STC12H 系列单片机
技术参考手册

STC MCU

技术支持网站：www.STCAI.com
官方技术论坛：www.STCAIMCU.com
资料更新日期：**2023/1/10**

目录

1 概述 1
2 特性、价格及管脚 2
2.1 STC12H1K08 系列 2
2.1.1 特性及价格 2
2.1.2 管脚图 5
2.1.3 管脚说明 7
3 功能脚切换 11
3.1 功能脚切换相关寄存器 11
3.1.1 外设端口切换控制寄存器 1（P_SW1），串口 1、SPI 切换 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 12
3.1.3 时钟选择寄存器（MCLKOCR） 12
3.1.4 高级 PWM 选择寄存器（PWMx_PS） 13
3.1.5 高级 PWM 功能脚选择寄存器（PWMx_ETRPS） 15
3.2 范例程序 16
3.2.1 串口 1 切换 16
3.2.2 串口 2 切换 17
3.2.3 SPI 切换 19
3.2.4 I2C 切换 20
3.2.5 主时钟输出切换 22
4 封装尺寸图 25
4.1 TSSOP20 封装尺寸图 25
4.2 QFN20 封装尺寸图（3mm*3mm） 26
4.3 LQFP32 封装尺寸图（9mm*9mm） 27
4.4 QFN32 封装尺寸图（4mm*4mm） 28
4.5 STC12H 系列单片机命名规则 29
5 ISP 下载及典型应用线路图 30
5.1 STC12H 系列 ISP 下载应用线路图 30
5.1.1 使用 STC-USB Link1D 工具下载，支持在线和脱机下载 30
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） 32
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） 34
5.1.4 使用一箭双雕之 USB 转串口工具下载 36
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 38
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 39
5.1.7 使用 RS-232 转换器下载，也可支持仿真 41
5.1.8 使用 PL2303-GL 下载，也可支持仿真 42
5.1.9 单片机电源控制参考电路 43
5.2 STC-ISP 下载软件高级应用 44
5.2.1 发布项目程序 44
5.2.2 程序加密后传输（防烧录时串口分析出程序） 48
5.2.3 发布项目程序+程序加密后传输结合使用 52

STCMCU

5.2.4 用户自定义下载（实现不停电下载） 53
5.3 驱动程序数字签名的相关说明 57
5.3.1 关于驱动程序强制数字签名 57
5.3.2 Windows 10 关闭驱动程序强制数字签名步骤 58
6 时钟、复位与电源管理 63
6.1 系统时钟控制 63
6.1.1 系统时钟选择寄存器（CLKSEL） 63
6.1.2 时钟分频寄存器（CLKDIV） 64
6.1.3 内部高速高精度 IRC 控制寄存器（HIRCCR） 64
6.1.4 外部振荡器控制寄存器（XOSCCR） 64
6.1.5 内部 32KHz 低速 IRC 控制寄存器（IRC32KCR） 65
6.1.6 主时钟输出控制寄存器（MCLKOCR） 65
6.2 STC12H 系列内部 IRC 频率调整 66
6.2.1 IRC 频段选择寄存器（IRCBAND） 66
6.2.2 内部 IRC 频率调整寄存器（IRTRIM） 66
6.2.3 内部 IRC 频率微调寄存器（LIRTRIM） 67
6.2.4 时钟分频寄存器（CLKDIV） 67
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 68
6.3 系统复位 71
6.3.1 看门狗复位（WDT_CONTR） 72
6.3.2 软件复位（IAP_CONTR） 74
6.3.3 低压复位（RSTCFG） 75
6.3.4 低电平上电复位参考电路（一般不需要） 76
6.3.5 低电平按键复位参考电路 76
6.3.6 传统 8051 高电平上电复位参考电路 77
6.4 外部晶振及外部时钟电路 78
6.4.1 外部晶振输入电路 78
6.4.2 外部时钟输入电路（P5.6 不可用作普通 I/O） 78
6.5 时钟停振/省电模式与系统电源管理 78
6.5.1 电源控制寄存器（PCON） 78
6.6 掉电唤醒定时器 80
6.6.1 掉电唤醒定时器计数寄存器（WKTCL，WKTCH） 80
6.7 范例程序 81
6.7.1 选择系统时钟源 81
6.7.2 主时钟分频输出 83
6.7.3 看门狗定时器应用 85
6.7.4 软复位实现自定义下载 87
6.7.5 低压检测 89
6.7.6 省电模式 91
6.7.7 使用 INT0/INT1/INT2/INT3/INT4 管脚中断唤醒省电模式 93
6.7.8 使用 T0/T1/T2/T3/T4 管脚中断唤醒省电模式 96
6.7.9 使用 RxD/RxD2 管脚中断唤醒省电模式 100
6.7.10 使用 I2C 的 SDA 脚唤醒 MCU 省电模式 103
6.7.11 使用掉电唤醒定时器唤醒省电模式 106

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 108
6.7.13 比较器中断唤醒省电模式，建议配合使用掉电唤醒定时器 111
6.7.14 使用 LVD 功能检测工作电压（电池电压） 113
7 存储器 118
7.1 程序存储器 118
7.2 数据存储器 119
7.2.1 内部 RAM 119
7.2.2 程序状态寄存器（PSW） 120
7.2.3 内部扩展 RAM，XRAM，XDATA 121
7.2.4 辅助寄存器（AUXR） 121
7.2.5 外部扩展 RAM，XRAM，XDATA 122
7.2.6 总线速度控制寄存器（BUS_SPEED） 122
7.2.7 8051 中可位寻址的数据存储器 123
7.3 存储器中的特殊参数，在 ISP 下载时可烧录进程序 FLASH 125
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) 126
~~7.3.2 读取内部 1.19V 参考信号源值 (从 RAM 中读取)~~ 129
7.3.3 读取全球唯一 ID 号 (从 Flash 程序存储器（ROM）中读取) 132
~~7.3.4 读取全球唯一 ID 号 (从 RAM 中读取)~~ 135
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) 138
~~7.3.6 读取 32K 掉电唤醒定时器的频率 (从 RAM 中读取)~~ 141
7.3.7 用户自定义内部 IRC 频率 (从 Flash 程序存储器（ROM）中读取) 145
~~7.3.8 用户自定义内部 IRC 频率 (从 RAM 中读取)~~ 150
8 特殊功能寄存器 153
8.1 STC12H1K08 系列 153
8.2 特殊功能寄存器列表 155
9 I/O 口 162
9.1 I/O 口相关寄存器 162
9.1.1 端口数据寄存器（Px） 164
9.1.2 端口模式配置寄存器（PxM0，PxM1） 164
9.1.3 端口上拉电阻控制寄存器（PxPU） 164
9.1.4 端口施密特触发控制寄存器（PxNCS） 165
9.1.5 端口电平转换速度控制寄存器（PxSR） 165
9.1.6 端口驱动电流控制寄存器（PxDR） 165
9.1.7 端口数字信号输入使能控制寄存器（PxIE） 166
9.2 配置 I/O 口 167
9.3 I/O 的结构图 168
9.3.1 准双向口（弱上拉） 168
9.3.2 推挽输出 168
9.3.3 高阻输入 169
9.3.4 开漏输出 169
9.3.5 新增 4.1K 上拉电阻 170
9.3.6 如何设置 I/O 口对外输出速度 170
9.3.7 如何设置 I/O 口电流驱动能力 171
9.3.8 如何降低 I/O 口对外辐射 171

9.4 范例程序 172
9.4.1 端口模式设置 172
9.4.2 双向口读写操作 173
9.5 一种典型三极管控制电路 176
9.6 典型发光二极管控制电路 176
9.7 混合电压供电系统 3V/5V 器件 I/O 口互连 177
9.8 如何让 I/O 口上电复位时为低电平 178
9.9 利用 74HC595 驱动 8 个数码管(串行扩展,3 根线)的线路图 179
9.10 I/O 口直接驱动 LED 数码管应用线路图 180
9.11 用 STC 系列 MCU 的 I/O 口直接驱动段码 LCD 181
10 指令系统 200
11 中断系统 204
11.1 STC12H 系列中断源 204
11.2 STC12H 中断结构图 206
11.3 STC12H 系列中断列表 207
11.4 中断相关寄存器 209
11.4.1 中断使能寄存器（中断允许位） 210
11.4.2 中断请求寄存器（中断标志位） 213
11.4.3 中断优先级寄存器 216
11.5 范例程序 219
11.5.1 INT0 中断（上升沿和下降沿），可同时支持上升沿和下降沿 219
11.5.2 INT0 中断（下降沿） 221
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 222
11.5.4 INT1 中断（下降沿） 224
11.5.5 INT2 中断（下降沿），只支持下降沿中断 226
11.5.6 INT3 中断（下降沿），只支持下降沿中断 228
11.5.7 INT4 中断（下降沿），只支持下降沿中断 230
11.5.8 定时器 0 中断 232
11.5.9 定时器 1 中断 233
11.5.10 定时器 2 中断 235
11.5.11 定时器 3 中断 237
11.5.12 定时器 4 中断 240
11.5.13 UART1 中断 242
11.5.14 UART2 中断 244
11.5.15 ADC 中断 247
11.5.16 LVD 中断 249
11.5.17 比较器中断 251
11.5.18 SPI 中断 253
11.5.19 I2C 中断 255
12 I/O 口中断 259
12.1 I/O 口中断相关寄存器 259
12.2 范例程序 261
12.2.1 P0 口下降沿中断 261
12.2.2 P1 口上升沿中断 265

12.2.3 P2 口低电平中断 268
12.2.4 P3 口高电平中断 272
13 定时器/计数器 277
13.1 定时器的相关寄存器 277
13.2 定时器 0/1 279
13.2.1 定时器 0/1 控制寄存器（TCON） 279
13.2.2 定时器 0/1 模式寄存器（TMOD） 279
13.2.3 定时器 0 模式 0（16 位自动重装载模式） 280
13.2.4 定时器 0 模式 1（16 位不可重装载模式） 281
13.2.5 定时器 0 模式 2（8 位自动重装载模式） 282
13.2.6 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） 282
13.2.7 定时器 1 模式 0（16 位自动重装载模式） 283
13.2.8 定时器 1 模式 1（16 位不可重装载模式） 284
13.2.9 定时器 1 模式 2（8 位自动重装载模式） 285
13.2.10 定时器 0 计数寄存器（TL0，TH0） 285
13.2.11 定时器 1 计数寄存器（TL1，TH1） 285
13.2.12 辅助寄存器 1（AUXR） 286
13.2.13 中断与时钟输出控制寄存器（INTCLKO） 286
13.2.14 定时器 0 计算公式 286
13.2.15 定时器 1 计算公式 288
13.3 定时器 2（24 位定时器，8 位预分频+16 位定时） 289
13.3.1 辅助寄存器 1（AUXR） 289
13.3.2 中断与时钟输出控制寄存器（INTCLKO） 289
13.3.3 定时器 2 计数寄存器（T2L，T2H） 289
13.3.4 定时器 2 的 8 位预分频寄存器（TM2PS） 289
13.3.5 定时器 2 工作模式 290
13.3.6 定时器 2 计算公式 290
13.4 定时器 3/4（24 位定时器，8 位预分频+16 位定时） 291
13.4.1 定时器 4/3 控制寄存器（T4T3M） 291
13.4.2 定时器 3 计数寄存器（T3L，T3H） 291
13.4.3 定时器 4 计数寄存器（T4L，T4H） 292
13.4.4 定时器 3 的 8 位预分频寄存器（TM3PS） 292
13.4.5 定时器 4 的 8 位预分频寄存器（TM4PS） 292
13.4.6 定时器 3 工作模式 292
13.4.7 定时器 4 工作模式 294
13.4.8 定时器 3 计算公式 294
13.4.9 定时器 4 计算公式 295
13.5 范例程序 296
13.5.1 定时器 0（模式 0－16 位自动重载），用作定时 296
13.5.2 定时器 0（模式 1－16 位不自动重载），用作定时 297
13.5.3 定时器 0（模式 2－8 位自动重载），用作定时 299
13.5.4 定时器 0（模式 3－16 位自动重载不可屏蔽中断），用作定时 301
13.5.5 定时器 0（外部计数－扩展 T0 为外部下降沿中断） 303
13.5.6 定时器 0（测量脉宽－INT0 高电平宽度） 305

13.5.7 定时器 0（模式 0），时钟分频输出 307
13.5.8 定时器 1（模式 0－16 位自动重载），用作定时 309
13.5.9 定时器 1（模式 1－16 位不自动重载），用作定时 311
13.5.10 定时器 1（模式 2－8 位自动重载），用作定时 312
13.5.11 定时器 1（外部计数－扩展 T1 为外部下降沿中断） 314
13.5.12 定时器 1（测量脉宽－INT1 高电平宽度） 316
13.5.13 定时器 1（模式 0），时钟分频输出 318
13.5.14 定时器 1（模式 0）做串口 1 波特率发生器 320
13.5.15 定时器 1（模式 2）做串口 1 波特率发生器 324
13.5.16 定时器 2（16 位自动重载），用作定时 328
13.5.17 定时器 2（外部计数－扩展 T2 为外部下降沿中断） 330
13.5.18 定时器 2，时钟分频输出 332
13.5.19 定时器 2 做串口 1 波特率发生器 334
13.5.20 定时器 2 做串口 2 波特率发生器 337
13.5.21 定时器 2 做串口 3 波特率发生器 341
13.5.22 定时器 2 做串口 4 波特率发生器 345
13.5.23 定时器 3（16 位自动重载），用作定时 349
13.5.24 定时器 3（外部计数－扩展 T3 为外部下降沿中断） 352
13.5.25 定时器 3，时钟分频输出 354
13.5.26 定时器 3 做串口 3 波特率发生器 356
13.5.27 定时器 4（16 位自动重载），用作定时 360
13.5.28 定时器 4（外部计数－扩展 T4 为外部下降沿中断） 363
13.5.29 定时器 4，时钟分频输出 365
13.5.30 定时器 4 做串口 4 波特率发生器 367
14 串口通信 372
14.1 串口相关寄存器 372
14.2 串口 1 373
14.2.1 串口 1 控制寄存器（SCON） 373
14.2.2 串口 1 数据寄存器（SBUF） 373
14.2.3 电源管理寄存器（PCON） 374
14.2.4 辅助寄存器 1（AUXR） 374
14.2.5 串口 1 模式 0，模式 0 波特率计算公式 374
14.2.6 串口 1 模式 1，模式 1 波特率计算公式 375
14.2.7 串口 1 模式 2，模式 2 波特率计算公式 378
14.2.8 串口 1 模式 3，模式 3 波特率计算公式 378
14.2.9 自动地址识别 379
14.2.10 串口 1 从机地址控制寄存器（SADDR，SADEN） 379
14.3 串口 2 381
14.3.1 串口 2 控制寄存器（S2CON） 381
14.3.2 串口 2 数据寄存器（S2BUF） 381
14.3.3 串口 2 模式 0，模式 0 波特率计算公式 381
14.3.4 串口 2 模式 1，模式 1 波特率计算公式 382
14.4 串口注意事项 384
14.5 范例程序 385

14.5.1 串口 1 使用定时器 2 做波特率发生器 385
14.5.2 串口 1 使用定时器 1（模式 0）做波特率发生器 388
14.5.3 串口 1 使用定时器 1（模式 2）做波特率发生器 392
14.5.4 串口 2 使用定时器 2 做波特率发生器 396
14.5.5 串口多机通讯 401
15 比较器，掉电检测，内部 1.19V 参考信号源 402
15.1 比较器内部结构图 402
15.2 比较器相关的寄存器 403
15.2.1 比较器控制寄存器 1（CMPCR1） 403
15.2.2 比较器控制寄存器 2（CMPCR2） 404
15.3 范例程序 405
15.3.1 比较器的使用（中断方式） 405
15.3.2 比较器的使用（查询方式） 407
15.3.3 比较器的多路复用应用（比较器+ADC 输入通道） 410
15.3.4 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） 412
15.3.5 比较器检测工作电压（电池电压） 413
16 IAP/EEPROM/DATA-FLASH 418
16.1 EEPROM 操作时间 418
16.2 EEPROM 相关的寄存器 418
16.2.1 EEPROM 数据寄存器（IAP_DATA） 418
16.2.2 EEPROM 地址寄存器（IAP_ADDR） 419
16.2.3 EEPROM 命令寄存器（IAP_CMD） 419
16.2.4 EEPROM 触发寄存器（IAP_TRIG） 419
16.2.5 EEPROM 控制寄存器（IAP_CONTR） 419
16.2.6 EEPROM 等待时间控制寄存器（IAP_TPS） 420
16.3 EEPROM 大小及地址 421
16.4 范例程序 423
16.4.1 EEPROM 基本操作 423
16.4.2 使用 MOVC 读取 EEPROM 426
16.4.3 使用串口送出 EEPROM 数据 430
17 ADC 模数转换，内部 1.19V 参考信号源 435
17.1 ADC 相关的寄存器 435
17.1.1 ADC 控制寄存器（ADC_CONTR），PWM 触发 ADC 控制 435
17.1.2 ADC 配置寄存器（ADCCFG） 436
17.1.3 ADC 转换结果寄存器（ADC_RES，ADC_RESL） 437
17.1.4 ADC 时序控制寄存器 437
17.2 ADC 相关计算公式 439
17.2.1 ADC 速度计算公式 439
17.2.2 ADC 转换结果计算公式 439
17.2.3 反推 ADC 输入电压计算公式 439
17.2.4 反推工作电压计算公式 440
17.3 10 位 ADC 静态特性 440
17.4 范例程序 441
17.4.1 ADC 基本操作（查询方式） 441

17.4.2 ADC 基本操作（中断方式）.....443
17.4.3 格式化 ADC 转换结果.....445
17.4.4 利用 ADC 第 15 通道测量外部电压或电池电压.....448
17.4.5 ADC 做电容感应触摸按键.....450
17.4.6 ADC 作按键扫描应用线路图.....463
17.4.7 检测负电压参考线路图.....464
17.4.8 常用加法电路在 ADC 中的应用.....465
18 PCA/CCP/PWM 应用.....466
18.1 PCA 相关的寄存器.....466
18.1.1 PCA 控制寄存器（CCON）.....467
18.1.2 PCA 模式寄存器（CMOD）.....467
18.1.3 PCA 计数器寄存器（CL，CH）.....467
18.1.4 PCA 模块模式控制寄存器（CCAPMn）.....468
18.1.5 PCA 模块模式捕获值/比较值寄存器（CCAPnL，CCAPnH）.....468
18.1.6 PCA 模块 PWM 模式控制寄存器（PCA_PWMn）.....468
18.2 PCA 工作模式.....470
18.2.1 捕获模式.....470
18.2.2 软件定时器模式.....470
18.2.3 高速脉冲输出模式.....471
18.2.4 PWM 脉宽调制模式及频率计算公式.....471
18.3 利用 CCP/PCA/PWM 模块实现 8~16 位 DAC 的参考线路图.....475
18.4 范例程序.....476
18.4.1 PCA 输出 PWM（6/7/8/10 位）.....476
18.4.2 PCA 捕获测量脉冲宽度.....479
18.4.3 PCA 实现 16 位软件定时.....483
18.4.4 PCA 输出高速脉冲.....486
18.4.5 PCA 扩展外部中断.....489
19 同步串行外设接口 SPI.....492
19.1 SPI 相关的寄存器.....492
19.1.1 SPI 状态寄存器（SPSTAT）.....492
19.1.2 SPI 控制寄存器（SPCTL），SPI 速度控制.....492
19.1.3 SPI 数据寄存器（SPDAT）.....493
19.2 SPI 通信方式.....494
19.2.1 单主单从.....494
19.2.2 互为主从.....494
19.2.3 单主多从.....495
19.3 配置 SPI.....496
19.4 数据模式.....498
19.5 范例程序.....499
19.5.1 SPI 单主单从系统主机程序（中断方式）.....499
19.5.2 SPI 单主单从系统从机程序（中断方式）.....501
19.5.3 SPI 单主单从系统主机程序（查询方式）.....503
19.5.4 SPI 单主单从系统从机程序（查询方式）.....505
19.5.5 SPI 互为主从系统程序（中断方式）.....508

19.5.6 SPI 互为主从系统程序（查询方式）.....510
20 I²C 总线.....514
20.1 I²C 相关的寄存器.....514
20.2 I²C 主机模式.....515
20.2.1 I2C 配置寄存器（I2CCFG），总线速度控制.....515
20.2.2 I2C 主机控制寄存器（I2CMSCR）.....516
20.2.3 I2C 主机辅助控制寄存器（I2CMSAUX）.....517
20.2.4 I2C 主机状态寄存器（I2CMSST）.....517
20.3 I²C 从机模式.....519
20.3.1 I2C 从机控制寄存器（I2CSLCR）.....519
20.3.2 I2C 从机状态寄存器（I2CSLST）.....519
20.3.3 I2C 从机地址寄存器（I2CSLADR）.....521
20.3.4 I2C 数据寄存器（I2CTXD，I2CRXD）.....522
20.4 范例程序.....523
20.4.1 I²C 主机模式访问 AT24C256（中断方式）.....523
20.4.2 I²C 主机模式访问 AT24C256（查询方式）.....529
20.4.3 I²C 主机模式访问 PCF8563.....535
20.4.4 I²C 从机模式（中断方式）.....540
20.4.5 I²C 从机模式（查询方式）.....545
20.4.6 测试 I²C 从机模式代码的主机代码.....549
21 16 位高级 PWM 定时器，支持正交编码器.....556
21.1 简介.....559
21.2 主要特性.....559
21.3 时基单元.....560
21.3.1 读写 16 位计数器.....560
21.3.2 16 位 PWMA_ARR 寄存器的写操作.....561
21.3.3 预分频器.....561
21.3.4 向上计数模式.....561
21.3.5 向下计数模式.....562
21.3.6 中间对齐模式（向上/向下计数）.....564
21.3.7 重复计数器.....565
21.4 时钟/触发控制器.....566
21.4.1 预分频时钟（CK_PSC）.....566
21.4.2 内部时钟源（f_{MASTER}）.....566
21.4.3 外部时钟源模式 1.....567
21.4.4 外部时钟源模式 2.....567
21.4.5 触发同步.....568
21.4.6 与 PWMB 同步.....570
21.5 捕获/比较通道.....573
21.5.1 16 位 PWMA_CCRi 寄存器的写流程.....574
21.5.2 输入模块.....574
21.5.3 输入捕获模式.....575
21.5.4 输出模块.....576
21.5.5 强制输出模式.....577

21.5.6 输出比较模式 577
21.5.7 PWM 模式 578
21.5.8 使用刹车功能（PWMFLT） 583
21.5.9 在外部事件发生时清除 OCiREF 信号 584
21.5.10 编码器接口模式 585
21.6 中断 586
21.7 PWMA/PWMB 寄存器描述 588
21.7.1 输出使能寄存器（PWMx_ENO） 588
21.7.2 输出附加使能寄存器（PWMx_IOAUX） 589
21.7.3 控制寄存器 1（PWMx_CR1） 589
21.7.4 控制寄存器 2（PWMx_CR2），及实时触发 ADC 591
21.7.5 从模式控制寄存器(PWMx_SMCR) 592
21.7.6 外部触发寄存器(PWMx_ETR) 594
21.7.7 中断使能寄存器(PWMx_IER) 595
21.7.8 状态寄存器 1(PWMx_SR1) 595
21.7.9 状态寄存器 2(PWMx_SR2) 596
21.7.10 事件产生寄存器（PWMx_EGR） 596
21.7.11 捕获/比较模式寄存器 1（PWMx_CCMR1） 597
21.7.12 捕获/比较模式寄存器 2（PWMx_CCMR2） 601
21.7.13 捕获/比较模式寄存器 3（PWMx_CCMR3） 602
21.7.14 捕获/比较模式寄存器 4（PWMx_CCMR4） 603
21.7.15 捕获/比较使能寄存器 1（PWMx_CCER1） 604
21.7.16 捕获/比较使能寄存器 2（PWMx_CCER2） 606
21.7.17 计数器高 8 位（PWMx_CNTRH） 607
21.7.18 计数器低 8 位（PWMx_CNTRL） 607
21.7.19 预分频器高 8 位（PWMx_PSCRH），输出频率计算公式 607
21.7.20 预分频器低 8 位（PWMx_PSCRL） 607
21.7.21 自动重装载寄存器高 8 位（PWMx_ARRH） 608
21.7.22 自动重装载寄存器低 8 位（PWMx_ARRL） 608
21.7.23 重复计数器寄存器（PWMx_RCR） 608
21.7.24 捕获/比较寄存器 1/5 高 8 位（PWMx_CCR1H） 608
21.7.25 捕获/比较寄存器 1/5 低 8 位（PWMx_CCR1L） 608
21.7.26 捕获/比较寄存器 2/6 高 8 位（PWMx_CCR2H） 609
21.7.27 捕获/比较寄存器 2/6 低 8 位（PWMx_CCR2L） 609
21.7.28 捕获/比较寄存器 3/7 高 8 位（PWMx_CCR3H） 609
21.7.29 捕获/比较寄存器 3/7 低 8 位（PWMx_CCR3L） 609
21.7.30 捕获/比较寄存器 4/8 高 8 位（PWMx_CCR4H） 609
21.7.31 捕获/比较寄存器 4/8 低 8 位（PWMx_CCR4L） 609
21.7.32 刹车寄存器（PWMx_BKR） 610
21.7.33 死区寄存器（PWMx_DTR） 611
21.7.34 输出空闲状态寄存器（PWMx_OISR） 611
21.8 范例程序 612
21.8.1 六步 PWM 驱动无刷直流马达(带 HALL) 612
21.8.2 BLDC 无刷直流电机驱动(无 HALL) 623

21.8.3 正交编码器模式 633
21.8.4 单脉冲模式（触发控制脉冲输出） 635
21.8.5 门控模式（输入电平使能计数器） 637
21.8.6 外部时钟模式 639
21.8.7 输入捕获模式测量脉冲周期（捕获上升沿到上升沿或者下降沿到下降沿） 641
21.8.8 输入捕获模式测量脉冲高电平宽度（捕获上升沿到下降沿） 643
21.8.9 输入捕获模式测量脉冲低电平宽度（捕获下降沿到上升沿） 644
21.8.10 输入捕获模式同时测量脉冲周期和占空比 645
21.8.11 带死区控制的 PWM 互补输出 646
21.8.12 PWM 端口做外部中断（下降沿中断或者上升沿中断） 648
21.8.13 输出任意周期和任意占空比的波形 649
21.8.14 使用 PWM 的 CEN 启动 PWMA 定时器，实时触发 ADC 650
21.8.15 利用 PWM 实现 16 位 DAC 的参考线路图 652
21.8.16 利用 PWM 实现互补 SPWM 652
22 增强型双数据指针 657
22.1 相关的特殊功能寄存器 657
22.1.1 第 1 组 16 位数据指针寄存器（DPTR0） 657
22.1.2 第 2 组 16 位数据指针寄存器（DPTR1） 657
22.1.3 数据指针控制寄存器（DPS） 657
22.1.4 数据指针控制寄存器（TA） 658
22.2 范例程序 660
22.2.1 示例代码 1 660
22.2.2 示例代码 2 661
23 MDU16 硬件 16 位乘除法器 663
23.1 相关的特殊功能寄存器 663
23.1.1 操作数 1 数据寄存器（MD0～MD3） 663
23.1.2 操作数 2 数据寄存器（MD4～MD5） 663
23.1.3 MDU 模式控制寄存器（ARCON），运算所需时钟数 664
23.1.4 MDU 操作控制寄存器（OPCON） 664
23.2 范例程序 666
附录 A 编译器（汇编器）/仿真器使用指南 668
附录 B 如何让传统的 8051 单片机学习板可仿真 673
附录 C STC-USB 驱动程序安装说明 675
附录 D USB 下载步骤演示 738
附录 E RS485 自动控制或 I/O 口控制线路图 742
附录 F STC 工具使用说明书 743
F.1 概述 743
F.2 系统可编程（ISP）流程说明 743
F.3 USB 型联机/脱机下载工具 U8W/U8W-Mini 744
F.3.1 安装 U8W/U8W-Mini 驱动程序 746
F.3.2 U8W 的功能介绍 749
F.3.3 U8W 的在线联机下载使用说明 750
F.3.4 U8W 的脱机下载使用说明 753
F.3.5 U8W-Mini 的功能介绍 761

F.3.6 U8W-Mini 的在线联机下载使用说明 762
F.3.7 U8W-Mini 的脱机下载使用说明 763
F.3.8 制作/更新 U8W/U8W-Mini 769
F.3.9 U8W/U8W-Mini 设置直通模式（可用于仿真） 771
F.3.10 U8W/U8W-Mini 的参考电路 771
F.4 STC 通用 USB 转串口工具 773
F.4.1 STC 通用 USB 转串口工具外观图 773
F.4.2 STC 通用 USB 转串口工具布局图 774
F.4.3 STC 通用 USB 转串口工具驱动安装 775
F.4.4 使用 STC 通用 USB 转串口工具下载程序到 MCU 776
F.4.5 使用 STC 通用 USB 转串口工具仿真用户代码 778
F.5 应用线路图 785
F.5.1 U8W 工具应用参考线路图 785
F.5.2 STC 通用 USB 转串口工具应用参考线路图 785
附录 G U8W 下载工具中 RS485 部分线路图 787
附录 H 运行用户程序时收到用户命令后自动启动 ISP 下载(不停电) 788
附录 I 使用 STC 的 IAP 系列单片机开发自己的 ISP 程序 790
附录 J 用户程序复位到系统区进行 ISP 下载的方法（不停电） 802
附录 K 使用第三方 MCU 对 STC12H 系列单片机进行 ISP 下载范例程序 808
附录 L 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 816
附录 M STC8H 系列正交解码示例（成都逐飞科技友情提供） 820
附录 N 在 Keil 中建立多文件项目的方法 824
附录 O 关于中断号大于 31 在 Keil 中编译出错的处理 828
附录 P 电气特性 838
P.1 绝对最大额定值 838
P.2 直流特性（3.3V） 839
P.3 直流特性（5.0V） 841
P.4 内部 IRC 温漂特性（参考温度 25℃） 842
P.5 低压复位门槛电压（测试温度 25℃） 842
附录 Q 应用注意事项 843
Q.1 关于 STC12H 系列 IO 口的注意事项 843
附录 R QFN/DFN 封装元器件焊接方法 844
附录 S STC12H 系列单片机取代 STC12C56/54 系列的注意事项 847
附录 T 更新记录 850

1 概述

STC12H 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，STC12H 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，STC12H 系列单片机仅需 147 个时钟，而传统 8051 则需要 1944 个时钟。STC12H 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(±0.3%，常温下+25℃)，−1.38%～+1.42%温飘(-40℃~+85℃)，−0.88%～+1.05%温飘(-20℃~+65℃)。ISP 编程时 4MHz~35MHz 宽范围可设置（注意：温度范围为-40℃~+85℃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（可适当调高或调低）、内部 32KHz 的低速 IRC、外部 4M~33M 晶振或外部时钟信号。用户代码中可自由选择时钟源，时钟源选定后可再经过 8-bit 的分频器分频后再将时钟信号提供给 CPU 和各个外设（如定时器、串口、SPI 等）。

MCU 提供两种低功耗模式：IDLE 模式和 STOP 模式。IDLE 模式下，MCU 停止给 CPU 提供时钟，CPU 无时钟，CPU 停止执行指令，但所有的外设仍处于工作状态，此时功耗约为 1.3mA（6MHz 工作频率）。STOP 模式即为主时钟停振模式，即传统的掉电模式/停电模式/停机模式，此时 CPU 和全部外设都停止工作，功耗可降低到 0.6uA@Vcc=5.0V，0.4uA@Vcc=3.3V。

掉电模式可以使用 INT0(P3.2)、INT1(P3.3)、INT2(P0.2)、INT3(P0.3)、INT4(P3.0)、T0(P3.4)、T1(P3.5)、T2(P1.2)、T3(P0.0)、T4(P0.2)、RXD(P3.0/P3.5/P1.6)、RXD2(P1.0/P5.6)、I2C_SDA(P1.6/P2.4/P3.3)、SPI_SS(P1.4/P2.2/P3.5)、CCP0(P3.7)、CCP1(P3.5)、CCP2(P2.0)、CCP2(P2.4)、全部 I/O 口中断以及比较器中断、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、高级 PWM 以及 I^2C、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。

STC12H 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	高级 PWM	PCA CCP PWM	CMP	SPI	I2C	MDU16	I/O 中断
STC12H1K08 系列	30	2(U1/U2)	5(T0/T1/T2/T3/T4)	15CH*10B	●	●	●	●	●	●	●

2 特性、价格及管脚

2.1 STC12H1K08 系列

2.1.1 特性及价格

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

单片机型号	工作电压（V）	Flash 程序存储器 10 万次 字节	idata，内部传统 8051 RAM 字节	xdata，内部大容量扩展 SRAM 字节	强大的双 DPTR 可增可减	EEPROM 10 万次 字节	I/O 口最多数量	串口并可掉电唤醒	SPI	I^2C	MDU16 硬件 16 位乘除法器	定时器/计数器（T0-T4 外部管脚也可掉电唤醒）	16 位高级 PWM 定时器 互补对称死区控制	PCA/CCP/PWM（可当外部中断并可掉电唤醒）	掉电唤醒专用定时器	15 路高速 ADC（8 路 PWM 可当 8 路 D/A 使用）	比较器（可当 1 路 A/D，可作外部掉电检测）	内部低压检测中断并可掉电唤醒	看门狗 复位定时器	内部高可靠复位（可选复位门槛电压）	内部高精准时钟（36MHz 以下可调）追频	可对外输出时钟及复位	程序加密后传输（防拦截）	可设置下次更新程序需口令	支持 RS485 下载	支持软件 USB 直接下载	本身就可在线仿真	价格及封装		2020 年新品供货信息
																												LQFP32	TSSOP20	
STC12H1K08	1.9-5.5	8K	256	1K	2	20K	30	2	有	有	有	5	8	4	有	10 位	有	有	有	4 级	有	是	有	是	是	是	是	¥2.5	¥2.2	大量供货
STC12H1K16	1.9-5.5	16K	256	1K	2	12K	30	2	有	有	有	5	8	4	有	10 位	有	有	有	4 级	有	是	有	是	是	是	是	¥2.6	¥2.3	
STC12H1K24	1.9-5.5	24K	256	1K	2	4K	30	2	有	有	有	5	8	4	有	10 位	有	有	有	4 级	有	是	有	是	是	是	是	¥2.7	¥2.4	
STC12H1K28	1.9-5.5	28K	256	1K	2	IAP	30	2	有	有	有	5	8	4	有	10 位	有	有	有	4 级	有	是	有	是	是	是	是	¥2.7	¥2.4	

➢ 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
- ✓ 26 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

➢ 工作温度

- ✓ -40℃～85℃（超温度范围应用请参考电器特性章节说明）

➢ **Flash 存储器**

- ✓ 最大 33K 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（ISP 编程时可进行上下调整）
  - ⊕ 误差±0.3%（常温下 25℃）
  - ⊕ -1.35%～+1.30%温漂（全温度范围，-40℃～85℃）
  - ⊕ -0.76%～+0.98%温漂（温度范围，-20℃～65℃）
- ✓ 内部 32KHz 低速 IRC（误差较大）
- ✓ 外部晶振（4MHz～33MHz）和外部时钟

  用户可自由选择上面的 3 种时钟源

➢ **复位**

- ✓ 硬件复位
  - ⊕ 上电复位，实测电压值为 1.69V～1.82V。（**在芯片未使能低压复位功能时有效**）

    上电复位电压由一个上限电压和一个下限电压组成的电压范围，当工作电压从 5V/3.3V 向下掉到上电复位的下限门槛电压时，芯片处于复位状态；当电压从 0V 上升到上电复位的上限门槛电压时，芯片解除复位状态。
  - ⊕ 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（实测为 1.90V～2.04V）、2.4V（实测为 2.30V～2.50V）、2.7V（实测为 2.61V～2.82V）、3.0V（实测为 2.90V～3.13V）。

    每级低压检测电压都是由一个上限电压和一个下限电压组成的电压范围，当工作电压从 5V/3.3V 向下掉到低压检测的下限门槛电压时，低压检测生效；当电压从 0V 上升到低压检测的上限门槛电压时，低压检测生效。
- ✓ 软件复位
  - ⊕ 软件方式写复位触发寄存器

➢ **中断**

- ✓ 提供 26 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、I^2C、比较器、PWM1、PWM2、PCA/CCP/PWM、~~**EXP0、EXP1、EXP2、EXP3、EXP4、EXP5**~~
- ✓ 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(P0.2)、INT3(P0.3)、INT4(P3.0)、T0(P3.4)、T1(P3.5)、T2(P1.2)、T3(P0.0)、T4(P0.2)、RXD(P3.0/P3.5/P1.6)、RXD2(P1.0/P5.6)、CCP0(P3.7)、CCP1(P3.5)、CCP2(P2.0)、CCP2(P2.4)、I2C_SDA(P1.6/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
- ✓ **8 路/2 组高级 PWM，可实现带死区的控制信号，并支持外部异常检测功能，另外还支持 16 位定时器、8 个外部中断、8 路外部捕获测量脉宽等功能**
- ✓ SPI：支持主机模式和从机模式以及主机/从机自动切换
- ✓ I^2C：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）**
- ~~✓ **I/O 口中断：所有的 I/O 均支持中断，每组 I/O 中断有独立的中断入口地址，所有的 I/O 中断可支持 4 种中**~~

~~断模式：高电平中断、低电平中断、上升沿中断、下降沿中断。~~经测试发现有问题，请暂时不要使用

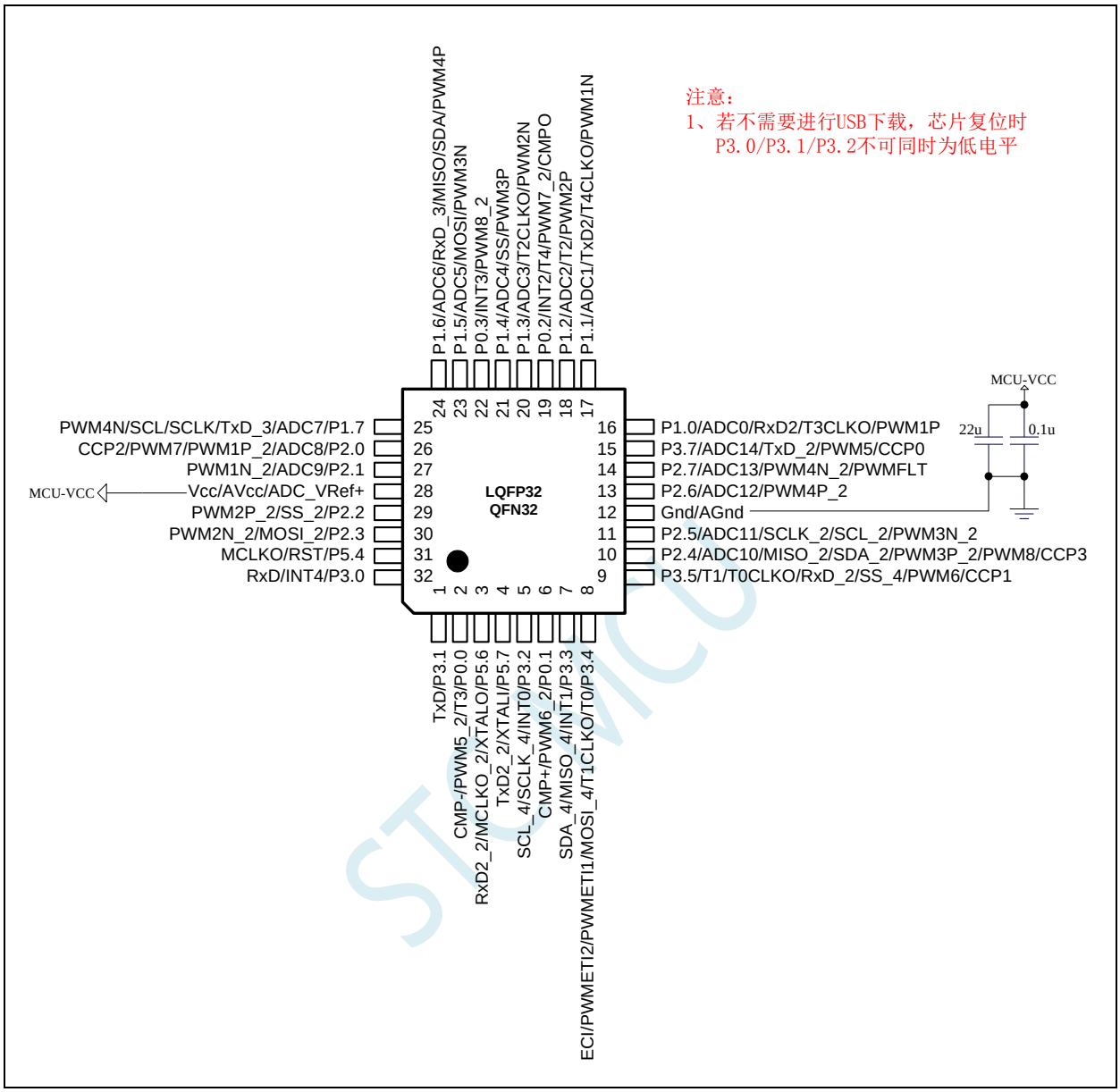
- 模拟外设
  - ✓ 超高速 ADC，支持 **10 位高精度** 15 通道（通道 0～通道 14）的模数转换，**速度最快能达到 500K（每秒进行 50 万次 ADC 转换）**
  - ✓ ADC 的通道 15 用于测试内部 1.19V 参考信号源（芯片在出厂时，内部参考信号源已调整为 1.19V）
  - ✓ 比较器，一组比较器（比较器的正端可选择 CMP+端口和所有的 ADC 输入端口，所以比较器可当作多路比较器进行分时复用）
  - ✓ DAC：8 路高级 PWM 定时器可当 8 路 DAC 使用，4 路 PCA/CCP/PWM 可当 4 路 DAC 使用

- **GPIO**
  - ✓ 最多可达 30 个 GPIO：P0.0~P0.3、P1.0~ P1.7、P2.0~P2.7、P3.0~P3.5、P3.7、P5.4、P5.6~P5.7
  - ✓ 所有的 GPIO 均支持如下 4 种模式：准双向口模式、强推挽输出模式、开漏输出模式、高阻输入模式
  - ✓ **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 口时必须先设置 IO 口模式。另外每个 I/O 均可独立使能内部 4K 上拉电阻**

- 封装
  - ✓ LQFP32（现货）、QFN32（暂无）、SOP28（暂无）、SKDIP28（暂无）、TSSOP20（暂无）、SOP20（暂无）、SOP16（暂无）

STC MCU

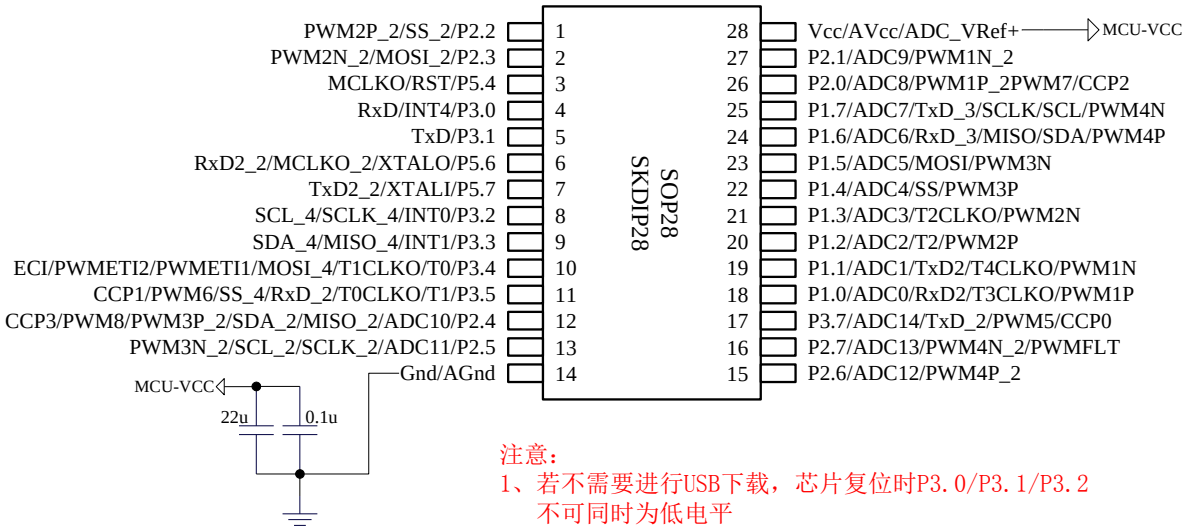
2.1.2 管脚图

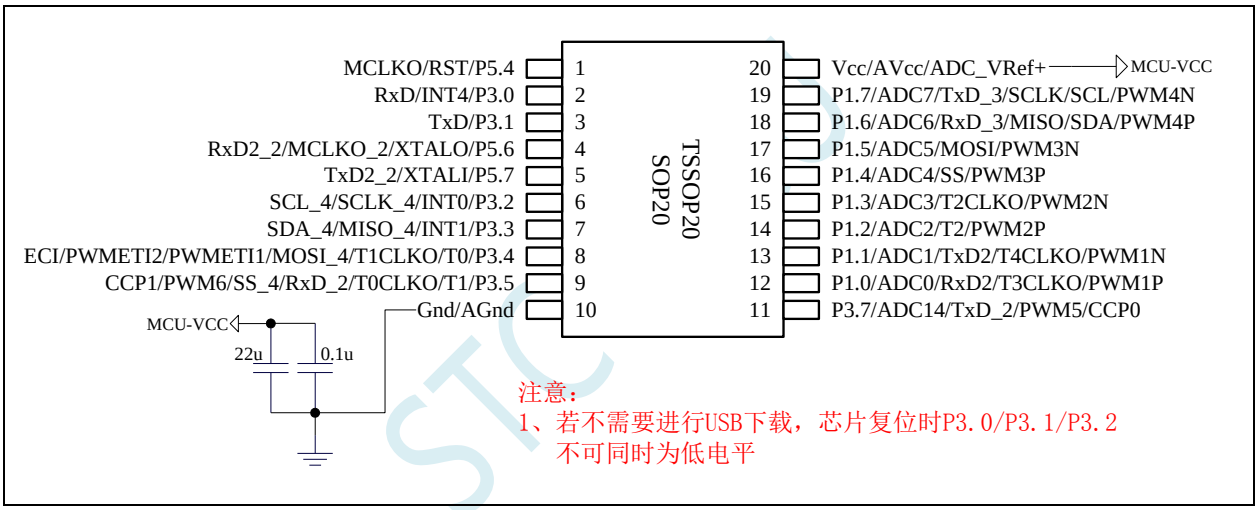


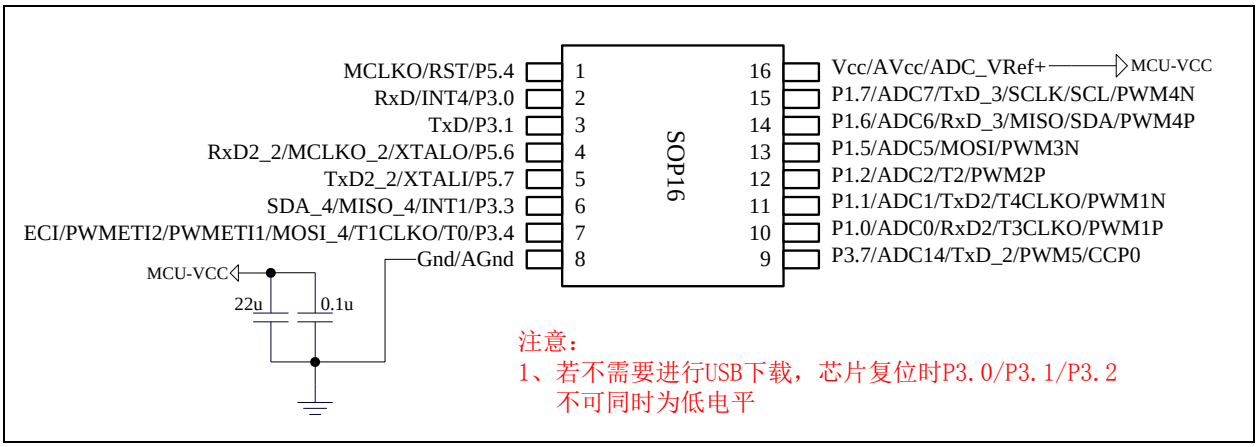
注意： 1、除 P3.0 和 P3.1 外，其余所有 I/O 口上电后的状态均为高阻输入状态，用户在使用 I/O 口时必须先设置 I/O 口模式

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


PWM2P_2/SS_2/P2.2 1
PWM2N_2/MOSI_2/P2.3 2
MCLKO/RST/P5.4 3
RxD/INT4/P3.0 4
TxD/P3.1 5
RxD2_2/MCLKO_2/XTALO/P5.6 6
TxD2_2/XTALI/P5.7 7
SCL_4/SCLK_4/INT0/P3.2 8
SDA_4/MISO_4/INT1/P3.3 9
ECI/PWMETI2/PWMETI1/MOSI_4/T1CLKO/T0/P3.4 10
CCP1/PWM6/SS_4/RxD_2/T0CLKO/T1/P3.5 11
CCP3/PWM8/PWM3P_2/SDA_2/MISO_2/ADC10/P2.4 12
PWM3N_2/SCL_2/SCLK_2/ADC11/P2.5 13
Gnd/AGnd 14
SOP28
SKDIP28
28 Vcc/AVcc/ADC_VRef+
MCU-VCC
27 P2.1/ADC9/PWM1N_2
26 P2.0/ADC8/PWM1P_2PWM7/CCP2
25 P1.7/ADC7/TxD_3/SCLK/SCL/PWM4N
24 P1.6/ADC6/RxD_3/MISO/SDA/PWM4P
23 P1.5/ADC5/MOSI/PWM3N
22 P1.4/ADC4/SS/PWM3P
21 P1.3/ADC3/T2CLKO/PWM2N
20 P1.2/ADC2/T2/PWM2P
19 P1.1/ADC1/TxD2/T4CLKO/PWM1N
18 P1.0/ADC0/RxD2/T3CLKO/PWM1P
17 P3.7/ADC14/TxD_2/PWM5/CCP0
16 P2.7/ADC13/PWM4N_2/PWMFLT
15 P2.6/ADC12/PWM4P_2
MCU-VCC
22u
0.1u
注意：
1、若不需要进行USB下载，芯片复位时P3.0/P3.1/P3.2
不可同时为低电平


MCLKO/RST/P5.4 1
RxD/INT4/P3.0 2
TxD/P3.1 3
RxD2_2/MCLKO_2/XTALO/P5.6 4
TxD2_2/XTALI/P5.7 5
SCL_4/SCLK_4/INT0/P3.2 6
SDA_4/MISO_4/INT1/P3.3 7
ECI/PWMETI2/PWMETI1/MOSI_4/T1CLKO/T0/P3.4 8
CCP1/PWM6/SS_4/RxD_2/T0CLKO/T1/P3.5 9
Gnd/AGnd 10
TSSOP20
SOP20
20 Vcc/AVcc/ADC_VRef+
MCU-VCC
19 P1.7/ADC7/TxD_3/SCLK/SCL/PWM4N
18 P1.6/ADC6/RxD_3/MISO/SDA/PWM4P
17 P1.5/ADC5/MOSI/PWM3N
16 P1.4/ADC4/SS/PWM3P
15 P1.3/ADC3/T2CLKO/PWM2N
14 P1.2/ADC2/T2/PWM2P
13 P1.1/ADC1/TxD2/T4CLKO/PWM1N
12 P1.0/ADC0/RxD2/T3CLKO/PWM1P
11 P3.7/ADC14/TxD_2/PWM5/CCP0
MCU-VCC
22u
0.1u
注意：
1、若不需要进行USB下载，芯片复位时P3.0/P3.1/P3.2
不可同时为低电平


MCLKO/RST/P5.4 1
RxD/INT4/P3.0 2
TxD/P3.1 3
RxD2_2/MCLKO_2/XTALO/P5.6 4
TxD2_2/XTALI/P5.7 5
SDA_4/MISO_4/INT1/P3.3 6
ECI/PWMETI2/PWMETI1/MOSI_4/T1CLKO/T0/P3.4 7
Gnd/AGnd 8
SOP16
16 Vcc/AVcc/ADC_VRef+
MCU-VCC
15 P1.7/ADC7/TxD_3/SCLK/SCL/PWM4N
14 P1.6/ADC6/RxD_3/MISO/SDA/PWM4P
13 P1.5/ADC5/MOSI/PWM3N
12 P1.2/ADC2/T2/PWM2P
11 P1.1/ADC1/TxD2/T4CLKO/PWM1N
10 P1.0/ADC0/RxD2/T3CLKO/PWM1P
9 P3.7/ADC14/TxD_2/PWM5/CCP0
MCU-VCC
22u
0.1u
注意：
1、若不需要进行USB下载，芯片复位时P3.0/P3.1/P3.2
不可同时为低电平

2.1.3 管脚说明

编号			名称	类型	说明
LQFP32 QFN32	SOP28 SKDIP28	TSSOP20 SOP20			
1	5	3	P3.1	I/O	标准 IO 口
			TxD	O	串口 1 的发送脚
2			P0.0	I/O	标准 IO 口
			T3	I	定时器 3 外部时钟输入
			PWM5_3	I/O	PWM5 的捕获输入和脉冲输出
			CMP-	I	比较器负极输入
3	6	4	P5.6	I/O	标准 IO 口
			RxD2_2	I	串口 2 的接收脚
			XTALO	O	外部晶振的输出脚
			MCLKO_2	O	主时钟分频输出
4	7	5	P5.7	I/O	标准 IO 口
			TxD2_2	O	串口 2 的发送脚
			XTALI	I	外部晶振/外部时钟的输入脚
5	8	6	P3.2	I/O	标准 IO 口
			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
6			P0.1	I/O	标准 IO 口
			PWM6_3	I/O	PWM6 的捕获输入和脉冲输出
			CMP+	I	比较器正极输入
7	9	7	P3.3	I/O	标准 IO 口
			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 的数据线
8	10	8	P3.4	I/O	标准 IO 口
			T0	I	定时器 0 外部时钟输入
			T1CLKO	O	定时器 1 时钟分频输出
			MOSI_4	I/O	SPI 主机输出从机输入
			PWMETI	I	PWM 外部触发输入脚
			PWMETI2	I	PWM 外部触发输入脚 2
			ECI	I	PCA 的外部脉冲输入

编号			名称	类型	说明
LQFP32 QFN32	SOP28 SKDIP28	TSSOP20 SOP20			
9	11	9	P3.5	I/O	标准 IO 口
			T1	I	定时器 1 外部时钟输入
			T0CLKO	O	定时器 0 时钟分频输出
			RxD_2	I	串口 1 的接收脚
			SS_4	I/O	SPI 从机选择
			PWM6	I/O	PWM6 的捕获输入和脉冲输出
			CCP1	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
10	12		P2.4	I/O	标准 IO 口
			ADC10	I	ADC 模拟输入通道 10
			MISO_2	I/O	SPI 主机输入从机输出
			SDA_2	I/O	I2C 的数据线
			PWM3P_2	I/O	PWM3 捕获输入和脉冲输出正极
			PWM8	I/O	PWM8 的捕获输入和脉冲输出
			CCP3	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
11	13		P2.5	I/O	标准 IO 口
			ADC11	I	ADC 模拟输入通道 11
			SCLK_2	I/O	SPI 的时钟脚
			SCL_2	I/O	I2C 的时钟线
			PWM3N_2	I/O	PWM3 的捕获输入和脉冲输出负极
12	14	10	Gnd	Gnd	地线
			AGnd	Gnd	ADC 地线
13	15		P2.6	I/O	标准 IO 口
			ADC12	I	ADC 模拟输入通道 12
			PWM4P_2	I/O	PWM4 捕获输入和脉冲输出正极
14	16		P2.7	I/O	标准 IO 口
			ADC13	I	ADC 模拟输入通道 13
			PWM4N_2	I/O	PWM4 的捕获输入和脉冲输出负极
			PWMFLT	I	PWM1 的外部异常检测脚
15	17	11	P3.7	I/O	标准 IO 口
			ADC14	I	ADC 模拟输入通道 14
			TxD_2	O	串口 1 的发送脚
			PWM5	I/O	PWM5 的捕获输入和脉冲输出
			CCP0	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出

编号			名称	类型	说明
LQFP32 QFN32	SOP28 SKDIP28	TSSOP20 SOP20			
16	18	12	P1.0	I/O	标准 IO 口
			ADC0	I	ADC 模拟输入通道 0
			RxD2	I	串口 2 的接收脚
			T3CLKO	O	定时器 3 时钟分频输出
			PWM1P	I/O	PWM1 捕获输入和脉冲输出正极
17	19	13	P1.1	I/O	标准 IO 口
			ADC1	I	ADC 模拟输入通道 1
			TxD2	O	串口 2 的发送脚
			T4CLKO	O	定时器 4 时钟分频输出
			PWM1N	I/O	PWM1 的捕获输入和脉冲输出负极
18	20	14	P1.2	I/O	标准 IO 口
			ADC2	I	ADC 模拟输入通道 2
			T2	I	定时器 2 外部时钟输入
			PWM2P	I/O	PWM2 捕获输入和脉冲输出正极
19			P0.2	I/O	标准 IO 口
			INT2	I	外部中断 2
			T4	I	定时器 4 外部时钟输入
			PWM7_2	I/O	PWM7 的捕获输入和脉冲输出
			CMPO	O	比较器输出
20	21	15	P1.3	I/O	标准 IO 口
			ADC3	I	ADC 模拟输入通道 3
			T2CLKO	O	定时器 2 时钟分频输出
			PWM2N	I/O	PWM2 的捕获输入和脉冲输出负极
21	22	16	P1.4	I/O	标准 IO 口
			ADC4	I	ADC 模拟输入通道 4
			SS	I/O	SPI 从机选择
			PWM3P	I/O	PWM3 捕获输入和脉冲输出正极
22			P0.3	I/O	标准 IO 口
			INT3	I	外部中断 3
			PWM8_2	I/O	PWM8 的捕获输入和脉冲输出
23	23	17	P1.5	I/O	标准 IO 口
			ADC5	I	ADC 模拟输入通道 5
			MOSI	I/O	SPI 主机输出从机输入
			PWM3N	I/O	PWM3 的捕获输入和脉冲输出负极

编号			名称	类型	说明
LQFP32 QFN32	SOP28 SKDIP28	TSSOP20 SOP20			
24	24	18	P1.6	I/O	标准 IO 口
			ADC6	I	ADC 模拟输入通道 6
			RxD_3	I	串口 1 的接收脚
			MISO	I/O	SPI 主机输入从机输出
			SDA	I/O	I2C 的数据线
			PWM4P	I/O	PWM4 捕获输入和脉冲输出正极
25	25	19	P1.7	I/O	标准 IO 口
			ADC7	I	ADC 模拟输入通道 7
			TxD_3	O	串口 1 的发送脚
			SCLK	I/O	SPI 的时钟脚
			SCL	I/O	I2C 的时钟线
			PWM4N	I/O	PWM4 的捕获输入和脉冲输出负极
26	26		P2.0	I/O	标准 IO 口
			ADC8	I	ADC 模拟输入通道 8
			PWM1P_2	I/O	PWM1 捕获输入和脉冲输出正极
			PWM7	I/O	PWM7 的捕获输入和脉冲输出
			CCP2	I/O	PCA 的捕获输入、脉冲输出和 PWM 输出
27	27		P2.1	I/O	标准 IO 口
			ADC9	I	ADC 模拟输入通道 9
			PWM1N_2	I/O	PWM1 的捕获输入和脉冲输出负极
28	28	20	Vcc	Vcc	电源脚
			AVcc	Vcc	ADC 电源
			ADC_VRef+	I	ADC 外部参考电压源输入脚
29	1		P2.2	I/O	标准 IO 口
			SS_2	I/O	SPI 从机选择
			PWM2P_2	I/O	PWM2 捕获输入和脉冲输出正极
30	2		P2.3	I/O	标准 IO 口
			MOSI_2	I/O	SPI 主机输出从机输入
			PWM2N_2	I/O	PWM2 的捕获输入和脉冲输出负极
31	3	1	P5.4	I/O	标准 IO 口
			RST	I	复位引脚
			MCLKO	O	主时钟分频输出
32	4	2	P3.0	I/O	标准 IO 口
			RxD	I	串口 1 的接收脚
			INT4	I	外部中断 4

3 功能脚切换

STC12H 系列单片机的特殊外设串口、SPI、PWM、I²C 以及总线控制脚可以在多个 I/O 直接进行切换，以实现一个外设当作多个设备进行分时复用。

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]		-	-	SPI_S[1:0]		0	-	nnxx,000x
P_SW2	外设端口切换寄存器 2	BAH	EAXFR	-	I2C_S[1:0]		-	-	-	S2_S	0x00,xxx0

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	MCLKODIV[6:0]							0000,0000
PWMA_PS	PWMA 切换寄存器	FEB2H	C4PS[1:0]		C3PS[1:0]		C2PS[1:0]		C1PS[1:0]		0000,0000
PWMB_PS	PWMB 切换寄存器	FEB6H	C8PS[1:0]		C7PS[1:0]		C6PS[1:0]		C5PS[1:0]		0000,0000
PWMA_ETRPS	PWMA 的 ETR 选择寄存器	FEB0H						BRKAPS	ETRAPS[1:0]		xxxx,x000
PWMB_ETRPS	PWMB 的 ETR 选择寄存器	FEB4H						BRKBPS	ETRBPS[1:0]		xxxx,x000

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

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

SPI_S[1:0]：SPI 功能脚选择位

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

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]		-	-	-	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]：I^2C 功能脚选择位

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

S2_S：串口 2 功能脚选择位

S2_S	RxD2	TxD2
0	P1.0	P1.1
1	P5.6	P5.7

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	P5.6

STCMCU

3.1.4 高级 PWM 选择寄存器（PWMx_PS）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_PS	FEB2H	C4PS[1:0]		C3PS[1:0]		C2PS[1:0]		C1PS[1:0]	
PWMB_PS	FEB6H	C8PS[1:0]		C7PS[1:0]		C6PS[1:0]		C5PS[1:0]	

C1PS[1:0]：高级 PWM 通道 1 输出脚选择位

C1PS[1:0]	PWM1P	PWM1N
00	P1.0	P1.1
01	P2.0	P2.1
10	-	-
11	-	-

C2PS[1:0]：高级 PWM 通道 2 输出脚选择位

C2PS[1:0]	PWM2P	PWM2N
00	P1.2	P1.3
01	P2.2	P2.3
10	-	-
11	-	-

C3PS[1:0]：高级 PWM 通道 3 输出脚选择位

C3PS[1:0]	PWM3P	PWM3N
00	P1.4	P1.5
01	P2.4	P2.5
10	-	-
11	-	-

C4PS[1:0]：高级 PWM 通道 4 输出脚选择位

C4PS[1:0]	PWM4P	PWM4N
00	P1.6	P1.7
01	P2.6	P2.7
10	-	-
11	-	-

STC MCU

C5PS[1:0]：高级 PWM 通道 5 输出脚选择位

C5PS[1:0]	PWM5
00	P3.7
01	P0.0
10	-
11	-

C6PS[1:0]：高级 PWM 通道 6 输出脚选择位

C6PS[1:0]	PWM6
00	P3.5
01	P0.1
10	-
11	-

C7PS[1:0]：高级 PWM 通道 7 输出脚选择位

C7PS[1:0]	PWM7
00	P2.0
01	P0.2
10	-
11	-

C8PS[1:0]：高级 PWM 通道 8 输出脚选择位

C8PS[1:0]	PWM8
00	P2.4
01	P0.3
10	-
11	-

STC MCU

3.1.5 高级 PWM 功能脚选择寄存器（PWMx_ETRPS）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ETRPS	FEB0H						BRKAPS	ETRAPS[1:0]	
PWMB_ETRPS	FEB4H						BRKBPS	ETRBPS[1:0]	

ETR1PS[1:0]：高级 PWM1 的外部触发脚 ERI 选择位

ETR1PS [1:0]	PWMETI
00	P3.4
01	-
10	-
11	-

ETR2PS[1:0]：高级 PWM2 的外部触发脚 ERI2 选择位

ETR2PS [1:0]	PWMETI2
00	P3.4
01	-
10	-
11	-

BRK1PS：高级 PWM1 的刹车脚 PWMFLT 选择位

BRK1PS	PWMFLT
0	P2.7
1	比较器的输出

BRK2PS：高级 PWM2 的刹车脚 PWMFLT2 选择位

BRK2PS	PWMFLT2
0	P2.7
1	比较器的输出

STCMCU

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     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;

    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

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW1     DATA     0A2H

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         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

            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     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;

    P_SW2 = 0x00;                   //RXD2/P1.0, TXD2/P1.1
//  P_SW2 = 0x01;                   //RXD2_2/P5.6, TXD2_2/P5.7

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA        0BAH

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
```

STC MCU

```
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     P_SW2,#00H          ;RXD2/P1.0, TXD2/P1.1
;       MOV     P_SW2,#01H          ;RXD2_2/P5.6, TXD2_2/P5.7

        SJMP    $

        END
```

3.2.3 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     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;

    P_SW1 = 0x00;           //SS/P1.4, MOSI/P1.5, MISO/P1.6, SCLK/P1.7
//  P_SW1 = 0x04;           //SS_2/P2.2, MOSI_2/P2.3, MISO_2/P2.4, SCLK_2/P2.5
//  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

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

            SJMP        $

            END
```

3.2.4 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     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;

        P_SW2 = 0x00;                   //SCL/P1.7, SDA/P1.6
//      P_SW2 = 0x10;                   //SCL_2/P2.5, SDA_2/P2.4
//      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

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

        SJMP    $

        END
```

3.2.5 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       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;

    P_SW2 = 0x80;
    CLKOCR = 0x04;                          //IRC/4 output via MCLKO/P5.4
//  CLKOCR = 0x84;                          //IRC/4 output via MCLKO_2/P5.6
    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

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

            SJMP        $
```

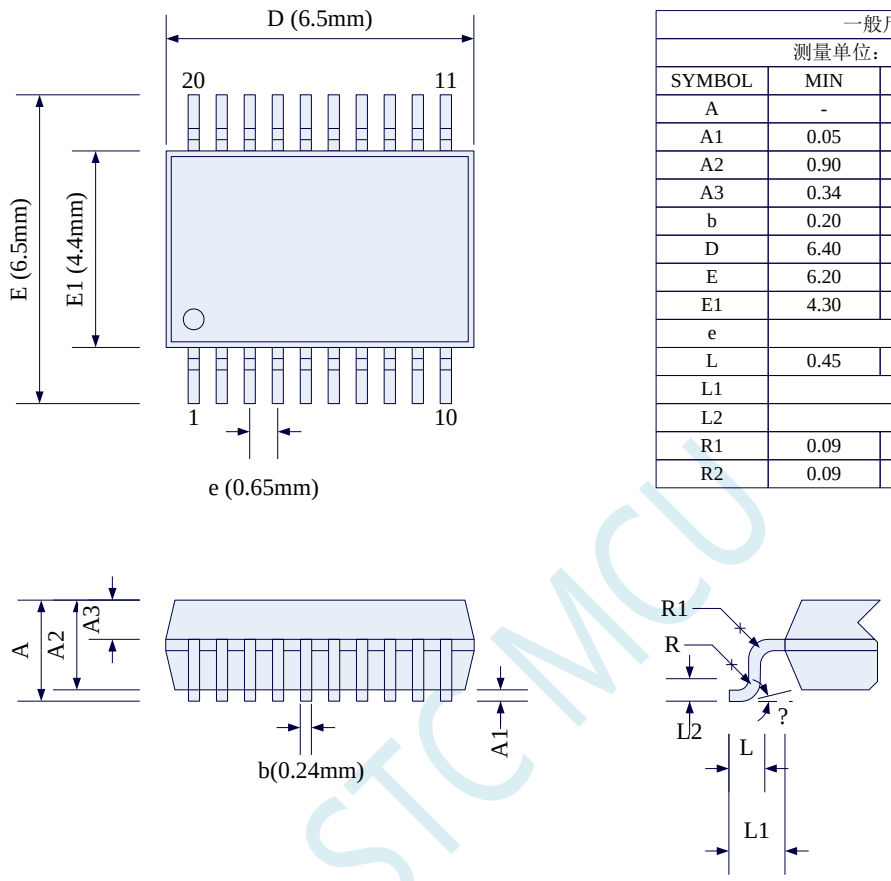
STC MCU

END

STC MCU

4 封装尺寸图

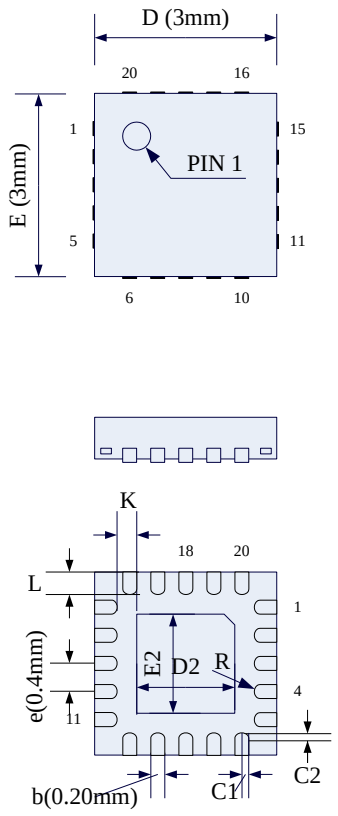
4.1 TSSOP20 封装尺寸图



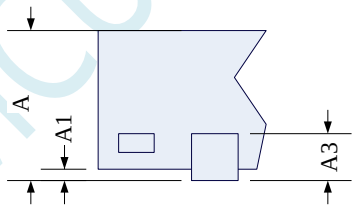
一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	-	-	1.20
A1	0.05	-	0.15
A2	0.90	1.00	1.05
A3	0.34	0.44	0.54
b	0.20	0.24	0.28
D	6.40	6.50	6.60
E	6.20	6.50	6.60
E1	4.30	4.40	4.50
e	0.65BSC		
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25BSC		
R1	0.09	-	-
R2	0.09	-	-

STCMCU

4.2 QFN20 封装尺寸图（3mm*3mm）

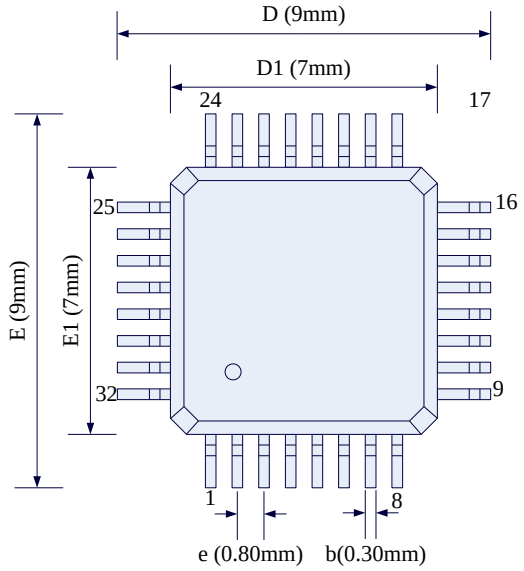


一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	0.70	0.75	0.80
A1	0	0.02	0.05
A2	0.50	0.55	0.60
A3	-	0.20REF	-
b	0.15	0.20	0.25
D	2.90	3.00	3.10
E	2.90	3.00	3.10
D2	1.40	1.50	1.60
E2	1.40	1.50	1.60
e	0.30	0.40	0.50
L	0.35	0.40	0.45
K	0.35REF		
R	0.085	-	-
C1	-	0.07	-
C2	-	0.07	-

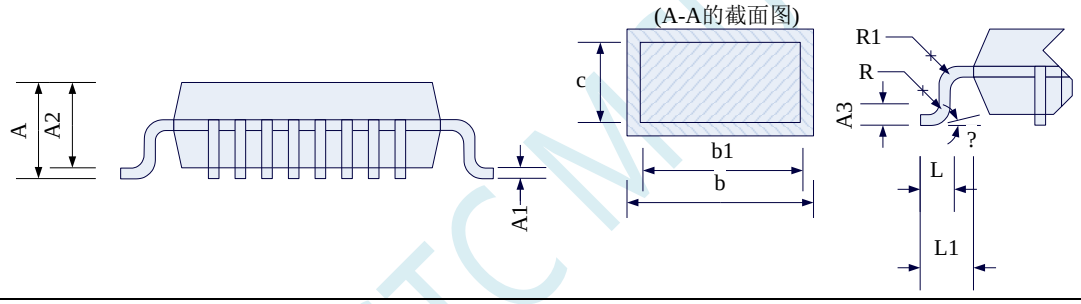


STC 现有 QFN20 封装芯片的背面金属片（衬底），在芯片内部并未接地，在用户的 PCB 板上可以接地，也可以不接地，不会对芯片性能造成影响

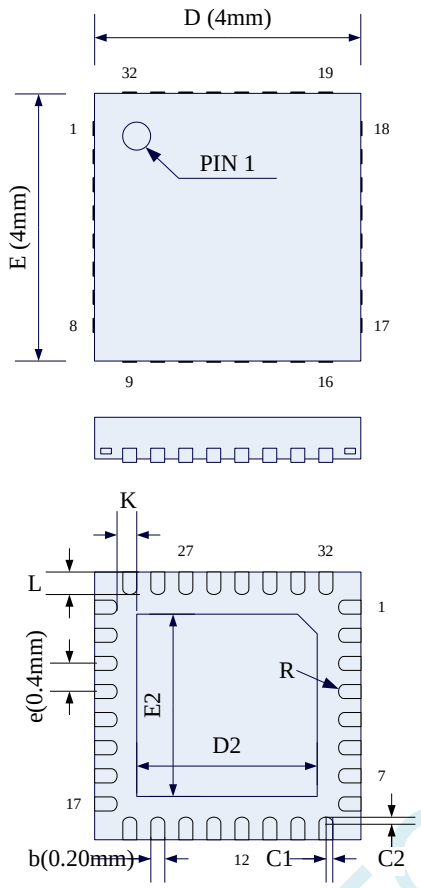
4.3 LQFP32 封装尺寸图（9mm*9mm）

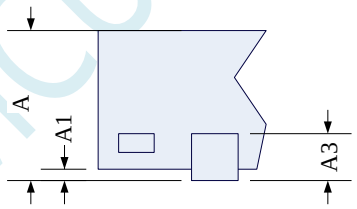


一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	1.45	1.55	1.65
A1	0.01	-	0.21
A2	1.35	1.40	1.45
A3	-	0.254	-
b1	0.30	0.35	0.40
b	0.31	0.37	0.43
c	-	0.127	-
D	8.80	9.00	9.20
D1	6.90	7.00	7.10
E	8.80	9.00	9.20
E1	6.90	7.00	7.10
e	0.70	0.80	0.90
L	0.43	-	0.71
L	1.00REF		
L1	0.25BSC		
R	0.1	-	0.25
R1	0.1	-	-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



4.4 QFN32 封装尺寸图（4mm*4mm）

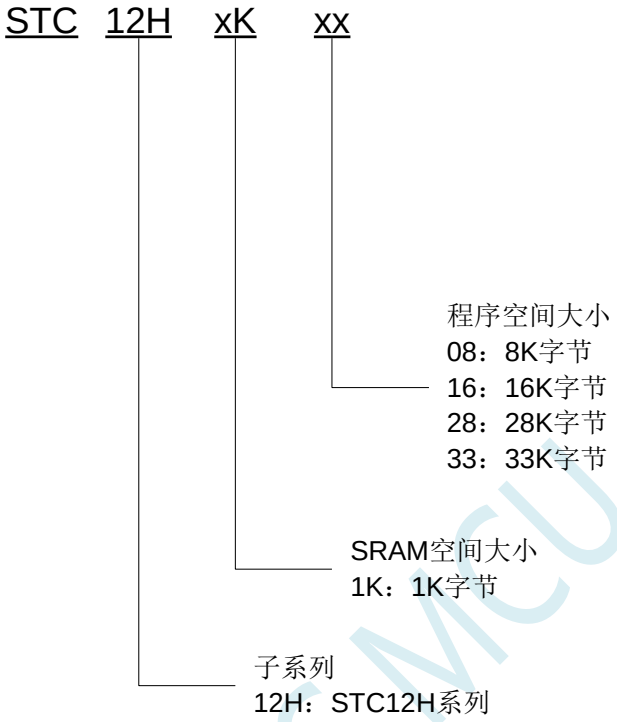




一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	0.70	0.75	0.80
A1	0	0.02	0.05
A2	0.50	0.55	0.60
A3	-	0.20REF	-
b	0.15	0.20	0.25
D	3.90	4.00	4.10
E	3.90	4.00	4.10
D2	2.60	2.70	2.80
E2	2.60	2.70	2.80
e	0.30	0.40	0.50
L	0.35	0.40	0.45
K	0.25REF		
R	0.09	-	-
C1	-	0.16	-
C2	-	0.16	-

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

4.5 STC12H 系列单片机命名规则

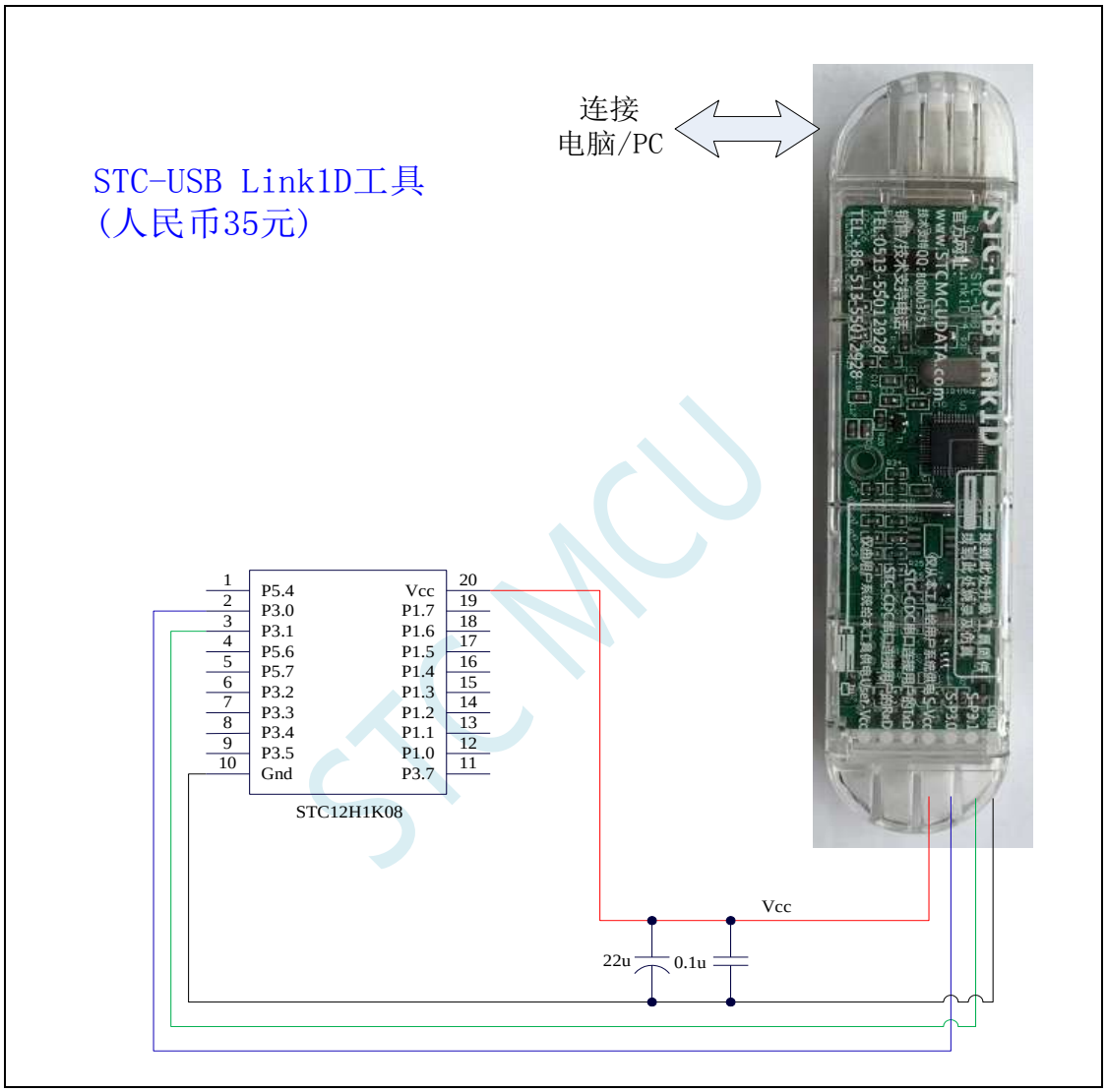


STC MCU

5 ISP 下载及典型应用线路图

5.1 STC12H 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：

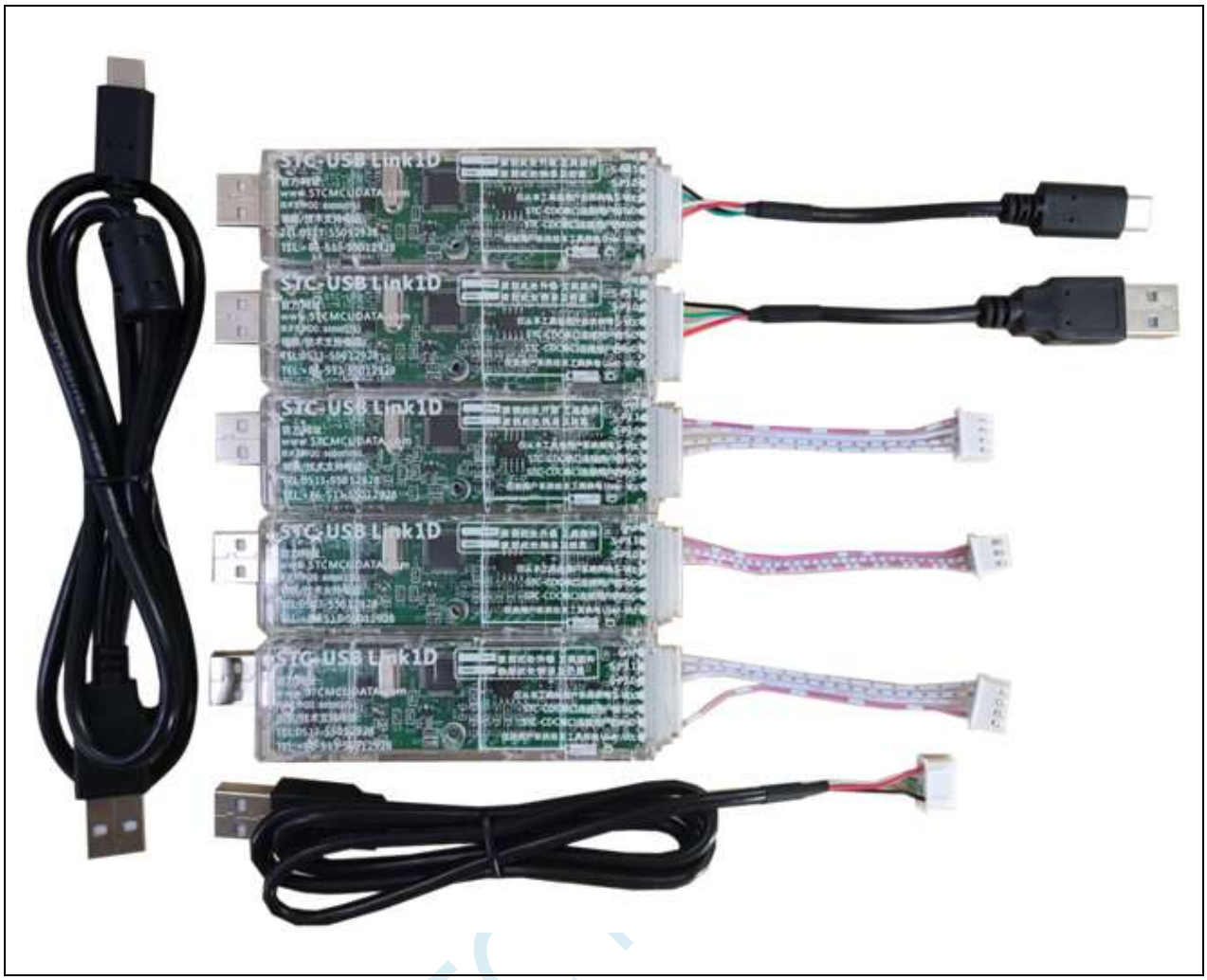
1、 按照如图所示的连接方式将 STC-USB Link1D 和目标芯片连接

2、 点击 STC-ISP 下载软件中的“下载/编程”按钮

3、 开始 ISP 下载

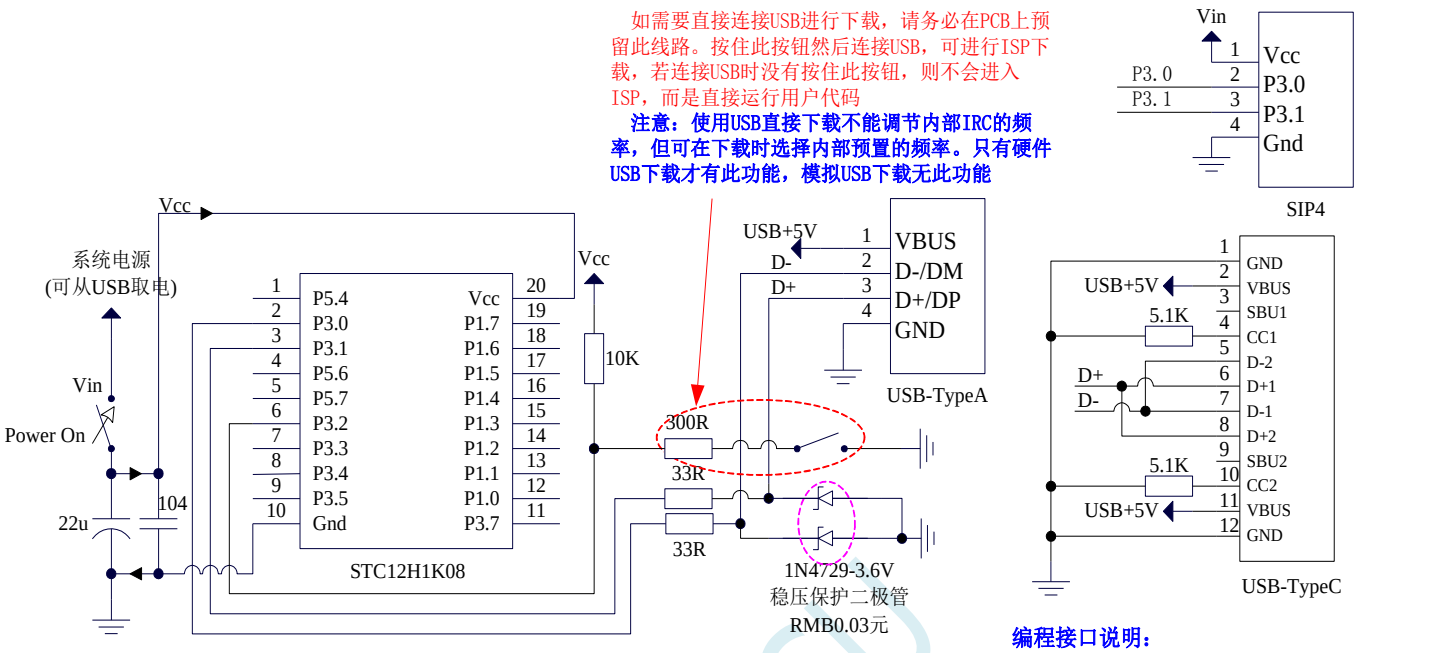
注意：若是使用 STC-USB Link1D 给目标系统供电，目标系统的总电流不能大于 200mA，否则会导致下载失败。

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。

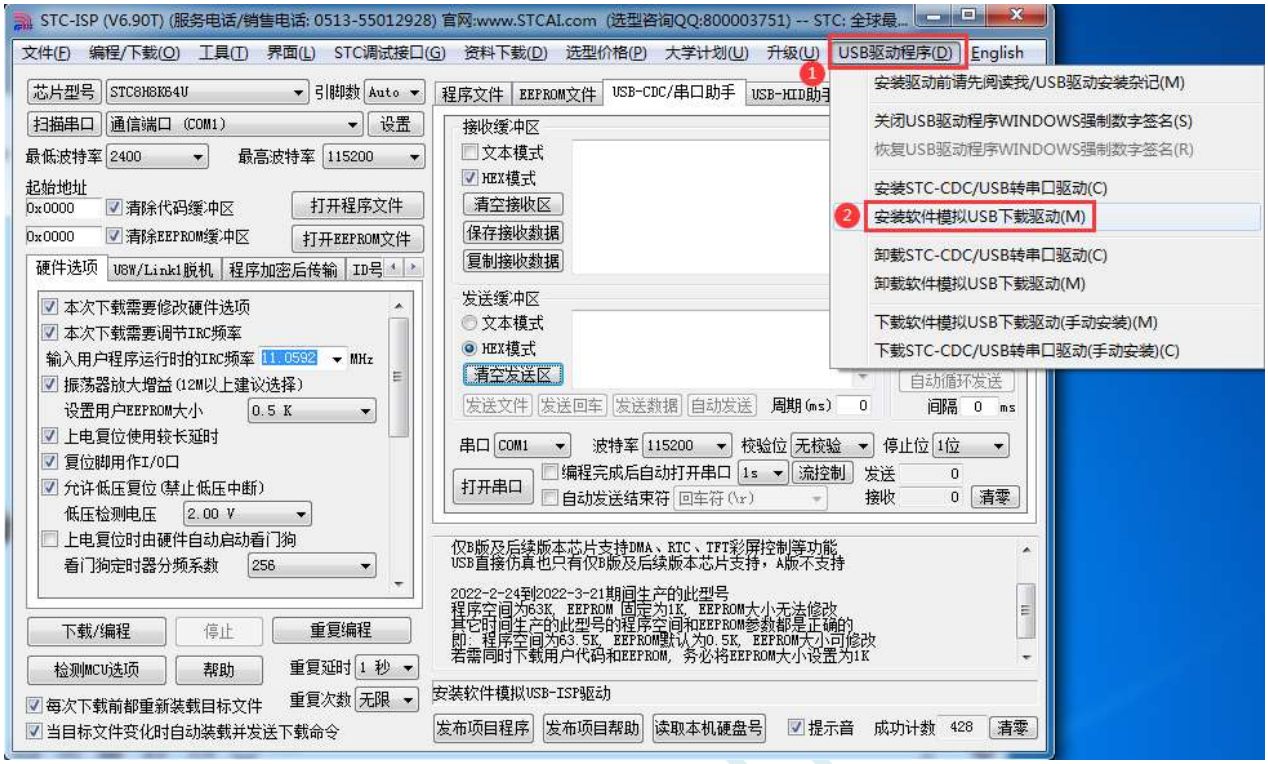


上面 **RMB35** 是配上面全部的线，是亏本补助大家的

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：

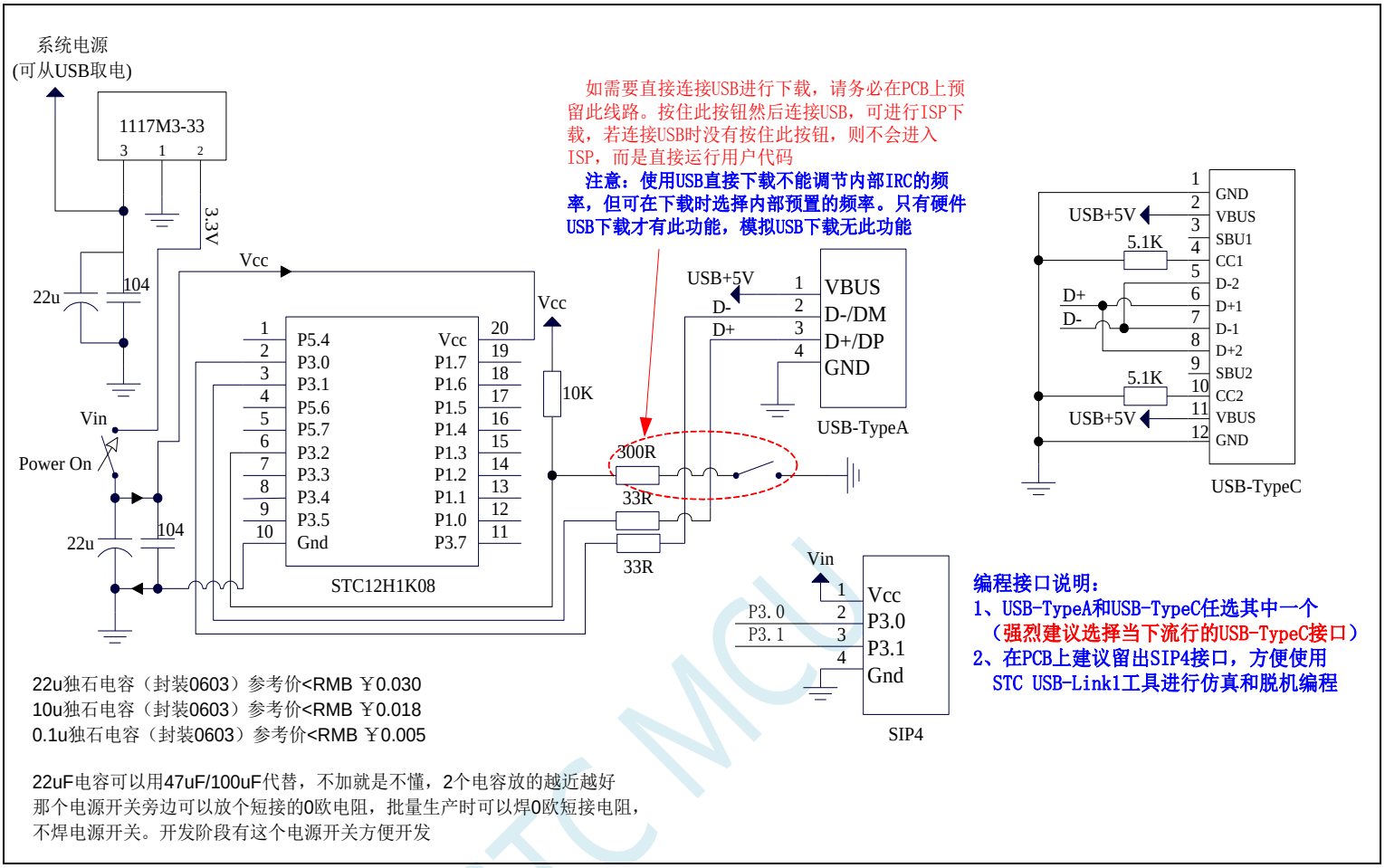
1、D-/P3.0, D+/P3.1 与 PC-USB 端口连接好

2、将 P3.2 与 GND 短接，实验箱板子上的 P3.2/INT0 按键按下

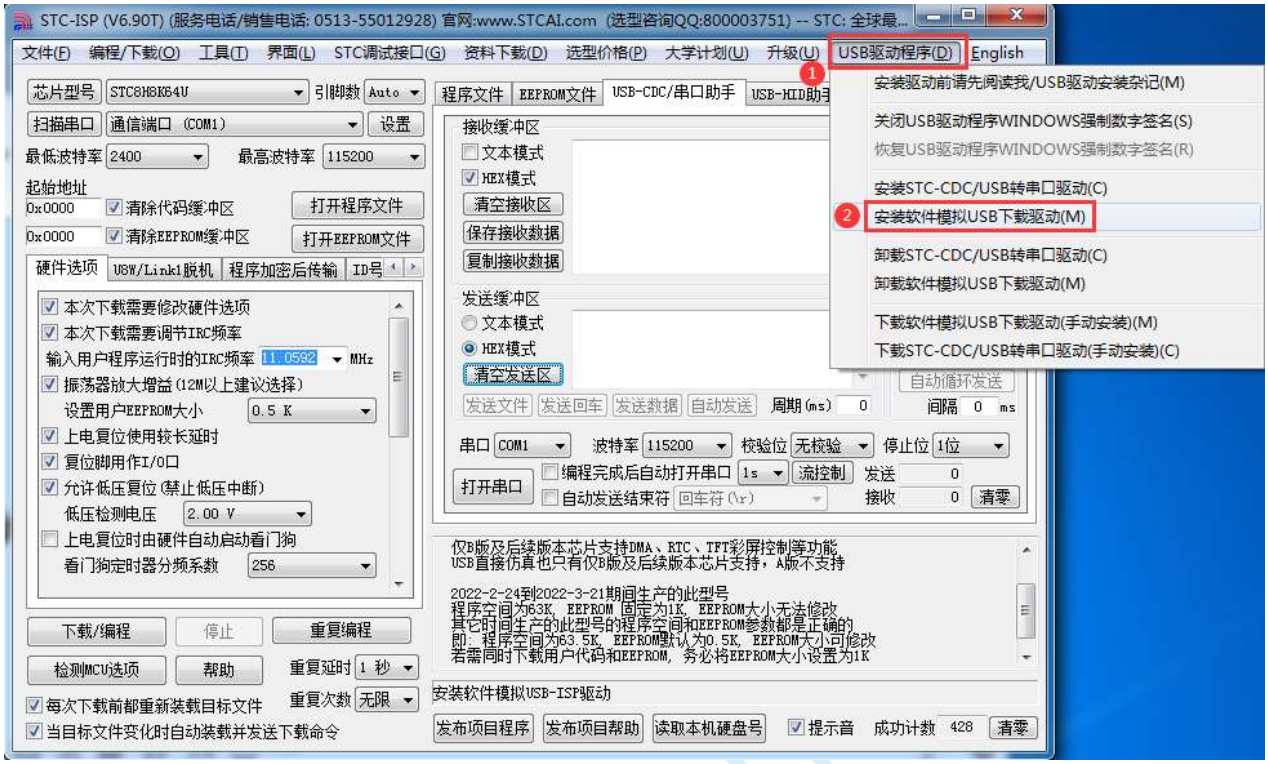
3、给目标芯片重新上电。若目标芯片已经停电，直接上电即可；若目标芯片处于通电状态，则需给目标芯片停电再上电（冷启动）。等待 STC-ISP 下载软件中自动识别出“STC USB Writer (HID1)”识别出来后,就与 P3.2 状态无关了（此时就不需要一直按着 P3.2 口了，一直按着把手累坏了不要紧，将按键按坏就麻烦了）。

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：

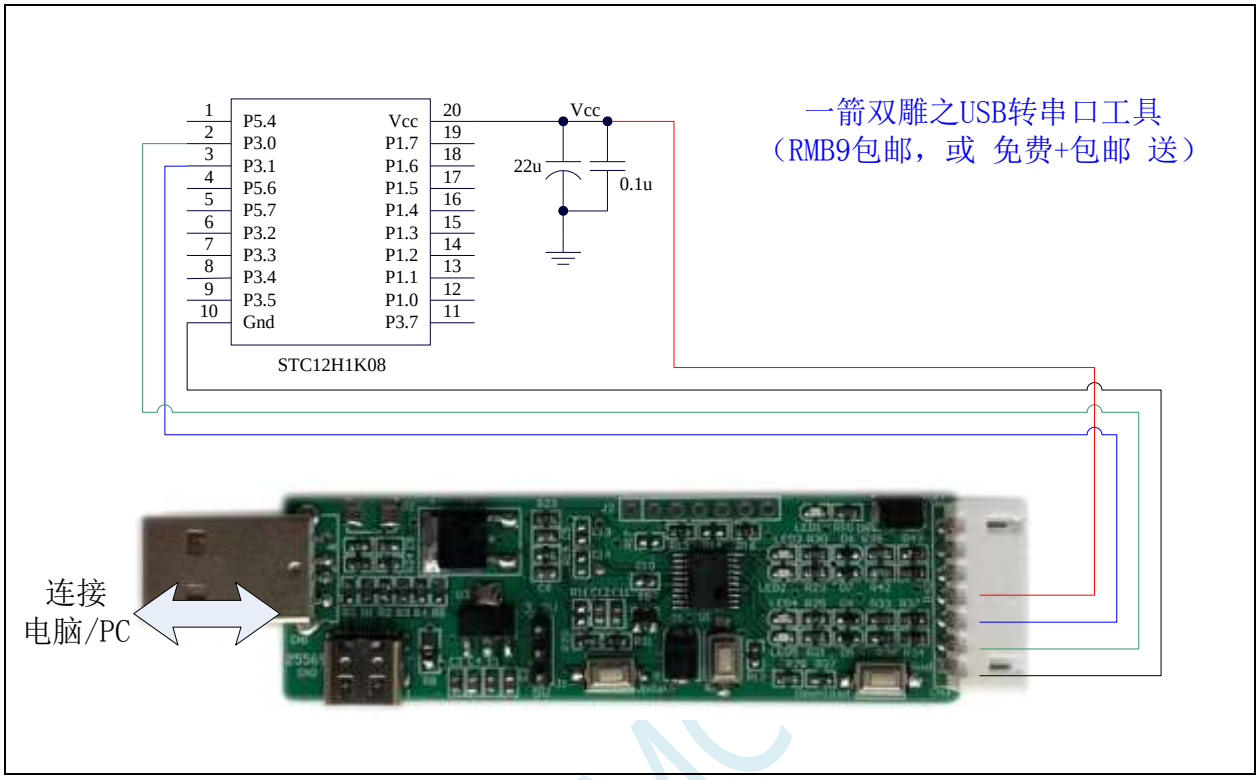
1、D-/P3.0, D+/P3.1 与 PC-USB 端口连接好

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

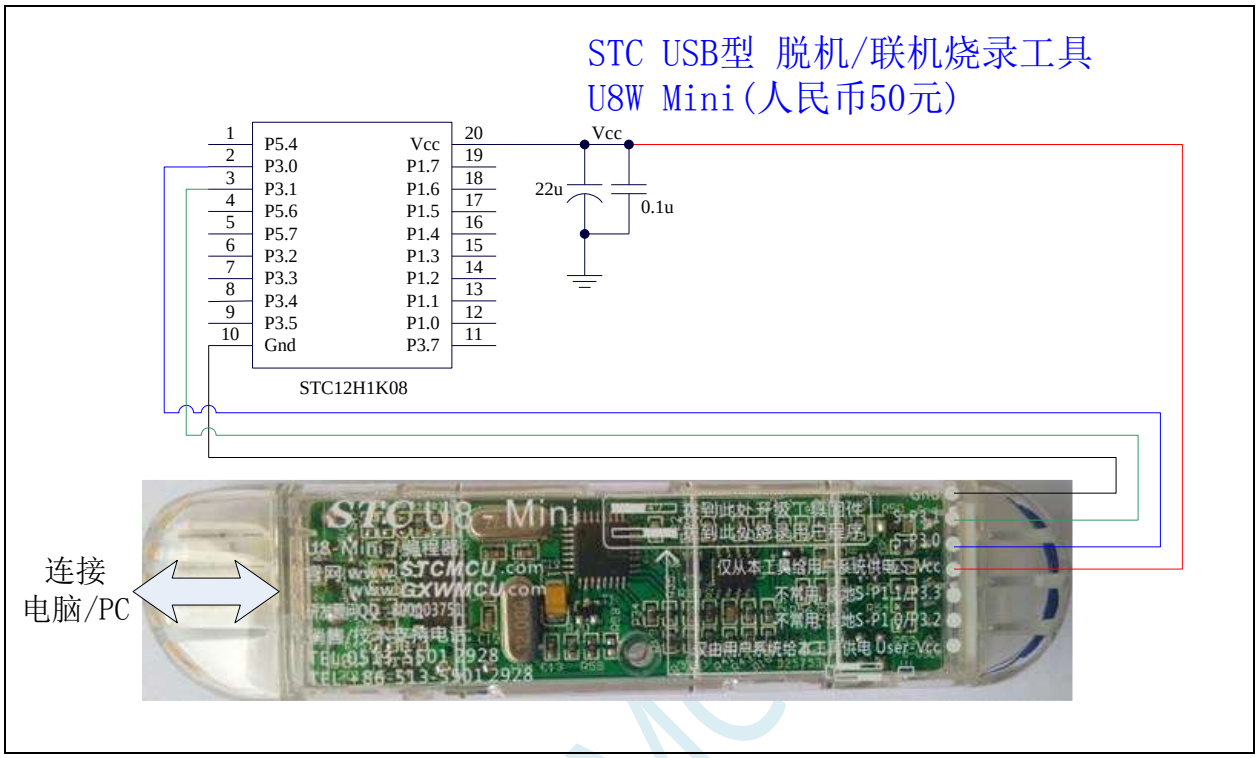
2、 点击 STC-ISP 下载软件中的“下载/编程”按钮

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：

4、 按照如图所示的连接方式将 U8-Mini 和目标芯片连接

5、 点击 STC-ISP 下载软件中的“下载/编程”按钮

6、 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。

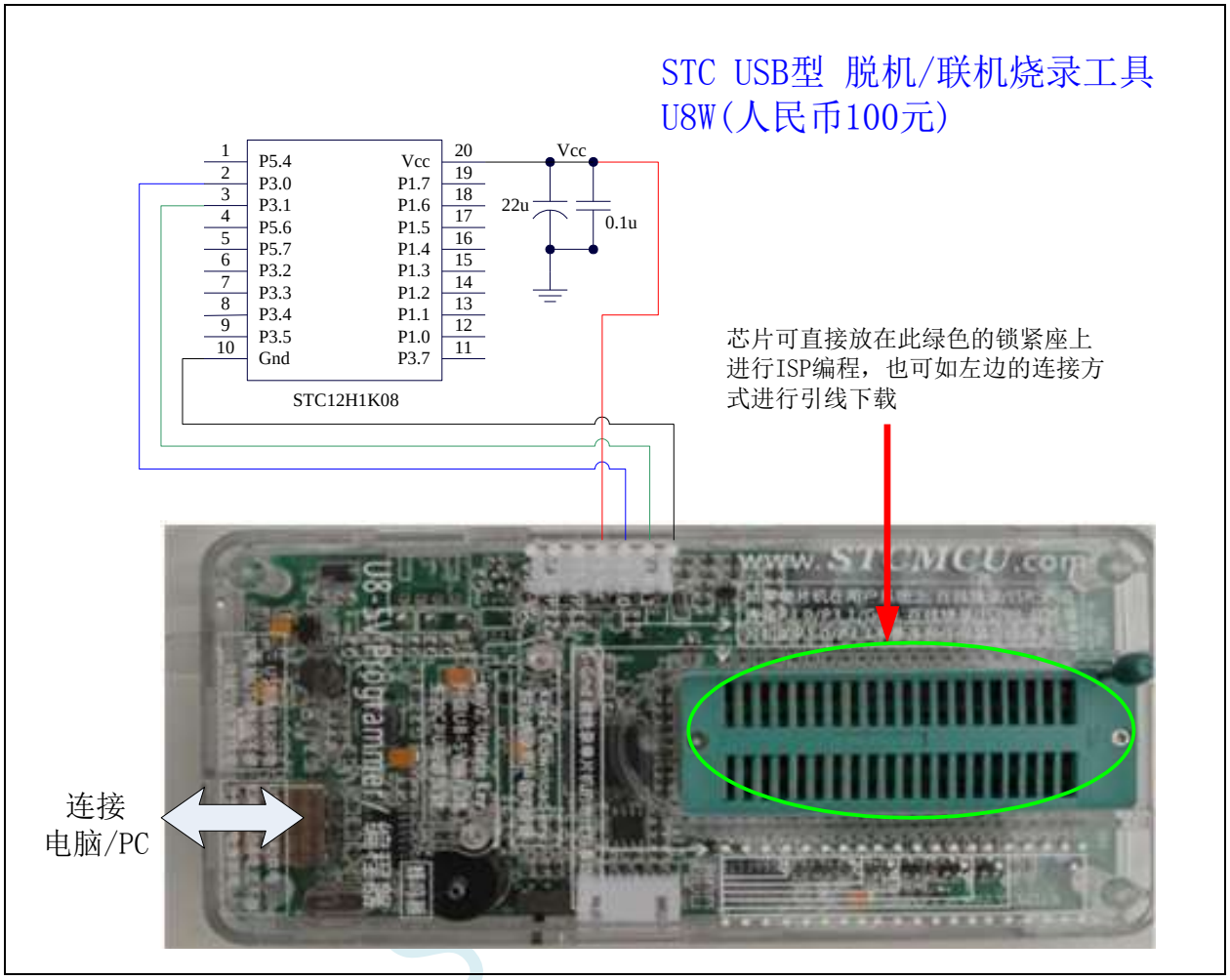
若要使用 U8-Mini 进行仿真，首先必须将 U8-Mini 设置为直通模式。U8W/U8W-Mini 实现 USB 转串口直通模式的方法如下：

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

5.1.6 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：

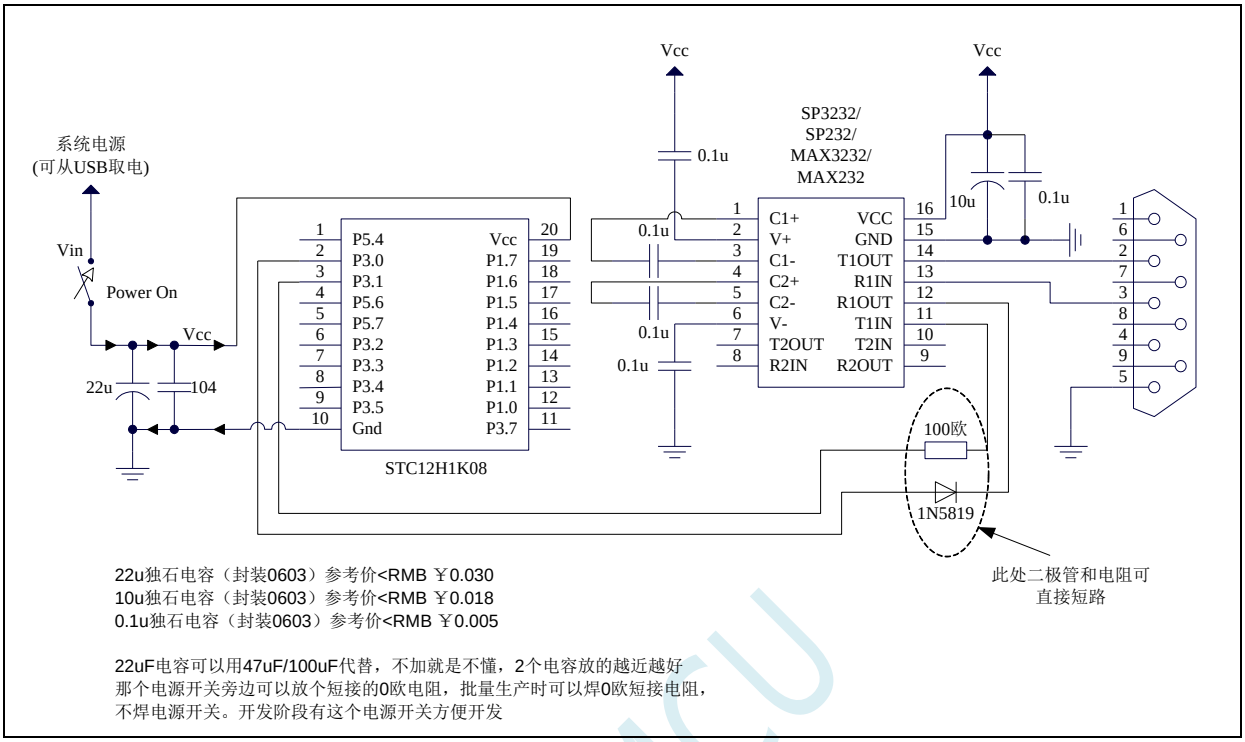
1、 首先 U8W/U8W-Mini 固件必须升级到 v1.37 及以上版本

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

STC MCU

5.1.7 使用 RS-232 转换器下载，也可支持仿真



ISP 下载步骤：

1、 给目标芯片停电

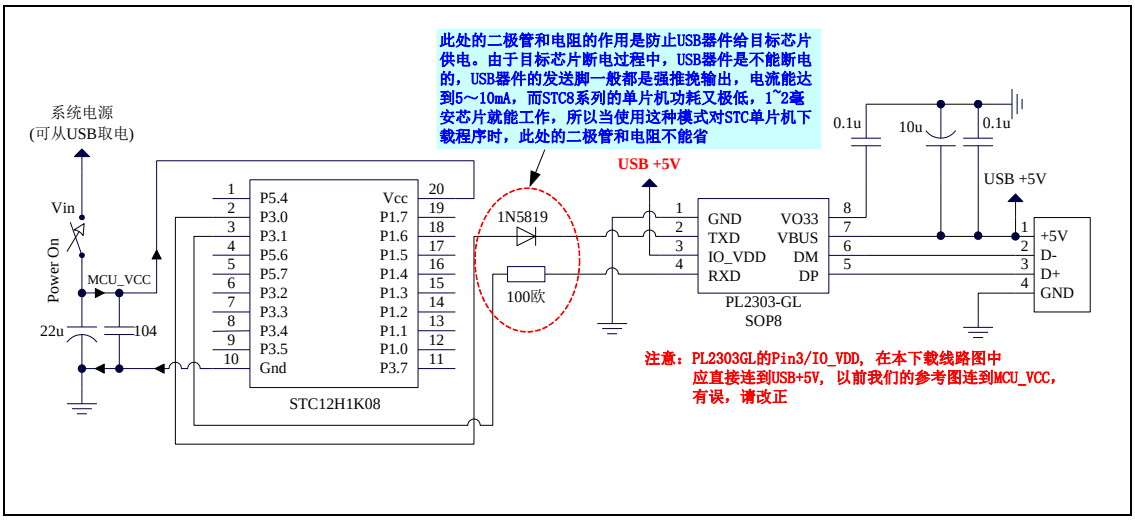
2、 点击 STC-ISP 下载软件中的“下载/编程”按钮

3、 给目标芯片上电

4、 开始 ISP 下载

STC MCU

5.1.8 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）

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

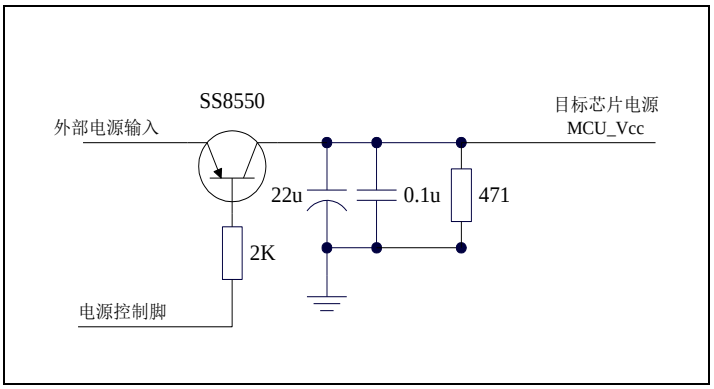
4、 给目标芯片上电

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。

STCMCU

5.1.9 单片机电源控制参考电路



STC MCU

5.2　STC-ISP 下载软件高级应用

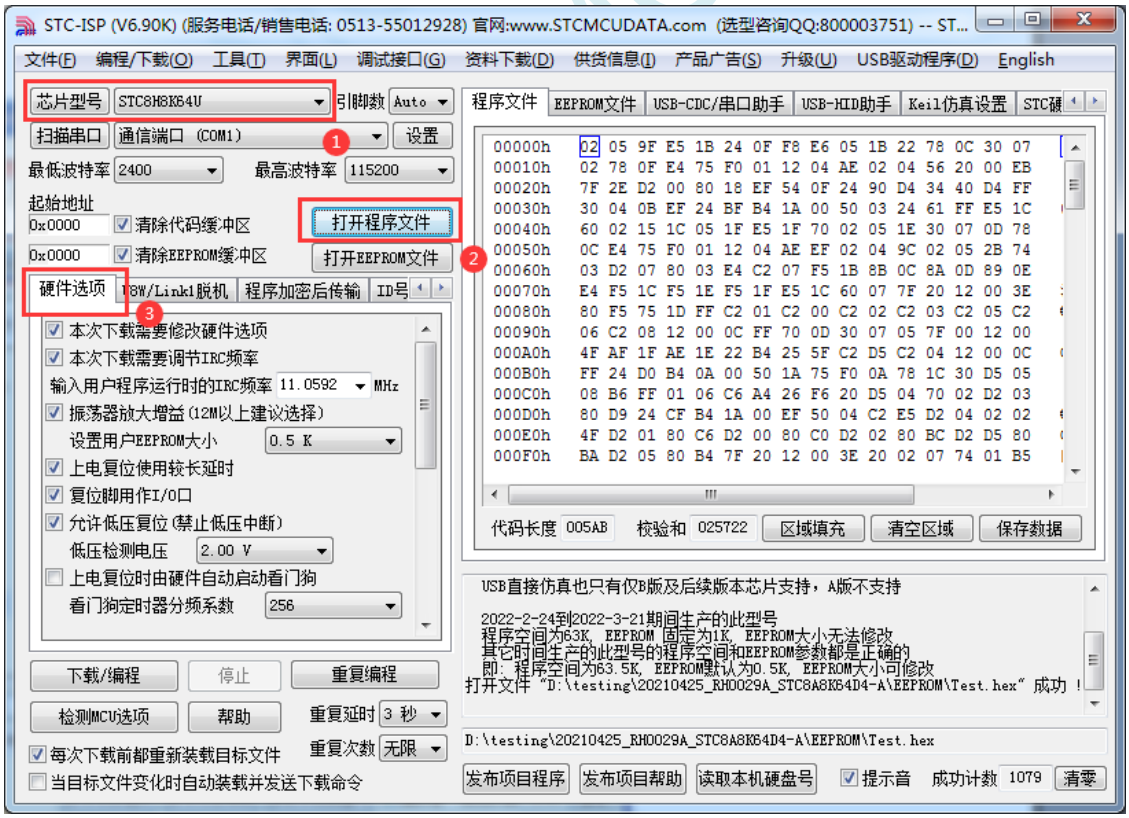
5.2.1　发布项目程序

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。

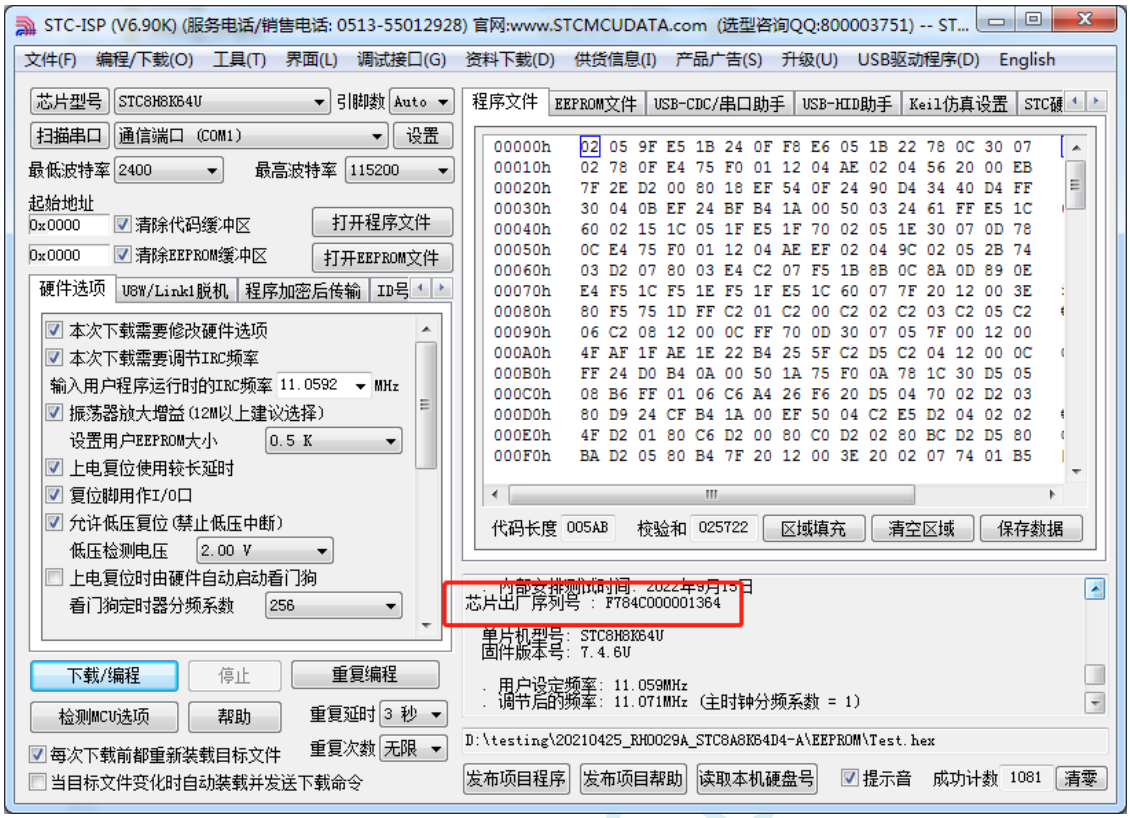
关于界面，用户可以自己进行定制（用户可以自行修改发布项目程序的标题、按钮名称以及帮助信息），同时用户还可以指定目标电脑的硬盘号和目标芯片的 ID 号，指定目标电脑的硬盘号后，便可以控制发布应用程序只能在指定的电脑上运行（防止烧录人员将程序轻易从电脑盗走,如通过网络发走,如通过 U 盘烤走,防不胜防,当然盗走你的电脑那就没办法那,所以 STC 的脱机下载工具比电脑烧录安全,能限制可烧录芯片数量,让前台文员小姐烧,让老板娘烧都可以），拷贝到其它电脑，应用程序不能运行。同样的，当指定了目标芯片的 ID 号后，那么用户代码只能下载到具有相应 ID 号的目标芯片中（对于一台设备要卖几千万的产品特别有用---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

发布项目程序详细的操作步骤如下：

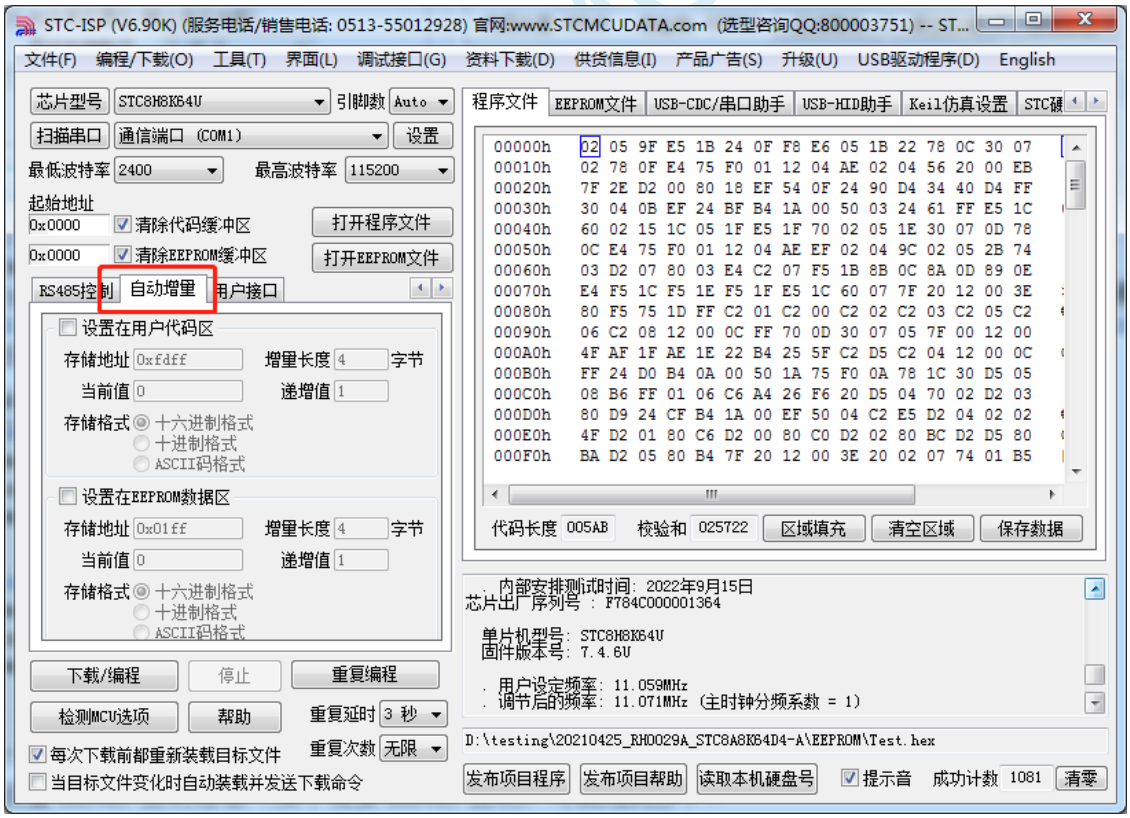
1、首先选择目标芯片的型号
2、打开程序代码文件
3、设置好相应的硬件选项



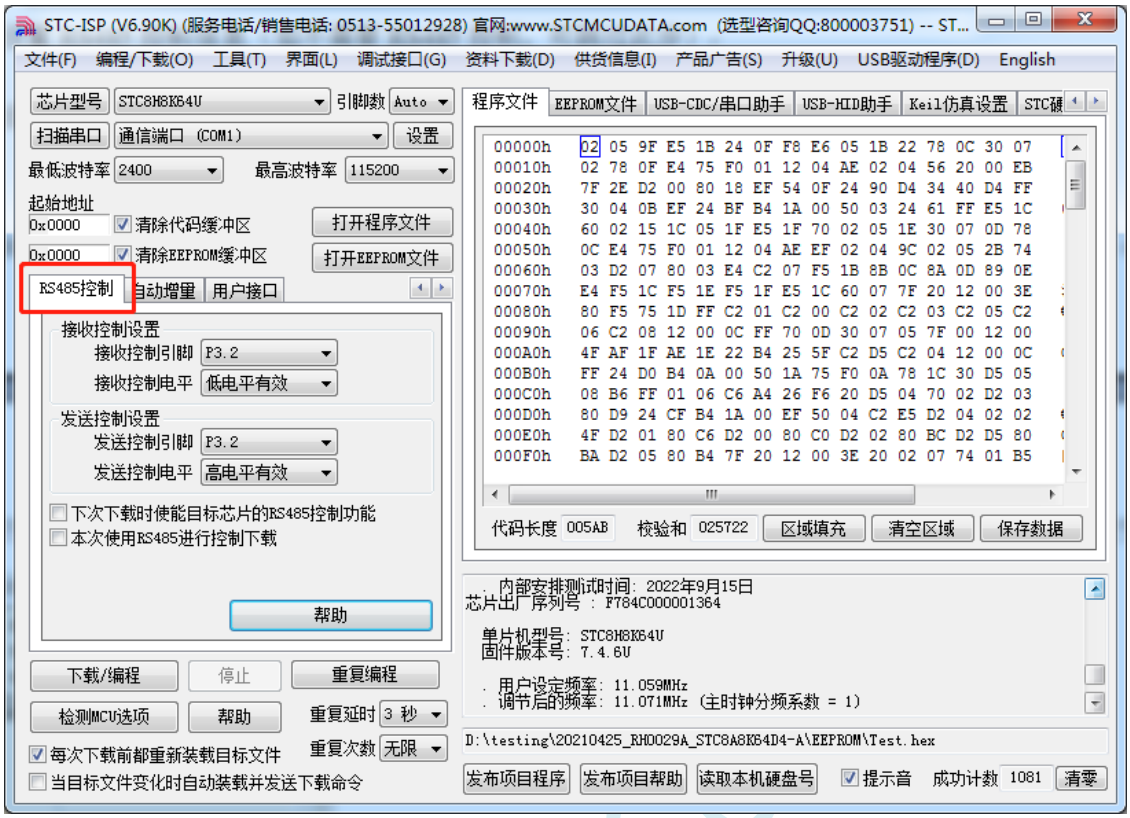
4、试烧一下芯片，并记下目标芯片的 ID 号，如下图所示，该芯片的 ID 号即为“F784C000001364”（如不需要对目标芯片的 ID 号进行校验，可跳过此步）



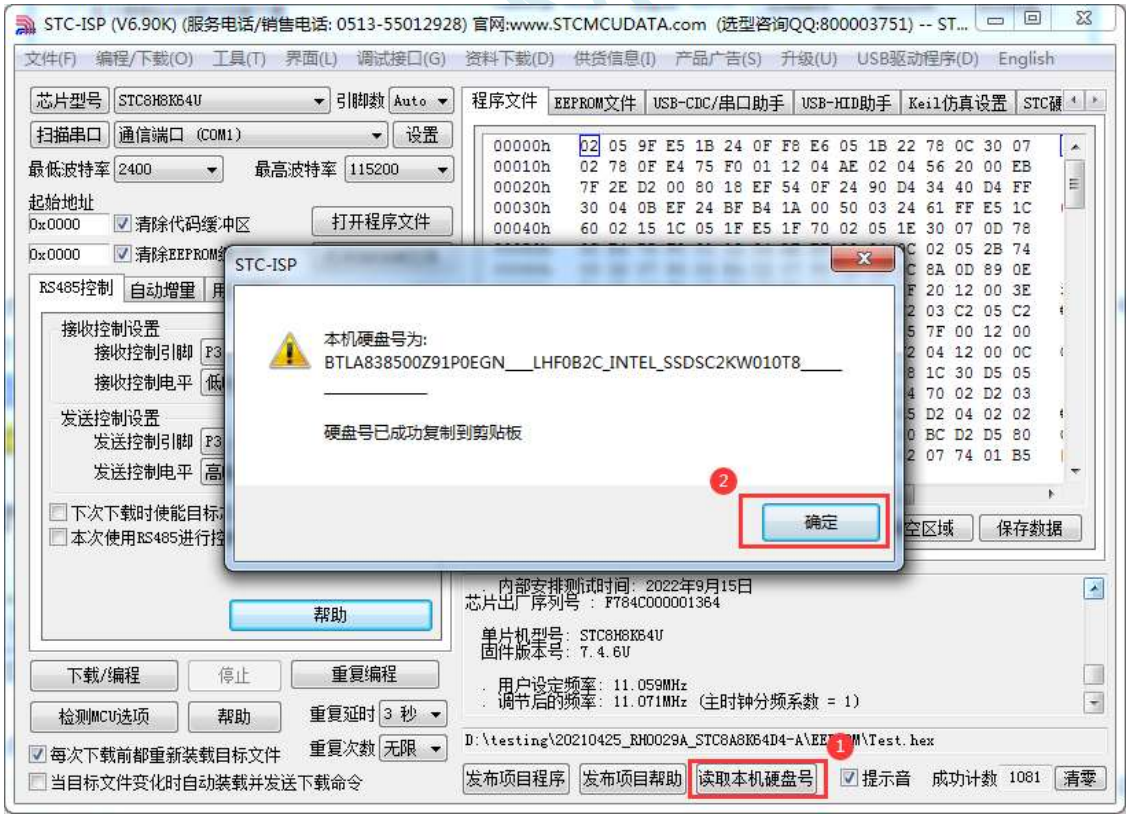
5、设置自动增量（如不需要自动增量，可跳过此步）



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）



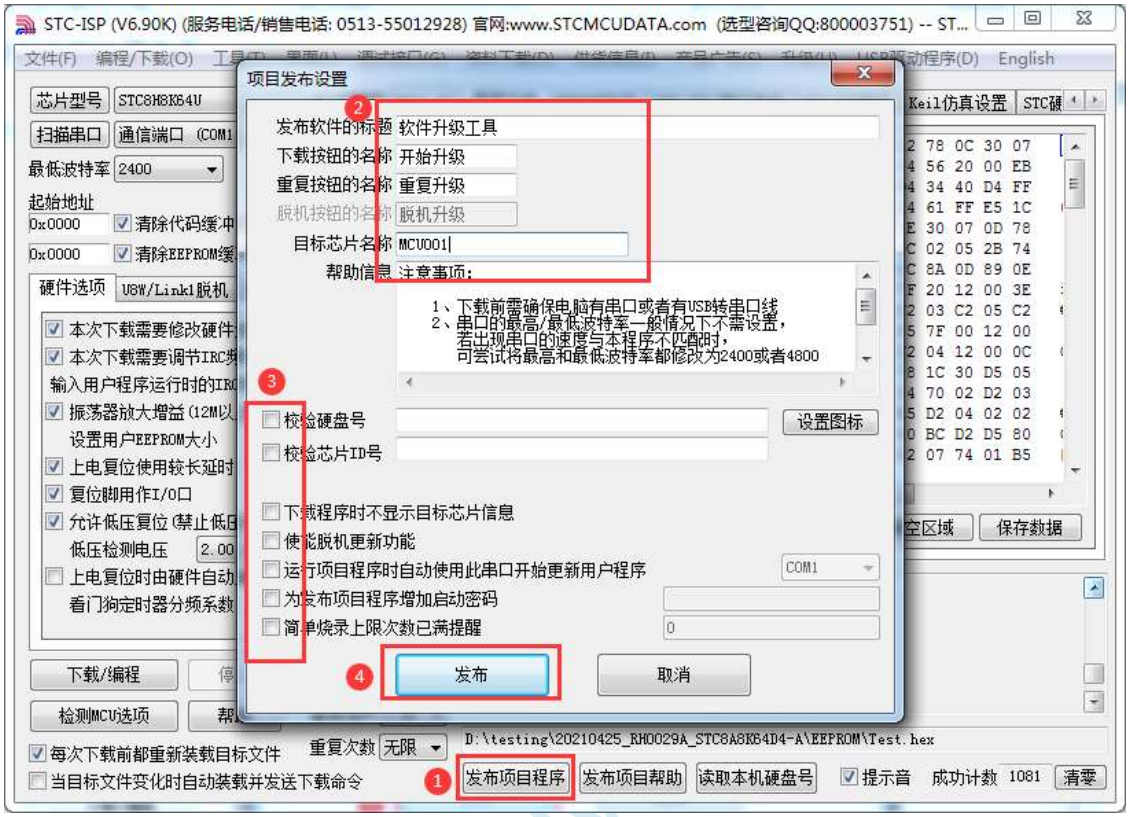
8、点击“发布项目程序”按钮，进入发布应用程序的设置界面。

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。发布的项目程序打界面如下图

5.2.2　程序加密后传输（防烧录时串口分析出程序）

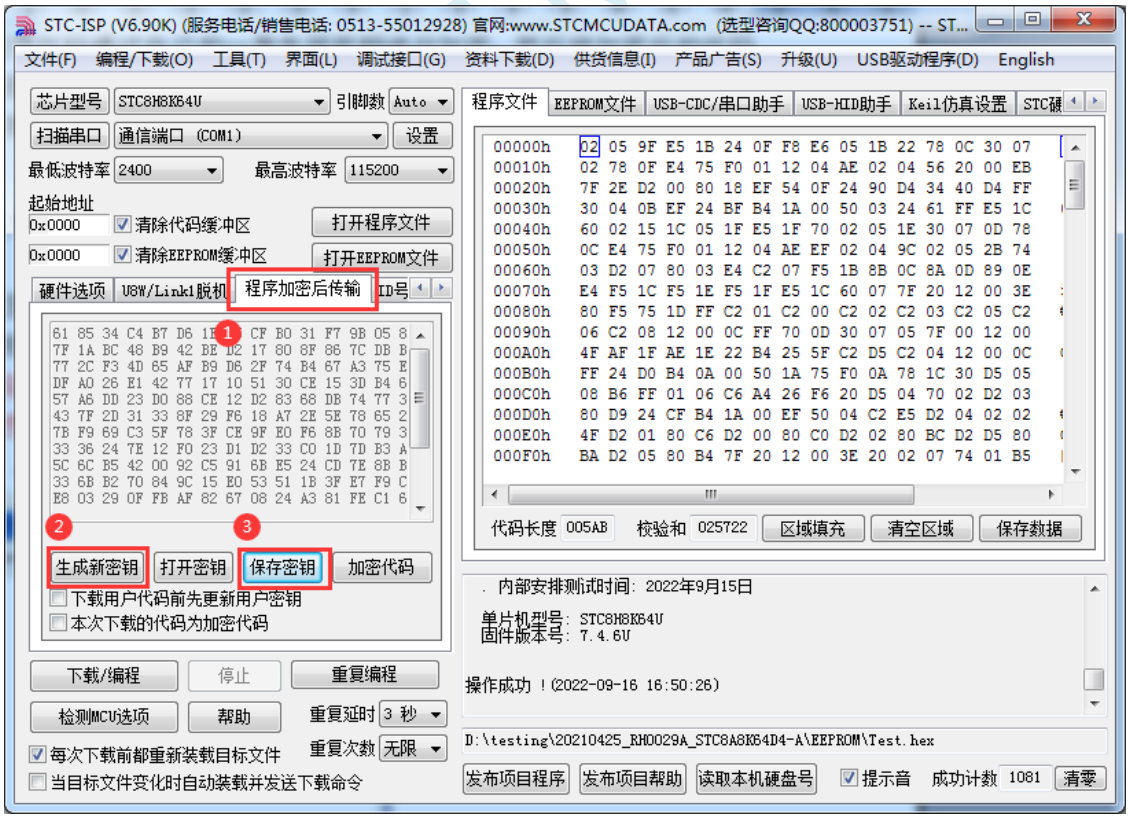
目前，所有的普通串口下载烧录编程都是采用明码通信的（电脑和目标芯片通信时，或脱机下载板和目标芯片通信时），问题：如果烧录人员通过分析下载烧录编程时串口通信的数据，高手是可以在烧录时在串口上引 2 根线出来，通过分析串口通信的数据分析出实际的用户程序代码的。当然用 STC 的脱机下载板烧程序总比用电脑烧程序强（防止烧录人员将程序轻易从电脑盗走，如通过网络发走，如通过 U 盘烤走，防不胜防，当然盗走你的电脑那就没办法那，所以 STC 的脱机下载工具比电脑烧录安全，让前台文员小姐烧，让老板娘烧都可以）。即使是 STC 全球首创的脱机下载工具，对于要防止天才的不法分子在脱机下载工具烧录的过程中通过分析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

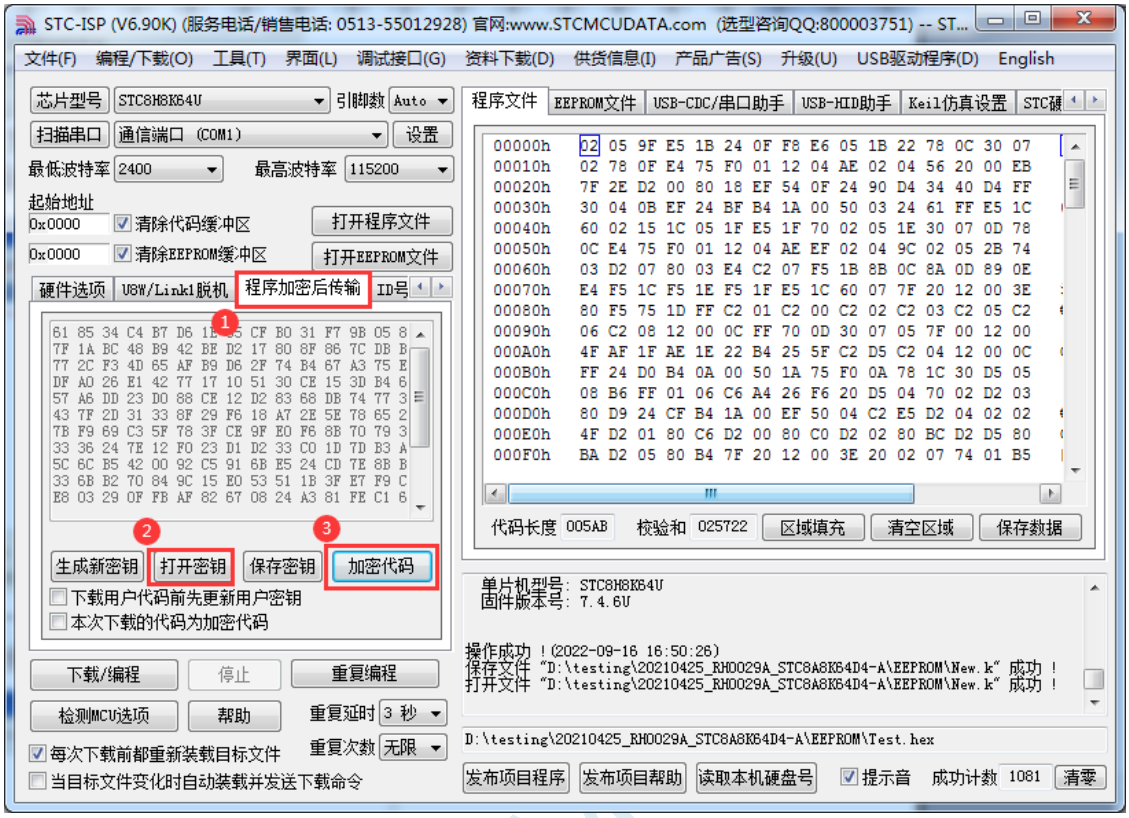
如下图，进入到“程序加密后传输”页面，点击“生成新密钥”按钮，即可在缓冲区显示新生成的 256 字节的密钥。然后点击“保存密钥”按钮，即可将生成的新密钥保存为以“.K”为扩展名的的密钥文件（**注意：这个密钥文件一定要保存好，以后发布的代码文件都需要使用这个密钥加密，而且这个密钥的生成是非重复的，即任何时候都不可能生成两个完全相同的密钥，所以一旦密钥文件丢失将无法重新获得**）。例如我们将密钥保存为“New.k”。



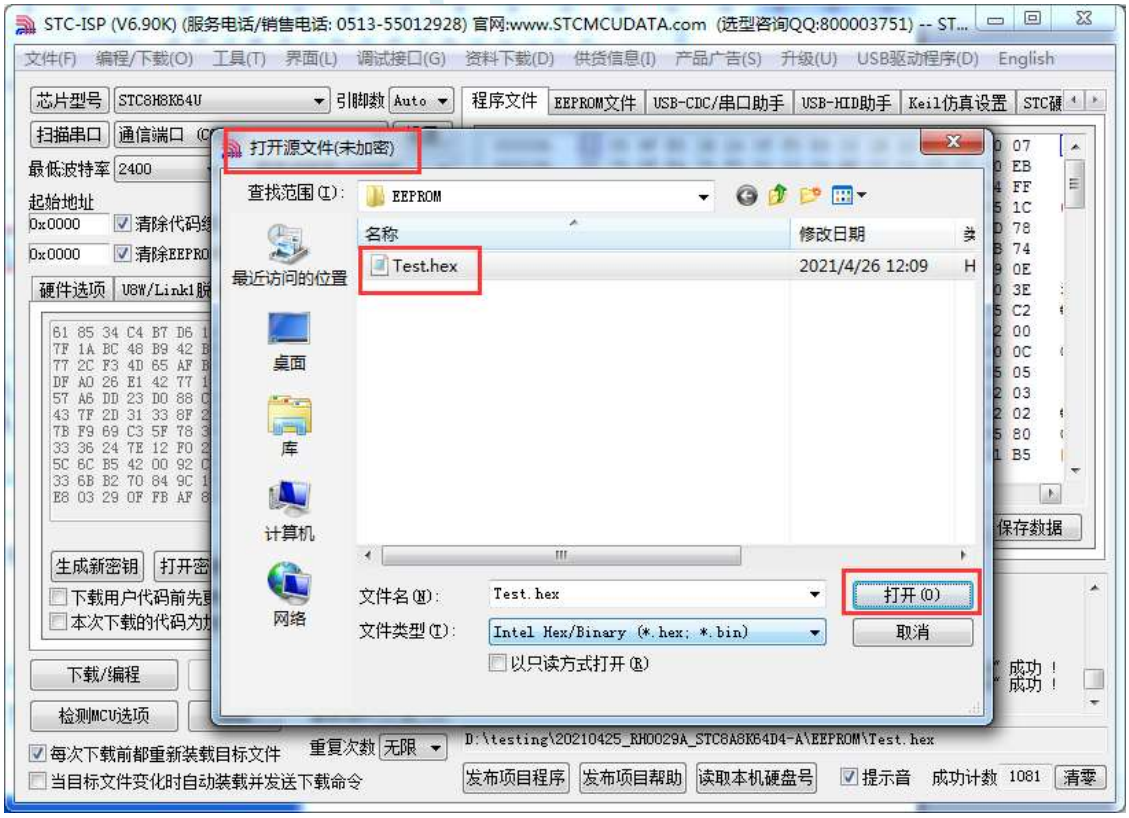
2、对代码文件加密

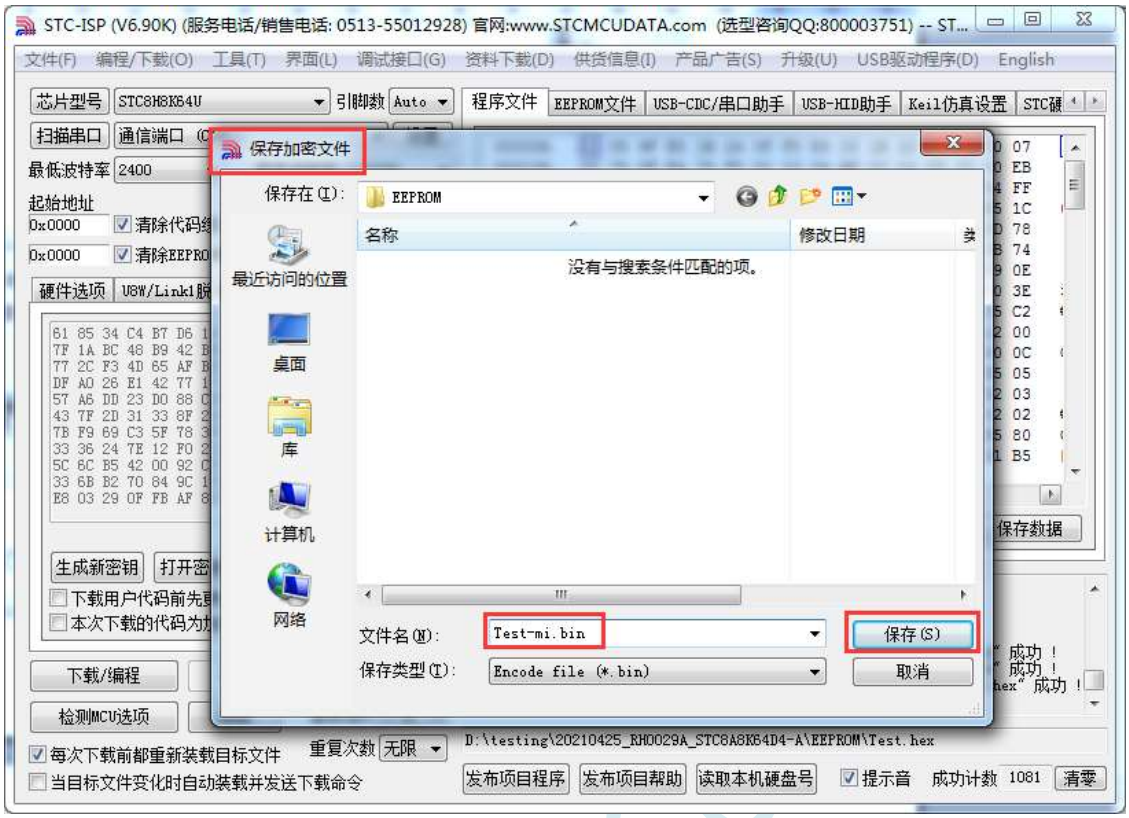
加密文件前，需要先打开我们自己的密钥。若缓冲区中存放的已经是我们的密钥，则不要再打开。如下图，在“程序加密后传输”页面中点击“打开密钥”按钮，打开我们之前保存的密钥文件，

