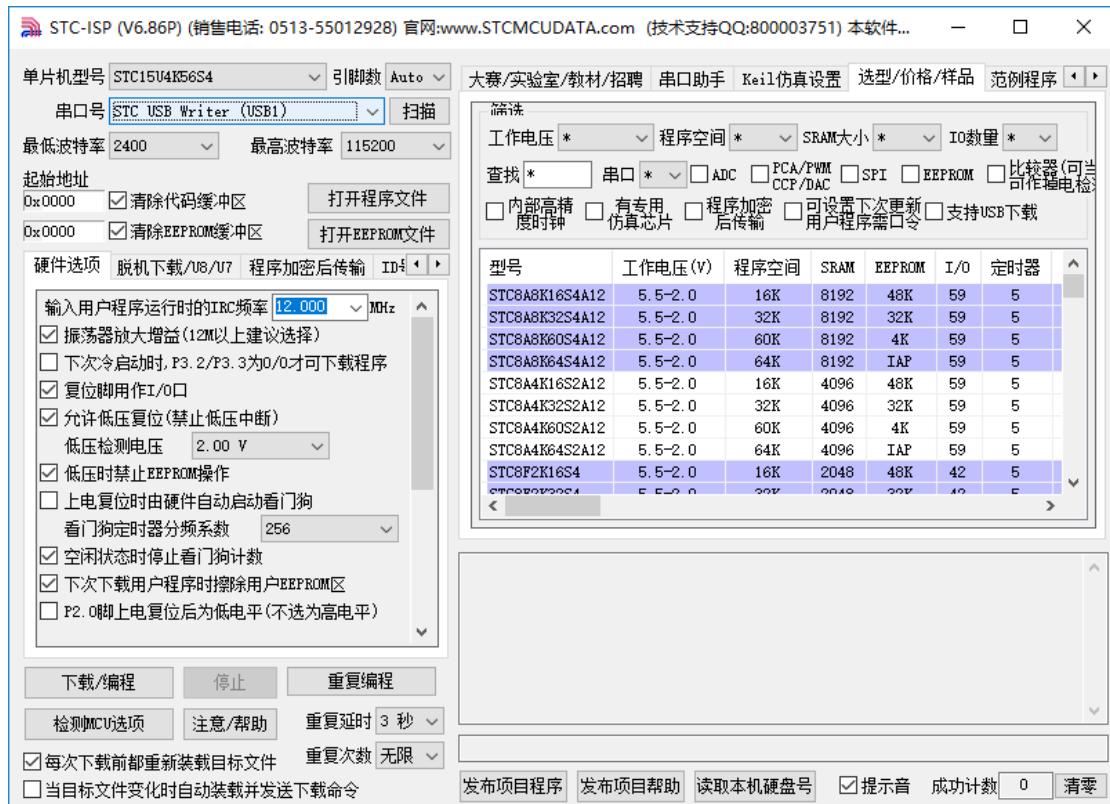
驱动安装的过程中，会弹出如下的警告画面，选择“始终安装此驱动程序软件”



出现下面的画面时，驱动程序就安装成功了



在回到 STC-ISP 的下载软件, 此时“串口号”的下拉列表中已自动选择了“STC USB Writer (USB1)”, 即可使用 USB 进行 ISP 下载了



