

深圳国芯人工智能有限公司

STC8G 系列单片机

技术参考手册

STCMCU

技术支持网站: www.STCAI.com

官方技术论坛: www.STCAIMCU.com

资料更新日期: 2023/1/17

目录

1	概述.....	1
2	特性、价格及管脚.....	2
2.1	STC8G1K08-36I-SOP8/DFN8 系列.....	3
2.1.1	特性及价格（有 16 位硬件乘除法器 MDU16，准 16 位单片机）.....	3
2.1.2	管脚图，最小系统.....	5
2.1.3	管脚说明.....	6
2.2	STC8G1K08A-36I-SOP8/DFN8/DIP8 系列	7
2.2.1	特性及价格（有 16 位硬件乘除法器 MDU16，准 16 位单片机）.....	7
2.2.2	管脚图，最小系统.....	9
2.2.3	管脚说明.....	10
2.3	STC8G1K08-38I-TSSOP20/QFN20/SOP16 系列.....	12
2.3.1	特性及价格	12
2.3.2	管脚图，最小系统.....	15
2.3.3	管脚说明.....	17
2.4	STC8G2K64S4-36I-LQFP48/32/QFN4832 系列（45 路增强型 PWM）.....	20
2.4.1	特性及价格（有 16 位硬件乘除法器 MDU16，准 16 位单片机）.....	20
2.4.2	管脚图，最小系统.....	23
2.4.3	管脚说明.....	26
2.5	STC8G2K64S2-36I-LQFP48/QFN48 系列（8 路增强型 PWM）.....	32
2.5.1	特性及价格（有 16 位硬件乘除法器 MDU16，准 16 位单片机）.....	32
2.5.2	管脚图，最小系统.....	35
2.5.3	管脚说明.....	37
2.6	STC8G1K08T-36I-TSSOP20（触摸按键/LED 系列）(用 STC8H4K64LR 取代)	43
2.6.1	特性及价格	43
2.6.2	管脚图，最小系统.....	46
2.6.3	管脚说明.....	47
2.7	STC15H 系列（传统 STC15 系列提升性能特殊型号）.....	51
2.7.1	特性及价格（有 16 位硬件乘除法器 MDU16，准 16 位单片机）.....	51
2.7.2	管脚图，最小系统.....	52
3	功能脚切换.....	55
3.1	功能脚切换相关寄存器	55
3.1.1	外设端口切换控制寄存器 1 (P_SW1)，串口 1、CCP、SPI 切换.....	55
3.1.2	外设端口切换控制寄存器 2 (P_SW2)，串口 2/3/4、I2C、比较器输出切换	56
3.1.3	时钟选择寄存器 (MCLKOCR)	57
3.2	范例程序	58
3.2.1	串口 1 切换	58
3.2.2	串口 2 切换	59
3.2.3	串口 3 切换	61
3.2.4	串口 4 切换	62
3.2.5	SPI 切换	64

3.2.6	PCA/CCP/PWM 切换	65
3.2.7	I2C 切换	67
3.2.8	比较器输出切换	69
3.2.9	主时钟输出切换	70
4	封装尺寸图	73
4.1	SOP8 封装尺寸图	73
4.2	DFN8 封装尺寸图 (3mm*3mm)	74
4.3	SOP16 封装尺寸图	75
4.4	TSSOP20 封装尺寸图	76
4.5	QFN20 封装尺寸图 (3mm*3mm)	77
4.6	LQFP32 封装尺寸图 (9mm*9mm)	78
4.7	QFN32 封装尺寸图 (4mm*4mm)	79
4.8	LQFP48 封装尺寸图 (9mm*9mm)	80
4.9	QFN48 封装尺寸图 (6mm*6mm)	81
4.10	LQFP64S 封装尺寸图 (12mm*12mm)	82
4.11	QFN64 封装尺寸图 (8mm*8mm)	83
4.12	STC8G 系列单片机命名规则	84
5	ISP 下载及典型应用线路图	85
5.1	STC8G 系列 ISP 下载应用线路图	85
5.1.1	使用 STC-USB Link1D 工具下载, 支持在线和脱机下载	85
5.1.2	软件模拟 USB 直接 ISP 下载, 建议尝试, 不支持仿真 (5V 系统)	87
5.1.3	软件模拟 USB 直接 ISP 下载, 建议尝试, 不支持仿真 (3.3V 系统)	89
5.1.4	使用一箭双雕之 USB 转串口工具下载	91
5.1.5	使用 U8-Mini 工具下载, 支持 ISP 在线和脱机下载, 也可支持仿真	93
5.1.6	使用 U8W 工具下载, 支持 ISP 在线和脱机下载, 也可支持仿真	94
5.1.7	使用 RS-232 转换器下载 (无独立 VREF 脚), 也可支持仿真	96
5.1.8	使用 RS-232 转换器下载 (有独立 VREF 脚、一般精度 ADC), 也可支持仿真	97
5.1.9	使用 RS-232 转换器下载 (有独立 VREF 脚、高精度 ADC), 也可支持仿真	98
5.1.10	使用 PL2303-GL 下载, 也可支持仿真	99
5.1.11	单片机电源控制参考电路	100
5.2	用 STC 一箭双雕之 USB 转双串口仿真 STC8 系列 MCU	101
5.3	STC-ISP 下载软件高级应用	110
5.3.1	发布项目程序	110
5.3.2	程序加密后传输 (防烧录时串口分析出程序)	114
5.3.3	发布项目程序+程序加密后传输结合使用	118
5.3.4	用户自定义下载 (实现不停电下载)	119
6	时钟、复位、省电模式与系统电源管理	123
6.1	系统时钟控制	123
6.1.1	系统时钟选择寄存器 (CLKSEL)	123
6.1.2	时钟分频寄存器 (CLKDIV)	124
6.1.3	内部高速高精度 IRC 控制寄存器 (HIRCCR)	124
6.1.4	外部振荡器控制寄存器 (XOSCCR)	124
6.1.5	内部 32KHz 低速 IRC 控制寄存器 (IRC32KCR)	125
6.1.6	主时钟输出控制寄存器 (MCLKOCR)	125

6.2	STC8G 系列内部 IRC 频率调整	126
6.2.1	IRC 频段选择寄存器 (IRCBAND)	126
6.2.2	内部 IRC 频率调整寄存器 (IRTRIM)	126
6.2.3	内部 IRC 频率微调寄存器 (LIRTRIM)	127
6.2.4	时钟分频寄存器 (CLKDIV)	127
6.2.5	分频出 3MHz 用户工作频率，并用户动态改变频率追频示例	128
6.3	系统复位	131
6.3.1	看门狗复位 (WDT_CONTR)	132
6.3.2	软件复位 (IAP_CONTR)	134
6.3.3	低压复位 (RSTCFG)	135
6.3.4	低电平上电复位参考电路 (一般不需要)	136
6.3.5	低电平按键手动复位参考电路	136
6.3.6	传统 8051 高电平上电复位参考电路	137
6.4	外部晶振及外部时钟电路	138
6.4.1	外部晶振输入电路	138
6.4.2	外部时钟输入电路 (P1.6 不可用作普通 I/O)	138
6.5	时钟停振/省电模式与系统电源管理	138
6.5.1	电源控制寄存器 (PCON)	138
6.6	掉电唤醒定时器	140
6.6.1	掉电唤醒定时器计数寄存器 (WKTCL, WKTCH)	140
6.7	范例程序	141
6.7.1	选择系统时钟源	141
6.7.2	主时钟分频输出	143
6.7.3	看门狗定时器应用	145
6.7.4	软复位实现自定义下载	147
6.7.5	低压检测	149
6.7.6	省电模式	151
6.7.7	使用 INT0/INT1/INT2/INT3/INT4 管脚中断唤醒省电模式	153
6.7.8	使用 T0/T1/T2/T3/T4 管脚中断唤醒 MCU 省电模式	156
6.7.9	使用 RxD/RxD2/RxD3/RxD4 管脚中断唤醒 MCU 省电模式	161
6.7.10	使用 I2C 的 SDA 脚唤醒 MCU 省电模式	164
6.7.11	使用掉电唤醒定时器唤醒省电模式	167
6.7.12	LVD 中断唤醒省电模式，建议配合使用掉电唤醒定时器	169
6.7.13	使用 CCP0/CCP1/CCP2 管脚中断唤醒 MCU 省电模式	171
6.7.14	比较器中断唤醒省电模式，建议配合使用掉电唤醒定时器	174
6.7.15	使用 LVD 功能检测工作电压 (电池电压)	177
7	存储器	182
7.1	程序存储器	182
7.2	数据存储器	183
7.2.1	内部 RAM	183
7.2.2	程序状态寄存器 (PSW)	183
7.2.3	内部扩展 RAM, XRAM, XDATA	184
7.2.4	辅助寄存器 (AUXR)	184
7.2.5	外部扩展 RAM, XRAM, XDATA	185

7.2.6	总线速度控制寄存器 (BUS_SPEED)	185
7.2.7	8051 中可位寻址的数据存储器	186
7.2.8	扩展 SFR 使能寄存器 EAXFR 的使用说明	188
7.3	存储器中的特殊参数, 在 ISP 下载时可烧录进程序 FLASH.....	189
7.3.1	读取内部 1.19V 参考信号源值 (从 Flash 程序存储器 (ROM) 中读取).....	191
7.3.2	读取内部 1.19V 参考信号源值 (从 RAM 中读取).....	194
7.3.3	读取全球唯一 ID 号 (从 Flash 程序存储器 (ROM) 中读取).....	197
7.3.4	读取全球唯一 ID 号 (从 RAM 中读取).....	200
7.3.5	读取 32K 掉电唤醒定时器的频率 (从 Flash 程序存储器 (ROM) 中读取).....	204
7.3.6	读取 32K 掉电唤醒定时器的频率 (从 RAM 中读取).....	207
7.3.7	用户自定义内部 IRC 频率 (从 Flash 程序存储器 (ROM) 中读取)	210
7.3.8	用户自定义内部 IRC 频率 (从 RAM 中读取)	215
8	特殊功能寄存器.....	218
8.1	STC8G1K08 系列	218
8.2	STC8G1K08-8Pin 系列	219
8.3	STC8G1K08A 系列	220
8.4	STC8G2K64S4 系列	221
8.5	STC8G2K64S2 系列	224
8.6	STC8G1K08T 系列	227
8.7	STC15H2K64S4 系列	229
8.8	特殊功能寄存器列表	232
9	I/O 口	246
9.1	I/O 口相关寄存器	246
9.1.1	端口数据寄存器 (Px)	247
9.1.2	端口模式配置寄存器 (PxM0, PxM1)	248
9.1.3	端口上拉电阻控制寄存器 (PxPU)	248
9.1.4	端口施密特触发控制寄存器 (PxNCS)	249
9.1.5	端口电平转换速度控制寄存器 (PxSR)	249
9.1.6	端口驱动电流控制寄存器 (PxDR)	249
9.1.7	端口数字信号输入使能控制寄存器 (PxIE)	250
9.2	配置 I/O 口	251
9.3	I/O 的结构图	253
9.3.1	准双向口 (弱上拉)	253
9.3.2	推挽输出	253
9.3.3	高阻输入	254
9.3.4	开漏输出	254
9.3.5	新增 4.1K 上拉电阻	255
9.3.6	如何设置 I/O 口对外输出速度	255
9.3.7	如何设置 I/O 口电流驱动能力	256
9.3.8	如何降低 I/O 口对外辐射	256
9.4	范例程序	257
9.4.1	端口模式设置	257
9.4.2	双向口读写操作	258
9.5	一种典型三极管控制电路	261

9.6	典型发光二极管控制电路	261
9.7	混合电压供电系统 3V/5V 器件 I/O 口互连	262
9.8	如何让 I/O 口上电复位时为低电平	263
9.9	利用 74HC595 驱动 8 个数码管(串行扩展,3 根线)的线路图	264
9.10	I/O 口直接驱动 LED 数码管应用线路图	265
9.11	用 STC 系列 MCU 的 I/O 口直接驱动段码 LCD	266
10	指令系统.....	285
11	中断系统.....	289
11.1	STC8G 系列中断源	289
11.2	STC8G 中断结构图	291
11.3	STC8G 系列中断列表	292
11.4	中断相关寄存器	295
11.4.1	中断使能寄存器 (中断允许位)	297
11.4.2	中断请求寄存器 (中断标志位)	302
11.4.3	中断优先级寄存器	305
11.5	范例程序	309
11.5.1	INT0 中断 (上升沿和下降沿), 可同时支持上升沿和下降沿	309
11.5.2	INT0 中断 (下降沿)	311
11.5.3	INT1 中断 (上升沿和下降沿), 可同时支持上升沿和下降沿	312
11.5.4	INT1 中断 (下降沿)	314
11.5.5	INT2 中断 (下降沿), 只支持下降沿中断	316
11.5.6	INT3 中断 (下降沿), 只支持下降沿中断	318
11.5.7	INT4 中断 (下降沿), 只支持下降沿中断	320
11.5.8	定时器 0 中断	322
11.5.9	定时器 1 中断	323
11.5.10	定时器 2 中断	325
11.5.11	定时器 3 中断	327
11.5.12	定时器 4 中断	330
11.5.13	UART1 中断	332
11.5.14	UART2 中断	334
11.5.15	UART3 中断	337
11.5.16	UART4 中断	339
11.5.17	ADC 中断	342
11.5.18	LVD 中断	344
11.5.19	PCA 中断	346
11.5.20	SPI 中断	349
11.5.21	比较器中断	351
11.5.22	PWM 中断	353
11.5.23	I2C 中断	355
12	定时器/计数器.....	359
12.1	定时器的相关寄存器	359
12.2	定时器 0/1	361
12.2.1	定时器 0/1 控制寄存器 (TCON)	361
12.2.2	定时器 0/1 模式寄存器 (TMOD)	361

12.2.3	定时器 0 模式 0 (16 位自动重装载模式)	362
12.2.4	定时器 0 模式 1 (16 位不可重装载模式)	363
12.2.5	定时器 0 模式 2 (8 位自动重装载模式)	364
12.2.6	定时器 0 模式 3 (不可屏蔽中断 16 位自动重装载, 实时操作系统节拍器)	364
12.2.7	定时器 1 模式 0 (16 位自动重装载模式)	365
12.2.8	定时器 1 模式 1 (16 位不可重装载模式)	366
12.2.9	定时器 1 模式 2 (8 位自动重装载模式)	367
12.2.10	定时器 0 计数寄存器 (TL0, TH0)	367
12.2.11	定时器 1 计数寄存器 (TL1, TH1)	367
12.2.12	辅助寄存器 1 (AUXR)	367
12.2.13	中断与时钟输出控制寄存器 (INTCLKO)	368
12.2.14	定时器 0 定时计算公式.....	368
12.2.15	定时器 1 定时计算公式.....	369
12.3	定时器 2 (24 位定时器, 8 位预分频+16 位定时)	370
12.3.1	辅助寄存器 1 (AUXR)	370
12.3.2	中断与时钟输出控制寄存器 (INTCLKO)	370
12.3.3	定时器 2 计数寄存器 (T2L, T2H)	370
12.3.4	定时器 2 的 8 位预分频寄存器 (TM2PS)	370
12.3.5	定时器 2 工作模式.....	371
12.3.6	定时器 2 计算公式.....	371
12.4	定时器 3/4 (24 位定时器, 8 位预分频+16 位定时)	372
12.4.1	定时器 4/3 控制寄存器 (T4T3M)	372
12.4.2	定时器 3 计数寄存器 (T3L, T3H)	372
12.4.3	定时器 4 计数寄存器 (T4L, T4H)	372
12.4.4	定时器 3 的 8 位预分频寄存器 (TM3PS)	373
12.4.5	定时器 4 的 8 位预分频寄存器 (TM4PS)	373
12.4.6	定时器 3 工作模式.....	373
12.4.7	定时器 4 工作模式.....	375
12.4.8	定时器 3 计算公式.....	375
12.4.9	定时器 4 计算公式.....	376
12.5	范例程序.....	377
12.5.1	定时器 0 (模式 0—16 位自动重载), 用作定时	377
12.5.2	定时器 0 (模式 1—16 位不自动重载), 用作定时	378
12.5.3	定时器 0 (模式 2—8 位自动重载), 用作定时	380
12.5.4	定时器 0 (模式 3—16 位自动重载不可屏蔽中断), 用作定时	382
12.5.5	定时器 0 (外部计数—扩展 T0 为外部下降沿中断)	384
12.5.6	定时器 0 (测量脉宽—INT0 高电平宽度)	386
12.5.7	定时器 0 (模式 0), 时钟分频输出	388
12.5.8	定时器 1 (模式 0—16 位自动重载), 用作定时	390
12.5.9	定时器 1 (模式 1—16 位不自动重载), 用作定时	392
12.5.10	定时器 1 (模式 2—8 位自动重载), 用作定时	393
12.5.11	定时器 1 (外部计数—扩展 T1 为外部下降沿中断)	395
12.5.12	定时器 1 (测量脉宽—INT1 高电平宽度)	397
12.5.13	定时器 1 (模式 0), 时钟分频输出	399

12.5.14	定时器 1 (模式 0) 做串口 1 波特率发生器	401
12.5.15	定时器 1 (模式 2) 做串口 1 波特率发生器	405
12.5.16	定时器 2 (16 位自动重载), 用作定时	409
12.5.17	定时器 2 (外部计数—扩展 T2 为外部下降沿中断)	411
12.5.18	定时器 2, 时钟分频输出	413
12.5.19	定时器 2 做串口 1 波特率发生器	415
12.5.20	定时器 2 做串口 2 波特率发生器	418
12.5.21	定时器 2 做串口 3 波特率发生器	422
12.5.22	定时器 2 做串口 4 波特率发生器	426
12.5.23	定时器 3 (16 位自动重载), 用作定时	430
12.5.24	定时器 3 (外部计数—扩展 T3 为外部下降沿中断)	433
12.5.25	定时器 3, 时钟分频输出	435
12.5.26	定时器 3 做串口 3 波特率发生器	437
12.5.27	定时器 4 (16 位自动重载), 用作定时	441
12.5.28	定时器 4 (外部计数—扩展 T4 为外部下降沿中断)	444
12.5.29	定时器 4, 时钟分频输出	446
12.5.30	定时器 4 做串口 4 波特率发生器	448
13	串口通信.....	453
13.1	串口相关寄存器	453
13.2	串口 1	454
13.2.1	串口 1 控制寄存器 (SCON)	454
13.2.2	串口 1 数据寄存器 (SBUF)	454
13.2.3	电源管理寄存器 (PCON)	455
13.2.4	辅助寄存器 1 (AUXR)	455
13.2.5	串口 1 模式 0, 模式 0 波特率计算公式	455
13.2.6	串口 1 模式 1, 模式 1 波特率计算公式	456
13.2.7	串口 1 模式 2, 模式 2 波特率计算公式	459
13.2.8	串口 1 模式 3, 模式 3 波特率计算公式	460
13.2.9	自动地址识别	460
13.2.10	串口 1 从机地址控制寄存器 (SADDR, SALEN)	460
13.3	串口 2	462
13.3.1	串口 2 控制寄存器 (S2CON)	462
13.3.2	串口 2 数据寄存器 (S2BUF)	462
13.3.3	串口 2 模式 0, 模式 0 波特率计算公式	462
13.3.4	串口 2 模式 1, 模式 1 波特率计算公式	463
13.4	串口 3	465
13.4.1	串口 3 控制寄存器 (S3CON)	465
13.4.2	串口 3 数据寄存器 (S3BUF)	465
13.4.3	串口 3 模式 0, 模式 0 波特率计算公式	465
13.4.4	串口 3 模式 1, 模式 1 波特率计算公式	466
13.5	串口 4	468
13.5.1	串口 4 控制寄存器 (S4CON)	468
13.5.2	串口 4 数据寄存器 (S4BUF)	468
13.5.3	串口 4 模式 0, 模式 0 波特率计算公式	468

13.5.4	串口 4 模式 1, 模式 1 波特率计算公式	469
13.6	串口注意事项	471
13.7	范例程序	472
13.7.1	串口 1 使用定时器 2 做波特率发生器	472
13.7.2	串口 1 使用定时器 1 (模式 0) 做波特率发生器	475
13.7.3	串口 1 使用定时器 1 (模式 2) 做波特率发生器	479
13.7.4	串口 2 使用定时器 2 做波特率发生器	483
13.7.5	串口 3 使用定时器 2 做波特率发生器	487
13.7.6	串口 3 使用定时器 3 做波特率发生器	491
13.7.7	串口 4 使用定时器 2 做波特率发生器	495
13.7.8	串口 4 使用定时器 4 做波特率发生器	499
13.7.9	串口多机通讯	504
13.7.10	串口转 LIN 总线	505
14	比较器, 掉电检测, 内部 1.19V 参考信号源	514
14.1	比较器内部结构图	514
14.2	比较器相关的寄存器	515
14.2.1	比较器控制寄存器 1 (CMPCR1)	515
14.2.2	比较器控制寄存器 2 (CMPCR2)	516
14.3	范例程序	517
14.3.1	比较器的使用 (中断方式)	517
14.3.2	比较器的使用 (查询方式)	519
14.3.3	比较器的多路复用应用 (比较器+ADC 输入通道)	522
14.3.4	比较器作外部掉电检测 (掉电过程中应及时保存用户数据到 EEPROM 中)	524
14.3.5	比较器检测工作电压 (电池电压)	525
15	IAP/EEPROM/DATA-FLASH	530
15.1	EEPROM 操作时间	530
15.2	EEPROM 相关的寄存器	530
15.2.1	EEPROM 数据寄存器 (IAP_DATA)	530
15.2.2	EEPROM 地址寄存器 (IAP_ADDR)	531
15.2.3	EEPROM 命令寄存器 (IAP_CMD)	531
15.2.4	EEPROM 触发寄存器 (IAP_TRIG)	531
15.2.5	EEPROM 控制寄存器 (IAP_CONTR)	531
15.2.6	EEPROM 等待时间控制寄存器 (IAP_TPS)	532
15.3	EEPROM 大小及地址	533
15.4	范例程序	537
15.4.1	EEPROM 基本操作	537
15.4.2	使用 MOVC 读取 EEPROM	540
15.4.3	使用串口送出 EEPROM 数据	544
15.4.4	串口 1 读写 EEPROM-带 MOVC 读	548
15.4.5	口令擦除写入-多扇区备份-串口 1 操作	555
16	ADC 模数转换, 内部 1.19V 参考信号源	564
16.1	ADC 相关的寄存器	564
16.1.1	ADC 控制寄存器 (ADC_CONTR), PWM 触发 ADC 控制	564
16.1.2	ADC 配置寄存器 (ADCCFG)	567

16.1.3	ADC 转换结果寄存器 (ADC_RES, ADC_RESL)	568
16.1.4	ADC 时序控制寄存器.....	569
16.2	ADC 相关计算公式.....	570
16.2.1	ADC 速度计算公式.....	570
16.2.2	ADC 转换结果计算公式.....	570
16.2.3	反推 ADC 输入电压计算公式.....	571
16.2.4	反推工作电压计算公式.....	572
16.3	10 位 ADC 静态特性.....	572
16.4	12 位 ADC 静态特性.....	572
16.5	ADC 应用参考线路图.....	573
16.5.1	无独立 VREF 脚参考线路图.....	573
16.5.2	有独立 VREF 脚、一般精度 ADC 参考线路图.....	574
16.5.3	有独立 VREF 脚、高精度 ADC 参考线路图.....	575
16.6	范例程序	576
16.6.1	ADC 基本操作 (查询方式)	576
16.6.2	ADC 基本操作 (中断方式)	578
16.6.3	格式化 ADC 转换结果.....	580
16.6.4	利用 ADC 第 15 通道测量外部电压或电池电压.....	583
16.6.5	ADC 做电容感应触摸按键.....	586
16.6.6	ADC 作按键扫描应用线路图.....	598
16.6.7	检测负电压参考线路图	600
16.6.8	常用加法电路在 ADC 中的应用.....	601
17	PCA/CCP/PWM 应用	602
17.1	PCA 相关的寄存器	602
17.1.1	PCA 控制寄存器 (CCON)	603
17.1.2	PCA 模式寄存器 (CMOD)	603
17.1.3	PCA 计数器寄存器 (CL, CH)	603
17.1.4	PCA 模块模式控制寄存器 (CCAPMn)	604
17.1.5	PCA 模块模式捕获值/比较值寄存器 (CCAPnL, CCAPnH)	604
17.1.6	PCA 模块 PWM 模式控制寄存器 (PCA_PWMn)	604
17.2	PCA 工作模式	606
17.2.1	捕获模式.....	606
17.2.2	软件定时器模式.....	606
17.2.3	高速脉冲输出模式.....	607
17.2.4	PWM 脉宽调制模式及频率计算公式.....	607
17.3	利用 CCP/PCA/PWM 模块实现 8~16 位 DAC 的参考线路图.....	611
17.4	范例程序	612
17.4.1	PCA 输出 PWM (6/7/8/10 位)	612
17.4.2	PCA 捕获测量脉冲宽度	615
17.4.3	PCA 实现 16 位软件定时	619
17.4.4	PCA 实现 16 位软件定时 (ECI 外部时钟模式)	622
17.4.5	PCA 输出高速脉冲	625
17.4.6	PCA 扩展外部中断	628
18	精度可达 15 位的增强型 PWM (最多可输出 45 路各自独立的 PWM)	632

18.1	PWM 相关的寄存器	633
18.1.1	增强型 PWM 全局配置寄存器 (PWMSET)	640
18.1.2	增强型 PWM 配置寄存器 (PWMCFGn)	641
18.1.3	PWM 中断标志寄存器 (PWMMnIF)	642
18.1.4	PWM 异常检测控制寄存器 (PWMMnFDCR)	642
18.1.5	PWM 计数器寄存器 (PWMMnCH, PWMMnCL)	643
18.1.6	PWM 时钟选择寄存器 (PWMMnCKS), 输出频率计算公式	644
18.1.7	PWM 触发 ADC 计数器寄存器 (PWMMnTADC)	644
18.1.8	PWM 电平输出设置计数值寄存器 (PWMMnT1, PWMMnT2)	645
18.1.9	PWM 通道控制寄存器 (PWMMnCR)	649
18.1.10	PWM 通道电平保持控制寄存器 (PWMMnHLD)	651
18.2	范例程序	653
18.2.1	输出任意周期和任意占空比的波形	653
18.2.2	两路 PWM 实现互补对称带死区控制的波形	655
18.2.3	PWM 实现渐变灯 (呼吸灯)	659
18.2.4	使用 PWM 触发 ADC 转换	663
18.2.5	产生 3 路相位差 120 度的互补带死区的 PWM 波形	668
18.2.6	输出占空比为 100% (固定输出高) 和 0% (固定输出低) 的 PWM 波形的方法 (以 PWM00 为例)	670
18.2.7	增强型 PWM-频率可调-脉冲计数	671
18.2.8	增强型 PWM 时钟输出应用 (系统时钟 2 分频输出)	674
19	同步串行外设接口 SPI.....	676
19.1	SPI 相关的寄存器	676
19.1.1	SPI 状态寄存器 (SPSTAT)	676
19.1.2	SPI 控制寄存器 (SPCTL), SPI 速度控制	676
19.1.3	SPI 数据寄存器 (SPDAT)	677
19.2	SPI 通信方式	678
19.2.1	单主单从	678
19.2.2	互为主从	678
19.2.3	单主多从	679
19.3	配置 SPI	680
19.4	数据模式	682
19.5	范例程序	683
19.5.1	SPI 单主单从系统主机程序 (中断方式)	683
19.5.2	SPI 单主单从系统从机程序 (中断方式)	685
19.5.3	SPI 单主单从系统主机程序 (查询方式)	687
19.5.4	SPI 单主单从系统从机程序 (查询方式)	689
19.5.5	SPI 互为主从系统程序 (中断方式)	692
19.5.6	SPI 互为主从系统程序 (查询方式)	694
20	I²C 总线	698
20.1	I ² C 相关的寄存器	698
20.2	I ² C 主机模式	699
20.2.1	I ² C 配置寄存器 (I2CCFG), 总线速度控制	699
20.2.2	I ² C 主机控制寄存器 (I2CMSCR)	700

20.2.3	I ² C 主机辅助控制寄存器 (I2CMSAUX)	701
20.2.4	I ² C 主机状态寄存器 (I2CMSST)	701
20.3	I ² C 从机模式.....	703
20.3.1	I ² C 从机控制寄存器 (I2CSLCR)	703
20.3.2	I ² C 从机状态寄存器 (I2CSLST)	703
20.3.3	I ² C 从机地址寄存器 (I2CSLADR)	705
20.3.4	I ² C 数据寄存器 (I2CTXD, I2CRXD)	706
20.4	范例程序	707
20.4.1	I ² C 主机模式访问 AT24C256 (中断方式)	707
20.4.2	I ² C 主机模式访问 AT24C256 (查询方式)	713
20.4.3	I ² C 主机模式访问 PCF8563.....	719
20.4.4	I ² C 从机模式 (中断方式)	724
20.4.5	I ² C 从机模式 (查询方式)	729
20.4.6	测试 I ² C 从机模式代码的主机代码.....	733
21	触摸按键控制器, 先工程样片, 可少量送样	740
21.1	触摸按键控制器内部框架图	741
21.2	低功耗触摸唤醒	742
21.3	仅做触摸按键功能时的操作步骤	742
21.4	低功耗触摸按键唤醒功能的操作步骤	742
21.5	触摸按键相关的寄存器	743
21.5.1	触摸按键使能寄存器 (TSCHENn)	744
21.5.2	触摸按键配置寄存器 (TSCFGn)	744
21.5.3	触摸按键低电模式唤醒时间控制寄存器 (TSWUTC)	745
21.5.4	触摸按键控制寄存器 (TSCTRL)	745
21.5.5	触摸按键状态寄存器 1 (TSSTA1)	746
21.5.6	触摸按键状态寄存器 2 (TSSTA2)	747
21.5.7	触摸按键时间控制寄存器 (TSRT)	748
21.5.8	触摸按键数据寄存器 (TSDAT)	748
21.5.9	触摸按键门槛值寄存器 (TSTH)	748
21.6	基本参考电路图以及注意事项	749
22	LED 驱动器.....	750
22.1	LED 驱动器内部框架图	751
22.2	LED 驱动相关的寄存器	751
22.2.1	COM 使能寄存器 (COMEN)	752
22.2.2	SEG 使能寄存器 (SEGEN)	752
22.2.3	LED 控制寄存器 (LEDCTRL)	752
22.2.4	LED 时钟分频寄存器 (LEDCKS)	753
22.2.5	LED 共阳模式数据寄存器 (COMn_DA)	754
22.2.6	LED 共阴模式数据寄存器 (COMn_DC)	754
22.3	LED 共阴模式 (LEDMODE = 00)	755
22.4	LED 共阳模式 (LEDMODE = 01)	756
22.5	LED 共阴/共阳模式 (LEDMODE = 10)	757
22.6	触摸按键与 LED 分时复用 I/O	758
22.7	共阴模式参考电路图	760

22.8	共阳模式参考电路图	760
22.9	共阴/共阳模式参考电路图	761
22.10	范例程序	762
22.10.1	共阴/共阳模式驱动 16 个 7 段数码管	762
23	增强型双数据指针	765
23.1	相关的特殊功能寄存器	765
23.1.1	第 1 组 16 位数据指针寄存器 (DPTR0)	765
23.1.2	第 2 组 16 位数据指针寄存器 (DPTR1)	765
23.1.3	数据指针控制寄存器 (DPS)	765
23.1.4	数据指针控制寄存器 (TA)	766
23.2	范例程序	768
23.2.1	示例代码 1	768
23.2.2	示例代码 2	769
24	MDU16 硬件 16 位乘除法器	771
24.1	相关的特殊功能寄存器	771
24.1.1	操作数 1 数据寄存器 (MD0~MD3)	771
24.1.2	操作数 2 数据寄存器 (MD4~MD5)	772
24.1.3	MDU 模式控制寄存器 (ARCON), 运算所需时钟数	772
24.1.4	MDU 操作控制寄存器 (OPCON)	773
24.2	关于 MDU16 的网友应用杂谈 (提供思路, 仅供参考)	774
24.3	范例程序	776
附录 A	编译器 (汇编器) /仿真器/头文件使用指南	778
附录 B	STC-ISP 下载软件高级应用	787
B.1	发布项目程序	787
B.2	程序加密后传输 (防烧录时串口分析出程序)	791
B.3	发布项目程序+程序加密后传输结合使用	795
B.4	用户自定义下载 (实现不停电下载)	796
附录 C	如何测试 I/O 口	800
附录 D	如何让传统的 8051 单片机学习板可仿真	801
附录 E	STC-USB 驱动程序安装说明	803
附录 F	USB 下载步骤演示	866
附录 G	RS485 自动控制或 I/O 口控制线路图	870
附录 H	STC 工具使用说明书	871
H.1	概述	871
H.2	系统可编程 (ISP) 流程说明	871
H.3	USB 型联机/脱机下载工具 U8W/U8W-Mini	872
H.3.1	安装 U8W/U8W-Mini 驱动程序	874
H.3.2	U8W 的功能介绍	877
H.3.3	U8W 的在线联机下载使用说明	878
H.3.4	U8W 的脱机下载使用说明	881
H.3.5	U8W-Mini 的功能介绍	889
H.3.6	U8W-Mini 的在线联机下载使用说明	890
H.3.7	U8W-Mini 的脱机下载使用说明	891
H.3.8	制作/更新 U8W/U8W-Mini	897

H.3.9	U8W/U8W-Mini 设置直通模式（可用于仿真）	899
H.3.10	U8W/U8W-Mini 的参考电路.....	899
H.4	STC 通用 USB 转串口工具	901
H.4.1	STC 通用 USB 转串口工具外观图	901
H.4.2	STC 通用 USB 转串口工具布局图	902
H.4.3	STC 通用 USB 转串口工具驱动安装	903
H.4.4	使用 STC 通用 USB 转串口工具下载程序到 MCU	904
H.4.5	使用 STC 通用 USB 转串口工具仿真用户代码	906
H.5	应用线路图	913
H.5.1	U8W 工具应用参考线路图	913
H.5.2	STC 通用 USB 转串口工具应用参考线路图	913
附录 I	STC 仿真使用说明书	915
I.1	概述	915
I.2	安装 Keil 软件	916
I.3	安装仿真驱动	917
I.4	串口直接仿真	920
I.4.1	制作串口仿真芯片	920
I.4.2	在 Keil 软件中进行串口仿真设置	923
I.4.3	在 Keil 软件中使用串口进行仿真	925
I.5	USB 直接仿真（目前只有 STC8H8K64U-B 版本芯片支持）	927
I.5.1	制作 USB 仿真芯片	927
I.5.2	在 Keil 软件中进行 USB 仿真设置	931
I.5.3	在 Keil 软件中使用 USB 进行仿真	933
附录 J	U8W 下载工具中 RS485 部分线路图	935
附录 K	运行用户程序时收到用户命令后自动启动 ISP 下载(不停电)	936
附录 L	使用 STC 的 IAP 系列单片机开发自己的 ISP 程序	938
附录 M	用户程序复位到系统区进行 ISP 下载的方法（不停电）	950
附录 N	使用第三方 MCU 对 STC8G 系列单片机进行 ISP 下载范例程序	956
附录 O	使用第三方应用程序调用 STC 发布项目程序对单片机进行 ISP 下载	964
附录 P	在 Keil 中建立多文件项目的方法	968
附录 Q	关于中断号大于 31 在 Keil 中编译出错的处理	972
附录 R	电气特性	982
R.1	绝对最大额定值	982
R.2	直流特性 (3.3V)	983
R.3	直流特性 (5.0V)	985
R.4	内部 IRC 温漂特性（参考温度 25°C）	986
R.5	低压复位门槛电压（测试温度 25°C）	986
附录 S	应用注意事项	987
S.1	STC8G1K08A 系列	987
S.2	STC8G2K64S4/S2 系列	987
S.3	STC8G1K08 系列	988
S.4	STC8G1K08T 系列	988
附录 T	触摸按键的 PCB 设计指导	989
附录 U	QFN/DFN 封装元器件焊接方法	991

附录 V	关于回流焊前是否要烘烤	994
附录 W	如何使用万用表检测芯片 I/O 口好坏	995
附录 X	STC8G 系列单片机取代 STC15 系列的注意事项	996
附录 Y	STC8G 系列单片机取代 STC8A/8F 系列的注意事项	998
附录 Z	STC15H 系列单片机取代 STC15F/L/W 系列的注意事项	999
附录 AA	更新记录	1001
附录 BB	STC8 系列命名花絮	1011

STC MCU

1 概述

STC8G 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，STC8G 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，STC8G 系列单片机仅需 147 个时钟，而传统 8051 则需要 1944 个时钟。STC8G 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 $+25^{\circ}\text{C}$)， $-1.38\% \sim +1.42\%$ 温飘($-40^{\circ}\text{C} \sim +85^{\circ}\text{C}$)， $-0.88\% \sim +1.05\%$ 温飘($-20^{\circ}\text{C} \sim +65^{\circ}\text{C}$)。ISP 编程时 4MHz~35MHz 宽范围可设置（注意：温度范围为 $-40^{\circ}\text{C} \sim +85^{\circ}\text{C}$ 时，最高频率须控制在 35MHz 以下），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：内部高精度 IRC 时钟（ISP 下载时可进行调节）、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 1.0mA（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 0.6uA@Vcc=5.0V，0.4uA@Vcc=3.3V。

掉电模式可以使用 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)、RXD(P3.0/P3.6/P1.6/P4.3)、RXD2(P1.0/P4.6)、RXD3(P0.0/P5.0)、RXD4(P0.2/P5.2)、CCP0(P1.1/P3.5/P2.5)、CCP1(P1.0/P3.6/P2.6)、CCP2(P3.7/P2.7)、I2C_SDA(P1.4/P2.4/P3.3) 以及比较器中断、低压检测中断、掉电唤醒定时器唤醒。

MCU 提供了丰富的数字外设（串口、定时器、PCA、PWM 以及 I²C、SPI）接口与模拟外设（超高速 ADC、比较器），可满足广大用户的设计需求。

STC8G 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	I/O	UART	定时器	ADC	增强型 PWM	PCA	CMP	SPI	I2C	MDU16	LED	触摸按键
STC8G1K08 系列	18	2	3	15_{CH}*10_B		●	●	●	●			
STC8G1K08-8Pin 系列	6	1	2					●	●	●		
STC8G1K08A 系列	6	1	2	6_{CH}*10_B		●		●	●	●		
STC8G2K64S4 系列	45	4	5	15_{CH}*10_B	●	●	●	●	●	●		
STC8G2K64S2 系列	45	2	5	15_{CH}*10_B	●	●	●	●	●	●		
STC8G1K08T 系列	16	1	3	15_{CH}*10_B		●	●	●	●		●	●
STC15H2K64S4 系列	42	4	5	15CH*10B	●	●	●	●	●	●		

2 特性、价格及管脚

STCMCU

2.1 STC8G1K08-36I-SOP8/DFN8 系列

2.1.1 特性及价格(有 16 位硬件乘除法器 MDU16, 准 16 位单片机)

➤ 选型价格 (不需要外部晶振、不需要外部复位)

单片机型号	工作电压 (V)	主力产品供货信息																									
		DFN8<3mm*3mm>		SOP8		价格及封装																					
STC8G1K08	1.9-5.5	8K	256	1K	2	4K	6	1	有	有	有	2	-	有	有	4 级	有	是	有	是	是	是	是	¥0.89	¥0.95	现货	
STC8G1K17	1.9-5.5	17K	256	1K	2	IAP	6	1	有	有	有	2	-	有	-	有	4 级	有	是	有	是	是	-	-	¥0.99	¥1.05	

注: 以上的单价为 10K 及以上订货量的价格, 量小则每片需增加 0.1 元人民币。当订货的总额达到或高于 3000 元时, 可免运费发货, 否则需要由客户承担运费。零售 10 片起售。

➤ 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
- ✓ 11 个中断源, 4 级中断优先级
- ✓ 支持在线仿真

➤ 工作电压

- ✓ 1.9V~5.5V
- ✓ 内建 LDO

➤ 工作温度

- ✓ -40°C~85°C (超温度范围应用请参考电器特性章节说明)

➤ Flash 存储器

- ✓ 最大 17K 字节 FLASH 程序存储器 (ROM), 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)
- ✓ 1024 字节内部扩展 RAM (内部 XDATA)

➤ 时钟控制

- ✓ 内部高精度 IRC (4MHz~38MHz, ISP 编程时可进行上下调整, 还可以用户软件分频到较低的频率工作, 如 100KHz)

- ◆ 误差±0.3%（常温下 25°C）
- ◆ -1.38%~+1.42% 温漂（全温度范围，-40°C~85°C）
- ◆ -0.88%~+1.05% 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。（在芯片未使能低压复位功能时有效）
 - ◆ 复位脚复位。出厂时 P5.4 默认为 I/O 口，ISP 下载时可将 P5.4 管脚设置为复位脚（注意：当设置 P5.4 管脚为复位脚时，复位电平为低电平）
 - ◆ 看门狗溢出复位
 - ◆ 低压检测复位，提供 4 级低压检测电压：2.0V、2.4V、2.7V、3.0V。
- ✓ 软件复位
 - ◆ 软件方式写复位触发寄存器

➤ 中断

- ✓ 提供 11 个中断源：INT0（支持上升沿和下降沿中断）、INT1（支持上升沿和下降沿中断）、INT2（只支持下降沿中断）、INT3（只支持下降沿中断）、INT4（只支持下降沿中断）、定时器 0、定时器 1、串口 1、LVD 低压检测、SPI、I²C
- ✓ 提供 4 级中断优先级
- ✓ 时钟停振模式下可以唤醒的中断：INT0(P3.2)、INT1(P3.3)、INT2(P3.6)、INT3(P3.7)、INT4(P3.0)、T0(P3.4)、T1(P3.5)、RXD(P3.0/P3.2/P5.4)、I2C_SDA(P3.3/P5.5) 以及低压检测中断、掉电唤醒定时器唤醒。

➤ 数字外设

- ✓ 2 个 16 位定时器：定时器 0、定时器 1，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
- ✓ 1 个高速串口：串口 1，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
- ✓ **MDU16：硬件 16 位乘除法器（支持 32 位除以 16 位、16 位除以 16 位、16 位乘 16 位、数据移位以及数据规格化等运算）**

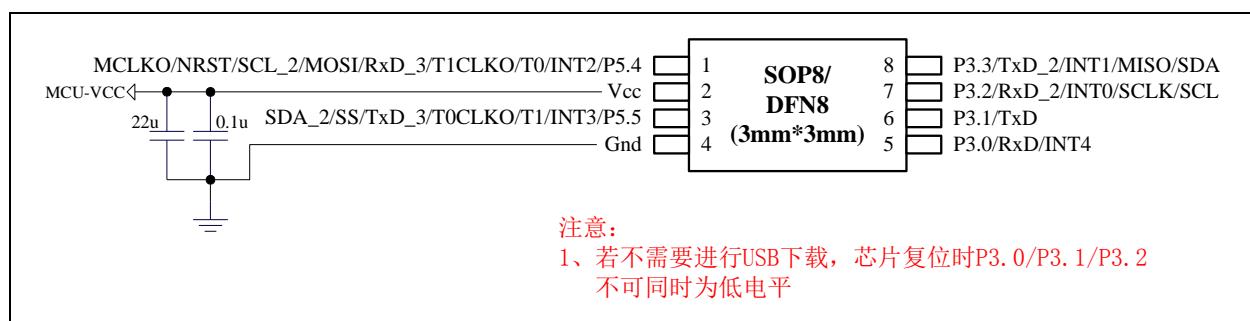
➤ GPIO

- ✓ 最多可达 6 个 GPIO：P3.0~P3.3、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
- ✓ **除 P3.0 和 P3.1 外，其余所有 I/O 口上电后的状态均为高阻输入状态，用户在使用 I/O 口时必须先设置 I/O 口模式，另外每个 I/O 均可独立使能内部 4K 上拉电阻**

➤ 封装

- ✓ SOP8，DFN8（3mm*3mm）

2.1.2 管脚图, 最小系统



2.1.3 管脚说明

编号		名称	类型	说明
SOP8				
1		P5.4	I/O	标准 I/O 口
		NRST	I	复位引脚 (低电平复位)
		MCLKO	O	主时钟分频输出
		INT2	I	外部中断 2
		T0	I	定时器 0 外部时钟输入
		T1CLKO	O	定时器 1 时钟分频输出
		RxD_3	I	串口 1 的接收脚
		MOSI	I/O	SPI 主机输出从机输入
		SCL_2	I/O	I2C 的时钟线
2		Vcc	Vcc	电源脚
3		P5.5	I/O	标准 I/O 口
		INT3	I	外部中断 3
		T1	I	定时器 1 外部时钟输入
		T0CLKO	O	定时器 0 时钟分频输出
		TxD_3	O	串口 1 的发送脚
		SS	I	SPI 的从机选择脚 (主机为输出)
		SDA_2	I/O	I2C 的数据线
4		Gnd	Gnd	地线
5		P3.0	I/O	标准 I/O 口
		RxD	I	串口 1 的接收脚
		INT4	I	外部中断 4
6		P3.1	I/O	标准 I/O 口
		TxD	O	串口 1 的发送脚
7		P3.2	I/O	标准 I/O 口
		INT0	I	外部中断 0
		SCLK	I/O	SPI 的时钟脚
		SCL	I/O	I2C 的时钟线
		RxD_2	I	串口 1 的接收脚
8		P3.3	I/O	标准 I/O 口
		INT1	I	外部中断 1
		MISO	I/O	SPI 主机输入从机输出
		SDA	I/O	I2C 的数据线
		TxD_2	O	串口 1 的发送脚

2.2 STC8G1K08A-36I-SOP8/DFN8/DIP8 系列

2.2.1 特性及价格(有 16 位硬件乘除法器 MDU16, 准 16 位单片机)

➤ 选型价格 (不需要外部晶振、不需要外部复位, 10 位 ADC, 6 通道)

单片机型号	工作电压 (V)	价格及封装												主刀产品供货信息						
		DIP8 (不建议使用)				SOP8				DFN8<3mm*3mm>										
STC8G1K08A	1.9-5.5	8K	256	1K	2	4K	6	1	有	有	2	3	有	10位	有 4级	是 有	是 是	是 是	是 是	¥0.99 ¥1.05 ¥1.3
STC8G1K17A	1.9-5.5	17K	256	1K	2	IAP	6	1	有	有	2	3	有	10位	有 4级	是 有	是 是	是 是	- - - -	¥1.1 ¥1.15 ¥1.4

注: 以上的单价为 10K 及以上订货量的价格, 量小则每片需增加 0.1 元人民币。当订货的总额达到或高于 3000 元时, 可免运费发货, 否则需要由客户承担运费。零售 10 片起售。

➤ 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
- ✓ 13 个中断源, 4 级中断优先级
- ✓ 支持在线仿真

➤ 工作电压

- ✓ 1.9V~5.5V
- ✓ 内建 LDO

➤ 工作温度

- ✓ -40°C~85°C (超温度范围应用请参考电器特性章节说明)

➤ Flash 存储器

- ✓ 最大 17K 字节 FLASH 程序存储器 (ROM), 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)
- ✓ 1024 字节内部扩展 RAM (内部 XDATA)

➤ 时钟控制

- ✓ 内部高精度 IRC (4MHz~38MHz, ISP 编程时可进行上下调整, 还可以用户软件分频到较低的频率工作, 如 100KHz)

- ◆ 误差±0.3%（常温下 25°C）
- ◆ -1.38%~+1.42% 温漂（全温度范围，-40°C~85°C）
- ◆ -0.88%~+1.05% 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。（在芯片未使能低压复位功能时有效）
 - ◆ 复位脚复位。出厂时 P5.4 默认为 I/O 口，ISP 下载时可将 P5.4 管脚设置为复位脚（注意：当设置 P5.4 管脚为复位脚时，复位电平为低电平）
 - ◆ 看门狗溢出复位
 - ◆ 低压检测复位，提供 4 级低压检测电压：2.0V、2.4V、2.7V、3.0V。
 - ✓ 软件复位
 - ◆ 软件方式写复位触发寄存器

- 中断
 - ✓ 提供 13 个中断源：INT0（支持上升沿和下降沿中断）、INT1（支持上升沿和下降沿中断）、INT2（只支持下降沿中断）、INT3（只支持下降沿中断）、INT4（只支持下降沿中断）、定时器 0、定时器 1、串口 1、ADC 模数转换、LVD 低压检测、SPI、I²C、PCA/CCP/PWM
 - ✓ 提供 4 级中断优先级
 - ✓ 时钟停振模式下可以唤醒的中断：INT0(P3.2)、INT1(P3.3)、INT2(P3.6)、INT3(P3.7)、INT4(P3.0)、T0(P3.4)、T1(P3.5)、RXD(P3.0/P3.2/P1.6/P5.4)、CCP0(P3.2/P3.1)、CCP1(P3.3)、CCP2(P5.4/P5.5)、I2C_SDA(P3.3/P5.5) 以及低压检测中断、掉电唤醒定时器唤醒。

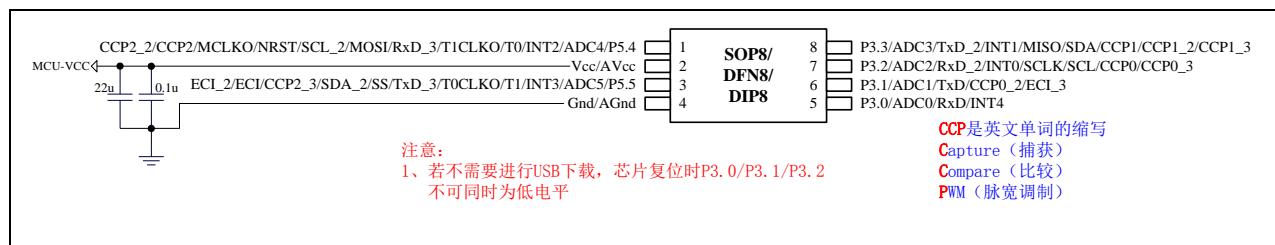
- 数字外设
 - ✓ 2 个 16 位定时器：定时器 0、定时器 1、其中定时器 0 的模式 3 具有 NMI（不可屏蔽中断）功能，定时器 0 和定时器 1 的模式 0 为 16 位自动重载模式
 - ✓ 1 个高速串口：串口 1，波特率时钟源最快可为 FOSC/4
 - ✓ 3 组 16 位 CCP/PCA/PWM 模块：CCP0、CCP1、CCP2，可用于捕获、高速脉冲输出，及 6/7/8/10 位的 PWM 输出
 - ✓ SPI：支持主机模式和从机模式以及主机/从机自动切换
 - ✓ I²C：支持主机模式和从机模式
 - ✓ **MDU16：硬件 16 位乘除法器（支持 32 位除以 16 位、16 位除以 16 位、16 位乘 16 位、数据移位以及数据规格化等运算）**

- 模拟外设
 - ✓ 超高速 ADC，支持 **10 位精度** 6 通道（通道 0~通道 5）的模数转换，**速度最快能达到 500K（每秒进行 50 万次 ADC 转换）**
 - ✓ ADC 的通道 15 用于测试内部 1.19V 参考信号源（芯片在出厂时，内部参考信号源已调整为 1.19V）
 - ✓ DAC：3 路 PCA/CCP/PWM 可当 3 路 DAC 使用

- GPIO
 - ✓ 最多可达 6 个 GPIO：P3.0~P3.3、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
 - ✓ **除 P3.0 和 P3.1 外，其余所有 I/O 口上电后的状态均为高阻输入状态，用户在使用 I/O 口时必须先设置 I/O 口模式，另外每个 I/O 均可独立使能内部 4K 上拉电阻**

- 封装
 - ✓ SOP8，DFN8（3mm*3mm），DIP8

2.2.2 管脚图, 最小系统



2.2.3 管脚说明

编号		名称	类型	说明
SOP8	DFN8			
1		P5.4	I/O	标准 I/O 口
		NRST	I	复位引脚（低电平复位）
		MCLKO	O	主时钟分频输出
		INT2	I	外部中断 2
		T0	I	定时器 0 外部时钟输入
		T1CLKO	O	定时器 1 时钟分频输出
		RxD_3	I	串口 1 的接收脚
		MOSI	I/O	SPI 主机输出从机输入
		SCL_2	I/O	I2C 的时钟线
		ADC4	I	ADC 模拟输入通道 4
		CCP2	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
		CCP2_2	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
2		Vcc	Vcc	电源脚
		AVcc	Vcc	ADC 电源脚
3		P5.5	I/O	标准 I/O 口
		INT3	I	外部中断 3
		T1	I	定时器 1 外部时钟输入
		T0CLKO	O	定时器 0 时钟分频输出
		TxD_3	O	串口 1 的发送脚
		SS	I	SPI 的从机选择脚（主机为输出）
		SDA_2	I/O	I2C 的数据线
		ADC5	I	ADC 模拟输入通道 5
		ECI	I	PCA 的外部脉冲输入
		ECI_2	I	PCA 的外部脉冲输入
4		Gnd	Gnd	地线
		AGnd	Gnd	ADC 地线
5		P3.0	I/O	标准 I/O 口
		RxD	I	串口 1 的接收脚
		INT4	I	外部中断 4
		ADC0	I	ADC 模拟输入通道 0
6		P3.1	I/O	标准 I/O 口
		TxD	O	串口 1 的发送脚
		ADC1	I	ADC 模拟输入通道 1
		ECI_3	I	PCA 的外部脉冲输入
		CCP0_2	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出

编号		名称	类型	说明
SOP8	DFN8			
7		P3.2	I/O	标准 I/O 口
		INT0	I	外部中断 0
		SCLK	I/O	SPI 的时钟脚
		SCL	I/O	I2C 的时钟线
		RxD_2	I	串口 1 的接收脚
		ADC2	I	ADC 模拟输入通道 2
		CCP0	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
		CCP0_3	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
8		P3.3	I/O	标准 I/O 口
		INT1	I	外部中断 1
		MISO	I/O	SPI 主机输入从机输出
		SDA	I/O	I2C 的数据线
		TxD_2	O	串口 1 的发送脚
		ADC3	I	ADC 模拟输入通道 3
		CCP1	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
		CCP1_2	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
		CCP1_3	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出

2.3 STC8G1K08-38I-TSSOP20/QFN20/SOP16 系列

2.3.1 特性及价格

➤ 选型价格（不需要外部晶振、不需要外部复位，10位ADC，15通道）

➤ 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

- ✓ 1.9V~5.5V
 - ✓ 内建 LDO

工作温度

- ✓ -40°C~85°C (超温度范围应用请参考电器特性章节说明)

➤ Flash 存储器

- ✓ 最大 17K 字节 FLASH 程序存储器（ROM），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)
 - ✓ 1024 字节内部扩展 RAM (内部 XDATA)

➤ 时钟控制

- ✓ 内部高精度 IRC (4MHz~36MHz, ISP 编程时可进行上下调整, 还可以用户软件分频到较低的频率工作, 如 100KHz)

- ◆ 误差±0.3%（常温下 25°C）
- ◆ -1.38%~+1.42% 温漂（全温度范围，-40°C~85°C）
- ◆ -0.88%~+1.05% 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~36MHz）和外部时钟

- **复位**
- ✓ 硬件复位
 - ◆ 上电复位。（在芯片未使能低压复位功能时有效）
 - ◆ 复位脚复位。出厂时 P5.4 默认为 I/O 口, ISP 下载时可将 P5.4 管脚设置为复位脚（注意：当设置 P5.4 管脚为复位脚时，复位电平为低电平）
 - ◆ 看门狗溢出复位
 - ◆ 低压检测复位，提供 4 级低压检测电压：2.0V、2.4V、2.7V、3.0V。
- ✓ 软件复位
 - ◆ 软件方式写复位触发寄存器

- **中断**
- ✓ 提供 16 个中断源：INT0（支持上升沿和下降沿中断）、INT1（支持上升沿和下降沿中断）、INT2（只支持下降沿中断）、INT3（只支持下降沿中断）、INT4（只支持下降沿中断）、定时器 0、定时器 1、定时器 2、串口 1、串口 2、ADC 模数转换、LVD 低压检测、SPI、I²C、比较器、PCA/CCP/PWM
- ✓ 提供 4 级中断优先级
- ✓ 时钟停振模式下可以唤醒的中断：INT0(P3.2)、INT1(P3.3)、INT2(P3.6)、INT3(P3.7)、INT4(P3.0)、T0(P3.4)、T1(P3.5)、T2(P1.2)、RXD(P3.0/P3.6/P1.6)、RXD2(P1.0)、CCP0(P1.1/P3.5)、CCP1(P1.0/P3.6)、CCP2(P3.7)、I2C_SDA(P1.4/P3.3) 以及比较器中断、低压检测中断、掉电唤醒定时器唤醒。

- **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
- ✓ 3 组 16 位 CCP/PCA/PWM 模块：CCP0、CCP1、CCP2，可用于捕获、高速脉冲输出，及 6/7/8/10 位的 PWM 输出
- ✓ SPI：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 **10 位精度** 15 通道（通道 0~通道 14）的模数转换，**速度最快能达到 500K (每秒进行 50 万次 ADC 转换)**
- ✓ ADC 的通道 15 用于测试内部 1.19V 参考信号源（芯片在出厂时，内部参考信号源已调整为 1.19V）
- ✓ 比较器，一组比较器（比较器的正端可选择 CMP+ 端口和所有的 ADC 输入端口，所以比较器可当作多路比较器进行分时复用）
- ✓ DAC：3 路 PCA/CCP/PWM 可当 3 路 DAC 使用

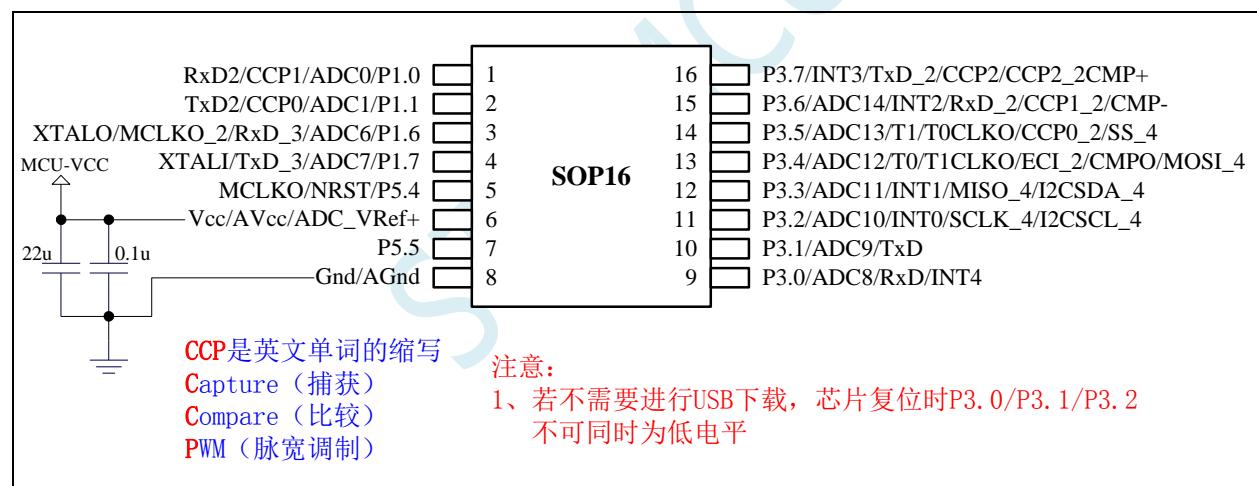
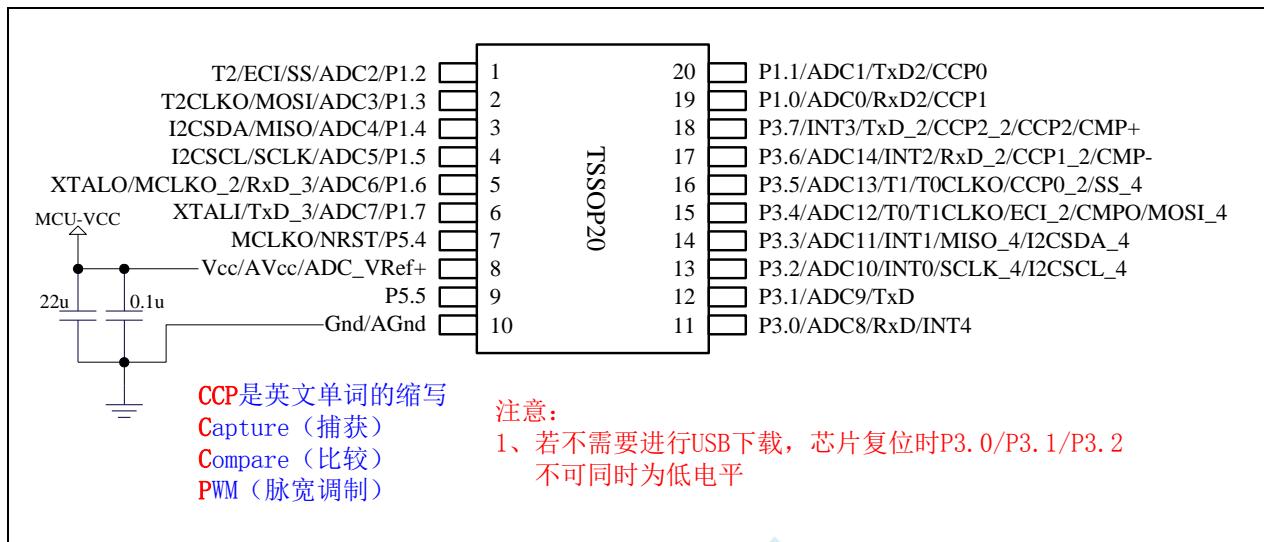
- **GPIO**
- ✓ 最多可达 18 个 GPIO：P1.0~P1.7、P3.0~P3.7、P5.4~P5.5
- ✓ 所有的 GPIO 均支持如下 4 种模式：准双向口模式、强推挽输出模式、开漏输出模式、高阻输入模式
- ✓ **除 P3.0 和 P3.1 外，其余所有 I/O 口上电后的状态均为高阻输入状态，用户在使用 I/O 口时必须先设置 I/O 口模式，另外每个 I/O 均可独立使能内部 4K 上拉电阻**

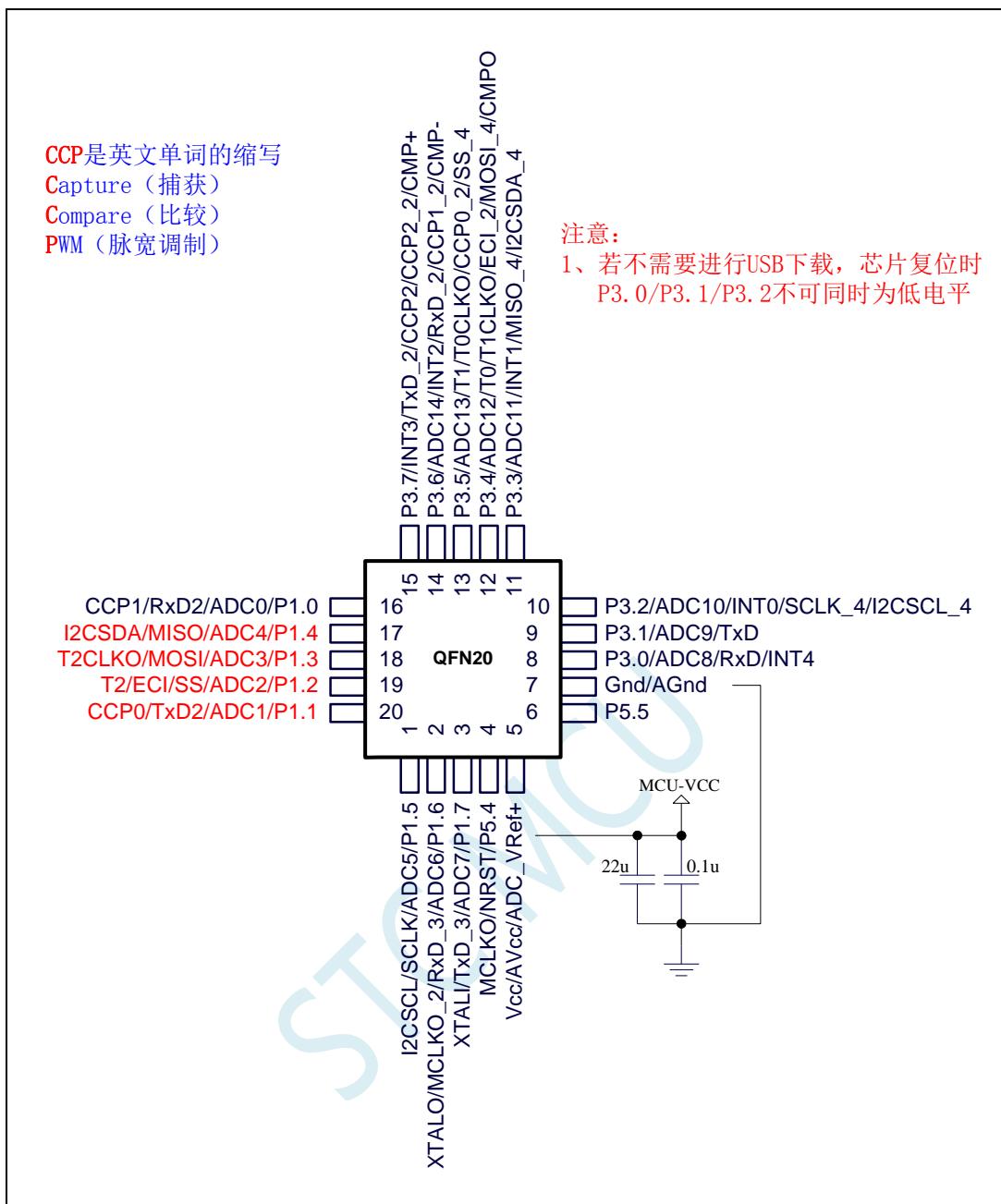
- **封装**

- ✓ TSSOP20、QFN20（3mm*3mm）、SOP20、DIP20、SOP16、DIP16

STCMCU

2.3.2 管脚图, 最小系统





注意:

- 1、除 P3.0 和 P3.1 外，其余所有 I/O 口上电后的状态均为高阻输入状态，用户在使用 I/O 口时必须先设置 I/O 口模式
- 2、所有的 I/O 口均可以设置为准双向口模式、强推挽输出模式、开漏输出模式或者高阻输入模式，另外每个 I/O 均可独立使能内部 4K 上拉电阻
- 3、当使能 P5.4 口为复位脚时，复位电平为低电平
- 4、对于 STC8G1K08 系列 B 版芯片，P5.4 作 I/O 口使用时，电流不要超过 50mA，也不要有力的冲击
- 5、STC8G1K08 系列芯片所支持的 USB 下载为 I/O 口软件模拟的 USB 通信，不可避免的受各种软硬件因素的影响，尤其是电脑端的不同软件和硬件版本的影响，导致有一定比例的芯片无法进行 USB 下载（实测约有 0.2% 无法 USB 下载）。建议批量生产还是使用普通的串口下载或 USB 转串口下载。

2.3.3 管脚说明

编号				名称	类型	说明
TSSOP20 DIP20	QFN20	SOP16 DIP16				
1	19			P1.2	I/O	标准 I/O 口
				ADC2	I	ADC 模拟输入通道 2
				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
2	18			P1.3	I/O	标准 I/O 口
				ADC3	I	ADC 模拟输入通道 3
				T2CLKO	O	定时器 2 时钟分频输出
				MOSI	I/O	SPI 主机输出从机输入
3	17			P1.4	I/O	标准 I/O 口
				ADC4	I	ADC 模拟输入通道 4
				MISO	I/O	SPI 主机输入从机输出
				SDA	I/O	I2C 接口的数据线
4	1			P1.5	I/O	标准 I/O 口
				ADC5	I	ADC 模拟输入通道 5
				SCLK	I/O	SPI 的时钟脚
				SCL	I/O	I2C 的时钟线
5	2	3		P1.6	I/O	标准 I/O 口
				ADC6	I	ADC 模拟输入通道 6
				RxD_3	I	串口 1 的接收脚
				MCLKO_2	O	主时钟分频输出
				XTALO	O	外部晶振脚
6	3	4		P1.7	I/O	标准 I/O 口
				ADC7	I	ADC 模拟输入通道 7
				TxD_3	O	串口 1 的发送脚
				XTALI	I	外部晶振脚
7	4	5		P5.4	I/O	标准 I/O 口
				NRST	I	复位引脚 (低电平复位)
				MCLKO	O	主时钟分频输出
8	5	6		Vcc	Vcc	电源脚
				AVcc	Vcc	ADC 电源
				ADC_VRef+	I	ADC 外部参考电压源输入脚, 要求不高时可直接接 MCU 的 VCC
9	6	7		P5.5	I/O	标准 I/O 口
10	7	8		Gnd	Gnd	地线
				AGnd	Gnd	ADC 地线

编号				名称	类型	说明
TSSOP20 DIP20	QFN20	SOP16 DIP16				
11	8	9		P3.0	I/O	标准 I/O 口
				RxD	I	串口 1 的接收脚
				ADC8	I	ADC 模拟输入通道 8
				INT4	I	外部中断 4
12	9	10		P3.1	I/O	标准 I/O 口
				TxD	O	串口 1 的发送脚
				ADC9	I	ADC 模拟输入通道 9
13	10	11		P3.2	I/O	标准 I/O 口
				INT0	I	外部中断 0
				ADC10	I	ADC 模拟输入通道 10
				SCL_4	I/O	I2C 的时钟线
				SCLK_4	I/O	SPI 的时钟脚
14	11	12		P3.3	I/O	标准 I/O 口
				INT1	I	外部中断 1
				ADC11	I	ADC 模拟输入通道 11
				SDA_4	I/O	I2C 的数据线
				MISO_4	I/O	SPI 主机输入从机输出
15	12	13		P3.4	I/O	标准 I/O 口
				T0	I	定时器 0 外部时钟输入
				T1CLKO	O	定时器 1 时钟分频输出
				ADC12	I	ADC 模拟输入通道 12
				ECI_2	I	PCA 的外部脉冲输入
				CMPO	O	比较器输出
16	13	14		MOSI_4	I/O	SPI 主机输出从机输入
				P3.5	I/O	标准 I/O 口
				T1	I	定时器 1 外部时钟输入
				T0CLKO	O	定时器 0 时钟分频输出
				ADC13	I	ADC 模拟输入通道 13
				CCP0_2	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
17	14	15		SS_4	I	SPI 的从机选择脚 (主机为输出)
				P3.6	I/O	标准 I/O 口
				INT2	I	外部中断 2
				RxD_2	I	串口 1 的接收脚
				ADC14	I	ADC 模拟输入通道 14
				CCP1_2	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
				CMP-	I	比较器负极输入

编号				名称	类型	说明
TSSOP20 DIP20	QFN20	SOP16 DIP16				
18	15	16		P3.7	I/O	标准 I/O 口
				INT3	I	外部中断 3
				TxD_2	O	串口 1 的发送脚
				CCP2	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
				CCP2_2	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
				CMP+	I	比较器正极输入
19	16	1		P1.0	I/O	标准 I/O 口
				RxD2	I	串口 2 的接收脚
				ADC0	I	ADC 模拟输入通道 0
				CCP1	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
20	20	2		P1.1	I/O	标准 I/O 口
				TxD2	O	串口 2 的发送脚
				ADC1	I	ADC 模拟输入通道 1
				CCP0	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出

2.4 STC8G2K64S4-36I-LQFP48/32/QFN4832 系列（45 路增强型 PWM）

2.4.1 特性及价格(有 16 位硬件乘除法器 MDU16, 准 16 位单片机)

- 选型价格（不需要外部晶振、不需要外部复位，10 位 ADC，15 通道）

单片机型号	主力产品供货信息												现货				
	价格及封装						QFN32<4mm*4mm>										
	LQFP32			QFN48<6mm*6mm>			LQFP48										
	本身即可在线仿真						可设置下次更新程序需口令						支持软件USB直接下载				
	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（36MHz以下可调）追频						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（可判1路A/D，可作外部掉电检测）						掉电唤醒专用定时器				
	15路高速ADC（全部PWM均可作D/A使用）						PCA/CCP/PWM（可判外部中断并可掉电唤醒）						支持上升沿中断、下降沿中断以及边沿中断				
	15位增强型PWM满足舞台灯光要求						定时器/计数器(T0-T4外部管脚也可掉电唤醒)						16位乘除法器				
	MDU16硬件16位乘除法器						串口并可掉电唤醒						EEPROM 10万次字节				
	强大的双DPIR可增可减						xdata，内部大容量扩展SRAM字节						idata，内部传统8051RAM字节				
	Flash程序存储器10万次字节						工作电压(V)										
STC8G2K32S4	1.9-5.5	32K	256	2K	2	32K	45	4	有	有	有	5	45	3	有		
STC8G2K48S4	1.9-5.5	48K	256	2K	2	16K	45	4	有	有	有	5	45	3	有		
STC8G2K60S4	1.9-5.5	60K	256	2K	2	4K	45	4	有	有	有	5	45	3	有		
STC8G2K64S4	1.9-5.5	64K	256	2K	2	IAP	45	4	有	有	有	5	45	3	有		

注：以上的单价为 10K 及以上订货量的价格，量小则每片需增加 0.1 元人民币。当订货的总额达到或高于 3000 元时，可免运费发货，否则需要由客户承担运费。零售 10 片起售。

如需要选择 48 管脚的芯片，建议选 LOFP48 封装，OFN48 需要提前订货。

- **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
 - ✓ 29 个中断源，4 级中断优先级
 - ✓ 支持在线仿真
 - **工作电压**
 - ✓ 1.9V~5.5V
 - ✓ 内建 LDO
 - **工作温度**
 - ✓ -40°C~85°C (**超温度范围应用请参考电器特性章节说明**)
 - **Flash 存储器**
 - ✓ 最大 64K 字节 FLASH 程序存储器 (ROM)，用于存储用户程序
 - ✓ 支持用户配置 EEPROM 大小，512 字节单页擦除，擦写次数高达 100 万次
 - ✓ 支持在系统编程方式 (ISP) 更新用户应用程序，无需专用烧录器
 - ✓ 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)

➤ 时钟控制

- ✓ 内部高精度 IRC (4MHz~38MHz, ISP 编程时可进行上下调整, 还可以用户软件分频到较低的频率工作, 如 100KHz)
 - ✧ 误差±0.3% (常温下 25°C)
 - ✧ -1.38%~+1.42% 温漂 (全温度范围, -40°C~85°C)
 - ✧ -0.88%~+1.05% 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~38MHz) 和外部时钟

➤ 复位

- ✓ 硬件复位
 - ✧ 上电复位。 (在芯片未使能低压复位功能时有效)
 - ✧ 复位脚复位。出厂时 P5.4 默认为 I/O 口, ISP 下载时可将 P5.4 管脚设置为复位脚 (注意: 当设置 P5.4 管脚为复位脚时, 复位电平为低电平)
 - ✧ 看门狗溢出复位
 - ✧ 低压检测复位, 提供 4 级低压检测电压: 2.0V、2.4V、2.7V、3.0V。
- ✓ 软件复位
 - ✧ 软件方式写复位触发寄存器

➤ 中断

- ✓ 提供 29 个中断源: INT0 (支持上升沿和下降沿中断)、INT1 (支持上升沿和下降沿中断)、INT2 (只支持下降沿中断)、INT3 (只支持下降沿中断)、INT4 (只支持下降沿中断)、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 PWM0、增强型 PWM1、增强型 PWM2、增强型 PWM3、增强型 PWM4、增强型 PWM5、增强型 PWM0 异常检测、增强型 PWM2 异常检测、增强型 PWM4 异常检测。
- ✓ 提供 4 级中断优先级
- ✓ 时钟停振模式下可以唤醒的中断: 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)、RXD(P3.0/P3.6/P1.6/P4.3)、RXD2(P1.0/P4.6)、RXD3(P0.0/P5.0)、RXD4(P0.2/P5.2)、CCP0(P1.1/P3.5/P2.5)、CCP1(P1.0/P3.6/P2.6)、CCP2(P3.7/P2.7)、I2C_SDA(P1.4/P2.4/P3.3) 以及比较器中断、低压检测中断、掉电唤醒定时器唤醒。

➤ 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
- ✓ 3 组 16 位 CCP/PCA/PWM 模块: CCP0、CCP1、CCP2, 可用于捕获、高速脉冲输出, 及 6/7/8/10 位的 PWM 输出
- ✓ 45 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 3 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
- ✓ **MDU16: 硬件 16 位乘除法器 (支持 32 位除以 16 位、16 位除以 16 位、16 位乘 16 位、数据移位以及数据规格化等运算)**

➤ 模拟外设

- ✓ 超高速 ADC，支持 **10 位精度** 15 通道（通道 0~通道 14）的模数转换，**速度最快能达到 500K (每秒进行 50 万次 ADC 转换)**
- ✓ ADC 的通道 15 用于测试内部 1.19V 参考信号源（芯片在出厂时，内部参考信号源已调整为 1.19V）
- ✓ 比较器，一组比较器（比较器的正端可选择 CMP+端口和所有的 ADC 输入端口，所以比较器可当作多路比较器进行分时复用）
- ✓ DAC: 3 路 PCA/CCP/PWM 可当 3 路 DAC 使用，45 路增强型 PWM 可当 45 路 DAC 使用

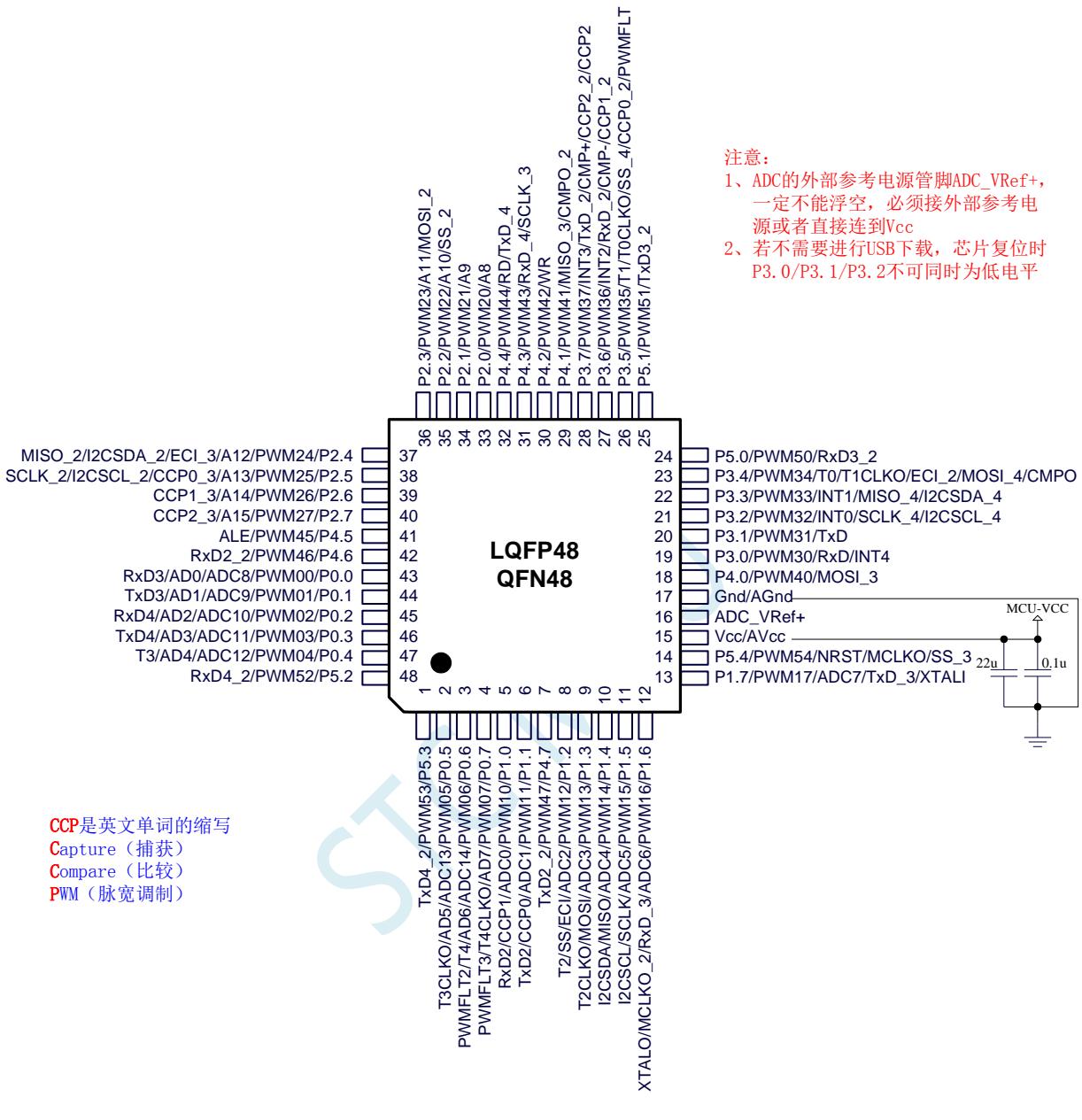
➤ **GPIO**

- ✓ 最多可达 45 个 GPIO: P0.0~P0.7、P1.0~P1.7、P2.0~P2.7、P3.0~P3.7、P4.0~P4.7、P5.0~P5.4
- ✓ 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
- ✓ **除 P3.0 和 P3.1 外，其余所有 I/O 口上电后的状态均为高阻输入状态，用户在使用 I/O 口时必须先设置 I/O 口模式，另外每个 I/O 均可独立使能内部 4K 上拉电阻**

➤ **封装**

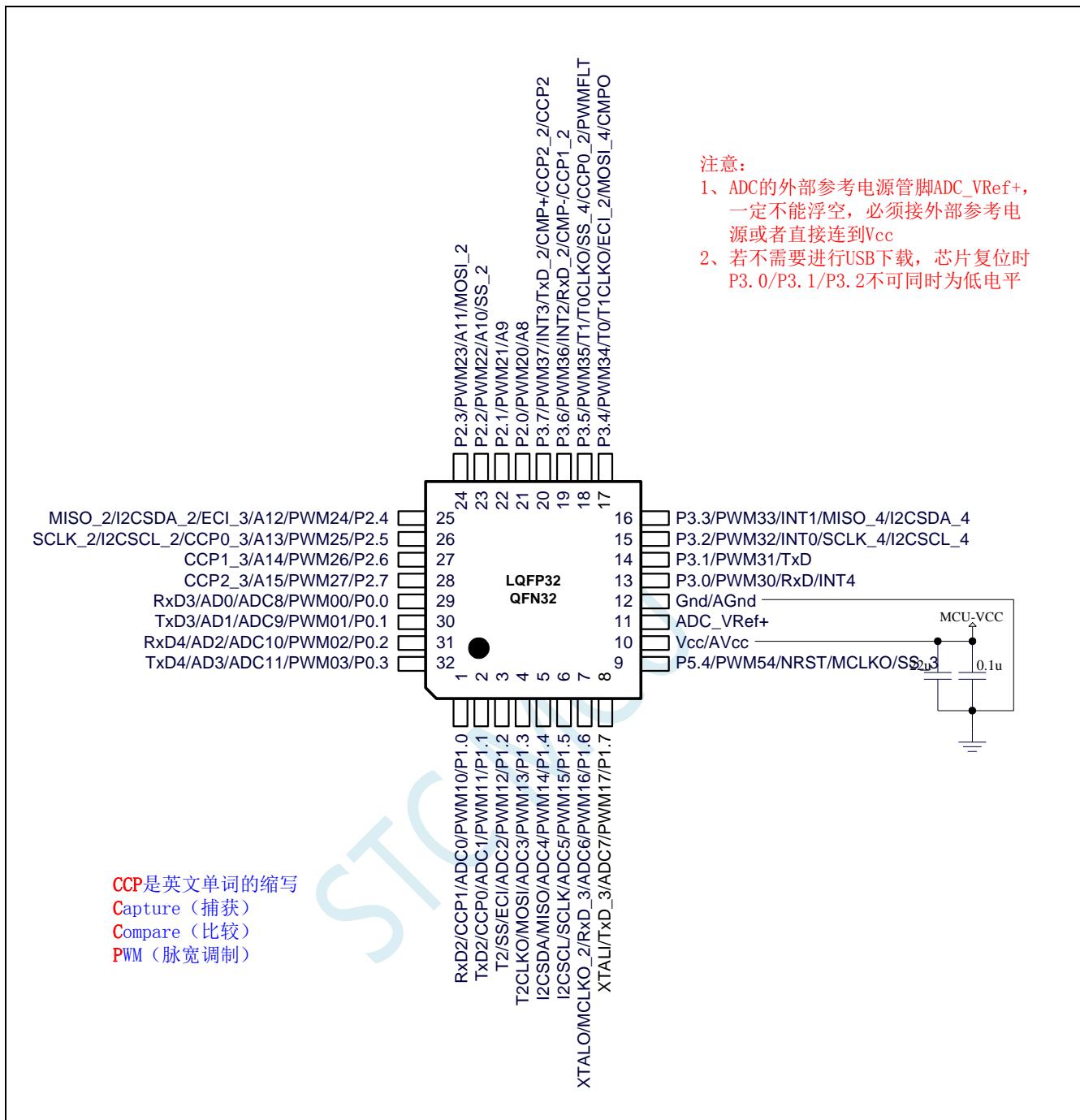
- ✓ LQFP48、QFN48、LQFP32

2.4.2 管脚图, 最小系统



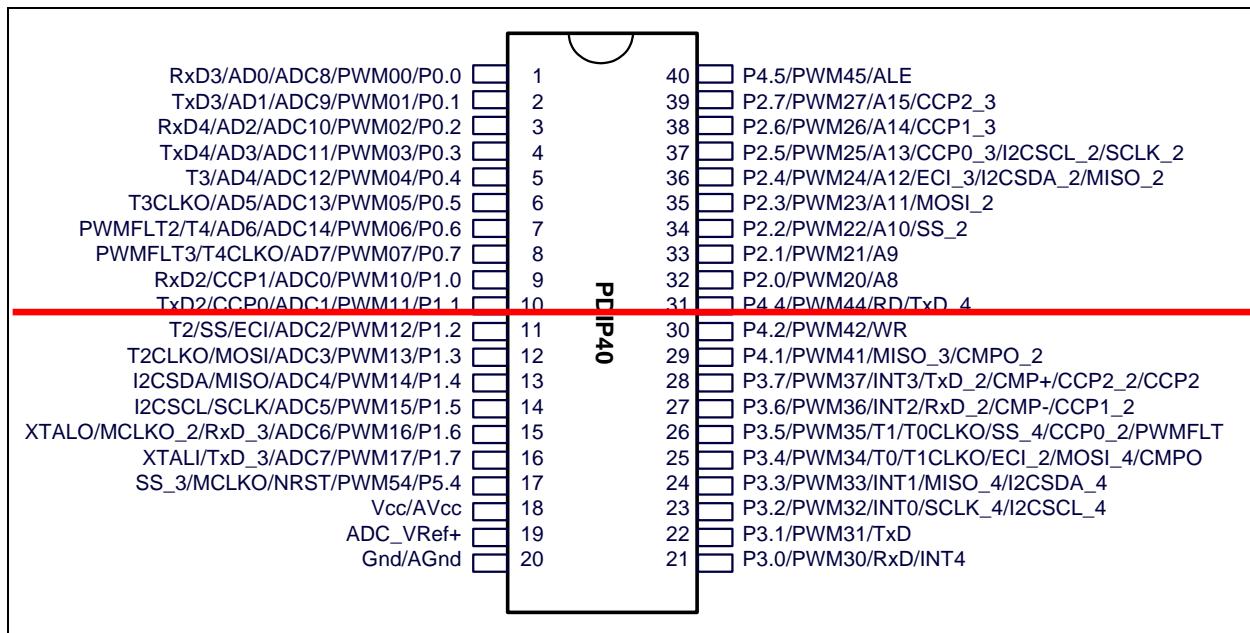
正看芯片丝印左下方小圆点处为第一脚

正看芯片丝印最下面一行最后一个字母为芯片版本号



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



2.4.3 管脚说明

编号		名称	类型	说明
LQFP48				
1		P5.3	I/O	标准 IO 口
		PWM53	O	增强 PWM 输出脚
		TxD4_2	O	串口 4 的发送脚
2		P0.5	I/O	标准 IO 口
		PWM05	O	增强 PWM 输出脚
		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
3		P0.6	I/O	标准 IO 口
		PWM06	O	增强 PWM 输出脚
		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
		PWMFLT2	I	增强 PWM 的外部异常检测脚
4		P0.7	I/O	标准 IO 口
		PWM07	O	增强 PWM 输出脚
		AD7	I	地址总线
		T4CLKO	O	定时器 4 时钟分频输出
		PWMFLT3	I	增强 PWM 的外部异常检测脚
5		P1.0	I/O	标准 IO 口
		PWM10	O	增强 PWM 输出脚
		ADC0	I	ADC 模拟输入通道 0
		CCP1	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
		RxD2	I	串口 2 的接收脚
6		P1.1	I/O	标准 IO 口
		PWM11	O	增强 PWM 输出脚
		ADC1	I	ADC 模拟输入通道 1
		CCP0	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
		TxD2	O	串口 2 的发送脚
7		P4.7	I/O	标准 IO 口
		PWM47	O	增强 PWM 输出脚
		TxD2_2	O	串口 2 的发送脚
8		P1.2	I/O	标准 IO 口
		PWM12	O	增强 PWM 输出脚
		ADC2	I	ADC 模拟输入通道 2
		ECI	I	PCA 的外部脉冲输入
		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

编号		名称	类型	说明
LQFP48				
9		P1.3	I/O	标准 IO 口
		PWM13	O	增强 PWM 输出脚
		ADC3	I	ADC 模拟输入通道 3
		MOSI	I/O	SPI 主机输出从机输入
		T2CLKO	O	定时器 2 时钟分频输出
10		P1.4	I/O	标准 IO 口
		PWM14	O	增强 PWM 输出脚
		ADC4	I	ADC 模拟输入通道 4
		MISO	I/O	SPI 主机输入从机输出
		SDA	I/O	I2C 接口的数据线
11		P1.5	I/O	标准 IO 口
		PWM15	O	增强 PWM 输出脚
		ADC5	I	ADC 模拟输入通道 5
		SCLK	I/O	SPI 的时钟脚
		SCL	I/O	I2C 的时钟线
12		P1.6	I/O	标准 IO 口
		PWM16	O	增强 PWM 输出脚
		ADC6	I	ADC 模拟输入通道 6
		RxD_3	I	串口 1 的接收脚
		MCLKO_2	O	主时钟分频输出
		XTALO	O	外部晶振的输出脚
13		P1.7	I/O	标准 IO 口
		PWM17	O	增强 PWM 输出脚
		ADC7	I	ADC 模拟输入通道 7
		TxD_3	O	串口 1 的发送脚
		XTALI	I	外部晶振/外部时钟的输入脚
14		P5.4	I/O	标准 IO 口
		PWM54	O	增强 PWM 输出脚
		NRST	I	复位引脚（低电平复位）
		MCLKO	O	主时钟分频输出
		SS_3	I	SPI 的从机选择脚（主机为输出）
15		Vcc	Vcc	电源脚
		AVcc	Vcc	ADC 电源脚
16		ADC_VRef+	I	ADC 外部参考电压源输入脚，要求不高时可直接接 MCU 的 VCC
17		Gnd	Gnd	地线
		AGnd	Gnd	ADC 地线

编号		名称	类型	说明
LQFP48				
18		P4.0	I/O	标准 IO 口
		PWM40	O	增强 PWM 输出脚
		MOSI_3	I/O	SPI 主机输出从机输入
19		P3.0	I/O	标准 IO 口
		PWM30	O	增强 PWM 输出脚
		RxD	I	串口 1 的接收脚
		INT4	I	外部中断 4
20		P3.1	I/O	标准 IO 口
		PWM31	O	增强 PWM 输出脚
		TxD	O	串口 1 的发送脚
21		P3.2	I/O	标准 IO 口
		PWM32	O	增强 PWM 输出脚
		INT0	I	外部中断 0
		SCLK_4	I/O	SPI 的时钟脚
		SCL_4	I/O	I2C 的时钟线
22		P3.3	I/O	标准 IO 口
		PWM33	O	增强 PWM 输出脚
		INT1	I	外部中断 1
		MISO_4	I/O	SPI 主机输入从机输出
		SDA_4	I/O	I2C 接口的数据线
23		P3.4	I/O	标准 IO 口
		PWM34	O	增强 PWM 输出脚
		T0	I	定时器 0 外部时钟输入
		T1CLKO	O	定时器 1 时钟分频输出
		ECI_2	I	PCA 的外部脉冲输入
		MOSI_4	I/O	SPI 主机输出从机输入
		CMPO	O	比较器输出
24		P5.0	I/O	标准 IO 口
		PWM50	O	增强 PWM 输出脚
		RxD3_2	I	串口 3 的接收脚
25		P5.1	I/O	标准 IO 口
		PWM52	O	增强 PWM 输出脚
		TxD3_2	O	串口 3 的发送脚

编号		名称	类型	说明
LQFP48				
26		P3.5	I/O	标准 IO 口
		PWM35	O	增强 PWM 输出脚
		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)
		CCP0_2	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
		PWMFLT	I	增强 PWM 的外部异常检测脚
27		P3.6	I/O	标准 IO 口
		PWM36	O	增强 PWM 输出脚
		INT2	I	外部中断 2
		RxD_2	I	串口 1 的接收脚
		CMP-	I	比较器负极输入
		CCP1_2	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
28		P3.7	I/O	标准 IO 口
		PWM37	O	增强 PWM 输出脚
		INT3	I	外部中断 3
		TxD_2	O	串口 1 的发送脚
		CMP+	I	比较器正极输入
		CCP2	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
		CCP2_2	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
29		P4.1	I/O	标准 IO 口
		PWM41	O	增强 PWM 输出脚
		MISO_3	I/O	SPI 主机输入从机输出
		CMPO_2	O	比较器输出
30		P4.2	I/O	标准 IO 口
		PWM42	O	增强 PWM 输出脚
		WR	O	外部总线的写信号线
31		P4.3	I/O	标准 IO 口
		PWM43	O	增强 PWM 输出脚
		RxD_4	I	串口 1 的接收脚
		SCLK_3	I/O	SPI 的时钟脚
32		P4.4	I/O	标准 IO 口
		PWM44	O	增强 PWM 输出脚
		RD	O	外部总线的读信号线
		TxD_4	O	串口 1 的发送脚
33		P2.0	I/O	标准 IO 口
		PWM20	O	增强 PWM 输出脚
		A8	I	地址总线

编号		名称	类型	说明
LQFP48				
34		P2.1	I/O	标准 IO 口
		PWM21	O	增强 PWM 输出脚
		A9	I	地址总线
35		P2.2	I/O	标准 IO 口
		PWM22	O	增强 PWM 输出脚
		A10	I	地址总线
		SS_2	I	SPI 的从机选择脚 (主机为输出)
36		P2.3	I/O	标准 IO 口
		PWM23	O	增强 PWM 输出脚
		A11	I	地址总线
		MOSI_2	I/O	SPI 主机输出从机输入
		CCP0_2	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
37		P2.4	I/O	标准 IO 口
		PWM24	O	增强 PWM 输出脚
		A12	I	地址总线
		ECI_3	I	PCA 的外部脉冲输入
		SDA_2	I/O	I2C 接口的数据线
		MISO_2	I/O	SPI 主机输入从机输出
38		P2.5	I/O	标准 IO 口
		PWM25	O	增强 PWM 输出脚
		A13	I	地址总线
		CCP0_3	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
		SCL_2	I/O	I2C 的时钟线
		SCLK_2	I/O	SPI 的时钟脚
39		P2.6	I/O	标准 IO 口
		PWM26	O	增强 PWM 输出脚
		A14	I	地址总线
		CCP1_3	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
40		P2.7	I/O	标准 IO 口
		PWM27	O	增强 PWM 输出脚
		A15	I	地址总线
		CCP2_3	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
41		P4.5	I/O	标准 IO 口
		PWM45	O	增强 PWM 输出脚
		ALE	O	地址锁存信号

编号		名称	类型	说明
LQFP48				
42		P4.6	I/O	标准 IO 口
		PWM46	O	增强 PWM 输出脚
		RxD2_2	I	串口 2 的接收脚
43		P0.0	I/O	标准 IO 口
		PWM00	O	增强 PWM 输出脚
		ADC8	I	ADC 模拟输入通道 8
		AD0	I	地址总线
		RxD3	I	串口 3 的接收脚
44		P0.1	I/O	标准 IO 口
		PWM01	O	增强 PWM 输出脚
		ADC9	I	ADC 模拟输入通道 9
		AD1	I	地址总线
		TxD3	O	串口 3 的发送脚
45		P0.2	I/O	标准 IO 口
		PWM02	O	增强 PWM 输出脚
		ADC10	I	ADC 模拟输入通道 10
		AD2	I	地址总线
		RxD4	I	串口 4 的接收脚
46		P0.3	I/O	标准 IO 口
		PWM03	O	增强 PWM 输出脚
		ADC11	I	ADC 模拟输入通道 11
		AD3	I	地址总线
		TxD4	O	串口 4 的发送脚
47		P0.4	I/O	标准 IO 口
		PWM04	O	增强 PWM 输出脚
		ADC12	I	ADC 模拟输入通道 12
		AD4	I	地址总线
		T3	I	定时器 3 外部时钟输入
48		P5.2	I/O	标准 IO 口
		PWM52	O	增强 PWM 输出脚
		RxD4_2	I	串口 4 的接收脚

2.5 STC8G2K64S2-36I-LQFP48/QFN48 系列（8 路增强型 PWM）

注意: STC8G2K64S2 系列只有 P2 口才有增强型 PWM, 其他口没有

2.5.1 特性及价格(有 16 位硬件乘除法器 MDU16, 准 16 位单片机)

➤ 选型价格 (不需要外部晶振、不需要外部复位, 10 位 ADC, 15 通道)

单片机型号	主力产品供货信息														
	价格及封装					LQFP32					QFN48				
STC8G2K32S2	1.9~5.5	32K	256	2K	2	32K	45	2	有	有	5	-	8	3	有
STC8G2K48S2	1.9~5.5	48K	256	2K	2	16K	45	2	有	有	5	-	8	3	有
STC8G2K64S2	1.9~5.5	64K	256	2K	2	IAP	45	2	有	有	5	45	3	有	10位

注: 以上的单价为 10K 及以上订货量的价格, 量小则每片需增加 0.1 元人民币。当订货的总额达到或高于 3000 元时, 可免运费发货, 否则需要由客户承担运费。零售 10 片起售。

➤ 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
- ✓ 27 个中断源, 4 级中断优先级
- ✓ 支持在线仿真

➤ 工作电压

- ✓ 1.9V~5.5V
- ✓ 内建 LDO

➤ 工作温度

- ✓ -40°C~85°C (超温度范围应用请参考电器特性章节说明)

➤ Flash 存储器

- ✓ 最大 64K 字节 FLASH 程序存储器 (ROM), 用于存储用户代码
- ✓ 支持用户配置 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)
- ✓ 2048 字节内部扩展 RAM (内部 XDATA)

➤ 时钟控制

- ✓ 内部高精度 IRC (4MHz~38MHz, ISP 编程时可进行上下调整, 还可以用户软件分频到较低的频率工作, 如 100KHz)
 - ✧ 误差±0.3% (常温下 25°C)
 - ✧ -1.38% ~ +1.42% 温漂 (全温度范围, -40°C ~ 85°C)
 - ✧ -0.88% ~ +1.05% 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~38MHz) 和外部时钟

➤ 复位

- ✓ 硬件复位
 - ✧ 上电复位。 (在芯片未使能低压复位功能时有效)
 - ✧ 复位脚复位。出厂时 P5.4 默认为 I/O 口, ISP 下载时可将 P5.4 管脚设置为复位脚 (注意: 当设置 P5.4 管脚为复位脚时, 复位电平为低电平)
 - ✧ 看门狗溢出复位
 - ✧ 低压检测复位, 提供 4 级低压检测电压: 2.0V、2.4V、2.7V、3.0V。
- ✓ 软件复位
 - ✧ 软件方式写复位触发寄存器

➤ 中断

- ✓ 提供 27 个中断源: INT0 (支持上升沿和下降沿中断)、INT1 (支持上升沿和下降沿中断)、INT2 (只支持下降沿中断)、INT3 (只支持下降沿中断)、INT4 (只支持下降沿中断)、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 PWM0**、**增强型 PWM1**、增强型 PWM2、**增强型 PWM3**、**增强型 PWM4**、**增强型 PWM5**、**增强型 PWM0 异常检测**、增强型 PWM2 异常检测、**增强型 PWM4 异常检测**。
- ✓ 提供 4 级中断优先级
- ✓ 时钟停振模式下可以唤醒的中断: 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)、RXD(P3.0/P3.6/P1.6/P4.3)、RXD2(P1.0/P4.6)、CCP0(P1.1/P3.5/P2.5)、CCP1(P1.0/P3.6/P2.6)、CCP2(P3.7/P2.7)、I2C_SDA(P1.4/P2.4/P3.3) 以及比较器中断、低压检测中断、掉电唤醒定时器唤醒。

➤ 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
- ✓ 3 组 16 位 CCP/PCA/PWM 模块: CCP0、CCP1、CCP2, 可用于捕获、高速脉冲输出, 及 6/7/8/10 位的 PWM 输出
- ✓ 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 3 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
- ✓ **MDU16: 硬件 16 位乘除法器 (支持 32 位除以 16 位、16 位除以 16 位、16 位乘 16 位、数据移位以及数据规格化等运算)**

➤ 模拟外设

- ✓ 超高速 ADC, 支持 **10 位精度** 15 通道 (通道 0~通道 14) 的模数转换, **速度最快能达到 500K (每秒进行 50 万次 ADC 转换)**
- ✓ ADC 的通道 15 用于测试内部 1.19V 参考信号源 (芯片在出厂时, 内部参考信号源已调整为 1.19V)

- ✓ 比较器，一组比较器（比较器的正端可选择 CMP+端口和所有的 ADC 输入端口，所以比较器可当作多路比较器进行分时复用）
- ✓ DAC: 3 路 PCA/CCP/PWM 可当 3 路 DAC 使用，45 路增强型 PWM 可当 45 路 DAC 使用

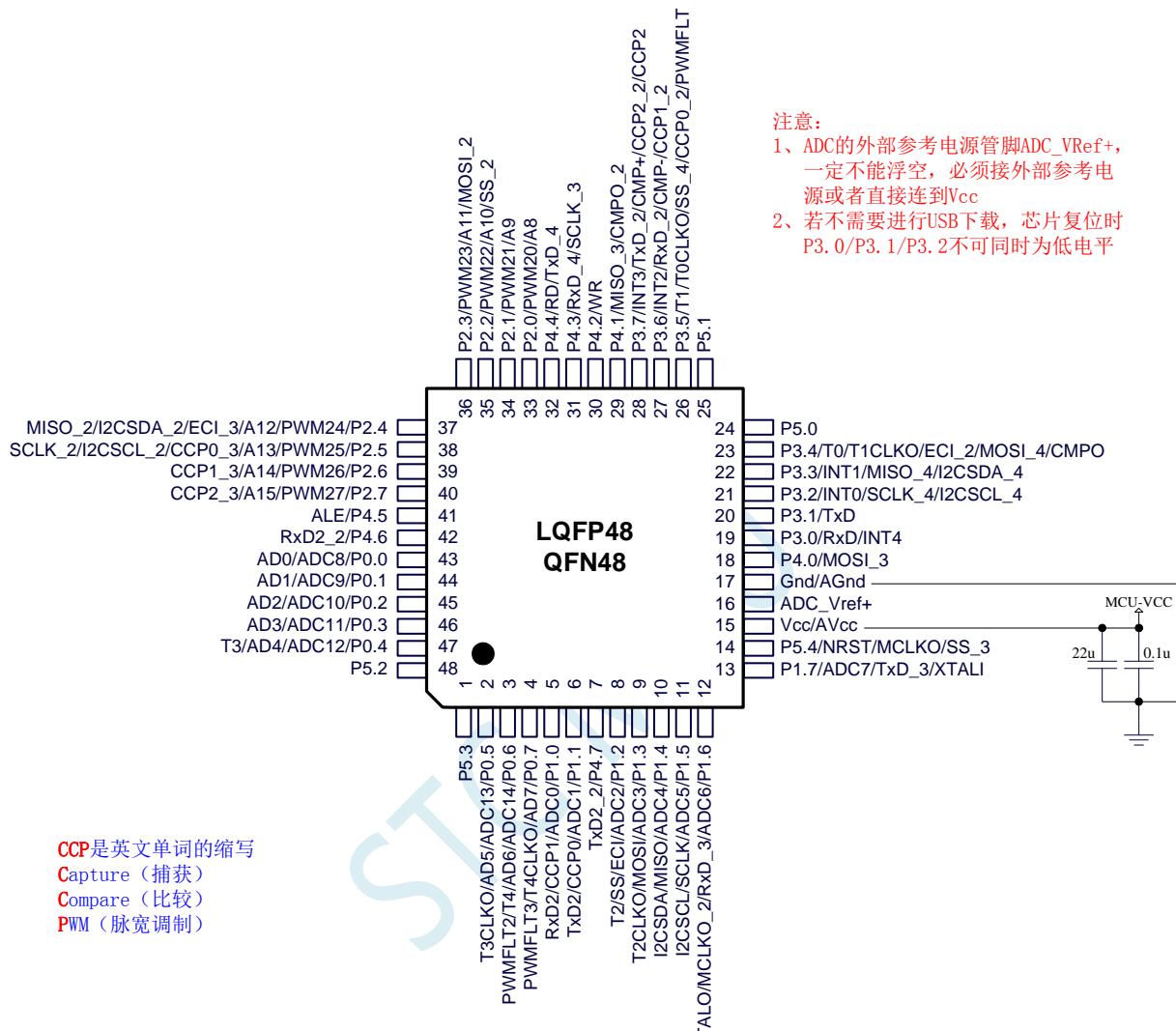
➤ **GPIO**

- ✓ 最多可达 45 个 GPIO: P0.0~P0.7、P1.0~P1.7、P2.0~P2.7、P3.0~P3.7、P4.0~P4.7、P5.0~P5.4
- ✓ 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
- ✓ **除 P3.0 和 P3.1 外，其余所有 I/O 口上电后的状态均为高阻输入状态，用户在使用 I/O 口时必须先设置 I/O 口模式，另外每个 I/O 均可独立使能内部 4K 上拉电阻**

➤ **封装**

- ✓ LQFP48、QFN48

2.5.2 管脚图, 最小系统



CCP是英文单词的缩写

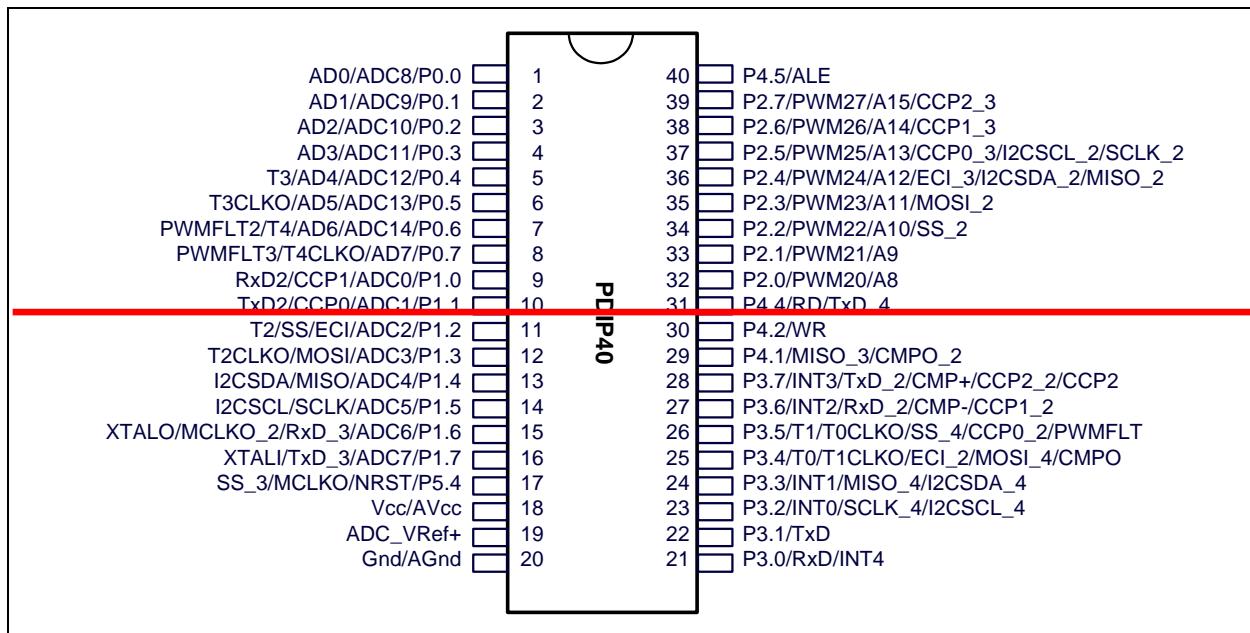
Capture (捕获)

Compare (比较)

PWM (脉宽调制)

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



2.5.3 管脚说明

编号		名称	类型	说明
LQFP48				
1		P5.3	I/O	标准 IO 口
		PWM53	Θ	增强 PWM 输出脚
2		P0.5	I/O	标准 IO 口
		PWM05	Θ	增强 PWM 输出脚
		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
3		P0.6	I/O	标准 IO 口
		PWM06	Θ	增强 PWM 输出脚
		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
		PWMFLT2	I	增强 PWM 的外部异常检测脚
4		P0.7	I/O	标准 IO 口
		PWM07	Θ	增强 PWM 输出脚
		AD7	I	地址总线
		T4CLKO	O	定时器 4 时钟分频输出
		PWMFLT3	I	增强 PWM 的外部异常检测脚
5		P1.0	I/O	标准 IO 口
		PWM10	Θ	增强 PWM 输出脚
		ADC0	I	ADC 模拟输入通道 0
		CCP1	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
		RxD2	I	串口 2 的接收脚
6		P1.1	I/O	标准 IO 口
		PWM11	Θ	增强 PWM 输出脚
		ADC1	I	ADC 模拟输入通道 1
		CCP0	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
		TxD2	O	串口 2 的发送脚
7		P4.7	I/O	标准 IO 口
		PWM47	Θ	增强 PWM 输出脚
		TxD2_2	O	串口 2 的发送脚
8		P1.2	I/O	标准 IO 口
		PWM12	Θ	增强 PWM 输出脚
		ADC2	I	ADC 模拟输入通道 2
		ECI	I	PCA 的外部脉冲输入
		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

编号		名称	类型	说明
LQFP48				
9		P1.3	I/O	标准 IO 口
		PWM13	Θ	增强 PWM 输出脚
		ADC3	I	ADC 模拟输入通道 3
		MOSI	I/O	SPI 主机输出从机输入
		T2CLKO	O	定时器 2 时钟分频输出
10		P1.4	I/O	标准 IO 口
		PWM14	Θ	增强 PWM 输出脚
		ADC4	I	ADC 模拟输入通道 4
		MISO	I/O	SPI 主机输入从机输出
		SDA	I/O	I2C 接口的数据线
11		P1.5	I/O	标准 IO 口
		PWM15	Θ	增强 PWM 输出脚
		ADC5	I	ADC 模拟输入通道 5
		SCLK	I/O	SPI 的时钟脚
		SCL	I/O	I2C 的时钟线
12		P1.6	I/O	标准 IO 口
		PWM16	Θ	增强 PWM 输出脚
		ADC6	I	ADC 模拟输入通道 6
		RxD_3	I	串口 1 的接收脚
		MCLKO_2	O	主时钟分频输出
		XTALO	O	外部晶振的输出脚
13		P1.7	I/O	标准 IO 口
		PWM17	Θ	增强 PWM 输出脚
		ADC7	I	ADC 模拟输入通道 7
		TxD_3	O	串口 1 的发送脚
		XTALI	I	外部晶振/外部时钟的输入脚
14		P5.4	I/O	标准 IO 口
		PWM54	Θ	增强 PWM 输出脚
		NRST	I	复位引脚 (低电平复位)
		MCLKO	O	主时钟分频输出
		SS_3	I	SPI 的从机选择脚 (主机为输出)
15		Vcc	Vcc	电源脚
		AVcc	Vcc	ADC 电源脚
16		ADC_Vref+	I	ADC 外部参考电压源输入脚, 要求不高时可直接接 MCU 的 VCC
17		Gnd	Gnd	地线
		AGnd	Gnd	ADC 地线

编号		名称	类型	说明
LQFP48				
18		P4.0	I/O	标准 IO 口
		PWM40	Θ	增强 PWM 输出脚
		MOSI_3	I/O	SPI 主机输出从机输入
19		P3.0	I/O	标准 IO 口
		PWM30	Θ	增强 PWM 输出脚
		RxD	I	串口 1 的接收脚
		INT4	I	外部中断 4
20		P3.1	I/O	标准 IO 口
		PWM34	Θ	增强 PWM 输出脚
		TxD	O	串口 1 的发送脚
21		P3.2	I/O	标准 IO 口
		PWM32	Θ	增强 PWM 输出脚
		INT0	I	外部中断 0
		SCLK_4	I/O	SPI 的时钟脚
		SCL_4	I/O	I2C 的时钟线
22		P3.3	I/O	标准 IO 口
		PWM33	Θ	增强 PWM 输出脚
		INT1	I	外部中断 1
		MISO_4	I/O	SPI 主机输入从机输出
		SDA_4	I/O	I2C 接口的数据线
23		P3.4	I/O	标准 IO 口
		PWM34	Θ	增强 PWM 输出脚
		T0	I	定时器 0 外部时钟输入
		T1CLKO	O	定时器 1 时钟分频输出
		ECI_2	I	PCA 的外部脉冲输入
		MOSI_4	I/O	SPI 主机输出从机输入
		CMPO	O	比较器输出
24		P5.0	I/O	标准 IO 口
		PWM50	Θ	增强 PWM 输出脚
		RxD3_2	I	串口 3 的接收脚
25		P5.1	I/O	标准 IO 口
		PWM52	Θ	增强 PWM 输出脚

编号		名称	类型	说明
LQFP48				
26		P3.5	I/O	标准 IO 口
		PWM35	Θ	增强 PWM 输出脚
		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)
		CCP0_2	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
		PWMFLT	I	增强 PWM 的外部异常检测脚
27		P3.6	I/O	标准 IO 口
		PWM36	Θ	增强 PWM 输出脚
		INT2	I	外部中断 2
		RxD_2	I	串口 1 的接收脚
		CMP-	I	比较器负极输入
		CCP1_2	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
28		P3.7	I/O	标准 IO 口
		PWM37	Θ	增强 PWM 输出脚
		INT3	I	外部中断 3
		TxD_2	O	串口 1 的发送脚
		CMP+	I	比较器正极输入
		CCP2	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
		CCP2_2	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
29		P4.1	I/O	标准 IO 口
		PWM41	Θ	增强 PWM 输出脚
		MISO_3	I/O	SPI 主机输入从机输出
		CMPO_2	O	比较器输出
30		P4.2	I/O	标准 IO 口
		PWM42	Θ	增强 PWM 输出脚
		WR	O	外部总线的写信号线
31		P4.3	I/O	标准 IO 口
		PWM43	Θ	增强 PWM 输出脚
		RxD_4	I	串口 1 的接收脚
		SCLK_3	I/O	SPI 的时钟脚
32		P4.4	I/O	标准 IO 口
		PWM44	Θ	增强 PWM 输出脚
		RD	O	外部总线的读信号线
		TxD_4	O	串口 1 的发送脚
33		P2.0	I/O	标准 IO 口
		PWM20	O	增强 PWM 输出脚
		A8	I	地址总线

编号		名称	类型	说明
LQFP48				
34		P2.1	I/O	标准 IO 口
		PWM21	O	增强 PWM 输出脚
		A9	I	地址总线
35		P2.2	I/O	标准 IO 口
		PWM22	O	增强 PWM 输出脚
		A10	I	地址总线
		SS_2	I	SPI 的从机选择脚 (主机为输出)
36		P2.3	I/O	标准 IO 口
		PWM23	O	增强 PWM 输出脚
		A11	I	地址总线
		MOSI_2	I/O	SPI 主机输出从机输入
		CCP0_2	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
37		P2.4	I/O	标准 IO 口
		PWM24	O	增强 PWM 输出脚
		A12	I	地址总线
		ECI_3	I	PCA 的外部脉冲输入
		SDA_2	I/O	I2C 接口的数据线
		MISO_2	I/O	SPI 主机输入从机输出
38		P2.5	I/O	标准 IO 口
		PWM25	O	增强 PWM 输出脚
		A13	I	地址总线
		CCP0_3	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
		SCL_2	I/O	I2C 的时钟线
		SCLK_2	I/O	SPI 的时钟脚
39		P2.6	I/O	标准 IO 口
		PWM26	O	增强 PWM 输出脚
		A14	I	地址总线
		CCP1_3	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
40		P2.7	I/O	标准 IO 口
		PWM27	O	增强 PWM 输出脚
		A15	I	地址总线
		CCP2_3	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
41		P4.5	I/O	标准 IO 口
		PWM45	O	增强 PWM 输出脚
		ALE	O	地址锁存信号

编号		名称	类型	说明
LQFP48				
42		P4.6	I/O	标准 IO 口
		PWM46	Θ	增强 PWM 输出脚
		RxD2_2	I	串口 2 的接收脚
43		P0.0	I/O	标准 IO 口
		PWM00	Θ	增强 PWM 输出脚
		ADC8	I	ADC 模拟输入通道 8
		AD0	I	地址总线
44		P0.1	I/O	标准 IO 口
		PWM01	Θ	增强 PWM 输出脚
		ADC9	I	ADC 模拟输入通道 9
		AD1	I	地址总线
45		P0.2	I/O	标准 IO 口
		PWM02	Θ	增强 PWM 输出脚
		ADC10	I	ADC 模拟输入通道 10
		AD2	I	地址总线
46		P0.3	I/O	标准 IO 口
		PWM03	Θ	增强 PWM 输出脚
		ADC11	I	ADC 模拟输入通道 11
		AD3	I	地址总线
47		P0.4	I/O	标准 IO 口
		PWM04	Θ	增强 PWM 输出脚
		ADC12	I	ADC 模拟输入通道 12
		AD4	I	地址总线
		T3	I	定时器 3 外部时钟输入
48		P5.2	I/O	标准 IO 口
		PWM52	Θ	增强 PWM 输出脚

2.6 STC8G1K08T-36I-TSSOP20（触摸按键/LED 系列）（用 STC8H4K64LR 取代）

2.6.1 特性及价格

➤ 选型价格（不需要外部晶振、不需要外部复位，10 位 ADC，15 通道）

单片机型号	2020 年新品供货信息											
	价格及封装				SOP16				QFN20 (3mm*3mm)			
本身即可在线仿真				TSSOP20								
STC8G1K08T	1.9-5.5	8K	256	1K	2	4K	16	1	有	有	有	有
STC8G1K17T	1.9-5.5	17K	256	1K	2	IAP	16	1	有	有	有	有

注：以上的单价为 10K 及以上订货量的价格，量小则每片需增加 0.1 元人民币。当订货的总额达到或高于 3000 元时，可免运费发货，否则需要由客户承担运费。零售 10 片起售。

➤ 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

- ✓ 1.9V~5.5V
- ✓ 内建 LDO

➤ 工作温度

- ✓ -40°C~85°C (超温度范围应用请参考电器特性章节说明)

➤ Flash 存储器

- ✓ 最大 17K 字节 FLASH 程序存储器 (ROM)，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)
- ✓ 1024 字节内部扩展 RAM (内部 XDATA)

➤ 时钟控制

- ✓ 内部高精度 IRC (4MHz~38MHz, ISP 编程时可进行上下调整, 还可以用户软件分频到较低的频率工作, 如 100KHz)
 - ◆ 误差±0.3% (常温下 25°C)
 - ◆ -1.38%~+1.42% 温漂 (全温度范围, -40°C~85°C)
 - ◆ -0.88%~+1.05% 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~38MHz) 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。 (在芯片未使能低压复位功能时有效)
 - ◆ 复位脚复位。出厂时 P5.4 默认为 I/O 口, ISP 下载时可将 P5.4 管脚设置为复位脚 (注意: 当设置 P5.4 管脚为复位脚时, 复位电平为低电平)
 - ◆ 看门狗溢出复位
 - ◆ 低压检测复位, 提供 4 级低压检测电压: 2.0V、2.4V、2.7V、3.0V。
- ✓ 软件复位
 - ◆ 软件方式写复位触发寄存器

➤ 中断

- ✓ 提供 16 个中断源: INT0 (支持上升沿和下降沿中断)、INT1 (支持上升沿和下降沿中断)、INT2 (只支持下降沿中断)、INT3 (只支持下降沿中断)、INT4 (只支持下降沿中断)、定时器 0、定时器 1、定时器 2、串口 1、ADC 模数转换、LVD 低压检测、SPI、I²C、比较器、PCA/CCP/PWM、触摸按键
- ✓ 提供 4 级中断优先级
- ✓ 时钟停振模式下可以唤醒的中断: INT0(P3.2)、INT1(P3.3)、INT2(P3.6)、INT3(P3.7)、INT4(P3.0)、T0(P3.4)、T1(P3.5)、T2(P1.2)、RXD(P3.0/P3.6/P1.6)、RXD2(P1.0)、CCP0(P1.1/P3.5)、CCP1(P1.0/P3.6)、CCP2(P3.7)、I2C_SDA(P1.4/ P3.3)以及比较器中断、低压检测中断、掉电唤醒定时器。

➤ 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
- ✓ 1 个高速串口: 串口 1, 波特率时钟源最快可为 FOSC/4
- ✓ 3 组 16 位 CCP/PCA/PWM 模块: CCP0、CCP1、CCP2, 可用于捕获、高速脉冲输出, 及 6/7/8/10 位的 PWM 输出
- ✓ SPI: 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 **10 位精度** 15 通道 (通道 0~通道 14) 的模数转换, **速度最快能达到 500K (每秒进行 50 万次 ADC 转换)**
- ✓ ADC 的通道 15 用于测试内部 1.19V 参考信号源 (芯片在出厂时, 内部参考信号源已调整为 1.19V)
- ✓ 比较器, 一组比较器 (比较器的正端可选择 CMP+ 端口和所有的 ADC 输入端口, 所以比较器可当作多路比较器进行分时复用)
- ✓ **触摸按键: 最多支持 16 个触摸按键; 每个触摸按键能够独立使能; 内部参考电压 4 级可调; 灵活的充放电时间设置以及内部工作频率设置; 支持低功耗触摸唤醒**
- ✓ **LED 驱动: 最多可驱动 128 (8*8*2) 个 LED; 可支持共阴模式、共阳模式以及共阴/共阳模式; 支持 8 级灰度调节 (亮度调节)**
- ✓ DAC: 3 路 PCA/CCP/PWM 可当 3 路 DAC 使用

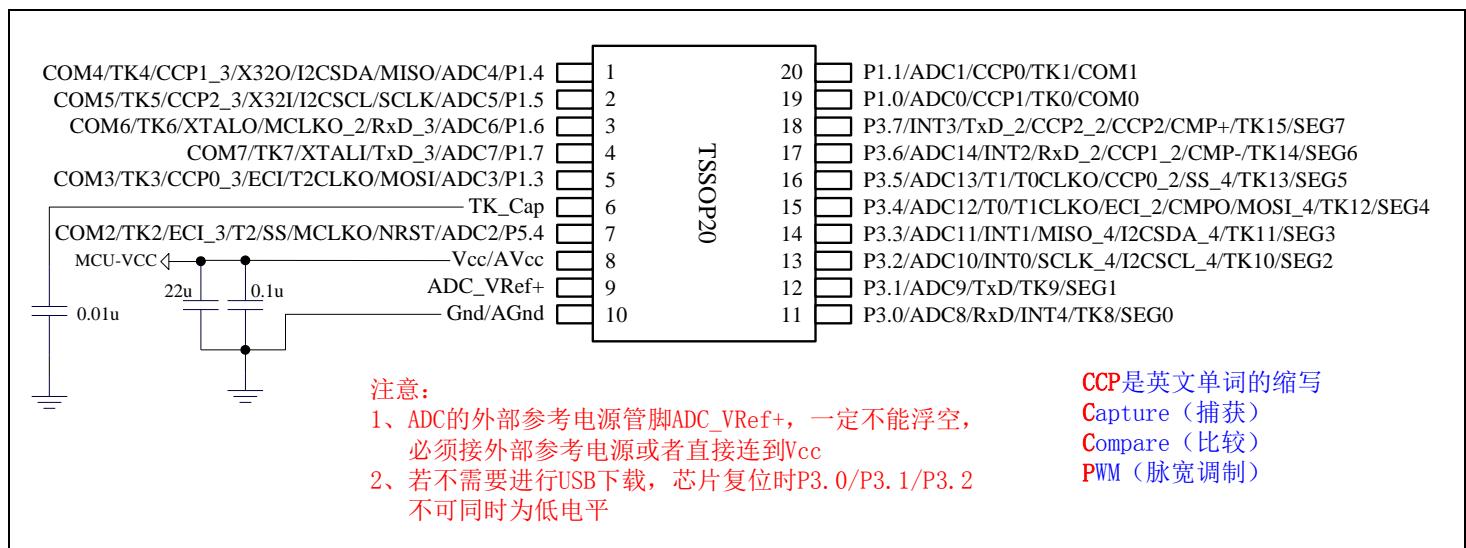
➤ **GPIO**

- ✓ 最多可达 16 个 GPIO: P1.0~P1.1、P1.3~P1.7、P3.0~P3.7、P5.4
- ✓ 所有的 GPIO 均支持如下 4 种模式: 混双向口模式、强推挽输出模式、开漏输出模式、高阻输入模式
- ✓ **除 P3.0 和 P3.1 外, 其余所有 I/O 口上电后的状态均为高阻输入状态, 用户在使用 I/O 口时必须先设置 I/O 口模式, 另外每个 I/O 均可独立使能内部 4K 上拉电阻**

➤ **封装**

- ✓ TSSOP20、QFN20 (3mm*3mm)

2.6.2 管脚图, 最小系统



2.6.3 管脚说明

编号		名称	类型	说明
TSSOP20	QFN20			
1		P1.4	I/O	标准 I/O 口
		ADC4	I	ADC 模拟输入通道 4
		MISO	I/O	SPI 主机输入从机输出
		SDA	I/O	I2C 接口的数据线
		X32O	O	外部晶振脚
		CCP1_3	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
		TK4	I	触摸按键
		COM4	O	LED 驱动
2		P1.5	I/O	标准 I/O 口
		ADC5	I	ADC 模拟输入通道 5
		SCLK	I/O	SPI 的时钟脚
		SCL	I/O	I2C 的时钟线
		X32I	I	外部晶振脚
		CCP2_3	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
		TK5	I	触摸按键
		COM5	O	LED 驱动
3		P1.6	I/O	标准 I/O 口
		ADC6	I	ADC 模拟输入通道 6
		RxD_3	I	串口 1 的接收脚
		MCLKO_2	O	主时钟分频输出
		XTALO	O	外部晶振脚
		TK6	I	触摸按键
		COM6	O	LED 驱动
4		P1.7	I/O	标准 I/O 口
		ADC7	I	ADC 模拟输入通道 7
		TxD_3	O	串口 1 的发送脚
		XTAL1	I	外部晶振脚
		TK7	I	触摸按键
		COM7	O	LED 驱动

编号		名称	类型	说明
TSSOP20	QFN20			
5		P1.3	I/O	标准 I/O 口
		ADC3	I	ADC 模拟输入通道 3
		T2CLKO	O	定时器 2 时钟分频输出
		MOSI	I/O	SPI 主机输出从机输入
		ECI	I	PCA 的外部脉冲输入
		CCP0_3	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
		TK3	I	触摸按键
		COM3	O	LED 驱动
6		TK_Cap	I	触摸按键外接电容
7		P5.4	I/O	标准 I/O 口
		ADC2	I	ADC 模拟输入通道 2
		NRST	I	复位引脚（低电平复位）
		MCLKO	O	主时钟分频输出
		SS	I/O	SPI 从机选择
		T2	I	定时器 2 外部时钟输入
		ECI_3	I	PCA 的外部脉冲输入
		TK2	I	触摸按键
		COM2	O	LED 驱动
8		Vcc	Vcc	电源脚
		AVcc	Vcc	ADC 电源
9		ADC_VRef+	I	ADC 外部参考电压源输入脚，要求不高时可直接接 MCU 的 VCC
10		Gnd	Gnd	地线
		AGnd	Gnd	ADC 地线
11		P3.0	I/O	标准 I/O 口
		RxD	I	串口 1 的接收脚
		ADC8	I	ADC 模拟输入通道 8
		INT4	I	外部中断 4
		TK8	I	触摸按键
		SEG0	O	LED 驱动
12		P3.1	I/O	标准 I/O 口
		TxD	O	串口 1 的发送脚
		ADC9	I	ADC 模拟输入通道 9
		TK9	I	触摸按键
		SEG1	O	LED 驱动

编号		名称	类型	说明
TSSOP20	QFN20			
13		P3.2	I/O	标准 I/O 口
		INT0	I	外部中断 0
		ADC10	I	ADC 模拟输入通道 10
		SCLK_4	I/O	SPI 的时钟脚
		SCL_4	I/O	I2C 的时钟线
		TK10	I	触摸按键
		SEG2	O	LED 驱动
14		P3.3	I/O	标准 I/O 口
		INT1	I	外部中断 1
		ADC11	I	ADC 模拟输入通道 11
		MISO_4	I/O	SPI 主机输入从机输出
		SDA_4	I/O	I2C 的数据线
		TK11	I	触摸按键
		SEG3	O	LED 驱动
15		P3.4	I/O	标准 I/O 口
		T0	I	定时器 0 外部时钟输入
		T1CLKO	O	定时器 1 时钟分频输出
		ADC12	I	ADC 模拟输入通道 12
		ECI_2	I	PCA 的外部脉冲输入
		CMPO	O	比较器输出
		MOSI_4	I/O	SPI 主机输出从机输入
		TK12	I	触摸按键
		SEG4	O	LED 驱动
16		P3.5	I/O	标准 I/O 口
		T1	I	定时器 1 外部时钟输入
		T0CLKO	O	定时器 0 时钟分频输出
		ADC13	I	ADC 模拟输入通道 13
		CCP0_2	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
		SS_4	I	SPI 的从机选择脚（主机为输出）
		TK13	I	触摸按键
		SEG5	O	LED 驱动
17		P3.6	I/O	标准 I/O 口
		INT2	I	外部中断 2
		RxD_2	I	串口 1 的接收脚
		ADC14	I	ADC 模拟输入通道 14
		CCP1_2	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
		CMP-	I	比较器负极输入
		TK14	I	触摸按键
		SEG6	O	LED 驱动

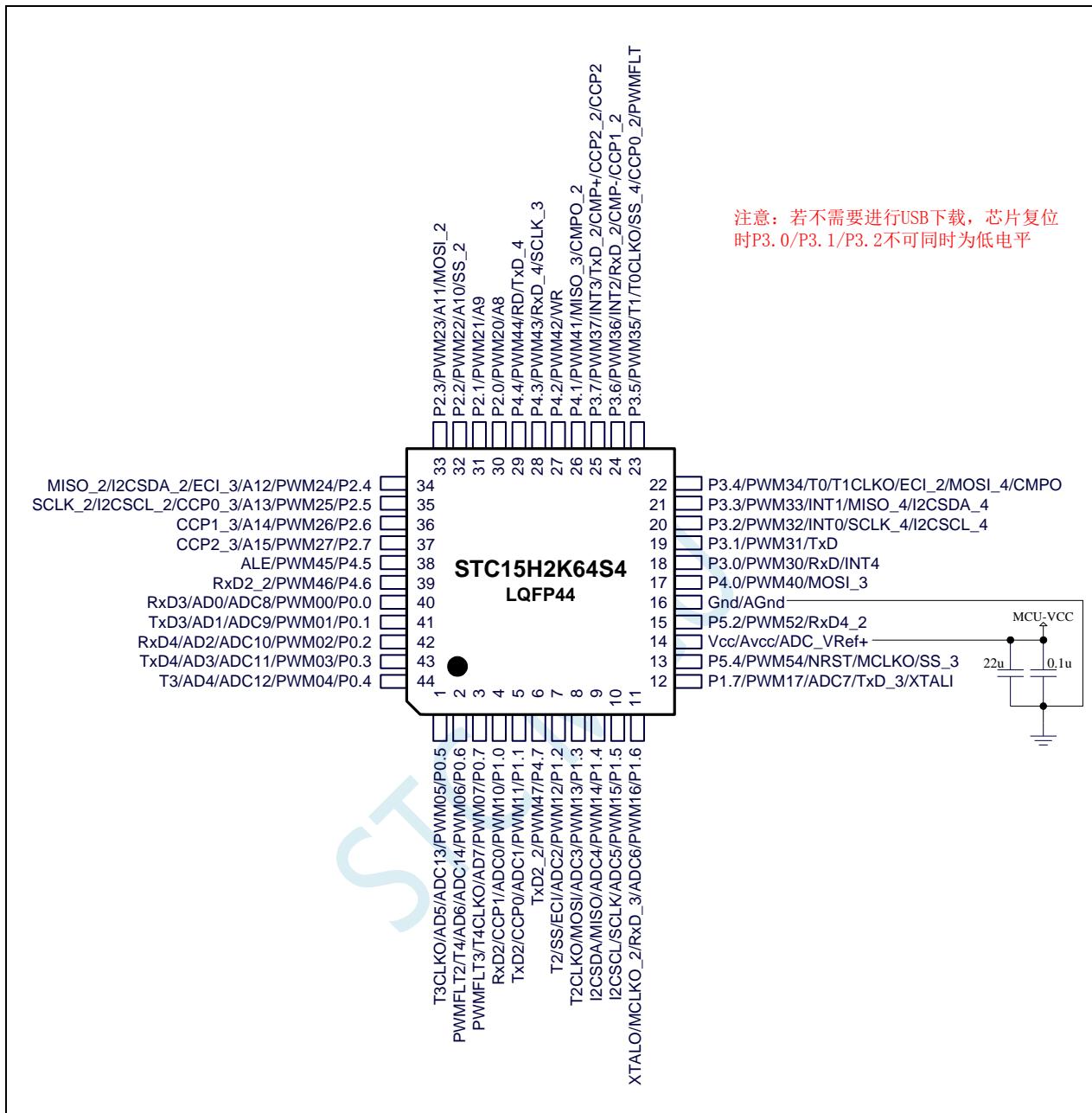
编号		名称	类型	说明
TSSOP20	QFN20			
18		P3.7	I/O	标准 I/O 口
		INT3	I	外部中断 3
		TxD_2	O	串口 1 的发送脚
		CCP2	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
		CCP2_2	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
		CMP+	I	比较器正极输入
		TK15	I	触摸按键
		SEG7	O	LED 驱动
19		P1.0	I/O	标准 I/O 口
		ADC0	I	ADC 模拟输入通道 0
		CCP1	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
		TK0	I	触摸按键
		COM0	O	LED 驱动
20		P1.1	I/O	标准 I/O 口
		ADC1	I	ADC 模拟输入通道 1
		CCP0	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
		TK1	I	触摸按键
		COM1	O	LED 驱动

2.7 STC15H 系列（传统 STC15 系列提升性能特殊型号）

2.7.1 特性及价格(有 16 位硬件乘除法器 MDU16, 准 16 位单片机)

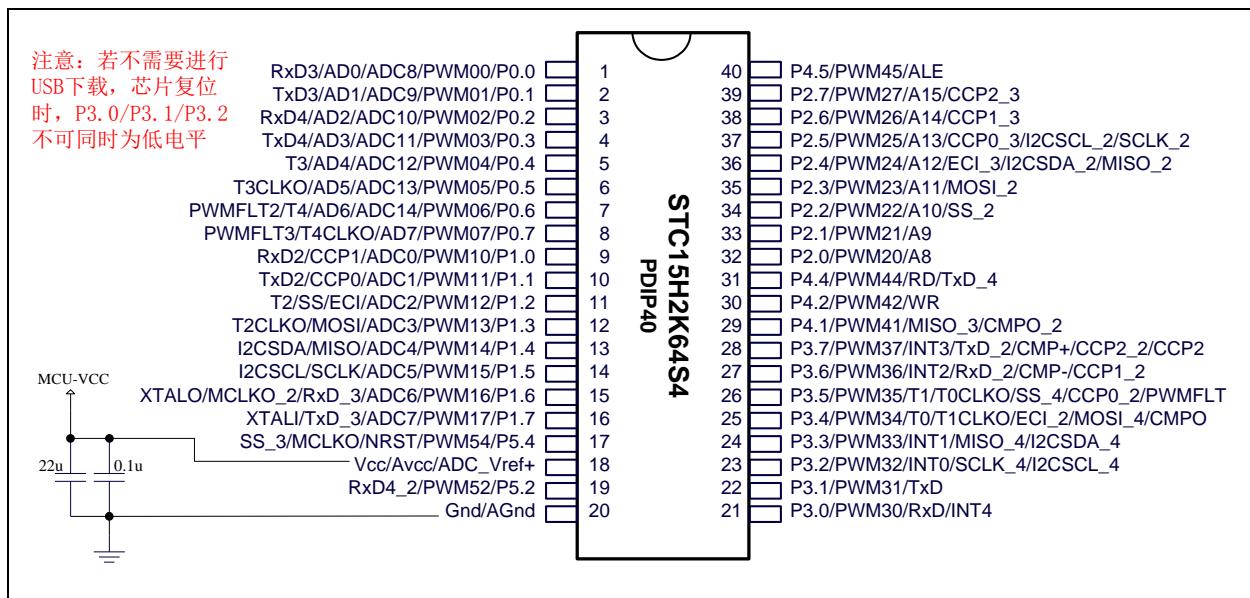
注意：由于传统 STC15F2K 系列的芯片的性能需要提升，外加目前晶圆供货紧张，所以使用 STC8G2K64S4 系列的晶圆生产上面的特殊型号，如需购买以上特殊型号，请提前订货

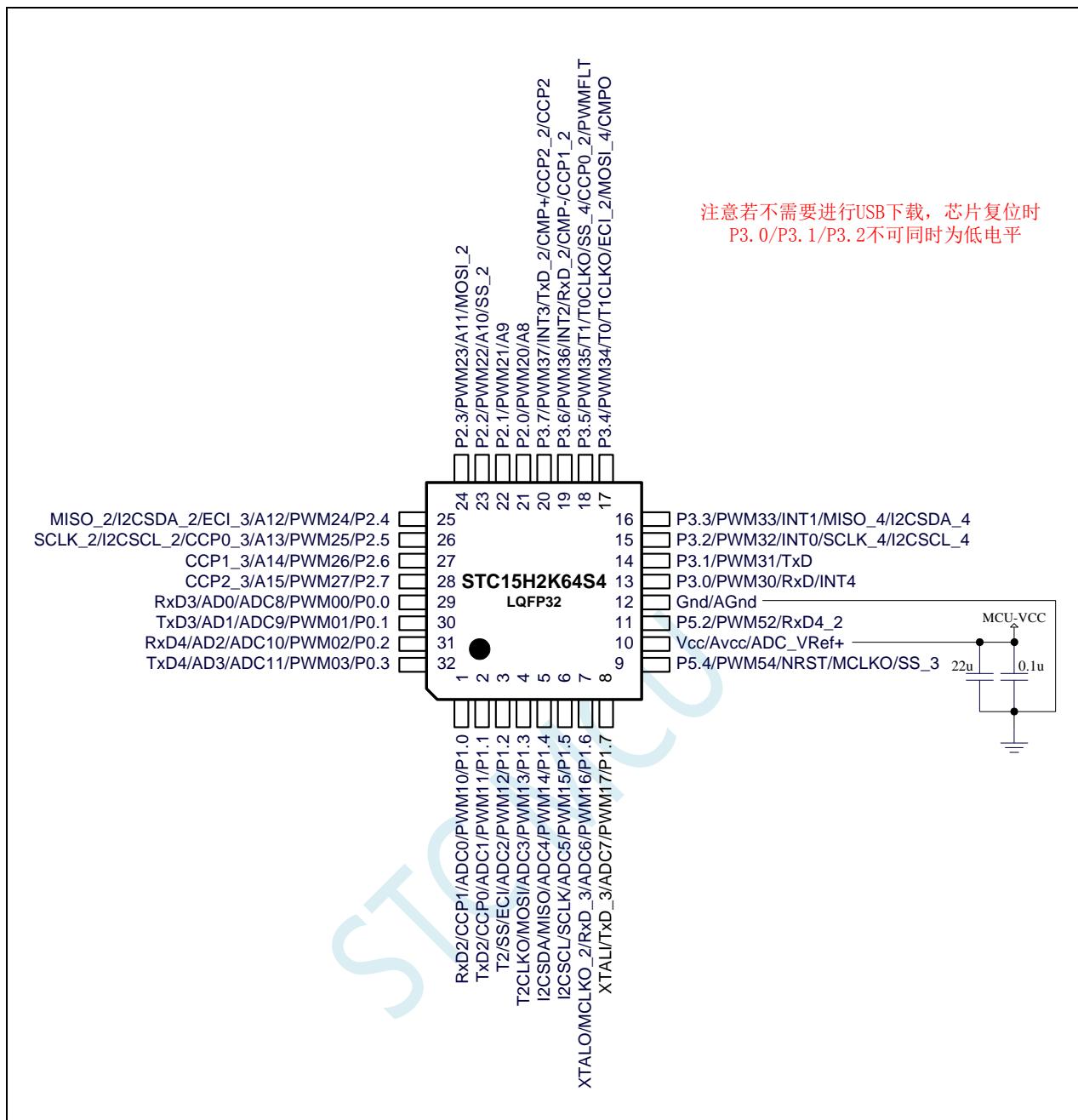
2.7.2 管脚图, 最小系统



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





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 功能脚切换

STC8G 系列单片机的特殊外设串口、SPI、PCA、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。

3.1 功能脚切换相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
P_SW1	外设端口切换寄存器 1	A2H	S1_S[1:0]		CCP_S[1:0]		SPI_S[1:0]		0	-	nnn,0000x
P_SW2	外设端口切换寄存器 2	BAH	EAXFR	-	I2C_S[1:0]	CMPO_S	S4_S	S3_S	S2_S	0x00,0000	

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
MCLKOCR	主时钟输出控制寄存器	FE05H	MCLKO_S								0000,0000

3.1.1 外设端口切换控制寄存器 1 (P_SW1)，串口 1、CCP、SPI 切换

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

S1_S[1:0]: 串口 1 功能脚选择位 ([STC8G1K08-8Pin 系列、STC8G1K08A 系列](#))

S1_S[1:0]	RxD	TxD
00	P3.0	P3.1
01	P3.2	P3.3
10	P5.4	P5.5
11	-	-

CCP_S[1:0]: PCA 功能脚选择位

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

CCP_S[1:0]: PCA 功能脚选择位 ([STC8G1K08A 系列](#))

CCP_S[1:0]	ECI	CCP0	CCP1	CCP2
00	P5.5	P3.2	P3.3	P5.4
01	P5.5	P3.1	P3.3	P5.4

10	P3.1	P3.2	P3.3	P5.5
11	-	-	-	-

CCP_S[1:0]: PCA 功能脚选择位 (**STC8G1K08T 系列**)

CCP_S[1:0]	ECI	CCP0	CCP1	CCP2
00	P1.3	P1.1	P1.0	P3.7
01	P3.4	P3.5	P3.6	P3.7
10	P5.4	P1.3	P1.4	P1.5
11	-	-	-	-

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	P5.4	P4.0	P4.1	P4.3
11	P3.5	P3.4	P3.3	P3.2

SPI_S[1:0]: SPI 功能脚选择位 (**STC8G1K08-8Pin 系列、STC8G1K08A 系列**)

SPI_S[1:0]	SS	MOSI	MISO	SCLK
00	P5.5	P5.4	P3.3	P3.2
01	-	-	-	-
10	-	-	-	-
11	-	-	-	-

3.1.2 外设端口切换控制寄存器 2 (P_SW2), 串口 2/3/4、I2C、

比较器输出切换

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

EAXFR: 扩展 RAM 区特殊功能寄存器 (XFR) 访问控制寄存器

0: 禁止访问 XFR

1: 使能访问 XFR。

当需要访问 XFR 时, 必须先将 EAXFR 置 1, 才能对 XFR 进行正常的读写

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

I2C_S[1:0]: I²C 功能脚选择位 (**STC8G1K08-8Pin 系列、STC8G1K08A 系列**)

I2C_S[1:0]	SCL	SDA
00	P3.2	P3.3
01	P5.4	P5.5
10	-	-
11	-	-

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	P4.6	P4.7

3.1.3 时钟选择寄存器 (MCLKOCR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
MCLKOCR	FE05H	MCLKO_S	MCLKODIV[6:0]						

MCLKO_S: 主时钟输出脚选择位

MCLKO_S	MCLKO
0	P5.4
1	P1.6

3.2 范例程序

3.2.1 串口1切换

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"

sfr P_SW1 = 0xa2;
sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P4M1 = 0xb3;
sfr P4M0 = 0xb4;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    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);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz
```

P_SW1	DATA	0A2H
P0M1	DATA	093H
P0M0	DATA	094H

```

P1M1      DATA      091H
P1M0      DATA      092H
P2M1      DATA      095H
P2M0      DATA      096H
P3M1      DATA      0B1H
P3M0      DATA      0B2H
P4M1      DATA      0B3H
P4M0      DATA      0B4H
P5M1      DATA      0C9H
P5M0      DATA      0CAH

        ORG      0000H
        LJMP    MAIN

        ORG      0100H
MAIN:
        MOV      SP, #5FH
        MOV      P0M0, #00H
        MOV      P0M1, #00H
        MOV      P1M0, #00H
        MOV      P1M1, #00H
        MOV      P2M0, #00H
        MOV      P2M1, #00H
        MOV      P3M0, #00H
        MOV      P3M1, #00H
        MOV      P4M0, #00H
        MOV      P4M1, #00H
        MOV      P5M0, #00H
        MOV      P5M1, #00H

        MOV      P_SW1,#00H      ;RXD/P3.0, TXD/P3.1
        MOV      P_SW1,#40H      ;RXD_2/P3.6, TXD_2/P3.7
        MOV      P_SW1,#80H      ;RXD_3/P1.6, TXD_3/P1.7
        MOV      P_SW1,#0C0H      ;RXD_4/P4.3, TXD_4/P4.4

        SJMP    $
END

```

3.2.2 串口 2 切换

C 语言代码

```
//测试工作频率为11.0592MHz
```

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

```
sfr      P_SW2      = 0xba;
sfr      P0M1      = 0x93;
sfr      P0M0      = 0x94;
sfr      P1M1      = 0x91;
sfr      P1M0      = 0x92;
sfr      P2M1      = 0x95;
sfr      P2M0      = 0x96;
sfr      P3M1      = 0xb1;
sfr      P3M0      = 0xb2;
```

```

sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x00;                                //RXD2/P1.0, TXD2/P1.1
//    P_SW2 = 0x01;                                //RXD2_2/P4.6, TXD2_2/P4.7

    while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

P_SW2	DATA	0BAH
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH
ORG		0000H
LJMP		MAIN
ORG		0100H
MAIN:	MOV	SP, #5FH
	MOV	P0M0, #00H
	MOV	P0M1, #00H
	MOV	P1M0, #00H
	MOV	P1M1, #00H
	MOV	P2M0, #00H
	MOV	P2M1, #00H
	MOV	P3M0, #00H
	MOV	P3M1, #00H

```

MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

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

SJMP    $

END

```

3.2.3 串口 3 切换

C 语言代码

//测试工作频率为 11.0592MHz

```

#include "reg51.h"

sfr    P_SW2      = 0xba;
sfr    P0M1       = 0x93;
sfr    P0M0       = 0x94;
sfr    P1M1       = 0x91;
sfr    P1M0       = 0x92;
sfr    P2M1       = 0x95;
sfr    P2M0       = 0x96;
sfr    P3M1       = 0xb1;
sfr    P3M0       = 0xb2;
sfr    P4M1       = 0xb3;
sfr    P4M0       = 0xb4;
sfr    P5M1       = 0xc9;
sfr    P5M0       = 0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x00;           //RXD3/P0.0, TXD3/P0.1
//    P_SW2 = 0x02;           //RXD3_2/P5.0, TXD3_2/P5.1

    while (1);
}

```

汇编代码

//测试工作频率为11.0592MHz

```

P_SW2      DATA      0BAH

P0M1      DATA      093H
P0M0      DATA      094H
P1M1      DATA      091H
P1M0      DATA      092H
P2M1      DATA      095H
P2M0      DATA      096H
P3M1      DATA      0B1H
P3M0      DATA      0B2H
P4M1      DATA      0B3H
P4M0      DATA      0B4H
P5M1      DATA      0C9H
P5M0      DATA      0CAH

        ORG      0000H
        LJMP     MAIN

        ORG      0100H
MAIN:
        MOV      SP, #5FH
        MOV      P0M0, #00H
        MOV      P0M1, #00H
        MOV      P1M0, #00H
        MOV      P1M1, #00H
        MOV      P2M0, #00H
        MOV      P2M1, #00H
        MOV      P3M0, #00H
        MOV      P3M1, #00H
        MOV      P4M0, #00H
        MOV      P4M1, #00H
        MOV      P5M0, #00H
        MOV      P5M1, #00H

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

        SJMP    $

END

```

3.2.4 串口4切换

C 语言代码

//测试工作频率为11.0592MHz

```

#include "reg51.h"

sfr      P_SW2      = 0xba;
sfr      P0M1      = 0x93;
sfr      P0M0      = 0x94;

```

```

sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x00;                                //RXD4/P0.2, TxD4/P0.3
//    P_SW2 = 0x04;                                //RXD4_2/P5.2, TxD4_2/P5.3

    while (1);
}

```

汇编代码

;测试工作频率为 11.0592MHz

P_SW2	DATA	0BAH
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH
	ORG	0000H
	LJMP	MAIN
MAIN:	ORG	0100H
	MOV	SP, #5FH
	MOV	P0M0, #00H
	MOV	P0M1, #00H

```

MOV      P1M0, #00H
MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

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

SJMP     $                  ;END

```

3.2.5 SPI 切换

C 语言代码

```
//测试工作频率为 11.0592MHz
```

```
#include "reg51.h"

sfr      P_SW1      = 0xa2;

sfr      P0M1      = 0x93;
sfr      P0M0      = 0x94;
sfr      P1M1      = 0x91;
sfr      P1M0      = 0x92;
sfr      P2M1      = 0x95;
sfr      P2M0      = 0x96;
sfr      P3M1      = 0xb1;
sfr      P3M0      = 0xb2;
sfr      P4M1      = 0xb3;
sfr      P4M0      = 0xb4;
sfr      P5M1      = 0xc9;
sfr      P5M0      = 0xca;
```

```
void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;
```

```
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/P5.4, MOSI_3/P4.0, MISO_3/P4.1, SCLK_3/P4.3
// P_SW1 = 0x0c;                                //SS_4/P3.5, MOSI_4/P3.4, MISO_4/P3.3, SCLK_4/P3.2

while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

```

P_SW1      DATA      0A2H

P0M1       DATA      093H
P0M0       DATA      094H
P1M1       DATA      091H
P1M0       DATA      092H
P2M1       DATA      095H
P2M0       DATA      096H
P3M1       DATA      0B1H
P3M0       DATA      0B2H
P4M1       DATA      0B3H
P4M0       DATA      0B4H
P5M1       DATA      0C9H
P5M0       DATA      0CAH

        ORG      0000H
        LJMP    MAIN

        ORG      0100H
MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     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/P5.4, MOSI_3/P4.0, MISO_3/P4.1, SCLK_3/P4.3
;        MOV     P_SW1,#0CH          ;SS_4/P3.5, MOSI_4/P3.4, MISO_4/P3.3, SCLK_4/P3.2

        SJMP   $

```

END

3.2.6 PCA/CCP/PWM 切换

C 语言代码

//测试工作频率为 11.0592MHz

```
#include "reg51.h"

sfr P_SW1 = 0xa2;

sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P4M1 = 0xb3;
sfr P4M0 = 0xb4;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW1 = 0x00;                                //ECI/P1.2, CCP0/P1.1, CCP1/P1.0, CCP2/P3.7
//    P_SW1 = 0x10;                                //ECI_2/P3.4, CCP0_2/P3.5, CCP1_2/P3.6, CCP2_2/P3.7
//    P_SW1 = 0x20;                                //ECI_3/P2.4, CCP0_3/P2.5, CCP1_3/P2.6, CCP2_3/P2.7

    while (1);
}
```

汇编代码

;测试工作频率为 11.0592MHz

P_SW1	DATA	0A2H
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H

```

P5M0      DATA      0CAH

        ORG      0000H
        LJMP    MAIN

        ORG      0100H
MAIN:
        MOV      SP, #5FH
        MOV      P0M0, #00H
        MOV      P0M1, #00H
        MOV      P1M0, #00H
        MOV      P1M1, #00H
        MOV      P2M0, #00H
        MOV      P2M1, #00H
        MOV      P3M0, #00H
        MOV      P3M1, #00H
        MOV      P4M0, #00H
        MOV      P4M1, #00H
        MOV      P5M0, #00H
        MOV      P5M1, #00H

        MOV      P_SW1,#00H      ;ECI/P1.2, CCP0/P1.1, CCP1/P1.0, CCP2/P3.7
        MOV      P_SW1,#10H      ;ECI_2/P3.4, CCP0_2/P3.5, CCP1_2/P3.6, CCP2_2/P3.7
        MOV      P_SW1,#20H      ;ECI_3/P2.4, CCP0_3/P2.5, CCP1_3/P2.6, CCP2_3/P2.7

        SJMP    $

END

```

3.2.7 I2C 切换

C 语言代码

//测试工作频率为 11.0592MHz

```

#include "reg51.h"

sfr      P_SW2      = 0xba;
sfr      P0M1      = 0x93;
sfr      P0M0      = 0x94;
sfr      P1M1      = 0x91;
sfr      P1M0      = 0x92;
sfr      P2M1      = 0x95;
sfr      P2M0      = 0x96;
sfr      P3M1      = 0xb1;
sfr      P3M0      = 0xb2;
sfr      P4M1      = 0xb3;
sfr      P4M0      = 0xb4;
sfr      P5M1      = 0xc9;
sfr      P5M0      = 0xca;

```

```

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;

```

```

P1M1 = 0x00;
P2M0 = 0x00;
P2M1 = 0x00;
P3M0 = 0x00;
P3M1 = 0x00;
P4M0 = 0x00;
P4M1 = 0x00;
P5M0 = 0x00;
P5M1 = 0x00;

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);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

P_SW2	DATA	0BAH
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH
ORG 0000H		
LJMP MAIN		
ORG 0100H		
MAIN:		
MOV	SP, #5FH	
MOV	P0M0, #00H	
MOV	P0M1, #00H	
MOV	P1M0, #00H	
MOV	P1M1, #00H	
MOV	P2M0, #00H	
MOV	P2M1, #00H	
MOV	P3M0, #00H	
MOV	P3M1, #00H	
MOV	P4M0, #00H	
MOV	P4M1, #00H	
MOV	P5M0, #00H	
MOV	P5M1, #00H	
MOV	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      $                 

END

```

3.2.8 比较器输出切换

C 语言代码

//测试工作频率为 11.0592MHz

```

#include "reg51.h"

sfr    P_SW2      = 0xba;
sfr    P0M1       = 0x93;
sfr    P0M0       = 0x94;
sfr    P1M1       = 0x91;
sfr    P1M0       = 0x92;
sfr    P2M1       = 0x95;
sfr    P2M0       = 0x96;
sfr    P3M1       = 0xb1;
sfr    P3M0       = 0xb2;
sfr    P4M1       = 0xb3;
sfr    P4M0       = 0xb4;
sfr    P5M1       = 0xc9;
sfr    P5M0       = 0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x00;           //CMPO/P3.4
//    P_SW2 = 0x08;           //CMPO_2/P4.1

    while (1);
}

```

汇编代码

//测试工作频率为 11.0592MHz

```

P_SW2      DATA      0BAH
P0M1       DATA      093H

```

```

P0M0      DATA      094H
P1M1      DATA      091H
P1M0      DATA      092H
P2M1      DATA      095H
P2M0      DATA      096H
P3M1      DATA      0B1H
P3M0      DATA      0B2H
P4M1      DATA      0B3H
P4M0      DATA      0B4H
P5M1      DATA      0C9H
P5M0      DATA      0CAH

        ORG      0000H
        LJMP     MAIN

        ORG      0100H
MAIN:
        MOV      SP, #5FH
        MOV      P0M0, #00H
        MOV      P0M1, #00H
        MOV      P1M0, #00H
        MOV      P1M1, #00H
        MOV      P2M0, #00H
        MOV      P2M1, #00H
        MOV      P3M0, #00H
        MOV      P3M1, #00H
        MOV      P4M0, #00H
        MOV      P4M1, #00H
        MOV      P5M0, #00H
        MOV      P5M1, #00H

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

        SJMP     $
END

```

3.2.9 主时钟输出切换

C 语言代码

//测试工作频率为11.0592MHz

```

#include "reg51.h"

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

sfr    P_SW2      = 0xba;

sfr    P0M1      = 0x93;
sfr    P0M0      = 0x94;
sfr    P1M1      = 0x91;
sfr    P1M0      = 0x92;
sfr    P2M1      = 0x95;
sfr    P2M0      = 0x96;
sfr    P3M1      = 0xb1;

```

```

sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;
    CLKOCR = 0x04;                                //HIRC/4 output via MCLK0/P5.4
//    CLKOCR = 0x84;                                //HIRC/4 output via MCLK0_2/P1.6
    P_SW2 = 0x00;

    while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

```

P_SW2      DATA      0BAH
CLKOCR     EQU       0FE05H

P0M1      DATA      093H
P0M0      DATA      094H
P1M1      DATA      091H
P1M0      DATA      092H
P2M1      DATA      095H
P2M0      DATA      096H
P3M1      DATA      0B1H
P3M0      DATA      0B2H
P4M1      DATA      0B3H
P4M0      DATA      0B4H
P5M1      DATA      0C9H
P5M0      DATA      0CAH

        ORG      0000H
        LJMP    MAIN

        ORG      0100H
MAIN:
        MOV      SP, #5FH
        MOV      P0M0, #00H
        MOV      P0M1, #00H
        MOV      P1M0, #00H

```

```
MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

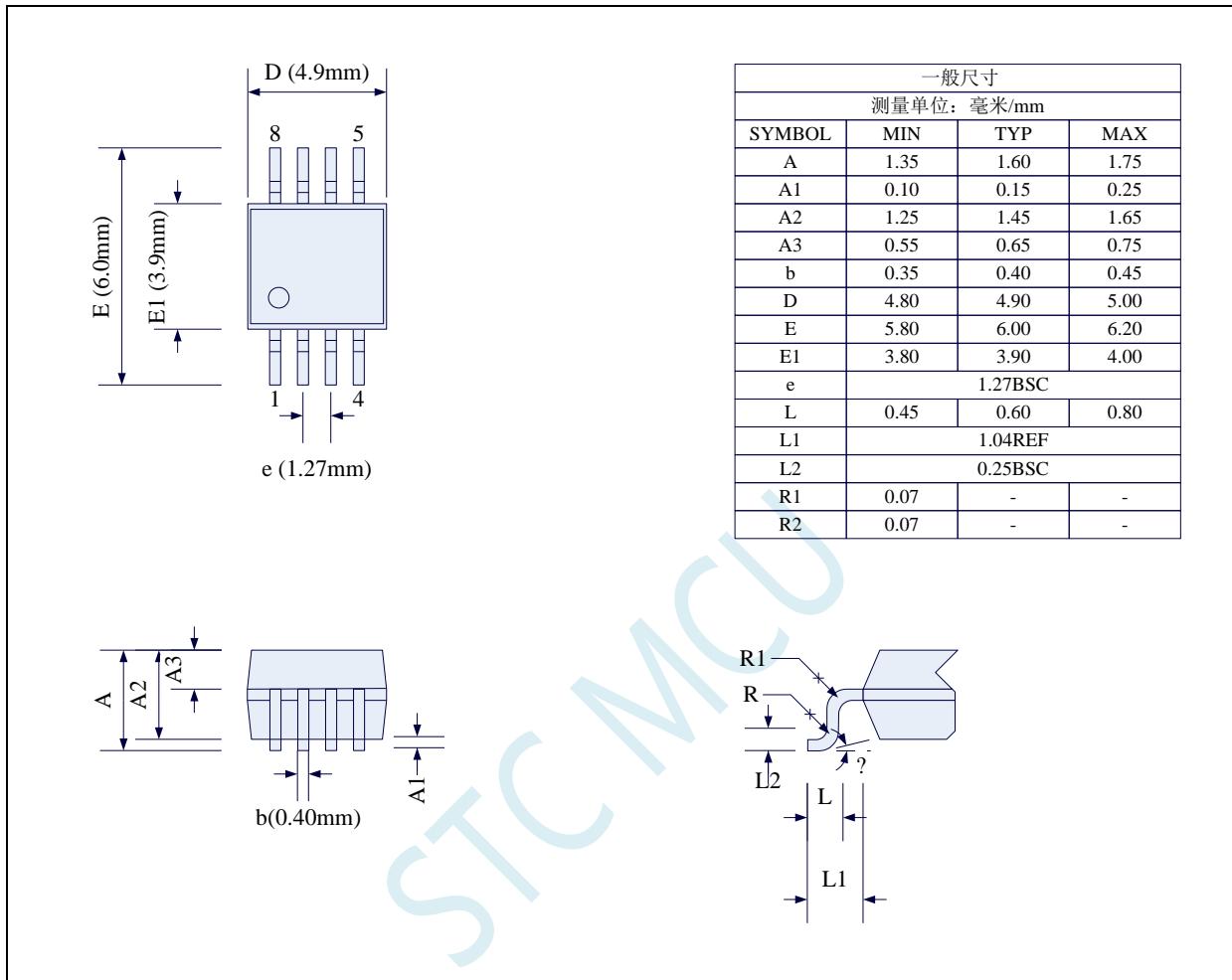
MOV      P_SW2,#80H
MOV      A,#04H          ;HIRC/4 output via MCLKO/P5.4
MOV      A,#84H          ;HIRC/4 output via MCLKO_2/P1.6
MOV      DPTR,#CLKOCR
MOVX    @DPTR,A
MOV      P_SW2,#00H

SJMP    $

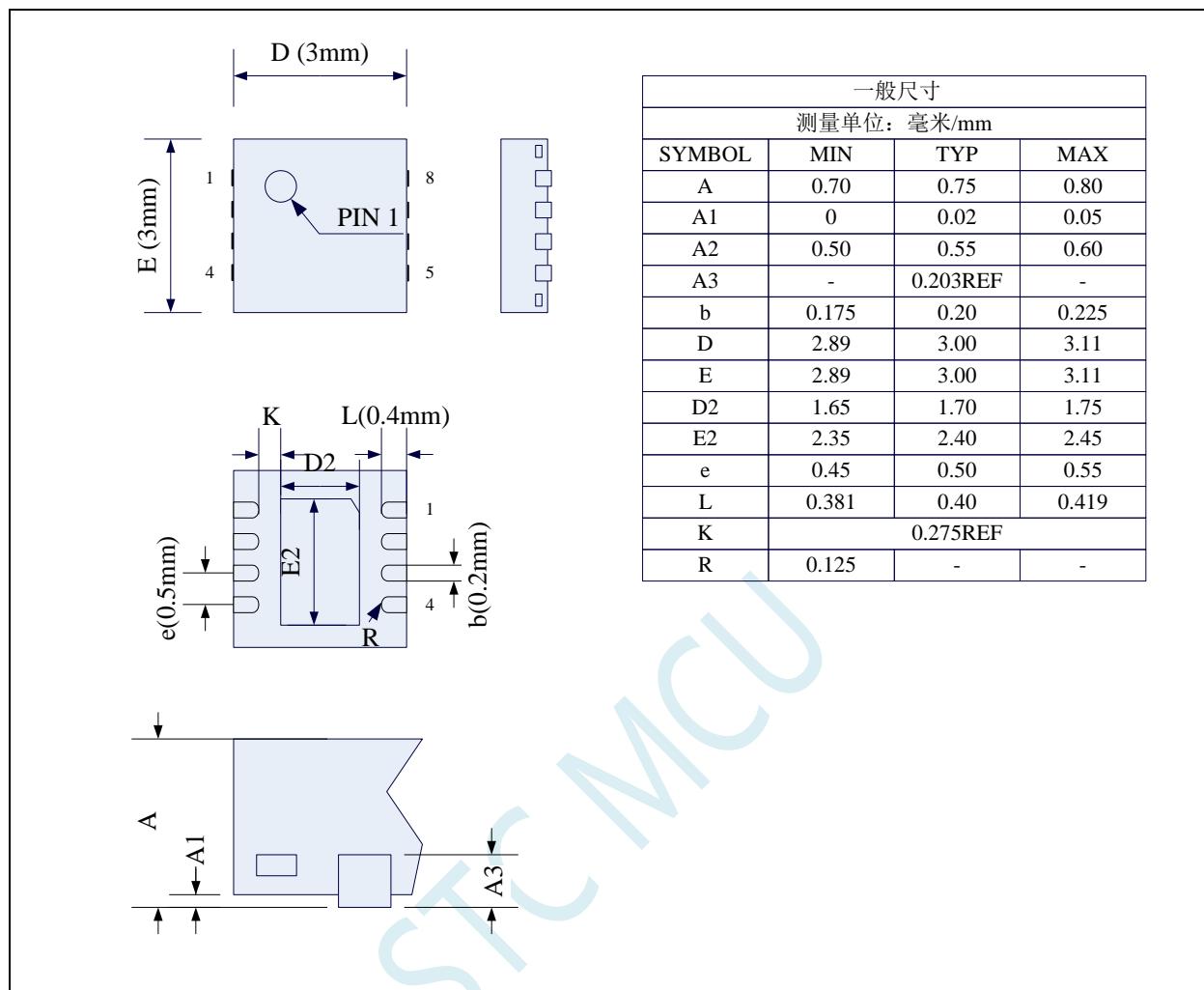
END
```

4 封装尺寸图

4.1 SOP8 封装尺寸图

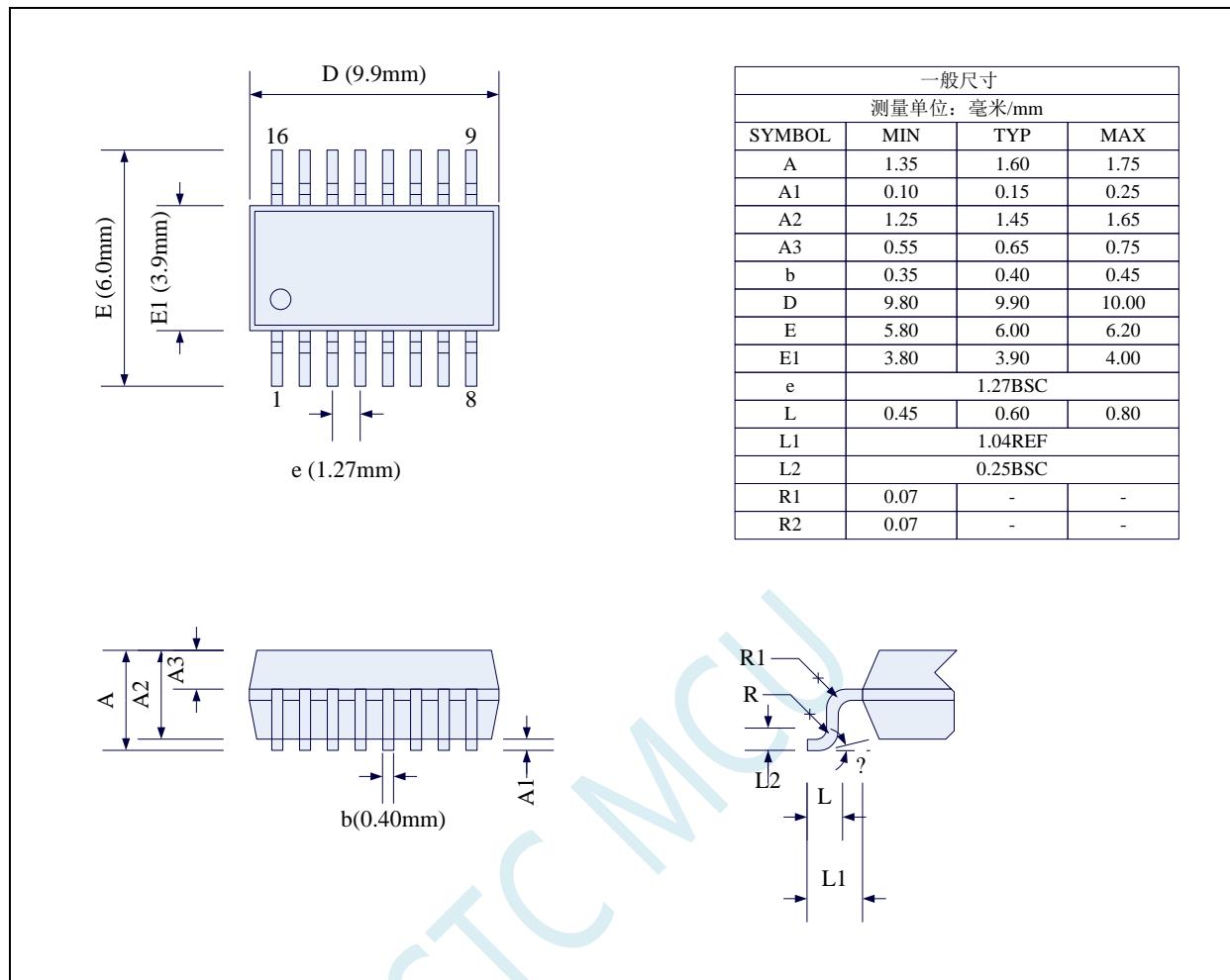


4.2 DFN8 封装尺寸图 (3mm*3mm)

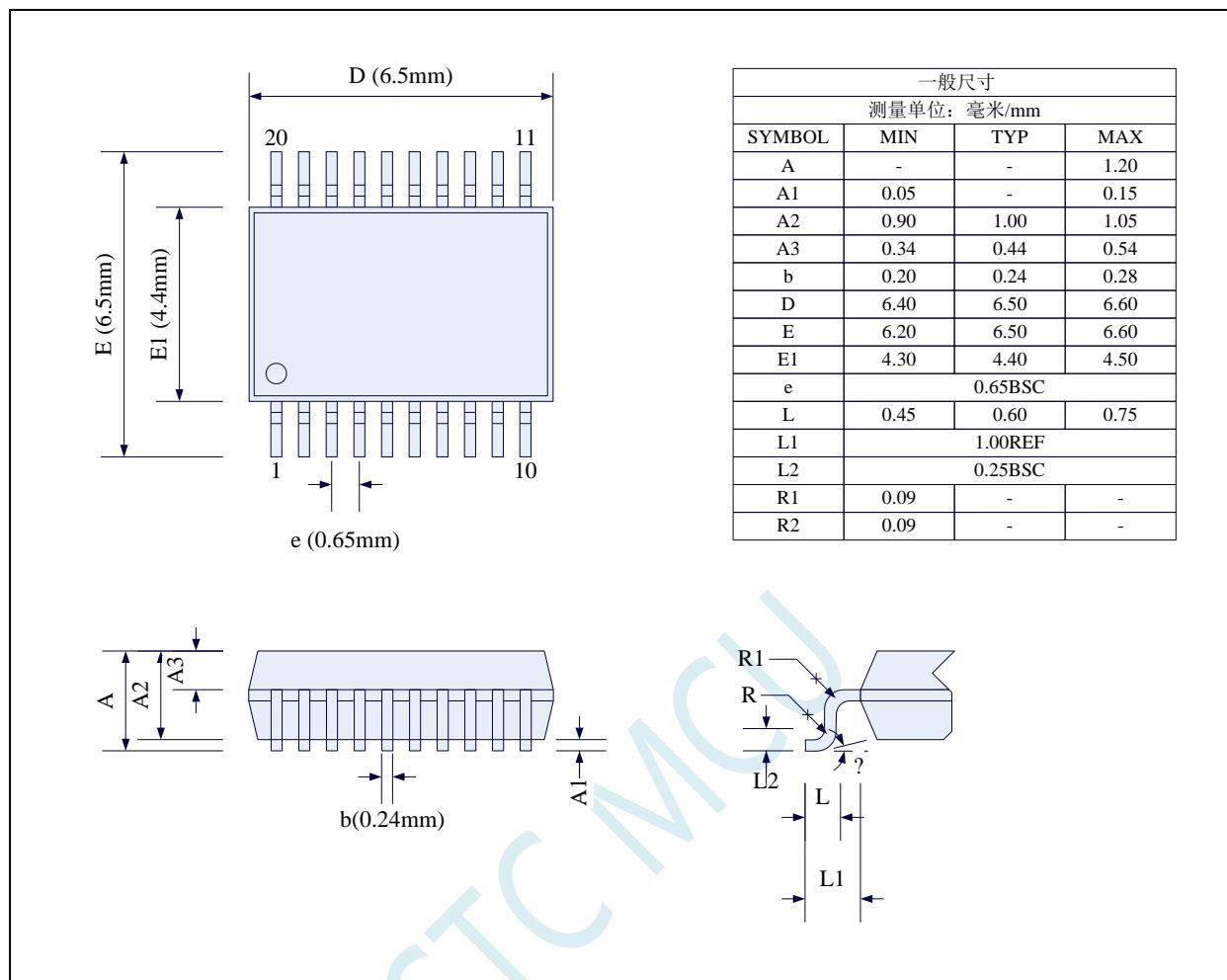


STC 现有 DFN8 封装芯片的背面金属片（衬底），在芯片内部并未接地，在用户的 PCB 板上可以接地，也可以不接地，不会对芯片性能造成影响

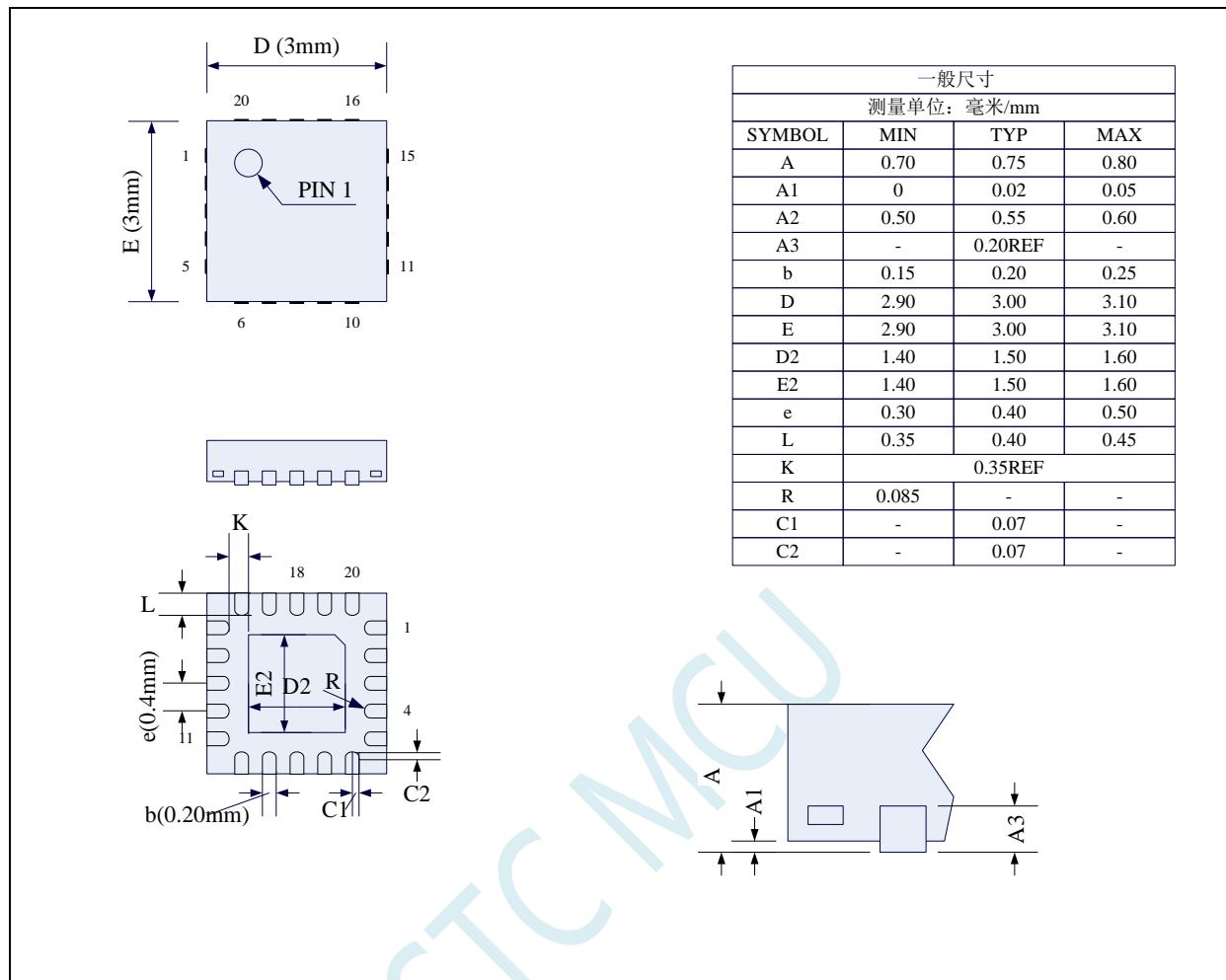
4.3 SOP16 封装尺寸图



4.4 TSSOP20 封装尺寸图

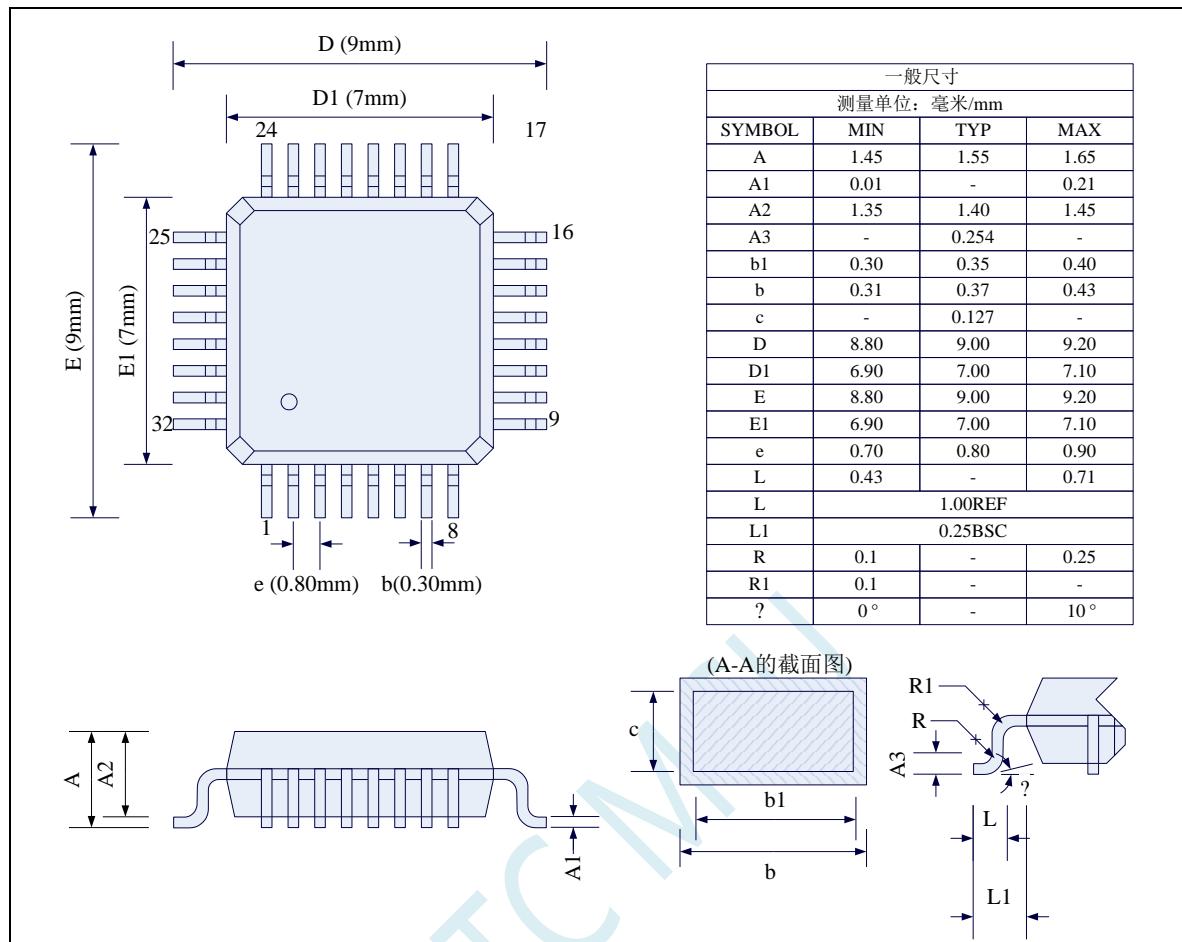


4.5 QFN20 封装尺寸图 (3mm*3mm)

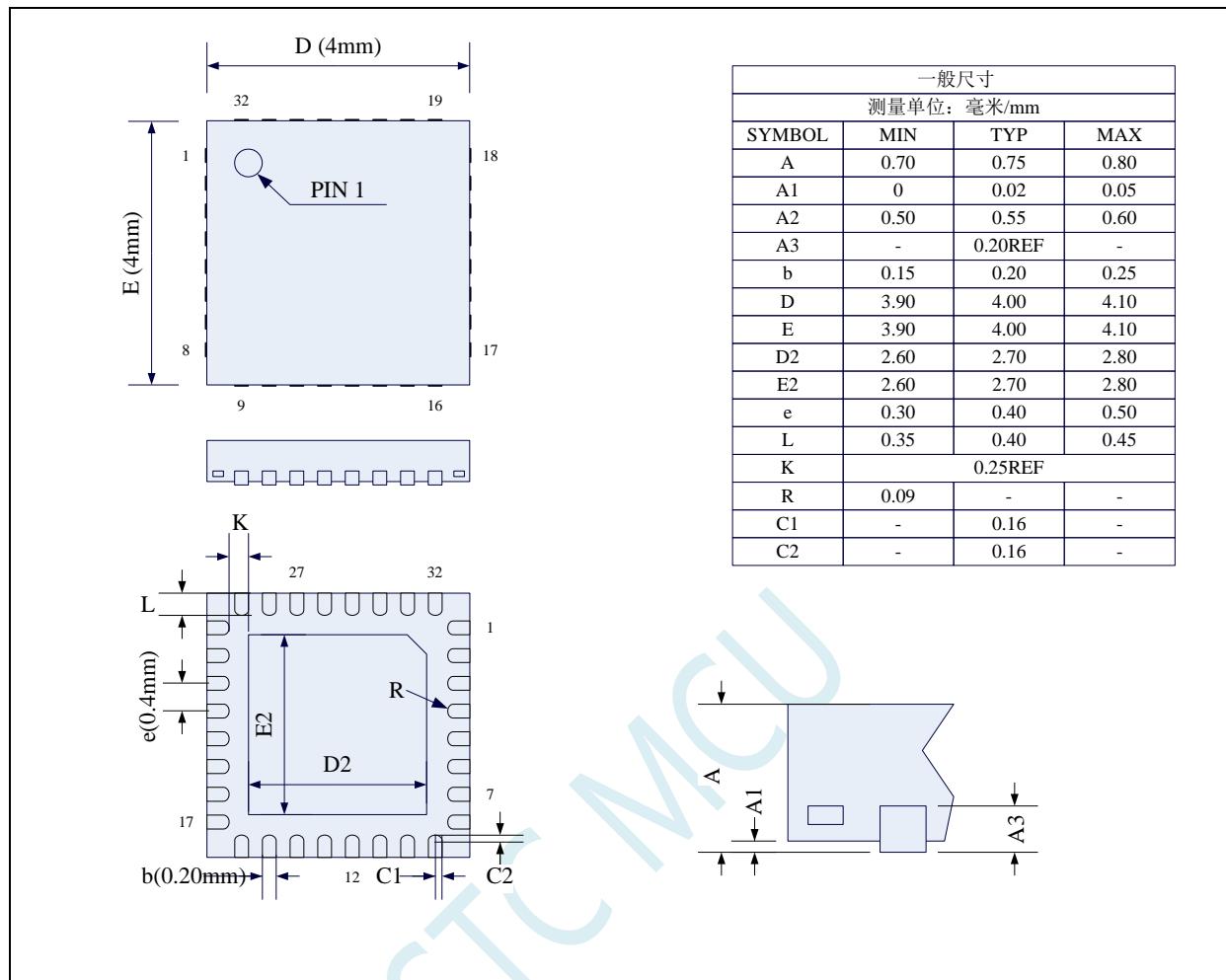


STC 现有 QFN20 封装芯片的背面金属片 (衬底), 在芯片内部并未接地, 在用户的 PCB 板上可以接地, 也可以不接地, 不会对芯片性能造成影响

4.6 LQFP32 封装尺寸图 (9mm*9mm)

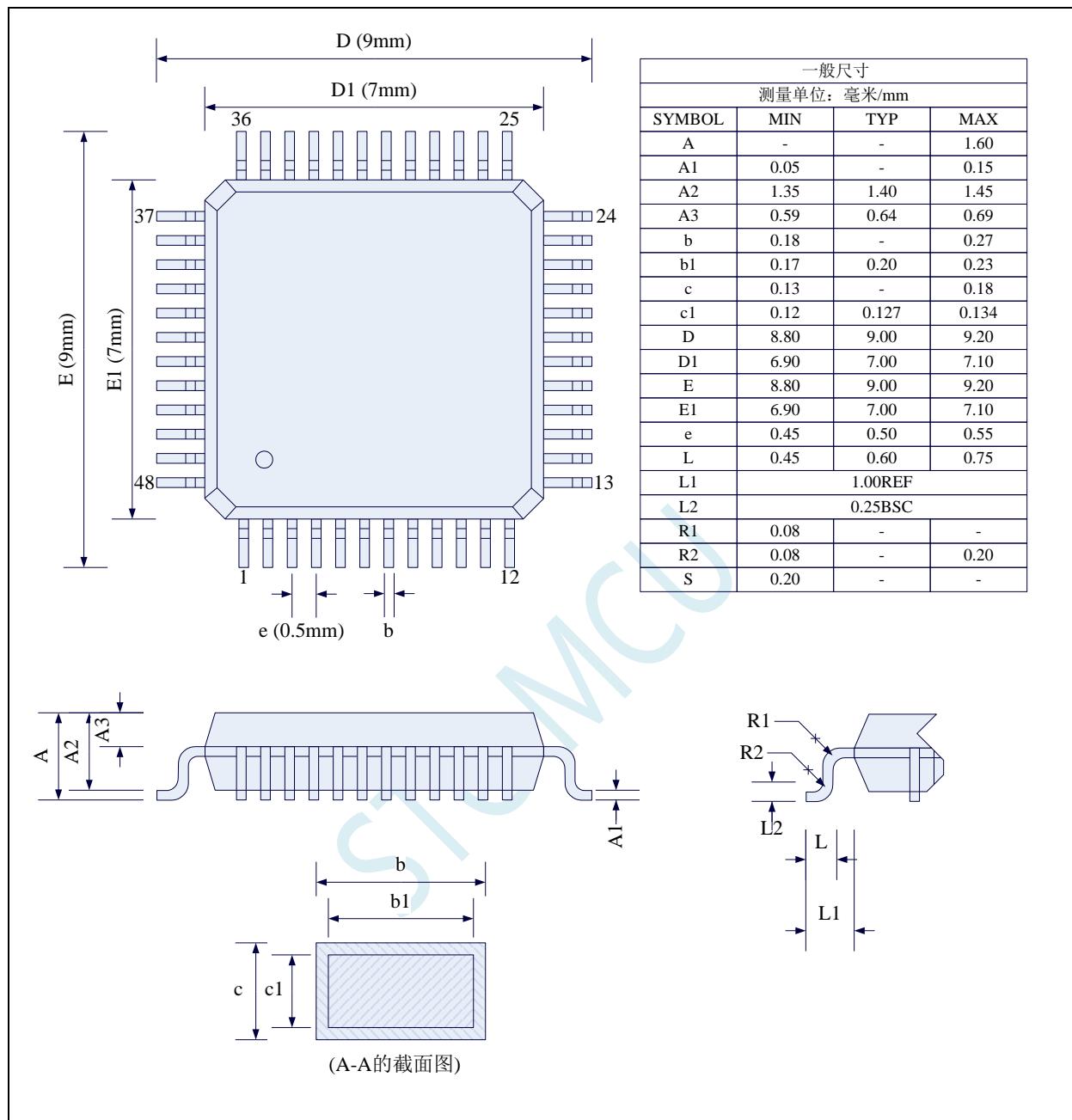


4.7 QFN32 封装尺寸图 (4mm*4mm)

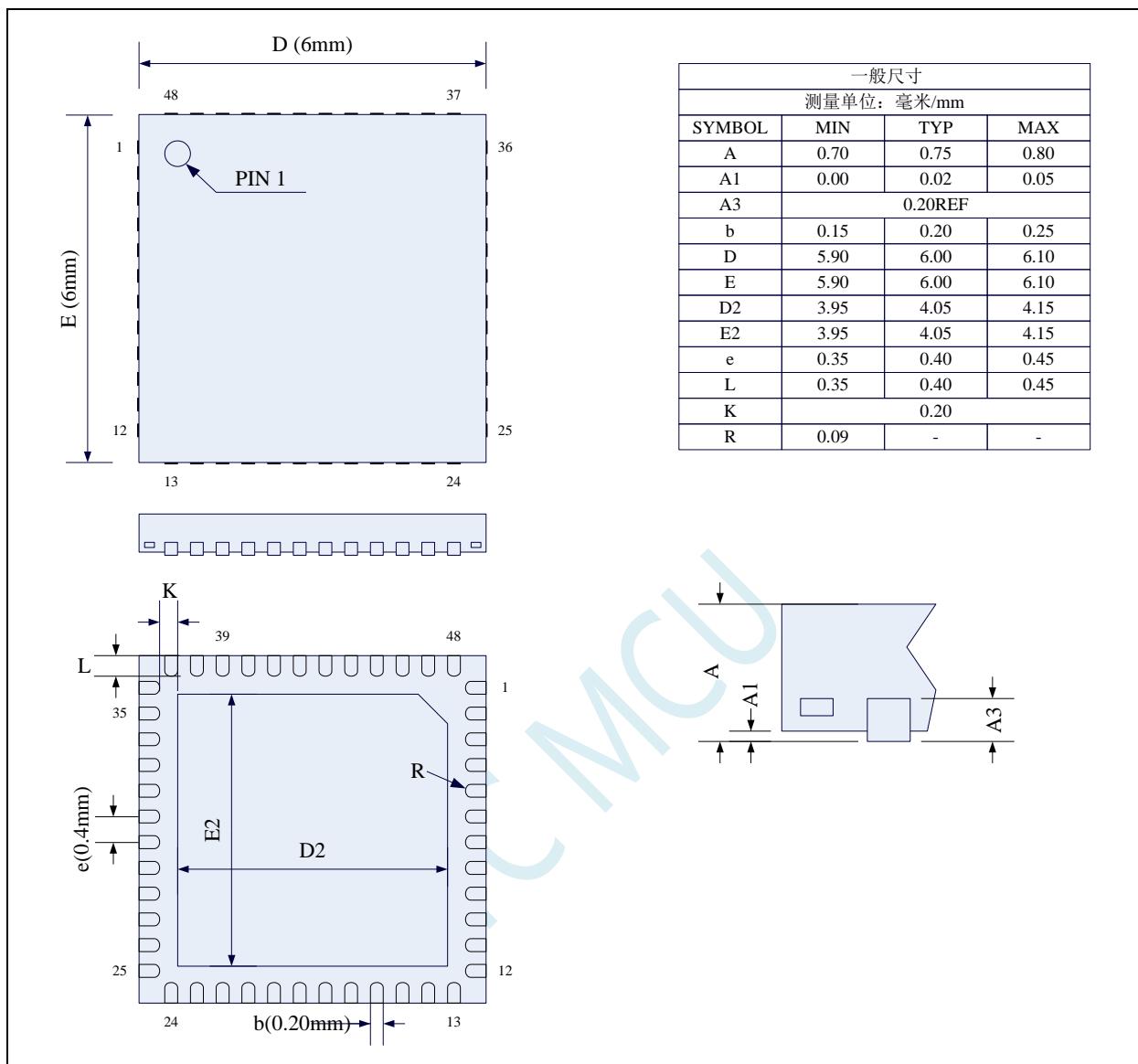


STC 现有 QFN32 封装芯片的背面金属片 (衬底), 在芯片内部并未接地, 在用户的 PCB 板上可以接地, 也可以不接地, 不会对芯片性能造成影响

4.8 LQFP48 封装尺寸图 (9mm*9mm)

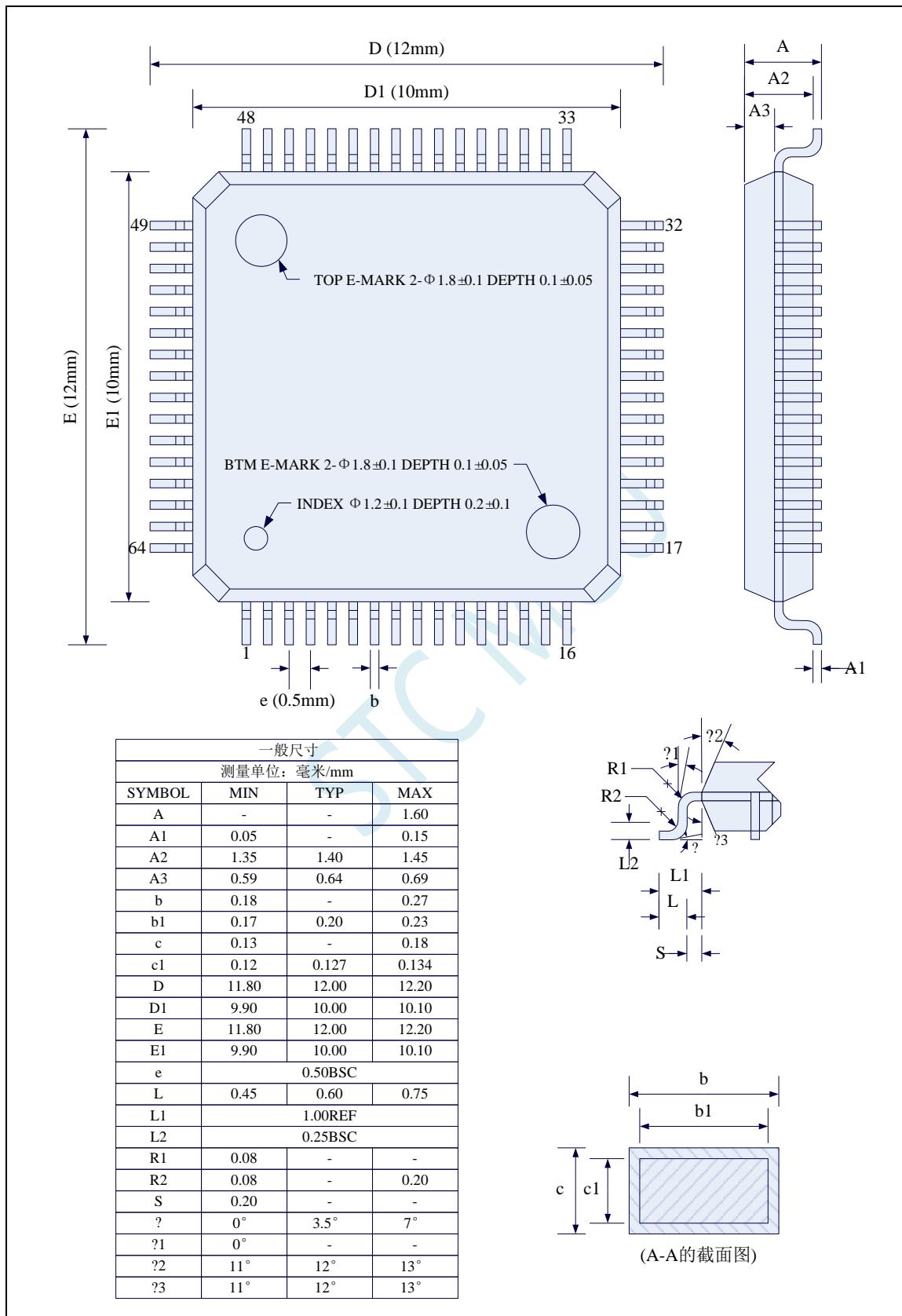


4.9 QFN48 封装尺寸图 (6mm*6mm)

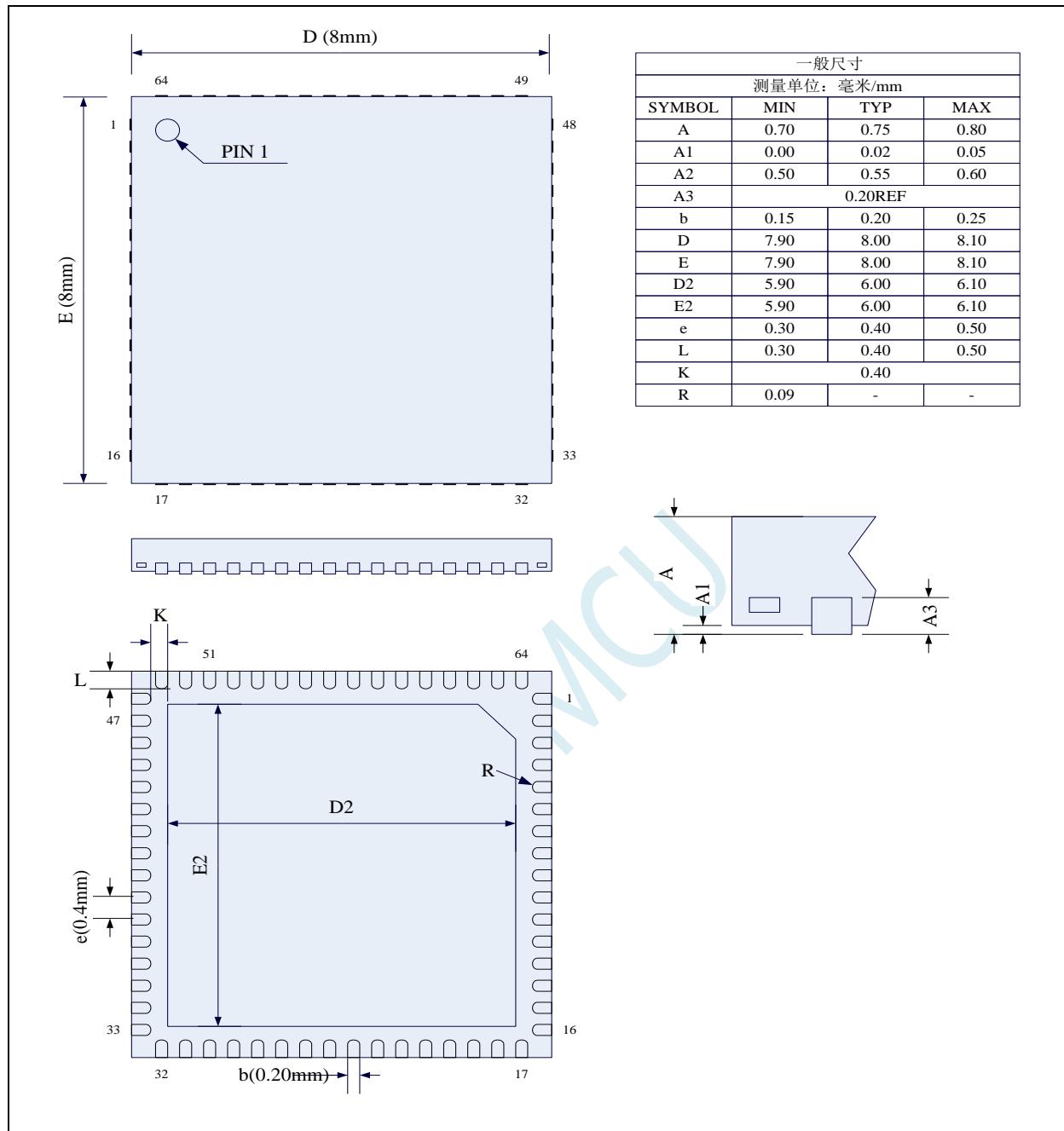


STC 现有 QFN48 封装芯片的背面金属片 (衬底), 在芯片内部并未接地, 在用户的 PCB 板上可以接地, 也可以不接地, 不会对芯片性能造成影响

4.10 LQFP64S 封装尺寸图 (12mm*12mm)

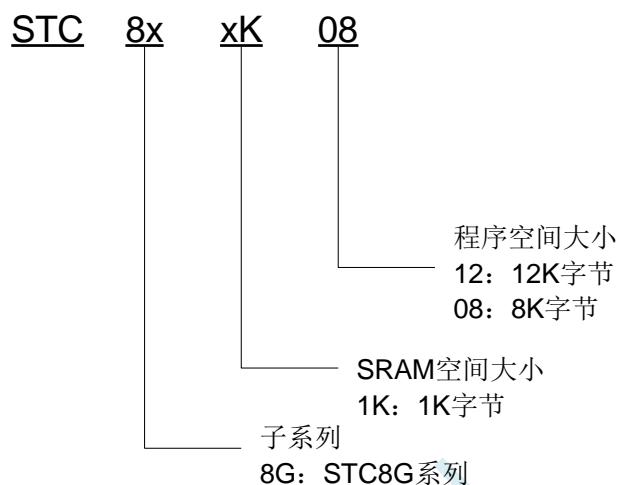


4.11 QFN64 封装尺寸图 (8mm*8mm)



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

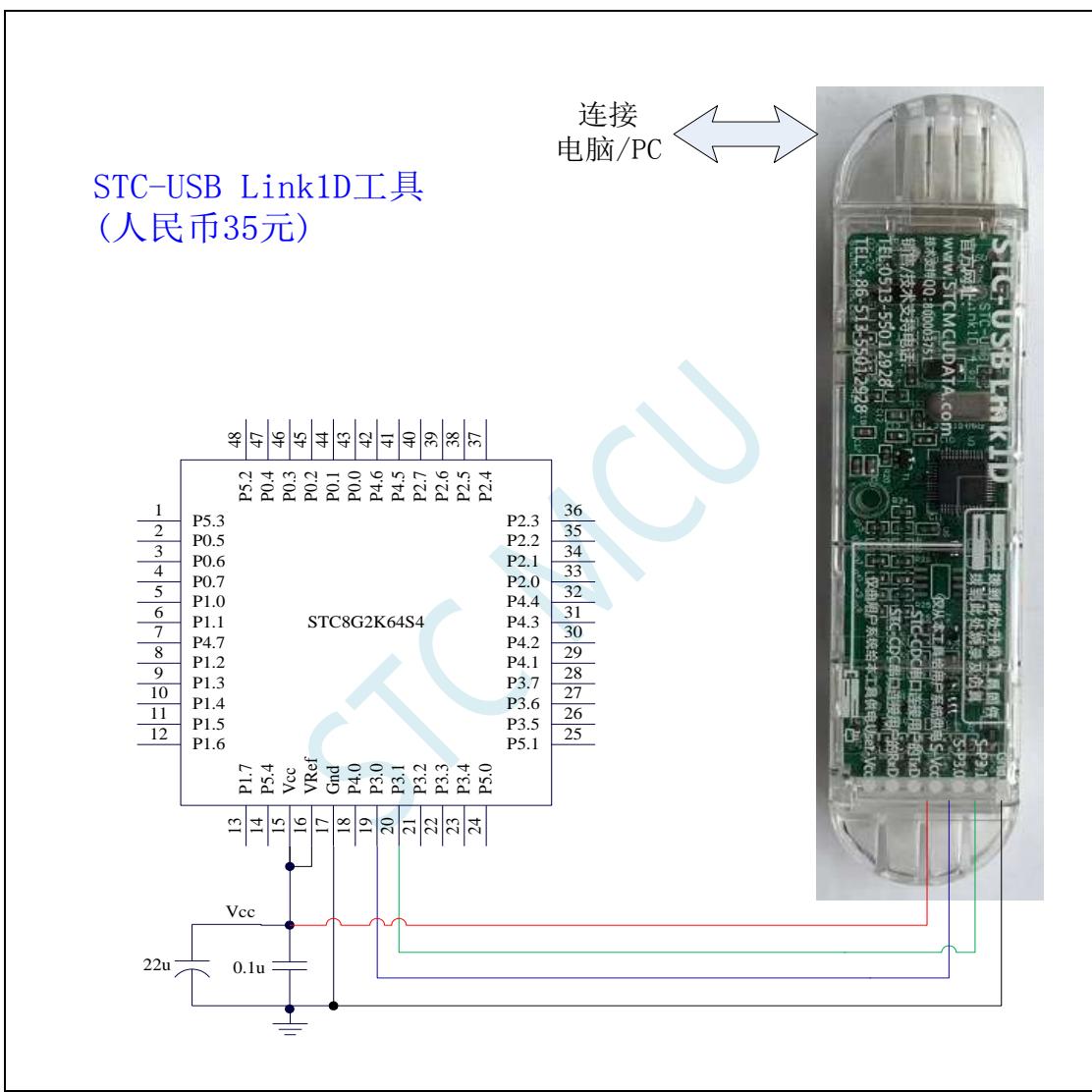
4.12 STC8G 系列单片机命名规则



5 ISP 下载及典型应用线路图

5.1 STC8G 系列 ISP 下载应用线路图

5.1.1 使用 STC-USB Link1D 工具下载，支持在线和脱机下载



ISP 下载步骤:

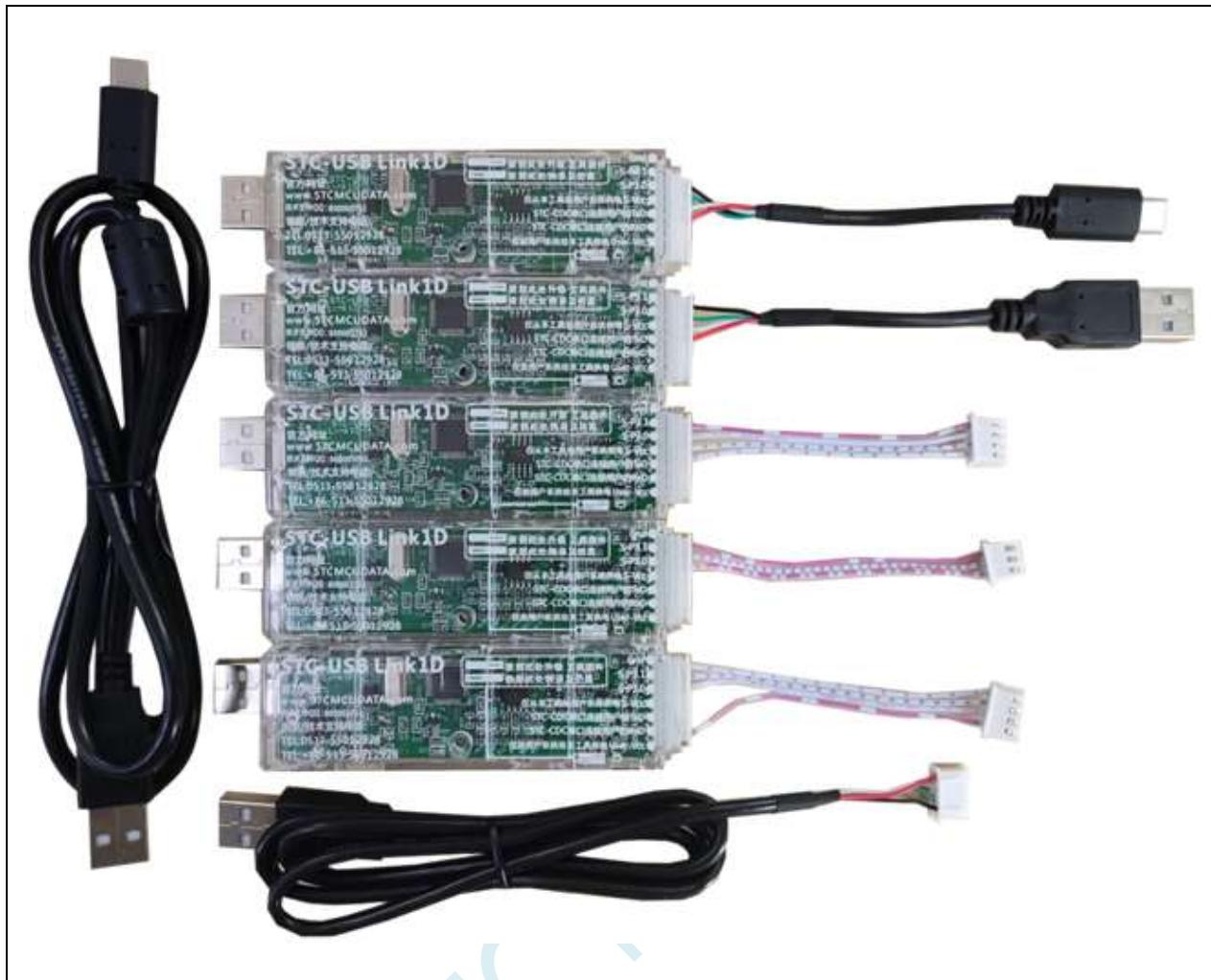
1、按照如图所示的连接方式将 STC-USB Link1D 和目标芯片连接

2、点击 STC-ISP 下载软件中的“下载/编程”按钮

3、开始 ISP 下载

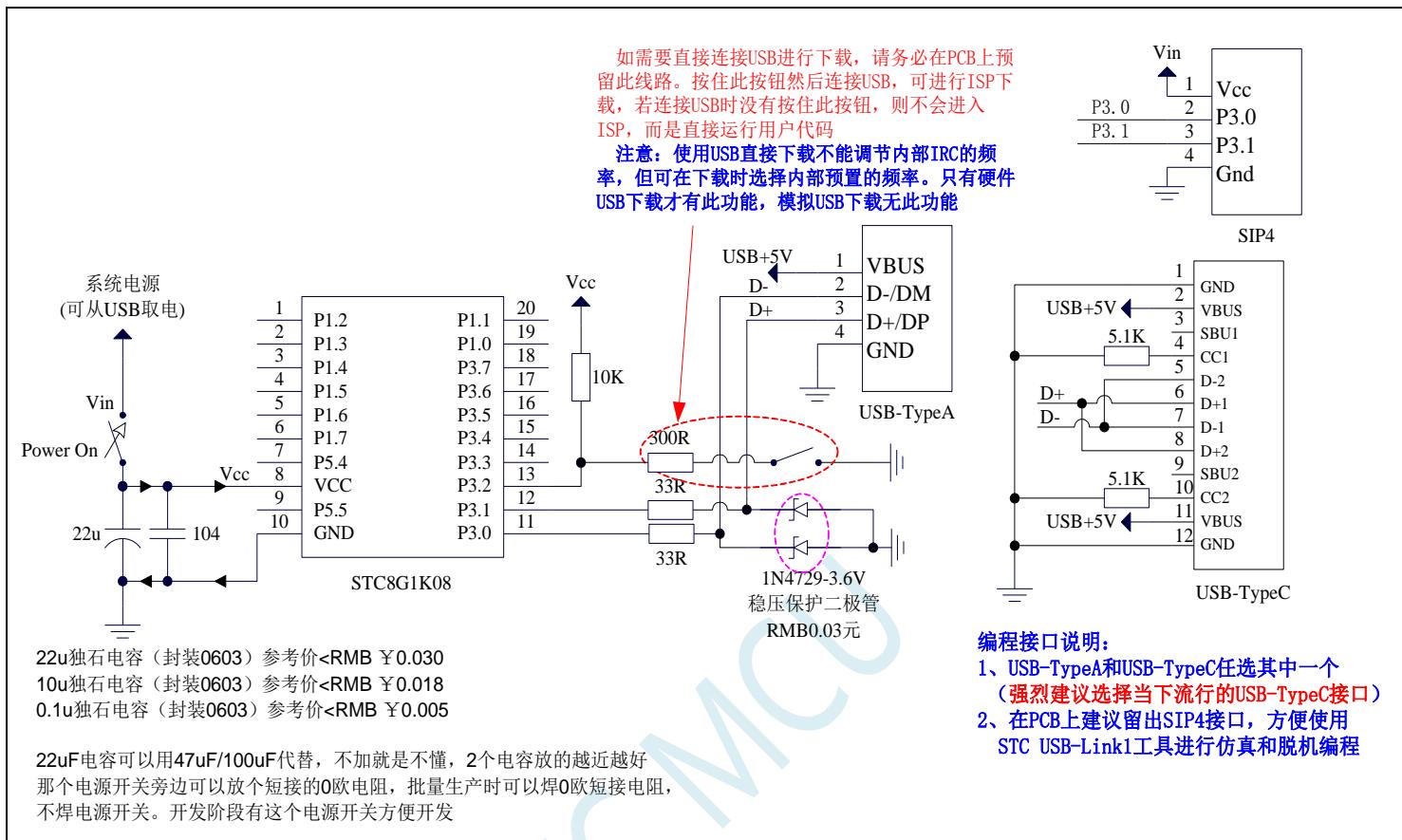
注意: 若是使用 STC-USB Link1D 给目标系统供电, 目标系统的总电流不能大于 200mA, 否则会导致下载失败。

注意: 目前有发现使用 USB 线供电进行 ISP 下载时, 由于 USB 线太细, 在 USB 线上的压降过大, 导致 ISP 下载时供电不足, 所以请在使用 USB 线供电进行 ISP 下载时, 务必使用 USB 加强线。

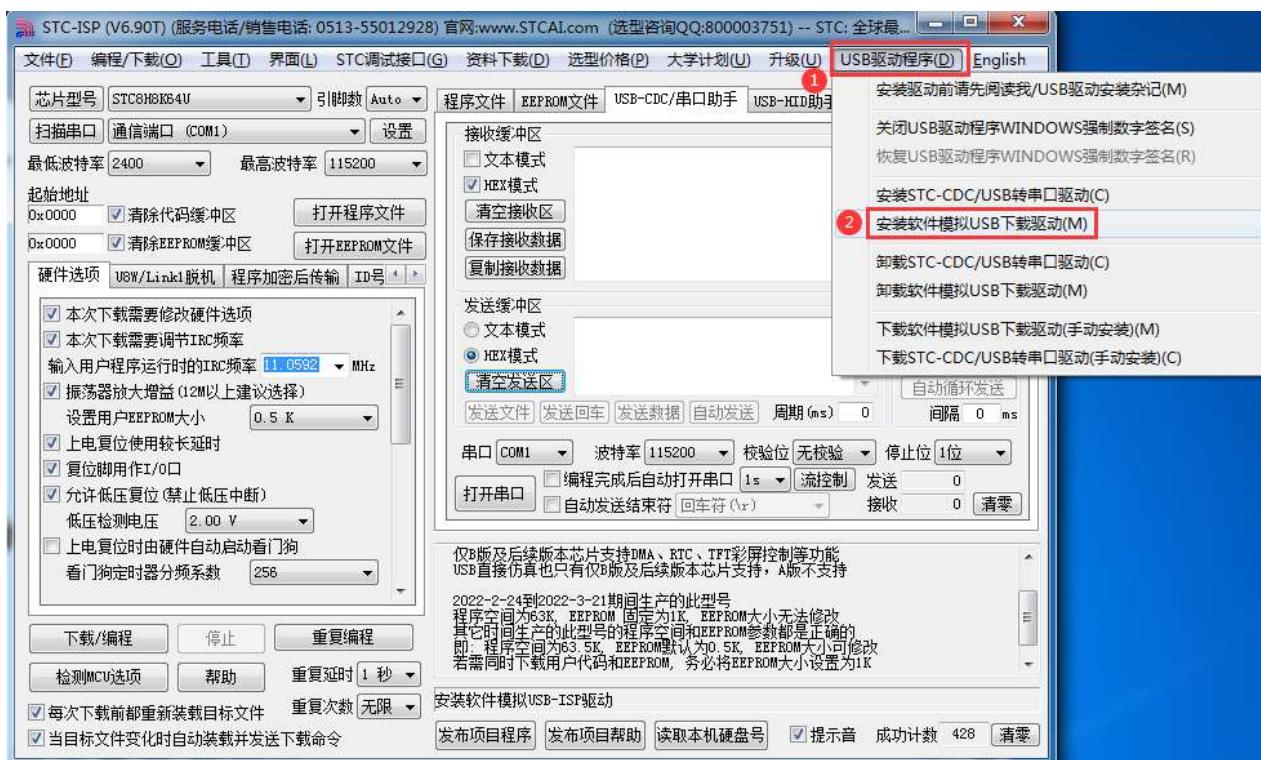


上面 RMB35 是配上面全部的线，是亏本补助大家的

5.1.2 软件模拟 USB 直接 ISP 下载, 建议尝试, 不支持仿真 (5V 系统)



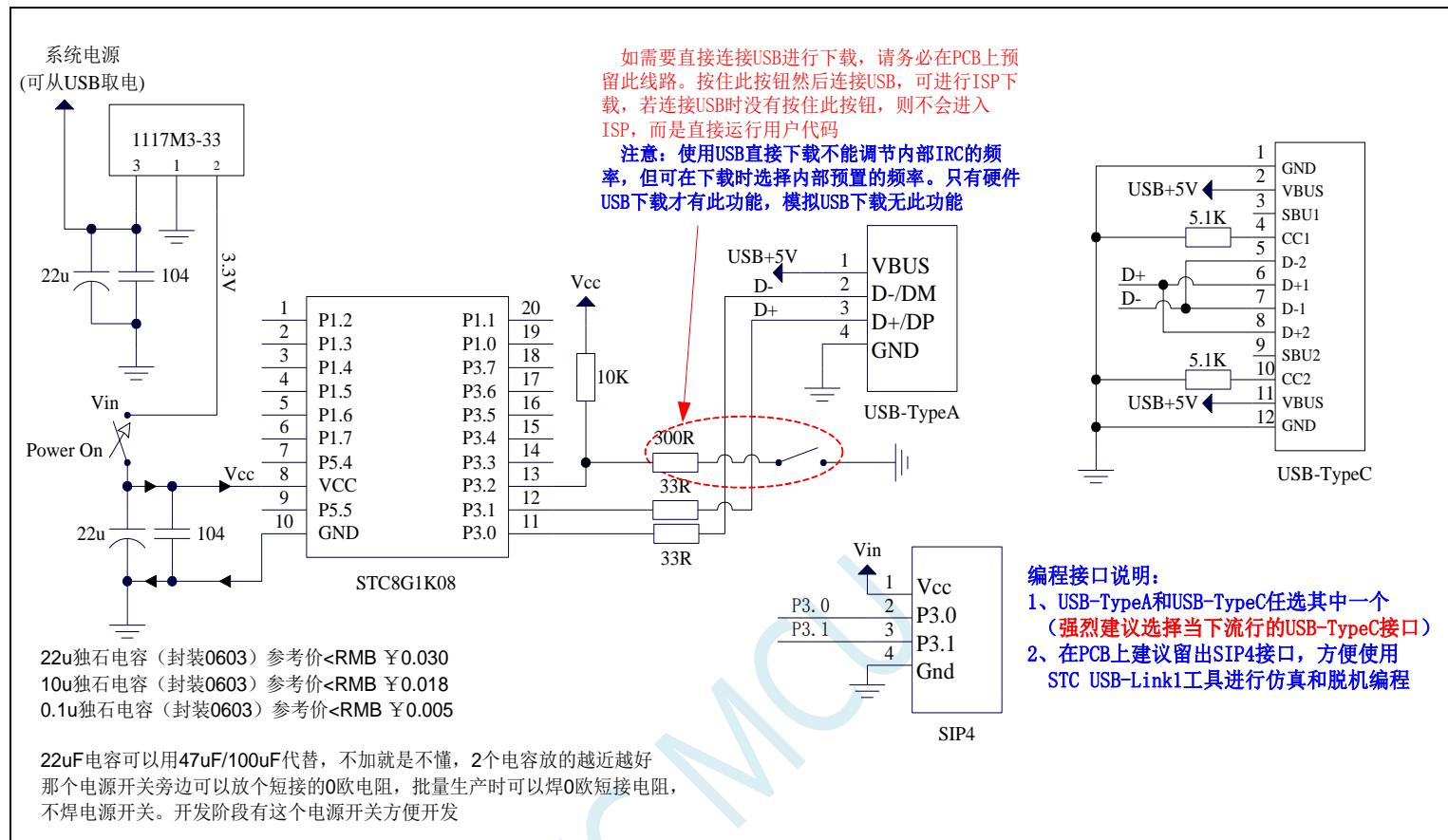
现在 STC 的不带硬件 USB 的 STC8G/STC8H 的 MCU, 基本都支持用软件模拟 USB 下载用户程序, 因为走的是 USB-SCAN 通信协议, 不管任何版本操作系统, 都要安装驱动。在 STC-ISP 下载软件如下图所示的地方安装驱动。



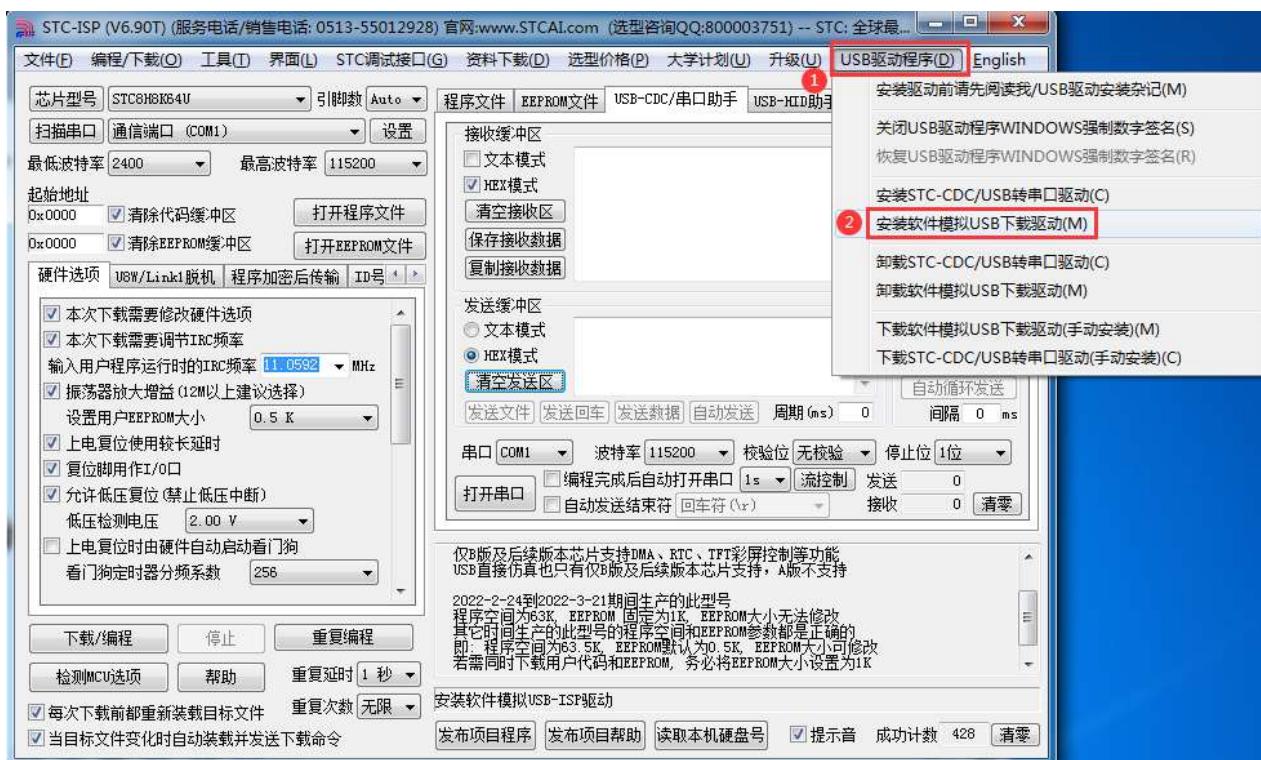
ISP 下载步骤:

- 1、D-/P3.0, D+/P3.1 与 PC-USB 端口连接好
- 2、将 P3.2 与 GND 短接, 实验箱板子上的 P3.2/INT0 按键按下
- 3、给目标芯片重新上电。若目标芯片已经停电, 直接上电即可; 若目标芯片处于通电状态, 则需给目标芯片停电再上电(冷启动)。等待 STC-ISP 下载软件中自动识别出“STC USB Writer (HID1)”识别出来后, 就与 P3.2 状态无关了(此时就不需要一直按着 P3.2 口了, 一直按着把手累坏了不要紧, 将按键按坏就麻烦了)。
- 4、点击下载软件中的“下载/编程”按钮(注意: USB 下载与串口下载的操作顺序不同, 千万千万不要先点下载按钮, 一定到等到电脑端识别出“STC USB Writer (HID1)”设备后, 才能点下载按钮开始下载)

5.1.3 软件模拟 USB 直接 ISP 下载, 建议尝试, 不支持仿真 (3.3V 系统)



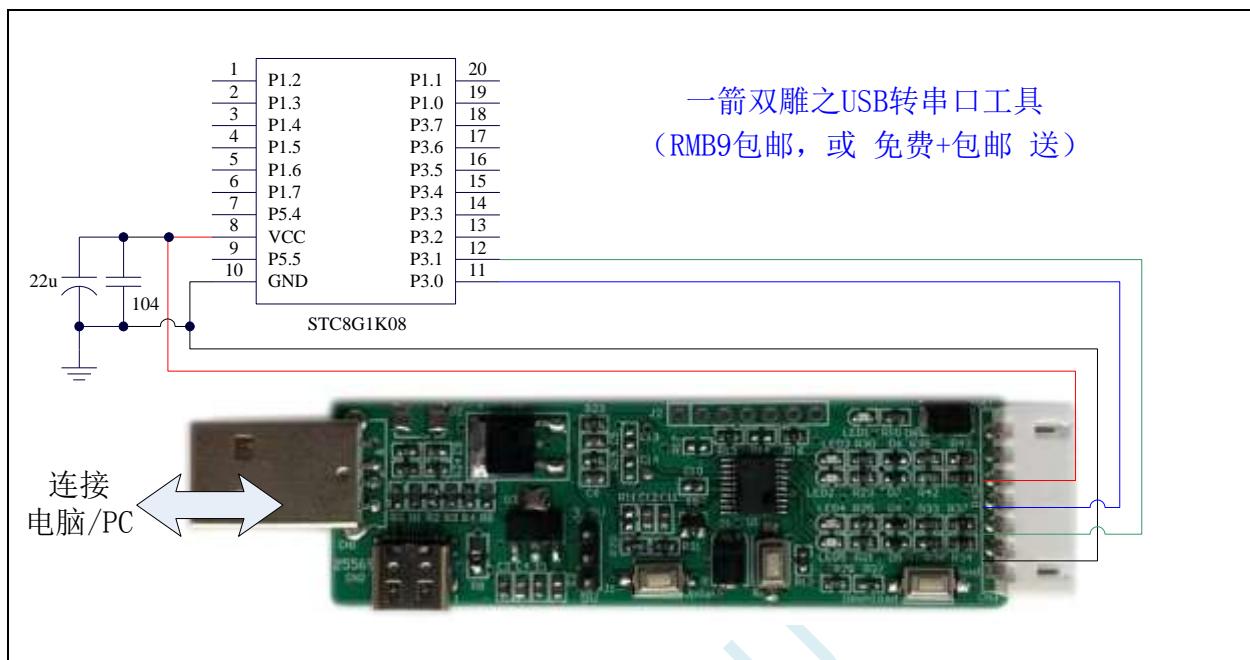
现在 STC 的不带硬件 USB 的 STC8G/STC8H 的 MCU, 基本都支持用软件模拟 USB 下载用户程序, 因为走的是 USB-SCAN 通信协议, 不管任何版本操作系统, 都要安装驱动。在 STC-ISP 下载软件如下图所示的地方安装驱动。



ISP 下载步骤:

- 1、D-/P3.0, D+/P3.1 与 PC-USB 端口连接好
- 2、将 P3.2 与 GND 短接, 实验箱板子上的 P3.2/INT0 按键按下
- 3、给目标芯片重新上电。若目标芯片已经停电, 直接上电即可; 若目标芯片处于通电状态, 则需给目标芯片停电再上电(冷启动)。等待 STC-ISP 下载软件中自动识别出“STC USB Writer (HID1)”识别出来后, 就与 P3.2 状态无关了(此时就不需要一直按着 P3.2 口了, 一直按着把手累坏了不要紧, 将按键按坏就麻烦了)。
- 4、点击下载软件中的“下载/编程”按钮(注意: USB 下载与串口下载的操作顺序不同, 千万千万不要先点下载按钮, 一定到等到电脑端识别出“STC USB Writer (HID1)”设备后, 才能点下载按钮开始下载)

5.1.4 使用一箭双雕之 USB 转串口工具下载



ISP 下载步骤:

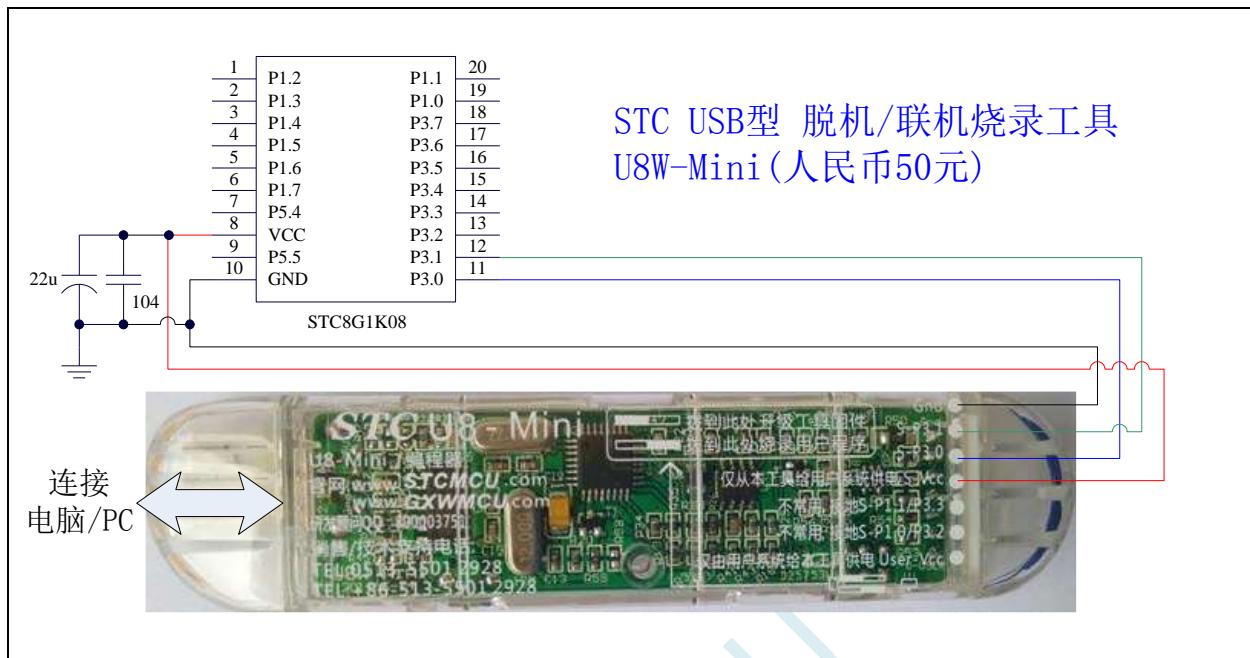
- 1、按照如图所示的连接方式将 USB 转串口工具和目标芯片连接
- 2、点击 STC-ISP 下载软件中的“下载/编程”按钮
- 3、开始 ISP 下载

注意: 目前有发现使用 USB 线供电进行 ISP 下载时, 由于 USB 线太细, 在 USB 线上的压降过大, 导致 ISP 下载时供电不足, 所以请在使用 USB 线供电进行 ISP 下载时, 务必使用 USB 加强线。



一箭双雕之USB转串口工具
(人民币9元包邮销售, 只含一条SIP7-SIP4的线, 亏本补助大家)

5.1.5 使用 U8-Mini 工具下载，支持 ISP 在线和脱机下载，也可支持仿真



ISP 下载步骤:

- 1、按照如图所示的连接方式将 U8-Mini 和目标芯片连接
- 2、点击 STC-ISP 下载软件中的“下载/编程”按钮
- 3、开始 ISP 下载

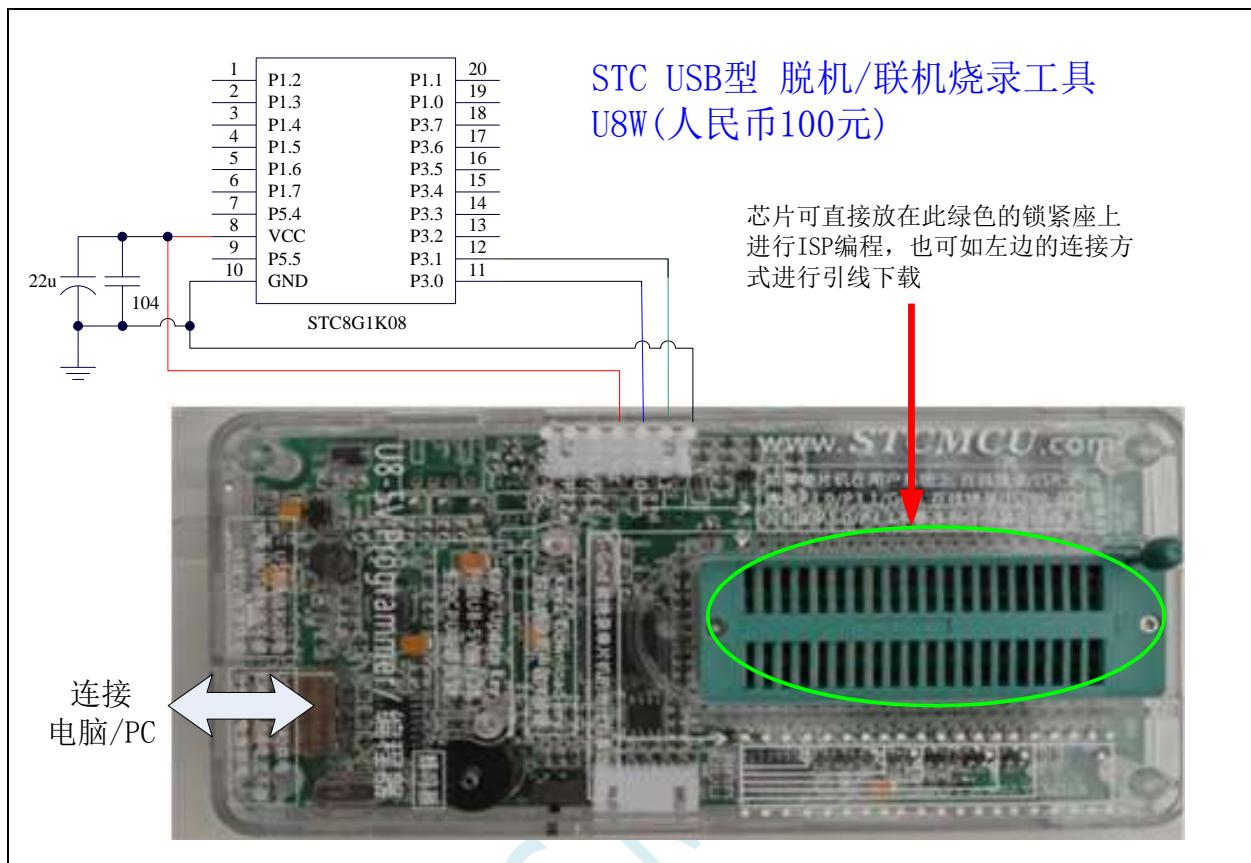
注意: 若是使用 U8-Mini 给目标系统供电, 目标系统的总电流不能大于 200mA, 否则会导致下载失败。

注意: 目前有发现使用 USB 线供电进行 ISP 下载时, 由于 USB 线太细, 在 USB 线上的压降过大, 导致 ISP 下载时供电不足, 所以请在使用 USB 线供电进行 ISP 下载时, 务必使用 USB 加强线。

若要使用 U8-Mini 进行仿真, 首先必须将 U8-Mini 设置为直通模式。U8W/U8W-Mini 实现 USB 转串口直通模式的方法如下:

- 1、首先 U8W/U8W-Mini 固件必须升级到 v1.37 及以上版本
- 2、U8W/U8W-Mini 上电后为正常下载模式, 此时按住工具上的 Key1 (下载) 按键不要松开, 再按一下 Key2 (电源) 按键, 然后放开 Key2 (电源) 按键 后, 再松开 Key1 (下载) 按键, U8W/U8W-Mini 会进入 USB 转串口直通模式。(按下 Key1 → 按下 Key2 → 松开 Key2 → 松开 Key1)
- 3、进入直通模式的 U8W/U8W-Mini 工具只是简单的 USB 转串口不具备脱机下载功能, 若要恢复 U8W/U8W-Mini 的原有功能, 只需要再次单独按一下 Key2 (电源) 按键 即可

5.1.6 使用 U8W 工具下载，支持 ISP 在线和脱机下载，也可支持仿真



ISP 下载步骤 (连线方式):

- 1、按照如图所示的连接方式将 U8W 和目标芯片连接
- 2、点击 STC-ISP 下载软件中的“下载/编程”按钮
- 3、开始 ISP 下载

注意：若是使用 U8W 给目标系统供电，目标系统的总电流不能大于 200mA，否则会导致下载失败。

ISP 下载步骤 (在板方式):