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



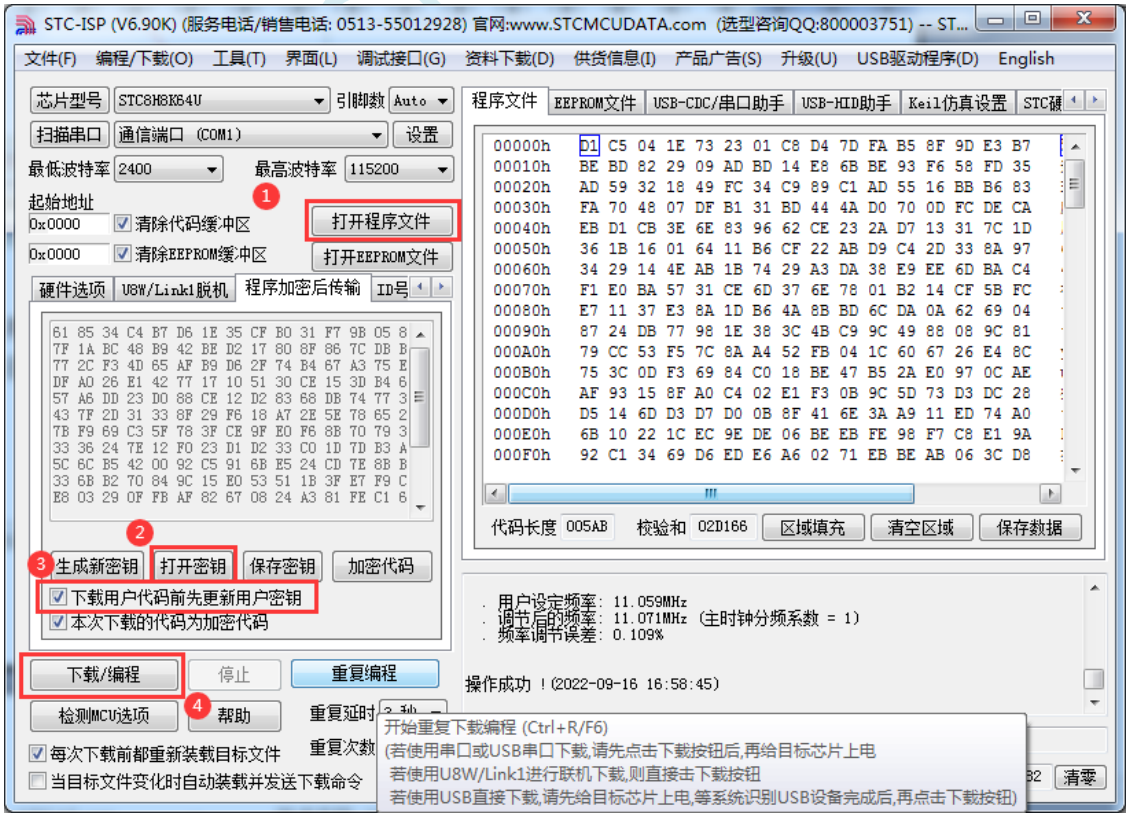
点击打开按钮后，马上有会弹出一个类似的对话框，但此时是对加密后的文件进行保存的对话框。如下图所示，点击保存按钮即可保存加密后的文件。





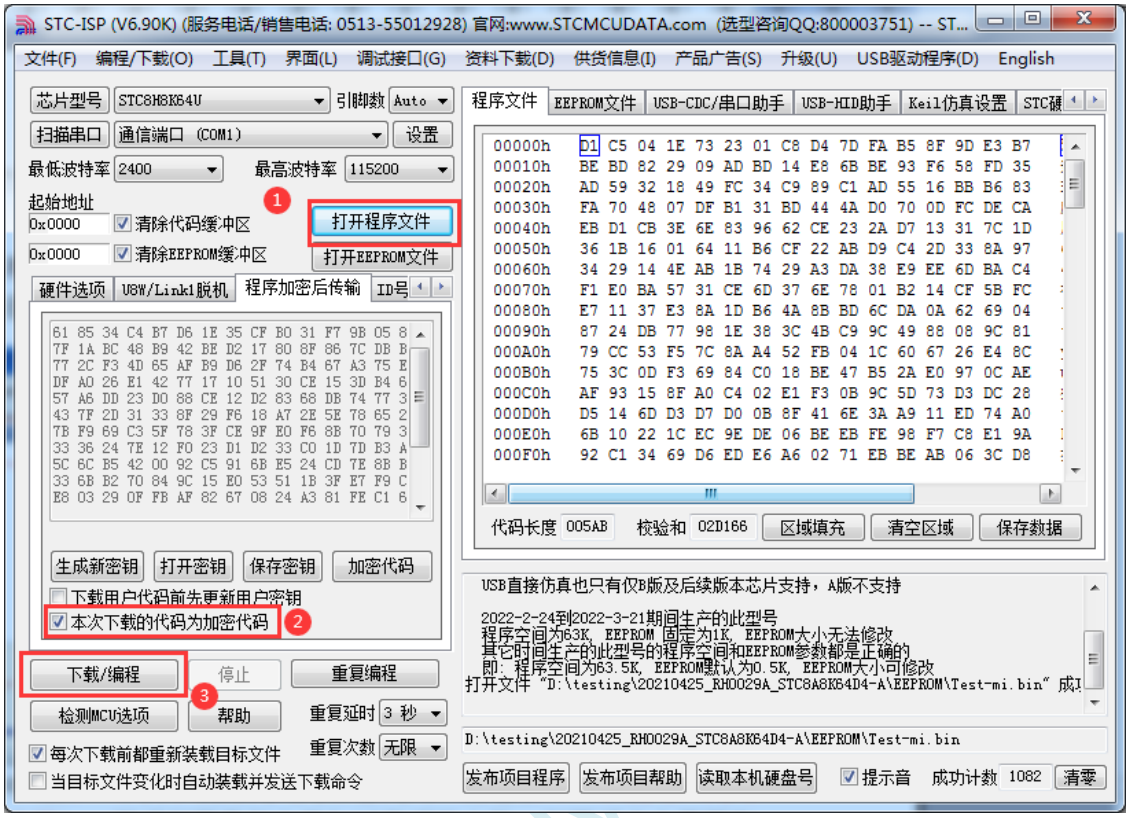
3、将用户密钥更新到目标芯片中

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



4、加密更新用户代码

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



对于一片新的 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）。

5.2.3 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。

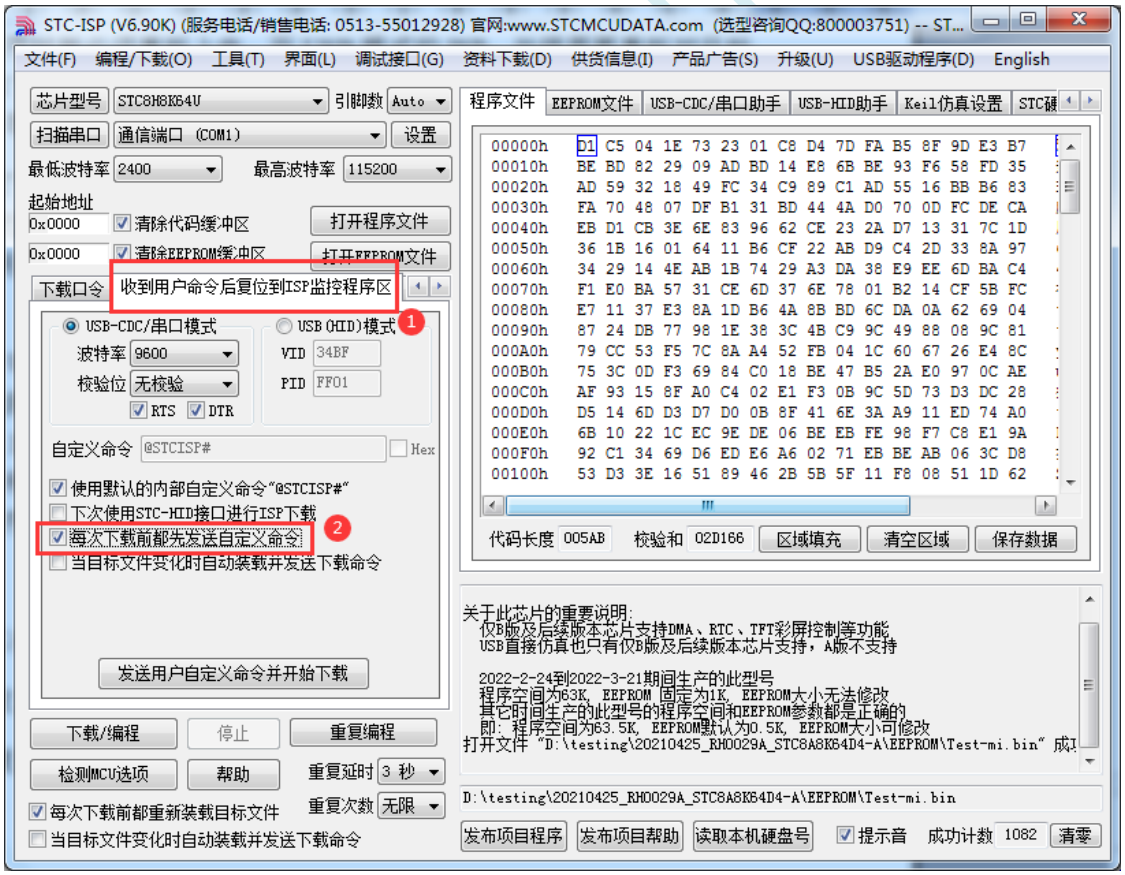
STC MCU

5.2.4　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。



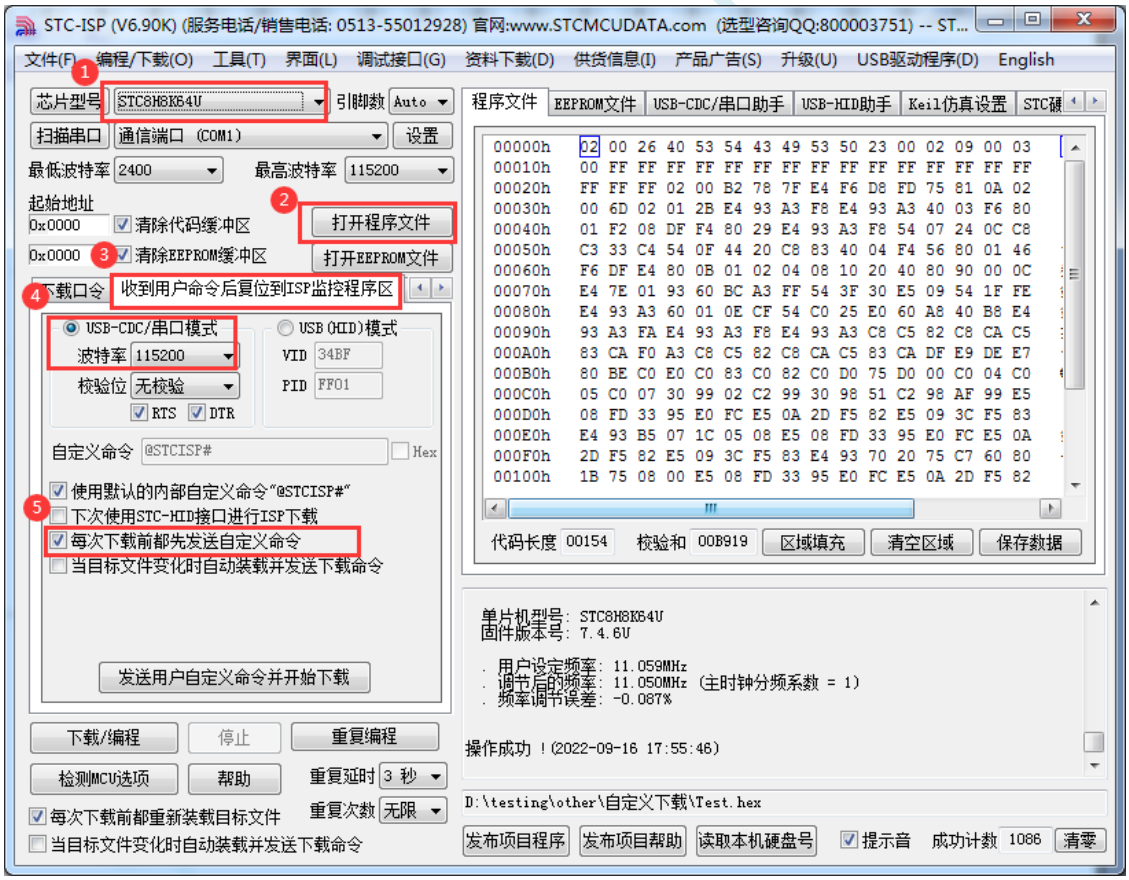
实现不停电 ISP 下载的步骤如下：

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;
```

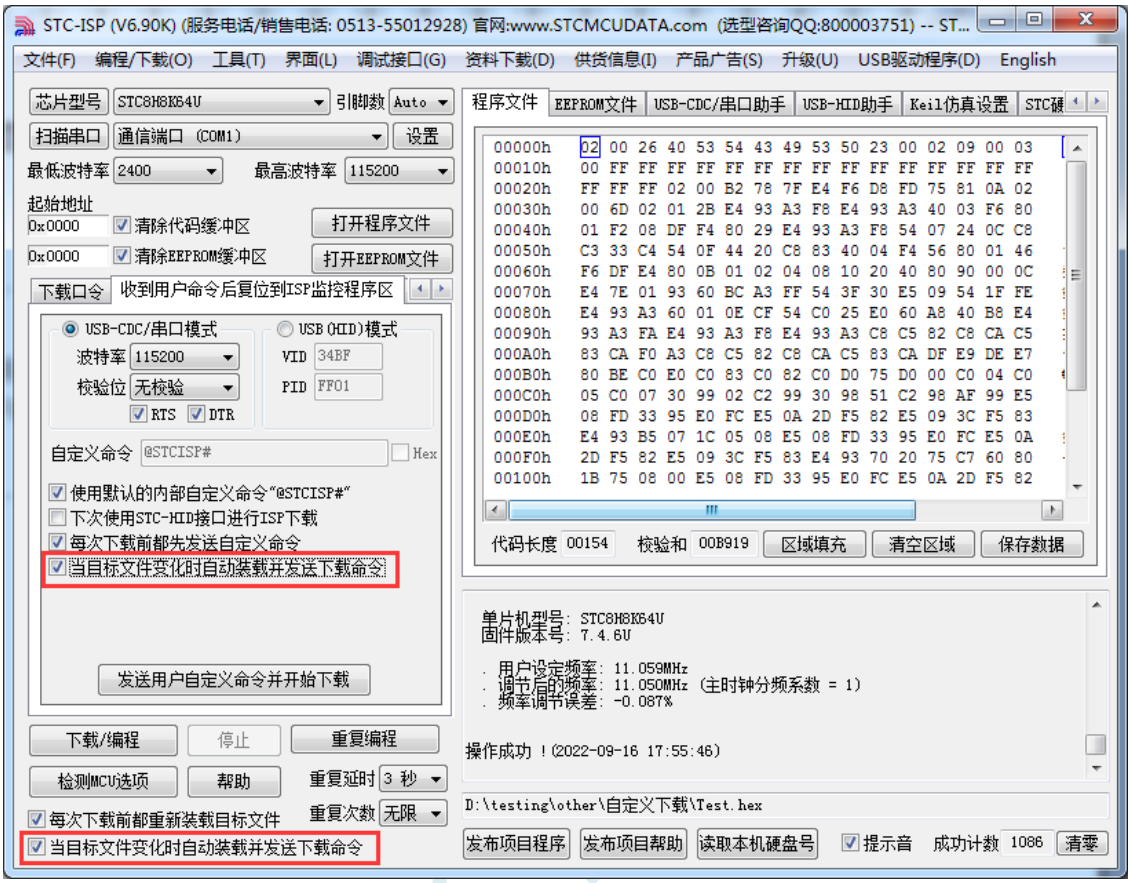
```
        P2M0 = 0x00; P2M1 = 0x00;
        P3M0 = 0x00; P3M1 = 0x00;

        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

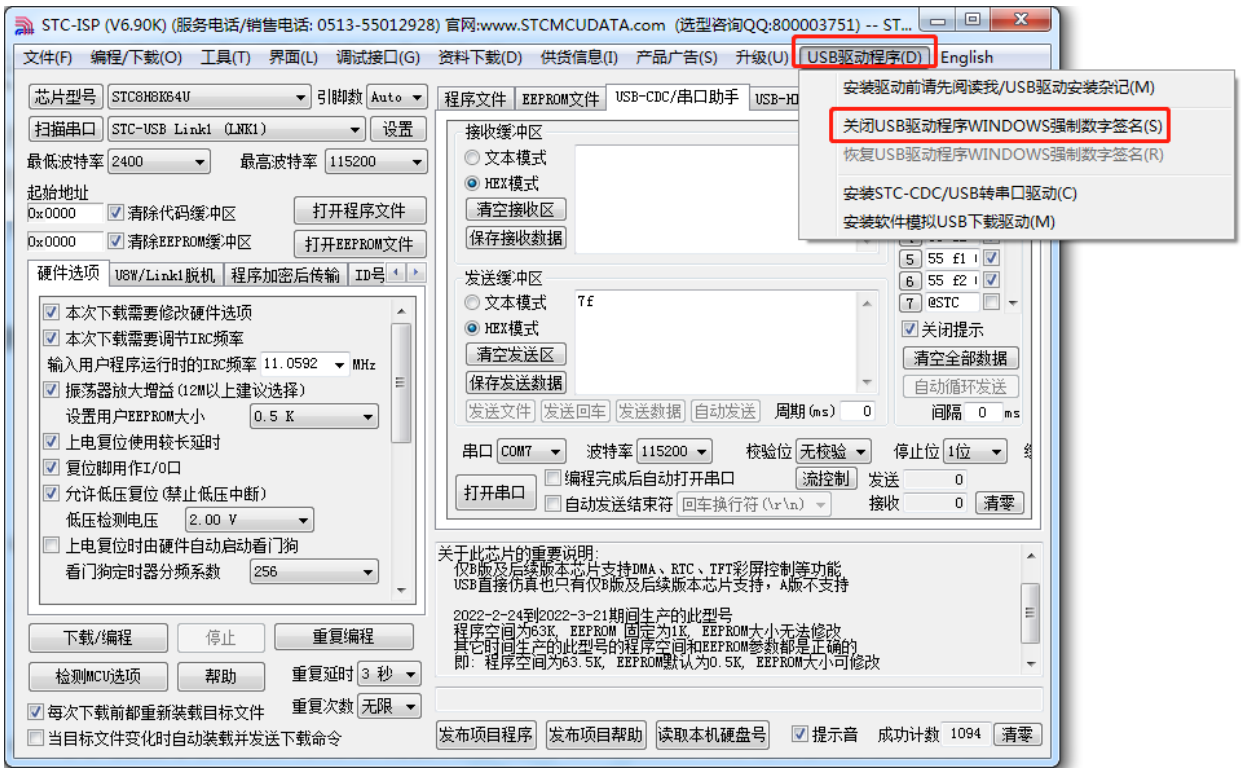


5.3 驱动程序数字签名的相关说明

5.3.1 关于驱动程序强制数字签名

Windows 系统从 Vista 版本开始的 64 位 Windows 系统就对驱动程序强制需要数字签名才能安装，否则驱动无法安装，USB 设备也无法正常使用。目前 STC 提供的驱动程序暂时没有做 WHQL 认证(微软徽标认证)，在没有关闭驱动程序强制数字签名前，在从 Vista 系统开始的 64 位 Windows 系统中可能无法安装成功。

STC-ISP 最新的下载软件提供了一个简单关闭驱动程序强制数字签名的功能，菜单目录如下图所示：

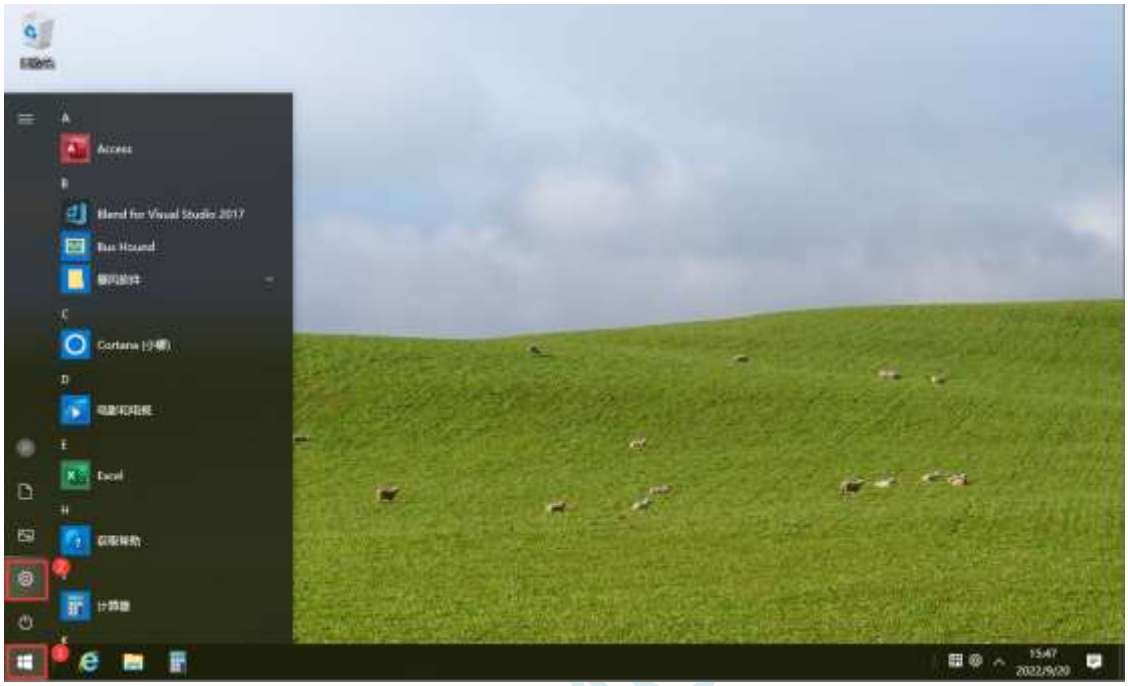


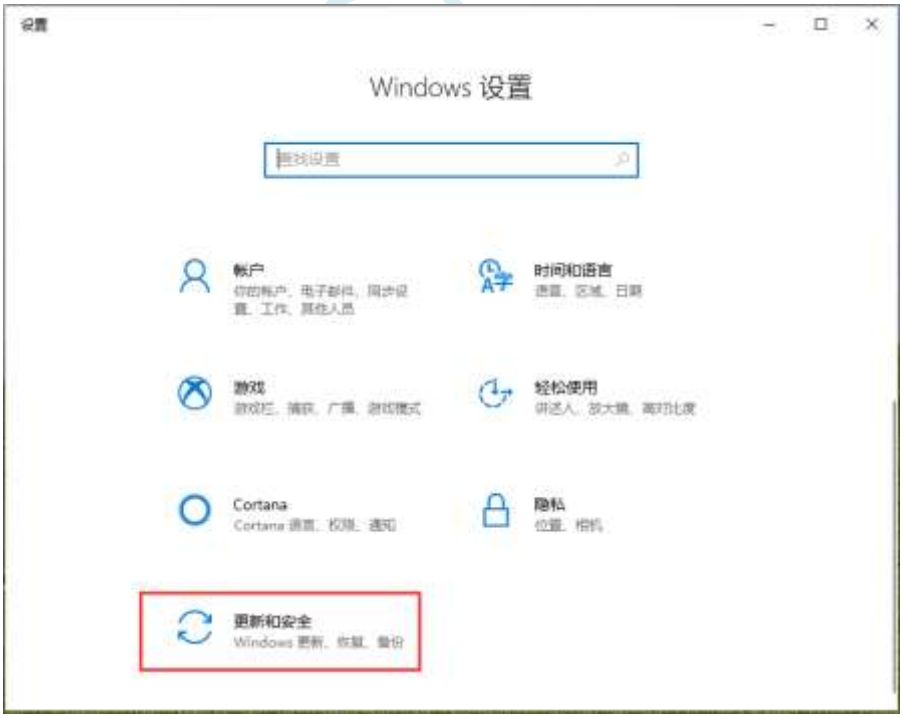
本菜单项是使用系统的 BCDEDIT 工具自动设置的 TestSigning 和 NoIntegrityChecks 两项参数以达到临时关闭驱动程序强制数字签名的目的。对于使能了安全启动的系统会无法使用 BCDEDIT 工具，需要在 BIOS 中将安全启动关闭（将 Secure Boot 设置为 Disable）

目前测试 WinXP/Win7-32/Win7-64 系统均可正常关闭数字签名，但 Win10 及后续系统无效。要在 Win10 及后续系统中关闭数字签名仍然需要从系统启动菜单中选择禁止强制数字签名来实现。详细步骤请参考下一小节

5.3.2 Windows 10 关闭驱动程序强制数字签名步骤

在“开始”菜单中选择“设置”功能





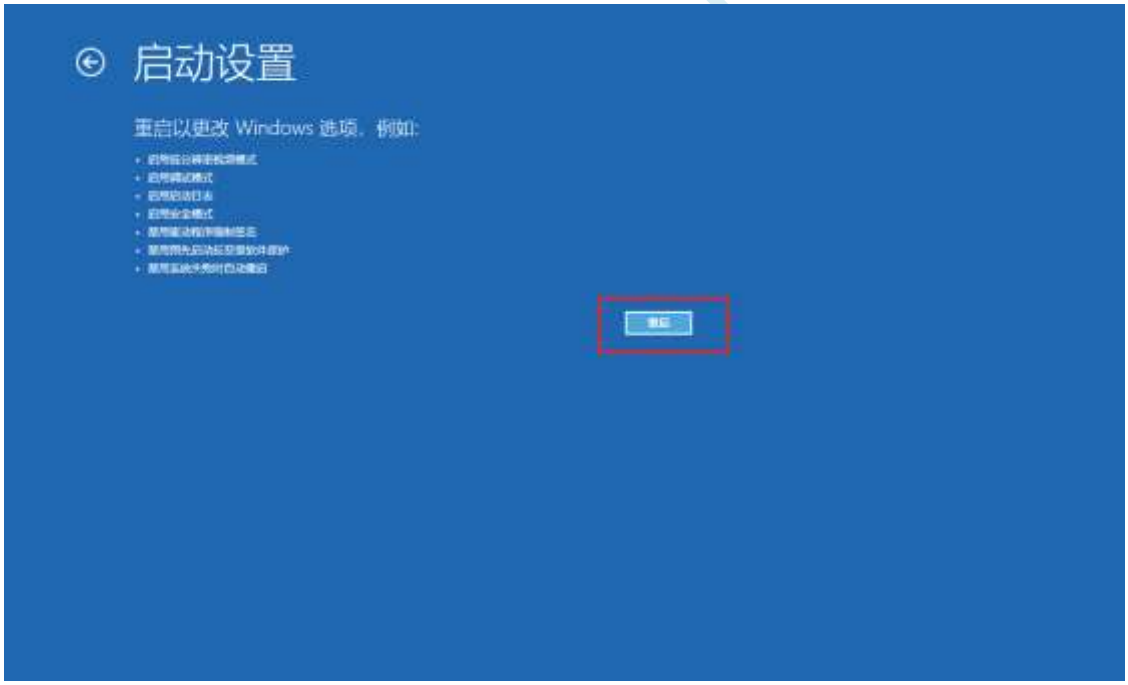
设置
主页
查找设置
更新和安全
Windows 更新
传递优化
Windows 安全中心
备份
疑难解答
恢复
激活
查找我的设备
开发者选项
恢复
重置此电脑
如果电脑未正常运行，重置电脑可能会解决问题。重置时，可以选择保留个人文件或删除个人文件，然后重新安装 Windows。
开始
高级启动
通过设备或磁盘(如 U 盘或 DVD)启动，更改你的电脑固件设置，更改 Windows 启动设置，或者从系统映像还原 Windows。 这将重新启动电脑。
立即重新启动
更多恢复选项
了解如何进行 Windows 的全新安装以便开始全新的体验
备份文件

选择一个选项
继续
关闭电脑
使用设备
疑难解答

疑难解答
重置此电脑
高级选项

高级选项
系统还原
启动修复
卸载更新
命令提示符
系统映像恢复
UEFI 固件设置
查看更多恢复选项

高级选项
启动设置
查看更多恢复选项


启动设置
重启以更改 Windows 选项，例如:

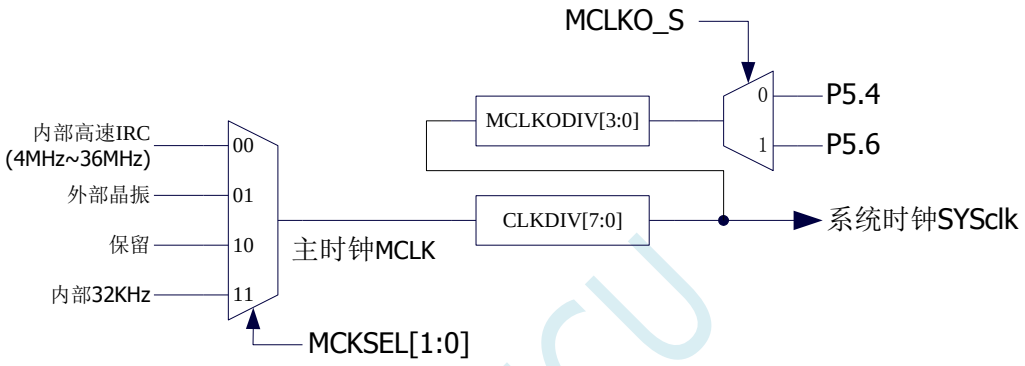
系统重新启动后，驱动程序的强制数字签名即可临时被关闭

6 时钟、复位与电源管理

6.1 系统时钟控制

系统时钟控制器为单片机的 CPU 和所有外设系统提供时钟源，系统时钟有 3 个时钟源可供选择：内部高精度 IRC、内部 32KHz 的 IRC（误差较大）、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
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	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	内部高速高精度 IRC
01	外部晶振或外部时钟
10	外部晶振或外部时钟
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：内部高速高精度 IRC 频率稳定标志位。（只读位）

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

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 XTALI（P5.7）**（此时 P5.6 口固定为高阻输入模式，可用于读取外部数字信号）**

1：外部时钟源是晶体振荡器。信号源连接单片机的 XTALI（P5.7）和 XTALO（P5.6）

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 P5.6 口

6.2 STC12H 系列内部 IRC 频率调整

STC12H 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％。

STC12H 系列内部 IRC 有两个频段，频段的中心频率分别为 20MHz 和 35MHz，20M 频段的调节范围约为 15.5MHz～27MHz，35M 频段的调节范围约为 27.5MHz～47MHz（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 39.5MHz，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	0000,00nn
LIRTRIM	IRC 频率微调寄存器	9EH	-	-	-	-	-	-	LIRTRIM[1:0]		0000,00nn
IRTRIM	IRC 频率调整寄存器	9FH	IRTRIM[7:0]								nnnn,nnnn
CLKDIV	时钟分频寄存器	FE01H	CLKDIV[7:0]								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 35MHz 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

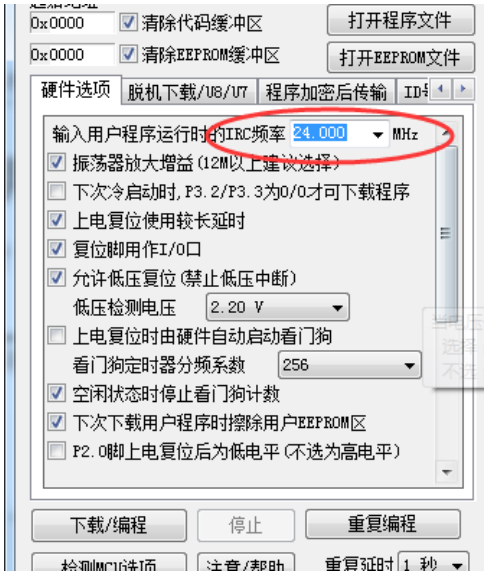
STC12H 系列内部的两个频段的可调范围分别为 15.5MHz～27MHz 和 25.3MHz～43.6MHz。虽然 35MHz 频段的上限可调到 40MHz 以上，但芯片内部的程序存储器无法运行到 40MHz 以上的速度，所以用户在 ISP 下载时设置内部 IRC 频率不能高于 40MHz，一般建议用户设置为 35MHz 以下。若用户需要较低的工作频率时，可使用 CLKDIV 寄存器对调节后的频率进行分频，例如用户需要 11.0592MHz 的频率，使用内部 IRC 直接调整是无法得到这个频率的，但可将内部 IRC 调整到 22.1184MHz，在使用 CLKDIV 进行 2 分频即可得到 11.0592MHz。

STC MCU

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 24MHz÷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++;                               //IRC 频率向上3‰进行微调（注意判断边界）
//  IRTRIM--;                               //IRC 频率向下3‰进行微调（注意判断边界）

    while (1);
}
```

汇编代码

```
;测试工作频率为24MHz

P_SW2       DATA        0BAH
IRTRIM      DATA        09FH

CLKSEL      EQU         0FE00H
CLKDIV      EQU         0FE01H
HIRCCR      EQU         0FE02H
XOSCCR      EQU         0FE03H
IRC32KCR    EQU         0FE04H

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,#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
```

STC MCU

6.3 系统复位

STC12H 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	-	IAP_WT[2:0]			0000,x000
RSTCFG	复位配置寄存器	FFH	-	ENLVR	-	P54RST	-	-	LVDS[1:0]		0000,0000

STC MCU

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

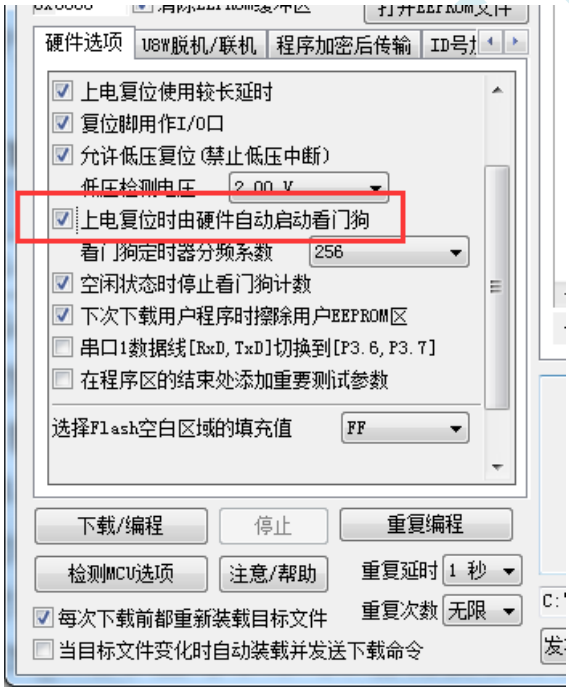
看门狗发生溢出时，硬件自动将此位置 1，需要软件清零。

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：

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

STC MCU

6.3.2 软件复位（IAP_CONTR）

IAP_CONTR（IAP 控制寄存器）

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

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

STC MCU

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（P54）

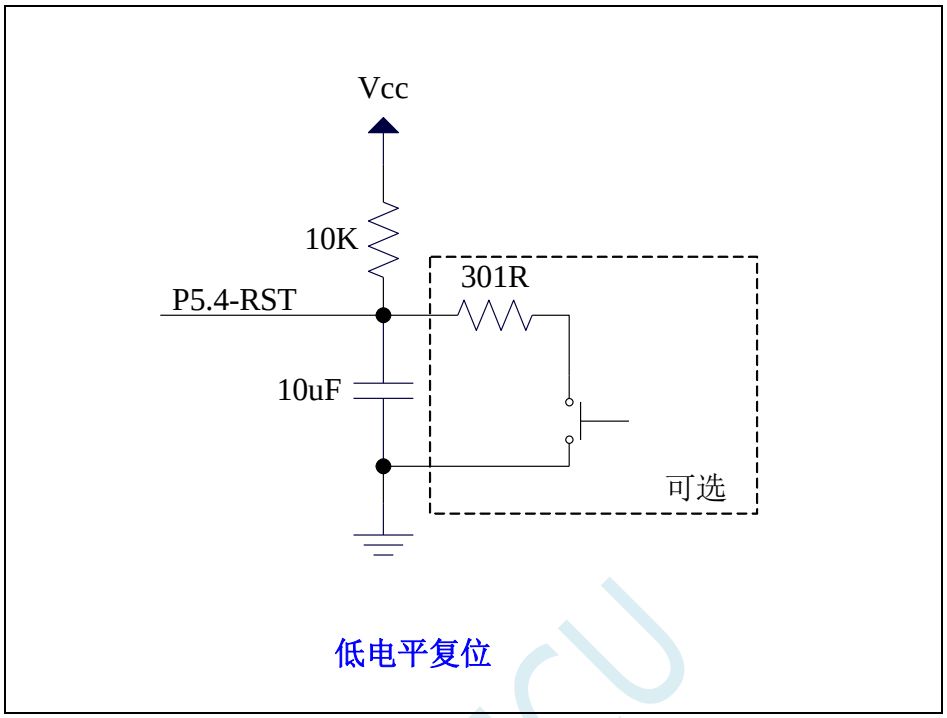
1：RST 管脚用作复位脚（低电平复位）

LVDS[1:0]：低压检测门槛电压设置

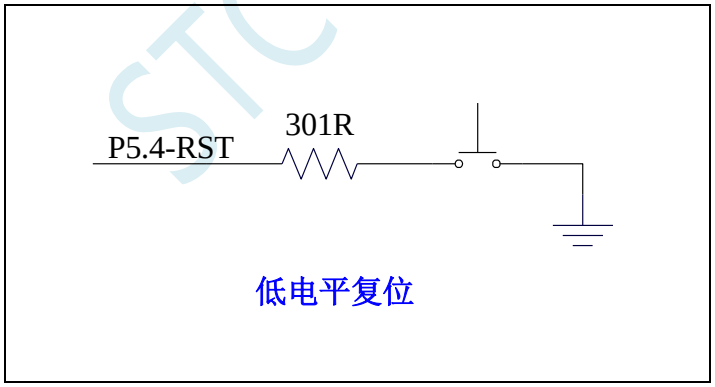
LVDS[1:0]	低压检测门槛电压
00	2.0V
01	2.4V
10	2.7V
11	3.0V

STC MCU

6.3.4 低电平上电复位参考电路（一般不需要）

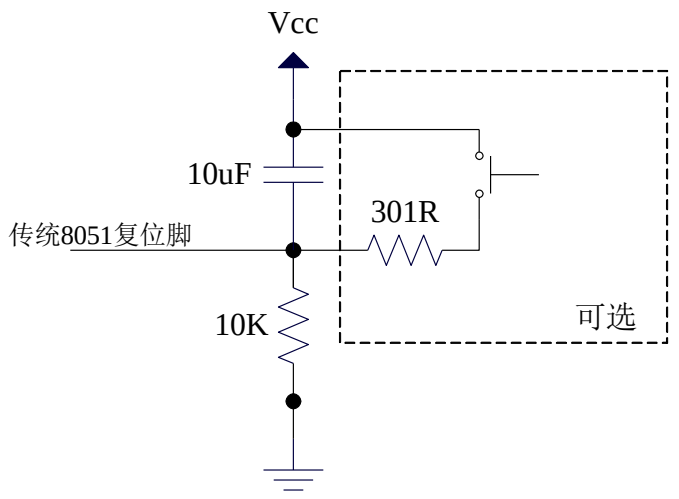


6.3.5 低电平按键复位参考电路



STCMCU

6.3.6 传统 8051 高电平上电复位参考电路



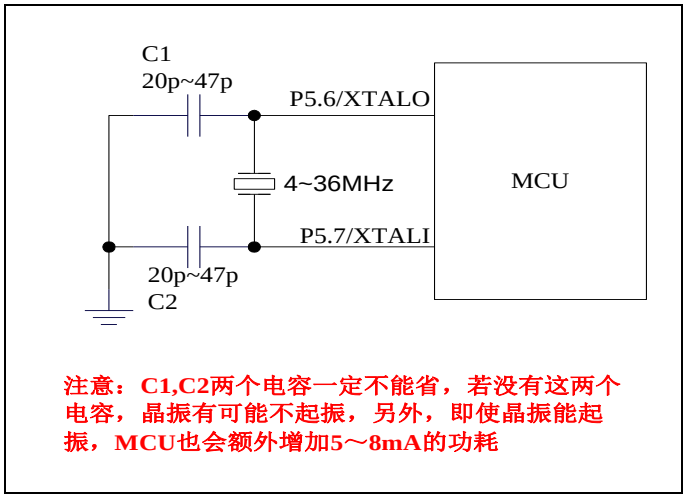
传统8051高电平复位电路

上图为传统 8051 的高电平复位电路，STC12H 的复位为低电平复位，与传统复位电路不同

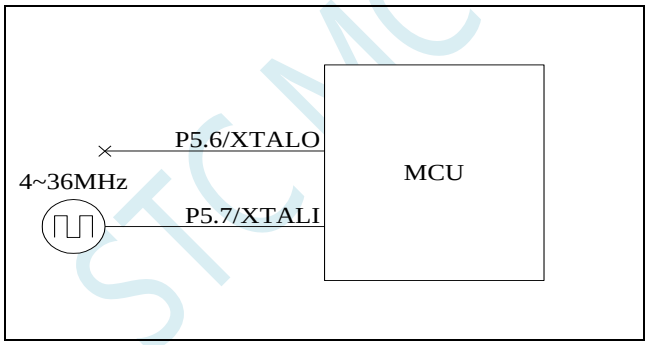
STC MCU

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（P5.6 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：上电标志位。当硬件自动将此位置 1。

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（0.4uA + 1.4uA + 1uA）。

STC MCU

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 7FFEH（即 32766）。（**计数值 0 和计数值 32767 为内部保留值，用户不能使用**）。内部掉电唤醒定时器有自己的内部时钟，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）

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

假设 F_{wt}=32KHz，则有：

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

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

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA        0BAH

CLKSEL      EQU         0FE00H
CLKDIV      EQU         0FE01H
HIRCCR      EQU         0FE02H
XOSCCR      EQU         0FE03H
IRC32KCR    EQU         0FE04H

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
```

STC MCU

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

;       MOV     P_SW2,#80H
;       MOV     A,#80H              ;启动内部 32K IRC
;       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       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;

    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 P5.6 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

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     P_SW2,#80H
;       MOV     A,#01H          ;主时钟输出到 P5.4 口
;       MOV     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
;       MOV     A,#84H          ;主时钟 4 分频输出到 P5.6 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     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;

//  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

WDT_CONTR   DATA        0C1H

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         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

            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     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;

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

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

汇编代码

```
;测试工作频率为 11.0592MHz

IAP_CONTR   DATA    0C7H

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

            SETB    P3.2
```

STC MCU

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

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 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 = 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
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

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

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

STC MCU

```
    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 = 1;
    _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_();
    P35 = 0;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

IDL         EQU         01H                     ;PCON.0
PD          EQU         02H                     ;PCON.1

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         0003H
            LJMP        INT0ISR

            ORG         0100H
INT0ISR:
            CPL         P3.4                    ;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

        SETB    EX0             ;使能 INT0 中断,用于唤醒 MCU
        SETB    EA
        NOP
        NOP
;       MOV     PCON,#IDL       ;MCU 进入 IDLE 模式
        MOV     PCON,#PD        ;MCU 进入掉电模式
        NOP
        NOP
        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     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 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)
    {
        P11 = ~P11;
    }
}
```

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

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     0003H
            LJMP    INT0ISR
            ORG     0013H
            LJMP    INT1ISR
            ORG     0053H
            LJMP    INT2ISR
            ORG     005BH
            LJMP    INT3ISR
            ORG     0083H
            LJMP    INT4ISR

            ORG     0100H
INT0ISR:
            CPL     P1.0            ;测试端口
            RETI
INT1ISR:
            CPL     P1.0            ;测试端口
            RETI
INT2ISR:
            CPL     P1.0            ;测试端口
            RETI
INT3ISR:
            CPL     P1.0            ;测试端口
            RETI
INT4ISR:
            CPL     P1.0            ;测试端口
            RETI
```

```
MAIN:
        MOV     SP, #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    IT0             ;使能 INT0 下降沿中断
        SETB    EX0             ;使能 INT0 中断

        CLR     IT1             ;使能 INT1 上升沿和下降沿中断
;       SETB    IT1             ;使能 INT1 下降沿中断
        SETB    EX1             ;使能 INT1 中断

        MOV     INTCLKO,#EX2    ;使能 INT2 下降沿中断
        ORL     INTCLKO,#EX3    ;使能 INT3 下降沿中断
        ORL     INTCLKO,#EX4    ;使能 INT4 下降沿中断

        SETB    EA

        MOV     PCON,#02H       ;MCU 进入掉电模式
        NOP                     ;掉电模式被唤醒后,MCU 首先会执行此语句
                                ;然后再进入中断服务程序
        NOP
        NOP
        NOP
LOOP:
        CPL     P1.1
        JMP     LOOP

        END
```

6.7.8 使用 T0/T1/T2/T3/T4 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     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     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 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;                        //启动定时器
    IE2 = ET2;                          //使能定时器中断

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

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

    EA = 1;

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

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

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

汇编代码

```
;测试工作频率为 11.0592MHz

T2L     DATA    0D7H
T2H     DATA    0D6H
T3L     DATA    0D5H
T3H     DATA    0D4H
T4L     DATA    0D3H
```

```
T4H         DATA    0D2H
T4T3M       DATA    0D1H
AUXR        DATA    8EH

IE2         DATA    0AFH
ET2         EQU     04H
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     000BH
            LJMP    TM0ISR
            ORG     001BH
            LJMP    TM1ISR
            ORG     0063H
            LJMP    TM2ISR
            ORG     009BH
            LJMP    TM3ISR
            ORG     00A3H
            LJMP    TM4ISR

            ORG     0100H
TM0ISR:
            CPL     P1.0            ;测试端口
            RETI
TM1ISR:
            CPL     P1.0            ;测试端口
            RETI
TM2ISR:
            CPL     P1.0            ;测试端口
            RETI
TM3ISR:
            CPL     P1.0            ;测试端口
            RETI
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     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           ;启动定时器
        MOV     IE2,#ET2            ;使能定时器中断

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

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

        SETB    EA

        MOV     PCON,#02H           ;MCU 进入掉电模式
        NOP                         ;掉电唤醒后不会立即进入中断服务程序,
                                    ;而是等到定时器溢出后才会进入中断服务程序
        NOP
        NOP
        NOP
LOOP:
        CPL     P1.1
        JMP     LOOP

        END
```

6.7.9 使用 RxD/RxD2 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     IE2     =   0xaf;
#define ES2         0x01
#define ES3         0x08
#define ES4         0x10

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

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 UART1_Isr() interrupt 4
{
}

void UART2_Isr() interrupt 8
{
}

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 下降沿唤醒
//  P_SW1 = 0x40;                   //RXD_2 下降沿唤醒
//  P_SW1 = 0x80;                   //RXD_3 下降沿唤醒
//  P_SW1 = 0xc0;                   //RXD_4 下降沿唤醒

    P_SW2 = 0x00;                   //RXD2 下降沿唤醒
//  P_SW2 = 0x01;                   //RXD2_2 下降沿唤醒
```

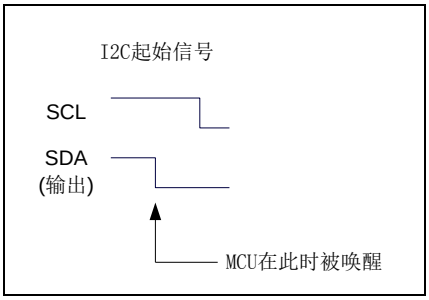
```
    ES = 1;                                     //使能串口中断
    IE2 = ES2;                                  //使能串口中断
    EA = 1;

    PCON = 0x02;                                //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_();
    _nop_();

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

汇编代码

```
;测试工作频率为 11.0592MHz

IE2         DATA        0AFH
ES2         EQU         01H

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         0100H
UART1ISR:
            RETI
UART2ISR:
            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 下降沿唤醒
;           MOV         P_SW1,#40H              ;RXD_2 下降沿唤醒
;           MOV         P_SW1,#80H              ;RXD_3 下降沿唤醒
;           MOV         P_SW1,#0C0H             ;RXD_4 下降沿唤醒

            MOV         P_SW2,#00H              ;RXD2 下降沿唤醒
;           MOV         P_SW2,#01H              ;RXD2_2 下降沿唤醒

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

            MOV         PCON,#02H               ;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
            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 下降沿唤醒
//  P_SW2 = 0x10;                       //SDA_2 下降沿唤醒
//  P_SW2 = 0x30;                       //SDA_4 下降沿唤醒
    P_SW2 |= 0x80;
    I2CCFG = 0x80;                      //使能 I2C 模块的从机模式
    I2CSLCR = 0x40;                     //使能起始信号中断
    EA = 1;

    PCON = 0x02;                        //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_();
    _nop_();

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

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
P4M0        DATA        0B4H
P5M1        DATA        0C9H
P5M0        DATA        0CAH

            ORG         0000H
            LJMP        MAIN
            ORG         00C3H
            LJMP        I2CISR

            ORG         0100H
I2CISR:
            PUSH        ACC
            PUSH        DPH
            PUSH        DPL
            ORL         PSW2,#80H
            MOV         DPTR,#I2CSLST
            MOVX        A,@DPTR
            ANL         A,#NOT 40H
            MOVX        @DPTR,A
            POP         DPL
            POP         DPH
            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         P_SW2,#00H          ;SDA 下降沿唤醒
//          MOV         P_SW2,#10H          ;SDA_2 下降沿唤醒
```

```
//          MOV     P_SW2,#30H          ;SDA_4 下降沿唤醒
            ORL     P_SW2,#80H
            MOV     DPTR,#I2CCFG
            MOV     A,#80H
            MOVX    @DPTR,A             ;使能 I2C 模块的从机模式
            MOV     DPTR,# I2CSLCR
            MOV     A,#40H              ;使能起始信号中断
            SETB    EA

            MOV     PCON,#02H           ;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
            NOP
LOOP:
            CPL     P1.1
            JMP     LOOP

            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    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;

    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_();
        P11 = ~P11;
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

WKTCL       DATA        0AAH
WKTCH       DATA        0ABH

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
```

STC MCU

```
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     WKTCL,#0FFH         ;设定掉电唤醒时钟约为 1 秒钟
        MOV     WKTCH,#87H

LOOP:
        NOP
        NOP
        MOV     PCON,#02H           ;MCU 进入掉电模式
        NOP
        NOP
        NOP
        NOP
        CPL     P1.1
        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;
#define ENLVR           0x40            //RSTCFG.6
#define LVD2V0          0x00            //LVD@2.0V
#define LVD2V4          0x01            //LVD@2.4V
#define LVD2V7          0x02            //LVD@2.7V
#define LVD3V0          0x03            //LVD@3.0V
sbit    ELVD        =   IE^6;
#define LVDF            0x20            //PCON.5

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

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 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_();                        //掉电唤醒后立即进入中断服务程序
    _nop_();
    _nop_();
    _nop_();

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

汇编代码

```
;测试工作频率为 11.0592MHz

RSTCFG      DATA    0FFH
ENLVR       EQU     40H             ;RSTCFG.6
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

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         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 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     CMPCR1      =   0xe6;
sfr     CMPCR2      =   0xe7;

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

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 CMP_Isr() interrupt 21
{
    CMPCR1 &= ~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;
```

STC MCU

```
    P5M1 = 0x00;

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

    PCON = 0x02;                            //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_();

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

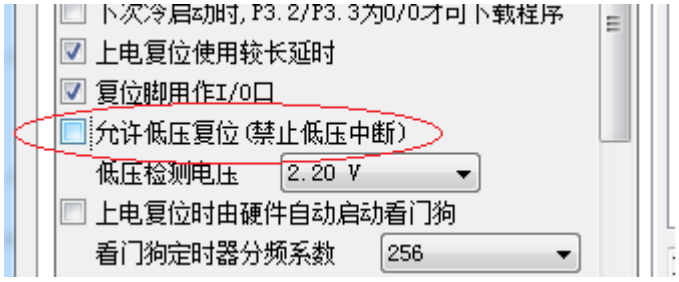
汇编代码

```
;测试工作频率为 11.0592MHz

CMPCR1      DATA        0E6H
CMPCR2      DATA        0E7H

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         00ABH
            LJMP        CMPISR

            ORG         0100H
CMPISR:
            ANL         CMPCR1,#NOT 40H     ;清中断标志
            CPL         P1.0                ;测试端口
            RETI

MAIN:
            MOV         SP, #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     CMPCR1,#80H         ;使能比较器模块
        ORL     CMPCR1,#30H         ;使能比较器边沿中断
        ANL     CMPCR1,#NOT 08H     ;P3.6 为CMP+输入脚
        ORL     CMPCR1,#04H         ;P3.7 为CMP-输入脚
        ORL     CMPCR1,#02H         ;使能比较器输出
        SETB    EA

        MOV     PCON,#02H           ;MCU 进入掉电模式
        NOP                         ;掉电唤醒后立即进入中断服务程序
        NOP
        NOP
        NOP

LOOP:
        CPL     P1.1
        JMP     LOOP

        END
```

6.7.14 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



C 语言代码

```
//测试工作频率为11.0592MHz

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

#define FOSC        11059200UL
#define T1MS        (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        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 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;
```

STC MCU

```
        delay();
        if (PCON & LVDF)
        {
            power >>= 1;
            RSTCFG = LVD2V7;
            delay();
            PCON &= ~LVDF;
            delay();
            if (PCON & LVDF)
            {
                power >>= 1;
                RSTCFG = LVD2V4;
                delay();
                PCON &= ~LVDF;
                delay();
                if (PCON & LVDF)
                {
                    power >>= 1;
                    RSTCFG = LVD2V2;
                    delay();
                    PCON &= ~LVDF;
                    delay();
                    if (PCON & LVDF)
                    {
                        power >>= 1;
                    }
                }
            }
        }
        RSTCFG = LVD3V0;
        P2 = ~power;    //P2.3~P2.0 显示电池电量
    }
}
```

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

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
        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     P5M1, #00H

        ANL     PCON,#NOT LVDF
        MOV     RSTCFG,#LVD3V0
LOOP:
        MOV     B,#0FH

        MOV     RSTCFG,#LVD3V0
        CALL    DELAY
        ANL     PCON,#NOT LVDF
        CALL    DELAY
        MOV     A,PCON
        ANL     A,#LVDF
        JZ      SKIP
        MOV     A,B
        CLR     C
        RRC     A
        MOV     B,A

        MOV     RSTCFG,#LVD2V7
        CALL    DELAY
        ANL     PCON,#NOT LVDF
        CALL    DELAY
        MOV     A,PCON
        ANL     A,#LVDF
        JZ      SKIP
        MOV     A,B
        CLR     C
        RRC     A
        MOV     B,A

        MOV     RSTCFG,#LVD2V4
        CALL    DELAY
        ANL     PCON,#NOT LVDF
        CALL    DELAY
        MOV     A,PCON
        ANL     A,#LVDF
        JZ      SKIP
        MOV     A,B
        CLR     C
        RRC     A
```

STC MCU

```
        MOV     B,A

        MOV     RSTCFG,#LVD2V2
        CALL    DELAY
        ANL     PCON,#NOT LVDF
        CALL    DELAY
        MOV     A,PCON
        ANL     A,#LVDF
        JZ      SKIP
        MOV     A,B
        CLR     C
        RRC     A
        MOV     B,A

SKIP:
        MOV     A,B
        CPL     A
        MOV     P2,A                    ;P2.3~P2.0 显示电池电量
        JMP     LOOP

DELAY:
        MOV     R0,#100
NEXT:
        NOP
        NOP
        NOP
        NOP
        DJNZ    R0,NEXT
        RET

        END
```

STC MCU

7 存储器

STC12H 系列单片机的程序存储器和数据存储器是各自独立编址的。由于没有提供访问外部程序存储器的总线，所有单片机的所有程序存储器都是片上 Flash 存储器，不能访问外部程序存储器。

STC12H 系列单片机内部集成了大容量的数据存储器。STC12H 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。

STC12H1K08 系列单片内部集成了 28K 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。

STC12H 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

STC12H 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）
STC12H1K08 系列	128 字节	128 字节	1024 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。

STCMCU

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。

STCMCU

7.2.3 内部扩展 RAM，XRAM，XDATA

STC12H 系列单片机片内除了集成 256 字节的内部 RAM 外，还集成了内部的扩展 RAM。访问内部扩展 RAM 的方法和传统 8051 单片机访问外部扩展 RAM 的方法相同，但是不影响 P0 口(数据总线和高八位地址总线)、P2 口(低八位地址总线)、以及 RD、WR 和 ALE 等端口上的信号。

在汇编语言中，内部扩展 RAM 通过 MOVX 指令访问，

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

在 C 语言中，可使用 xdata/pdata 声明存储类型即可。如：

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

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

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

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。

STC MCU

7.2.5　外部扩展 RAM，XRAM，XDATA

STC12H 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。STC12H 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

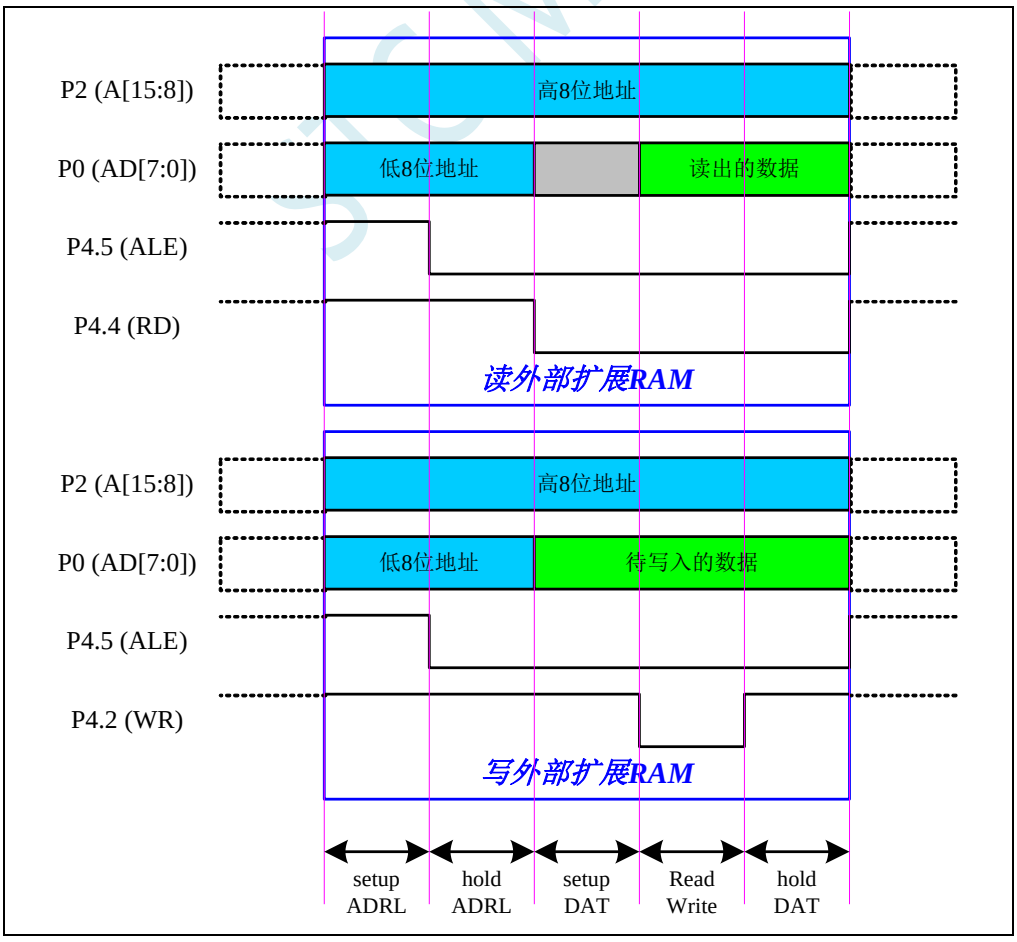
00：P4.4 为 RD，P4.2 为 WR

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

STCMCU

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

STC MCU

7.3 存储器中的特殊参数，在 ISP 下载时可烧录进程序 FLASH

STC12H 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
	STC12H1K08	STC12H1K16	STC12H1K28	
全球唯一ID号	1FF9H~1FFFH	2FF9H~2FFFH	43F9H~43FFH	7字节
内部1.19V参考信号源	1FF7H~1FF8H	2FF7H~2FF8H	43F7H~43F8H	毫伏（高字节在前）
32K掉电唤醒定时器的频率	1FF5H~1FF6H	2FF5H~2FF6H	43F5H~43F6H	Hz （高字节在前）
22.1184MHz的IRC参数（20M频段）	1FF4H	2FF4H	43F4H	—
24MHz的IRC参数（20M频段）	1FF3H	2FF3H	43F3H	—
20MHz的IRC参数（20M频段）	1FF2H	2FF2H	43F2H	固件版本为7.3.12U 以及后续版本有效
27MHz的IRC参数（35M频段）	1FF1H	2FF1H	43F1H	
30MHz的IRC参数（35M频段）	1FF0H	2FF0H	43F0H	
33.1776MHz的IRC参数（35M频段）	1FEFH	2FEFH	43EFH	
35MHz的IRC参数（35M频段）	1FEEH	2FEEH	43EEH	
36.864MHz的IRC参数（35M频段）	1FEDH	2FEDH	43EDH	
保留	1FECH	2FECH	43ECH	
保留	1FEBH	2FEBH	43EBH	
20M频段的VRTRIM参数	1FEAH	2FEAH	43EAH	
35M频段的VRTRIM参数	1FE9H	2FE9H	43E9H	

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	—

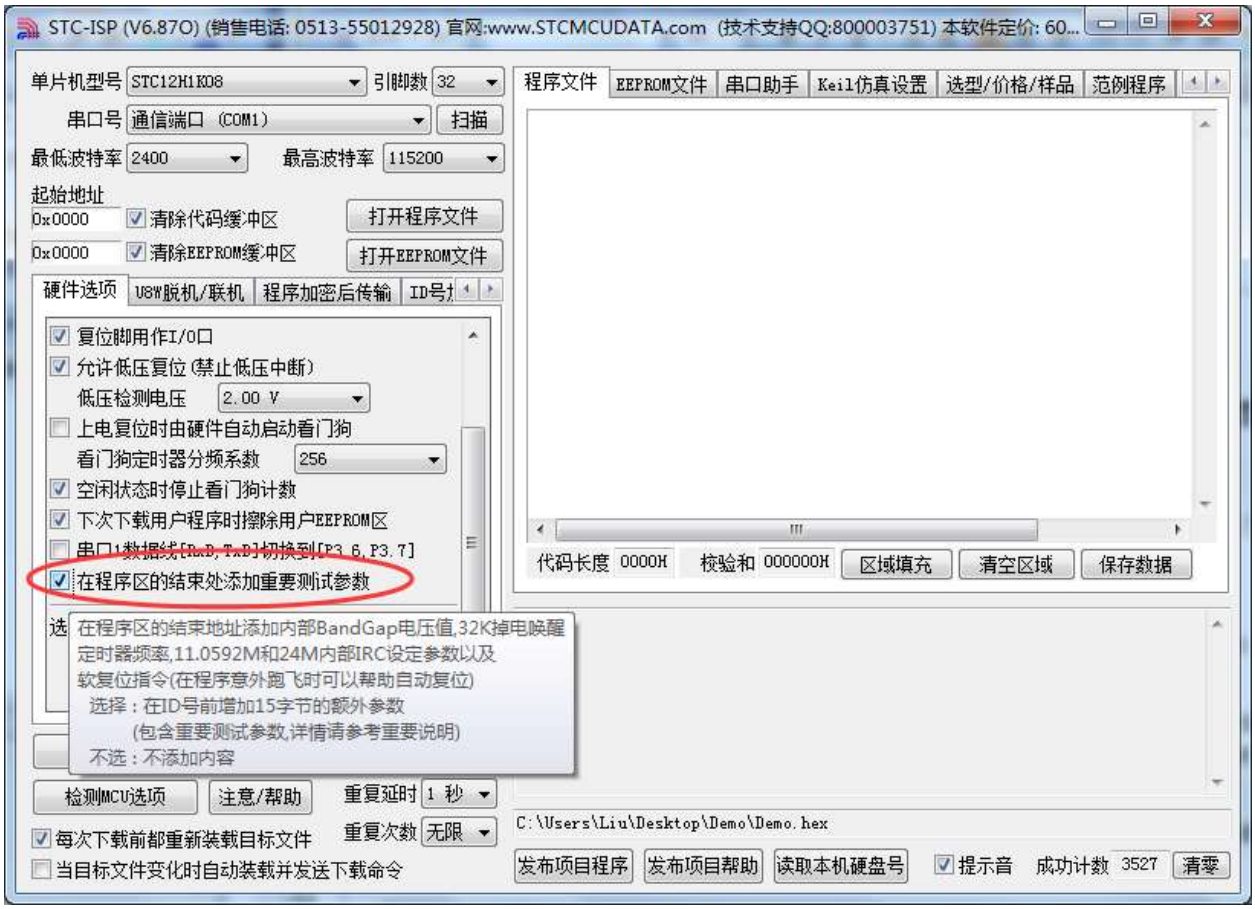
STC MCU

特别说明

1、由于 RAM 中的参数可能被修改，所以一般不建议用户使用，特别是用户使用 ID 号进行加密时，强烈建议用于读取 ROM 中的 ID 数据。

2、由于 STC12H1K28、STC12H1K33 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 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。

3、默认情况下，程序存储器中只有全球唯一 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         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;

bit         busy;
int         *BGV;

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

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

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

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

STC MCU

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

    BGV = (int code *)0x3ff7;                    // STC12H1K16
    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         03FF7H              ;STC12H1K16

BUSY        BIT         20H.0

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         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     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;

bit     busy;
int     *BGV;

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

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

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

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

STC MCU

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

    BGV = (int idata *)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

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         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     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;

bit     busy;
char    *ID;

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

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

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

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

STC MCU

```
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    char i;

    ID = (char code *)0x3ff9;                    // STC12H1K16
    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         03FF9H              ; STC12H1K16

BUSY        BIT         20H.0

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         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,#ID
            MOV     R1,#7
NEXT:       CLR     A
            MOVC    A,@A+DPTR
            LCALL   UART_SEND
            INC     DPTR
            DJNZ    R1,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         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;

bit         busy;
char        *ID;

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

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

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

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

STC MCU

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

    char i;

    ID = (char idata *)0xf1;
    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          DATA        0F1H

BUSY        BIT         20H.0

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         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,#ID
            MOV     R1,#7
NEXT:       MOV     A,@R0
            LCALL   UART_SEND
            INC     R0
            DJNZ    R1,NEXT

LOOP:
            JMP     LOOP

            END
```

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	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;

bit	busy;
int	*F32K;

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

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

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

void main()
```

```
{
    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 *)0x3ff5;                  // STC12H1K16
    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        EQU         03FF5H              ; STC12H1K16

BUSY        BIT         20H.0

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

LOOP:
            JMP     LOOP

            END
```

~~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	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;

bit	busy;
int	*F32K;

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

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

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

void main()
```

```
{
    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

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

LOOP:
        JMP     LOOP

        END
```

STC MCU

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STC12H1K08-20Pin 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     =    0x9d;
sfr        IRTRIM      =    0x9f;
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;
```

STC MCU

```
//      CLKDIV = 0x00;

        while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

;下表为 STC12H1K08-20Pin 的参数列表
ID_ROMADDR          EQU     01FF9H
VREF_ROMADDR        EQU     01FF7H
F32K_ROMADDR        EQU     01FF5H
T22M_ROMADDR        EQU     01FF4H          //22.1184MHz
T24M_ROMADDR        EQU     01FF3H          //24MHz
T20M_ROMADDR        EQU     01FF2H          //20MHz
T27M_ROMADDR        EQU     01FF1H          //27MHz
T30M_ROMADDR        EQU     01FF0H          //30MHz
T33M_ROMADDR        EQU     01FEFH          //33.1776MHz
T35M_ROMADDR        EQU     01FEEH          //35MHz
T36M_ROMADDR        EQU     01FEDH          //36.864MHz
VRT20M_ROMADDR      EQU     01FEAH          //VRTRIM_20M
VRT35M_ROMADDR      EQU     01FE9H          //VRTRIM_35M

P_SW2       DATA        0BAH
CLKSEL      EQU         0FE00H
CLKDIV      EQU         0FE01H

IRCBAND     DATA        09DH
IRCTRIM     DATA        09FH
VRTRIM      DATA        0A6H

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

;           ;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

P_SW2       DATA        0BAH
CLKDIV      EQU         0FE01H

IRTRIM      DATA        09FH

IRC22M      DATA        0FAH
IRC24M      DATA        0FBH

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     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
```

STC MCU

8 特殊功能寄存器

8.1 STC12H1K08 系列

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H		CH	CCAP0H	CCAP1H	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	T4T3M	T4H	T4L	T3H	T3L	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	WKTCL	WKTCH			TA	IE2
A0H	P2		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
FEF8H	PWMB_CCR6L	PWMB_CCR7H	PWMB_CCR7L	PWMB_CCR8H	PWMB_CCR8L	PWMB_BKR	PWMB_DTR	PWMB_OISR
FEF0H	PWMB_PSCRH	PWMB_PSCRL	PWMB_ARRH	PWMB_ARRL	PWMB_RCR	PWMB_CCR5H	PWMB_CCR5L	PWMB_CCR6H
FEE8H	PWMB_CCMR1	PWMB_CCMR2	PWMB_CCMR3	PWMB_CCMR4	PWMB_CCER1	PWMB_CCER2	PWMB_CNTRH	PWMB_CNTRL
FEE0H	PWMB_CR1	PWMB_CR2	PWMB_SMCR	PWMB_ETR	PWMB_IER	PWMB_SR1	PWMB_SR2	PWMB_EGR
FED8H	PWMA_CCR2L	PWMA_CCR3H	PWMA_CCR3L	PWMA_CCR4H	PWMA_CCR4L	PWMA_BKR	PWMA_DTR	PWMA_OISR
FED0H	PWMA_PSCRH	PWMA_PSCRL	PWMA_ARRH	PWMA_ARRL	PWMA_RCR	PWMA_CCR1H	PWMA_CCR1L	PWMA_CCR2H
FEC8H	PWMA_CCMR1	PWMA_CCMR2	PWMA_CCMR3	PWMA_CCMR4	PWMA_CCER1	PWMA_CCER2	PWMA_CNTRH	PWMA_CNTRL
FEC0H	PWMA_CR1	PWMA_CR2	PWMA_SMCR	PWMA_ETR	PWMA_IER	PWMA_SR1	PWMA_SR2	PWMA_EGR
FEB0H	PWMA_ETRPS	PWMA_ENO	PWMA_PS	PWMA_IOAUX	PWMB_ETRPS	PWMB_ENO	PWMB_PS	PWMB_IOAUX
FEA8H	ADCTIM							
FEA0H			TM2PS	TM3PS	TM4PS			
FE88H	I2CMSAUX							
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE30H		P1IE	P2IE					
FE28H	P0DR	P1DR	P2DR	P3DR		P5DR		
FE20H	P0SR	P1SR	P2SR	P3SR		P5SR		
FE18H	P0NCS	P1NCS	P2NCS	P3NCS		P5NCS		

FE10H	P0PU	P1PU	P2PU	P3PU		P5PU		
FE00H	CLKSEL	CLKDIV	HIRCCR	XOSCCR	IRC32KCR	MCLKOCR	IRCDB	
FD50H					CCAPM3	CCAP3L	CCAP3H	PCA_PWM3
FD30H	P0IM1	P1IM1	P2IM1	P3IM1		P5IM1		
FD20H	P0IM0	P1IM0	P2IM0	P3IM0		P5IM0		
FD10H	P0INTF	P1INTF	P2INTF	P3INTF		P5INTF		
FD00H	P0INTE	P1INTE	P2INTE	P3INTE		P5INTE		
FCF0H	MD3	MD2	MD1	MD0	MD5	MD4	ARCON	OPCON

STC MCU

8.2 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。

STC12H 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）、P5（C8H）、PSW（D0H）、CCON（D8H）、ACC（E0H）、B（F0H）

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
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寄存器	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000
P1	P1 端口	90H	P17	P16	P15	P14	P13	P12	P11	P10	1111,1111
P1M1	P1 口配置寄存器 1	91H	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	0100,0000
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]		0000,00nn
IRTRIM	IRC 频率调整寄存器	9FH	IRTRIM[7:0]								nnnn,nnnn
P2	P2 端口	A0H	P27	P26	P25	P24	P23	P22	P21	P20	1111,1111
P_SW1	外设端口切换寄存器 1	A2H	S1_S[1:0]				SPI_S[1:0]		0	-	nn00,000x
IE	中断允许寄存器	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000,0000
SADDR	串口 1 从机地址寄存器	A9H									0000,0000
WKTCL	掉电唤醒定时器低字节	AAH									1111,1111
WKTCH	掉电唤醒定时器高字节	ABH	WKTEN								0111,1111
TA	DPTR 时序控制寄存器	AEH									0000,0000
IE2	中断允许寄存器 2	AFH	-	ET4	ET3	-	-	ET2	ESPI	ES2	x000,0000
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
IP2	中断优先级控制寄存器 2	B5H	-	PI2C	PCMP	PX4	PPWMB	PPWMA	PSPI	PS2	0000,0000
IP2H	高中断优先级控制寄存器 2	B6H	-	PI2CH	PCMPH	PX4H	PPWMBH	PPWMAH	PSPIH	PS2H	0000,0000
IPH	高中断优先级控制寄存器	B7H	-	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H	x000,0000
IP	中断优先级控制寄存器	B8H	-	PLVD	PADC	PS	PT1	PX1	PT0	PX0	x000,0000
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]		-	-	-	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]				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
WDT_CONTR	看门狗控制寄存器	C1H	WDT_FLAG	-	EN_WDT	CLR_WDT	IDL_WDT	WDT_PS[2:0]			0x00,0000
IAP_DATA	IAP 数据寄存器	C2H									1111,1111
IAP_ADDRH	IAP 高地址寄存器	C3H									0000,0000
IAP_ADDRL	IAP 低地址寄存器	C4H									0000,0000
IAP_CMD	IAP 命令寄存器	C5H	-	-	-	-	-	-	CMD[1:0]		xxxx,xx00
IAP_TRIG	IAP 触发寄存器	C6H									0000,0000
IAP_CONTR	IAP 控制寄存器	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	-	-	-	0000,xxxx
P5	P5 端口	C8H	P57	P56	P55	P54	P53	P52	P51	P50	xx11,1111
P5M1	P5 口配置寄存器 1	C9H	P57M1	P56M1	P55M1	P54M1	P53M1	P52M1	P51M1	P50M1	xx11,1111
P5M0	P5 口配置寄存器 0	CAH	P57M0	P56M0	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,0000
CMOD	PCA 模式寄存器	D9H	CIDL	-	-	-	CPS[2:0]			ECF	0xxx,0000
CCAPM0	PCA 模块 0 模式控制寄存器	DAH	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0	x000,0000
CCAPM1	PCA 模块 1 模式控制寄存器	DBH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1	x000,0000
CCAPM2	PCA 模块 2 模式控制寄存器	DCH	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2	x000,0000
ADCCFG	ADC 配置寄存器	DEH	-	-	RESFMT	-	SPEED[3:0]				xx0x,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	LCDTY[5:0]						0000,0000

STC MCU

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
AUXINTIF	扩展外部中断标志寄存器	EFH	-	INT4IF	INT3IF	INT2IF	-	T4IF	T3IF	T2IF	x000,x000
B	B 寄存器	F0H								0000,0000
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
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
RSTCFG	复位配置寄存器	FFH	-	ENLVR	-	P54RST	-	-	LVDS[1:0]		0000,0000

下列特殊功能寄存器为扩展 SFR，逻辑地址位于 XDATA 区域，访问前需要将 P_SW2（BAH）寄存器的最高位（EAXFR）置 1，然后使用 MOVX A,@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	MCLKODIV[6:0]							0000,0000
IRCDB	内部高速振荡器去抖控制	FE06H	IRCDB_PAR[7:0]								1000,0000
P0PU	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
P5PU	P5 口上拉电阻控制寄存器	FE15H	P57PU	P56PU	P55PU	P54PU	P53PU	P52PU	P51PU	P50PU	xxx0,0000
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
P5NCS	P5 口施密特触发控制寄存器	FE1DH	P57NCS	P56NCS	P55NCS	P54NCS	P53NCS	P52NCS	P51NCS	P50NCS	xxx0,0000
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
P5SR	P5 口电平转换速率寄存器	FE25H	P57SR	P56SR	P55SR	P54SR	P53SR	P52SR	P51SR	P50SR	xxx1,1111
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
P5DR	P5 口驱动电流控制寄存器	FE2DH	P57DR	P56DR	P55DR	P54DR	P53DR	P52DR	P51DR	P50DR	xxx1,1111
P1IE	P1 口输入使能控制寄存器	FE31H	P17IE	P16IE	P15IE	P14IE	P13IE	P12IE	P11IE	P10IE	1111,1111
P2IE	P2 口输入使能控制寄存器	FE32H	-	P26IE	P25IE	P24IE	P23IE	P22IE	P21IE	P20IE	x111,1111
I2CCFG	I^2C 配置寄存器	FE80H	ENI2C	MSSL	MSSPEED[6:1]						0000,0000
I2CMSCR	I^2C 主机控制寄存器	FE81H	EMSI	-	-	-	MSCMD[3:0]				0xxx,0000
I2CMSST	I^2C 主机状态寄存器	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO	00xx,xx00
I2CSLCR	I^2C 从机控制寄存器	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0
I2CSLST	I^2C 从机状态寄存器	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000
I2CSLADR	I^2C 从机地址寄存器	FE85H	I2CSLADR[7:1]							MA	0000,0000
I2CTXD	I^2C 数据发送寄存器	FE86H									0000,0000
I2CRXD	I^2C 数据接收寄存器	FE87H									0000,0000
I2CMSAUX	I^2C 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
PWMA_ETRPS	PWMA 的 ETR 选择寄存器	FEB0H						BRKAPS	ETRAPS[1:0]		xxxx,x000
PWMA_ENO	PWMA 输出使能控制	FEB1H	ENO4N	ENO4P	ENO3N	ENO3P	ENO2N	ENO2P	ENO1N	ENO1P	0000,0000
PWMA_PS	PWMA 输出脚选择寄存器	FEB2H	C4PS[1:0]		C3PS[1:0]		C2PS[1:0]		C1PS[1:0]		0000,0000
PWMA_IOAUX	PWMA 辅助寄存器	FEB3H	AUX4N	AUX4P	AUX3N	AUX3P	AUX2N	AUX2P	AUX1N	AUX1P	0000,0000
PWMB_ETRPS	PWMB 的 ETR 选择寄存器	FEB4H						BRKBPS	ETRBPS[1:0]		xxxx,x000
PWMB_ENO	PWMB 输出使能控制	FEB5H	-	ENO8P	-	ENO7P	-	ENO6P	-	ENO5P	x0x0,x0x0
PWMB_PS	PWMB 输出脚选择寄存器	FEB6H	C8PS[1:0]		C7PS[1:0]		C6PS[1:0]		C5PS[1:0]		0000,0000
PWMB_IOAUX	PWMB 辅助寄存器	FEB7H	-	AUX8P	-	AUX7P	-	AUX6P	-	AUX5P	x0x0,x0x0
PWMA_CR1	PWMA 控制寄存器 1	FEC0H	ARPE	CMS[1:0]		DIR	OPM	URS	UDIS	CEN	0000,0000
PWMA_CR2	PWMA 控制寄存器 2	FEC1H	-	MMS[2:0]			-	COMS	-	CCPC	x000,x0x0
PWMA_SMCR	PWMA 从模式控制寄存器	FEC2H	MSM	TS[2:0]			-	SMS[2:0]			0000,x000
PWMA_ETR	PWMA 外部触发寄存器	FEC3H	ETP	ECE	ETPS[1:0]		ETF[3:0]				0000,0000
PWMA_IER	PWMA 中断使能寄存器	FEC4H	BIE	TIE	COMIE	CC4IE	CC3IE	CC2IE	CC1IE	UIE	0000,0000
PWMA_SR1	PWMA 状态寄存器 1	FEC5H	BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	UIF	0000,0000
PWMA_SR2	PWMA 状态寄存器 2	FEC6H	-	-	-	CC4OF	CC3OF	CC2OF	CC1OF	-	xxx0,000x
PWMA_EGR	PWMA 事件发生寄存器	FEC7H	BG	TG	COMG	CC4G	CC3G	CC2G	CC1G	UG	0000,0000
PWMA_CCMR1	PWMA 捕获模式寄存器 1	FEC8H	OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]		0000,0000
	PWMA 比较模式寄存器 1		IC1F[3:0]				IC1PSC[1:0]		CC1S[1:0]		0000,0000
PWMA_CCMR2	PWMA 捕获模式寄存器 2	FEC9H	OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]		0000,0000
	PWMA 比较模式寄存器 2		IC2F[3:0]				IC2PSC[1:0]		CC2S[1:0]		0000,0000
PWMA_CCMR3	PWMA 捕获模式寄存器 3	FECAH	OC3CE	OC3M[2:0]			OC3PE	OC3FE	CC3S[1:0]		0000,0000
	PWMA 比较模式寄存器 3		IC3F[3:0]				IC3PSC[1:0]		CC3S[1:0]		0000,0000
PWMA_CCMR4	PWMA 捕获模式寄存器 4	FECBH	OC4CE	OC4M[2:0]			OC4PE	OC4FE	CC4S[1:0]		0000,0000
	PWMA 比较模式寄存器 4		IC4F[3:0]				IC4PSC[1:0]		CC4S[1:0]		0000,0000
PWMA_CCER1	PWMA 捕获比较使能寄存器 1	FECCH	CC2NP	CC2NE	CC2P	CC2E	CC1NP	CC1NE	CC1P	CC1E	0000,0000
PWMA_CCER2	PWMA 捕获比较使能寄存器 2	FECDH	CC4NP	CC4NE	CC4P	CC4E	CC3NP	CC3NE	CC3P	CC3E	0000,0000
PWMA_CNTRH	PWMA 计数器高字节	FECEH									0000,0000

STCMCU

PWMA_CNTRL	PWMA 计数器低字节	FECFH									0000,0000
PWMA_PSCRH	PWMA 预分频高字节	FED0H									0000,0000
PWMA_PSCRL	PWMA 预分频低字节	FED1H									0000,0000
PWMA_ARRH	PWMA 自动重装寄存器高字节	FED2H									0000,0000
PWMA_ARRL	PWMA 自动重装寄存器低字节	FED3H									0000,0000
PWMA_RCR	PWMA 重复计数器寄存器	FED4H									0000,0000
PWMA_CCR1H	PWMA 比较捕获寄存器 1 高位	FED5H									0000,0000
PWMA_CCR1L	PWMA 比较捕获寄存器 1 低位	FED6H									0000,0000
PWMA_CCR2H	PWMA 比较捕获寄存器 2 高位	FED7H									0000,0000
PWMA_CCR2L	PWMA 比较捕获寄存器 2 低位	FED8H									0000,0000
PWMA_CCR3H	PWMA 比较捕获寄存器 3 高位	FED9H									0000,0000
PWMA_CCR3L	PWMA 比较捕获寄存器 3 低位	FEDAH									0000,0000
PWMA_CCR4H	PWMA 比较捕获寄存器 4 高位	FEDBH									0000,0000
PWMA_CCR4L	PWMA 比较捕获寄存器 4 低位	FEDCH									0000,0000
PWMA_BKR	PWMA 刹车寄存器	FEDDH	MOE	AOE	BKP	BKE	OSSR	OSSI	LOCK[1:0]		0000,000x
PWMA_DTR	PWMA 死区控制寄存器	FEDEH	DTG[7:0]								0000,0000
PWMA_OISR	PWMA 输出空闲状态寄存器	FEDFH	OIS4N	OIS4	OIS3N	OIS3	OIS2N	OIS2	OIS1N	OIS1	0000,0000
PWMB_CR1	PWMB 控制寄存器 1	FEE0H	ARPE	CMS[1:0]		DIR	OPM	URS	UDIS	CEN	0000,0000
PWMB_CR2	PWMB 控制寄存器 2	FEE1H	-	MMS[2:0]			-	COMS	-	CCPC	x000,x0x0
PWMB_SMCR	PWMB 从模式控制寄存器	FEE2H	MSM	TS[2:0]			-	SMS[2:0]			0000,x000
PWMB_ETR	PWMB 外部触发寄存器	FEE3H	ETP	ECE	ETPS[1:0]		ETF[3:0]				0000,0000
PWMB_IER	PWMB 中断使能寄存器	FEE4H	BIE	TIE	COMIE	CC8IE	CC7IE	CC6IE	CC5IE	UIE	0000,0000
PWMB_SR1	PWMB 状态寄存器 1	FEE5H	BIF	TIF	COMIF	CC8IF	CC7IF	CC6IF	CC5IF	UIF	0000,0000
PWMB_SR2	PWMB 状态寄存器 2	FEE6H	-	-	-	CC8OF	CC7OF	CC6OF	CC5OF	-	xxx0,000x
PWMB_EGR	PWMB 事件发生寄存器	FEE7H	BG	TG	COMG	CC8G	CC7G	CC6G	CC5G	UG	0000,0000
PWMB_CCMR1	PWMB 捕获模式寄存器 1	FEE8H	OC5CE	OC5M[2:0]			OC5PE	OC5FE	CC5S[1:0]		0000,0000
	PWMB 比较模式寄存器 1		IC5F[3:0]				IC5PSC[1:0]		CC5S[1:0]		0000,0000
PWMB_CCMR2	PWMB 捕获模式寄存器 2	FEE9H	OC6CE	OC6M[2:0]			OC6PE	OC6FE	CC6S[1:0]		0000,0000
	PWMB 比较模式寄存器 2		IC6F[3:0]				IC6PSC[1:0]		CC6S[1:0]		0000,0000
PWMB_CCMR3	PWMB 捕获模式寄存器 3	FEEAH	OC7CE	OC7M[2:0]			OC7PE	OC7FE	CC7S[1:0]		0000,0000
	PWMB 比较模式寄存器 3		IC7F[3:0]				IC7PSC[1:0]		CC7S[1:0]		0000,0000
PWMB_CCMR4	PWMB 捕获模式寄存器 4	FEEBH	OC8CE	OC8M[2:0]			OC8PE	OC8FE	CC8S[1:0]		0000,0000
	PWMB 比较模式寄存器 4		IC8F[3:0]				IC8PSC[1:0]		CC8S[1:0]		0000,0000
PWMB_CCER1	PWMB 捕获比较使能寄存器 1	FEECH	-	-	CC6P	CC6E	-	-	CC5P	CC5E	xx00,xx00
PWMB_CCER2	PWMB 捕获比较使能寄存器 2	FEEDH	-	-	CC8P	CC8E	-	-	CC7P	CC7E	xx00,xx00
PWMB_CNTRH	PWMB 计数器高字节	FEEEH									0000,0000
PWMB_CNTRL	PWMB 计数器低字节	FEEFH									0000,0000
PWMB_PSCRH	PWMB 预分频高字节	FEF0H									0000,0000
PWMB_PSCRL	PWMB 预分频低字节	FEF1H									0000,0000
PWMB_ARRH	PWMB 自动重装寄存器高字节	FEF2H									0000,0000
PWMB_ARRL	PWMB 自动重装寄存器低字节	FEF3H									0000,0000
PWMB_RCR	PWMB 重复计数器寄存器	FEF4H									0000,0000
PWMB_CCR5H	PWMB 比较捕获寄存器 1 高位	FEF5H									0000,0000
PWMB_CCR5L	PWMB 比较捕获寄存器 1 低位	FEF6H									0000,0000

STC MCU

PWMB_CCR6H	PWMB 比较捕获寄存器 2 高位	FEF7H									0000,0000
PWMB_CCR6L	PWMB 比较捕获寄存器 2 低位	FEF8H									0000,0000
PWMB_CCR7H	PWMB 比较捕获寄存器 3 高位	FEF9H									0000,0000
PWMB_CCR7L	PWMB 比较捕获寄存器 3 低位	FEFAH									0000,0000
PWMB_CCR8H	PWMB 比较捕获寄存器 4 高位	FEFBH									0000,0000
PWMB_CCR8L	PWMB 比较捕获寄存器 4 低位	FEFCH									0000,0000
PWMB_BKR	PWMB 刹车寄存器	FEFDH	MOE	AOE	BKP	BKE	OSSR	OSSI	LOCK[1:0]	-	0000,000x
PWMB_DTR	PWMB 死区控制寄存器	FEFEH	DTG[7:0]								0000,0000
PWMB_OISR	PWMB 输出空闲状态寄存器	FEFFH	-	OIS8	-	OIS7	-	OIS6	-	OIS5	x0x0,x0x0
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	0000,0000
P0INTE	P0 口中断使能寄存器	FD00H	P07INTE	P06INTE	P05INTE	P04INTE	P03INTE	P02INTE	P01INTE	P00INTE	0000,0000
P1INTE	P1 口中断使能寄存器	FD01H	P17INTE	P16INTE	P15INTE	P14INTE	P13INTE	P12INTE	P11INTE	P10INTE	0000,0000
P2INTE	P2 口中断使能寄存器	FD02H	P27INTE	P26INTE	P25INTE	P24INTE	P23INTE	P22INTE	P21INTE	P20INTE	0000,0000
P3INTE	P3 口中断使能寄存器	FD03H	P37INTE	P36INTE	P35INTE	P34INTE	P33INTE	P32INTE	P31INTE	P30INTE	0000,0000
P5INTE	P5 口中断使能寄存器	FD05H	P57INTE	P56INTE	P55INTE	P54INTE	P53INTE	P52INTE	P51INTE	P50INTE	0000,0000
P0INTF	P0 口中断标志寄存器	FD10H	P07INTF	P06INTF	P05INTF	P04INTF	P03INTF	P02INTF	P01INTF	P00INTF	0000,0000
P1INTF	P1 口中断标志寄存器	FD11H	P17INTF	P16INTF	P15INTF	P14INTF	P13INTF	P12INTF	P11INTF	P10INTF	0000,0000
P2INTF	P2 口中断标志寄存器	FD12H	P27INTF	P26INTF	P25INTF	P24INTF	P23INTF	P22INTF	P21INTF	P20INTF	0000,0000
P3INTF	P3 口中断标志寄存器	FD13H	P37INTF	P36INTF	P35INTF	P34INTF	P33INTF	P32INTF	P31INTF	P30INTF	0000,0000
P5INTF	P5 口中断标志寄存器	FD15H	P57INTF	P56INTF	P55INTF	P54INTF	P53INTF	P52INTF	P51INTF	P50INTF	0000,0000
P0IM0	P0 口中断模式寄存器 0	FD20H	P07IM0	P06IM0	P05IM0	P04IM0	P03IM0	P02IM0	P01IM0	P00IM0	0000,0000
P1IM0	P1 口中断模式寄存器 0	FD21H	P17IM0	P16IM0	P15IM0	P14IM0	P13IM0	P12IM0	P11IM0	P10IM0	0000,0000
P2IM0	P2 口中断模式寄存器 0	FD22H	P27IM0	P26IM0	P25IM0	P24IM0	P23IM0	P22IM0	P21IM0	P20IM0	0000,0000
P3IM0	P3 口中断模式寄存器 0	FD23H	P37IM0	P36IM0	P35IM0	P34IM0	P33IM0	P32IM0	P31IM0	P30IM0	0000,0000
P5IM0	P5 口中断模式寄存器 0	FD25H	P57IM0	P56IM0	P55IM0	P54IM0	P53IM0	P52IM0	P51IM0	P50IM0	0000,0000
P0IM1	P0 口中断模式寄存器 1	FD30H	P07IM1	P06IM1	P05IM1	P04IM1	P03IM1	P02IM1	P01IM1	P00IM1	0000,0000
P1IM1	P1 口中断模式寄存器 1	FD31H	P17IM1	P16IM1	P15IM1	P14IM1	P13IM1	P12IM1	P11IM1	P10IM1	0000,0000
P2IM1	P2 口中断模式寄存器 1	FD32H	P27IM1	P26IM1	P25IM1	P24IM1	P23IM1	P22IM1	P21IM1	P20IM1	0000,0000
P3IM1	P3 口中断模式寄存器 1	FD33H	P37IM1	P36IM1	P35IM1	P34IM1	P33IM1	P32IM1	P31IM1	P30IM1	0000,0000
P5IM1	P5 口中断模式寄存器 1	FD35H	P57IM1	P56IM1	P55IM1	P54IM1	P53IM1	P52IM1	P51IM1	P50IM1	0000,0000
CCAPM3	PCA 模块 3 模式控制寄存器	FD54H	-	ECOM3	CCAPP3	CCAPN3	MAT3	TOG3	PWM3	ECCF3	x000,0000
CCAP3L	PCA 模块 3 低字节	FD55H									0000,0000
CCAP3H	PCA 模块 3 高字节	FD56H									0000,0000
PCA_PWM3	PCA3 的 PWM 模式寄存器	FD57H	EBS3[1:0]		XCCAP3H[1:0]		XCCAP3L[1:0]		EPC3H	EPC3L	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

STC MCU

9 I/O 口

STC12H 系列单片机所有的 I/O 口均有 4 种工作模式：准双向口/弱上拉（标准 8051 输出口模式）、推挽输出/强上拉、高阻输入（电流既不能流入也不能流出）、开漏输出。可使用软件对 I/O 口的工作模式进行配置。

注意：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

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
P5	P5 端口	C8H	P57	P56	P55	P54	P53	P52	P51	P50	1111,1111
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	1111,1100
P3M0	P3 口配置寄存器 0	B2H	P37M0	P36M0	P35M0	P34M0	P33M0	P32M0	P31M0	P30M0	0000,0000
P5M1	P5 口配置寄存器 1	C9H	P57M1	P56M1	P55M1	P54M1	P53M1	P52M1	P51M1	P50M1	1111,1111
P5M0	P5 口配置寄存器 0	CAH	P57M0	P56M0	P55M0	P54M0	P53M0	P52M0	P51M0	P50M0	0000,0000

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
P0PU	P0 口上拉电阻控制寄存器	FE10H	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
P5PU	P5 口上拉电阻控制寄存器	FE15H	P57PU	P56PU	P55PU	P54PU	P53PU	P52PU	P51PU	P50PU	0000,0000
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
P5NCS	P5 口施密特触发控制寄存器	FE1DH	P57NCS	P56NCS	P55NCS	P54NCS	P53NCS	P52NCS	P51NCS	P50NCS	0000,0000
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

STC MCU

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
P5SR	P5 口电平转换速率寄存器	FE25H	P57SR	P56SR	P55SR	P54SR	P53SR	P52SR	P51SR	P50SR	1111,1111
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
P5DR	P5 口驱动电流控制寄存器	FE2DH	P57DR	P56DR	P55DR	P54DR	P53DR	P52DR	P51DR	P50DR	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
P2IE	P2 口输入使能控制寄存器	FE32H	-	P26IE	P25IE	P24IE	P23IE	P22IE	P21IE	P20IE	x111,1111

STC MCU

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
P5	C8H	P5.7	P5.6	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
P5M0	CAH	P57M0	P56M0	P55M0	P54M0	P53M0	P52M0	P51M0	P50M0
P5M1	C9H	P57M1	P56M1	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
P5PU	FE15H	P57PU	P56PU	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
P5NCS	FE1DH	P57NCS	P56NCS	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	复位值
P0SR	FE20H	P07SR	P06SR	P05SR	P04SR	P03SR	P02SR	P01SR	P00SR	1111,1111
P1SR	FE21H	P17SR	P16SR	P15SR	P14SR	P13SR	P12SR	P11SR	P10SR	1111,1111
P2SR	FE22H	P27SR	P26SR	P25SR	P24SR	P23SR	P22SR	P21SR	P20SR	1111,1111
P3SR	FE23H	P37SR	P36SR	P35SR	P34SR	P33SR	P32SR	P31SR	P30SR	1111,1111
P5SR	FE25H	-	-	-	P54SR	P53SR	P52SR	P51SR	P50SR	1111,1111

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	复位值
P0DR	FE28H	P07DR	P06DR	P05DR	P04DR	P03DR	P02DR	P01DR	P00DR	1111,1111
P1DR	FE29H	P17DR	P16DR	P15DR	P14DR	P13DR	P12DR	P11DR	P10DR	1111,1111
P2DR	FE2AH	P27DR	P26DR	P25DR	P24DR	P23DR	P22DR	P21DR	P20DR	1111,1111
P3DR	FE2BH	P37DR	P36DR	P35DR	P34DR	P33DR	P32DR	P31DR	P30DR	1111,1111
P5DR	FE2DH	-	-	-	P54DR	P53DR	P52DR	P51DR	P50DR	1111,1111

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
P1IE	FE31H	P17IE	P16IE	P15IE	P14IE	P13IE	P12IE	P11IE	P10IE
P2IE	FE32H	-	P26IE	P25IE	P24IE	P23IE	P22IE	P21IE	P20IE

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。

STC MCU

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。

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。

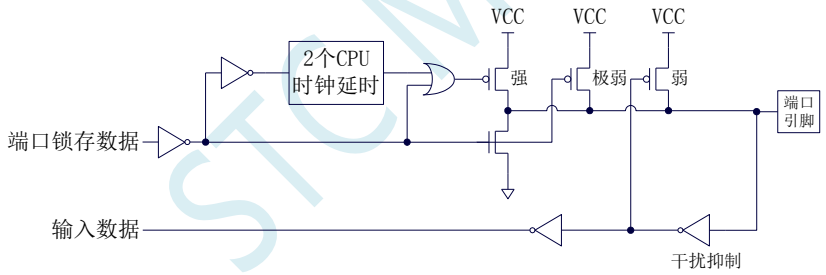
在 3 个上拉晶体管中，有 1 个上拉晶体管称为“弱上拉”，当端口寄存器为 1 且引脚本身也为 1 时打开。此上拉提供基本驱动电流使准双向口输出为 1。如果一个引脚输出为 1 而由外部装置下拉到低时，弱上拉关闭而“极弱上拉”维持开状态，为了把这个引脚强拉为低，外部装置必须有足够的灌电流能力使引脚上的电压降到门槛电压以下。对于 5V 单片机，“弱上拉”晶体管的电流约 250uA；对于 3.3V 单片机，“弱上拉”晶体管的电流约 150uA。

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.

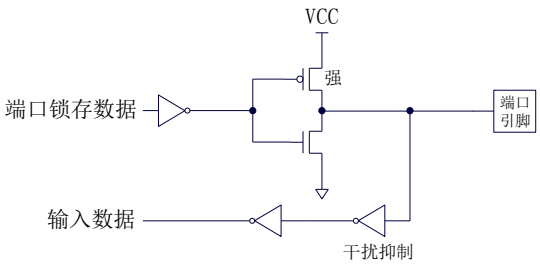
准双向口（弱上拉）输出如下图所示：



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：

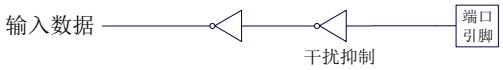


9.3.3 高阻输入

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：



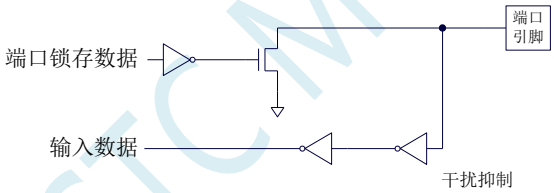
9.3.4 开漏输出

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。

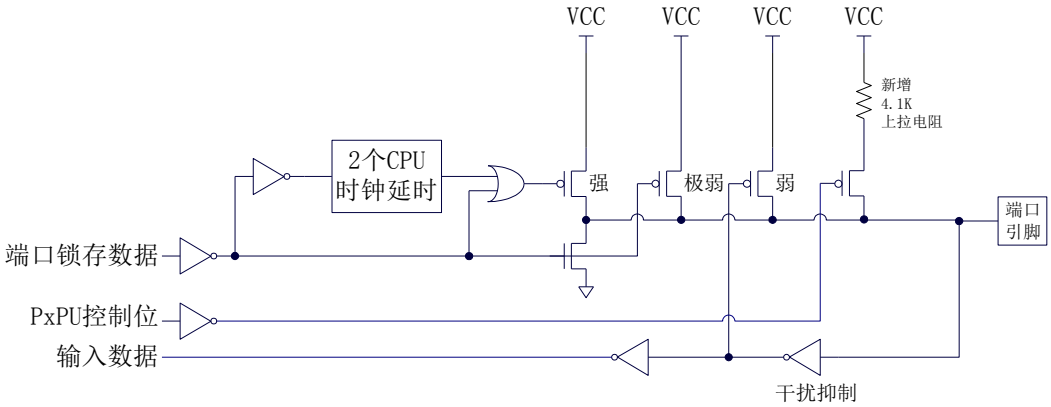
输出端口配置如下图所示：



STCMCU

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
P5PU	FE15H	P57PU	P56PU	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

STC MCU

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     P0M0    =   0x94;
sfr     P0M1    =   0x93;
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    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);
}
```

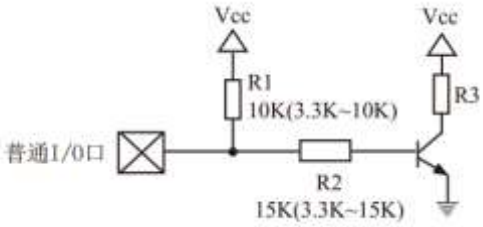
汇编代码

```
;测试工作频率为 11.0592MHz

P0M0        DATA        094H
P0M1        DATA        093H
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     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
```

STC MCU

9.5 一种典型三极管控制电路

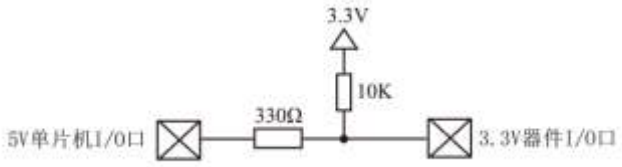


如果上拉控制，建议加上拉电阻 R1(3.3K～10K)，如果不加上拉电阻 R1(3.3K～10K)，建议 R2 的值在 15K 以上，或用强推挽输出。

9.6 典型发光二极管控制电路

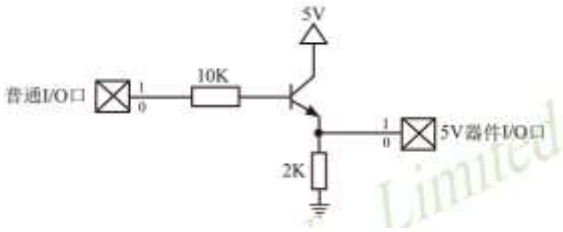
9.7 混合电压供电系统 3V/5V 器件 I/O 口互连

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



STC 的 3V 单片机连接 5V 器件时，为防止 3V 单片机承受不了 5V，如果相应的 I/O 口是输入，可在该 I/O 口上串接一个隔离二极管，隔离高压部分。外部信号电压高于单片机工作电压时截止，I/O 口因内内部上拉到高电平，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。

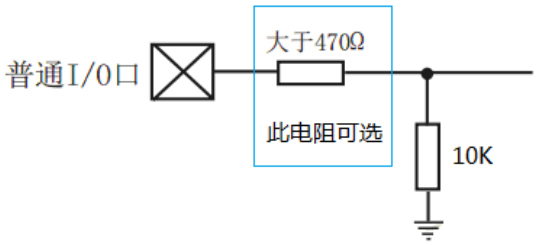
STC 的 3V 单片机连接 5V 器件时，为防止 3V 单片机承受不了 5V，如果相应的 I/O 口是输出，可用一个 NPN 三极管隔离，电路如下：



9.8 如何让 I/O 口上电复位时为低电平

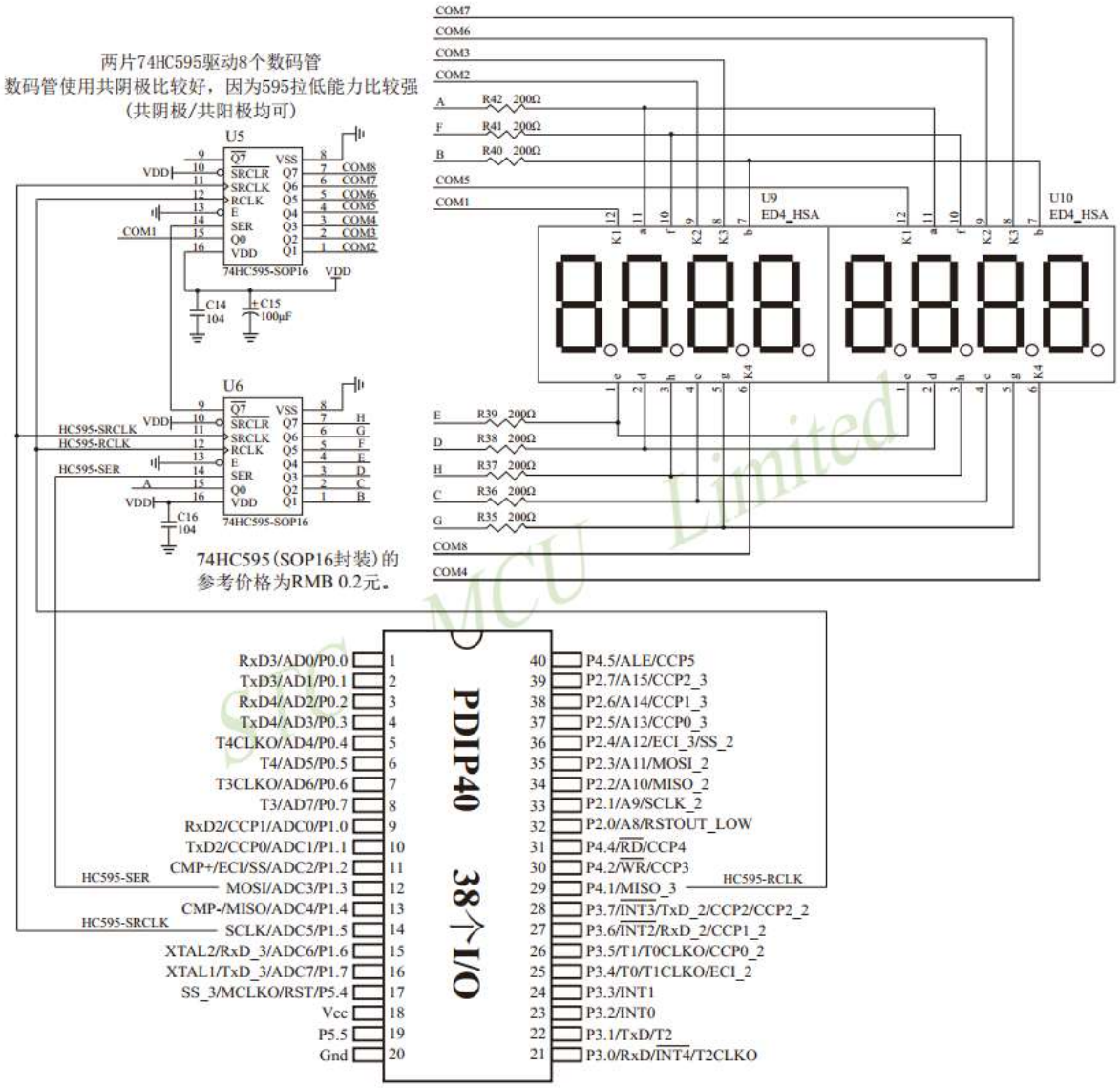
普通 8051 单片机上电复位时普通 I/O 口为弱上拉(准双向口)高电平输出，而很多实际应用要求上电时某些 I/O 口为低电平输出，否则所控制的系统(如马达)就会误动作，现 STC 的单片机由于既有弱上拉输出又有强推挽输出，就可以很轻松的解决此问题。

现可在 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。

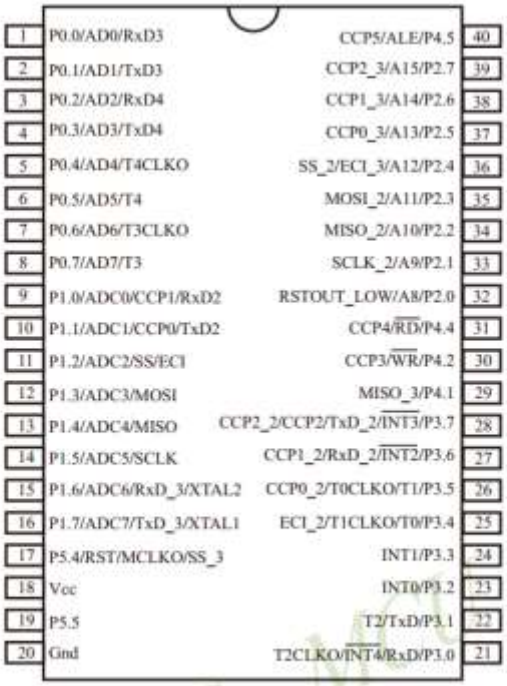


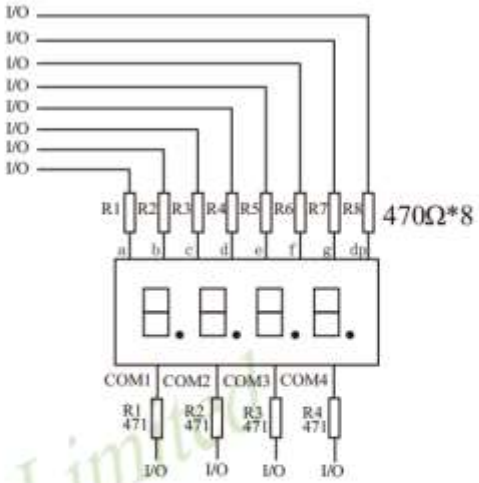
STC MCU

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

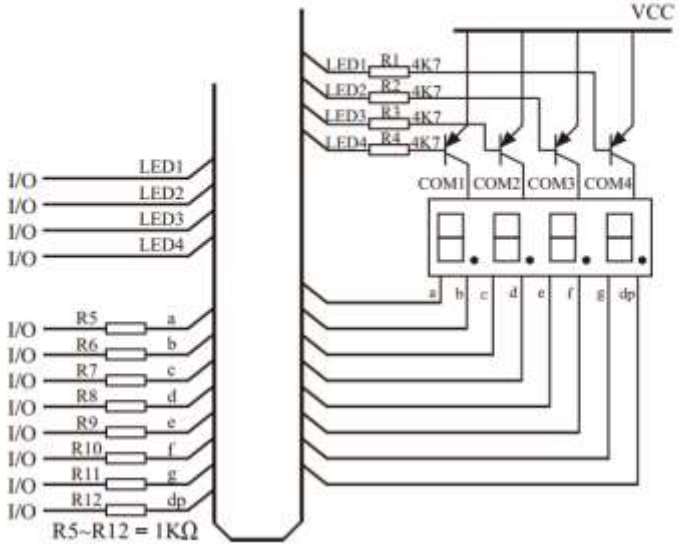




I/O口动态扫描驱动4个共阴极数码管参考电路图

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

I/O 口动态扫描驱动4个共阳极数码管参考电路图



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

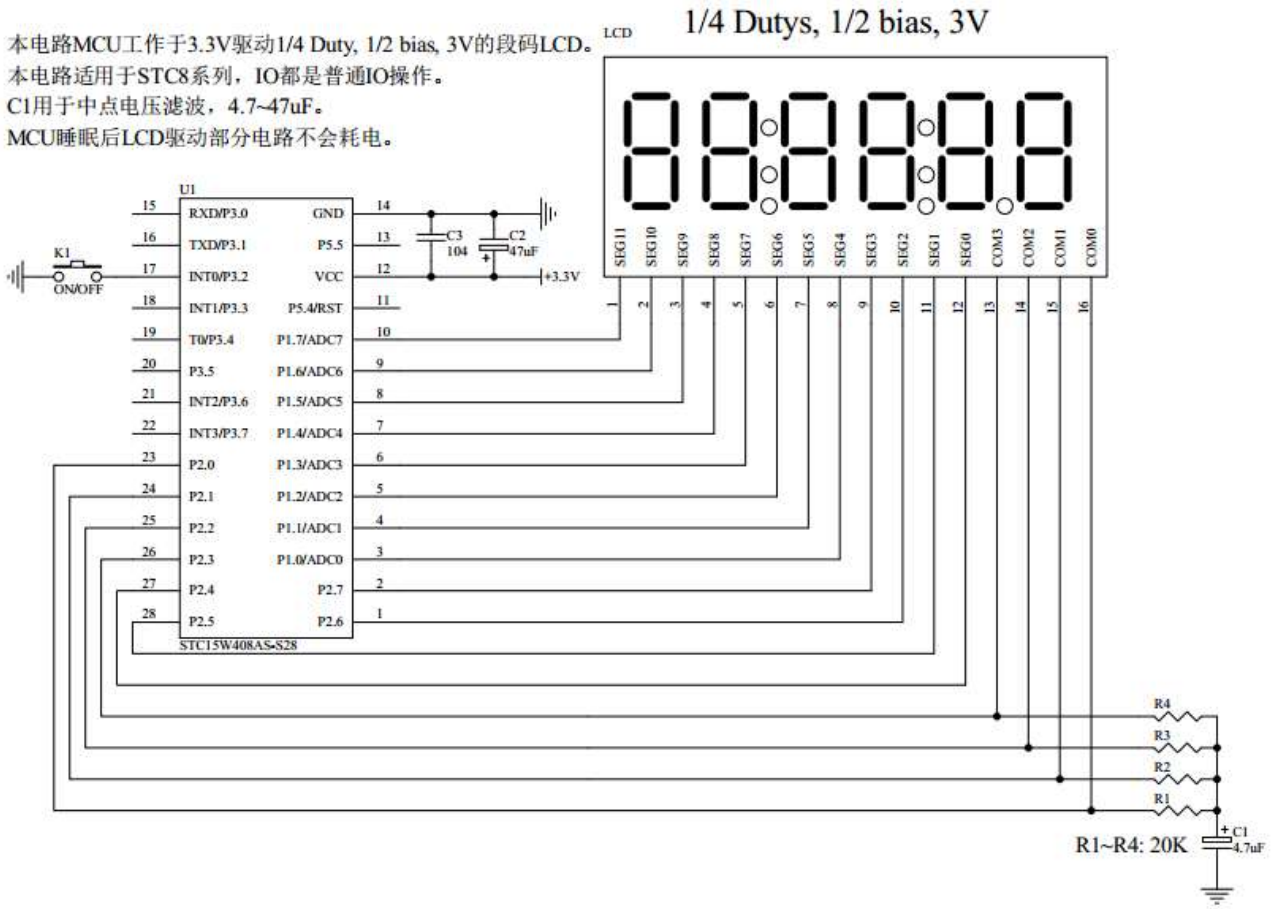


图 2：段码名称图

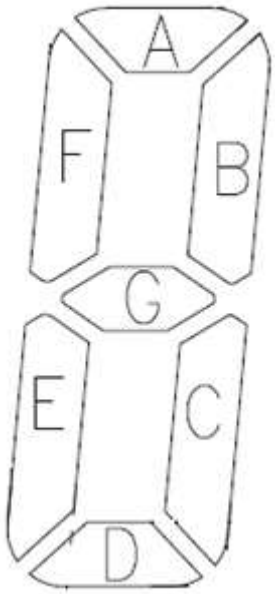


图 3：1/4Duty 1/2BIAS 扫描原理图

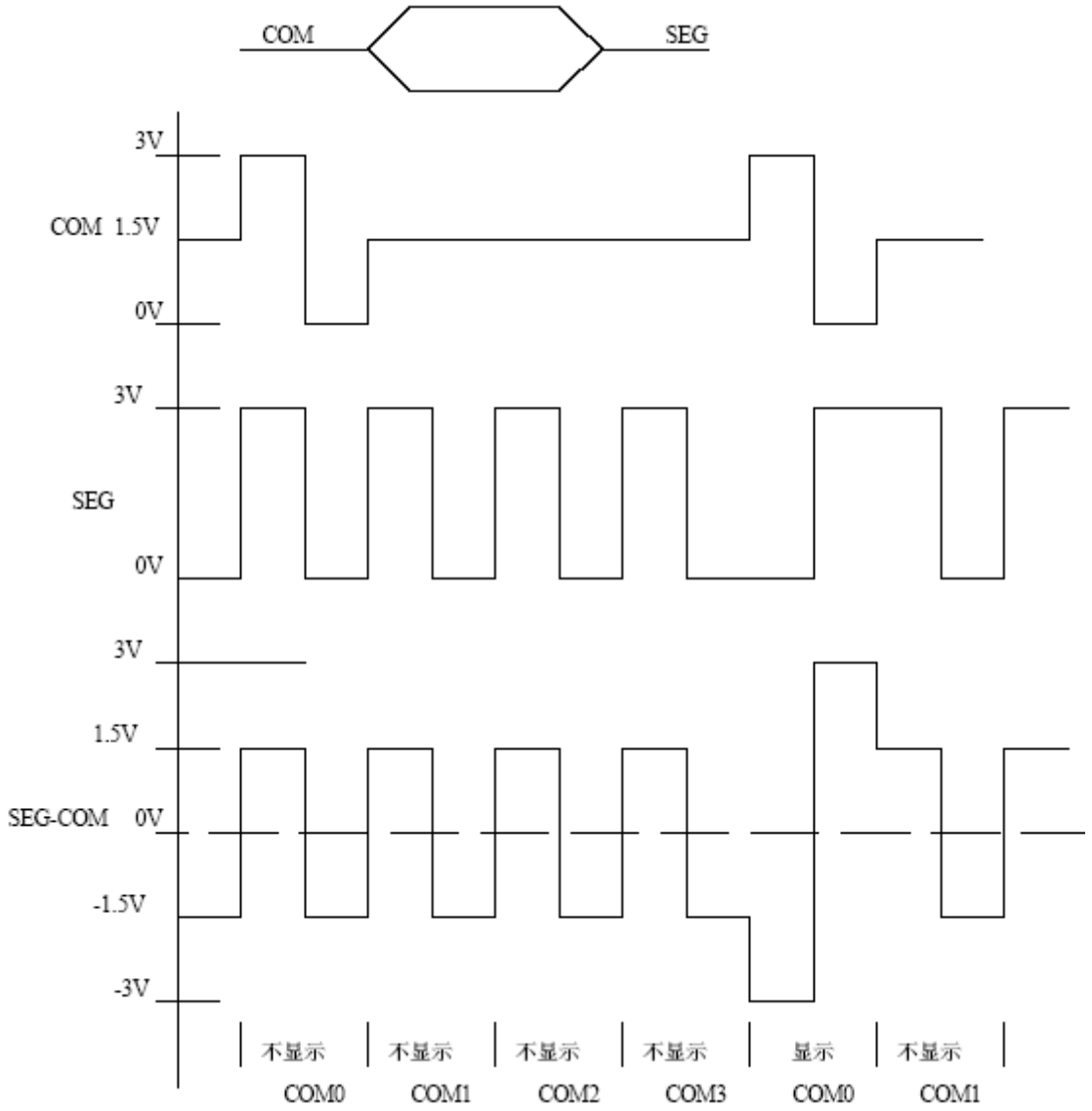


图 4：驱动 1/4Duty 1/3BIAS 3V LCD 的电路

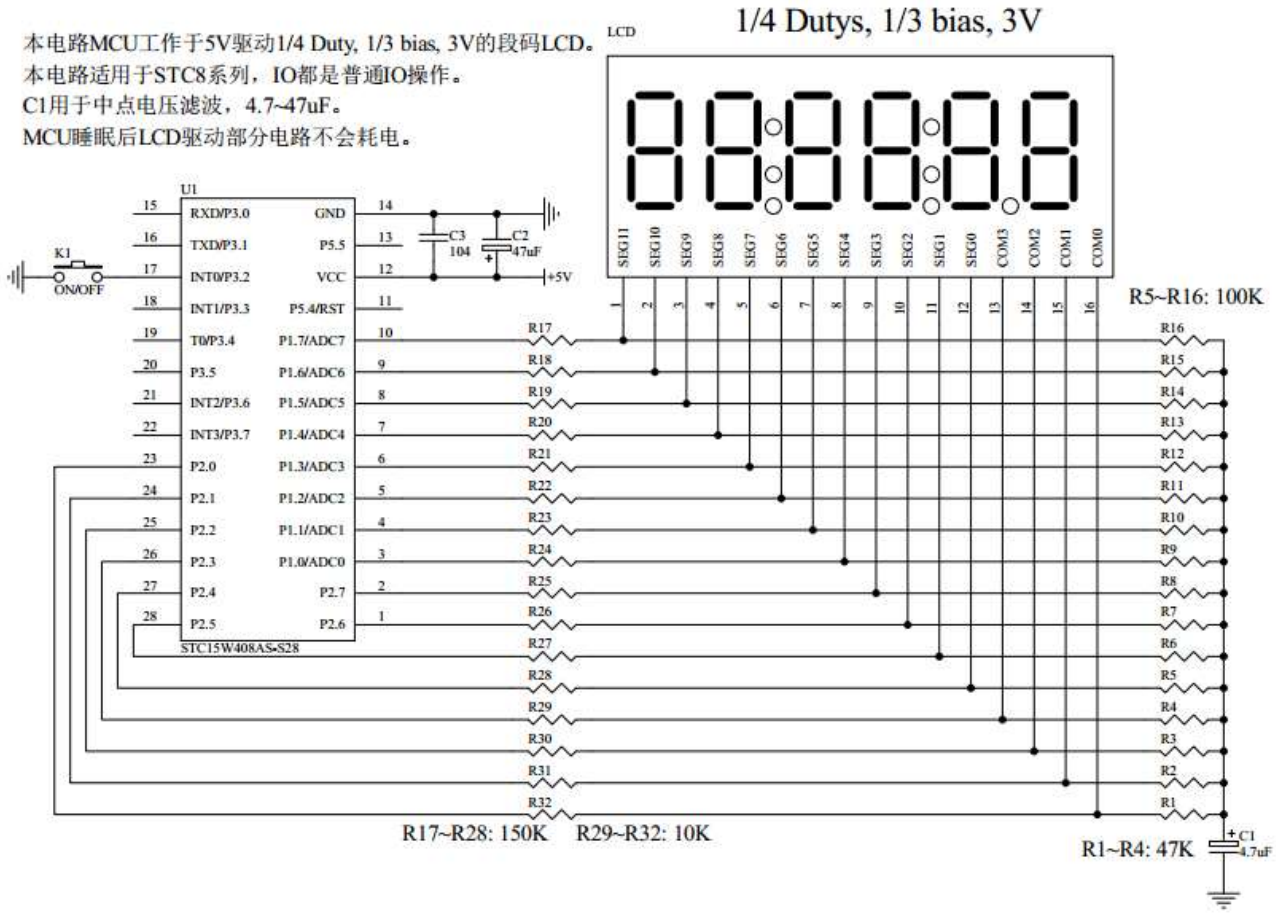
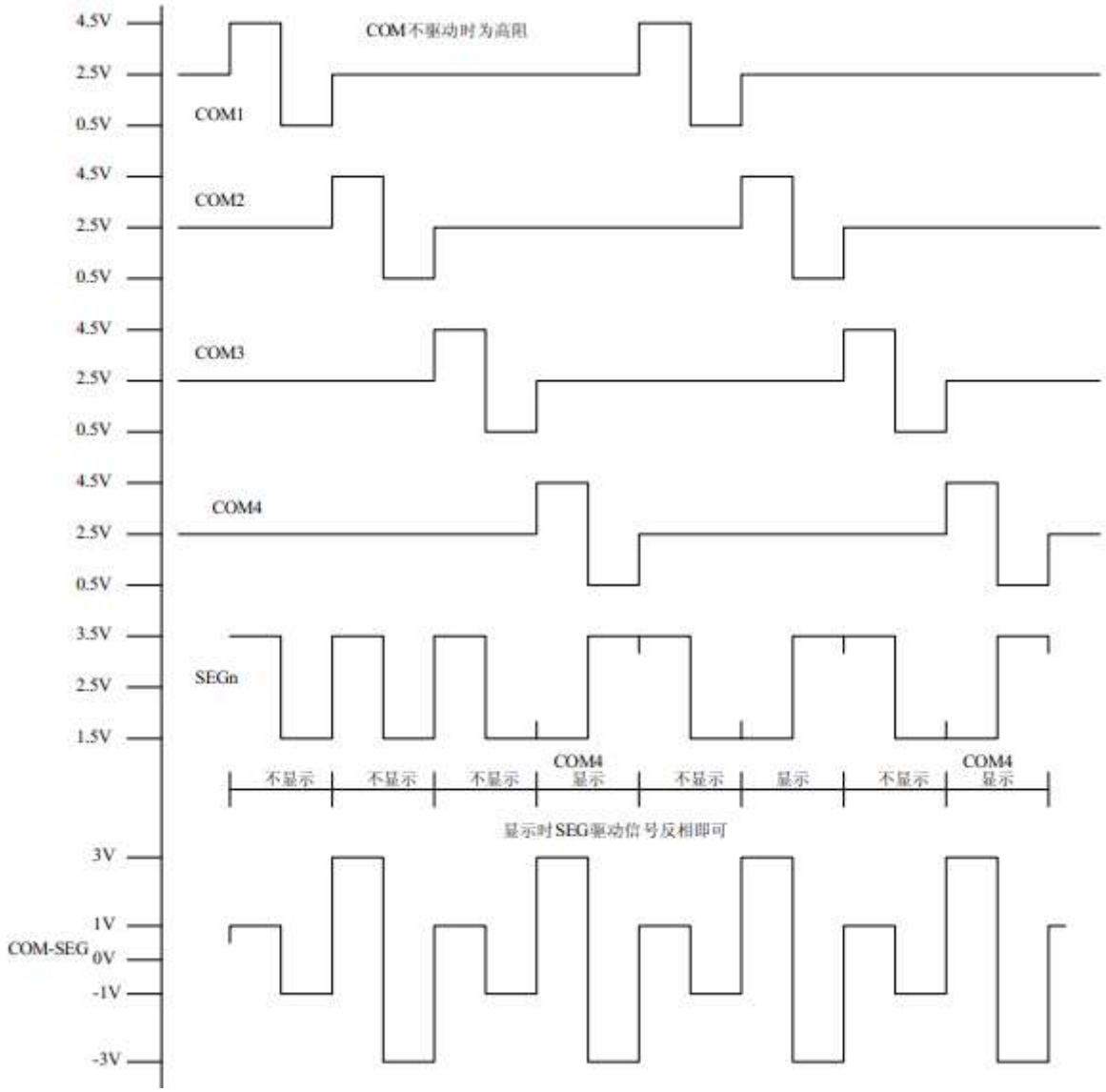


图 5：1/4Duty 1/3BIAS 扫描原理图



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

图 6：LCD 真值表和显存影射表

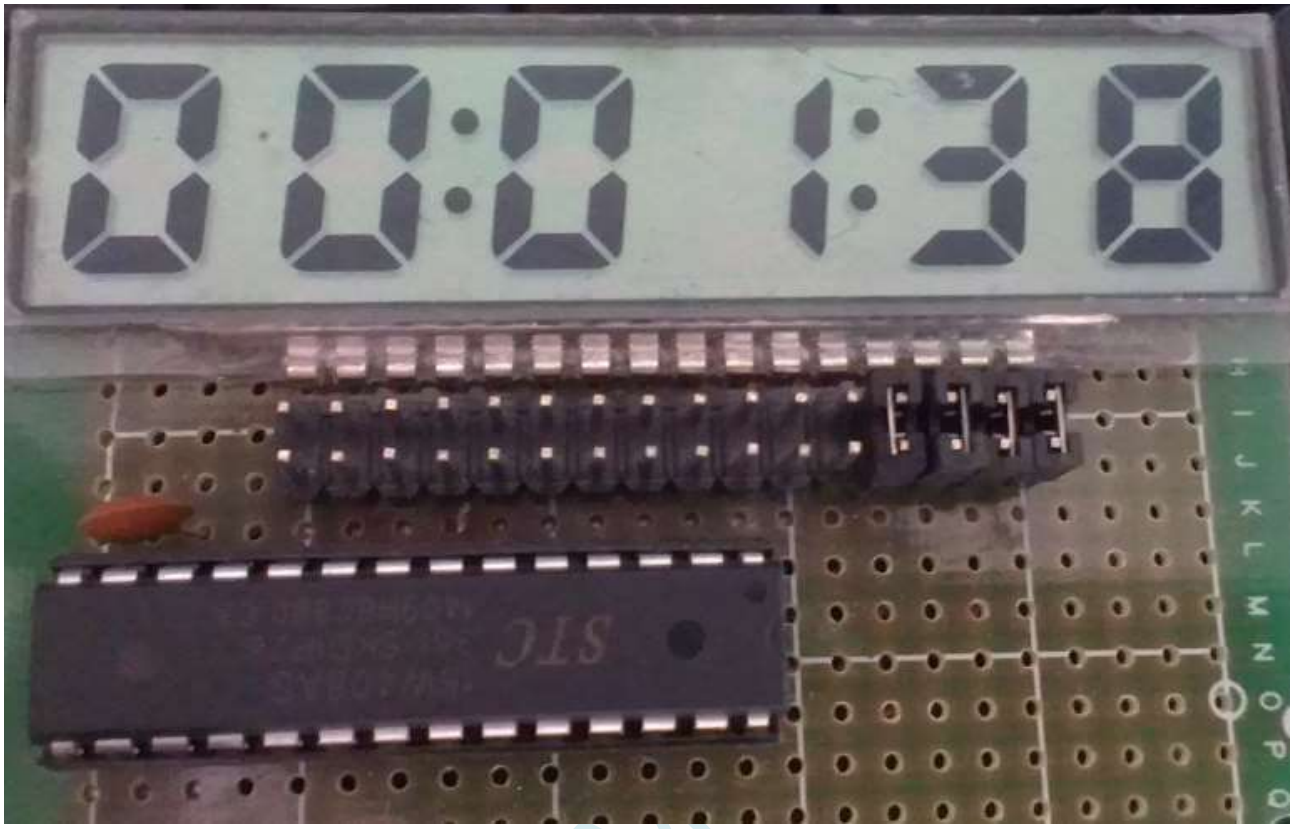
LCD真值表：

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

显存影射表：

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

图 7：驱动效果照片



本 LCD 扫描程序仅需要两个函数：

1、 LCD 段码扫描函数 void LCD_scan(void)

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

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

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

另外，用宏来显示、熄灭或闪烁冒号或小数点。

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 Dutys, 1/3 bias)。
上电后显示一个时间(时分秒).
P3.2 对地接一个开关,用来进入睡眠或唤醒.
*****************************************/

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

typedef    unsigned char    u8;
typedef    unsigned int     u16;
typedef    unsigned long    u32;

sfr AUXR = 0x8e;
```

```
sfr P1M1 = 0x91;
sfr P1M0 = 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 P1n_standard(bitn)      P1M1 &= ~(bitn), P1M0 &= ~(bitn)
#define P1n_push_pull(bitn)     P1M1 &= ~(bitn), P1M0 |= (bitn)
#define P1n_pure_input(bitn)    P1M1 |= (bitn), P1M0 &= ~(bitn)
#define P1n_open_drain(bitn)    P1M1 |= (bitn), P1M0 |= (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);
    P1n_push_pull(0xff);                            //segment 设置为推挽输出

    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);                               //全部COM 输出高阻, COM 为中点电压
    if(scan_index & 1)                                  //反相扫描
    {
        P1 = ~LCD_buff[j];
        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]);                            //某个COM 设置为推挽输出
    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_mask4[n]) | (k>>2*n);
    p++;
```

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

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

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

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

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     R0
            DJNZ    R2, L_ClearLcdRam

            LCALL   F_Timer0_init
            SETB    EA
```

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

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

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

                INC     cnt_500ms
                MOV     A, cnt_500ms
                CJNE    A, #250, L_Main_Loop
                                                ;500ms 到
                MOV     cnt_500ms, #0;

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

                CPL     B_Second
                JNB     B_Second, L_Main_Loop

                INC     second
                MOV     A, second
                CJNE    A, #60, L_Main_Load
                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             ; 1T mode
                MOV     TH0, #HIGH (-22118)     ; 2ms
                MOV     TL0, #LOW   (-22118)    ;
                SETB    TR0                     ; 开始运行
                RET

;*************************************************************
F_Timer0_Interrupt:     ;Timer0 1ms 中断函数
                PUSH    PSW                     ;PSW 入栈
                PUSH    ACC                     ;ACC 入栈
```

```
        PUSH    AR0
        PUSH    AR7
        PUSH    DPH
        PUSH    DPL

        LCALL   F_LCD_scan
        SETB    B_2ms

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

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

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

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

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

        MOV     R6, #5          ;LCD_load(5,second/10);
        MOV     A, second
        MOV     B, #10
        DIV     AB
        MOV     R7, A
        LCALL   F_LCD_load      ;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:                                        ;正相扫描
                MOV         A, @R0                  ;P1 = LCD_buff[j];
                MOV         P1, A
                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, #1, L_NotLoadChar1
                MOV     R0,                     #LCD_buff
                MOV     A, @R0
                MOV     C, B.3                  ;D
                MOV     ACC.6, C
                MOV     @R0, A

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

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

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

L_NotLoadChar1:
                CJNE    A, #2, L_NotLoadChar2
                MOV     R0,#LCD_buff
                MOV     A, @R0
                MOV     C, B.3                  ;D
                MOV     ACC.4, C
                MOV     @R0, A

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

```
        MOV     @R0, A

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

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

L_NotLoadChar2:
        CJNE    A, #3, L_NotLoadChar3
        MOV     R0,#LCD_buff
        MOV     A, @R0
        MOV     C, B.3          ;D
        MOV     ACC.2, C
        MOV     @R0, A

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

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

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

L_NotLoadChar3:
        CJNE    A, #4, L_NotLoadChar4
        MOV     R0,#LCD_buff
        MOV     A, @R0
        MOV     C, B.3          ;D
        MOV     ACC.0, C
        MOV     @R0, A
```

STC MCU

```
        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
```

STC MCU

```
L_NotLoadChar5:
        CJNE    A, #6, L_NotLoadChar6
        MOV     R0,#LCD_buff+4
        MOV     A, @R0
        MOV     C, B.3          ;D
        MOV     ACC.4, C
        MOV     @R0, A

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

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

        INC     R0
        MOV     A, @R0
        MOV     C, B.0          ;A
        MOV     ACC.4, C
        MOV     C, B.5          ;F
        MOV     ACC.5, C
        MOV     @R0, A
        RET
L_NotLoadChar6:
        RET

        END
```

STC MCU

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

STC MCU

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。

STC MCU

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 STC12H 系列中断源

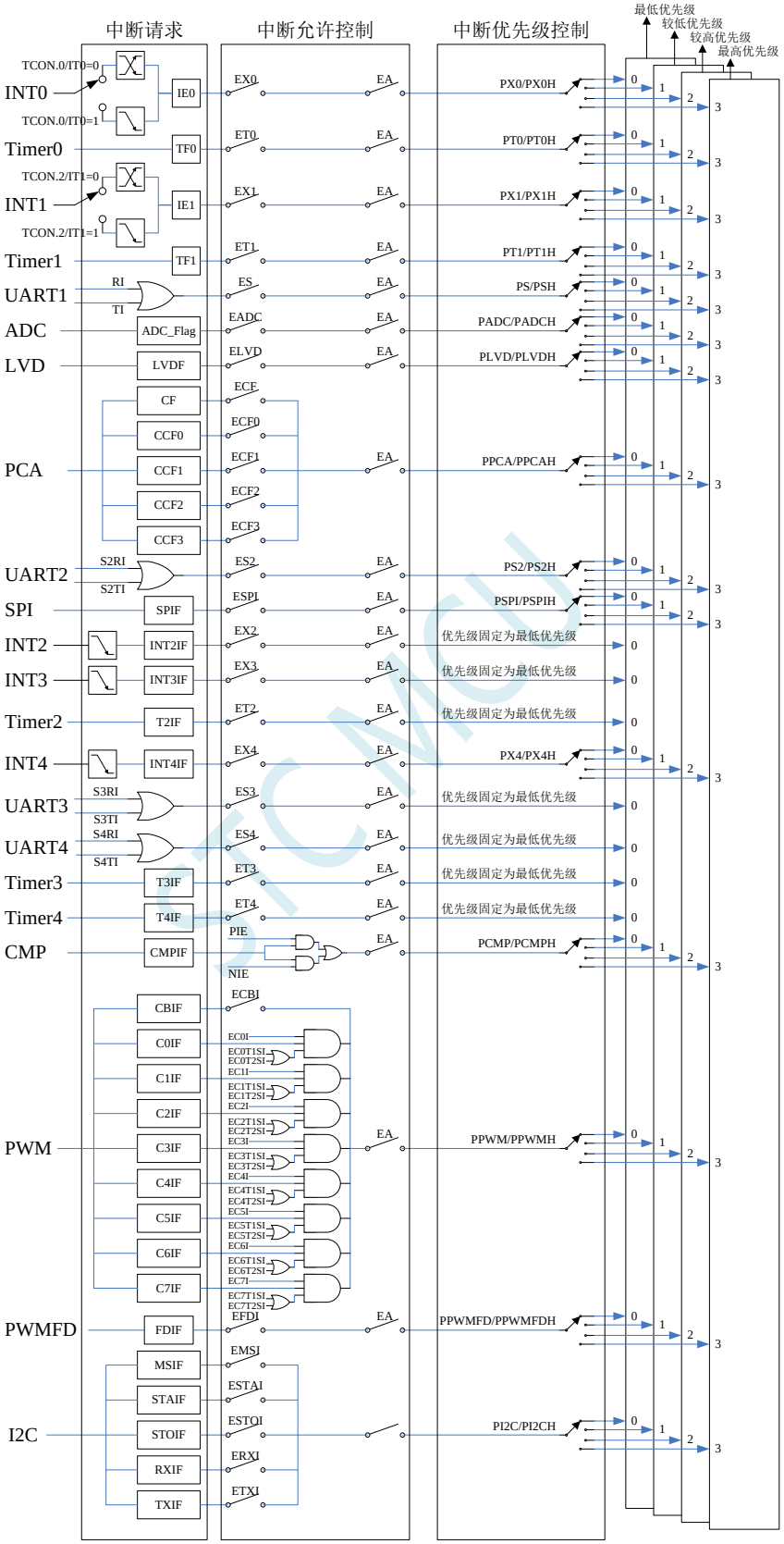
下表中 √ 表示对应的系列有相应的中断源

中断源	STC12H1K08 系列
外部中断 0 中断（INT0）	√
定时器 0 中断（Timer0）	√
外部中断 1 中断（INT1）	√
定时器 1 中断（Timer1）	√
串口 1 中断（UART1）	√
模数转换中断（ADC）	√
低压检测中断（LVD）	√
捕获中断（CCP/PCA/PWM）	√
串口 2 中断（UART2）	√
串行外设接口中断（SPI）	√
外部中断 2 中断（INT2）	√
外部中断 3 中断（INT3）	√
定时器 2 中断（Timer2）	√
外部中断 4 中断（INT4）	√

定时器 3 中断（Timer3）	√
定时器 4 中断（Timer4）	√
比较器中断（CMP）	√
I2C 总线中断	√
PWMA	√
PWMB	√
P0 口中断	√
P1 口中断	√
P2 口中断	√
P3 口中断	√
P4 口中断	
P5 口中断	√
P6 口中断	
P7 口中断	

STC MCU

11.2 STC12H 中断结构图



11.3 STC12H 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
Timer3	009BH	19		0	T3IF	ET3
Timer4	00A3H	20		0	T4IF	ET4
CMP	00ABH	21	PCMP,PCMPH	0/1/2/3	CMPIF	PIE|NIE
I2C	00C3H	24	PI2C,PI2CH	0/1/2/3	MSIF	EMSI
					STAIF	ESTAI
					RXIF	ERXI
					TXIF	ETXI
					STOIF	ESTOI

STC MCU

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
PWMA	00D3H	26	PPWMA,PPWMAH	0/1/2/3	PWMA_SR	PWMA_IER
PWMB	00DBH	27	PPWMB,PPWMBH	0/1/2/3	PWMB_SR	PWMB_IER
P0 中断	012BH	37		0	P0INTF	P0INTE
P1 中断	0133H	38		0	P1INTF	P1INTE
P2 中断	013BH	39		0	P2INTF	P2INTE
P3 中断	0143H	40		0	P3INTF	P3INTE
P5 中断	0153H	42		0	P5INTF	P5INTE

在 C 语言中声明中断服务程序

```
void    INT0_Routine(void)      interrupt  0;
void    TM0_Rountine(void)      interrupt  1;
void    INT1_Routine(void)      interrupt  2;
void    TM1_Rountine(void)      interrupt  3;
void    UART1_Routine(void)     interrupt  4;
void    ADC_Routine(void)       interrupt  5;
void    LVD_Routine(void)       interrupt  6;
void    PCA_Routine(void)       interrupt  7;
void    UART2_Routine(void)     interrupt  8;
void    SPI_Routine(void)       interrupt  9;
void    INT2_Routine(void)      interrupt  10;
void    INT3_Routine(void)      interrupt  11;
void    TM2_Routine(void)       interrupt  12;
void    INT4_Routine(void)      interrupt  16;
void    TM3_Routine(void)       interrupt  19;
void    TM4_Routine(void)       interrupt  20;
void    CMP_Routine(void)       interrupt  21;
void    I2C_Routine(void)       interrupt  24;
void    USB_Routine(void)       interrupt  25;
void    PWMA_Routine(void)      interrupt  26;
void    PWMB_Routine(void)      interrupt  27;

//void  P0Int_Routine(void)     interrupt  37;
//void  P1Int_Routine(void)     interrupt  38;
//void  P2Int_Routine(void)     interrupt  39;
//void  P3Int_Routine(void)     interrupt  40;
//void  P5Int_Routine(void)     interrupt  42;
```

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

STC MCU

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	-	ET4	ET3	-	-	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	-	PLVD	PADC	PS	PT1	PX1	PT0	PX0	x000,0000
IPH	高中断优先级控制寄存器	B7H	-	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H	x000,0000
IP2	中断优先级控制寄存器 2	B5H	-	PI2C	PCMP	PX4	PPWMB	PPWMA	PSPI	PS2	0000,0000
IP2H	高中断优先级控制寄存器 2	B6H	-	PI2CH	PCMPH	PX4H	PPWMBH	PPWMAH	PSPIH	PS2H	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
PCON	电源控制寄存器	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
ADC_CONTR	ADC 控制寄存器	BCH	ADC_POWER	ADC_START	ADC_FLAG	ADC_EPWMT	ADC_CHS[3:0]				000x,0000
SPSTAT	SPI 状态寄存器	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx,xxxx
CMPCR1	比较器控制寄存器 1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES	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^2C 主机控制寄存器	FE81H	EMSI	-	-	-	MSCMD[3:0]				0xxx,0000
I2CMSST	I^2C 主机状态寄存器	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO	00xx,xx00
I2CSLCR	I^2C 从机控制寄存器	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0
I2CSLST	I^2C 从机状态寄存器	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000
PWMA_IER	PWMA 中断使能寄存器	FEC4H	BIE	TIE	COMIE	CC4IE	CC3IE	CC2IE	CC1IE	UIE	0000,0000
PWMA_SR1	PWMA 状态寄存器 1	FEC5H	BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	UIF	0000,0000
PWMA_SR2	PWMA 状态寄存器 2	FEC6H	-	-	-	CC4OF	CC3OF	CC2OF	CC1OF	-	xxx0,000x
PWMB_IER	PWMB 中断使能寄存器	FEE4H	BIE	TIE	COMIE	CC8IE	CC7IE	CC6IE	CC5IE	UIE	0000,0000
PWMB_SR1	PWMB 状态寄存器 1	FEE5H	BIF	TIF	COMIF	CC8IF	CC7IF	CC6IF	CC5IF	UIF	0000,0000
PWMB_SR2	PWMB 状态寄存器 2	FEE6H	-	-	-	CC8OF	CC7OF	CC6OF	CC5OF	-	xxx0,000x
P0INTE	P0 口中断使能寄存器	FD00H	P07INTE	P06INTE	P05INTE	P04INTE	P03INTE	P02INTE	P01INTE	P00INTE	0000,0000
P1INTE	P1 口中断使能寄存器	FD01H	P17INTE	P16INTE	P15INTE	P14INTE	P13INTE	P12INTE	P11INTE	P10INTE	0000,0000
P2INTE	P2 口中断使能寄存器	FD02H	P27INTE	P26INTE	P25INTE	P24INTE	P23INTE	P22INTE	P21INTE	P20INTE	0000,0000
P3INTE	P3 口中断使能寄存器	FD03H	P37INTE	P36INTE	P35INTE	P34INTE	P33INTE	P32INTE	P31INTE	P30INTE	0000,0000
P5INTE	P5 口中断使能寄存器	FD05H	-	-	-	P54INTE	P53INTE	P52INTE	P51INTE	P50INTE	0000,0000
P0INTF	P0 口中断标志寄存器	FD10H	P07INTF	P06INTF	P05INTF	P04INTF	P03INTF	P02INTF	P01INTF	P00INTF	0000,0000
P1INTF	P1 口中断标志寄存器	FD11H	P17INTF	P16INTF	P15INTF	P14INTF	P13INTF	P12INTF	P11INTF	P10INTF	0000,0000
P2INTF	P2 口中断标志寄存器	FD12H	P27INTF	P26INTF	P25INTF	P24INTF	P23INTF	P22INTF	P21INTF	P20INTF	0000,0000
P3INTF	P3 口中断标志寄存器	FD13H	P37INTF	P36INTF	P35INTF	P34INTF	P33INTF	P32INTF	P31INTF	P30INTF	0000,0000
P5INTF	P5 口中断标志寄存器	FD15H	-	-	-	P54INTF	P53INTF	P52INTF	P51INTF	P50INTF	0000,0000

11.4.1 中断使能寄存器（中断允许位）

IE（中断使能寄存器）

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

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

0：CPU 屏蔽所有的中断申请

1：CPU 开放中断

ELVD：低压检测中断允许位。

0：禁止低压检测中断

1：允许低压检测中断

EADC：A/D 转换中断允许位。

0：禁止 A/D 转换中断

1：允许 A/D 转换中断

ES：串行口 1 中断允许位。

0：禁止串行口 1 中断

1：允许串行口 1 中断

ET1：定时/计数器 T1 的溢出中断允许位。

0：禁止 T1 中断

1：允许 T1 中断

EX1：外部中断 1 中断允许位。

0：禁止 INT1 中断

1：允许 INT1 中断

ET0：定时/计数器 T0 的溢出中断允许位。

0：禁止 T0 中断

1：允许 T0 中断

EX0：外部中断 0 中断允许位。

0：禁止 INT0 中断

1：允许 INT0 中断

IE2（中断使能寄存器 2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IE2	AFH	-	ET4	ET3	-	-	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

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。

STC MCU

0：禁止 SPI 中断

1：允许 SPI 中断

ES2：串行口 2 中断允许位。

0：禁止串行口 2 中断

1：允许串行口 2 中断

INTCLKO（外部中断与时钟输出控制寄存器）

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

EX4：外部中断 4 中断允许位。

0：禁止 INT4 中断

1：允许 INT4 中断

EX3：外部中断 3 中断允许位。

0：禁止 INT3 中断

1：允许 INT3 中断

EX2：外部中断 2 中断允许位。

0：禁止 INT2 中断

1：允许 INT2 中断

CMPCR1（比较器控制寄存器 1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR1	E6H	CMPEN	CMPIF	PIE	NIE	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[2:0]		
I2CSLCR	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST

EMSI：I^2C主机模式中断允许位。

0：禁止 I^2C 主机模式中断

1：允许 I^2C 主机模式中断

ESTAI：I^2C从机接收START事件中断允许位。

0：禁止 I^2C 从机接收 START 事件中断

1：允许 I^2C 从机接收 START 事件中断

ERXI：I^2C从机接收数据完成事件中断允许位。

0：禁止 I^2C 从机接收数据完成事件中断

1：允许 I^2C 从机接收数据完成事件中断

ETXI：I^2C从机发送数据完成事件中断允许位。

0：禁止 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

PWMA 中断使能寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_IER	FEC4H	BIE	TIE	COMIE	CC4IE	CC3IE	CC2IE	CC1IE	UIE

BIE：PWMA刹车中断允许位。

0：禁止 PWMA 刹车中断

1：允许 PWMA 刹车中断

TIE：PWMA触发中断允许位。

0：禁止 PWMA 触发中断

1：允许 PWMA 触发中断

COMIE：PWMA比较中断允许位。

0：禁止 PWMA 比较中断

1：允许 PWMA 比较中断

CC4IE：PWMA捕获比较通道4中断允许位。

0：禁止 PWMA 捕获比较通道 4 中断

1：允许 PWMA 捕获比较通道 4 中断

CC3IE：PWMA捕获比较通道3中断允许位。

0：禁止 PWMA 捕获比较通道 3 中断

1：允许 PWMA 捕获比较通道 3 中断

CC2IE：PWMA捕获比较通道2中断允许位。

0：禁止 PWMA 捕获比较通道 2 中断

1：允许 PWMA 捕获比较通道 2 中断

CC1IE：PWMA捕获比较通道1中断允许位。

0：禁止 PWMA 捕获比较通道 1 中断

1：允许 PWMA 捕获比较通道 1 中断

UIE：PWMA更新中断允许位。

0：禁止 PWMA 更新中断

1：允许 PWMA 更新中断

PWMB 中断使能寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMB_IER	FEE4H	BIE	TIE	COMIE	CC8IE	CC7IE	CC6IE	CC5IE	UIE