附录D 使用第三方MCU对STC8A/STC8F系列单片机进行ISP下载范例程序

C 语言代码

```
//注意: 使用本代码对STC8A/F系列的单片机进行下载时, 必须要执行了Download 代码之后,
//才能给目标芯片上电, 否则目标芯片将无法正确下载

#include "reg51.h"

typedef bit      BOOL;
typedef unsigned char   BYTE;
typedef unsigned short WORD;

//宏、常量定义
#define FALSE      0
#define TRUE       1
#define LOBYTE(w)   ((BYTE)(WORD)(w))
#define HIBYTE(w)   ((BYTE)((WORD)(w) >> 8))

#define MINBAUD    2400L
#define MAXBAUD    115200L

#define FOSC        11059200L           //主控芯片工作频率
#define BR(n)       (65536 - FOSC/4/(n)) //主控芯片串口波特率计算公式
#define TIMS        (65536 - FOSC/1000) //主控芯片 1ms 定时初值

#define FUSER       24000000L          //STC8A/F 系列目标芯片工作频率
#define RL(n)       (65536 - FUSER/4/(n)) //STC8A/F 系列目标芯片串口波特率计算公式

sfr     AUXR = 0x8e;
sfr     P3MI = 0xB1;
sfr     P3M0 = 0xB2;

//变量定义
BOOL f1ms;                      //1ms 标志位
BOOL UartBusy;                   //串口发送忙标志位
BOOL UartReceived;               //串口数据接收完成标志位
BYTE UartRecvStep;               //串口数据接收控制
BYTE TimeOut;                    //串口通讯超时计数器
BYTE xdata TxBuffer[256];        //串口数据发送缓冲区
BYTE xdata RxBuffer[256];        //串口数据接收缓冲区
char code DEMO[256];             //演示代码数据

//函数声明
void Initial(void);
void DelayXms(WORD x);
BYTE UartSend(BYTE dat);
void CommInit(void);
void CommSend(BYTE size);
BOOL Download(BYTE *pdat, long size);
```

```
//主函数入口
void main(void)
{
    P3M0 = 0x00;
    P3M1 = 0x00;

    Initial();
    if (Download(DEMO, 256))
    {
        // 下载成功
        P3 = 0xff;
        DelayXms(500);
        P3 = 0x00;
        DelayXms(500);
        P3 = 0xff;
    }
    else
    {
        // 下载失败
        P3 = 0xff;
        DelayXms(500);
        P3 = 0xf3;
        DelayXms(500);
        P3 = 0xff;
    }
}

while (1);
}
```

//1ms 定时器中断服务程序

```
void tm0(void) interrupt 1
{
    static BYTE Counter100;

    f1ms = TRUE;
    if (Counter100-- == 0)
    {
        Counter100 = 100;
        if (TimeOut) TimeOut--;
    }
}
```

```
//串口中断服务程序
void uart(void) interrupt 4
{
    static WORD RecvSum;
    static BYTE RecvIndex;
    static BYTE RecvCount;
    BYTE dat;

    if (TI)
    {
        TI = 0;
        UartBusy = FALSE;
    }

    if (RI)
    {
        RI = 0;
        dat = SBUF;
        switch (UartRecvStep)
        {
            case 1:
                if (dat != 0xb9) goto L_CheckFirst;
                UartRecvStep++;
                break;
            case 2:
                if (dat != 0x68) goto L_CheckFirst;
                UartRecvStep++;
                break;
            case 3:
                if (dat != 0x00) goto L_CheckFirst;
                UartRecvStep++;
                break;
            case 4:
                RecvSum = 0x68 + dat;
                RecvCount = dat - 6;
                RecvIndex = 0;
                UartRecvStep++;
                break;
            case 5:
                RecvSum += dat;
                RxBuffer[RecvIndex++] = dat;
                if (RecvIndex == RecvCount) UartRecvStep++;
                break;
            case 6:
                if (dat != HIBYTE(RecvSum)) goto L_CheckFirst;
                UartRecvStep++;
                break;
            case 7:
                if (dat != LOBYTE(RecvSum)) goto L_CheckFirst;
                UartRecvStep++;
                break;
            case 8:
                if (dat != 0x16) goto L_CheckFirst;
                UartReceived = TRUE;
                UartRecvStep++;
                break;
        }
        L_CheckFirst:
        case 0:
        default:
```

```
        CommInit();
        UartRecvStep = (dat == 0x46 ? 1 : 0);
        break;
    }
}
}

//系统初始化
void Initial(void)
{
    UartBusy = FALSE;

    SCON = 0xd0;                                //串口数据模式必须为8位数据+1位偶校验
    AUXR = 0xc0;
    TMOD = 0x00;
    TH0 = HIBYTE(TIMS);
    TL0 = LOBYTE(TIMS);
    TR0 = 1;
    TH1 = HIBYTE(BR(MINBAUD));
    TL1 = LOBYTE(BR(MINBAUD));
    TRI = 1;
    ET0 = 1;
    ES = 1;
    EA = 1;
}

//Xms 延时程序
void DelayXms(WORD x)
{
    do
    {
        fIms = FALSE;
        while (!fIms);
    } while (x--);
}

//串口数据发送程序
BYTE UartSend(BYTE dat)
{
    while (UartBusy);

    UartBusy = TRUE;
    ACC = dat;
    TB8 = P;
    SBUF = ACC;

    return dat;
}

//串口通讯初始化
void CommInit(void)
{
    UartRecvStep = 0;
    TimeOut = 20;
    UartReceived = FALSE;
}

//发送串口通讯数据包
void CommSend(BYTE size)
```

```

{
    WORD sum;
    BYTE i;

    UartSend(0x46);
    UartSend(0xb9);
    UartSend(0x6a);
    UartSend(0x00);
    sum = size + 6 + 0x6a;
    UartSend(size + 6);
    for (i=0; i<size; i++)
    {
        sum += UartSend(TxBuffer[i]);
    }
    UartSend(HIBYTE(sum));
    UartSend(LOBYTE(sum));
    UartSend(0x16);
    while (UartBusy);

    CommInit();
}

//对STC8A/F系列的芯片进行ISP 下载程序
BOOL Download(BYTE *pdat, long size)
{
    BYTE arg;
    BYTE offset;
    BYTE cnt;
    WORD addr;

    //握手
    CommInit();
    while (1)
    {
        if (UartRecvStep == 0)
        {
            UartSend(0x7f);
            DelayXms(10);
        }
        if (UartReceived)
        {
            arg = RxBuffer[4];
            if (RxBuffer[0] == 0x50) break;
            return FALSE;
        }
    }
}

//设置参数(设置从芯片使用最高的波特率以及擦除等待时间等参数)
TxBuffer[0] = 0x01;
TxBuffer[1] = arg;
TxBuffer[2] = 0x40;
TxBuffer[3] = HIBYTE(RL(MAXBAUD));
TxBuffer[4] = LOBYTE(RL(MAXBAUD));
TxBuffer[5] = 0x00;
TxBuffer[6] = 0x00;
TxBuffer[7] = 0x81;
CommSend(8);
while (1)
{
}