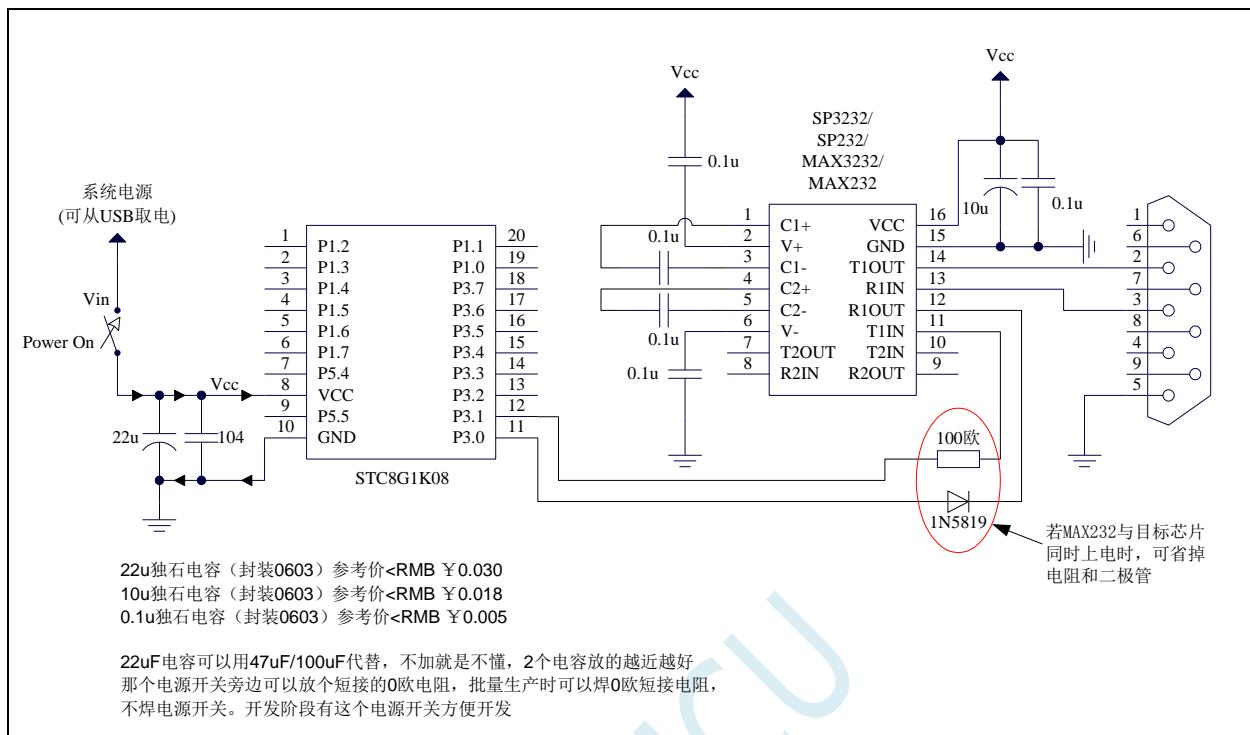
- 1、将芯片按照 1 脚靠近锁紧扳手、管脚向下靠齐的方向放置好目标芯片
- 2、点击 STC-ISP 下载软件中的“下载/编程”按钮
- 3、开始 ISP 下载

注意：目前有发现使用 USB 线供电进行 ISP 下载时，由于 USB 线太细，在 USB 线上的压降过大，导致 ISP 下载时供电不足，所以请在使用 USB 线供电进行 ISP 下载时，务必使用 USB 加强线。

若要使用 U8W 进行仿真，首先必须将 U8W 设置为直通模式。U8W/U8W-Mini 实现 USB 转串口直通模式的方法如下：

- 1、首先 U8W/U8W-Mini 固件必须升级到 v1.37 及以上版本
- 2、U8W/U8W-Mini 上电后为正常下载模式，此时按住工具上的 Key1（下载）按键不要松开，再按一下 Key2（电源）按键，然后放开 Key2（电源）按键 后，再松开 Key1（下载）按键，U8W/U8W-Mini 会进入 USB 转串口直通模式。（按下 Key1 ➔ 按下 Key2 ➔ 松开 Key2 ➔ 松开 Key1）
- 3、进入直通模式的 U8W/U8W-Mini 工具只是简单的 USB 转串口不具备脱机下载功能，若需要恢复 U8W/U8W-Mini 的原有功能，只需要再次单独按一下 Key2（电源）按键 即可

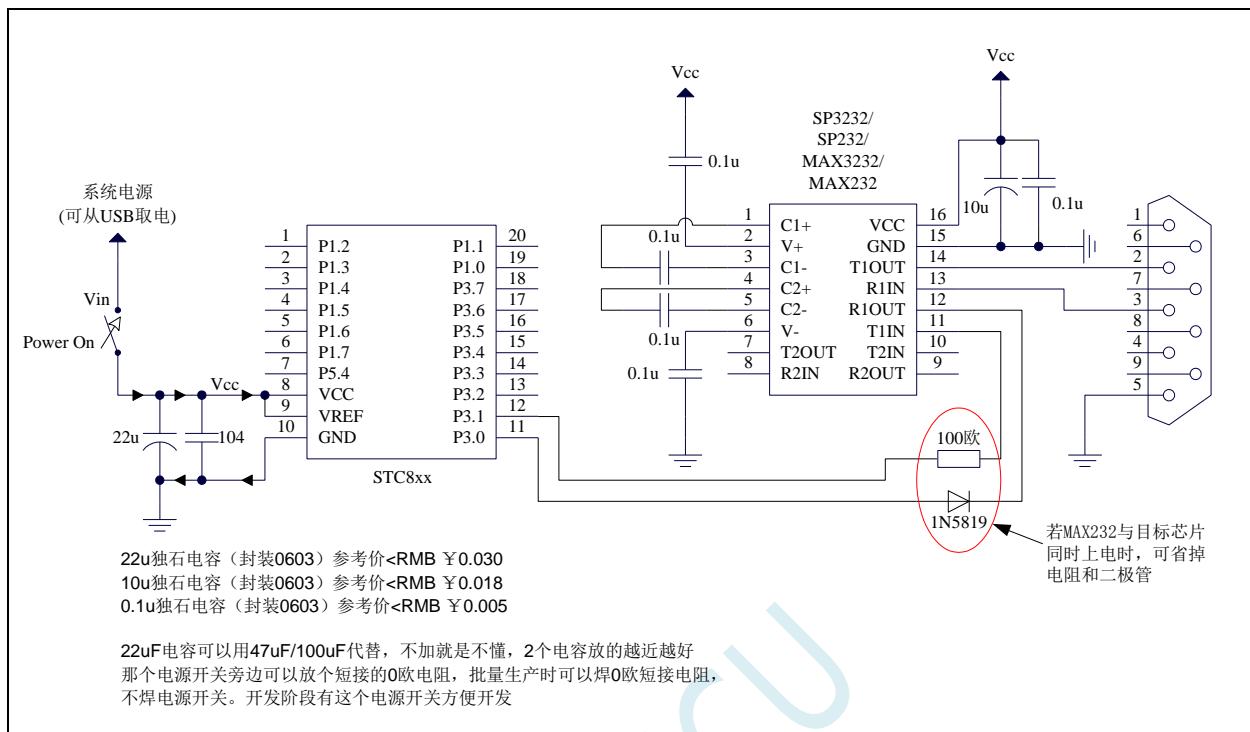
5.1.7 使用 RS-232 转换器下载 (无独立 VREF 脚), 也可支持仿真



ISP 下载步骤:

- 1、给目标芯片停电
- 2、点击 STC-ISP 下载软件中的“下载/编程”按钮
- 3、给目标芯片上电
- 4、开始 ISP 下载

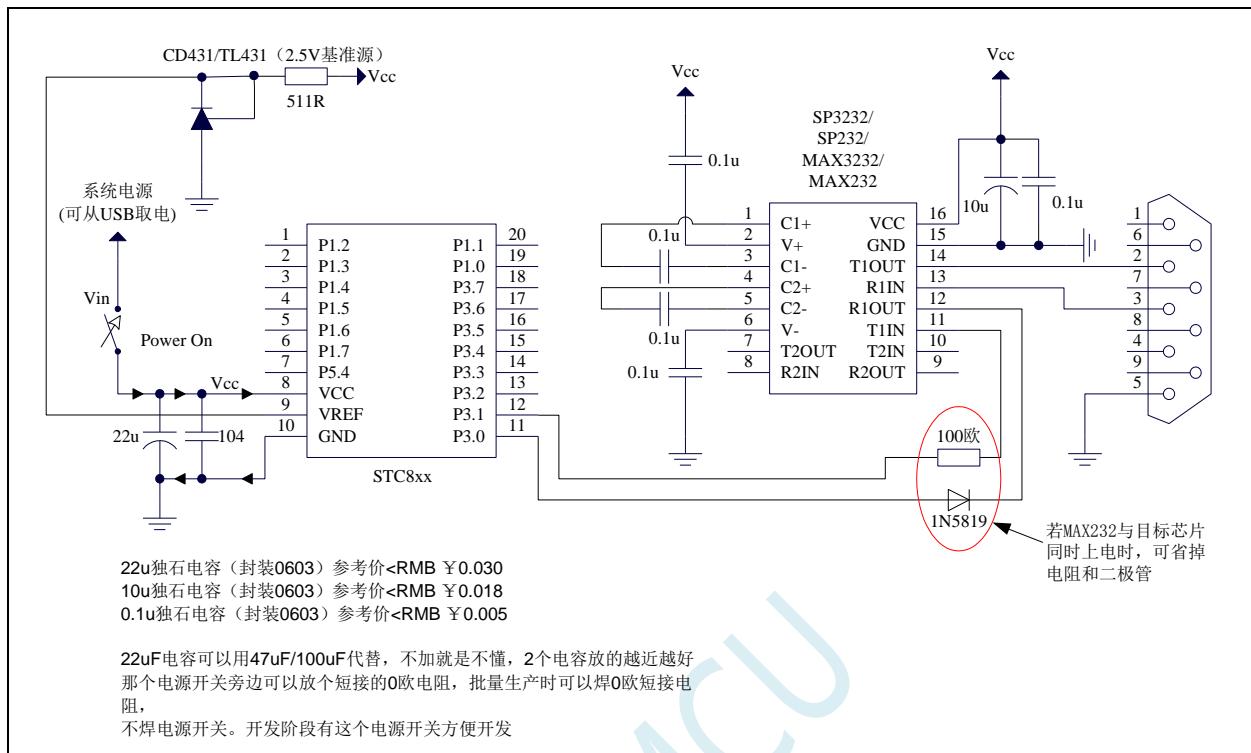
5.1.8 使用 RS-232 转换器下载 (有独立 VREF 脚、一般精度 ADC), 也可支持仿真



ISP 下载步骤:

- 1、给目标芯片停电
- 2、点击 STC-ISP 下载软件中的“下载/编程”按钮
- 3、给目标芯片上电
- 4、开始 ISP 下载

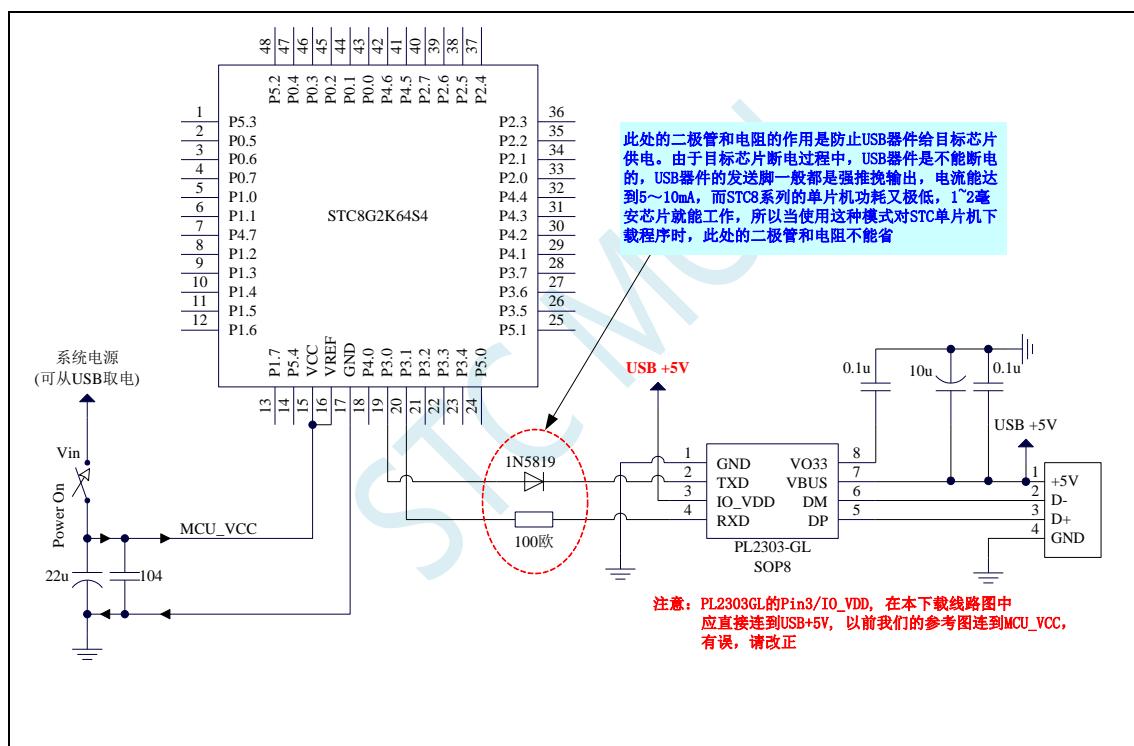
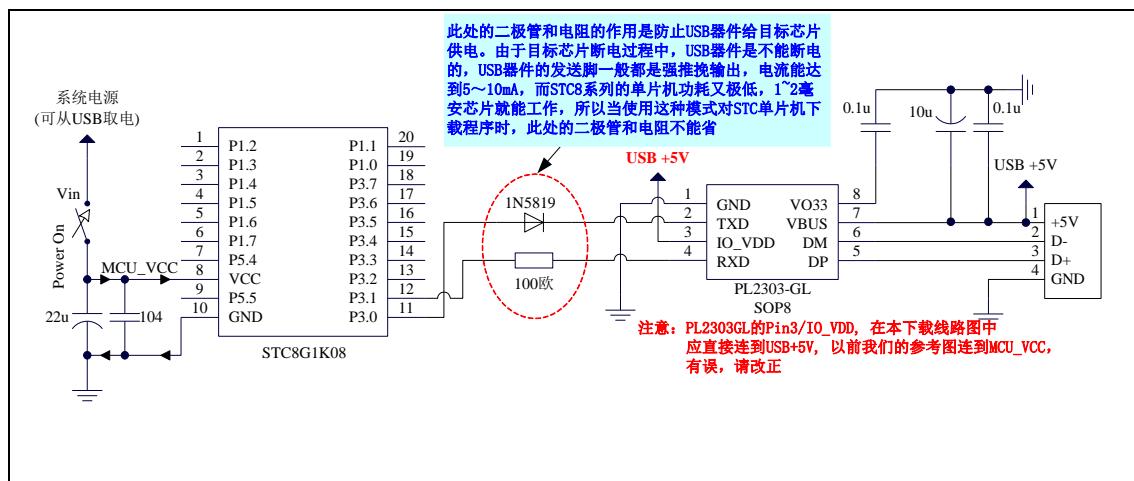
5.1.9 使用 RS-232 转换器下载 (有独立 VREF 脚、高精度 ADC), 也可支持仿真



ISP 下载步骤:

- 1、给目标芯片停电
- 2、点击 STC-ISP 下载软件中的“下载/编程”按钮
- 3、给目标芯片上电
- 4、开始 ISP 下载

5.1.10 使用 PL2303-GL 下载，也可支持仿真

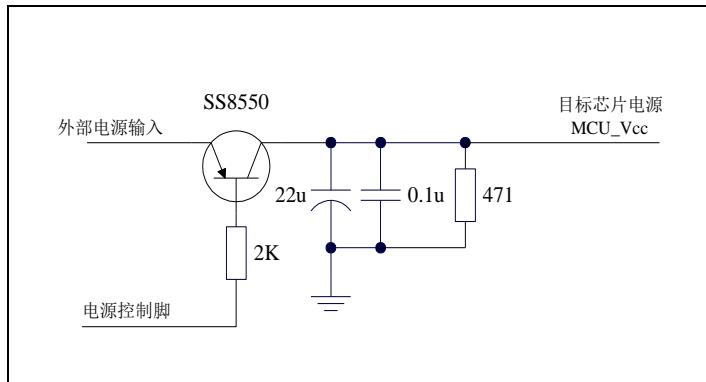


ISP 下载步骤:

- 1、给目标芯片停电，注意不能给 USB 转串口芯片停电（如：CH340、PL2303-GL 等）
注意：PL2303-SA 的部分波特率误差非常大，建议使用 PL2303-GL
- 2、由于 USB 转串口芯片的发送脚一般都是强推挽输出，必须在目标芯片的 P3.0 口和 USB 转串口芯片的发送脚之间串接一个二极管，否则目标芯片无法完全断电，达不到给目标芯片停电的目标。
- 3、点击 STC-ISP 下载软件中的“下载/编程”按钮
- 4、给目标芯片上电
- 5、开始 ISP 下载

注意：目前有发现使用 USB 线供电进行 ISP 下载时，由于 USB 线太细，在 USB 线上的压降过大，导致 ISP 下载时供电不足，所以请在使用 USB 线供电进行 ISP 下载时，务必使用 USB 加强线。

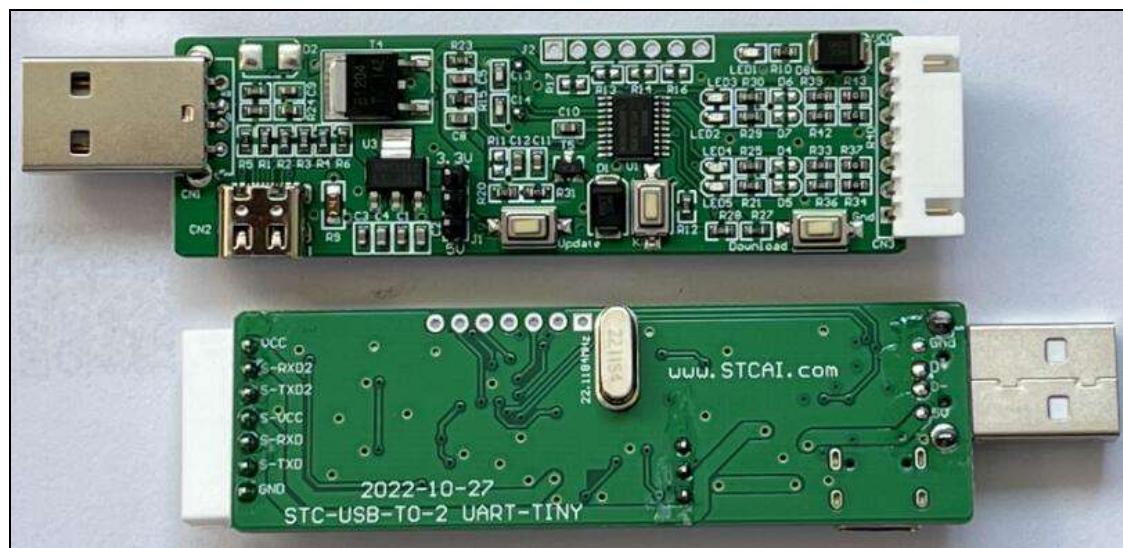
5.1.11 单片机电源控制参考电路



5.2 用 STC 一箭双雕之 USB 转双串口仿真 STC8 系列 MCU

先简单介绍下一箭双雕之 USB 转双串口工具

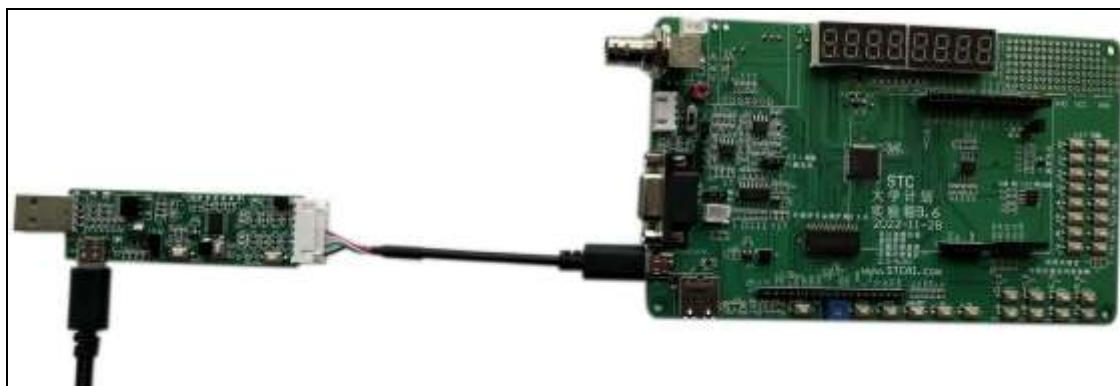
1、一箭双雕之 USB 转双串口工具外观图:



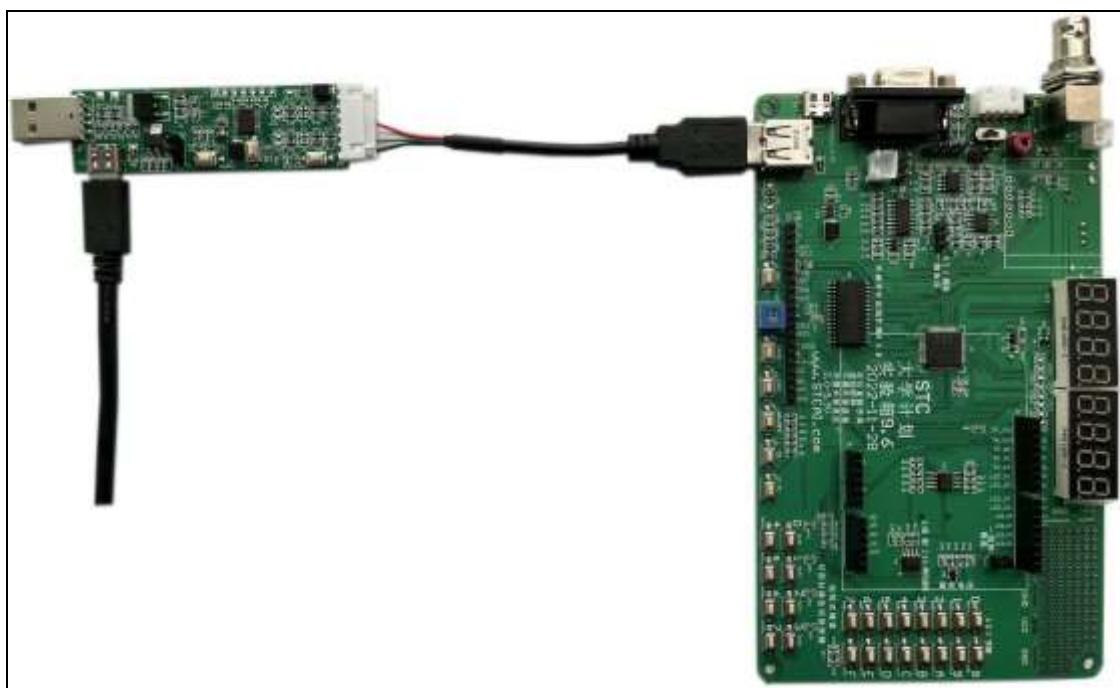
2、一箭双雕之 USB 转双串口工具几种常用连接线:



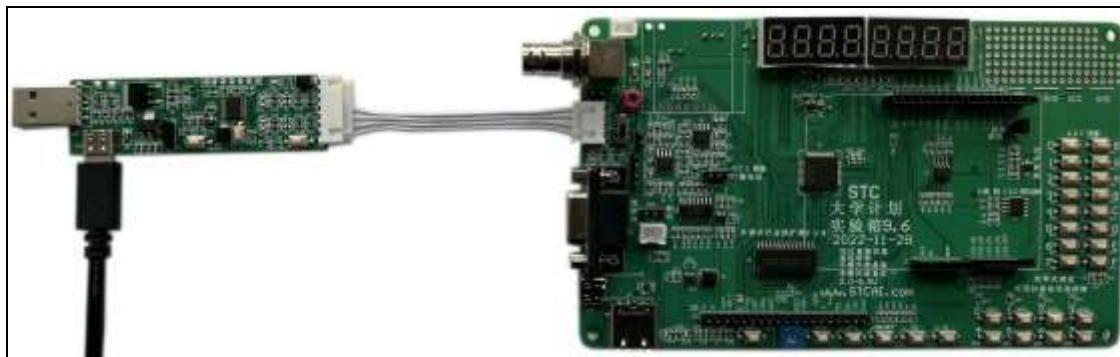
3、一箭双雕用 SIP7-USB-TypeC 对 STC8 系列 MCU 进行仿真/烧录，硬件连接图如下：



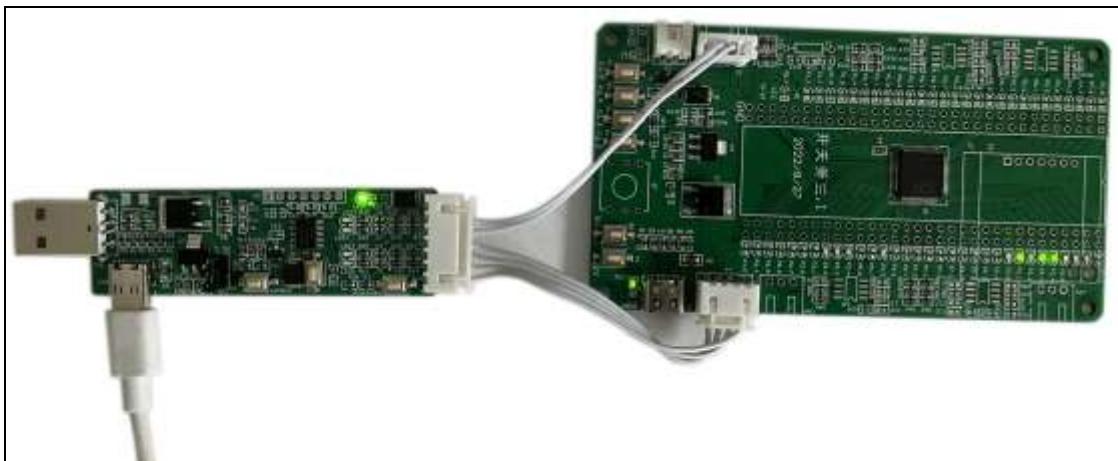
4、一箭双雕用 SIP7-USB-TypeA 对 STC8 系列 MCU 进行仿真/烧录，硬件连接图如下：



5、一箭双雕用 SIP7-SIP4/2.54mm 普通插座对 STC8 系列 MCU 进行仿真/烧录，硬件连接图如下：



6、一箭双雕，USB 扩展的 USB-CDC 串口 1 仿真；扩展的 USB-CDC 串口 2 与其他串口进行通信，硬件连接图如下：



7、将一箭双雕设置成普通的下载工具，可以参考这个官网论坛的这个帖子：

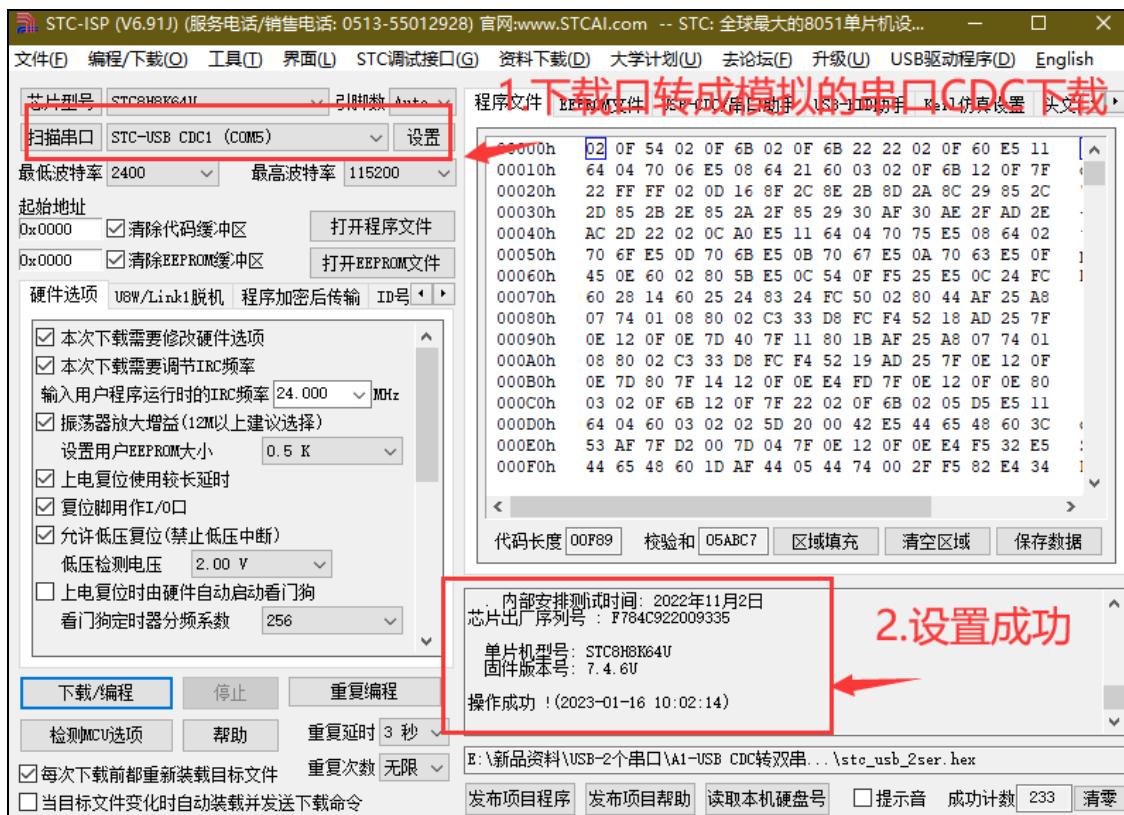
<https://www.stcaimcu.com/forum.php?mod=viewthread&tid=240&highlight=%E4%B8%80%E7%AE%AD%E5%8F%8C%E9%9B%95>

拿到 USB 转双串口工具后可对其烧录不同的固件来实现不同的功能，例如做串口工具、做烧录工具、做 OLED 示波器等等。固件烧录流程如下：

- 1) 使用 USB-TypeC 数据线或者通过 USB-TypeA 接口连接核心板到电脑；
- 2) 按住 P3.2 口按键不放；
- 3) 按一下电源开关按键（按下-松开），然后可松开 P3.2 口按键；

正常情况下在 STC-ISP 软件上就可以识别出“STC USB Writer (HID1)”设备：





软件设置如下：

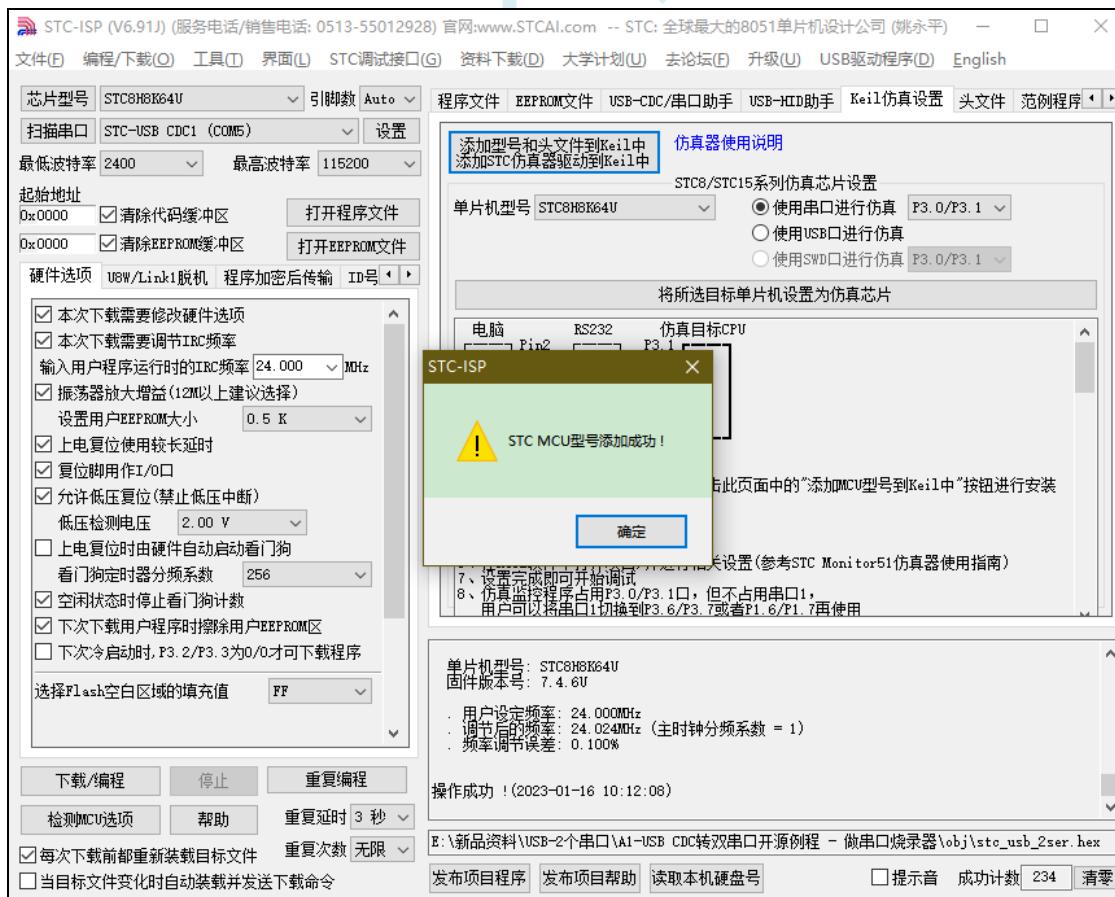
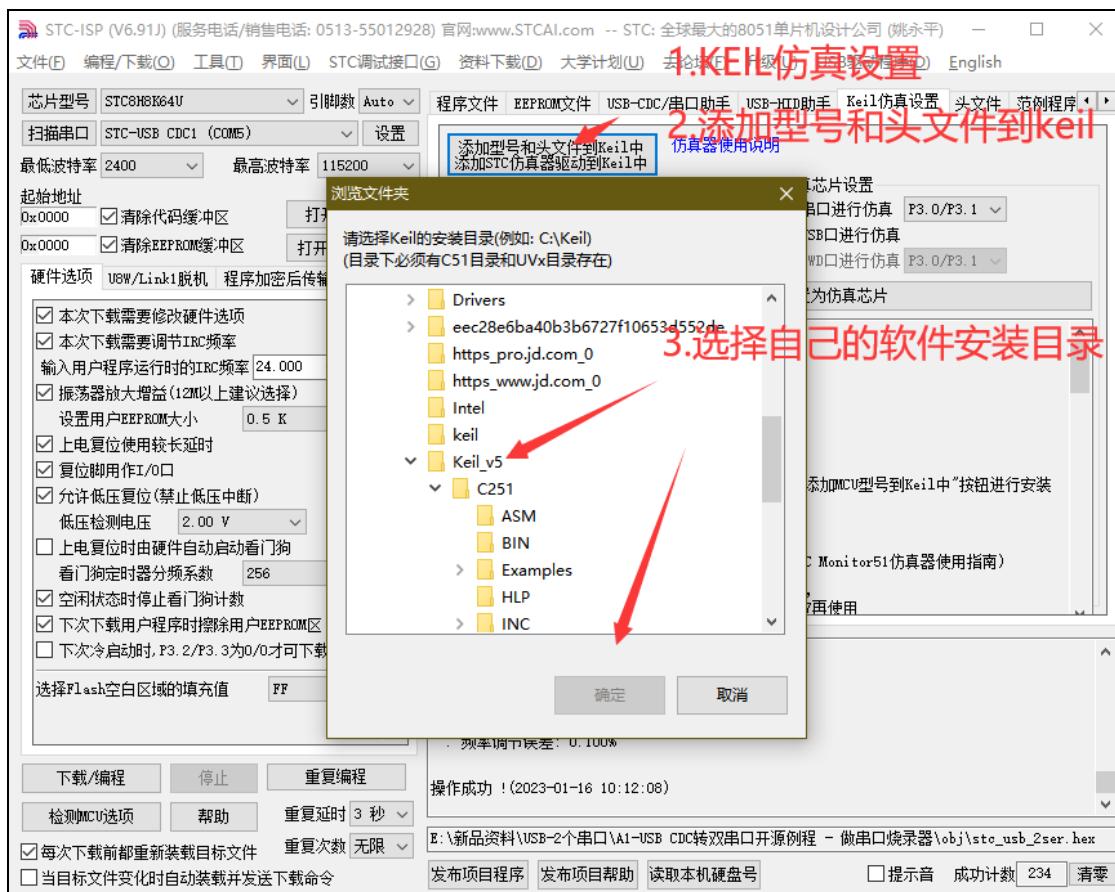
8、先去官网下载最新的 STC-ISP 软件，截止至目前最新版本是 STC-ISP (6.91J)，特别是仿真这块，STC-ISP (6.91J) 的 stcmmon51 仿真驱动程序版本已更新至 v1.18，经内部反复测试已经非常稳定。

(下载地址：工具软件-深圳国芯人工智能有限公司 <https://www.stcai.com/gjri>)



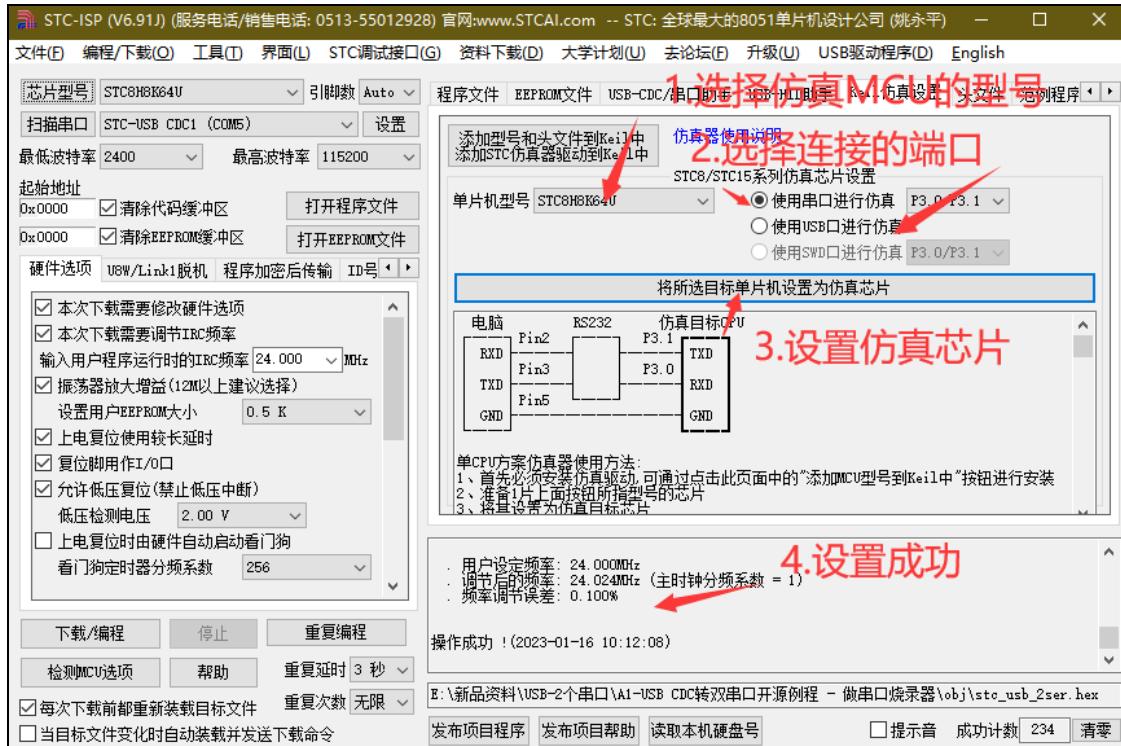
9、更新 KEIL 中的 STC 的资源：添加 STC 仿真器的固件和芯片型号到 KEIL 中。

(此步骤建议在每次 ISP 下载软件更新时都重新添加一次，保证仿真驱动都是最新的版本)

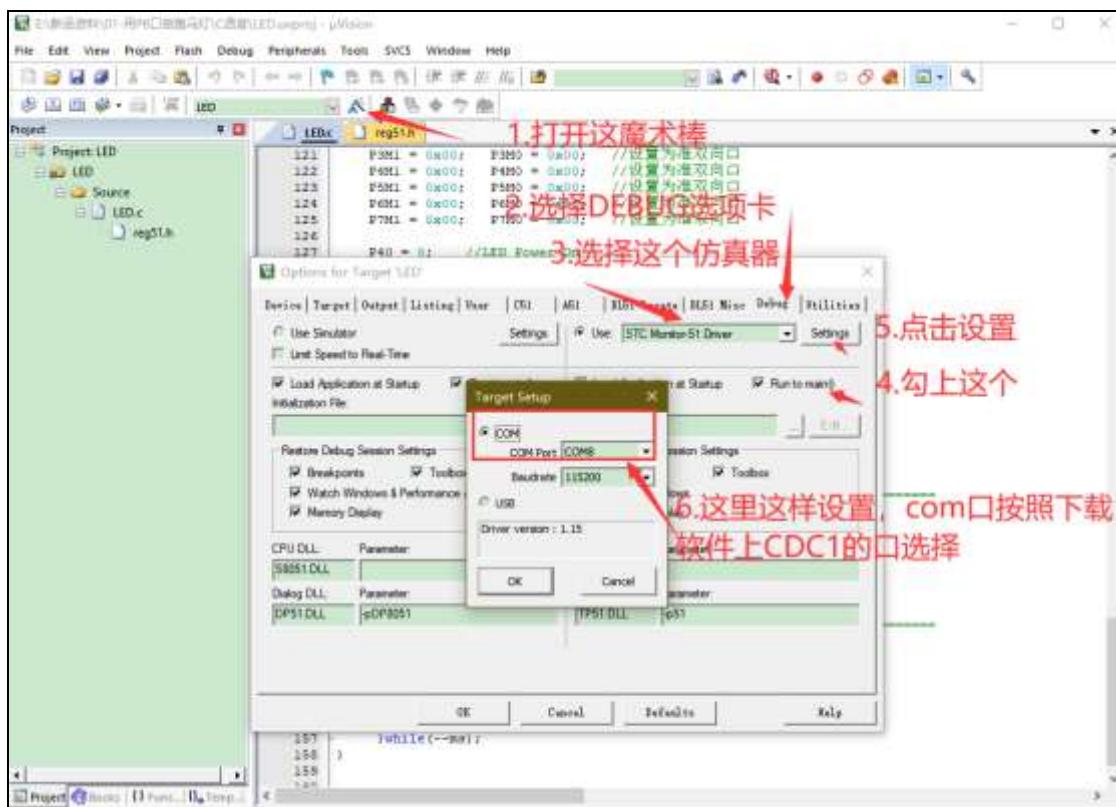


10、现在开始进行 KEIL 仿真的步骤，先将 STC8H8K64U 设置为仿真芯片，STC8H8K64U 目前仅支持串口和 USB 直接仿真。(这里选择了 P3.0/3.1 作为仿真端口，所以程序里不能出现任何占用 3.0 和 3.1 引脚的功能，仿真注意事项贴中也会说明，**先用点亮一个 LED 的程序进行测试，比较容易观察结果！**)
此时连接 STC8 的芯片，然后进行如下的设置将开天斧设置成仿真的主控芯片。

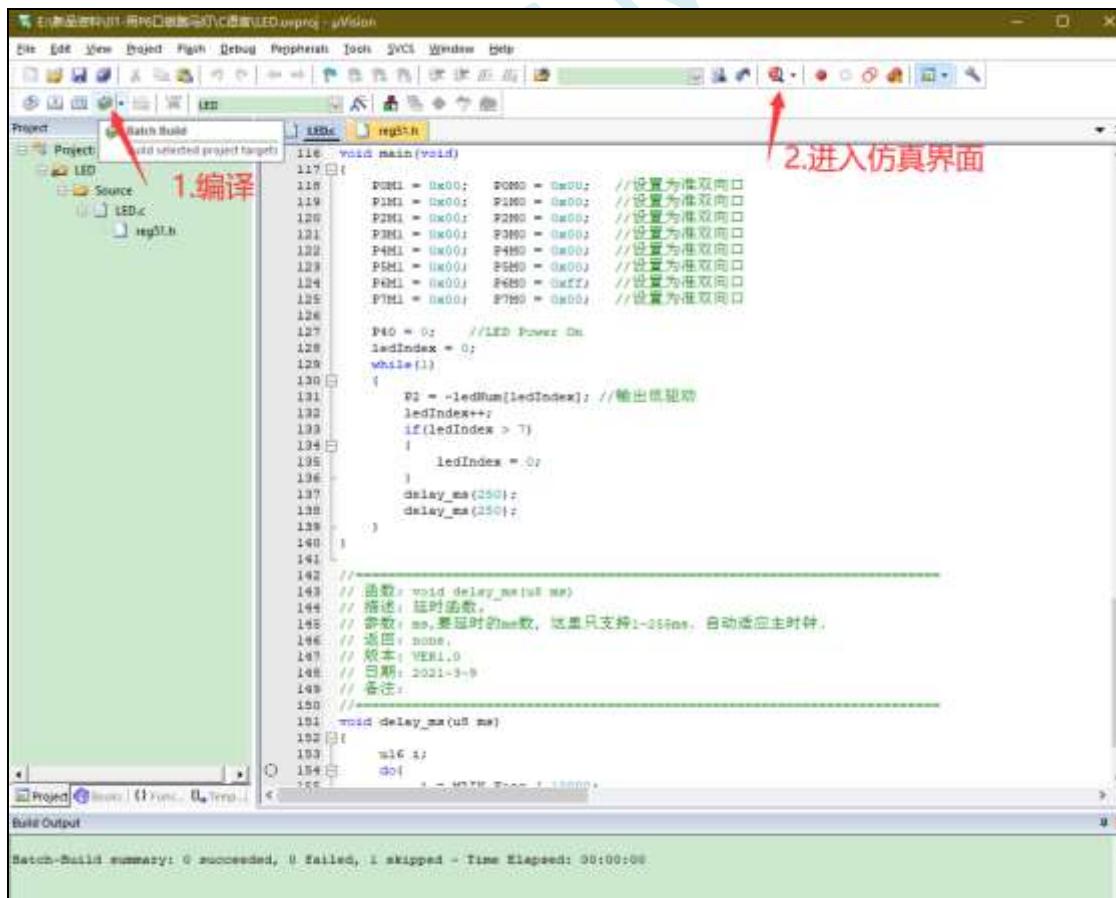
(注意一下这里的 IRC 频率一定要和程序里设置的主时钟一样!!)



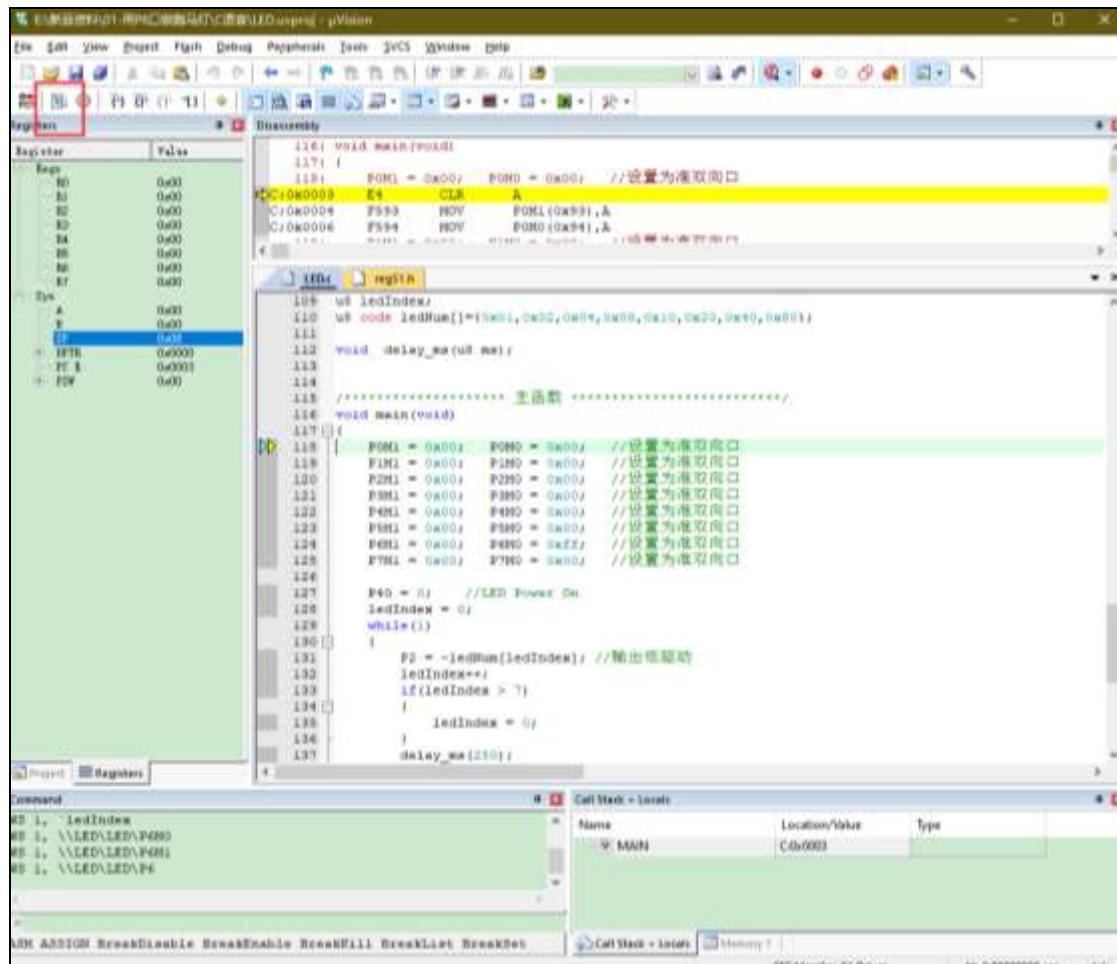
11、开天斧设置为仿真芯片（如何在 keil 中创建工程不在此贴详述），打开 KEIL 中建好的工程项目，直接去 KEIL 中设置仿真的路径。



12、这样就可以编译并且调试了。



13、出现下面这个界面，说明已经成功的进入了仿真模式，然后就可以用变量监测，断点等等的功能。



5.3 STC-ISP 下载软件高级应用

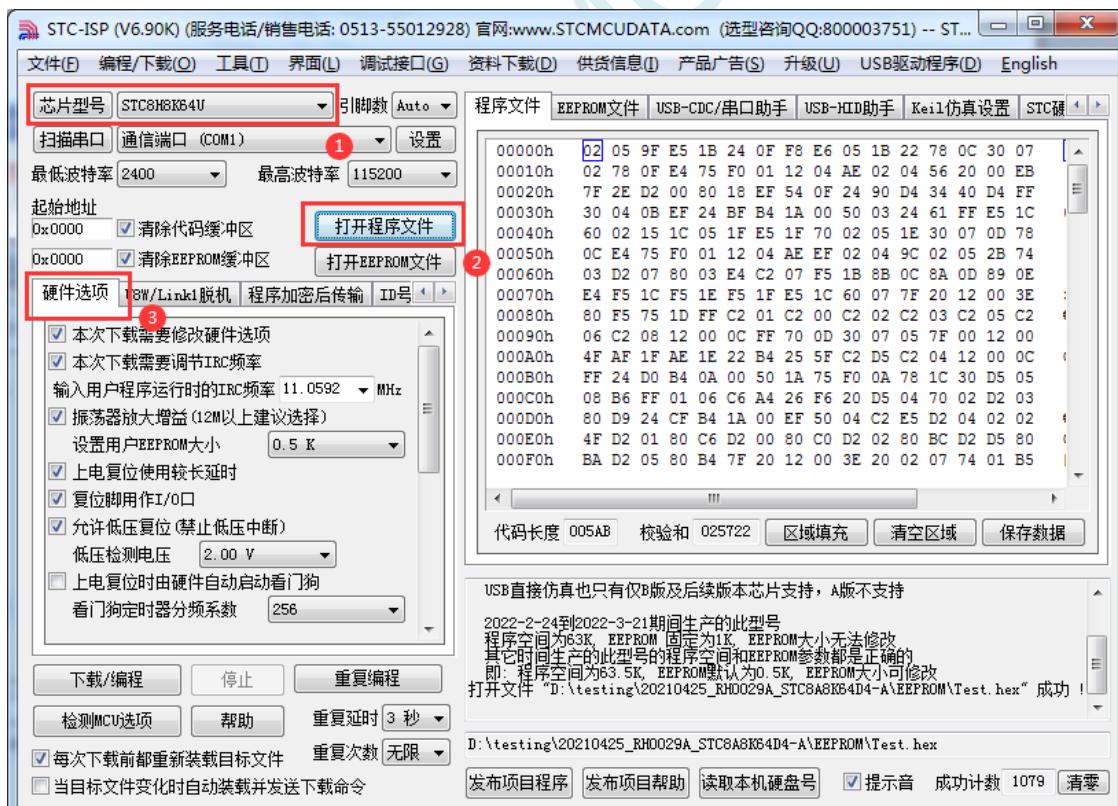
5.3.1 发布项目程序

发布项目程序功能主要是将用户的程序代码与相关的选项设置打包成为一个可以直接对目标芯片进行下载编程的超级简单的用户自己界面的可执行文件。

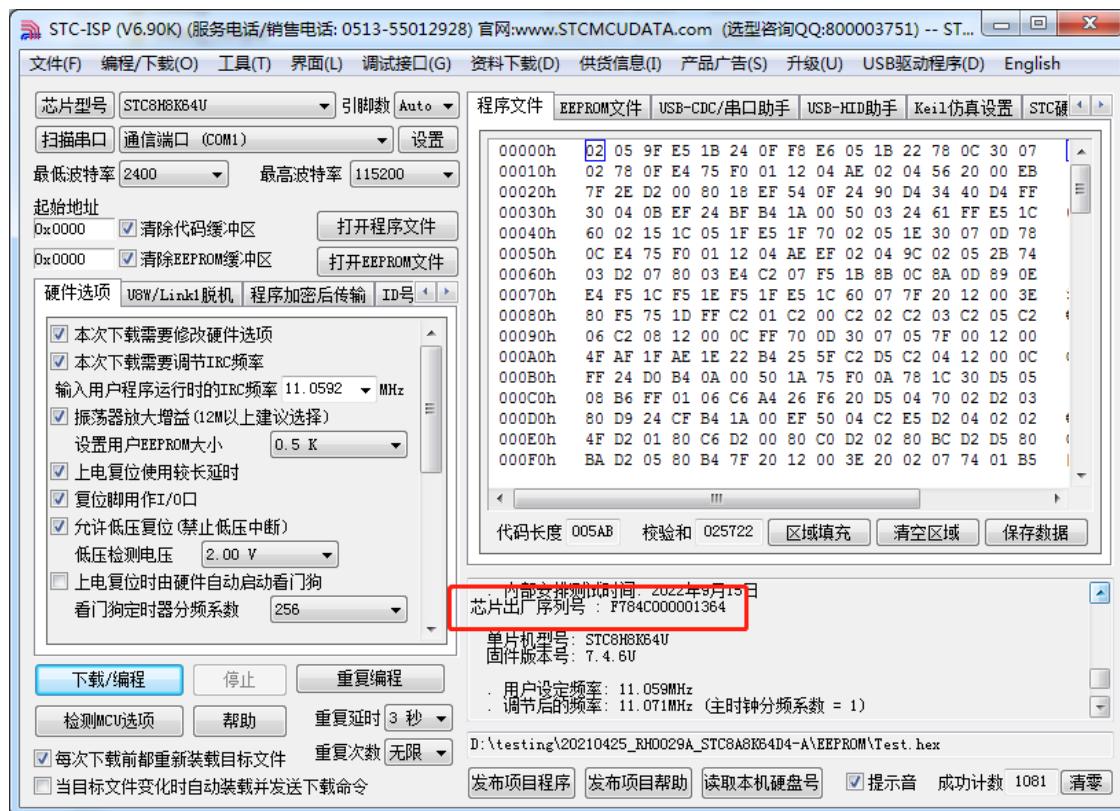
关于界面，用户可以自己进行定制（用户可以自行修改发布项目程序的标题、按钮名称以及帮助信息），同时用户还可以指定目标电脑的硬盘号和目标芯片的 ID 号，指定目标电脑的硬盘号后，便可以控制发布应用程序只能在指定的电脑上运行（防止烧录人员将程序轻易从电脑盗走，如通过网络发走，如通过 U 盘拷走，防不胜防，当然盗走你的电脑那就没办法了，所以 STC 的脱机下载工具比电脑烧录安全，能限制可烧录芯片数量，让前台文员小姐烧，让老板娘烧都可以），拷贝到其它电脑，应用程序不能运行。同样的，当指定了目标芯片的 ID 号后，那么用户代码只能下载到具有相应 ID 号的目标芯片中（对于一台设备要卖几千万的产品特别有用---坦克，可以发给客户自己升级，不需冒着生命危险跑到战火纷飞的伊拉克升级软件啦），对于 ID 号不一致的其它芯片，不能进行下载编程。

发布项目程序详细的操作步骤如下：

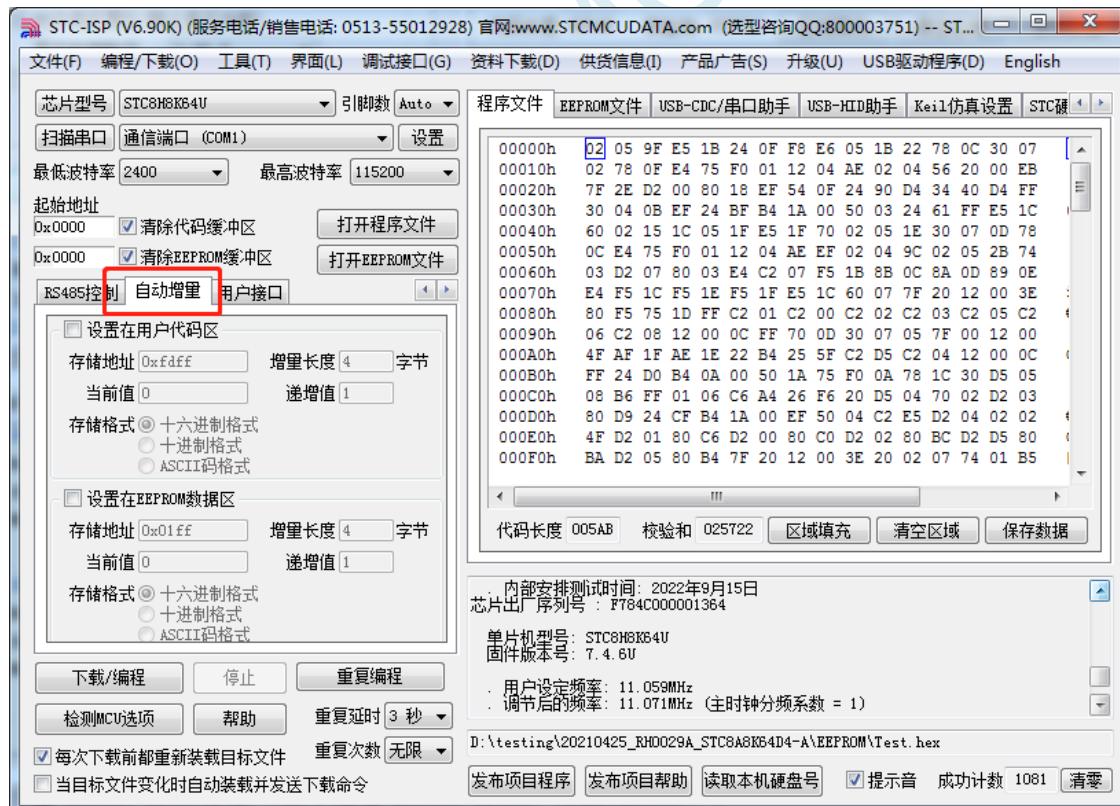
- 1、首先选择目标芯片的型号
- 2、打开程序代码文件
- 3、设置好相应的硬件选项



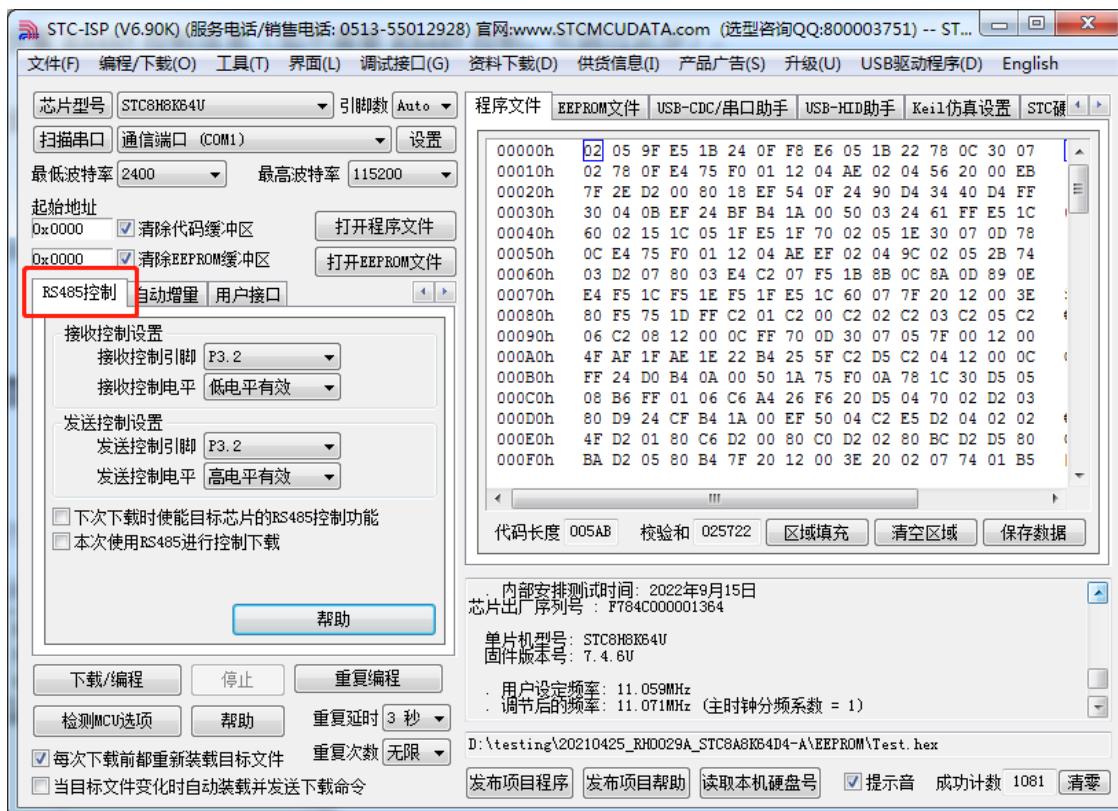
- 4、试烧一下芯片，并记下目标芯片的 ID 号，如下图所示，该芯片的 ID 号即为“F784C000001364”（如不需要对目标芯片的 ID 号进行校验，可跳过此步）



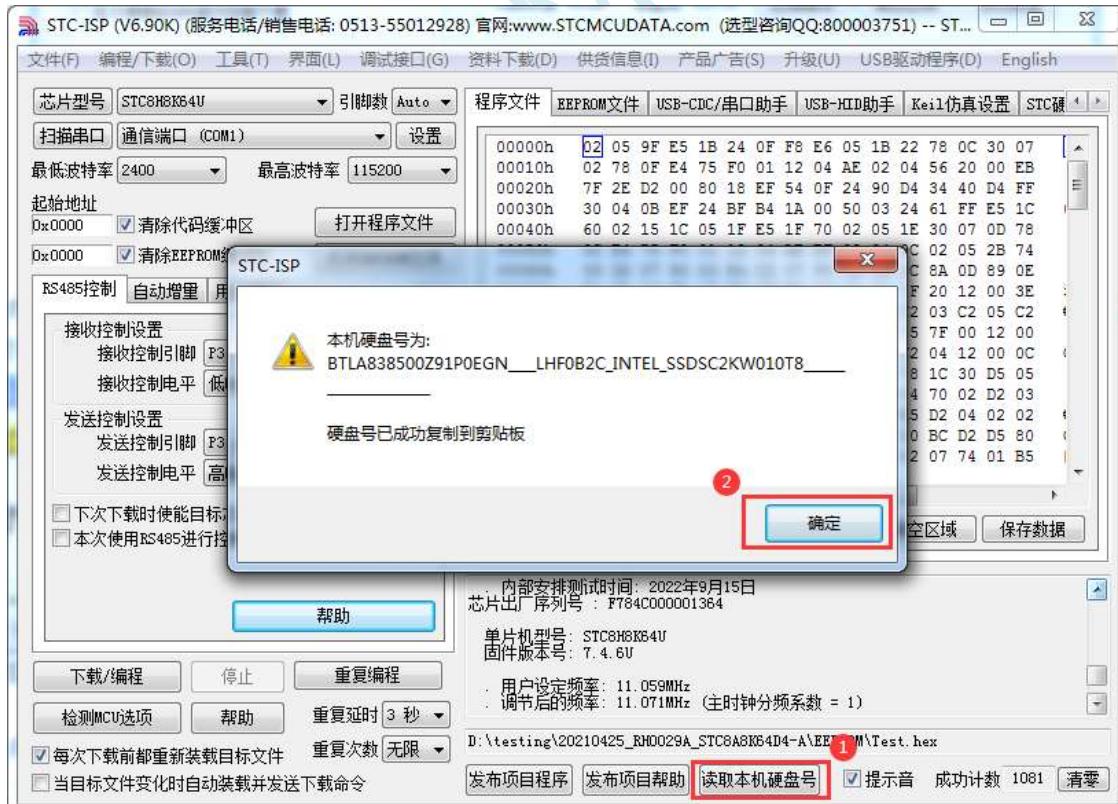
5、设置自动增量（如不需要自动增量，可跳过此步）



6、设置 RS485 控制信息（如不需要 RS485 控制，可跳过此步）



7、点击界面上的“读取本机硬盘号”按钮，并记下目标电脑的硬盘号（如不需要对目标电脑的硬盘号进行校验，可跳过此步）



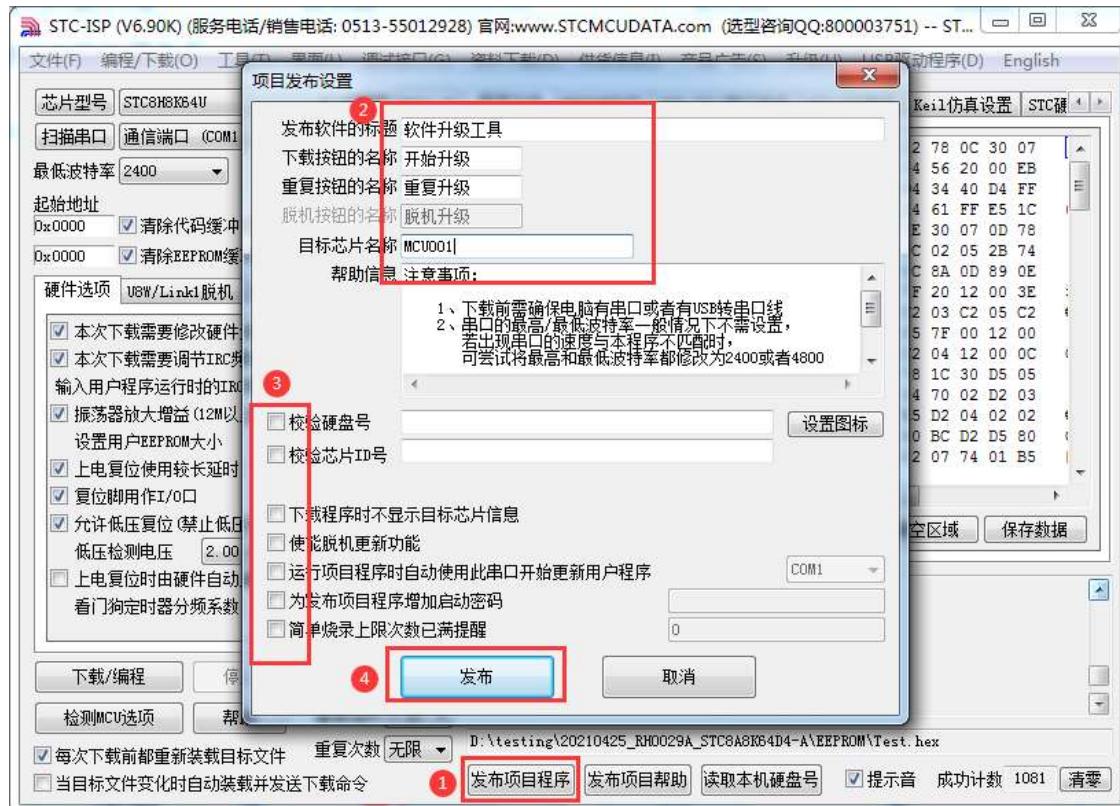
8、点击“发布项目程序”按钮，进入发布应用程序的设置界面。

9、根据各自的需要，修改发布软件的标题、下载按钮的名称、重复下载按钮的名称、自动增量的名称以及帮助信息

10、若需要校验目标电脑的硬盘号，则需要勾选上“校验硬盘号”，并在后面的文本框内输入前面所记

下的目标电脑的硬盘号

- 11、若需要校验目标芯片的 ID 号，则需要勾选上“校验芯片 ID 号”，并在后面的文本框内输入前面所记下的目标芯片的 ID 号



- 12、最后点击发布按钮，将项目发布程序保存，即可得到相应的可执行文件。发布的项目程序界面如下图



5.3.2 程序加密后传输（防烧录时串口分析出程序）

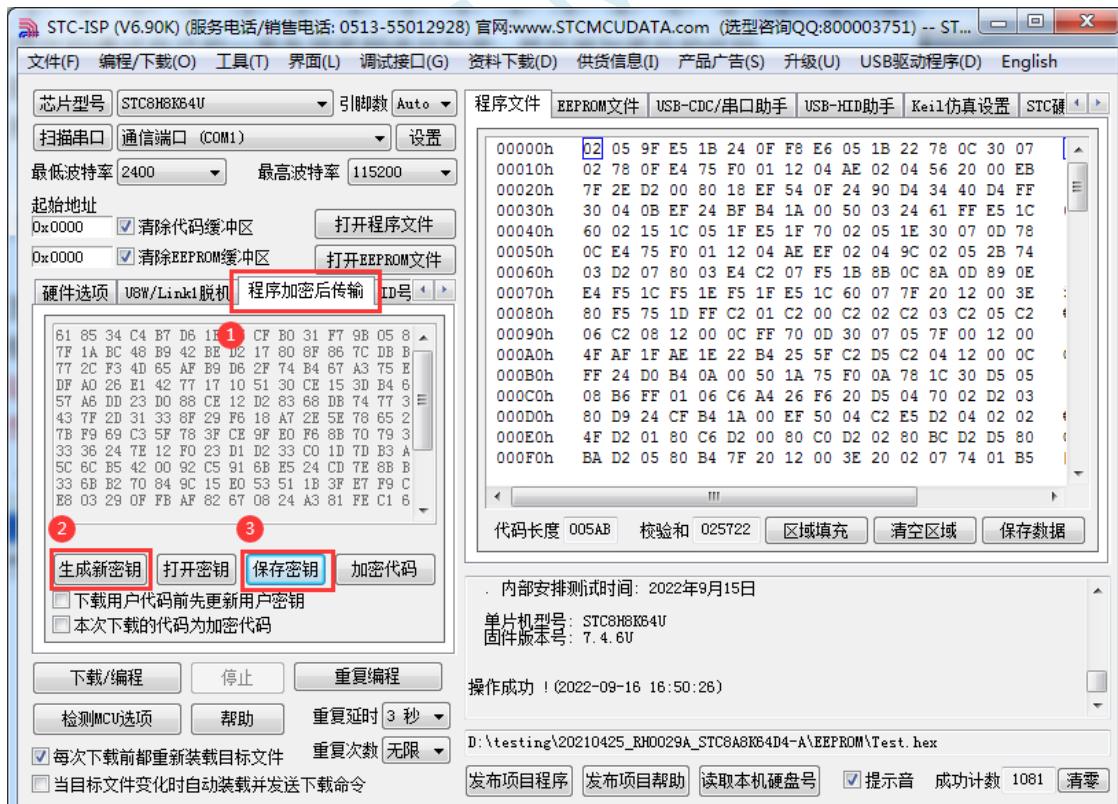
目前，所有的普通串口下载烧录编程都是采用明码通信的（电脑和目标芯片通信时，或脱机下载板和目标芯片通信时），问题：如果烧录人员通过分析下载烧录编程时串口通信的数据，高手是可以在烧录时在串口上引 2 根线出来，通过分析串口通信的数据分析出实际的用户程序代码的。**当然用 STC 的脱机下载板烧程序总比用电脑烧程序强（防止烧录人员将程序轻易从电脑盗走，如通过网络发走，如通过 U 盘拷走，防不胜防，当然盗走你的电脑那就没办法那，所以 STC 的脱机下载工具比电脑烧录安全，让前台文员小姐烧，让老板娘烧都可以）。**即使是 STC 全球首创的脱机下载工具，对于要防止天才的不法分子在脱机下载工具烧录的过程中通过分析串口通信的数据分析出实际的用户程序代码，也是没有办法达到要求的，这就需要用到最新的 STC 单片机所提供的程序加密后传输功能。

程序加密后传输下载是用户先将程序代码通过自己的一套专用密钥进行加密，然后将加密后的代码再通过串口下载，此时下载传输的是加密文件，通过串口分析出来的是加密后的乱码，如不通过派人潜入你公司盗窃你电脑里面的加密密钥，就无任何价值，便可起到防止在烧录程序时被烧录人员通过监测串口分析出代码的目的。

程序加密后传输功能的使用需要如下的几个步骤：

1、生成并保存新的密钥

如下图，进入到“程序加密后传输”页面，点击“生成新密钥”按钮，即可在缓冲区显示新生成的 256 字节的密钥。然后点击“保存密钥”按钮，即可将生成的新密钥保存为以“.K”为扩展名的密钥文件（**注意：这个密钥文件一定要保存好，以后发布的代码文件都需要使用这个密钥加密，而且这个密钥的生成是非重复的，即任何时候都不可能生成两个完全相同的密钥，所以一旦密钥文件丢失将无法重新获得**）。例如我们将密钥保存为“New.k”。

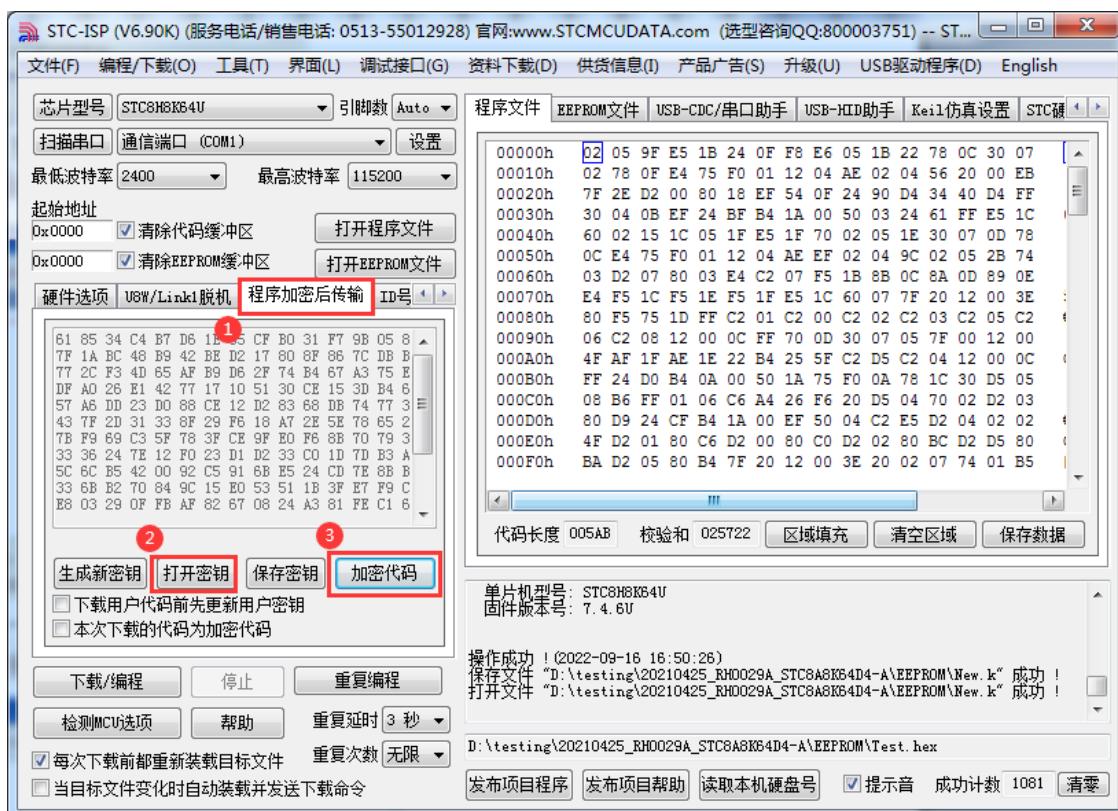


2、对代码文件加密

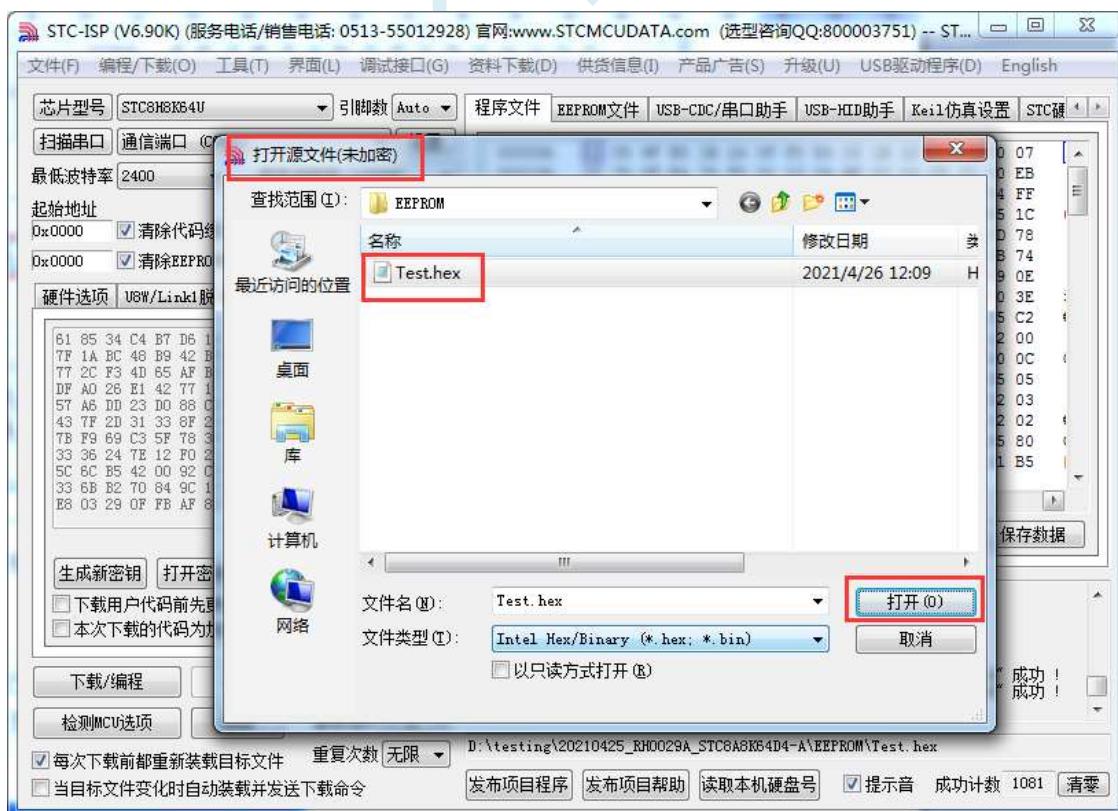
加密文件前，需要先打开我们自己的密钥。若缓冲区中存放的已经是我们的密钥，则不要再打开。

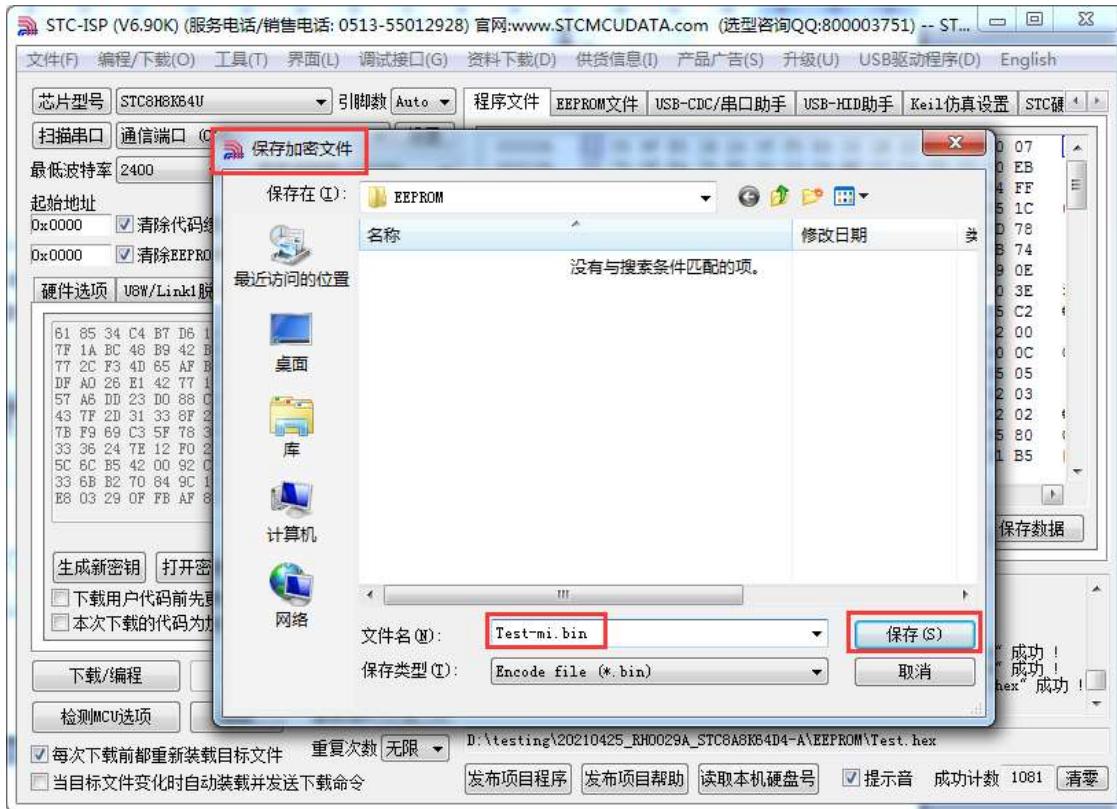
如下图，在“程序加密后传输”页面中点击“打开密钥”按钮，打开我们之前保存的密钥文件，

例如“New.k”，然后返回到“程序加密后传输”页面中点击“加密代码”按钮，如下图所示，首先会弹出“打开源文件(未加密)”的对话框，此时选择的是原始的未加密的代码文件



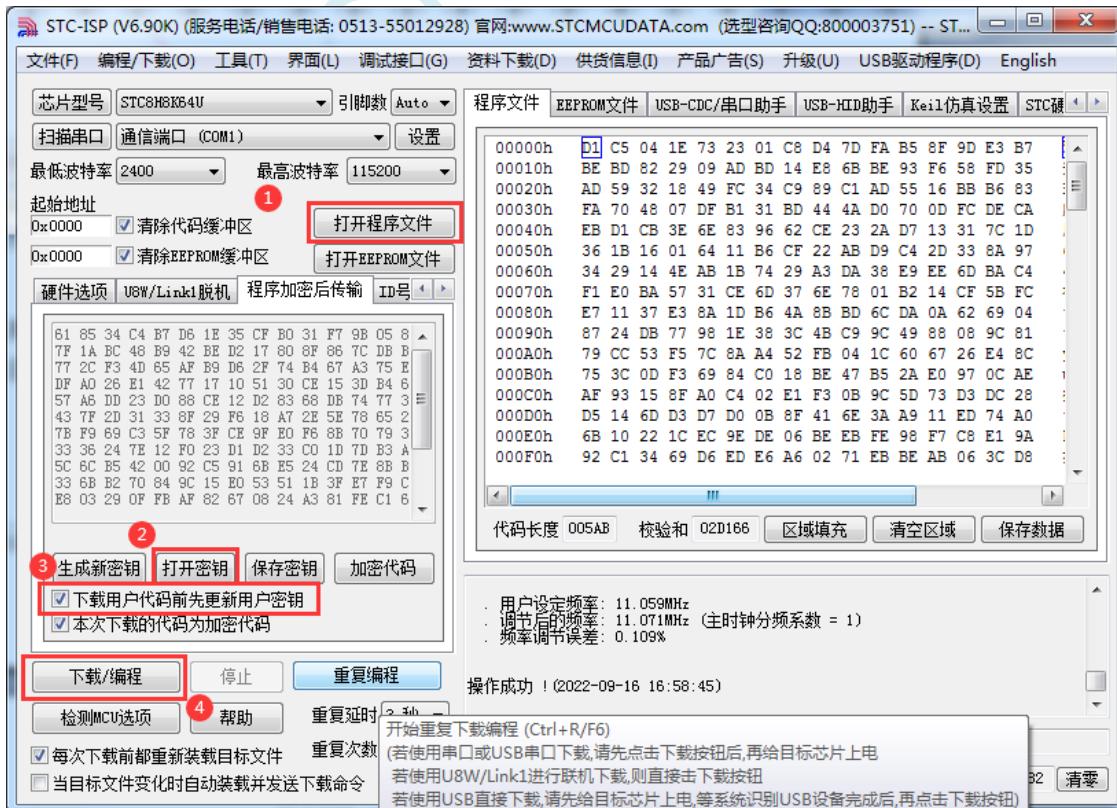
点击打开按钮后，马上会有弹出一个类似的对话框，但此时是对加密后的文件进行保存的对话框。如下图所示，点击保存按钮即可保存加密后的文件。





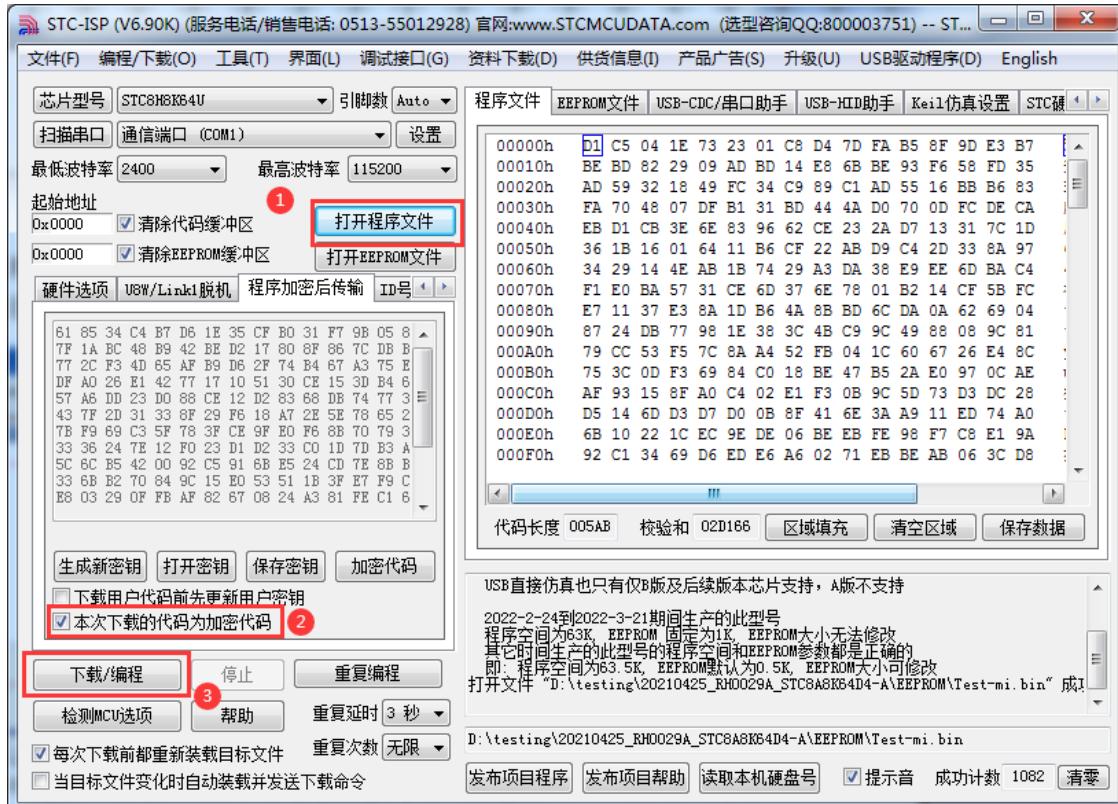
3、将用户密钥更新到目标芯片中

更新密钥前，需要先打开我们自己的密钥。若缓冲区中存放的已经是我们的密钥，则不要再打开。如下图，在“自定义加密下载”页面中点击“打开密钥”按钮，打开我们之前保存的密钥文件，例如“New.k”。密钥打开后，如下图所示，勾选上“下载用户代码前先更新用户密钥”选项和“本次下载的代码为加密代码”的选项，然后打开我们之前加密过后的文件，打开后点击界面左下角的“下载/编程”按钮，按正常方式对目标芯片下载完成即可更新用户密钥。



4、加密更新用户代码

密钥更新成功后，目标芯片便具有接收加密代码并还原的功能。此时若需要再次升级/更新代码，则只需要参考第二步的方法，将目标代码进行加密，然后如下图



对于一片新的 STC 单片机，可将步骤 3 和步骤 4 合并完成，即将密钥更新到目标单片机的同时也可将加密后的代码一并下载到单片机中，若已经执行过步骤 3（即已经将密钥更新到目标芯片中了），则后续的代码更新就只需要按照步骤 4，只需要在“程序加密后传输”页面中选择“本次下载的代码为加密代码”的选项（“**下载用户代码前先更新用户密钥**”选项不需要选了），然后打开我们之前加过密后的文件，打开后点击界面左下角的“下载/编程”按钮，按正常方式对目标芯片下载即可完成用用户自己专用的加密文件更新用户代码的目的（防止在烧录程序时被烧录人员通过监测串口分析出代码的目的）。

5.3.3 发布项目程序+程序加密后传输结合使用

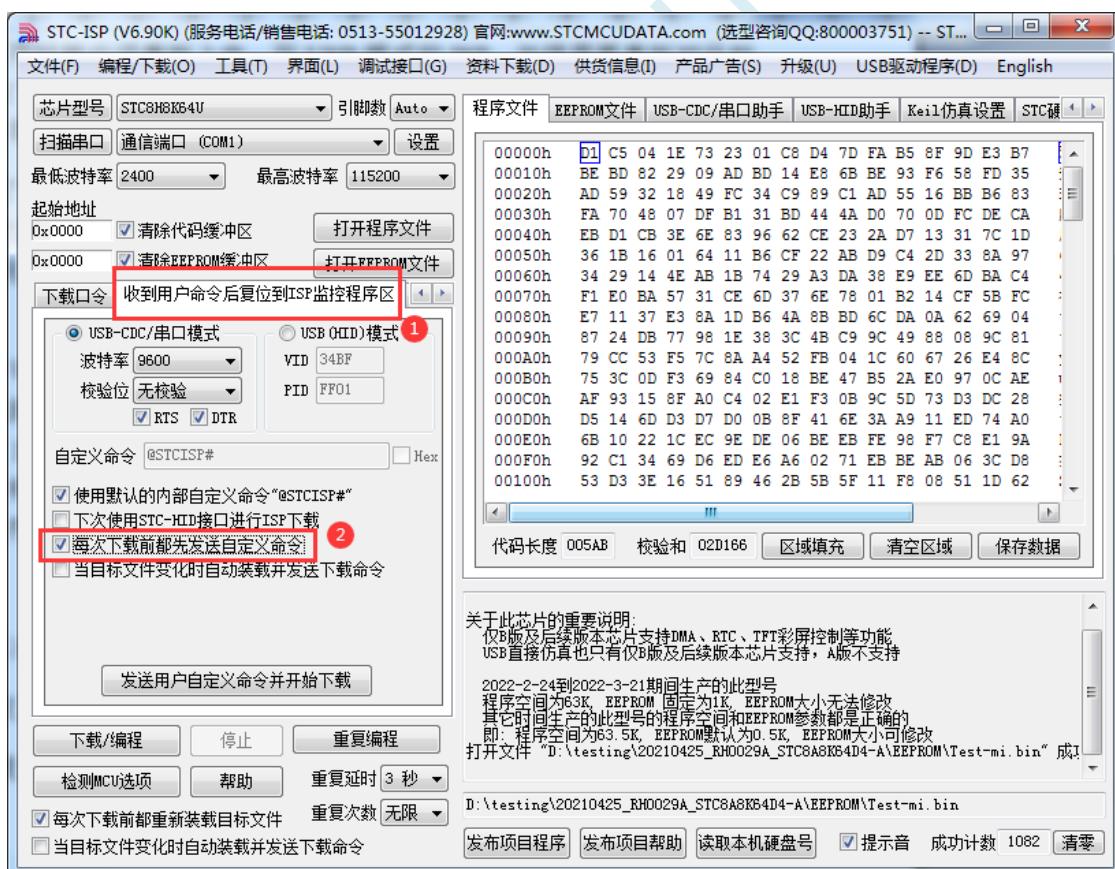
发布项目程序与程序加密后传输两项新的特殊功能可以结合在一起使用。首先程序加密后传输可以确保用户代码在烧录编程时串口通信传输过程当中的保密性，而发布项目程序可实现让最终使用者远程升级功能（方案公司的人员不需要亲自到场）。所以两项功能结合起来使用，非常适用于方案公司/生产商在软件需要更新时，让最终使用者自己对终端产品进行软件更新的目的，又确保现场烧录人员无法通过串口分析出有用程序，强烈建议方案公司使用。

5.3.4 用户自定义下载（实现不停电下载）

将用户的目标程序下载到 STC 单片机是通过执行单片机内部的 ISP 系统代码和上位机进行串口或者 USB 通讯来实现的。但 STC 单片机内部的 ISP 系统代码只有在每次重新停电再上电时才会被执行，这就要求用户每次需要对目标单片机更新程序时就必须重新上电，而 USB 模式的 ISP，处理需要重新对目标芯片上电外，还需要在上电时将 P3.2 口下拉到 GND。对于处于开发阶段的项目，需要频繁的修改代码、更新代码，每次下载都需要重新上电会导致操作非常麻烦。

STC 单片机在硬件设计时，增加了一个软复位寄存器 (IAP_CONTR)，让用户可以通过设置此寄存器来决定 CPU 复位后重新执行用户代码还是复位到 ISP 区执行 ISP 系统代码。当向 IAP_CONTR 寄存器写入 0x20 时，CPU 复位后重新执行用户代码；当向 IAP_CONTR 寄存器写入 0x60 时，CPU 复位后复位到 ISP 区执行 ISP 系统代码。

要实现不停电进行 ISP 下载，用户可以在程序中设计一段代码，例如检测一个特殊的按键、或者监控串口等待一个特殊的串口命令，当检测到满足下载条件时，就通过软件触发软复位寄存器复位到 ISP 区执行 ISP 系统代码，从而实现不停电 ISP 下载。当触发条件是外部按键时，则在用户代码中实时监控按键状态即可。若要实现 STC-ISP 软件和用户触发软复位完全同步，则需要使用 STC-ISP 软件中所提供的“收到用户命令后复位到 ISP 监控程序区”这个功能。



实现不停电 ISP 下载的步骤如下:

1、编写用户代码，并在用户代码中添加串口命令监控程序
(参考代码如下，测试单片机型号为 STC8H8K64U)

```
#include "stc8h.h"

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

char code *STCISPCMD = "@STCISP#";           //自定义下载命令
char index;

void uart_isr() interrupt 4
{
    char dat;

    if (TI)
    {
        TI = 0;
    }

    if (RI)
    {
        RI = 0;
        dat = SBUF;           //接收串口数据

        if (dat == STCISPCMD[index]) //判断接收的数据和当前的命令字符是否匹配
        {
            index++;           //若匹配则索引+1
            if (STCISPCMD[index] == '\0') //判断命令是否配完成
                IAP_CONTR = 0x60;       //若匹配完成则软复位到 ISP
        }
        else
        {
            index = 0;           //若不匹配,则需要从头开始
            if (dat == STCISPCMD[index])
                index++;
        }
    }
}

void main()
{
    P0M0 = 0x00; P0M1 = 0x00;
    P1M0 = 0x00; P1M1 = 0x00;
```

```

P2M0 = 0x00; P2M1 = 0x00;
P3M0 = 0x00; P3M1 = 0x00;

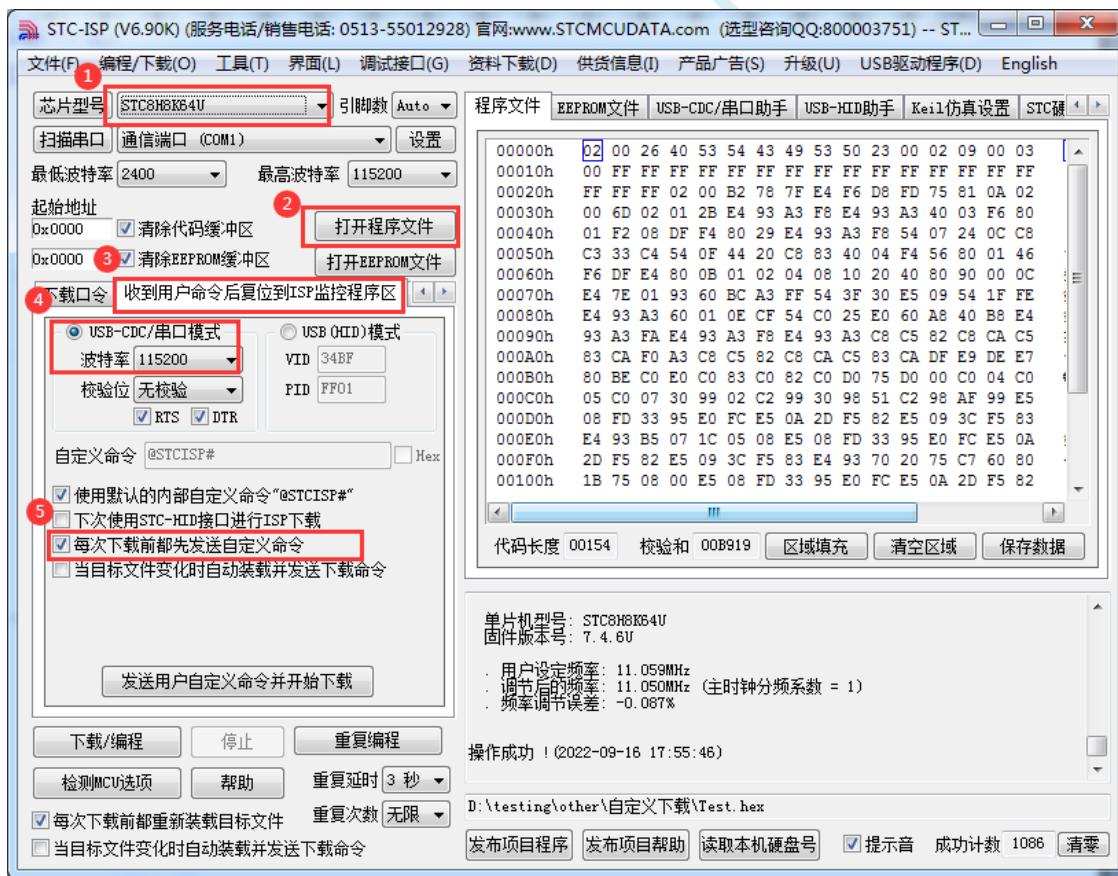
SCON = 0x50;           //串口初始化
AUXR = 0x40;
TMOD = 0x00;
TH1 = BAUD >> 8;
TL1 = BAUD;
TR1 = 1;
ES = 1;
EA = 1;

index = 0;             //初始化命令

while (1);
}

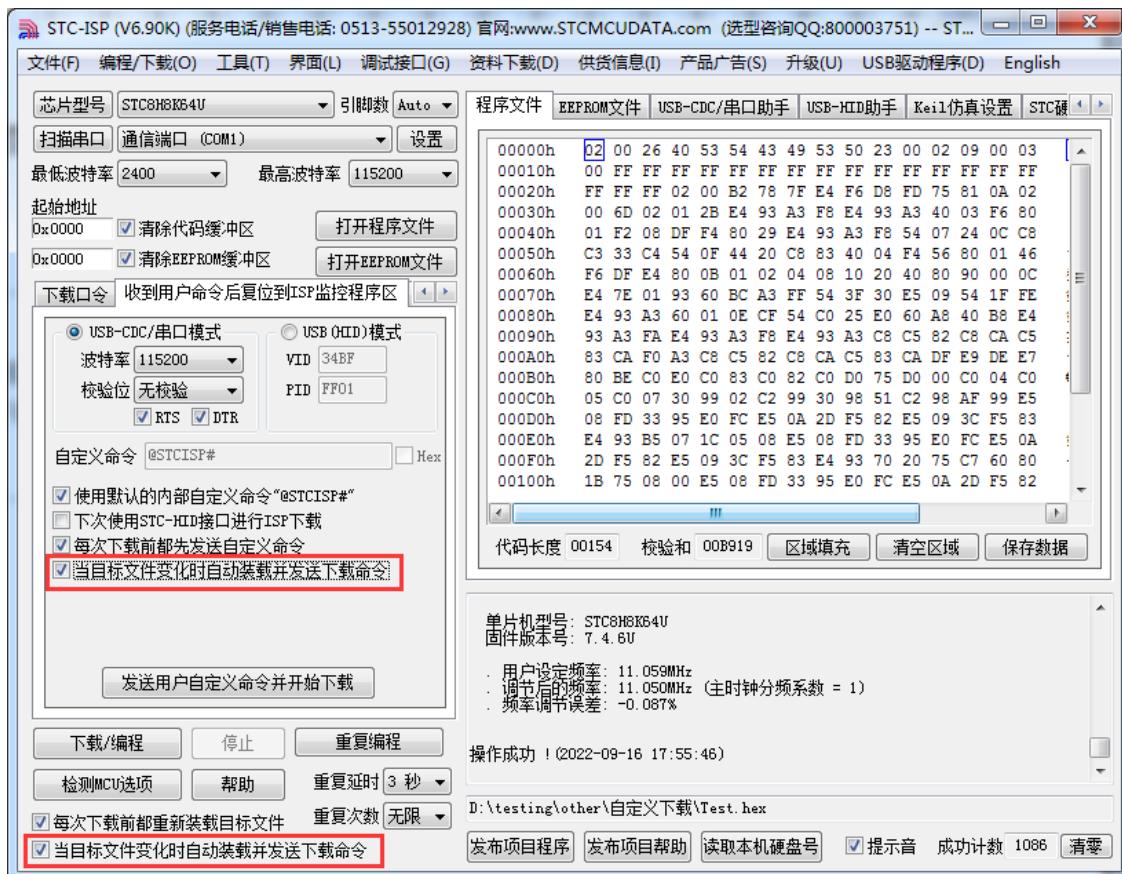
```

2、按下图所示的步骤进行设置自定义下载命令（范例使用 STC 默认命令“@STCISP#”）



3、第一次下载时需要对目标单片机重新上电，之后的每次更新只需要点击下载软件中的“下载/编程”按钮，下载软件自动将下载命令发送给目标单片机，目标单片机接收到命令后自动复位到系统 ISP 区，即可实现不停电更新用户代码。

4、STC-ISP 还可实现项目开发阶段，完全自动下载功能，即当下载软件侦测到目标代码被更新了，就会自动发送下载命令。要实现这个功能只需要勾选下图中的两个选项中的任意一个即可

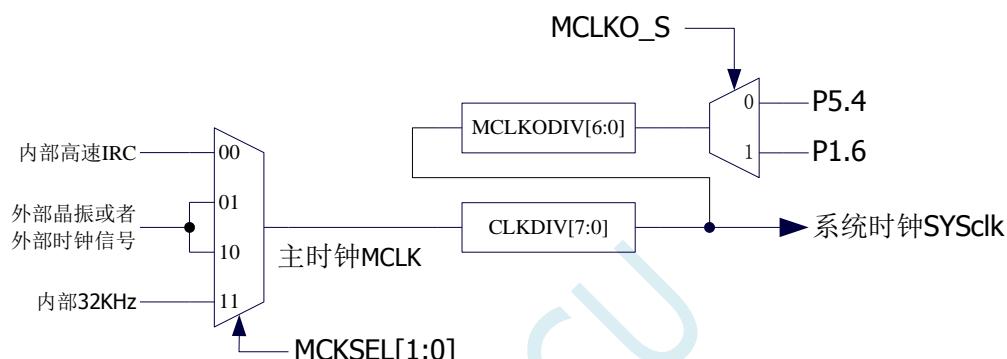


6 时钟、复位、省电模式与系统电源管理

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 IRC、内部 32KHz 的 IRC（误差较大）和外部晶振。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



系统时钟结构图

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	
CLKSEL	时钟选择寄存器	FE00H	-	-	-	-	-	-	-	MCKSEL[1:0]	xxxx,xx00
CLKDIV	时钟分频寄存器	FE01H	-	-	-	-	-	-	-	-	nnnn,nnnn
HIRCER	内部高速振荡器控制寄存器	FE02H	ENHIRC	-	-	-	-	-	-	HIRCST	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
MCLKOCR	主时钟输出控制寄存器	FE05H	MCLKO_S	-	MCLKODIV[6:0]	-	-	-	-	-	0000,0000

6.1.1 系统时钟选择寄存器 (CLKSEL)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CLKSEL	FE00H	-	-	-	-	-	-	-	MCKSEL[1:0]

MCKSEL[1:0]: 主时钟源选择

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

6.1.2 时钟分频寄存器 (CLKDIV)

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

注意: 用户程序复位后, 系统会自动根据上次 ISP 下载时所设定工作频率所需的分频系数来设置此寄存器的初始值

6.1.3 内部高速高精度 IRC 控制寄存器 (HIRCCR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
HIRCCR	FE02H	ENHIRC	-	-	-	-	-	-	HIRCST

ENHIRC: 内部高速高精度 IRC 使能位

- 0: 关闭内部高精度 IRC
- 1: 使能内部高精度 IRC

HIRCST: 内部高精度 IRC 频率稳定标志位。(只读位)

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

6.1.4 外部振荡器控制寄存器 (XOSCCR)

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) (**此时 P1.6 口固定为高阻输入模式, 可用于读取外部数字信号或当作 ADC 输入, 但一般不建议使用, 因为旁边的 P1.7 口有高频振荡信号会对 P1.6 的信号有影响**)
- 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。

6.1.5 内部 32KHz 低速 IRC 控制寄存器 (IRC32KCR)

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。

6.1.6 主时钟输出控制寄存器 (MCLKOCR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
MCLKOCR	FE05H	MCLKO_S	MCLKODIV[6:0]						

MCLKODIV[6:0]: 主时钟输出分频系数

(注意: 主时钟分频输出的时钟源是经过 CLKDIV 分频后的系统时钟)

MCLKODIV[6:0]	系统时钟分频输出频率
0000000	不输出时钟
0000001	SYSClk/1
0000010	SYSClk /2
0000011	SYSClk /3
...	...
1111110	SYSClk /126
1111111	SYSClk /127

MCLKO_S: 系统时钟输出管脚选择

0: 系统时钟分频输出到 P5.4 口

1: 系统时钟分频输出到 P1.6 口

6.2 STC8G 系列内部 IRC 频率调整

STC8G 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-1.35%~1.30%。

STC8G 系列内部 IRC 有两个频段，频段的中心频率分别为 20MHz 和 33MHz，20M 频段的调节范围约为 14.7MHz~26MHz，33M 频段的调节范围约为 24.5MHz~42.2MHz（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）。经实际测试，部分芯片的最高工作频率只能为 39MHz，所以为了安全起见，建议用户在 ISP 下载时设置 IRC 频率不要高于 35MHz。

注意：对于一般用户，内部 IRC 频率的调整可以不用关心，因为频率调整工作在进行 ISP 下载时已经自动完成了。所以若用户不需要自行调整频率，那么下面相关的 4 个寄存器也不能随意修改，否则可能会导致工作频率变化。

若用户需要在自己的代码中动态选择芯片预置的频率，请参考预置频率列表以及“用户自定义内部 IRC 频率”的范例程序

内部 IRC 频率调整主要使用下面的 4 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	
IRCBAND	IRC 频段选择	9DH	-	-	-	-	-	-	-	SEL	xxxx,xxxn
LIRTRIM	IRC 频率微调寄存器	9EH	-	-	-	-	-	-	-	LIRTRIM[1:0]	xxxx,xxnn
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]								nnnn,nnnn

6.2.1 IRC 频段选择寄存器 (IRCBAND)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IRCBAND	9DH	-	-	-	-	-	-	-	SEL

SEL: 频段选择

0: 选择 20MHz 频段

1: 选择 33MHz 频段

6.2.2 内部 IRC 频率调整寄存器 (IRTRIM)

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

IRTRIM[7:0]: 内部高精度 IRC 频率调整寄存器

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

6.2.3 内部 IRC 频率微调寄存器 (LIRTRIM)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LIRTRIM	9EH	-	-	-	-	-	-	-	IRTRIM[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%

6.2.4 时钟分频寄存器 (CLKDIV)

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。

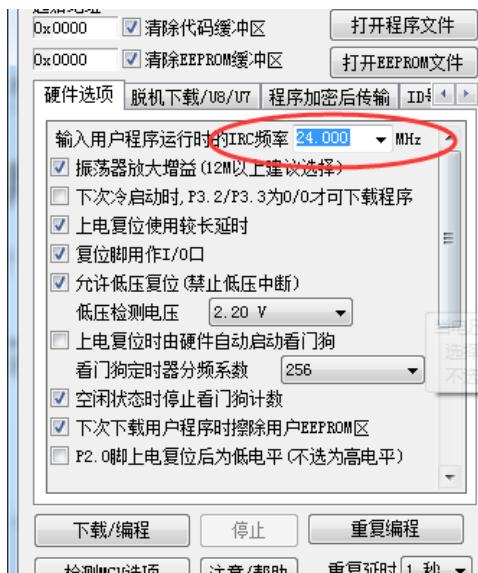
CLKDIV	系统时钟频率
0	MCLK/1
1	MCLK/1
2	MCLK/2
3	MCLK/3
...	...
x	MCLK/x
...	...
255	MCLK/255

STC8G 系列内部的两个频段的可调范围分别为 14.7MHz~26MHz 和 24.5MHz~42.2MHz。虽然 33MHz 频段的上限可调到 40MHz 以上, 但芯片内部的 Flash 程序存储器无法运行到 40MHz 以上的速度, 所以用户在 ISP 下载时设置内部 IRC 频率不能高于 40MHz, 一般建议用户设置为 35MHz 以下。若用户需要较低的工作频率时, 可使用 CLKDIV 寄存器对调节后的频率进行分频, 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。

6.2.5 分频出 3MHz 用户工作频率，并用户动态改变频率追频示例

为得到 3MHz 的频率，可使用 $24\text{MHz} \div 8$ 的方法。

首先在进行 ISP 下载时选择内部 IRC 工作频率为 24MHz，如下图所示，



然后在代码中选择时钟源为内部 IRC，并使用 CLKDIV 寄存器进行 8 分频。

C 语言代码

//测试工作频率为 24MHz

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

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

sfr P_SW2      = 0xba;
sfr IRTRIM    = 0x9f;

sfr P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

void main()
{
    P0M0 = 0x00;
```

```

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

P_SW2 = 0x80;
CLKSEL = 0x00;                                //选择内部IRC(默认)
CLKDIV = 0x08;                                //时钟8分频
P_SW2 = 0x00;

IRTRIM++;
// IRTRIM--;
//while (1);
}

```

汇编代码

; 测试工作频率为 24MHz

<i>P_SW2</i>	<i>DATA</i>	<i>0BAH</i>
<i>IRTRIM</i>	<i>DATA</i>	<i>09FH</i>
<i>CLKSEL</i>	<i>EQU</i>	<i>0FE00H</i>
<i>CLKDIV</i>	<i>EQU</i>	<i>0FE01H</i>
<i>HIRCCR</i>	<i>EQU</i>	<i>0FE02H</i>
<i>XOSCCR</i>	<i>EQU</i>	<i>0FE03H</i>
<i>IRC32KCR</i>	<i>EQU</i>	<i>0FE04H</i>
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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, #5FH</i>
	<i>MOV</i>	<i>P0M0, #00H</i>
	<i>MOV</i>	<i>P0M1, #00H</i>
	<i>MOV</i>	<i>P1M0, #00H</i>
	<i>MOV</i>	<i>P1M1, #00H</i>

```
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

MOV      P_SW2,#80H
MOV      A,#00H           ;选择内部IRC
MOV      DPTR,#CLKSEL
MOVX    @DPTR,A
MOV      A,#08H           ;时钟8 分频
MOV      DPTR,#CLKDIV
MOVX    @DPTR,A
MOV      P_SW2,#00H

INC      IRTRIM          ;IRC 频率向上3%进行微调（注意判断边界）
DEC      IRTRIM          ;IRC 频率向下3%进行微调（注意判断边界）

JMP      $
END
```

6.3 系统复位

STC8G 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，POR，1.7V 附近
- 低压复位，LVD-RESET（2.0V, 2.4V, 2.7V, 3.0V 附近）
- 复位脚复位（**低电平复位**）
- 看门狗复位

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

- 写 IAP_CONTR 的 SWRST 所触发的复位

相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
WDT CONTR	看门狗控制寄存器	C1H	WDT_FLAG	-	EN_WDT	CLR_WDT	IDL_WDT	WDT_PS[2:0]			0x00,0000
IAP CONTR	IAP 控制寄存器	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-				0000,xxxx
RSTCFG	复位配置寄存器	FFH	-	ENLVR	-	P54RST	-	-	LVDS[1:0]		x0x0,xx00

6.3.1 看门狗复位 (WDT_CONTR)

在工业控制/汽车电子/航空航天等需要高可靠性的系统中,为了防止“系统在异常情况下,受到干扰,MCU/CPU 程序跑飞,导致系统长时间异常工作”,通常是引进看门狗,如果 MCU/CPU 不在规定的时间内按要求访问看门狗,就认为 MCU/CPU 处于异常状态,看门狗就会强制 MCU/CPU 复位,使系统重新从头开始执行用户程序。

STC8 系列的看门狗复位是热启动复位中的硬件复位之一。STC8 系列单片机引进此功能,使单片机系统可靠性设计变得更加方便、简洁。STC8 系列看门狗复位状态结束后,系统固定从 ISP 监控程序区启动,与看门狗复位前 IAP_CONTR 寄存器的 SWBS 无关(注意: 此处与 STC15 系列 MCU 不同)

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

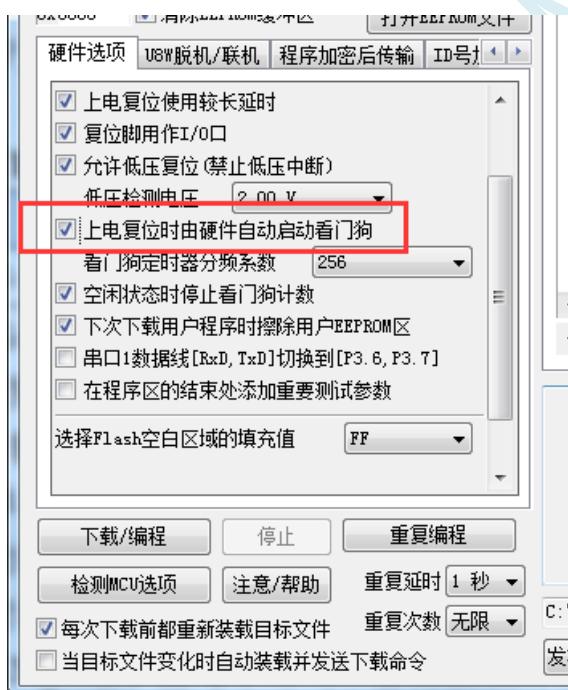
看门狗发生溢出时,硬件自动将此位置 1,需要软件清零。

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。

注意: 看门狗定时器可使用软件方式启动,也可硬件自动启动,一旦看门狗定时器启动后,软件将无法关闭,必须对单片机进行重新上电才可关闭。软件启动看门狗只需要对 EN_WDT 位写 1 即可。若需要硬件启动看门狗,则需要在 ISP 下载时进行如下图所示的设置:



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:

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

6.3.2 软件复位 (IAP_CONTR)

IAP CONTR (IAP 控制寄存器)**对 IAP 控制寄存器写 60H, 可达到对单片机冷启动的效果**

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP CONTR	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	-	-	-

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

6.3.3 低压复位 (RSTCFG)

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

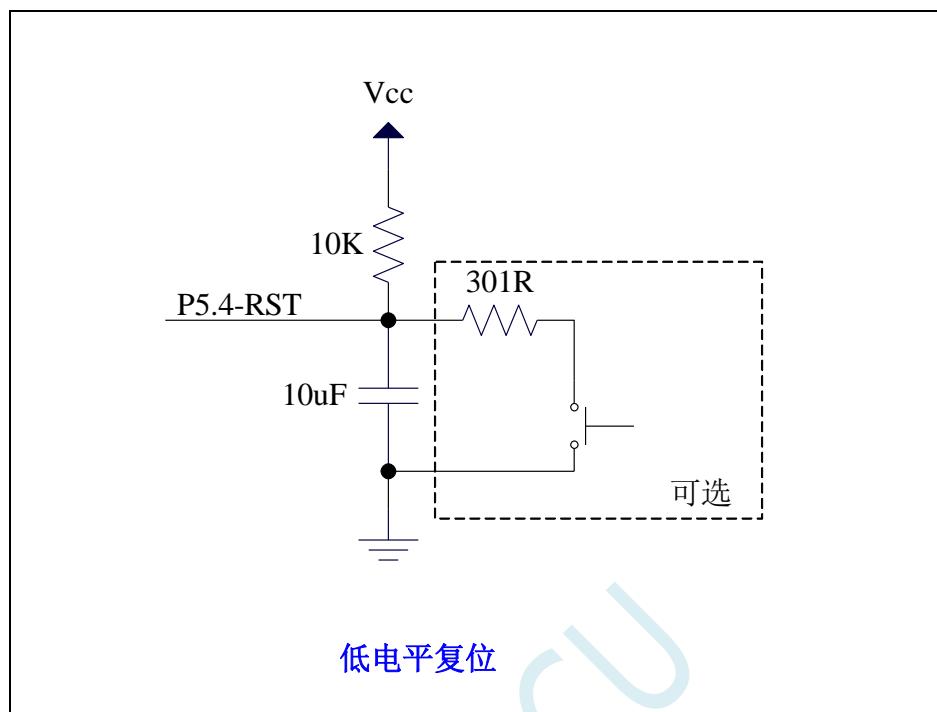
0: RST 管脚用作普通 I/O 口 (P5.4)

1: RST 管脚用作复位脚 (**低电平复位**)

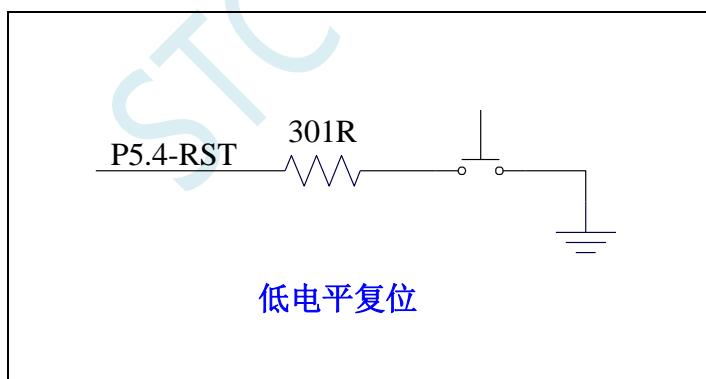
LVDS[1:0]: 低压检测门槛电压设置

LVDS[1:0]	低压检测门槛电压
00	2.0V
01	2.4V
10	2.7V
11	3.0V

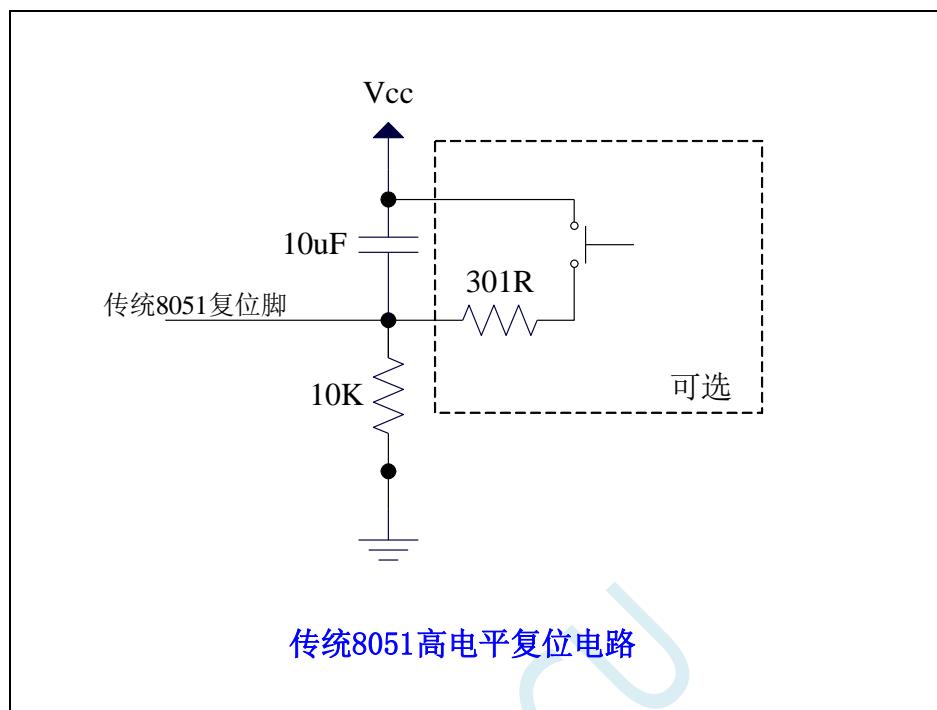
6.3.4 低电平上电复位参考电路（一般不需要）



6.3.5 低电平按键手动复位参考电路



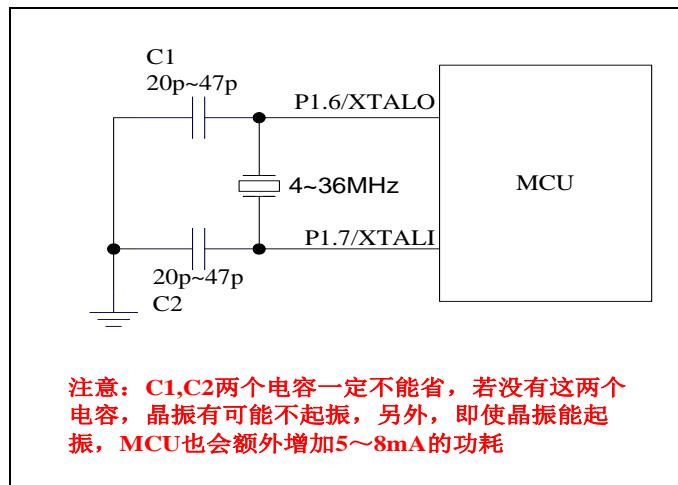
6.3.6 传统 8051 高电平上电复位参考电路



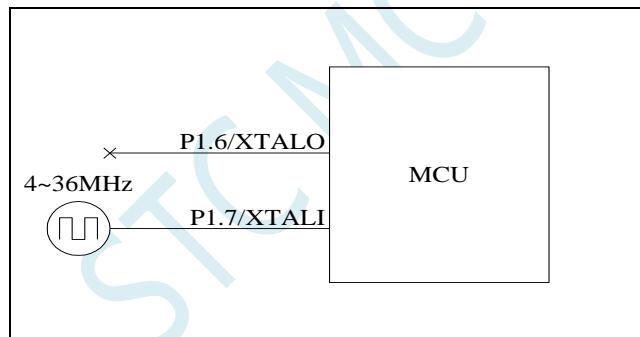
上图为传统 8051 的高电平复位电路，STC8G 的复位为低电平复位，与传统复位电路不同

6.4 外部晶振及外部时钟电路

6.4.1 外部晶振输入电路



6.4.2 外部时钟输入电路 (P1.6 不可用作普通 I/O)



6.5 时钟停振/省电模式与系统电源管理

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

6.5.1 电源控制寄存器 (PCON)

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

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

此位需要用户软件清零。

POF: 上电复位标志位。MCU 每次重新上电后, 硬件自动将此位置 1, 可软件将此位清零。

PD: 时钟停振模式/掉电模式/停电模式控制位

0: 无影响

1: 单片机进入时钟停振模式/掉电模式/停电模式, CPU 以及全部外设均停止工作。唤醒后硬件自动清零。**(注: 时钟停振模式下, CPU 和全部的外设均停止工作, 但 SRAM 和 XRAM 中的数据是一直维持不变的)**

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

注: 虽然 LVD 和比较器均可唤醒时钟停振模式化, 但时钟停振省电模式下, 不建议启动 LVD 和比较器, 否则硬件系统还会自动启动内部 1.19V 的高精准参考源, 这个高精准参考源有有相应的抗温漂和调校线路, 大约会额外增加 300uA 的耗电, 而 MCU 进入时钟停振模式后, 3.3V 工作电压时只耗约 0.4uA 的电流, 所以进入时钟停振模式时不建议开 LVD 和比较器。如果确实需要用, 建议开启掉电唤醒定时器, 掉电唤醒定时器只会增加约 1.4uA 的耗电, 这个耗电一般系统是可以接受的。让掉电唤醒定时器每 5 秒唤醒一次 MCU, 唤醒后可用 LVD、比较器、ADC 检测外部电池电压, 检测工作约耗时 1mS 后再进入时钟停振/省电模式, 这样增加的平均电流小于 1uA, 则整体功耗大约为 2.8uA ($0.4\mu A + 1.4\mu A + 1\mu A$)。

掉电模式可以使用 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)、RXD(P3.0/P3.6/P1.6/P4.3)、RXD2(P1.0/P4.6)、RXD3(P0.0/P5.0)、RXD4(P0.2/P5.2)、CCP0(P1.1/P3.5/P2.5)、CCP1(P1.0/P3.6/P2.6)、CCP2(P3.7/P2.7)、I2C_SDA(P1.4/P2.4/P3.3) 以及比较器中断、低压检测中断、掉电唤醒定时器唤醒。

6.6 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。

6.6.1 掉电唤醒定时器计数寄存器（WKTCL, WKTCH）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
WKTCL	AAH								
WKTCH	ABH	WKTEN							

WKTEN: 掉电唤醒定时器的使能控制位

- 0: 停用掉电唤醒定时器
- 1: 启用掉电唤醒定时器

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 32767 次，则应对{WKTCH[6:0],WKTCL[7:0]}写入 7FFE (即 32766)。（**计数值 0 和计数值 32767 为内部保留值，用户不能使用**）。内部掉电唤醒定时器有自己的内部时钟，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，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)

$$\text{掉电唤醒定时器定时时间} = \frac{10^6 \times 16 \times \text{计数次数}}{F_{\text{wt}}} \text{ (微秒)}$$

假设 F_{wt}=32KHz，则有：

{WKTCH[6:0],WKTCL[7:0]}	掉电唤醒专用定时器计数时间
0 (内部保留)	
1	$10^6 \div 32K \times 16 \times (1+1) \approx 1 \text{ 毫秒}$
9	$10^6 \div 32K \times 16 \times (1+9) \approx 5 \text{ 毫秒}$
99	$10^6 \div 32K \times 16 \times (1+99) \approx 50 \text{ 毫秒}$
999	$10^6 \div 32K \times 16 \times (1+999) \approx 0.5 \text{ 秒}$
4095	$10^6 \div 32K \times 16 \times (1+4095) \approx 2 \text{ 秒}$
32766	$10^6 \div 32K \times 16 \times (1+32766) \approx 16 \text{ 秒}$
32767 (内部保留)	

6.7 范例程序

6.7.1 选择系统时钟源

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

sfr P_SW2 = 0xba;
sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P4M1 = 0xb3;
sfr P4M0 = 0xb4;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;
    CLKSEL = 0x00;                                //选择内部IRC(默认)
    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

```

```

CLKSEL = 0x01;                                //选择外部晶振
P_SW2 = 0x00;
*/
/*
P_SW2 = 0x80;                                //启动内部32K IRC
IRC32KCR = 0x80;                            //等待时钟稳定
while (!(IRC32KCR & I));
CLKDIV = 0x00;                                //时钟不分频
CLKSEL = 0x03;                                //选择内部32K
P_SW2 = 0x00;
*/
while (I);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

<i>P_SW2</i>	<i>DATA</i>	<i>0BAH</i>
<i>CLKSEL</i>	<i>EQU</i>	<i>0FE00H</i>
<i>CLKDIV</i>	<i>EQU</i>	<i>0FE01H</i>
<i>HIRCCR</i>	<i>EQU</i>	<i>0FE02H</i>
<i>XOSCCR</i>	<i>EQU</i>	<i>0FE03H</i>
<i>IRC32KCR</i>	<i>EQU</i>	<i>0FE04H</i>
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</i>
	<i>ORG</i>	<i>0000H</i>
	<i>LJMP</i>	<i>MAIN</i>
	<i>ORG</i>	<i>0100H</i>
MAIN:	<i>MOV</i>	<i>SP, #5FH</i>
	<i>MOV</i>	<i>P0M0, #00H</i>
	<i>MOV</i>	<i>P0M1, #00H</i>
	<i>MOV</i>	<i>P1M0, #00H</i>
	<i>MOV</i>	<i>P1M1, #00H</i>
	<i>MOV</i>	<i>P2M0, #00H</i>
	<i>MOV</i>	<i>P2M1, #00H</i>
	<i>MOV</i>	<i>P3M0, #00H</i>
	<i>MOV</i>	<i>P3M1, #00H</i>
	<i>MOV</i>	<i>P4M0, #00H</i>
	<i>MOV</i>	<i>P4M1, #00H</i>
	<i>MOV</i>	<i>P5M0, #00H</i>
	<i>MOV</i>	<i>P5M1, #00H</i>

```

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

;      MOV      P_SW2,#80H
;      MOV      A,#0COH        ;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,#CLKSEL
;      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,#CLKSEL
;      MOVX    @DPTR,A
;      MOV      P_SW2,#00H

JMP    $

END

```

6.7.2 主时钟分频输出

C 语言代码

//测试工作频率为 11.0592MHz

```

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

#define MCLKOCR  (*(unsigned char volatile xdata *)0xfe05)

sfr P_SW2 = 0xba;
sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;

```

```

sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;
//    MCLKOCR = 0x01;           //主时钟输出到P5.4 口
//    MCLKOCR = 0x02;           //主时钟2 分频输出到P5.4 口
//    MCLKOCR = 0x04;           //主时钟4 分频输出到P5.4 口
//    MCLKOCR = 0x84;           //主时钟4 分频输出到P1.6 口
//    P_SW2 = 0x00;

    while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

```

P_SW2      DATA      0BAH
MCLKOCR   EQU       0FE05H

P0M1      DATA      093H
P0M0      DATA      094H
P1M1      DATA      091H
P1M0      DATA      092H
P2M1      DATA      095H
P2M0      DATA      096H
P3M1      DATA      0B1H
P3M0      DATA      0B2H
P4M1      DATA      0B3H
P4M0      DATA      0B4H
P5M1      DATA      0C9H
P5M0      DATA      0CAH

        ORG      0000H
        LJMP    MAIN

        ORG      0100H
MAIN:

```

```

MOV      SP, #5FH
MOV      P0M0, #00H
MOV      P0M1, #00H
MOV      P1M0, #00H
MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

MOV      P_SW2,#80H
;          A,#01H           ;主时钟输出到P5.4 口
;          A,#02H           ;主时钟2 分频输出到P5.4 口
MOV      A,#04H           ;主时钟4 分频输出到P5.4 口
;          A,#84H           ;主时钟4 分频输出到P1.6 口
MOV      DPTR,#MCLKOCR
MOVX    @DPTR,A
MOV      P_SW2,#00H

JMP      $
END

```

6.7.3 看门狗定时器应用

C 语言代码

//测试工作频率为 11.0592MHz

```

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

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

sfr    P0M1        = 0x93;
sfr    P0M0        = 0x94;
sfr    P1M1        = 0x91;
sfr    P1M0        = 0x92;
sfr    P2M1        = 0x95;
sfr    P2M0        = 0x96;
sfr    P3M1        = 0xb1;
sfr    P3M0        = 0xb2;
sfr    P4M1        = 0xb3;
sfr    P4M0        = 0xb4;
sfr    P5M1        = 0xc9;
sfr    P5M0        = 0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;

```

```

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

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

while (1)
{
//     WDT_CONTR = 0x33;         //清看门狗,否则系统复位
//     WDT_CONTR = 0x34;         //清看门狗,否则系统复位
//     WDT_CONTR = 0x37;         //清看门狗,否则系统复位

    Display();                //显示模块
    Scankey();                //按键扫描模块
    MotorDriver();             //电机驱动模块
}
}

```

汇编代码

; 测试工作频率为 11.0592MHz

<i>WDT CONTR</i>	<i>DATA</i>	<i>0CIH</i>
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH
ORG		0000H
LJMP		MAIN
ORG		0100H
MAIN:		
MOV		SP, #5FH
MOV		P0M0, #00H
MOV		P0M1, #00H
MOV		P1M0, #00H
MOV		P1M1, #00H
MOV		P2M0, #00H
MOV		P2M1, #00H

```

MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

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

LOOP:
;       MOV      WDT_CONTR,#33H      ;清看门狗,否则系统复位
;       MOV      WDT_CONTR,#34H      ;清看门狗,否则系统复位
;       MOV      WDT_CONTR,#37H      ;清看门狗,否则系统复位

LCALL    DISPLAY             ;显示模块
LCALL    SCANKEY            ;按键扫描模块
LCALL    MOTORDRIVER        ;电机驱动模块
JMP     LOOP                ;跳转到LOOP

END

```

6.7.4 软复位实现自定义下载

C 语言代码

//测试工作频率为11.0592MHz

```

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

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

sfr     P0M1     = 0x93;
sfr     P0M0     = 0x94;
sfr     P1M1     = 0x91;
sfr     P1M0     = 0x92;
sfr     P2M1     = 0x95;
sfr     P2M0     = 0x96;
sfr     P3M1     = 0xb1;
sfr     P3M0     = 0xb2;
sfr     P4M1     = 0xb3;
sfr     P4M0     = 0xb4;
sfr     P5M1     = 0xc9;
sfr     P5M0     = 0xca;

```

```

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;

```

```

P2M1 = 0x00;
P3M0 = 0x00;
P3M1 = 0x00;
P4M0 = 0x00;
P4M1 = 0x00;
P5M0 = 0x00;
P5M1 = 0x00;

P32 = I;                                //测试端口
P33 = I;                                //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
    }
}
}

```

汇编代码

; 测试工作频率为 11.0592MHz

IAP CONTR	DATA	0C7H
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH
ORG	0000H	
LJMP	MAIN	
ORG	0100H	
MAIN:		
MOV	SP, #5FH	
MOV	P0M0, #00H	
MOV	P0M1, #00H	
MOV	P1M0, #00H	
MOV	P1M1, #00H	
MOV	P2M0, #00H	
MOV	P2M1, #00H	
MOV	P3M0, #00H	
MOV	P3M1, #00H	
MOV	P4M0, #00H	
MOV	P4M1, #00H	
MOV	P5M0, #00H	
MOV	P5M1, #00H	
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

```

6.7.5 低压检测

C 语言代码

```

//测试工作频率为11.0592MHz

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

sfr     RSTCFG      = 0xff;           //RSTCFG@0
#define ENLVR        0x40             //LVD@2.0V
#define LVD2V0       0x00             //LVD@2.4V
#define LVD2V4       0x01             //LVD@2.7V
#define LVD2V7       0x02             //LVD@3.0V
#define LVD3V0       0x03
sbit    ELVD         = IE^6;          //PCON.5
#define LVDF        0x20
sbit    P32          = P3^2;

sfr     P0M1         = 0x93;
sfr     P0M0         = 0x94;
sfr     P1M1         = 0x91;
sfr     P1M0         = 0x92;
sfr     P2M1         = 0x95;
sfr     P2M0         = 0x96;
sfr     P3M1         = 0xb1;
sfr     P3M0         = 0xb2;
sfr     P4M1         = 0xb3;
sfr     P4M0         = 0xb4;
sfr     P5M1         = 0xc9;
sfr     P5M0         = 0xca;

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

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
}

```

```

P3M1 = 0x00;
P4M0 = 0x00;
P4M1 = 0x00;
P5M0 = 0x00;
P5M1 = 0x00;

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 = I;                                     //使能LVD中断
EA = I;

while (I);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

```

RSTCFG    DATA    0FFH
ENLVR     EQU      40H          ;RSTCFG6
LVD2V0    EQU      00H          ;LVD@2.0V
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

P0M1      DATA    093H
P0M0      DATA    094H
P1M1      DATA    091H
P1M0      DATA    092H
P2M1      DATA    095H
P2M0      DATA    096H
P3M1      DATA    0B1H
P3M0      DATA    0B2H
P4M1      DATA    0B3H
P4M0      DATA    0B4H
P5M1      DATA    0C9H
P5M0      DATA    0CAH

        ORG      0000H
        LJMP    MAIN
        ORG      0033H
        LJMP    LVDISR

        ORG      0100H

LVDISR:
        ANL      PCON,#NOT LVDF      ;清中断标志
        CPL      P3.2                ;测试端口
        RETI

MAIN:
        MOV      SP, #5FH
        MOV      P0M0, #00H
        MOV      P0M1, #00H
        MOV      P1M0, #00H
        MOV      P1M1, #00H
        MOV      P2M0, #00H

```

```

MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

;          ANL      PCON,#NOT LVDF      ;上电后需要先清LVDF 标志
MOV      RSTCFG,#ENLVR / LVD3V0  ;使能3.0V 时低压复位,不产生LVD 中断
MOV      RSTCFG,#LVD3V0        ;使能3.0V 时低压中断
SETB    ELVD                 ;使能LVD 中断
SETB    EA
JMP     $

END

```

6.7.6 省电模式

C 语言代码

//测试工作频率为 11.0592MHz

```

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

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

sfr P0M1     = 0x93;
sfr P0M0     = 0x94;
sfr P1M1     = 0x91;
sfr P1M0     = 0x92;
sfr P2M1     = 0x95;
sfr P2M0     = 0x96;
sfr P3M1     = 0xb1;
sfr P3M0     = 0xb2;
sfr P4M1     = 0xb3;
sfr P4M0     = 0xb4;
sfr P5M1     = 0xc9;
sfr P5M0     = 0xca;

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

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
}

```

```

P2M1 = 0x00;
P3M0 = 0x00;
P3M1 = 0x00;
P4M0 = 0x00;
P4M1 = 0x00;
P5M0 = 0x00;
P5M1 = 0x00;

EX0 = 1;                                //使能INT0 中断,用于唤醒MCU
EA = I;
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
PCON = IDL;                            //MCU 进入 IDLE 模式
// PCON = PD;                           //MCU 进入掉电模式
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
P35 = 0;

while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

<i>IDL</i>	<i>EQU</i>	<i>01H</i>	<i>;PCON.0</i>
<i>PD</i>	<i>EQU</i>	<i>02H</i>	<i>;PCON.1</i>
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>	
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>	
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>	
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>	
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>	
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>	
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>	
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>	
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>	
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>	
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>	
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</i>	
	<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>CPL</i>	<i>P3.4</i>	<i>; 测试端口</i>
	<i>RETI</i>		
<i>MAIN:</i>	<i>MOV</i>	<i>SP, #5FH</i>	
	<i>MOV</i>	<i>P0M0, #00H</i>	
	<i>MOV</i>	<i>P0M1, #00H</i>	
	<i>MOV</i>	<i>P1M0, #00H</i>	

```

MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

SETB    EX0          ;使能INT0 中断,用于唤醒MCU
SETB    EA
NOP
NOP
NOP
NOP
;
MOV      PCON,#IDL   ;MCU 进入 IDLE 模式
MOV      PCON,#PD     ;MCU 进入掉电模式
NOP
NOP
NOP
CLR      P3.5         ;测试端口
JMP      $             

END

```

6.7.7 使用 INT0/INT1/INT2/INT3/INT4 管脚中断唤醒省电模式

C 语言代码

//测试工作频率为 11.0592MHz

```

#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;

sfr      P0M1          = 0x93;
sfr      P0M0          = 0x94;
sfr      P1M1          = 0x91;
sfr      P1M0          = 0x92;
sfr      P2M1          = 0x95;
sfr      P2M0          = 0x96;
sfr      P3M1          = 0xb1;
sfr      P3M0          = 0xb2;
sfr      P4M1          = 0xb3;
sfr      P4M0          = 0xb4;
sfr      P5M1          = 0xc9;
sfr      P5M0          = 0xca;

```

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

    IT0 = 0;                                    //使能INT0 上升沿和下降沿中断
    // 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

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

    EA = 1;

    PCON = 0x02;                             //MCU 进入掉电模式
    _nop_();                                 //掉电模式被唤醒后,MCU 首先会执行此语句
                                              //然后再进入中断服务程序
    _nop_();
}
```

```

_nop_();
_nop_();

while (1)
{
    PII = ~PII;
}

```

汇编代码

; 测试工作频率为 11.0592MHz

<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>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</i>
	<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>0013H</i>
	<i>LJMP</i>	<i>INTIISR</i>
	<i>ORG</i>	<i>0053H</i>
	<i>LJMP</i>	<i>INT2ISR</i>
	<i>ORG</i>	<i>005BH</i>
	<i>LJMP</i>	<i>INT3ISR</i>
	<i>ORG</i>	<i>0083H</i>
	<i>LJMP</i>	<i>INT4ISR</i>
	<i>ORG</i>	<i>0100H</i>
<i>INT0ISR:</i>	<i>CPL</i>	<i>P1.0</i>
	<i>RETI</i>	; 测试端口
<i>INTIISR:</i>	<i>CPL</i>	<i>P1.0</i>
	<i>RETI</i>	; 测试端口
<i>INT2ISR:</i>	<i>CPL</i>	<i>P1.0</i>
	<i>RETI</i>	; 测试端口
<i>INT3ISR:</i>	<i>CPL</i>	<i>P1.0</i>
	<i>RETI</i>	; 测试端口
<i>INT4ISR:</i>		

CPL	P1.0	
RETI		; 测试端口
MAIN:		
MOV	SP, #5FH	
MOV	P0M0, #00H	
MOV	P0M1, #00H	
MOV	P1M0, #00H	
MOV	P1M1, #00H	
MOV	P2M0, #00H	
MOV	P2M1, #00H	
MOV	P3M0, #00H	
MOV	P3M1, #00H	
MOV	P4M0, #00H	
MOV	P4M1, #00H	
MOV	P5M0, #00H	
MOV	P5M1, #00H	
CLR	IT0	; 使能 INT0 上升沿和下降沿中断
;	SETB	; 使能 INT0 下降沿中断
	SETB	; 使能 INT0 中断
CLR	IT1	; 使能 INT1 上升沿和下降沿中断
;	SETB	; 使能 INT1 下降沿中断
	SETB	; 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		; 掉电模式被唤醒后, MCU 首先会执行此语句
NOP		; 然后再进入中断服务程序
NOP		
LOOP:		
CPL	P1.1	
JMP	LOOP	
END		

6.7.8 使用 T0/T1/T2/T3/T4 管脚中断唤醒 MCU 省电模式

C 语言代码

// 测试工作频率为 11.0592MHz

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

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;

sfr P0M1     = 0x93;
sfr P0M0     = 0x94;
sfr P1M1     = 0x91;
sfr P1M0     = 0x92;
sfr P2M1     = 0x95;
sfr P2M0     = 0x96;
sfr P3M1     = 0xb1;
sfr P3M0     = 0xb2;
sfr P4M1     = 0xb3;
sfr P4M0     = 0xb4;
sfr P5M1     = 0xc9;
sfr P5M0     = 0xca;

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

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

void TM2_Isr() interrupt 12
{
    P10 = !P10;                                // 测试端口
}

void TM3_Isr() interrupt 19
{
    P10 = !P10;                                // 测试端口
}

void TM4_Isr() interrupt 20
{
    P10 = !P10;                                // 测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
```

```

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

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

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;
IE2 = ET2;

T3L = 0x66;                                //65536-11.0592M/12/1000
T3H = 0xfc;
T4T3M = 0x08;
IE2 |= ET3;

T4L = 0x66;                                //65536-11.0592M/12/1000
T4H = 0xfc;
T4T3M |= 0x80;
IE2 |= ET4;

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_();
_nop_();
_nop_();

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

```

汇编代码

; 测试工作频率为 11.0592MHz

T2L	DATA	0D7H
T2H	DATA	0D6H
T3L	DATA	0D5H

<i>T3H</i>	DATA	0D4H
<i>T4L</i>	DATA	0D3H
<i>T4H</i>	DATA	0D2H
<i>T4T3M</i>	DATA	0DIH
<i>AUXR</i>	DATA	8EH

<i>IE2</i>	DATA	0AFH
<i>ET2</i>	EQU	04H
<i>ET3</i>	EQU	20H
<i>ET4</i>	EQU	40H

<i>AUXINTIF</i>	DATA	0EFH
<i>T2IF</i>	EQU	01H
<i>T3IF</i>	EQU	02H
<i>T4IF</i>	EQU	04H

<i>P0M1</i>	DATA	093H
<i>P0M0</i>	DATA	094H
<i>P1M1</i>	DATA	091H
<i>P1M0</i>	DATA	092H
<i>P2M1</i>	DATA	095H
<i>P2M0</i>	DATA	096H
<i>P3M1</i>	DATA	0B1H
<i>P3M0</i>	DATA	0B2H
<i>P4M1</i>	DATA	0B3H
<i>P4M0</i>	DATA	0B4H
<i>P5M1</i>	DATA	0C9H
<i>P5M0</i>	DATA	0CAH

<i>ORG</i>	0000H
<i>LJMP</i>	MAIN
<i>ORG</i>	000BH
<i>LJMP</i>	TM0ISR
<i>ORG</i>	001BH
<i>LJMP</i>	TM1ISR
<i>ORG</i>	0063H
<i>LJMP</i>	TM2ISR
<i>ORG</i>	009BH
<i>LJMP</i>	TM3ISR
<i>ORG</i>	00A3H
<i>LJMP</i>	TM4ISR

<i>TM0ISR:</i>	<i>ORG</i>	0100H
	<i>CPL</i>	P1.0
	<i>RETI</i>	; 测试端口

<i>TM1ISR:</i>	<i>CPL</i>	P1.0
	<i>RETI</i>	; 测试端口

<i>TM2ISR:</i>	<i>CPL</i>	P1.0
	<i>RETI</i>	; 测试端口

<i>TM3ISR:</i>	<i>CPL</i>	P1.0
	<i>RETI</i>	; 测试端口

<i>TM4ISR:</i>	<i>CPL</i>	P1.0
	<i>RETI</i>	; 测试端口

MAIN:

```

MOV      SP, #5FH
MOV      P0M0, #00H
MOV      P0M1, #00H
MOV      P1M0, #00H
MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

MOV      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

MOV      TL1,#66H          ;65536-11.0592M/12/1000
MOV      TH1,#0FCH
SETB    TR1             ;启动定时器
SETB    ET1              ;使能定时器中断

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

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

MOV      T4L,#66H          ;65536-11.0592M/12/1000
MOV      T4H,#0FCH
ORL     T4T3M,#80H
ORL     IE2,#ET4          ;启动定时器
                                ;使能定时器中断

SETB    EA

MOV      PCON,#02H        ;MCU 进入掉电模式
NOP
NOP
NOP          ;T0/T1/T2/T3/T4 的外部管脚唤醒后,
                ;不进入中断服务程序, 只是继续执行程序,
                ;与 INT0/INT1/INT2/INT3/INT4 的掉电唤醒不一样
NOP
NOP
NOP          ;建议多加几个 NOP 指令, 如 3 个以上

LOOP:
CPL     P1.1
JMP     LOOP

END

```

6.7.9 使用 RxD/RxD2/RxD3/RxD4 管脚中断唤醒 MCU 省电模式

C 语言代码

//测试工作频率为 11.0592MHz

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

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

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

sbit P1I = P1^1;

sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P4M1 = 0xb3;
sfr P4M0 = 0xb4;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;

void UART1_Isr() interrupt 4
{
}

void UART2_Isr() interrupt 8
{
}

void UART3_Isr() interrupt 17
{
}

void UART4_Isr() interrupt 18
{
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
}
```

```

P4M0 = 0x00;
P4M1 = 0x00;
P5M0 = 0x00;
P5M1 = 0x00;

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/P4.6 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 = 1;                                      //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 = 1;                                       //使能串口中断

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

while (1)
{
    PII = ~PII;
}

```

汇编代码

; 测试工作频率为 11.0592MHz

IE2	DATA	0AFH
ES2	EQU	01H
ES3	EQU	08H
ES4	EQU	10H
P_SW1	DATA	0A2H
P_SW2	DATA	0BAH
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H

```

P5M1      DATA      0C9H
P5M0      DATA      0CAH

ORG      0000H
LJMP     MAIN
ORG      0023H
LJMP     UART1ISR
ORG      0043H
LJMP     UART2ISR
ORG      008BH
LJMP     UART3ISR
ORG      0093H
LJMP     UART4ISR

ORG      0100H
UART1ISR: RETI
UART2ISR: RETI
UART3ISR: RETI
UART4ISR: RETI

MAIN:
MOV      SP, #5FH
MOV      P0M0, #00H
MOV      P0M1, #00H
MOV      P1M0, #00H
MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

MOV      P_SW1,#00H      ;RXD/P3.0 下降沿唤醒
;      MOV      P_SW1,#40H      ;RXD_2/P3.6 下降沿唤醒
;      MOV      P_SW1,#80H      ;RXD_3/P1.6 下降沿唤醒
;      MOV      P_SW1,#0C0H      ;RXD_4/P4.3 下降沿唤醒

;      MOV      P_SW2,#00H      ;RXD2/P1.0 下降沿唤醒
;      MOV      P_SW2,#01H      ;RXD2_2/P4.6 下降沿唤醒

;      MOV      P_SW2,#00H      ;RXD3/P0.0 下降沿唤醒
;      MOV      P_SW2,#02H      ;RXD3_2/P5.0 下降沿唤醒

;      MOV      P_SW2,#00H      ;RXD4/P0.2 下降沿唤醒
;      MOV      P_SW2,#04H      ;RXD4_2/P5.2 下降沿唤醒

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

```

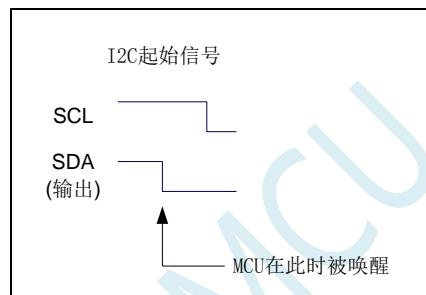
```

NOP ;RxD/RxD2/RxD3/RxD4 及相应的可切换的几组管脚下降
NOP ;沿唤醒后, 不进入中断服务程序, 只是继续执行程序,
MOV     PCON,#02H ;与INT0/INT1/INT2/INT3/INT4 的掉电唤醒不一样
NOP ;建议多加几个NOP 指令, 如3 个以上
NOP ;MCU 进入掉电模式
NOP ;掉电唤醒后不会进入中断服务程序,
NOP
NOP
LOOP:
    CPL     P1.1
    JMP     LOOP

END

```

6.7.10 使用 I2C 的 SDA 脚唤醒 MCU 省电模式



C 语言代码

//测试工作频率为 11.0592MHz

```

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

sfr P_SW2 = 0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CSLCR     (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST     (*(unsigned char volatile xdata *)0xfe84)

sbit P11 = P1^1;

sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P4M1 = 0xb3;
sfr P4M0 = 0xb4;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;

void i2c_isr() interrupt 24

```

```

{
    P_SW2 |= 0x80;
    I2CSLST &= ~0x40;
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x00;                                //SDA/P1.4 下降沿唤醒
//    P_SW2 = 0x10;                                //SDA_2/P2.4 下降沿唤醒
//    P_SW2 = 0x30;                                //SDA_4/P3.3 下降沿唤醒
    P_SW2 |= 0x80;                                //使能 I2C 模块的从机模式
    I2CCFG = 0x80;                                //使能起始信号中断
    I2CSLCR = 0x40;
    EA = 1;

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

    while (1)
    {
        P1I = ~P1I;
    }
}

```

汇编代码

; 测试工作频率为 11.0592MHz

P_SW2	DATA	0BAH
I2CCFG	XDATA	0FE80H
I2CSLCR	XDATA	0FE83H
I2CSLST	XDATA	0FE84H
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H

<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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>DPH</i>
	<i>PUSH</i>	<i>DPL</i>
	<i>ORL</i>	<i>P_SW2,#80H</i>
	<i>MOV</i>	<i>DPTR,#I2CSLST</i>
	<i>MOVX</i>	<i>A,@DPTR</i>
	<i>ANL</i>	<i>A,#NOT 40H</i>
	<i>MOVX</i>	<i>@DPTR,A</i>
	<i>POP</i>	<i>DPL</i>
	<i>POP</i>	<i>DPH</i>
	<i>POP</i>	<i>ACC</i>
	<i>RETI</i>	
<i>MAIN:</i>		
	<i>MOV</i>	<i>SP, #5FH</i>
	<i>MOV</i>	<i>P0M0, #00H</i>
	<i>MOV</i>	<i>P0M1, #00H</i>
	<i>MOV</i>	<i>P1M0, #00H</i>
	<i>MOV</i>	<i>P1M1, #00H</i>
	<i>MOV</i>	<i>P2M0, #00H</i>
	<i>MOV</i>	<i>P2M1, #00H</i>
	<i>MOV</i>	<i>P3M0, #00H</i>
	<i>MOV</i>	<i>P3M1, #00H</i>
	<i>MOV</i>	<i>P4M0, #00H</i>
	<i>MOV</i>	<i>P4M1, #00H</i>
	<i>MOV</i>	<i>P5M0, #00H</i>
	<i>MOV</i>	<i>P5M1, #00H</i>
//	<i>MOV</i>	<i>P_SW2,#00H</i> ;SDA/P1.4 下降沿唤醒
//	<i>MOV</i>	<i>P_SW2,#10H</i> ;SDA_2/P2.4 下降沿唤醒
	<i>MOV</i>	<i>P_SW2,#30H</i> ;SDA_4/P3.3 下降沿唤醒
	<i>ORL</i>	<i>P_SW2,#80H</i>
	<i>MOV</i>	<i>DPTR,#I2CCFG</i>
	<i>MOV</i>	<i>A,#80H</i>
	<i>MOVX</i>	<i>@DPTR,A</i> ;使能 I2C 模块的从机模式
	<i>MOV</i>	<i>DPTR,#I2CSLCR</i>
	<i>MOV</i>	<i>A,#40H</i> ;使能起始信号中断
	<i>SETB</i>	<i>EA</i>
	<i>MOV</i>	<i>PCON,#02H</i> ;MCU 进入掉电模式
	<i>NOP</i>	
<i>LOOP:</i>		
	<i>CPL</i>	<i>P1.1</i>
	<i>JMP</i>	<i>LOOP</i>

END

6.7.11 使用掉电唤醒定时器唤醒省电模式

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr WKTCL      = 0xaa;
sfr WKTCH      = 0xab;

sfr P0M1        = 0x93;
sfr P0M0        = 0x94;
sfr P1M1        = 0x91;
sfr P1M0        = 0x92;
sfr P2M1        = 0x95;
sfr P2M0        = 0x96;
sfr P3M1        = 0xb1;
sfr P3M0        = 0xb2;
sfr P4M1        = 0xb3;
sfr P4M0        = 0xb4;
sfr P5M1        = 0xc9;
sfr P5M0        = 0xca;

sbit P1I         = P1^1;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    WKTCL = 0xff;                                //设定掉电唤醒时钟约为1秒钟
    WKTCH = 0x87;

    while (1)
    {
        _nop_();
        _nop_();
        PCON = 0x02;                               //MCU 进入掉电模式
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
}
```

```

    P11 = ~PII;
}

```

汇编代码

; 测试工作频率为 11.0592MHz

<i>WKTCL</i>	DATA	<i>0AAH</i>
<i>WKTCH</i>	DATA	<i>0ABH</i>

<i>P0M1</i>	DATA	<i>093H</i>
<i>P0M0</i>	DATA	<i>094H</i>
<i>P1M1</i>	DATA	<i>091H</i>
<i>P1M0</i>	DATA	<i>092H</i>
<i>P2M1</i>	DATA	<i>095H</i>
<i>P2M0</i>	DATA	<i>096H</i>
<i>P3M1</i>	DATA	<i>0B1H</i>
<i>P3M0</i>	DATA	<i>0B2H</i>
<i>P4M1</i>	DATA	<i>0B3H</i>
<i>P4M0</i>	DATA	<i>0B4H</i>
<i>P5M1</i>	DATA	<i>0C9H</i>
<i>P5M0</i>	DATA	<i>0CAH</i>
ORG		<i>0000H</i>
LJMP		MAIN
ORG		<i>0100H</i>

MAIN:

MOV	<i>SP, #5FH</i>
MOV	<i>P0M0, #00H</i>
MOV	<i>P0M1, #00H</i>
MOV	<i>P1M0, #00H</i>
MOV	<i>P1M1, #00H</i>
MOV	<i>P2M0, #00H</i>
MOV	<i>P2M1, #00H</i>
MOV	<i>P3M0, #00H</i>
MOV	<i>P3M1, #00H</i>
MOV	<i>P4M0, #00H</i>
MOV	<i>P4M1, #00H</i>
MOV	<i>P5M0, #00H</i>
MOV	<i>P5M1, #00H</i>
MOV	<i>WKTCL,#0FFH</i>
MOV	<i>WKTCH,#87H</i>

; 设定掉电唤醒时钟约为 1 秒钟

LOOP:

NOP	
NOP	
MOV	<i>PCON,#02H</i>
NOP	
NOP	
NOP	
NOP	
CPL	<i>P1.1</i>
JMP	LOOP

END

6.7.12 LVD 中断唤醒省电模式，建议配合使用掉电唤醒定时器

时钟停振省电模式下，不建议启动 LVD 和比较器，否则硬件系统还会自动启动内部 1.19V 的高精准参考源，这个高精准参考源有有相应的抗温漂和调校线路，大约会额外增加 300uA 的耗电，而 MCU 进入时钟停振模式后，3.3V 工作电压时只耗约 0.4uA 的电流，所以进入时钟停振模式时不建议开 LVD 和比较器。如果确实需要用，建议开启掉电唤醒定时器，掉电唤醒定时器只会增加约 1.4uA 的耗电，这个耗电一般系统是可以接受的。让掉电唤醒定时器每 5 秒唤醒一次 MCU，唤醒后可用 LVD、比较器、ADC 检测外部电池电压，检测工作约耗时 1mS 后再进入时钟停振/省电模式，这样增加的平均电流小于 1uA，则整体功耗大约为 2.8uA (0.4uA + 1.4uA + 1uA)。

C 语言代码

//测试工作频率为 11.0592MHz

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

sfr RSTCFG      = 0xff;           //RSTCFG6
#define ENLVR       0x40             //LVD@2.0V
#define LVD2V0      0x00             //LVD@2.4V
#define LVD2V4      0x01             //LVD@2.7V
#define LVD2V7      0x02             //LVD@3.0V
#define LVD3V0      0x03
sbit ELVD        = IE^6;           //PCON.5
#define LVDF        0x20

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

sfr P0M1         = 0x93;
sfr P0M0         = 0x94;
sfr P1M1         = 0x91;
sfr P1M0         = 0x92;
sfr P2M1         = 0x95;
sfr P2M0         = 0x96;
sfr P3M1         = 0xb1;
sfr P3M0         = 0xb2;
sfr P4M1         = 0xb3;
sfr P4M0         = 0xb4;
sfr P5M1         = 0xc9;
sfr P5M0         = 0xca;

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

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
```

```

P2M0 = 0x00;
P2M1 = 0x00;
P3M0 = 0x00;
P3M1 = 0x00;
P4M0 = 0x00;
P4M1 = 0x00;
P5M0 = 0x00;
P5M1 = 0x00;

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_();
_nop_();
_nop_();
_nop_();

while (1)
{
    P1I = ~P1I;
}
}

```

汇编代码

; 测试工作频率为 11.0592MHz

RSTCFG	DATA	0FFH	
ENLVR	EQU	40H	;RSTCFG6
LVD2V0	EQU	00H	;LVD@2.0V
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
P0M1	DATA	093H	
P0M0	DATA	094H	
P1M1	DATA	091H	
P1M0	DATA	092H	
P2M1	DATA	095H	
P2M0	DATA	096H	
P3M1	DATA	0B1H	
P3M0	DATA	0B2H	
P4M1	DATA	0B3H	
P4M0	DATA	0B4H	
P5M1	DATA	0C9H	
P5M0	DATA	0CAH	
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, #5FH	
MOV	P0M0, #00H	
MOV	P0M1, #00H	
MOV	P1M0, #00H	
MOV	P1M1, #00H	
MOV	P2M0, #00H	
MOV	P2M1, #00H	
MOV	P3M0, #00H	
MOV	P3M1, #00H	
MOV	P4M0, #00H	
MOV	P4M1, #00H	
MOV	P5M0, #00H	
MOV	P5M1, #00H	
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		
NOP		
NOP		

LOOP:

CPL	P1.1	
JMP	LOOP	

END

6.7.13 使用 CCP0/CCP1/CCP2 管脚中断唤醒 MCU 省电模式

C 语言代码

//测试工作频率为 11.0592MHz

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

sfr CCON = 0xd8;
sbit CF = CCON^7;
sbit CR = CCON^6;
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 CCAPMI      = 0xdb;
sfr CCAPIL      = 0xeb;
sfr CCAP1H      = 0xfb;
sfr PCA_PWM1    = 0xf3;
sfr CCAPM2      = 0xdc;
sfr CCAP2L      = 0xec;
sfr CCAP2H      = 0xfc;
sfr PCA_PWM2    = 0xf4;

sfr P_SW1       = 0xa2;

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

sfr P0MI        = 0x93;
sfr P0M0        = 0x94;
sfr P1MI        = 0x91;
sfr P1M0        = 0x92;
sfr P2MI        = 0x95;
sfr P2M0        = 0x96;
sfr P3MI        = 0xb1;
sfr P3M0        = 0xb2;
sfr P4MI        = 0xb3;
sfr P4M0        = 0xb4;
sfr P5MI        = 0xc9;
sfr P5M0        = 0xca;

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

void main()
{
    P0M0 = 0x00;
    P0MI = 0x00;
    P1M0 = 0x00;
    P1MI = 0x00;
    P2M0 = 0x00;
    P2MI = 0x00;
    P3M0 = 0x00;
    P3MI = 0x00;
    P4M0 = 0x00;
    P4MI = 0x00;
    P5M0 = 0x00;
    P5MI = 0x00;

    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
    CCAPMI = 0x31;                               //使能 CCP2 口边沿唤醒功能
    CCAPM2 = 0x31;

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

```

```

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

while (1)
{
    P1I = ~P1I;
}

```

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>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>CCAP1L</i>	<i>DATA</i>	<i>0EBH</i>
<i>CCAP1H</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>P_SW1</i>	<i>DATA</i>	<i>0A2H</i>
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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:

ANL	CCON,#NOT 8FH	;清中断标志
CPL	P1.0	;测试端口
RETI		

MAIN:

MOV	SP, #5FH	
MOV	P0M0, #00H	
MOV	P0M1, #00H	
MOV	P1M0, #00H	
MOV	P1M1, #00H	
MOV	P2M0, #00H	
MOV	P2M1, #00H	
MOV	P3M0, #00H	
MOV	P3M1, #00H	
MOV	P4M0, #00H	
MOV	P4M1, #00H	
MOV	P5M0, #00H	
MOV	P5M1, #00H	
MOV	CCON,#00H	
MOV	CMOD,#08H	;PCA 时钟为系统时钟
MOV	CCAPM0,#31H	;使能 CCP0 口边沿唤醒功能
MOV	CCAPM1,#31H	;使能 CCP1 口边沿唤醒功能
MOV	CCAPM2,#31H	;使能 CCP2 口边沿唤醒功能
SETB	CR	;启动 PCA 计时器
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		
NOP		
NOP		

LOOP:

CPL	P1.1	
JMP	LOOP	

END

6.7.14 比较器中断唤醒省电模式，建议配合使用掉电唤醒定时器

时钟停振省电模式下，不建议启动 LVD 和比较器，否则硬件系统还会自动启动内部 1.19V 的高精准参考源，这个高精准参考源有有相应的抗温漂和调校线路，大约会额外增加 300uA 的耗电，而 MCU 进入时钟停振模式后，3.3V 工作电压时只耗约 0.4uA 的电流，所以进入时钟停振模式时不建议开 LVD 和比较器。如果确实需要用，建议开启掉电唤醒定时器，掉电唤醒定时器只会增加约 1.4uA 的耗电，这个耗电一般系统是可以接受的。让掉电唤醒定时器每 5 秒唤醒一次 MCU，唤醒后可用 LVD、比较器、ADC 检测外部电池电压，检测工作约耗时 1mS 后再进入时钟停振/省电模式，这样增加的平均电流小于 1uA，则整体功耗大约为 2.8uA (0.4uA + 1.4uA + 1uA)。

C 语言代码

```
//测试工作频率为11.0592MHz
```

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

```
#include "intrins.h"

sfr CMPCRI      = 0xe6;
sfr CMPCR2      = 0xe7;

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

sfr P0M1         = 0x93;
sfr P0M0         = 0x94;
sfr P1M1         = 0x91;
sfr P1M0         = 0x92;
sfr P2M1         = 0x95;
sfr P2M0         = 0x96;
sfr P3M1         = 0xb1;
sfr P3M0         = 0xb2;
sfr P4M1         = 0xb3;
sfr P4M0         = 0xb4;
sfr P5M1         = 0xc9;
sfr P5M0         = 0xca;

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

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CMPCR2 = 0x00;                                  //使能比较器模块
    CMPCRI = 0x80;                                  //使能比较器边沿中断
    CMPCRI |= 0x30;                                 //P3.6 为 CMP+ 输入脚
    CMPCRI &= ~0x08;                               //P3.7 为 CMP- 输入脚
    CMPCRI |= 0x04;                                 //使能比较器输出
    EA = I;

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

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

```

    }
}

```

汇编代码

; 测试工作频率为 11.0592MHz

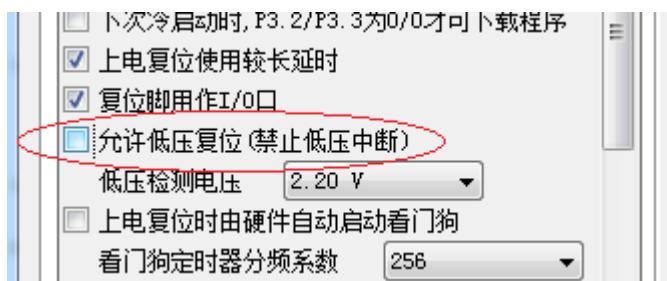
<i>CMPCR1</i>	<i>DATA</i>	<i>0E6H</i>	
<i>CMPCR2</i>	<i>DATA</i>	<i>0E7H</i>	
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>	
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>	
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>	
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>	
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>	
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>	
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>	
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>	
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>	
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>	
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>	
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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>ANL</i>	<i>CMPCR1,#NOT 40H</i>	; 清中断标志
	<i>CPL</i>	<i>P1.0</i>	; 测试端口
	<i>RETI</i>		
<i>MAIN:</i>			
	<i>MOV</i>	<i>SP, #5FH</i>	
	<i>MOV</i>	<i>P0M0, #00H</i>	
	<i>MOV</i>	<i>P0M1, #00H</i>	
	<i>MOV</i>	<i>P1M0, #00H</i>	
	<i>MOV</i>	<i>P1M1, #00H</i>	
	<i>MOV</i>	<i>P2M0, #00H</i>	
	<i>MOV</i>	<i>P2M1, #00H</i>	
	<i>MOV</i>	<i>P3M0, #00H</i>	
	<i>MOV</i>	<i>P3M1, #00H</i>	
	<i>MOV</i>	<i>P4M0, #00H</i>	
	<i>MOV</i>	<i>P4M1, #00H</i>	
	<i>MOV</i>	<i>P5M0, #00H</i>	
	<i>MOV</i>	<i>P5M1, #00H</i>	
	<i>MOV</i>	<i>CMPCR2,#00H</i>	
	<i>MOV</i>	<i>CMPCR1,#80H</i>	; 使能比较器模块
	<i>ORL</i>	<i>CMPCR1,#30H</i>	; 使能比较器边沿中断
	<i>ANL</i>	<i>CMPCR1,#NOT 08H</i>	; P3.6 为 CMP+ 输入脚
	<i>ORL</i>	<i>CMPCR1,#04H</i>	; P3.7 为 CMP- 输入脚
	<i>ORL</i>	<i>CMPCR1,#02H</i>	; 使能比较器输出
	<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

*NOP**NOP**NOP**LOOP:**CPL**P1.1**JMP**LOOP**END*

6.7.15 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

（建议使用 ADC 的第 15 通道检测电池电压，见 ADC 章节）



C 语言代码

//测试工作频率为11.0592MHz

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

#define FOSC          11059200UL
#define TIMS          (65536 - FOSC/4/100)

sfr    RSTCFG      = 0xff;
#define LVD2V0        0x00          //LVD@2.0V
#define LVD2V4        0x01          //LVD@2.4V
#define LVD2V7        0x02          //LVD@2.7V
#define LVD3V0        0x03          //LVD@3.0V

#define LVDF         0x20          //PCON.5

sfr    P0M1         = 0x93;
sfr    P0M0         = 0x94;
sfr    P1M1         = 0x91;
sfr    P1M0         = 0x92;
sfr    P2M1         = 0x95;
sfr    P2M0         = 0x96;
sfr    P3M1         = 0xb1;
sfr    P3M0         = 0xb2;
sfr    P4M1         = 0xb3;
sfr    P4M0         = 0xb4;
sfr    P5M1         = 0xc9;
sfr    P5M0         = 0xca;
```

```
void delay()
{
    int i;

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

void main()
{
    unsigned char power;

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

    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 = LVD2V0;
delay();
PCON &= ~LVDF;
delay();
if (PCON & LVDF)
{
    power >>= 1;
}
}
}
RSTCFG = LVD3V0;
P2 = ~power; //P2.3~P2.0 显示电池电量
}
}

```

汇编代码

; 测试工作频率为 11.0592MHz

RSTCFG	DATA	0FFH	
LVD2V0	EQU	00H	;LVD@2.0V
LVD2V4	EQU	01H	;LVD@2.4V
LVD2V7	EQU	02H	;LVD@2.7V
LVD3V0	EQU	03H	;LVD@3.0V
LVDF	EQU	20H	;PCON.5
P0M1	DATA	093H	
P0M0	DATA	094H	
P1M1	DATA	091H	
P1M0	DATA	092H	
P2M1	DATA	095H	
P2M0	DATA	096H	
P3M1	DATA	0B1H	
P3M0	DATA	0B2H	
P4M1	DATA	0B3H	
P4M0	DATA	0B4H	
P5M1	DATA	0C9H	
P5M0	DATA	0CAH	
	ORG	0000H	
	JMP	MAIN	
	ORG	0100H	
MAIN:			
	MOV	SP, #5FH	
	MOV	P0M0, #00H	
	MOV	P0M1, #00H	
	MOV	P1M0, #00H	
	MOV	P1M1, #00H	
	MOV	P2M0, #00H	
	MOV	P2M1, #00H	
	MOV	P3M0, #00H	
	MOV	P3M1, #00H	
	MOV	P4M0, #00H	
	MOV	P4M1, #00H	
	MOV	P5M0, #00H	

MOV	<i>P5M1, #00H</i>
ANL	<i>PCON,#NOT LVDF</i>
MOV	<i>RSTCFG,#LVD3V0</i>
LOOP:	
MOV	<i>B,#0FH</i>
MOV	<i>RSTCFG,#LVD3V0</i>
CALL	<i>DELAY</i>
ANL	<i>PCON,#NOT LVDF</i>
CALL	<i>DELAY</i>
MOV	<i>A,PCON</i>
ANL	<i>A,#LVDF</i>
JZ	<i>SKIP</i>
MOV	<i>A,B</i>
CLR	<i>C</i>
RRC	<i>A</i>
MOV	<i>B,A</i>
MOV	<i>RSTCFG,#LVD2V7</i>
CALL	<i>DELAY</i>
ANL	<i>PCON,#NOT LVDF</i>
CALL	<i>DELAY</i>
MOV	<i>A,PCON</i>
ANL	<i>A,#LVDF</i>
JZ	<i>SKIP</i>
MOV	<i>A,B</i>
CLR	<i>C</i>
RRC	<i>A</i>
MOV	<i>B,A</i>
MOV	<i>RSTCFG,#LVD2V4</i>
CALL	<i>DELAY</i>
ANL	<i>PCON,#NOT LVDF</i>
CALL	<i>DELAY</i>
MOV	<i>A,PCON</i>
ANL	<i>A,#LVDF</i>
JZ	<i>SKIP</i>
MOV	<i>A,B</i>
CLR	<i>C</i>
RRC	<i>A</i>
MOV	<i>B,A</i>
MOV	<i>RSTCFG,#LVD2V0</i>
CALL	<i>DELAY</i>
ANL	<i>PCON,#NOT LVDF</i>
CALL	<i>DELAY</i>
MOV	<i>A,PCON</i>
ANL	<i>A,#LVDF</i>
JZ	<i>SKIP</i>
MOV	<i>A,B</i>
CLR	<i>C</i>
RRC	<i>A</i>
MOV	<i>B,A</i>
SKIP:	
MOV	<i>A,B</i>
CPL	<i>A</i>
MOV	<i>P2,A</i>
	<i>;P2.3~P2.0 显示电池电量</i>

JMP **LOOP**

DELAY:

MOV **R0,#100**

NEXT:

NOP
NOP
NOP
NOP
DJNZ **R0,NEXT**
RET

END

7 存储器

STC8G 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，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。

STC8G 系列单片机内部集成了大容量的数据存储器。STC8G 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。

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。

STC8G1K08 系列、STC8G1K08-8Pin 系列、STC8G1K08A 系列、STC8G1K08T 系列单片内部集成了 12K 字节的 Flash 程序存储器 (ROM)。

STC8G2K64S4 系列、STC8G2K64S2 系列、STC15H2K64S4 系列单片内部集成了 64K 字节的 Flash 程序存储器 (ROM)。

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。

STC8G 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

STC8G 系列单片机内部集成的 RAM 可用于存放程序执行的中间结果和过程数据。

单片机系列	内部直接访问 RAM (DATA)	内部间接访问 RAM (IDATA)	内部扩展 RAM (XDATA)
STC8G1K08 系列	128 字节	128 字节	1024 字节
STC8G2K64S4 系列	128 字节	128 字节	2048 字节
STC8G2K64S2 系列	128 字节	128 字节	2048 字节
STC8G1K08-8Pin 系列	128 字节	128 字节	1024 字节
STC8G1K08A 系列	128 字节	128 字节	1024 字节
STC8G1K08T 系列	128 字节	128 字节	1024 字节
STC15H2K64S4 系列	128 字节	128 字节	2048 字节

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。

7.2.2 程序状态寄存器 (PSW)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PSW	D0H	CY	AC	F0	RS1	RS0	OV	F1	P

CY: 进/借位标志位。

AC: 辅组进/借位标志位。

F0: 用户标志位 0。

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)

OV: 溢出标志位。

F1: 用户标志位 1。

P: 奇偶校验标志位。

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。

7.2.3 内部扩展 RAM, XRAM, XDATA

STC8G 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 声明存储类型即可。如：

```
unsigned char xdata i;
```

注：pdata 即为 xdata 的低 256 字节，用 MOVX @Ri,A 和 MOVX A,@Ri 进行访问。但读写 pdata 类型的变量比 xdata 类型要慢，所有建议用户代码中均统一使用 xdata 在扩展 RAM 中声明变量。

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。

7.2.4 辅助寄存器 (AUXR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

EXTRAM: 扩展 RAM 访问控制

0: 访问内部扩展 RAM。

1: 内部扩展 RAM 被禁用。

7.2.5 外部扩展 RAM, XRAM, XDATA

STC8G 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。STC8G 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:

7.2.6 总线速度控制寄存器 (BUS_SPEED)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
BUS_SPEED	A1H	RW_S[1:0]							SPEED[2:0]

RW_S[1:0]: RD/WR 控制线选择位

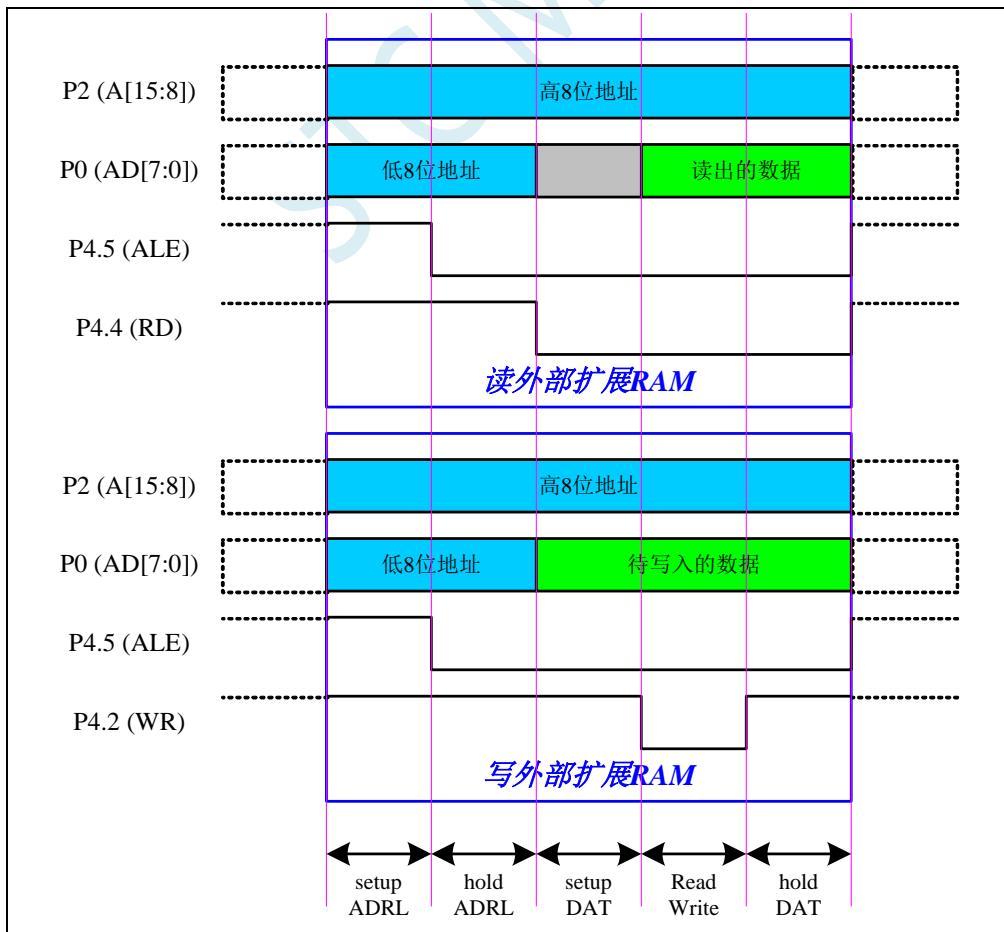
00: P4.4 为 RD, P4.2 为 WR

x1: 保留

SPEED[2:0]: 总线读写速度控制 (读写数据时控制信号和数据信号的准备时间和保持时间)

指令	时钟数	
	访问内部扩展 RAM	访问外部扩展 RAM
MOVX A,@Ri	3	3+5* (SPEED+1)
MOVX @Ri,A	3	3+5* (SPEED+1)
MOVX A,@DPTR	2	2+5* (SPEED+1)
MOVX @DPTR,A	2	2+5* (SPEED+1)

读写外部扩展 RAM 时序如下图所示:



7.2.7 8051 中可位寻址的数据存储器

8051 单片机内部可位寻址的数据存储器包括两部分：第一部分的地址范围为 00H~7FH，第二部分的地址范围是 80H~FFH。00H~7FH 的位寻址区域是数据区 20H~2FH 这 16 个字节的映射；而 80H~FFH 的位寻址区域则是所有的特殊功能寄存器中地址能被 8 整除的 16 个特殊功能寄存器（包括 80H、88H、90H、98H、A0H、A8H、B0H、B8H、C0H、C8H、D0H、D8H、E0H、E8H、F0H、F8H）的映射。

数据存储器地址	位寻址地址							
	B7	B6	B5	B4	B3	B2	B1	B0
F8H (P7)	FFH F8H. 7	FEH F8H. 6	FDH F8H. 5	FCH F8H. 4	FBH F8H. 3	FAH F8H. 2	F9H F8H. 1	F8H F8H. 0
F0H (B)	F7H F0H. 7	F6H F0H. 6	F5H F0H. 5	F4H F0H. 4	F3H F0H. 3	F2H F0H. 2	F1H F0H. 1	F0H F0H. 0
E8H (P6)	EFH E8H. 7	EEH E8H. 6	EDH E8H. 5	ECH E8H. 4	EBH E8H. 3	EAH E8H. 2	E9H E8H. 1	E8H E8H. 0
E0H (ACC)	E7H E0H. 7	E6H E0H. 6	E5H E0H. 5	E4H E0H. 4	E3H E0H. 3	E2H E0H. 2	E1H E0H. 1	E0H E0H. 0
D8H (CCON)	DFH D8H. 7	DEH D8H. 6	DDH D8H. 5	DCH D8H. 4	DBH D8H. 3	DAH D8H. 2	D9H D8H. 1	D8H D8H. 0
D0H (PSW)	D7H D0H. 7	D6H D0H. 6	D5H D0H. 5	D4H D0H. 4	D3H D0H. 3	D2H D0H. 2	D1H D0H. 1	D0H D0H. 0
C8H (P5)	CFH C8H. 7	CEH C8H. 6	CDH C8H. 5	CCH C8H. 4	CBH C8H. 3	CAH C8H. 2	C9H C8H. 1	C8H C8H. 0
C0H (P4)	C7H C0H. 7	C6H C0H. 6	C5H C0H. 5	C4H C0H. 4	C3H C0H. 3	C2H C0H. 2	C1H C0H. 1	C0H C0H. 0
B8H (IP)	BFH B8H. 7	BEH B8H. 6	BDH B8H. 5	BCH B8H. 4	BBH B8H. 3	BAH B8H. 2	B9H B8H. 1	B8H B8H. 0
B0H (P3)	B7H B0H. 7	B6H B0H. 6	B5H B0H. 5	B4H B0H. 4	B3H B0H. 3	B2H B0H. 2	B1H B0H. 1	B0H B0H. 0
A8H (IE)	AFH A8H. 7	AEH A8H. 6	ADH A8H. 5	ACH A8H. 4	ABH A8H. 3	AAH A8H. 2	A9H A8H. 1	A8H A8H. 0
A0H (P2)	A7H A0H. 7	A6H A0H. 6	A5H A0H. 5	A4H A0H. 4	A3H A0H. 3	A2H A0H. 2	A1H A0H. 1	A0H A0H. 0
98H (SCON)	9FH 98H. 7	9EH 98H. 6	9DH 98H. 5	9CH 98H. 4	9BH 98H. 3	9AH 98H. 2	99H 98H. 1	98H 98H. 0
90H (P1)	97H 90H. 7	96H 90H. 6	95H 90H. 5	94H 90H. 4	93H 90H. 3	92H 90H. 2	91H 90H. 1	90H 90H. 0
88H (TCON)	8FH 88H. 7	8EH 88H. 6	8DH 88H. 5	8CH 88H. 4	8BH 88H. 3	8AH 88H. 2	89H 88H. 1	88H 88H. 0
80H (P0)	87H 80H. 7	86H 80H. 6	85H 80H. 5	84H 80H. 4	83H 80H. 3	82H 80H. 2	81H 80H. 1	80H 80H. 0
2FH	7FH 2FH. 7	7EH 2FH. 6	7DH 2FH. 5	7CH 2FH. 4	7BH 2FH. 3	7AH 2FH. 2	79H 2FH. 1	78H 2FH. 0
2EH	77H 2EH. 7	76H 2EH. 6	75H 2EH. 5	74H 2EH. 4	73H 2EH. 3	72H 2EH. 2	71H 2EH. 1	70H 2EH. 0
2DH	6FH 2DH. 7	6EH 2DH. 6	6DH 2DH. 5	6CH 2DH. 4	6BH 2DH. 3	6AH 2DH. 2	69H 2DH. 1	68H 2DH. 0
2CH	67H 2CH. 7	66H 2CH. 6	65H 2CH. 5	64H 2CH. 4	63H 2CH. 3	62H 2CH. 2	61H 2CH. 1	60H 2CH. 0
2BH	5FH 2BH. 7	5EH 2BH. 6	5DH 2BH. 5	5CH 2BH. 4	5BH 2BH. 3	5AH 2BH. 2	59H 2BH. 1	58H 2BH. 0
2AH	57H 2AH. 7	56H 2AH. 6	55H 2AH. 5	54H 2AH. 4	53H 2AH. 3	52H 2AH. 2	51H 2AH. 1	50H 2AH. 0

29H	4FH 29H. 7	4EH 29H. 6	4DH 29H. 5	4CH 29H. 4	4BH 29H. 3	4AH 29H. 2	49H 29H. 1	48H 29H. 0
28H	47H 28H. 7	46H 28H. 6	45H 28H. 5	44H 28H. 4	43H 28H. 3	42H 28H. 2	41H 28H. 1	40H 28H. 0
27H	3FH 27H. 7	3EH 27H. 6	3DH 27H. 5	3CH 27H. 4	3BH 27H. 3	3AH 27H. 2	39H 27H. 1	38H 27H. 0
26H	37H 26H. 7	36H 26H. 6	35H 26H. 5	34H 26H. 4	33H 26H. 3	32H 26H. 2	31H 26H. 1	30H 26H. 0
25H	2FH 25H. 7	2EH 25H. 6	2DH 25H. 5	2CH 25H. 4	2BH 25H. 3	2AH 25H. 2	29H 25H. 1	28H 25H. 0
24H	27H 24H. 7	26H 24H. 6	25H 24H. 5	24H 24H. 4	23H 24H. 3	22H 24H. 2	21H 24H. 1	20H 24H. 0
23H	1FH 23H. 7	1EH 23H. 6	1DH 23H. 5	1CH 23H. 4	1BH 23H. 3	1AH 23H. 2	19H 23H. 1	18H 23H. 0
22H	17H 22H. 7	16H 22H. 6	15H 22H. 5	14H 22H. 4	13H 22H. 3	12H 22H. 2	11H 22H. 1	10H 22H. 0
21H	0FH 21H. 7	0EH 21H. 6	0DH 21H. 5	0CH 21H. 4	0BH 21H. 3	0AH 21H. 2	09H 21H. 1	08H 21H. 0
20H	07H 20H. 7	06H 20H. 6	05H 20H. 5	04H 20H. 4	03H 20H. 3	02H 20H. 2	01H 20H. 1	00H 20H. 0

7.2.8 扩展 SFR 使能寄存器 EAXFR 的使用说明

STC8G/8H 的扩展 SFR 地址范围为 OFA00H~0FFFFH, 如需访问 XFR 区域的扩展 SFR, 需要先将 EAXFR (P_SW2.7) 置 1, 并使用 MOVX A,@DPTR 和 MOVX @DPTR,A 这两条指令进行读写操作。XFR 的地址范围与外部扩展 RAM 地址的 OFA00H~0FFFFH 区域是重叠的。

1、若用户不使用外部扩展 RAM 或者外部扩展 RAM 的最大地址不超过 OFA00H (例如只外扩 32K RAM), 这种情况下不会有不同区域的访问地址冲突, 可以在[上电系统初始化时将 EAXFR 寄存器设置为 1 \(例如: P_SW2 |= 0x80;\)](#), 后续一直保持为 1 不用再修改, 即可正常访问 XFR 区域。

2、若用户有外扩 64K 的扩展 RAM, 则在访问 XFR 和外部扩展 RAM 时需要注意:

[访问 XFR 时需要将 EAXFR 寄存器位设置为 1;](#)

[访问地址范围在 OFA00H~0FFFFH 的外部扩展 RAM 时需要将 EAXFR 设置为 0;](#)

[访问地址范围在 0000H~0F9FFH 的外部扩展 RAM 时, 与 EAXFR 设置的值无关](#)

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

EAXFR: 扩展 RAM 区特殊功能寄存器 (XFR) 访问控制寄存器

0: 禁止访问 XFR

1: 使能访问 XFR。

当需要访问 XFR 时, 必须先将 EAXFR 置 1, 才能对 XFR 进行正常的读写。

7.3 存储器中的特殊参数，在 ISP 下载时可烧录进程程序 FLASH

STC8G 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 1.19V 参考信号源值以及 IRC 参数。

这些参数在 Flash 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
	STC8G1K04-8Pin STC8G1K04A STC8G1K04 STC8G1K04T	STC8G1K08-8Pin STC8G1K08A STC8G1K08 STC8G1K08T	STC8G1K12-8Pin STC8G1K12A STC8G1K12 STC8G1K12T	STC8G1K17-8Pin STC8G1K17A STC8G1K17 STC8G1K17T	
全球唯一 ID 号	0FF9H~0FFFH	1FF9H~1FFFH	2FF9H~2FFFH	43F9H~43FFH	7 字节
内部 1.19V 参考信号源	0FF7H~0FF8H	1FF7H~1FF8H	2FF7H~2FF8H	43F7H~43F8H	毫伏 (高字节在前)
32K 掉电唤醒定时器的频率	0FF5H~0FF6H	1FF5H~1FF6H	2FF5H~2FF6H	43F5H~43F6H	Hz (高字节在前)
22.1184MHz 的 IRC 参数	0FF4H	1FF4H	2FF4H	43F4H	—
24MHz 的 IRC 参数	0FF3H	1FF3H	2FF3H	43F3H	—
20MHz 的 IRC 参数	0FF2H	1FF2H	2FF2H	43F2H	固件版本为 7.3.12U 以及后续版本有效
27MHz 的 IRC 参数	0FF1H	1FF1H	2FF1H	43F1H	
30MHz 的 IRC 参数	0FF0H	1FF0H	2FF0H	43F0H	
33.1776MHz 的 IRC 参数	0FEFH	1FEFH	2FEFH	43EFH	
35MHz 的 IRC 参数	0FEEH	1FEEH	2FEEH	43EEH	
36.864MHz 的 IRC 参数	0FEDH	1FEDH	2FEDH	43EDH	
保留	0FECH	1FECH	2FECH	43ECH	
保留	0FEBH	1FEBH	2FEBH	43EBH	
20M 频段的 VRTRIM 参数	0FEAH	1FEAH	2FEAH	43EAH	
35M 频段的 VRTRIM 参数	0FE9H	1FE9H	2FE9H	43E9H	

参数名称	保存地址					参数说明
	STC8G2K16S2 STC8G2K16S4 STC15H2K16S4	STC8G2K32S2 STC8G2K32S4 STC15H2K32S4	STC8G2K48S2 STC8G2K48S4 STC15H2K48S4	STC8G2K60S2 STC8G2K60S4 STC15H2K60S4	STC8G2K64S2 STC8G2K64S4 STC15H2K64S4	
全球唯一 ID 号	3FF9H~3FFFH	7FF9H~7FFFH	0BFF9H~0BFFFH	0EFF9H~0EFFFH	0FDF9H~0FDFFH	7 字节
内部 1.19V 参考信号源	3FF7H~3FF8H	7FF7H~7FF8H	0BFF7H~0BFF8H	0EFF7H~0EFF8H	0FDF7H~0FDF8H	毫伏 (高字节在前)
32K 掉电唤醒定时器的频率	3FF5H~3FF6H	7FF5H~7FF6H	0BFF5H~0BFF6H	0EFF5H~0EFF6H	0FDF5H~0FDF6H	Hz (高字节在前)
22.1184MHz 的 IRC 参数	3FF4H	7FF4H	0BFF4H	0EFF4H	0FDF4H	—
24MHz 的 IRC 参数	3FF3H	7FF3H	0BFF3H	0EFF3H	0FDF3H	—
20MHz 的 IRC 参数	3FF2H	7FF2H	BFF2H	EFF2H	FDF2H	固件版本为 7.3.12U 以及后续版本有效
27MHz 的 IRC 参数	3FF1H	7FF1H	BFF1H	EFF1H	FDF1H	
30MHz 的 IRC 参数	3FF0H	7FF0H	BFF0H	EFF0H	FDF0H	
33.1776MHz 的 IRC 参数	3FEFH	7FEFH	BFEFH	EFEFH	FDEFH	
35MHz 的 IRC 参数	3FEEH	7FEEH	BFEEH	EFEEH	FDEEH	

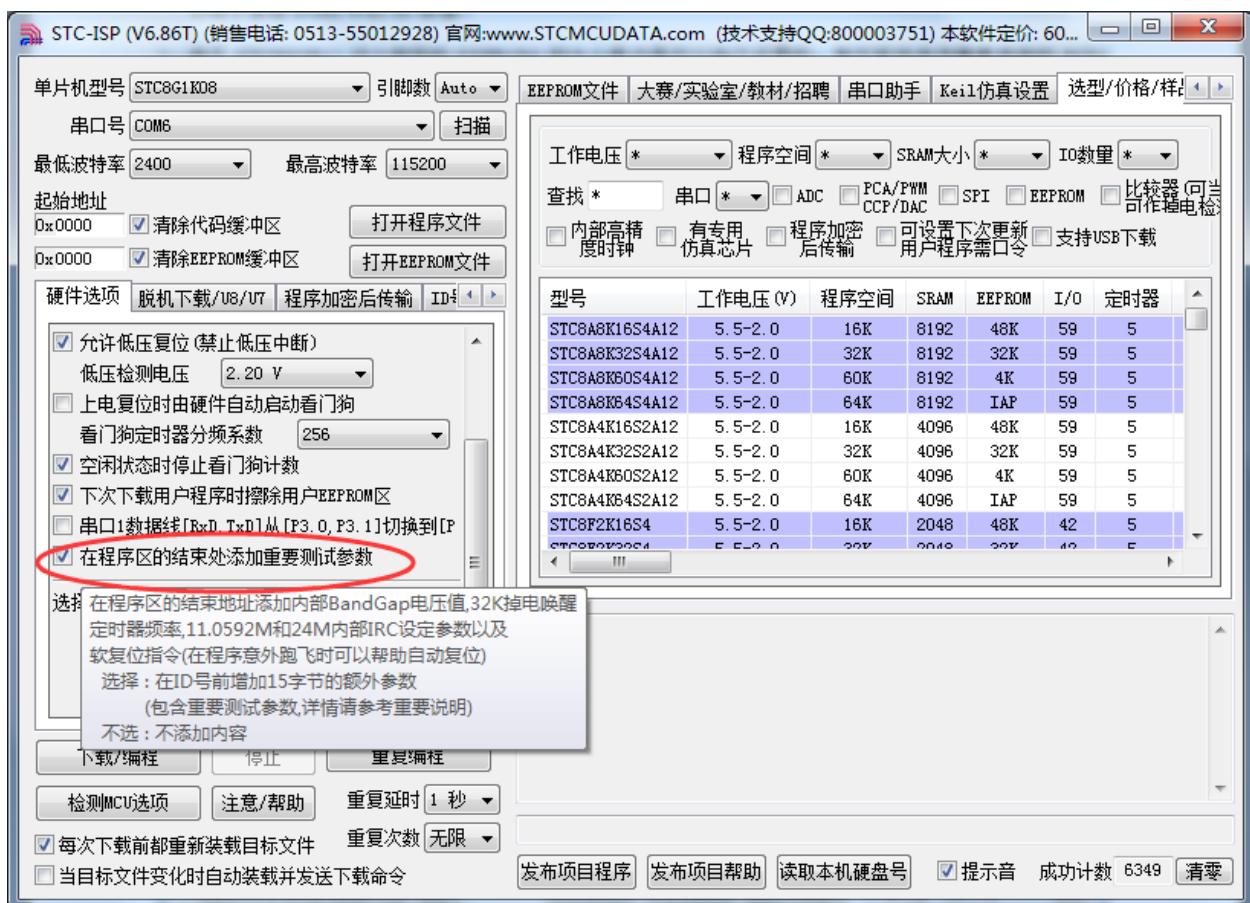
36.864MHz 的 IRC 参数	3FEDH	7FEDH	BFEDH	EFEDH	FDEDH	
保留	3FECH	7FECH	BFECH	EFECH	FDECH	
保留	3FEBH	7FEBH	BFEBH	EFEBH	FDEBH	
20M 频段的 VRTRIM 参数	3FEAH	7FEAH	BFEAH	EFEAH	FDEAH	
35M 频段的 VRTRIM 参数	3FE9H	7FE9H	BFE9H	EFE9H	FDE9H	

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
内部 1.19V 参考信号源	idata: 0EFH~0F0H	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	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 FLASH 程序存储器（ROM）中的 ID 数据。
- 由于 STC8G1K12、STC8G1K12-8Pin、STC8G1K12A、STC8G1K12T、STC8G1K17、STC8G1K17-8Pin、STC8G1K17A、STC8G1K17T、STC8G2K64S4、STC8G2K64S2、STC15H2K64S4 这几个型号的 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 FLASH 程序存储器（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，Flash 程序存储器（ROM）中只有全球唯一 ID 号的数据，而内部 1.19V 参考信号源值、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 读取内部 1.19V 参考信号源值 (从 Flash 程序存储器(ROM)中读取)

C 语言代码

```
//测试工作频率为 11.0592MHz
```

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

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

sfr AUXR      = 0x8e;

sfr P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

bit busy;
int *BGV;

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

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

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

void main()
{
    P0M0 = 0x00;
```

```

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

BGV = (int code *)0x1ff7; // STC8G1K08
UartInit();
ES = 1;
EA = 1;
UartSend(*BGV >> 8); // 读取内部1.19V 参考信号源的高字节
UartSend(*BGV); // 读取内部1.19V 参考信号源的低字节

while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

AUXR	DATA	8EH	
BGV	EQU	01FF7H	;STC8G1K08
BUSY BIT 20H.0			
P0M1	DATA	093H	
P0M0	DATA	094H	
P1M1	DATA	091H	
P1M0	DATA	092H	
P2M1	DATA	095H	
P2M0	DATA	096H	
P3M1	DATA	0B1H	
P3M0	DATA	0B2H	
P4M1	DATA	0B3H	
P4M0	DATA	0B4H	
P5M1	DATA	0C9H	
P5M0	DATA	0CAH	
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           ;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, #5FH
MOV      P0M0, #00H
MOV      P0M1, #00H
MOV      P1M0, #00H
MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

LCALL   UART_INIT
SETB    ES
SETB    EA

MOV      DPTR,#BGV
CLR     A
MOVC   A,@A+DPTR           ;读取内部1.19V 参考信号源的高字节
LCALL   UART_SEND
MOV      A,#1
MOVC   A,@A+DPTR           ;读取内部1.19V 参考信号源的低字节
LCALL   UART_SEND

```

LOOP:

```
JMP     LOOP
```

```
END
```

7.3.2 读取内部1.19V参考信号源值(从RAM中读取)**C 语言代码**

```
//测试工作频率为11.0592MHz
```

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

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

sfr AUXR      = 0x8e;
sfr P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

bit busy;
int *BGV;

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

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

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

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
```

```

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

P5M0 = 0x00;
P5M1 = 0x00;

BGV = (int iodata *)0xef;
UartInit();
ES = 1;
EA = 1;
UartSend(*BGV >> 8);           //读取内部1.19V 参考信号源的高字节
UartSend(*BGV);                 //读取内部1.19V 参考信号源的低字节

while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

AUXR	DATA	8EH
BGV	DATA	0EFH
BUSY	BIT	20H.0
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH
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           ;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, #5FH
MOV      P0M0, #00H
MOV      P0M1, #00H
MOV      P1M0, #00H
MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

LCALL   UART_INIT
SETB    ES
SETB    EA

MOV      R0,#BGV
MOV      A,@R0           ;读取内部 1.19V 参考信号源的高字节
LCALL   UART_SEND
INC     R0
MOV      A,@R0           ;读取内部 1.19V 参考信号源的低字节
LCALL   UART_SEND

```

LOOP:

```
JMP     LOOP
```

```
END
```

7.3.3 读取全球唯一 ID 号 (从 Flash 程序存储器 (ROM) 中读取)

C 语言代码

//测试工作频率为11.0592MHz

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

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

sfr AUXR      = 0x8e;
sfr P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

bit busy;
char *ID;

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

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

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

void main()
{
    char i;

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
```

```

P1M1 = 0x00;
P2M0 = 0x00;
P2M1 = 0x00;
P3M0 = 0x00;
P3M1 = 0x00;
P4M0 = 0x00;
P4M1 = 0x00;
P5M0 = 0x00;
P5M1 = 0x00;

ID = (char code *)0x1ff9; // STC8G1K08
UartInit();
ES = 1;
EA = 1;

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

while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

AUXR	DATA	8EH	
ID	EQU	01FF9H	; STC8G1K08
BUSY	BIT	20H.0	
P0M1	DATA	093H	
P0M0	DATA	094H	
P1M1	DATA	091H	
P1M0	DATA	092H	
P2M1	DATA	095H	
P2M0	DATA	096H	
P3M1	DATA	0B1H	
P3M0	DATA	0B2H	
P4M1	DATA	0B3H	
P4M0	DATA	0B4H	
P5M1	DATA	0C9H	
P5M0	DATA	0CAH	
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      TH1,#0FFH
SETB    TR1
MOV      AUXR,#40H
CLR     BUSY
RET

```

UART_SEND:

```

JB      BUSY,$
SETB    BUSY
MOV      SBUE,A
RET

```

MAIN:

```

MOV      SP, #5FH
MOV      P0M0, #00H
MOV      P0M1, #00H
MOV      P1M0, #00H
MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

LCALL   UART_INIT
SETB    ES
SETB    EA

```

NEXT:

```

MOV      DPTR,#ID
MOV      RI,#7
CLR     A
MOVC   A,@A+DPTR
LCALL   UART_SEND
INC    DPTR
DJNZ   RI,NEXT

```

LOOP:

```
JMP    LOOP
```

```
END
```

7.3.4 读取全球唯一 ID 号 (从 RAM 中读取)

C 语言代码

//测试工作频率为11.0592MHz

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

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

sfr AUXR      = 0x8e;

sfr P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

bit busy;
char *ID;

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

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

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

void main()
{
    char i;
```

```

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

ID = (char idata *)0xf1;
UartInit();
ES = I;
EA = I;

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

while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

AUXR	DATA	8EH
ID	DATA	0FIH
BUSY	BIT	20H.0
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH
ORG		0000H
LJMP		MAIN
ORG		0023H
LJMP		UART_ISR
ORG		0100H
UART_ISR:		
JNB		TI,CHKRI
CLR		TI

<i>CLR</i>	<i>BUSY</i>
<i>CHKRI:</i>	
<i>JNB</i>	<i>RI,UARTISR_EXIT</i>
<i>CLR</i>	<i>RI</i>
<i>UARTISR_EXIT:</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>T1L,#0E8H</i>
<i>MOV</i>	<i>T1H,#0FFH</i>
<i>SETB</i>	<i>TR1</i>
<i>MOV</i>	<i>AUXR,#40H</i>
<i>CLR</i>	<i>BUSY</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>MAIN:</i>	
<i>MOV</i>	<i>SP, #5FH</i>
<i>MOV</i>	<i>P0M0, #00H</i>
<i>MOV</i>	<i>P0M1, #00H</i>
<i>MOV</i>	<i>P1M0, #00H</i>
<i>MOV</i>	<i>P1M1, #00H</i>
<i>MOV</i>	<i>P2M0, #00H</i>
<i>MOV</i>	<i>P2M1, #00H</i>
<i>MOV</i>	<i>P3M0, #00H</i>
<i>MOV</i>	<i>P3M1, #00H</i>
<i>MOV</i>	<i>P4M0, #00H</i>
<i>MOV</i>	<i>P4M1, #00H</i>
<i>MOV</i>	<i>P5M0, #00H</i>
<i>MOV</i>	<i>P5M1, #00H</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,#ID</i>
<i>MOV</i>	<i>RI,#7</i>
<i>NEXT:</i>	
<i>MOV</i>	<i>A,@R0</i>
<i>LCALL</i>	<i>UART_SEND</i>
<i>INC</i>	<i>R0</i>
<i>DJNZ</i>	<i>RI,NEXT</i>
<i>LOOP:</i>	
<i>JMP</i>	<i>LOOP</i>
<i>END</i>	

7.3.5 读取 32K 掉电唤醒定时器的频率 (从 Flash 程序存储器 (ROM) 中读取)

C 语言代码

//测试工作频率为 11.0592MHz

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

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

sfr AUXR = 0x8e;
sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P4M1 = 0xb3;
sfr P4M0 = 0xb4;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;

bit busy;
int *F32K;

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

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

void UartSend(char dat)
```

```

{
    while (busy);
    busy = I;
    SBUF = dat;
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    F32K = (int code *)0x1ff5; // STC8G1K08
    UartInit();
    ES = I;
    EA = I;

    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);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

AUXR	DATA	8EH	
F32K	EQU	01FF5H	; STC8G1K08
BUSY	BIT	20H.0	
P0M1	DATA	093H	
P0M0	DATA	094H	
P1M1	DATA	091H	
P1M0	DATA	092H	
P2M1	DATA	095H	
P2M0	DATA	096H	
P3M1	DATA	0B1H	
P3M0	DATA	0B2H	
P4M1	DATA	0B3H	
P4M0	DATA	0B4H	
P5M1	DATA	0C9H	
P5M0	DATA	0CAH	
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	TH1,#0FFH
SETB	TR1
MOV	AUXR,#40H
CLR	BUSY
RET	
UART_SEND:	
JB	BUSY,\$
SETB	BUSY
MOV	SBUF,A
RET	
MAIN:	
MOV	SP, #5FH
MOV	P0M0, #00H
MOV	P0M1, #00H
MOV	P1M0, #00H
MOV	P1M1, #00H
MOV	P2M0, #00H
MOV	P2M1, #00H
MOV	P3M0, #00H
MOV	P3M1, #00H
MOV	P4M0, #00H
MOV	P4M1, #00H
MOV	P5M0, #00H
MOV	P5M1, #00H
LCALL	UART_INIT
SETB	ES
SETB	EA
MOV	DPTR,#F32K
CLR	A
MOVC	A,@A+DPTR
LCALL	UART_SEND
INC	DPTR
CLR	A
MOVC	A,@A+DPTR
LCALL	UART_SEND
LOOP:	
JMP	LOOP

END

7.3.6 读取32K掉电唤醒定时器的频率(从RAM中读取)

C 语言代码

```
//测试工作频率为11.0592MHz
```

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

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

sfr AUXR      = 0x8e;

sfr P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

bit busy;
int *F32K;

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

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

```

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

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    F32K = (int idata *)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);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

AUXR	DATA	8EH
F32K	DATA	0F8H
BUSY	BIT	20H.0
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH
ORG	DATA	0000H
LJMP	MAIN	
ORG	DATA	0023H
LJMP	DATA	UART_ISR

<p>ORG 0100H</p> <p>UART_ISR:</p> <table border="0"> <tbody> <tr><td>JNB</td><td>TI,CHKRI</td></tr> <tr><td>CLR</td><td>TI</td></tr> <tr><td>CLR</td><td>BUSY</td></tr> </tbody> </table> <p>CHKRI:</p> <table border="0"> <tbody> <tr><td>JNB</td><td>RI,UARTISR_EXIT</td></tr> <tr><td>CLR</td><td>RI</td></tr> </tbody> </table> <p>UARTISR_EXIT:</p> <table border="0"> <tbody> <tr><td>RETI</td></tr> </tbody> </table> <p>UART_INIT:</p> <table border="0"> <tbody> <tr><td>MOV</td><td>SCON,#50H</td></tr> <tr><td>MOV</td><td>TMOD,#00H</td></tr> <tr><td>MOV</td><td>TL1,#0E8H</td></tr> <tr><td>MOV</td><td>TH1,#0FFH</td></tr> <tr><td>SETB</td><td>TR1</td></tr> <tr><td>MOV</td><td>AUXR,#40H</td></tr> <tr><td>CLR</td><td>BUSY</td></tr> <tr><td>RET</td></tr> </tbody> </table> <p>UART_SEND:</p> <table border="0"> <tbody> <tr><td>JB</td><td>BUSY,\$</td></tr> <tr><td>SETB</td><td>BUSY</td></tr> <tr><td>MOV</td><td>SBUF,A</td></tr> <tr><td>RET</td></tr> </tbody> </table> <p>MAIN:</p> <table border="0"> <tbody> <tr><td>MOV</td><td>SP, #5FH</td></tr> <tr><td>MOV</td><td>P0M0, #00H</td></tr> <tr><td>MOV</td><td>P0M1, #00H</td></tr> <tr><td>MOV</td><td>P1M0, #00H</td></tr> <tr><td>MOV</td><td>P1M1, #00H</td></tr> <tr><td>MOV</td><td>P2M0, #00H</td></tr> <tr><td>MOV</td><td>P2M1, #00H</td></tr> <tr><td>MOV</td><td>P3M0, #00H</td></tr> <tr><td>MOV</td><td>P3M1, #00H</td></tr> <tr><td>MOV</td><td>P4M0, #00H</td></tr> <tr><td>MOV</td><td>P4M1, #00H</td></tr> <tr><td>MOV</td><td>P5M0, #00H</td></tr> <tr><td>MOV</td><td>P5M1, #00H</td></tr> <tr><td>LCALL</td><td>UART_INIT</td></tr> <tr><td>SETB</td><td>ES</td></tr> <tr><td>SETB</td><td>EA</td></tr> <tr><td>MOV</td><td>R0,#F32K</td></tr> <tr><td>MOV</td><td>A,@R0</td></tr> <tr><td>LCALL</td><td>UART_SEND</td></tr> <tr><td>INC</td><td>R0</td></tr> <tr><td>MOV</td><td>A,@R0</td></tr> <tr><td>LCALL</td><td>UART_SEND</td></tr> </tbody> </table> <p>LOOP:</p> <table border="0"> <tbody> <tr><td>JMP</td><td>LOOP</td></tr> <tr><td colspan="2">END</td></tr> </tbody> </table>	JNB	TI,CHKRI	CLR	TI	CLR	BUSY	JNB	RI,UARTISR_EXIT	CLR	RI	RETI	MOV	SCON,#50H	MOV	TMOD,#00H	MOV	TL1,#0E8H	MOV	TH1,#0FFH	SETB	TR1	MOV	AUXR,#40H	CLR	BUSY	RET	JB	BUSY,\$	SETB	BUSY	MOV	SBUF,A	RET	MOV	SP, #5FH	MOV	P0M0, #00H	MOV	P0M1, #00H	MOV	P1M0, #00H	MOV	P1M1, #00H	MOV	P2M0, #00H	MOV	P2M1, #00H	MOV	P3M0, #00H	MOV	P3M1, #00H	MOV	P4M0, #00H	MOV	P4M1, #00H	MOV	P5M0, #00H	MOV	P5M1, #00H	LCALL	UART_INIT	SETB	ES	SETB	EA	MOV	R0,#F32K	MOV	A,@R0	LCALL	UART_SEND	INC	R0	MOV	A,@R0	LCALL	UART_SEND	JMP	LOOP	END		<p>;65536-11059200/115200/4=0FFE8H</p> <p>;读取32K 频率的高字节</p> <p>;读取32K 频率的低字节</p>
JNB	TI,CHKRI																																																																																	
CLR	TI																																																																																	
CLR	BUSY																																																																																	
JNB	RI,UARTISR_EXIT																																																																																	
CLR	RI																																																																																	
RETI																																																																																		
MOV	SCON,#50H																																																																																	
MOV	TMOD,#00H																																																																																	
MOV	TL1,#0E8H																																																																																	
MOV	TH1,#0FFH																																																																																	
SETB	TR1																																																																																	
MOV	AUXR,#40H																																																																																	
CLR	BUSY																																																																																	
RET																																																																																		
JB	BUSY,\$																																																																																	
SETB	BUSY																																																																																	
MOV	SBUF,A																																																																																	
RET																																																																																		
MOV	SP, #5FH																																																																																	
MOV	P0M0, #00H																																																																																	
MOV	P0M1, #00H																																																																																	
MOV	P1M0, #00H																																																																																	
MOV	P1M1, #00H																																																																																	
MOV	P2M0, #00H																																																																																	
MOV	P2M1, #00H																																																																																	
MOV	P3M0, #00H																																																																																	
MOV	P3M1, #00H																																																																																	
MOV	P4M0, #00H																																																																																	
MOV	P4M1, #00H																																																																																	
MOV	P5M0, #00H																																																																																	
MOV	P5M1, #00H																																																																																	
LCALL	UART_INIT																																																																																	
SETB	ES																																																																																	
SETB	EA																																																																																	
MOV	R0,#F32K																																																																																	
MOV	A,@R0																																																																																	
LCALL	UART_SEND																																																																																	
INC	R0																																																																																	
MOV	A,@R0																																																																																	
LCALL	UART_SEND																																																																																	
JMP	LOOP																																																																																	
END																																																																																		

7.3.7 用户自定义内部IRC频率(从Flash程序存储器(ROM)中读取)

C 语言代码

//测试工作频率为11.0592MHz

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

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

//下表为STC8G1K08 的参数列表
#define ID_ROMADDR      ((unsigned char code *)0x1ff9)
#define VREF_ROMADDR    ((unsigned int code *)0x1ff7)
#define F32K_ROMADDR   ((unsigned int code *)0x1ff5)
#define T22M_ROMADDR   ((unsigned char code *)0x1ff4)           //22.1184MHz
#define T24M_ROMADDR   ((unsigned char code *)0x1ff3)           //24MHz
#define T20M_ROMADDR   ((unsigned char code *)0x1ff2)           //20MHz
#define T27M_ROMADDR   ((unsigned char code *)0x1ff1)           //27MHz
#define T30M_ROMADDR   ((unsigned char code *)0x1ff0)           //30MHz
#define T33M_ROMADDR   ((unsigned char code *)0x1fef)           //33.1776MHz
#define T35M_ROMADDR   ((unsigned char code *)0x1fee)           //35MHz
#define T36M_ROMADDR   ((unsigned char code *)0x1fed)           //36.864MHz
#define VRT20M_ROMADDR ((unsigned char code *)0x1fea)           //VRTRIM_20M
#define VRT35M_ROMADDR ((unsigned char code *)0x1fe9)           //VRTRIM_35M

sfr P_SW2      = 0xba;
sfr IRCBAND    = 0xd;
sfr IRTRIM     = 0xf;
sfr VRTRIM     = 0xa6;

sfr P1M1       = 0x91;
sfr P1M0       = 0x92;
sfr P0M1       = 0x93;
sfr P0M0       = 0x94;
sfr P2M1       = 0x95;
sfr P2M0       = 0x96;
sfr P3M1       = 0xb1;
sfr P3M0       = 0xb2;
sfr P4M1       = 0xb3;
sfr P4M0       = 0xb4;
sfr P5M1       = 0xc9;
sfr P5M0       = 0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
```

```
P3M0 = 0x00;  
P3M1 = 0x00;  
P4M0 = 0x00;  
P4M1 = 0x00;  
P5M0 = 0x00;  
P5M1 = 0x00;  
  
// //选择20MHz  
// P_SW2 = 0x80;  
// CLKDIV = 0x04;  
// IRTRIM = T20M_ROMADDR;  
// VRTRIM = VRT20M_ROMADDR;  
// IRCBAND = 0x00;  
// CLKDIV = 0x00;  
  
// //选择22.1184MHz  
// P_SW2 = 0x80;  
// CLKDIV = 0x04;  
// IRTRIM = T22M_ROMADDR;  
// VRTRIM = VRT20M_ROMADDR;  
// IRCBAND = 0x00;  
// CLKDIV = 0x00;  
  
// //选择24MHz  
P_SW2 = 0x80;  
CLKDIV = 0x04;  
IRTRIM = T24M_ROMADDR;  
VRTRIM = VRT20M_ROMADDR;  
IRCBAND = 0x00;  
CLKDIV = 0x00;  
  
// //选择27MHz  
// P_SW2 = 0x80;  
// CLKDIV = 0x04;  
// IRTRIM = T27M_ROMADDR;  
// VRTRIM = VRT35M_ROMADDR;  
// IRCBAND = 0x01;  
// CLKDIV = 0x00;  
  
// //选择30MHz  
// P_SW2 = 0x80;  
// CLKDIV = 0x04;  
// IRTRIM = T30M_ROMADDR;  
// VRTRIM = VRT35M_ROMADDR;  
// IRCBAND = 0x01;  
// CLKDIV = 0x00;  
  
// //选择33.1776MHz  
// P_SW2 = 0x80;  
// CLKDIV = 0x04;  
// IRTRIM = T33M_ROMADDR;  
// VRTRIM = VRT35M_ROMADDR;  
// IRCBAND = 0x01;  
// CLKDIV = 0x00;  
  
// //选择35MHz  
// P_SW2 = 0x80;  
// CLKDIV = 0x04;  
// IRTRIM = T35M_ROMADDR;
```

```
// VRTRIM = VRT35M_ROMADDR;
// IRCBAND = 0x01;
// CLKDIV = 0x00;

while (1);
}
```

汇编代码

; 测试工作频率为 11.0592MHz

; 下表为 STC8G1K08 的参数列表

<i>ID_ROMADDR</i>	<i>EQU</i>	<i>01FF9H</i>
<i>VREF_ROMADDR</i>	<i>EQU</i>	<i>01FF7H</i>
<i>F32K_ROMADDR</i>	<i>EQU</i>	<i>01FF5H</i>
<i>T22M_ROMADDR</i>	<i>EQU</i>	<i>01FF4H</i>
		//22.1184MHz
<i>T24M_ROMADDR</i>	<i>EQU</i>	<i>01FF3H</i>
		//24MHz
<i>T20M_ROMADDR</i>	<i>EQU</i>	<i>01FF2H</i>
		//20MHz
<i>T27M_ROMADDR</i>	<i>EQU</i>	<i>01FF1H</i>
		//27MHz
<i>T30M_ROMADDR</i>	<i>EQU</i>	<i>01FF0H</i>
		//30MHz
<i>T33M_ROMADDR</i>	<i>EQU</i>	<i>01FEFH</i>
		//33.1776MHz
<i>T35M_ROMADDR</i>	<i>EQU</i>	<i>01FEEH</i>
		//35MHz
<i>T36M_ROMADDR</i>	<i>EQU</i>	<i>01FEDH</i>
		//36.864MHz
<i>VRT20M_ROMADDR</i>	<i>EQU</i>	<i>01FEAH</i>
		//VRTRIM_20M
<i>VRT35M_ROMADDR</i>	<i>EQU</i>	<i>01FE9H</i>
		//VRTRIM_35M
<i>P_SW2</i>	<i>DATA</i>	<i>0BAH</i>
<i>CLKSEL</i>	<i>EQU</i>	<i>0FE00H</i>
<i>CLKDIV</i>	<i>EQU</i>	<i>0FE01H</i>
<i>IRCBAND</i>	<i>DATA</i>	<i>09DH</i>
<i>IRTRIM</i>	<i>DATA</i>	<i>09FH</i>
<i>VRTRIM</i>	<i>DATA</i>	<i>0A6H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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, #5FH</i>
<i>MOV</i>		<i>P0M0, #00H</i>
<i>MOV</i>		<i>P0M1, #00H</i>
<i>MOV</i>		<i>P1M0, #00H</i>
<i>MOV</i>		<i>P1M1, #00H</i>
<i>MOV</i>		<i>P2M0, #00H</i>
<i>MOV</i>		<i>P2M1, #00H</i>

```
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

; ;选择 20MHz
; MOV      P_SW2,#80H
; MOV      A,#4
; MOV      DPTR,#CLKDIV
; MOV      DPTR,#T20M_ROMADDR
; CLR      A
; MOVC    A,@A+DPTR
; MOV      IRTRIM,A
; MOV      DPTR,#VRT20M_ROMADDR
; CLR      A
; MOVC    A,@A+DPTR
; MOV      VRTRIM,A
; MOV      IRCBAND,#00H
; MOV      A,#0
; MOV      DPTR,#CLKDIV
; MOV      P_SW2,#00H

; ;选择 22.1184MHz
; MOV      P_SW2,#80H
; MOV      A,#4
; MOV      DPTR,#CLKDIV
; MOV      DPTR,#T22M_ROMADDR
; CLR      A
; MOVC    A,@A+DPTR
; MOV      IRTRIM,A
; MOV      DPTR,#VRT20M_ROMADDR
; CLR      A
; MOVC    A,@A+DPTR
; MOV      VRTRIM,A
; MOV      IRCBAND,#00H
; MOV      A,#0
; MOV      DPTR,#CLKDIV
; MOV      P_SW2,#00H

; ;选择 24MHz
MOV      P_SW2,#80H
MOV      A,#4
MOV      DPTR,#CLKDIV
MOV      DPTR,#T24M_ROMADDR
CLR      A
MOVC    A,@A+DPTR
MOV      IRTRIM,A
MOV      DPTR,#VRT20M_ROMADDR
CLR      A
MOVC    A,@A+DPTR
MOV      VRTRIM,A
MOV      IRCBAND,#00H
MOV      A,#0
MOV      DPTR,#CLKDIV
MOV      P_SW2,#00H

; ;选择 27MHz
```

```

;           MOV      P_SW2,#80H
;           MOV      A,#4
;           MOV      DPTR,#CLKDIV
;           MOV      DPTR,#T27M_ROMADDR
;           CLR      A
;           MOVC    A,@A+DPTR
;           MOV      IRTRIM,A
;           MOV      DPTR,#VRT35M_ROMADDR
;           CLR      A
;           MOVC    A,@A+DPTR
;           MOV      VRTRIM,A
;           MOV      IRCBAND,#01H
;           MOV      A,#0
;           MOV      DPTR,#CLKDIV
;           MOV      P_SW2,#00H

;选择30MHz
;           MOV      P_SW2,#80H
;           MOV      A,#4
;           MOV      DPTR,#CLKDIV
;           MOV      DPTR,#T30M_ROMADDR
;           CLR      A
;           MOVC    A,@A+DPTR
;           MOV      IRTRIM,A
;           MOV      DPTR,#VRT35M_ROMADDR
;           CLR      A
;           MOVC    A,@A+DPTR
;           MOV      VRTRIM,A
;           MOV      IRCBAND,#01H
;           MOV      A,#0
;           MOV      DPTR,#CLKDIV
;           MOV      P_SW2,#00H

;选择33.1776MHz
;           MOV      P_SW2,#80H
;           MOV      A,#4
;           MOV      DPTR,#CLKDIV
;           MOV      DPTR,#T33M_ROMADDR
;           CLR      A
;           MOVC    A,@A+DPTR
;           MOV      IRTRIM,A
;           MOV      DPTR,#VRT35M_ROMADDR
;           CLR      A
;           MOVC    A,@A+DPTR
;           MOV      VRTRIM,A
;           MOV      IRCBAND,#01H
;           MOV      A,#0
;           MOV      DPTR,#CLKDIV
;           MOV      P_SW2,#00H

;选择35MHz
;           MOV      P_SW2,#80H
;           MOV      A,#4
;           MOV      DPTR,#CLKDIV
;           MOV      DPTR,#T35M_ROMADDR
;           CLR      A
;           MOVC    A,@A+DPTR
;           MOV      IRTRIM,A
;           MOV      DPTR,#VRT35M_ROMADDR

```

```

;           CLR      A
;           MOVC    A,@A+DPTR
;           MOV     VRTRIM,A
;           MOV     IRCBAND,#01H
;           MOV     A,#0
;           MOV     DPTR,#CLKDIV
;           MOV     P_SW2,#00H

;选择36.864MHz
;           MOV     P_SW2,#80H
;           MOV     A,#4
;           MOV     DPTR,#CLKDIV
;           MOV     DPTR,#T36M_ROMADDR
;           CLR     A
;           MOVC   A,@A+DPTR
;           MOV     IRTRIM,A
;           MOV     DPTR,#VRT35M_ROMADDR
;           CLR     A
;           MOVC   A,@A+DPTR
;           MOV     VRTRIM,A
;           MOV     IRCBAND,#01H
;           MOV     A,#0
;           MOV     DPTR,#CLKDIV
;           MOV     P_SW2,#00H

JMP      $

END

```

7.3.8 用户自定义内部IRC频率(从RAM中读取)

C 语言代码

//测试工作频率为11.0592MHz

```

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

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

sfr P_SW2      = 0xba;
sfr IRTRIM    = 0x9f;

sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

```

```

char      *IRC22M;
char      *IRC24M;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

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

    P_SW2 = 0x80;
    CLKDIV = 0;                  //主时钟不预分频
    P_SW2 = 0x00;

    while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

<i>P_SW2</i>	<i>DATA</i>	<i>0BAH</i>
<i>CLKDIV</i>	<i>EQU</i>	<i>0FE01H</i>
<i>IRTRIM</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>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</i>
	<i>ORG</i>	<i>0000H</i>
	<i>LJMP</i>	<i>MAIN</i>
	<i>ORG</i>	<i>0100H</i>

MAIN:

```
MOV      SP, #5FH
MOV      P0M0, #00H
MOV      P0M1, #00H
MOV      P1M0, #00H
MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

;          MOV      R0,#IRC22M           ;装载 22.1184MHz 的 IRC 参数
;          MOV      IRTRIM,@R0
MOV      R0,#IRC24M           ;装载 24MHz 的 IRC 参数
MOV      IRTRIM,@R0