BIE：PWMB刹车中断允许位。

0：禁止 PWMB 刹车中断

1：允许 PWMB 刹车中断

TIE：PWMB触发中断允许位。

STC MCU

0：禁止 PWMB 触发中断
1：允许 PWMB 触发中断

COMIE：PWMB比较中断允许位。
0：禁止 PWMB 比较中断
1：允许 PWMB 比较中断

CC8IE：PWMB捕获比较通道8中断允许位。
0：禁止 PWMB 捕获比较通道 8 中断
1：允许 PWMB 捕获比较通道 8 中断

CC7IE：PWMB捕获比较通道7中断允许位。
0：禁止 PWMB 捕获比较通道 7 中断
1：允许 PWMB 捕获比较通道 7 中断

CC6IE：PWMB捕获比较通道6中断允许位。
0：禁止 PWMB 捕获比较通道 6 中断
1：允许 PWMB 捕获比较通道 6 中断

CC5IE：PWMB捕获比较通道5中断允许位。
0：禁止 PWMB 捕获比较通道 5 中断
1：允许 PWMB 捕获比较通道 5 中断

UIE：PWMB更新中断允许位。
0：禁止 PWMB 更新中断
1：允许 PWMB 更新中断

端口中断使能寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0INTE	FD00H	P07INTE	P06INTE	P05INTE	P04INTE	P03INTE	P02INTE	P01INTE	P00INTE
P1INTE	FD01H	P17INTE	P16INTE	P15INTE	P14INTE	P13INTE	P12INTE	P11INTE	P10INTE
P2INTE	FD02H	P27INTE	P26INTE	P25INTE	P24INTE	P23INTE	P22INTE	P21INTE	P20INTE
P3INTE	FD03H	P37INTE	P36INTE	P35INTE	P34INTE	P33INTE	P32INTE	P31INTE	P30INTE
P5INTE	FD05H	P57INTE	P56INTE	P55INTE	P54INTE	P53INTE	P52INTE	P51INTE	P50INTE

PnINTE.x：端口中断使能控制位（n=0~7，x=0~7）
0：关闭 Pn.x 口中断功能
1：使能 Pn.x 口中断功能

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，犹豫制造成本高，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，STC 已是实际的 8051 工业标准，就不要再考虑这兼容美国人的事了！

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

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。

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。

比较器控制寄存器 1

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

CMPIF：比较器中断请求标志。需要软件清零。

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：I^2C主机模式中断请求标志。需要软件清零。

ESTAI：I^2C从机接收START事件中断请求标志。需要软件清零。

ERXI：I^2C从机接收数据完成事件中断请求标志。需要软件清零。

ETXI：I^2C从机发送数据完成事件中断请求标志。需要软件清零。

ESTOI：I^2C从机接收STOP事件中断请求标志。需要软件清零。

PWMA 状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_SR1	FEC5H	BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	UIF
PWMA_SR2	FEC6H	-	-	-	CC4OF	CC3OF	CC2OF	CC1OF	-

BIF：PWMA刹车中断请求标志。需要软件清零。

TIF：PWMA触发中断请求标志。需要软件清零。

COMIF：PWMA比较中断请求标志。需要软件清零。

CC4IF：PWMA通道4发生捕获比较中断请求标志。需要软件清零。

CC3IF：PWMA通道3发生捕获比较中断请求标志。需要软件清零。

CC2IF：PWMA通道2发生捕获比较中断请求标志。需要软件清零。

CC1IF：PWMA通道1发生捕获比较中断请求标志。需要软件清零。

TIF：PWMA更新中断请求标志。需要软件清零。

CC4OF：PWMA通道4发生重复捕获中断请求标志。需要软件清零。

CC3OF：PWMA通道3发生重复捕获中断请求标志。需要软件清零。

CC2OF：PWMA通道2发生重复捕获中断请求标志。需要软件清零。

CC1OF：PWMA通道1发生重复捕获中断请求标志。需要软件清零。

PWMB 状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMB_SR1	FEE5H	BIF	TIF	COMIF	CC8IF	CC7IF	CC6IF	CC5IF	UIF
PWMB_SR2	FEE6H	-	-	-	CC8OF	CC7OF	CC6OF	CC5OF	-

BIF：PWMB刹车中断请求标志。需要软件清零。

TIF：PWMB触发中断请求标志。需要软件清零。

COMIF：PWMB比较中断请求标志。需要软件清零。

CC8IF：PWMB通道8发生捕获比较中断请求标志。需要软件清零。

CC7IF：PWMB通道7发生捕获比较中断请求标志。需要软件清零。

CC6IF：PWMB通道6发生捕获比较中断请求标志。需要软件清零。

CC5IF：PWMB通道5发生捕获比较中断请求标志。需要软件清零。

TIF：PWMB更新中断请求标志。需要软件清零。

CC8OF：PWMB通道8发生重复捕获中断请求标志。需要软件清零。

CC7OF：PWMB通道7发生重复捕获中断请求标志。需要软件清零。

CC6OF：PWMB通道6发生重复捕获中断请求标志。需要软件清零。

CC5OF：PWMB通道5发生重复捕获中断请求标志。需要软件清零。

端口中断标志寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0INTF	FD10H	P07INTF	P06INTF	P05INTF	P04INTF	P03INTF	P02INTF	P01INTF	P00INTF
P1INTF	FD11H	P17INTF	P16INTF	P15INTF	P14INTF	P13INTF	P12INTF	P11INTF	P10INTF
P2INTF	FD12H	P27INTF	P26INTF	P25INTF	P24INTF	P23INTF	P22INTF	P21INTF	P20INTF
P3INTF	FD13H	P37INTF	P36INTF	P35INTF	P34INTF	P33INTF	P32INTF	P31INTF	P30INTF
P5INTF	FD15H	P57INTF	P56INTF	P55INTF	P54INTF	P53INTF	P52INTF	P51INTF	P50INTF

PnINTF.x：端口中断请求标志位（n=0~7，x=0~7）

0：Pn.x 口没有中断请求

1：Pn.x 口有中断请求，若使能中断，则会进入中断服务程序。标志位需软件清 0。

11.4.3 中断优先级寄存器

除 INT2、INT3、定时器 2、定时器 3、定时器 4 以及全部的端口中断外，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	-	PLVD	PADC	PS	PT1	PX1	PT0	PX0
IPH	B7H	-	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H
IP2	B5H	-	PI2C	PCMP	PX4	PPWMB	PPWMA	PSPI	PS2
IP2H	B6H	-	PI2CH	PCMPH	PX4H	PPWMBH	PPWMAH	PSPIH	PS2H

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）

PS2H,PS2：串口2中断优先级控制位

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

PSPIH,PSPI：SPI中断优先级控制位

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

STC MCU

11：SPI 中断优先级为 3 级（最高级）

PPWMAH,PPWMA：高级PWMA中断优先级控制位

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

PPWMBH,PPWMB：高级PWMB中断优先级控制位

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

PX4H,PX4：外部中断4中断优先级控制位

00：INT4 中断优先级为 0 级（最低级）
01：INT4 中断优先级为 1 级（较低级）
10：INT4 中断优先级为 2 级（较高级）
11：INT4 中断优先级为 3 级（最高级）

PCMPH,PCMP：比较器中断优先级控制位

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

PI2CH,PI2C：I2C中断优先级控制位

00：I2C 中断优先级为 0 级（最低级）
01：I2C 中断优先级为 1 级（较低级）
10：I2C 中断优先级为 2 级（较高级）
11：I2C 中断优先级为 3 级（最高级）

STC MCU

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     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 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 = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

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         0003H
            LJMP        INT0ISR

            ORG         0100H
INT0ISR:
            JB          P3.2,RISING         ;判断上升沿和下降沿
            CPL         P1.0                ;测试端口
            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     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 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

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         0003H
            LJMP        INT0ISR

            ORG         0100H
INT0ISR:
            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

            SETB        IT0                     ;使能 INT0 下降沿中断
            SETB        EX0                     ;使能 INT0 中断
            SETB        EA
            JMP         $

            END
```

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     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 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
    EX1 = 1;                            //使能 INT1 中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

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         0013H
            LJMP        INT1ISR

            ORG         0100H
INT1ISR:
            JB          INT1,RISING             ;判断上升沿和下降沿
            CPL         P1.0                    ;测试端口
            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         IT1                     ;使能 INT1 上升沿和下降沿中断
            SETB        EX1                     ;使能 INT1 中断
            SETB        EA
            JMP         $

            END
```

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     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 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

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     0013H
        LJMP    INT1ISR

        ORG     0100H
```

```
INT1ISR:
        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

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

        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     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     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

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     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     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     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

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     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     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0x b1;
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 = 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

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     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     $
```

STC MCU

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     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 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;
    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

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

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

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         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
            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
            SETB        EA

            JMP         $

            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     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 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

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     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

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

            JMP     $

            END
```

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
```

```
T3L         DATA    0D5H
T3H         DATA    0D4H
T4T3M       DATA    0D1H
IE2         DATA    0AFH
ET3         EQU     20H
AUXINTIF    DATA    0EFH
T3IF        EQU     02H

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            ;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     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
```

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 = 1;

    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
```

STC MCU

```
        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     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 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;
    SBUF = 0x5a;                   //发送测试数据

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

T2L         DATA        0D7H
T2H         DATA        0D6H
AUXR        DATA        8EH

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         0023H
            LJMP        UART1ISR

            ORG         0100H
UART1ISR:
            JNB         TI,CHECKRI
            CLR         TI              ;清中断标志
            CPL         P1.0            ;测试端口
CHECKRI:
            JNB         RI,ISREXIT
            CLR         RI              ;清中断标志
```

```
            CPL         P1.1                    ;测试端口
ISREXIT:
            RETI

MAIN:
            MOV         SP, #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         SCON,#50H
            MOV         T2L,#0E8H               ;65536-11059200/115200/4=0FFE8H
            MOV         T2H,#0FFH
            MOV         AUXR,#15H               ;启动定时器
            SETB        ES                      ;使能串口中断
            SETB        EA
            MOV         SBUF,#5AH               ;发送测试数据

            JMP         $

            END
```

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     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    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;
    T2L = 0xe8;                             //65536-11059200/115200/4=0FFE8H
    T2H = 0xff;
    AUXR = 0x14;                            //启动定时器
    IE2 = ES2;                              //使能串口中断
    EA = 1;
    S2BUF = 0x5a;                           //发送测试数据

    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
```

```
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         0043H
            LJMP        UART2ISR

            ORG         0100H
UART2ISR:
            PUSH        ACC
            PUSH        PSW
            MOV         A,S2CON
            JNB         ACC.1,CHECKRI
            ANL         S2CON,#NOT 02H          ;清中断标志
            CPL         P1.2                    ;测试端口
CHECKRI:
            MOV         A,S2CON
            JNB         ACC.0,ISREXIT
            ANL         S2CON,#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         S2CON,#10H
            MOV         T2L,#0E8H               ;65536-11059200/115200/4=0FFE8H
            MOV         T2H,#0FFH
            MOV         AUXR,#14H               ;启动定时器
            MOV         IE2,#ES2                ;使能串口中断
            SETB        EA
            MOV         S2BUF,#5AH              ;发送测试数据
```

```
        JMP     $

        END
```

11.5.15 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     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 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;
```

STC MCU

```
    ADC_CONTR = 0xc0;                    //使能并启动 ADC 模块
    EADC = 1;                            //使能 ADC 中断
    EA = 1;

    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

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     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
```

11.5.16 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;
#define ENLVR           0x40            //RSTCFG.6
#define LVD2V2          0x00            //LVD@2.2V
#define LVD2V4          0x01            //LVD@2.4V
#define LVD2V7          0x02            //LVD@2.7V
#define LVD3V0          0x03            //LVD@3.0V
sbit    ELVD        =   IE^6;
#define LVDF            0x20            //PCON.5

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 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

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     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
```

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

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 CMP_Isr() interrupt 21
{
    CMPCR1 &= ~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;
    CMPCR1 = 0x80;                    //使能比较器模块
    CMPCR1 |= 0x30;                   //使能比较器边沿中断
    CMPCR1 &= ~0x08;                  //P3.6 为CMP+输入脚
    CMPCR1 |= 0x04;                   //P3.7 为CMP-输入脚
    CMPCR1 |= 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

CMPCR1      DATA        0E6H
CMPCR2      DATA        0E7H

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         00ABH
            LJMP        CMPISR

            ORG         0100H
CMPISR:
            ANL         CMPCR1,#NOT 40H     ;清中断标志
            CPL         P1.0                ;测试端口
            RETI

MAIN:
            MOV         SP, #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
```

STC MCU

```
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     CMPCR2,#00H
        MOV     CMPCR1,#80H         ;使能比较器模块
        ORL     CMPCR1,#30H         ;使能比较器边沿中断
        ANL     CMPCR1,#NOT 08H     ;P3.6 为CMP+输入脚
        ORL     CMPCR1,#04H         ;P3.7 为CMP-输入脚
        ORL     CMPCR1,#02H         ;使能比较器输出
        SETB    EA

        JMP     $

        END
```

11.5.18 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     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 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;
    SPDAT = 0x5a;                    //发送测试数据

    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

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         004BH
            LJMP        SPIISR

            ORG         0100H
SPIISR:
            MOV         SPSTAT,#0C0H        ;清中断标志
            CPL         P1.0                ;测试端口
            RETI

MAIN:
            MOV         SP, #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     SPCTL,#50H          ;使能 SPI 主机模式
        MOV     SPSTAT,#0C0H        ;清中断标志
        MOV     IE2,#ESPI           ;使能 SPI 中断
        SETB    EA
        MOV     SPDAT,#5AH          ;发送测试数据

        JMP     $

        END
```

11.5.19 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 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)
#d
efine   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    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;
    I2CCFG = 0xc0;                           //使能 I2C 主机模式
    I2CMSCR = 0x80;                          //使能 I2C 中断;
    P_SW2 = 0x00;
    EA = 1;

    P_SW2 = 0x80;
    I2CMSCR = 0x81;                          //发送起始命令
    P_SW2 = 0x00;

    while (1);
}
```

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

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         00C3H
            LJMP        I2CISR

            ORG         0100H
I2CISR:
            PUSH        ACC
            PUSH        DPL
            PUSH        DPH
            PUSH        P_SW2
            MOV         P_SW2,#80H
            MOV         DPTR,#I2CMSST
            MOVX        A,@DPTR
            ANL         A,#NOT 40H          ;清中断标志
            MOVX        @DPTR,A
            CPL         P1.0                ;测试端口
            POP         P_SW2
            POP         DPH
            POP         DPL
            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         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
```

STC MCU

12 I/O 口中断

~~STC12H 系列支持所有的 I/O 中断，且支持 4 种中断模式：下降沿中断、上升沿中断、低电平中断、高电平中断。每组 I/O 口都有独立的中断入口地址，且每个 I/O 可独立设置中断模式。~~

经测试发现 STC12H 系列的 I/O 口中断有问题，请暂时不要使用

12.1 I/O 口中断相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
P0INTE	P0 口中断使能寄存器	FD00H	P07INTE	P06INTE	P05INTE	P04INTE	P03INTE	P02INTE	P01INTE	P00INTE	0000,0000
P1INTE	P1 口中断使能寄存器	FD01H	P17INTE	P16INTE	P15INTE	P14INTE	P13INTE	P12INTE	P11INTE	P10INTE	0000,0000
P2INTE	P2 口中断使能寄存器	FD02H	P27INTE	P26INTE	P25INTE	P24INTE	P23INTE	P22INTE	P21INTE	P20INTE	0000,0000
P3INTE	P3 口中断使能寄存器	FD03H	P37INTE	P36INTE	P35INTE	P34INTE	P33INTE	P32INTE	P31INTE	P30INTE	0000,0000
P5INTE	P5 口中断使能寄存器	FD05H	P57INTE	P56INTE	P55INTE	P54INTE	P53INTE	P52INTE	P51INTE	P50INTE	0000,0000
P0INTF	P0 口中断标志寄存器	FD10H	P07INTF	P06INTF	P05INTF	P04INTF	P03INTF	P02INTF	P01INTF	P00INTF	0000,0000
P1INTF	P1 口中断标志寄存器	FD11H	P17INTF	P16INTF	P15INTF	P14INTF	P13INTF	P12INTF	P11INTF	P10INTF	0000,0000
P2INTF	P2 口中断标志寄存器	FD12H	P27INTF	P26INTF	P25INTF	P24INTF	P23INTF	P22INTF	P21INTF	P20INTF	0000,0000
P3INTF	P3 口中断标志寄存器	FD13H	P37INTF	P36INTF	P35INTF	P34INTF	P33INTF	P32INTF	P31INTF	P30INTF	0000,0000
P5INTF	P5 口中断标志寄存器	FD15H	P57INTF	P56INTF	P55INTF	P54INTF	P53INTF	P52INTF	P51INTF	P50INTF	0000,0000
P0IM0	P0 口中断模式寄存器 0	FD20H	P07IM0	P06IM0	P05IM0	P04IM0	P03IM0	P02IM0	P01IM0	P00IM0	0000,0000
P1IM0	P1 口中断模式寄存器 0	FD21H	P17IM0	P16IM0	P15IM0	P14IM0	P13IM0	P12IM0	P11IM0	P10IM0	0000,0000
P2IM0	P2 口中断模式寄存器 0	FD22H	P27IM0	P26IM0	P25IM0	P24IM0	P23IM0	P22IM0	P21IM0	P20IM0	0000,0000
P3IM0	P3 口中断模式寄存器 0	FD23H	P37IM0	P36IM0	P35IM0	P34IM0	P33IM0	P32IM0	P31IM0	P30IM0	0000,0000
P5IM0	P5 口中断模式寄存器 0	FD25H	P57IM0	P56IM0	P55IM0	P54IM0	P53IM0	P52IM0	P51IM0	P50IM0	0000,0000
P0IM1	P0 口中断模式寄存器 1	FD30H	P07IM1	P06IM1	P05IM1	P04IM1	P03IM1	P02IM1	P01IM1	P00IM1	0000,0000
P1IM1	P1 口中断模式寄存器 1	FD31H	P17IM1	P16IM1	P15IM1	P14IM1	P13IM1	P12IM1	P11IM1	P10IM1	0000,0000
P2IM1	P2 口中断模式寄存器 1	FD32H	P27IM1	P26IM1	P25IM1	P24IM1	P23IM1	P22IM1	P21IM1	P20IM1	0000,0000
P3IM1	P3 口中断模式寄存器 1	FD33H	P37IM1	P36IM1	P35IM1	P34IM1	P33IM1	P32IM1	P31IM1	P30IM1	0000,0000
P5IM1	P5 口中断模式寄存器 1	FD35H	P57IM1	P56IM1	P55IM1	P54IM1	P53IM1	P52IM1	P51IM1	P50IM1	0000,0000

端口中断使能寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0INTE	FD00H	P07INTE	P06INTE	P05INTE	P04INTE	P03INTE	P02INTE	P01INTE	P00INTE
P1INTE	FD01H	P17INTE	P16INTE	P15INTE	P14INTE	P13INTE	P12INTE	P11INTE	P10INTE
P2INTE	FD02H	P27INTE	P26INTE	P25INTE	P24INTE	P23INTE	P22INTE	P21INTE	P20INTE
P3INTE	FD03H	P37INTE	P36INTE	P35INTE	P34INTE	P33INTE	P32INTE	P31INTE	P30INTE
P5INTE	FD05H	P57INTE	P56INTE	P55INTE	P54INTE	P53INTE	P52INTE	P51INTE	P50INTE

PnINTE.x：端口中断使能控制位（n=0~7，x=0~7）

0：关闭 Pn.x 口中断功能

1：使能 Pn.x 口中断功能

端口中断标志寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0INTF	FD10H	P07INTF	P06INTF	P05INTF	P04INTF	P03INTF	P02INTF	P01INTF	P00INTF
P1INTF	FD11H	P17INTF	P16INTF	P15INTF	P14INTF	P13INTF	P12INTF	P11INTF	P10INTF
P2INTF	FD12H	P27INTF	P26INTF	P25INTF	P24INTF	P23INTF	P22INTF	P21INTF	P20INTF
P3INTF	FD13H	P37INTF	P36INTF	P35INTF	P34INTF	P33INTF	P32INTF	P31INTF	P30INTF
P5INTF	FD15H	P57INTF	P56INTF	P55INTF	P54INTF	P53INTF	P52INTF	P51INTF	P50INTF

PnINTF.x：端口中断请求标志位（n=0~7，x=0~7）

0：Pn.x 口没有中断请求

1：Pn.x 口有中断请求，若使能中断，则会进入中断服务程序。标志位需软件清 0。

端口中断模式配置寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0IM0	FD20H	P07IM0	P06IM0	P05IM0	P04IM0	P03IM0	P02IM0	P01IM0	P00IM0
P0IM1	FD30H	P07IM1	P06IM1	P05IM1	P04IM1	P03IM1	P02IM1	P01IM1	P00IM1
P1IM0	FD21H	P17IM0	P16IM0	P15IM0	P14IM0	P13IM0	P12IM0	P11IM0	P10IM0
P1IM1	FD31H	P17IM1	P16IM1	P15IM1	P14IM1	P13IM1	P12IM1	P11IM1	P10IM1
P2IM0	FD22H	P27IM0	P26IM0	P25IM0	P24IM0	P23IM0	P22IM0	P21IM0	P20IM0
P2IM1	FD32H	P27IM1	P26IM1	P25IM1	P24IM1	P23IM1	P22IM1	P21IM1	P20IM1
P3IM0	FD23H	P37IM0	P36IM0	P35IM0	P34IM0	P33IM0	P32IM0	P31IM0	P30IM0
P3IM1	FD33H	P37IM1	P36IM1	P35IM1	P34IM1	P33IM1	P32IM1	P31IM1	P30IM1
P5IM0	FD25H	P57IM0	P56IM0	P55IM0	P54IM0	P53IM0	P52IM0	P51IM0	P50IM0
P5IM1	FD35H	P57IM1	P56IM1	P55IM1	P54IM1	P53IM1	P52IM1	P51IM1	P50IM1

配置端口的模式

PnIM1.x	PnIM0.x	Pn.x 口中断模式
0	0	下降沿中断
0	1	上升沿中断
1	0	低电平中断
1	1	高电平中断

12.2 范例程序

12.2.1 P0 口下降沿中断

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr     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;

sfr     P_SW2   =   0xba;

#define P0INTE  (*(unsigned char volatile xdata *)0xfd00)
#define P0INTF  (*(unsigned char volatile xdata *)0xfd10)
#define P0IM0   (*(unsigned char volatile xdata *)0xfd20)
#define P0IM1   (*(unsigned char volatile xdata *)0xfd30)

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;
    P0IM0 = 0x00;                   //下降沿中断
    P0IM1 = 0x00;
    P0INTE = 0xff;                  //使能 P0 口中断
    P_SW2 &= ~0x80;
```

```
    EA = 1;

    while (1);
}

//由于中断向量大于 31，在 KEIL 中无法直接编译
//必须借用第 13 号中断入口地址
void common_isr() interrupt 13
{
    unsigned char psw2_st;
    unsigned char intf;

    psw2_st = P_SW2;
    P_SW2 |= 0x80;
    intf = P0INTF;
    if (intf)
    {
        P0INTF = 0x00;
        if (intf & 0x01)
        {
                                                    //P0.0 口中断
        }
        if (intf & 0x02)
        {
                                                    //P0.1 口中断
        }
        if (intf & 0x04)
        {
                                                    //P0.2 口中断
        }
        if (intf & 0x08)
        {
                                                    //P0.3 口中断
        }
        if (intf & 0x10)
        {
                                                    //P0.4 口中断
        }
        if (intf & 0x20)
        {
                                                    //P0.5 口中断
        }
        if (intf & 0x40)
        {
                                                    //P0.6 口中断
        }
        if (intf & 0x80)
        {
                                                    //P0.7 口中断
        }
    }
    P_SW2 = psw2_st;
}

// ISR.ASM
//将下面的代码保存为 ISR.ASM，然后将文件加入到项目中即可

            CSEG        AT 012BH                    ;P0 口中断入口地址
            JMP         P0INT_ISR
```

```
P0INT_ISR:
            JMP     006BH                   ;借用 13 号中断的入口地址
            END
```

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

P_SW2       DATA    0BAH

P0INTE      XDATA   0FD00H
P0INTF      XDATA   0FD10H
P0IM0       XDATA   0FD20H
P0IM1       XDATA   0FD30H

            ORG     0000H
            LJMP    MAIN

            ORG     012BH                   ;P0 口中断入口地址
P0INT_ISR:
            PUSH    ACC
            PUSH    B
            PUSH    DPL
            PUSH    DPH
            PUSH    P_SW2

            MOV     DPTR,#P0INTF
            MOVX    A,@DPTR
            MOV     B,A
            CLR     A
            MOVX    @DPTR,A
            MOV     A,B
CHECKP00:
            JNB     ACC.0,CHECKP01
            NOP                             ;P0.0 口中断
CHECKP01:
            JNB     ACC.1,CHECKP02
            NOP                             ;P0.1 口中断
CHECKP02:
            JNB     ACC.2,CHECKP03
```

```
            NOP                                 ;P0.2 口中断
CHECKP03
            JNB     ACC.3,CHECKP04
            NOP                                 ;P0.3 口中断
CHECKP04:
            JNB     ACC.4,CHECKP05
            NOP                                 ;P0.4 口中断
CHECKP05:
            JNB     ACC.5,CHECKP06
            NOP                                 ;P0.5 口中断
CHECKP06:
            JNB     ACC.6,CHECKP07
            NOP                                 ;P0.6 口中断
CHECKP07:
            JNB     ACC.7,P0ISREXIT
            NOP                                 ;P0.7 口中断

P0ISREXIT:
            POP     P_SW2
            POP     DPH
            POP     DPL
            POP     B
            POP     ACC
            RETI

            ORG     0200H
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

            ORL     P_SW2,#80H
            CLR     A
            MOV     DPTR,# P0IM0                ;下降沿中断
            MOVX    @DPTR,A
            MOV     DPTR,# P0IM1
            MOVX    @DPTR,A
            MOV     DPTR,# P0INTE
            MOV     A,#0FFH
            MOVX    @DPTR,A                     ;使能 P0 口中断
            ANL     P_SW2,#7FH

            SETB    EA

            JMP     $

            END
```

12.2.2 P1 口上升沿中断

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;

sfr     P_SW2   =   0xba;

#define P1INTE  (*(unsigned char volatile xdata *)0xfd01)
#define P1INTF  (*(unsigned char volatile xdata *)0xfd11)
#define P1IM0   (*(unsigned char volatile xdata *)0xfd21)
#define P1IM1   (*(unsigned char volatile xdata *)0xfd31)

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;
    P1IM0 = 0xff;                   //上升沿中断
    P1IM1 = 0x00;
    P1INTE = 0xff;                  //使能 P1 口中断
    P_SW2 &= ~0x80;

    EA = 1;

    while (1);
}
```

```
//由于中断向量大于31，在KEIL 中无法直接编译
//必须借用第13 号中断入口地址
void common_isr() interrupt 13
{
        unsigned char psw2_st;
        unsigned char intf;

        psw2_st = P_SW2;
        P_SW2 |= 0x80;
        intf = P1INTF;
        if (intf)
        {
                P1INTF = 0x00;
                if (intf & 0x01)
                {
                                                        //P1.0 口中断
                }
                if (intf & 0x02)
                {
                                                        //P1.1 口中断
                }
                if (intf & 0x04)
                {
                                                        //P1.2 口中断
                }
                if (intf & 0x08)
                {
                                                        //P1.3 口中断
                }
                if (intf & 0x10)
                {
                                                        //P1.4 口中断
                }
                if (intf & 0x20)
                {
                                                        //P1.5 口中断
                }
                if (intf & 0x40)
                {
                                                        //P1.6 口中断
                }
                if (intf & 0x80)
                {
                                                        //P1.7 口中断
                }
        }
        P_SW2 = psw2_st;
}

// ISR.ASM
//将下面的代码保存为ISR.ASM，然后将文件加入到项目中即可

                CSEG            AT 0133H                ;P1 口中断入口地址
                JMP             P1INT_ISR
P1INT_ISR:
                JMP             006BH                   ;借用13 号中断的入口地址
                END
```

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

P_SW2       DATA        0BAH

P1INTE      XDATA       0FD01H
P1INTF      XDATA       0FD11H
P1IM0       XDATA       0FD21H
P1IM1       XDATA       0FD31H

            ORG         0000H
            LJMP        MAIN

            ORG         0133H                   ;P1 口中断入口地址
P1INT_ISR:
            PUSH        ACC
            PUSH        B
            PUSH        DPL
            PUSH        DPH
            PUSH        P_SW2

            MOV         DPTR,#P1INTF
            MOVX        A,@DPTR
            MOV         B,A
            CLR         A
            MOVX        @DPTR,A
            MOV         A,B
CHECKP10:
            JNB         ACC.0,CHECKP11
            NOP                                 ;P1.0 口中断
CHECKP11:
            JNB         ACC.1,CHECKP12
            NOP                                 ;P1.1 口中断
CHECKP12:
            JNB         ACC.2,CHECKP13
            NOP                                 ;P1.2 口中断
CHECKP13
            JNB         ACC.3,CHECKP14
            NOP                                 ;P1.3 口中断
```

```
CHECKP14:
        JNB     ACC.4,CHECKP15
        NOP                             ;P1.4 口中断
CHECKP15:
        JNB     ACC.5,CHECKP16
        NOP                             ;P1.5 口中断
CHECKP16:
        JNB     ACC.6,CHECKP17
        NOP                             ;P1.6 口中断
CHECKP17:
        JNB     ACC.7,P1ISREXIT
        NOP                             ;P1.7 口中断

P1ISREXIT:
        POP     P_SW2
        POP     DPH
        POP     DPL
        POP     B
        POP     ACC
        RETI

        ORG     0200H
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

        ORL     P_SW2,#80H
        CLR     A
        MOV     DPTR,# P1IM0            ;下降沿中断
        MOVX    @DPTR,A
        MOV     DPTR,# P1IM1
        MOVX    @DPTR,A
        MOV     DPTR,# P1INTE
        MOV     A,#0FFH
        MOVX    @DPTR,A                 ;使能 P1 口中断
        ANL     P_SW2,#7FH

        SETB    EA

        JMP     $

        END
```

12.2.3 P2 口低电平中断

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    =   0x b2;
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;

sfr     P_SW2   =   0xba;

#define P2INTE  (*(unsigned char volatile xdata *)0xfd02)
#define P2INTF  (*(unsigned char volatile xdata *)0xfd12)
#define P2IM0   (*(unsigned char volatile xdata *)0xfd22)
#define P2IM1   (*(unsigned char volatile xdata *)0xfd32)

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;
    P2IM0 = 0x00;                   //低电平中断
    P2IM1 = 0xff;
    P2INTE = 0xff;                  //使能 P2 口中断
    P_SW2 &= ~0x80;

    EA = 1;

    while (1);
}

//由于中断向量大于 31，在 KEIL 中无法直接编译
//必须借用第 13 号中断入口地址
void common_isr() interrupt 13
{
    unsigned char psw2_st;
```

```
    unsigned char intf;

    psw2_st = P_SW2;
    P_SW2 |= 0x80;
    intf = P2INTF;
    if (intf)
    {
        P2INTF = 0x00;
        if (intf & 0x01)
        {
                                                    //P2.0 口中断
        }
        if (intf & 0x02)
        {
                                                    //P2.1 口中断
        }
        if (intf & 0x04)
        {
                                                    //P2.2 口中断
        }
        if (intf & 0x08)
        {
                                                    //P0.3 口中断
        }
        if (intf & 0x10)
        {
                                                    //P2.4 口中断
        }
        if (intf & 0x20)
        {
                                                    //P2.5 口中断
        }
        if (intf & 0x40)
        {
                                                    //P2.6 口中断
        }
        if (intf & 0x80)
        {
                                                    //P2.7 口中断
        }
    }
    P_SW2 = psw2_st;
}

// ISR.ASM
//将下面的代码保存为ISR.ASM，然后将文件加入到项目中即可

            CSEG    AT 013BH            ;P2 口中断入口地址
            JMP     P2INT_ISR
P2INT_ISR:
            JMP     006BH               ;借用 13 号中断的入口地址
            END
```

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

P_SW2       DATA        0BAH

P2INTE      XDATA       0FD02H
P2INTF      XDATA       0FD12H
P2IM0       XDATA       0FD22H
P2IM1       XDATA       0FD32H

            ORG         0000H
            LJMP        MAIN

            ORG         013BH                   ;P2 口中断入口地址
P2INT_ISR:
            PUSH        ACC
            PUSH        B
            PUSH        DPL
            PUSH        DPH
            PUSH        P_SW2

            MOV         DPTR,#P2INTF
            MOVX        A,@DPTR
            MOV         B,A
            CLR         A
            MOVX        @DPTR,A
            MOV         A,B
CHECKP20:
            JNB         ACC.0,CHECKP21
            NOP                                 ;P2.0 口中断
CHECKP21:
            JNB         ACC.1,CHECKP22
            NOP                                 ;P2.1 口中断
CHECKP22:
            JNB         ACC.2,CHECKP23
            NOP                                 ;P2.2 口中断
CHECKP23
            JNB         ACC.3,CHECKP24
            NOP                                 ;P2.3 口中断
CHECKP24:
            JNB         ACC.4,CHECKP25
            NOP                                 ;P2.4 口中断
CHECKP25:
            JNB         ACC.5,CHECKP26
            NOP                                 ;P2.5 口中断
```

```
CHECKP26:
        JNB     ACC.6,CHECKP27
        NOP                             ;P2.6 口中断
CHECKP27:
        JNB     ACC.7,P2ISREXIT
        NOP                             ;P2.7 口中断

P2ISREXIT:
        POP     P_SW2
        POP     DPH
        POP     DPL
        POP     B
        POP     ACC
        RETI

        ORG     0200H
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

        ORL     P_SW2,#80H
        CLR     A
        MOV     DPTR,# P2IM0            ;低电平中断
        MOVX    @DPTR,A
        MOV     DPTR,# P2IM1
        MOVX    @DPTR,A
        MOV     DPTR,# P2INTE
        MOV     A,#0FFH
        MOVX    @DPTR,A                 ;使能 P2 口中断
        ANL     P_SW2,#7FH

        SETB    EA

        JMP     $

        END
```

12.2.4 P3 口高电平中断

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;

sfr     P_SW2   =   0xba;

#define P3INTE  (*(unsigned char volatile xdata *)0xfd03)
#define P3INTF  (*(unsigned char volatile xdata *)0xfd13)
#define P3IM0   (*(unsigned char volatile xdata *)0xfd23)
#define P3IM1   (*(unsigned char volatile xdata *)0xfd33)

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;
    P3IM0 = 0xff;                   //高电平中断
    P3IM1 = 0xff;
    P3INTE = 0xff;                  //使能 P3 口中断
    P_SW2 &= ~0x80;

    EA = 1;

    while (1);
}

//由于中断向量大于 31，在 KEIL 中无法直接编译
//必须借用第 13 号中断入口地址
void common_isr() interrupt 13
{
    unsigned char psw2_st;
    unsigned char intf;

    psw2_st = P_SW2;
    P_SW2 |= 0x80;
    intf = P3INTF;
    if (intf)
```

```
    {
        P3INTF = 0x00;
        if (intf & 0x01)
        {
                                                    //P3.0 口中断
        }
        if (intf & 0x02)
        {
                                                    //P3.1 口中断
        }
        if (intf & 0x04)
        {
                                                    //P3.2 口中断
        }
        if (intf & 0x08)
        {
                                                    //P3.3 口中断
        }
        if (intf & 0x10)
        {
                                                    //P3.4 口中断
        }
        if (intf & 0x20)
        {
                                                    //P3.5 口中断
        }
        if (intf & 0x40)
        {
                                                    //P3.6 口中断
        }
        if (intf & 0x80)
        {
                                                    //P3.7 口中断
        }
    }
    P_SW2 = psw2_st;
}

// ISR.ASM
//将下面的代码保存为ISR.ASM，然后将文件加入到项目中即可

            CSEG        AT 0143H            ;P3 口中断入口地址
            JMP         P3INT_ISR
P3INT_ISR:
            JMP         006BH               ;借用 13 号中断的入口地址
            END
```

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

P_SW2       DATA        0BAH

P3INTE      XDATA       0FD03H
P3INTF      XDATA       0FD13H
P3IM0       XDATA       0FD23H
P3IM1       XDATA       0FD33H

            ORG         0000H
            LJMP        MAIN

            ORG         0143H                   ;P3 口中断入口地址
P3INT_ISR:
            PUSH        ACC
            PUSH        B
            PUSH        DPL
            PUSH        DPH
            PUSH        P_SW2

            MOV         DPTR,#P3INTF
            MOVX        A,@DPTR
            MOV         B,A
            CLR         A
            MOVX        @DPTR,A
            MOV         A,B
CHECKP30:
            JNB         ACC.0,CHECKP31
            NOP                                 ;P3.0 口中断
CHECKP31:
            JNB         ACC.1,CHECKP32
            NOP                                 ;P3.1 口中断
CHECKP32:
            JNB         ACC.2,CHECKP33
            NOP                                 ;P3.2 口中断
CHECKP33
            JNB         ACC.3,CHECKP34
            NOP                                 ;P3.3 口中断
CHECKP34:
            JNB         ACC.4,CHECKP35
            NOP                                 ;P3.4 口中断
CHECKP35:
            JNB         ACC.5,CHECKP36
            NOP                                 ;P3.5 口中断
CHECKP36:
            JNB         ACC.6,CHECKP37
            NOP                                 ;P3.6 口中断
CHECKP37:
            JNB         ACC.7,P3ISREXIT
            NOP                                 ;P3.7 口中断
```

STC MCU

```
P3ISREXIT:
        POP     P_SW2
        POP     DPH
        POP     DPL
        POP     B
        POP     ACC
        RETI

        ORG     0200H
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

        ORL     P_SW2,#80H
        CLR     A
        MOV     DPTR,# P3IM0            ;高电平中断
        MOVX    @DPTR,A
        MOV     DPTR,# P3IM1
        MOVX    @DPTR,A
        MOV     DPTR,# P3INTE
        MOV     A,#0FFH
        MOVX    @DPTR,A                 ;使能 P3 口中断
        ANL     P_SW2,#7FH

        SETB    EA

        JMP     $

        END
```

STC MCU

13 定时器/计数器

STC12H 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。

13.1 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	WKTEN								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

STC MCU

13.2 定时器 0/1

13.2.1 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。

13.2.2 定时器 0/1 模式寄存器（TMOD）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TMOD	89H	T1_GATE	T1_C/T	T1_M1	T1_M0	T0_GATE	T0_C/T	T0_M1	T0_M0

T1_GATE：控制定时器1，置1时只有在INT1脚为高及TR1控制位置1时才可打开定时器/计数器1。

T0_GATE：控制定时器0，置1时只有在INT0脚为高及TR0控制位置1时才可打开定时器/计数器0。

T1_C/T：控制定时器1用作定时器或计数器，清0则用作定时器（对内部系统时钟进行计数），置1用作计数器（对引脚T1/P3.5外部脉冲进行计数）。

T0_C/T：控制定时器0用作定时器或计数器，清0则用作定时器（对内部系统时钟进行计数），置1用作计数器（对引脚T0/P3.4外部脉冲进行计数）。

T1_M1/T1_M0：定时器定时器/计数器1模式选择

T1_M1	T1_M0	定时器/计数器1工作模式
0	0	16位自动重载模式 当[TH1,TL1]中的16位计数值溢出时，系统会自动将内部16位

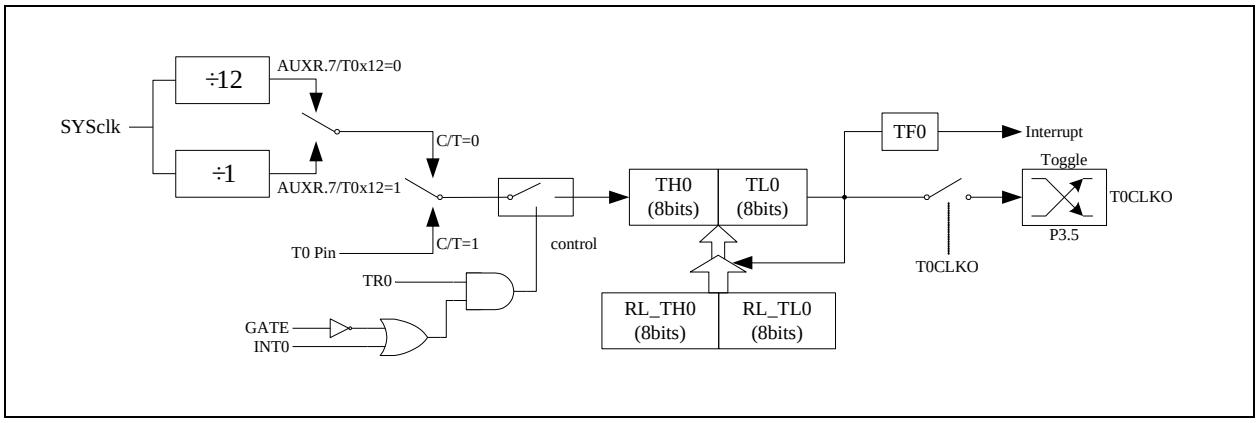
		重载寄存器中的重载值装入[TH1,TL1]中。
0	1	16位不自动重载模式 当[TH1,TL1]中的16位计数值溢出时，定时器1将从0开始计数
1	0	8位自动重载模式 当TL1中的8位计数值溢出时，系统会自动将TH1中的重载值装入TL1中。
1	1	T1停止工作

T0_M1/T0_M0：定时器定时器/计数器0模式选择

T0_M1	T0_M0	定时器/计数器0工作模式
0	0	16位自动重载模式 当[TH0,TL0]中的16位计数值溢出时，系统会自动将内部16位重载寄存器中的重载值装入[TH0,TL0]中。
0	1	16位不自动重载模式 当[TH0,TL0]中的16位计数值溢出时，定时器0将从0开始计数
1	0	8位自动重载模式 当TL0中的8位计数值溢出时，系统会自动将TH0中的重载值装入TL0中。
1	1	不可屏蔽中断的16位自动重载模式 与模式0相同，不可屏蔽中断，中断优先级最高，高于其他所有中断的优先级，并且不可关闭，可用作操作系统的系统节拍定时器，或者系统监控定时器。

13.2.3 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 = (SYSclk)/(65536-[RL_TH0, RL_TL0])/2

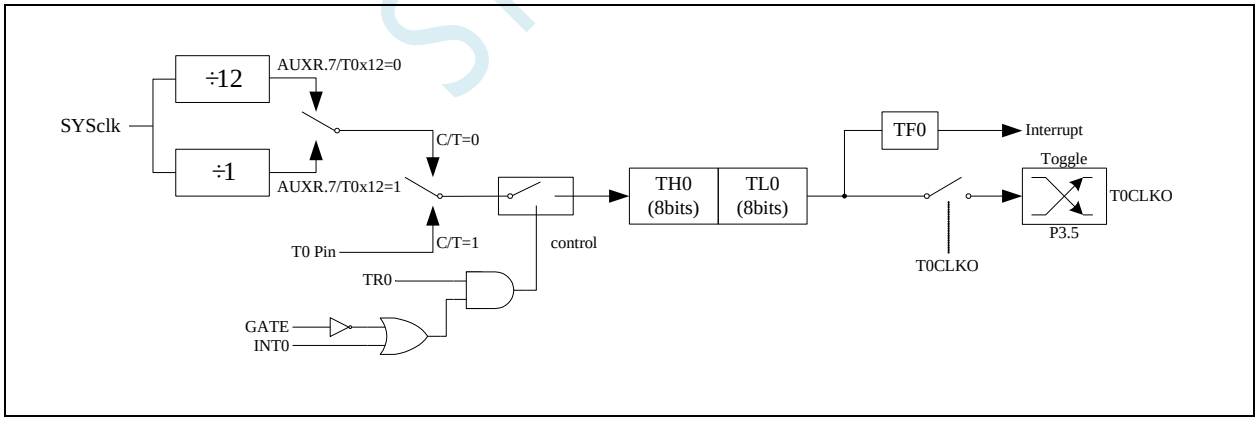
T0 工作在 12T 模式（AUXR.7/T0x12=0）时的输出时钟频率 = (SYSclk)/12/(65536-[RL_TH0, RL_TL0])/2

如果 C/T=1，定时器/计数器 T0 是对外部脉冲输入(P3.4/T0)计数，则：

输出时钟频率 = (T0_Pin_CLK) / (65536-[RL_TH0, RL_TL0])/2

13.2.4 定时器 0 模式 1（16 位不可重装载模式）

此模式下定时器/计数器 0 工作在 16 位不可重装载模式，如下图所示



定时器/计数器 0 的模式 1：16 位不可重装载模式

此模式下，定时器/计数器 0 配置为 16 位不可重装载模式，由 TL0 的 8 位和 TH0 的 8 位所构成。TL0 的 8 位溢出向 TH0 进位，TH0 计数溢出置位 TCON 中的溢出标志位 TF0。

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

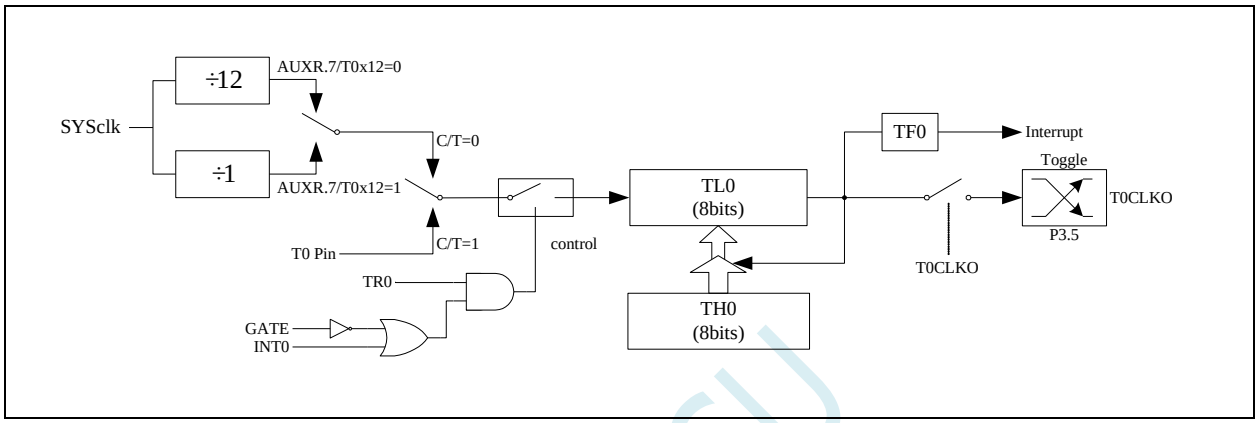
当 C/T=0 时，多路开关连接到系统时钟的分频输出，T0 对内部系统时钟计数，T0 工作在定时方式。当

C/T=1 时，多路开关连接到外部脉冲输入 P3.4/T0，即 T0 工作在计数方式。

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

13.2.5 定时器 0 模式 2（8 位自动重装载模式）

此模式下定时器/计数器 0 作为可自动重装载的 8 位计数器，如下图所示：



定时器/计数器 0 的模式 2：8 位自动重装载模式

TL0 的溢出不仅置位 TF0，而且将 TH0 的内容重新装入 TL0，TH0 内容由软件预置，重装时 TH0 内容不变。

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

如果 C/T=0，定时器/计数器 T0 对内部系统时钟计数，则：

T0 工作在 1T 模式（AUXR.7/T0x12=1）时的输出时钟频率 = (SYSclk)/(256-TH0)/2

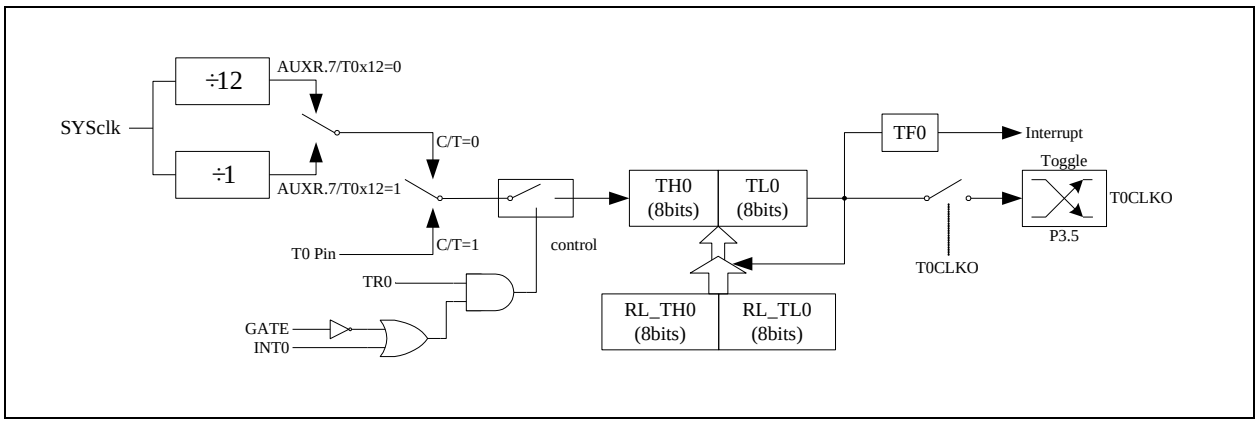
T0 工作在 12T 模式（AUXR.7/T0x12=0）时的输出时钟频率 = (SYSclk)/12/(256-TH0)/2

如果 C/T=1，定时器/计数器 T0 是对外部脉冲输入(P3.4/T0)计数，则：

输出时钟频率 = (T0_Pin_CLK) / (256-TH0)/2

13.2.6 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。

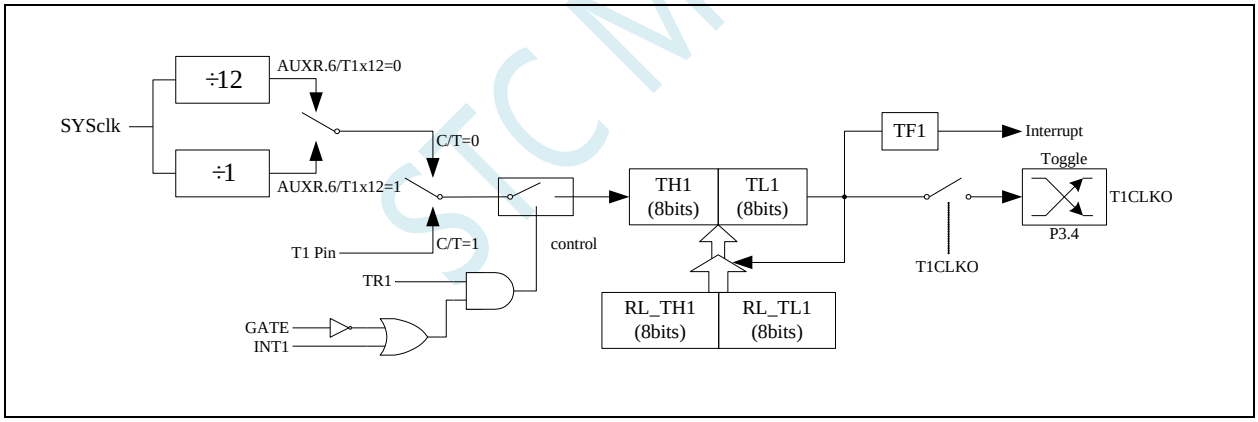


定时器/计数器 0 的模式 3：不可屏蔽中断的 16 位自动重装载模式

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。

13.2.7 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 P3.5/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

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

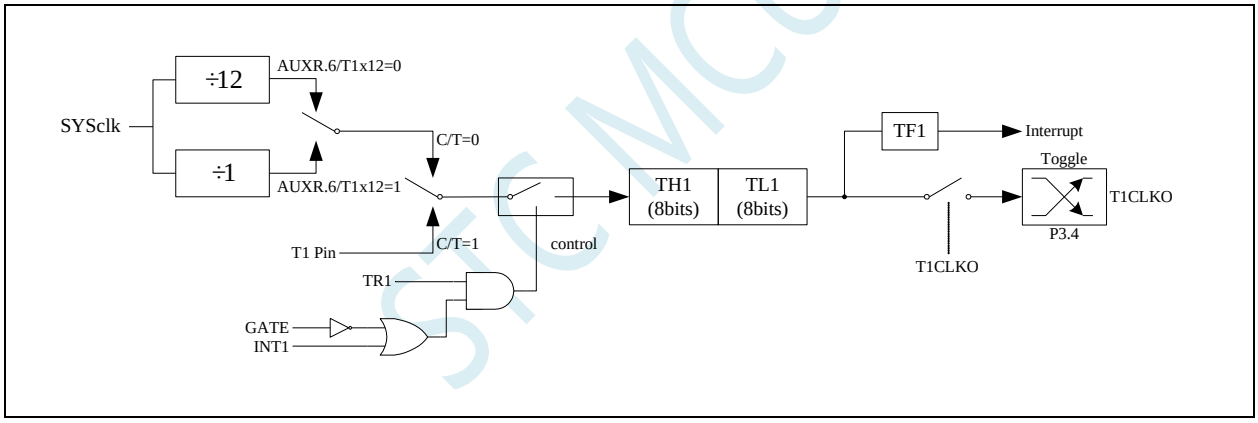
T1 工作在 12T 模式（AUXR.6/T1x12=0）时的输出时钟频率 = (SYSclk)/12/(65536-[RL_TH1, RL_TL1])/2

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

13.2.8 定时器 1 模式 1（16 位不可重装载模式）

此模式下定时器/计数器 1 工作在 16 位不可重装载模式，如下图所示



定时器/计数器 1 的模式 1：16 位不可重装载模式

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

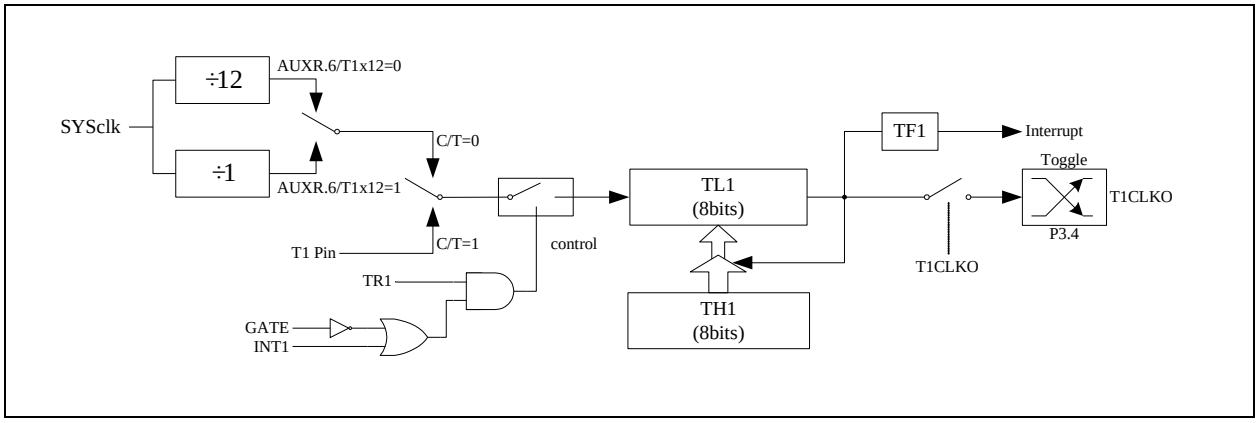
当 GATE=0(TMOD.7)时，如 TR1=1，则定时器计数。GATE=1 时，允许由外部输入 INT1 控制定时器 1，这样可实现脉宽测量。TR1 为 TCON 寄存器内的控制位，TCON 寄存器各位的具体功能描述见上节 TCON 寄存器的介绍。

当 C/T=0 时，多路开关连接到系统时钟的分频输出，T1 对内部系统时钟计数，T1 工作在定时方式。当 C/T=1 时，多路开关连接到外部脉冲输入 P3.5/T1，即 T1 工作在计数方式。

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

13.2.9 定时器 1 模式 2（8 位自动重装载模式）

此模式下定时器/计数器 1 作为可自动重装载的 8 位计数器，如下图所示：



定时器/计数器 1 的模式 2：8 位自动重装载模式

TL1 的溢出不仅置位 TF1，而且将 TH1 的内容重新装入 TL1，TH1 内容由软件预置，重装时 TH1 内容不变。

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

如果 C/T=0，定时器/计数器 T1 对内部系统时钟计数，则：

T1 工作在 1T 模式（AUXR.6/T1x12=1）时的输出时钟频率 = (SYSclk)/(256-TH1)/2

T1 工作在 12T 模式（AUXR.6/T1x12=0）时的输出时钟频率 = (SYSclk)/12/(256-TH1)/2

如果 C/T=1，定时器/计数器 T1 是对外部脉冲输入(P3.5/T1)计数，则：

输出时钟频率 = (T1_Pin_CLK) / (256-TH1)/2

13.2.10 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。

13.2.11 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。

13.2.12 辅助寄存器 1（AUXR）

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

13.2.13 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。

13.2.14 定时器 0 计算公式

定时器模式	定时器速度	周期计算公式
模式0/3 （16位自动重载）	1T	定时器周期 $= \frac{65536 - [TH0, TL0]}{SYSclk}$ （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）

STC MCU

13.2.15 定时器 1 计算公式

定时器模式	定时器速度	周期计算公式
模式0 （16位自动重载）	1T	定时器周期 $= \dfrac{65536 - [TH1, TL1]}{SYSclk}$ （自动重载）
	12T	定时器周期 $= \dfrac{65536 - [TH1, TL1]}{SYSclk} \times 12$ （自动重载）
模式1 （16位不自动重载）	1T	定时器周期 $= \dfrac{65536 - [TH1, TL1]}{SYSclk}$ （需软件装载）
	12T	定时器周期 $= \dfrac{65536 - [TH1, TL1]}{SYSclk} \times 12$ （需软件装载）
模式2 （8位自动重载）	1T	定时器周期 $= \dfrac{256 - TH1}{SYSclk}$ （自动重载）
	12T	定时器周期 $= \dfrac{256 - TH1}{SYSclk} \times 12$ （自动重载）

STC MCU

13.3 定时器 2（24 位定时器，8 位预分频+16 位定时）

13.3.1 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2用作定时器或计数器，清0则用作定时器（对内部系统时钟进行计数），置1用作计数器（对引脚T2/P1.2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）

13.3.2 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。

13.3.3 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。

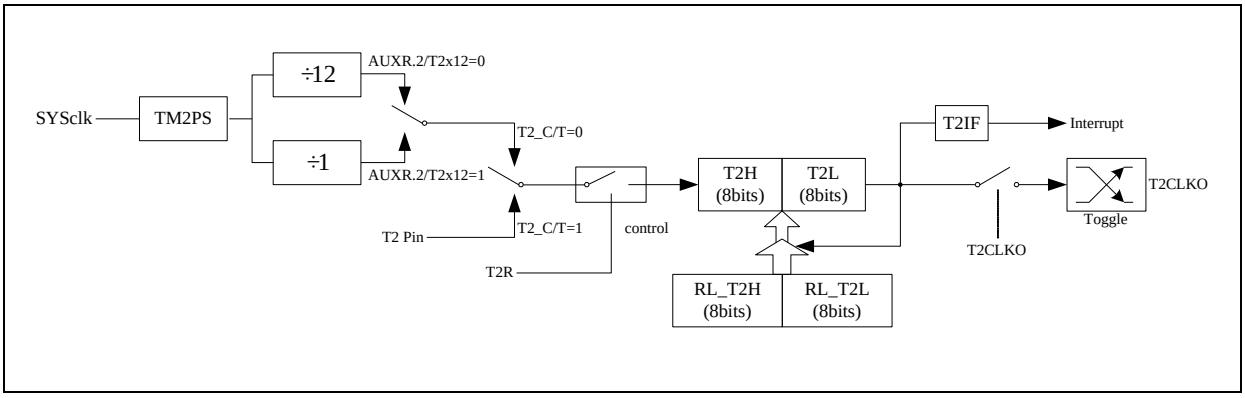
13.3.4 定时器 2 的 8 位预分频寄存器（TM2PS）

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 ÷ (TM2PS + 1)

13.3.5 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]。

13.3.6 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 $= \dfrac{65536 - [T2H, T2L]}{SYSclk/(TM2PS+1)}$ （自动重载）
12T	定时器周期 $= \dfrac{65536 - [T2H, T2L]}{SYSclk/(TM2PS+1)} \times 12$ （自动重载）

13.4 定时器 3/4（24 位定时器，8 位预分频+16 位定时）

13.4.1 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。

13.4.2 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。

13.4.3 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。

13.4.4 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 ÷ (TM3PS + 1)

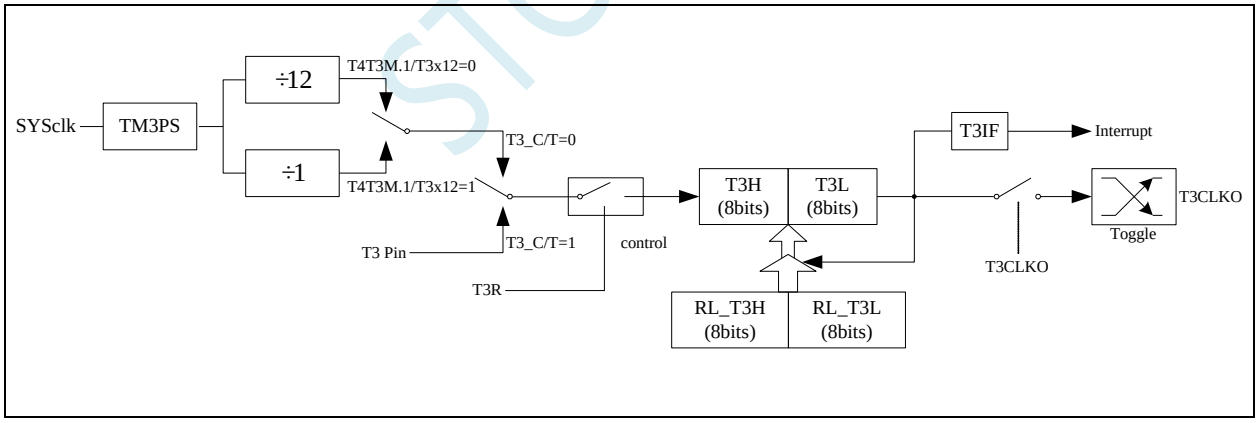
13.4.5 定时器 4 的 8 位预分频寄存器（TM4PS）

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 ÷ (TM4PS + 1)

13.4.6 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 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

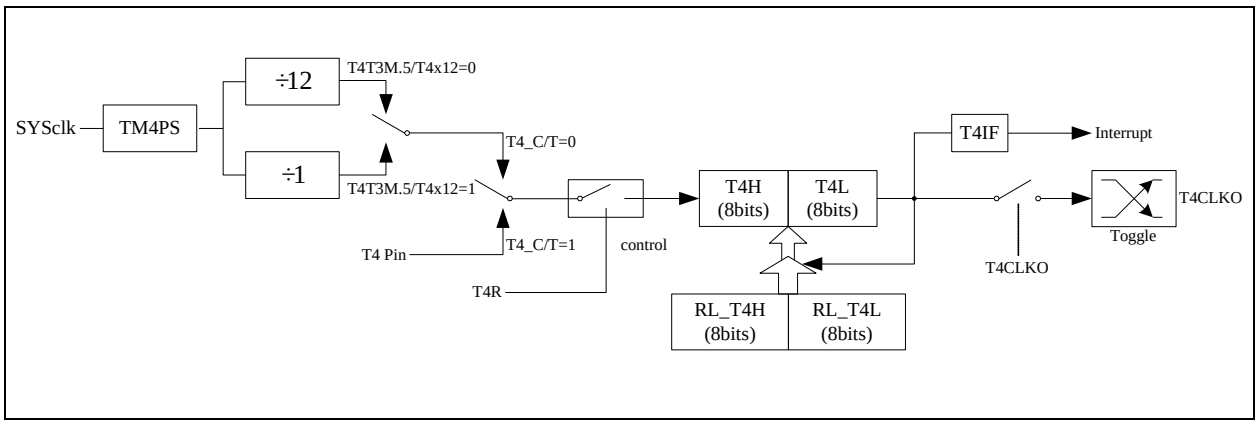
定时器 3 有两个隐藏的寄存器 RL_T3H 和 RL_T3L。RL_T3H 与 T3H 共有同一个地址，RL_T3L 与 T3L 共有同一个地址。当 T3R=0 即定时器/计数器 3 被禁止工作时，对 T3L 写入的内容会同时写入 RL_T3L，对 T3H 写入的内容也会同时写入 RL_T3H。当 T3R=1 即定时器/计数器 3 被允许工作时，对 T3L 写入内容，实际上不是写入当前寄存器 T3L 中，而是写入隐藏的寄存器 RL_T3L 中，对 T3H 写入内容，实际上也不是写入当前寄存器 T3H 中，而是写入隐藏的寄存器 RL_T3H，这样可以巧妙地实现 16 位重装载定时器。当读 T3H 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]。

STC MCU

13.4.7 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]。

13.4.8 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 $= \dfrac{65536 - [T3H, T3L]}{SYSclk/(TM3PS+1)}$ （自动重载）
12T	定时器周期 $= \dfrac{65536 - [T3H, T3L]}{SYSclk/(TM3PS+1)} \times 12$ （自动重载）

13.4.9 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 $= \dfrac{65536 - [T4H, T4L]}{SYSclk/(TM4PS+1)}$ （自动重载）
12T	定时器周期 $= \dfrac{65536 - [T4H, T4L]}{SYSclk/(TM4PS+1)} \times 12$ （自动重载）

STC MCU

13.5 范例程序

13.5.1 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     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 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

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         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,#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
```

13.5.2 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     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 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 1
    TL0 = 0x66;                     //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                        //启动定时器
    ET0 = 1;                        //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

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         000BH
            LJMP        TM0ISR

            ORG         0100H
TM0ISR:
            MOV         TL0,#66H            ;重设定时参数
            MOV         TH0,#0FCH
            CPL         P1.0                ;测试端口
            RETI

MAIN:
            MOV         SP, #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,#01H           ;模式 1
            MOV         TL0,#66H            ;65536-11.0592M/12/1000
            MOV         TH0,#0FCH
            SETB        TR0                 ;启动定时器
            SETB        ET0                 ;使能定时器中断
            SETB        EA

            JMP         $

            END
```

13.5.3 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     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 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

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         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,#02H               ;模式 2
            MOV         TL0,#0F4H               ;256-11.0592M/12/76K
            MOV         TH0,#0F4H
            SETB        TR0                     ;启动定时器
            SETB        ET0                     ;使能定时器中断
            SETB        EA

            JMP         $

            END
```

13.5.4 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     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 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

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     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,#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
```

13.5.5 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     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 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

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         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
```

13.5.6 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     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 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;                    //1T 模式
    TMOD = 0x08;                    //使能 GATE,INT0 为 1 时使能计时
    TL0 = 0x00;
    TH0 = 0x00;
    while (P32);                    //等待 INT0 为低
    TR0 = 1;                        //启动定时器
    IT0 = 1;                        //使能 INT0 下降沿中断
    EX0 = 1;
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR        DATA        8EH
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         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           ;1T 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
```

13.5.7 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     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;

    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
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     TMOD,#00H       ;模式 0
            MOV     TL0,#66H        ;65536-11.0592M/12/1000
            MOV     TH0,#0FCH
            SETB    TR0             ;启动定时器
```

STC MCU

```
        MOV     INTCLKO,#01H        ;使能时钟输出
        JMP     $
        END
```

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

```
    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

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         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         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
            SETB        EA

            JMP         $

            END
```

13.5.9 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     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 TM1_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
    TL1 = 0x66;                     //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                        //启动定时器
    ET1 = 1;                        //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为11.0592MHz

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         001BH
            LJMP        TM1ISR

            ORG         0100H
TM1ISR:
            MOV         TL1,#66H            ;重设定时参数
            MOV         TH1,#0FCH
            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,#10H           ;模式1
            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
            SETB        EA

            JMP         $

            END
```

13.5.10 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     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 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 = 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

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         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,#20H               ;模式 2
            MOV         TL1,#0F4H               ;256-11.0592M/12/76K
            MOV         TH1,#0F4H
            SETB        TR1                     ;启动定时器
            SETB        ET1                     ;使能定时器中断
            SETB        EA

            JMP         $

            END
```

13.5.11 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     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 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
    TL1 = 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

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         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,#40H               ;外部计数模式
            MOV         TL1,#0FFH
            MOV         TH1,#0FFH
            SETB        TR1                     ;启动定时器
            SETB        ET1                     ;使能定时器中断
            SETB        EA

            JMP         $

              END
```

13.5.12 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     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     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;                    //1T 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
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         0013H
            LJMP        INT1ISR

            ORG         0100H
INT1ISR:
            MOV         P0,TL1              ;TL1 为测量值低字节
            MOV         P1,TH1              ;TH1 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,#40H           ;1T 模式
            MOV         TMOD,#80H           ;使能 GATE,INT1 为 1 时使能计时
            MOV         TL1,#00H
            MOV         TH1,#00H
            JB          INT1,$              ;等待 INT1 为低
            SETB        TR1                 ;启动定时器
            SETB        IT1                 ;使能 INT1 下降沿中断
            SETB        EX1
            SETB        EA

            JMP         $

            END
```

13.5.13 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     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;

    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
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     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
```

13.5.14 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       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;

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

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

```
        }
        if (RI)
        {
                RI = 0;
                buffer[wptr++] = SBUF;
                wptr &= 0x0f;
        }
}

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

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

void UartSendStr(char *p)
{
        while (*p)
        {
                UartSend(*p++);
        }
}

void main()
{
        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)
        {
```

STC MCU

```
        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

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         0023H
            LJMP        UART_ISR

            ORG         0100H

UART_ISR:
            PUSH        ACC
            PUSH        PSW
            MOV         PSW,#08H

            JNB         TI,CHKRI
            CLR         TI
            CLR         BUSY
CHKRI:
            JNB         RI,UARTISR_EXIT
            CLR         RI
            MOV         A,WPTR
            ANL         A,#0FH
            ADD         A,#BUFFER
            MOV         R0,A
            MOV         @R0,SBUF
            INC         WPTR
UARTISR_EXIT:
            POP         PSW
            POP         ACC
```

```
            RETI

UART_INIT:
            MOV         SCON,#50H
            MOV         TMOD,#00H
            MOV         TL1,#0E8H               ;65536-11059200/115200/4=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
```

STC MCU

```
            ADD         A,#BUFFER
            MOV         R0,A
            MOV         A,@R0
            LCALL       UART_SEND
            INC         RPTR
            JMP         LOOP

STRING:     DB          'Uart Test !',0DH,0AH,00H

            END
```

13.5.15 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         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;

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                     ;16 bytes

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         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         TL1,#0FDH               ;256-11059200/115200/32=0FDH
            MOV         TH1,#0FDH
```

```
            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
            ADD         A,#BUFFER
            MOV         R0,A
            MOV         A,@R0
            LCALL       UART_SEND
            INC         RPTR
            JMP         LOOP
```

STC MCU

```
STRING:     DB          'Uart Test !',0DH,0AH,00H

            END
```

13.5.16 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     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 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;
```

STC MCU

```
    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

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         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
```

STC MCU

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

        JMP     $

        END
```

13.5.17 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     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 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 = 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

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     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

        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
```

13.5.18 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     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;
```

STC MCU

```
    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

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         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
```

13.5.19 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	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;

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

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         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               ;65536-11059200/115200/4=0FFE8H
            MOV         T2H,#0FFH
            MOV         AUXR,#15H
            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
            ADD         A,#BUFFER
            MOV         R0,A
            MOV         A,@R0
            LCALL       UART_SEND
            INC         RPTR
            JMP         LOOP

STRING:     DB          'Uart Test !',0DH,0AH,00H

            END
```

13.5.20 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	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;

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

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     0043H
            LJMP    UART2_ISR

            ORG     0100H

UART2_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            MOV     A,S2CON
            JNB     ACC.1,CHKRI
            ANL     S2CON,#NOT 02H
            CLR     BUSY
CHKRI:
            JNB     ACC.0,UART2ISR_EXIT
            ANL     S2CON,#NOT 01H
            MOV     A,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           ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#14H
            CLR     BUSY
            MOV     WPTR,#00H
            MOV     RPTR,#00H
            RET

UART2_SEND:
            JB      BUSY,$
            SETB    BUSY
```

```
            MOV         S2BUF,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         RPTR
            JMP         LOOP

STRING:     DB          'Uart Test !',0DH,0AH,00H

            END
```

13.5.21 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
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.1,CHKRI
            ANL     S3CON,#NOT 02H
            CLR     BUSY
CHKRI:
            JNB     ACC.0,UART3ISR_EXIT
            ANL     S3CON,#NOT 01H
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,S3BUF
            INC     WPTR
UART3ISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART3_INIT:
            MOV     S3CON,#10H
            MOV     T2L,#0E8H       ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#14H
            CLR     BUSY
            MOV     WPTR,#00H
            MOV     RPTR,#00H
            RET

UART3_SEND:
            JB      BUSY,$
            SETB    BUSY
            MOV     S3BUF,A
```

```
                RET

UART3_SENDSTR:
                CLR     A
                MOVC    A,@A+DPTR
                JZ      SEND3END
                LCALL   UART3_SEND
                INC     DPTR
                JMP     UART3_SENDSTR
SEND3END:
                RET

MAIN:
                MOV     SP, #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     RPTR
                JMP     LOOP

STRING:         DB      'Uart Test !',0DH,0AH,00H

                END
```

13.5.22 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;
}
```

STC MCU

```
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
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.1,CHKRI
            ANL     S4CON,#NOT 02H
            CLR     BUSY
CHKRI:
            JNB     ACC.0,UART4ISR_EXIT
            ANL     S4CON,#NOT 01H
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,S4BUF
            INC     WPTR
UART4ISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART4_INIT:
            MOV     S4CON,#10H
            MOV     T2L,#0E8H       ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#14H
            CLR     BUSY
            MOV     WPTR,#00H
            MOV     RPTR,#00H
            RET

UART4_SEND:
            JB      BUSY,$
            SETB    BUSY
            MOV     S4BUF,A
```

```
                RET

UART4_SENDSTR:
                CLR     A
                MOVC    A,@A+DPTR
                JZ      SEND4END
                LCALL   UART4_SEND
                INC     DPTR
                JMP     UART4_SENDSTR
SEND4END:
                RET

MAIN:
                MOV     SP, #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   UART4_INIT
                MOV     IE2,#10H
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART4_SENDSTR

LOOP:
                MOV     A,RPTR
                XRL     A,WPTR
                ANL     A,#0FH
                JZ      LOOP
                MOV     A,RPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     A,@R0
                LCALL   UART4_SEND
                INC     RPTR
                JMP     LOOP

STRING:         DB      'Uart Test !',0DH,0AH,00H

                END
```

13.5.23 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    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     009BH
            LJMP    TM3ISR

            ORG     0100H
TM3ISR:
            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     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
```

13.5.24 定时器 3（外部计数一扩展 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             =       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 = 0x0c;                           //设置外部计数模式并启动定时器
        IE2 = ET3;                              //使能定时器中断
        EA = 1;

        while (1);
}
```

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         009BH
            LJMP        TM3ISR

            ORG         0100H
TM3ISR:
            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         T3L,#66H                ;65536-11.0592M/12/1000
            MOV         T3H,#0FCH
            MOV         T4T3M,#0CH              ;设置外部计数模式并启动定时器
            MOV         IE2,#ET3                ;使能定时器中断
            SETB        EA

            JMP         $

            END
```

13.5.25 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    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         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
```

13.5.26 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;
```

STC MCU

```
    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
```

```
            ORG         0000H
            LJMP        MAIN
            ORG         008BH
            LJMP        UART3_ISR

            ORG         0100H

UART3_ISR:
            PUSH        ACC
            PUSH        PSW
            MOV         PSW,#08H

            MOV         A,S3CON
            JNB         ACC.1,CHKRI
            ANL         S3CON,#NOT 02H
            CLR         BUSY
CHKRI:
            JNB         ACC.0,UART3ISR_EXIT
            ANL         S3CON,#NOT 01H
            MOV         A,WPTR
            ANL         A,#0FH
            ADD         A,#BUFFER
            MOV         R0,A
            MOV         @R0,S3BUF
            INC         WPTR
UART3ISR_EXIT:
            POP         PSW
            POP         ACC
            RETI

UART3_INIT:
            MOV         S3CON,#50H
            MOV         T3L,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV         T3H,#0FFH
            MOV         T4T3M,#0AH
            CLR         BUSY
            MOV         WPTR,#00H
            MOV         RPTR,#00H
            RET

UART3_SEND:
            JB          BUSY,$
            SETB        BUSY
            MOV         S3BUF,A
            RET

UART3_SENDSTR:
            CLR         A
            MOVC        A,@A+DPTR
            JZ          SEND3END
            LCALL       UART3_SEND
            INC         DPTR
            JMP         UART3_SENDSTR
SEND3END:
            RET

MAIN:
            MOV         SP, #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         RPTR
            JMP         LOOP

STRING:     DB          'Uart Test !',0DH,0AH,00H

            END
```

13.5.27 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
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,#80H      ;启动定时器
            MOV     IE2,#ET4        ;使能定时器中断
            SETB    EA

            JMP     $
```

STC MCU

END

13.5.28 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
```

13.5.29 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;
    P3M1 = 0x00;

    T4L = 0x66;                    //65536-11.0592M/12/1000
    T4H = 0xfc;
    T4T3M = 0x90;                  //使能时钟输出并启动定时器

    while (1);
}
```

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         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
```

STC MCU

```
        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
```

13.5.30 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];
```

STC MCU

```
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 = 1;
```

STC MCU

```
    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        0D1H
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.1,CHKRI
```

```
            ANL     S4CON,#NOT 02H
            CLR     BUSY
CHKRI:
            JNB     ACC.0,UART4ISR_EXIT
            ANL     S4CON,#NOT 01H
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,S4BUF
            INC     WPTR
UART4ISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART4_INIT:
            MOV     S4CON,#50H
            MOV     T4L,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     T4H,#0FFH
            MOV     T4T3M,#0A0H
            CLR     BUSY
            MOV     WPTR,#00H
            MOV     RPTR,#00H
            RET

UART4_SEND:
            JB      BUSY,$
            SETB    BUSY
            MOV     S4BUF,A
            RET

UART4_SENDSTR:
            CLR     A
            MOVC    A,@A+DPTR
            JZ      SEND4END
            LCALL   UART4_SEND
            INC     DPTR
            JMP     UART4_SENDSTR
SEND4END:
            RET

MAIN:
            MOV     SP, #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   UART4_INIT
            MOV     IE2,#10H
```

```
            SETB        EA

            MOV         DPTR,#STRING
            LCALL       UART4_SENDSTR

LOOP:
            MOV         A,RPTR
            XRL         A,WPTR
            ANL         A,#0FH
            JZ          LOOP
            MOV         A,RPTR
            ANL         A,#0FH
            ADD         A,#BUFFER
            MOV         R0,A
            MOV         A,@R0
            LCALL       UART4_SEND
            INC         RPTR
            JMP         LOOP

STRING:     DB          'Uart Test !',0DH,0AH,00H

            END
```

STC MCU

14 串口通信

STC12H 系列单片机具有 2 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。

STC12H 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 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。

14.1 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
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

STCMCU

14.2 串口 1

14.2.1 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。

14.2.2 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。

14.2.3 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、模式 2、模式 3 的波特率加倍

SMOD0：帧错误检测控制位

0：无帧错检测功能

1：使能帧错误检测功能。此时 SCON 的 SM0/FE 为 FE 功能，即为帧错误检测标志位。

14.2.4 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*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

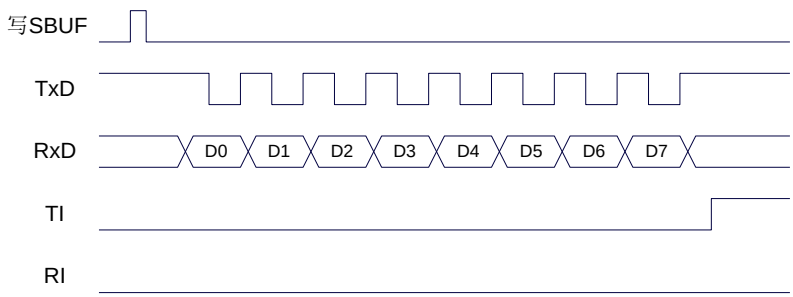
1：选择定时器 2 作为波特率发生器

14.2.5 串口 1 模式 0，模式 0 波特率计算公式

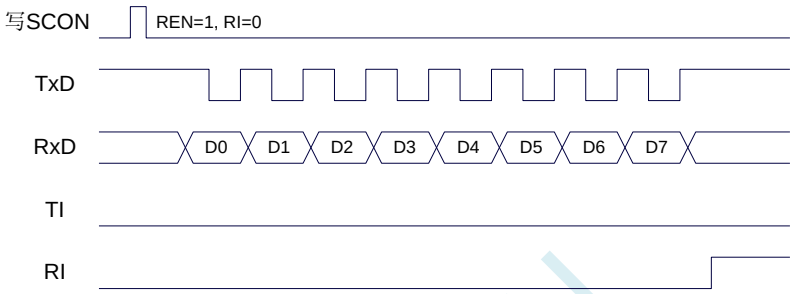
当串口 1 选择工作模式为模式 0 时，串行通信接口工作在同步移位寄存器模式，当串行口模式 0 的通信速度设置位 UART_M0x6 为 0 时，其波特率固定为系统时钟时钟的 12 分频（SYSclk/12）；当设置 UART_M0x6 为 1 时，其波特率固定为系统时钟频率的 2 分频（SYSclk/2）。RxD 为串行通讯的数据口，TxD 为同步移位脉冲输出脚，发送、接收的是 8 位数据，低位在先。

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



发送数据（串口1模式0）



接收数据（串口1模式0）

工作于模式 0 时，必须清 0 多机通信控制位 SM2，使之不影响 TB8 位和 RB8 位。由于波特率固定为 SYSclk/12 或 SYSclk/2，无需定时器提供，直接由单片机的时钟作为同步移位脉冲。

串口 1 模式 0 的波特率计算公式如下表所示（SYSclk 为系统工作频率）：

UART_M0x6	波特率计算公式
0	波特率 $= \frac{SYSclk}{12}$
1	波特率 $= \frac{SYSclk}{2}$

14.2.6 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。

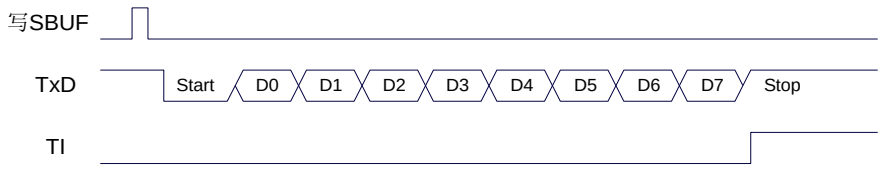
模式 1 的发送过程：串行通信模式发送时，数据由串行发送端 TxD 输出。当主机执行一条写 SBUF 的指令就启动串行通信的发送，写“SBUF”信号还把“1”装入发送移位寄存器的第 9 位，并通知 TX 控制单元开始发送。移位寄存器将数据不断右移送 TxD 端口发送，在数据的左边不断移入“0”作补充。当数据的最高位移到移位寄存器的输出位置，紧跟其后的是第 9 位“1”，在它的左边各位全为“0”，这个状态条件，使 TX 控制单元作最后一次移位输出，然后使允许发送信号“SEND”失效，完成一帧信息的发送，并置位中断请求位 TI，即 TI=1，向主机请求中断处理。

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：

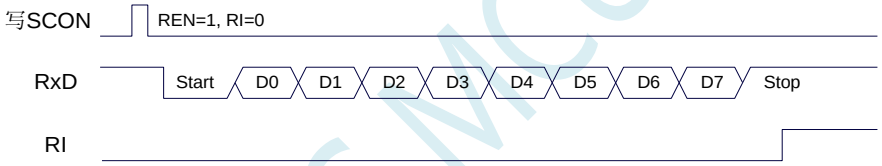
• RI=0；

• 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"。



发送数据（串口1模式1）



接收数据（串口1模式1）

串口 1 的波特率是可变的，其波特率可由定时器 1 或者定时器 2 产生。当定时器采用 1T 模式时（12 倍速），相应的波特率的速度也会相应提高 12 倍。

串口 1 模式 1 的波特率计算公式如下表所示：（SYSclk 为系统工作频率）

选择定时器	定时器速度	波特率计算公式
定时器2	1T	定时器2重载值 $= 65536 - \frac{SYSclk}{4\times波特率}$
	12T	定时器2重载值 $= 65536 - \frac{SYSclk}{12\times4\times波特率}$
定时器1模式0	1T	定时器1重载值 $= 65536 - \frac{SYSclk}{4\times波特率}$
	12T	定时器1重载值 $= 65536 - \frac{SYSclk}{12\times4\times波特率}$
定时器1模式2	1T	定时器1重载值 $= 256 - \frac{2^{SMOD}\times SYSclk}{32\times波特率}$
	12T	定时器1重载值 $= 256 - \frac{2^{SMOD}\times SYSclk}{12\times32\times波特率}$

下面为常用频率与常用波特率所对应定时器的重载值

频率 (MHz)	波特率	定时器 2		定时器 1 模式 0		定时器 1 模式 2			
						SMOD=1		SMOD=0	
		1T 模式	12T 模式	1T 模式	12T 模式	1T 模式	12T 模式	1T 模式	12T 模式
11.0592	115200	FFE8H	FFFEH	FFE8H	FFFEH	FAH	-	FDH	-
	57600	FFD0H	FFFCH	FFD0H	FFFCH	F4H	FFH	FAH	-
	38400	FFB8H	FFFAH	FFB8H	FFFAH	EEH	-	F7H	-
	19200	FF70H	FFF4H	FF70H	FFF4H	DCH	FDH	EEH	-
	9600	FEE0H	FFE8H	FEE0H	FFE8H	B8H	FAH	DCH	FDH
18.432	115200	FFD8H	-	FFD8H	-	F6H	-	FBH	-
	57600	FFB0H	-	FFB0H	-	ECH	-	F6H	-
	38400	FF88H	FFF6H	FF88H	FFF6H	E2H	-	F1H	-
	19200	FF10H	FFECH	FF10H	FFECH	C4H	FBH	E2H	-
	9600	FE20H	FFD8H	FE20H	FFD8H	88H	F6H	C4H	FBH
22.1184	115200	FFD0H	FFFCH	FFD0H	FFFCH	F4H	FFH	FAH	-
	57600	FFA0H	FFF8H	FFA0H	FFF8H	E8H	FEH	F4H	FFH
	38400	FF70H	FFF4H	FF70H	FFF4H	DCH	FDH	EEH	-
	19200	FEE0H	FFE8H	FEE0H	FFE8H	B8H	FAH	DCH	FDH
	9600	FDC0H	FFD0H	FDC0H	FFD0H	70H	F4H	B8H	FAH

14.2.7　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 $= \dfrac{\text{SYSclk}}{64}$
1	波特率 $= \dfrac{\text{SYSclk}}{32}$

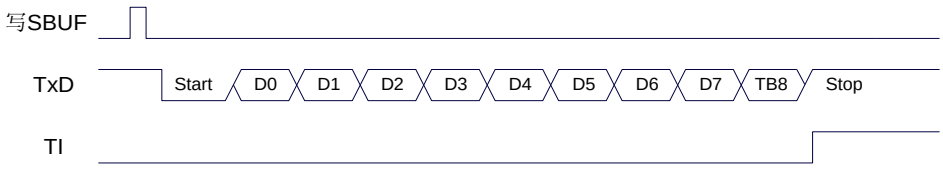
模式 2 和模式 1 相比，除波特率发生源略有不同，发送时由 TB8 提供给移位寄存器第 9 数据位不同外，其余功能结构均基本相同，其接收/发送操作过程及时序也基本相同。

当接收器接收完一帧信息后必须同时满足下列条件：

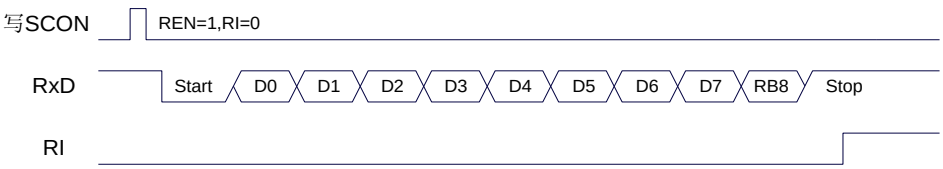
- RI=0
- SM2=0 或者 SM2=1 且接收到的第 9 数据位 RB8=1。

当上述两条件同时满足时，才将接收到的移位寄存器的数据装入 SBUF 和 RB8 中，RI 标志位被置 1，并向主机请求中断处理。如果上述条件有一个不满足，则刚接收到移位寄存器中的数据无效而丢失，也不置位 RI。无论上述条件满足与否，接收器又重新开始检测 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 Ð 议的约定，为多机通信提供了方便。



发送数据（串口1模式2）



接收数据（串口1模式2）

14.2.8　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。

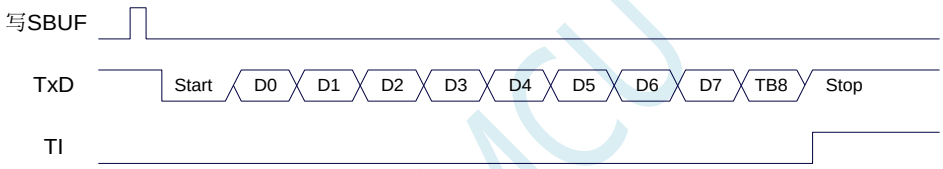
当接收器接收完一帧信息后必须同时满足下列条件：

• RI=0

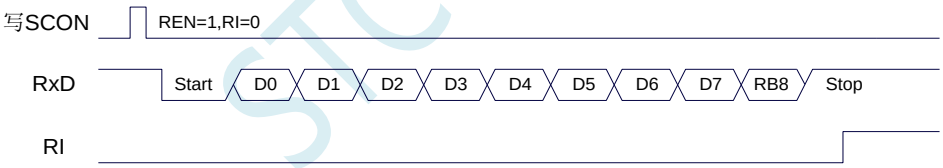
• SM2=0 或者 SM2=1 且接收到的第 9 数据位 RB8=1。

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。



发送数据（串口1模式3）



接收数据（串口1模式3）

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。

14.2.9 自动地址识别

14.2.10串口 1 从机地址控制寄存器（SADDR，SADEN）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SADDR	A9H								
SADEN	B9H								

SADDR：从机地址寄存器

SADEN：从机地址屏蔽位寄存器

自动地址识别功能典型应用在多机通讯领域，其主要原理是从机系统通过硬件比较功能来识别来自于主机串口数据流中的地址信息，通过寄存器 SADDR 和 SADEN 设置的本机的从机地址，硬件自动对从机地址进行过滤，当来自于主机的从机地址信息与本机所设置的从机地址相匹配时，硬件产生串口中

断；否则硬件自动丢弃串口数据，而不产生中断。当众多处于空闲模式的从机链接在一起时，只有从机地址相匹配的从机才会从空闲模式唤醒，从而可以大大降低从机 MCU 的功耗，即使从机处于正常工作状态也可避免不停地进入串口中断而降低系统执行效率。

要使用串口的自动地址识别功能，首先需要将参与通讯的 MCU 的串口通讯模式设置为模式 2 或者模式 3（通常都选择波特率可变的模式 3，因为模式 2 的波特率是固定的，不便于调节），并开启从机的 SCON 的 SM2 位。对于串口模式 2 或者模式 3 的 9 位数据位中，第 9 位数据（存放在 RB8 中）为地址/数据的标志位，当第 9 位数据为 1 时，表示前面的 8 位数据（存放在 SBUF 中）为地址信息。当 SM2 被设置为 1 时，从机 MCU 会自动过滤掉非地址数据（第 9 位为 0 的数据），而对 SBUF 中的地址数据（第 9 位为 1 的数据）自动与 SADDR 和 SADEN 所设置的本机地址进行比较，若地址相匹配，则会将 RI 置“1”，并产生中断，否则不予处理本次接收的串口数据。

从机地址的设置是通过 SADDR 和 SADEN 两个寄存器进行设置的。SADDR 为从机地址寄存器，里面存放本机的从机地址。SADEN 为从机地址屏蔽位寄存器，用于设置地址信息中的忽略位，设置方法如下：

例如
SADDR ＝ 11001010
SADEN ＝ 10000001
则匹配地址为 1xxxxxx0
即，只要主机送出的地址数据中的 bit0 为 0 且 bit7 为 1 就可以和本机地址相匹配

再例如
SADDR ＝ 11001010
SADEN ＝ 00001111
则匹配地址为 xxxx1010
即，只要主机送出的地址数据中的低 4 位为 1010 就可以和本机地址相匹配，而高 4 为被忽略，可以为任意值。

主机可以使用广播地址（FFH）同时选中所有的从机来进行通讯。

STC MCU

14.3 串口 2

14.3.1 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。

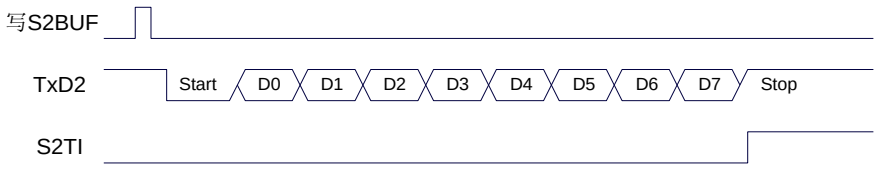
14.3.2 串口 2 数据寄存器（S2BUF）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S2BUF	9BH								

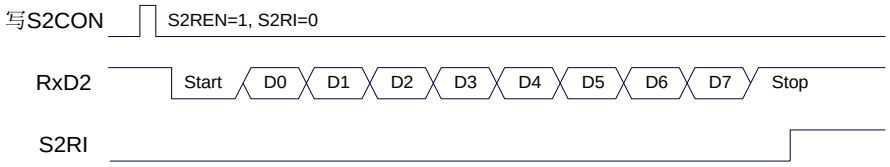
S2BUF：串口 1 数据接收/发送缓冲区。S2BUF 实际是 2 个缓冲器，读缓冲器和写缓冲器，两个操作分别对应两个不同的寄存器，1 个是只写寄存器（写缓冲器），1 个是只读寄存器（读缓冲器）。对 S2BUF 进行读操作，实际是读取串口接收缓冲区，对 S2BUF 进行写操作则是触发串口开始发送数据。

14.3.3 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。



发送数据（串口2模式0）



接收数据（串口2模式0）

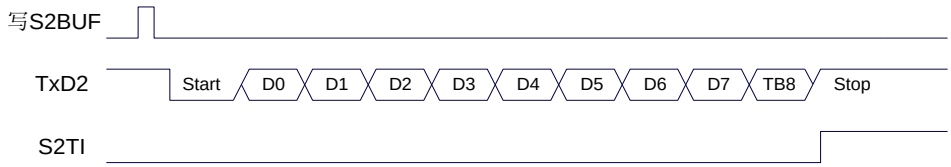
串口 2 的波特率是可变的，其波特率由定时器 2 产生。当定时器采用 1T 模式时（12 倍速），相应的波特率的速度也会相应提高 12 倍。

串口 2 模式 0 的波特率计算公式如下表所示：（SYSclk 为系统工作频率）

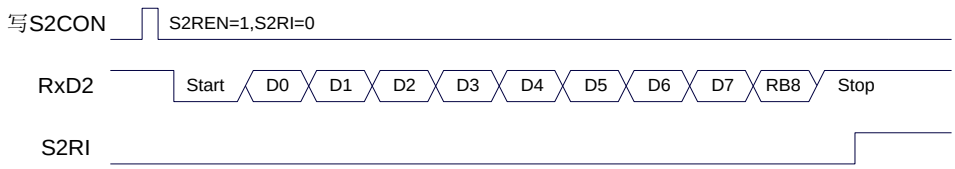
选择定时器	定时器速度	波特率计算公式
定时器2	1T	定时器2重载值 = 65536 − $\frac{\text{SYSclk}}{4\times\text{波特率}}$
	12T	定时器2重载值 = 65536 − $\frac{\text{SYSclk}}{12\times4\times\text{波特率}}$

14.3.4　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。



发送数据（串口2模式1）



接收数据（串口2模式1）

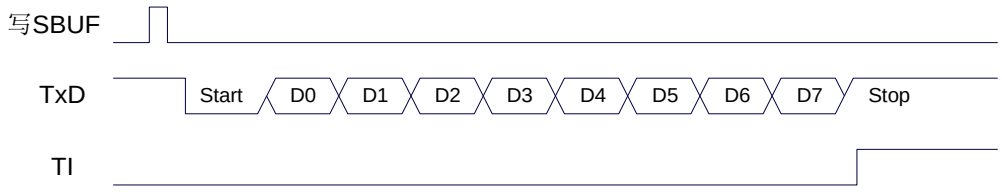
串口 2 模式 1 的波特率计算公式与模式 0 是完全相同的。请参考模式 0 的波特率计算公式。

STC MCU

14.4 串口注意事项

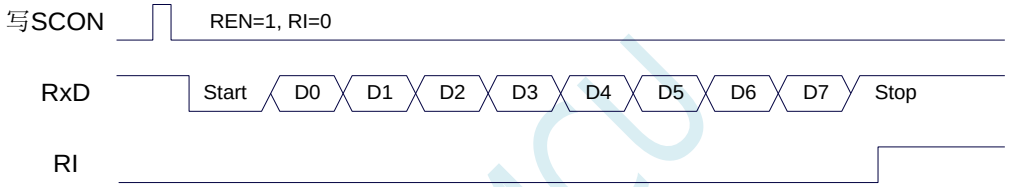
关于串口中断请求有如下问题需要注意：（串口 1、串口 2、串口 3、串口 4 均类似，下面以串口 1 为例进行说明）

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：



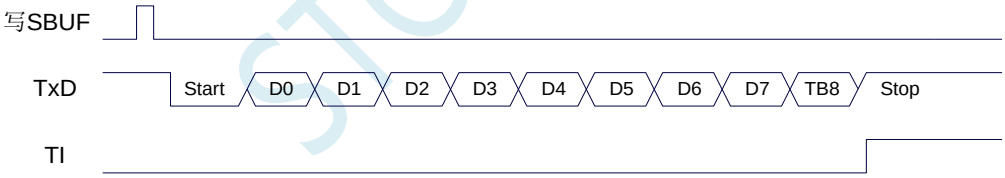
发送数据（8位数据）

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：



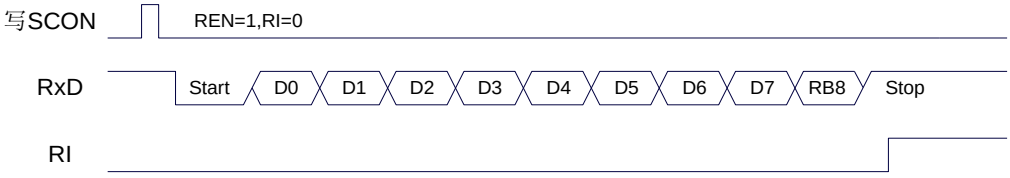
接收数据（8位数据）

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：



发送数据（9位数据）

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：



接收数据（9位数据）

14.5 范例程序

14.5.1 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           ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#15H
            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
            ADD         A,#BUFFER
            MOV         R0,A
            MOV         A,@R0
            LCALL       UART_SEND
            INC         RPTR
            JMP         LOOP

STRING:     DB          'Uart Test !',0DH,0AH,00H

            END
```

14.5.2 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;
	TL1 = BRT;
	TH1 = BRT >> 8;
	TR1 = 1;
	AUXR = 0x40;
	wptr = 0x00;
	rptr = 0x00;
	busy = 0;
}

void UartSend(char dat)
{
```

```
        while (busy);
        busy = 1;
        SBUF = dat;
}

void UartSendStr(char *p)
{
        while (*p)
        {
                UartSend(*p++);
        }
}

void main()
{
        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                     ;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         TMOD,#00H
            MOV         TL1,#0E8H           ;65536-11059200/115200/4=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
```

STC MCU

```
            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
            ADD         A,#BUFFER
            MOV         R0,A
            MOV         A,@R0
            LCALL       UART_SEND
            INC         RPTR
            JMP         LOOP

STRING:     DB          'Uart Test !',0DH,0AH,00H

            END
```

14.5.3 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;
    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)
{
```

STC MCU

```
        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                     ;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     TMOD,#20H
            MOV     TL1,#0FDH           ;256-11059200/115200/32=0FDH
            MOV     TH1,#0FDH
            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:
```

STC MCU

```
            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         RPTR
            JMP         LOOP

STRING:     DB          'Uart Test !',0DH,0AH,00H

            END
```

14.5.4 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++);
    }
```

STC MCU

```
}

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.1,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           ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#14H
            CLR     BUSY
            MOV     WPTR,#00H
            MOV     RPTR,#00H
            RET

UART2_SEND:
            JB      BUSY,$
            SETB    BUSY
            MOV     S2BUF,A
            RET

UART2_SENDSTR:
            CLR     A
            MOVC    A,@A+DPTR
            JZ      SEND2END
            LCALL   UART2_SEND
            INC     DPTR
            JMP     UART2_SENDSTR
SEND2END:
            RET

MAIN:
```

STC MCU

```
        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     RPTR
        JMP     LOOP

STRING: DB      'Uart Test !',0DH,0AH,00H

        END
```

STC MCU

14.5.5 串口多机通讯

现参考 STC15 系列数据手册，后续补充

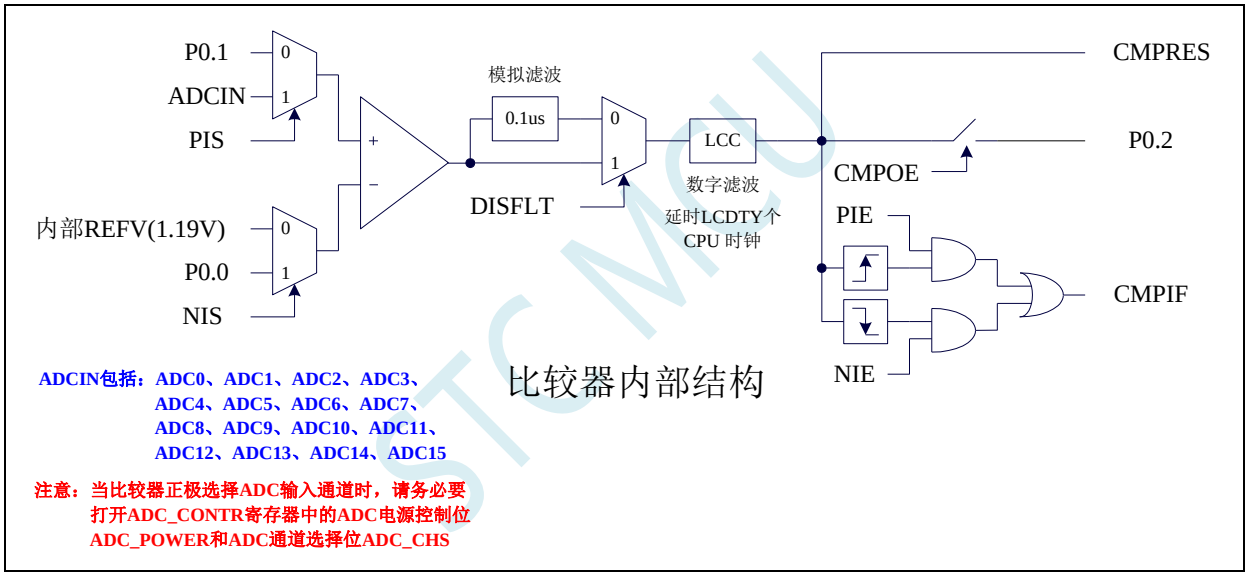
STC MCU

15 比较器，掉电检测，内部 1.19V 参考信号源

STC12H 系列单片机内部集成了一个比较器。比较器的正极可以是 P0.1 端口或者 ADC 的模拟输入通道，而负极可以 P0.0 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。

15.1 比较器内部结构图



15.2 比较器相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CMPCR1	比较器控制寄存器 1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES	0000,0000
CMPCR2	比较器控制寄存器 2	E7H	INVCMPO	DISFLT	LCDTY[5:0]						0000,0000

15.2.1 比较器控制寄存器 1（CMPCR1）

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

CMPEN：比较器模块使能位

0：关闭比较功能

1：使能比较功能

CMPIF：比较器中断标志位。当 PIE 或 NIE 被使能后，若产生相应的中断信号，硬件自动将 CMPIF 置 1，并向 CPU 提出中断请求。此标志位必须用户软件清零。

（注意：没有使能比较器中断时，硬件不会设置此中断标志，即使用查询方式访问比较器时，不能查询此中断标志）

PIE：比较器上升沿中断使能位。

0：禁止比较器上升沿中断。

1：使能比较器上升沿中断。使能比较器的比较结果由 0 变成 1 时产生中断请求。

NIE：比较器下降沿中断使能位。

0：禁止比较器下降沿中断。

1：使能比较器下降沿中断。使能比较器的比较结果由 1 变成 0 时产生中断请求。

PIS：比较器的正极选择位

0：选择外部端口 P0.1 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 P0.1，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，内部参考信号源调整为 1.19V）

1：选择外部端口 P0.0 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 P0.2

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。

15.2.2 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 P0.2 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 P0.2 输出高电平，反之输出低电平。

DISFLT：模拟滤波功能控制

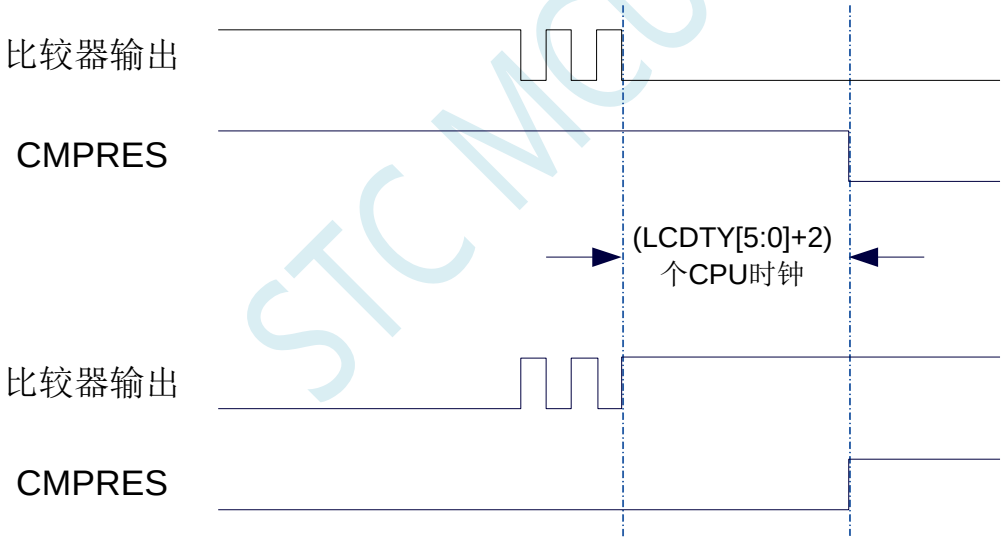
0：使能 0.1us 模拟滤波功能

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



15.3 范例程序

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

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 CMP_Isr() interrupt 21
{
    CMPCR1 &= ~0x40;                    //清中断标志
    if (CMPCR1 & 0x01)
    {
        P10 = !P10;                     //下降沿中断测试端口
    }
    else
    {
        P11 = !P11;                     //上升沿中断测试端口
    }
}

void main()
{
    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;                    //P0.1 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;                     //P0.0 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

    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

CMPCR1      DATA        0E6H
CMPCR2      DATA        0E7H

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         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:
        CPL     P1.0            ;下降沿中断测试端口
        POP     ACC
        RETI
RSING:
        CPL     P1.1            ;上升沿中断测试端口
        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     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     ;P0.1 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         ;P0.0 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
        SETB    EA

        JMP     $

        END
```

15.3.2 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     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;

    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;                    //P0.1 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;                     //P0.0 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

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         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         ;P0.1 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         ;P0.0 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              ;读取比较器比较结果
            JMP     LOOP

            END
```

15.3.3 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;
```

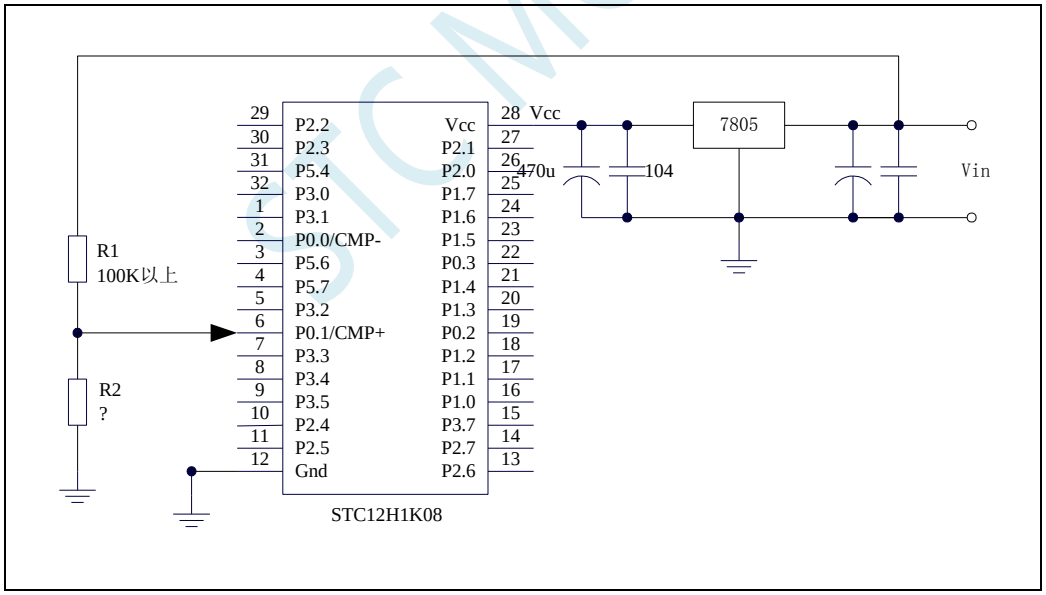
```
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P1M0 &= 0xfe;                          //设置 P1.0 为输入口
    P1M1 |= 0x01;
    ADC_CONTR = 0x80;                      //使能 ADC 模块并选择 P1.0 为 ADC 输入脚

    CMPCR2 = 0x00;
    CMPCR1 = 0x00;

    CMPCR1 |= 0x08;                        //ADC 输入脚为 CMP+输入脚
    CMPCR1 |= 0x04;                        //P0.0 为 CMP-输入脚
    CMPCR1 |= 0x02;                        //使能比较器输出
    CMPCR1 |= 0x80;                        //使能比较器模块

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

CMPCR1      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         ;P0.0 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
```

15.3.4　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）

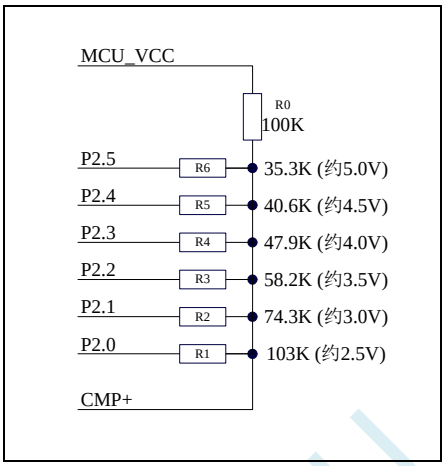


上图中电阻 R1 和 R2 对稳压块 7805 的前端电压进行分压，分压后的电压作为比较器 CMP+的外部输入与内部 1.19V 参考信号源进行比较。

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，内部参考信号源调整为 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

15.3.5 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 IO 口输出低电平，端口电压值接近 Gnd，未选通的通道，MCU 的 IO 口输出开漏模式的高，不影响其他通道）。

比较器的负端选择内部 1.19V 参考信号源，正端选择通过电阻分压后输入到 CMP+管脚的电压值。

初始化时 P2.5~P2.0 口均设置为开漏模式，并输出高。首先 P2.0 口输出低电平，此时若 Vcc 电压低于 2.5V 则比较器的比较值为 0，反之若 Vcc 电压高于 2.5V 则比较器的比较值为 1；

若确定 Vcc 高于 2.5V，则将 P2.0 口输出高，P2.1 口输出低电平，此时若 Vcc 电压低于 3.0V 则比较器的比较值为 0，反之若 Vcc 电压高于 3.0V 则比较器的比较值为 1；

若确定 Vcc 高于 3.0V，则将 P2.1 口输出高，P2.2 口输出低电平，此时若 Vcc 电压低于 3.5V 则比较器的比较值为 0，反之若 Vcc 电压高于 3.5V 则比较器的比较值为 1；

若确定 Vcc 高于 3.5V，则将 P2.2 口输出高，P2.3 口输出低电平，此时若 Vcc 电压低于 4.0V 则比较器的比较值为 0，反之若 Vcc 电压高于 4.0V 则比较器的比较值为 1；

若确定 Vcc 高于 4.0V，则将 P2.3 口输出高，P2.4 口输出低电平，此时若 Vcc 电压低于 4.5V 则比较器的比较值为 0，反之若 Vcc 电压高于 4.5V 则比较器的比较值为 1；

若确定 Vcc 高于 4.5V，则将 P2.4 口输出高，P2.5 口输出低电平，此时若 Vcc 电压低于 5.0V 则比较器的比较值为 0，反之若 Vcc 电压高于 5.0V 则比较器的比较值为 1。

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     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     P2M0    =   0x96;
sfr     P2M1    =   0x95;

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;

    P2M0 = 0x3f;                        //P2.5~P2.0 初始化为开漏模式
    P2M1 = 0x3f;
    P2 = 0xff;

    CMPCR2 = 0x10;                      //比较器结果经过 16 个去抖时钟后输出
    CMPCR1 = 0x00;
    CMPCR1 &= ~0x08;                    //P3.7 为 CMP+输入脚
    CMPCR1 &= ~0x04;                    //内部 1.19V 参考信号源为 CMP-输入脚
    CMPCR1 &= ~0x02;                    //禁止比较器输出
    CMPCR1 |= 0x80;                     //使能比较器模块

    while (1)
    {
        v = 0x00;                       //电压<2.5V
        P2 = 0xfe;                      //P2.0 输出 0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x01;                       //电压>2.5V
        P2 = 0xfd;                      //P2.1 输出 0
```

```
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x03;                                   //电压>3.0V
        P2 = 0xfb;                                  //P2.2 输出 0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x07;                                   //电压>3.5V
        P2 = 0xf7;                                  //P2.3 输出 0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x0f;                                   //电压>4.0V
        P2 = 0xef;                                  //P2.4 输出 0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x1f;                                   //电压>4.5V
        P2 = 0xdf;                                  //P2.5 输出 0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x3f;                                   //电压>5.0V
ShowVol:
        P2 = 0xff;
        P0 = ~v;
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

CMPCR1      DATA        0E6H
CMPCR2      DATA        0E7H

P2M0        DATA        096H
P2M1        DATA        095H
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     P2M0,#00111111B         ;P2.5~P2.0 初始化为开漏模式
        MOV     P2M1,#00111111B
        MOV     P2,#0FFH
        MOV     CMPCR2,#10H             ;比较器结果经过 16 个去抖时钟后输出
        MOV     CMPCR1,#00H
        ANL     CMPCR1,#NOT 08H         ;P3.7 为CMP+输入脚
        ANL     CMPCR1,#NOT 04H         ;内部 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     P2,#11111110B           ;P2.0 输出 0
        CALL    DELAY
        MOV     A,CMPCR1
        JNB     ACC.0,SKIP
        MOV     R0,#00000001B           ;电压>2.5V
        MOV     P2,#11111101B           ;P2.1 输出 0
        CALL    DELAY
        MOV     A,CMPCR1
        JNB     ACC.0,SKIP
        MOV     R0,#00000011B           ;电压>3.0V
        MOV     P2,#11111011B           P2.2 输出 0
        CALL    DELAY
        MOV     A,CMPCR1
        JNB     ACC.0,SKIP
        MOV     R0,#00000111B           ;电压>3.5V
        MOV     P2,#11110111B           ;P2.3 输出 0
        CALL    DELAY
        MOV     A,CMPCR1
        JNB     ACC.0,SKIP
        MOV     R0,#00001111B           ;电压>4.0V
        MOV     P2,#11101111B           ;P2.4 输出 0
        CALL    DELAY
        MOV     A,CMPCR1
        JNB     ACC.0,SKIP
        MOV     R0,#00011111B           ;电压>4.5V
        MOV     P2,#11011111B           ;P2.5 输出 0
        CALL    DELAY
        MOV     A,CMPCR1
        JNB     ACC.0,SKIP
        MOV     R0,#00111111B           ;电压>5.0V
SKIP:
        MOV     P2,#11111111B
        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

STC MCU

16 IAP/EEPROM/DATA-FLASH

STC12H 系列单片机内部集成了大容量的 EEPROM。利用 ISP/IAP 技术可将内部 Data Flash 当 EEPROM，擦写次数在 10 万次以上。EEPROM 可分为若干个扇区，每个扇区包含 512 字节。

注意：EEPROM 的写操作只能将字节中的 1 写为 0，当需要将字节中的 0 写为 1，则必须执行扇区擦除操作。EEPROM 的读/写操作是以 1 字节为单位进行，而 EEPROM 擦除操作是以 1 扇区（512 字节）为单位进行，在执行擦除操作时，如果目标扇区中有需要保留的数据，则必须预先将这些数据读取到 RAM 中暂存，待擦除完成后再将保存的数据和需要更新的数据一起再写回 EEPROM/DATA-FLASH。

所以在使用 EEPROM 时，建议同一次修改的数据放在同一个扇区，不是同一次修改的数据放在不同的扇区，不一定要用满。数据存储器的擦除操作是按扇区进行的（每扇区 512 字节）。

EEPROM 可用于保存一些需要在应用过程中修改并且掉电不丢失的参数数据。在用户程序中，可以对 EEPROM 进行字节读/字节编程/扇区擦除操作。在工作电压偏低时，建议不要进行 EEPROM 操作，以免发送数据丢失的情况。

16.1 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

16.2 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

16.2.1 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。

16.2.2 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

16.2.3 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。

16.2.4 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。

16.2.5 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

16.2.6 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

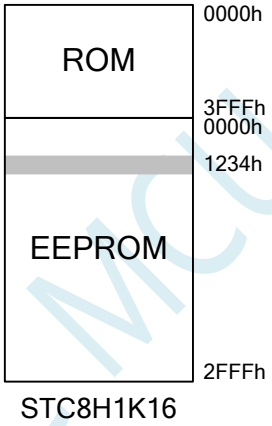
若工作频率为12MHz，则需要将IAP_TPS设置为12；若工作频率为24MHz，则需要将IAP_TPS设置为24，其他频率以此类推。

STC MCU

16.3 EEPROM 大小及地址

STC12H 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。

STC12H 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 STC12H1K16 这个型号为例，对目标地址进行详细说明：



STC12H1K16 的程序空间为 16K 字节（0000h~3FFFh），EEPROM 空间为 12K（0000h~2FFFh）。当需要对 EEPROM 物理地址 1234h 的单元进行读、写、擦除时，若使用 IAP 方式进行访问时，设置的目标地址为 1234h，即 IAP_ADDRH 设置 12h，IAP_ADDRL 设置 34h，然后设置相应的触发命令即可对 1234h 单元进行正确操作了。但若是使用 MOVC 方式读取 EEPROM 的 1234h 单元，则必须在 1234h 的基础上还有加上 ROM 空间的大小 4000h，即必须将 DPTR 设置为 5234h，然后才能使用 MOVC 指令进行读取。

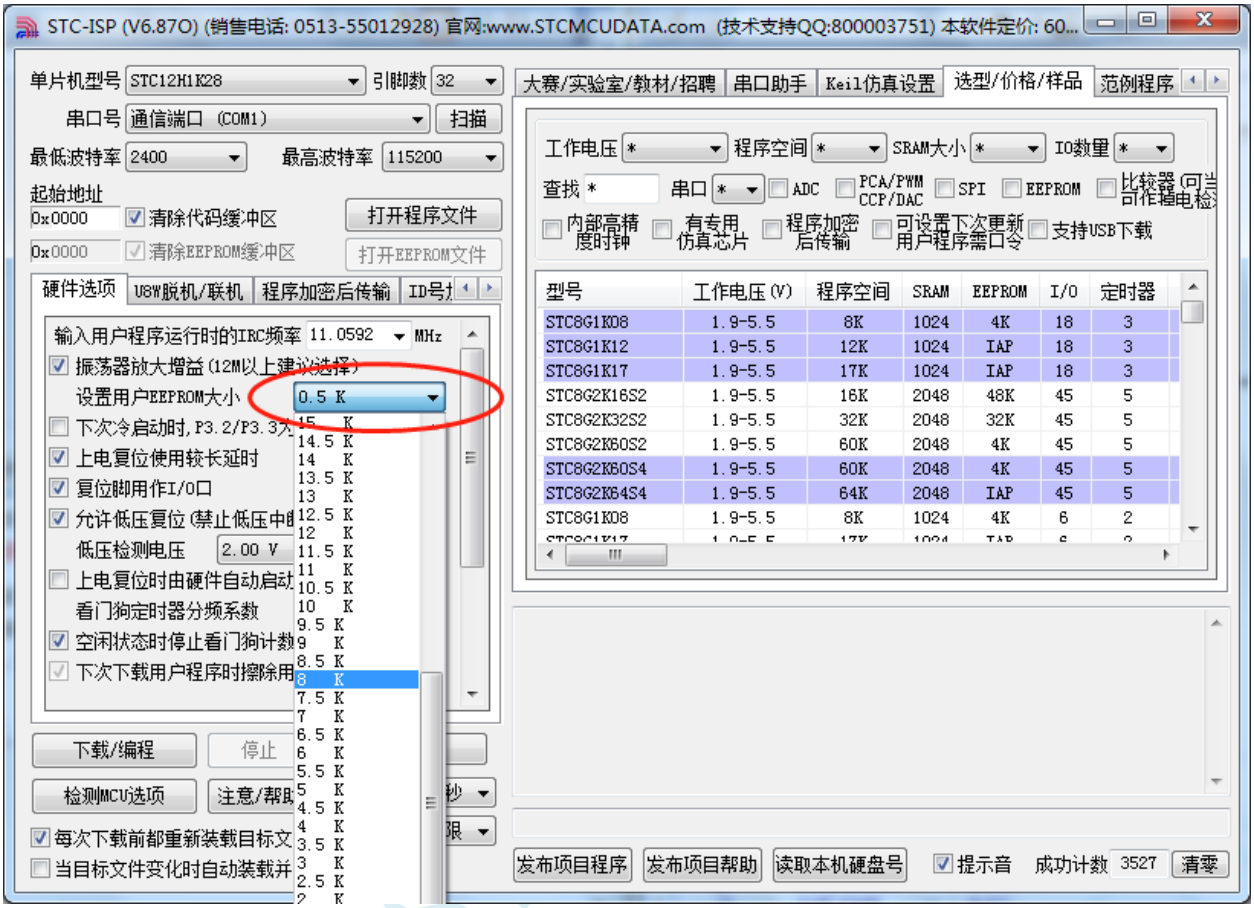
注意：由于擦除是以 512 字节为单位进行操作的，所以执行擦除操作时所设置的目标地址的低 9 位是无意义的。例如：执行擦除命令时，设置地址 1234H/1200H/1300H/13FFH，最终执行擦除的动作都是相同的，都是擦除 1200H~13FFH 这 512 字节。

不同型号内部 EEPROM 的大小及访问地址会存在差异，针对各个型号 EEPROM 的详细大小和地址请参考下表

型号	大小	扇区	IAP方式读/写/擦除		MOVC读取	
			起始地址	结束地址	起始地址	结束地址
STC12H1K08	4K	8	0000h	0FFFh	2000h	6FFFh
STC12H1K16	12K	24	0000h	2FFFh	4000h	6FFFh
STC12H1K24	4K	8	0000h	0FFFh	6000h	6FFFh
STC12H1K28	用户自定义[1]					

STC12H1K33	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：STC12H1K28 这个型号的 FLASH 为 28K，此时若用户想分出其中的 8K 作为 EEPROM 使用，则 EEPROM 的物理地址则为 28K 的最后 8K，物理地址为 5000h~6FFFh，当然，用户若使用 IAP 的方式进行访问，目标地址仍然从 0000h 开始，到 1FFFh 结束，当使用 MOVC 读取则需要从 5000h 开始，到 6FFFh 结束。

16.4 范例程序

16.4.1 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     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     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 = 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;

    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
```

```
IAP_ADDRL   DATA    0C4H
IAP_CMD     DATA    0C5H
IAP_TRIG    DATA    0C6H
IAP_CONTR   DATA    0C7H
IAP_TPS     DATA    0F5H

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

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

        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
```

16.4.2 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     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     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      0x4000H                 //STC12H1K16

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
}
```

STC MCU

```
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
IAP_ADDRL   DATA    0C4H
IAP_CMD     DATA    0C5H
IAP_TRIG    DATA    0C6H
IAP_CONTR   DATA    0C7H
IAP_TPS     DATA    0F5H

IAP_OFFSET  EQU     4000H               ;STC12H1K16

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

IAP_IDLE:
            MOV     IAP_CONTR,#0        ;关闭 IAP 功能
            MOV     IAP_CMD,#0          ;清除命令寄存器
            MOV     IAP_TRIG,#0         ;清除触发寄存器
```

```
        MOV     IAP_ADDRH,#80H      ;将地址设置到非 IAP 区域
        MOV     IAP_ADDRL,#0
        RET

IAP_READ:
        MOV     A,#LOW IAP_OFFSET   ;使用 MOVC 读取 EEPROM 需要加上相应的偏移
        ADD     A,DPL
        MOV     DPL,A
        MOV     A,@HIGH IAP_OFFSET
        ADDC    A,DPH
        MOV     DPH,A
        CLR     A
        MOVC    A,@A+DPTR           ;使用 MOVC 读取数据
        RET

IAP_PROGRAM:
        MOV     IAP_CONTR,#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
```

16.4.3 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     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     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;
}
```

STC MCU

```
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

AUXR        DATA        8EH
T2H         DATA        0D6H
T2L         DATA        0D7H

IAP_DATA    DATA        0C2H
IAP_ADDRH   DATA        0C3H
IAP_ADDRL   DATA        0C4H
IAP_CMD     DATA        0C5H
IAP_TRIG    DATA        0C6H
IAP_CONTR   DATA        0C7H
IAP_TPS     DATA        0F5H

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
```

```
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
```

STCMCU

```
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
```

STC MCU

17 ADC 模数转换，内部 1.19V 参考信号源

STC12H1K08 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）。

STC12H 系列的 ADC 最快速度：**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参考信号源，参考信号源值出厂时校准为 1.19V，由于制造误差以及测量误差，导致实际的内部参考信号源相比 1.19V，大约有±1%的误差。如果用户需要知道每一颗芯片的准确内部参考信号源值，可外接精准参考信号源，然后利用 ADC 的第 15 通道进行测量标定。

17.1 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

17.1.1 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 16 位高级 PWM 定时器章节

ADC_CHS[3:0]：ADC 模拟通道选择位

（注意：被选择为 ADC 输入通道的 I/O 口，必须设置 PxM0/PxM1 寄存器将 I/O 口模式设置为高阻输入模式。另外如果 MCU 进入掉电模式/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）

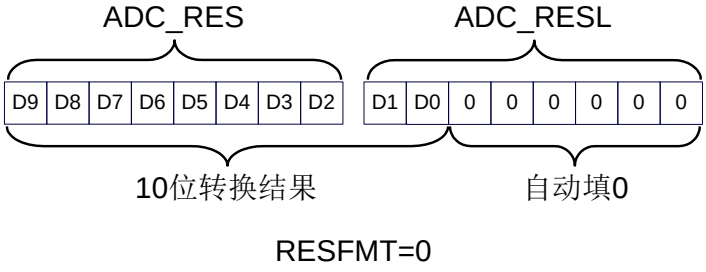
ADC_CHS[3:0]	ADC 通道
0000	P1.0/ADC0
0001	P1.1/ADC1
0010	P1.2/ADC2
0011	P1.3/ADC3
0100	P1.4/ADC4
0101	P1.5/ADC5
0110	P1.6/ADC6
0111	P1.7/ADC7
1000	P2.0/ADC8
1001	P2.1/ADC9
1010	P2.4/ADC10
1011	P2.5/ADC11
1100	P2.6/ADC12
1101	P2.7/ADC13
1110	P3.7/ADC14
1111	测试内部 1.19V

17.1.2 ADC 配置寄存器（ADCCFG）

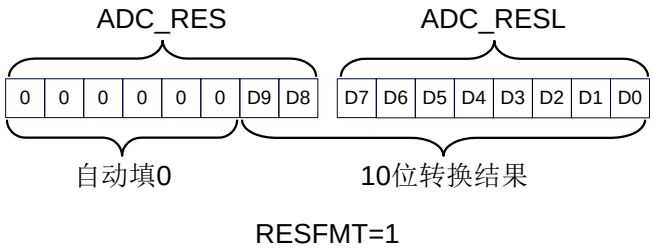
符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADCCFG	DEH	-	-	RESFMT	-	SPEED[3:0]			

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

17.1.3 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。

17.1.4 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**）

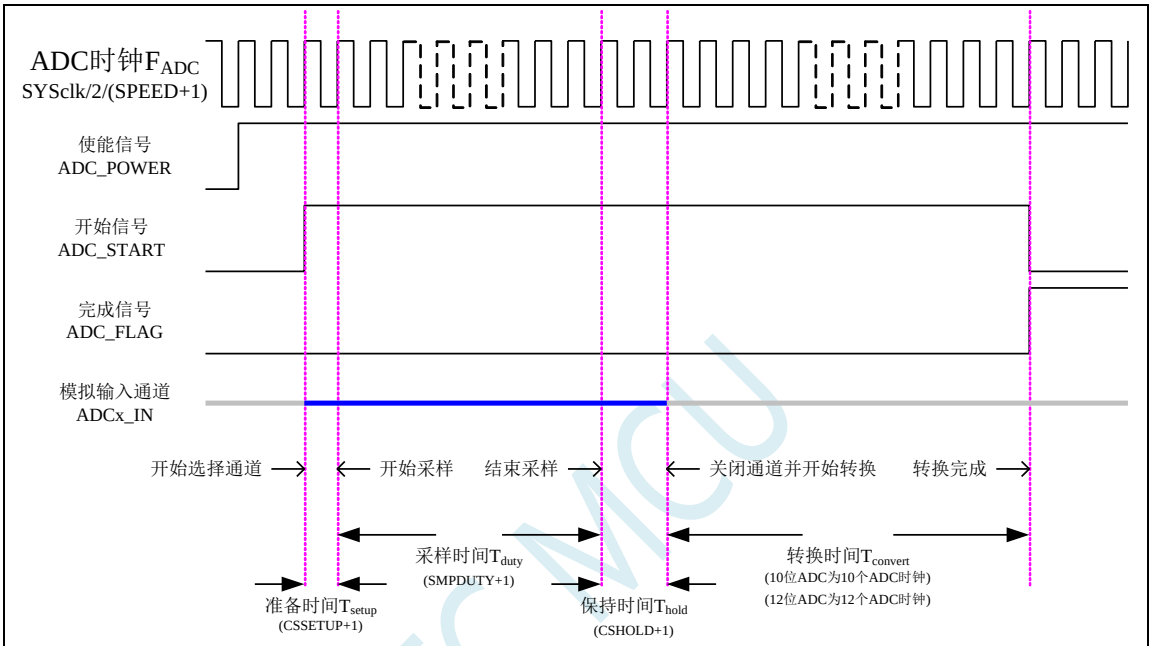
SMPDUTY[4:0]	占用 ADC 工作时钟数
00000	1
00001	2

...	...
01010	**11（默认值）**
...	...
11110	31
11111	32

ADC 数模转换时间：$T_{convert}$

10 位 ADC 的转换时间固定为 10 个 ADC 工作时钟

一个完整的 ADC 转换时间为： $T_{setup} + T_{duty} + T_{hold} + T_{convert}$，如下图所示



ADC整体转换时序图

STCMCU

17.2 ADC 相关计算公式

17.2.1 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：

$$10\text{位ADC转换速度} = \frac{\text{MCU工作频率SYSclk}}{2\times(\text{SPEED[3:0]} + 1)\times[(\text{CSSETUP} + 1) + (\text{CSHOLD} + 1) + (\text{SMPDUTY} + 1) + 10]}$$

注意：

- 10 位 ADC 的速度不能高于 500KHz
- 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）

17.2.2 ADC 转换结果计算公式

$$10\text{位ADC转换结果} = 1024 \times \frac{\text{ADC被转换通道的输入电压Vin}}{\text{MCU工作电压Vcc}}$$

17.2.3 反推 ADC 输入电压计算公式

$$\text{ADC被转换通道的输入电压Vin} = \text{MCU工作电压Vcc} \times \frac{10\text{位ADC转换结果}}{1024}$$

17.2.4 反推工作电压计算公式

$$\text{MCU工作电压Vcc} = 1024 \times \frac{\text{ADC被转换通道的输入电压Vin}}{\text{10位ADC转换结果}}$$

17.3 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

STC MCU

17.4 范例程序

17.4.1 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 cha volatile xdata *)0xfea8)

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;

    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

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     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
```

STC MCU

```
        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
LOOP:
        ORL     ADC_CONTR,#40H      ;启动 AD 转换
        NOP
        NOP
        MOV     A,ADC_CONTR         ;查询 ADC 完成标志
        JNB     ACC.5,$-2
        ANL     ADC_CONTR,#NOT 20H  ;清完成标志
        MOV     P2,ADC_RES          ;读取 ADC 结果

        SJMP    LOOP

        END
```

17.4.2 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     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 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;
    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

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     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
```

17.4.3 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     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;

    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
    ADC_CONTR |= 0x40;                  //启动 AD 转换
    _nop_();
    _nop_();
    while (!(ADC_CONTR & 0x20));        //查询 ADC 完成标志
    ADC_CONTR &= ~0x20;                 //清完成标志

    ADCCFG = 0x00;                      //设置结果左对齐
    ACC = ADC_RES;                      //A 存储 ADC 的 10 位结果的高 8 位
    B = ADC_RESL;                       //B[7:6]存储 ADC 的 10 位结果的低 2 位,B[5:0]为 0

//  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 存储 ADC 的 10 位结果的低 8 位

    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

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         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

            ORL         ADC_CONTR,#40H          ;启动 AD 转换
            NOP
            NOP
            MOV         A,ADC_CONTR             ;查询 ADC 完成标志
            JNB         ACC.5,$-2
            ANL         ADC_CONTR,#NOT 20H      ;清完成标志

            MOV         ADCCFG,#00H             ;设置结果左对齐
            MOV         A,ADC_RES               ;A 存储 ADC 的 10 位结果的高 8 位
            MOV         B,ADC_RESL              ;B[7:6]存储 ADC 的 10 位结果的低 2 位,B[5:0]为 0

;           MOV         ADCCFG,#20H             ;设置结果右对齐
```

```
;           MOV     A,ADC_RES           ;A[3:0]存储ADC 的10 位结果的高2 位,A[7:2]为0
;           MOV     B,ADC_RESL          ;B 存储ADC 的10 位结果的低8 位

            SJMP    $

            END
```

17.4.4 利用 ADC 第 15 通道测量外部电压或电池电压

STC12H 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。

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         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;

int         *BGV;                           //内部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;
        TL1 = 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;
        ADCTIM = 0x3f;                          //设置 ADC 内部时序
        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;
    ADCInit();                                  //ADC 初始化
    UartInit();                                 //串口初始化

    ES = 1;
    EA = 1;

//  ADCRead();
//  ADCRead();                                  //前两个数据建议丢弃

    res = 0;
    for (i=0; i<8; i++)
    {
        res += ADCRead();                       //读取 8 次数据
    }
    res >>= 3;                                  //取平均值

    vcc = (int)(4096L * *BGV / res);            //(12 位 ADC 算法) 计算 VREF 管脚电压,即电池电压
//  vcc = (int)(1024L * *BGV / res);            //(10 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**；

17.4.5 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。

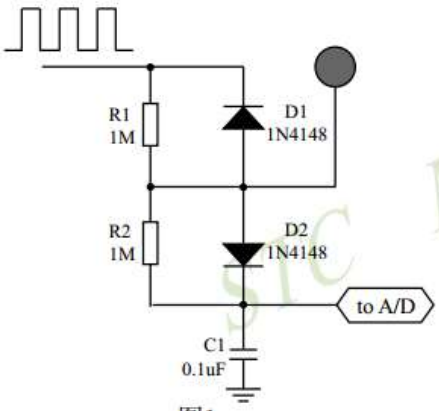


图1

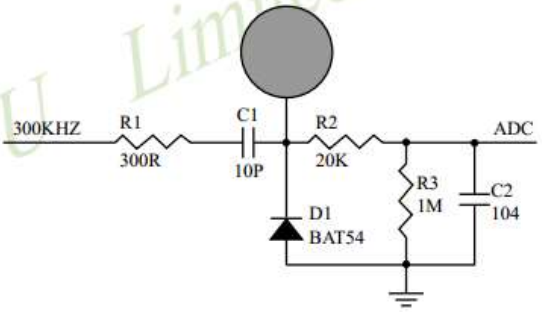


图2

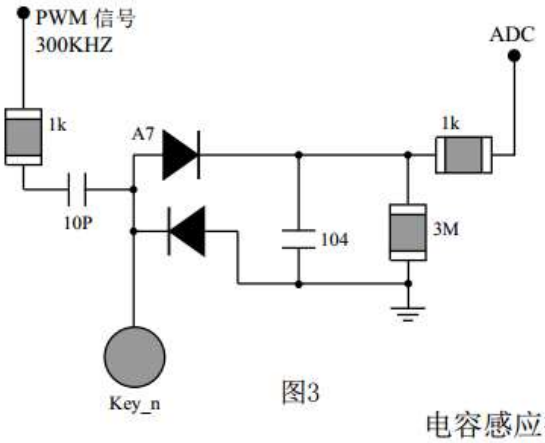


图3

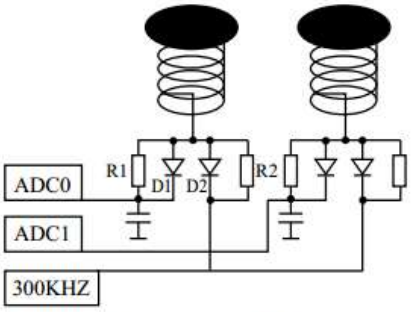
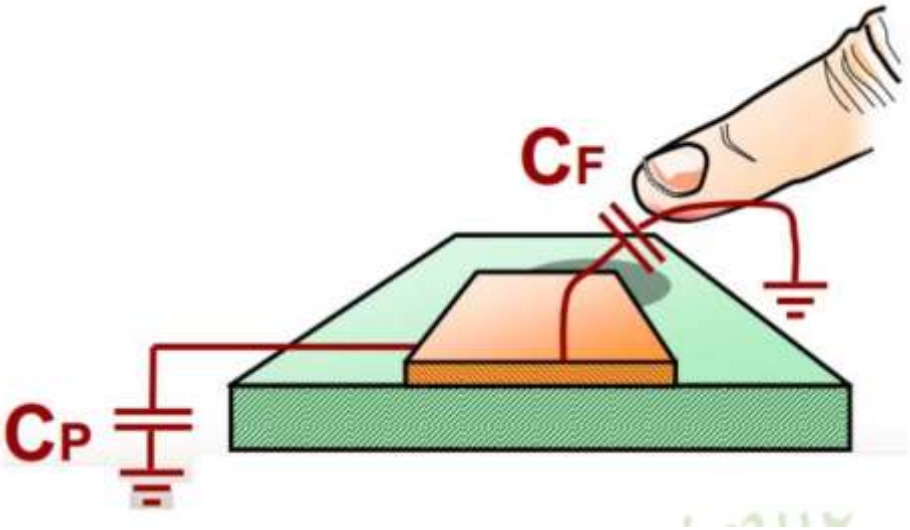


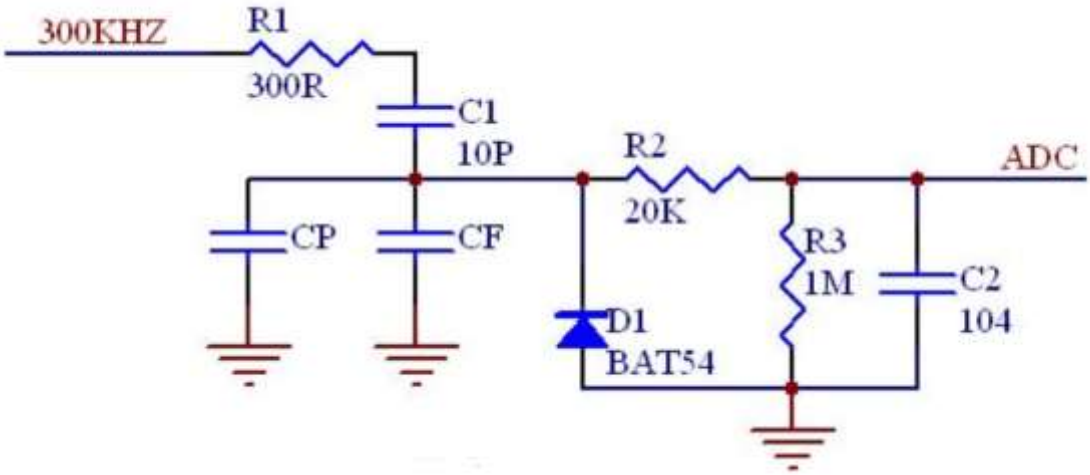
图4 加了感应弹簧

电容感应按键取样电路

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[TOUCH_CHANNEL];                   //当前 ADC 值
u16 idata adc_prev[TOUCH_CHANNEL];              //上一个 ADC 值
u16 idata TouchZero[TOUCH_CHANNEL];             //0 点 ADC 值
u8 idata TouchZeroCnt[TOUCH_CHANNEL];           //0 点自动跟踪计数
u8 cnt_250ms;

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;                               //Timer0 set as 1T mode
    AUXR2 |= 0x01;                              //允许输出时钟
    TMOD = 0;                                   //Timer0 set as Timer, 16 bits Auto Reload.
    TH0 = (u8)(Timer0_Reload >> 8);
    TL0 = (u8)Timer0_Reload;
    TR0 = 1;
    ADC_init();                                 //ADC 初始化
    delay_ms(50);                               //延时 50ms
    for (i=0; i<TOUCH_CHANNEL; i++)             //初始化 0 点和上一个值和 0 点自动跟踪计数
    {
        adc_prev[i] = 1023;
        TouchZero[i] = 1023;
        TouchZeroCnt[i] = 0;
    }
    cnt_250ms = 0;
    while (1)
    {
        delay_ms(50);                           //每隔 50ms 处理一次按键
```

```
            ShowLED();
            if (++cnt_250ms >= 5)
            {
                cnt_250ms = 0;
                AutoZero();                                  //每隔 250ms 处理一次 0 点自动跟踪
            }
        }
}

void delay_ms(u8 ms)
{
        unsigned int i;

        do
        {
            i = MAIN_Fosc / 13000;
            while(--i) ;
        } while(--ms);
}

void ADC_init(void)
{
        P1M0 = 0x00;                                         //8 路 ADC
        P1M1 = 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
                                                             //这是使用相邻 2 个采样的差的绝对值之和来检测。
{
        u8 i;
        u16 j,k;

        for(i=0; i<TOUCH_CHANNEL; i++)                       //处理 8 个通道
        {
            j = adc[i];
            k = j - adc_prev[i];                             //减前一个读数
            F0 = 0;                                          //按下
            if(k & 0x8000) F0 = 1, k = 0 - k;                //释放 求出两次采样的差值
            if(k >= 20)                                      //变化比较大
            {
                TouchZeroCnt[i] = 0;                         //如果变化比较大，则清 0 计数器
                if(F0) TouchZero[i] = j;                     //如果是释放，并且变化比较大，则直接替代
            }
            else                                             //变化比较小，则蠕动，自动 0 点跟踪
```

```
            {
                if(++TouchZeroCnt[i] >= 20)                 //连续检测到小变化 20 次/4 = 5 秒.
                {
                    TouchZeroCnt[i] = 0;
                    TouchZero[i] = adc_prev[i];             //变化缓慢的值作为 0 点     }
            }
            adc_prev[i] = j;                                //保存这一次的采样值
        }
}

u8 check_adc(u8 index)                                      //获取触摸信息函数 50ms 调用 1 次
                                                            //判断键按下或释放,有回差控制
{
        u16 delta;

        adc[index] = 1023 - Get_ADC10bitResult(index);      //获取 ADC 值, 转成按下键, ADC 值增加
        if(adc[index] < TouchZero[index]) return 0;         //比 0 点还小的值，则认为是键释放
        delta = adc[index] - TouchZero[index];
        if(delta >= 40) return 1;                           //键按下
        if(delta <= 20) return 0;                           //键释放
        return 2;                                           //保持原状态
}

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

Fosc_KHZ    EQU         24000                               ;定义主时钟 KHZ
```

```
Reload      EQU     (65536 - Fosc_KHZ/600)      ;Timer 0 重装值,对应 300KHZ

ADC_CONTR   DATA    0xBC                        ;带 AD 系列
ADC_RES     DATA    0xBD                        ;带 AD 系列
ADC_RESL    DATA    0xBE                        ;带 AD 系列
AUXR        DATA    0x8E
AUXR2       DATA    0x8F

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

CHANNEL     EQU     8                           ;ADC 通道数
ADC_90T     EQU     (3 SHL 5)                   ;ADC 时间 90T
ADC_180T    EQU     (2 SHL 5)                   ;ADC 时间 180T
ADC_360T    EQU     (1 SHL 5)                   ;ADC 时间 360T
ADC_540T    EQU     0                           ;ADC 时间 540T
ADC_FLAG    EQU     (1 SHL 4)                   ;软件清 0
ADC_START   EQU     (1 SHL 3)                   ;自动清 0

P_LED7      BIT     P2.7;
P_LED6      BIT     P2.6;
P_LED5      BIT     P2.5;
P_LED4      BIT     P2.4;
P_LED3      BIT     P2.3;
P_LED2      BIT     P2.2;
P_LED1      BIT     P2.1;
P_LED0      BIT     P2.0;
adc         EQU     30H                         ;当前 ADC 值 30H~3FH,两字节一个值
adc_prev    EQU     40H                         ;上一个 ADC 值 40H~4FH,两字节一个值
TouchZero   EQU     50H                         ;0 点 ADC 值 50H~5FH,两字节一个值
TouchZeroCnt EQU    60H                         ;0 点自动跟踪计数 60H~67H
cnt_250ms   DATA    68H

            ORG     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              ;Timer0 set as 1T mode
            ORL     AUXR2,#01H              ;允许输出时钟
            MOV     TMOD,#0                 ;Timer0 set as Timer,16 bits Auto Reload.
            MOV     TH0,#HIGH 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            ;[R2 R3] ,20,转
            MOV     A,#LOW (TouchZeroCnt)   ;如果变化比较大，则清 0 计数器 TouchZeroCnt[i] = 0;
            ADD     A,R5
            MOV     R0,A
            MOV     @R0,#0
            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     R1,A
        MOV     A,R4
        ADD     A,ACC
        ADD     A,#LOW (adc)
        MOV     R0,A
        MOV     A,@R0
        MOV     R6,A
        INC     R0
        MOV     A,@R0
        CLR     C
        SUBB    A,@R1                           ;计算 adc[index] - TouchZero[index]
        MOV     A,R6
        DEC     R1
        SUBB    A,@R1
        JNC     L_check_adc_1
        MOV     R7,#00H
        RET
L_check_adc_1:
        MOV     A,R4
        ADD     A,ACC
        ADD     A,#LOW (TouchZero+01H)
        MOV     R1,A
        MOV     A,R4
        ADD     A,ACC
        ADD     A,#LOW (adc+01H)
        MOV     R0,A
        CLR     C
        MOV     A,@R0
        SUBB    A,@R1
        MOV     R7,A
        DEC     R0
        MOV     A,@R0
        DEC     R1
        SUBB    A,@R1
        MOV     R6,A
        CLR     C
        MOV     A,R7
        SUBB    A,#40
```