```

```
if (TimeOut == 0) return FALSE;
if (UartReceived)
{
    if (RxBuffer[0] == 0x01) break;
    return FALSE;
}

//准备
TH1 = HIBYTE(BR(MAXBAUD));
TL1 = LOBYTE(BR(MAXBAUD));
DelayXms(10);
TxBuffer[0] = 0x05;
TxBuffer[1] = 0x00;
TxBuffer[2] = 0x00;
TxBuffer[3] = 0x5a;
TxBuffer[4] = 0xa5;
CommSend(5);
while (1)
{
    if (TimeOut == 0) return FALSE;
    if (UartReceived)
    {
        if (RxBuffer[0] == 0x05) break;
        return FALSE;
    }
}

//擦除
DelayXms(10);
TxBuffer[0] = 0x03;
TxBuffer[1] = 0x00;
TxBuffer[2] = 0x00;
TxBuffer[3] = 0x5a;
TxBuffer[4] = 0xa5;
CommSend(5);
TimeOut = 100;
while (1)
{
    if (TimeOut == 0) return FALSE;
    if (UartReceived)
    {
        if (RxBuffer[0] == 0x03) break;
        return FALSE;
    }
}

//写用户代码
DelayXms(10);
addr = 0;
TxBuffer[0] = 0x22;
TxBuffer[3] = 0x5a;
TxBuffer[4] = 0xa5;
offset = 5;
while (addr < size)
{
    TxBuffer[1] = HIBYTE(addr);
    TxBuffer[2] = LOBYTE(addr);
    cnt = 0;
```

```
while (addr < size)
{
    TxBuffer[cnt+offset] = pdat[addr];
    addr++;
    cnt++;
    if (cnt >= 128) break;
}
CommSend(cnt + offset);
while (1)
{
    if (TimeOut == 0) return FALSE;
    if (UartReceived)
    {
        if ((RxBuffer[0] == 0x02) && (RxBuffer[1] == 'T')) break;
        return FALSE;
    }
}
TxBuffer[0] = 0x02;
}

////写硬件选项
////如果不需要修改硬件选项,此步骤可直接跳过,此时所有的硬件选项
////都维持不变,MCU 的频率为上一次所调节频率
////若写硬件选项,MCU 的内部IRC 频率将被固定写为24M,其他选项恢复为出厂设置
////建议:第一次使用STC-ISP 下载软件将从芯片的硬件选项设置好
////以后再使用主芯片对从芯片下载程序时不写硬件选项
//DelayXms(10);
//for (cnt=0; cnt<128; cnt++)
//{
//    TxBuffer[cnt] = 0xff;
//}
//TxBuffer[0] = 0x04;
//TxBuffer[1] = 0x00;
//TxBuffer[2] = 0x00;
//TxBuffer[3] = 0x5a;
//TxBuffer[4] = 0xa5;
//TxBuffer[33] = arg;
//TxBuffer[34] = 0x00;
//TxBuffer[35] = 0x01;
//TxBuffer[41] = 0xbff;           //P5.4 为IO 口
//TxBuffer[42] = 0xbd;           //TxBuffer[42] = 0xad;      //P5.4 为复位脚
//TxBuffer[43] = 0xf7;
//TxBuffer[44] = 0xff;
//CommSend(45);
//while (1)
//{
//    if (TimeOut == 0) return FALSE;
//    if (UartReceived)
//    {
//        if ((RxBuffer[0] == 0x04) && (RxBuffer[1] == 'T')) break;
//        return FALSE;
//    }
//}
//
//下载完成
return TRUE;
}
```

```
char code DEMO[256] =  
{  
    0x80,0x00,0x75,0xB2,0xFF,0x75,0xB1,0x00,0x05,0xB0,0x11,0xE,0x80,0xFA,0xD8,0xFE,  
    0xD9,0xFC,0x22,  
};
```