MOV      P_SW2,#80H
MOV      A,#0                 ;主时钟不预分频
MOV      DPTR,#CLKDIV
MOVX    @DPTR,A
MOV      P_SW2,#00H

JMP      $

END
```

8 特殊功能寄存器

8.1 STC8G1K08 系列

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H		CH	CCAP0H	CCAPIH	CCAP2H			RSTCFG
F0H	B		PCA_PWM0	PCA_PWM1	PCA_PWM2	IAP_TPS		
E8H		CL	CCAP0L	CCAPIL	CCAP2L			AUXINTIF
E0H	ACC			DPS	DPL1	DPH1	CMPCR1	CMPCR2
D8H	CCON	CMOD	CCAPM0	CCAPM1	CCAPM2		ADCCFG	
D0H	PSW						T2H	T2L
C8H	P5	P5M1	P5M0			SPSTAT	SPCTL	SPDAT
C0H		WDT_CONTR	IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP CONTR
B8H	IP	SADEN	P_SW2		ADC_CONTR	ADC_RES	ADC_RESL	
B0H	P3	P3M1	P3M0			IP2	IP2H	IPH
A8H	IE	SADDR	WKTC	WKTC			TA	IE2
A0H			P_SW1					
98H	SCON	SBUF	S2CON	S2BUF		IRCBAND	LIRTRIM	IRTRIM
90H	P1	P1M1	P1M0					
88H	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	INTCLKO
80H		SP	DPL	DPH				PCON

↑

可位寻址 不可位寻址

注意: 寄存器地址能够被 8 整除的才可进行位寻址, 不能被 8 整除的则不可位寻址

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
FEA8H	ADCTIM							
FEA0H			TM2PS					
FE88H	I2CMSAUX							
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE30H		P1IE		P3IE				
FE28H		P1DR		P3DR		P5DR		
FE20H		P1SR		P3SR		P5SR		
FE18H		P1NCS		P3NCS		P5NCS		
FE10H		P1PU		P3PU		P5PU		
FE00H	CLKSEL	CLKDIV	HIRCCR	XOSCCR	IRC32KCR	MCLKOCR	IRCDB	

8.2 STC8G1K08-8Pin 系列

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H								RSTCFG
F0H	B					IAP_TPS		
E8H								AUXINTIF
E0H	ACC			DPS	DPL1	DPH1		
D8H								
D0H	PSW							
C8H	P5	P5M1	P5M0			SPSTAT	SPCTL	SPDAT
C0H		WDT_CONTR	IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP_CONTR
B8H	IP	SADEN	P_SW2					
B0H	P3	P3M1	P3M0			IP2	IP2H	IPH
A8H	IE	SADDR	WKTC	WKTC			TA	IE2
A0H			P_SW1					
98H	SCON	SBUF				IRCBAND	LIRTRIM	IRTRIM
90H								
88H	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	INTCLKO
80H		SP	DPL	DPH				PCON

可位寻址

不可位寻址

注意: 寄存器地址能够被 8 整除的才可进行位寻址, 不能被 8 整除的则不可位寻址

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
FCF0H	MD3	MD2	MD1	MD0	MD5	MD4	ARCON	OPCON
FE88H	I2CMSAUX							
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE28H				P3DR		P5DR		
FE20H				P3SR		P5SR		
FE18H				P3NCS		P5NCS		
FE10H				P3PU		P5PU		
FE00H	CLKSEL	CLKDIV	HIRCCR		IRC32KCR	MCLKOCR	IRCDB	

8.3 STC8G1K08A 系列

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H		CH	CCAP0H	CCAPIH	CCAP2H			RSTCFG
F0H	B		PCA_PWM0	PCA_PWM1	PCA_PWM2	IAP_TPS		
E8H		CL	CCAP0L	CCAP1L	CCAP2L			AUXINTIF
E0H	ACC			DPS	DPL1	DPH1		
D8H	CCON	CMOD	CCAPM0	CCAPM1	CCAPM2		ADCCFG	
D0H	PSW							
C8H	P5	P5M1	P5M0			SPSTAT	SPCTL	SPDAT
C0H		WDT_CONTR	IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP_CONTR
B8H	IP	SADEN	P_SW2		ADC_CONTR	ADC_RES	ADC_RESL	
B0H	P3	P3M1	P3M0			IP2	IP2H	IPH
A8H	IE	SADDR	WKTC	WKTC			TA	IE2
A0H			P_SW1					
98H	SCON	SBUF				IRCBAND	LIRTRIM	IRTRIM
90H								
88H	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	INTCLKO
80H		SP	DPL	DPH				PCON

可位寻址

不可位寻址

注意: 寄存器地址能够被 8 整除的才可进行位寻址, 不能被 8 整除的则不可位寻址

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
FCF0H	MD3	MD2	MD1	MD0	MD5	MD4	ARCON	OPCON
FEA8H	ADCTIM							
FE88H	I2CMSAUX							
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE30H				P3IE		P5IE		
FE28H				P3DR		P5DR		
FE20H				P3SR		P5SR		
FE18H				P3NCS		P5NCS		
FE10H				P3PU		P5PU		
FE00H	CLKSEL	CLKDIV	HIRCCR		IRC32KCR	MCLKOCR	IRCDB	

8.4 STC8G2K64S4 系列

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H		CH	CCAP0H	CCAPIH	CCAP2H		PWMCFG45	RSTCFG
F0H	B	PWMSET	PCA_PWM0	PCA_PWM1	PCA_PWM2	IAP_TPS	PWMCFG01	PWMCFG23
E8H		CL	CCAP0L	CCAP1L	CCAP2L		IP3H	AUXINTIF
E0H	ACC			DPS	DPL1	DPH1	CMPCR1	CMPCR2
D8H	CCON	CMOD	CCAPM0	CCAPM1	CCAPM2		ADCCFG	IP3
D0H	PSW	T4T3M	TH4	TL4	TH3	TL3	T2H	T2L
C8H	P5	P5M1	P5M0			SPSTAT	SPCTL	SPDAT
C0H	P4	WDT_CONTR	IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP_CONTR
B8H	IP	SADEN	P_SW2		ADC_CONTR	ADC_RES	ADC_RESL	
B0H	P3	P3M1	P3M0	P4M1	P4M0	IP2	IP2H	IPH
A8H	IE	SADDR	WKTC	WKTC	S3CON	S3BUF	TA	IE2
A0H	P2	BUS_SPEED	P_SW1					
98H	SCON	SBUF	S2CON	S2BUF		IRCBAND	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

可位寻址

不可位寻址

注意: 寄存器地址能够被 8 整除的才可进行位寻址, 不能被 8 整除的则不可位寻址

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
FCFOH	MD3	MD2	MD1	MD0	MD5	MD4	ARCON	OPCON
FFE8H	PWM27T1H	PWM27T1L	PWM27T2H	PWM27T2L	PWM27CR	PWM27HLD		
FFE0H	PWM26T1H	PWM26T1L	PWM26T2H	PWM26T2L	PWM26CR	PWM26HLD		
FFD8H	PWM25T1H	PWM25T1L	PWM25T2H	PWM25T2L	PWM25CR	PWM25HLD		
FFD0H	PWM24T1H	PWM24T1L	PWM24T2H	PWM24T2L	PWM24CR	PWM24HLD		
FFC8H	PWM23T1H	PWM23T1L	PWM23T2H	PWM23T2L	PWM31CR	PWM23HLD		
FFC0H	PWM22T1H	PWM22T1L	PWM22T2H	PWM22T2L	PWM22CR	PWM22HLD		
FFB8H	PWM21T1H	PWM21T1L	PWM21T2H	PWM21T2L	PWM21CR	PWM21HLD		
FFB0H	PWM20T1H	PWM20T1L	PWM20T2H	PWM20T2L	PWM20CR	PWM20HLD		
FFA0H	PWM2CH	PWM2CL	PWM2CKS	PWM2TADCH	PWM2TADCL	PWM2IF	PWM2FDCR	
FF98H	PWM17T1H	PWM17T1L	PWM17T2H	PWM17T2L	PWM17CR	PWM17HLD		
FF90H	PWM16T1H	PWM16T1L	PWM16T2H	PWM16T2L	PWM16CR	PWM16HLD		
FF88H	PWM15T1H	PWM15T1L	PWM15T2H	PWM15T2L	PWM15CR	PWM15HLD		
FF80H	PWM14T1H	PWM14T1L	PWM14T2H	PWM14T2L	PWM14CR	PWM14HLD		
FF78H	PWM13T1H	PWM13T1L	PWM13T2H	PWM13T2L	PWM31CR	PWM13HLD		
FF70H	PWM12T1H	PWM12T1L	PWM12T2H	PWM12T2L	PWM12CR	PWM12HLD		
FF68H	PWM11T1H	PWM11T1L	PWM11T2H	PWM11T2L	PWM11CR	PWM11HLD		
FF60H	PWM10T1H	PWM10T1L	PWM10T2H	PWM10T2L	PWM10CR	PWM10HLD		
FF50H	PWM1CH	PWM1CL	PWM1CKS			PWM1IF	PWM1FDCR	

FF48H	PWM07T1H	PWM07T1L	PWM07T2H	PWM07T2L	PWM07CR	PWM07HLD		
FF40H	PWM06T1H	PWM06T1L	PWM06T2H	PWM06T2L	PWM06CR	PWM06HLD		
FF38H	PWM05T1H	PWM05T1L	PWM05T2H	PWM05T2L	PWM05CR	PWM05HLD		
FF30H	PWM04T1H	PWM04T1L	PWM04T2H	PWM04T2L	PWM04CR	PWM04HLD		
FF28H	PWM03T1H	PWM03T1L	PWM03T2H	PWM03T2L	PWM31CR	PWM03HLD		
FF20H	PWM02T1H	PWM02T1L	PWM02T2H	PWM02T2L	PWM02CR	PWM02HLD		
FF18H	PWM01T1H	PWM01T1L	PWM01T2H	PWM01T2L	PWM01CR	PWM01HLD		
FF10H	PWM00T1H	PWM00T1L	PWM00T2H	PWM00T2L	PWM00CR	PWM00HLD		
FF00H	PWM0CH	PWM0CL	PWM0CKS	PWM0TADCH	PWM0TADCL	PWM0IF	PWM0FDCR	
FEA8H	ADCTIM							
FEA0H			TM2PS	TM3PS	TM4PS			
FE88H	I2CMSAUX							
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE30H	P0IE	P1IE						
FE28H	P0DR	P1DR	P2DR	P3DR	P4DR	P5DR		
FE20H	P0SR	P1SR	P2SR	P3SR	P4SR	P5SR		
FE18H	P0NCS	P1NCS	P2NCS	P3NCS	P4NCS	P5NCS		
FE10H	P0PU	P1PU	P2PU	P3PU	P4PU	P5PU		
FE00H	CLKSEL	CLKDIV	HIRCCR	XOSCCR	IRC32KCR	MCLKOCR	IRCDB	
FCE8H	PWM57T1H	PWM57T1L	PWM57T2H	PWM57T2L	PWM57CR	PWM57HLD		
FCE0H	PWM56T1H	PWM56T1L	PWM56T2H	PWM56T2L	PWM56CR	PWM56HLD		
FCD8H	PWM55T1H	PWM55T1L	PWM55T2H	PWM55T2L	PWM55CR	PWM55HLD		
FCD0H	PWM54T1H	PWM54T1L	PWM54T2H	PWM54T2L	PWM54CR	PWM54HLD		
FCC8H	PWM53T1H	PWM53T1L	PWM53T2H	PWM53T2L	PWM31CR	PWM53HLD		
FCC0H	PWM52T1H	PWM52T1L	PWM52T2H	PWM52T2L	PWM52CR	PWM52HLD		
FCB8H	PWM51T1H	PWM51T1L	PWM51T2H	PWM51T2L	PWM51CR	PWM51HLD		
FCB0H	PWM50T1H	PWM50T1L	PWM50T2H	PWM50T2L	PWM50CR	PWM50HLD		
FCA0H	PWM5CH	PWM5CL	PWM5CKS			PWM5IF	PWM5FDCR	
FC98H	PWM47T1H	PWM47T1L	PWM47T2H	PWM47T2L	PWM47CR	PWM47HLD		
FC90H	PWM46T1H	PWM46T1L	PWM46T2H	PWM46T2L	PWM46CR	PWM46HLD		
FC88H	PWM45T1H	PWM45T1L	PWM45T2H	PWM45T2L	PWM45CR	PWM45HLD		
FC80H	PWM44T1H	PWM44T1L	PWM44T2H	PWM44T2L	PWM44CR	PWM44HLD		
FC78H	PWM43T1H	PWM43T1L	PWM43T2H	PWM43T2L	PWM31CR	PWM43HLD		
FC70H	PWM42T1H	PWM42T1L	PWM42T2H	PWM42T2L	PWM42CR	PWM42HLD		
FC68H	PWM41T1H	PWM41T1L	PWM41T2H	PWM41T2L	PWM41CR	PWM41HLD		
FC60H	PWM40T1H	PWM40T1L	PWM40T2H	PWM40T2L	PWM40CR	PWM40HLD		
FC50H	PWM4CH	PWM4CL	PWM4CKS	PWM4TADCH	PWM4TADCL	PWM4IF	PWM4FDCR	
FC48H	PWM37T1H	PWM37T1L	PWM37T2H	PWM37T2L	PWM37CR	PWM37HLD		
FC40H	PWM36T1H	PWM36T1L	PWM36T2H	PWM36T2L	PWM36CR	PWM36HLD		
FC38H	PWM35T1H	PWM35T1L	PWM35T2H	PWM35T2L	PWM35CR	PWM35HLD		
FC30H	PWM34T1H	PWM34T1L	PWM34T2H	PWM34T2L	PWM34CR	PWM34HLD		
FC28H	PWM33T1H	PWM33T1L	PWM33T2H	PWM33T2L	PWM31CR	PWM33HLD		
FC20H	PWM32T1H	PWM32T1L	PWM32T2H	PWM32T2L	PWM32CR	PWM32HLD		
FC18H	PWM31T1H	PWM31T1L	PWM31T2H	PWM31T2L	PWM31CR	PWM31HLD		

FC10H	PWM30T1H	PWM30T1L	PWM30T2H	PWM30T2L	PWM30CR	PWM30HLD		
FC00H	PWM3CH	PWM3CL	PWM3CKS			PWM3IF	PWM3FDCR	

STCMCU

8.5 STC8G2K64S2 系列

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H		CH	CCAP0H	CCAPIH	CCAP2H		PWMCFG45	RSTCFG
F0H	B	PWMSET	PCA_PWM0	PCA_PWM1	PCA_PWM2	IAP_TPS	PWMCFG04	PWMCFG23
E8H		CL	CCAP0L	CCAP1L	CCAP2L		IP3H	AUXINTIF
E0H	ACC			DPS	DPL1	DPH1	CMPCR1	CMPCR2
D8H	CCON	CMOD	CCAPM0	CCAPM1	CCAPM2		ADCCFG	IP3
D0H	PSW	T4T3M	TH4	TL4	TH3	TL3	T2H	T2L
C8H	P5	P5M1	P5M0			SPSTAT	SPCTL	SPDAT
C0H	P4	WDT_CONTR	IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP_CONTR
B8H	IP	SADEN	P_SW2		ADC_CONTR	ADC_RES	ADC_RESL	
B0H	P3	P3M1	P3M0	P4M1	P4M0	IP2	IP2H	IPH
A8H	IE	SADDR	WKTC	WKTC			TA	IE2
A0H	P2	BUS_SPEED	P_SW1					
98H	SCON	SBUF	S2CON	S2BUF		IRCBAND	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

可位寻址

不可位寻址

注意: 寄存器地址能够被 8 整除的才可进行位寻址, 不能被 8 整除的则不可位寻址

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
FCFOH	MD3	MD2	MD1	MD0	MD5	MD4	ARCON	OPCON
FFE8H	PWM27T1H	PWM27T1L	PWM27T2H	PWM27T2L	PWM27CR	PWM27HLD		
FFE0H	PWM26T1H	PWM26T1L	PWM26T2H	PWM26T2L	PWM26CR	PWM26HLD		
FFD8H	PWM25T1H	PWM25T1L	PWM25T2H	PWM25T2L	PWM25CR	PWM25HLD		
FFD0H	PWM24T1H	PWM24T1L	PWM24T2H	PWM24T2L	PWM24CR	PWM24HLD		
FFC8H	PWM23T1H	PWM23T1L	PWM23T2H	PWM23T2L	PWM31CR	PWM23HLD		
FFC0H	PWM22T1H	PWM22T1L	PWM22T2H	PWM22T2L	PWM22CR	PWM22HLD		
FFB8H	PWM21T1H	PWM21T1L	PWM21T2H	PWM21T2L	PWM21CR	PWM21HLD		
FFB0H	PWM20T1H	PWM20T1L	PWM20T2H	PWM20T2L	PWM20CR	PWM20HLD		
FFA0H	PWM2CH	PWM2CL	PWM2CKS	PWM2TADCH	PWM2TADCL	PWM2IF	PWM2FDCR	
FF98H	PWM17T1H	PWM17T1L	PWM17T2H	PWM17T2L	PWM17CR	PWM17HLD		
FF90H	PWM16T1H	PWM16T1L	PWM16T2H	PWM16T2L	PWM16CR	PWM16HLD		
FF88H	PWM15T1H	PWM15T1L	PWM15T2H	PWM15T2L	PWM15CR	PWM15HLD		
FF80H	PWM14T1H	PWM14T1L	PWM14T2H	PWM14T2L	PWM14CR	PWM14HLD		
FF78H	PWM13T1H	PWM13T1L	PWM13T2H	PWM13T2L	PWM31CR	PWM13HLD		
FF70H	PWM12T1H	PWM12T1L	PWM12T2H	PWM12T2L	PWM12CR	PWM12HLD		
FF68H	PWM11T1H	PWM11T1L	PWM11T2H	PWM11T2L	PWM11CR	PWM11HLD		
FF60H	PWM10T1H	PWM10T1L	PWM10T2H	PWM10T2L	PWM10CR	PWM10HLD		
FF50H	PWM1GH	PWM1CL	PWM1CKS			PWM1HF	PWM1FDCR	

FF48H	PWM07T1H	PWM07T1L	PWM07T2H	PWM07T2L	PWM07CR	PWM07HLD		
FF40H	PWM06T1H	PWM06T1L	PWM06T2H	PWM06T2L	PWM06CR	PWM06HLD		
FF38H	PWM05T1H	PWM05T1L	PWM05T2H	PWM05T2L	PWM05CR	PWM05HLD		
FF30H	PWM04T1H	PWM04T1L	PWM04T2H	PWM04T2L	PWM04CR	PWM04HLD		
FF28H	PWM03T1H	PWM03T1L	PWM03T2H	PWM03T2L	PWM31CR	PWM03HLD		
FF20H	PWM02T1H	PWM02T1L	PWM02T2H	PWM02T2L	PWM02CR	PWM02HLD		
FF18H	PWM01T1H	PWM01T1L	PWM01T2H	PWM01T2L	PWM01CR	PWM01HLD		
FF10H	PWM00T1H	PWM00T1L	PWM00T2H	PWM00T2L	PWM00CR	PWM00HLD		
FF00H	PWM0CH	PWM0CL	PWM0CKS	PWM0TADCH	PWM0TADCL	PWM0IF	PWM0FDCLR	
FEA8H	ADCTIM							
FEA0H			TM2PS	TM3PS	TM4PS			
FE88H	I2CMSAUX							
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE30H	P0IE	P1IE						
FE28H	P0DR	P1DR	P2DR	P3DR	P4DR	P5DR		
FE20H	P0SR	P1SR	P2SR	P3SR	P4SR	P5SR		
FE18H	P0NCS	P1NCS	P2NCS	P3NCS	P4NCS	P5NCS		
FE10H	P0PU	P1PU	P2PU	P3PU	P4PU	P5PU		
FE00H	CLKSEL	CLKDIV	HIRCCR	XOSCCR	IRC32KCR	MCLKOCR	IRCDB	
FCE8H	PWM57T1H	PWM57T1L	PWM57T2H	PWM57T2L	PWM57CR	PWM57HLD		
FCE0H	PWM56T1H	PWM56T1L	PWM56T2H	PWM56T2L	PWM56CR	PWM56HLD		
FCD8H	PWM55T1H	PWM55T1L	PWM55T2H	PWM55T2L	PWM55CR	PWM55HLD		
FCD0H	PWM54T1H	PWM54T1L	PWM54T2H	PWM54T2L	PWM54CR	PWM54HLD		
FCC8H	PWM53T1H	PWM53T1L	PWM53T2H	PWM53T2L	PWM31CR	PWM53HLD		
FCC0H	PWM52T1H	PWM52T1L	PWM52T2H	PWM52T2L	PWM52CR	PWM52HLD		
FCB8H	PWM51T1H	PWM51T1L	PWM51T2H	PWM51T2L	PWM51CR	PWM51HLD		
FCB0H	PWM50T1H	PWM50T1L	PWM50T2H	PWM50T2L	PWM50CR	PWM50HLD		
FCA0H	PWM5CH	PWM5CL	PWM5CKS			PWM5IF	PWM5FDCLR	
FC98H	PWM47T1H	PWM47T1L	PWM47T2H	PWM47T2L	PWM47CR	PWM47HLD		
FC90H	PWM46T1H	PWM46T1L	PWM46T2H	PWM46T2L	PWM46CR	PWM46HLD		
FC88H	PWM45T1H	PWM45T1L	PWM45T2H	PWM45T2L	PWM45CR	PWM45HLD		
FC80H	PWM44T1H	PWM44T1L	PWM44T2H	PWM44T2L	PWM44CR	PWM44HLD		
FC78H	PWM43T1H	PWM43T1L	PWM43T2H	PWM43T2L	PWM31CR	PWM43HLD		
FC70H	PWM42T1H	PWM42T1L	PWM42T2H	PWM42T2L	PWM42CR	PWM42HLD		
FC68H	PWM41T1H	PWM41T1L	PWM41T2H	PWM41T2L	PWM41CR	PWM41HLD		
FC60H	PWM40T1H	PWM40T1L	PWM40T2H	PWM40T2L	PWM40CR	PWM40HLD		
FC50H	PWM4CH	PWM4CL	PWM4CKS	PWM4TADCH	PWM4TADCL	PWM4IF	PWM4FDCLR	
FC48H	PWM37T1H	PWM37T1L	PWM37T2H	PWM37T2L	PWM37CR	PWM37HLD		
FC40H	PWM36T1H	PWM36T1L	PWM36T2H	PWM36T2L	PWM36CR	PWM36HLD		
FC38H	PWM35T1H	PWM35T1L	PWM35T2H	PWM35T2L	PWM35CR	PWM35HLD		
FC30H	PWM34T1H	PWM34T1L	PWM34T2H	PWM34T2L	PWM34CR	PWM34HLD		
FC28H	PWM33T1H	PWM33T1L	PWM33T2H	PWM33T2L	PWM31CR	PWM33HLD		
FC20H	PWM32T1H	PWM32T1L	PWM32T2H	PWM32T2L	PWM32CR	PWM32HLD		
FC18H	PWM31T1H	PWM31T1L	PWM31T2H	PWM31T2L	PWM31CR	PWM31HLD		

FC10H	PWM30T1H	PWM30T1L	PWM30T2H	PWM30T2L	PWM30CR	PWM30HLD		
FC00H	PWM3CH	PWM3CL	PWM3CKS			PWM3IF	PWM3FDCR	

STCMCU

8.6 STC8G1K08T 系列

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H		CH	CCAP0H	CCAPIH	CCAP2H			RSTCFG
F0H	B		PCA_PWM0	PCA_PWM1	PCA_PWM2	IAP_TPS		
E8H		CL	CCAP0L	CCAP1L	CCAP2L			AUXINTIF
E0H	ACC			DPS	DPL1	DPH1	CMPCR1	CMPCR2
D8H	CCON	CMOD	CCAPM0	CCAPM1	CCAPM2		ADCCFG	
D0H	PSW						T2H	T2L
C8H	P5	P5M1	P5M0			SPSTAT	SPCTL	SPDAT
C0H		WDT_CONTR	IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP_CONTR
B8H	IP	SADEN	P_SW2		ADC_CONTR	ADC_RES	ADC_RESL	
B0H	P3	P3M1	P3M0			IP2	IP2H	IPH
A8H	IE	SADDR	WKTC	WKTC			TA	IE2
A0H			P_SW1					
98H	SCON	SBUF				IRCBAND	LIRTRIM	IRTRIM
90H	P1	P1M1	P1M0					
88H	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	INTCLKO
80H		SP	DPL	DPH				PCON

可位寻址

不可位寻址

注意: 寄存器地址能够被 8 整除的才可进行位寻址, 不能被 8 整除的则不可位寻址

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
FEA8H	ADCTIM							
FEA0H			TM2PS					
FE88H	I2CMSAUX							
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE30H		P1IE		P3IE		P5IE		
FE28H		P1DR		P3DR		P5DR		
FE20H		P1SR		P3SR		P5SR		
FE18H		P1NCS		P3NCS		P5NCS		
FE10H		P1PU		P3PU		P5PU		
FE00H	CLKSEL	CLKDIV	HIRCCR	XOSCCR	IRC32KCR	MCLKOCR	IRCDB	
FB68H	TSTH12H	TSTH12L	TSTH13H	TSTH13L	TSTH14H	TSTH14L	TSTH15H	TSTH15L
FB60H	TSTH08H	TSTH08L	TSTH09H	TSTH09L	TSTH10H	TSTH10L	TSTH11H	TSTH11L
FB58H	TSTH04H	TSTH04L	TSTH05H	TSTH05L	TSTH06H	TSTH06L	TSTH07H	TSTH07L
FB50H	TSTH00H	TSTH00L	TSTH01H	TSTH01L	TSTH02H	TSTH02L	TSTH03H	TSTH03L
FB48H	TSRT	TSDATH	TSDATL					
FB40H	TSCHEN1	TSCHEN2	TSCFG1	TSCFG2	TSWUTC	TSCTRL	TSSTA1	TSSTA2
FB28H	COM0_DC_H	COM1_DC_H	COM2_DC_H	COM3_DC_H	COM4_DC_H	COM5_DC_H	COM6_DC_H	COM7_DC_H
FB20H	COM0_DC_L	COM1_DC_L	COM2_DC_L	COM3_DC_L	COM4_DC_L	COM5_DC_L	COM6_DC_L	COM7_DC_L
FB18H	COM0_DA_H	COM1_DA_H	COM2_DA_H	COM3DA_H	COM4_DA_H	COM5_DA_H	COM6_DA_H	COM7_DA_H

FB10H	COM0_DA_L	COM1_DA_L	COM2_DA_L	COM3DA_L	COM4_DA_L	COM5_DA_L	COM6_DA_L	COM7_DA_L
FB00H	COMEN	SEGENL	SEGENH	LEDCTRL	LEDCKS			

STCMCU

8.7 STC15H2K64S4 系列

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H		CH	CCAP0H	CCAPIH	CCAP2H		PWMCFG45	RSTCFG
F0H	B	PWMSET	PCA_PWM0	PCA_PWM1	PCA_PWM2	IAP_TPS	PWMCFG01	PWMCFG23
E8H		CL	CCAP0L	CCAP1L	CCAP2L		IP3H	AUXINTIF
E0H	ACC			DPS	DPL1	DPH1	CMPCR1	CMPCR2
D8H	CCON	CMOD	CCAPM0	CCAPM1	CCAPM2		ADCCFG	IP3
D0H	PSW	T4T3M	TH4	TL4	TH3	TL3	T2H	T2L
C8H	P5	P5M1	P5M0			SPSTAT	SPCTL	SPDAT
C0H	P4	WDT_CONTR	IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP_CONTR
B8H	IP	SADEN	P_SW2		ADC_CONTR	ADC_RES	ADC_RESL	
B0H	P3	P3M1	P3M0	P4M1	P4M0	IP2	IP2H	IPH
A8H	IE	SADDR	WKTC	WKTC	S3CON	S3BUF	TA	IE2
A0H	P2	BUS_SPEED	P_SW1					
98H	SCON	SBUF	S2CON	S2BUF		IRCBAND	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

可位寻址

不可位寻址

注意: 寄存器地址能够被 8 整除的才可进行位寻址, 不能被 8 整除的则不可位寻址

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
FCFOH	MD3	MD2	MD1	MD0	MD5	MD4	ARCON	OPCON
FFE8H	PWM27T1H	PWM27T1L	PWM27T2H	PWM27T2L	PWM27CR	PWM27HLD		
FFE0H	PWM26T1H	PWM26T1L	PWM26T2H	PWM26T2L	PWM26CR	PWM26HLD		
FFD8H	PWM25T1H	PWM25T1L	PWM25T2H	PWM25T2L	PWM25CR	PWM25HLD		
FFD0H	PWM24T1H	PWM24T1L	PWM24T2H	PWM24T2L	PWM24CR	PWM24HLD		
FFC8H	PWM23T1H	PWM23T1L	PWM23T2H	PWM23T2L	PWM31CR	PWM23HLD		
FFC0H	PWM22T1H	PWM22T1L	PWM22T2H	PWM22T2L	PWM22CR	PWM22HLD		
FFB8H	PWM21T1H	PWM21T1L	PWM21T2H	PWM21T2L	PWM21CR	PWM21HLD		
FFB0H	PWM20T1H	PWM20T1L	PWM20T2H	PWM20T2L	PWM20CR	PWM20HLD		
FFA0H	PWM2CH	PWM2CL	PWM2CKS	PWM2TADCH	PWM2TADCL	PWM2IF	PWM2FDCR	
FF98H	PWM17T1H	PWM17T1L	PWM17T2H	PWM17T2L	PWM17CR	PWM17HLD		
FF90H	PWM16T1H	PWM16T1L	PWM16T2H	PWM16T2L	PWM16CR	PWM16HLD		
FF88H	PWM15T1H	PWM15T1L	PWM15T2H	PWM15T2L	PWM15CR	PWM15HLD		
FF80H	PWM14T1H	PWM14T1L	PWM14T2H	PWM14T2L	PWM14CR	PWM14HLD		
FF78H	PWM13T1H	PWM13T1L	PWM13T2H	PWM13T2L	PWM31CR	PWM13HLD		
FF70H	PWM12T1H	PWM12T1L	PWM12T2H	PWM12T2L	PWM12CR	PWM12HLD		
FF68H	PWM11T1H	PWM11T1L	PWM11T2H	PWM11T2L	PWM11CR	PWM11HLD		
FF60H	PWM10T1H	PWM10T1L	PWM10T2H	PWM10T2L	PWM10CR	PWM10HLD		
FF50H	PWM1CH	PWM1CL	PWM1CKS			PWM1IF	PWM1FDCR	

FF48H	PWM07T1H	PWM07T1L	PWM07T2H	PWM07T2L	PWM07CR	PWM07HLD		
FF40H	PWM06T1H	PWM06T1L	PWM06T2H	PWM06T2L	PWM06CR	PWM06HLD		
FF38H	PWM05T1H	PWM05T1L	PWM05T2H	PWM05T2L	PWM05CR	PWM05HLD		
FF30H	PWM04T1H	PWM04T1L	PWM04T2H	PWM04T2L	PWM04CR	PWM04HLD		
FF28H	PWM03T1H	PWM03T1L	PWM03T2H	PWM03T2L	PWM31CR	PWM03HLD		
FF20H	PWM02T1H	PWM02T1L	PWM02T2H	PWM02T2L	PWM02CR	PWM02HLD		
FF18H	PWM01T1H	PWM01T1L	PWM01T2H	PWM01T2L	PWM01CR	PWM01HLD		
FF10H	PWM00T1H	PWM00T1L	PWM00T2H	PWM00T2L	PWM00CR	PWM00HLD		
FF00H	PWM0CH	PWM0CL	PWM0CKS	PWM0TADCH	PWM0TADCL	PWM0IF	PWM0FDCR	
FEA8H	ADCTIM							
FEA0H			TM2PS	TM3PS	TM4PS			
FE88H	I2CMSAUX							
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE30H	P0IE	P1IE						
FE28H	P0DR	P1DR	P2DR	P3DR	P4DR	P5DR		
FE20H	P0SR	P1SR	P2SR	P3SR	P4SR	P5SR		
FE18H	P0NCS	P1NCS	P2NCS	P3NCS	P4NCS	P5NCS		
FE10H	P0PU	P1PU	P2PU	P3PU	P4PU	P5PU		
FE00H	CLKSEL	CLKDIV	HIRCCR	XOSCCR	IRC32KCR	MCLKOCR	IRCDB	
FCE8H	PWM57T1H	PWM57T1L	PWM57T2H	PWM57T2L	PWM57CR	PWM57HLD		
FCE0H	PWM56T1H	PWM56T1L	PWM56T2H	PWM56T2L	PWM56CR	PWM56HLD		
FCD8H	PWM55T1H	PWM55T1L	PWM55T2H	PWM55T2L	PWM55CR	PWM55HLD		
FCD0H	PWM54T1H	PWM54T1L	PWM54T2H	PWM54T2L	PWM54CR	PWM54HLD		
FCC8H	PWM53T1H	PWM53T1L	PWM53T2H	PWM53T2L	PWM31CR	PWM53HLD		
FCC0H	PWM52T1H	PWM52T1L	PWM52T2H	PWM52T2L	PWM52CR	PWM52HLD		
FCB8H	PWM51T1H	PWM51T1L	PWM51T2H	PWM51T2L	PWM51CR	PWM51HLD		
FCB0H	PWM50T1H	PWM50T1L	PWM50T2H	PWM50T2L	PWM50CR	PWM50HLD		
FCA0H	PWM5CH	PWM5CL	PWM5CKS			PWM5IF	PWM5FDCR	
FC98H	PWM47T1H	PWM47T1L	PWM47T2H	PWM47T2L	PWM47CR	PWM47HLD		
FC90H	PWM46T1H	PWM46T1L	PWM46T2H	PWM46T2L	PWM46CR	PWM46HLD		
FC88H	PWM45T1H	PWM45T1L	PWM45T2H	PWM45T2L	PWM45CR	PWM45HLD		
FC80H	PWM44T1H	PWM44T1L	PWM44T2H	PWM44T2L	PWM44CR	PWM44HLD		
FC78H	PWM43T1H	PWM43T1L	PWM43T2H	PWM43T2L	PWM31CR	PWM43HLD		
FC70H	PWM42T1H	PWM42T1L	PWM42T2H	PWM42T2L	PWM42CR	PWM42HLD		
FC68H	PWM41T1H	PWM41T1L	PWM41T2H	PWM41T2L	PWM41CR	PWM41HLD		
FC60H	PWM40T1H	PWM40T1L	PWM40T2H	PWM40T2L	PWM40CR	PWM40HLD		
FC50H	PWM4CH	PWM4CL	PWM4CKS	PWM4TADCH	PWM4TADCL	PWM4IF	PWM4FDCR	
FC48H	PWM37T1H	PWM37T1L	PWM37T2H	PWM37T2L	PWM37CR	PWM37HLD		
FC40H	PWM36T1H	PWM36T1L	PWM36T2H	PWM36T2L	PWM36CR	PWM36HLD		
FC38H	PWM35T1H	PWM35T1L	PWM35T2H	PWM35T2L	PWM35CR	PWM35HLD		
FC30H	PWM34T1H	PWM34T1L	PWM34T2H	PWM34T2L	PWM34CR	PWM34HLD		
FC28H	PWM33T1H	PWM33T1L	PWM33T2H	PWM33T2L	PWM31CR	PWM33HLD		
FC20H	PWM32T1H	PWM32T1L	PWM32T2H	PWM32T2L	PWM32CR	PWM32HLD		
FC18H	PWM31T1H	PWM31T1L	PWM31T2H	PWM31T2L	PWM31CR	PWM31HLD		

FC10H	PWM30T1H	PWM30T1L	PWM30T2H	PWM30T2L	PWM30CR	PWM30HLD		
FC00H	PWM3CH	PWM3CL	PWM3CKS			PWM3IF	PWM3FDCR	

STCMCU

8.8 特殊功能寄存器列表

注意: 寄存器地址能够被 8 整除的才可进行位寻址, 不能被 8 整除的则不可位寻址。

STC8G 系列能进行位寻址的寄存器: P0 (80H)、TCON (88H)、P1 (90H)、SCON (98H)、P2 (A0H)、IE (A8H)、P3 (B0H)、IP (B8H)、P4 (C0H)、P5 (C8H)、PSW (D0H)、CCON (D8H)、ACC (E0H)、P6 (E8H)、B (F0H)、P7 (F8H)

符号	描述	地址	位地址与符号								复位值	
			B7	B6	B5	B4	B3	B2	B1	B0		
P0	P0 端口	80H	P07	P06	P05	P04	P03	P02	P01	P00	1111,1111	
SP	堆栈指针	81H									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 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	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000	
P1	P1 端口	90H	P17	P16	P15	P14	P13	P12	P11	P10	1111,1111	
P1M1	P1 口配置寄存器 1	91H	P17M1	P16M1	P15M1	P14M1	P13M1	P12M1	P11M1	P10M1	1111,1111	
P1M0	P1 口配置寄存器 0	92H	P17M0	P16M0	P15M0	P14M0	P13M0	P12M0	P11M0	P10M0	0000,0000	
P0M1	P0 口配置寄存器 1	93H	P07M1	P06M1	P05M1	P04M1	P03M1	P02M1	P01M1	P00M1	1111,1111	
P0M0	P0 口配置寄存器 0	94H	P07M0	P06M0	P05M0	P04M0	P03M0	P02M0	P01M0	P00M0	0000,0000	
P2M1	P2 口配置寄存器 1	95H	P27M1	P26M1	P25M1	P24M1	P23M1	P22M1	P21M1	P20M1	1111,1111	
P2M0	P2 口配置寄存器 0	96H	P27M0	P26M0	P25M0	P24M0	P23M0	P22M0	P21M0	P20M0	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	0x00,0000	
S2BUF	串口 2 数据寄存器	9BH									0000,0000	
IRCBAND	IRC 频段选择检测	9DH	-	-	-	-	-	-	-	SEL	xxxx,xxxn	
LIRTRIM	IRC 频率微调寄存器	9EH	-	-	-	-	-	-	-	LIRTRIM[1:0]	xxxx,xxnn	
IRTRIM	IRC 频率调整寄存器	9FH									nnnn,nnnn	
P2	P2 端口	A0H	P27	P26	P25	P24	P23	P22	P21	P20	1111,1111	
BUS_SPEED	总线速度控制寄存器	A1H	RW_S[1:0]								SPEED[2:0]	00xx,x000
P_SW1	外设端口切换寄存器 1	A2H	S1_S[1:0]		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	
WK TCL	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	WKTN									0111,1111				
S3CON	串口 3 控制寄存器	ACH	S3SM0	S3ST4	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	ETKSUI	ET4	ET3	ES4	ES3	ET2	ESPI	ES2		0000,0000				
P3	P3 端口	B0H	P37	P36	P35	P34	P33	P32	P31	P30		1111,1111				
P3M1	P3 口配置寄存器 1	B1H	P37M1	P36M1	P35M1	P34M1	P33M1	P32M1	P31M1	P30M1		1111,1100				
P3M0	P3 口配置寄存器 0	B2H	P37M0	P36M0	P35M0	P34M0	P33M0	P32M0	P31M0	P30M0		0000,0000				
P4M1	P4 口配置寄存器 1	B3H	P47M1	P46M1	P45M1	P44M1	P43M1	P42M1	P41M1	P40M1		1111,1111				
P4M0	P4 口配置寄存器 0	B4H	P47M0	P46M0	P45M0	P44M0	P43M0	P42M0	P41M0	P40M0		0000,0000				
IP2	中断优先级控制寄存器 2	B5H	PPWM2FD	PI2C	PCMP	PX4	PPWM0FD	PPWM0	PSPI	PS2		0000,0000				
IP2H	高中断优先级控制寄存器 2	B6H	PPWM2FDH	PI2CH	PCMHP	PX4H	PPWM0FDH	PPWM0H	PSPIH	PS2H		0000,0000				
IPH	高中断优先级控制寄存器	B7H	PPCAH	PLVDH	PADC	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	-	I2C_S[1:0]	CMPO_S	S4_S	S3_S	S2_S		0x00,0000					
ADC_CONTR	ADC 控制寄存器	BCH	ADC_POWER	ADC_START	ADC_FLAG	ADC_EPWMT	ADC_CHS[3:0]					0000,0000				
ADC_RES	ADC 转换结果高位寄存器	BDH										0000,0000				
ADC_RESL	ADC 转换结果低位寄存器	BEH										0000,0000				
P4	P4 端口	C0H	P47	P46	P45	P44	P43	P42	P41	P40		1111,1111				
WDT CONTR	看门狗控制寄存器	C1H	WDT_FLAG	-	EN_WDT	CLR_WDT	IDL_WDT	WDT_PS[2:0]				0xn0,nnnn				
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	-	-	-	-		0000,xxxx				
P5	P5 端口	C8H	-	-	P55	P54	P53	P52	P51	P50		xx11,1111				
P5M1	P5 口配置寄存器 1	C9H	-	-	P55M1	P54M1	P53M1	P52M1	P51M1	P50M1		xx11,1111				
P5M0	P5 口配置寄存器 0	CAH	-	-	P55M0	P54M0	P53M0	P52M0	P51M0	P50M0		xx00,0000				
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	F1	P		0000,0000				
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,x000				
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	CCAP0	CCAPN0	MAT0	TOGO	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
ADCCFG	ADC 配置寄存器	DEH	-	-	RESFMT	-	SPEED[3:0]				xx0x,0000
IP3	中断优先级控制寄存器 3	DFH	PPWM4FD	PPWM5	PPWM4	PPWM3	PPWM2	PPWM1	PS4	PS3	0000,0000
ACC	累加器	E0H									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
CL	PCA 计数器低字节	E9H									0000,0000
CCAPOL	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
IP3H	高中断优先级控制寄存器 3	EEH	PPWM4FDH	PPWM5H	PPWM4H	PPWM3H	PPWM2H	PPWM1H	PS4H	PS3H	0000,0000
AUXINTIF	扩展外部中断标志寄存器	EFH	-	INT4IF	INT3IF	INT2IF	-	-	-	T2IF	x000,xxx0
B	B 寄存器	F0H									0000,0000
PWMSET	增强型 PWM 全局配置	F1H	ENGLBSET	PWMRST	ENPWM5	ENPWM4	ENPWM3	ENPWM2	ENPWM1	ENPWM0	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
IAP_TPS	IAP 等待时间控制寄存器	F5H	-	-			IAPTPS[5:0]				xx00,0000
PWMCFG01	增强型 PWM 配置寄存器	F6H	PWM1CBIF	EPWM1CBI	FLTPS0	PWM1CEN	PWM0CBIF	EPWM0CBI	ENPWM0TA	PWMOCEN	0000,0000
PWMCFG23	增强型 PWM 配置寄存器	F7H	PWM3CBIF	EPWM3CBI	FLTPS1	PWM3CEN	PWM2CBIF	EPWM2CBI	ENPWM2TA	PWM2CEN	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
PWMCFG45	增强型 PWM 配置寄存器	FEH	PWM5CBIF	EPWM5CBI	FLTPS2	PWM5CEN	PWM4CBIF	EPWM4CBI	ENPWM4TA	PWM4CEN	0000,0000
RSTCFG	复位配置寄存器	FFH	-	ENLVR	-	P54RST	-	-		LVDS[1:0]	xnxn,xxnn

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 @DPTR,A 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	
CLKSEL	时钟选择寄存器	FE00H	-	-	-	-	-	-	-	MCKSEL[1:0]	xxxx,xx00
CLKDIV	时钟分频寄存器	FE01H									nnnn,nnnn
HIRCCR	内部高速振荡器控制寄存器	FE02H	ENHIRC	-	-	-	-	-	-	HIRCST	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
MCLKOCR	主时钟输出控制寄存器	FE05H	MCLKO_S								0000,0000
IRCDB	内部高速振荡器去抖控制	FE06H									1000,0000
POPU	PO 口上拉电阻控制寄存器	FE10H	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU	0000,0000

P1PU	P1 口上拉电阻控制寄存器	FE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU	0000,0000	
P2PU	P2 口上拉电阻控制寄存器	FE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU	0000,0000	
P3PU	P3 口上拉电阻控制寄存器	FE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU	0000,0000	
P4PU	P4 口上拉电阻控制寄存器	FE14H	P47PU	P46PU	P45PU	P44PU	P43PU	P42PU	P41PU	P40PU	0000,0000	
P5PU	P5 口上拉电阻控制寄存器	FE15H	-	-	P55PU	P54PU	P53PU	P52PU	P51PU	P50PU	xx00,0000	
P0NCS	P0 口施密特触发控制寄存器	FE18H	P07NCS	P06NCS	P05NCS	P04NCS	P03NCS	P02NCS	P01NCS	P00NCS	0000,0000	
P1NCS	P1 口施密特触发控制寄存器	FE19H	P17NCS	P16NCS	P15NCS	P14NCS	P13NCS	P12NCS	P11NCS	P10NCS	0000,0000	
P2NCS	P2 口施密特触发控制寄存器	FE1AH	P27NCS	P26NCS	P25NCS	P24NCS	P23NCS	P22NCS	P21NCS	P20NCS	0000,0000	
P3NCS	P3 口施密特触发控制寄存器	FE1BH	P37NCS	P36NCS	P35NCS	P34NCS	P33NCS	P32NCS	P31NCS	P30NCS	0000,0000	
P4NCS	P4 口施密特触发控制寄存器	FE1CH	P47NCS	P46NCS	P45NCS	P44NCS	P43NCS	P42NCS	P41NCS	P40NCS	0000,0000	
P5NCS	P5 口施密特触发控制寄存器	FE1DH	-	-	P55NCS	P54NCS	P53NCS	P52NCS	P51NCS	P50NCS	xx00,0000	
P0SR	P0 口电平转换速率寄存器	FE20H	P07SR	P06SR	P05SR	P04SR	P03SR	P02SR	P01SR	P00SR	1111,1111	
P1SR	P1 口电平转换速率寄存器	FE21H	P17SR	P16SR	P15SR	P14SR	P13SR	P12SR	P11SR	P10SR	1111,1111	
P2SR	P2 口电平转换速率寄存器	FE22H	P27SR	P26SR	P25SR	P24SR	P23SR	P22SR	P21SR	P20SR	1111,1111	
P3SR	P3 口电平转换速率寄存器	FE23H	P37SR	P36SR	P35SR	P34SR	P33SR	P32SR	P31SR	P30SR	1111,1111	
P4SR	P4 口电平转换速率寄存器	FE24H	P47SR	P46SR	P45SR	P44SR	P43SR	P42SR	P41SR	P40SR	1111,1111	
P5SR	P5 口电平转换速率寄存器	FE25H	-	-	P55SR	P54SR	P53SR	P52SR	P51SR	P50SR	xx11,1111	
P0DR	P0 口驱动电流控制寄存器	FE28H	P07DR	P06DR	P05DR	P04DR	P03DR	P02DR	P01DR	P00DR	1111,1111	
P1DR	P1 口驱动电流控制寄存器	FE29H	P17DR	P16DR	P15DR	P14DR	P13DR	P12DR	P11DR	P10DR	1111,1111	
P2DR	P2 口驱动电流控制寄存器	FE2AH	P27DR	P26DR	P25DR	P24DR	P23DR	P22DR	P21DR	P20DR	1111,1111	
P3DR	P3 口驱动电流控制寄存器	FE2BH	P37DR	P36DR	P35DR	P34DR	P33DR	P32DR	P31DR	P30DR	1111,1111	
P4DR	P4 口驱动电流控制寄存器	FE2CH	P47DR	P46DR	P45DR	P44DR	P43DR	P42DR	P41DR	P40DR	1111,1111	
P5DR	P5 口驱动电流控制寄存器	FE2DH	-	-	P55DR	P54DR	P53DR	P52DR	P51DR	P50DR	xx00,0000	
P0IE	P0 口输入使能控制寄存器	FE30H	P07IE	P06IE	P05IE	P04IE	P03IE	P02IE	P11IE	P00IE	1111,1111	
P1IE	P1 口输入使能控制寄存器	FE31H	P17IE	P16IE	P15IE	P14IE	P13IE	P12IE	P11IE	P10IE	1111,1111	
P3IE	P3 口输入使能控制寄存器	FE33H	P37IE	P36IE	P35IE	P34IE	P33IE	P32IE	P31IE	P30IE	1111,1111	
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	0xxx,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	I2CSLADR[7:1]							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	
TM2PS	定时器 2 时钟预分频寄存器	FEA2H									0000,0000	
TM3PS	定时器 3 时钟预分频寄存器	FEA3H									0000,0000	
TM4PS	定时器 4 时钟预分频寄存器	FEA4H									0000,0000	
ADCTIM	ADC 时序控制寄存器	FEA8H	CSSETUP	CSHOLD[1:0]		SMPDUTY[4:0]					0010,1010	
PWM0CH	PWM0 计数器高字节	FF00H	-									x000,0000
PWM0CL	PWM0 计数器低字节	FF01H									0000,0000	
PWM0CKS	PWM0 时钟选择	FF02H	-	-	-	SELT2	PWM_PS[3:0]				xxx0,0000	
PWM0TADCH	PWM0 触发 ADC 计数高字节	FF03H	-									x000,0000
PWM0TADCL	PWM0 触发 ADC 计数低字节	FF04H									0000,0000	

PWM0IF	PWM0 中断标志寄存器	FF05H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWM0FDCR	PWM0 异常检测控制寄存器	FF06H	INVCMP	INVIO	ENFD	FLTFLO	EFDI	FDCMP	FDIO	FDIF	0000,0000
PWM00T1H	PWM00T1 计数值高字节	FF10H	-								x000,0000
PWM00T1L	PWM00T1 计数值低字节	FF11H									0000,0000
PWM00T2H	PWM00T2 计数值高字节	FF12H	-								x000,0000
PWM00T2L	PWM00T2 计数值低字节	FF13H									0000,0000
PWM00CR	PWM00 控制寄存器	FF14H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM00HLD	PWM00 电平保持控制寄存器	FF15H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM01T1H	PWM01T1 计数值高字节	FF18H	-								x000,0000
PWM01T1L	PWM01T1 计数值低字节	FF19H									0000,0000
PWM01T2H	PWM01T2 计数值高字节	FF1AH	-								x000,0000
PWM01T2L	PWM01T2 计数值低字节	FF1BH									0000,0000
PWM01CR	PWM01 控制寄存器	FF1CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM01HLD	PWM01 电平保持控制寄存器	FF1DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM02T1H	PWM02T1 计数值高字节	FF20H	-								x000,0000
PWM02T1L	PWM02T1 计数值低字节	FF21H									0000,0000
PWM02T2H	PWM02T2 计数值高字节	FF22H	-								x000,0000
PWM02T2L	PWM02T2 计数值低字节	FF23H									0000,0000
PWM02CR	PWM02 控制寄存器	FF24H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM02HLD	PWM02 电平保持控制寄存器	FF25H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM03T1H	PWM03T1 计数值高字节	FF28H	-								x000,0000
PWM03T1L	PWM03T1 计数值低字节	FF29H									0000,0000
PWM03T2H	PWM03T2 计数值高字节	FF2AH	-								x000,0000
PWM03T2L	PWM03T2 计数值低字节	FF2BH									0000,0000
PWM03CR	PWM03 控制寄存器	FF2CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM03HLD	PWM03 电平保持控制寄存器	FF2DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM04T1H	PWM04T1 计数值高字节	FF30H	-								x000,0000
PWM04T1L	PWM04T1 计数值低字节	FF31H									0000,0000
PWM04T2H	PWM04T2 计数值高字节	FF32H	-								x000,0000
PWM04T2L	PWM04T2 计数值低字节	FF33H									0000,0000
PWM04CR	PWM04 控制寄存器	FF34H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM04HLD	PWM04 电平保持控制寄存器	FF35H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM05T1H	PWM05T1 计数值高字节	FF38H	-								x000,0000
PWM05T1L	PWM05T1 计数值低字节	FF39H									0000,0000
PWM05T2H	PWM05T2 计数值高字节	FF3AH	-								x000,0000
PWM05T2L	PWM05T2 计数值低字节	FF3BH									0000,0000
PWM05CR	PWM05 控制寄存器	FF3CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM05HLD	PWM05 电平保持控制寄存器	FF3DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM06T1H	PWM06T1 计数值高字节	FF40H	-								x000,0000
PWM06T1L	PWM06T1 计数值低字节	FF41H									0000,0000
PWM06T2H	PWM06T2 计数值高字节	FF42H	-								x000,0000
PWM06T2L	PWM06T2 计数值低字节	FF43H									0000,0000
PWM06CR	PWM06 控制寄存器	FF44H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM06HLD	PWM06 电平保持控制寄存器	FF45H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00

PWM07T1H	PWM07T1 计数值高字节	FF48H	-								x000,0000
PWM07T1L	PWM07T1 计数值低字节	FF49H									0000,0000
PWM07T2H	PWM07T2 计数值高字节	FF4AH	-								x000,0000
PWM07T2L	PWM07T2 计数值低字节	FF4BH									0000,0000
PWM07CR	PWM07 控制寄存器	FF4CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM07HLD	PWM07 电平保持控制寄存器	FF4DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM1CH	PWM1 计数器高字节	FF50H	-								x000,0000
PWM1CL	PWM1 计数器低字节	FF51H									0000,0000
PWM1CKS	PWM1 时钟选择	FF52H	-	-	-	SELT2	PWM_PS[3:0]				xxx0,0000
PWM1IF	PWM1 中断标志寄存器	FF55H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWM1FDCR	PWM1 异常检测控制寄存器	FF56H	INVCMP	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF	0000,0000
PWM10T1H	PWM10T1 计数值高字节	FF60H	-								x000,0000
PWM10T1L	PWM10T1 计数值低字节	FF61H									0000,0000
PWM10T2H	PWM10T2 计数值高字节	FF62H	-								x000,0000
PWM10T2L	PWM10T2 计数值低字节	FF63H									0000,0000
PWM10CR	PWM10 控制寄存器	FF64H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM10HLD	PWM10 电平保持控制寄存器	FF65H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM11T1H	PWM11T1 计数值高字节	FF68H	-								x000,0000
PWM11T1L	PWM11T1 计数值低字节	FF69H									0000,0000
PWM11T2H	PWM11T2 计数值高字节	FF6AH	-								x000,0000
PWM11T2L	PWM11T2 计数值低字节	FF6BH									0000,0000
PWM11CR	PWM11 控制寄存器	FF6CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM11HLD	PWM11 电平保持控制寄存器	FF6DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM12T1H	PWM12T1 计数值高字节	FF70H	-								x000,0000
PWM12T1L	PWM12T1 计数值低字节	FF71H									0000,0000
PWM12T2H	PWM12T2 计数值高字节	FF72H	-								x000,0000
PWM12T2L	PWM12T2 计数值低字节	FF73H									0000,0000
PWM12CR	PWM12 控制寄存器	FF74H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM12HLD	PWM12 电平保持控制寄存器	FF75H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM13T1H	PWM13T1 计数值高字节	FF78H	-								x000,0000
PWM13T1L	PWM13T1 计数值低字节	FF79H									0000,0000
PWM13T2H	PWM13T2 计数值高字节	FF7AH	-								x000,0000
PWM13T2L	PWM13T2 计数值低字节	FF7BH									0000,0000
PWM13CR	PWM13 控制寄存器	FF7CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM13HLD	PWM13 电平保持控制寄存器	FF7DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM14T1H	PWM14T1 计数值高字节	FF80H	-								x000,0000
PWM14T1L	PWM14T1 计数值低字节	FF81H									0000,0000
PWM14T2H	PWM14T2 计数值高字节	FF82H	-								x000,0000
PWM14T2L	PWM14T2 计数值低字节	FF83H									0000,0000
PWM14CR	PWM14 控制寄存器	FF84H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM14HLD	PWM14 电平保持控制寄存器	FF85H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM15T1H	PWM15T1 计数值高字节	FF88H	-								x000,0000
PWM15T1L	PWM15T1 计数值低字节	FF89H									0000,0000
PWM15T2H	PWM15T2 计数值高字节	FF8AH	-								x000,0000

PWM15T2L	PWM15T2 计数值低字节	FF8BH										0000,0000						
PWM15CR	PWM15 控制寄存器	FF8CH	ENO	INI	-	-		ENI	ENT2I	ENT1I		00xx,x000						
PWM15HLD	PWM15 电平保持控制寄存器	FF8DH	-	-	-	-	-	-	HLDH	HLDL		xxxx,xx00						
PWM16T1H	PWM16T1 计数值高字节	FF90H	-										x000,0000					
PWM16T1L	PWM16T1 计数值低字节	FF91H										0000,0000						
PWM16T2H	PWM16T2 计数值高字节	FF92H	-										x000,0000					
PWM16T2L	PWM16T2 计数值低字节	FF93H										0000,0000						
PWM16CR	PWM16 控制寄存器	FF94H	ENO	INI	-	-		ENI	ENT2I	ENT1I		00xx,x000						
PWM16HLD	PWM16 电平保持控制寄存器	FF95H	-	-	-	-	-	-	HLDH	HLDL		xxxx,xx00						
PWM17T1H	PWM17T1 计数值高字节	FF98H	-										x000,0000					
PWM17T1L	PWM17T1 计数值低字节	FF99H										0000,0000						
PWM17T2H	PWM17T2 计数值高字节	FF9AH	-										x000,0000					
PWM17T2L	PWM17T2 计数值低字节	FF9BH										0000,0000						
PWM17CR	PWM17 控制寄存器	FF9CH	ENO	INI	-	-		ENI	ENT2I	ENT1I		00xx,x000						
PWM17HLD	PWM17 电平保持控制寄存器	FF9DH	-	-	-	-	-	-	HLDH	HLDL		xxxx,xx00						
PWM2CH	PWM2 计数器高字节	FFA0H	-										x000,0000					
PWM2CL	PWM2 计数器低字节	FFA1H										0000,0000						
PWM2CKS	PWM2 时钟选择	FFA2H	-	-	-	SELT2	PWM_PS[3:0]					xxx0,0000						
PWM2TADCH	PWM2 触发 ADC 计数高字节	FFA3H	-										x000,0000					
PWM2TADCL	PWM2 触发 ADC 计数低字节	FFA4H										0000,0000						
PWM2IF	PWM2 中断标志寄存器	FFA5H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF		0000,0000						
PWM2FDCR	PWM2 异常检测控制寄存器	FFA6H	INVCMP	INVIO	ENFD	FLTFLO	EFDI	FDCMP	FDIO	FDIF		0000,0000						
PWM20T1H	PWM20T1 计数值高字节	FFB0H	-										x000,0000					
PWM20T1L	PWM20T1 计数值低字节	FFB1H										0000,0000						
PWM20T2H	PWM20T2 计数值高字节	FFB2H	-										x000,0000					
PWM20T2L	PWM20T2 计数值低字节	FFB3H										0000,0000						
PWM20CR	PWM20 控制寄存器	FFB4H	ENO	INI	-	-		ENI	ENT2I	ENT1I		00xx,x000						
PWM20HLD	PWM20 电平保持控制寄存器	FFB5H	-	-	-	-	-	-	HLDH	HLDL		xxxx,xx00						
PWM21T1H	PWM21T1 计数值高字节	FFB8H	-										x000,0000					
PWM21T1L	PWM21T1 计数值低字节	FFB9H										0000,0000						
PWM21T2H	PWM21T2 计数值高字节	FFBAH	-										x000,0000					
PWM21T2L	PWM21T2 计数值低字节	FFBBH										0000,0000						
PWM21CR	PWM21 控制寄存器	FFBCH	ENO	INI	-	-		ENI	ENT2I	ENT1I		00xx,x000						
PWM21HLD	PWM21 电平保持控制寄存器	FFBDH	-	-	-	-	-	-	HLDH	HLDL		xxxx,xx00						
PWM22T1H	PWM22T1 计数值高字节	FFC0H	-										x000,0000					
PWM22T1L	PWM22T1 计数值低字节	FFC1H										0000,0000						
PWM22T2H	PWM22T2 计数值高字节	FFC2H	-										x000,0000					
PWM22T2L	PWM22T2 计数值低字节	FFC3H										0000,0000						
PWM22CR	PWM22 控制寄存器	FFC4H	ENO	INI	-	-		ENI	ENT2I	ENT1I		00xx,x000						
PWM22HLD	PWM22 电平保持控制寄存器	FFC5H	-	-	-	-	-	-	HLDH	HLDL		xxxx,xx00						
PWM23T1H	PWM23T1 计数值高字节	FFC8H	-										x000,0000					
PWM23T1L	PWM23T1 计数值低字节	FFC9H										0000,0000						
PWM23T2H	PWM23T2 计数值高字节	FFCAH	-										x000,0000					
PWM23T2L	PWM23T2 计数值低字节	FFCBH										0000,0000						

PWM23CR	PWM23 控制寄存器	FFCCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM23HLD	PWM23 电平保持控制寄存器	FFCDH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM24T1H	PWM24T1 计数值高字节	FFD0H	-								x000,0000
PWM24T1L	PWM24T1 计数值低字节	FFD1H									0000,0000
PWM24T2H	PWM24T2 计数值高字节	FFD2H	-								x000,0000
PWM24T2L	PWM24T2 计数值低字节	FFD3H									0000,0000
PWM24CR	PWM24 控制寄存器	FFD4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM24HLD	PWM24 电平保持控制寄存器	FFD5H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM25T1H	PWM25T1 计数值高字节	FFD8H	-								x000,0000
PWM25T1L	PWM25T1 计数值低字节	FFD9H									0000,0000
PWM25T2H	PWM25T2 计数值高字节	FFDAH	-								x000,0000
PWM25T2L	PWM25T2 计数值低字节	FFDBH									0000,0000
PWM25CR	PWM25 控制寄存器	FFDCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM25HLD	PWM25 电平保持控制寄存器	FFDDH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM26T1H	PWM26T1 计数值高字节	FFE0H	-								x000,0000
PWM26T1L	PWM26T1 计数值低字节	FFE1H									0000,0000
PWM26T2H	PWM26T2 计数值高字节	FFE2H	-								x000,0000
PWM26T2L	PWM26T2 计数值低字节	FFE3H									0000,0000
PWM26CR	PWM26 控制寄存器	FFE4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM26HLD	PWM26 电平保持控制寄存器	FFE5H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM27T1H	PWM27T1 计数值高字节	FFE8H	-								x000,0000
PWM27T1L	PWM27T1 计数值低字节	FFE9H									0000,0000
PWM27T2H	PWM27T2 计数值高字节	FFEAH	-								x000,0000
PWM27T2L	PWM27T2 计数值低字节	FFEBH									0000,0000
PWM27CR	PWM27 控制寄存器	FFECH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM27HLD	PWM27 电平保持控制寄存器	FFEDH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM3CH	PWM3 计数器高字节	FC00H	-								x000,0000
PWM3CL	PWM3 计数器低字节	FC01H									0000,0000
PWM3CKS	PWM3 时钟选择	FC02H	-	-	-	SELT2					xxx0,0000
PWM3IF	PWM3 中断标志寄存器	FC05H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWM3FDCR	PWM3 异常检测控制寄存器	FC06H	INVCMP	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF	0000,0000
PWM30T1H	PWM30T1 计数值高字节	FC10H	-								x000,0000
PWM30T1L	PWM30T1 计数值低字节	FC11H									0000,0000
PWM30T2H	PWM30T2 计数值高字节	FC12H	-								x000,0000
PWM30T2L	PWM30T2 计数值低字节	FC13H									0000,0000
PWM30CR	PWM30 控制寄存器	FC14H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM30HLD	PWM30 电平保持控制寄存器	FC15H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM31T1H	PWM31T1 计数值高字节	FC18H	-								x000,0000
PWM31T1L	PWM31T1 计数值低字节	FC19H									0000,0000
PWM31T2H	PWM31T2 计数值高字节	FC1AH	-								x000,0000
PWM31T2L	PWM31T2 计数值低字节	FC1BH									0000,0000
PWM31CR	PWM31 控制寄存器	FC1CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM31HLD	PWM31 电平保持控制寄存器	FC1DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM32T1H	PWM32T1 计数值高字节	FC20H	-								x000,0000

PWM32T1L	PWM32T1 计数值低字节	FC21H									0000,0000					
PWM32T2H	PWM32T2 计数值高字节	FC22H	-									x000,0000				
PWM32T2L	PWM32T2 计数值低字节	FC23H									0000,0000					
PWM32CR	PWM32 控制寄存器	FC24H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000					
PWM32HLD	PWM32 电平保持控制寄存器	FC25H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00					
PWM33T1H	PWM33T1 计数值高字节	FC28H	-									x000,0000				
PWM33T1L	PWM33T1 计数值低字节	FC29H									0000,0000					
PWM33T2H	PWM33T2 计数值高字节	FC2AH	-									x000,0000				
PWM33T2L	PWM33T2 计数值低字节	FC2BH									0000,0000					
PWM33CR	PWM33 控制寄存器	FC2CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000					
PWM33HLD	PWM33 电平保持控制寄存器	FC2DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00					
PWM34T1H	PWM34T1 计数值高字节	FC30H	-									x000,0000				
PWM34T1L	PWM34T1 计数值低字节	FC31H									0000,0000					
PWM34T2H	PWM34T2 计数值高字节	FC32H	-									x000,0000				
PWM34T2L	PWM34T2 计数值低字节	FC33H									0000,0000					
PWM34CR	PWM34 控制寄存器	FC34H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000					
PWM34HLD	PWM34 电平保持控制寄存器	FC35H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00					
PWM35T1H	PWM35T1 计数值高字节	FC38H	-									x000,0000				
PWM35T1L	PWM35T1 计数值低字节	FC39H									0000,0000					
PWM35T2H	PWM35T2 计数值高字节	FC3AH	-									x000,0000				
PWM35T2L	PWM35T2 计数值低字节	FC3BH									0000,0000					
PWM35CR	PWM35 控制寄存器	FC3CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000					
PWM35HLD	PWM35 电平保持控制寄存器	FC3DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00					
PWM36T1H	PWM36T1 计数值高字节	FC40H	-									x000,0000				
PWM36T1L	PWM36T1 计数值低字节	FC41H									0000,0000					
PWM36T2H	PWM36T2 计数值高字节	FC42H	-									x000,0000				
PWM36T2L	PWM36T2 计数值低字节	FC43H									0000,0000					
PWM36CR	PWM36 控制寄存器	FC44H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000					
PWM36HLD	PWM36 电平保持控制寄存器	FC45H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00					
PWM37T1H	PWM37T1 计数值高字节	FC48H	-									x000,0000				
PWM37T1L	PWM37T1 计数值低字节	FC49H									0000,0000					
PWM37T2H	PWM37T2 计数值高字节	FC4AH	-									x000,0000				
PWM37T2L	PWM37T2 计数值低字节	FC4BH									0000,0000					
PWM37CR	PWM37 控制寄存器	FC4CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000					
PWM37HLD	PWM37 电平保持控制寄存器	FC4DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00					
PWM4CH	PWM4 计数器高字节	FC50H	-									x000,0000				
PWM4CL	PWM4 计数器低字节	FC51H									0000,0000					
PWM4CKS	PWM4 时钟选择	FC52H	-	-	-	SELT2	PWM_PS[3:0]				xxx0,0000					
PWM4TADCH	PWM4 触发 ADC 计数高字节	FC53H	-									x000,0000				
PWM4TADCL	PWM4 触发 ADC 计数低字节	FC54H									0000,0000					
PWM4IF	PWM4 中断标志寄存器	FC55H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	COIF	0000,0000					
PWM4FDCR	PWM4 异常检测控制寄存器	FC56H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	0000,0000					
PWM40T1H	PWM40T1 计数值高字节	FC60H	-									x000,0000				
PWM40T1L	PWM40T1 计数值低字节	FC61H									0000,0000					

PWM40T2H	PWM40T2 计数值高字节	FC62H	-								x000,0000
PWM40T2L	PWM40T2 计数值低字节	FC63H									0000,0000
PWM40CR	PWM40 控制寄存器	FC64H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM40HLD	PWM40 电平保持控制寄存器	FC65H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM41T1H	PWM41T1 计数值高字节	FC68H	-								x000,0000
PWM41T1L	PWM41T1 计数值低字节	FC69H									0000,0000
PWM41T2H	PWM41T2 计数值高字节	FC6AH	-								x000,0000
PWM41T2L	PWM41T2 计数值低字节	FC6BH									0000,0000
PWM41CR	PWM41 控制寄存器	FC6CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM41HLD	PWM41 电平保持控制寄存器	FC6DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM42T1H	PWM42T1 计数值高字节	FC70H	-								x000,0000
PWM42T1L	PWM42T1 计数值低字节	FC71H									0000,0000
PWM42T2H	PWM42T2 计数值高字节	FC72H	-								x000,0000
PWM42T2L	PWM42T2 计数值低字节	FC73H									0000,0000
PWM42CR	PWM42 控制寄存器	FC74H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM42HLD	PWM42 电平保持控制寄存器	FC75H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM43T1H	PWM43T1 计数值高字节	FC78H	-								x000,0000
PWM43T1L	PWM43T1 计数值低字节	FC79H									0000,0000
PWM43T2H	PWM43T2 计数值高字节	FC7AH	-								x000,0000
PWM43T2L	PWM43T2 计数值低字节	FC7BH									0000,0000
PWM43CR	PWM43 控制寄存器	FC7CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM43HLD	PWM43 电平保持控制寄存器	FC7DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM44T1H	PWM44T1 计数值高字节	FC80H	-								x000,0000
PWM44T1L	PWM44T1 计数值低字节	FC81H									0000,0000
PWM44T2H	PWM44T2 计数值高字节	FC82H	-								x000,0000
PWM44T2L	PWM44T2 计数值低字节	FC83H									0000,0000
PWM44CR	PWM44 控制寄存器	FC84H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM44HLD	PWM44 电平保持控制寄存器	FC85H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM45T1H	PWM45T1 计数值高字节	FC88H	-								x000,0000
PWM45T1L	PWM45T1 计数值低字节	FC89H									0000,0000
PWM45T2H	PWM45T2 计数值高字节	FC8AH	-								x000,0000
PWM45T2L	PWM45T2 计数值低字节	FC8BH									0000,0000
PWM45CR	PWM45 控制寄存器	FC8CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM45HLD	PWM45 电平保持控制寄存器	FC8DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM46T1H	PWM46T1 计数值高字节	FC90H	-								x000,0000
PWM46T1L	PWM46T1 计数值低字节	FC91H									0000,0000
PWM46T2H	PWM46T2 计数值高字节	FC92H	-								x000,0000
PWM46T2L	PWM46T2 计数值低字节	FC93H									0000,0000
PWM46CR	PWM46 控制寄存器	FC94H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM46HLD	PWM46 电平保持控制寄存器	FC95H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM47T1H	PWM47T1 计数值高字节	FC98H	-								x000,0000
PWM47T1L	PWM47T1 计数值低字节	FC99H									0000,0000
PWM47T2H	PWM47T2 计数值高字节	FC9AH	-								x000,0000
PWM47T2L	PWM47T2 计数值低字节	FC9BH									0000,0000

PWM47CR	PWM47 控制寄存器	FC9CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM47HLD	PWM47 电平保持控制寄存器	FC9DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM5CH	PWM5 计数器高字节	FCA0H	-								x000,0000
PWM5CL	PWM5 计数器低字节	FCA1H									0000,0000
PWM5CKS	PWM5 时钟选择	FCA2H	-	-	-	SELT2			PWM_PS[3:0]		xxx0,0000
PWM5IF	PWM5 中断标志寄存器	FCA5H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	COIF	0000,0000
PWM5FDCR	PWM5 异常检测控制寄存器	FCA6H	INVCMP	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF	0000,0000
PWM50T1H	PWM50T1 计数值高字节	FCB0H	-								x000,0000
PWM50T1L	PWM50T1 计数值低字节	FCB1H									0000,0000
PWM50T2H	PWM50T2 计数值高字节	FCB2H	-								x000,0000
PWM50T2L	PWM50T2 计数值低字节	FCB3H									0000,0000
PWM50CR	PWM50 控制寄存器	FCB4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM50HLD	PWM50 电平保持控制寄存器	FCB5H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM51T1H	PWM51T1 计数值高字节	FCB8H	-								x000,0000
PWM51T1L	PWM51T1 计数值低字节	FCB9H									0000,0000
PWM51T2H	PWM51T2 计数值高字节	FCBAH	-								x000,0000
PWM51T2L	PWM51T2 计数值低字节	FCBBH									0000,0000
PWM51CR	PWM51 控制寄存器	FCBCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM51HLD	PWM51 电平保持控制寄存器	FCBDH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM52T1H	PWM52T1 计数值高字节	FCC0H	-								x000,0000
PWM52T1L	PWM52T1 计数值低字节	FCC1H									0000,0000
PWM52T2H	PWM52T2 计数值高字节	FCC2H	-								x000,0000
PWM52T2L	PWM52T2 计数值低字节	FCC3H									0000,0000
PWM52CR	PWM52 控制寄存器	FCC4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM52HLD	PWM52 电平保持控制寄存器	FCC5H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM53T1H	PWM53T1 计数值高字节	FCC8H	-								x000,0000
PWM53T1L	PWM53T1 计数值低字节	FCC9H									0000,0000
PWM53T2H	PWM53T2 计数值高字节	FCCAH	-								x000,0000
PWM53T2L	PWM53T2 计数值低字节	FCCBH									0000,0000
PWM53CR	PWM53 控制寄存器	FCCCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM53HLD	PWM53 电平保持控制寄存器	FCCDH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM54T1H	PWM54T1 计数值高字节	FCD0H	-								x000,0000
PWM54T1L	PWM54T1 计数值低字节	FCD1H									0000,0000
PWM54T2H	PWM54T2 计数值高字节	FCD2H	-								x000,0000
PWM54T2L	PWM54T2 计数值低字节	FCD3H									0000,0000
PWM54CR	PWM54 控制寄存器	FCD4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM54HLD	PWM54 电平保持控制寄存器	FCD5H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM55T1H	PWM55T1 计数值高字节	FCD8H	-								x000,0000
PWM55T1L	PWM55T1 计数值低字节	FCD9H									0000,0000
PWM55T2H	PWM55T2 计数值高字节	FCDAH	-								x000,0000
PWM55T2L	PWM55T2 计数值低字节	FCDBH									0000,0000
PWM55CR	PWM55 控制寄存器	FCDCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM55HLD	PWM55 电平保持控制寄存器	FCDDH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM56T1H	PWM56T1 计数值高字节	FCE0H	-								x000,0000

PWM56T1L	PWM56T1 计数值低字节	FCE1H									0000,0000				
PWM56T2H	PWM56T2 计数值高字节	FCE2H	-									x000,0000			
PWM56T2L	PWM56T2 计数值低字节	FCE3H									0000,0000				
PWM56CR	PWM56 控制寄存器	FCE4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000				
PWM56HLD	PWM56 电平保持控制寄存器	FCE5H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00				
PWM57T1H	PWM57T1 计数值高字节	FCE8H	-									x000,0000			
PWM57T1L	PWM57T1 计数值低字节	FCE9H									0000,0000				
PWM57T2H	PWM57T2 计数值高字节	FCEAH	-									x000,0000			
PWM57T2L	PWM57T2 计数值低字节	FCEBH									0000,0000				
PWM57CR	PWM57 控制寄存器	FCECH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000				
PWM57HLD	PWM57 电平保持控制寄存器	FCEDH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00				
MD3	MDU 数据寄存器	FCF0H	MD3[7:0]								0000,0000				
MD2	MDU 数据寄存器	FCF1H	MD2[7:0]								0000,0000				
MD1	MDU 数据寄存器	FCF2H	MD1[7:0]								0000,0000				
MD0	MDU 数据寄存器	FCF3H	MD0[7:0]								0000,0000				
MD5	MDU 数据寄存器	FCF4H	MD5[7:0]								0000,0000				
MD4	MDU 数据寄存器	FCF5H	MD4[7:0]								0000,0000				
ARCON	MDU 模式控制寄存器	FCF6H	MODE[2:0]			SC[4:0]						0000,0000			
OPCON	MDU 操作控制寄存器	FCF7H	-	MDOV	-	-	-	-	RST	ENOP	x0xx,xx00				
COMEN	COM 使能寄存器	FB00H	C7EN	C6EN	C5EN	C4EN	C3EN	C2EN	C1EN	C0EN	0000,0000				
SEGENL	SEG 使能寄存器	FB01H	S7EN	S6EN	S5EN	S4EN	S3EN	S2EN	S1EN	S0EN	0000,0000				
SEGENH	SEG 使能寄存器	FB02H	S15EN	S14EN	S13EN	S12EN	S11EN	S10EN	S9EN	S8EN	0000,0000				
LEDCTRL	LED 控制寄存器	FB03H	LEDON	-	LEDMODE[1:0]	-	LEDDUTY[2:0]				0x00,x000				
LEDCKS	LED 时钟分频寄存器	FB04H									0000,0001				
COM0_DA_L	共阳模式显示数据	FB10H									0000,0000				
COM1_DA_L	共阳模式显示数据	FB11H									0000,0000				
COM2_DA_L	共阳模式显示数据	FB12H									0000,0000				
COM3_DA_L	共阳模式显示数据	FB13H									0000,0000				
COM4_DA_L	共阳模式显示数据	FB14H									0000,0000				
COM5_DA_L	共阳模式显示数据	FB15H									0000,0000				
COM6_DA_L	共阳模式显示数据	FB16H									0000,0000				
COM7_DA_L	共阳模式显示数据	FB17H									0000,0000				
COM0_DA_H	共阳模式显示数据	FB18H									0000,0000				
COM1_DA_H	共阳模式显示数据	FB19H									0000,0000				
COM2_DA_H	共阳模式显示数据	FB1AH									0000,0000				
COM3_DA_H	共阳模式显示数据	FB1BH									0000,0000				
COM4_DA_H	共阳模式显示数据	FB1CH									0000,0000				
COM5_DA_H	共阳模式显示数据	FB1DH									0000,0000				
COM6_DA_H	共阳模式显示数据	FB1EH									0000,0000				
COM7_DA_H	共阳模式显示数据	FB1FH									0000,0000				
COM0_DC_L	共阴模式显示数据	FB20H									0000,0000				
COM1_DC_L	共阴模式显示数据	FB21H									0000,0000				
COM2_DC_L	共阴模式显示数据	FB22H									0000,0000				
COM3_DC_L	共阴模式显示数据	FB23H									0000,0000				

COM4_DC_L	共阴模式显示数据	FB24H									0000,0000								
COM5_DC_L	共阴模式显示数据	FB25H									0000,0000								
COM6_DC_L	共阴模式显示数据	FB26H									0000,0000								
COM7_DC_L	共阴模式显示数据	FB27H									0000,0000								
COM0_DC_H	共阴模式显示数据	FB28H									0000,0000								
COM1_DC_H	共阴模式显示数据	FB29H									0000,0000								
COM2_DC_H	共阴模式显示数据	FB2AH									0000,0000								
COM3_DC_H	共阴模式显示数据	FB2BH									0000,0000								
COM4_DC_H	共阴模式显示数据	FB2CH									0000,0000								
COM5_DC_H	共阴模式显示数据	FB2DH									0000,0000								
COM6_DC_H	共阴模式显示数据	FB2EH									0000,0000								
COM7_DC_H	共阴模式显示数据	FB2FH									0000,0000								
TSCHEN1	触摸按键使能寄存器 1	FB40H	TKEN7	TKEN6	TKEN5	TKEN4	TKEN3	TKEN2	TKEN1	TKEN0	0000,0000								
TSCHEN2	触摸按键使能寄存器 2	FB41H	TKEN15	TKEN14	TKEN13	TKEN12	TKEN11	TKEN10	TKEN9	TKEN8	0000,0000								
TSCFG1	触摸按键配置寄存器 1	FB42H	-	SCR[2:0]			-	DT[2:0]			x000,0000								
TSCFG2	触摸按键配置寄存器 2	FB43H	-	-	-	-	-	-	TSVR[1:0]		xxxx,xx00								
TSWUTC	触摸按键唤醒控制寄存器	FB44H										0000,0001							
TSCTRL	触摸按键控制寄存器	FB45H	TSGO	SINGLE	TSWAIT	TSWUCS	TSDCEN	TSWUEN	TSSAMP[1:0]		0000,0000								
TSSTA1	触摸按键状态寄存器 1	FB46H	LEDWK	-	-	-	TSWKCHN[3:0]					0xxx,0000							
TSSTA2	触摸按键状态寄存器 2	FB47H	TSIF	TSDOV	-	-	TSDNCHN[3:0]					00xx,0000							
TSRT	触摸按键时间控制寄存器	FB48H										0000,0001							
TSDATH	触摸按键数据高字节	FB49H										0000,0000							
TSDATL	触摸按键数据低字节	FB4AH										0000,0000							
TSTH00H	触摸按键 0 门槛值高字节	FB50H										0000,0000							
TSTH00L	触摸按键 0 门槛值低字节	FB51H										0000,0000							
TSTH01H	触摸按键 1 门槛值高字节	FB52H										0000,0000							
TSTH01L	触摸按键 1 门槛值低字节	FB53H										0000,0000							
TSTH02H	触摸按键 2 门槛值高字节	FB54H										0000,0000							
TSTH02L	触摸按键 2 门槛值低字节	FB55H										0000,0000							
TSTH03H	触摸按键 3 门槛值高字节	FB56H										0000,0000							
TSTH03L	触摸按键 3 门槛值低字节	FB57H										0000,0000							
TSTH04H	触摸按键 4 门槛值高字节	FB58H										0000,0000							
TSTH04L	触摸按键 4 门槛值低字节	FB59H										0000,0000							
TSTH05H	触摸按键 5 门槛值高字节	FB5AH										0000,0000							
TSTH05L	触摸按键 5 门槛值低字节	FB5BH										0000,0000							
TSTH06H	触摸按键 6 门槛值高字节	FB5CH										0000,0000							
TSTH06L	触摸按键 6 门槛值低字节	FB5DH										0000,0000							
TSTH07H	触摸按键 7 门槛值高字节	FB5EH										0000,0000							
TSTH07L	触摸按键 7 门槛值低字节	FB5FH										0000,0000							
TSTH08H	触摸按键 8 门槛值高字节	FB60H										0000,0000							
TSTH08L	触摸按键 8 门槛值低字节	FB61H										0000,0000							
TSTH09H	触摸按键 9 门槛值高字节	FB62H										0000,0000							
TSTH09L	触摸按键 9 门槛值低字节	FB63H										0000,0000							
TSTH10H	触摸按键 10 门槛值高字节	FB64H										0000,0000							

TSTH10L	触摸按键 10 门槛值低字节	FB65H		0000,0000
TSTH11H	触摸按键 11 门槛值高字节	FB66H		0000,0000
TSTH11L	触摸按键 11 门槛值低字节	FB67H		0000,0000
TSTH12H	触摸按键 12 门槛值高字节	FB68H		0000,0000
TSTH12L	触摸按键 12 门槛值低字节	FB69H		0000,0000
TSTH13H	触摸按键 13 门槛值高字节	FB6AH		0000,0000
TSTH13L	触摸按键 13 门槛值低字节	FB6BH		0000,0000
TSTH14H	触摸按键 14 门槛值高字节	FB6CH		0000,0000
TSTH14L	触摸按键 14 门槛值低字节	FB6DH		0000,0000
TSTH15H	触摸按键 15 门槛值高字节	FB6EH		0000,0000
TSTH15L	触摸按键 15 门槛值低字节	FB6FH		0000,0000

注: 特殊功能寄存器初始值意义

- 0: 初始值为 0;
- 1: 初始值为 1;
- n: 初始值与 ISP 下载时的硬件选项有关;
- x: 不存在这个位, 初始值不确定

9 I/O 口

产品线	最多 I/O 口数量
STC8G1K08 系列	18
STC8G1K08-8Pin 系列	6
STC8G1K08A 系列	6
STC8G2K64S4 系列	45
STC8G2K64S2 系列	45
STC8G1K08T 系列	16
STC15H2K64S4 系列	42

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。

关于 I/O 的注意事项：

- 1、 P3.0 和 P3.1 口上电后的状态为弱上拉双向口模式
- 2、 除 P3.0 和 P3.1 外，其余所有 IO 口上电后的状态均为高阻输入状态，用户在使用 IO 口前必须先设置 IO 口模式
- 3、 芯片上电时如果不需要使用 USB 进行 ISP 下载，P3.0/P3.1/P3.2 这 3 个 I/O 口不能同时为低电平，否则会进入 USB 下载模式而无法运行用户代码
- 4、 芯片上电时，若 P3.0 和 P3.1 同时为低电平，P3.2 口会短时间由高阻输入状态切换到双向口模式，用以读取 P3.2 口外部状态来判断是否需要进入 USB 下载模式
- 5、 当使用 P5.4 当作复位脚时，这个端口内部的 4K 上拉电阻会一直打开；但 P5.4 做普通 I/O 口时，基于这个 I/O 口与复位脚共享管脚的特殊考量，端口内部的 4K 上拉电阻依然会打开大约 6.5 毫秒时间，再自动关闭（当用户的电路设计需要使用 P5.4 口驱动外部电路时，请务必考虑上电瞬间会有 6.5 毫秒时间的高电平的问题）

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	P07	P06	P05	P04	P03	P02	P01	P00	1111,1111
P1	P1 端口	90H	P17	P16	P15	P14	P13	P12	P11	P10	1111,1111
P2	P2 端口	A0H	P27	P26	P25	P24	P23	P22	P21	P20	1111,1111
P3	P3 端口	B0H	P37	P36	P35	P34	P33	P32	P31	P30	1111,1111
P4	P4 端口	C0H	P47	P46	P45	P44	P43	P42	P41	P40	1111,1111
P5	P5 端口	C8H	-	-	P55	P54	P53	P52	P51	P50	xx11,1111
P0M1	P0 口配置寄存器 1	93H	P07M1	P06M1	P05M1	P04M1	P03M1	P02M1	P01M1	P00M1	1111,1111
P0M0	P0 口配置寄存器 0	94H	P07M0	P06M0	P05M0	P04M0	P03M0	P02M0	P01M0	P00M0	0000,0000

P1M1	P1 口配置寄存器 1	91H	P17M1	P16M1	P15M1	P14M1	P13M1	P12M1	P11M1	P10M1	1111,1111
P1M0	P1 口配置寄存器 0	92H	P17M0	P16M0	P15M0	P14M0	P13M0	P12M0	P11M0	P10M0	0000,0000
P2M1	P2 口配置寄存器 1	95H	P27M1	P26M1	P25M1	P24M1	P23M1	P22M1	P21M1	P20M1	1111,1111
P2M0	P2 口配置寄存器 0	96H	P27M0	P26M0	P25M0	P24M0	P23M0	P22M0	P21M0	P20M0	0000,0000
P3M1	P3 口配置寄存器 1	B1H	P37M1	P36M1	P35M1	P34M1	P33M1	P32M1	P31M1	P30M1	n111,1100
P3M0	P3 口配置寄存器 0	B2H	P37M0	P36M0	P35M0	P34M0	P33M0	P32M0	P31M0	P30M0	n000,0000
P4M1	P4 口配置寄存器 1	B3H	P47M1	P46M1	P45M1	P44M1	P43M1	P42M1	P41M1	P40M1	1111,1111
P4M0	P4 口配置寄存器 0	B4H	P47M0	P46M0	P45M0	P44M0	P43M0	P42M0	P41M0	P40M0	0000,0000
P5M1	P5 口配置寄存器 1	C9H	-	-	P55M1	P54M1	P53M1	P52M1	P51M1	P50M1	xx11,1111
P5M0	P5 口配置寄存器 0	CAH	-	-	P55M0	P54M0	P53M0	P52M0	P51M0	P50M0	xx00,0000

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
POPU	P0 口上拉电阻控制寄存器	FE10H	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU	0000,0000
P1PU	P1 口上拉电阻控制寄存器	FE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU	0000,0000
P2PU	P2 口上拉电阻控制寄存器	FE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU	0000,0000
P3PU	P3 口上拉电阻控制寄存器	FE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU	0000,0000
P4PU	P4 口上拉电阻控制寄存器	FE14H	P47PU	P46PU	P45PU	P44PU	P43PU	P42PU	P41PU	P40PU	0000,0000
P5PU	P5 口上拉电阻控制寄存器	FE15H	-	-	P55PU	P54PU	P53PU	P52PU	P51PU	P50PU	xx00,0000
P0NCS	P0 口施密特触发控制寄存器	FE18H	P07NCS	P06NCS	P05NCS	P04NCS	P03NCS	P02NCS	P01NCS	P00NCS	0000,0000
P1NCS	P1 口施密特触发控制寄存器	FE19H	P17NCS	P16NCS	P15NCS	P14NCS	P13NCS	P12NCS	P11NCS	P10NCS	0000,0000
P2NCS	P2 口施密特触发控制寄存器	FE1AH	P27NCS	P26NCS	P25NCS	P24NCS	P23NCS	P22NCS	P21NCS	P20NCS	0000,0000
P3NCS	P3 口施密特触发控制寄存器	FE1BH	P37NCS	P36NCS	P35NCS	P34NCS	P33NCS	P32NCS	P31NCS	P30NCS	0000,0000
P4NCS	P4 口施密特触发控制寄存器	FE1CH	P47NCS	P46NCS	P45NCS	P44NCS	P43NCS	P42NCS	P41NCS	P40NCS	0000,0000
P5NCS	P5 口施密特触发控制寄存器	FE1DH	-	-	P55NCS	P54NCS	P53NCS	P52NCS	P51NCS	P50NCS	xx00,0000
P0SR	P0 口电平转换速率寄存器	FE20H	P07SR	P06SR	P05SR	P04SR	P03SR	P02SR	P01SR	P00SR	1111,1111
P1SR	P1 口电平转换速率寄存器	FE21H	P17SR	P16SR	P15SR	P14SR	P13SR	P12SR	P11SR	P10SR	1111,1111
P2SR	P2 口电平转换速率寄存器	FE22H	P27SR	P26SR	P25SR	P24SR	P23SR	P22SR	P21SR	P20SR	1111,1111
P3SR	P3 口电平转换速率寄存器	FE23H	P37SR	P36SR	P35SR	P34SR	P33SR	P32SR	P31SR	P30SR	1111,1111
P4SR	P4 口电平转换速率寄存器	FE24H	P47SR	P46SR	P45SR	P44SR	P43SR	P42SR	P41SR	P40SR	1111,1111
P5SR	P5 口电平转换速率寄存器	FE25H	-	-	P55SR	P54SR	P53SR	P52SR	P51SR	P50SR	xx11,1111
P0DR	P0 口驱动电流控制寄存器	FE28H	P07DR	P06DR	P05DR	P04DR	P03DR	P02DR	P01DR	P00DR	1111,1111
P1DR	P1 口驱动电流控制寄存器	FE29H	P17DR	P16DR	P15DR	P14DR	P13DR	P12DR	P11DR	P10DR	1111,1111
P2DR	P2 口驱动电流控制寄存器	FE2AH	P27DR	P26DR	P25DR	P24DR	P23DR	P22DR	P21DR	P20DR	1111,1111
P3DR	P3 口驱动电流控制寄存器	FE2BH	P37DR	P36DR	P35DR	P34DR	P33DR	P32DR	P31DR	P30DR	1111,1111
P4DR	P4 口驱动电流控制寄存器	FE2CH	P47DR	P46DR	P45DR	P44DR	P43DR	P42DR	P41DR	P40DR	1111,1111
P5DR	P5 口驱动电流控制寄存器	FE2DH	-	-	P55DR	P54DR	P53DR	P52DR	P51DR	P50DR	xx11,1111
P0IE	P0 口输入使能控制寄存器	FE30H	P07IE	P06IE	P05IE	P04IE	P03IE	P02IE	P01IE	P00IE	1111,1111
P1IE	P1 口输入使能控制寄存器	FE31H	P17IE	P16IE	P15IE	P14IE	P13IE	P12IE	P11IE	P10IE	1111,1111
P3IE	P3 口输入使能控制寄存器	FE33H	P37IE	P36IE	P35IE	P34IE	P33IE	P32IE	P31IE	P30IE	1111,1111

9.1.1 端口数据寄存器 (Px)

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

P0	80H	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0
P1	90H	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
P2	A0H	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0
P3	B0H	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0
P4	C0H	P4.7	P4.6	P4.5	P4.4	P4.3	P4.2	P4.1	P4.0
P5	C8H	-	-	P5.5	P5.4	P5.3	P5.2	P5.1	P5.0

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

9.1.2 端口模式配置寄存器 (PxM0, PxM1)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0M0	94H	P07M0	P06M0	P05M0	P04M0	P03M0	P02M0	P01M0	P00M0
P0M1	93H	P07M1	P06M1	P05M1	P04M1	P03M1	P02M1	P01M1	P00M1
P1M0	92H	P17M0	P16M0	P15M0	P14M0	P13M0	P12M0	P11M0	P10M0
P1M1	91H	P17M1	P16M1	P15M1	P14M1	P13M1	P12M1	P11M1	P10M1
P2M0	96H	P27M0	P26M0	P25M0	P24M0	P23M0	P22M0	P21M0	P20M0
P2M1	95H	P27M1	P26M1	P25M1	P24M1	P23M1	P22M1	P21M1	P20M1
P3M0	B2H	P37M0	P36M0	P35M0	P34M0	P33M0	P32M0	P31M0	P30M0
P3M1	B1H	P37M1	P36M1	P35M1	P34M1	P33M1	P32M1	P31M1	P30M1
P4M0	B4H	P47M0	P46M0	P45M0	P44M0	P43M0	P42M0	P41M0	P40M0
P4M1	B3H	P47M1	P46M1	P45M1	P44M1	P43M1	P42M1	P41M1	P40M1
P5M0	CAH	-	-	P55M0	P54M0	P53M0	P52M0	P51M0	P50M0
P5M1	C9H	-	-	P55M1	P54M1	P53M1	P52M1	P51M1	P50M1

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

注意: 当有I/O口被选择为ADC输入通道时, 必须设置PxM0/PxM1寄存器将I/O口模式设置为输入模式。另外如果MCU进入掉电模式/时钟停振模式后, 仍需要使能ADC通道, 则需要设置PxIE寄存器关闭数字输入, 才能保证不会有额外的耗电

9.1.3 端口上拉电阻控制寄存器 (PxPU)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0PU	FE10H	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU
P1PU	FE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU
P2PU	FE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU
P3PU	FE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU

P4PU	FE14H	P47PU	P46PU	P45PU	P44PU	P43PU	P42PU	P41PU	P40PU
P5PU	FE15H	-	-	P55PU	P54PU	P53PU	P52PU	P51PU	P50PU

端口内部4.1K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 4.1K 上拉电阻
- 1: 使能端口内部的 4.1K 上拉电阻

9.1.4 端口施密特触发控制寄存器 (PxNCS)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0NCS	FE18H	P07NCS	P06NCS	P05NCS	P04NCS	P03NCS	P02NCS	P01NCS	P00NCS
P1NCS	FE19H	P17NCS	P16NCS	P15NCS	P14NCS	P13NCS	P12NCS	P11NCS	P10NCS
P2NCS	FE1AH	P27NCS	P26NCS	P25NCS	P24NCS	P23NCS	P22NCS	P21NCS	P20NCS
P3NCS	FE1BH	P37NCS	P36NCS	P35NCS	P34NCS	P33NCS	P32NCS	P31NCS	P30NCS
P4NCS	FE1CH	P47NCS	P46NCS	P45NCS	P44NCS	P43NCS	P42NCS	P41NCS	P40NCS
P5NCS	FE1DH	-	-	P55NCS	P54NCS	P53NCS	P52NCS	P51NCS	P50NCS

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。

9.1.5 端口电平转换速度控制寄存器 (PxSR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0SR	FE20H	P07SR	P06SR	P05SR	P04SR	P03SR	P02SR	P01SR	P00SR
P1SR	FE21H	P17SR	P16SR	P15SR	P14SR	P13SR	P12SR	P11SR	P10SR
P2SR	FE22H	P27SR	P26SR	P25SR	P24SR	P23SR	P22SR	P21SR	P20SR
P3SR	FE23H	P37SR	P36SR	P35SR	P34SR	P33SR	P32SR	P31SR	P30SR
P4SR	FE24H	P47SR	P46SR	P45SR	P44SR	P43SR	P42SR	P41SR	P40SR
P5SR	FE25H	-	-	P55SR	P54SR	P53SR	P52SR	P51SR	P50SR

控制端口电平转换的速度

- 0: 电平转换速度快，相应的上下冲会比较大
- 1: 电平转换速度慢，相应的上下冲比较小

9.1.6 端口驱动电流控制寄存器 (PxDR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0DR	FE28H	P07DR	P06DR	P05DR	P04DR	P03DR	P02DR	P01DR	P00DR
P1DR	FE29H	P17DR	P16DR	P15DR	P14DR	P13DR	P12DR	P11DR	P10DR
P2DR	FE2AH	P27DR	P26DR	P25DR	P24DR	P23DR	P22DR	P21DR	P20DR
P3DR	FE2BH	P37DR	P36DR	P35DR	P34DR	P33DR	P32DR	P31DR	P30DR
P4DR	FE2CH	P47DR	P46DR	P45DR	P44DR	P43DR	P42DR	P41DR	P40DR
P5DR	FE2DH	-	-	P55DR	P54DR	P53DR	P52DR	P51DR	P50DR

控制端口的驱动能力

- 0: 增强驱动能力**
- 1: 一般驱动能力**

9.1.7 端口数字信号输入使能控制寄存器（PxIE）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0IE	FE30H	P07IE	P06IE	P05IE	P04IE	P03IE	P02IE	P11IE	P00IE
P1IE	FE31H	P17IE	P16IE	P15IE	P14IE	P13IE	P12IE	P11IE	P10IE
P3IE	FE33H	P37IE	P36IE	P35IE	P34IE	P33IE	P32IE	P31IE	P30IE
P5IE	FE35H	-	-	P55IE	P54IE	P53IE	P52IE	P51IE	P50IE

数字信号输入使能控制

- 0: 禁止数字信号输入。若 I/O 被当作比较器输入口、ADC 输入口或者触摸按键输入口等模拟口时，进入时钟停振模式前，必须设置为 0，否则会有额外的耗电。
- 1: 使能数字信号输入。若 I/O 被当作数字口时，必须设置为 1，否 MCU 无法读取外部端口的电平。

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70mA,即从Vcc流入的电流建议不要超过70mA,从Gnd流出电流建议不要超过70mA,整体流入/流出电流建议都不要超过70mA。

(STC8G1K08A系列和STC8G1K08-8Pin系列,整个芯片的工作电流推荐不要超过35mA,即从Vcc流入的电流建议不要超过35mA,从Gnd流出电流建议不要超过35mA,整体流入/流出电流建议都不要超过35mA)

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。

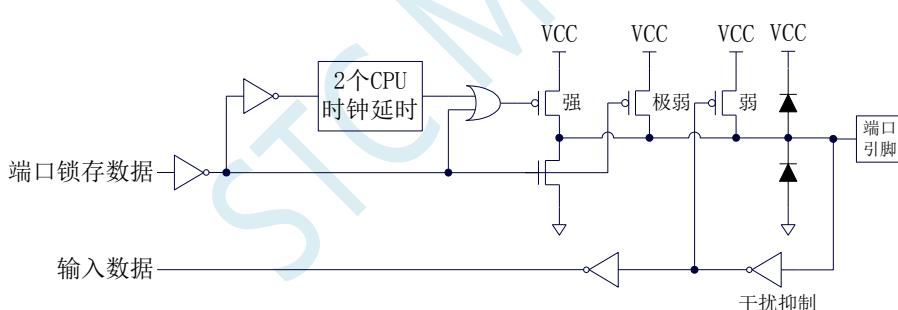
在 3 个上拉晶体管中，有 1 个上拉晶体管称为“弱上拉”，当端口寄存器为 1 且引脚本身也为 1 时打开。此上拉提供基本驱动电流使准双向口输出为 1。如果一个引脚输出为 1 而由外部装置下拉到低时，弱上拉关闭而“极弱上拉”维持开状态，为了把这个引脚强拉为低，外部装置必须有足够的灌电流能力使引脚上的电压降到门槛电压以下。对于 5V 单片机，“弱上拉”晶体管的电流约 250uA；对于 3.3V 单片机，“弱上拉”晶体管的电流约 150uA。

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。

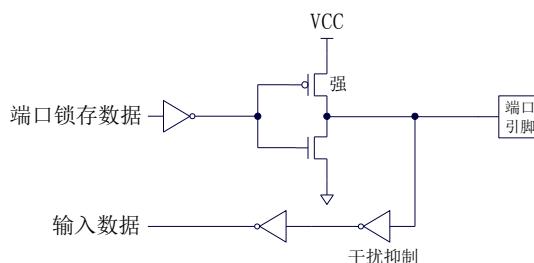
准双向口（弱上拉）输出如下图所示：



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：

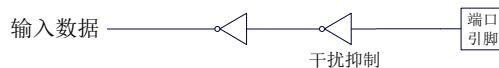


9.3.3 高阻输入

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:



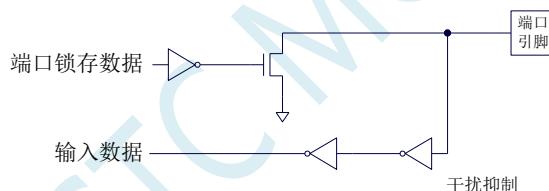
9.3.4 开漏输出

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。

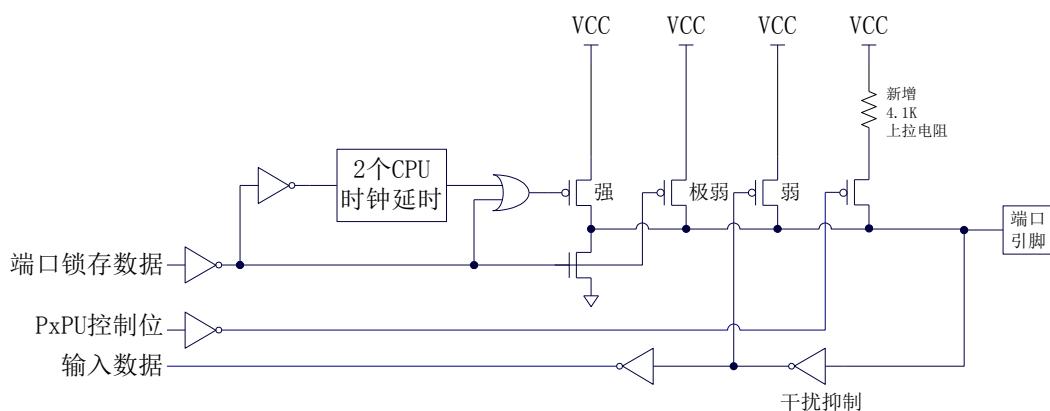
开漏端口带有一个施密特触发输入以及一个干扰抑制电路。

输出端口配置如下图所示:



9.3.5 新增 4.1K 上拉电阻

STC8 系列所有的 I/O 口内部均可使能一个大约 4.1K 的上拉电阻（由于制造误差，上拉电阻的范围可能为 3K~5K）



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	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU
P1PU	FE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU
P2PU	FE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU
P3PU	FE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU
P4PU	FE14H	P47PU	P46PU	P45PU	P44PU	P43PU	P42PU	P41PU	P40PU
P5PU	FE15H	-	-	P55PU	P54PU	P53PU	P52PU	P51PU	P50PU

端口内部4.1K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 4.1K 上拉电阻
- 1: 使能端口内部的 4.1K 上拉电阻

9.3.6 如何设置 I/O 口对外输出速度

当用户需要 I/O 口对外输出较快的频率时，可通过加大 I/O 口驱动电流以及增加 I/O 口电平转换速度以达到提高 I/O 口对外输出速度

设置 PxSR 寄存器，可用于控制 I/O 口电平转换速度，设置为 0 时相应的 I/O 口为快速翻转，设置为 1 时为慢速翻转。

设置 PxDR 寄存器，可用于控制 I/O 口驱动电流大小，设置为 1 时 I/O 输出为一般驱动电流，设置为 0 时为强驱动电流

9.3.7 如何设置 I/O 口电流驱动能力

若需要改变 I/O 口的电流驱动能力, 可通过设置 PxDR 寄存器来实现

设置 PxDR 寄存器, 可用于控制 I/O 口驱动电流大小, 设置为 1 时 I/O 输出为一般驱动电流, 设置为 0 时为强驱动电流

9.3.8 如何降低 I/O 口对外辐射

由于设置 PxSR 寄存器, 可用于控制 I/O 口电平转换速度, 设置 PxDR 寄存器, 可用于控制 I/O 口驱动电流大小

当需要降低 I/O 口对外的辐射时, 需要将 PxSR 寄存器设置为 1 以降低 I/O 口电平转换速度, 同时需要将 PxDR 寄存器设为 1 以降低 I/O 驱动电流, 最终达到降低 I/O 口对外辐射

9.4 范例程序

9.4.1 端口模式设置

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr P0M0      = 0x94;
sfr P0M1      = 0x93;
sfr P1M0      = 0x92;
sfr P1M1      = 0x91;
sfr P2M0      = 0x96;
sfr P2M1      = 0x95;
sfr P3M0      = 0xb2;
sfr P3M1      = 0xb1;
sfr P4M0      = 0xb4;
sfr P4M1      = 0xb3;
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);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz
```

P0M0	DATA	094H
P0M1	DATA	093H
P1M0	DATA	092H
P1M1	DATA	091H
P2M0	DATA	096H
P2M1	DATA	095H
P3M0	DATA	0B2H
P3M1	DATA	0B1H
P4M0	DATA	0B4H
P4M1	DATA	0B3H

```

P5M0      DATA      0CAH
P5M1      DATA      0C9H
P6M0      DATA      0CCH
P6M1      DATA      0CBH
P7M0      DATA      0E2H
P7M1      DATA      0E1H

        ORG      0000H
        LJMP    MAIN

        ORG      0100H
MAIN:
        MOV      SP, #5FH

        MOV      P0M0,#00H          ; 设置 P0.0~P0.7 为双向口模式
        MOV      P0M1,#00H
        MOV      P1M0,#0FFH          ; 设置 P1.0~P1.7 为推挽输出模式
        MOV      P1M1,#00H
        MOV      P2M0,#00H          ; 设置 P2.0~P2.7 为高阻输入模式
        MOV      P2M1,#0FFH
        MOV      P3M0,#0FFH          ; 设置 P3.0~P3.7 为开漏模式
        MOV      P3M1,#0FFH

        JMP      $
END

```

9.4.2 双向口读写操作

C 语言代码

//测试工作频率为 11.0592MHz

```

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

sfr    P0M1      = 0x93;
sfr    P0M0      = 0x94;
sfr    P1M1      = 0x91;
sfr    P1M0      = 0x92;
sfr    P2M1      = 0x95;
sfr    P2M0      = 0x96;
sfr    P3M1      = 0xb1;
sfr    P3M0      = 0xb2;
sfr    P4M1      = 0xb3;
sfr    P4M0      = 0xb4;
sfr    P5M1      = 0xc9;
sfr    P5M0      = 0xca;
sbit   P00       = P0^0;

```

```

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
}

```

```

P2M0 = 0x00;
P2M1 = 0x00;
P3M0 = 0x00;
P3M1 = 0x00;
P4M0 = 0x00;
P4M1 = 0x00;
P5M0 = 0x00;
P5M1 = 0x00;

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);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

```

P0M1      DATA      093H
P0M0      DATA      094H
P1M1      DATA      091H
P1M0      DATA      092H
P2M1      DATA      095H
P2M0      DATA      096H
P3M1      DATA      0B1H
P3M0      DATA      0B2H
P4M1      DATA      0B3H
P4M0      DATA      0B4H
P5M1      DATA      0C9H
P5M0      DATA      0CAH

        ORG      0000H
        LJMP     MAIN

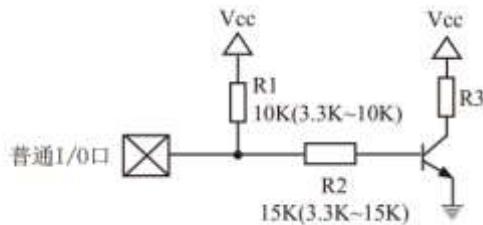
        ORG      0100H
MAIN:
        MOV      SP, #5FH
        MOV      P0M0, #00H
        MOV      P0M1, #00H
        MOV      P1M0, #00H
        MOV      P1M1, #00H
        MOV      P2M0, #00H
        MOV      P2M1, #00H
        MOV      P3M0, #00H
        MOV      P3M1, #00H
        MOV      P4M0, #00H
        MOV      P4M1, #00H
        MOV      P5M0, #00H
        MOV      P5M1, #00H

```

MOV	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		; 等待两个时钟
NOP		
MOV	C,P0.0	; 读取端口状态
JMP	\$	
END		

STCMCU

9.5 一种典型三极管控制电路



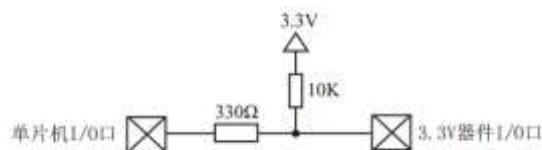
如果上拉控制，建议加上拉电阻 R1(3.3K~10K)，如果不加上拉电阻 R1(3.3K~10K)，建议 R2 的值在 15K 以上，或用强推挽输出。

9.6 典型发光二极管控制电路



9.7 混合电压供电系统 3V/5V 器件 I/O 口互连

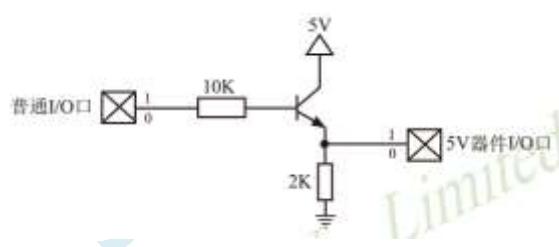
STC 系列宽电压单片机工作在 5V 时, 如需要直接连接 3.3V 器件时, 为防止 3.3V 器件承受不了 5V, 可将相应的单片机 I/O 口先串一个 330Ω 的限流电阻到 3.3V 器件 I/O 口, 程序初始化时将单片机的 I/O 口设置成开漏配置, 断开内部上拉电阻, 相应的 3.3V 器件 I/O 口外部加 10K 上拉电阻到 3.3V 器件的 Vcc, 这样高电平是 3.3V, 低电平是 0V, 输入输出一切正常。



STC 宽电压单片机工作在 3V 时, 如需要直接连接 5V 器件时, 如果相应的 I/O 口是输入, 可在该 I/O 口上串接一个隔离二极管, 隔离高压部分。外部信号电压高于单片机工作电压时截止, I/O 口因内内部上拉到高电平, 所以读 I/O 口状态是高电平; 外部信号电压为低时导通, I/O 口被钳位在 0.7V, 小于 0.8V 时单片机读 I/O 口状态是低电平。



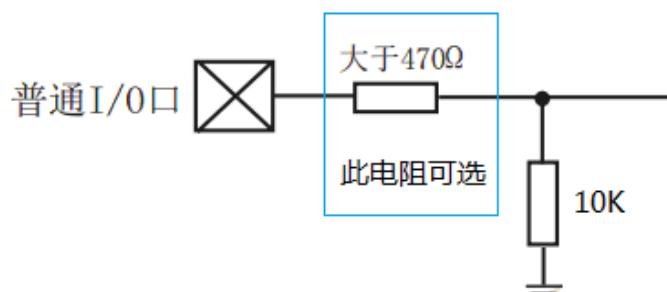
STC 宽电压单片机工作在 3V 时, 如需要直接连接 5V 器件时, 如果相应的 I/O 口是输出, 可用一个 NPN 三极管隔离, 电路如下:



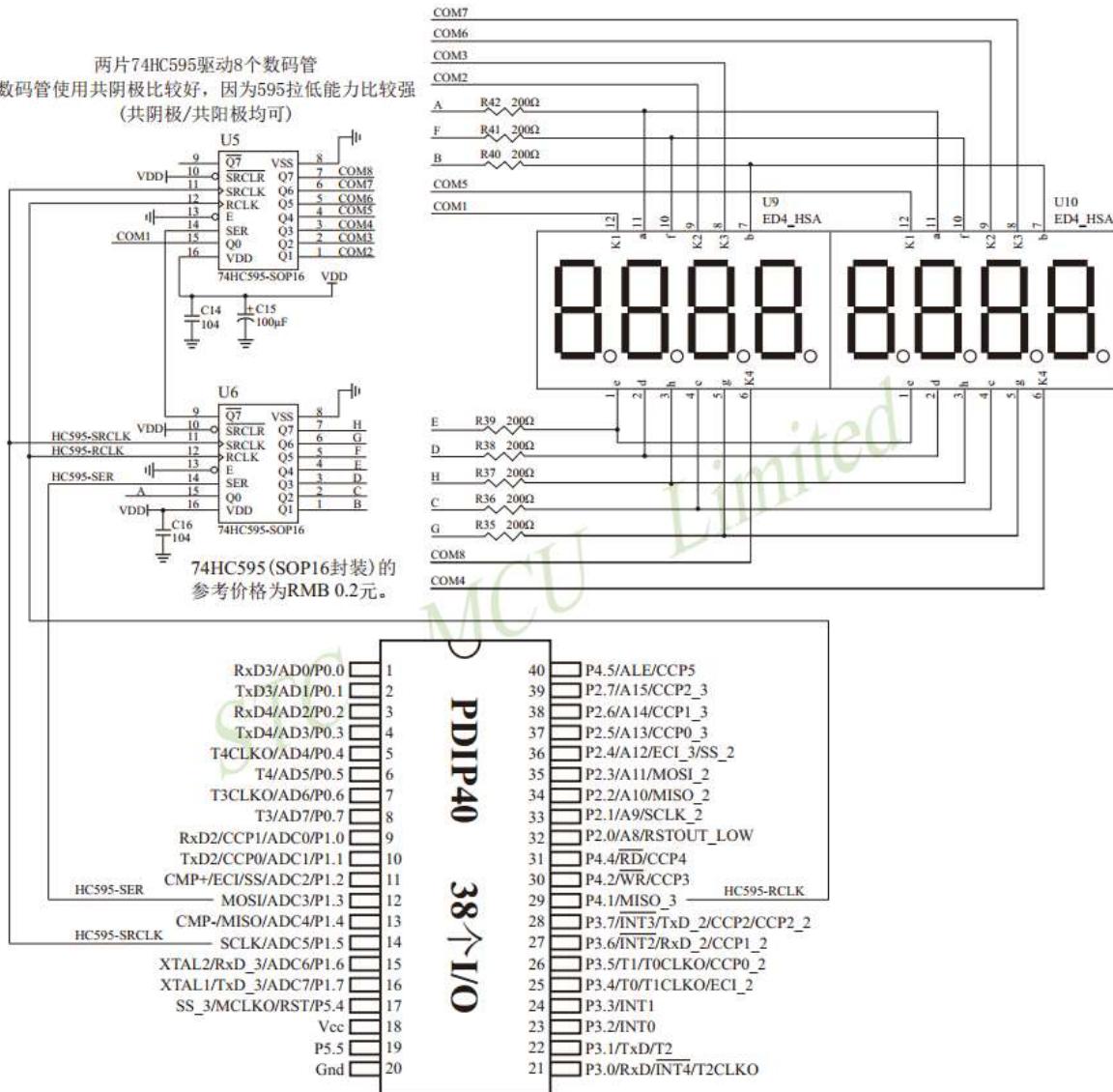
9.8 如何让 I/O 口上电复位时为低电平

传统 8051 单片机上电复位时普通 I/O 口为弱上拉(准双向口)高电平输出, 而很多实际应用要求上电时某些 I/O 口为低电平输出, 否则所控制的系统(如马达)就会误动作, 新一代 STC8G 系列和 STC8H 系列单片机由于所有的 I/O 复位后是高阻输入(除 P3.0/P3.1 是传统的弱上拉), 加一个下拉电阻就可保证上电时为低电平, 后续要改为高电平, 只需要将 I/O 的模式改为强推挽输出, 对外输出高电平即可。

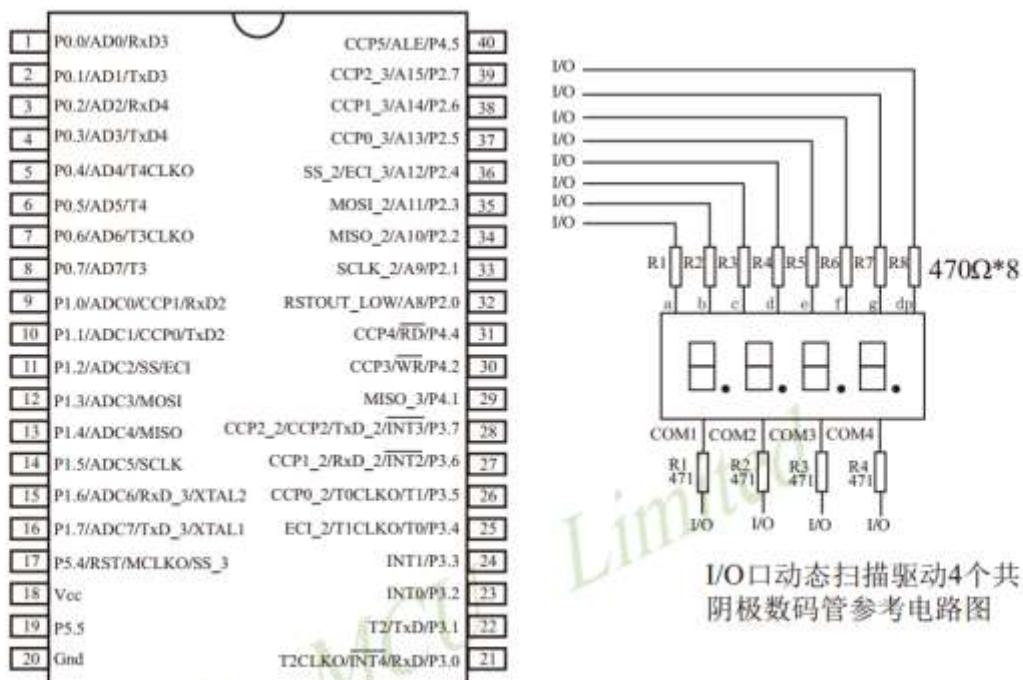
如下: 在 STC 的单片机 I/O 口上加一个下拉电阻(10K 左右), 这样上电复位时, 除了下载口 P3.0 和 P3.1 为弱上拉(准双向口)外, 其他 I/O 口均为高阻输入模式, 而外部有下拉电阻, 所以该 I/O 口上电复位时外部为低电平。如果要将此 I/O 口驱动为高电平, 可将此 I/O 口设置为强推挽输出, 而强推挽输出时, I/O 口驱动电流可达 20mA, 故肯定可以将该口驱动为高电平输出。



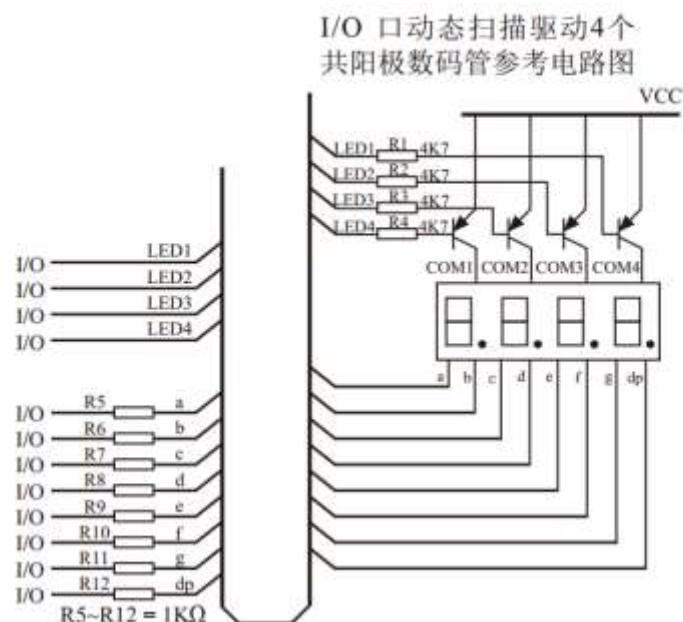
9.9 利用 74HC595 驱动 8 个数码管(串行扩展,3 根线)的线路图



9.10 I/O 口直接驱动 LED 数码管应用线路图



I/O 口动态扫描驱动数码时，可以一次点亮一个数码管中的8段，但为降低功耗，建议可以一次只点亮其中的4段或者2段



9.11 用 STC 系列 MCU 的 I/O 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。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, 则会降低成本, 工作更可靠。

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 (驱动会有几百 uA 电流)。

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 有一个扭转电压阀值, 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 3.0V、1/4 DUTY、1/3 BIAS, 表示 LCD 显示电压为 3.0V, 4 个 COM, 阀值大约是 1.5V, 当加在某段 LCD 两端电压为 3.0V 时显示, 而加 1.0V 时不显示。但是 LCD 对于驱动电压的反应不是很敏感的, 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 要用交流驱动, 其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 3.0V 或 3.3V 工作, 并且每个 COM 都串一个 20K 电阻接到一个电容 C1, RC 滤波后得到一个中点电压 1/2VDD。在轮到某个 COM 扫描时, 连接的 IO 设置成推挽输出, 其余 COM 设置成高阻, 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。每个 COM 通过 20K 电阻连接到电容 C1 上的 1/2VDD 电压, 而 SEG 根据是否显示输出高低电平, 这样加在 LCD 段上的电压, 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 接的 IO 通过电阻分压输出 1.5V、3.5V, COM 接的 IO 通过电阻分压输出 0.5V、2.5V (高阻时)、4.5V, 分压电阻公共点接到一个电容 C1, RC 滤波后得到一个中点电压 1/2VDD。在轮到某个 COM 扫描时, 设置成推挽输出, 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 根据是否显示输出高低电平, 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

本电路MCU工作于3.3V驱动1/4 Duty, 1/2 bias, 3V的段码LCD。
本电路适用于STC8系列，IO都是普通IO操作。

C1用于中点电压滤波，4.7~47uF。

MCU睡眠后LCD驱动部分电路不会耗电。

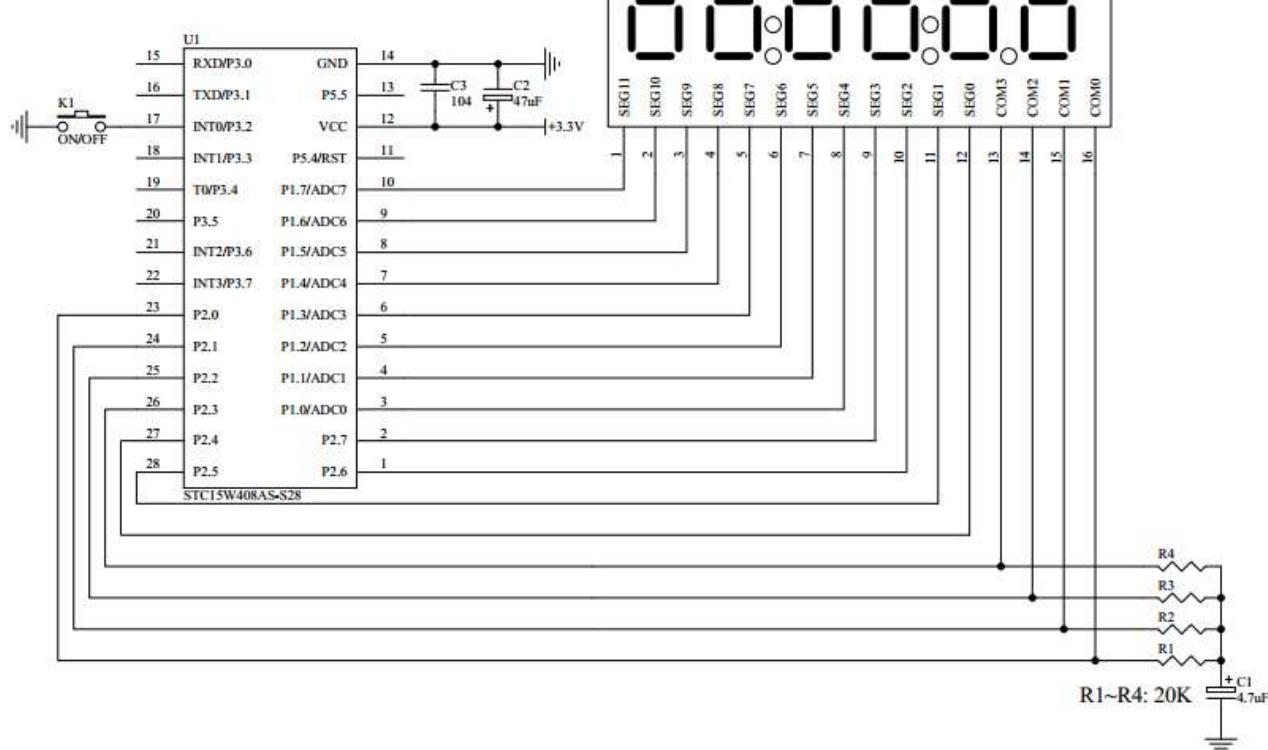


图 2: 段码名称图

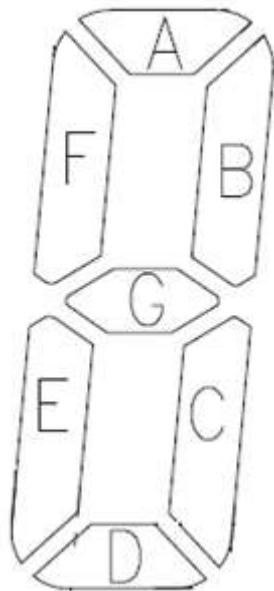


图 3: 1/4Duty 1/2BIAS 扫描原理图

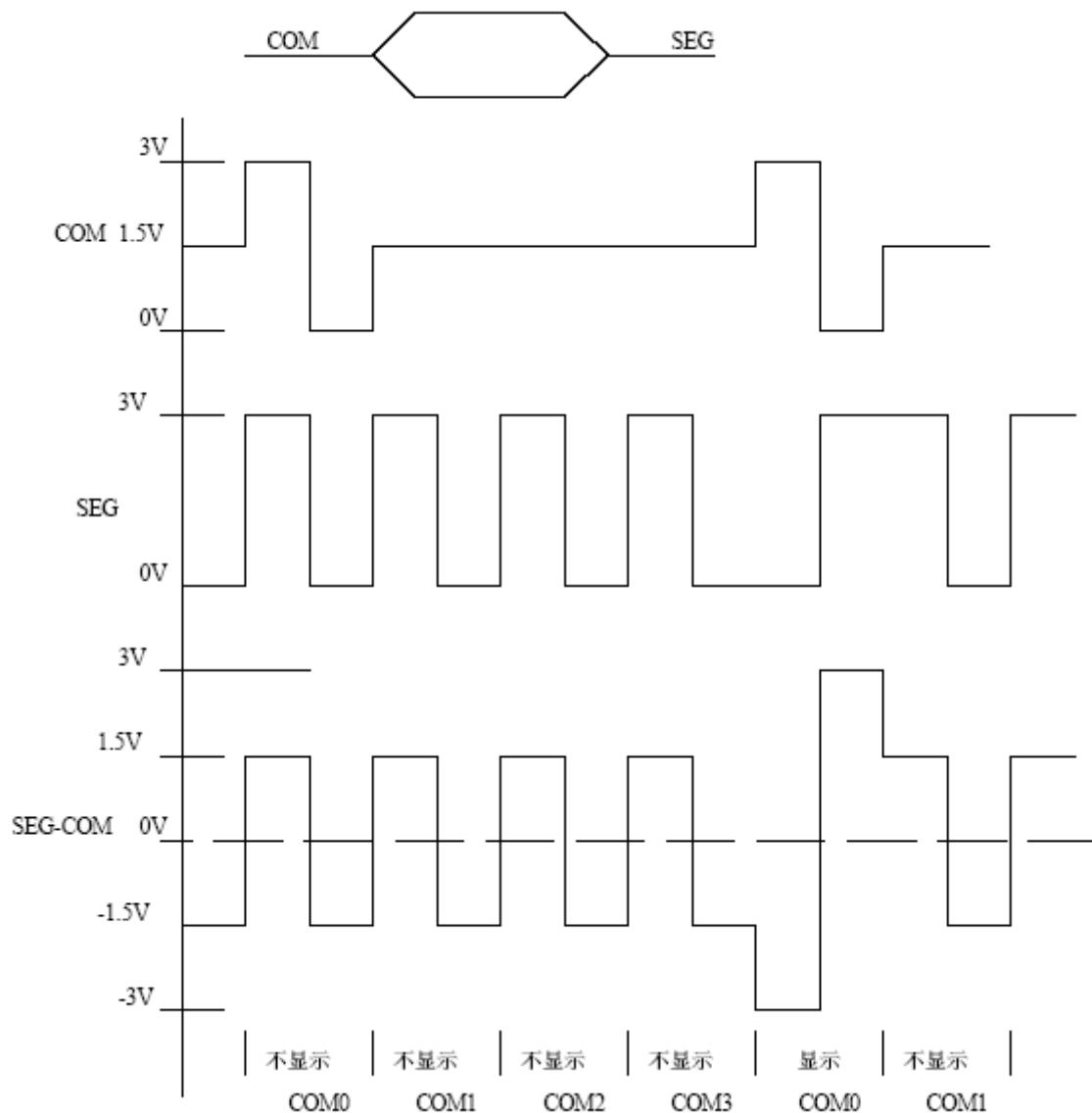


图 4: 驱动 1/4Duty 1/3BIAS 3V LCD 的电路

本电路MCU工作于5V驱动1/4 Duty, 1/3 bias, 3V的段码LCD。
 本电路适用于STC8系列，IO都是普通IO操作。
 C1用于中点电压滤波，4.7~47uF。
 MCU睡眠后LCD驱动部分电路不会耗电。

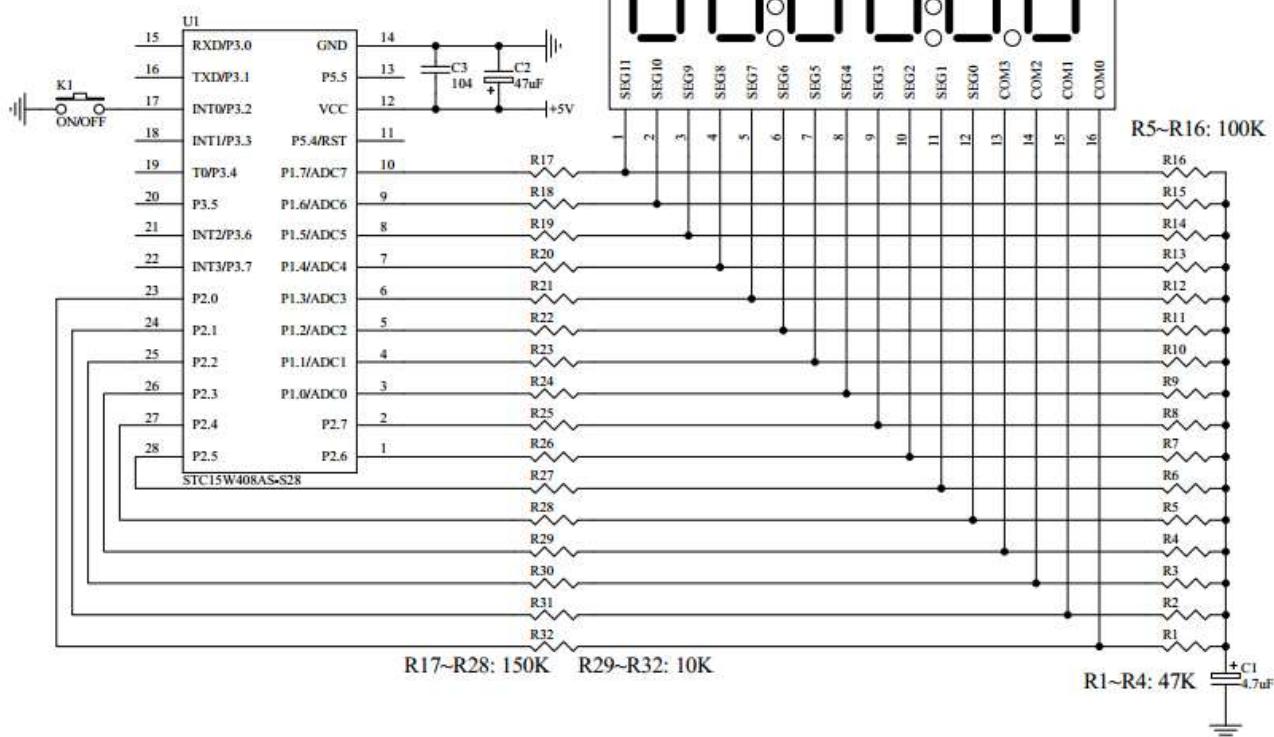
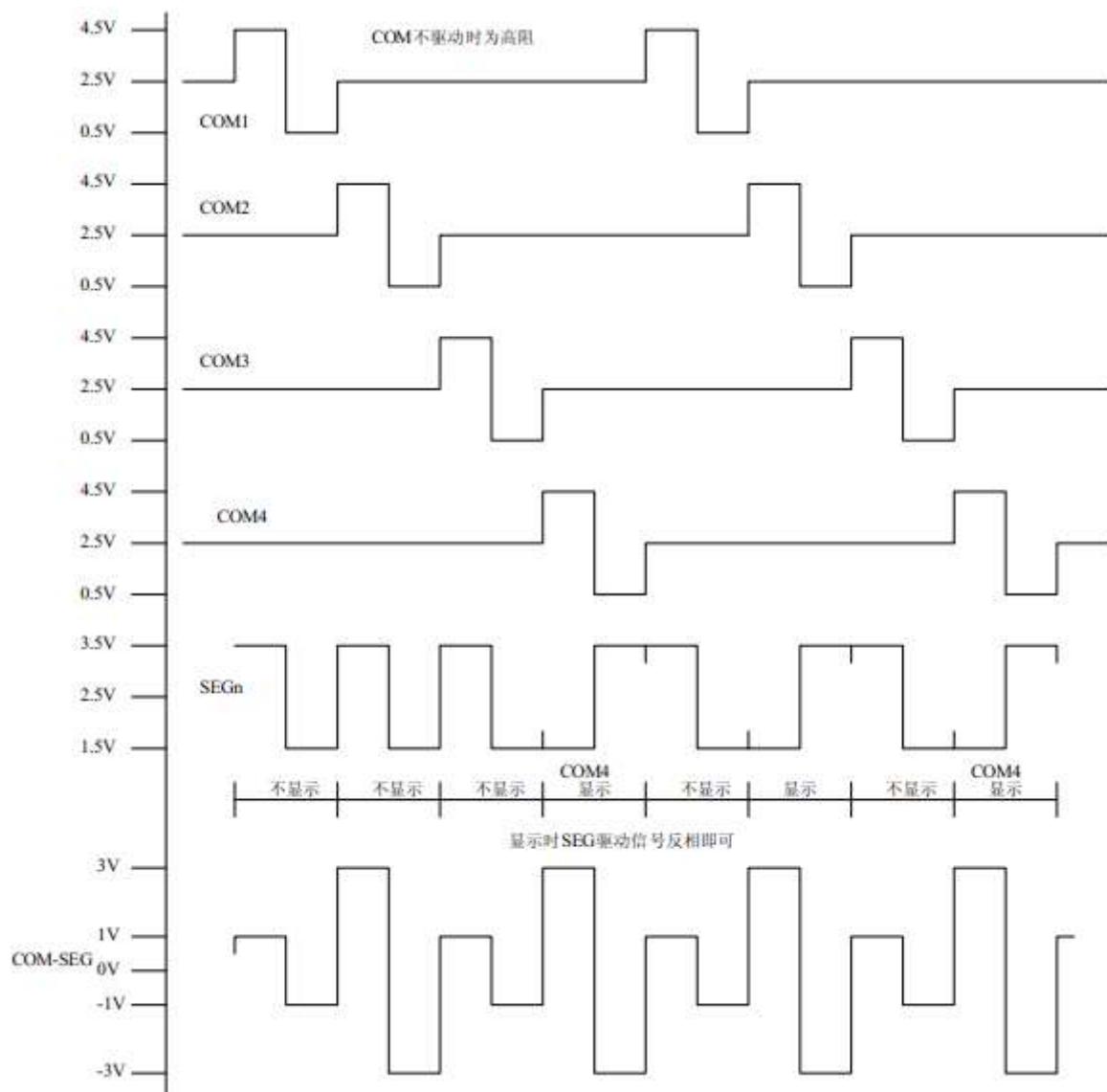


图 5: 1/4Duty 1/3BIAS 扫描原理图



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 真值表和显存映射表

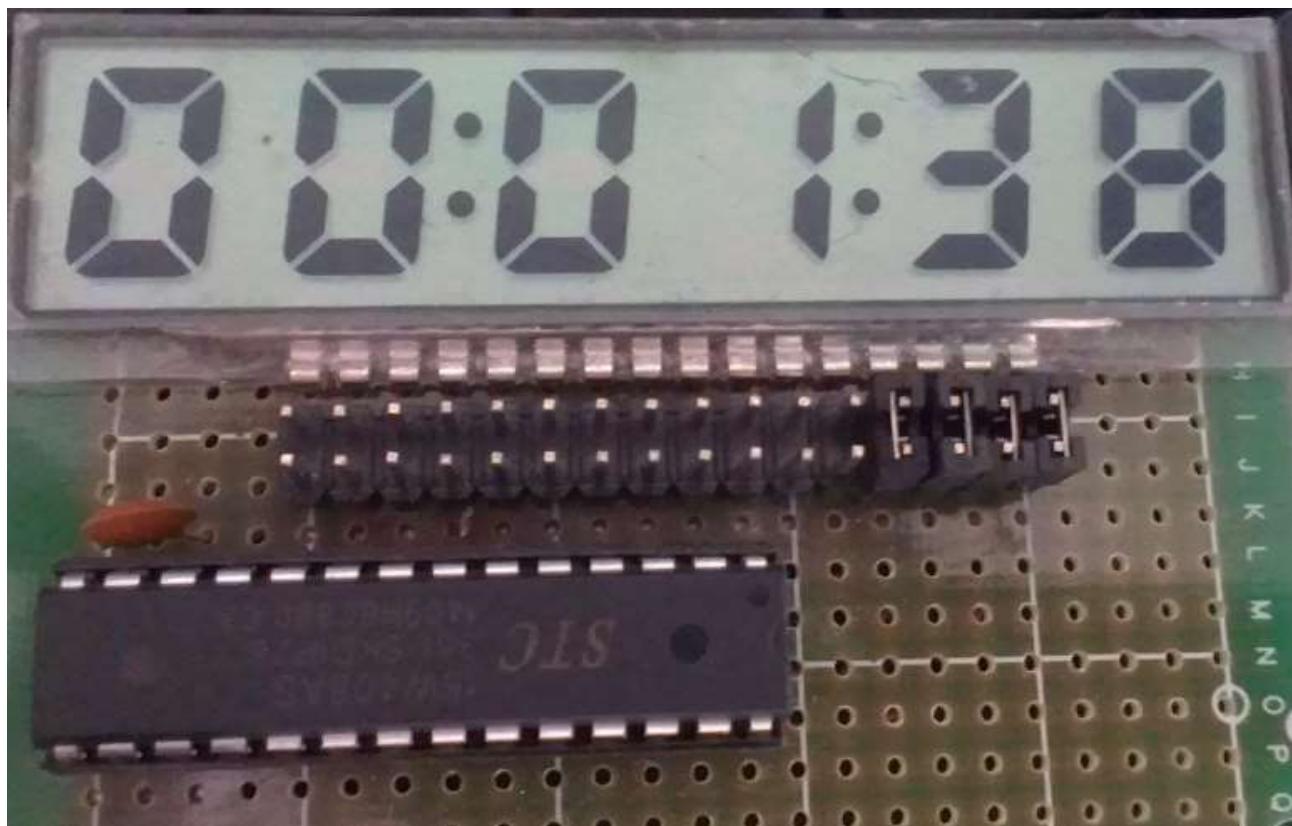
LCD真值表:

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	2:	2D	2.	3D	4:	4D	4.	5D	5.	6D	COM3				
1E	1C	2E	2C	3E	3C	4E	4C	5E	5C	6E	6C	COM2				
1G	1B	2G	2B	3G	3B	4G	4B	5G	5B	6G	6B				COM1	
1F	1A	2F	2A	3F	3A	4F	4A	5F	5A	6F	6A					COM0

显存映射表:

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

图 7: 驱动效果照片



本 LCD 扫描程序仅需要两个函数:

1、LCD 段码扫描函数 void LCD_scan(void)

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

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

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

另外, 用宏来显示、熄灭或闪烁冒号或小数点。

汇编代码

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

```
*****  
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
P6M1	DATA	0xCB
P6M0	DATA	0xCC
P7M1	DATA	0xE1
P7M0	DATA	0xE2
AUXR	DATA	0x8E
INT_CLKO	DATA	0x8F
IE2	DATA	0xAF
P4	DATA	0xC0
T2H	DATA	0xD6
T2L	DATA	0xD7

;*****

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

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

;*****

ORG	0000H
LJMP	F_Main
ORG	000BH
LJMP	F_Timer0_Interrupt

;*****

ORG	0100H
-----	-------

F_Main:

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

```

;*****
MOV      R2, #8
MOV      R0, #LCD_buff

L_ClearLcdRam:
MOV      @R0, #0
INC
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
MOV     second, #0              ; 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
MOV     hour, #0                ;24 小时到

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          ; 工作模式 0: 16 位自动重装
ANL      INT_CLKO, #NOT 0x01 ; 不输出时钟
ORL      AUXR, #0x80        ; IT mode
MOV      TH0, #HIGH (-22H8) ; 2ms
MOV      TL0, #LOW (-22H8) ;
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
MOV     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
MOV     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
MOV     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
MOV     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 ;R6 为第几个数字, 为1~6, R7 为要显示的数字

MOV      R6, #6          ;LCD_load(6,second%10);
MOV      A, second
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

```

RET

;*****

T_COM:

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

F_LCD_scan:

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

L_LCD_Scan3:

MOV	DPTR, #T_COM	;某个COM 设置为推挽输出
MOV	A, R7	
MOVC	A, @A+DPTR	
ORL	P2M0, A	
CPL	A	

```

ANL      P2M1, A

INC      scan_index           ;if(++scan_index == 8)      scan_index = 0;
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: ;R6 为第几个数字, 为1~6, R7 为要显示的数字

```

MOV      DPTR, #T_Display          ;i = t_display[dat];
MOV      A, R7
MOVC    A, @A+DPTR
MOV      B, A                  ;要显示的数字

MOV      A, R6
CJNE    A, #I, 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
```

E

N

D

C 语言代码

```
***** 功能说明 *****
```

用 STC15 系列测试 I/O 直接驱动段码 LCD(6 个 8 字 LCD, 1/4 Duty, 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
    PIn_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(!P32)                                     //键按下，准备睡眠
{
    LCD_CLR_2M;                               //显示时分间隔:
    LCD_CLR_4M;                               //显示分秒间隔:
    LCD_load(1,DIS_BLACK);
    LCD_load(2,DIS_BLACK);
    LCD_load(3,0);
    LCD_load(4,0x0F);
    LCD_load(5,0x0F);
    LCD_load(6,DIS_BLACK);

    while(!P32) delay_ms(10);                //等待释放按键
    delay_ms(50);
    while(!P32) delay_ms(10);                //再次等待释放按键

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

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

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

    LCD_SET_2M;                             //显示时分间隔:
    LCD_SET_4M;                             //显示分秒间隔:
    LoadRTC();                            //显示时间
    TR0 = 1;                                //打开定时器
    while(!P32) delay_ms(10);                //等待释放按键
    delay_ms(50);
    while(!P32) delay_ms(10);                //再次等待释放按键
}

}

}

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

} //Timer0 中断函数
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
        0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
        0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x77,0x7C,0x39,0x5E,0x79,0x71,
        //black -
        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_mask[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);

}

/*****显示时/分*****/
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为基地址变址寻址单元中的数据送入累加器	1	4
MOVC	A,@A+PC	以PC为基地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[2:0]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

(C 语言程序中使用中断号大于 31 的中断时, 在 Keil 中编译会报错, 解决办法请参考附录)

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。

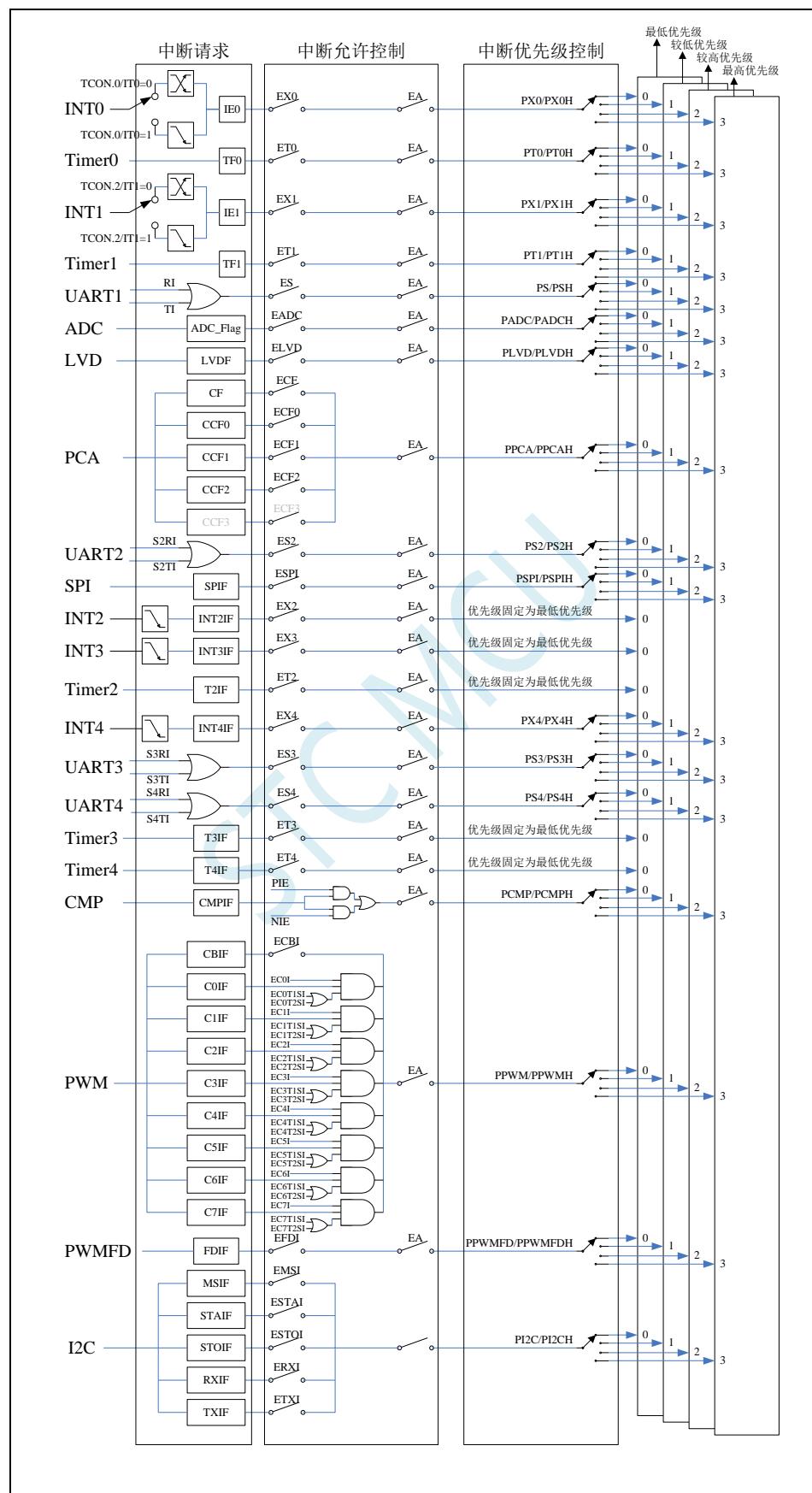
11.1 STC8G 系列中断源

下表中 √ 表示对应的系列有相应的中断源

中断源	STC8G1K08 系列	STC8G1K08 8PIN系列	STC8G1K08A 系列	STC8G2K64S4 系列	STC15H2K64S4 系列	STC8G2K64S2 系列	STC8G1K08T 系列
外部中断 0 中断 (INT0) 支持下降沿和边沿中断	√	√	√	√	√	√	√
定时器 0 中断 (Timer0)	√	√	√	√	√	√	√
外部中断 1 中断 (INT1) 支持下降沿和边沿中断	√	√	√	√	√	√	√
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) 支持下降沿、上升沿和边沿中断	√		√	√	√	√	√
串口 2 中断 (UART2)	√			√	√	√	
串行外设接口中断 (SPI)	√	√	√	√	√	√	√
外部中断 2 中断 (INT2) 支持下降沿中断	√	√	√	√	√	√	√

外部中断 3 中断 (INT3) 支持下降沿中断	√	√	√	√	√	√	√
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 PWM0 中断				√	√	√	
PWM0 异常检测中断 (PWM0FD)				√	√	√	
I2C 总线中断	√	√	√	√	√	√	√
触摸按键中断							√
增强型 PWM1 中断				√	√	✗	
增强型 PWM2 中断				✗	√	√	
增强型 PWM3 中断				✓	✓	✗	
增强型 PWM4 中断				✓	✓	✗	
增强型 PWM5 中断				✓	✓	✗	
PWM2 异常检测中断 (PWM2FD)				✓	✓	✓	
PWM4 异常检测中断 (PWM4FD)				✓	✓	✗	

11.2 STC8G 中断结构图



11.3 STC8G 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	PX0PX0H	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
PWM0	00B3H	22	PPWM0,PPWM0H	0/1/2/3	CBIF	ECBI
					CnIF	ECnI && ECnT1SI
						ECnI && ECnT2SI
PWM0FD	00BBH	23	PPWM0FD,PPWM0FDH	0/1/2/3	FDIF	EFDI
I2C	00C3H	24	PI2C,PI2CH	0/1/2/3	MSIF	EMSI
					STAIF	ESTAI
					RXIF	ERXI
					TXIF	ETXI
					STOIF	ESTOI
PWM1	00E3H	28	PPWM1,PPWM1H	0/1/2/3	CBIF	ECBI
					CnIF	ECnI && ECnT1SI
						ECnI && ECnT2SI
PWM2	00EBH	29	PPWM2,PPWM2H	0/1/2/3	CBIF	ECBI
					CnIF	ECnI && ECnT1SI
						ECnI && ECnT2SI
PWM3	00F3H	30	PPWM3,PPWM3H	0/1/2/3	CBIF	ECBI
					CnIF	ECnI && ECnT1SI
						ECnI && ECnT2SI
PWM4	00FBH	31	PPWM4,PPWM4H	0/1/2/3	CBIF	ECBI
					CnIF	ECnI && ECnT1SI
						ECnI && ECnT2SI
PWM5	0103H	32	PPWM5,PPWM5H	0/1/2/3	CBIF	ECBI
					CnIF	ECnI && ECnT1SI
						ECnI && ECnT2SI
PWM2FD	010BH	33	PPWM2FD,PPWM2FDH	0/1/2/3	FDIF	EFDI
PWM4FD	0113H	34	PPWM4FD,PPWM4FDH	0/1/2/3	FDIF	EFDI
TKSU	011BH	35	PTKSU,PTKSUH	0/1/2/3	TKIF	ETKSUI

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 PWM0_Routine(void)     interrupt 22;
void PWM0FD_Routine(void)   interrupt 23;
void I2C_Routine(void)      interrupt 24;
void PWM1_Routine(void)     interrupt 28;
void PWM2_Routine(void)     interrupt 29;
void PWM3_Routine(void)     interrupt 30;
void PWM4_Routine(void)     interrupt 31;

//void PWM5_Routine(void)     interrupt 32;
//void PWM2FD_Routine(void)   interrupt 33;
//void PWM4FD_Routine(void)   interrupt 34;
//void TKSU_Routine(void)     interrupt 35;
```

中断号超过31的C语言中断服务程序不能直接用interrupt声明, 请参考附录的处理方法, 汇编语言不受影响

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	ETKSUI	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	0000,0000
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	PPWM2FD PTKSU	PI2C	PCMP	PX4	PPWM0FD	PPWM0	PSPI	PS2	0000,0000
IP2H	高中断优先级控制寄存器 2	B6H	PPWM2FDH PTKSUH	PI2CH	PCMPH	PX4H	PPWM0FDH	PPWM0H	PSPIH	PS2H	0000,0000
IP3	中断优先级控制寄存器 3	DFH	PPWM4FD	PPWM5	PPWM4	PPWM3	PPWM2	PPWM1	PS4	PS3	0000,0000
IP3H	高中断优先级控制寄存器 3	EEH	PPWM4FDH	PPWM5H	PPWM4H	PPWM3H	PPWM2H	PPWM1H	PS4H	PS3H	0000,0000
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	SMODO	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
ADC_CONTR	ADC 控制寄存器	BCH	ADC_POWER	ADC_START	ADC_FLAG	ADC_EPWMT	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,x000
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
CMPCR1	比较器控制寄存器 1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES	0000,0000
PWMCFG01	增强型 PWM 配置寄存器	F6H	PWM1CBIF	EPWM1CBI	FLTPS0	PWM1CEN	PWM0CBIF	EPWM0CBI	ENPWM0TA	PWM0CEN	0000,0000
PWMCFG23	增强型 PWM 配置寄存器	F7H	PWM3CBIF	EPWM3CBI	FLTPS1	PWM3CEN	PWM2CBIF	EPWM2CBI	ENPWM2TA	PWM2CEN	0000,0000
PWMCFG45	增强型 PWM 配置寄存器	FEH	PWM5CBIF	EPWM5CBI	FLTPS2	PWM5CEN	PWM4CBIF	EPWM4CBI	ENPWM4TA	PWM4CEN	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	
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
PWM0IF	PWM0 中断标志寄存器	FF05H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWM0FDCR	PWM0 异常检测控制寄存器	FF06H	INV_CMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	0000,0000
PWM0OCR	PWM00 控制寄存器	FF14H	ENO	INI	-	-	ENI	ENT2I	ENT1I	00xx,x000	

PWM01CR	PWM01 控制寄存器	FF1CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM02CR	PWM02 控制寄存器	FF24H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM03CR	PWM03 控制寄存器	FF2CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM04CR	PWM04 控制寄存器	FF34H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM05CR	PWM05 控制寄存器	FF3CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM06CR	PWM06 控制寄存器	FF44H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM07CR	PWM07 控制寄存器	FF4CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM1IF	PWM1 中断标志寄存器	FF55H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWM1FDCR	PWM1 异常检测控制寄存器	FF56H	INVCM	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF	0000,0000
PWM10CR	PWM10 控制寄存器	FF64H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM11CR	PWM11 控制寄存器	FF6CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM12CR	PWM12 控制寄存器	FF74H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM13CR	PWM13 控制寄存器	FF7CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM14CR	PWM14 控制寄存器	FF84H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM15CR	PWM15 控制寄存器	FF8CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM16CR	PWM16 控制寄存器	FF94H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM17CR	PWM17 控制寄存器	FF9CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM2IF	PWM2 中断标志寄存器	FFA5H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWM2FDCR	PWM2 异常检测控制寄存器	FFA6H	INVCM	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	0000,0000
PWM20CR	PWM20 控制寄存器	FFB4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM21CR	PWM21 控制寄存器	FFBCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM22CR	PWM22 控制寄存器	FFC4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM23CR	PWM23 控制寄存器	FFCCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM24CR	PWM24 控制寄存器	FFD4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM25CR	PWM25 控制寄存器	FFDCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM26CR	PWM26 控制寄存器	FFE4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM27CR	PWM27 控制寄存器	FFECH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM3IF	PWM3 中断标志寄存器	FC05H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWM3FDCR	PWM3 异常检测控制寄存器	FC06H	INVCM	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF	0000,0000
PWM30CR	PWM30 控制寄存器	FC14H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM31CR	PWM31 控制寄存器	FC1CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM32CR	PWM32 控制寄存器	FC24H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM33CR	PWM33 控制寄存器	FC2CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM34CR	PWM34 控制寄存器	FC34H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM35CR	PWM35 控制寄存器	FC3CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM36CR	PWM36 控制寄存器	FC44H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM37CR	PWM37 控制寄存器	FC4CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM4IF	PWM4 中断标志寄存器	FC55H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWM4FDCR	PWM4 异常检测控制寄存器	FC56H	INVCM	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	0000,0000
PWM40CR	PWM40 控制寄存器	FC64H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM41CR	PWM41 控制寄存器	FC6CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM42CR	PWM42 控制寄存器	FC74H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM43CR	PWM43 控制寄存器	FC7CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM44CR	PWM44 控制寄存器	FC84H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000

PWM45CR	PWM45 控制寄存器	FC8CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM46CR	PWM46 控制寄存器	FC94H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM47CR	PWM47 控制寄存器	FC9CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM51IF	PWM5 中断标志寄存器	FCA5H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWM5FDCR	PWM5 异常检测控制寄存器	FCA6H	INVCM	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF	0000,0000
PWM50CR	PWM50 控制寄存器	FCB4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM51CR	PWM51 控制寄存器	FCBCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM52CR	PWM52 控制寄存器	FCC4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM53CR	PWM53 控制寄存器	FCCCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM54CR	PWM54 控制寄存器	FCD4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM55CR	PWM55 控制寄存器	FCDCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM56CR	PWM56 控制寄存器	FCE4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM57CR	PWM57 控制寄存器	FCECH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
TSSTA2	触摸按键状态寄存器 2	FB47H	TSIF	TSDOV	-	-		TSDNCHN[3:0]			00xx,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	ETKSUI	ET4	ET3	ES4	ES3	ET2	ESPI	ES2

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

ETKSUI: 触摸按键中断允许位。

- 0: 禁止触摸按键中断
1: 允许触摸按键中断

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]			ECF
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

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

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

I²C 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSCR	FE81H	EMSI	-	-	-	MSCMD[3:0]			
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

增强型 PWM 配置寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMCFG01	F6H	PWM1CBIF	EPWM1CBI	FLTPS0	PWM1CEN	PWM0CBIF	EPWM0CBI	ENPWM0TA	PWM0CEN
PWMCFG23	F7H	PWM3CBIF	EPWM3CBI	FLTPS1	PWM3CEN	PWM2CBIF	EPWM2CBI	ENPWM2TA	PWM2CEN
PWMCFG45	FEH	PWM5CBIF	EPWM5CBI	FLTPS2	PWM5CEN	PWM4CBIF	EPWM4CBI	ENPWM4TA	PWM4CEN

EPWM0CBI: 增强PWM0计数器中断允许位。

0: 禁止 PWM0 计数器中断
1: 允许 PWM0 计数器中断

EPWM1CBI: 增强PWM1计数器中断允许位。

0: 禁止 PWM1 计数器中断
1: 允许 PWM1 计数器中断

EPWM2CBI: 增强PWM2计数器中断允许位。

0: 禁止 PWM2 计数器中断
1: 允许 PWM2 计数器中断

EPWM3CBI: 增强PWM3计数器中断允许位。

0: 禁止 PWM3 计数器中断
1: 允许 PWM3 计数器中断

EPWM4CBI: 增强PWM4计数器中断允许位。

0: 禁止 PWM4 计数器中断
1: 允许 PWM4 计数器中断

EPWM5CBI: 增强PWM5计数器中断允许位。

0: 禁止 PWM5 计数器中断
1: 允许 PWM5 计数器中断

增强型 PWM 异常检测控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWM0FDCR	FF06H	INV_CMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF
PWM1FDCR	FF56H	INV_CMP	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF
PWM2FDCR	FFA6H	INV_CMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF
PWM3FDCR	FC06H	INV_CMP	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF
PWM4FDCR	FC56H	INV_CMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF
PWM5FDCR	FCA6H	INV_CMP	INVIO	ENFD	FLTFLIO	-	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
PWM00CR	FF14H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM01CR	FF1CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM02CR	FF24H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM03CR	FF2CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM04CR	FF34H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM05CR	FF3CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM06CR	FF44H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM07CR	FF4CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM10CR	FF64H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM11CR	FF6CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM12CR	FF74H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM13CR	FF7CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM14CR	FF84H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM15CR	FF8CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM16CR	FF94H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM17CR	FF9CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM20CR	FFB4H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM21CR	FFBCH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM22CR	FFC4H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM23CR	FFCCH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM24CR	FFD4H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM25CR	FFDCH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM26CR	FFE4H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM27CR	FFECH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM30CR	FC14H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM31CR	FC1CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM32CR	FC24H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM33CR	FC2CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM34CR	FC34H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM35CR	FC3CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM36CR	FC44H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM37CR	FC4CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM40CR	FC64H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM41CR	FC6CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM42CR	FC74H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM43CR	FC7CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM44CR	FC84H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM45CR	FC8CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM46CR	FC94H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM47CR	FC9CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM50CR	FCB4H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM51CR	FCBCH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM52CR	FCC4H	ENO	INI	-	-		ENI	ENT2I	ENT1I

PWM53CR	FCCCH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM54CR	FCD4H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM55CR	FCDCDH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM56CR	FCE4H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM57CR	FCECH	ENO	INI	-	-		ENI	ENT2I	ENT1I

ECI: PWM通道中断允许位。

0: 禁止 PWM 中断

1: 允许 PWM 中断

ET2SI: PWM通道第2个触发点中断允许位。

0: 禁止 PWM 的第 2 个触发点中断

1: 允许 PWM 的第 2 个触发点中断

ET1SI: PWM通道第1个触发点中断允许位。

0: 禁止 PWM 的第 1 个触发点中断

1: 允许 PWM 的第 1 个触发点中断

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。中断服务程序中硬件自动清零。

INT3IF: 外部中断3中断请求标志。中断服务程序中硬件自动清零。

INT2IF: 外部中断2中断请求标志。中断服务程序中硬件自动清零。

T4IF: 定时器4溢出中断标志。中断服务程序中硬件自动清零（注意：此位为只写寄存器，不可读）。

T3IF: 定时器3溢出中断标志。中断服务程序中硬件自动清零（注意：此位为只写寄存器，不可读）。

T2IF: 定时器2溢出中断标志。中断服务程序中硬件自动清零（注意：此位为只写寄存器，不可读）。

注意：

早期采用 0.35um 工艺的 1T 8051，STC15 系列增加了 16 位重装载定时器，全球 8051 首次大手笔，由于制造成本高，STC 可 16 位重装载的定时器 2/3/4 没有设计用户可以访问的中断请求标志位寄存器，只有内部隐藏的标志位，提供给用户软件清内部隐藏标志位的方法是：用户软件禁止定时器 2/3/4 中断时，硬件自动清定时器 2/3/4 内部隐藏中断请求标志位。

为了产品的一致性：

采用 0.18um 工艺的 STC8A/ STC8F 及后续 STC8G/STC8H/ STC8C/ STC12H 系列虽然增加了定时器 2/3/4 的用户可以访问的中断请求标志位寄存器, 但禁止定时器 2/3/4 中断时, 硬件自动清定时器 2/3/4 内部隐藏中断请求标志位的功能依然保留了。所以在定时器 2/3/4 没有停止计数时不要随意禁止定时器 2/3/4 中断, 否则实际起作用的隐藏的中断请求标志位会被清除掉, 会有可能, 计数器又溢出后, 又产生了隐藏的中断请求标志位被置 1 后, 去请求中断并在等待时, 却被用户误清除的事。

这与传统的 INTEL8048, 8051 不一样, 但 INTEL 已停产, 所以 STC 的新设计并没有考虑兼容传统 INTEL 的规格。

这是中国 STC 对 8051 的再发展。

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_EPWMT	ADC_CHS[3:0]			

ADC_FLAG: ADC转换完成中断请求标志。需要软件清零。

SPI 状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPSTAT	CDH	SPIF	WCOL	-	-	-	-	-	-

SPIF: SPI数据传输完成中断请求标志。需要软件清零。

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
CMPCCR1	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。

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。

增强型 PWM 配置寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMCFG01	F6H	PWM1CBIF	EPWM1CBI	FLTPS0	PWM1CEN	PWM0CBIF	EPWM0CBI	ENPWM0TA	PWM0CEN
PWMCFG23	F7H	PWM3CBIF	EPWM3CBI	FLTPS1	PWM3CEN	PWM2CBIF	EPWM2CBI	ENPWM2TA	PWM2CEN
PWMCFG45	FEH	PWM5CBIF	EPWM5CBI	FLTPS2	PWM5CEN	PWM4CBIF	EPWM4CBI	ENPWM4TA	PWM4CEN

PWM0CBIF: 增强型PWM0计数器中断请求标志。需要软件清零。

PWM1CBIF: 增强型PWM1计数器中断请求标志。需要软件清零。

PWM2CBIF: 增强型PWM2计数器中断请求标志。需要软件清零。

PWM3CBIF: 增强型PWM3计数器中断请求标志。需要软件清零。

PWM4CBIF: 增强型PWM4计数器中断请求标志。需要软件清零。

PWM5CBIF: 增强型PWM5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
PWM0IF	FF05H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF
PWM1IF	FF55H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF
PWM2IF	FFA5H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF

PWM3IF	FC05H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF
PWM4IF	FC55H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF
PWM5IF	FCA5H	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 异常检测控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWM0FDCR	FF06H	INVCM	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF
PWM1FDCR	FF56H	INVCM	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF
PWM2FDCR	FFA6H	INVCM	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF
PWM3FDCR	FC06H	INVCM	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF
PWM4FDCR	FC56H	INVCM	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF
PWM5FDCR	FCA6H	INVCM	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF

FDIF: 增强型PWM异常检测中断请求标志。需要软件清零。

触摸按键状态寄存器 2

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TSSTA2	FB47H	TSIF	TSDOV	-	-	TSDNCHN[3:0]			

TSIF: 触摸按键中断标志。需软件写1清零。

11.4.3 中断优先级寄存器

除 INT2、INT3、定时器 2、定时器 3 和定时器 4 外，其他中断均有 4 级中断优先级可设置

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	PPWM2FD PTKSU	PI2C	PCMP	PX4	PPWM0FD	PPWM0	PSPI	PS2
IP2H	B6H	PPWM2FDH PTKSUH	PI2CH	PCMHP	PX4H	PPWM0FDH	PPWM0H	PSPIH	PS2H

IP3	DFH	PPWM4FD	PPWM5	PPWM4	PPWM3	PPWM2	PPWM1	PS4	PS3
IP3H	EEH	PPWM4FDH	PPWM5H	PPWM4H	PPWM3H	PPWM2H	PPWM1H	PS4H	PS3H

PX0H,PX0: 外部中断0中断优先级控制位

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

PT0H,PT0: 定时器0中断优先级控制位

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

PX1H,PX1: 外部中断1中断优先级控制位

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

PT1H,PT1: 定时器1中断优先级控制位

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

PSH,PS: 串口1中断优先级控制位

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

PADCH,PADC: ADC中断优先级控制位

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

PLVDH,PLVD: 低压检测中断优先级控制位

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

PPCAH,PPCA: CCP/PCA/PWM中断优先级控制位

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

PS2H,PS2: 串口2中断优先级控制位

- 00: 串口 2 中断优先级为 0 级 (最低级)

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

PS3H,PS3: 串口3中断优先级控制位

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

PS4H,PS4: 串口4中断优先级控制位

- 00: 串口 4 中断优先级为 0 级 (最低级)
- 01: 串口 4 中断优先级为 1 级 (较低级)
- 10: 串口 4 中断优先级为 2 级 (较高级)
- 11: 串口 4 中断优先级为 3 级 (最高级)

PSPIH,PSPI: SPI中断优先级控制位

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

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)

PPWM0H,PPWM0: 增强型PWM0中断优先级控制位

- 00: 增强型 PWM0 中断优先级为 0 级 (最低级)
- 01: 增强型 PWM0 中断优先级为 1 级 (较低级)
- 10: 增强型 PWM0 中断优先级为 2 级 (较高级)
- 11: 增强型 PWM0 中断优先级为 3 级 (最高级)

PPWM1H,PPWM1: 增强型PWM1中断优先级控制位

- 00: 增强型 PWM1 中断优先级为 0 级 (最低级)
- 01: 增强型 PWM1 中断优先级为 1 级 (较低级)
- 10: 增强型 PWM1 中断优先级为 2 级 (较高级)
- 11: 增强型 PWM1 中断优先级为 3 级 (最高级)

PPWM2H,PPWM2: 增强型PWM2中断优先级控制位

- 00: 增强型 PWM2 中断优先级为 0 级 (最低级)
- 01: 增强型 PWM2 中断优先级为 1 级 (较低级)
- 10: 增强型 PWM2 中断优先级为 2 级 (较高级)
- 11: 增强型 PWM2 中断优先级为 3 级 (最高级)

PPWM3H,PPWM3: 增强型PWM3中断优先级控制位

- 00: 增强型 PWM3 中断优先级为 0 级 (最低级)
- 01: 增强型 PWM3 中断优先级为 1 级 (较低级)
- 10: 增强型 PWM3 中断优先级为 2 级 (较高级)
- 11: 增强型 PWM3 中断优先级为 3 级 (最高级)

PPWM4H,PPWM4: 增强型PWM4中断优先级控制位

- 00: 增强型 PWM4 中断优先级为 0 级 (最低级)
- 01: 增强型 PWM4 中断优先级为 1 级 (较低级)
- 10: 增强型 PWM4 中断优先级为 2 级 (较高级)
- 11: 增强型 PWM4 中断优先级为 3 级 (最高级)

PPWM5H,PPWM5: 增强型PWM5中断优先级控制位

- 00: 增强型 PWM5 中断优先级为 0 级 (最低级)
- 01: 增强型 PWM5 中断优先级为 1 级 (较低级)
- 10: 增强型 PWM5 中断优先级为 2 级 (较高级)
- 11: 增强型 PWM5 中断优先级为 3 级 (最高级)

PPWM0FDH,PPWM0FD: 增强型PWM0异常检测中断优先级控制位

- 00: PWM0FD 中断优先级为 0 级 (最低级)
- 01: PWM0FD 中断优先级为 1 级 (较低级)
- 10: PWM0FD 中断优先级为 2 级 (较高级)
- 11: PWM0FD 中断优先级为 3 级 (最高级)

PPWM2FDH,PPWM2FD: 增强型PWM2异常检测中断优先级控制位

- 00: PWM2FD 中断优先级为 0 级 (最低级)
- 01: PWM2FD 中断优先级为 1 级 (较低级)
- 10: PWM2FD 中断优先级为 2 级 (较高级)
- 11: PWM2FD 中断优先级为 3 级 (最高级)

PPWM4FDH,PPWM4FD: 增强型PWM4异常检测中断优先级控制位

- 00: PWM4FD 中断优先级为 0 级 (最低级)
- 01: PWM4FD 中断优先级为 1 级 (较低级)
- 10: PWM4FD 中断优先级为 2 级 (较高级)
- 11: PWM4FD 中断优先级为 3 级 (最高级)

PTKSUH,PTKSU: 触摸按键中断优先级控制位

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

11.5 范例程序

11.5.1 INT0 中断（上升沿和下降沿），可同时支持上升沿和下降沿

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

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

void INT0_Isr() interrupt 0
{
    if (P32)                                //判断上升沿和下降沿
    {
        P10 = !P10;                          //测试端口
    }
    else
    {
        P11 = !P11;                          //测试端口
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IT0 = 0;                                //使能 INT0 上升沿和下降沿中断
    EX0 = 1;                                //使能 INT0 中断
```

```

EA = I;
while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

```

P0M1      DATA      093H
P0M0      DATA      094H
P1M1      DATA      091H
P1M0      DATA      092H
P2M1      DATA      095H
P2M0      DATA      096H
P3M1      DATA      0B1H
P3M0      DATA      0B2H
P4M1      DATA      0B3H
P4M0      DATA      0B4H
P5M1      DATA      0C9H
P5M0      DATA      0CAH

        ORG      0000H
        LJMP    MAIN
        ORG      0003H
        LJMP    INT0ISR

        ORG      0100H
INT0ISR:
        JB      P3.2,RISING
        CPL     P1.0          ; 判断上升沿和下降沿
        RETI

RISING:
        CPL     P1.1          ; 测试端口
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        CLR     IT0           ; 使能 INT0 上升沿和下降沿中断
        SETB   EX0           ; 使能 INT0 中断
        SETB   EA
        JMP    $

END

```

11.5.2 INT0 中断（下降沿）

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

sbit P10      = P1^0;

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

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IT0 = 1; //使能 INT0 下降沿中断
    EX0 = 1; //使能 INT0 中断
    EA = 1;

    while (1);
}
```

汇编代码

;测试工作频率为 11.0592MHz

P0M1 DATA 093H

<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</i>
	<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>CPL</i>	<i>P1.0</i> ; 测试端口
	<i>RETI</i>	
<i>MAIN:</i>		
	<i>MOV</i>	<i>SP, #5FH</i>
	<i>MOV</i>	<i>P0M0, #00H</i>
	<i>MOV</i>	<i>P0M1, #00H</i>
	<i>MOV</i>	<i>P1M0, #00H</i>
	<i>MOV</i>	<i>P1M1, #00H</i>
	<i>MOV</i>	<i>P2M0, #00H</i>
	<i>MOV</i>	<i>P2M1, #00H</i>
	<i>MOV</i>	<i>P3M0, #00H</i>
	<i>MOV</i>	<i>P3M1, #00H</i>
	<i>MOV</i>	<i>P4M0, #00H</i>
	<i>MOV</i>	<i>P4M1, #00H</i>
	<i>MOV</i>	<i>P5M0, #00H</i>
	<i>MOV</i>	<i>P5M1, #00H</i>
	<i>SETB</i>	<i>IT0</i> ; 使能 INT0 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>	

11.5.3 INT1 中断（上升沿和下降沿），可同时支持上升沿和下降沿

C 语言代码

//测试工作频率为 11.0592MHz

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

sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
```

```

sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

sbit PI0       = P1^0;
sbit PI1       = 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()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

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

    while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H

<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</i>
	<i>ORG</i>	<i>0000H</i>
	<i>LJMP</i>	<i>MAIN</i>
	<i>ORG</i>	<i>0013H</i>
	<i>LJMP</i>	<i>INTIISR</i>
	<i>ORG</i>	<i>0100H</i>
<i>INTIISR:</i>	<i>JB</i>	<i>INTI,RISING</i>
	<i>CPL</i>	<i>P1.0</i> ;判断上升沿和下降沿
	<i>RETI</i>	<i>P1.0</i> ;测试端口
<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, #5FH</i>
	<i>MOV</i>	<i>P0M0, #00H</i>
	<i>MOV</i>	<i>P0M1, #00H</i>
	<i>MOV</i>	<i>P1M0, #00H</i>
	<i>MOV</i>	<i>P1M1, #00H</i>
	<i>MOV</i>	<i>P2M0, #00H</i>
	<i>MOV</i>	<i>P2M1, #00H</i>
	<i>MOV</i>	<i>P3M0, #00H</i>
	<i>MOV</i>	<i>P3M1, #00H</i>
	<i>MOV</i>	<i>P4M0, #00H</i>
	<i>MOV</i>	<i>P4M1, #00H</i>
	<i>MOV</i>	<i>P5M0, #00H</i>
	<i>MOV</i>	<i>P5M1, #00H</i>
	<i>CLR</i>	<i>IT1</i> ;使能INT1 上升沿和下降沿中断
	<i>SETB</i>	<i>EX1</i> ;使能INT1 中断
	<i>SETB</i>	<i>EA</i>
	<i>JMP</i>	<i>\$</i>
	<i>END</i>	

11.5.4 INT1 中断（下降沿）

C 语言代码

//测试工作频率为 11.0592MHz

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

```
sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
```

```

sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

sbit P10      = P1^0;

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

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IT1 = 1;                                    // 使能INT1 下降沿中断
    EX1 = 1;                                    // 使能INT1 中断
    EA = 1;

    while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

```

P0M1      DATA    093H
P0M0      DATA    094H
P1M1      DATA    091H
P1M0      DATA    092H
P2M1      DATA    095H
P2M0      DATA    096H
P3M1      DATA    0B1H
P3M0      DATA    0B2H
P4M1      DATA    0B3H
P4M0      DATA    0B4H
P5M1      DATA    0C9H
P5M0      DATA    0CAH

        ORG     0000H
        LJMP   MAIN
        ORG     0013H
        LJMP   INT1ISR

        ORG     0100H

```

INTIISR:

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

MAIN:

<i>MOV</i>	<i>SP, #5FH</i>	
<i>MOV</i>	<i>P0M0, #00H</i>	
<i>MOV</i>	<i>P0M1, #00H</i>	
<i>MOV</i>	<i>P1M0, #00H</i>	
<i>MOV</i>	<i>P1M1, #00H</i>	
<i>MOV</i>	<i>P2M0, #00H</i>	
<i>MOV</i>	<i>P2M1, #00H</i>	
<i>MOV</i>	<i>P3M0, #00H</i>	
<i>MOV</i>	<i>P3M1, #00H</i>	
<i>MOV</i>	<i>P4M0, #00H</i>	
<i>MOV</i>	<i>P4M1, #00H</i>	
<i>MOV</i>	<i>P5M0, #00H</i>	
<i>MOV</i>	<i>P5M1, #00H</i>	
<i>SETB</i>	<i>IT1</i>	; 使能 INT1 下降沿中断
<i>SETB</i>	<i>EX1</i>	; 使能 INT1 中断
<i>SETB</i>	<i>EA</i>	
<i>JMP</i>	<i>\$</i>	

END

11.5.5 INT2 中断（下降沿），只支持下降沿中断

C 语言代码

// 测试工作频率为 11.0592MHz

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

sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P4M1 = 0xb3;
sfr P4M0 = 0xb4;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;

sfr 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()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    INTCLKO = EX2;                            //使能INT2 中断
    EA = 1;

    while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

```

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

P0M1        DATA      093H
P0M0        DATA      094H
P1M1        DATA      091H
P1M0        DATA      092H
P2M1        DATA      095H
P2M0        DATA      096H
P3M1        DATA      0B1H
P3M0        DATA      0B2H
P4M1        DATA      0B3H
P4M0        DATA      0B4H
P5M1        DATA      0C9H
P5M0        DATA      0CAH

        ORG      0000H
        LJMP    MAIN
        ORG      0053H
        LJMP    INT2ISR

        ORG      0100H
INT2ISR:
        CPL     P1.0          ; 测试端口
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H

```

```

MOV      P0M1, #00H
MOV      P1M0, #00H
MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

MOV      INTCLKO,#EX2          ;使能INT2 中断
SETB    EA
JMP     $

END

```

11.5.6 INT3 中断（下降沿），只支持下降沿中断

C 语言代码

//测试工作频率为 11.0592MHz

```

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

sfr    P0M1      = 0x93;
sfr    P0M0      = 0x94;
sfr    P1M1      = 0x91;
sfr    P1M0      = 0x92;
sfr    P2M1      = 0x95;
sfr    P2M0      = 0x96;
sfr    P3M1      = 0xb1;
sfr    P3M0      = 0xb2;
sfr    P4M1      = 0xb3;
sfr    P4M0      = 0xb4;
sfr    P5M1      = 0xc9;
sfr    P5M0      = 0xca;

sfr    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()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
}

```

```

P1M1 = 0x00;
P2M0 = 0x00;
P2M1 = 0x00;
P3M0 = 0x00;
P3M1 = 0x00;
P4M0 = 0x00;
P4M1 = 0x00;
P5M0 = 0x00;
P5M1 = 0x00;

INTCLKO = EX3;                                //使能INT3 中断
EA = 1;

while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

```

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

P0M1        DATA      093H
P0M0        DATA      094H
P1M1        DATA      091H
P1M0        DATA      092H
P2M1        DATA      095H
P2M0        DATA      096H
P3M1        DATA      0B1H
P3M0        DATA      0B2H
P4M1        DATA      0B3H
P4M0        DATA      0B4H
P5M1        DATA      0C9H
P5M0        DATA      0CAH

        ORG      0000H
        LJMP    MAIN
        ORG      005BH
        LJMP    INT3ISR

        ORG      0100H
INT3ISR:
        CPL     P1.0           ; 测试端口
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H

```

```

MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

MOV      INTCLKO,#EX3          ;使能INT3 中断
SETB    EA
JMP     $

END

```

11.5.7 INT4 中断（下降沿），只支持下降沿中断

C 语言代码

//测试工作频率为 11.0592MHz

```

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

sfr    P0M1      = 0x93;
sfr    P0M0      = 0x94;
sfr    P1M1      = 0x91;
sfr    P1M0      = 0x92;
sfr    P2M1      = 0x95;
sfr    P2M0      = 0x96;
sfr    P3M1      = 0xb1;
sfr    P3M0      = 0xb2;
sfr    P4M1      = 0xb3;
sfr    P4M0      = 0xb4;
sfr    P5M1      = 0xc9;
sfr    P5M0      = 0xca;

sfr    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()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
}

```

```

P5M1 = 0x00;

INTCLKO = EX4;                                //使能INT4 中断
EA = I;

while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

```

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

P0M1        DATA      093H
P0M0        DATA      094H
P1M1        DATA      091H
P1M0        DATA      092H
P2M1        DATA      095H
P2M0        DATA      096H
P3M1        DATA      0B1H
P3M0        DATA      0B2H
P4M1        DATA      0B3H
P4M0        DATA      0B4H
P5M1        DATA      0C9H
P5M0        DATA      0CAH

ORG         0000H
LJMP        MAIN
ORG         0083H
LJMP        INT4ISR

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

MAIN:
MOV         SP, #5FH
MOV         P0M0, #00H
MOV         P0M1, #00H
MOV         P1M0, #00H
MOV         P1M1, #00H
MOV         P2M0, #00H
MOV         P2M1, #00H
MOV         P3M0, #00H
MOV         P3M1, #00H
MOV         P4M0, #00H
MOV         P4M1, #00H
MOV         P5M0, #00H
MOV         P5M1, #00H

MOV         INTCLKO,#EX4      ; 使能INT4 中断
SETB        EA
JMP         $

```

END

11.5.8 定时器 0 中断

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

sbit P10      = P1^0;

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

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

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

    while (1);
}
```

汇编代码

;测试工作频率为 11.0592MHz

```

P0M1      DATA      093H
P0M0      DATA      094H
P1M1      DATA      091H
P1M0      DATA      092H
P2M1      DATA      095H
P2M0      DATA      096H
P3M1      DATA      0B1H
P3M0      DATA      0B2H
P4M1      DATA      0B3H
P4M0      DATA      0B4H
P5M1      DATA      0C9H
P5M0      DATA      0CAH

        ORG      0000H
        LJMP    MAIN
        ORG      000BH
        LJMP    TM0ISR

TM0ISR:
        ORG      0100H
        CPL     P1.0
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     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

```

11.5.9 定时器 1 中断

C 语言代码

//测试工作频率为 11.0592MHz

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

sfr P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

sbit P10      = P1^0;

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

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x00; //5536-11.0592M/12/1000
    TL1 = 0x66;
    TH1 = 0xfc;
    TR1 = 1; //启动定时器
    ET1 = 1; //使能定时器中断
    EA = 1;

    while (1);
}

```

汇编代码

;测试工作频率为 11.0592MHz

P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H

<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</i>
	<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>ORG</i>	<i>0100H</i>
TMIISR:	<i>CPL</i>	<i>PI.0</i>
	<i>RETI</i>	; 测试端口
MAIN:		
	<i>MOV</i>	<i>SP, #5FH</i>
	<i>MOV</i>	<i>P0M0, #00H</i>
	<i>MOV</i>	<i>P0M1, #00H</i>
	<i>MOV</i>	<i>P1M0, #00H</i>
	<i>MOV</i>	<i>P1M1, #00H</i>
	<i>MOV</i>	<i>P2M0, #00H</i>
	<i>MOV</i>	<i>P2M1, #00H</i>
	<i>MOV</i>	<i>P3M0, #00H</i>
	<i>MOV</i>	<i>P3M1, #00H</i>
	<i>MOV</i>	<i>P4M0, #00H</i>
	<i>MOV</i>	<i>P4M1, #00H</i>
	<i>MOV</i>	<i>P5M0, #00H</i>
	<i>MOV</i>	<i>P5M1, #00H</i>
	<i>MOV</i>	<i>TMOD, #00H</i>
	<i>MOV</i>	<i>TL1, #66H</i>
	<i>MOV</i>	; 65536-11.0592M/12/1000
	<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>JMP</i>	<i>EA</i>
	<i>JMP</i>	\$
 END		

11.5.10 定时器 2 中断

C 语言代码

```
// 测试工作频率为 11.0592MHz
```

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

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

```

sfr AUXR      = 0x8e;
sfr IE2       = 0xaf;
#define ET2        0x04
sfr AUXINTIF   = 0xef;
#define T2IF      0x01

sfr P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

sbit P10      = P1^0;

void TM2_Isr() interrupt 12
{
    P10 = !P10;                                //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    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);
}

```

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

<i>AUXINTIF</i>	<i>DATA</i>	<i>0EFH</i>
<i>T2IF</i>	<i>EQU</i>	<i>01H</i>
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</i>
	<i>ORG</i>	<i>0000H</i>
	<i>LJMP</i>	<i>MAIN</i>
	<i>ORG</i>	<i>0063H</i>
	<i>LJMP</i>	<i>TM2ISR</i>
	<i>ORG</i>	<i>0100H</i>
<i>TM2ISR:</i>	<i>CPL</i>	<i>P1.0</i>
	<i>RETI</i>	; 测试端口
<i>MAIN:</i>		
	<i>MOV</i>	<i>SP, #5FH</i>
	<i>MOV</i>	<i>P0M0, #00H</i>
	<i>MOV</i>	<i>P0M1, #00H</i>
	<i>MOV</i>	<i>P1M0, #00H</i>
	<i>MOV</i>	<i>P1M1, #00H</i>
	<i>MOV</i>	<i>P2M0, #00H</i>
	<i>MOV</i>	<i>P2M1, #00H</i>
	<i>MOV</i>	<i>P3M0, #00H</i>
	<i>MOV</i>	<i>P3M1, #00H</i>
	<i>MOV</i>	<i>P4M0, #00H</i>
	<i>MOV</i>	<i>P4M1, #00H</i>
	<i>MOV</i>	<i>P5M0, #00H</i>
	<i>MOV</i>	<i>P5M1, #00H</i>
	<i>MOV</i>	<i>T2L, #66H</i>
	<i>MOV</i>	; 65536-11.0592M/12/1000
	<i>MOV</i>	<i>T2H, #0FCH</i>
	<i>MOV</i>	; 启动定时器
	<i>MOV</i>	<i>AUXR, #10H</i>
	<i>MOV</i>	; 使能定时器中断
	<i>SETB</i>	<i>IE2, #ET2</i>
		<i>EA</i>
	<i>JMP</i>	\$
	<i>END</i>	

11.5.11 定时器 3 中断

C 语言代码

// 测试工作频率为 11.0592MHz

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

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

sfr P1M1     = 0x91;
sfr P1M0     = 0x92;
sfr P0M1     = 0x93;
sfr P0M0     = 0x94;
sfr P2M1     = 0x95;
sfr P2M0     = 0x96;
sfr P3M1     = 0xb1;
sfr P3M0     = 0xb2;
sfr P4M1     = 0xb3;
sfr P4M0     = 0xb4;
sfr P5M1     = 0xc9;
sfr P5M0     = 0xca;

sbit P10      = P1^0;

void TM3_Isr() interrupt 19
{
    P10 = !P10; // 测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    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);
}
```

汇编代码

; 测试工作频率为 11.0592MHz

<i>T3L</i>	<i>DATA</i>	<i>0D5H</i>
<i>T3H</i>	<i>DATA</i>	<i>0D4H</i>
<i>T4T3M</i>	<i>DATA</i>	<i>0DIH</i>
<i>IE2</i>	<i>DATA</i>	<i>0AFH</i>
<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>PIM1</i>	<i>DATA</i>	<i>091H</i>
<i>PIM0</i>	<i>DATA</i>	<i>092H</i>
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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>
	<i>RETI</i>	; 测试端口
<i>MAIN:</i>		
	<i>MOV</i>	<i>SP, #5FH</i>
	<i>MOV</i>	<i>P0M0, #00H</i>
	<i>MOV</i>	<i>P0M1, #00H</i>
	<i>MOV</i>	<i>P1M0, #00H</i>
	<i>MOV</i>	<i>P1M1, #00H</i>
	<i>MOV</i>	<i>P2M0, #00H</i>
	<i>MOV</i>	<i>P2M1, #00H</i>
	<i>MOV</i>	<i>P3M0, #00H</i>
	<i>MOV</i>	<i>P3M1, #00H</i>
	<i>MOV</i>	<i>P4M0, #00H</i>
	<i>MOV</i>	<i>P4M1, #00H</i>
	<i>MOV</i>	<i>P5M0, #00H</i>
	<i>MOV</i>	<i>P5M1, #00H</i>
	<i>MOV</i>	<i>T3L,#66H</i>
	<i>MOV</i>	<i>T3H,#0FCH</i>
	<i>MOV</i>	<i>T4T3M,#08H</i>
	<i>MOV</i>	; 启动定时器
	<i>IE2,#ET3</i>	; 使能定时器中断
	<i>SETB</i>	<i>EA</i>
	<i>JMP</i>	\$
	<i>END</i>	

11.5.12 定时器 4 中断

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr T3L      = 0xd5;
sfr T3H      = 0xd4;
sfr T4L      = 0xd3;
sfr T4H      = 0xd2;
sfr T4T3M    = 0xd1;
sfr IE2      = 0xaf;
#define ET3      0x20
#define ET4      0x40
sfr AUXINTIF = 0xef;
#define T3IF     0x02
#define T4IF     0x04

sfr P1M1     = 0x91;
sfr P1M0     = 0x92;
sfr P0M1     = 0x93;
sfr P0M0     = 0x94;
sfr P2M1     = 0x95;
sfr P2M0     = 0x96;
sfr P3M1     = 0xb1;
sfr P3M0     = 0xb2;
sfr P4M1     = 0xb3;
sfr P4M0     = 0xb4;
sfr P5M1     = 0xc9;
sfr P5M0     = 0xca;

sbit P10      = P1^0;

void TM4_Isr() interrupt 20
{
    P10 = !P10; // 测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    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 = I;

while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

```

T3L      DATA      0D5H
T3H      DATA      0D4H
T4L      DATA      0D3H
T4H      DATA      0D2H
T4T3M    DATA      0D1H
IE2      DATA      0AFH
ET3      EQU       20H
ET4      EQU       40H
AUXINTIF DATA      0EFH
T3IF     EQU       02H
T4IF     EQU       04H

P1M1     DATA      091H
P1M0     DATA      092H
P0M1     DATA      093H
P0M0     DATA      094H
P2M1     DATA      095H
P2M0     DATA      096H
P3M1     DATA      0B1H
P3M0     DATA      0B2H
P4M1     DATA      0B3H
P4M0     DATA      0B4H
P5M1     DATA      0C9H
P5M0     DATA      0CAH

ORG      0000H
LJMP    MAIN
ORG      00A3H
LJMP    TM4ISR

ORG      0100H
TM4ISR:
CPL     P1.0          ; 测试端口
RETI

MAIN:
MOV     SP, #5FH
MOV     P0M0, #00H
MOV     P0M1, #00H
MOV     P1M0, #00H
MOV     P1M1, #00H
MOV     P2M0, #00H
MOV     P2M1, #00H
MOV     P3M0, #00H
MOV     P3M1, #00H
MOV     P4M0, #00H
MOV     P4M1, #00H
MOV     P5M0, #00H

```

```

MOV      P5M1, #00H
MOV      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

```

11.5.13 UART1 中断

C 语言代码

//测试工作频率为11.0592MHz

```

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

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

sfr    P0M1     = 0x93;
sfr    P0M0     = 0x94;
sfr    P1M1     = 0x91;
sfr    P1M0     = 0x92;
sfr    P2M1     = 0x95;
sfr    P2M0     = 0x96;
sfr    P3M1     = 0xb1;
sfr    P3M0     = 0xb2;
sfr    P4M1     = 0xb3;
sfr    P4M0     = 0xb4;
sfr    P5M1     = 0xc9;
sfr    P5M0     = 0xca;

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
        P11 = !P11;                            //测试端口
    }
}

void main()

```

```

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

    SCON = 0x50;
    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;                                     //发送测试数据
    SBUF = 0x5a;

    while (1);
}

```

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>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</i>
<i>ORG</i>		<i>0000H</i>
<i>LJMP</i>		<i>MAIN</i>
<i>ORG</i>		<i>0023H</i>
<i>LJMP</i>		<i>UARTIISR</i>
<i>ORG</i>		<i>0100H</i>
<i>UARTIISR:</i>		
<i>JNB</i>	<i>TI,CHECKRI</i>	
<i>CLR</i>	<i>TI</i>	; 清中断标志
<i>CPL</i>	<i>P1.0</i>	; 测试端口
<i>CHECKRI:</i>		
<i>JNB</i>	<i>RI,ISREXIT</i>	
<i>CLR</i>	<i>RI</i>	; 清中断标志

<i>CPL</i>	<i>P1.1</i>	
<i>ISREXIT:</i>		
<i>RETI</i>		
<i>MAIN:</i>		
<i>MOV</i>	<i>SP, #5FH</i>	
<i>MOV</i>	<i>P0M0, #00H</i>	
<i>MOV</i>	<i>P0M1, #00H</i>	
<i>MOV</i>	<i>P1M0, #00H</i>	
<i>MOV</i>	<i>P1M1, #00H</i>	
<i>MOV</i>	<i>P2M0, #00H</i>	
<i>MOV</i>	<i>P2M1, #00H</i>	
<i>MOV</i>	<i>P3M0, #00H</i>	
<i>MOV</i>	<i>P3M1, #00H</i>	
<i>MOV</i>	<i>P4M0, #00H</i>	
<i>MOV</i>	<i>P4M1, #00H</i>	
<i>MOV</i>	<i>P5M0, #00H</i>	
<i>MOV</i>	<i>P5M1, #00H</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>;启动定时器</i>
<i>SETB</i>	<i>ES</i>	<i>;使能串口中断</i>
<i>SETB</i>	<i>EA</i>	
<i>MOV</i>	<i>SBUF, #5AH</i>	<i>;发送测试数据</i>
<i>JMP</i>	\$	
<i>END</i>		

11.5.14 UART2 中断

C 语言代码

//测试工作频率为 11.0592MHz

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

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

sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
```

```

sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

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

    S2CON = 0x10;                                //65536-11059200/115200/4=0FFE8H
    T2L = 0xe8;
    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
    S2BUF = 0x5a;

    while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

T2L	DATA	0D7H
T2H	DATA	0D6H
AUXR	DATA	8EH
S2CON	DATA	9AH
S2BUF	DATA	9BH
IE2	DATA	0AFH
ES2	EQU	01H

<i>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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.I,CHECKRI</i>
	<i>ANL</i>	<i>S2CON,#NOT 02H</i>
	<i>CPL</i>	<i>PI.2</i> ;清中断标志
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>PI.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, #5FH</i>
	<i>MOV</i>	<i>P0M0, #00H</i>
	<i>MOV</i>	<i>P0M1, #00H</i>
	<i>MOV</i>	<i>P1M0, #00H</i>
	<i>MOV</i>	<i>P1M1, #00H</i>
	<i>MOV</i>	<i>P2M0, #00H</i>
	<i>MOV</i>	<i>P2M1, #00H</i>
	<i>MOV</i>	<i>P3M0, #00H</i>
	<i>MOV</i>	<i>P3M1, #00H</i>
	<i>MOV</i>	<i>P4M0, #00H</i>
	<i>MOV</i>	<i>P4M1, #00H</i>
	<i>MOV</i>	<i>P5M0, #00H</i>
	<i>MOV</i>	<i>P5M1, #00H</i>
	<i>MOV</i>	<i>S2CON,#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,#ES2</i> ;使能串口中断
	<i>SETB</i>	<i>EA</i>
	<i>MOV</i>	<i>S2BUF,#5AH</i> ;发送测试数据

JMP \$
END

11.5.15 UART3 中断

C 语言代码

//测试工作频率为 11.0592MHz

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

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

sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P4M1 = 0xb3;
sfr P4M0 = 0xb4;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;

sbit PI2 = P1^2;
sbit PI3 = P1^3;

void UART3_Isr() interrupt 17
{
    if (S3CON & 0x02)
    {
        S3CON &= ~0x02; //清中断标志
        PI2 = !PI2; //测试端口
    }
    if (S3CON & 0x01)
    {
        S3CON &= ~0x01; //清中断标志
        PI3 = !PI3; //测试端口
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
```

```

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

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

while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

T2L	DATA	0D7H
T2H	DATA	0D6H
AUXR	DATA	8EH
S3CON	DATA	0ACH
S3BUF	DATA	0ADH
IE2	DATA	0AFH
ES3	EQU	08H
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH
ORG	0000H	
LJMP	MAIN	
ORG	008BH	
LJMP	UART3ISR	
ORG	0100H	
UART3ISR:		
PUSH	ACC	
PUSH	PSW	
MOV	A,S3CON	
JNB	ACC.1,CHECKRI	
ANL	S3CON,#NOT 02H	; 清中断标志

CPL	P1.2	; 测试端口
CHECKRI:		
MOV	A,S3CON	
JNB	ACC.0,ISREXIT	
ANL	S3CON,#NOT 01H	; 清中断标志
CPL	P1.3	; 测试端口
ISREXIT:		
POP	PSW	
POP	ACC	
RETI		
MAIN:		
MOV	SP, #5FH	
MOV	P0M0, #00H	
MOV	P0M1, #00H	
MOV	P1M0, #00H	
MOV	P1M1, #00H	
MOV	P2M0, #00H	
MOV	P2M1, #00H	
MOV	P3M0, #00H	
MOV	P3M1, #00H	
MOV	P4M0, #00H	
MOV	P4M1, #00H	
MOV	P5M0, #00H	
MOV	P5M1, #00H	
MOV	S3CON,#10H	
MOV	T2L,#0E8H	; 65536-11059200/115200/4=0FFE8H
MOV	T2H,#0FFH	
MOV	AUXR,#14H	; 启动定时器
MOV	IE2,#ES3	; 使能串口中断
SETB	EA	
MOV	S3BUF,#5AH	; 发送测试数据
JMP	\$	
END		

11.5.16 UART4 中断

C 语言代码

// 测试工作频率为 11.0592MHz

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

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

sfr P0M1 = 0x93;
```

```

sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

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

void UART4_Isr() interrupt 18
{
    if(S4CON & 0x02)
    {
        S4CON &= ~0x02;                                //清中断标志
        P12 = !P12;                                   //测试端口
    }
    if(S4CON & 0x01)
    {
        S4CON &= ~0x01;                                //清中断标志
        P13 = !P13;                                   //测试端口
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

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

    while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

<i>T2L</i>	DATA	0D7H	
<i>T2H</i>	DATA	0D6H	
<i>AUXR</i>	DATA	8EH	
<i>S4CON</i>	DATA	84H	
<i>S4BUF</i>	DATA	85H	
<i>IE2</i>	DATA	0AFH	
<i>ES4</i>	EQU	10H	
<i>P0M1</i>	DATA	093H	
<i>P0M0</i>	DATA	094H	
<i>P1M1</i>	DATA	091H	
<i>P1M0</i>	DATA	092H	
<i>P2M1</i>	DATA	095H	
<i>P2M0</i>	DATA	096H	
<i>P3M1</i>	DATA	0B1H	
<i>P3M0</i>	DATA	0B2H	
<i>P4M1</i>	DATA	0B3H	
<i>P4M0</i>	DATA	0B4H	
<i>P5M1</i>	DATA	0C9H	
<i>P5M0</i>	DATA	0CAH	
	ORG	0000H	
	LJMP	MAIN	
	ORG	0093H	
	LJMP	UART4ISR	
	ORG	0100H	
<i>UART4ISR:</i>			
	PUSH	ACC	
	PUSH	PSW	
	MOV	A,S4CON	
	JNB	ACC.I,CHECKRI	
	ANL	S4CON,#NOT 02H	;清中断标志
	CPL	P1.2	;测试端口
<i>CHECKRI:</i>			
	MOV	A,S4CON	
	JNB	ACC.0,ISREXIT	
	ANL	S4CON,#NOT 01H	;清中断标志
	CPL	P1.3	;测试端口
<i>ISREXIT:</i>			
	POP	PSW	
	POP	ACC	
	RETI		
<i>MAIN:</i>			
	MOV	SP, #5FH	
	MOV	P0M0, #00H	
	MOV	P0M1, #00H	
	MOV	P1M0, #00H	
	MOV	P1M1, #00H	
	MOV	P2M0, #00H	
	MOV	P2M1, #00H	
	MOV	P3M0, #00H	
	MOV	P3M1, #00H	
	MOV	P4M0, #00H	
	MOV	P4M1, #00H	
	MOV	P5M0, #00H	
	MOV	P5M1, #00H	

```

MOV      S4CON,#10H
MOV      T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
MOV      T2H,#0FFH
MOV      AUXR,#14H          ;启动定时器
MOV      IE2,#ES4            ;使能串口中断
SETB    EA
MOV      S4BUF,#5AH          ;发送测试数据

JMP      $
END

```

11.5.17 ADC 中断

C 语言代码

//测试工作频率为 11.0592MHz

```

#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;

sfr    P0M1        = 0x93;
sfr    P0M0        = 0x94;
sfr    P1M1        = 0x91;
sfr    P1M0        = 0x92;
sfr    P2M1        = 0x95;
sfr    P2M0        = 0x96;
sfr    P3M1        = 0xb1;
sfr    P3M0        = 0xb2;
sfr    P4M1        = 0xb3;
sfr    P4M0        = 0xb4;
sfr    P5M1        = 0xc9;
sfr    P5M0        = 0xca;

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()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
}

```

```

P3M1 = 0x00;
P4M0 = 0x00;
P4M1 = 0x00;
P5M0 = 0x00;
P5M1 = 0x00;

ADCCFG = 0x00;                                //使能并启动ADC 模块
ADC_CONTR = 0xc0;                             //使能ADC 中断
EADC = I;
EA = I;

while (1);
}

```

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
EADC BIT IE.5

P0M1 DATA 093H
P0M0 DATA 094H
P1M1 DATA 091H
P1M0 DATA 092H
P2M1 DATA 095H
P2M0 DATA 096H
P3M1 DATA 0B1H
P3M0 DATA 0B2H
P4M1 DATA 0B3H
P4M0 DATA 0B4H
P5M1 DATA 0C9H
P5M0 DATA 0CAH

ORG 0000H
LJMP MAIN
ORG 002BH
LJMP ADCISR

ORG 0100H

ADCISR:
ANL ADC_CONTR,#NOT 20H ;清中断标志
MOV P0,ADC_RES          ;测试端口
MOV P2,ADC_RESL         ;测试端口
RETI

MAIN:
MOV SP, #5FH
MOV P0M0, #00H
MOV P0M1, #00H
MOV P1M0, #00H
MOV P1M1, #00H
MOV P2M0, #00H
MOV P2M1, #00H
MOV P3M0, #00H
MOV P3M1, #00H

```

```

MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

MOV      ADCCFG,#00H
MOV      ADC_CONTR,#0C0H      ;使能并启动ADC 模块
SETB    EADC                ;使能ADC 中断
SETB    EA

JMP     $

END

```

11.5.18 LVD 中断

C 语言代码

//测试工作频率为 11.0592MHz

```

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

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

sfr    P0M1        = 0x93;
sfr    P0M0        = 0x94;
sfr    P1M1        = 0x91;
sfr    P1M0        = 0x92;
sfr    P2M1        = 0x95;
sfr    P2M0        = 0x96;
sfr    P3M1        = 0xb1;
sfr    P3M0        = 0xb2;
sfr    P4M1        = 0xb3;
sfr    P4M0        = 0xb4;
sfr    P5M1        = 0xc9;
sfr    P5M0        = 0xca;
sbit   P10         = PI^0;

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

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
}

```

```

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

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);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

```

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

P0M1      DATA      093H
P0M0      DATA      094H
P1M1      DATA      091H
P1M0      DATA      092H
P2M1      DATA      095H
P2M0      DATA      096H
P3M1      DATA      0B1H
P3M0      DATA      0B2H
P4M1      DATA      0B3H
P4M0      DATA      0B4H
P5M1      DATA      0C9H
P5M0      DATA      0CAH

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, #5FH
MOV        P0M0, #00H

```

```

MOV      P0M1, #00H
MOV      P1M0, #00H
MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

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

11.5.19 PCA 中断

C 语言代码

```
//测试工作频率为 11.0592MHz
```

```

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

sfr    CCON      = 0xd8;
sbit   CF        = CCON^7;
sbit   CR        = CCON^6;
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    P0M1      = 0x93;
sfr    P0M0      = 0x94;
sfr    P1M1      = 0x91;
sfr    P1M0      = 0x92;
sfr    P2M1      = 0x95;
```

```

sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

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

    CCON = 0x00;
    CMOD = 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);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

CCON	DATA	0D8H
CF	BIT	CCON.7
CR	BIT	CCON.6
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

<i>CCAPMI</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>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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>JNB</i>	<i>CF,ISREXIT</i>
	<i>CLR</i>	<i>CF</i> ;清中断标志
	<i>CPL</i>	<i>P1.0</i> ;测试端口
<i>ISREXIT:</i>	<i>RETI</i>	
<i>MAIN:</i>	<i>MOV</i>	<i>SP, #5FH</i>
	<i>MOV</i>	<i>P0M0, #00H</i>
	<i>MOV</i>	<i>P0M1, #00H</i>
	<i>MOV</i>	<i>P1M0, #00H</i>
	<i>MOV</i>	<i>P1M1, #00H</i>
	<i>MOV</i>	<i>P2M0, #00H</i>
	<i>MOV</i>	<i>P2M1, #00H</i>
	<i>MOV</i>	<i>P3M0, #00H</i>
	<i>MOV</i>	<i>P3M1, #00H</i>
	<i>MOV</i>	<i>P4M0, #00H</i>
	<i>MOV</i>	<i>P4M1, #00H</i>
	<i>MOV</i>	<i>P5M0, #00H</i>
	<i>MOV</i>	<i>P5M1, #00H</i>
	<i>MOV</i>	<i>CCON,#00H</i>
	<i>MOV</i>	<i>CMOD,#09H</i> ;PCA 时钟为系统时钟,使能PCA 计时中断
	<i>SETB</i>	<i>CR</i> ;启动PCA 计时器
	<i>SETB</i>	<i>EA</i>
	<i>JMP</i>	\$
	<i>END</i>	

11.5.20 SPI 中断

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr SPSTAT      = 0xcd;
sfr SPCTL       = 0xce;
sfr SPDAT       = 0xcf;
sfr IE2          = 0xaf;
#define ESPI        0x02

sfr P0M1         = 0x93;
sfr P0M0         = 0x94;
sfr P1M1         = 0x91;
sfr P1M0         = 0x92;
sfr P2M1         = 0x95;
sfr P2M0         = 0x96;
sfr P3M1         = 0xb1;
sfr P3M0         = 0xb2;
sfr P4M1         = 0xb3;
sfr P4M0         = 0xb4;
sfr P5M1         = 0xc9;
sfr P5M0         = 0xca;

sbit P10          = PI^0;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;           //清中断标志
    P10 = !P10;             //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    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;                  //发送测试数据
    SPDAT = 0x5a;
```

```

    while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

<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>P0M1</i>	<i>DATA</i>	<i>093H</i>	
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>	
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>	
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>	
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>	
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>	
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>	
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>	
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>	
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>	
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>	
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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>CPL</i>	<i>P1.0</i>	; 测试端口
	<i>RETI</i>		
<i>MAIN:</i>	<i>MOV</i>	<i>SP, #5FH</i>	
	<i>MOV</i>	<i>P0M0, #00H</i>	
	<i>MOV</i>	<i>P0M1, #00H</i>	
	<i>MOV</i>	<i>P1M0, #00H</i>	
	<i>MOV</i>	<i>P1M1, #00H</i>	
	<i>MOV</i>	<i>P2M0, #00H</i>	
	<i>MOV</i>	<i>P2M1, #00H</i>	
	<i>MOV</i>	<i>P3M0, #00H</i>	
	<i>MOV</i>	<i>P3M1, #00H</i>	
	<i>MOV</i>	<i>P4M0, #00H</i>	
	<i>MOV</i>	<i>P4M1, #00H</i>	
	<i>MOV</i>	<i>P5M0, #00H</i>	
	<i>MOV</i>	<i>P5M1, #00H</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>MOV</i>	<i>SPDAT,#5AH</i>	; 发送测试数据
	<i>JMP</i>	\$	

END

11.5.21 比较器中断

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr CMPCRI      = 0xe6;
sfr CMPCR2      = 0xe7;

sfr P0M1        = 0x93;
sfr P0M0        = 0x94;
sfr P1M1        = 0x91;
sfr P1M0        = 0x92;
sfr P2M1        = 0x95;
sfr P2M0        = 0x96;
sfr P3M1        = 0xb1;
sfr P3M0        = 0xb2;
sfr P4M1        = 0xb3;
sfr P4M0        = 0xb4;
sfr P5M1        = 0xc9;
sfr P5M0        = 0xca;

sbit P10         = P1^0;

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

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CMPCR2 = 0x00;                                  //使能比较器模块
    CMPCRI = 0x80;                                  //使能比较器边沿中断
    CMPCRI |= 0x30;                                 //P3.6 为CMP+输入脚
    CMPCRI &= ~0x08;                               //P3.7 为CMP-输入脚
    CMPCRI |= 0x04;
```

```

CMPCRI /= 0x02;           //使能比较器输出
EA = 1;

while (1);
}

```

汇编代码

;测试工作频率为 11.0592MHz

CMPCRI	DATA	0E6H	
CMPCR2	DATA	0E7H	
P0M1	DATA	093H	
P0M0	DATA	094H	
P1M1	DATA	091H	
P1M0	DATA	092H	
P2M1	DATA	095H	
P2M0	DATA	096H	
P3M1	DATA	0B1H	
P3M0	DATA	0B2H	
P4M1	DATA	0B3H	
P4M0	DATA	0B4H	
P5M1	DATA	0C9H	
P5M0	DATA	0CAH	
ORG		0000H	
LJMP		MAIN	
ORG		00ABH	
LJMP		CMPISR	
ORG		0100H	
CMPISR:	ANL	CMPCRI,#NOT 40H	;清中断标志
	CPL	P1.0	;测试端口
	RETI		
MAIN:			
MOV		SP, #5FH	
MOV		P0M0, #00H	
MOV		P0M1, #00H	
MOV		P1M0, #00H	
MOV		P1M1, #00H	
MOV		P2M0, #00H	
MOV		P2M1, #00H	
MOV		P3M0, #00H	
MOV		P3M1, #00H	
MOV		P4M0, #00H	
MOV		P4M1, #00H	
MOV		P5M0, #00H	
MOV		P5M1, #00H	
MOV		CMPCR2,#00H	
MOV		CMPCRI,#80H	;使能比较器模块
ORL		CMPCRI,#30H	;使能比较器边沿中断
ANL		CMPCRI,#NOT 08H	;P3.6 为 CMP+ 输入脚
ORL		CMPCRI,#04H	;P3.7 为 CMP- 输入脚
ORL		CMPCRI,#02H	;使能比较器输出
SETB		EA	

JMP \$
END

11.5.22 PWM 中断

C 语言代码

//测试工作频率为 11.0592MHz

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

sfr P_SW2 = 0xba;
sfr PWMSET = 0xF1;
sfr PWMCFG01 = 0xF6;

#define PWM0CH (*(unsigned char volatile xdata *)0xFF00)
#define PWM0CL (*(unsigned char volatile xdata *)0xFF01)
#define PWM0CKS (*(unsigned char volatile xdata *)0xFF02)

sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P4M1 = 0xb3;
sfr P4M0 = 0xb4;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;

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

void PWM0_Isr() interrupt 22
{
    if (PWMCFG01 & 0x08)
    {
        PWMCFG01 &= ~0x08; //清中断标志
        P10 = !P10; //测试端口
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
```

```

P3M0 = 0x00;
P3M1 = 0x00;
P4M0 = 0x00;
P4M1 = 0x00;
P5M0 = 0x00;
P5M1 = 0x00;

PWMSET = 0x01;                                //使能 PWM0 模块 (必须先使能模块后面的设置才有效)

P_SW2 = 0x80;
PWM0CKS = 0x0f;                               //PWM0 时钟为系统时钟/16
PWM0CH = 0x01;                                //设置 PWM0 周期为 256 个 PWM0 时钟
PWM0CL = 0x00;
P_SW2 = 0x00;

PWMCFG01 = 0x05;                             //启动 PWM0 模块并使能 PWM0 计数器中断
EA = I;

while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

<i>P_SW2</i>	<i>DATA</i>	<i>0BAH</i>
<i>PWMSET</i>	<i>DATA</i>	<i>0F1H</i>
<i>PWMCFG01</i>	<i>DATA</i>	<i>0F6H</i>
<i>PWM0CH</i>	<i>EQU</i>	<i>0FF00H</i>
<i>PWM0CL</i>	<i>EQU</i>	<i>0FF01H</i>
<i>PWM0CKS</i>	<i>EQU</i>	<i>0FF02H</i>
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</i>
<i>ORG</i>		<i>0000H</i>
<i>LJMP</i>		<i>MAIN</i>
<i>ORG</i>		<i>00B3H</i>
<i>LJMP</i>		<i>PWM0ISR</i>
<i>ORG</i>		<i>0100H</i>
<i>PWM0ISR:</i>		
<i>PUSH</i>		<i>ACC</i>
<i>PUSH</i>		<i>PSW</i>
<i>MOV</i>		<i>A,PWMCFG01</i>
<i>JNB</i>		<i>ACC.3,JSREXIT</i>
<i>ANL</i>		<i>PWMCFG01,#NOT 08H</i>
		; 清中断标志

<i>CPL</i>	<i>P1.0</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, #5FH</i>	
<i>MOV</i>	<i>P0M0, #00H</i>	
<i>MOV</i>	<i>P0M1, #00H</i>	
<i>MOV</i>	<i>P1M0, #00H</i>	
<i>MOV</i>	<i>P1M1, #00H</i>	
<i>MOV</i>	<i>P2M0, #00H</i>	
<i>MOV</i>	<i>P2M1, #00H</i>	
<i>MOV</i>	<i>P3M0, #00H</i>	
<i>MOV</i>	<i>P3M1, #00H</i>	
<i>MOV</i>	<i>P4M0, #00H</i>	
<i>MOV</i>	<i>P4M1, #00H</i>	
<i>MOV</i>	<i>P5M0, #00H</i>	
<i>MOV</i>	<i>P5M1, #00H</i>	
<i>MOV</i>	<i>PWMSET,#01H</i>	; 使能 PWM0 模块 (必须先使能模块后面的设置才有效)
<i>MOV</i>	<i>P_SW2,#80H</i>	
<i>MOV</i>	<i>A,#0FH</i>	
<i>MOV</i>	<i>DPTR,#PWM0CKS</i>	
<i>MOVX</i>	<i>@DPTR,A</i>	; PWM0 时钟为系统时钟/16
<i>MOV</i>	<i>A,#01H</i>	
<i>MOV</i>	<i>DPTR,#PWM0CH</i>	; 设置 PWM0 周期为 256 个 PWM0 时钟
<i>MOVX</i>	<i>@DPTR,A</i>	
<i>MOV</i>	<i>A,#00H</i>	
<i>MOV</i>	<i>DPTR,#PWM0CL</i>	
<i>MOVX</i>	<i>@DPTR,A</i>	
<i>MOV</i>	<i>P_SW2,#00H</i>	
<i>MOV</i>	<i>PWMCFG01,#05H</i>	; 启动 PWM0 模块并使能 PWM0 计数器中断
<i>SETB</i>	<i>EA</i>	
<i>JMP</i>	\$	
END		

11.5.23 I2C 中断

C 语言代码

```
// 测试工作频率为 11.0592MHz
```

```
#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)

sfr P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;
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()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;                //使能I2C 主机模式
    I2CCFG = 0xc0;               //使能I2C 中断;
    I2CMSCR = 0x80;
    P_SW2 = 0x00;
    EA = 1;

    P_SW2 = 0x80;                //发送起始命令
    I2CMSCR = 0x81;
    P_SW2 = 0x00;

    while (1);
}
```

汇编代码

; 测试工作频率为 11.0592MHz

<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>P0M1</i>	<i>DATA</i>	<i>093H</i>
-------------	-------------	-------------

<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
-------------	-------------	-------------

<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
-------------	-------------	-------------

<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
-------------	-------------	-------------

<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
-------------	-------------	-------------

<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
-------------	-------------	-------------

<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
-------------	-------------	-------------

<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
-------------	-------------	-------------

<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
-------------	-------------	-------------

<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
-------------	-------------	-------------

<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
-------------	-------------	-------------

<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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>
------------	--------------

I2CISR:

<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>	
-------------	--

MAIN:

<i>MOV</i>	<i>SP, #5FH</i>
------------	-----------------

<i>MOV</i>	<i>P0M0, #00H</i>
------------	-------------------

<i>MOV</i>	<i>P0M1, #00H</i>
------------	-------------------

<i>MOV</i>	<i>P1M0, #00H</i>
------------	-------------------

<i>MOV</i>	<i>P1M1, #00H</i>
------------	-------------------

<i>MOV</i>	<i>P2M0, #00H</i>
------------	-------------------

```
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

MOV      P_SW2,#80H
MOV      A,#0C0H          ;使能I2C 主机模式
MOV      DPTR,#I2CCFG
MOVX    @DPTR,A
MOV      A,#80H           ;使能I2C 中断
MOV      DPTR,#I2CMSCR
MOVX    @DPTR,A
MOV      P_SW2,#00H
SETB    EA

MOV      P_SW2,#80H
MOV      A,#081H          ;发送起始命令
MOV      DPTR,#I2CMSCR
MOVX    @DPTR,A
MOV      P_SW2,#00H

JMP    $

END
```

12 定时器/计数器

产品线	定时器数量
STC8G1K08 系列	3
STC8G1K08-8Pin 系列	2
STC8G1K08A 系列	2
STC8G2K64S4 系列	5
STC8G2K64S2 系列	5
STC8G1K08T 系列	3
STC15H2K64S4 系列	5

STC8G 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，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
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	WKTEEN									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

符号	描述	地址	位地址与符号								复位值	
			B7	B6	B5	B4	B3	B2	B1	B0		
TM2PS	定时器 2 时钟预分频寄存器	FEA2H										0000,0000
TM3PS	定时器 3 时钟预分频寄存器	FEA3H										0000,0000
TM4PS	定时器 4 时钟预分频寄存器	FEA4H										0000,0000

12.2 定时器 0/1

12.2.1 定时器 0/1 控制寄存器 (TCON)

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

TF1: T1溢出中断标志。T1被允许计数以后，从初值开始加1计数。当产生溢出时由硬件将TF1位置“1”，并向CPU请求中断，一直保持到CPU响应中断时，才由硬件清“0”（也可由查询软件清“0”）。

TR1: 定时器T1的运行控制位。该位由软件置位和清零。当GATE (TMOD.7) =0, TR1=1时就允许T1开始计数，TR1=0时禁止T1计数。当GATE (TMOD.7) =1, TR1=1且INT1输入高电平时，才允许T1计数。

TF0: T0溢出中断标志。T0被允许计数以后，从初值开始加1计数，当产生溢出时，由硬件置“1”TF0，向CPU请求中断，一直保持CPU响应该中断时，才由硬件清0（也可由查询软件清0）。

TR0: 定时器T0的运行控制位。该位由软件置位和清零。当GATE (TMOD.3) =0, TR0=1时就允许T0开始计数，TR0=0时禁止T0计数。当GATE (TMOD.3) =1, TR0=1且INT0输入高电平时，才允许T0计数，TR0=0时禁止T0计数。

IE1: 外部中断1请求源 (INT1/P3.3) 标志。IE1=1，外部中断向CPU请求中断，当CPU响应该中断时由硬件清“0”IE1。

IT1: 外部中断源1触发控制位。IT1=0，上升沿或下降沿均可触发外部中断1。IT1=1，外部中断1程控为下降沿触发方式。

IE0: 外部中断0请求源 (INT0/P3.2) 标志。IE0=1外部中断0向CPU请求中断，当CPU响应外部中断时，由硬件清“0”IE0（边沿触发方式）。

IT0: 外部中断源0触发控制位。IT0=0，上升沿或下降沿均可触发外部中断0。IT0=1，外部中断0程控为下降沿触发方式。

12.2.2 定时器 0/1 模式寄存器 (TMOD)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TMOD	89H	T1_GATE	T1_C/T	T1_M1	T1_M0	T0_GATE	T0_C/T	T0_M1	T0_M0

T1_GATE: 控制定时器1，置1时只有在INT1脚为高及TR1控制位置1时才可打开定时器/计数器1。

T0_GATE: 控制定时器0，置1时只有在INT0脚为高及TR0控制位置1时才可打开定时器/计数器0。

T1_C/T: 控制定时器1用作定时器或计数器，清0则用作定时器（对内部系统时钟进行计数），置1用作计数器（对引脚T1/P3.5外部脉冲进行计数）。

T0_C/T: 控制定时器0用作定时器或计数器，清0则用作定时器（对内部系统时钟进行计数），置1用作计数器（对引脚T0/P3.4外部脉冲进行计数）。

T1_M1/T1_M0: 定时器定时器/计数器1模式选择

T1_M1	T1_M0	定时器/计数器1工作模式
0	0	16位自动重载模式 当[TH1,TL1]中的16位计数值溢出时，系统会自动将内部16位

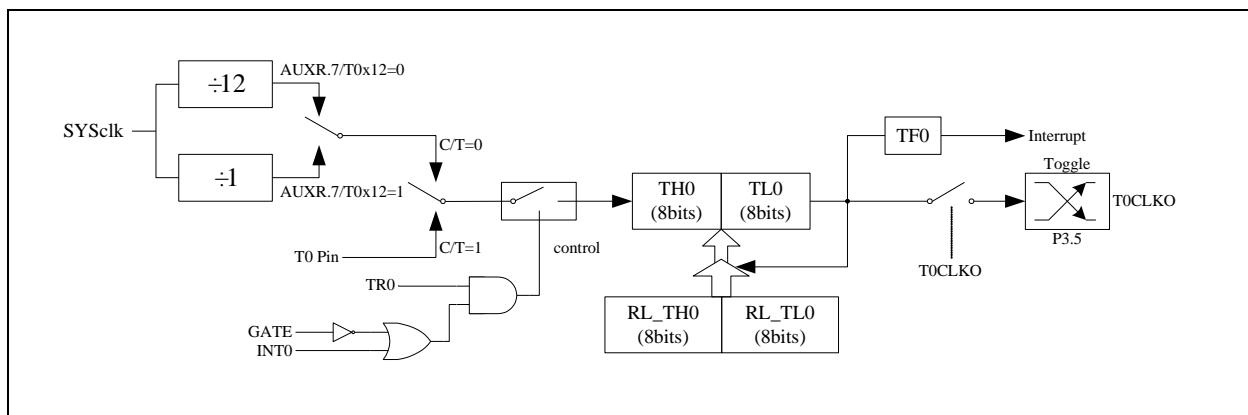
		重载寄存器中的重载值装入[TH1,TL1]中。
0	1	16位不自动重载模式 当[TH1,TL1]中的16位计数值溢出时，定时器1将从0开始计数
1	0	8位自动重载模式 当TL1中的8位计数值溢出时，系统会自动将TH1中的重载值装入TL1中。
1	1	T1停止工作

T0_M1/T0_M0: 定时器定时器/计数器0模式选择

		定时器/计数器0工作模式
0	0	16位自动重载模式 当[TH0,TL0]中的16位计数值溢出时，系统会自动将内部16位重载寄存器中的重载值装入[TH0,TL0]中。
0	1	16位不自动重载模式 当[TH0,TL0]中的16位计数值溢出时，定时器0将从0开始计数
1	0	8位自动重载模式 当TL0中的8位计数值溢出时，系统会自动将TH0中的重载值装入TL0中。
1	1	不可屏蔽中断的16位自动重载模式 与模式0相同，不可屏蔽中断，中断优先级最高，高于其他所有中断的优先级，并且不可关闭，可用作操作系统的系统节拍定时器，或者系统监控定时器。

12.2.3 定时器 0 模式 0 (16 位自动重装载模式)

此模式下定时器/计数器 0 作为可自动重装载的 16 位计数器，如下图所示：



定时器/计数器 0 的模式 0：16 位自动重装载模式

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

当 C/T=0 时，多路开关连接到系统时钟的分频输出，T0 对内部系统时钟计数，T0 工作在定时方式。当

C/T=1 时, 多路开关连接到外部脉冲输入 P3.4/T0, 即 T0 工作在计数方式。

STC 单片机的定时器 0 有两种计数速率: 一种是 12T 模式, 每 12 个时钟加 1, 与传统 8051 单片机相同; 另外一种是 1T 模式, 每个时钟加 1, 速度是传统 8051 单片机的 12 倍。T0 的速率由特殊功能寄存器 AUXR 中的 T0x12 决定, 如果 T0x12=0, T0 则工作在 12T 模式; 如果 T0x12=1, T0 则工作在 1T 模式。

定时器 0 有两个隐藏的寄存器 RL_TH0 和 RL_TL0。RL_TH0 与 TH0 共有同一个地址, RL_TL0 与 TL0 共有同一个地址。当 TR0=0 即定时器/计数器 0 被禁止工作时, 对 TL0 写入的内容会同时写入 RL_TL0, 对 TH0 写入的内容也会同时写入 RL_TH0。当 TR0=1 即定时器/计数器 0 被允许工作时, 对 TL0 写入内容, 实际上不是写入当前寄存器 TL0 中, 而是写入隐藏的寄存器 RL_TL0 中, 对 TH0 写入内容, 实际上也不是写入当前寄存器 TH0 中, 而是写入隐藏的寄存器 RL_TH0, 这样可以巧妙地实现 16 位重装载定时器。当读 TH0 和 TL0 的内容时, 所读的内容就是 TH0 和 TL0 的内容, 而不是 RL_TH0 和 RL_TL0 的内容。

当定时器 0 工作在模式 0 (TMOD[1:0]/[M1,M0]=00B) 时, [TH0,TL0] 的溢出不仅置位 TF0, 而且会自动将[RL_TH0,RL_TL0]的内容重新装入[TH0,TL0]。

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

如果 C/T=0, 定时器/计数器 T0 对内部系统时钟计数, 则:

T0 工作在 1T 模式 (AUXR.7/T0x12=1) 时的输出时钟频率 = (SYScclk)/(65536-[RL_TH0, RL_TL0])/2

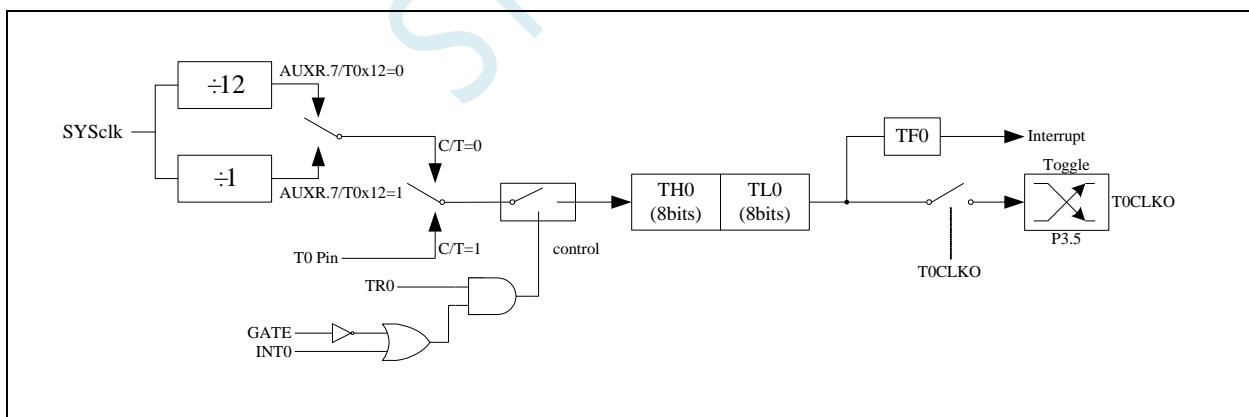
T0 工作在 12T 模式 (AUXR.7/T0x12=0) 时的输出时钟频率 = (SYScclk)/12/(65536-[RL_TH0, RL_TL0])/2

如果 C/T=1, 定时器/计数器 T0 是对外部脉冲输入(P3.4/T0)计数, 则:

输出时钟频率 = (T0_Pin_CLK) / (65536-[RL_TH0, RL_TL0])/2

12.2.4 定时器 0 模式 1 (16 位不可重装载模式)

此模式下定时器/计数器 0 工作在 16 位不可重装载模式, 如下图所示



定时器/计数器 0 的模式 1: 16 位不可重装载模式

此模式下, 定时器/计数器 0 配置为 16 位不可重装载模式, 由 TL0 的 8 位和 TH0 的 8 位所构成。TL0 的 8 位溢出向 TH0 进位, TH0 计数溢出置位 TCON 中的溢出标志位 TF0。

当 GATE=0(TM0D.3)时, 如 TR0=1, 则定时器计数。GATE=1 时, 允许由外部输入 INT0 控制定时器 0, 这样可实现脉宽测量。TR0 为 TCON 寄存器内的控制位, TCON 寄存器各位的具体功能描述见上节 TCON 寄存器的介绍。

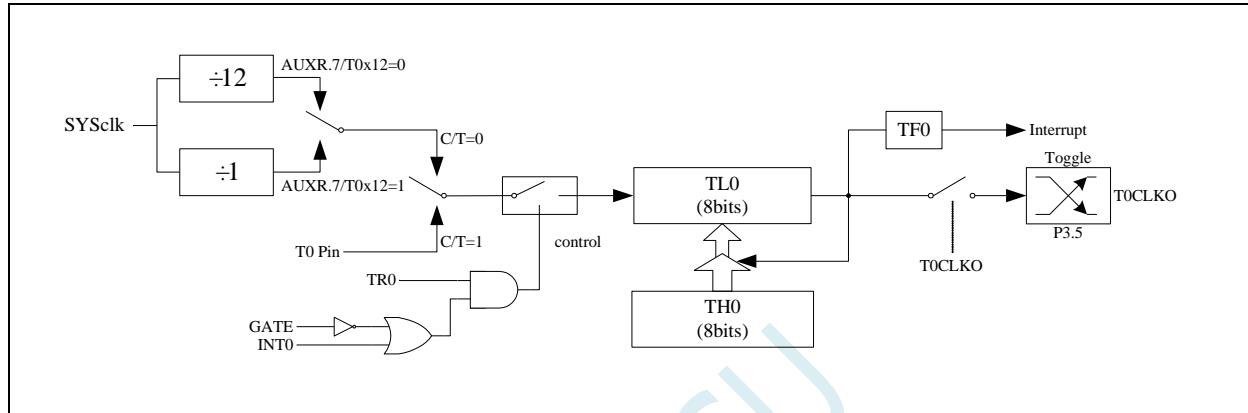
当 C/T=0 时, 多路开关连接到系统时钟的分频输出, T0 对内部系统时钟计数, T0 工作在定时方式。当

C/T=1 时, 多路开关连接到外部脉冲输入 P3.4/T0, 即 T0 工作在计数方式。

STC 单片机的定时器 0 有两种计数速率: 一种是 12T 模式, 每 12 个时钟加 1, 与传统 8051 单片机相同; 另外一种是 1T 模式, 每个时钟加 1, 速度是传统 8051 单片机的 12 倍。T0 的速率由特殊功能寄存器 AUXR 中的 T0x12 决定, 如果 T0x12=0, T0 则工作在 12T 模式; 如果 T0x12=1, T0 则工作在 1T 模式。

12.2.5 定时器 0 模式 2 (8 位自动重装载模式)

此模式下定时器/计数器 0 作为可自动重装载的 8 位计数器, 如下图所示:



定时器/计数器 0 的模式 2: 8 位自动重装载模式

TL0 的溢出不仅置位 TF0, 而且将 TH0 的内容重新装入 TL0, TH0 内容由软件预置, 重装时 TH0 内容不变。

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

如果 C/T=0, 定时器/计数器 T0 对内部系统时钟计数, 则:

$$\text{T0 工作在 1T 模式 (AUXR.7/T0x12=1) 时的输出时钟频率} = (\text{SYSelk})/(256-\text{TH0})/2$$

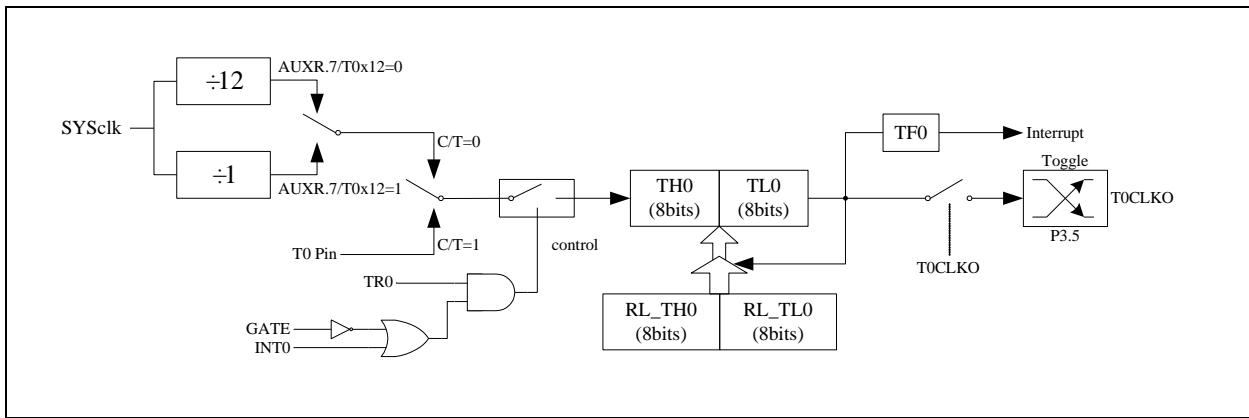
$$\text{T0 工作在 12T 模式 (AUXR.7/T0x12=0) 时的输出时钟频率} = (\text{SYSelk})/12/(256-\text{TH0})/2$$

如果 C/T=1, 定时器/计数器 T0 是对外部脉冲输入(P3.4/T0)计数, 则:

$$\text{输出时钟频率} = (\text{T0_Pin_CLK}) / (256-\text{TH0})/2$$

12.2.6 定时器 0 模式 3 (不可屏蔽中断 16 位自动重装载, 实时操作系统节拍器)

对定时器/计数器 0, 其工作模式模式 3 与工作模式 0 是一样的 (下图定时器模式 3 的原理图, 与工作模式 0 是一样的)。唯一不同的是: 当定时器/计数器 0 工作在模式 3 时, 只需允许 ET0/IE.1(定时器/计数器 0 中断允许位), 不需要允许 EA/IE.7(总中断使能位)就能打开定时器/计数器 0 的中断, 此模式下的定时器/计数器 0 中断与总中断使能位 EA 无关, 一旦工作在模式 3 下的定时器/计数器 0 中断被打开(ET0=1), 那么该中断是不可屏蔽的, 该中断的优先级是最高的, 即该中断不能被任何中断所打断, 而且该中断打开后既不受 EA/IE.7 控制也不再受 ET0 控制, 当 EA=0 或 ET0=0 时都不能屏蔽此中断。故将此模式称为不可屏蔽中断的 16 位自动重装载模式。

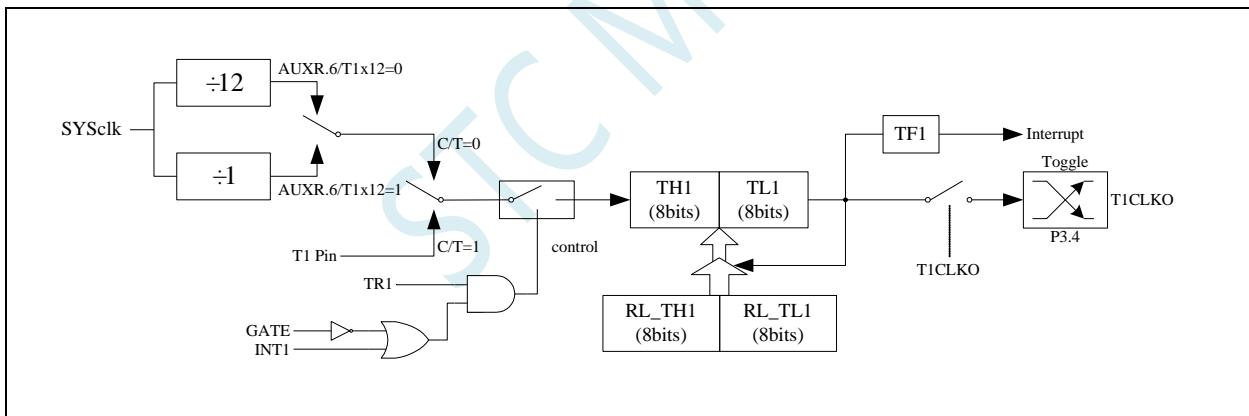


定时器/计数器 0 的模式 3: 不可屏蔽中断的 16 位自动重装载模式

注意: 当定时器/计数器 0 工作在模式 3(不可屏蔽中断的 16 位自动重装载模式)时, 不需要允许 EA/IE.7(总中断使能位), 只需允许 ET0/IE.1(定时器/计数器 0 中断允许位)就能打开定时器/计数器 0 的中断, 此模式下的定时器/计数器 0 中断与总中断使能位 EA 无关。一旦此模式下的定时器/计数器 0 中断被打开后, 该定时器/计数器 0 中断优先级就是最高的, 它不能被其它任何中断所打断(不管是比定时器/计数器 0 中断优先级低的中断还是比其优先级高的中断, 都不能打断此时的定时器/计数器 0 中断), 而且该中断打开后既不受 EA/IE.7 控制也不再受 ET0 控制了, 清零 EA 或 ET0 都不能关闭此中断。

12.2.7 定时器 1 模式 0 (16 位自动重装载模式)

此模式下定时器/计数器 1 作为可自动重装载的 16 位计数器, 如下图所示:



定时器/计数器 1 的模式 0: 16 位自动重装载模式

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

当 C/T=0 时, 多路开关连接到系统时钟的分频输出, T1 对内部系统时钟计数, T1 工作在定时方式。当 C/T=1 时, 多路开关连接到外部脉冲输入 P3.4/T1, 即 T1 工作在计数方式。

STC 单片机的定时器 1 有两种计数速率: 一种是 12T 模式, 每 12 个时钟加 1, 与传统 8051 单片机相同; 另外一种是 1T 模式, 每个时钟加 1, 速度是传统 8051 单片机的 12 倍。T1 的速率由特殊功能寄存器 AUXR 中的 T1x12 决定, 如果 T1x12=0, T1 则工作在 12T 模式; 如果 T1x12=1, T1 则工作在 1T 模式。

定时器 1 有两个隐藏的寄存器 RL_TH1 和 RL_TL1。RL_TH1 与 TH1 共有同一个地址, RL_TL1 与 TL1 共有同一个地址。当 TR1=0 即定时器/计数器 1 被禁止工作时, 对 TL1 写入的内容会同时写入 RL_TL1, 对

TH1 写入的内容也会同时写入 RL_TH1。当 TR1=1 即定时器/计数器 1 被允许工作时, 对 TL1 写入内容, 实际上不是写入当前寄存器 TL1 中, 而是写入隐藏的寄存器 RL_TL1 中, 对 TH1 写入内容, 实际上也不是写入当前寄存器 TH1 中, 而是写入隐藏的寄存器 RL_TH1, 这样可以巧妙地实现 16 位重装载定时器。当读 TH1 和 TL1 的内容时, 所读的内容就是 TH1 和 TL1 的内容, 而不是 RL_TH1 和 RL_TL1 的内容。

当定时器 1 工作在模式 1 (TMOD[5:4]/[M1,M0]=00B) 时, [TH1,TL1]的溢出不仅置位 TF1, 而且会自动将[RL_TH1,RL_TL1]的内容重新装入[TH1,TL1]。

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

如果 C/T=0, 定时器/计数器 T1 对内部系统时钟计数, 则:

T1 工作在 1T 模式 (AUXR.6/T1x12=1) 时的输出时钟频率 = (SYSclk)/(65536-[RL_TH1, RL_TL1])/2

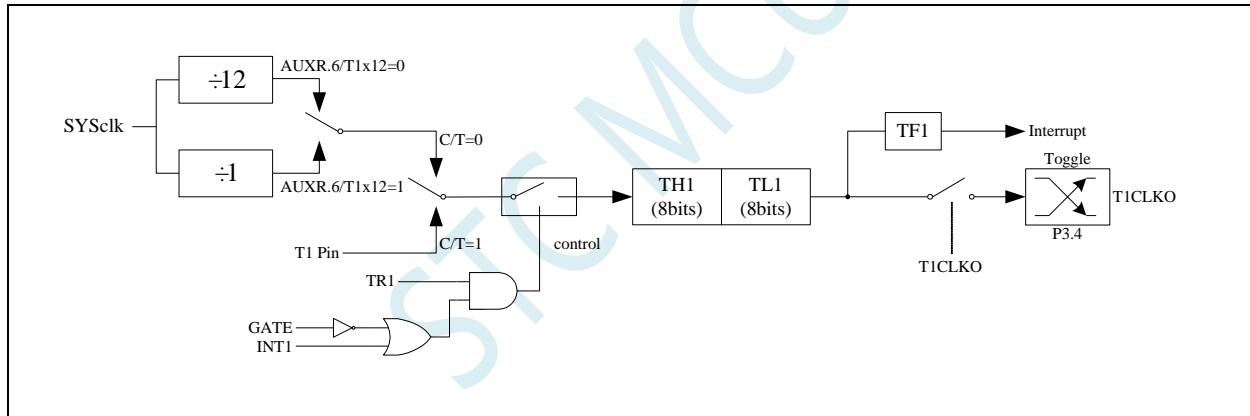
T1 工作在 12T 模式 (AUXR.6/T1x12=0) 时的输出时钟频率 = (SYSclk)/12/(65536-[RL_TH1, RL_TL1])/2

如果 C/T=1, 定时器/计数器 T1 是对外部脉冲输入(P3.5/T1)计数, 则:

输出时钟频率 = (T1_Pin_CLK) / (65536-[RL_TH1, RL_TL1])/2

12.2.8 定时器 1 模式 1 (16 位不可重装载模式)

此模式下定时器/计数器 1 工作在 16 位不可重装载模式, 如下图所示



定时器/计数器 1 的模式 1: 16 位不可重装载模式

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

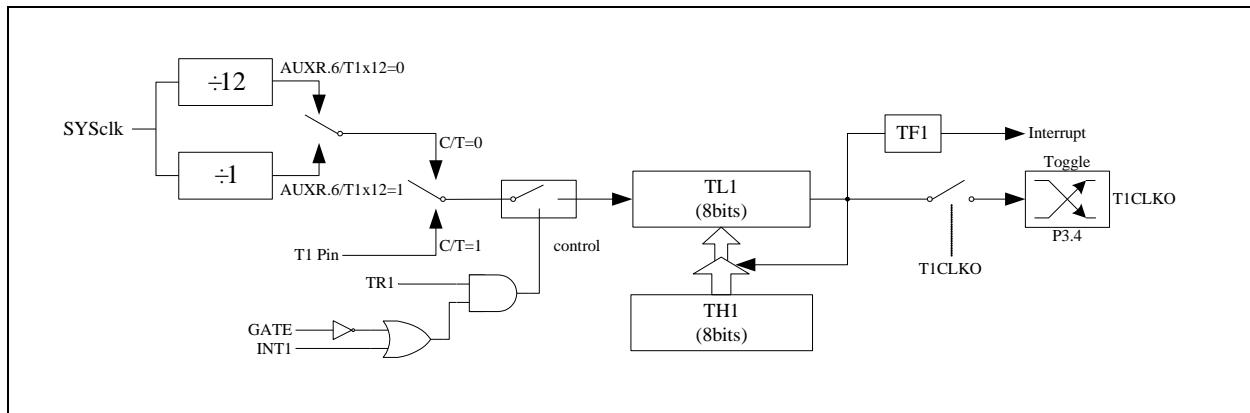
当 GATE=0(TM0D.7)时, 如 TR1=1, 则定时器计数。GATE=1 时, 允许由外部输入 INT1 控制定时器 1, 这样可实现脉宽测量。TR1 为 TCON 寄存器内的控制位, TCON 寄存器各位的具体功能描述见上节 TCON 寄存器的介绍。

当 C/T=0 时, 多路开关连接到系统时钟的分频输出, T1 对内部系统时钟计数, T1 工作在定时方式。当 C/T=1 时, 多路开关连接到外部脉冲输入 P3.5/T1, 即 T1 工作在计数方式。

STC 单片机的定时器 1 有两种计数速率: 一种是 12T 模式, 每 12 个时钟加 1, 与传统 8051 单片机相同; 另外一种是 1T 模式, 每个时钟加 1, 速度是传统 8051 单片机的 12 倍。T1 的速率由特殊功能寄存器 AUXR 中的 T1x12 决定, 如果 T1x12=0, T1 则工作在 12T 模式; 如果 T1x12=1, T1 则工作在 1T 模式

12.2.9 定时器 1 模式 2 (8 位自动重装载模式)

此模式下定时器/计数器 1 作为可自动重装载的 8 位计数器, 如下图所示:



定时器/计数器 1 的模式 2: 8 位自动重装载模式

TL1 的溢出不仅置位 TF1，而且将 TH1 的内容重新装入 TL1，TH1 内容由软件预置，重装时 TH1 内容不变。

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

如果 C/T=0，定时器/计数器 T1 对内部系统时钟计数，则：

$$T1 \text{ 工作在 } 1T \text{ 模式 (AUXR.6/T1x12=1) 时的输出时钟频率} = (\text{SYScclk})/(256-\text{TH1})/2$$

$$T1 \text{ 工作在 } 12T \text{ 模式 (AUXR.6/T1x12=0) 时的输出时钟频率} = (\text{SYScclk})/12/(256-\text{TH1})/2$$

如果 C/T=1，定时器/计数器 T1 是对外部脉冲输入(P3.5/T1)计数，则：

$$\text{输出时钟频率} = (T1_Pin_CLK) / (256-\text{TH1})/2$$

12.2.10 定时器 0 计数寄存器 (TL0, TH0)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TL0	8AH								
TH0	8CH								

当定时器/计数器0工作在16位模式（模式0、模式1、模式3）时，TL0和TH0组合成为一个16位寄存器，TL0为低字节，TH0为高字节。若为8位模式（模式2）时，TL0和TH0为两个独立的8位寄存器。

12.2.11 定时器 1 计数寄存器 (TL1, TH1)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TL1	8BH								
TH1	8DH								

当定时器/计数器1工作在16位模式（模式0、模式1）时，TL1和TH1组合成为一个16位寄存器，TL1为低字节，TH1为高字节。若为8位模式（模式2）时，TL1和TH1为两个独立的8位寄存器。

12.2.12 辅助寄存器 1 (AUXR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH								

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)

12.2.13 中断与时钟输出控制寄存器 (INTCLKO)

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

T0CLKO: 定时器0时钟输出控制

- 0: 关闭时钟输出
- 1: 使能 P3.5 口的是定时器 0 时钟输出功能
当定时器 0 计数发生溢出时, P3.5 口的电平自动发生翻转。

T1CLKO: 定时器1时钟输出控制

- 0: 关闭时钟输出
- 1: 使能 P3.4 口的是定时器 1 时钟输出功能
当定时器 1 计数发生溢出时, P3.4 口的电平自动发生翻转。

12.2.14 定时器 0 定时计算公式

定时器模式	定时器速度	周期计算公式
模式0/3 (16位自动重载)	1T	定时器周期 = $\frac{65536 - [TH0, TL0]}{SYSclk}$ (自动重载)
	12T	定时器周期 = $\frac{65536 - [TH0, TL0]}{SYSclk} \times 12$ (自动重载)
模式1 (16位不自动重载)	1T	定时器周期 = $\frac{65536 - [TH0, TL0]}{SYSclk}$ (需软件装载)
	12T	定时器周期 = $\frac{65536 - [TH0, TL0]}{SYSclk} \times 12$ (需软件装载)
模式2 (8位自动重载)	1T	定时器周期 = $\frac{256 - TH0}{SYSclk}$ (自动重载)
	12T	定时器周期 = $\frac{256 - TH0}{SYSclk} \times 12$ (自动重载)

12.2.15 定时器1定时计算公式

定时器模式	定时器速度	周期计算公式
模式0 (16位自动重载)	1T	定时器周期 = $\frac{65536 - [TH1, TL1]}{SYSclk}$ (自动重载)
	12T	定时器周期 = $\frac{65536 - [TH1, TL1]}{SYSclk} \times 12$ (自动重载)
模式1 (16位不自动重载)	1T	定时器周期 = $\frac{65536 - [TH1, TL1]}{SYSclk}$ (需软件装载)
	12T	定时器周期 = $\frac{65536 - [TH1, TL1]}{SYSclk} \times 12$ (需软件装载)
模式2 (8位自动重载)	1T	定时器周期 = $\frac{256 - TH1}{SYSclk}$ (自动重载)
	12T	定时器周期 = $\frac{256 - TH1}{SYSclk} \times 12$ (自动重载)

12.3 定时器 2 (24 位定时器, 8 位预分频+16 位定时)

12.3.1 辅助寄存器 1 (AUXR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

T2R: 定时器2的运行控制位

0: 定时器 2 停止计数

1: 定时器 2 开始计数

T2_C/T: 控制定时器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)

12.3.2 中断与时钟输出控制寄存器 (INTCLKO)

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

T2CLKO: 定时器2时钟输出控制

0: 关闭时钟输出

1: 使能 P1.3 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。

12.3.3 定时器 2 计数寄存器 (T2L, T2H)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T2L	D7H								
T2H	D6H								

定时器/计数器2的工作模式固定为16位重载模式, T2L和T2H组合成为一个16位寄存器, T2L为低字节,

T2H为高字节。当[T2H,T2L]中的16位计数值溢出时, 系统会自动将内部16位重载寄存器中的重载值装入[T2H,T2L]中。

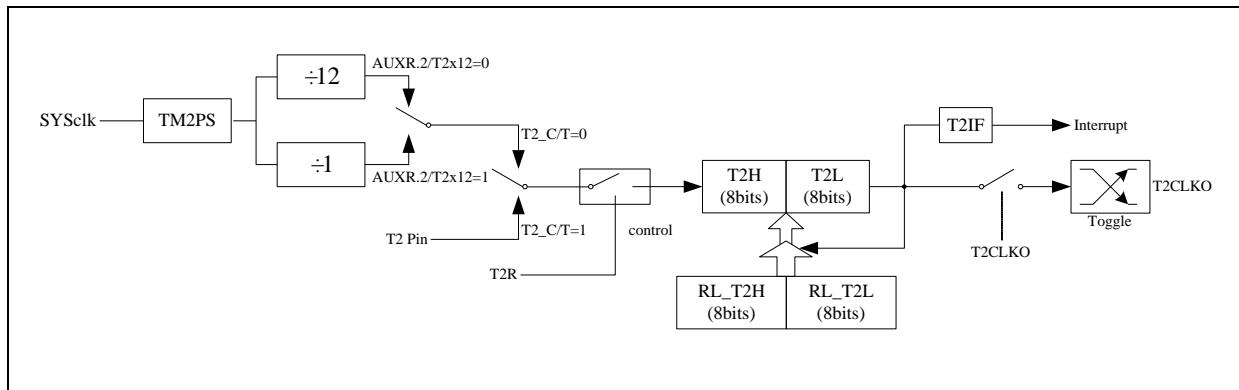
12.3.4 定时器 2 的 8 位预分频寄存器 (TM2PS)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TM2PS	FEA2H								

定时器2的时钟 = 系统时钟SYSclk \div (TM2PS + 1)

12.3.5 定时器 2 工作模式

定时器/计数器 2 的原理框图如下:



定时器/计数器 2 的工作模式: 16 位自动重装载模式

T2R/AUXR.4 为 AUXR 寄存器内的控制位, AUXR 寄存器各位的具体功能描述见上节 AUXR 寄存器的介绍。

当 T2_C/T=0 时, 多路开关连接到系统时钟输出, T2 对内部系统时钟计数, T2 工作在定时方式。当 T2_C/T=1 时, 多路开关连接到外部脉冲输 T2, 即 T2 工作在计数方式。

STC 单片机的定时器 2 有两种计数速率: 一种是 12T 模式, 每 12 个时钟加 1, 与传统 8051 单片机相同; 另外一种是 1T 模式, 每个时钟加 1, 速度是传统 8051 单片机的 12 倍。T2 的速率由特殊功能寄存器 AUXR 中的 T2x12 决定, 如果 T2x12=0, T2 则工作在 12T 模式; 如果 T2x12=1, T2 则工作在 1T 模式