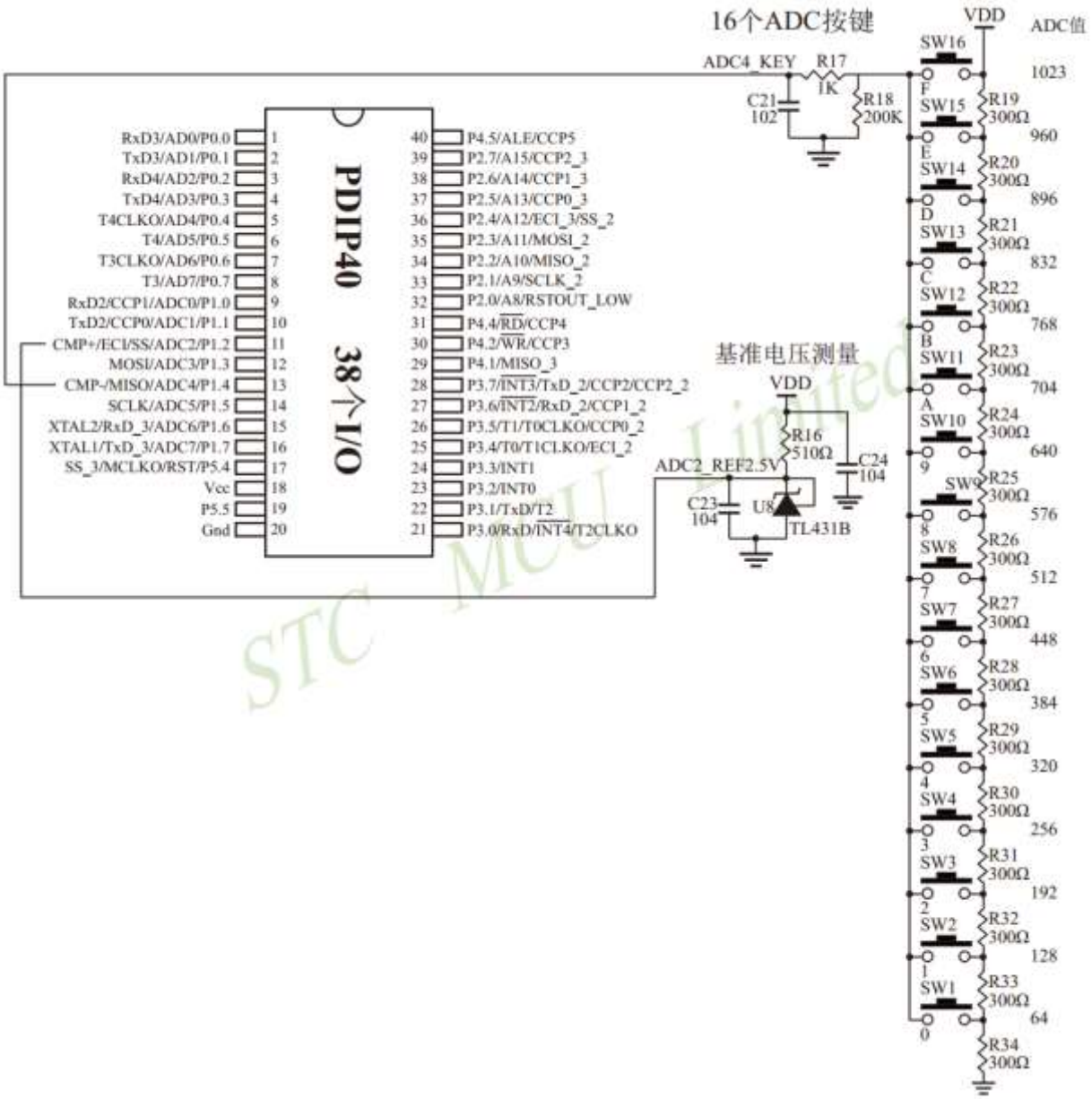
```
            MOV     A,R6
            SUBB    A,#00H
            JC      L_check_adc_2       ;if(delta < 40),转
            MOV     R7,#1               ;if(delta >= 40) return 1; //键按下 返回1
            RET
L_check_adc_2:
            SETB    C
            MOV     A,R7
            SUBB    A,#20
            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
            CPL     C
            MOV     P_LED1,C
L_QuitCheck1:
            MOV     R7,#2
            LCALL   F_check_adc
            MOV     A,R7
            ANL     A,#0FEH
            JNZ     L_QuitCheck2
            MOV     A,R7
            MOV     C,ACC.0
            CPL     C
            MOV     P_LED2,C
L_QuitCheck2:
            MOV     R7,#3
            LCALL   F_check_adc
            MOV     A,R7
            ANL     A,#0FEH
            JNZ     L_QuitCheck3
            MOV     A,R7
            MOV     C,ACC.0
            CPL     C
            MOV     P_LED3,C
L_QuitCheck3:
```

```
            MOV     R7,#4
            LCALL   F_check_adc
            MOV     A,R7
            ANL     A,#0FEH
            JNZ     L_QuitCheck4
            MOV     A,R7
            MOV     C,ACC.0
            CPL     C
            MOV     P_LED4,C
L_QuitCheck4:
            MOV     R7,#5
            LCALL   F_check_adc
            MOV     A,R7
            ANL     A,#0FEH
            JNZ     L_QuitCheck5
            MOV     A,R7
            MOV     C,ACC.0
            CPL     C
            MOV     P_LED5,C
L_QuitCheck5:
            MOV     R7,#6
            LCALL   F_check_adc
            MOV     A,R7
            ANL     A,#0FEH
            JNZ     L_QuitCheck6
            MOV     A,R7
            MOV     C,ACC.0
            CPL     C
            MOV     P_LED6,C
L_QuitCheck6:
            MOV     R7,#7
            LCALL   F_check_adc
            MOV     A,R7
            ANL     A,#0FEH
            JNZ     L_QuitCheck7
            MOV     A,R7
            MOV     C,ACC.0
            CPL     C
            MOV     P_LED7,C
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
```

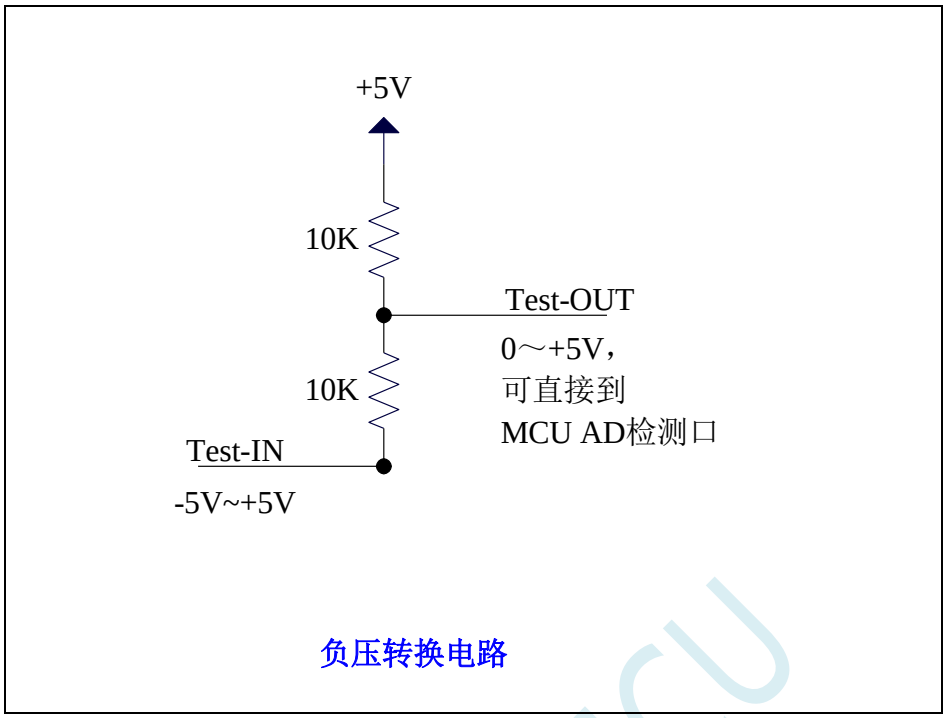
```
        POP     3
        RET

        END
```

17.4.6 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。

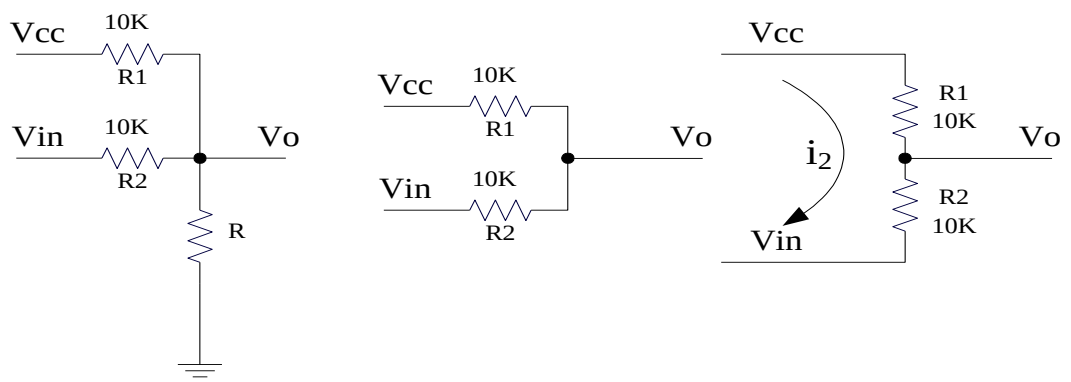


17.4.7 检测负电压参考线路图



STC MCU

17.4.8 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

公式 1：Vo = Vin + i_2 * R2

公式 2：i_2 = (Vcc - Vin) / (R1 + R2)｛条件：流向 Vo 的电流 ≈0｝

将 R1=R2 代入公式 2 得公式 3

公式 3：i_2 = (Vcc - Vin) / 2R2

将公式 3 代入公式 1 得公式 4

公式 4：Vo = (Vcc + Vin) / 2

根据公式 4，可以将以上电路看成加法电路。

在单片机的模数转换测量中，要求被测电压大于 0 并且小于 VCC。如果被测电压小于 0V，可以利用加法电路将被测电压提升到 0V 以上。此时对被测电压的变化范围有一定的要求：

把上述条件代入公式 4 可得到下面 2 式

(Vcc + Vin) / 2 > 0　　即 Vin > -Vcc

(Vcc + Vin) / 2 < Vcc　　即 Vin < Vcc

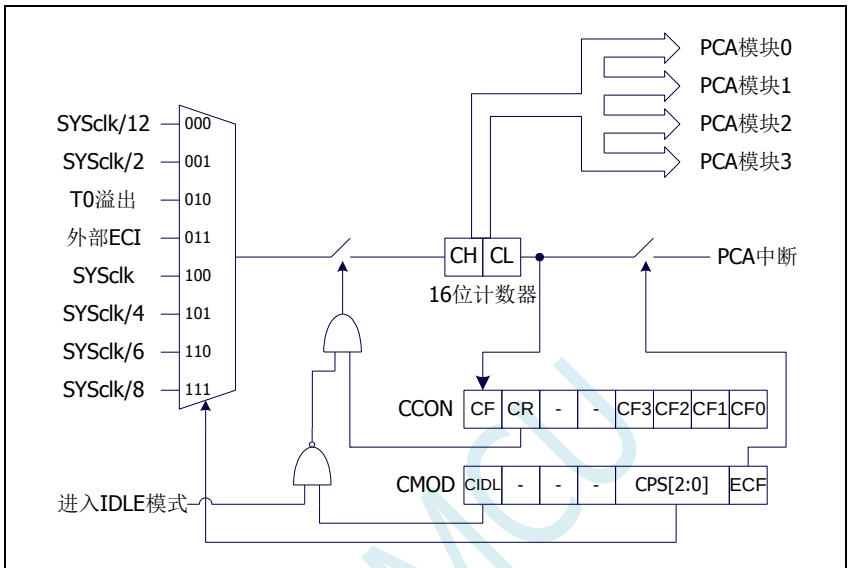
上面 2 式可以合起来：　**-Vcc < Vin < Vcc**

STC MCU

18 PCA/CCP/PWM 应用

STC12H 系列单片机内部集成了 4 组可编程计数器阵列（PCA/CCP/PWM）模块，可用于软件定时器、外部脉冲捕获、高速脉冲输出和 PWM 脉宽调制输出。

PCA 内部含有一个特殊的 16 位计数器，4 组 PCA 模块均与之相连接。PCA 计数器的结构图如下：



PCA计数器结构图

18.1 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
CCAPM3	PCA 模块 3 模式控制寄存器	FD54H	-	ECOM3	CCAPP3	CCAPN3	MAT3	TOG3	PWM3	ECCF3	x000,0000
CL	PCA 计数器低字节	E9H									0000,0000
CCAP0L	PCA 模块 0 低字节	EAH									0000,0000
CCAP1L	PCA 模块 1 低字节	EBH									0000,0000
CCAP2L	PCA 模块 2 低字节	ECH									0000,0000
CCAP3L	PCA 模块 3 低字节	FD55H									0000,0000
PCA_PWM0	PCA0 的 PWM 模式寄存器	F2H	EBS0[1:0]		XCCAP0H[1:0]		XCCAP0L[1:0]		EPC0H	EPC0L	0000,0000
PCA_PWM1	PCA1 的 PWM 模式寄存器	F3H	EBS1[1:0]		XCCAP1H[1:0]		XCCAP1L[1:0]		EPC1H	EPC1L	0000,0000
PCA_PWM2	PCA2 的 PWM 模式寄存器	F4H	EBS2[1:0]		XCCAP2H[1:0]		XCCAP2L[1:0]		EPC2H	EPC2L	0000,0000
PCA_PWM3	PCA3 的 PWM 模式寄存器	FD57H	EBS3[1:0]		XCCAP3H[1:0]		XCCAP3L[1:0]		EPC3H	EPC3L	0000,0000
CH	PCA 计数器高字节	F9H									0000,0000

CCAP0H	PCA 模块 0 高字节	FAH		0000,0000
CCAP1H	PCA 模块 1 高字节	FBH		0000,0000
CCAP2H	PCA 模块 2 高字节	FCH		0000,0000
CCAP3H	PCA 模块 3 高字节	FD56H		0000,0000

18.1.1 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=0,1,2,3）：PCA 模块中断标志。当 PCA 模块发生匹配或者捕获时，硬件自动将此位置 1，并向 CPU 提出中断请求。此标志位需要软件清零。

18.1.2 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
000	系统时钟/12
001	系统时钟/2
010	定时器 0 的溢出脉冲
011	ECI 脚的外部输入时钟
100	系统时钟
101	系统时钟/4
110	系统时钟/6
111	系统时钟 8

ECF：PCA 计数器溢出中断允许位。

0：禁止 PCA 计数器溢出中断

1：使能 PCA 计数器溢出中断

18.1.3 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。

18.1.4 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
CCAPM3	FD54H	-	ECOM3	CCAPP3	CCAPN3	MAT3	TOG3	PWM3	ECCF3

ECOMn：允许 PCA 模块 n 的比较功能
CCAPPn：允许 PCA 模块 n 进行上升沿捕获
CCAPNn：允许 PCA 模块 n 进行下降沿捕获
MATn：允许 PCA 模块 n 的匹配功能
TOGn：允许 PCA 模块 n 的高速脉冲输出功能
PWMn：允许 PCA 模块 n 的脉宽调制输出功能
ECCFn：允许 PCA 模块 n 的匹配/捕获中断

18.1.5 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								
CCAP3L	FD55H								
CCAP0H	FAH								
CCAP1H	FBH								
CCAP2H	FCH								
CCAP3H	FD56H								

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。

18.1.6 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
PCA_PWM3	FD57H	EBS3[1:0]		XCCAP3H[1:0]		XCCAP3L[1:0]		EPC3H	EPC3L

EBSn[1:0]：PCA 模块 n 的 PWM 位数控制

EBSn[1:0]	PWM 位数	重载值	比较值
00	8 位 PWM	{EPCnH, CCAPnH[7:0]}	{EPCnL, CCAPnL[7:0]}
01	7 位 PWM	{EPCnH, CCAPnH[6:0]}	{EPCnL, CCAPnL[6:0]}
10	6 位 PWM	{EPCnH, CCAPnH[5:0]}	{EPCnL, CCAPnL[5:0]}

11	10 位 PWM	{EPCnH, XCCAPnH[1:0], CCAPnH[7:0]}	{EPCnL, XCCAPnL[1:0], CCAPnL[7:0]}

XCCAPnH[1:0]：10 位 PWM 的第 9 位和第 10 位的重载值

XCCAPnL[1:0]：10 位 PWM 的第 9 位和第 10 位的比较值

EPCnH：PWM 模式下，重载值的最高位（8 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]。

STC MCU

18.2 PCA 工作模式

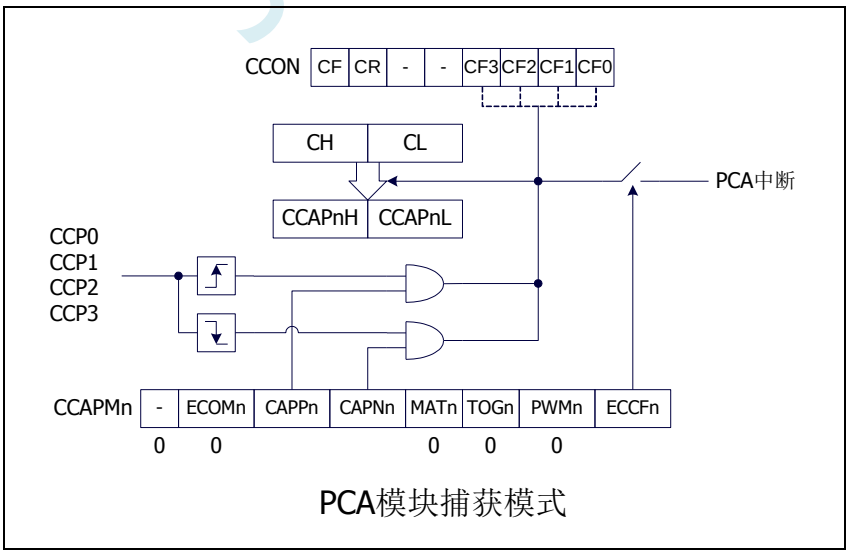
STC12H 系列单片机共有 4 组 PCA 模块，每组模块都可独立设置工作模式。模式设置如下所示：

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

18.2.1 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：



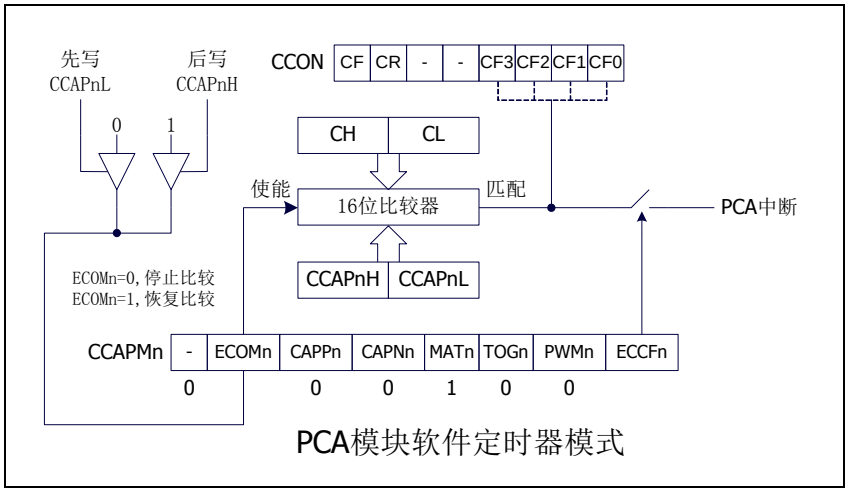
PCA模块捕获模式

18.2.2 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。

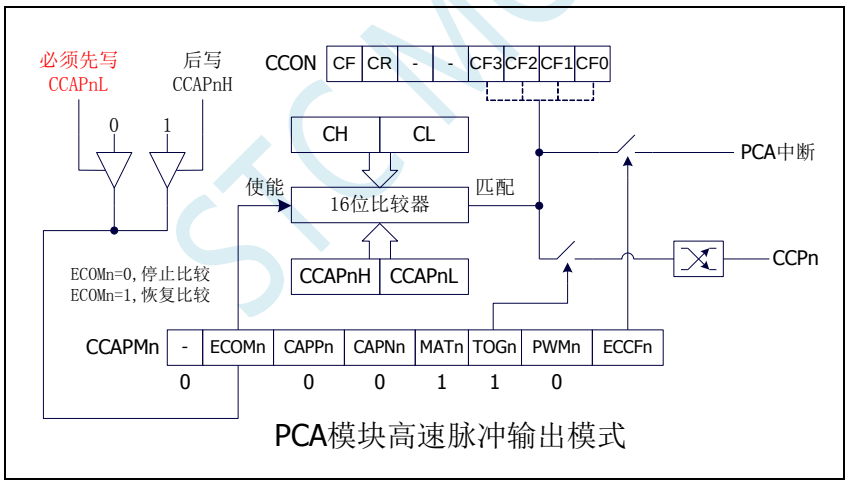
PCA 模块工作于软件定时器模式的结构图如下图所示：



18.2.3 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：



18.2.4 PWM 脉宽调制模式及频率计算公式

18.2.4.1 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。

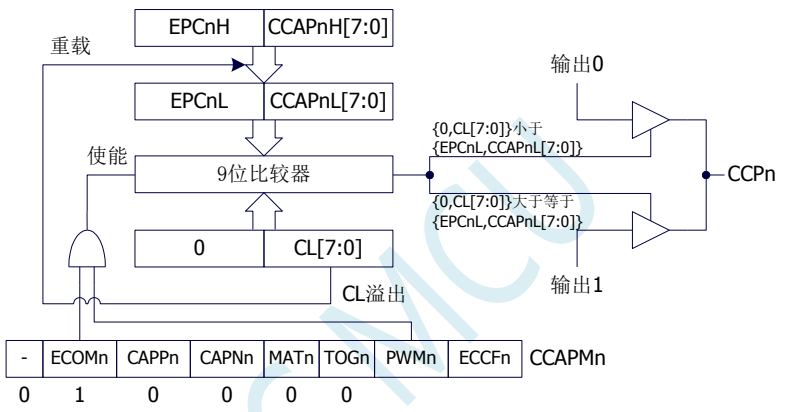
PCA_PWMn 寄存器中的 EBSn[1:0]设置为 00 时，PCA 模块 n 工作于 8 位 PWM 模式，此时将{0,CL[7:0]}与捕获寄存器{EPCnL,CCAPnL[7:0]}进行比较。当 PCA 模块工作于 8 位 PWM 模式时，由于所有模块共

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。

$$8\text{位模式的PWM频率} = \frac{\text{PCA时钟输入源频率}}{256}$$

当EPCnH=0及CCAPnH=00H时，PWM固定输出高
当EPCnH=1及CCAPnH=FFH时，PWM固定输出低

PCA 模块工作于 8 位 PWM 模式的结构图如下图所示：



PCA模块8位PWM模式

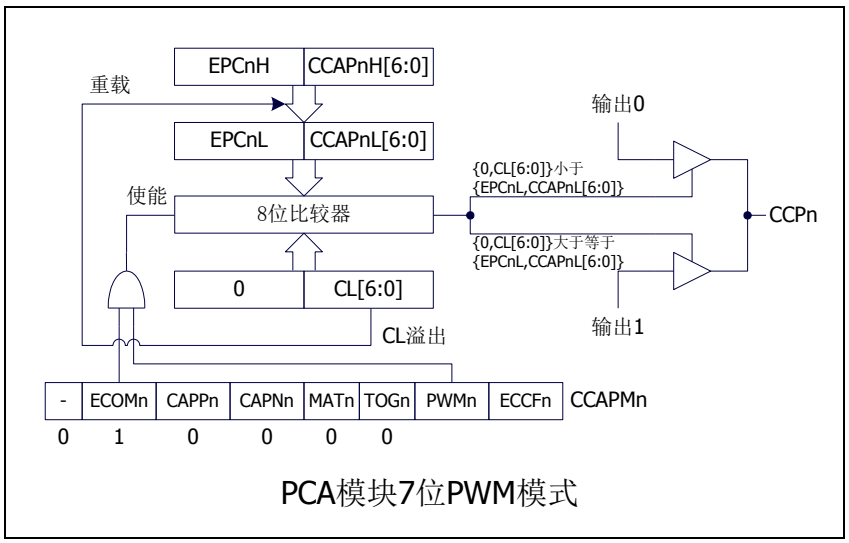
18.2.4.2　　7 位 PWM 模式

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。

$$7\text{位模式的PWM频率} = \frac{\text{PCA时钟输入源频率}}{128}$$

当EPCnH=0及CCAPnH=00H时，PWM固定输出高
当EPCnH=1及CCAPnH=FFH时，PWM固定输出低

PCA 模块工作于 7 位 PWM 模式的结构图如下图所示：



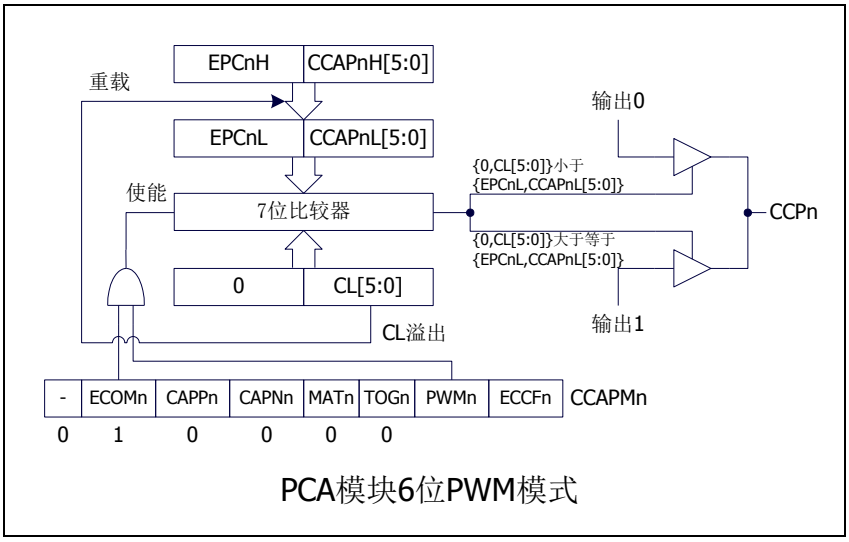
18.2.4.3 6 位 PWM 模式

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。

$$8\text{位模式的PWM频率} = \frac{\text{PCA时钟输入源频率}}{64}$$

当EPCnH=0及CCAPnH=00H时，PWM固定输出高
当EPCnH=1及CCAPnH=FFH时，PWM固定输出低

PCA 模块工作于 6 位 PWM 模式的结构图如下图所示：



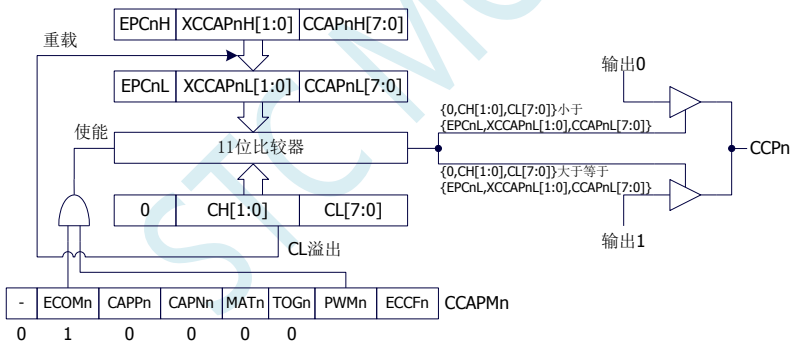
18.2.4.4 10 位 PWM 模式

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。

$$10\text{位模式的PWM频率} = \frac{\text{PCA时钟输入源频率}}{1024}$$

当EPCnH=0,XCCAP0H=0及CCAPnH=00H时，PWM固定输出高
当EPCnH=1,XCCAP0H=3及CCAPnH=FFH时，PWM固定输出低

PCA 模块工作于 10 位 PWM 模式的结构图如下图所示：

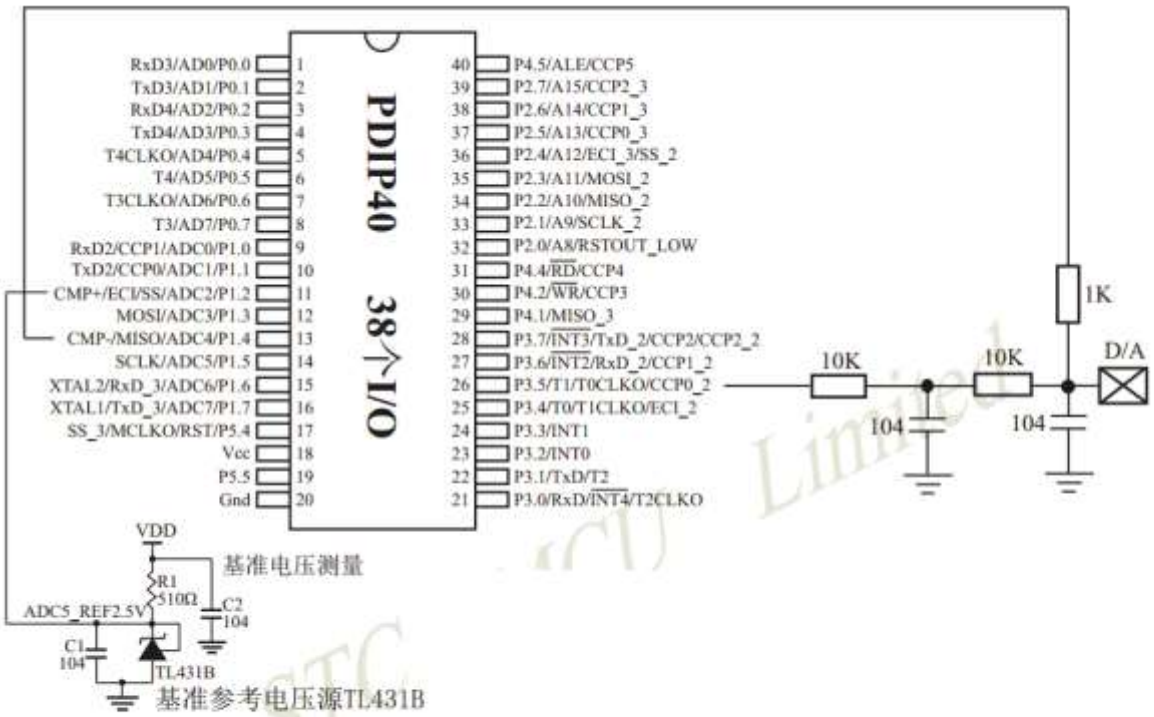


PCA模块10位PWM模式

18.2.4.5 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

18.3 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Vcc比较即可。

利用CCP/PCA模块的高速脉冲输出功能实现9~16位PWM来实现9~16位DAC，或用本身的硬件8位PWM来实现8位DAC，单片机本身也有10位ADC。

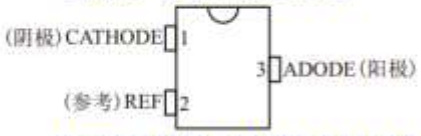
提示：

（1）PWM频率越高，输出波形越平滑。

（2）如果工作电压为5V，需输出1V电压，则设置高电平为1/5，低电平为4/5，则PWM输出电压就为1V。

（3）如果要输出高精准电压，建议用A/D检测输出的电压值，然后根据A/D检测的电压值逐步调整到所需要的电压。

基准参考电压源TL431B



SOT23-3封装，RMB￥0.15~0.3

基准参考电压源TL431B的符号

如应用简单，可无需基准参考电压源，直接与Vcc比较即可。

18.4 范例程序

18.4.1 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;
```

STC MCU

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

    CCON = 0x00;
    CMOD = 0x08;                    //PCA 时钟为系统时钟
    CL = 0x00;
    CH = 0x00;
//——6 位 PWM——
    CCAPM0 = 0x42;                  //PCA 模块 0 为 PWM 工作模式
    PCA_PWM0 = 0x80;                //PCA 模块 0 输出 6 位 PWM
    CCAP0L = 0x20;                  //PWM 占空比为 50%[(40H-20H)/40H]
    CCAP0H = 0x20;
//——7 位 PWM——
    CCAPM1 = 0x42;                  //PCA 模块 1 为 PWM 工作模式
    PCA_PWM1 = 0x40;                //PCA 模块 1 输出 7 位 PWM
    CCAP1L = 0x20;                  //PWM 占空比为 75%[(80H-20H)/80H]
    CCAP1H = 0x20;
//——8 位 PWM——
//  CCAPM2 = 0x42;                  //PCA 模块 2 为 PWM 工作模式
//  PCA_PWM2 = 0x00;                //PCA 模块 2 输出 8 位 PWM
//  CCAP2L = 0x20;                  //PWM 占空比为 87.5%[(100H-20H)/100H]
//  CCAP2H = 0x20;
//——10 位 PWM——
    CCAPM2 = 0x42;                  //PCA 模块 2 为 PWM 工作模式
    PCA_PWM2 = 0xc0;                //PCA 模块 2 输出 10 位 PWM
    CCAP2L = 0x20;                  //PWM 占空比为 96.875%[(400H-20H)/400H]
    CCAP2H = 0x20;
    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
CCAPM1      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
```

18.4.2 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     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;

unsigned char       cnt;                    //存储PCA 计时溢出次数
unsigned long       count0;                 //记录上一次的捕获值
unsigned long       count1;                 //记录本次的捕获值
unsigned long       length;                 //存储信号的时间长度

void PCA_Isr() interrupt 7
{
    if (CF)
```

```
        {
            CF = 0;
            cnt++;                                  //PCA 计时溢出次数+1
        }
        if (CCF0)
        {
            CCF0 = 0;
            count0 = count1;                        //备份上一次的捕获值
            ((unsigned char *)&count1)[3] = CCAP0L;
            ((unsigned char *)&count1)[2] = CCAP0H;
            ((unsigned char *)&count1)[1] = cnt;
            ((unsigned char *)&count1)[0] = 0;
            length = count1 - count0;               //length 保存的即为捕获的脉冲宽度
        }
}

void main()
{
        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
```

```
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
CCAP1H      DATA    0FBH
PCA_PWM1    DATA    0F3H
CCAPM2      DATA    0DCH
CCAP2L      DATA    0ECH
CCAP2H      DATA    0FCH
PCA_PWM2    DATA    0F4H

CNT         DATA    20H
COUNT0      DATA    21H             ;3 bytes
COUNT1      DATA    24H             ;3 bytes
LENGTH      DATA    27H             ;3 bytes, (COUNT1-COUNT0)

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
            JNB     CF,CHECKCCF0
            CLR     CF              ;清中断标志
            INC     CNT             ; PCA 计时溢出次数+1
CHECKCCF0:
            JNB     CCF0,ISREXIT
            CLR     CCF0
            MOV     COUNT0,COUNT1   ;备份上一次的捕获值
            MOV     COUNT0+1,COUNT1+1
            MOV     COUNT0+2,COUNT1+2
            MOV     COUNT1,CNT      ;保存本次的捕获值
            MOV     COUNT1+1,CCAP0H
            MOV     COUNT1+2,CCAP0L
            CLR     C               ;计算两次的捕获差值
            MOV     A,COUNT1+2
            SUBB    A,COUNT0+2
```

STC MCU

```
            MOV     LENGTH+2,A
            MOV     A,COUNT1+1
            SUBB    A,COUNT0+1
            MOV     LENGTH+1,A
            MOV     A,COUNT1
            SUBB    A,COUNT0
            MOV     LENGTH,A            ;LENGTH 保存的即为捕获的脉冲宽度
ISREXIT:
            POP     PSW
            POP     ACC
            RETI

MAIN:
            MOV     SP, #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     A
            MOV     CNT,A               ;用户变量初始化
            MOV     COUNT0,A
            MOV     COUNT0+1,A
            MOV     COUNT0+2,A
            MOV     COUNT1,A
            MOV     COUNT1+1,A
            MOV     COUNT1+2,A
            MOV     LENGTH,A
            MOV     LENGTH+1,A
            MOV     LENGTH+2,A

            MOV     CCON,#00H
            MOV     CMOD,#09H           ;PCA 时钟为系统时钟,使能 PCA 计时中断
            MOV     CL,#00H
            MOV     CH,#0H
            MOV     CCAPM0,#11H         ;PCA 模块 0 为 16 位捕获模式（下降沿捕获）
;           MOV     CCAPM0,#21H         ;PCA 模块 0 为 16 位捕获模式（上升沿捕获）
;           MOV     CCAPM0,#31H         ;PCA 模块 0 为 16 位捕获模式（边沿捕获）
            MOV     CCAP0L,#00H
            MOV     CCAP0H,#00H
            SETB    CR                  ;启动 PCA 计时器
            SETB    EA

            JMP     $

            END
```

18.4.3 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         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;

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 = 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
CCAP1L      DATA    0EBH
CCAP1H      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,#00H       ;PCA 时钟为系统时钟/12
            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
```

18.4.4 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       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;

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
CCAPM1      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

T38K4HZ     EQU     90H             ;11059200/2/38400

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 T38K4HZ
            MOV     CCAP0L,A
            MOV     A,CCAP0H
            ADDC    A,#HIGH T38K4HZ
            MOV     CCAP0H,A
            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,#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
```

18.4.5 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     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
{
    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
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         003BH
            LJMP        PCAISR

            ORG         0100H
PCAISR:
            CLR         CCF0
            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         CCON,#00H
            MOV         CMOD,#08H           ;PCA 时钟为系统时钟
            MOV         CL,#00H
            MOV         CH,#0H
            MOV         CCAPM0,#11H         ;扩展外部端口 CCP0 为下降沿中断口
;           MOV         CCAPM0,#21H         ;扩展外部端口 CCP0 为上升沿中断口
;           MOV         CCAPM0,#31H         ;扩展外部端口 CCP0 为边沿中断口
            MOV         CCAP0L,#0
            MOV         CCAP0H,#0
            SETB        CR                  ;启动 PCA 计时器
            SETB        EA

            JMP         $

            END
```

19 同步串行外设接口 SPI

STC12H 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。STC12H 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	SYSclk/4
01	SYSclk/8
10	SYSclk/16
11	SYSclk/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。

STC MCU

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）

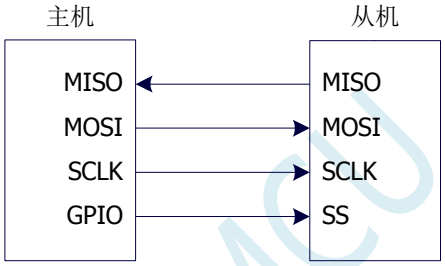
19.2.1 单主单从

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：



单主单从配置

19.2.2 互为主从

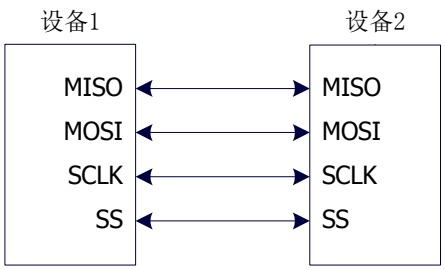
两个设备相连，主机和从机不固定。

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：

STC MCU



互为主从配置

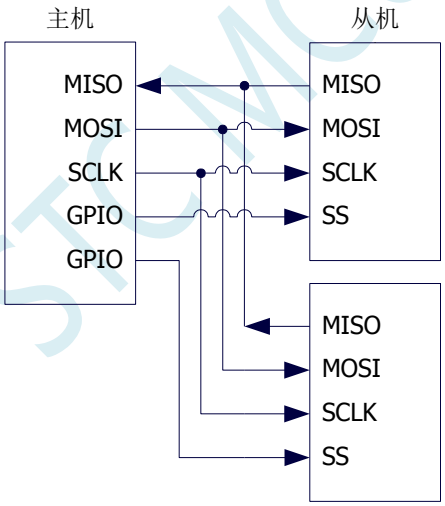
19.2.3 单主多从

多个设备相连，其中一个设备固定作为主机，其他设备固定作为从机。

主机设置：SSIG 设置为 1，MSTR 设置为 1，固定为主机模式。主机可以使用任意端口分别连接各个从机的 SS 管脚，拉低其中一个从机的 SS 脚即可使能相应的从机设备

从机设置：SSIG 设置为 0，SS 管脚作为从机的片选信号。

单主多从连接配置图如下所示：



单主多从配置

19.3 配置 SPI

控制位			通信端口				说明
SPEN	SSIG	MSTR	SS	MISO	MOSI	SCLK	
0	x	x	x	输入	输入	输入	关闭 SPI 功能，SS/MOSI/MISO/SCLK 均为普通 IO
1	0	0	0	输出	输入	输入	**从机模式**，且被选中
1	0	0	1	高阻	输入	输入	**从机模式**，但未被选中
1	0	1→0	0	输出	输入	输入	**从机模式**，不忽略 SS 且 MSTR 为 1 的主机模式，当 SS 管脚被拉低时，MSTR 将被硬件自动清零，工作模式将被被动设置为从机模式
1	0	1	1	输入	高阻	高阻	**主机模式**，空闲状态
					输出	输出	**主机模式**，激活状态
1	1	0	x	输出	输入	输入	**从机模式**
1	1	1	x	输入	输出	输出	**主机模式**

从机模式的注意事项：

当 CPHA＝0 时，SSIG 必须为 0（即不能忽略 SS 脚）。在每次串行字节开始还发送前 SS 脚必须拉低，并且在串行字节发送完后须重新设置为高电平。SS 管脚为低电平时不能对 SPDAT 寄存器执行写操作，否则将导致一个写冲突错误。CPHA=0 且 SSIG=1 时的操作未定义。

当 CPHA＝1 时，SSIG 可以置 1（即可以忽略脚）。如果 SSIG＝0，SS 脚可在连续传输之间保持低有效（即一直固定为低电平）。这种方式适用于固定单主单从的系统。

主机模式的注意事项：

在 SPI 中，传输总是由主机启动的。如果 SPI 使能（SPEN=1）并选择作为主机时，主机对 SPI 数据寄存器 SPDAT 的写操作将启动 SPI 时钟发生器和数据的传输。在数据写入 SPDAT 之后的半个到一个 SPI 位时间后，数据将出现在 MOSI 脚。写入主机 SPDAT 寄存器的数据从 MOSI 脚移出发送到从机的 MOSI 脚。同时从机 SPDAT 寄存器的数据从 MISO 脚移出发送到主机的 MISO 脚。

传输完一个字节后，SPI 时钟发生器停止，传输完成标志（SPIF）置位，如果 SPI 中断使能则会产生一个 SPI 中断。主机和从机 CPU 的两个移位寄存器可以看作是一个 16 位循环移位寄存器。当数据从主机移位传送到从机的同时，数据也以相反的方向移入。这意味着在一个移位周期中，主机和从机的数据相互交换。

通过 SS 改变模式

如果 SPEN=1，SSIG=0 且 MSTR=1，SPI 使能为主机模式，并将 SS 脚可配置为输入模式化或准双向口模式。这种情况下，另外一个主机可将该脚驱动为低电平，从而将该器件选择为 SPI 从机并向其发送数据。为了避免争夺总线，SPI 系统将该从机的 MSTR 清零，MOSI 和 SCLK 强制变为输入模式，而 MISO 则变为输出模式，同时 SPSTAT 的 SPIF 标志位置 1。

用户软件必须一直对 MSTR 位进行检测，如果该位被一个从机选择动作而被动清零，而用户想继续将 SPI 作为主机，则必须重新设置 MSTR 位，否则将一直处于从机模式。

写冲突

SPI 在发送时为单缓冲，在接收时为双缓冲。这样在前一次发送尚未完成之前，不能将新的数据写入移位寄存器。当发送过程中对数据寄存器 SPDAT 进行写操作时，WCOL 位将被置 1 以指示发生数据写冲突错误。在这种情况下，当前发送的数据继续发送，而新写入的数据将丢失。

当对主机或从机进行写冲突检测时，主机发生写冲突的情况是很罕见的，因为主机拥有数据传输的完全控制权。但从机有可能发生写冲突，因为当主机启动传输时，从机无法进行控制。

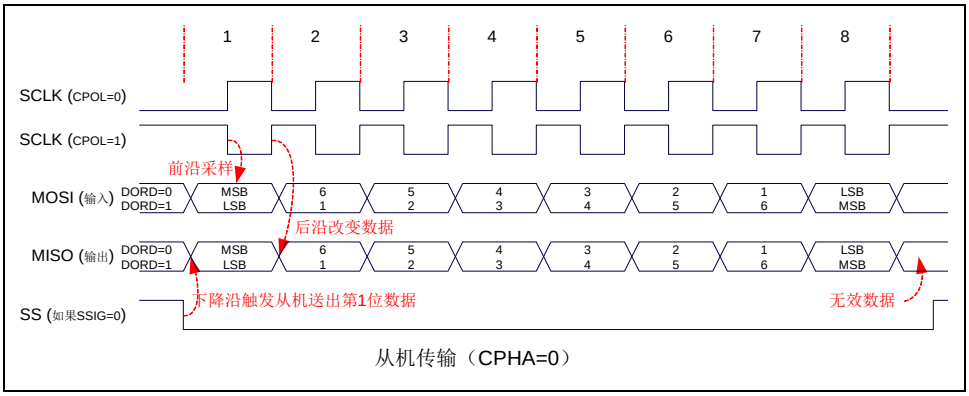
接收数据时，接收到的数据传送到一个并行读数据缓冲区，这样将释放移位寄存器以进行下一个数据的接收。但必须在下个字符完全移入之前从数据寄存器中读出接收到的数据，否则，前一个接收数据将丢失。

WCOL 可通过软件向其写入“1”清零。

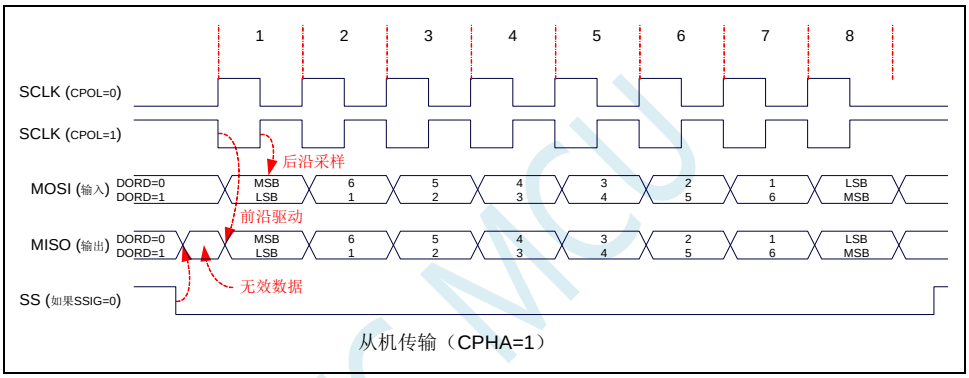
STC MCU

19.4 数据模式

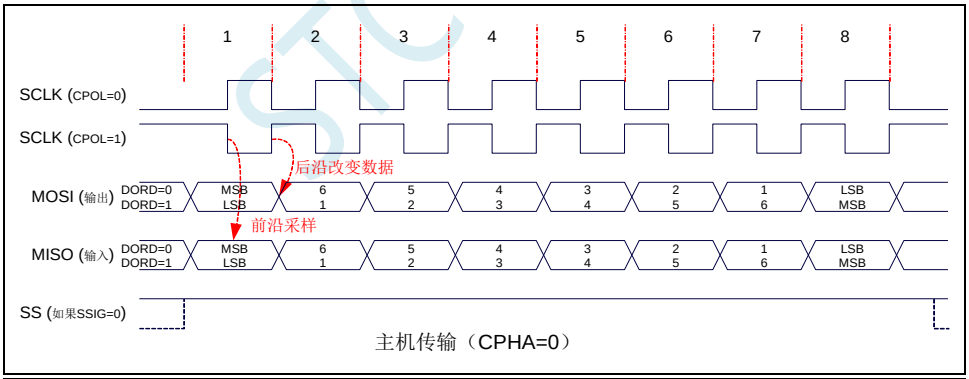
SPI 的时钟相位控制位 CPHA 可以让用户设定数据采样和改变时的时钟沿。时钟极性位 CPOL 可以让用户设定时钟极性。下面图例显示了不同时钟相位、极性设置下 SPI 通讯时序。



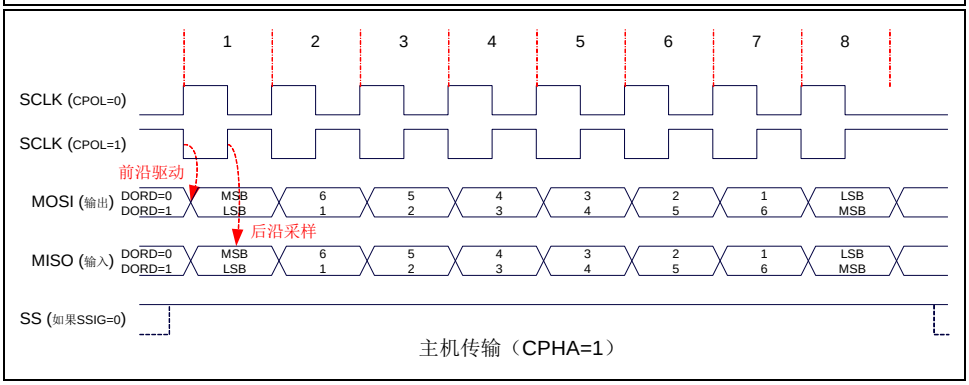
从机传输（CPHA=0）



从机传输（CPHA=1）



主机传输（CPHA=0）



主机传输（CPHA=1）

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     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    SS          =   P1^0;
sbit    LED         =   P1^1;

bit     busy;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                          //清中断标志
    SS = 1;                                 //拉高从机的 SS 管脚
    busy = 0;
    LED = !LED;                             //测试端口
}

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

STC MCU

```
    P5M1 = 0x00;

    LED = 1;
    SS = 1;
    busy = 0;

    SPCTL = 0x50;                          //使能 SPI 主机模式
    SPSTAT = 0xc0;                         //清中断标志
    IE2 = ESPI;                            //使能 SPI 中断
    EA = 1;

    while (1)
    {
        while (busy);
        busy = 1;
        SS = 0;                            //拉低从机 SS 管脚
        SPDAT = 0x5a;                      //发送测试数据
    }
}
```

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

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     004BH
            LJMP    SPIISR

            ORG     0100H
SPIISR:
            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     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    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

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         004BH
            LJMP        SPIISR

            ORG         0100H
SPIISR:
            MOV         SPSTAT,#0C0H            ;清中断标志
            MOV         SPDAT,SPDAT             ;将接收到的数据回传给主机
            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

            MOV         SPCTL,#40H              ;使能 SPI 从机模式
            MOV         SPSTAT,#0C0H            ;清中断标志
            MOV         IE2,#ESPI               ;使能 SPI 中断
            SETB        EA

            JMP         $

            END
```

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     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     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
```

```
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

            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     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    LED         =   P1^1;

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

SPSTAT      DATA        0CDH
SPCTL       DATA        0CEH
SPDAT       DATA        0CFH
IE2         DATA        0AFH
ESPI        EQU         02H

LED         BIT         P1.1

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         SPCTL,#40H          ;使能SPI从机模式
            MOV         SPSTAT,#0C0H        ;清中断标志

LOOP:
            MOV         A,SPSTAT            ;查询完成标志
            JNB         ACC.7,$-2
            MOV         SPSTAT,#0C0H        ;清中断标志
            MOV         SPDAT,SPDAT         ;将接收到的数据回传给主机
            CPL         LED
            JMP         LOOP

            END
```

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     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    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)
    {                                   //主机模式
        SS = 1;                         //拉高从机的 SS 管脚
        SPCTL = 0x40;                   //重新设置为从机待机
    }
    else
    {                                   //从机模式
        SPDAT = SPDAT;                  //将接收到的数据回传给主机
    }
    LED = !LED;                         //测试端口
}

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

STC MCU

```
    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
    IE2 = ESPI;                            //使能 SPI 中断
    EA = 1;

    while (1)
    {
        if (!KEY)                          //等待按键触发
        {
            SPCTL = 0x50;                  //使能 SPI 主机模式
            SS = 0;                        //拉低从机 SS 管脚
            SPDAT = 0x5a;                  //发送测试数据
            while (!KEY);                  //等待按键释放
        }
    }
}
```

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
KEY         BIT         P0.0

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         004BH
            LJMP        SPIISR

            ORG         0100H
SPIISR:
```

```
        PUSH    ACC
        MOV     SPSTAT,#0C0H        ;清中断标志
        MOV     A,SPCTL
        JB      ACC.4,MASTER
SLAVE:
        MOV     SPDAT,SPDAT         ;将接收到的数据回传给主机
        JMP     ISREXIT
MASTER:
        SETB    SS                  ;拉高从机的 SS 管脚
        MOV     SPCTL,#40H          ;重新设置为从机待机
ISREXIT:
        CPL     LED
        POP     ACC
        RETI

MAIN:
        MOV     SP, #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     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    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)
            {                                   //主机模式
                SS = 1;                         //拉高从机的SS 管脚
                SPCTL = 0x40;                   //重新设置为从机待机
            }
            else
            {                                   //从机模式
                SPDAT = SPDAT;                  //将接收到的数据回传给主机
            }
            LED = !LED;                         //测试端口
        }
    }
}
```

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
KEY         BIT         P0.0

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
```

STC MCU

```
        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

LOOP:
        JB      KEY,SKIP            ;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
SKIP:
        MOV     A,SPSTAT
        JNB     ACC.7,LOOP
        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
        CPL     LED
        JMP     LOOP
MASTER:
        SETB    SS                  ;拉高从机的 SS 管脚
        MOV     SPCTL,#40H          ;重新设置为从机待机
        CPL     LED
        JMP     LOOP

        END
```

STC MCU

20 I²C 总线

STC12H 系列的单片机内部集成了一个 I²C 串行总线控制器。I²C 是一种高速同步通讯总线，通讯使用 SCL（时钟线）和 SDA（数据线）两线进行同步通讯。对于 SCL 和 SDA 的端口分配，STC8 系列的单片机提供了切换模式，可将 SCL 和 SDA 切换到不同的 I/O 口上，以方便用户将一组 I²C 总线当作多组进行分时复用。

与标准 I²C 协议相比较，忽略了如下两种机制：

- 发送起始信号（START）后不进行仲裁
- 时钟信号（SCL）停留在低电平时不进行超时检测

STC12H 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	I²C 配置寄存器	FE80H	ENI2C	MSSL	MSSPEED[5:0]						0000,0000
I2CMSCR	I²C 主机控制寄存器	FE81H	EMSI	-	-	-	MSCMD[3:0]				0xxx,0000
I2CMSST	I²C 主机状态寄存器	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO	00xx,xx00
I2CSLCR	I²C 从机控制寄存器	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0
I2CSLST	I²C 从机状态寄存器	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000
I2CSLADR	I²C 从机地址寄存器	FE85H	I2CSLADR[7:1]							MA	0000,0000
I2CTXD	I²C 数据发送寄存器	FE86H									0000,0000
I2CRXD	I²C 数据接收寄存器	FE87H									0000,0000
I2CMSAUX	I²C 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＝F_{OSC} / 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）

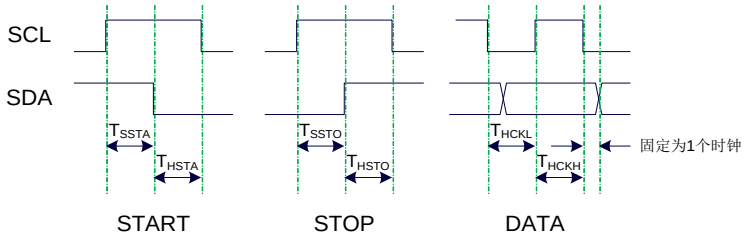
T_{HSTA}：起始信号的保持时间（Hold Time of START）

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，

MSSPEED＝(24M / 400K / 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

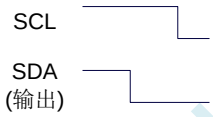
1：允许主机模式的中断

MSCMD[3:0]：主机命令

0000：待机，无动作。

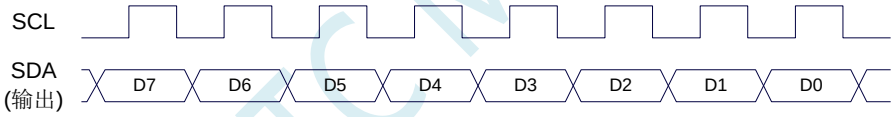
0001：起始命令。

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：



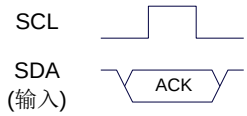
0010：发送数据命令。

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：



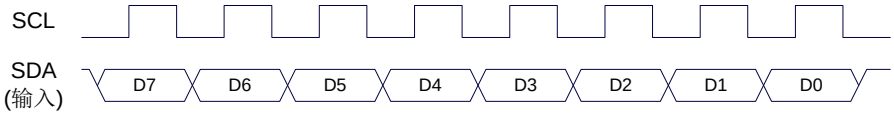
0011：接收 ACK 命令。

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：



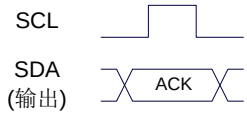
0100：接收数据命令。

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：



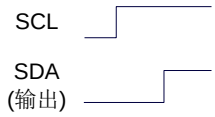
0101：发送 ACK 命令。

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。

STC MCU

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

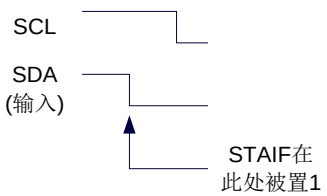
SLBUSY：从机模式时 I²C 控制器状态位（只读位）

0：控制器处于空闲状态

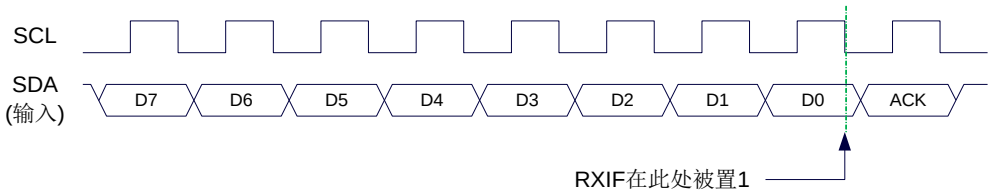
1：控制器处于忙碌状态

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。

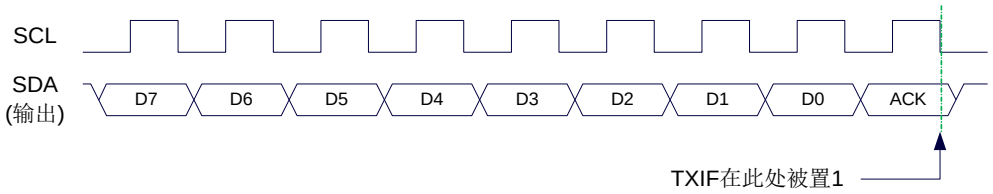
STAIF：从机模式时接收到 START 信号后的中断请求位。从机模式的 I²C 控制器接收到 START 信号后，硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 STAIF 位必须用软件清零。STAIF 被置 1 的时间点如下图所示：



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：



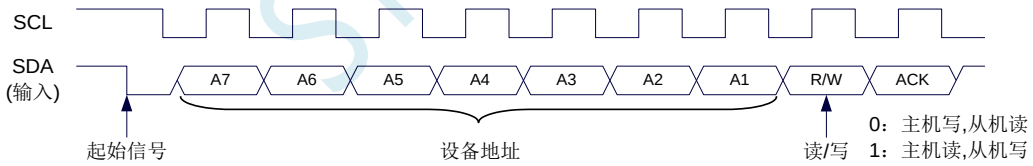
TXIF：从机模式时发送完成 1 字节的数据后的中断请求位。从机模式的 I²C 控制器发送完成 1 字节的数据并成功接收到 1 位 ACK 信号后，在第 9 个时钟的下降沿时硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 TXIF 位必须用软件清零。TXIF 被置 1 的时间点如下图所示：



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

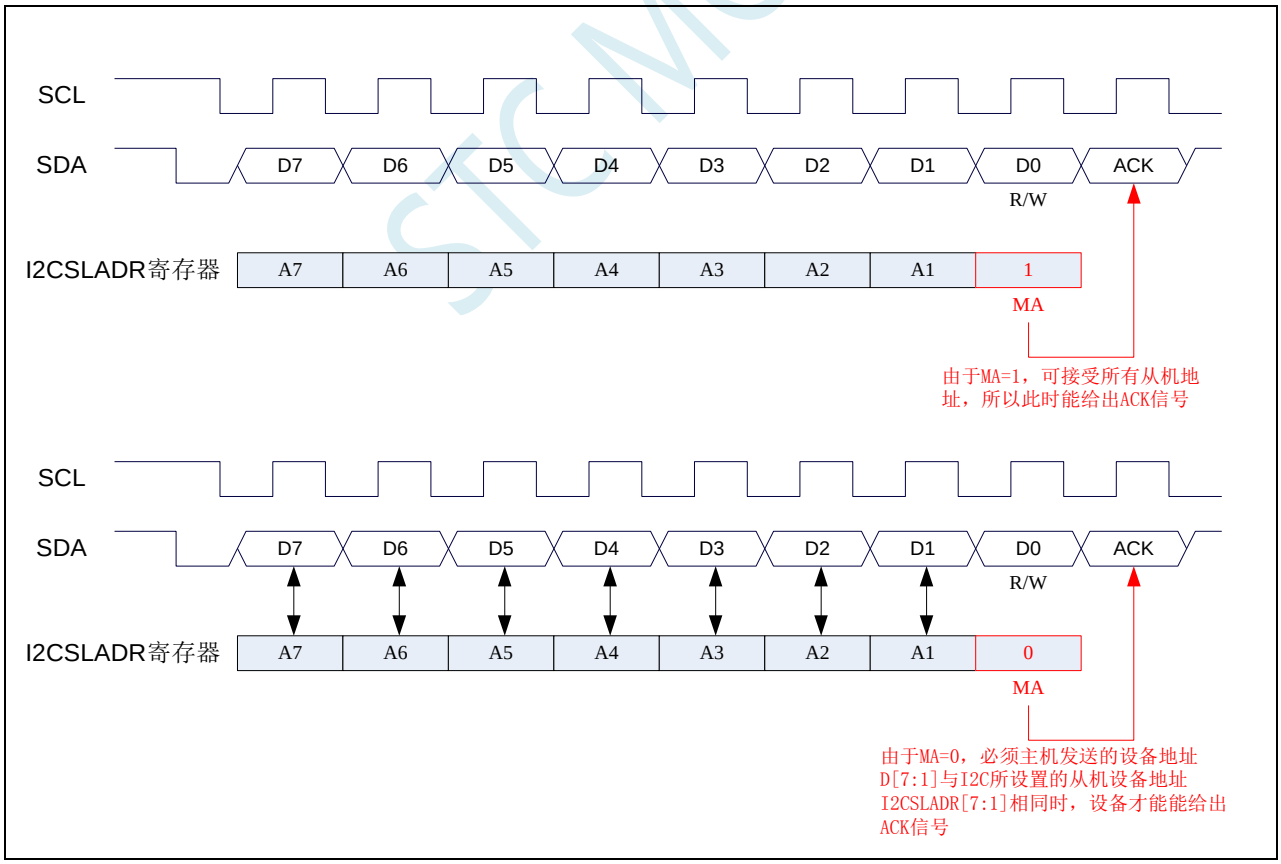
当 I^2C 控制器处于从机模式时，控制器在接收到 START 信号后，会继续检测接下来主机发送出的设备地址数据以及读/写信号。当主机发送出的设备地址与 I2CSLADR[7:1]中所设置的从机设备地址相同时，控制器才会向 CPU 发出中断求，请求 CPU 处理 I^2C 事件；否则若设备地址不同，I^2C 控制器继续监控，等待下一个起始信号，对下一个设备地址继续比较。

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 I^2C 发送数据寄存器，存放将要发送的 I^2C 数据

I2CRXD 是 I^2C 接收数据寄存器，存放接收完成的 I^2C 数据

STC MCU

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     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     busy;

void I2C_Isr() interrupt 24
{
    _push_(P_SW2);
    P_SW2 |= 0x80;
    if (I2CMSST & 0x40)
    {
        I2CMSST &= ~0x40;                   //清中断标志
        busy = 0;
    }
    _pop_(P_SW2);
}

void Start()
{
    busy = 1;
    I2CMSCR = 0x81;                         //发送 START 命令
```

```
        while (busy);
}

void SendData(char dat)
{
        I2CTXD = dat;                                   //写数据到数据缓冲区
        busy = 1;
        I2CMSCR = 0x82;                                 //发送 SEND 命令
        while (busy);
}

void RecvACK()
{
        busy = 1;
        I2CMSCR = 0x83;                                 //发送读 ACK 命令
        while (busy);
}

char RecvData()
{
        busy = 1;
        I2CMSCR = 0x84;                                 //发送 RECV 命令
        while (busy);
        return I2CRXD;
}

void SendACK()
{
        I2CMSST = 0x00;                                 //设置 ACK 信号
        busy = 1;
        I2CMSCR = 0x85;                                 //发送 ACK 命令
        while (busy);
}

void SendNAK()
{
        I2CMSST = 0x01;                                 //设置 NAK 信号
        busy = 1;
        I2CMSCR = 0x85;                                 //发送 ACK 命令
        while (busy);
}

void Stop()
{
        busy = 1;
        I2CMSCR = 0x86;                                 //发送 STOP 命令
        while (busy);
}

void Delay()
{
        int i;

        for (i=0; i<3000; i++)
        {
                _nop_();
                _nop_();
                _nop_();
                _nop_();
```

STC MCU

```
    }
}

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

P_SW2       DATA        0BAH

I2CCFG      XDATA       0FE80H
I2CMSCR     XDATA       0FE81H
I2CMSST     XDATA       0FE82H
I2CSLCR     XDATA       0FE83H
I2CSLST     XDATA       0FE84H
I2CSLADR    XDATA       0FE85H
I2CTXD      XDATA       0FE86H
I2CRXD      XDATA       0FE87H

SDA         BIT         P1.4
SCL         BIT         P1.5

BUSY        BIT         20H.0

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         00C3H
            LJMP        I2CISR

            ORG         0100H
I2CISR:
            PUSH        ACC
            PUSH        DPL
            PUSH        DPH

            MOV         DPTR,#I2CMSST       ;清中断标志
            MOVX        A,@DPTR
            ANL         A,#NOT 40H
            MOV         DPTR,#I2CMSST
            MOVX        @DPTR,A
            CLR         BUSY                ;复位忙标志

            POP         DPH
            POP         DPL
            POP         ACC
            RETI

START:
            SETB        BUSY
```

```
            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     R1,#0
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,#11100000B        ;设置 I2C 模块为主机模式
        MOV     DPTR,#I2CCFG
        MOVX    @DPTR,A
        MOV     A,#00000000B
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A
        SETB    EA

        CALL    START               ;发送起始命令
        MOV     A,#0A0H
        CALL    SENDDATA            ;发送设备地址+写命令
        CALL    RECVACK
        MOV     A,#000H             ;发送存储地址高字节
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#000H             ;发送存储地址低字节
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#12H              ;写测试数据 1
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#78H              ;写测试数据 2
        CALL    SENDDATA
        CALL    RECVACK
        CALL    STOP                ;发送停止命令

        CALL    DELAY               ;等待设备写数据

        CALL    START               ;发送起始命令
        MOV     A,#0A0H             ;发送设备地址+写命令
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#000H             ;发送存储地址高字节
        CALL    SENDDATA
```

```
        CALL    RECVACK
        MOV     A,#000H         ;发送存储地址低字节
        CALL    SENDDATA
        CALL    RECVACK
        CALL    START           ;发送起始命令
        MOV     A,#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     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;
```

```
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_();
```

STC MCU

```
            _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

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

SDA         BIT     P1.4
SCL         BIT     P1.5

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
START:
            MOV     A,#00000001B        ;发送 START 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
SENDDATA:
            MOV     DPTR,#I2CTXD        ;写数据到数据缓冲区
            MOVX    @DPTR,A
            MOV     A,#00000010B        ;发送 SEND 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
RECVACK:
            MOV     A,#00000011B        ;发送读 ACK 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
RECVDATA:
            MOV     A,#00000100B        ;发送 RECV 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
```

```
            CALL    WAIT
            MOV     DPTR,#I2CRXD        ;从数据缓冲区读取数据
            MOVX    A,@DPTR
            RET
SENDACK:
            MOV     A,#00000000B        ;设置 ACK 信号
            MOV     DPTR,#I2CMSST
            MOVX    @DPTR,A
            MOV     A,#00000101B        ;发送 ACK 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
SENDNAK:
            MOV     A,#00000001B        ;设置 NAK 信号
            MOV     DPTR,#I2CMSST
            MOVX    @DPTR,A
            MOV     A,#00000101B        ;发送 ACK 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
STOP:
            MOV     A,#00000110B        ;发送 STOP 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
WAIT:
            MOV     DPTR,#I2CMSST       ;清中断标志
            MOVX    A,@DPTR
            JNB     ACC.6,WAIT
            ANL     A,#NOT 40H
            MOVX    @DPTR,A
            RET

DELAY:
            MOV     R0,#0
            MOV     R1,#0
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,#11100000B        ;设置 I2C 模块为主机模式
MOV     DPTR,#I2CCFG
MOVX    @DPTR,A
MOV     A,#00000000B
MOV     DPTR,#I2CMSST
MOVX    @DPTR,A

CALL    START               ;发送起始命令
MOV     A,#0A0H
CALL    SENDDATA            ;发送设备地址+写命令
CALL    RECVACK
MOV     A,#000H             ;发送存储地址高字节
CALL    SENDDATA
CALL    RECVACK
MOV     A,#000H             ;发送存储地址低字节
CALL    SENDDATA
CALL    RECVACK
MOV     A,#12H              ;写测试数据 1
CALL    SENDDATA
CALL    RECVACK
MOV     A,#78H              ;写测试数据 2
CALL    SENDDATA
CALL    RECVACK
CALL    STOP                ;发送停止命令

CALL    DELAY               ;等待设备写数据

CALL    START               ;发送起始命令
MOV     A,#0A0H             ;发送设备地址+写命令
CALL    SENDDATA
CALL    RECVACK
MOV     A,#000H             ;发送存储地址高字节
CALL    SENDDATA
CALL    RECVACK
MOV     A,#000H             ;发送存储地址低字节
CALL    SENDDATA
CALL    RECVACK
CALL    START               ;发送起始命令
MOV     A,#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
```

20.4.3 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     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;

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;
```

STC MCU

```
    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

P_SW2       DATA        0BAH

I2CCFG      XDATA       0FE80H
I2CMSCR     XDATA       0FE81H
I2CMSST     XDATA       0FE82H
I2CSLCR     XDATA       0FE83H
I2CSLST     XDATA       0FE84H
I2CSLADR    XDATA       0FE85H
I2CTXD      XDATA       0FE86H
I2CRXD      XDATA       0FE87H

SDA         BIT         P1.4
SCL         BIT         P1.5

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
START:
            MOV         A,#00000001B        ;发送 START 命令
            MOV         DPTR,#I2CMSCR
            MOVX        @DPTR,A
            JMP         WAIT
SENDDATA:
            MOV         DPTR,#I2CTXD        ;写数据到数据缓冲区
            MOVX        @DPTR,A
            MOV         A,#00000010B        ;发送 SEND 命令
            MOV         DPTR,#I2CMSCR
            MOVX        @DPTR,A
            JMP         WAIT
RECVACK:
            MOV         A,#00000011B        ;发送读 ACK 命令
            MOV         DPTR,#I2CMSCR
            MOVX        @DPTR,A
            JMP         WAIT
RECVDATA:
            MOV         A,#00000100B        ;发送 RECV 命令
            MOV         DPTR,#I2CMSCR
            MOVX        @DPTR,A
            CALL        WAIT
            MOV         DPTR,#I2CRXD        ;从数据缓冲区读取数据
            MOVX        A,@DPTR
            RET
SENDACK:
            MOV         A,#00000000B        ;设置 ACK 信号
            MOV         DPTR,#I2CMSST
            MOVX        @DPTR,A
            MOV         A,#00000101B        ;发送 ACK 命令
            MOV         DPTR,#I2CMSCR
            MOVX        @DPTR,A
            JMP         WAIT
SENDNAK:
            MOV         A,#00000001B        ;设置 NAK 信号
            MOV         DPTR,#I2CMSST
            MOVX        @DPTR,A
            MOV         A,#00000101B        ;发送 ACK 命令
            MOV         DPTR,#I2CMSCR
            MOVX        @DPTR,A
            JMP         WAIT
STOP:
            MOV         A,#00000110B        ;发送 STOP 命令
            MOV         DPTR,#I2CMSCR
```

```
        MOVX    @DPTR,A
        JMP     WAIT
WAIT:
        MOV     DPTR,#I2CMSST           ;清中断标志
        MOVX    A,@DPTR
        JNB     ACC.6,WAIT
        ANL     A,#NOT 40H
        MOVX    @DPTR,A
        RET

DELAY:
        MOV     R0,#0
        MOV     R1,#0
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,#11100000B            ;设置 I2C 模块为主机模式
        MOV     DPTR,#I2CCFG
        MOVX    @DPTR,A
        MOV     A,#00000000B
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A

        CALL    START                   ;发送起始命令
        MOV     A,#0A2H
        CALL    SENDDATA                ;发送设备地址+写命令
        CALL    RECVACK
        MOV     A,#002H                 ;发送存储地址
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#00H                  ;设置秒值
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#00H                  ;设置分钟值
        CALL    SENDDATA
        CALL    RECVACK
```

```
            MOV     A,#12H          ;设置小时值
            CALL    SENDDATA
            CALL    RECVACK
            CALL    STOP            ;发送停止命令
LOOP:
            CALL    START           ;发送起始命令
            MOV     A,#0A2H         ;发送设备地址+写命令
            CALL    SENDDATA
            CALL    RECVACK
            MOV     A,#002H         ;发送存储地址
            CALL    SENDDATA
            CALL    RECVACK
            CALL    START           ;发送起始命令
            MOV     A,#0A3H         ;发送设备地址+读命令
            CALL    SENDDATA
            CALL    RECVACK
            CALL    RECVDATA        ;读取秒值
            MOV     P0,A
            CALL    SENDACK
            CALL    RECVDATA        ;读取分钟值
            MOV     P2,A
            CALL    SENDACK
            CALL    RECVDATA        ;读取小时值
            MOV     P3,A
            CALL    SENDNAK
            CALL    STOP            ;发送停止命令

            CALL    DELAY

            JMP     LOOP

            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     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;                                   //设备地址标志
bit     isma;                                   //存储地址标志
unsigned char       addr;
unsigned char pdata     buffer[256];

void I2C_Isr() interrupt 24
{
    _push_(P_SW2);
    P_SW2 |= 0x80;

    if (I2CSLST & 0x40)
    {
        I2CSLST &= ~0x40;                       //处理 START 事件
        isda = 1;                               //若为重复起始信号时必须作此设置
    }
    else if (I2CSLST & 0x20)
    {
        I2CSLST &= ~0x20;                       //处理 RECV 事件
        if (isda)
        {
            isda = 0;                           //处理 RECV 事件（RECV DEVICE ADDR）
        }
        else if (isma)
        {
            isma = 0;                           //处理 RECV 事件（RECV MEMORY ADDR）
            addr = I2CRXD;
            I2CTXD = buffer[addr];
        }
        else
        {
            buffer[addr++] = I2CRXD;            //处理 RECV 事件（RECV DATA）
        }
    }
    else if (I2CSLST & 0x10)
    {
        I2CSLST &= ~0x10;                       //处理 SEND 事件
        if (I2CSLST & 0x02)
        {
            I2CTXD = 0xff;                      //接收到 NAK 则停止读取数据
        }
        else
        {
            I2CTXD = buffer[++addr];            //接收到 ACK 则继续读取数据
        }
    }
    else if (I2CSLST & 0x08)
```

```
    {
        I2CSLST &= ~0x08;                    //处理 STOP 事件
        isda = 1;
        isma = 1;
    }

    _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
```

```
SDA         BIT         P1.4
SCL         BIT         P1.5
ISDA        BIT         20H.0                   ;设备地址标志
ISMA        BIT         20H.1                   ;存储地址标志

ADDR        DATA        21H

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         00C3H
            LJMP        I2CISR

            ORG         0100H
I2CISR:
            PUSH        ACC
            PUSH        PSW
            PUSH        DPL
            PUSH        DPH
            MOV         DPTR,#I2CSLST           ;检测从机状态
            MOVX        A,@DPTR
            JB          ACC.6,STARTIF
            JB          ACC.5,RXIF
            JB          ACC.4,TXIF
            JB          ACC.3,STOPIF
ISREXIT:
            POP         DPH
            POP         DPL
            POP         PSW
            POP         ACC
            RETI
STARTIF:
            ANL         A,#NOT 40H              ;处理 START 事件
            MOVX        @DPTR,A
            SETB        ISDA
            JMP         ISREXIT
RXIF:
            ANL         A,#NOT 20H              ;处理 RECV 事件
            MOVX        @DPTR,A
            MOV         DPTR,#I2CRXD
            MOVX        A,@DPTR
            JBC         ISDA,RXDA
            JBC         ISMA,RXMA
            MOV         R0,ADDR                 ;处理 RECV 事件（RECV DATA）
            MOVX        @R0,A
            INC         ADDR
```

```
        JMP     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.1,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,#01111000B            ;使能从机模式中断
        MOV     DPTR,#I2CSLCR
        MOVX    @DPTR,A

        SETB    ISDA                    ;用户变量初始化
        SETB    ISMA
        CLR     A
        MOV     ADDR,A
        MOV     R0,A
        MOVX    A,@R0
        MOV     DPTR,#I2CTXD
        MOVX    @DPTR,A

        SETB    EA

        SJMP    $

        END
```

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;                                   //设备地址标志
bit         isma;                                   //存储地址标志
unsigned char           addr;
unsigned char pdata     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)
        {
            I2CSLST &= ~0x40;                       //处理 START 事件
            isda = 1;                               //若为重复起始信号时必须作此设置
        }
        else if (I2CSLST & 0x20)
        {
            I2CSLST &= ~0x20;                       //处理 RECV 事件
            if (isda)
            {
                isda = 0;                           //处理 RECV 事件（RECV DEVICE ADDR）
            }
            else if (isma)
            {
                isma = 0;                           //处理 RECV 事件（RECV MEMORY ADDR）
                addr = I2CRXD;
                I2CTXD = buffer[addr];
```

```
                }
                else
                {
                        buffer[addr++] = I2CRXD;                //处理 RECV 事件（RECV DATA）
                }
        }
        else if (I2CSLST & 0x10)
        {
                I2CSLST &= ~0x10;                               //处理 SEND 事件
                if (I2CSLST & 0x02)
                {
                        I2CTXD = 0xff;                          //接收到 NAK 则停止读取数据
                }
                else
                {
                        I2CTXD = buffer[++addr];                //接收到 ACK 则继续读取数据
                }
        }
        else if (I2CSLST & 0x08)
        {
                I2CSLST &= ~0x08;                               //处理 STOP 事件
                isda = 1;
                isma = 1;
        }
    }
}
```

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

SDA         BIT         P1.4
SCL         BIT         P1.5
ISDA        BIT         20H.0                   ;设备地址标志
ISMA        BIT         20H.1                   ;存储地址标志

ADDR        DATA        21H

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         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,#00000000B            ;禁止从机模式中断
            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

LOOP:
            MOV         DPTR,#I2CSLST           ;检测从机状态
            MOVX        A,@DPTR
            JB          ACC.6,STARTIF
            JB          ACC.5,RXIF
            JB          ACC.4,TXIF
            JB          ACC.3,STOPIF
            JMP         LOOP
```

```
STARTIF:
        ANL     A,#NOT 40H      ;处理 START 事件
        MOVX    @DPTR,A
        SETB    ISDA
        JMP     LOOP
RXIF:
        ANL     A,#NOT 20H      ;处理 RECV 事件
        MOVX    @DPTR,A
        MOV     DPTR,#I2CRXD
        MOVX    A,@DPTR
        JBC     ISDA,RXDA
        JBC     ISMA,RXMA
        MOV     R0,ADDR         ;处理 RECV 事件（RECV DATA）
        MOVX    @R0,A
        INC     ADDR
        JMP     LOOP
RXDA:
        JMP     LOOP            ;处理 RECV 事件（RECV DEVICE ADDR）
RXMA:
        MOV     ADDR,A          ;处理 RECV 事件（RECV MEMORY ADDR）
        MOV     R0,A
        MOVX    A,@R0
        MOV     DPTR,#I2CTXD
        MOVX    @DPTR,A
        JMP     LOOP
TXIF:
        ANL     A,#NOT 10H      ;处理 SEND 事件
        MOVX    @DPTR,A
        JB      ACC.1,RXNAK
        INC     ADDR
        MOV     R0,ADDR
        MOVX    A,@R0
        MOV     DPTR,#I2CTXD
        MOVX    @DPTR,A
        JMP     LOOP
RXNAK:
        MOVX    A,#0FFH
        MOV     DPTR,#I2CTXD
        MOVX    @DPTR,A
        JMP     LOOP
STOPIF:
        ANL     A,#NOT 08H      ;处理 STOP 事件
        MOVX    @DPTR,A
        SETB    ISDA
        SETB    ISMA
        JMP     LOOP

        END
```

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     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;

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();
    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

P_SW2       DATA        0BAH

I2CCFG      XDATA       0FE80H
I2CMSCR     XDATA       0FE81H
I2CMSST     XDATA       0FE82H
I2CSLCR     XDATA       0FE83H
I2CSLST     XDATA       0FE84H
I2CSLADR    XDATA       0FE85H
I2CTXD      XDATA       0FE86H
I2CRXD      XDATA       0FE87H

SDA         BIT         P1.4
SCL         BIT         P1.5

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
START:
        MOV     A,#00000001B        ;发送 START 命令
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
SENDDATA:
        MOV     DPTR,#I2CTXD        ;写数据到数据缓冲区
        MOVX    @DPTR,A
        MOV     A,#00000010B        ;发送 SEND 命令
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
RECVACK:
        MOV     A,#00000011B        ;发送读 ACK 命令
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
RECVDATA:
        MOV     A,#00000100B        ;发送 RECV 命令
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        CALL    WAIT
        MOV     DPTR,#I2CRXD        ;从数据缓冲区读取数据
        MOVX    A,@DPTR
        RET
SENDACK:
        MOV     A,#00000000B        ;设置 ACK 信号
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A
        MOV     A,#00000101B        ;发送 ACK 命令
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
SENDNAK:
        MOV     A,#00000001B        ;设置 NAK 信号
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A
        MOV     A,#00000101B        ;发送 ACK 命令
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
STOP:
        MOV     A,#00000110B        ;发送 STOP 命令
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
WAIT:
        MOV     DPTR,#I2CMSST       ;清中断标志
        MOVX    A,@DPTR
        JNB     ACC.6,WAIT
        ANL     A,#NOT 40H
        MOVX    @DPTR,A
        RET

DELAY:
        MOV     R0,#0
        MOV     R1,#0
```

```
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,#11100000B        ;设置 I2C 模块为主机模式
        MOV     DPTR,#I2CCFG
        MOVX    @DPTR,A
        MOV     A,#00000000B
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A

        CALL    START               ;发送起始命令
        MOV     A,#5AH              ;
        CALL    SENDDATA            ;发送设备地址(010_1101B)+写命令(0B)
        CALL    RECVACK
        MOV     A,#000H             ;发送存储地址
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#12H              ;写测试数据 1
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#78H              ;写测试数据 2
        CALL    SENDDATA
        CALL    RECVACK
        CALL    STOP                ;发送停止命令

        CALL    DELAY               ;等待设备写数据

        CALL    START               ;发送起始命令
        MOV     A,#5AH              ;发送设备地址(010_1101B)+写命令(0B)
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#000H             ;发送存储地址
        CALL    SENDDATA
        CALL    RECVACK
        CALL    START               ;发送起始命令
        MOV     A,#5BH              ;发送设备地址(010_1101B)+读命令(1B)
```