备注: 用户若需要设置不同的工作频率, 可参考 7.3.7 和 7.3.8 章的范例代码

STCMCU

附录E 电气特性

绝对最大额定值

参数	最小值	最大值	单位
存储温度	-55	+125	°C
工作温度	-40	+85	°C
工作电压	2.0	5.5	V
VDD 对地电压	-0.3	+5.5	V
IO 口对地电压	-0.3	VDD+0.3	V

直流特性 (VSS=0V, VDD=5.0V, 测试温度=25°C) (STC8F2K 系列)

标号	参数	范围				测试环境
		最小值	典型值	最大值	单位	
I _{PD}	掉电模式电流 (SCC = 1)	-	0.08	-	uA	5.0V
	掉电模式电流 (SCC = 0)	-	1.5	-	uA	5.0V
I _{WKT}	掉电唤醒定时器	-	5	-	uA	5.0V
I _{LVD}	低压检测模块	-	260	-	uA	5.0V
I _{IDL}	空闲模式电流 (6MHz)	-	1.3	-	mA	5.0V
	空闲模式电流 (11.0592MHz)		1.7		mA	5.0V
	空闲模式电流 (20MHz)		2.3		mA	5.0V
	空闲模式电流 (22.1184MHz)	-	2.5	-	mA	5.0V
	空闲模式电流 (24MHz)	-	2.6	-	mA	5.0V
	空闲模式电流 (内部 32KHz)	-	850	-	uA	5.0V
I _{NOR}	正常模式电流 (6MHz)	-	2.7	-	mA	5.0V
	正常模式电流 (11.0592MHz)	-	3.8	-	mA	5.0V
	正常模式电流 (20MHz)	-	5.9	-	mA	5.0V
	正常模式电流 (22.1184MHz)	-	6.3	-	mA	5.0V
	正常模式电流 (24MHz)	-	6.5	-	mA	5.0V
	正常模式电流 (内部 32KHz)	-	950	-	uA	5.0V
I _{CC}	普通工作模式电流	-	4	20	mA	5.0V
V _{IL1}	输入低电平	-	-	1.4	V	5.0V (打开施密特触发)
		-	-	1.5	V	5.0V (关闭施密特触发)
V _{IH1}	输入高电平 (普通 I/O)	1.7	-	-	V	5.0V (打开施密特触发)
		1.6	-	-	V	5.0V (关闭施密特触发)
V _{IH2}	输入高电平 (复位脚)	1.6	-	1.7	V	5.0V
I _{OL1}	输出低电平的灌电流	-	20	-	mA	5.0V, 端口电压 0.45V
I _{OH1}	输出高电平电流 (双向模式)	200	270	-	uA	5.0V
I _{OH2}	输出高电平电流 (推挽模式)	-	20	-	mA	5.0V, 端口电压 2.4V
I _{IL}	逻辑 0 输入电流	-	-	50	uA	5.0V, 端口电压 0V
I _{TL}	逻辑 1 到 0 的转移电流	100	270	600	uA	5.0V, 端口电压 2.0V
R _{PU}	IO 口上拉电阻	4.1	4.2	4.4	KΩ	5.0V