定时器 2 有两个隐藏的寄存器 RL_T2H 和 RL_T2L。RL_T2H 与 T2H 共有同一个地址, RL_T2L 与 T2L 共有同一个地址。当 T2R=0 即定时器/计数器 2 被禁止工作时, 对 T2L 写入的内容会同时写入 RL_T2L, 对 T2H 写入的内容也会同时写入 RL_T2H。当 T2R=1 即定时器/计数器 2 被允许工作时, 对 T2L 写入内容, 实际上不是写入当前寄存器 T2L 中, 而是写入隐藏的寄存器 RL_T2L 中, 对 T2H 写入内容, 实际上也不是写入当前寄存器 T2H 中, 而是写入隐藏的寄存器 RL_T2H, 这样可以巧妙地实现 16 位重装载定时器。当读 T2H 和 T2L 的内容时, 所读的内容就是 T2H 和 T2L 的内容, 而不是 RL_T2H 和 RL_T2L 的内容。

[T2H,T2L]的溢出不仅置位中断请求标志位 (T2IF), 使 CPU 转去执行定时器 2 的中断程序, 而且会自动将[RL_T2H,RL_T2L]的内容重新装入[T2H,T2L]。

12.3.6 定时器 2 计算公式

定时器速度	周期计算公式
1T	定时器周期 = $\frac{65536 - [T2H, T2L]}{SYSclk/(TM2PS+1)}$ (自动重载)
12T	定时器周期 = $\frac{65536 - [T2H, T2L]}{SYSclk/(TM2PS+1)} \times 12$ (自动重载)

12.4 定时器 3/4 (24 位定时器, 8 位预分频+16 位定时)

12.4.1 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

T4R: 定时器4的运行控制位

0: 定时器 4 停止计数

1: 定时器 4 开始计数

T4_C/T: 控制定时器4用作定时器或计数器, 清0则用作定时器(对内部系统时钟进行计数), 置1用作计数器(对引脚T4/P0.6外部脉冲进行计数)。

T4x12: 定时器4速度控制位

0: 12T 模式, 即 CPU 时钟 12 分频 (FOSC/12)

1: 1T 模式, 即 CPU 时钟不分频分频 (FOSC/1)

T4CLKO: 定时器4时钟输出控制

0: 关闭时钟输出

1: 使能 P0.7 口的是定时器 4 时钟输出功能

当定时器 4 计数发生溢出时, P0.7 口的电平自动发生翻转。

T3R: 定时器3的运行控制位

0: 定时器 3 停止计数

1: 定时器 3 开始计数

T3_C/T: 控制定时器3用作定时器或计数器, 清0则用作定时器(对内部系统时钟进行计数), 置1用作计数器(对引脚T3/P0.4外部脉冲进行计数)。

T3x12: 定时器3速度控制位

0: 12T 模式, 即 CPU 时钟 12 分频 (FOSC/12)

1: 1T 模式, 即 CPU 时钟不分频分频 (FOSC/1)

T3CLKO: 定时器3时钟输出控制

0: 关闭时钟输出

1: 使能 P0.5 口的是定时器 3 时钟输出功能

当定时器 3 计数发生溢出时, P0.5 口的电平自动发生翻转。

12.4.2 定时器 3 计数寄存器 (T3L, T3H)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T3L	D5H								
T3H	D4H								

定时器/计数器3的工作模式固定为16位重载模式, T3L和T3H组合成为一个16位寄存器, T3L为低字节,

T3H为高字节。当[T3H,T3L]中的16位计数值溢出时, 系统会自动将内部16位重载寄存器中的重载值装入[T3H,T3L]中。

12.4.3 定时器 4 计数寄存器 (T4L, T4H)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T4L	D3H								

T4H	D2H	
-----	-----	--

定时器/计数器 4 的工作模式固定为 16 位重载模式, T4L 和 T4H 组合成为一个 16 位寄存器, T4L 为低字节, T4H 为高字节。当[T4H,T4L]中的 16 位计数值溢出时, 系统会自动将内部 16 位重载寄存器中的重载值装入[T4H,T4L]中。

12.4.4 定时器 3 的 8 位预分频寄存器 (TM3PS)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TM3PS	FEA3H								

定时器3的时钟 = 系统时钟SYSclk \div (TM3PS + 1)

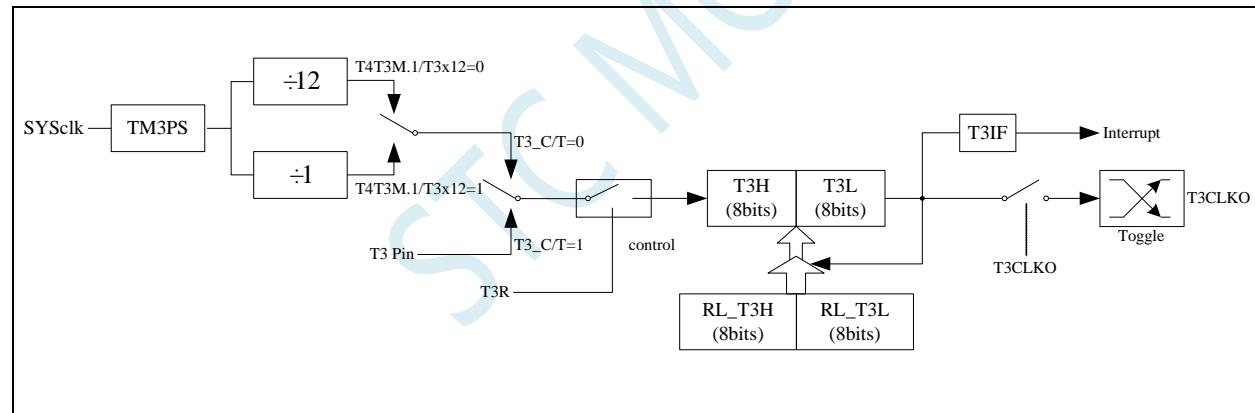
12.4.5 定时器 4 的 8 位预分频寄存器 (TM4PS)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TM4PS	FEA4H								

定时器4的时钟 = 系统时钟SYSclk \div (TM4PS + 1)

12.4.6 定时器 3 工作模式

定时器/计数器 3 的原理框图如下:



定时器/计数器 3 的工作模式: 16 位自动重装载模式

T3R/T4T3M.3 为 T4T3M 寄存器内的控制位, T4T3M 寄存器各位的具体功能描述见上节 T4T3M 寄存器的介绍。

当 T3_C/T=0 时, 多路开关连接到系统时钟输出, T3 对内部系统时钟计数, T3 工作在定时方式。当 T3_C/T=1 时, 多路开关连接到外部脉冲输入 T3, 即 T3 工作在计数方式。

STC 单片机的定时器 3 有两种计数速率: 一种是 12T 模式, 每 12 个时钟加 1, 与传统 8051 单片机相同; 另外一种是 1T 模式, 每个时钟加 1, 速度是传统 8051 单片机的 12 倍。T3 的速率由特殊功能寄存器 T4T3M 中的 T3x12 决定, 如果 T3x12=0, T3 则工作在 12T 模式; 如果 T3x12=1, T3 则工作在 1T 模式

定时器 3 有两个隐藏的寄存器 RL_T3H 和 RL_T3L。RL_T3H 与 T3H 共有同一个地址, RL_T3L 与 T3L 共有同一个地址。当 T3R=0 即定时器/计数器 3 被禁止工作时, 对 T3L 写入的内容会同时写入 RL_T3L, 对 T3H 写入的内容也会同时写入 RL_T3H。当 T3R=1 即定时器/计数器 3 被允许工作时, 对 T3L 写入内容, 实际上不是写入当前寄存器 T3L 中, 而是写入隐藏的寄存器 RL_T3L 中, 对 T3H 写入内容, 实际上也不是写入

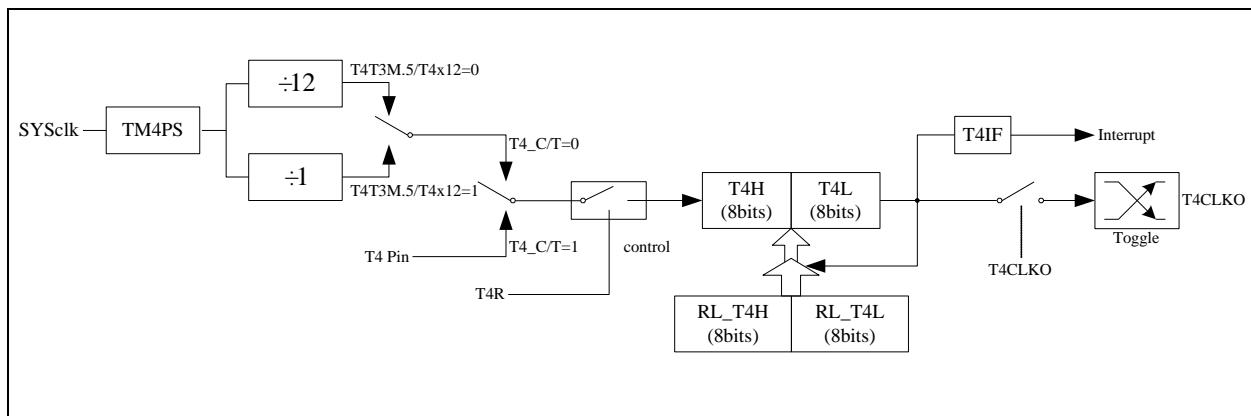
当前寄存器 T3H 中，而是写入隐藏的寄存器 RL_T3H，这样可以巧妙地实现 16 位重装载定时器。当读 T3H 和 T3L 的内容时，所读的内容就是 T3H 和 T3L 的内容，而不是 RL_T3H 和 RL_T3L 的内容。

[T3H,T3L]的溢出不仅置位中断请求标志位 (T3IF)，使 CPU 转去执行定时器 3 的中断程序，而且会自动将[RL_T3H,RL_T3L]的内容重新装入[T3H,T3L]。

STCMCU

12.4.7 定时器 4 工作模式

定时器/计数器 4 的原理框图如下:



定时器/计数器 4 的工作模式: 16 位自动重装载模式

T4R/T4T3M.7 为 T4T3M 寄存器内的控制位, T4T3M 寄存器各位的具体功能描述见上节 T4T3M 寄存器的介绍。

当 T4_C/T=0 时, 多路开关连接到系统时钟输出, T4 对内部系统时钟计数, T4 工作在定时方式。当 T4_C/T=1 时, 多路开关连接到外部脉冲输 T4, 即 T4 工作在计数方式。

STC 单片机的定时器 4 有两种计数速率: 一种是 12T 模式, 每 12 个时钟加 1, 与传统 8051 单片机相同; 另外一种是 1T 模式, 每个时钟加 1, 速度是传统 8051 单片机的 12 倍。T4 的速率由特殊功能寄存器 T4T3M 中的 T4x12 决定, 如果 T4x12=0, T4 则工作在 12T 模式; 如果 T4x12=1, T4 则工作在 1T 模式

定时器 4 有两个隐藏的寄存器 RL_T4H 和 RL_T4L。RL_T4H 与 T4H 共有同一个地址, RL_T4L 与 T4L 共有同一个地址。当 T4R=0 即定时器/计数器 4 被禁止工作时, 对 T4L 写入的内容会同时写入 RL_T4L, 对 T4H 写入的内容也会同时写入 RL_T4H。当 T4R=1 即定时器/计数器 4 被允许工作时, 对 T4L 写入内容, 实际上不是写入当前寄存器 T4L 中, 而是写入隐藏的寄存器 RL_T4L 中, 对 T4H 写入内容, 实际上也不是写入当前寄存器 T4H 中, 而是写入隐藏的寄存器 RL_T4H, 这样可以巧妙地实现 16 位重装载定时器。当读 T4H 和 T4L 的内容时, 所读的内容就是 T4H 和 T4L 的内容, 而不是 RL_T4H 和 RL_T4L 的内容。

[T4H,T4L]的溢出不仅置位中断请求标志位 (T4IF), 使 CPU 转去执行定时器 4 的中断程序, 而且会自动将[RL_T4H,RL_T4L]的内容重新装入[T4H,T4L]。

12.4.8 定时器 3 计算公式

定时器速度	周期计算公式
1T	定时器周期 = $\frac{65536 - [T3H, T3L]}{SYSclk/(TM3PS+1)}$ (自动重载)
12T	定时器周期 = $\frac{65536 - [T3H, T3L]}{SYSclk/(TM3PS+1)} \times 12$ (自动重载)

12.4.9 定时器 4 计算公式

定时器速度	周期计算公式
1T	定时器周期 = $\frac{65536 - [T4H, T4L]}{SYSclk/(TM4PS+1)}$ (自动重载)
12T	定时器周期 = $\frac{65536 - [T4H, T4L]}{SYSclk/(TM4PS+1)} \times 12$ (自动重载)

STCMCU

12.5 范例程序

12.5.1 定时器 0 (模式 0—16 位自动重载), 用作定时

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

sbit P10      = P1^0;

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

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    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);
}
```

汇编代码*;测试工作频率为 11.0592MHz*

```

P0M1      DATA      093H
P0M0      DATA      094H
P1M1      DATA      091H
P1M0      DATA      092H
P2M1      DATA      095H
P2M0      DATA      096H
P3M1      DATA      0B1H
P3M0      DATA      0B2H
P4M1      DATA      0B3H
P4M0      DATA      0B4H
P5M1      DATA      0C9H
P5M0      DATA      0CAH

ORG        0000H
LJMP       MAIN
ORG        000BH
LJMP       TM0ISR

ORG        0100H
TM0ISR:
CPL        P1.0
RETI

MAIN:
MOV        SP, #5FH
MOV        P0M0, #00H
MOV        P0M1, #00H
MOV        P1M0, #00H
MOV        P1M1, #00H
MOV        P2M0, #00H
MOV        P2M1, #00H
MOV        P3M0, #00H
MOV        P3M1, #00H
MOV        P4M0, #00H
MOV        P4M1, #00H
MOV        P5M0, #00H
MOV        P5M1, #00H

MOV        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

```

12.5.2 定时器 0 (模式 1—16 位不自动重载), 用作定时

C 语言代码

//测试工作频率为 11.0592MHz

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

sfr P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

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

    TMOD = 0x01;                                //模式 I
    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);
}
```

汇编代码

;测试工作频率为 11.0592MHz

P0M1	DATA	093H
P0M0	DATA	094H

<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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>0100H</i>
TM0ISR:		
	<i>MOV</i>	<i>TL0,#66H</i>
	<i>MOV</i>	<i>TH0,#0FCH</i>
	<i>CPL</i>	<i>P1.0</i>
	<i>RETI</i>	<i>; 测试端口</i>
MAIN:		
	<i>MOV</i>	<i>SP, #5FH</i>
	<i>MOV</i>	<i>P0M0, #00H</i>
	<i>MOV</i>	<i>P0M1, #00H</i>
	<i>MOV</i>	<i>P1M0, #00H</i>
	<i>MOV</i>	<i>P1M1, #00H</i>
	<i>MOV</i>	<i>P2M0, #00H</i>
	<i>MOV</i>	<i>P2M1, #00H</i>
	<i>MOV</i>	<i>P3M0, #00H</i>
	<i>MOV</i>	<i>P3M1, #00H</i>
	<i>MOV</i>	<i>P4M0, #00H</i>
	<i>MOV</i>	<i>P4M1, #00H</i>
	<i>MOV</i>	<i>P5M0, #00H</i>
	<i>MOV</i>	<i>P5M1, #00H</i>
	<i>MOV</i>	<i>TMOD,#01H</i>
	<i>MOV</i>	<i>TL0,#66H</i>
	<i>MOV</i>	<i>TH0,#0FCH</i>
	<i>SETB</i>	<i>TR0</i>
	<i>SETB</i>	<i>ET0</i>
	<i>SETB</i>	<i>EA</i>
	<i>JMP</i>	<i>\$</i>
	END	

12.5.3 定时器 0 (模式 2—8 位自动重载), 用作定时

C 语言代码

```
//测试工作频率为 11.0592MHz
```

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

```
#include "intrins.h"

sfr P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

sbit P10      = P1^0;

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

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    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);
}
```

汇编代码

; 测试工作频率为 11.0592MHz

P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H

<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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>0100H</i>
<i>TM0ISR:</i>	<i>CPL</i>	<i>P1.0</i>
	<i>RETI</i>	; 测试端口
<i>MAIN:</i>		
	<i>MOV</i>	<i>SP, #5FH</i>
	<i>MOV</i>	<i>P0M0, #00H</i>
	<i>MOV</i>	<i>P0M1, #00H</i>
	<i>MOV</i>	<i>P1M0, #00H</i>
	<i>MOV</i>	<i>P1M1, #00H</i>
	<i>MOV</i>	<i>P2M0, #00H</i>
	<i>MOV</i>	<i>P2M1, #00H</i>
	<i>MOV</i>	<i>P3M0, #00H</i>
	<i>MOV</i>	<i>P3M1, #00H</i>
	<i>MOV</i>	<i>P4M0, #00H</i>
	<i>MOV</i>	<i>P4M1, #00H</i>
	<i>MOV</i>	<i>P5M0, #00H</i>
	<i>MOV</i>	<i>P5M1, #00H</i>
	<i>MOV</i>	<i>TMOD, #02H</i>
	<i>MOV</i>	<i>TL0, #0F4H</i>
	<i>MOV</i>	<i>TH0, #0F4H</i>
	<i>SETB</i>	<i>TR0</i>
	<i>SETB</i>	<i>ET0</i>
	<i>SETB</i>	<i>EA</i>
	<i>JMP</i>	\$
 <i>END</i>		

12.5.4 定时器 0 (模式 3—16 位自动重载不可屏蔽中断), 用作定时

C 语言代码

```
//测试工作频率为 11.0592MHz
```

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

sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
```

```

sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

sbit P10      = P1^0;

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

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    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);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH
ORG		0000H
LJMP		MAIN
ORG		000BH

LJMP	TM0ISR	
ORG	0100H	
TM0ISR:		
CPL	P1.0	
RETI	; 测试端口	
MAIN:		
MOV	SP, #5FH	
MOV	P0M0, #00H	
MOV	P0M1, #00H	
MOV	P1M0, #00H	
MOV	P1M1, #00H	
MOV	P2M0, #00H	
MOV	P2M1, #00H	
MOV	P3M0, #00H	
MOV	P3M1, #00H	
MOV	P4M0, #00H	
MOV	P4M1, #00H	
MOV	P5M0, #00H	
MOV	P5M1, #00H	
MOV	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		

12.5.5 定时器 0 (外部计数—扩展 T0 为外部下降沿中断)

C 语言代码

// 测试工作频率为 11.0592MHz

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

sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P4M1 = 0xb3;
sfr P4M0 = 0xb4;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;

sbit P10 = P1^0;
```

```

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

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    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);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

```

P0M1      DATA      093H
P0M0      DATA      094H
P1M1      DATA      091H
P1M0      DATA      092H
P2M1      DATA      095H
P2M0      DATA      096H
P3M1      DATA      0B1H
P3M0      DATA      0B2H
P4M1      DATA      0B3H
P4M0      DATA      0B4H
P5M1      DATA      0C9H
P5M0      DATA      0CAH

        ORG      0000H
        LJMP     MAIN
        ORG      000BH
        LJMP     TM0ISR

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

MAIN:

```

```

MOV      SP, #5FH
MOV      P0M0, #00H
MOV      P0M1, #00H
MOV      P1M0, #00H
MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

MOV      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

```

12.5.6 定时器 0 (测量脉宽—INT0 高电平宽度)

C 语言代码

```

//测试工作频率为 11.0592MHz

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

sfr    AUXR      = 0x8e;

sfr    P0M1      = 0x93;
sfr    P0M0      = 0x94;
sfr    P1M1      = 0x91;
sfr    P1M0      = 0x92;
sfr    P2M1      = 0x95;
sfr    P2M0      = 0x96;
sfr    P3M1      = 0xb1;
sfr    P3M0      = 0xb2;
sfr    P4M1      = 0xb3;
sfr    P4M0      = 0xb4;
sfr    P5M1      = 0xc9;
sfr    P5M0      = 0xca;

void INT0_Isr() interrupt 0
{
    P0 = TL0;           //TL0 为测量值低字节
    P1 = TH0;           //TH0 为测量值高字节
}

void main()

```

```

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

    AUXR = 0x80;           //IT 模式
    TMOD = 0x08;           //使能GATE,INT0 为1 时使能计时
    TL0 = 0x00;
    TH0 = 0x00;
    while (P32);          //等待INT0 为低
    TR0 = 1;                //启动定时器
    IT0 = 1;                //使能INT0 下降沿中断
    EX0 = 1;
    EA = 1;

    while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

```

AUXR      DATA      8EH
P0M1      DATA      093H
P0M0      DATA      094H
P1M1      DATA      091H
P1M0      DATA      092H
P2M1      DATA      095H
P2M0      DATA      096H
P3M1      DATA      0B1H
P3M0      DATA      0B2H
P4M1      DATA      0B3H
P4M0      DATA      0B4H
P5M1      DATA      0C9H
P5M0      DATA      0CAH

        ORG      0000H
        LJMP     MAIN
        ORG      0003H
        LJMP     INT0ISR

        ORG      0100H
INT0ISR:
        MOV      P0,TL0      ;TL0 为测量值低字节
        MOV      P1,TH0      ;TH0 为测量值高字节
        RETI

MAIN:
        MOV      SP, #5FH
        MOV      P0M0, #00H

```

```

MOV      P0M1, #00H
MOV      P1M0, #00H
MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

MOV      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      P3.2,$              ;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

12.5.7 定时器 0 (模式 0), 时钟分频输出

C 语言代码

//测试工作频率为 11.0592MHz

```

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

sfr    INTCLKO    = 0x8f;

sfr    P0M1        = 0x93;
sfr    P0M0        = 0x94;
sfr    P1M1        = 0x91;
sfr    P1M0        = 0x92;
sfr    P2M1        = 0x95;
sfr    P2M0        = 0x96;
sfr    P3M1        = 0xb1;
sfr    P3M0        = 0xb2;
sfr    P4M1        = 0xb3;
sfr    P4M0        = 0xb4;
sfr    P5M1        = 0xc9;
sfr    P5M0        = 0xca;

```

```

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
}

```

```

P2M0 = 0x00;
P2M1 = 0x00;
P3M0 = 0x00;
P3M1 = 0x00;
P4M0 = 0x00;
P4M1 = 0x00;
P5M0 = 0x00;
P5M1 = 0x00;

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);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

```

INTCLKO    DATA      8FH
P0M1        DATA      093H
P0M0        DATA      094H
P1M1        DATA      091H
P1M0        DATA      092H
P2M1        DATA      095H
P2M0        DATA      096H
P3M1        DATA      0B1H
P3M0        DATA      0B2H
P4M1        DATA      0B3H
P4M0        DATA      0B4H
P5M1        DATA      0C9H
P5M0        DATA      0CAH

ORG          0000H
LJMP         MAIN

ORG          0100H

MAIN:
MOV          SP, #5FH
MOV          P0M0, #00H
MOV          P0M1, #00H
MOV          P1M0, #00H
MOV          P1M1, #00H
MOV          P2M0, #00H
MOV          P2M1, #00H
MOV          P3M0, #00H
MOV          P3M1, #00H
MOV          P4M0, #00H
MOV          P4M1, #00H
MOV          P5M0, #00H
MOV          P5M1, #00H

MOV          TMOD,#00H          ;模式0
MOV          TL0,#66H            ;65536-11.0592M/12/1000
MOV          TH0,#0FCH
SETB         TR0                ;启动定时器

```

MOV	INTCLK0,#01H	;使能时钟输出
JMP	\$	
END		

12.5.8 定时器1（模式0—16位自动重载），用作定时

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

sbit P10      = P1^0;

void T0_0_ISR() interrupt 3
{
    P10 = !P10;                                //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

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

```

while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

```

P0M1      DATA      093H
P0M0      DATA      094H
P1M1      DATA      091H
P1M0      DATA      092H
P2M1      DATA      095H
P2M0      DATA      096H
P3M1      DATA      0B1H
P3M0      DATA      0B2H
P4M1      DATA      0B3H
P4M0      DATA      0B4H
P5M1      DATA      0C9H
P5M0      DATA      0CAH

        ORG      0000H
        LJMP    MAIN
        ORG      001BH
        LJMP    TM1ISR

        ORG      0100H
TM1ISR:
        CPL     P1.0           ; 测试端口
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     TMOD, #00H          ; 模式 0
        MOV     TLI, #66H           ; 65536-11.0592M/12/1000
        MOV     TH1, #0FCH
        SETB   TR1                 ; 启动定时器
        SETB   ET1                 ; 使能定时器中断
        SETB   EA

        JMP     $

```

12.5.9 定时器 1 (模式 1—16 位不自动重载), 用作定时

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

sbit P10      = P1^0;

void TMI_Isr() interrupt 3
{
    TL1 = 0x66;                                //重设定时参数
    TH1 = 0xfc;
    P10 = !P10;                                //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x10;                                //模式 1
    //65536-11.0592M/12/1000
    TL1 = 0x66;
    TH1 = 0xfc;
    TR1 = 1;                                    //启动定时器
    ET1 = 1;                                    //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

;测试工作频率为 11.0592MHz

<i>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</i>
<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>ORG</i>	<i>0100H</i>	
TMIISR:		
<i>MOV</i>	<i>TL1,#66H</i>	;重设定时参数
<i>MOV</i>	<i>TH1,#0FCH</i>	
<i>CPL</i>	<i>P1.0</i>	;测试端口
<i>RETI</i>		
MAIN:		
<i>MOV</i>	<i>SP, #5FH</i>	
<i>MOV</i>	<i>P0M0, #00H</i>	
<i>MOV</i>	<i>P0M1, #00H</i>	
<i>MOV</i>	<i>P1M0, #00H</i>	
<i>MOV</i>	<i>P1M1, #00H</i>	
<i>MOV</i>	<i>P2M0, #00H</i>	
<i>MOV</i>	<i>P2M1, #00H</i>	
<i>MOV</i>	<i>P3M0, #00H</i>	
<i>MOV</i>	<i>P3M1, #00H</i>	
<i>MOV</i>	<i>P4M0, #00H</i>	
<i>MOV</i>	<i>P4M1, #00H</i>	
<i>MOV</i>	<i>P5M0, #00H</i>	
<i>MOV</i>	<i>P5M1, #00H</i>	
<i>MOV</i>	<i>TMOD,#10H</i>	;模式 1
<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>SETB</i>	<i>EA</i>	
<i>JMP</i>	\$	
 END		

12.5.10 定时器 1 (模式 2—8 位自动重载), 用作定时

C 语言代码

//测试工作频率为 11.0592MHz

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

sfr P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

sbit P10      = P1^0;

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

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    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);
}

```

汇编代码

;测试工作频率为 11.0592MHz

P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H

<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</i>
	<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>ORG</i>	<i>0100H</i>
TMIISR:	<i>CPL</i>	<i>PI.0</i>
	<i>RETI</i>	; 测试端口
MAIN:		
	<i>MOV</i>	<i>SP, #5FH</i>
	<i>MOV</i>	<i>P0M0, #00H</i>
	<i>MOV</i>	<i>P0M1, #00H</i>
	<i>MOV</i>	<i>P1M0, #00H</i>
	<i>MOV</i>	<i>P1M1, #00H</i>
	<i>MOV</i>	<i>P2M0, #00H</i>
	<i>MOV</i>	<i>P2M1, #00H</i>
	<i>MOV</i>	<i>P3M0, #00H</i>
	<i>MOV</i>	<i>P3M1, #00H</i>
	<i>MOV</i>	<i>P4M0, #00H</i>
	<i>MOV</i>	<i>P4M1, #00H</i>
	<i>MOV</i>	<i>P5M0, #00H</i>
	<i>MOV</i>	<i>P5M1, #00H</i>
	<i>MOV</i>	<i>TMOD, #20H</i>
	<i>MOV</i>	; 模式 2
	<i>MOV</i>	<i>TL1, #0F4H</i>
	<i>MOV</i>	; 256-11.0592M/12/76K
	<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>JMP</i>	<i>EA</i>
	<i>JMP</i>	\$
 END		

12.5.11 定时器 1 (外部计数—扩展 T1 为外部下降沿中断)

C 语言代码

```
//测试工作频率为 11.0592MHz
```

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

sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
```

```

sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

sbit P10      = P1^0;

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

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x40;                                // 外部计数模式
    TLI = 0xff;
    TH1 = 0xff;
    TR1 = 1;                                    // 启动定时器
    ET1 = 1;                                    // 使能定时器中断
    EA = 1;

    while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH

```

ORG      0000H
LJMP    MAIN
ORG      001BH
LJMP    TM1ISR

TM1ISR:
ORG      0100H
CPL      P1.0          ;测试端口
RETI

MAIN:
MOV      SP, #5FH
MOV      P0M0, #00H
MOV      P0M1, #00H
MOV      P1M0, #00H
MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

MOV      TMOD, #40H      ;外部计数模式
MOV      TLI, #0FFH
MOV      TH1, #0FFH
SETB    TR1              ;启动定时器
SETB    ET1              ;使能定时器中断
SETB    EA

JMP      $

END

```

12.5.12 定时器 1 (测量脉宽—INT1 高电平宽度)

C 语言代码

//测试工作频率为 11.0592MHz

```

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

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

```

```

sfr      P5M1      =  0xc9;
sfr      P5M0      =  0xca;

sfr      AUXR      =  0x8e;

void INT1_Isr() interrupt 2
{
    P0 = TL1;                      //TL1 为测量值低字节
    P1 = TH1;                      //TH1 为测量值高字节
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    AUXR = 0x40;                  //IT 模式
    TMOD = 0x80;                  //使能 GATE,INT1 为 1 时使能计时
    TL1 = 0x00;
    TH1 = 0x00;
    while (INT1);                //等待 INT1 为低
    TR1 = 1;                      //启动定时器
    IT1 = 1;                      //使能 INT1 下降沿中断
    EX1 = 1;
    EA = 1;

    while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

AUXR	DATA	8EH
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH
ORG	DATA	0000H
LJMP	DATA	MAIN

<i>ORG</i>	<i>0013H</i>	
<i>LJMP</i>	<i>INTIISR</i>	
<i>INTIISR:</i>	<i>ORG</i> <i>0100H</i>	
	<i>MOV</i> <i>P0,TL1</i>	<i>;TL1</i> 为测量值低字节
	<i>MOV</i> <i>P1,TH1</i>	<i>;TH1</i> 为测量值高字节
	<i>RETI</i>	
 <i>MAIN:</i>		
<i>MOV</i>	<i>SP, #5FH</i>	
<i>MOV</i>	<i>P0M0, #00H</i>	
<i>MOV</i>	<i>P0M1, #00H</i>	
<i>MOV</i>	<i>P1M0, #00H</i>	
<i>MOV</i>	<i>P1M1, #00H</i>	
<i>MOV</i>	<i>P2M0, #00H</i>	
<i>MOV</i>	<i>P2M1, #00H</i>	
<i>MOV</i>	<i>P3M0, #00H</i>	
<i>MOV</i>	<i>P3M1, #00H</i>	
<i>MOV</i>	<i>P4M0, #00H</i>	
<i>MOV</i>	<i>P4M1, #00H</i>	
<i>MOV</i>	<i>P5M0, #00H</i>	
<i>MOV</i>	<i>P5M1, #00H</i>	
<i>MOV</i>	<i>AUXR,#40H</i>	<i>;IT 模式</i>
<i>MOV</i>	<i>TMOD,#80H</i>	<i>;使能 GATE, INT1 为 1 时使能计时</i>
<i>MOV</i>	<i>TL1,#00H</i>	
<i>MOV</i>	<i>TH1,#00H</i>	
<i>JB</i>	<i>INT1,\$</i>	<i>;等待 INT1 为低</i>
<i>SETB</i>	<i>TR1</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>EX1</i>	
<i>SETB</i>	<i>EA</i>	
<i>JMP</i>	<i>\$</i>	
 <i>END</i>		

12.5.13 定时器 1 (模式 0), 时钟分频输出

C 语言代码

//测试工作频率为 11.0592MHz

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

sfr    INTCLKO    =  0x8f;
sfr    P0M1        =  0x93;
sfr    P0M0        =  0x94;
sfr    P1M1        =  0x91;
sfr    P1M0        =  0x92;
sfr    P2M1        =  0x95;
sfr    P2M0        =  0x96;
sfr    P3M1        =  0xb1;
```

```

sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x00;           //模式0
    TL1 = 0x66;           //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;              //启动定时器
    INTCLKO = 0x02;        //使能时钟输出

    while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

```

INTCLKO    DATA    8FH
P0M1       DATA    093H
P0M0       DATA    094H
P1M1       DATA    091H
P1M0       DATA    092H
P2M1       DATA    095H
P2M0       DATA    096H
P3M1       DATA    0B1H
P3M0       DATA    0B2H
P4M1       DATA    0B3H
P4M0       DATA    0B4H
P5M1       DATA    0C9H
P5M0       DATA    0CAH

        ORG      0000H
        LJMP    MAIN

        ORG      0100H
MAIN:
        MOV      SP, #5FH
        MOV      P0M0, #00H
        MOV      P0M1, #00H
        MOV      P1M0, #00H
        MOV      P1M1, #00H
        MOV      P2M0, #00H

```

```

MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

MOV      TMOD, #00H          ;模式0
MOV      TL1, #66H           ;65536-11.0592M/12/1000
MOV      TH1, #0FCH
SETB    TR1                  ;启动定时器
MOV      INTCLKO, #02H       ;使能时钟输出

JMP      $

```

END

12.5.14 定时器1（模式0）做串口1波特率发生器

C 语言代码

//测试工作频率为11.0592MHz

```

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

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

sfr    AUXR      = 0x8e;

sfr    P0M1      = 0x93;
sfr    P0M0      = 0x94;
sfr    P1M1      = 0x91;
sfr    P1M0      = 0x92;
sfr    P2M1      = 0x95;
sfr    P2M0      = 0x96;
sfr    P3M1      = 0xb1;
sfr    P3M0      = 0xb2;
sfr    P4M1      = 0xb3;
sfr    P4M0      = 0xb4;
sfr    P5M1      = 0xc9;
sfr    P5M0      = 0xca;

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;
    TLI = 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()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !r\n");

    while (1)
{
```

```

if (rptr != wptr)
{
    UartSend(buffer[rptr++]);
    rptr &= 0x0f;
}
}

```

汇编代码

; 测试工作频率为 11.0592MHz

<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>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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>

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>

CHKRI:

<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>

UARTISR_EXIT:

<i>POP</i>	<i>PSW</i>
<i>POP</i>	<i>ACC</i>

RETI***UART_INIT:***

```

MOV      SCON,#50H
MOV      TMOD,#00H
MOV      TLI,#0E8H           ;65536-II059200/II5200/4=0FFE8H
MOV      TH1,#0FFH
SETB    TR1
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, #5FH
MOV      P0M0, #00H
MOV      P0M1, #00H
MOV      P1M0, #00H
MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

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

```

<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>
<i>STRING:</i>	<i>DB</i>
	<i>'Uart Test !',0DH,0AH,00H</i>
	<i>END</i>

12.5.15 定时器 1（模式 2）做串口 1 波特率发生器

C 语言代码

//测试工作频率为11.0592MHz

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

#define FOSC      11059200UL
#define BRT       (256 - FOSC / 115200 / 32)

sfr AUXR      = 0x8e;

sfr P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

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

```
    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

;测试工作频率为 11.0592MHz

AUXR	DATA	8EH
BUSY	BIT	20H.0
WPTR	DATA	21H
RPTR	DATA	22H
BUFFER	DATA	23H
		<i>;16 bytes</i>
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH
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		TLI,#0FDH
MOV		<i>;256-11059200/115200/32=0FDH</i>
		TH1,#0FDH

<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>	

UART_SEND:

<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, #5FH</i>
<i>MOV</i>	<i>P0M0, #00H</i>
<i>MOV</i>	<i>P0M1, #00H</i>
<i>MOV</i>	<i>P1M0, #00H</i>
<i>MOV</i>	<i>P1M1, #00H</i>
<i>MOV</i>	<i>P2M0, #00H</i>
<i>MOV</i>	<i>P2M1, #00H</i>
<i>MOV</i>	<i>P3M0, #00H</i>
<i>MOV</i>	<i>P3M1, #00H</i>
<i>MOV</i>	<i>P4M0, #00H</i>
<i>MOV</i>	<i>P4M1, #00H</i>
<i>MOV</i>	<i>P5M0, #00H</i>
<i>MOV</i>	<i>P5M1, #00H</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: **DB** **'Uart Test !',0DH,0AH,00H**

END

12.5.16 定时器 2 (16 位自动重载), 用作定时

C 语言代码

//测试工作频率为 11.0592MHz

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

sfr T2L = 0xd7;
sfr T2H = 0xd6;
sfr AUXR = 0x8e;
sfr IE2 = 0xaf;
#define ET2 0x04
sfr AUXINTIF = 0xef;
#define T2IF 0x01

sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P4M1 = 0xb3;
sfr P4M0 = 0xb4;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;

sbit P10 = P1^0;

void TM2_Isr() interrupt 12
{
    P10 = !P10; //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;
```

```

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);
}

```

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

P0M1     DATA    093H
P0M0     DATA    094H
P1M1     DATA    091H
P1M0     DATA    092H
P2M1     DATA    095H
P2M0     DATA    096H
P3M1     DATA    0B1H
P3M0     DATA    0B2H
P4M1     DATA    0B3H
P4M0     DATA    0B4H
P5M1     DATA    0C9H
P5M0     DATA    0CAH

        ORG     0000H
        LJMP   MAIN
        ORG     0063H
        LJMP   TM2ISR

        ORG     0100H
TM2ISR:
        CPL    P1.0          ; 测试端口
        RETI

MAIN:
        MOV    SP, #5FH
        MOV    P0M0, #00H
        MOV    P0M1, #00H
        MOV    P1M0, #00H
        MOV    P1M1, #00H
        MOV    P2M0, #00H
        MOV    P2M1, #00H
        MOV    P3M0, #00H
        MOV    P3M1, #00H
        MOV    P4M0, #00H
        MOV    P4M1, #00H
        MOV    P5M0, #00H
        MOV    P5M1, #00H

```

<i>MOV</i>	<i>T2L,#66H</i>	<i>;65536-11.0592M/12/1000</i>
<i>MOV</i>	<i>T2H,#0FCH</i>	
<i>MOV</i>	<i>AUXR,#10H</i>	<i>;启动定时器</i>
<i>MOV</i>	<i>IE2,#ET2</i>	<i>;使能定时器中断</i>
<i>SETB</i>	<i>EA</i>	
<i>JMP</i>	<i>\$</i>	
 <i>END</i>		

12.5.17 定时器 2 (外部计数—扩展 T2 为外部下降沿中断)

C 语言代码

//测试工作频率为 11.0592MHz

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

sfr    T2L      = 0xd7;
sfr    T2H      = 0xd6;
sfr    AUXR     = 0x8e;
sfr    IE2      = 0xaf;
#define ET2      0x04
sfr    AUXINTIF = 0xef;
#define T2IF     0x01

sfr    P0M1    = 0x93;
sfr    P0M0    = 0x94;
sfr    P1M1    = 0x91;
sfr    P1M0    = 0x92;
sfr    P2M1    = 0x95;
sfr    P2M0    = 0x96;
sfr    P3M1    = 0xb1;
sfr    P3M0    = 0xb2;
sfr    P4M1    = 0xb3;
sfr    P4M0    = 0xb4;
sfr    P5M1    = 0xc9;
sfr    P5M0    = 0xca;

sbit   P10     = P1^0;

void TM2_Isr() interrupt 12
{
    P10 = !P10;                                //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
```

```

P3M1 = 0x00;
P4M0 = 0x00;
P4M1 = 0x00;
P5M0 = 0x00;
P5M1 = 0x00;

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 = I;

while (1);
}

```

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
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH
ORG	0000H	
LJMP	MAIN	
ORG	0063H	
LJMP	TM2ISR	
ORG	0100H	
TM2ISR:	CPL	PI.0 ; 测试端口
	RETI	
MAIN:		
MOV	SP, #5FH	
MOV	P0M0, #00H	
MOV	P0M1, #00H	
MOV	P1M0, #00H	
MOV	P1M1, #00H	
MOV	P2M0, #00H	
MOV	P2M1, #00H	
MOV	P3M0, #00H	

```

MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

MOV      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

```

12.5.18 定时器 2, 时钟分频输出

C 语言代码

```

//测试工作频率为11.0592MHz

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

sfr    T2L      = 0xd7;
sfr    T2H      = 0xd6;
sfr    AUXR     = 0x8e;
sfr    INTCLKO  = 0x8f;

sfr    P0M1     = 0x93;
sfr    P0M0     = 0x94;
sfr    P1M1     = 0x91;
sfr    P1M0     = 0x92;
sfr    P2M1     = 0x95;
sfr    P2M0     = 0x96;
sfr    P3M1     = 0xb1;
sfr    P3M0     = 0xb2;
sfr    P4M1     = 0xb3;
sfr    P4M0     = 0xb4;
sfr    P5M1     = 0xc9;
sfr    P5M0     = 0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
}
```

```

P5M1 = 0x00;

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);
}

```

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

P0M1     DATA    093H
P0M0     DATA    094H
P1M1     DATA    091H
P1M0     DATA    092H
P2M1     DATA    095H
P2M0     DATA    096H
P3M1     DATA    0B1H
P3M0     DATA    0B2H
P4M1     DATA    0B3H
P4M0     DATA    0B4H
P5M1     DATA    0C9H
P5M0     DATA    0CAH

        ORG      0000H
        LJMP    MAIN

        ORG      0100H

MAIN:
        MOV      SP, #5FH
        MOV      P0M0, #00H
        MOV      P0M1, #00H
        MOV      P1M0, #00H
        MOV      P1M1, #00H
        MOV      P2M0, #00H
        MOV      P2M1, #00H
        MOV      P3M0, #00H
        MOV      P3M1, #00H
        MOV      P4M0, #00H
        MOV      P4M1, #00H
        MOV      P5M0, #00H
        MOV      P5M1, #00H

        MOV      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

12.5.19 定时器 2 做串口 1 波特率发生器

C 语言代码

//测试工作频率为 11.0592MHz

```
#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 P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

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

    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}

```

汇编代码

; 测试工作频率为 11.0592MHz

AUXR	DATA	8EH
T2H	DATA	0D6H
T2L	DATA	0D7H
BUSY	BIT	20H.0

<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>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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>
------------	--------------

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>

CHKRI:

<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>

UARTISR_EXIT:

<i>POP</i>	<i>PSW</i>
<i>POP</i>	<i>ACC</i>
<i>RETI</i>	

UART_INIT:

<i>MOV</i>	<i>SCON,#50H</i>
<i>MOV</i>	<i>T2L,#0E8H</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>	

;65536-11059200/115200/4=0FFE8H

UART_SEND:

<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, #5FH</i>
<i>MOV</i>	<i>P0M0, #00H</i>
<i>MOV</i>	<i>P0M1, #00H</i>
<i>MOV</i>	<i>P1M0, #00H</i>
<i>MOV</i>	<i>P1M1, #00H</i>
<i>MOV</i>	<i>P2M0, #00H</i>
<i>MOV</i>	<i>P2M1, #00H</i>
<i>MOV</i>	<i>P3M0, #00H</i>
<i>MOV</i>	<i>P3M1, #00H</i>
<i>MOV</i>	<i>P4M0, #00H</i>
<i>MOV</i>	<i>P4M1, #00H</i>
<i>MOV</i>	<i>P5M0, #00H</i>
<i>MOV</i>	<i>P5M1, #00H</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>

<i>STRING:</i>	<i>DB</i>	<i>'Uart Test !',0DH,0AH,00H</i>
----------------	-----------	----------------------------------

<i>END</i>	
------------	--

12.5.20 定时器 2 做串口 2 波特率发生器

C 语言代码

//测试工作频率为 11.0592MHz

```
#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;

sfr P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

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

    Uart2Init();
    IE2 = 0x01;
    EA = 1;
    Uart2SendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart2Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}

```

汇编代码

; 测试工作频率为 11.0592MHz

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

<i>RPTR</i>	<i>DATA</i>	<i>22H</i>
<i>BUFFER</i>	<i>DATA</i>	<i>23H</i>

;16 bytes

<i>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</i>

<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.I,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>

```

MOV      S2BUFA,A
RET

```

UART2_SENDSTR:

```

CLR      A
MOVC    A,@A+DPTR
JZ      SEND2END
LCALL   UART2_SEND
INC     DPTR
JMP     UART2_SENDSTR

```

SEND2END:

```

RET

```

MAIN:

```

MOV      SP, #5FH
MOV      P0M0, #00H
MOV      P0M1, #00H
MOV      P1M0, #00H
MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

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

```

12.5.21 定时器 2 做串口 3 波特率发生器

C 语言代码

//测试工作频率为 11.0592MHz

```
#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;

sfr P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

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

    Uart3Init();
    IE2 = 0x08;
    EA = 1;
    Uart3SendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart3Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}

```

汇编代码

; 测试工作频率为 11.0592MHz

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

<i>RPTR</i>	<i>DATA</i>	<i>22H</i>	
<i>BUFFER</i>	<i>DATA</i>	<i>23H</i>	<i>;16 bytes</i>
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>	
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>	
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>	
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>	
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>	
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>	
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>	
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>	
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>	
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>	
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>	
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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>	
<i>UART3_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,S3CON</i>	
<i>JNB</i>		<i>ACC.I,CHKRI</i>	
<i>ANL</i>		<i>S3CON,#NOT 02H</i>	
<i>CLR</i>		<i>BUSY</i>	
<i>CHKRI:</i>			
<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>	
<i>UART3ISR_EXIT:</i>			
<i>POP</i>		<i>PSW</i>	
<i>POP</i>		<i>ACC</i>	
<i>RETI</i>			
<i>UART3_INIT:</i>			
<i>MOV</i>		<i>S3CON,#10H</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,#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>			
<i>UART3_SEND:</i>			
<i>JB</i>		<i>BUSY,\$</i>	
<i>SETB</i>		<i>BUSY</i>	

```

MOV      S3BUFA
RET

```

UART3_SENDSTR:

```

CLR      A
MOVC    A,@A+DPTR
JZ      SEND3END
LCALL   UART3_SEND
INC     DPTR
JMP     UART3_SENDSTR

```

SEND3END:

```

RET

```

MAIN:

```

MOV      SP, #5FH
MOV      P0M0, #00H
MOV      P0M1, #00H
MOV      P1M0, #00H
MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

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

```

12.5.22 定时器 2 做串口 4 波特率发生器

C 语言代码

//测试工作频率为 11.0592MHz

```
#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;

sfr P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

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

    Uart4Init();
    IE2 = 0x10;
    EA = 1;
    Uart4SendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart4Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}

```

汇编代码

; 测试工作频率为 11.0592MHz

AUXR	DATA	8EH
T2H	DATA	0D6H
T2L	DATA	0D7H
S4CON	DATA	84H
S4BUF	DATA	85H
IE2	DATA	0AFH
BUSY	BIT	20H.0
WPTR	DATA	21H

<i>RPTR</i>	<i>DATA</i>	<i>22H</i>
<i>BUFFER</i>	<i>DATA</i>	<i>23H</i>

;16 bytes

<i>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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>

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.I,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>S4BUFA,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:

<i>RET</i>	
------------	--

MAIN:

<i>MOV</i>	<i>SP, #5FH</i>
<i>MOV</i>	<i>P0M0, #00H</i>
<i>MOV</i>	<i>P0M1, #00H</i>
<i>MOV</i>	<i>P1M0, #00H</i>
<i>MOV</i>	<i>P1M1, #00H</i>
<i>MOV</i>	<i>P2M0, #00H</i>
<i>MOV</i>	<i>P2M1, #00H</i>
<i>MOV</i>	<i>P3M0, #00H</i>
<i>MOV</i>	<i>P3M1, #00H</i>
<i>MOV</i>	<i>P4M0, #00H</i>
<i>MOV</i>	<i>P4M1, #00H</i>
<i>MOV</i>	<i>P5M0, #00H</i>
<i>MOV</i>	<i>P5M1, #00H</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>

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>UART4_SEND</i>
<i>INC</i>	<i>RPT</i>
<i>JMP</i>	<i>LOOP</i>

<i>STRING:</i>	<i>DB</i>	<i>'Uart Test !',0DH,0AH,00H</i>
----------------	-----------	----------------------------------

<i>END</i>	
------------	--

12.5.23 定时器 3 (16 位自动重载), 用作定时

C 语言代码

//测试工作频率为 11.0592MHz

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

sfr T4T3M = 0xd1;
sfr T4L = 0xd3;
sfr T4H = 0xd2;
sfr T3L = 0xd5;
sfr T3H = 0xd4;
sfr T2L = 0xd7;
sfr T2H = 0xd6;
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

sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P4M1 = 0xb3;
sfr P4M0 = 0xb4;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;

sbit P10 = P1^0;

void TM3_Isr() interrupt 19
{
    P10 = !P10; //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    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);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

T4T3M	DATA	0DIH
T4L	DATA	0D3H
T4H	DATA	0D2H
T3L	DATA	0D5H
T3H	DATA	0D4H
T2L	DATA	0D7H
T2H	DATA	0D6H
AUXR	DATA	8EH
IE2	DATA	0AFH
ET2	EQU	04H
ET3	EQU	20H
ET4	EQU	40H
AUXINTIF	DATA	0EFH
T2IF	EQU	01H
T3IF	EQU	02H
T4IF	EQU	04H
P1M1	DATA	091H
P1M0	DATA	092H
P0M1	DATA	093H
P0M0	DATA	094H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH
ORG		0000H
LJMP		MAIN
ORG		009BH
LJMP		TM3ISR
ORG		0100H
TM3ISR:	CPL	P1.0
	RETI	; 测试端口
MAIN:	MOV	SP, #5FH
	MOV	P0M0, #00H
	MOV	P0M1, #00H
	MOV	P1M0, #00H
	MOV	P1M1, #00H
	MOV	P2M0, #00H

```

MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

MOV      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

```

12.5.24 定时器 3（外部计数—扩展 T3 为外部下降沿中断）

C 语言代码

//测试工作频率为 11.0592MHz

```

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

sfr    T4T3M    = 0xd1;
sfr    T4L      = 0xd3;
sfr    T4H      = 0xd2;
sfr    T3L      = 0xd5;
sfr    T3H      = 0xd4;
sfr    T2L      = 0xd7;
sfr    T2H      = 0xd6;
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

sfr    P1M1     = 0x91;
sfr    P1M0     = 0x92;
sfr    P0M1     = 0x93;
sfr    P0M0     = 0x94;
sfr    P2M1     = 0x95;
sfr    P2M0     = 0x96;
sfr    P3M1     = 0xb1;
sfr    P3M0     = 0xb2;
sfr    P4M1     = 0xb3;
sfr    P4M0     = 0xb4;
sfr    P5M1     = 0xc9;
sfr    P5M0     = 0xca;

```

```

sbit      P10          =    PI^0;

void TM3_Isr() interrupt 19
{
    P10 = !P10;           //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T3L = 0x66;           //65536-11.0592M/12/1000
    T3H = 0xfc;
    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 = I;

    while (1);
}

```

汇编代码

```

;测试工作频率为 11.0592MHz
T4T3M      DATA      0D1H
T4L        DATA      0D3H
T4H        DATA      0D2H
T3L        DATA      0D5H
T3H        DATA      0D4H
T2L        DATA      0D7H
T2H        DATA      0D6H
AUXR       DATA      8EH
IE2        DATA      0AFH
ET2        EQU       04H
ET3        EQU       20H
ET4        EQU       40H
AUXINTIF  DATA      0EFH
T2IF       EQU       01H
T3IF       EQU       02H
T4IF       EQU       04H

P1M1       DATA      091H
P1M0       DATA      092H
P0M1       DATA      093H
P0M0       DATA      094H
P2M1       DATA      095H
P2M0       DATA      096H
P3M1       DATA      0B1H

```

<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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>
	<i>RETI</i>	; 测试端口
<i>MAIN:</i>		
	<i>MOV</i>	<i>SP, #5FH</i>
	<i>MOV</i>	<i>P0M0, #00H</i>
	<i>MOV</i>	<i>P0M1, #00H</i>
	<i>MOV</i>	<i>P1M0, #00H</i>
	<i>MOV</i>	<i>P1M1, #00H</i>
	<i>MOV</i>	<i>P2M0, #00H</i>
	<i>MOV</i>	<i>P2M1, #00H</i>
	<i>MOV</i>	<i>P3M0, #00H</i>
	<i>MOV</i>	<i>P3M1, #00H</i>
	<i>MOV</i>	<i>P4M0, #00H</i>
	<i>MOV</i>	<i>P4M1, #00H</i>
	<i>MOV</i>	<i>P5M0, #00H</i>
	<i>MOV</i>	<i>P5M1, #00H</i>
	<i>MOV</i>	<i>T3L, #66H</i>
	<i>MOV</i>	<i>T3H, #0FCH</i>
	<i>MOV</i>	<i>T4T3M, #0CH</i>
	<i>MOV</i>	<i>IE2, #ET3</i>
	<i>SETB</i>	<i>EA</i>
	<i>JMP</i>	\$
 <i>END</i>		

12.5.25 定时器 3, 时钟分频输出

C 语言代码

//测试工作频率为 11.0592MHz

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

```
sfr T4T3M = 0xd1;
sfr T4L = 0xd3;
sfr T4H = 0xd2;
sfr T3L = 0xd5;
sfr T3H = 0xd4;
sfr T2L = 0xd7;
```

```

sfr    T2H      =  0xd6;
sfr    P1M1     =  0x91;
sfr    P1M0     =  0x92;
sfr    P0M1     =  0x93;
sfr    P0M0     =  0x94;
sfr    P2M1     =  0x95;
sfr    P2M0     =  0x96;
sfr    P3M1     =  0xb1;
sfr    P3M0     =  0xb2;
sfr    P4M1     =  0xb3;
sfr    P4M0     =  0xb4;
sfr    P5M1     =  0xc9;
sfr    P5M0     =  0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    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);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

T4T3M	DATA	0DIH
T4L	DATA	0D3H
T4H	DATA	0D2H
T3L	DATA	0D5H
T3H	DATA	0D4H
T2L	DATA	0D7H
T2H	DATA	0D6H
P1M1	DATA	091H
P1M0	DATA	092H
P0M1	DATA	093H
P0M0	DATA	094H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H

```

P5M1      DATA      0C9H
P5M0      DATA      0CAH

ORG       0000H
LJMP     MAIN

ORG       0100H
MAIN:
MOV      SP, #5FH
MOV      P0M0, #00H
MOV      P0M1, #00H
MOV      P1M0, #00H
MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

MOV      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

```

12.5.26 定时器 3 做串口 3 波特率发生器

C 语言代码

```
//测试工作频率为11.0592MHz
```

```

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

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

sfr      T4T3M      = 0xd1;
sfr      T4L       = 0xd3;
sfr      T4H       = 0xd2;
sfr      T3L       = 0xd5;
sfr      T3H       = 0xd4;
sfr      T2L       = 0xd7;
sfr      T2H       = 0xd6;
sfr      S3CON     = 0xac;
sfr      S3BUF     = 0xad;
sfr      IE2        = 0xaf;

sfr      P0M1      = 0x93;
sfr      P0M0      = 0x94;
sfr      P1M1      = 0x91;

```

```
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;
```

```
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()
{
    P0M0 = 0x00;
```

```

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

Uart3Init();
IE2 = 0x08;
EA = 1;
Uart3SendStr("Uart Test !\r\n");

while (1)
{
    if (rptr != wptr)
    {
        Uart3Send(buffer[rptr++]);
        rptr &= 0x0f;
    }
}
}

```

汇编代码

; 测试工作频率为 11.0592MHz

T4T3M	DATA	0D1H
T4L	DATA	0D3H
T4H	DATA	0D2H
T3L	DATA	0D5H
T3H	DATA	0D4H
T2L	DATA	0D7H
T2H	DATA	0D6H
S3CON	DATA	0ACh
S3BUF	DATA	0ADH
IE2	DATA	0AFH
BUSY	BIT	20H.0
WPTR	DATA	21H
RPTR	DATA	22H
BUFFER	DATA	23H
;16 bytes		
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH

<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.I,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>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>	

;65536-11059200/115200/4=0FFE8H

UART3_SEND:

<i>JB</i>	<i>BUSY,\$</i>
<i>SETB</i>	<i>BUSY</i>
<i>MOV</i>	<i>S3BUFA</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, #5FH</i>
------------	-----------------

<i>MOV</i>	<i>P0M0, #00H</i>
<i>MOV</i>	<i>P0M1, #00H</i>
<i>MOV</i>	<i>P1M0, #00H</i>
<i>MOV</i>	<i>P1M1, #00H</i>
<i>MOV</i>	<i>P2M0, #00H</i>
<i>MOV</i>	<i>P2M1, #00H</i>
<i>MOV</i>	<i>P3M0, #00H</i>
<i>MOV</i>	<i>P3M1, #00H</i>
<i>MOV</i>	<i>P4M0, #00H</i>
<i>MOV</i>	<i>P4M1, #00H</i>
<i>MOV</i>	<i>P5M0, #00H</i>
<i>MOV</i>	<i>P5M1, #00H</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>
<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>UART3_SEND</i>
<i>INC</i>	<i>RPTR</i>
<i>JMP</i>	<i>LOOP</i>
<i>STRING:</i>	<i>DB</i>
	<i>'Uart Test !',0DH,0AH,00H</i>
	<i>END</i>

12.5.27 定时器 4 (16 位自动重载), 用作定时

C 语言代码

//测试工作频率为 11.0592MHz

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

sfr T4T3M = 0xd1;
sfr T4L = 0xd3;
sfr T4H = 0xd2;
sfr T3L = 0xd5;
sfr T3H = 0xd4;
sfr T2L = 0xd7;
sfr T2H = 0xd6;
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

sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

sbit P10      = P1^0;

void TM4_Isr() interrupt 20
{
    P10 = !P10;           // 测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    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);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

T4T3M	DATA	0D1H
T4L	DATA	0D3H
T4H	DATA	0D2H

<i>T3L</i>	<i>DATA</i>	<i>0D5H</i>
<i>T3H</i>	<i>DATA</i>	<i>0D4H</i>
<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>IE2</i>	<i>DATA</i>	<i>0AFH</i>
<i>ET2</i>	<i>EQU</i>	<i>04H</i>
<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>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</i>
	<i>ORG</i>	<i>0000H</i>
	<i>LJMP</i>	<i>MAIN</i>
	<i>ORG</i>	<i>00A3H</i>
	<i>LJMP</i>	<i>TM4ISR</i>
<i>TM4ISR:</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, #5FH</i>
	<i>MOV</i>	<i>P0M0, #00H</i>
	<i>MOV</i>	<i>P0M1, #00H</i>
	<i>MOV</i>	<i>P1M0, #00H</i>
	<i>MOV</i>	<i>P1M1, #00H</i>
	<i>MOV</i>	<i>P2M0, #00H</i>
	<i>MOV</i>	<i>P2M1, #00H</i>
	<i>MOV</i>	<i>P3M0, #00H</i>
	<i>MOV</i>	<i>P3M1, #00H</i>
	<i>MOV</i>	<i>P4M0, #00H</i>
	<i>MOV</i>	<i>P4M1, #00H</i>
	<i>MOV</i>	<i>P5M0, #00H</i>
	<i>MOV</i>	<i>P5M1, #00H</i>
	<i>MOV</i>	<i>T4L,#66H</i>
	<i>MOV</i>	<i>T4H,#0FCH</i>
	<i>MOV</i>	<i>T4T3M,#80H</i>
	<i>MOV</i>	; 启动定时器
	<i>MOV</i>	<i>IE2,#ET4</i>
	<i>SETB</i>	; 使能定时器中断
		<i>EA</i>
	<i>JMP</i>	\$

END

12.5.28 定时器 4 (外部计数—扩展 T4 为外部下降沿中断)

C 语言代码

//测试工作频率为 11.0592MHz

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

sfr    T4T3M      = 0xd1;
sfr    T4L        = 0xd3;
sfr    T4H        = 0xd2;
sfr    T3L        = 0xd5;
sfr    T3H        = 0xd4;
sfr    T2L        = 0xd7;
sfr    T2H        = 0xd6;
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

sfr    P1M1       = 0x91;
sfr    P1M0       = 0x92;
sfr    P0M1       = 0x93;
sfr    P0M0       = 0x94;
sfr    P2M1       = 0x95;
sfr    P2M0       = 0x96;
sfr    P3M1       = 0xb1;
sfr    P3M0       = 0xb2;
sfr    P4M1       = 0xb3;
sfr    P4M0       = 0xb4;
sfr    P5M1       = 0xc9;
sfr    P5M0       = 0xca;

sbit   P10        = P1^0;

void TM4_Isr() interrupt 20
{
    P10 = !P10;           //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
```

```

P2M1 = 0x00;
P3M0 = 0x00;
P3M1 = 0x00;
P4M0 = 0x00;
P4M1 = 0x00;
P5M0 = 0x00;
P5M1 = 0x00;

T4L = 0x66;           //65536-11.0592M/12/1000
T4H = 0xfc;
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);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

T4T3M	DATA	0D1H
T4L	DATA	0D3H
T4H	DATA	0D2H
T3L	DATA	0D5H
T3H	DATA	0D4H
T2L	DATA	0D7H
T2H	DATA	0D6H
AUXR	DATA	8EH
IE2	DATA	0AFH
ET2	EQU	04H
ET3	EQU	20H
ET4	EQU	40H
AUXINTIF	DATA	0EFH
T2IF	EQU	01H
T3IF	EQU	02H
T4IF	EQU	04H
P1M1	DATA	091H
P1M0	DATA	092H
P0M1	DATA	093H
P0M0	DATA	094H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH
ORG	0000H	
LJMP	MAIN	
ORG	00A3H	
LJMP	TM4ISR	
ORG	0100H	
TM4ISR:	CPL	P1.0

; 测试端口

RETI***MAIN:***

```

MOV      SP, #5FH
MOV      P0M0, #00H
MOV      P0M1, #00H
MOV      P1M0, #00H
MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

MOV      T4L,#66H           ;65536-11.0592M/12/1000
MOV      T4H,#0FCH
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

```

12.5.29 定时器 4, 时钟分频输出

C 语言代码

//测试工作频率为 11.0592MHz

```

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

sfr    T4T3M    = 0xd1;
sfr    T4L      = 0xd3;
sfr    T4H      = 0xd2;
sfr    T3L      = 0xd5;
sfr    T3H      = 0xd4;
sfr    T2L      = 0xd7;
sfr    T2H      = 0xd6;

sfr    P1M1     = 0x91;
sfr    P1M0     = 0x92;
sfr    P0M1     = 0x93;
sfr    P0M0     = 0x94;
sfr    P2M1     = 0x95;
sfr    P2M0     = 0x96;
sfr    P3M1     = 0xb1;
sfr    P3M0     = 0xb2;
sfr    P4M1     = 0xb3;
sfr    P4M0     = 0xb4;
sfr    P5M1     = 0xc9;
sfr    P5M0     = 0xca;

```

```

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T4L = 0x66;                                //65536-11.0592M/12/1000
    T4H = 0xfc;                                 //使能时钟输出并启动定时器
    T4T3M = 0x90;

    while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

T4T3M	DATA	0D1H
T4L	DATA	0D3H
T4H	DATA	0D2H
T3L	DATA	0D5H
T3H	DATA	0D4H
T2L	DATA	0D7H
T2H	DATA	0D6H
P1M1	DATA	091H
P1M0	DATA	092H
P0M1	DATA	093H
P0M0	DATA	094H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH
ORG	DATA	0000H
LJMP	DATA	MAIN
ORG	DATA	0100H
MAIN:	MOV	SP, #5FH
	MOV	P0M0, #00H
	MOV	P0M1, #00H
	MOV	P1M0, #00H
	MOV	P1M1, #00H
	MOV	P2M0, #00H

```

MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

MOV      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

```

12.5.30 定时器 4 做串口 4 波特率发生器

C 语言代码

```
//测试工作频率为11.0592MHz
```

```

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

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

sfr    T4T3M      = 0xd1;
sfr    T4L        = 0xd3;
sfr    T4H        = 0xd2;
sfr    T3L        = 0xd5;
sfr    T3H        = 0xd4;
sfr    T2L        = 0xd7;
sfr    T2H        = 0xd6;
sfr    S4CON      = 0x84;
sfr    S4BUF      = 0x85;
sfr    IE2         = 0xaf;

sfr    P0M1       = 0x93;
sfr    P0M0       = 0x94;
sfr    P1M1       = 0x91;
sfr    P1M0       = 0x92;
sfr    P2M1       = 0x95;
sfr    P2M0       = 0x96;
sfr    P3M1       = 0xb1;
sfr    P3M0       = 0xb2;
sfr    P4M1       = 0xb3;
sfr    P4M0       = 0xb4;
sfr    P5M1       = 0xc9;
sfr    P5M0       = 0xca;

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

    Uart4Init();
    IE2 = 0x10;
```

```

EA = I;
Uart4SendStr("Uart Test !\r\n");

```

```

while (1)
{
    if (rptr != wptr)
    {
        Uart4Send(buffer[rptr++]);
        rptr &= 0x0f;
    }
}

```

汇编代码

;测试工作频率为 11.0592MHz

<i>T4T3M</i>	<i>DATA</i>	<i>0DIH</i>
<i>T4L</i>	<i>DATA</i>	<i>0D3H</i>
<i>T4H</i>	<i>DATA</i>	<i>0D2H</i>
<i>T3L</i>	<i>DATA</i>	<i>0D5H</i>
<i>T3H</i>	<i>DATA</i>	<i>0D4H</i>
<i>T2L</i>	<i>DATA</i>	<i>0D7H</i>
<i>T2H</i>	<i>DATA</i>	<i>0D6H</i>
<i>S4CON</i>	<i>DATA</i>	<i>84H</i>
<i>S4BUF</i>	<i>DATA</i>	<i>85H</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>
		;16 bytes
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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>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,A</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, #5FH</i>
<i>MOV</i>	<i>P0M0, #00H</i>
<i>MOV</i>	<i>P0M1, #00H</i>
<i>MOV</i>	<i>P1M0, #00H</i>
<i>MOV</i>	<i>P1M1, #00H</i>
<i>MOV</i>	<i>P2M0, #00H</i>
<i>MOV</i>	<i>P2M1, #00H</i>
<i>MOV</i>	<i>P3M0, #00H</i>
<i>MOV</i>	<i>P3M1, #00H</i>
<i>MOV</i>	<i>P4M0, #00H</i>
<i>MOV</i>	<i>P4M1, #00H</i>
<i>MOV</i>	<i>P5M0, #00H</i>
<i>MOV</i>	<i>P5M1, #00H</i>
<i>LCALL</i>	<i>UART4_INIT</i>

*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

13 串口通信

产品线	串口数量
STC8G1K08 系列	2
STC8G1K08-8Pin 系列	1
STC8G1K08A 系列	1
STC8G2K64S4 系列	4
STC8G2K64S2 系列	2
STC8G1K08T 系列	1
STC15H2K64S4 系列	4

STC8G 系列单片机具有 4 个全双工异步串行通信接口。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。

STC8G 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

13.2.1 串口 1 控制寄存器 (SCON)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SCON	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI

SM0/FE: 当PCON寄存器中的SMOD0位为1时, 该位为帧错误检测标志位。当UART在接收过程中检测到一个无效停止位时, 通过UART接收器将该位置1, 必须由软件清零。当PCON寄存器中的SMOD0位为0时, 该位和SM1一起指定串口1的通信工作模式, 如下表所示:

SM0	SM1	串口1工作模式	功能说明
0	0	模式0	同步移位串行方式
0	1	模式1	可变波特率8位数据方式
1	0	模式2	固定波特率9位数据方式
1	1	模式3	可变波特率9位数据方式

SM2: 允许模式 2 或模式 3 多机通信控制位。当串口 1 使用模式 2 或模式 3 时, 如果 SM2 位为 1 且 REN 位为 1, 则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位(即 RB8)来筛选地址帧, 若 RB8=1, 说明该帧是地址帧, 地址信息可以进入 SBUF, 并使 RI 为 1, 进而在中断服务程序中再进行地址号比较; 若 RB8=0, 说明该帧不是地址帧, 应丢掉且保持 RI=0。在模式 2 或模式 3 中, 如果 SM2 位为 0 且 REN 位为 1, 接收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。

13.2.2 串口 1 数据寄存器 (SBUF)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SBUF	99H								

SBUF: 串口 1 数据接收/发送缓冲区。SBUF 实际是 2 个缓冲器, 读缓冲器和写缓冲器, 两个操作分别对应两个不同的寄存器, 1 个是只写寄存器(写缓冲器), 1 个是只读寄存器(读缓冲器)。对 SBUF 进行读操作, 实际是读取串口接收缓冲区, 对 SBUF 进行写操作则是触发串口开始发送数据。

13.2.3 电源管理寄存器 (PCON)

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

SMOD: 串口 1 波特率控制位

0: 串口 1 的各个模式的波特率都不加倍

1: 串口 1 模式 1 (使用模式 2 的定时器 1 作为波特率发生器时有效)、模式 2、模式 3 (使用模式 2 的定时器 1 作为波特率发生器时有效) 的波特率加倍

SMOD0: 帧错误检测控制位

0: 无帧错检测功能

1: 使能帧错误检测功能。此时 SCON 的 SM0/FE 为 FE 功能, 即为帧错误检测标志位。

13.2.4 辅助寄存器 1 (AUXR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	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 $Fosc/12 \times 6 = Fosc/2$

S1ST2: 串口 1 波特率发生器选择位

0: 选择定时器 1 作为波特率发生器

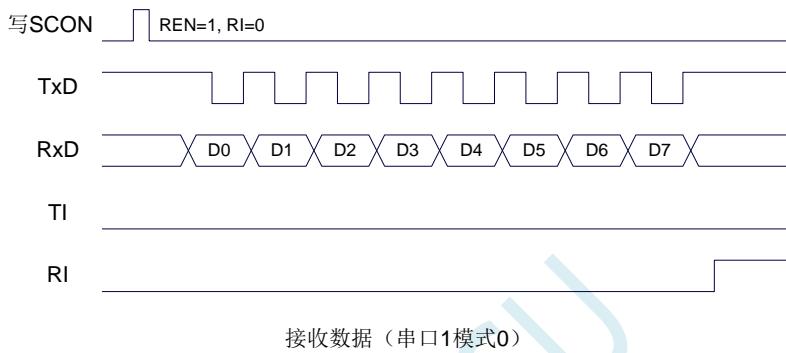
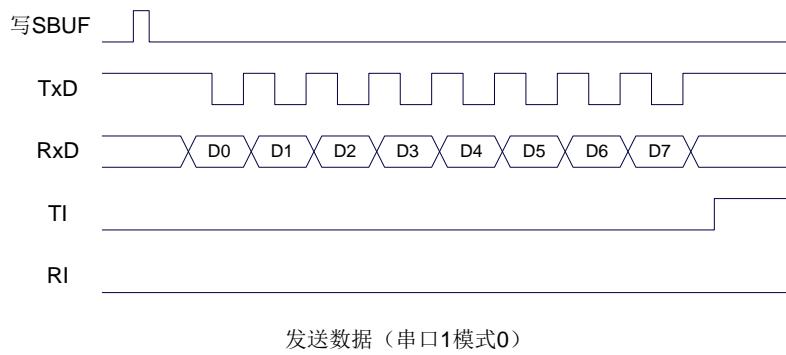
1: 选择定时器 2 作为波特率发生器

13.2.5 串口 1 模式 0, 模式 0 波特率计算公式

当串口 1 选择工作模式为模式 0 时, 串行通信接口工作在同步移位寄存器模式, 当串行口模式 0 的通信速度设置位 UART_M0x6 为 0 时, 其波特率固定为系统时钟时钟的 12 分频 (SYSclk/12); 当设置 UART_M0x6 为 1 时, 其波特率固定为系统时钟频率的 2 分频 (SYSclk/2)。RxD 为串行通讯的数据口, TxD 为同步移位脉冲输出脚, 发送、接收的是 8 位数据, 低位在先。

模式 0 的发送过程: 当主机执行将数据写入发送缓冲器 SBUF 指令时启动发送, 串行口即将 8 位数据以 SYSclk/12 或 SYSclk/2 (由 UART_M0x6 确定是 12 分频还是 2 分频) 的波特率从 RxD 管脚输出(从低位到高位), 发送完中断标志 TI 置 1, TxD 管脚输出同步移位脉冲信号。当写信号有效后, 相隔一个时钟, 发送控制端 SEND 有效(高电平), 允许 RxD 发送数据, 同时允许 TxD 输出同步移位脉冲。一帧(8 位)数据发送完毕时, 各控制端均恢复原状态, 只有 TI 保持高电平, 呈中断申请状态。在再次发送数据前, 必须用软件将 TI 清 0。

模式 0 的接收过程: 首先将接收中断请求标志 RI 清零并置位允许接收控制位 REN 时启动模式 0 接收过程。启动接收过程后, RxD 为串行数据输入端, TxD 为同步脉冲输出端。串行接收的波特率为 SYSclk/12 或 SYSclk/2 (由 UART_M0x6 确定是 12 分频还是 2 分频)。当接收完成一帧数据 (8 位) 后, 控制信号复位, 中断标志 RI 被置 1, 呈中断申请状态。当再次接收时, 必须通过软件将 RI 清 0



工作于模式 0 时，必须清 0 多机通信控制位 SM2，使之不影响 TB8 位和 RB8 位。由于波特率固定为 SYSclk/12 或 SYSclk/2，无需定时器提供，直接由单片机的时钟作为同步移位脉冲。

串口 1 模式 0 的波特率计算公式如下表所示 (SYSclk 为系统工作频率)：

UART_M0x6	波特率计算公式
0	波特率 = $\frac{\text{SYSclk}}{12}$
1	波特率 = $\frac{\text{SYSclk}}{2}$

13.2.6 串口 1 模式 1，模式 1 波特率计算公式

当软件设置 SCON 的 SM0、SM1 为 “01” 时，串行口 1 则以模式 1 进行工作。此模式为 8 位 UART 格式，一帧信息为 10 位：1 位起始位，8 位数据位（低位在先）和 1 位停止位。波特率可变，即可根据需要进行设置波特率。TxD 为数据发送口，RxD 为数据接收口，串行口全双工接受/发送。

模式 1 的发送过程：串行通信模式发送时，数据由串行发送端 TxD 输出。当主机执行一条写 SBUF 的指令就启动串行通信的发送，写 “SBUF” 信号还把 “1” 装入发送移位寄存器的第 9 位，并通知 TX 控制单元开始发送。移位寄存器将数据不断右移送 TxD 端口发送，在数据的左边不断移入 “0” 作补充。当数据的最高位移到移位寄存器的输出位置，紧跟其后的是第 9 位 “1”，在它的左边各位全为 “0”，这个状态条件，使 TX 控制单元作最后一次移位输出，然后使允许发送信号 “SEND” 失效，完成一帧信息的发送，并置位中断请求位 TI，即 TI=1，向主机请求中断处理。

模式 1 的接收过程: 当软件置位接收允许标志位 REN, 即 REN=1 时, 接收器便对 RxD 端口的信号进行检测, 当检测到 RxD 端口发送从“1”→“0”的下降沿跳变时就启动接收器准备接收数据, 并立即复位波特率发生器的接收计数器, 将 1FFH 装入移位寄存器。接收的数据从接收移位寄存器的右边移入, 已装入的 1FFH 向左边移出, 当起始位"0"移到移位寄存器的最左边时, 使 RX 控制器作最后一次移位, 完成一帧的接收。若同时满足以下两个条件:

- RI=0;
- SM2=0 或接收到的停止位为 1。

则接收到的数据有效, 实现装载入 SBUF, 停止位进入 RB8, RI 标志位被置 1, 向主机请求中断, 若上述两条件不能同时满足, 则接收到的数据作废并丢失, 无论条件满足与否, 接收器重又检测 RxD 端口上的"1"→"0"的跳变, 继续下一帧的接收。接收有效, 在响应中断后, RI 标志位必须由软件清 0。通常情况下, 串行通信工作于模式 1 时, SM2 设置为"0"。



串口 1 的波特率是可变的, 其波特率可由定时器 1 或者定时器 2 产生。当定时器采用 1T 模式时 (12 倍速), 相应的波特率的速度也会相应提高 12 倍。

串口 1 模式 1 的波特率计算公式如下表所示: (SYSclk 为系统工作频率)

选择定时器	定时器速度	重装值计算公式	波特率
定时器2	1T	定时器2重载值 = $65536 - \frac{SYSclk}{4 \times \text{波特率}}$	波特率 = $\frac{SYSclk}{4 \times (65536 - \text{定时器重装数})}$
	12T	定时器2重载值 = $65536 - \frac{SYSclk}{12 \times 4 \times \text{波特率}}$	波特率 = $\frac{SYSclk}{12 \times 4 \times (65536 - \text{定时器重装数})}$
定时器1 模式0	1T	定时器1重载值 = $65536 - \frac{SYSclk}{4 \times \text{波特率}}$	波特率 = $\frac{SYSclk}{4 \times (65536 - \text{定时器重装数})}$
	12T	定时器1重载值 = $65536 - \frac{SYSclk}{12 \times 4 \times \text{波特率}}$	波特率 = $\frac{SYSclk}{12 \times 4 \times (65536 - \text{定时器重装数})}$
定时器1 模式2	1T	定时器1重载值 = $256 - \frac{2^{\text{SMOD}} \times SYSclk}{32 \times \text{波特率}}$	波特率 = $\frac{2^{\text{SMOD}} \times SYSclk}{32 \times (256 - \text{定时器重装数})}$
	12T	定时器1重载值 = $256 - \frac{2^{\text{SMOD}} \times SYSclk}{12 \times 32 \times \text{波特率}}$	波特率 = $\frac{2^{\text{SMOD}} \times SYSclk}{12 \times 32 \times (256 - \text{定时器重装数})}$

下面为常用频率与常用波特率所对应定时器的重载值

频率 (MHz)	波特率	定时器 2		定时器 1 模式 0		定时器 1 模式 2			
		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	FFF8H	FFB8H	FFF8H	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

13.2.7 串口1模式2，模式2波特率计算公式

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

模式 2 的波特率固定为系统时钟的 64 分频或 32 分频 (取决于 PCON 中 SMOD 的值)

串口 1 模式 2 的波特率计算公式如下表所示 (SYSclk 为系统工作频率):

SMOD	波特率计算公式
0	波特率 = $\frac{\text{SYSclk}}{64}$
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。

当上述两条条件同时满足时, 才将接收到的移位寄存器的数据装入 SBUF 和 RB8 中, RI 标志位被置 1, 并向主机请求中断处理。如果上述条件有一个不满足, 则刚接收到移位寄存器中的数据无效而丢失, 也不置位 RI。无论上述条件满足与否, 接收器又重新开始检测 RxD 输入端口的跳变信息, 接收下一帧的输入信息。在模式 2 中, 接收到的停止位与 SBUF、RB8 和 RI 无关。

通过软件对 SCON 中的 SM2、TB8 的设置以及通信 D 议的约定, 为多机通信提供了方便。



TxD Start / D0 / D1 / D2 / D3 / D4 / D5 / D6 / D7 / TB8 / Stop

TI

发送数据 (串口1模式2)



RxD Start / D0 / D1 / D2 / D3 / D4 / D5 / D6 / D7 / RB8 / Stop

RI

接收数据 (串口1模式2)

13.2.8 串口 1 模式 3，模式 3 波特率计算公式

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

模式 3 和模式 1 相比，除发送时由 TB8 提供给移位寄存器第 9 数据位不同外，其余功能结构均基本相同，其接收‘发送操作过程及时序也基本相同。

当接收器接收完一帧信息后必须同时满足下列条件：

- RI=0
- SM2=0 或者 SM2=1 且接收到的第 9 数据位 RB8=1。

当上述两条条件同时满足时，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。

13.2.9 自动地址识别

13.2.10 串口 1 从机地址控制寄存器 (SADDR, SA DEN)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SADDR	A9H								
SA DEN	B9H								

SADDR: 从机地址寄存器

SA DEN: 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 SALEN 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

13.3.1 串口 2 控制寄存器 (S2CON)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S2CON	9AH	S2SM0	-	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。

13.3.2 串口 2 数据寄存器 (S2BUF)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S2BUF	9BH								

S2BUF: 串口 1 数据接收/发送缓冲区。S2BUF 实际是 2 个缓冲器, 读缓冲器和写缓冲器, 两个操作分别对应两个不同的寄存器, 1 个是只写寄存器 (写缓冲器), 1 个是只读寄存器 (读缓冲器)。对 S2BUF 进行读操作, 实际是读取串口接收缓冲区, 对 S2BUF 进行写操作则是触发串口开始发送数据。

13.3.3 串口 2 模式 0, 模式 0 波特率计算公式

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	重装值计算公式	波特率
定时器2	1T	定时器2重载值 = $65536 - \frac{SYSclk}{4 \times \text{波特率}}$	波特率 = $\frac{SYSclk}{4 \times (65536 - \text{定时器重装数})}$
	12T	定时器2重载值 = $65536 - \frac{SYSclk}{12 \times 4 \times \text{波特率}}$	波特率 = $\frac{SYSclk}{12 \times 4 \times (65536 - \text{定时器重装数})}$

13.3.4 串口 2 模式 1，模式 1 波特率计算公式

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。

13.4 串口 3

13.4.1 串口 3 控制寄存器 (S3CON)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S3CON	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI

S3SM0: 指定串口3的通信工作模式, 如下表所示:

S3SM0	串口3工作模式	功能说明
0	模式0	可变波特率8位数据方式
1	模式1	可变波特率9位数据方式

S3ST3: 选择串口 3 的波特率发生器

- 0: 选择定时器 2 为串口 3 的波特率发生器
- 1: 选择定时器 3 为串口 3 的波特率发生器

S3SM2: 允许串口 3 在模式 1 时允许多机通信控制位。在模式 1 时, 如果 S3SM2 位为 1 且 S3REN 位为 1, 则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位 (即 S3RB8) 来筛选地址帧: 若 S3RB8=1, 说明该帧是地址帧, 地址信息可以进入 S3BUF, 并使 S3RI 为 1, 进而在中断服务程序中再进行地址号比较; 若 S3RB8=0, 说明该帧不是地址帧, 应丢掉且保持 S3RI=0。在模式 1 中, 如果 S3SM2 位为 0 且 S3REN 位为 1, 接收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。

13.4.2 串口 3 数据寄存器 (S3BUF)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S3BUF	ADH								

S3BUF: 串口 1 数据接收/发送缓冲区。S3BUF 实际是 2 个缓冲器, 读缓冲器和写缓冲器, 两个操作分别对应两个不同的寄存器, 1 个是只写寄存器 (写缓冲器), 1 个是只读寄存器 (读缓冲器)。对 S3BUF 进行读操作, 实际是读取串口接收缓冲区, 对 S3BUF 进行写操作则是触发串口开始发送数据。

13.4.3 串口 3 模式 0, 模式 0 波特率计算公式

串行口 3 的模式 0 为 8 位数据位可变波特率 UART 工作模式。此模式一帧信息为 10 位: 1 位起始位,

8位数据位（低位在先）和1位停止位。波特率可变，可根据需要进行设置波特率。TxD3为数据发送口，RxD3为数据接收口，串行口全双工接受/发送。



串口3的波特率是可变的，其波特率可由定时器2或定时器3产生。当定时器采用1T模式时（12倍速），相应的波特率的速度也会相应提高12倍。

串口3模式0的波特率计算公式如下表所示：(SYSclk为系统工作频率)

选择定时器	定时器速度	重装值计算公式	波特率
定时器2	1T	定时器2重载值 = $65536 - \frac{SYSclk}{4 \times \text{波特率}}$	波特率 = $\frac{SYSclk}{4 \times (65536 - \text{定时器重装数})}$
	12T	定时器2重载值 = $65536 - \frac{SYSclk}{12 \times 4 \times \text{波特率}}$	波特率 = $\frac{SYSclk}{12 \times 4 \times (65536 - \text{定时器重装数})}$
定时器3	1T	定时器3重载值 = $65536 - \frac{SYSclk}{4 \times \text{波特率}}$	波特率 = $\frac{SYSclk}{4 \times (65536 - \text{定时器重装数})}$
	12T	定时器3重载值 = $65536 - \frac{SYSclk}{12 \times 4 \times \text{波特率}}$	波特率 = $\frac{SYSclk}{12 \times 4 \times (65536 - \text{定时器重装数})}$

13.4.4 串口3模式1，模式1波特率计算公式

串行口3的模式1为9位数据位可变波特率UART工作模式。此模式一帧信息为11位：1位起始位，9位数据位（低位在先）和1位停止位。波特率可变，可根据需要进行设置波特率。TxD3为数据发送口，RxD3为数据接收口，串行口全双工接受/发送。



串口 3 模式 1 的波特率计算公式与模式 0 是完全相同的。请参考模式 0 的波特率计算公式。

13.5 串口 4

13.5.1 串口 4 控制寄存器 (S4CON)

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。

13.5.2 串口 4 数据寄存器 (S4BUF)

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。

13.5.3 串口 4 模式 0, 模式 0 波特率计算公式

串行口 4 的模式 0 为 8 位数据位可变波特率 UART 工作模式。此模式一帧信息为 10 位: 1 位起始位, 8 位数据位 (低位在先) 和 1 位停止位。波特率可变, 可根据需要进行设置波特率。TxD4 为数据发送口,

RxD4 为数据接收口，串行口全双工接受/发送。



串口 4 的波特率是可变的，其波特率可由定时器 2 或定时器 4 产生。当定时器采用 1T 模式时（12 倍速），相应的波特率的速度也会相应提高 12 倍。

串口 4 模式 0 的波特率计算公式如下表所示：(SYSclk 为系统工作频率)

选择定时器	定时器速度	重装值计算公式	波特率
定时器2	1T	定时器2重载值 = $65536 - \frac{SYSclk}{4 \times \text{波特率}}$	波特率 = $\frac{SYSclk}{4 \times (65536 - \text{定时器重装数})}$
	12T	定时器2重载值 = $65536 - \frac{SYSclk}{12 \times 4 \times \text{波特率}}$	波特率 = $\frac{SYSclk}{12 \times 4 \times (65536 - \text{定时器重装数})}$
定时器4	1T	定时器4重载值 = $65536 - \frac{SYSclk}{4 \times \text{波特率}}$	波特率 = $\frac{SYSclk}{4 \times (65536 - \text{定时器重装数})}$
	12T	定时器4重载值 = $65536 - \frac{SYSclk}{12 \times 4 \times \text{波特率}}$	波特率 = $\frac{SYSclk}{12 \times 4 \times (65536 - \text{定时器重装数})}$

13.5.4 串口 4 模式 1，模式 1 波特率计算公式

串行口 4 的模式 1 为 9 位数据位可变波特率 UART 工作模式。此模式一帧信息为 11 位：1 位起始位，9 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD4 为数据发送口，RxD4 为数据接收口，串行口全双工接受/发送。

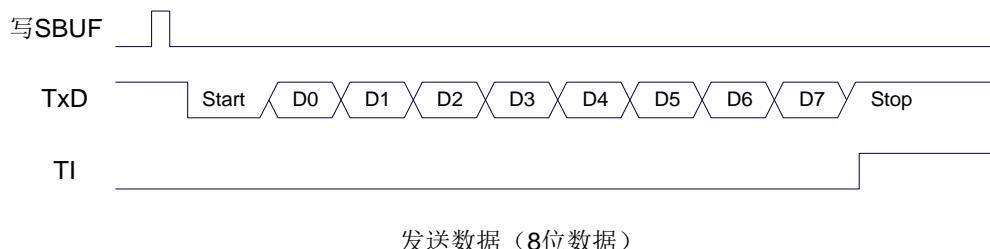


串口 4 模式 1 的波特率计算公式与模式 0 是完全相同的。请参考模式 0 的波特率计算公式。

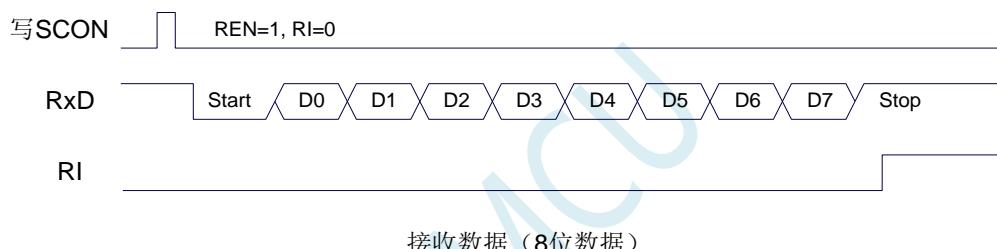
13.6 串口注意事项

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）

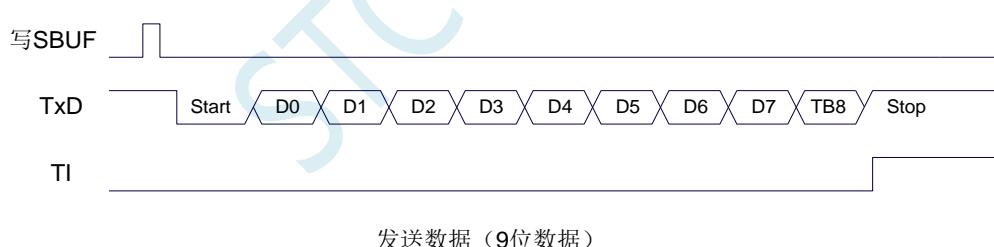
8 位数据模式时，发送完成约 1/3 个停止位后产生 TI 中断请求，如下图所示：



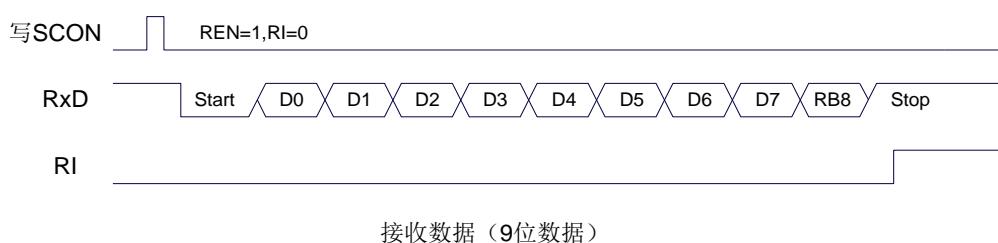
8 位数据模式时，接收完成一半个停止位后产生 RI 中断请求，如下图所示：



9 位数据模式时，发送完成约 1/3 个停止位后产生 TI 中断请求：



9 位数据模式时，一半个停止位后产生 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

C 语言代码

```
//测试工作频率为 11.0592MHz

#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 P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

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

    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}

```

汇编代码

; 测试工作频率为 11.0592MHz

AUXR	DATA	8EH
T2H	DATA	0D6H
T2L	DATA	0D7H

BUSY	BIT	20H.0
WPTR	DATA	21H
RPTR	DATA	22H
BUFFER	DATA	23H

;16 bytes

P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH

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,\$
-----------	----------------

<i>SETB</i>	<i>BUSY</i>
<i>MOV</i>	<i>SBUFA</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, #5FH</i>
<i>MOV</i>	<i>P0M0, #00H</i>
<i>MOV</i>	<i>P0M1, #00H</i>
<i>MOV</i>	<i>P1M0, #00H</i>
<i>MOV</i>	<i>P1M1, #00H</i>
<i>MOV</i>	<i>P2M0, #00H</i>
<i>MOV</i>	<i>P2M1, #00H</i>
<i>MOV</i>	<i>P3M0, #00H</i>
<i>MOV</i>	<i>P3M1, #00H</i>
<i>MOV</i>	<i>P4M0, #00H</i>
<i>MOV</i>	<i>P4M1, #00H</i>
<i>MOV</i>	<i>P5M0, #00H</i>
<i>MOV</i>	<i>P5M1, #00H</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>

<i>STRING:</i>	<i>DB</i>	<i>'Uart Test !',0DH,0AH,00H</i>
----------------	-----------	----------------------------------

<i>END</i>

13.7.2 串口 1 使用定时器 1（模式 0）做波特率发生器

C 语言代码

//测试工作频率为 11.0592MHz

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

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

sfr AUXR      = 0x8e;
sfr P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

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;
    TLI = 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()
{
    P0M0 = 0x00;
    P0MI = 0x00;
    P1M0 = 0x00;
    P1MI = 0x00;
    P2M0 = 0x00;
    P2MI = 0x00;
    P3M0 = 0x00;
    P3MI = 0x00;
    P4M0 = 0x00;
    P4MI = 0x00;
    P5M0 = 0x00;
    P5MI = 0x00;

    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}

```

汇编代码

; 测试工作频率为 11.0592MHz

AUXR	DATA	8EH
BUSY	BIT	20H.0
WPTR	DATA	21H
RPTR	DATA	22H
BUFFER	DATA	23H
		;16 bytes
P0MI	DATA	093H
P0M0	DATA	094H
P1MI	DATA	091H
P1M0	DATA	092H
P2MI	DATA	095H

<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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>

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>

CHKRI:

<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>

UARTISR_EXIT:

<i>POP</i>	<i>PSW</i>
<i>POP</i>	<i>ACC</i>
<i>RETI</i>	

UART_INIT:

<i>MOV</i>	<i>SCON,#50H</i>
<i>MOV</i>	<i>TMOD,#00H</i>
<i>MOV</i>	<i>TLL,#0E8H</i>
<i>MOV</i>	<i>TH1,#0FFH</i>
<i>SETB</i>	<i>TR1</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>	

;65536-11059200/115200/4=0FFE8H

UART_SEND:

<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, #5FH</i>	
<i>MOV</i>	<i>P0M0, #00H</i>	
<i>MOV</i>	<i>P0M1, #00H</i>	
<i>MOV</i>	<i>P1M0, #00H</i>	
<i>MOV</i>	<i>P1M1, #00H</i>	
<i>MOV</i>	<i>P2M0, #00H</i>	
<i>MOV</i>	<i>P2M1, #00H</i>	
<i>MOV</i>	<i>P3M0, #00H</i>	
<i>MOV</i>	<i>P3M1, #00H</i>	
<i>MOV</i>	<i>P4M0, #00H</i>	
<i>MOV</i>	<i>P4M1, #00H</i>	
<i>MOV</i>	<i>P5M0, #00H</i>	
<i>MOV</i>	<i>P5M1, #00H</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:	DB	'Uart Test !',0DH,0AH,00H
END		

13.7.3 串口 1 使用定时器 1（模式 2）做波特率发生器

C 语言代码

//测试工作频率为 11.0592MHz

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

#define FOSC 11059200UL
```

```
#define BRT (256 - FOSC / 115200 / 32)
```

```
sfr AUXR = 0x8e;
```

```
sfr P0M1 = 0x93;
```

```
sfr P0M0 = 0x94;
```

```
sfr P1M1 = 0x91;
```

```
sfr P1M0 = 0x92;
```

```
sfr P2M1 = 0x95;
```

```
sfr P2M0 = 0x96;
```

```
sfr P3M1 = 0xb1;
```

```
sfr P3M0 = 0xb2;
```

```
sfr P4M1 = 0xb3;
```

```
sfr P4M0 = 0xb4;
```

```
sfr P5M1 = 0xc9;
```

```
sfr P5M0 = 0xca;
```

```
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;
```

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

    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}

```

汇编代码

; 测试工作频率为 11.0592MHz

AUXR	DATA	8EH
BUSY	BIT	20H.0
WPTR	DATA	21H
RPTR	DATA	22H
BUFFER	DATA	23H
		<i>;16 bytes</i>
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH

<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>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>

CHKRI:

<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>

UARTISR_EXIT:

<i>POP</i>	<i>PSW</i>
<i>POP</i>	<i>ACC</i>
<i>RETI</i>	

UART_INIT:

<i>MOV</i>	<i>SCON,#50H</i>
<i>MOV</i>	<i>TMOD,#20H</i>
<i>MOV</i>	<i>TLI,#0FDH</i>
<i>MOV</i>	<i>TH1,#0FDH</i>
<i>SETB</i>	<i>TR1</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>	

;256-11059200/115200/32=0FDH

UART_SEND:

<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:

```

MOV      SP, #5FH
MOV      P0M0, #00H
MOV      P0M1, #00H
MOV      P1M0, #00H
MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

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

```

13.7.4 串口 2 使用定时器 2 做波特率发生器

C 语言代码

//测试工作频率为 11.0592MHz

```

#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;  
  
sfr P0M1     = 0x93;  
sfr P0M0     = 0x94;  
sfr P1M1     = 0x91;  
sfr P1M0     = 0x92;  
sfr P2M1     = 0x95;  
sfr P2M0     = 0x96;  
sfr P3M1     = 0xb1;  
sfr P3M0     = 0xb2;  
sfr P4M1     = 0xb3;  
sfr P4M0     = 0xb4;  
sfr P5M1     = 0xc9;  
sfr P5M0     = 0xca;
```

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

    Uart2Init();
    IE2 = 0x01;
    EA = 1;
    Uart2SendStr("Uart Test !r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart2Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}

```

汇编代码

; 测试工作频率为 11.0592MHz

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
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H

P5M0	DATA	0CAH
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.I,CHKRI
ANL	S2CON,#NOT 02H
CLR	BUSY

CHKRI:

JNB	ACC.0,UART2ISR_EXIT
ANL	S2CON,#NOT 01H
MOV	A,WPTR
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
MOV	T2H,#0FFH
MOV	AUXR,#14H
CLR	BUSY
MOV	WPTR,#00H
MOV	RPTR,#00H
RET	

;65536-11059200/115200/4=0FFE8H

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, #5FH
MOV      P0M0, #00H
MOV      P0M1, #00H
MOV      P1M0, #00H
MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

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

```

13.7.5 串口 3 使用定时器 2 做波特率发生器

C 语言代码

//测试工作频率为 11.0592MHz

```

#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;  
  
sfr P0M1     = 0x93;  
sfr P0M0     = 0x94;  
sfr P1M1     = 0x91;  
sfr P1M0     = 0x92;  
sfr P2M1     = 0x95;  
sfr P2M0     = 0x96;  
sfr P3M1     = 0xb1;  
sfr P3M0     = 0xb2;  
sfr P4M1     = 0xb3;  
sfr P4M0     = 0xb4;  
sfr P5M1     = 0xc9;  
sfr P5M0     = 0xca;
```

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

    Uart3Init();
    IE2 = 0x08;
    EA = 1;
    Uart3SendStr("Uart Test !r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart3Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}

```

汇编代码

; 测试工作频率为 11.0592MHz

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
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H

P5M0	DATA	0CAH
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.I,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
MOV	T2H,#0FFH
MOV	AUXR,#14H
CLR	BUSY
MOV	WPTR,#00H
MOV	RPTR,#00H
RET	

;65536-11059200/115200/4=0FFE8H

UART3_SEND:

JB	BUSY,\$
SETB	BUSY
MOV	S3BUFA
RET	

UART3_SENDSTR:

CLR	A
MOVC	A,@A+DPTR
JZ	SEND3END
LCALL	UART3_SEND
INC	DPTR
JMP	UART3_SENDSTR

SEND3END:

RET	
------------	--

MAIN:

```

MOV      SP, #5FH
MOV      P0M0, #00H
MOV      P0M1, #00H
MOV      P1M0, #00H
MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

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

```

13.7.6 串口 3 使用定时器 3 做波特率发生器

C 语言代码

```
//测试工作频率为11.0592MHz
```

```

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

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

sfr    T4T3M    = 0xd1;
sfr    T4L      = 0xd3;
sfr    T4H      = 0xd2;
sfr    T3L      = 0xd5;
sfr    T3H      = 0xd4;

```

```
sfr T2L      = 0xd7;  
sfr T2H      = 0xd6;  
sfr S3CON    = 0xac;  
sfr S3BUF    = 0xad;  
sfr IE2      = 0xaf;  
  
sfr P0MI     = 0x93;  
sfr P0M0     = 0x94;  
sfr P1M1     = 0x91;  
sfr P1M0     = 0x92;  
sfr P2M1     = 0x95;  
sfr P2M0     = 0x96;  
sfr P3M1     = 0xb1;  
sfr P3M0     = 0xb2;  
sfr P4M1     = 0xb3;  
sfr P4M0     = 0xb4;  
sfr P5M1     = 0xc9;  
sfr P5M0     = 0xca;
```

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

    Uart3Init();
    IE2 = 0x08;
    EA = 1;
    Uart3SendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart3Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}

```

汇编代码

; 测试工作频率为 11.0592MHz

<i>T4T3M</i>	<i>DATA</i>	<i>0D1H</i>
<i>T4L</i>	<i>DATA</i>	<i>0D3H</i>
<i>T4H</i>	<i>DATA</i>	<i>0D2H</i>
<i>T3L</i>	<i>DATA</i>	<i>0D5H</i>
<i>T3H</i>	<i>DATA</i>	<i>0D4H</i>
<i>T2L</i>	<i>DATA</i>	<i>0D7H</i>
<i>T2H</i>	<i>DATA</i>	<i>0D6H</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>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>

<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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>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>	

;65536-11059200/115200/4=0FFE8H

UART3_SEND:

<i>JB</i>	<i>BUSY,\$</i>
<i>SETB</i>	<i>BUSY</i>
<i>MOV</i>	<i>S3BUFA,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, #5FH</i>	
<i>MOV</i>	<i>P0M0, #00H</i>	
<i>MOV</i>	<i>P0M1, #00H</i>	
<i>MOV</i>	<i>P1M0, #00H</i>	
<i>MOV</i>	<i>P1M1, #00H</i>	
<i>MOV</i>	<i>P2M0, #00H</i>	
<i>MOV</i>	<i>P2M1, #00H</i>	
<i>MOV</i>	<i>P3M0, #00H</i>	
<i>MOV</i>	<i>P3M1, #00H</i>	
<i>MOV</i>	<i>P4M0, #00H</i>	
<i>MOV</i>	<i>P4M1, #00H</i>	
<i>MOV</i>	<i>P5M0, #00H</i>	
<i>MOV</i>	<i>P5M1, #00H</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		

13.7.7 串口 4 使用定时器 2 做波特率发生器

C 语言代码

```
//测试工作频率为11.0592MHz
```

```
#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;

sfr P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

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

    Uart4Init();
    IE2 = 0x10;
    EA = I;
    Uart4SendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart4Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}

```

汇编代码

; 测试工作频率为 11.0592MHz

<i>AUXR</i>	<i>DATA</i>	8EH
<i>T2H</i>	<i>DATA</i>	0D6H
<i>T2L</i>	<i>DATA</i>	0D7H
<i>S4CON</i>	<i>DATA</i>	84H
<i>S4BUF</i>	<i>DATA</i>	85H
<i>IE2</i>	<i>DATA</i>	0AFH
<i>BUSY</i>	<i>BIT</i>	20H.0
<i>WPTR</i>	<i>DATA</i>	21H
<i>RPTR</i>	<i>DATA</i>	22H
<i>BUFFER</i>	<i>DATA</i>	23H
		;16 bytes
<i>P0M1</i>	<i>DATA</i>	093H
<i>P0M0</i>	<i>DATA</i>	094H
<i>P1M1</i>	<i>DATA</i>	091H

<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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>
------------	--------------

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>	
<i>SEND4END:</i>		
<i>RET</i>		
<i>MAIN:</i>		
<i>MOV</i>	<i>SP, #5FH</i>	
<i>MOV</i>	<i>P0M0, #00H</i>	
<i>MOV</i>	<i>P0M1, #00H</i>	
<i>MOV</i>	<i>P1M0, #00H</i>	
<i>MOV</i>	<i>P1M1, #00H</i>	
<i>MOV</i>	<i>P2M0, #00H</i>	
<i>MOV</i>	<i>P2M1, #00H</i>	
<i>MOV</i>	<i>P3M0, #00H</i>	
<i>MOV</i>	<i>P3M1, #00H</i>	
<i>MOV</i>	<i>P4M0, #00H</i>	
<i>MOV</i>	<i>P4M1, #00H</i>	
<i>MOV</i>	<i>P5M0, #00H</i>	
<i>MOV</i>	<i>P5M1, #00H</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>	<i>'Uart Test !',0DH,0AH,00H</i>
<i>END</i>		

13.7.8 串口 4 使用定时器 4 做波特率发生器

C 语言代码

```
//测试工作频率为11.0592MHz
```

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

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

sfr T4T3M      = 0xd1;
sfr T4L        = 0xd3;
sfr T4H        = 0xd2;
sfr T3L        = 0xd5;
sfr T3H        = 0xd4;
sfr T2L        = 0xd7;
sfr T2H        = 0xd6;
sfr S4CON      = 0x84;
sfr S4BUF      = 0x85;
sfr IE2         = 0xaf;

sfr P0M1       = 0x93;
sfr P0M0       = 0x94;
sfr P1M1       = 0x91;
sfr P1M0       = 0x92;
sfr P2M1       = 0x95;
sfr P2M0       = 0x96;
sfr P3M1       = 0xb1;
sfr P3M0       = 0xb2;
sfr P4M1       = 0xb3;
sfr P4M0       = 0xb4;
sfr P5M1       = 0xc9;
sfr P5M0       = 0xca;

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 = I;  
    S4BUF = dat;  
}  
  
void Uart4SendStr(char *p)  
{  
    while (*p)  
    {  
        Uart4Send(*p++);  
    }  
}  
  
void main()  
{  
    P0M0 = 0x00;  
    P0M1 = 0x00;  
    P1M0 = 0x00;  
    P1M1 = 0x00;  
    P2M0 = 0x00;  
    P2M1 = 0x00;  
    P3M0 = 0x00;  
    P3M1 = 0x00;  
    P4M0 = 0x00;  
    P4M1 = 0x00;  
    P5M0 = 0x00;  
    P5M1 = 0x00;  
  
    Uart4Init();  
    IE2 = 0x10;  
    EA = I;  
    Uart4SendStr("Uart Test !r\n");  
  
    while (1)  
    {  
        if (rptr != wptr)  
        {  
            Uart4Send(buffer[rptr++]);  
            rptr &= 0x0f;  
        }  
    }  
}
```

汇编代码

; 测试工作频率为 11.0592MHz

T4T3M	DATA	0DIH
T4L	DATA	0D3H
T4H	DATA	0D2H
T3L	DATA	0D5H
T3H	DATA	0D4H
T2L	DATA	0D7H
T2H	DATA	0D6H
S4CON	DATA	84H
S4BUF	DATA	85H
IE2	DATA	0AFH

BUSY	BIT	20H.0
WPTR	DATA	21H
RPTR	DATA	22H
BUFFER	DATA	23H

;16 bytes

P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH

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.I,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
MOV	T4H,#0FFH
MOV	T4T3M,#0A0H
CLR	BUSY
MOV	WPTR,#00H
MOV	RPTR,#00H
RET	

;65536-11059200/115200/4=0FFE8H

UART4_SEND:

<i>JB</i>	<i>BUSY,\$</i>
<i>SETB</i>	<i>BUSY</i>
<i>MOV</i>	<i>S4BUFA,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:

	<i>RET</i>
--	------------

MAIN:

<i>MOV</i>	<i>SP, #5FH</i>
<i>MOV</i>	<i>P0M0, #00H</i>
<i>MOV</i>	<i>P0M1, #00H</i>
<i>MOV</i>	<i>P1M0, #00H</i>
<i>MOV</i>	<i>P1M1, #00H</i>
<i>MOV</i>	<i>P2M0, #00H</i>
<i>MOV</i>	<i>P2M1, #00H</i>
<i>MOV</i>	<i>P3M0, #00H</i>
<i>MOV</i>	<i>P3M1, #00H</i>
<i>MOV</i>	<i>P4M0, #00H</i>
<i>MOV</i>	<i>P4M1, #00H</i>
<i>MOV</i>	<i>P5M0, #00H</i>
<i>MOV</i>	<i>P5M1, #00H</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>

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>UART4_SEND</i>
<i>INC</i>	<i>RPTR</i>
<i>JMP</i>	<i>LOOP</i>

STRING:	DB	'Uart Test !',0DH,0AH,00H
----------------	-----------	----------------------------------

	END
--	------------

13.7.9 串口多机通讯

现参考 STC15 系列数据手册, 后续补充

STCMCU

13.7.10 串口转 LIN 总线

C 语言代码

//测试工作频率为 22.1184MHz

```
***** 功能说明 *****
本例程基于 STC8H8K64U 为主控芯片的实验箱8 进行编写测试, STC8G、STC8H 系列芯片可通用参考。
通过 UART 接口连接 LIN 收发器实现 LIN 总线信号收发测试例程。
UART1 通过串口工具连接电脑。
UART2 外接 LIN 收发器(TJA1020/I), 连接 LIN 总线。
将电脑串口发送的数据转发到 LIN 总线; 从 LIN 总线接收到的数据转发到电脑串口。
默认传输速率: 9600 波特率, 发送 LIN 数据前切换波特率, 发送13 个显性间隔信号。
下载时, 选择时钟 22.1184MHz (用户可自行修改频率)。
*****/
```

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

#define MAIN_Fosc 22118400L

typedef unsigned char u8;
typedef unsigned int u16;
typedef unsigned long u32;

sfr AUXR = 0x8E;
sfr S2CON = 0x9A;
sfr S2BUF = 0x9B;
sfr TH2 = 0xD6;
sfr TL2 = 0xD7;
sfr IE2 = 0xAF;
sfr INT_CLKO = 0x8F;
sfr P_SW1 = 0xA2;
sfr P_SW2 = 0xBA;

sfr P4 = 0xC0;
sfr P5 = 0xC8;
sfr P6 = 0xE8;
sfr P7 = 0xF8;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xB1;
sfr P3M0 = 0xB2;
sfr P4M1 = 0xB3;
sfr P4M0 = 0xB4;
sfr P5M1 = 0xC9;
sfr P5M0 = 0xCA;
sfr P6M1 = 0xCB;
sfr P6M0 = 0xCC;
sfr P7M1 = 0xE1;
sfr P7M0 = 0xE2;

sbit P00 = P0^0;
```

```

sbit P01      = P0^1;
sbit P02      = P0^2;
sbit P03      = P0^3;
sbit P04      = P0^4;
sbit P05      = P0^5;
sbit P06      = P0^6;
sbit P07      = P0^7;
sbit P10      = P1^0;
sbit P11      = P1^1;
sbit P12      = P1^2;
sbit P13      = P1^3;
sbit P14      = P1^4;
sbit P15      = P1^5;
sbit P16      = P1^6;
sbit P17      = P1^7;
sbit P20      = P2^0;
sbit P21      = P2^1;
sbit P22      = P2^2;
sbit P23      = P2^3;
sbit P24      = P2^4;
sbit P25      = P2^5;
sbit P26      = P2^6;
sbit P27      = P2^7;
sbit P30      = P3^0;
sbit P31      = P3^1;
sbit P32      = P3^2;
sbit P33      = P3^3;
sbit P34      = P3^4;
sbit P35      = P3^5;
sbit P36      = P3^6;
sbit P37      = P3^7;
sbit P40      = P4^0;
sbit P41      = P4^1;
sbit P42      = P4^2;
sbit P43      = P4^3;
sbit P44      = P4^4;
sbit P45      = P4^5;
sbit P46      = P4^6;
sbit P47      = P4^7;
sbit P50      = P5^0;
sbit P51      = P5^1;
sbit P52      = P5^2;
sbit P53      = P5^3;
sbit P54      = P5^4;
sbit P55      = P5^5;
sbit P56      = P5^6;
sbit P57      = P5^7;

sbit SLP_N    = P2^4; //0: Sleep

```

***** 用户定义宏 *****

```

#define Baudrate1      (65536UL - (MAIN_Fosc / 4) / 9600UL)
#define Baudrate2      (65536UL - (MAIN_Fosc / 4) / 9600UL)

#define Baudrate_Break (65536UL - (MAIN_Fosc / 4) / 6647UL) //发送显性间隔信号波特率

#define UART1_BUF_LENGTH 32
#define UART2_BUF_LENGTH 32

```

```

#define LIN_ID      0x31

u8 TX1_Cnt;                      //发送计数
u8 RX1_Cnt;                      //接收计数
u8 TX2_Cnt;                      //发送计数
u8 RX2_Cnt;                      //接收计数
bit B_TX1_Busy;                  //发送忙标志
bit B_TX2_Busy;                  //发送忙标志
u8 RX1_TimeOut;                  //接收缓冲
u8 RX2_TimeOut;                  //接收缓冲

u8 xdata RX1_Buffer[UART1_BUF_LENGTH]; //接收缓冲
u8 xdata RX2_Buffer[UART2_BUF_LENGTH]; //接收缓冲

void UART1_config(u8 brt);
void UART2_config(u8 brt);
void PrintString1(u8 *puts);
void delay_ms(u8 ms);
void UART1_TxByte(u8 dat);
void UART2_TxByte(u8 dat);
void Lin_Send(u8 *puts);
void SetTimer2Baudraye(u16 dat);

//=====
// 函数: void main(void)
// 描述: 主函数。
// 参数: none.
// 返回: none.
// 版本: VER1.0
// 日期: 2014-11-28
// 备注:
//=====

void main(void)
{
    u8 i;

    P0M1 = 0; P0M0 = 0;                      //设置为准双向口
    P1M1 = 0; P1M0 = 0;                      //设置为准双向口
    P2M1 = 0; P2M0 = 0;                      //设置为准双向口
    P3M1 = 0; P3M0 = 0;                      //设置为准双向口
    P4M1 = 0; P4M0 = 0;                      //设置为准双向口
    P5M1 = 0; P5M0 = 0;                      //设置为准双向口
    P6M1 = 0; P6M0 = 0;                      //设置为准双向口
    P7M1 = 0; P7M0 = 0;                      //设置为准双向口

    UART1_config(1);
    UART2_config(2);
    EA = I;                                //允许全局中断
    SLP_N = I;

    PrintString1("STC8H8K64U UART1 Test Programme!\r\n"); //UART1 发送一个字符串

    while (1)
    {
        delay_ms(1);
        if(RX1_TimeOut > 0)
        {
            if(--RX1_TimeOut == 0)              //超时,则串口接收结束

```

```

    {
        if(RX1_Cnt > 0)
        {
            Lin_Send(RX1_Buffer);           //将UART1 收到的数据发送到LIN 总线上
        }
        RX1_Cnt = 0;
    }

}

if(RX2_TimeOut > 0)
{
    if(--RX2_TimeOut == 0)           //超时,则串口接收结束
    {
        if(RX2_Cnt > 0)
        {
            for (i=0; i < RX2_Cnt; i++)   //遇到停止符0 结束
            {
                UART1_TxByte(RX2_Buffer[i]); //从LIN 总线收到的数据发送到UART1
            }
        }
        RX2_Cnt = 0;
    }
}
}

//=====
// 函数: void delay_ms(unsigned char ms)
// 描述: 延时函数。
// 参数: ms,要延时的ms 数, 这里只支持1~255ms. 自动适应主时钟。
// 返回: none.
// 版本: VER1.0
// 日期: 2013-4-1
// 备注:
//=====

void delay_ms(u8 ms)
{
    u16 i;
    do{
        i = MAIN_Fosc / 10000;
        while(--i);                  //10T per loop
    }while(--ms);
}

//=====
// 函数: u8          Lin_CheckPID(u8    id)
// 描述: ID 码加上校验符, 转成 PID 码。
// 参数: ID 码.
// 返回: PID 码.
// 版本: VER1.0
// 日期: 2020-12-2
// 备注:
//=====

u8 Lin_CheckPID(u8 id)
{
    u8 returnpid ;
    u8 P0 ;
    u8 P1 ;
}

```

```
P0 = (((id)^((id>>1)^((id>>2)^((id>>4)&0x01)))<<6;
P1 = (((~((id>>1)^((id>>3)^((id>>4)^((id>>5))))&0x01))<<7;

return pid = id|P0|P1 ;

}

//=====
// 函数: u8 LINCalcChecksum(u8 *dat)
// 描述: 计算校验码。
// 参数: 数据场传输的数据。
// 返回: 校验码。
// 版本: VER1.0
// 日期: 2020-12-2
// 备注:
//=====

static u8 LINCalcChecksum(u8 *dat)
{
    u16 sum = 0;
    u8 i;

    for(I = 0; i < 8; i++)
    {
        sum += dat[i];
        if(sum & 0xFF00)
        {
            sum = (sum & 0x00FF) + 1;
        }
    }
    sum ^= 0x00FF;
    return (u8)sum;
}

//=====
// 函数: void Lin_SendBreak(void)
// 描述: 发送显性间隔信号。
// 参数: none.
// 返回: none.
// 版本: VER1.0
// 日期: 2020-12-2
// 备注:
//=====

void Lin_SendBreak(void)
{
    SetTimer2Baudrate(Baudrate_Break);
    UART2_TxByte(0);
    SetTimer2Baudrate(Baudrate2);
}

//=====
// 函数: void Lin_Send(u8 *puts)
// 描述: 发送LIN总线报文。
// 参数: 待发送的数据场内容。
// 返回: none.
// 版本: VER1.0
// 日期: 2020-12-2
// 备注:
//=====
```

```
void Lin_Send(u8 *puts)
{
    u8 i;

    Lin_SendBreak();                                //Break
    UART2_TxByte(0x55);                            //SYNC
    UART2_TxByte(Lin_CheckPID(LIN_ID));            //LIN ID
    for(i=0;i<8;i++)
    {
        UART2_TxByte(puts[i]);
    }
    UART2_TxByte(LINCalcChecksum(puts));
}

//=====
// 函数: void UART1_TxByte(u8 dat)
// 描述: 发送一个字节.
// 参数: 无
// 返回: 无
// 版本: V1.0, 2014-6-30
//=====
void UART1_TxByte(u8 dat)
{
    SBUF = dat;
    B_TX1_Busy = 1;
    while(B_TX1_Busy);
}

//=====
// 函数: void UART2_TxByte(u8 dat)
// 描述: 发送一个字节.
// 参数: 无
// 返回: 无
// 版本: V1.0, 2014-6-30
//=====
void UART2_TxByte(u8 dat)
{
    S2BUF = dat;
    B_TX2_Busy = 1;
    while(B_TX2_Busy);
}

//=====
// 函数: void PrintString1(u8 *puts)
// 描述: 串口1发送字符串函数。
// 参数: puts:          字符串指针.
// 返回: none.
// 版本: VER1.0
// 日期: 2014-11-28
// 备注:
//=====
void PrintString1(u8 *puts)
{
    for ( ; *puts != 0; puts++)                  //遇到停止符0 结束
    {
        SBUF = *puts;
        B_TX1_Busy = 1;
        while(B_TX1_Busy);
    }
}
```

```
}

//=====
// 函数: void PrintString2(u8 *puts)
// 描述: 串口2 发送字符串函数。
// 参数: puts:          字符串指针。
// 返回: none.
// 版本: VER1.0
// 日期: 2014-11-28
// 备注:
//=====

//void PrintString2(u8 *puts)
//{
//    for (; *puts != 0; puts++)           //遇到停止符0 结束
//    {
//        S2BUF = *puts;
//        B_TX2_Busy = 1;
//        while(B_TX2_Busy);
//    }
//}

//=====
// 函数: SetTimer2Baudraye(u16 dat)
// 描述: 设置 Timer2 做波特率发生器。
// 参数: dat: Timer2 的重装值。
// 返回: none.
// 版本: VER1.0
// 日期: 2014-11-28
// 备注:
//=====

void SetTimer2Baudraye(u16 dat)
{
    AUXR &= ~(1<<4);                  //Timer stop
    AUXR &= ~(1<<3);                  //Timer2 set As Timer
    AUXR |= (1<<2);                   //Timer2 set as IT mode
    TH2  = dat / 256;
    TL2 = dat % 256;
    IE2 &= ~(1<<2);                  //禁止中断
    AUXR |= (1<<4);                   //Timer run enable
}

//=====
// 函数: void UART1_config(u8 brt)
// 描述: UART1 初始化函数。
// 参数: brt: 选择波特率, 2: 使用 Timer2 做波特率, 其它值: 使用 Timer1 做波特率
// 返回: none.
// 版本: VER1.0
// 日期: 2014-11-28
// 备注:
//=====

void UART1_config(u8 brt)
{
    /***** 波特率使用定时器2 *****/
    if(brt == 2)                       //SI BRT Use Timer2;
    {
        AUXR |= 0x01;
        SetTimer2Baudraye(Baudrate1);
    }
}
```

```

***** 波特率使用定时器1 *****
else
{
    TR1 = 0;
    AUXR &= ~0x01;
    AUXR |= (1<<6);
    TMOD &= ~(1<<6);
    TMOD &= ~0x30;
    TH1 = (u8)(Baudrate1 / 256);
    TL1 = (u8)(Baudrate1 % 256);
    ET1 = 0;                                //禁止中断
    INT_CLKO &= ~0x02;                      //不输出时钟
    TR1 = 1;
}
*****



SCON = (SCON & 0x3f) | 0x40;           //UART1 模式: 0x00: 同步移位输出,
//                                         0x40: 8 位数据, 可变波特率
//                                         0x80: 9 位数据, 固定波特率
//                                         0xc0: 9 位数据, 可变波特率
// PS    = 1;                            //高优先级中断
// ES    = 1;                            //允许中断
// REN   = 1;                            //允许接收
// P_SW1 &= 0x3f;                      //UART1switch to: 0x00: P3.0 P3.1,
//                                         0x40: P3.6 P3.7,
//                                         0x80: P1.6 P1.7,
//                                         0xC0: P4.3 P4.4
// P_SW1 |= 0x80;

B_TX1_Busy = 0;
TX1_Cnt = 0;
RX1_Cnt = 0;
}

=====

// 函数: void UART2_config(u8 brt)
// 描述: UART2 初始化函数。
// 参数: brt: 选择波特率, 2: 使用Timer2 做波特率, 其它值: 无效
// 返回: none.
// 版本: VER1.0
// 日期: 2014-11-28
// 备注:
=====

void UART2_config(u8 brt)
{
    if(brt == 2)
    {
        SetTimer2Baudrate(Baudrate2);

        S2CON &= ~(1<<7);                  //8 位数据, 1 位起始位, 1 位停止位, 无校验
        IE2 |= 1;                           //允许中断
        S2CON |= (1<<4);                 //允许接收
        P_SW2 &= ~0x01;
        P_SW2 |= 1;                        //UART2 switch to: 0: P1.0/P1.1, 1: P4.6/P4.7

        B_TX2_Busy = 0;
        TX2_Cnt = 0;
        RX2_Cnt = 0;
    }
}

```

```
}

//=====
// 函数: void UART1_int (void) interrupt UART1_VECTOR
// 描述: UART1 中断函数。
// 参数: nine.
// 返回: none.
// 版本: VER1.0
// 日期: 2014-11-28
// 备注:
//=====

void UART1_int (void) interrupt 4
{
    if(RI)
    {
        RI = 0;
        if(RX1_Cnt >= UART1_BUF_LENGTH) RX1_Cnt = 0;
        RX1_Buffer[RX1_Cnt] = SBUF;
        RX1_Cnt++;
        RX1_TimeOut = 5;
    }

    if(TI)
    {
        TI = 0;
        B_TX1_Busy = 0;
    }
}

//=====
// 函数: void UART2_int (void) interrupt UART2_VECTOR
// 描述: UART2 中断函数。
// 参数: nine.
// 返回: none.
// 版本: VER1.0
// 日期: 2014-11-28
// 备注:
//=====

void UART2_int (void) interrupt 8
{
    if((S2CON & 1) != 0)
    {
        S2CON &= ~1;                                //Clear Rx flag
        if(RX2_Cnt >= UART2_BUF_LENGTH) RX2_Cnt = 0;
        RX2_Buffer[RX2_Cnt] = S2BUF;
        RX2_Cnt++;
        RX2_TimeOut = 5;
    }

    if((S2CON & 2) != 0)
    {
        S2CON &= ~2;                                //Clear Tx flag
        B_TX2_Busy = 0;
    }
}
```

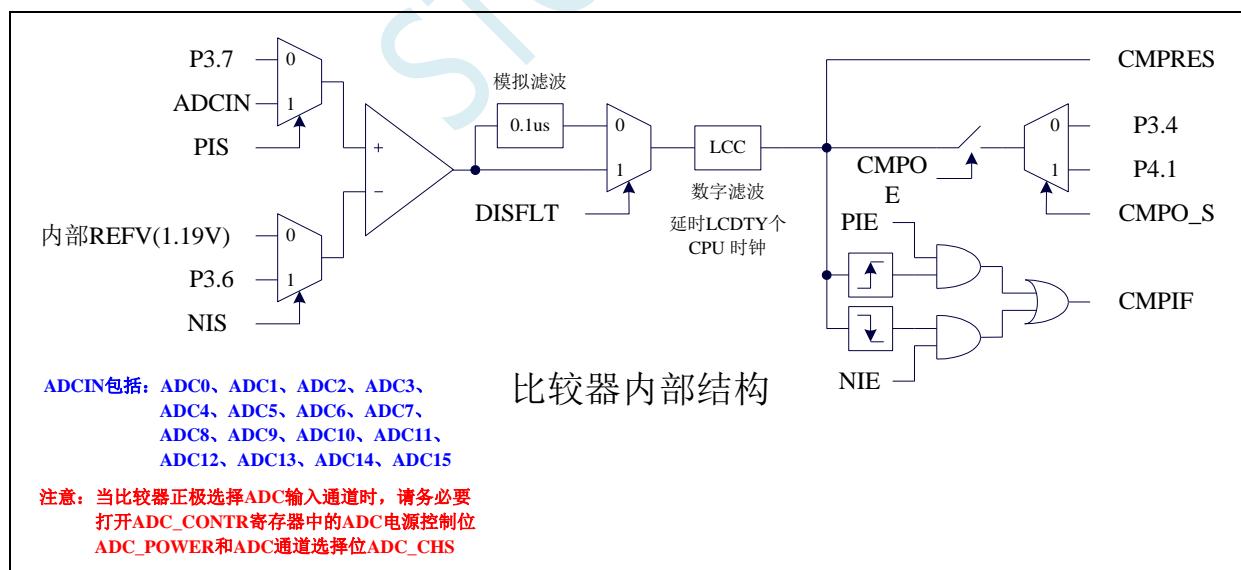
14 比较器, 掉电检测, 内部 1.19V 参考信号源

产品线	比较器
STC8G1K08 系列	●
STC8G1K08-8Pin 系列	
STC8G1K08A 系列	
STC8G2K64S4 系列	●
STC8G2K64S2 系列	●
STC8G1K08T 系列	●
STC15H2K64S4 系列	●

STC8G 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）。[通过多路选择器和分时复用可实现多个比较器的应用](#)

比较器内部有可程序控制的两级滤波：模拟滤波和数字滤波。模拟滤波可以过滤掉比较输入信号中的毛刺信号，数字滤波可以等待输入信号更加稳定后再进行比较。比较结果可直接通过读取内部寄存器位获得，也可将比较器结果正向或反向输出到外部端口。将比较结果输出到外部端口可用作外部事件的触发信号和反馈信号，可扩大比较的应用范围。

14.1 比较器内部结构图



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	CMPPEN	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

14.2.1 比较器控制寄存器 1 (CMPCR1)

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

CMPPEN: 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。
- (注意 1: 当比较器正极选择 ADC 输入通道时, 请务必要打开 ADC_CONTR 寄存器中的 ADC 电源控制位 ADC_POWER 和 ADC 通道选择位 ADC_CHS)

(注意 2: 当需要使用比较器中断唤醒掉电模式/时钟停振模式时, 比较器正极必须选择 P3.7, 不能使用 ADC 输入通道)

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 (芯片在出厂时, 内部参考电压调整为 1.19V)。
- 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。

14.2.2 比较器控制寄存器 2 (CMPCR2)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR2	E7H	INVCMPO	DISFLT					LCDTY[5:0]	

INVCMPO: 比较器结果输出控制

0: 比较器结果正向输出。若 CMPRES 为 0, 则 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。

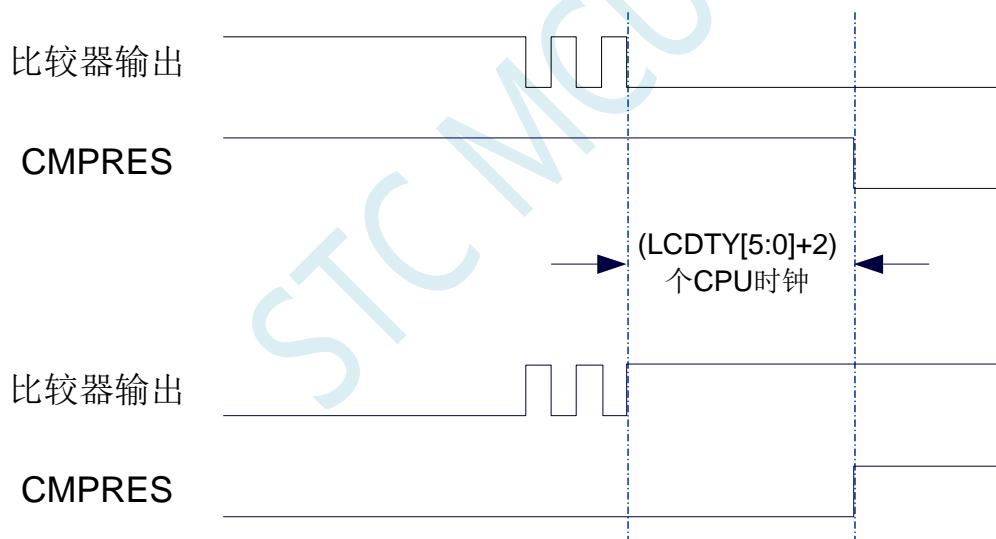
LCDTY[5:0]: 数字滤波功能控制

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。

注意: 当使能数字滤波功能后, 芯片内部实际的等待时钟需额外增加两个状态机切换时间, 即若

LCDTY 设置为 0 时, 为关闭数字滤波功能; 若 LCDTY 设置为非 0 值 n (n=1~63) 时, 则实际
的数字滤波时间为 (n+2) 个系统时钟



14.3 范例程序

14.3.1 比较器的使用（中断方式）

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr CMPCRI      = 0xe6;
sfr CMPCR2      = 0xe7;

sfr P0M1         = 0x93;
sfr P0M0         = 0x94;
sfr P1M1         = 0x91;
sfr P1M0         = 0x92;
sfr P2M1         = 0x95;
sfr P2M0         = 0x96;
sfr P3M1         = 0xb1;
sfr P3M0         = 0xb2;
sfr P4M1         = 0xb3;
sfr P4M0         = 0xb4;
sfr P5M1         = 0xc9;
sfr P5M0         = 0xca;

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

void CMP_Isr() interrupt 21
{
    CMPCRI &= ~0x40;                                //清中断标志
    if (CMPCRI & 0x01)
    {
        P10 = !P10;                                 //上升沿中断测试端口
    }
    else
    {
        P11 = !P11;                                 //下降沿中断测试端口
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
}
```

```

P5M1 = 0x00;

CMPCR2 = 0x00;
CMPCR2 &= ~0x80;                                // 比较器正向输出
// CMPCR2 |= 0x80;                                // 比较器反向输出
CMPCR2 &= ~0x40;                                // 使能 0.1us 滤波
// CMPCR2 |= 0x40;                                // 禁止 0.1us 滤波
// CMPCR2 &= ~0x3f;                                // 比较器结果直接输出
CMPCR2 |= 0x10;                                // 比较器结果经过 16 个去抖时钟后输出

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 1.19V 参考信号源为 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

EA = I;

while (I);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

CMPCRI	DATA	0E6H
CMPCR2	DATA	0E7H
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH
ORG		0000H
LJMP		MAIN
ORG		00ABH
LJMP		CMPISR
ORG		0100H
CMPISR:		
PUSH		ACC
ANL		CMPCR1,#NOT 40H ; 清中断标志
MOV		A,CMPCR1
JB		ACC.0,RSING

FALLING:

<i>CPL</i>	<i>PI.0</i>	;下降沿中断测试端口
<i>POP</i>	<i>ACC</i>	
<i>RETI</i>		

RSING:

<i>CPL</i>	<i>PI.1</i>	;上升沿中断测试端口
<i>POP</i>	<i>ACC</i>	
<i>RETI</i>		

MAIN:

```

MOV      SP, #5FH
MOV      P0M0, #00H
MOV      P0M1, #00H
MOV      P1M0, #00H
MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

MOV      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      ;使能 0.1us 滤波
; ORL      CMPCR2,#40H          ;禁止 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 1.19V 参考信号源为 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
SETB    EA

JMP     $

END

```

14.3.2 比较器的使用（查询方式）

C 语言代码

//测试工作频率为 11.0592MHz

```

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

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

sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P4M1 = 0xb3;
sfr P4M0 = 0xb4;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;

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

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CMPCR2 = 0x00;
    CMPCR2 &= ~0x80; // 比较器正向输出
    // CMPCR2 |= 0x80; // 比较器反向输出
    CMPCR2 &= ~0x40; // 使能 0.1us 滤波
    // CMPCR2 |= 0x40; // 禁止 0.1us 滤波
    CMPCR2 &= ~0x3f; // 比较器结果直接输出
    // CMPCR2 |= 0x10; // 比较器结果经过 16 个去抖时钟后输出

    CMPCR1 = 0x00;
    CMPCR1 |= 0x30; // 使能比较器边沿中断
    // CMPCR1 &= ~0x20; // 禁止比较器上升沿中断
    // CMPCR1 |= 0x20; // 使能比较器上升沿中断
    // CMPCR1 &= ~0x10; // 禁止比较器下降沿中断
    // CMPCR1 |= 0x10; // 使能比较器下降沿中断
    CMPCR1 &= ~0x08; // P3.7 为 CMP+ 输入脚
    // CMPCR1 |= 0x08; // ADC 输入脚为 CMP+ 输入脚
    CMPCR1 &= ~0x04; // 内部 1.19V 参考信号源为 CMP- 输入脚
    CMPCR1 |= 0x04; // P3.6 为 CMP- 输入脚
    // CMPCR1 &= ~0x02; // 禁止比较器输出
    CMPCR1 |= 0x02; // 使能比较器输出
    CMPCR1 |= 0x80; // 使能比较器模块
}

while (1)

```

```

{
    P10 = CMPCR1 & 0x01;           //读取比较器比较结果
}

```

汇编代码

; 测试工作频率为 11.0592MHz

```

CMPCR1      DATA      0E6H
CMPCR2      DATA      0E7H

P0M1         DATA      093H
P0M0         DATA      094H
P1M1         DATA      091H
P1M0         DATA      092H
P2M1         DATA      095H
P2M0         DATA      096H
P3M1         DATA      0B1H
P3M0         DATA      0B2H
P4M1         DATA      0B3H
P4M0         DATA      0B4H
P5M1         DATA      0C9H
P5M0         DATA      0CAH

        ORG      0000H
        LJMP     MAIN

        ORG      0100H

MAIN:
        MOV      SP, #5FH
        MOV      P0M0, #00H
        MOV      P0M1, #00H
        MOV      P1M0, #00H
        MOV      P1M1, #00H
        MOV      P2M0, #00H
        MOV      P2M1, #00H
        MOV      P3M0, #00H
        MOV      P3M1, #00H
        MOV      P4M0, #00H
        MOV      P4M1, #00H
        MOV      P5M0, #00H
        MOV      P5M1, #00H

        MOV      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      ; 使能 0.1us 滤波
        ORL      CMPCR2,#40H       ; 禁止 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      CMPCR1,#00H
        ORL      CMPCR1,#30H       ; 使能比较器边沿中断
        ANL      CMPCR1,#NOT 20H      ; 禁止比较器上升沿中断
        ORL      CMPCR1,#20H       ; 使能比较器上升沿中断
        ANL      CMPCR1,#NOT 10H      ; 禁止比较器下降沿中断
        ORL      CMPCR1,#10H       ; 使能比较器下降沿中断
        ANL      CMPCR1,#NOT 08H      ; P3.7 为 CMP+ 输入脚
        ORL      CMPCR1,#08H       ; ADC 输入脚为 CMP+ 输入脚

```

```

;          ANL      CMPCR1,#NOT 04H      ;内部1.19V 参考信号源为 CMP-输入脚
;          ORL      CMPCR1,#04H       ;P3.6 为 CMP-输入脚
;          ANL      CMPCR1,#NOT 02H      ;禁止比较器输出
;          ORL      CMPCR1,#02H       ;使能比较器输出
;          ORL      CMPCR1,#80H       ;使能比较器模块

LOOP:
    MOV      A,CMPCR1
    MOV      C,ACC.0
    MOV      P1.0,C
    JMP      LOOP             ;读取比较器比较结果

END

```

14.3.3 比较器的多路复用应用（比较器+ADC 输入通道）

由于比较器的正极可以选择 ADC 的模拟输入通道，因此可以通过多路选择器和分时复用可实现多个比较器的应用。

注意：当比较器正极选择 ADC 输入通道时，请务必要打开 ADC_CONTR 寄存器中的 ADC 电源控制位 ADC_POWER 和 ADC 通道选择位 ADC_CHS

C 语言代码

```

//测试工作频率为11.0592MHz

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

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

sfr    ADC_CONTR   =  0xbc;

sfr    P1M1         =  0x91;
sfr    P1M0         =  0x92;
sfr    P0M1         =  0x93;
sfr    P0M0         =  0x94;
sfr    P2M1         =  0x95;
sfr    P2M0         =  0x96;
sfr    P3M1         =  0xb1;
sfr    P3M0         =  0xb2;
sfr    P4M1         =  0xb3;
sfr    P4M0         =  0xb4;
sfr    P5M1         =  0xc9;
sfr    P5M0         =  0xca;

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

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
}
```

```

P2M1 = 0x00;
P3M0 = 0x00;
P3M1 = 0x00;
P4M0 = 0x00;
P4M1 = 0x00;
P5M0 = 0x00;
P5M1 = 0x00;

P1M0 &= 0xfe;                                //设置 P1.0 为输入口
P1M1 |= 0x01;                                 //使能 ADC 模块并选择 P1.0 为 ADC 输入脚

CMPCR2 = 0x00;
CMPCRI = 0x00;

CMPCRI |= 0x08;                                //ADC 输入脚为 CMP+ 输入脚
CMPCRI |= 0x04;                                //P3.6 为 CMP- 输入脚
CMPCRI |= 0x02;                                //使能比较器输出
CMPCRI |= 0x80;                                //使能比较器模块

while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

CMPCRI	DATA	0E6H
CMPCR2	DATA	0E7H
ADC_CONTR	DATA	0BCH
P1M1	DATA	091H
P1M0	DATA	092H
P0M1	DATA	093H
P0M0	DATA	094H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH
ORG	0000H	
LJMP	MAIN	
ORG	0100H	
MAIN:		
MOV	SP, #5FH	
MOV	P0M0, #00H	
MOV	P0M1, #00H	
MOV	P1M0, #00H	
MOV	P1M1, #00H	
MOV	P2M0, #00H	
MOV	P2M1, #00H	
MOV	P3M0, #00H	
MOV	P3M1, #00H	
MOV	P4M0, #00H	

```

MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

ANL      P1M0,#0FEH          ;设置 P1.0 为输入口
ORL      P1M1,#01H
MOV      ADC_CONTR,#80H       ;使能 ADC 模块并选择 P1.0 为 ADC 输入脚

MOV      CMPCR2,#00H
MOV      CMPCR1,#00H

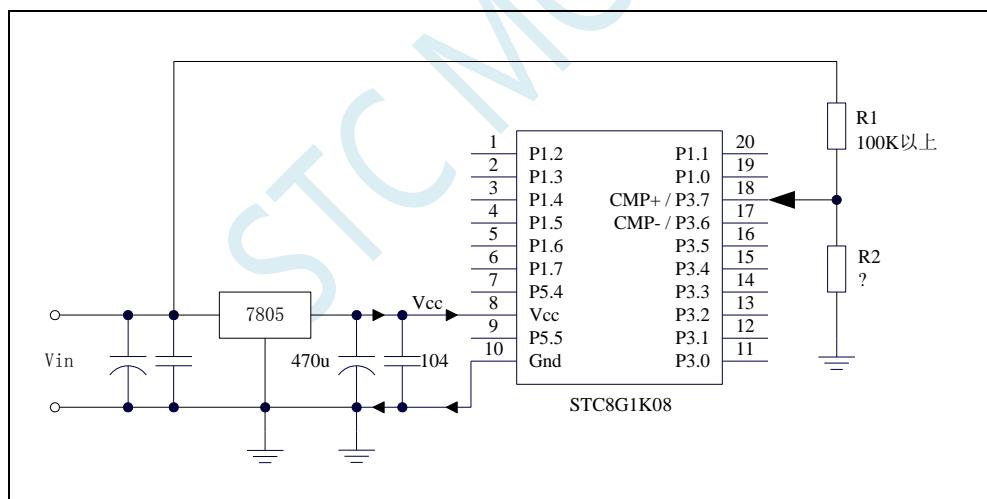
ORL      CMPCR1,#08H          ;ADC 输入脚为 CMP+ 输入脚
ORL      CMPCR1,#04H          ;P3.6 为 CMP- 输入脚
ORL      CMPCR1,#02H          ;使能比较器输出
ORL      CMPCR1,#80H          ;使能比较器模块

LOOP:
JMP      LOOP

END

```

14.3.4 比较器作外部掉电检测（掉电过程中应及时保存用户数据到 EEPROM 中）



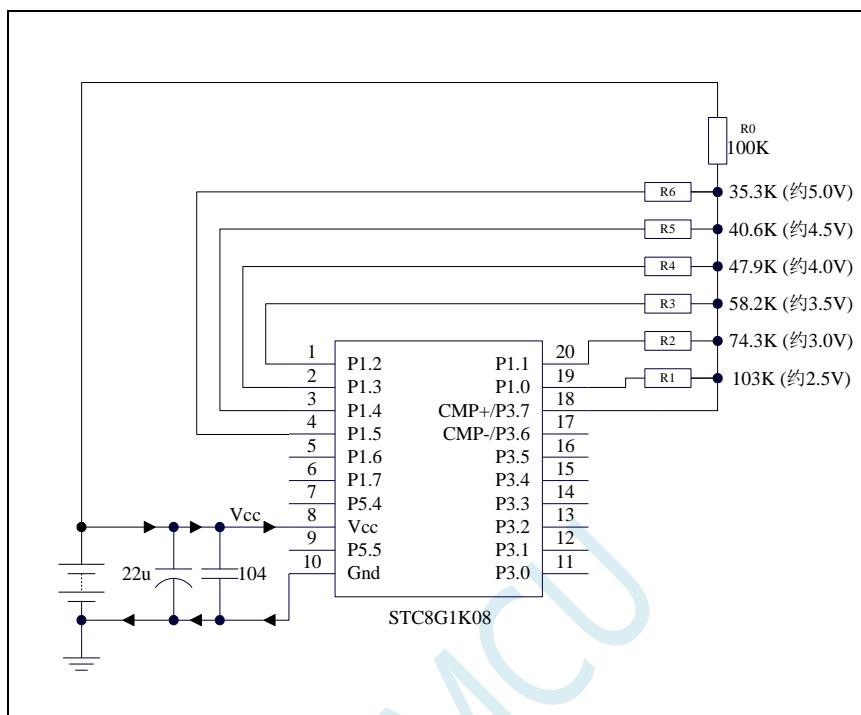
上图中电阻 R1 和 R2 对稳压块 7805 的前端电压进行分压，分压后的电压作为比较器 CMP+的外部输入与内部 1.19V 参考信号源进行比较。

一般当交流电在 220V 时，稳压块 7805 前端的直流电压为 11V，但当交流电压降到 160V 时，稳压块 7805 前端的直流电压为 8.5V。当稳压块 7805 前端的直流电压低于或等于 8.5V 时，该前端输入的直流电压被电阻 R1 和 R2 分压到比较器正极输入端 CMP+，CMP+端输入电压低于内部 1.19V 参考信号源，此时可产生比较器中断，这样在掉电检测时就有充足的时间将数据保存到 EEPROM 中。当稳压块 7805 前端的直流电压高于 8.5V 时，该前端输入的直流电压被电阻 R1 和 R2 分压到比较器正极输入端 CMP+，CMP+端输入电压高于内部 1.19V 参考信号源，此时 CPU 可继续正常工作。

内部 1.19V 参考信号源即为内部 BandGap 经过 OP 后的电压 REFV（**芯片在出厂时，内部参考电压调整为 1.19V**）。具体的数值要通过读取内部 1.19V 参考信号源在内部 RAM 区或者 Flash 程序存储器（ROM）区所占用的地址的值获得。对于 STC8 系列，内部 1.19V 参考信号源值在 RAM 和 Flash 程序

存储器（ROM）中的存储地址请参考“[存储器中的特殊参数](#)”章节

14.3.5 比较器检测工作电压（电池电压）



上图中，利用电阻分压的原理可以近似的测量出 MCU 的工作电压（选通的通道，MCU 的 I/O 口输出低电平，端口电压值接近 Gnd，未选通的通道，MCU 的 I/O 口输出开漏模式的高，不影响其他通道）。

比较器的负端选择内部 1.19V 参考信号源，正端选择通过电阻分压后输入到 CMP+管脚的电压值。

初始化时 P1.5~P1.0 口均设置为开漏模式，并输出高。首先 P1.0 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 P1.0 口输出高，P1.1 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 P1.1 口输出高，P1.2 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 P1.2 口输出高，P1.3 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 P1.3 口输出高，P1.4 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 P1.4 口输出高，P1.5 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。

C 语言代码

//测试工作频率为11.0592MHz

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

```

sfr CMPCRI      = 0xe6;
sfr CMPCTR2     = 0xe7;

sfr P0M1        = 0x93;
sfr P0M0        = 0x94;
sfr P1M1        = 0x91;
sfr P1M0        = 0x92;
sfr P2M1        = 0x95;
sfr P2M0        = 0x96;
sfr P3M1        = 0xb1;
sfr P3M0        = 0xb2;
sfr P4M1        = 0xb3;
sfr P4M0        = 0xb4;
sfr P5M1        = 0xc9;
sfr P5M0        = 0xca;

void delay()
{
    char i;
    for (i=0; i<20; i++);
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    unsigned char v;

    P1M0 = 0x3f;                                //P1.5~P1.0 初始化为开漏模式
    P1M1 = 0x3f;
    P1 = 0xff;

    CMPCRI2 = 0x10;                             //比较器结果经过16个去抖时钟后输出
    CMPCRI = 0x00;
    CMPCRI &= ~0x08;                            //P3.7 为CMP+输入脚
    CMPCRI &= ~0x04;                            //内部1.19V 参考信号源为CMP-输入脚
    CMPCRI &= ~0x02;                            //禁止比较器输出
    CMPCRI |= 0x80;                            //使能比较器模块

    while (1)
    {
        v = 0x00;                                //电压<2.5V
        P1 = 0xfe;                               //P1.0 输出0
        delay();
        if (!(CMPCRI & 0x01)) goto ShowVol;
        v = 0x01;                                //电压>2.5V
        P1 = 0xfd;                               //P1.1 输出0
    }
}

```

```

delay();
if (!(CMPCRI & 0x01)) goto ShowVol;
v = 0x03;                                //电压>3.0V
P1 = 0xfb;                                //P1.2 输出0
delay();
if (!(CMPCRI & 0x01)) goto ShowVol;
v = 0x07;                                //电压>3.5V
P1 = 0xf7;                                //P1.3 输出0
delay();
if (!(CMPCRI & 0x01)) goto ShowVol;
v = 0x0f;                                //电压>4.0V
P1 = 0xef;                                //P1.4 输出0
delay();
if (!(CMPCRI & 0x01)) goto ShowVol;
v = 0x1f;                                //电压>4.5V
P1 = 0xdf;                                //P1.5 输出0
delay();
if (!(CMPCRI & 0x01)) goto ShowVol;
v = 0x3f;                                //电压>5.0V

ShowVol:
P1 = 0xff;
P0 = ~v;
}
}

```

汇编代码

; 测试工作频率为 11.0592MHz

<i>CMPCRI</i>	<i>DATA</i>	<i>0E6H</i>
<i>CMPCR2</i>	<i>DATA</i>	<i>0E7H</i>
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</i>
<i>ORG</i>	<i>DATA</i>	<i>0000H</i>
<i>LJMP</i>	<i>DATA</i>	<i>MAIN</i>
<i>ORG</i>	<i>DATA</i>	<i>0100H</i>
<i>MAIN:</i>		
<i>MOV</i>	<i>DATA</i>	<i>SP, #5FH</i>
<i>MOV</i>	<i>DATA</i>	<i>P0M0, #00H</i>
<i>MOV</i>	<i>DATA</i>	<i>P0M1, #00H</i>
<i>MOV</i>	<i>DATA</i>	<i>P1M0, #00H</i>
<i>MOV</i>	<i>DATA</i>	<i>P1M1, #00H</i>
<i>MOV</i>	<i>DATA</i>	<i>P2M0, #00H</i>
<i>MOV</i>	<i>DATA</i>	<i>P2M1, #00H</i>
<i>MOV</i>	<i>DATA</i>	<i>P3M0, #00H</i>
<i>MOV</i>	<i>DATA</i>	<i>P3M1, #00H</i>

```

MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

MOV      P1M0,#0011111B ;P1.5~P1.0 初始化为开漏模式
MOV      P1M1,#0011111B
MOV      P1,#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 1.19V 参考信号源为 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,#00000000B ;电压<2.5V
MOV      P1,#1111110B ;P1.0 输出 0
CALL    DELAY
MOV      A,CMPCR1
JNB    ACC.0,SKIP
MOV      R0,#00000001B ;电压>2.5V
MOV      P1,#11111101B ;P1.1 输出 0
CALL    DELAY
MOV      A,CMPCR1
JNB    ACC.0,SKIP
MOV      R0,#000000011B ;电压>3.0V
MOV      P1,#111111011B ;P1.2 输出 0
CALL    DELAY
MOV      A,CMPCR1
JNB    ACC.0,SKIP
MOV      R0,#00000111B ;电压>3.5V
MOV      P1,#11110111B ;P1.3 输出 0
CALL    DELAY
MOV      A,CMPCR1
JNB    ACC.0,SKIP
MOV      R0,#00001111B ;电压>4.0V
MOV      P1,#11101111B ;P1.4 输出 0
CALL    DELAY
MOV      A,CMPCR1
JNB    ACC.0,SKIP
MOV      R0,#00011111B ;电压>4.5V
MOV      P1,#11011111B ;P1.5 输出 0
CALL    DELAY
MOV      A,CMPCR1
JNB    ACC.0,SKIP
MOV      R0,#00111111B ;电压>5.0V

SKIP:
MOV      P1,#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

```

STCMCU

15 IAP/EEPROM/DATA-FLASH

STC8G 系列单片机内部集成了大容量的 EEPROM。利用 ISP/IAP 技术可将内部 Data Flash 当 EEPROM，擦写次数在 10 万次以上。EEPROM 可分为若干个扇区，每个扇区包含 512 字节。

注意: EEPROM 的写操作只能将字节中的 1 写为 0，当需要将字节中的 0 写为 1，则必须执行扇区擦除操作。EEPROM 的读/写操作是以 1 字节为单位进行，而 EEPROM 擦除操作是以 1 扇区（512 字节）为单位进行，在执行擦除操作时，如果目标扇区中有需要保留的数据，则必须预先将这些数据读取到 RAM 中暂存，待擦除完成后再将保存的数据和需要更新的数据一起再写回 EEPROM/DATA-FLASH。

所以在使用时，建议同一次修改的数据放在同一个扇区，不是同一次修改的数据放在不同的扇区，不一定要用满。数据存储器的擦除操作是按扇区进行的（每扇区 512 字节）。

EEPROM 可用于保存一些需要在应用过程中修改并且掉电不丢失的参数数据。在用户程序中，可以对 EEPROM 进行字节读/字节编程/扇区擦除操作。在工作电压偏低时，建议不要进行 EEPROM 操作，以免发送数据丢失的情况。

15.1 EEPROM 操作时间

- 读取 1 字节：4 个系统时钟（使用 MOVC 指令读取更方便快捷）
- 编程 1 字节：约 30~40us（实际的编程时间为 6~7.5us，但还需要加上状态转换时间和各种控制信号的 SETUP 和 HOLD 时间）
- 擦除 1 扇区（512 字节）：约 4~6ms

EEPROM 操作所需时间是硬件自动控制的，用户只需要正确设置 IAP_TPS 寄存器即可。

IAP_TPS=系统工作频率/1000000 (小数部分四舍五入进行取整)

例如：系统工作频率为 12MHz，则 IAP_TPS 设置为 12

又例如：系统工作频率为 22.1184MHz，则 IAP_TPS 设置为 22

再例如：系统工作频率为 5.5296MHz，则 IAP_TPS 设置为 6

15.2 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	-	-	-	-	0000,xxxx
IAP_TPS	IAP 等待时间控制寄存器	F5H	-	-	IAPTPS[5:0]						xx00,0000

15.2.1 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。

15.2.2 EEPROM 地址寄存器 (IAP_ADDR)

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

15.2.3 EEPROM 命令寄存器 (IAP_CMD)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CMD	C5H	-	-	-	-	-	-	CMD[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。**注意：写操作只能将目标字节中的 1 写为 0，而不能将 0 写为 1。一般当目标字节不为 FFH 时，必须先擦除。**

11: 擦除 EEPROM。擦除目标地址所在的 1 页（1 扇区/512 字节）。**注意：擦除操作会一次擦除 1 个扇区（512 字节），整个扇区的内容全部变成 FFH。**

15.2.4 EEPROM 触发寄存器 (IAP_TRIG)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_TRIG	C6H								

设置完成 EEPROM 读、写、擦除的命令寄存器、地址寄存器、数据寄存器以及控制寄存器后，需要向触发寄存器 IAP_TRIG 依次写入 5AH、A5H（顺序不能交换）两个触发命令来触发相应的读、写、擦除操作。操作完成后，EEPROM 地址寄存器 IAP_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。

15.2.5 EEPROM 控制寄存器 (IAP CONTR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP CONTR	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	-	-	-

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

15.2.6 EEPROM 等待时间控制寄存器 (IAP_TPS)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_TPS	F5H	-	-						IAPTPS[5:0]

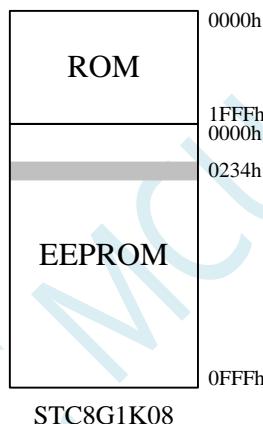
需要根据工作频率进行设置

若工作频率为12MHz, 则需要将IAP_TPS设置为12; 若工作频率为24MHz, 则需要将IAP_TPS设置为24, 其他频率以此类推。

15.3 EEPROM 大小及地址

STC8G 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。

STC8G 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 STC8G1K08 这个型号为例，对目标地址进行详细说明：



STC8G1K08 的程序空间为 8K 字节 (0000h~1FFFh)，EEPROM 空间为 4K (0000h~0FFFh)。当需要对 EEPROM 物理地址 0234h 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 0234h，即 IAP_ADDRH 设置 02h，IAP_ADDRL 设置 34h，然后设置相应的触发命令即可对 0234h 单元进行正确操作了。但若是使用 MOVC 方式读取 EEPROM 的 0234h 单元，则必须在 0234h 的基础上还有加上 Flash 程序存储器（ROM）空间的大小 2000h，即必须将 DPTR 设置为 2234h，然后才能使用 MOVC 指令进行读取。

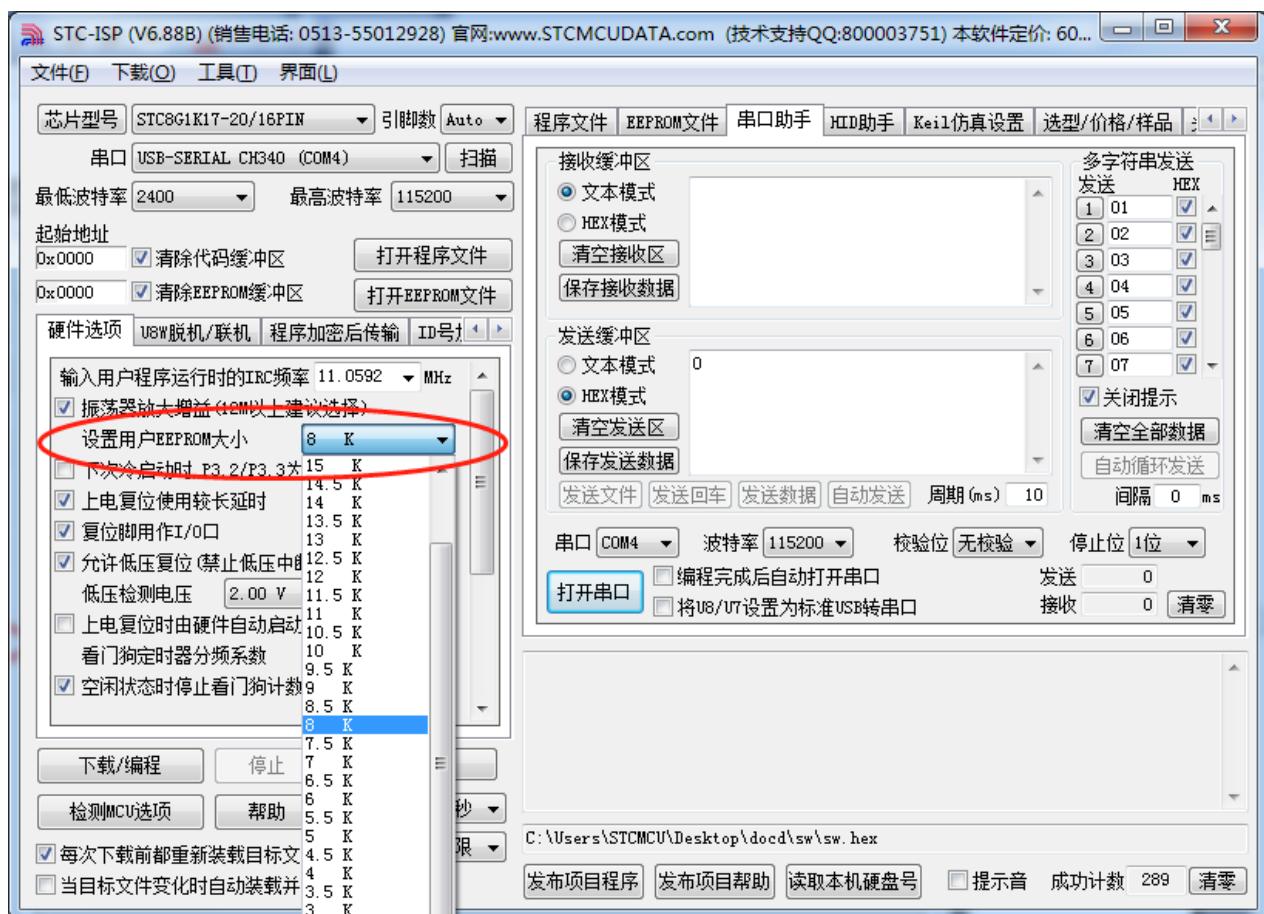
注意：由于擦除是以 512 字节为单位进行操作的，所以执行擦除操作时所设置的目标地址的低 9 位是无意义的。例如：执行擦除命令时，设置地址 0234H/0200H/0300H/03FFH，最终执行擦除的动作都是相同的，都是擦除 0200H~03FFH 这 512 字节。

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
STC8G1K04	8K	16	0000h	1FFFh	1000h	2FFFh
STC8G1K08	4K	8	0000h	0FFFh	2000h	2FFFh
STC8G1K12	用户自定义 ^[1]					
STC8G1K17	用户自定义 ^[1]					

STC8G1K08-8Pin	4K	8	0000h	0FFFh	2000h	2FFFh
STC8G1K12-8Pin	用户自定义 ^[1]					
STC8G1K17-8Pin	用户自定义 ^[1]					
STC8G1K08A	4K	8	0000h	0FFFh	2000h	2FFFh
STC8G1K12A	用户自定义 ^[1]					
STC8G1K17A	用户自定义 ^[1]					
STC8G1K08T	4K	8	0000h	0FFFh	2000h	2FFFh
STC8G1K12T	用户自定义 ^[1]					
STC8G1K17T	用户自定义 ^[1]					
STC8G2K32S4	32K	64	0000h	7FFFh	8000h	FFFFh
STC8G2K48S4	16K	32	0000h	3FFFh	C000h	FFFFh
STC8G2K60S4	4K	8	0000h	0FFFh	F000h	FFFFh
STC8G2K64S4	用户自定义 ^[1]					
STC8G2K32S2	32K	64	0000h	7FFFh	8000h	FFFFh
STC8G2K48S2	16K	32	0000h	3FFFh	C000h	FFFFh
STC8G2K60S2	4K	8	0000h	0FFFh	F000h	FFFFh
STC8G2K64S2	用户自定义 ^[1]					
STC15H2K32S4	32K	64	0000h	7FFFh	8000h	FFFFh
STC15H2K48S4	16K	32	0000h	3FFFh	C000h	FFFFh
STC15H2K60S4	4K	8	0000h	0FFFh	F000h	FFFFh
STC15H2K64S4	用户自定义 ^[1]					

^[1]: 这个为特殊型号，这个型号的 EEPROM 大小是可用在 ISP 下载时用户自己设置的。如下图所示：



用户可用根据自己的需要在整个 FLASH 空间中规划出任意不超过 FLASH 大小的 EEPROM 空间，但需要注意：**EEPROM 总是从后向前进行规划的。**

例如：STC8G1K17 这个型号的 FLASH 为 17K，此时若用户想分出其中的 4K 作为 EEPROM 使用，则 EEPROM 的物理地址则为 17K 的最后 4K，物理地址为 3400h~43FFh，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 0FFFh 结束，当使用 MOVC 读取则需要从 3400h 开始，到 43FFh 结束。**注意：STC8G1K17 这个型号的程序空间为 17K，即整个 17K 的范围均可运行程序，即使在 ISP 下载时将 17K 最后的 4K 设置为 EEPROM 使用，但这分配的 4K 空间仍然可以运行程序。其它可自定义 EEPROM 大小的型号与此类似。**

15.4 范例程序

15.4.1 EEPROM 基本操作

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

sfr IAP_DATA    = 0xC2;
sfr IAP_ADDRH   = 0xC3;
sfr IAP_ADDRL   = 0xC4;
sfr IAP_CMD     = 0xC5;
sfr IAP_TRIG    = 0xC6;
sfr IAP_CONTR   = 0xC7;
sfr IAP_TPS     = 0xF5;

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 = 0x80;                            //使能 IAP
    IAP_TPS = 12;                               //设置等待参数 12MHz
    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 = 0xa5;                            //写触发命令(0xa5)
    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 = 0x80;                                //使能IAP
    IAP TPS = 12;                                   //设置等待参数12MHz
    IAP CMD = 2;                                    //设置IAP写命令
    IAP ADDRLO = 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 = 0x80;                                //使能IAP
    IAP TPS = 12;                                   //设置等待参数12MHz
    IAP CMD = 3;                                    //设置IAP擦除命令
    IAP ADDRLO = 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()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IapErase(0x0400);
    P0 = IapRead(0x0400);                           //P0=0xff
    IapProgram(0x0400, 0x12);
    P1 = IapRead(0x0400);                           //P1=0x12

    while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

IAP_DATA	DATA	0C2H
IAP_ADDRH	DATA	0C3H

<i>IAP_ADDRL</i>	DATA	0C4H
<i>IAP_CMD</i>	DATA	0C5H
<i>IAP_TRIG</i>	DATA	0C6H
<i>IAP_CONTR</i>	DATA	0C7H
<i>IAP_TPS</i>	DATA	0F5H
<i>P0M1</i>	DATA	093H
<i>P0M0</i>	DATA	094H
<i>P1M1</i>	DATA	091H
<i>P1M0</i>	DATA	092H
<i>P2M1</i>	DATA	095H
<i>P2M0</i>	DATA	096H
<i>P3M1</i>	DATA	0B1H
<i>P3M0</i>	DATA	0B2H
<i>P4M1</i>	DATA	0B3H
<i>P4M0</i>	DATA	0B4H
<i>P5M1</i>	DATA	0C9H
<i>P5M0</i>	DATA	0CAH
ORG		0000H
LJMP		MAIN
ORG		0100H
IAP_IDLE:		
MOV		<i>IAP_CONTR</i> ,#0
MOV		<i>IAP_CMD</i> ,#0
MOV		<i>IAP_TRIG</i> ,#0
MOV		<i>IAP_ADDRH</i> ,#80H
MOV		<i>IAP_ADDRL</i> ,#0
RET		
IAP_READ:		
MOV		<i>IAP_CONTR</i> ,#80H
MOV		<i>IAP_TPS</i> ,#12
MOV		<i>IAP_CMD</i> ,#1
MOV		<i>IAP_ADDRL,DPL</i>
MOV		<i>IAP_ADDRH,DPH</i>
MOV		<i>IAP_TRIG</i> ,#5AH
MOV		<i>IAP_TRIG</i> ,#0A5H
NOP		
MOV		A, <i>IAP_DATA</i>
LCALL		<i>IAP_IDLE</i>
RET		
IAP_PROGRAM:		
MOV		<i>IAP_CONTR</i> ,#80H
MOV		<i>IAP_TPS</i> ,#12
MOV		<i>IAP_CMD</i> ,#2
MOV		<i>IAP_ADDRL,DPL</i>
MOV		<i>IAP_ADDRH,DPH</i>
MOV		<i>IAP_DATA</i> ,A
MOV		<i>IAP_TRIG</i> ,#5AH
MOV		<i>IAP_TRIG</i> ,#0A5H
NOP		
LCALL		<i>IAP_IDLE</i>
RET		;关闭IAP 功能
IAP_ERASE:		

<i>MOV</i>	<i>IAP_CONTR,#80H</i>	;使能 IAP
<i>MOV</i>	<i>IAP_TPS,#12</i>	;设置等待参数 12MHz
<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, #5FH</i>	
<i>MOV</i>	<i>P0M0, #00H</i>	
<i>MOV</i>	<i>P0M1, #00H</i>	
<i>MOV</i>	<i>P1M0, #00H</i>	
<i>MOV</i>	<i>P1M1, #00H</i>	
<i>MOV</i>	<i>P2M0, #00H</i>	
<i>MOV</i>	<i>P2M1, #00H</i>	
<i>MOV</i>	<i>P3M0, #00H</i>	
<i>MOV</i>	<i>P3M1, #00H</i>	
<i>MOV</i>	<i>P4M0, #00H</i>	
<i>MOV</i>	<i>P4M1, #00H</i>	
<i>MOV</i>	<i>P5M0, #00H</i>	
<i>MOV</i>	<i>P5M1, #00H</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

15.4.2 使用 MOVC 读取 EEPROM

C 语言代码

```
//测试工作频率为11.0592MHz
```

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

```
sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
```

```

sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

sfr IAP_DATA  = 0xC2;
sfr IAP_ADDRH = 0xC3;
sfr IAP_ADDRL = 0xC4;
sfr IAP_CMD   = 0xC5;
sfr IAP_TRIG  = 0xC6;
sfr IAP_CONTR = 0xC7;
sfr IAP_TPS   = 0xF5;

#define IAP_OFFSET 0x2000 //STC8G1K08

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 = 0x80;        //使能IAP
    IAP_TPS = 12;             //设置等待参数 12MHz
    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 = 0x80;        //使能IAP
    IAP_TPS = 12;             //设置等待参数 12MHz
    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()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IapErase(0x0400);
    P0 = IapRead(0x0400);                                //P0=0xff
    IapProgram(0x0400, 0x12);
    P1 = IapRead(0x0400);                                //P1=0x12

    while (1);
}

```

汇编代码

;测试工作频率为 11.0592MHz

<i>IAP_DATA</i>	<i>DATA</i>	0C2H	
<i>IAP_ADDRH</i>	<i>DATA</i>	0C3H	
<i>IAP_ADDRL</i>	<i>DATA</i>	0C4H	
<i>IAP_CMD</i>	<i>DATA</i>	0C5H	
<i>IAP_TRIG</i>	<i>DATA</i>	0C6H	
<i>IAP_CONTR</i>	<i>DATA</i>	0C7H	
<i>IAP_TPS</i>	<i>DATA</i>	0F5H	
<i>IAP_OFFSET</i>	<i>EQU</i>	2000H	;STC8G1K08
<i>P0M1</i>	<i>DATA</i>	093H	
<i>P0M0</i>	<i>DATA</i>	094H	
<i>P1M1</i>	<i>DATA</i>	091H	
<i>P1M0</i>	<i>DATA</i>	092H	
<i>P2M1</i>	<i>DATA</i>	095H	
<i>P2M0</i>	<i>DATA</i>	096H	
<i>P3M1</i>	<i>DATA</i>	0B1H	
<i>P3M0</i>	<i>DATA</i>	0B2H	
<i>P4M1</i>	<i>DATA</i>	0B3H	
<i>P4M0</i>	<i>DATA</i>	0B4H	
<i>P5M1</i>	<i>DATA</i>	0C9H	
<i>P5M0</i>	<i>DATA</i>	0CAH	
	<i>ORG</i>	0000H	
	<i>LJMP</i>	MAIN	
	<i>ORG</i>	0100H	
<i>IAP_IDLE:</i>			
	<i>MOV</i>	<i>IAP_CONTR</i> ,#0	;关闭IAP 功能
	<i>MOV</i>	<i>IAP_CMD</i> ,#0	;清除命令寄存器
	<i>MOV</i>	<i>IAP_TRIG</i> ,#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,#80H          ;使能IAP
MOV      IAP_TPS,#12              ;设置等待参数 12MHz
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)
NOP
LCALL   IAP_IDLE               ;关闭IAP 功能
RET

```

IAP_ERASE:

```

MOV      IAP_CONTR,#80H          ;使能IAP
MOV      IAP_TPS,#12              ;设置等待参数 12MHz
MOV      IAP_CMD,#3              ;设置IAP 擦除命令
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
LCALL   IAP_IDLE               ;关闭IAP 功能
RET

```

MAIN:

```

MOV      SP, #5FH
MOV      P0M0, #00H
MOV      P0M1, #00H
MOV      P1M0, #00H
MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

MOV      DPTR, #0400H
LCALL   IAP_ERASE
MOV      DPTR, #0400H
LCALL   IAP_READ

```

```

MOV      P0,A          ;P0=0FFH
MOV      DPTR,#0400H
MOV      A,#12H
LCALL   IAP_PROGRAM
MOV      DPTR,#0400H
LCALL   IAP_READ
MOV      P1,A          ;P1=12H

SJMP    $

END

```

15.4.3 使用串口送出 EEPROM 数据

C 语言代码

//测试工作频率为 11.0592MHz

```

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

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

sfr    P0M1      = 0x93;
sfr    P0M0      = 0x94;
sfr    P1M1      = 0x91;
sfr    P1M0      = 0x92;
sfr    P2M1      = 0x95;
sfr    P2M0      = 0x96;
sfr    P3M1      = 0xb1;
sfr    P3M0      = 0xb2;
sfr    P4M1      = 0xb3;
sfr    P4M0      = 0xb4;
sfr    P5M1      = 0xc9;
sfr    P5M0      = 0xca;

sfr    AUXR      = 0x8e;
sfr    T2H       = 0xd6;
sfr    T2L       = 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;
sfr    IAP_TPS    = 0xF5;

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 = 0x80;                            //使能IAP
    IAP_TPS = 12;                               //设置等待参数 12MHz
    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 = 0x80;                            //使能IAP
    IAP_TPS = 12;                               //设置等待参数 12MHz
    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 = 0x80;                            //使能IAP
    IAP_TPS = 12;                               //设置等待参数 12MHz
    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()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    UartInit();
    IapErase(0x0400);
    UartSend(IapRead(0x0400));
    IapProgram(0x0400, 0x12);
    UartSend(IapRead(0x0400));

    while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

<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>IAP_TPS</i>	<i>DATA</i>	<i>0F5H</i>
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</i>
<i>ORG</i>		<i>0000H</i>
<i>LJMP</i>		<i>MAIN</i>
<i>ORG</i>		<i>0100H</i>

UART_INIT:

MOV	SCON,#5AH	
MOV	T2L,#0E8H	;65536-11059200/115200/4=0FFE8H
MOV	T2H,#0FFH	
MOV	AUXR,#15H	
RET		

UART_SEND:

JNB	TI,\$	
CLR	TI	
MOV	SBUF,A	
RET		

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,#80H	;使能 IAP
MOV	IAP TPS,#12	;设置等待参数 12MHz
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,#80H	;使能 IAP
MOV	IAP TPS,#12	;设置等待参数 12MHz
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)
NOP		
LCALL	IAP IDLE	;关闭 IAP 功能
RET		

IAP_ERASE:

MOV	IAP CONTR,#80H	;使能 IAP
MOV	IAP TPS,#12	;设置等待参数 12MHz
MOV	IAP CMD,#3	;设置 IAP 擦除命令
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		
LCALL	IAP IDLE	;关闭 IAP 功能
RET		

MAIN:

```

MOV      SP, #5FH
MOV      P0M0, #00H
MOV      P0M1, #00H
MOV      P1M0, #00H
MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

LCALL    UART_INIT
MOV      DPTR, #0400H
LCALL    IAP_ERASE
MOV      DPTR, #0400H
LCALL    IAP_READ
LCALL    UART_SEND
MOV      DPTR, #0400H
MOV      A, #12H
LCALL    IAP_PROGRAM
MOV      DPTR, #0400H
LCALL    IAP_READ
LCALL    UART_SEND

SJMP    $

```

END

15.4.4 串口 1 读写 EEPROM-带 MOVC 读

C 语言代码 (main.c)

//测试工作频率为11.0592MHz

/* 本程序经过测试完全正常，不提供电话技术支持，如不能理解，请自行补充相关基础 */

***** 本程序功能说明 *****

STC8G 系列EEPROM 通用测试程序

请先别修改程序，直接下载'02-串口 1 读写 EEPROM-带 MOVC 读'里面的"UART-EEPROM.hex"测试。下载时选择主频 11.0592MHZ。

PC 串口设置：波特率 115200,8,n,1.

对EEPROM 做扇区擦除、写入 64 字节、读出 64 字节的操作。

命令例子：

E 0 对EEPROM 进行扇区擦除操作，E 表示擦除，数字 0 为 0 扇区(十进制 0~126，看具体 IC)。

W 0 对EEPROM 进行写入操作，W 表示写入，数字 0 为 0 扇区(十进制 0~126，看具体 IC)。从扇区的开始地址连续写 64 字节。

R 0 对EEPROM 进行IAP 读出操作，R 表示读出，数字 0 为 0 扇区(十进制 0~126，看具体 IC)。从扇区的开始地址连续读 64 字节。

M 0 对EEPROM 进行MOVC 读出操作(操作地址为扇区*512+偏移地址) 数字 0 为 0 扇区(十进制 0~126，看具体 IC)。从扇区的开始地址连续读 64 字节。

注意: 为了通用, 程序不识别扇区是否有效, 用户自己根据具体的型号来决定。

日期: 2019-6-10

```
#include "config.H"
#include "EEPROM.h"

#define Baudrate1           115200L
#define UART1_BUF_LENGTH    10
#define EEADDR_OFFSET        (8 * 1024)      //定义 EEPROM 用 MOVC 访问时加的偏移量,
                                                //等于 FLASH ROM 的大小对于 IAP 或 IRC 开头的,
                                                //则偏移量必须为 0
#define TimeOutSet1          5

/************ 本地常量声明 *****/
u8 code T_Strings[]={"去年今日此门中，人面桃花相映红。人面不知何处去，桃花依旧笑春风。"};

/************ 本地变量声明 *****/
u8 xdatatmp[70];
u8 xdataRX1_Buffer[UART1_BUF_LENGTH];
u8 RX1_Cnt;
u8 RX1_TimeOut;
bit B_RX1_Busy;

/************ 本地函数声明 *****/
void UART1_config(void);
void TX1_write2buff(u8 dat);           //写入发送缓冲
void PrintString1(u8 *puts);          //发送一个字符串

/************ 外部函数和变量声明 *****/
/******************/


u8 CheckData(u8 dat)
{
    if((dat >= '0') && (dat <= '9')) return (dat-'0');
    if((dat >= 'A') && (dat <= 'F')) return (dat-'A'+10);
    if((dat >= 'a') && (dat <= 'f')) return (dat-'a'+10);
    return 0xff;
}

u16 GetAddress(void)
{
    u16 address;
    u8 i;

    address = 0;
    if(RX1_Cnt < 3) return 65535;           //error
    if(RX1_Cnt <= 5)                         //5 个字节以内是扇区操作, 十进制
                                                //支持命令: E 0, E 12, E 120
                                                //          W 0, W 12, W 120
                                                //          R 0, R 12, R 120
    {
        for(i=2; i<RX1_Cnt; i++)
        {
            if(CheckData(RX1_Buffer[i]) > 9)
                return 65535;                  //error
        }
    }
}
```

```

        address = address * 10 + CheckData(RXI_Buffer[i]);
    }
    if(address < 124)                                //限制在0~123 扇区
    {
        address <<= 9;
        return (address);
    }
}
else if(RXI_Cnt == 8)                            //8 个字节直接地址操作，十六进制
                                                //支持命令: E 0x1234, W 0x12b3, R 0xA00
{
    if((RXI_Buffer[2] == '0') && ((RXI_Buffer[3] == 'x') || (RXI_Buffer[3] == 'X')))
    {
        for(i=4; i<8; i++)
        {
            if(CheckData(RXI_Buffer[i]) > 0x0F)
                return 65535;                      //error
            address = (address << 4) + CheckData(RXI_Buffer[i]);
        }
        if(address < 63488)                    //限制在0~123 扇区
            return (address);
    }
}
return 65535;                                     //error
}

//=====================================================================
// 函数: void delay_ms(u8 ms)
// 描述: 延时函数。
// 参数: ms, 要延时的ms 数, 这里只支持1~255ms. 自动适应主时钟
// 返回: none.
// 版本: VER1.0
// 日期: 2013-4-1
// 备注:
//=====================================================================
void delay_ms(u8 ms)
{
    u16 i;
    do
    {
        i = MAIN_Fosc / 10000;
        while(--i) ;
    }while(--ms);
}

//使用MOVC 读EEPROM
void EEPROM_MOVC_read_n(u16 EE_address, u8 *DataAddress, u16 number)
{
    u8 code *pc;

    pc = EE_address + EEADDR_OFFSET;
    do
    {
        *DataAddress = *pc;                  //读出的数据
        DataAddress++;
        pc++;
    }while(--number);
}

```

```

***** 主函数 *****
void main(void)
{
    u8 i;
    u16 addr;

    UART1_config();                                // 选择波特率 2: 使用 Timer2 做波特率
                                                //其它值: 使用 Timer1 做波特率
    EA = I;                                         //允许总中断

    PrintString1("STC8 系列MCU 用串口1 测试 EEPROM 程序!\r\n"); //UART1 发送一个字符串

    while(1)
    {
        delay_ms(1);
        if(RX1_TimeOut > 0)                         //超时计数
        {
            if(--RX1_TimeOut == 0)
            {
                if(RXI_Buffer[1] == ' ')
                {
                    addr = GetAddress();
                    if(addr < 63488)                      //限制在 0~123 扇区
                    {
                        if(RXI_Buffer[0] == 'E') //PC 请求擦除一个扇区
                        {
                            EEPROM_SectorErase(addr);
                            PrintString1("扇区擦除完成!\r\n");
                        }
                        else if(RXI_Buffer[0] == 'W') //PC 请求写入 EEPROM 64 字节数据
                        {
                            EEPROM_write_n(addr,T_Strings,64);
                            PrintString1("写入操作完成!\r\n");
                        }
                        else if(RXI_Buffer[0] == 'R') //PC 请求返回 64 字节EEPROM 数据
                        {
                            PrintString1("IAP 读出的数据如下:\r\n");
                            EEPROM_read_n(addr,tmp,64);
                            for(i=0; i<64; i++)
                                TX1_write2buff(tmp[i]); //将数据返回给串口
                            TX1_write2buff(0xd);
                            TX1_write2buff(0xa);
                        }
                        else if(RXI_Buffer[0] == 'M') //PC 请求返回 64 字节EEPROM 数据
                        {
                            PrintString1("MOVC 读出的数据如下:\r\n");
                            EEPROM_MOVC_read_n(addr,tmp,64);
                            for(i=0; i<64; i++)
                                TX1_write2buff(tmp[i]); //将数据返回给串口
                            TX1_write2buff(0xd);
                            TX1_write2buff(0xa);
                        }
                        else PrintString1("命令错误!\r\n");
                    }
                    else PrintString1("命令错误!\r\n");
                }
            }
        }
    }
}

```

```
        RX1_Cnt = 0;
    }
}
}

/*****发送一个字节 *****/
void TX1_write2buff(u8 dat)           //写入发送缓冲
{
    B_TX1_Busy = 1;                  //标志发送忙
    SBUF = dat;                     //发送一个字节
    while(B_TX1_Busy);             //等待发送完毕
}

//=====
// 函数: void PrintString1(u8 *puts)
// 描述: 串口1发送字符串函数。
// 参数: puts: 字符串指针。
// 返回: none.
// 版本: VER1.0
// 日期: 2014-11-28
// 备注:
//=====
void PrintString1(u8 *puts)           //发送一个字符串
{
    for (; *puts != 0; puts++)
    {
        TX1_write2buff(*puts);      //遇到停止符0 结束
    }
}

//=====
// 函数: void    UART1_config(void)
// 描述: UART1 初始化函数。
// 参数: none.
// 返回: none.
// 版本: VER1.0
// 日期: 2014-11-28
// 备注:
//=====
void    UART1_config(void)
{
    TR1 = 0;
    AUXR &= ~0x01;                //SI BRT Use Timer1;
    AUXR |= (1<<6);              //Timer1 set as 1T mode
    TMOD &= ~(1<<6);            //Timer1 set As Timer
    TMOD &= ~0x30;                //Timer1_16bitAutoReload;
    TH1 = (u8)((65536L-(MAIN_Fosc / 4) / Baudrate1) >> 8);
    TL1 = (u8)(65536L-(MAIN_Fosc / 4) / Baudrate1);
    ET1 = 0;                      // 禁止Timer1 中断
    INT_CLKO &= ~0x02;            // Timer1 不输出高速时钟
    TR1 = 1;                      // 运行Timer1

    S1_USE_P30P31();  P3n_standard(0x03);      //切换到 P3.0 P3.1
    //S1_USE_P36P37(); P3n_standard(0xc0);      //切换到 P3.6 P3.7
    //S1_USE_P16P17(); P1n_standard(0xc0);      //切换到 P1.6 P1.7
}
```

```

SCON = (SCON & 0x3f) / 0x40;           //UART1 模式 0x00: 同步移位输出,
//                                         0x40: 8 位数据 可变波特率,
//                                         0x80: 9 位数据 固定波特率,
//                                         0xc0: 9 位数据 可变波特率
// PS = 1;                                //高优先级中断
ES = 1;                                  //允许中断
REN = 1;                                 //允许接收

B_RXI_Busy = 0;
RXI_Cnt = 0;
}

//=====================================================================
// 函数: void UART1_int (void) interrupt UART1_VECTOR
// 描述: UART1 中断函数。
// 参数: none.
// 返回: none.
// 版本: VER1.0
// 日期: 2014-11-28
// 备注:
//=====================================================================
void UART1_int (void) interrupt 4
{
    if(RI)
    {
        RI = 0;
        if(RXI_Cnt >= UART1_BUF_LENGTH)
            RXI_Cnt = 0;                           //防溢出
        RXI_Buffer[RXI_Cnt++] = SBUF;
        RXI_TimeOut = TimeOutSet1;
    }

    if(TI)
    {
        TI = 0;
        B_RXI_Busy = 0;
    }
}

```

C 语言代码 (EEPROM.c)

```

//测试工作频率为 11.0592MHz

// 本程序是 STC 系列的内置 EEPROM 读写程序。

#include "config.h"
#include "eeprom.h"

//=====================================================================
// 函数: void ISP_Disable(void)
// 描述: 禁止访问ISP/IAP.
// 参数: non.
// 返回: non.
// 版本: V1.0, 2012-10-22
//=====================================================================
void DisableEEPROM(void)
{
    ISP_CONTR = 0;                         //禁止 ISP/IAP 操作
    IAP_TPS = 0;
}

```

```

ISP_CMD = 0;                                //去除 ISP/IAP 命令
ISP_TRIG = 0;                                //防止 ISP/IAP 命令误触发
ISP_ADDRH = 0xff;                            //清 0 地址高字节
ISP_ADDRL = 0xff;                            //清 0 地址低字节，指向非 EEPROM 区，防止误操作
}

//=====
// 函数: void EEPROM_read_n(u16 EE_address,u8 *DataAddress,u16 number)
// 描述: 从指定 EEPROM 首地址读出 n 个字节放指定的缓冲。
// 参数: EE_address: 读出 EEPROM 的首地址。
//        DataAddress: 读出数据放缓冲的首地址。
//        number:      读出的字节长度。
// 返回: non.
// 版本: V1.0, 2012-10-22
//=====

void EEPROM_read_n(u16 EE_address,u8 *DataAddress,u16 number)
{
    EA = 0;                                    //禁止中断
    ISP_CONTR = ISP_EN;                      //允许 ISP/IAP 操作
    IAP_TPS = (u8)(MAIN_Fosc / 1000000L);   //工作频率设置
    ISP_READ();                                //送字节读命令，命令不需改变时，不需重新送命令
    do
    {
        ISP_ADDRH = EE_address / 256;          //送地址高字节（地址需要改变时才需重新送地址）
        ISP_ADDRL = EE_address % 256;          //送地址低字节
        ISP_TRIG();                            //先送 5AH，再送 A5H 到 ISP/IAP 触发寄存器，每次都需要如此
                                                //送完 A5H 后，ISP/IAP 命令立即被触发启动
                                                //CPU 等待 IAP 完成后，才会继续执行程序。
        _nop_();
        _nop_();
        _nop_();
        *DataAddress = ISP_DATA;                //读出的数据送往
        EE_address++;                          //EE_address++
        DataAddress++;                        //DataAddress++
    }while(--number);

    DisableEEPROM();
    EA = 1;                                    //重新允许中断
}

//*****
// 扇区擦除函数 *****
//=====

// 函数: void EEPROM_SectorErase(u16 EE_address)
// 描述: 把指定地址的 EEPROM 扇区擦除。
// 参数: EE_address: 要擦除的扇区 EEPROM 的地址。
// 返回: non.
// 版本: V1.0, 2013-5-10
//=====

void EEPROM_SectorErase(u16 EE_address)
{
    EA = 0;                                    //禁止中断
                                                //只有扇区擦除，没有字节擦除，512 字节/扇区。
                                                //扇区中任意一个字节地址都是扇区地址。
    ISP_ADDRH = EE_address / 256;              //送扇区地址高字节（地址需要改变时才需重新送地址）
    ISP_ADDRL = EE_address % 256;              //送扇区地址低字节
    ISP_CONTR = ISP_EN;                      //允许 ISP/IAP 操作
    IAP_TPS = (u8)(MAIN_Fosc / 1000000L);   //工作频率设置
    ISP_ERASE();                            //送扇区擦除命令，命令不需改变时，不需重新送命令
}

```

```

ISP_TRIG();
_nop_();
_nop_();
_nop_();
DisableEEPROM();
EA = I;                                //重新允许中断
}

//=====
// 函数: void EEPROM_write_n(u16 EE_address,u8 *DataAddress,u16 number)
// 描述: 把缓冲的n个字节写入指定首地址的EEPROM.
// 参数: EE_address: 写入EEPROM 的首地址.
//        DataAddress: 写入源数据的缓冲的首地址.
//        number:      写入的字节长度.
// 返回: non.
// 版本: V1.0, 2012-10-22
//=====

void EEPROM_write_n(u16 EE_address,u8 *DataAddress,u16 number)
{
    EA = 0;                                //禁止中断

    ISP_CONTR = ISP_EN;                    //允许ISP/IAP 操作
    IAP_TPS = (u8)(MAIN_Fosc / 1000000L); //工作频率设置
    ISP_WRITE();                           //送字节写命令，命令不需改变时，不需重新送命令
    do
    {
        ISP_ADDRH = EE_address / 256;      //送地址高字节 (地址需要改变时才需重新送地址)
        ISP_ADDRL = EE_address % 256;      //送地址低字节
        ISP_DATA = *DataAddress;           //送数据到ISP_DATA，只有数据改变时才需重新送
        ISP_TRIG();                        //送数据
        _nop_();
        _nop_();
        _nop_();
        EE_address++;                     //地址加1
        DataAddress++;                   //数据指针加1
    }while(--number);

    DisableEEPROM();
    EA = I;                                //重新允许中断
}

```

15.4.5 口令擦除写入-多扇区备份-串口 1 操作

C 语言代码 (main.c)

```

//测试工作频率为11.0592MHz

/*
 * 本程序经过测试完全正常, 不提供电话技术支持, 如不能理解, 请自行补充相关基础 */
***** 本程序功能说明 *****

STC8G 系列 STC8H 系列 STC8C 系列 EEPROM 通用测试程序, 演示多扇区备份、有扇区错误则用正确扇区数据写入、全部扇区错误(比如第一次运行程序)则写入默认值.

```

每次写都写入 3 个扇区, 即冗余备份.
 每个扇区写一条记录, 写入完成后读出保存的数据和校验值跟源数据和校验值比较, 并从串口 1(P3.0 P3.1)返回结果(正确或错误提示).
 每条记录自校验 64 字节数据 2 字节校验值, 校验值 = 64 字节数据累加和 ^ 0x5555. ^0x5555 是为了保证写入的 66 个数据不全部为 0.

如果有扇区错误，则将正确扇区的数据写入错误扇区，如果3个扇区都错误，则均写入默认值。
擦除、写入、读出操作前均需要设置口令，如果口令不对则退出操作，并且每次退出操作都会清除口令。

请先别修改程序，直接下载“03-口令擦除写入-多扇区备份-串口1操作”里面的“UART-EEPROM.hex”测试。下载时选择主频11.0592MHZ。

PC 串口设置：波特率115200,8,n,1。

对EEPROM 做扇区擦除、写入 64 字节、读出 64 字节的操作。

命令例子：

使用串口助手发单个字符，大小写均可。

发E 或 e： 对EEPROM 进行扇区擦除操作，E 表示擦除，会擦除扇区0、1、2。

发 W 或 w： 对EEPROM 进行写入操作，W 表示写入，会写入扇区0、1、2，每个扇区连续写 64 字节，扇区0 写入0x0000~0x003f，扇区1 写入0x0200~0x023f，扇区0 写入0x0400~0x043f。

发 R 或 r： 对EEPROM 进行读出操作，R 表示读出，会读出扇区0、1、2，每个扇区连续读 64 字节，扇区0 读出0x0000~0x003f，扇区1 读出0x0200~0x023f，扇区0 读出0x0400~0x043f。

注意：为了通用，程序不识别扇区是否有效，用户自己根据具体的型号来决定。

日期: 2021-11-5

```
#include "config.H"
#include "EEPROM.h"

#define Baudrate1 115200L

***** 本地常量声明 *****
u8 code T_StringD[]={"去年今日此门中，人面桃花相映红。人面不知何处去，桃花依旧笑春风。"};
u8 code T_StringW[]={"横看成岭侧成峰，远近高低各不同。不识庐山真面目，只缘身在此山中。"};

***** 本地变量声明 *****
u8 xdata tmp[70]; //通用数据
u8 xdata SaveTmp[70]; //要写入的数组

bit B_TX1_Busy;
u8 cmd; //串口单字符命令

***** 本地函数声明 *****
void UART1_config(void);
void TX1_write2buff(u8 dat); //写入发送缓冲
void PrintString1(u8 *puts); //发送一个字符串

***** 外部函数和变量声明 *****
//读取EEPROM 记录，并且校验，返回校验结果 0 为正确，1 为错误 *****
u8 ReadRecord(u16 addr)
{
    u8 i;
    u16 ChckSum; //计算的累加和
    u16 j; //读取的累加和

    for(i=0; i<66; i++) tmp[i] = 0; //清除缓冲
}
```

```

PassWord = D_PASSWORD;           //给定口令
EEPROM_read_n(addr,tmp,66);     //读出扇区0
for(ChckSum=0, i=0; i<64; i++)
{
    ChckSum += tmp[i];          //计算累加和
    j = ((u16)tmp[64]<<8) + (u16)tmp[65];
    j ^= 0x5555;                //隔位取反，避免全0
    if(ChckSum != j)      return 1; //累加和错误，返回1
    return 0;                  //累加和正确，返回0
}

/**************************************** 写入 EEPROM 记录，并且校验，返回校验结果 0 为正确, 1 为错误 *****/
u8 SaveRecord(u16 addr)
{
    u8   i;
    u16 ChckSum;               //计算的累加和

    for(ChckSum=0, i=0; i<64; i++)
    {
        ChckSum += SaveTmp[i]; //计算累加和
        ChckSum ^= 0x5555;    //隔位取反，避免全0
        SaveTmp[64] = (u8)(ChckSum >> 8);
        SaveTmp[65] = (u8)ChckSum;

        PassWord = D_PASSWORD; //给定口令
        EEPROM_SectorErase(addr); //擦除一个扇区
        PassWord = D_PASSWORD; //给定口令
        EEPROM_write_n(addr, SaveTmp, 66); //写入扇区

        for(i=0; i<66; i++)
        {
            tmp[i] = 0;          //清除缓冲
            PassWord = D_PASSWORD; //给定口令
            EEPROM_read_n(addr,tmp,66); //读出扇区0
            for(i=0; i<66; i++) //数据比较
            {
                if(SaveTmp[i] != tmp[i]) //数据有错误，返回1
                    return 1;
            }
        }
        return 0;                //累加和正确，返回0
    }

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

    UART1_config();             //选择波特率 2: 使用 Timer2 做波特率
                                //其它值: 使用 Timer1 做波特率
    EA = I;                    //允许总中断

    PrintString1("STC8G-8H-8C 系列 MCU 用串口1 测试 EEPROM 程序!\r\n"); //UART1 发送一个字符串

    //上电读取 3 个扇区并校验，如果有扇区错误则将正确的
    //扇区写入错误扇区，如果 3 个扇区都错误，则写入默认值

    status = 0;
    if(ReadRecord(0x0000) == 0) //读扇区0
    {
        status |= 0x01;         //正确则标记 status.0=1
        for(i=0; i<64; i++) //保存在写缓冲
            SaveTmp[i] = tmp[i];
    }
}

```

```

}

if(ReadRecord(0x0200) == 0) //读扇区1
{
    status |= 0x02; //正确则标记status.1=1
    for(i=0; i<64; i++)
        SaveTmp[i] = tmp[i]; //保存在写缓冲
}
if(ReadRecord(0x0400) == 0) //读扇区2
{
    status |= 0x04; //正确则标记status.2=1
    for(i=0; i<64; i++)
        SaveTmp[i] = tmp[i]; //保存在写缓冲
}

if(status == 0) //所有扇区都错误 则写入默认值
{
    for(i=0; i<64; i++)
        SaveTmp[i] = T_StringD[i]; //读取默认值
}
else PrintString1("上电读取3个扇区数据均正确!\r\n"); //UART1 发送一个字符串提示

if((status & 0x01) == 0) //扇区0 错误 则写入默认值
{
    if(SaveRecord(0x0000) == 0)
        PrintString1("写入扇区0 正确!\r\n"); //写入记录0 扇区正确
    else
        PrintString1("写入扇区0 错误!\r\n"); //写入记录0 扇区错误
}
if((status & 0x02) == 0) //扇区1 错误 则写入默认值
{
    if(SaveRecord(0x0200) == 0)
        PrintString1("写入扇区1 正确!\r\n"); //写入记录1 扇区正确
    else
        PrintString1("写入扇区1 错误!\r\n"); //写入记录1 扇区错误
}
if((status & 0x04) == 0) //扇区2 错误 则写入默认值
{
    if(SaveRecord(0x0400) == 0)
        PrintString1("写入扇区2 正确!\r\n"); //写入记录2 扇区正确
    else
        PrintString1("写入扇区2 错误!\r\n"); //写入记录2 扇区错误
}

while(1)
{
    if(cmd != 0) //有串口命令
    {
        if((cmd >= 'a') && (cmd <= 'z'))
            cmd -= 0x20; //小写转大写

        if(cmd == 'E') //PC 请求擦除一个扇区
        {
            PassWord = D_PASSWORD; //给定口令
            EEPROM_SectorErase(0x0000); //擦除一个扇区
            PassWord = D_PASSWORD; //给定口令
            EEPROM_SectorErase(0x0200); //擦除一个扇区
            PassWord = D_PASSWORD; //给定口令
            EEPROM_SectorErase(0x0400); //擦除一个扇区
        }
    }
}

```

```

        PrintString1("扇区擦除完成!\r\n");
    }

    else if(cmd == 'W')                                //PC 请求写入 EEPROM 64 字节数据
    {
        for(i=0; i<64; i++)
            SaveTmp[i] = T_StringW[i];                //写入数值
        if(SaveRecord(0x0000) == 0)
            PrintString1("写入扇区0 正确!\r\n");        //写入记录0 扇区正确
        else
            PrintString1("写入扇区0 错误!\r\n");        //写入记录0 扇区错误
        if(SaveRecord(0x0200) == 0)
            PrintString1("写入扇区1 正确!\r\n");        //写入记录1 扇区正确
        else
            PrintString1("写入扇区1 错误!\r\n");        //写入记录1 扇区错误
        if(SaveRecord(0x0400) == 0)
            PrintString1("写入扇区2 正确!\r\n");        //写入记录2 扇区正确
        else
            PrintString1("写入扇区2 错误!\r\n");
    }

    else if(cmd == 'R')                                //PC 请求返回 64 字节 EEPROM 数据
    {
        if(ReadRecord(0x0000) == 0)                     //读出扇区0 的数据
        {
            PrintString1("读出扇区0 的数据如下:\r\n");
            for(i=0; i<64; i++)
                TX1_write2buff(tmp[i]);                //将数据返回给串口
            TX1_write2buff(0xd);                      //回车换行
            TX1_write2buff(0xa);
        }
        else PrintString1("读出扇区0 的数据错误!\r\n");

        if(ReadRecord(0x0200) == 0)                     //读出扇区1 的数据
        {
            PrintString1("读出扇区1 的数据如下:\r\n");
            for(i=0; i<64; i++)
                TX1_write2buff(tmp[i]);                //将数据返回给串口
            TX1_write2buff(0xd);                      //回车换行
            TX1_write2buff(0xa);
        }
        else PrintString1("读出扇区1 的数据错误!\r\n");

        if(ReadRecord(0x0400) == 0)                     //读出扇区2 的数据
        {
            PrintString1("读出扇区2 的数据如下:\r\n");
            for(i=0; i<64; i++)
                TX1_write2buff(tmp[i]);                //将数据返回给串口
            TX1_write2buff(0xd);                      //回车换行
            TX1_write2buff(0xa);
        }
        else PrintString1("读出扇区2 的数据错误!\r\n");
    }

    else PrintString1("命令错误!\r\n");
    cmd = 0;
}
}
/*****************************************/

```

```

/****************** 发送一个字节 *****/
void TX1_write2buff(u8 dat)           //写入发送缓冲
{
    B_TX1_Busy = 1;                  //标志发送忙
    SBUF = dat;                     //发送一个字节
    while(B_TX1_Busy);             //等待发送完毕
}

//=====
// 函数: void PrintString1(u8 *puts)
// 描述: 串口1发送字符串函数。
// 参数: puts: 字符串指针。
// 返回: none.
// 版本: VER1.0
// 日期: 2014-11-28
// 备注:
//=====

void PrintString1(u8 *puts)           //发送一个字符串
{
    for (; *puts != 0; puts++)        //遇到停止符0 结束
    {
        TX1_write2buff(*puts);
    }
}

//=====
// 函数: void    UART1_config(void)
// 描述: UART1 初始化函数。
// 参数: none.
// 返回: none.
// 版本: VER1.0
// 日期: 2014-11-28
// 备注:
//=====

void UART1_config(void)
{
    TR1 = 0;                         //SI BRT Use Timer1;
    AUXR &= ~0x01;                   //Timer1 set as 1T mode
    AUXR |=  (1<<6);              //Timer1 set As Timer
    TMOD &= ~(1<<6);              //Timer1_16bitAutoReload;
    TMOD &= ~0x30;                  //TH1 = (u8)((65536L-(MAIN_Fosc / 4) / Baudrate1) >> 8);
    TH1 = (u8)((65536L-(MAIN_Fosc / 4) / Baudrate1) >> 8);
    TL1 = (u8)(65536L-(MAIN_Fosc / 4) / Baudrate1);
    ET1 = 0;                         // 禁止 Timer1 中断
    INT_CLKO &= ~0x02;              // Timer1 不输出高速时钟
    TR1   = 1;                      // 运行 Timer1

    S1_USE_P30P31();    P3n_standard(0x03);      //切换到 P3.0 P3.1
    //S1_USE_P36P37();    P3n_standard(0xc0);      //切换到 P3.6 P3.7
    //S1_USE_P16P17();    P1n_standard(0xc0);      //切换到 P1.6 P1.7

    SCON = (SCON & 0x3f) / 0x40;          //UART1 模式 0x00: 同步移位输出,
    //                                0x40: 8 位数据 可变波特率
    //                                0x80: 9 位数据 固定波特率
    //                                0xc0: 9 位数据 可变波特率
    // PS   = 1;                      //高优先级中断
    // ES   = 1;                      //允许中断
    REN  = 1;                      //允许接收
}

```

```

    B_TX1_Busy = 0;
}

//=====
// 函数: void UART1_int (void) interrupt UART1_VECTOR
// 描述: UART1 中断函数。
// 参数: nine.
// 返回: none.
// 版本: VER1.0
// 日期: 2014-II-28
// 备注:
//=====

void UART1_int (void) interrupt 4
{
    if(RI)
    {
        RI = 0;
        cmd = SBUF;
    }

    if(TI)
    {
        TI = 0;
        B_TX1_Busy = 0;
    }
}

```

C 语言代码 (EEPROM.c)

```

// 测试工作频率为 11.0592MHz

// 本程序是 STC 系列的内置 EEPROM 读写程序。

#include "config.h"
#include "EEPROM.h"

u32 PassWord; //擦除 写入时需要的口令

//=====
// 函数: void ISP_Disable(void)
// 描述: 禁止访问ISP/IAP。
// 参数: non.
// 返回: non.
// 版本: V1.0, 2012-10-22
//=====

void DisableEEPROM(void)
{
    ISP_CONTR = 0; //禁止 ISP/IAP 操作
    IAP_TPS = 0;
    ISP_CMD = 0; //去除 ISP/IAP 命令
    ISP_TRIG = 0; //防止 ISP/IAP 命令误触发
    ISP_ADDRH = 0xff; //清 0 地址高字节
    ISP_ADDRL = 0xff; //清 0 地址低字节，指向非 EEPROM 区，防止误操作
}

//=====

// 函数: void EEPROM_read_n(u16 EE_address,u8 *DataAddress,u16 number)
// 描述: 从指定 EEPROM 首地址读出 n 个字节放指定的缓冲

```

```

// 参数: EE_address: 读出EEPROM 的首地址
//        DataAddress: 读出数据放缓冲的首地址
//        number:      读出的字节长度
// 返回: non.
// 版本: V1.0, 2012-10-22
//=====
void EEPROM_read_n(u16 EE_address,u8 *DataAddress,u16 number)
{
    if(PassWord == D_PASSWORD)                                //口令正确才会操作EEPROM
    {
        EA = 0;                                              //禁止中断
        ISP_CONTR = ISP_EN;                                  //允许ISP/IAP 操作
        IAP_TPS = (u8)(MAIN_Fosc / 1000000L);              //工作频率设置
        ISP_READ();                                         //送字节读命令, 命令不需改变时, 不需重新送命令
        do
        {
            ISP_ADDRH = EE_address / 256;                  //送地址高字节(地址需要改变时才需重新送地址)
            ISP_ADDRL = EE_address % 256;                  //送地址低字节
            if(PassWord == D_PASSWORD)                      //口令口令正确才触发操作
            {
                ISP_TRIG = 0x5A;                           //先送5AH, 再送A5H 到ISP/IAP 触发寄存器,
                ISP_TRIG = 0xA5;                           //每次都需要如此
                ISP_ADDRH = EE_address / 256;              //送完A5H 后, ISP/IAP 命令立即被触发启动
                ISP_ADDRL = EE_address % 256;              //CPU 等待IAP 完成后, 才会继续执行程序。
                _nop_();
                _nop_();
                _nop_();
                *DataAddress = ISP_DATA;                  //读出的数据送往
                EE_address++;
                DataAddress++;
            }
        }while(--number);

        DisableEEPROM();
        EA = 1;                                              //重新允许中断
    }
    PassWord = 0;                                         //清除口令
}

/***** 扇区擦除函数 *****/
//=====
// 函数: void EEPROM_SectorErase(u16 EE_address)
// 描述: 把指定地址的EEPROM 扇区擦除
// 参数: EE_address: 要擦除的扇区EEPROM 的地址
// 返回: non.
// 版本: V1.0, 2013-5-10
//=====
void EEPROM_SectorErase(u16 EE_address)
{
    if(PassWord == D_PASSWORD)                                //口令正确才会操作EEPROM
    {
        EA = 0;                                              //禁止中断
        //只有扇区擦除, 没有字节擦除, 512 字节/扇区。
        //扇区中任意一个字节地址都是扇区地址。
        ISP_ADDRH = EE_address / 256;                      //送扇区地址高字节(地址需要改变时才需重新送地址)
        ISP_ADDRL = EE_address % 256;                      //送扇区地址低字节
        ISP_CONTR = ISP_EN;                                 //允许ISP/IAP 操作
        IAP_TPS = (u8)(MAIN_Fosc / 1000000L);            //工作频率设置
        ISP_ERASE();                                       //送扇区擦除命令, 命令不需改变时, 不需重新送命令
        if(PassWord == D_PASSWORD)                          //口令口令正确才触发操作
    }
}

```

```

{
    ISP_TRIG = 0x5A;           //先送5AH，再送A5H到ISP/IAP 触发寄存器，
                                //每次都需要如此
    ISP_TRIG = 0xA5;           //送完A5H后，ISP/IAP 命令立即被触发启动
                                //CPU 等待IAP 完成后，才会继续执行程序。
    _nop_();
    _nop_();
    _nop_();
    DisableEEPROM();
    EA = 1;                   //重新允许中断
}
PassWord = 0;               //清除口令
}

//=====
// 函数: void EEPROM_write_n(u16 EE_address,u8 *DataAddress,u16 number)
// 描述: 把缓冲的n个字节写入指定首地址的EEPROM.
// 参数: EE_address: 写入EEPROM 的首地址.
//        DataAddress: 写入源数据的缓冲的首地址.
//        number:      写入的字节长度.
// 返回: non.
// 版本: V1.0, 2012-10-22
//=====

void EEPROM_write_n(u16 EE_address,u8 *DataAddress,u16 number)
{
    if(PassWord == D_PASSWORD)           //口令正确才会操作EEPROM
    {
        EA = 0;                         //禁止中断

        ISP_CONTR = ISP_EN;             //允许ISP/IAP 操作
        IAP_TPS = (u8)(MAIN_Fosc / 1000000L); //工作频率设置
        ISP_WRITE();                   //送字节写命令，命令不需改变时，不需重新送命令
        do
        {
            ISP_ADDRH = EE_address / 256; //送地址高字节（地址需要改变时才需重新送地址）
            ISP_ADDRL = EE_address % 256; //送地址低字节
            ISP_DATA = *DataAddress;     //送数据到ISP_DATA，只有数据改变时才需重新送
            if(PassWord == D_PASSWORD)   //口令正确才触发操作
            {
                ISP_TRIG = 0x5A;          //先送5AH，再送A5H到ISP/IAP 触发寄存器，
                                //每次都需要如此
                ISP_TRIG = 0xA5;          //送完A5H后，ISP/IAP 命令立即被触发启动
                                //CPU 等待IAP 完成后，才会继续执行程序。
                _nop_();
                _nop_();
                _nop_();
                EE_address++;           //EE_address++;
                DataAddress++;          //DataAddress++;
            }while(--number);

            DisableEEPROM();
            EA = 1;                   //重新允许中断
        }
        PassWord = 0;                 //清除口令
    }
}

```

16 ADC 模数转换，内部 1.19V 参考信号源

产品线	ADC 分辨率	ADC 通道数
STC8G1K08 系列	10 位	15
STC8G1K08-8Pin 系列		
STC8G1K08A 系列	10 位	6
STC8G2K64S4 系列	10 位	15
STC8G2K64S2 系列	10 位	15
STC8G1K08T 系列	10 位	15
STC15H2K64S4 系列	10 位	15

STC8G 系列单片机内部集成了一个 10 位高速 A/D 转换器。ADC 的时钟频率为系统频率 2 分频再经过用户设置的分频系数进行再次分频（ADC 的工作时钟频率范围为 SYSclk/2/1 到 SYSclk/2/16）。

STC8G 系列的 ADC 最快速度：**12 位 ADC 为 800K (每秒进行 80 万次 ADC 转换) , 10 位 ADC 为 500K (每秒进行 50 万次 ADC 转换)**

ADC 转换结果的数据格式有两种：左对齐和右对齐。可方便用户程序进行读取和引用。

注意：ADC 的第 15 通道只能用于检测内部 1.19V 参考信号源，参考电压值出厂时校准为 1.19V，由于制造误差以及测量误差，导致实际的内部参考电压相比 1.19V，大约有±1%的误差。如果用户需要知道每一颗芯片的准确内部参考电压值，可外接精准参考电压，然后利用 ADC 的第 15 通道进行测量标定。

如果芯片有 ADC 的外部参考电源管脚 ADC_VRef+, 则一定不能浮空，必须接外部参考电源或者直接连到 VCC

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_EPWMT	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	

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
ADCTIM	ADC 时序控制寄存器	FEA8H	CSSETUP	CSHOLD[1:0]		SMPDUTY[4:0]					0010,1010

16.1.1 ADC 控制寄存器 (ADC_CONTR), PWM 触发 ADC 控制

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

ADC_POWER: ADC 电源控制位

0: 关闭 ADC 电源

1: 打开 ADC 电源。

建议进入空闲模式和掉电模式前将 ADC 电源关闭，以降低功耗

特别注意:

1、给 MCU 的 内部 ADC 模块电源打开后，需等待约 1ms，等 MCU 内部的 ADC 电源稳定后再让 ADC 工作；

2、适当加长对外部信号的采样时间，就是对 ADC 内部采样保持电容的充电或放电时间,时间够，内部才能和外部电势相等。

ADC_START: ADC 转换启动控制位。写入 1 后开始 ADC 转换，转换完成后硬件自动将此位清零。

0: 无影响。即使 ADC 已经开始转换工作，写 0 也不会停止 A/D 转换。

1: 开始 ADC 转换，转换完成后硬件自动将此位清零。

ADC_FLAG: ADC 转换结束标志位。当 ADC 完成一次转换后，硬件会自动将此位置 1，并向 CPU 提出中断请求。此标志位必须软件清零。

ADC_EPWMT: 使能 PWM 实时触发 ADC 功能。详情请参考 15 位增强型 PWM 章节

ADC_CHS[3:0]: ADC 模拟通道选择位

(注意: 被选择为 ADC 输入通道的 I/O 口, 必须设置 PxM0/PxM1 寄存器将 I/O 口模式设置为高阻输入模式。另外如果 MCU 进入掉电模式/时钟停振模式后, 仍需要使能 ADC 通道, 则需要设置 PxIE 寄存器关闭数字输入通道, 以防止外部模拟输入信号忽高忽低而产生额外的功耗)

(注意: 下面表格中红色字体的通道代表不同系列可能在不同端口上, 红色只是表示凸显)

(STC8G1K08/STC8G1K08T 系列)

ADC_CHS[3:0]	ADC 通道
0000	P1.0/ADC0
0001	P1.1/ADC1
0010	P1.2/ADC2
0011	P1.3/ADC3
0100	P1.4/ADC4
0101	P1.5/ADC5
0110	P1.6/ADC6
0111	P1.7/ADC7
1000	P3.0/ADC8
1001	P3.1/ADC9
1010	P3.2/ADC10
1011	P3.3/ADC11
1100	P3.4/ADC12
1101	P3.5/ADC13
1110	P3.6/ADC14
1111	测试内部 1.19V

(STC8G2K64S4/STC8G2K64S2/STC15H2K64S4 系列)

ADC_CHS[3:0]	ADC 通道
0000	P1.0/ADC0
0001	P1.1/ADC1
0010	P1.2/ADC2
0011	P1.3/ADC3
0100	P1.4/ADC4
0101	P1.5/ADC5
0110	P1.6/ADC6
0111	P1.7/ADC7
1000	P0.0/ADC8
1001	P0.1/ADC9
1010	P0.2/ADC10
1011	P0.3/ADC11
1100	P0.4/ADC12
1101	P0.5/ADC13
1110	P0.6/ADC14
1111	测试内部 1.19V

(STC8G1K08A 系列)

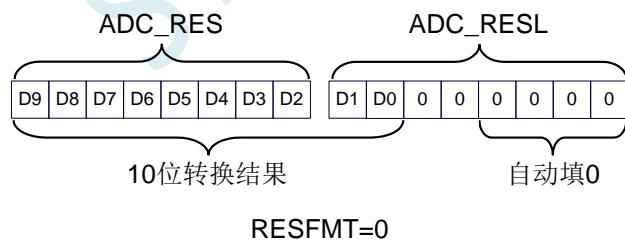
ADC_CHS[3:0]	ADC 通道
0000	P3.0/ADC0
0001	P3.1/ADC1
0010	P3.2/ADC2
0011	P3.3/ADC3
0100	P5.4/ADC4
0101	P5.5/ADC5
0110	无此通道
0111	无此通道
1000	无此通道
1001	无此通道
1010	无此通道
1011	无此通道
1100	无此通道
1101	无此通道
1110	无此通道
1111	测试内部 1.19V

16.1.2 ADC 配置寄存器 (ADCCFG)

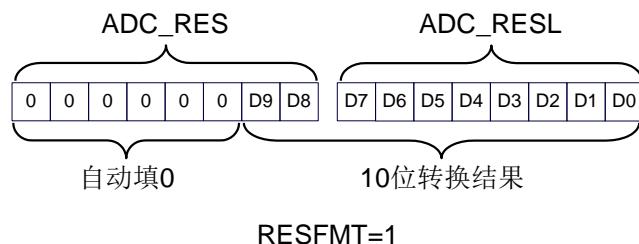
符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADCCFG	DEH	-	-	RESFMT	-	SPEED[3:0]			

RESFMT: ADC 转换结果格式控制位

0: 转换结果左对齐。ADC_RES 保存结果的高 8 位, ADC_RESL 保存结果的低 2 位。格式如下:



1: 转换结果右对齐。ADC_RES 保存结果的高 2 位, ADC_RESL 保存结果的低 8 位。格式如下:



SPEED[3:0]: 设置 ADC 工作时钟频率 { $F_{ADC} = SYSclk/2/(SPEED+1)$ }

SPEED[3:0]	给 ADC 的工作时钟频率
0000	SYSclk/2/1
0001	SYSclk/2/2
0010	SYSclk/2/3
...	...
1101	SYSclk/2/14
1110	SYSclk/2/15
1111	SYSclk/2/16

16.1.3 ADC 转换结果寄存器 (ADC_RES, ADC_RESL)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADC_RES	BDH								
ADC_RESL	BEH								

当 A/D 转换完成后，10 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。

16.1.4 ADC 时序控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADCTIM	FEA8H	CSSETUP	CSHOLD[1:0]		SMPDUTY[4:0]				

CSSETUP: ADC 通道选择时间控制 T_{setup}

CSSETUP	占用 ADC 工作时钟数
0	1 (默认值)
1	2

CSHOLD[1:0]: ADC 通道选择保持时间控制 T_{hold}

CSHOLD[1:0]	占用 ADC 工作时钟数
00	1
01	2 (默认值)
10	3
11	4

SMPDUTY[4:0]: ADC 模拟信号采样时间控制 T_{duty} (注意: SMPDUTY 一定不能设置小于 01010B)

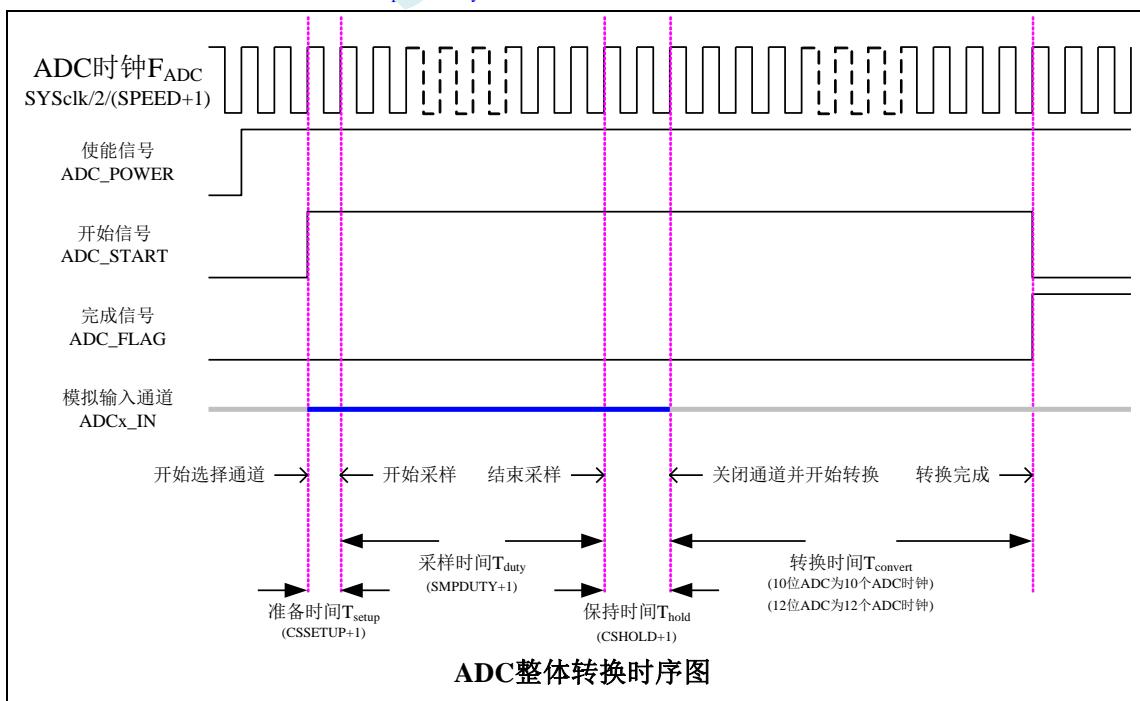
SMPDUTY[4:0]	占用 ADC 工作时钟数
00000	1
00001	2
...	...
01010	11 (默认值)
...	...
11110	31
11111	32

ADC 数模转换时间: T_{convert}

10 位 ADC 的转换时间为 10 个 ADC 工作时钟

12 位 ADC 的转换时间为 12 个 ADC 工作时钟

一个完整的 ADC 转换时间为: $T_{\text{setup}} + T_{\text{duty}} + T_{\text{hold}} + T_{\text{convert}}$, 如下图所示



16.2 ADC 相关计算公式

16.2.1 ADC 速度计算公式

ADC 的转换速度由 ADCCFG 寄存器中的 SPEED 和 ADCTIM 寄存器共同控制。转换速度的计算公式如下所示:

$$\text{10位ADC转换速度} = \frac{\text{MCU工作频率SYSclk}}{2 \times (\text{SPEED}[3:0] + 1) \times [(\text{CSSETUP} + 1) + (\text{CSHOLD} + 1) + (\text{SMPDUTY} + 1) + 10]}$$

$$\text{12位ADC转换速度} = \frac{\text{MCU工作频率SYSclk}}{2 \times (\text{SPEED}[3:0] + 1) \times [(\text{CSSETUP} + 1) + (\text{CSHOLD} + 1) + (\text{SMPDUTY} + 1) + 12]}$$

注意:

- 10 位 ADC 的速度不能高于 500KHz
- 12 位 ADC 的速度不能高于 800KHz
- SMPDUTY 的值不能小于 10, 建议设置为 15
- CSSETUP 可使用上电默认值 0
- CHOLD 可使用上电默认值 1 (ADCTIM 建议设置为 3FH)

16.2.2 ADC 转换结果计算公式

$$\text{10位ADC转换结果} = 1024 \times \frac{\text{ADC被转换通道的输入电压} V_{in}}{\text{MCU工作电压} V_{cc}} \quad (\text{无独立ADC_Vref+管脚})$$

$$\text{10位ADC转换结果} = 1024 \times \frac{\text{ADC被转换通道的输入电压} V_{in}}{\text{ADC外部参考源的电压}} \quad (\text{有独立ADC_Vref+管脚})$$

$$\text{12位ADC转换结果} = 4096 \times \frac{\text{ADC被转换通道的输入电压} V_{in}}{\text{MCU工作电压} V_{cc}} \quad (\text{无独立ADC_Vref+管脚})$$

$$\text{12位ADC转换结果} = 4096 \times \frac{\text{ADC被转换通道的输入电压} V_{in}}{\text{ADC外部参考源的电压}} \quad (\text{有独立ADC_Vref+管脚})$$

16.2.3 反推 ADC 输入电压计算公式

$$\text{ADC被转换通道的输入电压} V_{in} = \text{MCU工作电压} V_{cc} \times \frac{\text{10位ADC转换结果}}{1024} \quad (\text{无独立ADC_Vref+管脚})$$

$$\text{ADC被转换通道的输入电压} V_{in} = \text{ADC外部参考源的电压} \times \frac{\text{10位ADC转换结果}}{1024} \quad (\text{有独立ADC_Vref+管脚})$$

$$\text{ADC被转换通道的输入电压} V_{in} = \text{MCU工作电压} V_{cc} \times \frac{\text{12位ADC转换结果}}{4096} \quad (\text{无独立ADC_Vref+管脚})$$

$$\text{ADC被转换通道的输入电压} V_{in} = \text{ADC外部参考源的电压} \times \frac{\text{12位ADC转换结果}}{4096} \quad (\text{有独立ADC_Vref+管脚})$$

16.2.4 反推工作电压计算公式

当需要使用 ADC 输入电压和 ADC 转换结果反推工作电压时, 若目标芯片无独立的 ADC_Vref+管脚, 则可直接测量并使用下面公式, 若目标芯片有独立 ADC_Vref+管脚时, 则必须将 ADC_Vref+管脚连接到 Vcc 管脚。

$$\text{MCU工作电压Vcc} = 1024 \times \frac{\text{ADC被转换通道的输入电压Vin}}{10\text{位ADC转换结果}}$$

$$\text{MCU工作电压Vcc} = 4096 \times \frac{\text{ADC被转换通道的输入电压Vin}}{12\text{位ADC转换结果}}$$

16.3 10 位 ADC 静态特性

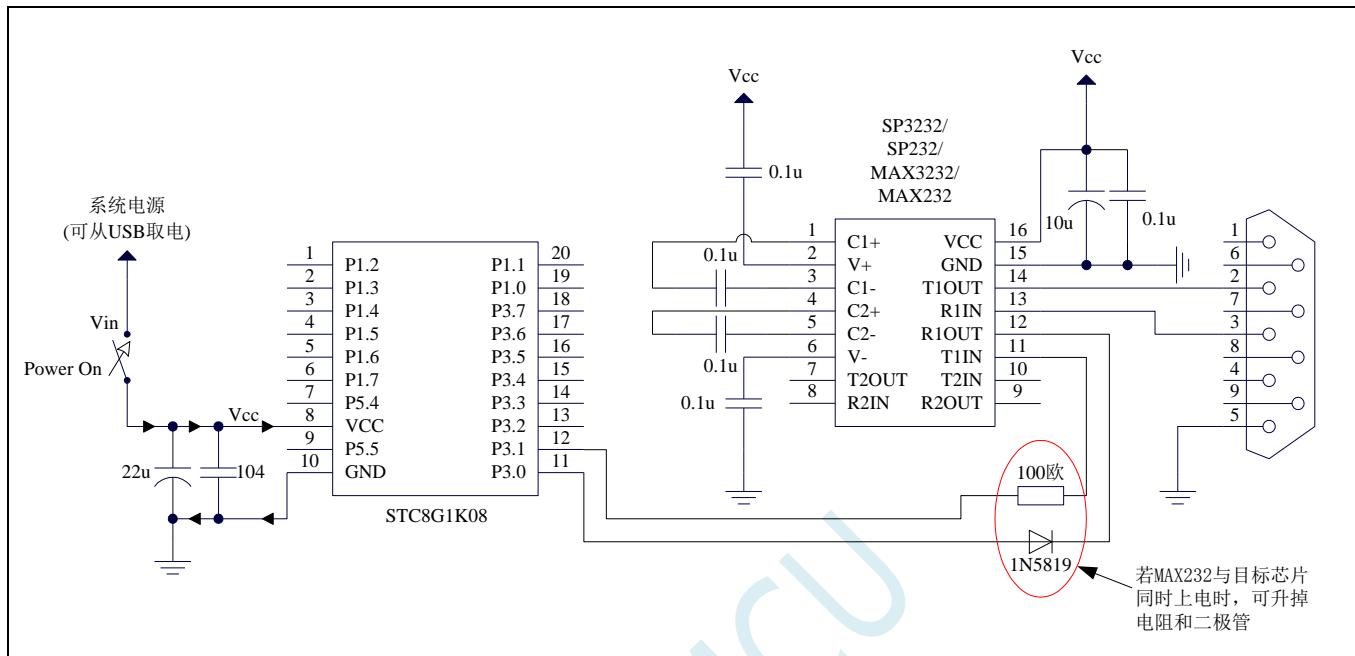
符号	描述	最小值	典型值	最大值	单位
RES	分辨率	-	10	-	Bits
E _T	整体误差	-	1.3	3	LSB
E _O	偏移误差	-	0.3	1	LSB
E _G	增益误差	-	0	1	LSB
E _D	微分非线性误差	-	0.7	1.5	LSB
E _I	积分非线性误差	-	1	2	LSB
R _{AIN}	通道等效电阻	-	∞	-	ohm
R _{ESD}	采样保持电容前串接的抗静电电阻	-	700	-	ohm
C _{ADC}	内部采样保持电容	-	16.5	-	pF

16.4 12 位 ADC 静态特性

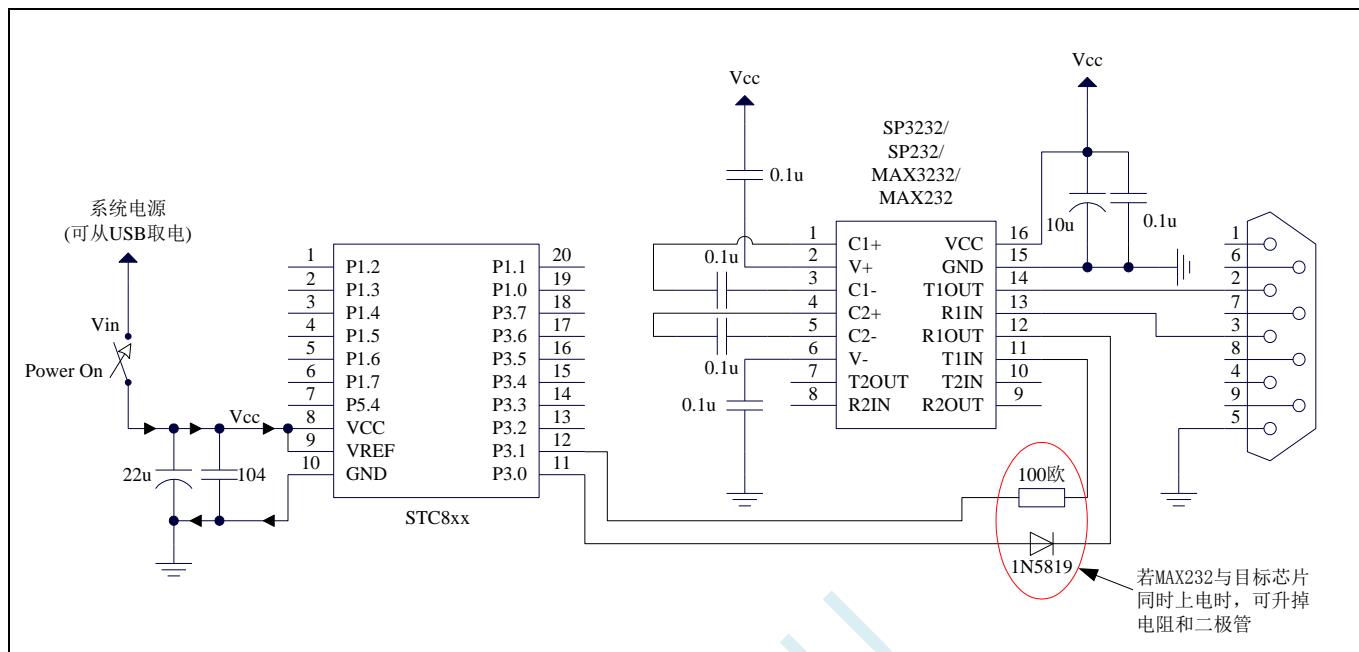
符号	描述	最小值	典型值	最大值	单位
RES	分辨率	-	12	-	Bits
E _T	整体误差	-	0.5	1	LSB
E _O	偏移误差	-	-0.1	1	LSB
E _G	增益误差	-	0	1	LSB
E _D	微分非线性误差	-	0.7	1.5	LSB
E _I	积分非线性误差	-	1	2	LSB
R _{AIN}	通道等效电阻	-	∞	-	ohm
R _{ESD}	采样保持电容前串接的抗静电电阻	-	700	-	ohm
C _{ADC}	内部采样保持电容	-	16.5	-	pF

16.5 ADC 应用参考线路图

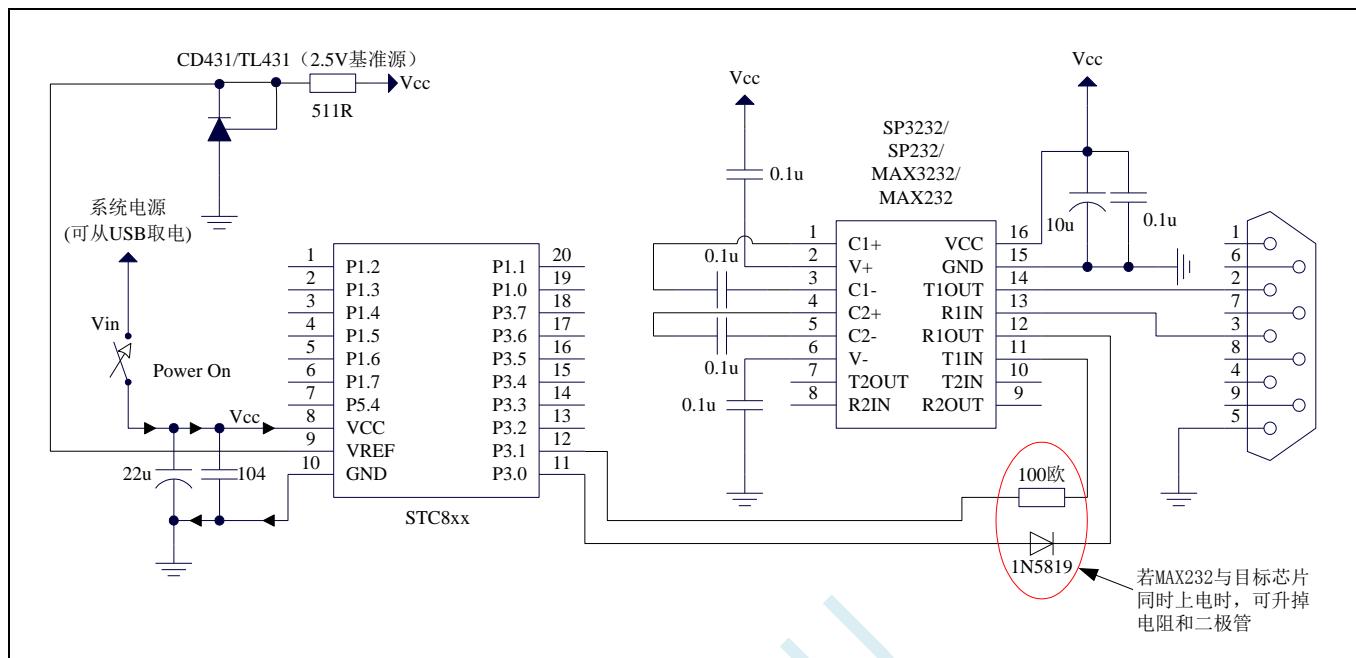
16.5.1 无独立 VREF 脚参考线路图



16.5.2 有独立 VREF 脚、一般精度 ADC 参考线路图



16.5.3 有独立 VREF 脚、高精度 ADC 参考线路图



16.6 范例程序

16.6.1 ADC 基本操作（查询方式）

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr ADC_CONTR = 0xbc;
sfr ADC_RES = 0xbd;
sfr ADC_RESL = 0xbe;
sfr ADCCFG = 0xde;

sfr P_SW2 = 0xba;
#define ADCTIM (*(unsigned char volatile xdata *)0xfea8)

sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P4M1 = 0xb3;
sfr P4M0 = 0xb4;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P1M0 = 0x00;                                // 设置 P1.0 为 ADC 口
    P1M1 = 0x01;
    P_SW2 |= 0x80;
    ADCTIM = 0x3f;                             // 设置 ADC 内部时序
    P_SW2 &= 0x7f;
    ADCCFG = 0x0f;                            // 设置 ADC 时钟为系统时钟/2/16
    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
}

```

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

P_SW2 DATA 0BAH
ADCTIM XDATA 0FEA8H

P0M1 DATA 093H
P0M0 DATA 094H
P1M1 DATA 091H
P1M0 DATA 092H
P2M1 DATA 095H
P2M0 DATA 096H
P3M1 DATA 0B1H
P3M0 DATA 0B2H
P4M1 DATA 0B3H
P4M0 DATA 0B4H
P5M1 DATA 0C9H
P5M0 DATA 0CAH

ORG 0000H
LJMP MAIN

ORG 0100H

MAIN:
MOV SP, #5FH
MOV P0M0, #00H
MOV P0M1, #00H
MOV P1M0, #00H
MOV P1M1, #00H
MOV P2M0, #00H
MOV P2M1, #00H
MOV P3M0, #00H
MOV P3M1, #00H
MOV P4M0, #00H
MOV P4M1, #00H
MOV P5M0, #00H
MOV P5M1, #00H

MOV P1M0,#00H           ; 设置 P1.0 为ADC 口
MOV P1M1,#01H
MOV P_SW2,#80H
MOV DPTR,#ADCTIM        ; 设置 ADC 内部时序
MOV A,#3FH

```

<i>MOVX</i>	<i>@DPTR,A</i>	
<i>MOV</i>	<i>P_SW2,#00H</i>	
<i>MOV</i>	<i>ADCCFG,#0FH</i>	; 设置 ADC 时钟为系统时钟/2/16
<i>MOV</i>	<i>ADC_CONTR,#80H</i>	; 使能 ADC 模块
<i>LOOP:</i>		
<i>ORL</i>	<i>ADC_CONTR,#40H</i>	; 启动 AD 转换
<i>NOP</i>		
<i>NOP</i>		
<i>MOV</i>	<i>A,ADC_CONTR</i>	; 查询 ADC 完成标志
<i>JNB</i>	<i>ACC.5,\$-2</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>SJMP</i>	<i>LOOP</i>	
<i>END</i>		

16.6.2 ADC 基本操作（中断方式）

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr ADC_CONTR = 0xbc;
sfr ADC_RES = 0xbd;
sfr ADC_RESL = 0xbe;
sfr ADCCFG = 0xde;

sfr P_SW2 = 0xba;
#define ADCTIM (*(unsigned char volatile xdata *)0xfea8)

sbit EADC = IE^5;

sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P4M1 = 0xb3;
sfr P4M0 = 0xb4;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;

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()
```

```

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

    P1M0 = 0x00;                                //设置P1.0 为ADC 口
    P1M1 = 0x01;
    P_SW2 /= 0x80;
    ADCTIM = 0x3f;                             //设置ADC 内部时序
    P_SW2 &= 0x7f;
    ADCCFG = 0x0f;                            //设置ADC 时钟为系统时钟/2/16
    ADC_CONTR = 0x80;                          //使能ADC 模块
    EADC = 1;                                 //使能ADC 中断
    EA = 1;                                   //启动AD 转换
    ADC_CONTR /= 0x40;                         //启动AD 转换

    while (1);
}

```

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
P_SW2	DATA	0BAH
ADCTIM	XDATA	0FEA8H
EADC	BIT	IE.5
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH
ORG		0000H
LJMP		MAIN
ORG		002BH
LJMP		ADCISR

ORG	0100H
ADCISR:	
ANL	ADC_CONTR,#NOT 20H ;清完成标志
MOV	P2,ADC_RES ;读取 ADC 结果
ORL	ADC_CONTR,#40H ;继续 AD 转换
RETI	
MAIN:	
MOV	SP, #5FH
MOV	P0M0, #00H
MOV	P0M1, #00H
MOV	P1M0, #00H
MOV	P1M1, #00H
MOV	P2M0, #00H
MOV	P2M1, #00H
MOV	P3M0, #00H
MOV	P3M1, #00H
MOV	P4M0, #00H
MOV	P4M1, #00H
MOV	P5M0, #00H
MOV	P5M1, #00H
MOV	P1M0,#00H ;设置 P1.0 为 ADC 口
MOV	P1M1,#01H
MOV	P_SW2,#80H
MOV	DPTR,#ADCTIM ;设置 ADC 内部时序
MOV	A,#3FH
MOVX	@DPTR,A
MOV	P_SW2,#00H
MOV	ADCCFG,#0FH ;设置 ADC 时钟为系统时钟/2/16
MOV	ADC_CONTR,#80H ;使能 ADC 模块
SETB	EADC ;使能 ADC 中断
SETB	EA
ORL	ADC_CONTR,#40H ;启动 AD 转换
SJMP	\$
END	

16.6.3 格式化 ADC 转换结果

C 语言代码

//测试工作频率为 11.0592MHz

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

sfr    ADC_CONTR = 0xbc;
sfr    ADC_RES  = 0xbd;
sfr    ADC_RESL = 0xbe;
sfr    ADCCFG   = 0xde;

sfr    P_SW2    = 0xba;
#define ADCTIM  (*(unsigned char volatile xdata *)0xfea8)
```

```

sfr P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P1M0 = 0x00;                                //设置P1.0 为ADC 口
    P1M1 = 0x01;
    P_SW2 /= 0x80;
    ADCTIM = 0x3f;
    P_SW2 &= 0x7f;
    ADCCFG = 0x0f;
    ADC_CONTR = 0x80;
    ADC_CONTR /= 0x40;
    _nop_();
    _nop_();
    while (!(ADC_CONTR & 0x20));
    ADC_CONTR &= ~0x20;

    ADCCFG = 0x00;
    ACC = ADC_RES;
    B = ADC_RESL;

//    ADCCFG = 0x20;                            //设置结果右对齐
//    ACC = ADC_RES;                           //A[1:0]存储ADC 的10 位结果的高2 位,A[7:2]为0
//    B = ADC_RESL;                           //B[7:6]存储ADC 的10 位结果的低2 位,B[5:0]为0

    while (1);
}