```
        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
```

STC MCU

21 16 位高级 PWM 定时器，支持正交编码器

STC12H 系列的单片机内部集成了 8 通道 16 位高级 PWM 定时器，分成两组周期可不同的 PWM，分别命名为 PWMA 和 PWMB(之前的数据手册曾命名为 PWM1 和 PWM2，但容易与芯片管脚名称混淆，故更改为 PWMA 和 PWMB)，可分别单独设置。第一组 PWM/PWMA 可配置成 4 组互补/对称/死区控制的 PWM 或捕捉外部信号，第二组 PWM/PWMB 可配置成 4 路 PWM 输出或捕捉外部信号。

第一组 PWM/PWMA 的时钟频率可以是系统时钟经过寄存器 PWMA_PSCRH 和 PWMA_PSCRL 进行分频后的时钟，分频值可以是 1~65535 之间的任意值。第二组 PWM/PWMB 的时钟频率可以是系统时钟经过寄存器 PWMB_PSCRH 和 PWMB_PSCRL 进行分频后的时钟，分频值可以是 1~65535 之间的任意值。两组 PWM 的时钟频率可分别独立设置。

第一组 PWM 定时器/PWMA 有 4 个通道（PWM1P/PWM1N、PWM2P/PWM2N、PWM3P/PWM3N、PWM4P/PWM4N），每个通道都可独立实现 PWM 输出（可设置带死区的互补对称 PWM 输出）、捕获和比较功能；第二组 PWM 定时器/PWMB 有 4 个通道（PWM5、PWM6、PWM7、PWM8），每个通道也可独立实现 PWM 输出、捕获和比较功能。两组 PWM 定时器唯一的区别是第一组可输出带死区的互补对称 PWM，而第二组只能输出单端的 PWM，其他功能完全相同。下面关于高级 PWM 定时器的介绍只以第一组为例进行说明。

当使用第一组 PWM 定时器输出 PWM 波形时，可单独使能 PWM1P/PWM2P/PWM3P/PWM4P 输出，也可单独使能 PWM1N/PWM2N/PWM3N/PWM4N 输出。例如：若单独使能了 PWM1P 输出，则 PWM1N 就不能再独立输出，除非 PWM1P 和 PWM1N 组成一组互补对称输出。PWMA 的 4 路输出是可分别独立设置的，例如：可单独使能 PWM1P 和 PWM2N 输出，也可单独使能 PWM2N 和 PWM3N 输出。若需要使用第一组 PWM 定时器进行捕获功能或者测量脉宽时，输入信号只能从每路的正端输入，即只有 PWM1P/PWM2P/PWM3P/PWM4P 才有捕获功能和测量脉宽功能。

两组高级 PWM 定时器对外部信号进行捕获时，可选择上升沿捕获或者下降沿捕获。如果需要同时捕获上升沿和下降沿，则可将输入信号同时接入到两路 PWM，使能其中一路捕获上升沿，另外一路捕获下降沿即可。**更强悍的是，将外部输入信号同时接入到两路 PWM 时，可同时捕获信号的周期值和占空比值**。

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/STC12H 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。

下面的说明中，PWMA 代表第一组 PWM 定时器，PWMB 代表第二组 PWM 定时器

第 1 组高级 PWM 定时器/PWMA 内部信号说明

TI1：外部时钟输入信号 1（PWM1P 管脚信号或者 PWM1P/PWM2P/PWM3P 相异或后的信号）
TI1F：经过 IC1F 数字滤波后的 TI1 信号
TI1FP：经过 CC1P/CC2P 边沿检测器后的 TI1F 信号
TI1F_ED：TI1F 的边沿信号
TI1FP1：经过 CC1P 边沿检测器后的 TI1F 信号
TI1FP2：经过 CC2P 边沿检测器后的 TI1F 信号
IC1：通过 CC1S 选择的通道 1 的捕获输入信号
OC1REF：输出通道 1 输出的参考波形（中间波形）
OC1：通道 1 的主输出信号（经过 CC1P 极性处理后的 OC1REF 信号）
OC1N：通道 1 的互补输出信号（经过 CC1NP 极性处理后的 OC1REF 信号）

TI2：外部时钟输入信号 2（PWM2P 管脚信号）
TI2F：经过 IC2F 数字滤波后的 TI2 信号
TI2F_ED：TI2F 的边沿信号
TI2FP：经过 CC1P/CC2P 边沿检测器后的 TI2F 信号
TI2FP1：经过 CC1P 边沿检测器后的 TI2F 信号
TI2FP2：经过 CC2P 边沿检测器后的 TI2F 信号
IC2：通过 CC2S 选择的通道 2 的捕获输入信号
OC2REF：输出通道 2 输出的参考波形（中间波形）
OC2：通道 2 的主输出信号（经过 CC2P 极性处理后的 OC2REF 信号）
OC2N：通道 2 的互补输出信号（经过 CC2NP 极性处理后的 OC2REF 信号）

TI3：外部时钟输入信号 3（PWM3P 管脚信号）
TI3F：经过 IC3F 数字滤波后的 TI3 信号
TI3F_ED：TI3F 的边沿信号
TI3FP：经过 CC3P/CC4P 边沿检测器后的 TI3F 信号
TI3FP3：经过 CC3P 边沿检测器后的 TI3F 信号
TI3FP4：经过 CC4P 边沿检测器后的 TI3F 信号
IC3：通过 CC3S 选择的通道 3 的捕获输入信号
OC3REF：输出通道 3 输出的参考波形（中间波形）
OC3：通道 3 的主输出信号（经过 CC3P 极性处理后的 OC3REF 信号）
OC3N：通道 3 的互补输出信号（经过 CC3NP 极性处理后的 OC3REF 信号）

TI4：外部时钟输入信号 4（PWM4P 管脚信号）
TI4F：经过 IC4F 数字滤波后的 TI4 信号
TI4F_ED：TI4F 的边沿信号
TI4FP：经过 CC3P/CC4P 边沿检测器后的 TI4F 信号
TI4FP3：经过 CC3P 边沿检测器后的 TI4F 信号
TI4FP4：经过 CC4P 边沿检测器后的 TI4F 信号
IC4：通过 CC4S 选择的通道 4 的捕获输入信号
OC4REF：输出通道 4 输出的参考波形（中间波形）
OC4：通道 4 的主输出信号（经过 CC4P 极性处理后的 OC4REF 信号）
OC4N：通道 4 的互补输出信号（经过 CC4NP 极性处理后的 OC4REF 信号）

ITR1：内部触发输入信号 1
ITR2：内部触发输入信号 2

TRC：固定为 TI1_ED
TRGI：经过 TS 多路选择器后的触发输入信号
TRGO：经过 MMS 多路选择器后的触发输出信号

ETR：外部触发输入信号（PWMETI1 管脚信号）
ETRP：经过 ETP 边沿检测器以及 ETPS 分频器后的 ETR 信号
ETRF：经过 ETF 数字滤波后的 ETRP 信号

BRK：刹车输入信号（PWMFLT）

CK_PSC：预分频时钟，PWMA_PSCR 预分频器的输入时钟
CK_CNT：PWMA_PSCR 预分频器的输出时钟，PWM 定时器的时钟

第 2 组高级 PWM 定时器/PWMB 内部信号说明

TI5：外部时钟输入信号 5（PWM5 管脚信号或者 PWM5/PWM6/PWM7 相异或后的信号）
TI5F：经过 IC5F 数字滤波后的 TI5 信号
TI5FP：经过 CC5P/CC6P 边沿检测器后的 TI5F 信号
TI5F_ED：TI5F 的边沿信号
TI5FP5：经过 CC5P 边沿检测器后的 TI5F 信号
TI5FP6：经过 CC6P 边沿检测器后的 TI5F 信号
IC5：通过 CC5S 选择的通道 5 的捕获输入信号
OC5REF：输出通道 5 输出的参考波形（中间波形）
OC5：通道 5 的主输出信号（经过 CC5P 极性处理后的 OC5REF 信号）

TI6：外部时钟输入信号 6（PWM6 管脚信号）
TI6F：经过 IC6F 数字滤波后的 TI6 信号
TI6F_ED：TI6F 的边沿信号
TI6FP：经过 CC5P/CC6P 边沿检测器后的 TI6F 信号
TI6FP5：经过 CC5P 边沿检测器后的 TI6F 信号
TI6FP6：经过 CC6P 边沿检测器后的 TI6F 信号
IC6：通过 CC6S 选择的通道 6 的捕获输入信号
OC6REF：输出通道 6 输出的参考波形（中间波形）
OC6：通道 6 的主输出信号（经过 CC6P 极性处理后的 OC6REF 信号）

TI7：外部时钟输入信号 7（PWM7 管脚信号）
TI7F：经过 IC7F 数字滤波后的 TI7 信号
TI7F_ED：TI7F 的边沿信号
TI7FP：经过 CC7P/CC8P 边沿检测器后的 TI7F 信号
TI7FP7：经过 CC7P 边沿检测器后的 TI7F 信号
TI7FP8：经过 CC8P 边沿检测器后的 TI7F 信号
IC7：通过 CC7S 选择的通道 7 的捕获输入信号
OC7REF：输出通道 7 输出的参考波形（中间波形）
OC7：通道 7 的主输出信号（经过 CC7P 极性处理后的 OC7REF 信号）

TI8：外部时钟输入信号 8（PWM8 管脚信号）
TI8F：经过 IC8F 数字滤波后的 TI8 信号
TI8F_ED：TI8F 的边沿信号
TI8FP：经过 CC7P/CC8P 边沿检测器后的 TI8F 信号
TI8FP7：经过 CC7P 边沿检测器后的 TI8F 信号
TI8FP8：经过 CC8P 边沿检测器后的 TI8F 信号
IC8：通过 CC8S 选择的通道 8 的捕获输入信号
OC8REF：输出通道 8 输出的参考波形（中间波形）
OC8：通道 8 的主输出信号（经过 CC8P 极性处理后的 OC8REF 信号）

21.1 简介

PWMA 由一个 16 位的自动装载计数器组成，它由一个可编程的预分频器驱动。

PWMA 适用于许多不同的用途：

- 基本的定时
- 测量输入信号的脉冲宽度（输入捕获）
- 产生输出波形（输出比较，PWM 和单脉冲模式）
- 对应与不同事件（捕获，比较，溢出，刹车，触发）的中断
- 与 PWMB 或者外部信号（外部时钟，复位信号，触发和使能信号）同步

PWMA 广泛的适用于各种控制应用中，包括那些需要中间对齐模式 PWM 的应用，该模式支持互补输出和死区时间控制。PWMA 的时钟源可以是内部时钟，也可以是外部的信号，可以通过配置寄存器来进行选择。

21.2 主要特性

PWMA 的特性包括：

- 16 位向上、向下、向上/下自动装载计数器
- 允许在指定数目的计数器周期之后更新定时器寄存器的重复计数器
- 16 位可编程（可以实时修改）预分频器，计数器时钟频率的分频系数为 1～65535 之间的任意数值
- 同步电路，用于使用外部信号控制定时器以及定时器互联
- 多达 4 个独立通道可以配置成：
  - 输入捕获
  - 输出比较
  - PWM 输出（边缘或中间对齐模式）
  - 六步 PWM 输出
  - 单脉冲模式输出
  - 支持 4 个死区时间可编程的通道上互补输出
- 刹车输入信号（PWMFLT）可以将定时器输出信号置于复位状态或者一个确定状态
- 外部触发输入引脚（PWMETI）
- 产生中断的事件包括：
  - 更新：计数器向上溢出/向下溢出，计数器初始化（通过软件或者内部/外部触发）
  - 触发事件（计数器启动、停止、初始化或者由内部/外部触发计数）
  - 输入捕获，测量脉宽

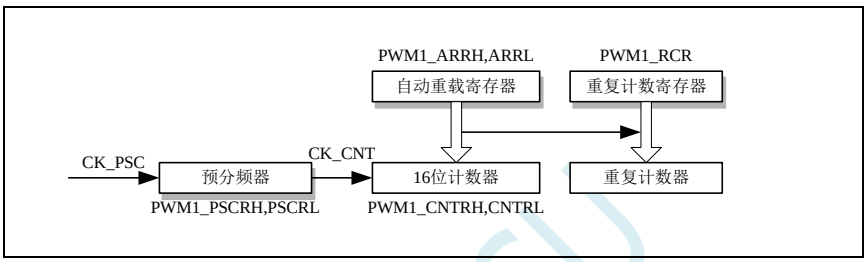
— 外部中断
— 输出比较
— 刹车信号输入

21.3 时基单元

PWMA 的时基单元包含：

- 16 位向上/向下计数器
- 16 位自动重载寄存器
- 重复计数器
- 预分频器

PWMA 时基单元



16 位计数器、预分频器、自动重载寄存器和重复计数器寄存器都可以通过软件进行读写操作。自动重载寄存器由预装载寄存器和影子寄存器组成。

可在在两种模式下写自动重载寄存器：

- 自动预装载已使能（PWMA_CR1 寄存器的 ARPE 位为 1）。在此模式下，写入自动重载寄存器的数据将被保存在预装载寄存器中，并在下一个更新事件（UEV）时传送到影子寄存器。
- 自动预装载已禁止（PWMA_CR1 寄存器的 ARPE 位为 0）。在此模式下，写入自动重载寄存器的数据将立即写入影子寄存器。

更新事件的产生条件：

- 计数器向上或向下溢出。
- 软件置位了 PWMA_EGR 寄存器的 UG 位。
- 时钟/触发控制器产生了触发事件。

在预装载使能时（ARPE=1），如果发生了更新事件，预装载寄存器中的数值（PWMA_ARR）将写入影子寄存器中，并且 PWMA_PSCR 寄存器中的值将写入预分频器中。置位 PWMA_CR1 寄存器的 UDIS 位将禁止更新事件（UEV）。预分频器的输出 CK_CNT 驱动计数器，而 CK_CNT 仅在 PWMA_CR1 寄存器的计数器使能位（CEN）被置位时才有效。

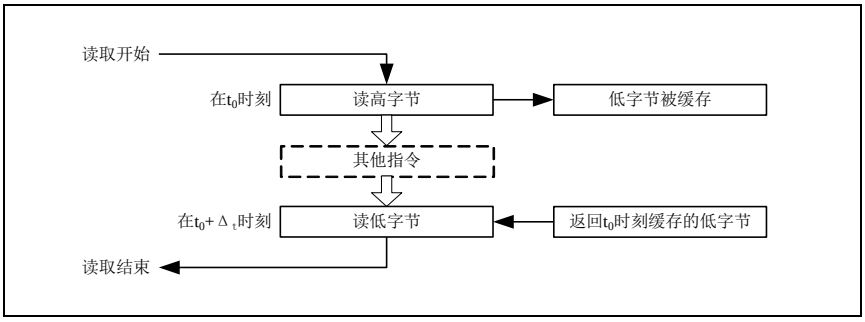
注意：实际的计数器在 CEN 位使能的一个时钟周期后才开始计数。

21.3.1 读写 16 位计数器

写计数器的操作没有缓存，在任何时候都可以写 PWMA_CNTRH 和 PWMA_CNTRL 寄存器，因此为避免写入了错误的数值，一般建议不要在计数器运行时写入新的数值。

读计数器的操作带有 8 位的缓存。用户必须先读定时器的高字节，在用户读了高字节后，低字节将

被自动缓存，缓存的数据将会一直保持直到 16 位数据的读操作完成。



21.3.2 16 位 PWMA_ARR 寄存器的写操作

预装载寄存器中的值将写入 16 位的 PWMA_ARR 寄存器中，此操作由两条指令完成，每条指令写入 1 个字节。必须先写高字节，后写低字节。

影子寄存器在写入高字节时被锁定，并保持到低字节写完。

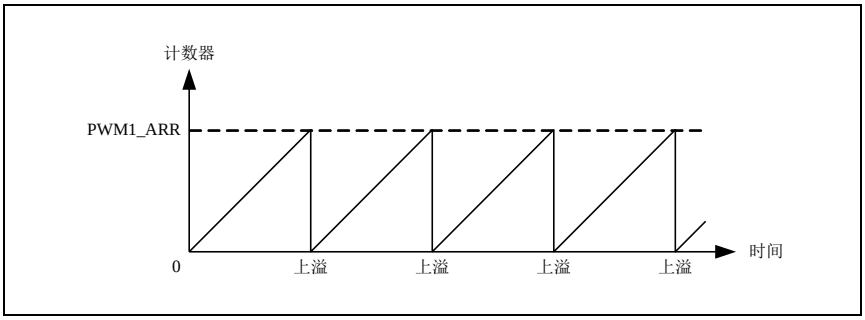
21.3.3 预分频器

PWMA 的预分频器基于一个由 16 位寄存器（PWMA_PSCR）控制的 16 位计数器。由于这个控制寄存器带有缓冲器，因此它能够在运行时被改变。预分频器可以将计数器的时钟频率按 1 到 65536 之间的任意值分频。预分频器的值由预装载寄存器写入，保存了当前使用值的影子寄存器在低字节写入时被载入。由于需两次单独的写操作来写 16 位寄存器，因此必须保证高字节先写入。新的预分频器的值在下一次更新事件到来时被采用。对 PWMA_PSCR 寄存器的读操作通过预装载寄存器完成。

计数器的频率计算公式：$f_{CK_CNT} = f_{CK_PSC} / (PSCR[15:0] + 1)$

21.3.4 向上计数模式

在向上计数模式中，计数器从 0 计数到用户定义的比较值（PWMA_ARR 寄存器的值），然后重新从 0 开始计数并产生一个计数器溢出事件，此时如果 PWMA_CR1 寄存器的 UDIS 位是 0，将会产生一个更新事件（UEV）。



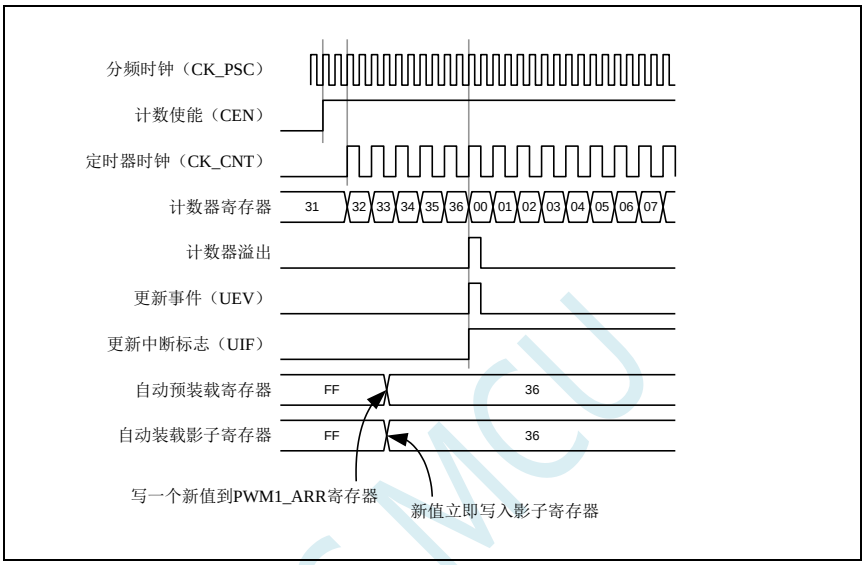
通过软件方式或者通过使用触发控制器置位 PWMA_EGR 寄存器的 UG 位同样也可以产生一个更新事件。使用软件置位 PWMA_CR1 寄存器的 UDIS 位，可以禁止更新事件，这样可以避免在更新预装载寄存器时更新影子寄存器。在 UDIS 位被清除之前，将不产生更新事件。但是在应该产生更新事件时，计数器仍会被清 0，同时预分频器的计数也被清 0（但预分频器的数值不变）。此外，如果设置了 PWMA_CR1 寄存器中的 URS 位（选择更新请求），设置 UG 位将产生一个更新事件 UEV，但硬件不设置 UIF 标志（即不产生中断请求）。这是为了避免在捕获模式下清除计数器时，同时产生更新和捕获中断。

当发生一个更新事件时，所有的寄存器都被更新，硬件依据 URS 位同时设置更新标志位（PWMA_SR 寄存器的 UIF 位）：

- 自动装载影子寄存器被重新置入预装载寄存器的值（PWMA_ARR）。
- 预分频器的缓存器被置入预装载寄存器的值（PWMA_PSC）。

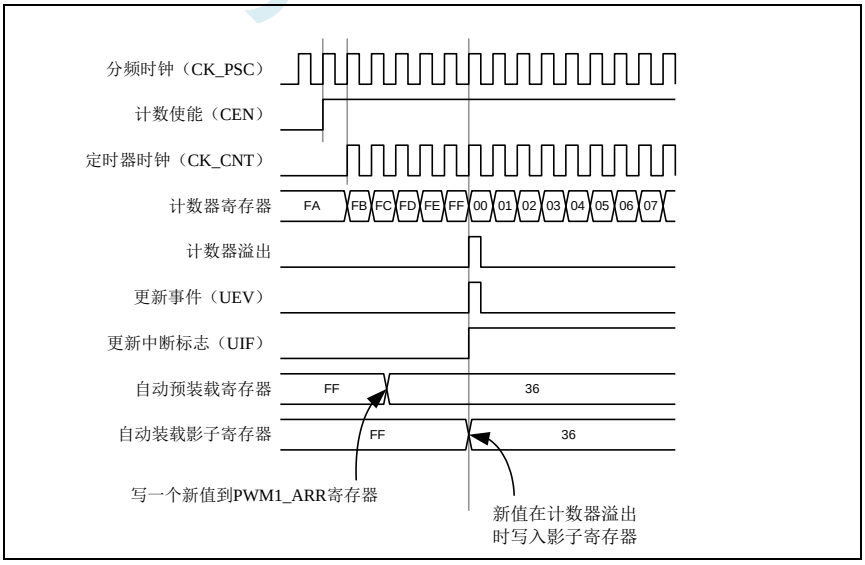
下图给出一些例子，说明当 PWMA_ARR=0x36 时，计数器在不同时钟频率下的动作。图中预分频为 2，因此计数器的时钟（CK_CNT）频率是预分频时钟（CK_PSC）频率的一半。图中禁止了自动装载功能（ARPE=0），所以在计数器达到 0x36 时，计数器溢出，影子寄存器立刻被更新，同时产生一个更新事件。

当 ARPE=0（ARR 不预装载），预分频为 2 时的计数器更新：



下图的预分频为 1，因此 CK_CNT 的频率与 CK_PSC 一致。图中使能了自动重载（ARPE=1），所以在计数器达到 0xFF 产生溢出。0x36 将在溢出时被写入，同时产生一个更新事件。

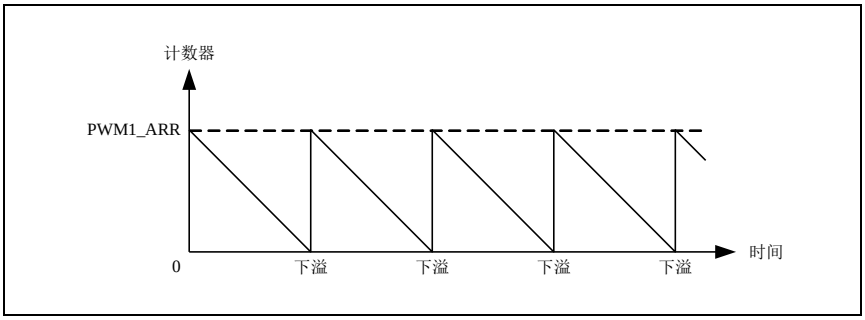
当 ARPE=1(PWMA_ARR 预装载)，预分频为 1 时的计数器更新：



21.3.5 向下计数模式

在向下模式中，计数器从自动装载的值（PWMA_ARR 寄存器的值）开始向下计数到 0，然后再从自

动装载的值重新开始计数，并产生一个计数器向下溢出事件。如果 PWMA_CR1 寄存器的 UDIS 位被清除，还会产生一个更新事件（UEV）。



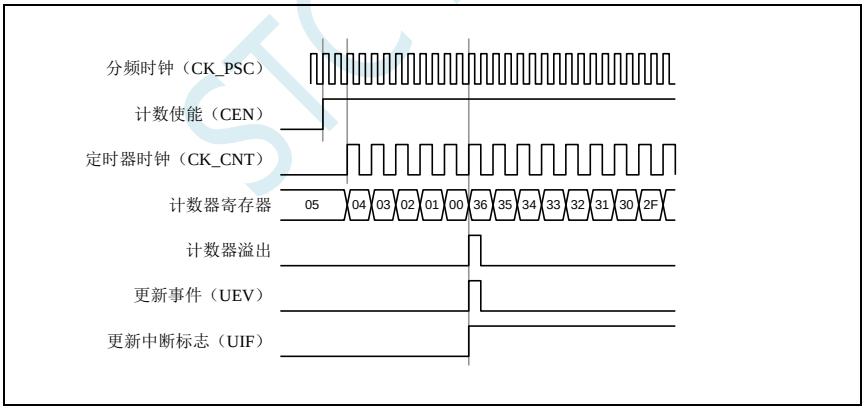
通过软件方式或者通过使用触发控制器置位 PWMA_EGR 寄存器的 UG 位同样也可以产生一个更新事件。置位 PWMA_CR1 寄存器的 UDIS 位可以禁止 UEV 事件。这样可以避免在更新预装载寄存器时更新影子寄存器。因此 UDIS 位清除之前不会产生更新事件。然而，计数器仍会从当前自动加载值重新开始计数，并且预分频器的计数器重新从 0 开始（但预分频器不能被修改）。此外，如果设置了 PWMA_CR1 寄存器中的 URS 位（选择更新请求），设置 UG 位将产生一个更新事件 UEV 但不设置 UIF 标志（因此不产生中断），这是为了避免在发生捕获事件并清除计数器时，同时产生更新和捕获中断。

当发生更新事件时，所有的寄存器都被更新，硬件依据 URS 位同时设置更新标志位（PWMA_SR 寄存器的 UIF 位）：

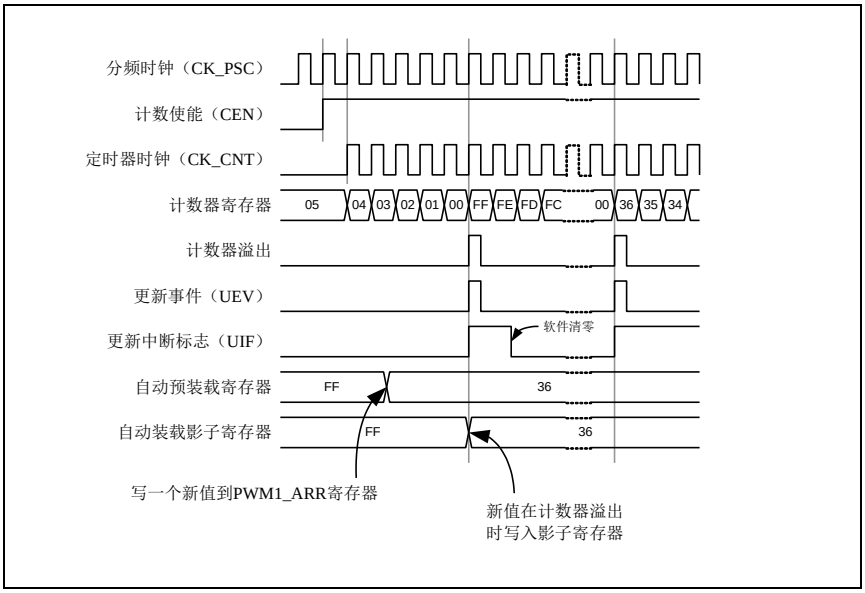
- 自动装载影子寄存器被重新置入预装载寄存器的值（PWMA_ARR）。
- 预分频器的缓存器被置入预装载寄存器的值（PWMA_PSC）。

以下是一些当 PWMA_ARR=0x36 时，计数器在不同时钟频率下的图表。下图描述了在向下计数模式下，预装载不使能时新的数值在下个周期时被写入。

当 ARPE=0（ARR 不预装载），预分频为 2 时的计数器更新：

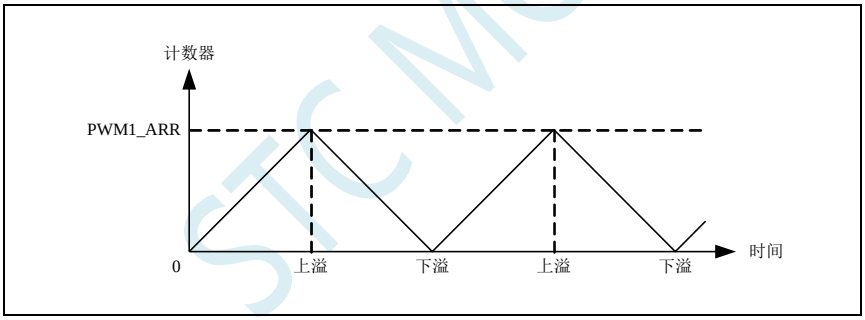


当 ARPE=1（ARR 预装载），预分频为 1 时的计数器更新



21.3.6 中间对齐模式（向上/向下计数）

在中央对齐模式，计数器从 0 开始计数到 PWMA_ARR 寄存器的值，产生一个计数器上溢事件，然后从 PWMA_ARR 寄存器的值向下计数到 0 并且产生一个计数器下溢事件；然后再从 0 开始重新计数。在此模式下，不能写入 PWMA_CR1 中的 DIR 方向位。它由硬件更新并指示当前的计数方向。



如果定时器带有重复计数器，在重复了指定次数（PWMA_RCR 的值）的向上和向下溢出之后会产生更新事件（UEV）。否则每一次的向上向下溢出都会产生更新事件。通过软件方式或者通过使用触发控制器置位 PWMA_EGR 寄存器的 UG 位同样也可以产生一个更新事件。此时，计数器重新从 0 开始计数，预分频器也重新从 0 开始计数。设置 PWMA_CR1 寄存器中的 UDIS 位可以禁止 UEV 事件。这样可以避免在更新预装载寄存器时更新影子寄存器。因此 UDIS 位被清为 0 之前不会产生更新事件。然而，计数器仍会根据当前自动重加载的值，继续向上或向下计数。如果定时器带有重复计数器，由于重复寄存器没有双重的缓冲，新的重复数值将立刻生效，因此在修改时需要小心。此外，如果设置了 PWMA_CR1 寄存器中的 URS 位（选择更新请求），设置 UG 位将产生一个更新事件 UEV 但不设置 UIF 标志（因此不产生中断），这是为了避免在发生捕获事件并清除计数器时，同时产生更新和捕获中断。

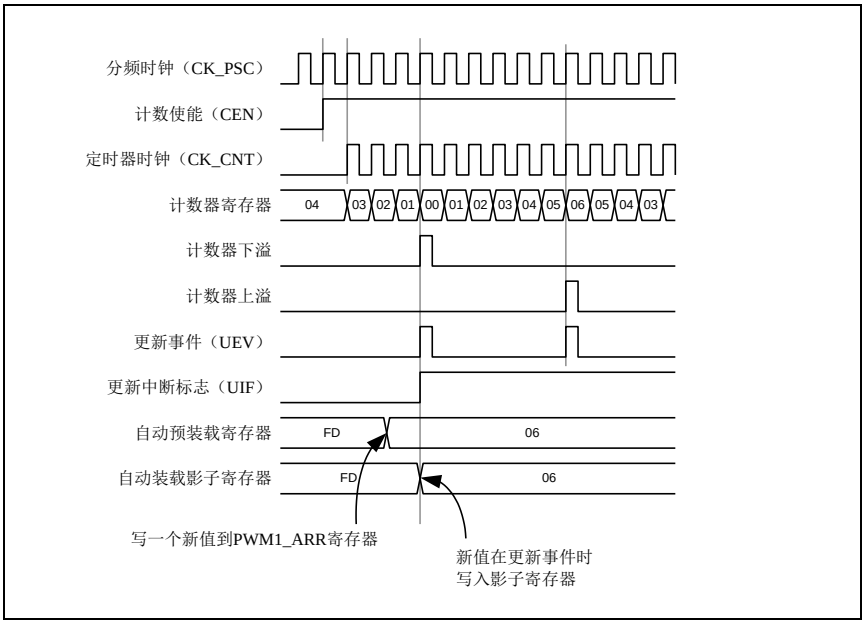
当发生更新事件时，所有的寄存器都被更新，硬件依据 URS 位更新标志位（PWMA_SR 寄存器中的 UIF 位）：

- 预分频器的缓存器被加载为预装载的值（PWMA_PSCR）。
- 当前的自动加载寄存器被更新为预装载值（PWMA_ARR）。

要注意到如果因为计数器溢出而产生更新，自动重装载寄存器将在计数器重载入之前被更新，因此下一个周期才是预期的值（计数器被装载为新的值）。

以下是一些计数器在不同时钟频率下的操作的例子：

内部时钟分频因子为 1，PWMA_ARR=0x6，ARPE＝1



使用中央对齐模式的提示：

- 启动中央对齐模式时，计数器将按照原有的向上/向下的配置计数。也就是说 PWMA_CR1 寄存器中的 DIR 位将决定计数器是向上还是向下计数。此外，软件不能同时修改 DIR 位和 CMS 位的值。
- 不推荐在中央对齐模式下，计数器正在计数时写计数器的值，这将导致不能预料的后果。具体的说：
  - 向计数器写入了比自动装载值更大的数值时（PWMA_CNT>PWMA_ARR），但计数器的计数方向不发生改变。例如计数器已经向上溢出，但计数器仍然向上计数。
  - 向计数器写入了 0 或者 PWMA_ARR 的值，但更新事件不发生。
- 安全使用中央对齐模式的计数器的方法是在启动计数器之前先用软件(置位 PWMA_EGR 寄存器的 UG 位）产生一个更新事件，并且不在计数器计数时修改计数器的值。

21.3.7 重复计数器

时基单元解释了计数器向上/向下溢出时更新事件（UEV）是如何产生的，然而事实上它只能在重复计数器的值达到 0 的时候产生。这个特性对产生 PWM 信号非常有用。

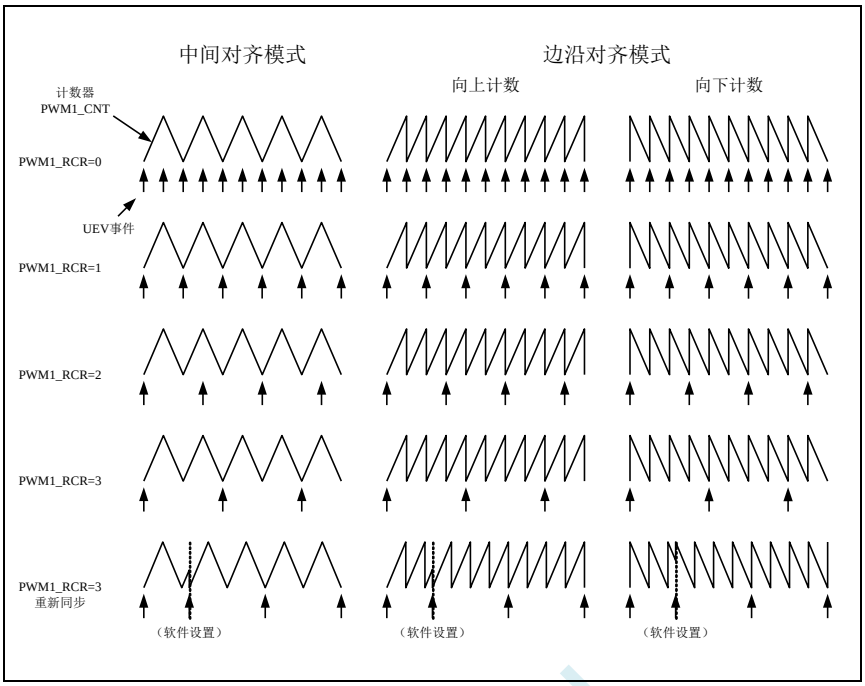
这意味着在每 N 次计数上溢或下溢时，数据从预装载寄存器传输到影子寄存器（PWMA_ARR 自动重载入寄存器，PWMA_PSCR 预装载寄存器，还有在比较模式下的捕获/比较寄存器 PWMA_CCRx），N 是 PWMA_RCR 重复计数寄存器中的值。

重复计数器在下述任一条件成立时递减：

- 向上计数模式下每次计数器向上溢出时
- 向下计数模式下每次计数器向下溢出时
- 中央对齐模式下每次上溢和每次下溢时。虽然这样限制了 PWM 的最大循环周期为 128，但它能够在每个 PWM 周期 2 次更新占空比。在中央对齐模式下，因为波形是对称的，如果每个 PWM 周期中仅刷新一次比较寄存器，则最大的分辨率为 $2*t_{CK_PSC}$。

重复计数器是自动加载的，重复速率由 PWMA_RCR 寄存器的值定义。当更新事件由软件产生或者通过硬件的时钟/触发控制器产生，则无论重复计数器的值是多少，立即发生更新事件，并且 PWMA_RCR 寄存器中的内容被重载入到重复计数器。

不同模式下更新速率的例子，及 PWMA_RCR 的寄存器设置



21.4 时钟/触发控制器

时钟/触发控制器允许用户选择计数器的时钟源，输入触发信号和输出信号，

21.4.1 预分频时钟（CK_PSC）

时基单元的预分频时钟（CK_PSC）可以由以下源提供：

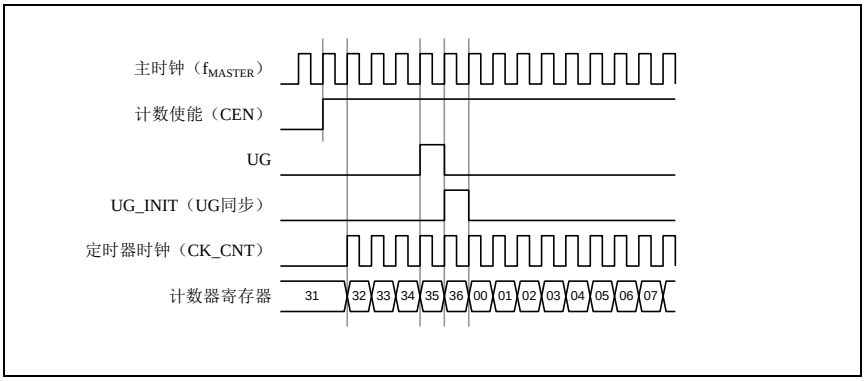
- 内部时钟（f_{MASTER}）
- 外部时钟模式 1：外部时钟输入（TIx）
- 外部时钟模式 2：外部触发输入 ETR
- 内部触发输入（ITRx）：使用一个 PWM 的 TRGO 做为另一个 PWM 的预分频时钟。

21.4.2 内部时钟源（f_{MASTER}）

如果同时禁止了时钟/触发模式控制器和外部触发输入（PWMA_SMCR 寄存器的 SMS=000，PWMA_ETR 寄存器的 ECE＝0），则 CEN、DIR 和 UG 位是实际上的控制位，并且只能被软件修改（UG 位仍被自动清除）。一旦 CEN 位被写成 1，预分频器的时钟就由内部时钟提供。

下图描述了控制电路和向上计数器在普通模式下，不带预分频器时的操作。

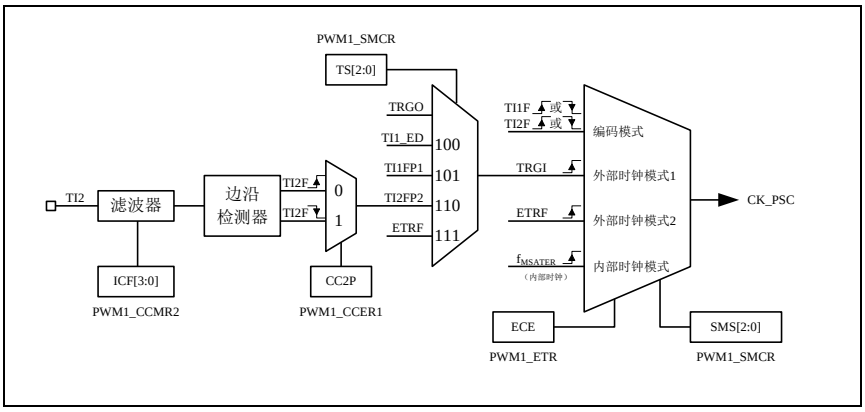
普通模式下的控制电路，f_{MASTER} 分频因子为 1



21.4.3 外部时钟源模式 1

当 PWMA_SMCR 寄存器的 SMS=111 时，此模式被选中。然后再通过 PWMA_SMCR 寄存器的 TS 选择 TRGI 的信号源。计数器可以在选定输入端的每个上升沿或下降沿计数。

下面的例子以 TI2 作为外部时钟



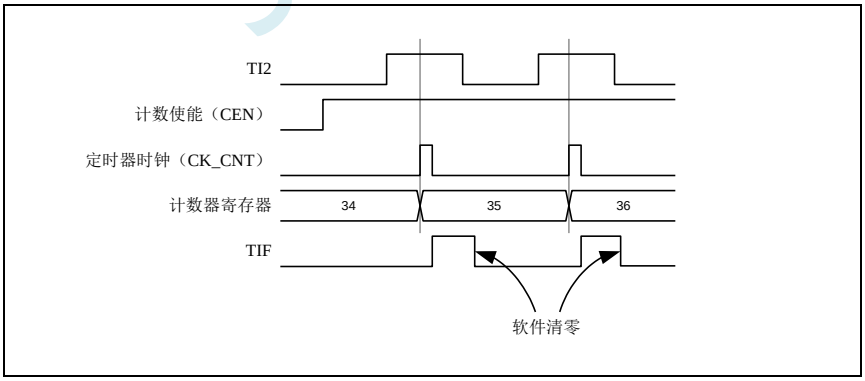
例如，要配置向上计数器在 TI2 输入端的上升沿计数，使用下列步骤：

1. 配置 PWMA_CCMR2 寄存器的 CC2S=01，使用通道 2 检测 TI2 输入的上升沿
2. 配置 PWMA_CCMR2 寄存器的 IC2F[3:0]位，选择输入滤波器带宽
3. 配置 PWMA_CCER1 寄存器的 CC2P=0，选定上升沿极性
4. 配置 PWMA_SMCR 寄存器的 SMS=111，配置计数器使用外部时钟模式 1
5. 配置 PWMA_SMCR 寄存器的 TS=110，选定 TI2 作为输入源
6. 设置 PWMA_CR1 寄存器的 CEN=1，启动计数器

当上升沿出现在 TI2，计数器计数一次，且触发标识位（PWMA_SR1 寄存器的 TIF 位）被置 1，如果使能了中断（在 PWMA_IER 寄存器中配置）则会产生中断请求。

在 TI2 的上升沿和计数器实际时钟之间的延时取决于在 TI2 输入端的重新同步电路。

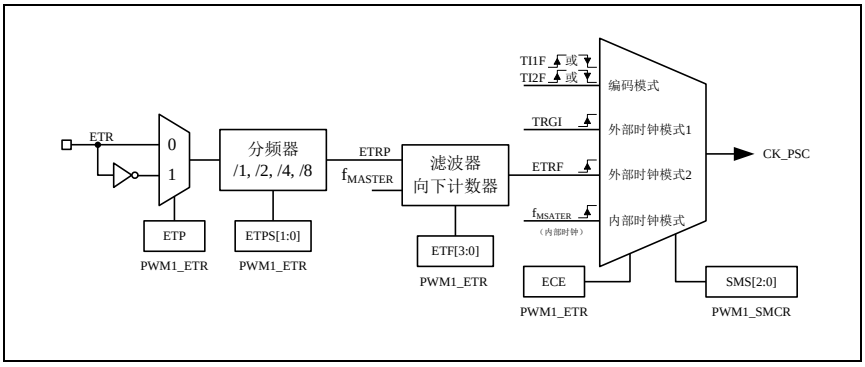
外部时钟模式 1 下的控制电路



21.4.4 外部时钟源模式 2

计数器能够在外部触发输入 ETR 信号的每一个上升沿或下降沿计数。将 PWMA_ETR 寄存器的 ECE 位写 1，即可选定此模式。（PWMA_SMCR 寄存器的 SMS=111 且 PWMA_SMCR 寄存器的 TS=111 时，也可选择此模式）

外部触发输入的总体框图：

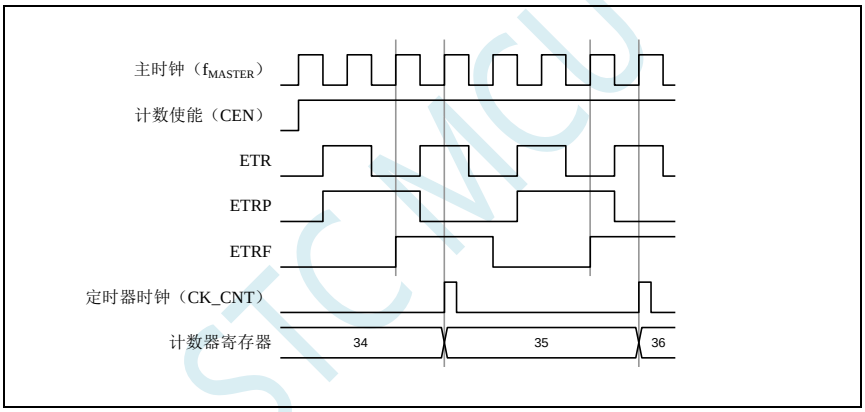


例如，要配置计数器在 ETR 信号的每 2 个上升沿时向上计数一次，需使用下列步骤：

1. 本例中不需要滤波器，配置 PWMA_ETR 寄存器的 ETF[3:0]=0000
2. 设置预分频器，配置 PWMA_ETR 寄存器的 ETPS[1:0]=01
3. 选择 ETR 的上升沿检测，配置 PWMA_ETR 寄存器的 ETP=0
4. 开启外部时钟模式 2，配置 PWMA_ETR 寄存器中的 ECE=1
5. 启动计数器，写 PWMA_CR1 寄存器的 CEN=1

计数器在每 2 个 ETR 上升沿计数一次。

外部时钟模式 2 下的控制电路



21.4.5 触发同步

PWMA 的计数器使用三种模式与外部的触发信号同步：

- 标准触发模式
- 复位触发模式
- 门控触发模式

标准触发模式

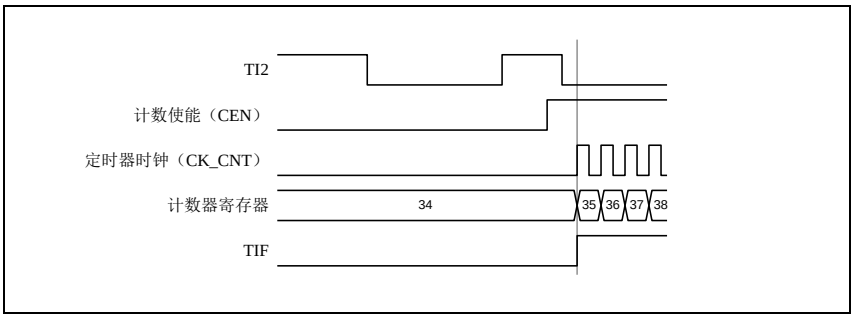
计数器的使能（CEN）依赖于选中的输入端上的事件。

在下面的例子中，计数器在 TI2 输入的上升沿开始向上计数：

1. 配置 PWMA_CCER1 寄存器的 CC2P＝0，选择 TI2 的上升沿做为触发条件。
2. 配置 PWMA_SMCR 寄存器的 SMS=110，选择计数器为触发模式。配置 PWMA_SMCR 寄存器的 TS=110，选择 TI2 作为输入源。

当 TI2 出现一个上升沿时，计数器开始在内部时钟驱动下计数，同时置位 TIF 标志。TI2 上升沿和计数器启动计数之间的延时取决于 TI2 输入端的重同步电路。

标准触发模式的控制电路



复位触发模式

在发生一个触发输入事件时，计数器和它的预分频器能够重新被初始化。同时，如果 PWMA_CR1 寄存器的 URS 位为低，还产生一个更新事件 UEV，然后所有的预装载寄存器（PWMA_ARR，PWMA_CCRx）都会被更新。

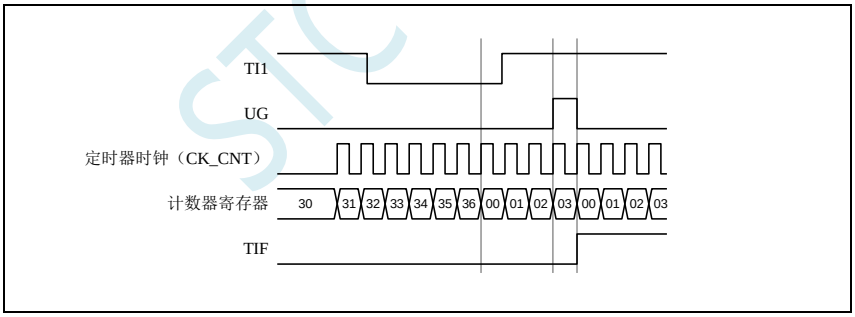
在以下的例子中，TI1 输入端的上升沿导致向上计数器被清零：

1. 配置 PWMA_CCER1 寄存器的 CC1P=0 来选择 TI1 的极性（只检测 TI1 的上升沿）。
2. 配置 PWMA_SMCR 寄存器的 SMS=100，选择定时器为复位触发模式。配置 PWMA_SMCR 寄存器的 TS=101，选择 TI1 作为输入源。
3. 配置 PWMA_CR1 寄存器的 CEN=1，启动计数器。

计数器开始依据内部时钟计数，然后正常计数直到 TI1 出现一个上升沿。此时，计数器被清零然后从 0 重新开始计数。同时，触发标志（PWMA_SR1 寄存器的 TIF 位）被置位，如果使能了中断（PWMA_IER 寄存器的 TIE 位），则产生一个中断请求。

下图显示当自动重装载寄存器 PWMA_ARR=0x36 时的动作。在 TI1 上升沿和计数器的实际复位之间的延时取决于 TI1 输入端的重同步电路。

复位触发模式下的控制电路



门控触发模式

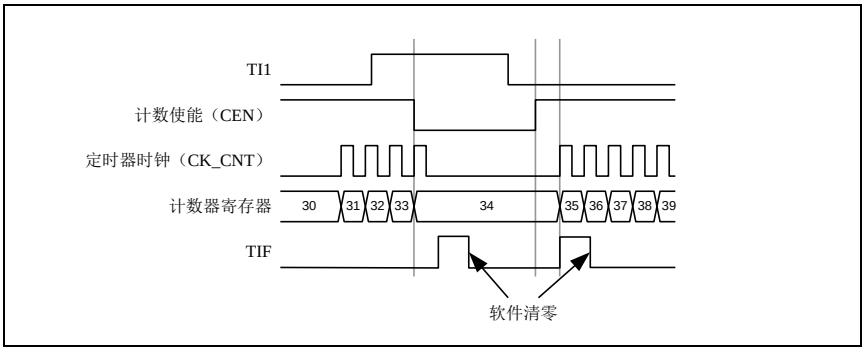
计数器由选中的输入端信号的电平使能。

在如下的例子中，计数器只在 TI1 为低时向上计数：

1. 配置 PWMA_CCER1 寄存器的 CC1P=1 来确定 TI1 的极性（只检测 TI1 上的低电平）。
2. 配置 PWMA_SMCR 寄存器的 SMS=101，选择定时器为门控触发模式，配置 PWMA_SMCR 寄存器中 TS=101，选择 TI1 作为输入源。
3. 配置 PWMA_CR1 寄存器的 CEN=1，启动计数器（在门控模式下，如果 CEN=0，则计数器不能启动，不论触发输入电平如何）。

只要 TI1 为低，计数器开始依据内部时钟计数，一旦 TI1 变高则停止计数。当计数器开始或停止时 TIF 标志位都会被置位。TI1 上升沿和计数器实际停止之间的延时取决于 TI1 输入端的重同步电路。

门控触发模式下的控制电路



外部时钟模式 2 联合触发模式

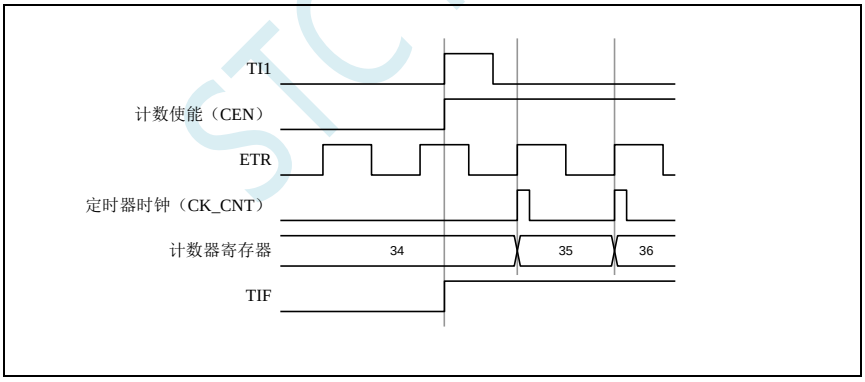
外部时钟模式 2 可以与另一个输入信号的触发模式一起使用。例如，ETR 信号被用作外部时钟的输入，另一个输入信号可用作触发输入（支持标准触发模式，复位触发模式和门控触发模式）。注意不能通过 PWMA_SMCR 寄存器的 TS 位把 ETR 配置成 TRGI。

在下面的例子中，一旦在 TI1 上出现一个上升沿，计数器即在 ETR 的每一个上升沿向上计数一次：

1. 通过 PWMA_ETR 寄存器配置外部触发输入电路。配置 ETPS=00 禁止预分频，配置 ETP=0 监测 ETR 信号的上升沿，配置 ECE=1 使能外部时钟模式 2。
2. 配置 PWMA_CCER1 寄存器的 CC1P=0 来选择 TI1 的上升沿触发。
3. 配置 PWMA_SMCR 寄存器的 SMS=110 来选择定时器为触发模式。配置 PWMA_SMCR 寄存器的 TS=101 来选择 TI1 作为输入源。

当 TI1 上出现一个上升沿时，TIF 标志被设置，计数器开始在 ETR 的上升沿计数。TI1 信号的上升沿和计数器实际时钟之间的延时取决于 TI1 输入端的重同步电路。ETR 信号的上升沿和计数器实际时钟之间的延时取决于 ETRP 输入端的重同步电路。

外部时钟模式 2+触发模式下的控制电路



21.4.6 与 PWMB 同步

在芯片中，定时器在内部互相联结，用于定时器的同步或链接。当某个定时器配置成主模式时，可以输出触发信号（TRGO）到那些配置为从模式的定时器来完成复位操作、启动操作、停止操作或者作为那些定时器的驱动时钟。

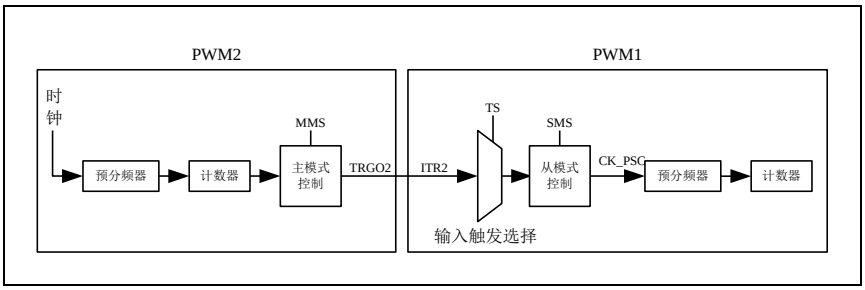
使用 PWMB 的 TRGO 作为 PWMA 的预分频时钟

例如，用户可以配置 PWMB 作为 PWMA 的预分频时钟，需进行如下配置：

1. 配置 PWMB 为主模式，使得在每个更新事件（UEV）时输出周期性的触发信号。配置 PWMB_CR2 寄存器的 MMS=010，使每个更新事件时 TRGO 能输出一个上升沿。
2. PWMB 输出的 TRGO 信号链接到 PWMA。PWMA 需要配置成触发从模式，使用 ITR2 作为输入触发信号。以上操作可以通过配置 PWMA_SMCR 寄存器的 TS=010 实现。

3. 配置 PWMA_SMCR 寄存器的 SMS=111 将时钟/触发控制器设置为外部时钟模式 1。此操作将使 PWMB 输出的周期性触发信号 TRGO 的上升沿驱动 PWMA 的时钟。
4. 最后，置位 PWMB 的 CEN 位（PWMB_CR1 寄存器中），使能两个 PWM。

主触发从模式的例子



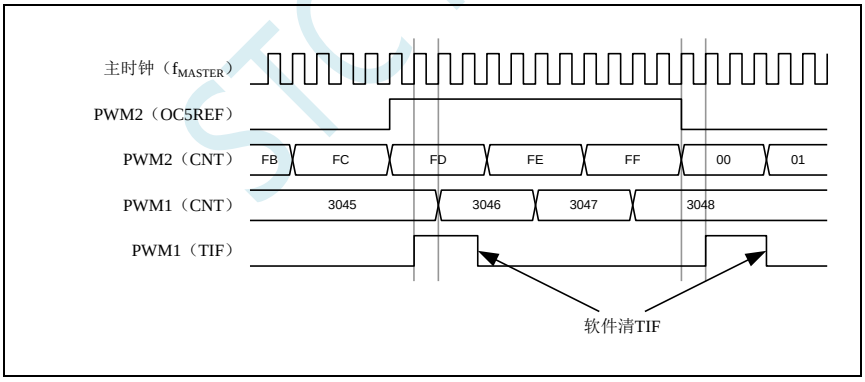
使用 PWMB 使能 PWMA

在本例中，我们用 PWMB 的比较输出使能 PWMA。PWMA 仅在 PWMB 的 OC1REF 信号为高时按照自己的驱动时钟计数。两个 PWM 都使用 4 分频的 f_{MASTER} 为时钟（$f_{CK_CNT} = f_{MASTER}/4$）。

1. 配置 PWMB 为主模式，将比较输出信号（OC5REF）作为触发信号输出。（配置 PWMB_CR2 寄存器的 MMS=100）。
2. 配置 PWMB 的 OC5REF 信号的波形（PWMB_CCMR1 寄存器）。
3. 配置 PWMA 把 PWMB 的输出作为自己的触发输入信号（配置 PWMA_SMCR 寄存器的 TS=010）。
4. 配置 PWMA 为门控触发模式（配置 PWMA_SMCR 寄存器的 SMS=101）。
5. 置位 CEN 位（PWMA_CR1 寄存器），使能 PWMA。
6. 置位 CEN 位（PWMB_CR1 寄存器），使能 PWMB。

注意：两个 PWM 的时钟并不同步，但仅影响 PWMA 的使能信号。

PWMB 的输出门控触发 PWMA

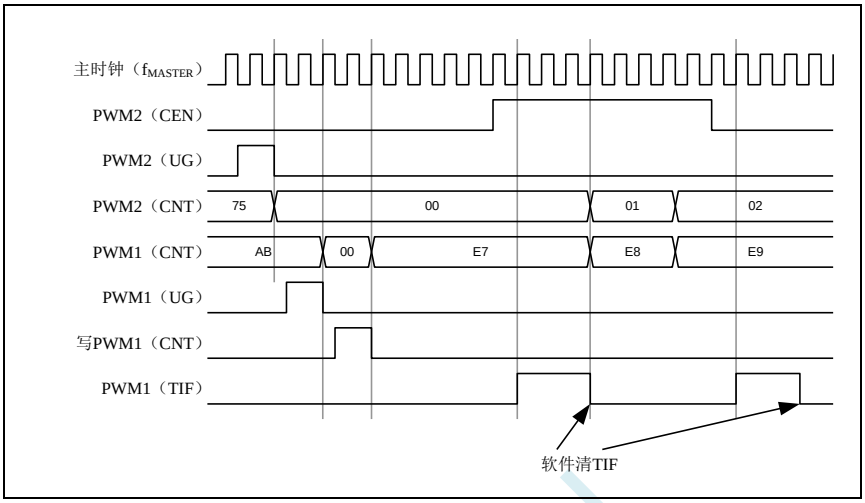


上图中，PWMA 的计数器和预分频器都没有在启动前初始化，所以都是从现有值开始计数的。如果在启动 PWMB 之前复位两个定时器，用户就可以写入期望的数值到 PWMA 的计数器，使之从指定值开始计数。对 PWMA 的复位操作可以通过软件写 PWMA_EGR 寄存器的 UG 位实现。

在下面这个例子中，我们使 PWMB 和 PWMA 同步。PWMB 为主模式并从 0 启动计数。PWMA 为触发从模式，并从 0xE7 启动计数。两个 PWM 采用相同的分频系数。当清除 PWMB_CR1 寄存器的 CEN 位时，PWMB 被禁止，同时 PWMA 停止计数。

1. 配置 PWMB 为主模式，将比较输出信号（OC5REF）作为触发信号输出。（配置 PWMB_CR2 寄存器的 MMS=100）。
2. 配置 PWMB 的 OC5REF 信号的波形（PWMB_CCMR1 寄存器）。
3. 配置 PWMA 把 PWMB 的输出作为自己的触发输入信号（配置 PWMA_SMCR 寄存器的 TS=010）。
4. 配置 PWMA 为门控触发模式（配置 PWMA_SMCR 寄存器的 SMS=101）。

5. 通过对 UG 位（PWMB_EGR 寄存器）写 1，复位 PWMB。
6. 通过对 UG 位（PWMA_EGR 寄存器）写 1，复位 PWMA。
7. 将 0xE7 写入 PWMA 的计数器中（PWMA_CNTRL），初始化 PWMA。
8. 通过对 CEN 位（PWMA_CR1 寄存器）写 1，使能 PWMA。
9. 通过对 CEN 位（PWMB_CR1 寄存器）写 1，启动 PWMB。
10. 通过对 CEN 位（PWMB_CR1 寄存器）写 0，停止 PWMB。



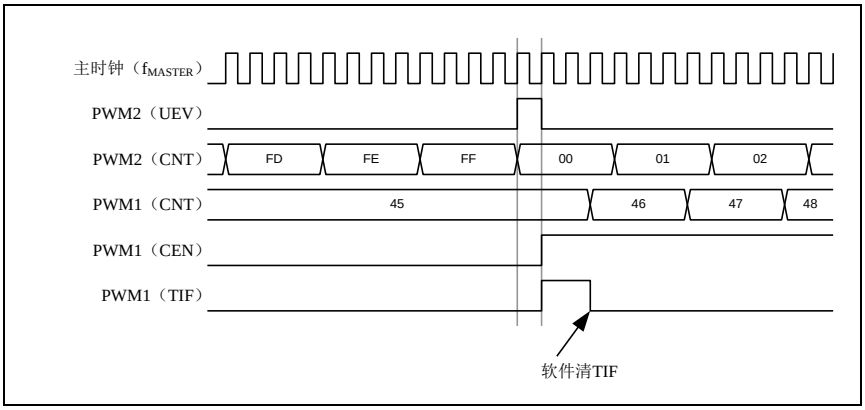
使用 PWMB 启动 PWMA

在本例中，我们用 PWMB 的更新事件来启动 PWMA。

PWMA 在 PWMB 发生更新事件时按照 PWMA 自己的驱动时钟从它的现有值开始计数（可以是非 0 值）。PWMA 在收到触发信号后自动使能 CEN 位，并开始计数，一直持续到用户向 PWMA_CR1 寄存器的 CEN 位写 0。两个 PWM 都使用 4 分频的 f_{MASTER} 作为驱动时钟（$f_{CK_CNT} = f_{MASTER}/4$）。

1. 配置 PWMB 为主模式，输出更新信号（UEV）。（配置 PWMB_CR2 寄存器的 MMS=010）。
2. 配置 PWMB 的周期（PWMB_ARR 寄存器）。
3. 配置 PWMA 用 PWMB 的输出作为输入的触发信号（配置 PWMA_SMCR 寄存器的 TS=010）。
4. 配置 PWMA 为触发模式（配置 PWMA_SMCR 寄存器的 SMS=110）。
5. 置位 CEN 位（PWMB_CR1 寄存器）启动 PWMB。

PWMB 的更新事件（PWMB-UEV）触发 PWMA



如同前面的例子，用户也可以在启动计数器前对它们初始化。

用外部信号同步的触发两个 PWM

在本例中，使用 TI1 的上升沿使能 PWMB，并同时使能 PWMA。为了保持定时器的对齐，PWMB 需要配置成主/从模式（对于 TI1 信号为从模式，对于 PWMA 为主模式）。

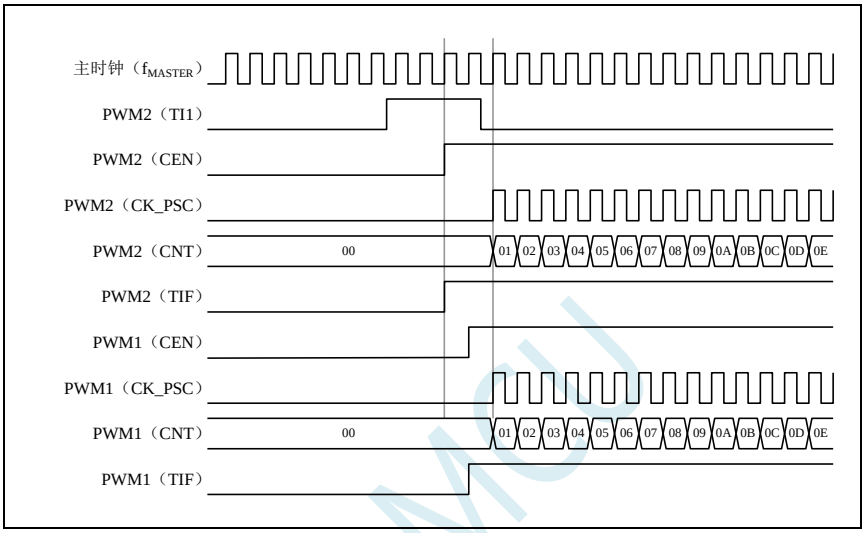
1. 配置 PWMB 为主模式，以输出使能信号作为 PWMA 的触发（配置 PWMB_CR2 寄存器的 MMS=001）。

2. 配置 PWMB 为从模式，把 TI1 信号作为输入的触发信号（配置 PWMB_SMCR 寄存器的 TS=100）。
3. 配置 PWMB 的触发模式（配置 PWMB_SMCR 寄存器的 SMS=110）。
4. 配置 PWMB 为主/从模式（配置 PWMB_SMCR 寄存器的 MSM=1）。
5. 配置 PWMA 以 PWMB 的输出为输入触发信号（配置 PWMA_SMCR 寄存器的 TS=010）。
6. 配置 PWMA 的触发模式（配置 PWMA_SMCR 寄存器的 SMS=110）。

当 TI1 上出现上升沿时，两个定时器同步的开始计数，并且 TIF 位都被置起。

注意：在本例中，两个定时器在启动前都进行了初始化（设置 UG 位），所以它们都从 0 开始计数，但是用户也可以通过修改计数器寄存器（PWMA_CNT）来插入一个偏移量，这样的话，在 PWMB 的 CK_PSC 信号和 CNT_EN 信号间会插入延时。

PWMB 的 TI1 信号触发 PWMB 和 PWMA

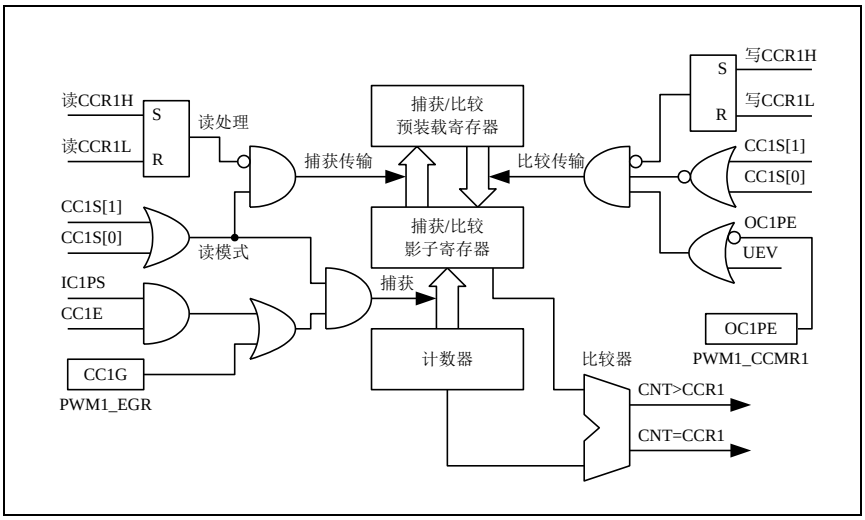


21.5 捕获/比较通道

PWM1P、PWM2P、PWM3P、PWM4P 可以用作输入捕获，PWM1P/PWM1N、PWM2P/PWM2N、PWM3P/PWM3N、PWM4P/PWM4N 可以输出比较，这个功能可以通过配置捕获/比较通道模式寄存器（PWMA_CCMRi）的 CCiS 通道选择位来实现，此处的 i 代表 1~4 的通道数。

每一个捕获/比较通道都是围绕着一个捕获/比较寄存器（包含影子寄存器）来构建的，包括捕获的输入部分（数字滤波、多路复用和预分频器）和输出部分（比较器和输出控制）。

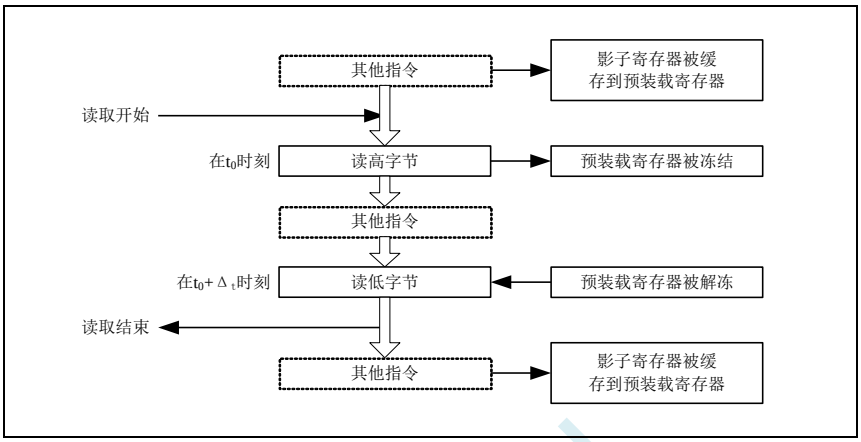
捕获/比较通道 1 的主要电路（其他通道与此类似）



捕获/比较模块由一个预装载寄存器和一个影子寄存器组成。读写过程仅操作预装载寄存器。在捕获模式下，捕获发生在影子寄存器上，然后再复制到预装载寄存器中。在比较模式下，预装载寄存器的内容被复制到影子寄存器中，然后影子寄存器的内容和计数器进行比较。

当通道被配置成输出模式时，可以随时访问 PWMA_CCRi 寄存器。

当通道被配置成输入模式时，对 PWMA_CCRi 寄存器的读操作类似于计数器的读操作。当捕获发生时，计数器的内容被捕获到 PWMA_CCRi 影子寄存器，随后再复制到预装载寄存器中。在读操作进行中，预装载寄存器是被冻结的。



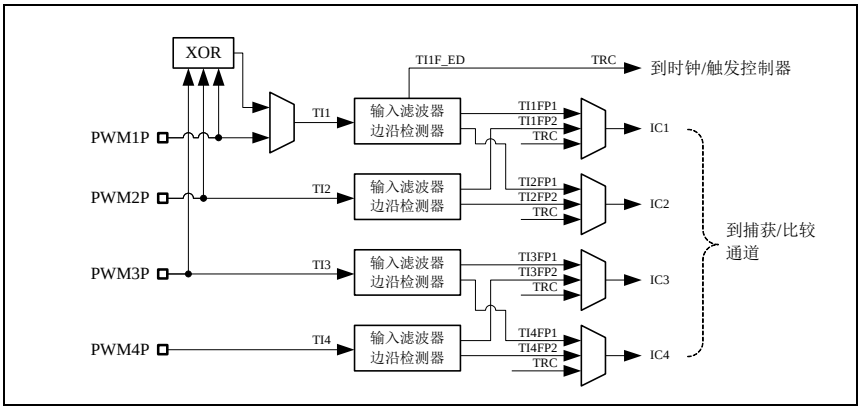
上图描述了 16 位的 CCRi 寄存器的读操作流程，被缓存的数据将保持不变直到读流程结束。在整个读流程结束后，如果仅仅读了 PWMA_CCRiL 寄存器，返回计数器数值的低位。如果在读了低位数据以后再读高位数据，将不再返回同样的低位数据。

21.5.1 16 位 PWMA_CCRi 寄存器的写流程

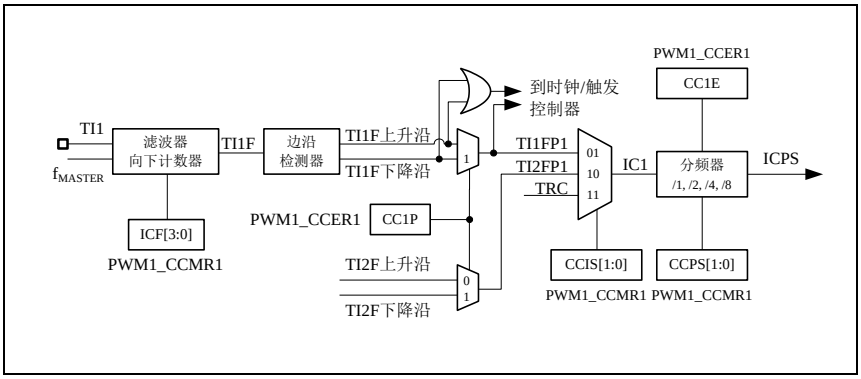
16 位 PWMA_CCRi 寄存器的写操作通过预装载寄存器完成。必需使用两条指令来完成整个流程，一条指令对应一个字节。必需先写高位字节。在写高位字节时，影子寄存器的更新被禁止直到低位字节的写操作完成。

21.5.2 输入模块

输入模块的框图



如图，输入部分对相应的 TIx 输入信号采样，并产生一个滤波后的信号 TIxF。然后，一个带极性选择的边缘监测器产生一个信号（TIxFPx），它可以作为触发模式控制器的输入触发或者作为捕获控制。该信号通过预分频后进入捕获寄存器（ICxPS）。



21.5.3 输入捕获模式

在输入捕获模式下，当检测到 ICi 信号上相应的边沿后，计数器的当前值被锁存到捕获/比较寄存器（PWMA_CCRx）中。当发生捕获事件时，相应的 CCiIF 标志（PWMA_SR 寄存器）被置 1。如果 PWMA_IER 寄存器的 CCiIE 位被置位，也就是使能了中断，则将产生中断请求。如果发生捕获事件时 CCiIF 标志已经为高，那么重复捕获标志 CCiOF（PWMA_SR2 寄存器）被置 1。写 CCiIF=0 或读取存储在 PWMA_CCRiL 寄存器中的捕获数据都可清除 CCiIF。写 CCiOF=0 可清除 CCiOF。

PWM 输入信号上升沿时捕获

以下例子说明如何在 TI1 输入的上升沿时捕获计数器的值到 PWMA_CCR1 寄存器中，步骤如下：

1. 选择有效输入端，设置 PWMA_CCMR1 寄存器中的 CC1S=01，此时通道被配置为输入，并且 PWMA_CCR1 寄存器变为只读。
2. 根据输入信号 TIi 的特点，可通过配置 PWMA_CCMR1 寄存器中的 IC1F 位来设置相应的输入滤波器的滤波时间。假设输入信号在最多 5 个时钟周期的时间内抖动，我们须配置滤波器的带宽长于 5 个时钟周期；因此我们可以连续采样 8 次，以确认在 TI1 上一次真实的边沿变换，即在 PWMA_CCMR1 寄存器中写入 IC1F=0011，此时，只有连续采样到 8 个相同的 TI1 信号，信号才为有效（采样频率为 f_{MASTER}）。
3. 选择 TI1 通道的有效转换边沿，在 PWMA_CCER1 寄存器中写入 CC1P=0（上升沿）。
4. 配置输入预分频器。在本例中，我们希望捕获发生在每一个有效的电平转换时刻，因此预分频器被禁止（写 PWMA_CCMR1 寄存器的 IC1PS=00）。
5. 设置 PWMA_CCER1 寄存器的 CC1E=1，允许捕获计数器的值到捕获寄存器中。
6. 如果需要，通过设置 PWMA_IER 寄存器中的 CC1IE 位允许相关中断请求。

当发生一个输入捕获时：

- 当产生有效的电平转换时，计数器的值被传送到 PWMA_CCR1 寄存器。
- CC1IF 标志被设置。当发生至少 2 个连续的捕获时，而 CC1IF 未曾被清除时，CC1OF 也被置 1。
- 如设置了 CC1IE 位，则会产生一个中断。

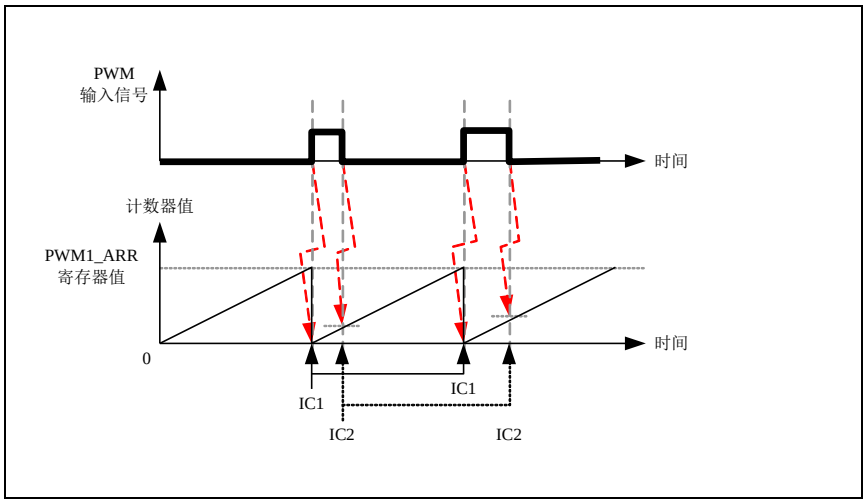
为了处理捕获溢出事件（CC1OF 位），建议在读出重复捕获标志之前读取数据，这是为了避免丢失在读出捕获溢出标志之后和读取数据之前可能产生的重复捕获信息。

注意：设置 PWMA_EGR 寄存器中相应的 CCiG 位，可以通过软件产生输入捕获中断。

PWM 输入信号测量

该模式是输入捕获模式的一个特例，除下列区别外，操作与输入捕获模式相同：

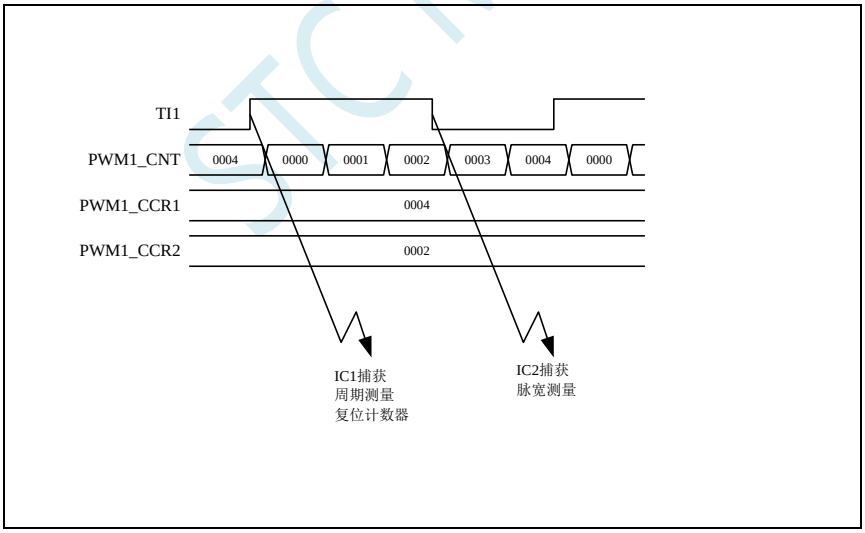
- 两个 ICi 信号被映射至同一个 TIi 输入。
- 这两个 ICi 信号的有效边沿的极性相反。
- 其中一个 TIiFP 信号被作为触发输入信号，而触发模式控制器被配置成复位触发模式。



例如，你可以用以下方式测量 TI1 上输入的 PWM 信号的周期（PWMA_CCR1 寄存器）和占空比（PWMA_CCR2 寄存器）。

1. 选择 PWMA_CCR1 的有效输入：置 PWMA_CCMR1 寄存器的 CC1S=01（选中 TI1FP1）。
2. 选择 TI1FP1 的有效极性：置 CC1P=0（上升沿有效）。
3. 选择 PWMA_CCR2 的有效输入：置 PWMA_CCMR2 寄存器的 CC2S=10（选中 TI1FP2）。
4. 选择 TI1FP2 的有效极性（捕获数据到 PWMA_CCR2）：置 CC2P=1（下降沿有效）。
5. 选择有效的触发输入信号：置 PWMA_SMCR 寄存器中的 TS=101（选择 TI1FP1）。
6. 配置触发模式控制器为复位触发模式：置 PWMA_SMCR 中的 SMS=100。
7. 使能捕获：置 PWMA_CCER1 寄存器中 CC1E=1，CC2E=1。

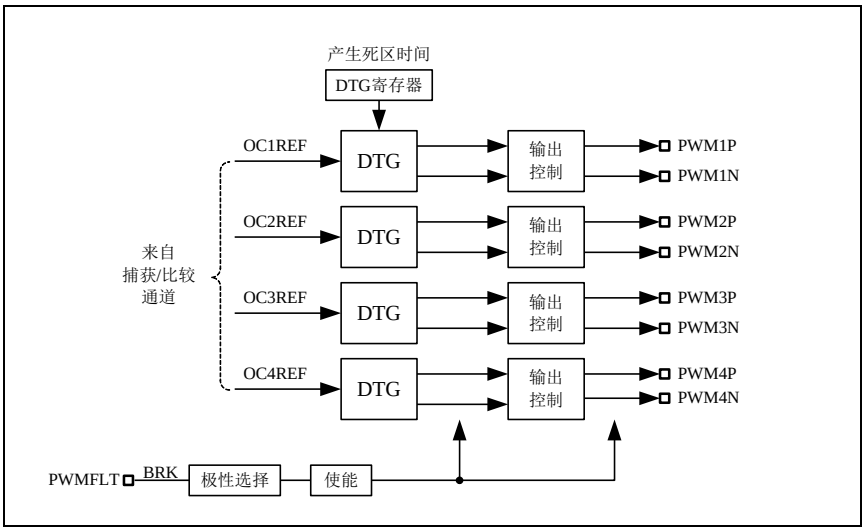
PWM 输入信号测量实例



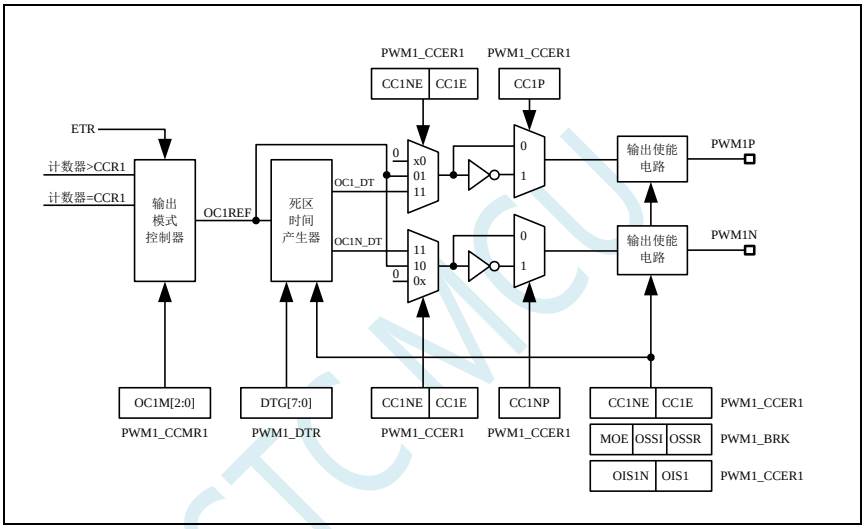
21.5.4 输出模块

输出模块会产生一个用来做参考的中间波形，称为 OCiREF（高有效）。刹车功能和极性的处理都在模块的最后处理。

输出模块框图



通道 1 详细的带互补输出的输出模块框图（其他通道类似）



21.5.5 强制输出模式

在输出模式下，输出比较信号能够直接由软件强制为高或低状态，而不依赖于输出比较寄存器和计数器间的比较结果。

置 PWMA_CCMRi 寄存器的 OCiM=101，可强制 OCiREF 信号为低。

置 PWMA_CCMRi 寄存器的 OCiM=100，可强制 OCiREF 信号为低。

OCi/OCiN 的输出是高还是低则取决于 CCiP/CCiNP 极性标志位。

该模式下，在 PWMA_CCRi 影子寄存器和计数器之间的比较仍然在进行，相应的标志也会被修改，也仍然会产生相应的中断。

21.5.6 输出比较模式

此模式用来控制一个输出波形或者指示一段给定的的时间已经达到。

当计数器与捕获/比较寄存器的内容相匹配时，有如下操作：

- 根据不同的输出比较模式，相应的 OCi 输出信号：
  - 保持不变（OCiM=000）
  - 设置为有效电平（OCiM=001）
  - 设置为无效电平（OCiM=010）

— 翻转（OCiM=011）

- 设置中断状态寄存器中的标志位（PWMA_SR1 寄存器中的 CCiIF 位）。
- 若设置了相应的中断使能位（PWMA_IER 寄存器中的 CCiIE 位），则产生一个中断。

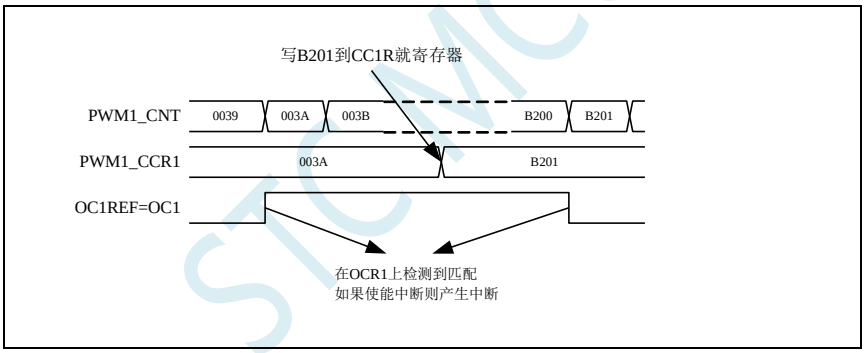
PWMA_CCMRi 寄存器的 OCiM 位用于选择输出比较模式，而 PWMA_CCMRi 寄存器的 CCiP 位用于选择有效和无效的电平极性。PWMA_CCMRi 寄存器的 OCiPE 位用于选择 PWMA_CCRi 寄存器是否需要使用预装载寄存器。在输出比较模式下，更新事件 UEV 对 OCiREF 和 OCi 输出没有影响。时间精度为计数器的一个计数周期。输出比较模式也能用来输出一个单脉冲。

输出比较模式的配置步骤：

1. 选择计数器时钟（内部、外部或者预分频器）。
2. 将相应的数据写入 PWMA_ARR 和 PWMA_CCRi 寄存器中。
3. 如果要产生一个中断请求，设置 CCiIE 位。
4. 选择输出模式步骤：
   1. 设置 OCiM=011，在计数器与 CCRi 匹配时翻转 OCiM 管脚的输出
   2. 设置 OCiPE = 0，禁用预装载寄存器
   3. 设置 CCiP = 0，选择高电平为有效电平
   4. 设置 CCiE = 1，使能输出
   5. 设置 PWMA_CR1 寄存器的 CEN 位来启动计数器

PWMA_CCRi 寄存器能够在任何时候通过软件进行更新以控制输出波形，条件是未使用预装载寄存器（OCiPE=0），否则 PWMA_CCRi 的影子寄存器只能在发生下一次更新事件时被更新。

输出比较模式，翻转 OC1



21.5.7 PWM 模式

脉冲宽度调制（PWM）模式可以产生一个由 PWMA_ARR 寄存器确定频率，由 PWMA_CCRi 寄存器确定占空比的信号。

在 PWMA_CCMRi 寄存器中的 OCiM 位写入 110（PWM 模式 1）或 111（PWM 模式 2），能够独立地设置每个 OCi 输出通道产生一路 PWM。必须设置 PWMA_CCMRi 寄存器的 OCiPE 位使能相应的预装载寄存器，也可以设置 PWMA_CR1 寄存器的 ARPE 位使能自动重装载的预装载寄存器（在向上计数模式或中央对称模式中）。

由于仅当发生一个更新事件的时候，预装载寄存器才能被传送到影子寄存器，因此在计数器开始计数之前，必须通过设置 PWMA_EGR 寄存器的 UG 位来初始化所有的寄存器。

OCi 的极性可以通过软件在 PWMA_CCERi 寄存器中的 CCiP 位设置，它可以设置为高电平有效或低电平有效。OCi 的输出使能通过 PWMA_CCERi 和 PWMA_BKR 寄存器中的 CCiE、MOE、OISi、OSSR 和 OSSI 位的组合来控制。

在 PWM 模式（模式 1 或模式 2）下，PWMA_CNT 和 PWMA_CCRi 始终在进行比较，（依据计数器的计数方向）以确定是否符合 PWMA_CCRi≤PWMA_CNT 或者 PWMA_CNT≤PWMA_CCRi。

根据 PWMA_CR1 寄存器中 CMS 位域的状态，定时器能够产生边沿对齐的 PWM 信号或中央对齐的

PWM 信号。

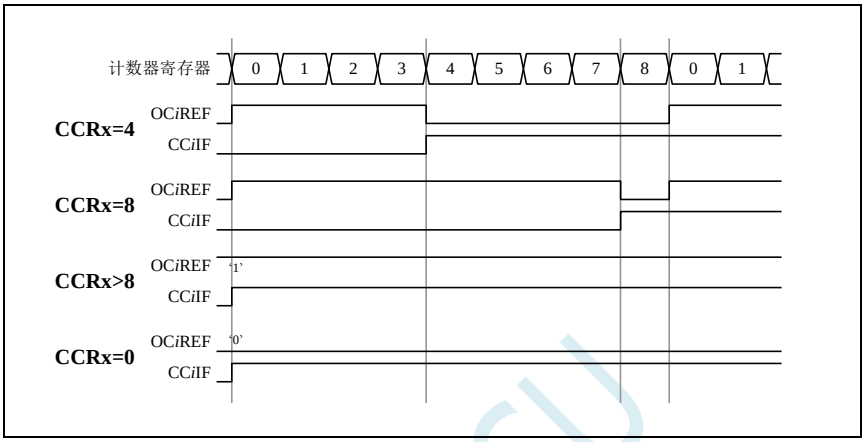
PWM 边沿对齐模式

向上计数配置

当 PWMA_CR1 寄存器中的 DIR 位为 0 时，执行向上计数。

下面是一个 PWM 模式 1 的例子。当 PWMA_CNT<PWMA_CCRi 时，PWM 参考信号 OCiREF 为高，否则为低。如果 PWMA_CCRi 中的比较值大于自动重装载值（PWMA_ARR），则 OCiREF 保持为高。如果比较值为 0，则 OCiREF 保持为低。

边沿对齐，PWM 模式 1 的波形（ARR=8）



向下计数的配置

当 PWMA_CR1 寄存器的 DIR 位为 1 时，执行向下计数。

在 PWM 模式 1 时，当 PWMA_CNT>PWMA_CCRi 时参考信号 OCiREF 为低，否则为高。如果 PWMA_CCRi 中的比较值大于 PWMA_ARR 中的自动重装载值，则 OCiREF 保持为高。该模式下不能产生占空比为 0%的 PWM 波形。

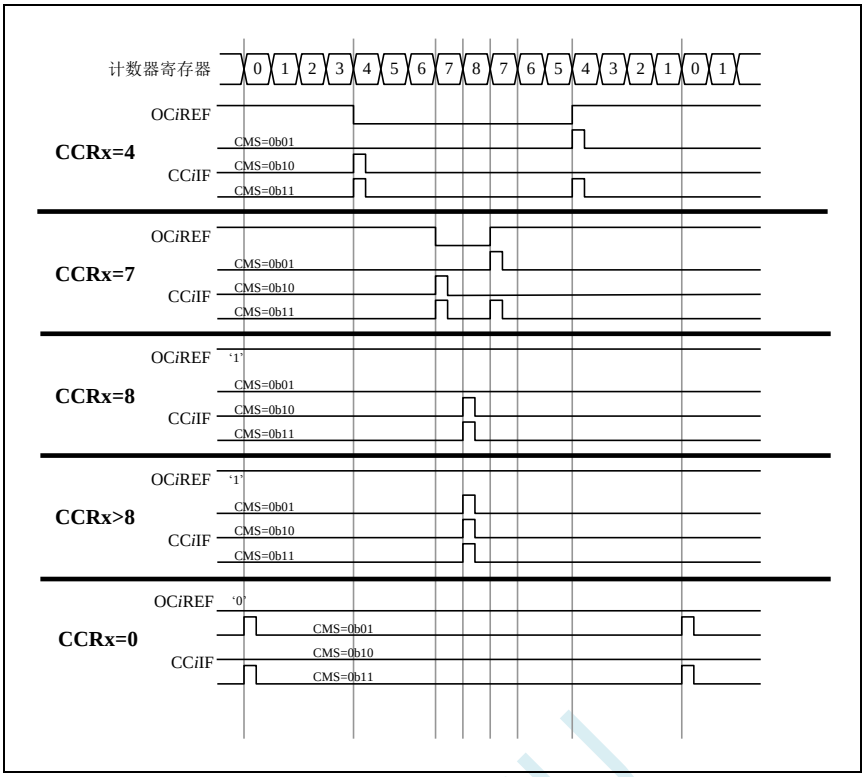
PWM 中央对齐模式

当 PWMA_CR1 寄存器中的 CMS 位不为'00'时为中央对齐模式（所有其他的配置对 OCiREF/OCi 信号都有相同的作用）。

根据不同的 CMS 位的设置，比较标志可以在计数器向上计数，向下计数，或向上和向下计数时被置 1。PWMA_CR1 寄存器中的计数方向位（DIR）由硬件更新，不要用软件修改它。

下面给出了一些中央对齐的 PWM 波形的例子：

- PWMA_ARR=8
- PWM 模式 1
- 标志位在以下三种情况下被置位：
  - 只有在计数器向下计数时（CMS=01）
  - 只有在计数器向上计数时（CMS=10）
  - 在计数器向上和向下计数时（CMS=11）
- 中央对齐的 PWM 波形（ARR=8）



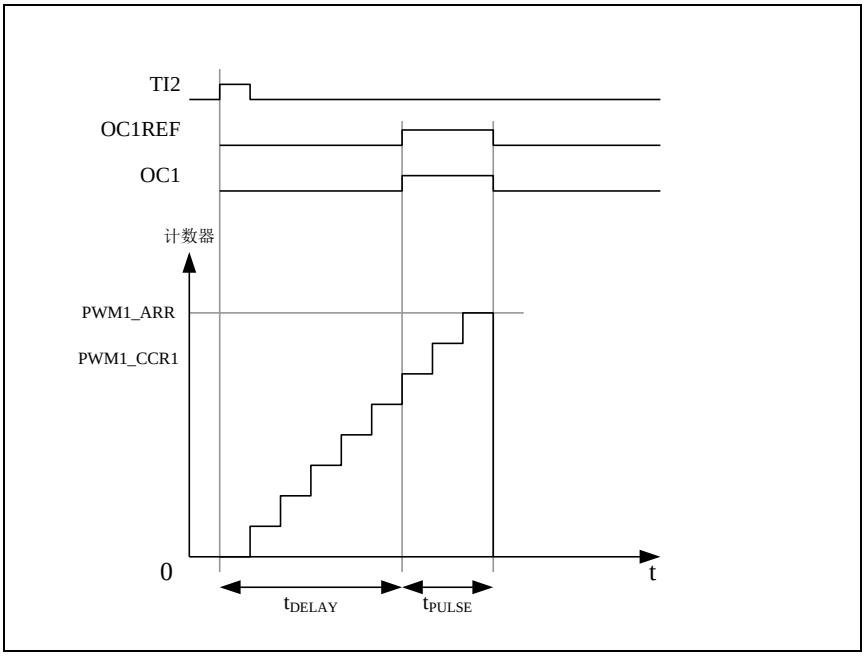
单脉冲模式

单脉冲模式（OPM）是前述众多模式的一个特例。这种模式允许计数器响应一个激励，并在一个程序可控的延时之后产生一个脉宽可控的脉冲。

可以通过时钟/触发控制器启动计数器，在输出比较模式或者 PWM 模式下产生波形。设置 PWMA_CR1 寄存器的 OPM 位将选择单脉冲模式，此时计数器自动地在下一个更新事件 UEV 时停止。仅当比较值与计数器的初始值不同时，才能产生一个脉冲。启动之前（当定时器正在等待触发），必须如下配置：

- 向上计数方式：计数器 CNT < CCRi ≤ ARR，
- 向下计数方式：计数器 CNT > CCRi。

单脉冲模式图例



例如，在从 TI2 输入脚上检测到一个上升沿之后延迟 t_{DELAY}，在 OC1 上产生一个 t_{PULSE} 宽度的正脉冲：（假定 IC2 作为触发 1 通道的触发源）

- 置 PWMA_CCMR2 寄存器的 CC2S=01，把 IC2 映射到 TI2。
- 置 PWMA_CCER1 寄存器的 CC2P=0，使 IC2 能够检测上升沿。
- 置 PWMA_SMCR 寄存器的 TS=110，使 IC2 作为时钟/触发控制器的触发源（TRGI）。
- 置 PWMA_SMCR 寄存器的 SMS=110（触发模式），IC2 被用来启动计数器。OPM 的波形由写入比较寄存器的数值决定（要考虑时钟频率和计数器预分频器）。
- t_{DELAY} 由 PWMA_CCR1 寄存器中的值定义。
- t_{PULSE} 由自动装载值和比较值之间的差值定义（PWMA_ARR – PWMA_CCR1）。
- 假定当发生比较匹配时要产生从 0 到 1 的波形，当计数器达到预装载值时要产生一个从 1 到 0 的波形，首先要置 PWMA_CCMR1 寄存器的 OCiM=111，进入 PWM 模式 2，根据需要有选择的设置 PWMA_CCMR1 寄存器的 OC1PE=1，置位 PWMA_CR1 寄存器中的 ARPE，使能预装载寄存器，然后在 PWMA_CCR1 寄存器中填写比较值，在 PWMA_ARR 寄存器中填写自动装载值，设置 UG 位来产生一个更新事件，然后等待在 TI2 上的一个外部触发事件。

在这个例子中，PWMA_CR1 寄存器中的 DIR 和 CMS 位应该置低。

因为只需要一个脉冲，所以设置 PWMA_CR1 寄存器中的 OPM=1，在下一个更新事件（当计数器从自动装载值翻转到 0）时停止计数。

OCx 快速使能（特殊情况）

在单脉冲模式下，对 TIi 输入脚的边沿检测会设置 CEN 位以启动计数器，然后计数器和比较值间的比较操作产生了单脉冲的输出。但是这些操作需要一定的时钟周期，因此它限制了可得到的最小延时 t_{DELAY}。

如果要以最小延时输出波形，可以设置 PWMA_CCMRi 寄存器中的 OCiFE 位，此时强制 OCiREF（和 OCx）直接响应激励而不再依赖比较的结果，输出的波形与比较匹配时的波形一样。OCiFE 只在通道配置为 PWMA 和 PWMB 模式时起作用。

互补输出和死区插入

PWMA 能够输出两路互补信号，并且能够管理输出的瞬时关断和接通，这段时间通常被称为死区，用户应该根据连接的输出器件和它们的特性（电平转换的延时、电源开关的延时等）来调整死区时间。

配置 PWMA_CCERi 寄存器中的 CCiP 和 CCiNP 位，可以为每一个输出独立地选择极性（主输出 OCi 或互补输出 OCiN）。互补信号 OCi 和 OCiN 通过下列控制位的组合进行控制：PWMA_CCERi 寄存器的 CCiE 和 CCiNE 位，PWMA_BKR 寄存器中的 MOE、OISi、OISiN、OSSI 和 OSSR 位。特别的是，在转换到 IDLE 状态时（MOE 下降到 0）死区控制被激活。

同时设置 CCiE 和 CCiNE 位将插入死区，如果存在刹车电路，则还要设置 MOE 位。每一个通道都有一个 8 位的死区发生器。

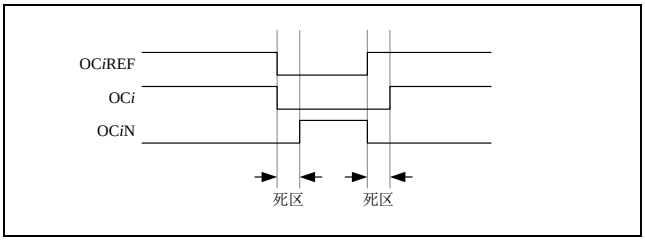
如果 OCi 和 OCiN 为高有效：

- OCi 输出信号与 OCiREF 相同，只是它的上升沿相对于 OCiREF 的上升沿有一个延迟。
- OCiN 输出信号与 OCiREF 相反，只是它的上升沿相对于 OCiREF 的下降沿有一个延迟。

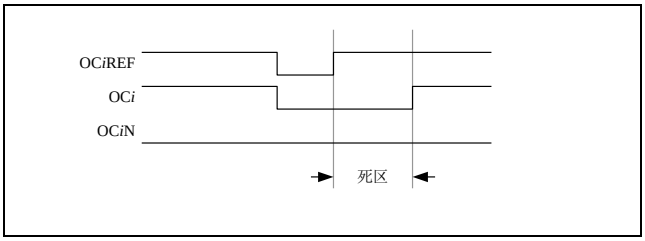
如果延迟大于当前有效的输出宽度（OCi 或者 OCiN），则不会产生相应的脉冲。

下列几张图显示了死区发生器的输出信号和当前参考信号 OCiREF 之间的关系。（假设 CCiP=0、CCiNP=0、MOE=1、CCiE=1 并且 CCiNE=1）

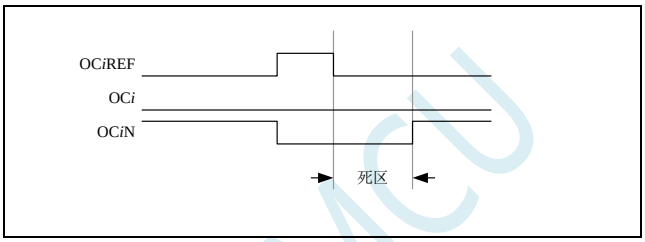
带死区插入的互补输出



死区波形延迟大于负脉冲



死区波形延迟大于正脉冲



每一个通道的死区延时都是相同的，是由 PWMA_DTR 寄存器中的 DTG 位编程配置。

重定向 OCiREF 到 OCi 或 OCiN

在输出模式下(强制输出、输出比较或 PWM 输出)，通过配置 PWMA_CCERi 寄存器的 CCiE 和 CCiNE 位，OCiREF 可以被重定向到 OCi 或者 OCiN 的输出。

这个功能可以在互补输出处于无效电平时，在某个输出上送出一个特殊的波形（例如 PWM 或者静态有效电平）。另一个作用是，让两个输出同时处于无效电平，或同时处于有效电平（此时仍然是带死区的互补输出）。

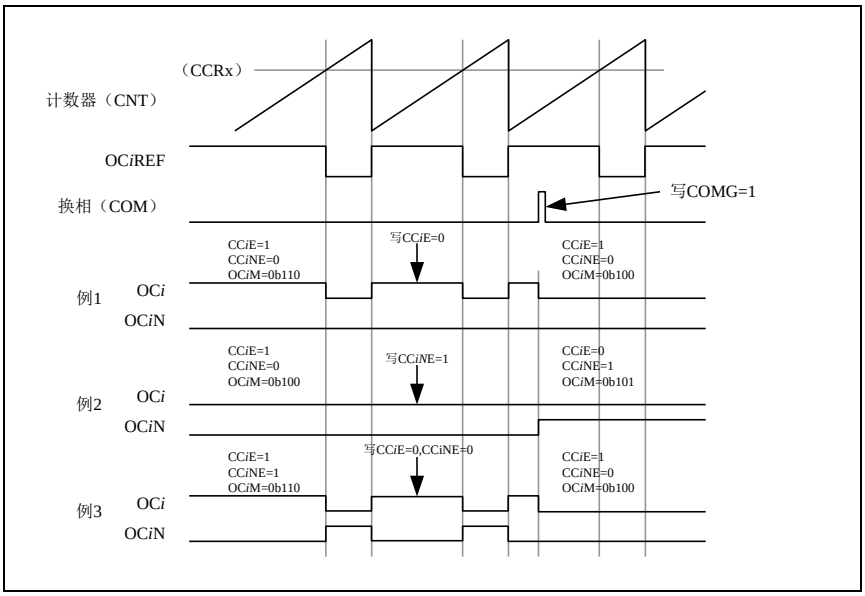
注：当只使能 OCiN（CCiE=0, CCiNE=1）时，它不会反相，而当 OCiREF 变高时立即有效。例如，如果 CCiNP=0，则 OCiN=OCiREF。另一方面，当 OCi 和 OCiN 都被使能时（CCiE=CCiNE=1），当 OCiREF 为高时 OCi 有效；而 OCiN 相反，当 OCiREF 低时 OCiN 变为有效。

针对马达控制的六步 PWM 输出

当在一个通道上需要互补输出时，预装载位有 OCiM、CCiE 和 CCiNE。在发生 COM 换相事件时，这些预装载位被传送到影子寄存器位。这样你就可以预先设置好下一步骤配置，并在同一个时刻同时修更改所有通道的配置。COM 可以通过设置 PWMA_EGR 寄存器的 COMG 位由软件产生，或在 TRGI 上升沿由硬件产生。

下图显示当发生 COM 事件时，三种不同配置下 OCx 和 OCxN 输出。

产生六步 PWM，使用 COM 的例子（OSSR=1）



21.5.8 使用刹车功能（PWMFLT）

刹车功能常用于马达控制中。当使用刹车功能时，依据相应的控制位（PWMA_BKR 寄存器中的 MOE、OSSI 和 OSSR 位），输出使能信号和无效电平都会被修改。

系统复位后，刹车电路被禁止，MOE 位为低。设置 PWMA_BKR 寄存器中的 BKE 位可以使能刹车功能。刹车输入信号的极性可以通过配置同一个寄存器中的 BKP 位选择。BKE 和 BKP 可以被同时修改。

MOE 下降沿相对于时钟模块可以是异步的，因此在实际信号（作用在输出端）和同步控制位（在 PWMA_BKR 寄存器中）之间设置了一个再同步电路。这个再同步电路会在异步信号和同步信号之间产生延迟。特别的，如果当它为低时写 MOE=1，则读出它之前必须先插入一个延时（空指令）才能读到正确的值。这是因为写入的是异步信号而读的是同步信号。

当发生刹车时（在刹车输入端出现选定的电平），有下述动作：

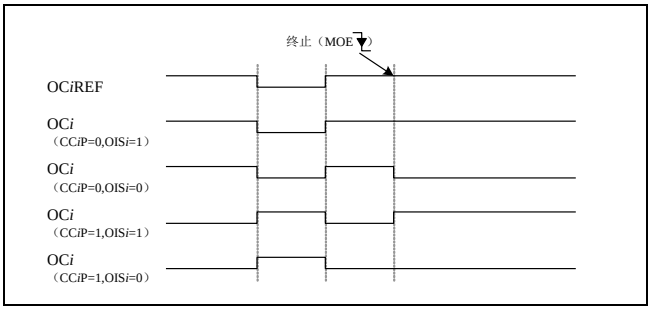
- MOE 位被异步地清除，将输出置于无效状态、空闲状态或者复位状态（由 OSSI 位选择）。这个特性在 MCU 的振荡器关闭时依然有效。
- 一旦 MOE=0，每一个输出通道输出由 PWMA_OISR 寄存器的 OISi 位设定的电平。如果 OSSI=0，则定时器不再控制输出使能信号，否则输出使能信号始终为高。
- 当使用互补输出时：
  - 输出首先被置于复位状态即无效的状态（取决于极性）。这是异步操作，即使定时器没有时钟时，此功能也有效。
  - 如果定时器的时钟依然存在，死区生成器将会重新生效，在死区之后根据 OISi 和 OISiN 位指示的电平驱动输出端口。即使在这种情况下，OCi 和 OCiN 也不能被同时驱动到有效的电平。注：因为重新同步 MOE，死区时间比通常情况下长一些（大约 2 个时钟周期）。
- 如果设置了 PWMA_IER 寄存器的 BIE 位，当刹车状态标志（PWMA_SR1 寄存器中的 BIF 位）为 1 时，则产生一个中断。
- 如果设置了 PWMA_BKR 寄存器中的 AOE 位，在下一个更新事件 UEV 时 MOE 位被自动置位。例如这可以用来进行波形控制，否则，MOE 始终保持低直到被再次置 1。这个特性可以被用在安全方面，你可以把刹车输入连到电源驱动的报警输出、热敏传感器或者其他安全器件上。

注：刹车输入为电平有效。所以，当刹车输入有效时，不能同时（自动地或者通过软件）设置 MOE。同时，状态标志 BIF 不能被清除。

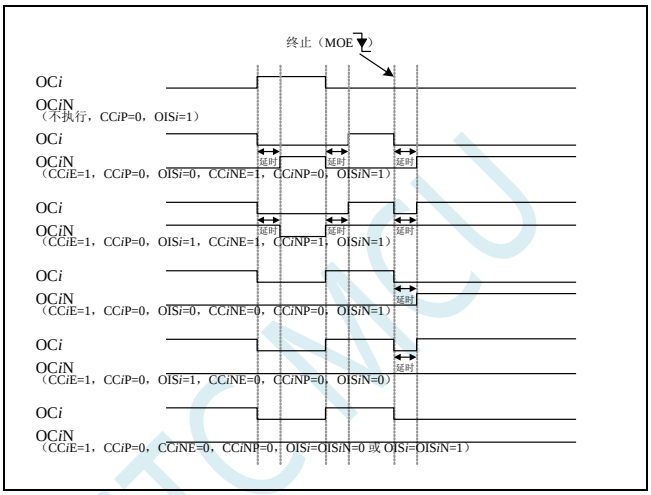
刹车由 BRK 输入产生，它的有效极性是可编程的，且由 PWMA_BKR 寄存器的 BKE 位开启或禁止。除了刹车输入和输出管理，刹车电路中还实现了写保护以保证应用程序的安全。它允许用户冻结几个配

置参数（OCi 极性和被禁止时的状态，OCiM 配置，刹车使能和极性）。用户可以通过 PWMA_BKR 寄存器的 LOCK 位，从三种级别的保护中选择一种。在 MCU 复位后 LOCK 位域只能被修改一次。

刹车响应的输出（不带互补输出的通道）



带互补输出的刹车响应的输出（PWMA 互补输出）



21.5.9　在外部事件发生时清除 OCiREF 信号

对于一个给定的通道，在 ETRF 输入端（设置 PWMA_CCMRi 寄存器中对应的 OCiCE 位为'1'）的高电平能够把 OCiREF 信号拉低，OCiREF 信号将保持为低直到发生下一次的更新事件 UEV。该功能只能用于输出比较模式和 PWM 模式，而不能用于强制模式。

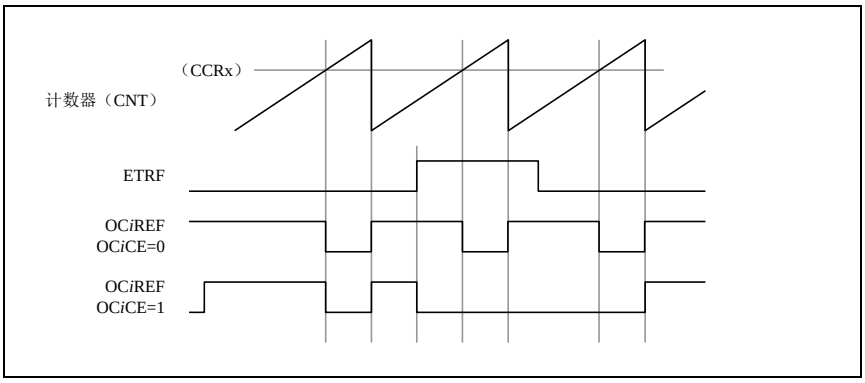
例如，OCiREF 信号可以联到一个比较器的输出，用于控制电流。这时，ETR 必须配置如下：

1. 外部触发预分频器必须处于关闭：PWMA_ETR 寄存器中的 ETPS[1:0]=00。
2. 必须禁止外部时钟模式 2：PWMA_ETR 寄存器中的 ECE=0。
3. 外部触发极性（ETP）和外部触发滤波器（ETF）可以根据需要配置。

下图显示了当 ETRF 输入变为高时，对应不同 OCiCE 的值，OCiREF 信号的动作。

在这个例子中，定时器 PWMA 被置于 PWM 模式。

ETR 清除 PWMA 的 OCiREF



21.5.10 编码器接口模式

编码器接口模式一般用于马达控制。

选择编码器接口模式的方法是：

- 如果计数器只在 TI2 的边沿计数，则置 PWMA_SMCR 寄存器中的 SMS=001；
- 如果只在 TI1 边沿计数，则置 SMS=010；
- 如果计数器同时在 TI1 和 TI2 边沿计数，则置 SMS=011。

通过设置 PWMA_CCER1 寄存器中的 CC1P 和 CC2P 位，可以选择 TI1 和 TI2 极性；如果需要，还可以对输入滤波器编程。

两个输入 TI1 和 TI2 被用来作为增量编码器的接口。假定计数器已经启动（PWMA_CR1 寄存器中的 CEN=1），则计数器在每次 TI1FP1 或 TI2FP2 上产生有效跳变时计数。TI1FP1 和 TI2FP2 是 TI1 和 TI2 在通过输入滤波器和极性控制后的信号。如果没有滤波和极性变换，则 TI1FP1=TI1，TI2FP2=TI2。根据两个输入信号的跳变顺序，产生了计数脉冲和方向信号。依据两个输入信号的跳变顺序，计数器向上或向下计数，同时硬件对 PWMA_CR1 寄存器的 DIR 位进行相应的设置。不管计数器是依靠 TI1 计数、依靠 TI2 计数或者同时依靠 TI1 和 TI2 计数，在任一输入端（TI1 或者 TI2）的跳变都会重新计算 DIR 位。

编码器接口模式基本上相当于使用了一个带有方向选择的外部时钟。这意味着计数器只在 0 到 PWMA_ARR 寄存器的自动装载值之间连续计数（根据方向，或是 0 到 ARR 计数，或是 ARR 到 0 计数）。所以在开始计数之前必须配置 PWMA_ARR。在这种模式下捕获器、比较器、预分频器、重复计数器、触发输出特性等仍工作如常。编码器模式和外部时钟模式 2 不兼容，因此不能同时操作。

编码器接口模式下，计数器依照增量编码器的速度和方向被自动的修改，因此计数器的内容始终指示着编码器的位置，计数方向与相连的传感器旋转的方向对应。

下表列出了所有可能的组合（假设 TI1 和 TI2 不同时变换）。

计数方向与编码器信号的关系

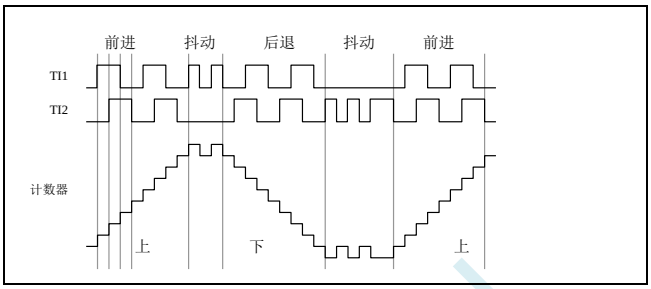
有效边沿	相对信号的电平 （TI1FP1 对应 TI2, TI2FP2 对应 TI1）	TI1FP1 信号		TI2FP2 信号	
		上升	下降	上升	下降
仅在 TI1 计数	高	向下计数	向上计数	不计数	不计数
	低	向上计数	向下计数	不计数	不计数
仅在 TI2 计数	高	不计数	不计数	向上计数	向下计数
	低	不计数	不计数	向下计数	向上计数
在 TI1 和 TI2 上计数	高	向下计数	向上计数	向上计数	向下计数
	低	向上计数	向下计数	向下计数	向上计数

一个外部的增量编码器可以直接与 MCU 连接而不需要外部接口逻辑。但是，一般使用比较器将编码器的差分输出转换成数字信号，这大大增加了抗噪声干扰能力。编码器输出的第三个信号表示机械零点，可以把它连接到一个外部中断输入并触发一个计数器复位。

下面是一个计数器操作的实例，显示了计数信号的产生和方向控制。它还显示了当选择了双边沿时，输入抖动是如何被抑制的；抖动可能会在传感器的位置靠近一个转换点时产生。在这个例子中，我们假定配置如下：

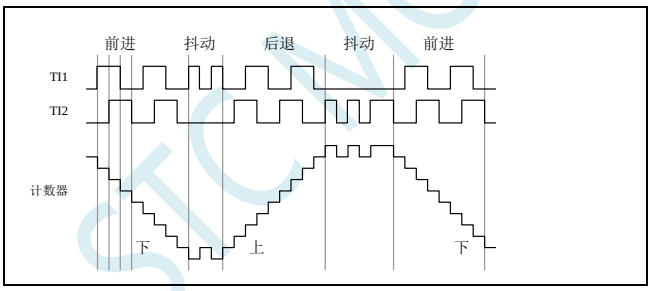
- CC1S=01（PWMA_CCMR1 寄存器，IC1FP1 映射到 TI1）
- CC2S=01（PWMA_CCMR2 寄存器，IC2FP2 映射到 TI2）
- CC1P=0（PWMA_CCER1 寄存器，IC1 不反相，IC1=TI1）
- CC2P=0（PWMA_CCER1 寄存器，IC2 不反相，IC2=TI2）
- SMS=011（PWMA_SMCR 寄存器，所有的输入均在上升沿和下降沿有效）.
- CEN=1（PWMA_CR1 寄存器，计数器使能）

编码器模式下的计数器操作实例



下图为当 IC1 极性反相时计数器的操作实例（CC1P=1，其他配置与上例相同）

IC1 反相的编码器接口模式实例



当定时器配置成编码器接口模式时，提供传感器当前位置的信息。使用另外一个配置在捕获模式下的定时器测量两个编码器事件的间隔，可以获得动态的信息（速度、加速度、减速度）。指示机械零点的编码器输出可被用做此目的。根据两个事件间的间隔，可以按照一定的时间间隔读出计数器。如果可能的话，你可以把计数器的值锁存到第三个输入捕获寄存器（捕获信号必须是周期的并且可以由另一个定时器产生）。

21.6 中断

PWMA/PWMB 各有 8 个中断请求源：

- 刹车中断
- 触发中断
- COM 事件中断
- 输入捕捉/输出比较 4 中断
- 输入捕捉/输出比较 3 中断
- 输入捕捉/输出比较 2 中断
- 输入捕捉/输出比较 1 中断
- 更新事件中断（如：计数器上溢，下溢及初始化）

为了使用中断特性，对每个被使用的中断通道，设置 PWM_IER/PWMB_IER 寄存器中相应的中断使能位：即 BIE，TIE，COMIE，CCiIE，UIE 位。通过设置 PWMA_EGR/PWMB_EGR 寄存器中的相应位，也可以用软件产生上述各个中断源。

STC MCU

21.7 PWMA/PWMB 寄存器描述

21.7.1 输出使能寄存器（PWMx_ENO）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ENO	FEB1H	ENO4N	ENO4P	ENO3N	ENO3P	ENO2N	ENO2P	ENO1N	ENO1P
PWMB_ENO	FEB5H	-	ENO8P	-	ENO7P	-	ENO6P	-	ENO5P

ENO8P：PWM8 输出控制位

0：禁止 PWM8 输出

1：使能 PWM8 输出

ENO7P：PWM7 输出控制位

0：禁止 PWM7 输出

1：使能 PWM7 输出

ENO6P：PWM6 输出控制位

0：禁止 PWM6 输出

1：使能 PWM6 输出

ENO5P：PWM5 输出控制位

0：禁止 PWM5 输出

1：使能 PWM5 输出

ENO4N：PWM4N 输出控制位

0：禁止 PWM4N 输出

1：使能 PWM4N 输出

ENO4P：PWM4P 输出控制位

0：禁止 PWM4P 输出

1：使能 PWM4P 输出

ENO3N：PWM3N 输出控制位

0：禁止 PWM3N 输出

1：使能 PWM3N 输出

ENO3P：PWM3P 输出控制位

0：禁止 PWM3P 输出

1：使能 PWM3P 输出

ENO2N：PWM2N 输出控制位

0：禁止 PWM2N 输出

1：使能 PWM2N 输出

ENO2P：PWM2P 输出控制位

0：禁止 PWM2P 输出

1：使能 PWM2P 输出

ENO1N：PWM1N 输出控制位

0：禁止 PWM1N 输出

1：使能 PWM1N 输出

ENO1P：PWM1P 输出控制位

0：禁止 PWM1P 输出

1：使能 PWM1P 输出

STC MCU

21.7.2 输出附加使能寄存器（PWMx_IOAUX）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_IOAUX	FEB3H	AUX4N	AUX4P	AUX3N	AUX3P	AUX2N	AUX2P	AUX1N	AUX1P
PWMB_IOAUX	FEB7H	-	AUX8P	-	AUX7P	-	AUX6P	-	AUX5P

AUX8P：PWM8 输出附加控制位

0：PWM8 的输出直接由 ENO8P 控制

1：PWM8 的输出由 ENO8P 和 PWMB_BKR 共同控制

AUX7P：PWM7 输出附加控制位

0：PWM7 的输出直接由 ENO7P 控制

1：PWM7 的输出由 ENO7P 和 PWMB_BKR 共同控制

AUX6P：PWM6 输出附加控制位

0：PWM6 的输出直接由 ENO6P 控制

1：PWM6 的输出由 ENO6P 和 PWMB_BKR 共同控制

AUX5P：PWM5 输出附加控制位

0：PWM5 的输出直接由 ENO5P 控制

1：PWM5 的输出由 ENO5P 和 PWMB_BKR 共同控制

AUX4N：PWM4N 输出附加控制位

0：PWM4N 的输出直接由 ENO4N 控制

1：PWM4N 的输出由 ENO4N 和 PWMA_BKR 共同控制

AUX4P：PWM4P 输出附加控制位

0：PWM4P 的输出直接由 ENO4P 控制

1：PWM4P 的输出由 ENO4P 和 PWMA_BKR 共同控制

AUX3N：PWM3N 输出附加控制位

0：PWM3N 的输出直接由 ENO3N 控制

1：PWM3N 的输出由 ENO3N 和 PWMA_BKR 共同控制

AUX3P：PWM3P 输出附加控制位

0：PWM3P 的输出直接由 ENO3P 控制

1：PWM3P 的输出由 ENO3P 和 PWMA_BKR 共同控制

AUX2N：PWM2N 输出附加控制位

0：PWM2N 的输出直接由 ENO2N 控制

1：PWM2N 的输出由 ENO2N 和 PWMA_BKR 共同控制

AUX2P：PWM2P 输出附加控制位

0：PWM2P 的输出直接由 ENO2P 控制

1：PWM2P 的输出由 ENO2P 和 PWMA_BKR 共同控制

AUX1N：PWM1N 输出附加控制位

0：PWM1N 的输出直接由 ENO1N 控制

1：PWM1N 的输出由 ENO1N 和 PWMA_BKR 共同控制

AUX1P：PWM1P 输出附加控制位

0：PWM1P 的输出直接由 ENO1P 控制

1：PWM1P 的输出由 ENO1P 和 PWMA_BKR 共同控制

21.7.3 控制寄存器 1（PWMx_CR1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0

PWMA_CR1	FEC0H	ARPEA	CMSA[1:0]	DIRA	OPMA	URSA	UDISA	CENA
PWMB_CR1	FEE0H	ARPEB	CMSB[1:0]	DIRB	OPMB	URSB	UDISB	CENB

ARPEn：自动预装载允许位（n=A,B）

0：PWMn_ARR 寄存器没有缓冲，它可以被直接写入

1：PWMn_ARR 寄存器由预装载缓冲器缓冲

CMSn[1:0]：选择对齐模式（n= A,B）

CMSn[1:0]	对齐模式	说明
00	边沿对齐模式	计数器依据方向位（DIR）向上或向下计数
01	中央对齐模式1	计数器交替地向上和向下计数。 配置为输出的通道的输出比较中断标志位， 只在计数器向下计数时被置1。
10	中央对齐模式2	计数器交替地向上和向下计数。 配置为输出的通道的输出比较中断标志位， 只在计数器向上计数时被置1。
11	中央对齐模式3	计数器交替地向上和向下计数。 配置为输出的通道的输出比较中断标志位， 在计数器向上和向下计数时均被置1。

注 1：在计数器开启时（CEN=1），不允许从边沿对齐模式转换到中央对齐模式。

注 2：在中央对齐模式下，编码器模式（SMS=001，010，011）必须被禁止。

DIRn：计数器的计数方向（n= A,B）

0：计数器向上计数；

1：计数器向下计数。

注：当计数器配置为中央对齐模式或编码器模式时，该位为只读。

OPMn：单脉冲模式（n= A,B）

0：在发生更新事件时，计数器不停止；

1：在发生下一次更新事件时，清除 CEN 位，计数器停止。

URSn：更新请求源（n= A,B）

0：如果 UDIS 允许产生更新事件，则下述任一事件产生一个更新中断：

- 寄存器被更新（计数器上溢/下溢）
- 软件设置 UG 位
- 时钟/触发控制器产生的更新

1：如果 UDIS 允许产生更新事件，则只有当下列事件发生时才产生更新中断，并 UIF 置 1：

- 寄存器被更新（计数器上溢/下溢）

UDISn：禁止更新（n= A,B）

0：一旦下列事件发生，产生更新（UEV）事件：

- 计数器溢出/下溢
- 产生软件更新事件
- 时钟/触发模式控制器产生的硬件复位 被缓存的寄存器被装入它们的预装载值。

1：不产生更新事件，影子寄存器（ARR、PSC、CCRx）保持它们的值。如果设置了 UG 位或时钟/触发控制器发出了一个硬件复位，则计数器和预分频器被重新初始化。

CENn：允许计数器（n= A,B）

0：禁止计数器；

1：使能计数器。

注：在软件设置了 CEN 位后，外部时钟、门控模式和编码器模式才能工作。然而触发模式可以自动地通过硬件设置 CEN 位。

21.7.4 控制寄存器 2（PWMx_CR2），及实时触发 ADC

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CR2	FEC1H	TI1S	MMSA[2:0]			-	COMSA	-	CCPCA
PWMB_CR2	FEE1H	TI5S	MMSB[2:0]			-	COMSB	-	CCPCB

TI1S：第一组 PWM/PWMA 的 TI1 选择

0：PWM1P 输入管脚连到 TI1（数字滤波器的输入）；

1：PWM1P、PWM2P 和 PWM3P 管脚经异或后连到第一组 PWM 的 TI1。

TI5S：第二组 PWM/PWMB 的 TI5 选择

0：PWM5 输入管脚连到 TI5（数字滤波器的输入）；

1：PWM5、PWM6 和 PWM7 管脚经异或后连到第二组 PWM 的 TI5。

MMSA[2:0]：主模式选择

MMSA[2:0]	主模式	说明
000	复位	PWMA_EGR寄存器的UG位被用于作为触发输出（TRGO）。如果触发输入（时钟/触发控制器配置为复位模式）产生复位，则TRGO上的信号相对实际的复位会有一个延迟
001	使能	计数器使能信号被用于作为触发输出（TRGO）。其用于启动ADC，以便控制在一段时间内使能ADC。计数器使能信号是通过CEN控制位和门控模式下的触发输入信号的逻辑或产生。除非选择了主/从模式，当计数器使能信号受控于触发输入时，TRGO上会有一个延迟。 注：当需要使用PWM触发ADC转换时，需要先设置ADC_CONTR寄存器中的ADC_POWER、ADC_CHS以及ADC_EPWMT，当PWM产生TRGO内部信号时，系统会自动设置ADC_START来启动AD转换。详细使用请参考范例程序“使用PWM的CEN启动PWMA定时器，实时触发ADC”
010	更新	更新事件被选为触发输出（TRGO）
011	比较脉冲	一旦发生一次捕获或一次比较成功，当CC1IF标志被置1时，触发输出送出一个正脉冲（TRGO）
100	比较	OC1REF信号被用于作为触发输出（TRGO）
101	比较	OC2REF信号被用于作为触发输出（TRGO）
110	比较	OC3REF信号被用于作为触发输出（TRGO）
111	比较	OC4REF信号被用于作为触发输出（TRGO）

MMSB[2:0]：主模式选择

MMSB[2:0]	主模式	说明
000	复位	PWMB_EGR寄存器的UG位被用于作为触发输出（TRGO）。如果触发输入（时钟/触发控制器配置为复位模式）产生复

		位，则TRGO上的信号相对实际的复位会有一个延迟
001	使能	计数器使能信号被用于作为触发输出（TRGO）。其用于启动多个PWM，以便控制在一段时间内使能从PWM。计数器使能信号是通过CEN控制位和门控模式下的触发输入信号的逻辑或产生。除非选择了主/从模式，当计数器使能信号受控于触发输入时，TRGO上会有一个延迟。
010	更新	更新事件被选为触发输出（TRGO）
011	比较脉冲	一旦发生一次捕获或一次比较成功，当CC5IF标志被置1时，触发输出送出一个正脉冲（TRGO）
100	比较	OC5REF信号被用于作为触发输出（TRGO）
101	比较	OC6REF信号被用于作为触发输出（TRGO）
110	比较	OC7REF信号被用于作为触发输出（TRGO）
111	比较	OC8REF信号被用于作为触发输出（TRGO）

注：只有第一组 PWM 的 TRGO 可用于触发启动 ADC

注：只有第二组 PWM 的 TRGO 可用于第一组 PWM 的 ITR2

COMSn：捕获/比较控制位的更新控制选择（n=A,B）

0：当 CCPCn=1 时，只有在 COMG 位置 1 的时候这些控制位才被更新

1：当 CCPCn=1 时，只有在 COMG 位置 1 或 TRGI 发生上升沿的时候这些控制位才被更新

CCPCn：捕获/比较预装载控制位（n= A,B）

0：CCIE，CCINE，CCiP，CCiNP 和 OCIM 位不是预装载的

1：CCIE，CCINE，CCiP，CCiNP 和 OCIM 位是预装载的；设置该位后，它们只在设置了 COMG 位后被更新。

注：该位只对具有互补输出的通道起作用。

21.7.5 从模式控制寄存器(PWMx_SMCR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_SMCR	FEC2H	MSMA	TSA[2:0]			-	SMSA[2:0]		
PWMB_SMCR	FEE2H	MSMB	TSB[2:0]			-	SMSB[2:0]		

MSMn：主/从模式（n= A,B）

0：无作用

1：触发输入（TRGI）上的事件被延迟了，以允许 PWMn 与它的从 PWM 间的完美同步（通过 TRGO）

TSA[2:0]：触发源选择

TSA[2:0]	触发源
000	–
001	–
010	内部触发ITR2
011	–
100	TI1的边沿检测器（TI1F_ED）
101	滤波后的定时器输入1（TI1FP1）
110	滤波后的定时器输入2（TI2FP2）

111	外部触发输入（ETRF）

TSB[2:0]：触发源选择

TSB[2:0]	触发源
000	-
001	-
010	-
011	-
100	TI5的边沿检测器（TI5F_ED）
101	滤波后的定时器输入1（TI5FP5）
110	滤波后的定时器输入2（TI5FP6）
111	外部触发输入（ETRF）

注：这些位只能在 SMS=000 时被改变，以避免在改变时产生错误的边沿检测。

SMSA[2:0]：时钟/触发/从模式选择

SMSA[2:0]	功能	说明
000	内部时钟模式	如果CEN=1，则预分频器直接由内部时钟驱动
001	编码器模式1	根据TI1FP1的电平，计数器在TI2FP2的边沿向上/下计数
010	编码器模式2	根据TI2FP2的电平，计数器在TI1FP1的边沿向上/下计数
011	编码器模式3	根据另一个输入的电平，计数器在TI1FP1和TI2FP2的边沿向上/下计数
100	复位模式	在选中的触发输入（TRGI）的上升沿时重新初始化计数器，并且产生一个更新 寄存器的信号
101	门控模式	当触发输入（TRGI）为高时，计数器的时钟开启。一旦触发输入变为低，则计 数器停止（但不复位）。计数器的启动和停止都是受控的
110	触发模式	计数器在触发输入TRGI的上升沿启动（但不复位），只有计数器的启动是受控 的
111	外部时钟模式1	选中的触发输入（TRGI）的上升沿驱动计数器。 注：如果TI1F_ED被选为触发输入（TS=100）时，不要使用门控模式。这是因为TI1F_ED在每次TI1F变化时只是输出一个脉冲，然而门控模式是要检查触发输入的电平

SMSB[2:0]：时钟/触发/从模式选择

SMSB[2:0]	功能	说明
000	内部时钟模式	如果CEN=1，则预分频器直接由内部时钟驱动
001	编码器模式1	根据TI5FP5的电平，计数器在TI6FP6的边沿向上/下计数
010	编码器模式2	根据TI6FP6的电平，计数器在TI5FP5的边沿向上/下计数
011	编码器模式3	根据另一个输入的电平，计数器在TI5FP5和TI6FP6的边沿向上/下计数
100	复位模式	在选中的触发输入（TRGI）的上升沿时重新初始化计数器，

		并且产生一个更新 寄存器的信号
101	门控模式	当触发输入（TRGI）为高时，计数器的时钟开启。一旦触发输入变为低，则计 数器停止（但不复位）。计数器的启动和停止都是受控的
110	触发模式	计数器在触发输入TRGI的上升沿启动（但不复位），只有计数器的启动是受控 的
111	外部时钟模式1	选中的触发输入（TRGI）的上升沿驱动计数器。 注：如果TI5F_ED被选为触发输入（TS=100）时，不要使用门控模式。这是因为TI5F_ED在每次TI5F变化时只是输出一个脉冲，然而门控模式是要检查触发输入的电平

21.7.6 外部触发寄存器(PWMx_ETR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ETR	FEC3H	ETP1	ECEA	ETPSA[1:0]		ETFA[3:0]			
PWMB_ETR	FEE3H	ETP2	ECEB	ETPSB[1:0]		ETFB[3:0]			

ETPn：外部触发 ETR 的极性（n= A,B）

0：高电平或上升沿有效

1：低电平或下降沿有效

ECEn：外部时钟使能（n= A,B）

0：禁止外部时钟模式 2

1：使能外部时钟模式 2，计数器的时钟为 ETRF 的有效沿。

注 1：ECE 置 1 的效果与选择把 TRGI 连接到 ETRF 的外部时钟模式 1 相同（PWMn_SMCR 寄存器中，SMS=111，TS=111）。

注 2：外部时钟模式 2 可与下列模式同时使用：触发标准模式；触发复位模式；触发门控模式。但是，此时 TRGI 决不能与 ETRF 相连（PWMn_SMCR 寄存器中，TS 不能为 111）。

注 3：外部时钟模式 1 与外部时钟模式 2 同时使能，外部时钟输入为 ETRF。

ETPSn：外部触发预分频器外部触发信号 EPRP 的频率最大不能超过 fMASTER/4。可用预分频器来降低 ETRP 的频率，当 EPRP 的频率很高时，它非常有用：（n= A,B）

00：预分频器关闭

01：EPRP 的频率/2

02：EPRP 的频率/4

03：EPRP 的频率/8

ETFn[3:0]：外部触发滤波器选择，该位域定义了 ETRP 的采样频率及数字滤波器长度。（n= A,B）

ETFn[3:0]	时钟数	ETF[3:0]	时钟数
0000	1	1000	48
0001	2	1001	64
0010	4	1010	80
0011	8	1011	96
0100	12	1100	128
0101	16	1101	160

0110	24	1110	192
0111	32	1111	256

21.7.7 中断使能寄存器(PWMx_IER)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_IER	FEC4H	BIEA	TIEA	COMIEA	CC4IE	CC3IE	CC2IE	CC1IE	UIEA
PWMB_IER	FEE4H	BIEB	TIEB	COMIEB	CC8IE	CC7IE	CC6IE	CC5IE	UIEB

BIEn：允许刹车中断（n= A,B）

0：禁止刹车中断；

1：允许刹车中断。

TIE：触发中断使能（n= A,B）

0：禁止触发中断；

1：使能触发中断。

COMIE：允许 COM 中断（n= A,B）

0：禁止 COM 中断；

1：允许 COM 中断。

CCnIE：允许捕获/比较 n 中断（n=1,2,3,4,5,6,7,8）

0：禁止捕获/比较 n 中断；

1：允许捕获/比较 n 中断。

UIEn：允许更新中断（n= A,B）

0：禁止更新中断；

1：允许更新中断。

21.7.8 状态寄存器 1(PWMx_SR1)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_SR1	FEC5H	BIFA	TIFA	COMIFA	CC4IF	CC3IF	CC2IF	CC1IF	UIFA
PWMB_SR1	FEE5H	BIFB	TIFB	COMIFB	CC8IF	CC7IF	CC6IF	CC5IF	UIFB

BIFn：刹车中断标记。一旦刹车输入有效，由硬件对该位置 1。如果刹车输入无效，则该位可由软件清 0。（n= A,B）

0：无刹车事件产生

1：刹车输入上检测到有效电平

TIFn：触发器中断标记。当发生触发事件时由硬件对该位置 1。由软件清 0。（n= A,B）

0：无触发器事件产生

1：触发中断等待响应

COMIFn：COM 中断标记。一旦产生 COM 事件该位由硬件置 1。由软件清 0。（n= A,B）

0：无 COM 事件产生

1：COM 中断等待响应

CC8IF：捕获/比较8中断标记，参考CC1IF描述

CC7IF：捕获/比较7中断标记，参考CC1IF描述

CC6IF：捕获/比较6中断标记，参考CC1IF描述

CC5IF：捕获/比较5中断标记，参考CC1IF描述

CC4IF：捕获/比较4中断标记，参考CC1IF描述

CC3IF：捕获/比较3中断标记，参考CC1IF描述

CC2IF：捕获/比较2中断标记，参考CC1IF描述

CC1IF：捕获/比较1中断标记。

如果通道CC1配置为输出模式：

当计数器值与比较值匹配时该位由硬件置1，但在中心对称模式下除外。它由软件清0。

0：无匹配发生；

1：PWMA_CNT 的值与 PWMA_CCR1 的值匹配。

注：在中心对称模式下，当计数器值为 0 时，向上计数，当计数器值为 ARR 时，向下计数（它从 0 向上计数到 ARR-1，再由 ARR 向下计数到 1）。因此，对所有的 SMS 位值，这两个值都不置标记。但是，如果 CCR1>ARR，则当 CNT 达到 ARR 值时，CC1IF 置 1。

如果通道CC1配置为输入模式：

当捕获事件发生时该位由硬件置1，它由软件清0或通过读PWMA_CCR1L清0。

0：无输入捕获产生

1：计数器值已被捕获至 PWMA_CCR1

UIFn：更新中断标记 当产生更新事件时该位由硬件置 1。它由软件清 0。（n= A,B）

0：无更新事件产生

1：更新事件等待响应。当寄存器被更新时该位由硬件置 1

- 若 PWMn_CR1 寄存器的 UDIS=0，当计数器上溢或下溢时
- 若 PWMn_CR1 寄存器的 UDIS=0、URS=0，当设置 PWMn_EGR 寄存器的 UG 位软件对计数器 CNT 重新初始化时
- 若 PWMn_CR1 寄存器的 UDIS=0、URS=0，当计数器 CNT 被触发事件重新初始化时

21.7.9 状态寄存器 2(PWMx_SR2)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_SR2	FEC6H	-	-	-	CC4OF	CC3OF	CC2OF	CC1OF	-
PWMB_SR2	FEE6H	-	-	-	CC8OF	CC7OF	CC6OF	CC5OF	-

CC8OF：捕获/比较8重复捕获标记。参见CC1OF描述。

CC7OF：捕获/比较7重复捕获标记。参见CC1OF描述。

CC6OF：捕获/比较6重复捕获标记。参见CC1OF描述。

CC5OF：捕获/比较5重复捕获标记。参见CC1OF描述。

CC4OF：捕获/比较4重复捕获标记。参见CC1OF描述。

CC3OF：捕获/比较3重复捕获标记。参见CC1OF描述。

CC2OF：捕获/比较2重复捕获标记。参见CC1OF描述。

CC1OF：捕获/比较1重复捕获标记。仅当相应的通道被配置为输入捕获时，该标记可由硬件置1。写0可清除该位。

0：无重复捕获产生；

1：计数器的值被捕获到 PWMA_CCR1 寄存器时，CC1IF 的状态已经为 1。

21.7.10 事件产生寄存器（PWMx_EGR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_EGR	FEC7H	BGA	TGA	COMGA	CC4G	CC3G	CC2G	CC1G	UGA
PWMB_EGR	FEE7H	BGB	TGB	COMGB	CC8G	CC7G	CC6G	CC5G	UGB

BGn：产生刹车事件。该位由软件置 1，用于产生一个刹车事件，由硬件自动清 0（n= A,B）

0：无动作

1：产生一个刹车事件。此时 MOE=0、BIF=1，若开启对应的中断（BIE=1），则产生相应的中断

TGn：产生触发事件。该位由软件置 1，用于产生一个触发事件，由硬件自动清 0（n= A,B）

0：无动作

1：TIF=1，若开启对应的中断（TIE=1），则产生相应的中断

COMGn：捕获/比较事件，产生控制更新。该位由软件置 1，由硬件自动清 0（n= A,B）

0：无动作

1：CCPC=1，允许更新 CCIE、CCINE、CCiP，CCiNP，OCIM 位。

注：该位只对拥有互补输出的通道有效

CC8G：产生捕获/比较 8 事件。参考 CC1G 描述

CC7G：产生捕获/比较 7 事件。参考 CC1G 描述

CC6G：产生捕获/比较 6 事件。参考 CC1G 描述

CC5G：产生捕获/比较 5 事件。参考 CC1G 描述

CC4G：产生捕获/比较 4 事件。参考 CC1G 描述

CC3G：产生捕获/比较 3 事件。参考 CC1G 描述

CC2G：产生捕获/比较 2 事件。参考 CC1G 描述

CC1G：产生捕获/比较 1 事件。产生捕获/比较 1 事件。该位由软件置 1，用于产生一个捕获/比较事件，由硬件自动清 0。

0：无动作；

1：在通道 CC1 上产生一个捕获/比较事件。

若通道 CC1 配置为输出：设置 CC1IF=1，若开启对应的中断，则产生相应的中断。

若通道 CC1 配置为输入：当前的计数器值被捕获至 PWMA_CCR1 寄存器，设置 CC1IF=1，若开启对应的中断，则产生相应的中断。若 CC1IF 已经为 1，则设置 CC1OF=1。

UGn：产生更新事件 该位由软件置 1，由硬件自动清 0。（n= A,B）

0：无动作；

1：重新初始化计数器，并产生一个更新事件。

注意预分频器的计数器也被清 0（但是预分频系数不变）。若在中心对称模式下或 DIR=0（向上计数）则计数器被清 0；若 DIR=1（向下计数）则计数器取 PWMn_ARR 的值。

21.7.11 捕获/比较模式寄存器 1（PWMx_CCMR1）

通道可用于捕获输入模式或比较输出模式，通道的方向由相应的 CCnS 位定义。该寄存器其它位的作用在输入和输出模式下不同。OCxx 描述了通道在输出模式下的功能，ICxx 描述了通道在输入模式下的功能。因此必须注意，同一个位在输出模式和输入模式下的功能是不同的。

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR1	FEC8H	OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]	
PWMB_CCMR1	FEE8H	OC5CE	OC5M[2:0]			OC5PE	OC5FE	CC5S[1:0]	

OCnCE：输出比较 n 清零使能。该位用于使能使用 PWMETI 引脚上的外部事件来清通道 n 的输出信号（OCnREF）（n=1,5）

0：OCnREF 不受 ETRF 输入的影响；

1：一旦检测到 ETRF 输入高电平，OCnREF=0。

OCnM[2:0]：输出比较 n 模式。该 3 位定义了输出参考信号 OCnREF 的动作，而 OCnREF 决定了 OCn

的值。OCnREF 是高电平有效，而 OCn 的有效电平取决于 CCnP 位。（n=1,5）

OCnM[2:0]	模式	说明
000	冻结	PWMn_CCR1与PWMn_CNT间的比较对OCnREF不起作用
001	匹配时设置通道n的输出为有效电平	当PWMn_CCR1=PWMn_CNT时，OCnREF输出高
010	匹配时设置通道n的输出为无效电平	当PWMn_CCR1=PWMn_CNT时，OCnREF输出低
011	翻转	当PWMn_CCR1=PWMn_CNT时，翻转OCnREF
100	强制为无效电平	强制OCnREF为低
101	强制为有效电平	强制OCnREF为高
110	PWM模式1	在向上计数时，当PWMn_CNT<PWMn_CCR1时OCnREF输出高，否则OCnREF输出低 在向下计数时，当PWMn_CNT>PWMn_CCR1时OCnREF输出低，否则OCnREF输出高
111	PWM模式2	在向上计数时，当PWMn_CNT<PWMn_CCR1时OCnREF输出低，否则OCnREF输出高 在向下计数时，当PWMn_CNT>PWMn_CCR1时OCnREF输出高，否则OCnREF输出低

注 1：一旦 LOCK 级别设为 3（PWMn_BKR 寄存器中的 LOCK 位）并且 CCnS=00（该通道配置成输出）则该位不能被修改。

注 2：在 PWM 模式 1 或 PWM 模式 2 中，只有当比较结果改变了或在输出比较模式中从冻结模式切换到 PWM 模式时，OCnREF 电平才改变。

注 3：在有互补输出的通道上，这些位是预装载的。如果 PWMn_CR2 寄存器的 CCPC=1，OCM 位只有在 COM 事件发生时，才从预装载位取新值。

OCnPE：输出比较 n 预装载使能（n=1,5）

0：禁止 PWMn_CCR1 寄存器的预装载功能，可随时写入 PWMn_CCR1 寄存器，并且新写入的数值立即起作用。

1：开启 PWMn_CCR1 寄存器的预装载功能，读写操作仅对预装载寄存器操作，PWMn_CCR1 的预装载值在更新事件到来时被加载至当前寄存器中。

注 1：一旦 LOCK 级别设为 3（PWMn_BKR 寄存器中的 LOCK 位）并且 CCnS=00（该通道配置成输出）则该位不能被修改。

注 2：为了操作正确，在 PWM 模式下必须使能预装载功能。但在单脉冲模式下（PWMn_CR1 寄存器的 OPM=1），它不是必须的。

OCnFE：输出比较 n 快速使能。该位用于加快 CC 输出对触发输入事件的响应。（n=1,5）

0：根据计数器与 CCRn 的值，CCn 正常操作，即使触发器是打开的。当触发器的输入有一个有效沿时，激活 CCn 输出的最小延时为 5 个时钟周期。

1：输入到触发器的有效沿的作用就象发生了一次比较匹配。因此，OC 被设置为比较电平而与比较结果无关。采样触发器的有效沿和 CC1 输出间的延时被缩短为 3 个时钟周期。OCFE 只在通道被配置成 PWMA 或 PWMB 模式时起作用。

CC1S[1:0]：捕获/比较 1 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC1S[1:0]	方向	输入脚

00	输出	
01	输入	IC1映射在TI1FP1上
10	输入	IC1映射在TI2FP1上
11	输入	IC1映射在TRC上。此模式仅工作在内部触发器输入被选中时（由PWMA_SMCR寄存器的TS位选择）

CC5S[1:0]：捕获/比较 5 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC5S[1:0]	方向	输入脚
00	输出	
01	输入	IC5映射在TI5FP5上
10	输入	IC5映射在TI6FP5上
11	输入	IC5映射在TRC上。此模式仅工作在内部触发器输入被选中时（由PWM5_SMCR寄存器的TS位选择）

注：CC1S 仅在通道关闭时（PWMA_CCER1 寄存器的 CC1E=0）才是可写的。

注：CC5S 仅在通道关闭时（PWM5_CCER1 寄存器的 CC5E=0）才是可写的。

STC MCU

通道配置为捕获输入模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR1	FEC8H	IC1F[3:0]				IC1PSC[1:0]		CC1S[1:0]	
PWMB_CCMR1	FEE8H	IC5F[3:0]				IC5PSC[1:0]		CC5S[1:0]	

ICnF[3:0]：输入捕获 n 滤波器选择，该位域定义了 TIn 的采样频率及数字滤波器长度。（n=1,5）

ICnF[3:0]	时钟数	ICnF[3:0]	时钟数
0000	1	1000	48
0001	2	1001	64
0010	4	1010	80
0011	8	1011	96
0100	12	1100	128
0101	16	1101	160
0110	24	1110	192
0111	32	1111	256

注：即使对于带互补输出的通道，该位域也是非预装载的，并且不会考虑 CCPC（PWMn_CR2 寄存器）的值

ICnPSC[1:0]：输入/捕获 n 预分频器。这两位定义了 CCn 输入（IC1）的预分频系数。（n=1,5）

00：无预分频器，捕获输入口上检测到的每一个边沿都触发一次捕获

01：每 2 个事件触发一次捕获

10：每 4 个事件触发一次捕获

11：每 8 个事件触发一次捕获

CC1S[1:0]：捕获/比较 1 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC1S[1:0]	方向	输入脚
00	输出	
01	输入	IC1映射在TI1FP1上
10	输入	IC1映射在TI2FP1上
11	输入	IC1映射在TRC上。此模式仅工作在内部触发器输入被选中时（由PWMA_SMCR寄存器的TS位选择）

CC5S[1:0]：捕获/比较 5 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC5S[1:0]	方向	输入脚
00	输出	
01	输入	IC5映射在TI5FP5上
10	输入	IC5映射在TI6FP5上
11	输入	IC5映射在TRC上。此模式仅工作在内部触发器输入被选中时（由PWM5_SMCR寄存器的TS位选择）

注：CC1S 仅在通道关闭时（PWMA_CCER1 寄存器的 CC1E=0）才是可写的。

注：CC5S 仅在通道关闭时（PWM5_CCER1 寄存器的 CC5E=0）才是可写的。

21.7.12 捕获/比较模式寄存器 2（PWMx_CCMR2）

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR2	FEC9H	OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]	
PWMB_CCMR2	FEE9H	OC6CE	OC6M[2:0]			OC6PE	OC6FE	CC6S[1:0]	

OCnCE：输出比较 n 清零使能。该位用于使能使用 PWMETI 引脚上的外部事件来清通道 n 的输出信号（OCnREF）（n=2,6）

0：OCnREF 不受 ETRF 输入的影响；

1：一旦检测到 ETRF 输入高电平，OCnREF=0。

OCnM[2:0]：输出比较 2 模式，参考 OC1M。（n=2,6）

OCnPE：输出比较 2 预装载使能，参考 OP1PE。（n=2,6）

CC2S[1:0]：捕获/比较 2 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC2S[1:0]	方向	输入脚
00	输出	
01	输入	IC2映射在TI2FP2上
10	输入	IC2映射在TI1FP2上
11	输入	IC2映射在TRC上。

CC6S[1:0]：捕获/比较 6 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC6S[1:0]	方向	输入脚
00	输出	
01	输入	IC6映射在TI6FP6上
10	输入	IC6映射在TI5FP6上
11	输入	IC6映射在TRC上。

通道配置为捕获输入模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR2	FEC9H	IC2F[3:0]				IC2PSC[1:0]		CC2S[1:0]	
PWMB_CCMR2	FEE9H	IC6F[3:0]				IC6PSC[1:0]		CC6S[1:0]	

ICnF[3:0]：输入捕获 n 滤波器选择，参考 IC1F。（n=2,6）

ICnPSC[1:0]：输入/捕获 n 预分频器，参考 IC1PSC。（n=2,6）

CC2S[1:0]：捕获/比较 2 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC2S[1:0]	方向	输入脚
00	输出	
01	输入	IC2映射在TI2FP2上
10	输入	IC2映射在TI1FP2上
11	输入	IC2映射在TRC上。

CC6S[1:0]：捕获/比较 6 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC6S[1:0]	方向	输入脚
00	输出	
01	输入	IC6映射在TI6FP6上

10	输入	IC6映射在TI5FP6上
11	输入	IC6映射在TRC上。

21.7.13 捕获/比较模式寄存器 3（PWMx_CCMR3）

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR3	FECAH	OC3CE	OC3M[2:0]			OC3PE	OC3FE	CC3S[1:0]	
PWMB_CCMR3	FEEAH	OC7CE	OC7M[2:0]			OC7PE	OC7FE	CC7S[1:0]	

OCnCE：输出比较 n 清零使能。该位用于使能使用 PWMETI 引脚上的外部事件来清通道 n 的输出信号（OCnREF）（n=3,7）

0：OCnREF 不受 ETRF 输入的影响；

1：一旦检测到 ETRF 输入高电平，OCnREF=0。

OCnM[2:0]：输出比较 3 模式，参考 OC1M。（n=3,7）

OCnPE：输出比较 3 预装载使能，参考 OP1PE。（n=3,7）

CC3S[1:0]：捕获/比较 3 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC3S[1:0]	方向	输入脚
00	输出	
01	输入	IC3映射在TI3FP3上
10	输入	IC3映射在TI4FP3上
11	输入	IC3映射在TRC上。

CC7S[1:0]：捕获/比较 7 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC7S[1:0]	方向	输入脚
00	输出	
01	输入	IC7映射在TI7FP7上
10	输入	IC7映射在TI8FP7上
11	输入	IC7映射在TRC上。

通道配置为捕获输入模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR3	FECAH	IC3F[3:0]				IC3PSC[1:0]		CC3S[1:0]	
PWMB_CCMR3	FEEAH	IC7F[3:0]				IC7PSC[1:0]		CC7S[1:0]	

ICnF[3:0]：输入捕获 n 滤波器选择，参考 IC1F。（n=3,7）

ICnPSC[1:0]：输入/捕获 n 预分频器，参考 IC1PSC。（n=3,7）

CC3S[1:0]：捕获/比较 3 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC3S[1:0]	方向	输入脚
00	输出	
01	输入	IC3映射在TI3FP3上
10	输入	IC3映射在TI4FP3上
11	输入	IC3映射在TRC上。

CC7S[1:0]：捕获/比较 7 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC7S[1:0]	方向	输入脚
00	输出	
01	输入	IC7映射在TI7FP7上
10	输入	IC7映射在TI8FP7上
11	输入	IC7映射在TRC上。

21.7.14 捕获/比较模式寄存器 4（PWMx_CCMR4）

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR4	FECBH	OC4CE	OC4M[2:0]			OC4PE	OC4FE	CC4S[1:0]	
PWMB_CCMR4	FEEBH	OC8CE	OC8M[2:0]			OC8PE	OC8FE	CC8S[1:0]	

OCnCE：输出比较 n 清零使能。该位用于使能使用 PWMETI 引脚上的外部事件来清通道 n 的输出信号（OCnREF）（n=4,8）

0：OCnREF 不受 ETRF 输入的影响；

1：一旦检测到 ETRF 输入高电平，OCnREF=0。

OCnM[2:0]：输出比较 n 模式，参考 OC1M。（n=4,8）

OCnPE：输出比较 n 预装载使能，参考 OP1PE。（n=4,8）

CC4S[1:0]：捕获/比较 4 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC4S[1:0]	方向	输入脚
00	输出	
01	输入	IC4映射在TI4FP4上
10	输入	IC4映射在TI3FP4上
11	输入	IC4映射在TRC上。

CC8S[1:0]：捕获/比较 8 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC8S[1:0]	方向	输入脚
00	输出	
01	输入	IC8映射在TI8FP8上
10	输入	IC8映射在TI7FP8上
11	输入	IC8映射在TRC上。

通道配置为捕获输入模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR4	FECBH	IC4F[3:0]				IC4PSC[1:0]		CC4S[1:0]	
PWMB_CCMR4	FEEBH	IC8F[3:0]				IC8PSC[1:0]		CC8S[1:0]	

ICnF[3:0]：输入捕获 n 滤波器选择，参考 IC1F。（n=4,8）

ICnPSC[1:0]：输入/捕获 n 预分频器，参考 IC1PSC。（n=4,8）

CC4S[1:0]：捕获/比较 4 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC4S[1:0]	方向	输入脚
00	输出	
01	输入	IC4映射在TI4FP4上

10	输入	IC4映射在TI3FP4上
11	输入	IC4映射在TRC上。

CC8S[1:0]：捕获/比较 8 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC8S[1:0]	方向	输入脚
00	输出	
01	输入	IC8映射在TI8FP8上
10	输入	IC8映射在TI7FP8上
11	输入	IC8映射在TRC上。

21.7.15 捕获/比较使能寄存器 1（PWMx_CCER1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCER1	FECCH	CC2NP	CC2NE	CC2P	CC2E	CC1NP	CC1NE	CC1P	CC1E
PWMB_CCER1	FEECH	-	-	CC6P	CC6E	-	-	CC5P	CC5E

CC6P：OC6 输入捕获/比较输出极性。参考 CC1P

CC6E：OC6 输入捕获/比较输出使能。参考 CC1E

CC5P：OC5 输入捕获/比较输出极性。参考 CC1P

CC5E：OC5 输入捕获/比较输出使能。参考 CC1E

CC2NP：OC2N 比较输出极性。参考 CC1NP

CC2NE：OC2N 比较输出使能。参考 CC1NE

CC2P：OC2 输入捕获/比较输出极性。参考 CC1P

CC2E：OC2 输入捕获/比较输出使能。参考 CC1E

CC1NP：OC1N 比较输出极性

0：高电平有效；

1：低电平有效。

注 1：一旦 LOCK 级别（PWMA_BKR 寄存器中的 LOCK 位）设为 3 或 2 且 CC1S=00（通道配置为输出），则该位不能被修改。

注 2：对于有互补输出的通道，该位是预装载的。如果 CCPC=1（PWMA_CR2 寄存器），只有在 COM 事件发生时，CC1NP 位才从预装载位中取新值。

CC1NE：OC1N 比较输出使能

0：关闭比较输出。

1：开启比较输出，其输出电平依赖于 MOE、OSSI、OSSR、OIS1、OIS1N 和 CC1E 位的值。

注：对于有互补输出的通道，该位是预装载的。如果 CCPC=1（PWMA_CR2 寄存器），只有在 COM 事件发生时，CC1NE 位才从预装载位中取新值。

CC1P：OC1 输入捕获/比较输出极性

CC1 通道配置为输出：

0：高电平有效

1：低电平有效

CC1 通道配置为输入或者捕获：

0：捕获发生在 TI1F 或 TI2F 的上升沿；

1：捕获发生在 TI1F 或 TI2F 的下降沿.

CC1E：OC1 输入捕获/比较输出使能

0：关闭输入捕获/比较输出；

1：开启输入捕获/比较输出。

注 1：一旦 LOCK 级别（PWMA_BKR 寄存器中的 LOCK 位）设为 3 或 2，则该位不能被修改。

注 2：对于有互补输出的通道，该位是预装载的。如果 CCPC=1（PWMA_CR2 寄存器），只有在 COM 事件发生时，CC1P 位才从预装载位中取新值。

STC MCU

带刹车功能的互补输出通道 OCi 和 OCiN 的控制位

控制位					输出状态	
MOE	OSSI	OSSR	CCiE	CCiNE	OCi 输出状态	OCiN 输出状态
1	X	0	0	0	输出禁止	输出禁止
		0	0	1	输出禁止	带极性的 OCiREF
		0	1	0	带极性的 OCiREF	输出禁止
		0	1	1	带极性和死区的 OCiREF	带极性和死区的反向 OCiREF
		1	0	0	输出禁止	输出禁止
		1	0	1	关闭状态 （输出使能且为无效电平） OCi=CCiP	带极性的 OCiREF
		1	1	0	带极性的 OCiREF	关闭状态 （输出使能且为无效电平） OCiN=CCiNP
		1	1	1	带极性和死区的 OCiREF	带极性和死区的反向 OCiREF
0	0	X	X	X	输出禁止	
	1				关闭状态（输出使能且为无效电平）异步地：OCi=CCiP，OCiN=CCiNP；然后，若时钟存在：经过一个死区时间后 OCi=OISi，OCiN=OISiN，假设 OISi 与 OISiN 并不都对应 OCi 和 OCiN 的有效电平。	

注：管脚连接到互补的 OCi 和 OCiN 通道的外部 I/O 管脚的状态，取决于 OCi 和 OCiN 通道状态和 GPIO 寄存器。

21.7.16 捕获/比较使能寄存器 2（PWMx_CCER2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCER2	FECDH	CC4NP	CC4NE	CC4P	CC4E	CC3NP	CC3NE	CC3P	CC3E
PWMB_CCER2	FEEDH	-	-	CC8P	CC8E	-	-	CC7P	CC7E

CC8P：OC8 输入捕获/比较输出极性。参考 CC1P
CC8E：OC8 输入捕获/比较输出使能。参考 CC1E
CC7P：OC7 输入捕获/比较输出极性。参考 CC1P
CC7E：OC7 输入捕获/比较输出使能。参考 CC1E
CC4NP：OC4N 比较输出极性。参考 CC1NP
CC4NE：OC4N 比较输出使能。参考 CC1NE
CC4P：OC4 输入捕获/比较输出极性。参考 CC1P
CC4E：OC4 输入捕获/比较输出使能。参考 CC1E
CC3NP：OC3N 比较输出极性。参考 CC1NP
CC3NE：OC3N 比较输出使能。参考 CC1NE
CC3P：OC3 输入捕获/比较输出极性。参考 CC1P
CC3E：OC3 输入捕获/比较输出使能。参考 CC1E

21.7.17 计数器高 8 位（PWMx_CNTRH）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CNTRH	FECEH	CNT1[15:8]							
PWMB_CNTRH	FEEEH	CNT2[15:8]							

CNTn[15:8]：计数器的高 8 位值（n= A,B）

21.7.18 计数器低 8 位（PWMx_CNTRL）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CNTRL	FECFH	CNT1[7:0]							
PWMB_CNTRL	FEEFH	CNT2[7:0]							

CNTn[7:0]：计数器的低 8 位值（n= A,B）

21.7.19 预分频器高 8 位（PWMx_PSCRH），输出频率计算公式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_PSCRH	FED0H	PSC1[15:8]							
PWMB_PSCRH	FEF0H	PSC2[15:8]							

PSCn[15:8]：预分频器的高 8 位值。（n= A,B）

预分频器用于对 CK_PSC 进行分频。计数器的时钟频率（fCK_CNT）等于 fCK_PSC/（PSCR[15:0]+1）。PSCR 包含了当更新事件产生时装入当前预分频器寄存器的值（更新事件包括计数器被 TIM_EGR 的 UG 位清 0 或被工作在复位模式的从控制器清 0）。这意味着为了使新的值起作用，必须产生一个更新事件。

PWM 输出频率计算公式

PWMA 和 PWMB 两组 PWM 的输出频率计算公式相同，且每组可设置不同的频率。

对齐模式	PWM输出频率计算公式
边沿对齐	$\text{PWM输出频率} = \dfrac{\text{系统工作频率SYSclk}}{(\text{PWMx_PSCR} + 1) \times (\text{PWMx_ARR} + 1)}$
中间对齐	$\text{PWM输出频率} = \dfrac{\text{系统工作频率SYSclk}}{(\text{PWMx_PSCR} + 1) \times \text{PWMx_ARR} \times 2}$

21.7.20 预分频器低 8 位（PWMx_PSCRL）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_PSCRL	FED1H	PSC1[7:0]							
PWMB_PSCRL	FEF1H	PSC2[7:0]							

PSCn[7:0]：预分频器的低 8 位值。（n= A,B）

21.7.21 自动重装载寄存器高 8 位（PWMx_ARRH）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ARRH	FED2H	ARR1[15:8]							
PWMB_ARRH	FEF2H	ARR2[15:8]							

ARRn[15:8]：自动重装载高 8 位值（n= A,B）

ARR 包含了将要装载入实际的自动重装载寄存器的值。当自动重装载的值为 0 时，计数器不工作。

21.7.22 自动重装载寄存器低 8 位（PWMx_ARRL）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ARRL	FED3H	ARR1[7:0]							
PWMB_ARRL	FEF3H	ARR2[7:0]							

ARRn[7:0]：自动重装载低 8 位值（n= A,B）

21.7.23 重复计数器寄存器（PWMx_RCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_RCR	FED4H	REP1[7:0]							
PWMB_RCR	FEF4H	REP2[7:0]							

REPn[7:0]：重复计数器值（n= A,B）

开启了预装载功能后，这些位允许用户设置比较寄存器的更新速率（即周期性地从预装载寄存器 传输到当前寄存器）；如果允许产生更新中断，则会同时影响产生更新中断的速率。每次向下计数器 REP_CNT 达到 0，会产生一个更新事件并且计数器 REP_CNT 重新从 REP 值开 始计数。由于 REP_CNT 只有在周期更新事件 U_RC 发生时才重载 REP 值，因此对 PWMn_RCR 寄存器写入的新值只在下次周期更新事件发生时才起作用。这意味着在 PWM 模式中，（REP+1）对应着：

- 在边沿对齐模式下，PWM 周期的数目；
- 在中心对称模式下，PWM 半周期的数目；

21.7.24 捕获/比较寄存器 1/5 高 8 位（PWMx_CCR1H）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR1H	FED5H	CCR1[15:8]							
PWMB_CCR5H	FEF5H	CCR5[15:8]							

CCRn[15:8]：捕获/比较 n 的高 8 位值（n=1,5）

若 CCn 通道配置为输出：CCRn 包含了装入当前比较值（预装载值）。如果在 PWMn_CCMR1 寄存器（OCnPE 位）中未选择预装载功能，写入的数值会立即传输至当前寄存器中。否则只有当更新事件发生时，此预装载值才传输至当前捕获/比较 n 寄存器中。当前比较值同计数器 PWMn_CNT 的值相比较，并在 OCn 端口上产生输出信号。

若 CCn 通道配置为输入：CCRn 包含了上一次输入捕获事件发生时的计数器值（此时该寄存器为只读）。

21.7.25 捕获/比较寄存器 1/5 低 8 位（PWMx_CCR1L）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0

PWMA_CCR1L	FED6H	CCR1[7:0]
PWMB_CCR5L	FEF6H	CCR5[7:0]

CCRn[7:0]：捕获/比较 n 的低 8 位值（n=1,5）

21.7.26 捕获/比较寄存器 2/6 高 8 位（PWMx_CCR2H）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR2H	FED7H	CCR2[15:8]							
PWMB_CCR6H	FEF7H	CCR6[15:8]							

CCRn[15:8]：捕获/比较 n 的高 8 位值（n=2,6）

21.7.27 捕获/比较寄存器 2/6 低 8 位（PWMx_CCR2L）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR2L	FED8H	CCR2[7:0]							
PWMB_CCR6L	FEF8H	CCR6[7:0]							

CCRn[7:0]：捕获/比较 n 的低 8 位值（n=2,6）

21.7.28 捕获/比较寄存器 3/7 高 8 位（PWMx_CCR3H）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR3H	FED9H	CCR3[15:8]							
PWMB_CCR7H	FEF9H	CCR7[15:8]							

CCRn[15:8]：捕获/比较 n 的高 8 位值（n=3,7）

21.7.29 捕获/比较寄存器 3/7 低 8 位（PWMx_CCR3L）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR3L	FEDAH	CCR3[7:0]							
PWMB_CCR7L	FEFAH	CCR7[7:0]							

CCRn[7:0]：捕获/比较 n 的低 8 位值（n=3,7）

21.7.30 捕获/比较寄存器 4/8 高 8 位（PWMx_CCR4H）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR4H	FEDBH	CCR4[15:8]							
PWMB_CCR8H	FEFBH	CCR8[15:8]							

CCRn[15:8]：捕获/比较 n 的高 8 位值（n=4,8）

21.7.31 捕获/比较寄存器 4/8 低 8 位（PWMx_CCR4L）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR4L	FEDCH	CCR4[7:0]							
PWMB_CCR8L	FEFCH	CCR8[7:0]							

CCRn[7:0]：捕获/比较 n 的低 8 位值（n=4,8）

STC MCU

21.7.32 刹车寄存器（PWMx_BKR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_BKR	FEDDH	MOEA	AOEA	BKPA	BKEA	OSSRA	OSSIA	LOCKA[1:0]	
PWMB_BKR	FEFDH	MOEB	AOEB	BKPB	BKEB	OSSRB	OSSIB	LOCKB[1:0]	

MOEn：主输出使能。一旦刹车输入有效，该位被硬件异步清 0。根据 AOE 位的设置值，该位可以由软件置 1 或被自动置 1。它仅对配置为输出的通道有效。（n= A,B）

0：禁止 OC 和 OCN 输出或强制为空闲状态

1：如果设置了相应的使能位（PWMn_CCERX 寄存器的 CCIE 位），则使能 OC 和 OCN 输出。

AOEn：自动输出使能（n= A,B）

0：MOE 只能被软件置 1；

1：MOE 能被软件置 1 或在下一个更新事件被自动置 1（如果刹车输入无效）。

注：一旦 LOCK 级别（PWMn_BKR 寄存器中的 LOCK 位）设为 1，则该位不能被修改

BKPn：刹车输入极性（n= A,B）

0：刹车输入低电平有效

1：刹车输入高电平有效

注：一旦 LOCK 级别（PWMn_BKR 寄存器中的 LOCK 位）设为 1，则该位不能被修改

BKEn：刹车功能使能（n= A,B）

0：禁止刹车输入（BRK）

1：开启刹车输入（BRK）

注：一旦 LOCK 级别（PWMn_BKR 寄存器中的 LOCK 位）设为 1，则该位不能被修改。

OSSRn：运行模式下“关闭状态”选择。该位在 MOE=1 且通道设为输出时有效（n= A,B）

0：当 PWM 不工作时，禁止 OC/OCN 输出（OC/OCN 使能输出信号=0）；

1：当 PWM 不工作时，一旦 CCiE=1 或 CCiNE=1，首先开启 OC/OCN 并输出无效电平，然后置 OC/OCN 使能输出信号=1。

注：一旦 LOCK 级别（PWMn_BKR 寄存器中的 LOCK 位）设为 2，则该位不能被修改。

OSSIn：空闲模式下“关闭状态”选择。该位在 MOE=0 且通道设为输出时有效。（n= A,B）

0：当 PWM 不工作时，禁止 OC/OCN 输出（OC/OCN 使能输出信号=0）；

1：当 PWM 不工作时，一旦 CCiE=1 或 CCiNE=1，OC/OCN 首先输出其空闲电平，然后 OC/OCN 使能输出信号=1。

注：一旦 LOCK 级别（PWMn_BKR 寄存器中的 LOCK 位）设为 2，则该位不能被修改。

LOCKn[1:0]：锁定设置。该位为防止软件错误而提供的写保护措施（n= A,B）

LOCKn[1:0]	保护级别	保护内容
00	无保护	寄存器无写保护
01	锁定级别1	不能写入PWMn_BKR寄存器的BKE、BKP、AOE位和PWMn_OISR寄存器的OISI位
10	锁定级别2	不能写入锁定级别1中的各位，也不能写入CC极性位以及OSSR/OSSI位
11	锁定级别3	不能写入锁定级别2中的各位，也不能写入CC控制位

注：由于 BKE、BKP、AOE、OSSR、OSSI 位可被锁定（依赖于 LOCK 位），因此在第一次写 PWMn_BKR 寄存器时必须对它们进行设置。

21.7.33 死区寄存器（PWMx_DTR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_DTR	FEDEH	DTGA[7:0]							
PWMB_DTR	FEFEH	DTGB[7:0]							

DTGn[7:0]：死区发生器设置。（n= A,B）

这些位定义了插入互补输出之间的死区持续时间。（t_{CK_PSC} 为 PWMn 的时钟脉冲）

DTGn[7:5]	死区时间
000	DTGn[7:0] * t_{CK_PSC}
001	
010	
011	
100	(64 + DTGn[6:0]) * 2 * t_{CK_PSC}
101	
110	(32 + DTGn[5:0]) * 8 * t_{CK_PSC}
111	(32 + DTGn[4:0]) * 16 * t_{CK_PSC}

21.7.34 输出空闲状态寄存器（PWMx_OISR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_OISR	FEDFH	OIS4N	OIS4	OIS3N	OIS3	OIS2N	OIS2	OIS1N	OIS1
PWMB_OISR	FEFFH	-	OIS8	-	OIS7	-	OIS6	-	OIS5

OIS8：空闲状态时 OC8 输出电平

OIS7：空闲状态时 OC7 输出电平

OIS6：空闲状态时 OC6 输出电平

OIS5：空闲状态时 OC5 输出电平

OIS4N：空闲状态时 OC4N 输出电平

OIS4：空闲状态时 OC4 输出电平

OIS3N：空闲状态时 OC3N 输出电平

OIS3：空闲状态时 OC3 输出电平

OIS2N：空闲状态时 OC2N 输出电平

OIS2：空闲状态时 OC2 输出电平

OIS1N：空闲状态时 OC1N 输出电平

0：当 MOE=0 时，则在一个死区时间后，OC1N=0；

1：当 MOE=0 时，则在一个死区时间后，OC1N=1。

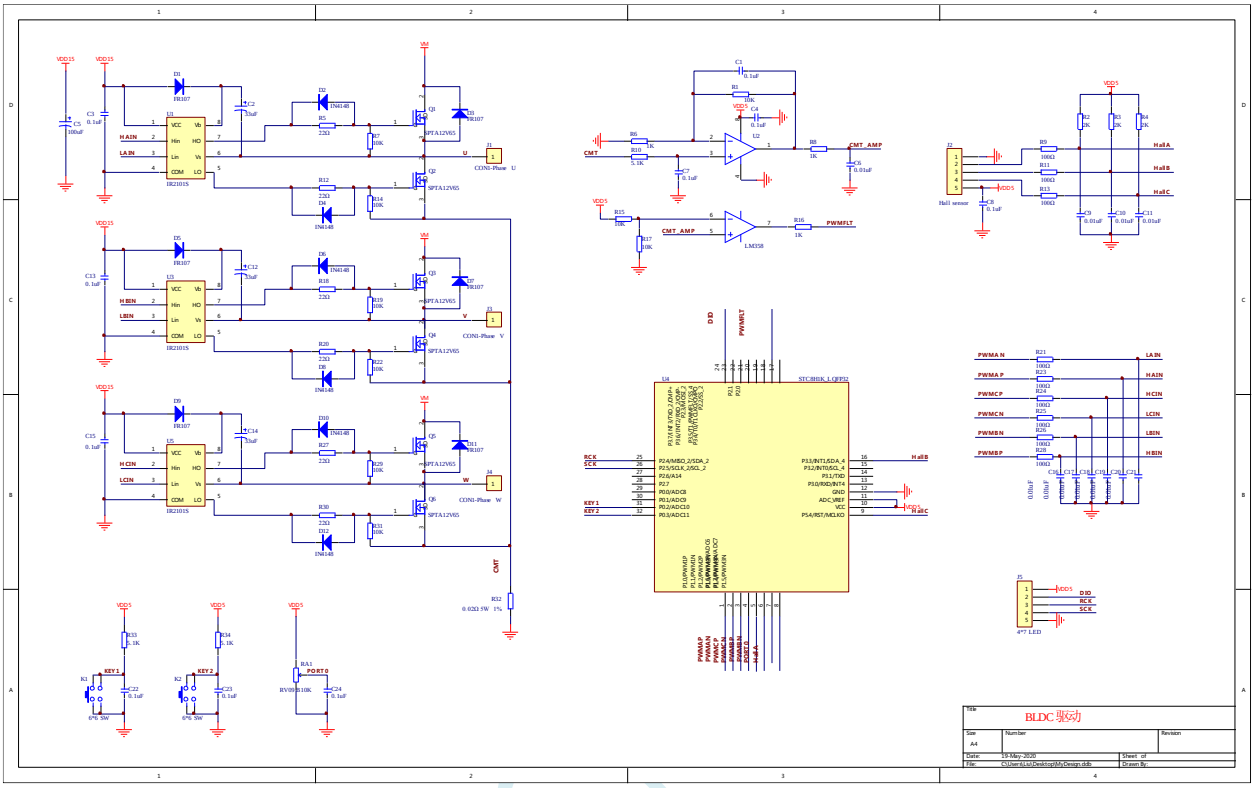
OIS1：空闲状态时 OC1 输出电平

0：当 MOE=0 时，如果 OC1N 使能，则在一个死区后，OC1=0；

1：当 MOE=0 时，如果 OC1N 使能，则在一个死区后，OC1=1。

21.8 范例程序

21.8.1 六步 PWM 驱动无刷直流马达(带 HALL)



C 语言代码

```
//测试工作频率为 11.0592MHz

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

typedef     unsigned char    u8;
typedef     unsigned int     u16;

typedef struct TIM1_struct
{
    volatile unsigned char CR1;          /*!< control register 1 */
    volatile unsigned char CR2;          /*!< control register 2 */
    volatile unsigned char SMCR;         /*!< Synchro mode control register */
    volatile unsigned char ETR;          /*!< external trigger register */
    volatile unsigned char IER;          /*!< interrupt enable register*/
    volatile unsigned char SR1;          /*!< status register 1 */
    volatile unsigned char SR2;          /*!< status register 2 */
    volatile unsigned char EGR;          /*!< event generation register */
    volatile unsigned char CCMR1;        /*!< CC mode register 1 */
    volatile unsigned char CCMR2;        /*!< CC mode register 2 */
    volatile unsigned char CCMR3;        /*!< CC mode register 3 */
    volatile unsigned char CCMR4;        /*!< CC mode register 4 */
    volatile unsigned char CCER1;        /*!< CC enable register 1 */
    volatile unsigned char CCER2;        /*!< CC enable register 2 */
```

```
    volatile unsigned char CNTRH;                    /*!< counter high */
    volatile unsigned char CNTRL;                    /*!< counter low */
    volatile unsigned char PSCRH;                    /*!< prescaler high */
    volatile unsigned char PSCRL;                    /*!< prescaler low */
    volatile unsigned char ARRH;                     /*!< auto-reload register high */
    volatile unsigned char ARRL;                     /*!< auto-reload register low */
    volatile unsigned char RCR;                      /*!< Repetition Counter register */
    volatile unsigned char CCR1H;                    /*!< capture/compare register 1 high */
    volatile unsigned char CCR1L;                    /*!< capture/compare register 1 low */
    volatile unsigned char CCR2H;                    /*!< capture/compare register 2 high */
    volatile unsigned char CCR2L;                    /*!< capture/compare register 2 low */
    volatile unsigned char CCR3H;                    /*!< capture/compare register 3 high */
    volatile unsigned char CCR3L;                    /*!< capture/compare register 3 low */
    volatile unsigned char CCR4H;                    /*!< capture/compare register 3 high */
    volatile unsigned char CCR4L;                    /*!< capture/compare register 3 low */
    volatile unsigned char BKR;                      /*!< Break Register */
    volatile unsigned char DTR;                      /*!< dead-time register */
    volatile unsigned char OISR;                     /*!< Output idle register */
}TIM1_TypeDef;

#define   TIM1_BaseAddress   0xFEC0
#define   TIM2_BaseAddress   0xFEE0

#define   TIM1               ((TIM1_TypeDef xdata*) TIM1_BaseAddress)
#define   TIM2               ((TIM1_TypeDef xdata*) TIM2_BaseAddress)

#define   PWMA_ETRPS         (*(unsigned char volatile xdata *) 0xFEB0)
#define   PWMA_ENO           (*(unsigned char volatile xdata *) 0xFEB1)
#define   PWMA_PS            (*(unsigned char volatile xdata *) 0xFEB2)
#define   PWMB_ENO           (*(unsigned char volatile xdata *) 0xFEB5)
#define   PWMB_PS            (*(unsigned char volatile xdata *) 0xFEB6)

sfr       ADC_CONTR    =     0xbc;
sfr       ADC_RES      =     0xbd;
sfr       ADC_RESL     =     0xbe;
sfr       ADCCFG       =     0xde;
sfr       CMPCR1       =     0xe6;
sfr       CMPCR2       =     0xe7;

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       P5M0         =     0xca;
sfr       P5M1         =     0xc9;
sfr       P5           =     0xc8;
sfr       P_SW2        =     0xba;

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    P50     =   P5^0;
sbit    P51     =   P5^1;
sbit    P52     =   P5^2;
sbit    P53     =   P5^3;
sbit    P54     =   P5^4;
sbit    P55     =   P5^5;

#define TRUE        1
#define FALSE       0

#define RV09_CH     6

#define TIM1_Period     ((u16)0x0180)
#define TIM1_STPulse    ((u16)342)

#define START       0x1A
#define RUN         0x1B
#define STOP        0x1C
#define IDLE        0x1D

#define TIM1_OCMODE_MASK        ((u8)0x70)
#define TIM1_OCCE_ENABLE        ((u8)0x80)
#define TIM1_OCCE_DISABLE       ((u8)0x00)
#define TIM1_OCMODE_TIMING      ((u8)0x00)
#define TIM1_OCMODE_ACTIVE      ((u8)0x10)
#define TIM1_OCMODE_INACTIVE    ((u8)0x20)
#define TIM1_OCMODE_TOGGLE      ((u8)0x30)
#define TIM1_FORCE_INACTIVE     ((u8)0x40)
#define TIM1_FORCE_ACTIVE       ((u8)0x50)
#define TIM1_OCMODE_PWMA        ((u8)0x60)
```

STC MCU

```
#define    TIM1_OCMODE_PWMB        ((u8)0x70)
#define    CC1_POLARITY_HIGH       ((u8)0x02)
#define    CC1N_POLARITY_HIGH      ((u8)0x08)
#define    CC2_POLARITY_HIGH       ((u8)0x20)
#define    CC2N_POLARITY_HIGH      ((u8)0x80)
#define    CC1_POLARITY_LOW        ((u8)~0x02)
#define    CC1N_POLARITY_LOW       ((u8)~0x08)
#define    CC2_POLARITY_LOW        ((u8)~0x20)
#define    CC2N_POLARITY_LOW       ((u8)~0x80)
#define    CC1_OCENABLE            ((u8)0x01)
#define    CC1N_OCENABLE           ((u8)0x04)
#define    CC2_OCENABLE            ((u8)0x10)
#define    CC2N_OCENABLE           ((u8)0x40)
#define    CC1_OCDISABLE           ((u8)~0x01)
#define    CC1N_OCDISABLE          ((u8)~0x04)
#define    CC2_OCDISABLE           ((u8)~0x10)
#define    CC2N_OCDISABLE          ((u8)~0x40)
#define    CC3_POLARITY_HIGH       ((u8)0x02)
#define    CC3N_POLARITY_HIGH      ((u8)0x08)
#define    CC4_POLARITY_HIGH       ((u8)0x20)
#define    CC4N_POLARITY_HIGH      ((u8)0x80)
#define    CC3_POLARITY_LOW        ((u8)~0x02)
#define    CC3N_POLARITY_LOW       ((u8)~0x08)
#define    CC4_POLARITY_LOW        ((u8)~0x20)
#define    CC4N_POLARITY_LOW       ((u8)~0x80)
#define    CC3_OCENABLE            ((u8)0x01)
#define    CC3N_OCENABLE           ((u8)0x04)
#define    CC4_OCENABLE            ((u8)0x10)
#define    CC4N_OCENABLE           ((u8)0x40)
#define    CC3_OCDISABLE           ((u8)~0x01)
#define    CC3N_OCDISABLE          ((u8)~0x04)
#define    CC4_OCDISABLE           ((u8)~0x10)
#define    CC4N_OCDISABLE          ((u8)~0x40)

void LED_OUT(u8 X);                          //LED 单字节串行移位函数

unsigned char code LED_0F[] =
{
    0xC0,0xF9,0xA4,0xB0,
    0x99,0x92,0x82,0xF8,
    0x80,0x90,0x8C,0xBF,
    0xC6,0xA1,0x86,0xFF,
    0xbf
};

#define    DIO         P23          //串行数据输入
#define    RCLK        P24          //时钟脉冲信号——上升沿有效
#define    SCLK        P25          //打入信号————上升沿有效

void DelayXus(unsigned char delayTime);
void DelayXms( unsigned char delayTime);
unsigned int ADC_Convert(u8 ch);
void PWM_Init(void);
void SPEED_ADJ();
unsigned char RD_HALL();
void MOTOR_START();
void MOTOR_STOP();
unsigned char KEY_detect();
void LED4_Display (unsigned int dat,unsigned char num);
```

```
unsigned char Display_num=1;
unsigned int Display_dat=0;
unsigned int Motor_speed;
unsigned char Motor_sta = IDLE;
unsigned char BRK_occur=0;
unsigned int TIM2_CAP1_v=0;
unsigned int CAP1_avg=0;
unsigned char CAP1_cnt=0;
unsigned long CAP1_sum=0;

void main(void)
{
	P_SW2 = 0x80;

	P1 = 0x00;
	P0M1 = 0x0C;
	P0M0 = 0x01;
	P1M1 = 0xc0;
	P1M0 = 0x3F;
	P2M1 = 0x00;
	P2M0 = 0x38;
	P3M1 = 0x28;
	P3M0 = 0x00;

	ET0=1;
	TR0=1;

	ADCCFG = 0x0f;
	ADC_CONTR = 0x80;

	PWMA_ENO = 0x3F;		//PWMA 输出使能
	PWMB_ENO = 0x00;		//PWMB 输出使能
	PWMA_PS = 0x00;		//PWMA pin 选择
	PWMB_PS = 0xd5;		//PWMB pin 选择

/*************************************************************
输出比较模式 PWMx_duty = [CCRx/(ARR + 1)]*100
*************************************************************/
/************PWMB 接 hall 传感器****************/
//////////////时基单元//////////////
	TIM2-> PSCRL = 15;
	TIM2-> ARRH = 0xff;		//自动重装载寄存器，计数器 overflow 点
	TIM2-> ARRL = 0xff;
	TIM2-> CCR4H = 0x00;
	TIM2-> CCR4L = 0x05;

//////////////通道配置//////////////
	TIM2-> CCMR1 = 0x43;		//通道模式配置
	TIM2-> CCMR2 = 0x41;
	TIM2-> CCMR3 = 0x41;
	TIM2-> CCMR4 = 0x70;
	TIM2-> CCER1 = 0x11;
	TIM2-> CCER2 = 0x11;

//////////////模式配置//////////////
	TIM2-> CR2 = 0xf0;
	TIM2-> CR1 = 0x81;
	TIM2-> SMCR = 0x44;
```

```
//////////使能&中断配置//////////
        TIM2-> BKR = 0x80;                          //主输出使能
        TIM2-> IER = 0x02;                          //使能中断


/************PWMA 控制马达换相****************/
//////////时基单元//////////
        TIM1-> PSCRH = 0x00;                        //预分频寄存器
        TIM1-> PSCRL = 0x00;
        TIM1-> ARRH = (u8)(TIM1_Period >> 8);
        TIM1-> ARRL = (u8)(TIM1_Period);

//////////通道配置//////////
        TIM1-> CCMR1 = 0x70;                        //通道模式配置
        TIM1-> CCMR2 = 0x70;
        TIM1-> CCMR3 = 0x70;
        TIM1-> CCER1 = 0x11;                        //配置通道输出使能和极性
        TIM1-> CCER2 = 0x01;                        //配置通道输出使能和极性
        TIM1-> OISR = 0xAA;                         //配置 MOE=0 时各通道输出电平

//////////模式配置//////////
        TIM1-> CR1 = 0xA0;
        TIM1-> CR2 = 0x24;
        TIM1-> SMCR = 0x20;

//////////使能&中断配置//////////
        TIM1-> BKR = 0x1c;
        TIM1-> CR1 |= 0x01;                         //使能计数器

        EA = 1;
        while (1)
        {
                P22=~P22;
                Display_dat = Motor_speed;          //Motor_speed

                switch(Motor_sta)
                {
                case START:
                        MOTOR_START();
                        Motor_sta = RUN;
                        break;
                case RUN:
                        SPEED_ADJ();
                        if((KEY_detect() == 2)||(BRK_occur == TRUE))
                                Motor_sta = STOP;
                        break;
                case STOP:
                        MOTOR_STOP();
                        Motor_sta = IDLE;
                        break;
                case IDLE:
                        if(KEY_detect()==1)
                                Motor_sta = START;
                        BRK_occur = FALSE;
                        Motor_speed = 0;
                        CAP1_avg = 0;
                        CAP1_cnt = 0;
                        CAP1_sum = 0;
```

```
                break;
            }
        }
}

void TIM0_ISR() interrupt 1
{
        TH0=0xf0;
        if(Display_num>8)
            Display_num=1;
        LED4_Display(Display_dat,Display_num);
        Display_num=(Display_num<<1);
}

void PWMA_ISR() interrupt 26
{
        if((TIM1->SR1 & 0x20))
        {
            switch(RD_HALL())
            {
            case 3:
                TIM1-> CCMR3 &= ~TIM1_OCMODE_MASK;
                TIM1-> CCMR3 |= TIM1_FORCE_INACTIVE;
                TIM1-> CCMR1 &= ~TIM1_OCMODE_MASK;
                TIM1-> CCMR1 |= TIM1_OCMODE_PWMB;
                break;
            case 2:
                TIM1-> CCER1 &= CC2N_POLARITY_LOW;
                TIM1-> CCER2 |= CC3N_POLARITY_HIGH;
                break;
            case 6:
                TIM1-> CCMR1 &= ~TIM1_OCMODE_MASK;
                TIM1-> CCMR1 |= TIM1_FORCE_INACTIVE;
                TIM1-> CCMR2 &= ~TIM1_OCMODE_MASK;
                TIM1-> CCMR2 |= TIM1_OCMODE_PWMB;
                break;
            case 4:
                TIM1-> CCER1 |= CC1N_POLARITY_HIGH;
                TIM1-> CCER2 &= CC3N_POLARITY_LOW;
                break;
            case 5:
                TIM1-> CCMR2 &= ~TIM1_OCMODE_MASK;
                TIM1-> CCMR2 |= TIM1_FORCE_INACTIVE;
                TIM1-> CCMR3 &= ~TIM1_OCMODE_MASK;
                TIM1-> CCMR3 |= TIM1_OCMODE_PWMB;
                break;
            case 1:
                TIM1-> CCER1 &= CC1N_POLARITY_LOW;
                TIM1-> CCER1 |= CC2N_POLARITY_HIGH;
                break;
            }

            CAP1_sum += TIM2_CAP1_v;
            CAP1_cnt++;
            if(CAP1_cnt==128)
            {
                CAP1_cnt=0;
                CAP1_avg = (CAP1_sum>>7);
                CAP1_sum = 0;
```

```
                Motor_speed = 5000000/CAP1_avg;
            }

            TIM1->SR1 &=~0x20;                          //清零
        }
        if((TIM1->SR1 & 0x80))                          //BRK
        {
            BRK_occur = TRUE;
            TIM1->SR1 &=~0x80;                          //清零
        }
}

void PWMB_ISR() interrupt 27
{
        if((TIM2->SR1 & 0x02))
        {
            TIM2_CAP1_v = TIM2-> CCR1H;
            TIM2_CAP1_v = (TIM2_CAP1_v<<8) + TIM2-> CCR1L;
            TIM2->SR1 &=~0x02;
        }
}

void DelayXus(unsigned char delayTime)
{
        int i = 0;
        while( delayTime--)
        {
            for( i = 0 ; i < 1 ; i++);
        }
}

void DelayXms( unsigned char delayTime )
{
        int i = 0;
        while( delayTime--)
        {
            for( i = 0 ; i < 2 ; i++)
            {
                DelayXus(100);
            }
        }
}

unsigned int ADC_Convert(u8 ch)
{
        u16 res=0;

        ADC_CONTR &= ~0x0f;
        ADC_CONTR |= ch;
        ADC_CONTR |= 0x40;
        DelayXus(1);
        while (!(ADC_CONTR & 0x20));
        ADC_CONTR &= ~0x20;

        res = ADC_RES;
        res = (res<<2)+(ADC_RESL>>6);
        return res;
}
```

STC MCU

```
void SPEED_ADJ()
{
    u16 ADC_result;

    ADC_result = (ADC_Convert(RV09_CH)/3);
    TIM1-> CCR1H = (u8)(ADC_result >> 8);          //计数器比较值
    TIM1-> CCR1L = (u8)(ADC_result);
    TIM1-> CCR2H = (u8)(ADC_result >> 8);
    TIM1-> CCR2L = (u8)(ADC_result);
    TIM1-> CCR3H = (u8)(ADC_result >> 8);
    TIM1-> CCR3L = (u8)(ADC_result);
}

unsigned char RD_HALL()
{
    unsigned char Hall_sta = 0;

    (P17)? (Hall_sta|=0x01) : (Hall_sta&=~0x01);
    (P54)? (Hall_sta|=0x02) : (Hall_sta&=~0x02);
    (P33)? (Hall_sta|=0x04) : (Hall_sta&=~0x04);

    return Hall_sta;
}

void MOTOR_START()
{
    u16 temp;
    u16 ADC_result;

    TIM1-> CCR1H = (u8)(TIM1_STPulse >> 8);        //计数器比较值
    TIM1-> CCR1L = (u8)(TIM1_STPulse);
    TIM1-> CCR2H = (u8)(TIM1_STPulse >> 8);
    TIM1-> CCR2L = (u8)(TIM1_STPulse);
    TIM1-> CCR3H = (u8)(TIM1_STPulse >> 8);
    TIM1-> CCR3L = (u8)(TIM1_STPulse);
    TIM1-> BKR |= 0x80;                            //主输出使能相当于总开关
    TIM1-> IER |= 0xA0;                            //使能中断

    switch(RD_HALL())
    {
    case 1:
        TIM1-> CCER1 &= CC1N_POLARITY_LOW;
        TIM1-> CCER1 |= CC2N_POLARITY_HIGH;
        TIM1-> CCER2 &= CC3N_POLARITY_LOW;
        TIM1-> CCMR3 &= ~TIM1_OCMODE_MASK;
        TIM1-> CCMR3 |= TIM1_FORCE_INACTIVE;
        TIM1-> CCMR2 &= ~TIM1_OCMODE_MASK;
        TIM1-> CCMR2 |= TIM1_FORCE_INACTIVE;
        TIM1-> CCMR1 &= ~TIM1_OCMODE_MASK;
        TIM1-> CCMR1 |= TIM1_OCMODE_PWMB;
        break;
    case 3:
        TIM1-> CCMR3 &= ~TIM1_OCMODE_MASK;
        TIM1-> CCMR3 |= TIM1_FORCE_INACTIVE;
        TIM1-> CCMR2 &= ~TIM1_OCMODE_MASK;
        TIM1-> CCMR2 |= TIM1_FORCE_INACTIVE;
        TIM1-> CCMR1 &= ~TIM1_OCMODE_MASK;
        TIM1-> CCMR1 |= TIM1_OCMODE_PWMB;
        TIM1-> CCER1 &= CC1N_POLARITY_LOW;
```

STC MCU

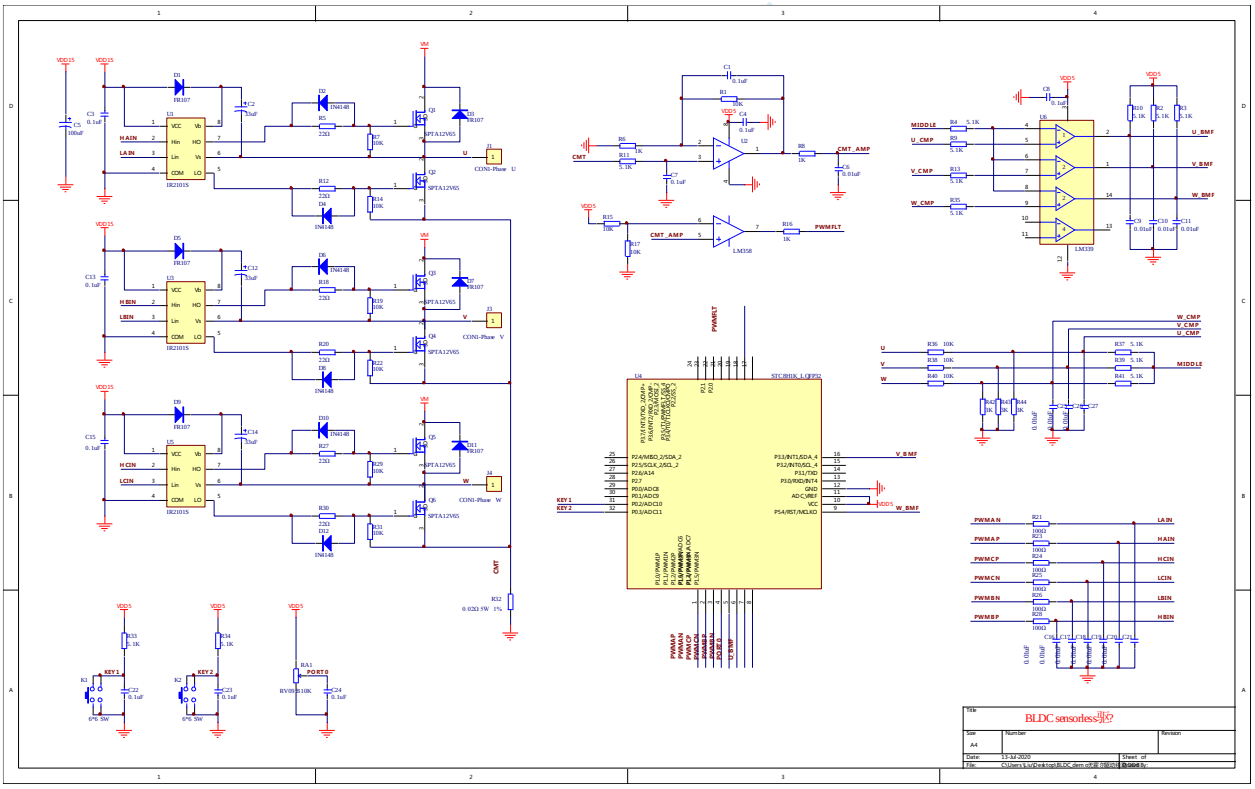
```
        TIM1-> CCER1 &= CC2N_POLARITY_LOW;
        TIM1-> CCER2 |= CC3N_POLARITY_HIGH;
        break;
    case 2:
        TIM1-> CCER1 &= CC1N_POLARITY_LOW;
        TIM1-> CCER1 &= CC2N_POLARITY_LOW;
        TIM1-> CCER2 |= CC3N_POLARITY_HIGH;
        TIM1-> CCMR1 &= ~TIM1_OCMODE_MASK;
        TIM1-> CCMR1 |= TIM1_FORCE_INACTIVE;
        TIM1-> CCMR2 &= ~TIM1_OCMODE_MASK;
        TIM1-> CCMR2 |= TIM1_OCMODE_PWMB;
        TIM1-> CCMR3 &= ~TIM1_OCMODE_MASK;
        TIM1-> CCMR3 |= TIM1_FORCE_INACTIVE;
        break;
    case 6:
        TIM1-> CCMR1 &= ~TIM1_OCMODE_MASK;
        TIM1-> CCMR1 |= TIM1_FORCE_INACTIVE;
        TIM1-> CCMR2 &= ~TIM1_OCMODE_MASK;
        TIM1-> CCMR2 |= TIM1_OCMODE_PWMB;
        TIM1-> CCMR3 &= ~TIM1_OCMODE_MASK;
        TIM1-> CCMR3 |= TIM1_FORCE_INACTIVE;
        TIM1-> CCER1 |= CC1N_POLARITY_HIGH;
        TIM1-> CCER1 &= CC2N_POLARITY_LOW;
        TIM1-> CCER2 &= CC3N_POLARITY_LOW;
        break;
    case 4:
        TIM1-> CCER1 |= CC1N_POLARITY_HIGH;
        TIM1-> CCER1 &= CC2N_POLARITY_LOW;
        TIM1-> CCER2 &= CC3N_POLARITY_LOW;
        TIM1-> CCMR1 &= ~TIM1_OCMODE_MASK;
        TIM1-> CCMR1 |= TIM1_FORCE_INACTIVE;
        TIM1-> CCMR2 &= ~TIM1_OCMODE_MASK;
        TIM1-> CCMR2 |= TIM1_FORCE_INACTIVE;
        TIM1-> CCMR3 &= ~TIM1_OCMODE_MASK;
        TIM1-> CCMR3 |= TIM1_OCMODE_PWMB;
        break;
    case 5:
        TIM1-> CCMR1 &= ~TIM1_OCMODE_MASK;
        TIM1-> CCMR1 |= TIM1_FORCE_INACTIVE;
        TIM1-> CCMR2 &= ~TIM1_OCMODE_MASK;
         TIM1-> CCMR2 |= TIM1_FORCE_INACTIVE;
        TIM1-> CCMR3 &= ~TIM1_OCMODE_MASK;
         TIM1-> CCMR3 |= TIM1_OCMODE_PWMB;
        TIM1-> CCER1 &= CC1N_POLARITY_LOW;
        TIM1-> CCER1 |= CC2N_POLARITY_HIGH;
        TIM1-> CCER2 &= CC3N_POLARITY_LOW;
        break;
    }
    ADC_result = (ADC_Convert(RV09_CH)/3);

    for(temp = TIM1_STPulse; temp > ADC_result; temp--)
    {
        TIM1-> CCR1H = (u8)(temp >> 8);                    //计数器比较值
        TIM1-> CCR1L = (u8)(temp);
        TIM1-> CCR2H = (u8)(temp >> 8);
        TIM1-> CCR2L = (u8)(temp);
        TIM1-> CCR3H = (u8)(temp >> 8);
        TIM1-> CCR3L = (u8)(temp);
        DelayXms(10);
```

```
        }
}

void MOTOR_STOP()
{
        TIM1-> BKR &= ~0x80;
        TIM1-> IER &= ~0xA0;
}

void LED4_Display (u16 dat,u8 num)
{
        switch(num)
        {
        case 0x01:
                LED_OUT(LED_0F[(dat/1)%10]);
                LED_OUT(0x01);
                RCLK = 0;
                RCLK = 1;
                break;
        case 0x02:
                LED_OUT(LED_0F[(dat/10)%10]
                LED_OUT(0x02);
                RCLK = 0;
                RCLK = 1;
                break;
        case 0x04:
                LED_OUT(LED_0F[(dat/100)%10]);
                LED_OUT(0x04);
                RCLK = 0;
                RCLK = 1;
                break;
        case 0x08:
                LED_OUT(LED_0F[(dat/1000)%10]);
                LED_OUT(0x08);
                RCLK = 0;
                RCLK = 1;
                break;
        }
}

void LED_OUT(u8 X)
{
        u8 i;

        for(i=8;i>=1;i--)
        {
                if (X&0x80) DIO=1;
                else DIO=0;
                X<<=1;
                SCLK = 0;
                SCLK = 1;
        }
}

unsigned char KEY_detect()
{
        if(!P02)
        {
                DelayXms(10);
```

STC MCU

```
        if(!P02)
        {
            return 1;
        }
        else return 0;
    }
    else if(!P03)
    {
        DelayXms(10);
        if(!P03)
        {
            return 2;
        }
        else return 0;
    }
    else return 0;
}
```

21.8.2 BLDC 无刷直流电机驱动(无 HALL)



C 语言代码

```
//测试工作频率为 11.0592MHz
//本例程实现如下功能: 通过 3 组 PWM 通道控制无霍尔马达运转
//本例程仅适用 57BL02 马达在 24V 无负载条件下演示

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

typedef    unsigned char    u8;
typedef    unsigned int     u16;
```

```
typedef struct TIM1_struct
{
    volatile unsigned char CR1;          /*!< control register 1 */
    volatile unsigned char CR2;          /*!< control register 2 */
    volatile unsigned char SMCR;         /*!< Synchro mode control register */
    volatile unsigned char ETR;          /*!< external trigger register */
    volatile unsigned char IER;          /*!< interrupt enable register*/
    volatile unsigned char SR1;          /*!< status register 1 */
    volatile unsigned char SR2;          /*!< status register 2 */
    volatile unsigned char EGR;          /*!< event generation register */
    volatile unsigned char CCMR1;        /*!< CC mode register 1 */
    volatile unsigned char CCMR2;        /*!< CC mode register 2 */
    volatile unsigned char CCMR3;        /*!< CC mode register 3 */
    volatile unsigned char CCMR4;        /*!< CC mode register 4 */
    volatile unsigned char CCER1;        /*!< CC enable register 1 */
    volatile unsigned char CCER2;        /*!< CC enable register 2 */
    volatile unsigned char CNTRH;        /*!< counter high */
    volatile unsigned char CNTRL;        /*!< counter low */
    volatile unsigned char PSCRH;        /*!< prescaler high */
    volatile unsigned char PSCRL;        /*!< prescaler low */
    volatile unsigned char ARRH;         /*!< auto-reload register high */
    volatile unsigned char ARRL;         /*!< auto-reload register low */
    volatile unsigned char RCR;          /*!< Repetition Counter register */
    volatile unsigned char CCR1H;        /*!< capture/compare register 1 high */
    volatile unsigned char CCR1L;        /*!< capture/compare register 1 low */
    volatile unsigned char CCR2H;        /*!< capture/compare register 2 high */
    volatile unsigned char CCR2L;        /*!< capture/compare register 2 low */
    volatile unsigned char CCR3H;        /*!< capture/compare register 3 high */
    volatile unsigned char CCR3L;        /*!< capture/compare register 3 low */
    volatile unsigned char CCR4H;        /*!< capture/compare register 3 high */
    volatile unsigned char CCR4L;        /*!< capture/compare register 3 low */
    volatile unsigned char BKR;          /*!< Break Register */
    volatile unsigned char DTR;          /*!< dead-time register */
    volatile unsigned char OISR;         /*!< Output idle register */
}TIM1_TypeDef;

#define   TIM1_BaseAddress   0xFEC0
#define   TIM2_BaseAddress   0xFEE0

#define   TIM1               ((TIM1_TypeDef xdata*) TIM1_BaseAddress)
#define   TIM2               ((TIM1_TypeDef xdata*) TIM2_BaseAddress)

#define   PWMA_ETRPS         (*(unsigned char volatile xdata *) 0xFEB0)
#define   PWMA_ENO           (*(unsigned char volatile xdata *) 0xFEB1)
#define   PWMA_PS            (*(unsigned char volatile xdata *) 0xFEB2)
#define   PWMB_ENO           (*(unsigned char volatile xdata *) 0xFEB5)
#define   PWMB_PS            (*(unsigned char volatile xdata *) 0xFEB6)

sfr       ADC_CONTR    =     0xbc;
sfr       ADC_RES      =     0xbd;
sfr       ADC_RESL     =     0xbe;
sfr       ADCCFG       =     0xde;
sfr       CMPCR1       =     0xe6;
sfr       CMPCR2       =     0xe7;

sfr       AUXR         =     0x8e;

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     P5M0    =   0xca;
sfr     P5M1    =   0xc9;
sfr     P5      =   0xc8;
sfr     P_SW2   =   0xba;

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    P50     =   P5^0;
sbit    P51     =   P5^1;
sbit    P52     =   P5^2;
sbit    P53     =   P5^3;
sbit    P54     =   P5^4;
sbit    P55     =   P5^5;

#define TRUE        1
#define FALSE       0

#define RV09_CH     6
```

STC MCU

```
#define   TIM1_Period        ((u16)280)
#define   TIM1_STPulse       ((u16)245)

#define   START              0x1A
#define   RUN                0x1B
#define   STOP               0x1C
#define   IDLE               0x1D

#define   TIM1_OCMODE_MASK         ((u8)0x70)
#define   TIM1_OCCE_ENABLE         ((u8)0x80)
#define   TIM1_OCCE_DISABLE        ((u8)0x00)
#define   TIM1_OCMODE_TIMING       ((u8)0x00)
#define   TIM1_OCMODE_ACTIVE       ((u8)0x10)
#define   TIM1_OCMODE_INACTIVE     ((u8)0x20)
#define   TIM1_OCMODE_TOGGLE       ((u8)0x30)
#define   TIM1_FORCE_INACTIVE      ((u8)0x40)
#define   TIM1_FORCE_ACTIVE        ((u8)0x50)
#define   TIM1_OCMODE_PWMA         ((u8)0x60)
#define   TIM1_OCMODE_PWMB         ((u8)0x70)
#define   CC1_POLARITY_HIGH        ((u8)0x02)
#define   CC1N_POLARITY_HIGH       ((u8)0x08)
#define   CC2_POLARITY_HIGH        ((u8)0x20)
#define   CC2N_POLARITY_HIGH       ((u8)0x80)
#define   CC1_POLARITY_LOW         ((u8)~0x02)
#define   CC1N_POLARITY_LOW        ((u8)~0x08)
#define   CC2_POLARITY_LOW         ((u8)~0x20)
#define   CC2N_POLARITY_LOW        ((u8)~0x80)
#define   CC1_OCENABLE             ((u8)0x01)
#define   CC1N_OCENABLE            ((u8)0x04)
#define   CC2_OCENABLE             ((u8)0x10)
#define   CC2N_OCENABLE            ((u8)0x40)
#define   CC1_OCDISABLE            ((u8)~0x01)
#define   CC1N_OCDISABLE           ((u8)~0x04)
#define   CC2_OCDISABLE            ((u8)~0x10)
#define   CC2N_OCDISABLE           ((u8)~0x40)
#define   CC3_POLARITY_HIGH        ((u8)0x02)
#define   CC3N_POLARITY_HIGH       ((u8)0x08)
#define   CC4_POLARITY_HIGH        ((u8)0x20)
#define   CC4N_POLARITY_HIGH       ((u8)0x80)
#define   CC3_POLARITY_LOW         ((u8)~0x02)
#define   CC3N_POLARITY_LOW        ((u8)~0x08)
#define   CC4_POLARITY_LOW         ((u8)~0x20)
#define   CC4N_POLARITY_LOW        ((u8)~0x80)
#define   CC3_OCENABLE             ((u8)0x01)
#define   CC3N_OCENABLE            ((u8)0x04)
#define   CC4_OCENABLE             ((u8)0x10)
#define   CC4N_OCENABLE            ((u8)0x40)
#define   CC3_OCDISABLE            ((u8)~0x01)
#define   CC3N_OCDISABLE           ((u8)~0x04)
#define   CC4_OCDISABLE            ((u8)~0x10)
#define   CC4N_OCDISABLE           ((u8)~0x40)

void UART_INIT();
void DelayXus(unsigned char delayTime);
void DelayXms( unsigned char delayTime);
unsigned int ADC_Convert(u8 ch);
void PWM_Init(void);
void SPEED_ADJ();
```

```
unsigned char RD_HALL();
void MOTOR_START();
void MOTOR_STOP();
unsigned char KEY_detect();

unsigned char Timer0_cnt=0xb0;
unsigned int HA=0;
unsigned int Motor_speed;
unsigned char Motor_sta =  IDLE;
unsigned char BRK_occur=0;
unsigned int TIM2_CAP1_v=0;
unsigned int CAP1_avg=0;
unsigned char CAP1_cnt=0;
unsigned long CAP1_sum=0;

void main(void)
{
    unsigned int temp=0;
    unsigned int ADC_result=0;

    P_SW2= 0x80;
    P1 = 0x00;
    P0M1 = 0x0C;
    P0M0 = 0x01;
    P1M1 = 0xc0;
    P1M0 = 0x3F;
    P2M1 = 0x00;
    P2M0 = 0x38;
    P3M1 = 0x88;
    P3M0 = 0x02;

    ET0=1;
    TR0=0;
    ADCCFG = 0x0f;
    ADC_CONTR = 0x80;

    PWMA_ENO = 0x3F;                    //PWMA 输出使能
    PWMB_ENO = 0x00;                    //PWMB 输出使能
    PWMA_PS                             = 0x00; /PWMA  pin  选择
    PWMB_PS                             = 0xD5;  //PWMB pin  选择

    /*********************************************************
    输出比较模式 PWMx_duty = [CCRx/(ARR + 1)]*100
    *********************************************************/
    /************PWMB   BMF 输入 ****************/
    //////////// 时基单元 ////////////
    TIM2-> PSCRL = 15;
    TIM2-> ARRH = 0xff;                 //自动重装载寄存器，计数器 overflow  点
    TIM2-> ARRL = 0xff;
    TIM2-> CCR4H = 0x00;
    TIM2-> CCR4L = 0x05;
    //////////// 通道配置 ////////////
    TIM2-> CCMR1 = 0xf3;                //通道模式配置
    TIM2-> CCMR2 = 0xf1;
    TIM2-> CCMR3 = 0xf1;
    TIM2-> CCMR4 = 0x70;
    TIM2-> CCER1 = 0x11;
    TIM2-> CCER2 = 0x11;
    //////////// 模式配置 ////////////
```

```
    TIM2-> CR2 = 0xf0;
    TIM2-> CR1 = 0x81;
    TIM2-> SMCR = 0x44;
    //////////// 使能 & 中断配置 ////////////
    TIM2-> BKR = 0x80;                          //主输出使能
    TIM2-> IER = 0x02;                          //使能中断
    /************PWMA 控制马达换相 ****************/
    //////////// 时基单元 ////////////
    TIM1-> PSCRH = 0x00;                        //预分频寄存器
    TIM1-> PSCRL = 0x00;
    TIM1-> ARRH = (u8)(TIM1_Period >> 8);
    TIM1-> ARRL = (u8)(TIM1_Period);
    //////////// 通道配置 ////////////
    TIM1-> CCMR1 = 0x70;                        //通道模式配置
    TIM1-> CCMR2 = 0x70;
    TIM1-> CCMR3 = 0x70;
    TIM1-> CCER1 = 0x11;                        //配置通道输出使能和极性
    TIM1-> CCER2 = 0x01;                        //配置通道输出使能和极性
    TIM1-> OISR = 0xAA;                         //配置 MOE=0 时各通道输出电平
    //////////// 模式配置 ////////////
    TIM1-> CR1 = 0xA0;
    TIM1-> CR2 = 0x24;
    TIM1-> SMCR = 0x20;
    TIM1-> BKR = 0x0c;
    //////////// 使能 & 中断配置 ////////////
    TIM1-> CR1 |= 0x01;                         //使能计数器
    EA = 1;

    UART_INIT();

    while (1)
    {
        switch(Motor_sta)
        {
            case START:
                MOTOR_START();
                Motor_sta = RUN;
                for(temp = TIM1_STPulse; temp > ADC_result; temp--)        //开环启动
                {
                    ADC_result = (ADC_Convert(RV09_CH)/4);
                    TIM1-> CCR1H = (u8)(temp >> 8);
                    TIM1-> CCR1L = (u8)(temp);
                    TIM1-> CCR2H = (u8)(temp >> 8);
                    TIM1-> CCR2L = (u8)(temp);
                    TIM1-> CCR3H = (u8)(temp >> 8);
                    TIM1-> CCR3L = (u8)(temp);
                    DelayXms(10);
                }
                break;
            case RUN:
                SPEED_ADJ();                    //马达调速
                if((BRK_occur == TRUE))
                Motor_sta = STOP;
                break;
            case STOP:
                MOTOR_STOP();
                Motor_sta = IDLE;
                break;
            case IDLE:
```