R _{PU}	IO 口上拉电阻	5.8	5.9	6.0	KΩ	3.3V
-----------------	----------	-----	-----	-----	----	------

直流特性 (VSS=0V, VDD=5.0V, 测试温度=25°C) (STC8A8K 系列)

标号	参数	范围				测试环境
		最小值	典型值	最大值	单位	
I _{PD}	掉电模式电流 (SCC = 1)	-	0.08	-	uA	5.0V
	掉电模式电流 (SCC = 0)	-	1.6	-	uA	5.0V
I _{WKT}	掉电唤醒定时器	-	5	-	uA	5.0V
I _{LVD}	低压检测模块	-	260	-	uA	5.0V
I _{IDL}	空闲模式电流 (6MHz)	-	1.5	-	mA	5.0V
	空闲模式电流 (11.0592MHz)		1.9		mA	5.0V
	空闲模式电流 (20MHz)		2.7		mA	5.0V
	空闲模式电流 (22.1184MHz)	-	3.0	-	mA	5.0V
	空闲模式电流 (24MHz)	-	3.2	-	mA	5.0V
	空闲模式电流 (内部 32KHz)	-	890	-	uA	5.0V
I _{NOR}	正常模式电流 (6MHz)	-	3.0	-	mA	5.0V
	正常模式电流 (11.0592MHz)	-	4.3	-	mA	5.0V
	正常模式电流 (20MHz)	-	6.8	-	mA	5.0V
	正常模式电流 (22.1184MHz)	-	7.4	-	mA	5.0V
	正常模式电流 (24MHz)	-	7.8	-	mA	5.0V
	正常模式电流 (内部 32KHz)	-	950	-	uA	5.0V
I _{CC}	普通工作模式电流	-	4	20	mA	5.0V
V _{IL1}	输入低电平	-	-	1.4	V	5.0V (打开施密特触发)
		-	-	1.5	V	5.0V (关闭施密特触发)
V _{IH1}	输入高电平 (普通 I/O)	2.2	-	-	V	5.0V (打开施密特触发)
		1.6	-	-	V	5.0V (关闭施密特触发)
V _{IH2}	输入高电平 (复位脚)	2.2	-	-	V	5.0V
I _{OL1}	输出低电平的灌电流	-	20	-	mA	5.0V, 端口电压 0.45V
I _{OH1}	输出高电平电流 (双向模式)	200	270	-	uA	5.0V
I _{OH2}	输出高电平电流 (推挽模式)	-	20	-	mA	5.0V, 端口电压 2.4V
I _{IL}	逻辑 0 输入电流	-	-	50	uA	5.0V, 端口电压 0V
I _{TL}	逻辑 1 到 0 的转移电流	100	270	600	uA	5.0V, 端口电压 2.0V
R _{PU}	IO 口上拉电阻	4.1	4.2	4.4	KΩ	5.0V
R _{PU}	IO 口上拉电阻	5.8	5.9	6.0	KΩ	3.3V

内部 IRC 温漂特性 (参考温度 25°C)