```

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

<i>ADCCFG</i>	<i>DATA</i>	<i>0DEH</i>
<i>P_SW2</i>	<i>DATA</i>	<i>0BAH</i>
<i>ADCTIM</i>	<i>XDATA</i>	<i>0FEA8H</i>
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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, #5FH</i>
	<i>MOV</i>	<i>P0M0, #00H</i>
	<i>MOV</i>	<i>P0M1, #00H</i>
	<i>MOV</i>	<i>P1M0, #00H</i>
	<i>MOV</i>	<i>P1M1, #00H</i>
	<i>MOV</i>	<i>P2M0, #00H</i>
	<i>MOV</i>	<i>P2M1, #00H</i>
	<i>MOV</i>	<i>P3M0, #00H</i>
	<i>MOV</i>	<i>P3M1, #00H</i>
	<i>MOV</i>	<i>P4M0, #00H</i>
	<i>MOV</i>	<i>P4M1, #00H</i>
	<i>MOV</i>	<i>P5M0, #00H</i>
	<i>MOV</i>	<i>P5M1, #00H</i>
	<i>MOV</i>	<i>P1M0,#00H</i> ; 设置 P1.0 为 ADC 口
	<i>MOV</i>	<i>P1M1,#01H</i>
	<i>MOV</i>	<i>P_SW2,#80H</i>
	<i>MOV</i>	<i>DPTR,#ADCTIM</i> ; 设置 ADC 内部时序
	<i>MOV</i>	<i>A,#3FH</i>
	<i>MOVX</i>	<i>@DPTR,A</i>
	<i>MOV</i>	<i>P_SW2,#00H</i>
	<i>MOV</i>	<i>ADCCFG,#0FH</i> ; 设置 ADC 时钟为系统时钟/2/16
	<i>MOV</i>	<i>ADC_CONTR,#80H</i> ; 使能 ADC 模块
	<i>ORL</i>	<i>ADC_CONTR,#40H</i> ; 启动 AD 转换
	<i>NOP</i>	
	<i>NOP</i>	
	<i>MOV</i>	<i>A,ADC_CONTR</i> ; 查询 ADC 完成标志
	<i>JNB</i>	<i>ACC.5,\$-2</i>
	<i>ANL</i>	<i>ADC_CONTR,#NOT 20H</i> ; 清完成标志
	<i>MOV</i>	<i>ADCCFG,#00H</i> ; 设置结果左对齐
	<i>MOV</i>	<i>A,ADC_RES</i> ; A 存储 ADC 的 10 位结果的高 8 位
	<i>MOV</i>	<i>B,ADC_RESL</i> ; B[7:6] 存储 ADC 的 10 位结果的低 2 位, B[5:0] 为 0
;	<i>MOV</i>	<i>ADCCFG,#20H</i> ; 设置结果右对齐

```

;           MOV      A,ADC_RES
;           MOV      B,ADC_RESL
;A[3:0]存储ADC 的10位结果的高2位,A[7:2]为0
;B 存储ADC 的10位结果的低8位

SJMP     $

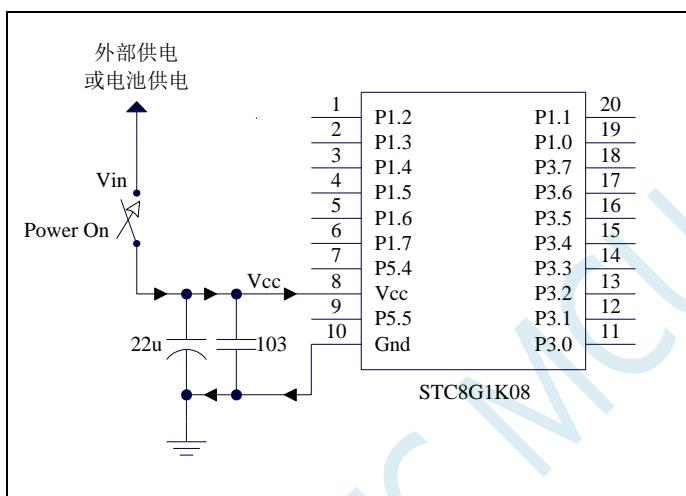
END

```

16.6.4 利用 ADC 第 15 通道测量外部电压或电池电压

STC8G 系列 ADC 的第 15 通道用于测量内部参考信号源, 由于内部参考信号源很稳定, 约为 1.19V, 且不会随芯片的工作电压的改变而变化, 所以可以通过测量内部 1.19V 参考信号源, 然后通过 ADC 的值便可反推出外部电压或外部电池电压。

下图为参考线路图:



C 语言代码

//测试工作频率为11.0592MHz

```

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

#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    P_SW2     = 0xba;
#define ADCTIM   (*(unsigned char volatile xdata *)0xfea8)

sfr    P0M1      = 0x93;
sfr    P0M0      = 0x94;
sfr    P1M1      = 0x91;
sfr    P1M0      = 0x92;
sfr    P2M1      = 0x95;

```

```
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

int *BGV;           //内部1.19V 参考信号源值存放在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;
    TLI = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    busy = 0;
}

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

void ADCInit()
{
    P_SW2 |= 0x80;           //设置ADC 内部时序
    ADCTIM = 0x3f;
    P_SW2 &= 0x7f;

    ADCCFG = 0x2f;           //设置ADC 时钟为系统时钟2/16
    ADC_CONTR = 0x8f;         //使能ADC 模块,并选择第15 通道
}

int ADCRead()
{
    int res;
```

```

ADC_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;

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

    BGV = (int idata *)0xef;                         //ADC 初始化
    ADCInit();                                       //串口初始化
    UartInit();

    ES = I;
    EA = I;

//    ADCRead();                                     //前两个数据建议丢弃
//    ADCRead();

    res = 0;
    for (i=0; i<8; i++)
    {
        res += ADCRead();                           //读取 8 次数据
    }
    res >>= 3;                                      //取平均值

    vcc = (int)(1024L * *BGV / res);               //10 位ADC 算法计算 VREF 管脚电压,即电池电压
//    vcc = (int)(4096L * *BGV / res);              //12 位ADC 算法计算 VREF 管脚电压,即电池电压
//    注意,此电压的单位为毫伏(mV)
    UartSend(vcc >> 8);                          //输出电压值到串口
    UartSend(vcc);

    while (1);
}

```

上面的方法是使用 ADC 的第 15 通道反推外部电池电压的。在 ADC 测量范围内，ADC 的外部测量电压与 ADC 的测量值是成正比例的，所以也可以使用 ADC 的第 15 通道反推外部通道输入电压，假设当前已获取了内部参考信号源电压为 BGV，内部参考信号源的 ADC 测量值为 res_{bg}，外部通道输入电压

的 ADC 测量值为 res_x , 则外部通道输入电压 $V_x = BGV / res_{bg} * res_x$;

16.6.5 ADC 做电容感应触摸按键

按键是电路最常用的零件之一, 是人机界面重要的输入方式, 我们最熟悉的是机械式按键, 但是机械按键有一个缺点(特别是便宜的按键), 触点有寿命, 很容易出现接触不良而失效。而非接触的按键则没有机械触点, 寿命长, 使用方便。

非接触的按键有多种方案, 而电容感应按键则是低成本的方案, 多年前一般是使用专门的 IC 来实现, 随着 MCU 功能的加强, 以及广大用户的实践经验, 直接使用 MCU 来做电容感应按键的技术已经成熟, 其中最典型最可靠的是使用 ADC 做的方案。

本文档详述使用 STC 带 ADC 的系列 MCU 做的方案, 可以使用任何带 ADC 功能的 MCU 来实现。下面前 3 个图是用得最多的方式, 原理都一样, 本文使用第 2 个图。

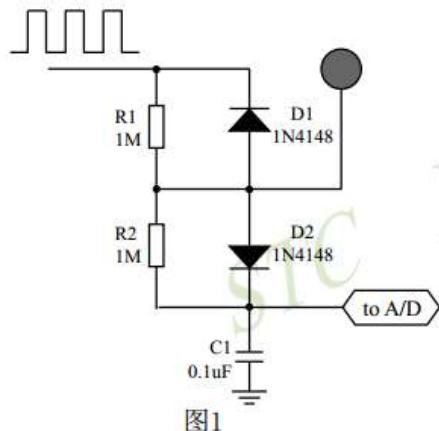


图1

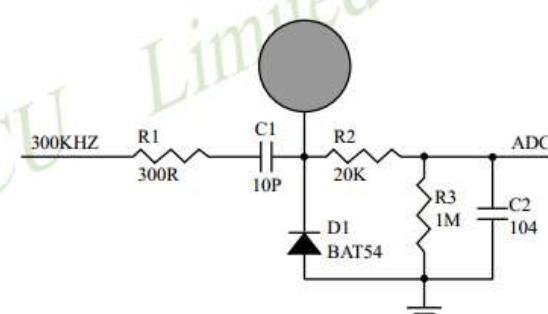


图2

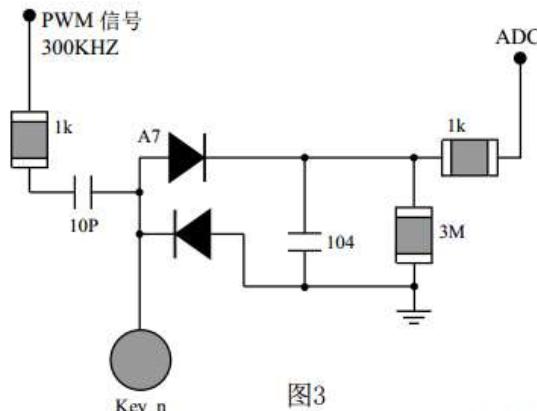


图3

电容感应按键取样电路

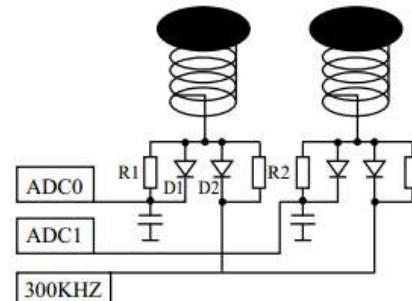
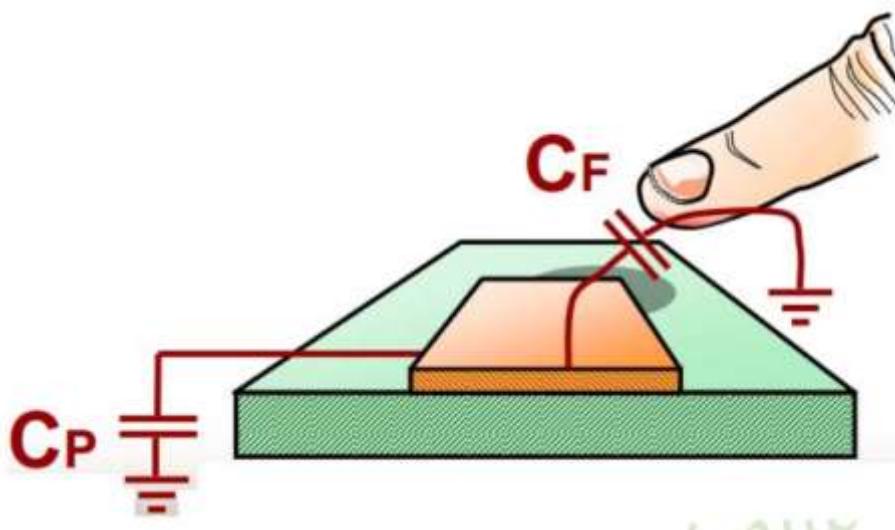
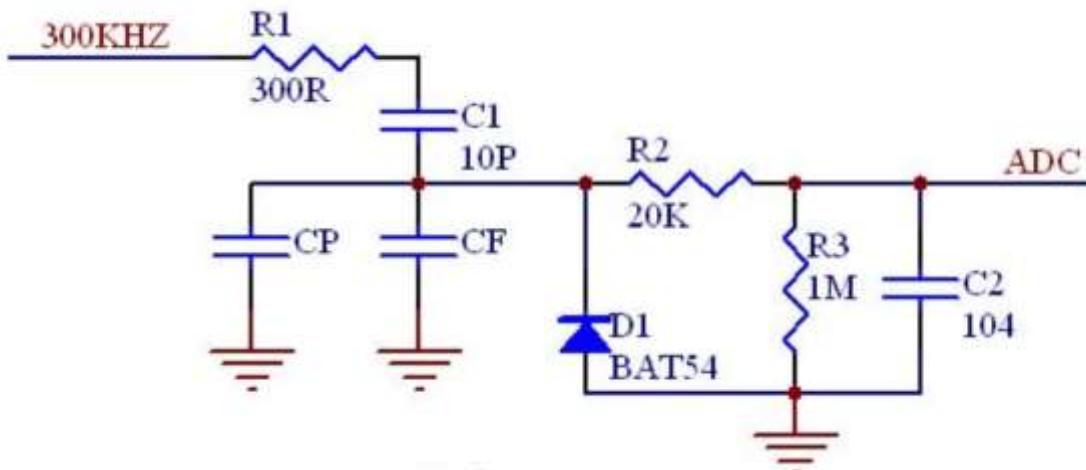


图4 加了感应弹簧

一般实际应用时, 都使用图 4 所示的感应弹簧来加大手指按下的面积。感应弹簧等效一块对地的金属板, 对地有一个电容 CP , 而手指按下后, 则再并联一个对地的电容 CF , 如下图所示。



下面为电路图的说明, CP 为金属板和分布电容, CF 为手指电容, 并联在一起与 C1 对输入的 300KHZ 方波进行分压, 经过 D1 整流, R2、C2 滤波后送 ADC, 当手指压上去后, 送去 ADC 的电压降低, 程序就可以检测出按键动作。



C 语言代码

//测试工作频率为24MHz

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

#define MAIN_Fosc 24000000UL           //定义主时钟
#define Timer0_Reload (65536UL -(MAIN_Fosc / 600000)) //Timer 0 重装值, 对应300KHZ

typedef unsigned char u8;
typedef unsigned int u16;
typedef unsigned long u32;

sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
```

```

sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

sfr ADC_CONTR = 0xBC;           //带AD 系列
sfr ADC_RES   = 0xBD;           //带AD 系列
sfr ADC_RESL  = 0xBE;           //带AD 系列
sfr AUXR     = 0x8E;
sfr AUXR2    = 0x8F;

#define CHANNEL 8                //ADC 通道数
#define ADC_90T (3<<5)          //ADC 时间 90T
#define ADC_180T (2<<5)          //ADC 时间 180T
#define ADC_360T (1<<5)          //ADC 时间 360T
#define ADC_540T 0                 //ADC 时间 540T
#define ADC_FLAG (1<<4)           //软件清0
#define ADC_START (1<<3)          //自动清0

sbit P_LED7    = P2^7;
sbit P_LED6    = P2^6;
sbit P_LED5    = P2^5;
sbit P_LED4    = P2^4;
sbit P_LED3    = P2^3;
sbit P_LED2    = P2^2;
sbit P_LED1    = P2^1;
sbit P_LED0    = P2^0;

u16 idata adc[CHANNEL];          //当前ADC 值
u16 idata adc_prev[CHANNEL];     //上一个ADC 值
u16 idata TouchZero[CHANNEL];    //0 点ADC 值
u8 idata TouchZeroCnt[CHANNEL];  //0 点自动跟踪计数
u8 cnt_250ms;

void delay_ms(u8 ms);
void ADC_init(void);
u16 Get_ADC10bitResult(u8 channel);
void AutoZero(void);
u8 check_adc(u8 index);
void ShowLED(void);

void main(void)
{
    u8 i;

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

```

```

delay_ms(50);
ET0 = 0;                                //初始化 Timer0 输出一个 300KHZ 时钟
TR0 = 0;
AUXR |= 0x80;
AUXR2 |= 0x01;
TMOD = 0;                                //Timer0 set as 1T mode
//允许输出时钟
TH0 = (u8)(Timer0_Reload >> 8);
TL0 = (u8)Timer0_Reload;
TR0 = 1;                                //Timer0 set as Timer, 16 bits Auto Reload.
ADC_init();                               //ADC 初始化
delay_ms(50);                            //延时 50ms
for (i=0; i< CHANNEL; i++)
{
    adc_prev[i] = 1023;
    TouchZero[i] = 1023;
    TouchZeroCnt[i] = 0;
}
cnt_250ms = 0;
while (1)
{
    delay_ms(50);                         //每隔 50ms 处理一次按键
    ShowLED();
    if (++cnt_250ms >= 5)
    {
        cnt_250ms = 0;
        AutoZero();                      //每隔 250ms 处理一次 0 点自动跟踪
    }
}
}

void delay_ms(u8 ms)
{
    unsigned int i;

    do
    {
        i = MAIN_Fosc / 10000;
        while(--i);
    } while(--ms);
}

void ADC_init(void)
{
    PIM0 = 0x00;                           //8 路 ADC
    PIM1 = 0xff;
    ADC_CONTR = 0x80;                      //允许 ADC
}
u16 Get_ADC10bitResult(u8 channel)
{
    ADC_RES = 0;
    ADC_RESL = 0;
    ADC_CONTR = 0x80 | ADC_90T | ADC_START | channel; //触发 ADC
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    while((ADC_CONTR & ADC_FLAG) == 0);           //等待 ADC 转换结束
}

```

```

ADC_CONTR = 0x80;                                //清除标志
return(((u16)ADC_RES << 2) | ((u16)ADC_RESL & 3)); //返回ADC 结果
}

void AutoZero(void)                                //250ms 调用一次
{
    u8 i;
    u16 j,k;

    for(i=0; i< CHANNEL; i++)                      //处理8 个通道
    {
        j = adc[i];
        k = j - adc_prev[i];
        F0 = 0;
        if(k & 0x8000) F0 = 1, k = 0 - k;
        if(k >= 20)                                         //变化比较大
        {
            TouchZeroCnt[i] = 0;
            if(F0) TouchZero[i] = j;
        }
        else
        {
            if(++TouchZeroCnt[i] >= 20)                  //连续检测到小变化 20 次/4 = 5 秒
            {
                TouchZeroCnt[i] = 0;
                TouchZero[i] = adc_prev[i];                //变化缓慢的值作为0 点
            }
        }
        adc_prev[i] = j;
    }
}

u8 check_adc(u8 index)                            //获取触摸信息函数 50ms 调用1 次
{
    u16 delta;

    adc[index] = 1023 - Get_ADC10bitResult(index);
    if(adc[index] < TouchZero[index]) return 0;
    delta = adc[index] - TouchZero[index];
    if(delta >= 40) return 1;                         //键按下
    if(delta <= 20) return 0;                          //键释放
    return 2;                                         //保持原状态
}

void ShowLED(void)
{
    u8 i;

    i = check_adc(0);
    if(i == 0) P_LED0 = 1;                            //指示灯灭
    if(i == 1) P_LED0 = 0;                            //指示灯亮
    i = check_adc(1);
    if(i == 0) P_LED1 = 1;                            //指示灯灭
    if(i == 1) P_LED1 = 0;                            //指示灯亮
    i = check_adc(2);
    if(i == 0) P_LED2 = 1;                            //指示灯灭
    if(i == 1) P_LED2 = 0;                            //指示灯亮
}

```

```

i = check_adc(3);
if(i == 0) P_LED3 = 1;                                //指示灯灭
if(i == 1) P_LED3 = 0;                                //指示灯亮
i = check_adc(4);
if(i == 0) P_LED4 = 1;                                //指示灯灭
if(i == 1) P_LED4 = 0;                                //指示灯亮
i = check_adc(5);
if(i == 0) P_LED5 = 1;                                //指示灯灭
if(i == 1) P_LED5 = 0;                                //指示灯亮
i = check_adc(6);
if(i == 0) P_LED6 = 1;                                //指示灯灭
if(i == 1) P_LED6 = 0;                                //指示灯亮
i = check_adc(7);
if(i == 0) P_LED7 = 1;                                //指示灯灭
if(i == 1) P_LED7 = 0;                                //指示灯亮
}

```

汇编代码

;测试工作频率为 24MHz

<i>Fosc_KHZ</i>	<i>EQU</i>	24000	; 定义主时钟 KHZ
<i>Reload</i>	<i>EQU</i>	(65536 - <i>Fosc_KHZ</i> /600)	; Timer 0 重装值, 对应 300KHZ
<i>ADC_CONTR</i>	<i>DATA</i>	0xBC	; 带 AD 系列
<i>ADC_RES</i>	<i>DATA</i>	0xBD	; 带 AD 系列
<i>ADC_RESL</i>	<i>DATA</i>	0xBE	; 带 AD 系列
<i>AUXR</i>	<i>DATA</i>	0x8E	
<i>AUXR2</i>	<i>DATA</i>	0x8F	
<i>P0M1</i>	<i>DATA</i>	093H	
<i>P0M0</i>	<i>DATA</i>	094H	
<i>P1M1</i>	<i>DATA</i>	091H	
<i>P1M0</i>	<i>DATA</i>	092H	
<i>P2M1</i>	<i>DATA</i>	095H	
<i>P2M0</i>	<i>DATA</i>	096H	
<i>P3M1</i>	<i>DATA</i>	0B1H	
<i>P3M0</i>	<i>DATA</i>	0B2H	
<i>P4M1</i>	<i>DATA</i>	0B3H	
<i>P4M0</i>	<i>DATA</i>	0B4H	
<i>P5M1</i>	<i>DATA</i>	0C9H	
<i>P5M0</i>	<i>DATA</i>	0CAH	
<i>CHANNEL</i>	<i>EQU</i>	8	; ADC 通道数
<i>ADC_90T</i>	<i>EQU</i>	(3 SHL 5)	; ADC 时间 90T
<i>ADC_180T</i>	<i>EQU</i>	(2 SHL 5)	; ADC 时间 180T
<i>ADC_360T</i>	<i>EQU</i>	(1 SHL 5)	; ADC 时间 360T
<i>ADC_540T</i>	<i>EQU</i>	0	; ADC 时间 540T
<i>ADC_FLAG</i>	<i>EQU</i>	(1 SHL 4)	; 软件清 0
<i>ADC_START</i>	<i>EQU</i>	(1 SHL 3)	; 自动清 0
<i>P_LED7</i>	<i>BIT</i>	P2.7;	
<i>P_LED6</i>	<i>BIT</i>	P2.6;	
<i>P_LED5</i>	<i>BIT</i>	P2.5;	
<i>P_LED4</i>	<i>BIT</i>	P2.4;	
<i>P_LED3</i>	<i>BIT</i>	P2.3;	
<i>P_LED2</i>	<i>BIT</i>	P2.2;	
<i>P_LED1</i>	<i>BIT</i>	P2.1;	

```

P_LED0      BIT      P2.0;
adc         EQU      30H           ;当前ADC 值 30H~3FH,两字节一个值
adc_prev    EQU      40H           ;上一个ADC 值 40H~4FH,两字节一个值
TouchZero   EQU      50H           ;0 点ADC 值 50H~5FH,两字节一个值
TouchZeroCnt EQU      60H           ;0 点自动跟踪计数 60H~67H
cnt_250ms  DATA     68H

        ORG      0000H
        LJMP    MAIN

        ORG      0100H
MAIN:
        MOV      SP,#0D0H
        MOV      P0M0,#00H
        MOV      P0M1,#00H
        MOV      P1M0,#00H
        MOV      P1M1,#00H
        MOV      P2M0,#00H
        MOV      P2M1,#00H
        MOV      P3M0,#00H
        MOV      P3M1,#00H
        MOV      P4M0,#00H
        MOV      P4M1,#00H
        MOV      P5M0,#00H
        MOV      P5M1,#00H

        MOV      R7,#50
        LCALL   F_delay_ms
        CLR     ET0          ;初始化 Timer0 输出一个300KHZ 时钟
        CLR     TR0
        ORL     AUXR,#080H
        ORL     AUXR2,#01H
        MOV     TMOD,#0
        MOV     TH0,#HIGH Reload
        MOV     TL0,#LOW Reload
        SETB   TR0
        LCALL   F_ADC_init
        MOV     R7,#50
        LCALL   F_delay_ms
        MOV     R0,#adc_prev   ;初始化上一个ADC 值

L_Init_Loop1:
        MOV     @R0,#03H
        INC     R0
        MOV     @R0,#0FFH
        INC     R0
        MOV     A,R0
        CJNE   A,#(adc_prev + CHANNEL * 2),L_Init_Loop1
        MOV     R0,#TouchZero  ;初始化 0 点 ADC 值

L_Init_Loop2:
        MOV     @R0,#03H
        INC     R0
        MOV     @R0,#0FFH
        INC     R0
        MOV     A,R0
        CJNE   A,#(TouchZero+CHANNEL * 2),L_Init_Loop2
        MOV     R0,#TouchZeroCnt ;初始化自动跟踪计数值

L_Init_Loop3:
        MOV     @R0,#0
        INC     R0

```

```

MOV      A,R0
CJNE    A,#(TouchZeroCnt + CHANNEL),L_Init_Loop3
MOV      cnt_250ms,#5

L_MainLoop:
MOV      R7,#50          ;延时 50ms
LCALL   F_delay_ms
LCALL   F_ShowLED       ;处理一次触摸键值
DJNZ    cnt_250ms,L_MainLoop
MOV      cnt_250ms,#5      ;250ms 处理一次 0 点自动跟踪
LCALL   F_AutoZero      ;自动跟踪零点
SJMP   L_MainLoop

F_ADC_init:
MOV      P1M0,#00H        ;8 路 ADC
MOV      P1M1,#0FFH
MOV      ADC_CONTR,#080H   ;允许 ADC
RET

F_Get_ADC10bitResult:
MOV      ADC_RES,#0
MOV      ADC_RESL,#0
MOV      A,R7
ORL     A,#0E8H          ;触发 ADC
MOV      ADC_CONTR,A
NOP
NOP
NOP
NOP

L_10bitADC_Loop1:
MOV      A,ADC_CONTR
JNB    ACC.4,L_10bitADC_Loop1 ;等待 ADC 转换结束
MOV      ADC_CONTR,#080H   ;清除标志
MOV      A,ADC_RES
MOV      B,#04H
MUL    AB
MOV      R7,A
MOV      R6,B
MOV      A,ADC_RESL
ANL    A,#03H
ORL     A,R7
MOV      R7,A
RET

F_AutoZero:           ;250ms 调用一次
                           ;这是使用相邻 2 个采样的差的绝对值之和来检测。
CLR    A
MOV    R5,A

L_AutoZero_Loop:
MOV    A,R5
ADD   A,ACC
ADD   A,#LOW (adc)
MOV    R0,A
MOV    A,@R0
MOV    R6,A
INC   R0
MOV    A,@R0
MOV    R7,A
MOV    A,R5
ADD   A,ACC

```

```

ADD      A,#LOW (adc_prev+01H)
MOV      R0,A
CLR      C
MOV      A,R7
SUBB    A,@R0
MOV      R3,A
MOV      A,R6
DEC      R0
SUBB    A,@R0
MOV      R2,A
CLR      F0 ;按下
JNB     ACC.7,L_AutoZero_1
SETB    F0
CLR      C
CLR      A
SUBB    A,R3
MOV      R3,A
MOV      A,R3
CLR      A
SUBB    A,R2
MOV      R2,A

L_AutoZero_1:
CLR      C           ;计算 [R2 R3] - #20,if(k >= 20)
MOV      A,R3
SUBB    A,#20
MOV      A,R2
SUBB    A,#00H
JC      L_AutoZero_2
MOV      A,#LOW (TouchZeroCnt) ;[R2 R3],20,转
ADD      A,R5           ;如果变化比较大, 则清0 计数器 TouchZeroCnt[i] = 0;
MOV      R0,A
MOV      @R0,#0
JNB     F0,L_AutoZero_3
MOV      A,R5
ADD      A,ACC
ADD      A,#LOW (TouchZero)
MOV      R0,A
MOV      @R0,6
INC      R0
MOV      @R0,7
SJMP    L_AutoZero_3

L_AutoZero_2:          ;变化比较小, 则蠕动, 自动0点跟踪
                           ;连续检测到小变化 20 次4 = 5 秒.
MOV      A,#LOW (TouchZeroCnt)
ADD      A,R5
MOV      R0,A
INC      @R0
MOV      A,@R0
CLR      C
SUBB    A,#20
JC      L_AutoZero_3           ;if(TouchZeroCnt[i] < 20), 转
MOV      @R0,#0           ;TouchZeroCnt[i] = 0;
MOV      A,R5           ;变化缓慢的值作为0点
ADD      A,ACC
ADD      A,#LOW (adc_prev)
MOV      R0,A
MOV      A,@R0
MOV      R2,A
INC      R0

```

```

MOV      A,@R0
MOV      R3,A
MOV      A,R5
ADD      A,ACC
ADD      A,#LOW (TouchZero)
MOV      R0,A
MOV      @R0,2
INC      R0
MOV      @R0,3
L_AutoZero_3:                                ;保存采样值 adc_prev[i] = j;
MOV      A,R5
ADD      A,ACC
ADD      A,#LOW (adc_prev)
MOV      R0,A
MOV      @R0,6
INC      R0
MOV      @R0,7
INC      R5
MOV      A,R5
XRL      A,#08H
JZ       $+5H
LJMP    L_AutoZero_Loop
RET

```

F_check_adc: ;判断键按下或释放,有回差控制

```

MOV R4,7
LCALL F_Get_ADC10bitResult      ;返回的ADC 值在 [R6 R7]
CLR      C
MOV      A,#0FFH
SUBB   A,R7
MOV      R7,A
MOV      A,#03H
SUBB   A,R6
MOV      R6,A
MOV      A,R4                  ;保存 adc[index]
ADD      A,ACC
ADD      A,#LOW (adc)
MOV      R0,A
MOV      @R0,6
INC      R0
MOV      @R0,7
MOV      A,R4
ADD      A,ACC
ADD      A,#LOW (TouchZero+01H)
MOV      RI,A
MOV      A,R4
ADD      A,ACC
ADD      A,#LOW (adc)
MOV      R0,A
MOV      A,@R0
MOV      R6,A
INC      R0
MOV      A,@R0
CLR      C
SUBB   A,@RI                  ;计算 adc[index] - TouchZero[index]
MOV      A,R6
DEC      RI
SUBB   A,@RI
JNC    L_check_adc_1

```

MOV R7,#00H

RET

L_check_adc_1:

MOV A,R4

ADD A,ACC

ADD A,#LOW (TouchZero+01H)

MOV RI,A

MOV A,R4

ADD A,ACC

ADD A,#LOW (adc+01H)

MOV R0,A

CLR C

MOV A,@R0

SUBB A,@RI

MOV R7,A

DEC R0

MOV A,@R0

DEC RI

SUBB A,@RI

MOV R6,A

CLR C

MOV A,R7

SUBB A,#40

MOV A,R6

SUBB A,#00H

JC L_check_adc_2

;if(delta < 40),转

MOV R7,#1

;if(delta >= 40) return 1; //键按下 返回1

RET

L_check_adc_2:

SETB C

MOV A,R7

SUBB A,#20

MOV A,R6

SUBB A,#00H

JNC L_check_adc_3

MOV R7,#0

RET

L_check_adc_3:

MOV R7,#2

RET

F_ShowLED:

MOV R7,#0

LCALL F_check_adc

MOV A,R7

ANL A,#0FEH

JNZ L_QuitCheck0

MOV A,R7

MOV C,ACC.0

CPL C

MOV P_LED0,C

L_QuitCheck0:

MOV R7,#1

LCALL F_check_adc

MOV A,R7

ANL A,#0FEH

JNZ L_QuitCheck1

MOV A,R7

MOV C,ACC.0

<i>CPL</i>	<i>C</i>
<i>MOV</i>	<i>P_LED1,C</i>
<i>L_QuitCheck1:</i>	
<i>MOV</i>	<i>R7,#2</i>
<i>LCALL</i>	<i>F_check_adc</i>
<i>MOV</i>	<i>A,R7</i>
<i>ANL</i>	<i>A,#0FEH</i>
<i>JNZ</i>	<i>L_QuitCheck2</i>
<i>MOV</i>	<i>A,R7</i>
<i>MOV</i>	<i>C,ACC.0</i>
<i>CPL</i>	<i>C</i>
<i>MOV</i>	<i>P_LED2,C</i>
<i>L_QuitCheck2:</i>	
<i>MOV</i>	<i>R7,#3</i>
<i>LCALL</i>	<i>F_check_adc</i>
<i>MOV</i>	<i>A,R7</i>
<i>ANL</i>	<i>A,#0FEH</i>
<i>JNZ</i>	<i>L_QuitCheck3</i>
<i>MOV</i>	<i>A,R7</i>
<i>MOV</i>	<i>C,ACC.0</i>
<i>CPL</i>	<i>C</i>
<i>MOV</i>	<i>P_LED3,C</i>
<i>L_QuitCheck3:</i>	
<i>MOV</i>	<i>R7,#4</i>
<i>LCALL</i>	<i>F_check_adc</i>
<i>MOV</i>	<i>A,R7</i>
<i>ANL</i>	<i>A,#0FEH</i>
<i>JNZ</i>	<i>L_QuitCheck4</i>
<i>MOV</i>	<i>A,R7</i>
<i>MOV</i>	<i>C,ACC.0</i>
<i>CPL</i>	<i>C</i>
<i>MOV</i>	<i>P_LED4,C</i>
<i>L_QuitCheck4:</i>	
<i>MOV</i>	<i>R7,#5</i>
<i>LCALL</i>	<i>F_check_adc</i>
<i>MOV</i>	<i>A,R7</i>
<i>ANL</i>	<i>A,#0FEH</i>
<i>JNZ</i>	<i>L_QuitCheck5</i>
<i>MOV</i>	<i>A,R7</i>
<i>MOV</i>	<i>C,ACC.0</i>
<i>CPL</i>	<i>C</i>
<i>MOV</i>	<i>P_LED5,C</i>
<i>L_QuitCheck5:</i>	
<i>MOV</i>	<i>R7,#6</i>
<i>LCALL</i>	<i>F_check_adc</i>
<i>MOV</i>	<i>A,R7</i>
<i>ANL</i>	<i>A,#0FEH</i>
<i>JNZ</i>	<i>L_QuitCheck6</i>
<i>MOV</i>	<i>A,R7</i>
<i>MOV</i>	<i>C,ACC.0</i>
<i>CPL</i>	<i>C</i>
<i>MOV</i>	<i>P_LED6,C</i>
<i>L_QuitCheck6:</i>	
<i>MOV</i>	<i>R7,#7</i>
<i>LCALL</i>	<i>F_check_adc</i>
<i>MOV</i>	<i>A,R7</i>
<i>ANL</i>	<i>A,#0FEH</i>
<i>JNZ</i>	<i>L_QuitCheck7</i>
<i>MOV</i>	<i>A,R7</i>

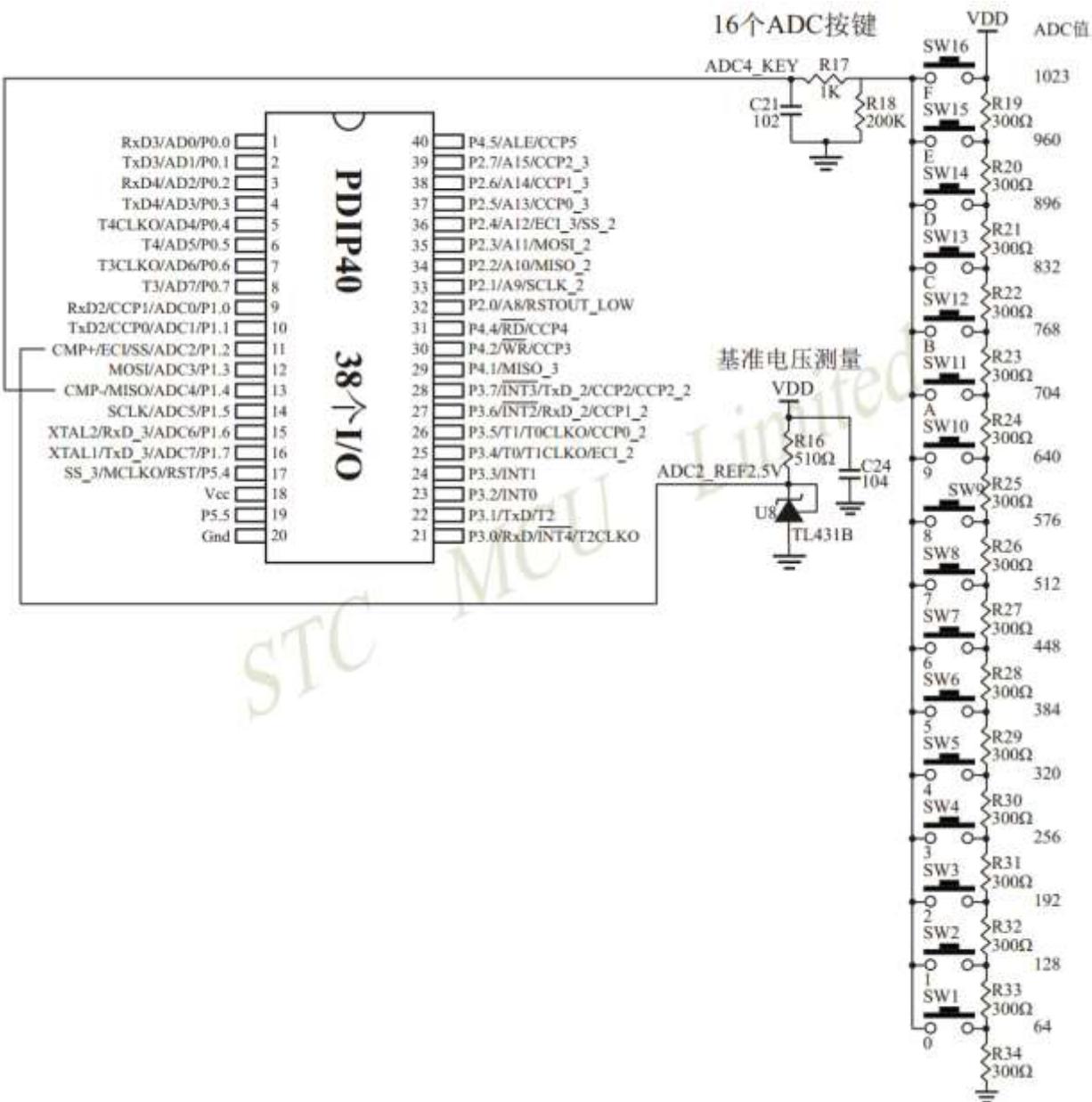
```
MOV      C,ACC.0
CPL      C
MOV      P_LED7,C
L_QuitCheck7:
RET

F_delay_ms:
PUSH    3
PUSH    4
L_delay_ms_1:
MOV      R3,#HIGH (Fosc_KHZ / 13)
MOV      R4,#LOW (Fosc_KHZ / 13)
L_delay_ms_2:
MOV      A,R4
DEC     R4
JNZ    L_delay_ms_3
DEC     R3
L_delay_ms_3:
DEC     A
ORL     A,R3
JNZ    L_delay_ms_2
DJNZ   R7,L_delay_ms_1
POP    4
POP    3
RET

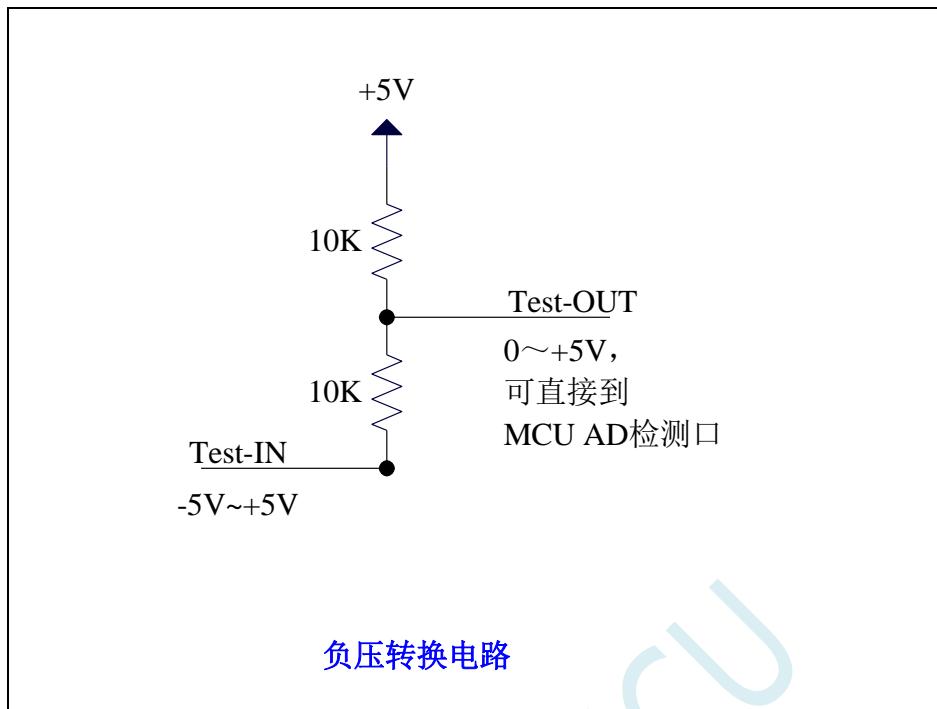
END
```

16.6.6 ADC 作按键扫描应用线路图

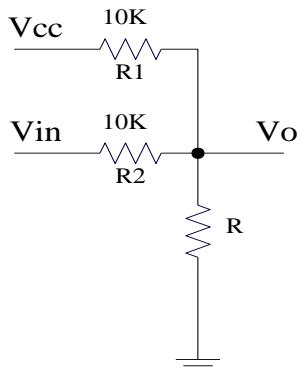
读 ADC 键的方法: 每隔 10ms 左右读一次 ADC 值, 并且保存最后 3 次的读数, 其变化比较小时再判断键。判断键有效时, 允许一定的偏差, 比如±16 个字的偏差。



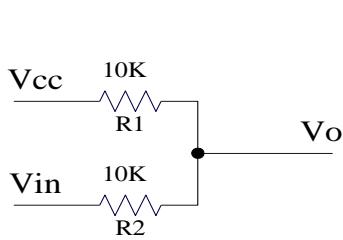
16.6.7 检测负电压参考线路图



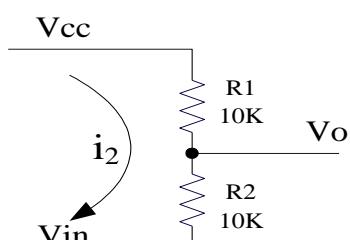
16.6.8 常用加法电路在 ADC 中的应用



常用加法电路



简化加法电路



变形成为分压电路形式

参照分压电路得到公式 1

$$\text{公式 1: } V_o = V_{in} + i_2 * R_2$$

$$\text{公式 2: } i_2 = (V_{cc} - V_{in}) / (R_1 + R_2) \quad \{ \text{条件: 流向 } V_o \text{ 的电流 } \approx 0 \}$$

将 $R_1=R_2$ 代入公式 2 得公式 3

$$\text{公式 3: } i_2 = (V_{cc} - V_{in}) / 2R_2$$

将公式 3 代入公式 1 得公式 4

$$\text{公式 4: } V_o = (V_{cc} + V_{in}) / 2$$

根据公式 4, 可以将以上电路看成加法电路。

在单片机的模数转换测量中, 要求被测电压大于 0 并且小于 VCC。如果被测电压小于 0V, 可以利用加法电路将被测电压提升到 0V 以上。此时对被测电压的变化范围有一定的要求:

把上述条件代入公式 4 可得到下面 2 式

$$(V_{cc} + V_{in}) / 2 > 0 \quad \text{即 } V_{in} > -V_{cc}$$

$$(V_{cc} + V_{in}) / 2 < V_{cc} \quad \text{即 } V_{in} < V_{cc}$$

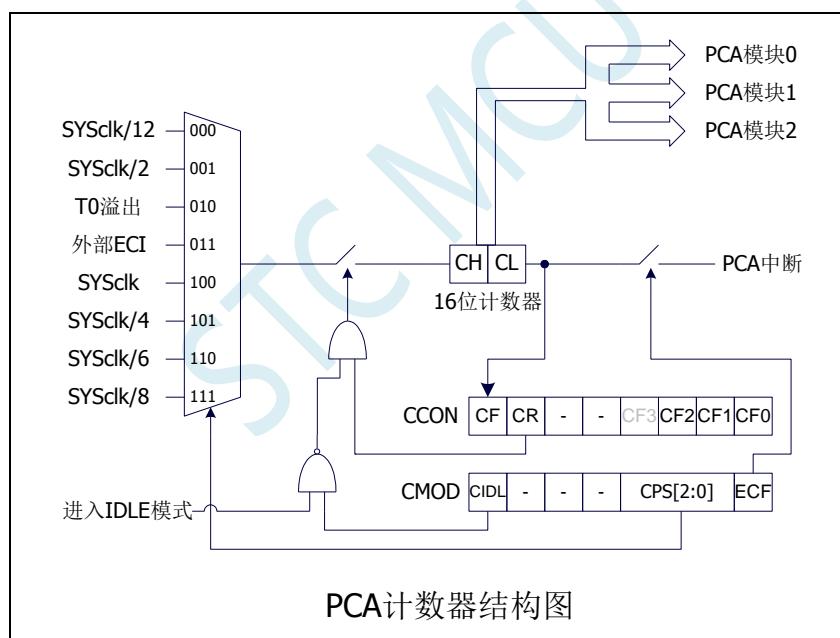
$$\text{上面 2 式可以合起来: } -V_{cc} < V_{in} < V_{cc}$$

17 PCA/CCP/PWM 应用

产品线	PCA
STC8G1K08 系列	●
STC8G1K08-8Pin 系列	
STC8G1K08A 系列	●
STC8G2K64S4 系列	●
STC8G2K64S2 系列	●
STC8G1K08T 系列	●
STC15H2K64S4 系列	●

STC8G 系列单片机内部集成了 3 组可编程计数器阵列 (PCA/CCP/PWM) 模块，可用于软件定时器、外部脉冲捕获、高速脉冲输出和 PWM 脉宽调制输出。

PCA 内部含有一个特殊的 16 位计数器，3 组 PCA 模块均与之相连接。PCA 计数器的结构图如下：



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,x000
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
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
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
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

17.1.1 PCA 控制寄存器 (CCON)

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): 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。

17.1.2 PCA 模式寄存器 (CMOD)

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	注意事项
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	外部时钟频率不能高于系统频率的 1/2
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

17.1.3 PCA 计数器寄存器 (CL, CH)

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。

17.1.4 PCA 模块模式控制寄存器 (CCAPMn)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CCAPM0	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

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

17.1.5 PCA 模块模式捕获值/比较值寄存器 (CCAPnL, CCAPnH)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CCAP0L	EAH								
CCAP1L	EBH								
CCAP2L	ECH								
CCAP0H	FAH								
CCAP1H	FBH								
CCAP2H	FCH								

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。

17.1.6 PCA 模块 PWM 模式控制寄存器 (PCA_PWMn)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PCA_PWM0	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

EBSn[1:0]: PCA 模块 n 的 PWM 位数控制

EBSn[1:0]	PWM 位数	重载值	比较值
00	8 位 PWM	{EPCnH, CCAPnH[7:0]}	{EPCnL, CCAPnL[7:0]}
01	7 位 PWM	{EPCnH, CCAPnH[6:0]}	{EPCnL, CCAPnL[6:0]}
10	6 位 PWM	{EPCnH, CCAPnH[5:0]}	{EPCnL, CCAPnL[5:0]}
11	10 位 PWM	{EPCnH, XCCAPnH[1:0], CCAPnH[7:0]}	{EPCnL, XCCAPnL[1:0], CCAPnL[7:0]}

XCCAPnH[1:0]: 10 位 PWM 的第 9 位和第 10 位的重载值

XCCAPnL[1:0]: 10 位 PWM 的第 9 位和第 10 位的比较值

EPCnH: PWM 模式下, 重载值的最高位 (8 位 PWM 的第 9 位, 7 位 PWM 的第 8 位, 6 位 PWM 的第 7 位, 10 位 PWM 的第 11 位)

EPCnL: PWM 模式下, 比较值的最高位 (8 位 PWM 的第 9 位, 7 位 PWM 的第 8 位, 6 位 PWM 的第 7 位, 10 位 PWM 的第 11 位)

注意: 在更新 10 位 PWM 的重载值时, 必须先写高两位 XCCAPnH[1:0], 再写低 8 位 CCAPnH[7:0]。

17.2 PCA 工作模式

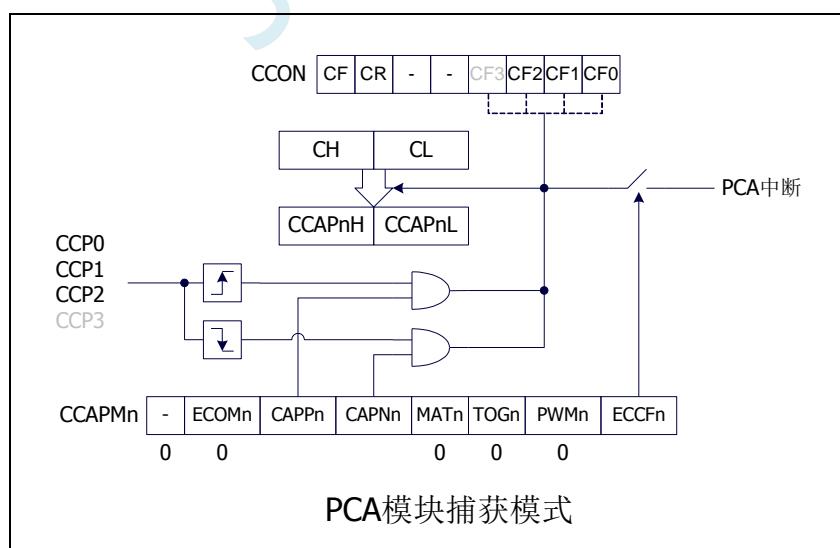
STC8 系列单片机共有 4 组 PCA 模块，每组模块都可独立设置工作模式。模式设置如下所示：

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 位高速脉冲输出

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 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：

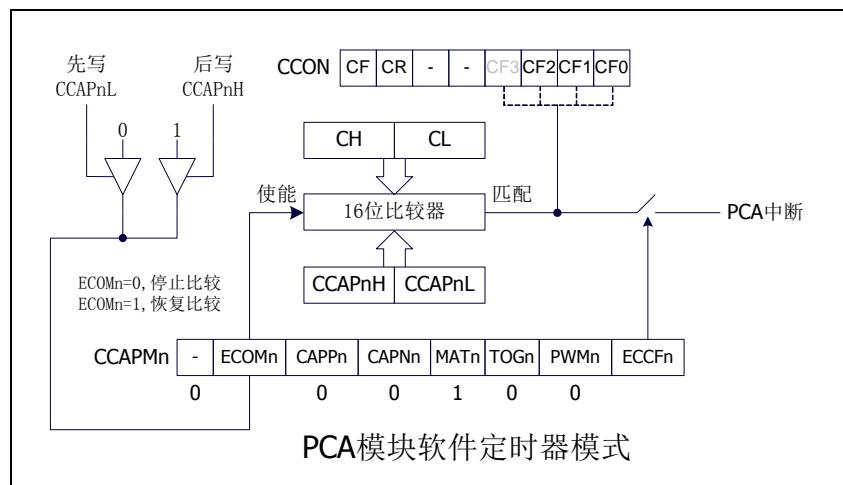


17.2.2 软件定时器模式

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。

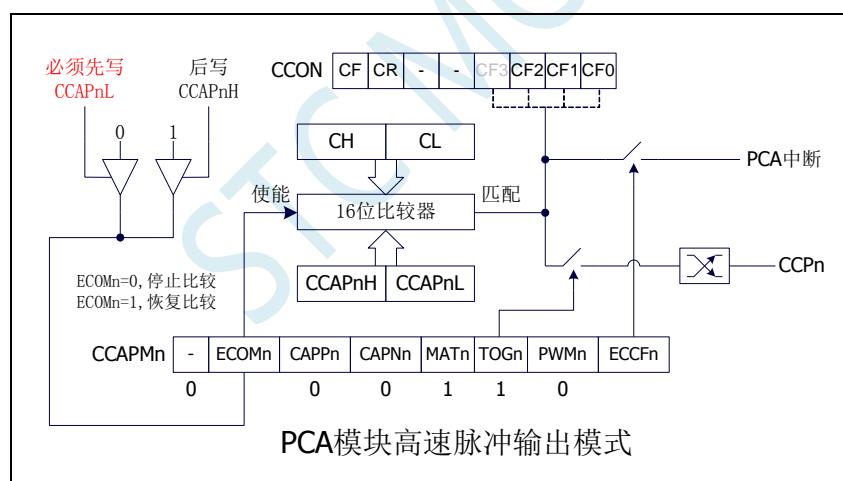
PCA 模块工作于软件定时器模式的结构图如下图所示:



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:



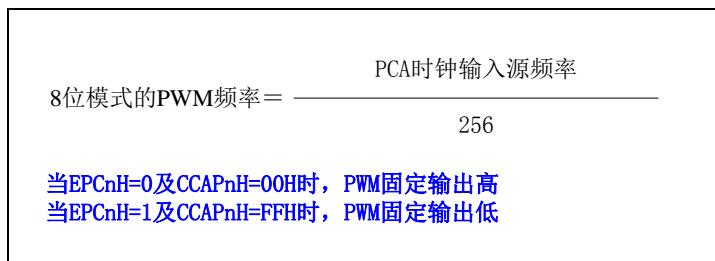
17.2.4 PWM 脉宽调制模式及频率计算公式

17.2.4.1 8 位 PWM 模式

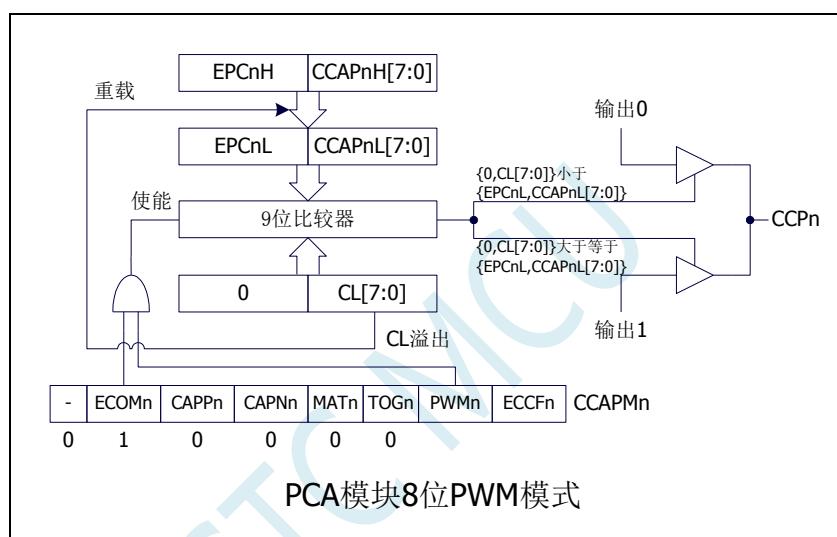
脉宽调制是使用程序来控制波形的占空比、周期、相位波形的一种技术, 在三相电机驱动、D/A 转换等场合有广泛的应用。STC8 系列单片机的 PCA 模块可以通过设定各自的 PCA_PWMn 寄存器使其工作于 8 位 PWM 或 7 位 PWM 或 6 位 PWM 或 10 位 PWM 模式。要使能 PCA 模块的 PWM 功能, 模块寄存器 CCAPMn 的 PWMn 和 ECOMn 位必须置 1。

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。

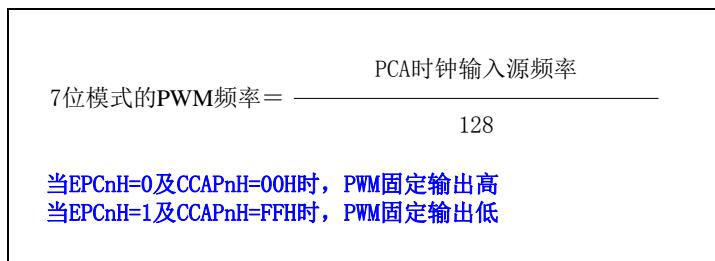


PCA 模块工作于 8 位 PWM 模式的结构图如下图所示:

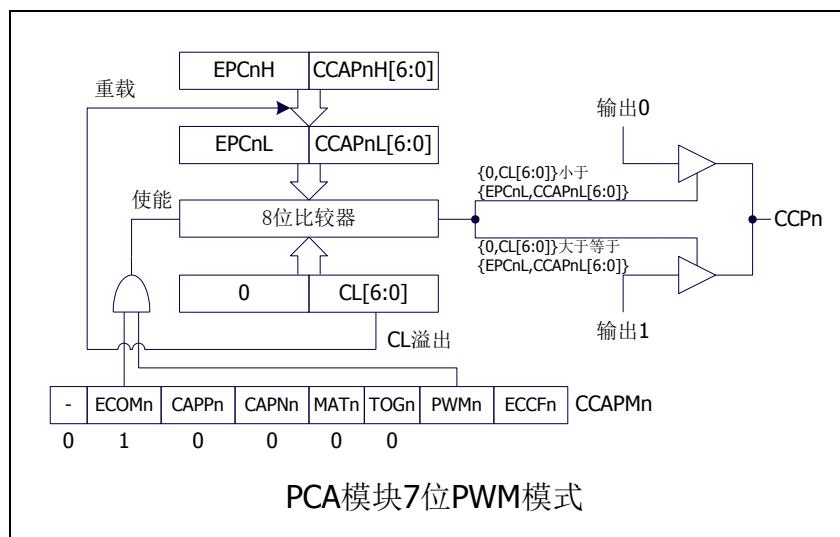


17.2.4.2 7 位 PWM 模式

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。

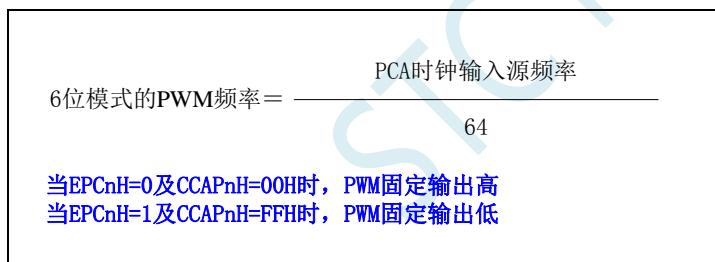


PCA 模块工作于 7 位 PWM 模式的结构图如下图所示:

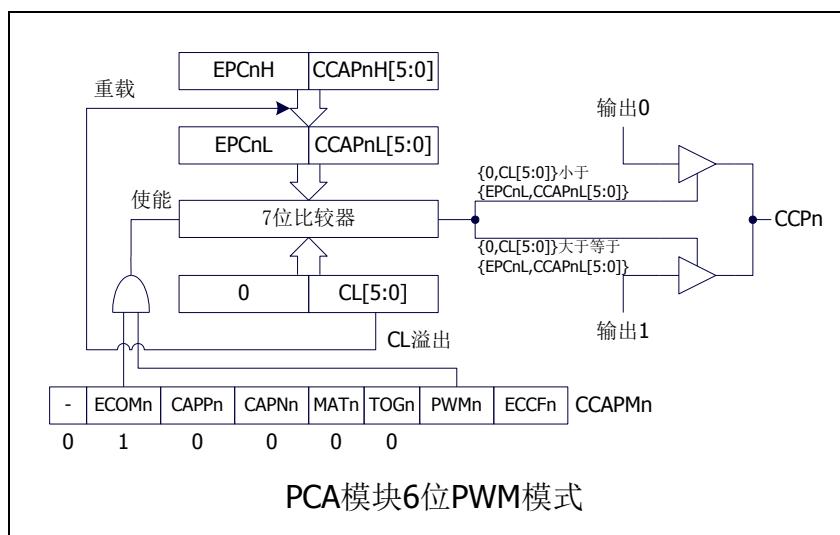


17.2.4.3 6 位 PWM 模式

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。

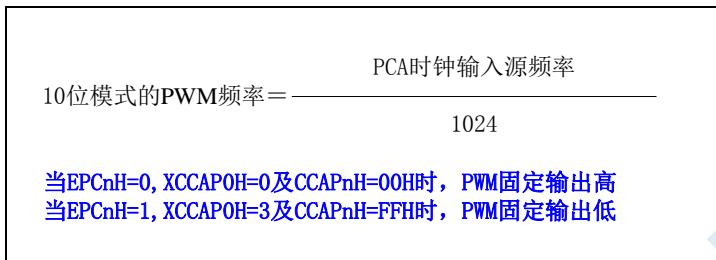


PCA 模块工作于 6 位 PWM 模式的结构图如下图所示：

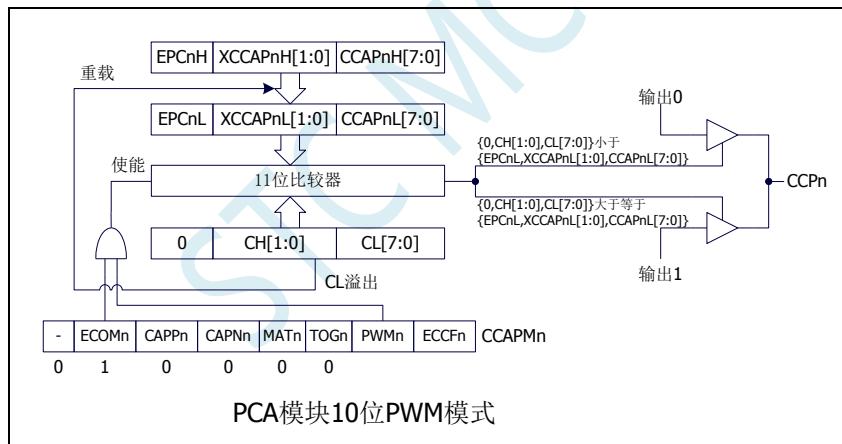


17.2.4.4 10 位 PWM 模式

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：

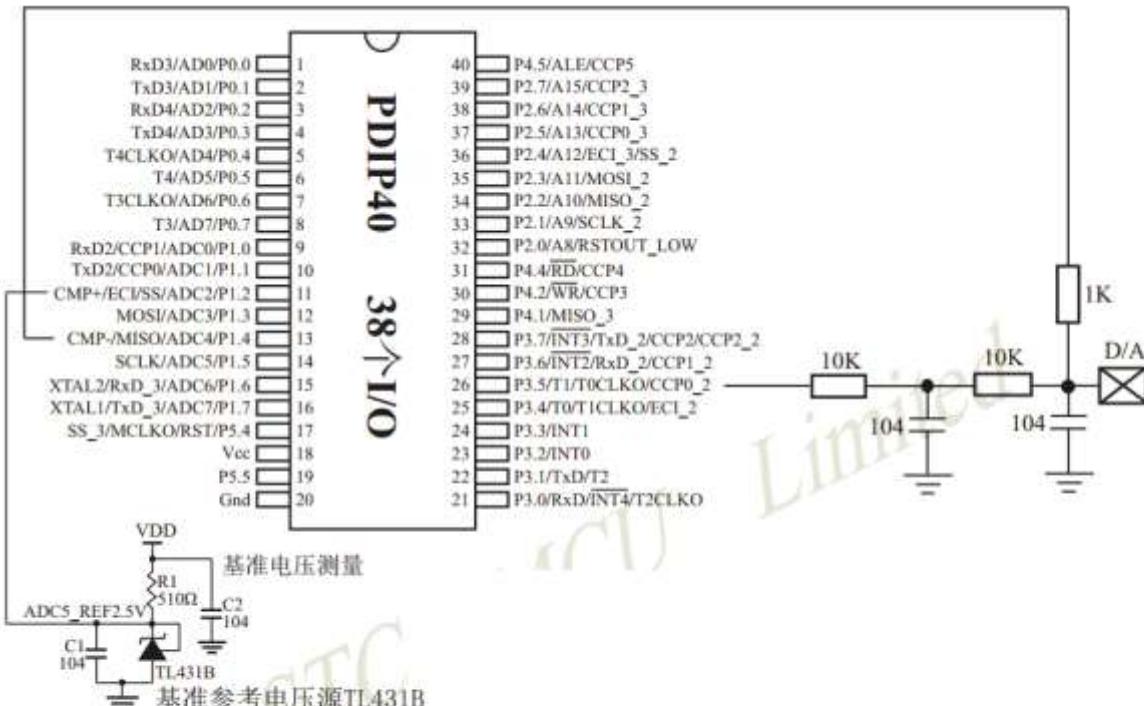


17.2.4.5 如何控制 PWM 固定输出高电平/低电平

当 PCA_PWMn &= 0xC0, CCAPnH = 0x00 时，PWM 固定输出高电平

当 PCA_PWMn |= 0x3F, CCAPnH = 0xFF 时，PWM 固定输出低电平

17.3 利用 CCP/PCA/PWM 模块实现 8~16 位 DAC 的参考线路图



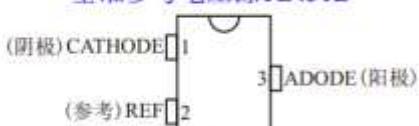
如应用简单，可无需基准参考电压源，直接与V_{CC}比较即可。

提示：

利用CCP/PCA模块的高速脉冲输出功能实现9~16位PWM来实现9~16位DAC，或用本身的硬件8位PWM来实现8位DAC，单片机本身也有10位ADC。

- (1) PWM频率越高，输出波形越平滑。
(2) 如果工作电压为5V，需输出1V电压，则设置高电平为1/5，低电平为4/5，则PWM输出电压就为1V。
(3) 如果要输出高精准电压，建议用A/D检测输出的电压值，然后根据A/D检测的电压值逐步调整到所需要的电压。

基准参考电压源TL431B



SOT23-3封装，RMB¥0.15~0.3

基准参考电压源TL431B的符号



如应用简单，可无需基准参考电压源，直接与V_{CC}比较即可。

17.4 范例程序

17.4.1 PCA 输出 PWM (6/7/8/10 位)

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr CCON      = 0xd8;
sbit CF       = CCON^7;
sbit CR       = CCON^6;
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 P0M1     = 0x93;
sfr P0M0     = 0x94;
sfr P1M1     = 0x91;
sfr P1M0     = 0x92;
sfr P2M1     = 0x95;
sfr P2M0     = 0x96;
sfr P3M1     = 0xb1;
sfr P3M0     = 0xb2;
sfr P4M1     = 0xb3;
sfr P4M0     = 0xb4;
sfr P5M1     = 0xc9;
sfr P5M0     = 0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
```

```

P4M0 = 0x00;
P4M1 = 0x00;
P5M0 = 0x00;
P5M1 = 0x00;

CCON = 0x00;
CMOD = 0x08;                                //PCA 时钟为系统时钟
CL = 0x00;
CH = 0x00;

//--6 位 PWM --
CCAPM0 = 0x42;                            //PCA 模块0 为 PWM 工作模式
PCA_PWM0 = 0x80;                           //PCA 模块0 输出6位 PWM
CCAP0L = 0x20;                             //PWM 占空比为50%[(40H-20H)/40H]
CCAP0H = 0x20;

//--7 位 PWM --
CCAPMI = 0x42;                            //PCA 模块1 为 PWM 工作模式
PCA_PWM1 = 0x40;                           //PCA 模块1 输出7位 PWM
CCAP1L = 0x20;                             //PWM 占空比为75%[(80H-20H)/80H]
CCAP1H = 0x20;

//--8 位 PWM --
// CCAPM2 = 0x42;                            //PCA 模块2 为 PWM 工作模式
// PCA_PWM2 = 0xc0;                           //PCA 模块2 输出8位 PWM
// CCAP2L = 0x20;                            //PWM 占空比为87.5%[(100H-20H)/100H]
// CCAP2H = 0x20;

//--10 位 PWM --
CCAPM2 = 0x42;                            //PCA 模块2 为 PWM 工作模式
PCA_PWM2 = 0xc0;                           //PCA 模块2 输出10位 PWM
CCAP2L = 0x20;                            //PWM 占空比为96.875%[(400H-20H)/400H]
CCAP2H = 0x20;

CR = 1;                                    //启动PCA 计时器

while (1);
}

```

汇编代码

;测试工作频率为 11.0592MHz

CCON	DATA	0D8H
CF	BIT	CCON.7
CR	BIT	CCON.6
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
CCAPMI	DATA	0DBH
CCAP1L	DATA	0EBH
CCAP1H	DATA	0FBH
PCA_PWM1	DATA	0F3H
CCAPM2	DATA	0DCH
CCAP2L	DATA	0ECH
CCAP2H	DATA	0FCH
PCA_PWM2	DATA	0F4H

```

P0M1      DATA      093H
P0M0      DATA      094H
P1M1      DATA      091H
P1M0      DATA      092H
P2M1      DATA      095H
P2M0      DATA      096H
P3M1      DATA      0B1H
P3M0      DATA      0B2H
P4M1      DATA      0B3H
P4M0      DATA      0B4H
P5M1      DATA      0C9H
P5M0      DATA      0CAH

        ORG      0000H
        LJMP    MAIN

        ORG      0100H
MAIN:
        MOV      SP, #5FH
        MOV      P0M0, #00H
        MOV      P0M1, #00H
        MOV      P1M0, #00H
        MOV      P1M1, #00H
        MOV      P2M0, #00H
        MOV      P2M1, #00H
        MOV      P3M0, #00H
        MOV      P3M1, #00H
        MOV      P4M0, #00H
        MOV      P4M1, #00H
        MOV      P5M0, #00H
        MOV      P5M1, #00H

        MOV      CCON,#00H
        MOV      CMOD,#08H ;PCA 时钟为系统时钟
        MOV      CL,#00H
        MOV      CH,#0H

;--6 位 PWM --
        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

;--7 位 PWM --
        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      CCAP1H,#20H

;--8 位 PWM --
        ;       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

;--10 位 PWM --
        MOV      CCAPM2,#42H ;PCA 模块2 为 PWM 工作模式
        MOV      PCA_PWM2,#0C0H ;PCA 模块2 输出10位 PWM
        MOV      CCAP2L,#20H ;PWM 占空比为96.875%[(400H-20H)/400H]
        MOV      CCAP2H,#20H
        SETB    CR      ;启动PCA计时器

```

JMP \$**END**

17.4.2 PCA 捕获测量脉冲宽度

C 语言代码

//测试工作频率为 11.0592MHz

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

sfr CCON      = 0xd8;
sbit CF        = CCON^7;
sbit CR        = CCON^6;
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 CCAPMI    = 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 P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

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++;
}
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()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    cnt = 0;                                         //用户变量初始化
    count0 = 0;
    count1 = 0;
    length = 0;
    CCON = 0x00;
    CMOD = 0x09;                                     //PCA 时钟为系统时钟,使能PCA 计时中断
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x11;                                    //PCA 模块0 为16 位捕获模式 (下降沿捕获)
//    CCAPM0 = 0x21;                                    //PCA 模块0 为16 位捕获模式 (上升沿捕获)
//    CCAPM0 = 0x31;                                    //PCA 模块0 为16 位捕获模式 (边沿捕获)
    CCAP0L = 0x00;
    CCAP0H = 0x00;
    CR = 1;                                         //启动PCA 计时器
    EA = 1;

    while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

CCON	DATA	0D8H
CF	BIT	CCON.7
CR	BIT	CCON.6
CCF2	BIT	CCON.2
CCF1	BIT	CCON.1

<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>CNT</i>	<i>DATA</i>	<i>20H</i>
<i>COUNT0</i>	<i>DATA</i>	<i>21H</i>
<i>COUNT1</i>	<i>DATA</i>	<i>24H</i>
<i>LENGTH</i>	<i>DATA</i>	<i>27H</i>
		<i>;3 bytes</i>
		<i>;3 bytes</i>
		<i>;3 bytes, (COUNT1-COUNT0)</i>
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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>JNB</i>		<i>CF,CHECKCCF0</i>
<i>CLR</i>		<i>CF</i>
<i>INC</i>		<i>;清中断标志</i>
		<i>; PCA 计时溢出次数+1</i>
<i>CHECKCCF0:</i>		
<i>JNB</i>		<i>CCF0,ISREXIT</i>
<i>CLR</i>		<i>CCF0</i>
<i>MOV</i>		<i>COUNT0,COUNT1</i>
<i>MOV</i>		<i>;备份上一次的捕获值</i>
<i>MOV</i>		<i>COUNT0+1,COUNT1+1</i>
<i>MOV</i>		<i>COUNT0+2,COUNT1+2</i>
<i>MOV</i>		<i>COUNT1,CNT</i>
<i>MOV</i>		<i>;保存本次的捕获值</i>
<i>MOV</i>		<i>COUNT1+1,CCAP0H</i>
<i>MOV</i>		<i>COUNT1+2,CCAP0L</i>
<i>CLR</i>		<i>C</i>
<i>MOV</i>		<i>;计算两次的捕获差值</i>
<i>SUBB</i>		<i>A,COUNT1+2</i>
		<i>A,COUNT0+2</i>

<i>MOV</i>	<i>LENGTH+2,A</i>	
<i>MOV</i>	<i>A,COUNT1+1</i>	
<i>SUBB</i>	<i>A,COUNT0+1</i>	
<i>MOV</i>	<i>LENGTH+1,A</i>	
<i>MOV</i>	<i>A,COUNT1</i>	
<i>SUBB</i>	<i>A,COUNT0</i>	
<i>MOV</i>	<i>LENGTH,A</i>	<i>;LENGTH 保存的即为捕获的脉冲宽度</i>
ISREXIT:		
<i>POP</i>	<i>PSW</i>	
<i>POP</i>	<i>ACC</i>	
<i>RETI</i>		
MAIN:		
<i>MOV</i>	<i>SP, #5FH</i>	
<i>MOV</i>	<i>P0M0, #00H</i>	
<i>MOV</i>	<i>P0M1, #00H</i>	
<i>MOV</i>	<i>P1M0, #00H</i>	
<i>MOV</i>	<i>P1M1, #00H</i>	
<i>MOV</i>	<i>P2M0, #00H</i>	
<i>MOV</i>	<i>P2M1, #00H</i>	
<i>MOV</i>	<i>P3M0, #00H</i>	
<i>MOV</i>	<i>P3M1, #00H</i>	
<i>MOV</i>	<i>P4M0, #00H</i>	
<i>MOV</i>	<i>P4M1, #00H</i>	
<i>MOV</i>	<i>P5M0, #00H</i>	
<i>MOV</i>	<i>P5M1, #00H</i>	
<i>CLR</i>	<i>A</i>	
<i>MOV</i>	<i>CNT,A</i>	<i>;用户变量初始化</i>
<i>MOV</i>	<i>COUNT0,A</i>	
<i>MOV</i>	<i>COUNT0+1,A</i>	
<i>MOV</i>	<i>COUNT0+2,A</i>	
<i>MOV</i>	<i>COUNT1,A</i>	
<i>MOV</i>	<i>COUNT1+1,A</i>	
<i>MOV</i>	<i>COUNT1+2,A</i>	
<i>MOV</i>	<i>LENGTH,A</i>	
<i>MOV</i>	<i>LENGTH+1,A</i>	
<i>MOV</i>	<i>LENGTH+2,A</i>	
<i>MOV</i>	<i>CCON,#00H</i>	
<i>MOV</i>	<i>CMOD,#09H</i>	<i>;PCA 时钟为系统时钟,使能 PCA 计时中断</i>
<i>MOV</i>	<i>CL,#00H</i>	
<i>MOV</i>	<i>CH,#0H</i>	
<i>MOV</i>	<i>CCAPM0,#11H</i>	<i>;PCA 模块 0 为 16 位捕获模式 (下降沿捕获)</i>
<i>MOV</i>	<i>CCAPM0,#21H</i>	<i>;PCA 模块 0 为 16 位捕获模式 (上升沿捕获)</i>
<i>MOV</i>	<i>CCAPM0,#31H</i>	<i>;PCA 模块 0 为 16 位捕获模式 (边沿捕获)</i>
<i>;</i>		
<i>;</i>		
<i>MOV</i>	<i>CCAP0L,#00H</i>	
<i>MOV</i>	<i>CCAP0H,#00H</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>\$</i>	
END		

17.4.3 PCA 实现 16 位软件定时

C 语言代码

//测试工作频率为 11.0592MHz

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

#define T50HZ      (11059200L / 12 / 2 / 50)

sfr CCON      = 0xd8;
sbit CF        = CCON^7;
sbit CR        = CCON^6;
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 CCAPMI    = 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 P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

sbit P10       = PI^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()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CCON = 0x00;
    CMOD = 0x00;                                //PCA 时钟为系统时钟/12
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x49;                             //PCA 模块0 为16位定时器模式
    value = T50HZ;
    CCAP0L = value;
    CCAP0H = value >> 8;
    value += T50HZ;
    CR = I;                                     //启动PCA 计时器
    EA = I;

    while (1);
}

```

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>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>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>T50HZ</i>	<i>EQU</i>	<i>2400H</i>	<i>;11059200/12/2/50</i>
--------------	------------	--------------	--------------------------

<i>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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>
<i>POP</i>	<i>PSW</i>
<i>POP</i>	<i>ACC</i>
<i>RETI</i>	

; 测试端口, 闪烁频率为 50Hz

MAIN:

<i>MOV</i>	<i>SP, #5FH</i>
<i>MOV</i>	<i>P0M0, #00H</i>
<i>MOV</i>	<i>P0M1, #00H</i>
<i>MOV</i>	<i>P1M0, #00H</i>
<i>MOV</i>	<i>P1M1, #00H</i>
<i>MOV</i>	<i>P2M0, #00H</i>
<i>MOV</i>	<i>P2M1, #00H</i>
<i>MOV</i>	<i>P3M0, #00H</i>
<i>MOV</i>	<i>P3M1, #00H</i>
<i>MOV</i>	<i>P4M0, #00H</i>
<i>MOV</i>	<i>P4M1, #00H</i>
<i>MOV</i>	<i>P5M0, #00H</i>
<i>MOV</i>	<i>P5M1, #00H</i>
<i>MOV</i>	<i>CCON,#00H</i>
<i>MOV</i>	<i>CMOD,#00H</i>
<i>MOV</i>	<i>CL,#00H</i>
<i>MOV</i>	<i>CH,#0H</i>
<i>MOV</i>	<i>CCAPM0,#49H</i>
<i>MOV</i>	<i>CCAP0L,#LOW T50HZ</i>
<i>MOV</i>	<i>CCAP0H,#HIGH T50HZ</i>
<i>SETB</i>	<i>CR</i>
<i>SETB</i>	<i>EA</i>

; PCA 时钟为系统时钟/12

; PCA 模块 0 为 16 位定时器模式

; 启动 PCA 计时器

JMP \$
END

17.4.4 PCA 实现 16 位软件定时 (ECI 外部时钟模式)

注意: 外部时钟频率不能高于系统频率的 1/2

C 语言代码

//测试工作频率为 11.0592MHz

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

#define T50HZ      (11059200L / 12 / 2 / 50)

sfr CCON      = 0xd8;
sbit CF        = CCON^7;
sbit CR        = CCON^6;
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 CCAPMI    = 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 P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

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

    CCON = 0x00;
    CMOD = 0x06;                                //PCA 时钟为从 ECI 端口输入的外部时钟
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x49;                                //PCA 模块 0 为 16 位定时器模式
    value = T50HZ;
    CCAP0L = value;
    CCAP0H = value >> 8;
    value += T50HZ;
    CR = I;                                      //启动 PCA 计时器
    EA = I;

    while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

CCON	DATA	0D8H
CF	BIT	CCON.7
CR	BIT	CCON.6
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*

T50HZ EQU *2400H* ;11059200/12/2/50

P0M1 DATA *093H*
P0M0 DATA *094H*
P1M1 DATA *091H*
P1M0 DATA *092H*
P2M1 DATA *095H*
P2M0 DATA *096H*
P3M1 DATA *0B1H*
P3M0 DATA *0B2H*
P4M1 DATA *0B3H*
P4M0 DATA *0B4H*
P5M1 DATA *0C9H*
P5M0 DATA *0CAH*

ORG *0000H*
LJMP MAIN
ORG *003BH*
LJMP PCAISR

ORG *0100H*

PCAISR:

PUSH ACC
PUSH PSW
CLR CCF0
MOV A,CCAP0L
ADD A,#LOW T50HZ
MOV CCAP0L,A
MOV A,CCAP0H
ADDC A,#HIGH T50HZ
MOV CCAP0H,A
CPL P1.0 ;测试端口,闪烁频率为50Hz
POP PSW
POP ACC
RETI

MAIN:

MOV SP, #5FH
MOV P0M0, #00H
MOV P0M1, #00H
MOV P1M0, #00H
MOV P1M1, #00H
MOV P2M0, #00H
MOV P2M1, #00H
MOV P3M0, #00H
MOV P3M1, #00H
MOV P4M0, #00H
MOV P4M1, #00H
MOV P5M0, #00H
MOV P5M1, #00H

MOV CCON,#00H
MOV CMOD,#06H

;PCA 时钟为从 ECI 端口输入的外部时钟

```

MOV      CL,#00H
MOV      CH,#0H
MOV      CCAPM0,#49H           ;PCA 模块0为16位定时器模式
MOV      CCAP0L,#LOW T50HZ
MOV      CCAP0H,#HIGH T50HZ
SETB    CR                   ;启动PCA计时器
SETB    EA

JMP      $

END

```

17.4.5 PCA 输出高速脉冲

C 语言代码

//测试工作频率为11.0592MHz

```

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

#define  T38K4HZ      (11059200L / 2 / 38400)

sfr    CCON      = 0xd8;
sbit   CF        = CCON^7;
sbit   CR        = CCON^6;
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    CCAPMI     = 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    P0M1      = 0x93;
sfr    P0M0      = 0x94;
sfr    P1M1      = 0x91;
sfr    P1M0      = 0x92;
sfr    P2M1      = 0x95;
sfr    P2M0      = 0x96;
sfr    P3M1      = 0xb1;
sfr    P3M0      = 0xb2;
sfr    P4M1      = 0xb3;
sfr    P4M0      = 0xb4;
sfr    P5M1      = 0xc9;

```

```

sfr      P5M0      = 0xca;
unsigned int      value;

void PCA_Isr() interrupt 7
{
    CCF0 = 0;
    CCAP0L = value;
    CCAP0H = value >> 8;
    value += T38K4HZ;
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CCON = 0x00;
    CMOD = 0x08;                                //PCA 时钟为系统时钟
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x4d;                                //PCA 模块0 为16位定时器模式并使能脉冲输出
    value = T38K4HZ;
    CCAP0L = value;
    CCAP0H = value >> 8;
    value += T38K4HZ;
    CR = 1;                                       //启动PCA 计时器
    EA = 1;

    while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

CCON	DATA	0D8H
CF	BIT	CCON.7
CR	BIT	CCON.6
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

<i>CCAPMI</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>T38K4HZ</i>	<i>EQU</i>	<i>90H</i>
		;11059200/2/38400
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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, #5FH</i>
	<i>MOV</i>	<i>P0M0, #00H</i>
	<i>MOV</i>	<i>P0M1, #00H</i>
	<i>MOV</i>	<i>P1M0, #00H</i>
	<i>MOV</i>	<i>P1M1, #00H</i>
	<i>MOV</i>	<i>P2M0, #00H</i>
	<i>MOV</i>	<i>P2M1, #00H</i>
	<i>MOV</i>	<i>P3M0, #00H</i>
	<i>MOV</i>	<i>P3M1, #00H</i>
	<i>MOV</i>	<i>P4M0, #00H</i>
	<i>MOV</i>	<i>P4M1, #00H</i>
	<i>MOV</i>	<i>P5M0, #00H</i>
	<i>MOV</i>	<i>P5M1, #00H</i>

```

MOV      CCON,#00H
MOV      CMOD,#08H          ;PCA 时钟为系统时钟
MOV      CL,#00H
MOV      CH,#0H
MOV      CCAPM0,#4DH        ;PCA 模块 0 为 16 位定时器模式并使能脉冲输出
MOV      CCAP0L,#LOW T38K4HZ
MOV      CCAP0H,#HIGH T38K4HZ
SETB    CR                 ;启动 PCA 计时器
SETB    EA
JMP     $

END

```

17.4.6 PCA 扩展外部中断

C 语言代码

//测试工作频率为 11.0592MHz

```

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

sfr    CCON      = 0xd8;
sbit   CF        = CCON^7;
sbit   CR        = CCON^6;
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    CCAPMI    = 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    P0M1      = 0x93;
sfr    P0M0      = 0x94;
sfr    P1M1      = 0x91;
sfr    P1M0      = 0x92;
sfr    P2M1      = 0x95;
sfr    P2M0      = 0x96;
sfr    P3M1      = 0xb1;
sfr    P3M0      = 0xb2;
sfr    P4M1      = 0xb3;
sfr    P4M0      = 0xb4;
sfr    P5M1      = 0xc9;

```

```

sfr      P5M0      = 0xca;
sbit     P10       = P1^0;

void PCA_Isr() interrupt 7
{
    CCF0 = 0;
    P10 = !P10;
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CCON = 0x00;
    CMOD = 0x08;           //PCA 时钟为系统时钟
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x11;         //扩展外部端口 CCP0 为下降沿中断口
//    CCAPM0 = 0x21;         //扩展外部端口 CCP0 为上升沿中断口
//    CCAPM0 = 0x31;         //扩展外部端口 CCP0 为边沿中断口
    CCAP0L = 0;
    CCAP0H = 0;
    CR = 1;                //启动 PCA 计时器
    EA = 1;

    while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

CCON	DATA	0D8H
CF	BIT	CCON.7
CR	BIT	CCON.6
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
CCAP0L	DATA	0EBH

<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>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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>CLR</i>	<i>CCF0</i>
	<i>CPL</i>	<i>P1.0</i>
	<i>RETI</i>	
<i>MAIN:</i>		
	<i>MOV</i>	<i>SP, #5FH</i>
	<i>MOV</i>	<i>P0M0, #00H</i>
	<i>MOV</i>	<i>P0M1, #00H</i>
	<i>MOV</i>	<i>P1M0, #00H</i>
	<i>MOV</i>	<i>P1M1, #00H</i>
	<i>MOV</i>	<i>P2M0, #00H</i>
	<i>MOV</i>	<i>P2M1, #00H</i>
	<i>MOV</i>	<i>P3M0, #00H</i>
	<i>MOV</i>	<i>P3M1, #00H</i>
	<i>MOV</i>	<i>P4M0, #00H</i>
	<i>MOV</i>	<i>P4M1, #00H</i>
	<i>MOV</i>	<i>P5M0, #00H</i>
	<i>MOV</i>	<i>P5M1, #00H</i>
	<i>MOV</i>	<i>CCON,#00H</i>
	<i>MOV</i>	<i>CMOD,#08H</i>
	<i>MOV</i>	<i>CL,#00H</i>
	<i>MOV</i>	<i>CH,#0H</i>
	<i>MOV</i>	<i>CCAPM0,#11H</i>
		; 扩展外部端口 CCP0 为下降沿中断口
;	<i>MOV</i>	<i>CCAPM0,#21H</i>
;	<i>MOV</i>	<i>CCAPM0,#31H</i>
		; 扩展外部端口 CCP0 为上升沿中断口
		; 扩展外部端口 CCP0 为边沿中断口
	<i>MOV</i>	<i>CCAP0L,#0</i>
	<i>MOV</i>	<i>CCAP0H,#0</i>
	<i>SETB</i>	<i>CR</i>
		; 启动 PCA 计时器
	<i>SETB</i>	<i>EA</i>
	<i>JMP</i>	\$

END

STCMCU

18 精度可达 15 位的增强型 PWM（最多可输出 45 路各自独立的 PWM）

产品线	增强型 PWM
STC8G1K08 系列	
STC8G1K08-8Pin 系列	
STC8G1K08A 系列	
STC8G2K64S4 系列	●
STC8G2K64S2 系列	●
STC8G1K08T 系列	
STC15H2K64S4 系列	●

(C 语言程序中使用中断号大于 31 的中断时，在 Keil 中编译会报错，解决办法请参考附录)

STC8G 部分系列单片机集成了 6 组增强型的 PWM 波形发生器，每组可单独设置周期，每组为各自独立的 8 路，由于 P5 口缺少 P5.5/P5.6/P5.7 这 3 个端子，所以 STC8G2K64S4 系列最多可输出 45 路 PWM。

第 0 组，共 8 路，PWM00~PWM07，使用 PWM0CH/PWM0CL 设置周期值；
 第 1 组，共 8 路，PWM10~PWM17，使用 PWM1CH/PWM1CL 设置周期值；
 第 2 组，共 8 路，PWM20~PWM27，使用 PWM2CH/PWM2CL 设置周期值；
 第 3 组，共 8 路，PWM30~PWM37，使用 PWM3CH/PWM3CL 设置周期值；
 第 4 组，共 8 路，PWM40~PWM47，使用 PWM4CH/PWM4CL 设置周期值；
 第 5 组，共 5 路，PWM50~PWM54，使用 PWM5CH/PWM5CL 设置周期值。

STC8G2K64S2 系列单片机集成了 1 组（P2 口）增强型的 PWM 波形发生器，P2 组可产生各自独立的 8 路 PWM，所以 STC8G2K64S2 系列最多可输出 8 路 PWM。PWM20~PWM27，使用 PWM2CH/PWM2CL 设置周期值。

PWM 的时钟源可以选择。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，可以非常灵活的控制每路 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。由于每组的 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。**注：增强型 PWM 只有输出功能，如果需要测量脉冲宽度，请使用本系列的 PCA/CCP/PWM 功能**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/P0.6/P0.7 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。

STC 三种硬件 PWM 比较：

兼容传统 8051 的 PCA/CCP/PWM：可输出 PWM 波形、捕获外部输入信号以及输出高速脉冲。可对外输出 6 位/7 位/8 位/10 位的 PWM 波形，6 位 PWM 波形的频率为 PCA 模块时钟源频率/64；7 位 PWM 波形的频率为 PCA 模块时钟源频率/128；8 位 PWM 波形的频率为 PCA 模块时钟源频率/256；10 位 PWM 波形的频率为 PCA 模块时钟源频率/1024。捕获外部输入信号，可捕获上升沿、下降沿或者同

时捕获上升沿和下降沿。

STC8G 系列的 15 位增强型 PWM: 只能对外输出 PWM 波形, 无输入捕获功能。对外输出 PWM 的频率以及占空比均可任意设置。通过软件干预, 可实现多路互补/对称/带死区的 PWM 波形。有外部异常检测功能以及实时触发 ADC 转换功能。

STC8H 系列的 16 位高级 PWM 定时器: 是目前 STC 功能最强的 PWM, 可对外输出任意频率以及任意占空比的 PWM 波形。无需软件干预即可输出互补/对称/带死区的 PWM 波形。能捕获外部输入信号, 可捕获上升沿、下降沿或者同时捕获上升沿和下降沿, 测量外部波形时, 可同时测量波形的周期值和占空比值。有正交编码功能、外部异常检测功能以及实时触发 ADC 转换功能。

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	
PWMSET	增强型 PWM 全局配置寄存器	F1H	ENGLBSET	PWMRST	ENPWM5	ENPWM4	ENPWM3	ENPWM2	ENPWM1	ENPWM0	0000,0000
PWMCFG01	增强型 PWM 配置寄存器	F6H	PWM1CBIF	EPWM1CBI	FLTPS0	PWM1CEN	PWM0CBIF	EPWM0CBI	ENPWM0TA	PWM0CEN	0000,0000
PWMCFG23	增强型 PWM 配置寄存器	F7H	PWM3CBIF	EPWM3CBI	FLTPS1	PWM3CEN	PWM2CBIF	EPWM2CBI	ENPWM2TA	PWM2CEN	0000,0000
PWMCFG45	增强型 PWM 配置寄存器	FEH	PWM5CBIF	EPWM5CBI	FLTPS2	PWM5CEN	PWM4CBIF	EPWM4CBI	ENPWM4TA	PWM4CEN	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	
PWM0CH	PWM0 计数器高字节	FF00H	-								x000,0000
PWM0CL	PWM0 计数器低字节	FF01H									0000,0000
PWM0CKS	PWM0 时钟选择	FF02H	-	-	-	SELT2	PWM_PS[3:0]				xxx0,0000
PWM0TADCH	PWM0 触发 ADC 计数高字节	FF03H	-								x000,0000
PWM0TADCL	PWM0 触发 ADC 计数低字节	FF04H									0000,0000
PWM0IF	PWM0 中断标志寄存器	FF05H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWM0FDCR	PWM0 异常检测控制寄存器	FF06H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	0000,0000
PWM00T1H	PWM00T1 计数值高字节	FF10H	-								x000,0000
PWM00T1L	PWM00T1 计数值低字节	FF11H									0000,0000
PWM00T2H	PWM00T2 计数值高字节	FF12H	-								x000,0000
PWM00T2L	PWM00T2 计数值低字节	FF13H									0000,0000
PWM00CR	PWM00 控制寄存器	FF14H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM00HLD	PWM00 电平保持控制寄存器	FF15H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM01T1H	PWM01T1 计数值高字节	FF18H	-								x000,0000
PWM01T1L	PWM01T1 计数值低字节	FF19H									0000,0000
PWM01T2H	PWM01T2 计数值高字节	FF1AH	-								x000,0000
PWM01T2L	PWM01T2 计数值低字节	FF1BH									0000,0000
PWM01CR	PWM01 控制寄存器	FF1CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM01HLD	PWM01 电平保持控制寄存器	FF1DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM02T1H	PWM02T1 计数值高字节	FF20H	-								x000,0000
PWM02T1L	PWM02T1 计数值低字节	FF21H									0000,0000
PWM02T2H	PWM02T2 计数值高字节	FF22H	-								x000,0000
PWM02T2L	PWM02T2 计数值低字节	FF23H									0000,0000
PWM02CR	PWM02 控制寄存器	FF24H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000

PWM02HLD	PWM02 电平保持控制寄存器	FF25H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM03T1H	PWM03T1 计数值高字节	FF28H	-								x000,0000
PWM03T1L	PWM03T1 计数值低字节	FF29H									0000,0000
PWM03T2H	PWM03T2 计数值高字节	FF2AH	-								x000,0000
PWM03T2L	PWM03T2 计数值低字节	FF2BH									0000,0000
PWM03CR	PWM03 控制寄存器	FF2CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM03HLD	PWM03 电平保持控制寄存器	FF2DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM04T1H	PWM04T1 计数值高字节	FF30H	-								x000,0000
PWM04T1L	PWM04T1 计数值低字节	FF31H									0000,0000
PWM04T2H	PWM04T2 计数值高字节	FF32H	-								x000,0000
PWM04T2L	PWM04T2 计数值低字节	FF33H									0000,0000
PWM04CR	PWM04 控制寄存器	FF34H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM04HLD	PWM04 电平保持控制寄存器	FF35H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM05T1H	PWM05T1 计数值高字节	FF38H	-								x000,0000
PWM05T1L	PWM05T1 计数值低字节	FF39H									0000,0000
PWM05T2H	PWM05T2 计数值高字节	FF3AH	-								x000,0000
PWM05T2L	PWM05T2 计数值低字节	FF3BH									0000,0000
PWM05CR	PWM05 控制寄存器	FF3CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM05HLD	PWM05 电平保持控制寄存器	FF3DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM06T1H	PWM06T1 计数值高字节	FF40H	-								x000,0000
PWM06T1L	PWM06T1 计数值低字节	FF41H									0000,0000
PWM06T2H	PWM06T2 计数值高字节	FF42H	-								x000,0000
PWM06T2L	PWM06T2 计数值低字节	FF43H									0000,0000
PWM06CR	PWM06 控制寄存器	FF44H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM06HLD	PWM06 电平保持控制寄存器	FF45H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM07T1H	PWM07T1 计数值高字节	FF48H	-								x000,0000
PWM07T1L	PWM07T1 计数值低字节	FF49H									0000,0000
PWM07T2H	PWM07T2 计数值高字节	FF4AH	-								x000,0000
PWM07T2L	PWM07T2 计数值低字节	FF4BH									0000,0000
PWM07CR	PWM07 控制寄存器	FF4CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM07HLD	PWM07 电平保持控制寄存器	FF4DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM1CH	PWM1 计数器高字节	FF50H	-								x000,0000
PWM1CL	PWM1 计数器低字节	FF51H									0000,0000
PWM1CKS	PWM1 时钟选择	FF52H	-	-	-	SELT2					xxx0,0000
PWM1IF	PWM1 中断标志寄存器	FF55H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	COIF	0000,0000
PWM1FDCR	PWM1 异常检测控制寄存器	FF56H	INVCM	INVIO	ENFD	FLTFLO	-	FDCMP	FDIO	FDIF	0000,0000
PWM10T1H	PWM10T1 计数值高字节	FF60H	-								x000,0000
PWM10T1L	PWM10T1 计数值低字节	FF61H									0000,0000
PWM10T2H	PWM10T2 计数值高字节	FF62H	-								x000,0000
PWM10T2L	PWM10T2 计数值低字节	FF63H									0000,0000
PWM10CR	PWM10 控制寄存器	FF64H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM10HLD	PWM10 电平保持控制寄存器	FF65H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM11T1H	PWM11T1 计数值高字节	FF68H	-								x000,0000
PWM11T1L	PWM11T1 计数值低字节	FF69H									0000,0000

PWM11T2H	PWM11T2 计数值高字节	FF6AH	-									x000,0000
PWM11T2L	PWM11T2 计数值低字节	FF6BH									0000,0000	
PWM11CR	PWM11 控制寄存器	FF6CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	0xx,x000	
PWM11HLD	PWM11 电平保持控制寄存器	FF6DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00	
PWM12T1H	PWM12T1 计数值高字节	FF70H	-									x000,0000
PWM12T1L	PWM12T1 计数值低字节	FF71H									0000,0000	
PWM12T2H	PWM12T2 计数值高字节	FF72H	-									x000,0000
PWM12T2L	PWM12T2 计数值低字节	FF73H									0000,0000	
PWM12CR	PWM12 控制寄存器	FF74H	ENO	INI	-	-		ENI	ENT2I	ENT1I	0xx,x000	
PWM12HLD	PWM12 电平保持控制寄存器	FF75H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00	
PWM13T1H	PWM13T1 计数值高字节	FF78H	-									x000,0000
PWM13T1L	PWM13T1 计数值低字节	FF79H									0000,0000	
PWM13T2H	PWM13T2 计数值高字节	FF7AH	-									x000,0000
PWM13T2L	PWM13T2 计数值低字节	FF7BH									0000,0000	
PWM13CR	PWM13 控制寄存器	FF7CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	0xx,x000	
PWM13HLD	PWM13 电平保持控制寄存器	FF7DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00	
PWM14T1H	PWM14T1 计数值高字节	FF80H	-									x000,0000
PWM14T1L	PWM14T1 计数值低字节	FF81H									0000,0000	
PWM14T2H	PWM14T2 计数值高字节	FF82H	-									x000,0000
PWM14T2L	PWM14T2 计数值低字节	FF83H									0000,0000	
PWM14CR	PWM14 控制寄存器	FF84H	ENO	INI	-	-		ENI	ENT2I	ENT1I	0xx,x000	
PWM14HLD	PWM14 电平保持控制寄存器	FF85H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00	
PWM15T1H	PWM15T1 计数值高字节	FF88H	-									x000,0000
PWM15T1L	PWM15T1 计数值低字节	FF89H									0000,0000	
PWM15T2H	PWM15T2 计数值高字节	FF8AH	-									x000,0000
PWM15T2L	PWM15T2 计数值低字节	FF8BH									0000,0000	
PWM15CR	PWM15 控制寄存器	FF8CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	0xx,x000	
PWM15HLD	PWM15 电平保持控制寄存器	FF8DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00	
PWM16T1H	PWM16T1 计数值高字节	FF90H	-									x000,0000
PWM16T1L	PWM16T1 计数值低字节	FF91H									0000,0000	
PWM16T2H	PWM16T2 计数值高字节	FF92H	-									x000,0000
PWM16T2L	PWM16T2 计数值低字节	FF93H									0000,0000	
PWM16CR	PWM16 控制寄存器	FF94H	ENO	INI	-	-		ENI	ENT2I	ENT1I	0xx,x000	
PWM16HLD	PWM16 电平保持控制寄存器	FF95H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00	
PWM17T1H	PWM17T1 计数值高字节	FF98H	-									x000,0000
PWM17T1L	PWM17T1 计数值低字节	FF99H									0000,0000	
PWM17T2H	PWM17T2 计数值高字节	FF9AH	-									x000,0000
PWM17T2L	PWM17T2 计数值低字节	FF9BH									0000,0000	
PWM17CR	PWM17 控制寄存器	FF9CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	0xx,x000	
PWM17HLD	PWM17 电平保持控制寄存器	FF9DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00	
PWM2CH	PWM2 计数器高字节	FFA0H	-									x000,0000
PWM2CL	PWM2 计数器低字节	FFA1H									0000,0000	
PWM2CKS	PWM2 时钟选择	FFA2H	-	-	-	SELT2	PWM_PS[3:0]				xxx0,0000	
PWM2TADCH	PWM2 触发 ADC 计数高字节	FFA3H	-									x000,0000

PWM2TADCL	PWM2 触发 ADC 计数低字节	FFA4H										0000,0000
PWM2IF	PWM2 中断标志寄存器	FFA5H	C7IF	C6IF	CSIF	C4IF	C3IF	C2IF	C1IF	C0IF		0000,0000
PWM2FDCR	PWM2 异常检测控制寄存器	FFA6H	INVCM	INVIO	ENFD	FLTFLO	EFDI	FDCMP	FDIO	FDIF		0000,0000
PWM20T1H	PWM20T1 计数值高字节	FFB0H	-									x000,0000
PWM20T1L	PWM20T1 计数值低字节	FFB1H										0000,0000
PWM20T2H	PWM20T2 计数值高字节	FFB2H	-									x000,0000
PWM20T2L	PWM20T2 计数值低字节	FFB3H										0000,0000
PWM20CR	PWM20 控制寄存器	FFB4H	ENO	INI	-	-		ENI	ENT2I	ENT1I		00xx,x000
PWM20HLD	PWM20 电平保持控制寄存器	FFB5H	-	-	-	-	-	-	HLDH	HLDL		xxxx,xx00
PWM21T1H	PWM21T1 计数值高字节	FFB8H	-									x000,0000
PWM21T1L	PWM21T1 计数值低字节	FFB9H										0000,0000
PWM21T2H	PWM21T2 计数值高字节	FFBAH	-									x000,0000
PWM21T2L	PWM21T2 计数值低字节	FFBBH										0000,0000
PWM21CR	PWM21 控制寄存器	FFBCH	ENO	INI	-	-		ENI	ENT2I	ENT1I		00xx,x000
PWM21HLD	PWM21 电平保持控制寄存器	FFBDH	-	-	-	-	-	-	HLDH	HLDL		xxxx,xx00
PWM22T1H	PWM22T1 计数值高字节	FFCOH	-									x000,0000
PWM22T1L	PWM22T1 计数值低字节	FFC1H										0000,0000
PWM22T2H	PWM22T2 计数值高字节	FFC2H	-									x000,0000
PWM22T2L	PWM22T2 计数值低字节	FFC3H										0000,0000
PWM22CR	PWM22 控制寄存器	FFC4H	ENO	INI	-	-		ENI	ENT2I	ENT1I		00xx,x000
PWM22HLD	PWM22 电平保持控制寄存器	FFC5H	-	-	-	-	-	-	HLDH	HLDL		xxxx,xx00
PWM23T1H	PWM23T1 计数值高字节	FFC8H	-									x000,0000
PWM23T1L	PWM23T1 计数值低字节	FFC9H										0000,0000
PWM23T2H	PWM23T2 计数值高字节	FFCAH	-									x000,0000
PWM23T2L	PWM23T2 计数值低字节	FFCBH										0000,0000
PWM23CR	PWM23 控制寄存器	FFCCH	ENO	INI	-	-		ENI	ENT2I	ENT1I		00xx,x000
PWM23HLD	PWM23 电平保持控制寄存器	FFCDH	-	-	-	-	-	-	HLDH	HLDL		xxxx,xx00
PWM24T1H	PWM24T1 计数值高字节	FFD0H	-									x000,0000
PWM24T1L	PWM24T1 计数值低字节	FFD1H										0000,0000
PWM24T2H	PWM24T2 计数值高字节	FFD2H	-									x000,0000
PWM24T2L	PWM24T2 计数值低字节	FFD3H										0000,0000
PWM24CR	PWM24 控制寄存器	FFD4H	ENO	INI	-	-		ENI	ENT2I	ENT1I		00xx,x000
PWM24HLD	PWM24 电平保持控制寄存器	FFD5H	-	-	-	-	-	-	HLDH	HLDL		xxxx,xx00
PWM25T1H	PWM25T1 计数值高字节	FFD8H	-									x000,0000
PWM25T1L	PWM25T1 计数值低字节	FFD9H										0000,0000
PWM25T2H	PWM25T2 计数值高字节	FFDAH	-									x000,0000
PWM25T2L	PWM25T2 计数值低字节	FFDBH										0000,0000
PWM25CR	PWM25 控制寄存器	FFDCH	ENO	INI	-	-		ENI	ENT2I	ENT1I		00xx,x000
PWM25HLD	PWM25 电平保持控制寄存器	FFDDH	-	-	-	-	-	-	HLDH	HLDL		xxxx,xx00
PWM26T1H	PWM26T1 计数值高字节	FFE0H	-									x000,0000
PWM26T1L	PWM26T1 计数值低字节	FFE1H										0000,0000
PWM26T2H	PWM26T2 计数值高字节	FFE2H	-									x000,0000
PWM26T2L	PWM26T2 计数值低字节	FFE3H										0000,0000
PWM26CR	PWM26 控制寄存器	FFE4H	ENO	INI	-	-		ENI	ENT2I	ENT1I		00xx,x000

PWM26HLD	PWM26 电平保持控制寄存器	FFE5H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM27T1H	PWM27T1 计数值高字节	FFE8H	-								x000,0000
PWM27T1L	PWM27T1 计数值低字节	FFE9H									0000,0000
PWM27T2H	PWM27T2 计数值高字节	FFEAH	-								x000,0000
PWM27T2L	PWM27T2 计数值低字节	FFEBH									0000,0000
PWM27CR	PWM27 控制寄存器	FFECH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM27HLD	PWM27 电平保持控制寄存器	FFEDH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM3CH	PWM3 计数器高字节	FC00H	-								x000,0000
PWM3CL	PWM3 计数器低字节	FC01H									0000,0000
PWM3CKS	PWM3 时钟选择	FC02H	-	-	-	SELT2					xxx0,0000
PWM3IF	PWM3 中断标志寄存器	FC05H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWM3FDCR	PWM3 异常检测控制寄存器	FC06H	INVCMP	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF	0000,0000
PWM30T1H	PWM30T1 计数值高字节	FC10H	-								x000,0000
PWM30T1L	PWM30T1 计数值低字节	FC11H									0000,0000
PWM30T2H	PWM30T2 计数值高字节	FC12H	-								x000,0000
PWM30T2L	PWM30T2 计数值低字节	FC13H									0000,0000
PWM30CR	PWM30 控制寄存器	FC14H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM30HLD	PWM30 电平保持控制寄存器	FC15H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM31T1H	PWM31T1 计数值高字节	FC18H	-								x000,0000
PWM31T1L	PWM31T1 计数值低字节	FC19H									0000,0000
PWM31T2H	PWM31T2 计数值高字节	FC1AH	-								x000,0000
PWM31T2L	PWM31T2 计数值低字节	FC1BH									0000,0000
PWM31CR	PWM31 控制寄存器	FC1CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM31HLD	PWM31 电平保持控制寄存器	FC1DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM32T1H	PWM32T1 计数值高字节	FC20H	-								x000,0000
PWM32T1L	PWM32T1 计数值低字节	FC21H									0000,0000
PWM32T2H	PWM32T2 计数值高字节	FC22H	-								x000,0000
PWM32T2L	PWM32T2 计数值低字节	FC23H									0000,0000
PWM32CR	PWM32 控制寄存器	FC24H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM32HLD	PWM32 电平保持控制寄存器	FC25H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM33T1H	PWM33T1 计数值高字节	FC28H	-								x000,0000
PWM33T1L	PWM33T1 计数值低字节	FC29H									0000,0000
PWM33T2H	PWM33T2 计数值高字节	FC2AH	-								x000,0000
PWM33T2L	PWM33T2 计数值低字节	FC2BH									0000,0000
PWM33CR	PWM33 控制寄存器	FC2CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM33HLD	PWM33 电平保持控制寄存器	FC2DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM34T1H	PWM34T1 计数值高字节	FC30H	-								x000,0000
PWM34T1L	PWM34T1 计数值低字节	FC31H									0000,0000
PWM34T2H	PWM34T2 计数值高字节	FC32H	-								x000,0000
PWM34T2L	PWM34T2 计数值低字节	FC33H									0000,0000
PWM34CR	PWM34 控制寄存器	FC34H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM34HLD	PWM34 电平保持控制寄存器	FC35H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM35T1H	PWM35T1 计数值高字节	FC38H	-								x000,0000
PWM35T1L	PWM35T1 计数值低字节	FC39H									0000,0000

PWM35T2H	PWM35T2 计数值高字节	FC3AH	-									x000,0000				
PWM35T2L	PWM35T2 计数值低字节	FC3BH									0000,0000					
PWM35CR	PWM35 控制寄存器	FC3CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	0xx,x000					
PWM35HLD	PWM35 电平保持控制寄存器	FC3DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00					
PWM36T1H	PWM36T1 计数值高字节	FC40H	-									x000,0000				
PWM36T1L	PWM36T1 计数值低字节	FC41H									0000,0000					
PWM36T2H	PWM36T2 计数值高字节	FC42H	-									x000,0000				
PWM36T2L	PWM36T2 计数值低字节	FC43H									0000,0000					
PWM36CR	PWM36 控制寄存器	FC44H	ENO	INI	-	-		ENI	ENT2I	ENT1I	0xx,x000					
PWM36HLD	PWM36 电平保持控制寄存器	FC45H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00					
PWM37T1H	PWM37T1 计数值高字节	FC48H	-									x000,0000				
PWM37T1L	PWM37T1 计数值低字节	FC49H									0000,0000					
PWM37T2H	PWM37T2 计数值高字节	FC4AH	-									x000,0000				
PWM37T2L	PWM37T2 计数值低字节	FC4BH									0000,0000					
PWM37CR	PWM37 控制寄存器	FC4CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	0xx,x000					
PWM37HLD	PWM37 电平保持控制寄存器	FC4DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00					
PWM4CH	PWM4 计数器高字节	FC50H	-									x000,0000				
PWM4CL	PWM4 计数器低字节	FC51H									0000,0000					
PWM4CKS	PWM4 时钟选择	FC52H	-	-	-	SELT2	PWM_PS[3:0]				xxx0,0000					
PWM4TADCH	PWM4 触发 ADC 计数高字节	FC53H	-									x000,0000				
PWM4TADCL	PWM4 触发 ADC 计数低字节	FC54H									0000,0000					
PWM4IF	PWM4 中断标志寄存器	FC55H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000					
PWM4FDCR	PWM4 异常检测控制寄存器	FC56H	INVCM	INVIO	ENFD	FLTFLO	EFDI	FDCMP	FDIO	FDIF	0000,0000					
PWM40T1H	PWM40T1 计数值高字节	FC60H	-									x000,0000				
PWM40T1L	PWM40T1 计数值低字节	FC61H									0000,0000					
PWM40T2H	PWM40T2 计数值高字节	FC62H	-									x000,0000				
PWM40T2L	PWM40T2 计数值低字节	FC63H									0000,0000					
PWM40CR	PWM40 控制寄存器	FC64H	ENO	INI	-	-		ENI	ENT2I	ENT1I	0xx,x000					
PWM40HLD	PWM40 电平保持控制寄存器	FC65H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00					
PWM41T1H	PWM41T1 计数值高字节	FC68H	-									x000,0000				
PWM41T1L	PWM41T1 计数值低字节	FC69H									0000,0000					
PWM41T2H	PWM41T2 计数值高字节	FC6AH	-									x000,0000				
PWM41T2L	PWM41T2 计数值低字节	FC6BH									0000,0000					
PWM41CR	PWM41 控制寄存器	FC6CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	0xx,x000					
PWM41HLD	PWM41 电平保持控制寄存器	FC6DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00					
PWM42T1H	PWM42T1 计数值高字节	FC70H	-									x000,0000				
PWM42T1L	PWM42T1 计数值低字节	FC71H									0000,0000					
PWM42T2H	PWM42T2 计数值高字节	FC72H	-									x000,0000				
PWM42T2L	PWM42T2 计数值低字节	FC73H									0000,0000					
PWM42CR	PWM42 控制寄存器	FC74H	ENO	INI	-	-		ENI	ENT2I	ENT1I	0xx,x000					
PWM42HLD	PWM42 电平保持控制寄存器	FC75H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00					
PWM43T1H	PWM43T1 计数值高字节	FC78H	-									x000,0000				
PWM43T1L	PWM43T1 计数值低字节	FC79H									0000,0000					
PWM43T2H	PWM43T2 计数值高字节	FC7AH	-									x000,0000				

PWM43T2L	PWM43T2 计数值低字节	FC7BH										0000,0000	
PWM43CR	PWM43 控制寄存器	FC7CH	ENO	INI	-	-		ENI	ENT2I	ENT1I		00xx,x000	
PWM43HLD	PWM43 电平保持控制寄存器	FC7DH	-	-	-	-	-	-	HLDH	HLDL		xxxx,xx00	
PWM44T1H	PWM44T1 计数值高字节	FC80H	-										x000,0000
PWM44T1L	PWM44T1 计数值低字节	FC81H										0000,0000	
PWM44T2H	PWM44T2 计数值高字节	FC82H	-										x000,0000
PWM44T2L	PWM44T2 计数值低字节	FC83H										0000,0000	
PWM44CR	PWM44 控制寄存器	FC84H	ENO	INI	-	-		ENI	ENT2I	ENT1I		00xx,x000	
PWM44HLD	PWM44 电平保持控制寄存器	FC85H	-	-	-	-	-	-	HLDH	HLDL		xxxx,xx00	
PWM45T1H	PWM45T1 计数值高字节	FC88H	-										x000,0000
PWM45T1L	PWM45T1 计数值低字节	FC89H										0000,0000	
PWM45T2H	PWM45T2 计数值高字节	FC8AH	-										x000,0000
PWM45T2L	PWM45T2 计数值低字节	FC8BH										0000,0000	
PWM45CR	PWM45 控制寄存器	FC8CH	ENO	INI	-	-		ENI	ENT2I	ENT1I		00xx,x000	
PWM45HLD	PWM45 电平保持控制寄存器	FC8DH	-	-	-	-	-	-	HLDH	HLDL		xxxx,xx00	
PWM46T1H	PWM46T1 计数值高字节	FC90H	-										x000,0000
PWM46T1L	PWM46T1 计数值低字节	FC91H										0000,0000	
PWM46T2H	PWM46T2 计数值高字节	FC92H	-										x000,0000
PWM46T2L	PWM46T2 计数值低字节	FC93H										0000,0000	
PWM46CR	PWM46 控制寄存器	FC94H	ENO	INI	-	-		ENI	ENT2I	ENT1I		00xx,x000	
PWM46HLD	PWM46 电平保持控制寄存器	FC95H	-	-	-	-	-	-	HLDH	HLDL		xxxx,xx00	
PWM47T1H	PWM47T1 计数值高字节	FC98H	-										x000,0000
PWM47T1L	PWM47T1 计数值低字节	FC99H										0000,0000	
PWM47T2H	PWM47T2 计数值高字节	FC9AH	-										x000,0000
PWM47T2L	PWM47T2 计数值低字节	FC9BH										0000,0000	
PWM47CR	PWM47 控制寄存器	FC9CH	ENO	INI	-	-		ENI	ENT2I	ENT1I		00xx,x000	
PWM47HLD	PWM47 电平保持控制寄存器	FC9DH	-	-	-	-	-	-	HLDH	HLDL		xxxx,xx00	
PWM5CH	PWM5 计数器高字节	FCA0H	-										x000,0000
PWM5CL	PWM5 计数器低字节	FCA1H										0000,0000	
PWM5CKS	PWM5 时钟选择	FCA2H	-	-	-	SELT2	PWM_PS[3:0]					xxx0,0000	
PWM5IF	PWM5 中断标志寄存器	FCA5H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF		0000,0000	
PWM5FDCR	PWM5 异常检测控制寄存器	FCA6H	INVCM	INVIO	ENFD	FLTFLO	-	FDCMP	FDIO	FDIF		0000,0000	
PWM50T1H	PWM50T1 计数值高字节	FCB0H	-										x000,0000
PWM50T1L	PWM50T1 计数值低字节	FCB1H										0000,0000	
PWM50T2H	PWM50T2 计数值高字节	FCB2H	-										x000,0000
PWM50T2L	PWM50T2 计数值低字节	FCB3H										0000,0000	
PWM50CR	PWM50 控制寄存器	FCB4H	ENO	INI	-	-		ENI	ENT2I	ENT1I		00xx,x000	
PWM50HLD	PWM50 电平保持控制寄存器	FCB5H	-	-	-	-	-	-	HLDH	HLDL		xxxx,xx00	
PWM51T1H	PWM51T1 计数值高字节	FCB8H	-										x000,0000
PWM51T1L	PWM51T1 计数值低字节	FCB9H										0000,0000	
PWM51T2H	PWM51T2 计数值高字节	FCBAH	-										x000,0000
PWM51T2L	PWM51T2 计数值低字节	FCBBH										0000,0000	
PWM51CR	PWM51 控制寄存器	FCBCB	ENO	INI	-	-		ENI	ENT2I	ENT1I		00xx,x000	
PWM51HLD	PWM51 电平保持控制寄存器	FCBDH	-	-	-	-	-	-	HLDH	HLDL		xxxx,xx00	

PWM52T1H	PWM52T1 计数值高字节	FCC0H	-								x000,0000
PWM52T1L	PWM52T1 计数值低字节	FCC1H								0000,0000	
PWM52T2H	PWM52T2 计数值高字节	FCC2H	-								x000,0000
PWM52T2L	PWM52T2 计数值低字节	FCC3H								0000,0000	
PWM52CR	PWM52 控制寄存器	FCC4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM52HLD	PWM52 电平保持控制寄存器	FCC5H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM53T1H	PWM53T1 计数值高字节	FCC8H	-								x000,0000
PWM53T1L	PWM53T1 计数值低字节	FCC9H								0000,0000	
PWM53T2H	PWM53T2 计数值高字节	FCCAH	-								x000,0000
PWM53T2L	PWM53T2 计数值低字节	FCCBH								0000,0000	
PWM53CR	PWM53 控制寄存器	FCCCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM53HLD	PWM53 电平保持控制寄存器	FCCDH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM54T1H	PWM54T1 计数值高字节	FCD0H	-								x000,0000
PWM54T1L	PWM54T1 计数值低字节	FCD1H								0000,0000	
PWM54T2H	PWM54T2 计数值高字节	FCD2H	-								x000,0000
PWM54T2L	PWM54T2 计数值低字节	FCD3H								0000,0000	
PWM54CR	PWM54 控制寄存器	FCD4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM54HLD	PWM54 电平保持控制寄存器	FCD5H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM55T1H	PWM55T1 计数值高字节	FCD8H	-								x000,0000
PWM55T1L	PWM55T1 计数值低字节	FCD9H								0000,0000	
PWM55T2H	PWM55T2 计数值高字节	FCDAH	-								x000,0000
PWM55T2L	PWM55T2 计数值低字节	FCDBH								0000,0000	
PWM55CR	PWM55 控制寄存器	FCDCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM55HLD	PWM55 电平保持控制寄存器	FCDDH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM56T1H	PWM56T1 计数值高字节	FCE0H	-								x000,0000
PWM56T1L	PWM56T1 计数值低字节	FCE1H								0000,0000	
PWM56T2H	PWM56T2 计数值高字节	FCE2H	-								x000,0000
PWM56T2L	PWM56T2 计数值低字节	FCE3H								0000,0000	
PWM56CR	PWM56 控制寄存器	FCE4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM56HLD	PWM56 电平保持控制寄存器	FCE5H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM57T1H	PWM57T1 计数值高字节	FCE8H	-								x000,0000
PWM57T1L	PWM57T1 计数值低字节	FCE9H								0000,0000	
PWM57T2H	PWM57T2 计数值高字节	FCEAH	-								x000,0000
PWM57T2L	PWM57T2 计数值低字节	FCEBH								0000,0000	
PWM57CR	PWM57 控制寄存器	FCECH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM57HLD	PWM57 电平保持控制寄存器	FCEDH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00

18.1.1 增强型 PWM 全局配置寄存器 (PWMSET)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMSET	F1H	ENGLBSET	PWMRST	ENPWM5	ENPWM4	ENPWM3	ENPWM2	ENPWM1	ENPWM0

ENGLBSET: 全局设置功能控制。

0: 6 组 PWM 采用各自独立设置方式。

每组 PWM 分别使用 PWMCFG01/PWMCFG23/PWMCFG45 相应的控制位就绪独立配置。

1: 6 组 PWM 采用统一设置方式。

每组 PWM 均采用 PWMCFG01 中的 PWM0 的设置进行配置。

PWMRST: 软件复位 6 组 PWM。

0: 无效

1: 复位所有 PWM 的 XFR 寄存器, 但不复位 SFR。(需要软件清零)

ENPWM5: PWM5 使能位 (包括 PWM50~PWM54)。

0: 关闭 PWM5

1: 使能 PWM5

ENPWM4: PWM4 使能位 (包括 PWM40~PWM47)。

0: 关闭 PWM4

1: 使能 PWM4

ENPWM3: PWM3 使能位 (包括 PWM30~PWM37)。

0: 关闭 PWM3

1: 使能 PWM3

ENPWM2: PWM2 使能位 (包括 PWM20~PWM27)。

0: 关闭 PWM2

1: 使能 PWM2

ENPWM1: PWM1 使能位 (包括 PWM10~PWM17)。

0: 关闭 PWM1

1: 使能 PWM1

ENPWM0: PWM0 使能位 (包括 PWM00~PWM07)。

0: 关闭 PWM0

1: 使能 PWM0

18.1.2 增强型 PWM 配置寄存器 (PWMCFGn)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMCFG01	F6H	PWM1CBIF	EPWM1CBI	FLTPS0	PWM1CEN	PWM0CBIF	EPWM0CBI	ENPWM0TA	PWM0CEN
PWMCFG23	F7H	PWM3CBIF	EPWM3CBI	FLTPS1	PWM3CEN	PWM2CBIF	EPWM2CBI	ENPWM2TA	PWM2CEN
PWMCFG45	FEH	PWM5CBIF	EPWM5CBI	FLTPS2	PWM5CEN	PWM4CBIF	EPWM4CBI	ENPWM4TA	PWM4CEN

PWMnCBIF: PWMn 计数器归零中断标志位。 $(n=0\sim 5)$

当 15 位的 PWMn 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。

EPWMnCBI: PWMn 计数器归零中断使能位。 $(n=0\sim 5)$

0: 关闭 PWMn 计数器归零中断 (PWMMnCBIF 依然会被硬件置位)

1: 使能 PWMn 计数器归零中断

PWMnCEN: PWMn 波形发生器开始计数。 $(n=0\sim 5)$

0: PWMn 停止计数

1: PWMn 计数器开始计数

关于 PWMMnCEN 控制位的重要说明:

- PWMMnCEN 一旦被使能后, 内部的 PWMMn 计数器会立即开始计数, 并与 T1/T2 的值进行比较。所以 PWMMnCEN 必须在其他所有的 PWM 设置 (包括 T1/T2 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 PWMMnCEN 位。
- 在 PWMMn 计数器计数的过程中, PWMMnCEN 控制位被关闭时, PWMMn 计数会立即停止, 当再次使能 PWMMnCEN 控制位时, PWMMn 的计数会从 0 开始重新计数, 而不会记忆 PWMMn 停止计数前的计数值

- 特别注意: 当 PWMnCEN 由 0 变为 1 时, 内部的 PWM 计数器是从之前的不确定值归零后重新开始计数, 所以此时会产生立即产生一个归零中断, 当用户需要使用 PWM 的归零中断时, 需特别注意这一点, 即第一个归零中断并不是真正的 PWM 周期记满后归零所产生的。

EPWMnTA: PWMn 是否与 ADC 关联。(n=0、2、4)

0: PWMn 与 ADC 不关联

1: PWMn 与 ADC 相关联。

允许在 PWMn 周期中某个时间点触发 A/D 转换, 使用 PWMnTADCH 和 PWMnTADCL 进行设置。

(注意: 需要同时使能 ADC_CONTR 寄存器中的 ADC_POWER 位和 ADC_EPWMT 位, PWM 只是会自动将 ADC_START 置 1, 只有 PWM0、PWM2、PWM4 可以触发 ADC)

FLTPS0、FLTPS1、FLTPS2: 外部异常检测脚选择控制位

FLTPS2	FLTPS1	FLTPS0	PWM0/PWM1/PWM3/PWM5 外部异常检测脚	PWM2 外部异常检测脚	PWM4 外部异常检测脚
0	0	0	PWMFLT (P3.5)	PWMFLT (P3.5)	PWMFLT (P3.5)
0	0	1	PWMFLT (P3.5)	PWMFLT2 (P0.6)	PWMFLT3 (P0.7)
0	1	0	PWMFLT (P3.5)	PWMFLT3 (P0.7)	PWMFLT2 (P0.6)
0	1	1	PWMFLT2 (P0.6)	PWMFLT2 (P0.6)	PWMFLT2 (P0.6)
1	0	0	PWMFLT2 (P0.6)	PWMFLT (P3.5)	PWMFLT3 (P0.7)
1	0	1	PWMFLT2 (P0.6)	PWMFLT3 (P0.7)	PWMFLT (P3.5)
1	1	0	PWMFLT3 (P0.7)	PWMFLT (P3.5)	PWMFLT2 (P0.6)
1	1	1	PWMFLT3 (P0.7)	PWMFLT2 (P0.6)	PWMFLT (P3.5)

18.1.3 PWM 中断标志寄存器 (PWMMnIF)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWM0IF	FF05H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF
PWM1IF	FF55H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF
PWM2IF	FFA5H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF
PWM3IF	FC05H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF
PWM4IF	FC55H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF
PWM5IF	FCA5H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF

CiIF: PWMn 的第 i 通道中断标志位。(n=0~5; i=0~7)

可设置在各路 PWM 的 T1 和 T2。当所设置的点发生匹配事件时, 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。

18.1.4 PWM 异常检测控制寄存器 (PWMMnFDCR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWM0FDCR	FF06H	INV_CMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF
PWM1FDCR	FF56H	INV_CMP	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF
PWM2FDCR	FFA6H	INV_CMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF
PWM3FDCR	FC06H	INV_CMP	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF
PWM4FDCR	FC56H	INV_CMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF
PWM5FDCR	FCA6H	INV_CMP	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF

INV_CMP: 比较器结果异常信号处理

0: 比较器结果由低变高为异常信号

1: 比较器结果由高变低为异常信号

INVIO: 外部 PWMFLT 端口异常信号处理

0: 外部 PWMFLT 端口信号由低变高为异常信号

1: 外部 PWMFLT 端口信号由高变低为异常信号

(注: 每组 PWM 的外部异常检测 PWMFLT 端口由 FLTPS0、FLTPS1、FLTPS 进行选择)

ENFD: PWMn 外部异常检测控制位。($n=0 \sim 5$)

0: 关闭 PWMn 外部异常检测功能

1: 使能 PWMn 外部异常检测功能

FLTFLIO: 发生 PWMn 外部异常时对 PWMn 输出口控制位。($n=0 \sim 5$)

0: 发生 PWMn 外部异常时, PWMn 的输出口不作任何改变

1: 发生 PWMn 外部异常时, PWMn 的输出口立即被设置为高阻输入模式。

(注: 只有 ENO=1 所对应的端口才会被强制悬空)

EFDI: PWMn 异常检测中断使能位。($n=0, 2, 4$)

0: 关闭 PWMn 异常检测中断 (FDIF 依然会被硬件置位)

1: 使能 PWMn 异常检测中断

FDCMP: 比较器输出异常检测使能位。($n=0 \sim 5$)

0: 比较器与 PWMn 无关

1: 设定 PWMn 异常检测源为比较器输出 (异常类型由 INVCMR 设定)

FDIO: PWMFLT 端口电平异常检测使能位。($n=0 \sim 5$)

0: PWMFLT 端口电平与 PWMn 无关

1: 设定 PWMn 异常检测源为 PWMFLT 端口 (异常类型由 INVIO 设定)

FDIF: PWMn 异常检测中断标志位。($n=0 \sim 5$)

当发生 PWMn 异常时, 硬件自动将此位置 1。当 EFDI==1 时, 程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

(注: 只有 PWM0、PWM2 和 PWM4 才会进中断, PWM1、PWM3、PWM5 有异常检测功能, 但不进中断服务程序)

18.1.5 PWM 计数器寄存器 (PWMnCH, PWMnCL)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWM0CH	FF00H	-							
PWM0CL	FF01H								
PWM1CH	FF50H	-							
PWM1CL	FF51H								
PWM2CH	FFA0H	-							
PWM2CL	FFA1H								
PWM3CH	FC00H	-							
PWM3CL	FC01H								
PWM4CH	FC50H	-							
PWM4CL	FC51H								
PWM5CH	FCA0H	-							
PWM5CL	FCA1H								

PWMnCH: PWMn 计数器周期值的高 7 位。($n=0 \sim 5$)

PWMnCL: PWMn 计数器周期值的低 8 位。($n=0 \sim 5$)

PWMn 计数器为一个 15 位的寄存器, 可设定 1~32767 之间的任意值作为 PWMn 的周期。PWMn 波形发生器内部的计数器从 0 开始计数, 每个 PWMn 时钟周期递增 1, 当内部计数器的计数值达到

[PWMnCH, PWMnCL]所设定的 PWMn 周期时, PWMn 波形发生器内部的计数器将会从 0 重新开始计数, 硬件会自动将 PWMn 归零中断标志位 PWMnCBIF 置 1, 若 EPWMnCBI=1, 程序将跳转到相应中断入口执行中断服务程序。

18.1.6 PWM 时钟选择寄存器 (PWMMnCKS), 输出频率计算公式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWM0CKS	FF02H	-	-	-	SELT2	PWM_PS[3:0]			
PWM1CKS	FF52H	-	-	-	SELT2	PWM_PS[3:0]			
PWM2CKS	FFA2H	-	-	-	SELT2	PWM_PS[3:0]			
PWM3CKS	FC02H	-	-	-	SELT2	PWM_PS[3:0]			
PWM4CKS	FC52H	-	-	-	SELT2	PWM_PS[3:0]			
PWM5CKS	FCA2H	-	-	-	SELT2	PWM_PS[3:0]			

SELT2: PWMn 时钟源选择。($n=0 \sim 5$)

0: PWMn 时钟源为系统时钟经分频器分频之后的时钟

1: PWMn 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]	PWMn 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 输出频率计算公式

6 组 PWM 的输出频率计算公式相同, 且每组可设置不同的频率。

时钟源选择 (SELT2)	PWM输出频率计算公式
SELT2=0 (系统时钟)	$\text{PWM输出频率} = \frac{\text{系统工作频率SYSclk}}{(\text{PWM_PS} + 1) \times ([\text{PWMMnCH}, \text{PWMMnCL}] + 1)}$
SELT2=1 (定时器2的溢出脉冲)	$\text{PWM输出频率} = \frac{\text{定时器2的溢出脉冲频率}}{([\text{PWMMnCH}, \text{PWMMnCL}] + 1)}$

18.1.7 PWM 触发 ADC 计数器寄存器 (PWMMnTADC)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWM0TADCH	FF03H	-							
PWM0TADCL	FF04H								
PWM2TADCH	FFA3H	-							

PWM2TADCL	FFA4H									
PWM4TADCH	FC53H	-								
PWM4TADCL	FC54H	-								

PWM_nTADCH: PWM_n 触发 ADC 时间点的高 7 位。(n=0、2、4)

PWM_nTADCL: PWM_n 触发 ADC 时间点的低 8 位。(n=0、2、4)

若 EPWM_nTA = 1 且 ADC_POWER=1,ADC_EPWMT=1 时, 在 PWM_n 的计数周期中, 当 PWM_n 的内部计数值与{PWM_nTADCH, PWM_nTADCL}所组成一个 15 位的寄存器的值相等时, 硬件自动触发 A/D 转换。

18.1.8 PWM 电平输出设置计数值寄存器 (PWMM_nT1, PWMM_nT2)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWM00T1H	FF10H	-							
PWM00T1L	FF11H								
PWM00T2H	FF12H	-							
PWM00T2L	FF13H								
PWM01T1H	FF18H	-							
PWM01T1L	FF19H								
PWM01T2H	FF1AH	-							
PWM01T2L	FF1BH								
PWM02T1H	FF20H	-							
PWM02T1L	FF21H								
PWM02T2H	FF22H	-							
PWM02T2L	FF23H								
PWM03T1H	FF28H	-							
PWM03T1L	FF29H								
PWM03T2H	FF2AH	-							
PWM03T2L	FF2BH								
PWM04T1H	FF30H	-							
PWM04T1L	FF31H								
PWM04T2H	FF32H	-							
PWM04T2L	FF33H								
PWM05T1H	FF38H	-							
PWM05T1L	FF39H								
PWM05T2H	FF3AH	-							
PWM05T2L	FF3BH								
PWM06T1H	FF40H	-							
PWM06T1L	FF41H								
PWM06T2H	FF42H	-							
PWM06T2L	FF43H								
PWM07T1H	FF48H	-							
PWM07T1L	FF49H								
PWM07T2H	FF4AH	-							
PWM07T2L	FF4BH								
PWM10T1H	FF60H	-							

PWM10T1L	FF61H		
PWM10T2H	FF62H	-	
PWM10T2L	FF63H		
PWM11T1H	FF68H	-	
PWM11T1L	FF69H		
PWM11T2H	FF6AH	-	
PWM11T2L	FF6BH		
PWM12T1H	FF70H	-	
PWM12T1L	FF71H		
PWM12T2H	FF72H	-	
PWM12T2L	FF73H		
PWM13T1H	FF78H	-	
PWM13T1L	FF79H		
PWM13T2H	FF7AH	-	
PWM13T2L	FF7BH		
PWM14T1H	FF80H	-	
PWM14T1L	FF81H		
PWM14T2H	FF82H	-	
PWM14T2L	FF83H		
PWM15T1H	FF88H	-	
PWM15T1L	FF89H		
PWM15T2H	FF8AH	-	
PWM15T2L	FF8BH		
PWM16T1H	FF90H	-	
PWM16T1L	FF91H		
PWM16T2H	FF92H	-	
PWM16T2L	FF93H		
PWM17T1H	FF98H	-	
PWM17T1L	FF99H		
PWM17T2H	FF9AH	-	
PWM17T2L	FF9BH		
PWM20T1H	FFB0H	-	
PWM20T1L	FFB1H		
PWM20T2H	FFB2H	-	
PWM20T2L	FFB3H		
PWM21T1H	FFB8H	-	
PWM21T1L	FFB9H		
PWM21T2H	FFBAH	-	
PWM21T2L	FFBBH		
PWM22T1H	FFC0H	-	
PWM22T1L	FFC1H		
PWM22T2H	FFC2H	-	
PWM22T2L	FFC3H		
PWM23T1H	FFC8H	-	

PWM23T1L	FFC9H		
PWM23T2H	FFCAH	-	
PWM23T2L	FFCBH		
PWM24T1H	FFD0H	-	
PWM24T1L	FFD1H		
PWM24T2H	FFD2H	-	
PWM24T2L	FFD3H		
PWM25T1H	FFD8H	-	
PWM25T1L	FFD9H		
PWM25T2H	FFDAH	-	
PWM25T2L	FFDBH		
PWM26T1H	FFE0H	-	
PWM26T1L	FFE1H		
PWM26T2H	FFE2H	-	
PWM26T2L	FFE3H		
PWM27T1H	FFE8H	-	
PWM27T1L	FFE9H		
PWM27T2H	FFEAH	-	
PWM27T2L	FFEBH		
PWM30T1H	FC10H	-	
PWM30T1L	FC11H		
PWM30T2H	FC12H	-	
PWM30T2L	FC13H		
PWM31T1H	FC18H	-	
PWM31T1L	FC19H		
PWM31T2H	FC1AH	-	
PWM31T2L	FC1BH		
PWM32T1H	FC20H	-	
PWM32T1L	FC21H		
PWM32T2H	FC22H	-	
PWM32T2L	FC23H		
PWM33T1H	FC28H	-	
PWM33T1L	FC29H		
PWM33T2H	FC2AH	-	
PWM33T2L	FC2BH		
PWM34T1H	FC30H	-	
PWM34T1L	FC31H		
PWM34T2H	FC32H	-	
PWM34T2L	FC33H		
PWM35T1H	FC38H	-	
PWM35T1L	FC39H		
PWM35T2H	FC3AH	-	
PWM35T2L	FC3BH		
PWM36T1H	FC40H	-	

PWM36T1L	FC41H		
PWM36T2H	FC42H	-	
PWM36T2L	FC43H		
PWM37T1H	FC48H	-	
PWM37T1L	FC49H		
PWM37T2H	FC4AH	-	
PWM37T2L	FC4BH		
PWM40T1H	FC60H	-	
PWM40T1L	FC61H		
PWM40T2H	FC62H	-	
PWM40T2L	FC63H		
PWM41T1H	FC68H	-	
PWM41T1L	FC69H		
PWM41T2H	FC6AH	-	
PWM41T2L	FC6BH		
PWM42T1H	FC70H	-	
PWM42T1L	FC71H		
PWM42T2H	FC72H	-	
PWM42T2L	FC73H		
PWM43T1H	FC78H	-	
PWM43T1L	FC79H		
PWM43T2H	FC7AH	-	
PWM43T2L	FC7BH		
PWM44T1H	FC80H	-	
PWM44T1L	FC81H		
PWM44T2H	FC82H	-	
PWM44T2L	FC83H		
PWM45T1H	FC88H	-	
PWM45T1L	FC89H		
PWM45T2H	FC8AH	-	
PWM45T2L	FC8BH		
PWM46T1H	FC90H	-	
PWM46T1L	FC91H		
PWM46T2H	FC92H	-	
PWM46T2L	FC93H		
PWM47T1H	FC98H	-	
PWM47T1L	FC99H		
PWM47T2H	FC9AH	-	
PWM47T2L	FC9BH		
PWM50T1H	FCB0H	-	
PWM50T1L	FCB1H		
PWM50T2H	FCB2H	-	
PWM50T2L	FCB3H		
PWM51T1H	FCB8H	-	

PWM51T1L	FCB9H							
PWM51T2H	FCBAH	-						
PWM51T2L	FCBBH							
PWM52T1H	FCC0H	-						
PWM52T1L	FCC1H							
PWM52T2H	FCC2H	-						
PWM52T2L	FCC3H							
PWM53T1H	FCC8H	-						
PWM53T1L	FCC9H							
PWM53T2H	FCCAH	-						
PWM53T2L	FCCBH							
PWM54T1H	FCD0H	-						
PWM54T1L	FCD1H							
PWM54T2H	FCD2H	-						
PWM54T2L	FCD3H							
PWM55T1H	FCD8H	-						
PWM55T1L	FCD9H							
PWM55T2H	FCDAH	-						
PWM55T2L	FCDBH							
PWM56T1H	FCE0H	-						
PWM56T1L	FCE1H							
PWM56T2H	FCE2H	-						
PWM56T2L	FCE3H							
PWM57T1H	FCE8H	-						
PWM57T1L	FCE9H							
PWM57T2H	FCEAH	-						
PWM57T2L	FCEBH							

PWM_niT1H: PWM_n 的通道 i 的 T1 计数器值的高 7 位。($n=0\sim5, i=0\sim7$)

PWM_niT1L: PWM_n 的通道 i 的 T1 计数器值的低 8 位。($n=0\sim5, i=0\sim7$)

PWM_niT2H: PWM_n 的通道 i 的 T2 计数器值的高 7 位。($n=0\sim5, i=0\sim7$)

PWM_niT2L: PWM_n 的通道 i 的 T2 计数器值的低 8 位。($n=0\sim5, i=0\sim7$)

每组 PWM 的每个通道的{PWM_niT1H, PWM_niT1L}和{PWM_niT2H, PWM_niT2L}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 波形的两个触发点。在 PWM_n 的计数周期中，当 PWM_n 的内部计数值与所设置的 T1 的值{PWM_niT1H, PWM_niT1L}相等时，PWM 的输出**低电平**；当 PWM_n 的内部计数值与 T2 的值{PWM_niT2H, PWM_niT2L}相等时，PWM 的输出**高电平**。

注意：当{PWM_niT1H, PWM_niT1L}与{PWM_niT2H, PWM_niT2L}的值设置相等时，若 PWM 的内部计数值与所设置的 T1/T2 的值相等，则会固定输出低电平。

18.1.9 PWM 通道控制寄存器 (PWMnCR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWM00CR	FF14H	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM01CR	FF1CH	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM02CR	FF24H	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM03CR	FF2CH	ENO	INI	-	-	-	ENI	ENT2I	ENT1I

PWM04CR	FF34H	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM05CR	FF3CH	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM06CR	FF44H	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM07CR	FF4CH	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM10CR	FF64H	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM11CR	FF6CH	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM12CR	FF74H	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM13CR	FF7CH	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM14CR	FF84H	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM15CR	FF8CH	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM16CR	FF94H	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM17CR	FF9CH	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM20CR	FFB4H	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM21CR	FFBCH	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM22CR	FFC4H	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM23CR	FFCCH	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM24CR	FFD4H	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM25CR	FFDCH	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM26CR	FFE4H	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM27CR	FFECH	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM30CR	FC14H	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM31CR	FC1CH	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM32CR	FC24H	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM33CR	FC2CH	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM34CR	FC34H	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM35CR	FC3CH	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM36CR	FC44H	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM37CR	FC4CH	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM40CR	FC64H	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM41CR	FC6CH	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM42CR	FC74H	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM43CR	FC7CH	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM44CR	FC84H	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM45CR	FC8CH	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM46CR	FC94H	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM47CR	FC9CH	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM50CR	FCB4H	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM51CR	FCBCH	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM52CR	FCC4H	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM53CR	FCCCH	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM54CR	FCD4H	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM55CR	FCDCH	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM56CR	FCE4H	ENO	INI	-	-	-	ENI	ENT2I	ENTII
PWM57CR	FCECH	ENO	INI	-	-	-	ENI	ENT2I	ENTII

ENO: PWMni 输出使能位。(n=0~5, i=0~7)

0: 第 n 路 PWM 的 i 通道相应 PWMni 端口为 GPIO

1: 第 n 路 PWM 的 i 通道相应 PWMni 端口为 PWM 输出口, 受 PWMn 波形发生器控制

INI: 设置 PWMni 输出端口的初始电平。(n=0~5, i=0~7)

0: 第 n 路 PWM 的 i 通道初始电平为低电平

1: 第 n 路 PWM 的 i 通道初始电平为高电平

ENI: 第 n 路 PWM 的 i 通道中断使能控制位。(n=0~5, i=0~7)

0: 关闭第 n 路 PWM 的 i 通道的 PWM 中断

1: 使能第 n 路 PWM 的 i 通道的 PWM 中断

ENT2I: 第 n 路 PWM 的 i 通道在第 2 个触发点中断使能控制位。(n=0~5, i=0~7)

0: 关闭第 n 路 PWM 的 i 通道在第 2 个触发点中断

1: 使能第 n 路 PWM 的 i 通道在第 2 个触发点中断

ENT1I: 第 n 路 PWM 的 i 通道在第 1 个触发点中断使能控制位。(n=0~5, i=0~7)

0: 关闭第 n 路 PWM 的 i 通道在第 1 个触发点中断

1: 使能第 n 路 PWM 的 i 通道在第 1 个触发点中断

18.1.10 PWM 通道电平保持控制寄存器 (PWMnHLD)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWM00HLD	FF15H	-	-	-	-	-	-	HLDH	HLDL
PWM01HLD	FF1DH	-	-	-	-	-	-	HLDH	HLDL
PWM02HLD	FF25H	-	-	-	-	-	-	HLDH	HLDL
PWM03HLD	FF2DH	-	-	-	-	-	-	HLDH	HLDL
PWM04HLD	FF35H	-	-	-	-	-	-	HLDH	HLDL
PWM05HLD	FF3DH	-	-	-	-	-	-	HLDH	HLDL
PWM06HLD	FF45H	-	-	-	-	-	-	HLDH	HLDL
PWM07HLD	FF4DH	-	-	-	-	-	-	HLDH	HLDL
PWM10HLD	FF65H	-	-	-	-	-	-	HLDH	HLDL
PWM11HLD	FF6DH	-	-	-	-	-	-	HLDH	HLDL
PWM12HLD	FF75H	-	-	-	-	-	-	HLDH	HLDL
PWM13HLD	FF7DH	-	-	-	-	-	-	HLDH	HLDL
PWM14HLD	FF85H	-	-	-	-	-	-	HLDH	HLDL
PWM15HLD	FF8DH	-	-	-	-	-	-	HLDH	HLDL
PWM16HLD	FF95H	-	-	-	-	-	-	HLDH	HLDL
PWM17HLD	FF9DH	-	-	-	-	-	-	HLDH	HLDL
PWM20HLD	FFB5H	-	-	-	-	-	-	HLDH	HLDL
PWM21HLD	FFBDH	-	-	-	-	-	-	HLDH	HLDL
PWM22HLD	FFC5H	-	-	-	-	-	-	HLDH	HLDL
PWM23HLD	FFCDH	-	-	-	-	-	-	HLDH	HLDL
PWM24HLD	FFD5H	-	-	-	-	-	-	HLDH	HLDL
PWM25HLD	FFDDH	-	-	-	-	-	-	HLDH	HLDL
PWM26HLD	FFE5H	-	-	-	-	-	-	HLDH	HLDL
PWM27HLD	FFEDH	-	-	-	-	-	-	HLDH	HLDL
PWM30HLD	FC15H	-	-	-	-	-	-	HLDH	HLDL
PWM31HLD	FC1DH	-	-	-	-	-	-	HLDH	HLDL

PWM32HLD	FC25H	-	-	-	-	-	-	HLDH	HLDL
PWM33HLD	FC2DH	-	-	-	-	-	-	HLDH	HLDL
PWM34HLD	FC35H	-	-	-	-	-	-	HLDH	HLDL
PWM35HLD	FC3DH	-	-	-	-	-	-	HLDH	HLDL
PWM36HLD	FC45H	-	-	-	-	-	-	HLDH	HLDL
PWM37HLD	FC4DH	-	-	-	-	-	-	HLDH	HLDL
PWM40HLD	FC65H	-	-	-	-	-	-	HLDH	HLDL
PWM41HLD	FC6DH	-	-	-	-	-	-	HLDH	HLDL
PWM42HLD	FC75H	-	-	-	-	-	-	HLDH	HLDL
PWM43HLD	FC7DH	-	-	-	-	-	-	HLDH	HLDL
PWM44HLD	FC85H	-	-	-	-	-	-	HLDH	HLDL
PWM45HLD	FC8DH	-	-	-	-	-	-	HLDH	HLDL
PWM46HLD	FC95H	-	-	-	-	-	-	HLDH	HLDL
PWM47HLD	FC9DH	-	-	-	-	-	-	HLDH	HLDL
PWM50HLD	FCB5H	-	-	-	-	-	-	HLDH	HLDL
PWM51HLD	FCBDH	-	-	-	-	-	-	HLDH	HLDL
PWM52HLD	FCC5H	-	-	-	-	-	-	HLDH	HLDL
PWM53HLD	FCCDH	-	-	-	-	-	-	HLDH	HLDL
PWM54HLD	FCD5H	-	-	-	-	-	-	HLDH	HLDL
PWM55HLD	FCDDH	-	-	-	-	-	-	HLDH	HLDL
PWM56HLD	FCE5H	-	-	-	-	-	-	HLDH	HLDL
PWM57HLD	FCEDH	-	-	-	-	-	-	HLDH	HLDL

HLDH: 第 n 路 PWM 的 i 通道强制输出高电平控制位。($n=0 \sim 5, i=0 \sim 7$)

0: 第 n 路 PWM 的 i 通道正常输出

1: 第 n 路 PWM 的 i 通道强制输出高电平

HLDL: 第 n 路 PWM 的 i 通道强制输出低电平控制位。($n=0 \sim 5, i=0 \sim 7$)

0: 第 n 路 PWM 的 i 通道正常输出

1: 第 n 路 PWM 的 i 通道强制输出低电平

18.2 范例程序

18.2.1 输出任意周期和任意占空比的波形

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr P_SW2 = 0xba;

sfr PWMSET = 0xF1;
sfr PWMCFG01 = 0xF6;
sfr PWMCFG23 = 0xF7;
sfr PWMCFG45 = 0xFE;

#define PWM0C (*(unsigned int volatile xdata *)0xFF00)
#define PWM0CH (*(unsigned char volatile xdata *)0xFF00)
#define PWM0CL (*(unsigned char volatile xdata *)0xFF01)
#define PWM0CKS (*(unsigned char volatile xdata *)0xFF02)
#define PWM0TADC (*(unsigned int volatile xdata *)0xFF03)
#define PWM0TADCH (*(unsigned char volatile xdata *)0xFF03)
#define PWM0TADCL (*(unsigned char volatile xdata *)0xFF04)
#define PWM0IF (*(unsigned char volatile xdata *)0xFF05)
#define PWM0FDCR (*(unsigned char volatile xdata *)0xFF06)
#define PWM00T1 (*(unsigned int volatile xdata *)0xFF10)
#define PWM00T1H (*(unsigned char volatile xdata *)0xFF10)
#define PWM00T1L (*(unsigned char volatile xdata *)0xFF11)
#define PWM00T2H (*(unsigned char volatile xdata *)0xFF12)
#define PWM00T2 (*(unsigned int volatile xdata *)0xFF12)
#define PWM00T2L (*(unsigned char volatile xdata *)0xFF13)
#define PWM00CR (*(unsigned char volatile xdata *)0xFF14)
#define PWM00HLD (*(unsigned char volatile xdata *)0xFF15)

sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P4M1 = 0xb3;
sfr P4M0 = 0xb4;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
```

```

P2M1 = 0x00;
P3M0 = 0x00;
P3M1 = 0x00;
P4M0 = 0x00;
P4M1 = 0x00;
P5M0 = 0x00;
P5M1 = 0x00;

PWMSET = 0x01;                                //使能 PWM0 模块 (必须先使能模块后面的设置才有效)

P_SW2 = 0x80;
PWM0CKS = 0x00;                               //PWM0 时钟为系统时钟
PWM0C = 0x1000;                               //设置 PWM0 周期为 1000H 个 PWM 时钟
PWM00T1= 0x0100;                            //在计数值为 100H 地方 PWM00 通道输出低电平
PWM00T2= 0x0500;                            //在计数值为 500H 地方 PWM00 通道输出高电平
PWM00CR= 0x80;                                //使能 PWM00 输出
P_SW2 = 0x00;

PWMCFG01 = 0x01;                                //启动 PWM0 模块

while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

P_SW2	DATA	0BAH
PWMSET	DATA	0F1H
PWMCFG01	DATA	0F6H
PWMCFG23	DATA	0F7H
PWMCFG45	DATA	0FEH
PWM0CH	EQU	0FF00H
PWM0CL	EQU	0FF01H
PWM0CKS	EQU	0FF02H
PWM0TADCH	EQU	0FF03H
PWM0TADCL	EQU	0FF04H
PWM0IF	EQU	0FF05H
PWM0FDCR	EQU	0FF06H
PWM00T1H	EQU	0FF10H
PWM00T1L	EQU	0FF11H
PWM00T2H	EQU	0FF12H
PWM00T2L	EQU	0FF13H
PWM00CR	EQU	0FF14H
PWM00HLD	EQU	0FF15H
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH

<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, #5FH</i>
<i>MOV</i>	<i>P0M0, #00H</i>
<i>MOV</i>	<i>P0M1, #00H</i>
<i>MOV</i>	<i>P1M0, #00H</i>
<i>MOV</i>	<i>P1M1, #00H</i>
<i>MOV</i>	<i>P2M0, #00H</i>
<i>MOV</i>	<i>P2M1, #00H</i>
<i>MOV</i>	<i>P3M0, #00H</i>
<i>MOV</i>	<i>P3M1, #00H</i>
<i>MOV</i>	<i>P4M0, #00H</i>
<i>MOV</i>	<i>P4M1, #00H</i>
<i>MOV</i>	<i>P5M0, #00H</i>
<i>MOV</i>	<i>P5M1, #00H</i>
 <i>MOV</i>	<i>PWMSET,#01H</i>
	;使能 PWM0 模块 (必须先使能模块后面的设置才有效)
<i>MOV</i>	<i>P_SW2,#80H</i>
<i>CLR</i>	<i>A</i>
<i>MOV</i>	<i>DPTR,#PWM0CKS</i>
<i>MOVX</i>	<i>@DPTR,A</i>
<i>MOV</i>	<i>A,#10H</i>
<i>MOV</i>	<i>DPTR,#PWM0CH</i>
<i>MOVX</i>	<i>@DPTR,A</i>
<i>MOV</i>	<i>A,#00H</i>
<i>MOV</i>	<i>DPTR,#PWM0CL</i>
<i>MOVX</i>	<i>@DPTR,A</i>
<i>MOV</i>	<i>A,#0IH</i>
<i>MOV</i>	<i>DPTR,#PWM00T1H</i>
<i>MOVX</i>	<i>@DPTR,A</i>
<i>MOV</i>	<i>A,#00H</i>
<i>MOV</i>	<i>DPTR,#PWM00T1L</i>
<i>MOVX</i>	<i>@DPTR,A</i>
<i>MOV</i>	<i>A,#05H</i>
<i>MOV</i>	<i>DPTR,#PWM00T2H</i>
<i>MOVX</i>	<i>@DPTR,A</i>
<i>MOV</i>	<i>A,#00H</i>
<i>MOV</i>	<i>DPTR,#PWM00T2L</i>
<i>MOVX</i>	<i>@DPTR,A</i>
<i>MOV</i>	<i>A,#80H</i>
<i>MOV</i>	<i>DPTR,#PWM00CR</i>
<i>MOVX</i>	<i>@DPTR,A</i>
<i>MOV</i>	<i>P_SW2,#00H</i>
 <i>MOV</i>	<i>PWMCFG01,#01H</i>
	;启动 PWM0 模块
<i>JMP</i>	<i>\$</i>
 <i>END</i>	

18.2.2 两路 PWM 实现互补对称带死区控制的波形

C 语言代码

//测试工作频率为 11.0592MHz

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

sfr P_SW2 = 0xba;

sfr PWMSET = 0xF1;
sfr PWMCFG01 = 0xF6;
sfr PWMCFG23 = 0xF7;
sfr PWMCFG45 = 0xFE;

#define PWM0C      (*(unsigned int volatile xdata *)0xFF00)
#define PWM0CH     (*(unsigned char volatile xdata *)0xFF00)
#define PWM0CL      (*(unsigned char volatile xdata *)0xFF01)
#define PWM0CKS     (*(unsigned char volatile xdata *)0xFF02)
#define PWM0TADC    (*(unsigned int volatile xdata *)0xFF03)
#define PWM0TADCH   (*(unsigned char volatile xdata *)0xFF03)
#define PWM0TADCL   (*(unsigned char volatile xdata *)0xFF04)
#define PWM0IF      (*(unsigned char volatile xdata *)0xFF05)
#define PWM0FDCR    (*(unsigned char volatile xdata *)0xFF06)
#define PWM00T1      (*(unsigned int volatile xdata *)0xFF10)
#define PWM00T1H     (*(unsigned char volatile xdata *)0xFF10)
#define PWM00T1L     (*(unsigned char volatile xdata *)0xFF11)
#define PWM00T2      (*(unsigned int volatile xdata *)0xFF12)
#define PWM00T2H     (*(unsigned char volatile xdata *)0xFF12)
#define PWM00T2L     (*(unsigned char volatile xdata *)0xFF13)
#define PWM00CR      (*(unsigned char volatile xdata *)0xFF14)
#define PWM00HLD     (*(unsigned char volatile xdata *)0xFF15)
#define PWM0IT1      (*(unsigned int volatile xdata *)0xFF18)
#define PWM0IT1H     (*(unsigned char volatile xdata *)0xFF18)
#define PWM0IT1L     (*(unsigned char volatile xdata *)0xFF19)
#define PWM0IT2      (*(unsigned int volatile xdata *)0xFF1A)
#define PWM0IT2H     (*(unsigned char volatile xdata *)0xFF1A)
#define PWM0IT2L     (*(unsigned char volatile xdata *)0xFF1B)
#define PWM01CR      (*(unsigned char volatile xdata *)0xFF1C)
#define PWM01HLD     (*(unsigned char volatile xdata *)0xFF1D)

sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P4M1 = 0xb3;
sfr P4M0 = 0xb4;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
}
```

```

P2M1 = 0x00;
P3M0 = 0x00;
P3M1 = 0x00;
P4M0 = 0x00;
P4M1 = 0x00;
P5M0 = 0x00;
P5M1 = 0x00;

PWMSET = 0x01;                                //使能 PWM0 模块 (必须先使能模块后面的设置才有效)

P_SW2 = 0x80;
PWM0CKS = 0x00;                               //PWM0 时钟为系统时钟
PWM0C = 0x0800;                               //设置 PWM0 周期为 0800H 个 PWM 时钟
PWM00T1= 0x0100;                            //PWM00 在计数值为 100H 地方输出低电平
PWM00T2= 0x0700;                            //PWM00 在计数值为 700H 地方输出高电平
PWM01T2= 0x0080;                            //PWM01 在计数值为 0080H 地方输出高电平
PWM01T1= 0x0780;                            //PWM01 在计数值为 0780H 地方输出低电平
PWM00CR= 0x80;                                //使能 PWM00 输出
PWM01CR= 0x80;                                //使能 PWM01 输出
P_SW2 = 0x00;

PWMCFG01 = 0x01;                                //启动 PWM0 模块

while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

P_SW2	DATA	0BAH
PWMSET	DATA	0FIH
PWMCFG01	DATA	0F6H
PWMCFG23	DATA	0F7H
PWMCFG45	DATA	0FEH
PWM0CH	EQU	0FF00H
PWM0CL	EQU	0FF01H
PWM0CKS	EQU	0FF02H
PWM0TADCH	EQU	0FF03H
PWM0TADCL	EQU	0FF04H
PWM0IF	EQU	0FF05H
PWM0FDCR	EQU	0FF06H
PWM00T1H	EQU	0FF10H
PWM00T1L	EQU	0FF11H
PWM00T2H	EQU	0FF12H
PWM00T2L	EQU	0FF13H
PWM00CR	EQU	0FF14H
PWM00HLD	EQU	0FF15H
PWM01T1H	EQU	0FF18H
PWM01T1L	EQU	0FF19H
PWM01T2H	EQU	0FF1AH
PWM01T2L	EQU	0FF1BH
PWM01CR	EQU	0FF1CH
PWM01HLD	EQU	0FF1DH
P0M1	DATA	093H
P0M0	DATA	094H

<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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, #5FH</i>	
<i>MOV</i>	<i>P0M0, #00H</i>	
<i>MOV</i>	<i>P0M1, #00H</i>	
<i>MOV</i>	<i>P1M0, #00H</i>	
<i>MOV</i>	<i>P1M1, #00H</i>	
<i>MOV</i>	<i>P2M0, #00H</i>	
<i>MOV</i>	<i>P2M1, #00H</i>	
<i>MOV</i>	<i>P3M0, #00H</i>	
<i>MOV</i>	<i>P3M1, #00H</i>	
<i>MOV</i>	<i>P4M0, #00H</i>	
<i>MOV</i>	<i>P4M1, #00H</i>	
<i>MOV</i>	<i>P5M0, #00H</i>	
<i>MOV</i>	<i>P5M1, #00H</i>	
<i>MOV</i>	<i>PWMSET,#01H</i>	;使能 PWM0 模块 (必须先使能模块后面的设置才有效)
<i>MOV</i>	<i>P_SW2,#80H</i>	
<i>CLR</i>	<i>A</i>	
<i>MOV</i>	<i>DPTR,#PWM0CKS</i>	
<i>MOVX</i>	<i>@DPTR,A</i>	;PWM0 时钟为系统时钟
<i>MOV</i>	<i>A,#08H</i>	
<i>MOV</i>	<i>DPTR,#PWM0CH</i>	;设置 PWM0 周期为 0800H 个 PWM 时钟
<i>MOVX</i>	<i>@DPTR,A</i>	
<i>MOV</i>	<i>A,#00H</i>	
<i>MOV</i>	<i>DPTR,#PWM0CL</i>	
<i>MOVX</i>	<i>@DPTR,A</i>	
<i>MOV</i>	<i>A,#01H</i>	
<i>MOV</i>	<i>DPTR,#PWM00T1H</i>	;PWM0 在计数值为 0100H 地方输出低电平
<i>MOVX</i>	<i>@DPTR,A</i>	
<i>MOV</i>	<i>A,#00H</i>	
<i>MOV</i>	<i>DPTR,#PWM00T1L</i>	
<i>MOVX</i>	<i>@DPTR,A</i>	
<i>MOV</i>	<i>A,#07H</i>	
<i>MOV</i>	<i>DPTR,#PWM00T2H</i>	;PWM0 在计数值为 0700H 地方输出高电平
<i>MOVX</i>	<i>@DPTR,A</i>	
<i>MOV</i>	<i>A,#00H</i>	
<i>MOV</i>	<i>DPTR,#PWM00T2L</i>	
<i>MOVX</i>	<i>@DPTR,A</i>	
<i>MOV</i>	<i>A,#00H</i>	
<i>MOV</i>	<i>DPTR,#PWM01T2H</i>	;PWM01 在计数值为 0080H 地方输出高电平
<i>MOVX</i>	<i>@DPTR,A</i>	
<i>MOV</i>	<i>A,#80H</i>	
<i>MOV</i>	<i>DPTR,#PWM01T2L</i>	

```

MOVX      @DPTR,A
MOV       A,#07H
MOV       DPTR,#PWM01T1H      ;PWM01 在计数值为0780H 地方输出低电平
MOVX      @DPTR,A
MOV       A,#80H
MOV       DPTR,#PWM01T1L
MOVX      @DPTR,A
MOV       A,#080H
MOV       DPTR,#PWM00CR      ;使能 PWM00 输出
MOVX      @DPTR,A
MOV       A,#80H
MOV       DPTR,#PWM01CR      ;使能 PWM01 输出
MOVX      @DPTR,A
MOV       P_SW2,#00H

MOV       PWMCFG01,#01H      ;启动 PWM0 模块
JMP       $

```

END

18.2.3 PWM 实现渐变灯（呼吸灯）

C 语言代码

//测试工作频率为11.0592MHz

```

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

#define CYCLE      0x1000

sfr    P_SW2      = 0xba;
sfr    PWMSET     = 0xF1;
sfr    PWMCFG01   = 0xF6;
sfr    PWMCFG23   = 0xF7;
sfr    PWMCFG45   = 0xFE;

#define PWM0C      (*(unsigned int volatile xdata *)0xFF00)
#define PWM0CH     (*(unsigned char volatile xdata *)0xFF00)
#define PWM0CL      (*(unsigned char volatile xdata *)0xFF01)
#define PWM0CKS    (*(unsigned char volatile xdata *)0xFF02)
#define PWM0TADC   (*(unsigned int volatile xdata *)0xFF03)
#define PWM0TADCH  (*(unsigned char volatile xdata *)0xFF03)
#define PWM0TADCL  (*(unsigned char volatile xdata *)0xFF04)
#define PWM0IF      (*(unsigned char volatile xdata *)0xFF05)
#define PWM0FDCR   (*(unsigned char volatile xdata *)0xFF06)
#define PWM00T1     (*(unsigned int volatile xdata *)0xFF10)
#define PWM00T1H    (*(unsigned char volatile xdata *)0xFF10)
#define PWM00T1L    (*(unsigned char volatile xdata *)0xFF11)
#define PWM00T2H    (*(unsigned char volatile xdata *)0xFF12)
#define PWM00T2     (*(unsigned int volatile xdata *)0xFF12)
#define PWM00T2L    (*(unsigned char volatile xdata *)0xFF13)
#define PWM00CR     (*(unsigned char volatile xdata *)0xFF14)
#define PWM00HLD    (*(unsigned char volatile xdata *)0xFF15)

```

```
sfr P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P1M1      = 0x91;
sfr P1M0      = 0x92;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

void PWM0_Isr() interrupt 22
{
    static bit dir = 1;
    static int val = 0;

    if (PWMCFG01 & 0x08)
    {
        PWMCFG01 &= ~0x08;                                //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;
        PWM00T2 = val;
        _pop_(P_SW2);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    PWMSET = 0x01;                                    //使能 PWM0 模块 (必须先使能模块后面的设置才有效)

    P_SW2 = 0x80;                                     // PWM0 时钟为系统时钟
    PWM0CKS = 0x00;                                   // 设置 PWM0 周期
    PWM0C = CYCLE;
```

```

PWM00T1= 0x0000;
PWM00T2= 0x0001;
PWM00CR= 0x80;                                //使能 PWM00 输出
P_SW2 = 0x00;

PWMCFG01 = 0x05;                          //启动 PWM0 模块并使能 PWM0 中断
EA = I;

while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

CYCLE	EQU	1000H
P_SW2	DATA	0BAH
PWMSET	DATA	0FIH
PWMCFG01	DATA	0F6H
PWMCFG23	DATA	0F7H
PWMCFG45	DATA	0FEH
PWM0CH	EQU	0FF00H
PWM0CL	EQU	0FF01H
PWM0CKS	EQU	0FF02H
PWM0TADCH	EQU	0FF03H
PWM0TADCL	EQU	0FF04H
PWM0IF	EQU	0FF05H
PWM0FDSCR	EQU	0FF06H
PWM00T1H	EQU	0FF10H
PWM00T1L	EQU	0FF11H
PWM00T2H	EQU	0FF12H
PWM00T2L	EQU	0FF13H
PWM00CR	EQU	0FF14H
PWM00HLD	EQU	0FF15H
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH
DIR	BIT	20H.0
VALL	DATA	21H
VALH	DATA	22H
ORG	0000H	
LJMP	MAIN	
ORG	00B3H	
LJMP	PWM0ISR	

ORG	0100H	
PWM0ISR:		
PUSH	ACC	
PUSH	PSW	
PUSH	DPL	
PUSH	DPH	
PUSH	P_SW2	
MOV	P_SW2,#80H	
MOV	A,PWMCFG01	
JNB	ACC.3,ISREXIT	
ANL	PWMCFG01,#NOT 08H	; 清中断标志
JNB	DIR,PWMDN	
PWMUP:		
MOV	A,VALL	
ADD	A,#I	
MOV	VALL,A	
MOV	A,VALH	
ADDC	A,#0	
MOV	VALH,A	
CJNE	A,#HIGH CYCLE,SETPWM	
MOV	A,VALL	
CJNE	A,#LOW CYCLE,SETPWM	
CLR	DIR	
JMP	SETPWM	
PWMDN:		
MOV	A,VALL	
ADD	A,#0FFH	
MOV	VALL,A	
MOV	A,VALH	
ADDC	A,#0FFH	
MOV	VALH,A	
JNZ	SETPWM	
MOV	A,VALL	
CJNE	A,#I,SETPWM	
SETB	DIR	
SETPWM:		
MOV	A,VALH	
MOV	DPTR,#PWM00T2H	
MOVX	@DPTR,A	
MOV	A,VALL	
MOV	DPTR,#PWM00T2L	
MOVX	@DPTR,A	
ISREXIT:		
POP	P_SW2	
POP	DPH	
POP	DPL	
POP	PSW	
POP	ACC	
RETI		
MAIN:		
MOV	SP, #5FH	
MOV	P0M0, #00H	
MOV	P0M1, #00H	
MOV	P1M0, #00H	
MOV	P1M1, #00H	
MOV	P2M0, #00H	

```

MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

SETB    DIR
MOV      VALH,#00H
MOV      VALL,#01H

MOV      PWMSET,#01H           ;使能 PWM0 模块（必须先使能模块后面的设置才有效）

MOV      P_SW2,#80H
CLR      A
MOV      DPTR,#PWM0CKS
MOVX   @DPTR,A               ;PWM0 时钟为系统时钟
MOV      A,#HIGH CYCLE
MOV      DPTR,#PWM0CH
MOVX   @DPTR,A               ;设置 PWM0 周期
MOV      A,#LOW CYCLE
MOV      DPTR,#PWM0CL
MOVX   @DPTR,A
MOV      A,#00H
MOV      DPTR,#PWM00T1H
MOVX   @DPTR,A
MOV      A,#00H
MOV      DPTR,#PWM00T1L
MOVX   @DPTR,A
MOV      A,VALH
MOV      DPTR,#PWM00T2H
MOVX   @DPTR,A
MOV      A,VALL
MOV      DPTR,#PWM00T2L
MOVX   @DPTR,A
MOV      A,#80H
MOV      DPTR,#PWM00CR           ;使能 PWM0 输出
MOVX   @DPTR,A
MOV      P_SW2,#00H

MOV      PWMCFG01,#05H          ;启动 PWM0 模块并使能 PWM0 中断
SETB    EA
JMP     $

END

```

18.2.4 使用 PWM 触发 ADC 转换

C 语言代码

//测试工作频率为11.0592MHz

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

```
sfr P_SW2 = 0xba;
sfr PWMSET = 0xF1;
sfr PWMCFG01 = 0xF6;
sfr PWMCFG23 = 0xF7;
sfr PWMCFG45 = 0xFE;

#define PWM0C      (*(unsigned int volatile xdata *)0xFF00)
#define PWM0CH     (*(unsigned char volatile xdata *)0xFF00)
#define PWM0CL      (*(unsigned char volatile xdata *)0xFF01)
#define PWM0CKS     (*(unsigned char volatile xdata *)0xFF02)
#define PWM0TADC    (*(unsigned int volatile xdata *)0xFF03)
#define PWM0TADCH   (*(unsigned char volatile xdata *)0xFF03)
#define PWM0TADCL   (*(unsigned char volatile xdata *)0xFF04)
#define PWM0IF      (*(unsigned char volatile xdata *)0xFF05)
#define PWM0FDCR    (*(unsigned char volatile xdata *)0xFF06)
#define PWM00T1      (*(unsigned int volatile xdata *)0xFF10)
#define PWM00T1H     (*(unsigned char volatile xdata *)0xFF10)
#define PWM00T1L     (*(unsigned char volatile xdata *)0xFF11)
#define PWM00T2H     (*(unsigned char volatile xdata *)0xFF12)
#define PWM00T2      (*(unsigned int volatile xdata *)0xFF12)
#define PWM00T2L     (*(unsigned char volatile xdata *)0xFF13)
#define PWM00CR      (*(unsigned char volatile xdata *)0xFF14)
#define PWM00HLD     (*(unsigned char volatile xdata *)0xFF15)

sfr ADC_CONTR = 0xbc;
#define ADC_POWER 0x80
#define ADC_START 0x40
#define ADC_FLAG 0x20
#define ADC_EPWMT 0x10
sfr ADC_RES = 0xbd;
sfr ADC_RESL = 0xbe;

sbit EADC = IE^5;

sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P4M1 = 0xb3;
sfr P4M0 = 0xb4;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;

void delay()
{
    int i;
    for (i=0; i<100; i++);
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
```

```

P1M1 = 0x01;
P2M0 = 0x00;
P2M1 = 0x00;
P3M0 = 0x00;
P3M1 = 0x00;
P4M0 = 0x00;
P4M1 = 0x00;
P5M0 = 0x00;
P5M1 = 0x00;

ADC_CONTR = ADC_POWER / ADC_EPWMT / 0; //选择P1.0 为ADC 输入通道
delay(); //等待ADC 电源稳定
EADC = 1;

PWMSET = 0x01; //使能 PWM0 模块 (必须先使能模块后面的设置才有效)

P_SW2 = 0x80;
PWM0CKS = 0x00; //PWM0 时钟为系统时钟
PWM0C = 0x1000; //设置 PWM0 周期为 1000H 个 PWM 时钟
PWM00T1= 0x0100; //在计数值为 100H 地方 PWM00 通道输出低电平
PWM00T2= 0x0500; //在计数值为 500H 地方 PWM00 通道输出高电平
PWM0TADC = 0x0200; //设置 ADC 触发点
PWM00CR= 0x80; //使能 PWM00 输出
P_SW2 = 0x00;

PWMCFG01 = 0x07; //启动 PWM0 模块并使能 PWM0 中断以及 ADC 触发
EA = 1;

while (1);
}

void pwm0_isr() interrupt 22
{
    if (PWMCFG01 & 0x08)
    {
        PWMCFG01 &= ~0x08;
    }
}

void ADC_ISR() interrupt 5
{
    ADC_CONTR &= ~ADC_FLAG;
}

```

汇编代码

; 测试工作频率为 11.0592MHz

P_SW2	DATA	0BAH
PWMSET	DATA	0FIH
PWMCFG01	DATA	0F6H
PWMCFG23	DATA	0F7H
PWMCFG45	DATA	0FEH
PWM0CH	EQU	0FF00H
PWM0CL	EQU	0FF01H
PWM0CKS	EQU	0FF02H
PWM0TADCH	EQU	0FF03H

<i>PWM0TADCL</i>	<i>EQU</i>	<i>0FF04H</i>
<i>PWM0IF</i>	<i>EQU</i>	<i>0FF05H</i>
<i>PWM0FDCR</i>	<i>EQU</i>	<i>0FF06H</i>
<i>PWM00T1H</i>	<i>EQU</i>	<i>0FF10H</i>
<i>PWM00T1L</i>	<i>EQU</i>	<i>0FF11H</i>
<i>PWM00T2H</i>	<i>EQU</i>	<i>0FF12H</i>
<i>PWM00T2L</i>	<i>EQU</i>	<i>0FF13H</i>
<i>PWM00CR</i>	<i>EQU</i>	<i>0FF14H</i>
<i>PWM00HLD</i>	<i>EQU</i>	<i>0FF15H</i>
<i>ADC_CONTR</i>	<i>DATA</i>	<i>0BCH</i>
<i>ADC_POWER</i>	<i>EQU</i>	<i>080H</i>
<i>ADC_START</i>	<i>EQU</i>	<i>040H</i>
<i>ADC_FLAG</i>	<i>EQU</i>	<i>020H</i>
<i>ADC_EPWMT</i>	<i>EQU</i>	<i>010H</i>
<i>ADC_RES</i>	<i>DATA</i>	<i>0BDH</i>
<i>ADC_RESL</i>	<i>DATA</i>	<i>0BEH</i>
<i>EADC</i>	<i>BIT</i>	<i>IE.5</i>
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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>00B3H</i>
	<i>LJMP</i>	<i>PWM0ISR</i>
	<i>ORG</i>	<i>0100H</i>
<i>ADCISR:</i>	<i>ANL</i>	<i>ADC_CONTR,#NOT ADC_FLAG</i>
	<i>RETI</i>	
<i>PWM0ISR:</i>		
	<i>PUSH</i>	<i>ACC</i>
	<i>MOV</i>	<i>A,PWMCFG01</i>
	<i>JNB</i>	<i>ACC.3,ISREXIT</i>
	<i>ANL</i>	<i>PWMCFG01,#NOT 08H</i>
<i>ISREXIT:</i>	<i>POP</i>	<i>ACC</i>
	<i>RETI</i>	
<i>MAIN:</i>		
	<i>MOV</i>	<i>SP, #5FH</i>
	<i>MOV</i>	<i>P0M0, #00H</i>
	<i>MOV</i>	<i>P0M1, #00H</i>
	<i>MOV</i>	<i>P1M0, #00H</i>

```

MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

MOV      ADC_CONTR,# ADC_POWER /ADC_EPWM0T
SETB    EADC

MOV      PWMSET,#01H ;使能 PWM0 模块（必须先使能模块后面的设置才有效）

MOV      P_SW2,#80H
CLR      A
MOV      DPTR,#PWM0CKS
MOVX   @DPTR,A ;PWM0 时钟为系统时钟
MOV      DPTR,#PWM0CH ;设置 PWM0 周期为 1000H 个 PWM 时钟
MOVX   @DPTR,A
MOV      A,#00H
MOV      DPTR,#PWM0CL
MOVX   @DPTR,A
MOV      A,#01H
MOV      DPTR,#PWM00T1H ;在计数值为 100H 地方 PWM00 通道输出低电平
MOVX   @DPTR,A
MOV      A,#00H
MOV      DPTR,#PWM00T1L
MOVX   @DPTR,A
MOV      A,#05H
MOV      DPTR,#PWM00T2H ;在计数值为 500H 地方 PWM00 通道输出高电平
MOVX   @DPTR,A
MOV      A,#00H
MOV      DPTR,#PWM00T2L
MOVX   @DPTR,A
MOV      A,#02H
MOV      DPTR,#PWM0TADCH ;置 ADC 触发点
MOVX   @DPTR,A
MOV      A,#00H
MOV      DPTR,#PWM0TADCL
MOVX   @DPTR,A
MOV      A,#80H
MOV      DPTR,#PWM00CR ;使能 PWM0 输出
MOVX   @DPTR,A
MOV      P_SW2,#00H

MOV      PWMCFG01,#07H ;启动 PWM0 模块并使能 PWM0 中断以及 ADC 触发
SETB    EA

JMP      $

END

```

18.2.5 产生 3 路相位差 120 度的互补带死区的 PWM 波形

C 语言代码

```
//测试工作频率为24MHz

#include "reg51.h"

#define MAIN_Fosc 24000000L //定义主时钟

sbit P20 = P2^0;
sbit P21 = P2^1;
sbit P22 = P2^2;
sbit P23 = P2^3;
sbit P24 = P2^4;
sbit P25 = P2^5;

sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P_SW2 = 0xBA; //外设端口切换寄存器2
sfr PWMSET = 0xF1; //增强型 PWM 全局配置寄存器
sfr PWMCFG01 = 0xF6; //增强型 PWM 配置寄存器
sfr PWMCFG23 = 0xF7; //增强型 PWM 配置寄存器

#define PWM0C (*(unsigned int volatile xdata *)0xFF00)

#define PWM2CKS (*(unsigned char volatile xdata *)0xffa2)
#define PWM2C (*(unsigned int volatile xdata *)0xffa0)
#define PWM20T1 (*(unsigned int volatile xdata *)0xffb0)
#define PWM20T2 (*(unsigned int volatile xdata *)0xffb2)
#define PWM20CR (*(unsigned char volatile xdata *)0xffb4)
#define PWM20HLD (*(unsigned char volatile xdata *)0xffb5)
#define PWM21T1 (*(unsigned int volatile xdata *)0xffb8)
#define PWM21T2 (*(unsigned int volatile xdata *)0xffba)
#define PWM21CR (*(unsigned char volatile xdata *)0xffbc)
#define PWM21HLD (*(unsigned char volatile xdata *)0xffbd)
#define PWM22T1 (*(unsigned int volatile xdata *)0xffc0)
#define PWM22T2 (*(unsigned int volatile xdata *)0xffc2)
#define PWM22CR (*(unsigned char volatile xdata *)0xffc4)
#define PWM22HLD (*(unsigned char volatile xdata *)0xffc5)
#define PWM23T1 (*(unsigned int volatile xdata *)0xffc8)
#define PWM23T2 (*(unsigned int volatile xdata *)0xffca)
#define PWM23CR (*(unsigned char volatile xdata *)0xffcc)
#define PWM23HLD (*(unsigned char volatile xdata *)0xffcd)
#define PWM24T1 (*(unsigned int volatile xdata *)0xffd0)
#define PWM24T2 (*(unsigned int volatile xdata *)0xffd2)
#define PWM24CR (*(unsigned char volatile xdata *)0xffd4)
#define PWM24HLD (*(unsigned char volatile xdata *)0xffd5)
#define PWM25T1 (*(unsigned int volatile xdata *)0xffd8)
#define PWM25T2 (*(unsigned int volatile xdata *)0xffda)
#define PWM25CR (*(unsigned char volatile xdata *)0xffdc)
#define PWM25HLD (*(unsigned char volatile xdata *)0xffdd)

#define P2n_push_pull(bitn) P2M1 &= ~(bitn), P2M0 |= (bitn)

#define ENO 0x80 /* 1: 允许对应引脚为 PWM, 0: 对应引脚为 GPIO */
#define CKS_IT 0 /* PWM 正常输出 */
#define PWM_Normal 0x00 /* PWM 正常输出 */
```

```

#define ENPWM2      0x04          /* 1: 使能 PWM2(PWM20~PWM27), 0: 关闭 PWM2 */
#define PWM2CEN    0x01          /* 1: PWM2 开始计数, 0: 停止计数 */
#define EAXSFR()     P_SW2 |= 0x80
#define EAXRAM()     P_SW2 &= ~0x80

/***************** 功能说明 *****/
本程序适用于 STC8G2K64S4 系列 STC8G2K64S2 系列。
P2.0+P2.1 P2.2+P2.3 P2.4+P2.5 输出 3 路相位差 120 度的互补带死区 PWM, PWM 频率为 50KHz, 死区时间为 0.5us.
*****/


void PWM2_config(void);

void main(void)
{
    PWM2_config();
    while (1);
}

//=====
// 函数: void PWM2_config(void)
// 描述: PWM 配置函数。
// 参数: none.
// 返回: none.
// 版本: VER1.0
// 日期: 2020-5-17
// 备注:
//=====

void PWM2_config(void)
{
    EAXSFR();                                //访问 XFR. 头文件中的宏
    PWMCFG23 &= 0xf0;                         //允许 P2(P2.0~P2.7) 做 PWM
    PWMSET /= ENPWM2;                          //选择 PWM2 时钟, CKS_TIMER2, CKS_IT ~ CKS_16T
    PWM2CKS = CKS_IT;                         //设置 PWM2 周期 = PWM2C + 1
    PWM2C    = 480;

    PWM20T2  = 12;                            //T2 输出高电平时刻
    PWM20T1  = 160;//800;                     //T1 输出低电平时刻
    PWM20HLD = PWM_Normal;                   //PWM 正常输出
                                                //((PWM_KeepHigh:PWM 强制输出高电平,
                                                //PWM_KeepLow: PWM 强制输出低电平)
                                                //ENO: 允许 PWM 输出
    PWM20CR  = ENO;                           //PWM 输出口设置为推挽输出

    P20 = 0;                                 //T2 输出高电平时刻
    P2n_push_pull(1<<0);                   //T1 输出低电平时刻
                                                //PWM 正常输出
                                                //((PWM_KeepHigh:PWM 强制输出高电平,
                                                //PWM_KeepLow: PWM 强制输出低电平)
                                                //ENO: 允许 PWM 输出
    PWM21CR  = ENO;                           //PWM 输出口设置为推挽输出

    PWM21T2  = 172;//812;                     //T2 输出高电平时刻
    PWM21T1  = 0;                            //T1 输出低电平时刻
    PWM21HLD = PWM_Normal;                   //PWM 正常输出
                                                //((PWM_KeepHigh:PWM 强制输出高电平,
                                                //PWM_KeepLow: PWM 强制输出低电平)
                                                //ENO: 允许 PWM 输出
    PWM21CR  = ENO;                           //PWM 输出口设置为推挽输出

    PWM22T2  = 172;//812;                     //T2 输出高电平时刻
    PWM22T1  = 320;//1600;                    //T1 输出低电平时刻
    PWM22HLD = PWM_Normal;                   //PWM 正常输出
                                                //((PWM_KeepHigh:PWM 强制输出高电平,
                                                //PWM_KeepLow: PWM 强制输出低电平)
                                                //ENO: 允许 PWM 输出
    PWM22CR  = ENO;
}

```

```

P22 = 0;                                //PWM 输出口设置为推挽输出

PWM23T2  = 332;//16I2;
PWM23T1  = 160;//800;
PWM23HLD = PWM_Normal;

PWM23CR  = ENO;
P23 = 0;
P2n_push_pull(1<<2);                  //PWM 输出口设置为推挽输出

PWM24T2  = 332;//16I2;
PWM24T1  = 480;//2400;
PWM24HLD = PWM_Normal;

PWM24CR  = ENO;
P24 = 0;
P2n_push_pull(1<<3);                  //PWM 输出口设置为推挽输出

PWM25T2  = 12;
PWM25T1  = 320;//1600;
PWM25HLD = PWM_Normal;

PWM25CR  = ENO;
P25 = 0;
P2n_push_pull(1<<4);                  //PWM 输出口设置为推挽输出

PWMCFG23 /= PWM2CEN;                   //启动计数器, 开始PWM输出, 初始化最后执行的语句
}

//***** PWM 失效中断函数*****
void PWMFD_int (void) interrupt 23
{
}

```

18.2.6 输出占空比为 100%（固定输出高）和 0%（固定输出低）的 PWM 波形的方法（以 PWM00 为例）

18.2.6.1 方法 1：禁止输出 PWM

禁止 PWM 输出后，对应 IO 称为普通 IO，需要这个 IO 输出高或低，自行设置。

```

PWM00CR /= 0x80;                         //ENO=1: 允许 PWM 输出
delay_ms(5);                             //输出 5ms
P00 = 1;                                 //连续输出高电平
PWM00CR &= ~0x80;                        //ENO=0: 禁止 PWM 输出
delay_ms(5);                             //关闭 PWM, P0.0 输出高电平 5ms

PWM00CR /= 0x80;                         //ENO=1: 允许 PWM 输出
delay_ms(5);                             //输出 5ms

```

```

P00 = 0;                                //连续输出低电平
PWM00CR &= ~0x80;                      //ENO=0:禁止PWM 输出
delay_ms(5);                            //关闭PWM, P0.0 输出高电平5ms

```

18.2.6.2 方法 2: PWM00T2 设置输出高电平时刻 (一般我设置为 0)

PWM00T1 设置输出低电平时刻, PWM00T1-PWM00T2 就是输出高电平时间。

如果将 PWM00T1 设置的数值比周期值大, 则不会输出低电平, 输出 100% 占空比。

18.2.6.3 方法 3: 直接使用 PWMnHLD 寄存器 (重点推荐)

使用“PWM 通道电平保持控制寄存器 PWMnHLD”直接设置输出高或低, 这个寄存器就是专门用于设置连续输出高或低的。

```

PWM00HLD = 0x00;                        //PWM 正常输出
PWM00HLD = 0x01;                        //PWM 输出连续低电平
PWM00HLD = 0x02;                        //PWM 输出连续高电平

```

18.2.7 增强型 PWM-频率可调-脉冲计数

C 语言代码

//测试工作频率为 24MHz

```

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

sfr    P_SW2      = 0xba;
sfr    P4          = 0xC0;
sfr    P5          = 0xC8;
sfr    P6          = 0xE8;
sfr    P7          = 0xF8;

sfr    PWMSET      = 0xF1;
sfr    PWMCFG01    = 0xF6;
sfr    PWMCFG23    = 0xF7;
sfr    PWMCFG45    = 0xFE;

#define PWM0C        (*(unsigned int volatile xdata *)0xFF00)
#define PWM0CH       (*(unsigned char volatile xdata *)0xFF00)
#define PWM0CL       (*(unsigned char volatile xdata *)0xFF01)
#define PWM0CKS      (*(unsigned char volatile xdata *)0xFF02)
#define PWM0TADC     (*(unsigned int volatile xdata *)0xFF03)
#define PWM0TADCH    (*(unsigned char volatile xdata *)0xFF03)
#define PWM0TADCL    (*(unsigned char volatile xdata *)0xFF04)
#define PWM0IF        (*(unsigned char volatile xdata *)0xFF05)
#define PWM0FDCR     (*(unsigned char volatile xdata *)0xFF06)
#define PWM00T1       (*(unsigned int volatile xdata *)0xFF10)
#define PWM00T1H      (*(unsigned char volatile xdata *)0xFF10)
#define PWM00T1L      (*(unsigned char volatile xdata *)0xFF11)

```

```

#define  PWM00T2H  (*(unsigned char volatile xdata *)0xFF12)
#define  PWM00T2  (*(unsigned int volatile xdata *)0xFF12)
#define  PWM00T2L  (*(unsigned char volatile xdata *)0xFF13)
#define  PWM00CR  (*(unsigned char volatile xdata *)0xFF14)
#define  PWM00HLD  (*(unsigned char volatile xdata *)0xFF15)
#define  PWM01CR  (*(unsigned char volatile xdata *)0xFF1C)
#define  PWM02CR  (*(unsigned char volatile xdata *)0xFF24)
#define  PWM01T1  (*(unsigned int volatile xdata *)0xFF18)
#define  PWM01T2  (*(unsigned int volatile xdata *)0xFF1A)
#define  PWM02T1  (*(unsigned int volatile xdata *)0xFF20)
#define  PWM02T2  (*(unsigned int volatile xdata *)0xFF22)

sfr  P0M1      =  0x93;
sfr  P0M0      =  0x94;
sfr  P1M1      =  0x91;
sfr  P1M0      =  0x92;
sfr  P2M1      =  0x95;
sfr  P2M0      =  0x96;
sfr  P3M1      =  0xb1;
sfr  P3M0      =  0xb2;
sfr  P4M1      =  0xb3;
sfr  P4M0      =  0xb4;
sfr  P5M1      =  0xc9;
sfr  P5M0      =  0xca;

#define  MAIN_Fosc  11059200UL

unsigned char Counter;
unsigned int Period;
bit DirFlag;

void delay_ms(unsigned char ms);
void PeriodSet(unsigned int period);

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Period = 0x1000;
    DirFlag = 0;

    PWMSET = 0x01;                                //使能 PWM0 模块(必须先使能模块后面的设置才有效)
    P_SW2 = 0x80;                                  //使能 XFR 访问
    PWM0CKS = 0x00;                                //PWM0 时钟为系统时钟

    PWM00T1= 0x0000;                             //在计数值为 00H 地方 PWM00 通道输出低电平
    PWM00CR= 0x80;                               //使能 PWM00 输出
}

```

```
EA = I;

while (1)
{
    delay_ms(10);
    PeriodSet(Period); //设置周期、占空比
    EA = 0;
    PWMCFG01 = 0x05; //启动 PWM0 模块, 使能计数器归零中断
    _nop_();
    _nop_();
    PWMCFG01 &= ~0x08; //跳过第一个启动归零中断
    EA = I;

    if(DirFlag)
    {
        Period++; //周期递增
        if(Period >= 0x1000)
        {
            DirFlag = 0;
        }
    }
    else
    {
        Period--; //周期递减
        if(Period <= 0x0100)
        {
            DirFlag = 1;
        }
    }
}

void pwm0_isr(void) interrupt 22
{
    if(PWMCFG01 & 0x08) //判断计数器溢出标志位
    {
        Counter++;
        if(Counter >= 10) //计数 10 个脉冲后关闭 PWM 计数器
        {
            Counter = 0;
            PWMCFG01 = 0x00;
        }
        else
        {
            PWMCFG01 &= ~0x08; //清标志位
        }
    }
}

//=====
// 函数: void PeriodSet(unsigned int period)
// 描述: PWM 周期设置函数。
// 参数: period, 要设置的周期数。
// 返回: none.
// 版本: VER1.0
// 日期: 2021-08-23
// 备注:
//=====
```

```

void PeriodSet(unsigned int period)
{
    PWM0C = period;                                //设置 PWM0 周期为 period 个 PWM 时钟
    PWM00T2= (period>>1);                        //在计数值为 Period/2 地方 PWM00 通道输出高电平
}

=====

// 函数: void delay_ms(unsigned char ms)
// 描述: 延时函数。
// 参数: ms, 要延时的ms 数, 这里只支持1~255ms. 自动适应主时钟。
// 返回: none.
// 版本: VER1.0
// 日期: 2021-01-05
// 备注:
=====

void delay_ms(unsigned char ms)
{
    unsigned int i;
    do{
        i = MAIN_Fosc / 10000;
        while(--i);
    }while(--ms);
}

```

18.2.8 增强型 PWM 时钟输出应用（系统时钟 2 分频输出）

C 语言代码

```

//测试工作频率为 24MHz

#include "stc8g.h"

#define FOSC          24000000UL
#define PWM_PERIOD    (2-1)                                //定义 PWM 周期值
                                                        //((频率=FOSC/(PWM_PERIOD+1)=12MHz)

void SYS_Init();
void PWM_Init();

void main()
{
    SYS_Init();
    PWM_Init();

    while (1);
}

void SYS_Init()
{
    P_SW2 |= 0x80;                                       //扩展寄存器(XFR)访问使能

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

```
P6M1 = 0x00;    P6M0 = 0x00;  
P7M1 = 0x00;    P7M0 = 0x00;  
}  
  
void PWM_Init()  
{  
    PWMSET = 0x01;                      //使能模块  
    PWM0CKS = 0x00;                     //设置时钟源为系统时钟  
    PWM0C = PWM_PERIOD;                 //设置周期值  
    PWM00T1 = 0;                        //设置低电平输出点  
    PWM00T2 = 1;                        //设置高电平输出点  
    PWM00CR = 0x80;                     //使能输出  
    PWMCFG01 = 0x01;                    //开始计时  
}
```

STCMCU

19 同步串行外设接口 SPI

产品线	SPI
STC8G1K08 系列	●
STC8G1K08-8Pin 系列	●
STC8G1K08A 系列	●
STC8G2K64S4 系列	●
STC8G2K64S2 系列	●
STC8G1K08T 系列	●
STC15H2K64S4 系列	●

STC8G 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。STC8G 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

19.1.1 SPI 状态寄存器 (SPSTAT)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPSTAT	CDH	SPIF	WCOL	-	-	-	-	-	-

SPIF: SPI 中断标志位。

当发送/接收完成 1 字节的数据后，硬件自动将此位置 1，并向 CPU 提出中断请求。当 SSIG 位被设置为 0 时，由于 SS 管脚电平的变化而使得设备的主/从模式发生改变时，此标志位也会被硬件自动置 1，以标志设备模式发生变化。

注意：此标志位必须用户通过软件方式向此位写 1 进行清零。

WCOL: SPI 写冲突标志位。

当 SPI 在进行数据传输的过程中写 SPDAT 寄存器时，硬件将此位置 1。

注意：此标志位必须用户通过软件方式向此位写 1 进行清零。

19.1.2 SPI 控制寄存器 (SPCTL), SPI 速度控制

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPCTL	CEH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR[1:0]	

SSIG: SS 引脚功能控制位

0: SS 引脚确定器件是主机还是从机

1: 忽略 SS 引脚功能，使用 MSTR 确定器件是主机还是从机

SPEN: SPI 使能控制位

0: 关闭 SPI 功能
1: 使能 SPI 功能

DORD: SPI 数据位发送/接收的顺序

0: 先发送/接收数据的高位 (MSB)
1: 先发送/接收数据的低位 (LSB)

MSTR: 器件主/从模式选择位

设置主机模式:

若 SSIG=0, 则 SS 管脚必须为高电平且设置 MSTR 为 1
若 SSIG=1, 则只需要设置 MSTR 为 1 (忽略 SS 管脚的电平)

设置从机模式:

若 SSIG=0, 则 SS 管脚必须为低电平 (与 MSTR 位无关)
若 SSIG=1, 则只需要设置 MSTR 为 0 (忽略 SS 管脚的电平)

CPOL: SPI 时钟极性控制

0: SCLK 空闲时为低电平, SCLK 的前时钟沿为上升沿, 后时钟沿为下降沿
1: SCLK 空闲时为高电平, SCLK 的前时钟沿为下降沿, 后时钟沿为上升沿

CPHA: SPI 时钟相位控制

0: 数据 SS 管脚为低电平驱动第一位数据并在 SCLK 的后时钟沿改变数据, 前时钟沿采样数据 (必须 SSIG=0)
1: 数据在 SCLK 的前时钟沿驱动, 后时钟沿采样

SPR[1:0]: SPI 时钟频率选择

SPR[1:0]	SCLK 频率
00	SYScclk/4
01	SYScclk/8
10	SYScclk/16
11	SYScclk/32

19.1.3 SPI 数据寄存器 (SPDAT)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPDAT	CFH								

SPI 发送/接收数据缓冲器。

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:



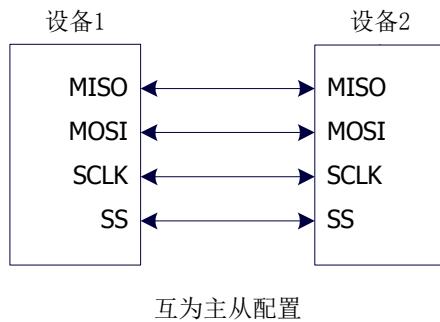
19.2.2 互为主从

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。此时两个设备都是不忽略 SS 的主机模式。当其中一个设备需要启动传输时, 可将自己的 SS 脚设置为输出模式并输出低电平, 拉低对方的 SS 脚, 这样另一个设备就被强行设置为从机模式了。

设置方法 2: 两个设备初始化时都将自己设置成忽略 SS 的从机模式, 即将 SSIG 设置为 1, MSTR 设置为 0。当其中一个设备需要启动传输时, 先检测 SS 管脚的电平, 如果时候高电平, 就将自己设置成忽略 SS 的主模式, 即可进行数据传输了。

互为主从连接配置图如下所示:



互为主从配置

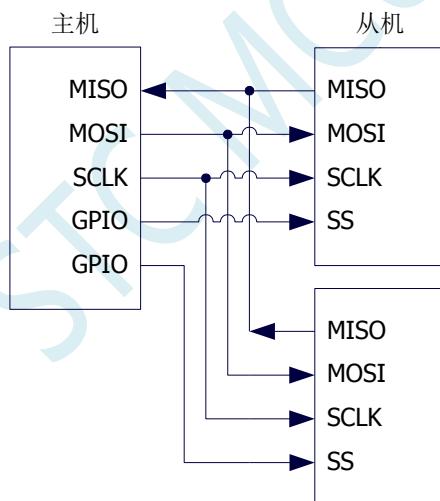
19.2.3 单主多从

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 I/O
1	0	0	0	输出	输入	输入	从机模式, 且被选中
1	0	0	1	高阻	输入	输入	从机模式, 但未被选中
1	0	1→0	0	输出	输入	输入	从机模式, 不忽略 SS 且 MSTR 为 1 的主机模式, 当 SS 管脚被拉低时, MSTR 将被硬件自动清零, 工作模式将被被动设置为从机模式
1	0	1	1	输入	高阻	高阻	主机模式, 空闲状态
					输出	输出	主机模式, 激活状态
1	1	0	x	输出	输入	输入	从机模式
1	1	1	x	输入	输出	输出	主机模式

从机模式的注意事项:

当 CPHA=0 时, SSIG 必须为 0 (即不能忽略 SS 脚)。在每次串行字节开始还发送前 SS 脚必须拉低, 并且在串行字节发送完后须重新设置为高电平。SS 管脚为低电平时不能对 SPDAT 寄存器执行写操作, 否则将导致一个写冲突错误。CPHA=0 且 SSIG=1 时的操作未定义。

当 CPHA=1 时, SSIG 可以置 1 (即可以忽略脚)。如果 SSIG=0, SS 脚可在连续传输之间保持低有效 (即一直固定为低电平)。这种方式适用于固定单主单从的系统。

主机模式的注意事项:

在 SPI 中, 传输总是由主机启动的。如果 SPI 使能 (SPEN=1) 并选择作为主机时, 主机对 SPI 数据寄存器 SPDAT 的写操作将启动 SPI 时钟发生器和数据的传输。在数据写入 SPDAT 之后的半个到一个 SPI 位时间后, 数据将出现在 MOSI 脚。写入主机 SPDAT 寄存器的数据从 MOSI 脚移出发送到从机的 MOSI 脚。同时从机 SPDAT 寄存器的数据从 MISO 脚移出发送到主机的 MISO 脚。

传输完一个字节后, SPI 时钟发生器停止, 传输完成标志 (SPIF) 置位, 如果 SPI 中断使能则会产生一个 SPI 中断。主机和从机 CPU 的两个移位寄存器可以看作是一个 16 位循环移位寄存器。当数据从主机移位传送到从机的同时, 数据也以相反的方向移入。这意味着在一个移位周期中, 主机和从机的数据相互交换。

通过 SS 改变模式

如果 SPEN=1, SSIG=0 且 MSTR=1, SPI 使能为主机模式, 并将 SS 脚可配置为输入模式化或准双向模式。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。

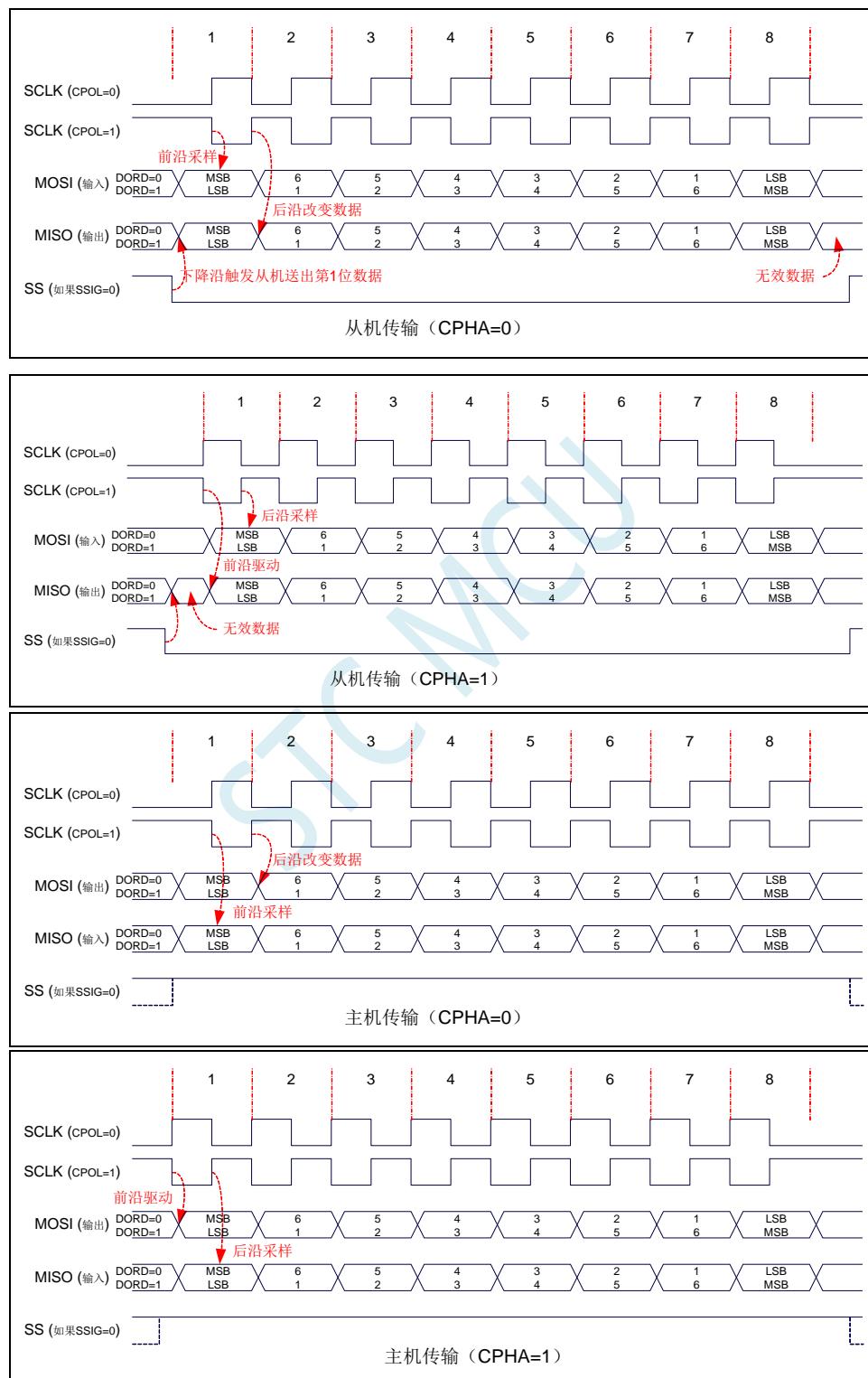
当对主机或从机进行写冲突检测时，主机发生写冲突的情况是很罕见的，因为主机拥有数据传输的完全控制权。但从机有可能发生写冲突，因为当主机启动传输时，从机无法进行控制。

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）

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr SPSTAT      = 0xcd;
sfr SPCTL       = 0xce;
sfr SPDAT       = 0xcf;
sfr IE2          = 0xaf;
#define ESPI        0x02

sfr P0M1         = 0x93;
sfr P0M0         = 0x94;
sfr P1M1         = 0x91;
sfr P1M0         = 0x92;
sfr P2M1         = 0x95;
sfr P2M0         = 0x96;
sfr P3M1         = 0xb1;
sfr P3M0         = 0xb2;
sfr P4M1         = 0xb3;
sfr P4M0         = 0xb4;
sfr P5M1         = 0xc9;
sfr P5M0         = 0xca;

sbit SS           = PI^0;
sbit LED          = PI^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;                                         //测试端口
    LED = !LED;
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
}
```

```

P5M1 = 0x00;

LED = I;
SS = I;
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 = I;

while (1)
{
    while (busy);
    busy = I;
    SS = 0;                //拉低从机SS 管脚
    SPDAT = 0x5a;          //发送测试数据
}

```

汇编代码

; 测试工作频率为 11.0592MHz

SPSTAT	DATA	0CDH
SPCTL	DATA	0CEH
SPDAT	DATA	0CFH
IE2	DATA	0AFH
ESPI	EQU	02H
BUSY	BIT	20H.0
SS	BIT	P1.0
LED	BIT	P1.1
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH
ORG	0000H	
LJMP	MAIN	
ORG	004BH	
LJMP	SPIISR	
ORG	0100H	
 <i>SPIISR:</i>		
MOV	SPSTAT,#0C0H	; 清中断标志
SETB	SS	; 拉高从机的SS 管脚
CLR	BUSY	
CPL	LED	
RETI		

MAIN:

```

MOV      SP, #5FH
MOV      P0M0, #00H
MOV      P0M1, #00H
MOV      P1M0, #00H
MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

SETB    LED
SETB    SS
CLR     BUSY

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

LOOP:
JB      BUSY,$
SETB    BUSY
CLR     SS                  ;拉低从机 SS 管脚
MOV      SPDAT,#5AH          ;发送测试数据
JMP     LOOP

END

```

19.5.2 SPI 单主单从系统从机程序（中断方式）

C 语言代码

//测试工作频率为 11.0592MHz

```

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

sfr    SPSTAT      = 0xcd;
sfr    SPCTL       = 0xce;
sfr    SPDAT       = 0xcf;
sfr    IE2         = 0xaf;
#define ESPI        0x02

sfr    P0M1        = 0x93;
sfr    P0M0        = 0x94;
sfr    P1M1        = 0x91;
sfr    P1M0        = 0x92;
sfr    P2M1        = 0x95;
sfr    P2M0        = 0x96;
sfr    P3M1        = 0xb1;

```

```

sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

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

    SPCTL = 0x40;           //使能SPI 从机模式
    SPSTAT = 0xc0;           //清中断标志
    IE2 = ESPI;              //使能SPI 中断
    EA = 1;

    while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

SPSTAT	DATA	0CDH
SPCTL	DATA	0CEH
SPDAT	DATA	0CFH
IE2	DATA	0AFH
ESPI	EQU	02H
LED	BIT	P1.1
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H

<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>	
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>	
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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>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>RETI</i>		
<i>MAIN:</i>			
	<i>MOV</i>	<i>SP, #5FH</i>	
	<i>MOV</i>	<i>P0M0, #00H</i>	
	<i>MOV</i>	<i>P0M1, #00H</i>	
	<i>MOV</i>	<i>P1M0, #00H</i>	
	<i>MOV</i>	<i>P1M1, #00H</i>	
	<i>MOV</i>	<i>P2M0, #00H</i>	
	<i>MOV</i>	<i>P2M1, #00H</i>	
	<i>MOV</i>	<i>P3M0, #00H</i>	
	<i>MOV</i>	<i>P3M1, #00H</i>	
	<i>MOV</i>	<i>P4M0, #00H</i>	
	<i>MOV</i>	<i>P4M1, #00H</i>	
	<i>MOV</i>	<i>P5M0, #00H</i>	
	<i>MOV</i>	<i>P5M1, #00H</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>MOV</i>	<i>IE2,#ESPI</i>	;使能 SPI 中断
	<i>SETB</i>	<i>EA</i>	
	<i>JMP</i>	\$	
	<i>END</i>		

19.5.3 SPI 单主单从系统主机程序（查询方式）

C 语言代码

//测试工作频率为11.0592MHz

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

sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
```

```

sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

sfr SPSTAT    = 0xcd;
sfr SPCTL     = 0xce;
sfr SPDAT     = 0xcf;
sfr IE2        = 0xaf;
#define ESPI      0x02

sbit SS        = P1^0;
sbit LED       = P1^1;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    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
    }
}

```

汇编代码

; 测试工作频率为 11.0592MHz

SPSTAT	DATA	0CDH
SPCTL	DATA	0CEH
SPDAT	DATA	0CFH
IE2	DATA	0AFH
ESPI	EQU	02H
SS	BIT	P1.0
LED	BIT	P1.1

```

P0M1      DATA      093H
P0M0      DATA      094H
P1M1      DATA      091H
P1M0      DATA      092H
P2M1      DATA      095H
P2M0      DATA      096H
P3M1      DATA      0B1H
P3M0      DATA      0B2H
P4M1      DATA      0B3H
P4M0      DATA      0B4H
P5M1      DATA      0C9H
P5M0      DATA      0CAH

        ORG      0000H
        LJMP     MAIN

        ORG      0100H
MAIN:
        MOV      SP, #5FH
        MOV      P0M0, #00H
        MOV      P0M1, #00H
        MOV      P1M0, #00H
        MOV      P1M1, #00H
        MOV      P2M0, #00H
        MOV      P2M1, #00H
        MOV      P3M0, #00H
        MOV      P3M1, #00H
        MOV      P4M0, #00H
        MOV      P4M1, #00H
        MOV      P5M0, #00H
        MOV      P5M1, #00H

        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

```

19.5.4 SPI 单主单从系统从机程序（查询方式）

C 语言代码

//测试工作频率为11.0592MHz

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

sfr SPSTAT      = 0xcd;
sfr SPCTL       = 0xce;
sfr SPDAT       = 0xcf;
sfr IE2          = 0xaf;
#define ESPI        0x02

sfr P0M1         = 0x93;
sfr P0M0         = 0x94;
sfr P1M1         = 0x91;
sfr P1M0         = 0x92;
sfr P2M1         = 0x95;
sfr P2M0         = 0x96;
sfr P3M1         = 0xb1;
sfr P3M0         = 0xb2;
sfr P4M1         = 0xb3;
sfr P4M0         = 0xb4;
sfr P5M1         = 0xc9;
sfr P5M0         = 0xca;

sbit LED          = PI^1;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                                //清中断标志
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    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
    }
}
```

汇编代码

; 测试工作频率为 11.0592MHz

<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>P0M1</i>	<i>DATA</i>	<i>093H</i>	
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>	
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>	
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>	
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>	
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>	
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>	
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>	
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>	
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>	
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>	
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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, #5FH</i>	
	<i>MOV</i>	<i>P0M0, #00H</i>	
	<i>MOV</i>	<i>P0M1, #00H</i>	
	<i>MOV</i>	<i>P1M0, #00H</i>	
	<i>MOV</i>	<i>P1M1, #00H</i>	
	<i>MOV</i>	<i>P2M0, #00H</i>	
	<i>MOV</i>	<i>P2M1, #00H</i>	
	<i>MOV</i>	<i>P3M0, #00H</i>	
	<i>MOV</i>	<i>P3M1, #00H</i>	
	<i>MOV</i>	<i>P4M0, #00H</i>	
	<i>MOV</i>	<i>P4M1, #00H</i>	
	<i>MOV</i>	<i>P5M0, #00H</i>	
	<i>MOV</i>	<i>P5M1, #00H</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>		

19.5.5 SPI 互为主从系统程序（中断方式）

C 语言代码

//测试工作频率为 11.0592MHz

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

sfr SPSTAT      = 0xcd;
sfr SPCTL       = 0xce;
sfr SPDAT       = 0xcf;
sfr IE2          = 0xaf;
#define ESPI        0x02

sfr P0M1         = 0x93;
sfr P0M0         = 0x94;
sfr P1M1         = 0x91;
sfr P1M0         = 0x92;
sfr P2M1         = 0x95;
sfr P2M0         = 0x96;
sfr P3M1         = 0xb1;
sfr P3M0         = 0xb2;
sfr P4M1         = 0xb3;
sfr P4M0         = 0xb4;
sfr P5M1         = 0xc9;
sfr P5M0         = 0xca;

sbit SS           = P1^0;
sbit LED          = P1^1;
sbit KEY          = P0^0;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                                //清中断标志
    if (SPCTL & 0x10)
    {
        SS = 1;                                    //主机模式
        SPCTL = 0x40;                             //拉高从机的SS管脚
                                                //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()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
}
```

```

P4M1 = 0x00;
P5M0 = 0x00;
P5M1 = 0x00;

LED = I;
KEY = I;
SS = I;

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 = I;

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
    }
}
}

```

汇编代码

; 测试工作频率为 11.0592MHz

<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>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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>		

```

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, #5FH
MOV      P0M0, #00H
MOV      P0M1, #00H
MOV      P1M0, #00H
MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

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

```

19.5.6 SPI 互为主从系统程序（查询方式）

C 语言代码

//测试工作频率为 11.0592MHz

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

```
#include "intrins.h"

sfr SPSTAT      = 0xcd;
sfr SPCTL       = 0xce;
sfr SPDAT       = 0xcf;
sfr IE2          = 0xaf;
#define ESPI        0x02

sfr P0M1         = 0x93;
sfr P0M0         = 0x94;
sfr P1M1         = 0x91;
sfr P1M0         = 0x92;
sfr P2M1         = 0x95;
sfr P2M0         = 0x96;
sfr P3M1         = 0xb1;
sfr P3M0         = 0xb2;
sfr P4M1         = 0xb3;
sfr P4M0         = 0xb4;
sfr P5M1         = 0xc9;
sfr P5M0         = 0xca;

sbit SS          = P1^0;
sbit LED         = P1^1;
sbit KEY         = P0^0;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    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;                            //拉高从机的SS 管脚
}
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
}
}
}

```

汇编代码

; 测试工作频率为 11.0592MHz

<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>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</i>
<i>ORG</i>		<i>0000H</i>
<i>LJMP</i>		<i>MAIN</i>
<i>ORG</i>		<i>0100H</i>
MAIN:		
<i>MOV</i>		<i>SP, #5FH</i>
<i>MOV</i>		<i>P0M0, #00H</i>
<i>MOV</i>		<i>P0M1, #00H</i>
<i>MOV</i>		<i>P1M0, #00H</i>
<i>MOV</i>		<i>P1M1, #00H</i>
<i>MOV</i>		<i>P2M0, #00H</i>
<i>MOV</i>		<i>P2M1, #00H</i>
<i>MOV</i>		<i>P3M0, #00H</i>
<i>MOV</i>		<i>P3M1, #00H</i>
<i>MOV</i>		<i>P4M0, #00H</i>
<i>MOV</i>		<i>P4M1, #00H</i>
<i>MOV</i>		<i>P5M0, #00H</i>
<i>MOV</i>		<i>P5M1, #00H</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>		

20 I²C 总线

产品线	I ² C
STC8G1K08 系列	●
STC8G1K08-8Pin 系列	●
STC8G1K08A 系列	●
STC8G2K64S4 系列	●
STC8G2K64S2 系列	●
STC8G1K08T 系列	●
STC15H2K64S4 系列	●

STC8G 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，STC8G 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

STC8G 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)

STC 创新：STC 的 I²C 串行总线控制器工作在从机模式时，SDA 管脚的下降沿信号可以唤醒进入掉电模式的 MCU。(注意：由于 I²C 传输速度比较快，MCU 唤醒后第一包数据一般是不正确的)

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[5:0]								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	0xx,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	I2CSLADR[7:1]								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

20.2.1 I²C 配置寄存器 (I2CCFG), 总线速度控制

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CCFG	FE80H	ENI2C	MSSL						MSSPEED[5:0]

ENI2C: I²C 功能使能控制位

0: 禁止 I²C 功能

1: 允许 I²C 功能

MSSL: I²C 工作模式选择位

0: 从机模式

1: 主机模式

MSSPEED[5:0]: I²C 总线速度 (等待时钟数) 控制, **I²C 总线速度=Fosc / 2 / (MSSPEED * 2 + 4)**

MSSPEED[5:0]	对应的时钟数
0	4
1	6
2	8
...	...
X	2x+4
...	...
62	128
63	130

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)

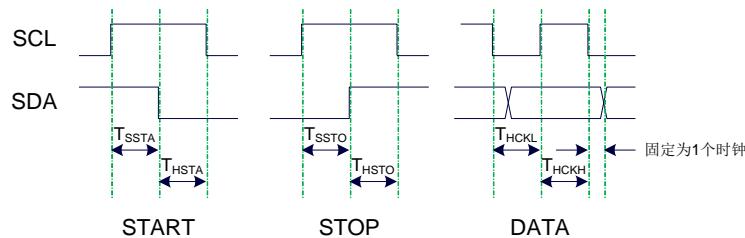
T_{HSTA}: 起始信号的保持时间 (Hold Time of START)

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)

T_{HCKH}: 时钟信号的高电平保持时间 (Hold Time of SCL High)



例 1: 当 MSSPEED=10 时, T_{SSTA}=T_{HSTA}=T_{SSTO}=T_{HSTO}=T_{HCKL}=24/FOSC

例 2: 当 24MHz 的工作频率下需要 400K 的 I²C 总线速度时,

$$\text{MSSPEED} = (24\text{M} / 400\text{K} / 2 - 4) / 2 = 13$$

20.2.2 I²C 主机控制寄存器 (I2CMSCR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSCR	FE81H	EMSI	-	-	-				MSCMD[3:0]

EMSI: 主机模式中断使能控制位

0: 关闭主机模式的中断

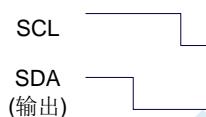
1: 允许主机模式的中断

MSCMD[3:0]: 主机命令

0000: 待机, 无动作。

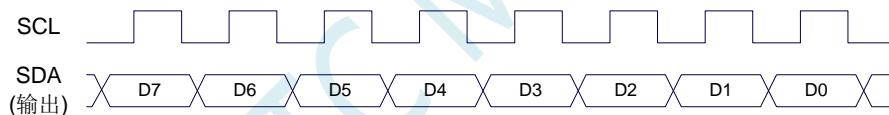
0001: 起始命令。

发送 START 信号。如果当前 I²C 控制器处于空闲状态, 即 MSBUSY (I2CMSST.7) 为 0 时, 写此命令会使控制器进入忙状态, 硬件自动将 MSBUSY 状态位置 1, 并开始发送 START 信号; 若当前 I²C 控制器处于忙状态, **写此命令可触发发送 START 信号**。发送 START 信号的波形如下图所示:



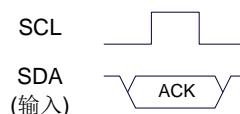
0010: 发送数据命令。

写此命令后, I²C 总线控制器会在 SCL 管脚上产生 8 个时钟, 并将 I2CTXD 寄存器里面数据按位送到 SDA 管脚上 (先发送高位数据)。发送数据的波形如下图所示:



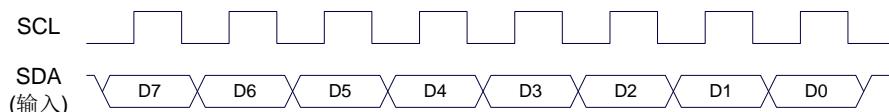
0011: 接收 ACK 命令。

写此命令后, I²C 总线控制器会在 SCL 管脚上产生 1 个时钟, 并将从 SDA 端口上读取的数据保存到 MSACKI (I2CMSST.1)。接收 ACK 的波形如下图所示:



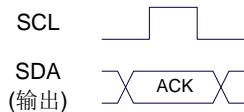
0100: 接收数据命令。

写此命令后, I²C 总线控制器会在 SCL 管脚上产生 8 个时钟, 并将从 SDA 端口上读取的数据依次左移到 I2CRXD 寄存器 (先接收高位数据)。接收数据的波形如下图所示:



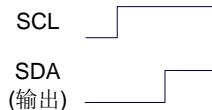
0101: 发送 ACK 命令。

写此命令后, I²C 总线控制器会在 SCL 管脚上产生 1 个时钟, 并将 MSACKO (I2CMSST.0) 中的数据发送到 SDA 端口。发送 ACK 的波形如下图所示:



0110: 停止命令。

发送 STOP 信号。写此命令后, 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。

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。

20.2.3 I²C 主机辅助控制寄存器 (I2CMSAUX)

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。

20.2.4 I²C 主机状态寄存器 (I2CMSST)

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。

20.3 I²C 从机模式

20.3.1 I²C 从机控制寄存器 (I2CSLCR)

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

20.3.2 I²C 从机状态寄存器 (I2CSLST)

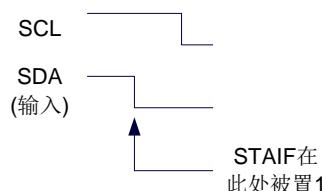
符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLST	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	-	SLACKI	SLACKO

SLBUSY: 从机模式时 I²C 控制器状态位 (只读位)

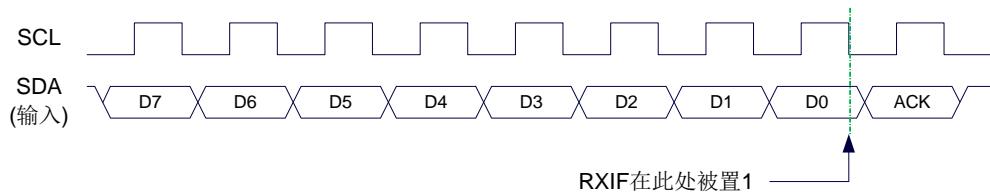
- 0: 控制器处于空闲状态
- 1: 控制器处于忙碌状态

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相同时，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。

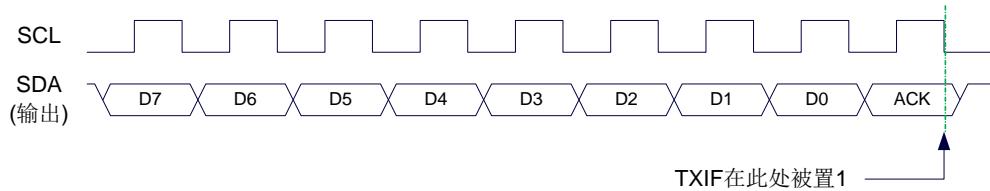
STAIF: 从机模式时接收到 START 信号后的中断请求位。从机模式的 I²C 控制器接收到 START 信号后，硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 STAIF 位必须用软件清零。STAIF 被置 1 的时间点如下图所示：



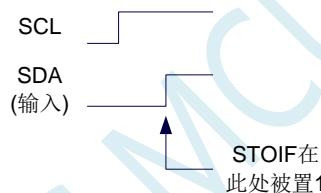
RXIF: 从机模式时接收到 1 字节的数据后的中断请求位。从机模式的 I²C 控制器接收到 1 字节的数据后，在第 8 个时钟的下降沿时硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 RXIF 位必须用软件清零。RXIF 被置 1 的时间点如下图所示：



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：

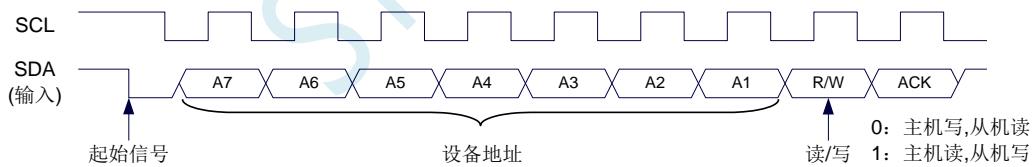


STOIF: 从机模式时接收到 STOP 信号后的中断请求位。从机模式的 I²C 控制器接收到 STOP 信号后，硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 STOIF 位必须用软件清零。STOIF 被置 1 的时间点如下图所示：



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。



20.3.3 I²C 从机地址寄存器 (I2CSLADR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLADR	FE85H								I2CSLADR[7:1] MA

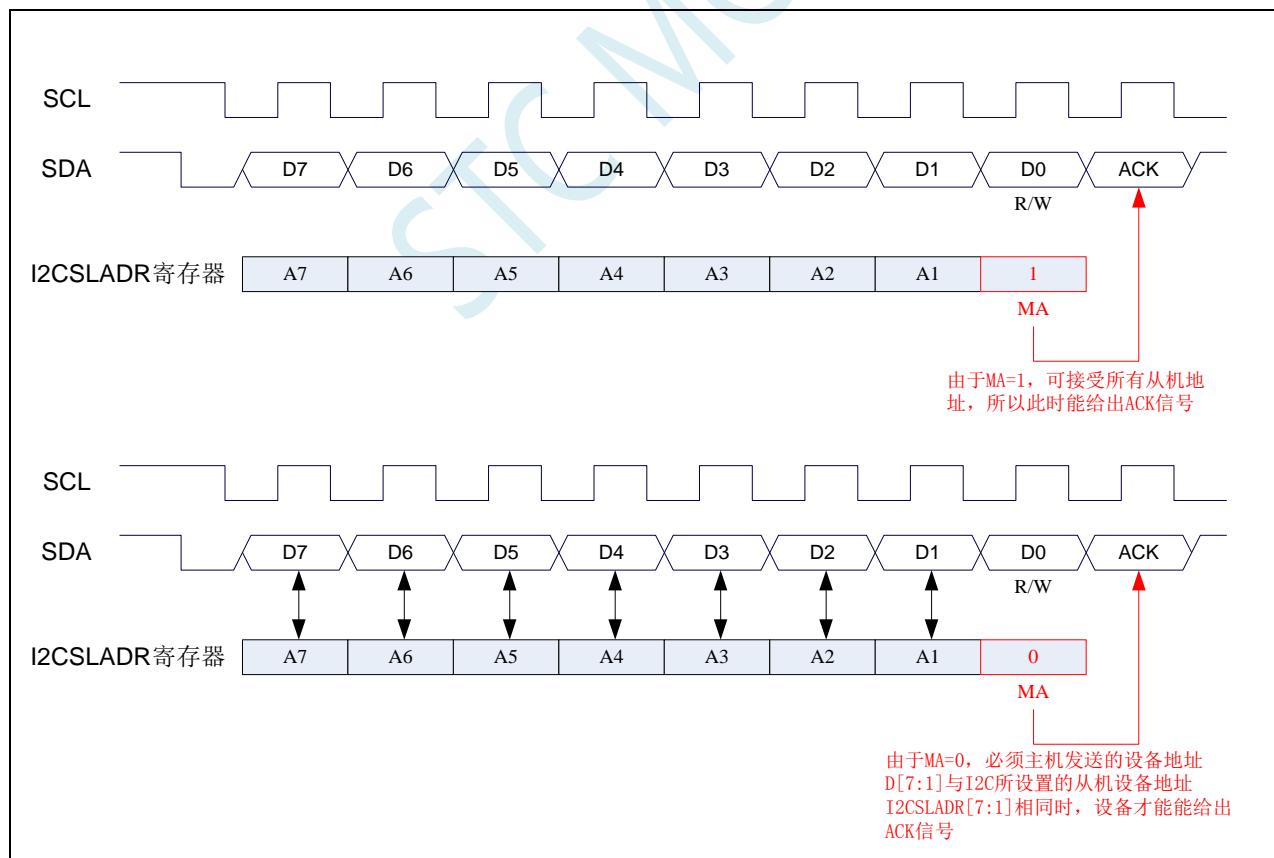
I2CSLADR[7:1]: 从机设备地址

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 I2CSLADR[7:1]中所设置的从机设备地址相同时, 控制器才会向 CPU 发出中断请求, 请求 CPU 处理 I²C 事件; 否则若设备地址不同, I²C 控制器继续监控, 等待下一个起始信号, 对下一个设备地址继续比较。

MA: 从机设备地址比较控制

- 0: 设备地址必须与 I2CSLADR[7:1]相同
- 1: 忽略 I2CSLADR[7:1]中的设置, 接受所有的设备地址

说明: I²C 总线协议规定 I²C 总线上最多可挂载 128 个 I²C 设备 (理论值), 不同的 I²C 设备用不同的 I²C 从机设备地址进行识别。I²C 主机发送完成起始信号后, 发送的第一个数据 (DATA0) 的高 7 位即为从机设备地址 (DATA0[7:1]为 I²C 设备地址), 最低位为读写信号。当 I²C 设备从机地址寄存器 MA (I2CSLADR.0) 为 1 时, 表示 I²C 从机能够接受所有的设备地址, 此时主机发送的任何设备地址, 即 DATA0[7:1]为任何值, 从机都能响应。当 I²C 设备从机地址寄存器 MA (I2CSLADR.0) 为 0 时, 主机发送的设备地址 DATA0[7:1]必须与从机的设备地址 I2CSLADR[7:1]相同时才能访问此从机设备



20.3.4 I²C 数据寄存器 (I2CTXD, I2CRXD)

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)

C 语言代码

```
//测试工作频率为 11.0592MHz

#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)

sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P4M1 = 0xb3;
sfr P4M0 = 0xb4;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;

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;                                     //发送 SEND 命令
    I2CMSCR = 0x82;
    while (busy);
}

void RecvACK()
{
    busy = 1;                                     //发送读 ACK 命令
    I2CMSCR = 0x83;
    while (busy);
}

char RecvData()
{
    busy = 1;                                     //发送 RECV 命令
    I2CMSCR = 0x84;
    while (busy);
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;                               //设置 ACK 信号
    busy = 1;                                     //发送 ACK 命令
    I2CMSCR = 0x85;
    while (busy);
}

void SendNAK()
{
    I2CMSST = 0x01;                               //设置 NAK 信号
    busy = 1;                                     //发送 ACK 命令
    I2CMSCR = 0x85;
    while (busy);
}

void Stop()
{
    busy = 1;                                     //发送 STOP 命令
    I2CMSCR = 0x86;
    while (busy);
}

void Delay()
{
    int i;

    for (i=0; i<3000; i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
}
```

```

        }

}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    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);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

<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>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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>
------------	--------------

I2CISR:

<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>	; 清中断标志
------------	----------------------	---------

<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>	; 复位忙标志
------------	-------------	---------

<i>POP</i>	<i>DPH</i>
------------	------------

<i>POP</i>	<i>DPL</i>
------------	------------

<i>POP</i>	<i>ACC</i>
------------	------------

<i>RETI</i>

START:

<i>SETB</i>	<i>BUSY</i>
-------------	-------------

MOV	A,#10000001B	;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	
SETB	BUSY	
MOV	A,#10000010B	;发送 SEND 命令
MOV	DPTR,#I2CMSCR	
MOVX	@DPTR,A	
JMP	WAIT	
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	RI,#0	
DELAY1:		

```

NOP
NOP
NOP
NOP
DJNZ      RI,DELAY1
DJNZ      R0,DELAY1
RET

```

MAIN:

```

MOV      SP, #5FH
MOV      P0M0, #00H
MOV      P0M1, #00H
MOV      P1M0, #00H
MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

MOV      P_SW2,#80H

MOV      A,#III100000B          ;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,#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,#I2H                  ;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,#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,#0A1H          ;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

20.4.2 I²C 主机模式访问 AT24C256 (查询方式)

C 语言代码

//测试工作频率为 11.0592MHz

```

#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)

sfr      P0M1      = 0x93;
sfr      P0M0      = 0x94;
sfr      P1M1      = 0x91;
sfr      P1M0      = 0x92;
sfr      P2M1      = 0x95;
sfr      P2M0      = 0x96;
sfr      P3M1      = 0xb1;
sfr      P3M0      = 0xb2;
sfr      P4M1      = 0xb3;
sfr      P4M0      = 0xb4;
sfr      P5M1      = 0xc9;
sfr      P5M0      = 0xca;

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 RcvACK()
{
    I2CMSCR = 0x03;                                //发送读 ACK 命令
    Wait();
}

char RcvData()
{
    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()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    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);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

<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>P0M1</i>	<i>DATA</i>	<i>093H</i>	
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>	
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>	
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>	
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>	
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>	
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>	
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>	
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>	
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>	
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>	
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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>SENDATA:</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>	
<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>		
SENDACK:		
<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>	
SENDNAK:		
<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>	
<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>	
STOP:		
<i>MOV</i>	<i>A,#00000110B</i>	;发送STOP命令
<i>MOV</i>	<i>DPTR,#I2CMSCR</i>	
<i>MOVX</i>	<i>@DPTR,A</i>	
<i>JMP</i>	<i>WAIT</i>	
WAIT:		
<i>MOV</i>	<i>DPTR,#I2CMSST</i>	;清中断标志
<i>MOVX</i>	<i>A,@DPTR</i>	
<i>JNB</i>	<i>ACC.6,WAIT</i>	
<i>ANL</i>	<i>A,#NOT 40H</i>	
<i>MOVX</i>	<i>@DPTR,A</i>	
<i>RET</i>		
DELAY:		
<i>MOV</i>	<i>R0,#0</i>	
<i>MOV</i>	<i>R1,#0</i>	
DELAY1:		
<i>NOP</i>		
<i>DJNZ</i>	<i>R1,DELAY1</i>	
<i>DJNZ</i>	<i>R0,DELAY1</i>	
<i>RET</i>		
MAIN:		
<i>MOV</i>	<i>SP, #5FH</i>	
<i>MOV</i>	<i>P0M0, #00H</i>	
<i>MOV</i>	<i>P0M1, #00H</i>	
<i>MOV</i>	<i>P1M0, #00H</i>	
<i>MOV</i>	<i>P1M1, #00H</i>	
<i>MOV</i>	<i>P2M0, #00H</i>	
<i>MOV</i>	<i>P2M1, #00H</i>	
<i>MOV</i>	<i>P3M0, #00H</i>	
<i>MOV</i>	<i>P3M1, #00H</i>	
<i>MOV</i>	<i>P4M0, #00H</i>	
<i>MOV</i>	<i>P4M1, #00H</i>	
<i>MOV</i>	<i>P5M0, #00H</i>	
<i>MOV</i>	<i>P5M1, #00H</i>	

```

MOV      P_SW2,#80H

MOV      A,#III00000B          ; 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    SENDATA             ; 发送设备地址+写命令
CALL    RECVACK
MOV      A,#000H              ; 发送存储地址高字节
CALL    SENDATA
CALL    RECVACK
MOV      A,#000H              ; 发送存储地址低字节
CALL    SENDATA
CALL    RECVACK
MOV      A,#I2H               ; 写测试数据 1
CALL    SENDATA
CALL    RECVACK
MOV      A,#78H               ; 写测试数据 2
CALL    SENDATA
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    SENDATA
CALL    RECVACK
MOV      A,#000H              ; 发送存储地址高字节
CALL    SENDATA
CALL    RECVACK
MOV      A,#000H              ; 发送存储地址低字节
CALL    SENDATA
CALL    RECVACK
CALL    START                ; 发送起始命令
MOV      A,#0A1H              ; 发送设备地址+读命令
CALL    SENDATA
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

```

20.4.3 I²C 主机模式访问 PCF8563

C 语言代码

//测试工作频率为 11.0592MHz

```
#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)

sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P4M1 = 0xb3;
sfr P4M0 = 0xb4;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;

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()  
{  
    P0M0 = 0x00;  
    P0M1 = 0x00;  
    P1M0 = 0x00;  
    P1M1 = 0x00;  
    P2M0 = 0x00;  
    P2M1 = 0x00;  
    P3M0 = 0x00;  
    P3M1 = 0x00;  
    P4M0 = 0x00;  
    P4M1 = 0x00;  
    P5M0 = 0x00;  
    P5M1 = 0x00;  
  
    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();
}

```

汇编代码

; 测试工作频率为 11.0592MHz

<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>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>

<i>P1M1</i>	<i>DATA</i>	<i>091H</i>	
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>	
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>	
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>	
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>	
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>	
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>	
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>	
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>	
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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>	
<i>RECVDATA:</i>			
	<i>MOV</i>	<i>A,#000000100B</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,#000000101B</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,#000000001B</i>	;设置 NAK 信号
	<i>MOV</i>	<i>DPTR,#I2CMSST</i>	
	<i>MOVX</i>	<i>@DPTR,A</i>	
	<i>MOV</i>	<i>A,#000000101B</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>STOP:</i>			
	<i>MOV</i>	<i>A,#000000110B</i>	;发送 STOP 命令
	<i>MOV</i>	<i>DPTR,#I2CMSCR</i>	

```

MOVX      @DPTR,A
JMP
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, #5FH
MOV        P0M0, #00H
MOV        P0M1, #00H
MOV        P1M0, #00H
MOV        P1M1, #00H
MOV        P2M0, #00H
MOV        P2M1, #00H
MOV        P3M0, #00H
MOV        P3M1, #00H
MOV        P4M0, #00H
MOV        P4M1, #00H
MOV        P5M0, #00H
MOV        P5M1, #00H

MOV        P_SW2,#80H

MOV        A,#III00000B           ;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

```

<i>MOV</i>	<i>A,#I2H</i>	; 设置小时值
<i>CALL</i>	<i>SENDATA</i>	
<i>CALL</i>	<i>RECVACK</i>	
<i>CALL</i>	<i>STOP</i>	; 发送停止命令
LOOP:		
<i>CALL</i>	<i>START</i>	; 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>SENDATA</i>	
<i>CALL</i>	<i>RECVACK</i>	
<i>MOV</i>	<i>A,#002H</i>	; 发送存储地址
<i>CALL</i>	<i>SENDATA</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>SENDATA</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>	
END		

20.4.4 I²C 从机模式（中断方式）

C 语言代码

// 测试工作频率为 11.0592MHz

```
#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)

sfr PIM1 = 0x91;
sfr PIM0 = 0x92;
sfr P0M1 = 0x93;
```

```

sfr P0M0      = 0x94;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

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 xdata buffer[256];

void I2C_Isr() interrupt 24
{
    _push_(P_SW2);
    P_SW2 |= 0x80;

    if (I2CSLST & 0x40)
    {
        I2CSLST &= ~0x40;           //处理 START 事件
        isda = 1;                  //若为重复起始信号时必须作此设置
    }
    else if (I2CSLST & 0x20)
    {
        I2CSLST &= ~0x20;           //处理 RECV 事件
        if (isda)                  //处理 RECV 事件 (RECV DEVICE ADDR)
        {
            isda = 0;
        }
        else if (isma)             //处理 RECV 事件 (RECV MEMORY ADDR)
        {
            isma = 0;
            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()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    I2CCFG = 0x81;                                //使能 I2C 从机模式
    I2CSLADR = 0x5a;                                //设置从机设备地址寄存器 I2CSLADR=0101_1010B
                                                    //即 I2CSLADR[7:1]=010_1101B, MA=0B。
                                                    //由于 MA 为 0, 主机发送的设备地址必须与
                                                    //I2CSLADR[7:1] 相同才能访问此 I2C 从机设备。
                                                    //主机若需要写数据则要发送 5AH(0101_1010B)
                                                    //主机若需要读数据则要发送 5BH(0101_1011B)

    I2CSLST = 0x00;
    I2CSLCR = 0x78;                                //使能从机模式中断
    EA = 1;

    isda = 1;                                       //用户变量初始化
    isma = 1;
    addr = 0;
    I2CTXD = buffer[addr];

    while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

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

<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>PIM1</i>	<i>DATA</i>	<i>091H</i>	
<i>PIM0</i>	<i>DATA</i>	<i>092H</i>	
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>	
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>	
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>	
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>	
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>	
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>	
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>	
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>	
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>	
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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>SETB</i>	<i>ISDA</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>	