```
                    if(KEY_detect()==1)
                    Motor_sta = START;                    //启动马达
                    BRK_occur = FALSE;
                    Motor_speed = 0;
                    CAP1_avg = 0;
                    CAP1_cnt = 0;
                    CAP1_sum = 0;
                    break;
            }
        }
}

void TIM0_ISR() interrupt 1
{
        if(Motor_sta ==  START)
        {
                if(Timer0_cnt<0xe0) Timer0_cnt++;
                TH0=Timer0_cnt;

                switch(HA%6)
                {
                case 0:
                        TIM1-> CCMR3 &= ~TIM1_OCMODE_MASK;
                        TIM1-> CCMR3 |= TIM1_FORCE_INACTIVE;
                        TIM1-> CCMR1 &= ~TIM1_OCMODE_MASK;
                        TIM1-> CCMR1 |= TIM1_OCMODE_PWMB;
                        break;
                case 1:
                        TIM1-> CCER1 &= CC2N_POLARITY_LOW;
                        TIM1-> CCER2 |= CC3N_POLARITY_HIGH;
                         break;
                case 2:
                        TIM1-> CCMR1 &= ~TIM1_OCMODE_MASK;
                        TIM1-> CCMR1 |= TIM1_FORCE_INACTIVE;
                        TIM1-> CCMR2 &= ~TIM1_OCMODE_MASK;
                        TIM1-> CCMR2 |= TIM1_OCMODE_PWMB;
                        break;
                case 3:
                        TIM1-> CCER1 |= CC1N_POLARITY_HIGH;
                        TIM1-> CCER2 &= CC3N_POLARITY_LOW;
                        break;
                case 4:
                        TIM1-> CCMR2 &= ~TIM1_OCMODE_MASK;
                        TIM1-> CCMR2 |= TIM1_FORCE_INACTIVE;
                        TIM1-> CCMR3 &= ~TIM1_OCMODE_MASK;
                        TIM1-> CCMR3 |= TIM1_OCMODE_PWMB;
                        break;
                case 5:
                        TIM1-> CCER1 &= CC1N_POLARITY_LOW;
                        TIM1-> CCER1 |= CC2N_POLARITY_HIGH;
                        break;
                }
                HA++;
        }

        if(Motor_sta == RUN)
        {
                TR0=0;
                switch(RD_HALL())
```

```
            {
            case 3:
                TIM1-> CCMR3 &= ~TIM1_OCMODE_MASK;
                TIM1-> CCMR3 |= TIM1_FORCE_INACTIVE;
                TIM1-> CCMR1 &= ~TIM1_OCMODE_MASK;
                TIM1-> CCMR1 |= TIM1_OCMODE_PWMB;
                break;
            case 1:
                TIM1-> CCER1 &= CC2N_POLARITY_LOW;
                TIM1-> CCER2 |= CC3N_POLARITY_HIGH;
                break;
            case 5:
                TIM1-> CCMR1 &= ~TIM1_OCMODE_MASK;
                TIM1-> CCMR1 |= TIM1_FORCE_INACTIVE;
                TIM1-> CCMR2 &= ~TIM1_OCMODE_MASK;
                TIM1-> CCMR2 |= TIM1_OCMODE_PWMB;
                break;
            case 4:
                TIM1-> CCER1 |= CC1N_POLARITY_HIGH;
                TIM1-> CCER2 &= CC3N_POLARITY_LOW;
                break;
            case 6:
                TIM1-> CCMR2 &= ~TIM1_OCMODE_MASK;
                TIM1-> CCMR2 |= TIM1_FORCE_INACTIVE;
                TIM1-> CCMR3 &= ~TIM1_OCMODE_MASK;
                TIM1-> CCMR3 |= TIM1_OCMODE_PWMB;
                break;
            case 2:
                TIM1-> CCER1 &= CC1N_POLARITY_LOW;
                TIM1-> CCER1 |= CC2N_POLARITY_HIGH;
                break;
            }
        }
}

void PWMA_ISR() interrupt 26
{
        if((TIM1->SR1 & 0x20))
        {
            P00=0;
            CAP1_sum += TIM2_CAP1_v;
            CAP1_cnt++;
            if(CAP1_cnt==128)
            {
                CAP1_cnt=0;
                CAP1_avg = (CAP1_sum>>7);
                CAP1_sum = 0;
                Motor_speed = 5000000/CAP1_avg;
            }
            TIM1->SR1 &=~0x20;                          //清零
        }
        if((TIM1->SR1 & 0x80))                          //BRK
        {
            BRK_occur = TRUE;
            TIM1->SR1 &=~0x80;                          //清零
        }
}

void PWMB_ISR() interrupt 27
```

```
{
        unsigned char ccr_tmp=0;

        if((TIM2->SR1 & 0X02))
        {
                ccr_tmp = TIM2-> CCR1H;
                if(ccr_tmp>1)                                   //软件滤波
                {
                        TIM2_CAP1_v = ccr_tmp;
                        TIM2_CAP1_v = (TIM2_CAP1_v<<8) + TIM2->CCR1L;
                        if(Motor_sta == RUN)                    //换向 delay 计时
                        {
                                TR0=1;
                                TH0 = 256-(TIM2_CAP1_v>>9);
                        }
                }
                TIM2->SR1 &=~0X02;
        }
}

void UART_INIT()
{
        SCON = 0x50;                                            //8 位可变波特率
        AUXR = 0x40;                                            //定时器 1 为 1T 模式
        TMOD = 0x20;                                            //定时器 1 为模式 0(16 位自动重载)

        TL1 = 254;
        TH1 = 254;
//      ET1 = 0;
        TR1 = 1;
}

void DelayXus(unsigned char delayTime)
{
        int i = 0;
        while( delayTime--)
        {
                for( i = 0 ; i < 1 ; i++);
        }
}

void DelayXms( unsigned char delayTime )
{
        int i = 0;
        while( delayTime--)
        {
                for( i = 0 ; i < 2 ; i++)
                {
                        DelayXus(100);
                }
        }
}

unsigned int ADC_Convert(u8 ch)
{
        u16 res=0;

        ADC_CONTR &= ~0x0f;
        ADC_CONTR |= ch;
```

```
    ADC_CONTR |= 0x40;
    DelayXus(1);
    while (!(ADC_CONTR & 0x20));
    ADC_CONTR &= ~0x20;

    res = ADC_RES;
    res = (res<<2)+(ADC_RESL>>6);

    if (res < 360) res=360;
    if (res > 900) res=900;

    return res;
}

void SPEED_ADJ()
{
    u16 ADC_result;

    ADC_result = (ADC_Convert(RV09_CH)/4);          //调速旋钮 ADC 采样
    TIM1-> CCR1H = (u8)(ADC_result >> 8);           //计数器比较值
    TIM1-> CCR1L = (u8)(ADC_result);
    TIM1-> CCR2H = (u8)(ADC_result >> 8);
    TIM1-> CCR2L = (u8)(ADC_result);
    TIM1-> CCR3H = (u8)(ADC_result >> 8);
    TIM1-> CCR3L = (u8)(ADC_result);
}

unsigned char RD_HALL()                             //读霍尔传感器
{
    unsigned char Hall_sta = 0;

    DelayXus(40);
    (P17)? (Hall_sta|=0x01) : (Hall_sta&=~0x01);
    (P54)? (Hall_sta|=0x02) : (Hall_sta&=~0x02);
    (P33)? (Hall_sta|=0x04) : (Hall_sta&=~0x04);

    return Hall_sta;
}

void MOTOR_START()
{
    TIM1-> CCR1H = (u8)(TIM1_STPulse >> 8);         //计数器比较值
    TIM1-> CCR1L = (u8)(TIM1_STPulse);
    TIM1-> CCR2H = (u8)(TIM1_STPulse >> 8);
    TIM1-> CCR2L = (u8)(TIM1_STPulse);
    TIM1-> CCR3H = (u8)(TIM1_STPulse >> 8);
    TIM1-> CCR3L = (u8)(TIM1_STPulse);
    TIM1-> BKR |= 0x80;                             //主输出使能相当于总开关
    TIM1-> IER = 0x00;                              //使能中断
    TR0 = 1;

    while (HA < 6*20);

    TIM1-> IER = 0xa0;                              //使能中断
}

void MOTOR_STOP()
{
    TIM1-> BKR &= ~0x80;
```

```
    TIM1-> IER &= ~0x20;
}

unsigned char KEY_detect()
{
    if(!P37)
    {
        DelayXms(10);
        if(!P37)
        {
            return 1;
        }
        else return 0;
    }
    else if(!P03)
    {
        DelayXms(10);
        if(!P03)
        {
            return 2;
        }
        else return 0;
    }
    else return 0;
}
```

21.8.3 正交编码器模式

C 语言代码

```
//测试工作频率为 11.0592MHz

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

typedef struct TIM1_struct
{
    volatile unsigned char CR1;         /*!< control register 1 */
    volatile unsigned char CR2;         /*!< control register 2 */
    volatile unsigned char SMCR;        /*!< Synchro mode control register */
    volatile unsigned char ETR;         /*!< external trigger register */
    volatile unsigned char IER;         /*!< interrupt enable register*/
    volatile unsigned char SR1;         /*!< status register 1 */
    volatile unsigned char SR2;         /*!< status register 2 */
    volatile unsigned char EGR;         /*!< event generation register */
    volatile unsigned char CCMR1;       /*!< CC mode register 1 */
    volatile unsigned char CCMR2;       /*!< CC mode register 2 */
    volatile unsigned char CCMR3;       /*!< CC mode register 3 */
    volatile unsigned char CCMR4;       /*!< CC mode register 4 */
    volatile unsigned char CCER1;       /*!< CC enable register 1 */
    volatile unsigned char CCER2;       /*!< CC enable register 2 */
    volatile unsigned char CNTRH;       /*!< counter high */
    volatile unsigned char CNTRL;       /*!< counter low */
    volatile unsigned char PSCRH;       /*!< prescaler high */
    volatile unsigned char PSCRL;       /*!< prescaler low */
    volatile unsigned char ARRH;        /*!< auto-reload register high */
    volatile unsigned char ARRL;        /*!< auto-reload register low */
    volatile unsigned char RCR;         /*!< Repetition Counter register */
```

```
    volatile unsigned char CCR1H;                    /*!< capture/compare register 1 high */
    volatile unsigned char CCR1L;                    /*!< capture/compare register 1 low */
    volatile unsigned char CCR2H;                    /*!< capture/compare register 2 high */
    volatile unsigned char CCR2L;                    /*!< capture/compare register 2 low */
    volatile unsigned char CCR3H;                    /*!< capture/compare register 3 high */
    volatile unsigned char CCR3L;                    /*!< capture/compare register 3 low */
    volatile unsigned char CCR4H;                    /*!< capture/compare register 3 high */
    volatile unsigned char CCR4L;                    /*!< capture/compare register 3 low */
    volatile unsigned char BKR;                      /*!< Break Register */
    volatile unsigned char DTR;                      /*!< dead-time register */
    volatile unsigned char OISR;                     /*!< Output idle register */
}TIM1_TypeDef;

#define     TIM1_BaseAddress    0xFEC0

#define     TIM1                ((TIM1_TypeDef xdata*)TIM1_BaseAddress)
#define     PWMA_ENO            (*(unsigned char volatile xdata *)0xFEB1)
#define     PWMA_PS             (*(unsigned char volatile xdata *)0xFEB2)

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

sbit        P03         =       P0^3;

unsigned char cnt_H, cnt_L;

void main(void)
{
    P_SW2 = 0x80;

    P1M1 = 0x0f;
    P1M0 = 0x00;

    PWMA_ENO = 0x00;                                 //配置成 TRGI 的 pin 需关掉 ENO 对应 bit 并配成 input
    PWMA_PS = 0x00;                                  //00:PWM at P1

    TIM1-> PSCRH = 0x00;                             //预分频寄存器
    TIM1-> PSCRL = 0x00;

    TIM1-> CCMR1   = 0x21;                           //通道模式配置为输入，接编码器,滤波器 4 时钟
    TIM1-> CCMR2   = 0x21;                           //通道模式配置为输入，接编码器,滤波器 4 时钟

    TIM1-> SMCR    = 0x03;                           //编码器模式 3

    TIM1-> CCER1   = 0x55;                           //配置通道使能和极性
    TIM1-> CCER2   = 0x55;                           //配置通道使能和极性

    TIM1-> IER   = 0x02;                             //使能中断

    TIM1-> CR1 |= 0x01;                              //使能计数器

    EA = 1;

    while (1);
}
```

```
/******************* PWM 中断读编码器计数值*************************/
void PWMA_ISR() interrupt 26
{
	if (TIM1->SR1 & 0X02)
	{
		P03 = ~P03;
		cnt_H = TIM1->CCR1H;
		cnt_L = TIM1->CCR1L;
		TIM1->SR1 & = ~0X02;
	}
}
```

21.8.4 单脉冲模式（触发控制脉冲输出）

C 语言代码

```
//测试工作频率为 11.0592MHz

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

typedef struct TIM1_struct
{
	volatile unsigned char CR1;		/*!< control register 1 */
	volatile unsigned char CR2;		/*!< control register 2 */
	volatile unsigned char SMCR;		/*!< Synchro mode control register */
	volatile unsigned char ETR;		/*!< external trigger register */
	volatile unsigned char IER;		/*!< interrupt enable register*/
	volatile unsigned char SR1;		/*!< status register 1 */
	volatile unsigned char SR2;		/*!< status register 2 */
	volatile unsigned char EGR;		/*!< event generation register */
	volatile unsigned char CCMR1;		/*!< CC mode register 1 */
	volatile unsigned char CCMR2;		/*!< CC mode register 2 */
	volatile unsigned char CCMR3;		/*!< CC mode register 3 */
	volatile unsigned char CCMR4;		/*!< CC mode register 4 */
	volatile unsigned char CCER1;		/*!< CC enable register 1 */
	volatile unsigned char CCER2;		/*!< CC enable register 2 */
	volatile unsigned char CNTRH;		/*!< counter high */
	volatile unsigned char CNTRL;		/*!< counter low */
	volatile unsigned char PSCRH;		/*!< prescaler high */
	volatile unsigned char PSCRL;		/*!< prescaler low */
	volatile unsigned char ARRH;		/*!< auto-reload register high */
	volatile unsigned char ARRL;		/*!< auto-reload register low */
	volatile unsigned char RCR;		/*!< Repetition Counter register */
	volatile unsigned char CCR1H;		/*!< capture/compare register 1 high */
	volatile unsigned char CCR1L;		/*!< capture/compare register 1 low */
	volatile unsigned char CCR2H;		/*!< capture/compare register 2 high */
	volatile unsigned char CCR2L;		/*!< capture/compare register 2 low */
	volatile unsigned char CCR3H;		/*!< capture/compare register 3 high */
	volatile unsigned char CCR3L;		/*!< capture/compare register 3 low */
	volatile unsigned char CCR4H;		/*!< capture/compare register 3 high */
	volatile unsigned char CCR4L;		/*!< capture/compare register 3 low */
	volatile unsigned char BKR;		/*!< Break Register */
	volatile unsigned char DTR;		/*!< dead-time register */
	volatile unsigned char OISR;		/*!< Output idle register */
}TIM1_TypeDef;

#define	TIM1_BaseAddress	0xFEC0
```

```
#define   TIM1          ((TIM1_TypeDef xdata*)TIM1_BaseAddress)
#define   PWMA_ENO      (*(unsigned char volatile xdata *)0xFEB1)
#define   PWMA_PS       (*(unsigned char volatile xdata *)0xFEB2)

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

sbit      P03       =   P0^3;

void main(void)
{
    P_SW2 = 0x80;

    P0M1 = 0x00;
    P0M0 = 0xFF;
    P1M1 = 0x0c;
    P1M0 = 0xF3;

    PWMA_ENO = 0xF3;                    //IO 输出 PWM
    PWMA_PS = 0x00;                     //00:PWM at P1

    /*********************************************************
    PWMx_duty = [CCRx/(ARR + 1)]*100
    *********************************************************/
    //配置成 TRGI 的 pin 需关掉 ENO 对应 bit 并配成 input
    TIM1-> PSCRH = 0x00;                //预分频寄存器
    TIM1-> PSCRL = 0x00;
    TIM1-> DTR = 0x00;                  //死区时间配置

    TIM1-> CCMR1 = 0x68;                //通道模式配置
    TIM1-> CCMR2 = 0x01;                //配置成输入通道
    TIM1-> CCMR3 = 0x68;
    TIM1-> CCMR4 = 0x68;

    TIM1-> SMCR   = 0x66;

    TIM1-> ARRH = 0x08;                 //自动重装载寄存器，计数器 overflow 点
    TIM1-> ARRL = 0x00;

    TIM1-> CCR1H = 0x04;                //计数器比较值
    TIM1-> CCR1L = 0x00;
    TIM1-> CCR2H = 0x02;
    TIM1-> CCR2L = 0x00;
    TIM1-> CCR3H = 0x01;
    TIM1-> CCR3L = 0x00;
    TIM1-> CCR4H = 0x01;
    TIM1-> CCR4L = 0x00;

    TIM1-> CCER1 = 0x55;                //配置通道输出使能和极性
    TIM1-> CCER2 = 0x55;                //配置通道输出使能和极性

    TIM1-> BKR = 0x80;                  //主输出使能 相当于总开关
    TIM1-> IER = 0x02;                  //使能中断
    TIM1-> CR1 = 0x08;                  //单脉冲模式
    TIM1-> CR1 |= 0x01;                 //使能计数器
```

```
    EA = 1;
    while (1);
}

void PWMA_ISR() interrupt 26
{
    if (TIM1->SR1 & 0X02)
    {
        P03 = ~P03;
        TIM1->SR1 & = ~0X02;
    }
}
```

21.8.5 门控模式（输入电平使能计数器）

C 语言代码

```
//测试工作频率为 11.0592MHz

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

typedef struct TIM1_struct
{
    volatile unsigned char CR1;         /*!< control register 1 */
    volatile unsigned char CR2;         /*!< control register 2 */
    volatile unsigned char SMCR;        /*!< Synchro mode control register */
    volatile unsigned char ETR;         /*!< external trigger register */
    volatile unsigned char IER;         /*!< interrupt enable register*/
    volatile unsigned char SR1;         /*!< status register 1 */
    volatile unsigned char SR2;         /*!< status register 2 */
    volatile unsigned char EGR;         /*!< event generation register */
    volatile unsigned char CCMR1;       /*!< CC mode register 1 */
    volatile unsigned char CCMR2;       /*!< CC mode register 2 */
    volatile unsigned char CCMR3;       /*!< CC mode register 3 */
    volatile unsigned char CCMR4;       /*!< CC mode register 4 */
    volatile unsigned char CCER1;       /*!< CC enable register 1 */
    volatile unsigned char CCER2;       /*!< CC enable register 2 */
    volatile unsigned char CNTRH;       /*!< counter high */
    volatile unsigned char CNTRL;       /*!< counter low */
    volatile unsigned char PSCRH;       /*!< prescaler high */
    volatile unsigned char PSCRL;       /*!< prescaler low */
    volatile unsigned char ARRH;        /*!< auto-reload register high */
    volatile unsigned char ARRL;        /*!< auto-reload register low */
    volatile unsigned char RCR;         /*!< Repetition Counter register */
    volatile unsigned char CCR1H;       /*!< capture/compare register 1 high */
    volatile unsigned char CCR1L;       /*!< capture/compare register 1 low */
    volatile unsigned char CCR2H;       /*!< capture/compare register 2 high */
    volatile unsigned char CCR2L;       /*!< capture/compare register 2 low */
    volatile unsigned char CCR3H;       /*!< capture/compare register 3 high */
    volatile unsigned char CCR3L;       /*!< capture/compare register 3 low */
    volatile unsigned char CCR4H;       /*!< capture/compare register 3 high */
    volatile unsigned char CCR4L;       /*!< capture/compare register 3 low */
    volatile unsigned char BKR;         /*!< Break Register */
    volatile unsigned char DTR;         /*!< dead-time register */
    volatile unsigned char OISR;        /*!< Output idle register */
}TIM1_TypeDef;
```

STC MCU

```
#define     TIM1_BaseAddress    0xFEC0

#define     TIM1                ((TIM1_TypeDef xdata*)TIM1_BaseAddress)
#define     PWMA_ENO            (*(unsigned char volatile xdata *)0xFEB1)
#define     PWMA_PS             (*(unsigned char volatile xdata *)0xFEB2)

sfr         P0M0        =       0x94;
sfr         P0M1        =       0x93;
sfr         P1M0        =       0x92;
sfr         P1M1        =       0x91;
sfr         P3M0        =       0xb2;
sfr         P3M1        =       0xb1;
sfr         P_SW2       =       0xba;

sbit        P03         =       P0^3;

void main(void)
{
        P_SW2 = 0x80;

        P0M1 = 0x00;
        P0M0 = 0xFF;
        P1M1 = 0x00;
        P1M0 = 0xFF;
        P3M1 = 0x04;
        P3M0 = 0x00;

        PWMA_ENO = 0xFF;                        //IO 输出 PWM
        PWMA_PS = 0x00;                         //00:PWM at P1

/*********************************************************
PWMx_duty = [CCRx/(ARR + 1)]*100
*********************************************************/
//配置成 TRGI 的 pin 需关掉 ENO 对应 bit 并配成 input
        TIM1-> PSCRH = 0x00;                    //预分频寄存器
        TIM1-> PSCRL = 0x00;
        TIM1-> DTR = 0x00;                      //死区时间配置

        TIM1-> CCMR1 = 0x68;                    //通道模式配置
        TIM1-> CCMR2 = 0x68;                    //配置成输入通道
        TIM1-> CCMR3 = 0x68;
        TIM1-> CCMR4 = 0x68;

        TIM1-> SMCR = 0x75;                     //门控触发模式 ETRF 输入

        TIM1-> ARRH = 0x08;                     //自动重装载寄存器，计数器 overflow 点
        TIM1-> ARRL = 0x00;

        TIM1-> CCR1H = 0x04;                    //计数器比较值
        TIM1-> CCR1L = 0x00;                    //
        TIM1-> CCR2H = 0x02;                    //
        TIM1-> CCR2L = 0x00;                    //
        TIM1-> CCR3H = 0x01;                    //
        TIM1-> CCR3L = 0x00;                    //
        TIM1-> CCR4H = 0x01;                    //
        TIM1-> CCR4L = 0x00;                    //

        TIM1-> CCER1 = 0x55;                    //配置通道输出使能和极性
        TIM1-> CCER2 = 0x55;                    //配置通道输出使能和极性
```

```
    TIM1-> BKR = 0x80;                          //主输出使能 相当于总开关
    TIM1-> IER = 0x02;                          //使能中断

    TIM1-> CR1 |= 0x01;                         //使能计数器

    EA = 1;
    while (1) ;
}

void PWMA_ISR() interrupt 26
{
    if(TIM1->SR1 & 0X02)
    {
        P03 = ~P03;
        TIM1->SR1 &=~0X02;
    }
}
```

21.8.6 外部时钟模式

C 语言代码

```
//测试工作频率为 11.0592MHz

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

typedef struct TIM1_struct
{
    volatile unsigned char CR1;             /*!< control register 1 */
    volatile unsigned char CR2;             /*!< control register 2 */
    volatile unsigned char SMCR;            /*!< Synchro mode control register */
    volatile unsigned char ETR;             /*!< external trigger register */
    volatile unsigned char IER;             /*!< interrupt enable register*/
    volatile unsigned char SR1;             /*!< status register 1 */
    volatile unsigned char SR2;             /*!< status register 2 */
    volatile unsigned char EGR;             /*!< event generation register */
    volatile unsigned char CCMR1;           /*!< CC mode register 1 */
    volatile unsigned char CCMR2;           /*!< CC mode register 2 */
    volatile unsigned char CCMR3;           /*!< CC mode register 3 */
    volatile unsigned char CCMR4;           /*!< CC mode register 4 */
    volatile unsigned char CCER1;           /*!< CC enable register 1 */
    volatile unsigned char CCER2;           /*!< CC enable register 2 */
    volatile unsigned char CNTRH;           /*!< counter high */
    volatile unsigned char CNTRL;           /*!< counter low */
    volatile unsigned char PSCRH;           /*!< prescaler high */
    volatile unsigned char PSCRL;           /*!< prescaler low */
    volatile unsigned char ARRH;            /*!< auto-reload register high */
    volatile unsigned char ARRL;            /*!< auto-reload register low */
    volatile unsigned char RCR;             /*!< Repetition Counter register */
    volatile unsigned char CCR1H;           /*!< capture/compare register 1 high */
    volatile unsigned char CCR1L;           /*!< capture/compare register 1 low */
    volatile unsigned char CCR2H;           /*!< capture/compare register 2 high */
    volatile unsigned char CCR2L;           /*!< capture/compare register 2 low */
    volatile unsigned char CCR3H;           /*!< capture/compare register 3 high */
    volatile unsigned char CCR3L;           /*!< capture/compare register 3 low */
    volatile unsigned char CCR4H;           /*!< capture/compare register 3 high */
    volatile unsigned char CCR4L;           /*!< capture/compare register 3 low */
```

STC MCU

```
    volatile unsigned char BKR;                     /*!< Break Register */
    volatile unsigned char DTR;                     /*!< dead-time register */
    volatile unsigned char OISR;                    /*!< Output idle register */
}TIM1_TypeDef;

#define   TIM1_BaseAddress   0xFEC0

#define   TIM1               ((TIM1_TypeDef xdata*)TIM1_BaseAddress)
#define   PWMA_ENO           (*(unsigned char volatile xdata *)0xFEB1)
#define   PWMA_PS            (*(unsigned char volatile xdata *)0xFEB2)

sfr       P0M0       =   0x94;
sfr       P0M1       =   0x93;
sfr       P1M0       =   0x92;
sfr       P1M1       =   0x91;
sfr       P3M0       =   0xb2;
sfr       P3M1       =   0xb1;
sfr       P_SW2      =   0xba;

sbit      P03        =   P0^3;

void main(void)
{
    P_SW2 = 0x80;

    P0M1 = 0x00;
    P0M0 = 0xFF;
    P1M1 = 0x00;
    P1M0 = 0xFF;
    P3M1 = 0x04;
    P3M0 = 0x00;

    PWMA_ENO = 0xFF;                                //IO 输出 PWM
    PWMA_PS = 0x00;                                 //00:PWM at P1

/*********************************************************
PWMx_duty = [CCRx/(ARR + 1)]*100
*********************************************************/
//配置成 TRGI 的 pin 需关掉 ENO 对应 bit 并配成 input
    TIM1-> PSCRH = 0x00;                            //预分频寄存器
    TIM1-> PSCRL = 0x00;
    TIM1-> DTR = 0x00;                              //死区时间配置

    TIM1-> CCMR1 = 0x68;                            //通道模式配置
    TIM1-> CCMR2 = 0x68;                            //配置成输入通道
    TIM1-> CCMR3 = 0x68;
    TIM1-> CCMR4 = 0x68;

    TIM1-> SMCR = 0x77;                             //ETRF 输入

    TIM1-> ARRH = 0x08;                             //自动重装载寄存器，计数器 overflow 点
    TIM1-> ARRL = 0x00;

    TIM1-> CCR1H = 0x04;                            //计数器比较值
    TIM1-> CCR1L = 0x00;
    TIM1-> CCR2H = 0x02;
    TIM1-> CCR2L = 0x00;
    TIM1-> CCR3H = 0x01;
    TIM1-> CCR3L = 0x00;
```

```
    TIM1-> CCR4H = 0x01;
    TIM1-> CCR4L = 0x00;

    TIM1-> CCER1 = 0x55;                    //配置通道输出使能和极性
    TIM1-> CCER2 = 0x55;                    //配置通道输出使能和极性

    TIM1-> BKR = 0x80;                      //主输出使能 相当于总开关
    TIM1-> IER = 0x02;                      //使能中断
    TIM1-> CR1 |= 0x01;                     //使能计数器

    EA = 1;
    while (1);
}

void PWMA_ISR() interrupt 26
{
    if(TIM1->SR1 & 0X02)
    {
        P03 = ~P03;
        TIM1->SR1 &=~0X02;
    }
}
```

21.8.7 输入捕获模式测量脉冲周期（捕获上升沿到上升沿或者下降沿到下降沿）

C 语言代码

```
//测试工作频率为 11.0592MHz

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

typedef struct TIM1_struct
{
    volatile unsigned char CR1;             /*!< control register 1 */
    volatile unsigned char CR2;             /*!< control register 2 */
    volatile unsigned char SMCR;            /*!< Synchro mode control register */
    volatile unsigned char ETR;             /*!< external trigger register */
    volatile unsigned char IER;             /*!< interrupt enable register*/
    volatile unsigned char SR1;             /*!< status register 1 */
    volatile unsigned char SR2;             /*!< status register 2 */
    volatile unsigned char EGR;             /*!< event generation register */
    volatile unsigned char CCMR1;           /*!< CC mode register 1 */
    volatile unsigned char CCMR2;           /*!< CC mode register 2 */
    volatile unsigned char CCMR3;           /*!< CC mode register 3 */
    volatile unsigned char CCMR4;           /*!< CC mode register 4 */
    volatile unsigned char CCER1;           /*!< CC enable register 1 */
    volatile unsigned char CCER2;           /*!< CC enable register 2 */
    volatile unsigned char CNTRH;           /*!< counter high */
    volatile unsigned char CNTRL;           /*!< counter low */
    volatile unsigned char PSCRH;           /*!< prescaler high */
    volatile unsigned char PSCRL;           /*!< prescaler low */
    volatile unsigned char ARRH;            /*!< auto-reload register high */
    volatile unsigned char ARRL;            /*!< auto-reload register low */
    volatile unsigned char RCR;             /*!< Repetition Counter register */
    volatile unsigned char CCR1H;           /*!< capture/compare register 1 high */
```

```
    volatile unsigned char CCR1L;                    /*!< capture/compare register 1 low */
    volatile unsigned char CCR2H;                    /*!< capture/compare register 2 high */
    volatile unsigned char CCR2L;                    /*!< capture/compare register 2 low */
    volatile unsigned char CCR3H;                    /*!< capture/compare register 3 high */
    volatile unsigned char CCR3L;                    /*!< capture/compare register 3 low */
    volatile unsigned char CCR4H;                    /*!< capture/compare register 3 high */
    volatile unsigned char CCR4L;                    /*!< capture/compare register 3 low */
    volatile unsigned char BKR;                      /*!< Break Register */
    volatile unsigned char DTR;                      /*!< dead-time register */
    volatile unsigned char OISR;                     /*!< Output idle register */
}TIM1_TypeDef;

#define    TIM1_BaseAddress    0xFEC0

#define    TIM1                ((TIM1_TypeDef xdata*)TIM1_BaseAddress)
#define    PWMA_ENO            (*(unsigned char volatile xdata *)0xFEB1)
#define    PWMA_PS             (*(unsigned char volatile xdata *)0xFEB2)

sfr        P0M0        =       0x94;
sfr        P0M1        =       0x93;
sfr        P1M0        =       0x92;
sfr        P1M1        =       0x91;
sfr        P3M0        =       0xb2;
sfr        P3M1        =       0xb1;
sfr        P_SW2       =       0xba;

sbit       P03         =       P0^3;

int        cap;

void main(void)
{
    P_SW2 = 0x80;

    P0M1 = 0x00;
    P0M0 = 0xFF;
    P1M1 = 0x0c;
    P1M0 = 0xF3;

    PWMA_ENO = 0xF3;                                 //IO 输出 PWM
    PWMA_PS = 0x00;                                  //00:PWM at P1

/*配置成 TRGI 的 pin 需关掉 ENO 对应 bit 并配成 input*/
    TIM1-> PSCRH = 0x00;                             //预分频寄存器
    TIM1-> PSCRL = 0x00;
    TIM1-> DTR = 0x00;                               //死区时间配置

    TIM1-> CCMR1 = 0x68;                             //通道模式配置
    TIM1-> CCMR2 = 0x01;                             //配置成输入通道
    TIM1-> CCMR3 = 0x68;
    TIM1-> CCMR4 = 0x68;

    TIM1-> SMCR = 0x66;

    TIM1-> CCER1 = 0x55;                             //配置通道输出使能和极性
    TIM1-> CCER2 = 0x55;                             //配置通道输出使能和极性

    TIM1-> IER = 0x04;                               //使能中断
```

```
    TIM1-> CR1 |= 0x01;                                  //使能计数器

    EA = 1;
    while (1);
}

/*通道 2 输入，捕获数据通过 TIM1-> CCR2H / TIM1-> CCR2L  读取 */
void PWMA_ISR() interrupt 26
{
    if(TIM1->SR1 & 0X02)
    {
        P03 = ~P03;
        TIM1->SR1 &=~0X02;
    }
    if(TIM1->SR1 & 0X04)
    {
        P03 = ~P03;
        cap = TIM1-> CCR2H;                              //读取 CCR2H
        cap = (cap << 8) + TIM1-> CCR2L;                 //读取 CCR2L
        TIM1->SR1 &=~0X04;
    }
}
```

21.8.8 输入捕获模式测量脉冲高电平宽度（捕获上升沿到下降沿）

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     P1M0        =   0x92;
sfr     P1M1        =   0x91;
sfr     P3M0        =   0xb2;
sfr     P3M1        =   0xb1;
sfr     P5M0        =   0xca;
sfr     P5M1        =   0xc9;

#define PWMA_CR1        (*(unsigned char volatile xdata *)0xfec0)
#define PWMA_IER        (*(unsigned char volatile xdata *)0xfec4)
#define PWMA_SR1        (*(unsigned char volatile xdata *)0xfec5)
#define PWMA_CCMR1      (*(unsigned char volatile xdata *)0xfec8)
#define PWMA_CCMR2      (*(unsigned char volatile xdata *)0xfec9)
#define PWMA_CCER1      (*(unsigned char volatile xdata *)0xfecc)
#define PWMA_CCR1       (*(unsigned int volatile xdata *)0xfed5)
#define PWMA_CCR2       (*(unsigned int volatile xdata *)0xfed7)

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;
```

```
                                                    //(CC1 捕获 TI1 上升沿,CC2 捕获 TI1 下降沿)
    PWMA_CCER1 = 0x00;
    PWMA_CCMR1 = 0x01;                              //CC1 为输入模式,且映射到 TI1FP1 上
    PWMA_CCMR2 = 0x02;                              //CC2 为输入模式,且映射到 TI1FP2 上
    PWMA_CCER1 = 0x11;                              //使能 CC1/CC2 上的捕获功能
    PWMA_CCER1 |= 0x00;                             //设置捕获极性为 CC1 的上升沿
    PWMA_CCER1 |= 0x20;                             //设置捕获极性为 CC2 的下降沿
    PWMA_CR1 = 0x01;

    PWMA_IER = 0x04;                                //使能 CC2 捕获中断
    EA = 1;

    while (1);
}

void PWMA_ISR() interrupt 26
{
    unsigned int cnt;

    if (PWMA_SR1 & 0x04)
    {
        PWMA_SR1 &= ~0x04;

        cnt = PWMA_CCR2 - PWMA_CCR1;                //差值即为高电平宽度
    }
}
```

21.8.9 输入捕获模式测量脉冲低电平宽度（捕获下降沿到上升沿）

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     P1M0        =   0x92;
sfr     P1M1        =   0x91;
sfr     P3M0        =   0xb2;
sfr     P3M1        =   0xb1;
sfr     P5M0        =   0xca;
sfr     P5M1        =   0xc9;

#define PWMA_CR1        (*(unsigned char volatile xdata *)0xfec0)
#define PWMA_IER        (*(unsigned char volatile xdata *)0xfec4)
#define PWMA_SR1        (*(unsigned char volatile xdata *)0xfec5)
#define PWMA_CCMR1      (*(unsigned char volatile xdata *)0xfec8)
#define PWMA_CCMR2      (*(unsigned char volatile xdata *)0xfec9)
#define PWMA_CCER1      (*(unsigned char volatile xdata *)0xfecc)
#define PWMA_CCR1       (*(unsigned int volatile xdata *)0xfed5)
#define PWMA_CCR2       (*(unsigned int volatile xdata *)0xfed7)

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
```

```
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;

                                                    //(CC1 捕获 TI1 上升沿,CC2 捕获 TI1 下降沿)
    PWMA_CCER1 = 0x00;
    PWMA_CCMR1 = 0x01;                              //CC1 为输入模式,且映射到 TI1FP1 上
    PWMA_CCMR2 = 0x02;                              //CC2 为输入模式,且映射到 TI1FP2 上
    PWMA_CCER1 = 0x11;                              //使能 CC1/CC2 上的捕获功能
    PWMA_CCER1 |= 0x00;                             //设置捕获极性为 CC1 的上升沿
    PWMA_CCER1 |= 0x20;                             //设置捕获极性为 CC2 的下降沿
    PWMA_CR1 = 0x01;

    PWMA_IER = 0x02;                                //使能 CC1 捕获中断
    EA = 1;

    while (1);
}

void PWMA_ISR() interrupt 26
{
    unsigned int cnt;

    if (PWMA_SR1 & 0x02)
    {
        PWMA_SR1 &= ~0x02;

        cnt = PWMA_CCR1 - PWMA_CCR2;                //差值即为低电平宽度
    }
}
```

21.8.10输入捕获模式同时测量脉冲周期和占空比

注意：只有 PWM1P、PWM2P、PWM5、PWM6 这些端口上才可同时测量周期和占空比

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     P1M0      =   0x92;
sfr     P1M1      =   0x91;
sfr     P3M0      =   0xb2;
sfr     P3M1      =   0xb1;
sfr     P5M0      =   0xca;
sfr     P5M1      =   0xc9;

#define PWMA_CR1      (*(unsigned char volatile xdata *)0xfec0)
#define PWMA_SMCR     (*(unsigned char volatile xdata *)0xfec2)
#define PWMA_IER      (*(unsigned char volatile xdata *)0xfec4)
#define PWMA_SR1      (*(unsigned char volatile xdata *)0xfec5)
#define PWMA_CCMR1    (*(unsigned char volatile xdata *)0xfec8)
#define PWMA_CCMR2    (*(unsigned char volatile xdata *)0xfec9)
#define PWMA_CCER1    (*(unsigned char volatile xdata *)0xfecc)
#define PWMA_CCR1     (*(unsigned int volatile xdata *)0xfed5)
#define PWMA_CCR2     (*(unsigned int volatile xdata *)0xfed7)
```

```
void main()
{
	P1M0 = 0x00;
	P1M1 = 0x00;
	P3M0 = 0x00;
	P3M1 = 0x00;
	P5M0 = 0x00;
	P5M1 = 0x00;

	P_SW2 = 0x80;

	PWMA_CCER1 = 0x00;		//(CC1 捕获 TI1 上升沿,CC2 捕获 TI1 下降沿)
	PWMA_CCMR1 = 0x01;		//CC1 捕获周期宽度,CC2 捕获高电平宽度
	PWMA_CCMR2 = 0x02;		//CC1 为输入模式,且映射到 TI1FP1 上
	PWMA_CCER1 = 0x11;		//CC2 为输入模式,且映射到 TI1FP2 上
	PWMA_CCER1 |= 0x00;		//使能 CC1/CC2 上的捕获功能
	PWMA_CCER1 |= 0x20;		//设置捕获极性为 CC1 的上升沿
	PWMA_SMCR = 0x54;		//设置捕获极性为 CC2 的下降沿
	PWMA_CR1 = 0x01;		//TS=TI1FP1,SMS=TI1 上升沿复位模式

	PWMA_IER = 0x06;		//使能 CC1/CC2 捕获中断
	EA = 1;

	while (1);
}

void PWMA_ISR() interrupt 26
{
	unsigned int cnt;

	if (PWMA_SR1 & 0x02)
	{
		PWMA_SR1 &= ~0x02;

		cnt = PWMA_CCR1;		//CC1 捕获周期宽度
	}
	if (PWMA_SR1 & 0x04)
	{
		PWMA_SR1 &= ~0x04;

		cnt = PWMA_CCR2;		//CC2 捕获占空比（高电平宽度）
	}
}
```

21.8.11带死区控制的 PWM 互补输出

C 语言代码

```
//测试工作频率为 11.0592MHz

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

typedef struct TIM1_struct
{
```

```
    volatile unsigned char CR1;                 /*!< control register 1 */
    volatile unsigned char CR2;                 /*!< control register 2 */
    volatile unsigned char SMCR;                /*!< Synchro mode control register */
    volatile unsigned char ETR;                 /*!< external trigger register */
    volatile unsigned char IER;                 /*!< interrupt enable register*/
    volatile unsigned char SR1;                 /*!< status register 1 */
    volatile unsigned char SR2;                 /*!< status register 2 */
    volatile unsigned char EGR;                 /*!< event generation register */
    volatile unsigned char CCMR1;               /*!< CC mode register 1 */
    volatile unsigned char CCMR2;               /*!< CC mode register 2 */
    volatile unsigned char CCMR3;               /*!< CC mode register 3 */
    volatile unsigned char CCMR4;               /*!< CC mode register 4 */
    volatile unsigned char CCER1;               /*!< CC enable register 1 */
    volatile unsigned char CCER2;               /*!< CC enable register 2 */
    volatile unsigned char CNTRH;               /*!< counter high */
    volatile unsigned char CNTRL;               /*!< counter low */
    volatile unsigned char PSCRH;               /*!< prescaler high */
    volatile unsigned char PSCRL;               /*!< prescaler low */
    volatile unsigned char ARRH;                /*!< auto-reload register high */
    volatile unsigned char ARRL;                /*!< auto-reload register low */
    volatile unsigned char RCR;                 /*!< Repetition Counter register */
    volatile unsigned char CCR1H;               /*!< capture/compare register 1 high */
    volatile unsigned char CCR1L;               /*!< capture/compare register 1 low */
    volatile unsigned char CCR2H;               /*!< capture/compare register 2 high */
    volatile unsigned char CCR2L;               /*!< capture/compare register 2 low */
    volatile unsigned char CCR3H;               /*!< capture/compare register 3 high */
    volatile unsigned char CCR3L;               /*!< capture/compare register 3 low */
    volatile unsigned char CCR4H;               /*!< capture/compare register 3 high */
    volatile unsigned char CCR4L;               /*!< capture/compare register 3 low */
    volatile unsigned char BKR;                 /*!< Break Register */
    volatile unsigned char DTR;                 /*!< dead-time register */
    volatile unsigned char OISR;                /*!< Output idle register */
}TIM1_TypeDef;

#define     TIM1_BaseAddress    0xFEC0

#define     TIM1                ((TIM1_TypeDef xdata*)TIM1_BaseAddress)
#define     PWMA_ENO            (*(unsigned char volatile xdata *)0xFEB1)
#define     PWMA_PS             (*(unsigned char volatile xdata *)0xFEB2)

sfr         P0M0        =       0x94;
sfr         P0M1        =       0x93;
sfr         P1M0        =       0x92;
sfr         P1M1        =       0x91;
sfr         P3M0        =       0xb2;
sfr         P3M1        =       0xb1;
sfr         P_SW2       =       0xba;

sbit        P03         =       P0^3;

void main(void)
{
    P_SW2 = 0x80;

    P0M1 = 0x00;
    P0M0 = 0xFF;
    P1M1 = 0x00;
    P1M0 = 0xFF;
```

```
    PWMA_ENO = 0xFF;                        //IO 输出 PWM
    PWMA_PS = 0x00;                         //00:PWM at P1

/*********************************************************
PWMx_duty = [CCRx/(ARR + 1)]*100
*********************************************************/
    TIM1-> PSCRH = 0x00;                    //预分频寄存器
    TIM1-> PSCRL = 0x00;
    TIM1-> DTR = 0x00;                      //死区时间配置

    TIM1-> CCMR1 = 0x68;                    //通道模式配置
    TIM1-> CCMR2 = 0x68;
    TIM1-> CCMR3 = 0x68;
    TIM1-> CCMR4 = 0x68;

    TIM1-> ARRH = 0x08;                     //自动重装载寄存器，计数器 overflow 点
    TIM1-> ARRL = 0x00;

    TIM1-> CCR1H = 0x04;                    //计数器比较值
    TIM1-> CCR1L = 0x00;
    TIM1-> CCR2H = 0x02;
    TIM1-> CCR2L = 0x00;
    TIM1-> CCR3H = 0x01;
    TIM1-> CCR3L = 0x00;
    TIM1-> CCR4H = 0x01;
    TIM1-> CCR4L = 0x00;

    TIM1-> CCER1 = 0x55;                    //配置通道输出使能和极性
    TIM1-> CCER2 = 0x55;                    //配置通道输出使能和极性

    TIM1-> BKR = 0x80;                      //主输出使能 相当于总开关
    TIM1-> IER = 0x02;                      //使能中断
    TIM1-> CR1 = 0x01;                      //使能计数器

    EA = 1;
    while (1);
}

void PWMA_ISR() interrupt 26
{
    if(TIM1->SR1 & 0X02)
    {
        P03 = ~P03;
        TIM1->SR1 &=~0X02;
    }
}
```

21.8.12PWM 端口做外部中断（下降沿中断或者上升沿中断）

C 语言代码

```
//测试工作频率为 11.0592MHz

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

#define    PWMA_CR1        (*(unsigned char volatile xdata *)0xfec0)
#define    PWMA_IER        (*(unsigned char volatile xdata *)0xfec4)
```

```
#define     PWMA_SR1        (*(unsigned char volatile xdata *)0xfec5)
#define     PWMA_CCMR1      (*(unsigned char volatile xdata *)0xfec8)
#define     PWMA_CCER1      (*(unsigned char volatile xdata *)0xfecc)

sfr         P0M0        =   0x94;
sfr         P0M1        =   0x93;
sfr         P1M0        =   0x92;
sfr         P1M1        =   0x91;
sfr         P3M0        =   0xb2;
sfr         P3M1        =   0xb1;

sfr         P_SW2       =   0xba;

sbit        P37         =   P3^7;

void main(void)
{
    P_SW2 = 0x80;

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

    P_SW2 = 0x80;
                                        //(捕获 PWM1P 上升沿/下降沿)
    PWMA_CCER1 = 0x00;
    PWMA_CCMR1 = 0x01;                  //CC1 为输入模式,且映射到 TI1FP1 上
    PWMA_CCER1 = 0x01;                  //使能 CC1 上的捕获功能
    PWMA_CCER1 |= 0x00;                 //设置捕获极性为 CC1 的上升沿
//  PWMA_CCER1 |= 0x02;                 //设置捕获极性为 CC1 的下降沿
    PWMA_CR1 = 0x01;
    PWMA_IER = 0x02;
    EA = 1;

    while (1);
}

void PWMA_ISR() interrupt 26
{
    if(PWMA_SR1 & 0X02)
    {
        P37 = ~P37;
        PWMA_SR1 &=~0X02;
    }
}
```

21.8.13 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;
```

```
#define    PWMA_CCER1    (*(unsigned char volatile xdata *)0xfecc)
#define    PWMA_CCMR1    (*(unsigned char volatile xdata *)0xfec8)
#define    PWMA_ENO      (*(unsigned char volatile xdata *)0xfeb1)
#define    PWMA_BKR      (*(unsigned char volatile xdata *)0xfedd)
#define    PWMA_CCR1     (*(unsigned int volatile xdata *)0xfed5)
#define    PWMA_ARR      (*(unsigned int volatile xdata *)0xfed2)
#define    PWMA_CR1      (*(unsigned char volatile xdata *)0xfec0)

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;
    PWMA_CCER1 = 0x00;          //写 CCMRx 前必须先清零 CCERx 关闭通道
    PWMA_CCMR1 = 0x60;          //设置 CC1 为 PWMA 输出模式
    PWMA_CCER1 = 0x01;          //使能 CC1 通道
    PWMA_CCR1 = 100;            //设置占空比时间
    PWMA_ARR = 500;             //设置周期时间
    PWMA_ENO = 0x01;            //使能 PWM1P 端口输出
    PWMA_BKR = 0x80;            //使能主输出
    PWMA_CR1 = 0x01;            //开始计时

    while (1);
}
```

21.8.14使用 PWM 的 CEN 启动 PWMA 定时器，实时触发 ADC

C 语言代码

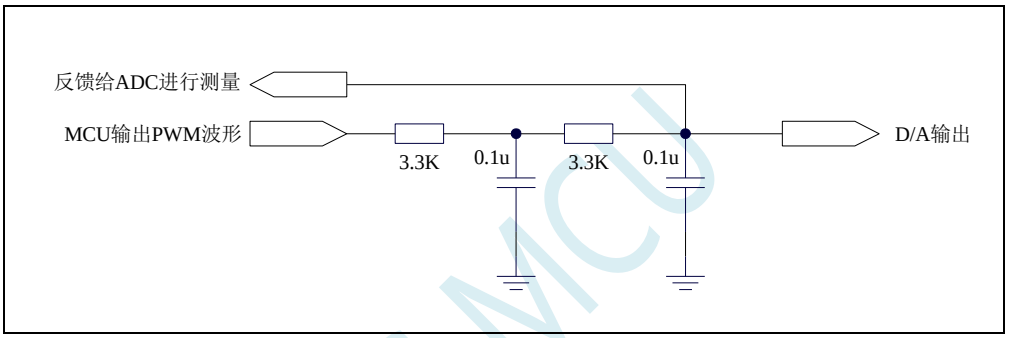
```
//测试工作频率为 11.0592MHz
```

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

#define    PWMA_CR1      (*(unsigned char volatile xdata *)0xfec0)
#define    PWMA_CR2      (*(unsigned char volatile xdata *)0xfec1)
#define    PWMA_IER      (*(unsigned char volatile xdata *)0xfec4)
#define    PWMA_SR1      (*(unsigned char volatile xdata *)0xfec5)
#define    PWMA_CCMR1    (*(unsigned char volatile xdata *)0xfec8)
#define    PWMA_CCER1    (*(unsigned char volatile xdata *)0xfecc)
#define    PWMA_ARR      (*(unsigned int volatile xdata *)0xfed2)

sfr        P0M0      =   0x94;
sfr        P0M1      =   0x93;
sfr        P1M0      =   0x92;
sfr        P1M1      =   0x91;
sfr        P3M0      =   0xb2;
sfr        P3M1      =   0xb1;

sfr        P_SW2     =   0xba;

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;

void delay()
{
    int i;
    for (i=0; i<100; i++);
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x01;
    P3M0 = 0x00;
    P3M1 = 0x00;

    P_SW2 |= 0x80;

    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;

    PWMA_CR2 = 0x10;                            //CEN 信号为 TRGO,可用于触发 ADC
    PWMA_ARR = 5000;
    PWMA_IER = 0x01;
    PWMA_CR1 = 0x01;                            //设置 CEN 启动 PWMA 定时器，实时触发 ADC
    EA = 1;

    while (1);
}

void ADC_ISR() interrupt 5
```

```
{
    ADC_CONTR &= ~ADC_FLAG;
}

void PWMA_ISR() interrupt 26
{
    if(PWMA_SR1 & 0x01)
    {
        PWMA_SR1 &=~0x01;
    }
}
```

21.8.15 利用 PWM 实现 16 位 DAC 的参考线路图

STC12H 系列单片机的高级 PWM 定时器可输出 16 位的 PWM 波形，再经过两级低通滤波即可产生 16 位的 DAC 信号，通过调节 PWM 波形的高电平占空比即可实现 DAC 信号的改变。应用线路图如下图所示，输出的 DAC 信号可输入到 MCU 的 ADC 进行反馈测量。



21.8.16 利用 PWM 实现互补 SPWM

高级 PWM 定时器 PWM1P/PWM1N，PWM2P/PWM2N，PWM3P/PWM3N，PWM4P/PWM4N 每个通道都可独立实现 PWM 输出，或者两两互补对称输出。演示使用 PWM1P，PWM1N 产生互补的 SPWM。主时钟选择 24MHZ，PWM 时钟选择 1T，PWM 周期 2400，死区 12 个时钟(0.5us)，正弦波表用 200 点，输出正弦波频率 = 24000000 / 2400 / 200 = 50 HZ。

本程序仅仅是一个 SPWM 的演示程序，用户可以通过上面的计算方法修改 PWM 周期和正弦波的点数和幅度。本程序输出频率固定，如果需要变频，请用户自己设计变频方案。

C 语言代码

```
//测试工作频率为24MHz

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

#define    MAIN_Fosc        24000000L              //定义主时钟

typedef    unsigned char    u8;
typedef    unsigned int     u16;
typedef    unsigned long    u32;

sfr        TH2          =   0xD6;
sfr        TL2          =   0xD7;
sfr        IE2          =   0xAF;
sfr        INT_CLKO     =   0x8F;
```

```
sfr     AUXR    =   0x8E;
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;

/****************************** 用户定义宏 ***********************************/

#define     PWMA_ENO        (*(unsigned char  volatile xdata *)  0xFEB1)
#define     PWMA_PS         (*(unsigned char  volatile xdata *)  0xFEB2)
#define     PWMB_ENO        (*(unsigned char  volatile xdata *)  0xFEB5)
#define     PWMB_PS         (*(unsigned char  volatile xdata *)  0xFEB6)

#define     PWMA_CR1        (*(unsigned char  volatile xdata *)  0xFEC0)
#define     PWMA_CR2        (*(unsigned char  volatile xdata *)  0xFEC1)
#define     PWMA_SMCR       (*(unsigned char  volatile xdata *)  0xFEC2)
#define     PWMA_ETR        (*(unsigned char  volatile xdata *)  0xFEC3)
#define     PWMA_IER        (*(unsigned char  volatile xdata *)  0xFEC4)
#define     PWMA_SR1        (*(unsigned char  volatile xdata *)  0xFEC5)
#define     PWMA_SR2        (*(unsigned char  volatile xdata *)  0xFEC6)
#define     PWMA_EGR        (*(unsigned char  volatile xdata *)  0xFEC7)
#define     PWMA_CCMR1      (*(unsigned char  volatile xdata *)  0xFEC8)
#define     PWMA_CCMR2      (*(unsigned char  volatile xdata *)  0xFEC9)
#define     PWMA_CCMR3      (*(unsigned char  volatile xdata *)  0xFECA)
#define     PWMA_CCMR4      (*(unsigned char  volatile xdata *)  0xFECB)
#define     PWMA_CCER1      (*(unsigned char  volatile xdata *)  0xFECC)
#define     PWMA_CCER2      (*(unsigned char  volatile xdata *)  0xFECD)
#define     PWMA_CNTRH      (*(unsigned char  volatile xdata *)  0xFECE)
#define     PWMA_CNTRL      (*(unsigned char  volatile xdata *)  0xFECF)
#define     PWMA_PSCRH      (*(unsigned char  volatile xdata *)  0xFED0)
#define     PWMA_PSCRL      (*(unsigned char  volatile xdata *)  0xFED1)
#define     PWMA_ARRH       (*(unsigned char  volatile xdata *)  0xFED2)
#define     PWMA_ARRL       (*(unsigned char  volatile xdata *)  0xFED3)
#define     PWMA_RCR        (*(unsigned char  volatile xdata *)  0xFED4)
#define     PWMA_CCR1H      (*(unsigned char  volatile xdata *)  0xFED5)
#define     PWMA_CCR1L      (*(unsigned char  volatile xdata *)  0xFED6)
#define     PWMA_CCR2H      (*(unsigned char  volatile xdata *)  0xFED7)
#define     PWMA_CCR2L      (*(unsigned char  volatile xdata *)  0xFED8)
#define     PWMA_CCR3H      (*(unsigned char  volatile xdata *)  0xFED9)
#define     PWMA_CCR3L      (*(unsigned char  volatile xdata *)  0xFEDA)
```

```
#define    PWMA_CCR4H    (*(unsigned char  volatile xdata *)  0xFEDB)
#define    PWMA_CCR4L    (*(unsigned char  volatile xdata *)  0xFEDC)
#define    PWMA_BKR      (*(unsigned char  volatile xdata *)  0xFEDD)
#define    PWMA_DTR      (*(unsigned char  volatile xdata *)  0xFEDE)
#define    PWMA_OISR     (*(unsigned char  volatile xdata *)  0xFEDF)

/*****************************************************************************/

#define    PWMA_1        0x00                    //P:P1.0   N:P1.1
#define    PWMA_2        0x01                    //P:P2.0   N:P2.1
#define    PWMA_3        0x02                    //P:P6.0   N:P6.1

#define    PWMB_1        0x00                    //P:P1.2/P5.4   N:P1.3
#define    PWMB_2        0x04                    //P:P2.2   N:P2.3
#define    PWMB_3        0x08                    //P:P6.2   N:P6.3

#define    PWM3_1        0x00                    //P:P1.4   N:P1.5
#define    PWM3_2        0x10                    //P:P2.4   N:P2.5
#define    PWM3_3        0x20                    //P:P6.4   N:P6.5

#define    PWM4_1        0x00                    //P:P1.6   N:P1.7
#define    PWM4_2        0x40                    //P:P2.6   N:P2.7
#define    PWM4_3        0x80                    //P:P6.6   N:P6.7
#define    PWM4_4        0xC0                    //P:P3.4   N:P3.3

#define    ENO1P         0x01
#define    ENO1N         0x02
#define    ENO2P         0x04
#define    ENO2N         0x08
#define    ENO3P         0x10
#define    ENO3N         0x20
#define    ENO4P         0x40
#define    ENO4N         0x80

/*************  本地变量声明  **************/

unsigned int code T_SinTable[]=
{
    1220, 1256, 1292, 1328, 1364, 1400, 1435, 1471,
    1506, 1541, 1575, 1610, 1643, 1677, 1710, 1742,
    1774, 1805, 1836, 1866, 1896, 1925, 1953, 1981,
    2007, 2033, 2058, 2083, 2106, 2129, 2150, 2171,
    2191, 2210, 2228, 2245, 2261, 2275, 2289, 2302,
    2314, 2324, 2334, 2342, 2350, 2356, 2361, 2365,
    2368, 2369, 2370, 2369, 2368, 2365, 2361, 2356,
    2350, 2342, 2334, 2324, 2314, 2302, 2289, 2275,
    2261, 2245, 2228, 2210, 2191, 2171, 2150, 2129,
    2106, 2083, 2058, 2033, 2007, 1981, 1953, 1925,
    1896, 1866, 1836, 1805, 1774, 1742, 1710, 1677,
    1643, 1610, 1575, 1541, 1506, 1471, 1435, 1400,
    1364, 1328, 1292, 1256, 1220, 1184, 1148, 1112,
    1076, 1040, 1005,  969,  934,  899,  865,  830,
     797,  763,  730,  698,  666,  635,  604,  574,
     544,  515,  487,  459,  433,  407,  382,  357,
     334,  311,  290,  269,  249,  230,  212,  195,
     179,  165,  151,  138,  126,  116,  106,   98,
      90,   84,   79,   75,   72,   71,   70,   71,
      72,   75,   79,   84,   90,   98,  106,  116,
     126,  138,  151,  165,  179,  195,  212,  230,
```

```
        249,   269,   290,   311,   334,   357,   382,   407,
        433,   459,   487,   515,   544,   574,   604,   635,
        666,   698,   730,   763,   797,   830,   865,   899,
        934,   969, 1005, 1040, 1076, 1112, 1148, 1184,
};

u16 PWMA_Duty;
u8 PWM_Index;                                   //SPWM 查表索引

/******************** 主函数 *************************/
void main(void)
{
    P0M1 = 0;   P0M0 = 0;   //设置为准双向口
    P1M1 = 0;   P1M0 = 0;   //设置为准双向口
    P2M1 = 0;   P2M0 = 0;   //设置为准双向口
    P3M1 = 0;   P3M0 = 0;   //设置为准双向口
    P4M1 = 0;   P4M0 = 0;   //设置为准双向口
    P5M1 = 0;   P5M0 = 0;   //设置为准双向口
    P6M1 = 0;   P6M0 = 0;   //设置为准双向口
    P7M1 = 0;   P7M0 = 0;   //设置为准双向口

    PWMA_Duty = 1220;

    P_SW2 |= 0x80;

    PWMA_CCER1 = 0x00;                          //写 CCMRx 前必须先清零 CCxE 关闭通道
    PWMA_CCER2 = 0x00;
    PWMA_CCMR1 = 0x60;                          //通道模式配置
//  PWMA_CCMR2 = 0x60;
//  PWMA_CCMR3 = 0x60;
//  PWMA_CCMR4 = 0x60;
    PWMA_CCER1 = 0x05;                          //配置通道输出使能和极性
//  PWMA_CCER2 = 0x55;

    PWMA_ARRH = 0x09;                           //设置周期时间
    PWMA_ARRL = 0x60;

    PWMA_CCR1H = (u8)(PWMA_Duty >> 8);          //设置占空比时间
    PWMA_CCR1L = (u8)(PWMA_Duty);

    PWMA_DTR = 0x0C;                            //设置死区时间

    PWMA_ENO = 0x00;
    PWMA_ENO |= ENO1P;                          //使能输出
    PWMA_ENO |= ENO1N;                          //使能输出
//  PWMA_ENO |= ENO2P;                          //使能输出
//  PWMA_ENO |= ENO2N;                          //使能输出
//  PWMA_ENO |= ENO3P;                          //使能输出
//  PWMA_ENO |= ENO3N;                          //使能输出
//  PWMA_ENO |= ENO4P;                          //使能输出
//  PWMA_ENO |= ENO4N;                          //使能输出

    PWMA_PS = 0x00;                             //高级 PWM 通道输出脚选择位
    PWMA_PS |= PWMA_3;                          //选择 PWMA_3 通道
//  PWMA_PS |= PWMB_3;                          //选择 PWMB_3 通道
//  PWMA_PS |= PWM3_3;                          //选择 PWM3_3 通道
//  PWMA_PS |= PWM4_3;                          //选择 PWM4_3 通道

    PWMA_BKR = 0x80;                            //使能主输出
```

```
    PWMA_IER = 0x01;                        //使能中断
    PWMA_CR1 |= 0x01;                       //开始计时

    P_SW2 &= 0x7f;

    EA = 1;                                 //打开总中断

    while (1)
    {
    }
}

/******************** 中断函数 ************************/
void PWMA_ISR() interrupt 26
{
    P_SW2 |= 0x80;
    if (PWMA_SR1 & 0x01)
    {
        PWMA_SR1 &=~0x01;
        PWMA_Duty = T_SinTable[PWM_Index];
        if (++PWM_Index >= 200)
            PWM_Index = 0;

        PWMA_CCR1H = (u8)(PWMA_Duty >> 8);      //设置占空比时间
        PWMA_CCR1L = (u8)(PWMA_Duty);
    }
    PWMA_SR1 = 0;
    P_SW2 &= 0x7f;
}
```

STC MCU

22 增强型双数据指针

STC12H 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

22.1 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

22.1.1 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

22.1.2 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

22.1.3 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
```

22.1.4 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）
```

STC MCU

22.2 范例程序

22.2.1 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

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         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,#10111000B      ;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,
```

```
                                        ;完成后 DPTR1 自动减 1 并将 DPTR0 设置为 DPTR
        DJNZ    R7,COPY_NEXT            ;

        SJMP    $

        END
```

22.2.2 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

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

        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
```

STC MCU

23 MDU16 硬件 16 位乘除法器

STC12H 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。

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	
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	0000,0000

23.1.1 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]							

23.1.2 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}

23.1.3 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

23.1.4 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。

STC MCU

23.2 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 *)0xfcf0)
#define    MD3U16        (*(unsigned int volatile xdata *)0xfcf0)
#define    MD1U16        (*(unsigned int volatile xdata *)0xfcf2)
#define    MD5U16        (*(unsigned int volatile xdata *)0xfcf4)

#define    MD3           (*(unsigned char volatile xdata *)0xfcf0)
#define    MD2           (*(unsigned char volatile xdata *)0xfcf1)
#define    MD1           (*(unsigned char volatile xdata *)0xfcf2)
#define    MD0           (*(unsigned char volatile xdata *)0xfcf3)
#define    MD5           (*(unsigned char volatile xdata *)0xfcf4)
#define    MD4           (*(unsigned char volatile xdata *)0xfcf5)
#define    ARCON         (*(unsigned char volatile xdata *)0xfcf6)
#define    OPCON         (*(unsigned char volatile xdata *)0xfcf7)