温度	范围
-40°C ~ 85°C	-1.8% ~ +0.8%
-20°C ~ 65°C	-1.0% ~ +0.5%

附录F 更新记录

● 2019/5/29

1. 增加使用第三方 MCU 对 STC8H 单片机进行 ISP 下载的示例代码（附录 D）
2. 增加使用 IO 口直接驱动 LCD 的示例代码（P9.5.3）

● 2018/11/16

1. 在应用注意事项中总结 STC8A 系列芯片和 STC8F 系列芯片在使用中的一些注意事项
2. 删除 STC8 系列中串口中继功能的说明部分（STC8 系列不支持串口中继功能）
3. 增加 Win10 系统中 STC-USB 驱动的安装方法

● 2018/8/14

1. 增加 STC8 系列 QFN64 和 QFN48 的封装尺寸图（15.3）

● 2018/6/26

1. 增加 STC8 系列 EEPROM 大小以及地址的说明（15.3）
2. 增加使用 MOVC 读取 EEPROM 的范例程序（15.4.2）

● 2018/5/7

1. 增加使用 CLR EA 指令屏蔽中断的重要说明

● 2018/4/28

1. 增加 STC8F2K64S4 型号 E 版芯片的重要说明
2. 增加 STC8F2K64S2 型号 E 版芯片的重要说明
3. 增加 STC8A8K64S4A12 型号 G 版芯片的重要说明
4. 增加 STC8A4K64S2A12 型号 G 版芯片的重要说明

● 2018/4/25

1. 增加使用 PL2303-GL 进行 ISP 下载的参考线路图（5.1.3、5.2.4）
2. 增加 STC8C 系列的提前通知和管脚图（2.6、3.6）
3. 增加 STC8P 系列的提前通知和管脚图（2.7~2.9、3.8~3.10）
4. 增加 STC8 系列使用外部晶振进行仿真的重要说明
5. 增加 STC8 系列内部 IRC 频率调整机制的说明章节（6.2）
6. 增加 STC15 系列内部 IRC 频率调整机制的说明章节（6.3）

● 2018/3/23

1. 修改 STC8F1K08S2 芯片 SOP16 管脚名称错误 (3.1.5)

● 2018/3/20

1. 增加从 ROM 和 RAM 中读取重要测试参数的范例程序 (8.3)
 - a) 从存储器中读取 Bandgap 电压值
 - b) 从存储器中读取全球唯一 ID 号
 - c) 从存储器中读取 32K 掉电唤醒定时器频率
 - d) 从存储器中读取 IRC 参数手动设置内部 IRC 频率

● 2018/3/13

1. 增加 STC8F1K08S2 型号 SOP16 的管脚图

● 2018/3/6

1. 增加利用比较器作掉电检测的范例程序 (15.3.3)
2. 增加利用比较器检测工作电压 (电池电压) 的范例程序 (15.3.4)
3. 增加利用芯片内部的 LVD (低电压检测) 检测工作电压 (电池电压) 的范例程序 (7.4.13)
4. 增加 STC8F1K08S2 的特性说明和管脚图
5. 增加 STC8H1K08S2A10 的特性说明和管脚图

● 2018/1/30

1. 去掉范例程序中"using、_at_"的使用
2. 对 P2.0 的 RSTV 功能进行说明
3. 注明内部参考电压的电压值
4. 调整部分参考线路图

● 2017/11/27

1. 增加掉电唤醒 IO 口的说明, 增加掉电唤醒范例程序

● 2017/11/7

1. 修正主时钟分频输出的时钟为经 CLKDIV 分频后的系统时钟 (**之前错误的描述为主时钟分频输出的时钟为 CLKDIV 分频之前的主时钟**)
2. STC8 系列的比较器中断可设置 4 级中断优先级 (之前的版本的资料描述有误)
3. 增加中断系统的结构框图
4. 增加 ADC 的第 16 通道测试外部电池电压的测试范例代码 (第 17.3.4 章)
5. 增加读取 STC8 系列 MCU 内部的重要参数的范例代码 (第 8.3 章)

● 2017/11/2

1. 增加 STC8H1K08S2A10 系列和 STC8H1K08S2 系列的 SOP16 管脚图
2. 修正 EEPROM 编程和擦除时钟数
3. 增加 STC8F 系列和 STC8A 系列 EEPROM 编程和擦除等待时间设置的重要说明