	JMP	ISREXIT	
RXDA:	JMP	ISREXIT	; 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	ISREXIT	
TXIF:	ANL	A,#NOT 10H	; 处理 SEND 事件
	MOVX	@DPTR,A	
	JB	ACC.I,RXNAK	
	INC	ADDR	
	MOV	R0,ADDR	
	MOVX	A,@R0	
	MOV	DPTR,#I2CTXD	
	MOVX	@DPTR,A	
	JMP	ISREXIT	
RXNAK:	MOVX	A,#0FFH	
	MOV	DPTR,#I2CTXD	
	MOVX	@DPTR,A	
	JMP	ISREXIT	
STOPIF:	ANL	A,#NOT 08H	; 处理 STOP 事件
	MOVX	@DPTR,A	
	SETB	ISDA	
	SETB	ISMA	
	JMP	ISREXIT	
MAIN:	MOV	SP, #5FH	
	MOV	P0M0, #00H	
	MOV	P0M1, #00H	
	MOV	P1M0, #00H	
	MOV	P1M1, #00H	
	MOV	P2M0, #00H	
	MOV	P2M1, #00H	
	MOV	P3M0, #00H	
	MOV	P3M1, #00H	
	MOV	P4M0, #00H	
	MOV	P4M1, #00H	
	MOV	P5M0, #00H	
	MOV	P5M1, #00H	
	MOV	P_SW2,#80H	
	MOV	A,#10000001B	; 使能 I2C 从机模式
	MOV	DPTR,#I2CCFG	
	MOVX	@DPTR,A	
	MOV	A,#01011010B	; 设置从机设备地址寄存器 I2CSLADR=0101_1010B ; 即 I2CSLADR[7:1]=010_1101B, MA=0B。 ; 由于 MA 为 0, 主机发送的设备地址必须与 ; I2CSLADR[7:1] 相同才能访问此 I2C 从机设备。 ; 主机若需要写数据则要发送 5AH(0101_1010B) ; 主机若需要读数据则要发送 5BH(0101_1011B)
	MOV	DPTR,#I2CSLADR	

```

MOVX      @DPTR,A
MOV        A,#00000000B
MOV        DPTR,#I2CSLST
MOVX      @DPTR,A
MOV        A,#0111000B          ;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

20.4.5 I²C 从机模式（查询方式）

C 语言代码

//测试工作频率为11.0592MHz

```

#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)

sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P4M1 = 0xb3;
sfr P4M0 = 0xb4;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;

```

```

sbit SDA      = P1^4;
sbit SCL      = P1^5;

bit  isda;
bit  isma;
unsigned char    addr;
unsigned char xdata  buffer[256];

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    I2CCFG = 0x81;                                //使能I2C 从机模式
    I2CSLADR = 0x5a;                               //设置从机设备地址寄存器I2CSLADR=0101_1010B
                                                    //即I2CSLADR[7:1]=010_1101B,MA=0B。
                                                    //由于MA 为0,主机发送的的设备地址必须与
                                                    //I2CSLADR[7:1]相同才能访问此I2C 从机设备。
                                                    //主机若需要写数据则要发送5AH(0101_1010B)
                                                    //主机若需要读数据则要发送5BH(0101_1011B)

    I2CSLST = 0x00;
    I2CSLCR = 0x00;                               //禁止从机模式中断

    isda = 1;                                     //用户变量初始化
    isma = 1;
    addr = 0;
    I2CTXD = buffer[addr];

    while (1)
    {
        if (I2CSLST & 0x40)                      //处理START 事件
        {
            I2CSLST &= ~0x40;                     //若为重复起始信号时必须作此设置
            isda = 1;
        }
        else if (I2CSLST & 0x20)                  //处理RECV 事件
        {
            I2CSLST &= ~0x20;
            if (isda)                            //处理RECV 事件 (RECV DEVICE ADDR)
            {
                isda = 0;
            }
            else if (isma)                      //处理RECV 事件 (RECV MEMORY ADDR)
            {
                isma = 0;
                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;
    }
}
}

```

汇编代码

;测试工作频率为11.0592MHz

<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>
<i>ADDR</i>	<i>DATA</i>	<i>21H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>

<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>	
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</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, #5FH</i>	
	<i>MOV</i>	<i>P0M0, #00H</i>	
	<i>MOV</i>	<i>P0M1, #00H</i>	
	<i>MOV</i>	<i>P1M0, #00H</i>	
	<i>MOV</i>	<i>P1M1, #00H</i>	
	<i>MOV</i>	<i>P2M0, #00H</i>	
	<i>MOV</i>	<i>P2M1, #00H</i>	
	<i>MOV</i>	<i>P3M0, #00H</i>	
	<i>MOV</i>	<i>P3M1, #00H</i>	
	<i>MOV</i>	<i>P4M0, #00H</i>	
	<i>MOV</i>	<i>P4M1, #00H</i>	
	<i>MOV</i>	<i>P5M0, #00H</i>	
	<i>MOV</i>	<i>P5M1, #00H</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>	;设置从机设备地址寄存器 I2CSLADR=0101_1010B ;即 I2CSLADR[7:1]=010_1101B,MA=0B。 ;由于 MA 为 0, 主机发送的设备地址必须与 ;I2CSLADR[7:1]相同才能访问此 I2C 从机设备。 ;主机若需要写数据则要发送 5AH(0101_1010B) ;主机若需要读数据则要发送 5BH(0101_1011B)
	<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>	
	<i>MOVX</i>	<i>A,@R0</i>	
	<i>MOV</i>	<i>DPTR,#I2CTXD</i>	
	<i>MOVX</i>	<i>@DPTR,A</i>	
<i>LOOP:</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>JMP</i>	<i>LOOP</i>	

STARTIF:

<i>ANL</i>	<i>A,#NOT 40H</i>	; <i>处理 START 事件</i>
<i>MOVX</i>	<i>@DPTR,A</i>	
<i>SETB</i>	<i>ISDA</i>	
<i>JMP</i>	<i>LOOP</i>	

RXIF:

<i>ANL</i>	<i>A,#NOT 20H</i>	; <i>处理 RECV 事件</i>
<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>	; <i>处理 RECV 事件 (RECV DATA)</i>
<i>MOVX</i>	<i>@R0,A</i>	
<i>INC</i>	<i>ADDR</i>	
<i>JMP</i>	<i>LOOP</i>	

RXDA:

<i>JMP</i>	<i>LOOP</i>	; <i>处理 RECV 事件 (RECV DEVICE ADDR)</i>
------------	-------------	----------------------------------------

RXMA:

<i>MOV</i>	<i>ADDR,A</i>	; <i>处理 RECV 事件 (RECV MEMORY ADDR)</i>
<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>LOOP</i>	

TXIF:

<i>ANL</i>	<i>A,#NOT 10H</i>	; <i>处理 SEND 事件</i>
<i>MOVX</i>	<i>@DPTR,A</i>	
<i>JB</i>	<i>ACC,I,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>LOOP</i>	

RXNAK:

<i>MOVX</i>	<i>A,#0FFH</i>	
<i>MOV</i>	<i>DPTR,#I2CTXD</i>	
<i>MOVX</i>	<i>@DPTR,A</i>	
<i>JMP</i>	<i>LOOP</i>	

STOPIF:

<i>ANL</i>	<i>A,#NOT 08H</i>	; <i>处理 STOP 事件</i>
<i>MOVX</i>	<i>@DPTR,A</i>	
<i>SETB</i>	<i>ISDA</i>	
<i>SETB</i>	<i>ISMA</i>	
<i>JMP</i>	<i>LOOP</i>	

END

20.4.6 测试 I²C 从机模式代码的主机代码

C 语言代码

//测试工作频率为 11.0592MHz

#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)

sfr PIMI      = 0x91;
sfr P1M0      = 0x92;
sfr P0M1      = 0x93;
sfr P0M0      = 0x94;
sfr P2M1      = 0x95;
sfr P2M0      = 0x96;
sfr P3M1      = 0xb1;
sfr P3M0      = 0xb2;
sfr P4M1      = 0xb3;
sfr P4M0      = 0xb4;
sfr P5M1      = 0xc9;
sfr P5M0      = 0xca;

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

    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(010_1101B)+写命令(0B)
    RecvACK();                                       //接收ACK 命令
    SendData(0x00);                                //发送存储地址
}
```

```

RecvACK();
SendData(0x12);                                //写测试数据1
RecvACK();
SendData(0x78);                                //写测试数据2
RecvACK();
Stop();                                         //发送停止命令

Start();                                         //发送起始命令
SendData(0x5a);                                //发送设备地址(010_1101B)+写命令(0B)
RecvACK();
SendData(0x00);                                //发送存储地址高字节
RecvACK();
Start();                                         //发送起始命令
SendData(0x5b);                                //发送设备地址(010_1101B)+读命令(1B)
RecvACK();
P0 = RecvData();                               //读取数据1
SendACK();
P2 = RecvData();                               //读取数据2
SendNAK();
Stop();                                         //发送停止命令

P_SW2 = 0x00;

while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

<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>PIMI</i>	<i>DATA</i>	<i>091H</i>
<i>PIM0</i>	<i>DATA</i>	<i>092H</i>
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</i>
<i>ORG</i>	<i>DATA</i>	<i>0000H</i>
<i>LJMP</i>	<i>DATA</i>	<i>MAIN</i>

<i>ORG</i>	<i>0100H</i>	
START:		
<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>	
SENDDATA:		
<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>	
RECVACK:		
<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:		
<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>		
SENDACK:		
<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>	
SENDNAK:		
<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>	
<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>	
STOP:		
<i>MOV</i>	<i>A,#00000110B</i>	;发送 STOP 命令
<i>MOV</i>	<i>DPTR,#I2CMSCR</i>	
<i>MOVX</i>	<i>@DPTR,A</i>	
<i>JMP</i>	<i>WAIT</i>	
WAIT:		
<i>MOV</i>	<i>DPTR,#I2CMSST</i>	;清中断标志
<i>MOVX</i>	<i>A,@DPTR</i>	
<i>JNB</i>	<i>ACC.6, WAIT</i>	
<i>ANL</i>	<i>A,#NOT 40H</i>	
<i>MOVX</i>	<i>@DPTR,A</i>	
<i>RET</i>		
DELAY:		
<i>MOV</i>	<i>R0,#0</i>	
<i>MOV</i>	<i>RI,#0</i>	

DELAY1:

```

NOP
NOP
NOP
NOP
DJNZ      R1,DELAY1
DJNZ      R0,DELAY1
RET

```

MAIN:

```

MOV      SP, #5FH
MOV      P0M0, #00H
MOV      P0M1, #00H
MOV      P1M0, #00H
MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

MOV      P_SW2,#80H

MOV      A,#III00000B ;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,#5AH
CALL    SENDATA        ;发送设备地址(010_1101B)+写命令(0B)
CALL    RECVACK
MOV      A,#000H         ;发送存储地址
CALL    SENDATA
CALL    RECVACK
MOV      A,#12H          ;写测试数据 1
CALL    SENDATA
CALL    RECVACK
MOV      A,#78H          ;写测试数据 2
CALL    SENDATA
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
CALL    SENDATA        ;发送设备地址(010_1101B)+写命令(0B)
CALL    RECVACK
MOV      A,#000H         ;发送存储地址
CALL    SENDATA
CALL    RECVACK
CALL    START          ;发送起始命令
MOV      A,#5BH          ;发送设备地址(010_1101B)+读命令(1B)

```

<i>CALL</i>	<i>SENDATA</i>
<i>CALL</i>	<i>RECVACK</i>
<i>CALL</i>	<i>RECVDATA</i>
	; 读取数据 1
<i>MOV</i>	<i>P0,A</i>
<i>CALL</i>	<i>SENDACK</i>
<i>CALL</i>	<i>RECVDATA</i>
	; 读取数据 2
<i>MOV</i>	<i>P2,A</i>
<i>CALL</i>	<i>SENDNAK</i>
<i>CALL</i>	<i>STOP</i>
	; 发送停止命令
<i>JMP</i>	\$
<i>END</i>	

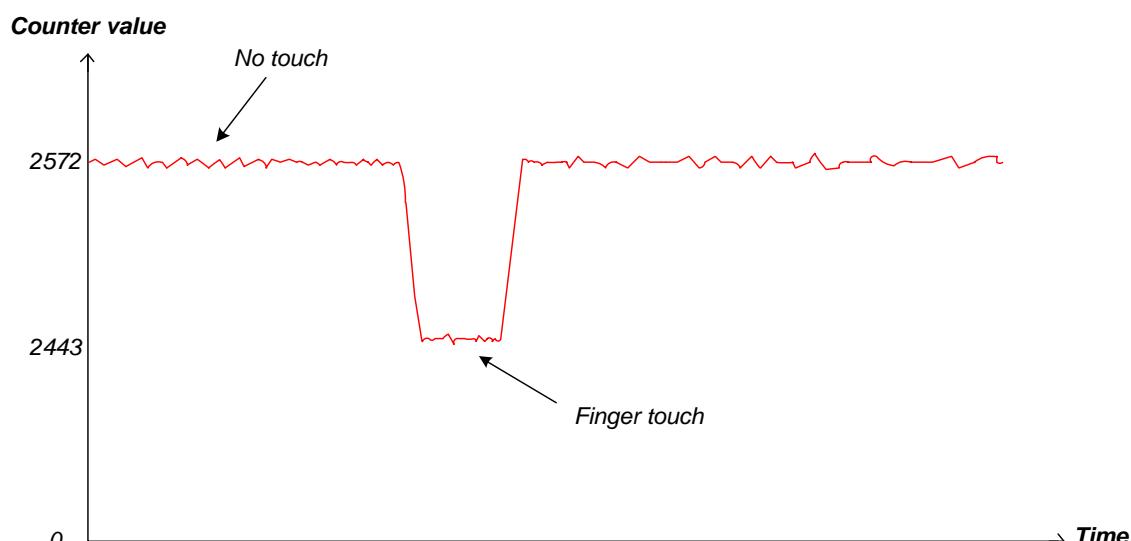
STCMCU

21 触摸按键控制器，先工程样片，可少量送样

产品线	触摸按键
STC8G1K08 系列	
STC8G1K08-8Pin 系列	
STC8G1K08A 系列	
STC8G2K64S4 系列	
STC8G2K64S2 系列	
STC8G1K08T 系列	●
STC15H2K64S4 系列	

STC8G 系列部分型号的单片机内部集成了一个触摸按键控制器 (TSU)，最大能连接 16 个按键，能够侦测手指触摸于按键电极后导致的微小电容变化，并将之量化为 16-bit 的数字。从原理上看来，TSU 模块与 16-bit ADC 雷同，差异仅是 ADC 为侦测及量化仿真电压或电流，而 TSU 是侦测及量化电容大小。触摸按键的感测，应用时需要外加一个约 10nF 至 47nF 之间的固定电容当参考，如果应用环境温度变化快速且激烈的环境下，应选用低温飘特性电容，才能保持 TSU 输出数据不会有太大变化，避免软件误判。

下图为一说明，对某一个按键做连续感测，在没有手指碰触时，计数器输出值为 2572 附近，当有手指触摸后，计数器输出变成 2443 附近，有无手指触摸的计数器输出值差异约 5%



TSU 模块最大能连接 16 个按键，通过 TSCHEN1 及 TSCHEN2 两个八位寄存器能对每一个触摸按键口独立使能，不做触摸按键的 I/O 口，仍能维持其原本 GPIO 或特殊功能如 LED。开关电容电路的频率，透过 SCR[2:0]来选择，建议频率不要超过 12.5MHz，能避免产生较大的误差，TSU 模块内部比较器的参考电压，透过 TSVR[1:0]来选取，共有四段可选，改变参考电压会改变触摸感测时间及灵敏度。

触摸按键扫描可配置成连续不断地循环扫描或是扫描已使能键一轮后就停止，这功能由 SINGLE 位来控制，配置寄存器 TSSAMP[1:0]允许 TSU 模块对同一个通道做最多连续 4 次的扫描采样并计算数据的平均值，这有硬件滤波的效果。而 WAIT 位，则能让 TSU 模块在 TSIF 标志位为 1 时，进入等待模式，直到软件将 WAIT 位清除为 0，TSU 才会继续做按键扫描，这有助于负担沉重的 CPU 有足够的时间做处理。外部电容 Cref 值推荐 10nF~47nF 范围，Cref 的起始放电至零的时间必需充足，这可以透过 DT[2:0] 弹性调整放电时间。

当完成一个按键的扫描时，16 位计数器的输出值会被写入 TSDATAH 及 TSDATAL 中，硬件也同时将标志位 TSIF 设成 1，同时会把已扫描完成的触摸信道号码写入 TSDNCHN[3:0]，若模块外部中断控制器使能，TSIF 能向 CPU 发出中断请求，软件能读取 TSDNCHN[3:0]寄存器内容以判别发出 TSIF 中断的是哪一个触摸通道，而软件读取 TSWKCHN[3:0]与 TSGO 则能实时了解 TSU 模块正在扫描的状态与正在扫描的按键号。如果 16 位计数器发生溢位情形，硬件会将 TSDOV 标志位设成 1。

TSU 模块能与 LED 驱动电路做 I/O 口分时复用，当 TSRT 寄存器内容不为零时，即代表当 TSGO 使能 TSU 工作后是要一起使能 LED 驱动电路并分时复用 I/O 口。因此在 TSGO 使能前，软件必需先依步配置好 TSU 模块与 LED 驱动电路的相关寄存器，最后才使能 TSGO。在 TSU/LED 分时多任务模式下，为了维持 LED 固定的 frame rate，若分配给按键扫描的时间已到达但正扫描的按键尚未完成，则该按键将于 LED 分时工作时间终止后，由重新扫描该未完成按键开始新一轮按键扫描。

21.1 触摸按键控制器内部框架图

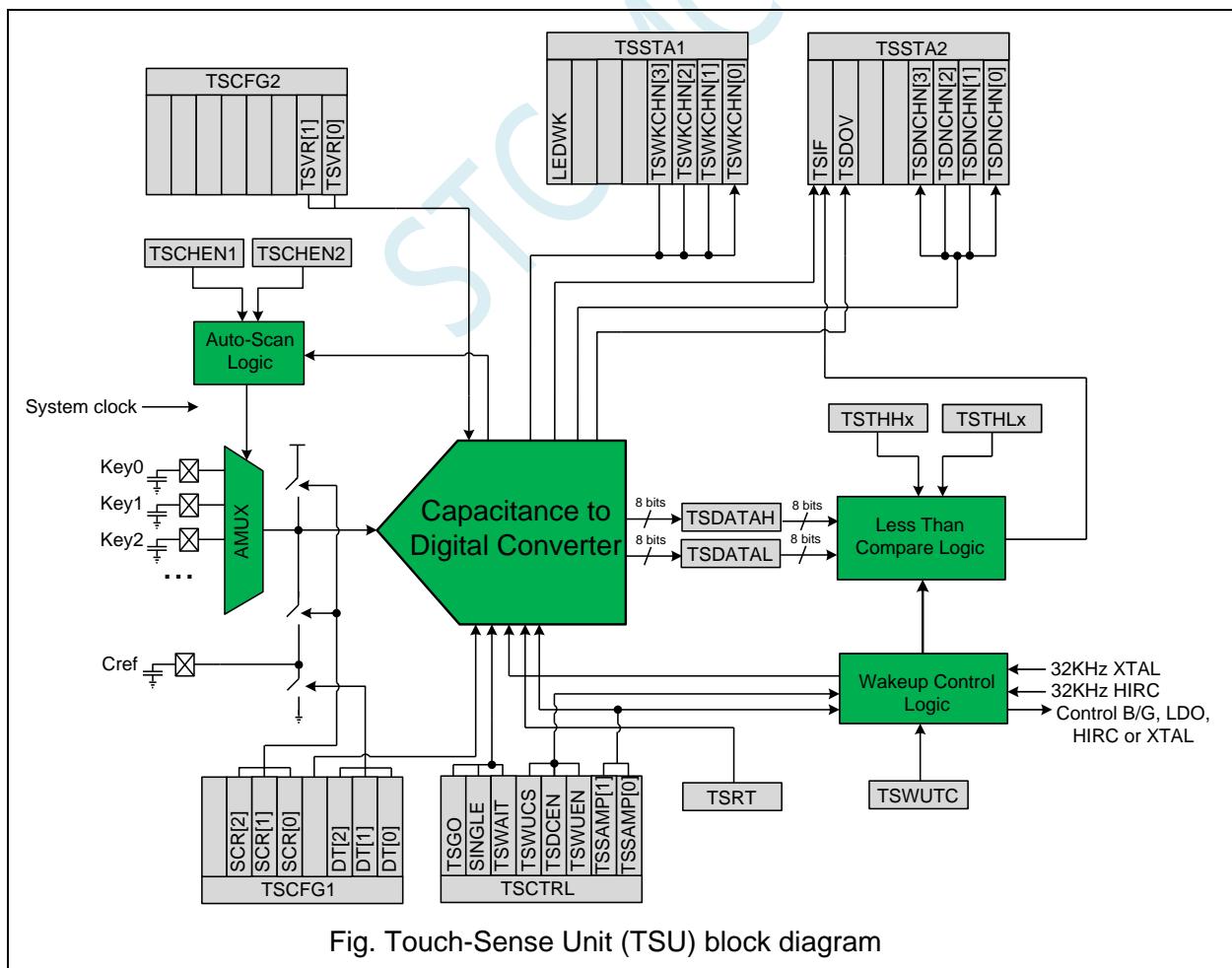


Fig. Touch-Sense Unit (TSU) block diagram

21.2 低功耗触摸唤醒

TSU 模块内部有一专用计时与控制电路, 能接受外部 32KHz 晶振或内部 32KHz RC 振荡电路为时钟源, 每隔一段时间唤醒 TSU 模块做按键扫描, 以 duty control 方式实现低功耗触摸唤醒功能。每一个触摸通道有一个专属的 16 位阀值寄存器{TSTHHx,TSTHLx}, 当唤醒使能位 TSWUEN 置 1 及数字比较器使能位 TSDCEN 置 1 后 MCU 进入 power-down 状态, 整个芯片进入低功耗触摸唤醒模式, TSU 模块能重复不断地规则性短暂自我唤醒做按键扫描以便唤醒 CPU, 当按键扫描的数据结果(或者数据平均结果)小于设定的阀值时, 硬件会将 TSIF 置 1 并唤醒 CPU 离开 power-down 状态。TSU 模块内部有一硬件平均电路, 能够对同一通道最多连续四次的扫描采样数据结果做平均计算, 这透过 TSSAMP[1:0]寄存器来配置采样的次数, 而{TSDATAH, TSDATAL}则存放着数据平均值。

21.3 仅做触摸按键功能时的操作步骤

1. 选择需要扫描的信道, 寄存器为 TSCHEN1 及 TSCHEN2
2. 设置 TSRT 内容为 0x00, 代表没有开启 LED 驱动分时多任务功能
3. 依据 Cref 大小及触摸按键电容大小配置开关频率 SCR[2:0]与放电时间 DT[2:0], 根据所需要的扫描时间及灵敏度选择内部比较器参考电压 TSVR[1:0]
4. 配置 SINGLE 位来决定是扫描一轮自动停止或者是连续不停扫描, 配置 TSSAMP[1:0]能让一通道的重复采样最多四次, 若 CPU 任务繁重, 配置 TSWAIT 以利用 TSIF 状态来延时下一个信道的扫描
5. 必要的话, 配置 TSDCEN 以使能内部数字比较功能
6. 设置 TSGO=1, 触摸按键开始扫描, 软件读取 TSWKCHN[3:0]来了解当下在扫描那一个信道, 每扫完一个通道, 硬件会将 TSIF 设置成 1, 同时完成的信道号码会写入 TSDNCHN[3:0]中, 若有溢位发生, TSOV 也会被设置成 1, 软件应读取这些寄存器以决定接下来怎么做。TSIF 及 TSOV 仅能由硬件设置及软件清除
7. 若 SINGLE=1, 一轮扫完后硬件将自动将 TSGO 清除并结束扫描, 否则 TSGO 维持在 1, 继续新一轮扫描。
8. 若软件想停止触摸按键扫描, 可随时将 TSGO 设置成 0。

21.4 低功耗触摸按键唤醒功能的操作步骤

1. 选择需要扫描的信道, 寄存器为 TSCHEN1 及 TSCHEN2
2. 务必设置 TSRT 内容为 0x00, 此时不能开启 LED 驱动分时多任务功能
3. 配置开关频率 SCR[2:0], 放电时间 DT[2:0]及选择内部比较器参考电压 TSVR[1:0]
4. 配置 SINGLE 位为 0 做连续不停扫描, 配置 TSWAIT 为 0。
5. 配置 TSWUCS, 决定唤醒控制器的时钟来源是外部 32KHz 晶振或内部 32KHz IRC。
6. 配置 TSWUTC, 以决定 TSU 每隔多久需要起来干活, 干活完就自动进入省电模式
7. 配置 TSSAMP[1:0]以决定每一个信道的扫描采样次数, 配置 TSDCEN=1 使能内部数字比较功能
8. 配置好每一个信道的唤醒阀值{TSTHHx, TSTHLx}, 这阀值会与扫描结果的平均值做比较
9. 使能 TSWUEN=1, 设置 TSPD=1 以关闭 TSU 模块模拟电源, 以及使能 TSIF 能唤醒 CPU, 然后让 MCU 进入 power-down 状态。MCU 一旦进入 power-down 状态, TSU 内部的唤醒控制器即开始工作, 周期性控制 TSU 模块的电源开关、按键扫描、数据比较等等。
10. 当数据结果低于所设定的阀值, 硬件会将 TSIF 设成 1, 将该按键号码写入 TSDNCHN[3:0]中 CPU 被唤醒, 低功耗触摸唤醒模式结束。

11. CPU 被唤醒后, 除了可以直接读取 TSDNCHN[3:0]认定是那个按键有触摸外, 也可以在正常工作模式再施行按键扫描以确认是否唤醒是因为噪声干扰所致。

21.5 触摸按键相关的寄存器

符号	描述	地址	位地址与符号								复位值		
			B7	B6	B5	B4	B3	B2	B1	B0			
TSCHEN1	触摸按键使能寄存器 1	FB40H	TKEN7	TKEN6	TKEN5	TKEN4	TKEN3	TKEN2	TKEN1	TKEN0	0000,0000		
TSCHEN2	触摸按键使能寄存器 2	FB41H	TKEN15	TKEN14	TKEN13	TKEN12	TKEN11	TKEN10	TKEN9	TKEN8	0000,0000		
TSCFG1	触摸按键配置寄存器 1	FB42H	-	SCR[2:0]			-	DT[2:0]			x000,x000		
TSCFG2	触摸按键配置寄存器 2	FB43H	-	-	-	-	-	-	TSVR[1:0]	xxxx,xx00			
TSWUTC	触摸按键唤醒控制寄存器	FB44H									0000,0001		
TSCTRL	触摸按键控制寄存器	FB45H	TSGO	SINGLE	TSWAIT	TSWUCS	TSDCEN	TSWUEN	TSSAMP[1:0]		0000,0000		
TSSTA1	触摸按键状态寄存器 1	FB46H	LEDWK	-	-	-		TSWKCHN[3:0]			0xxx,0000		
TSSTA2	触摸按键状态寄存器 2	FB47H	TSIF	TSDOV	-	-		TSDNCHN[3:0]			00xx,0000		
TSRT	触摸按键时间控制寄存器	FB48H									0000,0001		
TSDATH	触摸按键数据高字节	FB49H									0000,0000		
TSDATL	触摸按键数据低字节	FB4AH									0000,0000		
TSTH00H	触摸按键 0 门槛值高字节	FB50H									0000,0000		
TSTH00L	触摸按键 0 门槛值低字节	FB51H									0000,0000		
TSTH01H	触摸按键 1 门槛值高字节	FB52H									0000,0000		
TSTH01L	触摸按键 1 门槛值低字节	FB53H									0000,0000		
TSTH02H	触摸按键 2 门槛值高字节	FB54H									0000,0000		
TSTH02L	触摸按键 2 门槛值低字节	FB55H									0000,0000		
TSTH03H	触摸按键 3 门槛值高字节	FB56H									0000,0000		
TSTH03L	触摸按键 3 门槛值低字节	FB57H									0000,0000		
TSTH04H	触摸按键 4 门槛值高字节	FB58H									0000,0000		
TSTH04L	触摸按键 4 门槛值低字节	FB59H									0000,0000		
TSTH05H	触摸按键 5 门槛值高字节	FB5AH									0000,0000		
TSTH05L	触摸按键 5 门槛值低字节	FB5BH									0000,0000		
TSTH06H	触摸按键 6 门槛值高字节	FB5CH									0000,0000		
TSTH06L	触摸按键 6 门槛值低字节	FB5DH									0000,0000		
TSTH07H	触摸按键 7 门槛值高字节	FB5EH									0000,0000		
TSTH07L	触摸按键 7 门槛值低字节	FB5FH									0000,0000		
TSTH08H	触摸按键 8 门槛值高字节	FB60H									0000,0000		
TSTH08L	触摸按键 8 门槛值低字节	FB61H									0000,0000		
TSTH09H	触摸按键 9 门槛值高字节	FB62H									0000,0000		
TSTH09L	触摸按键 9 门槛值低字节	FB63H									0000,0000		
TSTH10H	触摸按键 10 门槛值高字节	FB64H									0000,0000		
TSTH10L	触摸按键 10 门槛值低字节	FB65H									0000,0000		
TSTH11H	触摸按键 11 门槛值高字节	FB66H									0000,0000		
TSTH11L	触摸按键 11 门槛值低字节	FB67H									0000,0000		
TSTH12H	触摸按键 12 门槛值高字节	FB68H									0000,0000		
TSTH12L	触摸按键 12 门槛值低字节	FB69H									0000,0000		

TSTH13H	触摸按键 13 门槛值高字节	FB6AH							0000,0000
TSTH13L	触摸按键 13 门槛值低字节	FB6BH							0000,0000
TSTH14H	触摸按键 14 门槛值高字节	FB6CH							0000,0000
TSTH14L	触摸按键 14 门槛值低字节	FB6DH							0000,0000
TSTH15H	触摸按键 15 门槛值高字节	FB6EH							0000,0000
TSTH15L	触摸按键 15 门槛值低字节	FB6FH							0000,0000

21.5.1 触摸按键使能寄存器 (TSCHENn))

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TSCHEN1	FB40H	TKEN7	TKEN6	TKEN5	TKEN4	TKEN3	TKEN2	TKEN1	TKEN0
TSCHEN2	FB41H	TKEN15	TKEN14	TKEN13	TKEN12	TKEN11	TKEN10	TKEN9	TKEN8

TKENn: 触摸按键使能位 (n=0~15)

0: 相应的 TKn 管脚为 GPIO

1: 相应的 TKn 管脚为触摸按键

21.5.2 触摸按键配置寄存器 (TSCFGn)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TSCFG1	FB42H	-	SCR[2:0]			-	DT[2:0]		
TSCFG2	FB43H	-	-	-	-	-	-	-	TSVR[1:0]

SCR: 配置触摸按键控制器内部的开关电容工作频率 (频率越高充电时间越短)

$$\text{开关电容工作频率} = \frac{\text{系统工作频率}}{2 * (\text{SCR}[2:0] + 1)}$$

DT[2:0]: 配置触摸按键控制器内部的Cref的起始对地放电时间

DT[2:0]	放电时间
000	125个系统时钟
001	250个系统时钟
010	500个系统时钟
011	1000个系统时钟
100	2000个系统时钟
101	2500个系统时钟
110	5000个系统时钟
111	7500个系统时钟

TSVR[1:0]: 配置触摸按键控制器内部的参考电压

TSVR[1:0]	参考电压
00	1/4 AVcc

01	1/2 AVcc
10	5/8 AVcc
11	3/4 AVcc

21.5.3 触摸按键低电模式唤醒时间控制寄存器 (TSWUTC)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TSWUTC	FB44H								

TSWUTC 寄存器用于配置每隔多久唤醒一次触摸按键控制器

$$\text{唤醒频率} = \frac{F_{32K}}{32 * 8 * \text{TSWUTC}[7:0]}$$

例如：若外挂 32.768KHz 的晶振，且 TSWUTC = 0x80，

则触摸按键控制器的唤醒频率则为 $32768 / (32 * 8 * 0x80) = 1\text{Hz}$ ，即 1 秒唤醒一次。

注意：若设置的唤醒频率太快，唤醒的时间不足以完成一轮按键扫描，则触摸按键控制器会持续不停的扫描，而无法进入省电模式。

21.5.4 触摸按键控制寄存器 (TSCTRL)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TSCTRL	FB45H	TSGO	SINGLE	TSWAIT	TSWUCS	TSDCEN	TSWUEN	TSSAMP[1:0]	

TSGO：触摸按键控制器启动控制

- 0：触摸按键控制器不工作
- 1：触摸按键控制器开始启动。

SINGLE：单次扫描模式控制

- 0：重复扫描模式
 - 1：单次扫描模式。当 TSGO=1 时，当完成一轮按键扫描后，硬件自动将 TSGO 清 0 终止扫描。
- 注意：**当 TSGO=1，且 TSRT 不为 0 时，代表触摸按键控制器与 LED 驱动器为共享 GPIO 且分时多工，此时 SINGLE 控制位无效。

TSWAIT：触摸按键控制器等待控制

- 0：触摸按键控制器自动重复扫描
- 1：完成一轮扫描后，TSIF 被硬件置 1，此时触摸按键控制器会暂停扫描，直到 TSIF 标志位被清 0 后才开始下一轮的扫描。

TSWUCS：低功耗模式下触摸按键控制器时钟源选择

- 0：低功耗模式下触摸按键控制器时钟源为内部 32K 的 IRC
- 1：低功耗模式下触摸按键控制器时钟源为外部 32K 晶振

TSDCEN：触摸按键控制器内部16位数字比较器控制位

- 0：关闭触摸按键控制器内部 16 位数字比较器
- 1：使能触摸按键控制器内部 16 位数字比较器

注意：当 TSU 内部数字比较器被使能，只有当触摸感测数据结果 { TSDATAH, TSDATAL } 小于该

对应通道所设定的阀值 { TSTHHx, TSTHLx }时, TSIF 才会被设置 1。此功能用于低功耗模式下可实行触摸唤醒 CPU

TSWUEN: 触摸按键控制器低功耗唤醒使能位

0: 关闭触摸按键控制器低功耗唤醒功能

1: 使能触摸按键控制器低功耗唤醒功能。使能后当 MCU 进入 power-down 状态, 旋即进入触摸按键低功耗唤醒 MCU 模式。在此模式下, 触摸按键控制器内部低功耗时序控制电路将周期性使能 TSU 做按键摸扫描, 以 duty control 维持极低的平均电流。TSWUEN 只有在 MCU 进入 power-down mode 以及 TSIF 为 0 时才会真正有作用。

注意: 此芯片内部有两个 32K 的振荡器, 一个是外部的 32K 晶振, 另一个是内部的 IRC32K 振荡器。在非 STOP-mode 的情况下: 内部 IRC32K 的使能机制单纯是透过 XFR:IRC32KCR[7]; 外部的 X32K 的使能机制单纯是透过 XFR:X2KCR[7]。在 STOP-mode 的情况下: 内部 IRC32K 的使能机制除了 XFR:IRC32KCR[7] 必须被设 1 外, 还必须 SFR:ENWKT 或是 XFR:TSWUEN 被设 1; 外部的 X32K 的使能机制除了 XFR:X32KCR[7] 必须被设 1 外, 还必须 XFR:TSWUEN 被设 1。强调的重点是: TSWUEN 在 STOP-mode 的情况下还扮演启动或是不启动 32K 振荡器的角色。

TSSAMP[1:0]: 单一触摸通道重复扫描次

TSSAMP [1:0]	重复扫描次
00	1次
01	2次
10	3次
11	4次

注意: 中断标志位 TSIF 只有当扫描同一按键的次数到达 TSSAMP 的配置时才会被置 1, 这时 {TSDATAH, TSDATAL}里是写入的是数次结果的平均值。但若有任一次溢位发生, 硬件会将 TSDOV 置 1。在 TSWKEN 使能的情况下, 平均值必须小于门槛内容才会导致中断标志位 TSIF 置 1, 才能唤醒 CPU。

21.5.5 触摸按键状态寄存器 1 (TSSTA1)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TSSTA1	FB46H	LEDWK	-	-	-				TSWKCHN[3:0]

LEDWK: 触摸按键控制器与LED驱动器分时多工时的工作状态

0: LED 驱动器处于等待状态, 触摸按键控制器处于工作状态

1: LED 驱动器处于工作状态, 触摸按键控制器处于等待状态

TSWKCHN [3:0]: 触摸通道扫描状态

TSWKCHN [3:0]	触摸通道状态
0000	触摸通道 0 正在被扫描
0001	触摸通道 1 正在被扫描
0010	触摸通道 2 正在被扫描
0011	触摸通道 3 正在被扫描
0100	触摸通道 4 正在被扫描

0101	触摸通道 5 正在被扫描
0110	触摸通道 6 正在被扫描
0111	触摸通道 7 正在被扫描
1000	触摸通道 8 正在被扫描
1001	触摸通道 9 正在被扫描
1010	触摸通道 10 正在被扫描
1011	触摸通道 11 正在被扫描
1100	触摸通道 12 正在被扫描
1101	触摸通道 13 正在被扫描
1110	触摸通道 14 正在被扫描
1111	触摸通道 15 正在被扫描

21.5.6 触摸按键状态寄存器 2 (TSSTA2)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TSSTA2	FB47H	TSIF	TSDOV	-	-	TSDNCHN[3:0]			

TSIF: 触摸按键通道扫描完成标志 (软件写1清0)

0: 还未完成扫描

1: 当完成 TSSAMP 所设定的扫描次数时, 硬件会将 TSIF 置 1, TSIF 可以对 CPU 发出中断请求。
如果是在低功耗唤醒模式下, 还必需加上扫描数据结果值低于设定的门槛, TSIF 才会被设成 1。

注意: TSIF 只能由硬件置 1, 软件无法将 TSIF 设成 1。特别注意的是软件写 1 至 TSIF 会将 TSIF 清为 0, 软件写 0 至 TSIF 则对内容毫无影响。当 TSWAIT=1 时且 TSIF 为 1 时, TSU 处于暂停等待状态, 等待 CPU 忙完后将 TSIF 清为 0 后才会继续下一个按键的扫描。

TSDOV: 按键扫描数据溢出标志 (软件写1清0)

0: 按键扫描数据没有溢出, 扫描数据小于或等于 0xFFFF

1: 按键扫描数据发生溢出, 扫描数据大于 0xFFFF。此时软件配置(如 TSVR)或系统硬件必需调整以免溢位发生。TSDOV 仅能由硬件设 1, 软件必需对 TSDOV 写入 1 方能将其清除, 软件对此位写 0 无任何作用。

TSDNCHN [3:0]: 触摸通道完成状态

TSDNCHN [3:0]	触摸通道状态
0000	触摸通道 0 完成扫描
0001	触摸通道 1 完成扫描
0010	触摸通道 2 完成扫描
0011	触摸通道 3 完成扫描
0100	触摸通道 4 完成扫描
0101	触摸通道 5 完成扫描
0110	触摸通道 6 完成扫描
0111	触摸通道 7 完成扫描
1000	触摸通道 8 完成扫描

1001	触摸通道 9 完成扫描
1010	触摸通道 10 完成扫描
1011	触摸通道 11 完成扫描
1100	触摸通道 12 完成扫描
1101	触摸通道 13 完成扫描
1110	触摸通道 14 完成扫描
1111	触摸通道 15 完成扫描

21.5.7 触摸按键时间控制寄存器 (TSRT)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TSRT	FB48H								

TSRT 寄存器用于配置触摸按键控制器和 LED 驱动器分时多工。

当 TSRT 不为 00 时，表示触摸按键控制器和 LED 驱动器为分时多工模式。触摸按键控制器分得的时间长度为 $TSRT * T_{LED}$ (T_{LED} 请参考 LED 驱动说明部分)

21.5.8 触摸按键数据寄存器 (TSDAT)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TSDATH	FB49H								TSDAT[15:8]
TSDATL	FB4AH								TSDAT[7:0]

TSDAT[15:0]: 触摸按键扫描的数据

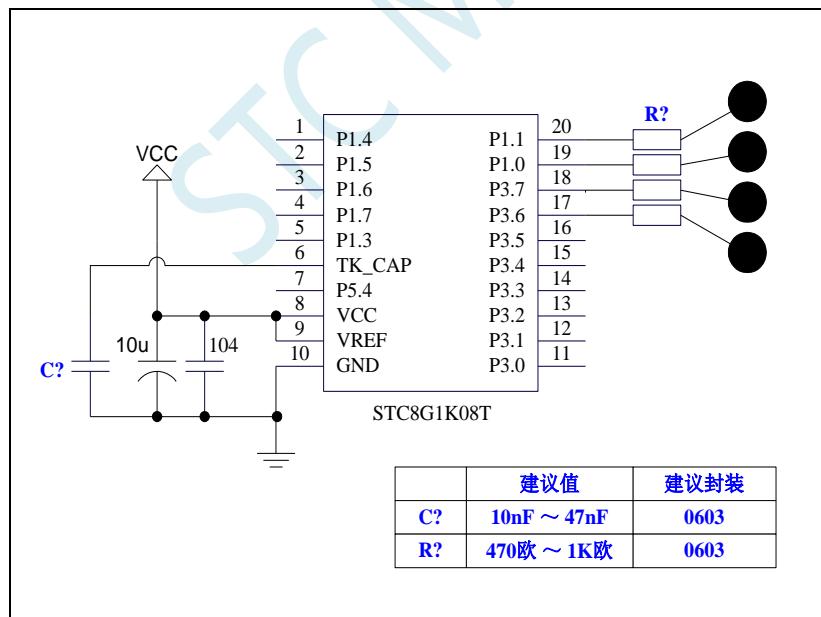
21.5.9 触摸按键门槛值寄存器 (TSTH)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TSTH00H	FB50H								TSTH00[15:8]
TSTH00L	FB51H								TSTH00[7:0]
TSTH01H	FB52H								TSTH01[15:8]
TSTH01L	FB53H								TSTH01[7:0]
TSTH02H	FB54H								TSTH02[15:8]
TSTH02L	FB55H								TSTH02[7:0]
TSTH03H	FB56H								TSTH03[15:8]
TSTH03L	FB57H								TSTH03[7:0]
TSTH04H	FB58H								TSTH04[15:8]
TSTH04L	FB59H								TSTH04[7:0]
TSTH05H	FB5AH								TSTH05[15:8]
TSTH05L	FB5BH								TSTH05[7:0]
TSTH06H	FB5CH								TSTH06[15:8]
TSTH06L	FB5DH								TSTH06[7:0]
TSTH07H	FB5EH								TSTH07[15:8]
TSTH07L	FB5FH								TSTH07[7:0]
TSTH08H	FB60H								TSTH08[15:8]

TSTH08L	FB61H	TSTH08[7:0]
TSTH09H	FB62H	TSTH09[15:8]
TSTH09L	FB63H	TSTH09[7:0]
TSTH10H	FB64H	TSTH10[15:8]
TSTH10L	FB65H	TSTH10[7:0]
TSTH11H	FB66H	TSTH11[15:8]
TSTH11L	FB67H	TSTH11[7:0]
TSTH12H	FB68H	TSTH12[15:8]
TSTH12L	FB69H	TSTH12[7:0]
TSTH13H	FB6AH	TSTH13[15:8]
TSTH13L	FB6BH	TSTH13[7:0]
TSTH14H	FB6CH	TSTH14[15:8]
TSTH14L	FB6DH	TSTH14[7:0]
TSTH15H	FB6EH	TSTH15[15:8]
TSTH15L	FB6FH	TSTH15[7:0]

TSTHn[15:0]: 触摸按键扫描数据门槛值。当数字比较器被使能后，只有当扫描数据低于此门坎值，TSIF 才会被硬件置 1。

21.6 基本参考电路图以及注意事项



注意：参考线路图中，C?为触摸按键的灵敏度调节电容（**不要使用陶瓷电容，强烈建议使用温度误差较小的独石电容**），R?为ESD保护电阻，在PCB布板时，C?与R?请务必尽量靠近IC管脚。

22 LED 驱动器

产品线	LED
STC8G1K08 系列	
STC8G1K08-8Pin 系列	
STC8G1K08A 系列	
STC8G2K64S4 系列	
STC8G2K64S2 系列	
STC8G1K08T 系列	●
STC15H2K64S4 系列	

STC8G 系列部分型号的单片机内部集成了一个 LED 驱动器。

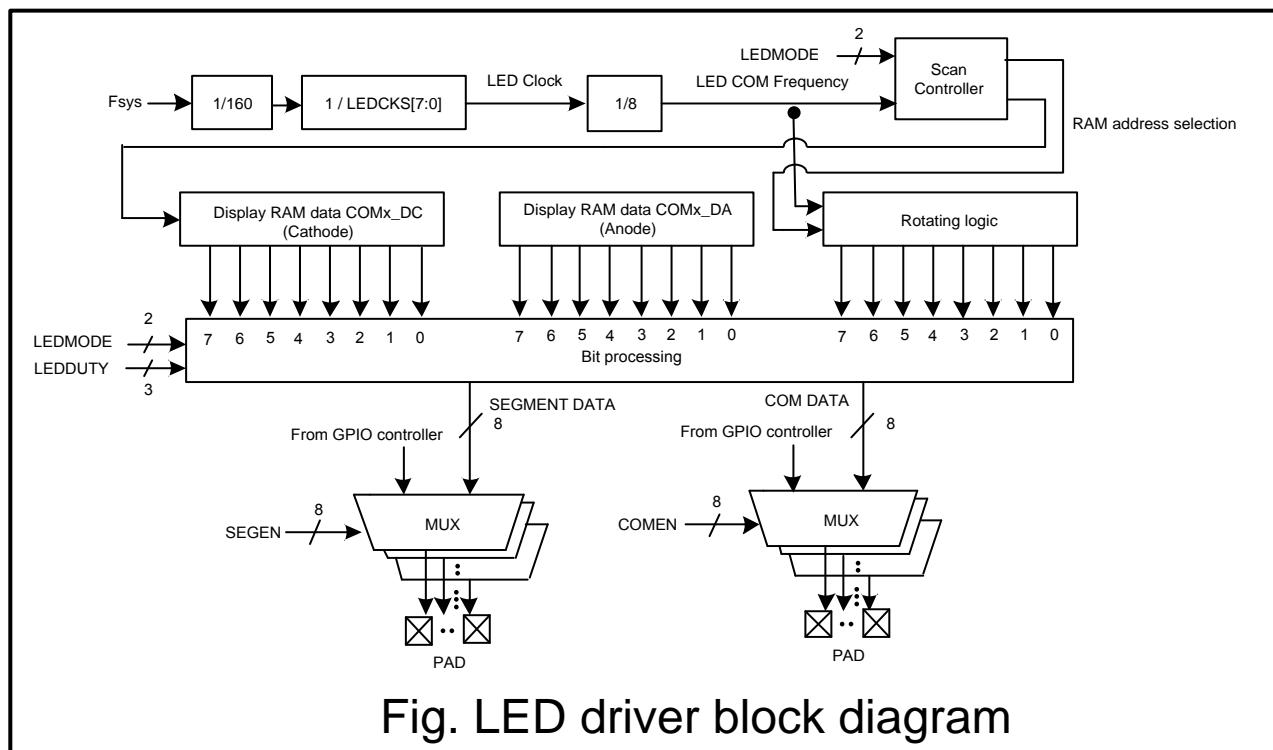
LED 驱动电路包含一个时序控制器，8 个 COM 输出引脚及 8 个 SEGMENT 输出引脚。每一个引脚有一个对应的寄存器使能位，能独立控制该引脚使能与否，没使能的引脚能当做 GPIO 或其它功能的引脚。

LED 驱动支持共阴、共阳、共阴/共阳三种模式，同时能选择 1/8 ~ 8/8 占空比来调节灰度，因此仅需透过软件即可调节 LED 及数码管的亮度。

上电复位后，LEDON 使能位为 0，LED 驱动电路关闭。配置 LEDON 为 1 使能 LED 驱动电路，当 LEDMODE=00 时，驱动电路工作于共阴模式，此时被选中的 COM 输出低电平，被选中的 SEGMENT 且希望 LED 发亮则输出高电平，因此 SEGMENT 与 COM 两端点间的 LED 顺向偏压导通发亮。同理，当 LEDMODE=01 时，驱动电路工作于共阳模式，此时被选中的 COM 输出高电平，被选中的 SEGMENT 且希望 LED 发亮则输出低电平，因此 SEGMENT 与 COM 两端点间的 LED 逆向偏压导通发亮。当 LEDMODE=10 时，驱动电路工作于共阴/共阳分时驱动模式，COM 的电平为低电平与高电平分时交错，LED 导通发亮原理与共阴、共阳是相同的。

共阴模式及共阳模式下 display RAM 是独立的地址，共阴/共阳分时多任务模式下 display RAM 的位置也是从独立的地址去读取。

22.1 LED 驱动器内部框架图



22.2 LED 驱动相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
COMEN	COM 使能寄存器	FB00H	C7EN	C6EN	C5EN	C4EN	C3EN	C2EN	C1EN	C0EN	0000,0000
SEGENL	SEG 使能寄存器	FB01H	S7EN	S6EN	S5EN	S4EN	S3EN	S2EN	S1EN	SOEN	0000,0000
SEGENH	SEG 使能寄存器	FB02H	S15EN	S14EN	S13EN	S12EN	S11EN	S10EN	S9EN	S8EN	0000,0000
LEDCTRL	LED 控制寄存器	FB03H	LEDON	-	LED MODE[1:0]	-	LEDDUTY[2:0]				0x00,x000
LEDCKS	LED 时钟分频寄存器	FB04H									0000,0001
COM0_DA_L	共阳模式显示数据	FB10H									0000,0000
COM1_DA_L	共阳模式显示数据	FB11H									0000,0000
COM2_DA_L	共阳模式显示数据	FB12H									0000,0000
COM3_DA_L	共阳模式显示数据	FB13H									0000,0000
COM4_DA_L	共阳模式显示数据	FB14H									0000,0000
COM5_DA_L	共阳模式显示数据	FB15H									0000,0000
COM6_DA_L	共阳模式显示数据	FB16H									0000,0000
COM7_DA_L	共阳模式显示数据	FB17H									0000,0000
COM0_DA_H	共阳模式显示数据	FB18H									0000,0000
COM1_DA_H	共阳模式显示数据	FB19H									0000,0000
COM2_DA_H	共阳模式显示数据	FB1AH									0000,0000
COM3_DA_H	共阳模式显示数据	FB1BH									0000,0000

COM4_DA_H	共阳模式显示数据	FB1CH							0000,0000
COM5_DA_H	共阳模式显示数据	FB1DH							0000,0000
COM6_DA_H	共阳模式显示数据	FB1EH							0000,0000
COM7_DA_H	共阳模式显示数据	FB1FH							0000,0000
COM0_DC_L	共阴模式显示数据	FB20H							0000,0000
COM1_DC_L	共阴模式显示数据	FB21H							0000,0000
COM2_DC_L	共阴模式显示数据	FB22H							0000,0000
COM3_DC_L	共阴模式显示数据	FB23H							0000,0000
COM4_DC_L	共阴模式显示数据	FB24H							0000,0000
COM5_DC_L	共阴模式显示数据	FB25H							0000,0000
COM6_DC_L	共阴模式显示数据	FB26H							0000,0000
COM7_DC_L	共阴模式显示数据	FB27H							0000,0000
COM0_DC_H	共阴模式显示数据	FB28H							0000,0000
COM1_DC_H	共阴模式显示数据	FB29H							0000,0000
COM2_DC_H	共阴模式显示数据	FB2AH							0000,0000
COM3_DC_H	共阴模式显示数据	FB2BH							0000,0000
COM4_DC_H	共阴模式显示数据	FB2CH							0000,0000
COM5_DC_H	共阴模式显示数据	FB2DH							0000,0000
COM6_DC_H	共阴模式显示数据	FB2EH							0000,0000
COM7_DC_H	共阴模式显示数据	FB2FH							0000,0000

22.2.1 COM 使能寄存器 (COMEN)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
COMEN	FB00H	C7EN	C6EN	C5EN	C4EN	C3EN	C2EN	C1EN	C0EN

CnEN: COMn使能控制位 (n=0~7)

0: 不使能 COMn, 仍然为 GPIO

1: 使能 COMn, 对应的 I/O 在 LEDON=1 时输出 COM 的驱动波形。

22.2.2 SEG 使能寄存器 (SEGEN)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SEGENL	FB01H	S7EN	S6EN	S5EN	S4EN	S3EN	S2EN	S1EN	SOEN
SEGENH	FB02H	S15EN	S14EN	S13EN	S12EN	S11EN	S10EN	S9EN	S8EN

SnEN: SEGn使能控制位 (n=0~15)

0: 不使能 SEGn, 仍然为 GPIO

1: 使能 SEGn, 对应的 I/O 在 LEDON=1 时输出 SEG 的驱动波形。

22.2.3 LED 控制寄存器 (LEDCTRL)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LEDCTRL	FB03H	LEDON	-	LEDMODE[1:0]	-	LEDDUTY[2:0]			

LEDON: LED驱动器使能控制

0: 关闭 LED 驱动器

1: 使能 LED 驱动器。

LEDMODE[1:0]: LED驱动模式

LEDMODE[1:0]	驱动模式
00	共阴模式
01	共阳模式
10	共阴/共阳模式
11	保留

LEDDUTY[2:0]: LED灰度调节

LEDDUTY[2:0]	LED占空比	LED亮度
000	8/8	100%
001	7/8	87.5%
010	6/8	75%
011	5/8	62.5%
100	4/8	50%
101	3/8	37.5%
110	2/8	25%
111	1/8	12.5%

22.2.4 LED 时钟分频寄存器 (LEDCKS)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LEDCKS	FB04H								

LEDCKS: LED时钟分频控制。

$$\text{LED工作频率} = \frac{\text{SYSclk}}{160 * \text{LEDCKS}[7:0]}$$

寄存器 LEDCKS 的值若设置太大，会导致 LED 产生闪烁现象，一般 LED 刷新频率大于或等于 75Hz 则不会有明显的闪烁

$$\text{建议值: } \frac{\text{SYSclk}}{160 * 8 * N_{COM} * \text{LEDCKS}[7:0]} \geq 75\text{Hz}$$

N_{COM} : 使能的COM数, 若为共阴/共阳模式, 则为COM数的两倍

例如: 若单片机的工作频率为 11.0592MHz, LED 的工作模式为共阴/共阳模式, COMEN 设置为 0FFH, 即使能 8 个 COM, 则 $11059200 / 160 / 8 / 16 / \text{LEDCKS} \geq 75$, 得到 $\text{LEDCKS} \leq 7.2$, 所以 LEDCKS 建议设置为 7。

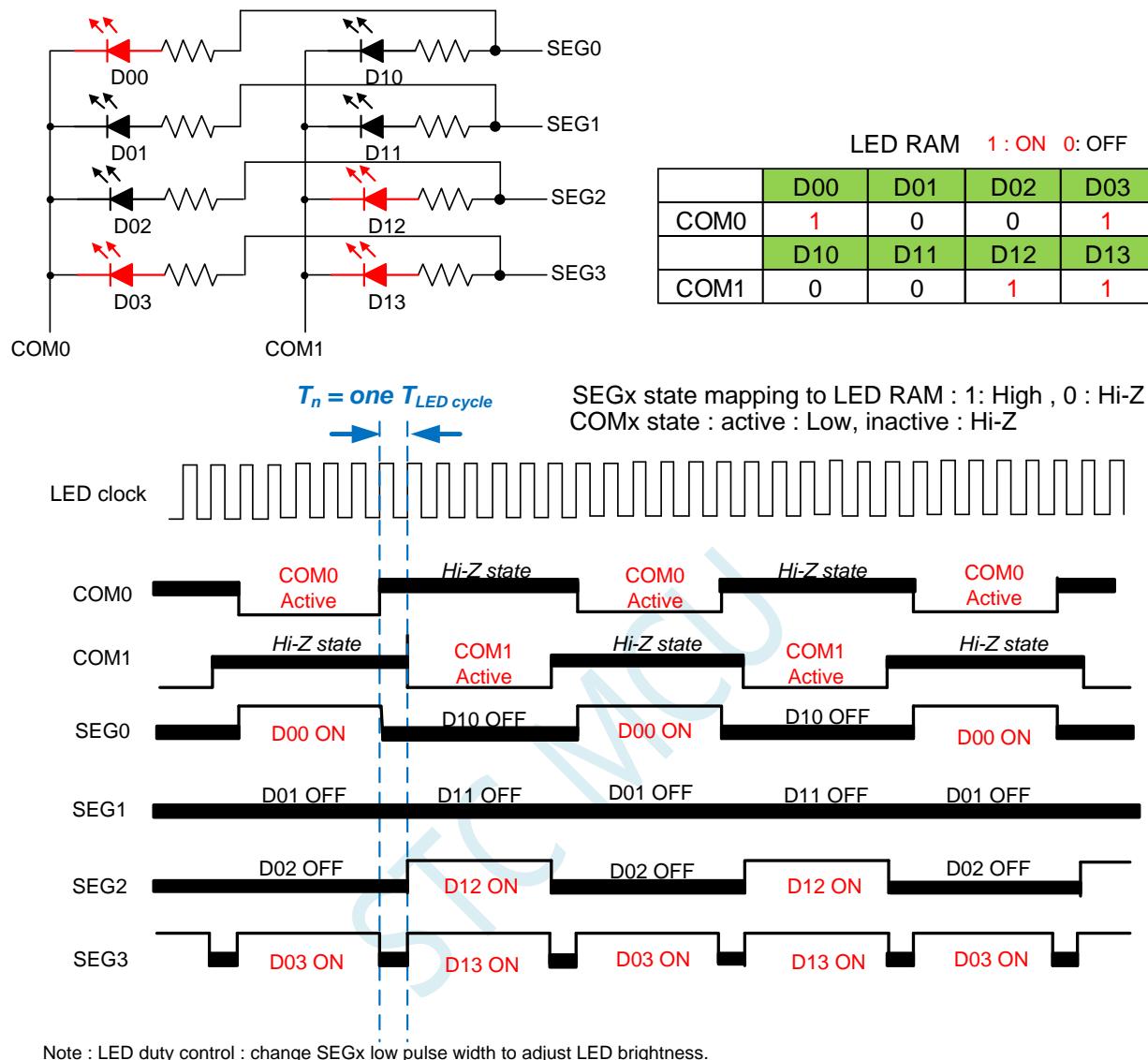
22.2.5 LED 共阳模式数据寄存器 (COMn_DA)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
COM0_DA_L	FB10H								
COM1_DA_L	FB11H								
COM2_DA_L	FB12H								
COM3_DA_L	FB13H								
COM4_DA_L	FB14H								
COM5_DA_L	FB15H								
COM6_DA_L	FB16H								
COM7_DA_L	FB17H								
COM0_DA_H	FB18H								
COM1_DA_H	FB19H								
COM2_DA_H	FB1AH								
COM3_DA_H	FB1BH								
COM4_DA_H	FB1CH								
COM5_DA_H	FB1DH								
COM6_DA_H	FB1EH								
COM7_DA_H	FB1FH								

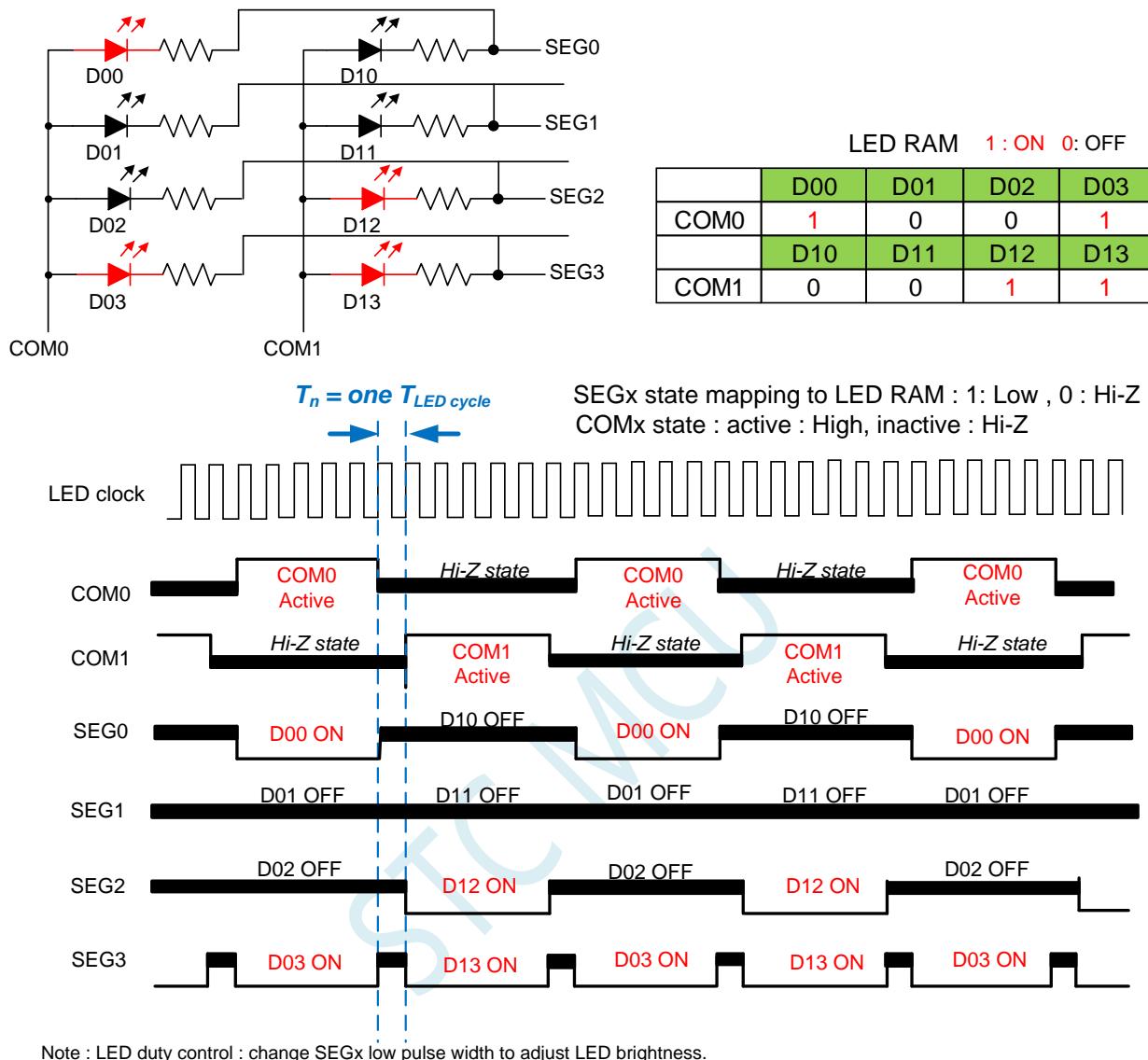
22.2.6 LED 共阴模式数据寄存器 (COMn_DC)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
COM0_DC_L	FB20H								
COM1_DC_L	FB21H								
COM2_DC_L	FB22H								
COM3_DC_L	FB23H								
COM4_DC_L	FB24H								
COM5_DC_L	FB25H								
COM6_DC_L	FB26H								
COM7_DC_L	FB27H								
COM0_DC_H	FB28H								
COM1_DC_H	FB29H								
COM2_DC_H	FB2AH								
COM3_DC_H	FB2BH								
COM4_DC_H	FB2CH								
COM5_DC_H	FB2DH								
COM6_DC_H	FB2EH								
COM7_DC_H	FB2FH								

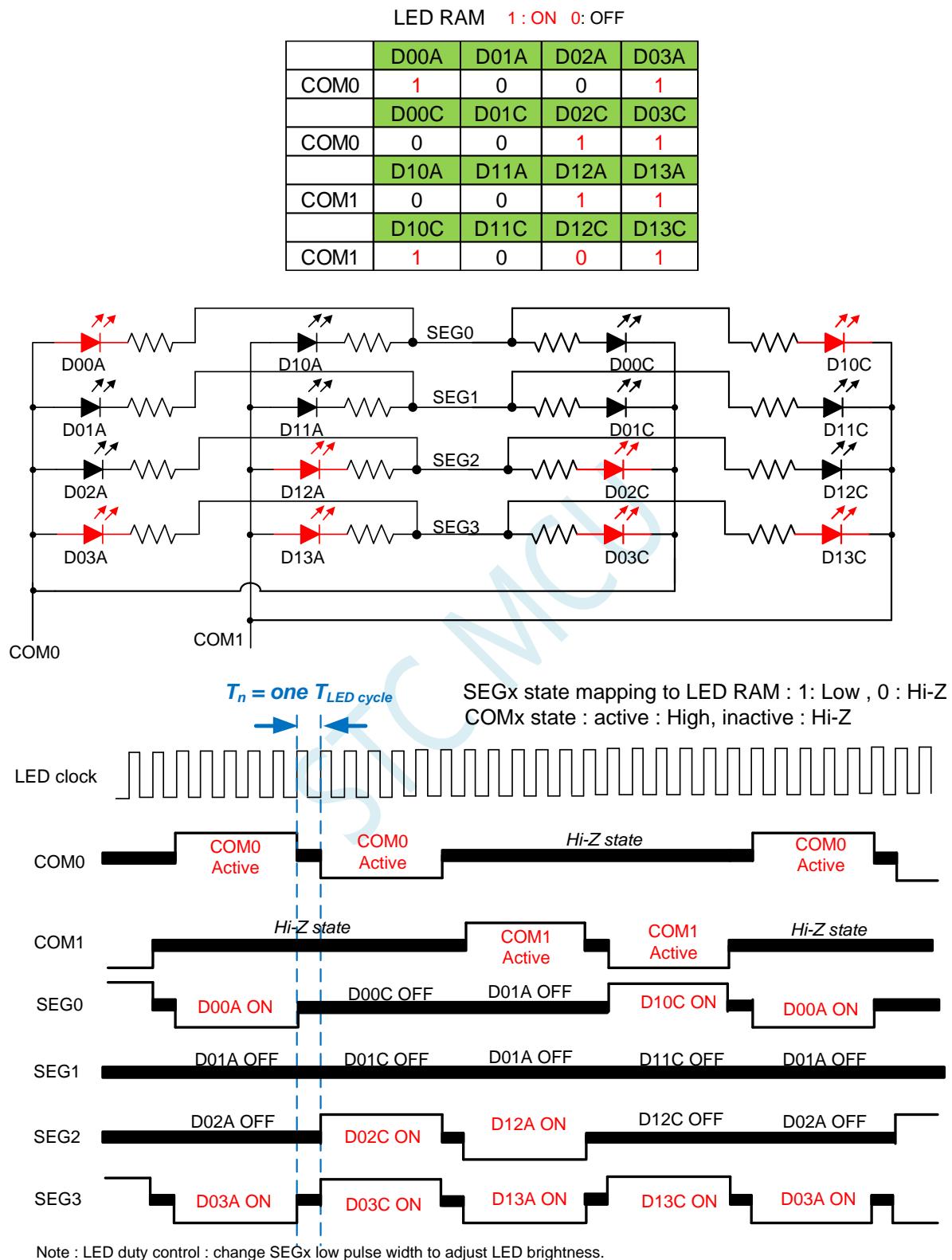
22.3 LED 共阴模式 (LEDMODE = 00)



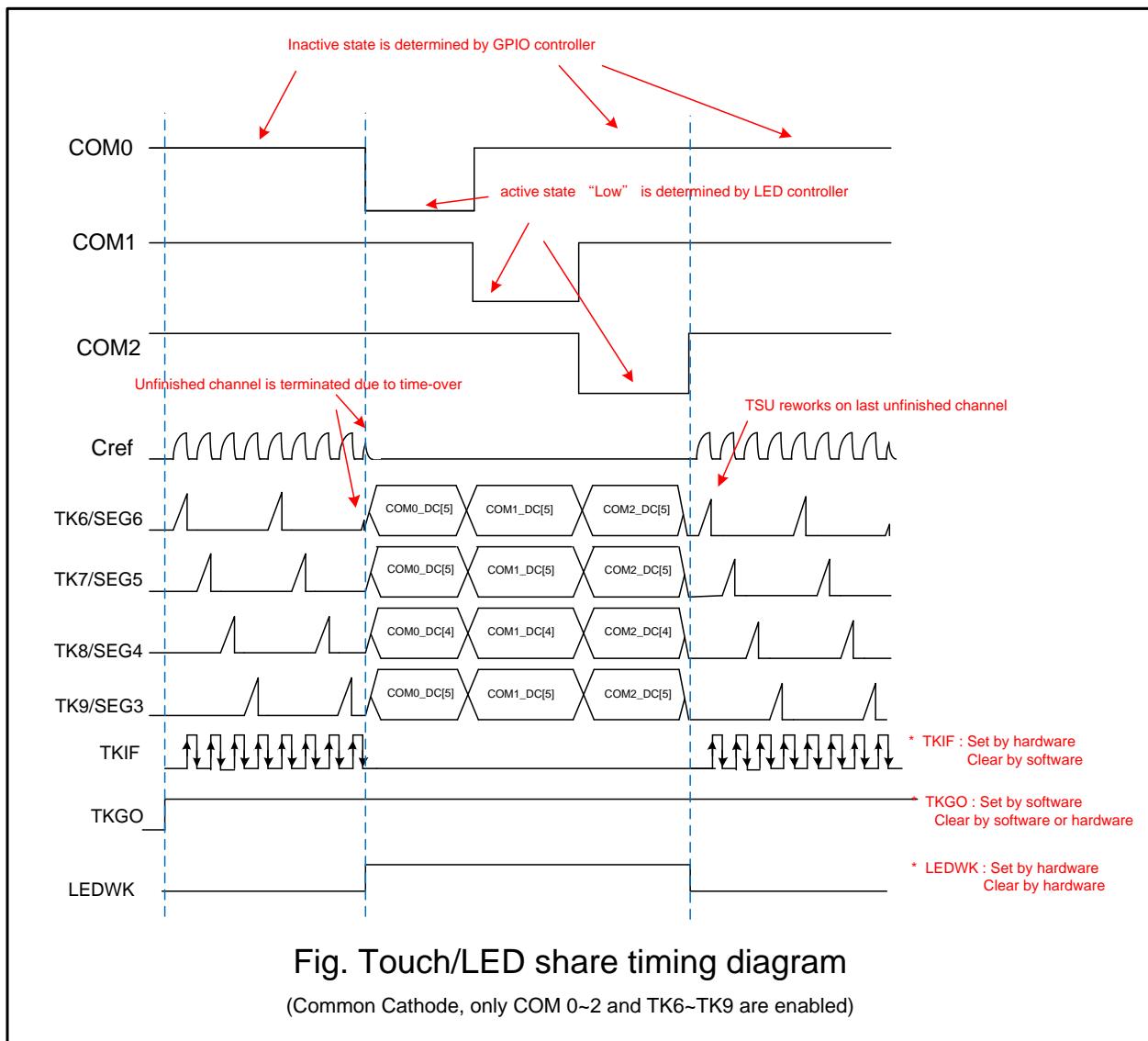
22.4 LED 共阳模式 (LEDMODE = 01)



22.5 LED 共阴/共阳模式 (LEDMODE = 10)



22.6 触摸按键与 LED 分时复用 I/O



操作步骤:

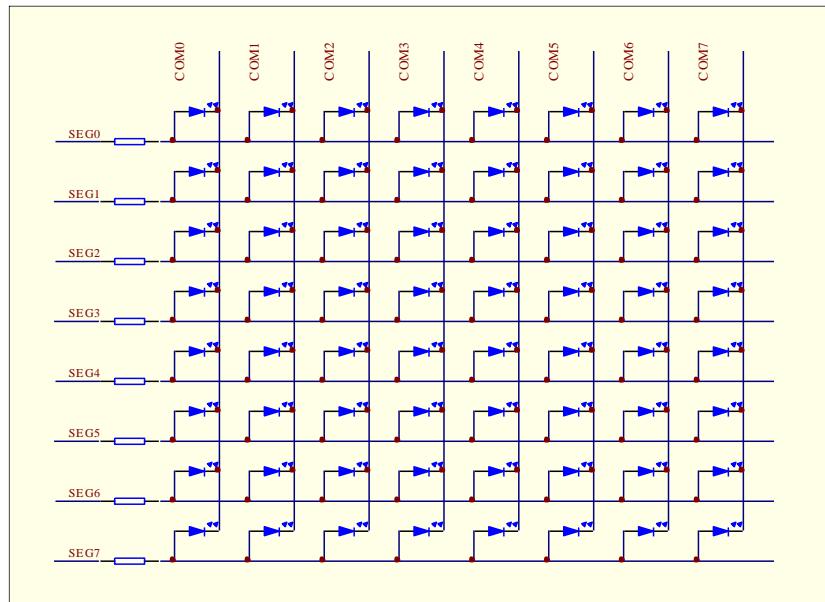
1. 选择需要扫描的触摸按键通道，寄存器为 TSCHEN1 及 TSCHEN2。
2. 配置开关频率 SCR[2:0]，放电时间 DT[2:0]及选择内部比较器参考电压 TSVR[1:0]。
3. 配置 TSSAMP[1:0]以决定同一信道的重复扫描次数，若 CPU 任务繁重，配置 TSWAIT 以利用 TSIF 状态来延时下一个信道的扫描。
4. 必要的话，配置 TSDCEN 以使能内部数字比较功能。
5. 设置 TSRT 内容，当 TSRT 内容不为 0x00 代表没有将开启 LED 驱动分时多任务功能。
6. 配置 SEGEN、COMEN 寄存器。
7. 配置 LEDCKS 以决定每一个 COM 作用的时间长度，这需要与 TKRT 寄存器一起考虑以计算出 LED 的刷新率。
8. 根据 LED 所需的工作模式及辉度配置 LEDMODE[1:0]及 LEDDUTY。
9. 写入数据至 LED 的数据寄存器 COMx_DC 及 COM_DA 中。
10. 设置 TSGO=1，触摸按键开始扫描。

- a) 软件可读取 TSWKCHN[3:0]来了解当下那一个信道正被扫描, 每扫完一个通道, 硬件会将 TSIF 设置成 1, 同时完成的信道号码会写入 TSDNCHN[3:0]中, 若有溢位发生, TSOV 也会被设置成 1, 软件应读取这些寄存器以决定接下来怎么做。TSIF 及 TSOV 仅能由硬件设置及软件清除。
 - b) 当切换成 LED 工作时间时, 软件读取 LEDWK 位会得到 1, 用以判断当下是触摸按键控制器正在工作或 LED 驱动电路正在工作
 - c) 软件依实际需要, 持续更新 LED 数据寄存器
11. 若软件想终止触摸按键与 LED 分时多任务模式, 需写入 TSGO=0, 则多任务模式终止, 触摸按键与 LED 皆不工作, I/O 口的控制权回归 GPIO 控制器

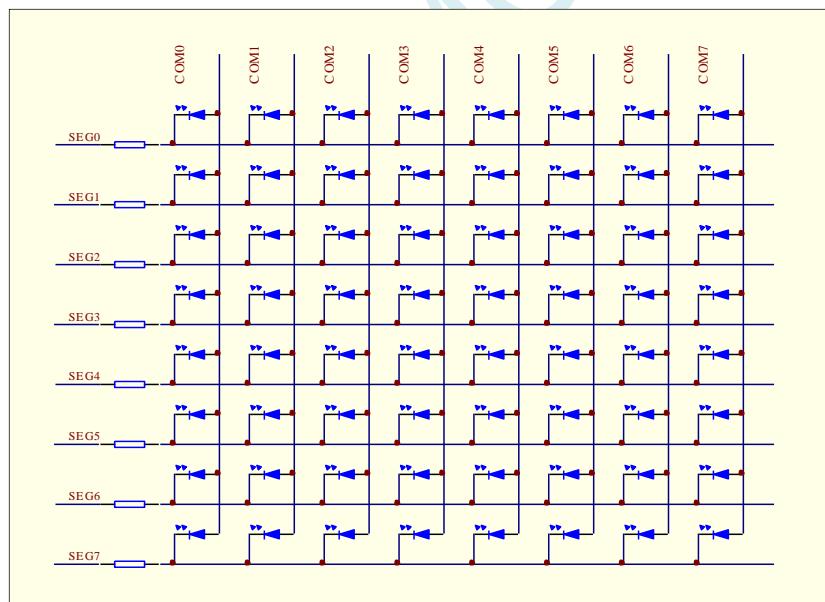
注意:

1. SINGLE 控制位在触摸按键与 LED 分时多任务模式下无效, 只有软件写入 TSGO 才能控制模块的开启和关闭
2. 由于触摸按键扫描时间终止并转换为 LED 作用时间时, 最后的那一个按键几乎都没完成, 这时硬件的处理是不会发出 TSIF 也不会更新触摸数据相关的寄存器, 但是硬件会记住该信道号码, 等 LED 作用期间终止并转换为触摸按键扫描时间时, 会由该信道重新扫描来启始新一轮扫描。
3. LED 与触摸复用的电路图如上,要注意的是 LED 的发光颜色不同,代表的 LED 等效电容大小会不同,电容愈大对触摸按键愈不利,零敏度会下降。一般而言,红光 LED 电容值可能为 35pF,但是黄光会高达 100pF, 这时如果要提高触摸按键的灵敏度,可以串一个 1N4148 二级管。1N4148 本身电容仅有 4pF,1N4148 串一个 100pF 的黄光 LED,对该键的寄生电容会比 4pF 略小

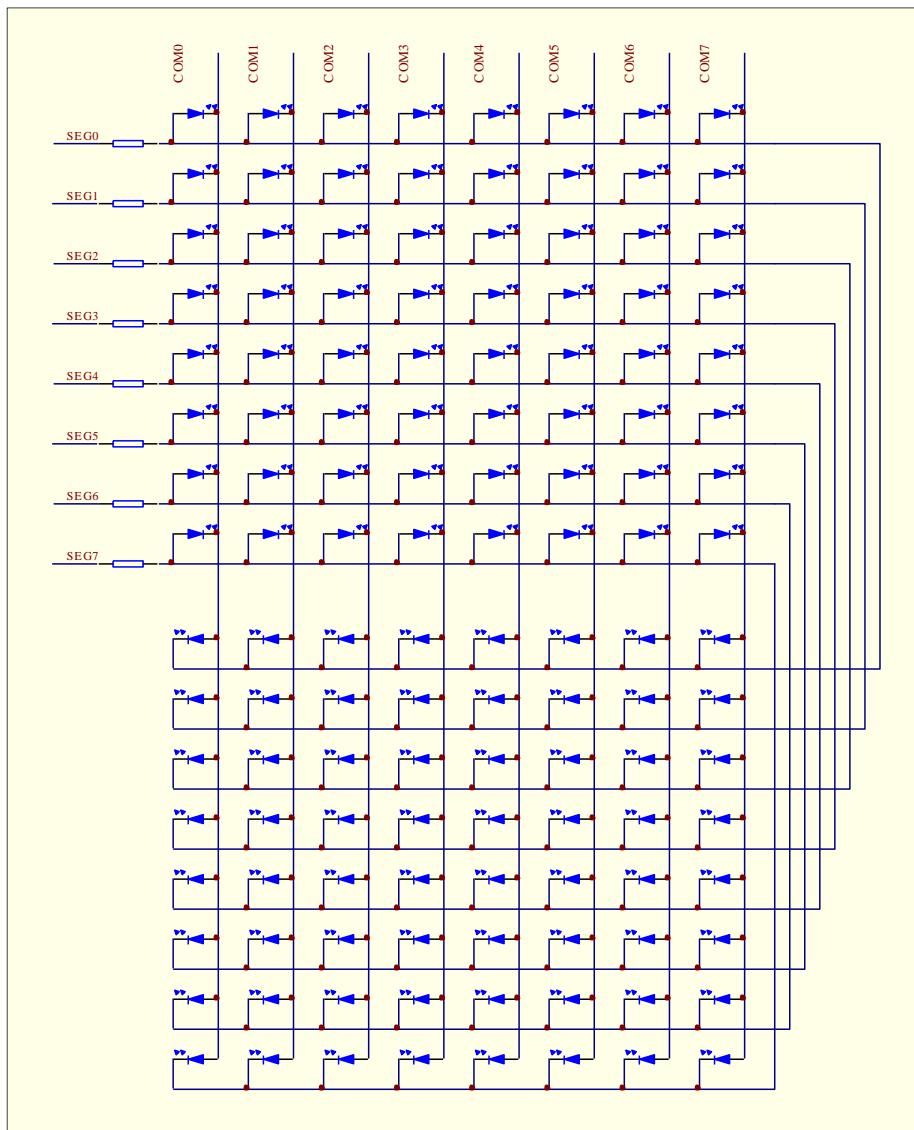
22.7 共阴模式参考电路图



22.8 共阳模式参考电路图

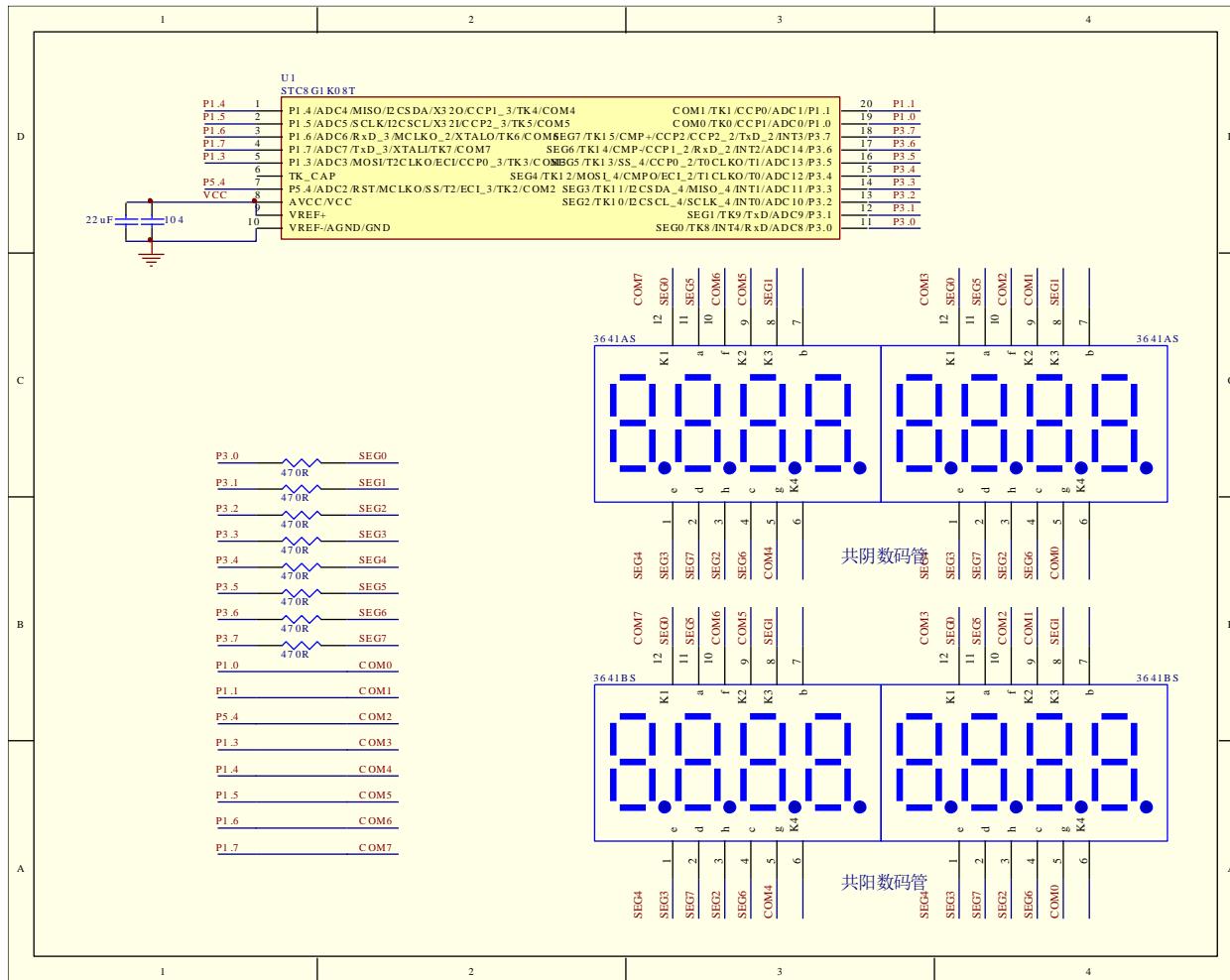


22.9 共阴/共阳模式参考电路图



22.10 范例程序

22.10.1 共阴/共阳模式驱动 16 个 7 段数码管



C 语言代码

//测试工作频率为 11.0592MHz

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

sfr    P0M1      = 0x93;
sfr    P0M0      = 0x94;
sfr    P1M1      = 0x91;
sfr    P1M0      = 0x92;
sfr    P2M1      = 0x95;
sfr    P2M0      = 0x96;
sfr    P3M1      = 0xb1;
sfr    P3M0      = 0xb2;
sfr    P4M1      = 0xb3;
sfr    P4M0      = 0xb4;
sfr    P5M1      = 0xc9;
sfr    P5M0      = 0xca;
```

```

#define COMEN      (*(unsigned char volatile xdata *)0xfb00)
#define SEGENL    (*(unsigned char volatile xdata *)0xfb01)
#define LEDCTR    (*(unsigned char volatile xdata *)0xfb03)
#define LEDCKS    (*(unsigned char volatile xdata *)0xfb04)
#define COM0_DA    (*(unsigned char volatile xdata *)0xfb10)
#define COM1_DA    (*(unsigned char volatile xdata *)0xfb11)
#define COM2_DA    (*(unsigned char volatile xdata *)0xfb12)
#define COM3_DA    (*(unsigned char volatile xdata *)0xfb13)
#define COM4_DA    (*(unsigned char volatile xdata *)0xfb14)
#define COM5_DA    (*(unsigned char volatile xdata *)0xfb15)
#define COM6_DA    (*(unsigned char volatile xdata *)0xfb16)
#define COM7_DA    (*(unsigned char volatile xdata *)0xfb17)
#define COM0_DC    (*(unsigned char volatile xdata *)0xfb20)
#define COM1_DC    (*(unsigned char volatile xdata *)0xfb21)
#define COM2_DC    (*(unsigned char volatile xdata *)0xfb22)
#define COM3_DC    (*(unsigned char volatile xdata *)0xfb23)
#define COM4_DC    (*(unsigned char volatile xdata *)0xfb24)
#define COM5_DC    (*(unsigned char volatile xdata *)0xfb25)
#define COM6_DC    (*(unsigned char volatile xdata *)0xfb26)
#define COM7_DC    (*(unsigned char volatile xdata *)0xfb27)

char code PATTERN[16] =
{
    0x3f,          //0
    0x06,          //1
    0x5b,          //2
    0x4f,          //3
    0x66,          //4
    0x6d,          //5
    0x7d,          //6
    0x27,          //7
    0x7f,          //8
    0x6f,          //9
    0x77,          //A
    0x7c,          //b
    0x39,          //C
    0x5E,          //d
    0x79,          //E
    0x71,          //F
};

void main()
{
    P1M0 = 0xff;
    P1M1 = 0x00;
    P3M0 = 0xff;
    P3M1 = 0x00;
    P5M0 = 0x10;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    COMEN = 0xff;           //使能 COM0~COM7
    SEGENL = 0xff;          //使能 SEG0~SEG7
    LEDCTRL = 0x20;         //LED 驱动共阴/共阳模式
    LEDCKS = 7;             //设置 LED 刷新频率

    COM0_DA = PATTERN[0];   //设置 LED 显示内容
}

```

```
COM1_DA = PATTERN[1];
COM2_DA = PATTERN[2];
COM3_DA = PATTERN[3];
COM4_DA = PATTERN[4];
COM5_DA = PATTERN[5];
COM6_DA = PATTERN[6];
COM7_DA = PATTERN[7];

COM0_DC = PATTERN[8];
COM1_DC = PATTERN[9];
COM2_DC = PATTERN[10];
COM3_DC = PATTERN[11];
COM4_DC = PATTERN[12];
COM5_DC = PATTERN[13];
COM6_DC = PATTERN[14];
COM7_DC = PATTERN[15];

LEDCTRL |= 0x80;                                //启动 LED 驱动

P_SW2 &= ~0x80;

while (1);
}
```

23 增强型双数据指针

STC8G 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

23.1 相关的特殊功能寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
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

23.1.1 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

23.1.2 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

23.1.3 数据指针控制寄存器 (DPS)

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
```

23.1.4 数据指针控制寄存器 (TA)

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)

STCMCU

23.2 范例程序

23.2.1 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

; 测试工作频率为 11.0592MHz

```

P0M1      DATA      093H
P0M0      DATA      094H
P1M1      DATA      091H
P1M0      DATA      092H
P2M1      DATA      095H
P2M0      DATA      096H
P3M1      DATA      0B1H
P3M0      DATA      0B2H
P4M1      DATA      0B3H
P4M0      DATA      0B4H
P5M1      DATA      0C9H
P5M0      DATA      0CAH

        ORG      0000H
        LJMP     MAIN

        ORG      0100H
MAIN:
        MOV      SP, #5FH
        MOV      P0M0, #00H
        MOV      P0M1, #00H
        MOV      P1M0, #00H
        MOV      P1M1, #00H
        MOV      P2M0, #00H
        MOV      P2M1, #00H
        MOV      P3M0, #00H
        MOV      P3M1, #00H
        MOV      P4M0, #00H
        MOV      P4M1, #00H
        MOV      P5M0, #00H
        MOV      P5M1, #00H

        MOV      DPS,#00100000B      ;使能 TSL,并选择 DPTR0
        MOV      DPTR,#1000H          ;将 1000H 写入 DPTR0 后选择 DPTR1 为 DPTR
        MOV      DPTR,#0103H          ;将 0103H 写入 DPTR1 中
        MOV      DPS,#1011000B        ;设置 DPTR1 为递减模式,DPTR0 为递加模式,使能 TSL
                                    ;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
                                    ;完成后 DPTR0 自动加 1 并将 DPTR1 设置为 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DPTR
SJMP      $               ;
END

```

23.2.2 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

;测试工作频率为 11.0592MHz

```

P0M1      DATA      093H
P0M0      DATA      094H
P1M1      DATA      091H
P1M0      DATA      092H
P2M1      DATA      095H
P2M0      DATA      096H
P3M1      DATA      0B1H
P3M0      DATA      0B2H
P4M1      DATA      0B3H
P4M0      DATA      0B4H
P5M1      DATA      0C9H
P5M0      DATA      0CAH

ORG       0000H
LJMP     MAIN

ORG       0100H
MAIN:
MOV      SP, #5FH
MOV      P0M0, #00H
MOV      P0M1, #00H
MOV      P1M0, #00H
MOV      P1M1, #00H
MOV      P2M0, #00H
MOV      P2M1, #00H
MOV      P3M0, #00H
MOV      P3M1, #00H
MOV      P4M0, #00H
MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

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 读取数据后 DPTR0 自动加 1

```

MOV *P0,A* ;数据输出到P0 口
SJMP *\$*
END

STCMCU

24 MDU16 硬件 16 位乘除法器

产品线	MDU16
STC8G1K08 系列	
STC8G1K08-8Pin 系列	●
STC8G1K08A 系列	●
STC8G2K64S4 系列	●
STC8G2K64S2 系列	●
STC8G1K08T 系列	
STC15H2K64S4 系列	●

STC8G 系列部分型号的单片机内部集成了 MDU16/16 位硬件乘除法器。

支持如下数据运算:

- 数据规格化 (需要 3~20 个时钟的运算时间)
- 逻辑左移 (需要 3~18 个时钟的运算时间)
- 逻辑右移 (需要 3~18 个时钟的运算时间)
- 16 位乘以 16 位 (需要 10 个时钟的运算时间)
- 16 位除以 16 位 (需要 9 个时钟的运算时间)
- 32 位除以 16 位 (需要 17 个时钟的运算时间)

所有的操作都是基于无符号整形数据类型。

24.1 相关的特殊功能寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
MD3	MDU 数据寄存器	FCF0H	MD3[7:0]								0000,0000
MD2	MDU 数据寄存器	FCF1H	MD2[7:0]								0000,0000
MD1	MDU 数据寄存器	FCF2H	MD1[7:0]								0000,0000
MD0	MDU 数据寄存器	FCF3H	MD0[7:0]								0000,0000
MD5	MDU 数据寄存器	FCF4H	MD5[7:0]								0000,0000
MD4	MDU 数据寄存器	FCF5H	MD4[7:0]								0000,0000
ARCON	MDU 模式控制寄存器	FCF6H	MODE[2:0]			SC[4:0]					0000,0000
OPCON	MDU 操作控制寄存器	FCF7H	-	MDOV	-	-	-	-	RST	ENOP	x0xx,xx00

24.1.1 操作数 1 数据寄存器 (MD0~MD3)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
MD3	FCF0H								MD3[7:0]
MD2	FCF1H								MD2[7:0]
MD1	FCF2H								MD1[7:0]
MD0	FCF3H								MD0[7:0]

24.1.2 操作数 2 数据寄存器 (MD4~MD5)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
MD5	FCF4H								MD5[7:0]
MD4	FCF5H								MD4[7:0]

32位除以16位除法:

被除数: {MD3,MD2,MD1,MD0}

除数: {MD5,MD4}

商: {MD3,MD2,MD1,MD0}

余数: {MD5,MD4}

16位除以16位除法:

被除数: {MD1,MD0}

除数: {MD5,MD4}

商: {MD1,MD0}

余数: {MD5,MD4}

16位乘以16位乘法:

被乘数: {MD1,MD0}

乘数: {MD5,MD4}

积: {MD3,MD2,MD1,MD0}

32 位逻辑左移/逻辑右移

操作数: {MD3,MD2,MD1,MD0}

32 位数据规格化:

操作数: {MD3,MD2,MD1,MD0}

24.1.3 MDU 模式控制寄存器 (ARCON), 运算所需时钟数

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ARCON	FCF6H			MODE[2:0]				SC[4:0]	

MODE[2:0]: MDU模式选择

MODE[2:0]	模式	时钟数	操作说明
1	逻辑右移	3~18	将{MD3,MD2,MD1,MD0}中的数据右移SC[4:0]位, MD3的高位补0
2	逻辑左移	3~18	将{MD3,MD2,MD1,MD0}中的数据左移SC[4:0]位, MD0的低位补0
3	数据规格化	3~20	对{MD3,MD2,MD1,MD0}中的数据进行逻辑左移, 将数据 高位的0全部移出, 使MD3的最高位为1, 逻辑左移的位数 被记录在SC[4:0]中
4	16位×16位	10	{MD1,MD0} × {MD5,MD4} = {MD3,MD2,MD1,MD0}
5	16位÷16位	9	{MD1,MD0} ÷ {MD5,MD4} = {MD1,MD0}…{MD5,MD4}
6	32位÷16位	17	{MD3,MD2,MD1,MD0} ÷ {MD5,MD4} = {MD3,MD2,MD1,MD0}…{MD5,MD4}

其他	无效		
----	----	--	--

SC[4:0]: 数据移动位数

当 MDU 为移动模式时, SC 用于设置左移/右移的位数

当 MDU 为数据规格化模式时, SC 为数据规格化后数据所移动的实际位数

24.1.4 MDU 操作控制寄存器 (OPCON)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
OPCON	FCF7H	-	MDOV	-	-	-	-	RST	ENOP

MDOV: MDU溢出标志位 (只读标志位)

在如下几种情况时, MDOV 会被硬件自动置 1:

- 1、除数为 0 时;
- 2、乘法的积大于 0FFFFH 时;

当软件写 OPCON.0 (EN) 或者写 ARCON 时, 硬件会自动清除 MDOV

RST: 软件复位 MDU 乘除单元。写 1 触发软件复位, MDU 复位完成后硬件自动清零。

注: 软件复位 MDU 乘除单元时, ARCON 寄存器的值会被清除。

ENOP: MDU 模块使能。写 1 触发 MDU 模块开始计算, 当 MDU 计算完成后, 硬件自动将 ENOP 清零。

软件可以在对 ENOP 置 1 后, 循环的查询 ENOP, 当 ENOP 由 1 变 0 则表示计算完成。

24.2 关于 MDU16 的网友应用杂谈（提供思路，仅供参考）

网友 1：“数据规格化用下面的一个简单例子说明”

1、有一个 7 位小数精度的数据: 0.0000123, 由于数据位宽有限, 如果需要有效利用位宽, 就需要把前面的数据左移, 比如左移后数据为 0.123e-4, 其中指数-4 保存在另一个寄存器, 记录左移的次数就是记录指数的大小。原来寄存器数据转换为 0.123。这样就把数据右边的位宽腾出来, 可以保证后续计算的精度。上面只是用十进制简单的说明规格化原理, 二进制原理也是一样的。其中浮点和定点(整数)转换就必须使用规格化的原理, 如果两个浮点数相加减时的指数不一样, 也需要进行规格化处理(这个过程叫作对阶)。如果两个浮点数的指数相差非常大, 相加减时就会出现大数吃小数的问题。比如: $0.123e+4 - 0.12e-4 = 0.123e+4 - 0.000000012e+4 = 0.123e+4$ 。结果就是被减数, 这是因为在减操作前, 两个浮点数的指数需要完全一致(对阶), 需要把指数小的浮点数进行移位, 使指数变为+4.但是数据宽度是有限的 7 位小数精度, $0.000000012e+4$ 这个数右边的数据会被截断变为 $0.0000000e+4 = 0$.

网友 2：“关于 STC8G 的 MDU 功能, 我分享一点自己的体会, 有不对的请大家批评指教, 共同提高。”

- 1、功能 1 和 2 对于缩减和扩展整数数据很有效。首先在进行双操作数运算时, 如果两个数的长度不一样, 需要转换为相同的长度进行才进行运算。比如 32 位整数乘 8 位整数, 就要将 8 位转换为 32 位。其次对 AD 采样的结果, 转换为指定的位数精度时也需要位移。最后, 比如对网络通讯, 需要提取数据的某几位出来进行命令解析或者数据分解合成, 位移都是很重要的。由于 8051 只有移动 1 位的指令, 多位移动需要借助额外的循环代码, 需要很多个指令周期, 因此使用 MDU 将比 51 汇编指令快数倍。
- 2、功能 3 是整数转换为浮点数必须的功能。对于满精度的 32 位整数, 实现这个功能一般要超过 100 个指令周期, 因此 MDU 对转速度的提升是比较大的。由于像 AD 设备输出、像各种三轴加速度输出, 一般都是整数的(比如 16 位的), 要进行实数运算, 要进行三角函数运算, 整数的输出必须要转换为浮点数, 而且每次采集数据都要进行这数据类型转换, 需要转换的次数就很多了。对于高速数据采集和像无人机控制这样的应用, 如果采用 DMU 对整体性能的提高就很可观了。
- 3、功能 6 是定点实数运算必须的除法功能, 功能 4 是功能 6 对应的 16 位 $\times 16$ 位结果为 32 位的乘法运算。功能 6 的最常见应用是数据处理中的标度转换, 比如对于将参考电压为 5 伏的 10 位 AD 采集的整数转换 3 位数码管的 2 位固定小数点进行显示的运算公式为: $N32=ADN*500/1023$ 。这时只要(1) 将 AD 采样值 AND 送 MX (DM1MD0), (2) 送 500 到 NX (MD5MD4), (3) 执行功能 4, 结果是 32 位的了, (4) 送 1023 到 NX (MD5MD4), (3) 执行功能 6, 16 位的结果就在 MX 中了, 取回来就行了。另一个常见的应用就是在 TFT 之类的点阵屏上画点和线, 比如数字示波器, 这些都需要进行坐标变换的乘除法-先乘为 32 位整数, 再除以 16 位整数得到 16 位结果。
- 4、功能 4 和功能 6 的组合是实现离散卷积的硬件基础。如果不采用浮点加速硬件, 实现浮点数的四则运算比实现整数的四则运算要慢一个量级, 因此前辈们发明了用整数变量来实现卷积的方法。首先比如我们常见的将 JPG 图像数据转换为 RGB 图像数据或者相反, 就需要进行傅里叶变换, 由于图像数据的长度是固定的(8 位或者 16 位), 因此就可以用离散傅里叶变换来实现, 其中基本只用到 8 位或者 16 位的整数乘法和极少量的 32 位乘除法。这样, 我们早期的数码相机才有可能实现。其次 PS 图像处理中常见的各种模板处理, 也使用的是二维矩阵卷积方法, 也是需要巨量的对整数的(8 位图像视图像大小需要 16 位和 32 位的中间计算结果)乘加计算, 使用离散卷积将极大的提高运算速度。因此有 MDU 的 STC8 单片机不仅可以用于实时采集和显示图像, 也可以实时处理图像。最后人工智能也涉及大量的矢量和矩阵运算, 比如神经网络卷积, 这些都可

以用功能 4 和功能 6 的组合实现, MDU 应该可以在小型智能化场景中得到应用。只是要实现这些功能, 需要 STC8 的增强型双数据指针的配合, 需要专门的知识结构, 专门编制出函数库来提供给用户使用, 才能发挥 STC8 的 MDU 巨大优势。

STCMCU

24.3 范例程序

C 语言代码

```
//测试工作频率为 11.0592MHz

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

#define MD3U32      (*(unsigned long volatile xdata *)0xfcfc0)
#define MD3U16      (*(unsigned int volatile xdata *)0xfcfc0)
#define MD1U16      (*(unsigned int volatile xdata *)0xfcfc2)
#define MD5U16      (*(unsigned int volatile xdata *)0xfcfc4)

#define MD3      (*(unsigned char volatile xdata *)0xfcfc0)
#define MD2      (*(unsigned char volatile xdata *)0xfcfc1)
#define MD1      (*(unsigned char volatile xdata *)0xfcfc2)
#define MD0      (*(unsigned char volatile xdata *)0xfcfc3)
#define MD5      (*(unsigned char volatile xdata *)0xfcfc4)
#define MD4      (*(unsigned char volatile xdata *)0xfcfc5)
#define ARCON      (*(unsigned char volatile xdata *)0xfcfc6)
#define OPCON      (*(unsigned char volatile xdata *)0xfcfc7)

sfr P_SW2 = 0xBA;

///////////////////////////////
//16 位乘 16 位
/////////////////////////////
unsigned long res;
unsigned int dat1, dat2;

P_SW2 |= 0x80;                                //访问扩展寄存器 xsfr
MD1U16 = dat1;                               //dat1 用户给定
MD5U16 = dat2;                               //dat2 用户给定
ARCON = 4 << 5;                            //16 位*16 位, 乘法模式
OPCON = 1;                                    //启动计算
while((OPCON & 1) != 0);                    //等待计算完成
res = MD3U32;                                //32 位结果

///////////////////////////////
//32 位除以 16 位
/////////////////////////////
unsigned long res;
unsigned long dat1;
unsigned int dat2;

P_SW2 |= 0x80;                                //访问扩展寄存器 xsfr
MD3U32 = dat1;                               //dat1 用户给定
MD5U16 = dat2;                               //dat2 用户给定
ARCON = 6 << 5;                            //32 位/16 位, 除法模式
OPCON = 1;                                    //启动计算
while((OPCON & 1) != 0);                    //等待计算完成
res = MD3U32;                                //32 位商, 16 位余数在 MD5U16 中
```

//左移或右移:

```
unsigned long res;
unsigned long dat1;
unsigned char num; //移位的位数, 用户给定

MD3U32 = dat1;
ARCON = (2 << 5) + num; //dat1 用户给定
//ARCON = (1 << 5) + num; //32 位左移模式
OPCON = 1; //32 位右移模式
while((OPCON & 1) != 0); //启动计算
res = MD3U32; //等待计算完成
//32 位结果
```

附录A 编译器（汇编器）/仿真器/头文件使用指南

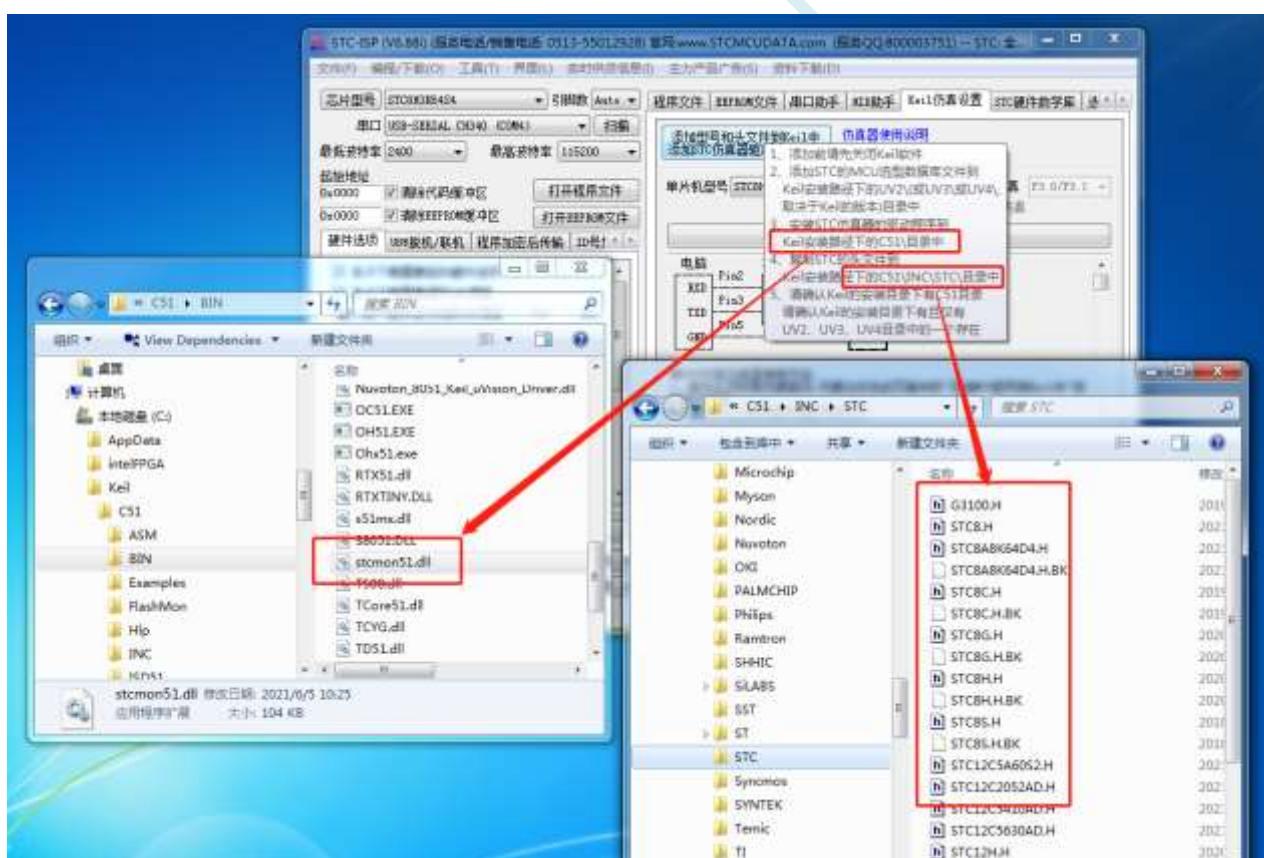
A: STC 单片机应使用何种编译器/汇编器？

Q: 任何老式的 8051 编译器/汇编器都可以支持，现流行使用 Keil C51

A: Keil 环境中，应如何包含头文件

Q: 按照下面图示的步骤安装完驱动和头文件后，新建项目时选择 STC 相应的单片机型号，在源文件中直接使用 “#include <stc8g.h>” 即可完成头文件的包含。如果新建项目时选择的 Intel 的 8052/87C52/87C54/87C58 或 Philips 的 P87C52/P87C54/P87C58 编译，头文件包含<reg51.h>即可，不过 STC 新增的特殊功能寄存器则需要用户自己声明。

1、安装 Keil 版本的仿真驱动

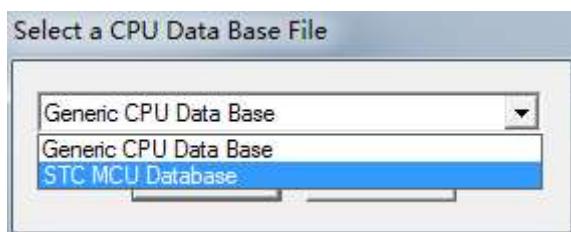


如上图，首先选择“Keil 仿真设置”页面，点击“添加 MCU 型号到 Keil 中”，在出现的如下的目录选择窗口中，定位到 Keil 的安装目录(一般可能为“C:\Keil\”)，“确定”后出现下图中右边所示的提示信息，表示安装成功。添加头文件的同时也会安装 STC 的 Monitor51 仿真驱动 STCMON51.DLL，驱动与头文件的安装目录如上图所示。

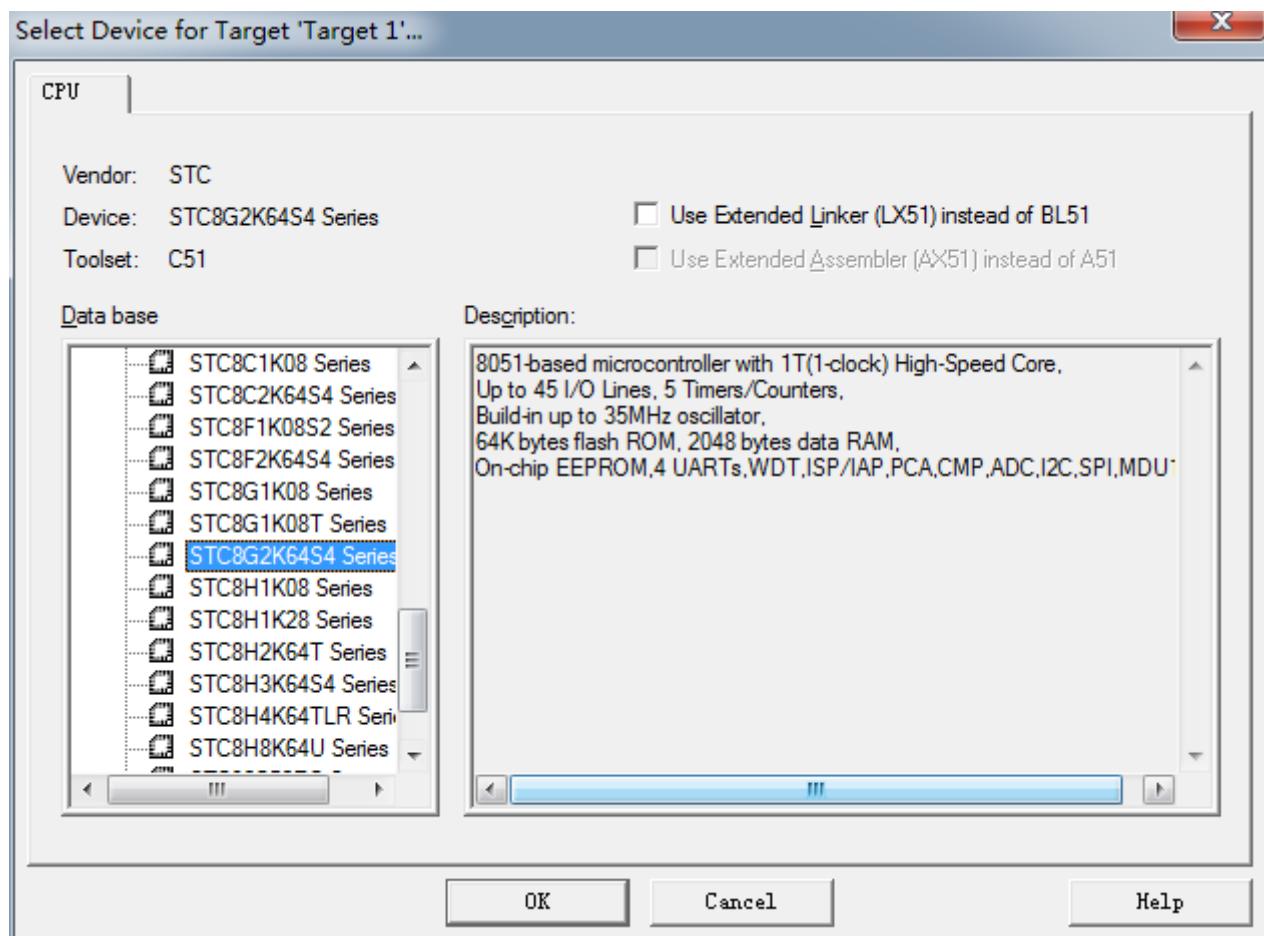


2、在 Keil 中创建项目

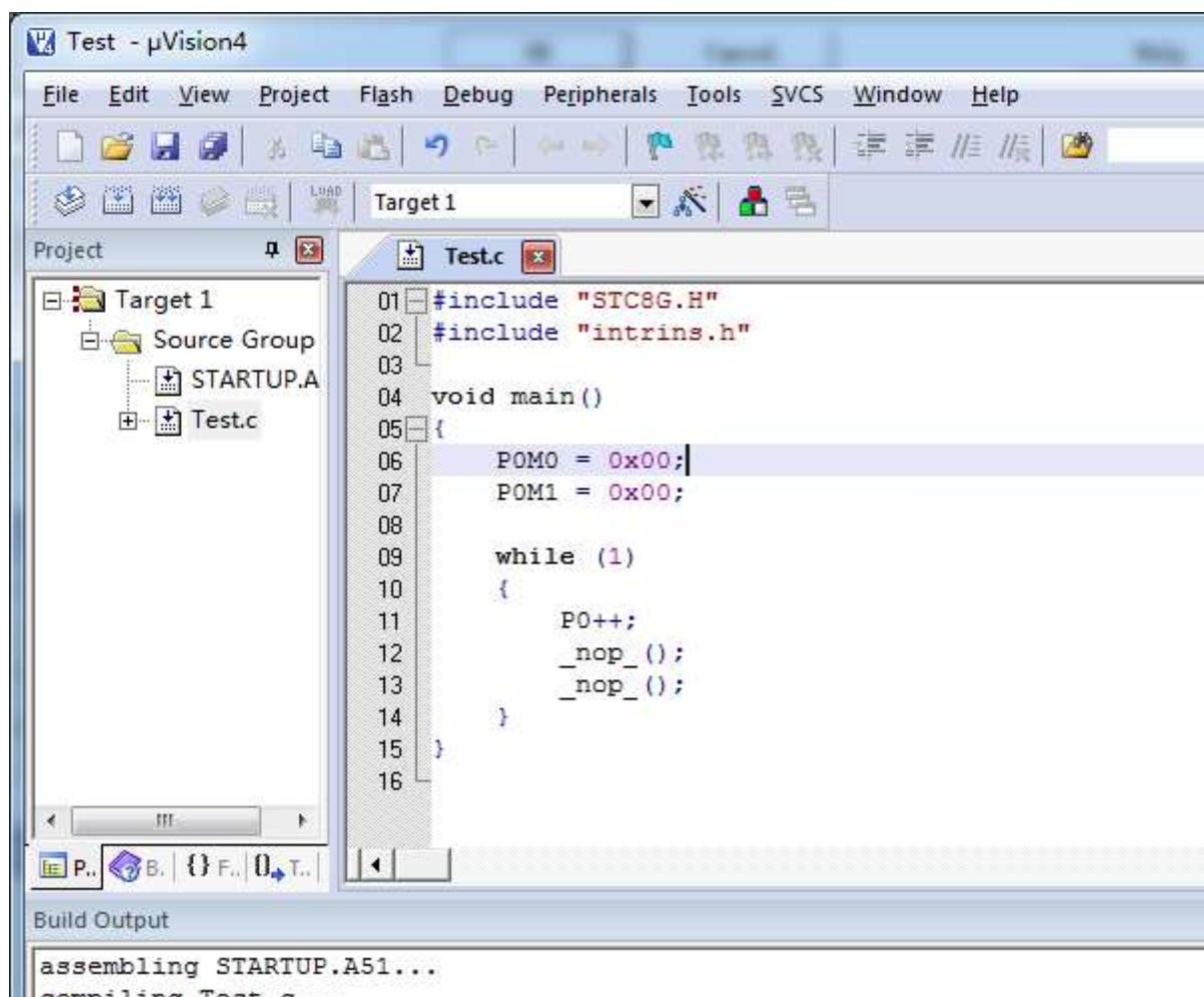
若第一步的驱动安装成功，则在 Keil 中新建项目时选择芯片型号时，便会有“STC MCU Database”的选择项，如下图



然后从列表中选择响应的 MCU 型号，我们在此选择“STC8G2K64S4”的型号，点击“确定”完成选择



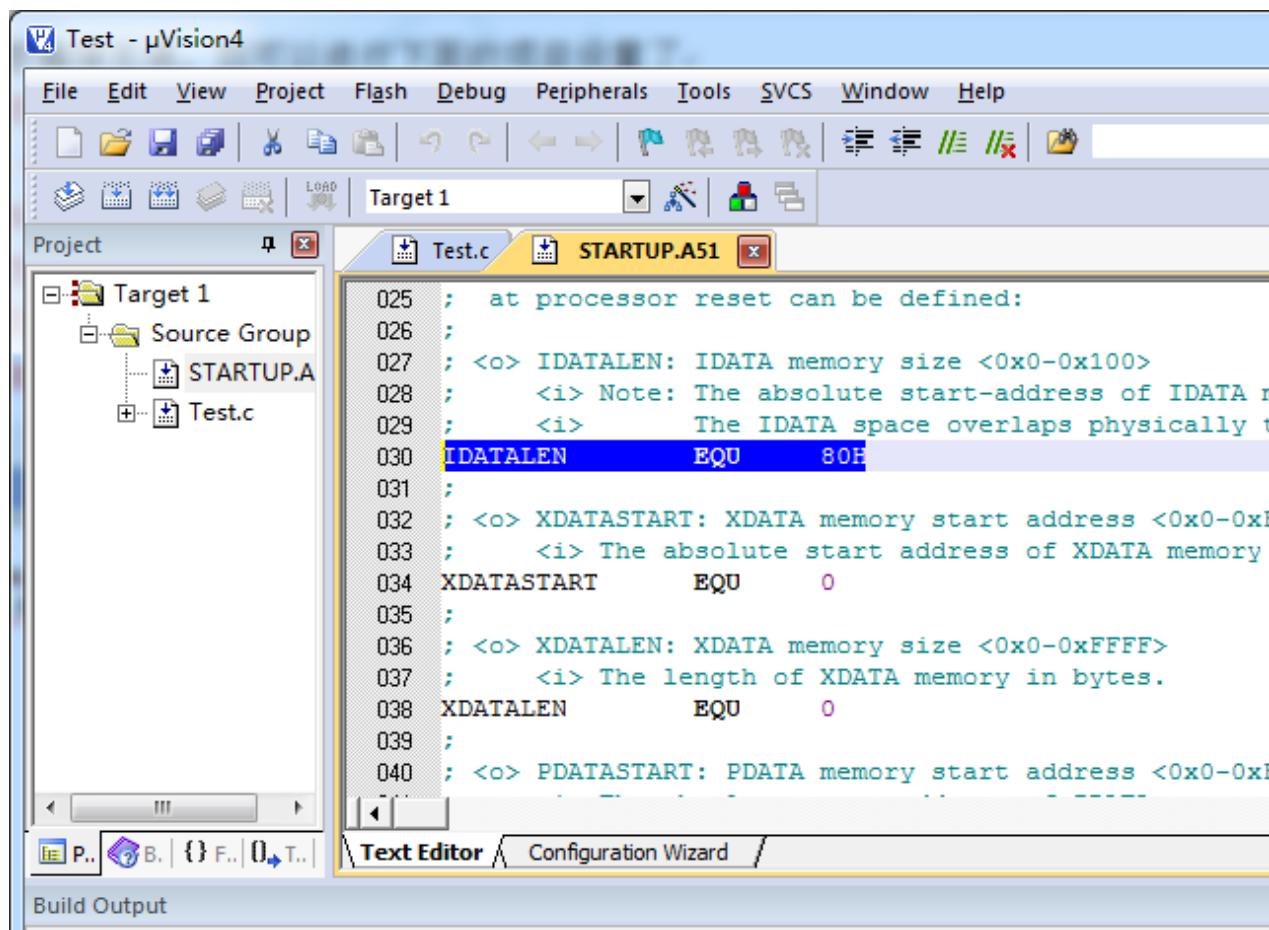
添加源代码文件到项目中，如下图：



保存项目，若编译无误，则可以进行下面的项目设置了

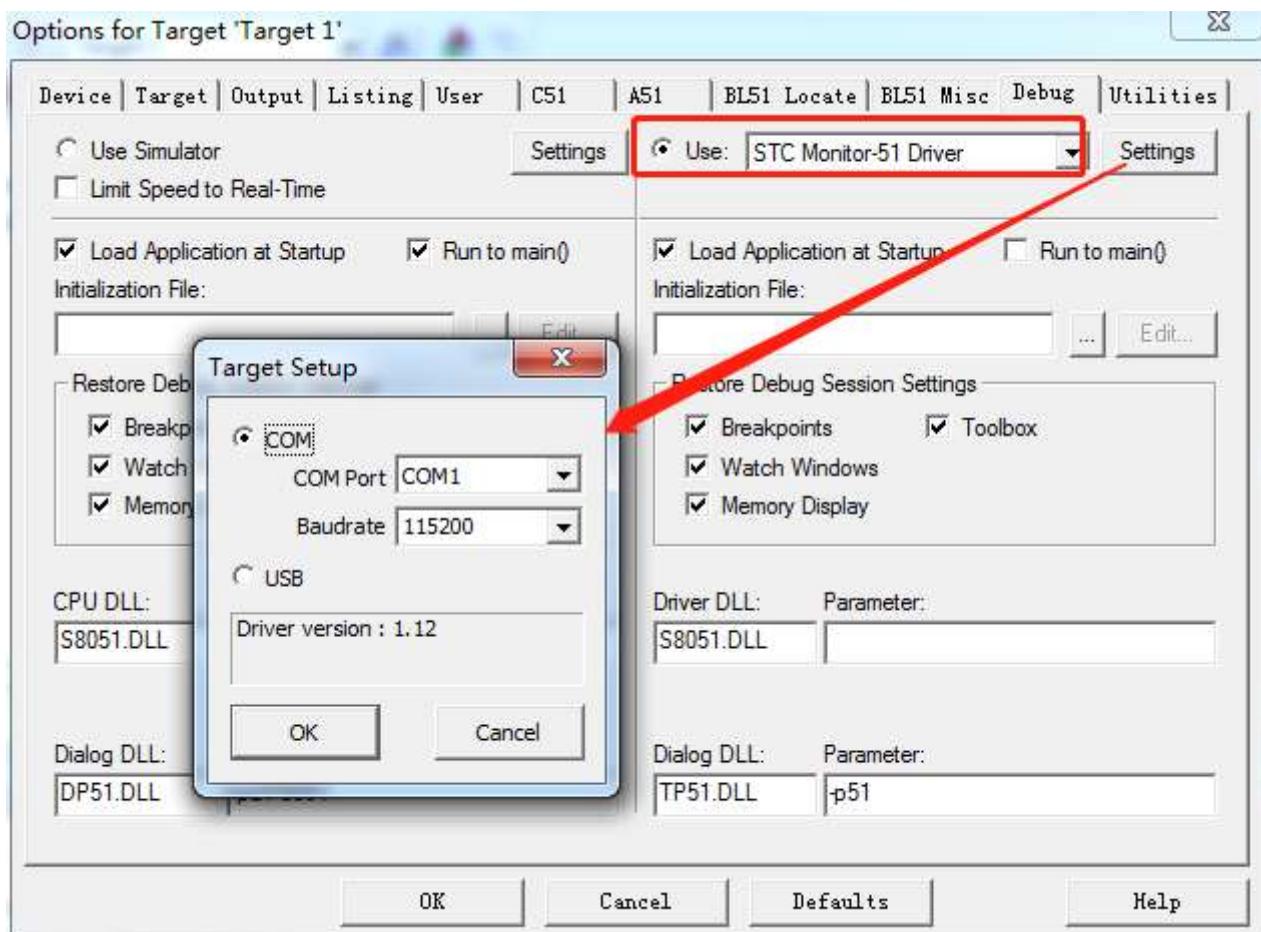
附加说明一点：

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 OFFH，则会将 IDATA 的 00-FF 的 RAM 初始化为 0。



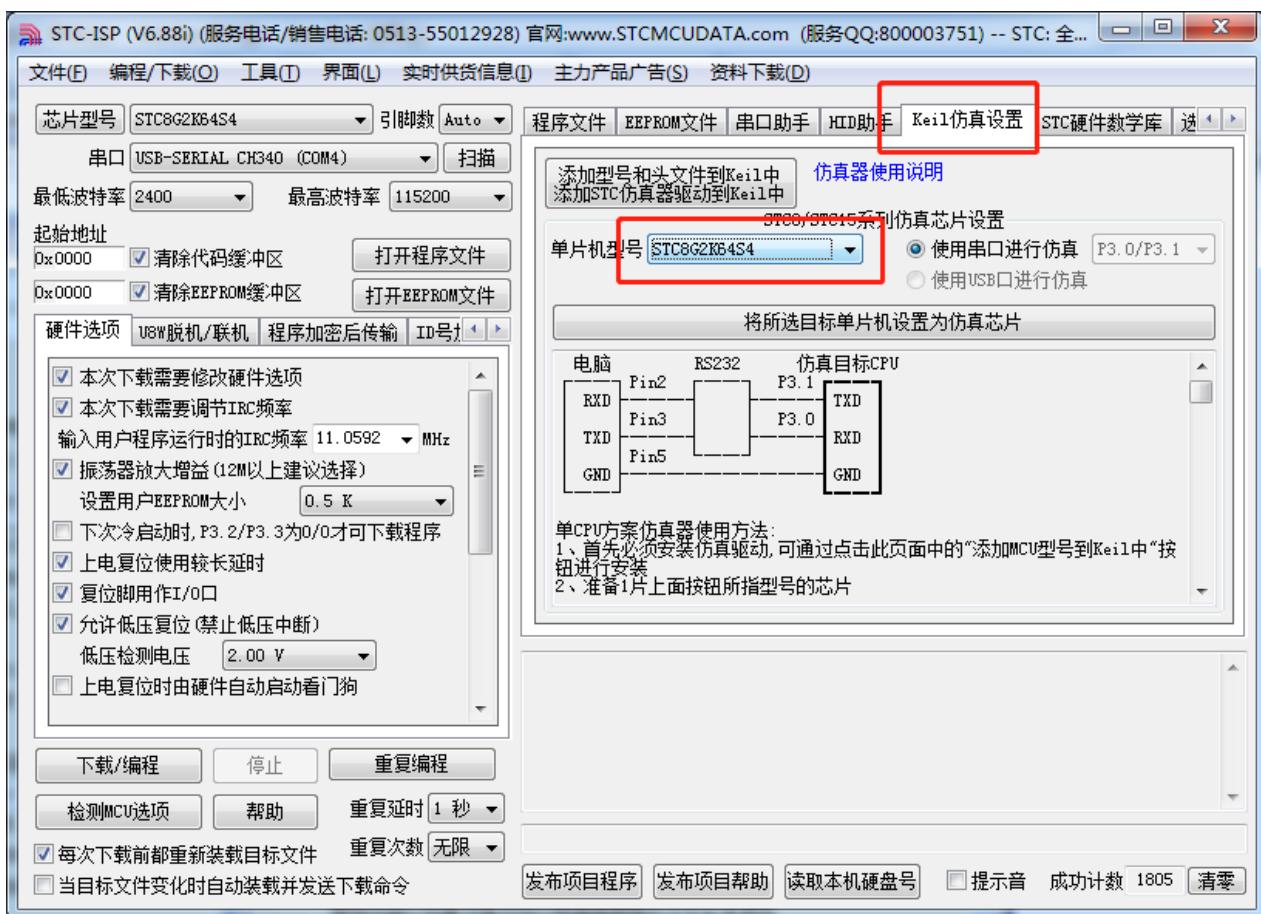
虽然 STC8 系列的单片机的 IDATA 大小为 256 字节（00-7F 的 DATA 和 80H-FFH 的 IDATA），但由于在 RAM 的最后 17 个字节有写入 ID 号以及相关的测试参数，若用户在程序中需要使用这一部分数据，则一定不要将 IDATALEN 定义为 256。

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。

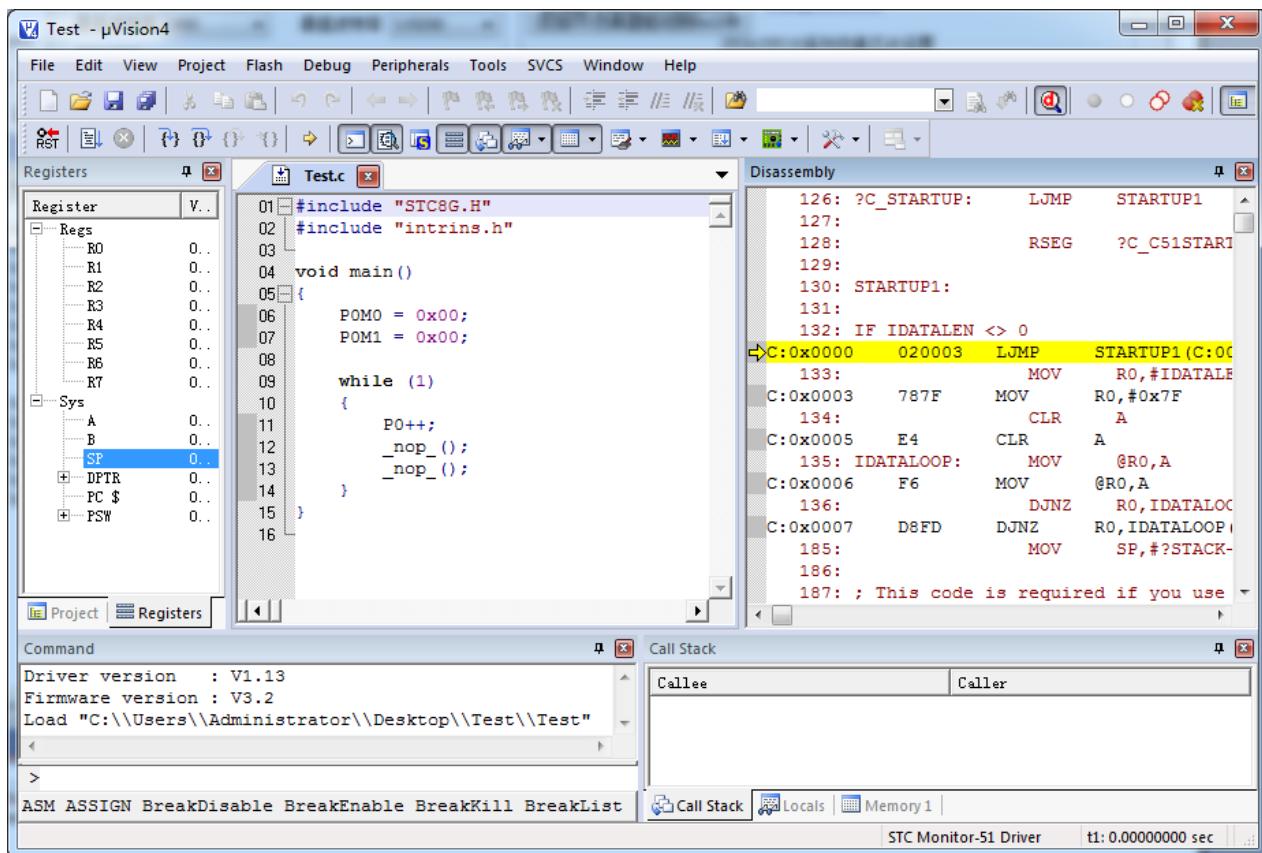
5、开始仿真

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。

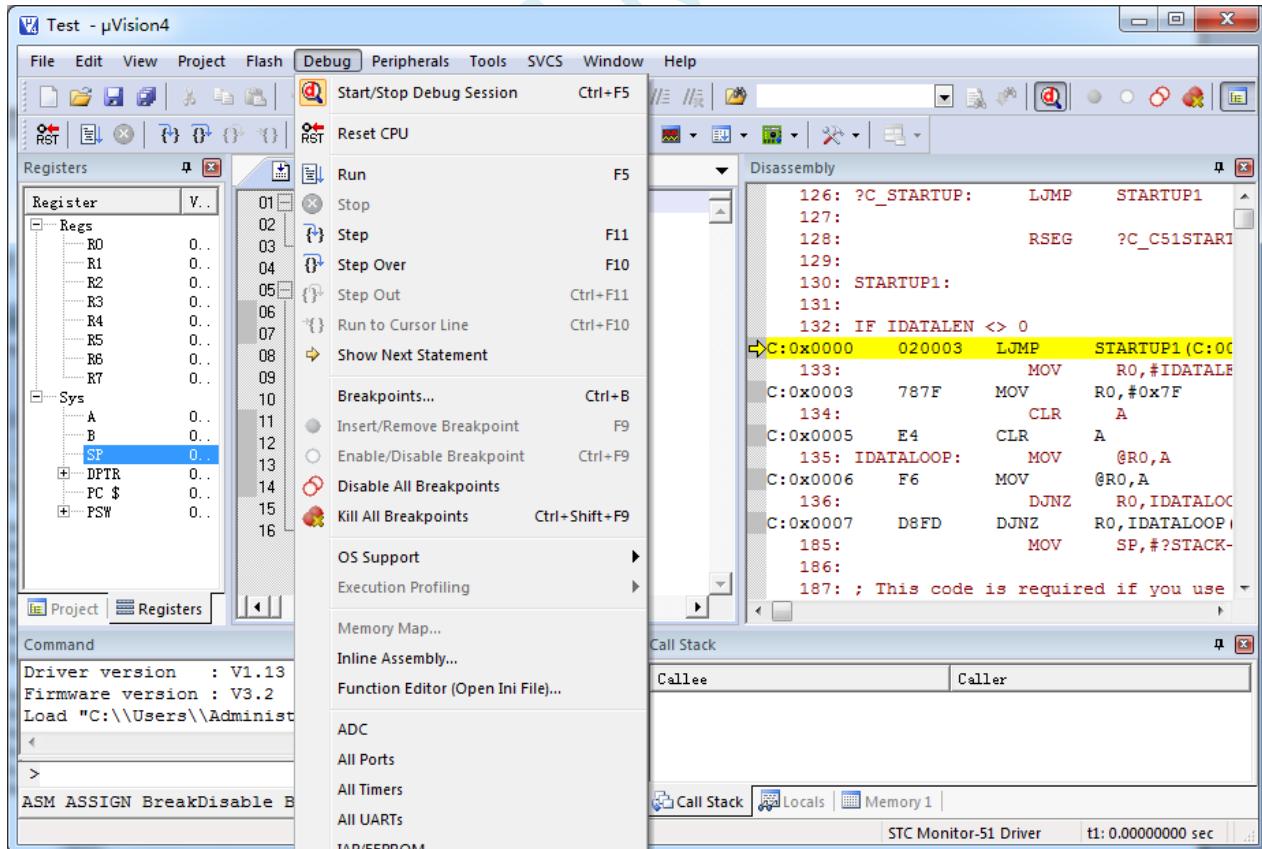
若硬件连接无误的话，将会进入到类似于下面的调试界面，并在命令输出窗口显示当前的仿真驱动版本号和当前仿真监控代码固件的版本号

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）。



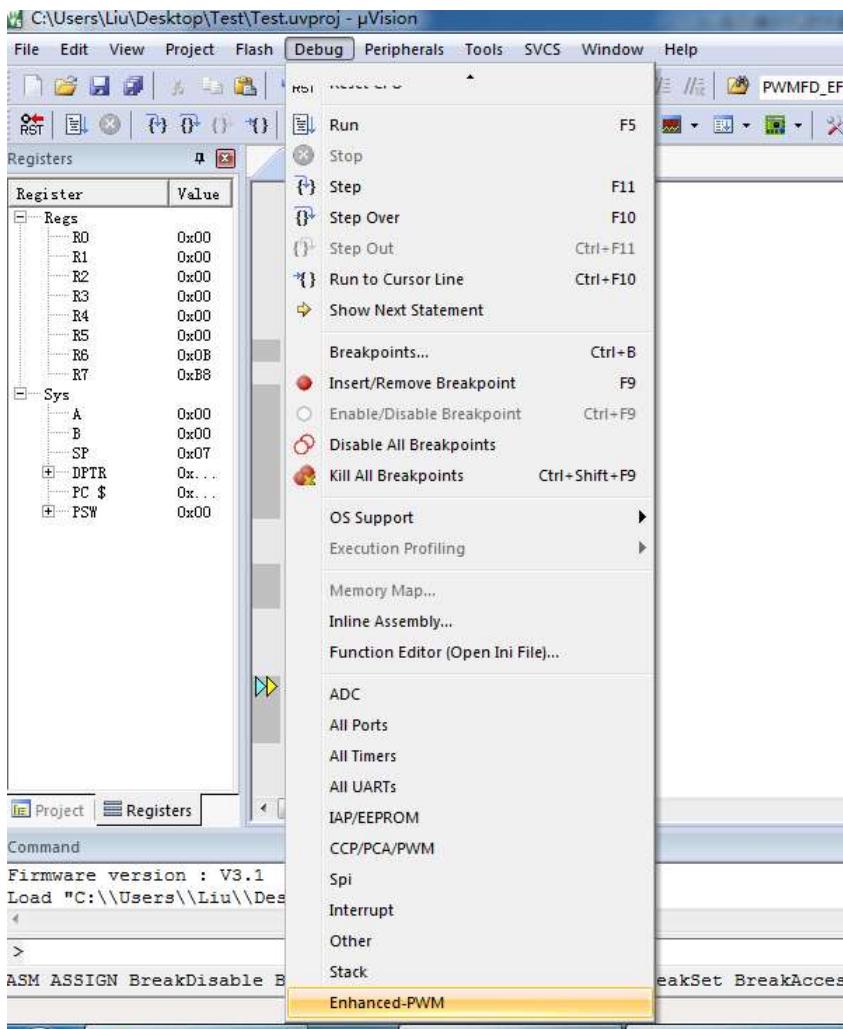
6、仿真过程中，寄存器的查看

在仿真的过程中，可查看 MCU 相关的寄存器。所有的寄存器列表在“Debug”菜单的底端。如下图所示：



在上图“Debug”菜单的最底端，还有一个黑色的小三角，这表示还有隐藏的项目（主要是由于显示版面大小的原因）

将鼠标仿真小三角上即可自动拖出所有的项目，如下图：



仿真注意事项：

- 1、仿真监控程序占用 P3.0/P3.1 两个端口，但不占用串口 1，用户可以将串口 1 切换到 P3.6/P3.7 或者 P1.6/P1.7 再使用
- 2、仿真监控程序占用内部扩展 RAM(XDATA)的最后 768 字节，用户不可对这个区域的 XDATA 进行写操作

附录B STC-ISP 下载软件高级应用

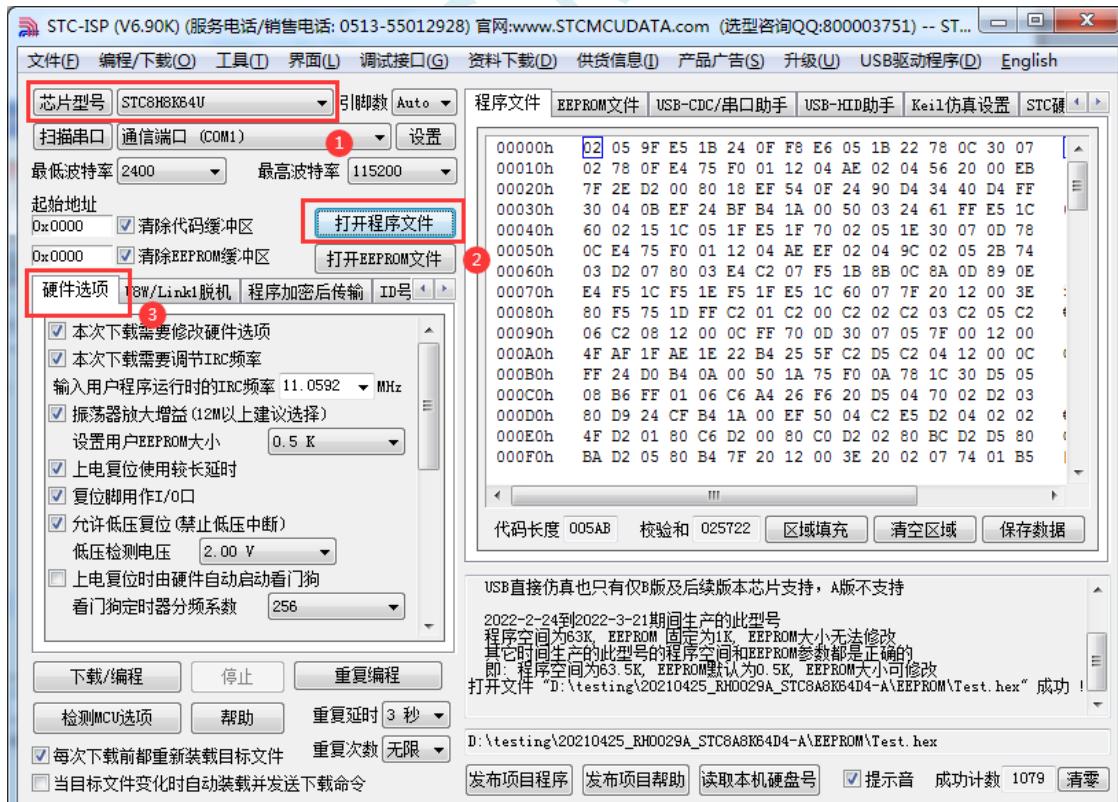
B.1 发布项目程序

发布项目程序功能主要是将用户的程序代码与相关的选项设置打包成为一个可以直接对目标芯片进行下载编程的超级简单的用户自己界面的可执行文件。

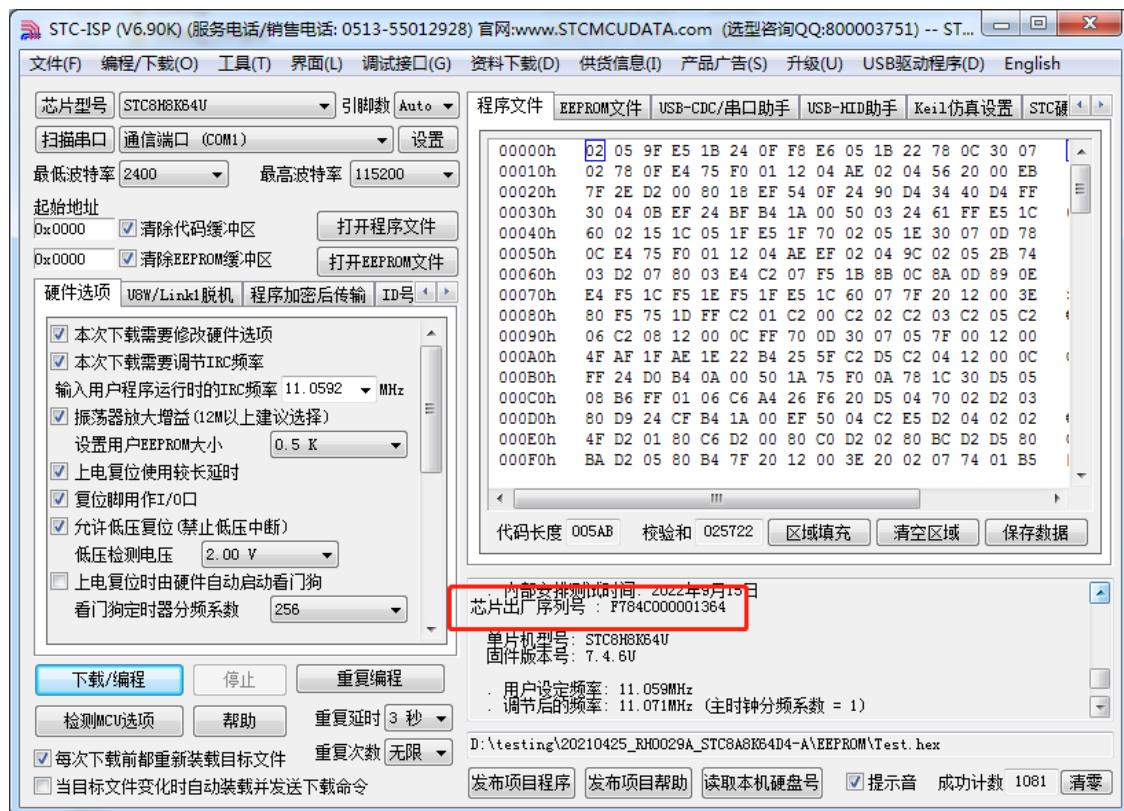
关于界面，用户可以自己进行定制（用户可以自行修改发布项目程序的标题、按钮名称以及帮助信息），同时用户还可以指定目标电脑的硬盘号和目标芯片的 ID 号，指定目标电脑的硬盘号后，便可以控制发布应用程序只能在指定的电脑上运行（防止烧录人员将程序轻易从电脑盗走，如通过网络发走，如通过 U 盘拷走，防不胜防，当然盗走你的电脑那就没办法了，所以 STC 的脱机下载工具比电脑烧录安全，能限制可烧录芯片数量，让前台文员小姐烧，让老板娘烧都可以），拷贝到其它电脑，应用程序不能运行。同样的，当指定了目标芯片的 ID 号后，那么用户代码只能下载到具有相应 ID 号的目标芯片中（对于一台设备要卖几千万的产品特别有用---坦克，可以发给客户自己升级，不需冒着生命危险跑到战火纷飞的伊拉克升级软件啦），对于 ID 号不一致的其它芯片，不能进行下载编程。

发布项目程序详细的操作步骤如下：

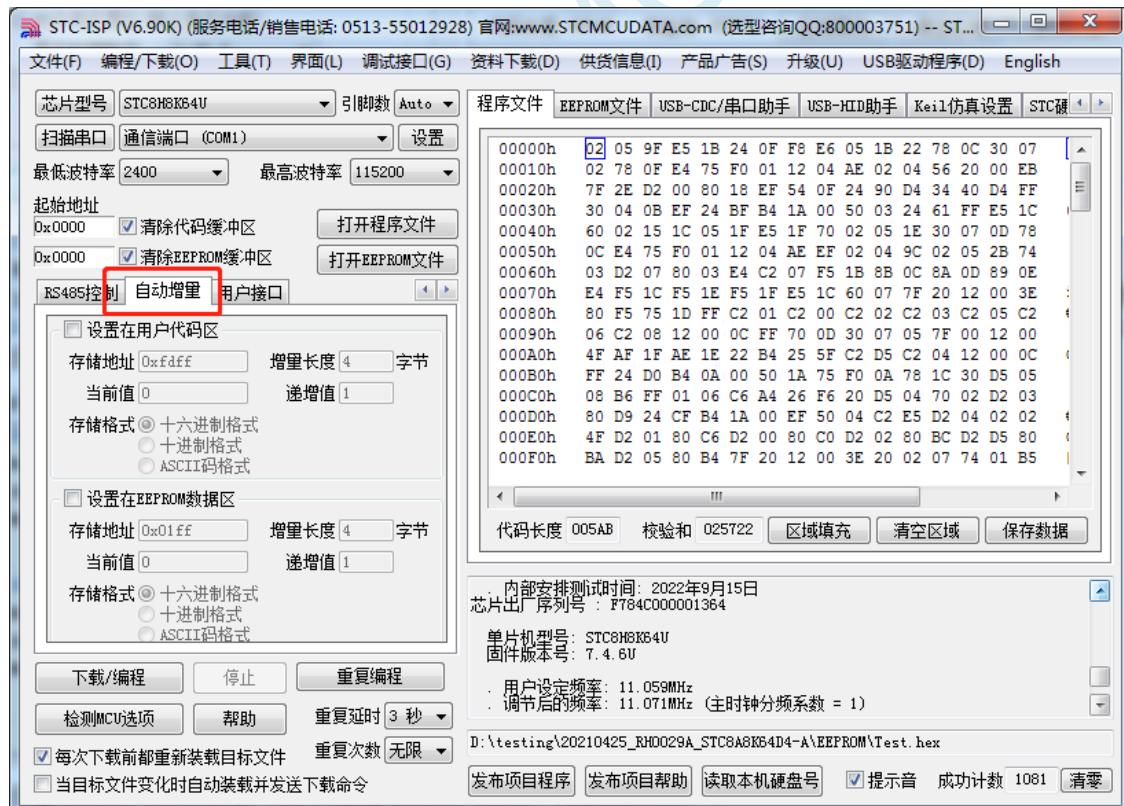
- 1、首先选择目标芯片的型号
- 2、打开程序代码文件
- 3、设置好相应的硬件选项



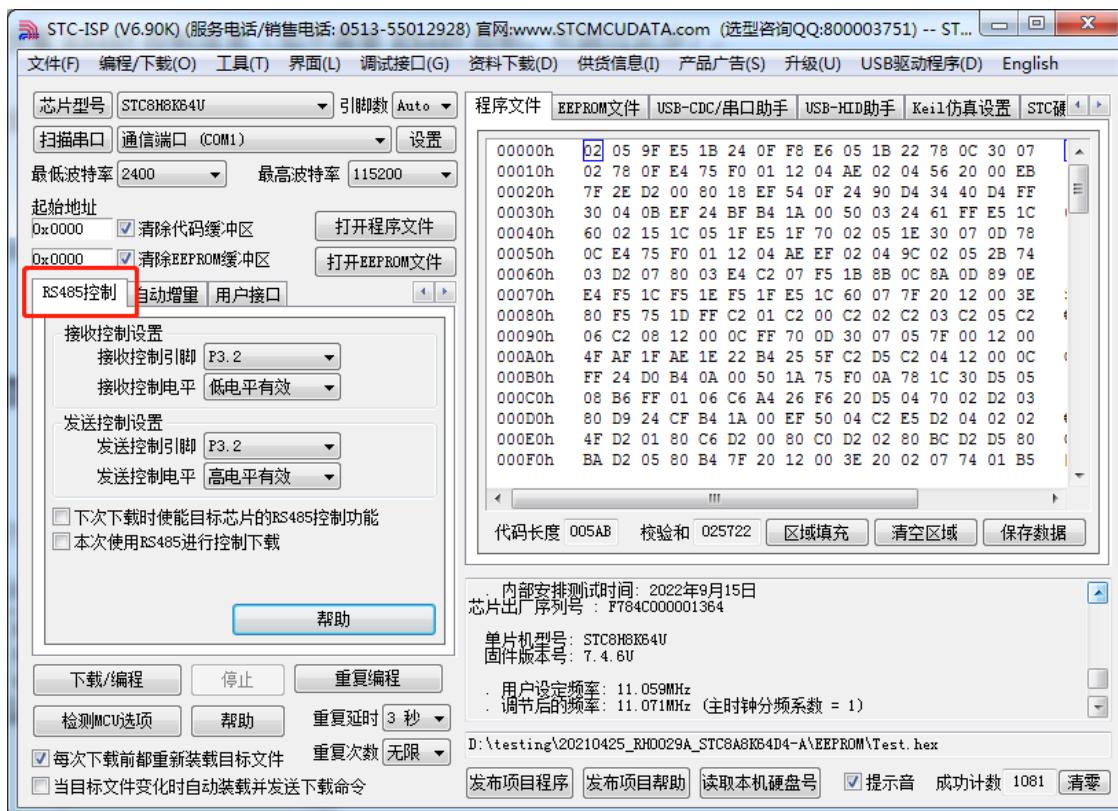
- 4、试烧一下芯片，并记下目标芯片的 ID 号，如下图所示，该芯片的 ID 号即为“F784C000001364”（如不需要对目标芯片的 ID 号进行校验，可跳过此步）



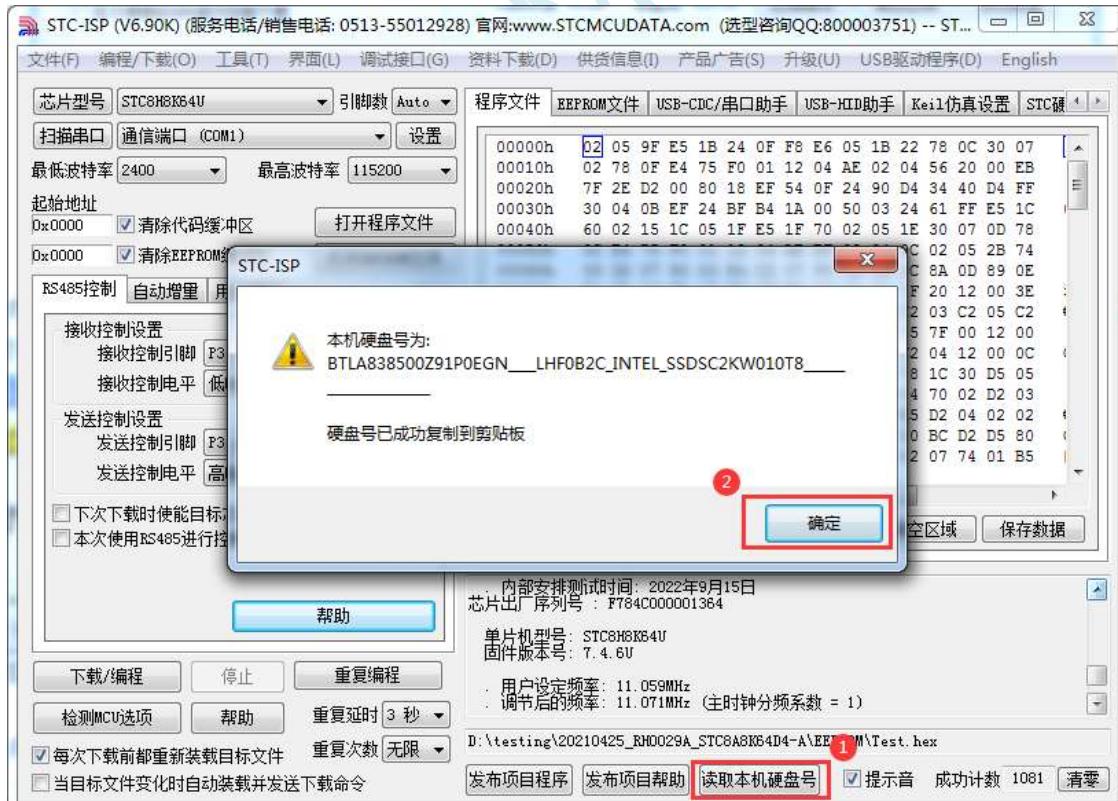
5、设置自动增量（如不需要自动增量，可跳过此步）



6、设置 RS485 控制信息（如不需要 RS485 控制，可跳过此步）



7、点击界面上的“读取本机硬盘号”按钮，并记下目标电脑的硬盘号（如不需要对目标电脑的硬盘号进行校验，可跳过此步）



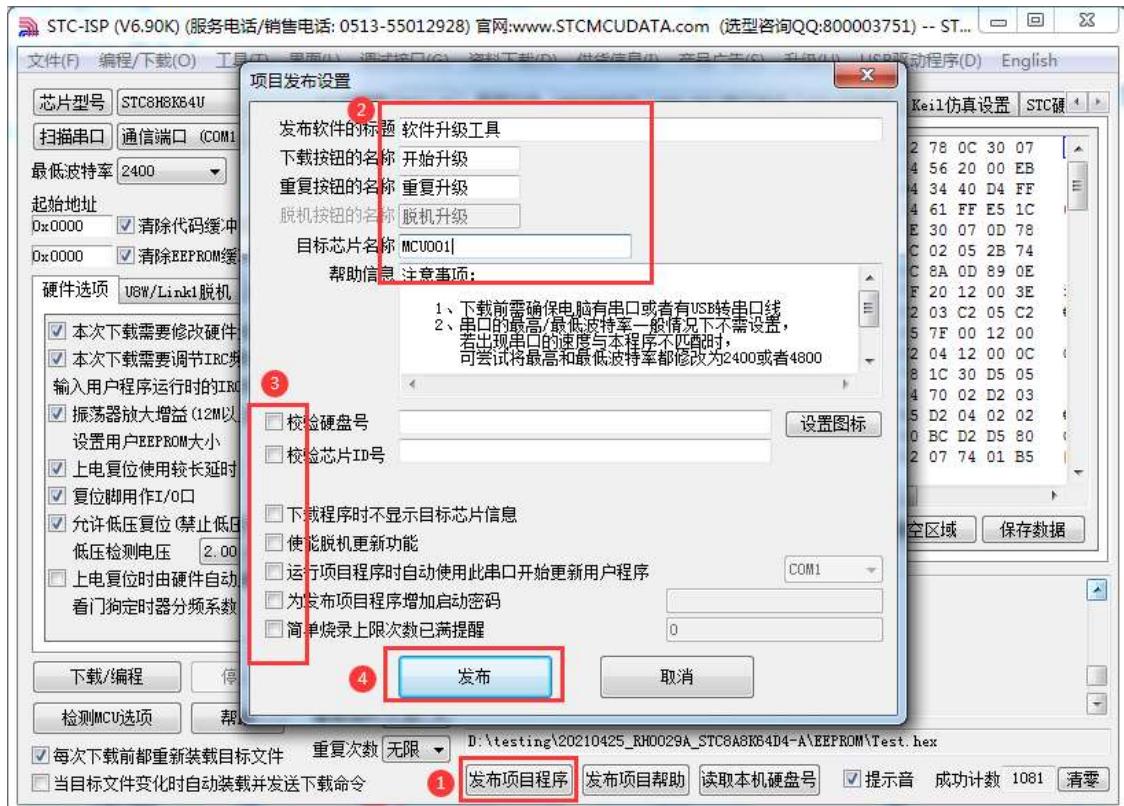
8、点击“发布项目程序”按钮，进入发布应用程序的设置界面。

9、根据各自的需要，修改发布软件的标题、下载按钮的名称、重复下载按钮的名称、自动增量的名称以及帮助信息

10、若需要校验目标电脑的硬盘号，则需要勾选上“校验硬盘号”，并在后面的文本框内输入前面所记

下的目标电脑的硬盘号

- 11、若需要校验目标芯片的 ID 号，则需要勾选上“校验芯片 ID 号”，并在后面的文本框内输入前面所记下的目标芯片的 ID 号



- 12、最后点击发布按钮，将项目发布程序保存，即可得到相应的可执行文件。发布的项目程序界面如下图



B.2 程序加密后传输（防烧录时串口分析出程序）

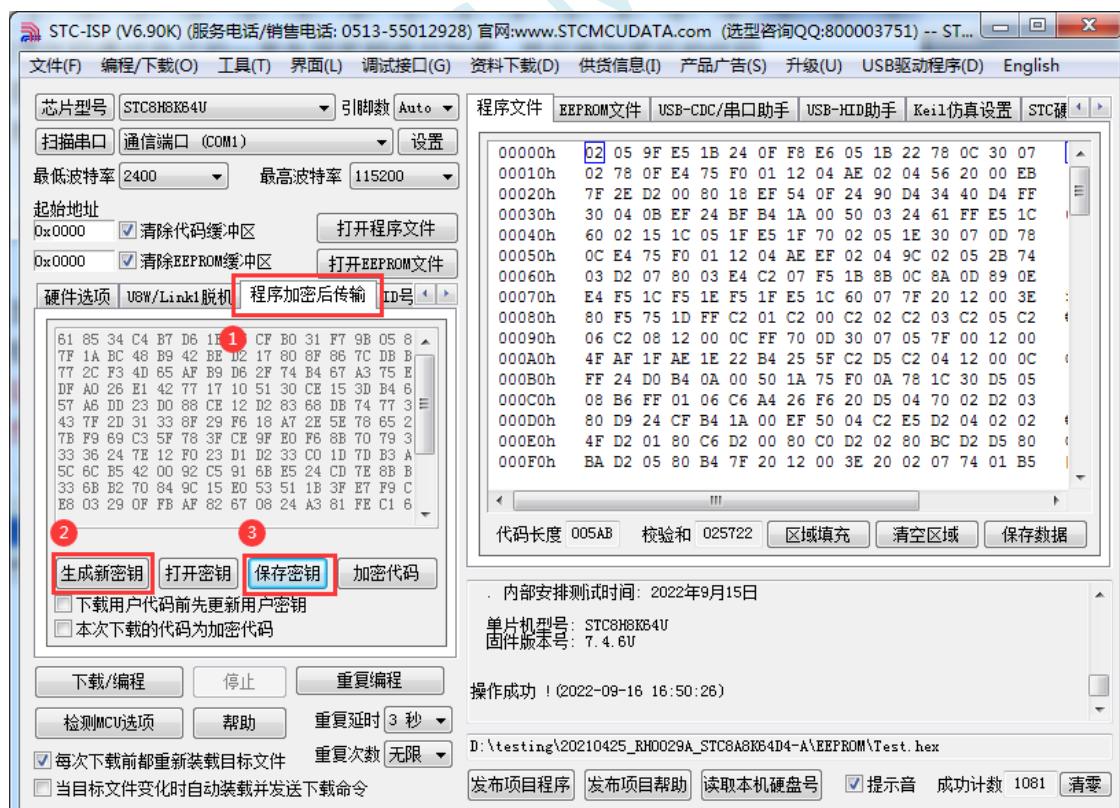
目前，所有的普通串口下载烧录编程都是采用明码通信的（电脑和目标芯片通信时，或脱机下载板和目标芯片通信时），问题：如果烧录人员通过分析下载烧录编程时串口通信的数据，高手是可以在烧录时在串口上引 2 根线出来，通过分析串口通信的数据分析出实际的用户程序代码的。**当然用 STC 的脱机下载板烧程序总比用电脑烧程序强（防止烧录人员将程序轻易从电脑盗走，如通过网络发走，如通过 U 盘拷走，防不胜防，当然盗走你的电脑那就没办法那，所以 STC 的脱机下载工具比电脑烧录安全，让前台文员小姐烧，让老板娘烧都可以）。**即使是 STC 全球首创的脱机下载工具，对于要防止天才的不法分子在脱机下载工具烧录的过程中通过分析串口通信的数据分析出实际的用户程序代码，也是没有办法达到要求的，这就需要用到最新的 STC 单片机所提供的程序加密后传输功能。

程序加密后传输下载是用户先将程序代码通过自己的一套专用密钥进行加密，然后将加密后的代码再通过串口下载，此时下载传输的是加密文件，通过串口分析出来的是加密后的乱码，如不通过派人潜入你公司盗窃你电脑里面的加密密钥，就无任何价值，便可起到防止在烧录程序时被烧录人员通过监测串口分析出代码的目的。

程序加密后传输功能的使用需要如下的几个步骤：

1、生成并保存新的密钥

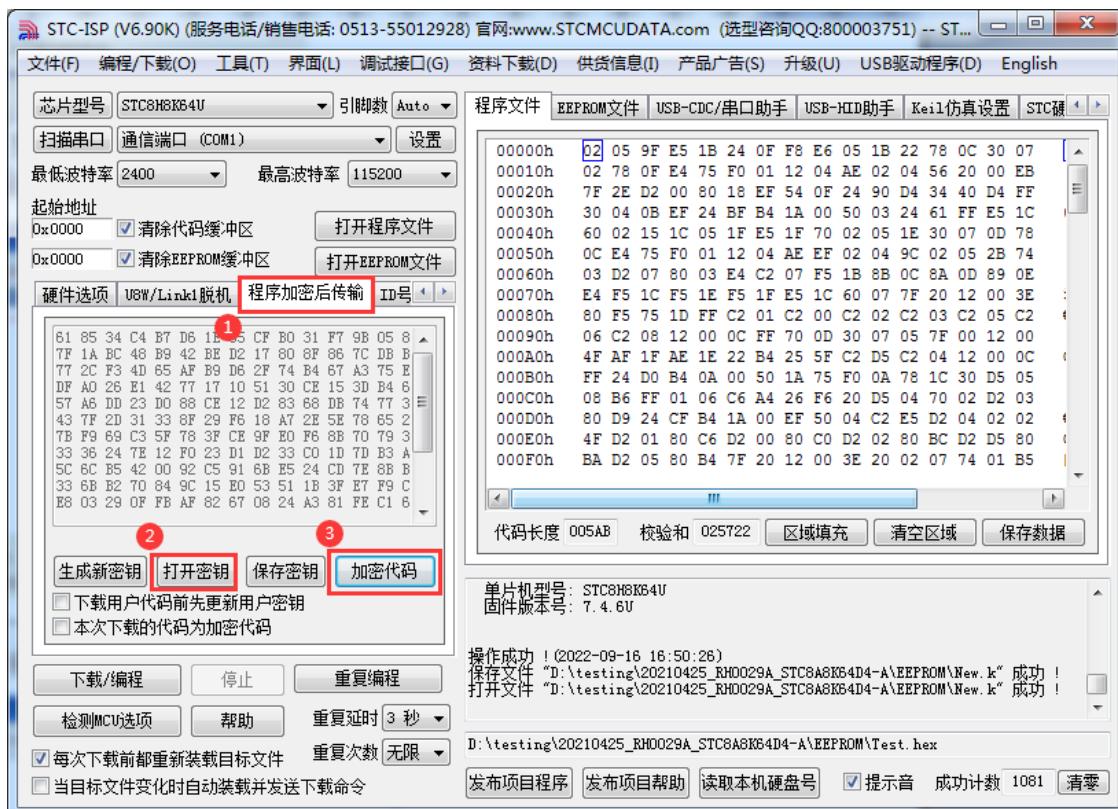
如下图，进入到“程序加密后传输”页面，点击“生成新密钥”按钮，即可在缓冲区显示新生成的 256 字节的密钥。然后点击“保存密钥”按钮，即可将生成的新密钥保存为以“.K”为扩展名的密钥文件（**注意：这个密钥文件一定要保存好，以后发布的代码文件都需要使用这个密钥加密，而且这个密钥的生成是非重复的，即任何时候都不可能生成两个完全相同的密钥，所以一旦密钥文件丢失将无法重新获得**）。例如我们将密钥保存为“New.k”。



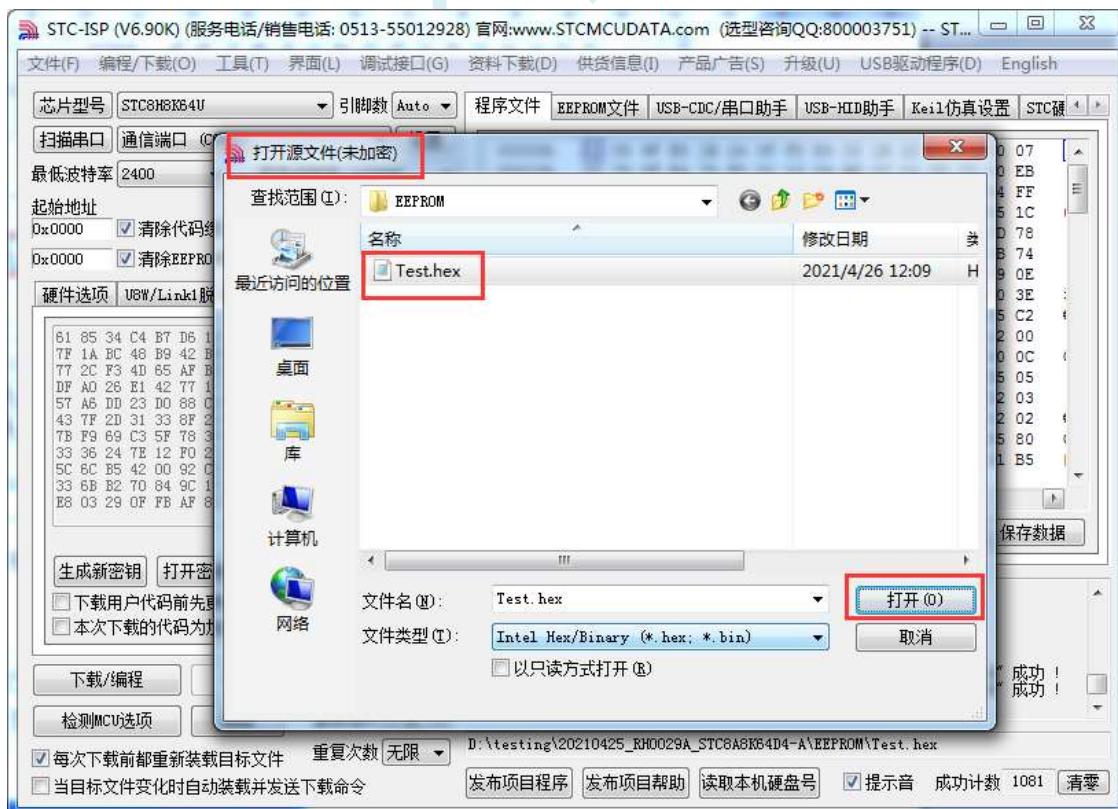
2、对代码文件加密

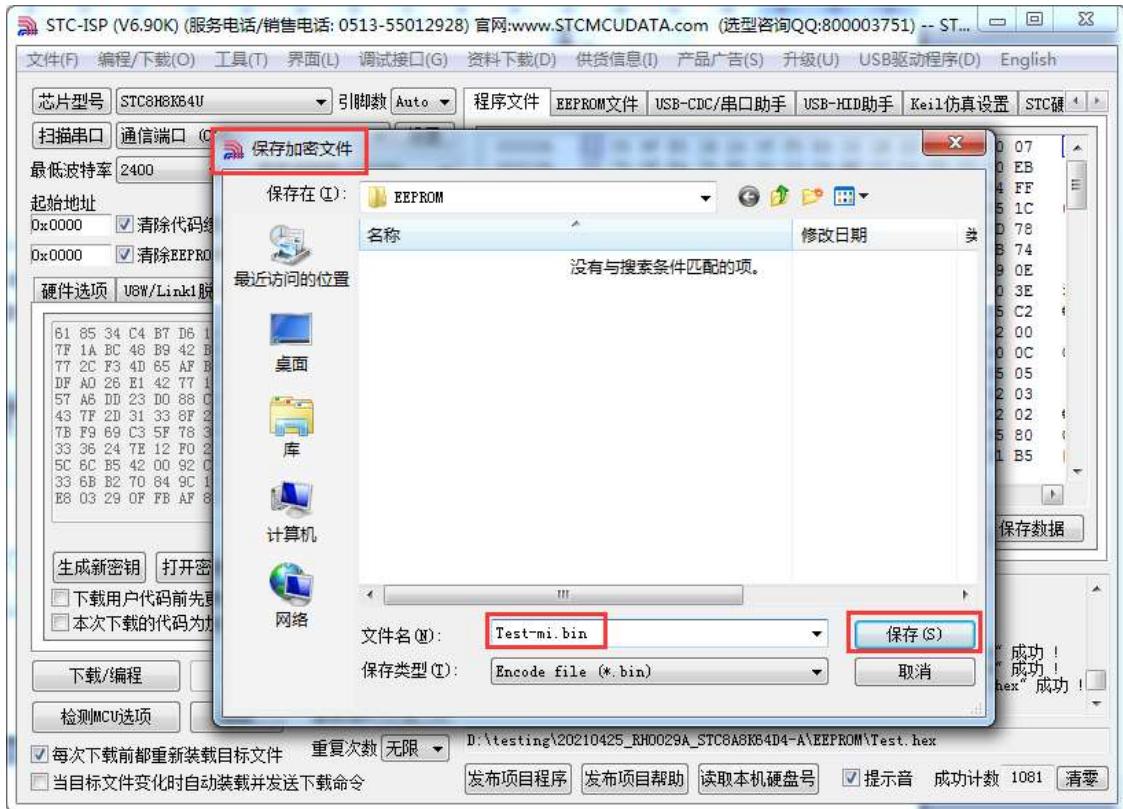
加密文件前，需要先打开我们自己的密钥。若缓冲区中存放的已经是我们的密钥，则不要再打开。如下图，在“程序加密后传输”页面中点击“打开密钥”按钮，打开我们之前保存的密钥文件，

例如“New.k”，然后返回到“程序加密后传输”页面中点击“加密代码”按钮，如下图所示，首先会弹出“打开源文件(未加密)”的对话框，此时选择的是原始的未加密的代码文件



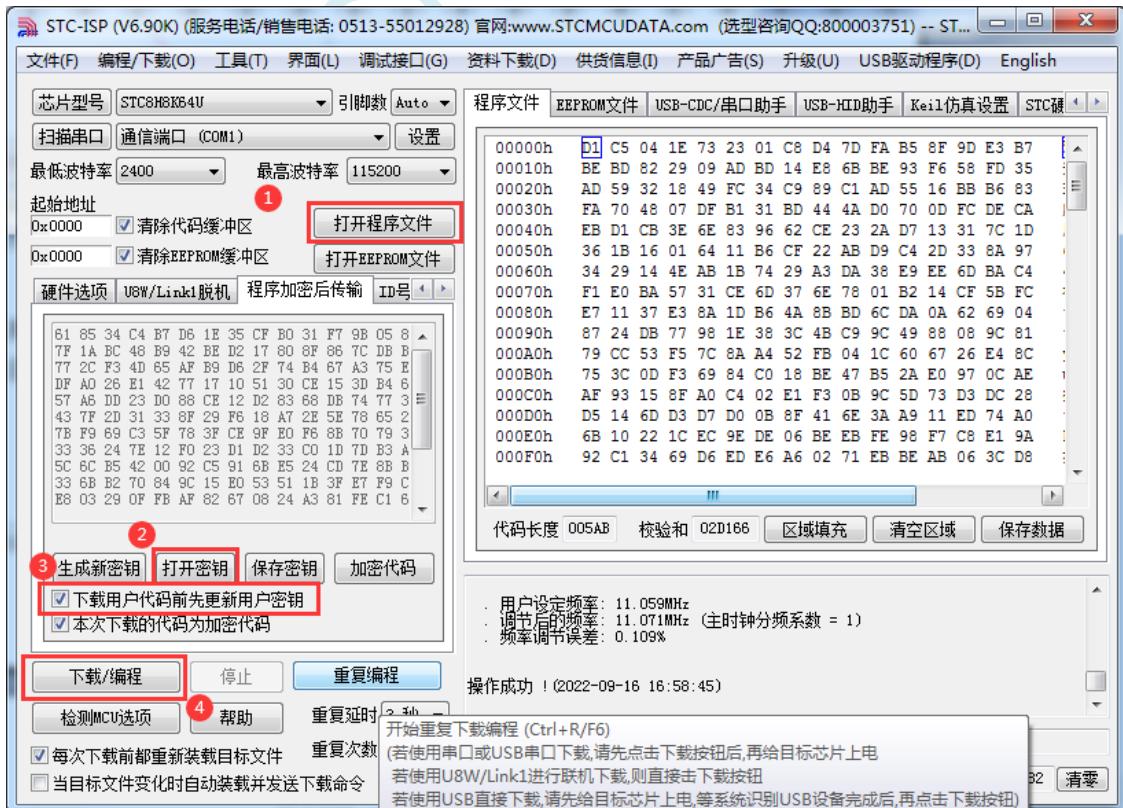
点击打开按钮后，马上会有弹出一个类似的对话框，但此时是对加密后的文件进行保存的对话框。如下图所示，点击保存按钮即可保存加密后的文件。





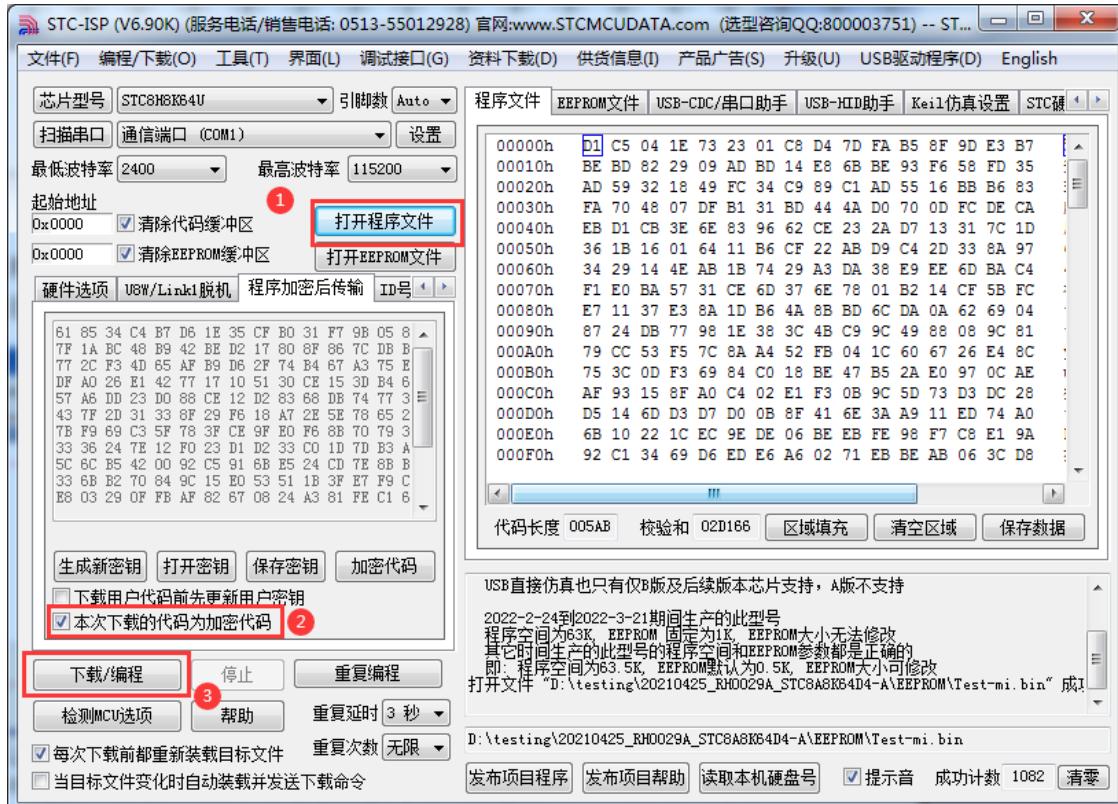
3、将用户密钥更新到目标芯片中

更新密钥前，需要先打开我们自己的密钥。若缓冲区中存放的已经是我们的密钥，则不要再打开。如下图，在“自定义加密下载”页面中点击“打开密钥”按钮，打开我们之前保存的密钥文件，例如“New.k”。密钥打开后，如下图所示，勾选上“下载用户代码前先更新用户密钥”选项和“本次下载的代码为加密代码”的选项，然后打开我们之前加密过后的文件，打开后点击界面左下角的“下载/编程”按钮，按正常方式对目标芯片下载完成即可更新用户密钥。



4、加密更新用户代码

密钥更新成功后，目标芯片便具有接收加密代码并还原的功能。此时若需要再次升级/更新代码，则只需要参考第二步的方法，将目标代码进行加密，然后如下图



对于一片新的 STC 单片机，可将步骤 3 和步骤 4 合并完成，即将密钥更新到目标单片机的同时也可将加密后的代码一并下载到单片机中，若已经执行过步骤 3（即已经将密钥更新到目标芯片中了），则后续的代码更新就只需要按照步骤 4，只需要在“程序加密后传输”页面中选择“本次下载的代码为加密代码”的选项（“**下载用户代码前先更新用户密钥**”选项不需要选了），然后打开我们之前加过密后的文件，打开后点击界面左下角的“下载/编程”按钮，按正常方式对目标芯片下载即可完成用用户自己专用的加密文件更新用户代码的目的（防止在烧录程序时被烧录人员通过监测串口分析出代码的目的）。

B.3 发布项目程序+程序加密后传输结合使用

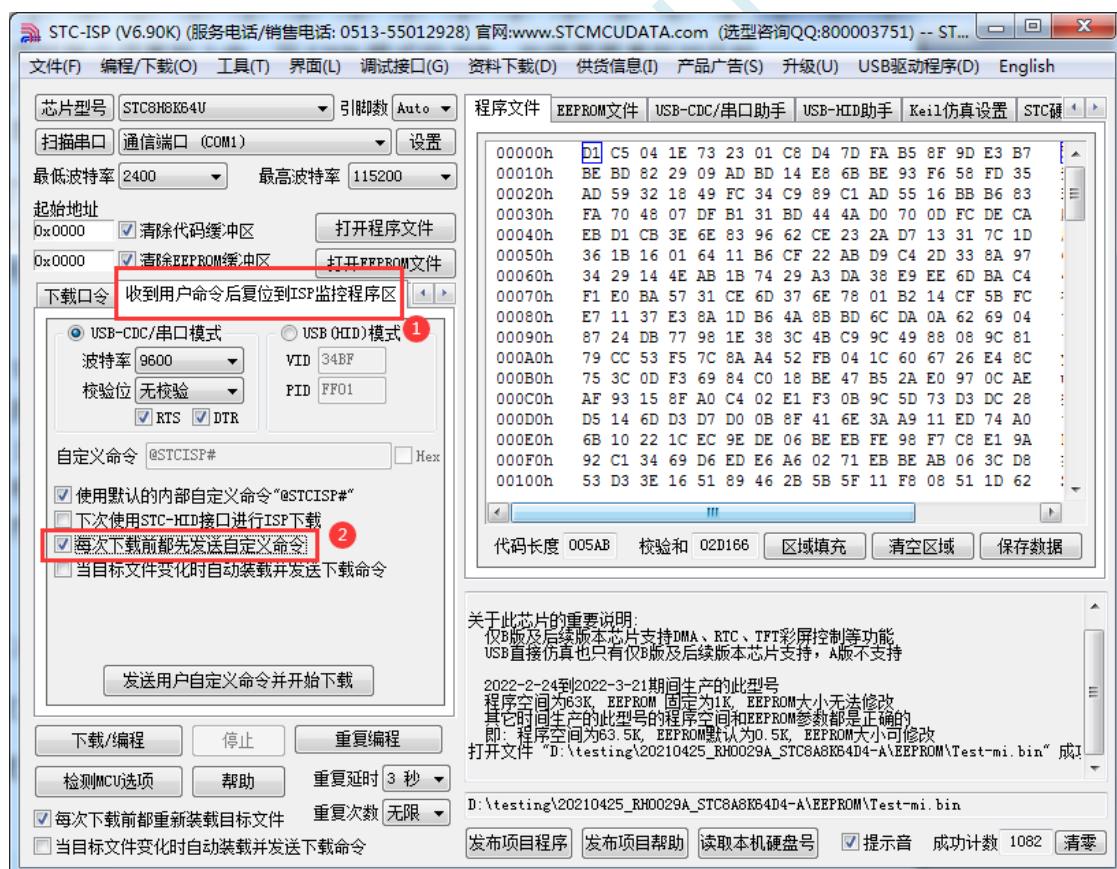
发布项目程序与程序加密后传输两项新的特殊功能可以结合在一起使用。首先程序加密后传输可以确保用户代码在烧录编程时串口通信传输过程当中的保密性，而发布项目程序可实现让最终使用者远程升级功能（方案公司的人员不需要亲自到场）。所以两项功能结合起来使用，非常适用于方案公司/生产商在软件需要更新时，让最终使用者自己对终端产品进行软件更新的目的，又确保现场烧录人员无法通过串口分析出有用程序，强烈建议方案公司使用。

B.4 用户自定义下载（实现不停电下载）

将用户的目标程序下载到 STC 单片机是通过执行单片机内部的 ISP 系统代码和上位机进行串口或者 USB 通讯来实现的。但 STC 单片机内部的 ISP 系统代码只有在每次重新停电再上电时才会被执行，这就要求用户每次需要对目标单片机更新程序时就必须重新上电，而 USB 模式的 ISP，处理需要重新对目标芯片上电外，还需要在上电时将 P3.2 口下拉到 GND。对于处于开发阶段的项目，需要频繁的修改代码、更新代码，每次下载都需要重新上电会导致操作非常麻烦。

STC 单片机在硬件设计时，增加了一个软复位寄存器 (IAP_CONTR)，让用户可以通过设置此寄存器来决定 CPU 复位后重新执行用户代码还是复位到 ISP 区执行 ISP 系统代码。当向 IAP_CONTR 寄存器写入 0x20 时，CPU 复位后重新执行用户代码；当向 IAP_CONTR 寄存器写入 0x60 时，CPU 复位后复位到 ISP 区执行 ISP 系统代码。

要实现不停电进行 ISP 下载，用户可以在程序中设计一段代码，例如检测一个特殊的按键、或者监控串口等待一个特殊的串口命令，当检测到满足下载条件时，就通过软件触发软复位寄存器复位到 ISP 区执行 ISP 系统代码，从而实现不停电 ISP 下载。当触发条件是外部按键时，则在用户代码中实时监控按键状态即可。若要实现 STC-ISP 软件和用户触发软复位完全同步，则需要使用 STC-ISP 软件中所提供的“收到用户命令后复位到 ISP 监控程序区”这个功能。



实现不停电 ISP 下载的步骤如下:

1、编写用户代码，并在用户代码中添加串口命令监控程序
(参考代码如下，测试单片机型号为 STC8H8K64U)

```
#include "stc8h.h"

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

char code *STCISPCMD = "@STCISP#";           //自定义下载命令
char index;

void uart_isr() interrupt 4
{
    char dat;

    if (TI)
    {
        TI = 0;
    }

    if (RI)
    {
        RI = 0;
        dat = SBUF;           //接收串口数据

        if (dat == STCISPCMD[index]) //判断接收的数据和当前的命令字符是否匹配
        {
            index++;           //若匹配则索引+1
            if (STCISPCMD[index] == '\0') //判断命令是否配完成
                IAP_CONTR = 0x60;       //若匹配完成则软复位到 ISP
        }
        else
        {
            index = 0;           //若不匹配,则需要从头开始
            if (dat == STCISPCMD[index])
                index++;
        }
    }
}

void main()
{
    P0M0 = 0x00; P0M1 = 0x00;
    P1M0 = 0x00; P1M1 = 0x00;
```

```

P2M0 = 0x00; P2M1 = 0x00;
P3M0 = 0x00; P3M1 = 0x00;

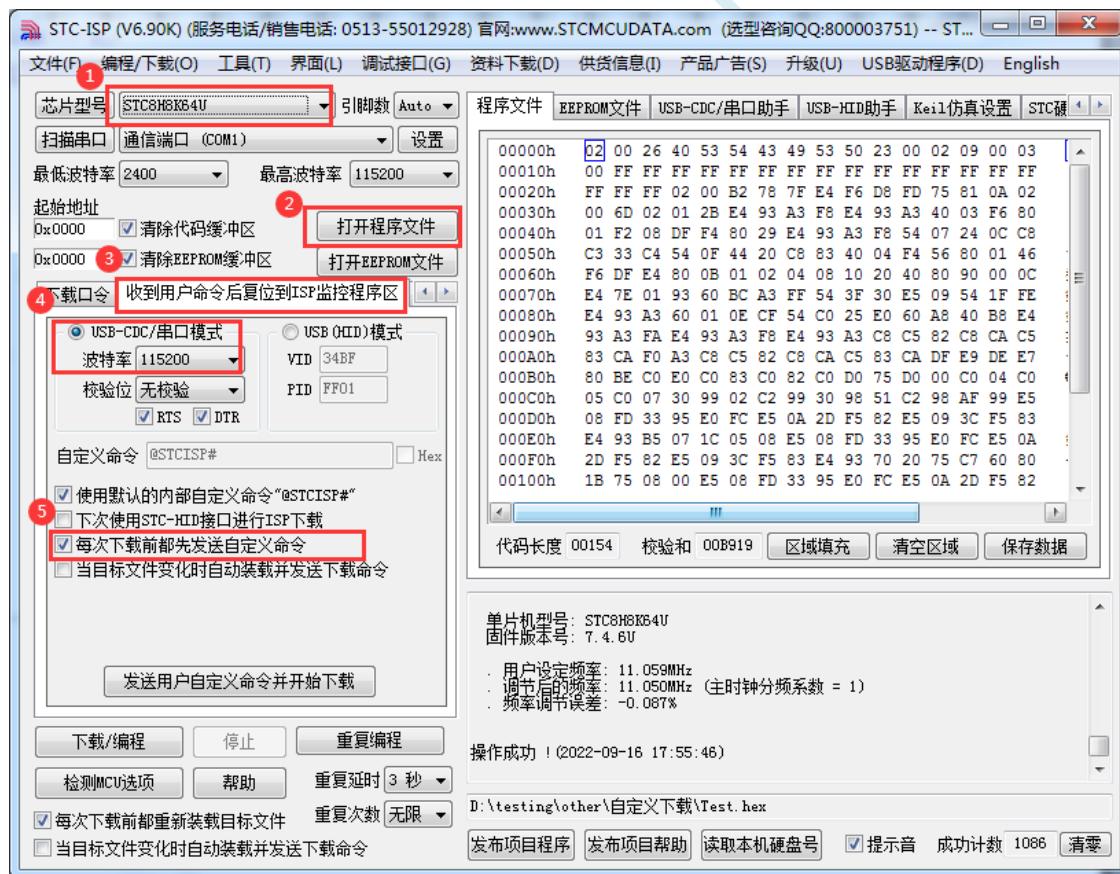
SCON = 0x50;           //串口初始化
AUXR = 0x40;
TMOD = 0x00;
TH1 = BAUD >> 8;
TL1 = BAUD;
TR1 = 1;
ES = 1;
EA = 1;