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 = data2;                         //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;                              //dat1 用户给定
ARCON = (2 << 5) + num;                     //32 位左移模式
//ARCON = (1 << 5) + num;                   //32 位右移模式
OPCON = 1;                                  //启动计算
while((OPCON & 1) != 0);                    //等待计算完成
res = MD3U32;                               //32 位结果
```

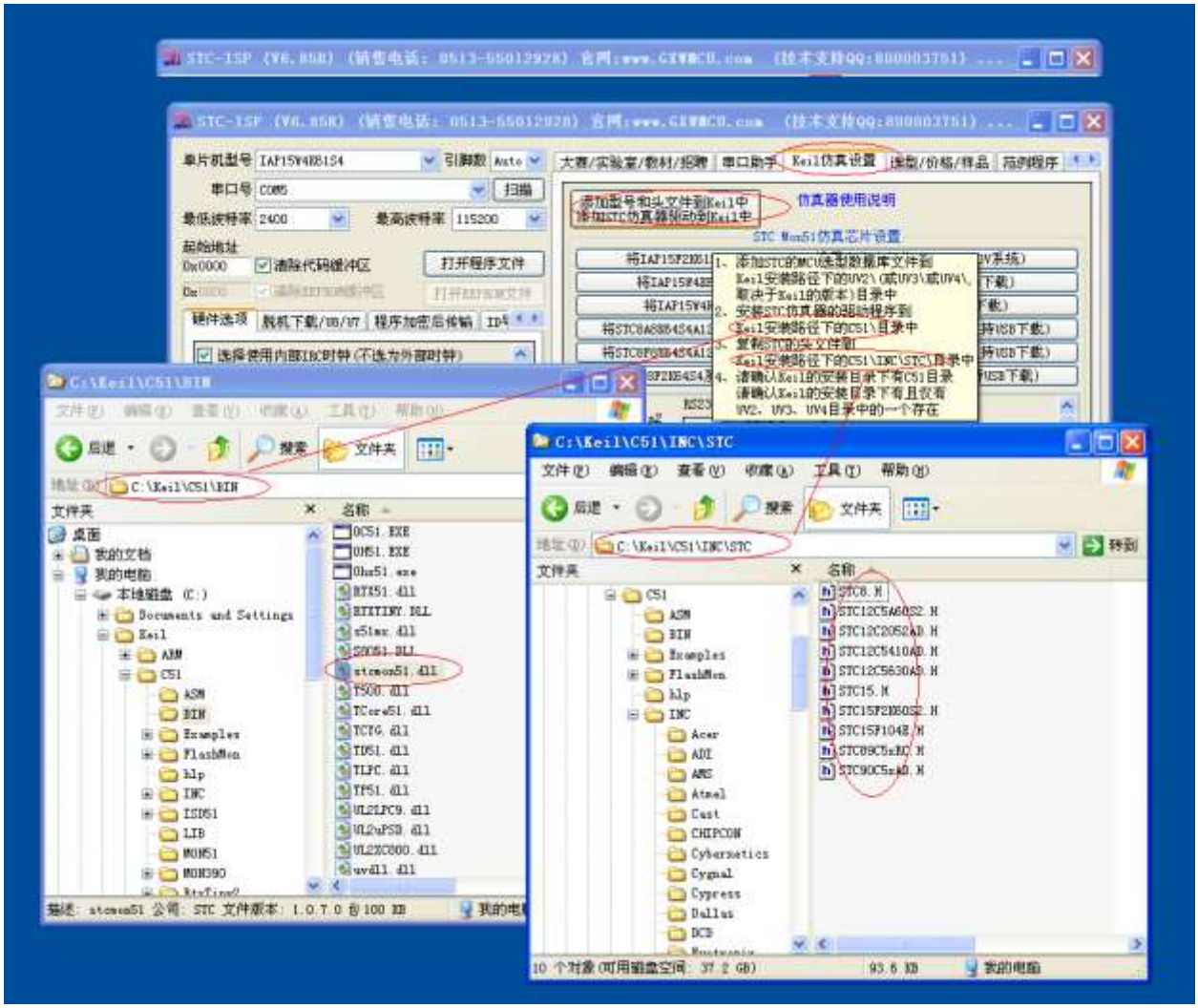
STC MCU

附录A　编译器（汇编器）/仿真器使用指南

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 <STC12H.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。

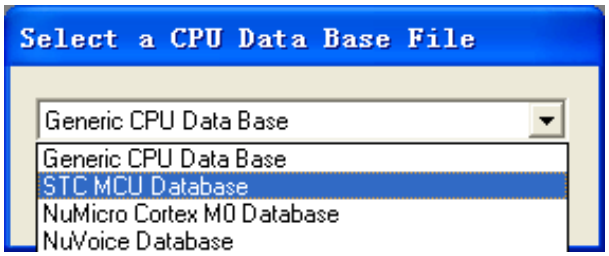
1、安装 Keil 版本的仿真驱动



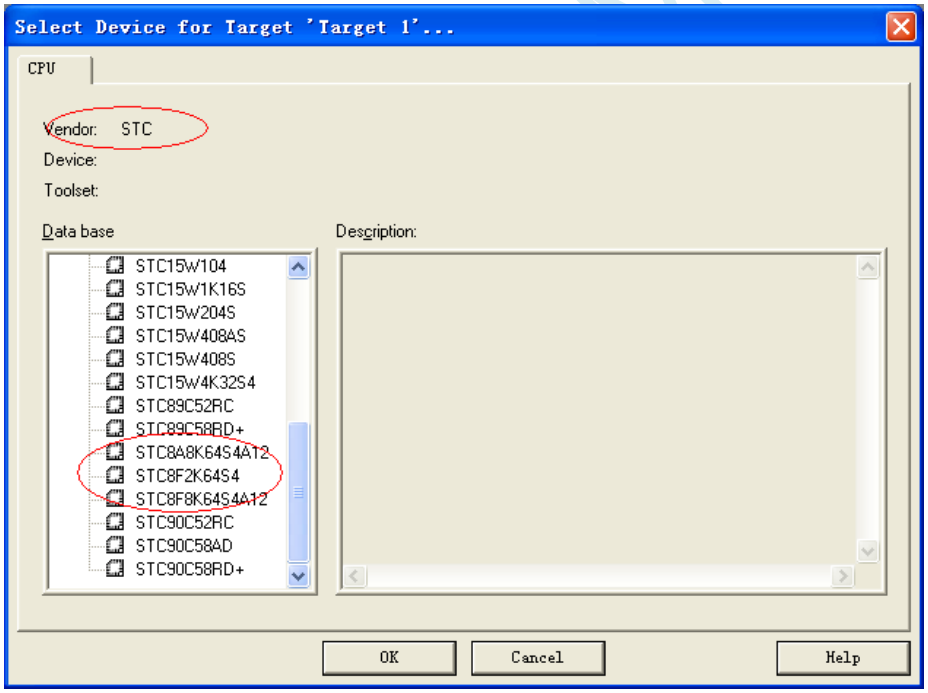
如上图，首先选择“Keil 仿真设置”页面，点击“添加 MCU 型号到 Keil 中”，在出现的如下的目录选择窗口中，定位到 Keil 的安装目录(一般可能为“C:\Keil\”)，“确定”后出现下图中右边所示的提示信息，表示安装成功。添加头文件的同时也会安装 STC 的 Monitor51 仿真驱动 STCMON51.DLL，驱动与头文件的的安装目录如上图所示。

2、在 Keil 中创建项目

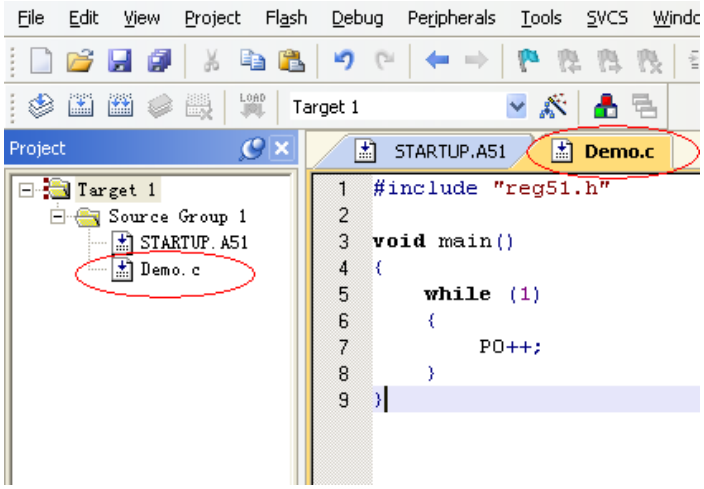
若第一步的驱动安装成功，则在 Keil 中新建项目时选择芯片型号时，便会有“STC MCU Database”的选择项，如下图



然后从列表中选择响应的 MCU 型号，我们在此选择“STC8A8K64S4”的型号，点击“确定”完成选择



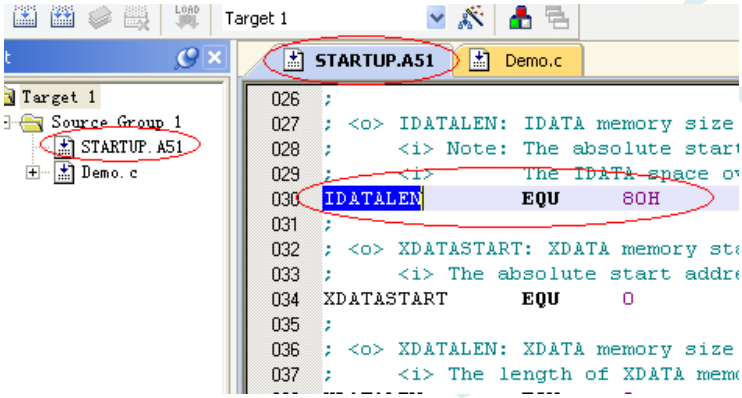
添加源代码文件到项目中，如下图：



保存项目，若编译无误，则可以进行下面的项目设置了

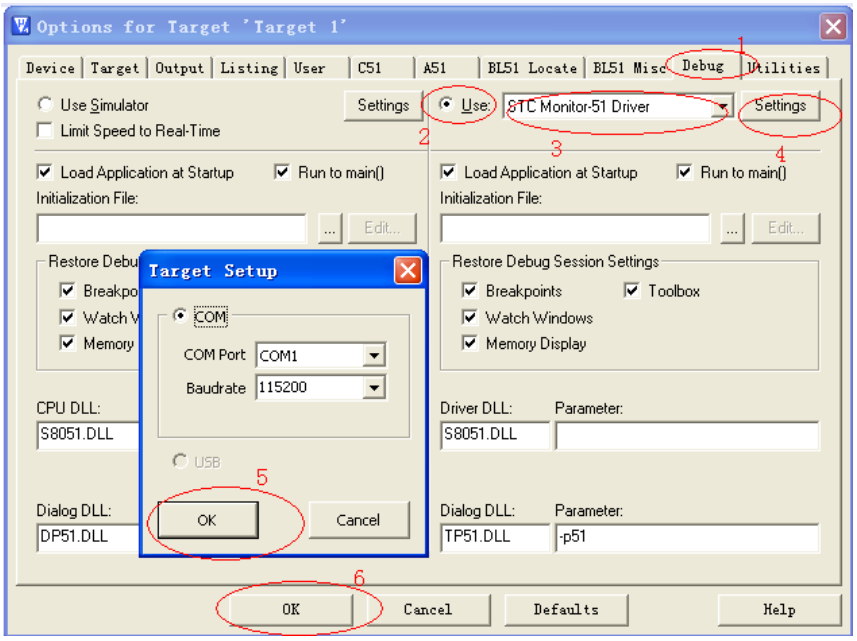
附加说明一点：

当创建的是 C 语言项目，且有将启动文件“STARTUP.A51”添加到项目中时，里面有一个命名为“IDATALEN”的宏定义，它是用来定义 IDATA 大小的一个宏，默认值是 128，即十六进制的 80H，同时它也是启动文件中需要初始化为 0 的 IDATA 的大小。所以当 IDATA 定义为 80H，那么 STARTUP.A51 里面的代码则会将 IDATA 的 00-7F 的 RAM 初始化为 0；同样若将 IDATA 定义为 0FFH，则会将 IDATA 的 00-FF 的 RAM 初始化为 0。



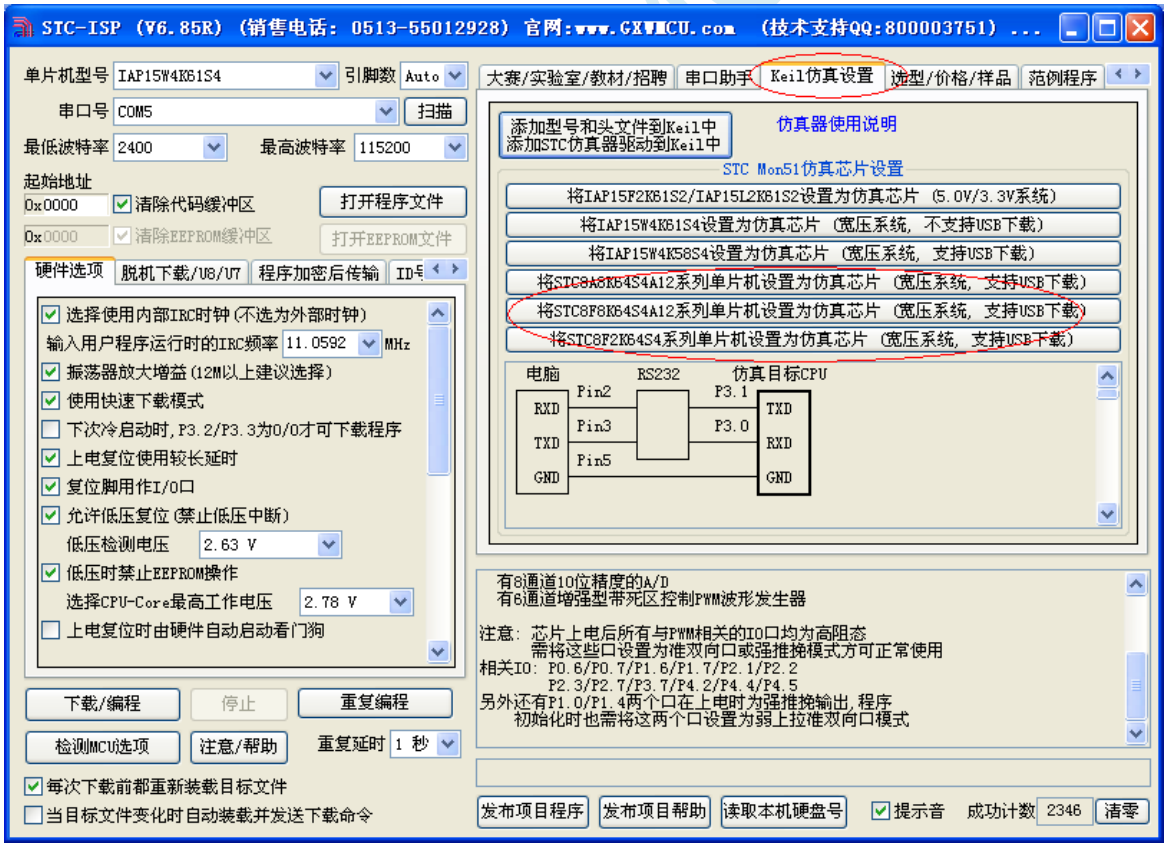
虽然 STC8 系列的单片机的 IDATA 大小为 256 字节（00-7F 的 DATA 和 80H-FFH 的 IDATA），但由于在 RAM 的最后 17 个字节有写入 ID 号以及相关的测试参数，若用户在程序中需要使用这一部分数据，则一定不要将 IDATALEN 定义为 256。

3、项目设置，选择 STC 仿真驱动



如上图，首先进入到项目的设置页面，选择“Debug”设置页，第 2 步选择右侧的硬件仿真“Use …”，第 3 步，在仿真驱动下拉列表中选择“STC Monitor-51 Driver”项，然后点击“Settings”按钮，进入下面的设置画面，对串口的端口号和波特率进行设置，波特率一般选择 115200。到此设置便完成了。

4、创建仿真芯片



准备一颗 STC8A 系列或者 STC8F 系列的芯片，并通过下载板连接到电脑的串口，然后如上图，选择正确的芯片型号，然后进入到“Keil 仿真设置”页面，点击相应型号的按钮，当程序下载完成后仿真器便制作完成了。

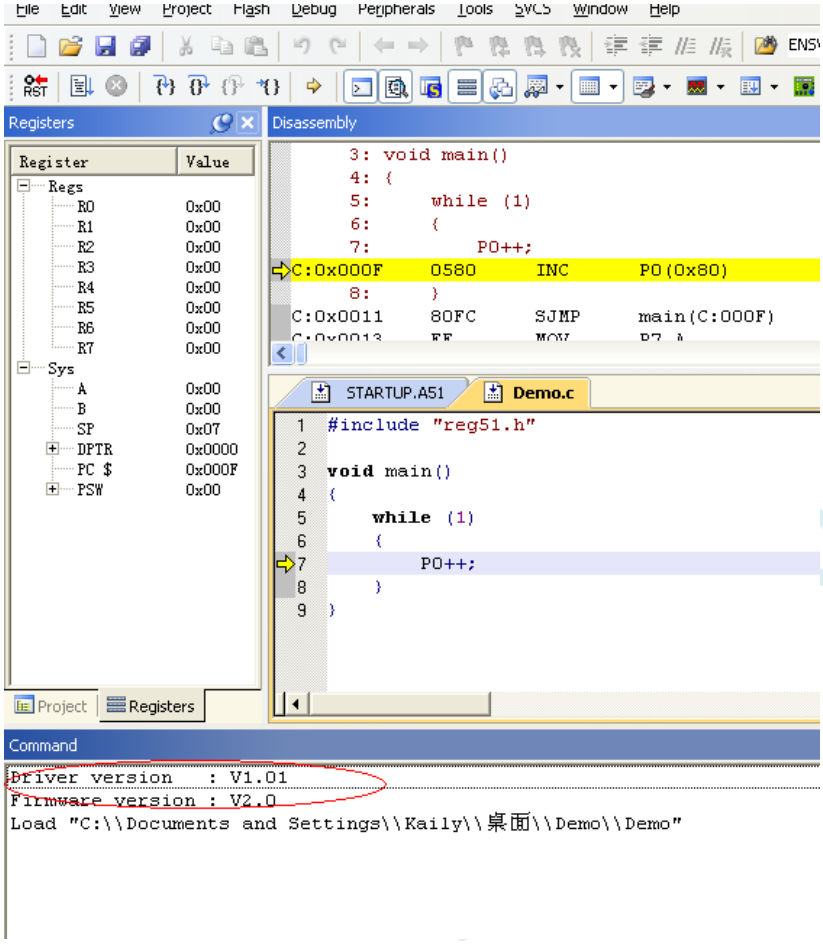
5、开始仿真

将制作完成的仿真芯片通过串口与电脑相连接。

将前面我们所创建的项目编译至没有错误后，按“Ctrl+F5”开始调试。

若硬件连接无误的话，将会进入到类似于下面的调试界面，并在命令输出窗口显示当前的仿真驱动版本号和当前仿真监控代码固件的版本号

断点设置的个数目前最大允许 20 个（理论上可设置任意个，但是断点设置得过多会影响调试的速度）。



仿真注意事项：

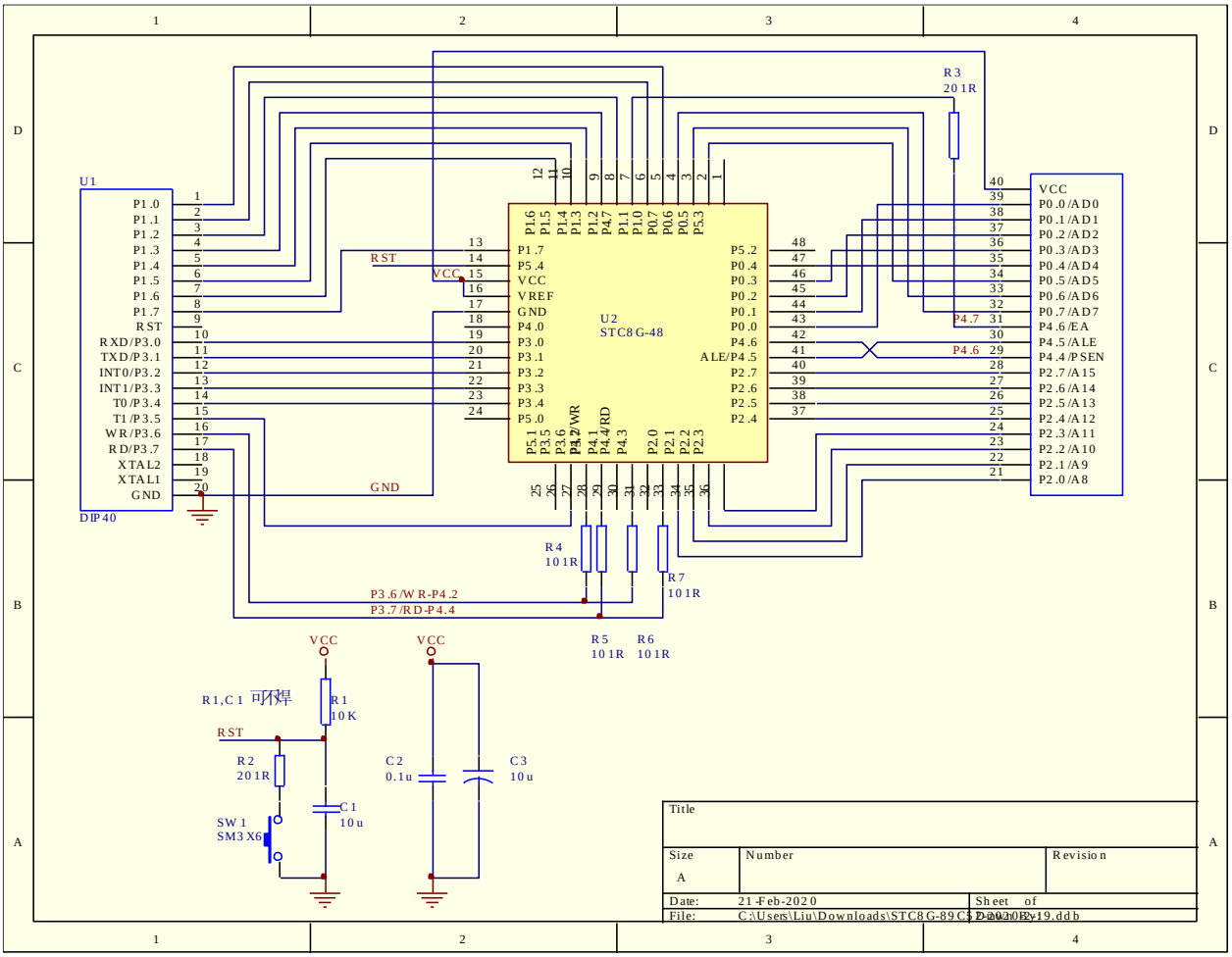
1、 仿真监控程序占用 P3.0/P3.1 两个端口，但不占用串口 1，用户可以将串口 1 切换到 P3.6/P3.7 或者 P1.6/P1.7 再使用

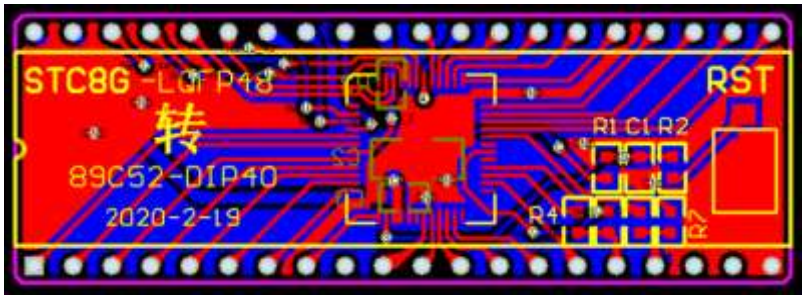
2、 仿真监控程序占用内部扩展 RAM(XDATA)的最后 768 字节，用户不可对这个区域的 XDATA 进行写操作

附录B　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。

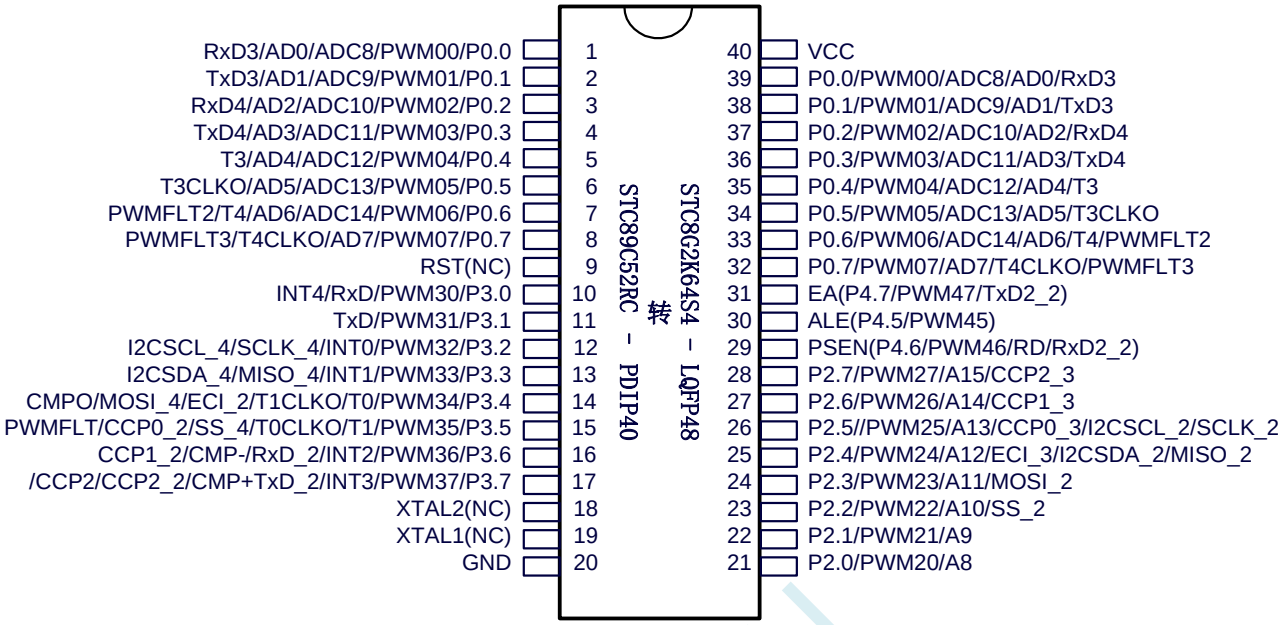
下图是转换板的原理图和 PCB 板图





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



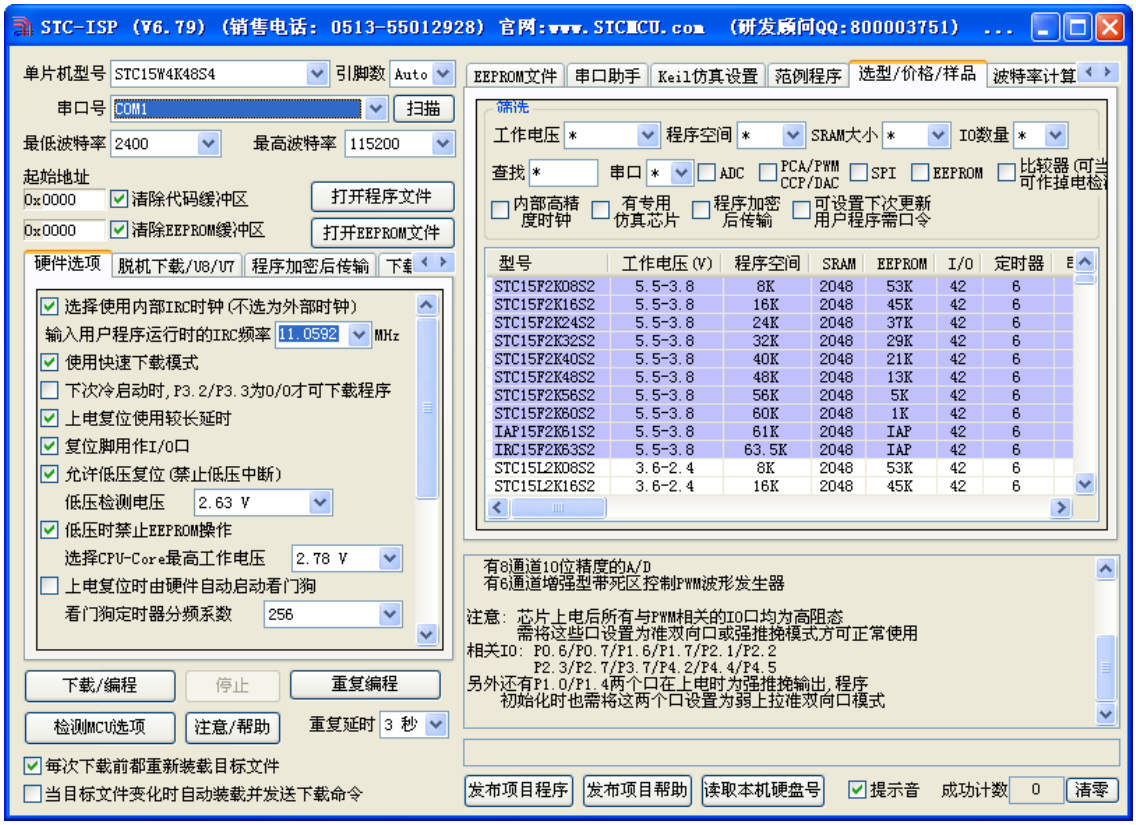
注意：

- ✓ 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，3.6 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C　STC-USB 驱动程序安装说明

Windows XP 安装方法

打开 V6.79 版（或者更新的版本）的 STC-ISP 下载软件，下载软件会自动将驱动文件复制到相关的系统目录

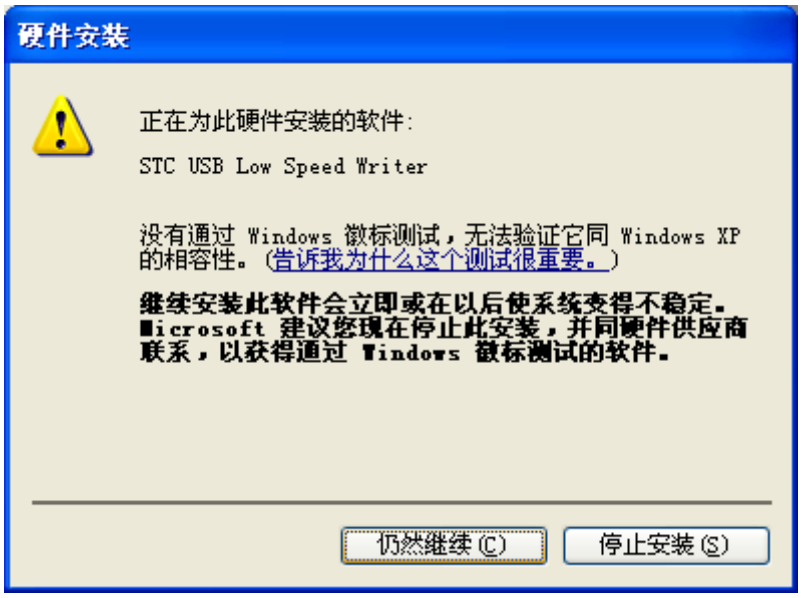


插入 USB 设备，系统找到设备后自动弹出如下对话框，选择其中的“否，暂时不”项

STC MCU

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

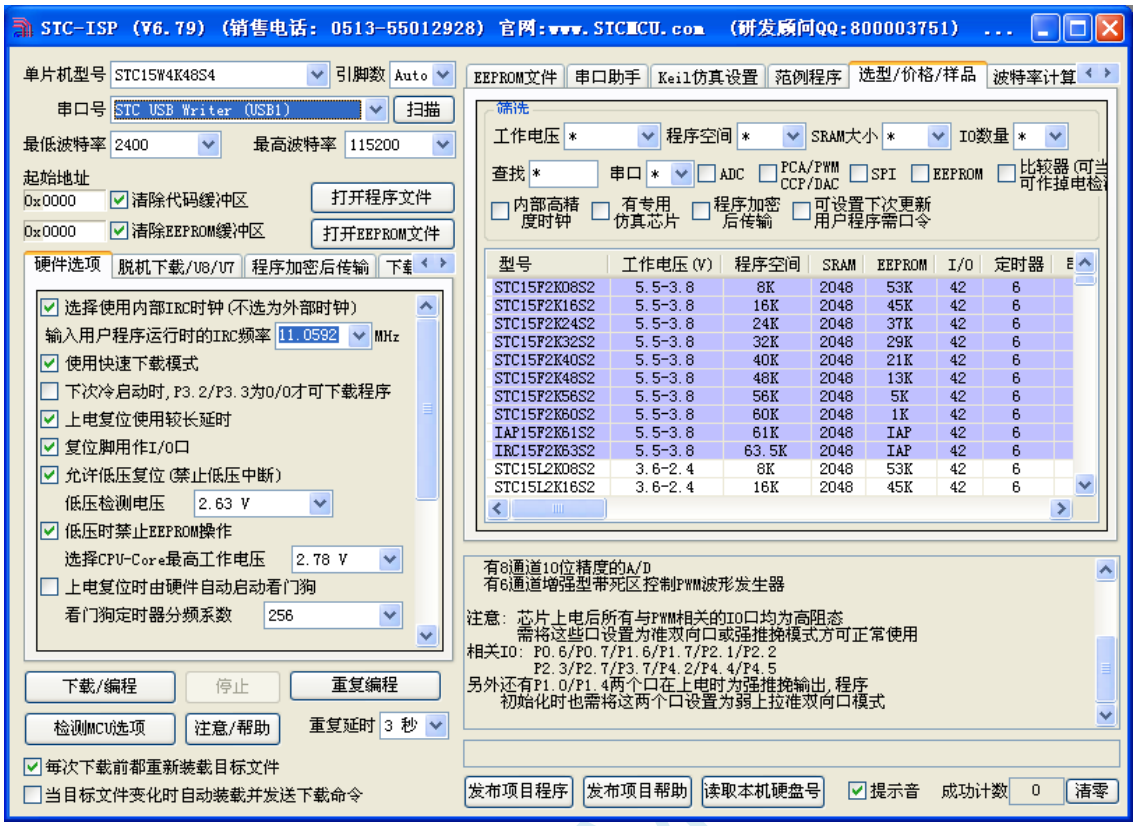


STC MCU

接下系统会自动安装驱动，如下图

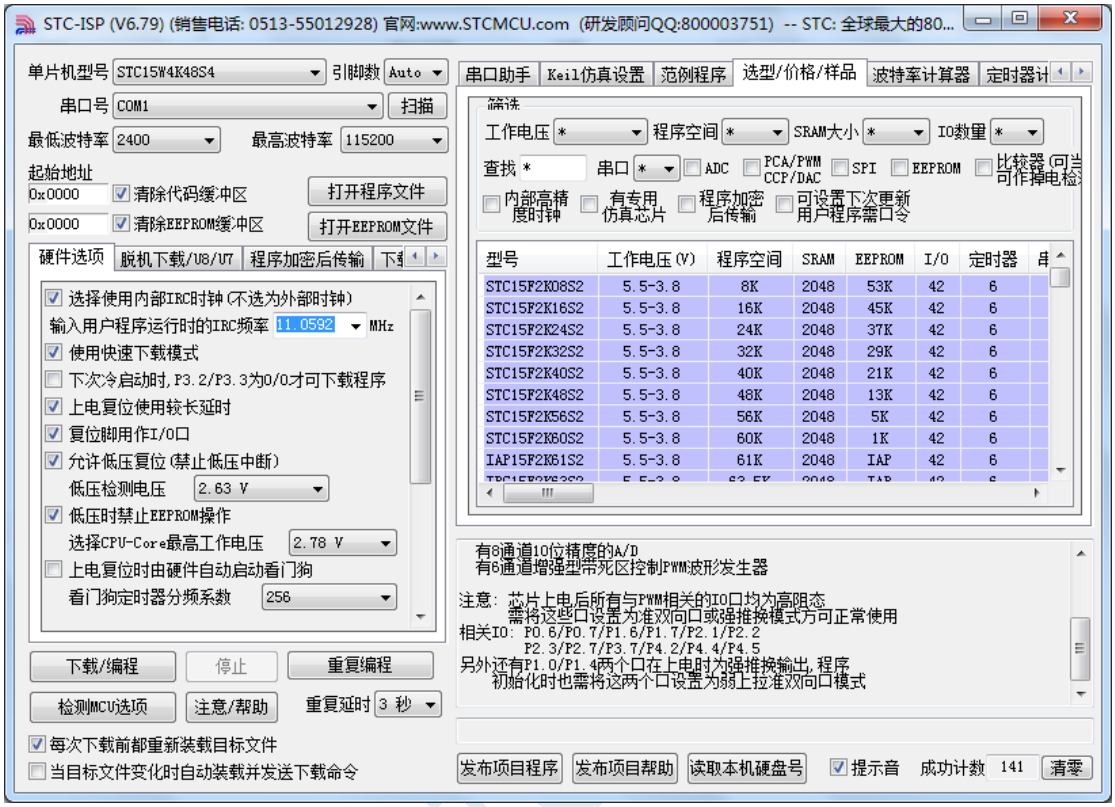
出现下面的的对话框表示驱动安装完成

此时，之前打开的 STC-ISP 下载软件中的串口号列表会自动选择所插入的 USB 设备，并显示设备名称为“STC USB Writer (USB1)” ，如下图：



Windows 7（32 位）安装方法

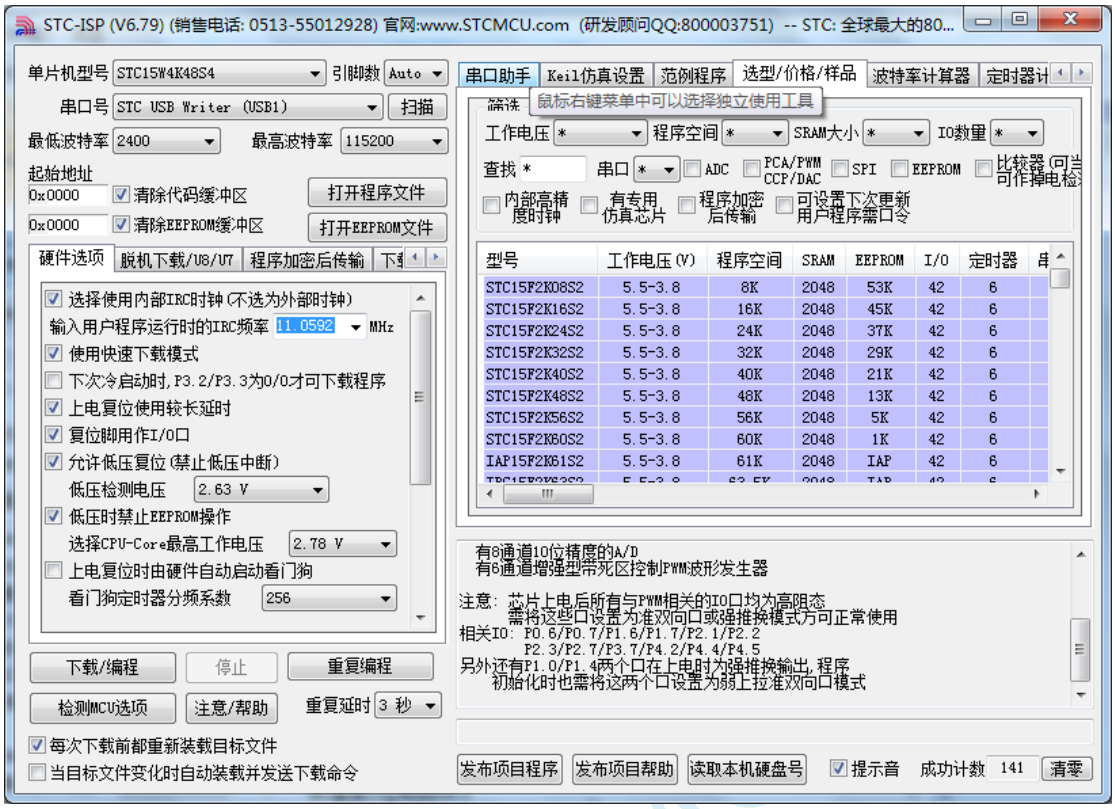
打开 V6.79 版（或者更新的版本）的 STC-ISP 下载软件，下载软件会自动将驱动文件复制到相关的系统目录



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。

STC MCU

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：

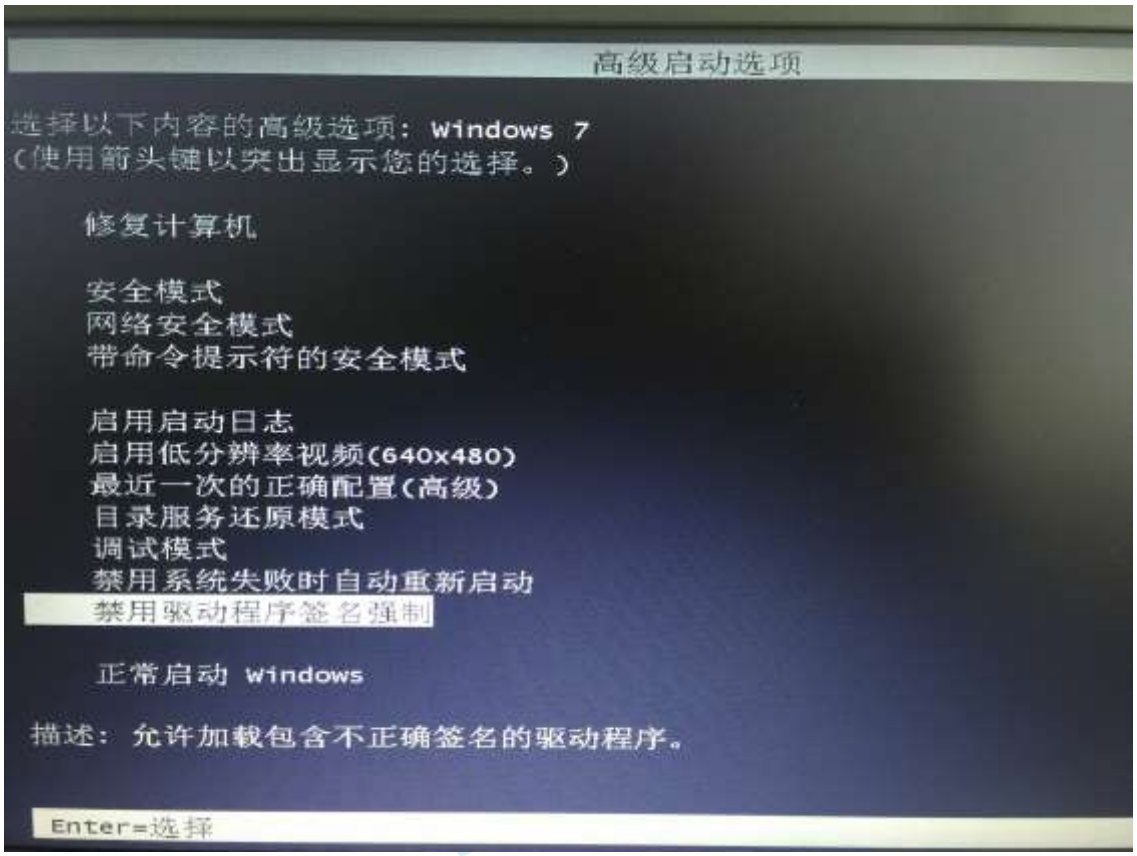


注：若 Windows 7 下，系统并没有自动安装驱动，则驱动的安装方法请参考 **Windows 8（32 位）的安装方法**

Windows 7（64 位）安装方法

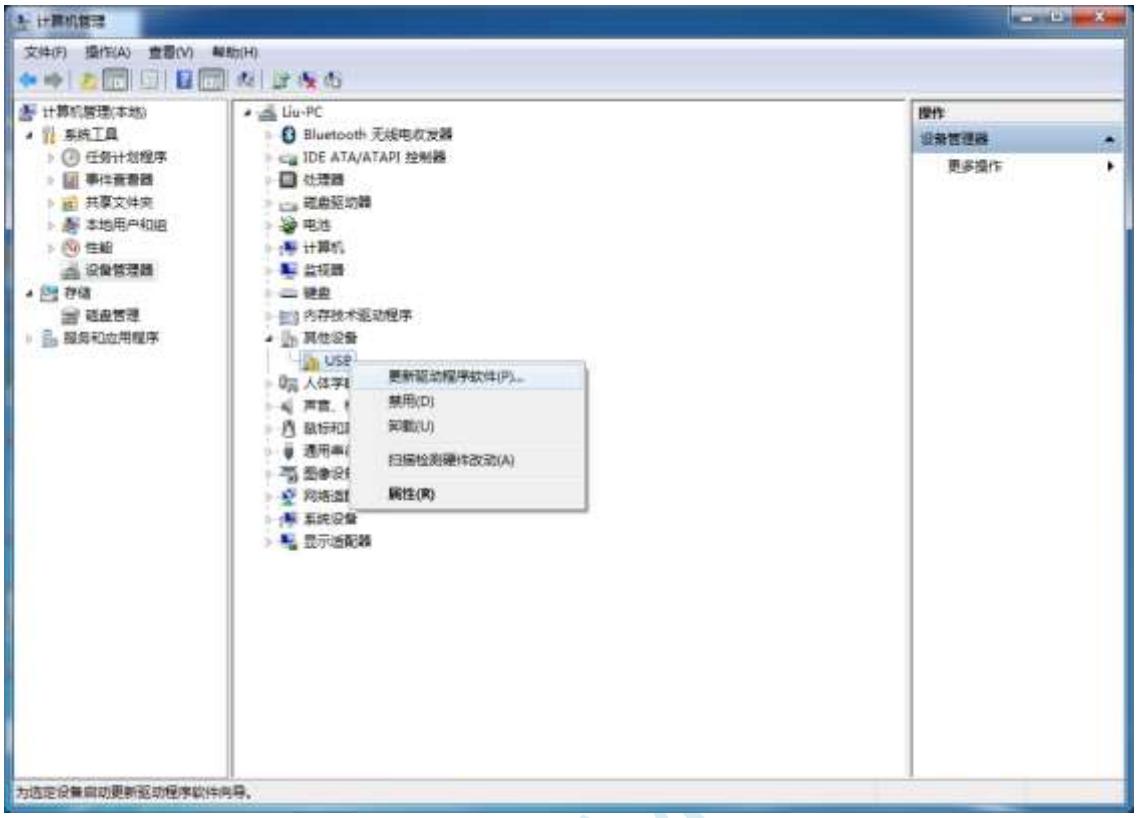
由于 Windows7 64 位操作系统在默认状态下，对于没有数字签名的驱动程序是不能安装成功的。所以在安装 STC-USB 驱动前，需要按照如下步骤，暂时跳过数字签名，即可顺利安装成功。

首先重启电脑，并一直按住 F8，直到出现下面启动画面



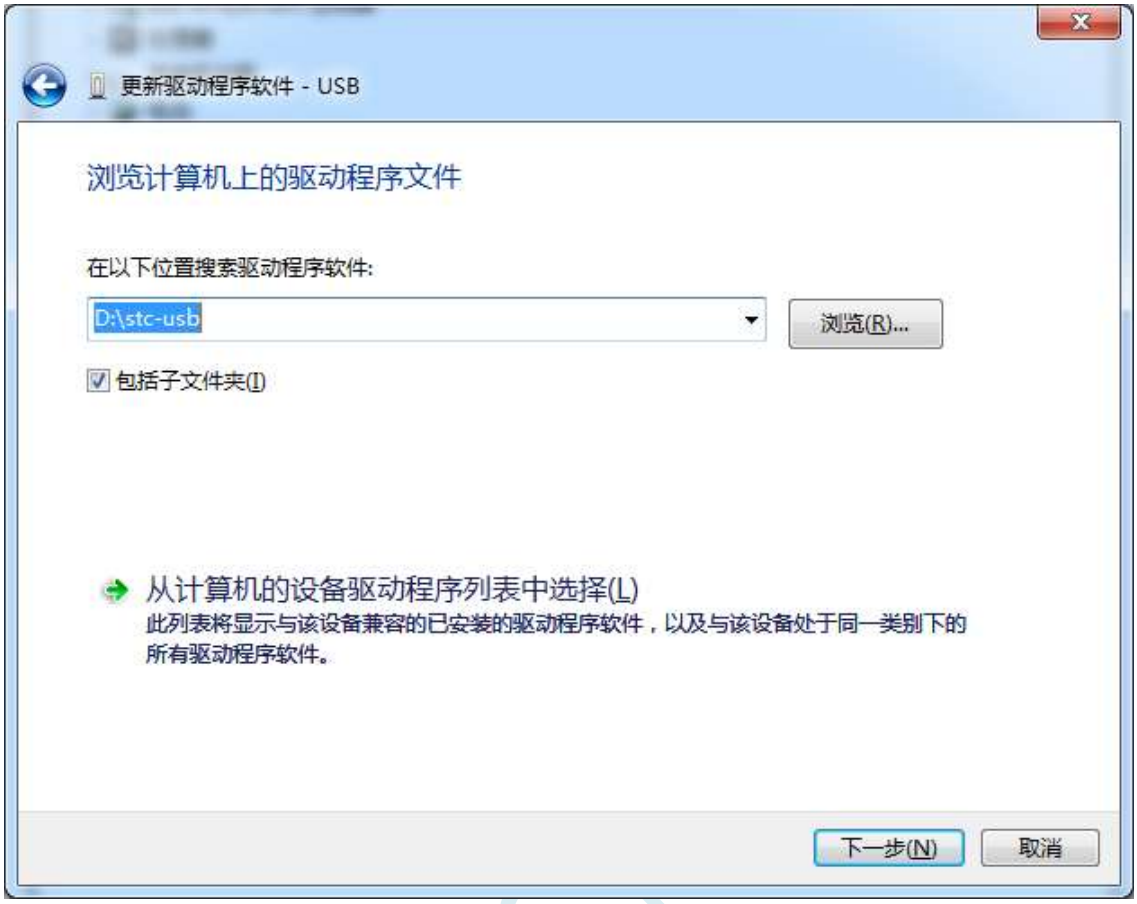
选择“禁用驱动程序签名强制”。启动后即可暂时关闭数字签名验证功能

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”

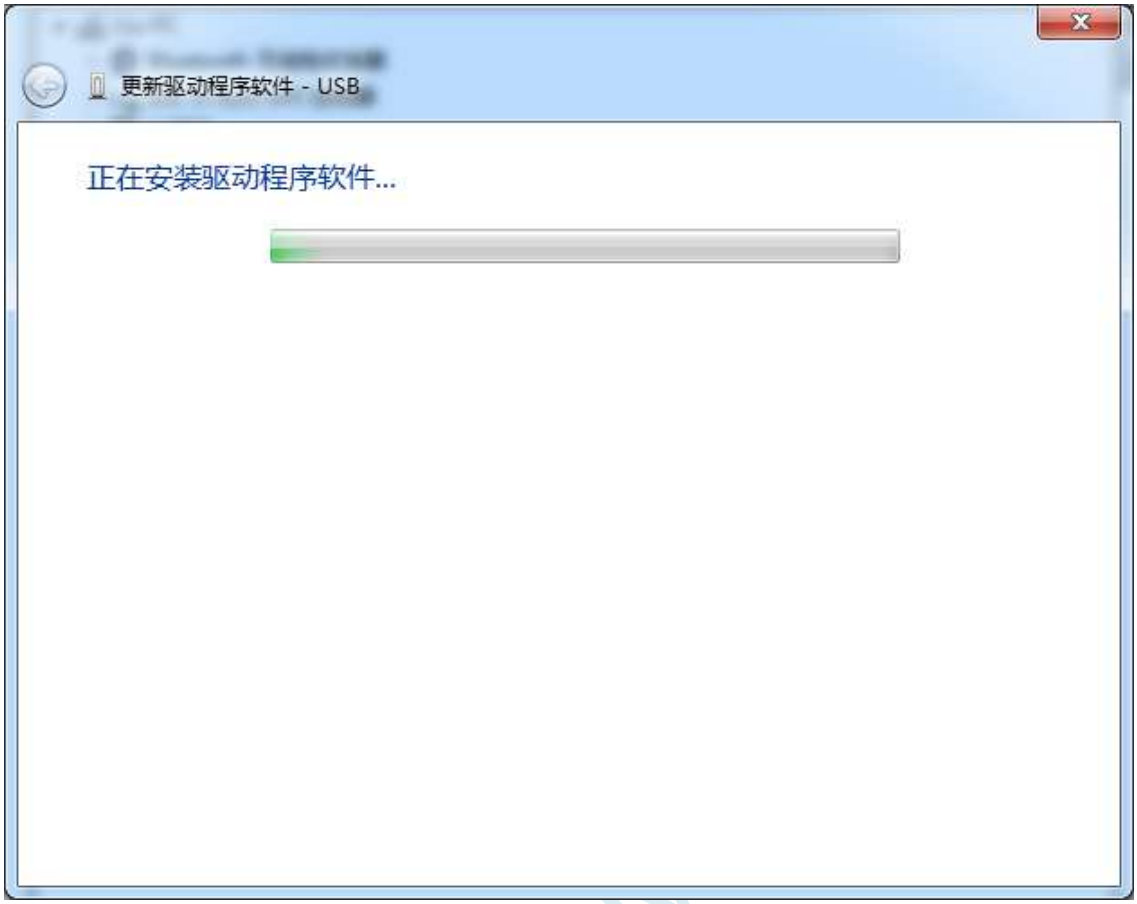
单击下面对话框中的“浏览”按钮，找到之前 STC-USB 驱动程序的存放目录（例如：之前的示例目录为“D:\STC-USB”，用户将路径定位到实际的解压目录）



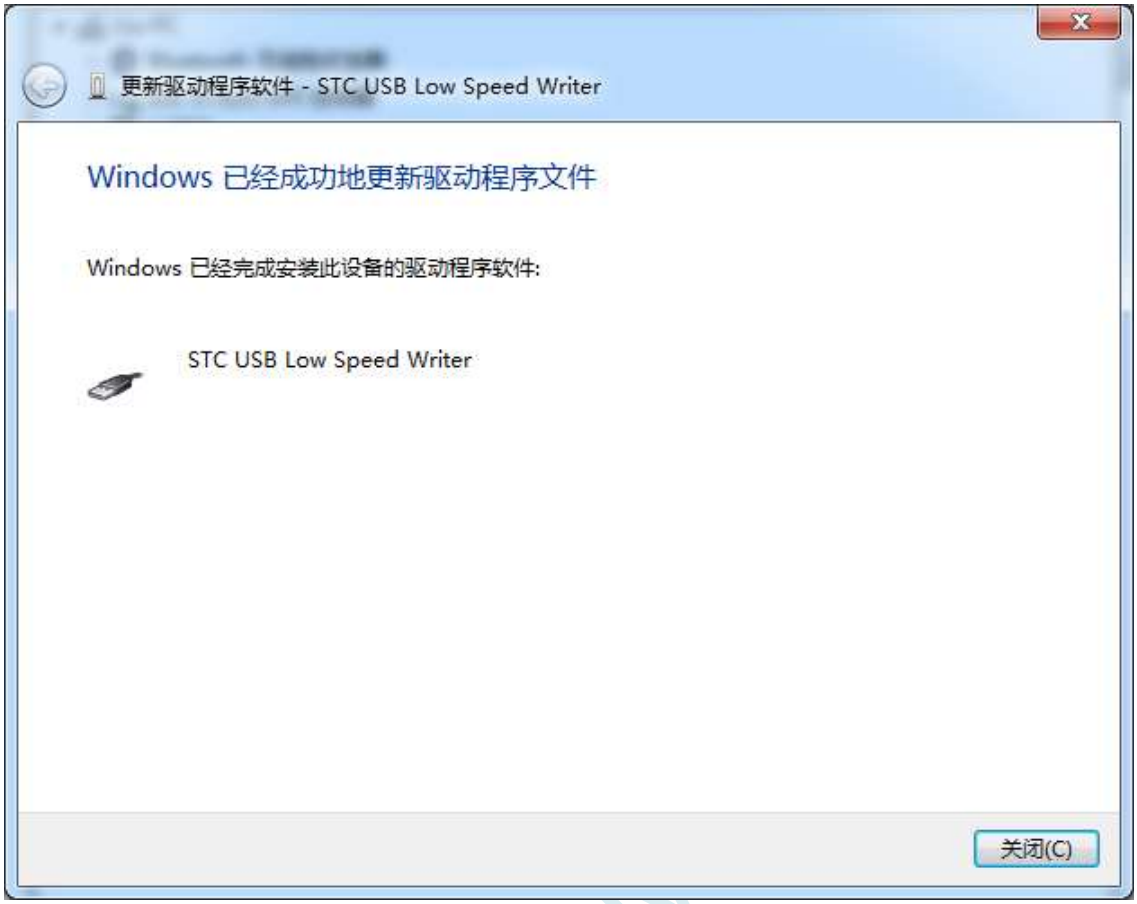
驱动程序开始安装时，会弹出如下对话框，选择“始终安装此驱动程序软件”

STC MCU

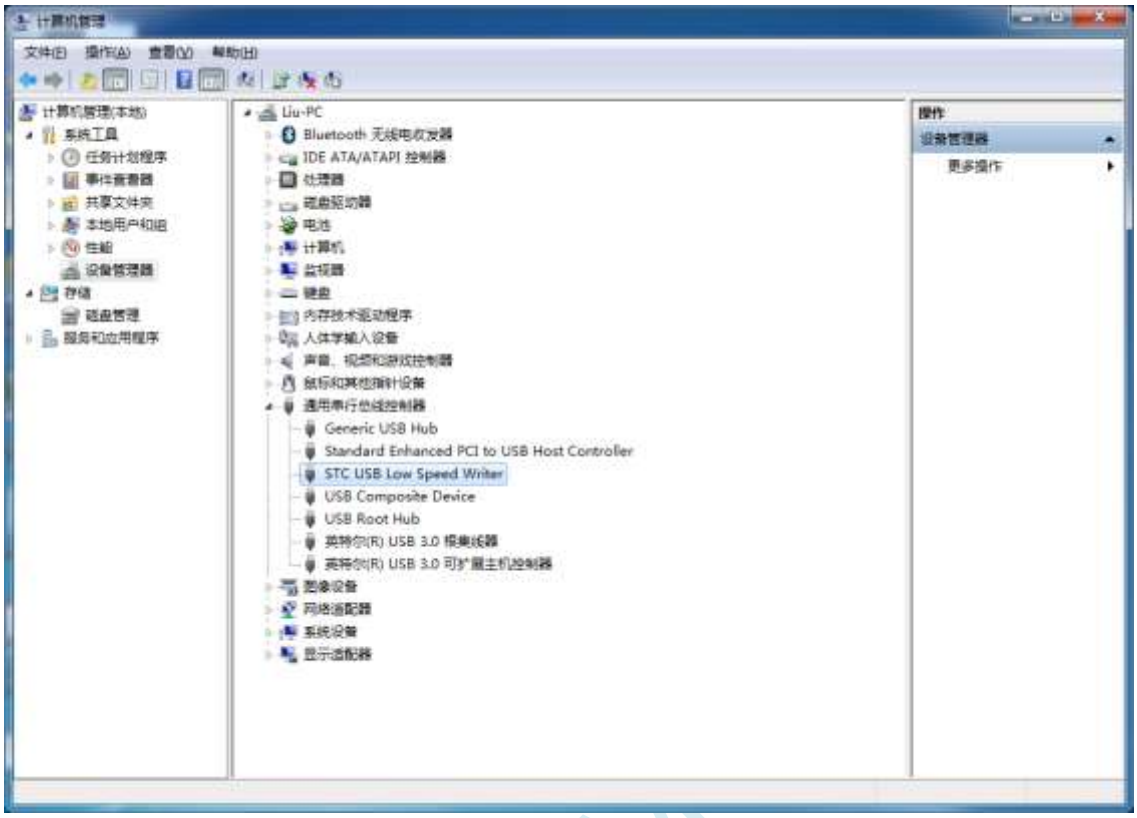
接下来，系统会自动安装驱动，如下图



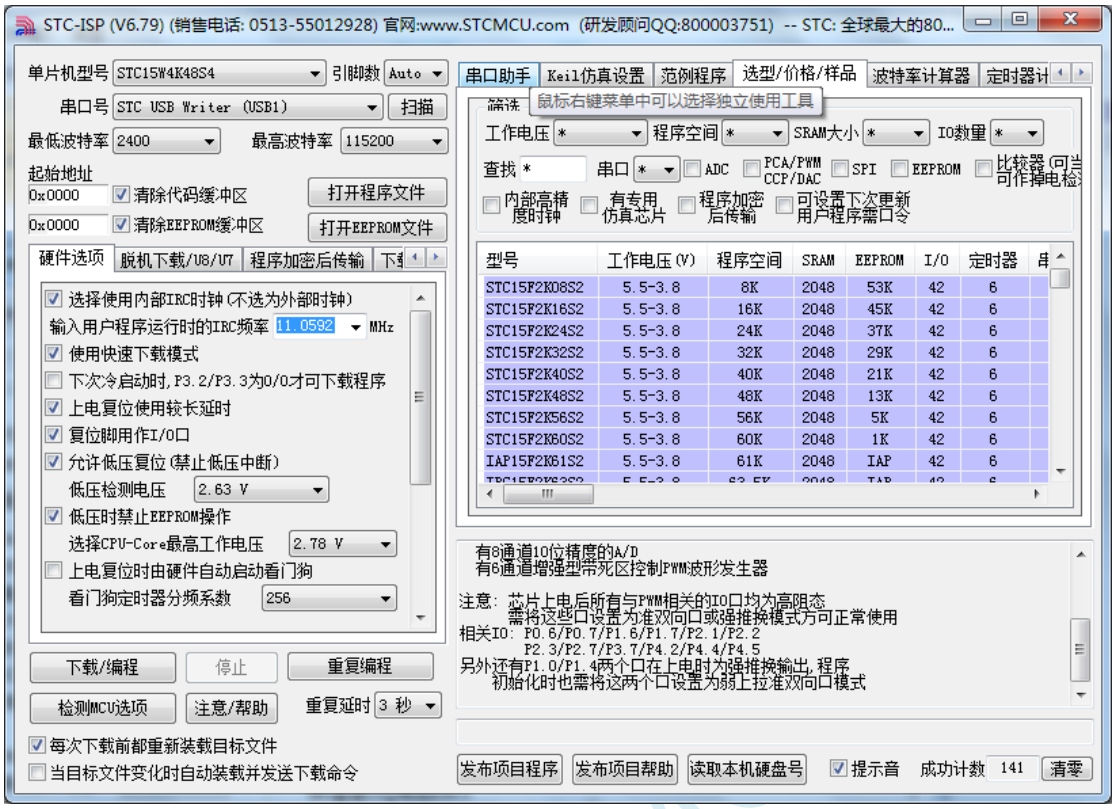
出现下面的的对话框表示驱动安装完成



此时在设备管理器中，之前带有黄色感叹号的设备，此时会显示为“STC USB Low Speed Writer”的设备名

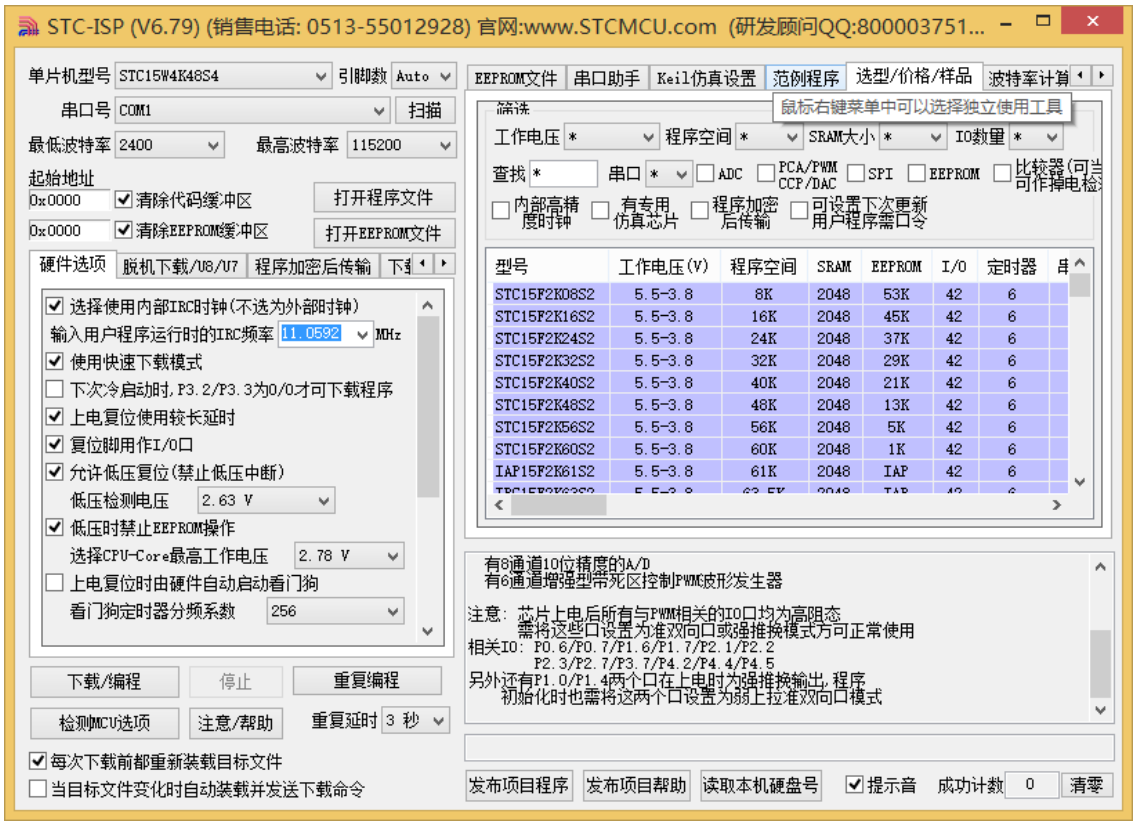


在之前打开的 STC-ISP 下载软件中的串口号列表会自动选择所插入的 USB 设备，并显示设备名称为“STC USB Writer (USB1)”，如下图：

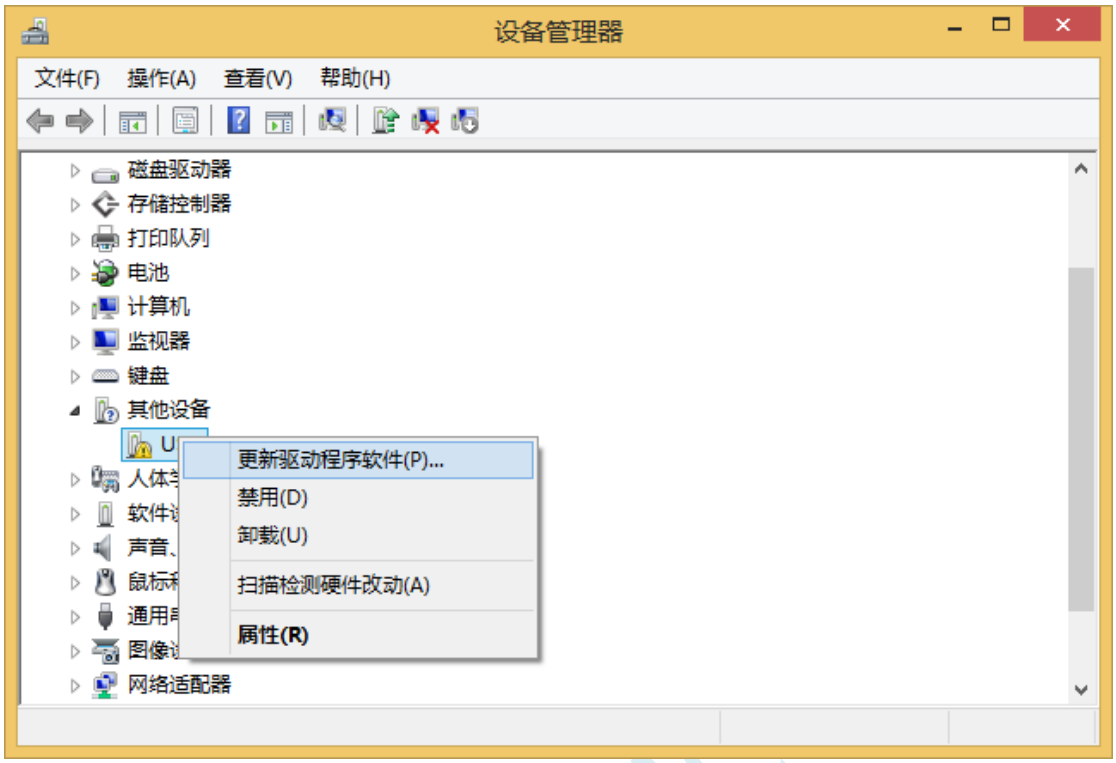


Windows 8（32 位）安装方法

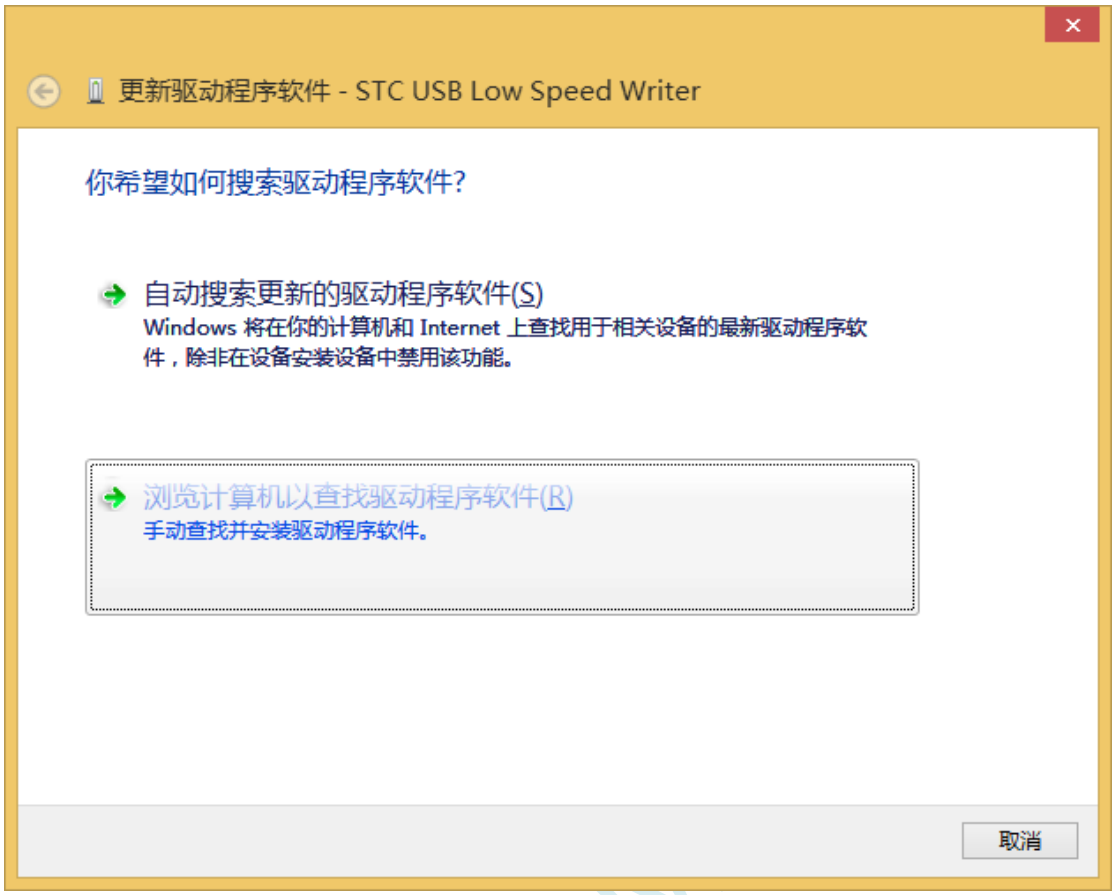
打开 V6.79 版（或者更新的版本）的 STC-ISP 下载软件（由于权限的原因，在 Windows 8 中下载软件不会将驱动文件复制到相关的系统目录，需要用户手动安装。首先从 STC 官方网站下载“stc-isp-15xx-v6.79.zip”（或更新版本），下载后解压到本地磁盘，则 STC-USB 的驱动文件也会被解压到当前解压目录中的“STC-USB Driver”中（例如将下载的压缩文件“stc-isp-15xx-v6.79.zip”解压到“F:\”，则 STC-USB 驱动程序在“F:\STC-USB Driver”目录中））



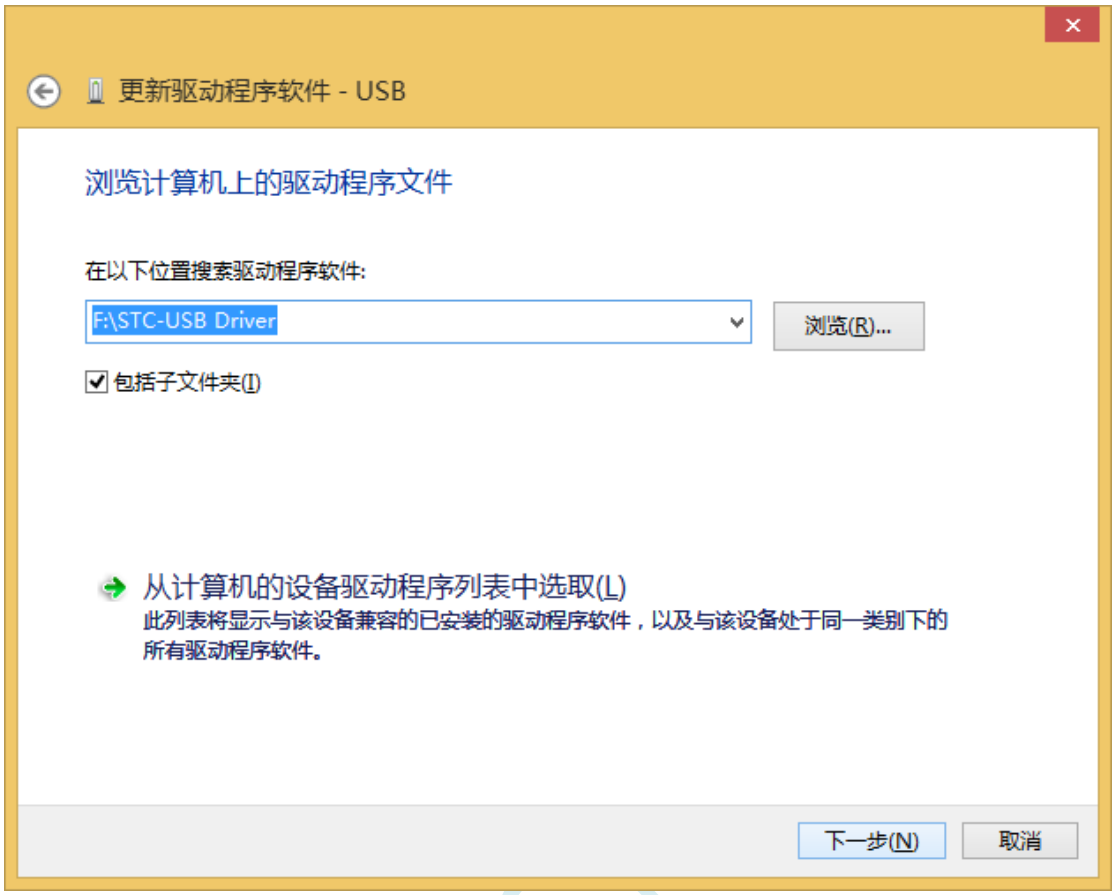
插入 USB 设备，并打开“设备管理器”。找到设备列表中带黄色感叹号的 USB 设备，在设备的右键菜单中，选择“更新驱动程序软件”



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”



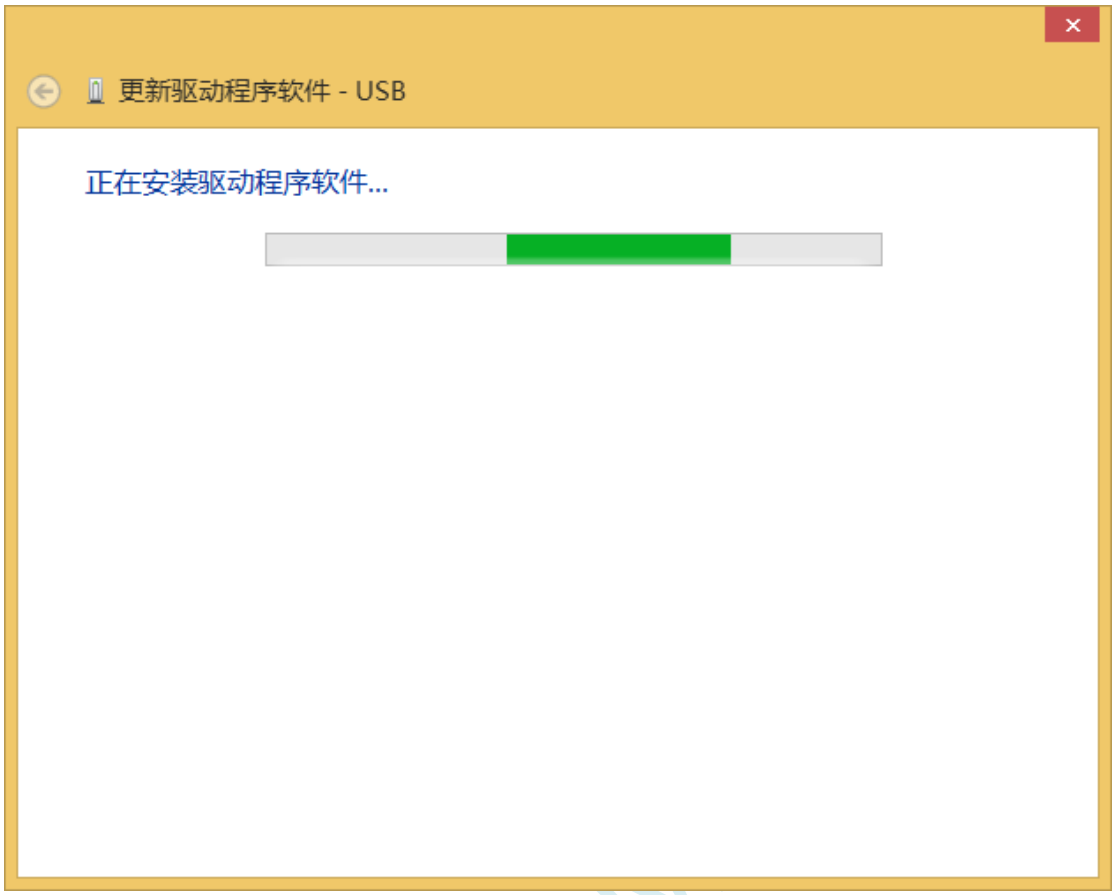
单击下面对话框中的“浏览”按钮，找到之前 STC-USB 驱动程序的存放目录（例如：之前的示例目录为“F:\STC-USB Driver”，用户将路径定位到实际的解压目录）



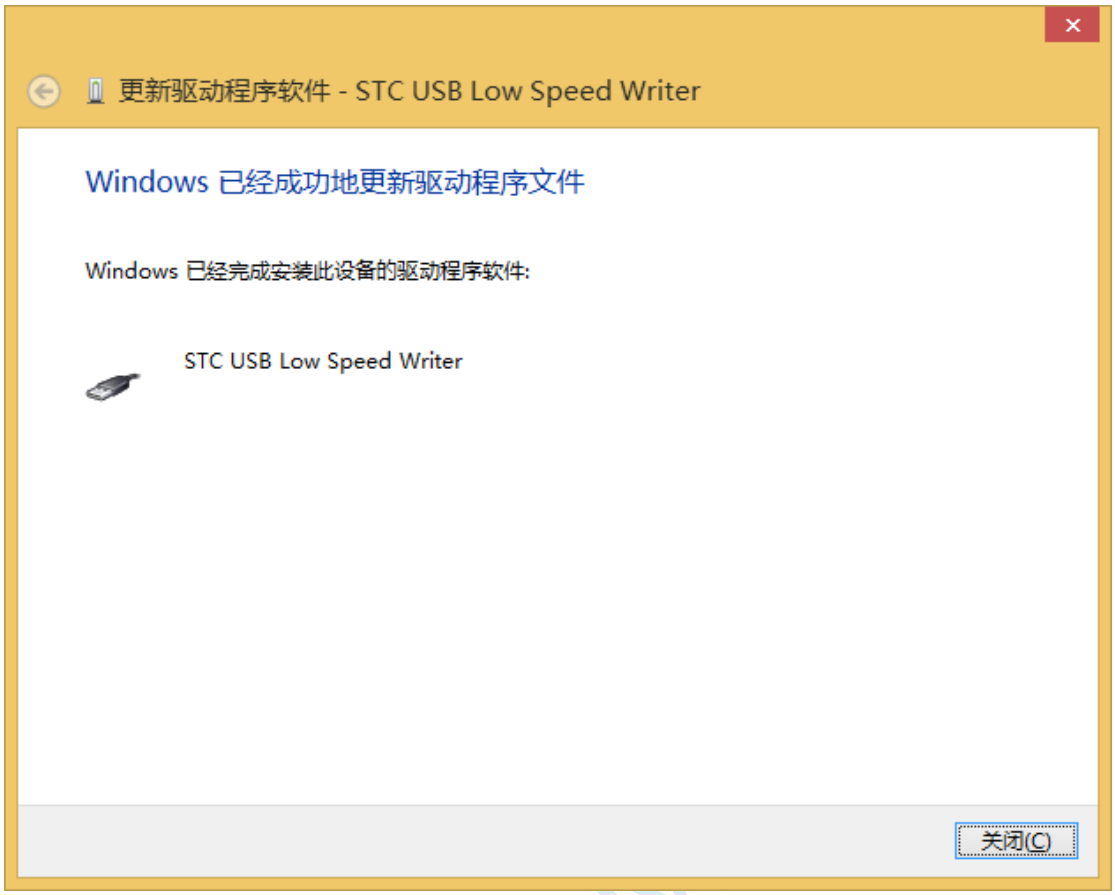
驱动程序开始安装时，会弹出如下对话框，选择“始终安装此驱动程序软件”

STC MCU

接下来，系统会自动安装驱动，如下图

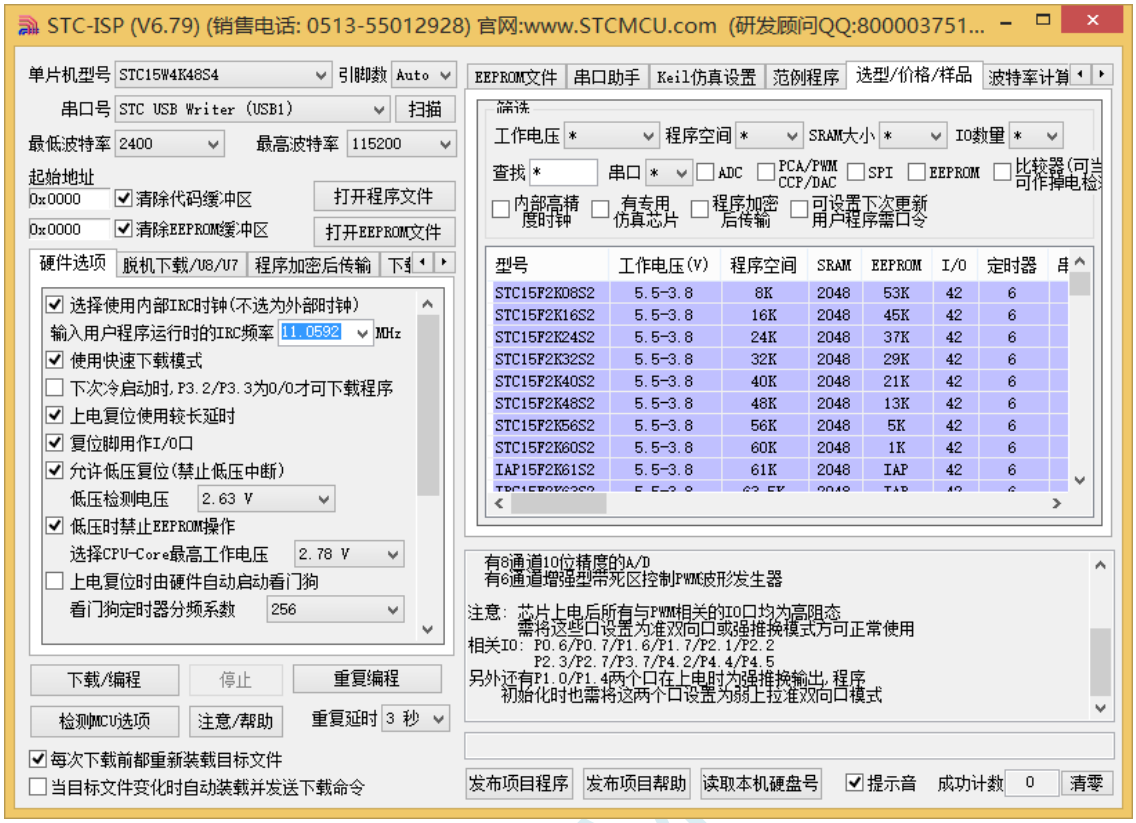


出现下面的的对话框表示驱动安装完成



此时在设备管理器中，之前带有黄色感叹号的设备，此时会显示为“STC USB Low Speed Writer”的设备名

在之前打开的 STC-ISP 下载软件中的串口号列表会自动选择所插入的 USB 设备，并显示设备名称为“STC USB Writer (USB1)”，如下图：

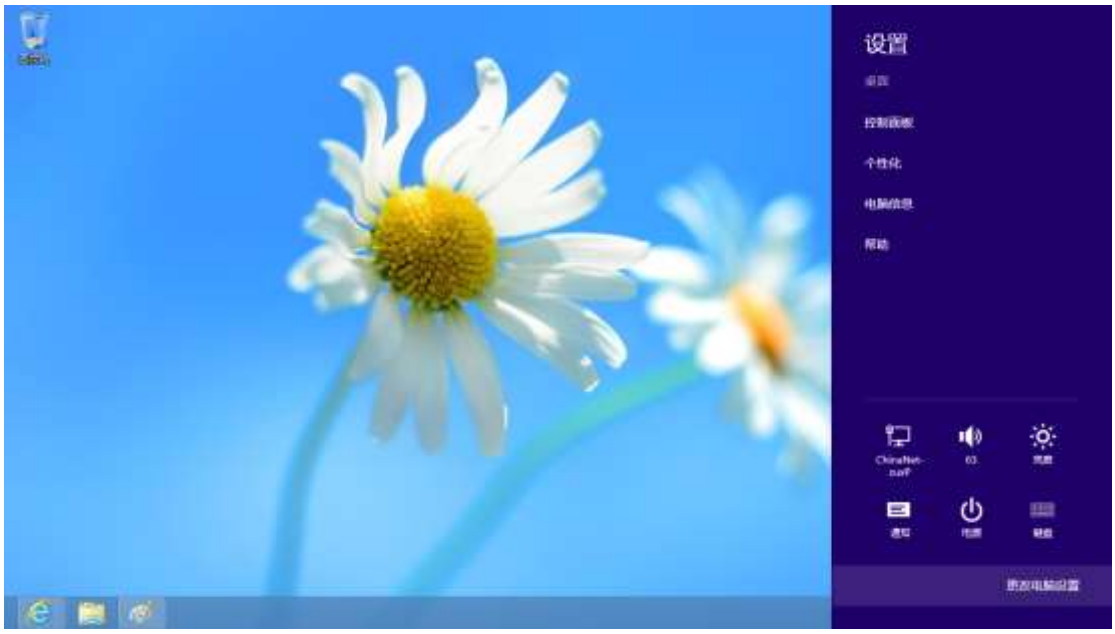


Windows 8（64 位）安装方法

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



STC MCU

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

STC MCU

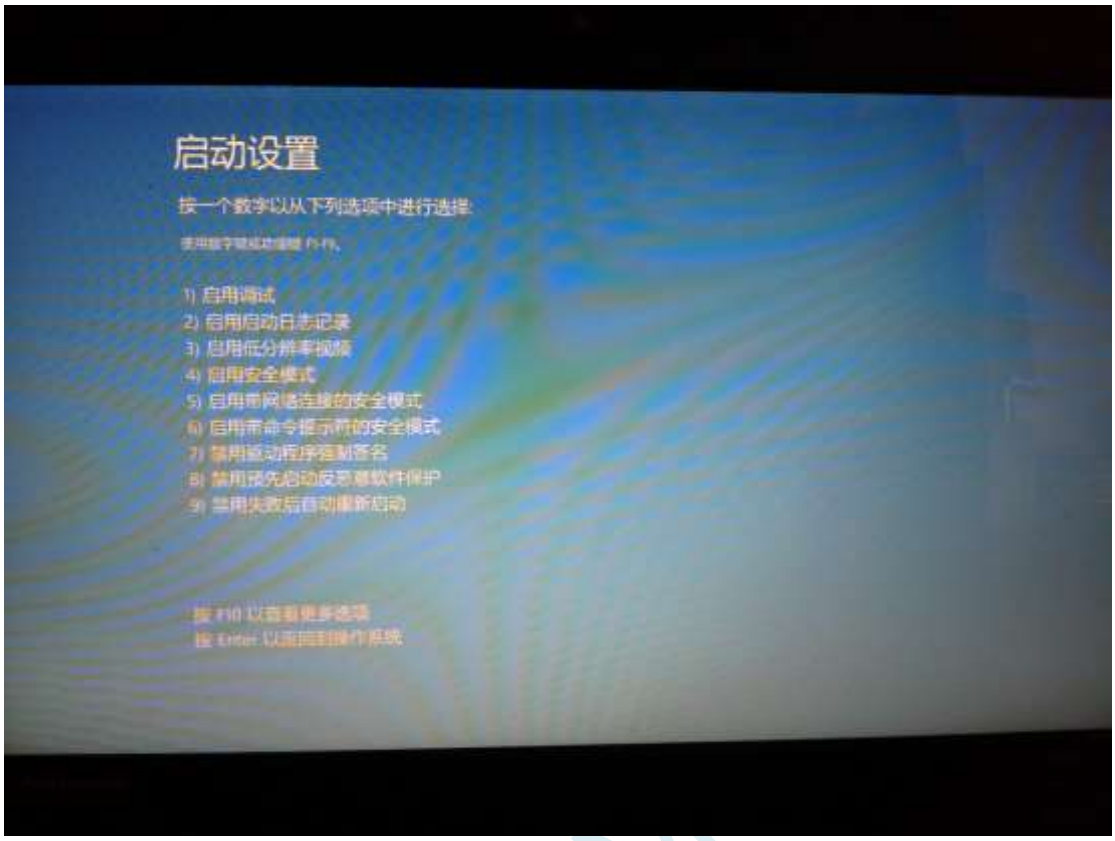
在下面的界面中，选择“疑难解答”项

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



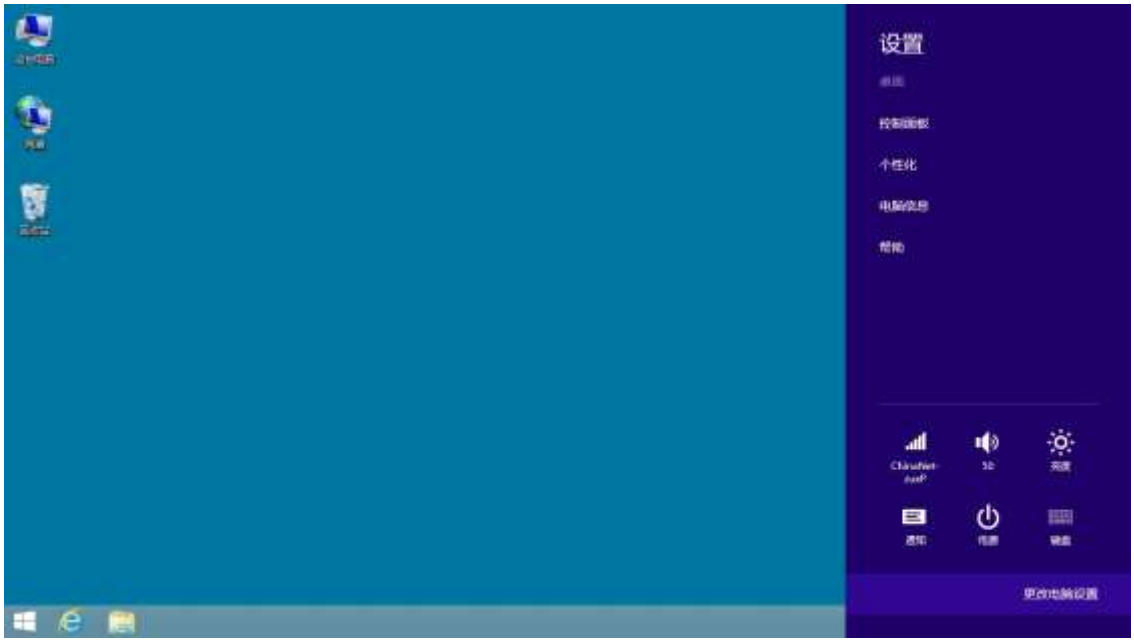
启动到 Windows 8 后，按照 **Windows 8（32 位）的安装方法**即可完成驱动的安装

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



STC MCU

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”）

STC MCU

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

STC MCU

接下来的操作与 Window 8 的步骤相同

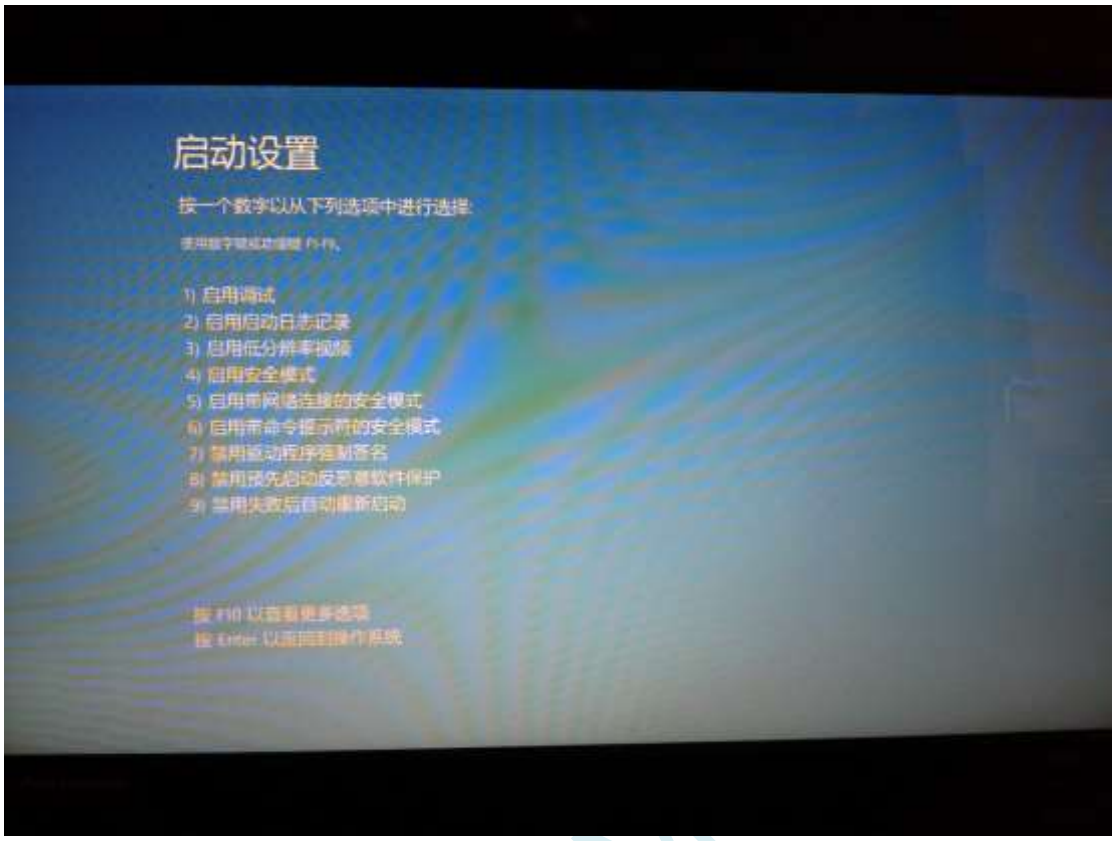
在下面的界面中，选择“疑难解答”项

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

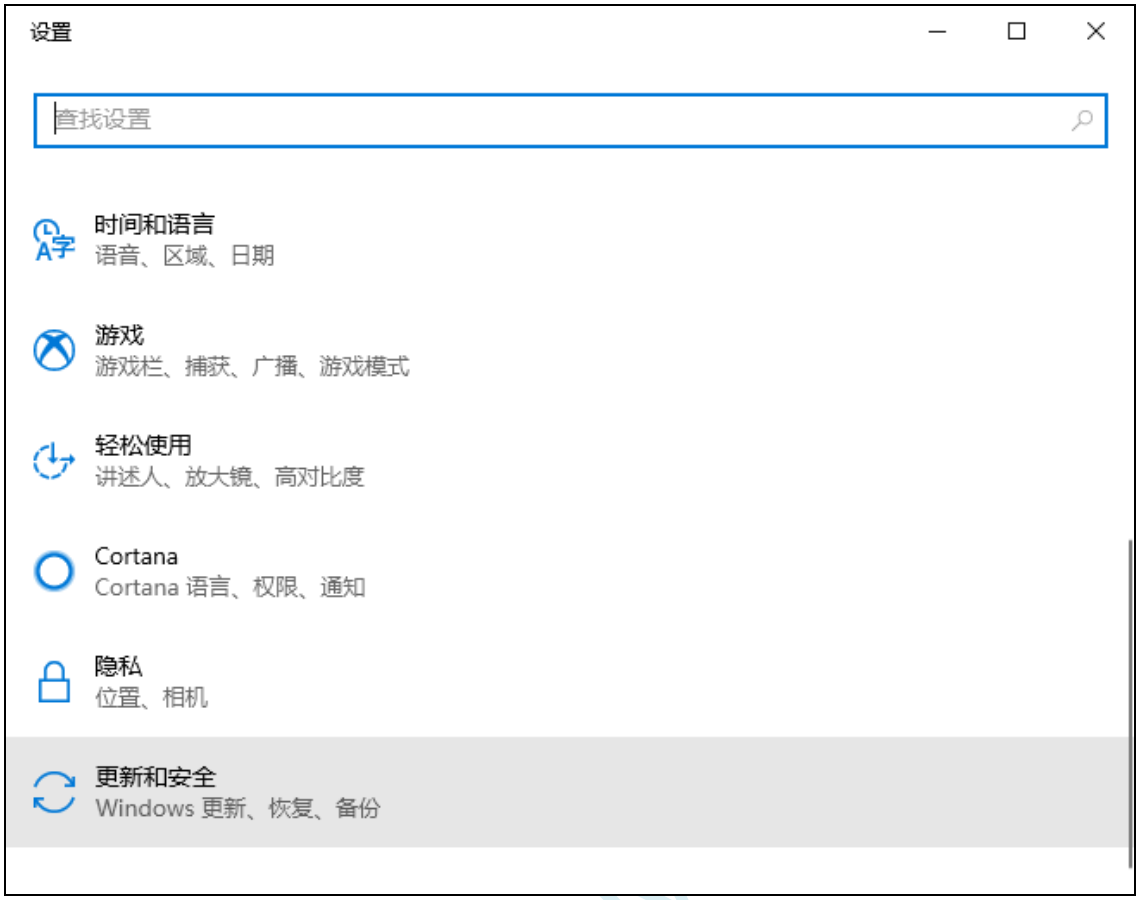
由于 Windows10 64 位操作系统在默认状态下，对于没有数字签名的驱动程序是不能安装成功的。所以在安装 STC-USB 驱动前，需要按照如下步骤，暂时跳过数字签名，即可顺利安装成功。

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

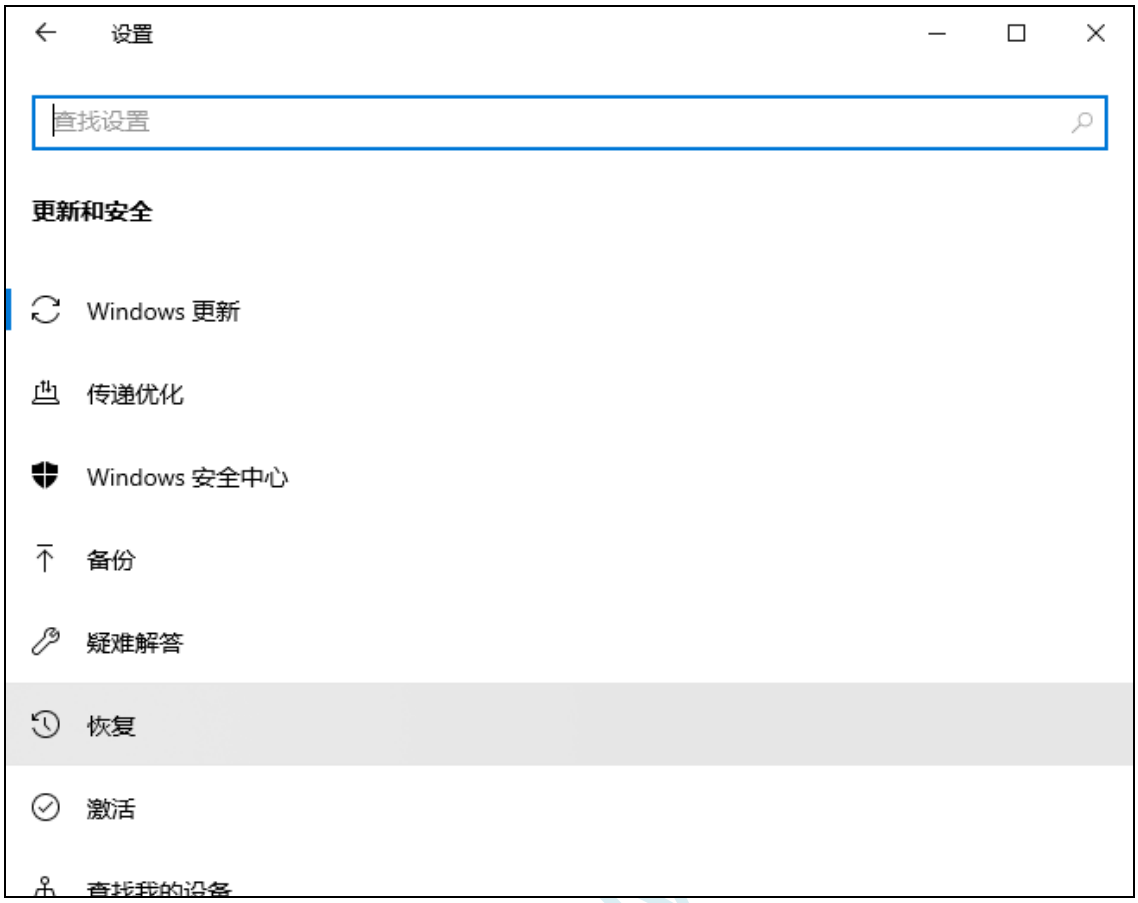
鼠标右键点击“开始”菜单，选择“设置”选项

应用和功能(F)
电源选项(O)
事件查看器(V)
系统(Y)
设备管理器(M)
网络连接(W)
磁盘管理(K)
计算机管理(G)
Windows PowerShell(I)
Windows PowerShell (管理员)(A)
任务管理器(T)
设置(N)
文件资源管理器(E)
搜索(S)
运行(R)
关机或注销(U)
桌面(D)
STC-USB Driver
英
13:33
2018/11/16

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

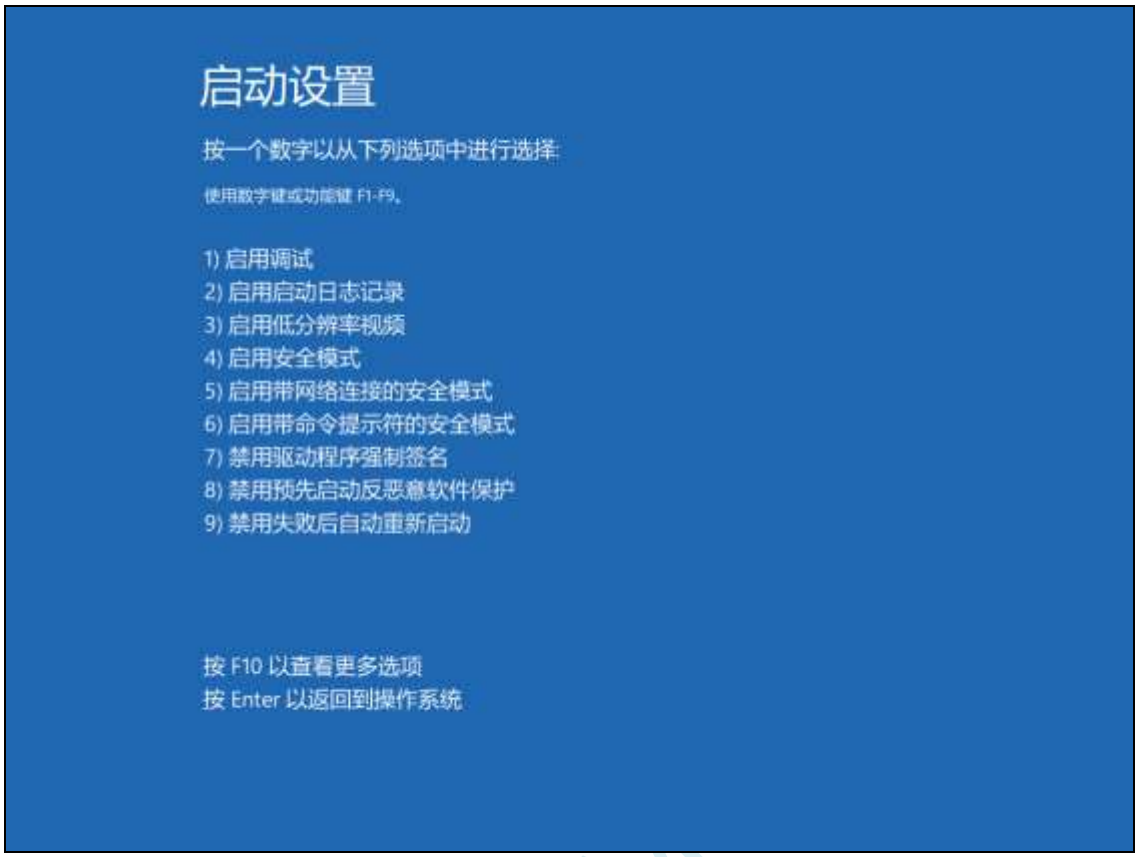
在疑难解答界面中选择“高级选项”

然后选择“查看更多恢复选项”

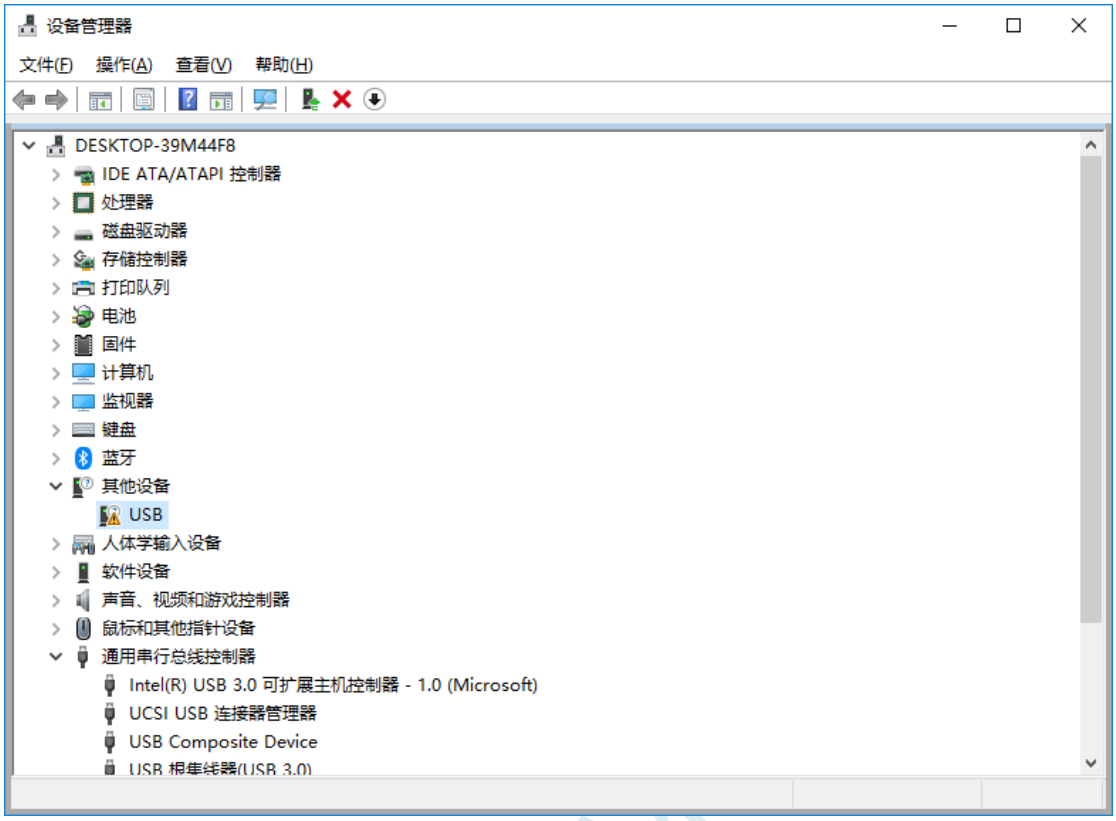
选择“启动设置”项

出现如下画面后，点击“重启”按钮重启电脑

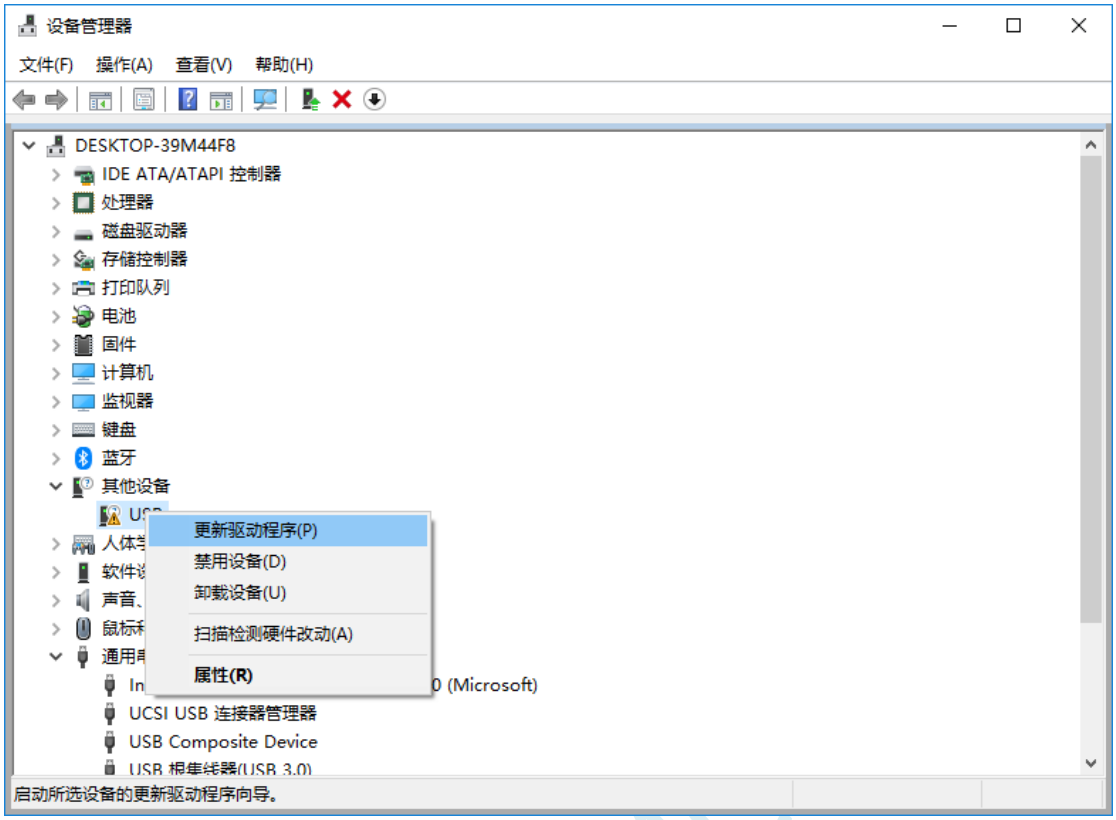
电脑重启后，会弹出“启动设置”界面，按“F7”按钮来选择“禁止驱动程序强制签名”项



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

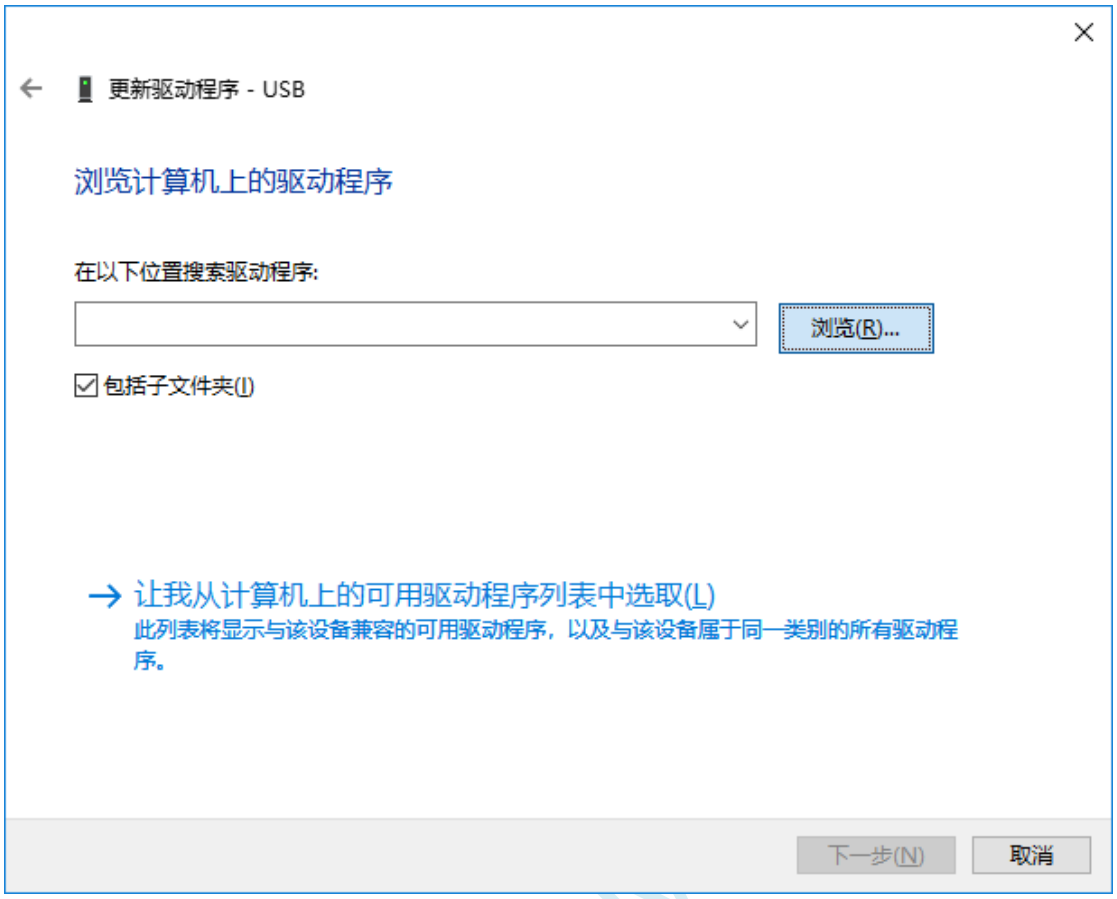


鼠标右键单击未知设备，选择右键菜单中的“更新驱动程序”

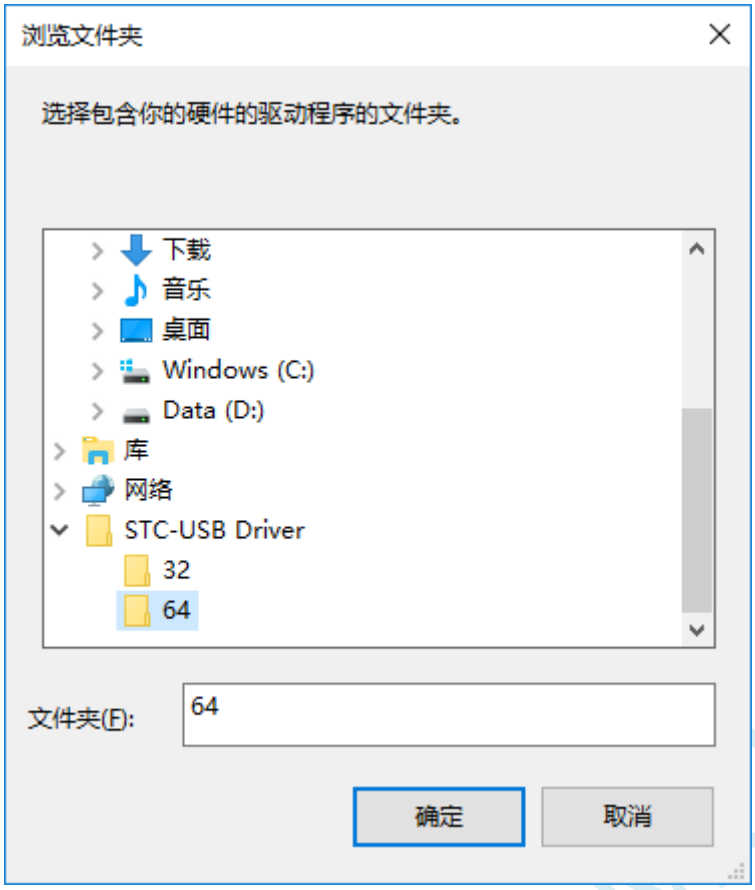


在弹出的驱动安装程序选择画面中，选择“浏览我的计算机以查找驱动程序软件”项

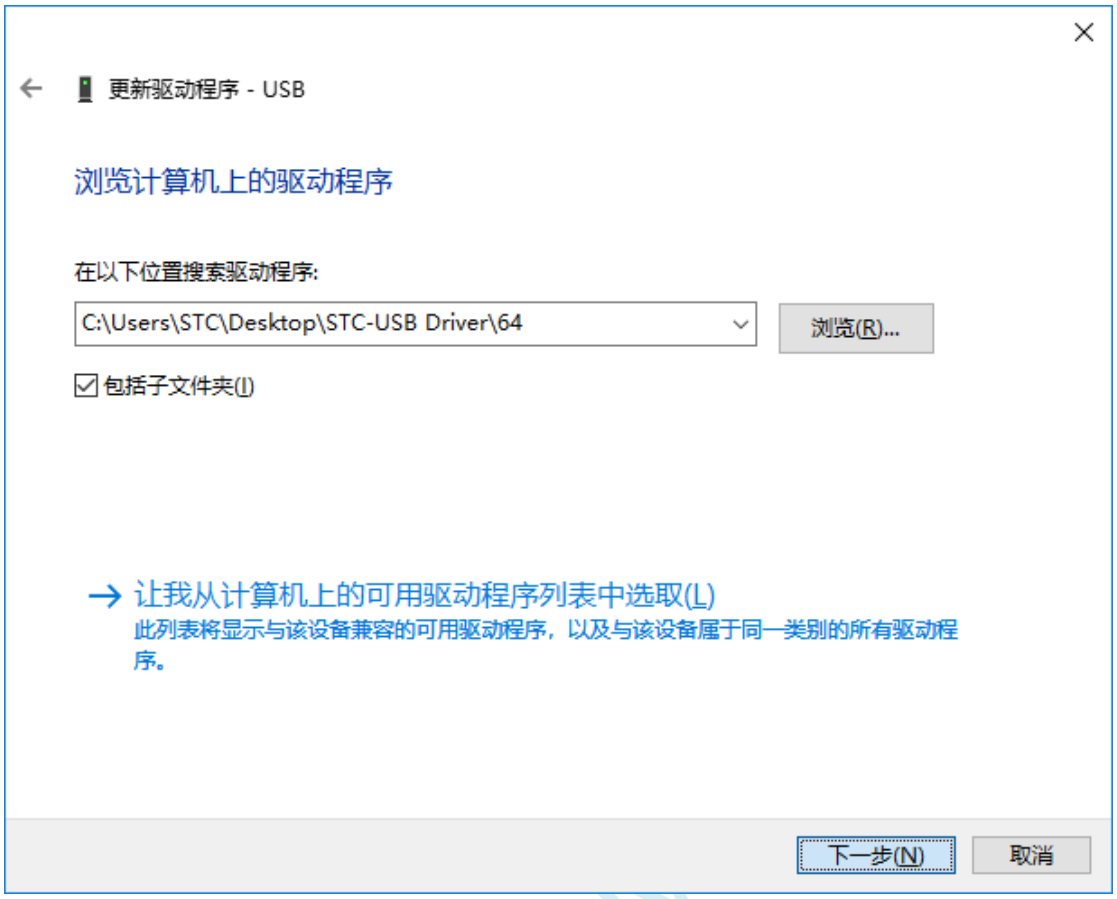
在如下界面中，点击“浏览”按钮



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

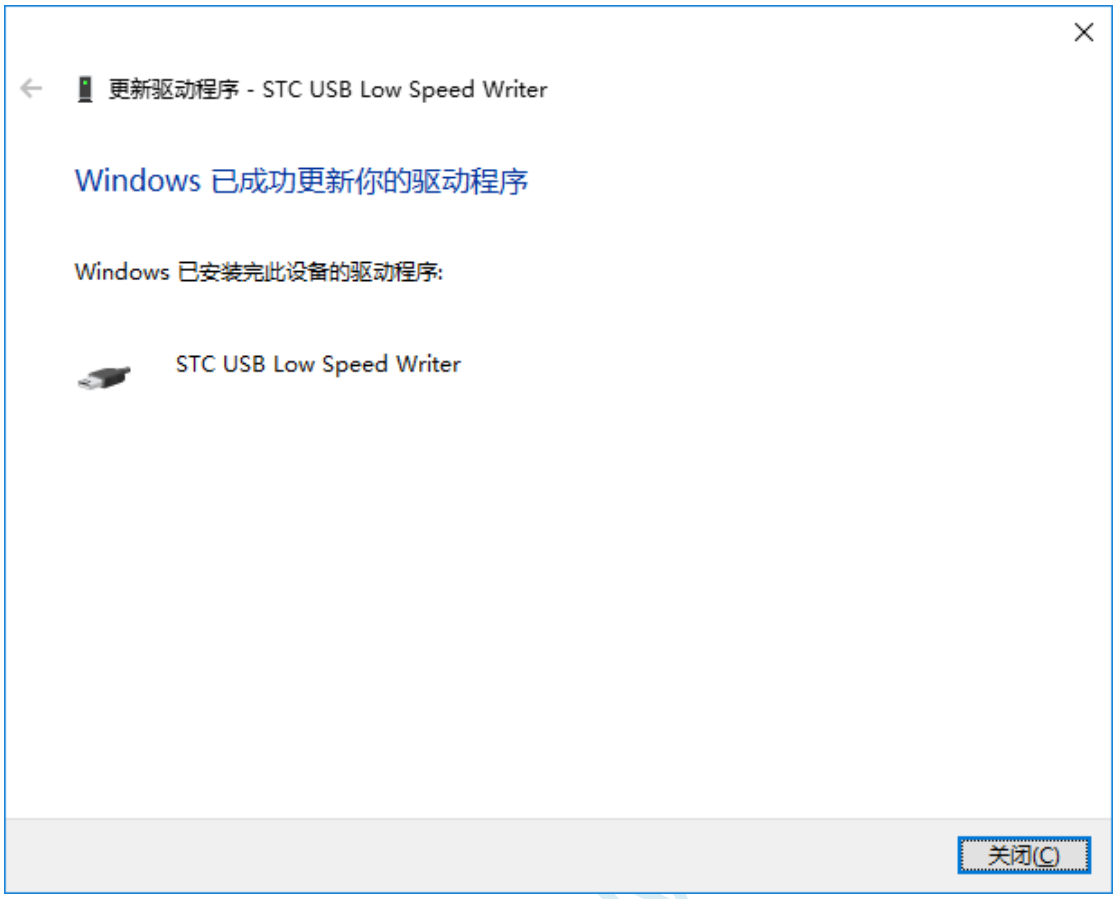


点击“下一步”开始安装驱动

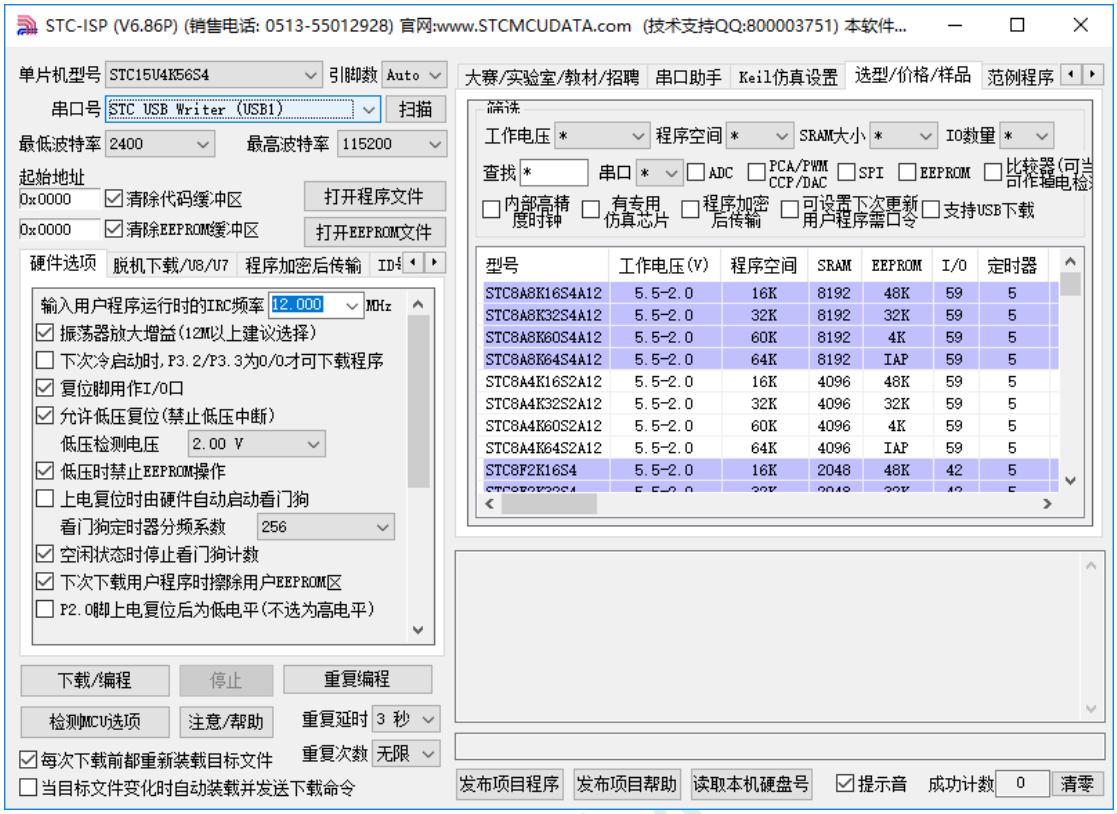


驱动安装的过程中，会弹出如下的警告画面，选择“始终安装此驱动程序软件”

出现下面的画面时，驱动程序就安装成功了

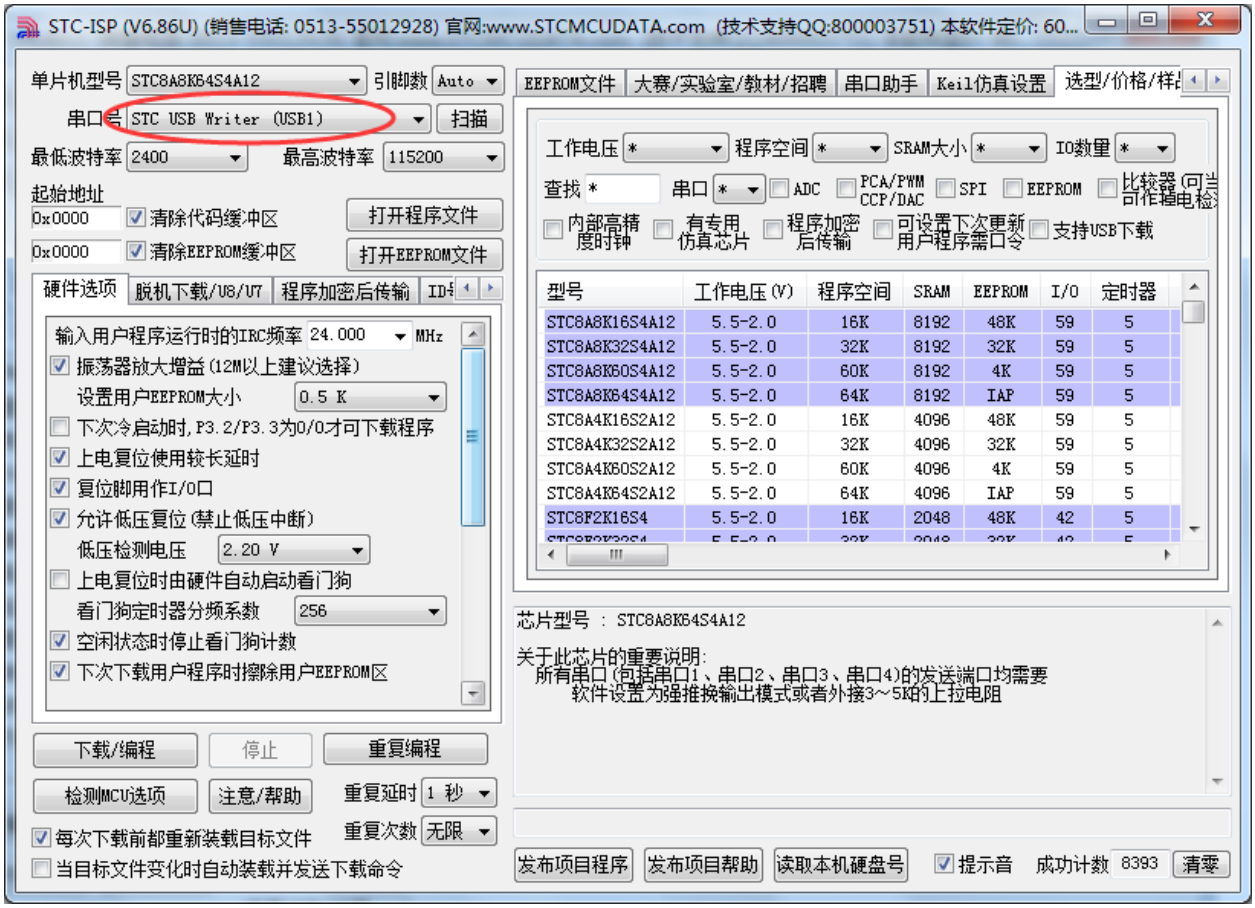


在回到 STC-ISP 的下载软件，此时“串口号”的下拉列表中已自动选择了“STC USB Writer (USB1)”，即可使用 USB 进行 ISP 下载了

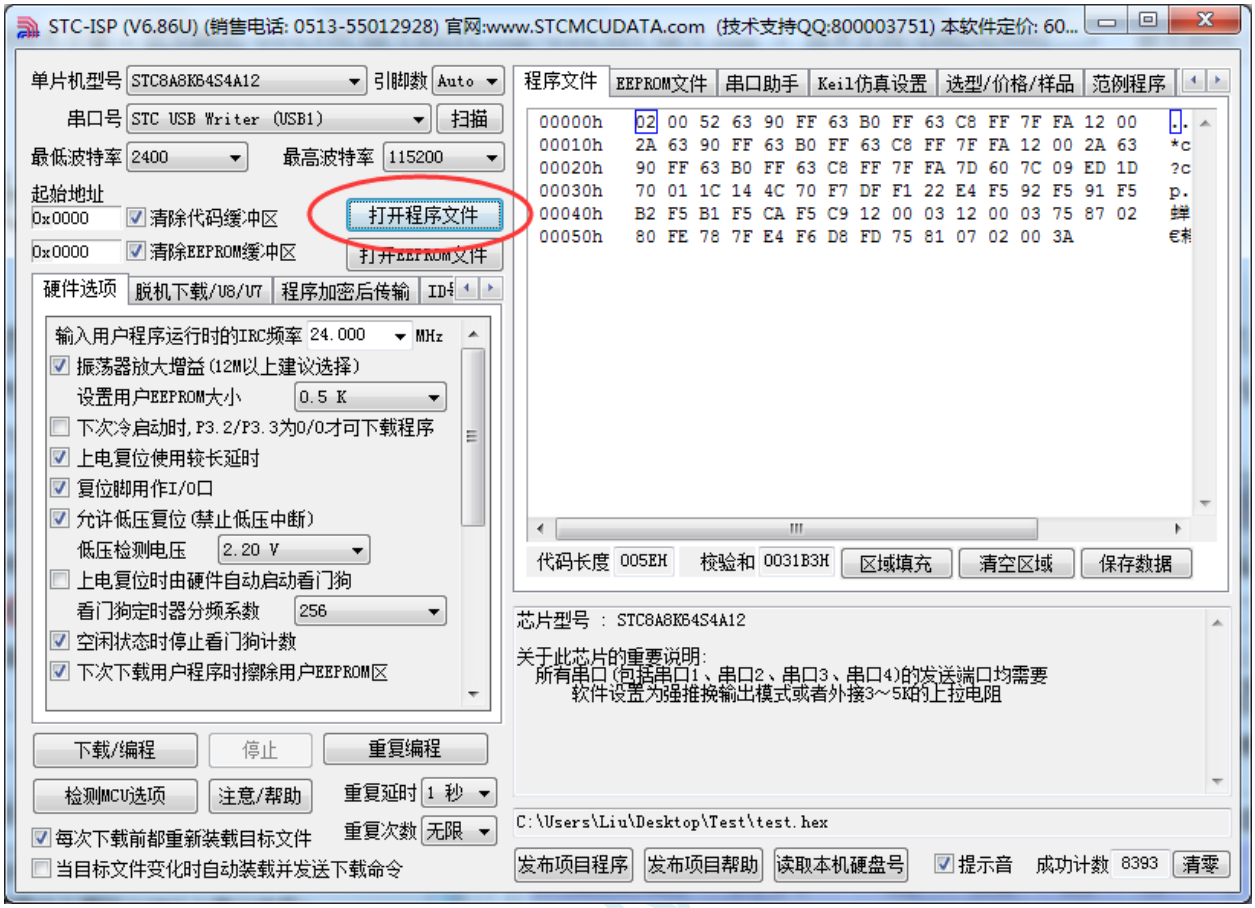


附录D　USB 下载步骤演示

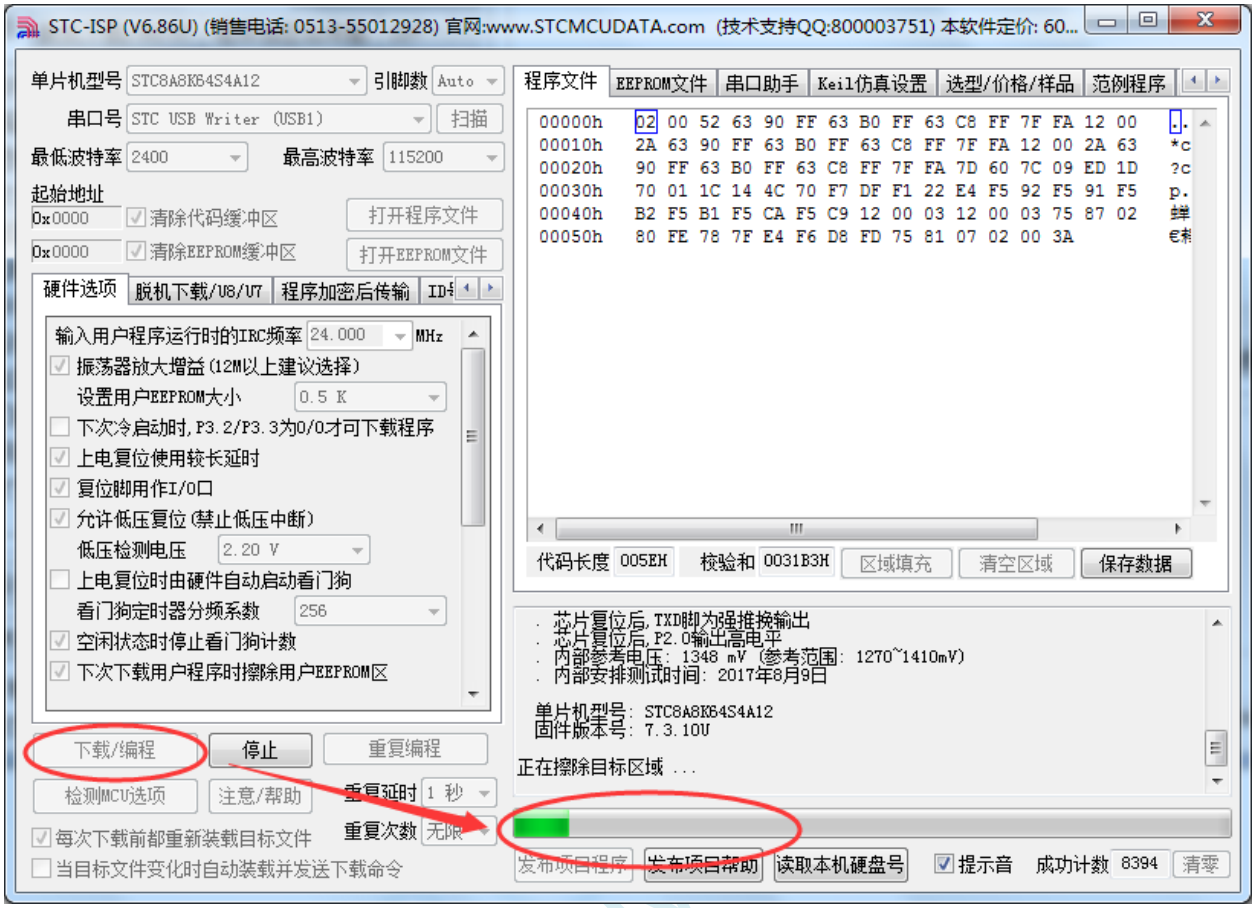
1、首先参考 P5.1.5 章的应用线路图连接好单片机，并将目标芯片的 P3.2 口连接到 Gnd，然后将系统连接到 PC 端的 USB 端口上。打开 ISP 下载软件，即可在下载软件的串口号中自动搜索到“STC USB Writer (USB1)”的 USB 设备



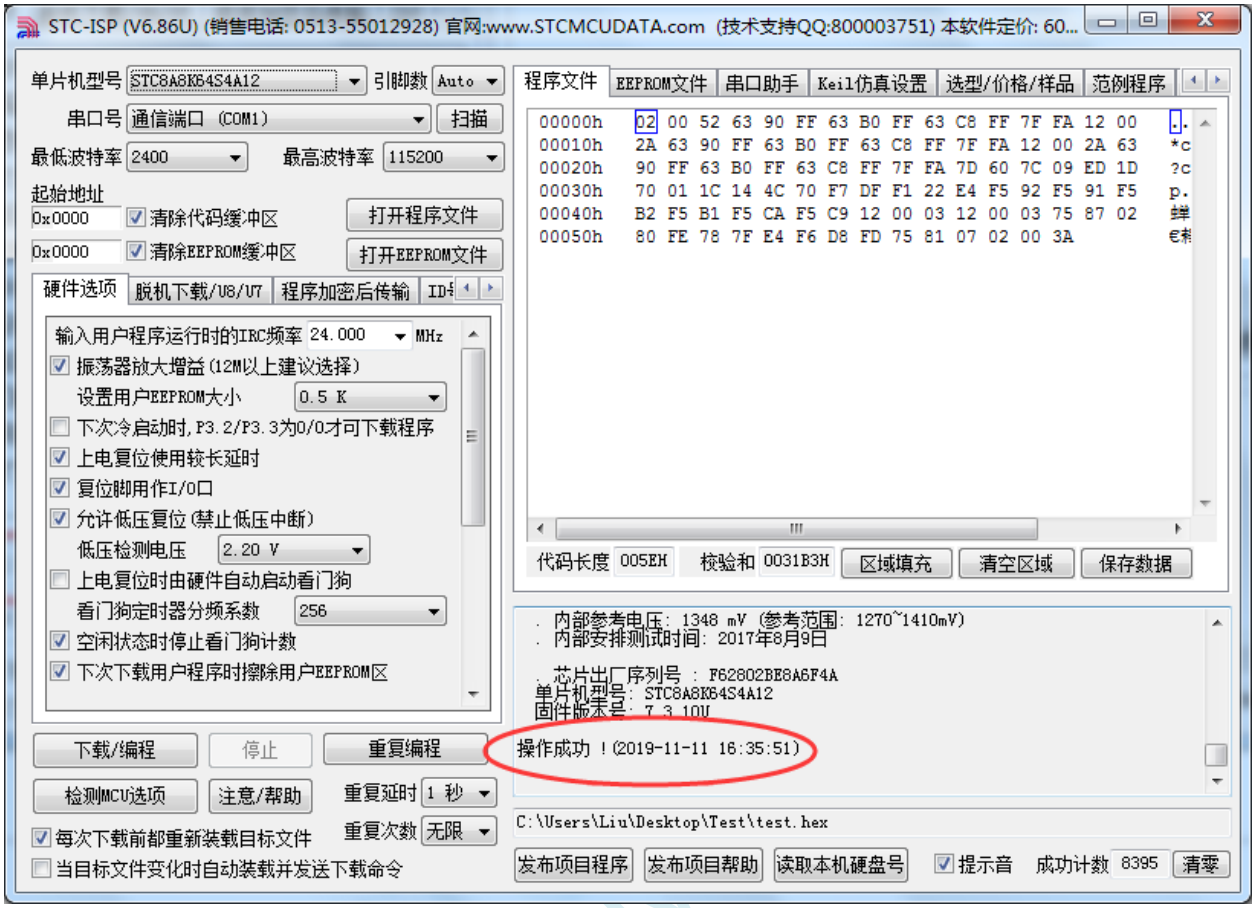
2、打开用户代码程序


STC-ISP (V6.86U) (销售电话: 0513-55012928) 官网:www.STCMCUDATA.com (技术支持QQ:800003751) 本软件定价: 60...
单片机型号 STC8A8K64S4A12
引脚数 Auto
串口号 STC USB Writer (USB1)
扫描
最低波特率 2400
最高波特率 115200
起始地址
0x0000
清除代码缓冲区
打开程序文件
0x0000
清除EEPROM缓冲区
打开EEPROM文件
硬件选项
脱机下载/U8/U7
程序加密后传输
输入用户程序运行时的IRC频率 24.000 MHz
振荡器放大增益(12M以上建议选择)
设置用户EEPROM大小 0.5 K
下次冷启动时,P3.2/P3.3为0/0才可下载程序
上电复位使用较长延时
复位脚用作I/O口
允许低压复位(禁止低压中断)
低压检测电压 2.20 V
上电复位时由硬件自动启动看门狗
看门狗定时器分频系数 256
空闲状态时停止看门狗计数
下次下载用户程序时擦除用户EEPROM区
下载/编程
停止
重复编程
检测MCU选项
注意/帮助
重复延时 1 秒
重复次数 无限
每次下载前都重新装载目标文件
当目标文件变化时自动装载并发送下载命令
程序文件
EEPROM文件
串口助手
Keil仿真设置
选型/价格/样品
范例程序
00000h 02 00 52 63 90 FF 63 B0 FF 63 C8 FF 7F FA 12 00
00010h 2A 63 90 FF 63 B0 FF 63 C8 FF 7F FA 12 00 2A 63
00020h 90 FF 63 B0 FF 63 C8 FF 7F FA 7D 60 7C 09 ED 1D
00030h 70 01 1C 14 4C 70 F7 DF F1 22 E4 F5 92 F5 91 F5
00040h B2 F5 B1 F5 CA F5 C9 12 00 03 12 00 03 75 87 02
00050h 80 FE 78 7F E4 F6 D8 FD 75 81 07 02 00 3A
代码长度 005EH
校验和 0031B3H
区域填充
清空区域
保存数据
芯片型号 : STC8A8K64S4A12
关于此芯片的重要说明:
所有串口(包括串口1、串口2、串口3、串口4)的发送端口均需要
软件设置为强推挽输出模式或者外接3~5K的上拉电阻
C:\Users\Liu\Desktop\Test\test.hex
发布项目程序
发布项目帮助
读取本机硬盘号
提示音
成功计数 8393
清零

3、点击“下载/编程”按钮开始下载用户代码

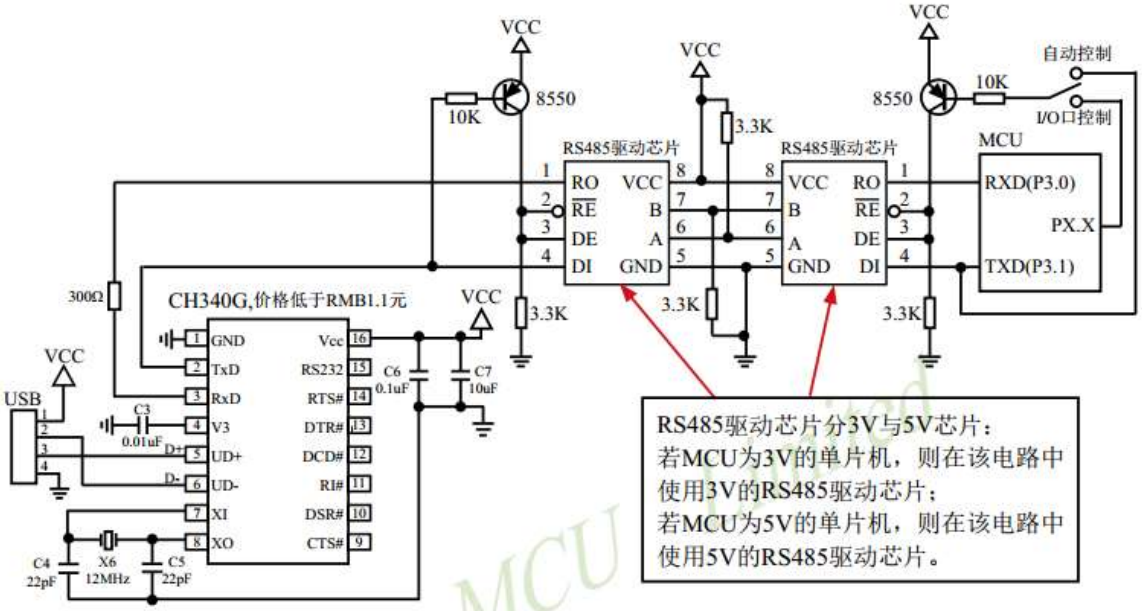


4、直到提示“操作成功”，表示程序代码下载完成。

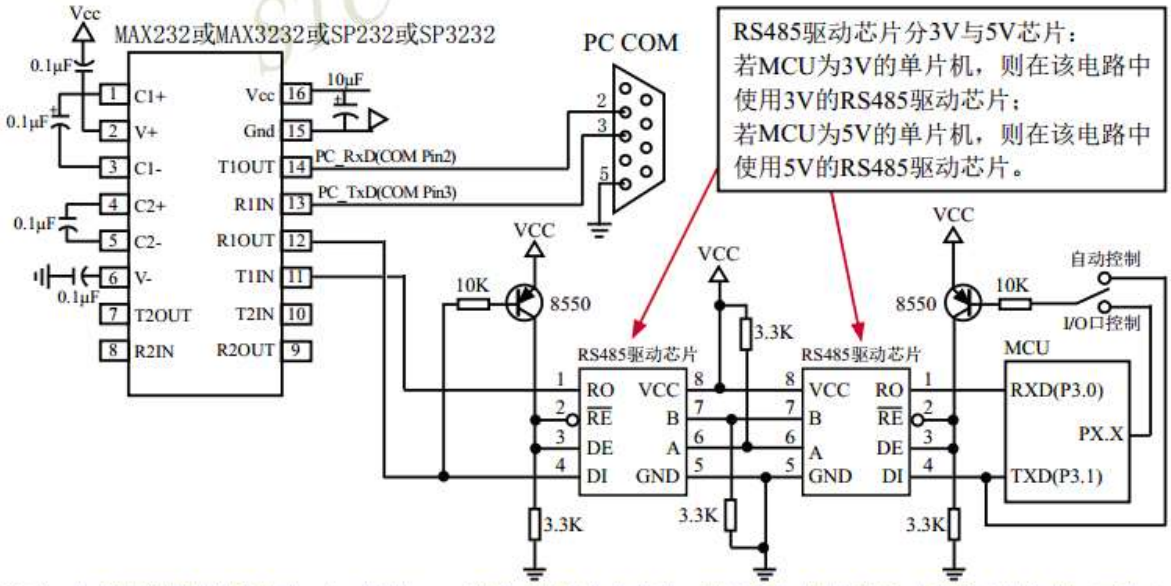


附录E　RS485 自动控制或 I/O 口控制线路图

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

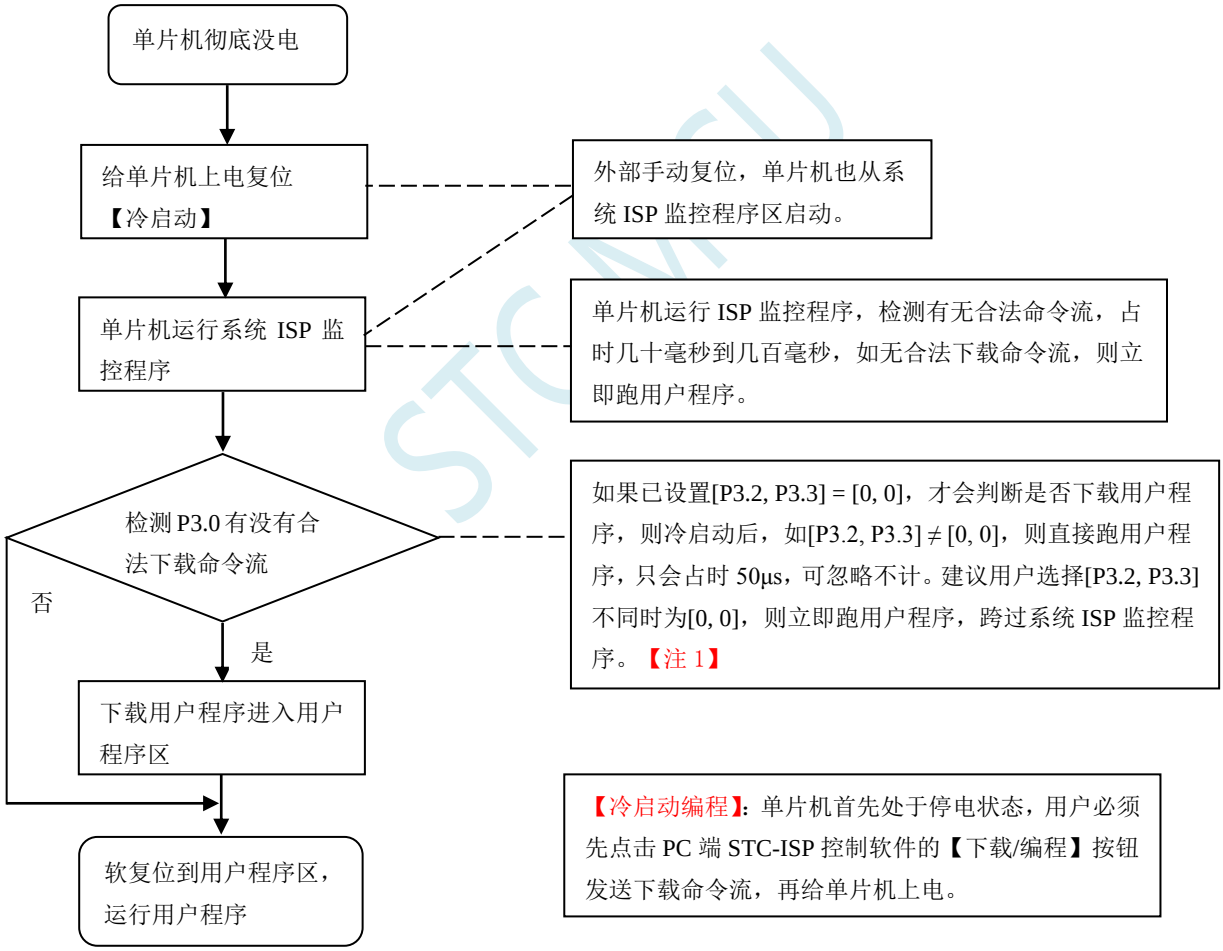
附录F　STC 工具使用说明书

F.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

F.2　系统可编程（ISP）流程说明

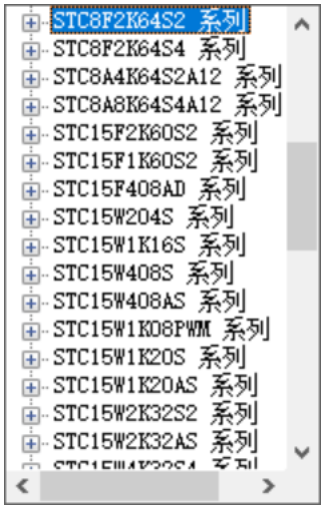


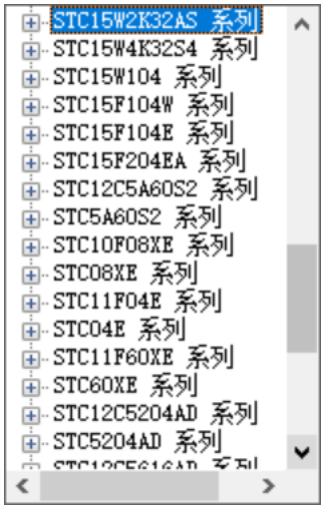
注意：因 [P3.0, P3.1] 作下载/仿真用(下载/仿真接口仅可用 [P3.0, P3.1])，故建议用户将串口 1 放在 P3.6/P3.7 或 P1.6/P1.7 ，若用户不想切换，坚持使用 P3.0/P3.1 工作或作为串口 1 进行通信，则务必在下载程序时，在软件上勾选“下次冷启动时，P3.2/P3.3 为 0/0 时才可以下载程序”。【注 1】

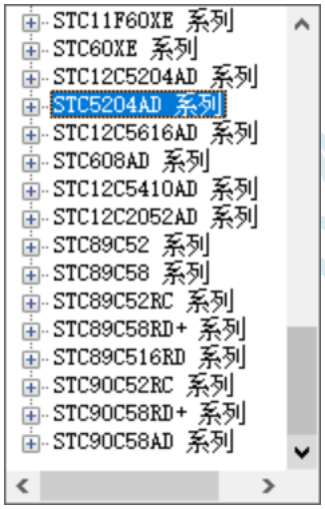
【注 1】：STC15，STC8 系列及以后新出的芯片的烧录保护引脚为 P3.2/P3.3，之前早期芯片的烧录保护引脚为 P1.0/P1.1。

F.3 USB 型联机/脱机下载工具 U8W/U8W-Mini

U8W/U8W-Min 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正面图

U8W反面图

U8W-Mini正反面图

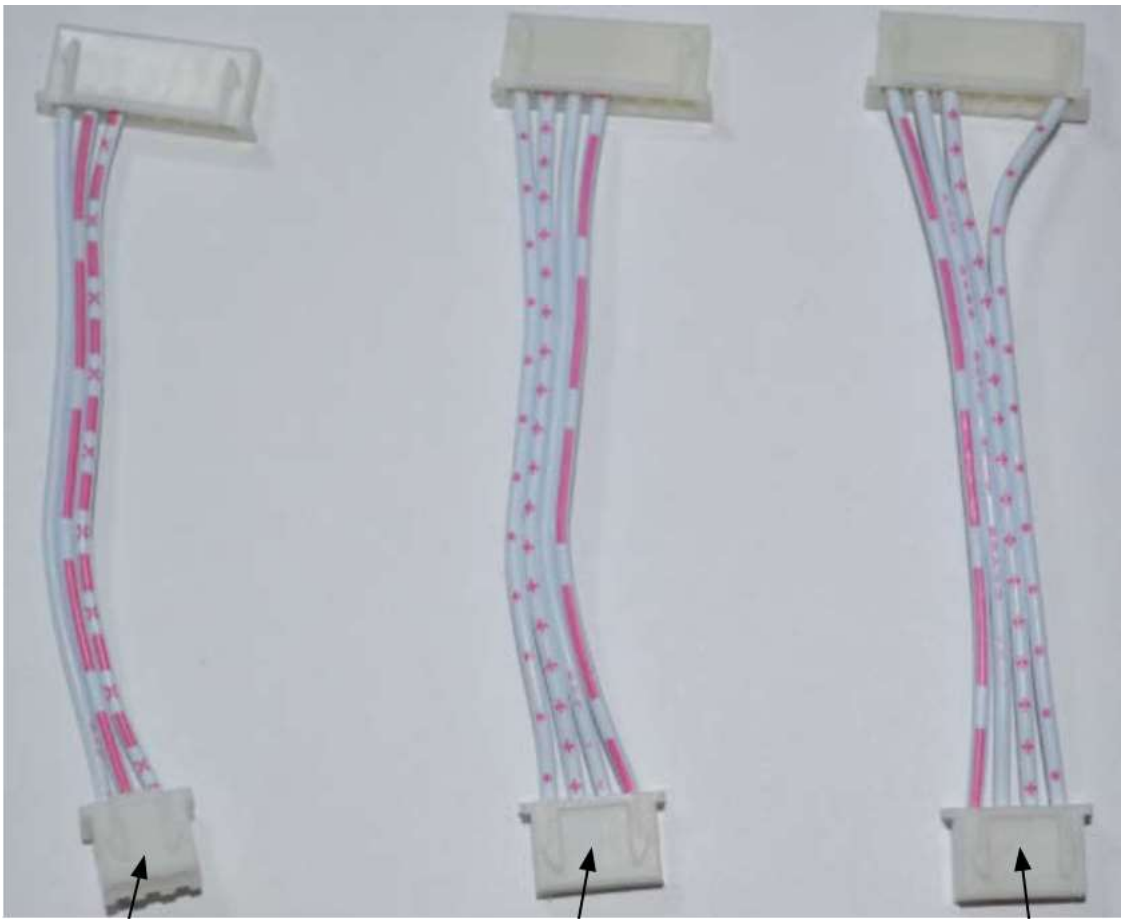
U8W-Mini工具的体积仅有一个U盘大小，其功能与U8W相同，但无锁紧座，价格仅为RMB 50元，欢迎来电订购！

另外还有如下的一些线材与工具相搭配使用，如：

（1）两头公的 USB 连接线(如下图左所示）及 USB-Micro 连接线(如下图右所示)：

注意：此 USB 线为我公司特别定制的 USB 加强线，可确保直接用 USB 供电时能够下载成功。而市面上一些比较劣质的两头公的 USB 线，内阻太大而导致压降很大（如 USB 空载时的电压为 5.0V 左右，当使用劣质的 USB 线连接 U8W/U8W-Mini/U8/U8-Mini，到我们的下载板上的电压可能降到 4.2V 或者更低，从而导致芯片处于复位状态而无法成功下载）。

（2）U8W/U8W-Mini 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：



U8W/U8W-Mini 与用户系统各自独立供电的连接线

U8W/U8W-Mini 给用户系统供电的连接线

用户系统给 U8W/U8W-Mini 供电的连接线

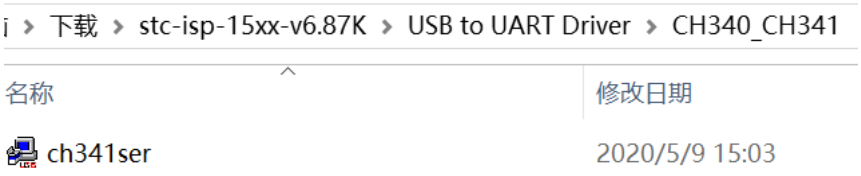
F.3.1 安装 U8W/U8W-Mini 驱动程序

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-isp-15xx-v6.87K\USB to UART Driver\CH340_CH341：

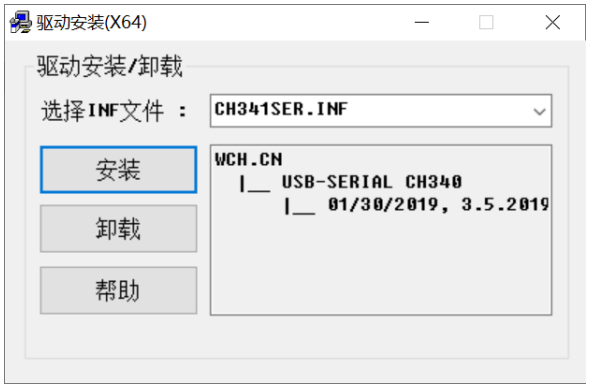


通过 STC 的官方网站或在最新的 STC-ISP 下载软件中手动下载驱动程序

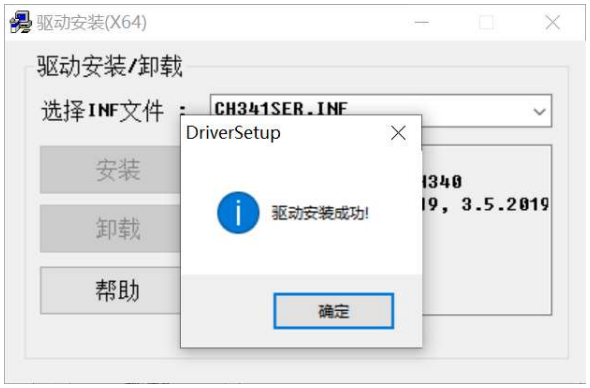
在 STC 的官方网站上或在最新的 STC-ISP 下载软件中手动下载驱动程序，驱动的下载链接为： U8 编程器 USB 转串口驱动 （ http://www.stcmcu.com/STCISP/CH341SER.exe）。网站上及 STC-ISP 下载软件上的驱动地址如下图所示：

安装 U8W/U8W-Mini 的驱动程序

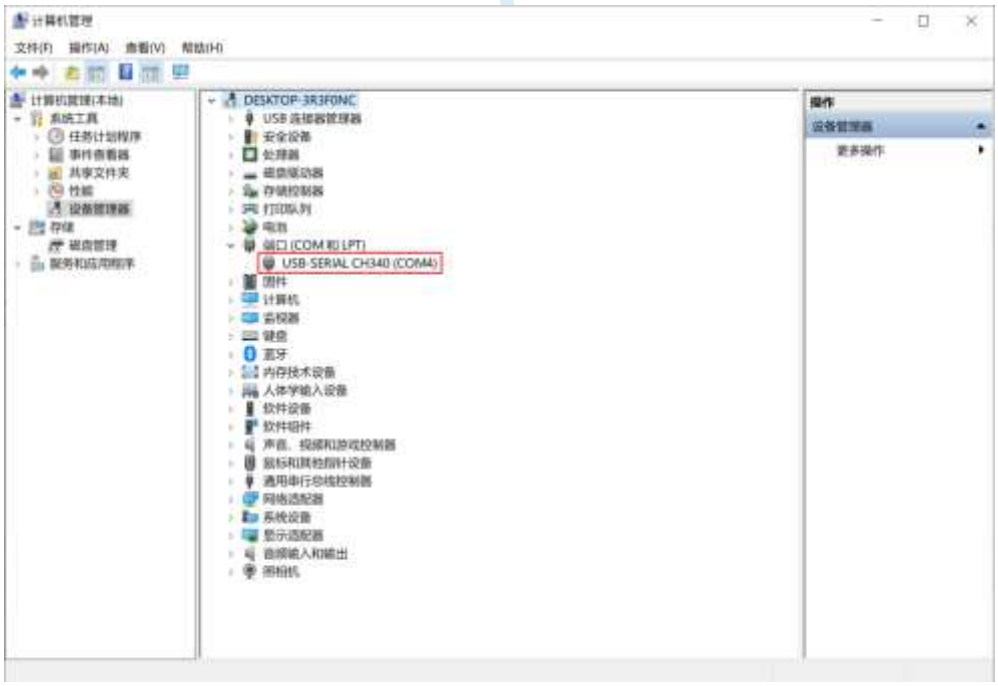
驱动程序下载到本机后，直接双击可执行程序并运行，出现下图所示的界面，点击“安装”按钮开始自动安装驱动：



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）：



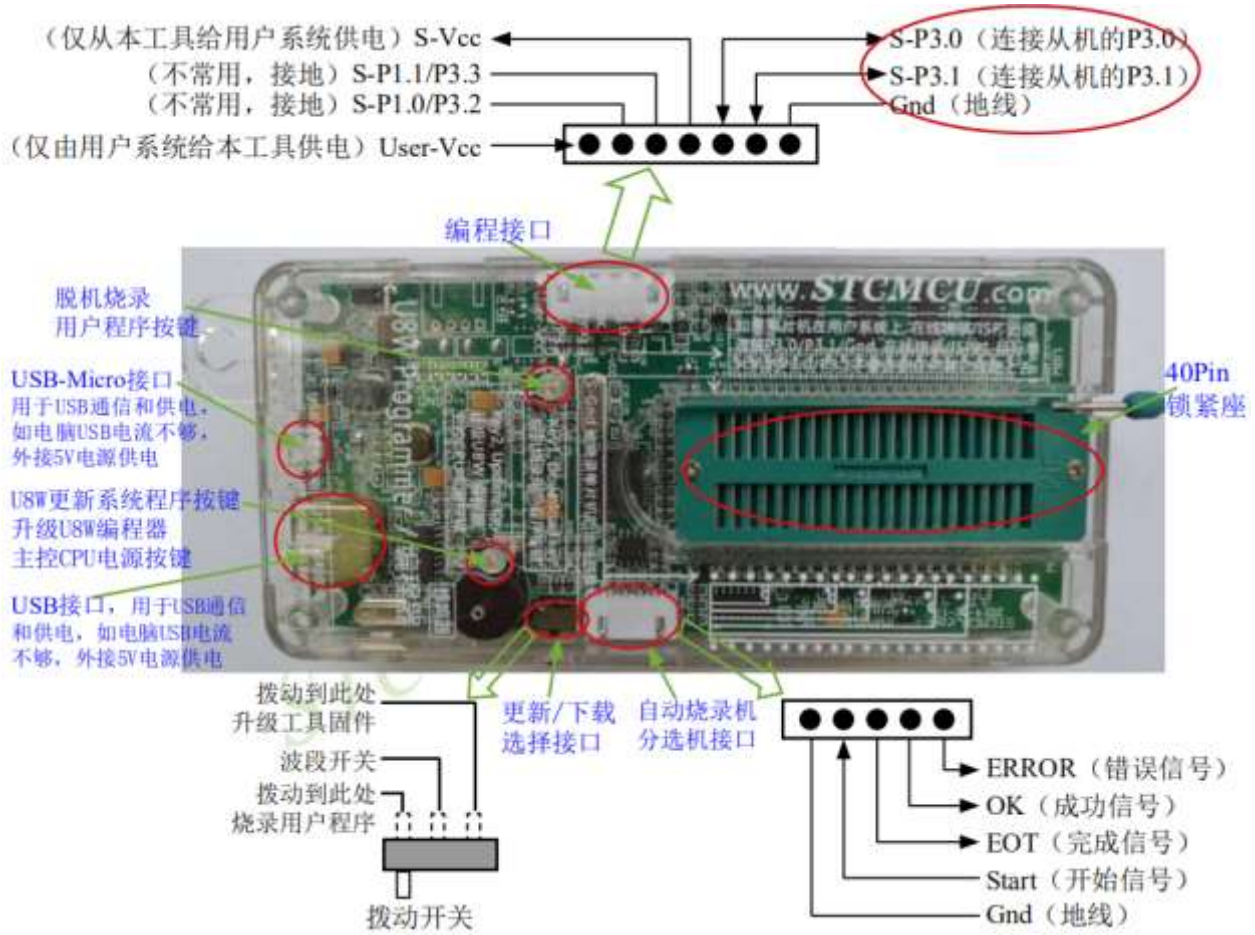
注意：在后面使用 STC-ISP 下载软件时，选择的串口号必须选择与此相对应的串口号，如下图所示：

F.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

STC MCU



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。

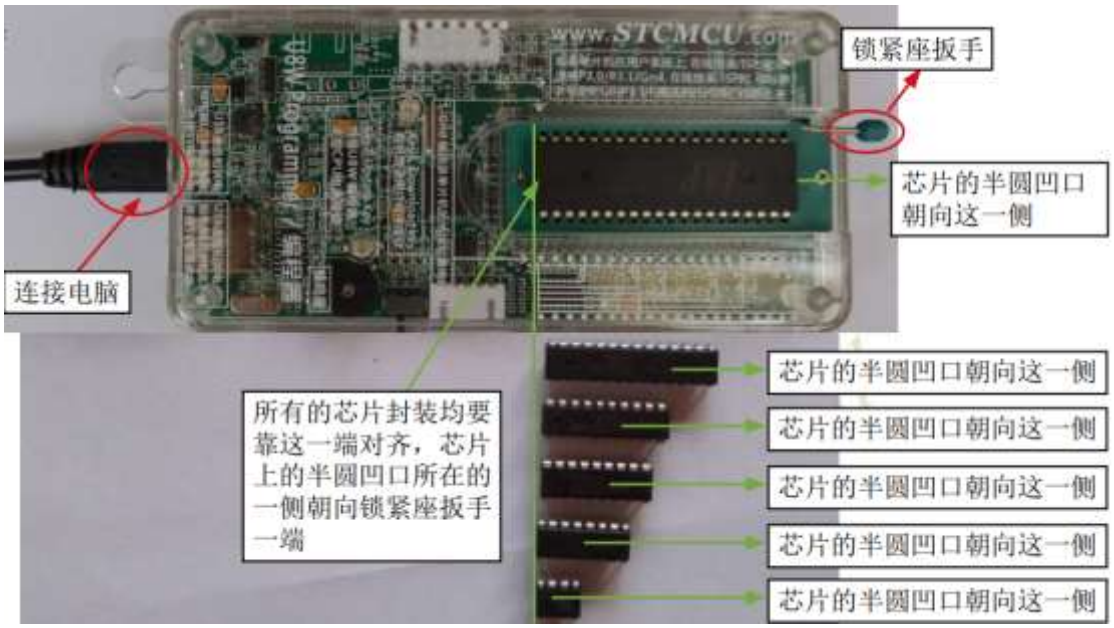
（拨动开关连接方式请参考上图）

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。

F.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：

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；

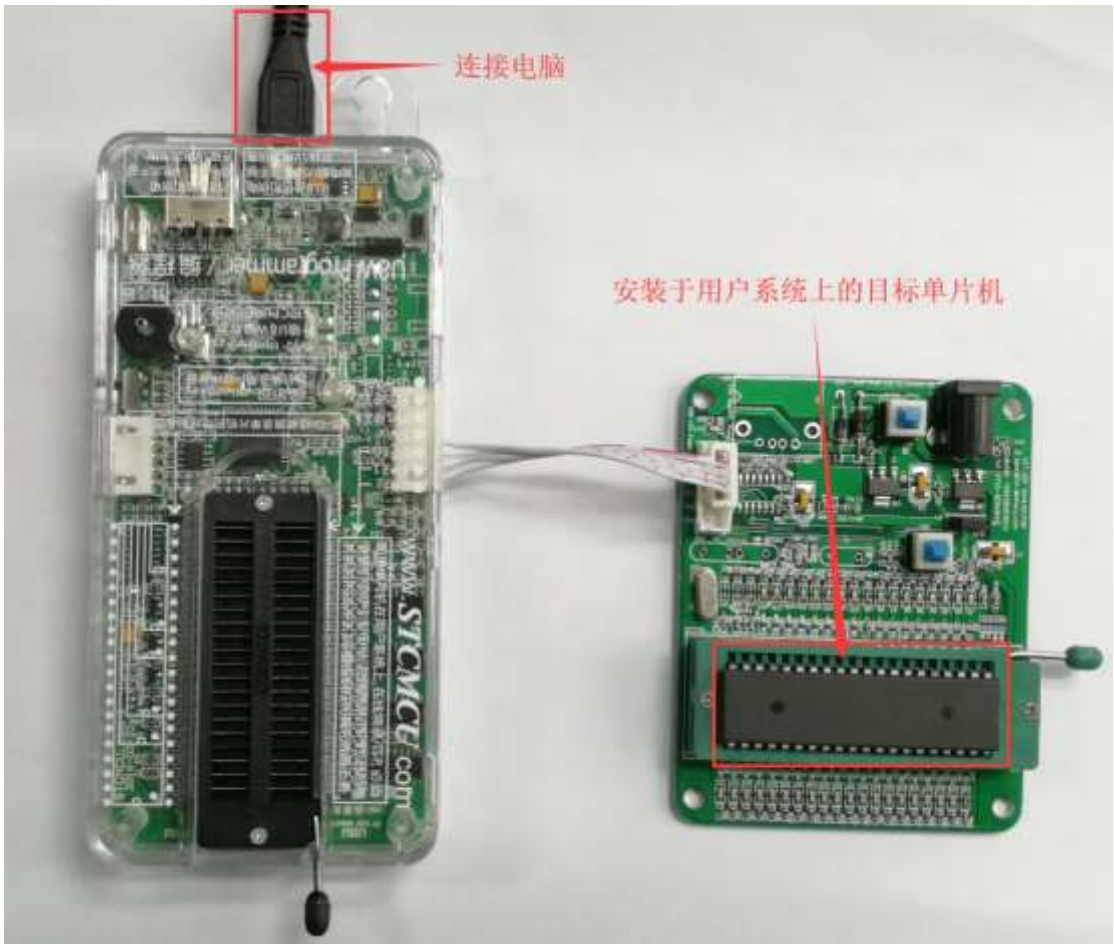
7显示烧录过程的步骤信息，烧录完成提示“操作成功！”。

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 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）。

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目标单片机相连接，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）。

F.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：

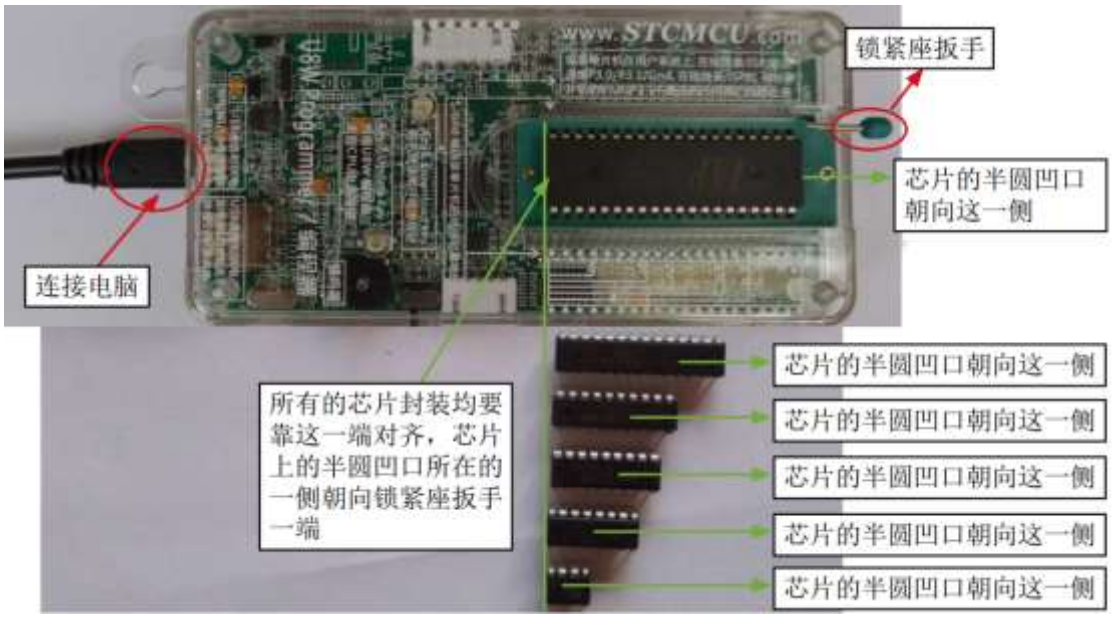
（2）在 STC-ISP 下载软件中按如下图所示的步骤进行设置：

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。

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：

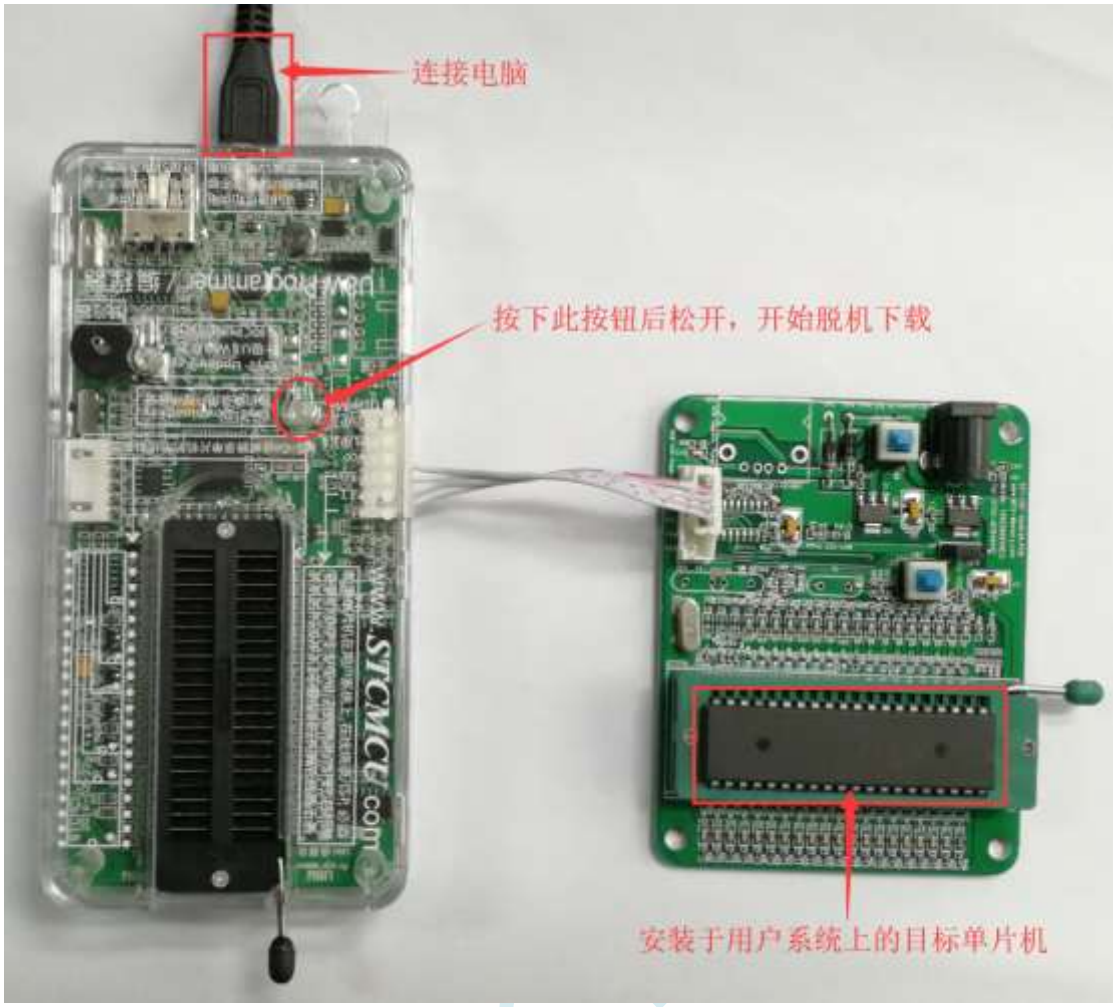
（2）在 STC-ISP 下载软件中按如下图所示的步骤进行设置：

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。

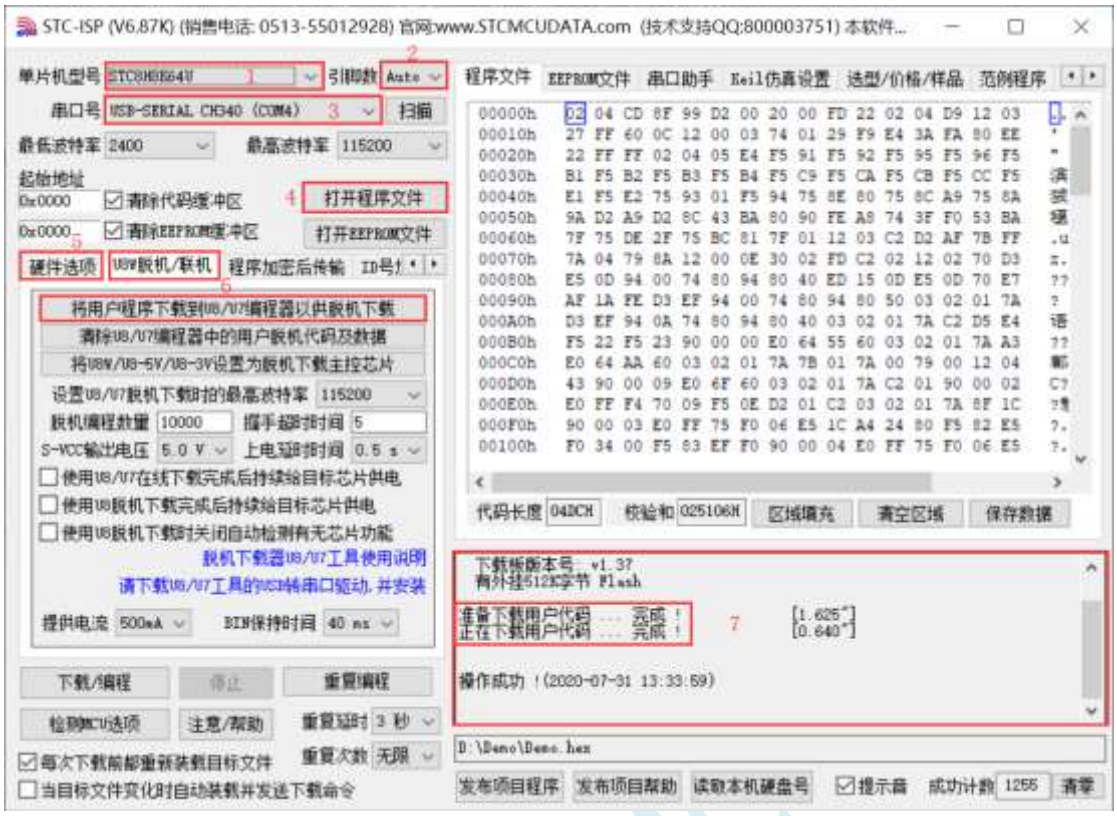
目标芯片由用户系统引线连接 U8W 并通过用户系统给 U8W 供电进行脱机下载

（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

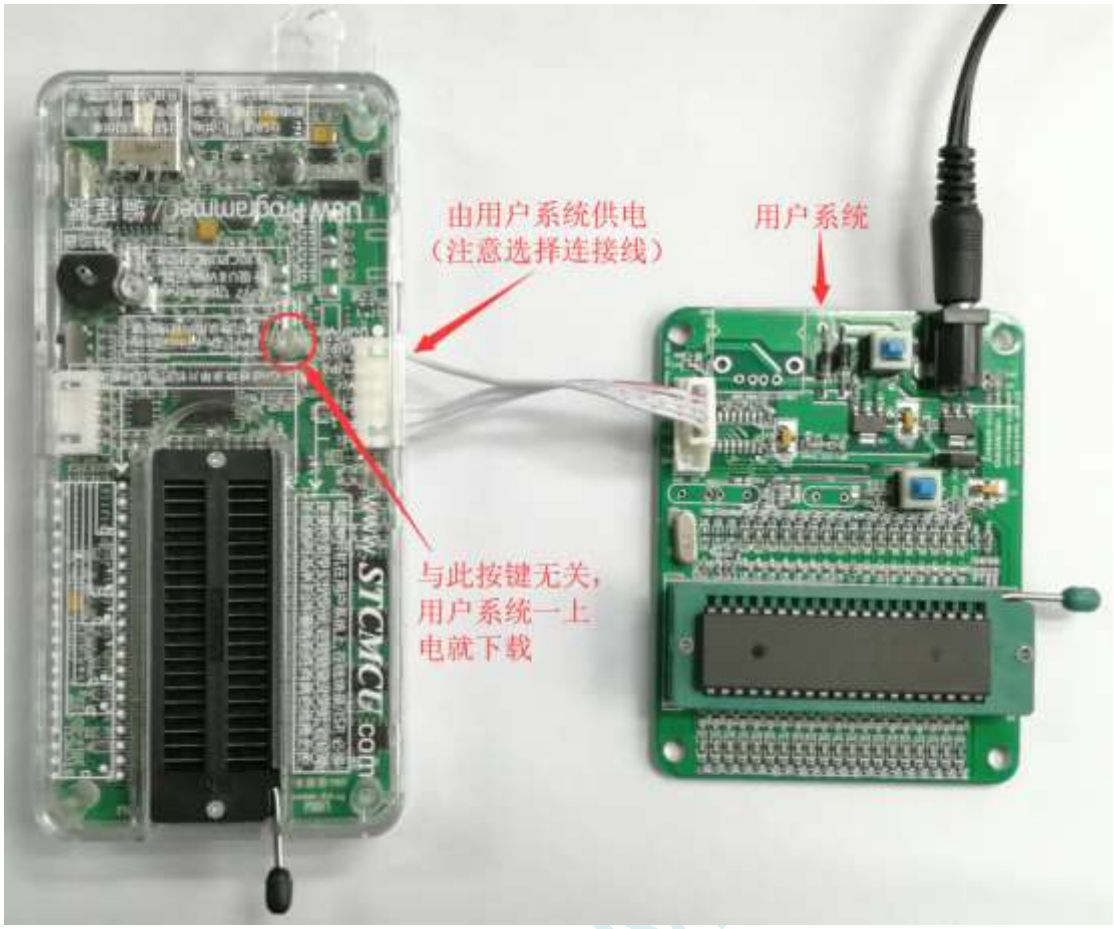
软件使用）。



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。

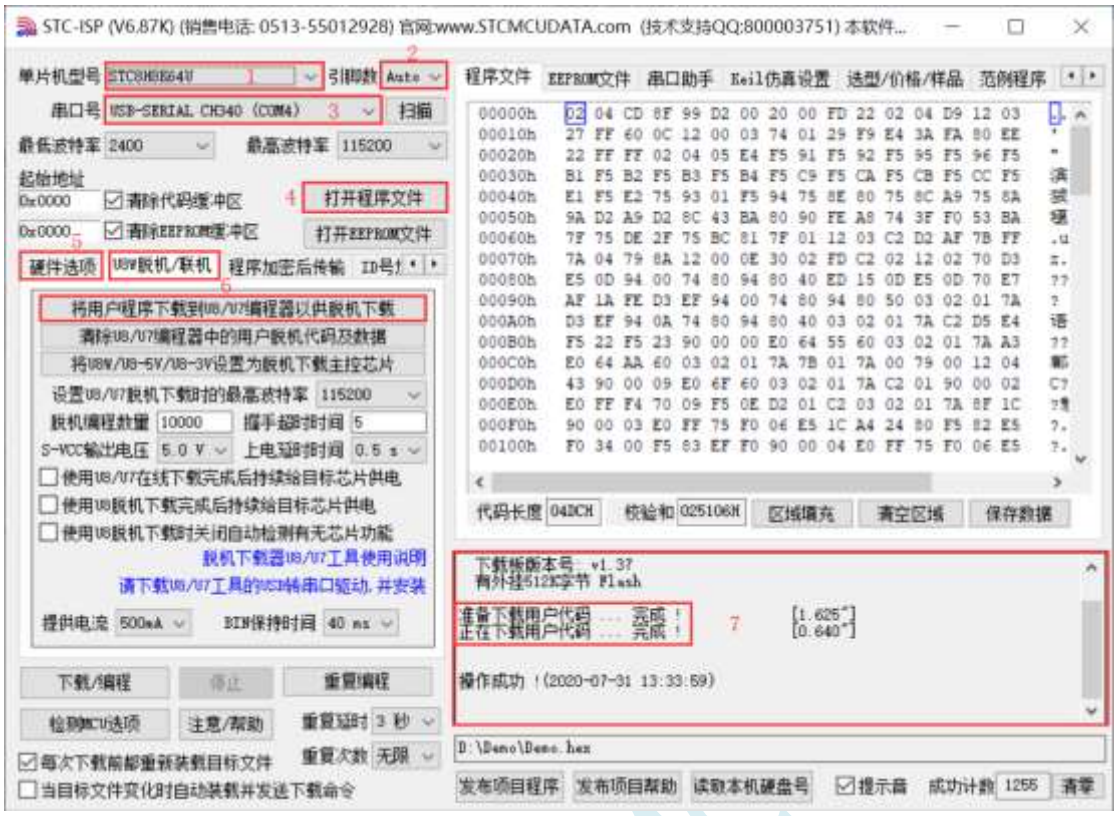
目标芯片由用户系统引线连接 U8W 且 U8W 与用户系统各自独立供电进行脱机下载

（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

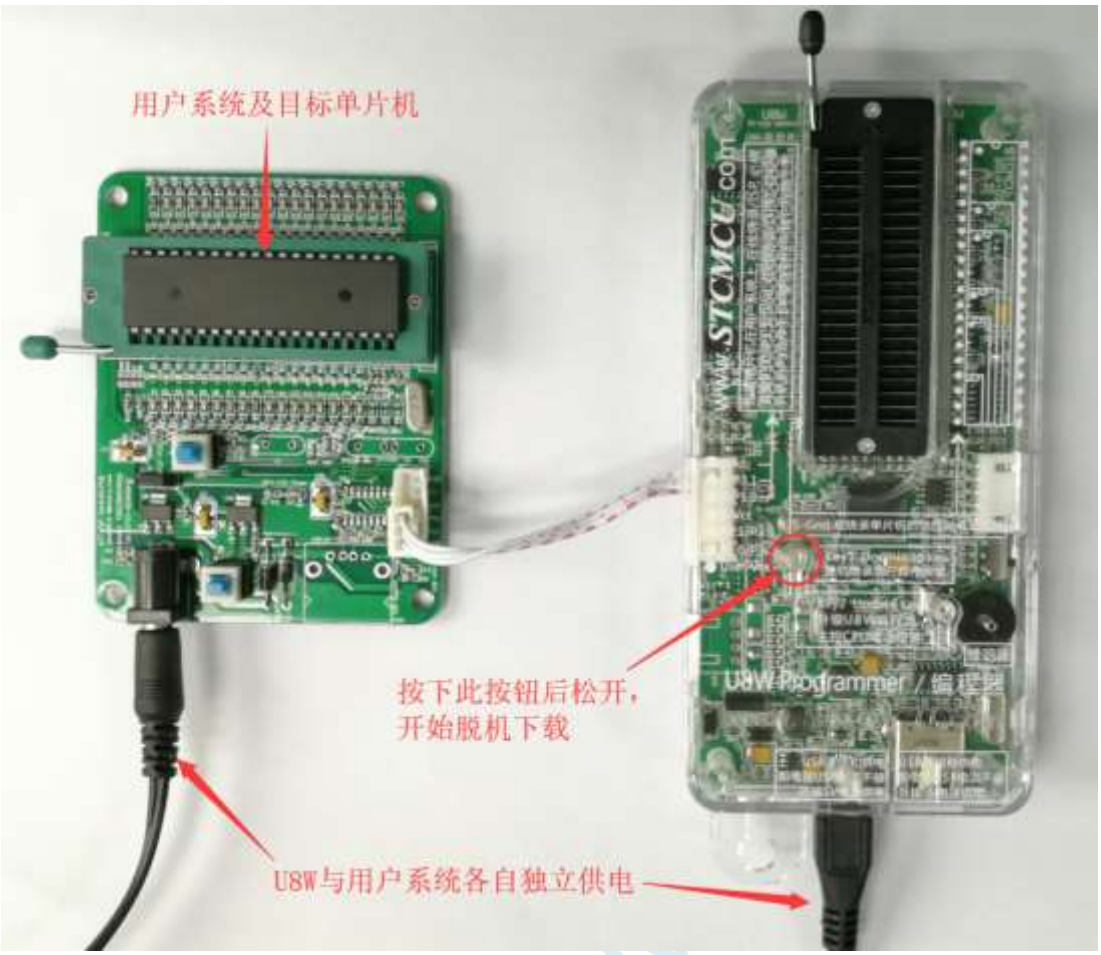
软件使用)。



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。

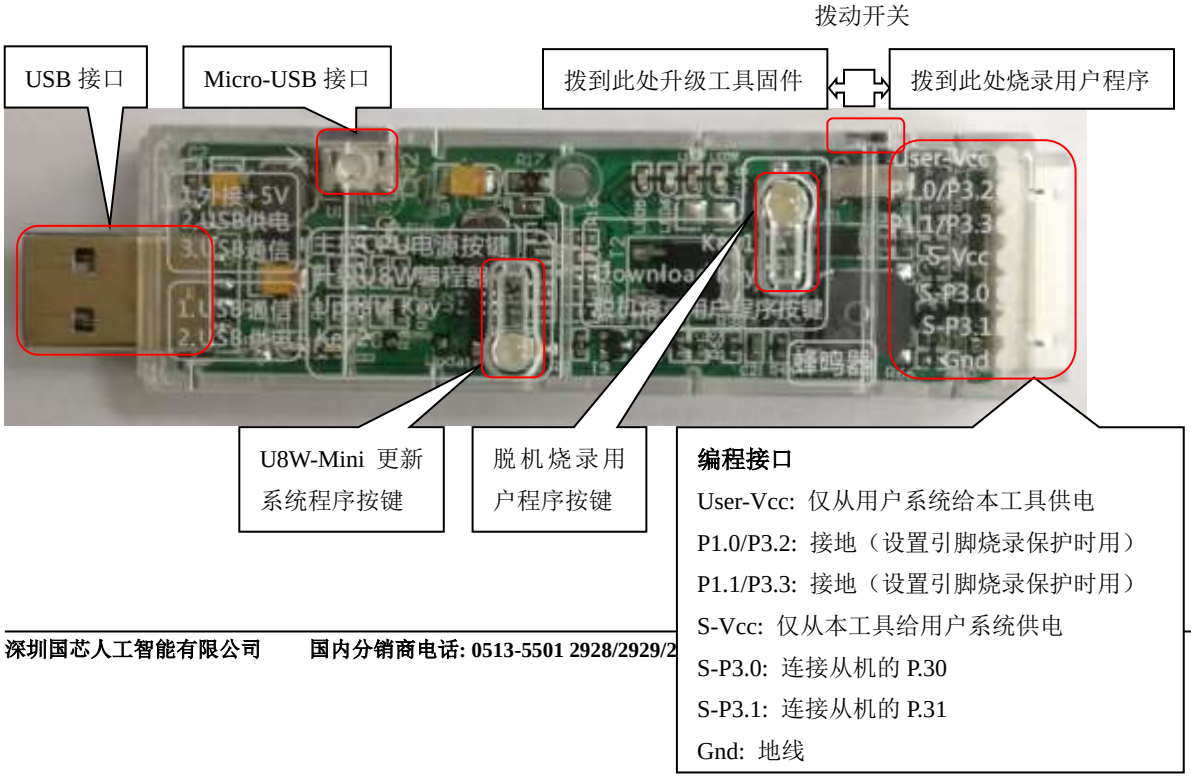
(3)然后按下图所示的方式连接 U8W 与用户系统，并将图中所示按钮先按下后松开，准备开始脱机下载，最后给用户系统上电/开电源，下载用户程序正式开始：



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。

F.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：

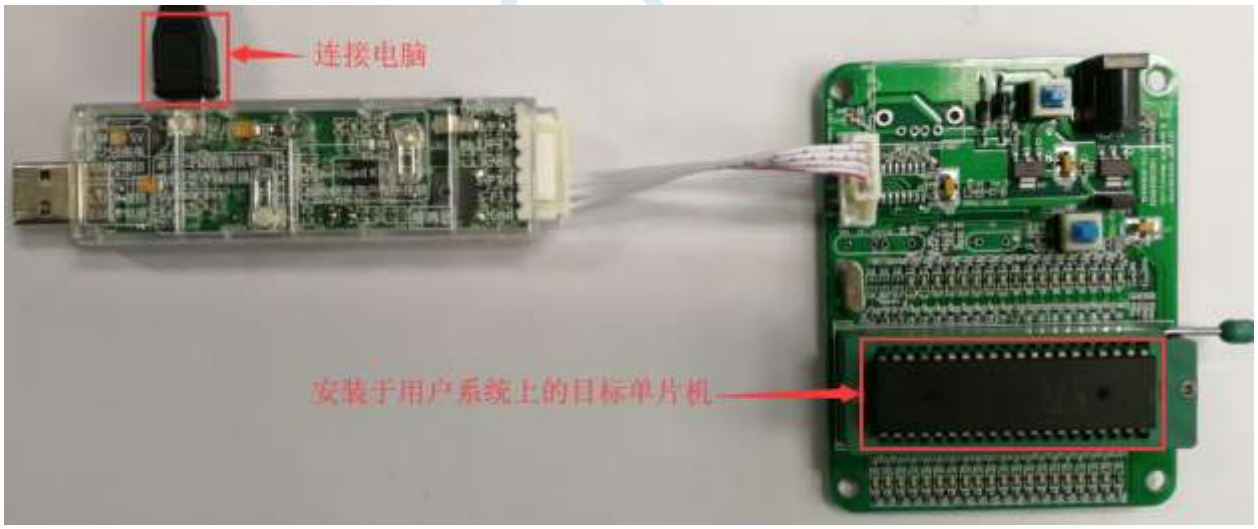


编程接口：根据不同的供电方式，使用不同的下载连接线连接 U8W-Mini 下载板和用户系统。
U8W-Mini 更新系统程序按键：用于更新 U8W-Mini 工具，当有新版本的 U8W 固件时，需要按下此按键对 U8W-Mini 的主控芯片进行更新（注意：必须先将更新/下载选择接口上的拨动开关拨动到升级工具固件）。
脱机下载用户程序按钮：开始脱机下载按钮。首先 PC 将脱机代码下载到 U8W-Mini 上，然后使用下载连接线将用户系统连接到 U8W-Mini，再按下此按钮即可开始脱机下载（每次上电时也会立即开始下载用户代码）。
更新/下载选择接口：当需要对 U8W-Mini 的底层固件进行升级时，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。

F.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目标单片机相连接，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）。

F.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：

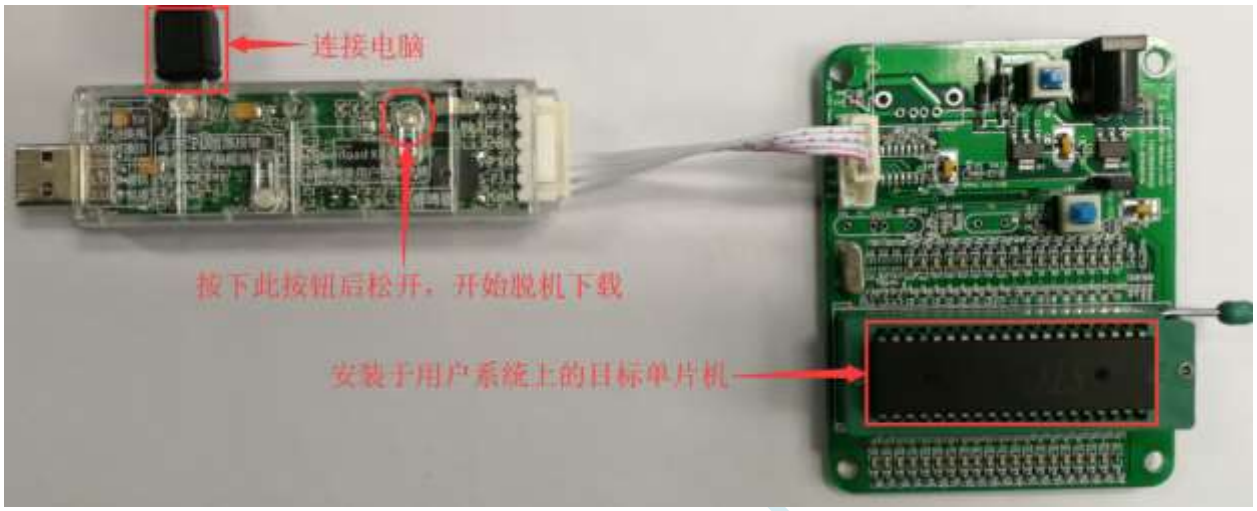
（2）在 STC-ISP 下载软件中按如下图所示的步骤进行设置：

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：

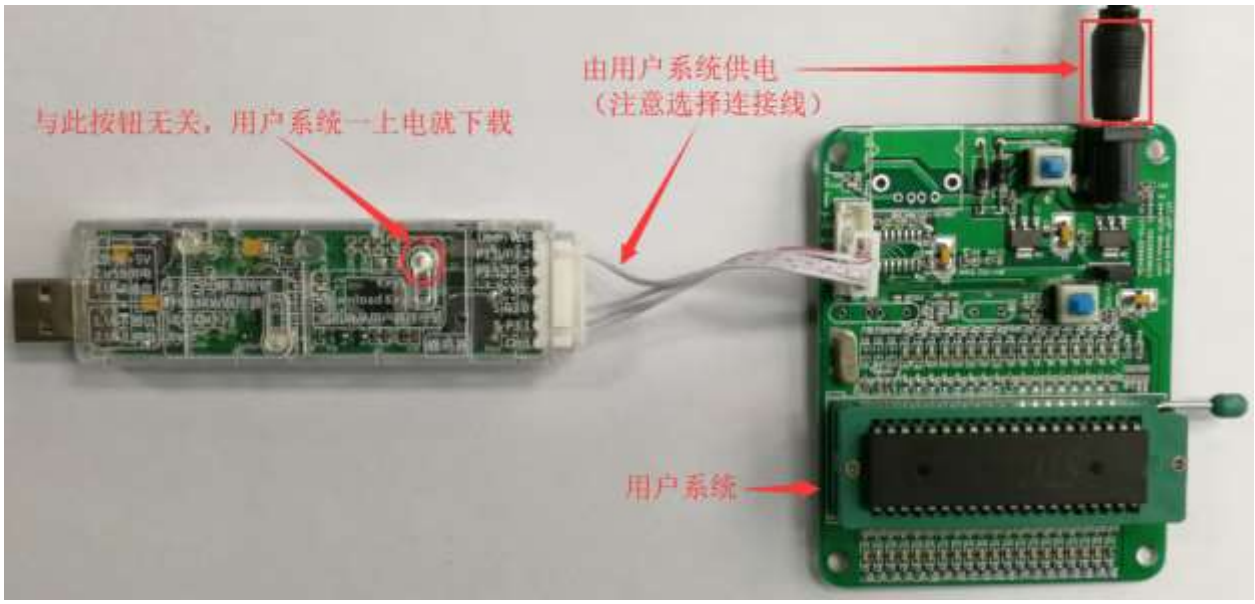
（2）在 STC-ISP 下载软件中按如下图所示的步骤进行设置：

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：

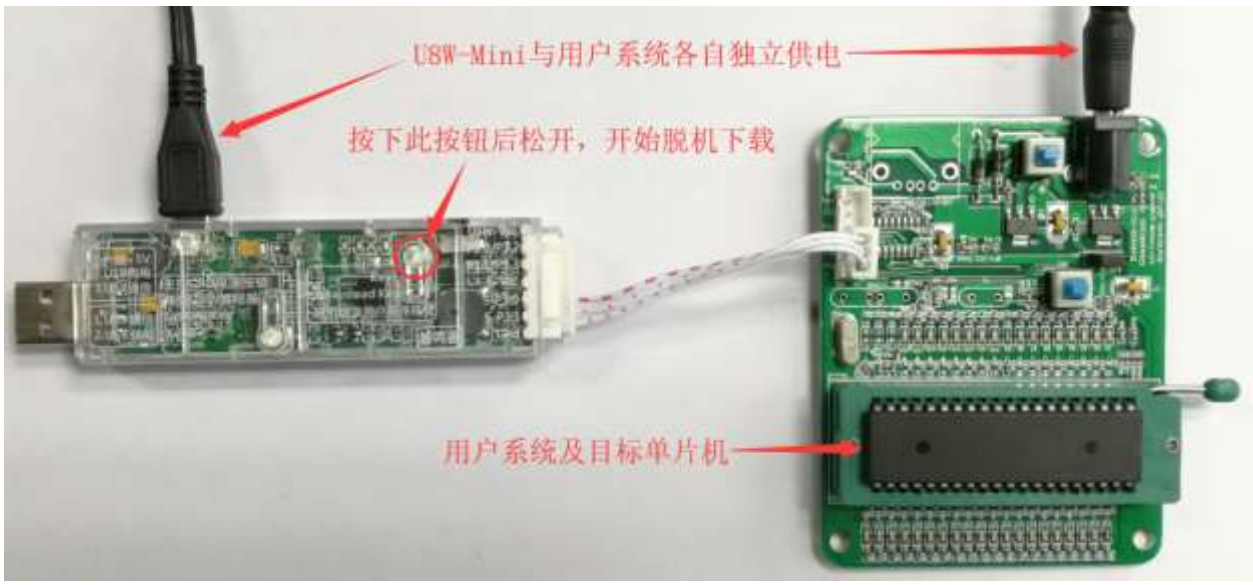
（2）在 STC-ISP 下载软件中按如下图所示的步骤进行设置：

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）。

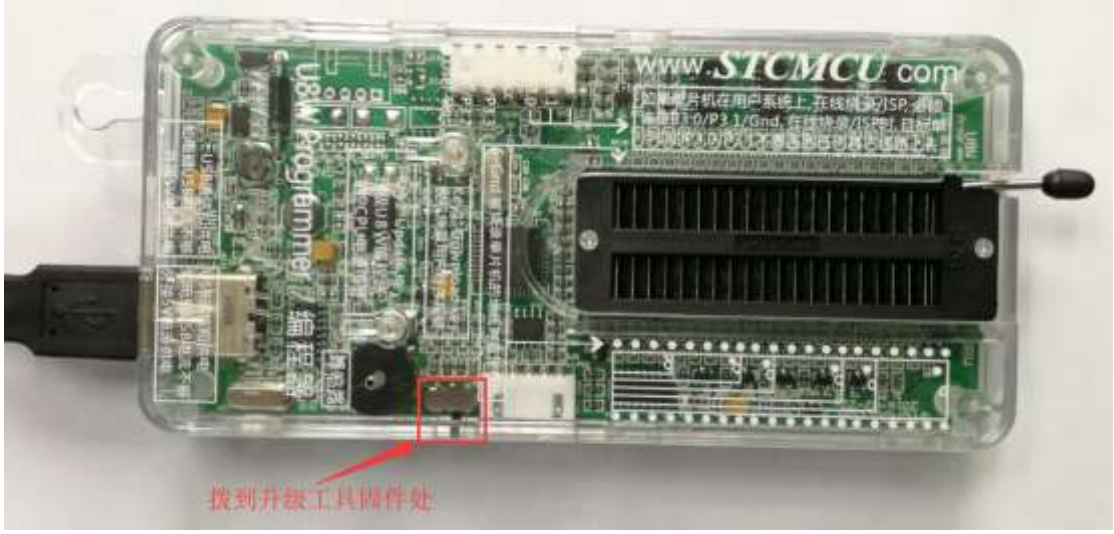
（3）然后按下图所示的方式连接 U8W-Mini 与用户系统，并将图中所示按钮先按下后松开，准备开始脱机下载，最后给用户系统上电/开电源，下载用户程序正式开始：



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。

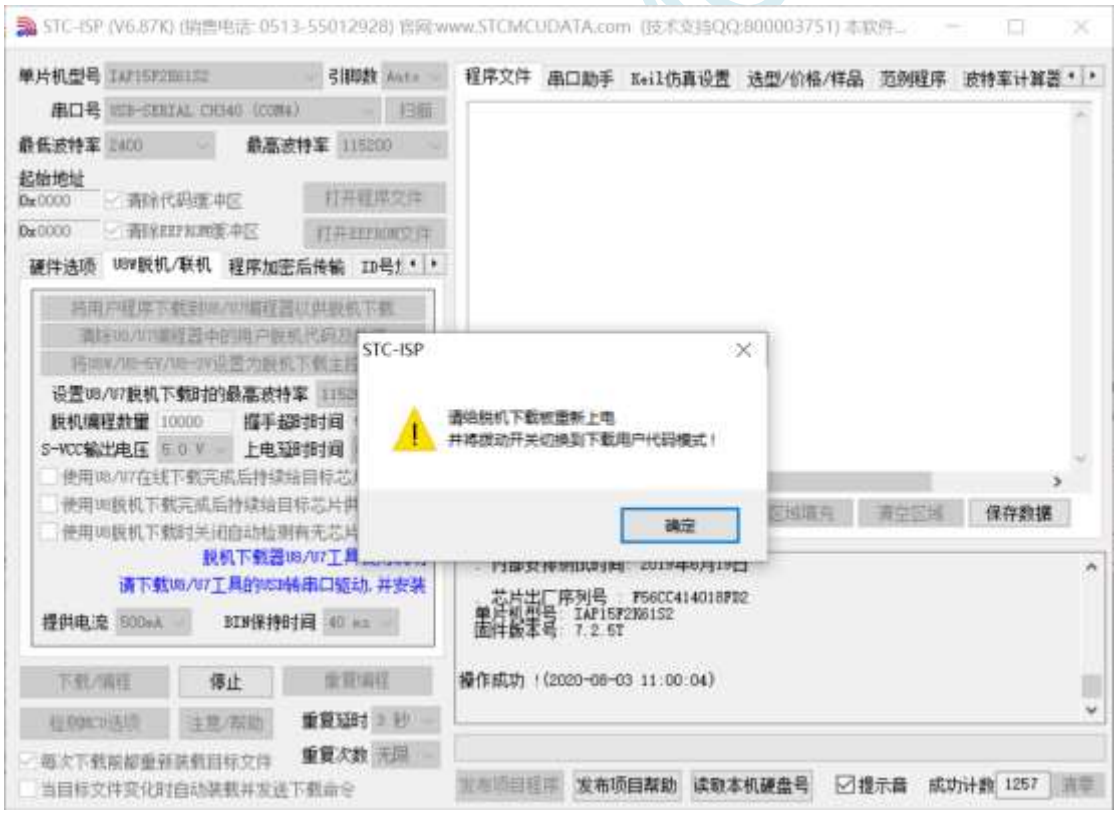
F.3.8 制作/更新 U8W/U8W-Mini

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：

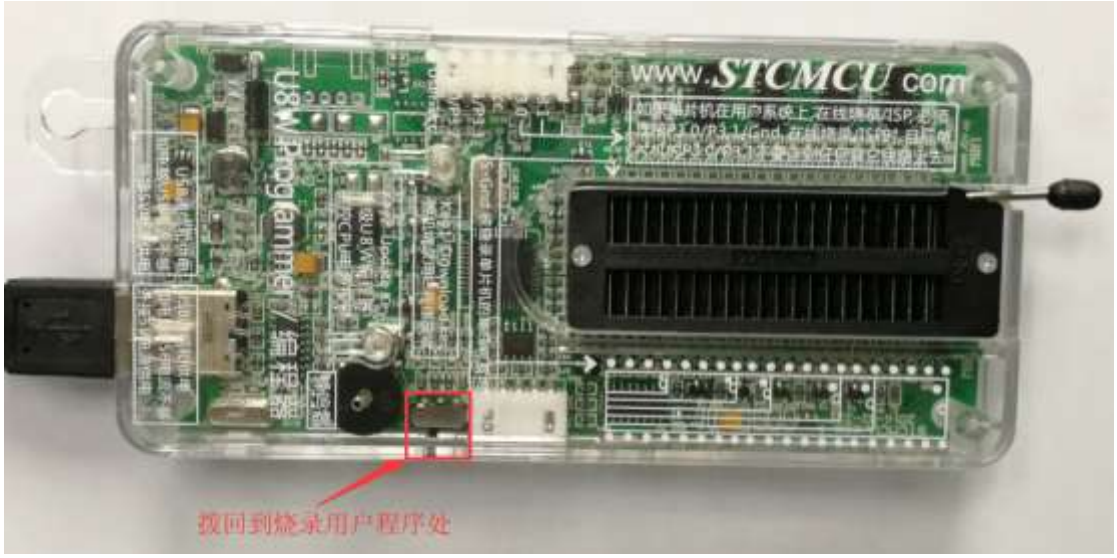


然后在 STC-ISP 下载程序中的“U8W 脱机/联机”页面中点击“将 U8W/U8-5V/U8-3V 设置为脱机下载主控芯片”按钮，如下图：（注意：一定要选择 U8W 所对应的串口）

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）



F.3.9 U8W/U8W-Mini 设置直通模式（可用于仿真）

若要使用 U8W/U8-Mini 进行仿真，首先必须将 U8W/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。

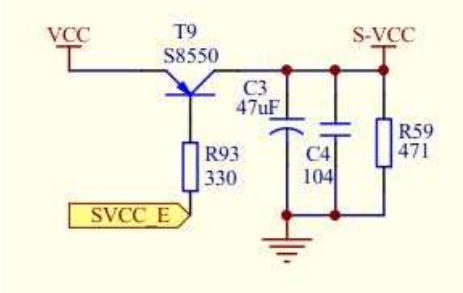
F.3.10 U8W/U8W-Mini 的参考电路

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：

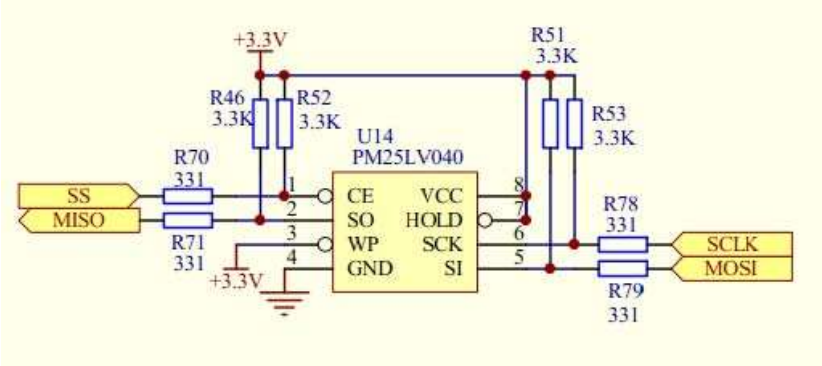
脚位功能	端口	功能描述
电源控制脚	P2.6	低位有效
下载通讯脚	P1.0	串口 RXD，连接目标芯片的 TXD（P3.1）
	P1.1	串口 TXD，连接目标芯片的 RXD（P3.0）
编程按键	P3.6	低有效
显示	P3.2	LED1
	P3.3	LED2
	P3.4	LED3
	P5.5	LED4
外挂串行 Flash 控制脚	P2.4	Flash 的 CE 脚
	P2.2	Flash 的 SO 脚
	P2.3	Flash 的 SI 脚
	P2.1	Flash 的 SCLK 脚

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）
蜂鸣器（BEEP）控制	P2.5	高有效（高电平发出声音）

电源控制部分参考电路图：

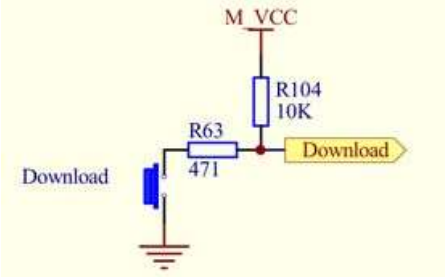


Flash 控制部分参考电路图：

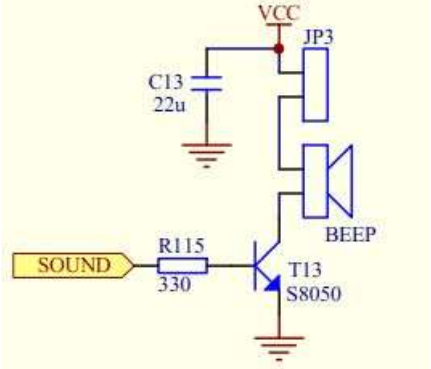


用户程序大于 41K 时需要此 Flash 存储器

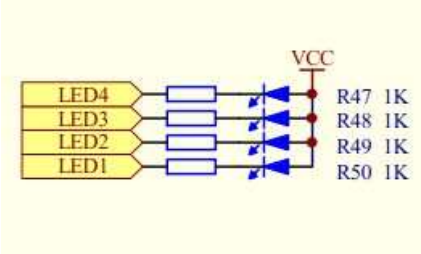
按键部分参考电路图：



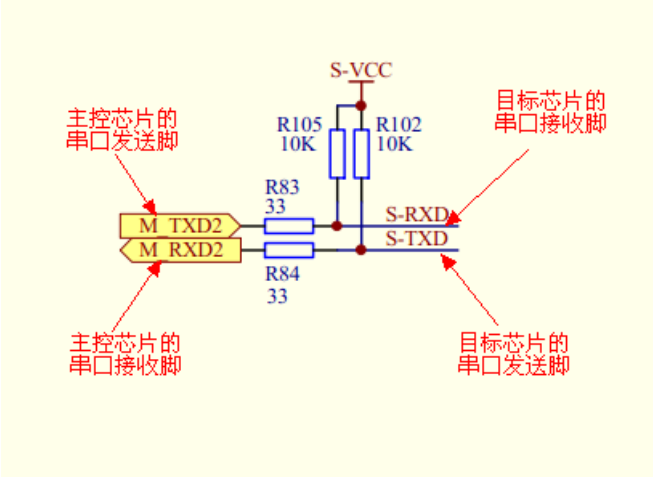
蜂鸣器部分参考电路图：



LED 显示部分参考电路图：



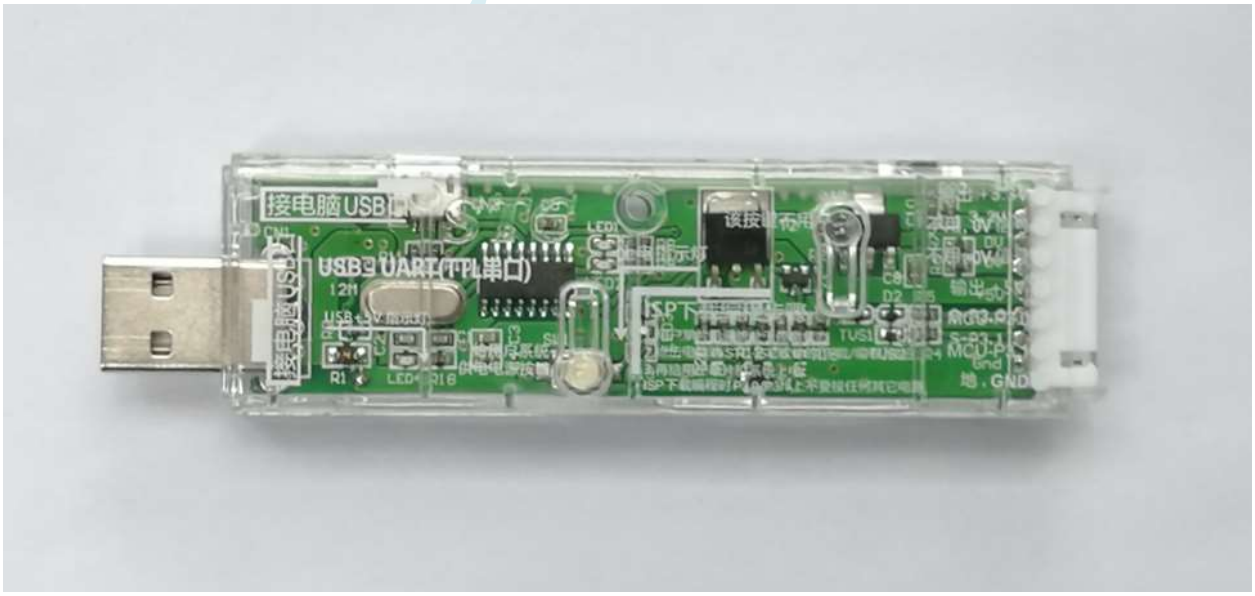
串口通讯脚连接部分参考电路图：



F.4 STC 通用 USB 转串口工具

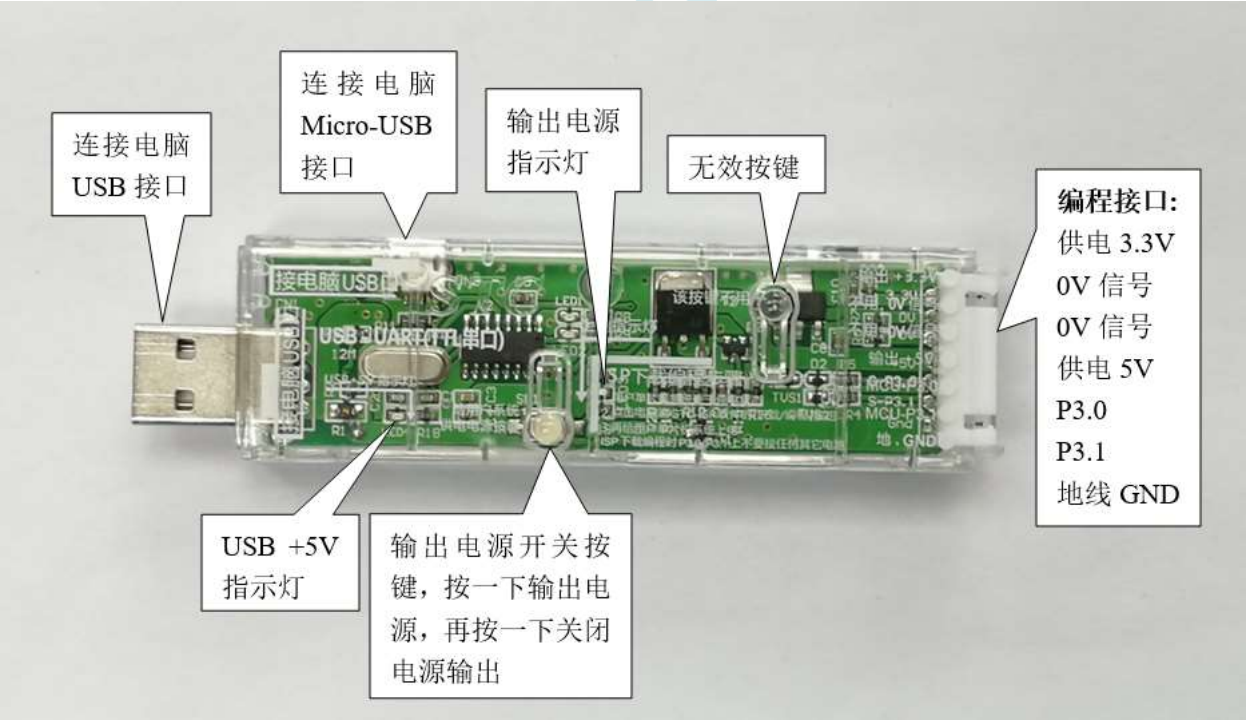
F.4.1 STC 通用 USB 转串口工具外观图

正面：



背面：

F.4.2 STC 通用 USB 转串口工具布局图



在此，需要对“电源开关”进行说明：

此按钮的作用与自锁开关相同，在开关按钮第一次按时，开关接通电源并保持，即自锁，在开关按钮第二次按时，开关断开电源。鉴于自锁开关使用过程中容易损坏的特点，我们设计了一套利用轻触开关替代自锁开关功能的电路，提高工具的使用寿命。

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

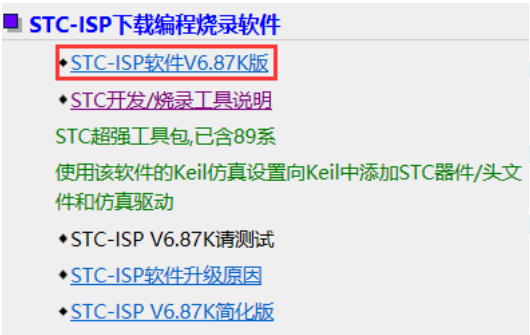
程序，所以使用 STC 通用 USB 转串口工具下载程序到 MCU 的正确步骤为：

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】

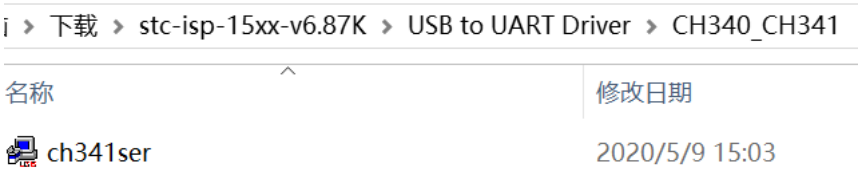
此外，USB 接口与 Micro-USB 接口是相同的功能，用户根据需要连接其中一个接口到电脑即可。
编程接口的 0V 信号脚内部有 470 欧姆电阻接地，如果设置了 P1.0/P1.1=0/0 或者 P3.2/P3.3=0/0 时才能下载，可将 P1.0，P1.1 或者 P3.2，P3.3 接到 0V 信号脚。

F.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：

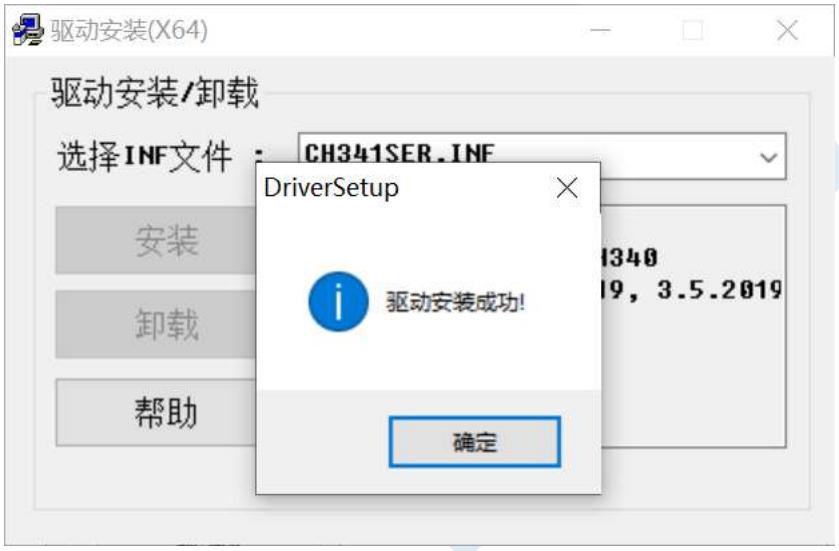


下载后进行解压，CH340 的驱动安装包路径 stc-isp-15xx-v6.87K\USB to UART Driver\CH340_CH341：



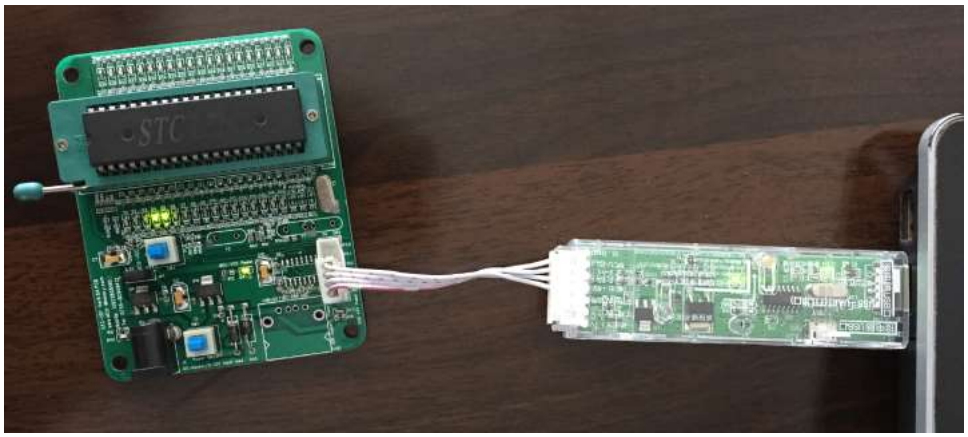
以 STC 官网提供的 CH341SER 串口驱动为例，双击“CH341SER.exe”安装包，在弹出的主界面点击“安装”按钮开始安装驱动程序：

然后弹出驱动安装成功对话框，点击“确定”按钮完成安装：

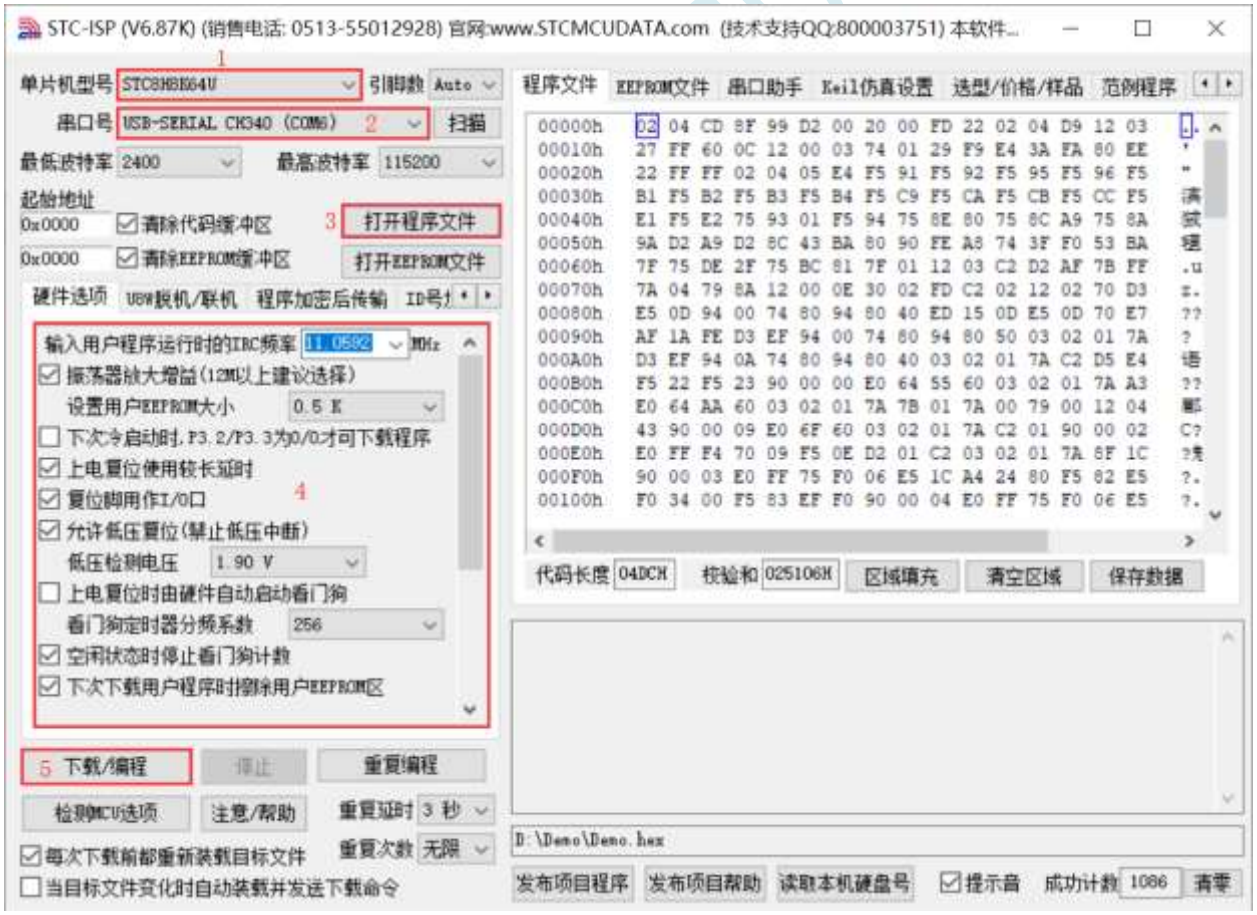


F.4.4 使用 STC 通用 USB 转串口工具下载程序到 MCU

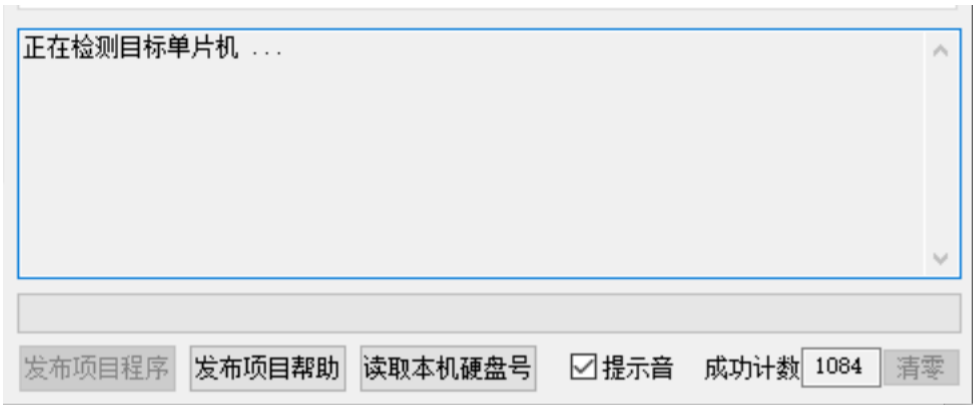
1. 使用 STC 通用 USB 转串口工具将待烧录 MCU 与电脑进行连接：



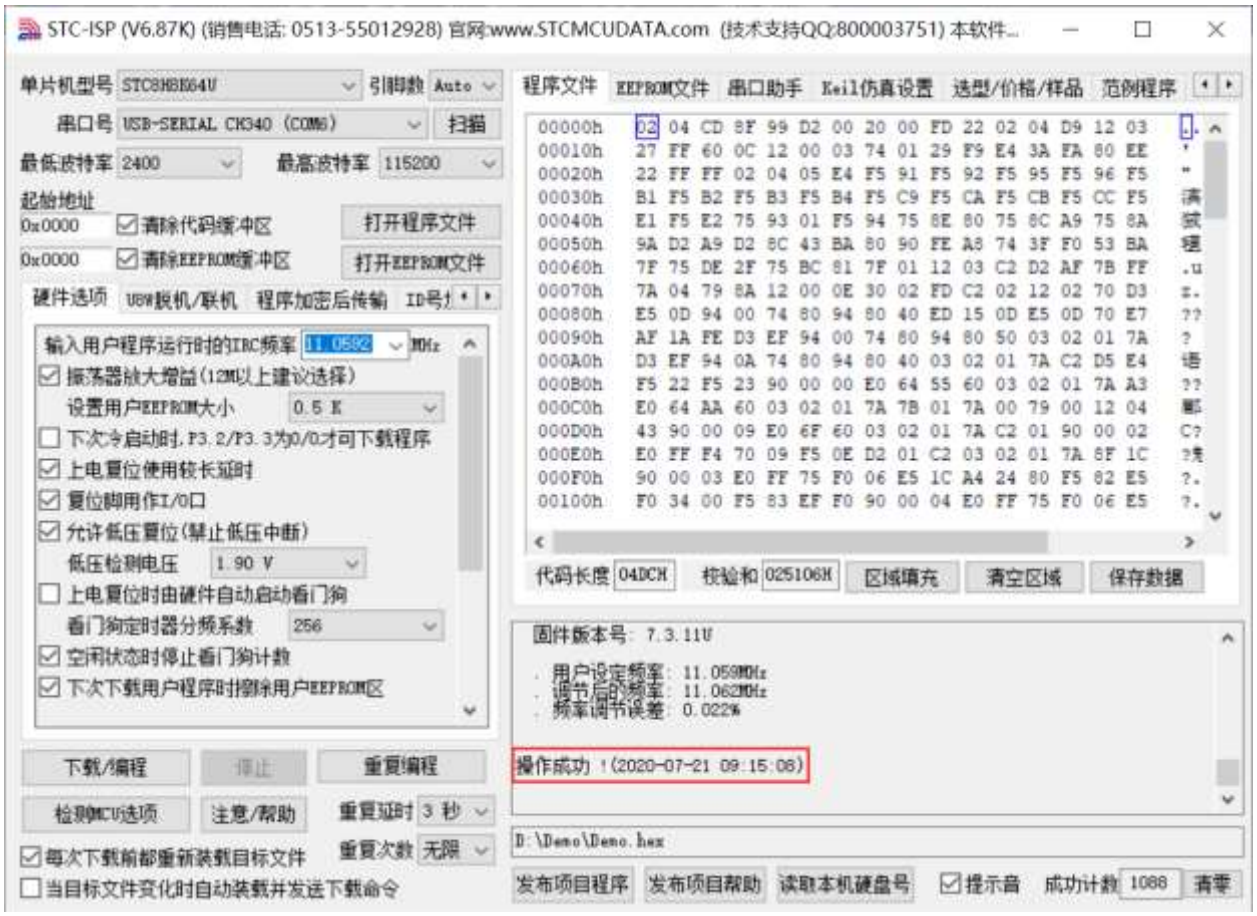
2. 打开 STC-ISP 软件；
3. 选择烧录芯片对应的型号；
4. 选择 STC 通用 USB 转串口工具所识别的串口号（当 STC 通用 USB 转串口工具与电脑正确连接后，软件会自动扫描并识别名称为“USB-SERIAL CH340 (COMx)”串口，具体的 COM 编号会因电脑不同而不同）。当有多个 USB 转串口线与电脑相连时，则必须手动选择；
5. 加载烧录程序；
6. 设置烧录选项；
7. 点击“下载/编程”按钮；



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!”。



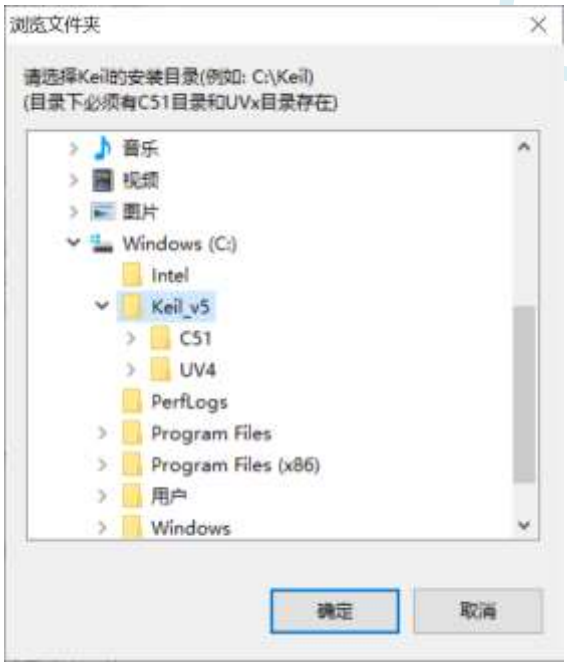
F.4.5 使用 STC 通用 USB 转串口工具仿真用户代码

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：

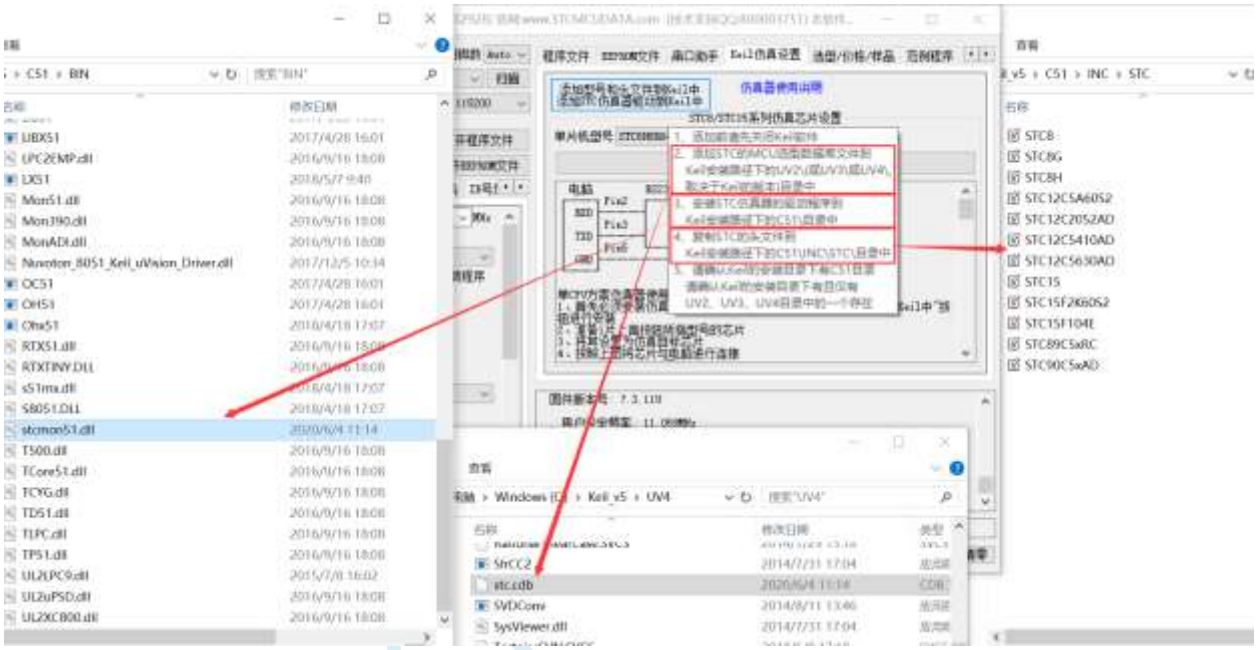
按下后会出现如下画面：



将目录定位到 Keil 软件的安装目录，然后确定。

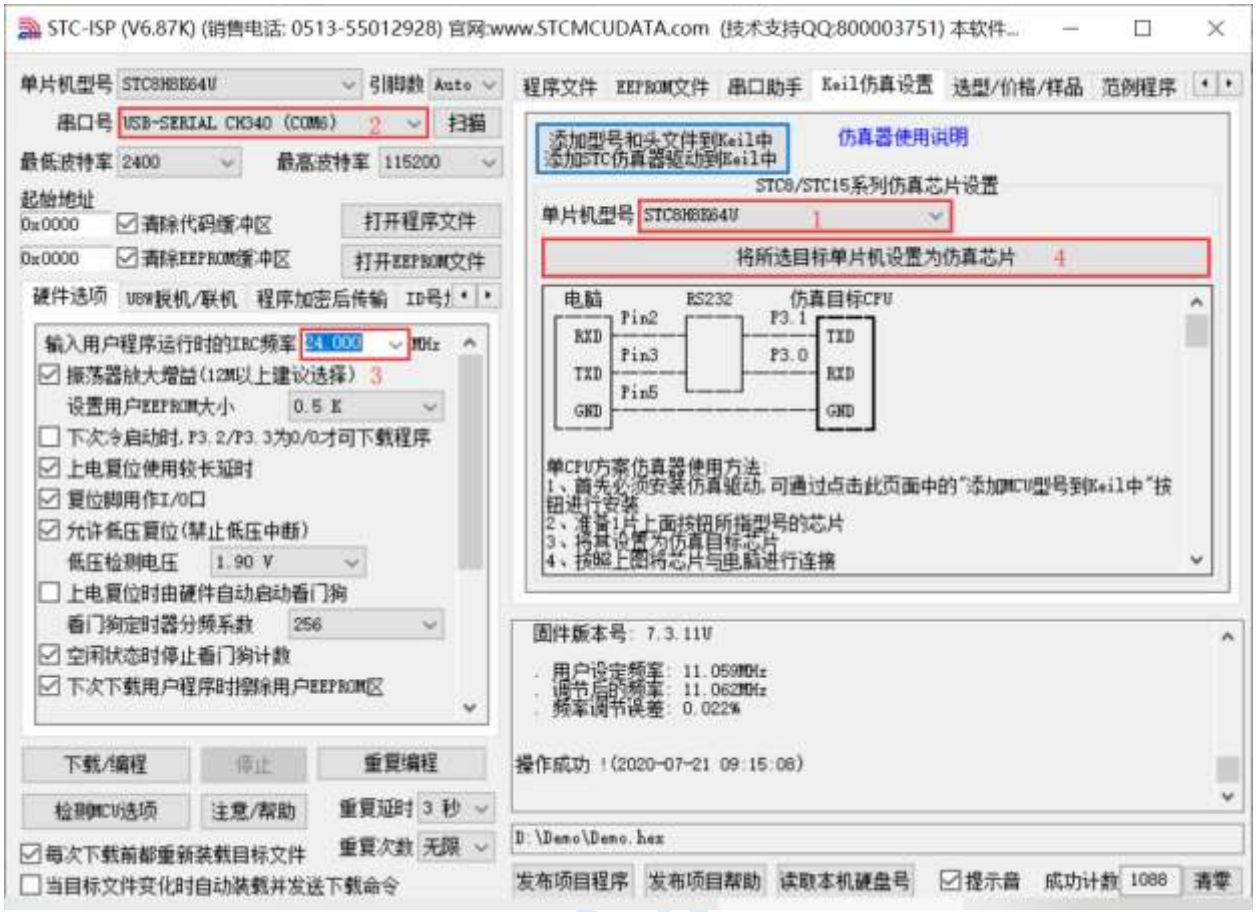
安装成功后会弹出如下的提示框：

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：

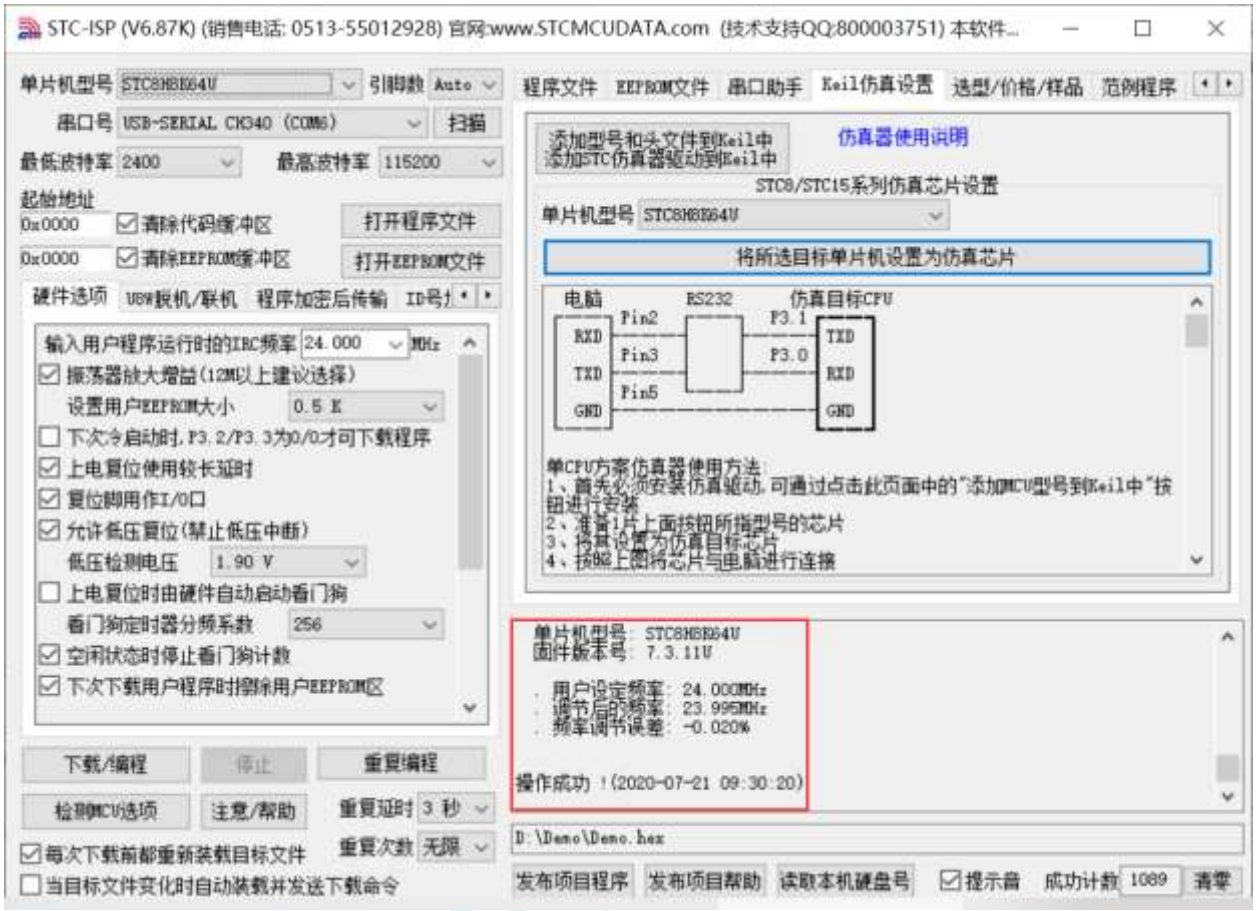
首先使用 STC 通用 USB 转串口工具将 MCU 与电脑进行连接；
然后打开 STC 的 ISP 下载软件，并在串口号的下拉列表中选择串口工具所对应的串口号；
选择 MCU 单片机型号；
选择用户程序运行的 IRC 频率，制作仿真芯片时选择的频率与所仿真的用户程序所设置的频率一致，才能达到真实的运行效果。



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：

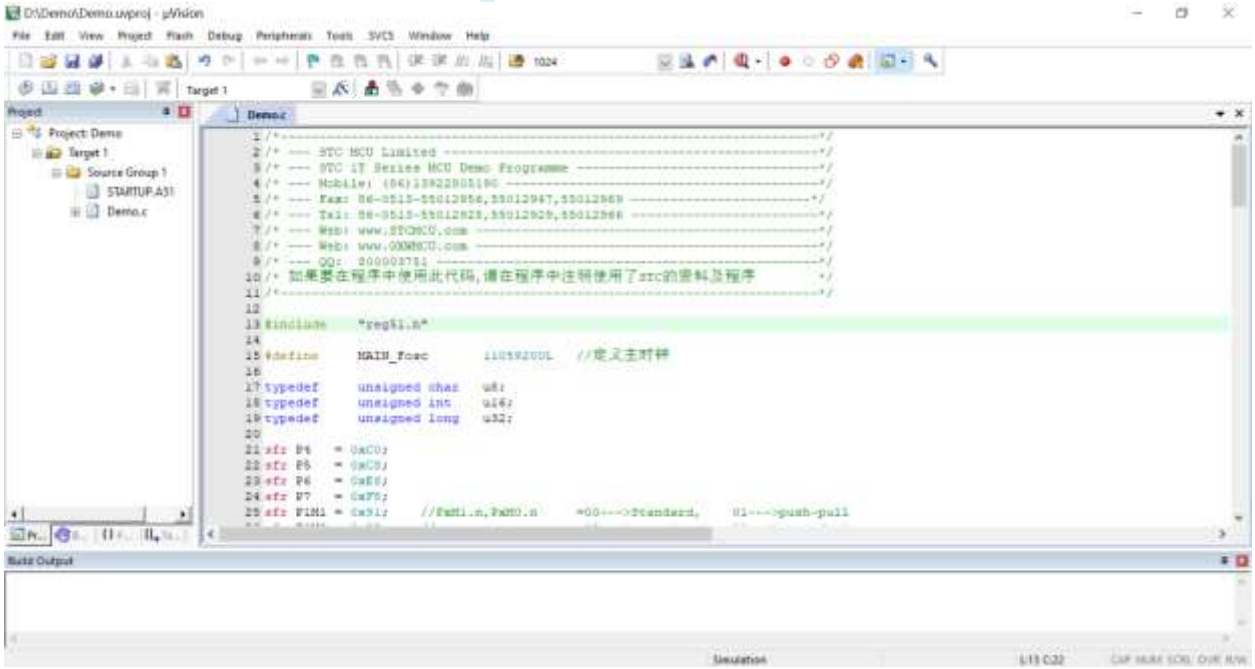
接下来需要按一下 STC 通用 USB 转串口工具上的“**电源开关**”给 MCU 供电【冷启动】，即可开始制作仿真芯片。

若设置成功，会出现如下的画面：



到此，仿真芯片便制作成功了。

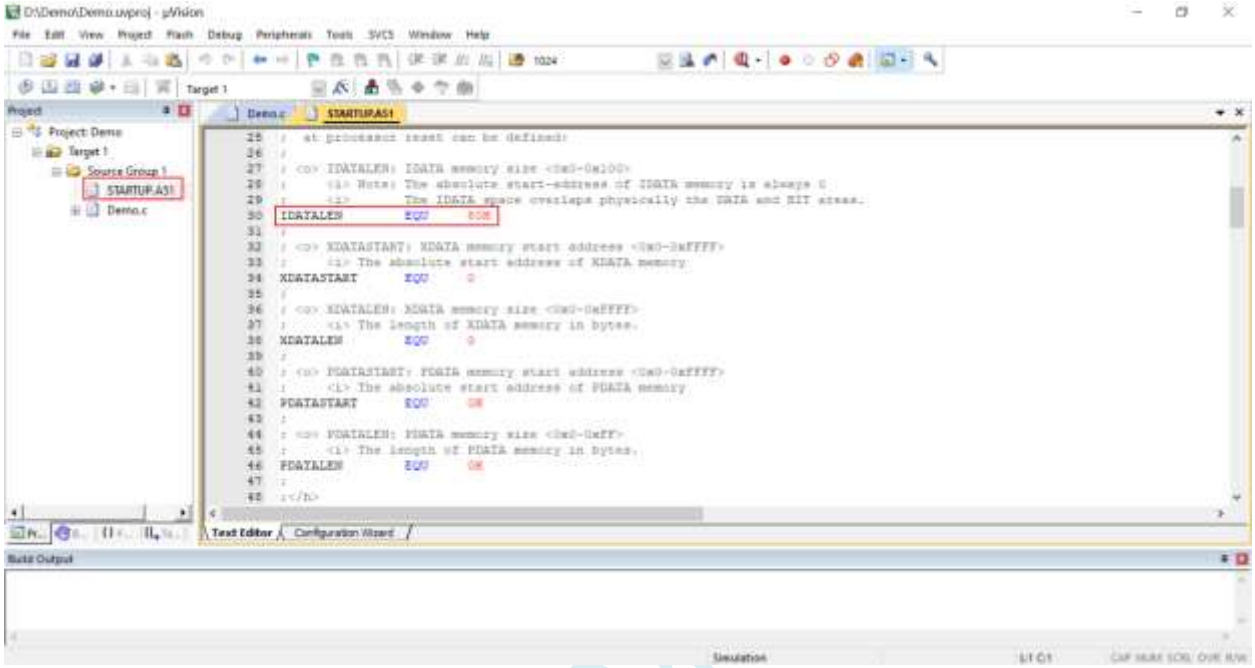
接下来我们打开一个项目进行仿真：



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 0FFH，则会将 IDATA 的 00-FF 的 RAM 初始化为 0。



我们所选的 STC12H 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 …”；

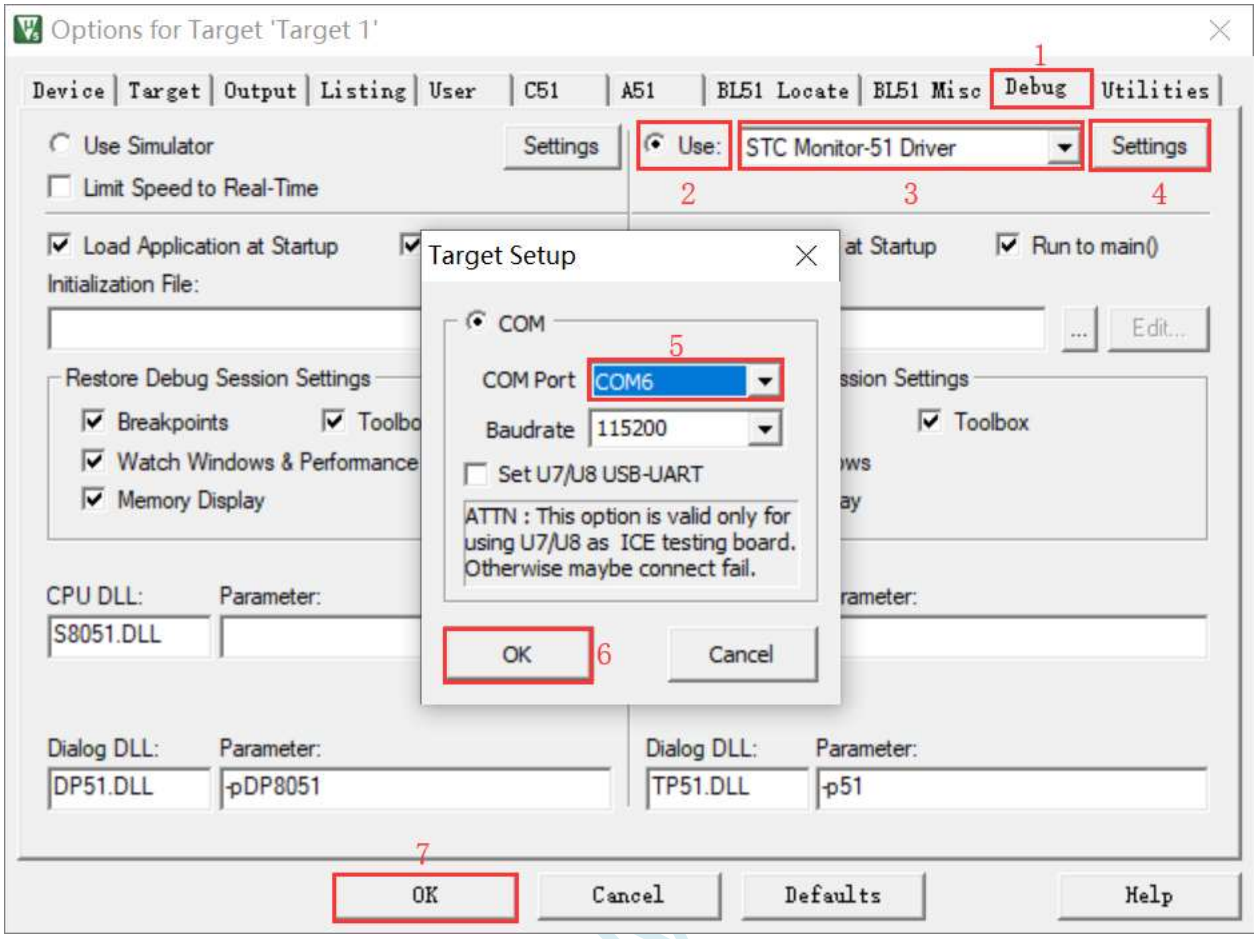
第 3 步、在仿真驱动下拉列表中选择“STC Monitor-51 Driver”项；

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。

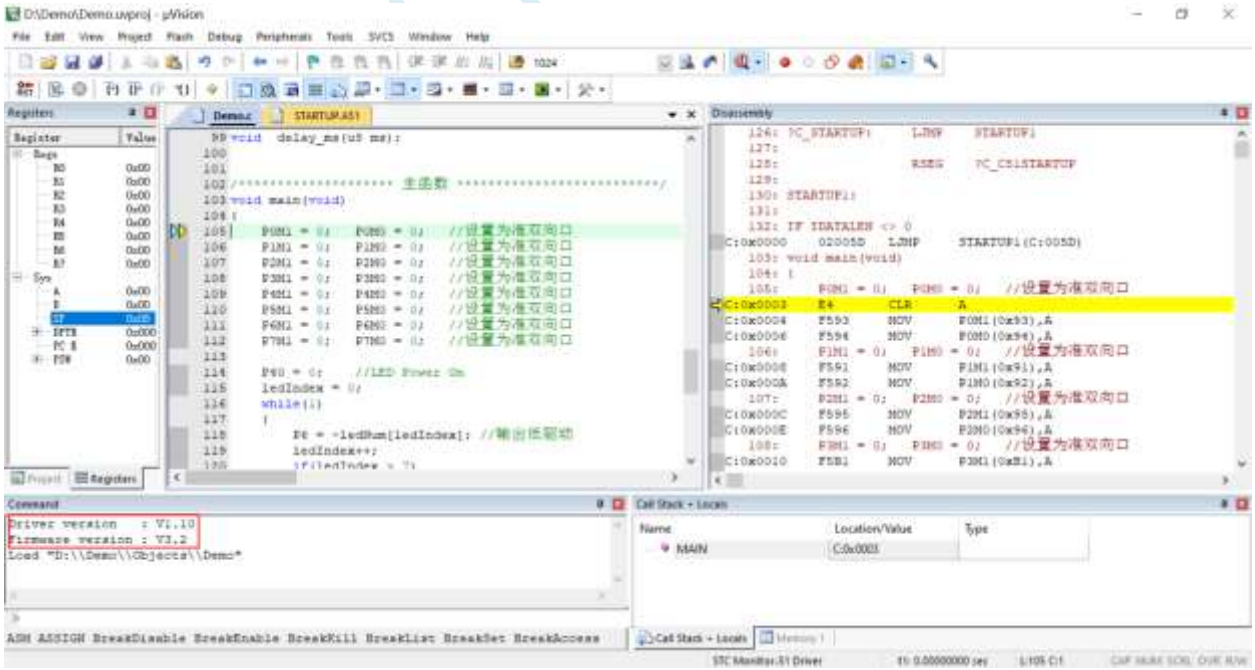
确定完成仿真设置。

详细步骤如下图所示：

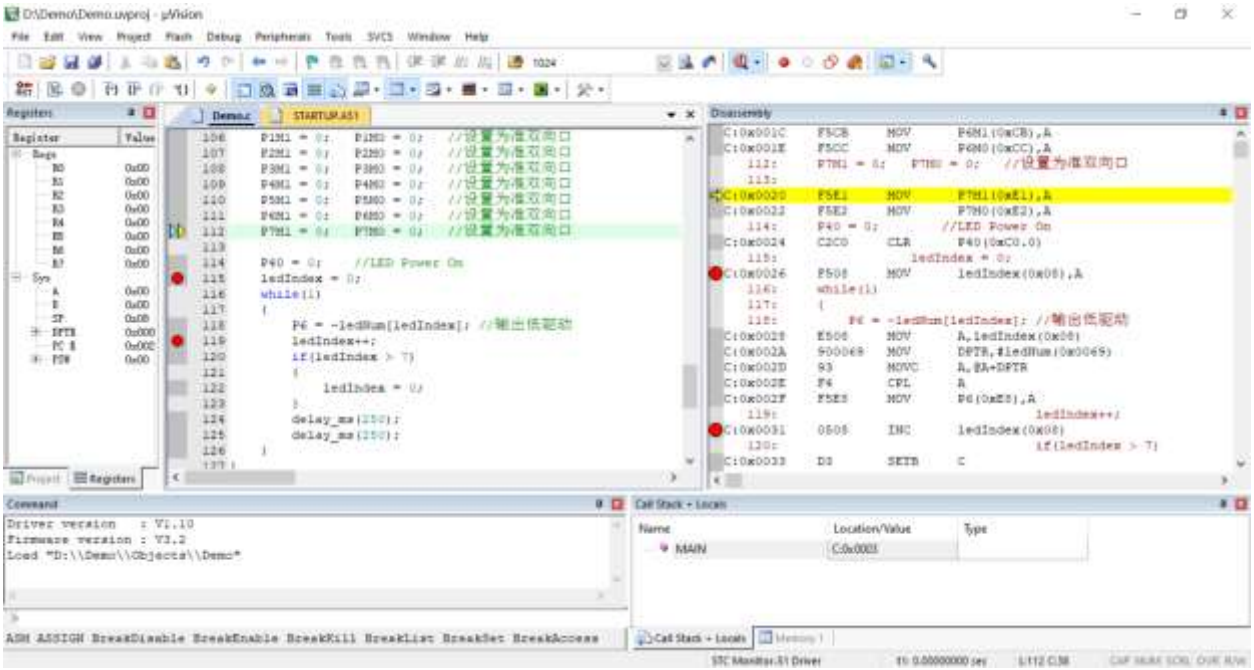


完成了上面所有的工作后，即可在 Keil 软件中按“Ctrl+F5”开始仿真调试。

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：



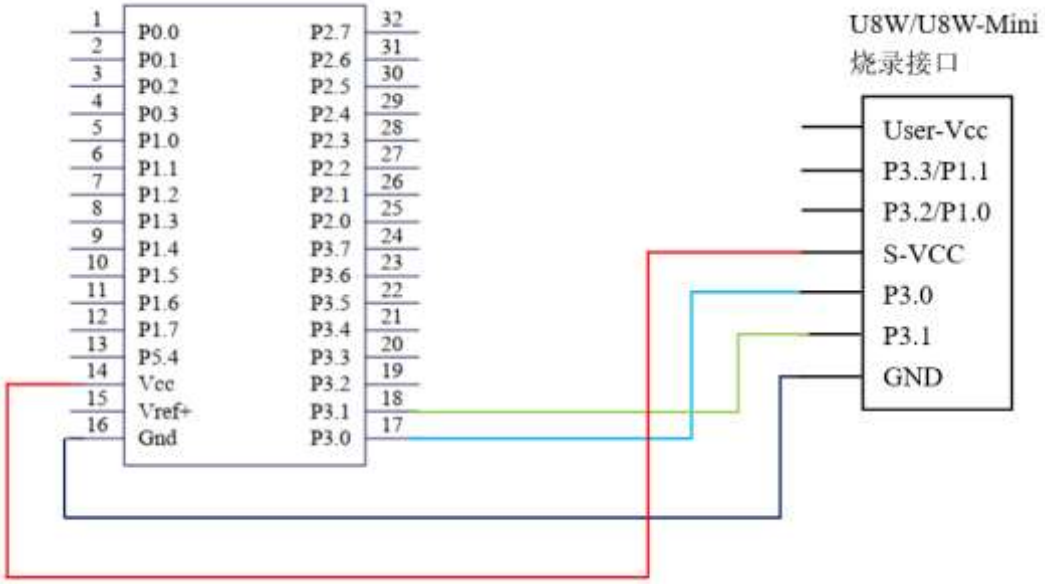
仿真调试过程中，可执行复位、全速运行、单步运行、设置断点等多中操作。



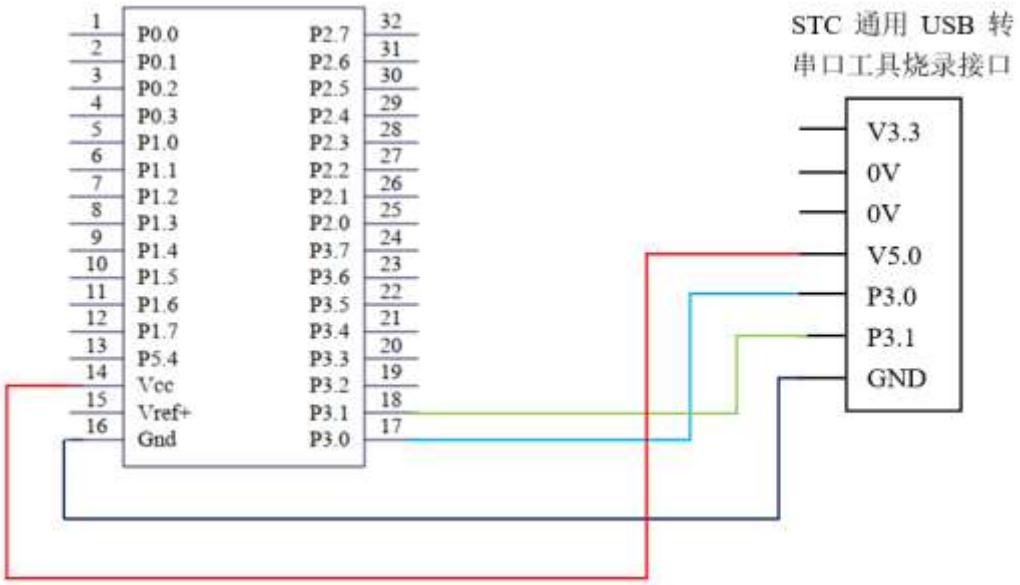
如上图所示，可在程序中设置多个断点，断点设置的个数目前最大允许 20 个（理论上可设置任意个，但是断点设置得过多会影响调试的速度）。

F.5 应用线路图

F.5.1 U8W 工具应用参考线路图

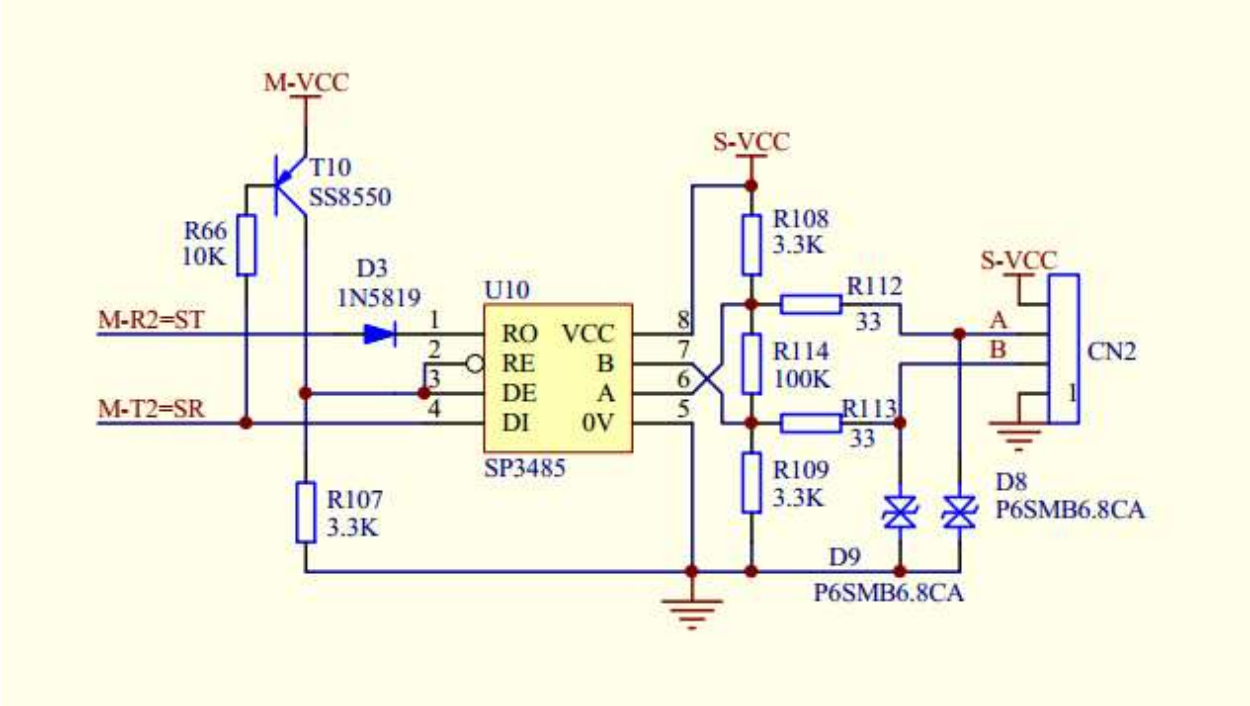


F.5.2 STC 通用 USB 转串口工具应用参考线路图


1 P0.0
2 P0.1
3 P0.2
4 P0.3
5 P1.0
6 P1.1
7 P1.2
8 P1.3
9 P1.4
10 P1.5
11 P1.6
12 P1.7
13 P5.4
14 Vcc
15 Vref+
16 Gnd
P2.7 32
P2.6 31
P2.5 30
P2.4 29
P2.3 28
P2.2 27
P2.1 26
P2.0 25
P3.7 24
P3.6 23
P3.5 22
P3.4 21
P3.3 20
P3.2 19
P3.1 18
P3.0 17
STC 通用 USB 转
串口工具烧录接口
V3.3
0V
0V
V5.0
P3.0
P3.1
GND

STC MCU

附录G　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

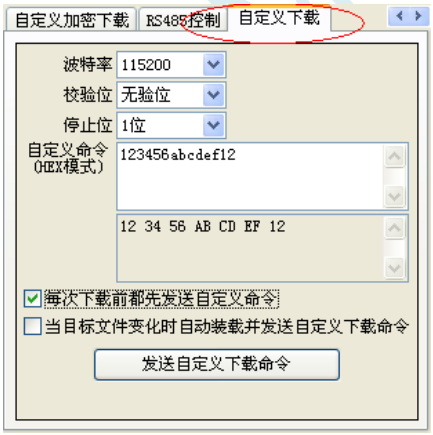
附录H 运行用户程序时收到用户命令后自动启动 ISP 下载(不停电)

“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

Function
Data (Hex)
IRP_MJ_CREATE Port Opened - stc-isp-15xx.exe
IOCTL_SERIAL_SET_BA... Baud Rate: 115200
IOCTL_SERIAL_SET_LI... StopBits: 1, Parity: No, DataBits: 8
IRP_MJ_WRITE Length: 0001, Data: 12
IRP_MJ_WRITE Length: 0001, Data: 34
IRP_MJ_WRITE Length: 0001, Data: 56
IRP_MJ_WRITE Length: 0001, Data: AB
IRP_MJ_WRITE Length: 0001, Data: CD
IRP_MJ_WRITE Length: 0001, Data: EF
IRP_MJ_WRITE Length: 0001, Data: 12
IOCTL_SERIAL_SET_BA... Baud Rate: 2400
IOCTL_SERIAL_SET_LI... StopBits: 1, Parity: Even, DataBits: 8
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
1. 用户的串口设置
2. 用户的自定义命令
3. ISP的串口设置及命令
STC-ISP (V6.38) 官方网站:www.STCMCU.com
单片机型号
引脚数
Auto
用户号
通讯端口 (COM1)
扫描
最低波特率
2400
最高波特率
115200
起始地址
0x0000
清除代码缓冲区
打开程序文件
自定义加密下载
RS485控制
自定义下载
波特率
115200
校验位
无校验位
停止位
1位
自定义命令
(HEX模式)
123456abcdef12
12 34 56 AB CD EF 12
每次下载前都先发送自定义命令
当目标文件变化时 自动装载并发送自定义下载命令
发送自定义下载命令

STC MCU

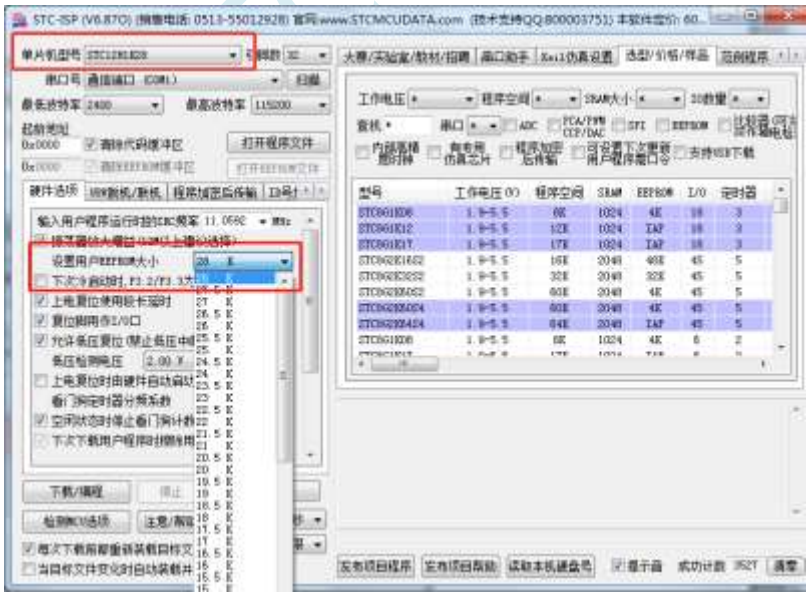
附录I　使用 STC 的 IAP 系列单片机开发自己的 ISP 程序

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。

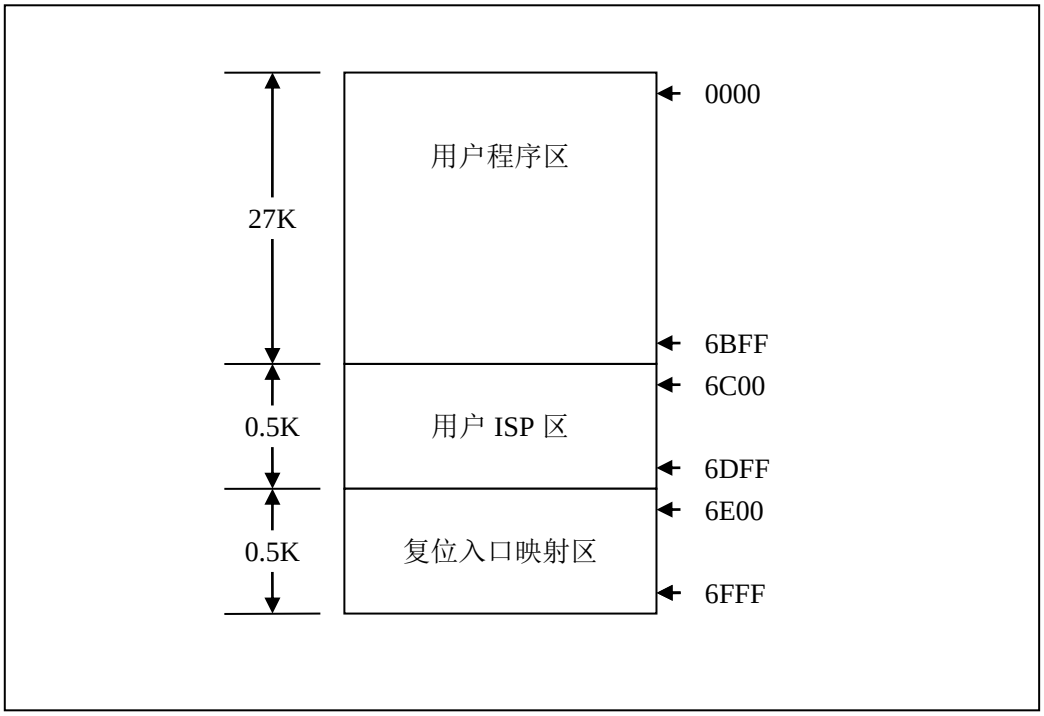
STC12H 系列单片机中的所有可以在 ISP 下载时用户自定义 EEPROM 大小的型号均为 IAP 系列单片机。目前 STC12H 系列有如下型号的单片机为 IAP 系列：STC12H1K28、STC12H1K33。本文以 STC12H1K28 为例，详细说明使用 STC 的 IAP 单片机开发用户自己的 ISP 程序的方法，并给出了基于 Keil 环境的汇编和 C 源码。

第一步：内部 FLASH 规划

由于 STC12H 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 28K 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。

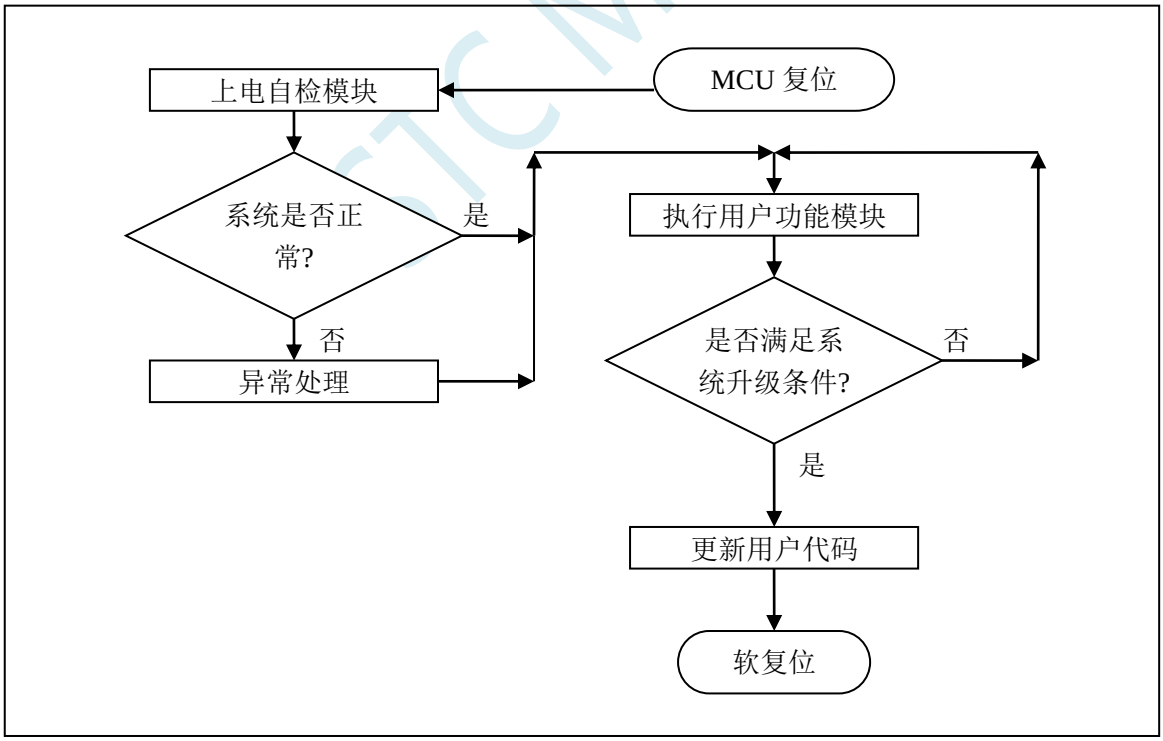


下面假设用户已将整个的 28K 的程序空间已全部设置为 EEPROM，现将整个 28K 程序空间作如下划分：



FLASH 空间中，从地址 0000H 开始的连续 27K 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
```

```
UARTBAUD    EQU     0FFE8H          ;定义串口波特率(65536-11059200/4/115200)

AUXR        DATA    08EH            ;附加功能控制寄存器
WDT_CONTR   DATA    0C1H            ;看门狗控制寄存器
IAP_DATA    DATA    0C2H            ;IAP 数据寄存器
IAP_ADDRH   DATA    0C3H            ;IAP 高地址寄存器
IAP_ADDRL   DATA    0C4H            ;IAP 低地址寄存器
IAP_CMD     DATA    0C5H            ;IAP 命令寄存器
IAP_TRIG    DATA    0C6H            ;IAP 命令触发寄存器
IAP_CONTR   DATA    0C7H            ;IAP 控制寄存器
IAP_TPS     DATA    0F5H            ;IAP 等待时间控制寄存器

ISPCODE     EQU     06C00H          ;ISP 模块入口地址(1 页),同时也是外部接口地址
APENTRY     EQU     06E00H          ;应用程序入口地址数据(1 页)

            ORG     0000H

            LJMP    ISP_ENTRY       ;系统复位入口

RESET:
            MOV     SCON,#50H       ;设置串口模式(8 位数据位,无校验位)
            MOV     AUXR,#40H       ;定时器 1 为 1T 模式
            MOV     TMOD,#00H       ;定时器 1 工作于模式 0(16 位重装载)
            MOV     TH1,#HIGH UARTBAUD  ;设置重载值
            MOV     TL1,#LOW UARTBAUD
            SETB    TR1             ;启动定时器 1
NEXT1:
            MOV     R0,#16
NEXT2:
            JNB     RI,$            ;等待串口数据
            CLR     RI
            MOV     A,SBUF
            CJNE    A,#7FH,NEXT1    ;判断是否为 7F
            DJNZ    R0,NEXT2
            LJMP    ISP_DOWNLOAD    ;跳转到下载界面

            ORG     ISPCODE

ISP_DOWNLOAD:
            CLR     A
            MOV     PSW,A           ;ISP 模块使用第 0 组寄存器
            MOV     IE,A            ;关闭所有中断
            CLR     RI              ;清除串口接收标志
            SETB    TI              ;置串口发送标志
            CLR     TR0
            MOV     SP,#5FH         ;设置堆栈指针

            MOV     A,#5AH          ;返回 5A55 到 PC,表示 ISP 擦除模块已准备就绪
            LCALL   ISP_SENDUART
            MOV     A,#055H
            LCALL   ISP_SENDUART
            LCALL   ISP_RECVACK     ;接收应答数据

            MOV     IAP_ADDRL,#0    ;首先在第 2 页起始地址写 "LJMP ISP_ENTRY" 指令
            MOV     IAP_ADDRH,#02H
            LCALL   ISP_ERASEIAP
            MOV     A,#02H
            LCALL   ISP_PROGRAMIAP  ;编程用户代码复位向量代码
```

```
        MOV     A,#HIGH         ISP_ENTRY
        LCALL   ISP_PROGRAMIAP  ;编程用户代码复位向量代码
        MOV     A,#LOW ISP_ENTRY
        LCALL   ISP_PROGRAMIAP  ;编程用户代码复位向量代码

        MOV     IAP_ADDRL,#0    ;用户代码地址从 0 开始
        MOV     IAP_ADDRH,#0
        LCALL   ISP_ERASEIAP
        MOV     A,#02H
        LCALL   ISP_PROGRAMIAP  ;编程用户代码复位向量代码
        MOV     A,#HIGH         ISP_ENTRY
        LCALL   ISP_PROGRAMIAP  ;编程用户代码复位向量代码
        MOV     A,#LOW ISP_ENTRY
        LCALL   ISP_PROGRAMIAP  ;编程用户代码复位向量代码

        MOV     IAP_ADDRL,#0    ;新代码缓冲区地址
        MOV     IAP_ADDRH,#02H
        MOV     R7,#124         ;擦除 62.5K 字节
ISP_ERASEAP:
        LCALL   ISP_ERASEIAP
        INC     IAP_ADDRH       ;目标地址+512
        INC     IAP_ADDRH
        DJNZ    R7,ISP_ERASEAP  ;判断是否擦除完成

        MOV     IAP_ADDRL,#LOW APENTRY
        MOV     IAP_ADDRH,#HIGH APENTRY
        LCALL   ISP_ERASEIAP

        MOV     A,#5AH          ;返回 5AA5 到 PC,表示 ISP 编程模块已准备就绪
        LCALL   ISP_SENDUART
        MOV     A,#0A5H
        LCALL   ISP_SENDUART
        LCALL   ISP_RECVACK     ;接收应答数据

        LCALL   ISP_RECVUART    ;接收长度高字节
        MOV     R0,A
        LCALL   ISP_RECVUART    ;接收长度低字节
        MOV     R1,A
        CLR     C               ;将总长度-3
        MOV     A,#03H
        SUBB    A,R1
        MOV     DPL,A
        CLR     A
        SUBB    A,R0
        MOV     DPH,A           ;总长度补码存入 DPTR

        LCALL   ISP_RECVUART    ;映射用户代码复位入口代码到映射区
        LCALL   ISP_PROGRAMIAP  ;0000
        LCALL   ISP_RECVUART
        LCALL   ISP_PROGRAMIAP  ;0001
        LCALL   ISP_RECVUART
        LCALL   ISP_PROGRAMIAP  ;0002

        MOV     IAP_ADDRL,#03H  ;用户代码起始地址
        MOV     IAP_ADDRH,#00H
ISP_PROGRAMNEXT:
        LCALL   ISP_RECVUART    ;接收代码数据
        LCALL   ISP_PROGRAMIAP  ;编程到用户代码区
        INC     DPTR
```

```
            MOV     A,DPL
            ORL     A,DPH
            JNZ     ISP_PROGRAMNEXT         ;长度检测

ISP_SOFTRESET:
            MOV     IAP_CONTR,#20H          ;软件复位系统
            SJMP    $


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_ADDRL,#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_ADDRL               ;测试地址 FC01H
            MOV     IAP_DATA,#45H           ;测试数据 2
            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               ;设置串口模式(8 位数据位,无校验位)
            MOV     AUXR,#40H               ;定时器 1 为 1T 模式
            MOV     TMOD,#00H               ;定时器 1 工作于模式 0(16 位重装载)
            MOV     TH1,#HIGH UARTBAUD      ;设置重载值
            MOV     TL1,#LOW UARTBAUD
            SETB    TR1                     ;启动定时器 1
            SETB    TR0

            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 5A69 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
```

```
            MOV     SCON,A
            MOV     AUXR,A
            LJMP    APENTRY             ;正常运行用户程序

ISP_RECVACK:
            LCALL   ISP_RECVUART
            JC      GOTOAP
            XRL     A,#7FH
            JZ      ISP_RECVACK         ;跳过同步数据
            CJNE    A,#25H,GOTOAP       ;应答数据 1 检测
            LCALL   ISP_RECVUART
            JC      GOTOAP
            CJNE    A,#69H,GOTOAP       ;应答数据 2 检测
            RET

ISP_RECVUART:
            CLR     A
            MOV     TL0,A               ;初始化超时定时器
            MOV     TH0,A
            CLR     TF0
            MOV     WDT_CONTR,#17H      ;清看门狗
ISP_RECVWAIT:
            JBC     TF0,ISP_RECVTIMEOUT ;超时检测
            JNB     RI,ISP_RECVWAIT     ;等待接收完成
            MOV     A,SBUF              ;读取串口数据
            CLR     RI                  ;清除标志
            CLR     C                   ;正确接收串口数据
            RET
ISP_RECVTIMEOUT:
            SETB    C                   ;超时退出
            RET

ISP_SENDUART:
            MOV     WDT_CONTR,#17H      ;清看门狗
            JNB     TI,ISP_SENDUART     ;等待前一个数据发送完成
            CLR     TI                  ;清除标志
            MOV     SBUF,A              ;发送当前数据
            RET

ISP_ERASEIAP:
            MOV     WDT_CONTR,#17H      ;清看门狗
            MOV     IAP_CMD,#3          ;擦除命令
            MOV     IAP_TRIG,#5AH       ;触发 ISP 命令
            MOV     IAP_TRIG,#0A5H
            NOP
            NOP
            NOP
            NOP
            RET

ISP_PROGRAMIAP:
            MOV     WDT_CONTR,#17H      ;清看门狗
            MOV     IAP_CMD,#2          ;编程命令
            MOV     IAP_DATA,A          ;将当前数据送 IAP 数据寄存器
            MOV     IAP_TRIG,#5AH       ;触发 ISP 命令
            MOV     IAP_TRIG,#0A5H
            NOP
            NOP
            NOP
```

```
        NOP
        MOV     A,IAP_ADDRL         ;IAP 地址+1
        ADD     A,#01H
        MOV     IAP_ADDRL,A
        MOV     A,IAP_ADDRH
        ADDC    A,#00H
        MOV     IAP_ADDRH,A
        RET


        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 6C00H

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 6C00H 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    0x6c00              //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;                                     //清接收完成标志
    }
}

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)
    {
        P1++;
    }
}
```