● **2017/10/31**

1. 更新芯片封装图
2. 更新 ADC 典型应用线路图

● **2017/8/9**

1. 修正 STC8A4K64S2A12 系列的 IO 口数量为 59 个
2. 更新选型价格表

● **2017/8/1**

1. 更新应用注意事项

● **2017/7/27**

1. 更新选型及价格表

● **2017/7/12**

1. 增加 STC8F2K64S4 系列 D 版芯片的重要说明
2. STC8F2K64S4 系列 D 版芯片开始送样 (可选封装 LQFP44、LQFP32、PDIP40)

● **2017/7/3**

1. STC8F2K 系列增加 TSSOP20 和 SOP16 的管脚图

● **2017/6/30**

1. 增加 QFN32、TSSOP20、SOP16 的封装尺寸图
2. 更新代修正的重要说明

● **2017/5/17**

1. 增加 STC8F2K64S2 系列
2. 增加 STC8A4K64S2A12 系列
3. 增加 STC8F1K08S2A10 系列
4. 增加 STC8F1K08S2 系列

● **2017/5/12**

1. 增加了 I2C 主模式的辅助命令的说明 (辅助命令仅对 STC8F2K64S4 系列的 C 版芯片、

STC8A8K64S4A12 系列的 E 版芯片有效)

2. 增加了使用 U8W、U8-Mini 和 PL2303 进行 ISP 下载的参考线路图
3. 增加了端口内部上拉电阻和施密特触发控制的说明
4. 更新了 STC8 系列单片机的选型以及参考价格

● 2017/3/1

1. 增加重要说明章节以及仿真器使用的章节

● 2016/12/22

1. STC8F2K64S4 系列 C 版芯片无 PCA/CCP/PWM 功能
2. STC8A8K64S4A12 系列 B 版芯片有下列几点需要注意
 - e) 串口 1 若使用定时器 1 的模式 2 做波特率发生器时, SMOD (PCON.7) 位必须使能
 - f) 所有的串口 (包括串口 1、串口 2、串口 3 和串口 4) 在接收数据时, 都必须是两位停止位, 否则可能会出现数据丢失
 - g) 在访问内部扩展 XSFR 时, 需要将 EXTRAM (AUXR.1) 位置 1, 否则在写入 XSFR 时会影响内部 XRAM 的最后 512 字节

● 2016/11/22

1. 增加范例程序
2. STC8F2K64S4 系列 B 版芯片有下列几点需要注意
 - a) 串口 1 若使用定时器 1 的模式 2 做波特率发生器时, SMOD (PCON.7) 位必须使能
 - b) 所有的串口 (包括串口 1、串口 2、串口 3 和串口 4) 在接收数据时, 都必须是两位停止位, 否则可能会出现数据丢失
 - c) 在访问内部扩展 XSFR 时, 需要将 EXTRAM (AUXR.1) 位置 1, 否则在写入 XSFR 时会影响内部 XRAM 的最后 512 字节

● 2016/9/13

1. 增加 CAN 总线功能 (定义 CAN 的管脚、CAN 中断相关 SFR)
CAN 总线的功能 SFR 需要讨论后进行定义
2. 增加 IP3 和 IP3H, 用于设置串口 3 和串口 4 的中断优先级
3. **上面两项功能处于规划阶段, 目前芯片并无此功能**

● 2016/5/6

1. 选型价格表中增加 I²C 选项
2. 修正概述中关于串口 4 的端口切换的描述错误

● 2016/4/27

1. 修改指令表中部分错误的指令执行时间

● **2016/4/22**

1. 修改 STC8A8K64S4 系列的 LQFP64S 的封装图和管脚排列
2. 修改 STC8F8K64S4 系列的 LQFP64S 的封装图和管脚排列

● **2016/4/15**

1. 修改 STC8A8K64S4 系列的 LQFP44、LQFP48 的封装图和管脚排列
2. 修改 STC8F8K64S4 系列的 LQFP44、LQFP48 的封装图和管脚排列
3. 修改 STC8F2K64S4 系列的 LQFP44 的封装图和管脚排列