index = 0;             //初始化命令

while (1);
}

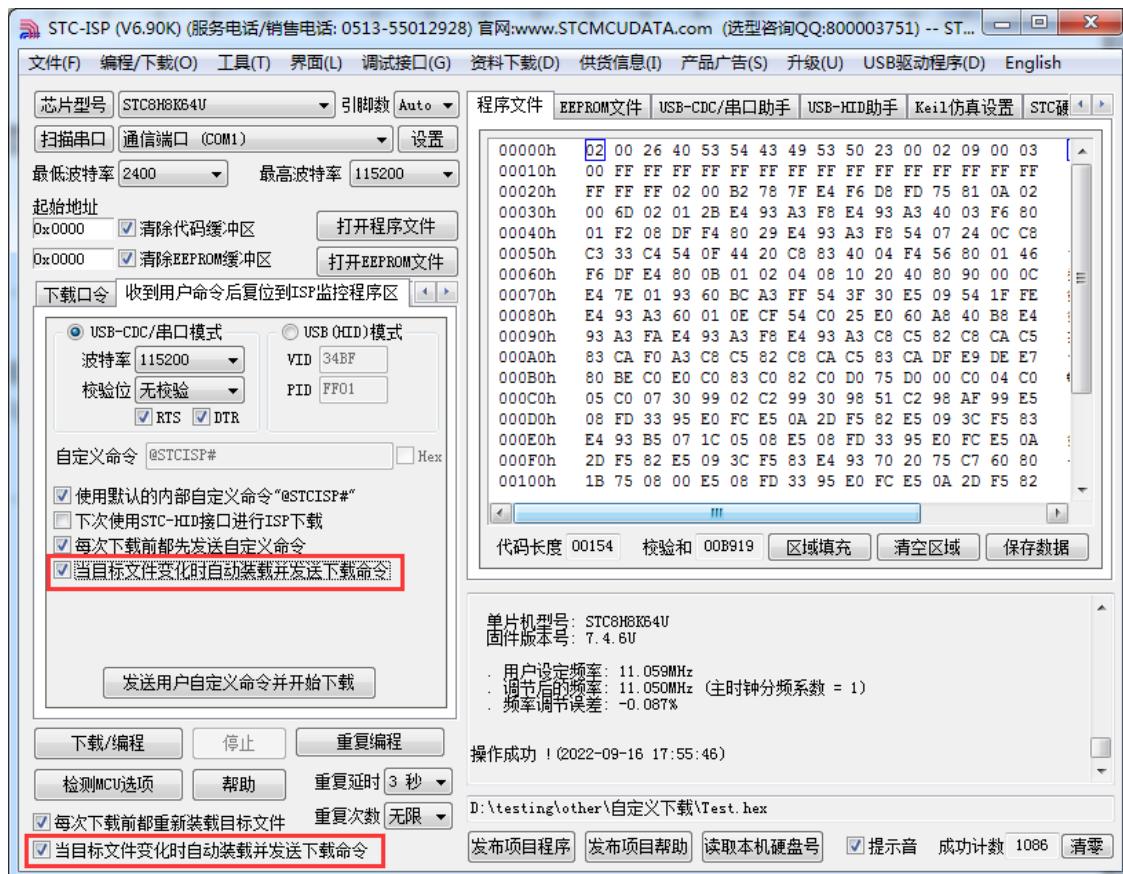
```

2、按下图所示的步骤进行设置自定义下载命令（范例使用 STC 默认命令“@STCISP#”）

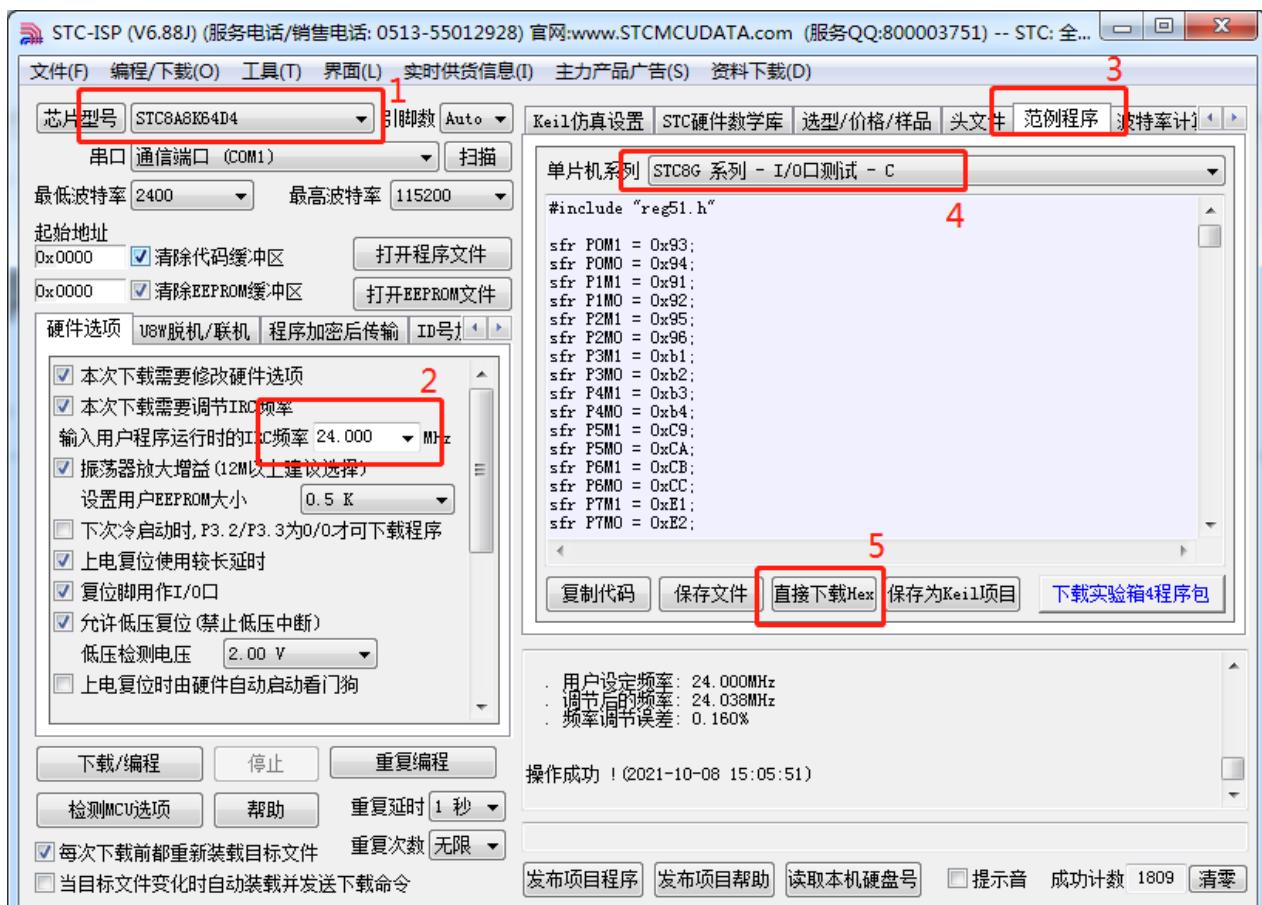


3、第一次下载时需要对目标单片机重新上电，之后的每次更新只需要点击下载软件中的“下载/编程”按钮，下载软件自动将下载命令发送给目标单片机，目标单片机接收到命令后自动复位到系统 ISP 区，即可实现不停电更新用户代码。

4、STC-ISP 还可实现项目开发阶段，完全自动下载功能，即当下载软件侦测到目标代码被更新了，就会自动发送下载命令。要实现这个功能只需要勾选下图中的两个选项中的任意一个即可



附录C 如何测试 I/O 口



测试 I/O 口步骤:

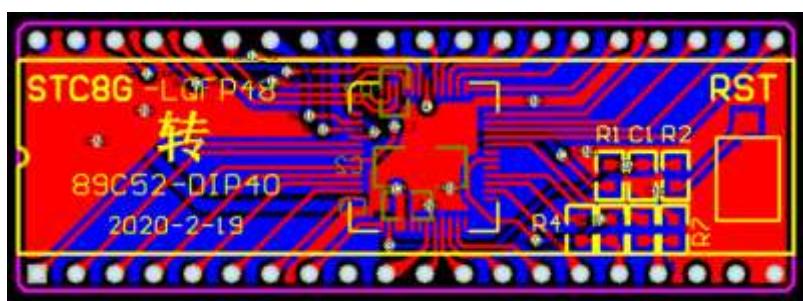
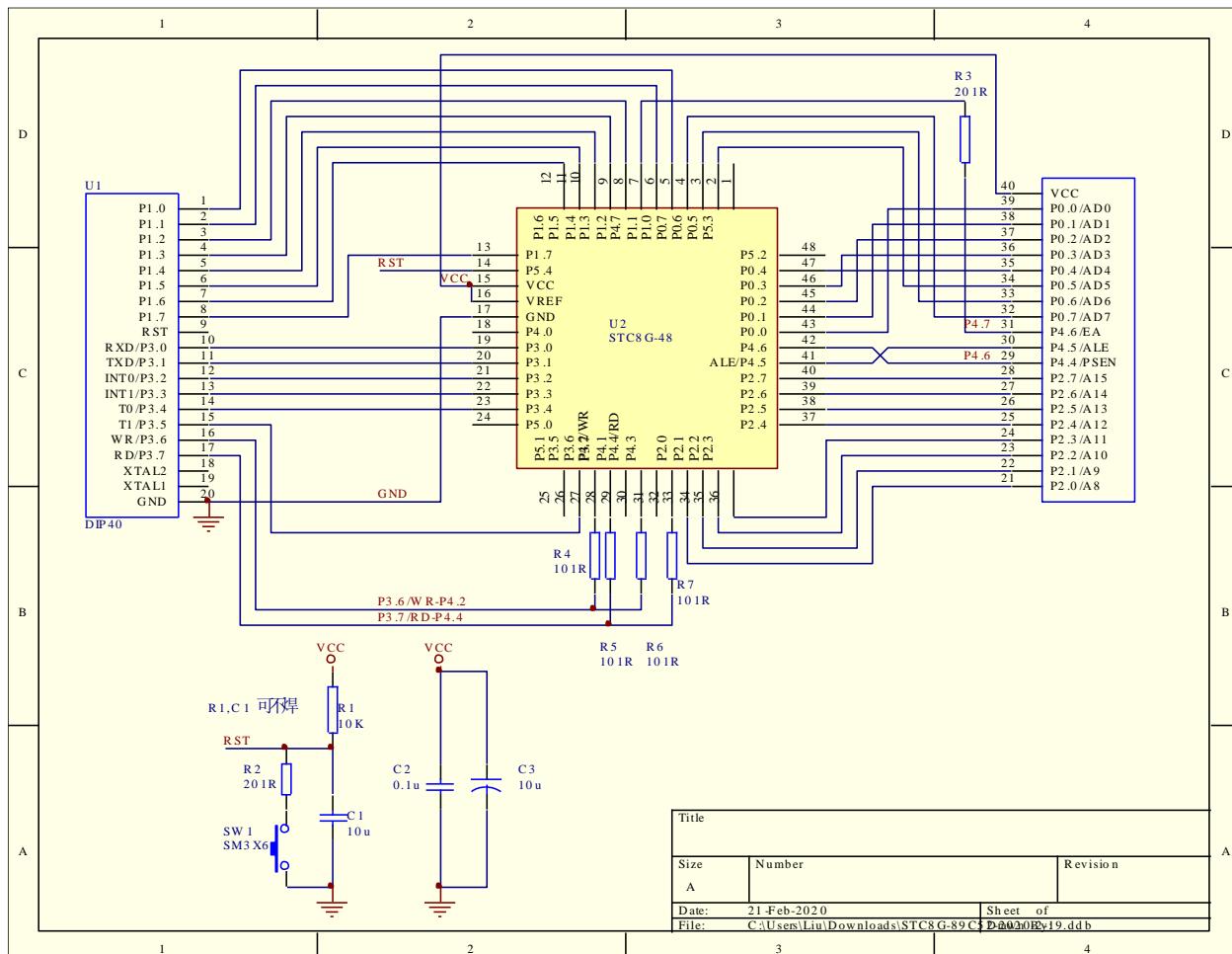
- 1、选择单片机型号
- 2、设置测试程序的工作频率 (24MHz)
- 3、打开“范例程序”页
- 4、选择 STC8G 或者 STC8H 系列中的“I/O 口测试”程序
- 5、点击页面中的“直接下载 Hex”

下载完成后，会对所有的 I/O 口执行流水灯程序，此时可在 I/O 口接 LED 或者用示波器即可看到波形。

附录D 如何让传统的 8051 单片机学习板可仿真

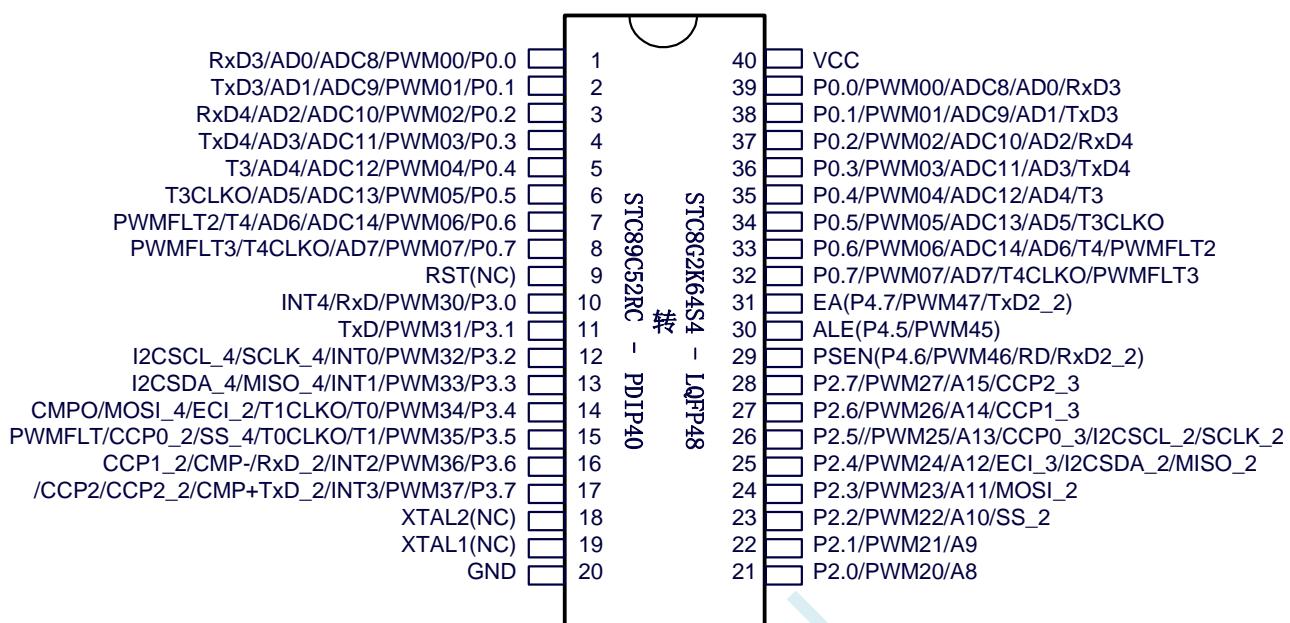
传统的 8051 单片机学习板不具有仿真功能，让传统的 8051 单片机学习板可仿真需要借助转换板，转换板的实物图如下图所示，转换后的引脚排布与传统 8051 的脚位基本一致，从而可以实现标准 8051 学习板的仿真功能。

下图是转换板的原理图和 PCB 板图



该转换板可进行 STC8G 系列 LQFP48 转 STC89C52RC/STC89C58RD+系列仿真用。

下图为转换板功能示意图



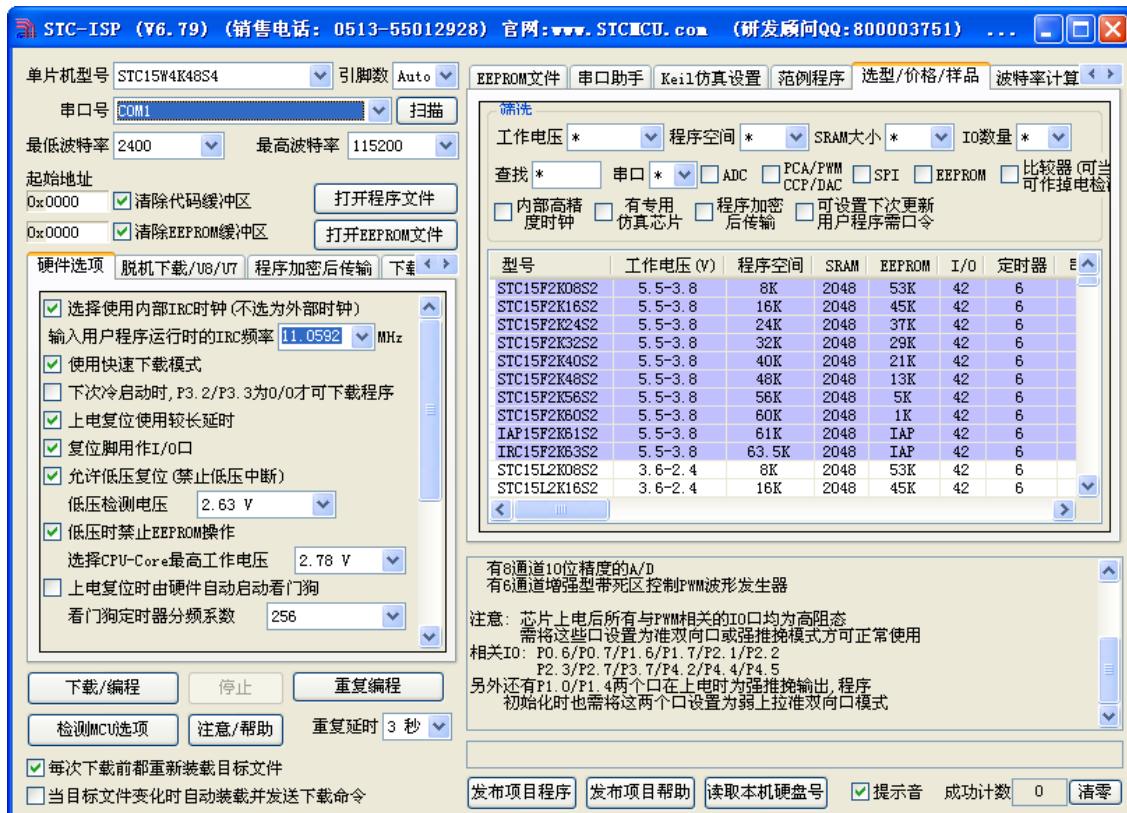
注意:

- ✓ 由于内置高精准 R/C 时钟，因此不需要外部晶振, XTAL1 和 XTAL2 是空的。
- ✓ WR 和 RD 是 (WR/P4.2 和 RD/P4.4)，而不是传统的 (WR/P3.6 和 RD/P3.7)。
(转换板中, P4.2 与 P3.6 连接在一起, P4.4 与 P3.7 连接在一起。当用户需要用此转换板访问外部总线时, 需要将 P3.6 和 P3.7 设置为高阻输入模式, 从而使 P4.2 和 P4.4 正常输出总线读写信号; 若不需要访问外部总线, 则需将 P4.2 和 P4.4 设置高阻输入模式, P3.6 和 P3.7 即为普通 I/O。)
- ✓ 由于 STC8G 系列 MCU 是低电平复位, 与传统 8051 的高电平复位不兼容, 因此 RST 管脚是悬空, 而用转换板上的复位按键加复位电路取代

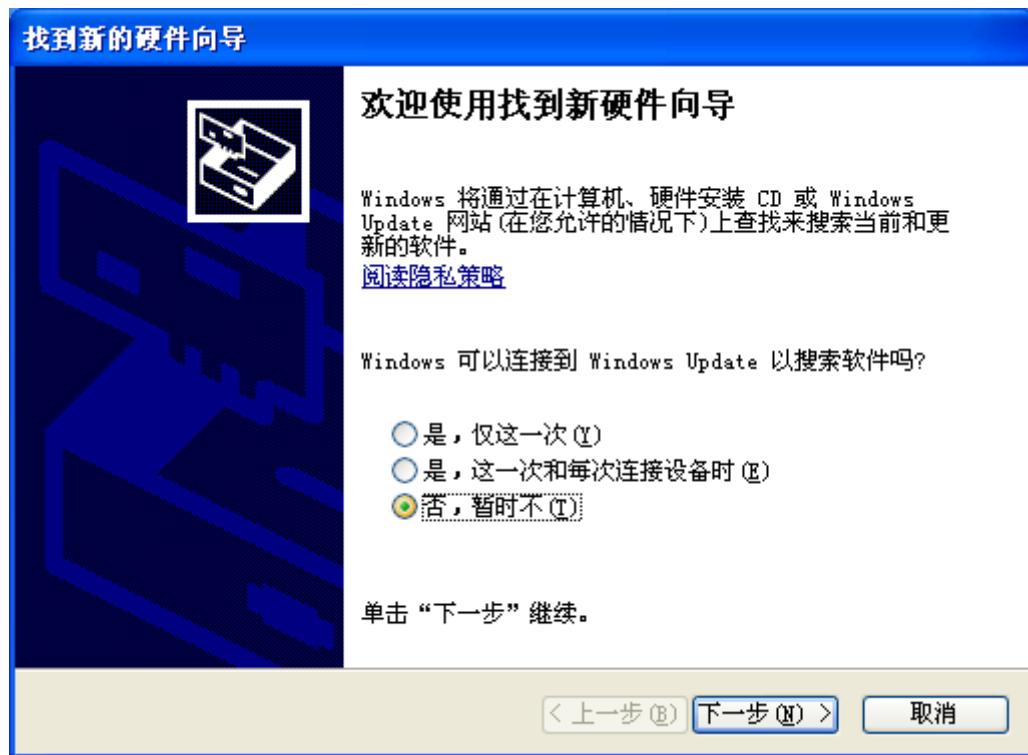
附录E STC-USB 驱动程序安装说明

Windows XP 安装方法

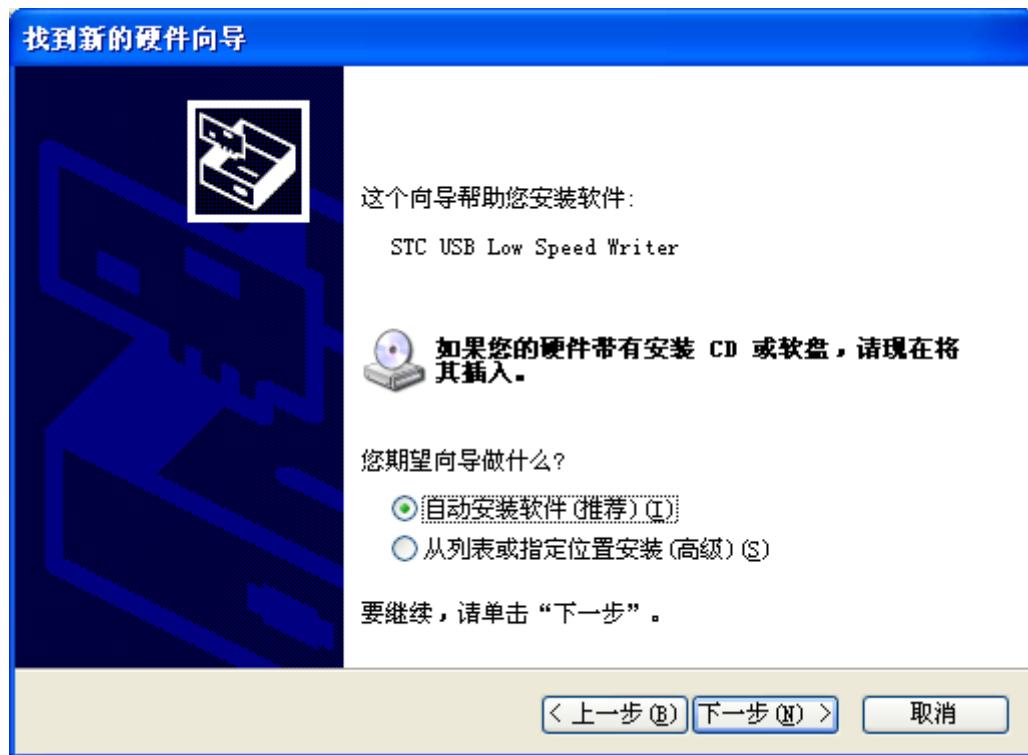
打开 V6.79 版（或者更新的版本）的 STC-ISP 下载软件，下载软件会自动将驱动文件复制到相关的系统目录



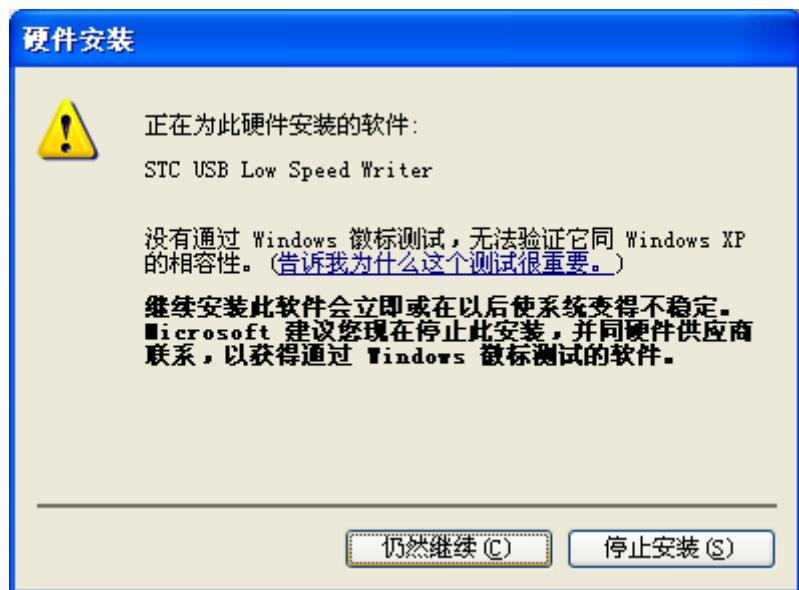
插入 USB 设备，系统找到设备后自动弹出如下对话框，选择其中的“否，暂时不”项



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



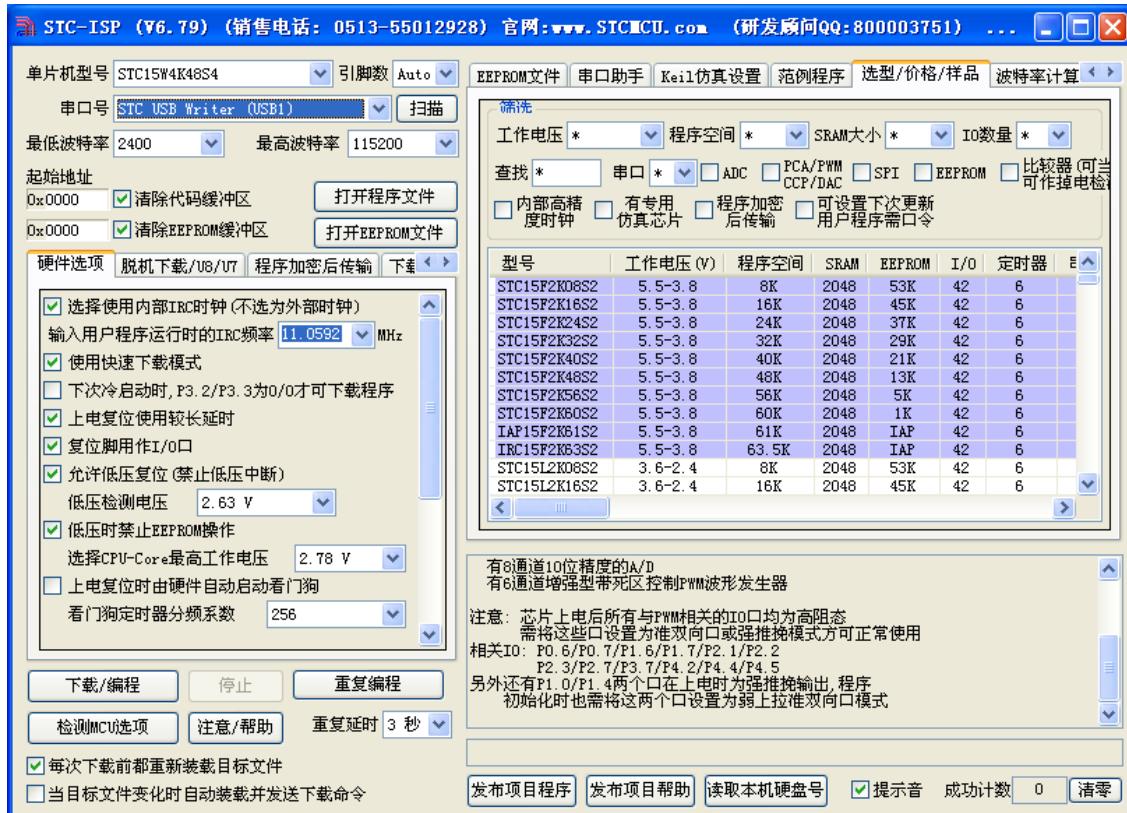
接着系统会自动安装驱动，如下图



出现下面的对话框表示驱动安装完成

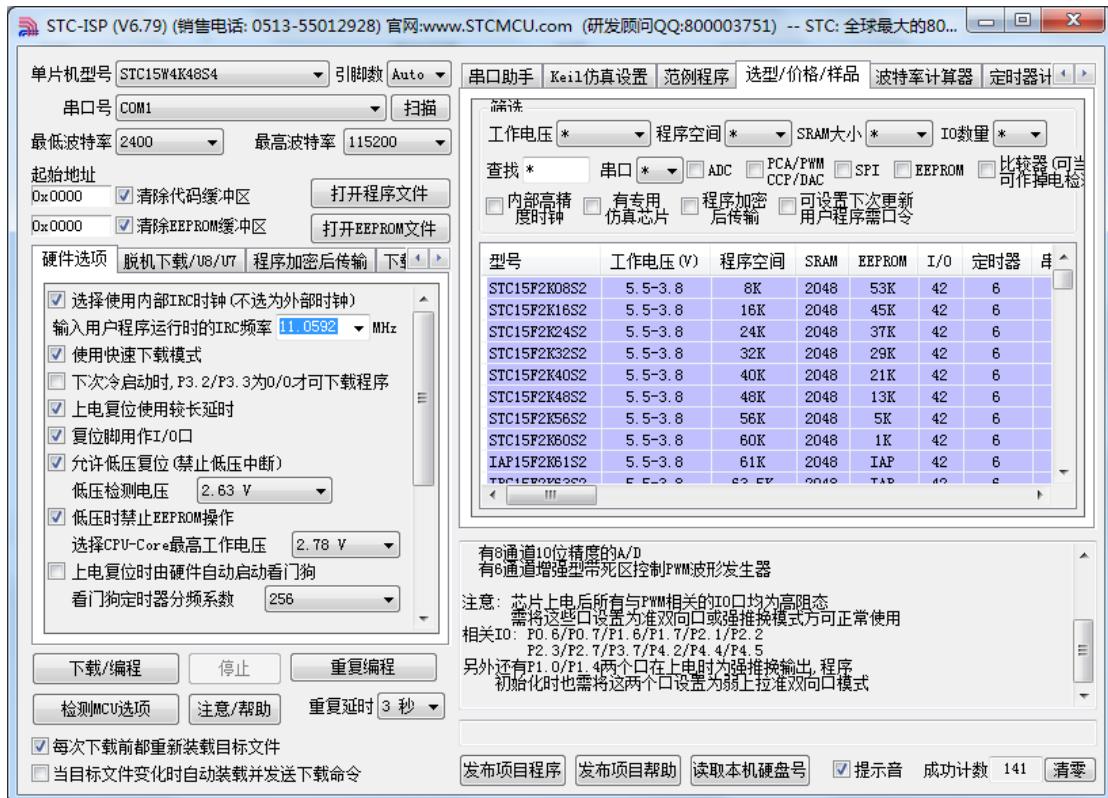


此时，之前打开的 STC-ISP 下载软件中的串口号列表会自动选择所插入的 USB 设备，并显示设备名称为“STC USB Writer (USB1)”，如下图：



Windows 7 (32位) 安装方法

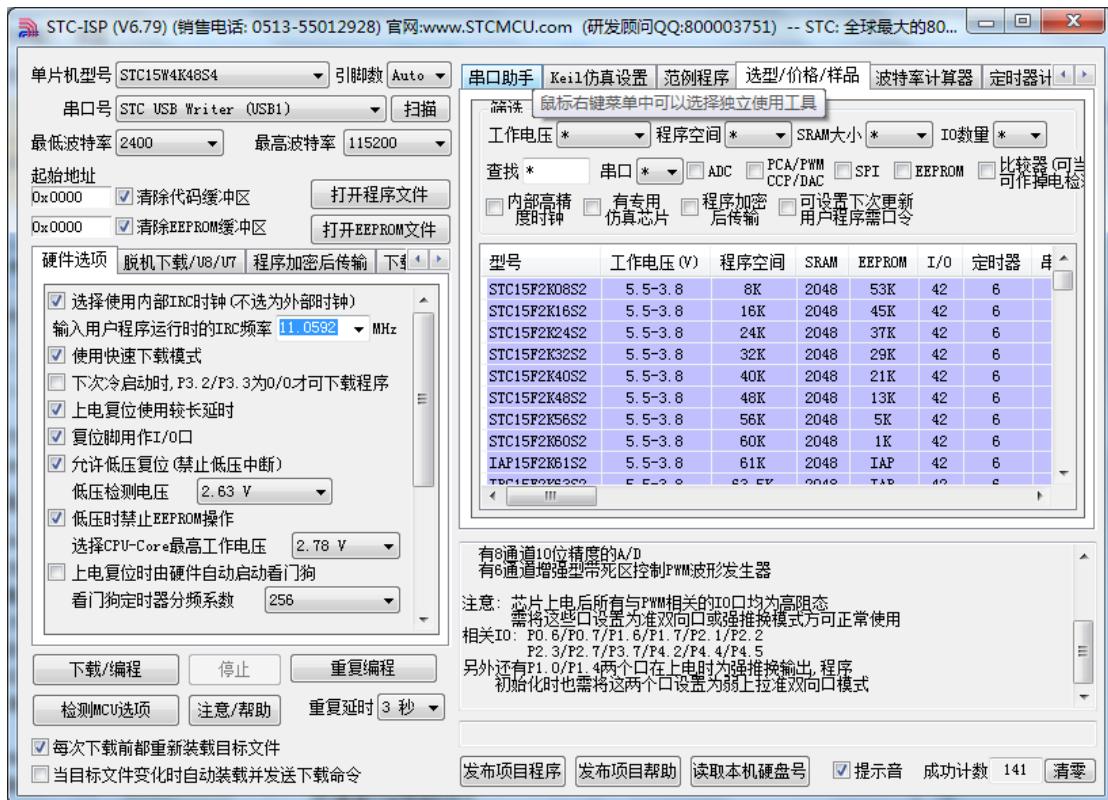
打开 V6.79 版（或者更新的版本）的 STC-ISP 下载软件，下载软件会自动将驱动文件复制到相关的系统目录



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:

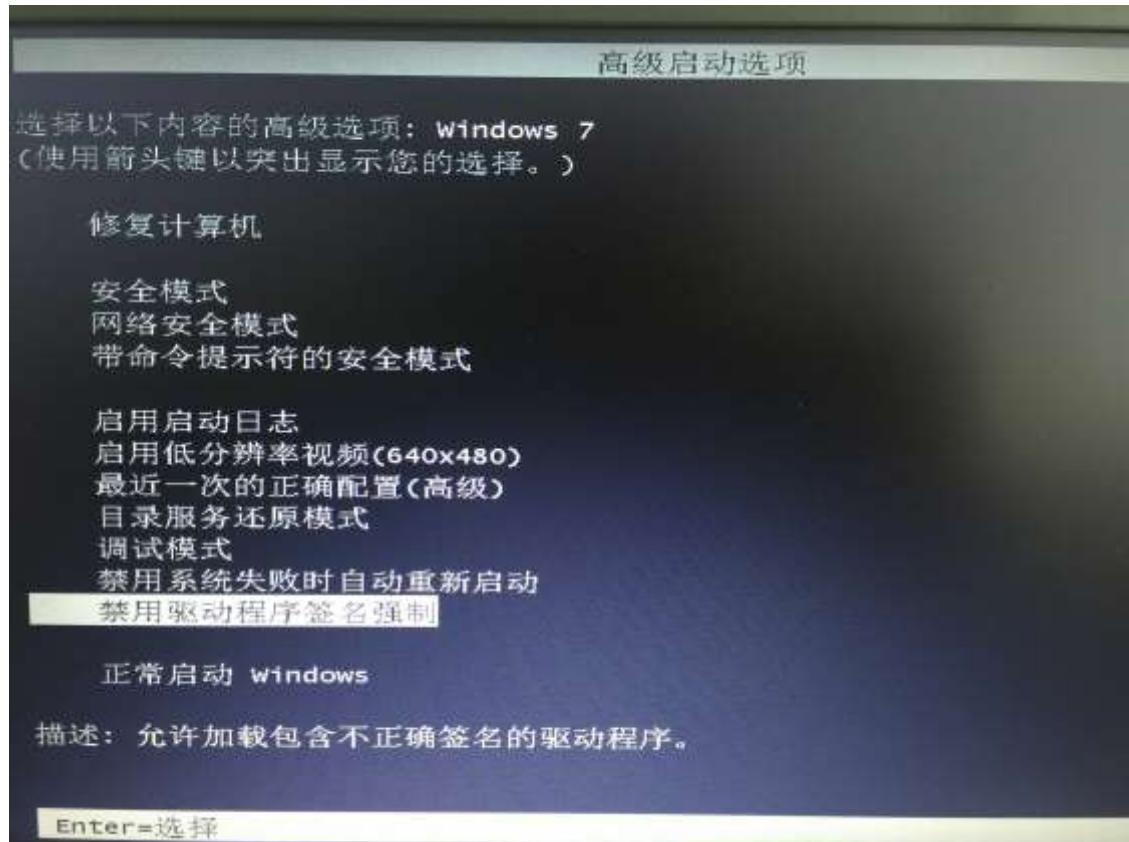


注:若 Windows 7 下,系统并没有自动安装驱动,则驱动的安装方法请参考 [Windows 8 \(32 位\) 的安装方法](#)

Windows 7 (64位) 安装方法

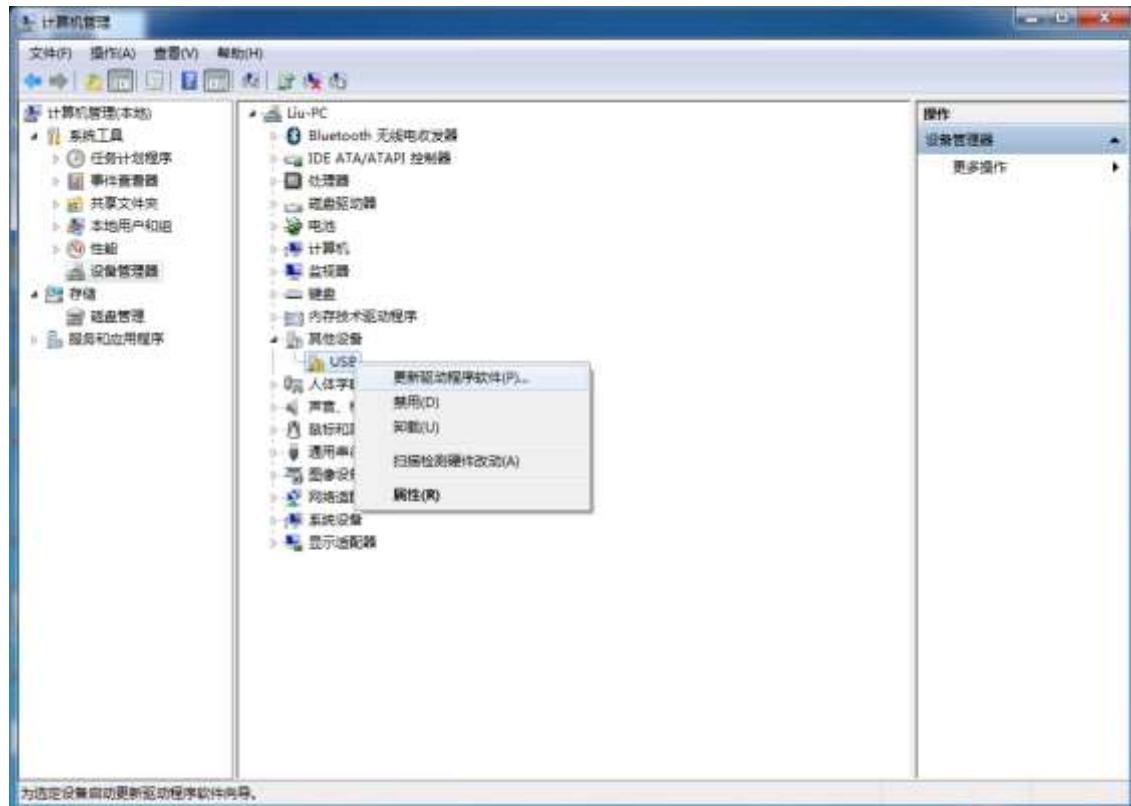
由于 Windows7 64位操作系统在默认状态下，对于没有数字签名的驱动程序是不能安装成功的。所以在安装 STC-USB 驱动前，需要按照如下步骤，暂时跳过数字签名，即可顺利安装成功。

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

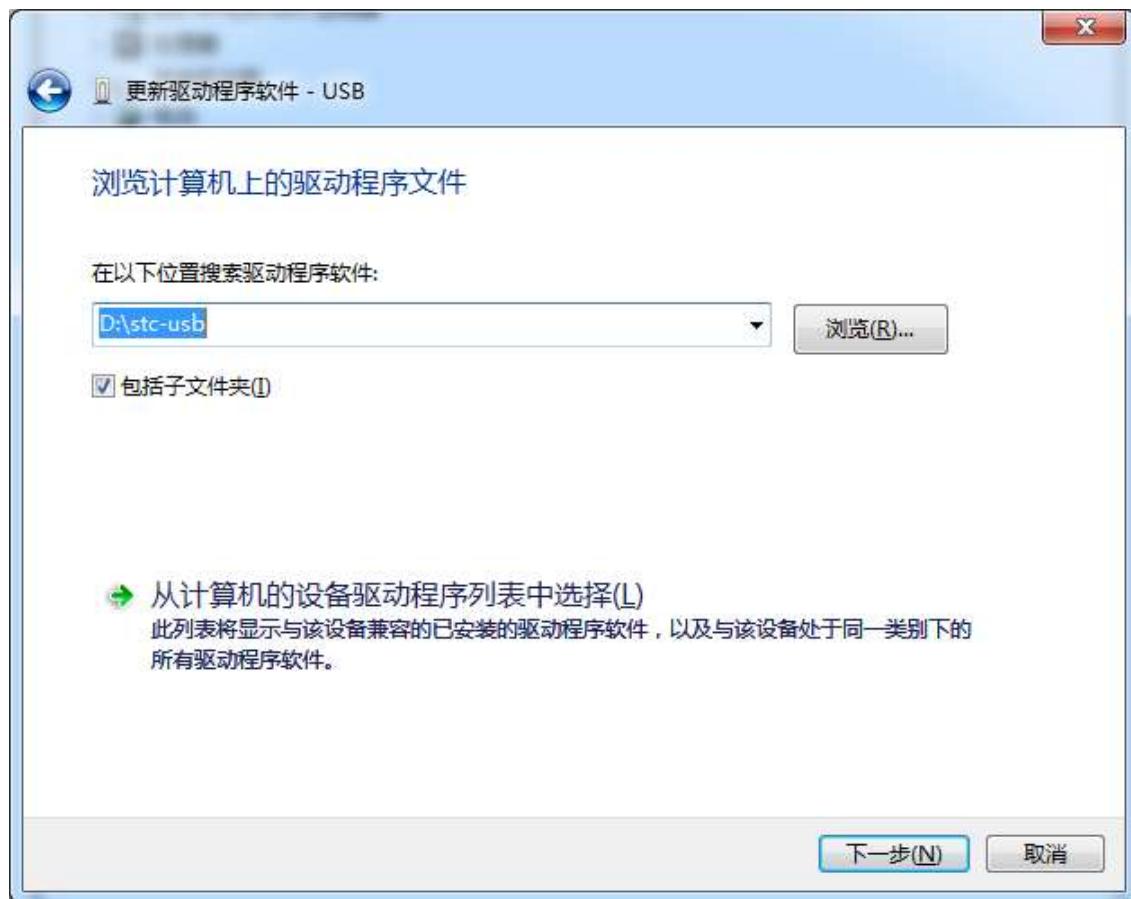
插入 USB 设备，并打开“设备管理器”。找到设备列表中带黄色感叹号的 USB 设备，在设备的右键菜单中，选择“更新驱动程序软件”



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”



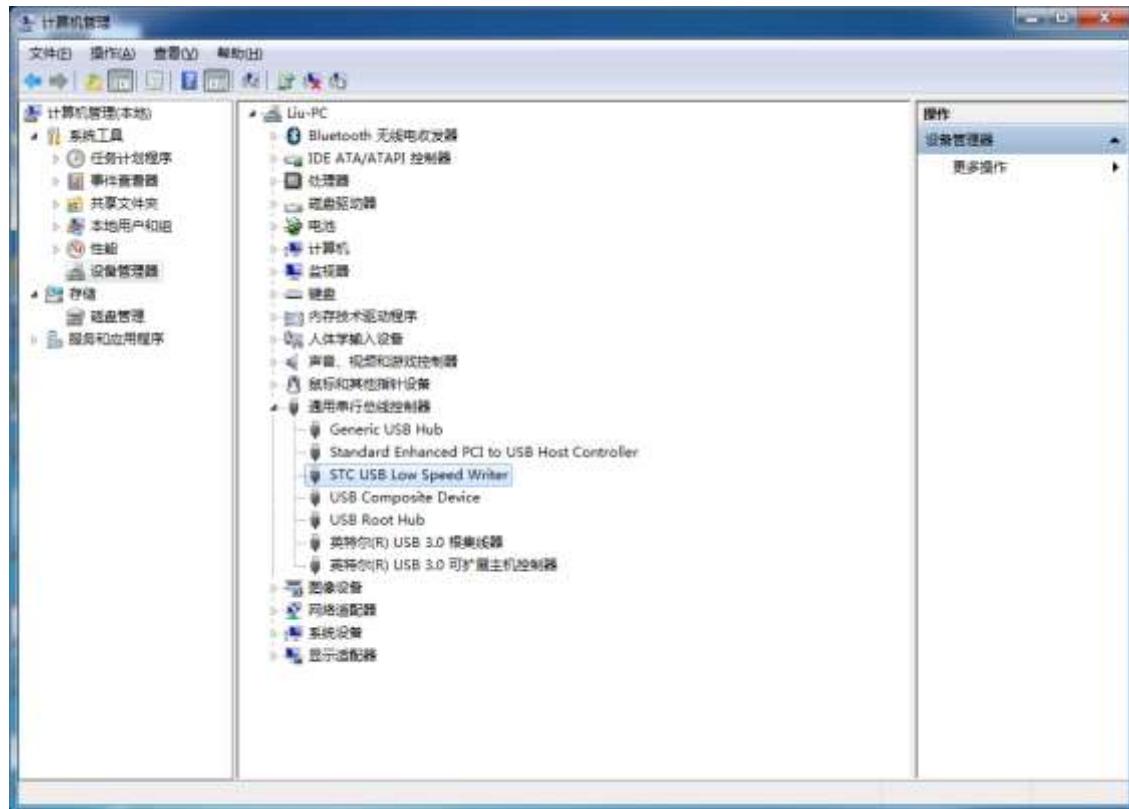
接下来，系统会自动安装驱动，如下图



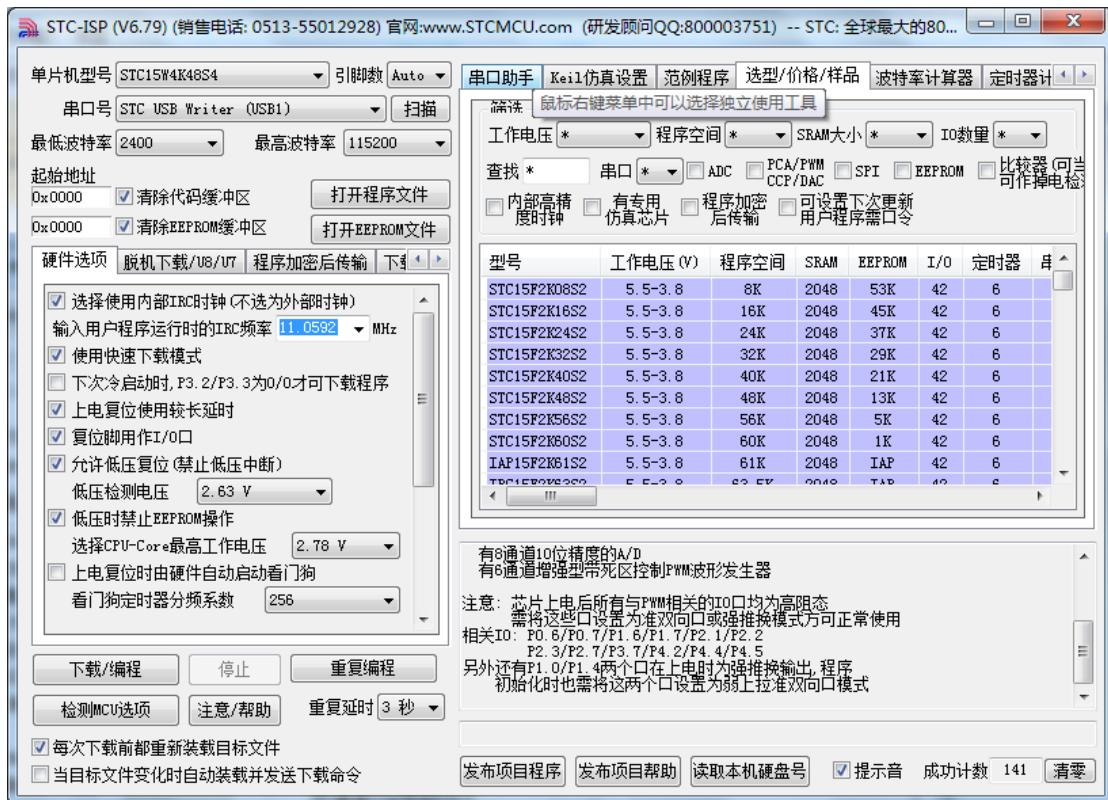
出现下面的对话框表示驱动安装完成



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



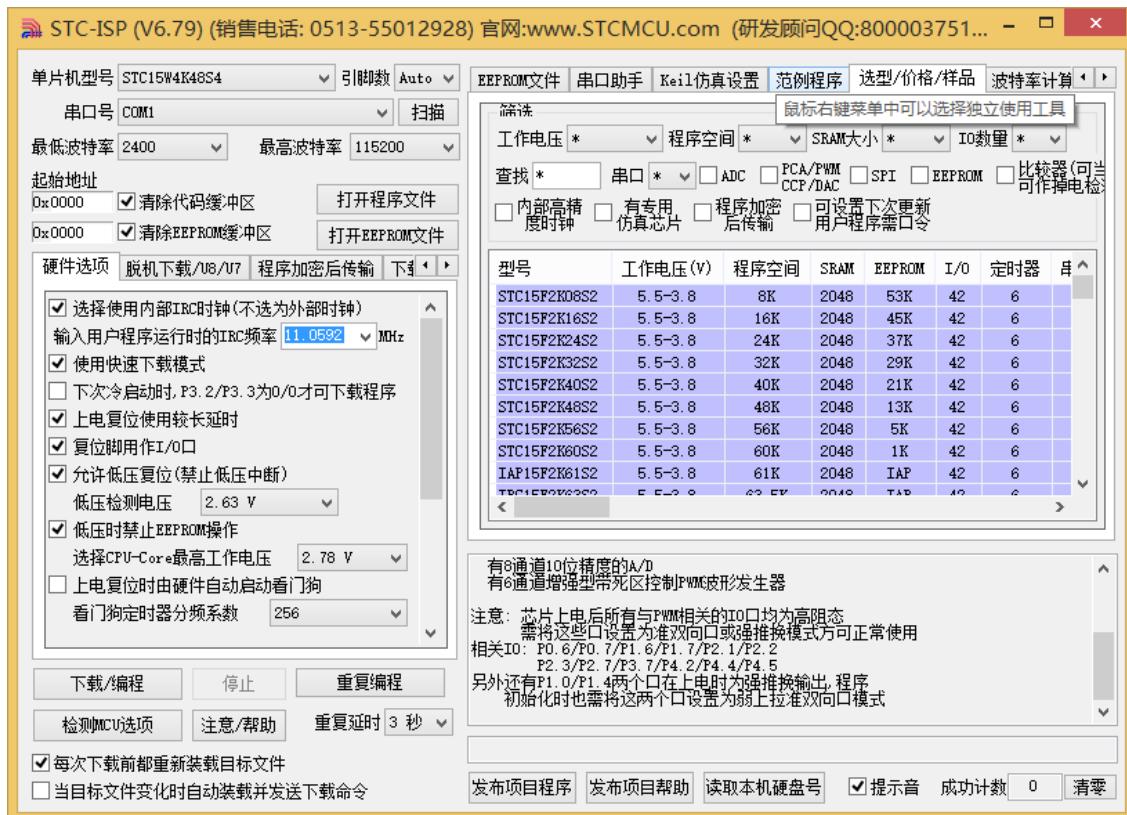
在之前打开的 STC-ISP 下载软件中的串口号列表会自动选择所插入的 USB 设备，并显示设备名称为“STC USB Writer (USB1)”，如下图：



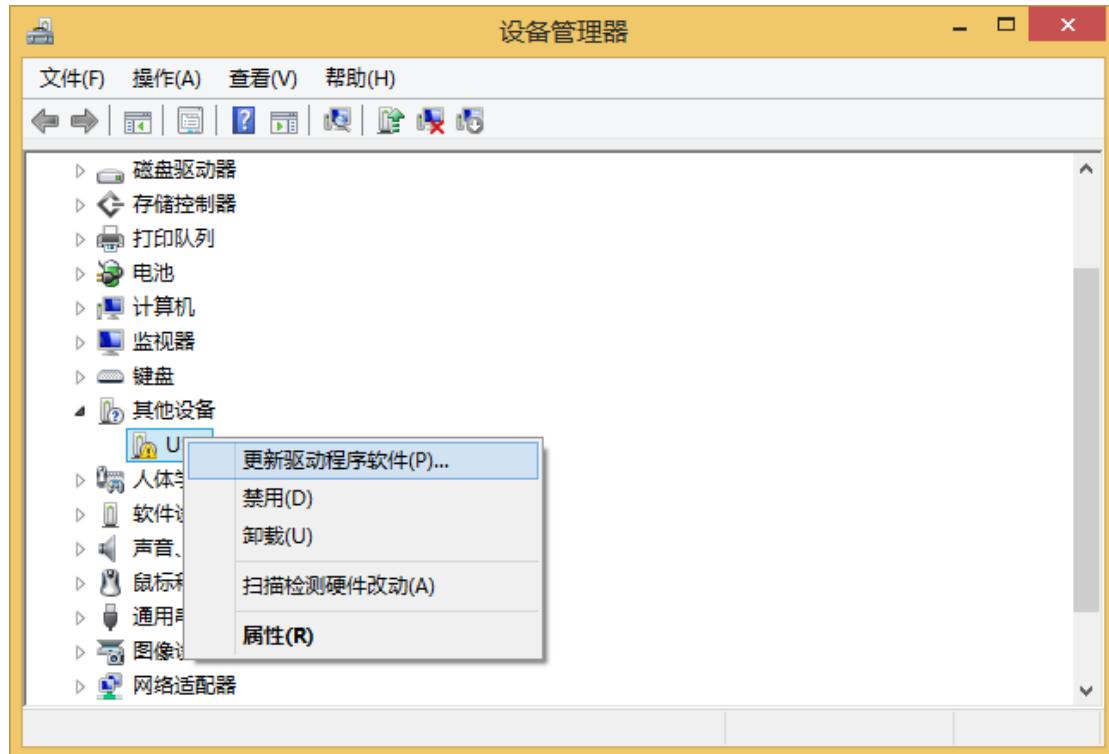
Windows 8 (32位) 安装方法

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

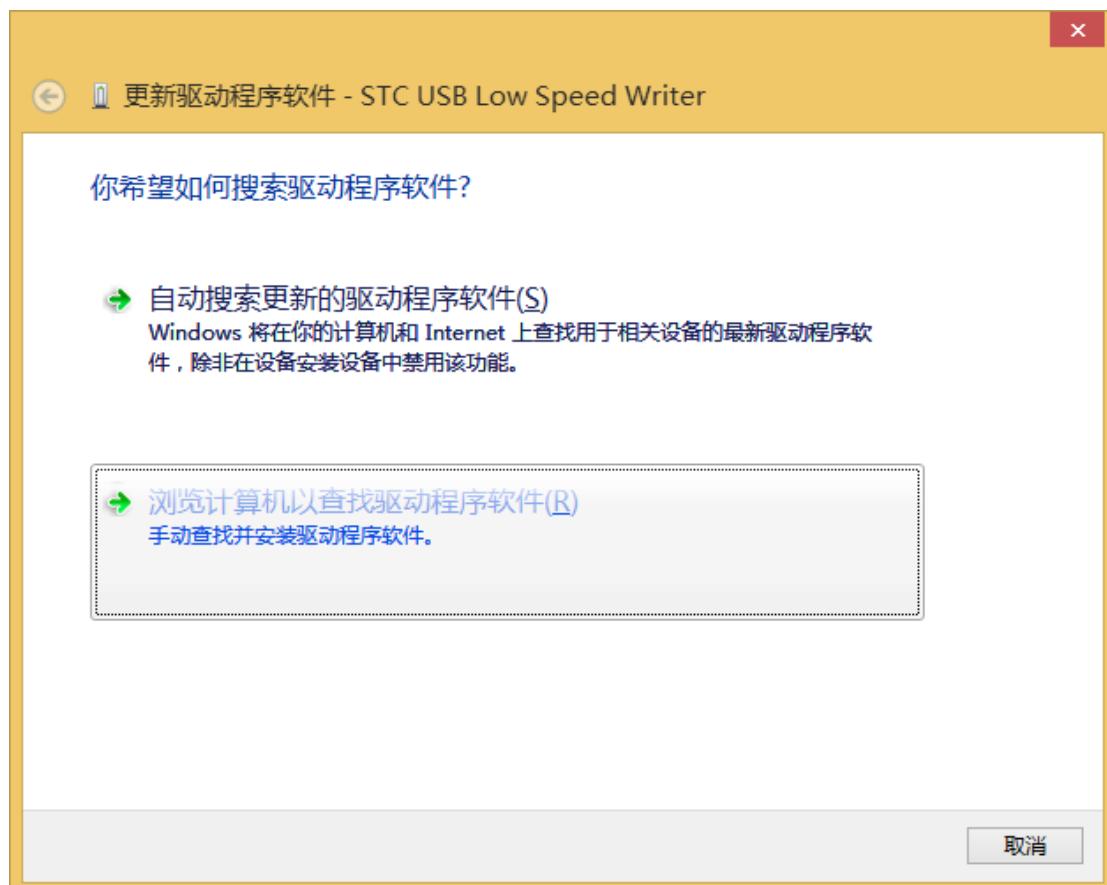
“stc-isp-15xx-v6.79.zip”（或更新版本），下载后解压到本地磁盘，则 STC-USB 的驱动文件也会被解压到当前解压目录中的“STC-USB Driver”中（例如将下载的压缩文件“stc-isp-15xx-v6.79.zip”解压到“F:\”，则 STC-USB 驱动程序在“F:\STC-USB Driver”目录中）



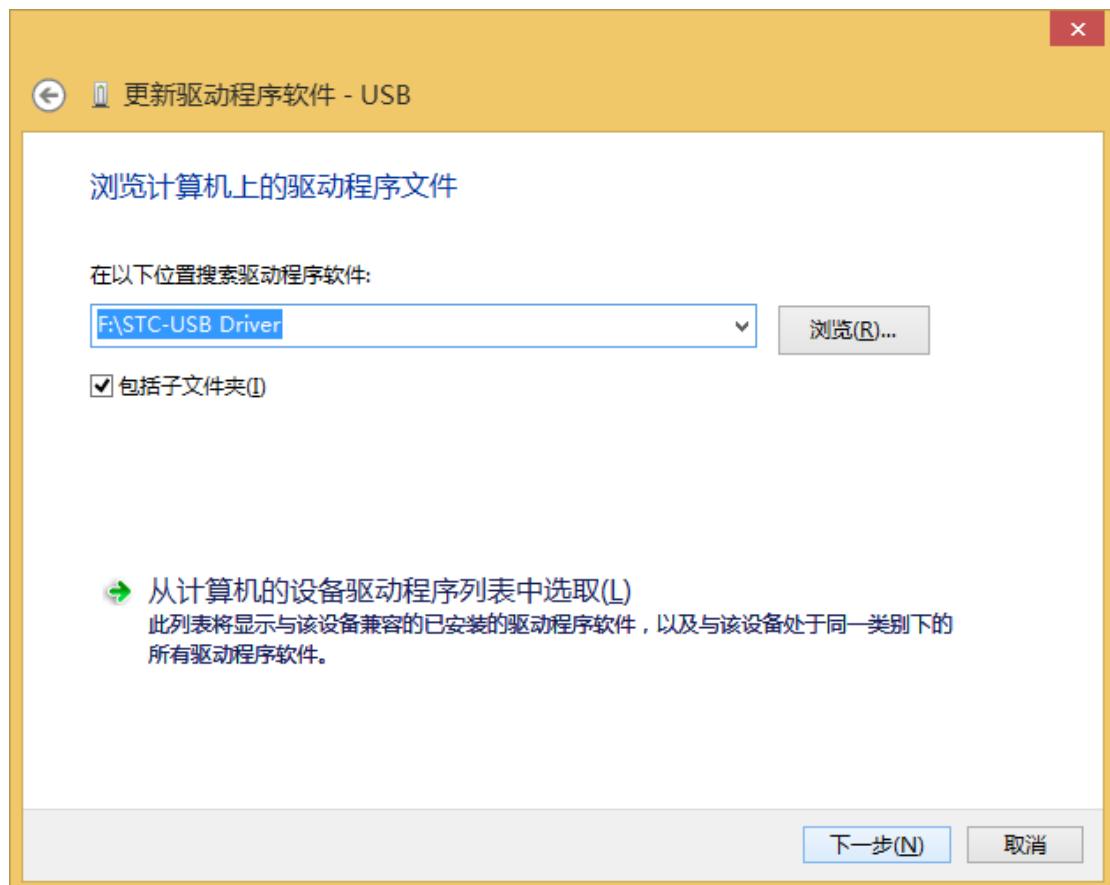
插入 USB 设备，并打开“设备管理器”。找到设备列表中带黄色感叹号的 USB 设备，在设备的右键菜单中，选择“更新驱动程序软件”



在下面的对话框中选择“浏览计算机以查找驱动程序软件”



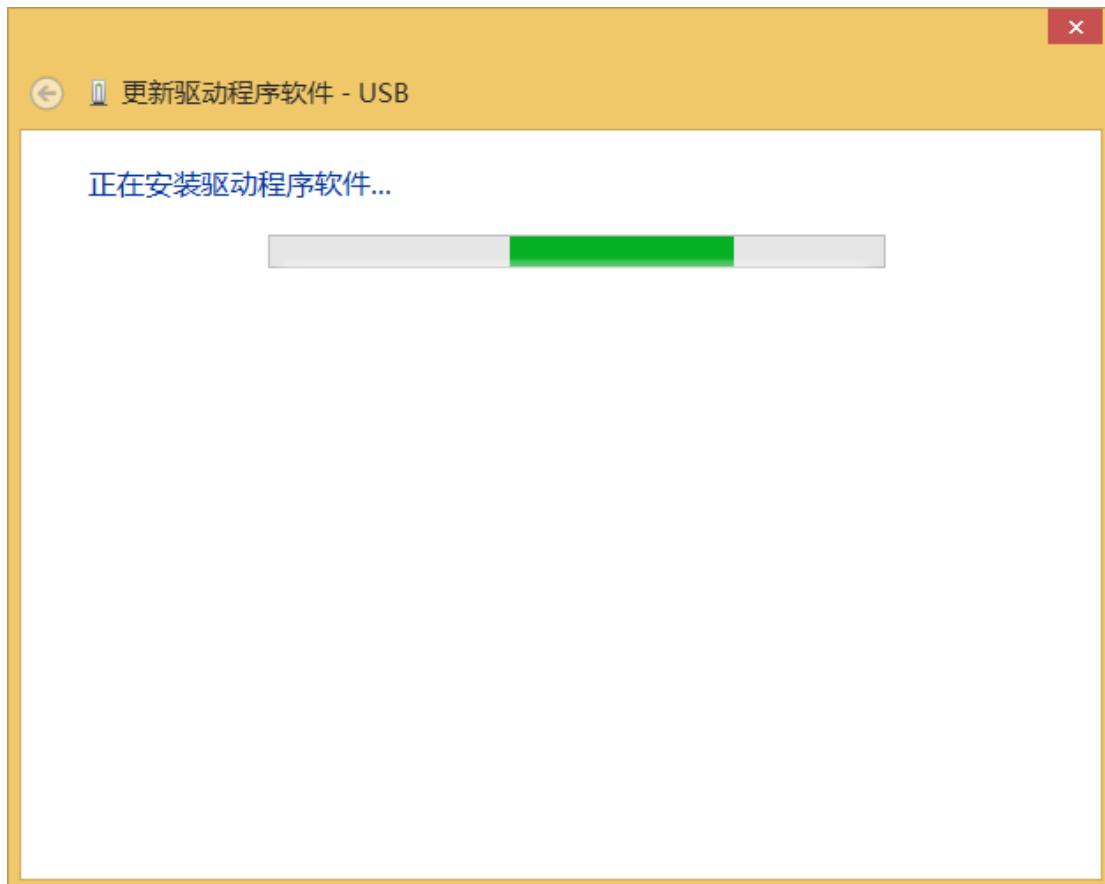
单击下面对话框中的“浏览”按钮，找到之前 STC-USB 驱动程序的存放目录（例如：之前的示例目录为“F:\STC-USB Driver”，用户将路径定位到实际的解压目录）



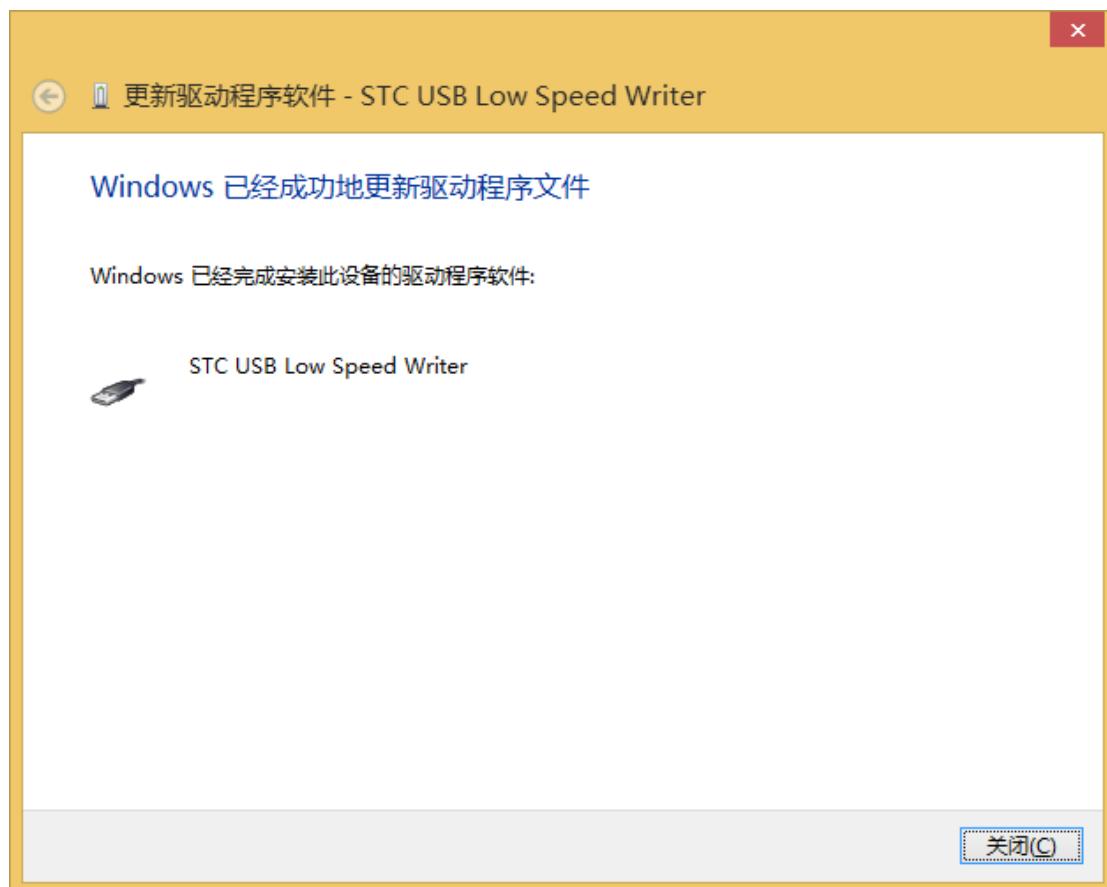
驱动程序开始安装时，会弹出如下对话框，选择“始终安装此驱动程序软件”



接下来，系统会自动安装驱动，如下图



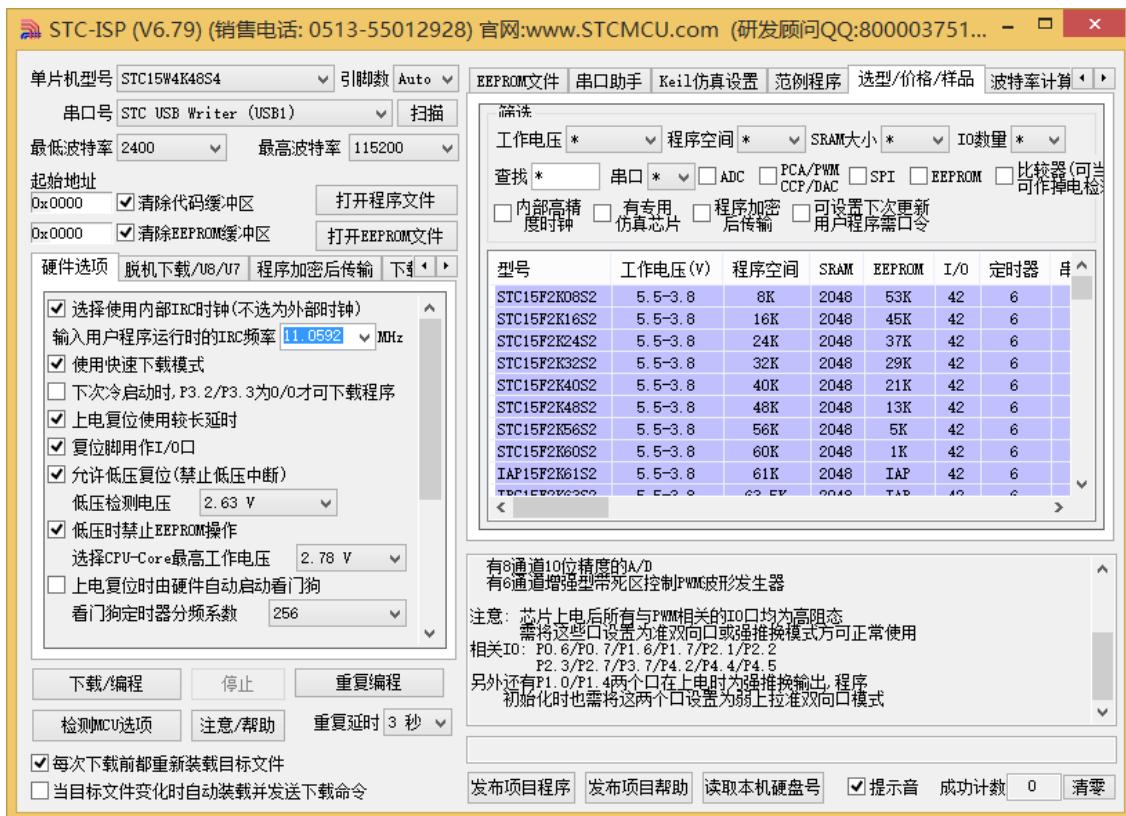
出现下面的对话框表示驱动安装完成



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：



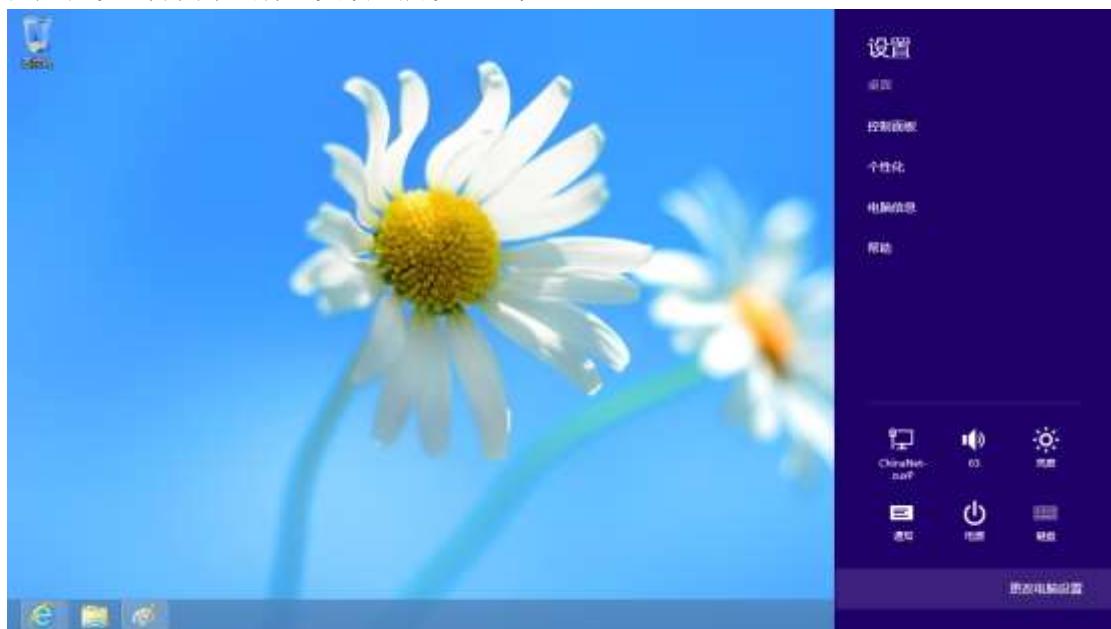
Windows 8 (64 位) 安装方法

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



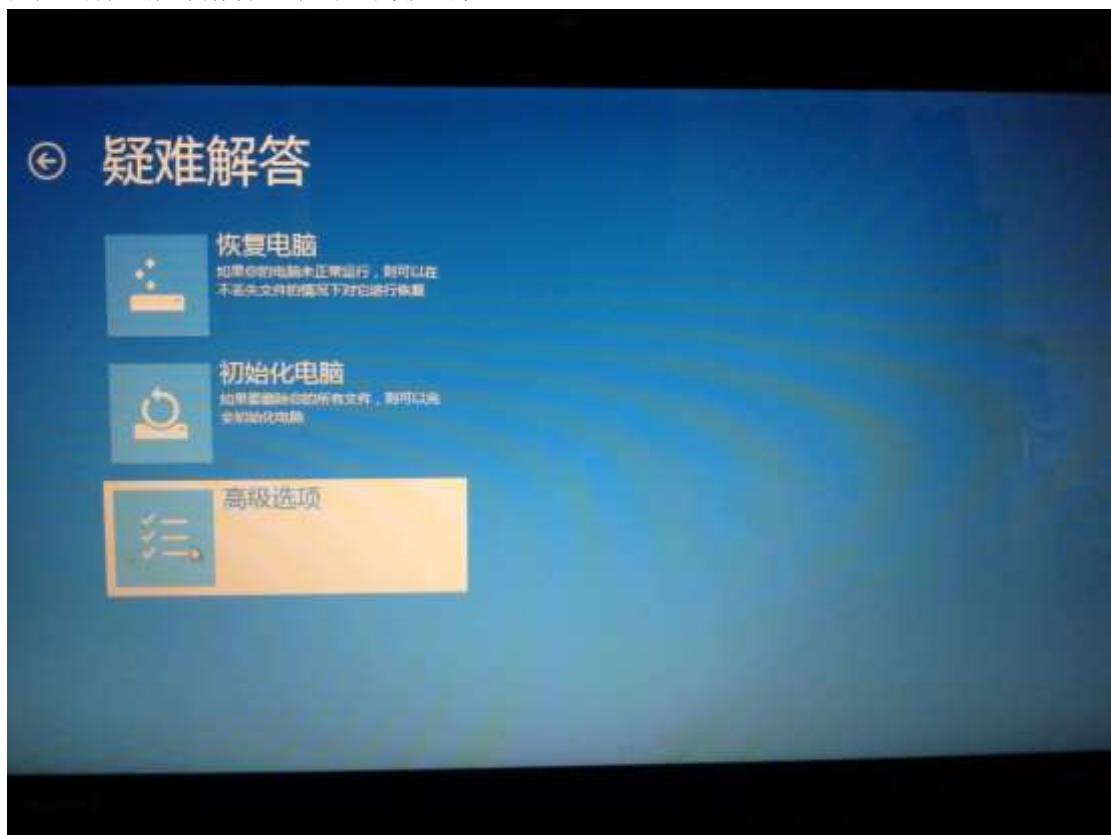
在电脑设置中，选择“常规”属性页中“高级启动”项下面的“立即启动”按钮



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”



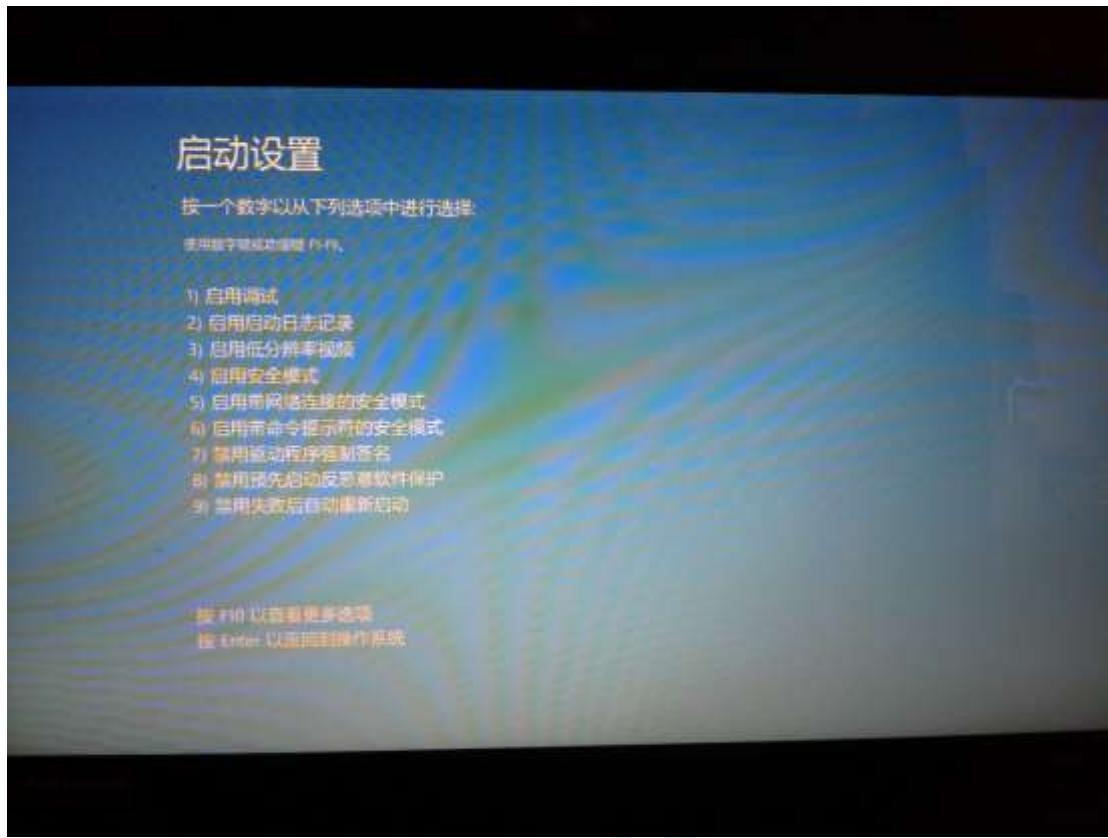
在下面的“高级选项”界面中，选择“启动设置”



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。

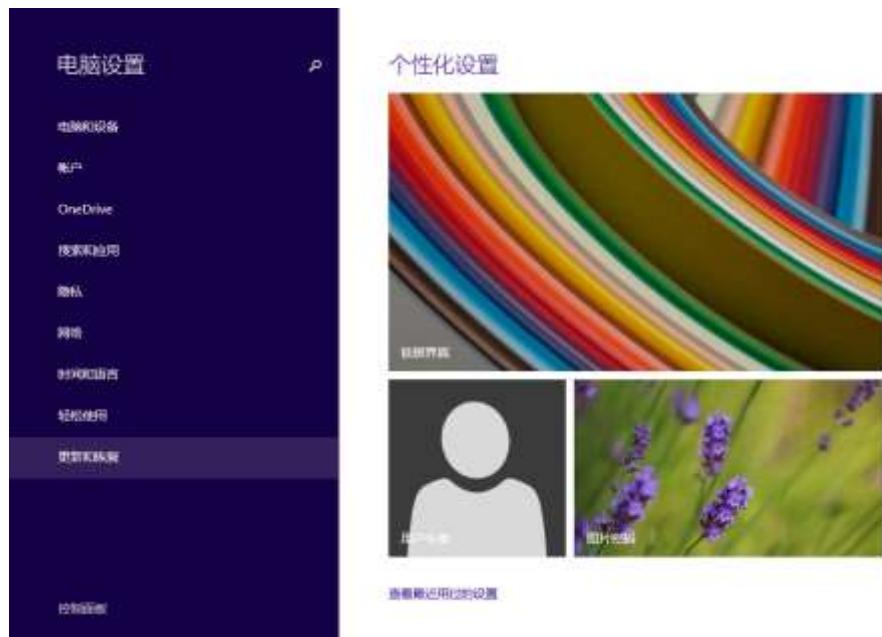
首先将鼠标移动到屏幕的右下角, 选择其中的“设置”按钮



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”）



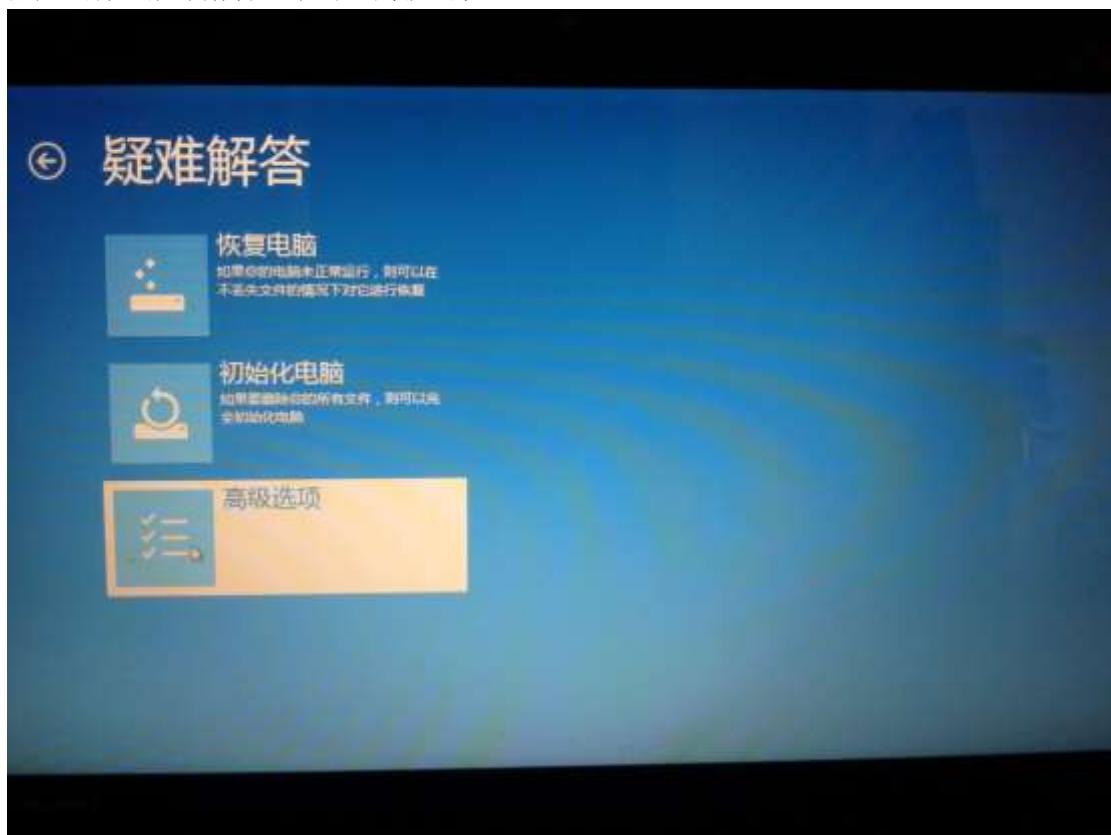
在更新和恢复页面中选择“恢复”属性页，单击“高级启动”项下面的“立即启动”按钮



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”



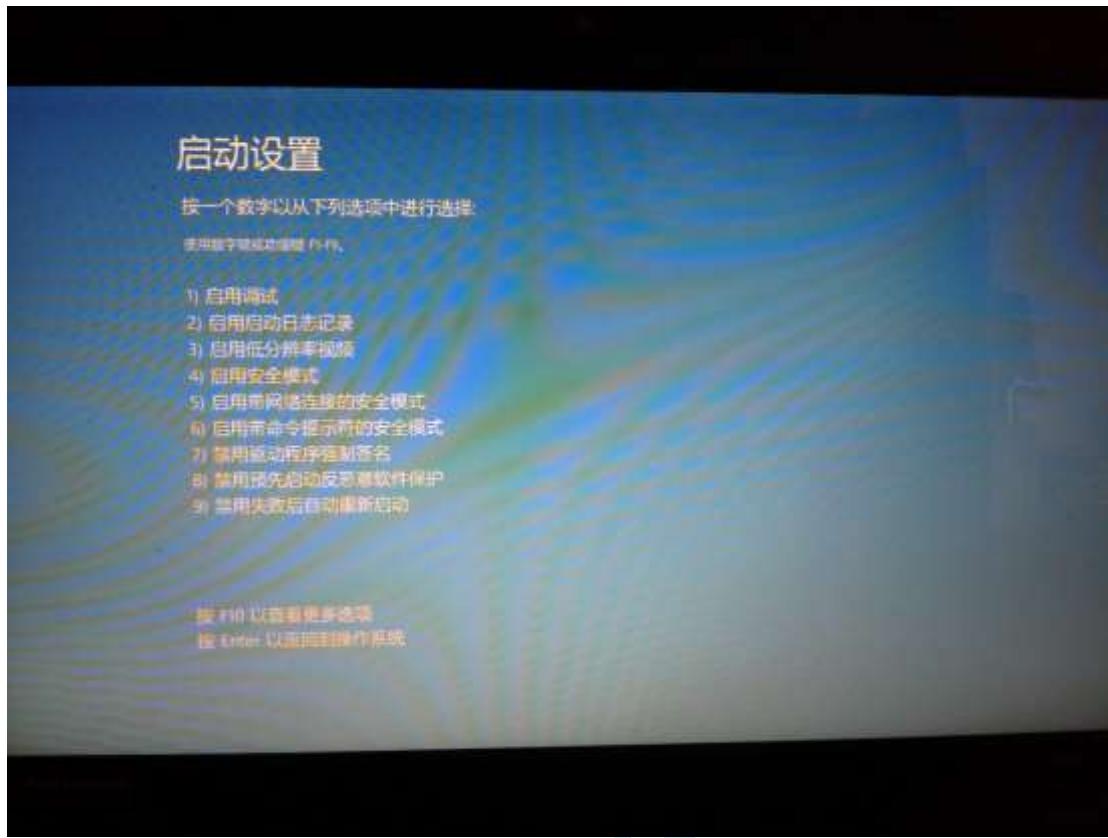
在下面的“高级选项”界面中，选择“启动设置”



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



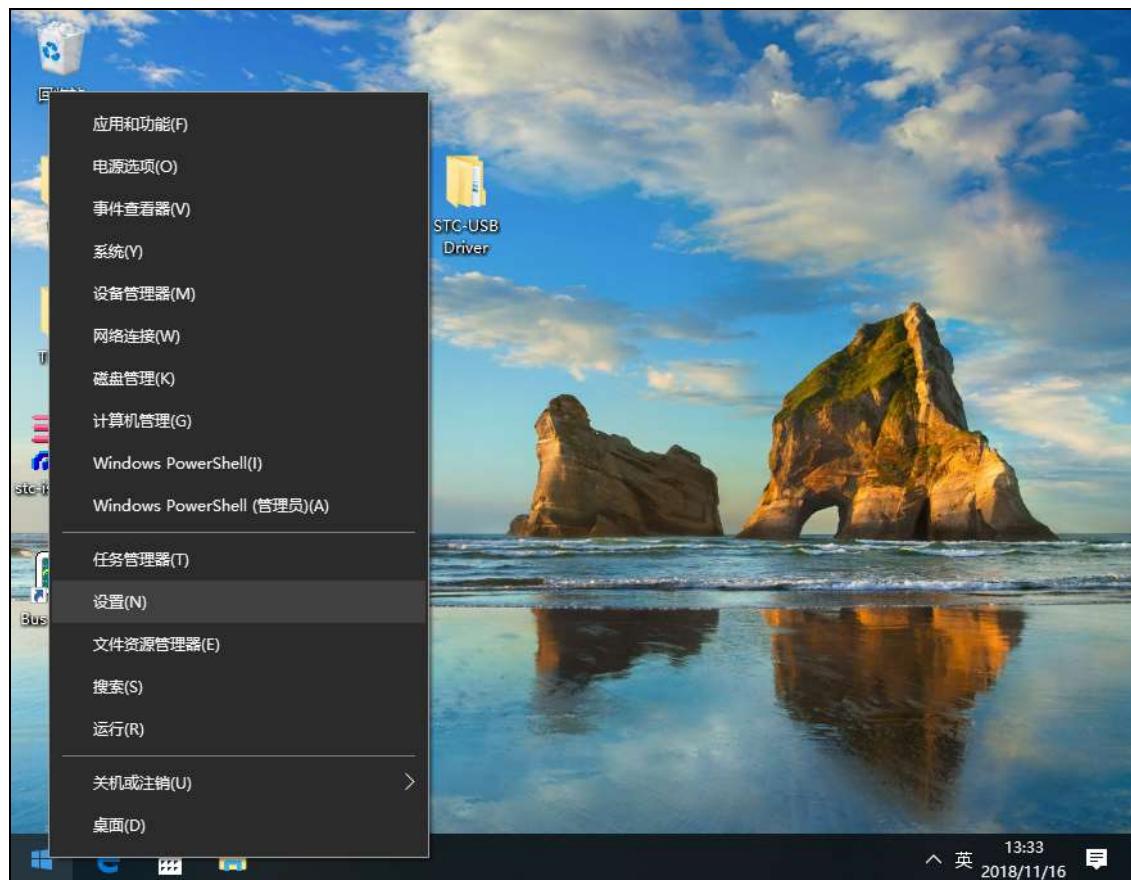
启动到 Windows 8 后，按照 [Windows 8 \(32 位\) 的安装方法](#)即可完成驱动的安装

Windows10 (64位) 安装方法

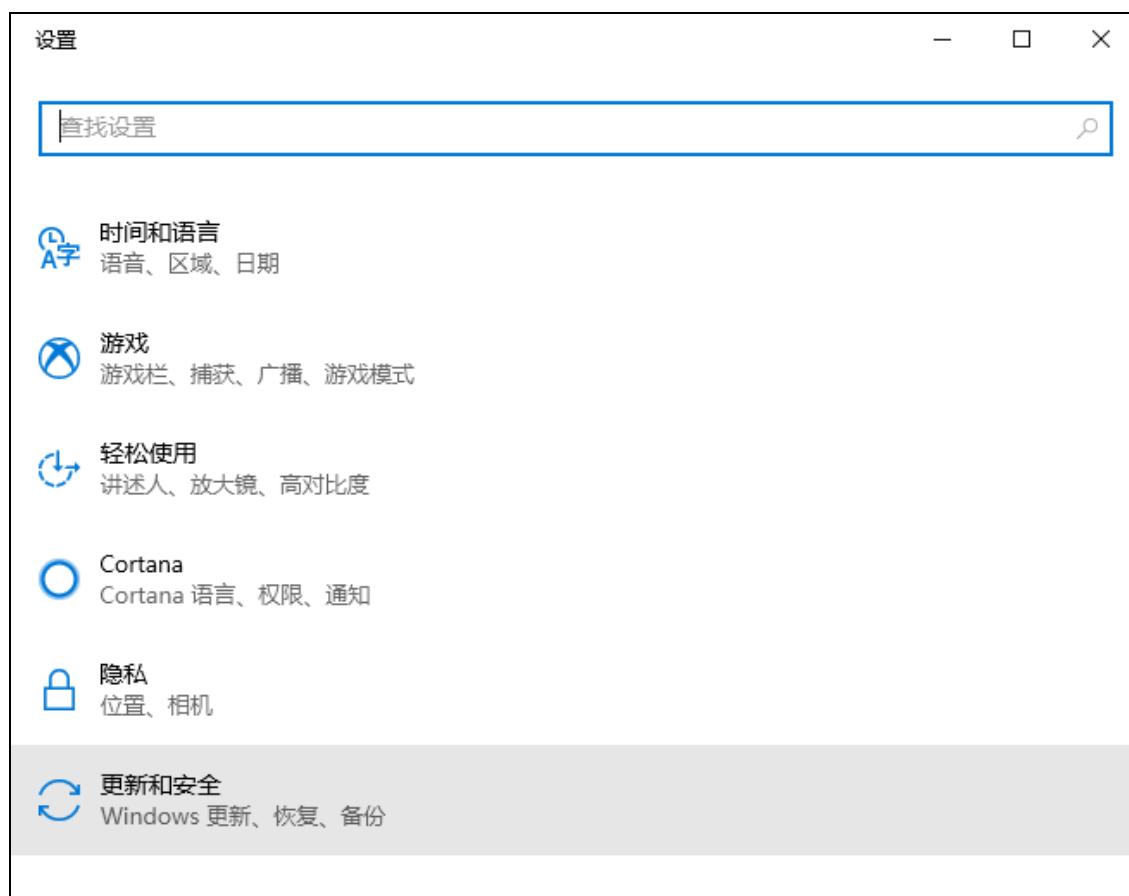
由于 Windows10 64位操作系统在默认状态下，对于没有数字签名的驱动程序是不能安装成功的。所以在安装 STC-USB 驱动前，需要按照如下步骤，暂时跳过数字签名，即可顺利安装成功。

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”



然后选择“查看更多恢复选项”



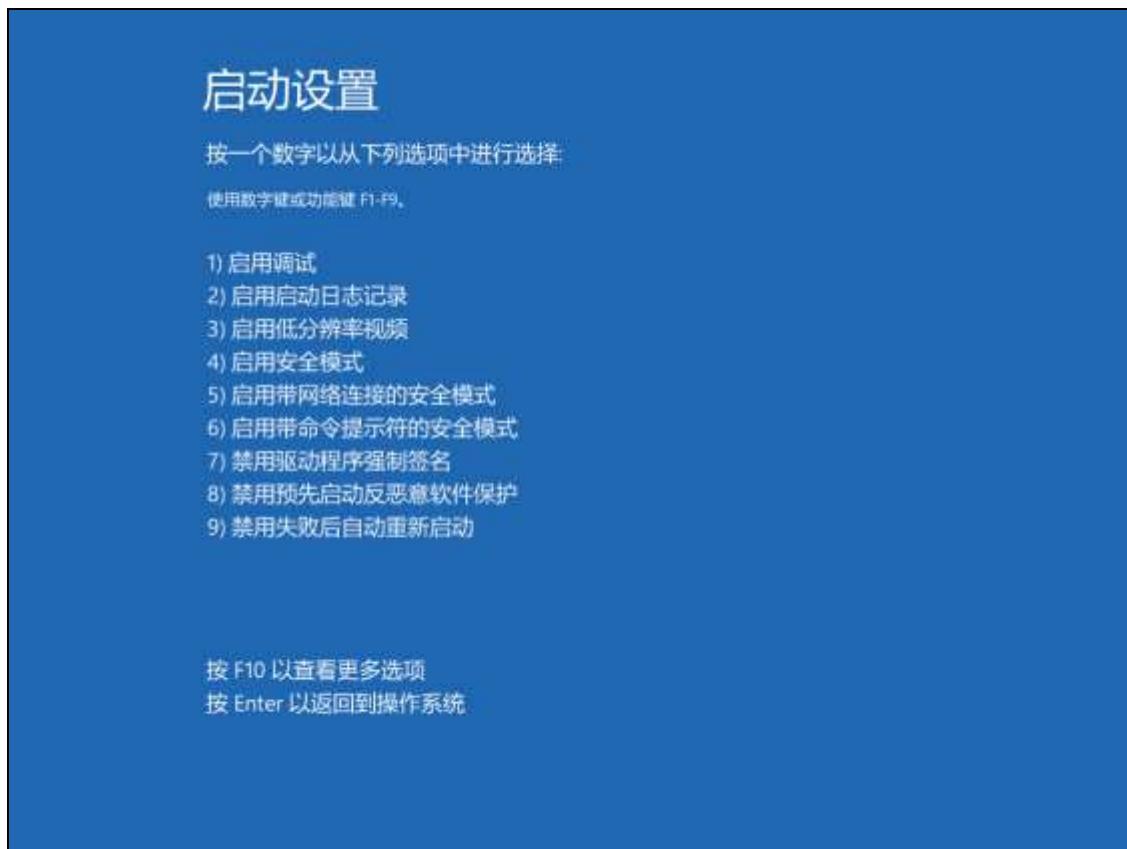
选择“启动设置”项



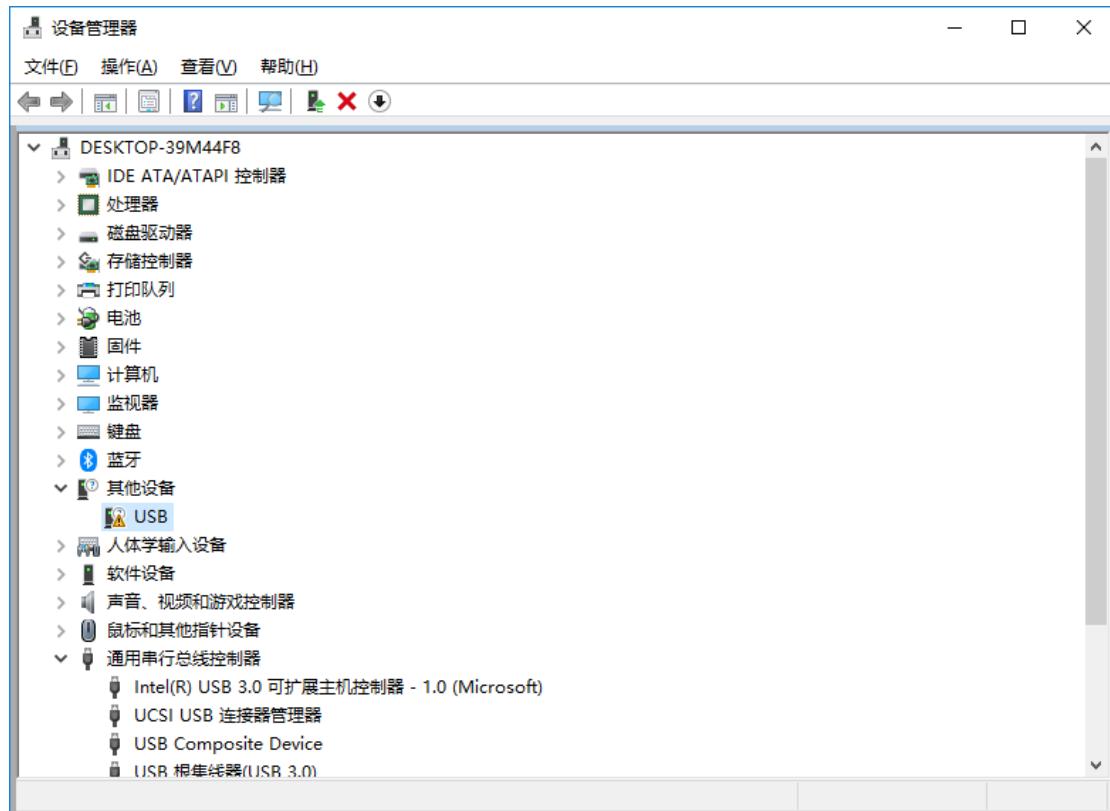
出现如下画面后，点击“重启”按钮重启电脑



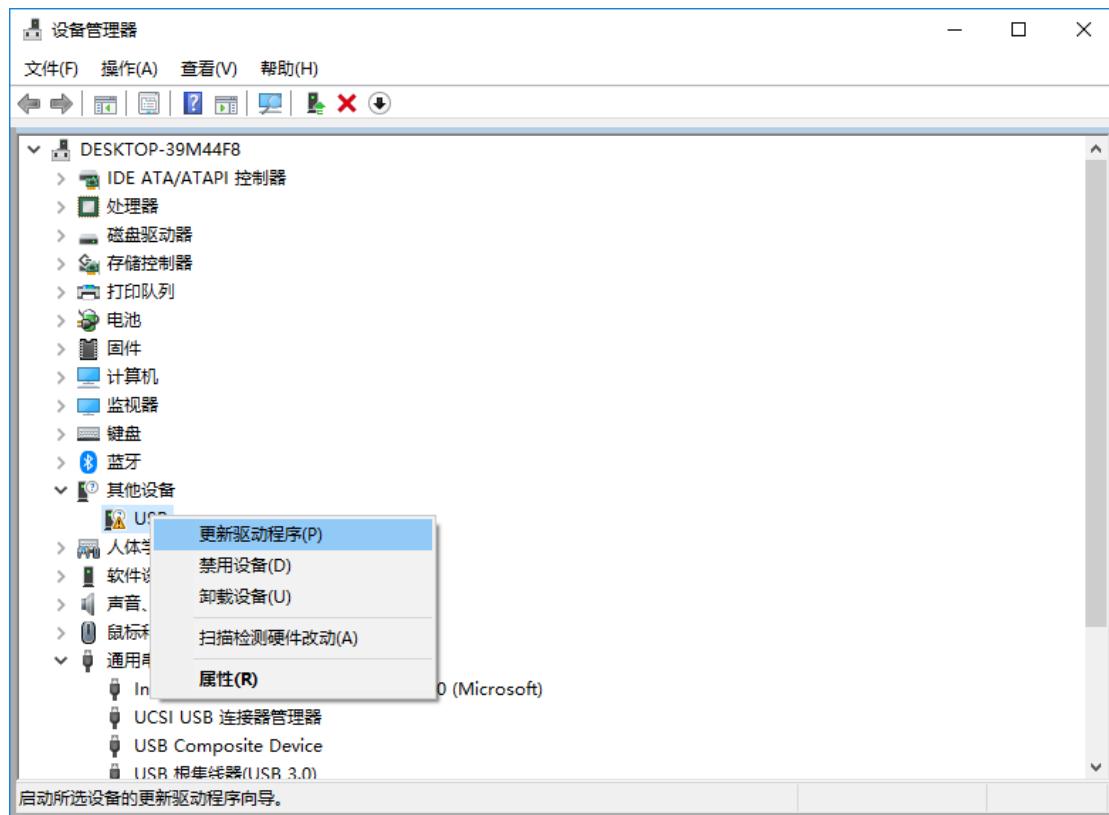
电脑重启后，会弹出“启动设置”界面，按“F7”按钮来选择“禁止驱动程序强制签名”项



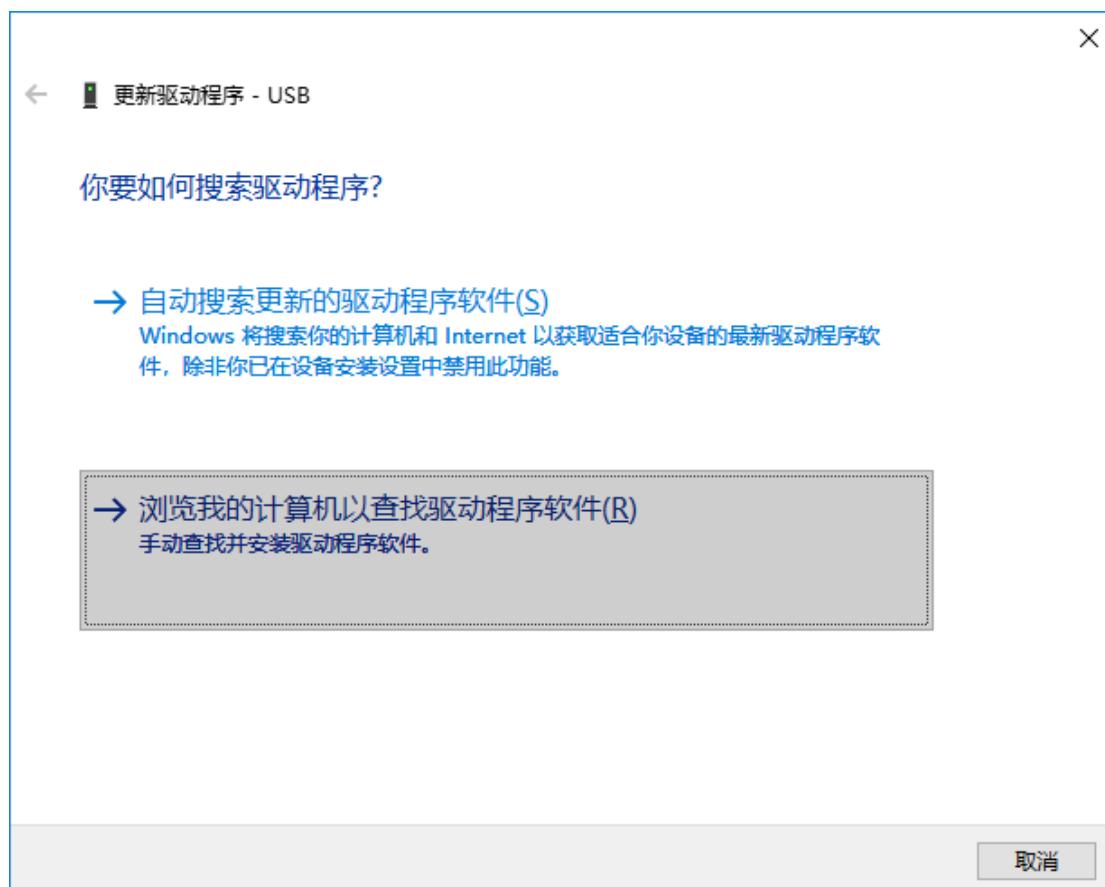
电脑启动完成后，将准备好的芯片用 USB 线与电脑相连，并打开“设备管理器”，此时由于驱动还没有开始安装，所以在设备管理器中会显示为一个带感叹号的未知设备



鼠标右键单击未知设备，选择右键菜单中的“更新驱动程序”



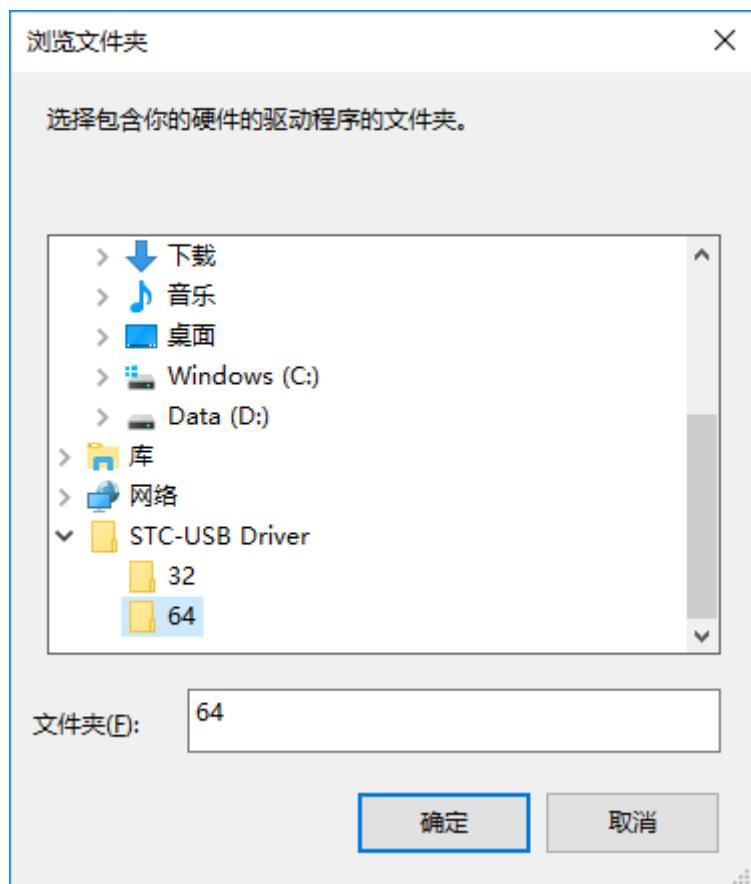
在弹出的驱动安装程序选择画面中，选择“浏览我的计算机以查找驱动程序软件”项



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



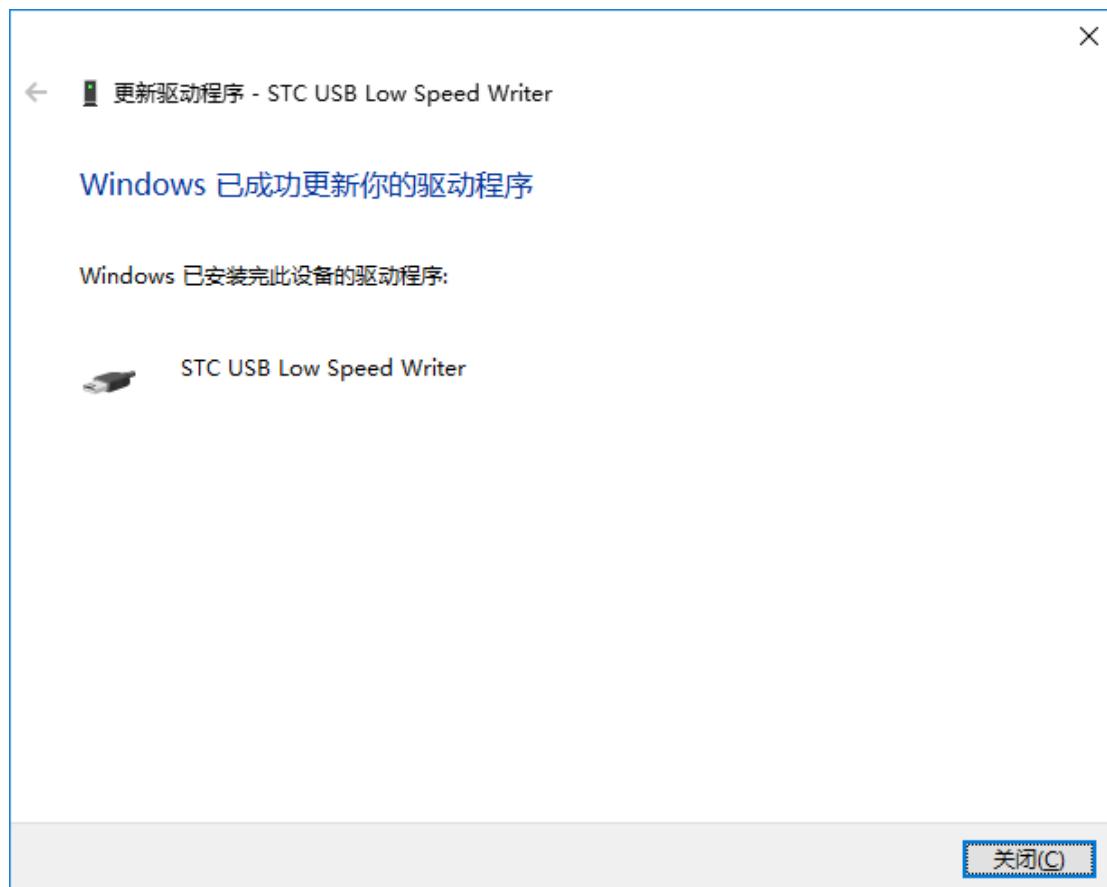
点击“下一步”开始安装驱动



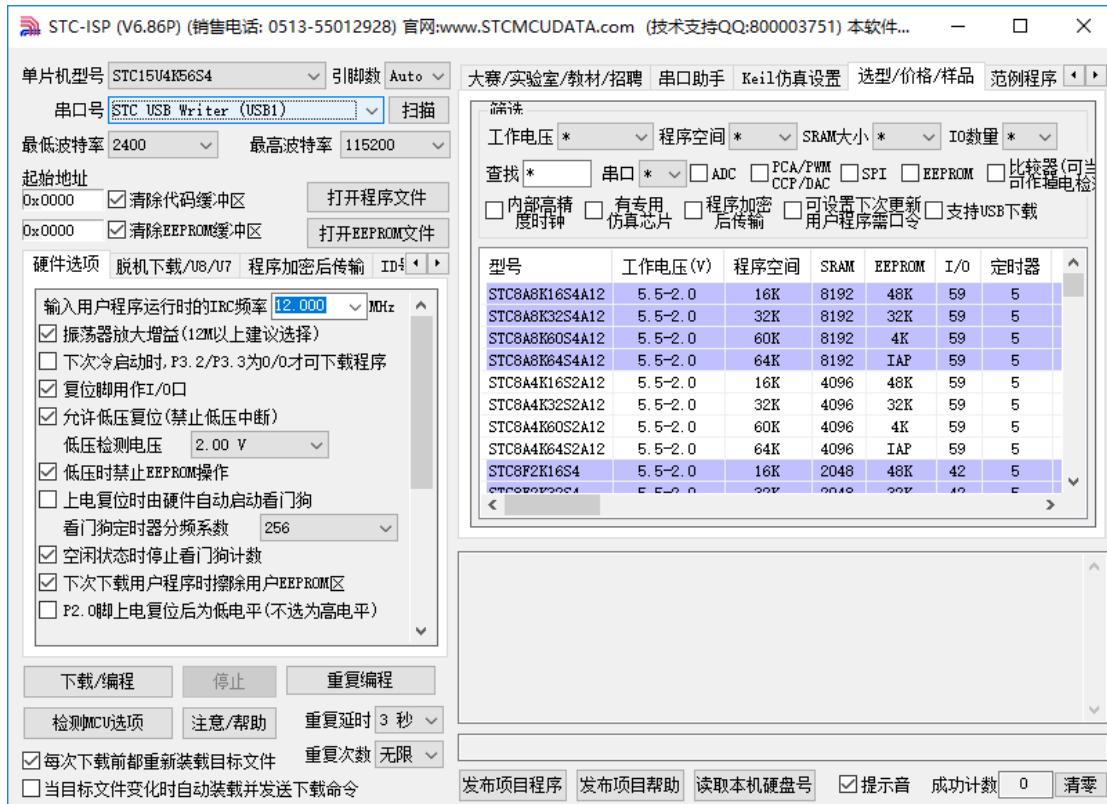
驱动安装的过程中，会弹出如下的警告画面，选择“始终安装此驱动程序软件”



出现下面的画面时，驱动程序就安装成功了

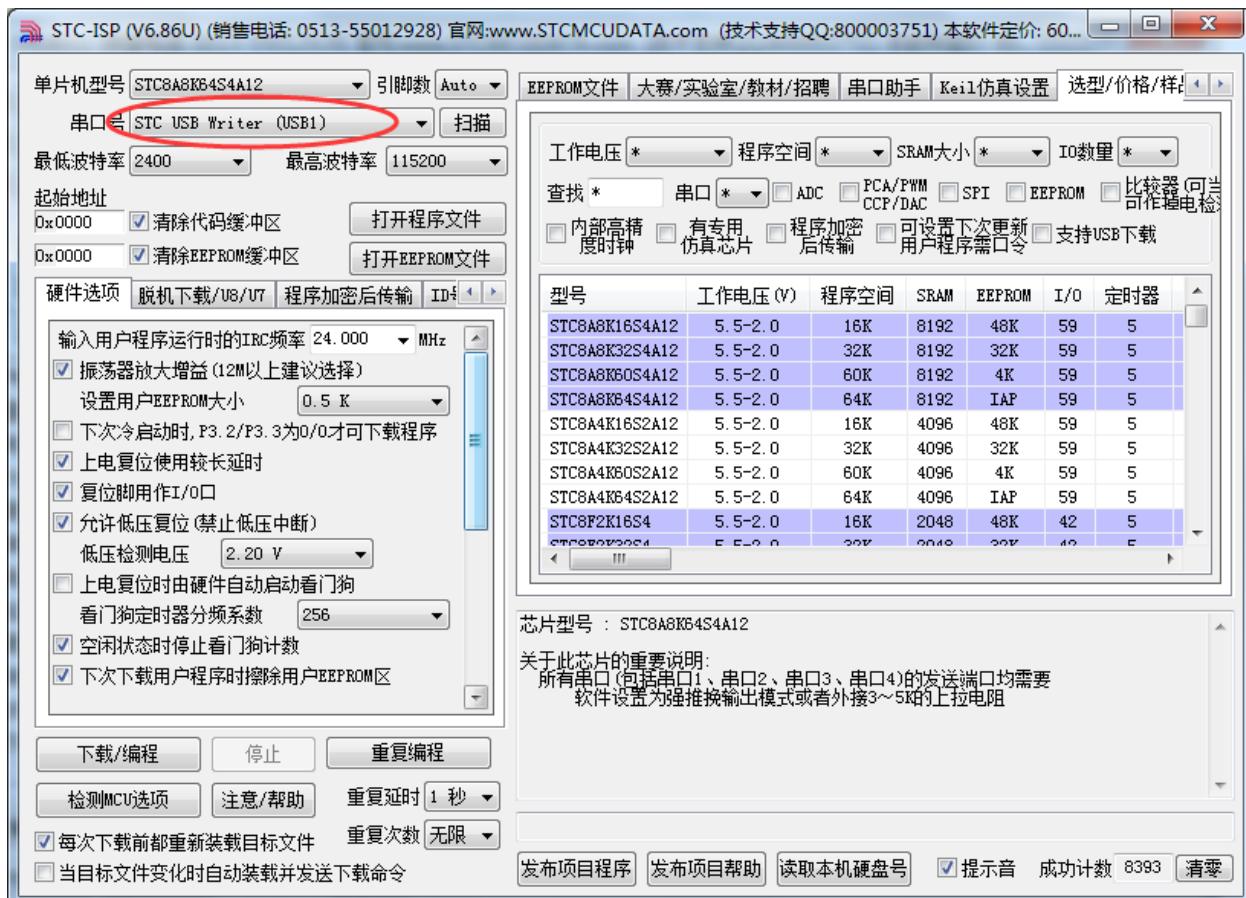


在回到 STC-ISP 的下载软件，此时“串口号”的下拉列表中已自动选择了“STC USB Writer (USB1)”，即可使用 USB 进行 ISP 下载了

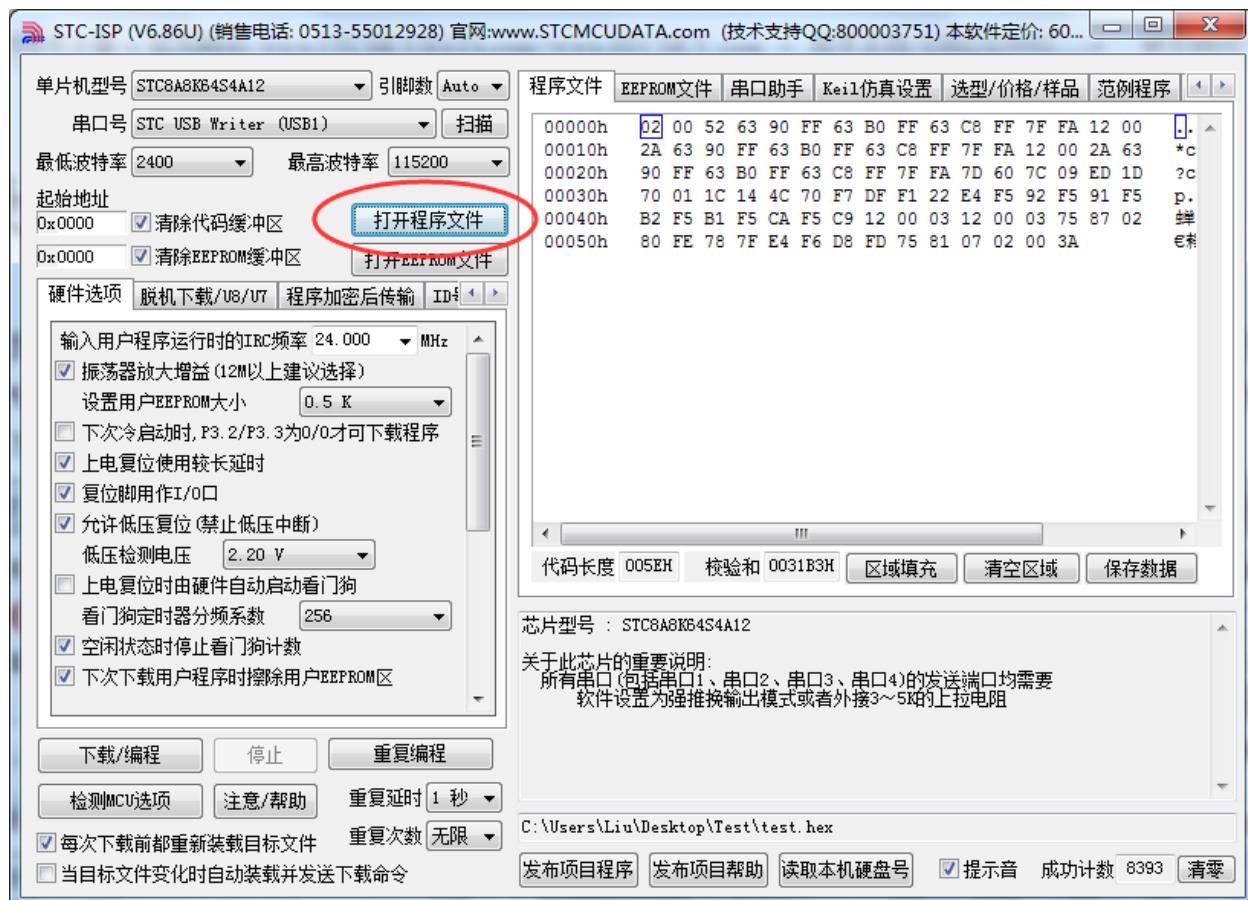


附录F USB 下载步骤演示

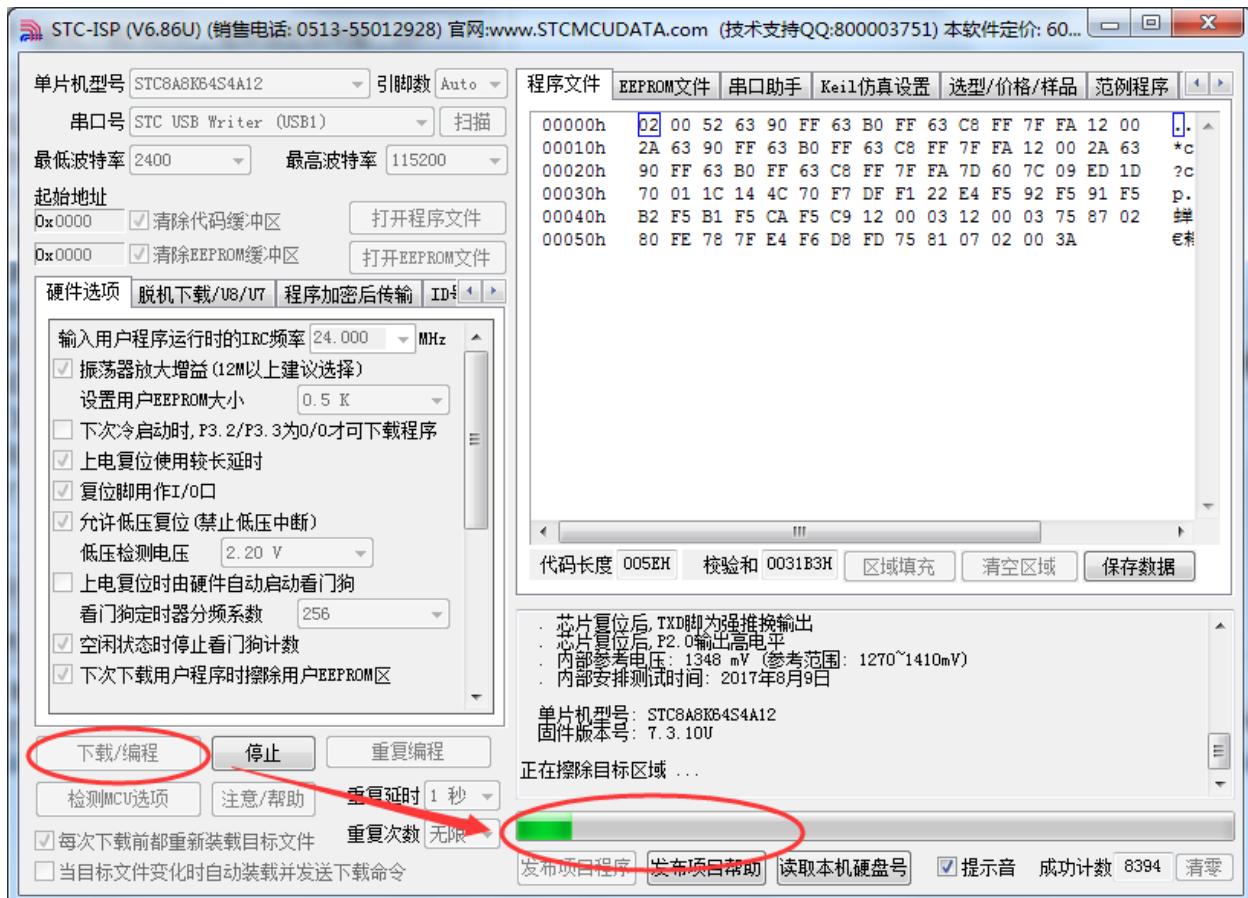
1、首先参考 P5.1.5 章的应用线路图连接好单片机，并将目标芯片的 P3.2 口连接到 Gnd，然后将系统连接到 PC 端的 USB 端口上。打开 ISP 下载软件，即可在下载软件的串口号中自动搜索到“STC USB Writer (USB1)”的 USB 设备



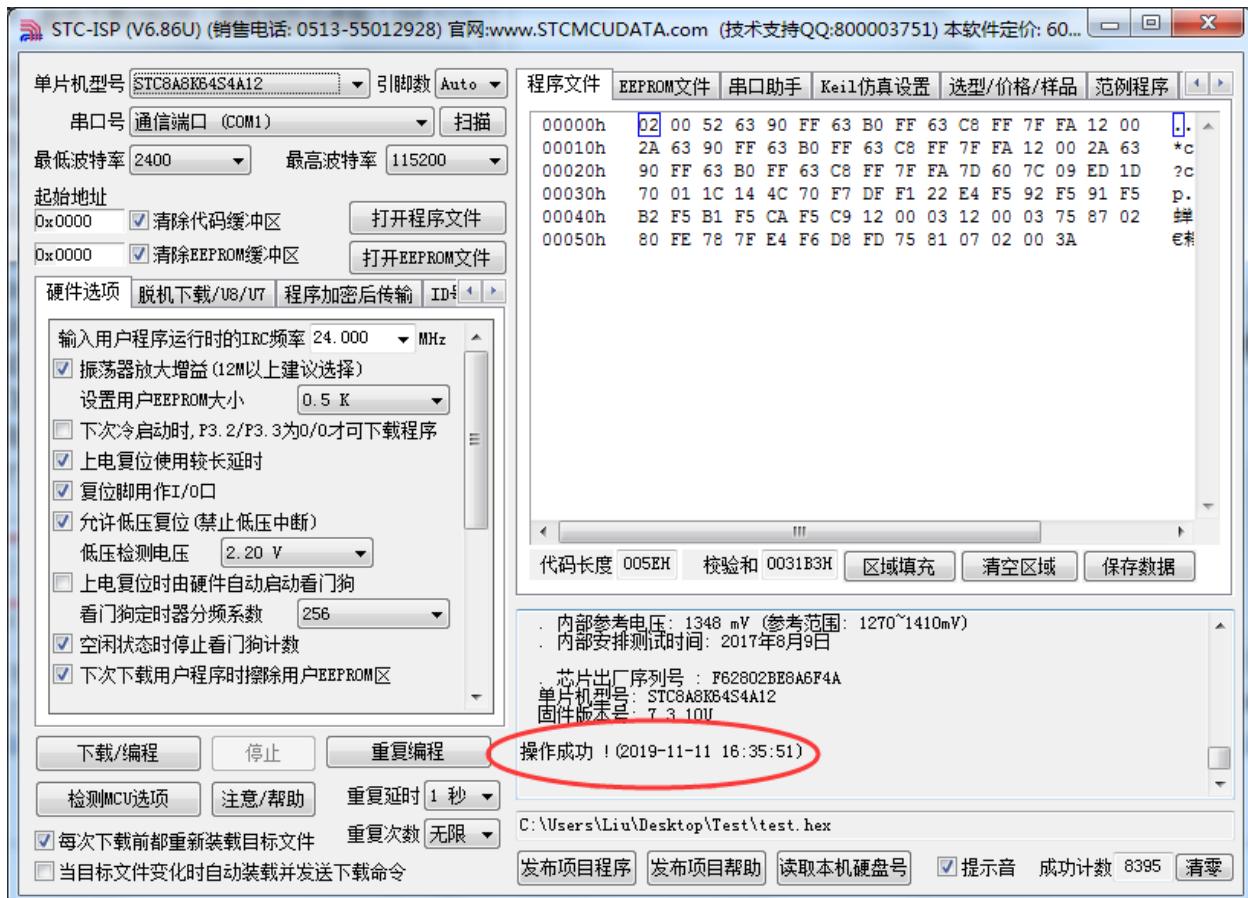
2、打开用户代码程序



3、点击“下载/编程”按钮开始下载用户代码

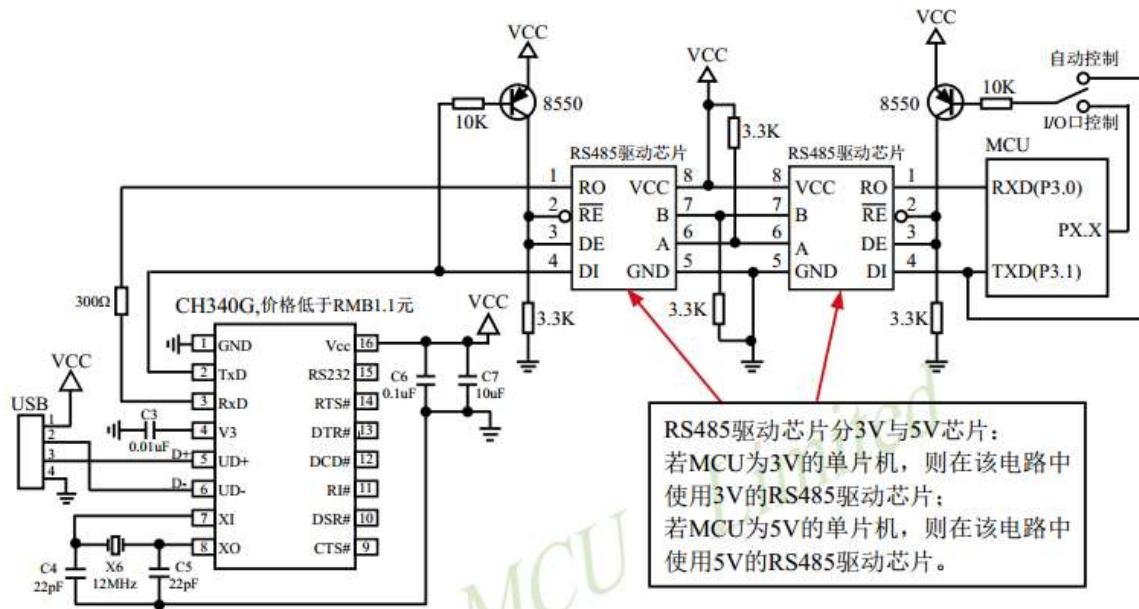


4、直到提示“操作成功”，表示程序代码下载完成。

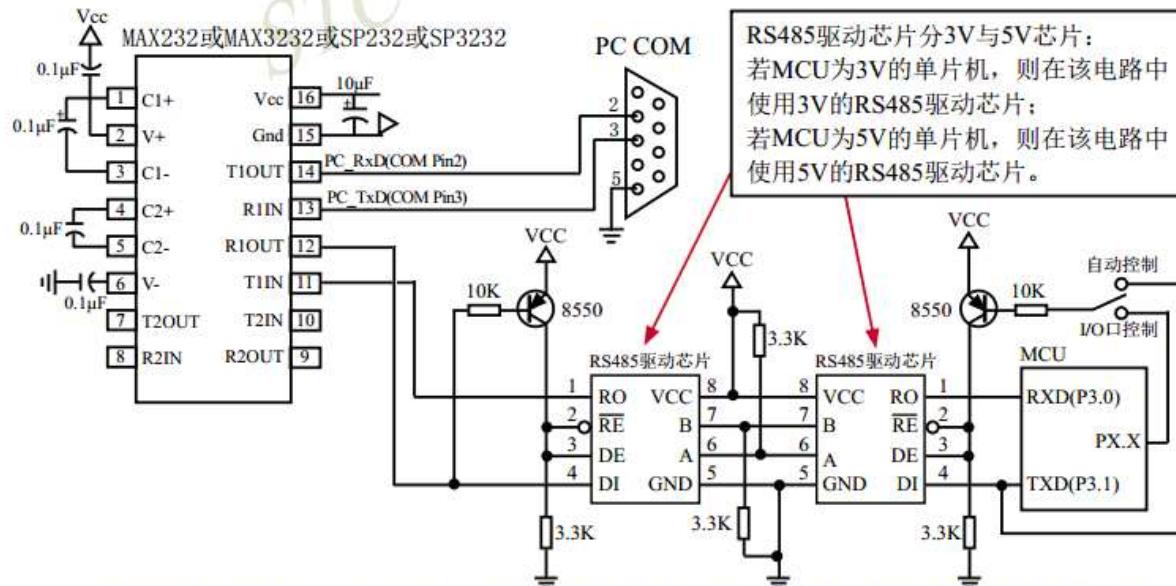


附录G RS485 自动控制或 I/O 口控制线路图

1、利用 USB 转串口连接电脑的 RS485 控制下载线路图(自动控制或 I/O 口控制)



2、利用 RS232 转串口连接电脑的 RS485 控制下载线路图(自动控制或 I/O 口控制)



注意：如果要设置单片机某个I/O口控制RS485发送或接收命令有效，则必须将单片机焊入电路板之前先用U8下载工具结合电脑ISP软件对该单片机进行“RS485控制”设置并烧录一下（如上节所述），否则将单片机实现不了RS485控制功能。

建议用户将本节所述“RS485控制下载线路图(自动控制或I/O口控制)”设计到您的用户板上

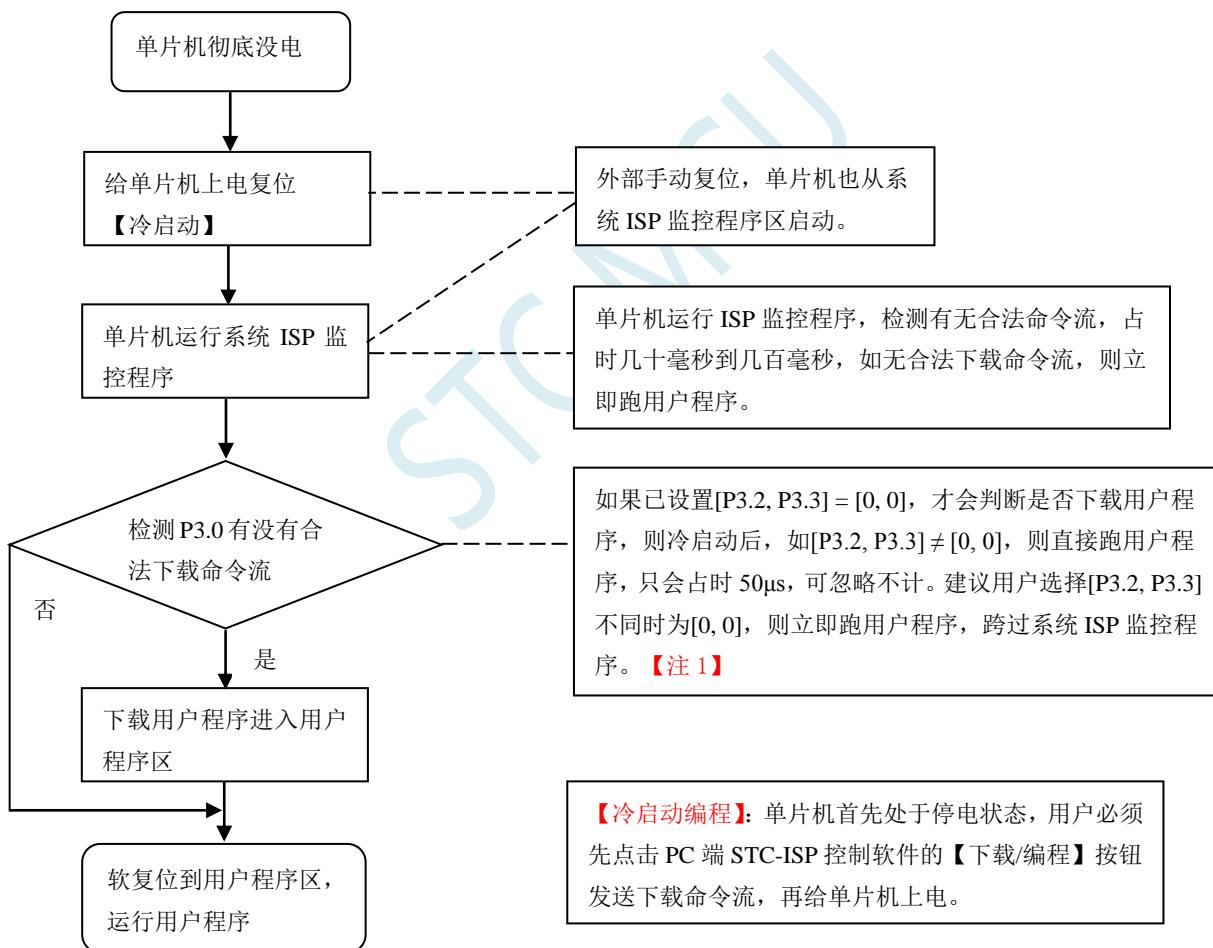
附录H STC 工具使用说明书

H.1 概述

U8W/U8W-Mini 是一款集在线联机下载和脱机下载于一体的编程工具系列。STC 通用 USB 转串口工具则是支持在线下载与在线仿真的编程工具。

工具类型	在线下载	脱机下载	烧录座下载	在线仿真	价格(人民币)
U8W	支持	支持	支持	需设置直通模式	100 元
U8W-Mini	支持	支持	不支持	需设置直通模式	50 元
通用 USB 转串口	支持	不支持	不支持	支持	30 元

H.2 系统可编程 (ISP) 流程说明

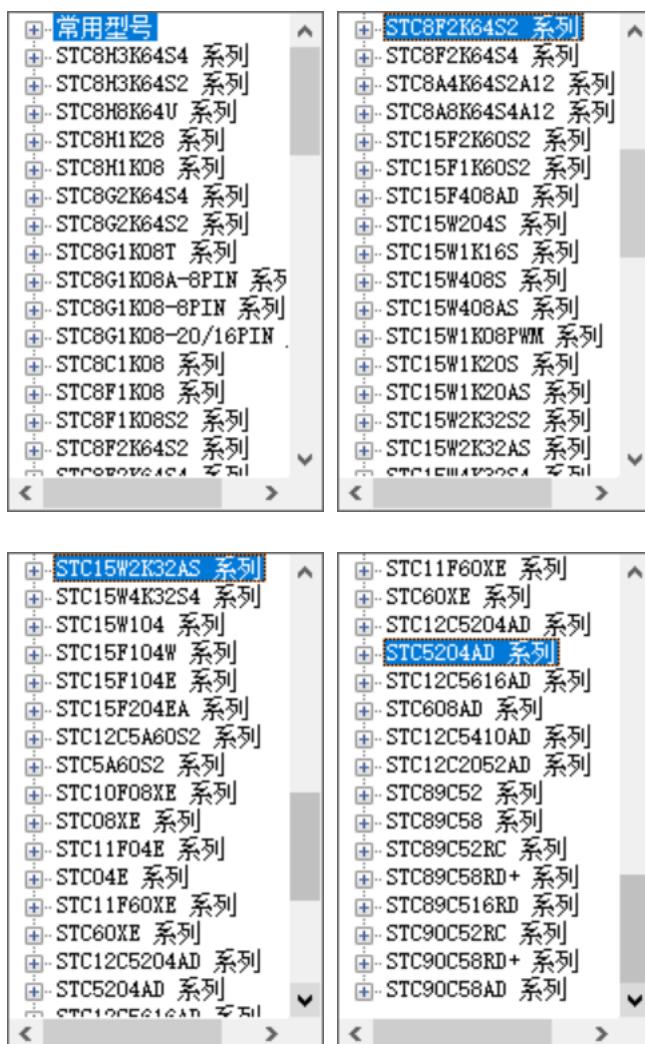


注意: 因 [P3.0, P3.1] 作下载/仿真用(下载/仿真接口仅可用 [P3.0, P3.1]), 故建议用户将串口 1 放在 P3.6/P3.7 或 P1.6/P1.7, 若用户不想切换, 坚持使用 P3.0/P3.1 工作或作为串口 1 进行通信, 则务必在下载程序时, 在软件上勾选“下次冷启动时, P3.2/P3.3 为 0/0 时才可以下载程序”。【注 1】

【注 1】: STC15, STC8 系列及以后新出的芯片的烧录保护引脚为 P3.2/P3.3, 之前早期芯片的烧录保护引脚为 P1.0/P1.1。

H.3 USB 型联机/脱机下载工具 U8W/U8W-Mini

U8W/U8W-Mini 的应用范围可支持 STC 目前的全部系列的 MCU, Flash 程序空间和 EEPROM 数据空间不受限制。支持包括如下和即将推出的 STC 全系列芯片:



脱机下载工具可以在脱离电脑的情况下进行下载工作，可用于批量生产和远程升级。脱机下载板可支持自动增量、下载次数限制以及用户程序加密后传输等多种功能。

下图为 U8W 工具的正反面图以及 U8W-Mini 的正反面图:



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

另外还有如下的一些线材与工具相搭配使用，如：



注意：此 USB 线为我公司特别定制的 USB 加强线，可确保直接用 USB 供电时能够下载成功。而市面上一些比较劣质的两头公的 USB 线，内阻太大而导致压降很大（如 USB 空载时的电压为 5.0V 左右，当使用劣质的 USB 线连接 U8W/U8W-Mini/U8/U8-Mini，到我们的下载板上的电压可能降到 4.2V 或者更低，从而导致芯片处于复位状态而无法成功下载）。

(2) U8W/U8W-Mini 与用户系统连接的下载连接线(即 U8W/U8W-Mini 与用户板上的目标单片机的连接线)，如下图所示：



H.3.1 安装 U8W/U8W-Mini 驱动程序

U8W/U8W-Mini 下载板上使用了一颗 CH340 的 USB 转串口通用芯片。这样可以省去部分没有串口的电脑必须额外买一个 USB 转串口工具才可下载的麻烦。但 CH340 和其它 USB 转串口工具一样，在使用之前必须先安装驱动程序。

通过下载 STC-ISP 软件包获取驱动程序

以下是 STC 官网 (www.STCMCUDATA.com) 提供的 STC-ISP 软件包下载位置:



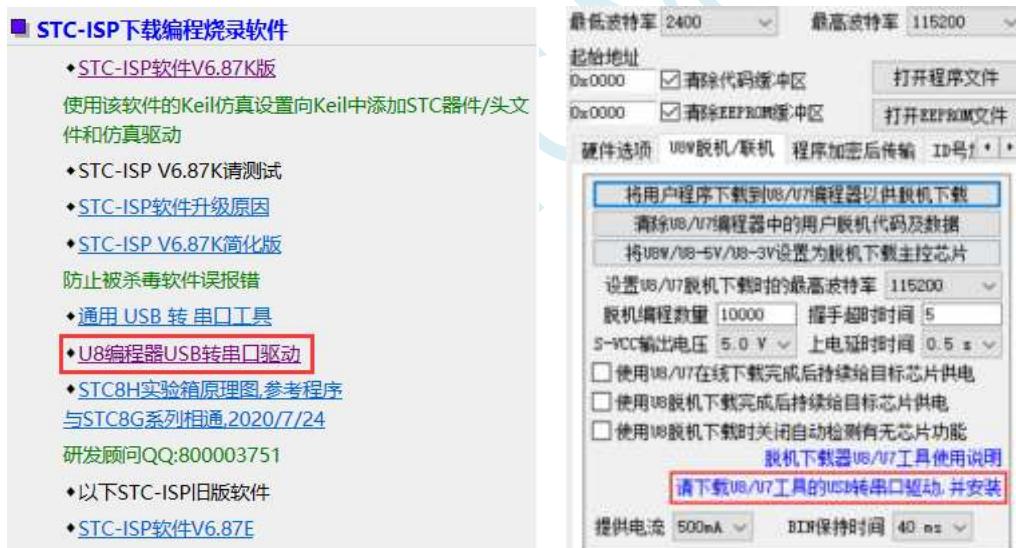
下载后进行解压, CH340 的驱动安装包路径 stc-ispp-15xx-v6.87K\USB to UART Driver\CH340_CH341:

i > 下载 > stc-ispp-15xx-v6.87K > USB to UART Driver > CH340_CH341

名称	修改日期
ch341ser	2020/5/9 15:03

通过 STC 的官方网站或在最新的 STC-ISP 下载软件中手动下载驱动程序

在 STC 的官方网站上或在最新的 STC-ISP 下载软件中手动下载驱动程序, 驱动的下载链接为: [U8 编程器 USB 转串口驱动](http://www.stcmcu.com/STCISP/CH341SER.exe) (<http://www.stcmcu.com/STCISP/CH341SER.exe>)。网站上及 STC-ISP 下载软件上的驱动地址如下图所示:

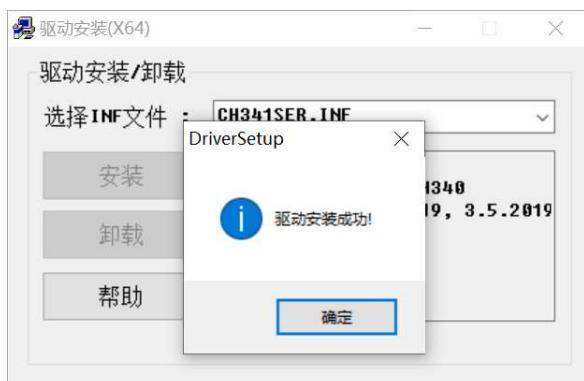


安装 U8W/U8W-Mini 的驱动程序

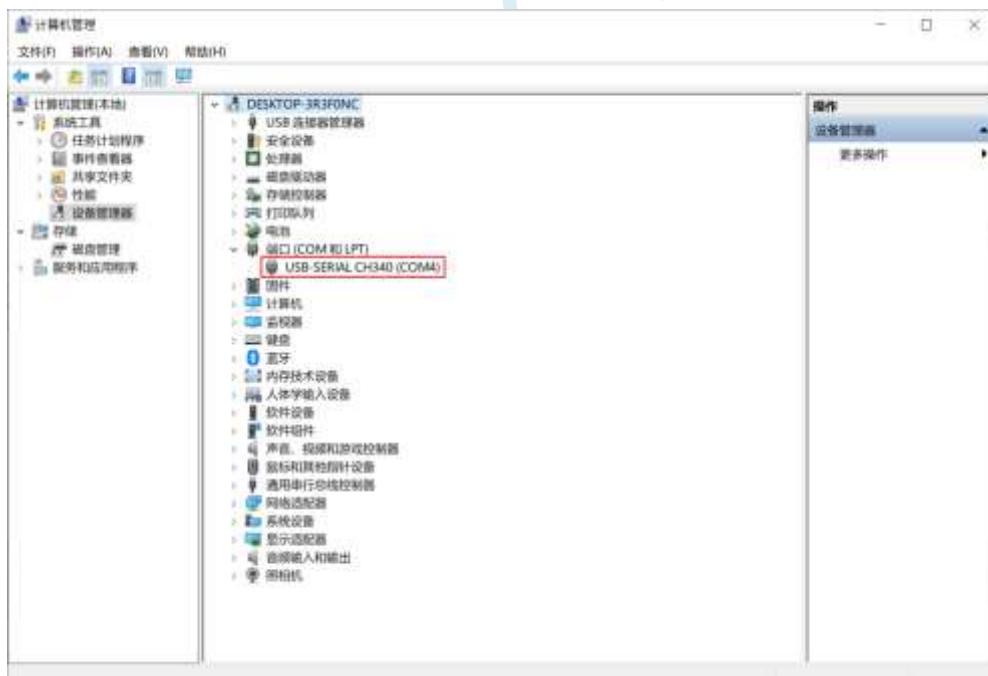
驱动程序下载到本机后, 直接双击可执行程序并运行, 出现下图所示的界面, 点击“安装”按钮开始自动安装驱动:



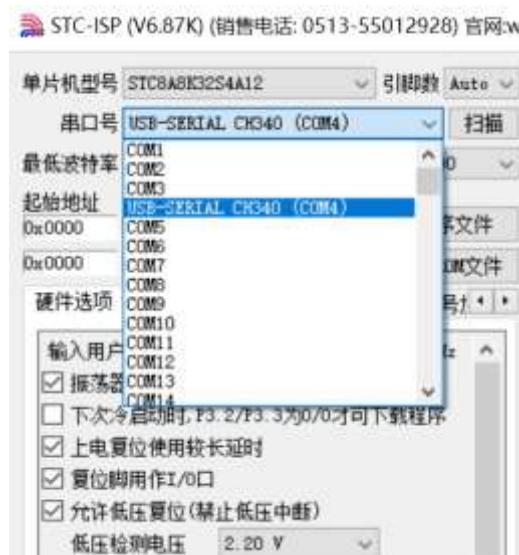
然后弹出驱动安装成功对话框，点击“确定”按钮完成安装：



然后使用 STC 提供的 USB 连接线将 U8W/U8W-Mini 下载板连接到电脑，打开电脑的设备管理器，在端口设备类下面，如果有类似“USB-SERIAL CH340 (COMx)”的设备，就表示 U8W/U8W-Mini 可以正常使用了。如下图所示（不同的电脑，串口号可能会不同）：



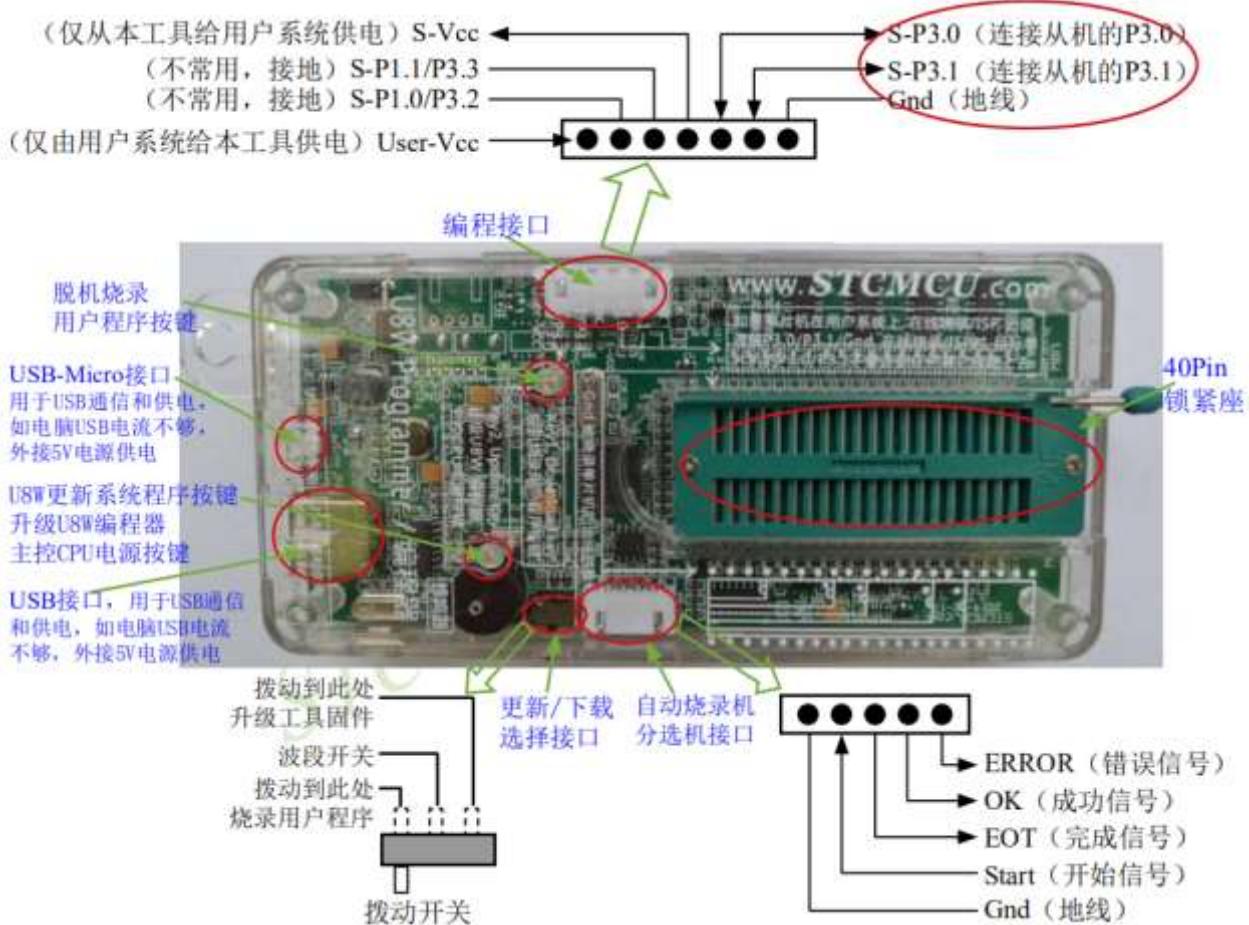
注意：在后面使用 STC-ISP 下载软件时，选择的串口号必须选择与此相对应的串口号，如下图所示：



H.3.2 U8W 的功能介绍

下面详细介绍 U8W 工具的各主要接口及功能:

如果单片机在用户系统上，在线烧录/ISP 时必须连接 P3.0/P3.1/Gnd，在线烧录/ISP 时，目标单片机的 P3.0/P3.1 不要连到任何其他线路上去



编程接口: 根据不同的供电方式, 使用不同的下载连接线连接 U8W 下载板和用户系统。

U8W 更新系统程序按键: 用于更新 U8W 工具, 当有新版本的 U8W 固件时, 需要按下此按键对 U8W 的主控芯片进行更新 (注意: 必须先将更新/下载选择接口上的拨动开关拨动到升级工具固件)。

脱机下载用户程序按钮: 开始脱机下载按钮。首先 PC 将脱机代码下载到 U8W 板上, 然后使用下载连接线将用户系统连接到 U8W, 再按下此按钮即可开始脱机下载 (每次上电时也会立即开始下载用户代码)。

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

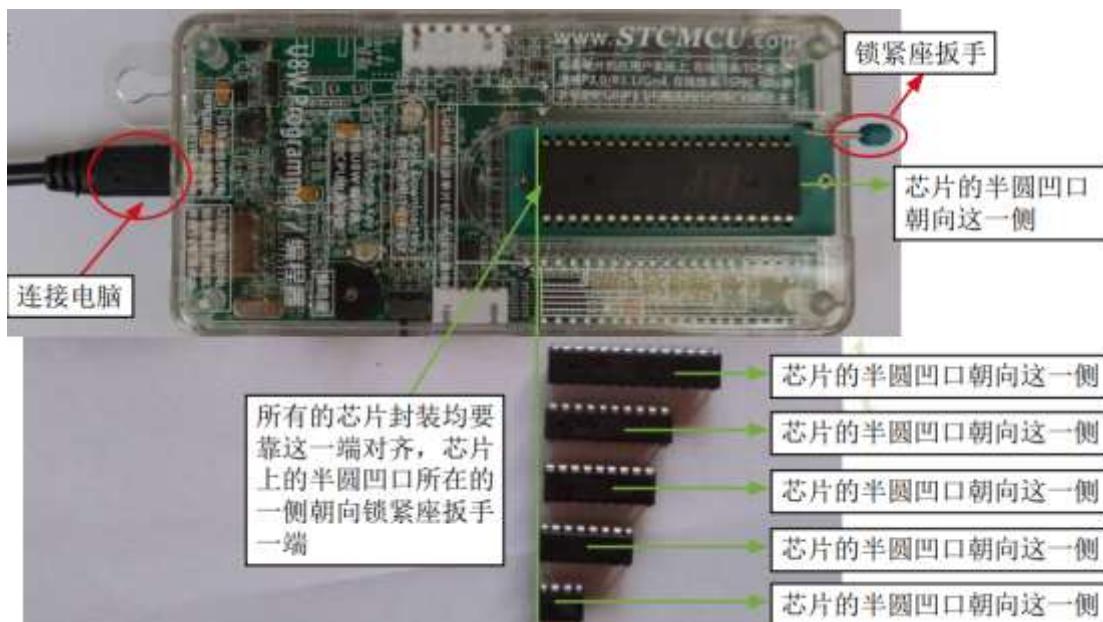
(拨动开关连接方式请参考上图)

自动烧录机/分选机接口: 是用于控制自动烧录机/分选机进行自动生产的控制接口。

H.3.3 U8W 的在线联机下载使用说明

目标芯片安装于 U8W 锁紧座上并由 U8W 连接电脑进行在线联机下载

首先使用 STC 提供的 USB 连接线将 U8W 连接电脑, 再将目标单片机按如下图所示的方向安装在 U8W 上:



然后使用 STC-ISP 下载软件下载程序，步骤如下：



- 选择单片机型号；
- 选择引脚数，芯片直接安装于 U8W 上下载时，一定要注意选择正确的引脚数，否则将会下载失败；
- 选择 U8W 所对应的串口号；
- 打开目标文件（HEX 格式或者 BIN 格式）；
- 设置硬件选项；
- 点击“下载/编程”按钮开始烧录；
- 显示烧录过程的步骤信息，烧录完成提示“操作成功！”。

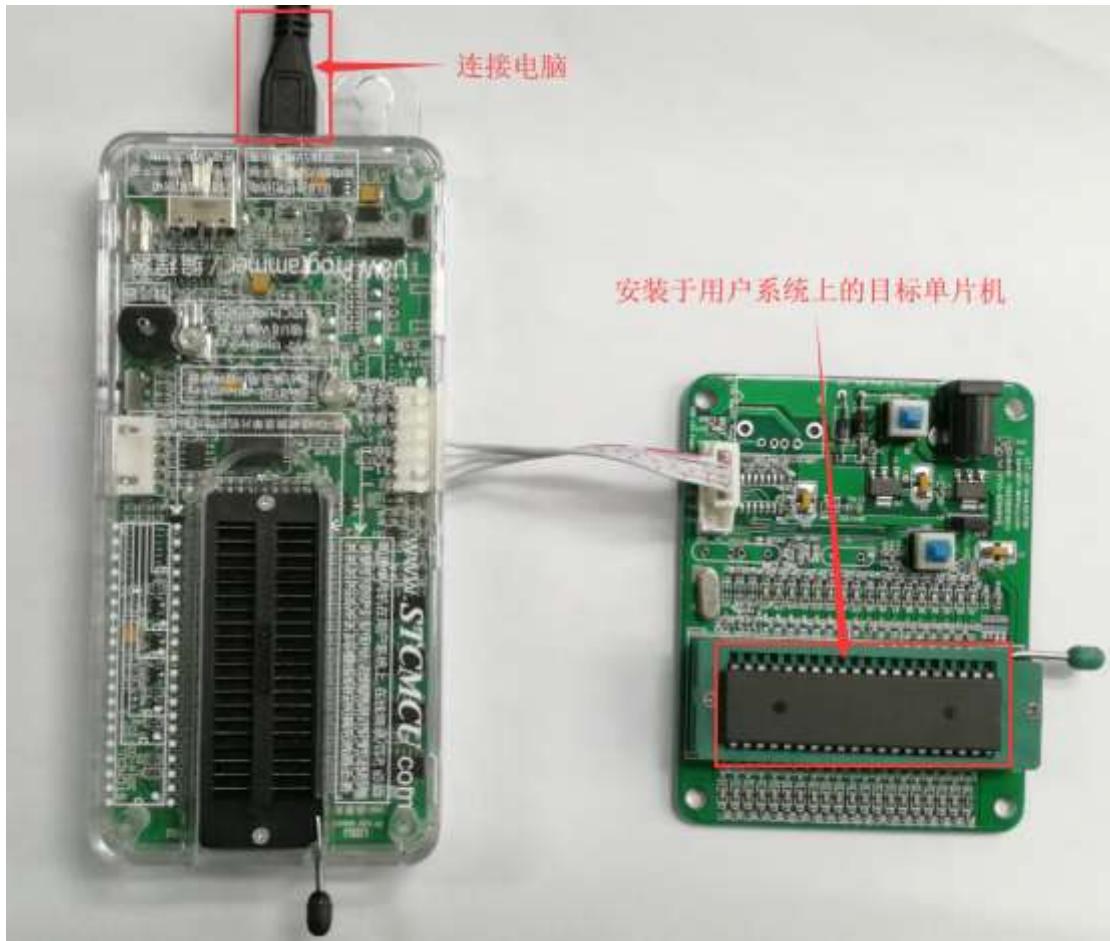
当信息框中有输出下载板的版本号信息以及外挂 Flash 的相应信息时，表示已正确检测到 U8W 下载工具。

下载的过程中, U8W 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后, 若下载成功, 则 4 个 LED 会同时亮、同时灭; 若下载失败, 则 4 个 LED 全部不亮。

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

目标芯片通过用户系统引线连接 U8W 并由 U8W 连接电脑进行在线联机下载

首先使用 STC 提供的 USB 连接线将 U8W 连接电脑, 再将 U8W 通过下载线与用户系统的单片机相连接, 连接方式如下图所示:



然后使用 STC-ISP 下载软件下载程序, 步骤如下:

1. 选择单片机型号;
2. 选择 U8W 所对应的串口号;
3. 打开目标文件 (HEX 格式或者 BIN 格式) ;
4. 设置硬件选项;
5. 点击“下载/编程”按钮开始烧录;
6. 显示烧录过程的步骤信息, 烧录完成提示“操作成功!”。



当信息框中有输出下载板的版本号信息以及外挂 Flash 的相应信息时，表示已正确检测到 U8W 下载工具。下载的过程中，U8W 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

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

H.3.4 U8W 的脱机下载使用说明

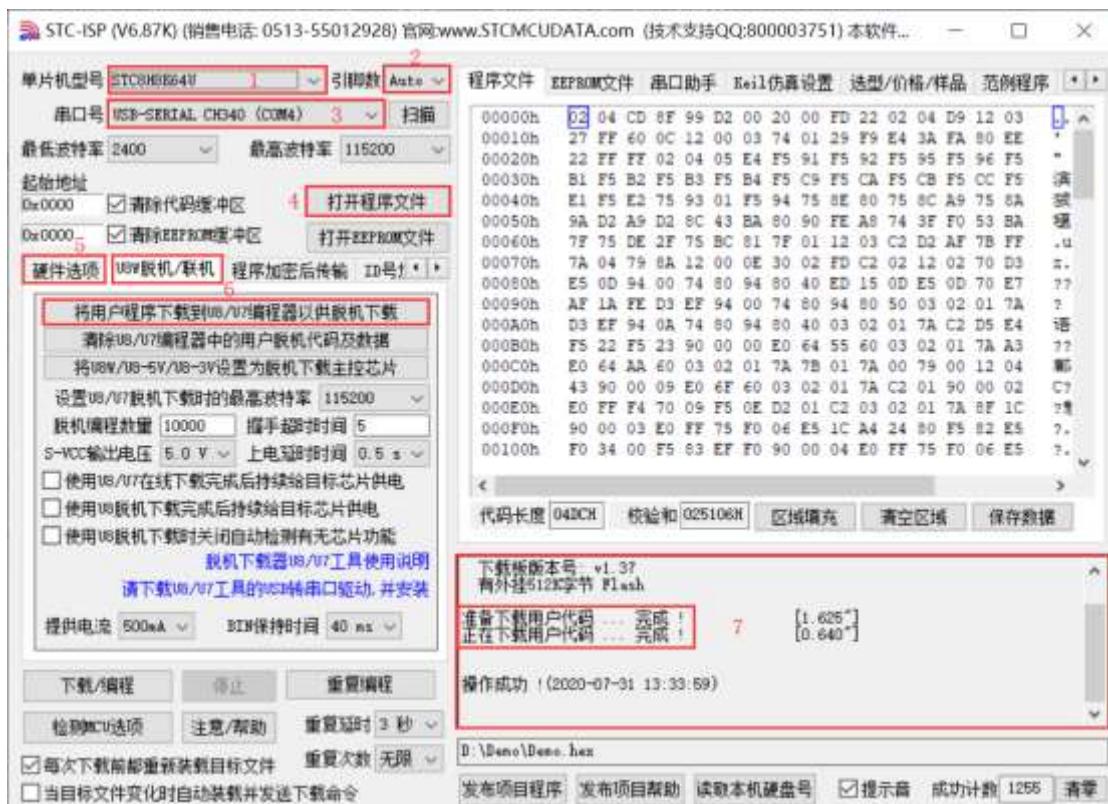
目标芯片安装于 U8W 座锁紧上并通过 USB 连接电脑给 U8W 供电进行脱机下载

使用 USB 给 U8W 供电从而进行脱机下载的步骤如下：

(1) 使用 STC 提供的 USB 连接线将 U8W 下载板连接到电脑，如下图：



(2) 在 STC-ISP 下载软件中按如下图所示的步骤进行设置：

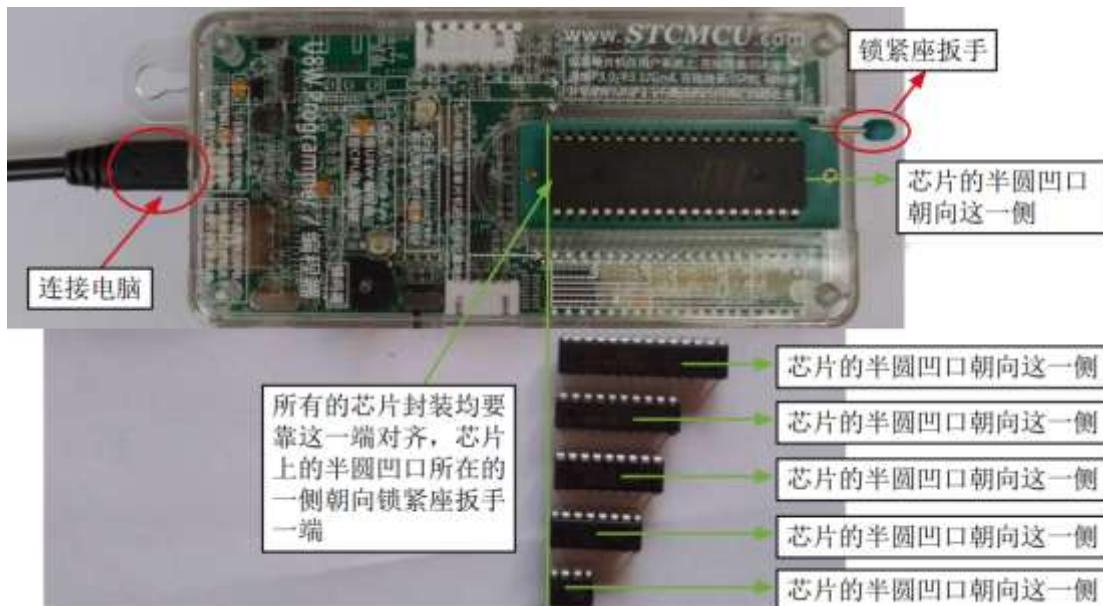


- 选择单片机型号；
- 选择引脚数，芯片直接安装于 U8W 上下载时，一定要注意选择正确的引脚数，否则将会下载失败；
- 选择 U8W 所对应的串口号；
- 打开目标文件（HEX 格式或者 BIN 格式）；
- 设置硬件选项；
- 选择“U8W 脱机/联机”标签，设置脱机编程选项，注意 S-VCC 输出电压与目标芯片工作电压匹配；点击“将用户程序下载到 U8/U7 编程器以供脱机下载”按钮；
- 显示设置过程的步骤信息，设置完成提示“操作成功！”。

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

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

(3) 再将目标单片机如下图所示的方向放在 U8W 下载工具，如下图所示：



(4) 然后按下如下图所示的按钮后松开, 即可开始脱机下载:



下载的过程中, U8W 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后, 若下载成功, 则 4 个 LED 会同时亮、同时灭; 若下载失败, 则 4 个 LED 全部不亮。

脱机下载即插即用烧录功能介绍:

1. 以上步骤完成(1)、(2)步之后 U8W 连接电脑上电时默认处于即插即用烧录状态;
2. 按照第(3)步指示将芯片放入烧录座, 在锁紧座扳手的同时, U8W 会自动开始烧录;
3. 通过指示灯显示烧录过程跟烧录结果;
4. 烧录完成后松开座扳手, 取出芯片;
5. 重复 2, 3, 4 步骤可进行连续烧录, 省掉按按钮触发烧录的动作。

目标芯片由用户系统引线连接 U8W 并通过 USB 连接电脑给 U8W 供电进行脱机下载

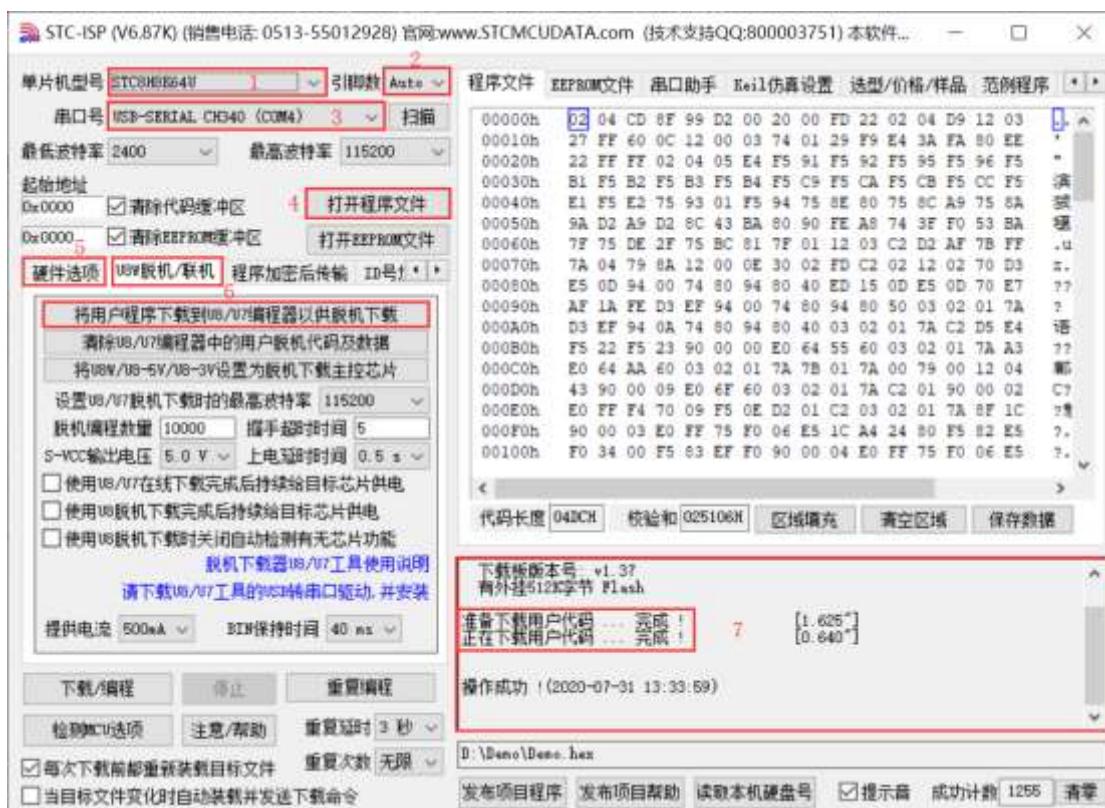
使用 USB 给 U8W 供电从而进行脱机下载的步骤如下:

- (1) 使用 STC 提供的 USB 连接线将 U8W 下载板连接到电脑, 如下图:



(2) 在 STC-ISP 下载软件中按如下图所示的步骤进行设置:

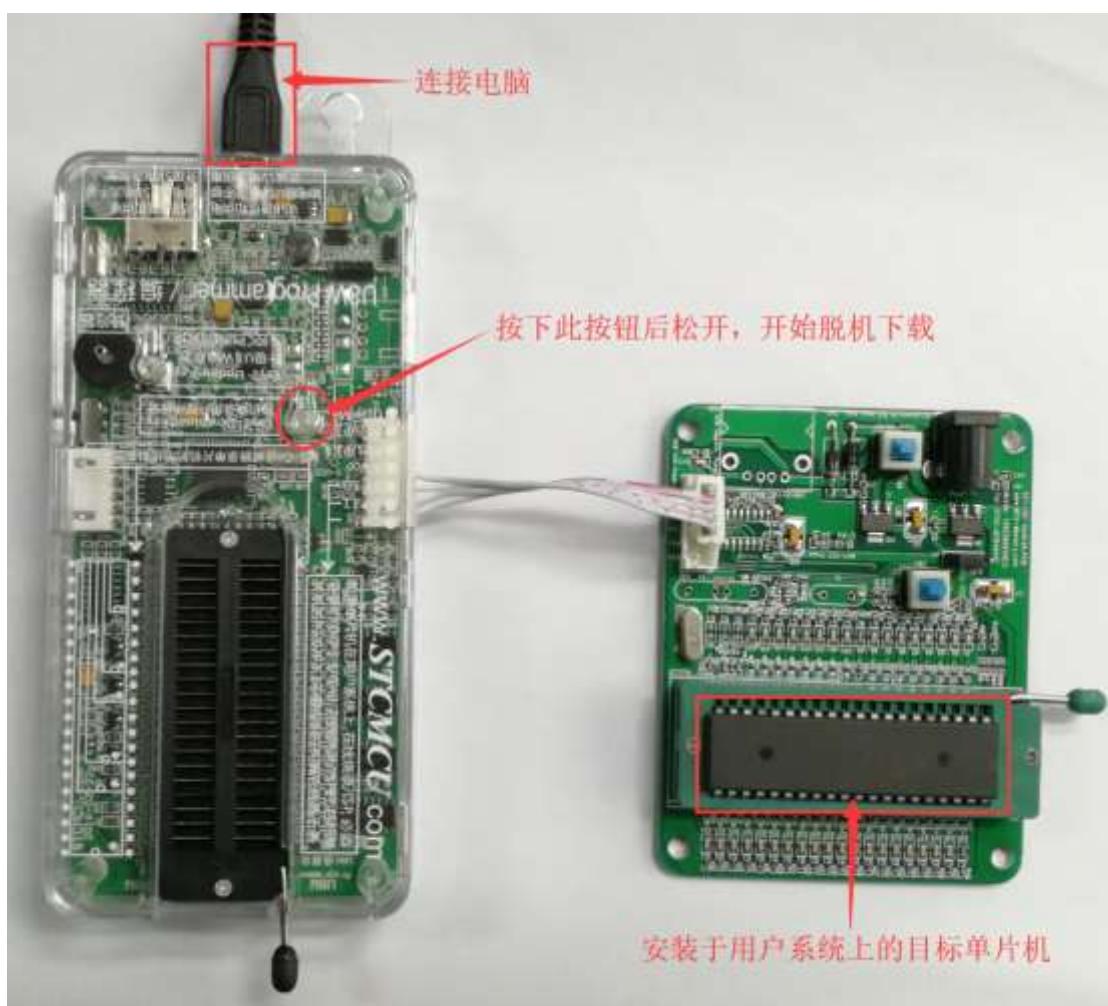
建议用户用最新版本的 STC-ISP 下载软件(请随时留意 STC 官方网站 <http://www.STCMCUDATA.com> 中 STC-ISP 下载软件的更新, 强烈建议用户在官方网站 <http://www.STCMCUDATA.com> 中下载最新版本的软件使用)。



1. 选择单片机型号;
2. 选择引脚数, 芯片直接安装于 U8W 上下载时, 一定要注意选择正确的引脚数, 否则将会下载失败;
3. 选择 U8W 所对应的串口号;
4. 打开目标文件 (HEX 格式或者 BIN 格式);
5. 设置硬件选项;
6. 选择“U8W 脱机/联机”标签, 设置脱机编程选项, 注意 S-VCC 输出电压与目标芯片工作电压匹配;
点击“将用户程序下载到 U8/U7 编程器以供脱机下载”按钮;
7. 显示设置过程的步骤信息, 设置完成提示“操作成功!”。

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

(3) 然后使用连接线连接电脑、将 U8W 下载工具以及用户系统 (目标单片机) 如下图所示的方式连接起来, 并按下图所示的按钮后松开, 即可开始脱机下载:



下载的过程中，U8W 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

目标芯片由用户系统引线连接 U8W 并通过用户系统给 U8W 供电进行脱机下载

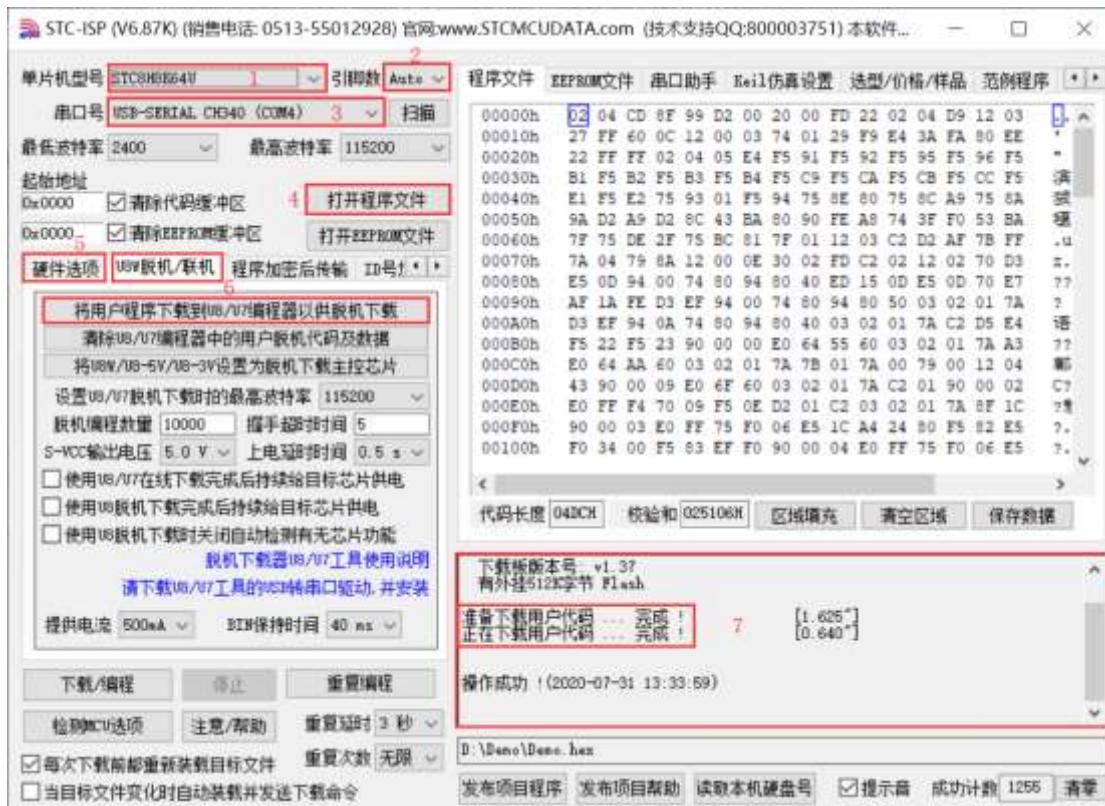
(1) 首先使用 STC 提供的 USB 连接线将 U8W 下载板连接到电脑，如下图：



(2) 在 STC-ISP 下载软件中按如下图所示的步骤进行设置：

建议用户用最新版本的 STC-ISP 下载软件（请随时留意 STC 官方网站 <http://www.STCMCUDATA.com> 中 STC-ISP 下载软件的更新，强烈建议用户在官方网站 <http://www.STCMCUDATA.com> 中下载最新版本的

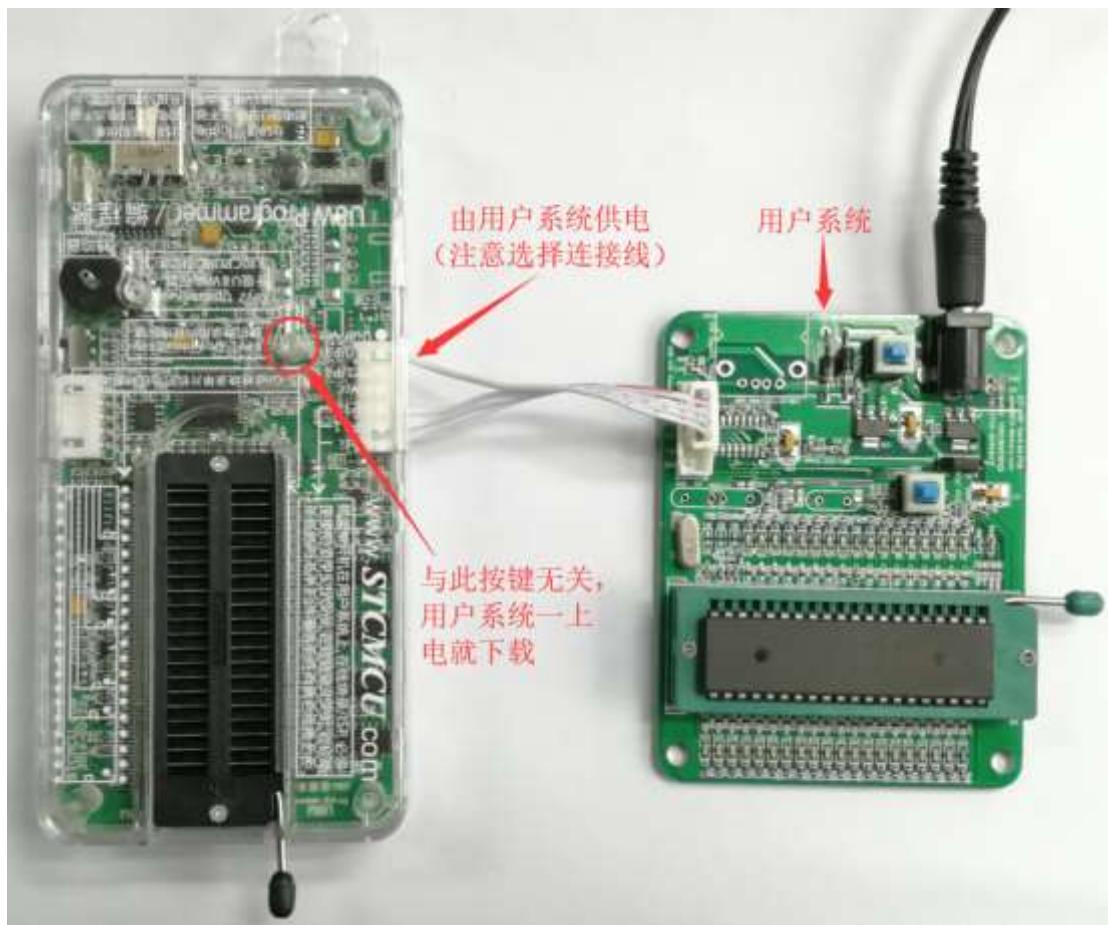
软件使用)。



1. 选择单片机型号；
2. 选择引脚数，芯片直接安装于 U8W 上下载时，一定要注意选择正确的引脚数，否则将会下载失败；
3. 选择 U8W 所对应的串口号；
4. 打开目标文件（HEX 格式或者 BIN 格式）；
5. 设置硬件选项；
6. 选择“U8W 脱机/联机”标签，设置脱机编程选项，注意 S-VCC 输出电压与目标芯片工作电压匹配；点击“将用户程序下载到 U8/U7 编程器以供脱机下载”按钮；
7. 显示设置过程的步骤信息，设置完成提示“操作成功！”。

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

(3) 然后按下图所示的方式连接 U8W 与用户系统，给用户系统供电，即可开始脱机下载：



下载的过程中，U8W 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

目标芯片由用户系统引线连接 U8W 且 U8W 与用户系统各自独立供电进行脱机下载

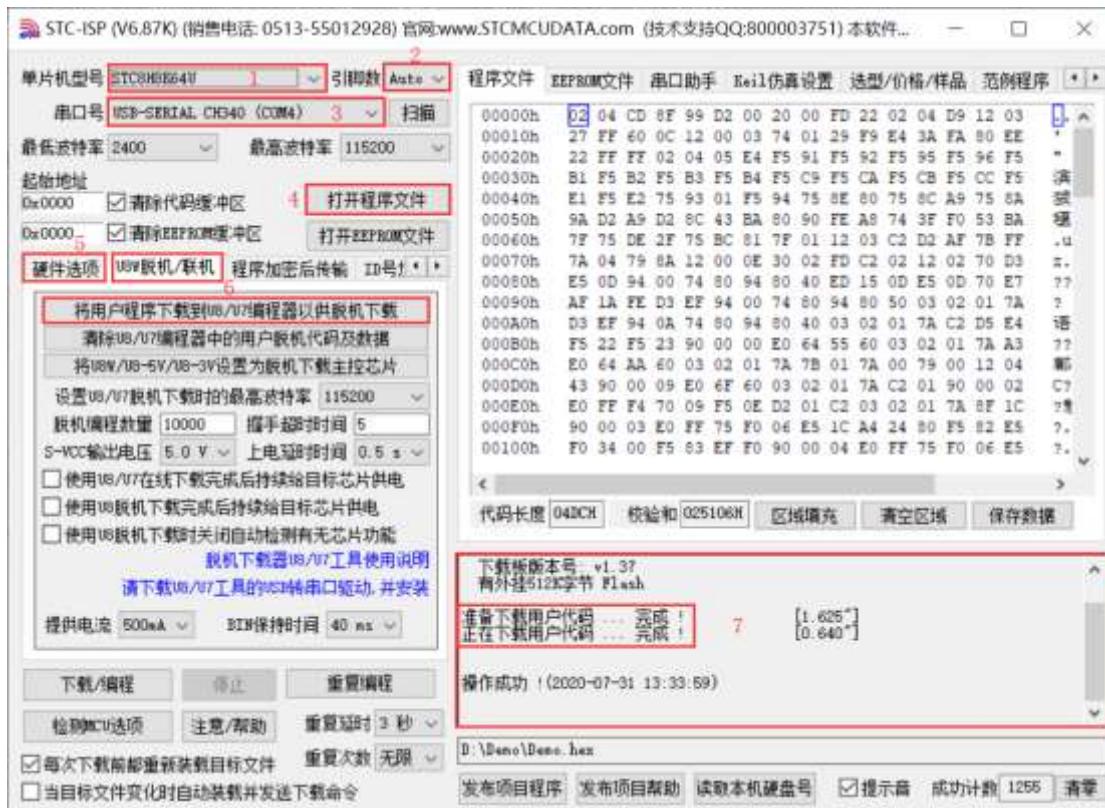
(1) 首先使用 STC 提供的 USB 连接线将 U8W 下载板连接到电脑，如下图：



(2) 在 STC-ISP 下载软件中按如下图所示的步骤进行设置：

建议用户用最新版本的 STC-ISP 下载软件（请随时留意 STC 官方网站 <http://www.STCMCUDATA.com> 中 STC-ISP 下载软件的更新，强烈建议用户在官方网站 <http://www.STCMCUDATA.com> 中下载最新版本的

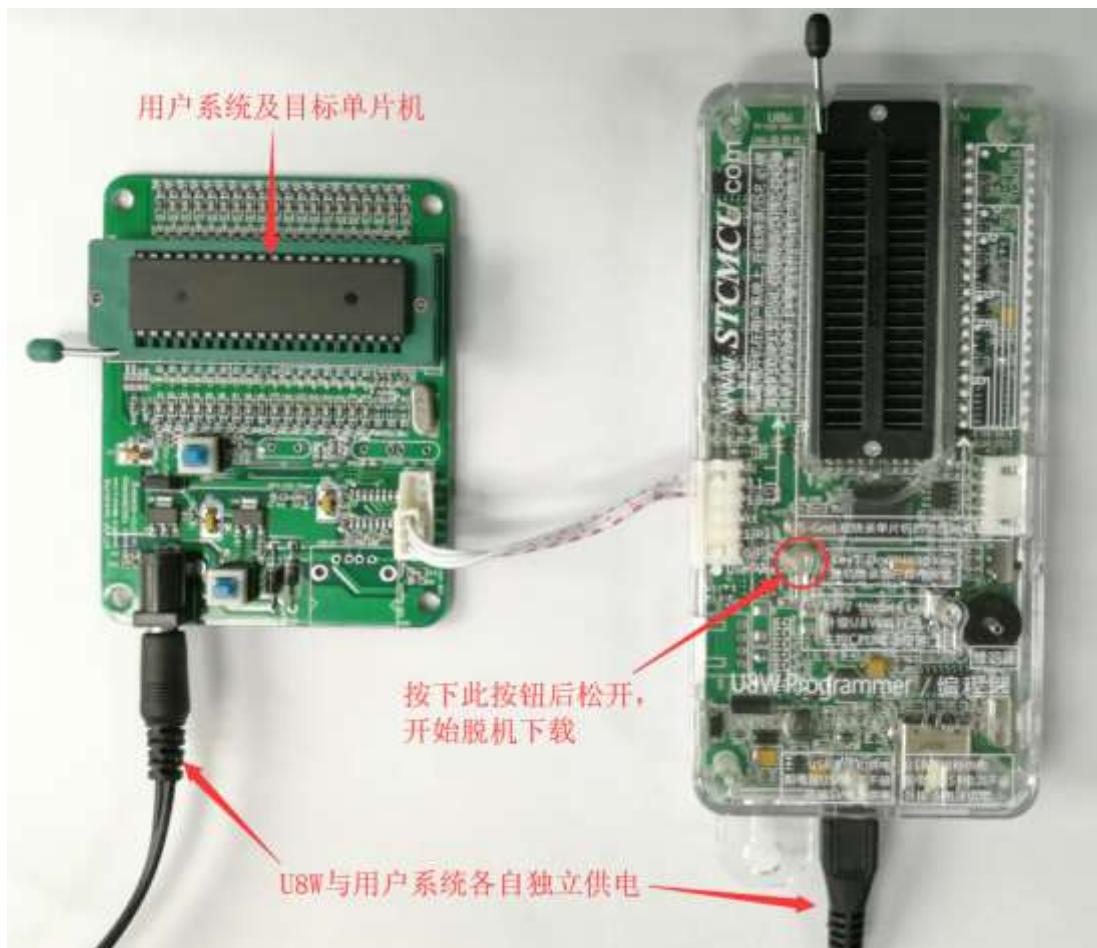
软件使用)。



1. 选择单片机型号；
2. 选择引脚数，芯片直接安装于 U8W 上下载时，一定要注意选择正确的引脚数，否则将会下载失败；
3. 选择 U8W 所对应的串口号；
4. 打开目标文件（HEX 格式或者 BIN 格式）；
5. 设置硬件选项；
6. 选择“U8W 脱机/联机”标签，设置脱机编程选项，注意 S-VCC 输出电压与目标芯片工作电压匹配；点击“将用户程序下载到 U8/U7 编程器以供脱机下载”按钮；
7. 显示设置过程的步骤信息，设置完成提示“操作成功！”。

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

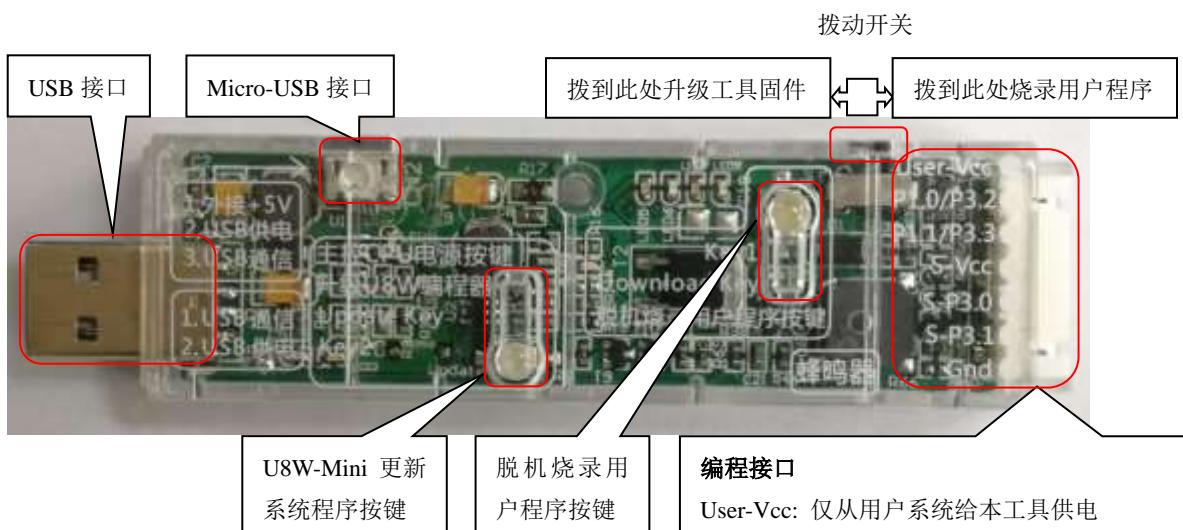
(3)然后按下图所示的方式连接 U8W 与用户系统，并将图中所示按钮先按下后松开，准备开始脱机下载，最后给用户系统上电/开电源，下载用户程序正式开始：



下载的过程中，U8W 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

H.3.5 U8W-Mini 的功能介绍

下面详细介绍 U8W-Mini 工具的各主要接口及功能：



编程接口: 根据不同的供电方式, 使用不同的下载连接线连接 U8W-Mini 下载板和用户系统。

U8W-Mini 更新系统程序按键: 用于更新 U8W-Mini 工具, 当有新版本的 U8W 固件时, 需要按下此按键对 U8W-Mini 的主控芯片进行更新 (注意: 必须先将更新/下载选择接口上的拨动开关拨动到升级工具固件)。

脱机下载用户程序按钮: 开始脱机下载按钮。首先 PC 将脱机代码下载到 U8W-Mini 上, 然后使用下载连接线将用户系统连接到 U8W-Mini, 再按下此按钮即可开始脱机下载 (每次上电时也会立即开始下载用户代码)。

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

USB 接口: USB 接口与 Micro-USB 接口是相同的功能, 用户根据需要连接其中一个接口到电脑即可。

H.3.6 U8W-Mini 的在线联机下载使用说明

目标芯片通过用户系统引线连接 U8W-Mini 并由 U8W-Mini 连接电脑进行在线联机下载

首先使用 STC 提供的 USB 连接线将 U8W-Mini 连接电脑, 再将 U8W-Mini 通过下载线与用户系统的单片机相连接, 连接方式如下图所示:



然后使用 STC-ISP 下载软件下载程序, 步骤如下:



1. 选择单片机型号；
2. 选择引脚数，芯片直接安装于 U8W-Mini 上下载时，一定要注意选择正确的引脚数，否则将会下载失败；
3. 选择 U8W-Mini 所对应的串口号；
4. 打开目标文件（HEX 格式或者 BIN 格式）；
5. 设置硬件选项；
6. 点击“下载/编程”按钮开始烧录；
7. 显示烧录过程的步骤信息，烧录完成提示“操作成功！”。

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

下载的过程中，U8W-Mini 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

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

H.3.7 U8W-Mini 的脱机下载使用说明

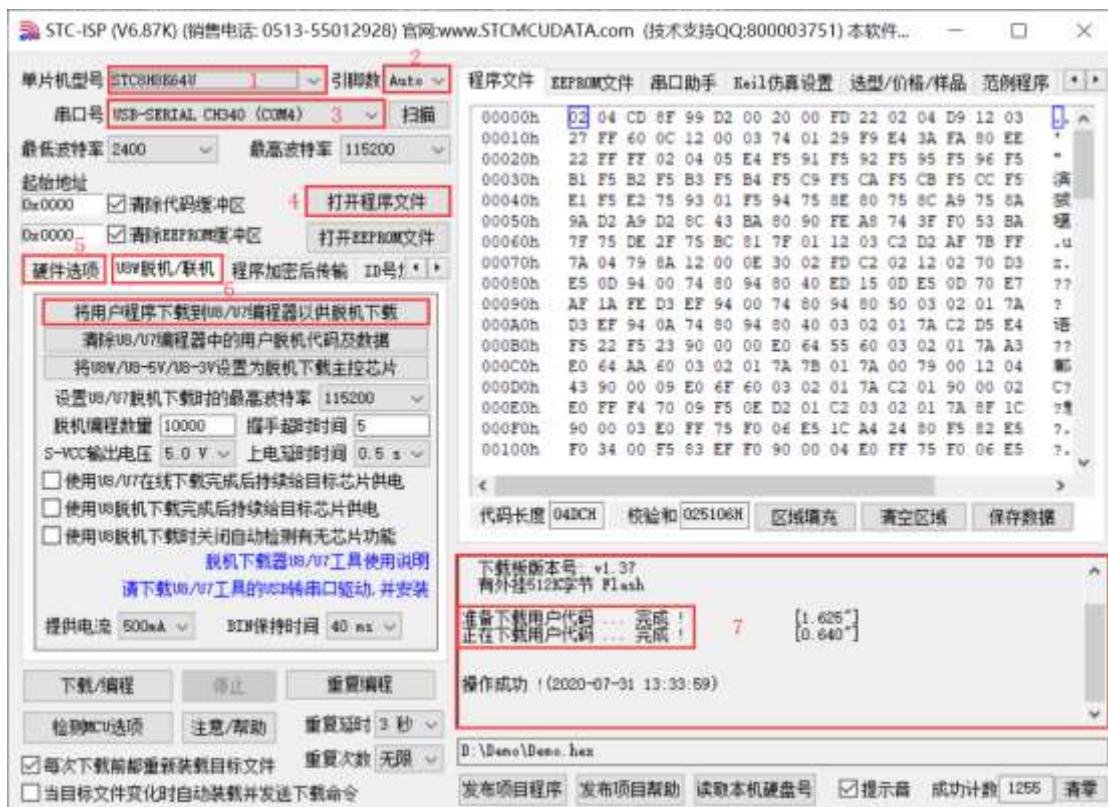
目标芯片由用户系统引线连接 U8W-Mini 并通过 USB 连接电脑给 U8W-Mini 供电进行脱机下载

使用 USB 给 U8W-Mini 供电从而进行脱机下载的步骤如下：

- (1) 使用 STC 提供的 USB 连接线将 U8W-Mini 下载板连接到电脑，如下图：



(2) 在 STC-ISP 下载软件中按如下图所示的步骤进行设置:

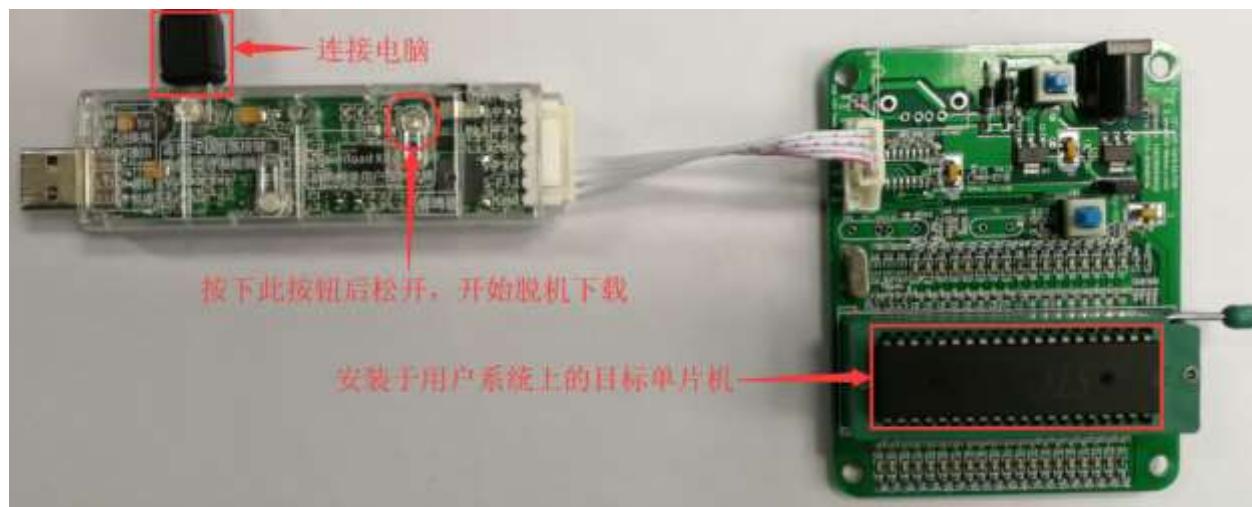


1. 选择单片机型号；
2. 选择引脚数，芯片直接安装于 U8W-Mini 上下载时，一定要注意选择正确的引脚数，否则将会下载失败；
3. 选择 U8W-Mini 所对应的串口号；
4. 打开目标文件（HEX 格式或者 BIN 格式）；
5. 设置硬件选项；
6. 选择“U8W 脱机/联机”标签，设置脱机编程选项，注意 S-VCC 输出电压与目标芯片工作电压匹配；点击“将用户程序下载到 U8/U7 编程器以供脱机下载”按钮；
7. 显示设置过程的步骤信息，设置完成提示“操作成功！”。

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

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

(3) 然后使用连接线连接电脑、将 U8W-Mini 下载工具以及用户系统(目标单片机)如下图所示的方式连接起来, 并按下图所示的按钮后松开, 即可开始脱机下载:



下载的过程中, U8W-Mini 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后, 若下载成功, 则 4 个 LED 会同时亮、同时灭; 若下载失败, 则 4 个 LED 全部不亮。

目标芯片由用户系统引线连接 U8W-Mini 并通过用户系统给 U8W-Mini 供电进行脱机下载

(1) 首先使用 STC 提供的 USB 连接线将 U8W-Mini 下载板连接到电脑, 如下图:



(2) 在 STC-ISP 下载软件中按如下图所示的步骤进行设置:

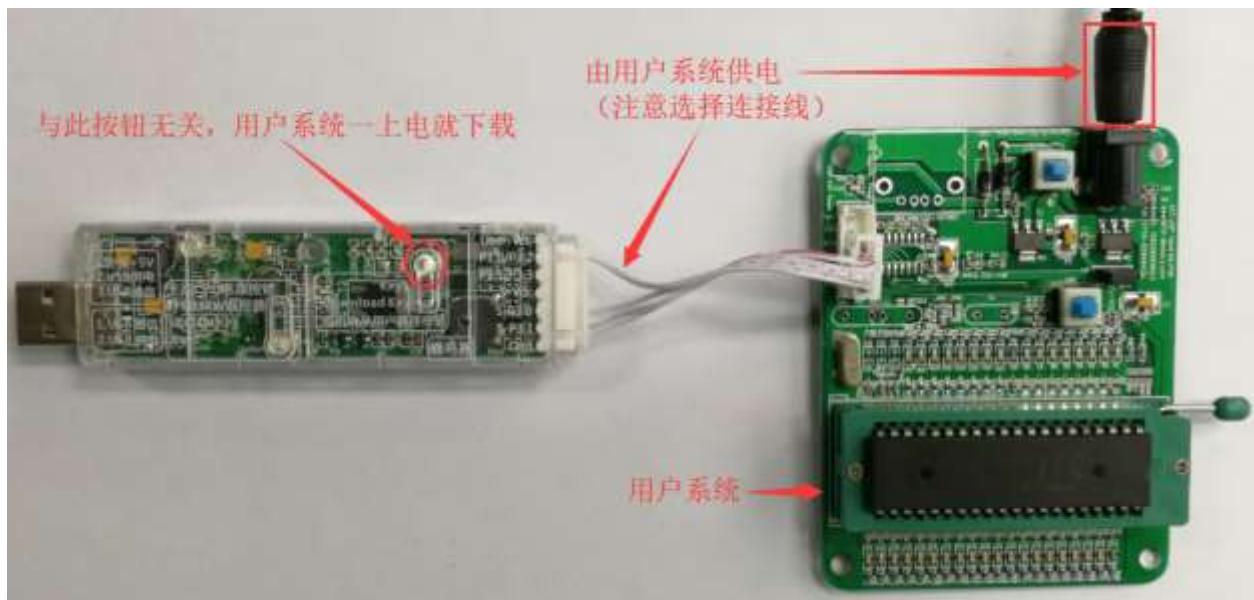


1. 选择单片机型号；
2. 选择引脚数，芯片直接安装于 U8W-Mini 上下载时，一定要注意选择正确的引脚数，否则将会下载失败；
3. 选择 U8W-Mini 所对应的串口号；
4. 打开目标文件（HEX 格式或者 BIN 格式）；
5. 设置硬件选项；
6. 选择“U8W 脱机/联机”标签，设置脱机编程选项，注意 S-VCC 输出电压与目标芯片工作电压匹配；点击“将用户程序下载到 U8/U7 编程器以供脱机下载”按钮；
7. 显示设置过程的步骤信息，设置完成提示“操作成功！”。

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

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

(3) 然后按下图所示的方式连接 U8W-Mini 与用户系统，用户系统一上电就开始脱机下载：



下载的过程中，U8W-Mini 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

目标芯片由用户系统引线连接 U8W-Mini 且 U8W-Mini 与用户系统各自独立供电进行脱机下载

(1) 首先使用 STC 提供的 USB 连接线将 U8W-Mini 下载板连接到电脑，如下图：



(2) 在 STC-ISP 下载软件中按如下图所示的步骤进行设置：

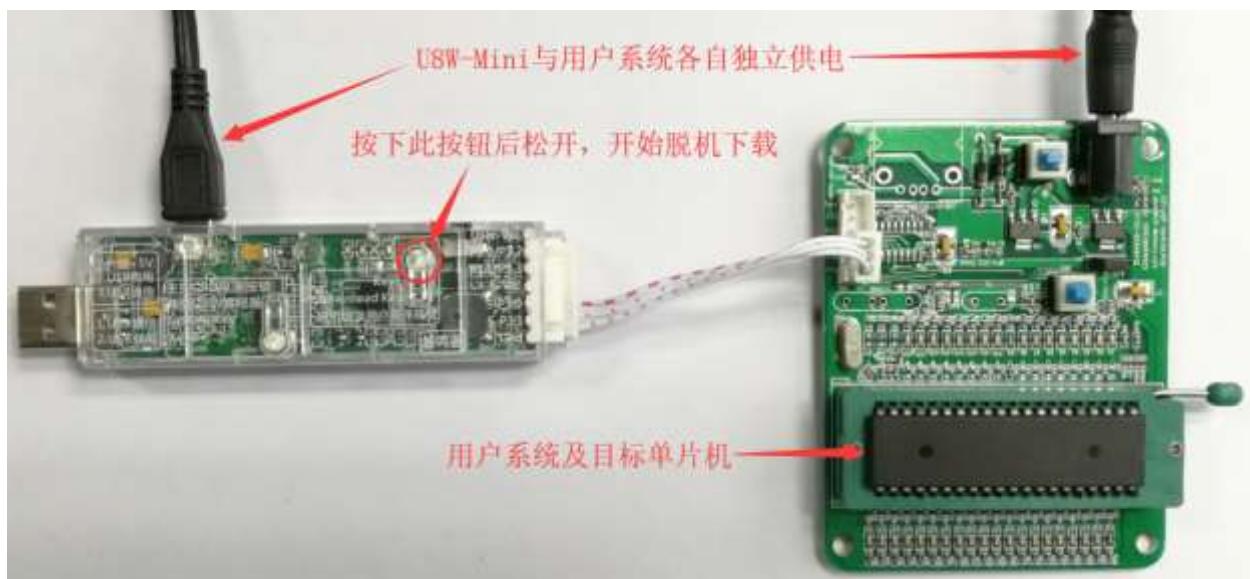


1. 选择单片机型号；
2. 选择引脚数，芯片直接安装于 U8W-Mini 上下载时，一定要注意选择正确的引脚数，否则将会下载失败；
3. 选择 U8W-Mini 所对应的串口号；
4. 打开目标文件（HEX 格式或者 BIN 格式）；
5. 设置硬件选项；
6. 选择“U8W 脱机/联机”标签，设置脱机编程选项，注意 S-VCC 输出电压与目标芯片工作电压匹配；点击“将用户程序下载到 U8/U7 编程器以供脱机下载”按钮；
7. 显示设置过程的步骤信息，设置完成提示“操作成功！”。

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

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

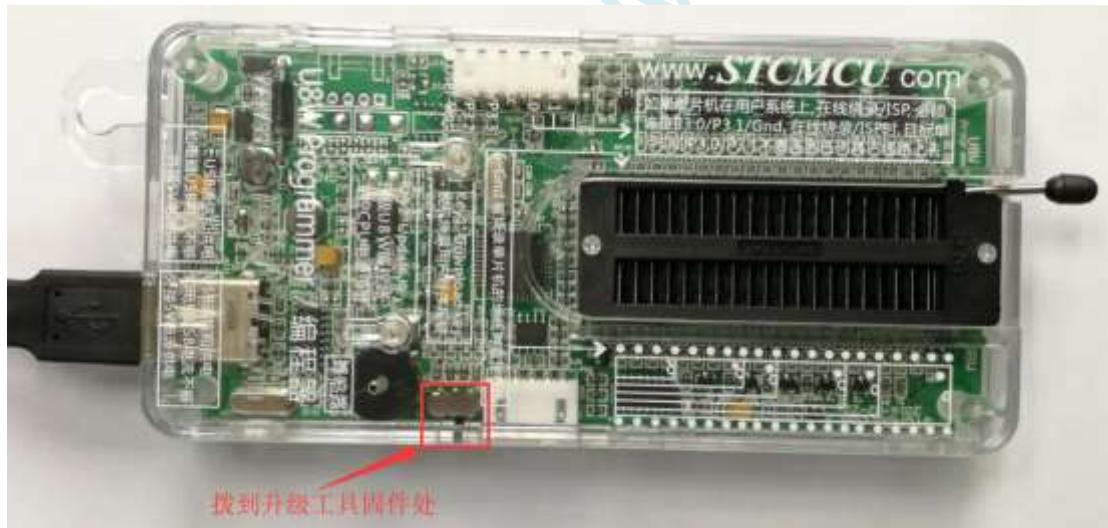
(3) 然后按下图所示的方式连接 U8W-Mini 与用户系统，并将图中所示按钮先按下后松开，准备开始脱机下载，最后给用户系统上电/开电源，下载用户程序正式开始：



下载的过程中, U8W-Mini 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后, 若下载成功, 则 4 个 LED 会同时亮、同时灭; 若下载失败, 则 4 个 LED 全部不亮。

H.3.8 制作/更新 U8W/U8W-Mini

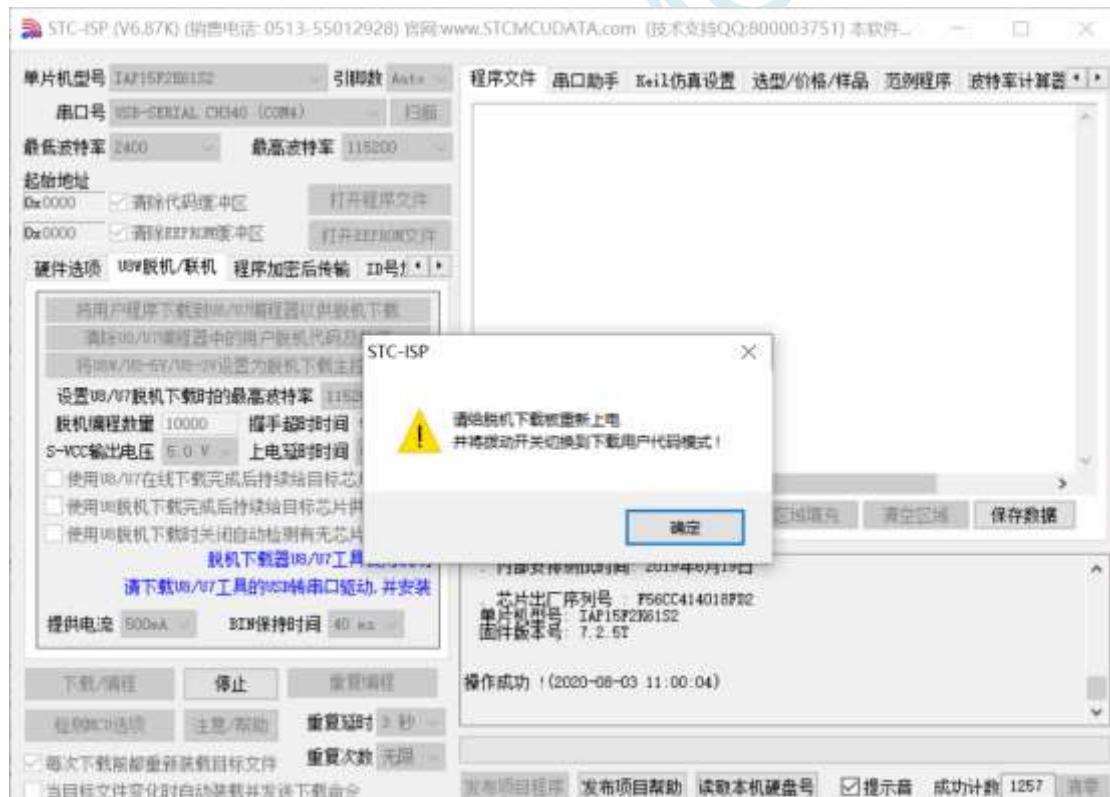
制作 U8W/U8W-Mini 下载母片的过程类似, 为节约篇幅, 下文以 U8W 为例, 详述如何制作 U8W 下载母片。在制作 U8W 下载母片之前需要将 U8W 下载板的“更新/下载选择接口”拨到“升级工具固件”, 如下图所示:



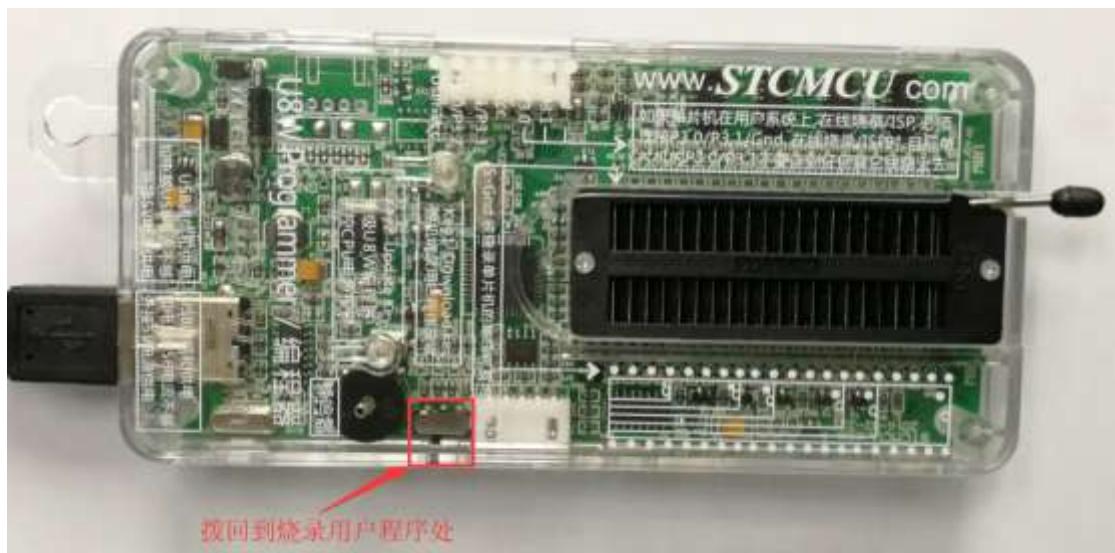
然后在 STC-ISP 下载程序中的“U8W 脱机/联机”页面中点击“将 U8W/U8-5V/U8-3V 设置为脱机下载主控芯片”按钮, 如下图: (注意: 一定要选择 U8W 所对应的串口)



在出现如下画面表示 U8W 控制芯片制作完成:



制作完成后，一定不要忘记将 U8W 的“更新/下载选择接口”拨回到“烧录用户程序”模式，并将 U8W 下载工具重新上电，如下图所示：(否则将不能正常进行烧录)



H.3.9 U8W/U8W-Mini 设置直通模式（可用于仿真）

若要使用 U8W/U8W-Mini 进行仿真，首先必须将 U8W/U8W-Mini 设置为直通模式。U8W/U8W-Mini 实现 USB 转串口直通模式的方法如下：

1. 首先 U8W/U8W-Mini 固件必须升级到 v1.37 及以上版本；
2. U8W/U8W-Mini 上电后为正常下载模式，此时按住工具上的 Key1（下载）按键不要松开，再按一下 Key2（电源）按键，然后放开 Key2（电源）按键后，再松开 Key1（下载）按键，U8W/U8W-Mini 会进入 USB 转串口直通模式。（按下 Key1 → 按下 Key2 → 松开 Key2 → 松开 Key1）；
3. 进入直通模式的 U8W/U8W-Mini 工具只是简单的 USB 转串口不具备脱机下载功能，若需要恢复 U8W/U8W-Mini 的原有功能，只需要再次单独按一下 Key2（电源）按键即可。

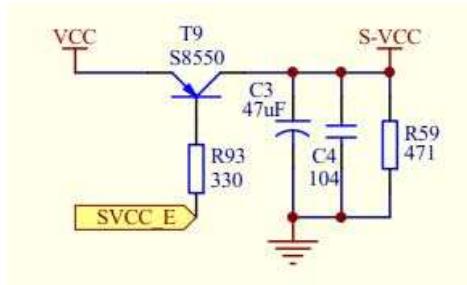
H.3.10 U8W/U8W-Mini 的参考电路

USB 型联机/脱机下载板 U8W/U8W-Mini 为用户提供了如下的常用控制接口：

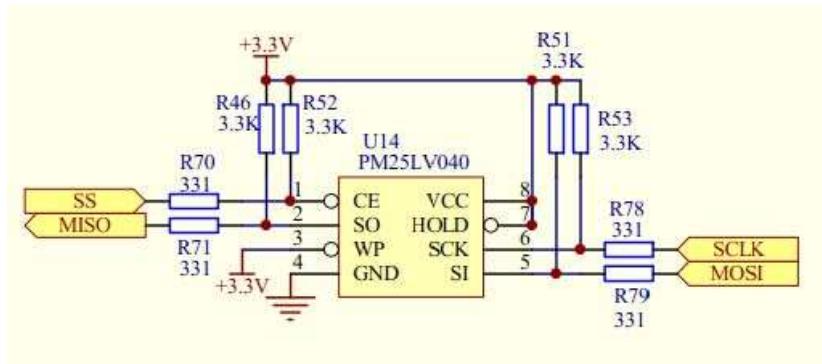
脚位功能	端口	功能描述
电源控制脚	P2.6	低位有效
下载通讯脚	P1.0	串口 RXD，连接目标芯片的 TXD (P3.1)
	P1.1	串口 TXD，连接目标芯片的 RXD (P3.0)
编程按键	P3.6	低有效
显示	P3.2	LED1
	P3.3	LED2
	P3.4	LED3
	P5.5	LED4
外挂串行 Flash 控制脚	P2.4	Flash 的 CE 脚
	P2.2	Flash 的 SO 脚
	P2.3	Flash 的 SI 脚
	P2.1	Flash 的 SCLK 脚

全自动烧录工具 分选机信号	P3.6	起始信号
	P1.5	完成信号
	P5.4	OK 信号 (良品信号)
	P3.7	ERROR 信号 (不良品信号)
	P2.5	高有效 (高电平发出声音)

电源控制部分参考电路图:

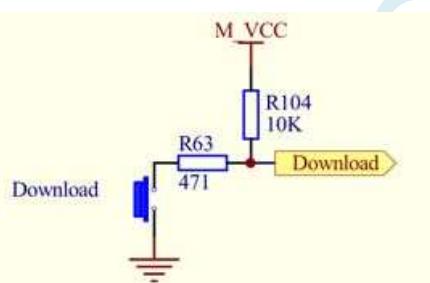


Flash 控制部分参考电路图:

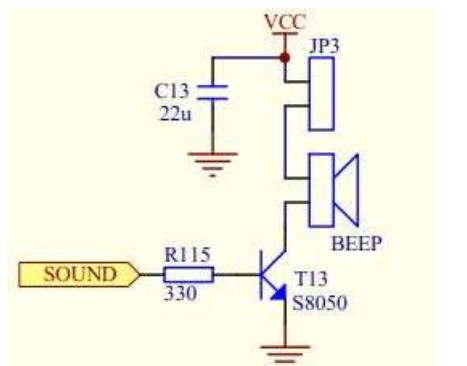


用户程序大于 41K 时需要此 Flash 存储器

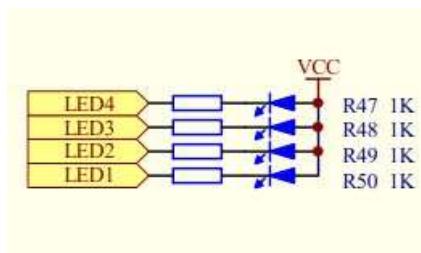
按键部分参考电路图:



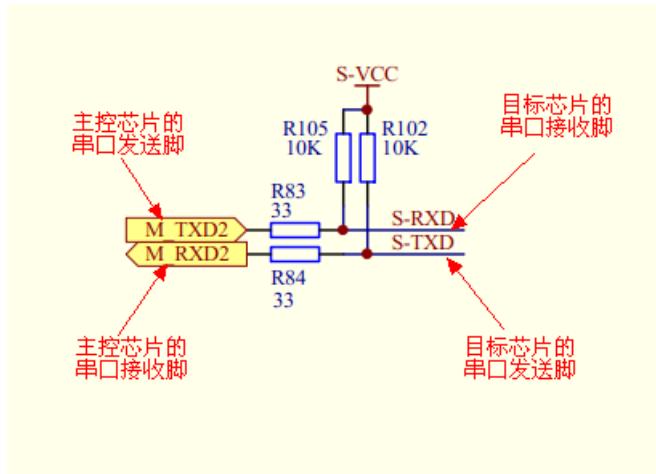
蜂鸣器部分参考电路图:



LED 显示部分参考电路图:



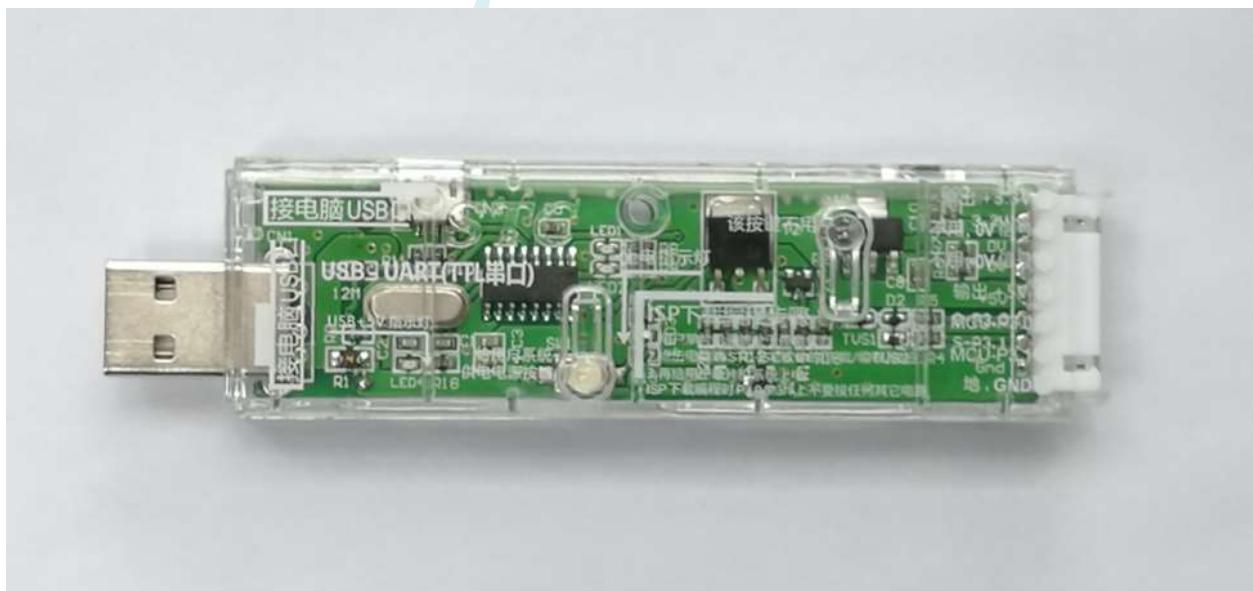
串口通讯脚连接部分参考电路图:



H.4 STC 通用 USB 转串口工具

H.4.1 STC 通用 USB 转串口工具外观图

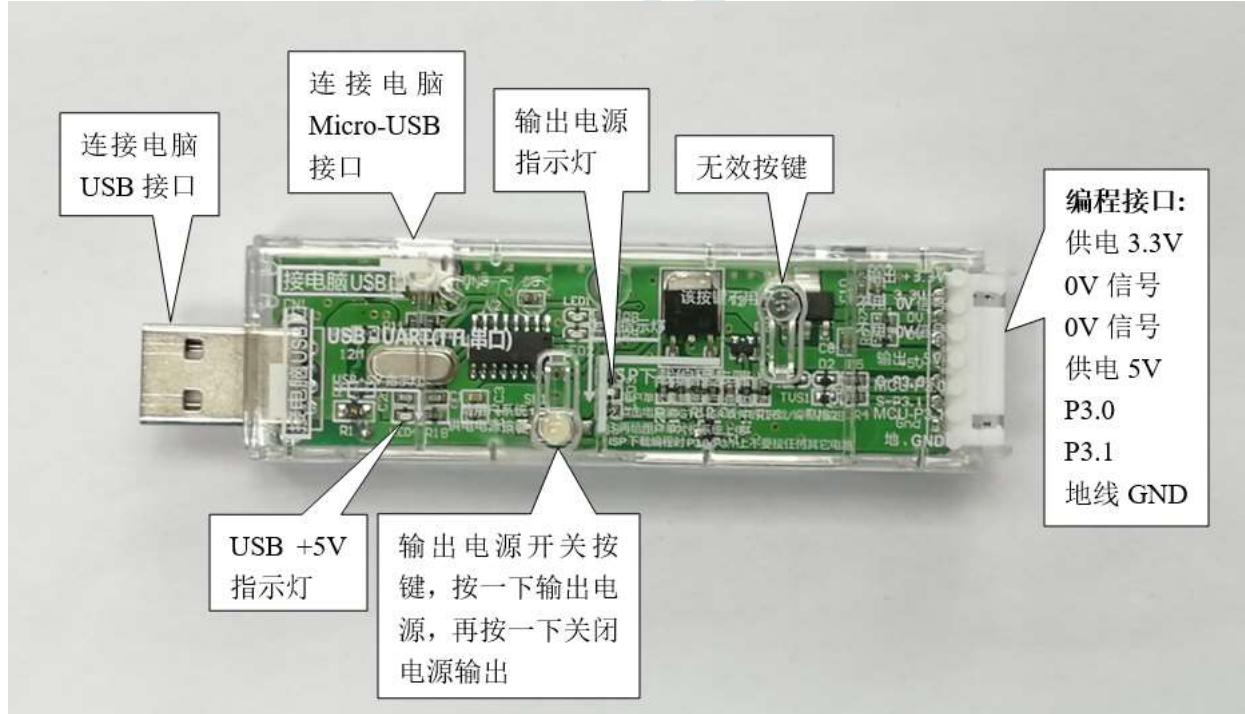
正面:



背面:



H.4.2 STC 通用 USB 转串口工具布局图



而对于 STC 的单片机，要想进行 ISP 下载，则必须是在上电复位时接收到串口命令才会开始执行 ISP

程序，所以使用 STC 通用 USB 转串口工具下载程序到 MCU 的正确步骤为：

1. 使用 STC 通用 USB 转串口工具将待烧录 MCU 与电脑进行连接；
2. 打开 STC 的 ISP 下载软件；
3. 选择单片机型号；
4. 选择 STC 通用 USB 转串口工具所对应的串口；
5. 打开目标文件（HEX 格式或者 BIN 格式）；
6. 点击 ISP 下载软件中的“下载/编程”按钮；
7. 按一下 STC 通用 USB 转串口工具上的“电源开关”给 MCU 供电，即可开始下载。

【冷启动烧录】

此外，USB 接口与 Micro-USB 接口是相同的功能，用户根据需要连接其中一个接口到电脑即可。

编程接口的 0V 信号脚内部有 470 欧姆电阻接地，如果设置了 P1.0/P1.1=0/0 或者 P3.2/P3.3=0/0 时才能下载，可将 P1.0，P1.1 或者 P3.2，P3.3 接到 0V 信号脚。

H.4.3 STC 通用 USB 转串口工具驱动安装

STC 通用 USB 转串口工具采用 CH340 USB 转串口芯片(可以外挂晶振，更精准)，只要下载通用的 CH340 串口驱动程序进行安装即可，以下是 STC 官网 (www.STCMCUDATA.com) 提供的 CH341SER 串口驱动下载位置：



下载后进行解压，CH340 的驱动安装包路径 `stc-ispl-15xx-v6.87K\USB to UART Driver\CH340_CH341`：

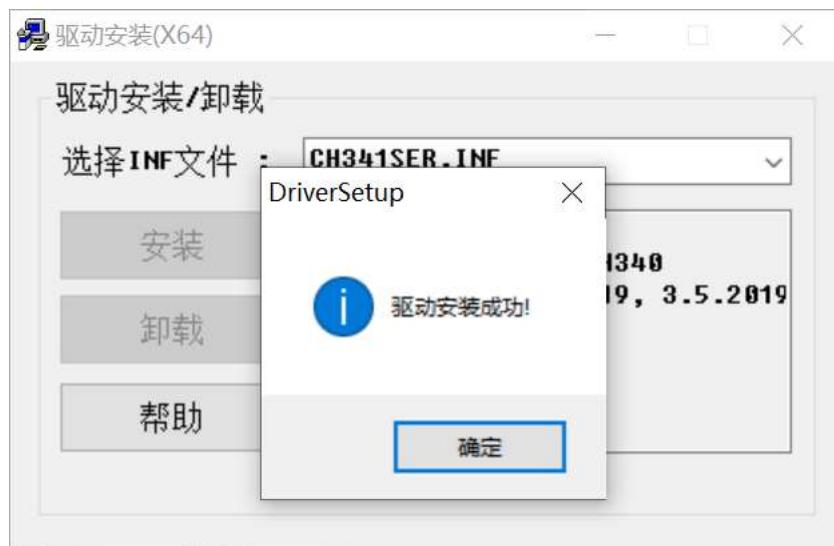
i > 下载 > `stc-ispl-15xx-v6.87K` > `USB to UART Driver` > `CH340_CH341`

名称	修改日期
<code>ch341ser</code>	2020/5/9 15:03

以 STC 官网提供的 CH341SER 串口驱动为例，双击“CH341SER.exe”安装包，在弹出的主界面点击“安装”按钮开始安装驱动程序：



然后弹出驱动安装成功对话框，点击“确定”按钮完成安装：

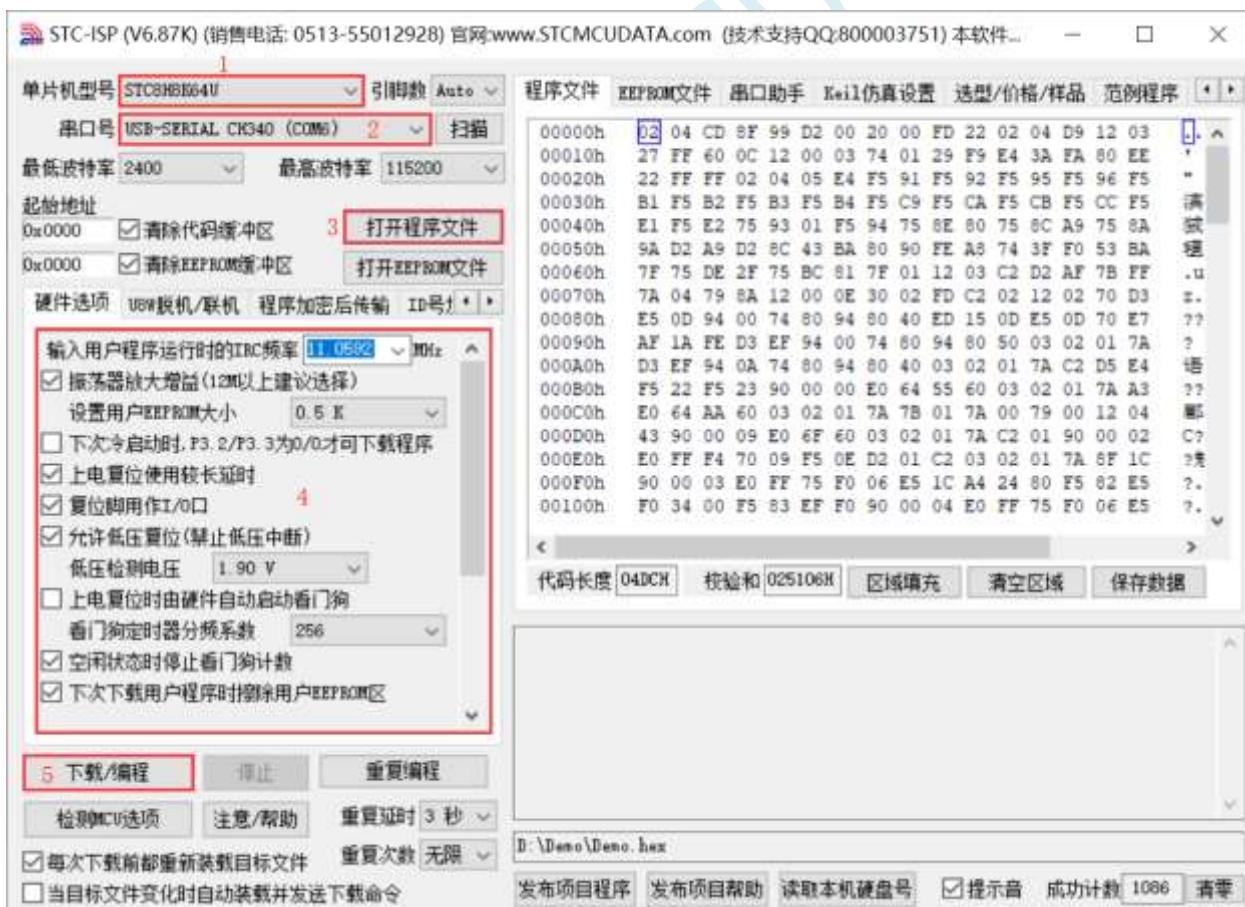


H.4.4 使用 STC 通用 USB 转串口工具下载程序到 MCU

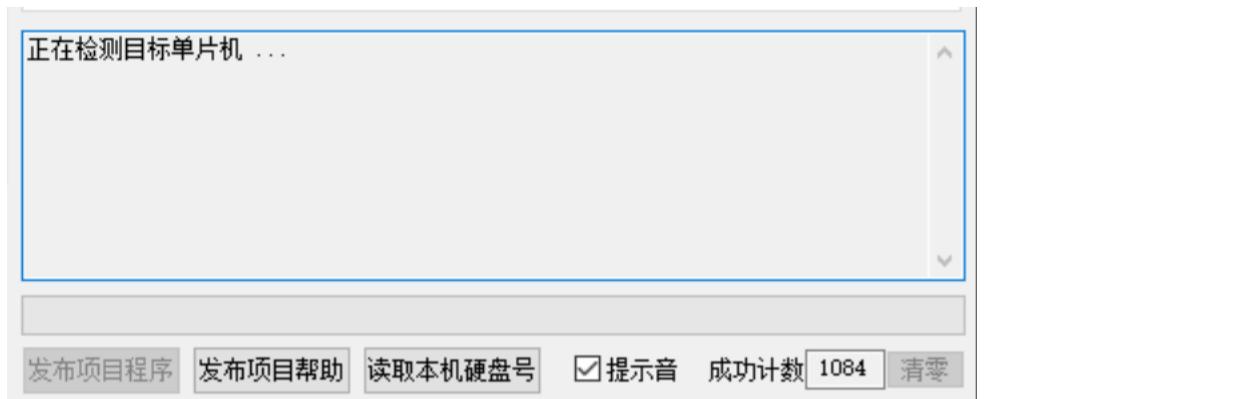
1. 使用 STC 通用 USB 转串口工具将待烧录 MCU 与电脑进行连接：



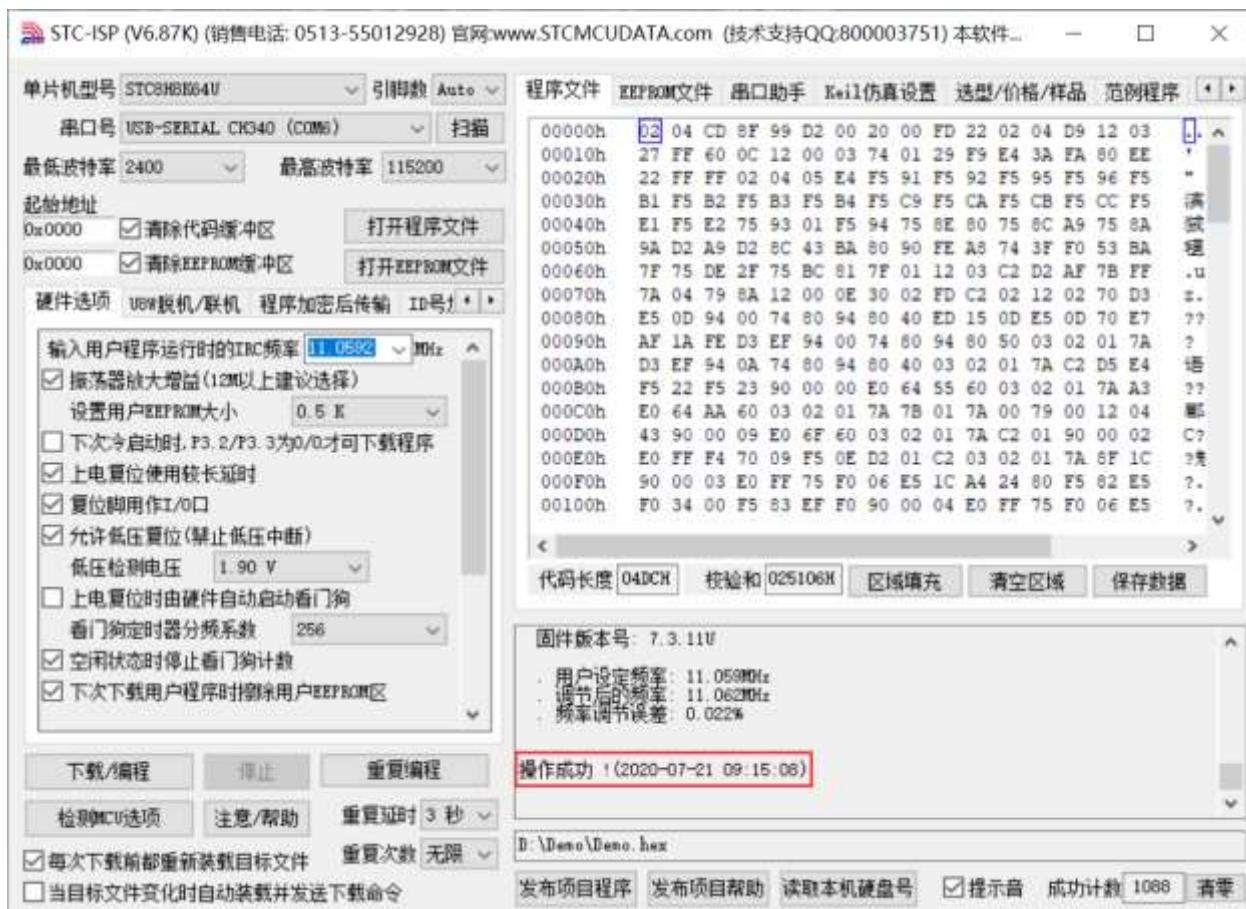
2. 打开 STC-ISP 软件;
3. 选择烧录芯片对应的型号;
4. 选择 STC 通用 USB 转串口工具所识别的串口号 (当 STC 通用 USB 转串口工具与电脑正确连接后, 软件会自动扫描并识别名称为“USB-SERIAL CH340 (COMx)”串口, 具体的 COM 编号会因电脑不同而不同)。当有多个 USB 转串口线与电脑相连时, 则必须手动选择;
5. 加载烧录程序;
6. 设置烧录选项;
7. 点击“下载/编程”按钮;



8. 右下角提示框显示“正在检测目标单片机 ...”时按一下 STC 通用 USB 转串口工具上的“电源开关”给 MCU 供电, 即可开始下载【冷启动烧录】;



9. 等待下载结束，若下载成功，右下角提示框会显示“操作成功！”。



H.4.5 使用 STC 通用 USB 转串口工具仿真用户代码

目前 STC 的仿真都是基于 Keil 环境的，所以若需要使用 STC 通用 USB 转串口工具仿真用户代码，则必须要安装 Keil 软件。

Keil 软件安装完成后，还需要安装 STC 的仿真驱动。STC 的仿真驱动的安装步骤如下：

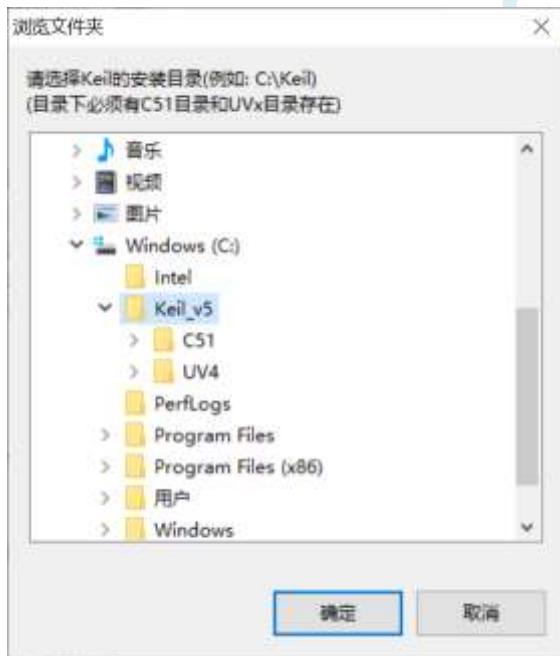
首先开 STC-ISP 下载软件；

然后在软件右边功能区的“Keil 仿真设置”页面中点击“添加型号和头文件到 Keil 中”添加 STC 仿真器

驱动到 Keil 中”按钮:

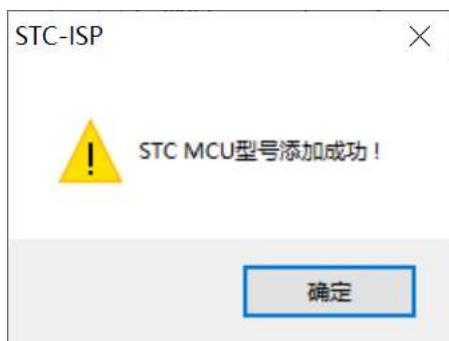


按下后会出现如下画面:

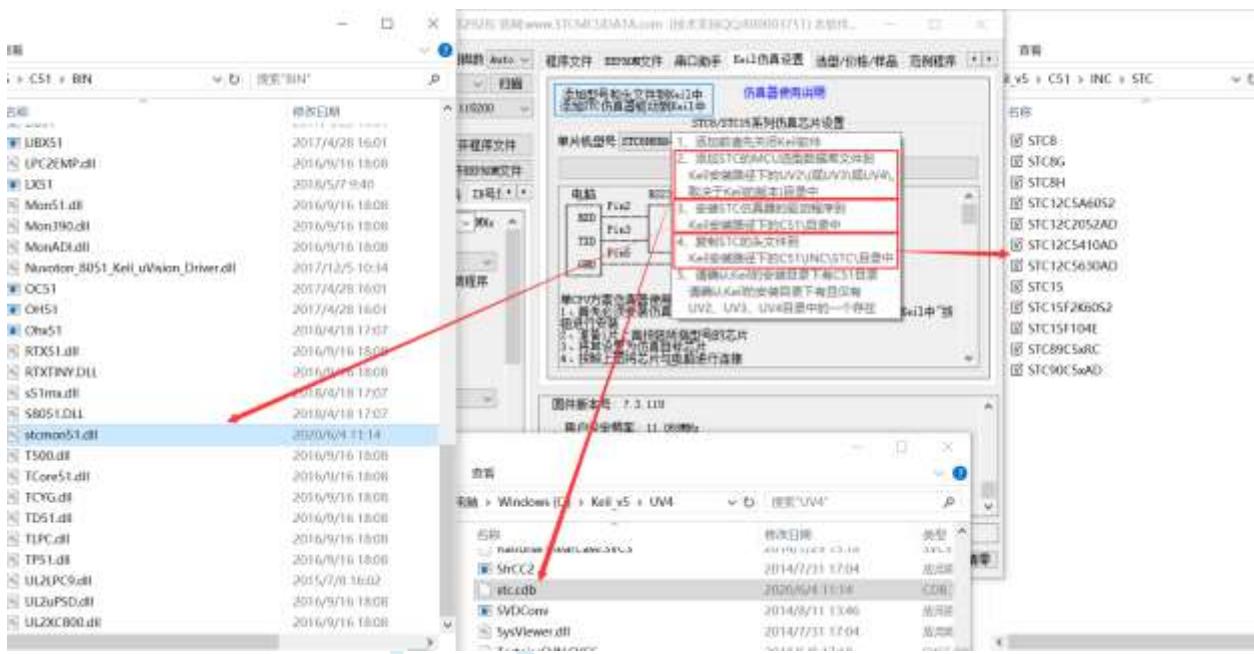


将目录定位到 Keil 软件的安装目录，然后确定。

安装成功后会弹出如下的提示框:



在 Keil 的相关目录中可以看到如下的文件，即表示驱动正确安装了。



由于在默认状态下，STC 的主控芯片并不是一颗仿真芯片，不具有仿真功能，所以若需要进行仿真，则还需要将 STC 的主控芯片设置为仿真芯片。

制作仿真芯片步骤如下：

首先使用 STC 通用 USB 转串口工具将 MCU 与电脑进行连接；

然后打开 STC 的 ISP 下载软件，并在串口号的下拉列表中选择串口工具所对应的串口号；

选择 MCU 单片机型号；

选择用户程序运行的 IRC 频率，制作仿真芯片时选择的频率与所仿真的用户程序所设置的频率一致，才能达到真实的运行效果。

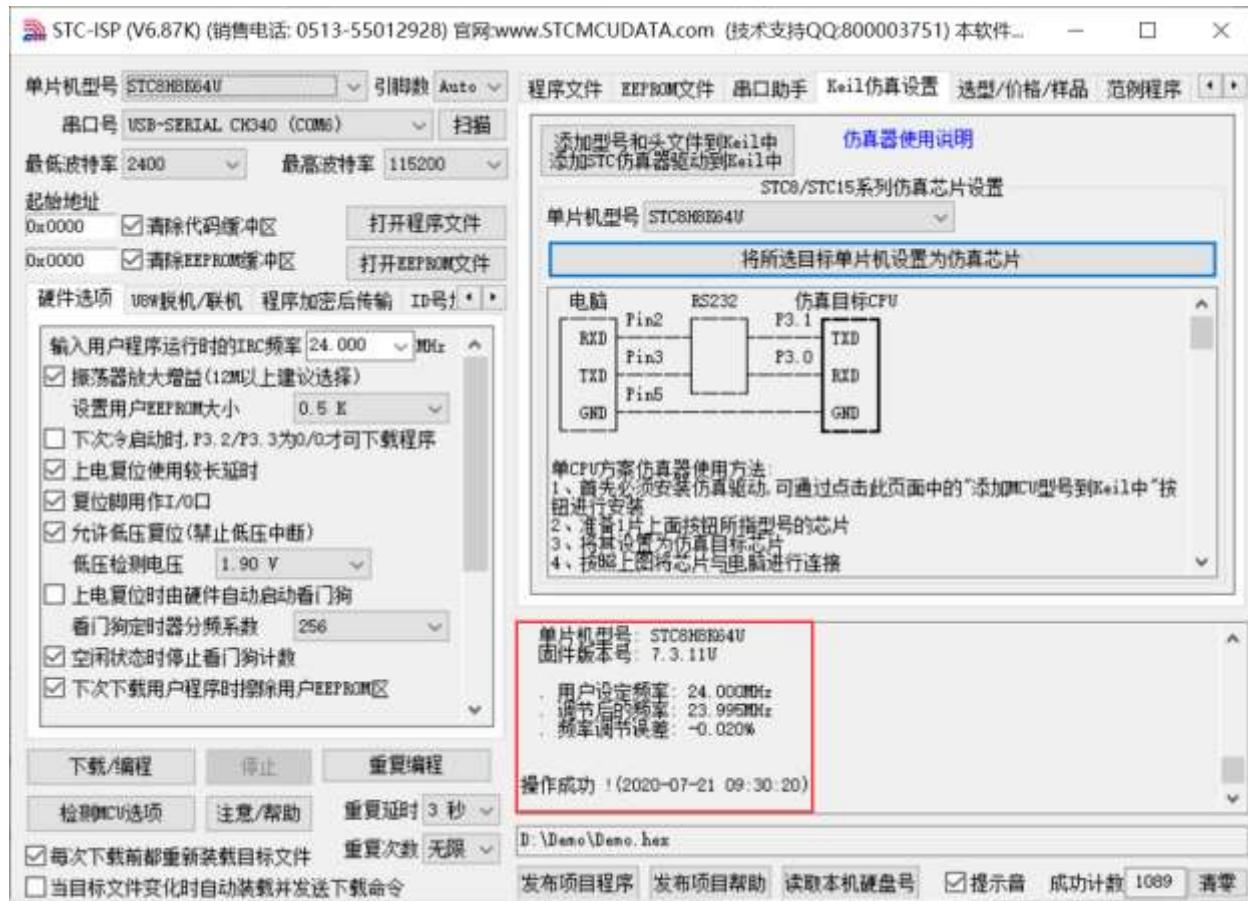


然后在软件右边功能区的“Keil 仿真设置”页面中点击“将所选目标单片机设置为仿真芯片”按钮，按下后会出现如下画面：



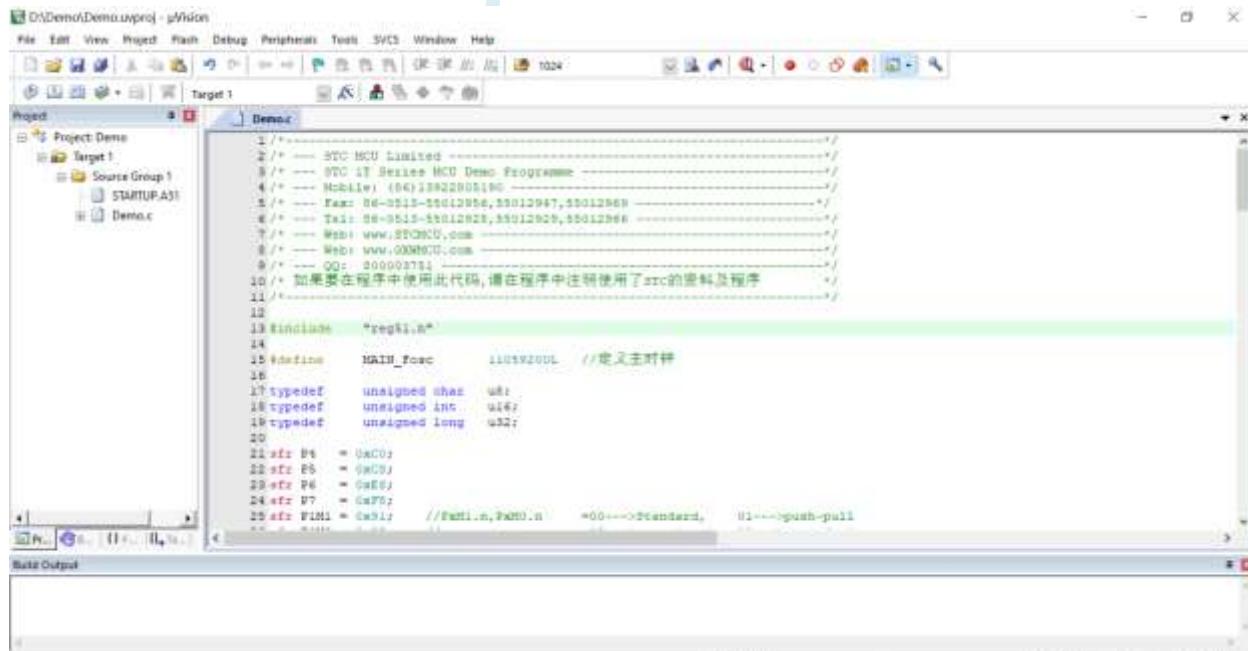
接下来需要按一下 STC 通用 USB 转串口工具上的“电源开关”给 MCU 供电【冷启动】，即可开始制作仿真芯片。

若设置成功，会出现如下的画面：



至此，仿真芯片便制作成功了。

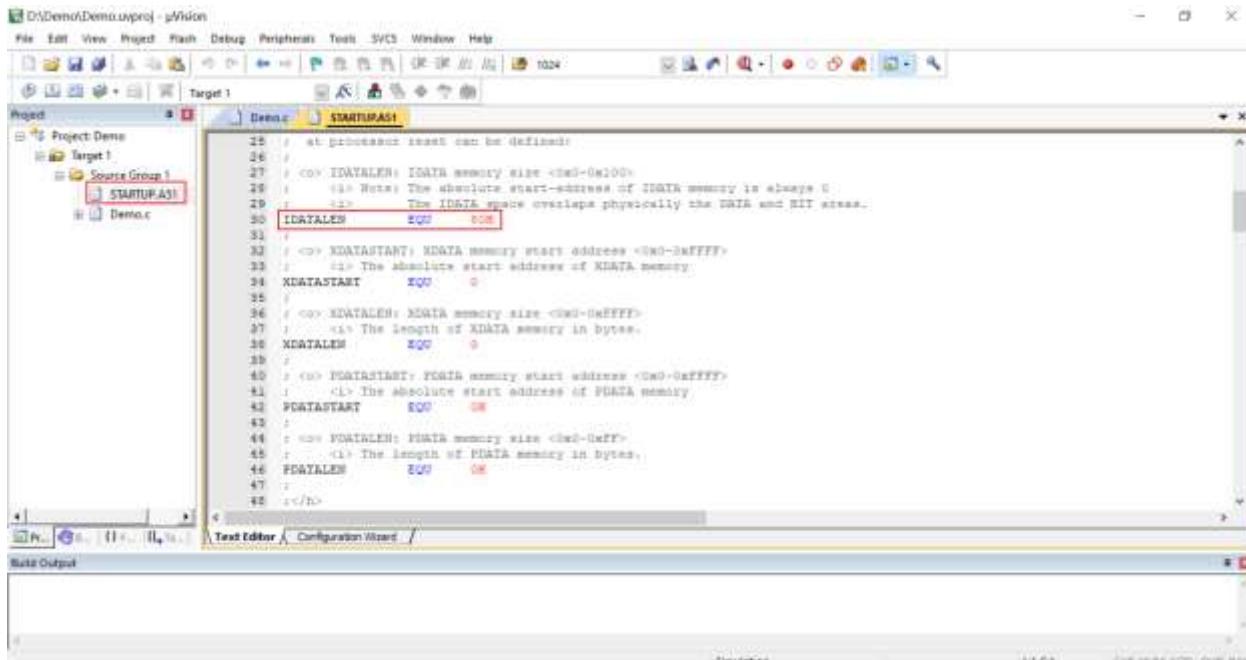
接下来我们打开一个项目进行仿真：



然后进行下面的项目设置。

附加说明一点:

当创建的是 C 语言项目，且有将启动文件“STARTUP.A51”添加到项目中时，里面有一个命名为“IDATALEN”的宏定义，它是用来定义 IDATA 大小的一个宏，默认值是 128，即十六进制的 80H，同时它也是启动文件中需要初始化为 0 的 IDATA 的大小。所以当 IDATA 定义为 80H，那么 STARTUP.A51 里面的代码则会将 IDATA 的 00-7F 的 RAM 初始化为 0；同样若将 IDATA 定义为 OFFH，则会将 IDATA 的 00-FF 的 RAM 初始化为 0。



我们所选的 STC8H 系列的单片机的 IDATA 大小为 256 字节（00-7F 的 DATA 和 80H-FFH 的 IDATA），但由于在 RAM 的最后 17 个字节有写入 ID 号以及相关的测试参数，若用户在程序中需要使用这一部分数据，则一定不要将 IDATALEN 定义为 256。

按下快捷键“Alt+F7”或者选择菜单“Project”中的“Option for Target ‘Target1’”

在“Option for Target ‘Target1’”对话框中对项目进行配置：

第 1 步、进入到项目的设置页面，选择“Debug”设置页；

第 2 步、选择右侧的硬件仿真“Use ...”；

第 3 步、在仿真驱动下拉列表中选择“STC Monitor-51 Driver”项；

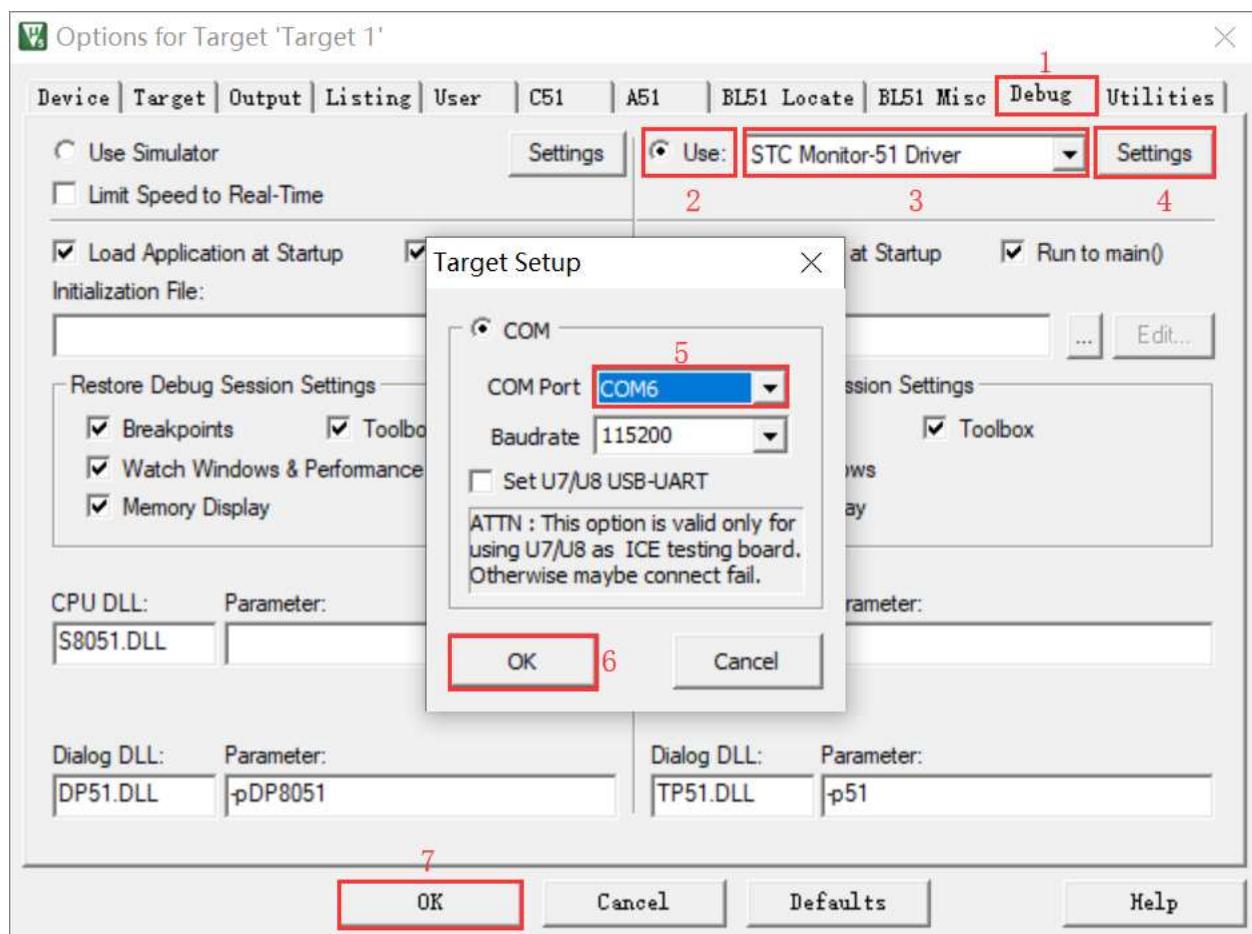
第 4 步、点击“Settings”按钮，进入串口的设置画面；

第 5 步、对串口的端口号和波特率进行设置，串口号要选择 STC 通用 USB 转串口工具所对应的串口，

波特率一般选择 115200 或者 57600。

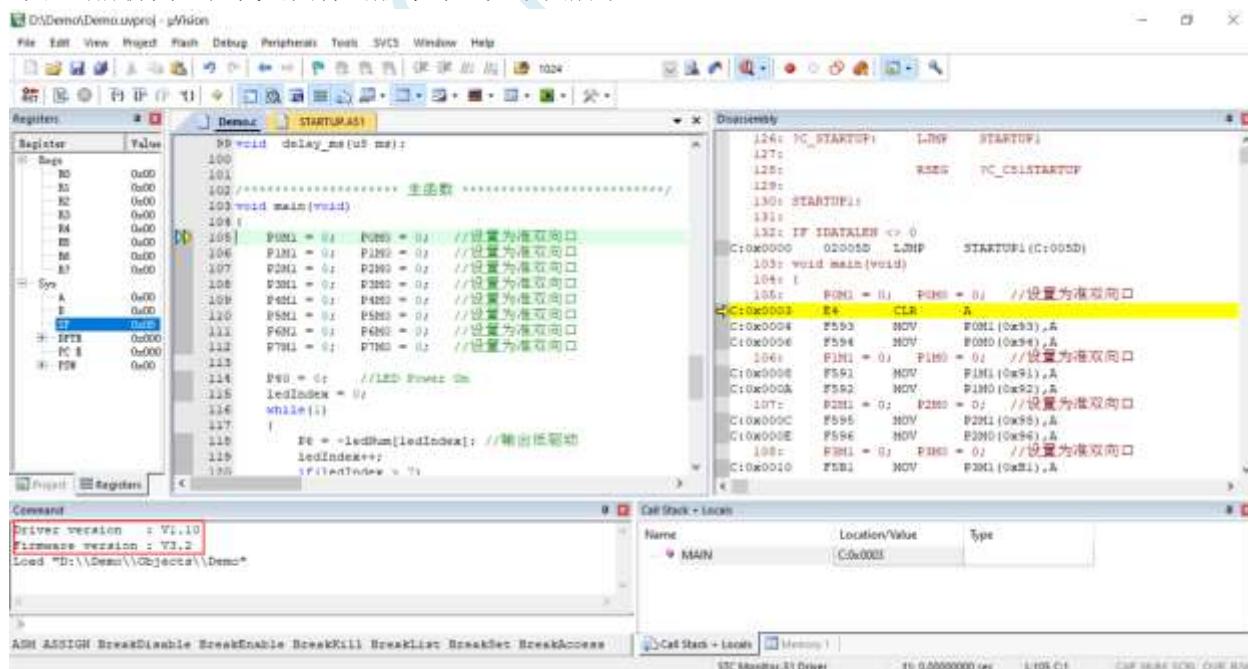
确定完成仿真设置。

详细步骤如下图所示：

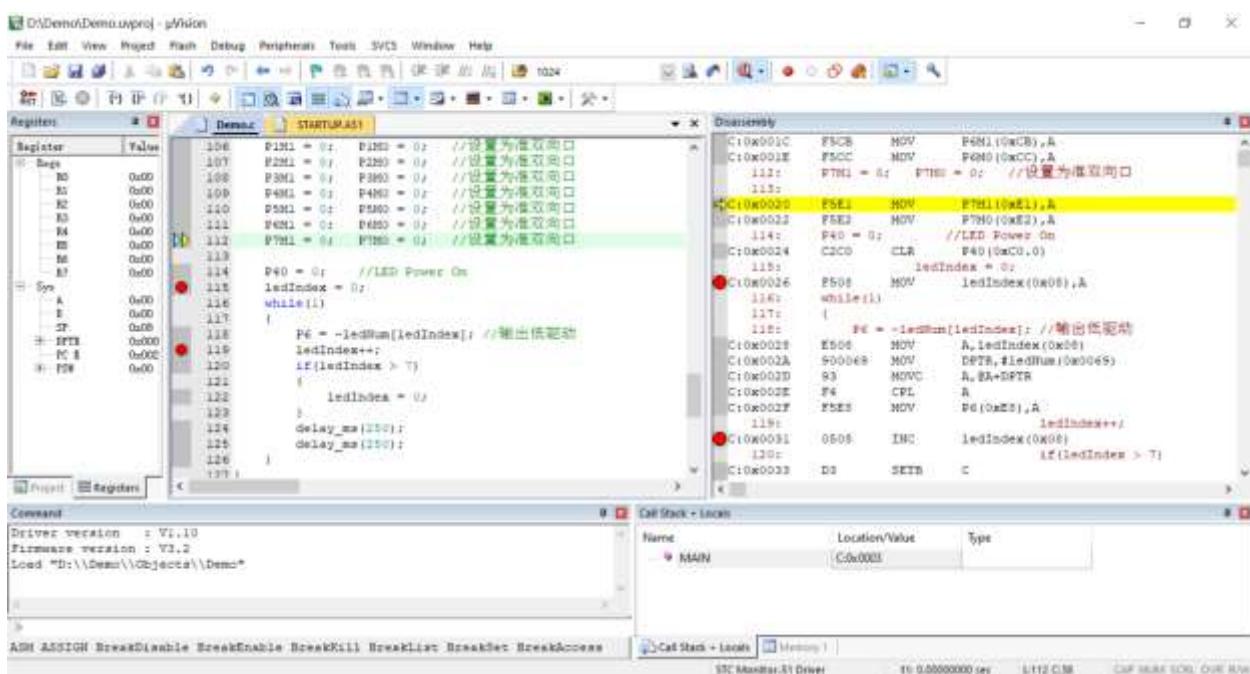


完成了上面所有的工作后，即可在 Keil 软件中按“Ctrl+F5”开始仿真调试。

若硬件连接无误的话，将会进入到类似于下面的调试界面，并在命令输出窗口显示当前的仿真驱动版本号和当前仿真监控代码固件的版本号，如下图所示：



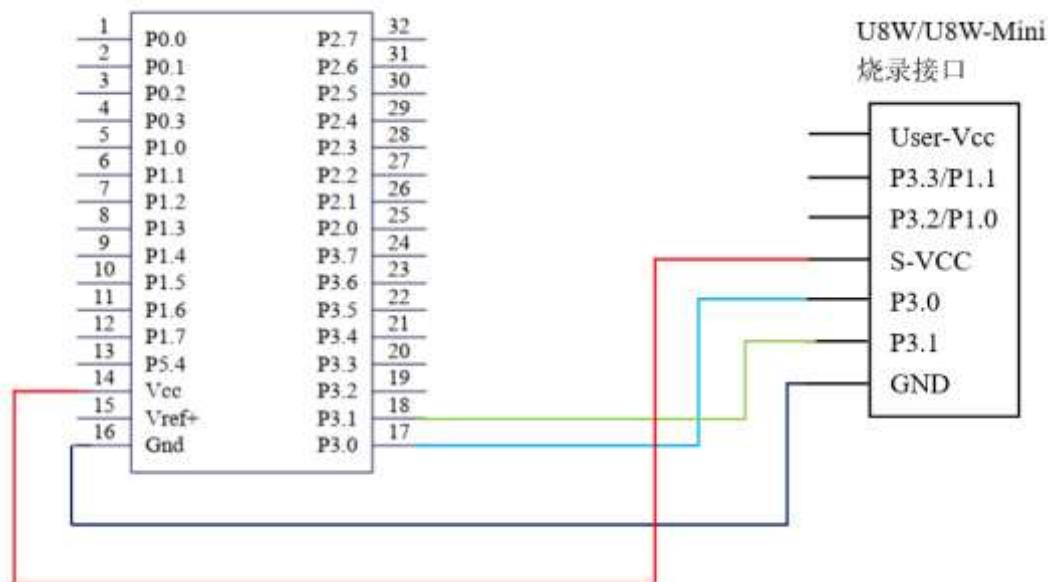
仿真调试过程中，可执行复位、全速运行、单步运行、设置断点等多中操作。



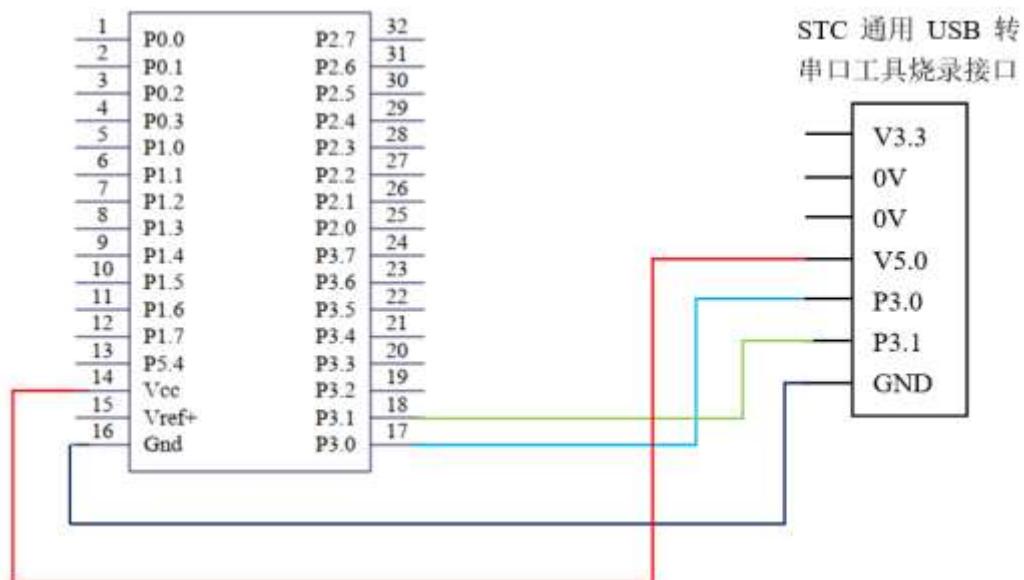
如上图所示，可在程序中设置多个断点，断点设置的个数目前最大允许 20 个
(理论上可设置任意个，但是断点设置得过多会影响调试的速度)。

H.5 应用线路图

H.5.1 U8W 工具应用参考线路图



H.5.2 STC 通用 USB 转串口工具应用参考线路图



附录I STC 仿真使用说明书

I.1 概述

STC8G/8H 系列单片机均支持在线仿真。支持包括下载用户代码、芯片复位、全速运行、单步运行、设置断点（理论断点个数为无限个，但为了提高仿真效率，目前限制为最多 20 个断点）、查看变量等基本的仿真操作，方便用户调试代码，查找代码中的逻辑错误，进而缩短项目开发周期。

仿真接口可为 USB 或者串口，单片机本身就是仿真器，不需要额外的仿真器即可实现全部的仿真功能。相应的 USB 口或者串口本为仿真专用端口，但当关闭仿真功能后，用户将可随意将仿真接口当作 GPIO、USB 或者串口进行使用。

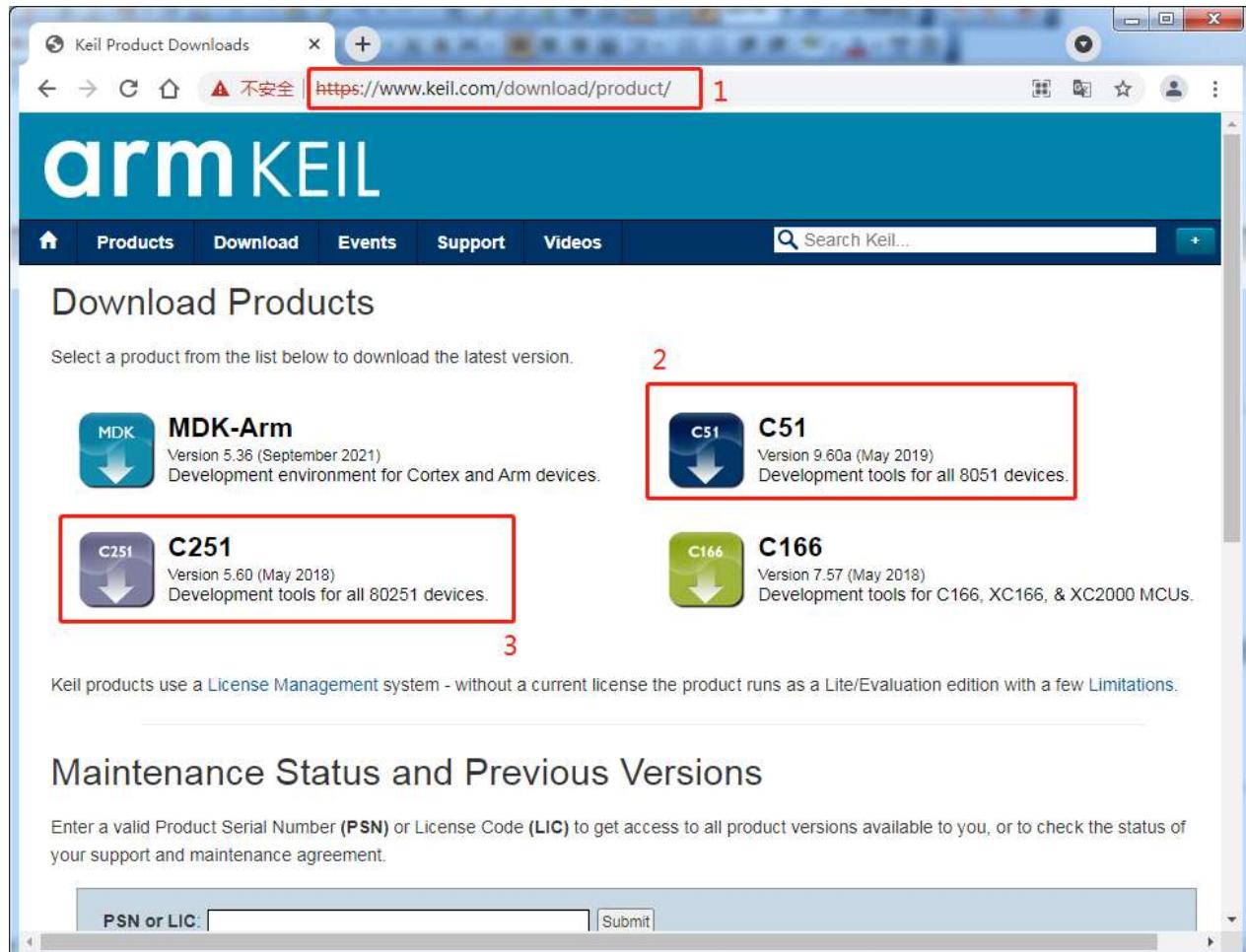
目前所有单片机的仿真模式均为软件监控仿真模式，会占用系统部分资源，各系列单片机仿真时所占用的资源如下表所示：

单片机系列	仿真接口	占用资源		
		端口	数据存储器 (XDATA)	程序存储器
STC8H8K64U 系列-B 版本	USB 串口	D+, D-	768 字节 (1D00H-1FFFH)	0 字节
		P3.0, P3.1	768 字节 (1D00H-1FFFH)	0 字节
		P3.6, P3.7		
		P1.6, P1.7		
		P4.3, P4.4		
STC8H8K64U 系列-A 版本	串口	P3.0, P3.1	768 字节 (1D00H-1FFFH)	0 字节
		P3.6, P3.7		
		P1.6, P1.7		
		P4.3, P4.4		
STC8H4K64T 系列	串口	P3.0, P3.1	768 字节 (0D00H-0FFFH)	0 字节
STC8H3K64S4 系列	串口	P3.0, P3.1	768 字节 (0900H-0BFFFH)	0 字节
STC8H1K16 系列	串口	P3.0, P3.1	768 字节 (0100H-03FFH)	0 字节
STC8H1K08 系列	串口	P3.0, P3.1	768 字节 (0100H-03FFH)	0 字节
STC8G2K64S4 系列	串口	P3.0, P3.1	768 字节 (0500H-07FFH)	0 字节
STC8G1K08 系列	串口	P3.0, P3.1	768 字节 (0100H-03FFH)	0 字节
STC8C2K64S4 系列	串口	P3.0, P3.1	768 字节 (0500H-07FFH)	0 字节
STC8A8K64D4 系列	串口	P3.0, P3.1	768 字节 (1D00H-1FFFH)	0 字节
STC8A8K64S4A12 系列	串口	P3.0, P3.1	768 字节 (1D00H-1FFFH)	0 字节
STC8F2K64S4 系列	串口	P3.0, P3.1	768 字节 (0500H-07FFH)	0 字节
STC8F1K08S2 系列	串口	P3.0, P3.1	768 字节 (0100H-03FFH)	0 字节
IAP15W4K58S4	串口	P3.0, P3.1	768 字节 (0D00H-0FFFH)	0 字节
IAP15F2K61S2	串口	P3.0, P3.1	768 字节 (0500H-07FFH)	0 字节

I.2 安装 Keil 软件

STC 单片机的仿真基于 Keil 开发环境，所以在进行仿真前，必须先安装 Keil 软件。

可在下图所示的地址下载 C51 和 C251 开发包



注意：最新的 Keil-UV5 软件默认是不包含 8051 和 80251 的工具包的，必须手动下载并安装。

I.3 安装仿真驱动

安装完成 Keil 开发环境后，还需要安装 STC 专用仿真驱动程序。

步骤如下：

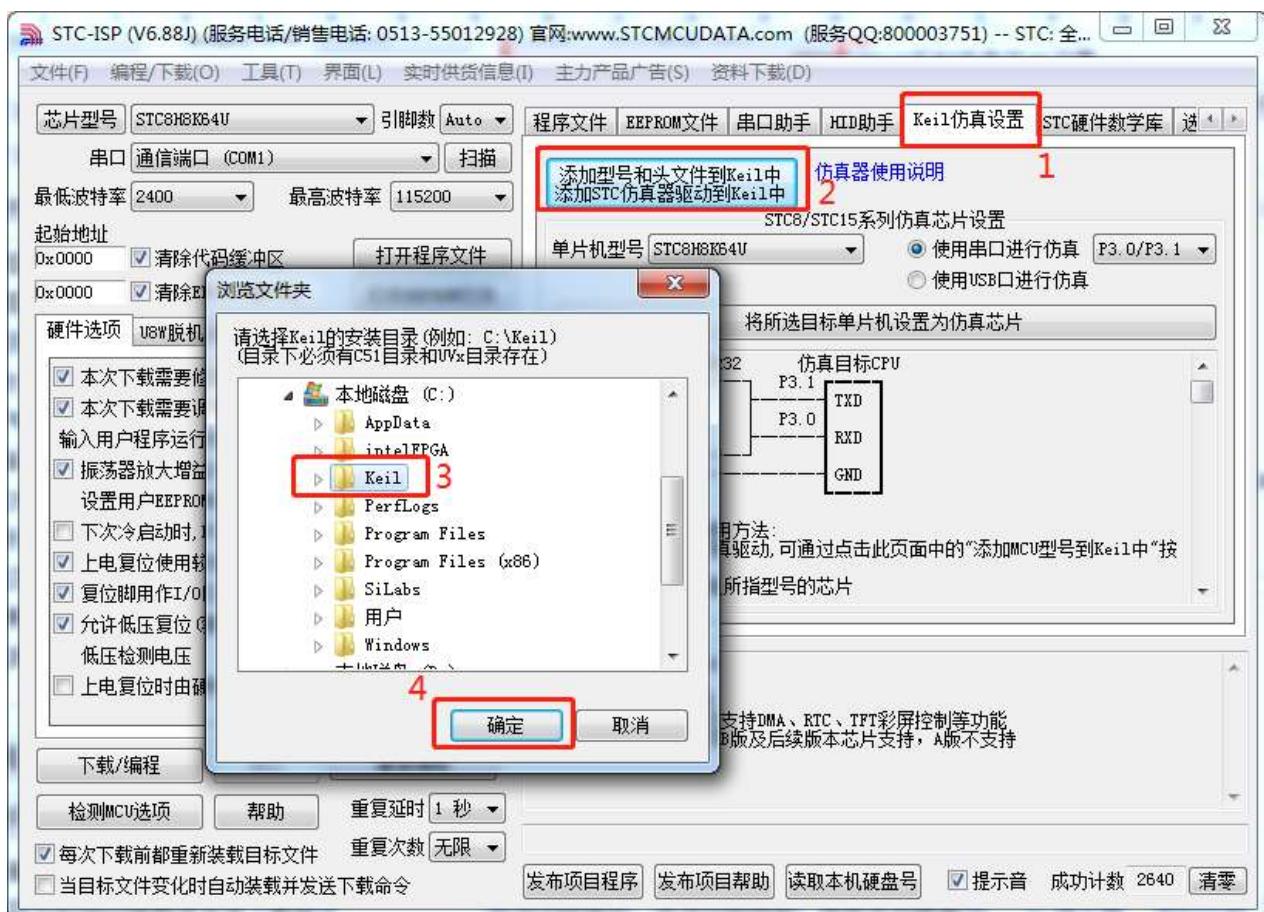
首先从 STC 官网下载最新的 STC-ISP 下载软件

The screenshot shows the official website for STCMCUData.com. The URL 'stcmcudata.com' is highlighted in a red box in the browser's address bar. The main content area displays information about the latest STC-ISP software version V6.88J, including its features (supporting color screen/ADC/4 serial ports/SPI) and price (RMB1.98). To the left, there is a sidebar for 'STC MCU 资料下载' (STC MCU Documentation Download) which lists various documentation links. To the right, there is a sidebar for 'STC-ISP 下载编程烧录软件' (STC-ISP Download Programming and Flashing Software) which lists the software version and its features.

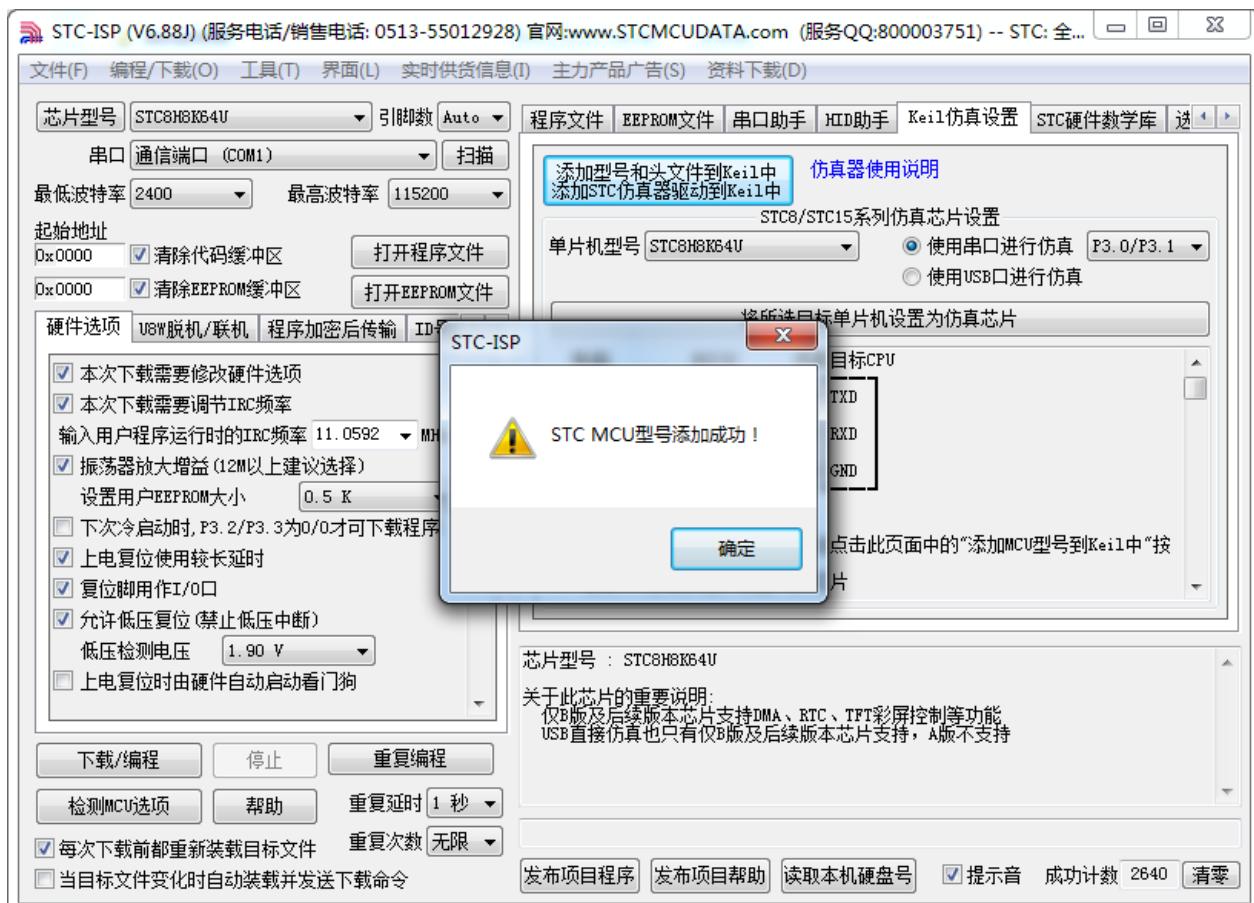
下载并解压完成后，打开软件包中的“stc-isp-vxx.exe”可执行文件

名称	修改日期	类型	大小
STC-USB Driver	2014/8/29 18:17	文件夹	
USB to UART Driver	2014/10/9 11:54	文件夹	
readme.txt	2020/6/9 14:43	文本文档	1 KB
stc-isp-v6.88J.exe	2021/10/20 17:07	应用程序	2,114 KB
STC-USB 驱动安装说明.pdf	2020/6/9 14:27	Foxit Reader PD...	3,585 KB

点击下载软件“Keil 仿真设置”页面中的“添加型号和头文件...”按钮（如下图“2”）



在弹出的“浏览文件夹”窗口中，选中 Keil 的安装目录（一般 Keil 的安装目录为“c:\keil”），点击确定后，若弹出“STC MCU 型号添加成功”则表示驱动已安装完成。



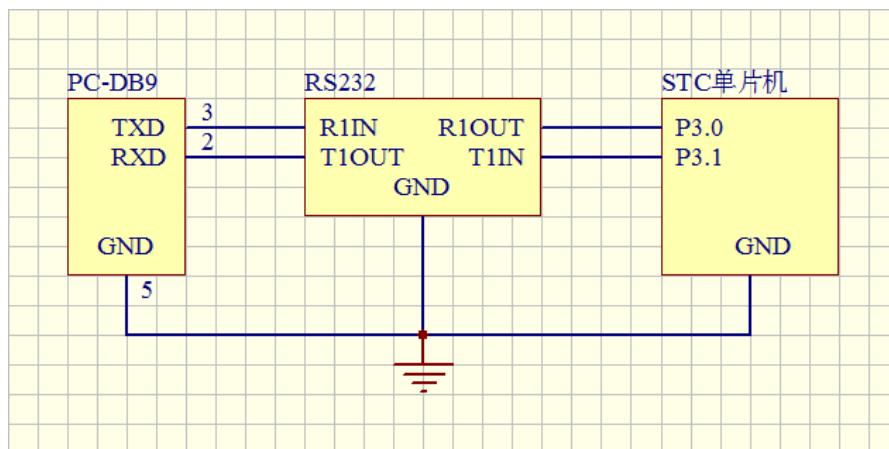
I.4 串口直接仿真

I.4.1 制作串口仿真芯片

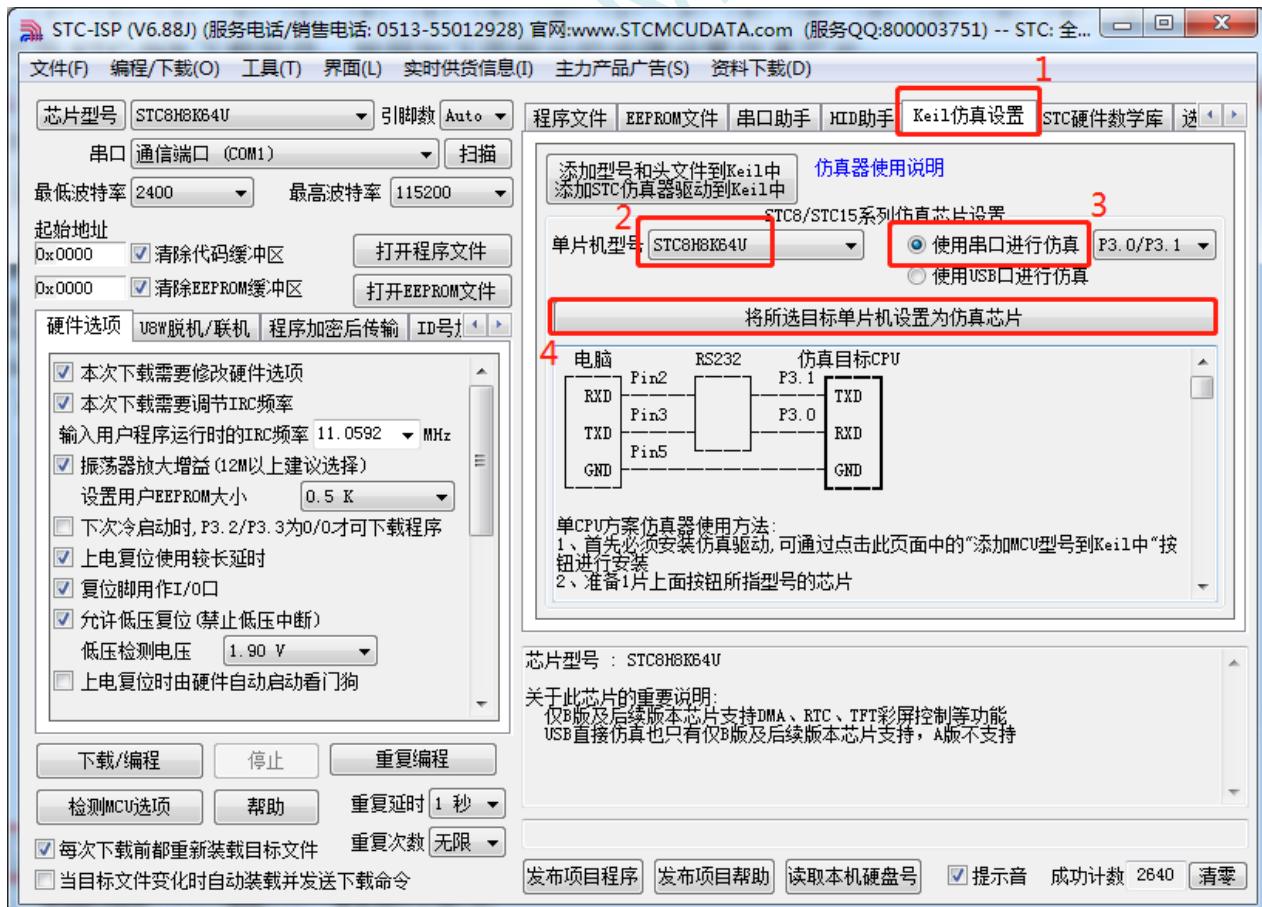
STC 单片机出厂时，仿真功能默认是关闭的，若要使用仿真功能，则需使用 STC-ISP 下载软件将目标单片机设置为仿真芯片。

设置步骤如下：

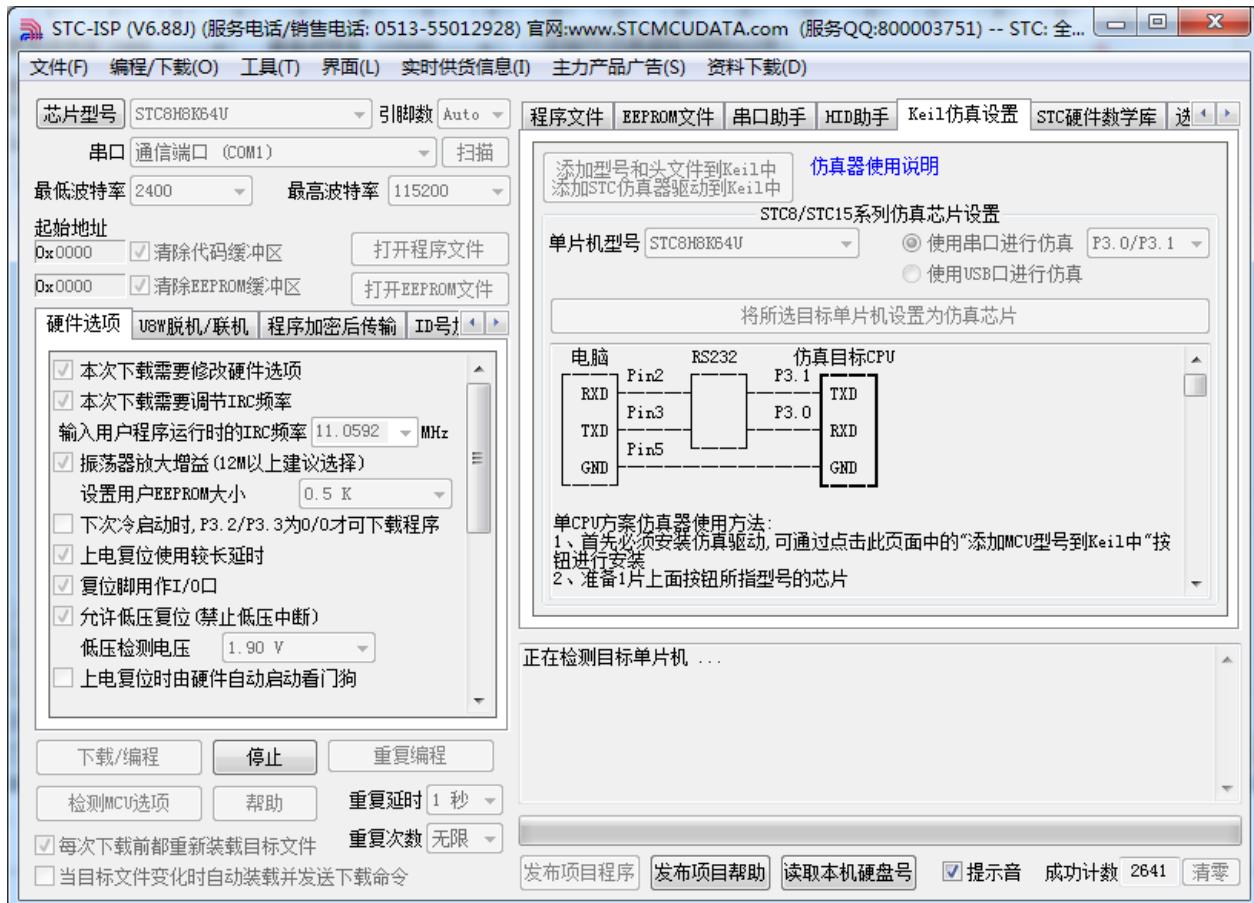
首先将目标芯片如下图所示的方式和电脑的串口连接在一起，并将单片机断电



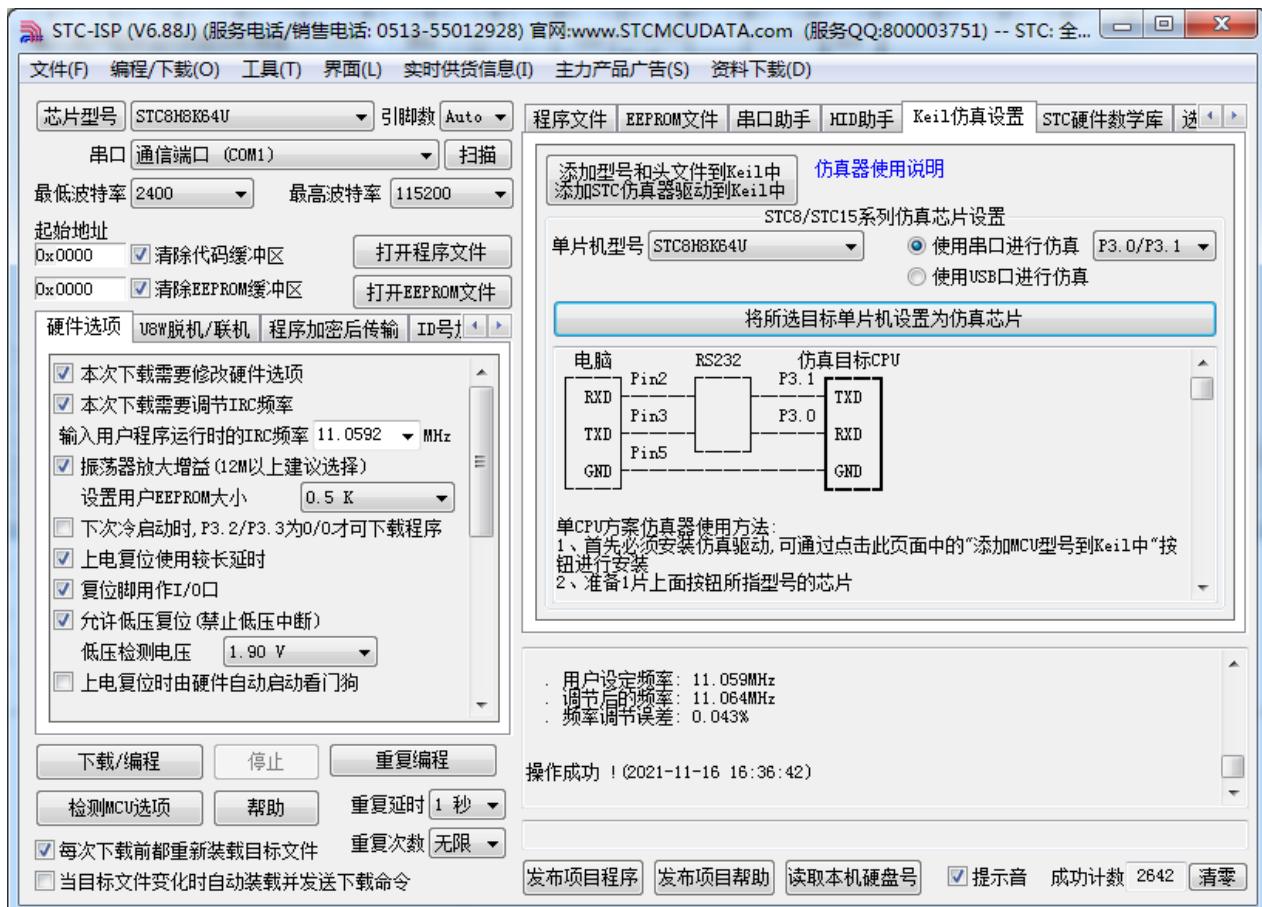
打开 STC-ISP 下载软件，按照如下图所示的步骤设置仿真芯片



当出现如下画面时，再给单片机上电

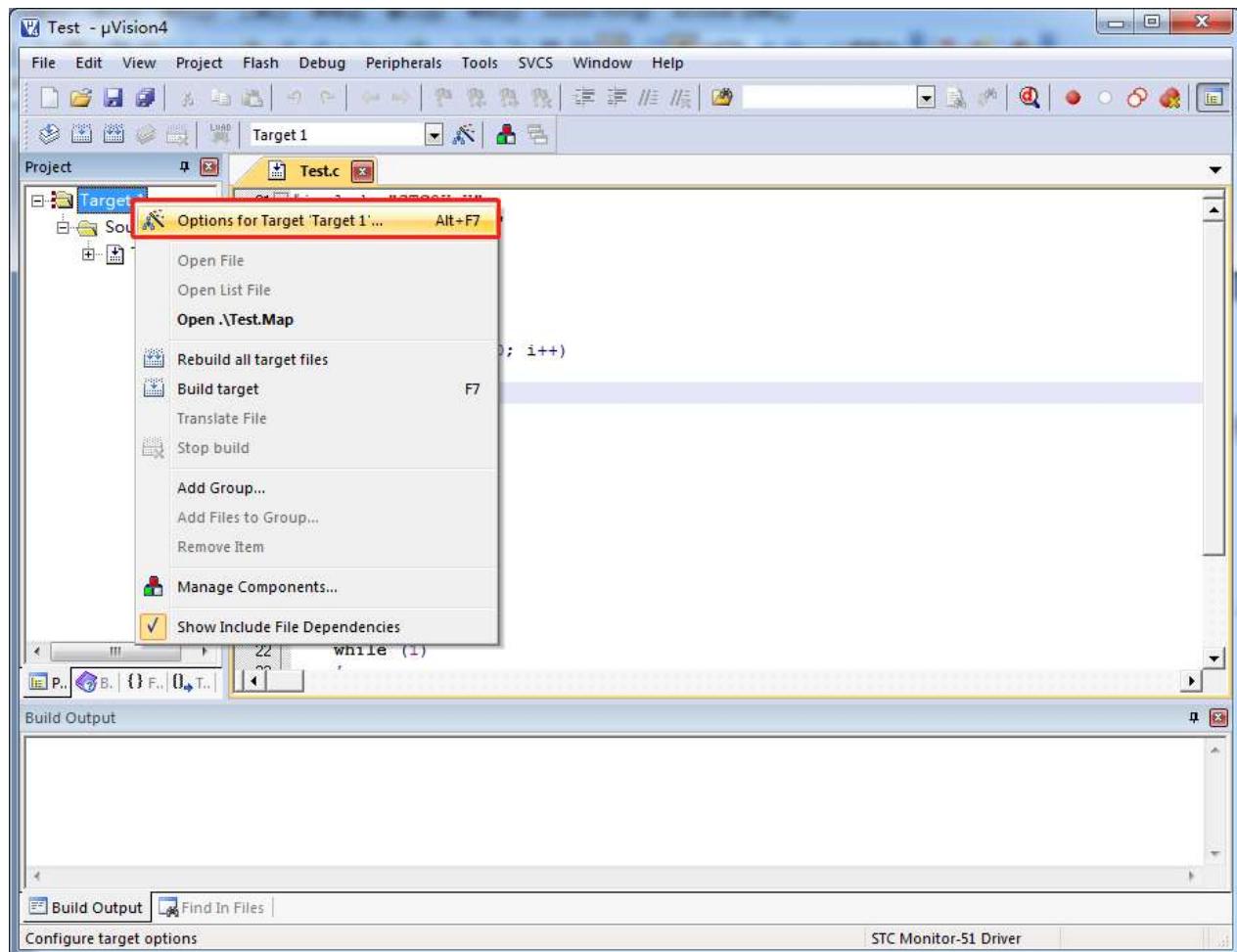


下载完成后，仿真芯片即制作完成

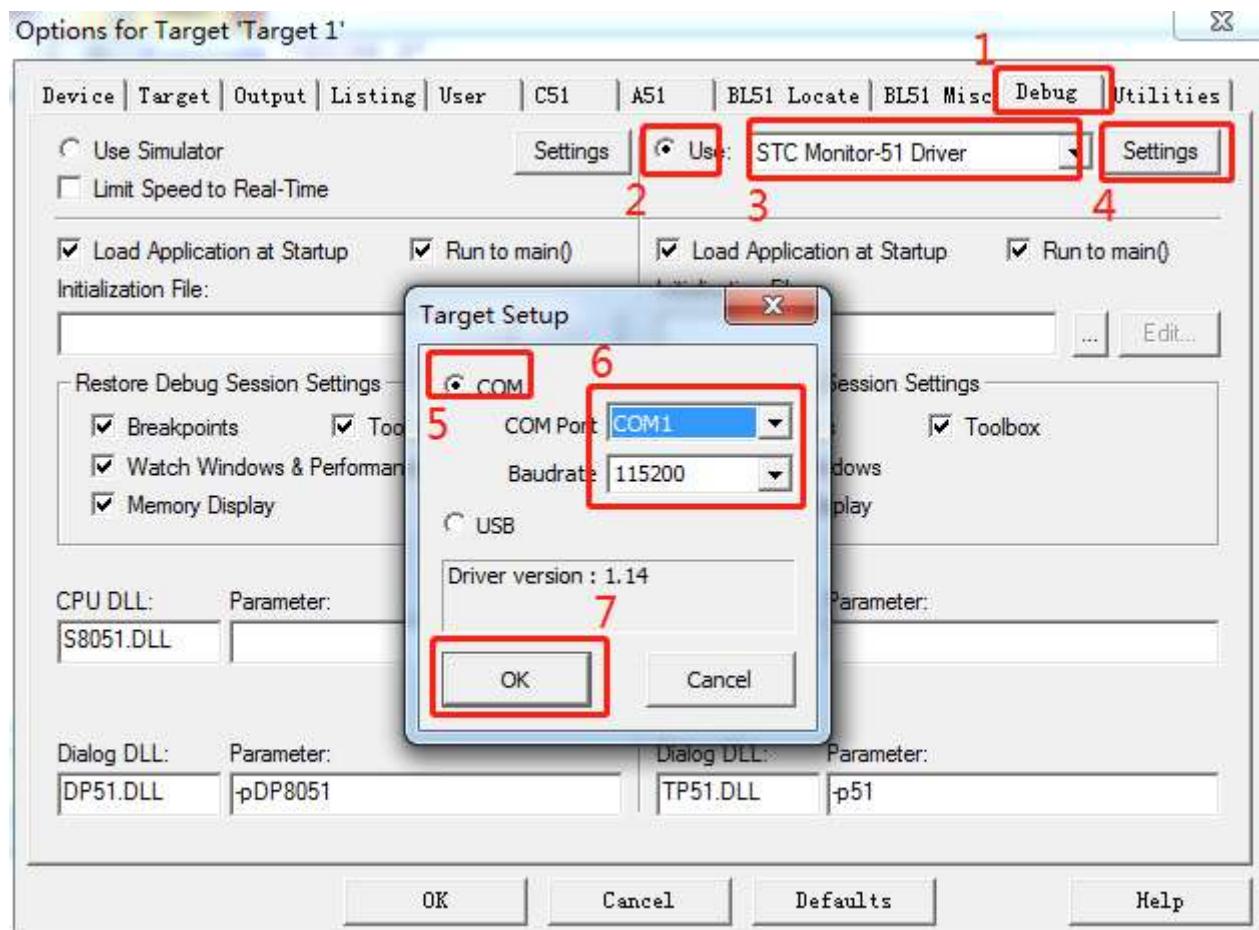


I.4.2 在 Keil 软件中进行串口仿真设置

在 Keil 软件中打开项目文件，并在下图所示的右键菜单中点击“Options for ...”



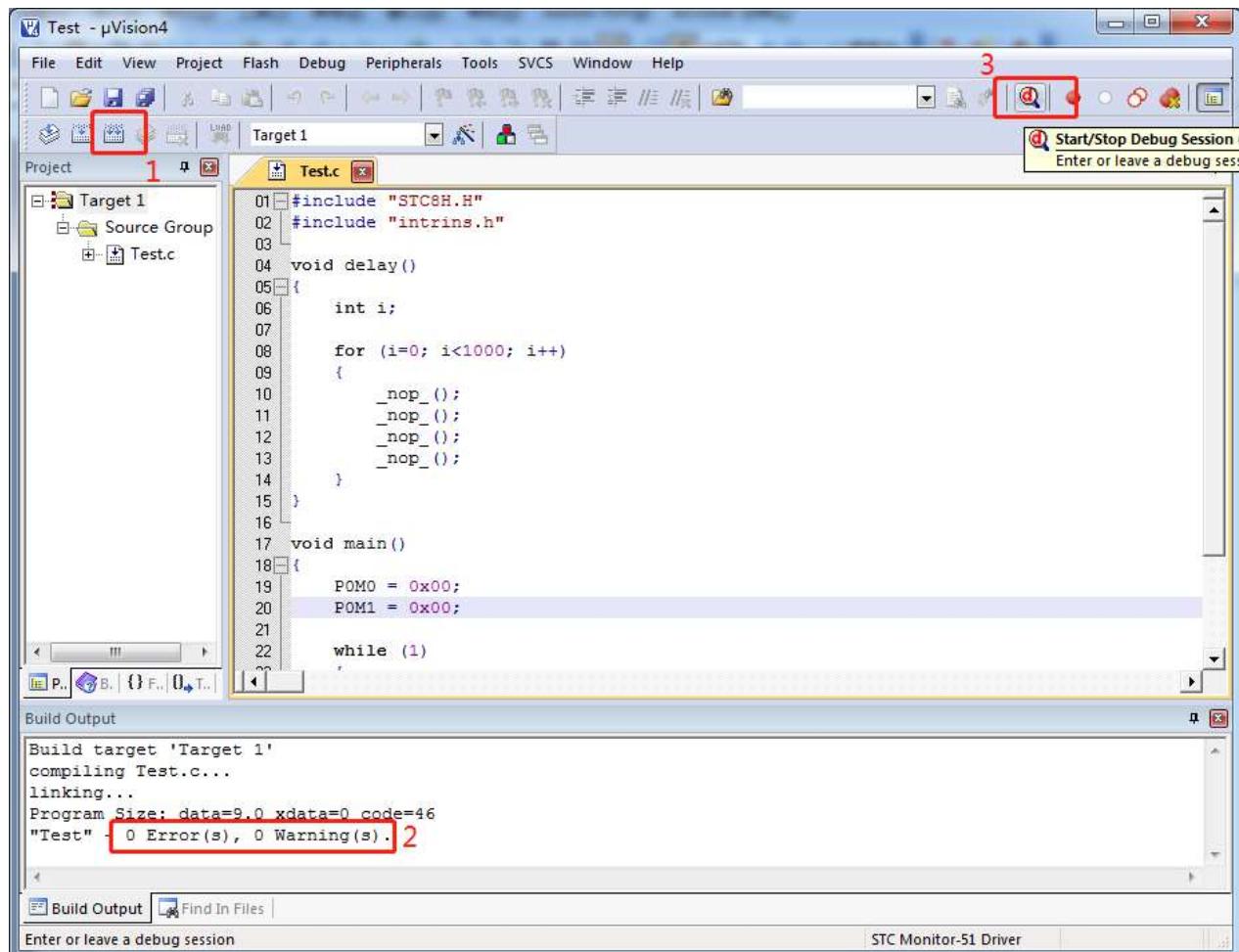
在项目选项中, 按如下图所示的步骤进行串口仿真设置

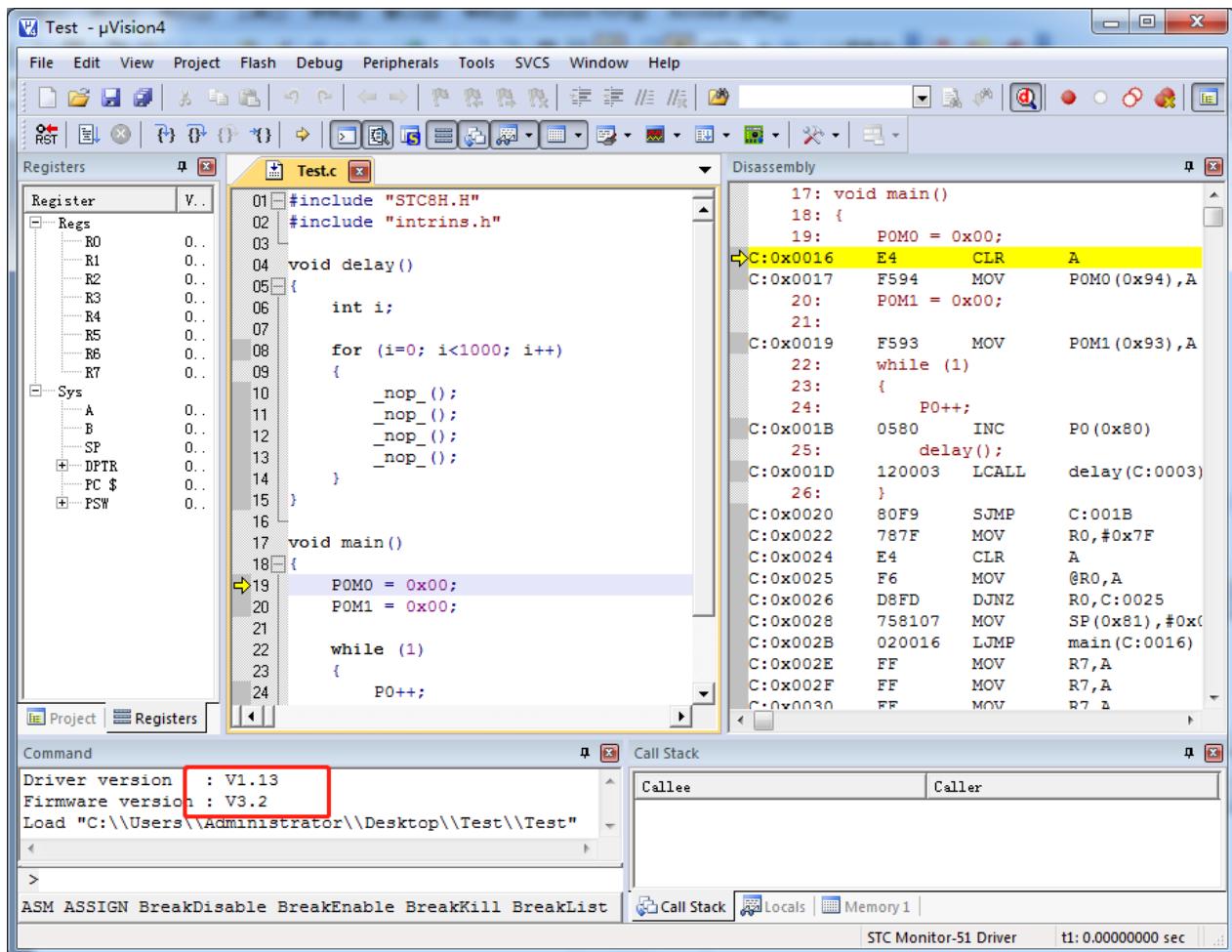


注意: 串口请根据实际的连接进行选择, 波特率一般选择 115200

I.4.3 在 Keil 软件中使用串口进行仿真

在 Keil 环境下，编辑完成源代码，并编译无误后，即可开始仿真





若芯片制作和连接均无误，则会如上图所示显示仿真驱动版本，并可正确下载用户代码到单片机，接下来便可进行运行、单步、断点等调试功能了。

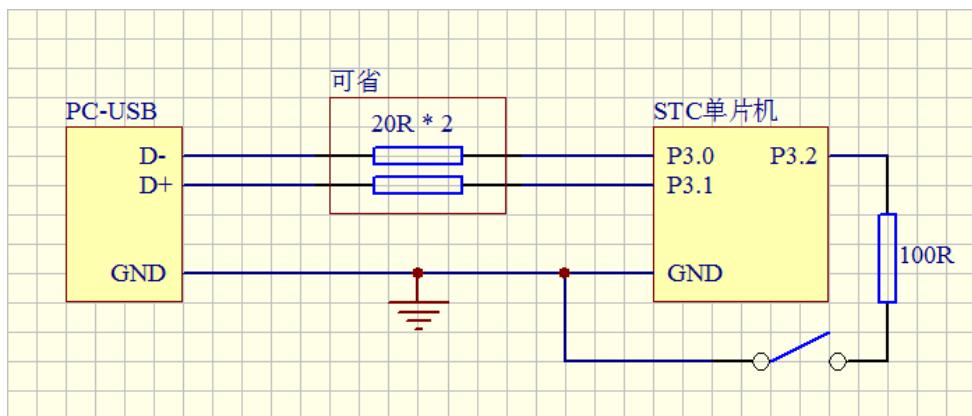
I.5 USB 直接仿真(目前只有 STC8H8K64U-B 版本芯片支持)

I.5.1 制作 USB 仿真芯片

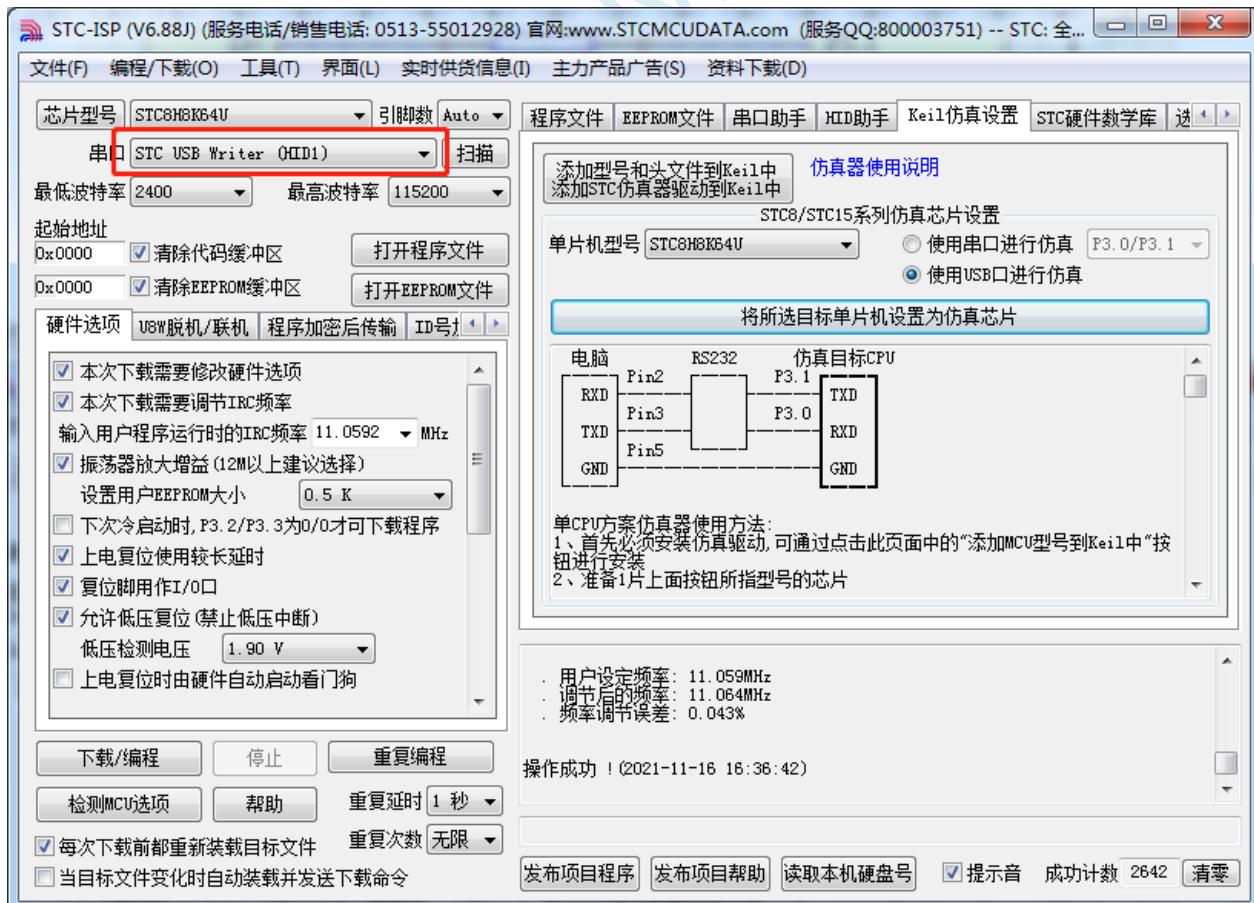
制作 USB 仿真芯片，可按照 4.1 小节的步骤，使用串口 ISP 制作，也可以使用 USB-ISP 的方法制作，本小节将介绍如何使用 USB-ISP 制作。

设置步骤如下：

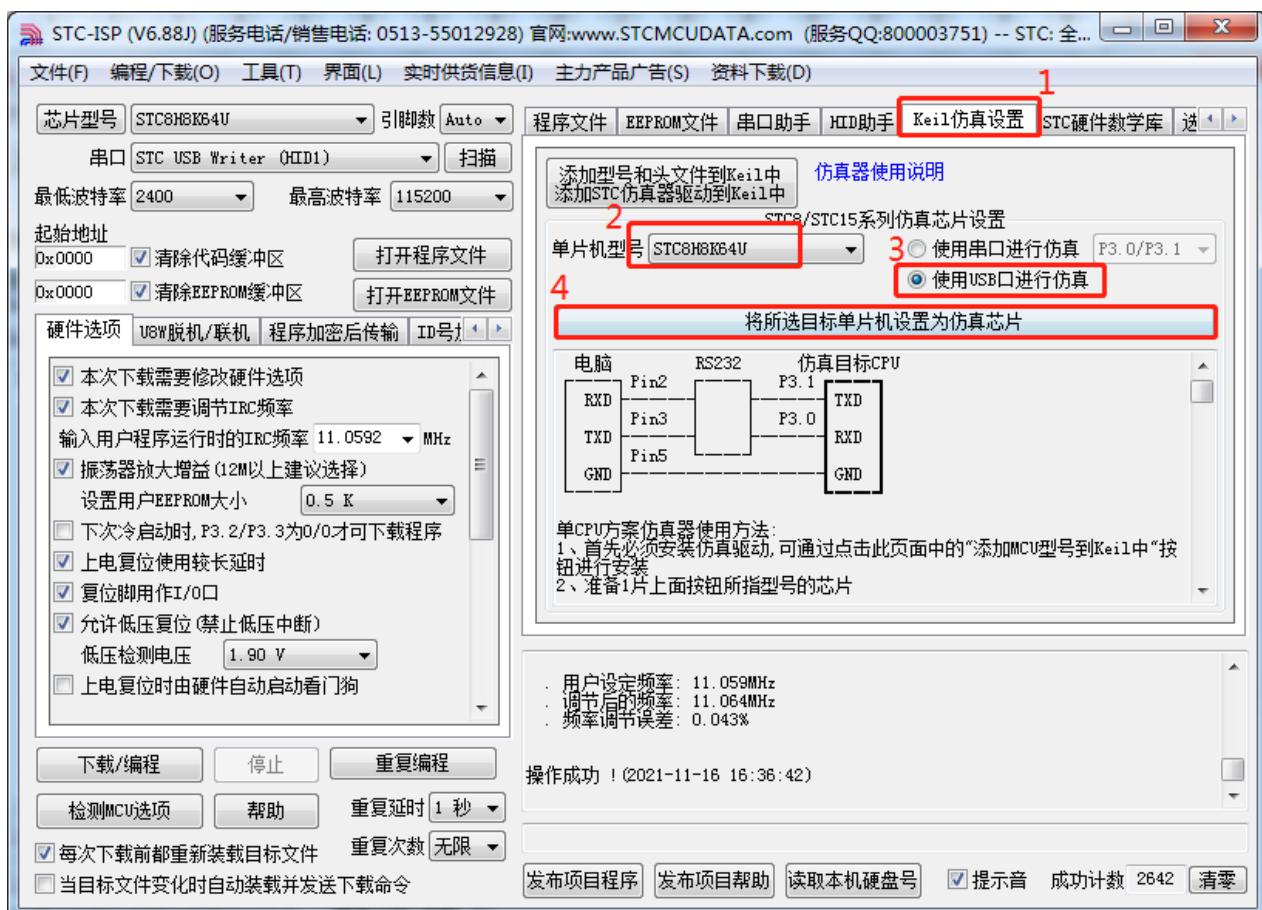
首先将目标芯片如下图所示的方式和电脑的串口连接在一起，并将 P3.2 短路通过开关连接到 GND，然后给单片机上电



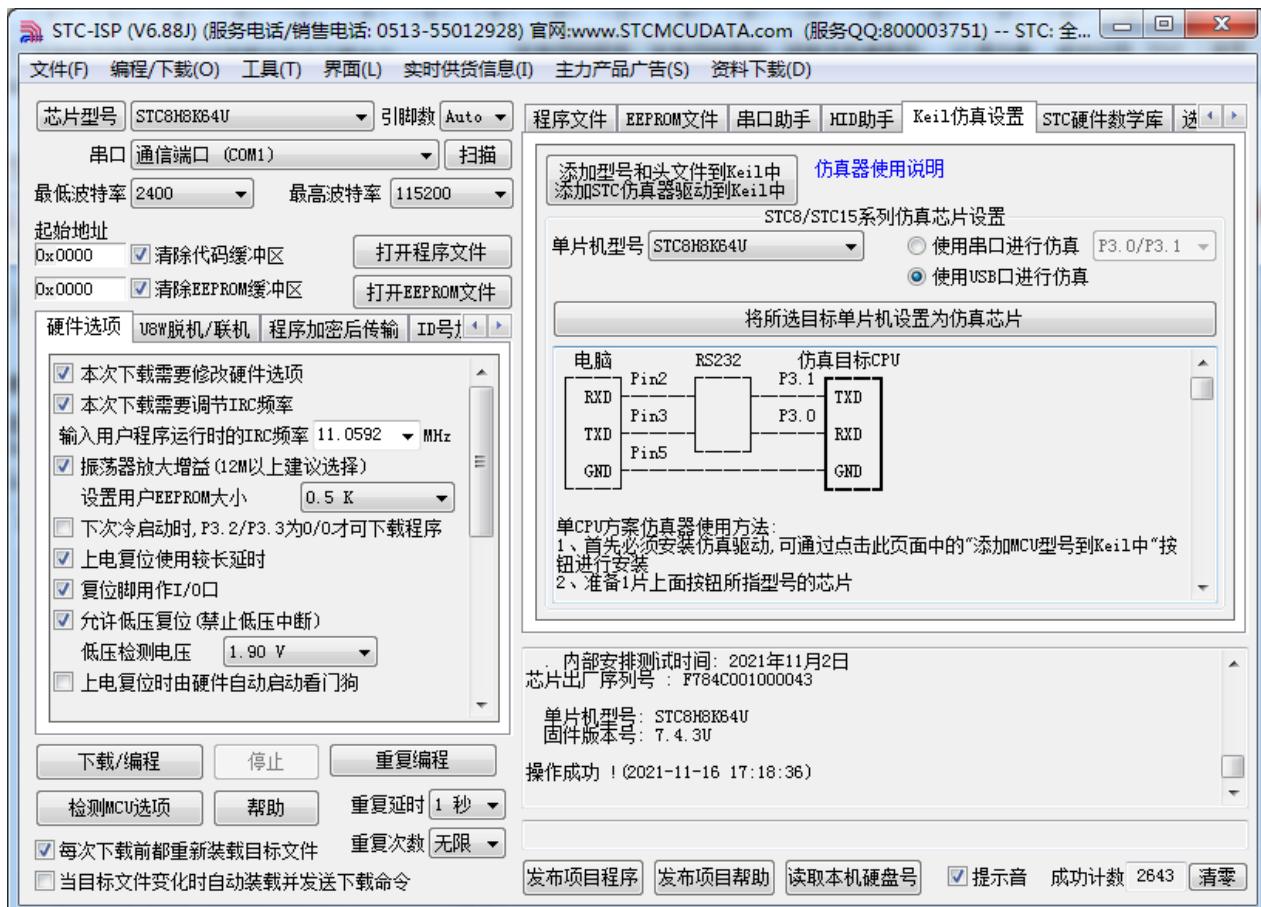
若在 ISP 软件中能自动扫描到“STC USB Writer (HID1)”表示连接正确



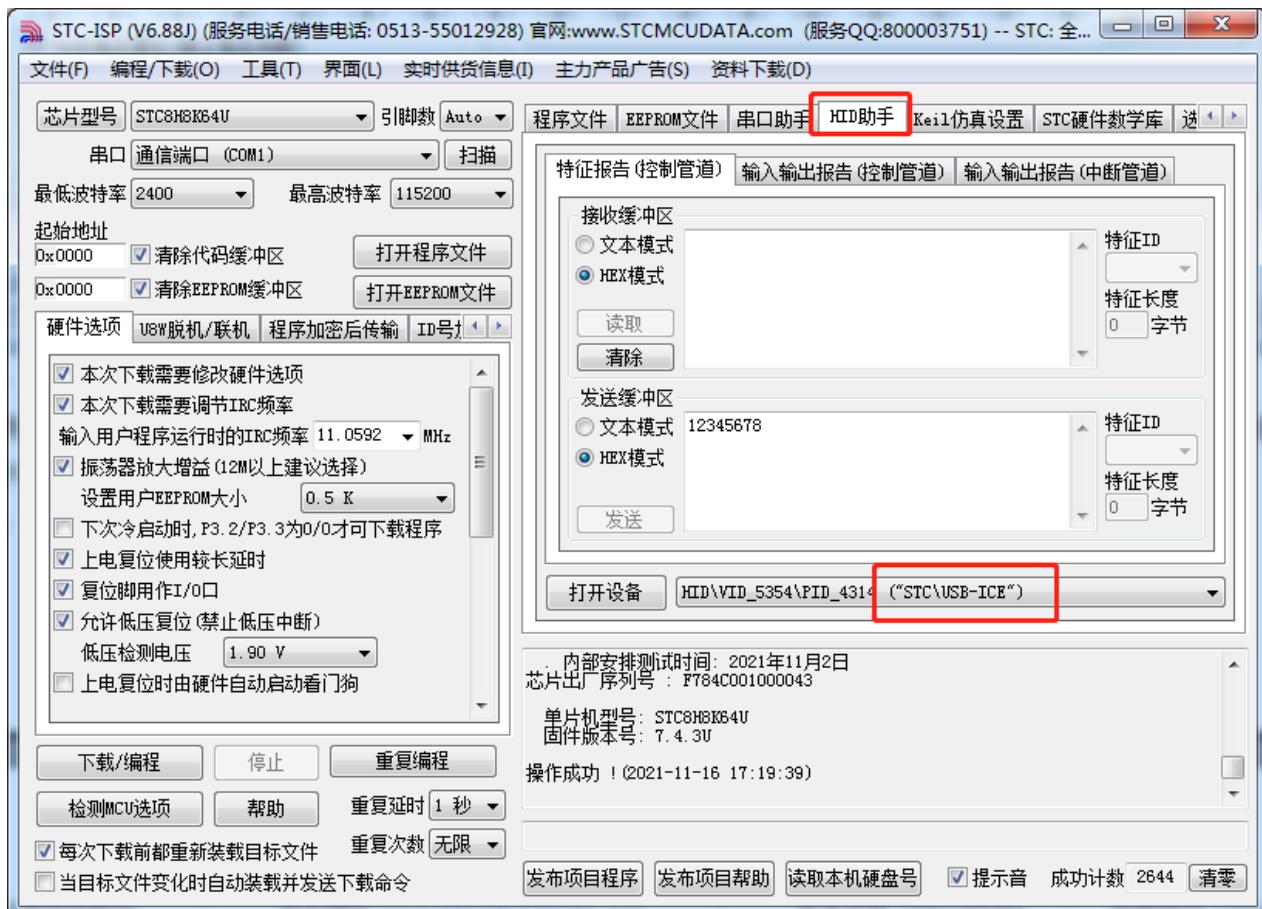
接下来在 STC-ISP 下载软件中, 按照如下图所示的步骤设置仿真芯片



下载完成后如下图所示

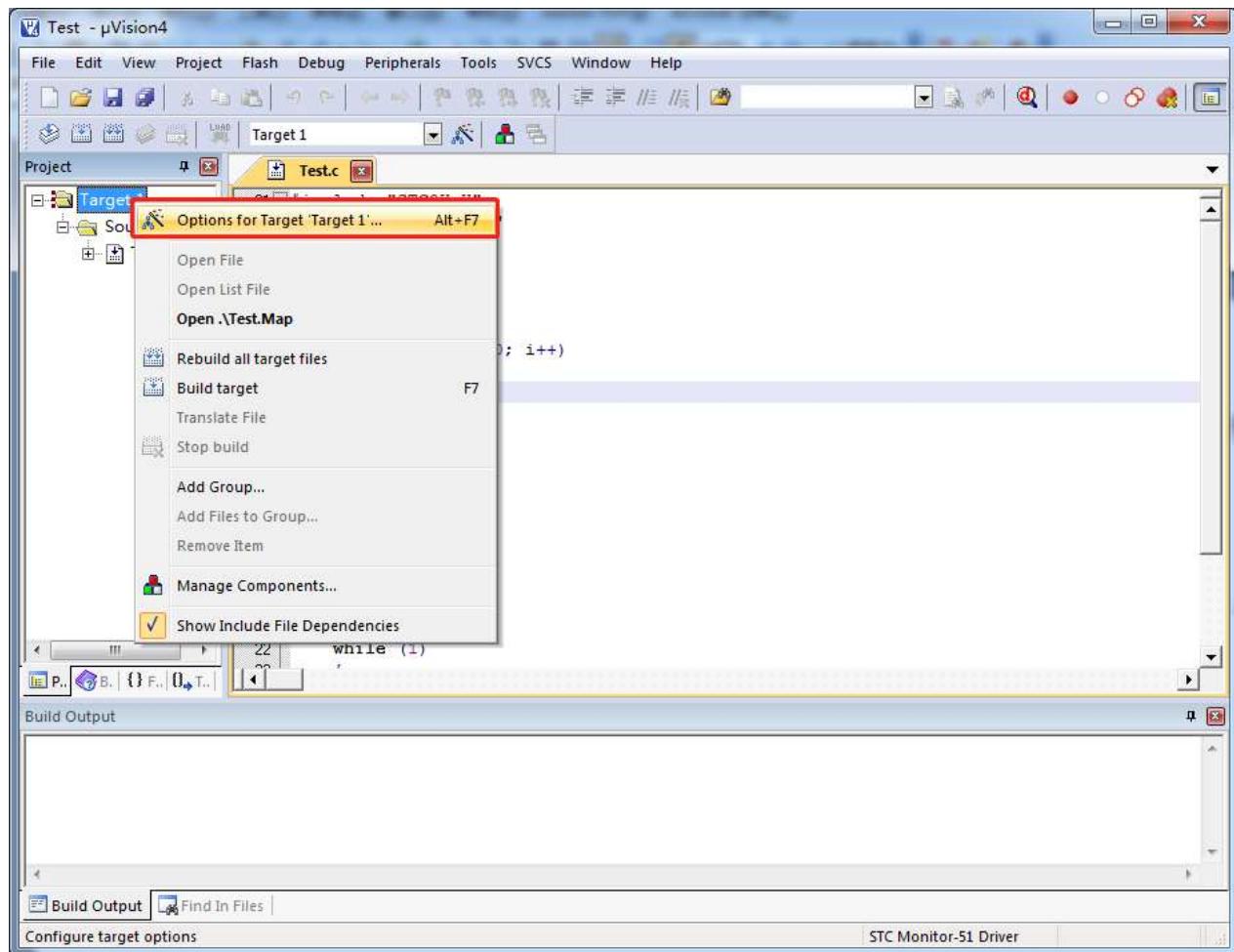


制作完成后，需要将 P3.2 口的接地开关断开，并重新对单片机上电，若在下载软件的“HID 助手”中能检测到“STC\USB-ICE”设备，则表示 USB 仿真芯片制作成功

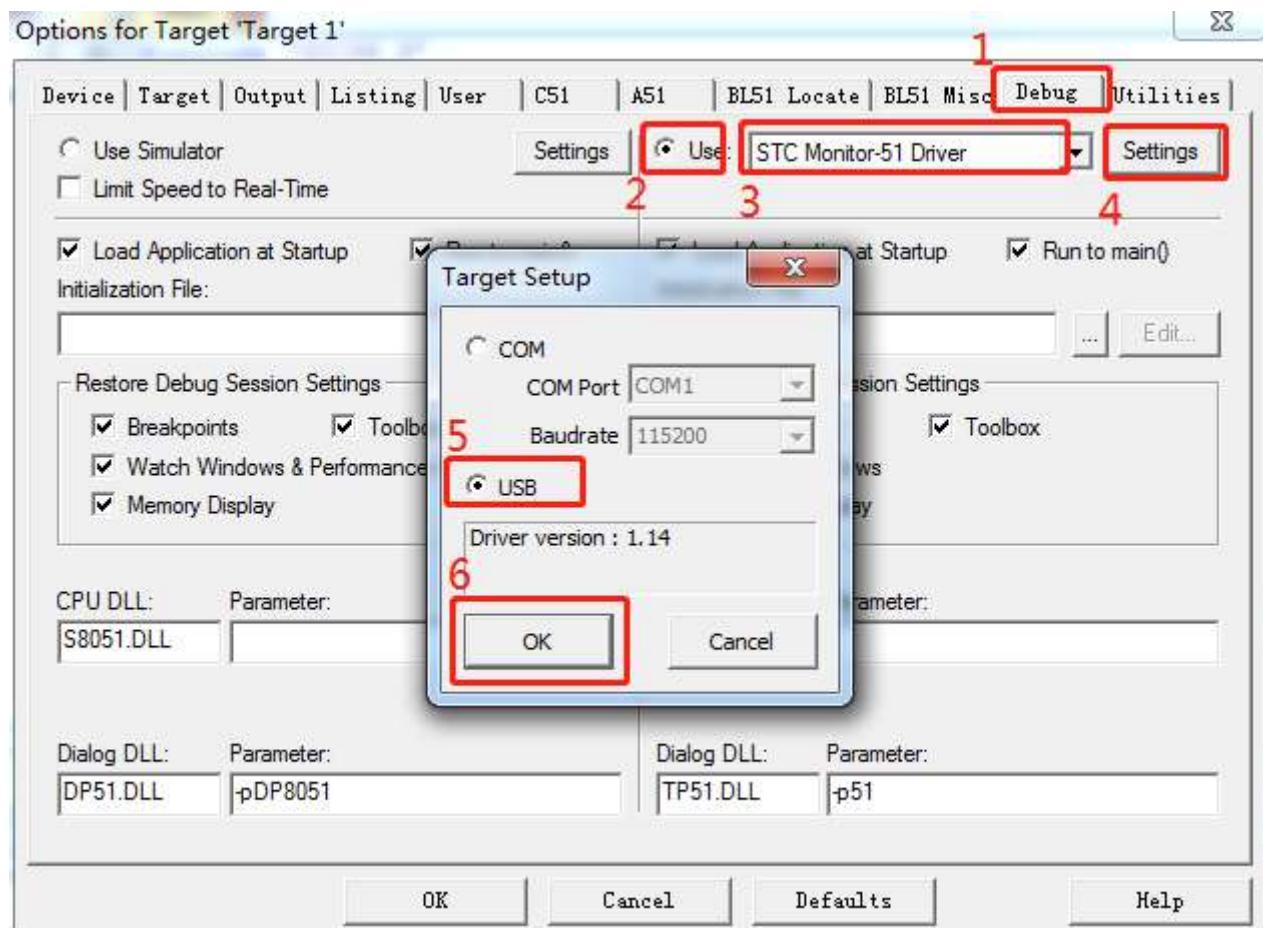


I.5.2 在 Keil 软件中进行 USB 仿真设置

在 Keil 软件中打开项目文件，并在下图所示的右键菜单中点击“Options for ...”

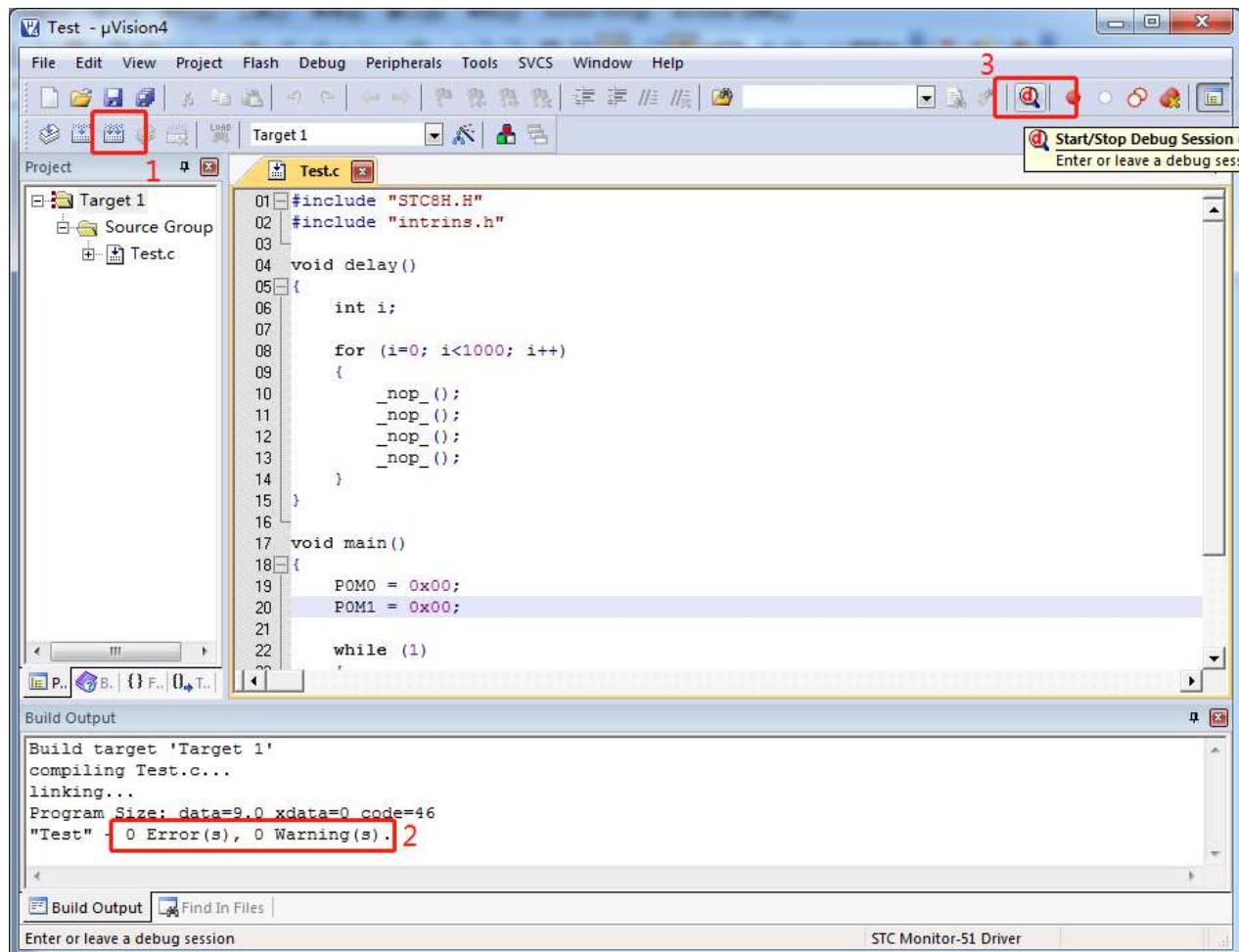


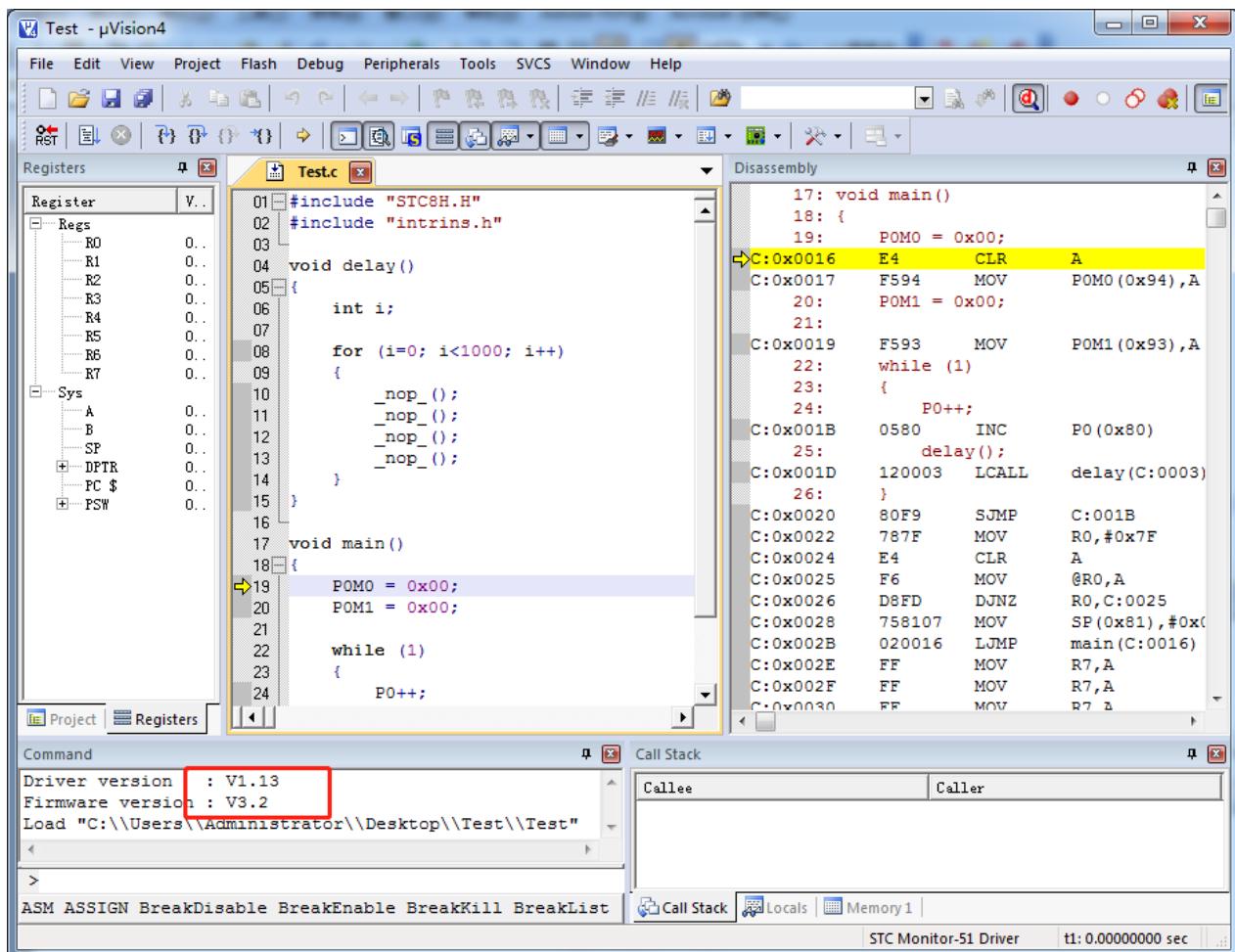
在项目选项中, 按如下图所示的步骤进行 USB 仿真设置



I.5.3 在 Keil 软件中使用 USB 进行仿真

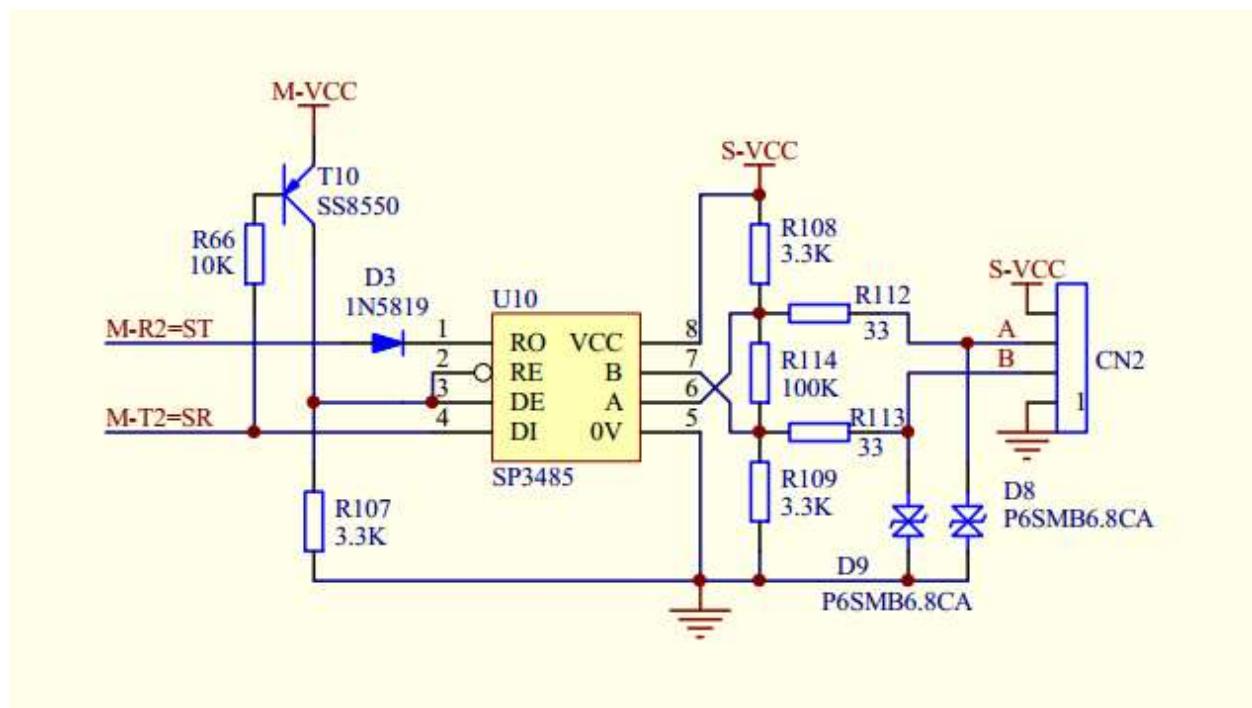
在 Keil 环境下，编辑完成源代码，并编译无误后，即可开始仿真





若芯片制作和连接均无误，则会如上图所示显示仿真驱动版本，并可正确下载用户代码到单片机，接下来便可进行运行、单步、断点等调试功能了。

附录J U8W 下载工具中 RS485 部分线路图



BOM 清单:

标号	型号	封装	备注
U10	SP3485EN	SOP8	RS485 芯片
R66	10K	0603	电阻
R107	3.3K	0603	电阻
R108	3.3K	0603	电阻
R109	3.3K	0603	电阻
R112	33R	0603	电阻
R113	33R	0603	电阻
R114	100K	0603	电阻
T10	SS8550	SOT-23	PNP 三极管
D3	1N5819	0603	肖特基二极管
D8	P6SMB6.8CA	DO-214AA	TVS 二极管
D9	P6SMB6.8CA	DO-214AA	TVS 二极管
CN2		SIP4	通信接口

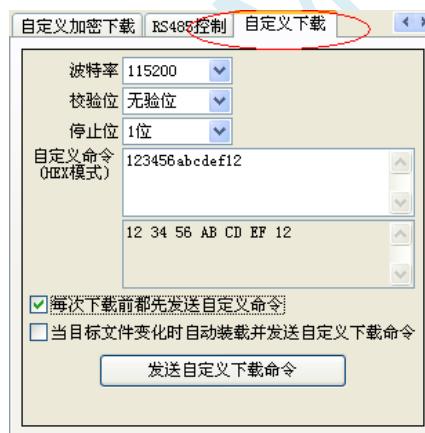
附录K 运行用户程序时收到用户命令后自动启动 ISP 下载(不停电)

“用户自定义下载”与“用户自定义加密下载”是两种完全不同功能。相对用户自定义加密下载的功能而言，用户自定义下载的功能要简单一些。

具体的功能为：电脑或脱机下载板在开始发送真正的 ISP 下载编程握手命令前，先发送用户自定义的一串命令（关于这一串串口命令，用户可以根据自己在应用程序中的串口设置来设置波特率、校验位以及停止位），然后再立即发送 ISP 下载编程握手命令。

“用户自定义下载”这一功能主要是在项目的早期开发阶段，实现不断电（不用给目标芯片重新上电）即可下载用户代码。具体的实现方法是：用户需要在自己的程序中加入一段检测自定义命令的代码，当检测到后，执行一句“MOV IAP_CONTR,#60H”的汇编代码或者“`IAP_CONTR = 0x60;`”的 C 语言代码，MCU 就会自动复位到 ISP 区域执行 ISP 代码。

如下图所示，将自定义命令设置为波特率为 115200、无校验位、一位停止位的命令序列：0x12、0x34、0x56、0xAB、0xCD、0xEF、0x12。当勾选上“每次下载前都先发送自定义命令”的选项后，即可实现自定义下载功能



点击“发送自定义下载命令”或者点击界面左下角的“下载/编程”按钮，应用程序便会发送如下所示的串口数据



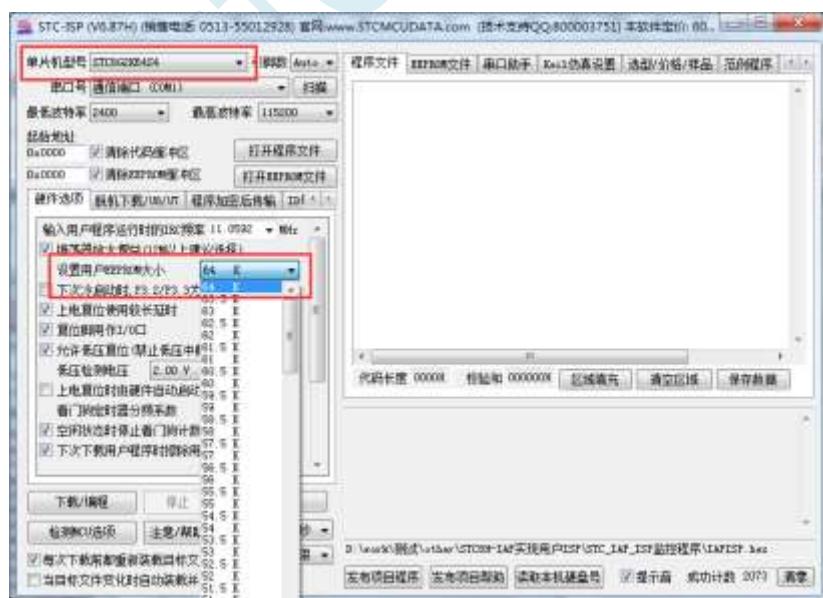
附录L 使用 STC 的 IAP 系列单片机开发自己的 ISP 程序

随着 IAP (In-Application-Programming) 技术在单片机领域的不断发展，给应用系统程序代码升级带来了极大的方便。STC 的串口 ISP (In-System-Programming) 程序就是使用 IAP 功能来对用户的程序进行在线升级的，但是出于对用户代码的安全着想，底层代码和上层应用程序都没有开源，为此 STC 推出了 IAP 系列单片机，即整颗 MCU 的 Flash 空间，用户均可在自己的程序中进行改写，从而使得有用户需要开发自己的 ISP 程序的想法得以实现。

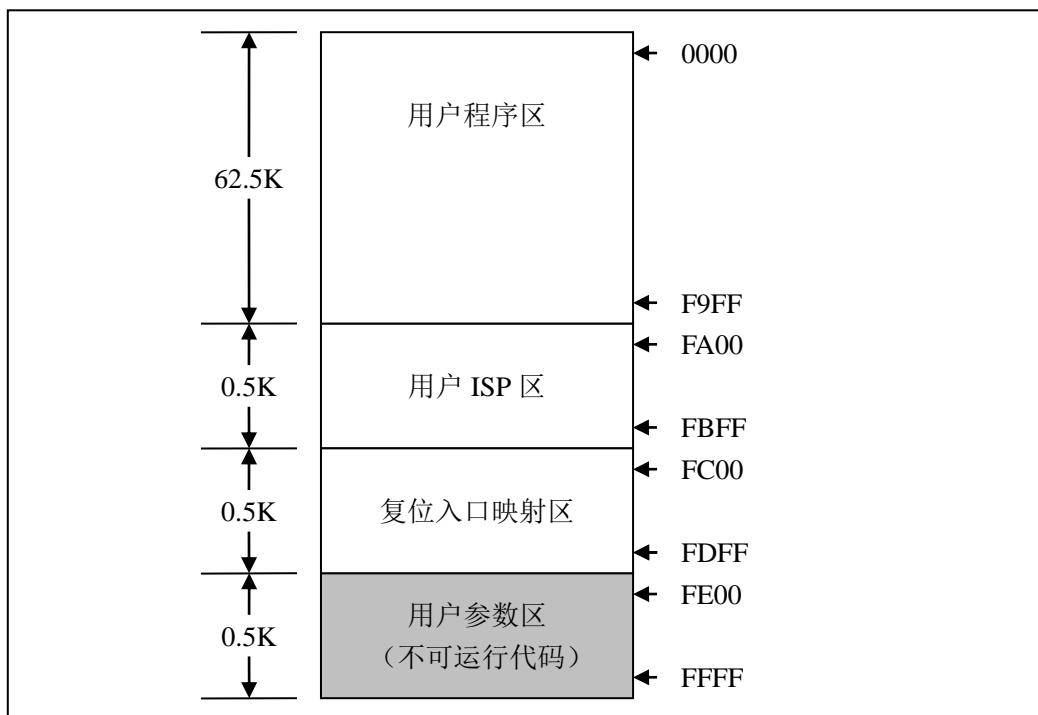
STC8G 系列单片机中的所有可以在 ISP 下载时用户自定义 EEPROM 大小的型号均为 IAP 系列单片机。目前 STC8H 系列有如下型号的单片机为 IAP 系列：STC8G1K12、STC8G1K17、STC8G1K12A、STC8G1K17A、STC8G1K12-8Pin、STC8G1K17-8Pin、STC8G1K12T、STC8G1K17T、STC8G2K64S2、STC8G2K64S4。本文以 STC8G2K64S4 为例，详细说明使用 STC 的 IAP 单片机开发用户自己的 ISP 程序的方法，并给出了基于 Keil 环境的汇编和 C 源码。

第一步：内部 FLASH 规划

由于 STC8G 系列的 IAP 型号单片机的 EEPROM 是在 ISP 下载时用户自己设置的，所以若用户需要实现自己的 ISP，则在下载用户自己的 ISP 程序时，需要按照下图是方式，将全部的 64K 都设置为 EEPROM，让用户程序空间和 EEPROM 空间完全重合，这样才能实现用户对自己程序空间进行修改和更新。

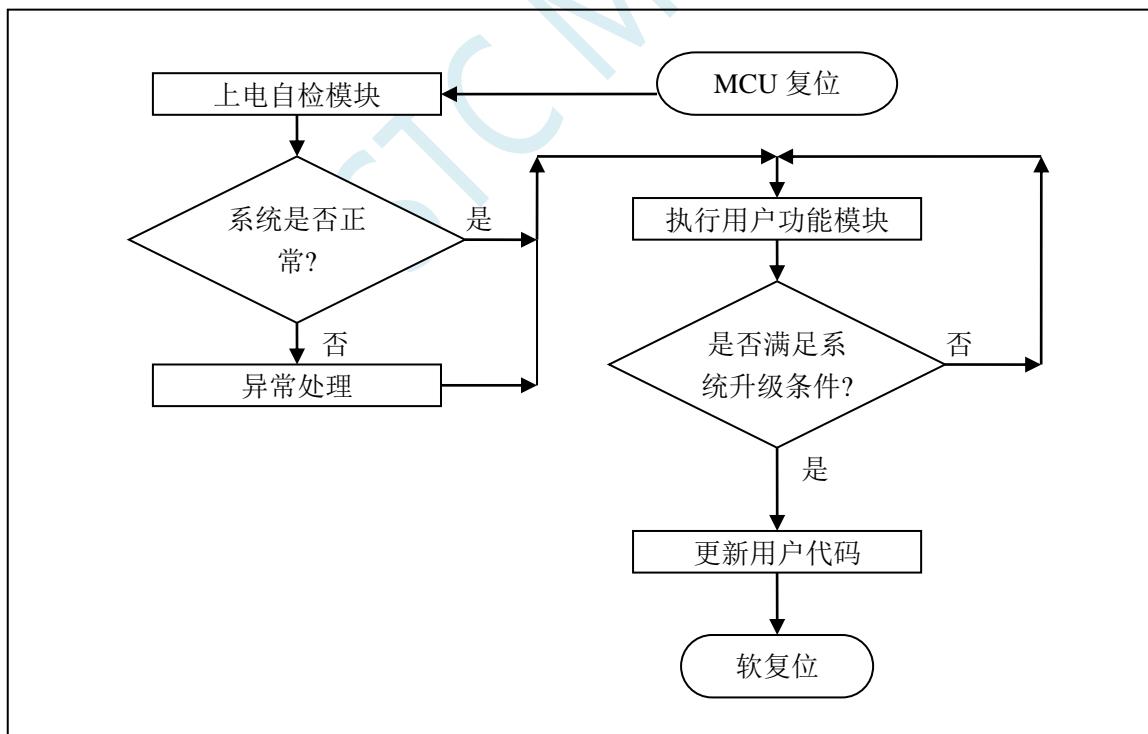


下面假设用户已将整个的 64K 的程序空间已全部设置为 EEPROM，现将整个 64K 程序空间作如下划分：



FLASH空间中，从地址0000H开始的连续62.5K字节的空间为用户程序区。当满足特定的下载条件时，需要用户将PC跳转到用户ISP程序区，此时可对用户程序区进行擦除和改写，以达到更新用户程序的目的。

第二步、程序的基本框架



第三步、下位机固件程序说明

下位机固件程序包括两部分：ISP（ISP代码）和AP（用户代码）

ISP代码（汇编代码）

; 测试工作频率为 11.0592MHz

<i>UARTBAUD</i>	<i>EQU</i>	<i>0FFE8H</i>	; 定义串口波特率 (65536-11059200/4/115200)
<i>AUXR</i>	<i>DATA</i>	<i>08EH</i>	; 附加功能控制寄存器
<i>WDT CONTR</i>	<i>DATA</i>	<i>0C1H</i>	; 看门狗控制寄存器
<i>IAP DATA</i>	<i>DATA</i>	<i>0C2H</i>	; IAP 数据寄存器
<i>IAP ADDRH</i>	<i>DATA</i>	<i>0C3H</i>	; IAP 高地址寄存器
<i>IAP ADDRL</i>	<i>DATA</i>	<i>0C4H</i>	; IAP 低地址寄存器
<i>IAP CMD</i>	<i>DATA</i>	<i>0C5H</i>	; IAP 命令寄存器
<i>IAP TRIG</i>	<i>DATA</i>	<i>0C6H</i>	; IAP 命令触发寄存器
<i>IAP CONTR</i>	<i>DATA</i>	<i>0C7H</i>	; IAP 控制寄存器
<i>IAP TPS</i>	<i>DATA</i>	<i>0F5H</i>	; IAP 等待时间控制寄存器
<i>ISPCODE</i>	<i>EQU</i>	<i>0FA00H</i>	; ISP 模块入口地址(1页), 同时也是外部接口地址
<i>APENTRY</i>	<i>EQU</i>	<i>0FC00H</i>	; 应用程序入口地址数据(1页)
<i>ORG</i> <i>0000H</i>			
<i>LJMP</i>	<i>ISP_ENTRY</i>	; 系统复位入口	
RESET:			
<i>MOV</i>	<i>SCON,#50H</i>	; 设置串口模式(8位数据位, 无校验位)	
<i>MOV</i>	<i>AUXR,#40H</i>	; 定时器1为1T模式	
<i>MOV</i>	<i>TMOD,#00H</i>	; 定时器1工作于模式0(16位重装载)	
<i>MOV</i>	<i>TH1,#HIGH UARTBAUD</i>	; 设置重载值	
<i>MOV</i>	<i>TL1,#LOW UARTBAUD</i>		
<i>SETB</i>	<i>TR1</i>	; 启动定时器1	
NEXT1:	<i>MOV</i>	<i>R0,#16</i>	
NEXT2:	<i>JNB</i>	<i>RI,\$</i>	; 等待串口数据
	<i>CLR</i>	<i>RI</i>	
	<i>MOV</i>	<i>A,SBUF</i>	
	<i>CJNE</i>	<i>A,#7FH,NEXT1</i>	; 判断是否为7F
	<i>DJNZ</i>	<i>R0,NEXT2</i>	
	<i>LJMP</i>	<i>ISP_DOWNLOAD</i>	; 跳转到下载界面
<i>ORG</i> <i>ISPCODE</i>			
ISP_DOWNLOAD:			
<i>CLR</i>	<i>A</i>		
<i>MOV</i>	<i>PSW,A</i>	; ISP模块使用第0组寄存器	
<i>MOV</i>	<i>IE,A</i>	; 关闭所有中断	
<i>CLR</i>	<i>RI</i>	; 清除串口接收标志	
<i>SETB</i>	<i>TI</i>	; 置串口发送标志	
<i>CLR</i>	<i>TR0</i>		
<i>MOV</i>	<i>SP,#5FH</i>	; 设置堆栈指针	
<i>MOV</i>	<i>A,#5AH</i>	; 返回5A 55到PC, 表示ISP擦除模块已准备就绪	
<i>LCALL</i>	<i>ISP_SENDUART</i>		
<i>MOV</i>	<i>A,#055H</i>		
<i>LCALL</i>	<i>ISP_SENDUART</i>		
<i>LCALL</i>	<i>ISP_RECVACK</i>	; 接应收应答数据	
<i>MOV</i>	<i>IAP_ADDRL,#0</i>	; 首先在第2页起始地址写 "LJMP ISP_ENTRY" 指令	
<i>MOV</i>	<i>IAP_ADDRH,#02H</i>		
<i>LCALL</i>	<i>ISP_ERASEIAP</i>		
<i>MOV</i>	<i>A,#02H</i>		
<i>LCALL</i>	<i>ISP_PROGRAMIAP</i>	; 编程用户代码复位向量代码	

<i>MOV</i>	<i>A,#HIGH</i>	<i>ISP_ENTRY</i>
<i>L CALL</i>	<i>ISP_PROGRAMIAP</i>	;编程用户代码复位向量代码
<i>MOV</i>	<i>A,#LOW ISP_ENTRY</i>	
<i>L CALL</i>	<i>ISP_PROGRAMIAP</i>	;编程用户代码复位向量代码
<i>MOV</i>	<i>IAP_ADDRL,#0</i>	;用户代码地址从 0 开始
<i>MOV</i>	<i>IAP_ADDRH,#0</i>	
<i>L CALL</i>	<i>ISP_ERASEIAP</i>	
<i>MOV</i>	<i>A,#02H</i>	
<i>L CALL</i>	<i>ISP_PROGRAMIAP</i>	;编程用户代码复位向量代码
<i>MOV</i>	<i>A,#HIGH</i>	<i>ISP_ENTRY</i>
<i>L CALL</i>	<i>ISP_PROGRAMIAP</i>	;编程用户代码复位向量代码
<i>MOV</i>	<i>A,#LOW ISP_ENTRY</i>	
<i>L CALL</i>	<i>ISP_PROGRAMIAP</i>	;编程用户代码复位向量代码
<i>MOV</i>	<i>IAP_ADDRL,#0</i>	;新代码缓冲区地址
<i>MOV</i>	<i>IAP_ADDRH,#02H</i>	
<i>MOV</i>	<i>R7,#124</i>	;擦除 62.5K 字节
 <i>ISP_ERASEAP:</i>		
<i>L CALL</i>	<i>ISP_ERASEIAP</i>	
<i>INC</i>	<i>IAP_ADDRH</i>	;目标地址+512
<i>INC</i>	<i>IAP_ADDRH</i>	
<i>DJNZ</i>	<i>R7,ISP_ERASEAP</i>	;判断是否擦除完成
<i>MOV</i>	<i>IAP_ADDRL,#LOW APENTRY</i>	
<i>MOV</i>	<i>IAP_ADDRH,#HIGH APENTRY</i>	
<i>L CALL</i>	<i>ISP_ERASEIAP</i>	
<i>MOV</i>	<i>A,#5AH</i>	;返回 5A A5 到 PC, 表示 ISP 编程模块已准备就绪
<i>L CALL</i>	<i>ISP_SENDUART</i>	
<i>MOV</i>	<i>A,#0A5H</i>	
<i>L CALL</i>	<i>ISP_SENDUART</i>	
<i>L CALL</i>	<i>ISP_RECVACK</i>	;接收应答数据
<i>L CALL</i>	<i>ISP_RECVUART</i>	;接收长度高字节
<i>MOV</i>	<i>R0,A</i>	
<i>L CALL</i>	<i>ISP_RECVUART</i>	;接收长度低字节
<i>MOV</i>	<i>RI,A</i>	
<i>CLR</i>	<i>C</i>	;将总长度-3
<i>MOV</i>	<i>A,#03H</i>	
<i>SUBB</i>	<i>A,RI</i>	
<i>MOV</i>	<i>DPL,A</i>	
<i>CLR</i>	<i>A</i>	
<i>SUBB</i>	<i>A,R0</i>	
<i>MOV</i>	<i>DPH,A</i>	;总长度补码存入 DPTR
<i>L CALL</i>	<i>ISP_RECVUART</i>	;映射用户代码复位入口代码到映射区
<i>L CALL</i>	<i>ISP_PROGRAMIAP</i>	;0000
<i>L CALL</i>	<i>ISP_RECVUART</i>	
<i>L CALL</i>	<i>ISP_PROGRAMIAP</i>	;0001
<i>L CALL</i>	<i>ISP_RECVUART</i>	
<i>L CALL</i>	<i>ISP_PROGRAMIAP</i>	;0002
<i>MOV</i>	<i>IAP_ADDRL,#03H</i>	;用户代码起始地址
<i>MOV</i>	<i>IAP_ADDRH,#0H</i>	
 <i>ISP_PROGRAMNEXT:</i>		
<i>L CALL</i>	<i>ISP_RECVUART</i>	;接收代码数据
<i>L CALL</i>	<i>ISP_PROGRAMIAP</i>	;编程到用户代码区
<i>INC</i>	<i>DPTR</i>	

MOV *A,DPL*
ORL *A,DPH*
JNZ *ISP_PROGRAMNEXT* ;长度检测

ISP_SOFTRESET:

MOV *IAP_CONTR,#20H* ;软件复位系统
SJMP *\$*

ISP_ENTRY:

MOV *WDT CONTR,#17H* ;清看门狗
MOV *IAP CONTR,#80H* ;使能 IAP 功能
MOV *IAP TPS,#11* ;设置 IAP 等待时间参数
MOV *IAP ADDR,#LOW ISP_DOWNLOAD*
MOV *IAP ADDRH,#HIGH ISP_DOWNLOAD*
MOV *IAP DATA,#00H* ;测试数据 1
MOV *IAP CMD,#1* ;读命令
MOV *IAP TRIG,#5AH* ;触发 ISP 命令
MOV *IAP TRIG,#0A5H*
MOV *A,IAP DATA*
CJNE *A,#0E4H,ISP_ENTRY* ;若无法读出数据则需要等待电压稳定
INC *IAP ADDR*
MOV *IAP DATA,#45H* ;测试地址 FC01H
MOV *IAP CMD,#1* ;读命令
MOV *IAP TRIG,#5AH* ;触发 ISP 命令
MOV *IAP TRIG,#0A5H*
MOV *A,IAP DATA*
CJNE *A,#0F5H,ISP_ENTRY* ;若无法读出数据则需要等待电压稳定

MOV *SCON,#50H*
MOV *AUXR,#40H*
MOV *TMOD,#00H*
MOV *TH1,#HIGH UARTBAUD*
MOV *Tl1,#LOW UARTBAUD*
SETB *TR1*
SETB *TR0* ;启动定时器 1

LCALL *ISP_RECVUART* ;检测是否有串口数据
JC *GOTOAP*
MOV *R0,#16*

ISP_CHECKNEXT:

LCALL *ISP_RECVUART* ;接收同步数据
JC *GOTOAP*
CJNE *A,#7FH,GOTOAP* ;判断是否为 7F
DJNZ *R0,ISP_CHECKNEXT*
MOV *A,#5AH* ;返回 5A 69 到 PC, 表示 ISP 模块已准备就绪
LCALL *ISP_SENDUART*
MOV *A,#69H*
LCALL *ISP_SENDUART*
LCALL *ISP_RECVACK* ;接收应答数据
LJMP *ISP_DOWNLOAD* ;跳转到下载界面

GOTOAP:

CLR *A* ;将 SFR 恢复为复位值
MOV *TCON,A*
MOV *TMOD,A*
MOV *TL0,A*
MOV *TH0,A*
MOV *TL1,A*
MOV *TH1,A*

<i>MOV</i>	<i>SCON,A</i>
<i>MOV</i>	<i>AUXR,A</i>
<i>LJMP</i>	<i>APENTRY</i>

;正常运行用户程序

ISP_RECVACK:

<i>LCALL</i>	<i>ISP_RECVUART</i>
<i>JC</i>	<i>GOTOAP</i>
<i>XRL</i>	<i>A,#7FH</i>
<i>JZ</i>	<i>ISP_RECVACK</i>
<i>CJNE</i>	<i>A,#25H,GOTOAP</i>
<i>LCALL</i>	<i>ISP_RECVUART</i>
<i>JC</i>	<i>GOTOAP</i>
<i>CJNE</i>	<i>A,#69H,GOTOAP</i>
<i>RET</i>	

;跳过同步数据
;应答数据1 检测
;应答数据2 检测

ISP_RECVUART:

<i>CLR</i>	<i>A</i>
<i>MOV</i>	<i>TL0,A</i>
<i>MOV</i>	<i>TH0,A</i>
<i>CLR</i>	<i>TF0</i>
<i>MOV</i>	<i>WDT_CONTR,#17H</i>

;初始化超时定时器
;清看门狗

ISP_RECVWAIT:

<i>JBC</i>	<i>TF0,ISP_RECVTIMEOUT</i>
<i>JNB</i>	<i>RI,ISP_RECVWAIT</i>
<i>MOV</i>	<i>A,SBUF</i>
<i>CLR</i>	<i>RI</i>
<i>CLR</i>	<i>C</i>
<i>RET</i>	

;超时检测
;等待接收完成
;读取串口数据
;清除标志
;正确接收串口数据

ISP_RECVTIMEOUT:

<i>SETB</i>	<i>C</i>
<i>RET</i>	

;超时退出

ISP_SENDUART:

<i>MOV</i>	<i>WDT_CONTR,#17H</i>
<i>JNB</i>	<i>TI,ISP_SENDUART</i>
<i>CLR</i>	<i>TI</i>
<i>MOV</i>	<i>SBUF,A</i>
<i>RET</i>	

;清看门狗
;等待前一个数据发送完成
;清除标志
;发送当前数据

ISP_ERASEIAP:

<i>MOV</i>	<i>WDT_CONTR,#17H</i>
<i>MOV</i>	<i>IAP_CMD,#3</i>
<i>MOV</i>	<i>IAP_TRIG,#5AH</i>
<i>MOV</i>	<i>IAP_TRIG,#0A5H</i>
<i>NOP</i>	
<i>NOP</i>	
<i>NOP</i>	
<i>NOP</i>	
<i>RET</i>	

ISP_PROGRAMIAP:

<i>MOV</i>	<i>WDT_CONTR,#17H</i>
<i>MOV</i>	<i>IAP_CMD,#2</i>
<i>MOV</i>	<i>IAP_DATA,A</i>
<i>MOV</i>	<i>IAP_TRIG,#5AH</i>
<i>MOV</i>	<i>IAP_TRIG,#0A5H</i>
<i>NOP</i>	
<i>NOP</i>	
<i>NOP</i>	

;清看门狗
;编程命令
;将当前数据送 IAP 数据寄存器
;触发 ISP 命令