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     06C00H          ;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     ACC
        RETI

START:
        MOV     R0,#7FH             ;清 RAM
        CLR     A
        MOV     @R0,A
        DJNZ    R0,$-1
        MOV     SP,#7FH             ;初始化 SP

        MOV     SCON,#50H           ;设置串口模式(8 位可变,无校验位)
        MOV     AUXR,#15H           ;BRT 工作于 1T 模式,启动 BRT
        MOV     TMOD,#00H           ;定时器 1 工作于模式 0(16 位重装载)
        MOV     TH1,#HIGH UARTBAUD  ;设置重载值
        MOV     TL1,#LOW UARTBAUD
        SETB    TR1                 ;启动定时器 1
        SETB    ES                  ;使能串口中断
        SETB    EA                  ;开中断总开关

MAIN:
        INC     P0
        SJMP    MAIN

        END
```

STC MCU

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 6C00H 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 06C00H”指令进行调用，如下图

```
UARTBAUD     EQU     0FFE8H          ;定义串口波特率 (65536-11059200/4/115200)
ISPPROGRAM   EQU     06C00H          ;ISP下载程序入口地址

AUXR         DATA    08EH            ;附件功能控制寄存器
```

```
        CLR     TI                      ;清除标志
CHECKRI:
        JNB     RI,UARTISR_EXIT         ;检测接收中断
        CLR     RI                      ;清除标志
        MOV     A,SBUF
        CJNE    A,#7FH,ISNOT7F
        INC     CNT7F
        MOV     A,CNT7F
        CJNE    A,#16,UARTISR_EXIT
        LJMP    ISPPROGRAM              ;调用下载模块(****重要语句****)
ISNOT7F:
        MOV     CNT7F,#0
UARTISR_EXIT:
        POP     PSW
        POP     ACC
        RETI

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 0x6C00，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      0x6c00                  //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;                         //清接收完成标志
    }
```

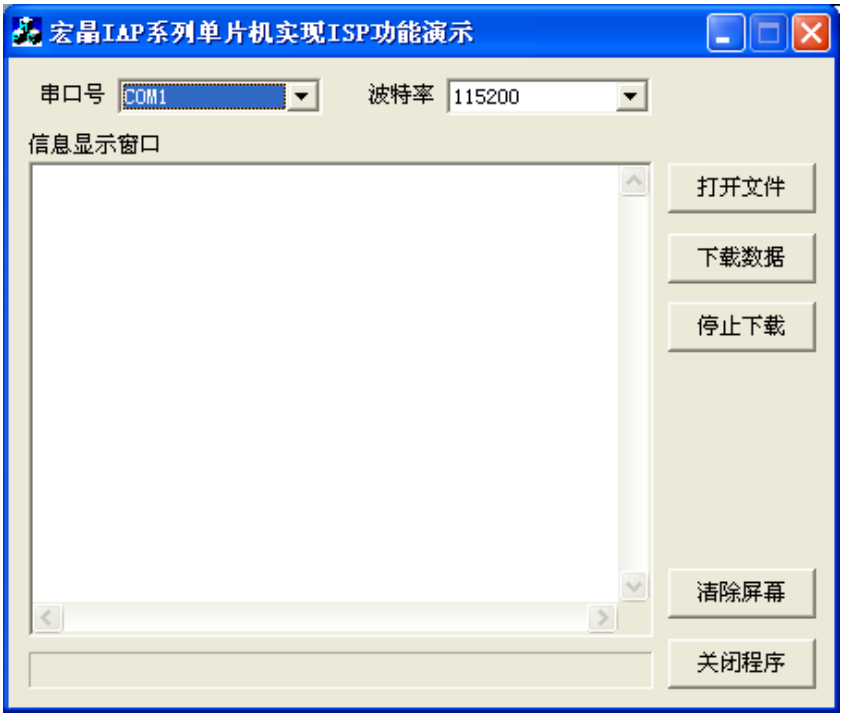
第四步、上位机应用程序说明

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

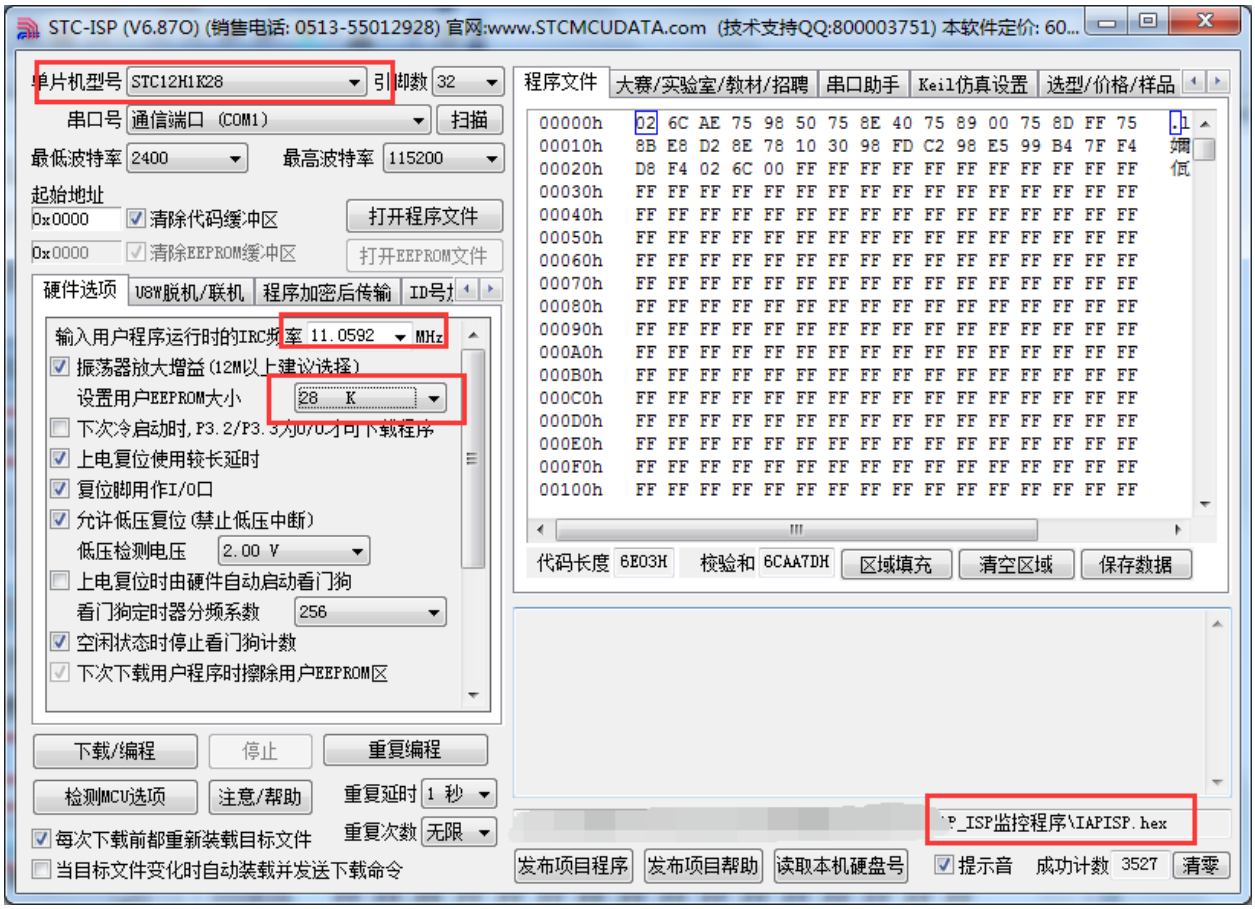
- 打开上位机界面，如下图



- 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J 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

        ...                                 //用户代码
    }
}
```

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     P3M1        =   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;
    P3M1 = 0x00;

    INTCLKO |= 0x40;                            //使能 INT4 中断
    EA = 1;

    while (1)
    {
        ...                                     //用户代码
    }
}
```

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

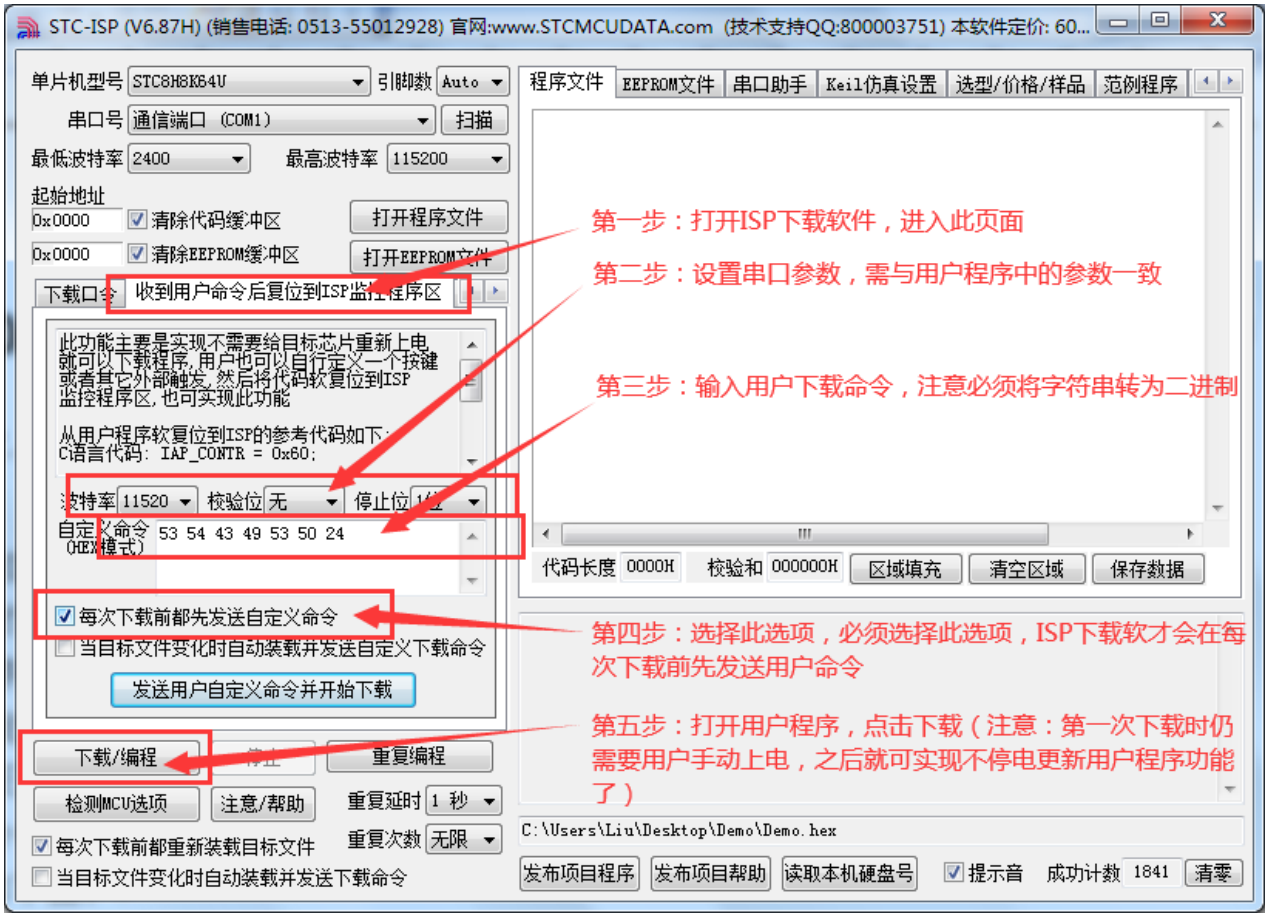
```
//测试工作频率为11.0592MHz
```

```
#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 / 4 / 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 == 'I') 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
    }
}
```

STC MCU

附录K 使用第三方 MCU 对 STC12H 系列单片机进行 ISP 下载范例程序

C 语言代码

```
//注意:使用本代码对 STC12H 系列的单片机进行下载时,必须要执行了 Download 代码之后,
//才能给目标芯片上电,否则目标芯片将无法正确下载

#include "reg51.h"

typedef     bit             BOOL;
typedef     unsigned char   BYTE;
typedef     unsigned short  WORD;

//宏、常量定义
#define     FALSE           0
#define     TRUE            1
#define     LOBYTE(w)       ((BYTE)(WORD)(w))
#define     HIBYTE(w)       ((BYTE)((WORD)(w) >> 8))

#define     MINBAUD         2400L
#define     MAXBAUD         115200L

#define     FOSC            11059200L           //主控芯片工作频率
#define     BR(n)           (65536 - FOSC/4/(n))    //主控芯片串口波特率计算公式
#define     T1MS            (65536 - FOSC/1000)     //主控芯片 1ms 定时初值

#define     FUSER           24000000L           //STC12H 系列目标芯片工作频率
#define     RL(n)           (65536 - FUSER/4/(n))   //STC12H 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 f1ms;                      //1ms 标志位
BOOL UartBusy;                  //串口发送忙标志位
BOOL UartReceived;              //串口数据接收完成标志位
BYTE UartRecvStep;              //串口数据接收控制
BYTE TimeOut;                   //串口通讯超时计数器
BYTE xdata TxBuffer[256];       //串口数据发送缓冲区
BYTE xdata RxBuffer[256];       //串口数据接收缓冲区
char code DEMO[256];            //演示代码数据

//函数声明
void Initial(void);
void DelayXms(WORD x);
BYTE UartSend(BYTE dat);
void CommInit(void);
void CommSend(BYTE size);
```

```
BOOL Download(BYTE *pdat, long size);

//主函数入口
void main(void)
{
	P3M0 = 0x00;
	P3M1 = 0x00;

	Initial();
	if (Download(DEMO, 256))
	{
		//下载成功
		P3 = 0xff;
		DelayXms(500);
		P3 = 0x00;
		DelayXms(500);
		P3 = 0xff;
		DelayXms(500);
		P3 = 0x00;
		DelayXms(500);
		P3 = 0xff;
		DelayXms(500);
		P3 = 0x00;
		DelayXms(500);
		P3 = 0xff;
	}
	else
	{
		//下载失败
		P3 = 0xff;
		DelayXms(500);
		P3 = 0xf3;
		DelayXms(500);
		P3 = 0xff;
		DelayXms(500);
		P3 = 0xf3;
		DelayXms(500);
		P3 = 0xff;
		DelayXms(500);
		P3 = 0xf3;
		DelayXms(500);
		P3 = 0xff;
	}

	while (1);
}

//1ms 定时器中断服务程序
void tm0(void) interrupt 1
{
	static BYTE Counter100;

	f1ms = TRUE;
	if (Counter100-- == 0)
	{
		Counter100 = 100;
		if (TimeOut) TimeOut--;
	}
}
```

STC MCU

```
//串口中断服务程序
void uart(void) interrupt 4
{
	static WORD RecvSum;
	static BYTE RecvIndex;
	static BYTE RecvCount;
	BYTE dat;

	if (TI)
	{
		TI = 0;
		UartBusy = FALSE;
	}

	if (RI)
	{
		RI = 0;
		dat    = SBUF;
		switch (UartRecvStep)
		{
		case 1:
			if (dat != 0xb9) goto L_CheckFirst;
			UartRecvStep++;
			break;
		case 2:
			if (dat != 0x68) goto L_CheckFirst;
			UartRecvStep++;
			break;
		case 3:
			if (dat != 0x00) goto L_CheckFirst;
			UartRecvStep++;
			break;
		case 4:
			RecvSum = 0x68 + dat;
			RecvCount = dat - 6;
			RecvIndex = 0;
			UartRecvStep++;
			break;
		case 5:
			RecvSum += dat;
			RxBuffer[RecvIndex++] = dat;
			if (RecvIndex == RecvCount) UartRecvStep++;
			break;
		case 6:
			if (dat != HIBYTE(RecvSum)) goto L_CheckFirst;
			UartRecvStep++;
			break;
		case 7:
			if (dat != LOBYTE(RecvSum)) goto L_CheckFirst;
			UartRecvStep++;
			break;
		case 8:
			if (dat != 0x16) goto L_CheckFirst;
			UartReceived = TRUE;
			UartRecvStep++;
			break;
L_CheckFirst:
		case 0:
```

```
            default:
                CommInit();
                UartRecvStep = (dat == 0x46 ? 1 : 0);
                break;
            }
        }
}

//系统初始化
void Initial(void)
{
        UartBusy = FALSE;

        SCON = 0xd0;                    //串口数据模式必须为 8 位数据+1 位偶检验
        AUXR = 0xc0;
        TMOD = 0x00;
        TH0 = HIBYTE(T1MS);
        TL0 = LOBYTE(T1MS);
        TR0 = 1;
        TH1 = HIBYTE(BR(MINBAUD));
        TL1 = LOBYTE(BR(MINBAUD));
        TR1 = 1;
        ET0 = 1;
        ES = 1;
        EA = 1;
}

//Xms 延时程序
void DelayXms(WORD x)
{
        do
        {
            f1ms = FALSE;
            while (!f1ms);
        } while (x--);
}

//串口数据发送程序
BYTE UartSend(BYTE dat)
{
        while (UartBusy);

        UartBusy = TRUE;
        ACC = dat;
        TB8 = P;
        SBUF = ACC;

        return dat;
}

//串口通讯初始化
void CommInit(void)
{
        UartRecvStep = 0;
        TimeOut = 20;
        UartReceived = FALSE;
}

//发送串口通讯数据包
```

```
void CommSend(BYTE size)
{
	WORD sum;
	BYTE i;

	UartSend(0x46);
	UartSend(0xb9);
	UartSend(0x6a);
	UartSend(0x00);
	sum = size + 6 + 0x6a;
	UartSend(size + 6);
	for (i=0; i<size; i++)
	{
		sum  += UartSend(TxBuffer[i]);
	}
	UartSend(HIBYTE(sum));
	UartSend(LOBYTE(sum));
	UartSend(0x16);
	while (UartBusy);

	CommInit();
}

//对 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
    ////如果不需要修改硬件选项,此步骤可直接跳过,此时所有的硬件选项
    ////都维持不变,MCU 的频率为上一次所调节频率
    ////若写硬件选项,MCU 的内部 IRC 频率将被固定写为 24M, ,其他选项恢复为出厂设置
    ////建议:第一次使用 STC-ISP 下载软件将从芯片的硬件选项设置好
    ////以后再使用主芯片对从芯片下载程序时不写硬件选项
    //DelayXms(10);
    //for (cnt=0; cnt<128; cnt++)
    //{
    //      TxBuffer[cnt] = 0xff;
    //}
    //TxBuffer[0] = 0x04;
    //TxBuffer[1] = 0x00;
    //TxBuffer[2] = 0x00;
    //TxBuffer[3] = 0x5a;
    //TxBuffer[4] = 0xa5;
    //TxBuffer[33] = arg;
    //TxBuffer[34] = 0x00;
    //TxBuffer[35] = 0x01;
    //TxBuffer[41] = 0xbf;
    //TxBuffer[42] = 0xbd;                //P5.4 为 IO 口
    ////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,
};
```

备注：用户若需要设置不同的工作频率，可参考 7.3.7 和 7.3.8 章的范例代码

STC MCU

附录L 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       = 1046;
    const UINT ID_STOP          = 1044;
    const UINT ID_COMPORT       = 1009;
    const UINT ID_PROGRESS      = 1044;

    //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;
}
```

```
}
```

STC MCU

附录M STC8H 系列正交解码示例（成都逐飞科技友情提供）

受 STC 委托，本文意在分享使用 STC8H 系列单片机的增强型 PWM 模块实现正交解码功能，以进一步实现对正交编码信号输出的编码器进行双向速度测量。

硬件平台：逐飞 STC8H8K-64 脚核心板+逐飞 1024 线 mini 正交编码器
编译环境：keil V9.60
上位机：串口助手

从逐飞科技之前的开源库中我们可以了解到，开源库里没有正交解码例程，主要原因是因为 STC8H 只有两个 PWM 模块，假如我们推荐使用正交编码编码器，就意味着一个编码器就需要占用一个 PWM 模块，然而今年节能组要求制作平衡小车，就意味着有两个电机，这样就需要两个编码器，那么单片机的两个 PWM 模块就会都被占用，然而小车的电机控制也需要 PWM 功能，所以并没有推荐大家用 PWM 模块来实现正交解码，而是推荐大家使用带方向输出的编码器，这样 PWM 模块就可以留给电机使用。

当然也可以有另一种设想，使用一个 PWM 模块去实现对一个正交编码的编码器进行测速，另一个电机使用带方向信号的编码器，用普通定时器去捕获脉冲，这种方案是可行的，但没必要这么麻烦。

还有一点需要注意，使用 PWM 模块计数和使用定时器模块捕获脉冲计数的方式是不一样的。PWM 模块去捕获编码器数据是通过边沿计数，也就是说这个模块的是在发生上升沿或者下降沿的时候都会计数，而定时器捕获脉冲，是获取高低电平翻转的次数。这里我们通过实验就会发现，用同一个的正交编码编码器转动 360°，PWM 模块采集编码器数据是定时器捕获脉冲数据的两倍，但这个数据并不是精度

变高，只是单片机的计数方式导致了结果翻倍。

下面是使用 STC8H8K64U 采集正交编码信号输出编码器的示例程序：

C 语言代码

```
#include "headfile.h"

int16 encoder_data;

//-----------------------------------------------------------------
//@brief        PWMA 模块正交解码初始化
//@param        void
//@return       void
//@since        v1.0
//Sample usage: PWMA_encoder_init();                    //初始化正交解码
//@note
//-----------------------------------------------------------------
void PWMA_encoder_init(void)
{
    P_SW2 |= 1<<7;                                      //使能访问 XFR
    PWMA_ARR = 0xFFFF;                                  //设置自动重装载值,当自动重装载的值为 0 时，
                                                        //计数器不工作。
    PWMA_CCMR1 |= 1<<0;                                 //IC1 映射在 TI1FP1 上，即使用 P10 引脚获取方向
    PWMA_CCMR2 |= 1<<0;                                 //IC2 映射在 TI2FP2 上，即使用 P22 引脚捕获边沿跳
    PWMA_SMCR |= 1<<0;                                  //编码器模式 1 根据 TI1FP1 的电平，
                                                        //计数器在 TI2FP2 的边沿向上/下计数
    PWMA_CR1 |= 1<<0;                                   //使能 PWMA 计数器
    PWMA_PS |= 1<<2;                                    //PWMA 的通道 1 使用 P10,PWMB 的通道 2 使用 P22。
}

//-----------------------------------------------------------------
// @brief       PWMA 模块获取正交解码数值
// @param       void
// @return      void
// @since       v1.0
// Sample usage: encoder_data = PWMA_get_encoder();     //获取正交解码数值
// @note
//-----------------------------------------------------------------
int16 PWMA_get_encoder(void)
{
    int16 res;

    res = PWMA_CNTR;                                    //保存当前计数器的值
    PWMA_CNTR = 0;                                      //清空计数器
    return res;
}

//-----------------------------------------------------------------
// @brief       定时器 0 5ms 中断服务函数
// @param       void
// @return      void
// @since       v1.0
// Sample usage:
// @note
//-----------------------------------------------------------------
void TM0_Isr() interrupt 1
{
    encoder_data = PWMA_get_encoder();                  //获取正交解码编码器数值
}
```

```
void main()
{
    DisableGlobalIRQ();                                  //关闭总中断
    board_init();                                        //初始化内部寄存器，勿删除此句代码。
    pit_timer_ms(TIM_0, 5);                              //初始化定时器，5ms 执行一次
    PWMA_encoder_init();                                 //PWMA 模块初始化为正交解码功能
    EnableGlobalIRQ();                                   //开启总中断
    while(1)
    {
        delay_ms(100);                                   //每 100ms 输出一次打印信息
        printf("encoder_data = %d \r\n" ,encoder_data);  //串口 1 打印编码器数据
    }
}
```

演示视频链接：https://www.bilibili.com/video/BV1zT4y177Ht

视频说明：我们将编写好的例程编译，然后下载到单片机，打开串口助手接收单片机的打印数据，旋转编码器观察数据变化，我们发现当编码器不旋转时输出数据为 0，当编码器朝不同方向旋转时可以输出正负两种数值，旋转越快，数值的绝对值越大，正负用来表示两个旋转方向，其中哪个方向为正，哪个方向为负是可以自己定义的。同时我们从程序示例中也看到打印数据是 100ms 一次，而数据采集是 5ms 一次，所以打印出来的数据相当于是间断的，同时因为编码器是 1024 线的高精度，所以观察到数据变化比较大，但如果是使用电机空载固定 PWM 占空比驱动，可以看到编码器的数据输出是十分稳定的。

串口助手接收数据截图：

下图是正交编码的编码器顺时针旋转且角速度逐渐增大的数据

STC MCU

```
SSCOM V5.13.1 串口/网络数据调试器,
通讯端口 串口设置 显示 发送 多字符
[12:28:09.163]收←◆encoder_data = 0
[12:28:09.275]收←◆encoder_data = 0
[12:28:09.388]收←◆encoder_data = 0
[12:28:09.501]收←◆encoder_data = 0
[12:28:09.613]收←◆encoder_data = 0
[12:28:09.724]收←◆encoder_data = 6
[12:28:09.838]收←◆encoder_data = 15
[12:28:09.950]收←◆encoder_data = 18
[12:28:10.062]收←◆encoder_data = 44
[12:28:10.175]收←◆encoder_data = 51
[12:28:10.288]收←◆encoder_data = 67
[12:28:10.400]收←◆encoder_data = 67
[12:28:10.513]收←◆encoder_data = 78
[12:28:10.625]收←◆encoder_data = 68
[12:28:10.737]收←◆encoder_data = 63
[12:28:10.850]收←◆encoder_data = 87
[12:28:10.962]收←◆encoder_data = 112
```

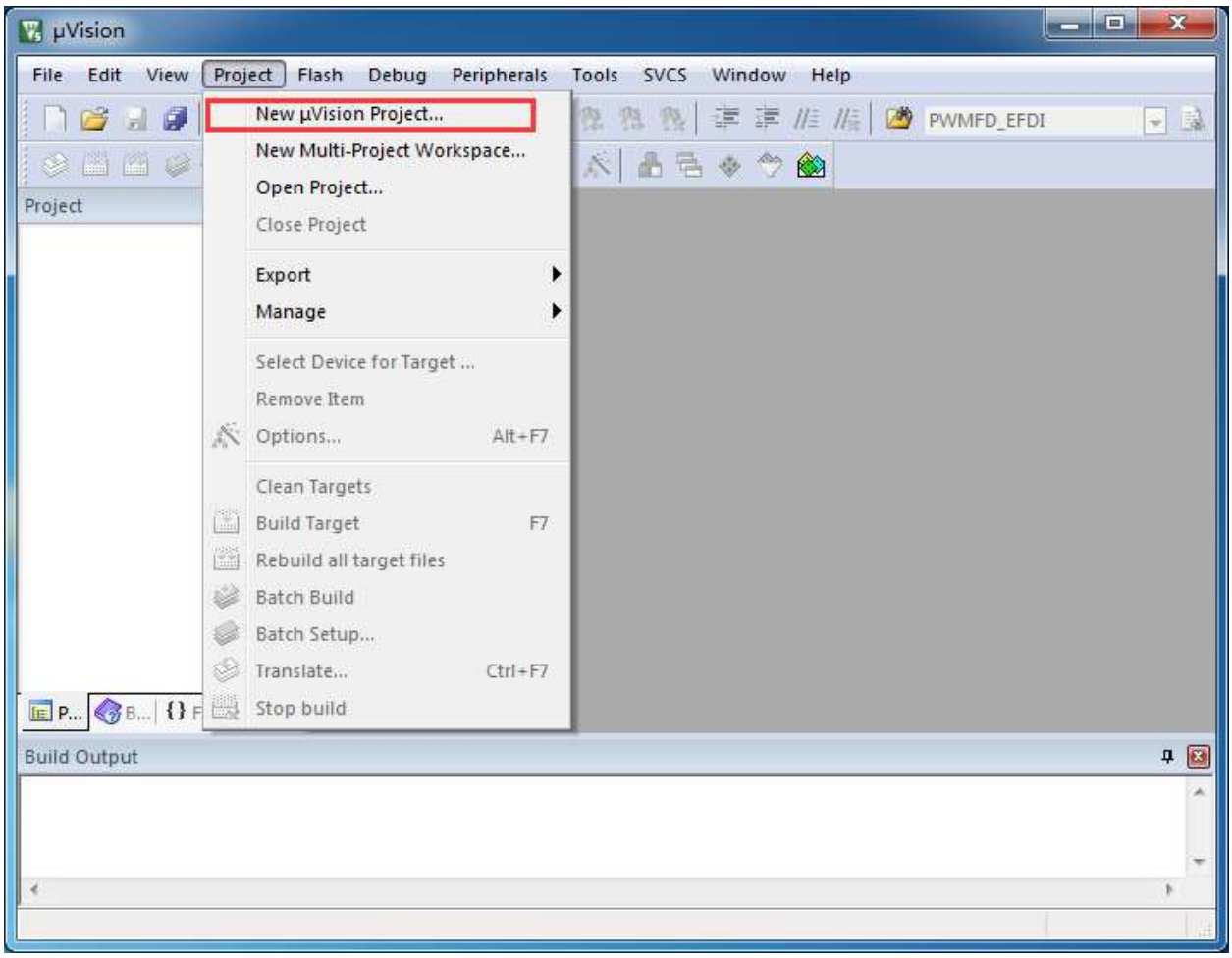
下图是正交编码的编码器逆时针旋转且角速度逐渐增大时的数据：

```
SSCOM V5.13.1 串口/网络数据调试器,
通讯端口 串口设置 显示 发送 多字符
[12:28:23.330]收←◆encoder_data = 0
[12:28:23.443]收←◆encoder_data = 0
[12:28:23.554]收←◆encoder_data = 0
[12:28:23.667]收←◆encoder_data = 0
[12:28:23.780]收←◆encoder_data = 0
[12:28:23.892]收←◆encoder_data = -74
[12:28:24.005]收←◆encoder_data = -120
[12:28:24.117]收←◆encoder_data = -152
[12:28:24.229]收←◆encoder_data = -165
[12:28:24.342]收←◆encoder_data = -210
```

附录N 在 Keil 中建立多文件项目的方法

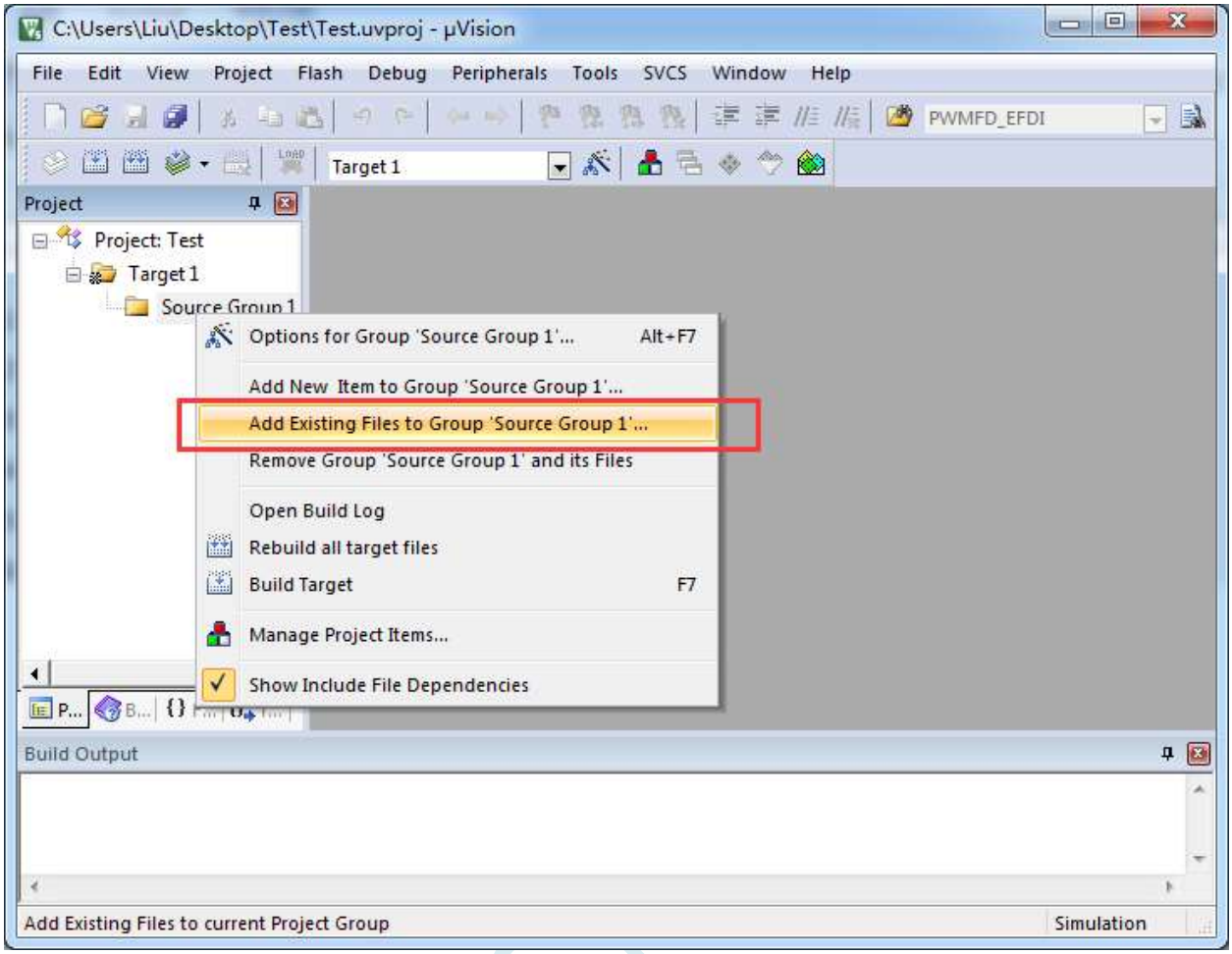
在 Keil 中，一般比较小的项目都只有一个源文件，但对于一些稍微复杂的项目往往需要多个源文件建立多文件项目的方法如下：

1、首先打开 Keil，在菜单“Project”中选择“New uVision Project ...”

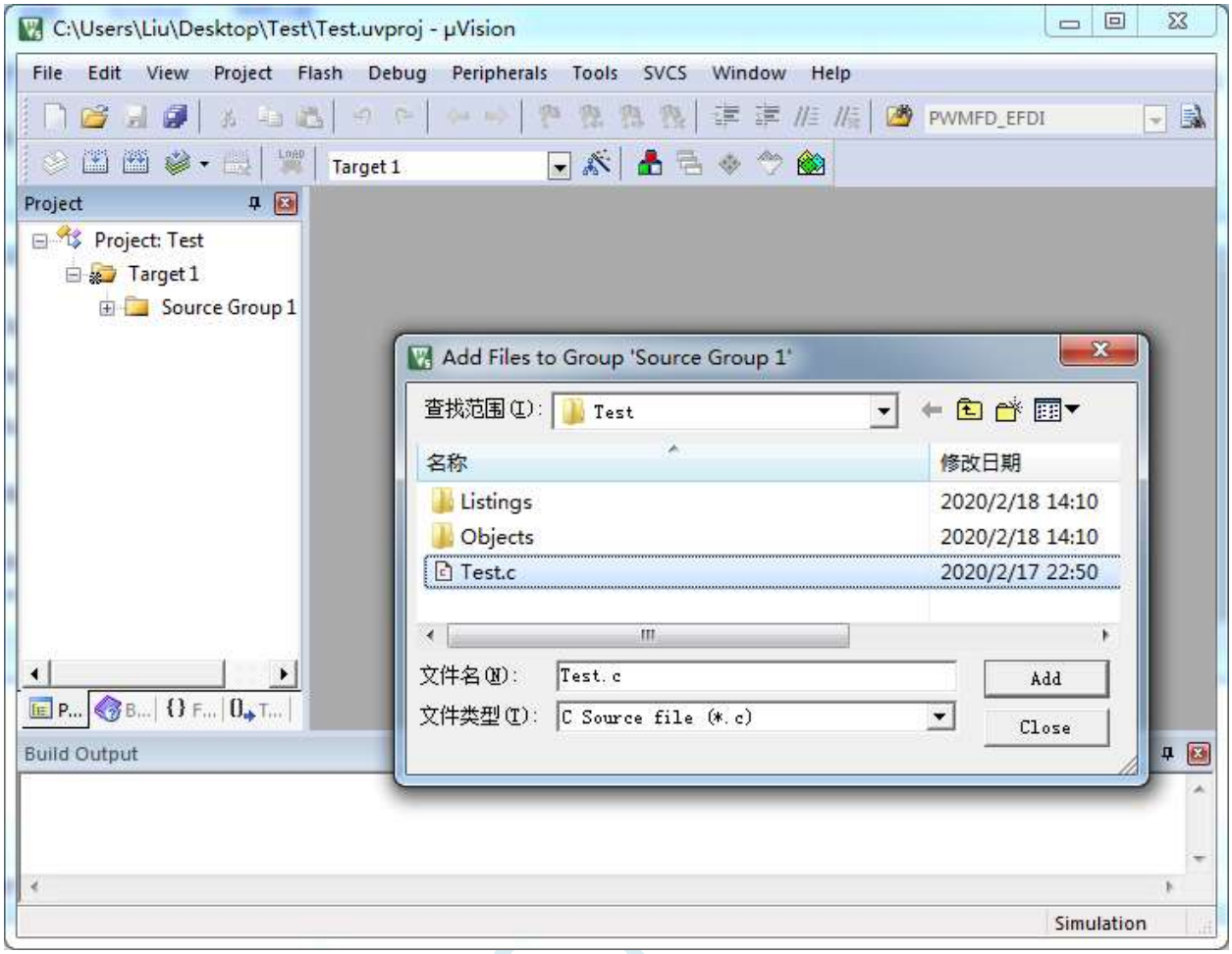


即可完成一个空项目的建立

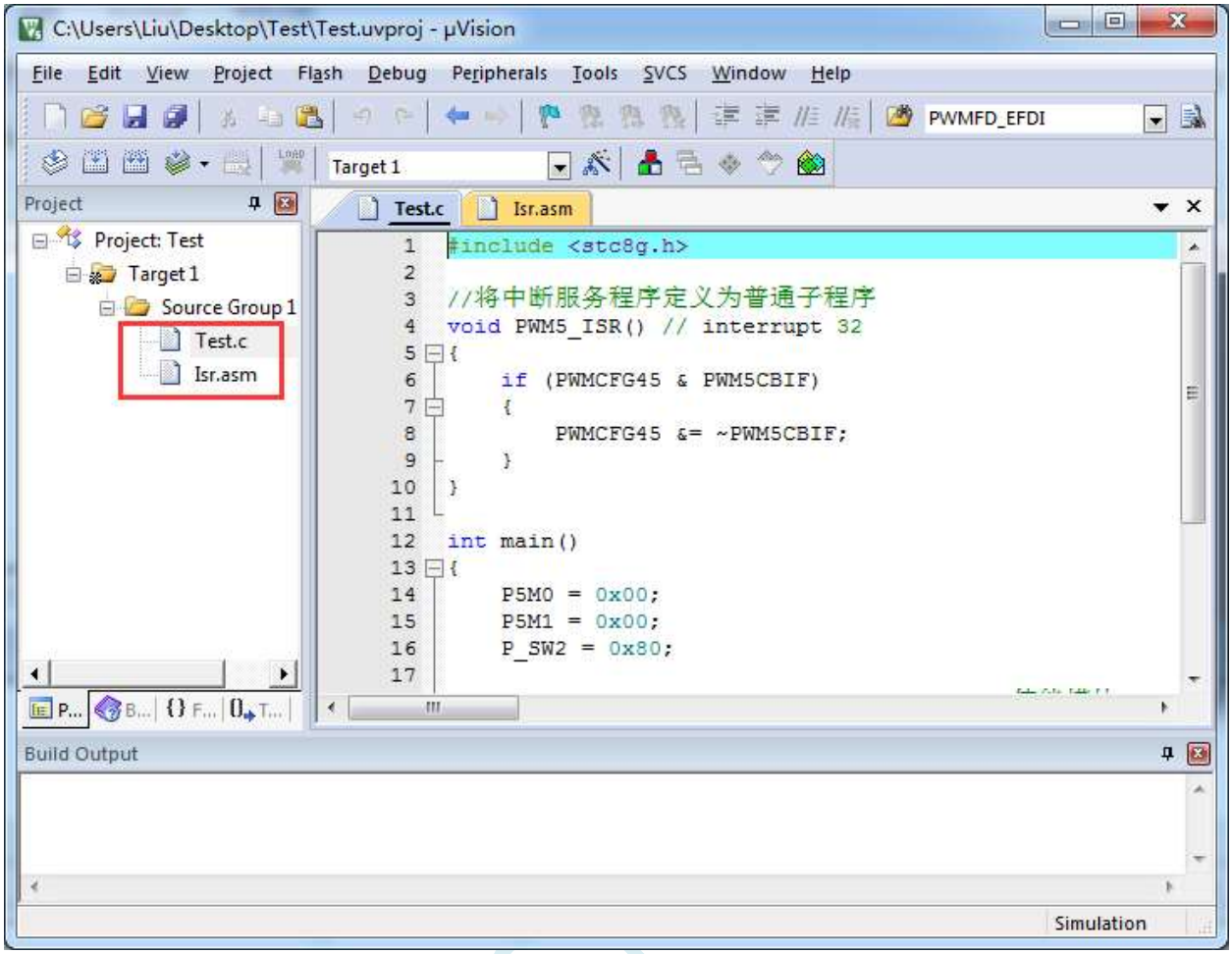
2、在空项目的项目树中，鼠标右键单击“Source Group 1”，并选择右键菜单中的“Add Existing Files to Group "Source Group 1" ...”



3、在弹出的文件对话框中，多次添加源文件

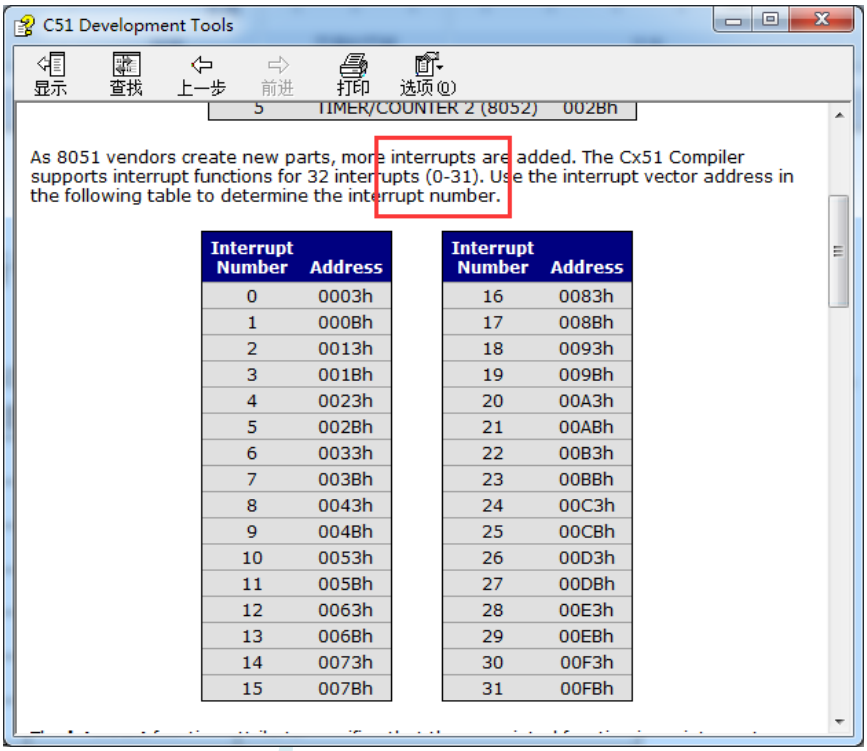


如下图所示即可完成多文件项目的建立


C:\Users\Liu\Desktop\Test\Test.uvproj - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
PWMFD_EFDI
Target 1
Project
Project: Test
Target 1
Source Group 1
Test.c
Isr.asm
Test.c
Isr.asm
#include <stc8g.h>
//将中断服务程序定义为普通子程序
void PWM5_ISR() // interrupt 32
{
if (PWMCFG45 & PWM5CBIF)
{
PWMCFG45 &= ~PWM5CBIF;
}
}
int main()
{
P5M0 = 0x00;
P5M1 = 0x00;
P_SW2 = 0x80;
Build Output
Simulation

附录O 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。



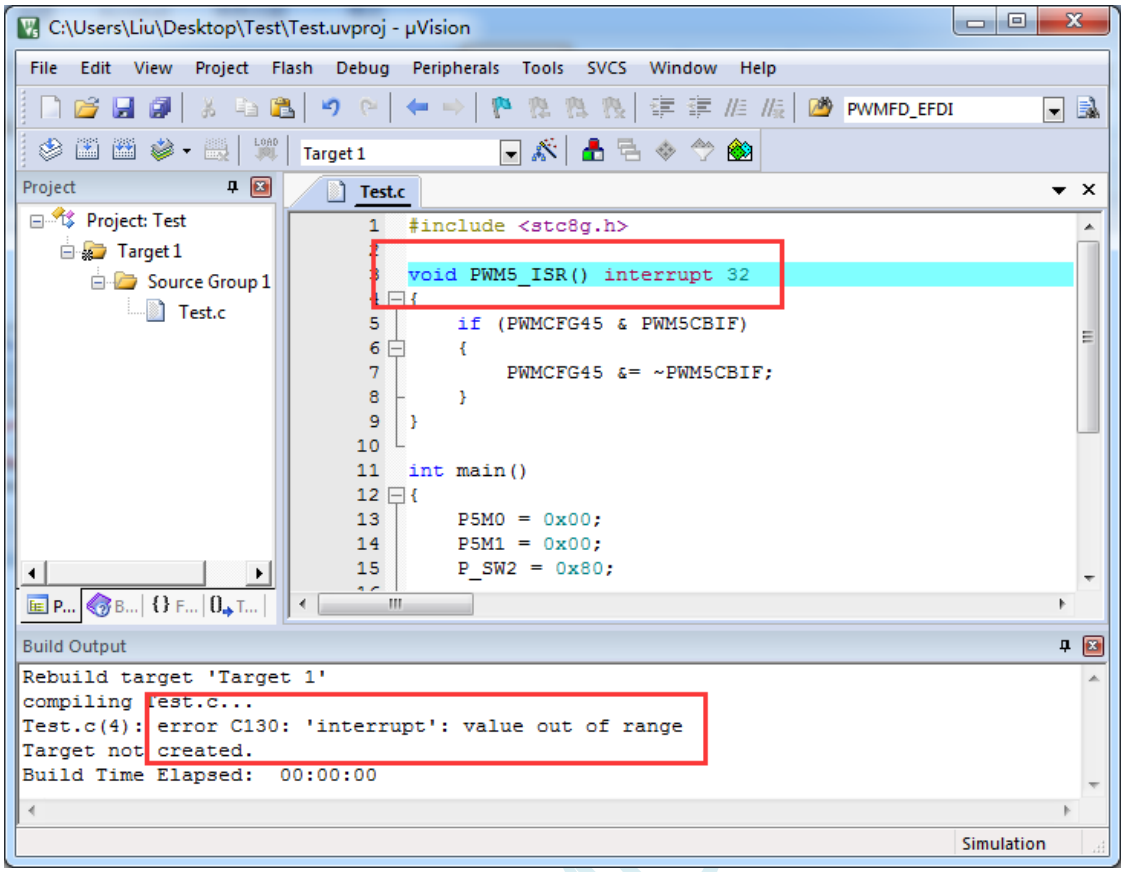
Interrupt Number	Address	Interrupt Number	Address
0	0003h	16	0083h
1	000Bh	17	008Bh
2	0013h	18	0093h
3	001Bh	19	009Bh
4	0023h	20	00A3h
5	002Bh	21	00ABh
6	0033h	22	00B3h
7	003Bh	23	00BBh
8	0043h	24	00C3h
9	004Bh	25	00CBh
10	0053h	26	00D3h
11	005Bh	27	00DBh
12	0063h	28	00E3h
13	006Bh	29	00EBh
14	0073h	30	00F3h
15	007Bh	31	00FBh

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	PWMA
27	00DB H	PWMB
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：



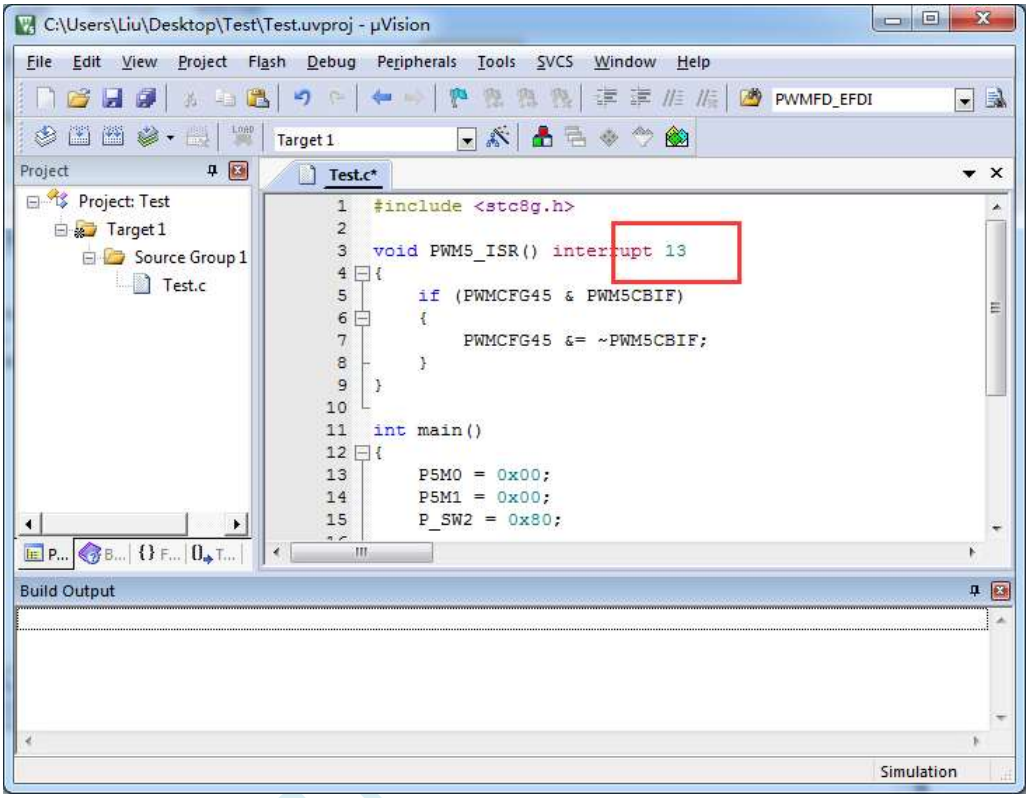
处理这种错误有如下三种方法：（均需要借助于汇编代码，优先推荐使用方法 1）

方法 1：借用 13 号中断向量

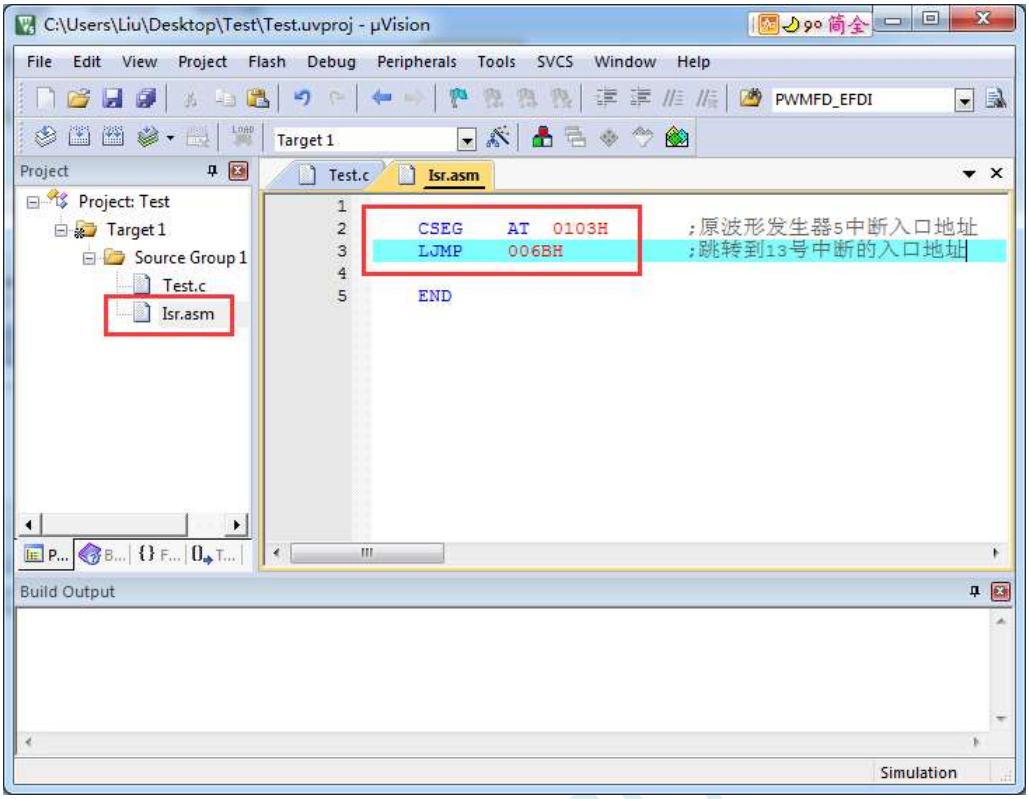
0~31 号中断中，第 13 号是保留中断号，我们可以借用此中断号

操作步骤如下：

1、将我们报错的中断号改为“13”，如下图：

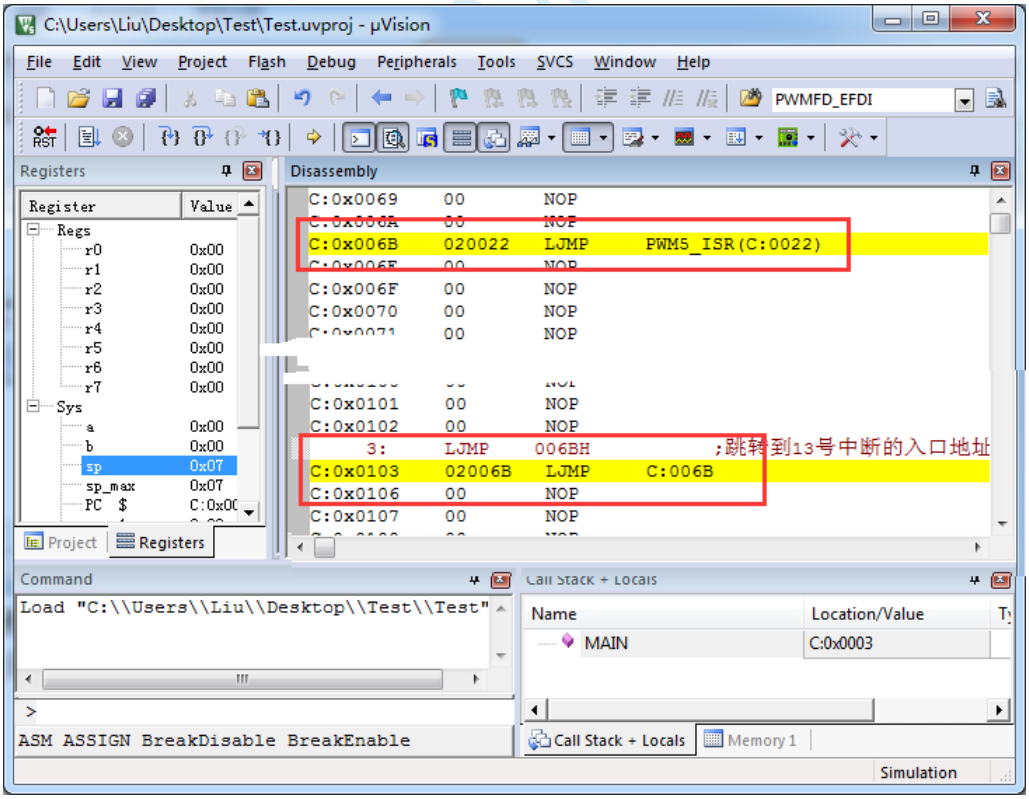


2、新建一个汇编语言文件，比如“isr.asm”，加入到项目，并在地址“0103H”的地方添加一条“LJMP 006BH”，如下图：

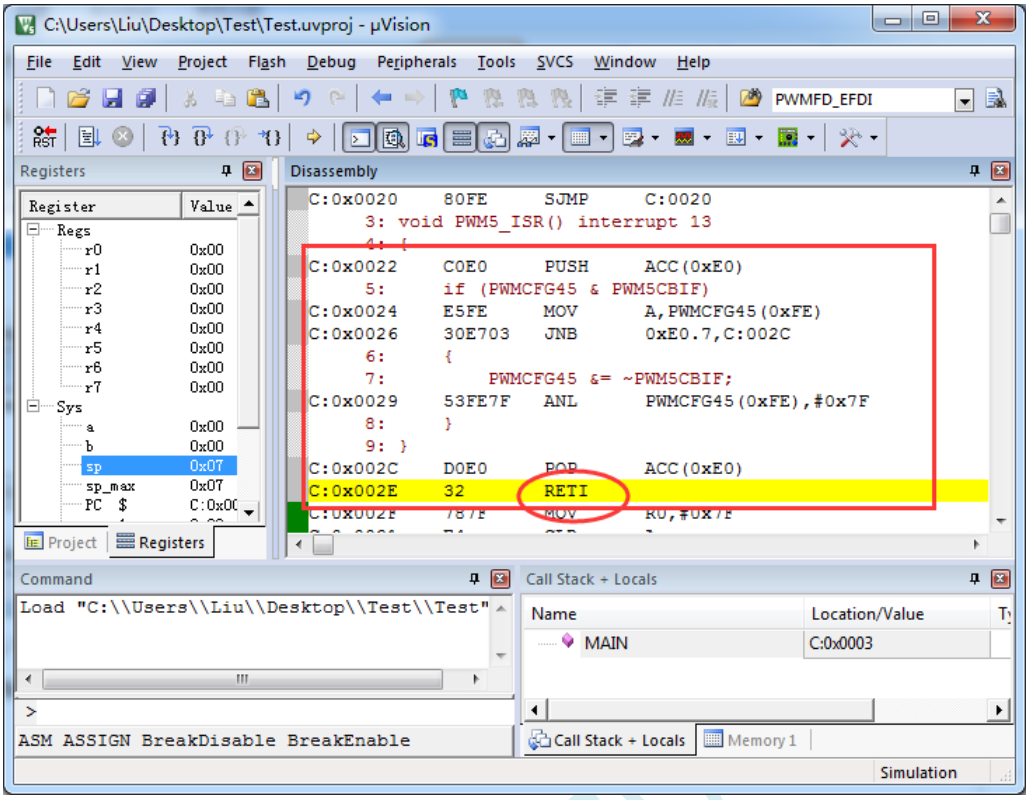


3、编译即可通过。

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



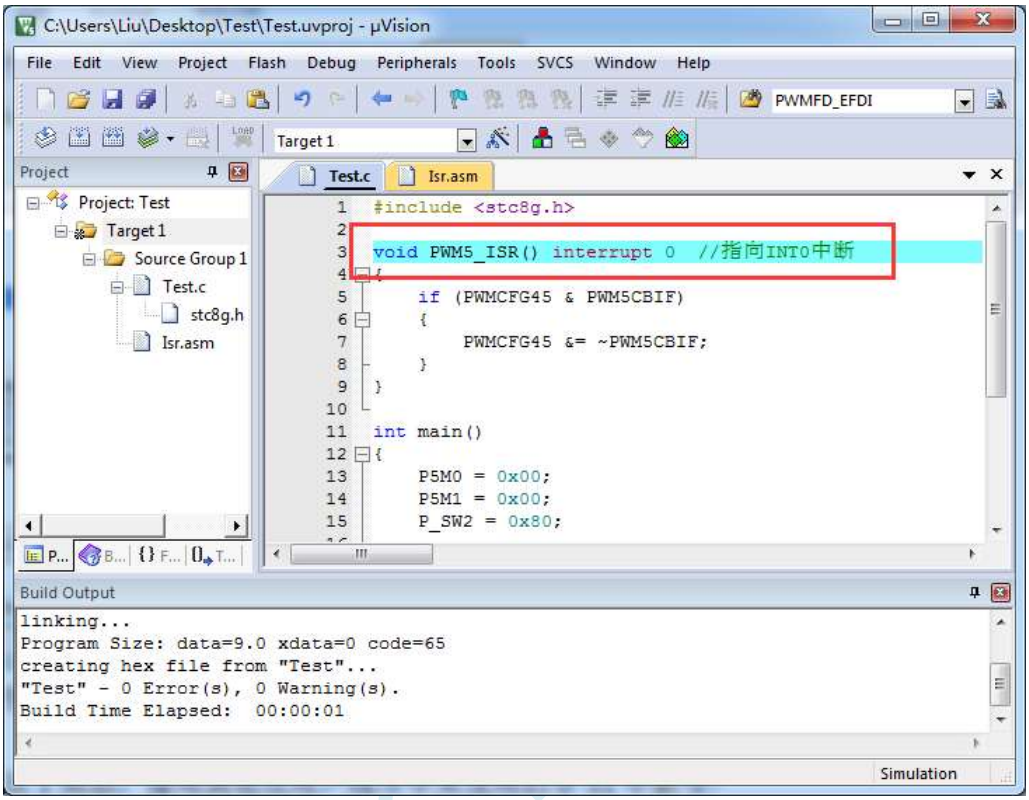
当发生 PWM5 中断时，硬件会自动跳转到 0103H 地址执行“LJMP　　006BH”，然后在 006BH 处再执行“LJMP　　PWM5_ISR”即可跳转到真正的中断服务程序，如下图：

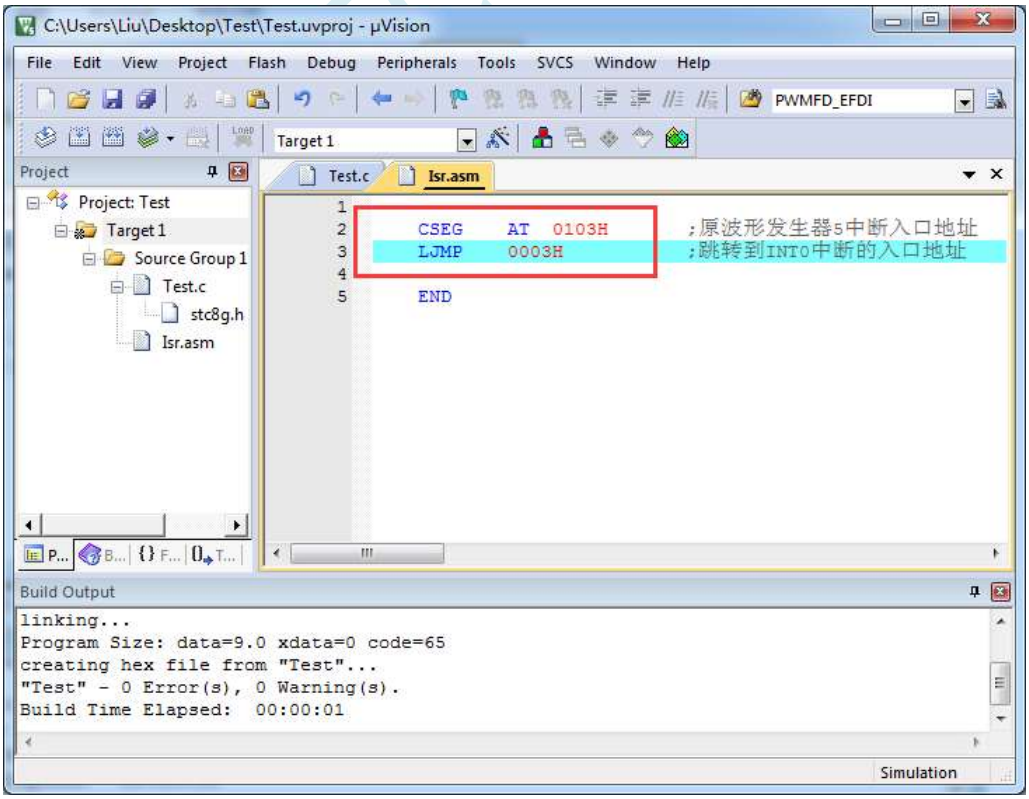


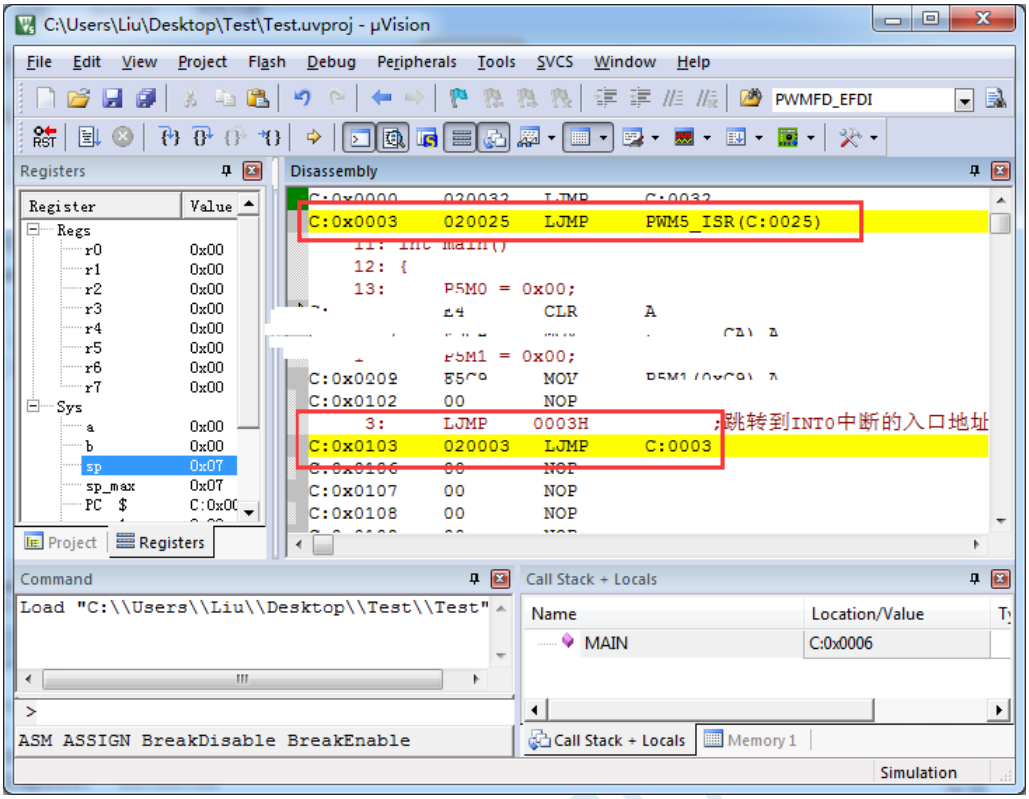
中断服务程序执行完成后，再通过 RETI 指令返回。整个中断响应过程只是多执行了一条 LJMP 语句而已。

方法 2：与方法 1 类似，借用用户程序中未使用的 0~31 的中断号

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





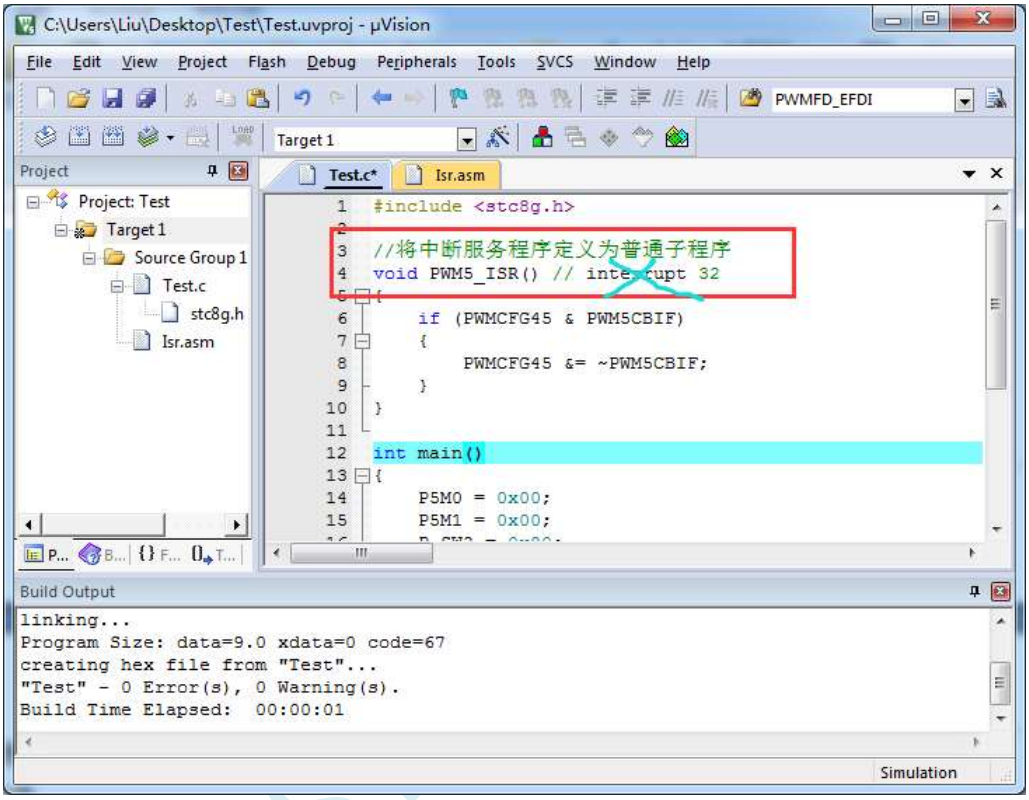


执行效果与方法 1 相同，此方法适用于需要重映射多个中断号大于 31 的情况。

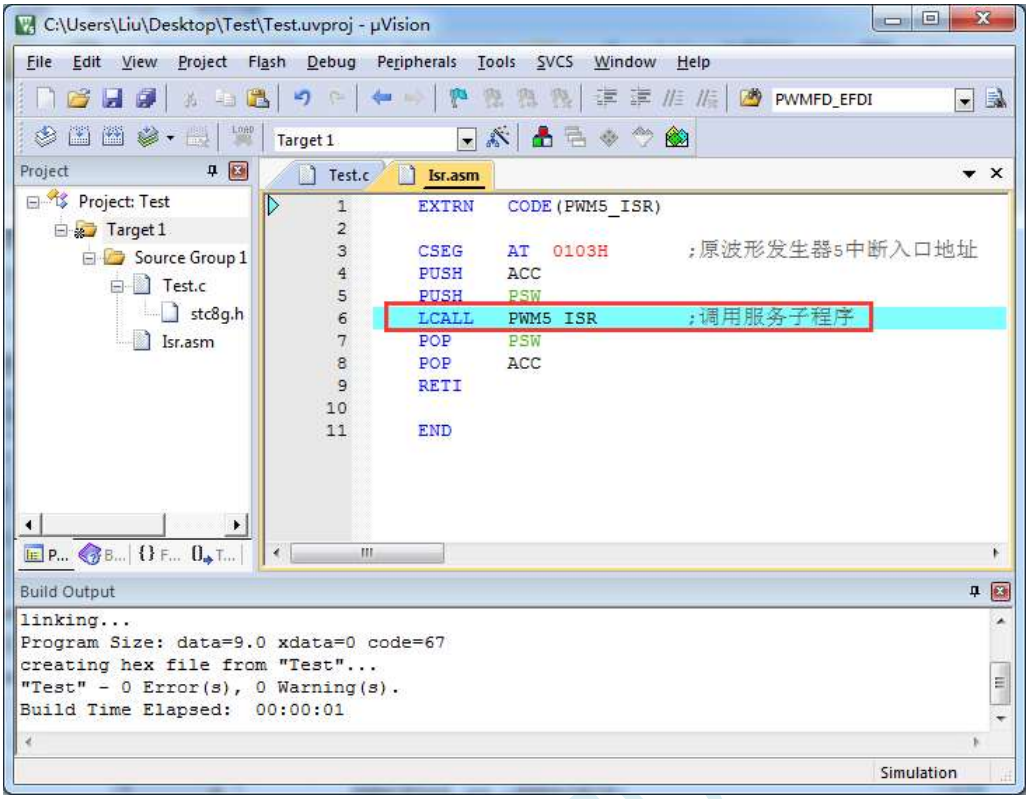
方法 3：将中断服务程序定义成子程序，然后在汇编代码中的中断入口地址中使用 LCALL 指令执行服务程序

操作步骤如下：

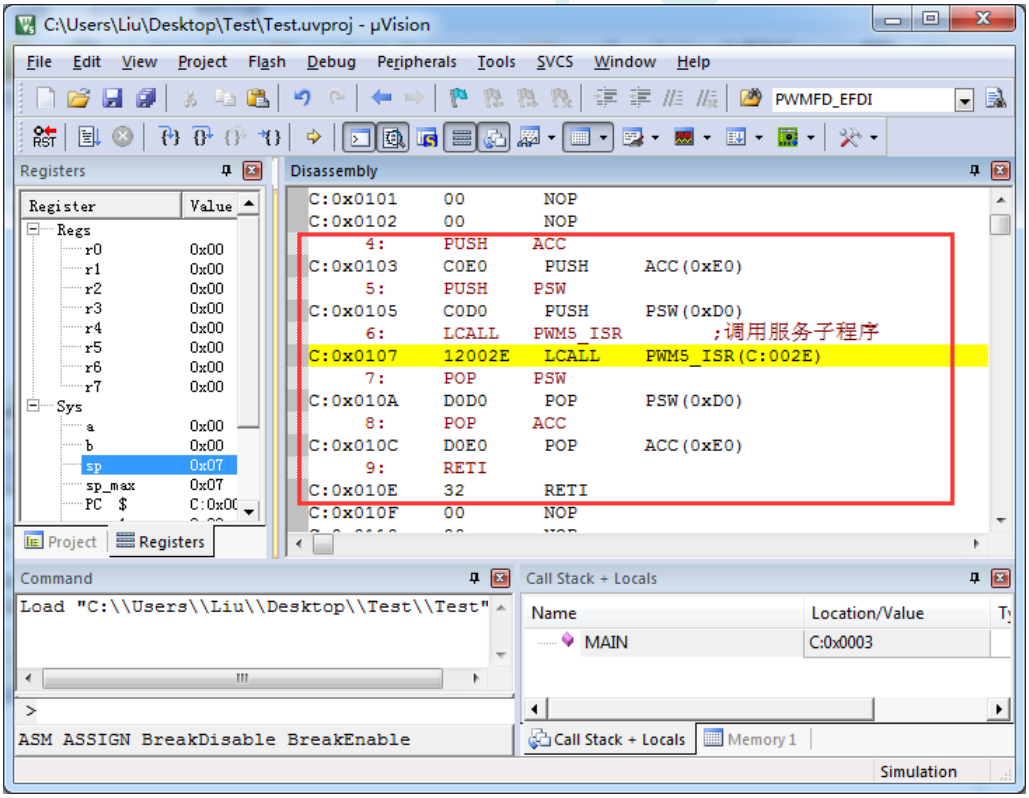
1、首先将中断服务程序去掉“interrupt”属性，定义成普通子程序



2、然后在汇编文件的 0103H 地址输入如下图所示的代码



3、编译通过后，即可发现在 0103H 地址的地方即为中断服务程序



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

附录P 电气特性

P.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	℃	
工作温度	-40	+85	℃	若工作温度高于 85℃（如 125℃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或晶振。另外温度高时频率跑不快，如果必须使用内部 IRC 时钟，建议使用 24M 以下的工作频率；如果系统必须运行在较高频率，则请使用外部高可靠有源时钟。 若工作温度为-55℃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	

STC MCU

P.2 直流特性（3.3V）

（VSS=0V，VDD=3.3V，测试温度=25℃）

标号	参数	范围				测试环境
		最小值	典型值	最大值	单位	
I_{PD}	掉电模式电流	-	0.4	-	uA	
I_{WKT}	掉电唤醒定时器	-	1.5	-	uA	
I_{LVD}	低压检测模块功耗	-	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	最快速度		10		MHz	关闭所有模拟和数字滤波

器	模拟滤波时间		0.1		us	
	数字滤波时间		0		系统时钟	LCDTY＝0
			n+2			LCDTY=n (n=1~63)
I_{PD2}	使能比较器时掉电模式功耗	-	400	-	uA	
I_{PD3}	使能 LVD 时掉电模式功耗	-	470	-	uA	

STC MCU

P.3 直流特性（5.0V）

（VSS=0V，VDD=5.0V，测试温度=25℃）

标号	参数	范围				测试环境
		最小值	典型值	最大值	单位	
I_{PD}	掉电模式电流	-	0.6	-	uA	
I_{WKT}	掉电唤醒定时器	-	4.4	-	uA	
I_{LVD}	低压检测模块功耗	-	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	最快速度		10		MHz	关闭所有模拟和数字滤波

器	模拟滤波时间		0.1		us	
	数字滤波时间		0		系统时钟	LCDTY＝0
			n+2			LCDTY=n (n=1~63)
I_{PD2}	使能比较器时掉电模式功耗	-	460	-	uA	
I_{PD3}	使能 LVD 时掉电模式功耗	-	520	-	uA	

P.4 内部 IRC 温漂特性（参考温度 25℃）

温度	范围		
	最小值	典型值	最大值
-40℃～85℃		-1.38%～+1.42%	
-20℃～65℃		-0.88%～+1.05%	

P.5 低压复位门槛电压（测试温度 25℃）

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）	

STCMCU

附录Q 应用注意事项

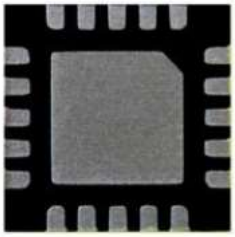
Q.1 关于 STC12H 系列 IO 口的注意事项

1. STC12H 系列芯片的 IO 口，除了 ISP 下载口 P3.0 和 P3.1 外，其余的 IO 口上电后的初始模式均为高阻输入模式，用户无法直接输出电平，所以用户在程序初始化的地方必须要使用 PxM0 和 PxM1 两个寄存器初始化相应的 IO 模式，才能正常使用。
2. STC12H 系列芯片所有的 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. STC12H 系列芯片不会自动为特殊 IO 设置 IO 口模式，如 ADC 口、串口、I2C 口以及 SPI 口，必须用户自行将相应的口设置为合适的模式
4. 若使能 P5.4 管脚为复位脚，则复位电平为低电平
5. 特别注意：由于 STC12H 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。特别是对于 28 管脚和 20 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。
6. **STC12H 系列 A 版本芯片的 I/O 口中断经测试发现有问题，请暂时不要使用**

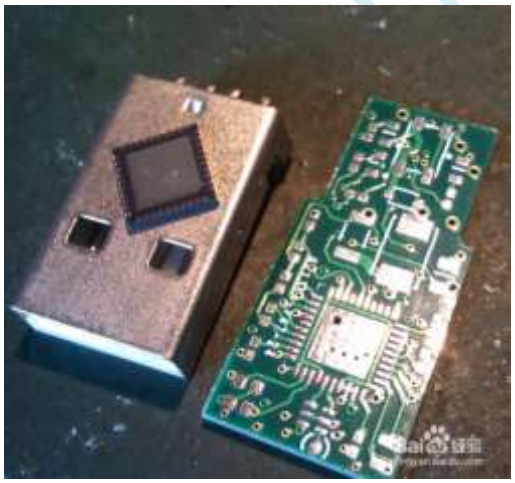
STCMCU

附录R　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。

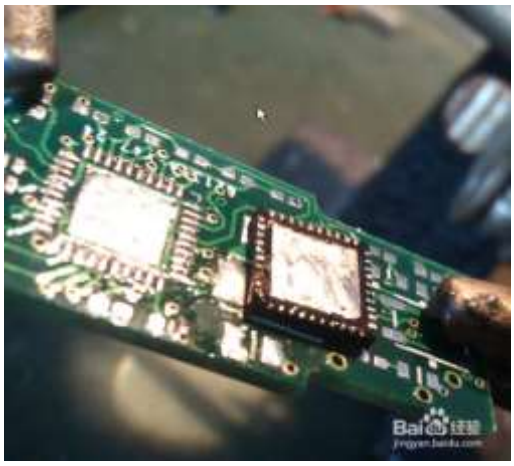


1、　首先需要准备如下工具：电烙铁、热风枪、镊子、固定架等工具
2、　需要焊接的 PCB 板和芯片如下图：



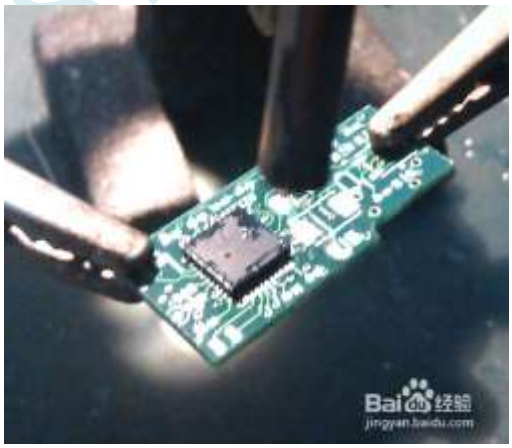
3、　先给板上芯片的焊盘上锡：

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。



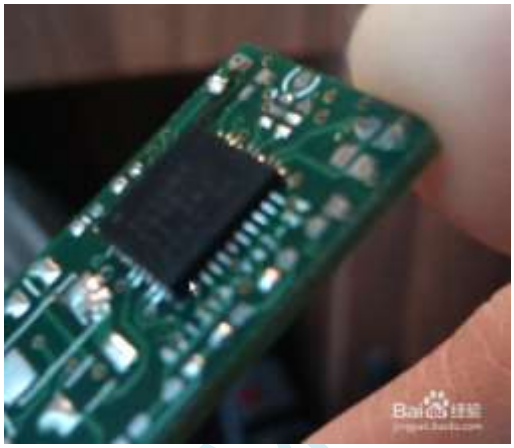
5、 调整热风枪温度，实际出风大概在 240 度左右，因为风枪质量不一样，根据实际情况调节。

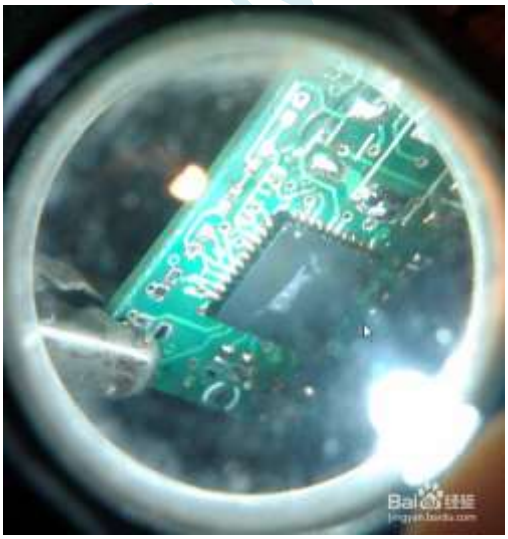
6、 把芯片放到焊盘上，一定要放正，然后用热风枪对着它吹，速度要均匀，直到锡溶化，一般 20 秒内。



7、 用烙铁给芯片侧引脚上锡

8、 焊接完成后的效果

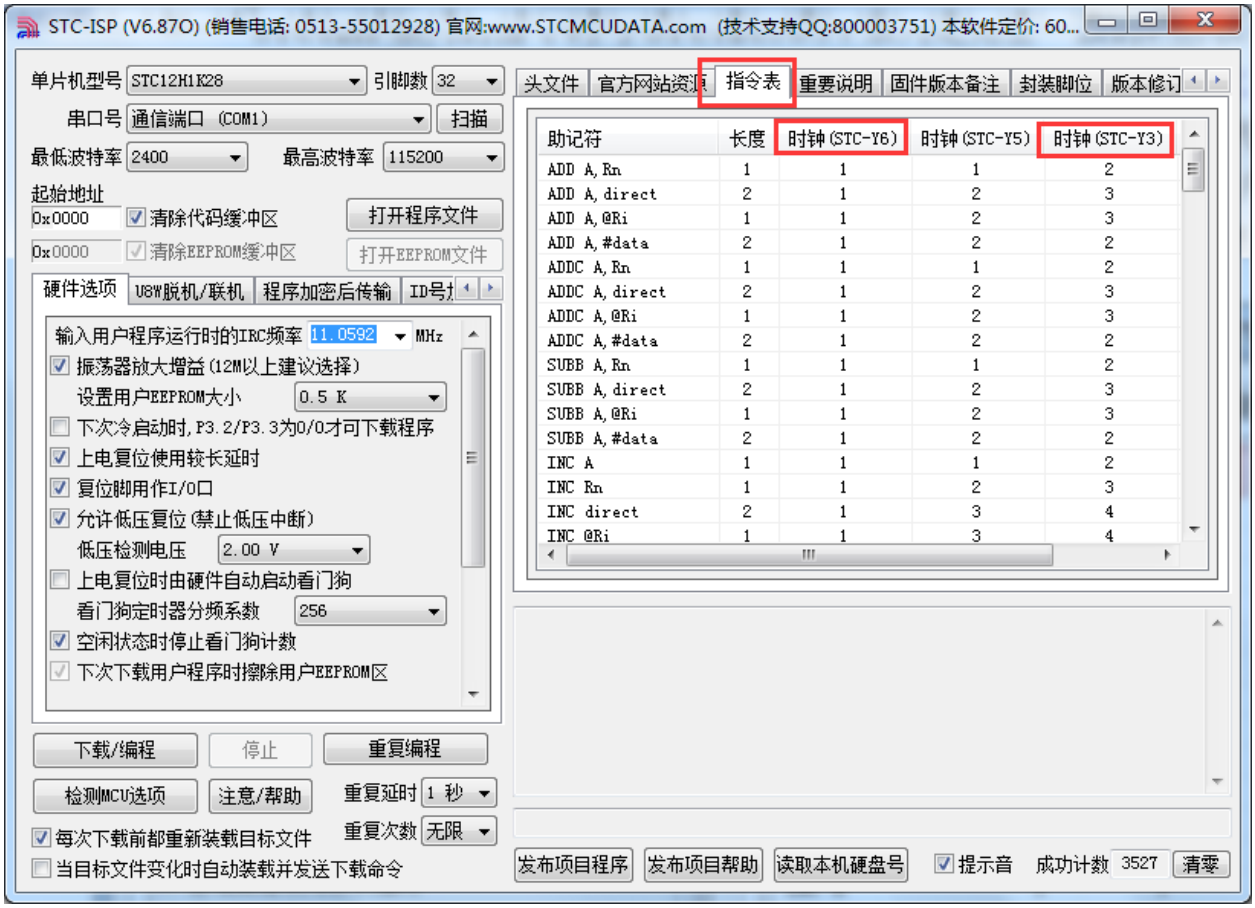




附录S　STC12H 系列单片机取代 STC12C56/54 系列的注意事项

■ 单片机指令

STC12H 系列的指令码与 STC12C56/54 系列是完全一致的，所以 STC12C56/54 系列的代码移植到 STC12H 上，运行依然正确，但 STC12H 系列的指令速度比 STC12C56/54 系列要快，STC12C56/54 系列的指令系统属于 STC-Y3 系列指令，而 STC12H 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：



■ 工作电压

STC12H 系列为宽压芯片，工作电压范围为 1.9V～5.5V，STC12C56/54 系列分为 3V 系列和 5V 系列，3V 芯片的工作电压范围为 2.4V～3.6V，5V 芯片的工作电压范围为 3.5V～5.5V。

■ 低压检测

STC12H 系列内部低压检测为 4 级可选，STC12C56/54 系列位两级可选。

■ 系统时钟源

STC12H 系列系统时钟可为内部高速高精度 IRC、外部晶振和内部 32K 之一，为软件动态配置，STC12C56/54 系列只能 ISP 下载时硬件选择内部 IRC 或者外部晶振。

■ 内部扩展 RAM

STC12H 系列内部扩展 RAM 为 1K 字节，STC12C56/54 系列内部扩展 RAM 为 512 字节

■ I/O 口

STC12H 系列单片机上电后，I/O 的模式与 STC12C56/54 系列不一样。STC12C56/54 系列单 STC12H 系列单片机上电后，I/O 的模式与 STC12C56/54 系列不一样。STC12C56/54 系列单片机所有 I/O 口上电后都是 8051 的准双向口模式，而 STC12H 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 STC12C56/54 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。STC12H 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 STC12H 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 STC12H 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

STC12H 系列 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），此时复位脚为低电平时，单片机处于复位状态，高电平时单片机解除复位状态。对于 STC12C56/54 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。

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

STC12H 系列的 EEPROM 相关特殊功能寄存器与 STC12C56/54 不兼容。另外擦除和编程的等待时间设置也不一样。STC12C56/54 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，STC12H 系列新增了一个寄存器 IAP_TPS（SFR 地址：0F5H），专用于设置 EEPROM 擦除和编程的等待时间，且用户不需要去计算，只需要根据当前 CPU 的工作频率，直接填入 IAP_TPS 即可，硬件会自动计算等待时间。（比如：当前 CPU 的工作频率为 24MHz，则只需要向 IAP_TPS 填入 24 即可）

■ 定时器

STC12H 系列有 5 个 16 位定时器，STC12C56/54 只有两个定时器。

STC12H 系列的定时器 0 和定时器 1 的模式 0 为 16 位自动重装载模式，而 STC12C56/54 系列的定时器 0 和定时器 1 的模式 0 为 13 位不自动重装载模式。

STC12H 系列的定时器 0 的模式 3 为不可屏蔽中断的 16 位自动重装载模式，而 STC12C56/54 系列的定时器 0 的模式 3 为双 8 位定时器模式。

■ 串口

STC12H 系列有两个串口，STC12C56/54 只有 1 个。STC12H 系列的所有串口最高波特率均可达到系统主频/4，STC12C56/54 最快只能系统主频/16。

■ ADC

STC12H 系列和 STC12C56/54 系列的 ADC_CONTR、ADC_RES、ADC_RESL3 个寄存器地址不兼容。STC12H 系列另外新增加了两个寄存器：ADCCFG 和 ADCTIM。

STC12C56/54 系列开始 ADC 转换位 ADC_START 位于寄存器 ADC_CONTR 的 BIT3，而 STC12H 系列的位于 ADC_CONTR 的 BIT6

STC12C56/54 系列 ADC 转换完成标志位 ADC_FLAG 位于寄存器 ADC_CONTR 的 BIT4，而 STC12H 系列的位于 ADC_CONTR 的 BIT5

STC12C56/54 系列 ADC 速度控制为 ADC_SPEED 位于寄存器 ADC_CONTR 的 BIT6-BIT5，而 STC12H 系列的位于 ADCCFG 的 BIT3-BIT0

STC12C56/54 系列 ADC 转换结果的对齐控制位 ADRJ 位于寄存器 CLK_DIV 的 BIT5，而 STC12H 系列的对齐控制位 RESFMT 位于 ADCCFG 的 BIT5

STC12H 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

■ PCA/PWM/CCP

STC12H 系列的 PCA/PWM/CCP 的第 0~2 组模块的特殊功能寄存器地址与 STC12C56/54 是兼容的，但第 3 组不兼容，STC12C56/54 的第 3 组寄存器定义在 SFR 区，而 STC12H 的第 3 组寄存器定义在 XFR 区。

STC12H 系列的的 PWM 模式可输出 6 位/7 位/8 位/10 位 PWM，STC12C56/54 系列只能固定输出 8 位。

■ SPI

STC12H 系列的 SPI 相关特殊功能寄存器地址与 STC12C56/54 不兼容。

STC12H 系列的 SPI 速度控制分别为系统时钟/4、/8、/16、/32，STC12C56/54 系列的 SPI 速度控制分别为系统时钟/4、/16、/64、/128

■ 看门狗

STC12H 系列的看门狗相关特殊功能寄存器地址与 STC12C56/54 不兼容。

STCMCU

附录T 更新记录

- **2023/1/10**

  1. 增加论坛链接
  2. 更新参考线路图说明

- **2022/12/23**

  1. 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

- **2022/9/21**

  1. 修正比较器章节的错别字
  2. 更新官方网址
  3. 更新选型价格表
  4. 增加 STC-ISP 高级应用章节
  5. 增加关闭驱动程序强制数字签名说明章节

- **2022/3/9**

  1. 更正文档中的笔误
  2. 更新特殊功能寄存器列表（STC12H 系列只有 P1IE 和 P2IE）
  3. 更新有关 I/O 口中断的应用注意事项

- **2021/12/17**

  1. 修正定时器 2/3/4 的定时计算公式

- **2021/10/29**

  1. 更正文档中的错别字

STCMCU

2. 增加 SOP16 的管脚图

- **2021/8/26**

1. 修正 ADC 章节范例程序中的注释错误

- **2021/6/26**

1. 修正晶振管脚端口名称错误

- **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

- **2021/2/26**

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

- **2021/2/4**

1. 更正 CLKDIV 寄存器的复位初始值
2. 增加特殊功能寄存器初始值说明
3. 在管脚图下增加应用参考线路图

- **2020/11/25**

1. 更正部分范例程序中的错误
2. 更新样品价格表
3. 更正文档中的描述错误

- **2020/10/16**

1. 更新 STC12H 系列单片机的参考价格
2. 增加外部晶振电路的负载电容说明

- **2020/9/23**

1. 创建 STC12H 系列单片机技术参考手册文档

STC MCU

产 品 授 权 书

致：江苏国芯科技有限公司

STC12H 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 STC12H 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。

深圳国芯人工智能有限公司