```

NOP
MOV      A,IAP_ADDR           ;IAP 地址+1
ADD      A,#01H
MOV      IAP_ADDRH,A
MOV      A,IAP_ADDRH
ADD      A,#00H
MOV      IAP_ADDRH,A
RET

ORG      APENTRY
LJMP    RESET

END

```

ISP 代码包括如下外部接口模块:

ISP_DOWNLOAD: 程序下载入口地址, 绝对地址 **FA00H**

ISP_ENTRY: 上电系统自检程序 (系统自动调用)

对于用户程序而言, 用户只需要在满足下载条件时, 将 PC 值跳转到 ISPPROGRAM (即 FA00H 的绝对地址), 即可实现代码更新。

用户代码 (C 语言代码)

```

//测试工作频率为 11.0592MHz

#include "reg51.h"

#define FOSC          11059200L           //系统时钟频率
#define BAUD         (65536 - FOSC/4/115200) //定义串口波特率
#define ISPPROGRAM    0xfa00             //ISP 下载程序入口地址

sfr AUXR        = 0x8e;               //波特率发生器控制寄存器
sfr P1M0        = 0x92;
sfr P1M1        = 0x91;

void (*IspProgram)() = ISPPROGRAM;   //定义指针函数
char cnt7f;                          //Isp_Check 内部使用的变量

void uart() interrupt 4              //串口中断服务程序
{
    if (TI) TI = 0;                  //发送完成中断
    if (RI)                         //接收完成中断
    {
        if (SBUF == 0x7f)
        {
            cnt7f++;
            if (cnt7f >= 16)
            {
                IspProgram();          //调用下载模块(****重要语句****)
            }
        }
        else
        {
            cnt7f = 0;
        }
    }
}

```

```

        }
        RI = 0;
    }

void main()
{
    SCON = 0x50;                                //定义串口模式为8bit 可变,无校验位
    AUXR = 0x40;
    TH1 = BAUD >> 8;
    TL1 = BAUD;
    TR1 = 1;
    ES = 1;                                     //使能串口中断
    EA = 1;                                     //打开全局中断开关

    P1M0 = 0;
    P1M1 = 0;

    while (1)
    {
        PI++;
    }
}

```

用户代码 (汇编代码)

;测试工作频率为 11.0592MHz

UARTBAUD EQU	0FFE8H	; 定义串口波特率 (65536-11059200/4/115200)
ISPPROGRAM EQU	0FA00H	; ISP 下载程序入口地址
AUXR DATA	08EH	; 附件功能控制寄存器
CNT7F DATA	60H	; 接收7F 的计数器
ORG	0000H	
LJMP	START	; 系统复位入口
ORG	0023H	
LJMP	UART_ISR	; 串口中断入口
UART_ISR:		
PUSH	ACC	
PUSH	PSW	
JNB	TI,CHECKRI	; 检测发送中断
CLR	TI	; 清除标志
CHECKRI:		
JNB	RI,UARTISR_EXIT	; 检测接收中断
CLR	RI	; 清除标志
MOV	A,SBUF	
CJNE	A,#7FH,ISNOT7F	
INC	CNT7F	
MOV	A,CNT7F	
CJNE	A,#16,UARTISR_EXIT	
LJMP	ISPPROGRAM	; 调用下载模块(****重要语句****)
ISNOT7F:		
MOV	CNT7F,#0	
UARTISR_EXIT:		
POP	PSW	

*POP
RETI*

ACC

START:

<i>MOV</i>	<i>R0,#7FH</i>	;清RAM
<i>CLR</i>	<i>A</i>	
<i>MOV</i>	<i>@R0,A</i>	
<i>DJNZ</i>	<i>R0,\$-1</i>	
<i>MOV</i>	<i>SP,#7FH</i>	;初始化SP
<i>MOV</i>	<i>SCON,#50H</i>	;设置串口模式(8位可变,无校验位)
<i>MOV</i>	<i>AUXR,#15H</i>	;BRT 工作于1T 模式启动BRT
<i>MOV</i>	<i>TMOD,#00H</i>	;定时器1 工作于模式0(16位重装载)
<i>MOV</i>	<i>TH1,#HIGH UARTBAUD</i>	;设置重载值
<i>MOV</i>	<i>TL1,#LOW UARTBAUD</i>	
<i>SETB</i>	<i>TR1</i>	;启动定时器1
<i>SETB</i>	<i>ES</i>	;使能串口中断
<i>SETB</i>	<i>EA</i>	;开中断总开关

MAIN:

<i>INC</i>	<i>P0</i>
<i>SJMP</i>	<i>MAIN</i>

END

用户代码可以使用 C 或者汇编语言编写，但对于汇编代码需要注意一点：位于 0000H 的复位入口地址处的指令必须是一个长跳转语句（类似 LJMP START）。在用户代码中，需要设置好串口，并在满足下载条件时，将 PC 值跳转到 ISPPROGRAM（即 FA00H 的绝对地址），以实现代码更新。对于汇编代码，我们可以使用“LJMP OFA00H”指令进行调用，如下图

UARTBAUD	EQU	OFFE8H	: 定义串口波特率 (65536-11059200/4/115200)
ISPPROGRAM	EQU	OFA00H	: ISP 下载程序入口地址
AUXR	DATA	08EH	: 附件功能控制寄存器

```

18      CLR    TI           ; 清除标志
19  CHECKRI:
20      JNB    RI, UARTISR_EXIT   ; 检测接收中断
21      CLR    RI           ; 清除标志
22      MOV    A, SBUF
23      CJNE   A, #7FH, ISNOT7F
24      INC    CNT7F
25      MOV    A, CNT7F
26      CJNE   A, #16, UARTISR_EXIT
27      LJMP   ISPPROGRAM     ; 调用下载模块 (****重要语句****)
28  ISNOT7F:
29      MOV    CNT7F, #0
30  UARTISR_EXIT:
31      POP    PSW
32      POP    ACC
33      RETI
34
35  START:

```

在 C 代码中，必须定义一个函数指针变量，并将此变量赋值为 0xFA00，然后再调用，如下图

```

#include "reg51.h"

#define FOSC          11059200L           // 系统时钟频率
#define BAUD         (65536 - FOSC/4/115200) // 定义串口波特率
#define ISPPROGRAM    0xfa00             // ISP 下载程序入口地址

sfr AUXR      = 0x8e;                  // 波特率发生器控制寄存器
sfr P1M0      = 0x92;
sfr P1M1      = 0x91;

void (*IspProgram) () = ISPPROGRAM;    // 定义指针函数
char cnt7f;                            // Isp_Check 内部使用的变量

void uart() interrupt 4                // 串口中断服务程序
{
    if (TI) TI = 0;                   // 发送完成中断
    if (RI) RI = 0;                   // 接收完成中断
    {
        if (SBUF == 0x7f)
        {
            cnt7f++;
            if (cnt7f >= 16)
            {
                IspProgram();          // 调用下载模块 (****重要语句****)
            }
        }
        else
        {
            cnt7f = 0;
        }
        RI = 0;                      // 清接收完成标志
    }
}

```

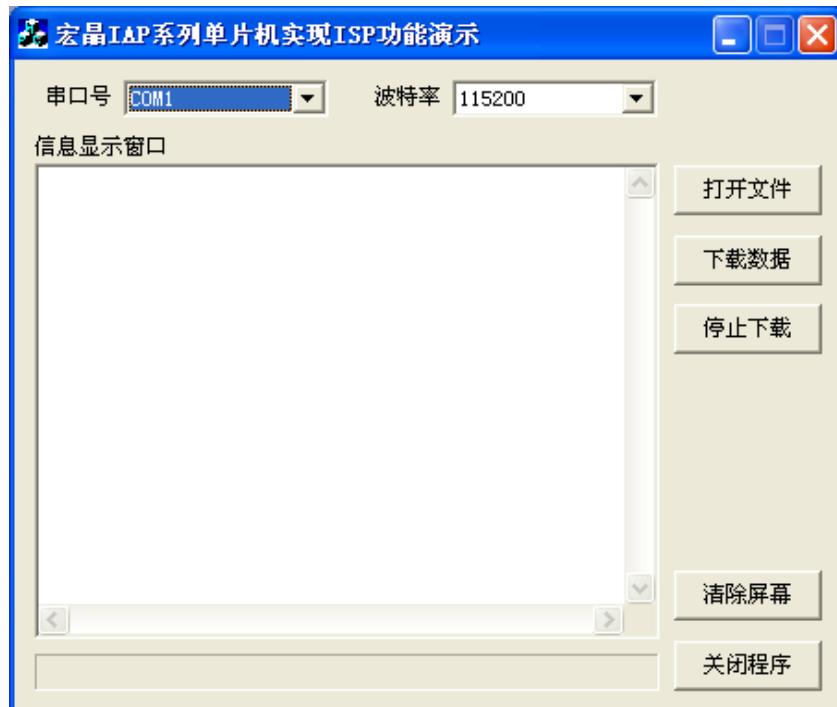
第四步、上位机应用程序说明

上位机的程序是基于 MFC 的对话框项目，对于串口的访问是直接调用 Windows 的 API 函数，而没有使用串口控件，从而省去的控件的注册以及系统版本不兼容的诸多问题。界面较简单，只是为这一功能的实现提供了一个框架，其他的功能及要求均还可以往上面添加。

上位机程序的核心模块是基于类 CISPDlg 的一个友元函数 “UINT Download(LPVOID pParam); ”，此函数负责与下位机通讯，发送各种通讯命令来完成对用户程序的更新。用户可以根据各自不同的需求增加命令。

第五步、上位机应用程序的使用方法

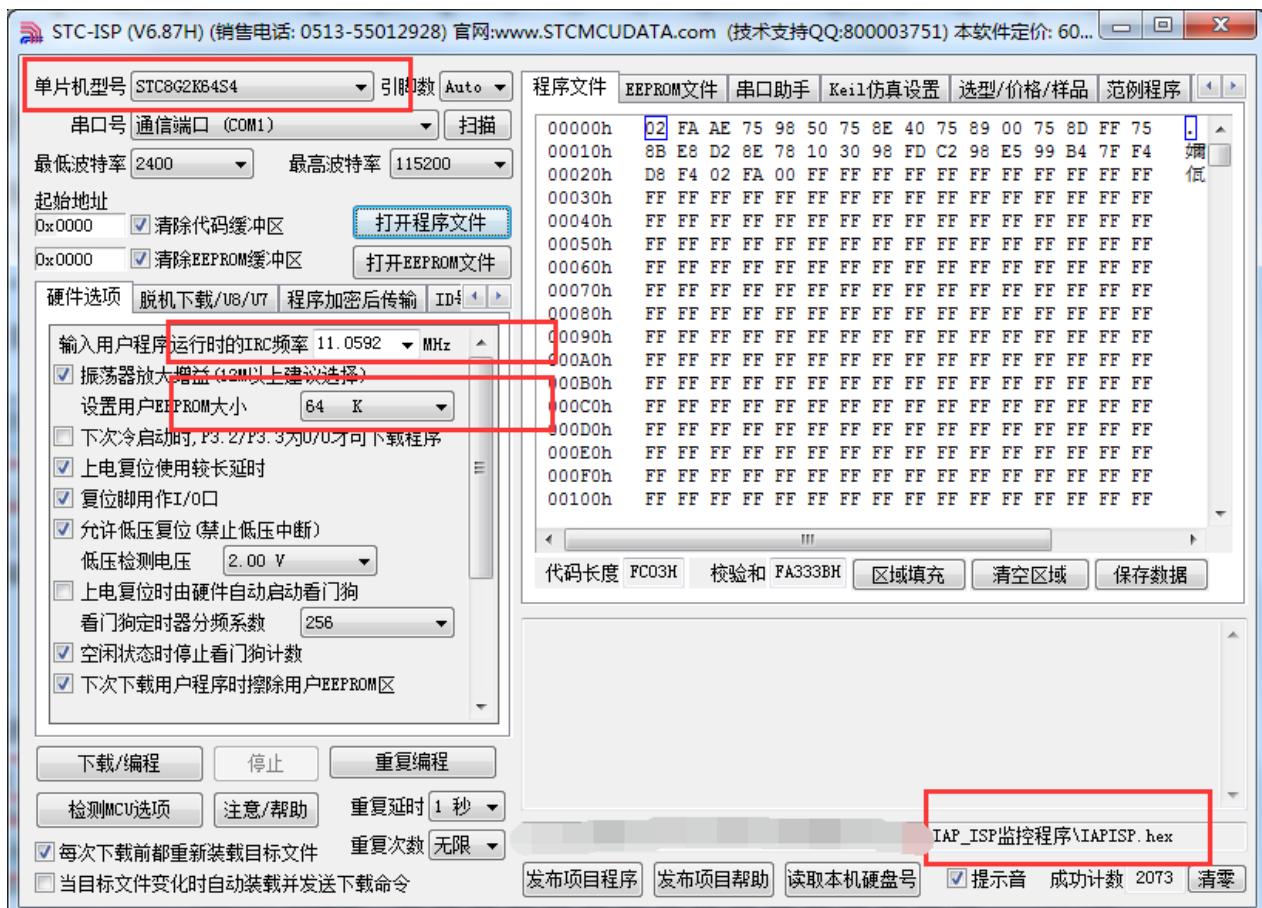
- 打开上位机界面，如下图



- 选择串口号，设置与下位机相同的串口波特率
- 打开要下载的源数据文件，Bin 或者 Intel hex 格式均可以
- 点击“下载数据”按钮即可开始下载数据

第六步、下位机固件程序的使用方法

下位机的目标文件有两个“IAPISP.hex”和“AP.hex”，对于一块新的单片机，第一次必须使用 ISP 下载工具将“IAPISP.hex”写入到芯片内，如下图所示。之后再更新便不再需要写“IAPISP.hex”这个文件了，附件中的“AP.hex”只是一个用户程序的模板，当满足下载条件时，用户只需要将 PC 值跳转到 FA00H 的地址，即可实现代码更新。



附录M 用户程序复位到系统区进行 ISP 下载的方法（不停电）

当项目处于开发阶段时，需要反复的下载用户代码到目标芯片中进行代码验证，而 STC 的单片机进行正常的 ISP 下载都需要对目标芯片进行重新上电，从而会使得项目在开发阶段比较繁琐。为此 STC 单片机增加了一个特殊功能寄存器 IAP_CONTR，当用户向此寄存器写入 0x60，即可实现软件复位到系统区，进而实现不停电就可进行 ISP 下载。

但是用户如何判断是否正在进行 ISP 下载？何时向寄存器 IAP_CONTR 写 0x60 触发软复位？就这两个问题，下面分别介绍四种判断方法：

使用 P3.0 口检测串口起始信号

STC 单片机的串口 ISP 固定使用 P3.0 和 P3.1 两个端口，当 ISP 下载软件开始下载时，会发送握手命令到单片机的 P3.0 口。若用户的 P3.0 和 P3.1 只是专门用于 ISP 下载，则可使用 P3.0 口检测串口的起始信号来判断 ISP 下载。

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr     IAP CONTR = 0xc7;
sfr     P3M0      = 0xb2;
sfr     P3M1      = 0xb1;

sbit    P30       = P3^0;

void main()
{
    P3M0 = 0x00;
    P3M1 = 0x00;
    P30 = 1;

    while (1)
    {
        if (!P30) IAP CONTR = 0x60;           //P3.0 的低电平即为串口起始信号
                                                //软件复位到系统区

        ...
    }
}
```

使用 P3.0/INT4 口的下降沿中断，检测串口起始信号

方法 B 与方法 A 类似，不同在于方法 A 使用的是查询方式，方法 B 使用中断方式。因为 STC 单片机的 P3.0 口为 INT4 的中断口。

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr IAP_CONTR = 0xc7;
sfr INTCLKO = 0x8f;
sfr P3M0 = 0xb2;
sfr P3MI = 0xb1;

void Int4Isr() interrupt 16 //INT4 中断服务程序
{
    IAP_CONTR = 0x60; //串口起始信号触发 INT4 中断
    //软件复位到系统区
}

void main()
{
    P3M0 = 0x00;
    P3MI = 0x00;

    INTCLKO |= 0x40; //使能 INT4 中断
    EA = 1;

    while (1)
    {
        ...
    }
}
```

使用 P3.0/RxD 口的串口接收，检测 ISP 下载软件发送的 7F

方法 A 与方法 B 都非常简单，但容易受干扰，如果 P3.0 口有任何一个干扰信号，都会触发软件复位，所以方法 C 是对串口数据进行校验。

STC 的 ISP 下载软件进行 ISP 下载时，首先都会使用最低波特率（一般是 2400）+ 偶校验 9+1 位停止位连续发送握手命令 7F，因此用户可以在程序中，将串口设置为 9 位数据位+2400 波特率，然后持续检测 7F，比如连续检测到 8 个 7F 表示可确定需要进行 ISP 下载，此时再触发软件复位。

C 语言代码

```
//测试工作频率为 11.0592MHz
```

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

#define FOSC      11059200UL
#define BR2400    (65536 - FOSC / 4 / 2400)

sfr IAP_CONTR = 0xc7;
sfr AUXR      = 0x8e;
sfr P3M0      = 0xb2;
sfr P3M1      = 0xb1;

char cnt7f;

void UartIsr() interrupt 4           //串口中断服务程序
{
    if (TI)
    {
        TI = 0;
    }

    if (RI)
    {
        RI = 0;
        if ((SBUF == 0x7f) && (RB8 == 1))           //ISP 下载软件发送的握手命令 7F
            //7F 的偶校验位为1
        {
            if (++cnt7f == 8)                      //当连续检测到8个7F后
                IAP_CONTR = 0x60;                  //复位到系统区
        }
        else
        {
            cnt7f = 0;
        }
    }
}

void main()
{
    P3M0 = 0x00;
    P3M1 = 0x00;

    SCON = 0xd0;                           //设置串口为9位数据位
    TMOD = 0x00;
    AUXR = 0x40;
    TH1 = BR2400 >> 8;                  //设置串口波特率为2400
    TL1 = BR2400;
    TR1 = 1;
    ES = 1;
    EA = 1;

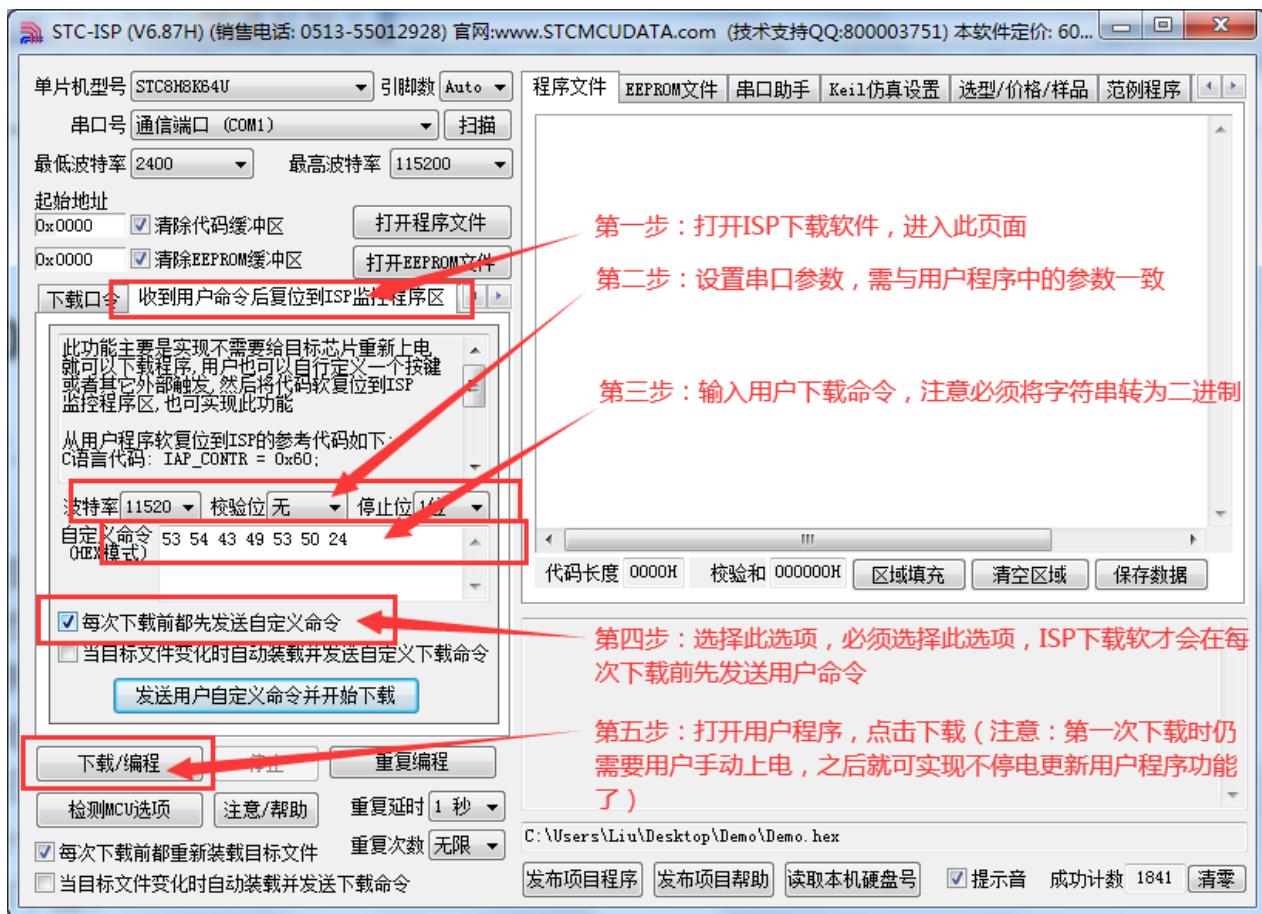
    cnt7f = 0;

    while (1)
    {
        ...                                //用户代码
    }
}
```

使用 P3.0/RxD 串口接收，检测 ISP 下载软件发送的用户下载命令

如果用户代码中需要使用串口进行通信，则上面的 3 中方法可能都不太适用，此时可以使用 STC 的 ISP 下载软件提供的接口，定制一组专用的用户下载命令（可指定波特率、校验位和停止位），若使能此功能，ISP 下载软件在进行 ISP 下载前，会使用用户指定的波特率、校验位和停止位发送用户下载命令，然后再发送握手命令。用户只需要在自己的代码中监控串口命令序列，当检测到有正确的用户下载命令时，软件复位到系统区即可实现不停电进行 ISP 功能。

下面假设用户下载命令为字符串“STCISP\$”，串口设置为波特率 115200，无校验位和 1 位停止位。ISP 下载软件中的设置如下图：



用户示例代码如下：

C 语言代码

```
//测试工作频率为 11.0592MHz
```

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

#define FOSC      11059200UL
#define BR115200  (65536 - FOSC / 115200)

sfr IAP_CONTR = 0xc7;
sfr AUXR     = 0x8e;
sfr P3M0     = 0xb2;
```

```
sfr      P3M1      =      0xb1;

char stage;

void UartIsr() interrupt 4          //串口中断服务程序
{
    char dat;

    if (TI)
    {
        TI = 0;
    }

    if (RI)
    {
        RI = 0;

        dat = SBUF;
        switch (stage)
        {
        case 0:
        default:
            L_Check1st:
            if (dat == 'S') stage = 1;
            else stage = 0;
            break;
        case 1:
            if (dat == 'T') stage = 2;
            else goto L_Check1st;
            break;
        case 2:
            if (dat == 'C') stage = 3;
            else goto L_Check1st;
            break;
        case 3:
            if (dat == 'T') stage = 4;
            else goto L_Check1st;
            break;
        case 4:
            if (dat == 'S') stage = 5;
            else goto L_Check1st;
            break;
        case 5:
            if (dat == 'P') stage = 6;
            else goto L_Check1st;
            break;
        case 6:
            if (dat == '$')           //当检测到正确的用户下载命令时
                IAP_CONTR = 0x60;   //复位到系统区
            else goto L_Check1st;
            break;
        }
    }
}

void main()
{
    P3M0 = 0x00;
    P3M1 = 0x00;
```

```
SCON = 0x50;                                //设置用户串口模式为8位数据位
TMOD = 0x00;
AUXR = 0x40;
TH1 = BR2400 >> 8;                         //设置串口波特率为115200
TL1 = BR2400;
TR1 = 1;
ES = 1;
EA = 1;

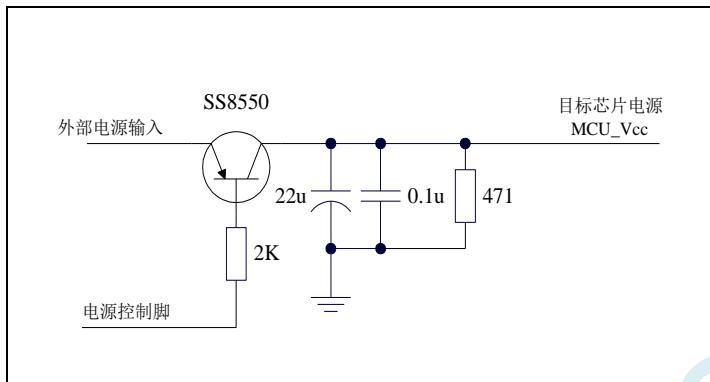
stage = 0;

while (1)
{
    ...
}                                            //用户代码
```

STCMCU

附录N 使用第三方 MCU 对 STC8G 系列单片机进行 ISP 下载范例程序

STC 芯片 ISP 下载需要对目标进行硬件复位才能进入 ISP 下载模式。当使用第三方 MCU 对 STC 芯片进行 ISP 下载时，建议使用下面的电源控制电路来实现。



C 语言代码

//注意: 使用本代码对 STC8G 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           //STC8G 系列目标芯片工作频率
#define RL(n)      (65536 - FUSER/4/(n)) //STC8G 系列目标芯片串口波特率计算公式

sfr AUXR = 0x8e;
sfr P3M1 = 0xB1;
sfr P3M0 = 0xB2;

//变量定义
BOOL fImS;                         //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;
        DelayXms(500);
        P3 = 0xf3;
        DelayXms(500);
        P3 = 0xff;
        DelayXms(500);
        P3 = 0xf3;
        DelayXms(500);
        P3 = 0xff;
    }
}

while (1);
```

```
}

//1ms 定时器中断服务程序
void tm0(void) interrupt 1
{
    static BYTE Counter100;

    f1ms = TRUE;
    if (Counter100-- == 0)
    {
        Counter100 = 100;
        if (TimeOut) TimeOut--;
    }
}

//串口中断服务程序
void uart(void) interrupt 4
{
    static WORD RecvSum;
    static BYTE RecvIndex;
    static BYTE RecvCount;
    BYTE dat;

    if (TI)
    {
        TI = 0;
        UartBusy = FALSE;
    }

    if (RI)
    {
        RI = 0;
        dat = SBUF;
        switch (UartRecvStep)
        {
            case 1:
                if (dat != 0xb9) goto L_CheckFirst;
                UartRecvStep++;
                break;
            case 2:
                if (dat != 0x68) goto L_CheckFirst;
                UartRecvStep++;
                break;
            case 3:
                if (dat != 0x00) goto L_CheckFirst;
                UartRecvStep++;
                break;
            case 4:
                RecvSum = 0x68 + dat;
                RecvCount = dat - 6;
                RecvIndex = 0;
                UartRecvStep++;
                break;
            case 5:
                RecvSum += dat;
                RxBuffer[RecvIndex++] = dat;
                if (RecvIndex == RecvCount) UartRecvStep++;
                break;
            case 6:
```

```
if (dat != HIBYTE(RecvSum))    goto L_CheckFirst;
UartRecvStep++;
break;
case 7:
if (dat != LOBYTE(RecvSum))    goto L_CheckFirst;
UartRecvStep++;
break;
case 8:
if (dat != 0x16) goto L_CheckFirst;
UartReceived = TRUE;
UartRecvStep++;
break;
L_CheckFirst:
case 0:
default:
CommInit();
UartRecvStep = (dat == 0x46 ? 1 : 0);
break;
}
}
}

//系统初始化
void Initial(void)
{
UartBusy = FALSE;

SCON = 0xd0;                                //串口数据模式必须为8位数据+1位偶检验
AUXR = 0xc0;
TMOD = 0x00;
TH0 = HIBYTE(TIMS);
TL0 = LOBYTE(TIMS);
TR0 = 1;
TH1 = HIBYTE(BR(MINBAUD));
TL1 = LOBYTE(BR(MINBAUD));
TR1 = 1;
ET0 = 1;
ES = 1;
EA = 1;
}

//Xms 延时程序
void DelayXms(WORD x)
{
do
{
f1ms = FALSE;
while (!f1ms);
} while (x--);
}

//串口数据发送程序
BYTE UartSend(BYTE dat)
{
while (UartBusy);

UartBusy = TRUE;
ACC = dat;
TB8 = P;
```

```
SBUF = ACC;  
  
    return dat;  
}  
  
//串口通讯初始化  
void CommInit(void)  
{  
    UartRecvStep = 0;  
    TimeOut = 20;  
    UartReceived = FALSE;  
}  
  
//发送串口通讯数据包  
void CommSend(BYTE size)  
{  
    WORD sum;  
    BYTE i;  
  
    UartSend(0x46);  
    UartSend(0xb9);  
    UartSend(0x6a);  
    UartSend(0x00);  
    sum = size + 6 + 0x6a;  
    UartSend(size + 6);  
    for (i=0; i<size; i++)  
    {  
        sum += UartSend(TxBuffer[i]);  
    }  
    UartSend(HIBYTE(sum));  
    UartSend(LOBYTE(sum));  
    UartSend(0x16);  
    while (UartBusy);  
  
    CommInit();  
}  
  
//对STC15H 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] = 0x97;
CommSend(8);
while (1)
{
    if (TimeOut == 0) return FALSE;
    if (UartReceived)
    {
        if (RxBuffer[0] == 0x01) break;
        return FALSE;
    }
}

//准备
TH1 = HIBYTE(BR(MAXBAUD));
TL1 = LOBYTE(BR(MAXBAUD));
DelayXms(10);
TxBuffer[0] = 0x05;
TxBuffer[1] = 0x00;
TxBuffer[2] = 0x00;
TxBuffer[3] = 0x5a;
TxBuffer[4] = 0xa5;
CommSend(5);
while (1)
{
    if (TimeOut == 0) return FALSE;
    if (UartReceived)
    {
        if (RxBuffer[0] == 0x05) break;
        return FALSE;
    }
}

//擦除
DelayXms(10);
TxBuffer[0] = 0x03;
TxBuffer[1] = 0x00;
TxBuffer[2] = 0x00;
TxBuffer[3] = 0x5a;
TxBuffer[4] = 0xa5;
CommSend(5);
TimeOut = 100;
while (1)
{
    if (TimeOut == 0) return FALSE;
    if (UartReceived)
    {
        if (RxBuffer[0] == 0x03) break;
        return FALSE;
    }
}
```

```

        }

    }

//写用户代码
DelayXms(10);
addr = 0;
TxBuffer[0] = 0x22;
TxBuffer[3] = 0x5a;
TxBuffer[4] = 0xa5;
offset = 5;
while (addr < size)
{
    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I/O 口
//TxBuffer[42] = 0xbd;           //P5.4 为复位脚
//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,0x75,0xB1,0x00,0x05,0xB0,0x11,0x0E,0x80,0xFA,0xD8,0xFE,
    0xD9,0xFC,0x22,
};
```

STCMCU

附录O 使用第三方应用程序调用 STC 发布项目 程序对单片机进行 ISP 下载

使用 STC 的 ISP 下载软件生成的发布项目程序为可执行的 EXE 格式文件，用户可直接双击发布的项目程序运行进行 ISP 下载，也可在第三方的应用程序中调用发布项目程序进行 ISP 下载。下面介绍两种调用的方法。

简单调用

在第三方应用程序中只是简单创建发布项目程序的进程，其他的所有下载操作均在发布项目程序中进行，第三方应用程序此时只需要等待发布项目程序操作完成后，清理现场即可。

VC 代码

```
BOOL IspProcess()
{
    //定义相关变量
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    CString path;

    //发布项目程序的完整路径
    path = _T("D:\Work\Upgrade.exe");

    //变量初始化
    memset(&si, 0, sizeof(STARTUPINFO));
    memset(&pi, 0, sizeof(PROCESS_INFORMATION));

    //设置启动变量
    si.cb = sizeof(STARTUPINFO);
    GetStartupInfo(&si);
    si.wShowWindow = SW_SHOWNORMAL;
    si.dwFlags = STARTF_USESHOWWINDOW;

    //创建发布项目程序进程
    if (CreateProcess(NULL, (LPTSTR)(LPCTSTR)path, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))
    {
        //等待发布项目程序操作完成
        //由于此处会阻塞主进程，所以建议新建工作进程，在工作进程中进行等待
        WaitForSingleObject(pi.hProcess, INFINITE);

        //清理工作
        CloseHandle(pi.hThread);
        CloseHandle(pi.hProcess);

        return TRUE;
    }
    else
    {
        AfxMessageBox(_T("创建进程失败 !"));
    }
}
```

```

        return FALSE;
    }
}

```

高级调用

在第三方应用程序创建发布项目程序的进程，并在第三方应用程序中进行包括选择串口、开始 ISP 编程、停振 ISP 编程以及关闭发布项目程序等的全部 ISP 下载操作，而不需要在发布项目程序中进行界面互动。

VC 代码

```

//定义回调函数参数的数据结构
struct CALLBACK_PARAM
{
    DWORD dwProcessId;           //主进程ID
    HWND hMainWnd;               //主窗口句柄
};

//枚举窗口的回调函数，用于获取主窗口句柄
BOOL CALLBACK EnumWindowCallBack(HWND hWnd, LPARAM lParam)
{
    CALLBACK_PARAM *pcp = (CALLBACK_PARAM *)lParam;
    DWORD id;

    GetWindowThreadProcessId(hWnd, &id);
    if ((pcp->dwProcessId == id) && (GetParent(hWnd) == NULL))
    {
        pcp->hMainWnd = hWnd;

        return FALSE;
    }

    return TRUE;
}

BOOL IspProcess()
{
    //定义相关变量
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    CALLBACK_PARAM cp;
    CString path;

    //发布项目程序中部分控件的ID
    const UINT ID_PROGRAM      = 1013;
    const UINT ID_STOP         = 1012;
    const UINT ID_COMPORT      = 1001;
    const UINT ID_PROGRESS     = 1000;

    //发布项目程序的完整路径
    path = _T("D:\\Work\\Upgrade.exe");

    //变量初始化
    memset(&si, 0, sizeof(STARTUPINFO));
    memset(&pi, 0, sizeof(PROCESS_INFORMATION));
    memset(&cp, 0, sizeof(CALLBACK_PARAM));
}

```

```
//设置启动变量
si.cb = sizeof(STARTUPINFO);
GetStartupInfo(&si);
si.wShowWindow = SW_SHOWNORMAL;
//此处若设置为SW_HIDE,就不会显示发布项目程序
//的操作界面,所有的ISP 操作都可在后台进行
si.dwFlags = STARTF_USESHOWWINDOW;

//创建发布项目程序进程
if (CreateProcess(NULL, (LPTSTR)(LPCTSTR)path, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))
{
    //等待发布项目程序进程初始化完成
    WaitForInputIdle(pi.hProcess, 5000);

    //获取发布项目程序的主窗口句柄
    cp.dwProcessId = pi.dwProcessId;
    cp.hMainWnd = NULL;
    EnumWindows(EnumWindowCallBack, (LPARAM)&cp);

    if (cp.hMainWnd != NULL)
    {
        HWND hProgram;
        HWND hStop;
        HWND hPort;

        //获取发布项目程序主窗口中部分控件句柄
        hProgram = ::GetDlgItem(cp.hMainWnd, ID_PROGRAM);
        hStop = ::GetDlgItem(cp.hMainWnd, ID_STOP);
        hPort = ::GetDlgItem(cp.hMainWnd, ID_COMPORT);

        //设置发布项目程序中的串口号, 第3 个参数为0:COM1, 1:COM2, 2:COM3, ...
        ::SendMessage(hPort, CB_SETCURSEL, 0, 0);

        //触发编程按钮开始ISP 编程
        ::SendMessage(hProgram, BM_CLICK, 0, 0);

        //等待编程完成,
        //由于此处会阻塞主进程, 所以建议新建工作进程, 在工作进程中进行等待
        while (!::IsWindowEnabled(hProgram));

        //编程完成后关闭发布项目程序
        ::SendMessage(cp.hMainWnd, WM_CLOSE, 0, 0);
    }

    //等待进程结束
    WaitForSingleObject(pi.hProcess, INFINITE);

    //清理工作
    CloseHandle(pi.hThread);
    CloseHandle(pi.hProcess);

    return TRUE;
}
else
{
    AfxMessageBox(_T("创建进程失败 !"));

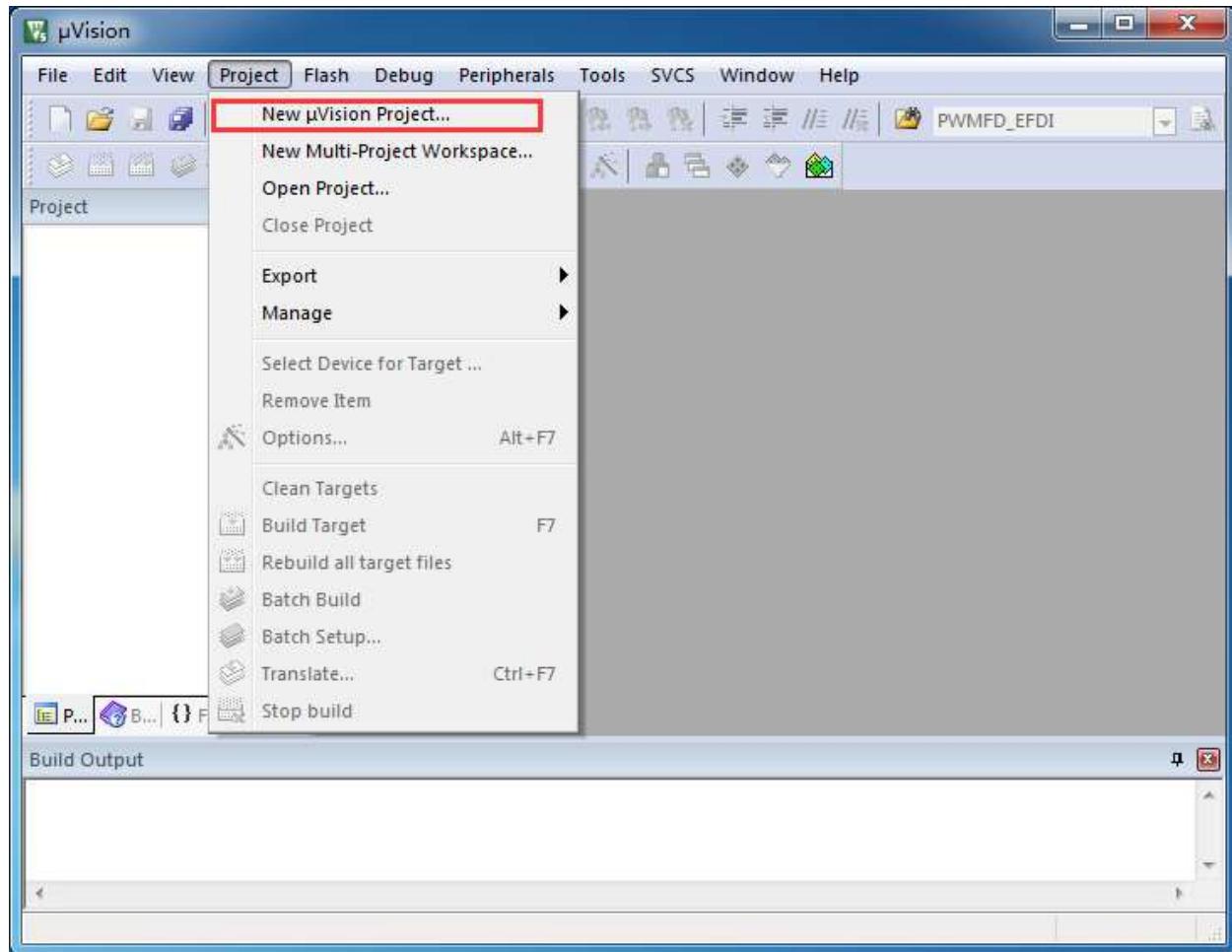
    return FALSE;
}
```

STCMCU

附录P 在 Keil 中建立多文件项目的方法

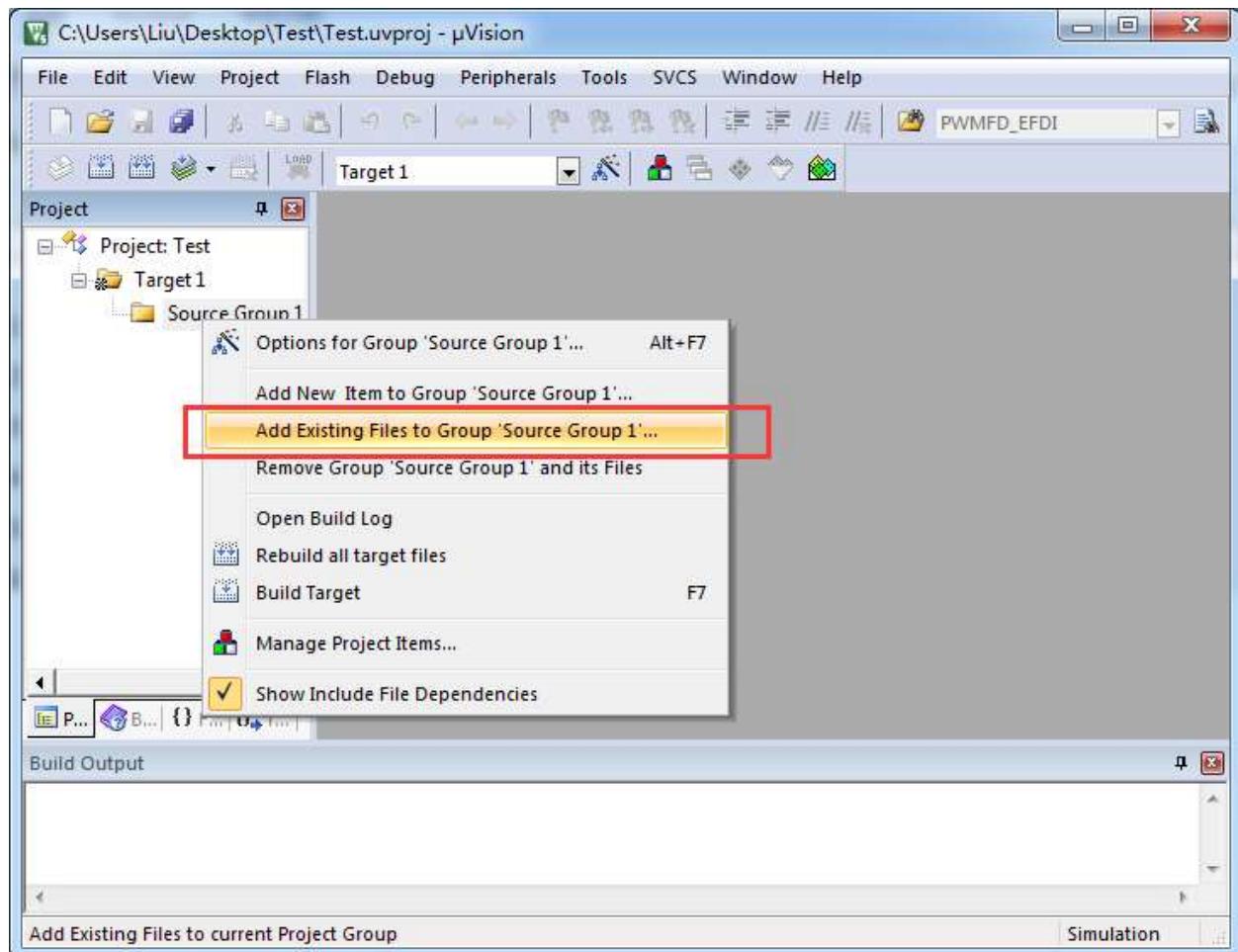
在 Keil 中，一般比较小的项目都只有一个源文件，但对于一些稍微复杂的项目往往需要多个源文件建立多文件项目的方法如下：

1、首先打开 Keil，在菜单“Project”中选择“New uVision Project ...”

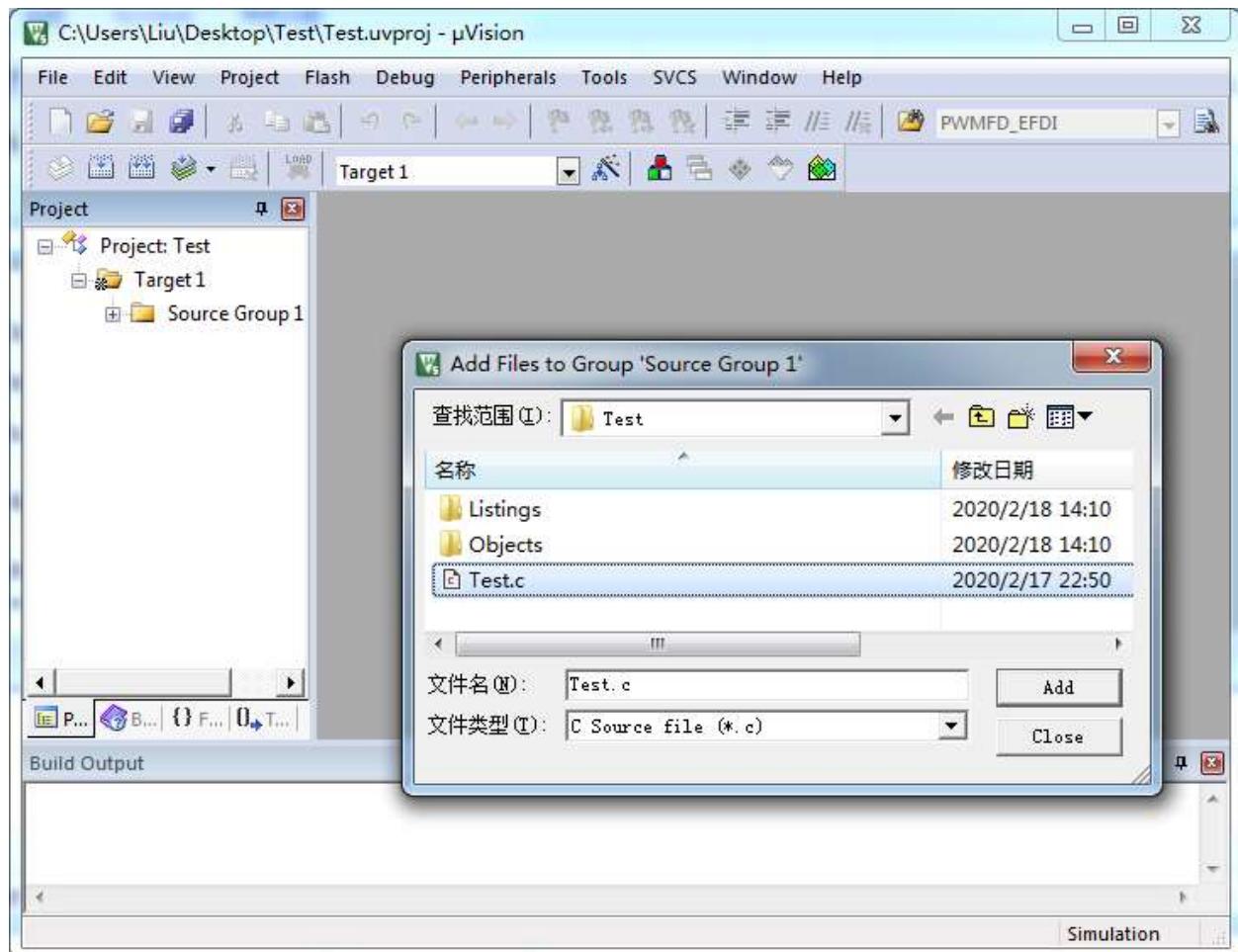


即可完成一个空项目的建立

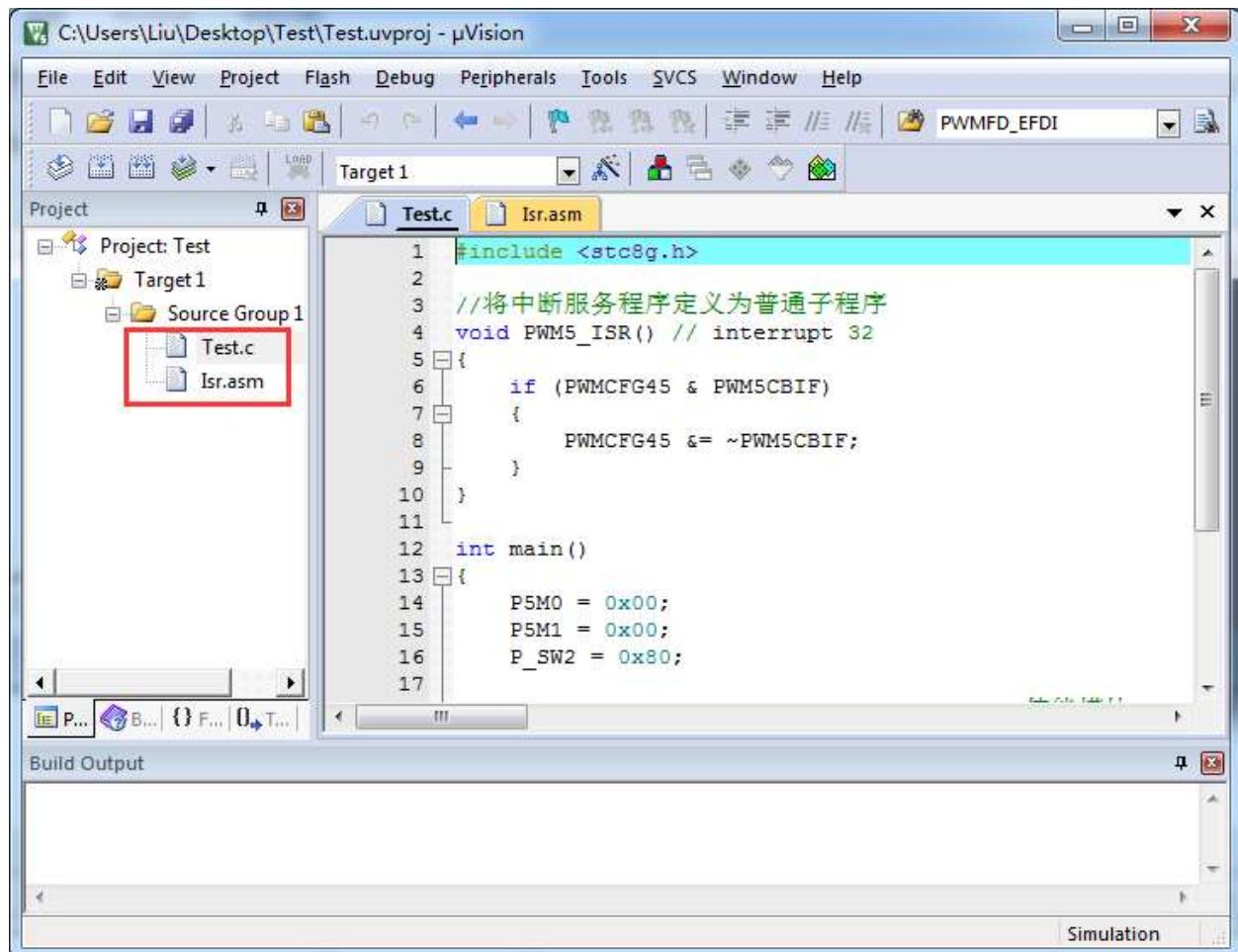
2、在空项目的项目树中，鼠标右键单击“Source Group 1”，并选择右键菜单中的“Add Existing Files to Group "Source Group 1" ...”



3、在弹出的文件对话框中，多次添加源文件

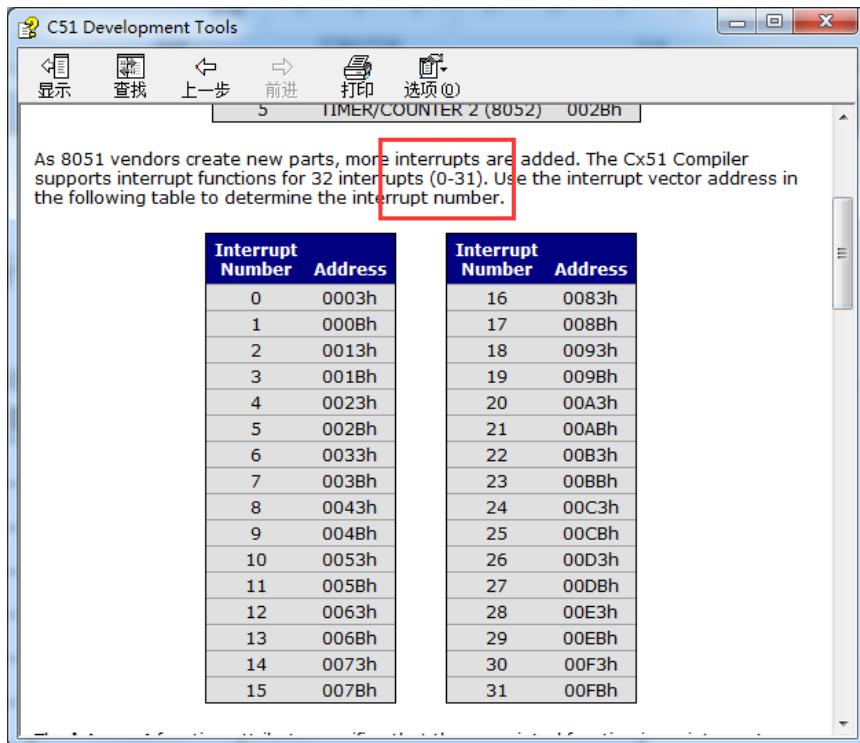


如下图所示即可完成多文件项目的建立



附录Q 关于中断号大于 31 在 Keil 中编译出错的处理

在 Keil 的 C51 编译环境下，中断号只支持 0~31，即中断向量必须小于 0100H。

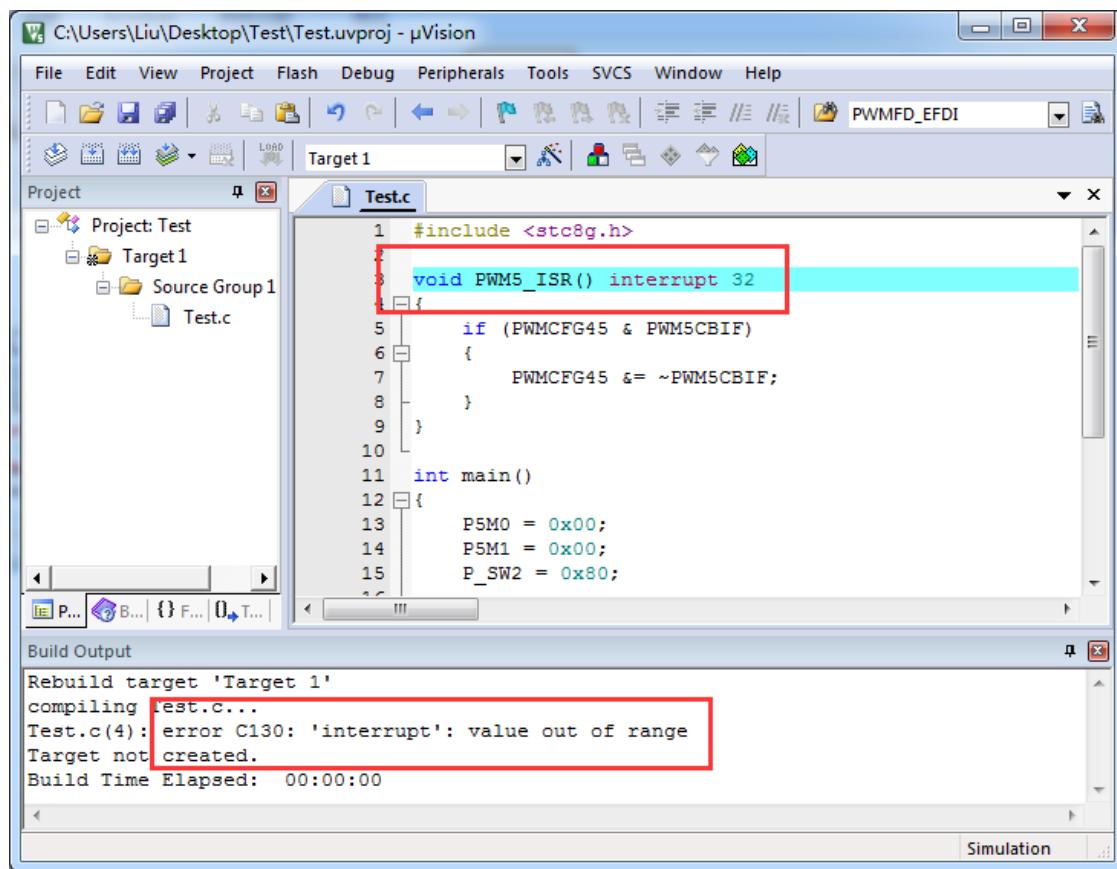


下表是 STC 目前所有系列的中断列表：

中断号	中断向量	中断类型
0	0003 H	INT0
1	000B H	定时器 0
2	0013 H	INT1
3	001B H	定时器 1
4	0023 H	串口 1
5	002B H	ADC
6	0033 H	LVD
7	003B H	PCA
8	0043 H	串口 2
9	004B H	SPI
10	0053 H	INT2
11	005B H	INT3
12	0063 H	定时器 2
13	006B H	
14	0073 H	系统内部中断

15	007B H	系统内部中断
16	0083 H	INT4
17	008B H	串口 3
18	0093 H	串口 4
19	009B H	定时器 3
20	00A3 H	定时器 4
21	00AB H	比较器
22	00B3 H	波形发生器 0
23	00BB H	波形发生器异常 0
24	00C3 H	I2C
25	00CB H	USB
26	00D3 H	PWM1
27	00DB H	PWM2
28	00E3 H	波形发生器 1
29	00EB H	波形发生器 2
30	00F3 H	波形发生器 3
31	00FB H	波形发生器 4
32	0103 H	波形发生器 5
33	010B H	波形发生器异常 2
34	0113 H	波形发生器异常 4
35	011B H	触摸按键
36	0123 H	RTC
37	012B H	P0 口中断
38	0133 H	P1 口中断
39	013B H	P2 口中断
40	0143 H	P3 口中断
41	014B H	P4 口中断
42	0153 H	P5 口中断
43	015B H	P6 口中断
44	0163 H	P7 口中断
45	016B H	P8 口中断
46	0173 H	P9 口中断

不难发现，从波形发生器 5 中断开始，后面所有的中断服务程序，在 keil 中均会编译出错，如下图所示：



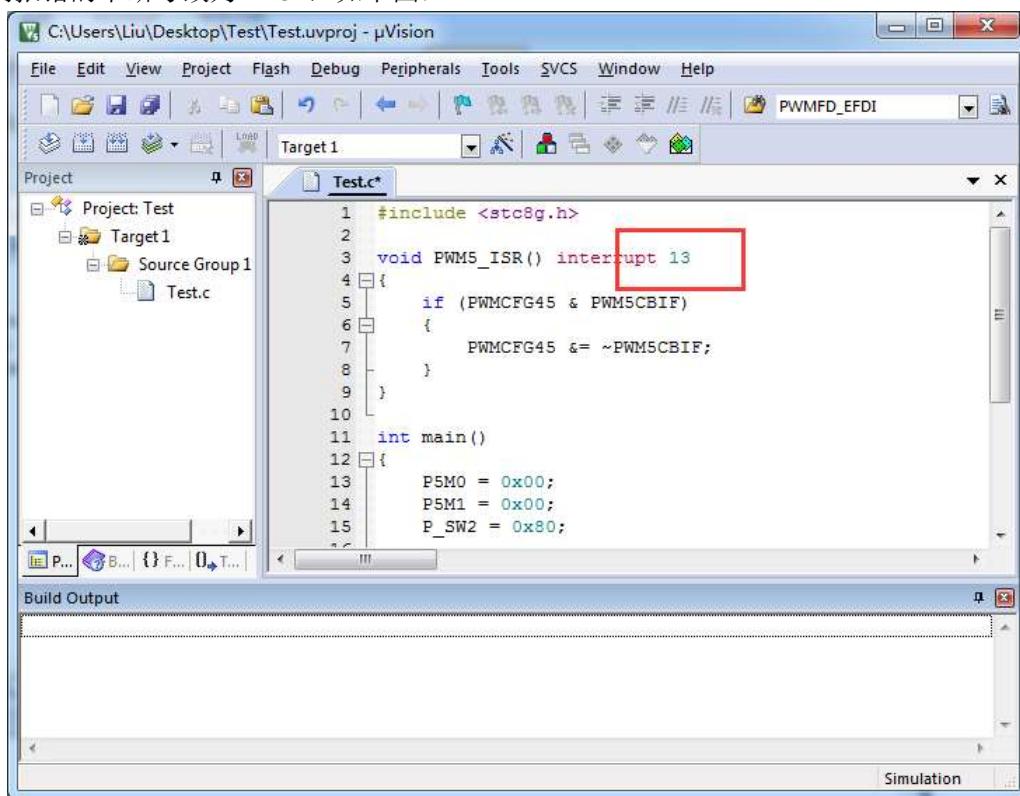
处理这种错误有如下三种方法: (均需要借助于汇编代码, 优先推荐使用方法 1)

方法 1：借用 13 号中断向量

0~31 号中断中，第 13 号是保留中断号，我们可以借用此中断号

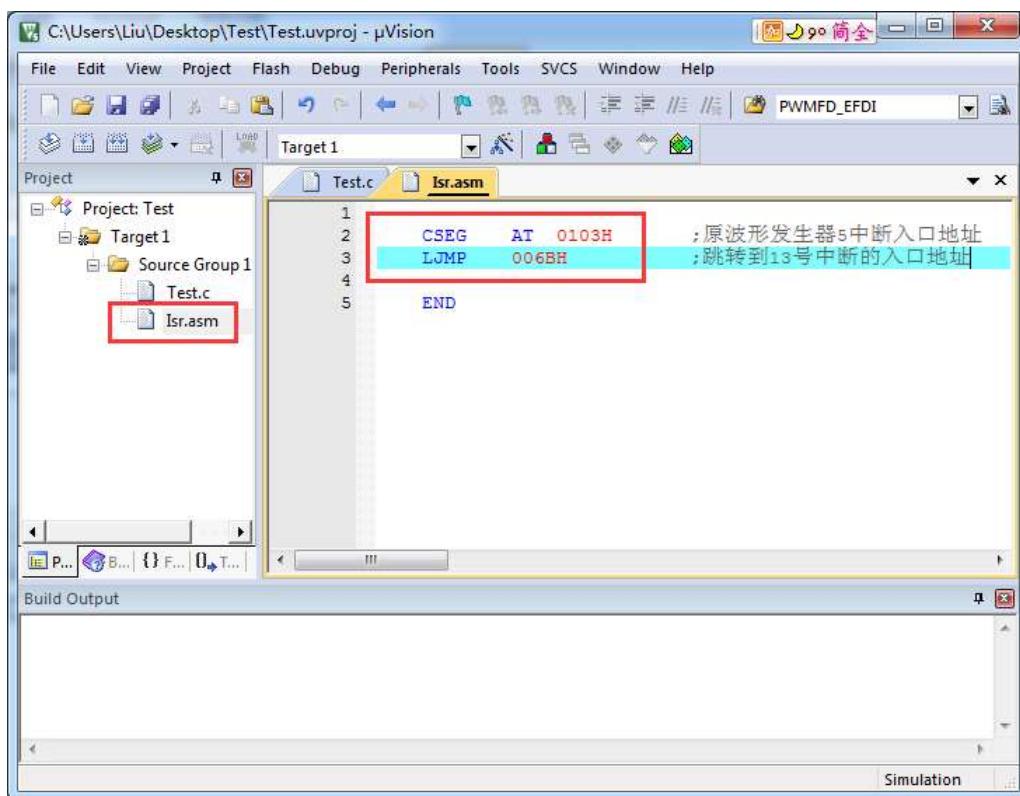
操作步骤如下：

1、将我们报错的中断号改为“13”，如下图：



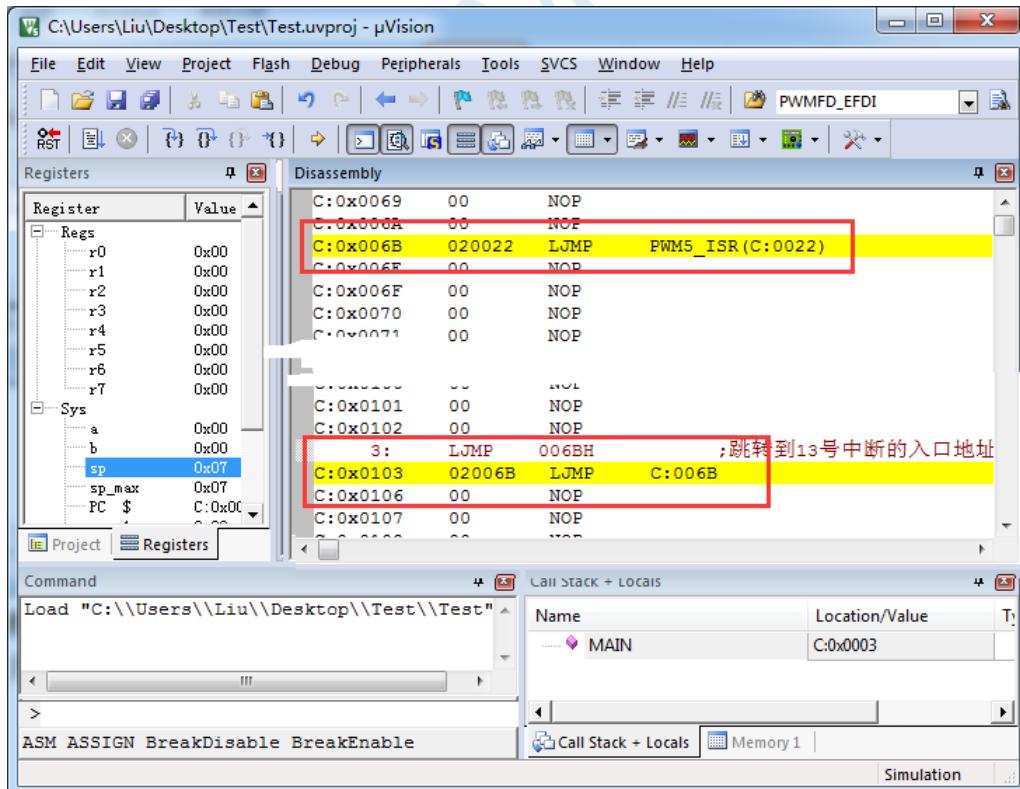
```
1 #include <stc8g.h>
2
3 void PWM5_ISR() interrupt 13
4 {
5     if (PWMCFG45 & PWM5CBIF)
6     {
7         PWMCFG45 &= ~PWM5CBIF;
8     }
9 }
10
11 int main()
12 {
13     P5M0 = 0x00;
14     P5M1 = 0x00;
15     P_SW2 = 0x80;
```

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

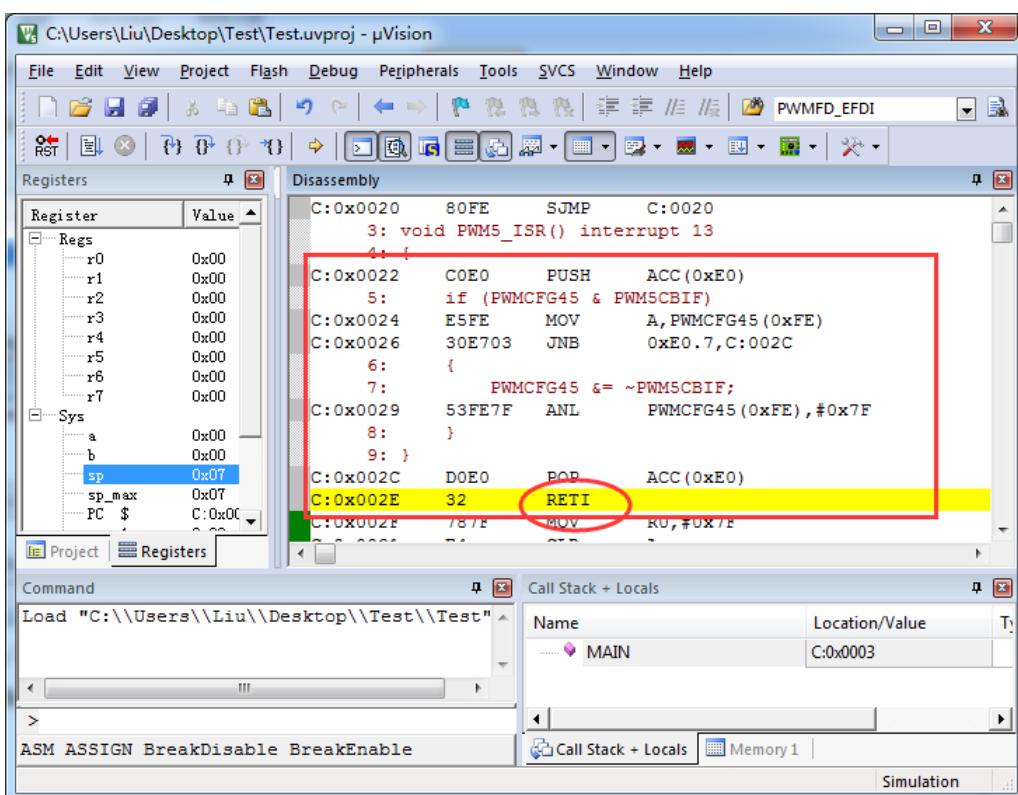


3、编译即可通过。

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



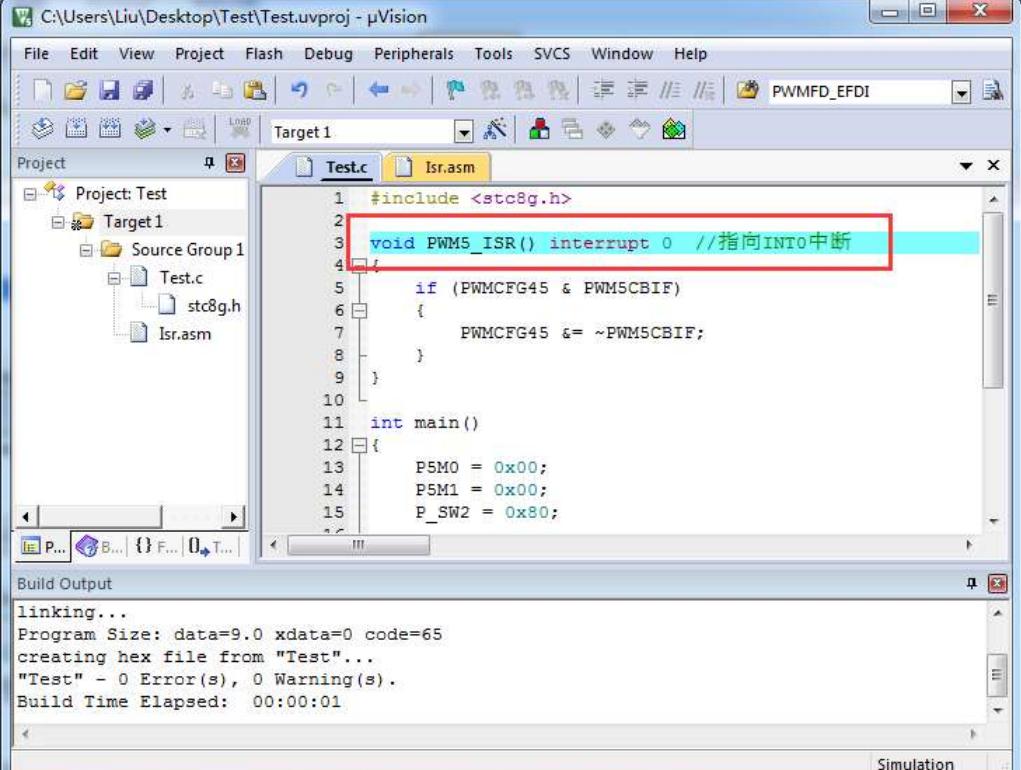
当发生 PWM5 中断时，硬件会自动跳转到 0103H 地址执行“LJMP 006BH”，然后在 006BH 处再执行“LJMP PWM5_ISR”即可跳转到真正的中断服务程序，如下图：



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

方法 2: 与方法 1 类似, 借用用户程序中未使用的 0~31 的中断号

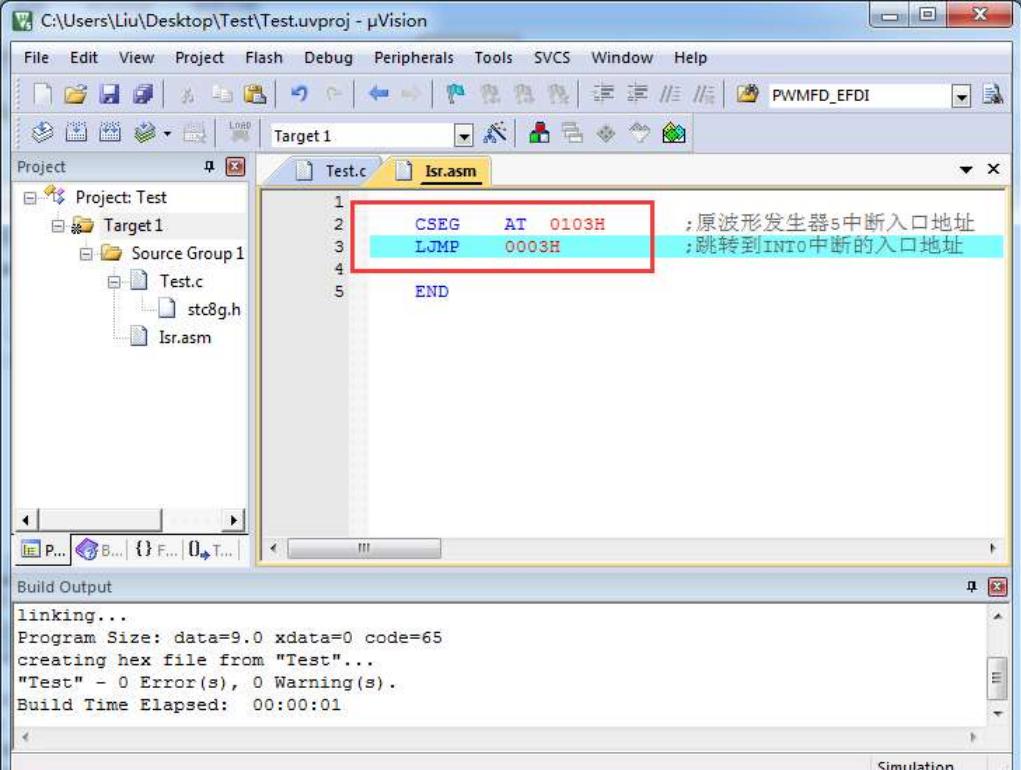
比如在用户的代码中, 没有使用 INT0 中断, 则可将上面的代码作类似与方法 1 的修改:



```

1 #include <stc8g.h>
2
3 void PWM5_ISR() interrupt 0 //指向INT0中断
4 {
5     if (PWMCFG45 & PWM5CBIF)
6     {
7         PWMCFG45 &= ~PWM5CBIF;
8     }
9 }
10
11 int main()
12 {
13     P5M0 = 0x00;
14     P5M1 = 0x00;
15     P_SW2 = 0x80;

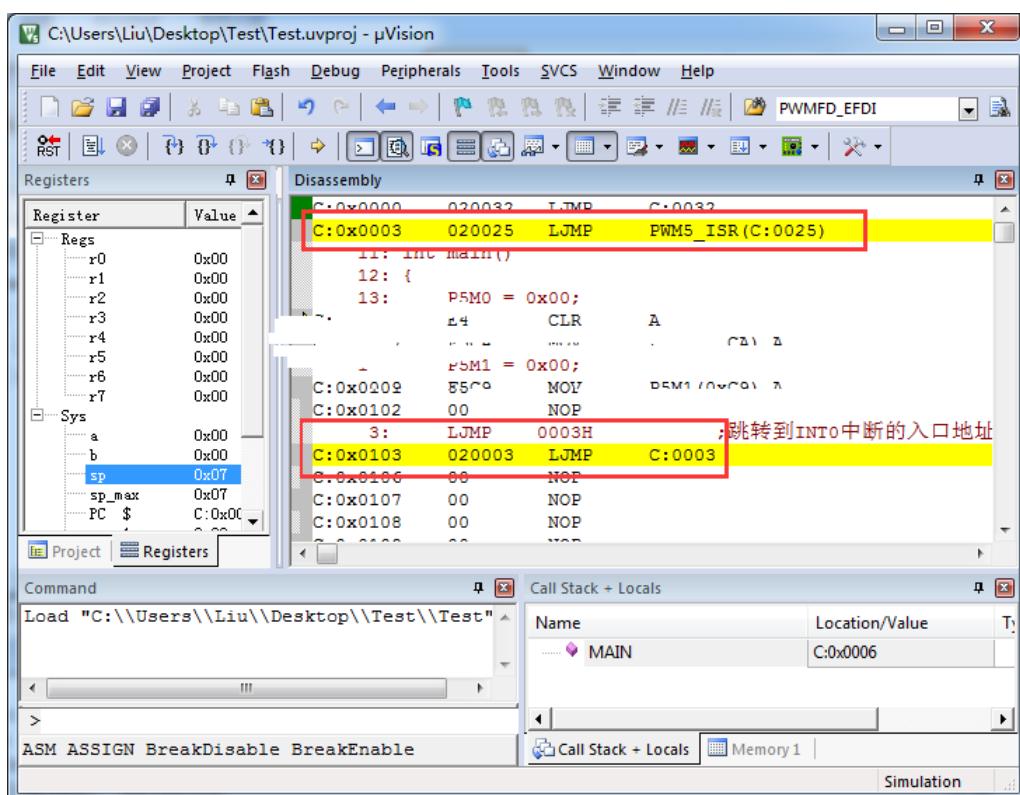
```



```

1 CSEG      AT 0103H
2 LJMP      0003H ;原波形发生器5中断入口地址
3
4 END

```

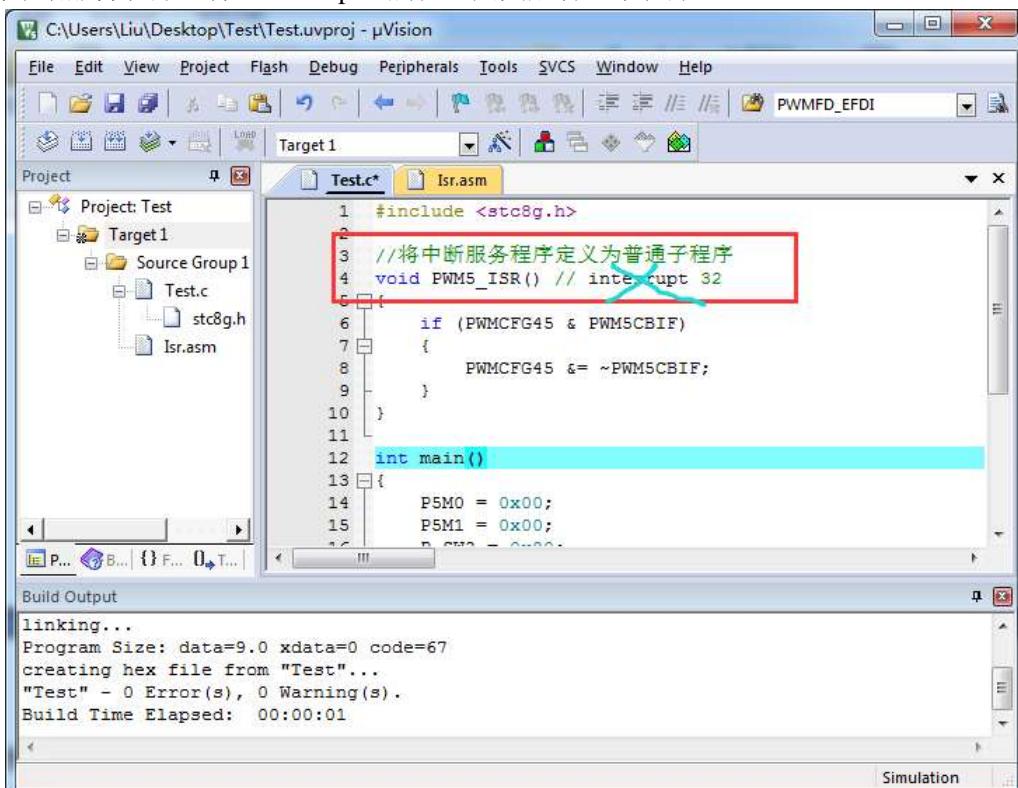


执行效果与方法 1 相同，此方法适用于需要重映射多个中断号大于 31 的情况。

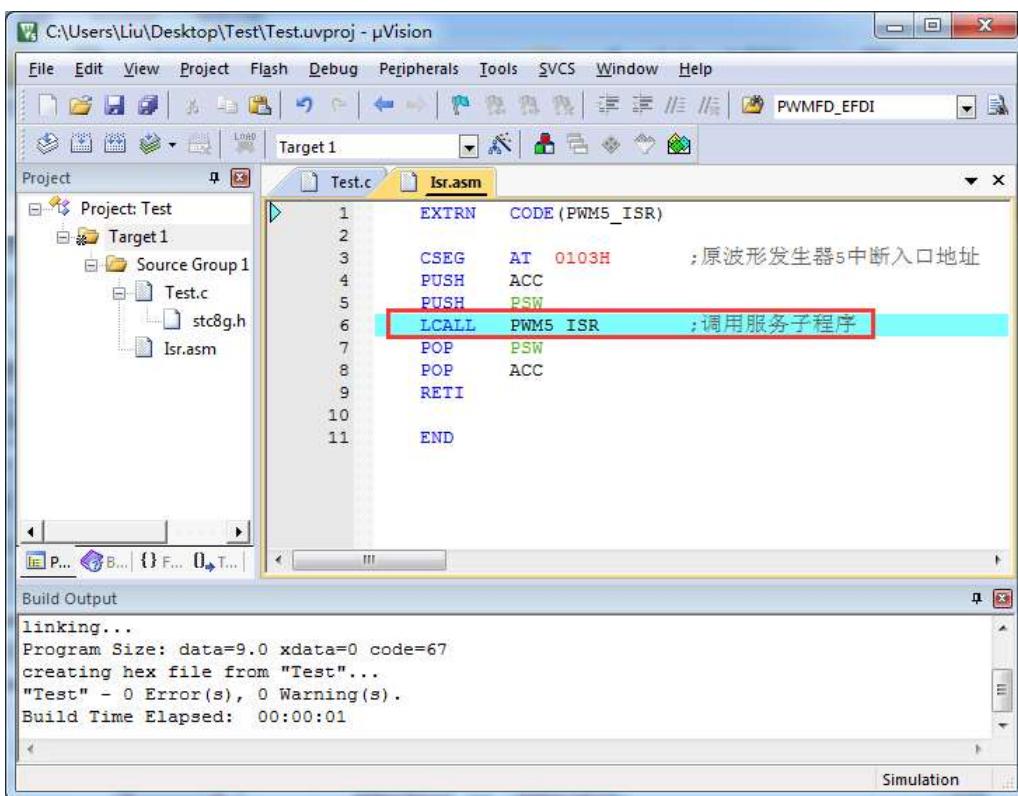
方法 3: 将中断服务程序定义成子程序, 然后在汇编代码中的中断入口地址中使用 LCALL 指令执行服务程序

操作步骤如下:

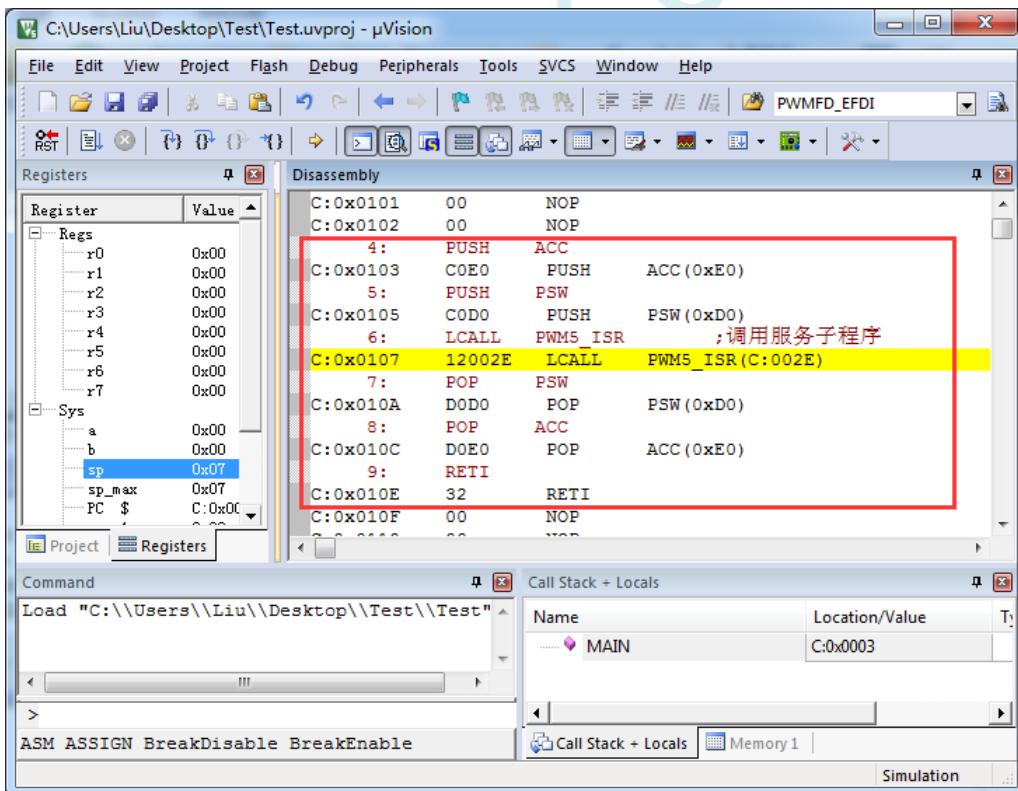
- 首先将中断服务程序去掉“interrupt”属性, 定义成普通子程序



- 然后在汇编文件的 0103H 地址输入如下图所示的代码



3、编译通过后，即可发现在 0103H 地址的地方即为中断服务程序



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

附录R 电气特性

R.1 绝对最大额定值

参数	最小值	最大值	单位	说明
存储温度	-55	+150	°C	
工作温度	-40	+85	°C	若工作温度高于 85°C (如 125°C 附近)，由于内部 IRC 时钟的频率在高温时的温漂大，建议使用外部高温时钟或晶振。另外温度高时频率跑不快，建议使用 24M 以下的工作频率；如果系统必须运行在较高温度，务请使用外部高可靠低频有源时钟。 若工作温度为 -55°C 附近，则工作电压不能太低，强烈建议 MCU-VCC 电压不要低于 3.0V，另外电源的上升速度也必须尽量快，最好能控制在毫秒级
工作电压	1.9	5.5	V	
VDD 对地电压	-0.3	+5.5	V	
I/O 口对地电压	-0.3	VDD+0.3	V	

R.2 直流特性 (3.3V)

(VSS=0V, VDD=3.3V, 测试温度=25°C)

标号	参数	范围				测试环境及说明
		最小值	典型值	最大值	单位	
I _{PD}	掉电模式电流	-	0.4	-	uA	
I _{WKT}	掉电唤醒定时器	-	1.4	-	uA	
I _{LVD}	低压检测模块功耗	-	10	-	uA	
I _{CMP}	比较器功耗	-	90	-	uA	
I _{IDL}	空闲模式电流 (内部 32KHz)	-	0.48	-	mA	相当于传统 8051 的 0.5M
	空闲模式电流 (6MHz)	-	0.88	-	mA	相当于传统 8051 的 79M
	空闲模式电流 (12MHz)	-	1.00	-	mA	相当于传统 8051 的 158M
	空闲模式电流 (24MHz)	-	1.16	-	mA	相当于传统 8051 的 317M
I _{NOR}	正常模式电流 (内部 32KHz)	-	0.48	-	mA	相当于传统 8051 的 0.5M
	正常模式电流 (500KHz)	-	0.88	-	mA	相当于传统 8051 的 7M
	正常模式电流 (600KHz)	-	0.88	-	mA	相当于传统 8051 的 8M
	正常模式电流 (700KHz)	-	0.90	-	mA	相当于传统 8051 的 9M
	正常模式电流 (800KHz)	-	0.91	-	mA	相当于传统 8051 的 11M
	正常模式电流 (900KHz)	-	0.91	-	mA	相当于传统 8051 的 12M
	正常模式电流 (1MHz)	-	0.94	-	mA	相当于传统 8051 的 13M
	正常模式电流 (2MHz)	-	1.05	-	mA	相当于传统 8051 的 26M
	正常模式电流 (3MHz)	-	1.17	-	mA	相当于传统 8051 的 40M
	正常模式电流 (4MHz)	-	1.26	-	mA	相当于传统 8051 的 53M
	正常模式电流 (5MHz)	-	1.40	-	mA	相当于传统 8051 的 66M
	正常模式电流 (6MHz)	-	1.49	-	mA	相当于传统 8051 的 79M
	正常模式电流 (12MHz)	-	2.09	-	mA	相当于传统 8051 的 158M
	正常模式电流 (24MHz)	-	3.16	-	mA	相当于传统 8051 的 317M
V _{IL1}	输入低电平	-	-	0.99	V	打开施密特触发
		-	-	1.07	V	关闭施密特触发
V _{IH1}	输入高电平 (普通 I/O)	1.18	-	-	V	打开施密特触发
		1.09	-	-	V	关闭施密特触发
V _{IH2}	输入高电平 (复位脚)	1.18	-	0.99	V	
I _{OL1}	输出低电平的灌电流	-	20	-	mA	端口电压 0.45V
I _{OH1}	输出高电平电流 (双向模式)	200	270	-	uA	
I _{OH2}	输出高电平电流 (推挽模式)	-	20	-	mA	端口电压 2.4V
I _{IL}	逻辑 0 输入电流	-	-	50	uA	端口电压 0V
I _{TL}	逻辑 1 到 0 的转移电流	100	270	600	uA	端口电压 2.0V
R _{PU}	I/O 口上拉电阻	5.8	5.9	6.0	KΩ	
I/O 速度	I/O 大电流驱动, I/O 快速转换		25		MHz	PxDR=0, PxSR=0
	I/O 小电流驱动, I/O 快速转换		22		MHz	PxDR=1, PxSR=0
	I/O 大电流驱动, I/O 慢速转换		16		MHz	PxDR=0, PxSR=1
	I/O 小电流驱动, I/O 慢速转换		12		MHz	PxDR=1, PxSR=1
比较器	最快速度		10		MHz	关闭所有模拟和数字滤波

	模拟滤波时间		0.1		us	
	数字滤波时间		0		系统	LCDTY=0
			n+2		时钟	LCDTY=n (n=1~63)
I _{PD2}	使能比较器时掉电模式功耗	-	400	-	uA	
I _{PD3}	使能 LVD 时掉电模式功耗	-	470	-	uA	

STCMCU

R.3 直流特性 (5.0V)

(VSS=0V, VDD=5.0V, 测试温度=25°C)

标号	参数	范围				测试环境及说明
		最小值	典型值	最大值	单位	
I _{PD}	掉电模式电流	-	0.6	-	uA	
I _{WKT}	掉电唤醒定时器	-	3.6	-	uA	
I _{LVD}	低压检测模块功耗	-	30	-	uA	
I _{CMP}	比较器功耗	-	90	-	uA	
I _{IDL}	空闲模式电流 (内部 32KHz)	-	0.58	-	mA	相当于传统 8051 的 0.5M
	空闲模式电流 (6MHz)	-	0.98	-	mA	相当于传统 8051 的 79M
	空闲模式电流 (12MHz)	-	1.10	-	mA	相当于传统 8051 的 158M
	空闲模式电流 (24MHz)	-	1.25	-	mA	相当于传统 8051 的 317M
I _{NOR}	正常模式电流 (内部 32KHz)	-	0.58	-	mA	相当于传统 8051 的 0.5M
	正常模式电流 (500KHz)		0.97		mA	相当于传统 8051 的 7M
	正常模式电流 (600KHz)		0.97		mA	相当于传统 8051 的 8M
	正常模式电流 (700KHz)		1.00		mA	相当于传统 8051 的 9M
	正常模式电流 (800KHz)		1.01		mA	相当于传统 8051 的 11M
	正常模式电流 (900KHz)		1.01		mA	相当于传统 8051 的 12M
	正常模式电流 (1MHz)		1.03		mA	相当于传统 8051 的 13M
	正常模式电流 (2MHz)		1.15		mA	相当于传统 8051 的 26M
	正常模式电流 (3MHz)		1.27		mA	相当于传统 8051 的 40M
	正常模式电流 (4MHz)		1.35		mA	相当于传统 8051 的 53M
	正常模式电流 (5MHz)		1.49		mA	相当于传统 8051 的 66M
	正常模式电流 (6MHz)	-	1.59	-	mA	相当于传统 8051 的 79M
	正常模式电流 (12MHz)	-	2.19	-	mA	相当于传统 8051 的 158M
	正常模式电流 (24MHz)	-	3.27	-	mA	相当于传统 8051 的 317M
V _{IL1}	输入低电平	-	-	1.32	V	打开施密特触发
		-	-	1.48	V	关闭施密特触发
V _{IH1}	输入高电平 (普通 I/O)	1.60	-	-	V	打开施密特触发
		1.54	-	-	V	关闭施密特触发
V _{IH2}	输入高电平 (复位脚)	1.60	-	1.32	V	
I _{OL1}	输出低电平的灌电流	-	20	-	mA	端口电压 0.45V
I _{OH1}	输出高电平电流 (双向模式)	200	270	-	uA	
I _{OH2}	输出高电平电流 (推挽模式)	-	20	-	mA	端口电压 2.4V
I _{IL}	逻辑 0 输入电流	-	-	50	uA	端口电压 0V
I _{TL}	逻辑 1 到 0 的转移电流	100	270	600	uA	端口电压 2.0V
R _{PU}	I/O 口上拉电阻	4.1	4.2	4.4	KΩ	
I/O 速度	I/O 大电流驱动, I/O 快速转换		36		MHz	PxDR=0, PxSR=0
	I/O 小电流驱动, I/O 快速转换		32		MHz	PxDR=1, PxSR=0
	I/O 大电流驱动, I/O 慢速转换		26		MHz	PxDR=0, PxSR=1
	I/O 小电流驱动, I/O 慢速转换		22		MHz	PxDR=1, PxSR=1
比较器	最快速度		10		MHz	关闭所有模拟和数字滤波

	模拟滤波时间		0.1		us	
	数字滤波时间		0		系统	LCDTY=0
			n+2		时钟	LCDTY=n (n=1~63)
I _{PD2}	使能比较器时掉电模式功耗	-	460	-	uA	
I _{PD3}	使能 LVD 时掉电模式功耗	-	520	-	uA	

R.4 内部 IRC 温漂特性（参考温度 25°C）

温度	范围		
	最小值	典型值	最大值
-40°C ~ 85°C		-1.38% ~ +1.42%	
-20°C ~ 65°C		-0.88% ~ +1.05%	

R.5 低压复位门槛电压（测试温度 25°C）

级别	电压		
	最小值	典型值 (实测值)	最大值
POR		(1.69V~1.82V)	
LVR0		2.0V (1.88V~1.99V)	
LVR1		2.4V (2.28V~2.45V)	
LVR2		2.7V (2.58V~2.76V)	
LVR3		3.0V (2.86V~3.06V)	

附录S 应用注意事项

S.1 STC8G1K08A 系列

1. STC8G1K08A 系列目前量产的是 B 版芯片, 已没有任何问题, 请放心使用。
2. STC8G1K08A 系列 A 版芯片的 PCA 中断的关闭中断指令无法在一个时钟内完成, 用户必须在关闭中断指令后多加 1 个 NOP 指令。(由于使能或者关闭 EA 总中断能够在一个时钟内生效, 若用户需要立即屏蔽中断, 可使用关 EA 的方法)。此问题不影响芯片的正常使用。
3. **特别注意: 由于 STC8G 系列的所有 I/O (除了 ISP 下载口 P3.0/P3.1 外) 在上电后都是高阻输入模式, I/O 外部电平不固定, 此时如果 MCU 直接进入掉电模式/停机模式, 会导致 I/O 有额外的耗电, 所有在 MCU 进入掉电模式/停机模式前, 必须将所有 I/O 口都根据实际情况设置好 I/O 口的模式, 对于所有没有使用的外部悬空的 I/O 都需要设置为准双向口, 并固定输出高电平。特别是部分管脚的芯片, 由于芯片内部有部分 I/O 口并没有打线到外部管脚, 所以这些 I/O 也是处于悬空状态的, 这部分 I/O 也需要设置为准双向口, 并固定输出高电平。**

S.2 STC8G2K64S4/S2 系列

1. STC8G2K64S4/S2 系列目前量产的 C 版芯片, 除 PCA 中断的关闭中断指令无法在一个时钟内完成这个问题外, A 版和 B 版的其他已知问题均已全部修改正确。
2. STC8G2K64S4/S2 系列目前量产的 B 版芯片, PCA 脉冲输出问题和 P2.0/P2.1 口输出 PWM 波形的问题已修改正确。
3. STC8G2K64S4/S2 系列目前量产的 B 版芯片的 PCA 中断的关闭中断指令无法在一个时钟内完成, 用户必须在关闭中断指令后多加 1 个 NOP 指令。(由于使能或者关闭 EA 总中断能够在一个时钟内生效, 若用户需要立即屏蔽中断, 可使用关 EA 的方法)。此问题不影响芯片的正常使用。
4. STC8G2K64S4/S2 系列目前量产的 B 版芯片, 当使能 P0.5 口输出 PWM 波形时, 在发生外部异常时, P0.5 口输出会立即终止, 但硬件并没有将 P0.5 口设置为高阻输入状态, 而是切换为若上拉双向口模式。所以若项目中有需要使能 P0.5 口的 PWM 输出功能, 请注意发生异常时 P0.5 口仍能向外输出 20~30uA 的电流。
5. STC8G2K64S4/S2 系列目前量产的 B 版芯片, 上电后所有端口的电平转换速度控制寄存器的初始值为 00H, 即上电后默认为快速翻转速度, 与其他系列不同, 其他系列的电平转换速度控制寄存器的初始值为 FFH, 即上电后默认为慢速翻转速度。
6. STC8G2K64S4/S2 系列目前量产的 B 版芯片, 当需要使用增强型 PWM 输出波形时, CPU 不能进入省电模式, IDLE 模式/待机模式和 STOP 模式/停机模式都不行。
7. STC8G2K64S4/S2 系列 A 版芯片的 PCA 高速脉冲输出功能会受到同一组 I/O 口翻转的影响, 详情请参考本节 STC8G1K08 系列的参考代码
8. STC8G2K64S4/S2 系列 A 版芯片的增强型 PWM 功能在 P2.0 和 P2.1 口存在 BUG, 其他的 43 个 I/O 口均可正确输出 PWM 波形, 建议不要使用 P2.0 和 P2.1 输出 PWM 波形。
9. STC8G2K64S4/S2 系列 A 版芯片的 PCA 中断的关闭中断指令无法在一个时钟内完成, 用户必须在关闭中断指令后多加 1 个 NOP 指令。(由于使能或者关闭 EA 总中断能够在一个时钟内生效, 若用户需要立即屏蔽中断, 可使用关 EA 的方法)。此问题不影响芯片的正常使用

使用。

10. 特别注意: 由于 STC8G 系列的所有 I/O (除了 ISP 下载口 P3.0/P3.1 外) 在上电后都是高阻输入模式, I/O 外部电平不固定, 此时如果 MCU 直接进入掉电模式/停机模式, 会导致 I/O 有额外的耗电, 所有在 MCU 进入掉电模式/停机模式前, 必须将所有 I/O 口都根据实际情况设置好 I/O 口的模式, 对于所有没有使用的外部悬空的 I/O 都需要设置为准双向口, 并固定输出高电平。特别是部分管脚的芯片, 由于芯片内部有部分 I/O 口并没有打线到外部管脚, 所以这些 I/O 也是处于悬空状态的, 这部分 I/O 也需要设置为准双向口, 并固定输出高电平。

S.3 STC8G1K08 系列

1. STC8G1K08 系列 C 版芯片和 D 版芯片的 LVD 中断、定时器 2 中断、INT2 中断、INT3 中断和 INT4 中断的关闭中断指令无法在一个时钟内完成, 用户必须在关闭中断指令后多加 1 个 NOP 指令。(D 版本芯片在 C 版本芯片的基础上修改了高速脉冲输出会受到同一组 I/O 口翻转影响的问题)
2. STC8G1K08 系列 C 版芯片的 PCA 高速脉冲输出功能会受到同一组 I/O 口翻转的影响, 建议不要使用高速脉冲输出功能 (D 版芯片无此问题)。
3. 特别注意: 由于 STC8G 系列的所有 I/O (除了 ISP 下载口 P3.0/P3.1 外) 在上电后都是高阻输入模式, I/O 外部电平不固定, 此时如果 MCU 直接进入掉电模式/停机模式, 会导致 I/O 有额外的耗电, 所有在 MCU 进入掉电模式/停机模式前, 必须将所有 I/O 口都根据实际情况设置好 I/O 口的模式, 对于所有没有使用的外部悬空的 I/O 都需要设置为准双向口, 并固定输出高电平。特别是部分管脚的芯片, 由于芯片内部有部分 I/O 口并没有打线到外部管脚, 所以这些 I/O 也是处于悬空状态的, 这部分 I/O 也需要设置为准双向口, 并固定输出高电平。

S.4 STC8G1K08T 系列

1. STC8G1K08T 系列 B 版芯片已在送样中, 请停止使用 A 版芯片, 使用 B 版芯片进行测试
2. 特别注意: 由于 STC8G 系列的所有 I/O (除了 ISP 下载口 P3.0/P3.1 外) 在上电后都是高阻输入模式, I/O 外部电平不固定, 此时如果 MCU 直接进入掉电模式/停机模式, 会导致 I/O 有额外的耗电, 所有在 MCU 进入掉电模式/停机模式前, 必须将所有 I/O 口都根据实际情况设置好 I/O 口的模式, 对于所有没有使用的外部悬空的 I/O 都需要设置为准双向口, 并固定输出高电平。特别是部分管脚的芯片, 由于芯片内部有部分 I/O 口并没有打线到外部管脚, 所以这些 I/O 也是处于悬空状态的, 这部分 I/O 也需要设置为准双向口, 并固定输出高电平。

附录T 触摸按键的 PCB 设计指导

触摸按键对 PCB 设计的要求比较严格，否则其效果会大打折扣甚至失败。建议用户在设计 PCB 时遵循以下几点原则：

1. 遵循通常的数模混合电路设计的基本原则。

电容式触摸按键模块集成了精密电容测量的模拟电路，因此进行 PCB 设计时应该把它看成一个独立的模拟电路对待。遵循通常的数模混合电路设计的基本原则。

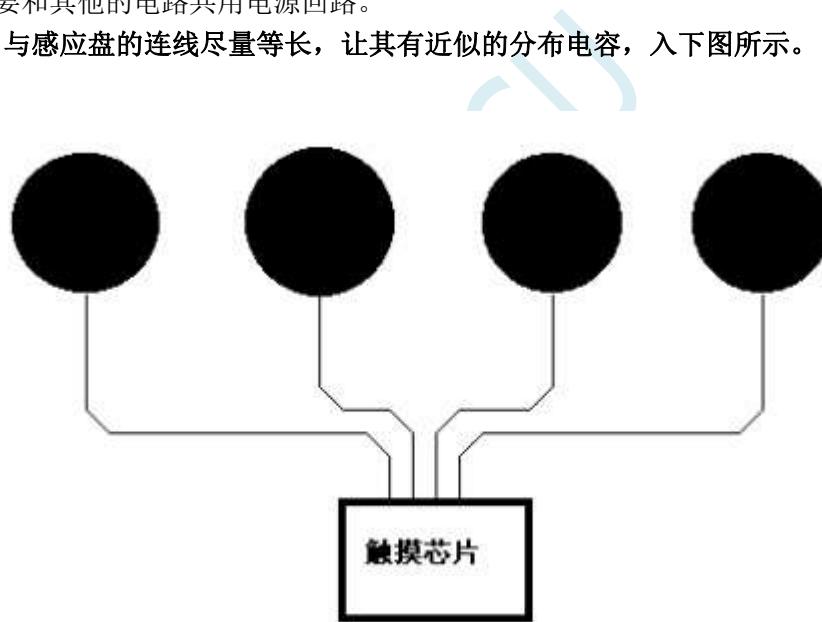
2. 采用星形接地

触摸芯片的地线不要和其他电路公用，应该单独连到板子电源输入的接地点，也就是通常说的采用“星形接地”。

3. 电源上产生的噪声对触摸芯片的影响

电源纹波、噪声应该尽量小，最好用一根独立的走线从板子的供电点取电并增加滤波措施，不要和其他的电路共用电源回路。

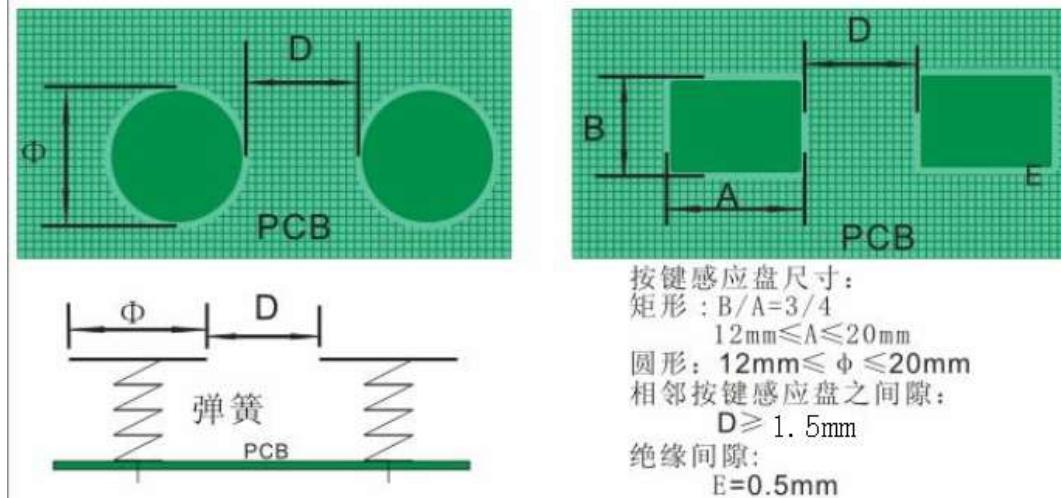
4. IC 与感应盘的连线尽量等长，让其有近似的分布电容，入下图所示。



5. 按键感应盘（电容传感器）大小和间隙

在满足面板的美学设计要求的情况下，必须通过合理安排的感应盘大小和间隔尺寸，来获得最佳的触摸感应效果。感应盘放在底层，IC 也放在底层，感应盘与 IC 连线不要有过孔。相邻感应盘边沿的间隔最好在 1.5mm 以上（下图中的尺寸 D），如果 PCB 面积允许，尽量取大一些间隔。铺铜与感应盘的间隔为 0.5mm（下图中的尺寸 E）。

在家用电器应用中, 以下推荐的感应盘大小和间距的尺寸可获得最佳触摸感应效果



6. 铺铜处理

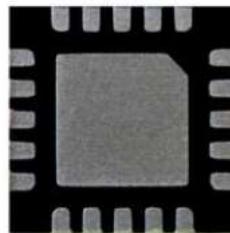
底层可以铺网格铜或实铜均可, 注意铺铜与感应盘的间隔为 0.5mm。顶层印刷按键的丝印信息, 丝印的外框形状与底层感应盘一致, 顶层对应底层感应盘的地方不能铺铜, 否则会屏蔽掉触摸动作。顶层铺铜与底层铺铜一样即可。

7. 走线处理

感应盘与 IC 的连线使用比较小的线宽为好, 比如 10~15mil 之间。感应盘到触摸芯片的连线不要跨越强干扰、高频、大电流的线。感应盘到触摸芯片的连线周围 1.5mm 内不要走其他信号线, 越远离越好。顶层对应底层感应盘和连接线的地方, 最好不要放任何线。

附录U QFN/DFN 封装元器件焊接方法

STC 产品的封装形式中，增加了现在比较流行的 QFN 和 DFN 的封装。由于这种封装形式的芯片芯片的管脚在芯片底部，手工焊接有一定的难度。市面上有专门做工程样品焊接的小公司，可承接工程样品打样。如用户需要自行焊接，可参考下面的焊接方法。



- 1、首先需要准备如下工具：电烙铁、热风枪、镊子、固定架等工具
- 2、需要焊接的 PCB 板和芯片如下图：



- 3、先给板上芯片的焊盘上锡：



- 4、然后给芯片底部上锡，这个上完锡后要弄平，尽量减少锡，但不能没有。



- 5、调整热风枪温度，实际出风大概在 240 度左右，因为风枪质量不一样，根据实际情况调节。



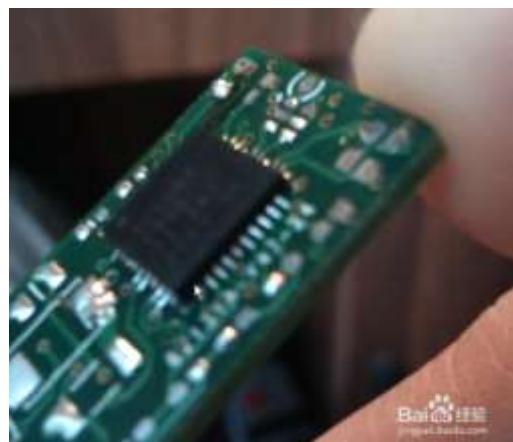
- 6、把芯片放到焊盘上，一定要放正，然后用热风枪对着它吹，速度要均匀，直到锡溶化，一般 20 秒内。



- 7、用烙铁给芯片侧引脚上锡



8、 焊接完成后的效果



附录V 关于回流焊前是否要烘烤

根据国际湿气敏感性等级 3 (MSL3) 规范的要求, 贴片元器件在拆开真空包装后, 168 小时内, 7 天内, 必须回流焊贴片完成, 如未完成, 必须再次高温烘烤。

SOP/TSSOP 塑料管耐不了 100 度以上的高温, 拆开真空包装后 7 天内必须回流焊贴片完成, 否则回流焊前去除耐不了 100 度以上高温的塑料管, 放到金属托盘中, 重新烘烤: 110~125°C, 4~8 个小时都可以

LQFP/QFN/DFN 托盘能耐 100 度以上的高温, 拆开真空包装后 7 天内必须回流焊贴片完成, 否则回流焊前必须重新烘烤: 110~125°C, 4~8 个小时都可以

附录W 如何使用万用表检测芯片 I/O 口好坏

根据国际湿气敏感性等级 3 (MSL3) 规范的要求, 贴片元器件在拆开真空包装后, 168 小时内, 7 天内, 必须回流焊贴片完成, 如未完成, 必须再次高温烘烤。如果没有高温烘烤的流程, 直接进行回流焊, 则可能由于芯片内外受热不均导致芯片内部金属线被拉断, 最终出现的现象是芯片 I/O 口损坏。

STC 的单片机在芯片设计时, 每个 I/O 口都有两个分别到 VCC 和 GND 的保护二极管, 用万用表的二极管监测档可以进行测量。可使用此方法简单判断 I/O 管脚的好坏情况。使用万用表测量方法如下 (注: 这里使用的是数字万用表)

首先将万用表调到二极管检测挡位, 被测芯片不要供电, 将万用表的**红表笔**连接到被测芯片的**GND 管脚**, **黑表笔**依次测量每个 I/O 口, 如果万用表显示的参数为 0.7V 左右, 则表示芯片的内部 I/O 到 GND 的保护二极管正常, 即打线也是完好的, 若显示的参数为 0V, 则表示芯片内部的打线已被拉断。

上面的方法是检测芯片内部的打线情况的方法。

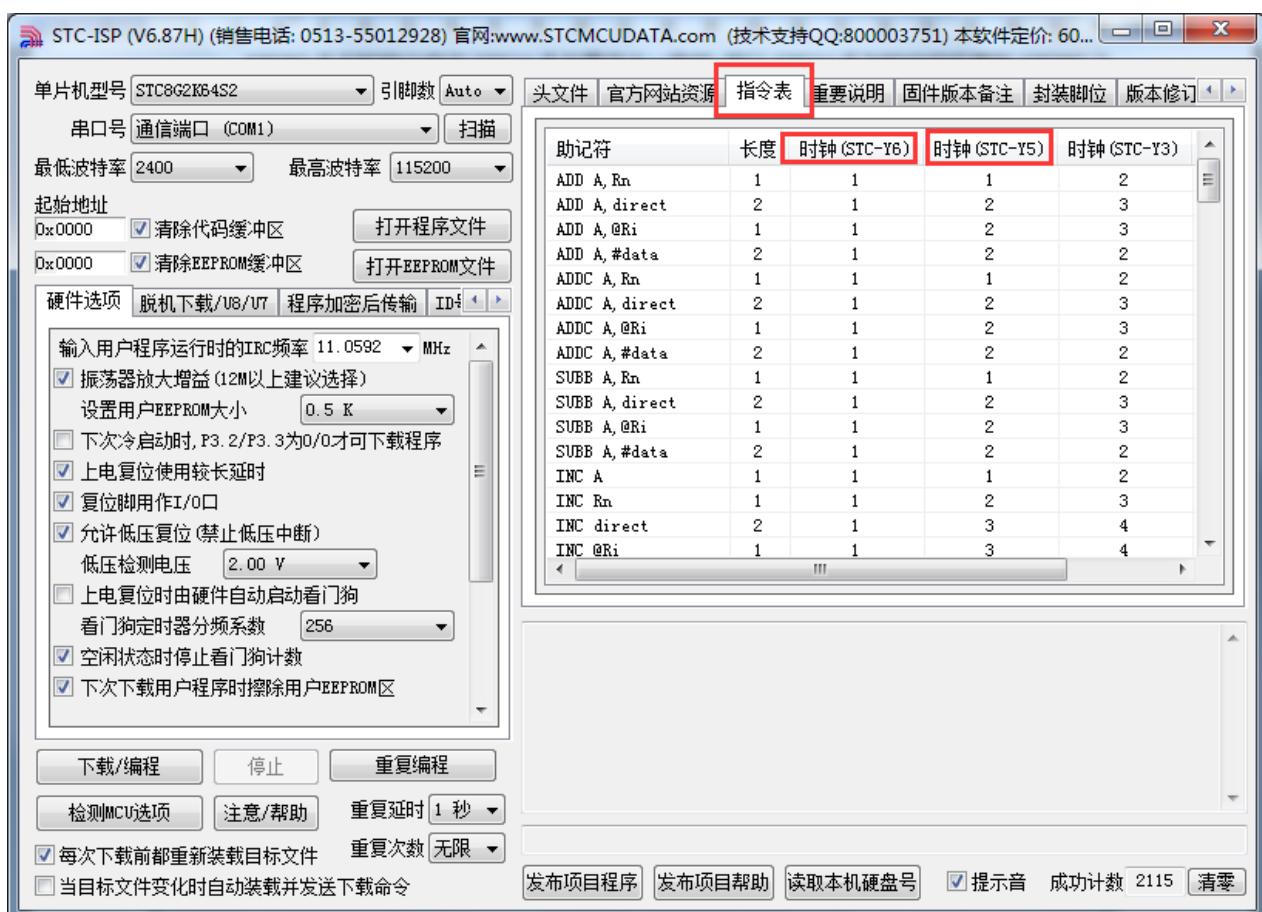
另外, 如果用户板上, 单片机的管脚没有加保护电路, 一旦出现过流或者过压都可能导致 I/O 烧坏。为检测管脚是否被烧坏, 除了使用上面的方法检测 I/O 口到 GND 的保护二极管外, 还需要检测 I/O 口到 VCC 的保护二极管。使用万用表检测 I/O 口到 VCC 的保护二极管的方法如下:

首先将万用表调到二极管检测挡位, 被测芯片不要供电, 将万用表的**黑表笔**连接到被测芯片的**VCC 管脚**, **红表笔**依次测量每个 I/O 口, 如果万用表显示的参数为 0.7V 左右, 则表示芯片的内部 I/O 到 VCC 的保护二极管正常, 若显示的参数为 0V, 则表示芯片此端口已被损坏。

附录X STC8G 系列单片机取代 STC15 系列的注意事项

■ 单片机指令

STC8G 系列的指令码与 STC15 系列是完全一致的，所以 STC15 系列的代码移植到 STC8G 上，运行依然正确，但 **STC8G 系列的指令速度比 STC15 系列要快**，STC15 系列的指令系统属于 STC-Y5 系列指令，而 STC8G 系列的指令属于 STC-Y6 系列指令，STC-Y6 系列的大部分指令执行都只需要一个 CPU 时钟。如果用户代码中有指令延时的代码，则需要进行调整。有个每条指令的对比可参考 STC 下载软件的指令表，如下图：



■ I/O 口

STC8G 系列单片机上电后，I/O 的模式与 STC15 系列不一样。STC15 系列单片机所有 I/O 口上电后都是 8051 的准双向口模式，而 **STC8G 系列单片机的 I/O 中，除了 ISP 下载脚 P3.0/P3.1 为准双向口模式外，其余的所有 I/O 口在上电后都是高阻输入模式**。传统的 8051 和 STC 的 15 系列单片机上电后即为准双向口模式并输出高电平，经常有客户的系统中使用 I/O 驱动马达或者 LED 灯，因此会出现单片机上电的瞬间马达会动一下或者 LED 会闪一下。STC8G 系列的 I/O 上电后为高阻输入模式，就可避免马达和 LED 的这种误动作。

由于 STC8G 系列单片机的 I/O 中，除了 ISP 下载脚 P3.0/P3.1 为准双向口模式外，其余的所有 I/O 口在上电后都是高阻输入模式，所以当用户需要 STC8G 系列的 I/O 口向外输出信号前，必须先使用 PxM0 和 PxM1 两个寄存器对 I/O 的工作模式进行设置。

■ 复位脚

STC8G 系列和 STC15 系列的 P5.4 口一般情况下是当作普通 I/O 口使用的，当用户在 ISP 下载时设置了 P5.4 为复位脚功能时，P5.4 口则为单片机的复位脚（RESET 脚）。对于 STC15H 系列，复位脚为高电平时单片机处于复位状态，低电平时单片机解除复位状态。而 STC8 系列与 STC15H 系列的复位电平是向反的，即对于 STC8G 系列，复位脚为低电平时，单片机处于复位状态，高电平时单片机解除复位状态。

所以当用户使能 P5.4 口的复位脚功能是需要注意复位电平的问题。

■ ADC

STC8G 系列和 STC15 系列的 ADC_CONTR、ADC_RES、ADC_RESL3 个寄存器地址相同的。但 STC8G 系列另外新增加了两个寄存器：[ADCCFG](#) 和 [ADCTIM](#)。

STC15 系列 [开始 ADC 转换位 ADC_START](#) 位于寄存器 ADC_CONTR 的 BIT3，而 STC8G 系列的位于 ADC_CONTR 的 BIT6

STC15 系列 [ADC 转换完成标志位 ADC_FLAG](#) 位于寄存器 ADC_CONTR 的 BIT4，而 STC8G 系列的位于 ADC_CONTR 的 BIT5

STC15 系列 [ADC 速度控制为 ADC_SPEED](#) 位于寄存器 ADC_CONTR 的 BIT6-BIT5，而 STC8G 系列的位于 ADCCFG 的 BIT3-BIT0

STC15 系列 [ADC 转换结果的对齐控制位 ADRJ](#) 位于寄存器 CLK_DIV 的 BIT5，而 STC8G 系列的对齐控制位 [RESFMT](#) 位于 ADCCFG 的 BIT5

STC8G 系列新增了更为精准的 ADC 转换时序控制机制，通过寄存器 [ADCTIM](#) 进行设置

■ EEPROM

STC15 系列的 EEPROM 擦除和编程的等待时间用寄存器 IAP_CONTR 的 Bit2-Bit0 设置，设置的只是一个大概的频率范围值，[STC8G 系列新增了一个寄存器 IAP_TPS \(SFR 地址: 0F5H\)](#)，专用于设置 EEPROM 擦除和编程的等待时间，且用户不需要去计算，只需要根据当前 CPU 的工作频率，直接填入 IAP_TPS 即可，硬件会自动计算等待时间。（比如：当前 CPU 的工作频率为 24MHz，则只需要向 IAP_TPS 填入 24 即可）

附录Y STC8G 系列单片机取代 STC8A/8F 系列的注意事项

■ I/O 口

STC8G 系列单片机上电后, I/O 的模式与 STC8A/8F 系列不一样。STC8A/8F 系列单片机所有 I/O 口上电后都是 8051 的准双向口模式, 而 STC8G 系列单片机的 I/O 中, 除了 ISP 下载脚 P3.0/P3.1 为准双向口模式外, 其余的所有 I/O 口在上电后都是高阻输入模式。传统的 8051 和 STC 的 15/8A/8F 系列单片机上电后即为准双向口模式并输出高电平, 经常有客户的系统中使用 I/O 驱动马达或者 LED 灯, 因此会出现单片机上电的瞬间马达会动一下或者 LED 会闪一下。STC8G 系列的 I/O 上电后为高阻输入模式, 就可避免马达和 LED 的这种误动作。

由于 STC8G 系列单片机的 I/O 中, 除了 ISP 下载脚 P3.0/P3.1 为准双向口模式外, 其余的所有 I/O 口在上电后都是高阻输入模式, 所以当用户需要 STC8G 系列的 I/O 口向外输出信号前, 必须先使用 PxM0 和 PxM1 两个寄存器对 I/O 的工作模式进行设置。

■ 复位脚

STC8G 系列和 STC8A/8F 系列的 P5.4 口一般情况下是当作普通 I/O 口使用的, 当用户在 ISP 下载时设置了 P5.4 为复位脚功能时, P5.4 口则为单片机的复位脚 (RESET 脚)。对于 STC8A/8F 系列, 复位脚为高电平时单片机处于复位状态, 低电平时单片机解除复位状态。而 STC8G 系列与 STC8A/8F 系列的复位电平是向反的, 即对于 STC8G 系列, 复位脚为低电平时, 单片机处于复位状态, 高电平时单片机解除复位状态。

所以当用户使能 P5.4 口的复位脚功能是需要注意复位电平的问题。

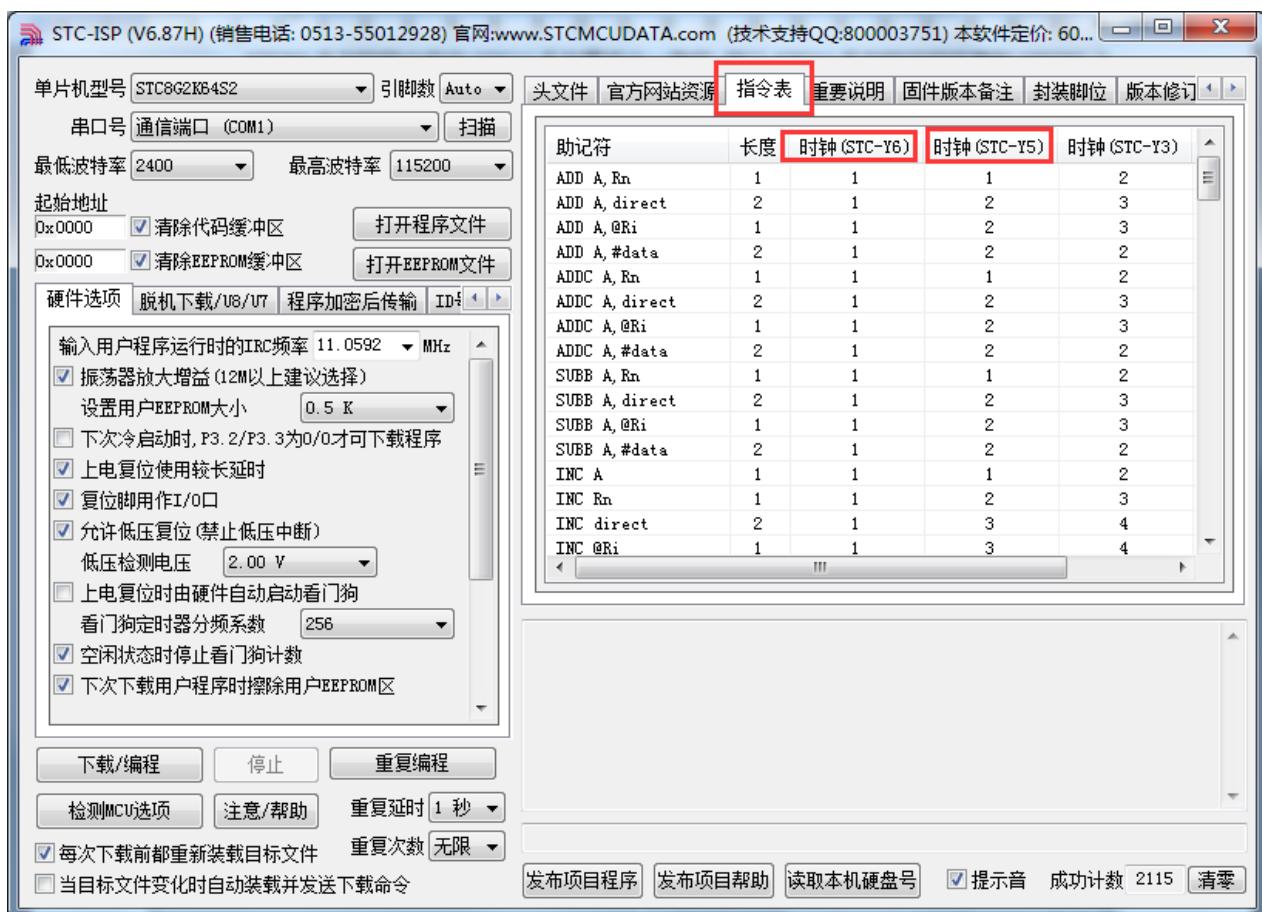
■ EEPROM

STC8A/8F 系列的 EEPROM 擦除和编程的等待时间用寄存器 IAP_CONTR 的 Bit2-Bit0 设置, 设置的只是一个大概的频率范围值, STC8G 系列新增了一个寄存器 IAP_TPS (SFR 地址: 0F5H), 专用于设置 EEPROM 擦除和编程的等待时间, 且用户不需要去计算, 只需要根据当前 CPU 的工作频率, 直接填入 IAP_TPS 即可, 硬件会自动计算等待时间。(比如: 当前 CPU 的工作频率为 24MHz, 则只需要向 IAP_TPS 填入 24 即可)

附录Z STC15H 系列单片机取代 STC15F/L/W 系列的注意事项

■ 单片机指令

STC15H 系列的指令码与 STC15F/L/W 系列是完全一致的，所以 STC15F/L/W 系列的代码移植到 STC15H 上，运行依然正确，但 **STC15H 系列的指令速度比 STC15F/L/W 系列要快**，STC15F/L/W 系列的指令系统属于 STC-Y5 系列指令，而 STC15H 系列的指令属于 STC-Y6 系列指令，STC-Y6 系列的大部分指令执行都只需要一个 CPU 时钟。如果用户代码中有指令延时的代码，则需要进行调整。有个每条指令的对比可参考 STC 下载软件的指令表，如下图：



■ I/O 口

STC15H 系列单片机上电后，I/O 的模式与 STC15F/L/W 系列不一样。STC15F/L/W 系列单片机所有 I/O 口上电后都是 8051 的准双向口模式，而 **STC15H 系列单片机的 I/O 中，除了 ISP 下载脚 P3.0/P3.1 为准双向口模式外，其余的所有 I/O 口在上电后都是高阻输入模式**。传统的 8051 和 STC 的 15 系列单片机上电后即为准双向口模式并输出高电平，经常有客户的系统中使用 I/O 驱动马达或者 LED 灯，因此会出现单片机上电的瞬间马达会动一下或者 LED 会闪一下。STC15H 系列的 I/O 上电后为高阻输入模式，就可避免马达和 LED 的这种误动作。

由于 STC15H 系列单片机的 I/O 中，除了 ISP 下载脚 P3.0/P3.1 为准双向口模式外，其余的所有 I/O 口在上电后都是高阻输入模式，所以当用户需要 STC15H 系列的 I/O 口向外输出信号前，必须先使用 PxM0 和 PxM1 两个寄存器对 I/O 的工作模式进行设置。

■ 复位脚

STC15H 系列和 STC15F/L/W 系列的 P5.4 口一般情况下是当作普通 I/O 口使用的，当用户在 ISP 下载时设置了 P5.4 为复位脚功能时，P5.4 口则为单片机的复位脚 (RESET 脚)。对于 STC15H 系列，复位脚为高电平时单片机处于复位状态，低电平时单片机解除复位状态。而 STC8 系列与 STC15H 系列的复位电平是向反的，即对于 STC15H 系列，复位脚为低电平时，单片机处于复位状态，高电平时单片机解除复位状态。

所以当用户使能 P5.4 口的复位脚功能是需要注意复位电平的问题。

■ ADC

STC15H 系列和 STC15F/L/W 系列的 ADC_CONTR、ADC_RES、ADC_RESL3 个寄存器地址相同的。但 STC15H 系列另外新增加了两个寄存器：[ADCCFG](#) 和 [ADCTIM](#)。

STC15F/L/W 系列 [开始 ADC 转换位 ADC_START](#) 位于寄存器 ADC_CONTR 的 BIT3，而 STC15H 系列的位于 ADC_CONTR 的 BIT6

STC15F/L/W 系列 [ADC 转换完成标志位 ADC_FLAG](#) 位于寄存器 ADC_CONTR 的 BIT4，而 STC15H 系列的位于 ADC_CONTR 的 BIT5

STC15F/L/W 系列 [ADC 速度控制为 ADC_SPEED](#) 位于寄存器 ADC_CONTR 的 BIT6-BIT5，而 STC15H 系列的位于 ADCCFG 的 BIT3-BIT0

STC15F/L/W 系列 [ADC 转换结果的对齐控制位 ADRJ](#) 位于寄存器 CLK_DIV 的 BIT5，而 STC15H 系列的对齐控制位 [RESFMT](#) 位于 ADCCFG 的 BIT5

STC15H 系列新增了更为精准的 ADC 转换时序控制机制，通过寄存器 [ADCTIM](#) 进行设置

■ EEPROM

STC15F/L/W 系列的 EEPROM 擦除和编程的等待时间用寄存器 IAP_CONTR 的 Bit2-Bit0 设置，设置的只是一个大概的频率范围值，[STC15H 系列新增了一个寄存器 IAP_TPS \(SFR 地址: 0F5H\)](#)，专用于设置 EEPROM 擦除和编程的等待时间，且用户不需要去计算，只需要根据当前 CPU 的工作频率，直接填入 IAP_TPS 即可，硬件会自动计算等待时间。(比如：当前 CPU 的工作频率为 24MHz，则只需要向 IAP_TPS 填入 24 即可)

■ 比较器

STC15W 系列的比较器正极为 P5.5，负极为 P5.4，STC15H 系列比较器正极为 P3.7，负极为 P3.6。

附录AA 更新记录

● 2023/1/17

1. 增加“用 STC 一箭双雕之 USB 转双串口仿真 STC8 系列 MCU”章节

● 2023/1/10

1. 增加论坛链接
2. 增强型 PWM 章节增加系统时钟分频输出范例
3. 更新参考线路图说明

● 2022/12/23

1. 更新选型和价格表
2. 更新 I2C 从机代码范例程序（提高代码兼容性）

● 2022/11/14

1. ISP 下载参考线路图中的电容统一建议使用 22uF+0.1uF (104) 的组合

● 2022/10/31

1. 增加使用 STC-USB Link1D 工具进行 ISP 下载的参考线路图
2. 增加软件模拟 USB 进行 ISP 下载的参考线路图

● 2022/9/20

1. 更新选型价格表
2. 更新官方网址
3. 增加 STC-ISP 高级应用章节
4. 增加关闭驱动程序强制数字签名说明章节

● 2022/9/6

1. 修正 I/O 口章节中的笔误

● 2022/8/4

1. 加 STC8G2K64S2 型号

● 2022/7/6

1. 修正比较器章节的错别字
2. I/O 口章节增加 I/O 口使用注意事项的相关说明
3. 存储器章节增加 EAXFR 寄存器使用说明小节
4. 更新选型价格表

● 2022/5/5

1. 在管脚图下面增加通过芯片丝印识别第一脚和辨别芯片版本的方法
2. 增加使用 USB 直接下载时 P3.2 口的说明

● 2022/4/22

3. 更新芯片型号以及供货信息。
4. 更新串口波特率计算说明
5. 附录中增加“如何使用万用表检测芯片 I/O 口好坏”章节

● 2022/3/9

6. 更新文档。

● 2022/2/28

1. 修改 LED 章节的寄存器地址描述错误部分。

● 2022/2/8

1. 建议用户代码中不要使用 pdata, 统一使用 xdata 在扩展 RAM 中声明变量。

● 2022/1/25

1. 更新文档中有关 TL431/CD431 的参考线路图

● 2022/1/20

1. 增加第三方 MCU 对 STC 芯片进行 ISP 下载时的电源控制参考电路图

● 2021/12/17

1. 将 EEPROM 应用范例程序进行重新排版
2. 增加附录“STC 仿真使用说明书”章节
3. 所有管脚图中复位脚名称修改为 NRST
4. 修改定时器 2/3/4 的定时计算公式
5. 更新选型和价格表

● 2021/11/16

1. 更正文档中的错别字
2. 增加 EEPROM 应用范例程序

● 2021/10/08

1. 更正文档中已发现的错别字
2. 更新选项价格表中有关 CCP/PCA/PWM 外部中断的描述
3. 在中断系统章节的中断源表格中增加外部中断说明
4. 附录中仿真器章节的使用新版本软件进行截图
5. 在附录中增加“如何测试 I/O 口”章节

● 2021/9/26

1. 在增强型 PWM 章节增加有关归零中断的使用注意事项

● 2021/8/26

1. 修正 ADC 章节范例程序中的注释错误
2. 增强型 PWM 章节增加范例程序
3. 在附录中增加“关于回流焊前烘烤的注意事项”章节

● 2021/7/19

1. 增加 STC15H 系列特殊型号
2. 增加 STC15H 系列取代 STC15F/L/W 系列的注意事项

● 2021/7/7

1. 增加 STC8G1K08A 系列的 DIP8 的管脚说明
2. 增加 STC8G1K08 系列的 DIP16 和 DIP20 的管脚说明
3. 增强型 PWM 章节增加“产生 3 路相位差 120 度的互补带死区的 PWM 波形”的范例程序
4. 增强型 PWM 章节增加“输出占空比为 100% 和 0% 的 PWM 波形”的范例程序

● 2021/6/26

1. 修正增强型 PWM 输出频率计算公式
2. 增加 STC8G2K48S4 型号
3. 增加 STC8G2K48S2 型号

● 2021/5/10

1. 增加 ADC 电源开关延时说明
2. 增加使用 ADC 第 15 通道反推外部通道输入电压原理说明及计算公式
3. 修改部分系列的最大可用 FLASH 大小的错误描述

4. 增加定时器 2/3/4 中断标志位的相关说明

● 2021/3/8

1. 使用 STC8G1K17 型号举例用户如何规划自己的 EEPROM
2. 附录中增加 STC8 系列命名花絮

● 2021/2/26

1. 增加有关模拟 USB 下载的说明
2. 增加定时器 2、定时器 3、定时器 4 的 8 位时钟预分频寄存器说明
3. 触摸按键芯片的触摸按键灵敏度调节电容说明（建议使用独石电容）

● 2021/2/4

1. 更正 CLKDIV 寄存器的复位初始值
2. 更新驱动共阴/共阳 LED 参考线路图
3. 更新 STC8G2K64S4-QFN32 的价格
4. 增加特殊功能寄存器初始值说明
5. 修改 I/O 口章节中 3V/5V 器件混连系统应用中的描述错误
6. 在管脚图下增加应用参考线路图

● 2020/12/4

1. 更正部分范例程序中的错误
2. 增加串口转 LIN 总线范例程序

● 2020/11/25

1. 更正部分范例程序中的错误
2. 更新中断结构图
3. 更新 STC8G2K64S4/STC8G2K64S2 系列的应用注意事项
4. 增加 PCA 模块使用 ECI 外部时钟模式的范例程序

● 2020/10/16

1. 更新 STC8G2K64S4 型号的 LQFP32 封装的价格
2. 更正 STC8G1K08 系列管脚说明中的错误
3. 增加外部晶振电路的负载电容说明

● 2020/9/23

1. 更新应用注意事项
2. 修改 PCA/CCP/PWM 章节中部分描述错误

● 2020/9/4

1. 修改文档中的部分错别字
2. 更新 STC8G2K64S4/S2 的应用注意事项
3. 定时器章节完善定时器 0/1/2/3/4 内部硬件框图
4. 更正定时器 0 的模式 3 停止说明（定时器 0 的模式 3 为不可屏蔽中断，一旦启动，软件将无法停止，必须芯片复位才能停止）
5. 整理文档章节顺序
6. 增加 STC8G2K64S4-LQFP32 管脚图
7. 典型应用线路图章节中增加“单片机电源控制参考电路”
8. 7.3 章“存储器中的特殊参数”的特殊参数列表中，增加 20M、27M、30M、33.1776M、35M 以及 36.864MHz 的 IRC 参数
9. 更新 7.3.7 章“用户自定义内部 IRC 频率”的范例代码

● 2020/8/26

1. 增加定时器计算公式章节
2. 增加串口波特率计算公式章节
3. 增加 15 位增强型 PWM 输出频率计算公式章节
4. 增加 ADC 相关计算公式章节
5. 增加 12 位 ADC 静态参数参考数据
6. 增加 MDU16 运算所需时钟数参数
7. 增加 EEPROM 操作所需时间参数
8. 将所有章节中特殊功能寄存器都单独列为目录小节，方便查找
9. 增加 STC8G 系列单片机取代 STC8A/8F 系列的注意事项

● 2020/8/21

1. 修改文档中部分描述有误的地方
2. 增加 15 位增强型 PWM 的特性描述

● 2020/8/10

1. 增加看门狗定时器章节
2. 整理掉电唤醒定时器章节
3. 更新应用注意事项
4. 增加有关 STC 下载工具使用说明附录章节

● 2020/8/6

1. 对工作温度进行说明
2. 增加使用通用 USB 转串口工具下载的应用线路图
3. 更新应用注意事项

● 2020/7/16

1. 增加 BUS_SPEED 寄存器的说明
2. 增加 QFN/DFN 封装形式芯片的焊接指导
3. 增加 EEPROM 编程说明
4. 在下载应用线路图章节中增加设置 U8W/U8-Mini 为直通模式的方法

● 2020/7/3

1. 增加附录章节, “不停电用户程序复位到系统区进行 ISP 下载的方法”
2. 增加附录章节, “使用 STC 的 IAP 系列单片机开发自己的 ISP 程序”
3. 增加附录章节, “STC8G 系列单片机取代 STC15 系列的注意事项”
4. 增加附录章节, “官方网址说明”
5. ADC 章节中, 增加 ADC 转换时序图
6. 删减 STC8G2K64S2 系列中 P0/P1/P3/P4/P5 口上的增强型 PWM

● 2020/6/15

1. 增加 ADC_VRef+管脚说明
2. 增加使用 USB 转串口参考电路中的二极管和电阻的说明
3. 修正 I/O 口驱动电流控制寄存器 PxDR 的描述 (1: 普通驱动电流; 0: 强驱动电流)
4. 增加 I2C 从机设备地址的说明

● 2020/6/8

1. 增加 ADC 最快转换速度的说明
2. 详细说明 I²C 总线速度设置
3. 更新模拟 USB 模式 ISP 下载参考线路图
4. 在封装图中增加 DFN8、QFN20、QFN32、QFN48、QFN64 衬底说明
5. 增加 PCA 输出 10 位 PWM 的范例程序
6. 增加比较器多路复用 (比较器+ADC 输入) 应用的范例程序

● 2020/5/29

1. ADC 章节增加加法电路应用
2. 增加寄存器 EAXFR 的说明
3. 增加使用第三方应用程序调用发布项目程序的方法

● 2020/5/25

1. ADC 章节增加负压检测电路
2. 修正部分图片中的乱码

● 2020/5/20

1. 更新电气特性中的使能低压检测唤醒功能时, 时钟停振模式的功耗参数
2. 更新电气特性中的使能比较器掉电唤醒功能时, 时钟停振模式的功耗参数
3. ADC 范例程序中增加设置 ADCTIM 寄存器用于控制 ADC 内部时序

4. 更正文档中的部分笔误
5. 在每个单片机系列的特性中增加可用于时钟停振模式唤醒的中断
6. PCA 章节中增加 PWM 频率计算公式以及输出全高电平和全低电平的方法
7. 在 ISP 下载应用线路图中增加 ISP 下载步骤指导
8. 增加掉电唤醒定时器唤醒省电模式范例程序

● 2020/5/14

1. 增加比较器多路复用说明
2. 增加 PWM 触发 ADC 范例程序
3. ADC 章节中增加 ADC 工作时钟频率描述
4. ADC 章节中增加 ADC 参考线路图
5. 更新电气特性中的低压检测、比较器等的功耗参数
6. 增加上电复位和按键复位参考电路图

● 2020/4/29

1. 将串口下载参考电路图中, MCU 的 TxD 脚上串联的电阻由 300 欧姆改为 100 欧姆
2. 修正使用 PL2303GL 进行 ISP 下载的参考电路图中供电部分的错误

● 2020/4/26

1. 修正 DFN8 (3mm*3mm) 的封装尺寸图
2. 更新电气特性中 I/O 速度参数
3. 更新电气特性中比较器速度参数
4. 增加 LED 驱动的范例程序
5. 增加 STC8G2K64S4 系列和 STC8G2K64S2 系列的 PDIP40 的参考管脚图
6. 更正 13.6 章节串口注意事项中设置 TI 和 RI 的时间点
7. 增加比较器章节中有关模拟滤波和数字滤波的说明
8. 修正附录 E 中, MAX232 与 RS485 连线错误

● 2020/4/8

1. 增加 STC8G1K08T 系列芯片的应用注意事项
2. 修正“存储器中的特殊参数”章节中关于 STC8G2K64S4 型号的参数
3. 更正“利用 ADC 第 15 通道测量外部电压或电池电压”章节中计算电压的公式
4. 修正 STC8G1K08T 系列有关 LED 驱动的特殊功能寄存器
5. 增加掉电唤醒定时器寄存器的使用说明
6. 更新 I/O 口章节中有关整体驱动电流的内容

● 2020/3/26

1. IRC24MCR 寄存器更名为 HIRCCR
2. 增加 STC8G2K64S4-LQFP48 型号使用 PL2303GL 下载参考线路图
3. 增加 STC8G2K64S4-LQFP48 型号直接软模拟 USB 下载参考线路图
4. 更新直流特性中的芯片在不同工作频率下的功耗

- 5. 更新应用注意事项
- 6. 在增强型 PWM 章节开头增加说明部分
- 7. 增加“触摸按键 PCB 设计指导”附录章节

● 2020/3/6

- 1. 重新整理管脚图章节结构
- 2. 更正 STC8G1K17、STC8G1K17A 型号的芯片特性

● 2020/3/5

- 1. 修正触摸按键中断向量入口地址
- 2. 修正触摸按键中断使能位及中断优先级的特殊功能寄存器位

● 2020/3/4

- 1. 更正文档中有关内部 1.19V 参考信号源描述错误的部分
- 2. 更正 PWM 章节中部分示例代码的错误
- 3. 增加一般精度 ADC 和高精度 ADC 的应用线路图
- 4. 增加 ADC 模块的静态参数
- 5. 增加 STC8G1K08T 触摸按键系列
- 6. 增加触摸按键控制器说明章节
- 7. 增加触摸按键参考电路图
- 8. 增加 LED 驱动器说明章节

● 2020/2/26

- 1. 增加 DFN8 封装图

● 2020/2/24

- 1. 增加“关于中断号大于 31 在 Keil 中编译出错的处理”章节
- 2. 增加“在 Keil 中建立多文件项目的方法”章节
- 3. 增加“仿真时查看全部的寄存器的方法”章节
- 4. 增加 STC8G2K64S4 系列 A 版芯片的应用注意事项
- 5. 增加“使用 I2C 的数据线 SDA 唤醒 MCU 省电模式”范例
- 6. 增加“如何让传统的 8051 单片机学习板可仿真”章节
- 7. 更新“用 STC 系列 MCU 的 I/O 口直接驱动段码 LCD”章节内容
- 8. 更新“STC8G1K08 系列特性及价格”中的型号与价格
- 9. 增加“新增 4.1K 上拉电阻”章节
- 10. 在“存储器”章节中增加“8051 中可位寻址的数据存储器”小节

● 2020/1/20

- 1. 增加“利用 PCA 模块实现 DAC”参考线路图
- 2. 增加“一种典型三极管控制”电路

3. 增加“典型发光二极管控制”电路
4. 增加“混合电压供电系统 3V/5V 器件 I/O 口互连”参考电路
5. 增加“如何让 I/O 口上电复位时为低电平”参考电路
6. 增加“利用 74HC595 驱动 8 个数码管(串行扩展,3 根线)”参考电路
7. 增加“I/O 口直接驱动 LED 数码管”参考电路
8. 增加“运行用户程序时收到用户命令后自动启动 ISP 下载”说明

● 2020/1/17

1. 增加 MDU16 运算时钟数说明
2. 修正重要说明里面的描述

● 2020/1/15

1. 增加“ADC 做电容感应触摸按键”章节
2. 增加“ADC 作按键扫描应用线路图”章节
3. 增加附录“RS485 自动控制或 I/O 口控制线路图”
4. 增加附录“U8W 下载工具中 RS485 部分线路图”

● 2019/12/30

1. 更新芯片选型以及价格表
2. 增加产生用户指定频率的示例代码

● 2019/12/24

1. 将比较器章节参考电路图中的主控芯片修改为 STC8G 系列单片机
2. 将 ADC 章节参考电路图中的主控芯片修改为 STC8G 系列单片机
3. 将 ADC 模块中用于测量内部电压的通道统一命名为第 15 通道
4. 将原 STC8G1K08 系列改名为 STC8G1K08 系列
5. 将原 STC8G2K64S4 系列改名为 STC8G2K64S4 系列
6. 增加 STC8G1K08-8Pin 系列
7. 增加 STC8G1K08A 系列
8. 增加 STC8G2K64S2 系列
9. 修正 LVR 电压以及电气特性相关参数

● 2019/11/27

1. 增加 STC8G2K64S4 系列
2. 增加 STC8G1K08 系列重要说明

● 2019/11/11

1. 增加产品授权书
2. 统一文档中有关 Flash 程序存储器和 ROM 的称谓
3. 增加 USB 下载步骤演示（附录 C）

● 2019/10/22

1. 增加 QFN20 管脚图
2. 增加 QFN20 封装尺寸图
3. 更新范例程序

● 2019/10/15

1. 修正 LVR 的四级电压
2. 修正内部高精度 IRC 的温漂范围
3. 修正内部 1.19V 参考信号源
4. 更新直流特性表格数据

● 2019/10/09

1. 移除电源控制寄存器 (VOCTRL) 部分, STC8G 系列无此功能
2. 修改 LVR 的四级电压
3. 修正 IRC 的两个频率范围

● 2019/8/13

1. 创建 STC8G 系列单片机技术参考手册文档

附录BB STC8 系列命名花絮

STC8A: 字母 “A” 代表 ADC，是 STC 12 位 ADC 的起航产品

STC8F: 无 ADC、PWM 和 PCA 功能，现 STC8F 的改版芯片与原始的 STC8F 管脚完全兼容，但内部设计进行了优化和更新，用户需要修改程序，所以命名为 STC8C

STC8C: 字母 “C” 代表改版，是 STC8F 的改版芯片

STC8G: 字母 “G” 最初是芯片生产时打错字了，后来将错就错，定义 G 系列为 “GOOD” 系列，STC8G 系列简单易学

STC8H: 字母 “H” 取自 “高”的英文单词 “High” 的首字母，“高” 表示 “16 位高级 PWM”

本系列产品标准销售合同

一. 产品质量标准：货物为全新正品。符合 ROHS 质量标准。

二. 供方责任：如是供方质量问题，经双方确认后，需方退回芯片，有一换一，质保一年。

三. 需方责任：

A、验收：在快递送货到时，需方确认数量无误，无芯片散落，无管脚变形，无其他品质异常情况后再签收。如有异常需方不能签收，由快递公司承担责任。一经需方签收，需方就是认可供方已按要求完成该订单，不再有其他连带责任。

B、保管及贴片加工：根据国际湿敏度 3 (MSL3) 规范的要求，贴片元器件在拆开真空包装后，168 小时内，7 天内，必须回流焊贴片完成。LQFP/QFN/DFN 托盘能耐 100 度以上的高温，拆开真空包装后 7 天内必须回流焊贴片完成，如未完成，回流焊前必须重新烘烤：110~125°C，4~8 个小时都可以 SOP/TSSOP 塑料管耐不了 100 度以上的高温，拆开真空包装后 7 天内必须回流焊贴片完成，否则回流焊前先去除耐不了 100 度以上高温的塑料管，放到金属托盘中，重新烘烤：110~125°C，4~8 个小时都可以

由于经常有客户退回来的货物中含有来历不明产品，且贴片原器件拆开真空包装后，需要在 168 小时/7 天内完成回流焊贴片工序。

我司无产能对退回器件再进行重新详细检测，再进行重新烘烤，无能力对客户退回的所谓未拆封芯片进行评估，为保证全体客户的利益，产品一经出库，概不退换，以确保品质，确保所有客户的安全。

四. 解决纠纷方式：对本合同不详尽之处或产生争议，双方协商解决。协商不成在供方所在地申请仲裁。

五. 其他条款：合同一式两份。自双方签署起生效。供方若因外力因素而导致无法交货，供方应及时通知需方，并重新协商本合同相关事宜，需方免除供方应承担的义务。本合同未能列入条款可在合同附件详细列入。

六. 本合同双方代表签字且款到后方可生效。

备注：如特殊情况，买方买的型号要更换成其他型号，供方也同意的：

1，开机 13 小时高温烘烤， 1000 元一次

2，开机测试 RMB500 一次， +0.2 元/片

产品授权书

致：江苏国芯科技有限公司

STC8G 系列和 STC8H 系列产品的知识产权归深圳国芯人工智能有限公司所有。现授权江苏国芯科技有限公司可从事 STC8G 系列和 STC8H 系列产品在中国的推广和销售工作。

授权单位：深圳国芯人工智能有限公司

授权时限：2019 年 10 月 24 日 - 2024 年 12 月 31 日



自主产权，生产可控

深圳国芯人工智能有限公司是中华人民共和国大陆独资企业，按中国法律法规独立运营的企业，注册地址在深圳市前海深港合作区前湾一路1号A栋201室。

本手册所描述的器件是在中国境内自主研发，具备独立自主知识产权。

产品核心研发在中国境内，具备芯片设计、封装设计、结构设计、可靠性设计、器件仿真、工艺模拟等全部设计能力；产品核心研发团队人员及带头人全部为我国境内人员组成，其中研发团队带头人研发从业年限十年以上，具备长期、稳定的后续支持能力，具有在我国境内申请的专利证书及软件著作权等。

晶圆制造：本器件设计完成后的晶圆制造加工，在中华人民共和国大陆境内的晶圆厂加工制造完成，受中华人民共和国法律法规管理监管和控制，完全可控。

封装制造：本器件设计完成后的封装制造，在中华人民共和国大陆境内的封装厂加工完成，受中华人民共和国法律法规管理监管和控制，完全可控。

测试：本器件设计完成后的测试，在中华人民共和国大陆境内测试完成，受中华人民共和国法律法规管理监管和控制，完全可控。

本器件全部关键工艺均在我国自有生产线上完成，可以长期供货，无被断供的困扰。

特此说明。

